Target Scan Application

Junchao Wu

4th Year Project Report
Computer Science
School of Informatics
University of Edinburgh

2023

Abstract

This report discusses the background, design, implementation and evaluation of the Target Scan Android application. This application is capable to process target paper images, automatically output the scores, and store all shooting records locally. It also includes data analysis features to help users improve their shooting skills. The recognition algorithm was implemented using traditional image processing techniques, and several OpenCV library functions are used to simplify the implementation. The scoring algorithm calculates the Euclidean distance and compares it to the radius of each ring. The application is written in Kotlin, and each component is explained in detail. An evaluation of the application and the algorithm was carried out, and some limitations and potential improvements are also discussed. Finally, the achievements of this project and general remarks are given.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Junchao Wu*)

Acknowledgements

I would like to thank my supervisor Professor Julian Bradfield for his continuous support and guidance throughout my final year. The weekly meetings were very helpful for me to keep making progress on the project and not procrastinating, and the advice on writing the dissertation was also valuable.

I would also like to thank my parents, my friends for their continuous support and encouragement.

Table of Contents

Chapter 1

Introduction

Rifle shooting is a civilian sport enjoyed by millions of people around the world. Players fire at targets from certain distances and score points based on the number of rings hit. This project focused on small-bore rifle shooting, which using .22 ammunition and firing from 25 yards (Figure 1.1). There are 10 targets on the target paper. This sport is explained in more detail in Section 2.1.

In the normal process, users manually determine the score of each bullet hole. For bullet holes that are close to the ring, a gauge is needed to accurately determine whether lower scores should be given. Figure 2.2 clearly illustrates this situation. Therefore, the scoring process requires additional tools and a lot of time if there are many targets. Moreover, it is not convenient to view old target papers and records, as they are stored separately. This project ”Target Scan Application” aims to address these inconveniences by digitising and automating this process. This application is developed for the Android platform, as it has larger market share and it is easier to distribute the application installation package. Users can use the application to capture target paper images, and the application can automatically recognise the targets and bullet holes from the target paper images, followed by scoring and storing. Users can also view existing records. In addition, a data analysis feature was added to this project, to make a use of the existing records and help users improve their shooting skills.

After a year of research and development, I successfully completed this project and implemented all the expected features. The application was built from scratch, including the design of the image processing algorithm, which is a core part of the project. It digitises and automates the process of scoring and recording target papers, and is capable to recognise all targets from target papers under different lighting conditions, and provide scores with high accuracy. Several methods of evaluation were also carried out to ensure the application meets the specifications set during the design stage. The detailed information of this project, including the background, design, implementation and evaluation are discussed in the following chapters.

The Background chapter 2 is intended to let readers quickly grasp the background knowledge needed for this project and some of the concepts discussed in this report. Readers are provided with basic information about rifle shooting sport, and some

Figure 1.1: Small-bore rifle shooting[18]

previous research on this topic is discussed. Furthermore, important image processing techniques used in this project and some terminologies which are frequently used in the Android development are explained. The next chapter 3 explains the design decisions made before and during the development, and the design of three main functions. It also provides specifications for the image processing algorithm and the application, which are used in the evaluation. Implementation chapter 4 explains the implementation of each activity in the application, as well as all steps in the image processing algorithm. Readers can understand how the image processing algorithm works, and how each activity is implemented to provide the expected functionality. In evaluation chapter 5, the application and the image processing algorithm were evaluated by several quantitative and qualitative analysis methods, including hardware usage, accuracy, processing time, etc. A simple user evaluation was also carried out. In conclusion chapter 6, the achievements of this project, some limitations and possible improvements are discussed.

Chapter 2

Background

2.1 Rifle Shooting Background

Rifle shooting has a long history dating back to the 1800s.[20] The earliest shooting clubs and organisations were formed in Switzerland during the 1820s, followed shortly by other European countries and North America. The first international shooting competition was held at the 1896 Summer Olympics. In the United Kingdom, rifle shooting developed as a civilian sport from military requirements. In the early 1900s, civilians were expected to have some shooting skills to defend themselves if the regular army forces failed to stop an invasion. As a result, more and more clubs sprang up and costs also dropped, making it more accessible as a hobby. The Second World War also accelerated the spread of rifle shooting among civilians. After the war ended, rifle shooting became primarily a civilian sport. More national and international competitions were established and new types of events were added to the existing ones. Today, rifle shooting is enjoyed by millions of people around the world, regardless of gender, age or skill level.

This project focuses on small-bore rifle shooting, which involves using .22 calibre target rifles to fire .22 rimfire ammunition at paper, cardboard or electronic targets. Distances for this type of shooting are typically 15-25 yards indoors, 50 yards/meters and 100 yards outdoors.[5] The size and style of the target papers vary depends on the distance. In this project, the 25-yard target paper with ten black targets arranged in a 4-2-4 layout from top to bottom was used (Figure 2.1). The diameter of each black target is 2.05 inches (5.207 centimetres) with six rings scoring from ten to five points.

2.2 Scoring the Target Papers

After shooting, targets are scored using a 5.6 mm (.22 in) gauge (Figure 2.3).[3] The target has a center ring worth 10 points, and each outer ring is worth one point less. If the gauge overlaps with the white line and its edge is outside of the line, a lower score is given (Figure 2.2). Since the hole left on the target paper cannot represent the actual size of the bullet, scoring algorithms should first find the center and use it to estimate

Figure 2.1: Target paper for 25 yards rifle shooting[24]

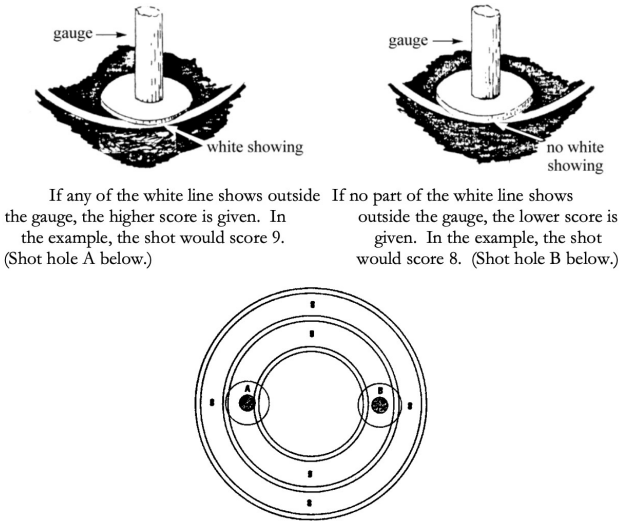

Figure 2.2: Use of the 5.6 mm gauge on OUTWARD gauging targets[3]

the actual area of the bullet hole.

2.3 Previous Research on Targets Recognition Topic

There has been some research into this topic. However, most existing solutions are designed for fixed shooting environments, where a stationary camera captures targets and automatically scores them. In these cases, the perspective transformation process can be set manually by the operator and bullet hole positions can be obtained by comparing two images before and after the shot. In contrast, in my case the users need to use their phone's camera to capture the images of target papers, so the lighting conditions and image distortion vary. However, it is still helpful to understand how they processed target images to make bullet holes more identifiable, and then separated them from the background.

One approach, proposed by Widayaka Parama in 2019 [25] uses contour detection to

Figure 2.3: A sample of 5.6mm gauge[23]

obtain the positions of the center and rings, followed by background subtraction to separate bullet holes from targets. The Euclidean distance between the center and the center of each bullet hole is then compared to the radius of each ring to determine a score. This approach resulted in a 91% accuracy, and the main weakness was in the image subtraction process, which creates noise that causes errors in subsequent bullet hole detection process. Another paper proposed in 2009 [7] uses similar methods. The researchers used subtraction to segment bullet holes from the target, then traced the boundary of each segmented hole, and fitted ovals to them. In 2011, Cuiliu Ye introduced a military automatic target-scoring system includes both the hardware and the software.[27] The workflow of this auto-grading system can be divided into the following steps: target image rectification, bullet-spot's extraction, bullet-spot's recognition and scoring.

Another approach [12] involves placing a blue paper underneath the target paper before taking photos and searching for blue pixels in the images to detect bullet holes. This is easy to implement and accurate. However, it is unrealistic to ask users to bring coloured papers with them in my case. A paper proposed by Aryan Peb[2] shows a clear image processing path applied to the original image to make it suitable for bullet hole segmentation. It starts with grayscale conversion, followed by Gaussian blurring to suppress random noise. Next, morphological operations and binarization are used to retrieve outer rings and remove small unwanted regions. Finally, boundary contours are extracted from binary images, and polygon approximation is built, followed by an ellipse fitting algorithm from OpenCV, and the largest ellipse is the entire target. For bullet hole segmentation, they used machine learning techniques to determine whether black areas in an image represent bullet holes or other noise.

2.4 Image Processing Methods

In this section, the main image processing techniques used in this project are discussed.

2.4.1 Grayscale Conversion

Grayscale conversion is commonly used in image processing. For some tasks, such as identifying edges and features, grayscale contains most of the relevant information, so removing colours can simplify the calculations and reduce the processing time. The OpenCV library provides the cv2.cvtColor() function to perform this operation. The formula used is Y = 0.299 R + 0.587 G + 0.114 B.[15] The reason for not using a simple average of RGB colours is that humans are more sensitive to green compare to blue and red. This formula takes this into account, and the produced grayscale image looks more

natural to the human eye.[6]

2.4.2 Adaptive Threshold

Threshold is a fixed value to binarise a grayscale image. Global threshold replaces all pixels with an intensity less than the threshold with black pixels, or white pixels if the intensity is greater than the threshold. In adaptive threshold, the threshold value is calculated for each smaller region in an image. The OpenCV library provides the cv2.adaptiveThreshold() function to perform this operation.[17] It has two methods for calculating the adaptive threshold values, ADAPTIVE_THRESH_MEAN_C and ADAPTIVE_THRESH_GAUSSIAN_C. The first one uses the mean of the neighbourhood area and the second one uses the weighted sum of the neighbourhood values where weights come from a Gaussian window. Users also need to specify a block size that determines the size of the neighbourhood area. In this project, an adaptive threshold method is used to remove the influence of shadows.

2.4.3 Median Blur

Median blur is a non-linear filter used to remove noise from an image. It works by replacing each pixel value with the median of neighbouring pixel values under a kernel area.[26] The kernel size must be positive and odd. The OpenCV library provides the cv2.medianBlur() function to perform this operation. Users can specify a kernel size as a parameter. Compared to Gaussian blur and averaging blur, the use of median calculation makes the median blur filter highly effective against salt-and-pepper noise, which is common in processed images.[22] Since some morphological transformations can magnify shapes in the image, it is important to remove noise from the image first.

