FYP FINAL REPORT

IMAGE PROCESSING / SHAPE DETECTION

Submitted by

NAICHUAN ZHANG

Student No.: 18111521

Supervisor: Patrick Healy

BSc in Computer Systems

University *of* Limerick

April 2020

Table of Contents

Chapter 1: **Project Summary**

The aim of this project is to investigate the use of Android and computer vision together by developing an Android application, named *Image Processor*, that can perform geometric shape detection based on a given image, which can be obtained through either the system camera or gallery. In addition, the application will also be able to generate panorama by stitching multiple selected images by user, in order to perform shape detection.

This project mainly uses Android Studio as the development tool with OpenCV4Android library along with Java Native Interface (JNI) in Android to realize an Android powered mobile application with the main purpose of image shape analysis and research.

Chapter 2: **Introduction**

This chapter provides a brief overview of the background and implication of the project research. And it also shows readers with a roadmap for the overall presentation and structure of the report.

2.1 Development Background

The ever-changing development of multimedia technology and computer networks has enabled people to obtain an increasing amount of image information in their daily lives, as well as more image information that needs to be processed. As an important information carrier, image plays an import role of multimedia information. Therefore, image processing technology has been widely used in all lines of work.

Sometimes we need to automatically detect the color of the target in the image and its corresponding shape in real life and some specific professional fields. Therefore, shape detection is widely used and significantly important in the field of traditional image processing and machine vision, since people are often interested in the targets with regular geometric shapes, such as the detection of machine components in industrial production, the location of circular nucleus in biological images and the identification of signs with specific shapes in highway transportation, and so on.

2.2 Computer Vision

The computer vision field focuses on facilitating a computer to understand or interpret visual information from image or video sequences and it originated in the late 1950s and early 1960s (Bebis *et al.* 2003).

2.3 OpenCV

OpenCV (Open Source Computer Vision) is a cross-platform computer vision library based on BSD license (open source) distribution. It is a powerful library of functions and methods which enable us to actualize complex detection algorithms across multiple platforms. Places that need detection in real time, this library will be extremely suitable.

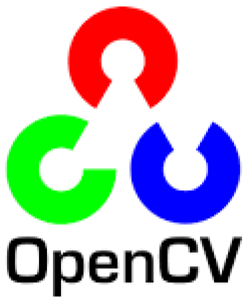
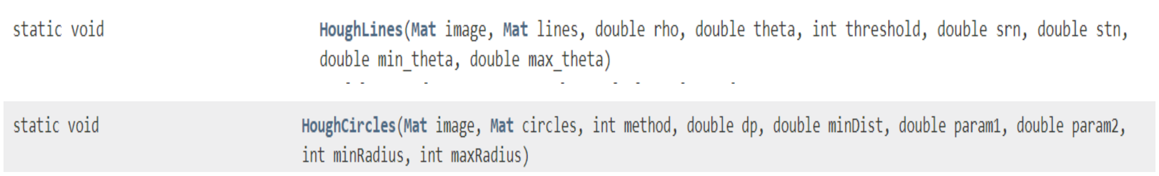


Figure 2.3.1: OpenCV logo (OpenCV 2019)

The OpenCV4Android is selected as the main library for the project because it is lightweight and efficient. It has achieved a rich number of optimized computer vision algorithms with excellent performance and speed accuracy. For example, **Figure 2.3.2** lists two of the functions provided in OpenCV which are used to find lines and circles in a binary image through Hough Transform method.



```
static void    HoughLines(Mat image, Mat lines, double rho, double theta, int threshold, double srn, double stn,
               double min_theta, double max_theta)

static void    HoughCircles(Mat image, Mat circles, int method, double dp, double minDist, double param1, double param2,
               int minRadius, int maxRadius)
```

Figure 2.3.2: HoughLines() and HoughCircles() in OpenCV (OpenCV 2019)

2.4 Motivations

Vision, as one of the most important human senses, plays a crucial role in people's daily life. Human vision can easily recognize objects in a scene by using a variety of information sources, but this can be quite a tedious task for a computer. We know that computer vision has received much attention in recent years, and this research field so far has spawned a large number of fast-growing, practical applications including image processing, such as biometric recognition, face detection and so on. Object detection, as one of the five key components of computer vision, is particularly important. The study of shapes has been a topic repeatedly discussed in object detection, as shape is the main source of information for detecting objects. In addition, as far as I know, this technique is still not very mature so far for various reasons. Therefore, during the research of this project, I will mainly focus on the part of shape detection.

From a personal perspective, I have always been interested in the field of computer vision, and I also want to make computer vision one of my future research areas, so it is also a significant factor in my motivation for this project.

2.5 Objectives

The main objectives to be achieved for the project are listed as follows:

- Gain a deep understanding of the field of computer vision
- Consolidate the Android system architecture and improve proficiency in using Android Studio for Android mobile app development
- Gain knowledge and understanding of OpenCV4Android library as well as the Java programming language
- Implement geometric shape detection and image stitching features
- Get familiar with Java Native Interface (JNI) in Android and also C++ programming language
- Develop a reusable application by applying design patterns, etc.

2.6 Roadmap

This report is divided into five chapters apart from Chapter 1 and 2, the structure is as follows:

Chapter 3 discusses the image processing technologies and algorithms that will be applied in the project as well as some related works. Chapter 4 gives a detailed demonstration of system analysis and design. Chapter 5 provides a detailed demonstration of the implementation of the full scope of the project with appropriate screenshots and explanations. Chapter 6 will mainly focus on the testing process of this project. And finally, Chapter 7 will be an ‘Evaluation and Conclusion’ chapter which will evaluate and summarize the project and also includes my personal summary and future work. **Table 2.6.1** below gives an indication to the structure of the report.

Chapter 3 **Research**	discuss the image processing technologies and algorithms that will be applied in the project as well as some related works
Chapter 4 **Analysis and Design**	gives a detailed demonstration of system analysis and design
Chapter 5 **Implementation**	provides a detailed demonstration of the implementation of the full scope of the project with appropriate screenshots and explanations
Chapter 6 **Testing**	mainly focus on the testing process of this project, will be expanded from three aspects: unit testing, UI testing and system testing
Chapter 7 **Evaluation & Conclusion**	evaluate and summarize the project and also includes personal summary as well as future work

Table 2.6.1: Table chart for the structure of the report

Chapter 3: **Research**

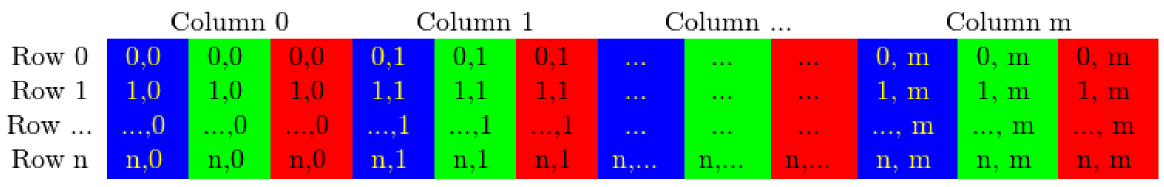
Based on the previous chapter, this chapter will detail the reading materials covered in the entire project research process so far and the algorithms that will be applied in the project (such as Hough Transform, etc.).

3.1 Image Representation in OpenCV

In OpenCV, the image is stored and represented as a custom object called *Mat*. This object stores the information such as rows, columns, data and so on that can be used to uniquely identify and recreate the image when required (Kapur and Thakkar 2015).

The color models used in image representation mostly are RGB and Grayscale Color Models. RGB model is based on three basic colors (Red, Green and Blue), which are superimposed to different degrees to produce various colors. The different colors are obtained by adjusting the proportion of each basic color, and the proportion is evaluated by fractional values between 0 and 1 in OpenCV, for instance. The Grayscale color model is relatively simpler than RGB model, which only holds information about the intensity in each pixel, that is, it can be also described as a black-and-white image.

In OpenCV, a colored image consists of three channels, which refers to Blue, Green and Red (BGR) respectively (**Figure 3.1.1**) while a grayscale image only has one channel.



	Column 0			Column 1			Column ...			Column m		
Row 0	0,0	0,0	0,0	0,1	0,1	0,1	...	...	...	0, m	0, m	0, m
Row 1	1,0	1,0	1,0	1,1	1,1	1,1	...	...	...	1, m	1, m	1, m
Row ...	...,0	...,0	...,0	...,1	...,1	...,1	...	...	...	..., m	..., m	..., m
Row n	n,0	n,0	n,0	n,1	n,1	n,1	n,...	n,...	n,...	n, m	n, m	n, m

Figure 3.1.1: BGR color model in OpenCV (OpenCV 2019)

In addition, there is another popular color model in OpenCV, which is called HSV (Hue, Saturation and Value). This model was firstly proposed for better digital processing of colors by Smith A.R. in 1987, which uses a hexagonal cone to represent.

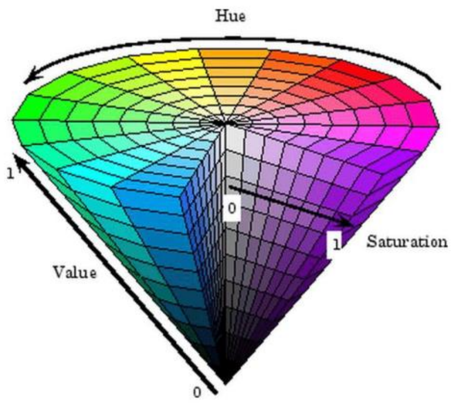


Figure 3.1.2: HSV color model (ResearchGate 2014)

3.2 Shape Detection Overview

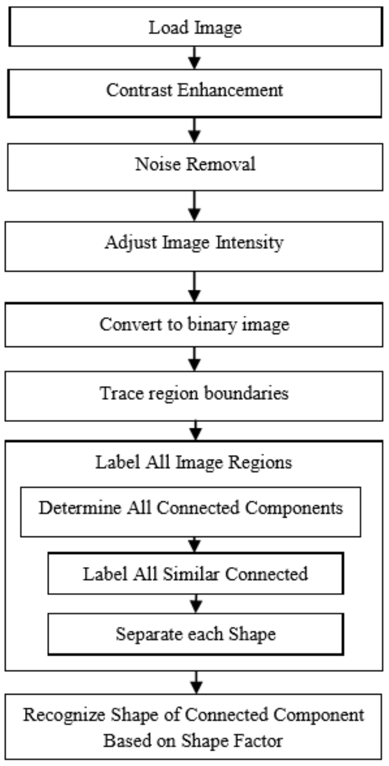


Figure 3.2.1: Shape detection block diagram (Abbadi 2013)

The **Figure 3.2.1** above shows an overall procedure to perform shape detection. As stated by Elrefaei, in order to be able to detect the shape of different objects, a simple implementation idea can be shown as following:

1) Image acquisition, the image resources can be obtained in a certain way. In this project, the image resource will be accessed either from a built-in camera from the device or from the system gallery (refer to **Chapter 4** for details).
2) Image pre-processing, which basically includes image binarization (grayscale), noise reduction, etc.
3) Feature extraction
4) Edge detection / contour extraction

5) Geometric shape identification
6) Contour drawing and results displaying

(Elrefaei *et al.* 2017)

3.3 Image Preprocessing

The main purpose of this procedure is to selectively highlight the useful features in the source image and appropriately attenuate the secondary features so as to increase the reliability of detection results (Yang 2015). Since the specific process of image preprocessing varies in different situations, only a few common preprocessing methods will be introduced.

3.3.1 Image Binarization

The very first step to preprocess the image is to convert the RGB color image accessed from camera/gallery to a grayscale image, which is also known as image binarization. As mentioned above, when we need to process colored images, we are often required to deal with three channels in order, which will consume a lot of time, so in order to improve efficiency, we have to convert it to a binary image before anything else.

3.3.2 Noise Reduction

Noise is the random variation in brightness or color information in images captured (Swain 2018). Image noise reduction is an image enhancement step to satisfy some special analysis requirements by improving image quality and enriching information. In other words, this can also be understood as emphasizing the useful features of the image and suppressing the useless features of the image. In order to remove noise in an image, we usually use blurring operations to achieve that.

Mean Blur – Mean blur is the simplest form of blurring operations. It uses a linear method to average the pixel values over the entire window range. However, this method destroys the details of the image while denoising the image

Gaussian Blur – The gaussian blur is similar to the mean blur in that it takes the mean value of the pixels in the filter window as the output. The coefficient of the window template is different from that of the mean blur. The template coefficient of gaussian blur decreases with the increase of the distance to the center of the template. Therefore, compared with the mean value blur, the gaussian blur has less blurring effect on the image

Median Blur – One of the common types of noise represent in images is called *salt-and-pepper noise* (**Figure 3.3.2.1**). Median blur is a good way to get rid of this type of noise. It averages the pixels covered by the kernel in ascending/descending order, and set the value of the middle element as the final value of the anchor pixel (Kapur and Thakkar 2015)

Figure 3.3.2.1: Example of salt-and-pepper noise (Wikipedia 2019)

Of the several denoising methods described above, each has its own API in OpenCV library. The following figure shows the results of different blurring methods of an image.

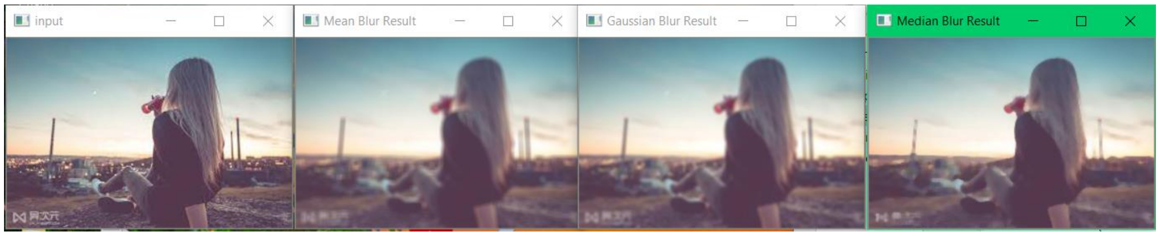


Figure 3.3.2.2: Blurring test results (Input, Median Blur Result, Gaussian Blur Result and Median Blur Result, from left to right)

3.4 Edge Detection

3.4.1 Canny Edge Detector

One of the easiest ways to detect edge on an image is to directly smooth the image by using filters (such as Gaussian blur) and then use the Sobel operator to calculator the image gradient and perform threshold processing with hysteresis. The purpose of the final step is to ensure that a pixel has a gradient magnitude greater than the upper threshold (AI Shack 2018). **Figure 3.4.1.1** shows a test result by applying Canny Detector. We can see the edges produced by Canny are very thin, maybe as thin as a pixel, and the overall strength of the detected edges is consistent.

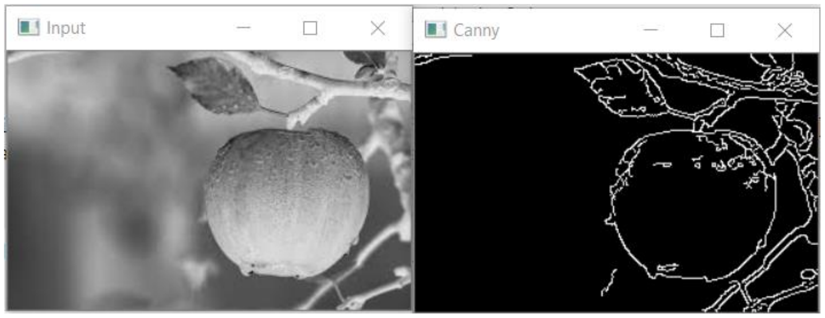


Figure 3.4.1.1: Canny test result

3.4.2 Sobel Operator

The Sobel operator contains two sets of 3x3 matrices, which are horizontal and vertical templates, respectively. Convolving them with the image plane, we can get the horizontal and vertical brightness difference approximations, respectively (Kapur and Thakkar 2015).

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Then calculate the absolute gradient at each pixel by using

$$G = \sqrt[2]{G_x^2 + G_y^2}$$

Compared with Canny's test result, the edges produced by Sobel have strong and weak edges and good noise resistance (shown in **Figure 3.4.2.1**).

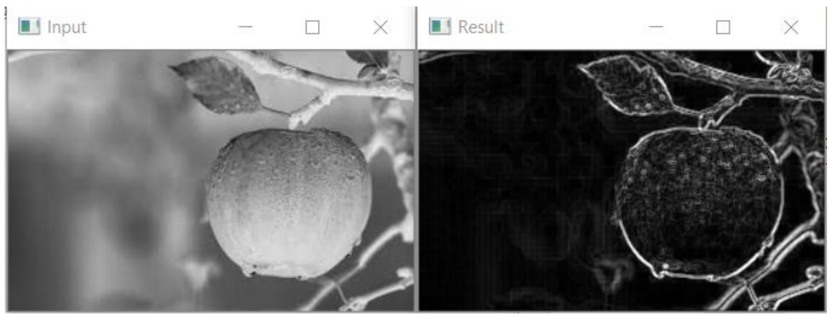


Figure 3.4.2.1: Sobel test result

3.5 Hough Transform

Hough Transform is one of the basic methods for identifying geometric shapes from images in image processing. The classical Hough Transform can be used to detect straight lines from a given image, and later extends to the recognition of objects of any shapes, mostly used to detect circles and ellipses, although the complexity of the transformation increases with the number of parameters needed to describe the shape (Chetan and Anita 2015).

3.5.1 Line Detection

As is known, in a Cartesian coordinate system, a straight line can be defined with two points, $A = (x1, y1)$ and $B = (x2, y2)$ (shown in **Figure 3.5.1.1**). And it can be represented by the equation $y = kx + b$, where k and b are parameters, respectively slope and intercept. All lines passing through point A can be expressed as $y1 = kx1 + b$, where k and b can be various values. The main idea of the Hough transform is to exchange parameters and variables of the equation, that is, take x , y as known quantity and k , b as coordinate axes, then the equation of A can be converted to $b = -x1k + y1$ and for B it is $b = -x2k + y2$, and the common point of two lines $(k1, b1)$ is the line that uniquely identifies the connections A and B in Cartesian coordinate (shown in **Figure 3.5.1.2**). However, there are still some problems left. Since lines perpendicular to the x -axis will have an k -value of infinity. This will force the parameter space k b to have infinite size (Kimme *et al.*1975).

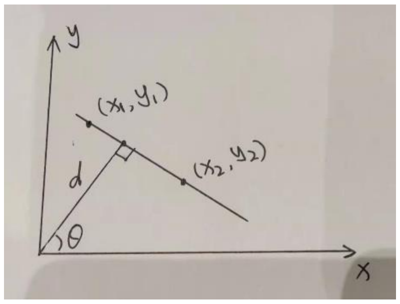


Figure 3.5.1.1: Line Detection-1

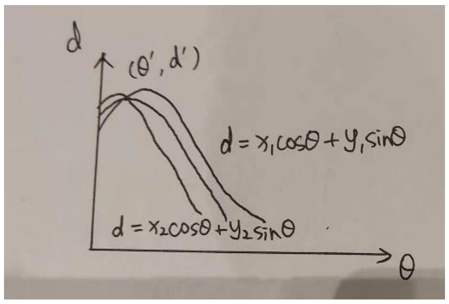

d
(θ', d')
$d = x_1\cos\theta + y_1\sin\theta$
$d = x_2\cos\theta + y_2\sin\theta$
θ

Figure 3.5.1.2: Line Detection-2

3.5.2 Circle Detection

For a circle, the general equation is $(x - a)2 + (y - b)2 = r2$, where (a, b) is the center of the circle and r is the radius. Generally, a circle can be uniquely identified as $C = (a, b, r)$.

The principle of Hough circle detection and line detection is similar. Hough circle transform is a process of converting a circle from a 2D image space into a point in 3D parameter space determined by three parameters above. Therefore, the circle determined by any three points on the circumference should correspond to a point in the 3D parameter space after Hough transform. However, this method of detecting circles is difficult to be practically applied since both time and space complexity in the three-dimensional space greatly reduce the computational efficiency.

3.6 Polygon Detection

In order to be able to detect more complicated geometric shapes like triangles, rectangles, squares or others, Elrefaei proposed the follow algorithm:

1. Find contours and apply Douglas-Peucker algorithm to approximate a contour shape to another shape with less number of vertices depending on a specified precision
2. Check the approximated vertices value:
    a. If it is three, then the shape is a triangle
    b. If it is four, the shape is a square or a rectangle or any polygonal have Four vertices. To specify the square and the rectangle, calculate the angle between two vertices by the dot product. Then to specify which one is a square and which one is a rectangle, calculate the ratio between the width and the height. Any other polygonal have four vertices is detected as “Other”
    c. If it is greater than four, the shape is detected as “Other”
3. Draw the contours of the detected shapes

(Elrefaei *et al.* 2017, pp.24-25)

3.7 Git

Git is an open source system that helps developers implement version control very easily and effectively. To be honest, I had never tried any version control systems before I came to Ireland. Working with local students brought me a chance to get in touch with Git. So far, I have used Git in some projects, but I still don’t know enough about it. Therefore, I continue to use Git as the version control tool for this project. From this perspective, using Android Studio as a development tool will have great benefits, as it comes with a powerful version control tool by default, which greatly simplifies the steps of directly using CLI. However, I still need to use Git command with Git Bash in some cases, so this urges me to learn some useful tutorials online to better deal with different situations during the development.

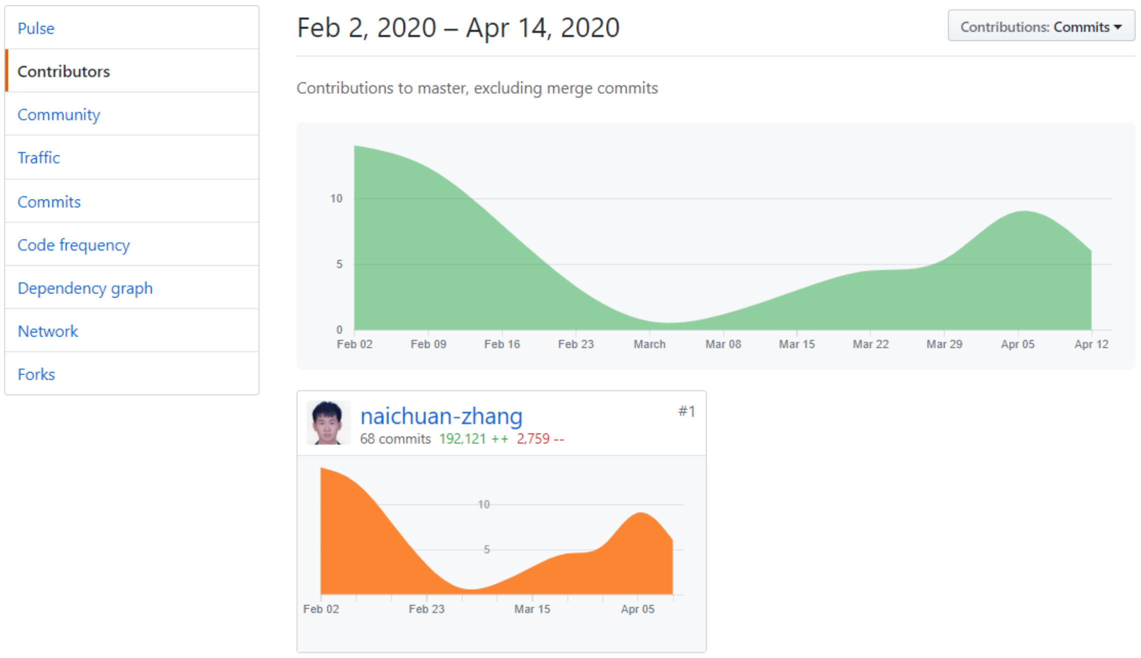


Figure 3.7.1: Project repository on GitHub

Chapter 4: **Analysis and Design**

This chapter mainly demonstrates how the project framework is constructed by providing details of analysis and design processes.

