Journal of Biomedical Science and Bioengineering

MoFlus: An Open-Source Android Software for Fluorescence-Based Point of Care

Panji Wisnu Wirawan*, Adi Wibowo

Department of Informatics, Diponegoro University, Indonesia

**Coresponding author : panji@lecturer.undip.ac.id*

ABSTRACT. High-sensitivity fluorescence-based tests are utilized to monitor various activities in life science research. These tests are specifically used as health monitoring tools to detect diseases. Fluorescence-based test facilities in rural areas and developing countries, however, remain limited. Point-of-care (POC) tests based on fluorescence detection have become a solution to the limitations of fluorescence-based tools in developing countries. POC software for smartphone cameras was generally developed for specific devices and tools, and it ability to select the desired region of interest (ROI) is limited. In this work, we developed Mobile Fluorescence Spectroscopy (MoFlus), an open-source Android software for camera-based POC. We mainly aimed to develop camera-based POC software that can be used for the dynamic selection of ROI; the number of samples; and the types of detection, color, data, and for communication with servers. MoFlus facilitated the use of touch screens and data given that it was developed on the basis of the SurfaceView library in Android and Javascript object notation applications. Moreover, the function and endurance of the app when used multiple times and with different numbers of images were tested.

Keywords: Point of Care, Android, SurfaceView, Fluorescence, ROI
Article History: Received 06 July 2021; Received in revised form 21 December 2021; Accepted 26 December 2021; Available online: 28 December 2021
DOI: 10.14710/jbiomes2021.v1i2.39-45

1. INTRODUCTION

High-sensitivity fluorescence-based tests are widely used to monitor various activities, such as molecular dynamics and interactions; enzymatic activity; signal transduction; cell health; and molecule, organelle, or cell distribution, in life science research[1]. Various fluorescence-based health condition monitoring tools have been useful for resolving disease detection problems. These tools include real-time thermal cyclers, fluorescence microscopes, and fluorescence spectrophotometers. They are used for various purposes, such as quantitative gene expression analysis, SNP analysis, fluorescence immunoassay, and drug target validation, as well as the genotypic quantification of antigens, such as viral particles and bacteria[2]. However, facilities for these analyses are limited in remote areas and in developing countries. Diseases that are often faced by developing countries include diabetes, heart disease, cancer, malaria, pneumonia, diarrhea, and HIV/AIDS. These diseases are estimated to cause the deaths of more than 15 million people each year. Fluorescence-based point-of-care (POC) tests can be a solution to the lack of facilities in developing countries. The contribution of POC tests to the health sector has exerted a crucial impact on disease prevention or detection. Previous work has focused on developing simple, inexpensive, and reliable POC tools for rapid diagnosis. POC devices involve the use of affordable, portable, and efficient materials. They can be easily transported and can be used at any time to meet the needs of rapid health care. The development of smartphone technology has enabled the construction of fluorescence-based and smartphone-based POC for disease detection[3].