2.4.4 Morphological Transformation

Morphological operations are simple transformations applied to the shapes and structures within the images. [21] In general, they can make some parts of the image larger or smaller. The Figure 2.4 shows some common morphological operations and their effect on the original image. The OpenCV library provides the cv2.morphologyEx() function to perform these operations. Once the original photo is converted to binary image, morphological operations are used to extract the targets from the target paper. It is also used to extract the bullet hole from each target.

2.4.5 Contour Detection

A contour is an outline of an area where inner pixels have the same property (colour, intensity, etc...)[13]. It can extract the shape or structure in images, and is a useful tool for shape analysis and object detection[10] (Fig 2.5). The OpenCV library provides the cv2.findContours() to find contours from a given image. It is recommended to use binary image for contour detection. In this project, it is used to find the targets and bullet holes from the background.

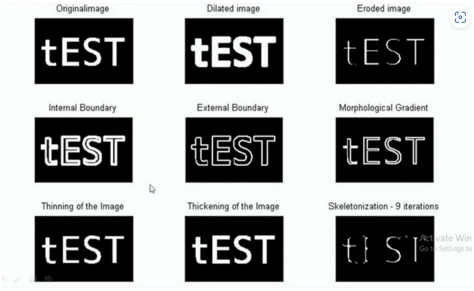

Figure 2.4: Effects of several morphological operations[21]

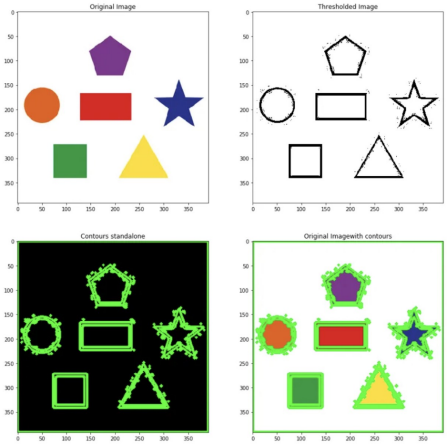

Figure 2.5: Examples of contours detection[10]

2.5 Terminology

2.5.1 Activity

In Android development, an activity refers to a full screen page view with its functional code. An application typically consists of multiple activities, and users can navigate between them by clicking on buttons.

2.5.2 SharedPreferences

SharedPreferences is a data storage option provided by Android. It is an XML file that can be accessed and modified by relevant functions, and is used to store and retrieve small amounts of primitive data as key/value pairs. The usage of SharedPreferences file in this application is explained in Section 4.1.5.2.

2.5.3 UI Components

Here are some common UI components used in the project, the name and their functions are explained:

- EditText - Display text in it, the text can be edited.
- TextView - Displays text in it, read only.
- ImageView - Display a image in it.
- RecyclerView - Display a scroll list in it.
- SeekBar - A draggable thumb that users can touch the thumb and drag left or right to select a value or an option among all stops. Common examples are the brightness control and volume control on Android phones.

Chapter 3

Design

This chapter contains two sections. In Section 3.1, key design decisions are discussed, and in Section 3.2, specifications for the project components are set.

3.1 Design Decisions

In this section, key design decisions made in this project are discussed. During the development, the project was divided into two main tasks: creating the Android application and designing an image processing algorithm. The image processing algorithm is an critical component of the application, as it involves converting target paper photos taken by users into binary images that are suitable for contour detection, followed by extracting targets and bullet holes from the background, and finally grading them. It requires extensive research and experimentation to achieve high accuracy. so it is developed prior to the application.

3.1.1 Programming Language

3.1.1.1 Application Development

Java and Kotlin are the two official language options for developing Android applications. For a long time, Java was the only choice available. Kotlin 1.0 was launched by JetBrains in 2016 and later, with Google's support, it was introduced as an alternative to Java for mobile development. Kotlin offers some advantages over Java, including simpler coding and null reference elimination, and it is fully compatible with Java. Since Google I/O in 2019, Android mobile development has been Kotlin-first, and most official tutorials and documentation are now written in Kotlin[11].

After a few years of transition, the majority of developers have switched to using Kotlin, and it is easy to get answers to questions about Kotlin programming on forums like Stackoverflow. In addition, the Android Studio IDE includes a feature to seamlessly convert Java code into Kotlin code, and Kotlin is interoperable with Java, meaning that they can be used side by side in the same project. For all these reasons mentioned above, I finally chose Kotlin as the main programming language for this project.

3.1.1.2 Image Processing Algorithm

As explained in Section 3.1.2, I decided to use traditional image processing methods to develop the algorithm. The OpenCV library provides a range of useful operations in the form of functions and is available in both Python and Java. I chose Python as my developing language because of its ”what you see is what you get” experience with Jupyter Notebook, which allows me to check intermediate steps to adjust the algorithm and fix bugs. In addition, using Python helps avoid bugs related to variable types. Therefore, it is way more efficient than Java. Another reason for choosing Python is that there is no equivalent to the Numpy library in Java. Although Java has the Mat Class for matrix manipulation, it is less intuitive and more complex compared to Numpy. Finally, there are more OpenCV resources available in Python that I could learn from.

3.1.1.3 Integrating the Algorithm into the Application

I had two options to integrate the image processing algorithm into the application. The first one is to rewrite the validated image processing algorithm in Java. The advantage of using native language in the application is that it is more efficient and fast. However, I found this approach is much harder than I expected due to the differences in function usage between OpenCV(Java) and OpenCV(Python). Besides, the Mat Class in Java used for representing matrix is also hard to use compared to the Numpy Class in Python. Moreover, Java is a static programming language and the type of variables are defined by developers, and I always got type errors during development. Furthermore, I still need to test and adjust the algorithm in Jupyter Notebook and update the improvements to the Java version, which is inconvenient. Therefore, I finally abandoned this approach.

The second option is to use Python code directly in the Android application via a framework called ”Chaquopy”[4]. It can execute Python scripts through an interface class and store the results into Kotlin variables. After trying this method, I found that it solved my previous problems. It was easy to implement, and I could simply copy the codes from Jupyter notebook to update the algorithm.

3.1.2 Target/Bullet Hole Recognition Methods

There are two ways to recognise targets or bullet holes from the background. The first option is to use traditional image processing techniques, and the second option is to use machine learning. I chose the traditional approach because targets and bullet holes have obvious colour and shape features, so they are relatively easy to distinguish from the background. Also, using machine learning requires me to manually create and label the dataset, which is time-consuming and I also lacked experience in this field. The OpenCV library[14] provides a range of traditional image processing methods, and played an important role in my implementation.

3.1.3 Grading Methods

I devised two methods of grading the targets. The first method is to find the centre of both the target and bullet hole, calculate their Euclidean distance, divide it by the

Figure 3.1: An extreme example of perspective distortion

radius of the target in the image to get a ratio, multiply this by the actual radius to get the distance on the real target. Adding the bullet radius to this actual distance and comparing it with each ring's radius, I can get the score of the target. Another method is to check directly whether the edge of the bullet hole overlaps a particular ring.

Each method has its advantages and disadvantages. The first method is relatively easy to implement, but the accuracy can be affected by the perspective distortion in photos. This occurs when the image sensor is not parallel to the target paper. Although I have a method (described in Section 4.3.3.3) to restore elliptic targets to perfect circles, the perspective transformation cannot be restored (Fig 3.1 shows an extreme example) . However, the effect is minimal when the angle between the sensor and the paper is small, and users can be guided by the application to avoid taking such images. The second method doesn't have this problem, but it also has some issues. After a bullet penetrates through the target paper, the hole left on the paper may not be the same size as the bullet (this is one reason to use gauges for scoring), and the edge of the hole may also be torn. As a result, there is also some error in the measurement. Moreover, this method requires a precise recognition of each ring, which can be difficult if the lighting condition is poor, as the rings are thin white lines. Overall, I chose the first method because of its simplicity and robust recognition results.

3.1.4 Data Storage

The application parameters and shot records are stored in data.xml file as key-value pairs. For records, the key is the filename of each image, and the value is either a comment (for unprocessed records) or "Processed" (for processed records). Each image has a unique name consisting of discipline, date, and ID (3 digits), in the format "discipline+date+ID(3 digits).jpg". Including this information in the filenames makes them easy to be accessed in activities. A SQLite database is used to store processed records with all relevant information. The implementation is discussed in Section 4.1.5.

3.1.5 Application Functions

The application is designed to have three main functions: loading and processing images, viewing records, and analysing data.

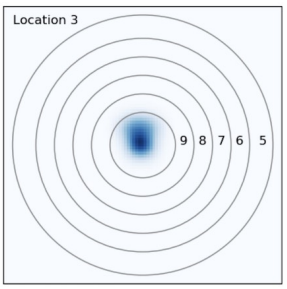

Figure 3.2: A distribution of several shots

3.1.5.1 Loading and Processing Images

Users use this function to load target papers into the application. The first step is to enter information, including discipline, date and comment. Next, users capture an image of the target paper or upload one from albums. Once they confirm the image to be used, it is stored in the application folder along with a corresponding record in the XML file. Users have the option of processing images immediately or at a later time. Processing takes a few seconds, and the result is displayed on the screen. Users can view them and edit incorrect scores before saving the processed record to the database.

3.1.5.2 Viewing Records

Users use this function to view local records. Records are organised in a scrollable list and grouped by the month and date for easy access. Each row in the list corresponds to a record and displays basic information including ID and processing status. Clicking on a row takes the user to a detailed record activity. It displays the information and image of the targets, along with the scores and bullet holes positions. Users can edit the comment and scores on this activity, or delete the record. If the image has not been processed, there is a button that takes the user to the processing activity.

3.1.5.3 Analysing Data

Users use this function to view graphs of the analysed data. For each bullet hole, the offset from the target centre is stored during processing, and they are used to generate a heat map that illustrates the distribution of shots (Fig 3.2). Users can select a specific time period or number of records to generate the heat map. They also have the option to generate heat maps for specific positions (there are 10 targets in the target paper). This feature is useful for users to learn from the distribution of shots and adjust their posture for the next shot.

3.1.6 Theme

A light colour theme is designed for this application. The background is white, while the buttons and toolbars are soft lime green (#80BEDC39), and the text is dark green (#1C4E1F). This colour scheme looks friendly and makes the content easy to read.

3.2 Specification

This section specifies the specifications for the project components that need to be met. These specifications serve as a basis for evaluating the success of the project.

3.2.1 Image Processing Algorithm

- **Capability:** Given a photo of the target paper, the algorithm must be able to identify the targets and the bullet holes from the background, and provide scores along with other relevant information.
- **Determinism:** For the same photo, the algorithm must provide the same output.
- **Accuracy:** The algorithm should be able to recognise targets and provide scores with high accuracy.
- **Robustness:** The algorithm should be able to identify photos taken in low-light conditions or with minor perspective distortion.
- **Speed:** The algorithm should have a reasonable processing time.
- **Automation:** The algorithm should be able to process images automatically without additional input from users.