4.1 Requirements Analysis

4.1.1 Functional Requirements

This system should have the following functions:

1. **Image resource acquisition**

   User can generally access image sources in two approaches: 1) by taking photos with device's build-in camera; 2) by selecting pictures through the device's gallery. The acquired image resource is then saved by the system.

2. **Image processing**

   User can edit and modify images, for example, grayscale the image.

3. **Image persistence storage**

   All user-obtained image resources through the system can be saved in the system persistently. The image won't be lost when the users restart the application.

4. **Image viewing and saving**

   All user-obtained image resources can be viewed repeatedly in the history and user can also download a specific image to the current smart phone.

5. **Shape detection**

User can select an image and detect different geometric shapes (triangles, quadrangles, pentagons, hexagons, circles and ellipses, etc.) in the given image. The detection results can be viewed.

6. Image management

Apart from editing the images, user can also delete, update images through the app.

7. Image sharing

User can share images through email, message or some other third-party applications on the phone.

8. Image alignment and stitching

In order to compose images into a panorama, user is allowed to use the image stitch provided by the system to take multiple consecutive images, and then can stitch the acquired images into a panoramic image.

9. Real Time Detection

The system provides a custom camera to dynamically detect geometric shapes in real time. This module allows users to adjust a variety of camera parameters, such as camera zooming, and also supports user adjustment of detection thresholds. In addition, the real time detection tool also supports multiple display views, such as grayscale, canny edge detection, contour detection and other features. Users are also allowed to turn on the flash light.

10. Dark mode theme

The dark mode theme is provided by the system, user can turn on or off the dark mode through the setting interface in the system. This user preference will be persisted.

11. Internationalization (display language)

In order to provide a better user experience, the system is equipped with a multi-language switching function, users are allowed to switch display language in the settings interface, the currently available languages include English and Simplified Chinese. This user preference will be persisted.

Based on the above functions, the system strives to be friendly and beautiful user interface, operate conveniently and quickly, and makes the system as practical and perfect as possible.

4.1.2 Use Case Analysis

The use case analysis will be presented both in written and graphical form. Based on the functional requirements described previously, the following use cases are derived:

Actor	Use Cases
App User	Take photo through camera
	Select image from gallery
	View history
	View images (Photo View)
	Detect geometric shapes
	Download images to phone
	Real time geometric shape detection
	Stitch images using stitcher
	Share images
	Change display language
	Change dark mode theme

Table 4.1.2.1: Use case specification

According to **Figure 4.1.2.1** below, the app user is shown as the sole actor for the application. There are two options for a user to acquire image resources from the application, either by taking a photo or selecting from gallery. The users are not allowed to check history, perform shape detection or stitch images through stitcher before completing the Save image use case can be executed. In addition, an app user can also preview all images saved in the history and user is allowed to download a specific image to the current phone. If a user wants to share the image, the sharing process might be interrupted due to internet connection or for some other reasons. Users can also change display language (Internationalization) or change theme according to personal preference under the settings screen.

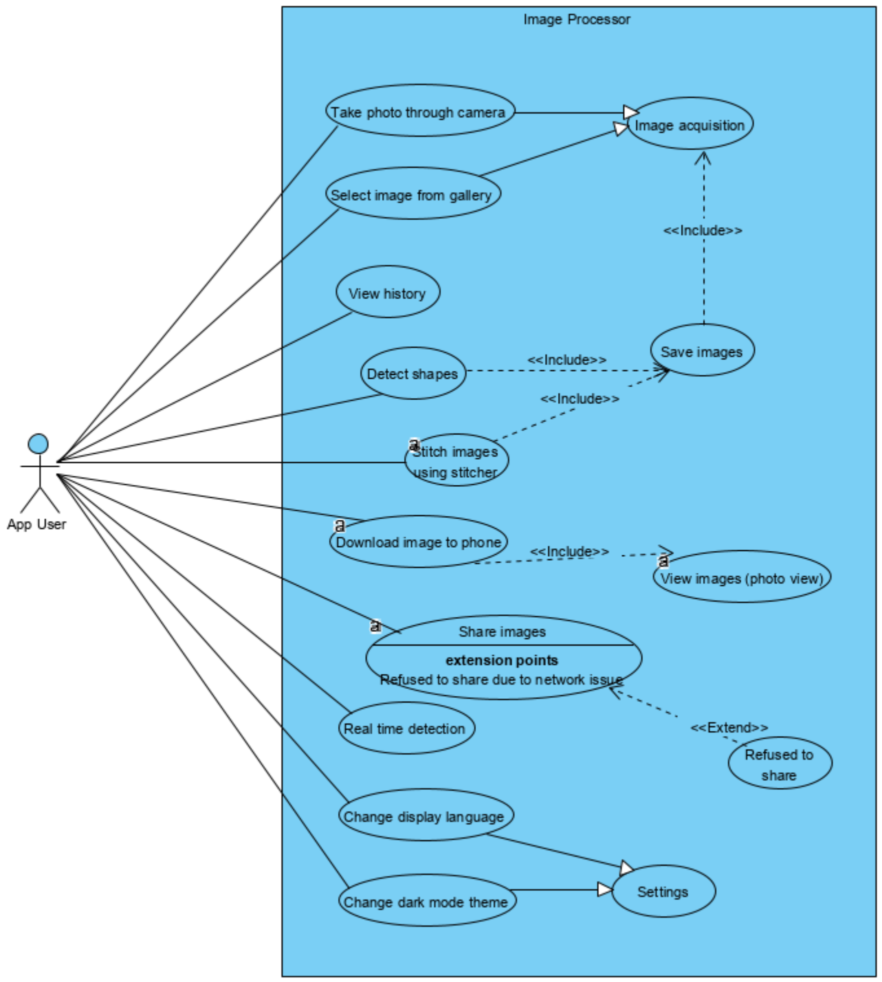


Figure 4.1.2.1: Use case diagram

4.1.3 Non-Functional Requirements

1) Performance

Performance is one of the most important evaluation indexes of an application. It is essential to ensure that the app can run smoothly and with no bugs. Rational usage of threads is the key to ensure stable performance.

2) Extensibility

The system's architecture should allow new features to be added or to be modified when a better implementation is available. Given that with system like this, high cohesion and managed dependencies with services well-specified through interfaces are vitally important. The MVVM (Model-View-ViewModel) architectural pattern should be followed when developing components to allow the system to be independently extensible with respect to adding new views using existing models. In addition, following design patterns is also the key to guarantee high extensibility of the system.

3) Usability

The system needs to ensure that the user can understand and use the app well without any guidelines or helps of other people.

4) Maintainability

Keeping the application well-organized and breaking the system down into smaller parts so that each component will be more focused and easier to be understood. Compliance with MVVM's architectural model can help me to achieve this much easier.

4.2 GUI Prototypes

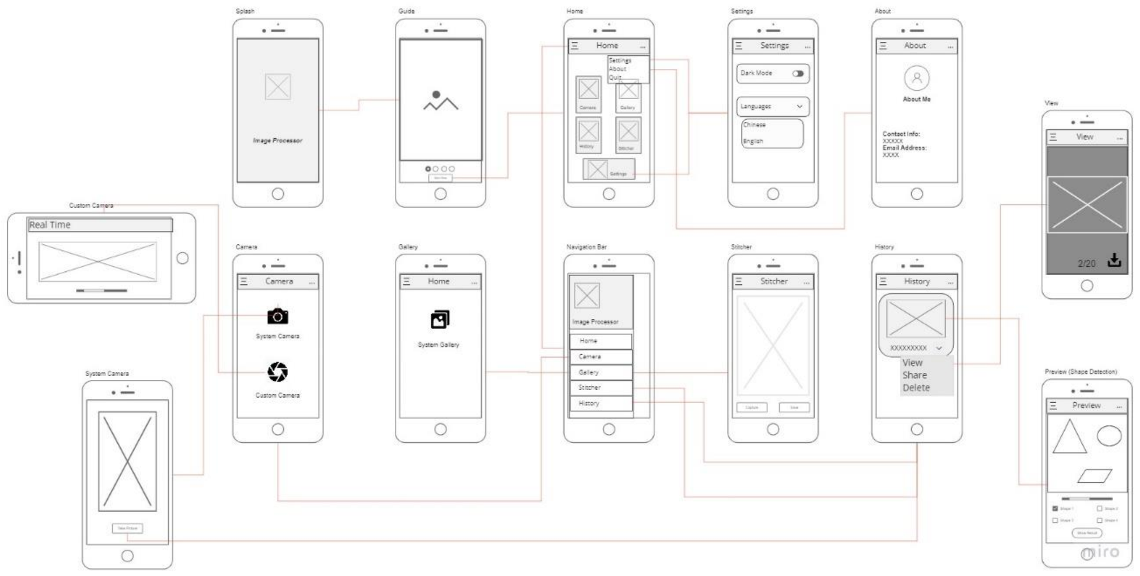
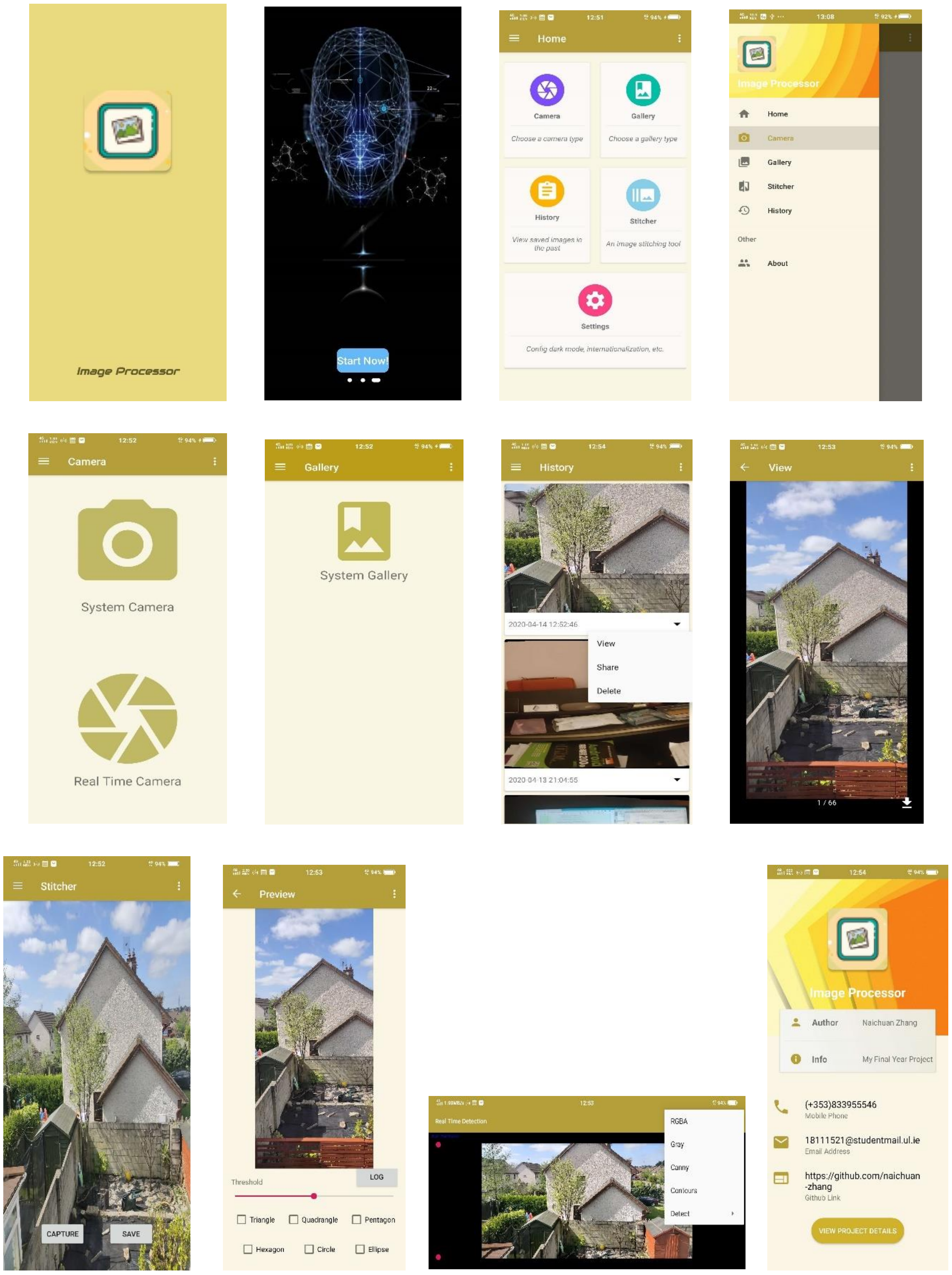


Figure 4.2.1: GUI Prototypes

As shown in **Figure 4.2.1**, it depicts an overview of the wireframes for the proposed application, it presents a general idea of the expected appearance of the final product, how screens interact with each other and how widgets will lay out on the screen, whereas **Figure 4.2.2** demonstrates screenshots of some main GUIs for the current application, which are Splash, Guide, Home, Navbar, Camera, Gallery, History, Photo View, Stitcher, Detection, Real Time Detection and About Me screens, respectively (from left to right).



Author Naichuan Zhang

Info My Final Year Project

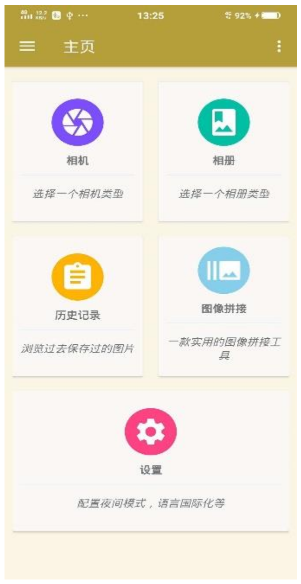
(+353)833955546
Mobile Phone

18111521@studentmail.ul.ie
Email Address

https://github.com/naichuan-zhang
Github Link

Figure 4.2.2: Screenshots of main GUIs

As described in functional requirements, users can modify the display languages of this mobile application, the program currently supports two languages, which are English (default) and Simplified Chinese, respectively. Here is a Home screen displayed in Simplified Chinese (shown in **Figure 4.2.3**). In addition, users can also personalize the app to be shown in dark mode (shown in **Figure 4.2.4**).



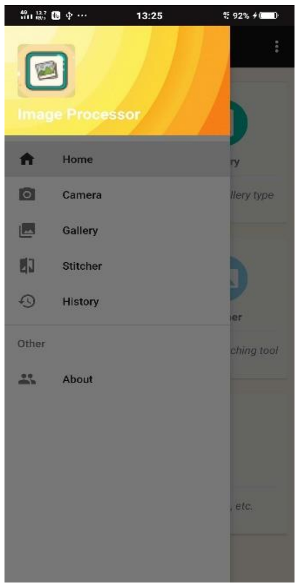


Figure 4.2.3: in Simplified Chinese

Figure 4.2.4: in Dark Mode

4.3 Database

As we know, persistent storage of data is an issue that needs to be considered in almost all Android app development processes. To achieve data persistence, people usually save data through SharedPreference, which is the best way to keep smaller key-value pairs of primitive data types. But when we need to store structured data, using a database is often easier. In Android, Room provides an abstraction layer over SQLite to allow fluent database access while harnessing the full power of SQLite (Android Developers 2019). As shown in **Figure 4.3.1**, the three basic components of Room are Database, Dao and Entity.

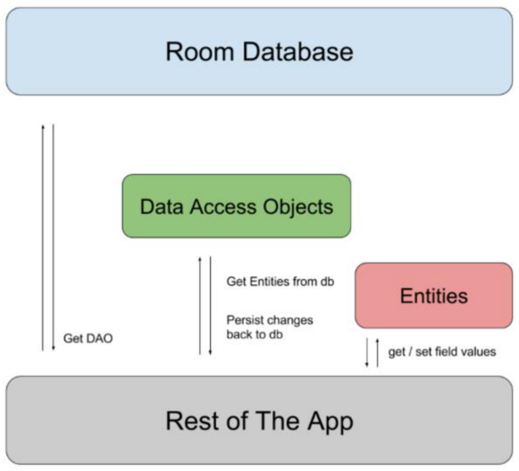


Figure 4.3.1: Room Architecture (MindOrks 2018)

In this project, in order to enable users to easily obtain image resources saved in the application, it requires a Room database to persistently store the image data accessed by users. This application only requires one entity in the database to save image data (shown in **Figure 4.3.2**). We can see that the *Image* table has a primary key called *imageID* which can be automatically generated by Room. *imageName* is an attribute used to store the filename of the image resource, which is generated once after a user wants to save this image captured from camera or obtained through system gallery. The *imageDate* and *imageURI* is used to store the saved date and the uniquely identified store path in an Android device, respectively. And *imageSource* is used to mark the source of the image, which can be from camera, gallery or a stitched image by image stitcher.

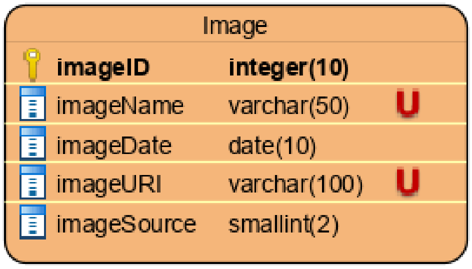
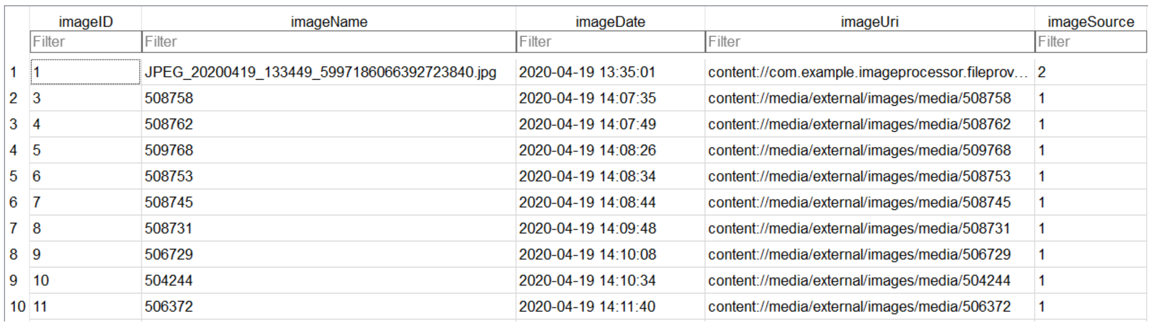


Figure 4.3.2: *Image* Entity

Figure 4.3.3 shows some data saved in the *Image* table. These data are obtained through *DB Browser for SQLite*.



	imageID	imageName	imageDate	imageUri	imageSource
	Filter	Filter	Filter	Filter	Filter
1	1	JPEG_20200419_133449_5997186066392723840.jpg	2020-04-19 13:35:01	content://com.example.imageprocessor.fileprov…	2
2	3	508758	2020-04-19 14:07:35	content://media/external/images/media/508758	1
3	4	508762	2020-04-19 14:07:49	content://media/external/images/media/508762	1
4	5	509768	2020-04-19 14:08:26	content://media/external/images/media/509768	1
5	6	508753	2020-04-19 14:08:34	content://media/external/images/media/508753	1
6	7	508745	2020-04-19 14:08:44	content://media/external/images/media/508745	1
7	8	508731	2020-04-19 14:09:48	content://media/external/images/media/508731	1
8	9	506729	2020-04-19 14:10:08	content://media/external/images/media/506729	1
9	10	504244	2020-04-19 14:10:34	content://media/external/images/media/504244	1
10	11	506372	2020-04-19 14:11:40	content://media/external/images/media/506372	1

Figure 4.3.3: Some saved data in *Image* table

4.4 Analysis Sketches

4.4.1 Candidate Classes Identification

Data Driven Design is applied to identify all candidate classes. Here is an initial list of all possible candidate classes I found from functional requirements by using noun identification technique.

List of Possible Candidate Classes				
system	image	photo	camera	user
gallery	shapes	lines	circles	ellipses
rectangles	history	home	settings	share
about	display language	theme	stitcher	real time detection

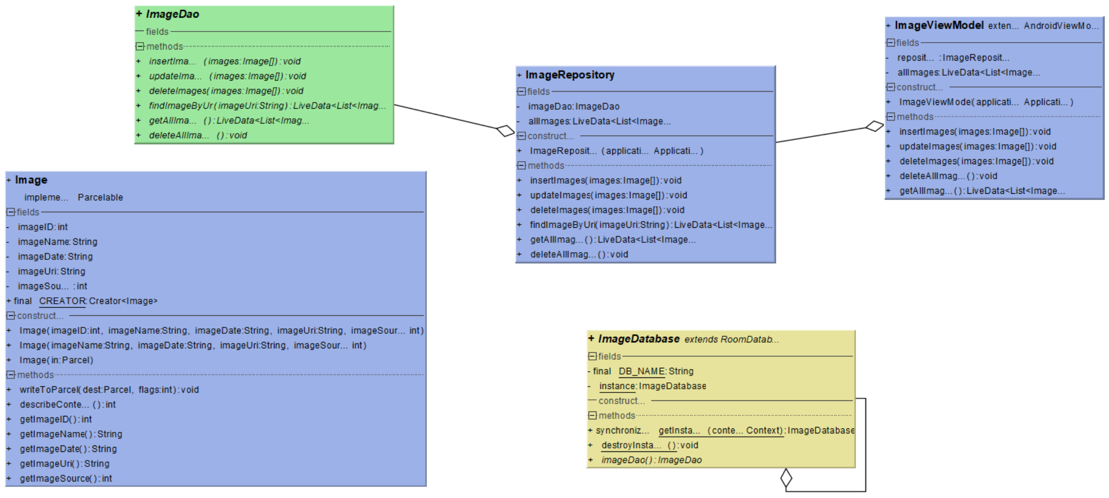
Table 4.4.1.1: List of possible candidate classes

After careful consideration and filtering, I intend to eliminate/modify the following possible candidate classes (in red) for the following reasons.

