上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

学士学位论文

THESIS OF BACHELOR

Project Title Vector Information Pick up from the Tire-2D Section Image

Name1 Lina Zhang ID1 5143709208 Major ECE

Name2 Jiaqing Ni ID2 5143709230 Major ECE

Name3 Rishang Xu ID3 5143709097 Major ECE

Name4 Yiting Luo ID4 5143709181 Major ECE

Name5 Yucheng Shi ID5 5143709089 Major ECE

Supervisor Yong Long

School UM-SJTU Joint Institute

Semester 2017-2018-Summer

上海交通大学
毕业设计（论文）学术诚信声明

本人郑重声明：所呈交的毕业设计（论文），是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

作者签名：张立娜 倪佳清 史玉成
徐日上 骆仪婷

日期：2018 年 8 月 6 日

上海交通大学
毕业设计（论文）版权使用授权书

本毕业设计（论文）作者同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本毕业设计（论文）的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本毕业设计（论文）。

保密☑，在10年解密后适用本授权书。

本论文属于

不保密□。

（请在以上方框内打“√”）

作者签名：张立娜 倪佳清 史玉成 徐日上 骆仪婷　　指导教师签名：

日期：2018 年 8 月 6 日　　日期：2018 年 8 月 6 日

Vector Information Pick up from the Tire-2D Section Image

VE450 Major Design Experience

Summer 2018

Group 10

Sponsor:

GITI Company, Rui Deng (deng.rui1@giti.com)

Section Instructor:

UM-SJTU Joint Institute, Yong Long (yong.long@sjtu.edu.cn)

UM-SJTU Joint Institute, Yunlong Guo(yunlong.guo@sjtu.edu.cn)

Group Members:

Lina Zhang (linazh@umich.edu)

Jiaqing Ni (jiaqni@umich.edu)

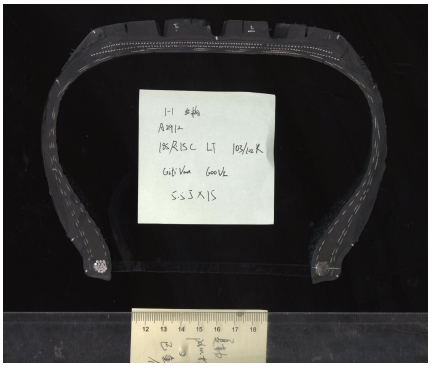
Rishang Xu (xurishang@sjtu.edu.cn)

Yiting Luo (luoluo@umich.edu)

Yucheng Shi (yuchengs@umich.edu)

Date:

08/06/2018



Executive Summary

The need of our sponsor, Giti Company, is to measure the dimensions of tire's section profile automatically and precisely. As for now, the dimensions of the tire slice can only be measured manually, which is of low efficiency. The motivation of our project is to use software methods to extract the vector information of the tire and to measure the dimensions automatically and precisely.

The input sample image is shown below in Figure 1. It is a scan image of a real tire's longitudinal cutted slice. The specifications of the project is that the software we developed could transform several geometrical elements, including the inner and outer contour, the key points and the belt plies information in the scan image of a tire's 2D section plane into vector data. Refer to the sample (vector) output in Figure 1 for the position of the elements to be detected. Each element should be denoted using independent, smooth vector curves. The vector output should exclude the influences of burr, shadow and depth of field. The tolerance deviation between any point on the detected contour and the corresponding point on the actual contour is within 0.5 mm. And the vector curve representing belt ply should pass through the center of each dots.

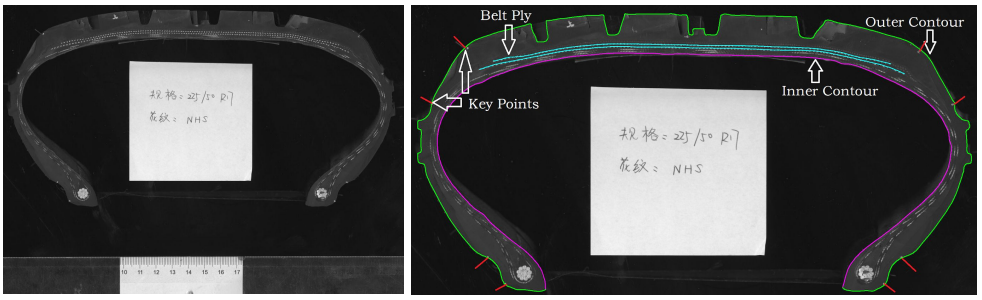


Figure 1. The input image and corresponding sample vector output

The concept we chose is to use image processing to extract contours and belt plies, then use the contour information to detect key points. We use this concept since it's stable and fast comparing to other generated designs. The final design is similar to our generates concept and the outcome meets our expectations.

The operation platform we are using is Python3. The libraries we use are open sourced so the cost of reproducing the project according to our method is zero. The test results indicate that our code could generate ideal outputs unless extreme situations (misty figures and large highlight area on the background) occur.

To sum up, the performance of our project is stable and the outcome meets the requirements of our sponsors. We found a balance between the accuracy and the runtime and the total processing time is within 60 seconds per input image. Some details like the connection of beltply dots could be improved if we have more time.

Table of Contents

PART I Introduction

A. Background

Tires are used for supporting the weight of vehicles, providing traction on the surface that vehicles travel over and reducing the amplitude of vibration of vehicles' components. The tires for the automobile are inflated structures so the structure of the radial cross section is one of the most significant parts in the design of tires. The dimensions of tires' radial cross sections have a significant impact on tires' performances, which include the magnitude of traction, the amount of generated heat and wearprofness.

Due to the significance of tire, the analysis of tires' radial cross sections is an important part in analyzing the quality of products for tire companies, since the structure of the cross section determines the performances of the tire. For Giti company, the measurements are based on conventional measuring tools for now. First of all, positions of geometrical elements like inner contour, outer contour, and key points are marked out manually using white ink. Like Figure 2 shows, the key points are marked out using short white lines and the inner contour is marked using a continuous white curve. Then line tapes, vernier calipers, and straightedges are used manually on the material object to gain desired dimensions with the help of these white marks.

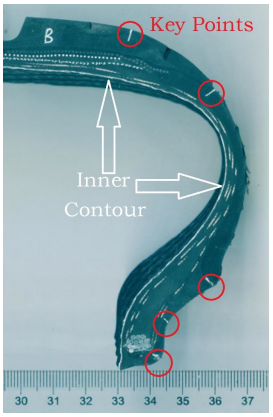


Figure 2. The manually made white marks on the section profile

However, the problem for GITI's existing measuring procedure is that it's scarcely

possible to be unified and normalized. The error could be caused by the change of the width of the white mark, the tilt of the ruler, and observational error, which goes against the quality control. What is more, the existing method is of low efficiency, manually measuring is time consuming, which may take 10 to 15 minutes every single section profile. To recognize the inner and outer contours needs about 10 minutes and 3 minutes for belt ply recognition. To find all of the key points, it needs about 1 minute (see footnote for more information).[1] A method which could enable users measuring the dimensions of tires' radial cross section accurately and efficiently is needed in this field.

Since the existing method provided by Giti is of low efficiency, we turn to look at other benchmarks. For most of the patents which are used for measuring the dimensions of tires, they require extra hardware components. Sensor systems and cameras are used in these patents, which causes extra expenses. To decrease the cost, Giti looked into other possible solutions. The innovation idea provided by Giti company is to use the technique of image processing to extract tire's geometrical information.

B. Expected Outcome

The geometrical information includes inner contour, outer contour, belt ply and key points (see figure below). All the information can be used for measuring the dimensions of the tire, so these elements should be detected precisely in order to ensure the accuracy of measurement. After we extract all the information mentioned above, we are supposed to vectorize it so the final output should be a vectorized image whose coordinates represent real distances. Finally, the vector image could be used for measuring distances in AutoCAD directly by moving the cursor.

[1] The sentence responds to Dr. Qianli Chen's comment in Design Review 3. She asked what is Giti's previous solution to the problem and how much time it would take.

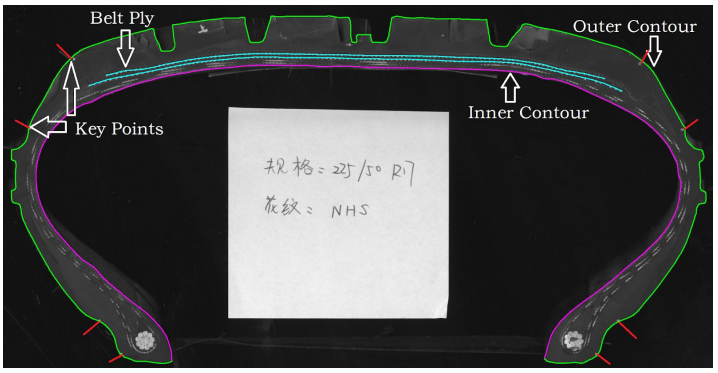


Figure 3. The sample output for the project

Due to the problems of the existing methods and the requirements of accurate measurement (error within 1 mm), the requirements for this project includes detecting the elements precisely, high automation extent and short runtime to get vector image.

Thus, using image processing to extract the geometrical information of the tire is innovative. The feature of using image processing is that it can overcome shortcomings such as disunity, low efficiency, and high expenses.

C. Competitive and Related Products

There are prior arts of similar usage which are used in this field, however, the approaches are quite different from ours'. One of the prior arts is also used for measuring the section profile shape of a tire, but it uses a sensor including a slit-ray generator, a camera, a calibration block, a robot hand and a robot driver in order to control the measuring position and measure the apparatus [1]. The purpose is similar but this approach requires more hardware components and it could not detect belt ply's information. Though robot arms are used, the whole procedure is not automatic, a driver should monitor the position of the robot hand. The measurement is rather accurate since images from different positions are taken, but it is of low efficiency since gaining the images takes a lot of time. A flowchart of this prior art is shown below.

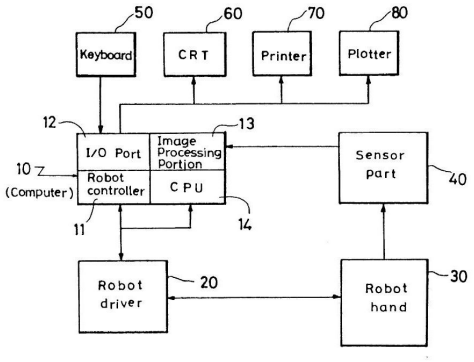


Figure 4. Patent for measuring tire's section profile shape [1]

Another possible solution to detect the information of belt ply is to use a sensor system which can ascertain the disposition and condition of ply wires in a tire [2]. A diagram of this patent is shown below. This requires additional components and therefore add costs to the company. The sensors are clung to the tire by clamps so the whole structure is not stable enough. There are many other patents (like [3]) which focus on monitoring the manufacturing procedure and feedback the inner structure of the tire. It is useful and the parameters gained in this procedure could be used as references. But the methods couldn't be used on the finished product since the finished tire is already a whole part and it would be impossible to disassemble the belt ply and the other structures. Measuring the dimensions of a finished product is still necessary because assembled tire products may deviate from the design even though the manufacturing procedures are under monitoring.

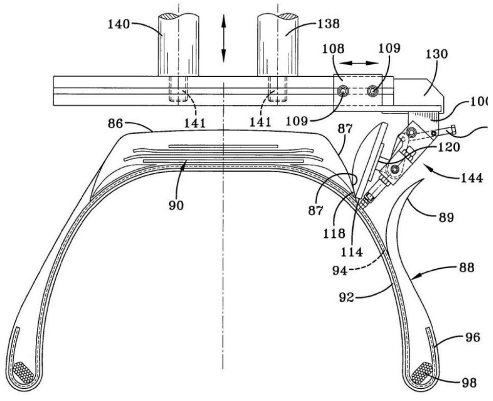


Figure 5. Patent for detecting the information of belt ply [2]

Therefore, solving the auto-measuring problem by using image processing technique is innovative in this industry and it has advantages. We don't require additional hardware components and additional labors to gain the apparatus data, which is of high efficiency. The approach for our project is image processing, which is widely used but the object we are dealing with is unique.

The input for our project is the scanned image of a real tire's longitudinal cut slice. The expected outcome is the extracted vector information. The extracted information includes the inner contour, the outer contour, the key points and the belt ply information. These pieces of information should be separated lines in order to be used in auto-measuring. To prevent the influences of other elements, the vector information we detected should exclude the influences of burr, shadow, and depth of field. The size of the vector information should be the same as the original CAD design of the tire.

PART II Customer Requirements & Engineering Specifications

A. Customer Requirements

The requirements for this project include

1. Detect the inner contour and the outer contour of the tires' radial cross section
2. Detect the belt ply
3. Detect the key points
4. Appropriate runtime
5. Convert the detected information into vectors
6. Achieve curve fitting
7. High automatic level
8. Low expense

The quantified engineering specifications include:

1. For the detection of contour, the influence of shadow, burr, and depth of field should be excluded. The schematic diagram of these interference factors is shown below. According to the sponsor, the deviation of the detected contour and the real contour should be within 0.5 mm since the measured dimensions' error is supposed to be less than 1 mm.

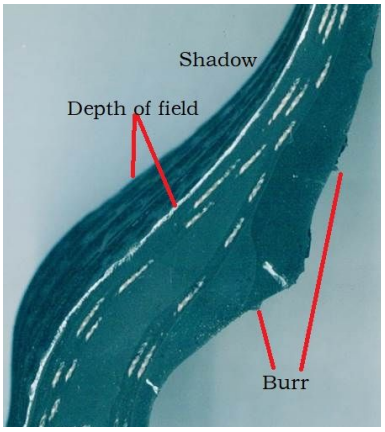


Figure 6. Schematic diagram of interference factors

We could compare the contour we detected with the ideal contour provided by our sponsor. Let $C = \{p_1, p_2, ..., p_n\}$ be our detected contour, where $p_i = (x_i, y_i)$ are points on the contour. Let the ideal contour be expressed as $p = (x, y)$, then the error between the detected contour and the real contour could be written as:

$$\min_{1 \le i \le n} L_2(p, p_i) = \min_{1 \le i \le n} \sqrt{(x - x_i)^2 + (y - y_i)^2}$$

2. For the detection of belt ply, the vector information should pass through the centers of the belt ply dots for precision, like Figure 7 shows. Since the centers of the dots are more representative of the position of belt ply. If branching exists (like the first two rows in Figure 7), the branches should be connected separately instead of connecting the dots all together.

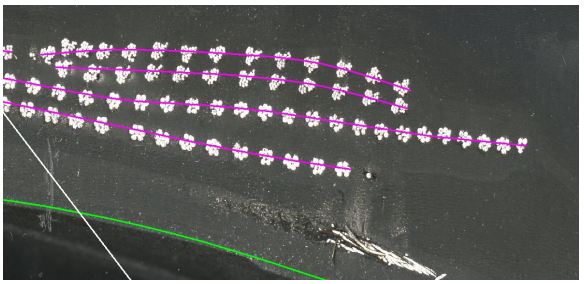


Figure 7. The connection of belt ply dots

The number of passed dots should be no less than 95% of the true points. The definition of pass-through means the given point $p_i = (x_i, y_i)$ should be in the range of the belt ply shown on the image.

3. The key points should be detected and marked precisely. All of the key points should be detected and the deviation of the detected key point and the real key point should be within 1 mm since the measured dimensions' error is supposed to be less than 1 mm and key points are used for finding out the measuring position.

4. The runtime is not in high demand. According to our sponsor, run time for each input image (including manually operation time) within 60 seconds is tolerable. Since we have manually operation time (around 15 seconds), the program's running time would be around 45 seconds.

5. The conversion should be of the same size as the original CAD design.

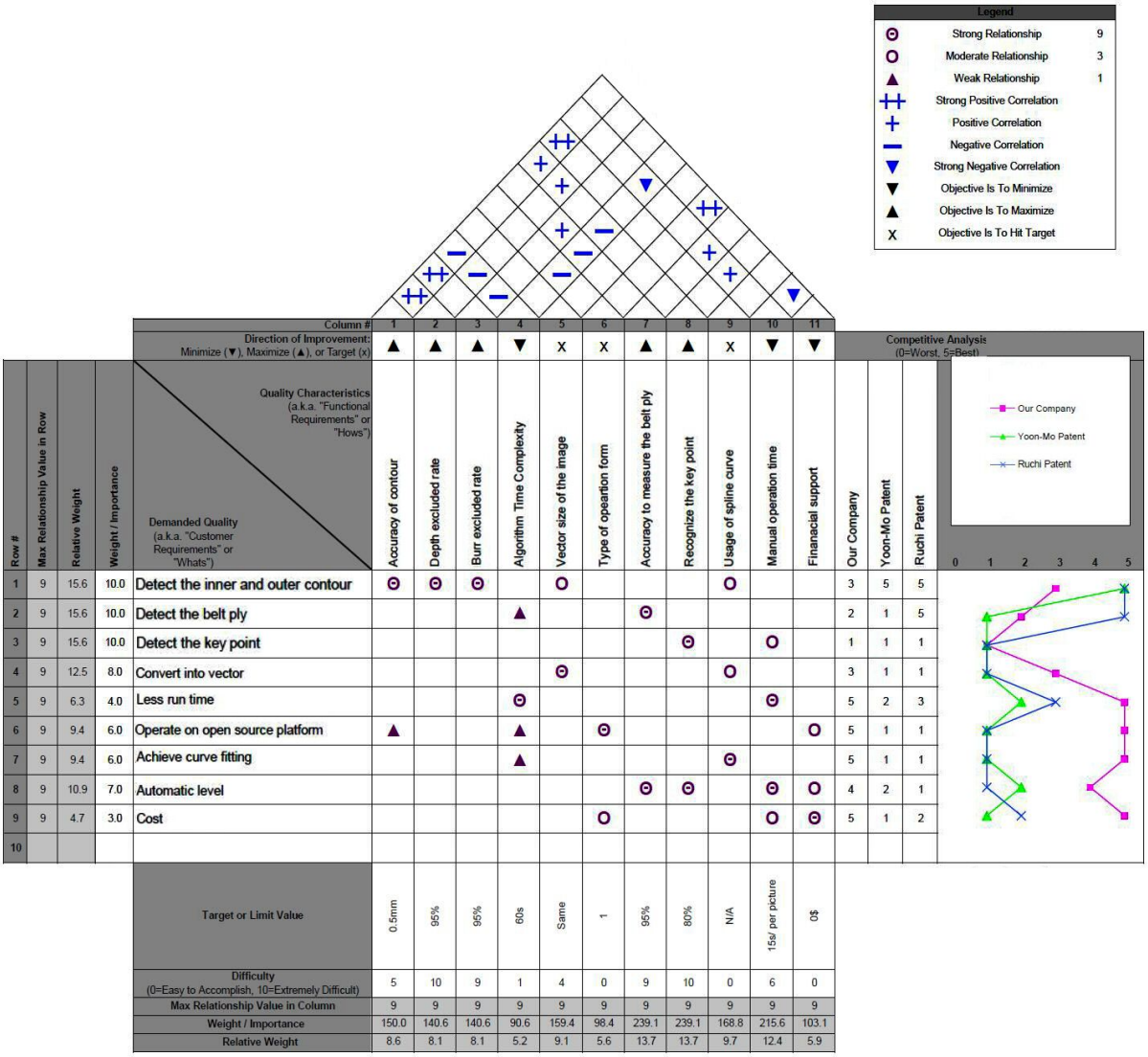
6. The platform we are using is python, since it's open source and free.