The smartphone camera has a dominant role in disease detection[3]. Smartphone cameras can be used not only to capture images but also to process images and detect objects in the captured image. Smartphones have been selected as POC tools given their wireless connectivity, high-resolution photography, and good portability [4]. Camera-based POC devices have undergone extensive development in recent years. In 2015, the Ozcan research group[5] developed a smartphone-based POC device for reading fluorescent signals in enzyme-linked immunosorbent assays for measles, mumps, and herpes (HSV-1 and HSV-2). The developed software could be used to monitor 96 walls by observing 15 pixels with a fixed radius in each wall through the thresholding method. The whole image is then analyzed through machine learning outside of the smartphone. Machine learning frameworks such as Learning-to-rank can be used to analyze the resulting image[6]. An enzyme-free nucleic acid amplification method for detecting influenza virus DNA and miR-316 microRNA was developed in 2018[7]. In 2017, Priye et al.[8] developed a POC fluorescence device based on reverse-transcription loop-mediated isothermal amplification for ZIKA virus detection. The developed software (LAMP2GO) could be used to control focal length, exposure time, and ISO during image acquisition. Moreover, sample selection from multiple regions of interest (ROIs) could be conducted after image acquisition. In 2017, Chen et al.[9] developed a smartphone-based POC device with the help of long pass filters and macro lenses with magnifications of up to 12.5×. Chen did not use viruses that attack humans as inputs and instead applied viruses that attack horses (S. Equi, S. Zoo, EHV-1, and EHV-4). The output of the device is fluorescence. In the same year, Jalal et al.[10] developed a paper-based reagent strip POC device with a lab-on-chip scheme with urine as an input for the detection of urine components and characteristics, including glucose, protein, pH, and red blood cells, through hue colorimetric information extraction. Song et al.[11] developed a smart cup system in 2018. They conducted image processing analysis to detect bioluminescence signals emitted by the ZIKV virus in urine and HIV in the blood. Lin[12] combined dsDNA with SYBR Green1 to detect green fluorescence signals emitted by streptomycin in food. Chan et al.[13] utilized a repurposed 3D printer to extract and purify MP-based NA to detect ZIKV in human urine samples. In this assay, fluorescence signals are emitted upon the irradiation of sample tubes with blue LED. The signals are then recorded by a smartphone camera with an orange plastic filter. Akraa et al.[14] developed a POC device for the detection of chronic kidney diseases based on urine albumin analysis. It also used a smartphone device to detect fluorescence.

The development of POC technology focuses on the use of smartphone cameras. Smartphone cameras on POC devices can be operated by using default or specific applications that can control extreme focal lengths, exposure time, and ISO settings Berg et al.[5] captured images by using the NOKIA Lumia 1020 application. Priye et al.[8] developed applications that could control focal length, exposure time, and ISO. However, smartphone-based POC technologies are specific for certain devices and tools, and their ability to choose the desired ROI is limited. Thus, users or medical personnel are unable to select the desired ROI. In addition, most of the developed POC devices have a limited number of points. These technologies also cannot dynamically determine the number of points. POCs are also limited by their inability to store and process data and to apply data as a repository for the source of decisions. Thus, each POC device must take measurements of its own.

In this work, we developed a universal application for a camera-based POC device. We mainly focused on developing camera-based POC software to complement the limitations of existing applications. The technology was developed on the basis of SurfaceView libraries in Android applications. SurfaceView enables the dynamic development of multitarget detection by using touch screens. This paper was organized in reference to Chondros et al.[15] Part 2 begins with a system description of the POC system. This section presents the required requirements and design details of the system. The implementation and application testing of the system are discussed in succeeding sections.

2. 2. System Description

2.1 Requirement

Mobile Fluorescence Spectroscopy (MoFlus) is an Android-based application that can read color expression values from the observations of samples on the basis of color. The smartphone camera is used to acquire images of samples that will be perceived as expression level values. The user can determine the spots to be selected for the color expression value. The results of observing the selected color values will be saved. The fluorescence of a sample might be taken as a color expression value. The color expression processing of images can be performed in real time or in batch. In real-time processing, the image of the observed object is obtained directly from the camera and is utilized to support the retrieval of reaction data from the sample. The user needs to specify a certain range of duration for image acquisition with MoFlus to obtain the color expression value of the sample. The changes in the fluorescence intensity of the sample can be observed by using the time-series data obtained through real-time observation. In batch processing, previously acquired photographs are subjected to color extraction. Thus, this strategy can be used to analyze various samples from different photos.

MoFlus outputs color expression in the form of red green blue (RGB) and hue saturation value (HSV) color models. These results can be stored and can be sent for further processing given the computation limitations of smartphones.

MoFlus has the following requirements:

a) More than one observation point should be determined to enable the observation of additional samples. In contrast to taking observations from only one sample or one photo at a time, taking observations from numerous samples prevents the excessive repetition of observations.
b) Multiple points should be observed simultaneously. Numerous samples should be observed simultaneously. Simultaneous processing can be conducted with multithreading programming techniques.
c) Color expression should be captured with RGB and HSV color models. The choice of different color expressions allows the user to select the color expression that is in accordance with the analysis that must be performed next.
d) Color expression must be captured through real-time live imaging and batch imaging from previously acquired images.
e) Observations that are ready for further processing must be saved such that they can be taken at any time when needed in the future.