Classes	Explanations
system	Part of meta-language of functional requirements
photo	Same as image
user	Not considered in the application
lines, circles, ellipses, rectangles	Covered by shapes
display language, theme	Covered by settings

Table 4.4.1.2: Eliminated classes with explanations

4.4.2 Analysis-Time Class Diagrams



+ ImageDao
fields
methods
+ insertIma... (images:Image[]):void
+ updateIma... (images:Image[]):void
+ deleteImages(images:Image[]):void
+ findImageByUr(imageUri:String):LiveData<List<Imag...
+ getAllIma... ():LiveData<List<Imag...
+ deleteAllIma... ():void
+ ImageRepository
fields
- imageDao:ImageDao
- allImages:LiveData<List<Image...
construct...
+ ImageReposit... (applicati... Applicati...)
methods
+ insertImages(images:Image[]):void
+ updateImages(images:Image[]):void
+ deleteImages(images:Image[]):void
+ findImageByUri(imageUri:String):LiveData<List<Image...
+ getAllImag...():LiveData<List<Image...
+ deleteAllImag...():void
+ ImageViewModel exten... AndroidViewMo...
fields
- reposit... :ImageReposit...
- allImages:LiveData<List<Image...
construct...
+ ImageViewMode(applicati... Applicati...)
methods
+ insertImages(images:Image[]):void
+ updateImages(images:Image[]):void
+ deleteImages(images:Image[]):void
+ deleteAllImag...():void
+ getAllImag...():LiveData<List<Image...
+ Image
impleme... Parcelable
fields
- imageID:int
- imageName:String
- imageDate:String
- imageUri:String
- imageSou... :int
+ final CREATOR:Creator<Image>
construct...
+ Image(imageID:int, imageName:String, imageDate:String, imageUri:String, imageSour... int)
+ Image(imageName:String, imageDate:String, imageUri:String, imageSour... int)
+ Image(in:Parcel)
methods
+ writeToParcel(dest:Parcel, flags:int):void
+ describeConte...():int
+ getImageID():int
+ getImageName():String
+ getImageDate():String
+ getImageUri():String
+ getImageSource():int
+ ImageDatabase extends RoomDatab...
fields
- final DB_NAME:String
- instance:ImageDatabase
construct...
methods
+ synchroniz... getInsta... (conte... Context):ImageDatabase
+ destroyInsta... ():void
+ imageDao():ImageDao

Figure 4.4.2.1: Class diagram-1 (Database)

In order to fit the GUI prototypes, I introduced five Activities in the application, which are **SplashActivity**, **StartActivity**, **MainActivity**, **SettingsActivity** and **RealTimeActivity** respectively.

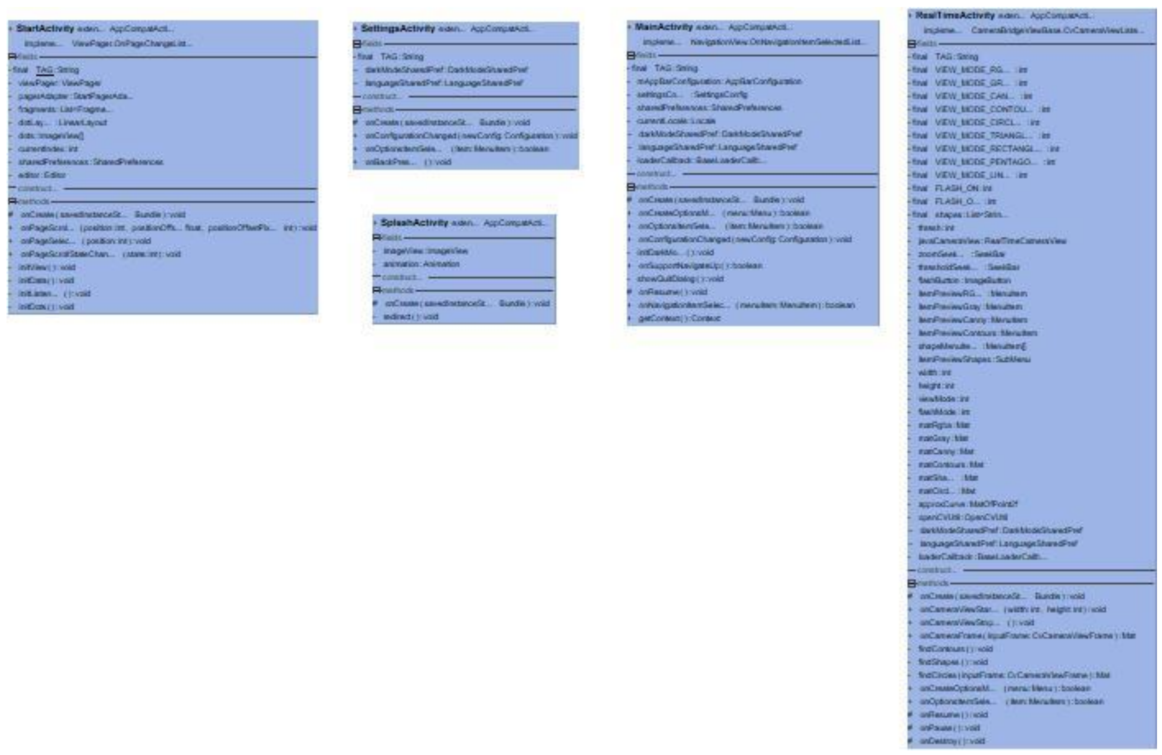


Figure 4.4.2.2: Class diagram-2 (Activities)

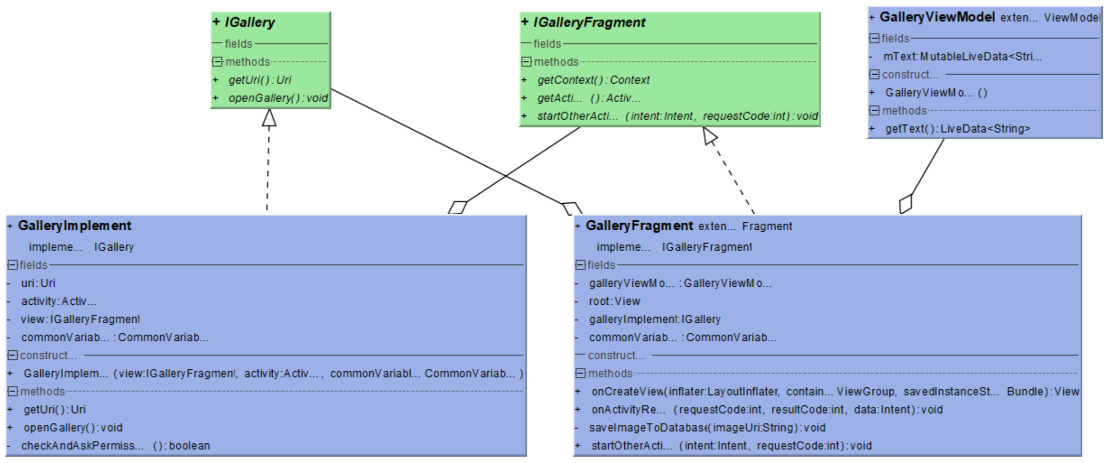

+ IGallery
fields
methods
+ getUri() : Uri
+ openGallery() : void
+ IGalleryFragment
fields
methods
+ getContext() : Context
+ getActi... () : Activ...
+ startOtherActi... (intent:Intent, requestCode:int) : void
+ GalleryViewModel exten... ViewModel
fields
- mText: MutableLiveData<Stri...
construct...
+ GalleryViewMo... ()
methods
+ getText() : LiveData<String>
+ GalleryImplement
impleme... IGallery
fields
- uri: Uri
- activity: Activ...
- view: IGalleryFragment
- commonVariab... : CommonVariab...
construct...
+ GalleryImplem... (view:IGalleryFragment, activity:Activ..., commonVariabl... CommonVariab...)
methods
+ getUri() : Uri
+ openGallery() : void
- checkAndAskPermiss... () : boolean
+ GalleryFragment exten... Fragment
impleme... IGalleryFragment
fields
- galleryViewMo... : GalleryViewMo...
- root: View
- galleryImplement: IGallery
- commonVariab... : CommonVariab...
construct...
methods
+ onCreateView(inflater:LayoutInflater, contain... ViewGroup, savedInstanceSt... Bundle) : View
+ onActivityRe... (requestCode:int, resultCode:int, data:Intent) : void
- saveImageToDatabase(imageUri:String) : void
+ startOtherActi... (intent:Intent, requestCode:int) : void

Figure 4.4.2.3: Class diagram-3 (Gallery part)

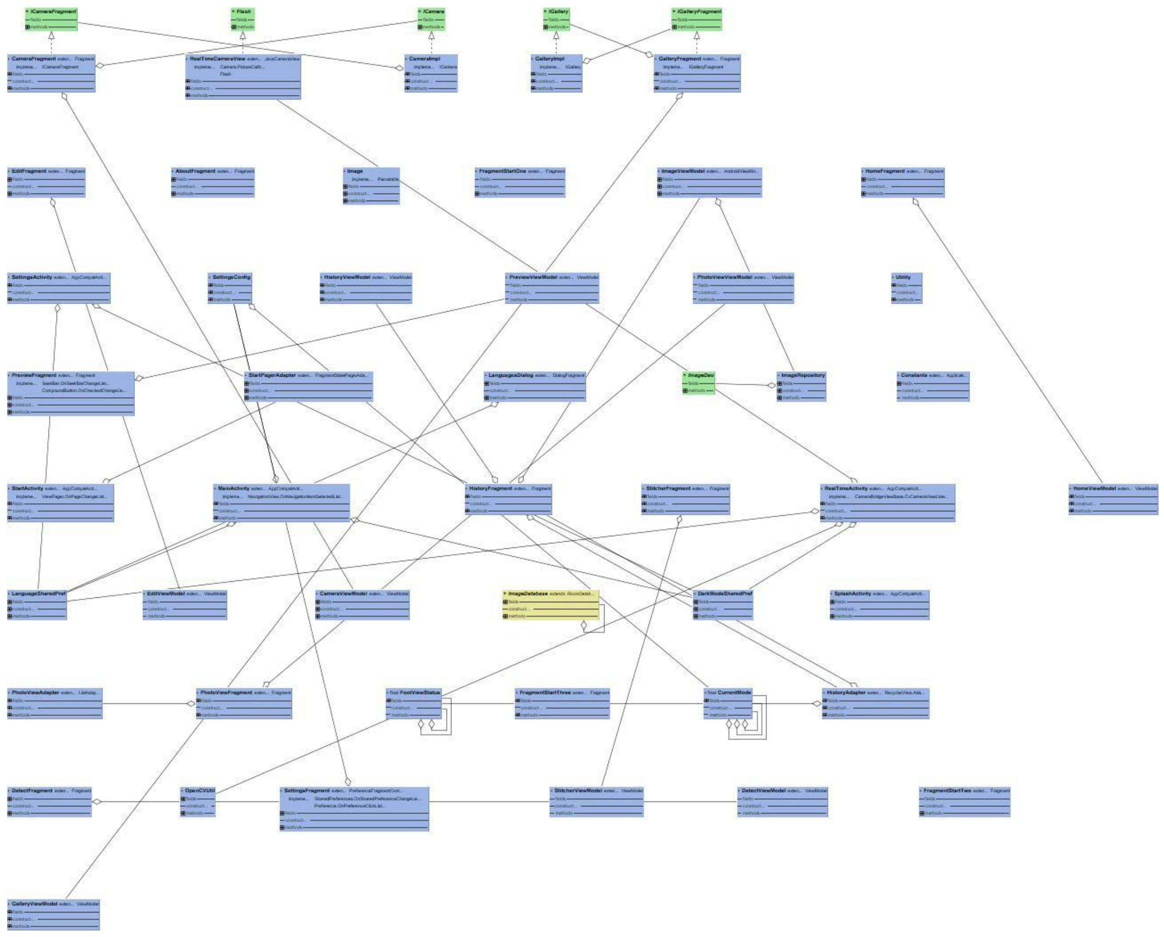


Figure 4.4.2.4: Class diagram overview

Chapter 5: **Implementation**

In this chapter, I will focus on the implementation of the full scope of the project.

5.1 Development Environment Overview

5.1.1 Android Development

Android is a mobile operating system which is led and developed by Google and Open Handset Alliance (OHA). It is originally designed for touch screen mobile devices such as smartphones and tablet computers (ElProCus 2020). The Android mobile OS is a powerful development platform based on the Linux kernel and multi-threading. It has low development costs and is currently the fastest growing mobile platform. The most direct way of Android application development is to use the Android SDK with an IDE (such as Android Studio, Eclipse, etc.) and which can be done on Microsoft Windows, Mac OS X or Linux.

Android application development currently supports Java, Kotlin, C++ and other programming languages. Most of the back-end functionality of this application is implemented in Java, while XML is used for the front-end layout design and general configurations of the project. In addition, the implementation of the image stitcher has also used the Android NDK in order to support C++ programming language.

5.1.2 Justification of Development Environment

As is known, the two mainstream Android app development IDEs on Windows today are Android Studio and Eclipse. In this project, Android Studio ver3.5 is justified as the preferred development environment of the app for the following reasons. Firstly, Android Studio is developed by Google and based on IntelliJ IDEA, which means it can help us complete code completion, refactoring and code analysis very efficiently. Secondly, Android Studio is faster than Eclipse, with integrated version control system (such as Git), Gradle and a variety of powerful plug-in support. And most importantly, Android Studio now is the official Android development IDE promoted by Google Inc.

5.2 Activities and Fragments

Android Activity is one of the four most basic and most commonly used components in Android. It provides an interface to complete the operation instruction for the developers to easily operate and visualize the front-end user interface. In layman's terms, each screen of the app's user interface is an activity, so it is also termed as a view interface.

A Fragment can be used as an independent component by an activity. It is designed to be reusable UI layouts with logic embedded inside of activities. And an activity can be composed of several (*0* - *n*) fragments, each of which has its own independent lifecycle without being constrained by the main activity.

As stated in the previous chapter, the entire application is composed of 5 basic activities, some of which contain multiple fragments. The following sections will give a detailed description of these five activities.

5.2.1 Splash Activity

As is known, at present, many Android apps on the market basically display a splash screen when they are initially launched, which is used as the first transition interface to enter the application and displays a logo information. In addition to providing a better user experience, the main purpose of a splash screen is to reserve time for the program to handle some relatively time-consuming operations, such as data loading, network requests, etc. There is no data need to be loaded in the background of the splash activity in this project, it is designed to provide an interactive experience to the app users.

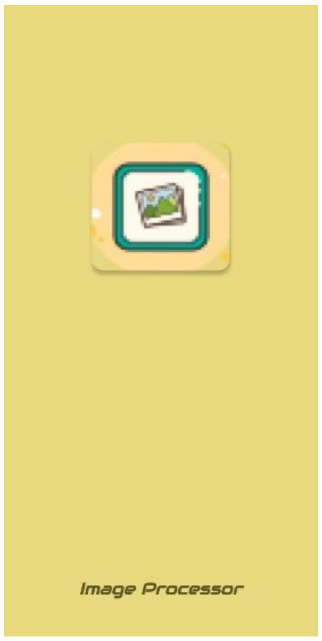


Figure 5.2.1.1: Screenshot of the Splash activity

5.2.2 Start Activity

The Start activity in the application acts as a very basic guide page for the user. It consists of three fragment components; each one contains an independent image resource for display. In order to allow users to swipe through different image resources, ViewPager was introduced. ViewPager is a layout manager that allows the user to slide pages left and right, and manages multiple pages to be displayed through FragmentStatePagerAdapter. It is generally used with fragments to better manage the lifecycle of different pages.

Because the activity plays the role of guiding users how to use the application, the activity will only be displayed when the user uses the application for the first time. When the user restarts the app for the second time, the screen will directly jump to Main Activity instead of being shown twice. This feature is implemented through SharedPreference, that is, when a user accesses for the first time, an attribute is assigned and persisted. Every time this activity is executed, it first determines whether the user is using it for the first time, and then performs the corresponding actions.

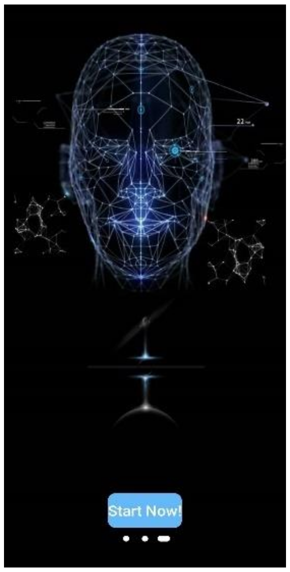


Figure 5.2.2.1: Screenshot of the Start activity

5.2.3 Main Activity

Main Activity is the core activity of this app. It contains almost all the main features except for the real time shape detection and settings. The entire frame of the activity set up with Navigation Drawer (since *Android Studio ver3.2*). Navigation Drawer is a panel that displays available navigation options on the left edge of the screen, which only appears when the user swipes from the left side of the screen or clicks on the top-left drawer icon in the action bar. **Figure 5.2.2.1** below shows the Main Activity with an opened navigation drawer layout.

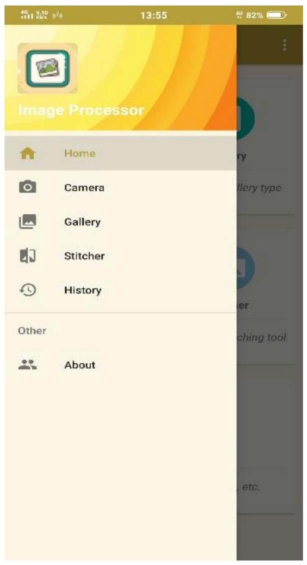


Figure 5.2.2.1: Navigation Drawer in Main Activity

Navigation graph, NavHost and NavController are the three major parts of the navigation component. The figure below illustrates the navigation graph. It records all the fragments of the application and the relationship between them.

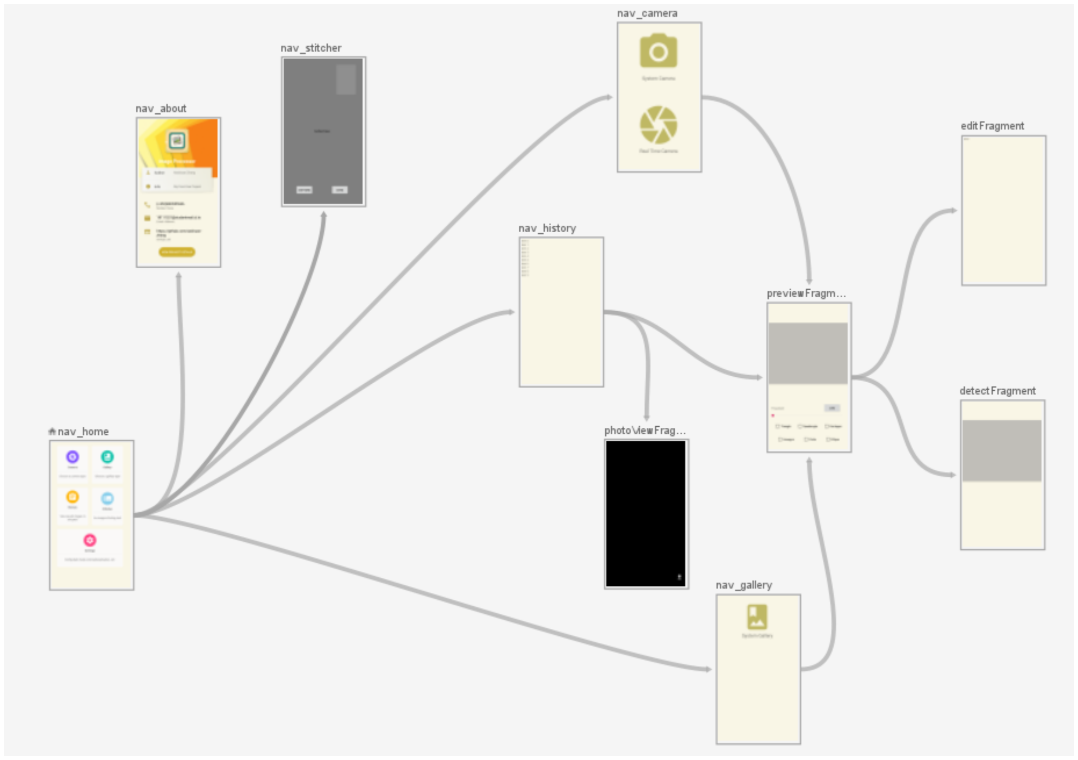


Figure 5.2.3.1: Navigation graph

As we can see from **Figure 5.2.3.1**, there are several fragments can be listed as shown below:

Home is the default fragment of the whole navigation structure, it is connected to all other main fragments. It consists of 5 CardView buttons that can navigate to the corresponding five main pages (*Camera*, *Gallery*, *History*, *Stitcher* and *Settings*) using NavController (*manage navigations between fragment destinations when certain events occur*).

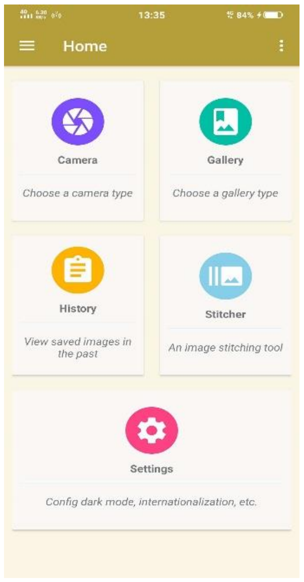


Figure 5.2.3.2: *Home* fragment

Camera fragment has a *system camera* and *real time camera* buttons. When a user selects *system camera*, this fragment will launch a behind-the-scenes activity which loads the default camera on the phone and save the resulting picture to the *History*. This will then navigate to the *preview* fragment to display the result image. Whereas the *real time camera* triggers Real Time Activity which will be detailed later.