7. As for the curve fitting, we would use spline curves to connect the dots we get in image processing.

8. The automatic level should be high since the start point of the project is to reduce the manual operation. For each image input, the manual operation time should be less than 15 seconds per image.

9. The expense for our project should be moderate. Since it's a software project, the cost would be near zero.

B. QFD Chart



Legend		
⊙	Strong Relationship	9
O	Moderate Relationship	3
▲	Weak Relationship	1
++	Strong Positive Correlation	
+	Positive Correlation	
−	Negative Correlation	
▼	Strong Negative Correlation	
▼	Objective Is To Minimize	
▲	Objective Is To Maximize	
X	Objective Is To Hit Target	

Row #	Max Relationship Value in Row	Relative Weight	Weight / Importance	Demanded Quality (a.k.a. "Customer Requirements" or "Whats") \ Quality Characteristics (a.k.a. "Functional Requirements" or "Hows")	1 Accuracy of contour	2 Depth excluded rate	3 Burr excluded rate	4 Algorithm Time Complexity	5 Vector size of the image	6 Type of opeartion form	7 Accuracy to measure the belt ply	8 Recognize the key point	9 Usage of spline curve	10 Manual operation time	11 Finanacial support	Our Company	Yoon-Mo Patent	Ruchi Patent
				Direction of Improvement: Minimize (▼), Maximize (▲), or Target (x)	▲	▲	▲	▼	x	x	▲	▲	x	▼	▼	Competitive Analysis (0=Worst, 5=Best)		
1	9	15.6	10.0	Detect the inner and outer contour	⊙	⊙	⊙		O				O			3	5	5
2	9	15.6	10.0	Detect the belt ply				▲			⊙					2	1	5
3	9	15.6	10.0	Detect the key point								⊙		O		1	1	1
4	9	12.5	8.0	Convert into vector					⊙				O			3	1	1
5	9	6.3	4.0	Less run time				⊙						⊙		5	2	3
6	9	9.4	6.0	Operate on open source platform	▲			▲		⊙					O	5	1	1
7	9	9.4	6.0	Achieve curve fitting				▲					⊙			5	1	1
8	9	10.9	7.0	Automatic level							⊙	⊙		⊙	O	4	2	1
9	9	4.7	3.0	Cost						O				O	⊙	5	1	2
10																		
				Target or Limit Value	0.5mm	95%	95%	60s	Same	1	95%	80%	N/A	15s/ per picture	0$			
				Difficulty (0=Easy to Accomplish, 10=Extremely Difficult)	5	10	9	1	4	0	9	10	0	6	0			
				Max Relationship Value in Column	9	9	9	9	9	9	9	9	9	9	9			
				Weight / Importance	150.0	140.6	140.6	90.6	159.4	98.4	239.1	239.1	168.8	215.6	103.1			
				Relative Weight	8.6	8.1	8.1	5.2	9.1	5.6	13.7	13.7	9.7	12.4	5.9			

Chart 1. Project QFD chart

The QFD chart is shown above. It is developed by listing the detailed demands and comparing the significances of these requirements. We sorted the significance of these specific demands from high to low and listed them in the main body of the chart. The relevant benchmarks are also listed in the right portion of the chart in order to be compared with our solution.

According to the QFD chart, the demanded quality column lists all the required demands. Weight/Value part represents the significances of each demanded quality.

The target/limit value part listed out the quantified data of these demands. The two benchmarks have their drawbacks since neither of them could detect the contour, the belt ply and the key points at the same time. Meanwhile, both of the benchmarks need extra costs due to the accessional hardware components. The vectorization and the curve fitting targets couldn't be accomplished by these benchmarks either.

D. Summary of Engineering Requirements

Specific Requirements	Quantified Specifications
Contour detection	Deviation within 0.5 mm
Belt ply detection	Passed dots over 95%
Key points detection	Deviation within 1 mm
Total run time (include manual operation time)	Within 60s
Vector conversion	Same size as the original sample
Curve fitting	Spline curve
Automatic level	Manual operation time within 15s
Cost	0

PART III Concept Generation

According to the sponsor, to measure the dimensions of the tire, we need the position information of three elements, the contour, the belt ply and the key points. So the

whole project is naturally divided into three detection parts. The major problem is how to detect these elements, how to select the consequences of these parts, how to make use of the information we get from one detection part to make the final result more accurate, and how to combine the parts to get the output.

The generated concept includes:

1. Using convolutional neural network to extract tire's geometrical elements.
2. Separating these elements by setting a threshold value for grey values.
3. Using information of key points to help detect contour and belt ply.
4. Detecting turning points on the contour to help detect key points.
5. Using cameras to eliminate the influences of the depth of field.
6. Using the information of the contour to detect the key points.

At first, we think of using CNN (convolutional neural network) to extract the geometrical information. It is straightforward since we are provided with the sample input and the sample output. We thought we could use the input and output result to train the CNN until it could generate the reasonable result.

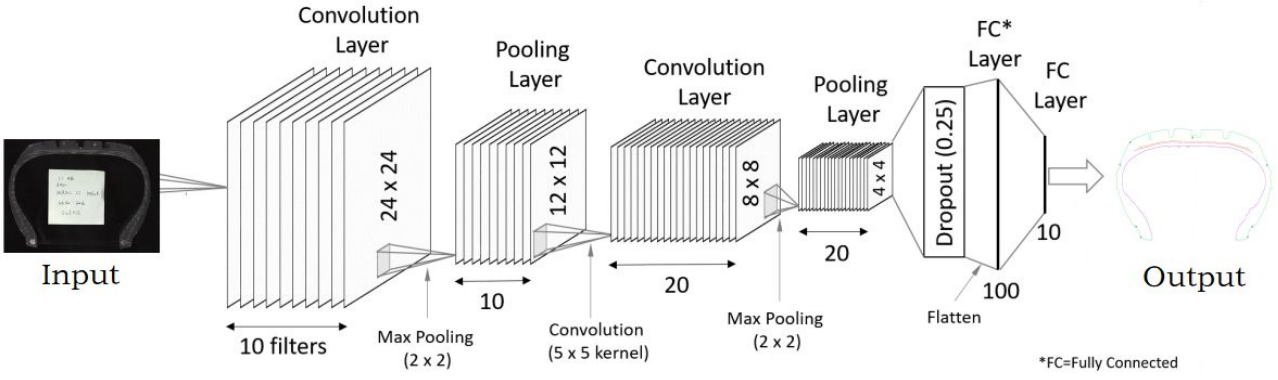


Figure 8. Schematic diagram of CNN

The second concept is due to our observation that the key points and the belt points are rather bright, the tire is grey and the background is dark. The method we think out is to set a threshold value to separate elements which are of different grey level.

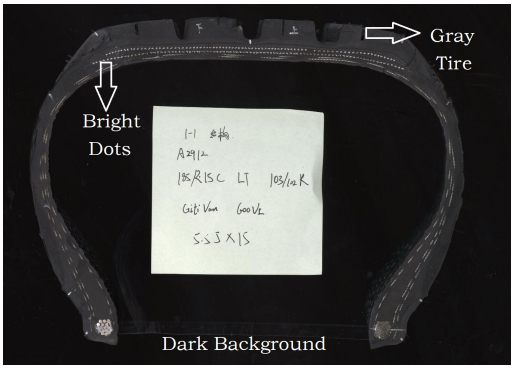


Figure 9. Elements of different grey levels

The next concept is to use information of key points to help detect the contour and locate the belt ply dots. That is because key points are always on the outer contour and we can use those dots to help with the contour detection. As for the belt ply, we can search within the region surrounded by key points, as the figure below shows.

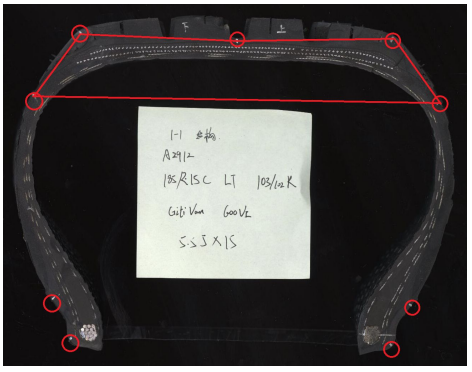


Figure 10. Schematic diagram of using key points in the detection

Another concept we generated is to detect the turning points on the contour in order to determine the general position of the other elements. Since we thought there are patterns for the location of the key points. If we get the values of the curvatures of the outer contour, we could get the positions of the key points according to the curvatures.

During our experiments, we found that the depth of field influences the detection of contour a lot, due to their similarity in pattern and color. If the problem of depth of field is solved, then the whole problem is a lot easier, so we tried other solutions which could eliminate the influences of the depth of field. If we take the photograph of a slice of the object from two different angles, the depth of field could be different on these two images due to the angles of the camera. Then we could process these two images together to get the contour information. The noise points could be eliminated too, which makes the detection of belt ply and key points easier (see appendix for more information).

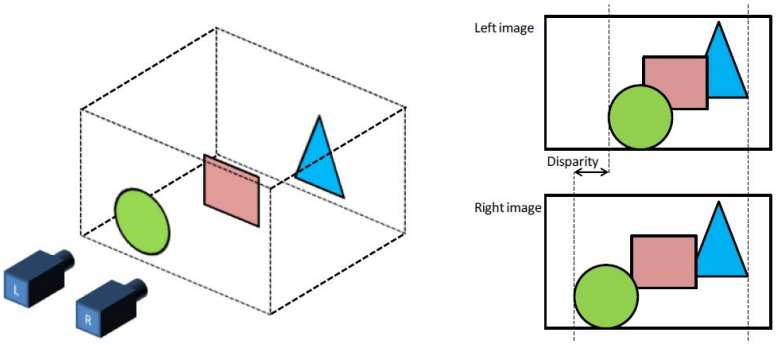


Figure 11. Concept diagram for eliminating depth of field using two cameras

Since the difference between the color of the tire and that of background always exist, we consider that we could use image processing technique similar to edge detection to get the contour, then we would use the information of the contour to detect the key points.

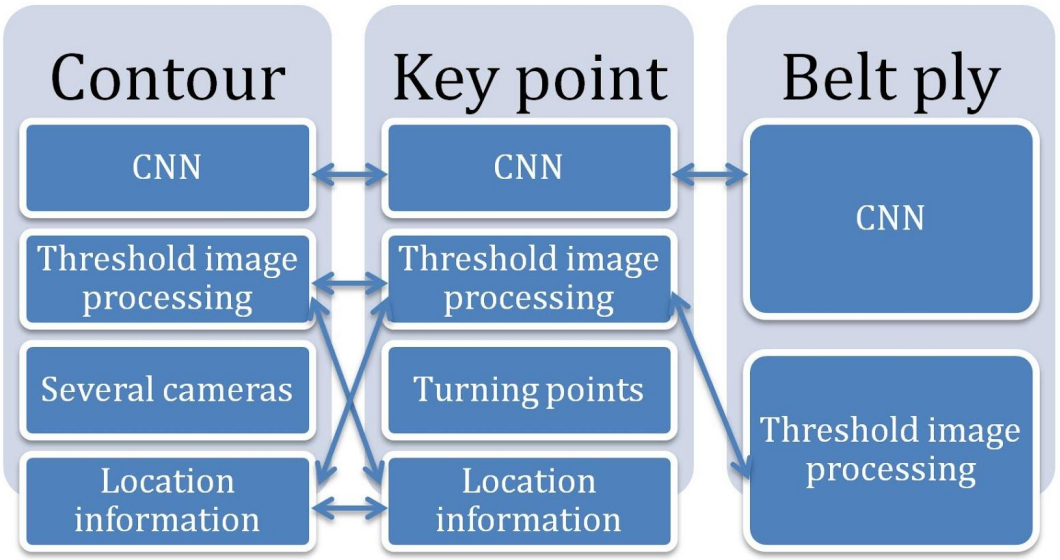


Figure 12. Morphological charts

The chart shows for the three sub functions (contour detection, key point detection , and belt ply detection), the concepts generated and the relations between them.

PART IV Concept Selection Process

Design Criterion	Weight Factor	Unit	CNN			Grey Value Detection			Key Points Detection First			Turning Points Detection			Contour Detection First		
			Value	Score	Rating	Value	Score	Rating	Value	Score	Rating	Value	Score	Rating	Value	Score	Rating
Accuracy of Contour	0.22	%	95	10	2.22	60	6	1.33	60	6	1.33	60	6	1.33	95	10	2.22
Accuracy of Belt Ply	0.22	%	95	10	2.22	50	5	1.11	20	2	0.22	30	3	0.67	80	8	1.77
Accuracy of Key Points	0.22	%	95	10	2.22	30	3	0.67	98	10	2.22	80	8	1.77	95	10	2.22
Time	0.11	S	10	2	0.22	1	10	1.11	3	8	0.88	5	7	0.77	5	7	0.77
Cost	0.07	¥	100	0	0	0	10	0.67	0	10	0.67	0	10	0.67	0	10	0.67
Automation Level	0.16	EXP	Fair	5	0.78	High	8	1.24	Good	7	1.09	Good	7	1.09	Good	7	1.09
Total					7.66			6.13			6.41			6.3			9.19

Chart 2. Scoring Matrix of Concept Generated

This is the scoring matrix of our concept. From the matrix we find the concept of "Contour detection first" has the largest rate. Therefore, we choose this way to solve the problem.

A. Use Convolutional Neural Network

At first, we think of using CNN (convolutional neural network) to extract the information we need, since we are provided with the sample input and the sample geometrical elements output we are supposed to get. We thought we could use the input and output result to train the CNN, if the training batch size is large enough, theoretically, we can get the output result at last. However, we found that the training set we could get is very limited. Giti could provide only tens of inputs instead of thousands of them. And as mentioned before, Giti company now get the desired output vector image by drawing along the contour, the belt ply and the key points manually in AutoCAD. It is time-consuming but we need a large training set to get a reasonable result. But using CNN means that the algorithm is always growing and the program is continuously improving.

Advantages: high precision, compatible with many situations, growing

Disadvantages: need large training set, slow

B. Use Grey Value Detection

Then, we turn to look at the different methods. Since we observed that the key points and the belt points are rather bright, the tire is grey and the background is dark, one method is to set a threshold value to separate elements which are of different grey level. However, there are many other bright elements on the picture, apart from the belt ply dots and the key points. What is more, for different input images, the grey value varies a lot, which is hard to analyze.

Advantages: simple, straightforward, short runtime

Disadvantages: hard to improve, low accuracy

C. Use Key Points Detection

The next concept is to use information of key points to help detect the contour and locate the belt ply dots. That is because key points are always on the outer contour with similar sizes and colors. We can detect the key points according to the patterns of them. After we get the positions of key points, we can use those dots to help with the contour detection. As for the belt ply, we can search within the region surrounded by key points.

Advantages: fast, straightforward

Disadvantages: unstable, the whole system can't work out if key point detection has errors

D. Use Turning Points Detection

Another concept we generated is to detect the turning points on the contour in order to determine the general position of the other elements. Since we thought there are patterns for the location of the key points. If we get the values of the curvatures of the outer contour, we could get the positions of the key points according to the curvatures.

Advantages: fast, straightforward

Disadvantages: low accuracy, unstable

E. Use Contour Detection

The feature of the three parts that we noticed is that the key points are located on the edge of the outer contour and the key points are in the region. On the other hand, we notice that there are noise points in the background part, some of them have similar sizes as the key points and the belt ply dots, which could influence these two detection parts. What is more, there are distinct color differences between the major part of the tire's cross section and the deep color background. Since the differences always exist, we consider that we could use image processing technique similar to

edge detection to get the contour, then we would use the information of the contour to detect the key points.

Advantages: fast, stable, high accuracy

Disadvantages: require high accuracy for contour detection

PART V Selected Concept Generation

A. Overview

Our final choice for the concept is detecting the edge at first, then use the contour information to detect the key points. The block diagram is shown below.

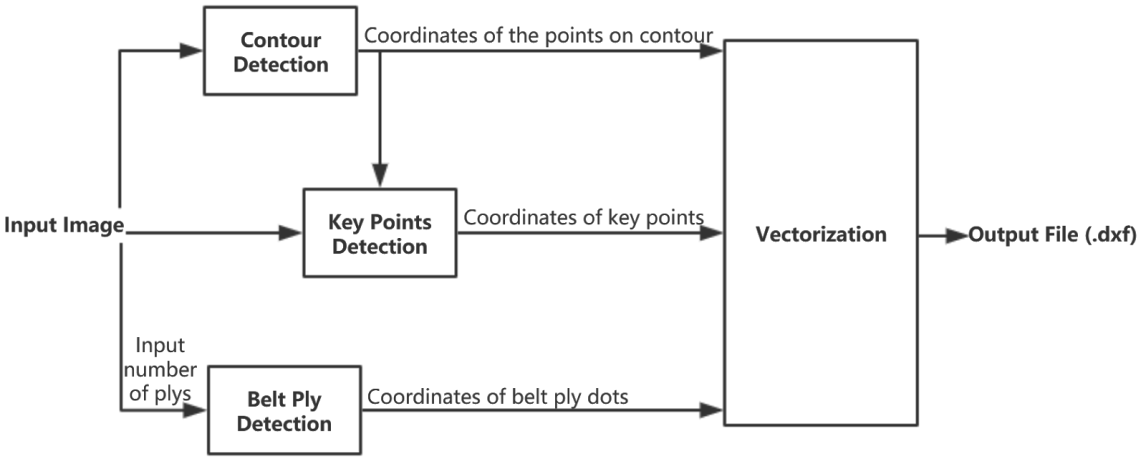


Figure 13. Overall block diagram

B. Analyze Specifications

The scientific fields involved in our project includes algorithm choice and subfunction structure design. Our design is based on the consideration of the performance, runtime, automation, and stability.