The defined requirements are then used as the basis for software design by mapping requirements into software components**.**

2.2.Design

This section focuses on the design of the components in MoFlus. The representation of components and their relationships is illustrated in Figure 1. Architectural depictions were generated by using component diagrams with connectors and connections in reference to Ozkaya[16].

The constructed components are derived on the basis of the previously described requirements. Each requirement represents a component. Thus, four main components for handling smartphone camera access, data modeling, storage processing, and data transmission to the server exist. Each successive component is called the Detection Camera, DetectionModel, DetectionStorage, and Server. The four components are organized into a component called the Main component, which is also tasked with real-time or batch color expression processing. The Helper component is used to facilitate computation by other components. It is responsible for facilitating various processes, such as image processing and CSV file formation. Figure 1 illustrates the overall MoFlus architecture.

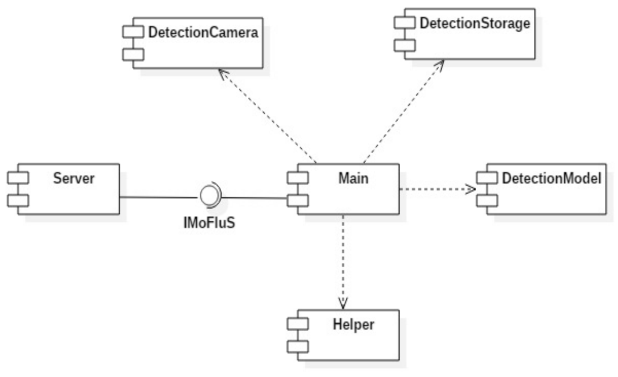


Figure 1. MoFlus Architecture

Each component can be described as follows:

a) DetectionCamera manages detection with the camera by utilizing the SurfaceView Android library. SurfaceView is used to shoot media per frame in real-time mode or per image in batch mode. RGB or HSV values are taken in this component, wherein each frame of the captured image will be changed to the bitmap matrix form. Each location of the sample ROI point will be squared in accordance with the specified size, and color expression value is calculated.
b) DetectionModel is used for components that accommodate all required modeling classes, such as sample ROI position, data samples, color models, observational graphs, and several classes needed for data storage and processing in the cloud.
c) DetectionStorage regulates storage. The local and online storage media used by MoFlus are SQLite and MongoDB Database, respectively.
d) Server regulates the transmission of observations to classification servers by making observations in the form of Javascript object notation (JSON).
e) Helper facilitates image conversion, folder management, and CSV formation. This component is a new component. Although it is not a requirement, it is useful for providing computation facilities to other components.

Each component will be assembled and employed by the Main component on Android to shape overall software performance. Each component will communicate with each other in accordance with their specific functions, which are coordinated by the Main component. Communication among each component is described in Figure 2.