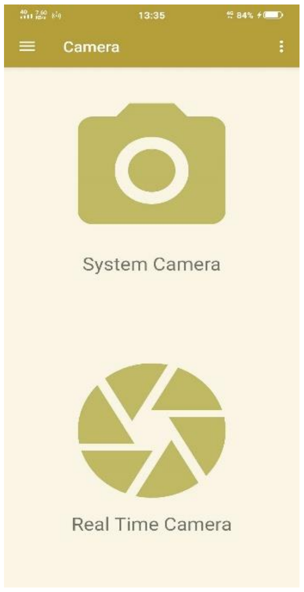


Figure 5.2.3.3: *Camera* fragment

Gallery fragment only has one button which can invoke the system gallery to allow user to pick an image to save into the *history*. And this will also then be navigated to the *preview* fragment with selected image.

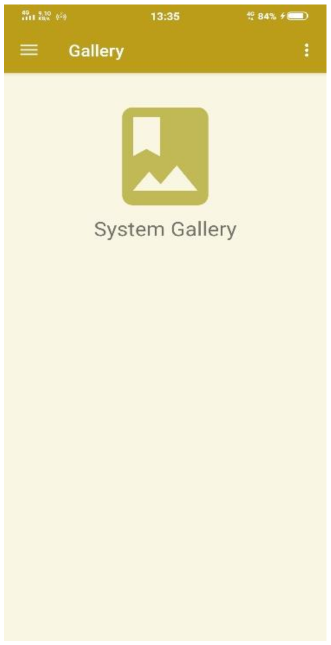


Figure 5.2.3.4: *Gallery* fragment

Preview fragment can only be jumped to indirectly through *camera*, *gallery* and *history* fragments. All of the image processing and shape detection takes place in this fragment. Before doing this, the fragment sets the obtained image URI from bundles and image will be displayed on ImageView. The ImageView can show both the original image passed from system camera/gallery and the processed image result. In addition, a seekbar is also provided to dynamically adjust the threshold parameter of the detection. And there are six checkboxes on the fragment, each checkbox corresponds to a type of geometric shapes to detect. After one checkbox is selected, a processed image with detected results will be refreshed on the image view. For the convenience of checking detection results, users can quickly view the detection records by clicking *LOG* button.

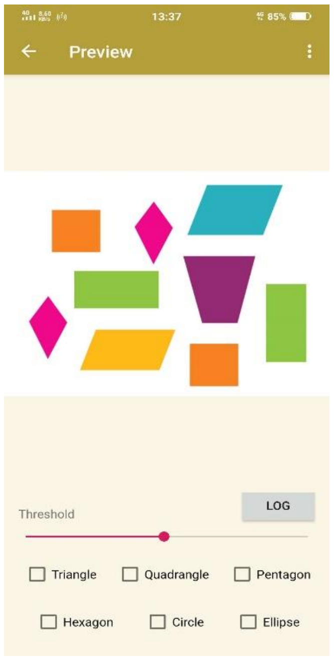


Figure 5.2.3.5: *Preview* fragment

Stitcher fragment implements an image stitching tool. When a user selects the stitcher from the navigation drawer, the built-in system camera will be invoked and a camera view will be displayed. User can capture an image by pressing *CAPTURE* button, if the image is successfully captured, user will see this image in the upper right corner of the screen. User will need to capture at least to pictures in order to generate a stitched image. When the user clicks the *SAVE* button, the stitcher will pop up a processing dialog to prompt the user to wait, after the stitching processes finish, a message will be shown to tell the user whether the stitching has been successfully done.

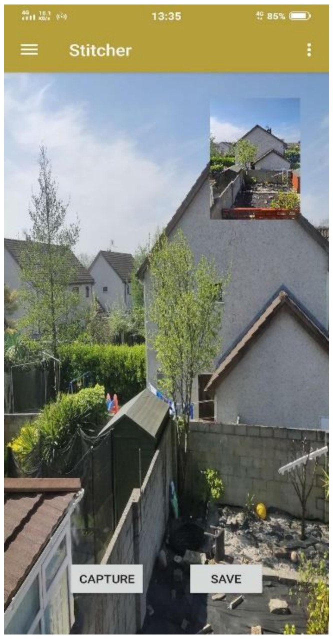


Figure 5.2.3.6: *Stitcher* fragment

History is a fragment used to display and manage the saved images in the past. Images preserved in the history are usually from camera, gallery or stitcher. The whole fragment is made up of RecyclerView. Compared with ListView, RecyclerView is more advanced and flexible. It acts as a container for displaying a large set of data and can be scrolled very effectively by keeping a limited number of views. In this fragment, all recycler views are wrapped in the SwipeRefreshLayout. This layout enables the ability to drop down and refresh history. It is worth mentioning that each cell of history contains an ImageView to display the thumbnail of the image, TextView to display the store date of the image, and a downward arrow to allow users to perform more operations (view image as large, share and delete image). In order to prevent the lag and screen freezing problems might be caused due to the long loading time of the image, Glide was applied. Glide is an Android image loading and caching library that focuses on the smooth loading of a large number of images. In addition, ShimmerLayout was also used as placeholder in ImageView in case of image loading failure. When a user clicks the thumbnail, the app will navigate to preview fragment along with the selected image data (image URI, etc.).

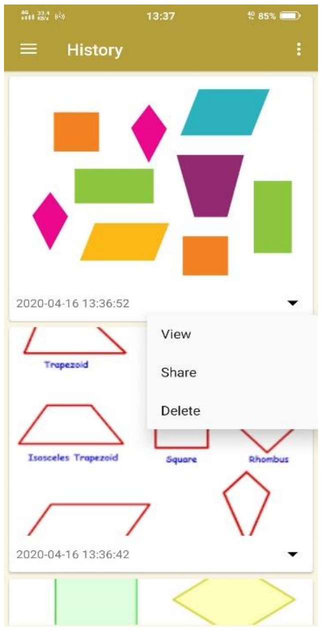


Figure 5.2.3.7: *History* fragment

<u>*View*</u> fragment is called when a user clicks *View* menu item (can be found by clicking the downward arrow in the *history* fragment). The fragment allows the user to display the current image as a large image. Users can also swipe left and right to view all the images saved in the history. The index of the current image is marked at the bottom of the screen. In addition, users can click the download icon in the bottom right corner of the screen to download the image to the local album. The system will then prompt the user if the download was successful. This fragment is implemented by using PhotoView and ViewPager2. As is known, PhotoView is commonly used as an image preview widget in Android. It inherits from ImageView, so it has more advanced features than ImageView, such as photo zooming and rotation with gesture.

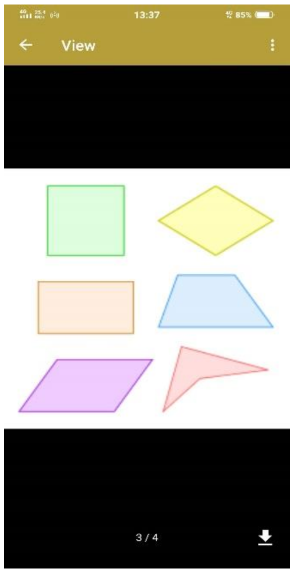


Figure 5.2.3.8: *View* fragment

<u>*About*</u> fragment shows the information (logo and name) of this app and my personal information, including my phone number, email address, GitHub address and a link to the website where this project is located. When user clicks on the phone number, the app opens the dialing interface; when user clicks on the email address, the app opens all third-party apps to send an email; when user clicks on GitHub link, the app opens this webpage in a browser. All these features were implemented with Android Intent.

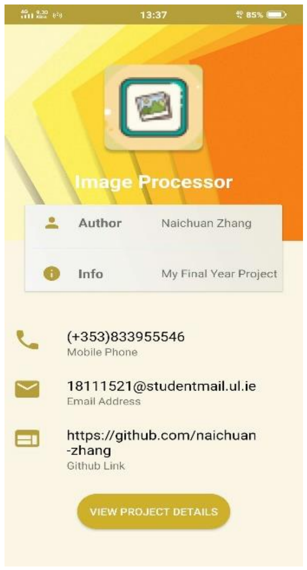


Figure 5.2.3.9: *About* fragment

5.2.4 Real Time Activity

As mentioned previously, this activity gets launched after the user selects the *Real Time Camera* from *camera* fragment. It contains a camera view, which is mainly implemented by JavaCameraView in OpenCV. Instead of directly adding JavaCamereView in the activity's layout, it defines a custom class named RealTimeCameraView. This class inherited from the original JavaCameraView and added a seekbar capable of zooming the camera view and a function of turning on/off the flashlight. On the top left corner of the activity, the OpenCV's current frame rate is shown and updated. Apart from this, in order to enable users to manually adjust the threshold during the real time detection, a seekbar is provided at the top of the screen, and the camera view will also be updated in time. Furthermore, this activity is also equipped with an overflow menu. The camera view can be displayed in multiple forms by selecting different menu items. For example, the camera view can be changed to gray by the user, and Canny can be used to detect all edges in the camera's field of view. Contours can be dynamically displayed by clicking *Contours* menu item. And most importantly, user can also select Detect menu item, this will open a submenu which contains all the available geometric shape to detect. When a user selects a geometric shape, the geometric objects on the camera view that fit this shape will be marked in different colors.

The specific implementation of the real time detection will be discussed later.

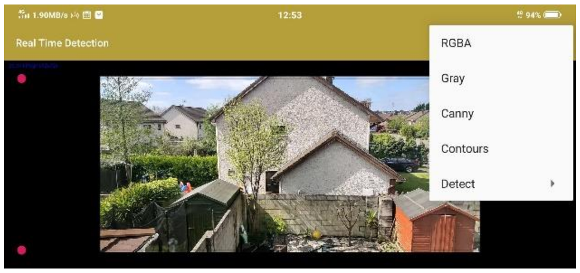
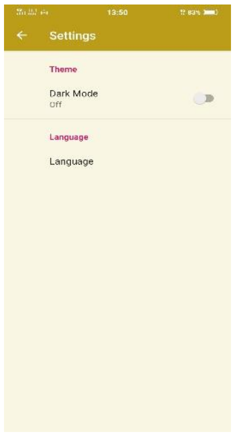


Figure 5.2.4.1: Screenshot of the Real Time activity

5.2.5 Settings Activity

The Settings activity gets called after the user selects *Settings* menu item from the app’s action bar or clicks *Settings* button from home page. This activity contains all the global configurations associated with the application. The current available configurations are dark mode settings and language settings. User can turn on/off the dark theme by clicking the dark mode switch. The app will then automatically change the global color to the relevant mode. And this preference will be preserved when the user reboots the application a second time. And similarly, the language settings currently only support two languages, which are English and Simplified Chinese respectively. When user clicks the *Language* item, a dialog with language lists will be popped up. In addition, the app can detect the language being used on current device (smartphone) to display that language as the default for the application. This user preference will also be preserved by the system.


Settings

Theme

Dark Mode
Off

Language

Language

Figure 5.2.5.1: Screenshot of the Settings activity

5.3 Preprocessing

The preprocessing for this application can be essentially split into two parts. The first one includes distance, angle finding and judgement of parallel. The second part is the image preprocessing. As mentioned in the previous chapter, OpenCV is the main library used for doing image preprocessing. The steps of image preprocessing are crucially important for the subsequent shape detection procedures. The main purpose of image preprocessing is to eliminate irrelevant information in the image to the greatest extent, and highlight the detectability of relevant information and simplify the data on image. Therefore, this section will focus on the preliminary preparation of the project for the realization of shape detection feature.

5.3.1 Distance Finding

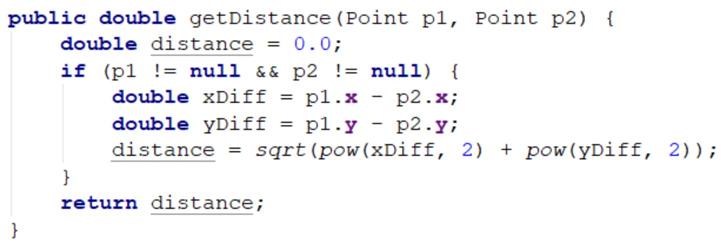
Before preprocessing the image using OpenCV, it is important to figure out how to calculate the distance given two points. This method will be mainly used for subsequent quadrangle detection.

Let’s assume we have two points $p1\ (x1,\ y1)$ and $p2\ (x2,\ y2)$. According to the distance formula derived from the Pythagorean theorem,

$$distance = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

it is easy to calculate the straight-line distance between points $p1$ and $p2$.

The code snippet below (**Figure 5.3.1.1**) illustrates the *getDistance()* method applied in the project. This method takes two points at which the distance to be measured as an incoming parameter and returns the calculated distance result.



```
public double getDistance(Point p1, Point p2) {
    double distance = 0.0;
    if (p1 != null && p2 != null) {
        double xDiff = p1.x - p2.x;
        double yDiff = p1.y - p2.y;
        distance = sqrt(pow(xDiff, 2) + pow(yDiff, 2));
    }
    return distance;
}
```

Figure 5.3.1.1: The distance finding method

5.3.2 Angle Finding

Compared with the distance calculation mentioned earlier, the calculation process of the angle between the two sides is more complicated. This angle finding method is very useful when detecting quadrangles.

Now, we can start by defining three points on a plane, which are $p0\ (x0,\ y0)$, $p1\ (x1,\ y1)$ and $p2\ (x2,\ y2)$ respectively. Point $p0$ will be treated as the vertex point. The graph (**Figure 5.3.2.1**) is roughly as follows,

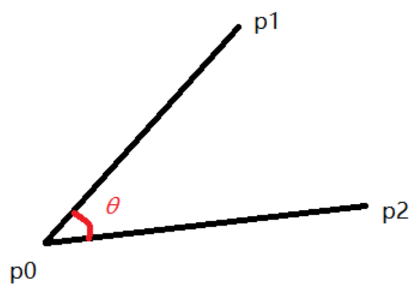


Figure 5.3.2.1: The angle of three points

As shown in the **Figure 5.3.2.1**, we can see $\angle\theta$ is the angle we are about the calculate. According to the formula of the angle between two straight lines:

$$\cos\theta = \frac{\boldsymbol{a_1}\boldsymbol{a_2} + \boldsymbol{b_1}\boldsymbol{b_2}}{\sqrt{(\boldsymbol{a_1}^2 + \boldsymbol{b_1}^2)(\boldsymbol{a_2}^2 + \boldsymbol{b_2}^2)}}$$

, where $\boldsymbol{a_1}$, $\boldsymbol{a_2}$, $\boldsymbol{b_1}$ and $\boldsymbol{b_2}$ are vectors. In this example, $\boldsymbol{a_1} = x1 - x0$, $\boldsymbol{a_2} = x2 - x0$, $\boldsymbol{b_1} = y1 - y0$ and $\boldsymbol{b_2} = y2 - y0$, respectively. Substituting the values into the above formula, we can get the cosine value of $\angle\theta$. However, we are required to calculate the actual angle degree rather than its cosine value, that is, to figure out the value of $\angle\theta$, so we need to introduce inverse trigonometric functions to get the radian first, and then convert the radian to degree of angle. That is,

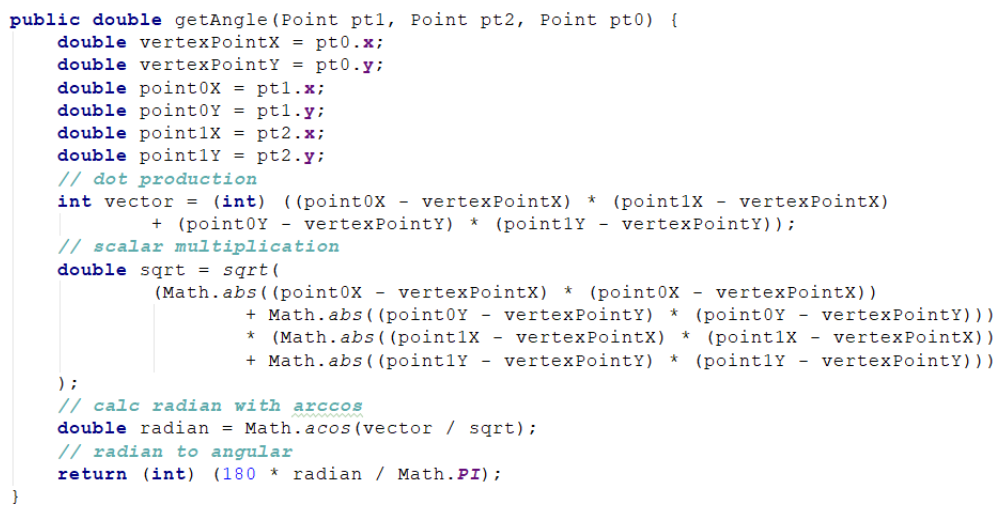
$$\theta = \arccos(\cos\theta)$$

The θ value now we get is in radian system, obviously we still need to convert the radian to a specific degree by the formula below,

$$degree = radian\ \frac{180}{\pi}$$

$degree$ is the actual angle degree that we want.

The code snippet below (**Figure 5.3.2.2**) shows the method in the project to get the angle by applying the above procedures. We can see there are three Point (defined in OpenCV) objects passed as parameters, where *pt0* is regarded as the vertex, so the result returned by this method is the actual degree of ∠*pt1-pt0-pt2*.



```
public double getAngle(Point pt1, Point pt2, Point pt0) {
    double vertexPointX = pt0.x;
    double vertexPointY = pt0.y;
    double point0X = pt1.x;
    double point0Y = pt1.y;
    double point1X = pt2.x;
    double point1Y = pt2.y;
    // dot production
    int vector = (int) ((point0X - vertexPointX) * (point1X - vertexPointX)
            + (point0Y - vertexPointY) * (point1Y - vertexPointY));
    // scalar multiplication
    double sqrt = sqrt(
            (Math.abs((point0X - vertexPointX) * (point0X - vertexPointX))
                    + Math.abs((point0Y - vertexPointY) * (point0Y - vertexPointY)))
                    * (Math.abs((point1X - vertexPointX) * (point1X - vertexPointX))
                    + Math.abs((point1Y - vertexPointY) * (point1Y - vertexPointY)))
    );
    // calc radian with arccos
    double radian = Math.acos(vector / sqrt);
    // radian to angular
    return (int) (180 * radian / Math.PI);
}
```

Figure 5.3.2.2: The angle finding method

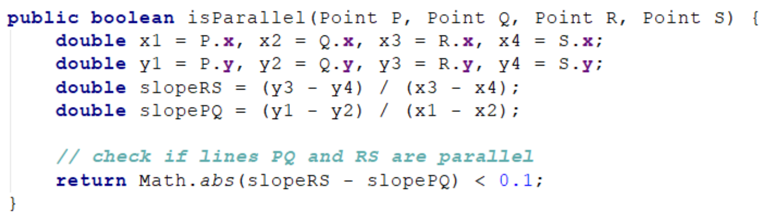
5.3.3 Parallelism Judgement

In order to realize the detection of parallelograms and trapezoids in quadrangles, another problem to be discussed is how to judge the parallelism given two lines. Similar to the previous sections, we define four individual points on a plane, $p\ (x1,\ y1)$, $q\ (x2,\ y2)$, $r\ (x3,\ y3)$ and $s\ (x4,\ y4)$. Now if we want to check if line pq and line rs are parallel to each other, we only need to calculate their slopes by applying formula

$$slope = \frac{y2 - y1}{x2 - x1}$$

After we get the slopes of both these two lines, we can compare if they are equals to each other. If slopes are equal, the two lines then are parallel to each other; otherwise, they are not parallel.

The code snippet below (**Figure 5.3.3.1**) is the detailed implementation of this method.



```
public boolean isParallel(Point P, Point Q, Point R, Point S) {
    double x1 = P.x, x2 = Q.x, x3 = R.x, x4 = S.x;
    double y1 = P.y, y2 = Q.y, y3 = R.y, y4 = S.y;
    double slopeRS = (y3 - y4) / (x3 - x4);
    double slopePQ = (y1 - y2) / (x1 - x2);

    // check if lines PQ and RS are parallel
    return Math.abs(slopeRS - slopePQ) < 0.1;
}
```

Figure 5.3.3.1: The method for checking parallelism

5.3.4 Grayscale and Binarization

So far, we have basically completed the first part of preprocessing. From this section, we will detail the content of the second part - *image preprocessing*.

As stated in the previous research, the first thing during the preprocessing stage is to convert the colored RGB image into grayscale format. Because for some operations such as the Canny edge detection method in OpenCV, the input value of the image data must be in a grayscale format. Therefore, this step can effectively prevent some unnecessary problems in the future. As shown in **Figure 5.3.4.2**, this figure shows the result after a grayscale operation by the original colored image shown in **Figure 5.3.4.1**. As we can see, the grayscale image loses the original color information, but retains the color level.

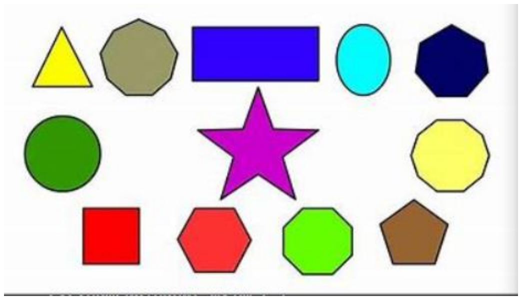


Figure 5.3.4.1: Example – original image (StudentProjectWorks 2015)

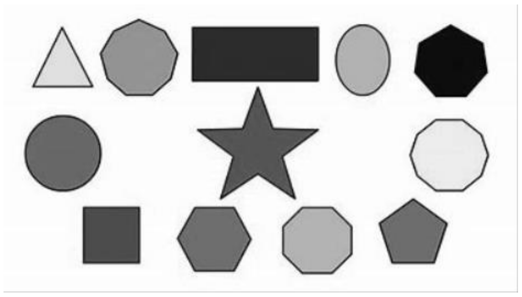


Figure 5.3.4.2: Example – grayscale image

The existence of the grayscale image reduces the amount of unnecessary information contained in the original image and thus increases the operation speed. However, sometimes in order to further increase the operation speed, the use of binary images is also very common. **Figure 5.3.4.3** demonstrates the result image after binarization with a threshold of 100. We can notice that the binary image now only contains black and white.

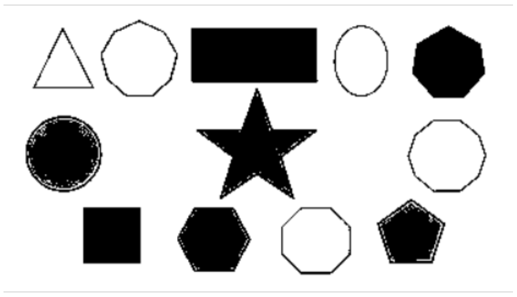
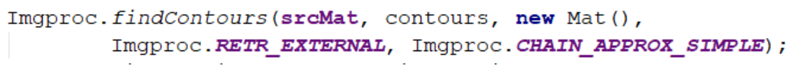


Figure 5.3.4.3: Example – binary image (*threshold* = 100)

5.3.5 Contours Finding

After the previous image binarization process, now we have obtained a binary image. To further improve the detection speed and accuracy, we can use the blurring methods mentioned in our research section to further denoise the binary image. All these operations are done for a better detection result.

