UNIVERSITY OF CALIFORNIA
SANTA CRUZ

PANORAMIC IMAGING FOR TEXT SPOTTING

A thesis submitted in partial satisfaction of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Peng Ren

March 2018

The Thesis of Peng Ren
is approved:

Professor Roberto Manduchi, Chair

Professor Ricardo Sanfelice

Professor Chen Qian

Tyrus Miller
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

Abstract

Panoramic imaging for text spotting

by

Peng Ren

Visually impaired people are a group of people who have some degree of vision loss. In order to help these people getting text information of surrounding space, I build an app that can detect text for visually impaired people. At first, the app takes images automatically, and then the google OCR API is used to find text spots and perform word recognition. After that, my app will notify the user of the text contents and positions. Since there is overlapping between adjacent images, the app might detect one text twice if the text exists in both photos. To solve this problem, homography matrices among pictures are calculated using feature detection and matching techniques. Besides, the panorama is used to illustrate my results. As a result, the app can detect text efficiently and reduce various difficulties that visually impaired people might meet in walking, navigating and so on.

Acknowledgments

First and foremost, I would like to show my great gratitude to my current thesis advisor professor Roberto Manduchi for providing me professional guidance on every step of this research. I'm also pretty thankful to professor Ricardo Sanfelice and professor Chen Qian for spending their precious time reviewing my thesis. After that, I'm grateful for all the teachers who helped me build a solid foundation for the academic competence. Last but not least, I would like to specifically thank Siyang Qin and Leo who gave me valuable advice on this project.

Chapter 1

Introduction

1.1 Background

As sighted people, we get an indispensable part of the information from text around us. For example, traffic signs along the street will instruct us to drive cars properly, billboards will bring us information about various kinds of stores and commodities. Besides, when we enter a building, the room number and text beside a door will give us information about the room. In addition to that, the ”EXIT” sign can let us know where the emergency exit is, which would be helpful when emergencies occur. However, for visually impaired people, things can be entirely different. The visually impaired people have vision loss in some degrees, and they cannot get information through their eyes efficiently. Thus, people with such disability might have difficulties in various aspects such as navigation, recognition and so on.

Since the text around sighted people provide an essential part of the information they can get using their eyes, it is a reasonable claim that a considerable part of difficulties caused by

visual impairment can be reduced if the visually impaired people have an approach to obtain text positions and text contents around them. To reduce such kind of difficulties and make visually impaired people's life more convenient, I built an Android app that can help visually impaired individuals obtain text information around them.

To use this app, the visually impaired user needs to stand in one place and open the system. Instructions will be given to the user so that they can hold the phone horizontally and rotate it slowly from left to right. Then the app will take pictures at specific orientations automatically. Every time an image is obtained, the app will use google OCR API to find text regions in each image and perform word recognition. After that, the app will inform the user of the results using voice instructions. In this way, visually impaired people can obtain the text information around them and have a better understanding of the real-world situation.

1.2 Descriptions of the thesis

In next section, I discuss the merits and drawbacks of several works done by other people related to this field. In Chapter 3, I describe the concepts of camera calibration, image distortion correction, and cylindrical projection. Notice that these are well-known concepts and they are only used to give context for this thesis. Chapter 4 is dedicated to user interface design, the method I used to capture high-quality images, text detection process, and user notification means. Chapter 5 describes the well-known notions of feature matching and image transformation. Then, it talks about my solution to the repeated text notification problem caused by image overlapping. This Chapter also focuses on the implementation of my ideas. Besides, it

introduces three ways to optimize the code performance. After that, I also explain the reason why I do not use OpenCV stitching API. Finally, I draw a conclusion and talk about the future works in chapter 6.

Chapter 2

Related Work

Several recent papers have used various techniques to navigate visually impaired people. Wang *et al.* [1] built a wearable vision-based feedback system using a haptic device, a camera, and an embedded computer. It can help visually impaired people walk safely, detect obstacles and specific objects. Though this system can navigate the visually impaired user by providing feedback on objects, it might be a little bit clumsy to wear such an equipment, and only vibrations are used to instruct the user. Oh *et al.* [2] proposed an indoor navigation aid system for visually impaired people. In this system, a computer is used to plan the optimal indoor paths for the visually impaired user in real-time. The system also contains an app that can generate voice instructions and the visually impaired user can operate this app by swiping or tapping the phone. Though this system can help the user arrive in unfamiliar places, a database used for path planning must be created by instructors first. Besides, the network is essential for the communication between computer and smartphone. Lin *et al.* [3] built a smartphone-based guiding system. This system captures images using a mobile phone, then process pictures in

a backend server. As a result, it can recognize obstacles and use the result to assist visually impaired people. Niu *et al.* [4] presented a wearable assistive technology that can help visually impaired people locate door handles and instruct them to open doors. Yi *et al.* [5] described a system that can help visually impaired people read printed text on hand-held objects. For this system, the user first needs to shake an object, then the system can extract the moving object region automatically and process the word recognition. After that, the visually impaired user can hear the text contents.

The Seeing AI [6] is an app that designed for visually impaired people. It can read short text in front of the phone camera, which is somewhat similar to the app. However, it's not convenient for the visually impaired people to get all the text around them utilizing this app and it's easy for them to miss text. Besides, the user may hear the same text multiple times, which can be quite confusing. In addition to that, it doesn't work well when the words are far away from the visually impaired user.

Here are the contributions of this thesis:

1) I developed an app that can read the text found by google OCR API out for visually impaired people in the realistic environment .

2) I specifically designed an user interface that is suitable for visually impaired people.

3) I presented a technique that can detect text in large size images utilizing google OCR API. Using this technique, my app can detect small words and obtain good quality text.

4) I proposed a solution to the repeated text notification problem caused by image overlapping.

5) I figured out several ways that can speed up the image stitching process on Android device.