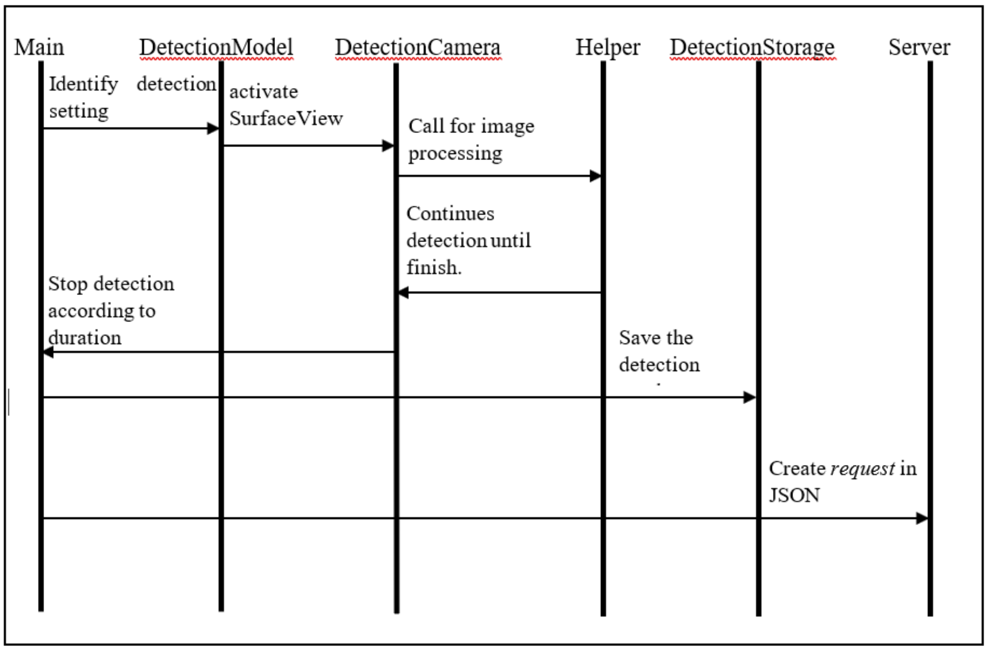


Figure 2. Interaction between components

As depicted in Figure 2, the Main component acts as the main coordinator that will call on the other components to take color descriptions. As stated in the above description, the DetectionModel, will be called by the Main component to identify detection settings, such as color selection and sample number settings. The Main Component will access and activate SurfaceView setting on the DetectionCamera component to handle image acquisition in the real-time mode. Therefore, the batch mode does not require SurfaceView settings for image processing. Image processing during observation will be assisted by Helper and DetectionCamera components. The Helper component provides assistance to image processing by the DetectionCamera component. Processing will be taken within the specified period (finish time). Observations will be stored by the DetectionStorage component. Detection results can also be processed through transmission to an external server with Server components, which are not discussed in detail in this paper.

2.3.Design for Multisample Observation

This work presents a technique for how a software application can observe numerous samples for obtaining color expressions. The application is designed to be implemented specifically as an Android-based application.

MoFlus users can make observations by selecting desired spots from the sample objects observed on the smartphone screen. The user selects a sample by selecting (touching) a sample of observations on display. The pixel sizes of observations can be set from 1 pixel to 200 pixels. Many of these points are also selected for flexible observation. Thus, the user can change previously selected observation points.

The image acquisition period can be set in milliseconds in real-time mode. The duration of the observation can also be set in addition to the shooting interval to enable the application to observe the reaction movements of the observed samples. The observations are displayed as a line chart to show the changes in the color intensity of the observed sample. The GraphView library on Android is used to develop these features.

SurfaceView is an Android library class that is derived from the View class. This class is used because of its ability to update the screen quickly. The ability to update the screen quickly is needed because real-time observation requires continuous and simultaneous image acquisition and processing specific areas in the image until the end of the set duration. If this capability is only charged to the View class, the processing will be heavy because the View class is a graphical user interface (GUI) thread that handles all user interactions.

Using only the SurfaceView library will fail to meet the requirements of MoFlus because sample image processing is required. SurfaceView's ability to handle further sample image processing must be improved. The SurfaceView class used by MoFlus is an extension class that was specifically developed to enable SurfaceView to handle sample image processing. The extension class is named DetectionSurfaceView (Figure 3). In addition to the capabilities for image processing, the DetectionSurfaceView class is also designed to have the ability to show the sample spots chosen by the user.

DetectionSurfaceView overrides several standard methods in the SurfaceView class. These methods include onMeasure, onDraw, and onPreviewFrame. onMeasure controls the resolution of SurfaceView on smartphone screens. onDraw is used to draw spots on SurfaceView. The onPreviewFrame is overriden to enable the supporting DetectionHelper to acquire and process images. DetectionHelper is created to help several processes during image acquisition and processing.