3.2.2 Application

- **Intuitiveness:** The user interface should be intuitive and require no additional training.
- **Responsiveness:** The response time to user actions must be short enough that it is imperceptible to users.
- **Functionality:** All functions discussed in Section 3.1.5 should be implemented.
- **Robustness:** The application should remain stable and not crash during regular use.
- **Compatibility:** The application should run on most Android phones. It requires that the application to be compatible with old Android versions and not demanding for the CPU and memory.

Chapter 4

Implementation

This chapter contains two sections. Section 4.1 provides an overview of the application, covering typical scenarios, activity interconnections, permission requirements, OS compatibility and data storage details. Section 4.2 provides explanations for the functions and implementation details of 10 activities in the application, as well as the image processing algorithm.

4.1 Application Overview

4.1.1 Typical Scenarios

The following sequences illustrate typical scenarios for using the application.

4.1.1.1 Add New Records

1. When the application is launched for the first time, users are prompted to grant certain permissions (Explained in Section 4.1.3).
2. Users click the ”Add New” button to start adding a record.
3. Users input information about to the record, including discipline, date of the shot and comment.
4. The camera feed is displayed on the screen and users can press the button to take a photo. Alternatively, they can upload a photo from albums.
5. The photo is displayed on the screen. Users can rotate it if it is necessary.
6. Users confirm the photo, and they can process the photo immediately or later. The photo is saved to existing records if they choose to process later.
7. It takes about 5-15 seconds (depending on hardware) to process the photo.
8. Once the photo is processed, the scores are displayed on the screen.
9. Users can view the cropped target images from the original target paper image, and correct the scores if needed. Once complete, the results are then saved.

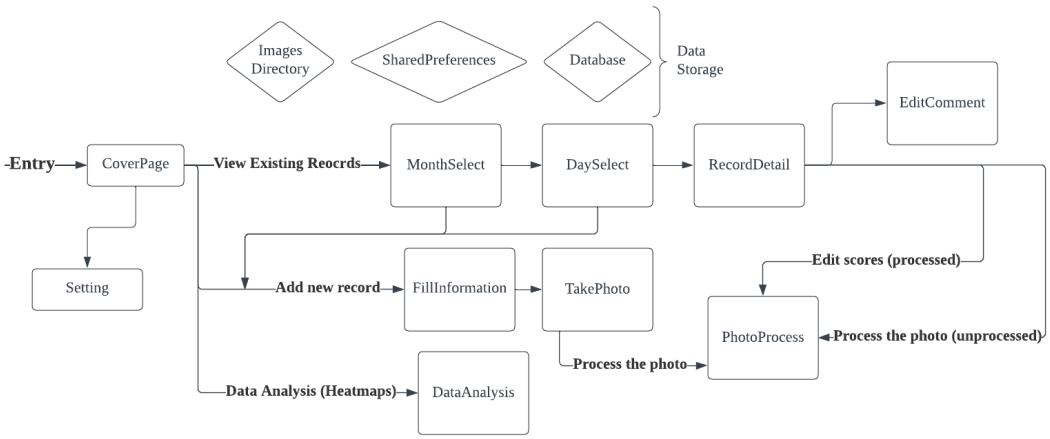

Figure 4.1: Flowchart

4.1.1.2 View Existing Records

1. Users click on the ”Records” button.
2. Users can select a month and a scroll list will appear with each row corresponding to a day in that month which has records. The default month is the current month.
3. By clicking on a row, users enter an activity containing all the records for that day in a scroll list. Each record is displayed as a separate row in the scroll list.
4. Users click on a row to view the record details, including discipline, date, comment, scores and target images.
5. Users use a SeekBar to switch between the whole target paper image or individual target images. The image for a single target can be switched between a cropped one from the original image or a rendered graph.
6. Users have the option to edit comments or scores by clicking on their respective buttons. Once complete, they click ”Confirm” button to save changes. They can also click the trash-bin icon to delete this record.

4.1.1.3 Data Analysis

1. Users click the ”Analysis” button to enter the data analysis activity.
2. Users use a scroll bar to select a time period or a number of newest records for generating the heatmap. They can also choose to generate heatmap based on shots from a particular target position or from all positions.
3. Users click the ”Show Heatmap” button to generate the heatmap.

4.1.2 Flowchart

Figure 4.1 is the flowchart for the application. Each box represents an activity, and each diamond represents a data storage option. This flowchart illustrates the connection

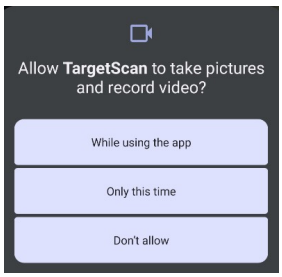

Figure 4.2: Camera permission request

between activities, and the flow of three typical scenarios which explained in Section 4.1.1.

4.1.3 Permissions Required

The following permission are required and users are prompted to grant them in the first launch:

- Camera. (Figure 4.2)
- Write external storage. This permission is only required for Android devices that use the Android Software Development Kit (SDK) version 28 or earlier.

4.1.4 OS Compatibility

SDK is the abbreviation of Software Development Kit, also known as API level. Each version of the Android operating system has a specific API level, and application's minimum SDK must be equal or higher than the system's API level to be compatible.

The application is compiled with SDK 33. The minimum SDK is 26, which corresponds to Android 8.0, so it run on devices with Android 8.0 or higher[19]. According to Figure 4.3, 93.3% Android devices are compatible with this application.

The size of the installation package is 172.1MB, and the size of the installed application is 172MB.

4.1.5 Data Storage

This section explains how data and images are stored in the local.

4.1.5.1 Images Storage

After capturing or uploading a photo, the application saves a local copy in its folder. The name of the photo follows this format: "discipline ID+date+photo ID.jpg". The discipline ID is 1 digit used to represent disciplines. In this project there is only one discipline "small-bore rifle shooting", and its ID is 0. Dates are written as "yyyy-mm-dd". The photo ID has 3 digits, ranging from 001 to 999, and is incremented for photos taken on the same day, to ensure the names are unique. For example, the first image for small-bore rifle shooting on 2023/03/09 is named as "02023-03-09001.jpg". The

Version	SDK / API level	Version code	Codename	Cumulative usage [1]	Year
Android 14 DEV	Level 34	UPSIDE_DOWN_CAKE	Upside Down Cake	—	*TBD*
Android 13	Level 33	TIRAMISU	Tiramisu [2]	14.4%	2022
	▪ **targetSdk** will need to be 33+ for new apps and app updates by August 2023.				
Android 12	Level 32 Android 12L	S_V2	Snow Cone [2]	37.9%	
	Level 31 Android 12	S			2021
	▪ **targetSdk** must be 31+ for new apps and app updates.				
Android 11	Level 30	R	Red Velvet Cake [2]	60.4%	2020
Android 10	Level 29	Q	Quince Tart [2]	78.0%	2019
Android 9	Level 28	P	Pie	86.6%	2018
Android 8	Level 27 Android 8.1	O_MR1	Oreo	91.6%	2017
	Level 26 Android 8.0	O		93.3%	
Android 7	Level 25 Android 7.1	N_MR1	Nougat	94.4%	2016
	Level 24 Android 7.0	N		96.5%	
Android 6	Level 23	M	Marshmallow	98.2%	2015
Android 5	Level 22 Android 5.1	LOLLIPOP_MR1	Lollipop	99.3%	2015
	Level 21 Android 5.0	LOLLIPOP, L		99.5%	2014
	▪ Jetpack Compose requires a **minSdk** of 21 or higher.				

Figure 4.3: Table of Android version and SDK level[19]

benefit of putting this information in the image names is that I can parse them directly to retrieve the information, rather than having to query the database.

4.1.5.2 Application Data

There is a SharedPreferences file used in this application to store the application data. It is named as data.xml and contains the following pairs:

- totalNum: int, used to store the number of records.
- ascendingOrder: boolean, used to represent the order of scroll lists (ascending or descending).
- image name: String. Each pair represents a record, the key is the image name and the value is the comment for unprocessed records or ”Processed” for processed records.

4.1.5.3 Database for Processed Records

A SQLite database is used to store the complete information for processed records. The table ShootingRecords has the following keys:

- filename tinytext primary key, the name of the image file.
- targetNum tinyint, the number of the targets found in the target paper.
- scores tinytext, the scores of the records in a format of ”score1,score2,score3”. Scores are converted to string and connected by ”,”.

- discipline tinyint, the discipline ID, like 0.
- year tinyint, the year of the shot, like 2023.
- month tinyint, the month of the shot, ranging from 1 (January) to 12 (December).
- day tinyint, the day of the shot, ranging from 1 to 31.
- comment tinytext, the comment entered by users.
- vectors tinytext, the offset of each bullet hole from the target center in a format ”y1_cord_offset,x1_cord_offset.y2_cord_offset,x2_cord_offset”. ”.” is used to separate offsets for each bullet hole, and ”,” is used to separate the offset on y axis (vertical) and x axis (horizontal). The up or right side of the axes are positive. For example, ”6,-9” means the bullet hole is located 6 pixels up to the center and 9 pixels left to the center.
- positions tinytext, the position and radius of each target in the original image in a format ”x1_pos,y1_pos,radius1.x2_pos,y2_pos,radius2”. ”.” is used to separate positions and radii for each target, and ”,” is used to separate the coordinate on x axis (horizontal), the coordinate on y axis and the radius. The zero point is on the top left of the image. For example, ”2560,681,352” means the target centre is located at (2560,681) and the radius of the target is 352 pixels.

4.2 Application Activities

This section explains all activities in the application. These activities are the place to implement user interfaces and functions.

4.2.1 CoverPage Activity

This is the first page users see when they open the application (Figure 4.4). It acts as a cover for the application, and has four large buttons that direct users to functions. The button names, the corresponding activities and the functions are:

- Records - MonthSelect Activity - view existing records
- Add New - FillInformation Activity - add new records
- Settings - Setting Activity - change settings
- Analysis - DataAnalysis Activity - view analysis graphs

As the launcher activity, it is responsible for initializing the database and SharedPreferences file (as explained in 4.1.5) during the first launch. It also requests camera permissions if they have not been granted before.