Chapter 3

Image Preprocessing

By calibrating a camera, one can know its intrinsic matrix and the distortion coefficients. Using the distortion coefficients, the image distortion can be corrected, and the intrinsic parameters are essential for the following tasks. Besides, the cylindrical projection can also be useful in many aspects of this project. Thus, I would like to introduce the related well-known concepts first to give context for my work.

3.1 Camera calibration

Even if two cameras share the same model, the differences between them still exist. In other words, each camera has its own parameters and distortion. To get the unique parameters of a camera and correct the camera distortion, one always need to do the camera calibration. This process ensures a good result.

To calibrate a camera, I would like to use the Zhang's multiplane calibration method [7]: First, a 10*7 chessboard is prepared. Then, images from multiple views of the chessboard

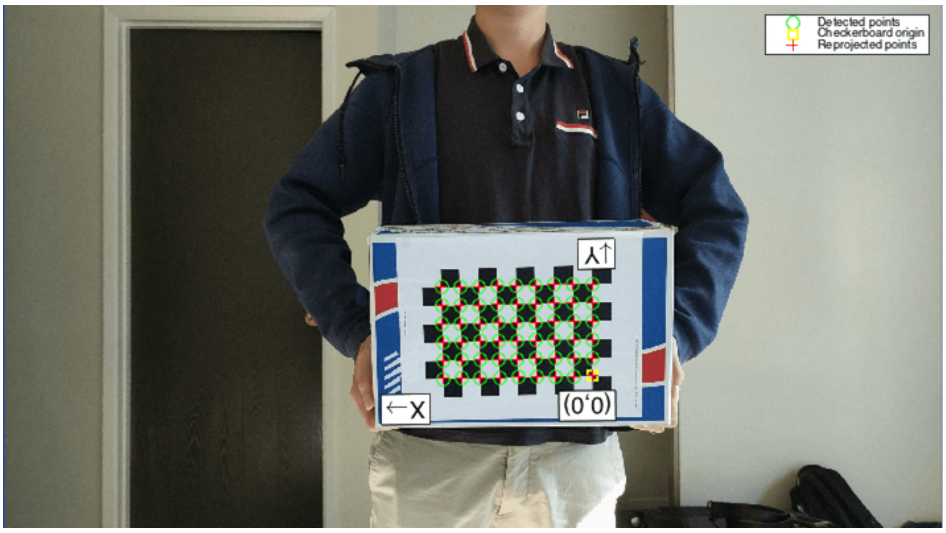


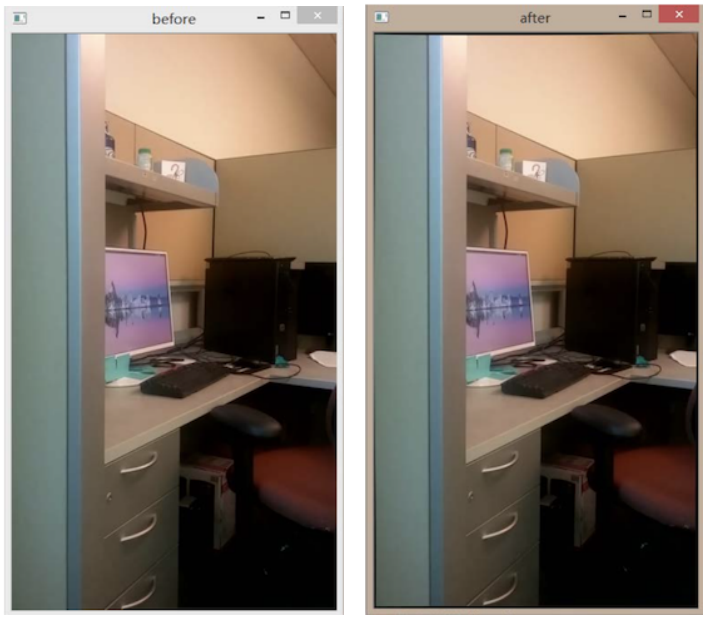
Figure 3.1: A snapshot of the camera calibration process

are taken. The number of pictures shouldn't be less than 10, and it shouldn't be larger than 20, I obtain 19 images in my case. After that, I use the Single Camera calibration App from MATLAB to process these images and get the camera calibration parameters. Fig. 3.1 shows a snapshot of this process.

In my case, I would like to get three kinds of parameters: the intrinsic matrix, the radial distortion, and the tangential distortion. The intrinsic matrix contains four parameters:

$$\begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

Both of f_x and f_y are camera focal lengths. I obtain the camera focal length by taking an average of them. (u_0, v_0) represents the principal point of the image. In other words, it is the center of an image. I will talk about the distortion parameters in next section.



(a) (b)

Figure 3.2: An image distortion correction example. The **left** image shows the original image, and the **right** image shows the result after distortion correction process.

3.2 Image distortion correction

Two kinds of distortion can exist in an image: the radial distortion and the tangential distortion. There are also two kinds of distortion coefficients I can get from the camera calibration process: The radial distortion coefficients and the tangential distortion coefficients. The radial distortion coefficients are used to correct the radial distortion. The tangential distortion coefficients are used to correct the tangential distortion. Fig. 3.2 gives an example of this process.

3.3 Cylindrical projection

I use cylindrical projection technique frequently in this project. For example, I need to project text bounding-boxes to the cylindrical coordinate system before informing the user of text positions. It is also pretty crucial to perform the cylindrical projection before generating a panorama from multiple images. Thus, I would like to introduce such concept first. Let the center of an image be (x_c, y_c). Let the focus length of the camera be f. To project the image to cylindrical coordinate, I project each pixel in the original image (x, y) to a new point (x', y') using the following formulas. Fig. 3.3 shows an example of cylindrical projection. First, every pixel in Fig. 3.3 (a) is projected to a cylinder whose radius is f using the formulas below. Then, the cylinder is unrolled to a flat plane. After that, the Fig. 3.3 (b) is obtained.

$$x' = f * tan^{-1}\frac{x}{f} + x_c \tag{3.2}$$

$$y' = f * \frac{y}{\sqrt{x^2 + f^2}} + y_c \tag{3.3}$$

(a) (b)

Figure 3.3: A Cylindrical projection example. The **left** image shows the original image, and the **right** image shows the result after cylindrical projection process.