After analyzing those elements which are related to the project, there are tradeoffs between these elements like the runtime ad performance contradict each other. We found out that detecting contour first and use the information of the contour in order

to get other geometrical elements has many advantages. Due to its' stability and good performance, we chose it as our solution to the problem.

C. Determine Specific Parameters

We have several main parameters in our project. In the input, we have the parameter to determine how many belt plies we are going to recognize. This parameter determines the operation function of belt ply and the dilate and erode level of the picture. The second parameter is the threshold. There are several different thresholds in different condition to eliminate the influence of the noise. Also, we have another parameter which is called dpi(dots per inch). Dpi is a parameter which relates the vector information to its true distance. According to its name, it records that in every inch in the real distance we convert it into such number of dots on our screen.

Since our project is a software project, it is nonsense to determine the parameter of shape or other value. And with the help of image processing, lots of parameter such as the gray value of each dots can be detected directly from the image.

D. Engineering Logic

To complete our project, there are several steps which are listed in the following.

1. Plan

We made our project plan according to the information we get during the kick off meeting and the schedule of the capstone design.

2. Search

We did literature research and find several papers to solve part of the problem.

According to the search results and the feedback from the kick off meeting, we initially formed some thoughts to solve the question

3. Design

Using the method of concept generation and selection, we choose the best way to solve the problem and divide the entire project into three major tasks. To solve the three tasks, we go through several times of meeting and researching.

4. Test

To evaluate and test our plan, we do some experiment to test the potential solution in each major tasks. And then we evaluate the results with our expectation.

5. Revise

We make the prototype and deliver it to the sponsor to get more feedback. Meanwhile, we try to make our project to fulfill more functions by ourselves.

6. Delivery

Finally, after output vector image satisfies the demands of our sponsor, we organize our code, insert the comments, add instructions in the readme file and pack it up to the sponsor as our deliverable[2].

PART VI Final Parameter Analysis

1. `min_size`: After thresholding, noise smaller than `min_size` will be removed. We determine this parameter by manual binary trials. The largest noise we have seen is of size 12000, while the smallest one is about 0. So we start by testing `min_size = 6000` on all the test images. We see that 6000 is too large because this `min_size` sometimes remove part of the tire 2D cross section if thresholded at a relatively high level. Therefore, our testing scope reduces from 0 to 12000 to 0 to 6000. We then test and find that `min_size = 3000` works well on all test images, so 3000 is chosen to be the value we use.
2. `num_of_seg`: This is the parameter indicating how many segments we want in the step of detecting belt plies. We determine this parameter by a process similar to

[2] The sentence responds to Prof. Pradeep Ray's comment in Design Review 2. He asked what is our final deliverable for this project.

cross-validation. We use 5-fold, namely, we divide all the images into five sets. Each time, we pick one set as our validation set and the rest sets as test sets. Optimal `num_of_seg` is calculated on test sets and the accuracy is recorded on the validation set. At last, we average the first three `num_of_seg` of better accuracy to be our final choice. The final value we use is 36.

3. `max_test_num`: This parameter denotes how many tests on the threshold we want to perform based on our estimated threshold. This is really a parameter about trade-offs. Increasing `max_test_num` by 1 means increasing runtime by approximately 1 second, but at the same time, we may get more accurate threshold because we perform one more test. For now, the value we use is 15 so we get a relatively accurate threshold in a relatively short time.
4. `cluster_dist`: This parameter denotes the minimal distance between two points so that they are identified as one cluster. We test on all test images and find the average distance between points (after Ramer-Douglas-Peucker), which is $\bar{d} = 35$ pixels, and the standard deviation $\sigma = 15$. By Law of Large Number, we may approximate the distribution of point distances as a normal distribution, namely,

$$D \sim Normal(\bar{d},\ \sigma) = Normal(35,\ 15).$$

Therefore, about 84% of point distances are below $35 + 15 = 50$. We simplify points (identified as a cluster) to two points so that we have a desirable average point distance of 60. In practice, the desirable average point distance varies from contour segments to contour segments, 60 is just an empirical value acceptable for all segments.

5. `epsilon`: This is the parameter we mention in Ramer-Douglas-Peucker Algorithm. According to our sponsor, the contour detection error should be within 1mm, i.e., $e < 0.5$ mm. Normally, the DPI of our test image is (600, 600), which means a single pixel has height and width of $\frac{2.54}{600} = 0.0423$ mm, therefore we allow

an error of $0.5/0.0423 = 11.8 \approx 11$ pixels. In practice, since we also gain error in `secondReduceCnt` and `thirdReduceCnt`, we use $\varepsilon = 5$ pixels.

PART VII Final Design

Our design can be decomposed into three parts, namely contour detection, key points detection and belt ply detection.

A. Contour Detection

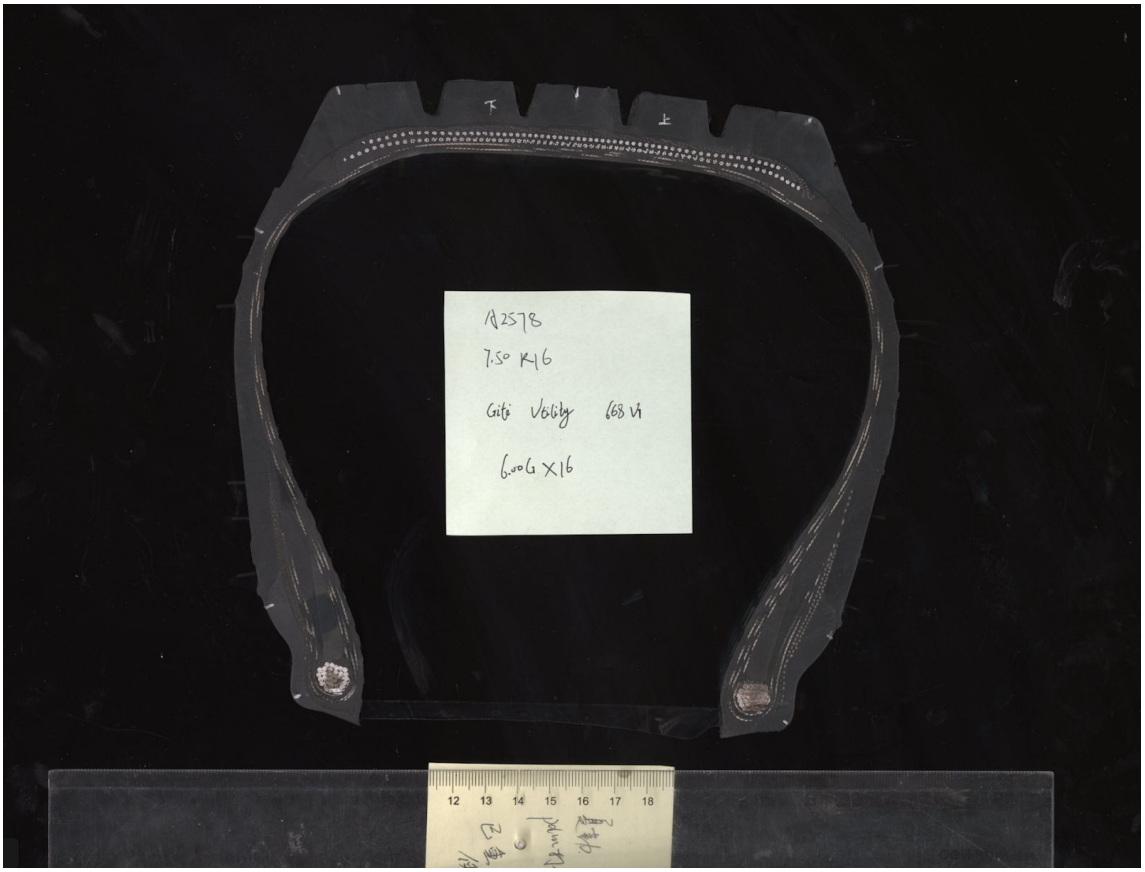


Figure 14. Original image with white paper and ruler

Given a raw tire 2D cross-section image, our first step is to extract the rough tire cross-section itself from its noisy background. To do so, a threshold is estimated and applied to the image. The threshold we used is estimated as followed:

1. we plot the histogram of the grayscale image (if it is RGB, then it is converted into grayscale) ;
2. we find the first two local maxima of the histogram;
3. the gray level at which the histogram attains minimal between the gray levels that attain the first two local maximal is used as the threshold.

Note that the gray level that attains the first local maximal is roughly the average gray level of the background, and the gray level that attains the second local maximal is roughly the average gray level of the tire 2D cross-section. Therefore, the threshold we used is between the average gray levels of the tire cross-section and the background at least. Moreover, the threshold we used has the property that the image after thresholding is most stable in the sense that a small change in threshold leads to least change on the thresholded image.

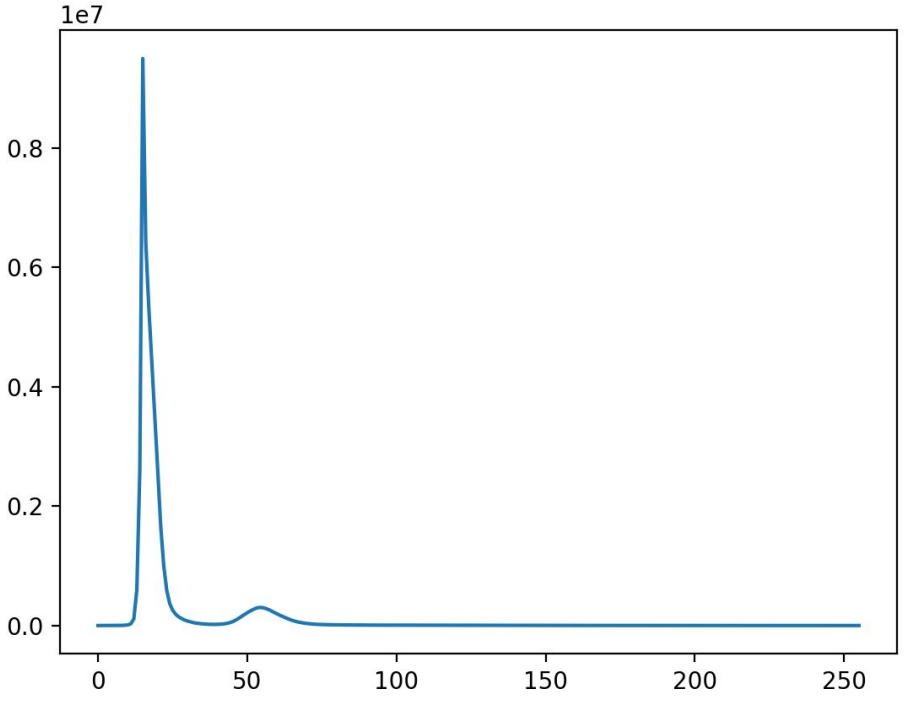


Figure 15. Histogram of the grayscale image

After thresholding, what we get is a binary image with a lot of noise. Our next stop is to remove different types of noise from the binary image. There are basically three types of noise as followed:

I. background noise: This type of noise occurs in that although the background is on average darker than the threshold we use, there is a possibility that some tiny points on the background exceed the threshold. These points after thresholding will become bright points scattering in the background.

II. internal noise: Internal noise occurs because the internal area (of the tire cross-section) is filled with extremely bright elements, such as belt plies. At the phase of contour extraction, we do not need the elements on the tire cross-section, so these elements are also treated as noise.

III. other noise: By other noise, we mean manually added noise such as white description paper or rulers at the bottom of the image.

In fact, to get better results, type III noise is removed at the very beginning (before thresholding). To remove the piece of white description paper, an empirical threshold (200 in grayscale) is applied to get another binary image.

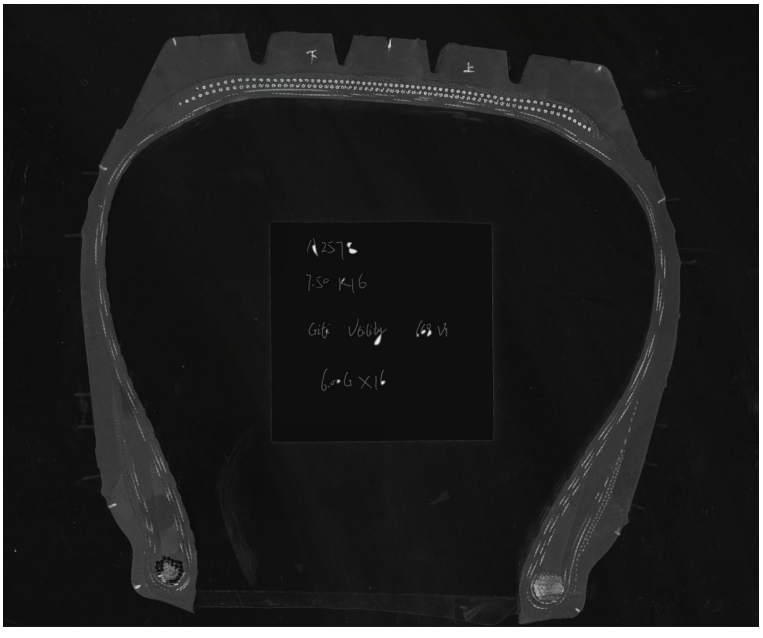


Figure 16. Output after removing the ruler and paper

After removing small objects and holes on the binary image, we obtain a mask of the white paper. We then use the mask to eliminate the white paper from the raw image. To remove the ruler, the sum of gray levels along the x-axis is calculated. Empirically, the argumental minimal of the lower half of the image is the border that divides the tire cross-section and the ruler. Removing type I and type II noise is relatively easier because OpenCV provides functions to remove blobs or holes smaller than a given area.

After the noise removal step, we get a clean binary image with only the tire cross-section. To extract the real contour, we start from the rough contour of this clean binary image. This step is achieved using OpenCV functions.

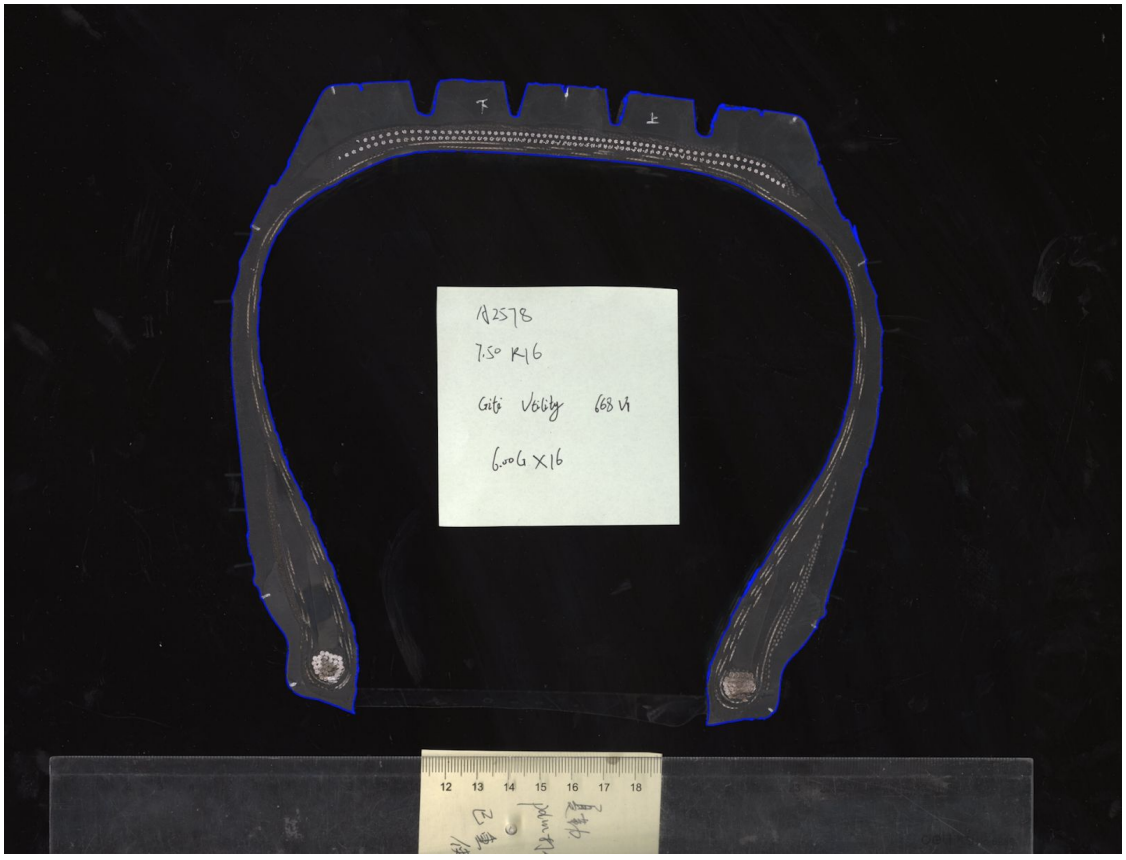


Figure 17. Rough contour

It is worth noticing that the rough contour we get is very inaccurate (just as its name) because there are two problems, namely burr problem and depth of field problem. To

get the real contour, we sequentially apply three algorithms to refine the rough contour.