4.2.2 MonthSelect Activity

In order to find the desired records easily, the records are grouped by days, and days are grouped by months. This activity (Figure 4.5) is designed to let users select a month

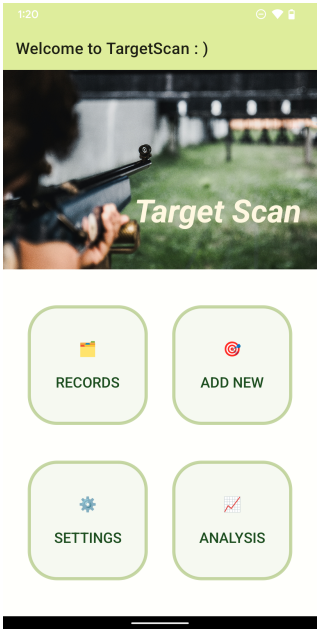

Figure 4.4: CoverPage activity

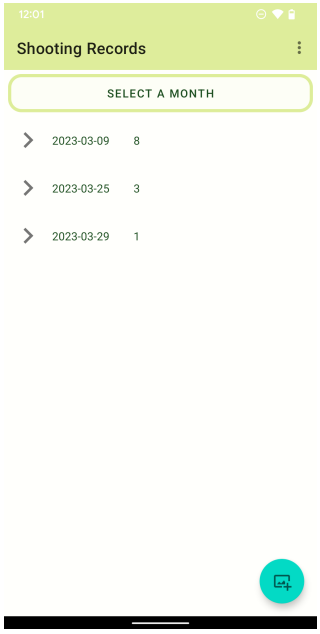

Figure 4.5: MonthSelect activity

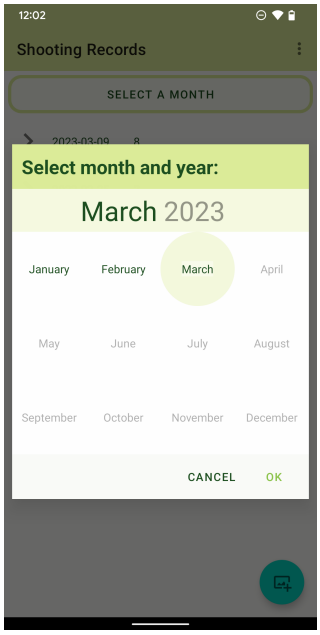

Figure 4.6: The picker in MonthSelect activity

and view days that have records in this month, and then they can click on a day to go to the DaySelect activity. The string value of date is also sent to the DaySelect activity, so it can display the desired records.

At the top of the activity, there is a button that allows users to select a month from a picker (Figure 4.6). The picker is an open source library called ”MonthYearPickerDialog”.[8] The main component of this activity is the RecycleView, which is a scroll list showing all days with records in the selected month. The list can be in either descending or ascending order, depending on the custom setting. If there is no record in the selected month, the list will be replaced by a TextView with the text ”No record in this month”. In the bottom right corner, there is a button to add new record.

In this activity, setupAdapter(year, month) function is used to add content to the scroll list. It retrieves all filenames from the image directory and applies a filter to scan images that have the given year and month values in their filenames. A dictionary ”dateCorr” is used to count the number of records for each day. Rows with the date and the number of records on that day are then inserted into the scroll list and displayed.

4.2.3 DaySelect Activity

This activity is designed to let users view records on a certain day (Figure 4.7). It has a similar scroll list, and a button at the bottom right corner for adding new records. Users can click on a record to view it in RecordDetail activity for more detail. The filename is also sent to the RecordDetail activity to display the desired record.

It has a similar setupAdapter() function as the one explained in above Section 4.2.2. Once the function gets the filtered list of filenames, it visits the SharedPreferences file to

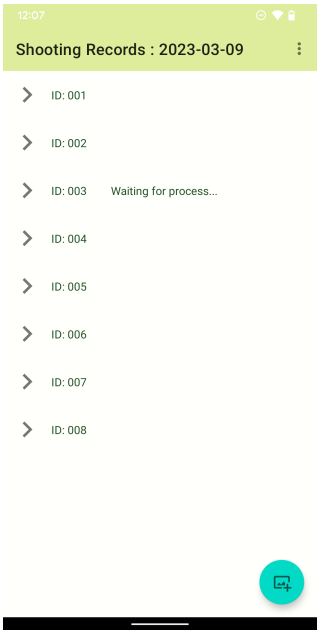

Figure 4.7: DaySelect activity

Figure 4.8: RecordDetail activity - unprocessed records

Figure 4.9: RecordDetail activity - processed records

check the processing status (processed or not) for each image. A dictionary ”photoCorr” is used to store the status for each image. Rows with the image ID and the status are then inserted into the scroll list and displayed.

4.2.4 RecordDetail Activity

This activity shows detailed information and graphs for a given record. At the top of the screen, there is an ImageView that displays the original target paper by default (Figure 4.9). When the activity is created, it receives a filename from the previous activity to load the image from directory. If the loaded image is vertically oriented, it will be rotated counterclockwise by 90 degrees. Below the image, a TextView lists basic information including discipline, date and comment. A SeekBar at the bottom allows users to load different images by dragging the thumb along a horizontal bar. From left to right, the options are: original target paper image, arrow/digit graph and cropped image of each target. The benefit of using a SeekBar is that it can provide multiple options in a simple UI component. My initial implementation used a bunch of buttons to select the image to display, and this made the UI look complex and messy. At the bottom, there are three buttons that correspond to the functions of deleting the record, editing comments and editing scores. If users attempts to delete the record, a dialog box will appear asking them to confirm their decision. ”Edit Comment” button is used to create the EditComment activity and ”Edit” button is used to create the PhotoProcess activity.

If the record has not been processed (Figure 4.8), the SeekBar will not be visible. In this case, the ”Edit” button is renamed to ”Process”, and clicking it will take users to

Figure 4.10: Arrow graph

Figure 4.11: Digit graph

the PhotoProcess activity.

The example in Figure 4.10 and 4.11 illustrates the arrow graph and the digit graph. The arrow graph consists of 10 arrows, each corresponding to a target locates on the same position. The length of the arrows represents the score (shorter arrow means higher score), and the direction of the arrows indicates the offset of bullet holes from target centres. If a bullet completely misses the target, an ”x” label is used, and if a bullet hole’s position is unclear (not detected), a ”?” label is used. The digit graph consists of 10 scores at positions corresponding to the targets. Both graphs are intuitive and easy to understand.

When arrow/digit graph or cropped image is selected, a toggle next to it becomes visible and it allows users to switch between two display modes. For arrow/digit graphs, the toggle switches between these two graphs. For cropped target images, the toggle switches between the rendered and original images. The default option is the rendered image (Figure 4.12) as it looks clearer, while the original image (Figure 4.13) is used as a backup for verification. However, if the user edits the target’s score and there is more than 1 point of difference compare to the algorithm-generated score, then only the original image can be displayed, as it indicates that the recognised bullet hole position is incorrect.

There are several graphs used in this activity, and they are generated by a Python script ”dataAnalysis.py” using the Chaquopy framework. I used the matplotlib Python library to create these graphs, because it is easy to use and the graphs look nice. The database contains a key called ”vector”, which stores a string containing all bullet hole offsets (as described in Section 4.1.5). This string is then sent to the Python function via a Chaquopy instance. Within this function, the string is parsed to extract all the offsets, and used to generate the graphs. The graphs consist of circles, dots, text, arrows, and

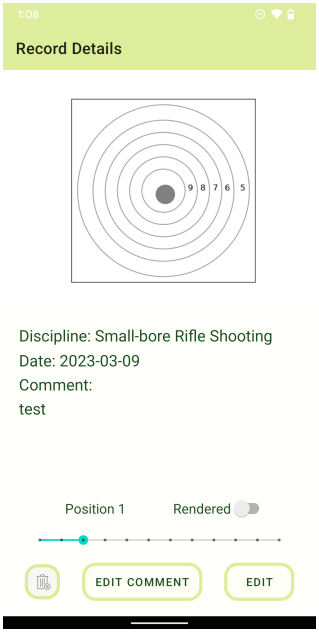

Figure 4.12: Rendered target image

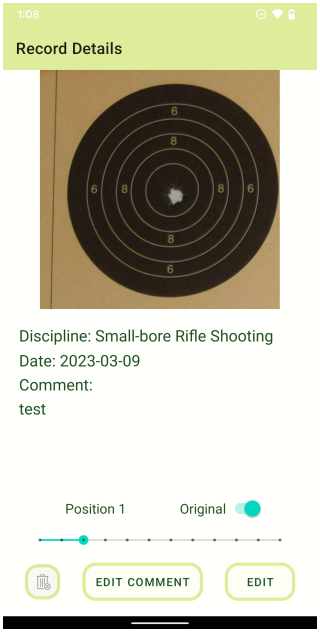

Figure 4.13: Original target image

appropriate matplotlib functions are used.

4.2.5 EditComment Activity

This activity is used to edit comment for records (Figure 4.14). It displays the basic information about the record, and provides a text box to edit the comment. Users can click on ”Cancel” button to cancel the operation or click on ”Confirm” button to save it. If the record is not processed, the comment is stored in the SharedPreferences file, otherwise it is stored in the database, so the activity needs to update the record in the correct location.

4.2.6 FillInformation Activity

This activity is the first step to add a new record (Figure 4.15). In this step, users need to select the date of the shot and enter comments (optional). The default date is set to the current date and the maximum comment length is 150 characters. Once completed, they can proceed by clicking on the ”Next” button which will take them to the TakePhoto activity. The information will also be sent to this activity. Since the screen space in TakePhoto activity is limited, some guidance on capturing proper photos is provided in this activity.

4.2.7 TakePhoto Activity

This activity prepares the photo for processing. The photo can be uploaded from an album or captured directly in this activity. At the top of the screen, there is some guidance to help users capture proper photos. Users will be prompted to grant camera

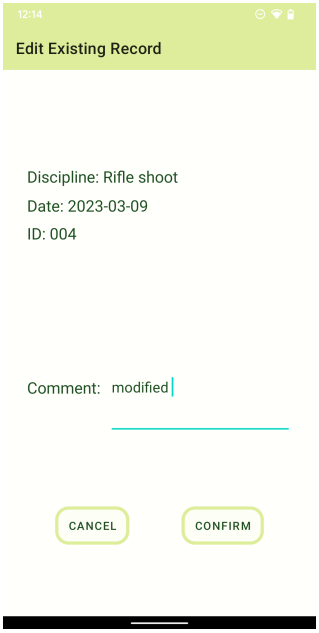

Figure 4.14: EditComment activity

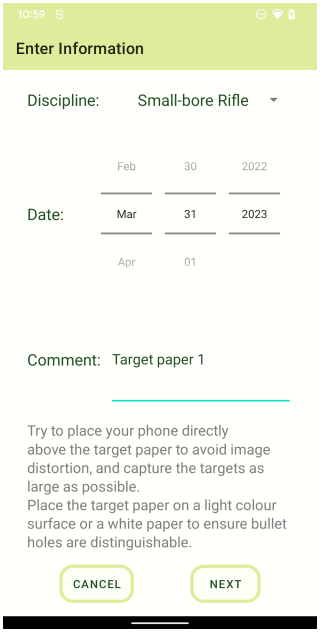

Figure 4.15: FIIIInformation activity

Figure 4.16: TakePhoto activity - camera feed on

Figure 4.17: TakePhoto activity - photo loaded

Figure 4.18: PhotoProcess activity - start process

permission if it has not been granted during the first launch. Once created, the activity (Figure 4.16) uses the camera API to display a live camera feed in the PreviewView (similar to ImageView but displaying a camera feed). There are 5 buttons at the bottom of the screen. The buttons in the first row are used to upload photo from albums or capture photos. The buttons in the second row are ”Cancel”, ”Rotate 90°” and ”Use it”. The last two buttons will only be enabled once a photo has been loaded, otherwise clicking on them will get a hint message. Clicking on the ”Cancel” button will take users back to the FillInformation activity, and the previously entered information will be automatically filled in.