Contours finding is one of the most basic but important steps when it comes to shape detection. The following code snippet (**Figure 5.3.5.1**) shows the most important code part to implement contours finding. This *findContours*() method, provided by the OpenCV library, comprises of 5 basic parameters. The *contours* is used to store all the detected contours in a list, *RETR_EXTERNAL* defines a contour retrieval mode which detects only the outermost contours, whereas *CHAIN_APPROX_SIMPLE* represents that only the endpoint coordinates in the same direction are reserved. For example, a rectangle contour only needs four points to store the contour information.



```
Imgproc.findContours(srcMat, contours, new Mat(),
        Imgproc.RETR_EXTERNAL, Imgproc.CHAIN_APPROX_SIMPLE);
```

Figure 5.3.5.1: The contours finding method

Figure 5.3.5.2 demonstrates the output results after executing the contours finding operation described above. We can see all the detected contours are marked in red.

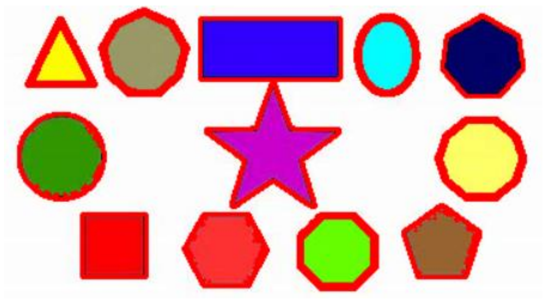


Figure 5.3.5.2: Example – contours finding

5.3.6 Center Finding

Up to now, we have been able to successfully find all the contours present in the image. In order to achieve the feature of shape detection, we need to further find the center points of the detected contours. The purpose of doing this step is to provide convenience for the subsequent circle and ellipse detection.

Before introducing center finding approach, firstly we need to introduce two concepts: one is centroid and the other is image moment. In mathematics, a centroid (geometric center) of a plane figure is the arithmetic mean position of all the points in the figure (Wikipedia 2020). For a geometric figure with a regular shape and uniform mass distribution, if it has more than two axes of symmetry, the centroid can be obtained by making two intersecting straight lines. If it does not have an axis of symmetry, the determination of centroid should be discussed separately. As for a triangle, for instance, where the medians intersect is the centroid. However, although this method of finding centroid is straightforward, it is difficult to be applied in practice, so we have to introduce the concept of image moments. In the field of computer vision, an image moment is a certain particular weighted average (moment) of the image pixels' intensities (Wikipedia 2020). Given the formula below,

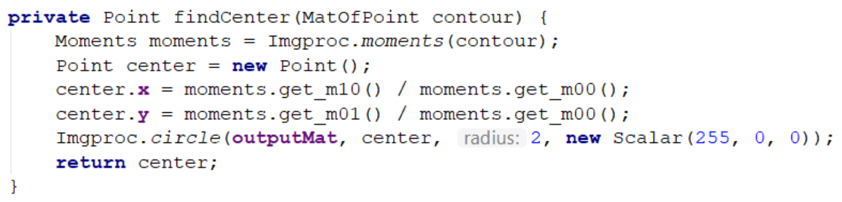
$$m_{m,n} = \sum_{x=0}^{\infty}\sum_{y=0}^{\infty}(x - c_x)^m \, (y - c_y)^n \, f(x, y)$$

(AI Shack 2020)

$f(x, y)$ represents the gray value of the image at point (x, y), since the image here is single channel. We can see that when the image is a binary image, $m_{0,0}$ refers to the sum of all white areas on the image. Then we can simply ignore the constants c_x and c_y. Because for a binary image, the pixel is either 0 (black) or 1 (white). Therefore, $m_{1,0}$ can represent the accumulation of the x coordinate values of all white areas on the image. Then, the 1st order moment can be used to calculate the centroid of the binary image, that is,

$$x = \frac{m_{1,0}}{m_{0,0}} \text{ and } y = \frac{m_{0,1}}{m_{0,0}}$$

Similar to the previous contours finding, the OpenCV library also provides us with a direct way to obtain the center point of a contour. The code snippet in **Figure 5.3.6.1** shows the method adopted for finding center points.



```java
private Point findCenter(MatOfPoint contour) {
    Moments moments = Imgproc.moments(contour);
    Point center = new Point();
    center.x = moments.get_m10() / moments.get_m00();
    center.y = moments.get_m01() / moments.get_m00();
    Imgproc.circle(outputMat, center,  radius: 2, new Scalar(255, 0, 0));
    return center;
}
```

Figure 5.3.6.1: The center finding method

Figure 5.3.6.2 shows a screenshot of an image after performing center finding method. We can see all the detected centers have been circled in red.

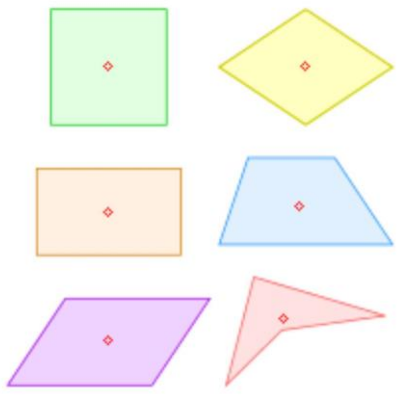
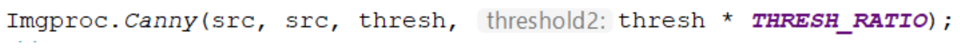


Figure 5.3.6.2: Example – center finding

5.3.7 Threshold Adjustment

The previous sections have basically covered all the preparatory work required for shape detection. As is known, one of the most important criteria for edge detection is that important edges are not lost and there should be no false edges.

When doing image processing, image binarization is a routine operation, because different threshold values will produce different results, so the selection of threshold is critical. In order to facilitate the observation and comparison of the binarization effect of images under different thresholds, I added seekbars to every part of the project that needed shape detection to dynamically adjust the threshold. It is worth mentioning that in the project, Canny edge detector was adopted to process the images to be detected in the early stage. As is shown in **Figure 5.3.7.1**, the Canny method contains four parameters where the last two represent thresholds, these are used to control the upper and lower limits of the threshold. Therefore, I pass the two threshold values as variables that can be adjusted through the seekbar.



```
Imgproc.Canny(src, src, thresh,  threshold2: thresh * THRESH_RATIO);
```

Figure 5.3.7.1: Canny edge detector

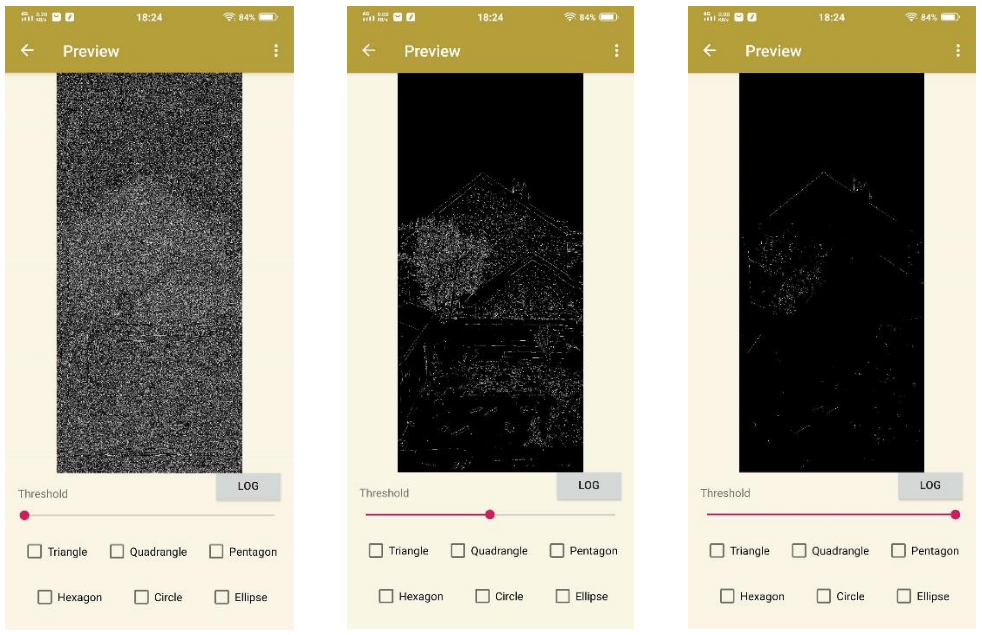


Figure 5.3.7.2: Output results with different thresh values (*thresh* = *1*, *thresh* = *127*, *thresh* = *254*, from left to right)

5.4 Shape Detection

This section will talk about the techniques and algorithms used to detect different geometric shapes.

5.4.1 2D Geometric Shapes Overview

We all know that the application of geometry is ubiquitous in our daily life, and all the objects we see are composed of various geometric shapes. Geometric shapes can be categorized into 2D and 3D geometric shapes according to the dimension. In this project, we are considering 2D shapes, and 3D geometric shapes will not be discussed in this report.

In a 2-dimensional environment, the most basic geometric shapes are circles, triangles, rectangles, etc. All these shapes can be divided into multiple categories according to their unique geometric properties. For example, according to the level of symmetry, they can be divided into regular geometry, semi-regular geometry and irregular geometry. But the most well-known classifications at present is to divide them into two categories based on their property of boundlessness, which are polygons and curves. These categories have been applied in the project, which adopted two distinct approaches in terms of shape detection.

For *curves*, it mainly includes the circle and ellipse we often mentioned. When we continue to refine *polygons*, we can get further divide it into triangle, quadrangle, pentagon, hexagon and many other categories by the number of sides. And these categories can continue to be split into more specific geometric shapes. Therefore, we can summarize a figure (**Figure 5.4.1.1**) contains all types of shapes that will be used for detection.

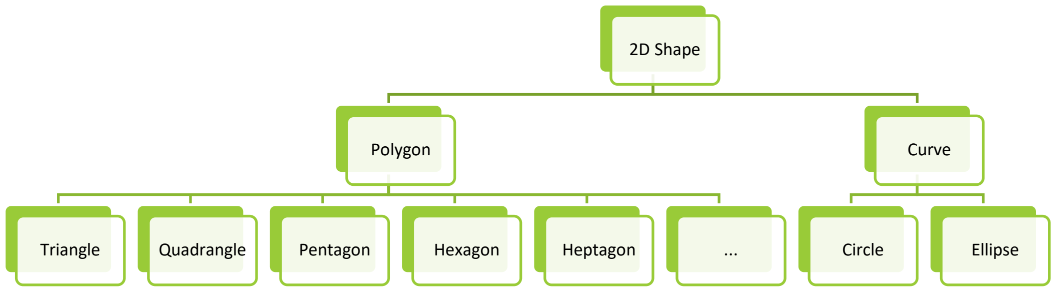


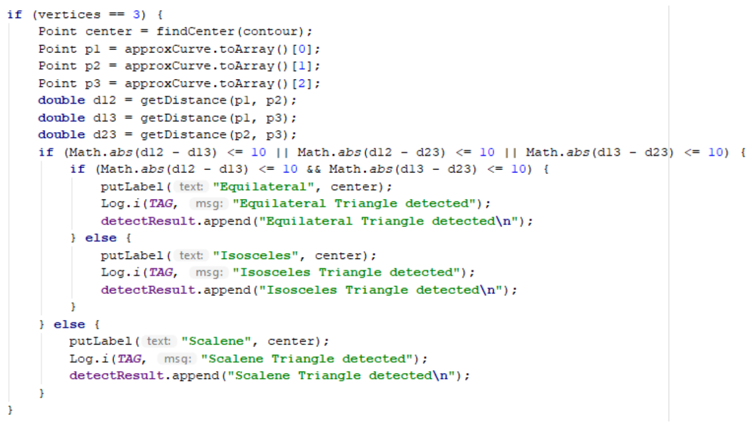
Figure 5.4.1.1: 2D shapes classification

5.4.2 Triangle Detection

We know that triangle is the simplest polygon. A triangle is simply a closed figure consisting of three line segments connected end to end. According to the geometric properties, triangle can be divided by either sides or angles. According to the angle, they can be divided into right triangle, acute triangle and obtuse triangle. In this project, triangles are divided into *equilateral*, *isosceles* and *scalene* by their sides.

If we use the distance finding method mentioned in the previous section, it is not hard to get all three edges' lengths of a triangle. After we get the length of each side, we can direct judge by the following approach: If all the three sides are equal in length, the triangle is equilateral; if only two sides are equal, it is an isosceles triangle; otherwise, it is a scalene triangle.

Figure 5.4.2.1 below shows the code implementation for triangle detection using this approach and **Figure 5.4.2.2** demonstrates an example of detection results.



```java
if (vertices == 3) {
    Point center = findCenter(contour);
    Point p1 = approxCurve.toArray()[0];
    Point p2 = approxCurve.toArray()[1];
    Point p3 = approxCurve.toArray()[2];
    double d12 = getDistance(p1, p2);
    double d13 = getDistance(p1, p3);
    double d23 = getDistance(p2, p3);
    if (Math.abs(d12 - d13) <= 10 || Math.abs(d12 - d23) <= 10 || Math.abs(d13 - d23) <= 10) {
        if (Math.abs(d12 - d13) <= 10 && Math.abs(d13 - d23) <= 10) {
            putLabel( text: "Equilateral", center);
            Log.i(TAG,  msg: "Equilateral Triangle detected");
            detectResult.append("Equilateral Triangle detected\n");
        } else {
            putLabel( text: "Isosceles", center);
            Log.i(TAG,  msg: "Isosceles Triangle detected");
            detectResult.append("Isosceles Triangle detected\n");
        }
    } else {
        putLabel( text: "Scalene", center);
        Log.i(TAG,  msg: "Scalene Triangle detected");
        detectResult.append("Scalene Triangle detected\n");
    }
}
```

Figure 5.4.2.1: Triangle detection implementation

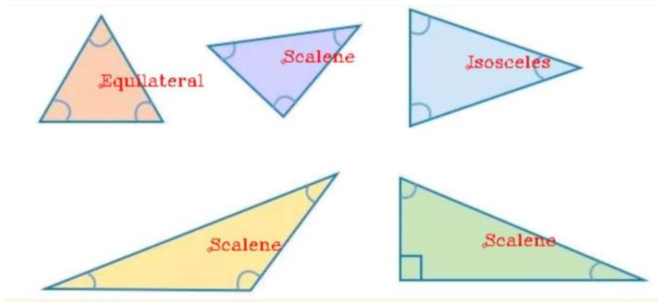


Figure 5.4.2.2: Example – triangle detection

5.4.3 Quadrangle Detection

Quadrangle detection is relatively more complicated compared with triangle. A quadrangle refers to a polygon with four sides and four vertices and the sum of its internal angles is 360 degrees. A general quadrangle can be identified by checking whether the total number of edges is 4. However, we know that quadrangle can continue to be divided into multiple types. In this project, some common quadrangle types will be identified during the shape detection process. Therefore, it is necessary to figure out the different types of quadrangle and the hierarchical relationship between these types. As is shown in **Figure 5.4.3.1**, it shows some types of quadrangles which is considered in the shape detection and also the hierarchical structure between them.

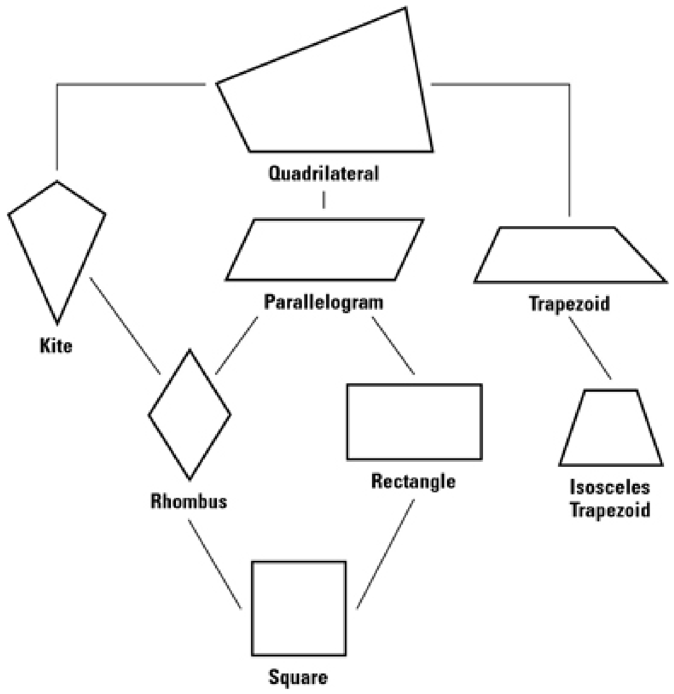


Figure 5.4.3.1: Hierarchical structure of quadrangles (Dummies 2020)

According to the figure above, we can summarize the following approach to identify all those shapes in our project. We can start by calculating the degrees of four angles and the length of the four edges by using the methods shown previously. If the four edges are equal in length and all four angles are 90 degrees, the quadrangle is a *square*; and if the four angles are 90 degrees but only the opposite edges are equal, it is a *rectangle*. All remaining quadrangles have two obtuse angles and two acute angles. Then we can judge the parallelogram, isosceles trapezoid, rhombus and kite by the properties of the angles. Firstly, if there are two pairs of angles that are equal and two pairs of edges are parallel to each other, it is a *parallelogram*; on this basis, if the four edges are equal in length, then it is *rhombus*. In addition, if there are two pairs of equal angles but only one pair of opposite edges are parallel but not equal, it is *isosceles trapezoid*; and if there is only one pair of equal angles with two pairs of equal edges, it is a *kite*. And on this basis, if there are no equal angles but one pair of parallel edges, then it is a *general trapezoid*. Otherwise, if all conditions are not satisfied, it is judged as an *irregular quadrangle*.

The following figure shows an image with the detection results, which used the quadrangle detection methods stated above.

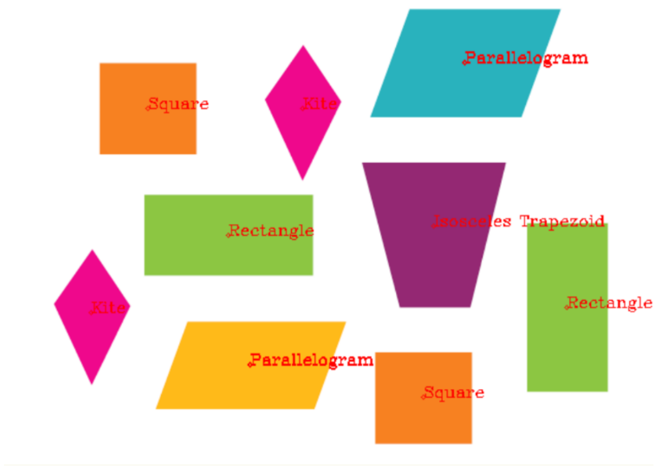


Figure 5.4.3.2: Example – quadrangle detection

5.4.4 Pentagon & Hexagon Detection

The detection process of pentagons and hexagons is relatively simple. From a less strict point of view, in addition to the ordinary pentagon, a pentagon can also be a regular pentagon, where all five sides are equal and all five angles are equal (similar for hexagon detection). In this project, the determination of pentagon and hexagon does not involve regular pentagon and regular hexagon, so the algorithm is to judge the vertex number of the detected contour, that is, if vertex number is 5, then it is a pentagon; if vertex number is 6, then it is a hexagon. **Figure 5.4.4.1** and **Figure 5.4.4.2** illustrate some detection results of pentagons and hexagons, respectively.

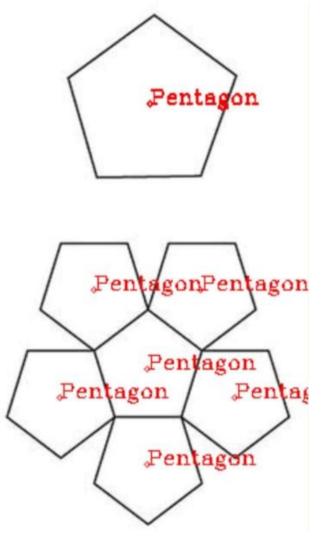


Figure 5.4.4.1: Example – pentagon detection

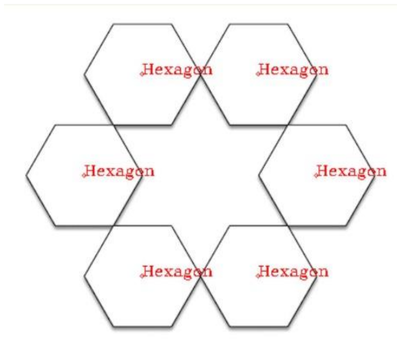


Figure 5.4.4.2: Example – hexagon detection

5.4.5 Circle Detection

Compared with the previous polygon detection methods, the detection methods of circle and ellipse are different because they both belong to curves. First, let's start with the geometric property of the circle. A circle can be defined as a two-dimensional shape made by drawing a curve with the same distance from the center (MathIsFun 2018).