Chapter 4

App Structure

In this Chapter, I first introduce the user interface I designed for visually impaired people. Then, I describe the text detection process. After that, I present a technique that can perform text detection in high-resolution images utilizing google OCR API. Besides, I talk about the two user notification schemes I come up with.

4.1 User interface design

When the visually impaired user use the camera to take pictures, there are two options: he or she can hold the phone either horizontally or vertically. Since usually there is no text on the ceiling or floor, the top and bottom parts of the picture are wasted if the user takes vertical images. To avoid this wasting and detect text efficiently, the pictures should be taken horizontally.

As mentioned above, I would like to make sure the app can take images automatically when users rotate their phones. But here comes a question: when should the user stop and take

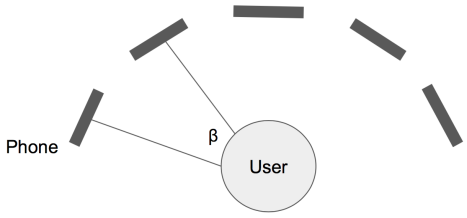


Figure 4.1: Image caption

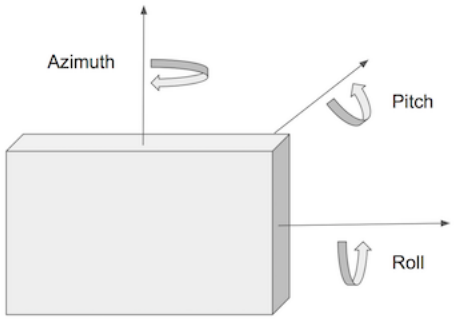


Figure 4.2: The mobile phone orientation. The orientation of a phone can be determined by three different angles: azimuth, pitch, and roll.

pictures? As shown in Fig. 4.1, the phone will take images automatically every β degrees when the user rotates it slowly. A reasonable value of β should be selected. But before that, let's talk about a way to obtain the phone orientation. Three different angles are used to determine the orientation of a phone: azimuth, pitch, and roll. Fig. 4.2 illustrates such a concept.

The azimuth value can represent the rotation of a phone if the phone is hold horizontally. At first, MAGNETIC_FIELD sensor is used together with ACCELEROMETER sensor to get azimuth value. However, the result is not stable enough due to the instability of the MAGNETIC_FIELD sensor. Then, I tried the TYPE_GAME_ROTATION -_VECTOR sensor. Since it does not use the geomagnetic field, it's pretty stable. The only defect of it is that its Y axis

(a) (b)

Figure 4.3: Two adjacent images without overlapping. There is a gap between **left** image and **right** image. Thus, the text "Computer" is separated.

doesn't point north. But I don't care about such thing. Thus, it is a perfect choice.

After obtaining the phone orientation, let's discuss the value of β. First, the β value should not be too small. Otherwise, the app will stop and take images too many times. The value of β also cannot be too large. Some text will be separated in such condition, and it's hard to recover the original text. Fig. 4.3 shows an example.

I did several experiments to figure out a reasonable value of β. As a result, half of the horizontal field of view (FOV) is a good choice. The horizontal FOV is an area that can be captured by a camera in the horizontal direction, which is usually expressed as an angle of view. To get the FOV of a camera, I capture one image and project it to the cylindrical coordinate system. Then, as shown in Fig. 4.4. The width of the image in cylindrical coordinate system is L, and the radius of the cylinder is the camera focus length f. O represents the vertical axis of the cylinder. Then I can calculate the camera FOV using the following formula:

$$FOV = \frac{L}{f} \tag{4.1}$$

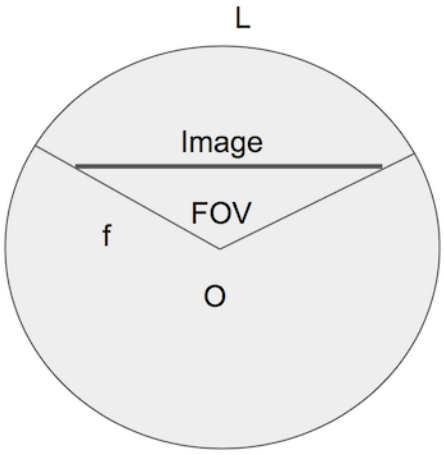


Figure 4.4: FOV calculation

The FOV is 65 degrees in my case. Thus, the value of β is set to 33 degrees. As a result, the user can stop rotating the mobile phone and let my app capture an image automatically every 33 degrees.

4.2 User interface optimization

Since visually impaired people cannot get information using their eyes efficiently, they may meet unexpected difficulties when they use an app designed for sighted people. For example, they cannot see text instructions shown on the screen. Thus, if one want to design an app that is suitable for visually impaired people, the user interface should be designed carefully. Here are several ways I used to make the app easy to use.

First and foremost, the app should not use text or graphics to inform the visually impaired user for that they have difficulty in reading. Preferably, I use short music and speech

instructions instead.

Secondly, the on-screen button should not be used in such an app since it is pretty hard for visually impaired people to get the button position. Instead, I use volume buttons. It is much easier for the visually impaired people to figure out the location of a physical button than an on-screen button.

Thirdly, I would like to make sure that the user can use this app to take high-quality images. The app may fail to find text on an image if its image quality is low. One of the most common reasons that can cause this is motion blur. The photo will be blurry if the user move rapidly during the exposure process.

To avoid the motion blur, people should not move when they capture scenes. Sighted people can hold their phones as still as they can to obtain clear pictures. However, visually impaired people usually do not have such concept. They also have no idea about when they should stop rotating their phone for a snapshot. In addition to that, they do not know how long they should hold the camera still to get a photograph clear enough. Thus, it is necessary for the app to give instructions about these things.