1. Ramer-Douglas-Peucker Algorithm: the algorithm recursively divides a set of ordered points. The begin and end points are automatically reserved. After that, the furthest point from the line segment between the begin and end points is found. If the distance from this point to the line segment is smaller than a given epsilon, then this point is discarded, otherwise, it is reserved. The algorithm then performs the same operation on the divided two parts. It is self-evident from the description that the number of points in the set will be reduced. At the same time, the maximal error from the real point set to the set after this algorithm is less than the given epsilon with probability. The pseudo code of the Ramer-Douglas-Peucker Algorithm is attached in the following:

```
function DouglasPeucker(PointList[], epsilon)
    // Find the point with the maximum distance
    dmax = 0
    index = 0
    end = length(PointList)
    for i = 2 to ( end - 1) {
        d = perpendicularDistance(PointList[i],
Line(PointList[1], PointList[end]))
        if ( d > dmax ) {
            index = i
            dmax = d
        }
    }

    // If max distance is greater than epsilon,
recursively simplify
    if ( dmax > epsilon ) {
        // Recursive call
        recResults1[] =
DouglasPeucker(PointList[1...index], epsilon)
        recResults2[] =
DouglasPeucker(PointList[index...end], epsilon)

        // Build the result list
        ResultList[] =
{recResults1[1...length(recResults1)-1],
recResults2[1...length(recResults2)]}
    } else {
```

```
            ResultList[] = {PointList[1], PointList[end]}
        }
        // Return the result
        return ResultList[]
end

function firstReduceCnt(PointList[])
     for contourSegment in contourSegments {
          contourSegment =
DouglasPeucker(contourSegment, 5)
     }
```

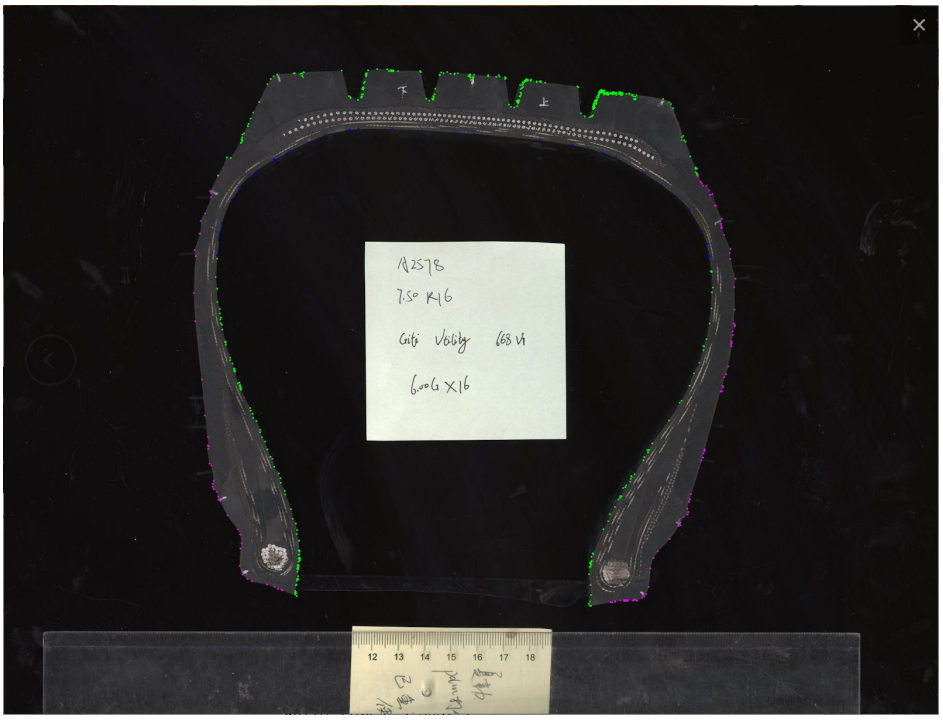


Figure 18. Output after Ramer-Douglas-Peucker Algorithm

2. Cluster Removal: there will be sometimes cluster of points on the rough contour because the contour is obtained by finding contour of a thresholded binary image. At this step, a parameter denotes the minimal distance between two points so that they are not identify as cluster is carefully examined under experiments.

```
function secondReduceCnt(PointList[], cluster_dist)
     idx_1 = 1
     end = length(PointList)
     while (idx_2 < end) {
          // Find cluster start point
          while (idx_2 < end) and
(dist(PointList[idx_2], PointList[idx_2 - 1]) <
cluster_dist)) {
```

```
                idx_2 = idx_2 + 1
            }
            idx_1 = idx_2 - 1

            // Find cluster end point
            while (idx_1 < end) and
(dist(PointList[idx_2], PointList[idx_2 - 1]) >=
cluster_dist)) {
                idx_2 = idx_2 + 1
            }
        }

        // Remove Points indexed from idx_1 to idx_2
        RemovePoints(PointList, idx_1, idx_2)
```

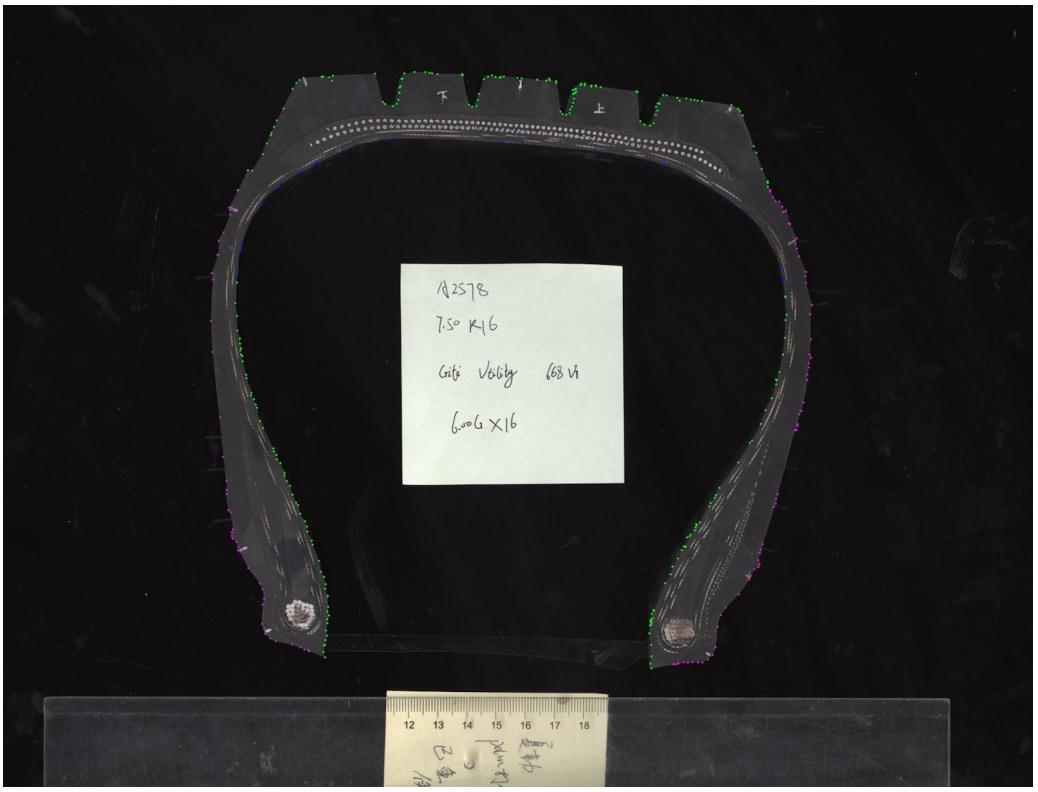


Figure 19. Output after cluster removal

3. Support points finding: This is the key step to partially solve depth of field problem. From observation, we find that the structure of the real contour is actually maintained by a subset of points on the rough contour. These points are named support points in the sense that they are there to prevent the contour from shrinking. These points are found by iterating the rough contour counter-clockwisely. For a given support point, the next support point is found by first finding a list of suspects

(points that are within a given distance from the support point) and then choosing the innermost point.

```
function removeBackWardPoints(PointList[], direction,
givenVal)
     end = length(PointList)
     for index = 1 to end
          PointDirection = PointList[index] -
PointList[index - 1]
          ScaledInnerProduct = PointDirection *
direction / (norm(PointDirection*direction))
          if (ScaledInnerProduct < givenVal) {
               delete(PointList[index])
               break
          }

function thirdReduceCnt(PointList[], sticklength)
     // inner contour is further segmented into five
pieces
     contourSegments = furthurSegment(contourSegments)
     for contourSegment in contourSegments {
          end = length(contourSegment)
          for (index = 1 to end) {
               direction =
get_direction(contourSegment[index])
               NearestPointList =
getPointList(PointList, sticklength)
               if NearestPointList is empty {
                    index = index + 1
                    continue
               } else {
                    NextPoint =
getInnerMostPoint(NearestPointList)
                    index = index_of(NextPoint)
               }
          }
     }
     for i = 1 to 100 {
          for contourSegment in contourSegments {
               direction = get_direction(contourSegment)
               contourSegment =
removeBackWardPoints(contourSegment, direction, 0.5)
          }
     }
```

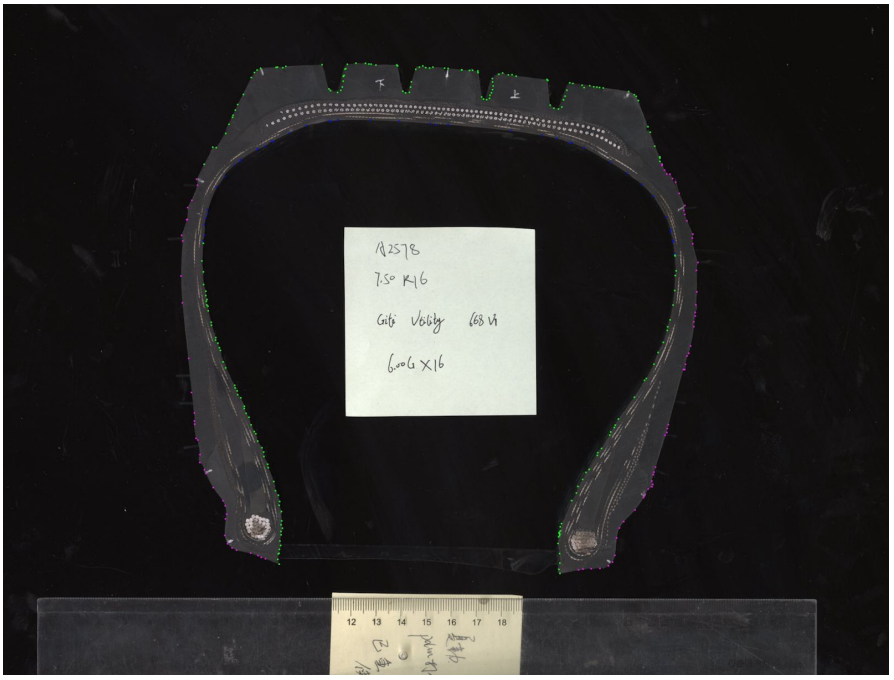


Figure 20. Output after finding support points

These three algorithms work together to give our estimated real contour: Ramer-Douglas-Peucker algorithm ensures the error between the estimated real contour and the rough contour is reasonable; cluster removal makes the contour concise and clears the way for support points finding; support points finding then make sure most burr and depth of field problems are solved.

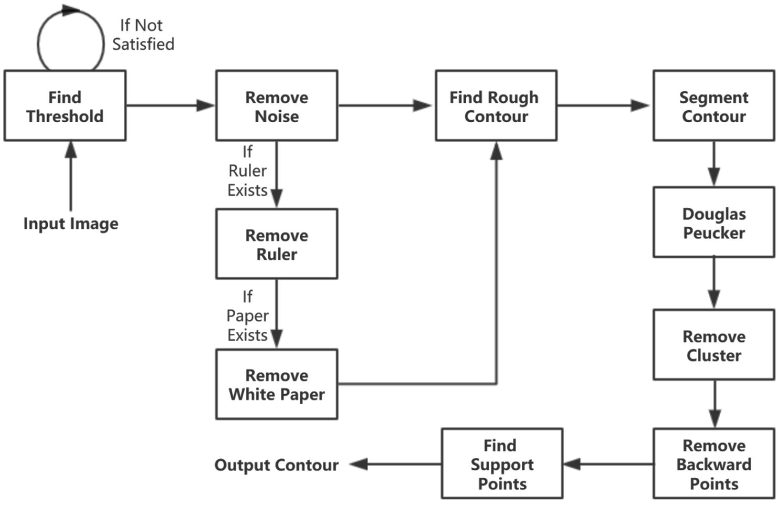


Figure 21. Flowchart of contour detection

B. Key Points Detection

Key points detection is done by iterating on the outer contour we find in the contour detection part. When iterating, if the number of white pixels near this point exceeds a given amount, this point is labeled as a suspect for the key point. In fact, what we actually get by iterating the contour are several clusters of suspects. For each cluster, we then calculate their center to give a real key point.

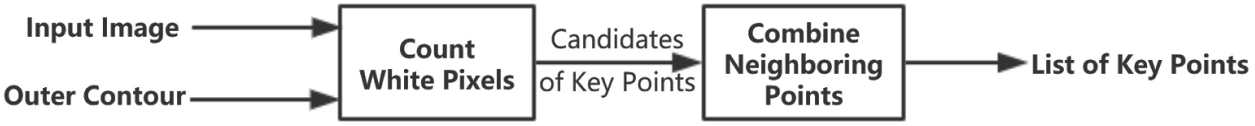


Figure 22. Flowchart of key point detection

C. Belt Ply Detection

Belt ply detection also relies on the contour segmentation we extract from contour detection. By finding the uppermost point from both inner contour and the outer contour, we give an estimation of the area in which belt plies lie. This area is then extracted for further analyze. After edge detection on this area, we cut it into several smaller segments. We then dilate and erode each small segment with proper kernels in hope that each belt ply is connected and not overlapped with each other. The kernel we use is estimated by the direction of inner contour that lies in the segment.

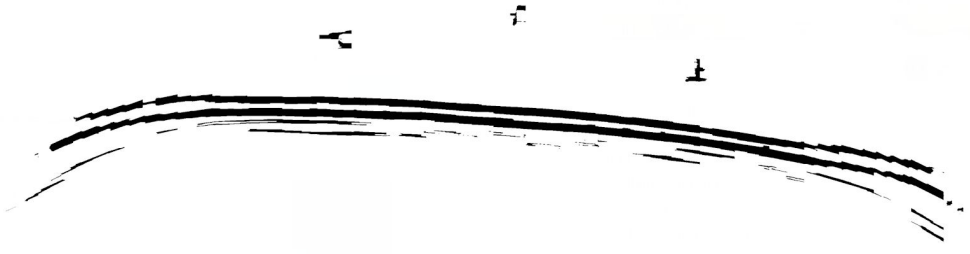


Figure 23. Output after dilating and eroding

After dilating and eroding the area with different kernels, we find contours of each belt plies. The points on each contour is then sorted according to their x-coordinates. For each x-coordinates, y-coordinates is calculated to be the average of the y-coordinates of points whose x-coordinates are close enough to the x-coordinate. We then connect these points and get several smooth and relatively accurate belt plies.

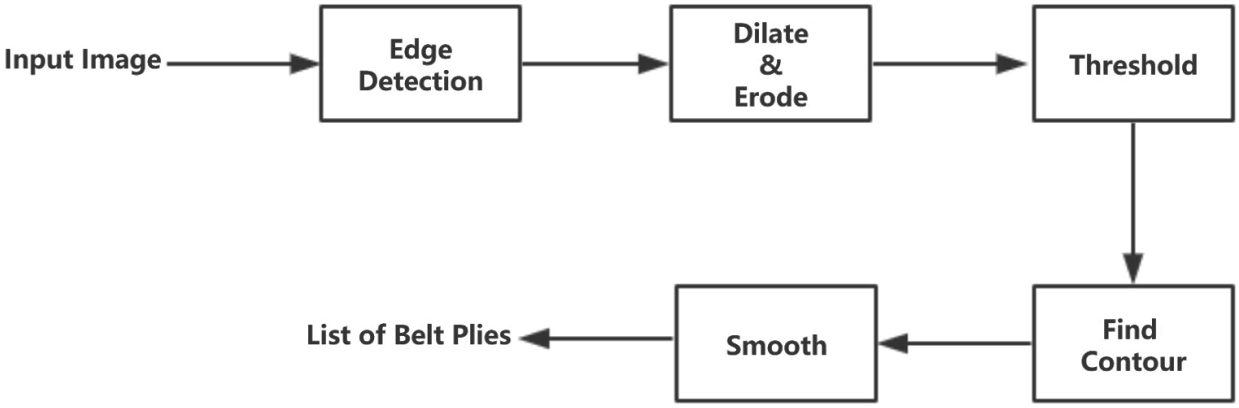


Figure 24. Flowchart of belt ply detection

D. Final result

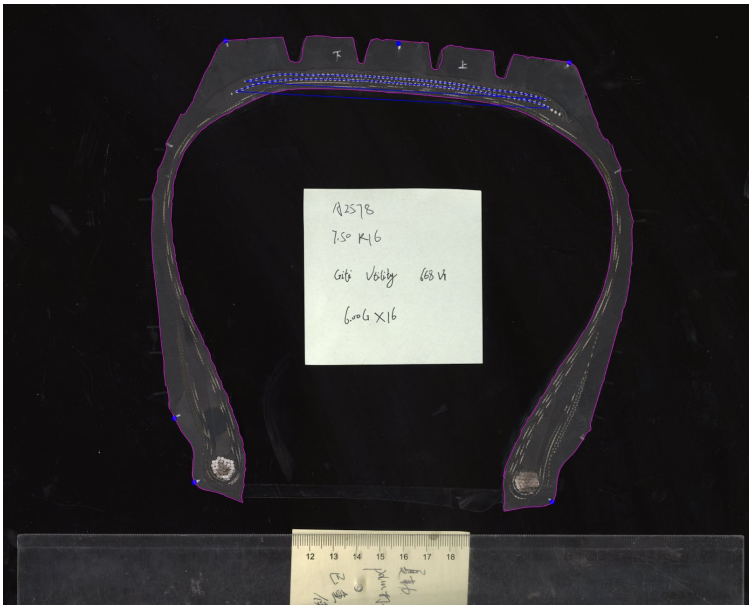


Figure 25. Final result

PART VIII Manufacturing Plan

Programming language

In this project, we will use Python 3 as our programming language, and it is the only programming language we use in the entire project.

Source library

We use existing libraries, which includes OpenCV, Pillow, Numpy, Skimage and Dxfwrite in Python.

Software platform

Our project is based on Python programming language so that the operation platform is all the computer which install the Python3 and the library we used.

Algorithm

The general algorithm we use is image processing namely findcontour function in Python. And we use three sub-algorithms, Ramer-Douglas-Peucker algorithm, cluster removal and support points finding, which we have already explained in the previous part of the report.

Budget consideration

In this project, we are required to use the open source code, so that our planning budget is zero.

PART IX Test Results

1. Contour detection

The engineering specification for contour detection is the deviation from our output to the corresponding sample answer should be less than 0.5 mm. Therefore, we will design a test program to compare our output and the sample answer. We try to find the maximum distance between the two contours. We record every distance between

the point from the test result and the corresponding point on the graph of sample answer. GITI provide 31 sample answers, so we repeat the test 31 times. There is one outlier in the samples, and the final pass rate is 28/30. The contour of the tire is mistakenly incised so it is in the condition of the outlier. The outlier is shown in the following and GITI satisfies the pass rate.