To avoid having shadows on the papers, the phone’s flashlight is turned on while capturing the photo, and Figure 4.16 and 4.17 provides a comparison. This can allow the algorithm to perform well under different lighting conditions, which has been proved in evaluation chapter 5.2.2.1. After uploading or capturing the photo, the camera feed is turned off and replaced by an ImageView to display the photo (Figure 4.17). The photo is saved locally, and the ”Capture” button is renamed to ”Retake photo”. Clicking on this button restarts the camera and resets the activity to the original state. Discarded photos are automatically deleted, so no disk space is wasted.

It is important that photos are oriented correctly, because targets’ position is relevant for drawing graphs. To achieve this, users should align the phone’s long side with the paper’s long side, and ensure top side of the paper is on top of the screen. The purpose of having a ”rotate” button is to rotate the photo after it is loaded. Users should also ensure the target paper is placed on a light colour surface or a white paper, so the bullet holes are distinguishable. Once completed, users can click on the ”Use it” button to start the PhotoProcess activity, and the record will be saved.

4.2.8 PhotoProcess Activity

This activity has two uses, processing photos and editing scores for processed records. When it is created by other activities, an Boolean value ”editonly” is provided to indicate which mode should be used.

Processing the given photo uses all the functions of this activity. There are two buttons at the bottom (Figure 4.18), labelled ”Process later” and ”Process”. If users choose to process later, then the activity will be destroyed to exit.

Once users click on the ”Process” button, the process starts and the button is renamed to ”Waiting...”. It takes a few seconds depending on the hardware. A timer appears in the centre of the screen, to let users know that the application is still working in the background. Users can stop the process at any time using the ”Process later” button. The processing algorithm runs in an independent thread, so it doesn’t block the UI thread or the main thread.

After processing, the image is replaced with a labelled image (Figure 4.19) containing sequence numbers to indicate the corresponding relationship between targets and scores. The sequence number increases from top to bottom and left to right. Below the image, users can see the number of detected targets and their respective scores. The scores are displayed in EditText, and can be edited by users. A SeekBar is located at the bottom of

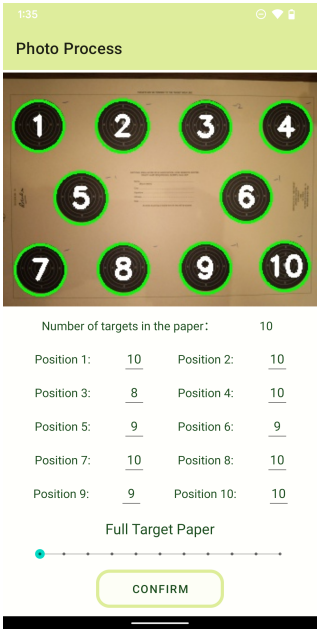

Figure 4.19: PhotoProcess activity - full target paper image displayed

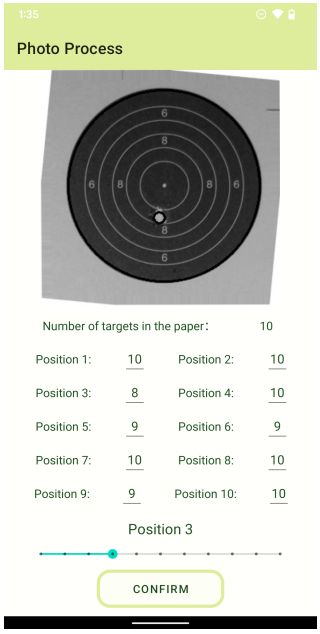

Figure 4.20: PhotoProcess activity - single target image displayed

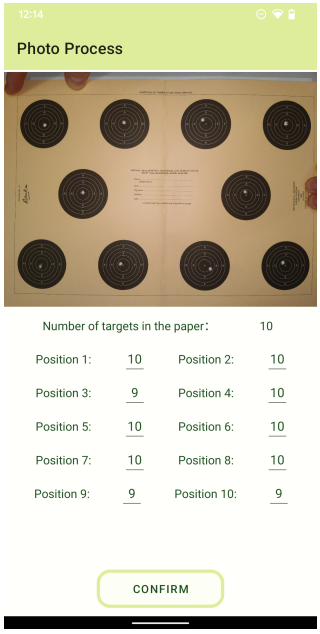

Figure 4.21: PhotoProcess activity - edit scores for processed records

the screen. Users can drag the thumb to view the labelled target paper image and the recognised target images. In the recognised target images (Figure 4.20), the target and bullet hole are circled. Users can check that the recognised result is correct and modify it if necessary. After confirming that the result is correct, users can save it by clicking on the ”Confirm” button. The record is added to the database, and the corresponding value in the SharedPreferences file is changed to ”Processed”.

As I explained in Section 3.1.1.3, the image processing algorithm was written in Python, and chaquopy framework is used to run it in the Android project. The implementation of the algorithm is written in file ”imageProcess.py” and explained in Section 4.3.

When the activity is used to modify the score for processed record, all processing steps are skipped. The original target image is displayed on top of the screen and the result is filled into the EditText (Figure 4.21). Users can modify it and update the new result to the database. The SeekBar is hidden as there is no cropped target image available.

4.2.9 DataAnalysis Activity

This activity generates bullet hole heatmaps based on past records, to help users improve their shooting skills. The UI and implementation are a bit similar to the RecordRetail activity, with an ImageView at the top and a series of images to display. A guidance section explains how to use this function, while two toggles in the centre of the screen allow users to specify whether they want to generate heatmaps based on time period or number of records, as well as whether they want to use targets from all positions or specific ones to generate heatmaps. Depending on shooter’s habit and posture, different target positions may have varying distributions for bullet holes. Therefore, generating

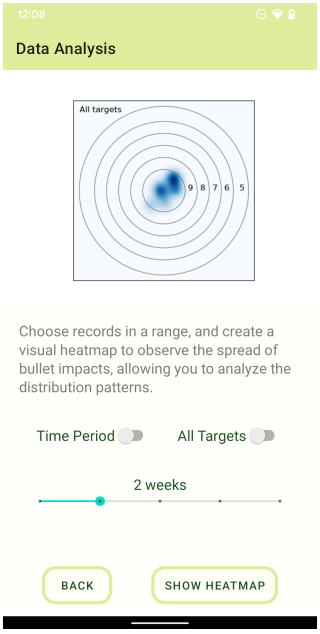

Figure 4.22: DataAnalysis activity example 1

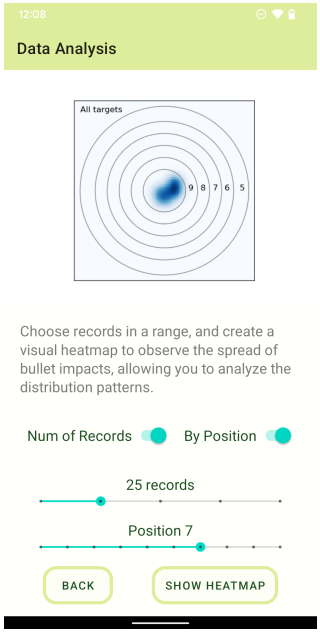

Figure 4.23: DataAnalysis activity example 2

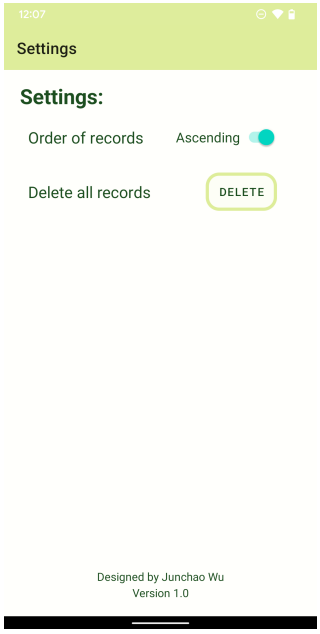

Figure 4.24: Setting activity

heatmaps for specific positions is useful. Two SeekBars below the toggles allow users to select a range of time periods or number of newest records (depending on the state of the left toggle) and a position (if a specific position is used to generate the heatmap, otherwise this SeekBar is invisible). The options are: 1 week, 2 weeks, 1 month, 3 months, or 10, 20, 50, 100 records, or all targets. Figure 4.22 shows a heatmap using all targets form records in last 2 weeks, and Figure 4.23 shows a heatmap using targets at position 7 from 25 newest records.

Due to the slight latency in generating the heatmap, in order to improve the user experience, users need to click on the ”Show heatmap” button after selecting options. If there are fewer records than the selected option (a number of records), a hint message will appear and all available records will be used.

This activity also uses Python matplotlib library and Chaquopy framework to draw the graphs, and the functions are written in ”dataAnalysis.py”. It initialises a 100*100 grid and counts the number of bullets located in each box. These numbers are stored in a separate matrix, and then smoothed using a Gaussian filter. Finally, plt.pcolormesh() is used to generate the graph and map the numbers to colours.

4.2.10 Setting Activity

This activity includes custom settings (Figure 4.24). Two functions are available. Users can change the order of records in scroll lists, and delete all records. If users attempts to delete all records, a dialog box will appear asking them to confirm their decision. It also contains copyright information at the bottom of the screen.

4.3 Image Processing Algorithm

4.3.1 Third-party Libraries

The following third-party libraries were used in the algorithm. They were included in the installation package, so users doesn't need to install them nor the python framework.

- opencv-python
- matplotlib
- numpy
- scipy

4.3.2 Target Recognition

4.3.2.1 Pre-process

Firstly, the photo is loaded using cv2.imread() and converted to greyscale for better processing efficiency. A colour copy is also saved, which will later be labelled with circles and text for users to view. The image is rotated counterclockwise by 90 degrees if it is vertical. A border is added to the image, because the targets with surrounding area will be cropped, and if a target is very close to the boundary, an out of boundary error may occur. Finally, the image is resized to a smaller image with 300 pixels wide while maintaining its aspect ratio. A smaller image still contains enough information while significantly reducing the processing time.