My app plays a camera shutter sound when the camera begins taking a picture. The visually impaired user will hear ”Please stop for a snapshot”. Then he or she will know it’s time to stop rotating. The app will play ”Please move the phone slowly to the right” after the whole process is finished. Then the user will know it’s time to move the phone. I also use the gyroscope sensor to detect whether the phone is stable or not. The phone can only take images when it is stable. Using these methods, the app can obtain high-quality images.

In addition to that, an image that is out of focus can also be blurry. This problem

can be avoided by choosing the camera focusing mode correctly. There are two choices: One can either select the fixed focus mode and keep the focal length to infinity or choose the FOCUS_MODE_CONTINUOUS_PICTURE mode, in which the focal length changes automatically and frequently. If the focal length is set to infinity, the near subject will be somewhat blurry, which will lead to a poor result. After several experiments, I figured out that the FOCUS_MODE_CONTINUOUS_PICTURE mode is much better than the first one in my case. When the image caption function is called, the focal length of the camera will be fixed after it is changed automatically. Then an image is taken. Next, the camera focal length changes automatically again.

Fourthly, I notice that the visually impaired people usually have difficulty in holding the phone horizontally. I would like to make sure the phone is held horizontally and the short edge of a mobile phone is perpendicular to the ground. But the visually impaired people also do not have such concept. Thus, the accelerometer sensor is used to monitor the placement of the phone. The app will give voice instructions such as ”tilt the device up”, ”tilt the device down”, ”rotate left”, ”rotate right” to assist the user.

Fifthly, the visually impaired user can pause the app whenever he or she touches the screen. The app will say ”paused”. If the user wants to begin using the app, he or she can touch the screen again. Then the user will hear ”resume” and my app will restart the whole process. I think this function can improve the user experience.

Here is how my app works: When the system begins to operate, the visually impaired user first pushes one of the volume buttons when he or she hears the voice instruction: ”Ready to scan”. Then the app begins taking the picture automatically and process the images. After

that, the user can get informed of the text detection result, and he or she will hear: ”Please move the phone slowly to the right.” Next, the user will hear ”Stop for snapshot” when he or she rotates the phone for 33 degrees. After that, the user can stop and let the app capture images automatically again. The whole process will come to an end after the user has rotated the phone for 360 degrees. Notice that several voice instructions are given to make sure the user can hold the mobile phone horizontally.

4.3 Text detection

4.3.1 OCR processing

After obtaining high-quality images, I am going to use the Optical Character Recognition (OCR) technique to do text detection and word recognition in each image. The OCR is a conversion technology that can convert text images to text strings. Currently, I am using the google OCR API. It can not only perform the word recognition but also find text bounding-boxes in an image. In addition, it is well-documented and free. To get a good OCR result, the resolution of the input image should not be too low. Otherwise, the app may have difficulty in figuring out the small text on the image for that the text images will be blurry if the resolution is low. Thus, I would like to increase the image size as large as I can. In this app, the image size is 3840 x 2160. At first, the API does not work very well when I just input the original image. According to the google document, my image size is much larger than the recommended size (1024 x 768) [8].

I came up with a solution to this problem: I split the image into ten smaller pictures

(a) (b)

Figure 4.5: An image separation example. The **left** image shows the original image, and the **right** image shows the image separation result. The **left** image is splitted into ten smaller images.

(1440 x1440). Fig. 4.5 illustrate this process. Notice that there must be overlappings among these images. Otherwise, some text will be separated. Sometimes it is hard to recover them. Then, I use google OCR API to process these smaller images. After all the text on these images are obtained, every text bounding-box are drawn on the original image and a bounding-boxes aggregation process is performed: If bounding-boxes are overlapped, the external rectangle of them are found and a new bounding box is created. The longest text in these bounding-boxes is also associated with the new bounding box.

Now a bunch of contiguous sets that consist of words on the same vertical axis can be obtained [9]. Each contiguous set is consists of a bounding-box and its text content. Since the number of bounding-boxes sometimes can be large, I would like to cluster these bounding-boxes together at first. Here is my simple algorithm: First, I project each text bounding box to x coordinate. If two bounding-boxes are overlapping in x coordinate, I will add them to the same group. Otherwise, they belong to different groups. After this process, I can obtain several bounding box clusters. Then, for each group, I find the text contents associated with the

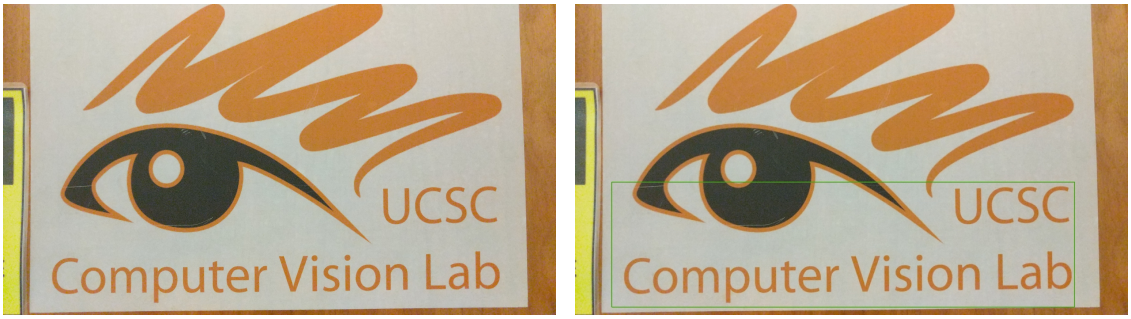
(a) (b)

Figure 4.6: A bounding-boxes aggregation example. The **left** image shows all the text bounding-boxes in various color. The **right** image shows the result after bounding-boxes aggregation process. As we can see, the number of bounding-boxes in the **right** image is much less than that in the **left** image.

bounding-boxes in the group and combine those text together. Next, I assign the resulting text to the group. I also get an external rectangle of these bounding-boxes in each group. A bounding box aggregation is performed again in this way. Fig. 4.6 gives an example of the whole process.