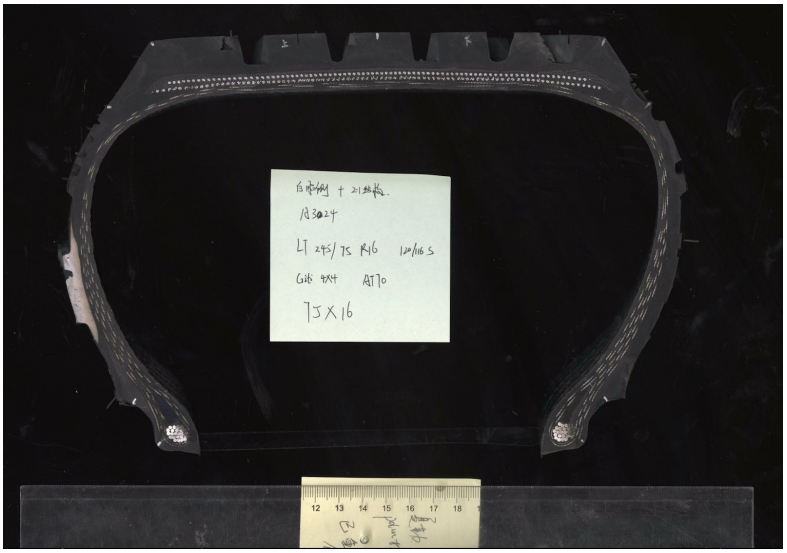


Figure 26. Outlier input picture of contour

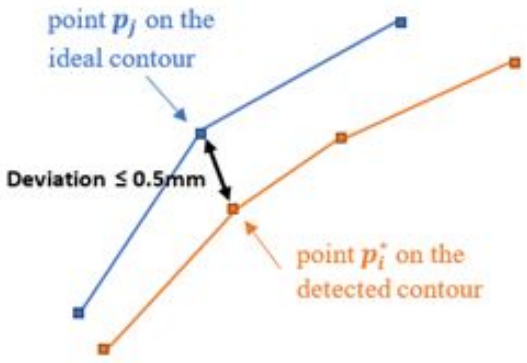


Figure 27. Evaluation criterion of contour

2. Belt ply detection

The engineering specification for belt ply detection is the passing rate of dots should be more than 95%. GITI provides 31 sample test cases so we did 31 times test. There are 7 outliers since there is unexpected interference on the picture. The sample of outliers is in the following. In this outlier, there is a piece of paper under the inner contour which is a kind of illegal input. As soon as we get our output, comparing with the origin input, we use human observe to judge whether the result fit the requirement or not. We manually calculate the pass rate of each test result. And the final pass rate is that 21 of 24 tests pass the rate of 95%.

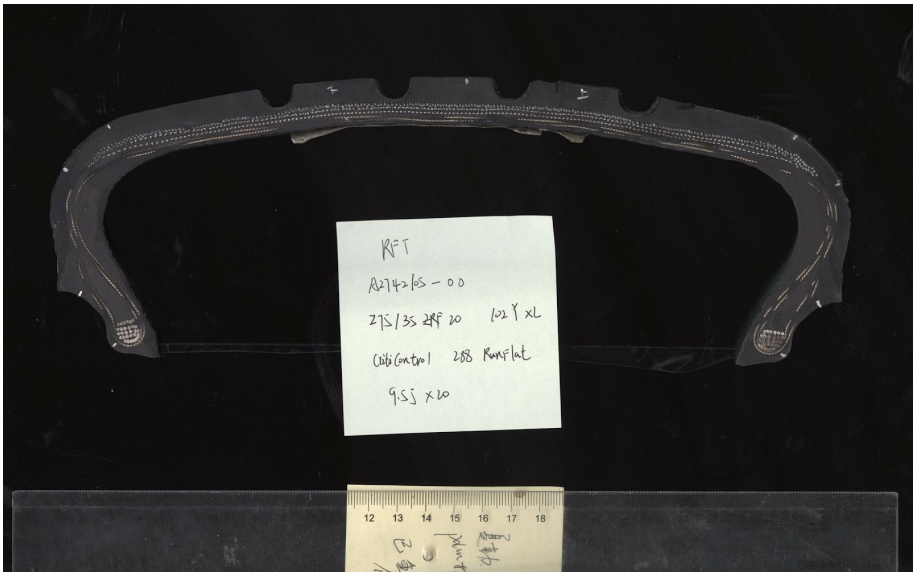


Figure 28. Outlier input picture of the belt ply

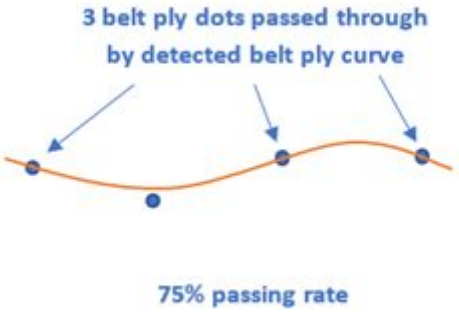


Figure 29. Evaluation criterion of belt ply

3. Key points detection

The engineering specification for key points detection is the deviation from our output to the corresponding sample answer should be less than 1 mm. Therefore, we will design a test program to compare our output and the sample answer. We try to find the maximum distance between the two corresponding points. GITI provide 31 sample answers, we repeat the test 31 times. The definition of the longest distance is the pixel distance from the coordinate of output point to the middle center of the real key point multiplies the rate of DPI. And every time in the test, our contour detection has the deviation within 1 mm which actually fit the requirement.

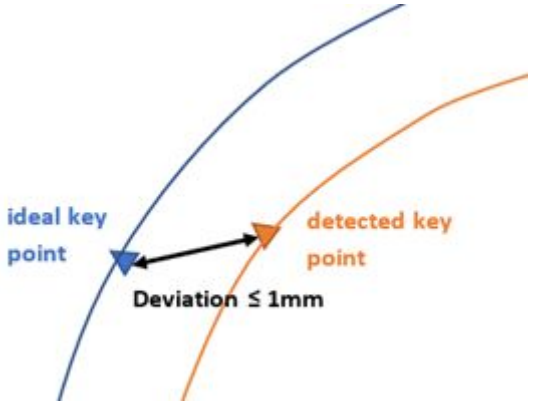


Figure 30. Evaluation criterion of the key points

4. Total run time (include manual operation time)

The engineering specification for total run time is 60 seconds. GITI provides 31 sample test cases so we did 31 times test. We manually time the operation time as soon as we start running our code and stop timing until the final result is created. And every time in the test, our total run time is less than 60 seconds which actually fits the requirement.

5. Vector conversion

The engineering specification for vector conversion is the output .dxf file should have the same size as the original input file. Therefore, we compare the size of our output

and the sample answer. We repeat the test thirty-one times. Multiplying the rate of DPI and the pixel we get from the test result, we calculate the real size of the result. And every time in the test, our output .dxf file has the same size as the input file which actually fit the requirement.

6. Curve fitting

The engineering specification for curve fitting is to use a spline curve. The result is easily human observed and we require GITI's mentors to judge whether our output file meet the requirement or not. Since the judgment criterion is whether the output shape is formed by spline curves, it is the only way to use human observation to watch the detail of the output result.

7. Automatic level

The engineering specification for the automatic level which means manual operation time is 15 seconds. When the user runs the program, he or she needs to input the number of belt ply. The number is easily humanly observed and three of GITI's mentors have the average time of 6 seconds to input the data, which is less than the limit of 15 seconds. Since GITI's mentors are newbies of the program, and the timing result is on average, we can conclude that the result is reasonable.

8. Cost

The engineering specification for cost is 0 yuan. The entire project doesn't cost any money from beginning to end so that the cost is zero. And during our test, we don't cost any money, either.

PART X Engineering Changes Notice

1. Initialization of images has been changed. We assumed the ruler on the input image would align along the bottom, but after DR#3 our sponsor provided new test

images with the ruler not touching the bottom at all. So in the initialization part, we change the code to detect the first local minimum if there are two local minimum.

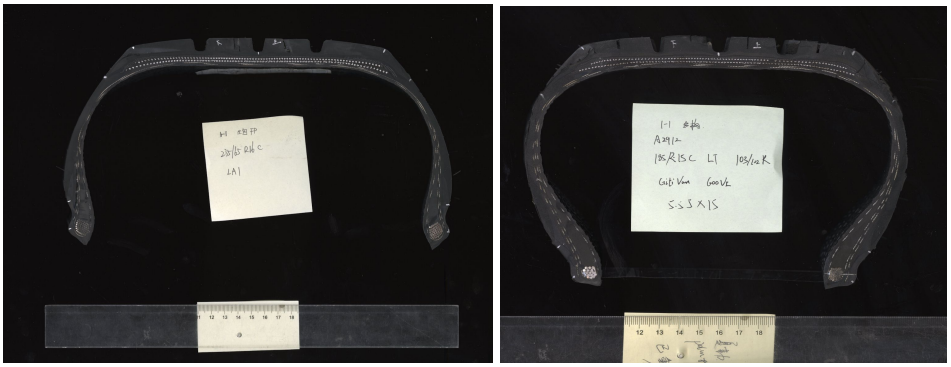


Figure 31. Ruler with/without touching the bottom

2. The threshold is now determined by estimation and trials. Compared to the original design, our program runs 40% faster. Method of estimation has also been changed. Based on the originally estimated threshold, we further increase the threshold so that a very small portion of pixels (which are believed to be the depth of field) are excluded.
3. The function `secondReduceCnt` has been updated. Now we no longer remove backward points at the beginning, instead of removing backward points is performed after clusters are removed. Original design leads to the problem that sometimes `secondRuduceCnt` gives up too many details about the contour. This problem was brought up by our sponsor after DR#3.
4. The original design of the function `thirdReduceCnt` removes crucial points on the contour if the inner contour has four turning points instead of two. Our sponsor found this problem and required a change in design. The function `thirdReduceCnt` now further segments inner contour into 5 pieces.

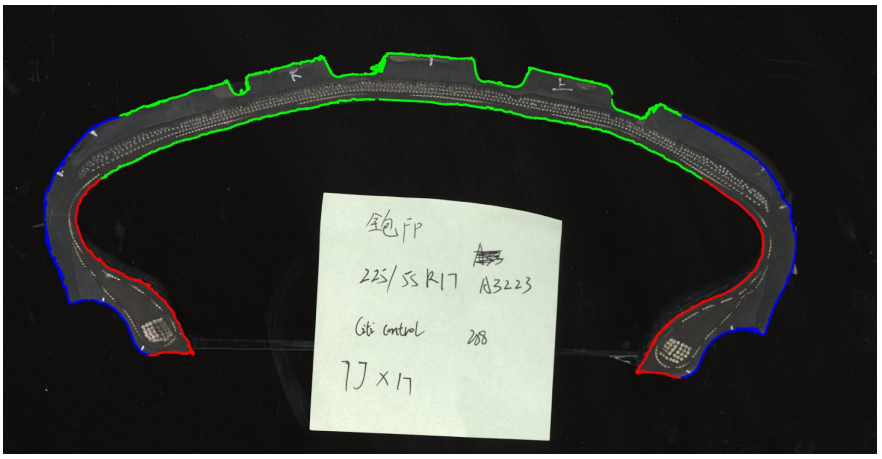


Figure 32. Segments of contour

PART XI Project Plan

A. Milestones

Milestone 1 (6/11/2018)

We did some literature research and figure out the conceptual idea of the whole project. We apply some existed edge detection algorithm to the sample input image and look into possible solutions to the detection of belt ply. Report and slides for design review 1 are finished.

Milestone 2 (6/27/2018)

The inner and outer contour detection is achieved. The algorithm used for excluding the influences of burrs, shadows, and depth of field is waiting to be polished. Belt ply detection is achieved.

Milestone 3 (7/18/2018)

Key points detection is achieved. Different inputs are tested to figure out the reliability of the algorithm. Vectorization is completed. Slides, report and readme file are in progress at this stage. In details, we remove interference from slips and rulers,

and distinguish close belt plies. Before design review 3, multiple pictures automation finished. We also connect to the sponsor company to negotiate a secret agreement.

Final EXPO (8/8/2018)

The whole algorithm is integrated. Prototyping video is taken. The performance of the algorithm is recorded. Slides, report and readme file are finished.

B. Division of Labor

Sort the priority of the tasks from high to low: Detection of contours, detection of belt ply, detection of key points, achieve vectorization, exclude the shadows, burrs, and depth of fields.

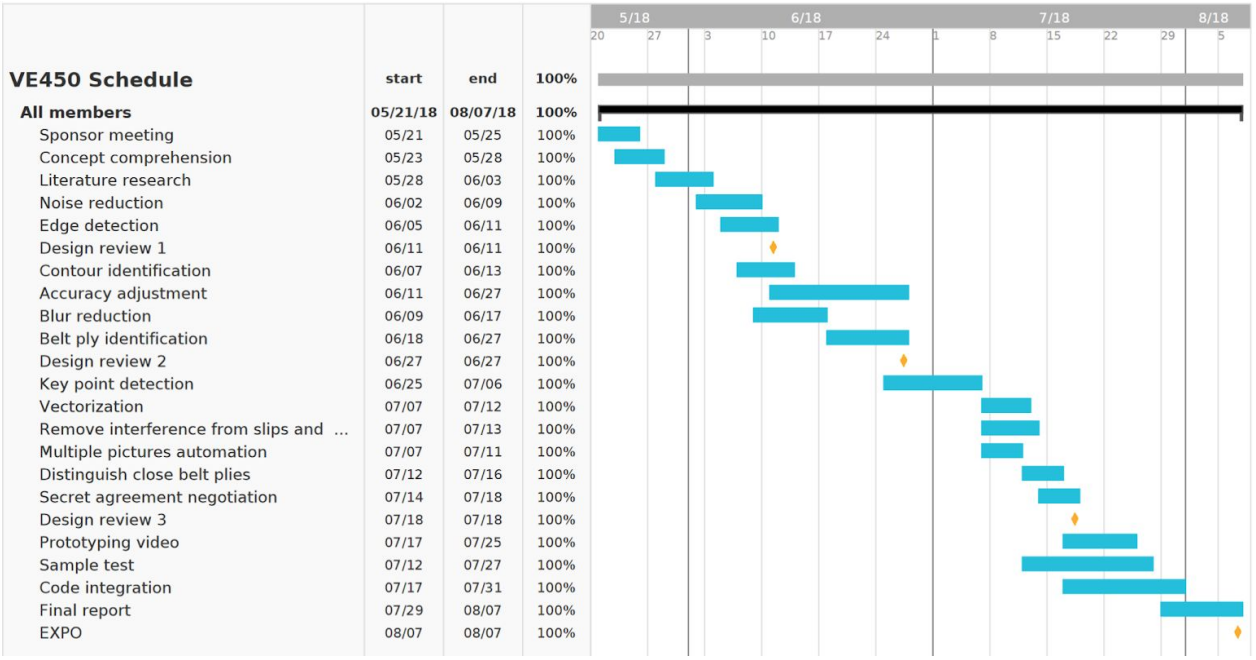
According to the priority, two members would work on the contour detection, one member works on detecting the belt ply and key points, one work on excluding the shadows, burrs and depth of fields and one work on the report and slides. After the extraction is finished, all of us would spend some time looking into vectorization. Then before the design review, the report and slides would be peer-reviewed by us all.

Since it is a software project and the platform we are using is Python, so the budget could be neglected.

C. Detailed Schedule & Gantt Chart

In the first week, we had a sponsor meeting to understand customer requirements. Then we organized the concept and did literature research. All group members focused on noise reduction and edge detection. After the first milestone, Lina Zhang and Yucheng Shi finish the task to identify the inner and outer contour. Then they do some accuracy adjustment to make it better. Jiaqing Ni, Rishang Xu, and Yiting Luo finished the task of blur reduction so that Lina Zhang, Jiaqing Ni and Rishang Xu will do the next step to identify the belt ply. Before the third milestone, Yiting Luo and Yucheng Shi will realize the key point detection and optimize the algorithm. Then all team members will pay attention to vectorization. After we finish some sample test,

we go through the final stage. We will improve the code performance and integrate the code together. In details, the improvement includes remove interference from slips and rulers, and distinguish close belt plies. Jiaqing Ni and Lina Zhang will take charge of it. Before design review 3, multiple pictures automation will be finished by Yucheng Shi. Rishang Xu and Yiting Luo also connect to the sponsor company to negotiate a secret agreement. Finally, we finish the report and do the EXPO. The corresponding revised Gantt chart is shown below.



VE450 Schedule	start	end	100%
All members	05/21/18	08/07/18	100%
Sponsor meeting	05/21	05/25	100%
Concept comprehension	05/23	05/28	100%
Literature research	05/28	06/03	100%
Noise reduction	06/02	06/09	100%
Edge detection	06/05	06/11	100%
Design review 1	06/11	06/11	100%
Contour identification	06/07	06/13	100%
Accuracy adjustment	06/11	06/27	100%
Blur reduction	06/09	06/17	100%
Belt ply identification	06/18	06/27	100%
Design review 2	06/27	06/27	100%
Key point detection	06/25	07/06	100%
Vectorization	07/07	07/12	100%
Remove interference from slips and ...	07/07	07/13	100%
Multiple pictures automation	07/07	07/11	100%
Distinguish close belt plies	07/12	07/16	100%
Secret agreement negotiation	07/14	07/18	100%
Design review 3	07/18	07/18	100%
Prototyping video	07/17	07/25	100%
Sample test	07/12	07/27	100%
Code integration	07/17	07/31	100%
Final report	07/29	08/07	100%
EXPO	08/07	08/07	100%

Figure 33. Project Gantt Chart

PART XII Analysis of Potential Problems

In our project, we are going to eliminate a question in belt ply part. If there are two or more belt plies diverge during the line, the recognition at the end of the belt ply is not well. The recognition procedure will combine the two lines together since they are too close. We are going to segment the complex belt ply, for example, to let the program analyze the end of belt ply with two lines. There is a way to solve the problem that we can build a database and model the pattern of belt ply. However, the company refused our requirement since they don't want to leak the data (see

footnote for more information)[3]. Secondly, we need to automate the program more such as rotate the input image automatically. Thirdly, we need to make our program more user-friendly. To make the users more convenient to know which type of tire they are recognized, usually, a slice of paper will be stuck under the inner contour of the tire. Since our algorithm segment the tire into three part and set the different threshold values to image processing, it is hard for us to eliminate the influence caused by that paper right now.