Other methods, such as changeDotPosition, startDetection, and startAutoFocus, are added to extend the capabilities of SurfaceView. changeDotPosition is used to move the position of observation points. startDetection is used to initiate observation, and startAutoFocus is used to direct the camera to capture images to the point of observation

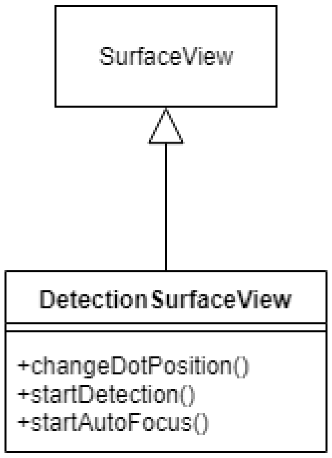


Figure 3. DetectionSurfaceView is inherited from SurfaceView

3. IMPLEMENTATION

This section focuses on implementation, especially the implementation of designs for multisample observations and the storage of observations. Implementation is carried out using the Java programming language on the Android version of the SDK minimum of 19. The application is deployed on a Samsung Galaxy J5 Pro smartphone. Our source code is available on Github[17] with the address https://github.com/bowoadi/MobileFluorescenceSpectroscopy.

Implementation is completed by making components that are important for object detection. These components include DetectionCamera, DetectionModel, Helper, and Main. DetectionStorage and Server were created after the development of the components for object detection. Each component contains important classes or key classes, which will be discussed in the next section. We do not use semiautomatic method[18] to select the key classes, but we select the important classes manually. The discussion follows the order of development and starts from sample observation and continues to saving results.

3.1.Sample Observation

3.1.1.Sample Selection

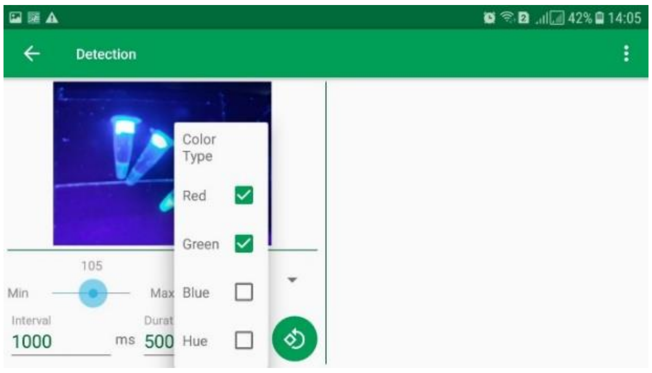
MoFlus has the ability to observe multiple samples on a smartphone screen display. The user selects the spots on the samples to be occupied for the acquisition of color expression value, and the location of the observation points can be stored for further observations. Prior to observations, some parameters need to be set, such as the duration of observation, time interval of observation, pixel size, and color type to be observed (RGB or HSV). The display for the observation setting is shown in Figure 4 [a].

A preview picture will be displayed through a dialog message at the time of observation (Figure 4 [b]). The preview picture is used to select the sample to be observed. The user selects the sample by touching the smartphone screen. The selected samples will be marked with points. These points are formed by using custom drawing, which is an approach for drawing certain patterns on an object. This approach utilizes the Android Canvas and Paint library. The Canvas object is used to set what will be drawn, whereas the Paint class sets image properties, such as color, thickness, and stroke.

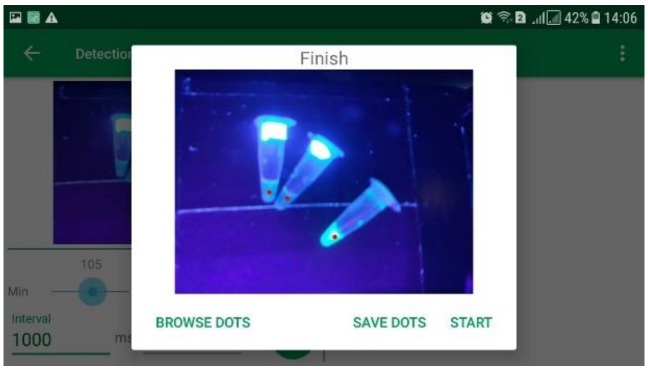
A preview image is generated in Bitmap format to support the use of Canvas. The Copy method is used to remove the effect of spot position on preview images. The succeeding images will note the effect on the basis of spot position. The copy image from the Copy method is converted to ARGB_8888 format to obtain the best image results. This format can store each color channel in 8 bits. Code construction is shown in Figure 5.