In addition to that, there is another case I should consider. What if the text is too large? As we can see in Fig. 4.7 (a), the text "Computer" occupies a large fraction of the image. Notice that the app is only able to see part of the text in smaller images in this case. To solve this problem, I resize the whole image into a smaller one (1920 x 1080) and use the OCR API to process it. Since the text occupies a large fraction of the picture, it will still be clear even if the whole image is resized. Thus, the app can always detect the text regardless of its size. Fig. 4.7 (b) shows my result.

However, here comes another problem. Notice that eleven pictures should be processed each time. Such process can be time-consuming. To solve this problem, I first try the AsyncTask. It is a class that can allow us to perform background operations efficiently. Using



(a) (b)

Figure 4.7: A text detection example. The **left** image is the original image, and the **right** image shows the text detection result. Notice that the app is able to detect ”Computer” even it occupies a large fraction of the image.

the AsyncTask, I can process eleven images in parallel. The average runtime decreases a lot in this way. However, since the AsyncTask is not designed for time-consuming operations, it is not suitable for my tasks. As a consequence, the app will crash if one of the threads runs a little bit longer. To avoid this problem, I try the ThreadPoolExecutor, which is a manager for multiple threads. It is designed for threads that run long period. To use the ThreadPoolExecutor, I first create an instance of it and add all my tasks into a queue. Then, the works in the queue can be executed at the same time. As a result, the average runtime of this process is 2580ms, which is acceptable in my case.

4.3.2 Bounding box direction calculation

After getting the text result, I use the TextToSpeech API on Android for synthesizing speech from text and notifying the user. In order to present text positions, the app projects each center of the aggregated bounding-boxes to the cylindrical coordinate system. Then, as shown in Fig. 4.8, the app can get a corresponding angle of every bounding box. O represents the

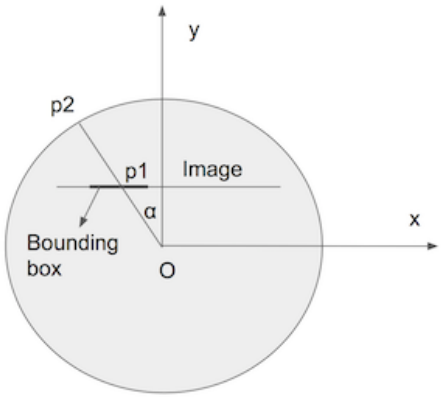


Figure 4.8: Bounding box direction calculation

vertical axis of the cylinder. p1 is the bounding box center. I can get p2 if I project p1 to the cylindrical coordinate system. Then, α is the corresponding angle of the bounding box.

4.4 User notification

In this section, I would like to talk about the two user notification schemes I designed. At first, I tried to inform the user after taking all images. To begin this process, the visually impaired user first pushes the volume up button. Then all the text information found by my app will be read out. Next, the user can select one text region using the Bing speech recognition API from Microsoft and follow the voice instructions given by the app. After that, he or she can rotate the phone to the text direction. For example, as we can see in Fig. 4.9, the user needs to rotate their phone from left to right so that they can reach the text direction. My app first calculates the angle differences between the current camera orientation and the text position. Then the app will notify the user ”move left”. Once the user rotates the phone to the target

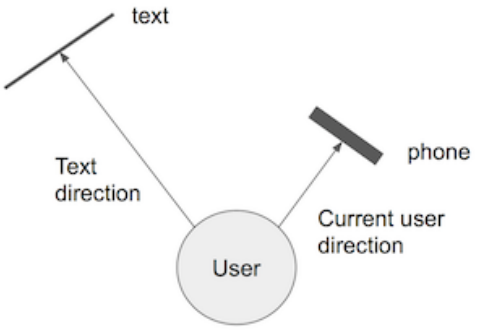


Figure 4.9: User notification example

angle, my app will play a prompt tone. The text in that direction will be read out again. After that, the user can push up volume button and select another region.

This user interface works pretty well. However, the whole process seems a little bit clumsy for that the user needs to rotate their phone back after rotating their phone from left to right. Thus, I come up with another solution: every time the user takes an image, he or she will get informed of the OCR result. The user experience becomes better in this way. Fig. 4.10 illustrates the way I used to inform the user of the text bounding-box positions. For example, if the angle of a bounding-box falls into the [-fov/8, fov/8] interval, the app will tell the user that the text direction is at 12 o'clock.

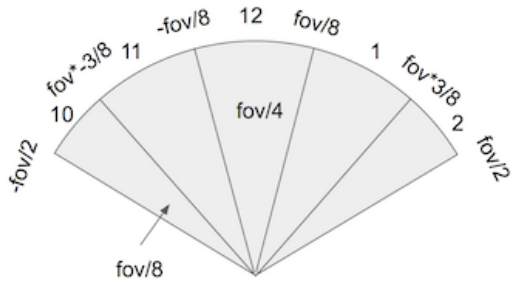


Figure 4.10: The second user notification solution

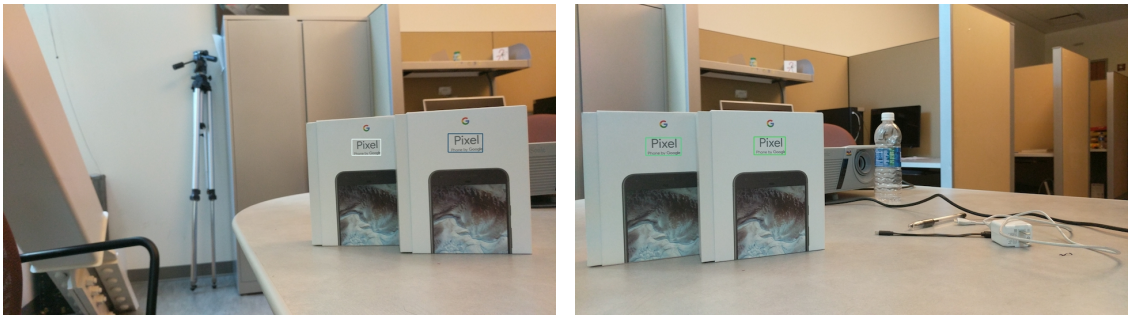
Chapter 5

Image Stitching

Image stitching is the process of generating panorama by combining overlapping images. In this Chapter, I first talk about a problem caused by image overlapping. Next, I describe the well-known concepts of feature detection and image transformation to give context of my work. Then I figure out a way to solve the problem utilizing homography matrices among images. After that, I will discuss the implementation of my C++ image stitching code on Android devices. Besides, I describe several methods I used to optimize the performance of my program. Next, I compare my program to the OpenCV stitching API.