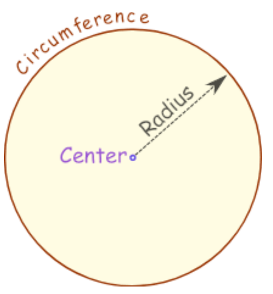
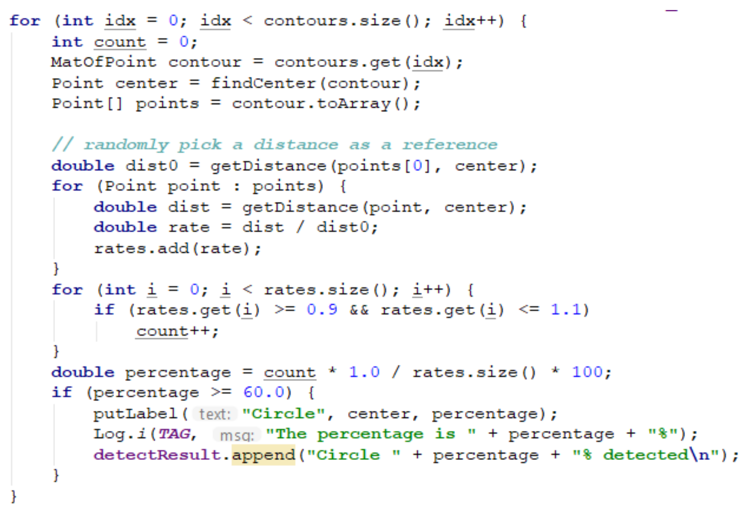


Figure 5.4.5.1: circle (MathIsFun 2018)

In this project, a method of voting statistics has been adopted in circle detection. The detailed implementation can be summarized as follows: the first thing we need to do is to get a contour with OpenCV, then we can use the center finding approach (this has been described in previous section) to get the center of this contour. This contour we get is made up of many points. We pick the first point saved in the contour as a reference. According to the property of circles, we can easily get the distance between the center point to every single point on the contour and also we need to define a threshold to evaluate the possibility of a circle. Finally, we count all the threshold values that satisfy a specific range and then an approximate rate that the contour is circle can be calculated. **Figure 5.4.5.2** illustrates the code snippet of detecting circles.



```
for (int idx = 0; idx < contours.size(); idx++) {
    int count = 0;
    MatOfPoint contour = contours.get(idx);
    Point center = findCenter(contour);
    Point[] points = contour.toArray();

    // randomly pick a distance as a reference
    double dist0 = getDistance(points[0], center);
    for (Point point : points) {
        double dist = getDistance(point, center);
        double rate = dist / dist0;
        rates.add(rate);
    }
    for (int i = 0; i < rates.size(); i++) {
        if (rates.get(i) >= 0.9 && rates.get(i) <= 1.1)
            count++;
    }
    double percentage = count * 1.0 / rates.size() * 100;
    if (percentage >= 60.0) {
        putLabel( text: "Circle", center, percentage);
        Log.i(TAG,  msg: "The percentage is " + percentage + "%");
        detectResult.append("Circle " + percentage + "% detected\n");
    }
}
```

Figure 5.4.5.2: Circle detection implementation

In addition, as we mentioned earlier in the research section, in OpenCV, it provides a method for detecting circles with Hough circles. However, this method has certain limitations compared with the method we just described. First, Hough circle can only be used to detect standard circles. When we want to detect some circles that are not standard (such as hand-drawn circles), it will detect many non-existent circles, so we can see its detection results can be easily interfered to some extent.

Figure 5.4.5.3 and **Figure 5.4.5.4** show the detection results under two different conditions respectively. We can see in **Figure 5.4.5.3**, all circles are regular circles, so the detection results are 100% circles. **Figure 5.4.5.4**, on the other hand, contains three hand-drawn circles with different probabilities. Among them, one contour is not marked as a circle, because the probability of this contour is less than 60%.

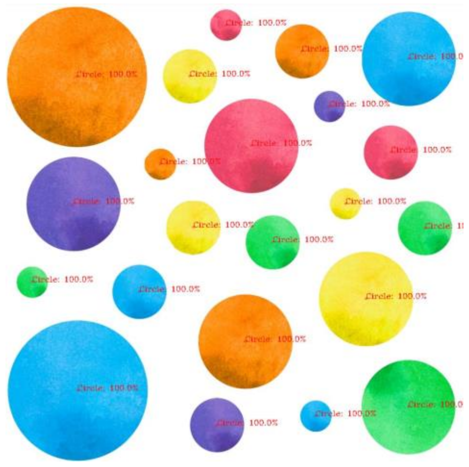


Figure 5.4.5.3: Example 1 – circle detection

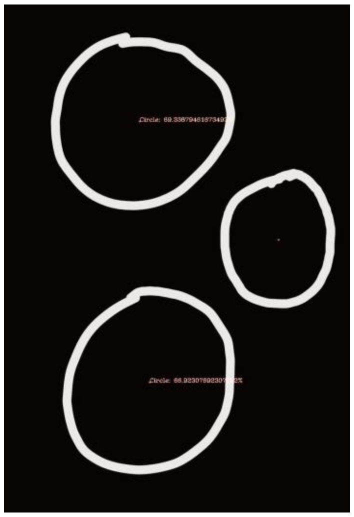


Figure 5.4.5.4: Example 2 – circle detection

5.4.6 Ellipse Detection

Similar to the circle detection, a method of calculating probability has also been adopted in ellipse detection of my project. Ellipse can be regarded as the trajectory of the moving point P in a plane where the sum of the distances to fixed points F1 and F2 is equal to a constant. And F1 and F2 are known as the two focal points of the ellipse (**Figure 5.4.6.1**).

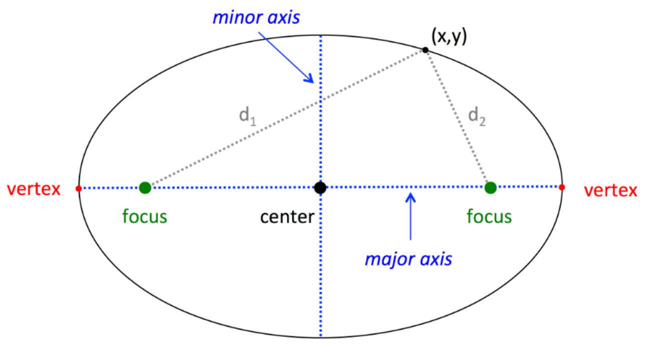


Figure 5.4.6.1: ellipse (SoftSchools 2020)

The following are the specific steps of implementing ellipse detection in my project: firstly, we need to get center point of the contour and all distances between the center point and every single point on the contour using the same approaches mentioned in circle detection part. We have to save all the distances into an array. After that, we get the maximum distance and the minimum distance as the lengths of semi-major axis and semi-minor axis respectively. In addition, we have to know the positions of two focal points. According to the formula

$$dist = \sqrt{l^2 - s^2}$$

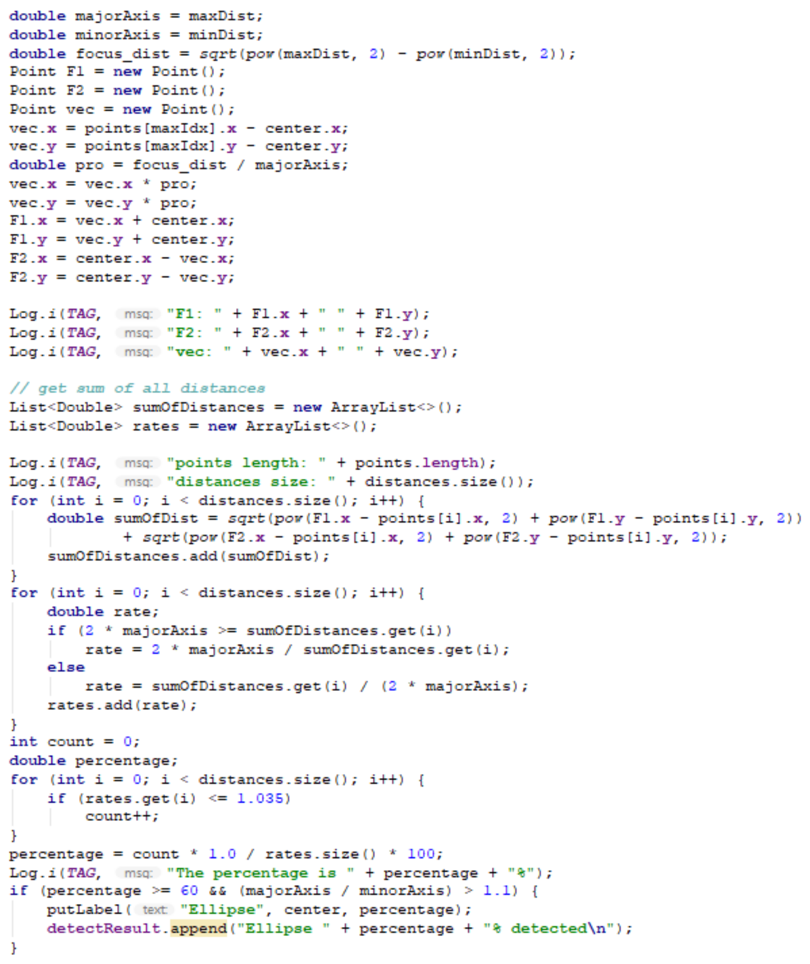
, where l is the semi-major axis, s is the semi-minor axis and $dist$ is the distance from each focal point to the center. With the aid of the formula, we can get the positions of the two focal points. After that, we have to calculate the sum of distances from a point on the contour to the fixed points F1 and F2 and save them into an array by using the formula

$$|PF1| + |PF2| = 2a \ (2a > |F1F2|)$$

, where $2a$ is what we need to calculate. Finally, we can set a threshold and get the final probability, this is similar to the circle detection.

Figure 5.4.6.2 demonstrates the core implementation of ellipse detection.

Figure 5.4.6.3 and **Figure 5.4.6.4** shows two detection results under two different circumstances (regular ellipse detection and hand-drawn ellipse detection).



```java
double majorAxis = maxDist;
double minorAxis = minDist;
double focus_dist = sqrt(pow(maxDist, 2) - pow(minDist, 2));
Point F1 = new Point();
Point F2 = new Point();
Point vec = new Point();
vec.x = points[maxIdx].x - center.x;
vec.y = points[maxIdx].y - center.y;
double pro = focus_dist / majorAxis;
vec.x = vec.x * pro;
vec.y = vec.y * pro;
F1.x = vec.x + center.x;
F1.y = vec.y + center.y;
F2.x = center.x - vec.x;
F2.y = center.y - vec.y;

Log.i(TAG,  msg: "F1: " + F1.x + " " + F1.y);
Log.i(TAG,  msg: "F2: " + F2.x + " " + F2.y);
Log.i(TAG,  msg: "vec: " + vec.x + " " + vec.y);

// get sum of all distances
List<Double> sumOfDistances = new ArrayList<>();
List<Double> rates = new ArrayList<>();

Log.i(TAG,  msg: "points length: " + points.length);
Log.i(TAG,  msg: "distances size: " + distances.size());
for (int i = 0; i < distances.size(); i++) {
    double sumOfDist = sqrt(pow(F1.x - points[i].x, 2) + pow(F1.y - points[i].y, 2))
            + sqrt(pow(F2.x - points[i].x, 2) + pow(F2.y - points[i].y, 2));
    sumOfDistances.add(sumOfDist);
}
for (int i = 0; i < distances.size(); i++) {
    double rate;
    if (2 * majorAxis >= sumOfDistances.get(i))
        rate = 2 * majorAxis / sumOfDistances.get(i);
    else
        rate = sumOfDistances.get(i) / (2 * majorAxis);
    rates.add(rate);
}
int count = 0;
double percentage;
for (int i = 0; i < distances.size(); i++) {
    if (rates.get(i) <= 1.035)
        count++;
}
percentage = count * 1.0 / rates.size() * 100;
Log.i(TAG,  msg: "The percentage is " + percentage + "%");
if (percentage >= 60 && (majorAxis / minorAxis) > 1.1) {
    putLabel( text: "Ellipse", center, percentage);
    detectResult.append("Ellipse " + percentage + "% detected\n");
}
```

Figure 5.4.6.2: Ellipse detection implementation (core)

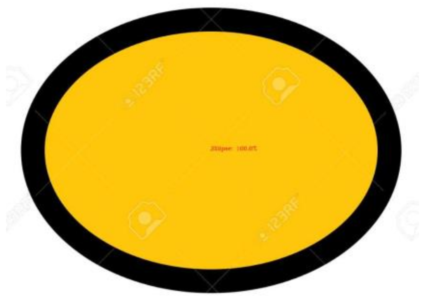


Figure 5.4.6.3: Example 1 – ellipse detection

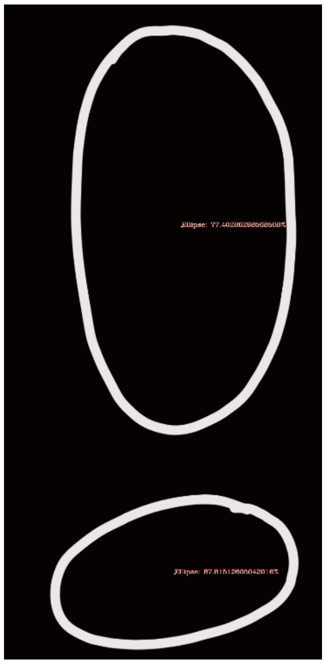


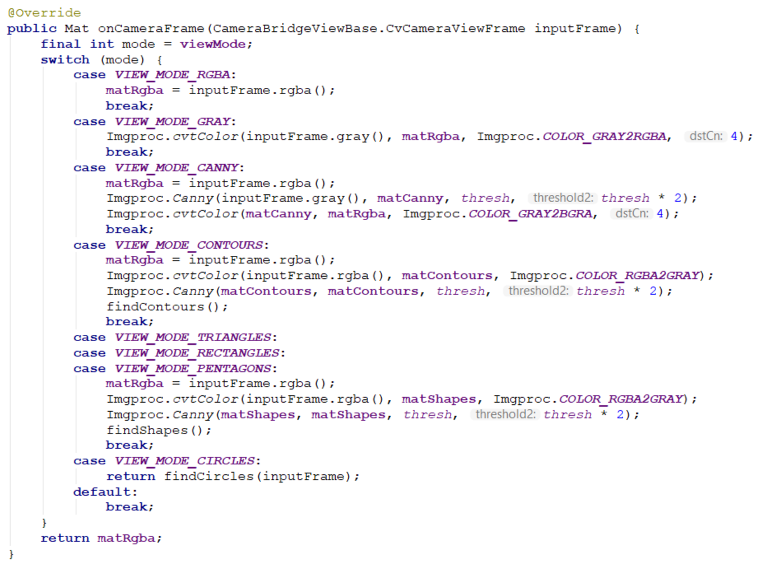
Figure 5.4.6.4: Example 2 – ellipse detection

5.5 Real-Time Shape Detection

In this project, real-time shape detection feature has also been achieved. Generally speaking, the algorithms used in real-time shape detection is roughly similar to the shape detection algorithms described in the previous section. Therefore, in this section, we will not focus on the shape detection part.

The main different between real-time detection and previous static shape detection is that the implementation of this part is based on *JavaCameraView*. *JavaCameraView* is a class which is implemented inside OpenCV library. It is inherited from *CameraBridgeViewBase*, that extends *SurfaceView* and uses standard Android camera API (OpenCV 2020). With the aid of this tool, we can easily let the program capture preview frames from the camera in real time. This *RealTimeActivity* implements the *CvCameraViewListener* interface, which allows the activity to subscribe to the *onCameraFrame* callback. As we can see from

Figure 5.5.1, each frame is passed as a parameter by the *onCameraFrame* method, and we can return a processed *Mat* object as the return value.



```
@Override
public Mat onCameraFrame(CameraBridgeViewBase.CvCameraViewFrame inputFrame) {
    final int mode = viewMode;
    switch (mode) {
        case VIEW_MODE_RGBA:
            matRgba = inputFrame.rgba();
            break;
        case VIEW_MODE_GRAY:
            Imgproc.cvtColor(inputFrame.gray(), matRgba, Imgproc.COLOR_GRAY2RGBA,  dstCn: 4);
            break;
        case VIEW_MODE_CANNY:
            matRgba = inputFrame.rgba();
            Imgproc.Canny(inputFrame.gray(), matCanny, thresh,  threshold2: thresh * 2);
            Imgproc.cvtColor(matCanny, matRgba, Imgproc.COLOR_GRAY2BGRA,  dstCn: 4);
            break;
        case VIEW_MODE_CONTOURS:
            matRgba = inputFrame.rgba();
            Imgproc.cvtColor(inputFrame.rgba(), matContours, Imgproc.COLOR_RGBA2GRAY);
            Imgproc.Canny(matContours, matContours, thresh,  threshold2: thresh * 2);
            findContours();
            break;
        case VIEW_MODE_TRIANGLES:
        case VIEW_MODE_RECTANGLES:
        case VIEW_MODE_PENTAGONS:
            matRgba = inputFrame.rgba();
            Imgproc.cvtColor(inputFrame.rgba(), matShapes, Imgproc.COLOR_RGBA2GRAY);
            Imgproc.Canny(matShapes, matShapes, thresh,  threshold2: thresh * 2);
            findShapes();
            break;
        case VIEW_MODE_CIRCLES:
            return findCircles(inputFrame);
        default:
            break;
    }
    return matRgba;
}
```

Figure 5.5.1: Real-time shape detection implementation (core)

As mentioned before, this project does not directly use the *JavaCameraView* provided by OpenCV, but redefines a subclass inherited from *JavaCameraView* called *RealTimeCameraView* (**Figure 5.5.2**). This class supports zooming feature and flashlight. Since the implementation of JavaCameraView uses the camera API provided by Android, we can obtain the camera parameters and configure them accordingly to achieve zooming and flashlight support.

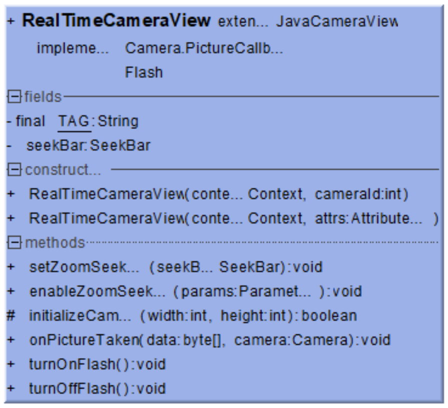

+ RealTimeCameraView exten... JavaCameraView
impleme... Camera.PictureCallb...
Flash
fields
- final TAG:String
- seekBar:SeekBar
construct...
+ RealTimeCameraView(conte... Context, cameraId:int)
+ RealTimeCameraView(conte... Context, attrs:Attribute...)
methods
+ setZoomSeek... (seekB... SeekBar):void
+ enableZoomSeek... (params:Paramet...):void
# initializeCam... (width:int, height:int):boolean
+ onPictureTaken(data:byte[], camera:Camera):void
+ turnOnFlash():void
+ turnOffFlash():void

Figure 5.5.2: *RealTimeCameraView* class

5.6 Image Stitcher

Image stitcher is another important feature in this application. So in this section, I will talk about this image stitching tool.

5.6.1 Environment Preparation

In this project, all the functions related to OpenCV can be completed directly through Java. but for image stitching, we must first introduce Android NDK support as the stitching module is unavailable in OpenCV's Java SDK. The environment preparation for this part is crucially important. Since I personally didn't know much about the Android NDK development, I encountered many obstacles in the implementation of this part, especially in the early configuration.

In my project, I used *CMakeLists.txt* configuration file for NDK development. When we use Android Native to write projects, the mainstream approach now is to use *CMakeLists.txt* for project configuration. To put it simply, CMake is a cross-platform build tool that supports project set-up with multiple languages, including C/C++/Java, while *CMakeLists.txt* is a CMake build script.

Figure 5.6.1.1 shows a part of the CMake configuration for OpenCV with OpenCV stitching module.

```
set(lib_src_DIR ${CMAKE_SOURCE_DIR}/../jniLibs/${ANDROID_ABI}/libopencv_java3.so)
include_directories(${CMAKE_SOURCE_DIR}/include)
add_library(
        opencv_java3-lib
        SHARED
        IMPORTED)
set_target_properties(
        opencv_java3-lib
        PROPERTIES IMPORTED_LOCATION
        ${lib_src_DIR})

add_library( # Sets the name of the library.
        native-lib

        # Sets the library as a shared library.
        SHARED

        # Provides a relative path to your source file(s).
        native-lib.cpp

        # TODO: add here ...
        )

# Searches for a specified prebuilt library and stores the path as a
# variable. Because CMake includes system libraries in the search path by
# default, you only need to specify the name of the public NDK library
# you want to add. CMake verifies that the library exists before
# completing its build.

find_library( # Sets the name of the path variable.
        log-lib

        # Specifies the name of the NDK library that
        # you want CMake to locate.
        log)

file(GLOB CVLIBS
        I:/opencv-3.4.8-android-sdk/OpenCV-android-sdk/sdk/native/staticlibs/${ANDROID_ABI}/*.a)
```

Figure 5.6.1.1: *CMakeLists.txt* file (core)

In addition to using NDK to make the project support C++ code, we also need to consider how to realize the interaction between Java and C++ languages. Java Native Interface (aka JNI) can be applied. In order to achieve the interaction with JNI, we need to declare a method with *native* keyword in Java file and implement this method in C++. **Figure 5.6.1.2** below shows the declaration of a native method which implements the image stitching feature.

```
public native int stitchImages(Object[] images, int size, long addrSrcRes);
```

Figure 5.6.1.2: *stitchImages()* native method

5.6.2 Image Stitching Process Overview

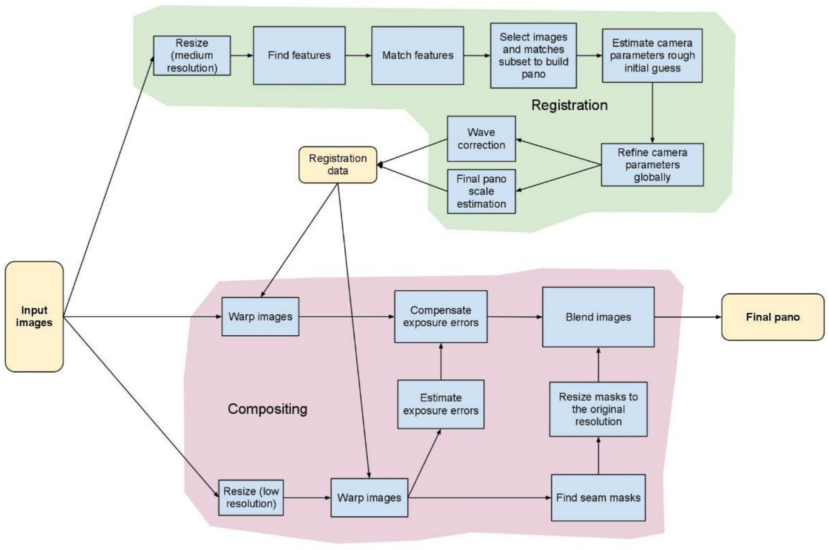

Resize (medium resolution)
Find features
Match features
Select images and matches subset to build pano
Estimate camera parameters rough initial guess
Registration
Wave correction
Refine camera parameters globally
Registration data
Final pano scale estimation
Input images
Warp images
Compensate exposure errors
Blend images
Final pano
Compositing
Estimate exposure errors
Resize masks to the original resolution
Resize (low resolution)
Warp images
Find seam masks

Figure 5.6.2.1: Image stitching pipeline (OpenCV 2020)

Figure 5.6.2.1 above shows a basic image stitching pipeline provided by OpenCV. As we can see, the image stitching algorithm generally consists of two basic parts: image registration and image composition. More specifically, the steps of the algorithm can be summarized as follows:

1) Resize images to medium resolution. This step should be done before other stitching algorithms are executed. The main purpose of this step is to increase the execution speed of image stitching process. We usually resize a high-resolution image to medium resolution since some modules like *SeamFinder* and *ExposureCompensator* in OpenCV will take a long time when working with the original image.
2) Features detection. Features are specific patterns that are unique and can be easily tracked and compared (Kapur and Thakkar 2015). Image feature matching in computer vision is based on the features, so how to find the feature points in an image is very important. In the stitching module, ORB and SURF are the two algorithms to perform features detection. compared with SURF, ORB is faster in speed but has a lower robustness.