PART XIII Discussion

In our final design, we add some functions which don't include in the proposal. We automatically remove the interference of ruler, paper, and white line. The information we need when we input the code is the tire type so that the program is highly automatically. We add the subprogram to detect whether there is a ruler or paper on the image and use the algorithm to remove it. The detailed algorithm is explained in the part of engineering changing notice. Meanwhile, we provide two kinds of contour detection way so that the user can judge the trade-off between operation time and accuracy. However, our design has weakness in belt ply part. We are not allowed to build the database of belt ply patterns so that it is possible to have particular cases of belt ply that we can't give a wonderful solution under that condition.

PART XIV Recommendations

In this project, there are some recommendations for both sponsors and future students.

For sponsors, it is better to define each element more clearly in the proposal. At the beginning of the project, we were confused at the definition of key points and felt

[3] The sentence responds to Dr. Yunlong Guo's comment in Design Review 3. He asked why we didn't build a database of patterns for belt ply to solve the oversight.

hard to recognize them. It is better to use more sample pictures to show the detailed definition of every element. Secondly, during the test, it is better for the sponsor to send the exact test samples to us so that we can modify the program more efficiently. In this project, some of the test samples are classified. Therefore we can't directly find out what the problem is and go straight to the heart of the problem. And this problem influences the final percent of the pass since some of the samples are outliers which don't fit the requirement of input. Thirdly, to better solve the problem of depth of field, the sponsor can consider putting two or more camera to take the picture. With two different points of view, we can calculate the depth of the picture. With the depth, we can remove the depth of field.

For students, we suggest future students ask more information about tires to the company. Since it is a project about image processing, Convolutional Neural Network is a more precise way to handle the problem. For example, to solve the particular case of belt ply, the best solution is to build a database of the patterns of belt ply. However, GITI doesn't allow us this time. Another recommendation for solving the subjective the problems is to design a User Interface to let the tester select the parameters.

PART XV Conclusions

Our project aims to develop a software which could transform several geometrical elements in tire-2D section image into vector data. There are influence factors needed to be excluded, like the burrs, the depth of field, the shadow, the noise, etc. Then the vector data could be directly used for measuring the dimensions of the tire. After the output vector image satisfies the demands of our sponsor, we organize our code, insert the comments, add instructions in the readme file and pack it up to the sponsor as our deliverable. The whole project should be of a high automatic level and accurate. The division of labor is arranged properly.

PART XVI Acknowledgements

We would like to thank GITI company for providing us the chance and the resources. We would like to thank Rui Deng from GITI company for guiding us and providing innovative ideas for the project. We would also like to thank Prof. Long, Prof. Ma, the other course instructors and teaching assistants for assisting us over many office hours with our projects.

PART XVII Bill of Materials

Our project is based on programming and it is free.

PART XVIII Bios

张立娜 Lina Zhang
Major: ECE + CE
Experience:
Fine-grained Dog Breed Classification: CNN, SIFT, SVM
Object Collecting and Sorting Robot: Detection, Tracking
Phone: 13127759645
Email: linazh@umich.edu

Future Plan: My name is Lina Zhang. I major in ECE in JI and CE in UM. I will work as a software development engineer for Amazon in Bay Area after graduation. I will return to the team in which I interned last summer. I really look forward for the life in Bay Area, with sunshine and good Asian food.

史玉成 Yucheng Shi

Major: ECE + Math

Experience:

Mathematical modeling, machine learning

Phone: 13817152006

Email: yuchengs@umich.edu

Future plan: My name is Yucheng Shi. I major in ECE at JI and Honor Mathematics at U-M. I will finish another major CS at U-M in the coming school year. After that, I will probably apply for PhD programs in pure math or master programs in CS.

倪佳清 Jiaqing Ni

Major: ECE + CS

Experience:

CNN & Image Classification

Adobe Photoshop/Illustrator

Phone: 15900402661

Email: jiaqni@umich.edu

Future plan: For the coming school year, I will continue study at UM in CS master's program, specifically in the field of data mining and machine learning. I hope to gain some practical experience through researching and internship during the future two years to prepare myself for a position in IT companies when I obtain master's degree.

骆仪婷 Yiting Luo
Major: ECE + CE
Experience:
Image processing (machine learning, pattern detection, compressive sensing)
Phone: 18217233273
Email: luoluo@umich.edu

Future Plan: I major in ECE in JI then I learn CE in UM. I plan to finish my master degree program of Electrical Engineering in Texas A&M University.

徐日上 Rishang Xu
Major: ECE
Experience:
Image processing (point feature matching)
Phone: 18217559392
Email: xurishang@sjtu.edu.cn

Future Plan:My name is Rishang Xu and I major in ECE in JI. I plan to finish my master degree program of Electrical Engineering in Texas A&M University. Since I am interested in signal processing and digital circuit, I choose the mixed signal track in TAMU.

PART XIX REFERENCES

[1] Y. Yoon-Mo, J. Jae-Moon, K Seon-II, K Ki-Jeon, "Non-contact measuring apparatus for the section profile of a tire and its method," United States Patent, Patent Number: 5,506,683

[2] Ruchi, S. Dasgupta, "Method for evaluating ply wire anomalies in a tire", United States Patent, Patent Number: US 7,416,624 B2

[3] A. Jacques, "Shaping and laying a tire belt ply," United States Patent, Patent Number: US 7,138,021 B2

[4] Zhang Y, Funkhouser T. Deep Depth Completion of a Single RGB-D Image[J]. 2018.

[5] Liu F, Shen C, Lin G, et al. Learning Depth from Single Monocular Images Using Deep Convolutional Neural Fields[J]. IEEE Transactions on Pattern Analysis & Machine Intelligence, 2016, 38(10):2024-2039.

PART XX Appendix

As for the concept of using cameras instead of using scanner in our project, the Giti company indicates that it is of high cost for them to change there facilities so we didn't put the concept in our concept selection part.

The principle of the concept is called parallax. When looking out of the side window of a moving car, the distant scenery seems to move slowly while the lamp posts flash by at a high speed. Parallax can be exploited to extract geometrical information from a scene. From multiple captures of the same scene from different viewpoints, it is possible to estimate the distance of the objects.

By tracking the displacement of points between the alternate images, the distance of those points from the camera can be determined. Different disparities between the points in the two images of a stereo pair are the result of parallax. When a point in the scene is projected onto the image planes of two horizontally displaced cameras, the closer the point is to the camera baseline, the more difference is observed in its relative location on the image planes. Stereo matching aims to identify the corresponding points and retrieve their displacement to reconstruct the geometry of the scene as a depth map.

After we have the depth map, we can use it on our input image to eliminate the influence of depth of field. However, the facility of Giti company is scanner, which has different principle from cameras. It makes this concept hard to be implemented.