5.1 Repeated text notification problem

As shown in Fig. 4.3, the overlapping is necessary between two adjacent images. Otherwise, the text will be separated. However, this brings another problem: If one object occurs in both of the adjacent images, my app will read the words on the item twice. Such a problem caused by image overlapping will lead to a poor user experience. In addition to that,



(a) (b)

Figure 5.1: The Repeated text notification problem. The **left** image is the first image I took, and the **right** is the second image I took after rotating my phone for 33 degrees. The ”pixel phone by GOOGLE” occurs two times in each image.

if there are two same objects, the result can be confusing. Since the two google pixel packing boxes both occur in Fig. 5.1 (a) and Fig. 5.1 (b), the app will read ”pixel phone by Google” four times, which is indeed a disaster. In the following sections, I will discuss the solution of this problem.

5.2 Stitching method

5.2.1 Feature detection and matching

To figure out the relationships among images, I need to extract features from these images at first. A feature refers to a specific structure which is different from its immediate surroundings in an image. If I can match features in one image with features in another one, I will be able to estimate a homography matrix, which represents the relationship between two images.

To extract features, a suitable feature detector and descriptor should be selected first.

Open Source Computer Vision Library (OpenCV) is an open source computer vision and machine learning software library[10]. There are three feature descriptors available in OpenCV: speeded up robust features (SURF)[11], Oriented FAST and Rotated BRIEF (ORB)[12], and Scale-Invariant Feature Transform (SIFT)[13]. I would like to make a comparison of the three feature descriptors above. SIFT is the slowest and the most accurate descriptor among the three. SURF is much faster than SIFT while its accuracy is a bit lower than SIFT. ORB is the quickest descriptor among the three. But it is also the most inaccurate one. In my case, I would like to choose SURF for that SIFT is just too computationally heavy for a mobile phone, and ORB cannot return a good result in some cases. SURF can detect image features efficiently.

After features are detected in two images, I will use a matcher to match those features. OpenCV provides two kinds of matcher: the FLANN based matcher and the Brute-Force Matcher. The first matcher is faster than the second one. But it is also more inaccurate. I choose the Brute-Force Matcher to get a better result. Then I get the two best matches and apply D.Lowe's ratio test [13]. In this way, I can pick up good matching features between two images.

5.2.2 Image transformation

After I have obtained the matching features, I use RANSAC[15] to calculate a reasonable homography matrix. Though I implement the RANSAC algorithm by myself, I figure out that the OpenCV provides the RANSAC API, which is more stable and robust. Thus, I use the OpenCV RANSAC API instead. The result is pretty good.

To illustrate my result, I would like to create a panorama from multiple images. Here

(a) (b) (c)

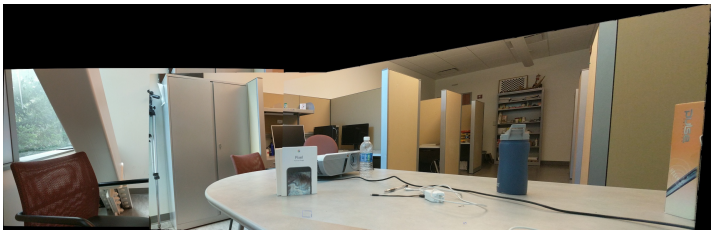
Figure 5.2: Three original images before stitching together

is an example: As shown in Fig. 5.2, there are three images, and I would like to stitch them together. Let these images be image1, image2, and image3. Notice that all these three images should be projected to the cylindrical coordinate system first. Otherwise, as Fig. 5.3 (a) illustrates, the panorama can be weird.

Let the homography matrix between image1 and image2 be H_{21}. Let the homography matrix between image2 and image3 be H_{32}. Let the homography matrix between image1 and image3 be H_{31}. To stitch image1 and image2 together, I keep the first image still and wrap the second image using homography matrix H_{21}. In this case, I use the following well-known formula to transform every pixel in image2 $(P2_x, P2_y)$ to a new point $(P2'_x, P2'_y)$. Notice that $(p2_x, p2_y, p2_z)$ is the Intermediate result.

$$\begin{bmatrix} P2'_x \\ P2'_y \\ 1 \end{bmatrix} = \begin{bmatrix} p2_x/p2_z \\ p2_y/p2_z \\ 1 \end{bmatrix} = \begin{bmatrix} p2_x \\ p2_y \\ p2_z \end{bmatrix} = H_{21} * \begin{bmatrix} P2_x \\ P2_y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} * \begin{bmatrix} P2_x \\ P2_y \\ 1 \end{bmatrix} \tag{5.1}$$

Similarly, if one want to stitch image 2 and image3 together, then one can use the following formula. Every pixel in image3 $(P3_x, P3_y)$ is transformed to a new point $(P3'_x, P3'_y)$.



(a)

(b)

Figure 5.3: The Panorama results. One will get the **top** image if the cylindrical projection is not performed before image stitching process. One will get the **bottom** image if the cylindrical projection is performed at first. Notice that the **top** image looks wried while the **bottom** image looks much better.