3) Features matching. Once features have been extracted from all images, each feature is matched to its k-nearest neighbors in feature space. This is usually done by using a k-d tree to find approximate nearest neighbors. The goal of this step is to find all matching images, since each image could potentially match every other one. (Basic stitching pipeline n.d.)
4) Sort the images and save images with high confidence in the same set, delete the matching between the images with lower confidence, so that we can get he image sequence that can be matched correctly. By doing this, all the matches whose confidence is higher than the threshold are combined into one set.
5) Roughly estimate camera parameters for all images, then find the rotation matrices.
6) Bundle adjustment. The purpose of this step is to estimate the rotation matrices more accurately with a bundle adjuster, which can ensure a better image alignment result.
7) Wave correction. The images have been well-aligned by the previous step. However, the adjustment only gives us a relative rotation between each matched pair, the 3D rotation to a chosen world coordinate is assumed to be the unity matrices. This will lead to a wavy effect in the final stitched image. (Basic stitching pipeline n.d.) This step can be applied to avoid this wavy effect.
8) Image Warping. In OpenCV, there are both cylindrical and spherical image warping methods. Once surface mapping is done, the overlapping pixels between each pair can be calculated. (Basic stitching pipeline n.d.)
9) Exposure Compensation. This step is used to solve the different in illumination of different parts of the output image caused by different exposures of different images.
10) Seam Finding. Object motion and spatial alignment errors will cause ghosting artifacts if the images are composite sequentially. Therefore, an optimal seam finding method is required to find seams in overlapping area of source images. (Basic stitching pipeline n.d.)

11) Image blending. After the seam is found in the previous step, OpenCV's *MultiBandBlender* can be used to blend the overlapping parts between different images. And finally, we can get the final stitched image.

5.6.3 Image Stitching

Since there are many algorithms used for image stitching and the contents are also relatively complex, OpenCV encapsulates all the stitching algorithms into a *Stitcher* class, so that we don't need to pay more attention to the specific implementation process in the stitching algorithm. **Figure 5.6.3.1** shows the pipeline for implementing image stitching with the provided *Stitcher* class in OpenCV.

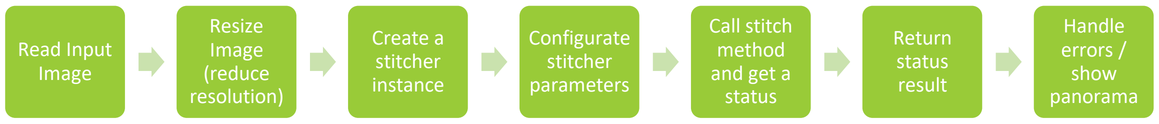
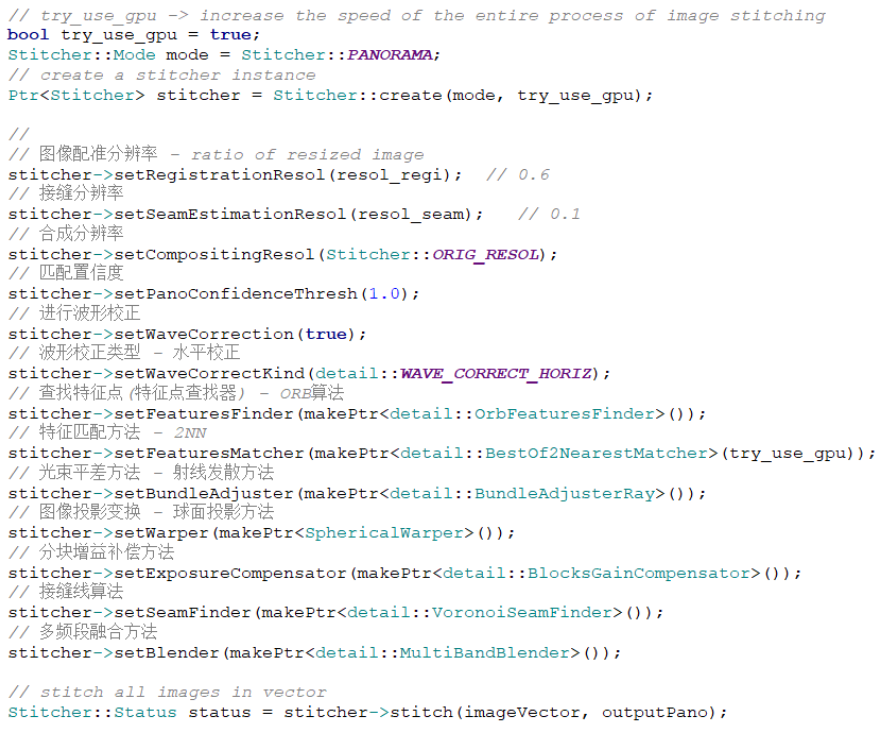

Read Input Image
Resize Image (reduce resolution)
Create a stitcher instance
Configurate stitcher parameters
Call stitch method and get a status
Return status result
Handle errors / show panorama

Figure 5.6.3.1: Pipeline for image stitching implementation

Figure 5.6.3.2 shows the core implementation of image stitching feature in C++.



```
// try_use_gpu -> increase the speed of the entire process of image stitching
bool try_use_gpu = true;
Stitcher::Mode mode = Stitcher::PANORAMA;
// create a stitcher instance
Ptr<Stitcher> stitcher = Stitcher::create(mode, try_use_gpu);

//
// 图像配准分辨率 - ratio of resized image
stitcher->setRegistrationResol(resol_regi);  // 0.6
// 接缝分辨率
stitcher->setSeamEstimationResol(resol_seam);   // 0.1
// 合成分辨率
stitcher->setCompositingResol(Stitcher::ORIG_RESOL);
// 匹配置信度
stitcher->setPanoConfidenceThresh(1.0);
// 进行波形校正
stitcher->setWaveCorrection(true);
// 波形校正类型 - 水平校正
stitcher->setWaveCorrectKind(detail::WAVE_CORRECT_HORIZ);
// 查找特征点(特征点查找器) - ORB算法
stitcher->setFeaturesFinder(makePtr<detail::OrbFeaturesFinder>());
// 特征匹配方法 - 2NN
stitcher->setFeaturesMatcher(makePtr<detail::BestOf2NearestMatcher>(try_use_gpu));
// 光束平差方法 - 射线发散方法
stitcher->setBundleAdjuster(makePtr<detail::BundleAdjusterRay>());
// 图像投影变换 - 球面投影方法
stitcher->setWarper(makePtr<SphericalWarper>());
// 分块增益补偿方法
stitcher->setExposureCompensator(makePtr<detail::BlocksGainCompensator>());
// 接缝线算法
stitcher->setSeamFinder(makePtr<detail::VoronoiSeamFinder>());
// 多频段融合方法
stitcher->setBlender(makePtr<detail::MultiBandBlender>());

// stitch all images in vector
Stitcher::Status status = stitcher->stitch(imageVector, outputPano);
```

Figure 5.6.3.2: Core implementation of image stitching in C++

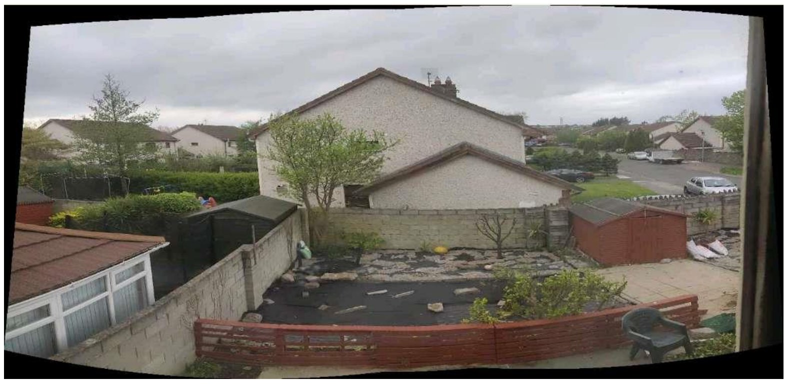


Figure 5.6.3.3: Example – image stitcher

5.7 Settings

The settings feature is the last thing I want to discuss in this chapter. As mentioned previously, the settings screen can be opened by clicking *Settings* menu item from the app's action bar or be selected through home screen. This section includes two parts, one is the implementation of dark mode feature, and the other is about the internationalization.

5.7.1 Dark Mode

As we know, a good mobile app should not be only limited to the realization of functional requirements. How to make a good user experience is equally important as for an app developer. A night mode configuration now has become a standard for almost every single mobile app. In my application, users can turn on/off the dark mode by pressing the *Dark Mode* switch and the entire app will be refreshed immediately to the current mode (**Figure 5.7.1.1**). This settings screen is implemented with a preference fragment which is inherited from *PreferenceFragmentCompat* class. Unlike a typical fragment, the layout of the preference fragment is built by various subclasses of the Preference class declared in the XML file, rather than using the View object to build the user interface.

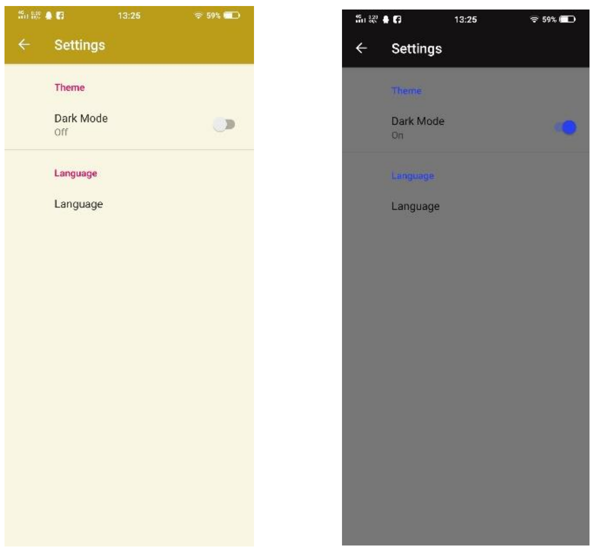
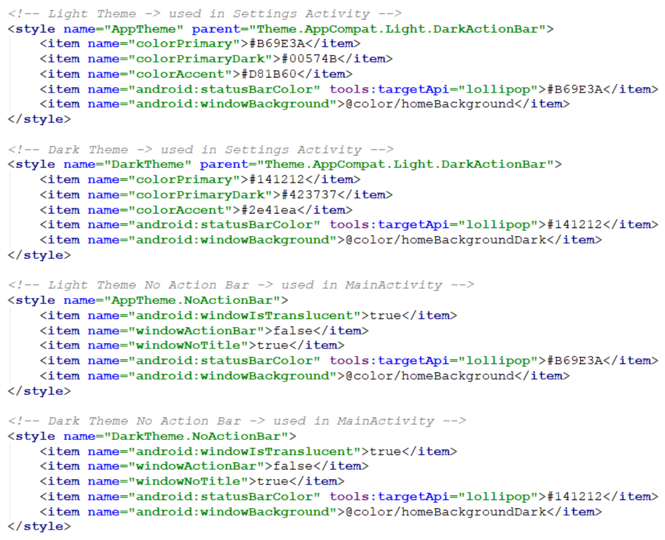


Figure 5.7.1.1: Dark mode off/on

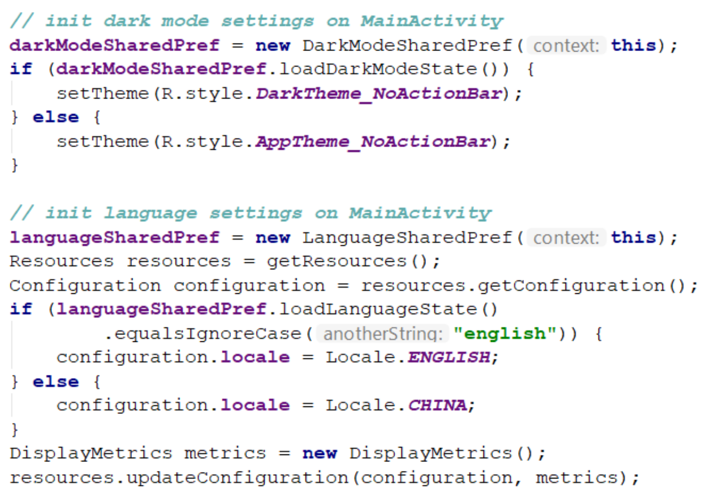
In fact, the dark mode feature can be achieved by simply changing the global theme in Android. As shown below in **Figure 5.7.1.2**, we define a custom dark mode theme that extends the *Theme.AppCompat.Light.DarkActionBar* styles in Android. By assigning different colors to widgets, we can achieve the dark mode effect.



```xml
<!-- Light Theme -> used in Settings Activity -->
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <item name="colorPrimary">#B69E3A</item>
    <item name="colorPrimaryDark">#00574B</item>
    <item name="colorAccent">#D81B60</item>
    <item name="android:statusBarColor" tools:targetApi="lollipop">#B69E3A</item>
    <item name="android:windowBackground">@color/homeBackground</item>
</style>

<!-- Dark Theme -> used in Settings Activity -->
<style name="DarkTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <item name="colorPrimary">#141212</item>
    <item name="colorPrimaryDark">#423737</item>
    <item name="colorAccent">#2e41ea</item>
    <item name="android:statusBarColor" tools:targetApi="lollipop">#141212</item>
    <item name="android:windowBackground">@color/homeBackgroundDark</item>
</style>

<!-- Light Theme No Action Bar -> used in MainActivity -->
<style name="AppTheme.NoActionBar">
    <item name="android:windowIsTranslucent">true</item>
    <item name="windowActionBar">false</item>
    <item name="windowNoTitle">true</item>
    <item name="android:statusBarColor" tools:targetApi="lollipop">#B69E3A</item>
    <item name="android:windowBackground">@color/homeBackground</item>
</style>

<!-- Dark Theme No Action Bar -> used in MainActivity -->
<style name="DarkTheme.NoActionBar">
    <item name="android:windowIsTranslucent">true</item>
    <item name="windowActionBar">false</item>
    <item name="windowNoTitle">true</item>
    <item name="android:statusBarColor" tools:targetApi="lollipop">#141212</item>
    <item name="android:windowBackground">@color/homeBackgroundDark</item>
</style>
```

Figure 5.7.1.2: *styles.xml* configuration

It is noted that when we restart the application without saving the preferences manually, we might lose the preferences that we have set up. In order to fix this issue, I used *SharePreferences* to save and access the current mode. So, every time before an activity is created, we firstly need to get the dark mode settings from *SharePreferences* and apply it to our application. The following figure (**Figure 5.7.1.3**) shows the code to be executed in *MainActivity* before *super.onCreate()* method is invoked.

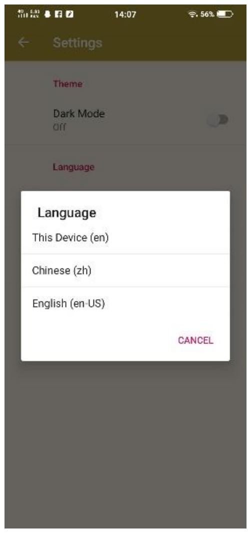


```
// init dark mode settings on MainActivity
darkModeSharedPref = new DarkModeSharedPref( context: this);
if (darkModeSharedPref.loadDarkModeState()) {
    setTheme(R.style.DarkTheme_NoActionBar);
} else {
    setTheme(R.style.AppTheme_NoActionBar);
}

// init language settings on MainActivity
languageSharedPref = new LanguageSharedPref( context: this);
Resources resources = getResources();
Configuration configuration = resources.getConfiguration();
if (languageSharedPref.loadLanguageState()
        .equalsIgnoreCase( anotherString: "english")) {
    configuration.locale = Locale.ENGLISH;
} else {
    configuration.locale = Locale.CHINA;
}
DisplayMetrics metrics = new DisplayMetrics();
resources.updateConfiguration(configuration, metrics);
```

Figure 5.7.1.3: Persistence implementation

5.7.2 Display Language (Internationalization)

Internationalization is another essential feature for this app. Unlike the dark mode, when the user clicks the *Language* option in the settings screen, a dialog box with a list of selectable languages will be displayed (**Figure 5.7.2.1**). The current app supports both English and Simplified Chinese.



14:07

Settings

Theme

Dark Mode
Off

Language

Language

This Device (en)

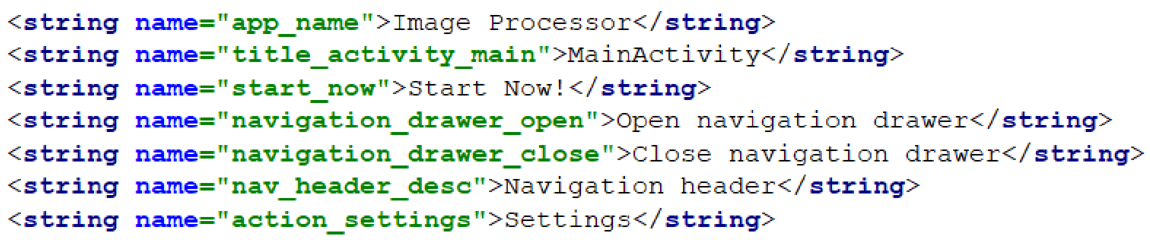
Chinese (zh)

English (en-US)

CANCEL

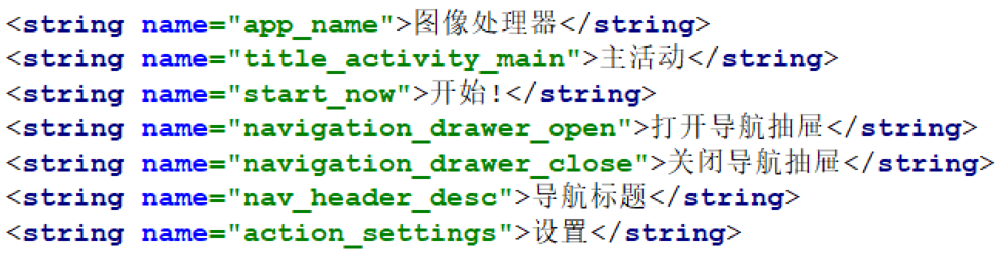
Figure 5.7.2.1: Language dialog

In Android, this feature can be achieved by adding *string.xml* files in different languages under the */res* file directory and switching the display language by modifying the locale in the Configuration. **Figure 5.7.2.2** and **Figure 5.7.2.3** demonstrate a part of string.xml files in two language modes.



```
<string name="app_name">Image Processor</string>
<string name="title_activity_main">MainActivity</string>
<string name="start_now">Start Now!</string>
<string name="navigation_drawer_open">Open navigation drawer</string>
<string name="navigation_drawer_close">Close navigation drawer</string>
<string name="nav_header_desc">Navigation header</string>
<string name="action_settings">Settings</string>
```

Figure 5.7.2.2: Part of *strings.xml* (en)



```
<string name="app_name">图像处理器</string>
<string name="title_activity_main">主活动</string>
<string name="start_now">开始!</string>
<string name="navigation_drawer_open">打开导航抽屉</string>
<string name="navigation_drawer_close">关闭导航抽屉</string>
<string name="nav_header_desc">导航标题</string>
<string name="action_settings">设置</string>
```

Figure 5.7.2.3: Part of *strings.xml* (zh)

In addition, this feature has also achieved the persistence using the techniques mentioned in the last section (as shown in **Figure 5.7.1.3**).

Chapter 6: **Testing**

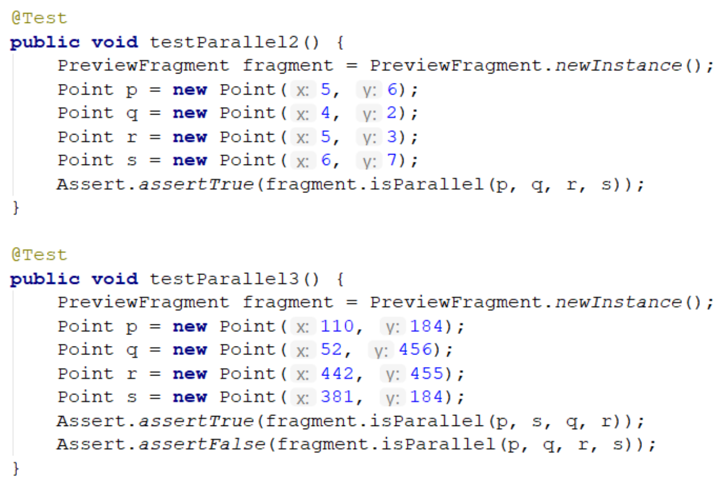
Software testing is an execution process of a program. The purpose is to find and correct errors in the tested software as much as possible and improve the reliability of the software. It is an essential but complex task in the software lifecycle, and it is significantly important to guarantee the software reliability. Nowadays, with the popularity of mobile devices today, the use of mobile applications is also increasing. Therefore, the testing process is equally important for a mobile application.

In this chapter, I will focus on the testing process of the program and discuss from three aspects, unit testing, UI testing and system testing.

6.1 Unit Testing

Unit testing is the first part of this chapter. Unit testing can be used to validate each unit of the application performs as expected. A unit may be a method, a class, a view, etc.