4.3.2.2 Image Conversion

These steps convert the greyscale image into a binary image and emphasise the targets, and it can significantly increase the accuracy of contour detection. These steps are performed on the original size image, because the processing time is very short and more information is helpful.

Firstly, adaptive threshold is used to binarise the image. There are two adaptive methods available in OpenCV library, mean thresholding and Gaussian thresholding. After testing with several images, I found that both methods can emphasise targets from the background and minimise the influence of shadows, but Gaussian thresholding doesn't have dark lines near the edge of the shadow and the background is cleaner. Figure 4.25 shows a comparison of two methods applied on the same target paper, which also reflects this phenomena. Therefore, I chose Gaussian thresholding. The complete parameters are cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 99, 1).

Secondly, cv2.medianBlur(img, 25) is used to clean the background. The Figure 4.26 shows the result that the grey background becomes disconnected black dots. Next, the black and white are inverted (Figure 4.27), so that the targets become white and background becomes black with white noise on it. It is followed by a morghological closing operation with a 5*5 ones kernel (Figure 4.28), and it can eliminate the small

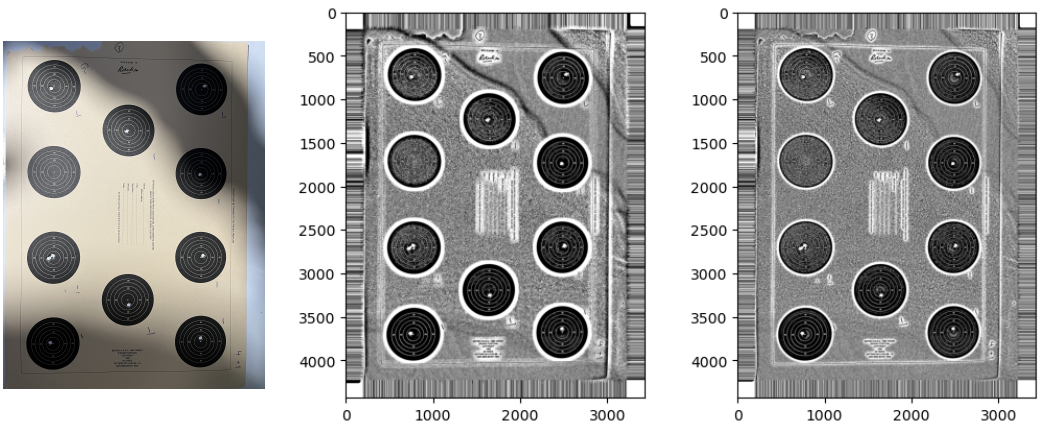

Figure 4.25: Comparison of mean thresholding (middle) and Gaussian thresholding (right)

black points on the white object[16]. Therefore, targets are cleaned again. Finally, cv2.medianBlur(img, 25) is used again to reduce small noise (Figure 4.29), and make the black border around the targets becomes wider.

4.3.2.3 Contour Detection

In this step, contour detection is used to find contours. The complete parameters are cv2.findContours(img, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE). cv2.RETR_TREE indicates that all contours will be retrieved, and cv2.CHAIN_APPROX_NONE indicates that all the boundary points will be stored[13]. This function returns all contours regardless of the size or shape, so it is normal to have hundreds of contours, and most of them are small noise. Therefore, several filters are used to find targets from the contours.

For each contour, I found its minimum enclosing circle using cv2.minEnclosingCircle(contour), and the centre position and radius of the circle are returned. If the radius is less than 0.05*width (300 pixels) or the contour area is less than 0.8*circle area, the contour is considered as noise. This filter effectively removes most false contours, but there may still be one or two contours that have similar round shapes and sizes but are not actual targets. The second filter is based on the idea that all targets have similar radii, calculates the IQR (Interquartile range = Upper Quartile – Lower Quartile = Q3 – Q1) of the radii from rest contours, and contours that satisfy $Q1 - IQR * 1.5 \leq radius \leq Q3 + IQR * 1.5$ are considered to be targets. However, if the photo is well captured, the IQR may be very small as the perspective distortion is negligible, so $IQR = max(1, IQR)$ is used to avoid this situation.

After applying these two filters, all targets are found and their radii and centre positions are known. However, the order of these contours in the list is random, so I need to reorder them in a fixed way to know their actual positions. Figure 4.30 shows the problem of directly using unordered contours, and in Figure 4.31 the targets are labelled in the expected order after reordering. To do this, I divided the image into small areas and scan them from left to right and from top to bottom. The contours are added to a new list as they are scanned and after that they are ordered.

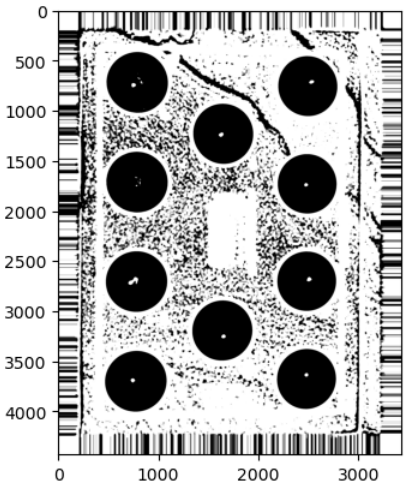

Figure 4.26: Image after the first median blur operation

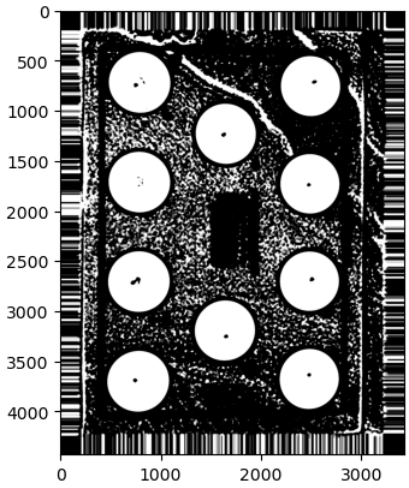

Figure 4.27: Image after inverting black and white

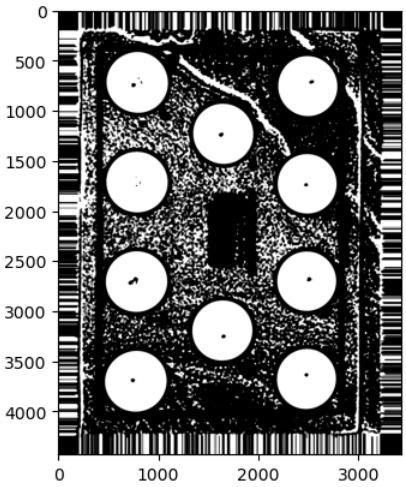

Figure 4.28: Image after the morgholog-ical closing operation

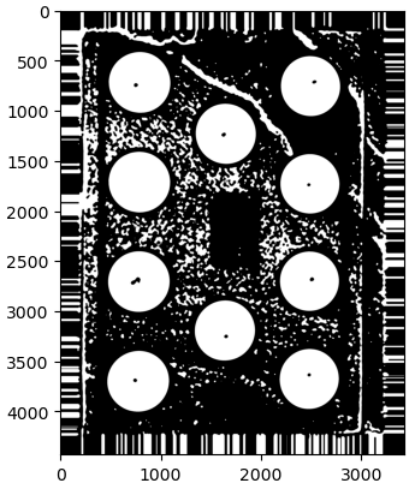

Figure 4.29: Image after the second median blur operation

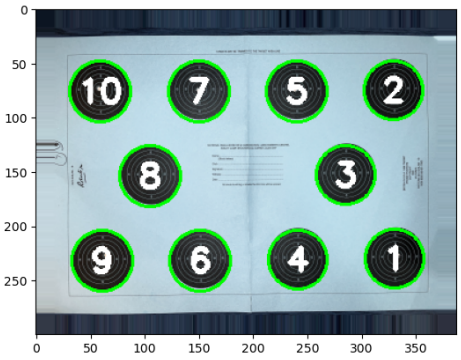

Figure 4.30: Contours before reordering

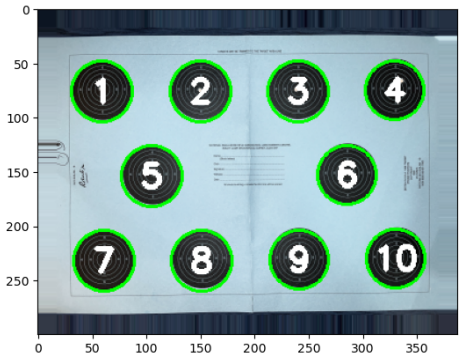

Figure 4.31: Contours after reordering

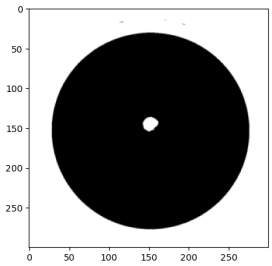

Figure 4.32: A cropped binary target image

4.3.3 Bullet Hole Recognition

4.3.3.1 Tilt Correction

After the target recognition process, the targets are identified. An additional step before cropping them is to correct the tilt. When the image is captured, the long side of the paper and the phone cannot be perfectly parallel, so the target paper image is tilted by small degrees. This needs to be corrected, because the offset of the bullet holes from the target centres are used to generate rendered target graphs and heatmaps, and the solution is to find the tilt angle and reverse it.

The coordinates of the centres of target 1 and target 4 (Figure 4.31 shows the sequence number of each target) are used to calculate the tilt angle. Connecting these two points, and drawing a vertical line through the centre of target 1 (the image is in vertical orientation), then the angle between the two lines is the tilt angle. Since the coordinates are known, an inverse trigonometric function can be used to calculate this angle. Finally, the target paper image is rotated by an inverse angle to correct it.

4.3.3.2 Crop Targets

Once the tilt is corrected, targets can be cropped from the image. The binary target paper image with original size is used so that the cropped target images are not too small. I mask the target with a slightly larger circle, so that only the target remains and the background is completely white, and then I can crop the target. I also add a border to the target to have more space for operations. A cropped binary target image is shown in Figure 4.32. Finally, I resized the cropped target images to 300 pixels wide while maintaining its aspect ratio.

4.3.3.3 Restore Ellipse to Circle

Due to the perspective distortion, the targets are not perfect circles, so the first step is to convert the ellipse shape back to circle shape. cv2.fitEllipse(contour) is used to fit an ellipse to the target, and major axis, minor axis and rotation angle are returned. Using these values I can perform rotation and affine transformation on the cropped target images, and the result is shown in Figure 4.33 (this is an extreme case and users should avoid capturing such images).