Notice that $(p3_x, p3_y, p3_z)$ is the Intermediate result.

$$\begin{bmatrix} P3'_x \\ P3'_y \\ 1 \end{bmatrix} = \begin{bmatrix} p3_x/p3_z \\ p3_y/p3_z \\ 1 \end{bmatrix} = \begin{bmatrix} p3_x \\ p3_y \\ p3_z \end{bmatrix} = H_{32} * \begin{bmatrix} P3_x \\ P3_y \\ 1 \end{bmatrix} \tag{5.2}$$

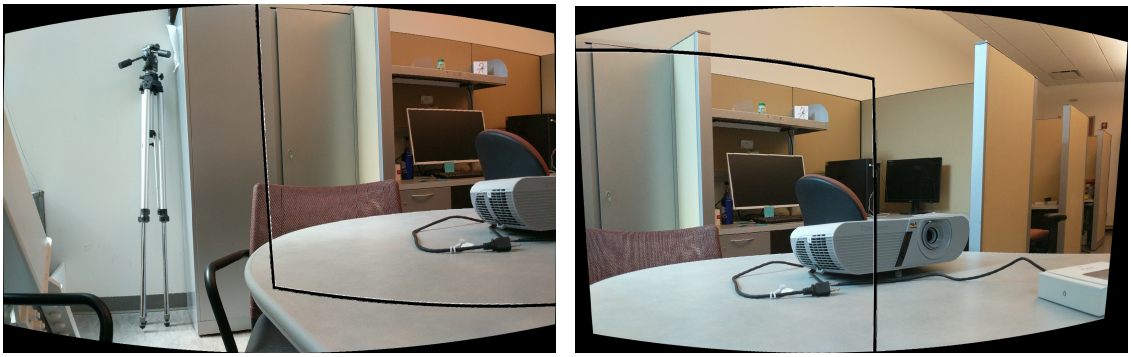
In addition to that, if one want to stitch image1 and image3 together, one can first transform every pixel in image3 $(P3_x, P3_y)$ to a new point $(P3'_x, P3'_y)$ in image2 coordinate using H_{32}. Then one can transform every point in image2 coordinate $(P3'_x, P3'_y)$ to a new point $(P3''_x, P3''_y)$ in image1 coordinate using H_{21}. The following formula illustrates this process:

$$\begin{bmatrix} P3''_x \\ P3''_y \\ 1 \end{bmatrix} = H_{21} * \begin{bmatrix} P3'_x \\ P3'_y \\ 1 \end{bmatrix} = H_{21} * H_{32} * \begin{bmatrix} P3_x \\ P3_y \\ 1 \end{bmatrix} \tag{5.3}$$

Thus, if one want to transform $(P3_x, P3_y)$ to $(P3''_x, P3''_y)$, one can use the following formula:

$$\begin{bmatrix} P3''_x \\ P3''_y \\ 1 \end{bmatrix} = H_{31} * \begin{bmatrix} P3_x \\ P3_y \\ 1 \end{bmatrix} = H_{21} * H_{32} * \begin{bmatrix} P3_x \\ P3_y \\ 1 \end{bmatrix} \tag{5.4}$$

So, if one want to stitch image3 with image1 and image2, one can stitch image1 and image2 at first. Then, one can obtain a new homography matrix H_{31} using the following formula. Next, the H_{31} can be used to wrap image3. Fig. 5.3 (b) shows the stitching result.



(a) (b)

Figure 5.4: Two images with overlapping areas

$$H_{31} = H_{21} * H_{32} \tag{5.5}$$

Besides, Fig. 5.4 and Fig. 5.5 illustrate the overlapping area between two images.

Now the panorama is obtained. I tried to use the google OCR API to process the panorama. However, this idea does not work for the following reasons: Firstly, the panorama is not seamless in my case, which will decrease the quality of the OCR result. As shown in Fig. 5.6, there are two images. Fig. 5.7 shows the stitching result of these images. The OCR API can not find the text ”Pixel phone by Google” for that it is splitted. Secondly, as I have mentioned, the google OCR API doesn't work well if the input image size is too large. Unfortunately, the size of panorama is always pretty large for that the panorama is a combination of several images. Thus, I do not perform text detection on a panorama.

Figure 5.5: An illustration of the overlapping area between two images.

(a) (b)

Figure 5.6: Two original images

Figure 5.7: A panorama with artifacts. The text "Pixel phone by Google" is splitted. Thus, it would be hard for the OCR API to find the text.

5.3 Repeated text notification avoidance

In order to avoid reading the same text multiple times, I come up with this solution: Since the homography matrices represent the relationship among images, I can know the relationship among text bounding-boxes in different photos utilizing the homography matrices. Then, if I transform these text bounding-boxes into the same coordinate system utilizing the homography matrices, I can figure out whether there are overlappings among these bounding-boxes. If one bounding box is overlapped with another one, then I can claim that these two bounding-boxes are generated from the same text. In this case, I would like to select the larger bounding box and only read words in it instead of reading text in two bounding-boxes. Here is an example: Fig. 5.8 shows two pictures taken by my app automatically. First and foremost, I find all the text bounding-boxes in these two images. Then I project these bounding-boxes to the cylindrical coordinate system. After that, I use the homography matrix between these two images to transform the bounding-boxes in the second image, and see if these bounding-

(a) (b)

Figure 5.8: Two original images

boxes are overlapped with the bounding-boxes in the first image. Given a bounding box, we can determine whether we should read the text in it or not. Fig. 5.9 shows the bounding-boxes transformation result. In this way, the app is able to read every text around the visually impaired user only once.

5.4 Implementation

Since my app is for Android devices, the JAVA is used. The OpenCV is also necessary for that I need to deal with many tasks related to the image. However, the OpenCV for JAVA is much less well-documented than the OpenCV for C++. Since there is a tool named Native Development Toolkit (NDK) that can allow us to program in C++ for Android devices, I select OpenCV for C++ instead.

Figure 5.9: The bounding-boxes transformation result. The pink bounding-boxes come from one image while the blue bounding-boxes come from another image. Since there are two pink bounding-boxes overlapped with two blue bounding-boxes, my app can easily figure out that the text ”Pixel phone by google” and ”AQUAFINA” shouldn't be read twice. In addition, the app will read the text ”WHITE BOARD CARE” once.