Object paint (Figure 5) is used to adjust the color of the dot formed when the user selects the observed dots. The color settings use setColor, where the color used is red. A dot is formed when the user selects the observation point. It is made in the form of a circle with a radius that is calculated as the height of the preview image divided by 100 to ensure that the dot image that is formed is visible such that the selected location is clearly visible to the user. The user-selected location is stored in the DotLocation class, wherein several attributes for storage locations exist. The drawCircle method that is set in accordance with the obtained location and other properties, such as the Paint object and variable radius, is used to draw the dot. This process sequence is completed by setting the imageView with an image that has been given a detection location.

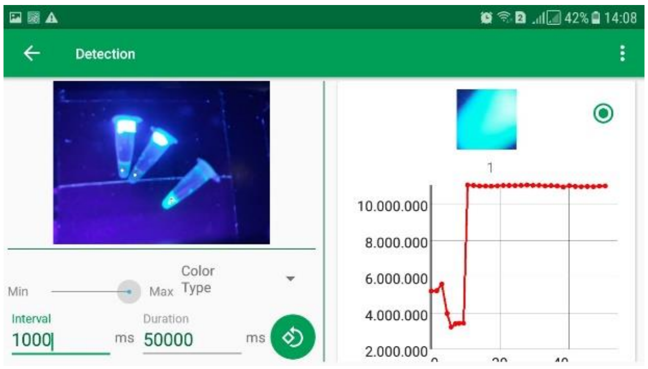
The user selects the sample observation location in a preview image. The preview image is an image that has been transformed (resized) from the original image. The transformation of original images into preview images involves an Android library from the android.graphics.Matrix package.



[a]



[b]



[c]

Figure 4. Observation in the MoFlus application

```
Bitmap previewPicture =
  ((BitmapDrawable) imageView.
  getDrawable()).getBitmap().copy(
  Bitmap.Config.ARGB_8888, true);

Canvas canvas = new
  Canvas(previewPicture);

Paint paint = new Paint();
```

Figure 5. Preview of picture construction code

```
paint.setColor(Color.RED);
canvas.drawBitmap(previewPicture,
  new Matrix(), null);

int radius =
  CameraSettings.PREVIEW_HEIGHT/100;

canvas.drawCircle(dotLocation.getX()
 ,dotLocation.getY(), radius,
 paint);
imageView.setImageBitmap(
 previewPicture);
```

Figure 6. Additional settings for Paint and Canvas objects

```
public void onClick(View view) {
  . . .
  detectionSurfaceView.
  startDetection( countdown_textview,
  start_delay);
  . . .
}
```

Figure 7. StartDetection method to start detection

```
if (((int) imageView.getTag()) ==
   graphDatas.size())
{
 //code for additional settings.
 //startDetection method called here
}
```

Figure 8. Checking before detection begin

```
dataResult.getGraphDatas().
  clear();

dataResult.getGraphDatas().
  addAll(graphDatas);

detectionHelper.setDataResult(
  dataResult,
  getApplicationContext());

enableProcessProgress();

pixel_location_imageview.setImageBitmap(
  BitmapHelper.
  getBitmapPixelLocation(
  dotLocations));

dialog.dismiss();
```

Figure 9. Additional settings to start detection

OnTouch will record the dot position, and OnClick will identify the dot position for the selected sample and draw a red dot in accordance with the chosen position during the selection of the detection location in the preview image (Figure 6). The drawing work shown in Figure 7 and Figure 8 is an implementation of onClick. Detection starts when the location has been selected and the user presses the "Start" button in the dialog message. Calling the startDetection of the DetectionSurfaceView class will initiate detection. Several components related to the components on the detection page, namely graph data, must be checked prior to detection.