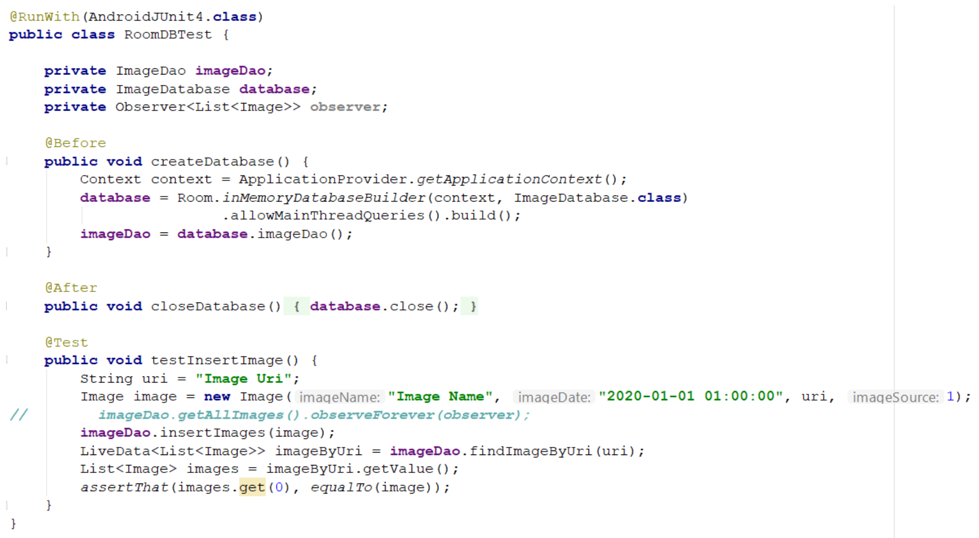
For this program, I generally did unit testing from two aspects, back-end and front-end. The unit testing part for the project covers almost all the main functions in the program. I will only display some of the main tests in this part. For example, in order to ensure that the finding methods mentioned in the previous chapter (distance finding, angle finding, parallelism judgement, etc.) can get the correct judgement results, I ran some tests on each finding method with different incoming parameters. Here are some test cases for the parallelism judgement method I used in the project, this method is used to check if two lines are parallel to each other (as shown in **Figure 6.1.1**).



```java
@Test
public void testParallel2() {
    PreviewFragment fragment = PreviewFragment.newInstance();
    Point p = new Point( x: 5,  y: 6);
    Point q = new Point( x: 4,  y: 2);
    Point r = new Point( x: 5,  y: 3);
    Point s = new Point( x: 6,  y: 7);
    Assert.assertTrue(fragment.isParallel(p, q, r, s));
}

@Test
public void testParallel3() {
    PreviewFragment fragment = PreviewFragment.newInstance();
    Point p = new Point( x: 110,  y: 184);
    Point q = new Point( x: 52,  y: 456);
    Point r = new Point( x: 442,  y: 455);
    Point s = new Point( x: 381,  y: 184);
    Assert.assertTrue(fragment.isParallel(p, s, q, r));
    Assert.assertFalse(fragment.isParallel(p, q, r, s));
}
```

Figure 6.1.1: Parallelism judgement testing

In addition, Room framework is also an important part for database, it has been tested with the following test case as shown in **Figure 6.1.2**.

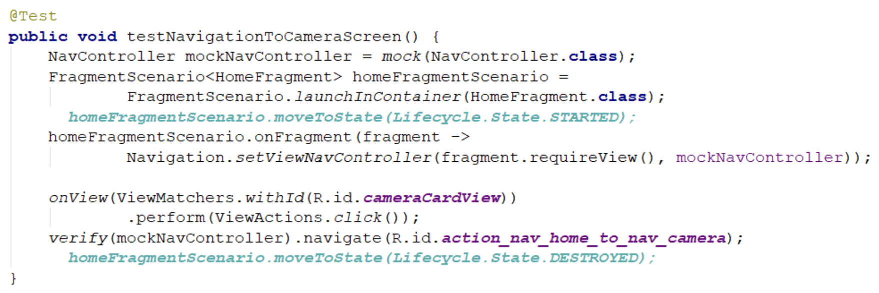


```java
@RunWith(AndroidJUnit4.class)
public class RoomDBTest {

    private ImageDao imageDao;
    private ImageDatabase database;
    private Observer<List<Image>> observer;

    @Before
    public void createDatabase() {
        Context context = ApplicationProvider.getApplicationContext();
        database = Room.inMemoryDatabaseBuilder(context, ImageDatabase.class)
                .allowMainThreadQueries().build();
        imageDao = database.imageDao();
    }

    @After
    public void closeDatabase() { database.close(); }

    @Test
    public void testInsertImage() {
        String uri = "Image Uri";
        Image image = new Image( imageName: "Image Name",  imageDate: "2020-01-01 01:00:00", uri,  imageSource: 1);
//        imageDao.getAllImages().observeForever(observer);
        imageDao.insertImages(image);
        LiveData<List<Image>> imageByUri = imageDao.findImageByUri(uri);
        List<Image> images = imageByUri.getValue();
        assertThat(images.get(0), equalTo(image));
    }
}
```

Figure 6.1.2: Room DB testing

6.2 UI Testing

The UI testing in Android can be done in multiple ways. Generally, the most commonly used tools for UI testing are Mockito with Espresso. In order to use these tools in our testing, we firstly need to add their dependencies to Gradle. The following figure shows a simple UI testing for *home* fragment. As we mentioned earlier, the main purpose of the *home* fragment is to navigate the user to other fragments. We can see in **Figure 6.2.1**, it is for testing the navigation from *home* to *camera* fragment.

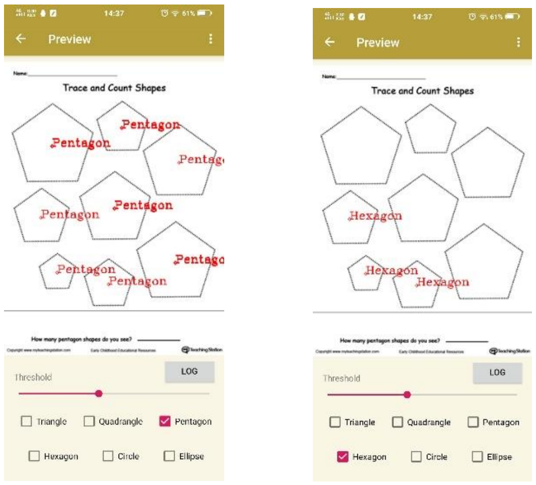


```
@Test
public void testNavigationToCameraScreen() {
    NavController mockNavController = mock(NavController.class);
    FragmentScenario<HomeFragment> homeFragmentScenario =
            FragmentScenario.launchInContainer(HomeFragment.class);
      homeFragmentScenario.moveToState(Lifecycle.State.STARTED);
    homeFragmentScenario.onFragment(fragment ->
            Navigation.setViewNavController(fragment.requireView(), mockNavController));

    onView(ViewMatchers.withId(R.id.cameraCardView))
            .perform(ViewActions.click());
    verify(mockNavController).navigate(R.id.action_nav_home_to_nav_camera);
      homeFragmentScenario.moveToState(Lifecycle.State.DESTROYED);
}
```

Figure 6.2.1: *Home* fragment testing

6.3 System Testing

Shape detection is the core feature of this application. So this part is mainly placed on the testing of shape detection feature. The test was conducted by detecting shapes on many different of images. The images are mainly from the Internet, some screenshots (used to test hand-drawn shapes) or camera or a stitched one with image stitcher. For static shape detection, I have tested on more than fifty images with multiple different kinds of shapes. In most cases, the selected shapes can be correctly detected. But the shapes sometimes can be detected incorrectly (a pentagon maybe wrongly detected as a hexagon, see **Figure 6.3.1**). The reason for this error is because when the image obtained on the Internet has a low resolution, the program may detect an extra vertex on the contour found by the program, so the detected total number of vertices do not match the actually one. In **Figure 6.3.1**, we can see there are three pentagons on the image are detected as both pentagons and hexagons, that is because my algorithm allows the program to find both external and internal contours (*Imgproc.RETR_TREE*). So there are two contours are found for each one.

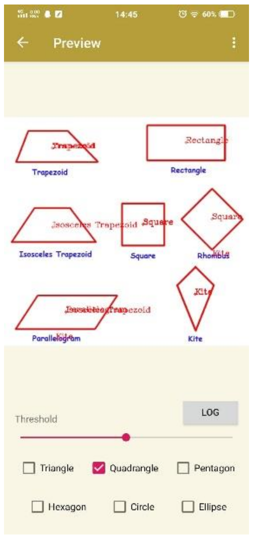


Left: ☐ Triangle ☐ Quadrangle ☑ Pentagon ☐ Hexagon ☐ Circle ☐ Ellipse

Right: ☐ Triangle ☐ Quadrangle ☐ Pentagon ☑ Hexagon ☐ Circle ☐ Ellipse

Figure 6.3.1: Screenshot of a failed test - 1

In addition, another situation that can cause detection errors is when the image source contains text, the program sometimes can detect some shapes that do not exist at all (shown in **Figure 6.3.2**).



☐ Triangle ☑ Quadrangle ☐ Pentagon ☐ Hexagon ☐ Circle ☐ Ellipse

Figure 6.3.2: Screenshot of a failed test - 2

We know that the images obtained from the camera usually have a relatively higher resolution and there are many interference factors, which may also cause some problems in the detection process (**Figure 6.3.3**). One of the possible problems should be the text of detection results displayed are small. This is because the unit in OpenCV is based on pixels and this problem can be avoided by checking the log outputs. In addition, the presence of these interference factors may also have some non-existent shapes detected. In order to avoid this problem as much as possible, I used the blurring methods introduced earlier and also filtered out a part of the detected contours whose area is less than a certain range.

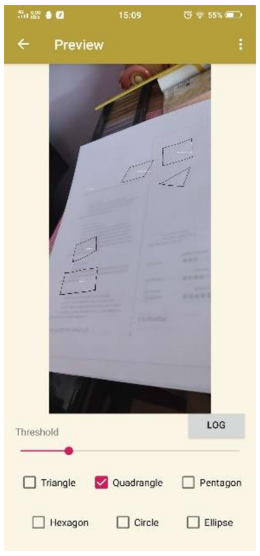


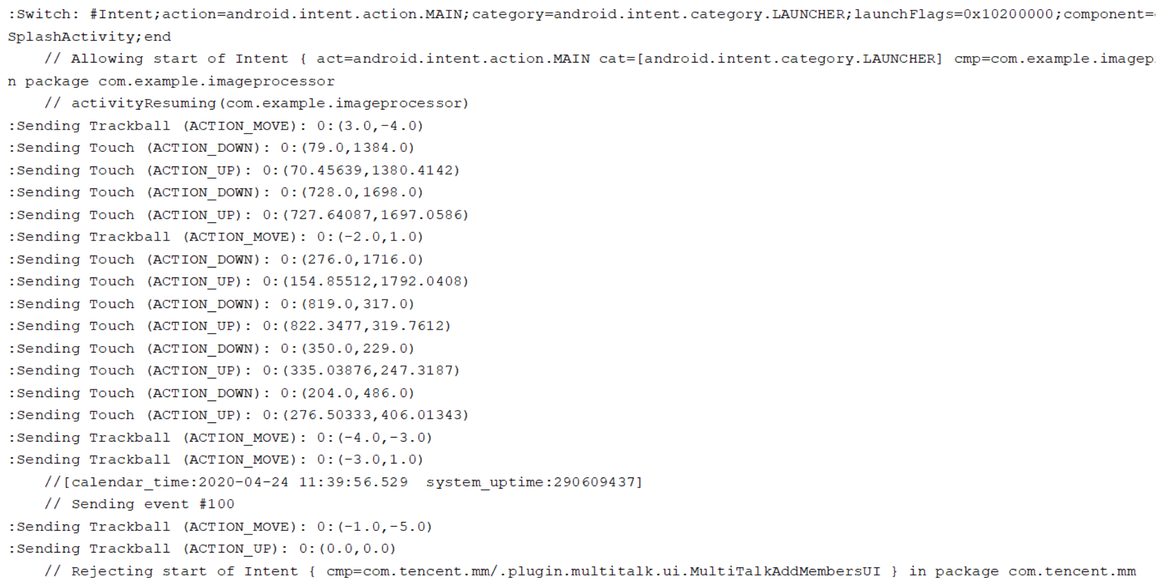
Figure 6.3.3: Screenshot of a failed test - 3

Some tested images have a complex background rather than a uniform background. This factor can cause some edges are wrongly detected during edge detection process, and plus the influence of image noise can also affect the accuracy of output results. But in most cases, we can adjust the threshold value to optimize the detection results.

During the process of system testing, in order to be able to test the execution performance of the application, I also performed some simple stress testing. Stress testing is a type of software testing that mainly measures the application on its robustness and error handling capabilities under extremely heavy load conditions (*What is stress testing in software testing? tools, types, examples* n.d.).

Monkey is the tool I used for stress testing in Android. It is a program that runs on the emulator or device and generates pseudo-random streams of user events such as clicks, touches, or gestures, as well as a number of system-level events (Android Developers 2020). Monkey test is fast and effective way for testing the stability and robustness of an Android app.

In this project, command *`adb shell monkey -p com.example.imageprocessor -v 1000`* is executed in the console in order to perform stress testing with the *Monkey*. *`com.example.imageprocessor`* is the app's package name and *`1000`* identifies the total number of events to be executed. The **Figure 6.3.4** demonstrates a part of outputs in the console when executing the command.



```
:Switch: #Intent;action=android.intent.action.MAIN;category=android.intent.category.LAUNCHER;launchFlags=0x10200000;component=
SplashActivity;end
    // Allowing start of Intent { act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] cmp=com.example.imagep
n package com.example.imageprocessor
    // activityResuming(com.example.imageprocessor)
:Sending Trackball (ACTION_MOVE): 0:(3.0,-4.0)
:Sending Touch (ACTION_DOWN): 0:(79.0,1384.0)
:Sending Touch (ACTION_UP): 0:(70.45639,1380.4142)
:Sending Touch (ACTION_DOWN): 0:(728.0,1698.0)
:Sending Touch (ACTION_UP): 0:(727.64087,1697.0586)
:Sending Trackball (ACTION_MOVE): 0:(-2.0,1.0)
:Sending Touch (ACTION_DOWN): 0:(276.0,1716.0)
:Sending Touch (ACTION_UP): 0:(154.85512,1792.0408)
:Sending Touch (ACTION_DOWN): 0:(819.0,317.0)
:Sending Touch (ACTION_UP): 0:(822.3477,319.7612)
:Sending Touch (ACTION_DOWN): 0:(350.0,229.0)
:Sending Touch (ACTION_UP): 0:(335.03876,247.3187)
:Sending Touch (ACTION_DOWN): 0:(204.0,486.0)
:Sending Touch (ACTION_UP): 0:(276.50333,406.01343)
:Sending Trackball (ACTION_MOVE): 0:(-4.0,-3.0)
:Sending Trackball (ACTION_MOVE): 0:(-3.0,1.0)
    //[calendar_time:2020-04-24 11:39:56.529  system_uptime:290609437]
    // Sending event #100
:Sending Trackball (ACTION_MOVE): 0:(-1.0,-5.0)
:Sending Trackball (ACTION_UP): 0:(0.0,0.0)
    // Rejecting start of Intent { cmp=com.tencent.mm/.plugin.multitalk.ui.MultiTalkAddMembersUI } in package com.tencent.mm
```

Figure 6.3.4: Part of stress testing outputs

Chapter 7: **Evaluation & Conclusion**

7.1 Evaluation

In this report, an Android application named *Image Processor* that is able to detect geometric shapes in an image that can be loaded from system camera or gallery was developed. The detected image can also be a result image after stitching through the image stitcher provided in the application. User can view and manage all the saved images in history. In this app, user is allowed to use the image stitcher to capture multiple images and generate a panorama, and then use the shape detection module to detect geometric shapes on this stitched image.

This application has been tested on *OPPO R15* and *VIVO X27*, and their Android versions are *Android_8.0* and *Android_9.0*, respectively. The test results indicate that all the functions of the application can be executed successfully on both versions. Based on the previous testing results, the *Image Processor* application has succeeded in recognizing 2D geometric shapes in an image with their specific shape types (as for a quadrangle, it can be rectangle, parallelogram, kite, etc.). This application is only designed for detecting some basic geometric shapes, so it is not very effective in detecting some complex shapes. As for the image stitching module, this feature is implemented with OpenCV's Stitcher class, when there's enough features detected for the images to be stitched, the program can successfully stitch all the given images within a limited time. Sometimes this module may fail the stitching process, which is mainly due to the lack of detected features on the images. The settings are another core feature of this application. Due to the time constraint, this module currently only support dark mode and internationalization configurations, it can be said that, in the project development process, the threshold value is used multiple times in shape detection, so in order to facilitate the user to customize the attribute settings of threshold, as well as the saved images, more settings options should be considered in the future. And for history view, the whole module has been refactored and now it can provide a good user experience and have solved the problem of stuttering (mentioned in my interim report) with the aid of Glide, *etc*.

In general, the program currently has met all the requirements set during the phase of design. However, the program was not perfect and requires modifications, and there is still something need to be tidied up, as what I have described previously.

7.2 Conclusion

Shape detection is one of the most significant tasks with multi-applications in the field of computer vision. The rapid rise of robotics and artificial intelligence in recent years has driven the development of shape detection. The study of shapes has always been repeatedly emphasized over the years. We know that shape is one of the main sources of information and also the basis of various technologies, such as object detection.

To be honest, I hadn’t really touched the field of computer vision before this project, and the OpenCV library used in this project is also a fresh attempt. For me, Android is fairly familiar, but during the development of this project, I also tried to use some new technologies that have never tried before, such as Room, ViewPager2, etc. Generally speaking, this FYP gave me a great opportunity to gain a deeper understanding of computer vision and to re-consolidate and improve my Android-related skills.

During the development of the entire project, including research, I inevitably encountered many difficulties. Some of the problems have been discussed in the Implementation section. However, there are still some more problems I have encountered during the development. One of the biggest problems I encountered in the early project construction process was how to display to acquired images to the screen appropriately. I spent a long time on this problem, but finally, with the help of StackOverflow, this obstacle has been overcome. In addition, as I mentioned in my implementation part, in order to implement image stitching feature in Android, I inevitably need to configurate the NDK environment (CMake) to support C++ programming language, so that I can use stitching module provided in OpenCV. For native development to support OpenCV, I have to manually configure the CMake file. But just supporting the general OpenCV library is not enough for this feature, because if you try to use the stitching module in OpenCV in this case, you will still get an error, so we need to import the stitching module separately. The entire native development configuration was the most difficult part for me during the implementation, probably it is because I had never used CMake before. But finally, I solved this problem.

At the end, I want to thank my supervisor for his patient guidance and help during the entire FYP process. Without his trust and help to me during this time, I would not have an achievement like today. The reason I chose this topic at the time was mainly because of my interest and devotion to this area. I deeply believe that in the future, whether for my further education or in my career, I will definitely benefit a lot from this experience. And I personally also cherish this opportunity very much, it makes me realize the charm of computer vision.

Thanks for reading.

References

AI Shack (2017) *Image Moments*, available: https://aishack.in/tutorials/image-moments/ [accessed 18 April 2020].

AI Shack (2018) *The canny edge detector*, available: http://aishack.in/tutorials/canny-edge-detector/ [accessed 23 December 2019].

Android Developers, available: https://developer.android.com/topic/libraries/architecture/room [accessed 12 December 2019].

Bebis, G., Egbert, D. and Mubarak Shah, (2003). Review of computer vision education. *IEEE Trans. Educ.*, 46(1), pp.2-21.

Chetan, S.M. and Anita, P.P. (2015) 'Circular Hough transform for detecting and measuring circles of object', International Journal on Recent and Innovation Trends in Computing and Communication, 3(2), 563-566, available: https://www.slideshare.net/editorijritcc1/circularhough-transform-for-detecting-and-measuring-circles-of-object [accessed 9 November 2019].

Dummies (2020) *Identifying the seven quadrilaterals* [image], available: https://www.dummies.com/education/math/geometry/identifying-the-seven-quadrilaterals/ [accessed 21 April 2020].

El Abbadi, N. (2013) ‘Automatic detection and recognize different shapes in an image’, *ResearchGate*, available: https://www.researchgate.net/publication/259949841_Automatic_Detection_and_Recognize_Different_Shapes_in_an_Image [accessed 7 November 2019].

ElProCus (2020) *What Everybody Ought to Know About Android: Introduction, Features & Applications*, available: https://www.elprocus.com/what-is-android-introduction-features-applications/ [accessed 14 April 2020].

Elrefaei L.A., Al-musawa M.O. and Al-gohany N.A. (2017) ‘Development of an Android application for object detection based on color, shape, or local features’, *The International Journal of Multimedia & Its Applications*, 9(1), available: https://arxiv.org/ftp/arxiv/papers/1703/1703.03848.pdf [accessed 7 November 2019].

Basic stitching pipeline (n.d.) Google Sites, available: https://sites.google.com/site/ritpanoramaapp/project-stage-iii [accessed 22 April 2020].

Kapur, S. and Thakkar N. (2015) Mastering OpenCV Android Application Programming, Birmingham: Packt Publishing.

Kimme, C., Ballard, D. and Sklansky, J. (1975) “Finding circles by an array of accumulators”, In the Communications of ACM, pp. 120-122.

MathIsFun (2018) *Definition of circle*, available: https://www.mathsisfun.com/definitions/circle.html [accessed 21 April 2020].

MathIsFun (2018) *Definition of circle* [image], available: https://www.mathsisfun.com/definitions/circle.html [accessed 21 April 2020].

MindOrks (2018) *Medium* [image], available: https://medium.com/mindorks/room-with-rxjava-and-dagger-2722f4420651 [accessed 12 December 2019].

OpenCV (2020) *Android Development with OpenCV*, available: https://docs.opencv.org/2.4/doc/tutorials/introduction/android_binary_package/dev_with_OCV_on_Android.html [accessed 22 April 2020].

OpenCV (2020) *Stitching pipeline* [image], available: https://docs.opencv.org/2.4/modules/stitching/doc/introduction.html [accessed 22 April 2020].

OpenCV (2019) *logo* [image], available: https://opencv.org/ [accessed 7 November 2019].

ResearchGate (2014) *HSV color space-Hue, saturation, value* [image], available: https://www.researchgate.net/figure/HSV-color-space-Hue-saturation-value_fig1_284698928 [accessed 7 November 2019].

SoftSchools (2020) *Ellipse: standard equation* [image], available: https://www.softschools.com/math/pre_calculus/ellipse_standard_equation/ [accessed 21 April 2020].

StudentProjectWorks (2015) *Learning Shapes studentprojectworks* [image], available: https://studentprojectworks.wordpress.com/2015/07/23/learning-shapes-studentprojectworks/ [accessed 17 April 2020].

Swain, A. (2018) *Noise in digital image processing*, Medium, available: https://medium.com/image-vision/noise-in-digital-image-processing-55357c9fab71 [accessed 8 November 2019].

Wikipedia (2019) *An image with salt-and-pepper noise* [image], available: https://en.wikipedia.org/wiki/Salt-and-pepper_noise [accessed 8 November 2019].

Wikipedia (2020) *Centroid*, available: https://en.wikipedia.org/wiki/Centroid [accessed 18 April 2020].

Wikipedia (2020) Image moment, available: https://en.wikipedia.org/wiki/Image_moment [accessed 18 April 2020].

What is stress testing in software testing? tools, types, examples (n.d.) Guru99.com, available: https://www.guru99.com/stress-testing-tutorial.html [accessed 24 April 2020].

Yang, H.Z. (2015) *Automated fruit and flower counting using digital image analysis*, Universiti Tunku Abdul Rahman, available: http://eprints.utar.edu.my/1813/1/BEE-2015-1005052-1.pdf [accessed 8 November 2019].