Code:

```
import cv2
import numpy as np
from skimage import morphology, feature, exposure
from PIL import Image
import functools, time
from dxfwrite import DXFEngine as dxf
from matplotlib import pyplot as plt
import matplotlib.image as mpimg

Out_File_Type = 'jpg'
max_test_num = 15
```

```
num_of_seg = 36
safe_seg_coeff = 0.9

def metric(fn):
 '''Decorator to time functions.'''
 @functools.wraps(fn)
 def wrapper(args, **kwargs):
      start_time = time.time()
      f = fn(args, **kwargs)
      end_time = time.time()
      print('Function %s takes %s' % (fn.name, end_time -
start_time))
 return wrapper

class TireCrossSection:

 """This class provides methods to extract vector information.

 TireCrossSection is written to extract vector information
from a
 tire 2D cross section image. The image can be gray-16bit or
gray-8bit
 or RGB-24bit. Two types of disturbance (white description
paper,
 ruler) are allowed. To extract contour, key points and belt
ply,
 the type of the tire and the number of belt ply are needed.

 """

 def __init__(self, imageName, isPCR = True, belt_num = 2,
datadir = '../../data', outDir = '.', save = False):
      """Initialize the TireCrossSection object and save
arguments

      Arguments:
      imageName -- the name of the raw image
      isPCR -- indicator of a PCR tire (default True)
      whitepaper -- indicator of whether a white description
paper exists
      ruler -- indicator of whether a ruler exists
      belt_num -- number of belt ply (defautl 2)
      datadir -- the diractory that contains the raw image
(default ../../data)
      outDir -- the diractory where outputs will go (default
current dir)
```

```
      save -- indicator of saving intermediate step results
(default False)

      """
      # Save arguments
      self._imageName = imageName
      self._isPCR = isPCR
      self._belt_num = belt_num
      self._datadir = datadir
      self._outDir = outDir
      self._save = save

      # Extract filename w|o extension and extension
      self._fileType = imageName[imageName.index('.') + 1 :]
      self._imageName_plain = imageName[:imageName.index('.')]

      # Open file with PIL to get access to dpi info and mode
info
      image_PIL = Image.open( self._datadir + '/' +
self._imageName)
      self.dpi = image_PIL.info['dpi']
      print('Image mode is ' + image_PIL.mode)
      if image_PIL.mode == 'RGB':
           self._originalImage = cv2.imread(self._datadir +
'/' + self._imageName)
           self._grayImage = cv2.cvtColor(self._originalImage,
cv2.COLOR_RGB2GRAY)
           self._grayImage_8 = self._grayImage.copy()
           self._blankImage = self._originalImage.copy()
           self._bitNum = 8
      elif image_PIL.mode == "I;16":
           #  If an image is 16 bit gray blankImage is divided
by 256 for display
           self._originalImage = cv2.imread(self._datadir +
'/' + self._imageName, 2)
           self._grayImage = self._originalImage.copy()
           self._grayImage_8 = self._grayImage.copy()/256
           self._blankImage = self._originalImage.copy()/256
           self._bitNum = 16
      else:
           self._originalImage = cv2.imread(self._datadir +
'/' + self._imageName)
           self._grayImage = cv2.cvtColor(self._originalImage,
cv2.COLOR_RGB2GRAY)
           self._grayImage_8 = self._grayImage.copy()
           self._blankImage = self._originalImage.copy()
           self._bitNum = 8
```

```
        self._h, self._w = self._originalImage.shape[:2]

        self._removeRuler()

        # rough histogram with white paper and ruler on img
        histr = cv2.calcHist([self._grayImage], [0], None,
[2**self._bitNum], [0, 2**self._bitNum])
        _local_maximal = []
        for i in range(2**(self._bitNum - 8) * 10,
int(2**self._bitNum/3)):
            flag = True
            for index in range(1, 2**(self._bitNum - 8) * 10):
                flag = flag and (histr[i + index] < histr[i])
            for index in range(1, 2**(self._bitNum - 8) * 10):
                flag = flag and (histr[i - index] < histr[i])
            if flag:
                _local_maximal.append(i)

        num_of_white_pixel =
len(self._grayImage_8[self._grayImage_8 > 200])

        if num_of_white_pixel * 30 > self._h * self._w:
            self._removeWhitePaper(_local_maximal[0])

        if self._save:
            outFileName = self._imageName_plain+'-init.'+
Out_File_Type
            cv2.imwrite(self._outDir + '/' + outFileName,
self._grayImage_8)
            print('Initialized result is saved as ' +
outFileName)

        # recalculate true histogram
        histr = cv2.calcHist([self._grayImage], [0], None,
[2**self._bitNum], [0, 2**self._bitNum])
        plt.plot(histr)
        plt.show()
        _local_maximal = []
        for i in range(2**(self._bitNum - 8) * 10,
int(2**self._bitNum/3)):
            flag = True
            for index in range(1, 2**(self._bitNum - 8) * 10):
                flag = flag and (histr[i + index] < histr[i])
            for index in range(1, 2**(self._bitNum - 8) * 10):
                flag = flag and (histr[i - index] < histr[i])
            if flag:
```

```
                _local_maximal.append(i)
        threshold_list = histr[_local_maximal[0]:
_local_maximal[1]]
        self._threshold_lwb = _local_maximal[0] +
np.argmin(threshold_list)

        self._threshold = self._threshold_lwb
        while histr[self._threshold]-threshold_list.min() <
(histr[_local_maximal[1]]-threshold_list.min())*(1/10):
            self._threshold = self._threshold + 1

        self._threshold_upb = self._threshold_lwb
        while histr[self._threshold_upb]-threshold_list.min() <
(histr[_local_maximal[1]]-threshold_list.min())*(6/10):
            self._threshold_upb = self._threshold_upb + 1


 def _removeRuler(self):
        """Remove the ruler on the image. """

        safe_end = int(safe_seg_coeff * self._h)
        half_img = self._grayImage_8[int(self._h/2) : safe_end,
:]
        _half_sum = np.sum(half_img, axis = 1)
        y_index = int(self._h/2) + (_half_sum).argmin()
        y_value = _half_sum[y_index - int(self._h/2)]
        yp_value = np.sum(self._grayImage_8[y_index: safe_end,
:], axis = 1).max()
        print(y_index)
        print(y_value, yp_value)
        if yp_value > y_value * 3:
            self._originalImage = self._originalImage[:y_index,
:]
            self._grayImage = self._grayImage[:y_index, :]
            self._blamkImage = self._blankImage[:y_index, :]
            self._grayImage_8 = self._grayImage_8[:y_index, :]
            self._h, self._w = self._originalImage.shape[:2]



 def _removeWhitePaper(self, graylevel):
        """Remove the white description paper from the image.
"""
        raw_mask = cv2.inRange(self._grayImage, np.array([0]),
np.array([2**(self._bitNum - 1)]))
        cleaned = morphology.remove_small_objects(raw_mask,
min_size = 3000, connectivity = 2)
```

```
        # dtype is np.uint8 because mask dtype should be
np.uint8
        mask = np.array(morphology.remove_small_holes(cleaned,
min_size = 3000, connectivity = 1), dtype = np.uint8)
        self._grayImage = cv2.bitwise_and(self._grayImage,
self._grayImage, mask = mask)

#       cv2.imwrite('removeWhitePaper.jpg', self._grayImage)
        mask[mask == 0] = graylevel

        self._grayImage = self._grayImage + mask
        if self._bitNum == 8:
             self._grayImage_8 = self._grayImage.copy()
        else:
             self._grayImage_8 = self._grayImage.copy()/256




    def _removeNoise(self, threshold, omit = False):
        """Remove noisy points by thresholding.

        Arguments:
        threshold -- the threshold used for thresholding
        omit -- indicator of outputing descriptionally

        Returns:
        A clean (w|o noise) binary image"""
        if not omit:
             print('THRESHOLD = ' + str(threshold) + ', removing
noise...')
        lower = np.array([0])
        upper = np.array([threshold])
        binaryImg = cv2.inRange(self._grayImage, lower, upper)
        img_label = morphology.label(binaryImg)
        cleaned = morphology.remove_small_objects(img_label,
min_size = 3000, connectivity = 2)
        # Clean image is always 8-bit
        return np.array(255 *
morphology.remove_small_holes(cleaned, min_size = 3000,
connectivity = 1), dtype = np.uint8)


    def _findRoughContour(self, omit = False):
        """Find the rough contour based on _roughtContour

        Arguments:
```

```
        omit -- indicator of descriptional output

        Returns:
        The area (number of points enclosed) of _roughContour
"""
        if not omit:
             print('Looking for rough contour...')
        roughContours =  cv2.findContours(self._cleanImage
,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)[1]
        max = 0
        for contour in roughContours:
             if (max < cv2.contourArea(contour)) and ([[0, 0]]
not in contour):
                  max = cv2.contourArea(contour)
                  self._roughContour = contour
        if not omit:
             print('Rough contour has area ' +
str(cv2.contourArea(self._roughContour)))
        return cv2.contourArea(self._roughContour)


 def RoughContour(self, start_level = 50, step = 1, test_num =
10, omit = False):
        """Find optimal threshold parameter

        Arguments:
        start_level -- The start point of threshold testing
        step -- The step of threshold testing
        test_num -- number of testing that will be carried out
        omit -- indicator of descriptional output

        Returns:
        Best _roughContour """
        if not omit:
             print('Optimizing cv2.inRange THRESHOLD:')
        self._cleanImage = self._removeNoise(start_level - step,
omit = omit)
        prevArea = self._findRoughContour(omit = omit)
        deltaArea = np.ones(test_num) * prevArea
        for threshold in range(start_level, start_level +
test_num * step, step):
             self._cleanImage = self._removeNoise(threshold,
omit = omit)
             currArea = self._findRoughContour(omit = omit)
             if (prevArea - currArea) > deltaArea.min() * 5:
                  print('####################################')
                  print('## Thresholding aborted: Check! ##')
```

```
                print('## Result may not be optimized! ##')
                print('####################################')
                break
          deltaArea[int((threshold - start_level)/step)] =
prevArea - currArea
          prevArea = currArea
     THRESHOLD = (np.argmin(deltaArea) - 1) * step +
start_level

     if (THRESHOLD < self._threshold):
          THRESHOLD = self._threshold
     if not omit:
          print('Optimized THRESHOLD is ' + str(THRESHOLD))

     self._cleanImage = self._removeNoise(THRESHOLD, omit =
omit)
     self._findRoughContour(omit = omit)

     if self._save:
          outFileName =
self._imageName_plain+'-roughContour.'+ Out_File_Type
          bg = self._blankImage.copy()
          cv2.drawContours(bg, self._roughContour, -1, (255,
0, 0), 10)
          cv2.imwrite(self._outDir + '/' + outFileName, bg)
          if not omit:
               print('Done. Result is saved as ' +
outFileName)
     else:
          if not omit:
               print('Done. Result not saved.')


 def SimpleRoughContour(self):
     self._cleanImage = self._removeNoise(self._threshold,
omit = False)
     self._findRoughContour(omit = False)

     if self._save:
          outFileName =
self._imageName_plain+'-roughContour.'+ Out_File_Type
          bg = self._blankImage.copy()
          cv2.drawContours(bg, self._roughContour, -1, (255,
0, 0), 10)
          cv2.imwrite(self._outDir + '/' + outFileName, bg)
          print('Done. Result is saved as ' + outFileName)
     else:
```

```
            print('Done. Result not saved.')
#def RevisedSimpleRoughContour(self):


 def _contourP2P(self, beginPoint, endPoint, cnt):
      """ Connect two points on a contour

      Arugments:
      beginPoint -- the start point of the connection
      endPoint -- the end point of the conncetion
      cnt -- the contour on which the operation is performed

      Returns:
      A new contour from beginPoint to endPoint """
      beginPointIndex =
cnt.tolist().index([beginPoint.tolist()])
      endPointIndex = cnt.tolist().index([endPoint.tolist()])

      if (beginPointIndex < endPointIndex):
           return cnt[beginPointIndex: endPointIndex]
      else:
           return np.concatenate([cnt[beginPointIndex:],
cnt[:endPointIndex]])


 def _findSegCnt(self, cnt, omit = False):
      """Segment the contour into 6 parts

      Arguments:
      cnt -- the contour on which the operation is performed
      omit -- the indicator of descriptional output """
      if not omit:
           print('Looking for segamentation points...')

      mid_width = self._w / 2

      left_half_cnt = cnt[cnt[:,:,0] < mid_width]
      right_half_cnt = cnt[cnt[:,:,0] > mid_width]

      right_bot_pt =
right_half_cnt[right_half_cnt[:,1].argmax()]
      left_bot_pt = left_half_cnt[left_half_cnt[:,1].argmax()]

      o_toppest_pt = cnt[cnt[:,:, 1].argmin()][0]
      l1 = int((left_bot_pt[1] + right_bot_pt[1])/2)
      l2 = o_toppest_pt[1]
```

```
      Sl_cnt = np.array([pt for pt in cnt if (pt[0][1] <
(1.0/5)*l1+(4.0/5)*l2) and (pt[0][1] > (1.0/6)*l1+(5.0/6)*l2)])
      top_left_pt =
np.array(Sl_cnt[Sl_cnt[:,:,0].argmin()][0])
      top_right_pt =
np.array(Sl_cnt[Sl_cnt[:,:,0].argmax()][0])

      if not omit:
           print('Done.')

      if not omit:
           print('Connecting segamentation points...')
      self._cnt_seg_top = self._contourP2P(top_left_pt,
top_right_pt, cnt)
      self._cnt_seg_left = self._contourP2P(left_bot_pt,
top_left_pt, cnt)
      self._cnt_seg_right = self._contourP2P(top_right_pt,
right_bot_pt, cnt)
      self._cnt_seg_other = self._contourP2P(right_bot_pt,
left_bot_pt, cnt)
      self._cnt_seg_outer = self._contourP2P(left_bot_pt,
right_bot_pt, cnt)

      if not omit:
           print('seg inner contour...')
      i_toppest_pt =
self._cnt_seg_other[self._cnt_seg_other[:,:, 1].argmin()][0]
      l2 = i_toppest_pt[1]
      Sl_cnt_other = np.array([pt for pt in
self._cnt_seg_other if (pt[0][1] < (3.0/10)*l1+(7.0/10)*l2) and
(pt[0][1] > (3.0/10-1.0/30)*l1+(7.0/10 + 1.0/30)*l2)])
      top_left_pt_other =
np.array(Sl_cnt_other[Sl_cnt_other[:,:,0].argmin()][0])
      top_right_pt_other =
np.array(Sl_cnt_other[Sl_cnt_other[:,:,0].argmax()][0])

      self._cnt_seg_other_left =
self._contourP2P(top_left_pt_other, left_bot_pt, cnt)
      self._cnt_seg_other_right =
self._contourP2P(right_bot_pt, top_right_pt_other, cnt)
      self._cnt_seg_other_top =
self._contourP2P(top_right_pt_other, top_left_pt_other, cnt)

      if self._save:
           outFileName = self._imageName_plain+'-segContour.'+
Out_File_Type
           bg = self._blankImage.copy()
```

```
            cv2.drawContours(bg, self._cnt_seg_left, -1, (255,
0, 0), 15)
            cv2.drawContours(bg, self._cnt_seg_top, -1, (0,
255, 0), 15)
            cv2.drawContours(bg, self._cnt_seg_right, -1, (255,
0, 0), 15)
            cv2.drawContours(bg, self._cnt_seg_other_top, -1,
(0, 255, 0), 15)
            cv2.drawContours(bg, self._cnt_seg_other_left, -1,
(0, 0, 255), 15)
            cv2.drawContours(bg, self._cnt_seg_other_right, -1,
(0, 0, 255), 15)

            cv2.imwrite(self._outDir + '/' + outFileName, bg)
            if not omit:
                 print('Done. Result is saved as ' +
outFileName)
      else:
            if not omit:
                 print('Done. Result not saved.')


 def _firstReduceCntPt(self):
      """Reduce points by approxPolyDP"""
      print("First reducing contour points (approx poly
DP)...")
      self._approx_left = cv2.approxPolyDP(self._cnt_seg_left,
5, False)
      self._approx_right =
cv2.approxPolyDP(self._cnt_seg_right, 5, False)
      self._approx_top = cv2.approxPolyDP(self._cnt_seg_top,
5, False)
      self._approx_other =
cv2.approxPolyDP(self._cnt_seg_other, 5, False)
      self._approx_other_top =
cv2.approxPolyDP(self._cnt_seg_other_top, 5, False)
      self._approx_other_left =
cv2.approxPolyDP(self._cnt_seg_other_left, 5, False)
      self._approx_other_right =
cv2.approxPolyDP(self._cnt_seg_other_right, 5, False)
      for i in range(10):
            self._approx_left =
self._remove_bw_pts(self._approx_left, 0)
            self._approx_right =
self._remove_bw_pts(self._approx_right, 0)
            self._approx_top =
self._remove_bw_pts(self._approx_top, -0.3)
```

```
        if self._save:
            outFileName =
self._imageName_plain+'-fRedContour.'+ Out_File_Type
            bg = self._blankImage.copy()
            cv2.drawContours(bg, self._approx_left, -1, (255,
0, 255), 15)
            cv2.drawContours(bg, self._approx_top, -1, (0, 255,
0), 15)
            cv2.drawContours(bg, self._approx_right, -1, (255,
0, 255), 15)
            cv2.drawContours(bg, self._approx_other_left, -1,
(0, 255, 0), 15)
            cv2.drawContours(bg, self._approx_other_right, -1,
(0, 255, 0), 15)
            cv2.drawContours(bg, self._approx_other_top, -1,
(255, 0, 0), 15)
            cv2.imwrite(self._outDir + '/' + outFileName, bg)
#           plt.imshow(bg)
#           plt.show()
            print('Done. Result is saved as ' + outFileName)
        else:
            print('Done. Result not saved.')

    def _secondReduceCntPt_helper(self, approx_cnt,
cluster_dist):
        """Remove cluster on a contour

        If points are closed enough, then they are identify as a
        cluster. Only begin and end points are reserved

        Arugments:
        approx_cnt -- the contour on which the operation is
performed
        cluster_dist -- the minimum distance to identify a
cluster

        Returns:
        cnt -- the contour after cluster removal """
        idx_2 = 1
        indicator = np.ones(len(approx_cnt))

        while (idx_2 < len(approx_cnt)):
            while ((idx_2 < len(approx_cnt)) and
(np.sum((approx_cnt[idx_2][0]-approx_cnt[idx_2-1][0])**2) >
cluster_dist**2)):
                idx_2 = idx_2 + 1
```

```
            idx_1 = idx_2 - 1
            while ((idx_2 < len(approx_cnt)) and
(np.sum((approx_cnt[idx_2][0]-approx_cnt[idx_2-1][0])**2) <=
cluster_dist**2)):
                idx_2 = idx_2 + 1
            if (idx_2 - idx_1 > 2):
                indicator[idx_1 + 1: idx_2 -1] =
np.zeros(idx_2-idx_1 - 2)

      cnt = approx_cnt[indicator > 0]
      return cnt

 def _secondReduceCntPt(self):
      """Remove cluster on all segmentation"""
      print('Second reducing contour points (cluster
removal)...')
      self._approx_left =
self._secondReduceCntPt_helper(self._approx_left, 25)
      self._approx_right =
self._secondReduceCntPt_helper(self._approx_right, 25)
      self._approx_top =
self._secondReduceCntPt_helper(self._approx_top, 25)
#     self._approx_other =
self._secondReduceCntPt_helper(self._approx_other, 25)
      self._approx_other_top =
self._secondReduceCntPt_helper(self._approx_other_top, 25)
      self._approx_other_left =
self._secondReduceCntPt_helper(self._approx_other_left, 30)
      self._approx_other_right =
self._secondReduceCntPt_helper(self._approx_other_right, 30)
      if self._save:
            outFileName =
self._imageName_plain+'-sRedContour.'+ Out_File_Type
            bg = self._blankImage.copy()
            cv2.drawContours(bg, self._approx_left, -1, (255,
0, 255), 15)
            cv2.drawContours(bg, self._approx_top, -1, (0, 255,
0), 15)
            cv2.drawContours(bg, self._approx_right, -1, (255,
0, 255), 15)
            cv2.drawContours(bg, self._approx_other_left, -1,
(0, 255, 0), 15)
            cv2.drawContours(bg, self._approx_other_right, -1,
(0, 255, 0), 15)
            cv2.drawContours(bg, self._approx_other_top, -1,
(255, 0, 0), 15)
            cv2.imwrite(self._outDir + '/' + outFileName, bg)
```

```
            print('Done. Result is saved as ' + outFileName)
        else:
            print('Done. Result not saved.')

    def _findSuppPt(self, cnt, sticklength, axis, is_max, coeff):
        """Find support points on a contour

        Contour points are reduced according to whether they are
        inner than other candidates.

        Arguments:
        cnt -- the contour on which the operation is performed
        sticklength -- the radius of candidates
        axis -- 0 for x-axis, 1 for y-axis
        coeff -- 1 for upper side, -1 for other

        Returns:
        contour with only support points"""
        SuppPts_indicator = np.zeros(len(cnt))
        nextPtIndex = 0
        for index, pt in enumerate(cnt):
            if index < nextPtIndex:
                continue
            SuppPts_indicator[nextPtIndex] = 1
            EnuPts = [item for item in enumerate(cnt) if
((item[0] > index) and np.sum((item[1] - pt)**2) <=
sticklength**2)]
            if len(EnuPts) == 0:
                nextPtIndex = index + 1
            else:
                x_0 = pt[0][0]
                y_0 = pt[0][1]
                if (axis == 0) and (is_max == True):
                    while (True):
                        x_list = np.array([item[1][0][0] for
item in EnuPts])
                        y_list = np.array([item[1][0][1] for
item in EnuPts])
                        nextPtIndex, susp_pt =
EnuPts[np.argmax(x_list)]
                        y_1 = susp_pt[0][1]
                        x_1 = susp_pt[0][0]
                        susp_list = [item for item in EnuPts
if (item[1]!=susp_pt).all() and (item[1][0][1]-y_0 >
coeff*(y_0-y_1)/(x_0-x_1)*(item[1][0][0]-x_0))]
                        if len(susp_list) == 0:
                            break
```

```
                              else:
                                   EnuPts = susp_list
                         elif (axis == 0) and (is_max == False):
                              while (True):
                                   x_list = np.array([item[1][0][0] for
item in EnuPts])
                                   y_list = np.array([item[1][0][1] for
item in EnuPts])
                                   nextPtIndex, susp_pt =
EnuPts[np.argmin(x_list)]
                                   y_1 = susp_pt[0][1]
                                   x_1 = susp_pt[0][0]
                                   susp_list = [item for item in EnuPts
if (item[1]!=susp_pt).all() and (item[1][0][1]-y_0 >
coeff*(y_0-y_1)/(x_0-x_1)*(item[1][0][0]-x_0))]
                                   if len(susp_list) == 0:
                                        break
                                   else:
                                        EnuPts = susp_list
                         elif (axis == 1) and (is_max == True):
                              while (True):
                                   x_list = np.array([item[1][0][0] for
item in EnuPts])
                                   y_list = np.array([item[1][0][1] for
item in EnuPts])
                                   nextPtIndex, susp_pt =
EnuPts[np.argmax(y_list)]
                                   y_1 = susp_pt[0][1]
                                   x_1 = susp_pt[0][0]
                                   susp_list = [item for item in EnuPts
if (item[1]!=susp_pt).all() and (item[1][0][1]-y_0 >
coeff*(y_0-y_1)/(x_0-x_1)*(item[1][0][0]-x_0))]
                                   if len(susp_list) == 0:
                                        break
                                   else:
                                        EnuPts = susp_list
                         elif (axis == 1) and (is_max == False):

                              while (True):
                                   x_list = np.array([item[1][0][0] for
item in EnuPts])
                                   y_list = np.array([item[1][0][1] for
item in EnuPts])
                                   nextPtIndex, susp_pt =
EnuPts[np.argmin(y_list)]
                                   y_1 = susp_pt[0][1]
                                   x_1 = susp_pt[0][0]
```

```
                              susp_list = [item for item in EnuPts
if (item[1]!=susp_pt).all() and (item[1][0][1]-y_0 >
coeff*(y_0-y_1)/(x_0-x_1)*(item[1][0][0]-x_0))]
                              if len(susp_list) == 0:
                                   break
                              else:
                                   EnuPts = susp_list

      return cnt[SuppPts_indicator > 0]


 def _remove_bw_pts(self, contour, v):
      """remove backward points

      Arguments:
      contour: the contour on which this operation is
performed

      Returns:
      the contour after removing backward points"""
      indicator = np.ones(len(contour))
      for index in range(len(contour)-2):
           vec1 = contour[index + 2] - contour[index + 1]
           vec2 = contour[index + 1] - contour[index]
           if (np.sum(vec2 *
vec1)/(np.sqrt(np.sum(vec1*vec1)*np.sum(vec2*vec2)))  <= v) and
(np.sqrt(np.sum(vec1 * vec1)) < 100 or
np.sqrt(np.sum(vec2*vec2)) < 100):
                indicator[index + 1] = 0
                break
      return contour[indicator > 0]


 def _thirdReduceCntPt(self):
      """find support points on all segmentations"""

      print('Third reducing contour points (looking for
support pts)...')

      _approx_other_left_top =
self._approx_other_left[self._approx_other_left[:, :,
1].argmin()]
      _approx_other_right_top =
self._approx_other_right[self._approx_other_right[:,:,
1].argmin()]
```

```
        _approx_other_left_bot =
self._approx_other_left[self._approx_other_left[:,:,
1].argmax()]
        _approx_other_right_bot =
self._approx_other_right[self._approx_other_right[:,:,
1].argmax()]
#       print(_approx_other_left_top)
#       print(_approx_other_left_bot)
#       print(_approx_other_right_top)
#       print(_approx_other_right_bot)
        seg_left_h = int(_approx_other_left_top[0, 1] * 0.15 +
_approx_other_left_bot[0, 1] * 0.85)
        seg_right_h = int(_approx_other_right_top[0, 1] * 0.15 +
_approx_other_right_bot[0, 1] * 0.85)
#       print(seg_left_h)
#       print(seg_right_h)
        _approx_other_left_seg_1 =
self._approx_other_left[self._approx_other_left[:,:,1] <
seg_left_h]
        _approx_other_left_seg_2 =
self._approx_other_left[self._approx_other_left[:,:,1] >
seg_left_h]
        _approx_other_right_seg_1 =
self._approx_other_right[self._approx_other_right[:,:,1] <
seg_right_h]
        _approx_other_right_seg_2 =
self._approx_other_right[self._approx_other_right[:,:,1] >
seg_right_h]
        _approx_other_left_seg_1 = np.array([ np.array([pt]) for
pt in _approx_other_left_seg_1])
        _approx_other_left_seg_2 = np.array([ np.array([pt]) for
pt in _approx_other_left_seg_2])
        _approx_other_right_seg_1 = np.array([ np.array([pt])
for pt in _approx_other_right_seg_1])
        _approx_other_right_seg_2 = np.array([ np.array([pt])
for pt in _approx_other_right_seg_2])
#       print(_approx_other_left_seg_1)

#       print(_approx_other_right_seg_1)
        for i in range(100):
            self._approx_left =
self._remove_bw_pts(self._approx_left, 0)
            self._approx_right =
self._remove_bw_pts(self._approx_right, 0)
            self._approx_top =
self._remove_bw_pts(self._approx_top, -0.3)
```

```
            self._approx_other_top =
self._remove_bw_pts(self._approx_other_top, 0.7)
            _approx_other_left_seg_1 =
self._remove_bw_pts(_approx_other_left_seg_1, 0.7)
            _approx_other_right_seg_1 =
self._remove_bw_pts(_approx_other_right_seg_1, 0.7)
#     print(_approx_other_right_seg_1)

      self._approx_top = self._findSuppPt(self._approx_top,
50, 0, True, 1)
      self._approx_left = self._findSuppPt(self._approx_left,
100, 1, False, 1)
      self._approx_right =
self._findSuppPt(self._approx_right, 100, 1, True, -1)
      self._approx_other_top =
self._findSuppPt(self._approx_other_top, 50, 0, False, -1)
      _approx_other_left_seg_1 =
self._findSuppPt(_approx_other_left_seg_1, 50, 1, True, -1)
      _approx_other_right_seg_1 =
self._findSuppPt(_approx_other_right_seg_1, 50, 1, False, 1)

      for i in range(10):
            self._approx_left =
self._remove_bw_pts(self._approx_left, 0)
            self._approx_right =
self._remove_bw_pts(self._approx_right, 0)
            self._approx_top =
self._remove_bw_pts(self._approx_top, -0.3)
            self._approx_other_top =
self._remove_bw_pts(self._approx_other_top, 0.7)
            _approx_other_left_seg_1 =
self._remove_bw_pts(_approx_other_left_seg_1, 0.4)
            _approx_other_right_seg_1 =
self._remove_bw_pts(_approx_other_right_seg_1, 0.4)

      self._approx_other_left =
np.concatenate([_approx_other_left_seg_1,
_approx_other_left_seg_2])
      self._approx_other_right =
np.concatenate([_approx_other_right_seg_2,
_approx_other_right_seg_1])

      if self._save:
            outFileName =
self._imageName_plain+'-tRedContour.'+ Out_File_Type
            bg = self._blankImage.copy()
```

```
            cv2.drawContours(bg, self._approx_left, -1, (255,
0, 255), 15)
            cv2.drawContours(bg, self._approx_top, -1, (0, 255,
0), 15)
            cv2.drawContours(bg, self._approx_right, -1, (255,
0, 255), 15)
            cv2.drawContours(bg, self._approx_other_left, -1,
(0, 255, 0), 15)
            cv2.drawContours(bg, self._approx_other_right, -1,
(0, 255, 0), 15)
            cv2.drawContours(bg, self._approx_other_top, -1,
(255, 0, 0), 15)
            cv2.imwrite(self._outDir + '/' + outFileName, bg)
            print('Done. Result is saved as ' + outFileName)
      else:
            print('Done. Result not saved.')

 def detect_key_point(self, outer_contour, gray, mode='PCR'):
      """Detect key point from tire-2D section image and
return a list of key points."""
      # hyperparameter
      window_size = 25
      num_white_threshold = 100
      if mode == 'TBR':
            num_white_threshold = 300

      # Iterate along outer contour
      # Find key point candidates according to number of white
pixels in the window
      key_point = []
      index = []
      for i in range(len(outer_contour)):
            col_min = int(outer_contour[i, 0, 0] - window_size)
            col_max = int(outer_contour[i, 0, 0] + window_size)
            row_min = int(outer_contour[i, 0, 1] - window_size)
            row_max = int(outer_contour[i, 0, 1] + window_size)

            patch = gray[row_min:row_max, col_min:col_max]
            num_white = np.sum(patch > 144)

            if (num_white > num_white_threshold):
                  key_point.append(outer_contour[i, 0])
                  index.append(i)

      # Combine neighboring key point candidates to one key
point
      index = np.array(index)
```

```
        key_point = np.array(key_point)
        index_diff = index[1:] - index[:len(index)-1]
        division = np.argwhere(index_diff > 50)
        final_point = []
        prev = 0
        for i in range(len(division)):
             point = np.mean(key_point[prev:(division[i, 0] +
1)], axis = 0)
             final_point.append(point)
             prev = division[i, 0] + 1

        point = np.mean(key_point[prev:], axis = 0)
        final_point.append(point)

        final_point = np.expand_dims(np.array(final_point,
dtype=int), axis=1)
        return final_point

 def _detect_edge(self, src):
        """Run an edge detection algorithm on the original
image."""
        scale = 1
        delta = 0
        ddepth = cv2.CV_16S

        gray = cv2.GaussianBlur(src, (3, 3), 0)

        grad_x = cv2.Sobel(gray, ddepth, 1, 0, ksize=3,
scale=scale, delta=delta, borderType=cv2.BORDER_DEFAULT)
        grad_y = cv2.Sobel(gray, ddepth, 0, 1, ksize=3,
scale=scale, delta=delta, borderType=cv2.BORDER_DEFAULT)

        abs_grad_x = cv2.convertScaleAbs(grad_x)
        abs_grad_y = cv2.convertScaleAbs(grad_y)

        grad = cv2.addWeighted(abs_grad_x, 0.5, abs_grad_y, 0.5,
0)

        lower = 30
        upper = 255
        mask = cv2.inRange(grad, lower, upper)

        kernel =
cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
        opened_mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN,
kernel)
```

```
      masked_img = cv2.bitwise_and(grad, grad, mask=
opened_mask)

      return masked_img


 def _remove_noise(self, img, min_size):
      """Remove connected components smaller than min_size."""
      boolimg = np.array(img/255, dtype = bool)
      cleaned = morphology.remove_small_objects(boolimg,
min_size=min_size, connectivity=2)
      cleaned = np.array(cleaned, dtype=np.uint8)*255
      return cleaned


 def _find_tire_element(self, im, belt_num, filterRadius=10):
      """Find beltply."""
      filterSize = 2 * filterRadius + 1
      sigma = filterRadius*2
#     print(im.dtype)

      contours= cv2.findContours(im, mode=cv2.RETR_EXTERNAL,
method=cv2.CHAIN_APPROX_SIMPLE, offset=(0, 0))[1]
      bg = im.copy()
#     cv2.drawContours(bg, contours, -1, (255, 0, 255), 1)
#     cv2.imwrite("beltply.jpg", bg)
      # Find contours with large areas
      contours = sorted(contours, key = cv2.contourArea,
reverse=True)
      contour_list = []
      if belt_num < len(contours):
           for i in range(belt_num):
                contour_list.append(contours[i])
      else:
           contour_list = contours

      smoothContours = []
      for j in range(len(contour_list)):
           len1 = len(contour_list[j]) + 2 * filterRadius
           idx = len(contour_list[j]) - filterRadius
           x = np.zeros(len1)
           y = np.zeros(len1)
           for i in range(len1):
                x[i] = float(contour_list[j][(idx + i) %
len(contour_list[j])][0, 0])
                y[i] = float(contour_list[j][(idx + i) %
len(contour_list[j])][0, 1])
```

```
            # Sort points on the contour according to x
coordinates
            arr_idx = np.argsort(x)
            x = x[arr_idx]
            y = y[arr_idx]

            # Filter y coordinates using a Gaussian filter
            xFilt = np.expand_dims(x, axis=1)
            yFilt = cv2.GaussianBlur(y, ksize=(filterSize,
filterSize), sigmaX=sigma, sigmaY=sigma)

            smooth = np.zeros((len(contour_list[j]), 2))
            for i in range(filterRadius, len(contour_list[j]) +
filterRadius):
                smooth[i - filterRadius, 0] = xFilt[i, 0]
                smooth[i - filterRadius, 1] = yFilt[i, 0]
            smoothContours.append(smooth.astype(int))

      return smoothContours

 def _create_kernel(self, h, w, left = True):
      kernel = np.zeros((h, w), np.uint8)
      for x in range(w):
            if left:
                y =  h - 1 - int(h/w * x)
            else:
                y = int(h/w * x)
            if y < 0:
                y = 0
            kernel[y, x] = 1
#     print(kernel)
      return kernel


 def new_detect_beltply(self, src, outer_contour,
inner_contour, belt_num = 2):
      """Detect beltply from tire-2D section image and return
a list of lists containing points on beltplies."""
      # hyperparameter
      min_size = 500
      filterRadius = 10

      #
      # Extract approximate beltply area
      upper_b = np.min(outer_contour[:, 0, 1])
      lower_b = np.min(inner_contour[:, 0, 1])
```

```
        left_b = np.min(inner_contour[:, 0, 0])
        right_b = np.max(inner_contour[:, 0, 0])

        belt = src[upper_b:(3*lower_b-upper_b), left_b:right_b]

        # Edge detection
        gray = self._detect_edge(belt)
        _, gray = cv2.threshold(gray, 255/4, 255,
type=cv2.THRESH_BINARY)

        width = int((right_b - left_b) / num_of_seg)
        img_seg = []
        for num in range(num_of_seg-1):
             img_seg.append(gray[:, num*width: (num+1)*width])
        img_seg.append(gray[:, (num_of_seg - 1)*width:])


        for num in range(num_of_seg):
             l1 = 60
             l2 = 65
             est_list = np.array([pt for pt in inner_contour if
((num+1)*width >= (pt[0][0]-left_b)) and ((pt[0][0]-left_b) >=
num*width)])
             left_pt =
np.array(est_list[est_list[:,:,0].argmin()][0])
             right_pt =
np.array(est_list[est_list[:,:,0].argmax()][0])
#                print(left_pt, right_pt)
             if (left_pt == right_pt).all():
                  ratio_i = 0
             else:
                  ratio_i = (left_pt[1]-
right_pt[1])/(left_pt[0] - right_pt[0])
             est_list = np.array([pt for pt in outer_contour if
((num+1)*width >= (pt[0][0]-left_b)) and ((pt[0][0]-left_b) >=
num*width)])
             left_pt =
np.array(est_list[est_list[:,:,0].argmin()][0])
             right_pt =
np.array(est_list[est_list[:,:,0].argmax()][0])
             if (left_pt == right_pt).all():
                  ratio_o = 0
             else:
                  ratio_o = (left_pt[1]-
right_pt[1])/(left_pt[0] - right_pt[0])
#           print(str(num) + ' ' + str(ratio_o) + ' ' +
str(ratio_i))
```

```
            if (self._isPCR):
                 if (ratio_i > -0.3 and ratio_i < 0.3):
                      ratio = ratio_i
                 else:
                      if (ratio_i > -0.6 and ratio_i < 0.6):
                           ratio = ratio_i * 0.9
                      else:
                           ratio = 0.85* (ratio_i + ratio_o)/2
            else:
                 if (num <= 7*num_of_seg/20 or num >=
13*num_of_seg/20) and (ratio_o >= -0.15 and ratio_o <= 0.15):
                      ratio = ratio_o
                 else:
                      if (ratio_i > -0.8 and ratio_i < 0.8):
                           ratio = ratio_i * 0.6
                      else:
                           ratio = ratio_i * 0.35
#           print(ratio)
            delta_x_1 =
max(int(np.sqrt(l1*l1/(1+ratio*ratio))), 1)
            delta_y_1 = max(int(delta_x_1 * abs(ratio)), 1)
            delta_x_2 =
max(int(np.sqrt(l2*l2/(1+ratio*ratio))), 1)
            delta_y_2 = max(int(delta_x_2 * abs(ratio)), 1)
            delta_x_3 = max(int(4*delta_x_2/5), 1)
            delta_y_3 = max(int(4*delta_y_2/5), 1)
            delta_x_4 = max(int(1*delta_x_2/2), 1)
            delta_y_4 = max(int(1*delta_y_2/2), 1)
#           print(delta_x_1, delta_y_1)
            if ratio < 0:
                 kernel_1 = self._create_kernel(delta_y_1,
delta_x_1, left = True)
                 kernel_2 = self._create_kernel(delta_y_2,
delta_x_2, left = True)
                 kernel_3 = self._create_kernel(delta_y_3,
delta_x_3, left = True)
                 kernel_4 = self._create_kernel(delta_y_4,
delta_x_4, left = True)
            else:
                 kernel_1 = self._create_kernel(delta_y_1,
delta_x_1, left = False)
                 kernel_2 = self._create_kernel(delta_y_2,
delta_x_2, left = False)
                 kernel_3 = self._create_kernel(delta_y_3,
delta_x_3, left = False)
                 kernel_4 = self._create_kernel(delta_y_4,
delta_x_4, left = False)
```

```
            ans = cv2.dilate(img_seg[num], kernel_1,
iterations= 1)
            if self._isPCR:
                 img_seg[num] = cv2.erode(ans, kernel_2,
iterations = 1)
                 Ar = (lower_b - upper_b)/(right_b - left_b)
                 if (Ar < 1/13):
                      if (Ar > 1/15):
#                          print('kernel_3')
                          img_seg[num] =
cv2.erode(img_seg[num], kernel_3, iterations = 1)
                      else:
#                          print('kernel_4')
                          img_seg[num] =
cv2.erode(img_seg[num], kernel_4, iterations = 1)
            else:
                 img_seg[num] = cv2.erode(ans, kernel_2,
iterations = 1)

      img_seg = tuple(img_seg)
      img = np.concatenate(img_seg, axis = 1)

#     print(img.shape)
      # Threshold
#     _, img = cv2.threshold(img, 255/4, 255,
type=cv2.THRESH_BINARY)

      # Remove noise
      if self._save:
           outFileName =
self._imageName_plain+'-plainConncet.'+ Out_File_Type
           cv2.imwrite(self._outDir + '/' + outFileName, img)

      cleaned = self._remove_noise(img, min_size=min_size)

      cleaned = self._remove_noise(255-cleaned,
min_size=min_size)

      kernel =
cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(4,4))
      if (self._isPCR == False):
           cleaned = cv2.dilate(cleaned, kernel, iterations =
4)
      else:
           cleaned = cv2.dilate(cleaned, kernel, iterations =
1)
```

```
        kernel = np.ones((2,1))
        cleaned = cv2.erode(cleaned, kernel, iterations = 1)
        kernel = np.ones((1, 5))
        cleaned = cv2.dilate(cleaned, kernel, iterations = 1)
        kernel = np.ones((1, 3))
        cleaned = cv2.dilate(cleaned, kernel, iterations = 1)
        if self._save:
             outFileName =
self._imageName_plain+'-dilateErode.'+ Out_File_Type
             cv2.imwrite(self._outDir + '/' + outFileName,
cleaned)

        result = np.zeros(src.shape, dtype=np.uint8)
        result[upper_b:(3*lower_b-upper_b), left_b:right_b] =
255-cleaned
#       cv2.imwrite('cleaned.jpg',cleaned)
#       cv2.imwrite('test.jpg', result)
        # Find contour
        smoothContours = self._find_tire_element(result,
belt_num=belt_num, filterRadius=filterRadius)

        return smoothContours


  def fullTest(self, isPCR = True, simpleTest = False):
        """Perform a full test on the raw image

        Arguments:
        isPCR -- indicator of a PCR tire

        Returns:
        a dxf file with all vectors required"""

        if not simpleTest:
             step = int((self._threshold_upb -
self._threshold_lwb)/max_test_num)
             if step < 1:
                  step = 1
             self.RoughContour(self._threshold_lwb, step,
self._threshold_upb)

        else:
             self.SimpleRoughContour()

        self._findSegCnt(self._roughContour)

        if isPCR:
```

```
            final_points =
self.detect_key_point(self._cnt_seg_outer, self._grayImage_8,
mode='PCR')
      else:
            final_points =
self.detect_key_point(self._cnt_seg_outer, self._grayImage_8,
mode='TBR')

      self._firstReduceCntPt()
      self._secondReduceCntPt()
      self._thirdReduceCntPt()

      beltply_pt =
self.new_detect_beltply(self._grayImage_8.copy(),
self._cnt_seg_outer, self._cnt_seg_other_top, belt_num =
self._belt_num)

      final_cnt = np.concatenate([self._approx_left,
self._approx_top, self._approx_right, self._approx_other_right,
self._approx_other_top, self._approx_other_left])

      outFileName = self._imageName_plain+'-final.'+
Out_File_Type
      bg = self._blankImage.copy()

#     final_cnt = self._remove_bw_pts(final_cnt)
#     cv2.drawContours(bg, [beltply_pt], -1, (255, 255, 255),
5)
      cv2.drawContours(bg, final_points, -1, (255, 0, 0), 50)
      cv2.drawContours(bg, [final_cnt], -1, (255, 0, 255), 3)

      for index in range(self._belt_num):
            cv2.drawContours(bg, [beltply_pt[index]], -1,
(255,0,0), 5)

      cv2.imwrite(self._outDir + '/' + outFileName, bg)
      print('Final result is saved as ' + outFileName)

      self._final_cnt_seg_outer =
np.concatenate([self._approx_left, self._approx_top,
self._approx_right])
      self._final_cnt_seg_inner =
np.concatenate([self._approx_other_right,
self._approx_other_top, self._approx_other_left])
      self._final_cnt = final_cnt
      self._key_points = final_points
      self._beltply_pt = beltply_pt
```

```
 def beltplyTest(self, simpleTest = True):
      if not simpleTest:
           step = int((self._threshold_upb -
self._threshold_lwb)/max_test_num)
           if step < 1:
                step = 1
           self.RoughContour(self._threshold_lwb, step,
self._threshold_upb)
      else:
           self.SimpleRoughContour()

      self._findSegCnt(self._roughContour)
      bg = self._blankImage.copy()
      beltply_pt =
self.new_detect_beltply(self._grayImage_8.copy(),
self._cnt_seg_outer, self._cnt_seg_other_top, belt_num =
self._belt_num)
#     print(len(beltply_pt))
      for index in range(self._belt_num):
           cv2.drawContours(bg, [beltply_pt[index]], -1,
(255,0,0), 5)

      cv2.imwrite(self._outDir + '/' + self._imageName_plain +
'beltply.jpg', bg)


 def cntTest(self, simpleTest = True):
      """A partial test on the contour vector"""

      if not simpleTest:
           step = int((self._threshold_upb -
self._threshold_lwb)/max_test_num)
           if step < 1:
                step = 1
           self.RoughContour(self._threshold_lwb, step,
self._threshold_upb)
      else:
           self.SimpleRoughContour()

      self._findSegCnt(self._roughContour)

      self._firstReduceCntPt()
      self._secondReduceCntPt()
      self._thirdReduceCntPt()
```

```
      final_cnt = np.concatenate([self._approx_left,
self._approx_top, self._approx_right, self._approx_other_right,
self._approx_other_top, self._approx_other_left])

      outFileName = self._imageName_plain+'-final.'+
Out_File_Type
      bg = self._blankImage.copy()
      final_cnt = self._remove_bw_pts(final_cnt)
      self._final_cnt = final_cnt
      cv2.drawContours(bg, [final_cnt], -1, (255, 0, 255), 3)
      cv2.imwrite(self._outDir + '/' + outFileName, bg)
      print('Final result is saved as ' + outFileName)

 def vectorize(self):
      """Vectorize all elements to give a dxf"""

      print('Vectorizing...')
      belt_num = self._belt_num
      final_cnt_outer = self._final_cnt_seg_outer
      o_contour_pt_list = [((25.40*item[0, 0]/self.dpi[0]),
(25.40*(self._h-item[0, 1])/self.dpi[1])) for item in
final_cnt_outer]
      final_cnt_inner = self._final_cnt_seg_inner
      i_contour_pt_list = [((25.40*item[0, 0]/self.dpi[0]),
(25.40*(self._h-item[0, 1])/self.dpi[1])) for item in
final_cnt_inner]
      outFileName = self._imageName_plain+'-final.dxf'
      dwg = dxf.drawing(self._outDir +'/'+ outFileName)


      for index in range(belt_num):
           beltply_pt_list =
[((25.40*item[0]/self.dpi[0]),(25.40*(self._h-item[1])/self.dpi
[1])) for item in self._beltply_pt[index]]
           dwg.add(dxf.polyline(beltply_pt_list, color = 20))


      key_pt = self._key_points
      key_pt_list =
[(int(25.40*item[0]/self.dpi[0]),int(25.40*(self._h-item[1])/se
lf.dpi[1])) for item, in key_pt]


      for pt in key_pt_list:
           dwg.add(dxf.circle(radius = 1, center = pt, color =
255))
```

```
      o_final_cnt_polyline = dxf.polyline(o_contour_pt_list,
color = 100)
      dwg.add(o_final_cnt_polyline)
      i_final_cnt_polyline = dxf.polyline(i_contour_pt_list,
color = 200)
      dwg.add(i_final_cnt_polyline)


      dwg.save()
      print('Done.')


if __name__ == "__main__":
 filename = "PCR-1-1-2-15.jpg"
 tags = filename.split('-')
 if tags[0]=='PCR':
      isPCR = True
 else:
      isPCR = False

 tire = TireCrossSection(filename, isPCR = isPCR ,save = True,
belt_num = int(tags[3]), outDir = '.', datadir = '.')
 tire.fullTest(simpleTest = True)
#tire.vectorize()
#tire.beltplyTest(simpleTest = True)
```