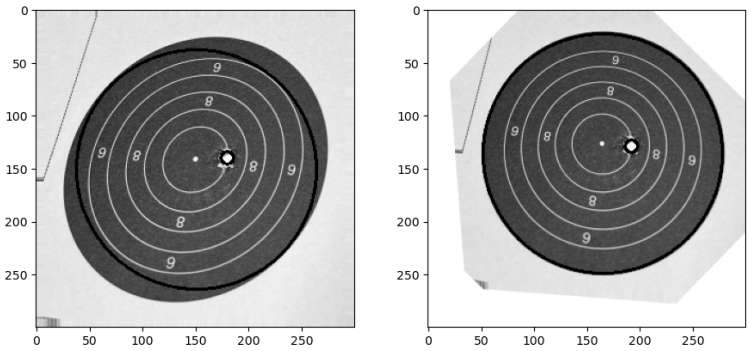

Figure 4.33: Images before/after using the 'ellipse to circle' operation

4.3.3.4 Eliminate Noise

Sometimes the cropped binary target image still contains unwanted white pixels. To remove them, a morphological opening operation is used, as it can remove noise[16]. Five kernels of size 7,13,19,25,31 are iterated in an ascending order. Larger kernels can remove larger area of white pixels, but the shape of the bullet hole also becomes rounder, making the estimation of the bullet hole centre less accurate. Therefore, if two contours are found, then the iteration is stopped, otherwise a morphological opening operation with a larger kernel size is applied on the original image (more than two contours indicates that noise still exists).

4.3.3.5 Contour Detection

It is similar to the contour detection step in target recognition (Section 4.3.2.3). Minimum enclosing circles of the contours is calculated, and a filter is used to accept contours with radius greater than 0.02*image width, radius less than 0.475*image width and contour area greater than 0.4*circle area. It can remove potential noise and an unwanted contour that has the same diameter as the image width. Two contours are left, the small one is the bullet hole and the large one is the target. The radius and centre position for the target is stored. For the bullet hole, I found that using a maximum circle inside the contour to represent the bullet hole is more accurate than the minimum enclosing circle, especially when the edge of the bullet hole is torn and the contour has a irregular shape. Its radius and centre position are stored for the bullet hole. Figure 4.34 is a comparison of two methods.

4.3.4 Scoring

Since the centres and the radii of the target and the bullet hole are known, the Euclidean distance from the target centre to the outer edge of the bullet hole can be easily calculated. By dividing the radius of the target in the image by the radius of the target in the real world, I can get the ratio and convert the distance to the real world scale. By comparing this distance with the radius of each ring (listed in Figure 4.35), a score is given to the target.

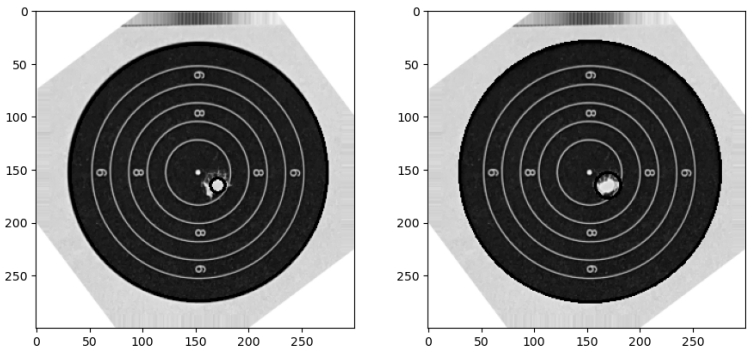

Figure 4.34: Comparison of two methods (left: maximum circle inside the contour, right: minimum enclosing circle)

APPENDIX A

TARGET DIMENSIONS

Sizes of scoring rings for edge touching on NSRA Targets. All dimensions are in millimetres. The theoretical dimensions have been rounded to two places of decimals. Prevailing atmospheric conditions may affect target paper leading to minor differences between printed and theoretical dimensions.

RIFLE TARGETS

1989 SERIES - SHORT RANGE

Scoring Ring	(Outward gauging - see Rule 5.1.7)			
Value	25 yards	20 yards	15 yards	25 metres
Centre Dot	1.00	1.00	1.00	1.00
1st scoring ring	12.92	11.45	9.99	13.60
2nd	20.23	17.30	14.38	21.60
3rd	27.55	23.16	18.77	29.60
4th	34.86	29.01	23.16	37.60
5th	42.18	34.86	27.55	45.60
6th	49.49	40.71	31.93	
7th	56.81	46.56	36.32	
8th	64.12	52.42	40.71	
9th	71.44	58.27	45.10	
10th	78.75	64.12	49.49	
Aiming Mark	51.39	41.11	30.84	56.20

Figure 4.35: The diameters of each ring[3]

Chapter 5

Evaluation

In this chapter, the implementation of the application and the image processing algorithm used are evaluated using several quantitative and qualitative analysis methods. In general, the application meets the design specifications and all functions are successfully implemented. It is also able to provide a pleasant and intuitive user experience. A simple user evaluation was carried out, and the feedback is summarised. Some limitations and possible extensions are discussed at the end of this chapter.

5.1 Application Benchmark

5.1.1 Response Time

The response time for most operations is negligible. One exception is that there is a delay of one or two seconds for the first graph to be generated. This is because the ”Chaquopy” Python framework takes one or two seconds to launch, and the graphs are generated using a Python script. Once the framework is launched, subsequent graphs will not experience the delay.

5.1.2 Error Handling

The application is thoroughly tested to ensure that users will not experience any crash under normal scenarios. Null object checks and try-catch structures are used where necessary to prevent unexpected crashes.

If users make mistakes, such as clicking on buttons that are temporarily disabled or entering incorrect content in text boxes (e.g. a score greater than 10), a hint message appears at the bottom of the screen explaining what was wrong.

5.1.3 Memory and CPU Usage

The following statistics were obtained on a Pixel 3 Android phone, which was released in 2018. This phone is equipped with a Snapdragon 845 chipset and 4GB of memory, running on the Android 12 operating system. Figure 5.1 shows the memory and CPU

Figure 5.1: Device profiler

usage over a period of time for this application. The application used around 576MB of memory, with peak usage reaching up to 652MB when processing images. For most idle activities, the CPU usage remained close to 0%, and the peak usage reached 13% when processing images. Therefore, this application doesn't require powerful hardware, and is capable to run smoothly on mainstream devices.

5.2 Image Processing Algorithm

5.2.1 Processing Time

The processing time varies depending on the CPU of the phone, so I used 5 different phones to process same images and recorded the time they used. The phones are: Samsung S23 Ultra, Samsung S22 Ultra, Samsung S21, Samsung S10, Google Pixel 3. These phones are released in different years from 2018 to 2023, so they can represent different levels of CPU performance over the years. Single core scores from Geekbench 5 are used to quantitatively indicate the CPU performance[1]. The Android Emulator used in Android studio IDE is also included for the comparison. Every phone is tested with 5 images and the average time used is calculated, to avoid error. The results are listed in the Table 5.2, and the chipset, release date of each phone are also included.

According to the experimental results, the processing time varies from 5 seconds to 15 seconds, and the correlation coefficient of the average processing time and CPU performance is -0.9893, indicating that they are negatively correlated and highly correlated. Considering that the average lifespan of Android phones is 2-3 years[9], most users could experience a processing time of less than 10 seconds, and the time will get shorter as time goes on. Even if users are still using old phones bought several years ago, a processing time of less than 15 seconds is still acceptable.

	Image 1 (seconds)	Image 2 (seconds)	Image 3 (seconds)	Image 4 (seconds)	Image 5 (seconds)	Average time used	Geekbench 5 Single Core Score	Chipset	Release Date
S23 Ultra	7	6	5	5	5	5.6	1508	Snapdragon 8 Gen 2	2023 Feb
S22 Ultra	7	6	8	9	9	7.8	1177	Exynos 2200	2022 Feb
S21	8	10	7	10	10	9	1110	Exynos 2100	2021 Jan
S10	13	12	11	11	11	11.6	758	Snapdragon 855	2019 Mar
Pixel 3	15	15	15	15	15	15	522	Snapdragon 845	2018 Oct
					Correlation:		-0.9893		

Figure 5.2: Average processing time for different Android phones[1]

5.2.2 Accuracy

The image processing algorithm consists of three parts. In the first part, the algorithm need to identify all the targets from the target papers. In the second part, the algorithm needs to identify bullet holes from the targets. In the third part, the algorithm needs to give correct scores. They are evaluated in following subsections. 13 target papers are used in the evaluation, and 128 targets are valid for scoring (the rest 2 targets are unused). The phone used for testing is Google Pixel 3, which has a 12.2-megapixel sensor with resolution 3024 * 4032 pixels.

5.2.2.1 Target Recognition Accuracy

This accuracy test was carried out twice under two different lighting conditions. The first test was performed in daylight, which is ideal for taking photos as there is no shadow on the paper and allowing shorter exposure time to avoid blurry images. In contrast, the second test was carried out at night under indoor lighting with only two bulbs on the ceiling. As a result, the room was relative dark, and when I took the images, my arms blocked some of the light causing obvious shadows to be cast on the paper. It represents a poor lighting condition.

According to the data presented in Table 5.3, this part of the algorithm successfully recognised all targets in 13 target papers under two lighting conditions. This experiment proves that using a phone's flashlight can minimise the influence of poor lighting conditions, and the algorithm is robust enough. As a result, the subsequent parts of the algorithm were only tested under the same indoor lighting condition. The 100% recognition accuracy also confirms the effectiveness of this part of the algorithm.

5.2.2.2 Bullet Hole Recognition Accuracy

This part of the algorithm successfully recognised all 128 bullet holes from 128 valid targets. These bullet holes were correctly circled in the cropped target images. The 100% recognition accuracy confirms the effectiveness of this part of the algorithm as well.

5.2.2.3 Scoring Accuracy

The algorithm was able to provide 124 correct scores out of a total of 128 targets, achieved an accuracy of 96.875%. The four incorrect scores are caused by the same

	Daylight	Indoor lighting				
	Recognised targets	Recognised targets	Valid targets	Bullet holes detected	Correct Scores	Comment
Target paper 1	10	10	10	10	10	
Target paper 2	10	10	10	10	9	
Target paper 3	10	10	10	10	10	
Target paper 4	10	10	10	10	9	
Target paper 5	10	10	10	10	10	
Target paper 6	10	10	10	10	10	
Target paper 7	10	10	10	10	9	
Target paper 8	10	10	8	8	8	Two targets are unused.
Target paper 9	10	10	10	10	10	
Target paper 10	10	10	10	10	9	
Target paper 11	10	10	10	10	10	
Target paper 12	10	10	10	10	10	
Target paper 13	10	10	10	10	10	
Accuracy			128		124	96.875%

Figure 5.3: Accuracy evaluation results

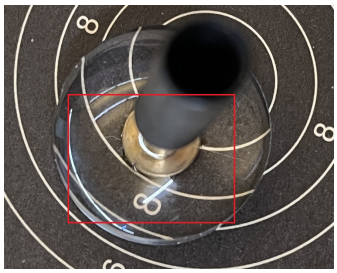

Figure 5.4: An target that was incorrectly scored

reason, that is the outer edge of the bullet hole is very close to a ring with a distance of less than 0.2mm, or overlaps the white line. Figure 5.4 shows an example that was incorrectly scored as 8, while the correct score is 9. The gauge has a magnifier, so the actual distance is even smaller than shown in the figure. In image recognition, the result cannot be that precise, so such an error is inevitable. Therefore, the third part of the algorithm is still considered to be successful and accurate.