5.5 Performance optimization and comparison

The program works pretty well as I expected. However, since I use SURF together with the Brute-Force Matcher for better performance, and the image size is pretty large (1920 x 1080), the whole process is quite time-consuming. To reduce the execution time of my program, the POSIX threads is used together with Threading Building Blocks. I also down sample the input images at first. Besides, I compare my program to the OpenCV stitching API in this section.

5.5.1 POSIX threads

POSIX Threads is always referred to pthreads, which is a kind of parallel execution model. By using this in my C++ code, I can execute part of my code parallelly. For instance, I can perform the image distortion correction, cylindrical projection, and image transformation for two images in parallel. POSIX Threads will decrease the execution time of the code.

However, the OpenCV feature detection API and feature matching API cannot be executed in parallel in this way.

5.5.2 Threading Building Blocks

To solve the problem above, I use the Threading Building Blocks (TBB), which is a widely used C++ template library for task parallelism [14]. In order to use that, I manage to rebuild the OpenCV together with TBB for Android platform. Now, it takes less time to run the feature detection and matching codes. As a result, the execution time decreases again.

5.5.3 Image down-sampling

Actually, there is no need to obtain that many correct matching features to get an accurate homography matrix. As I have mentioned, only a minimum of four pairs of good matchings are needed to get a homography matrix. Thus, I decrease the input images sizes by a factor of five. As a result, the average number of matching features between two adjacent images reduce to 42 while the result is still great. Now my app can stitch two 1920*1080 images together using 2160ms on average.

Notice that I do not need to generate a panorama. It is only used to illustrate my result. What I need is the homography matrices among images. Thus, I would like to decline the wrapping procedure, and the execution time of my stitching program decreases to 1759ms on average in this way.

Next, I would like to compare my program to the OpenCV stitching API at first. I will also describe the reason why I didn't choose OpenCV stitching API. OpenCV also pro-

vides a high-level stitching API. One can input several images and get a panorama utilizing the API. There is no need to input camera parameters since the API will estimate the parameters using input images by itself. However, this process is time-consuming. Since I have already obtained the camera parameters by myself, my program does not need such step anymore. Besides, the OpenCV stitching API does many extra steps such as seam masks finding, exposure compensation, and image blending to make the panorama looks better while the panorama is not necessary in my app. Thus, my program is much faster. In addition to that, the source code of this API is complex. Therefore it's hard for me to take control of the whole process. So, I use my own program instead of OpenCV stitching API in this project.

Chapter 6

Conclusion and Future Work

In order to help visually impaired people obtain text information around them, I build an app that can take images automatically and inform the user of text contents and positions. I specifically design the user interface to make sure it is suitable for visually impaired people. Since the google OCR API does not work well when the input image size is large, I propose a technique to solve this problem. I also present a way to solve the repeated text notification problem caused by image overlapping. After that, I figure out various techniques that can be used to decrease the execution time of the whole process. In addition to that, I compare my stitching program to the OpenCV stitching API. As a result, my program has better performance.

In the future, I would like to analyze all the text obtained by the app and filter the meaningless information out. Besides, I am also interested in figuring out a way that can further speed up the whole process and improve user experience.

Bibliography

[1] H.-C. Wang, R. K. Katzschmann, S. Teng, B. Araki, L. Giarre , and D. Rus, Enabling independent navigation for visually impaired people through a wearable vision-based feedback system, in Robotics and Automation (ICRA), 2017 IEEE International Conference on. IEEE, 2017, pp. 65336540.

[2] Oh Y., Kao WL., Min BC. (2017) Indoor Navigation Aid System Using No Positioning Technique for Visually Impaired People. In: Stephanidis C. (eds) HCI International 2017 Posters' Extended Abstracts. HCI 2017. Communications in Computer and Information Science, vol 714. Springer, Cham

[3] B.-S. Lin, C.-C. Lee and P.-Y. Chiang, "Simple Smartphone-Based Guiding System for Visually Impaired People," Sensors, vol. 17, no. 6, p. 1371, 13 Jun 2017.

[4] L. Niu, C. Qian, J.-R. Rizzo, T. Hudson, Z. Li, S. Enright, E. Sperling, K. Conti, E. Wong and Y. Fang, "A Wearable Assistive Technology for the Visually Impaired with Door Knob Detection and Real-Time Feedback for Hand-to-Handle Manipulation," 2017.

[5] Chucai. Yi, Y. Tian, Areis. Arditi, "Portable Camera-Based-Assistive-Text-and-Product Label Reading From Hand-Held Objects for Blind Persons", IEEE/ASME TRANSACTIONS ON MECHATRONICS.

[6] Seeing AI (n.d.). Retrieved March 16, 2018, from Microsoft: https://www.microsoft.com/en-us/seeing-ai/

[7] Z. Zhang. "A flexible new technique for camera calibration." IEEE Transactions on Pattern Analysis and Machine Intelligence. vol. 22(11), pp. 1330-1334 (2000).

[8] Supported Images (n.d.). Retrieved March 16, 2018, from Google Cloud Vision API Documentation: https://cloud.google.com/vision/docs/supported-files

[9] Text Recognition API Overview (2017, October 24). Retrieved March 16, 2018, from Mobile Vision: https://developers.google.com/vision/text-overview

[10] About (n.d.). Retrieved March 16, 2018, from OpenCV: https://opencv.org/about.html

[11] Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool, "SURF: Speeded Up Robust Features", Computer Vision and Image Understanding (CVIU), Vol. 110, No. 3, pp. 346–359, 2008

[12] Rublee, Ethan, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: an efficient alternative to SIFT or SURF. In Computer Vision (ICCV), 2011 IEEE International Conference on, pp. 2564-2571. IEEE, 2011.

[13] D. G. Lowe. Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision, 60(2):91110, 2004. 1, 2

[14] Why Use Intel TBB? (n.d.). Retrieved March 16, 2018, from Threading Building Blocks: https://www.threadingbuildingblocks.org

[15] Martin A. Fischler & Robert C. Bolles (June 1981). "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography" (PDF). Comm. ACM. 24 (6): 381395. doi:10.1145/358669.358692