5.3 Specification Evaluation

In this section, the application and the image processing algorithm are evaluated against the specifications listed in Section 3.2.

5.3.1 Image Processing Algorithm

The algorithm fulfils all the specifications set during the design stage. It can process target paper images fast and automatically, and provide scores with high accuracy. With the use of flashlight, the algorithm is also robust for poor lighting conditions. The recognition result is also consistent for the same image.

- **Capability:** Given a photo of the target paper, the algorithm is able to identify the targets and the bullet holes from the background, and provide scores along with other relevant information.
- **Determinism:** The output for the same image is consistent.
- **Accuracy:** The algorithm can recognise targets and bullet holes from the image and provide scores with high accuracy, and achieved 96.875% accuracy in the evaluation.
- **Robustness:** The accuracy and capability of the algorithm are not effected by poor lighting conditions, and perspective distortion and image tilt are also reverted by corresponding operations.
- **Speed:** Most phones take less than 10 seconds to finish the processing.
- **Automation:** Users don't need to provide additional information or input for the processing.

5.3.2 Application

The application fulfils all the specifications set during the design stage. It is easy to use and responsive to user operations. It is also well engineered to be compatible with most Android phones and didn't crash during the evaluation and testing. All functions discussed in 3.1.5 were implemented successfully.

- **Intuitiveness:** The user interface is easy to understand, and guidance are provided in the application to help users take proper photos. Therefore, no additional training is needed. Moreover, hint messages are appeared when users make incorrect operations or inputs.
- **Responsiveness:** The response time of most operations is neglectable. One exception is loading a graph for the first time, as explained in Section 5.1.1.
- **Functionality:** All functions designed for this application were implemented.
- **Robustness:** The application is carefully tested and does not crash under normal scenarios. Error handling structures were implemented when they were necessary.
- **Compatibility:** The application can run on more than 93% Android phones, and is not demanding for the CPU and memory.

5.4 User Evaluation

Due to time constraints, this project doesn't include a formal user evaluation. However, I still invited 5 students to use the application and provide feedback about the overall user experience and any confusion they experienced during the trial. This survey was compiled with GDPR regulations, and no data relating to the participants was stored. The feedback is summarised and listed below.

- All users were satisfied with the accuracy of the recognition and the design of the user interfaces.
- No crash occurred during the tests.
- Two students mentioned that it would be helpful to have a tutorial or boarding pages in the first time of launching the application, to familiarise themselves with the features of this application.
- One student mentioned that waiting for the processing was boring when he had several target papers to add at the same time. He suggested that the images could be processed in the background.

5.5 Project Limitation

Though the project fulfils all the specifications and is considered to be successful, there are still some limitations. They are discussed below.

- The guidance in this application is relative simple. I put some notes in the user interfaces where I thought the operations required extra caution. If users read the notes, they can smoothly use these functions. However, some users may ignore the text in the application, or fail to understand the meaning if they are not familiar with the technology. A better solution is to use some illustrations or animated GIFs instead of text.
- The processing time for most phones is less than 10 seconds, which is good, but users have to wait in the activity for the processing to finish. It would be better if the processing could be done in the background, while users can add new records or browse other records at the same time.
- Due to time constraints, only a simple user evaluation was carried out and 5 students were participated. User feedback is valuable for improving the user experience, so it would be nice to have a formal user evaluation with more participants and collect questionnaires from them.

5.6 Future Work

In this section, I discussed some potential improvements for this application.

- **Better guidance:** Provide a boarding activity to introduce the features of this application. Replace the text guidance in the application with illustrations or GIFs to help users understand the usage of some functions.
- **More shooting disciplines support:** This application only supports one type of the target paper, which is used for 25-yard .22 small-bore rifle shooting. The image processing algorithm needs to be adjusted to support other types of target paper.

- **Multi-threads processing:** Rewrite image processing algorithm functions to support multi-threads processing. A shorter waiting time is always better for users.
- **More data analysis features:** More data analysis techniques can be used to extract information from the records. This information can help users to better understand their shooting skills.
- **Storage on clouds:** In this application all the data is stored locally, so there is a risk to lose the records if the phone is broken. Third-party cloud services such as OneDrive and Dropbox can be integrated into the application to provide backup solutions.

Chapter 6

Conclusions

6.1 Project Achievements

The following achievements are made in this project.

- Provided a detailed literature review on the topic of target recognition and automatic scoring.
- Provided background readings about the rifle shooting sport and some image processing techniques.
- Obtained a lot of experience with image processing techniques, OpenCV library and Android development.
- Created an image processing algorithm with high accuracy to recognise targets and bullet holes from the background, and then automatically score them.
- Created a data analysis tool that uses shooting records to generate heatmaps of the bullet holes to help users improve their shooting skills.
- Created an Android application for 25-yard .22 small-bore rifle shooting that contains automatic scoring, recording and analysis features.
- Designed intuitive user interfaces for the application, and provided a pleasant user experience.
- Provided a detailed evaluation for the application and the algorithm.

6.2 General Remarks

The implementation of this project, including the application and the image processing algorithm, was thoroughly discussed in this report. All of the functionalities mentioned in the project description and all of the specifications set during the design stage were achieved, and the evaluation showed high accuracy of the algorithm and great usability of the application. Although there are some limitations that can be improved, the overall

completeness of this project is considered high. Therefore, this project is successfully completed.

Bibliography

[1] Geekbench 5. Android benchmarks, 2023. https://browser.geekbench.com/android-benchmarks, Last accessed on 2022-3-30.

[2] Peb Aryan. Vision based automatic target scoring system for mobile shooting range. pages 325–329, 01 2012.

[3] National Small bore Rifle Association. Rules, 2022. https://www.nsra.co.uk/index.php/home/reference/rules, Last accessed on 2022-3-18.

[4] Chaquopy. Chaquopy - python sdk for android, 2023. https://chaquo.com/chaquopy/, Last accessed on 2022-3-27.

[5] Redford Rifle Club. Precision small-bore rifle, air rifle and air pistol target shooting are 'olympic sports', 2022. https://www.bedfordrifleclub.co.uk/introduction, Last accessed on 2022-3-18.

[6] Denis. Converting rgb to grayscale/intensity, 2022. https://stackoverflow.com/questions/687261/converting-rgb-to-grayscale-intensity, Last accessed on 2022-3-18.

[7] Penghua Ding, Xuewu Zhang, Xinnan Fan, and Qianqian Cheng. Design of automatic target-scoring system of shooting game based on computer vision. In *2009 IEEE International Conference on Automation and Logistics*, pages 825–830, 2009.

[8] Lakisau Dzmitry. Monthyearpickerdialog github repository, 2022. https://github.com/Dzmitry-Lakisau/MonthYearPickerDialog, Last accessed on 2022-3-24.

[9] Simo Elalj. How long does a smartphone last?, 2023. https://www.refurb.me/blog/how-long-does-a-smartphone-last-replacement-cycle, Last accessed on 2022-3-31.

[10] Akula Hemanth. Contours in images, 2020. https://medium.com/swlh/contours-in-images-a58b4c12c0ff, Last accessed on 2022-3-18.

[11] Kotlin. Kotlin for android, 2023. https://kotlinlang.org/docs/android-overview.html, Last accessed on 2022-3-27.

[12] Yuan-Chi Lin, Shaou-Gang Miaou, Ying-Cheng Lin, and Shih-Lun Chen. An automatic scoring system for air pistol shooting competition based on image recognition of target sheets. In *2015 IEEE International Conference on Consumer Electronics - Taiwan*, pages 140–141, 2015.

[13] OpenCV. Contour approximation method, 2023. https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html, Last accessed on 2022-3-18.

[14] OpenCV. Home - opencv, 2023. https://opencv.org/, Last accessed on 2022-3-27.

[15] OpenCV. Miscellaneous image transformations, 2023. https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html, Last accessed on 2022-3-18.

[16] OpenCV. Morphological transformations, 2023. https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html, Last accessed on 2022-3-27.

[17] OpenCV. Opencv - adaptive threshold, 2023. https://www.tutorialspoint.com/opencv/opencv_adaptive_threshold.htm, Last accessed on 2022-3-18.

[18] Civilian Marksmanship Program. Cmp smallbore program, 2023. https://thecmp.org/competitions/cmp-smallbore-program/, Last accessed on 2022-3-27.

[19] Open Source Repository. Android api levels, 2023. https://apilevels.com/, Last accessed on 2022-3-27.

[20] Shanklin Rifle and Pistol Club. History of smallbore rifle, 2020. https://shanklinrifleandpistolclub.org.uk/introduction-to-shooting/history-of-smallbore-rifle, Last accessed on 2022-3-18.

[21] Adrian Rosebrock. Opencv morphological operations, 2021. https://pyimagesearch.com/2021/04/28/opencv-morphological-operations/, Last accessed on 2022-3-18.

[22] Abhishek Sharma. Blurrings in cv2, simple blur,box blur,gaussian blur and median blur, 2023. https://towardsdev.com//blurrings-in-cv2-simple-blur-box-blur-gaussian-blur-and-median-blur-2d8718f8b2a8, Last accessed on 2022-3-18.

[23] NSRA Shop. Nsra .22a scoring gauge, 2023. https://www.nsrashop.co.uk/collections/scoring/products/nsra-177-22-scoring-gauge?variant=1001187023, Last accessed on 2022-3-27.

[24] Miniature-Calibre Rifles Research Site. 25 yards target paper image, 2023. https://www.rifleman.org.uk/Images/NSRA%201989%2025yd.jpg, Last accessed on 2022-3-27.

[25] Parama Widayaka, Hendra Kusuma, and Muhammad Attamimi. Automatic shooting scoring system based on image processing. *Journal of Physics: Conference Series*, 1201:012047, 05 2019.

[26] Wikipedia. Median filter, 2023. https://en.wikipedia.org/wiki/Median_filter, Last accessed on 2022-3-18.

[27] Cuiliu Ye and Hong Mi. The technology of image processing used in automatic target-scoring system. In *2011 Fourth International Joint Conference on Computational Sciences and Optimization*, pages 349–352, 2011.