Detection locations are selected on the basis of the prearranged sample requirements. Each sample has one detection location. During implementation, the sample is reallocated into a class called Sample. Checking the suitability of the sample selected with the graph.imageView.getTag() before detection yields the memory that is obtained when selecting the sample location. Detection can begin by checking available memory and graphs of available data.

Several additional settings are needed to manage existing data after detection as shown in Figure 9. The DataResult class is used to store data obtained after detection. All data are deleted with the clear() method and overwritten with the new data graph with the addAll() method to prevent collision with the results of the previous data graph. Both methods are the default method of ArrayList objects from Android. Data in graphical form are formed into ArrayList objects given the requirement for more than one sample.

enableProcess() is an additional method for changing the appearance of the process. A setImageBitmap method is used to arrange dot-marked bitmap images to ImageView. Merging implementation in Figures 3, 4, and 5 will complete the implementation of the OnClick method for the "START" button and will immediately start detection.

3.1.2.Sample Observation

Starting detection will automatically activate Surface View, a derivative of the View class on Android that can update the screen quickly. All processes are charged to the GUI thread that handles all user interactions when the simple View is used. Processes that require excessive rendering will be handled by SurfaceView.

DetectionSurfaceView is the class that will play additional roles during detection. This class includes an onPreviewFrame method, which must be implemented (overriden) when using the Camera.PreviewCallback class. The onPreviewFrame method runs when the view surface is active. Processing time must be checked to ensure that processing does not exceed the specified duration before initiating processing. If processing exceeds the set duration, the finishProcess () method, a method in the DetectionSurfaceView class, will be added. Processing will continue when the current time is still less than the set duration.

The UI (user interface) is updated, and user interactions are handled during detection. Updating the UI in MoFlus involves updating data graphs and camera previews. Handling user interaction occurs when a user changes the location position. The Handler class is used to handle threads on Android. As shown in Figure 10, the Handler constructor uses the getMainLooper method to obtain the main thread from the application because the UI must be updated from the application's main thread.

```
private Handler handler = new Handler(Looper.getMainLooper());
. . .

public void onPreviewFrame(byte[] bytes, final Camera camera) {
        . . . .
        if (detectionHelper.isProcess()) {
                handler.post(new Runnable() {
                        @Override
                        public void run() {
                        try {
                        camera.takePicture(null, null, jpegCallback);
                        } catch (RuntimeException e) {
                                Log.d("CameraError", e.getMessage());
                        }
                    }
                });
            }
}
```

Figure 10. The use of Handler in MoFlus.

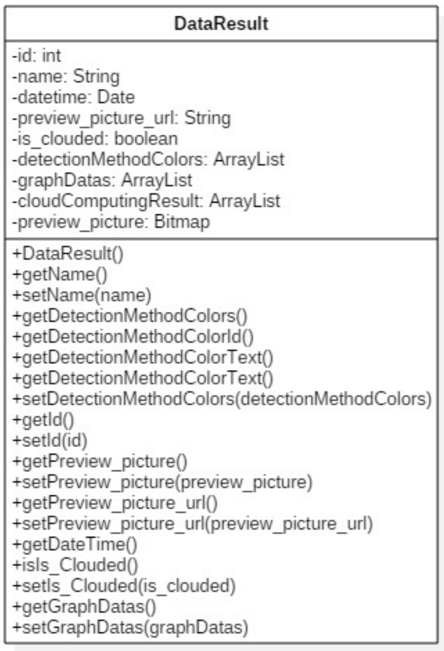
3.2.Chart Display

Each line chart depicted in Figure 4 (c) reflects the color expression value from each sample point. The x axis shows the time when the point value was taken and the y axis shows the sum of the values of the red, green, and blue values. Graph display is set in the detection method. The graph does not yet exist and only exists when detection occurs during the beginning of detection page formation. The values on the graph are obtained in real time. Value retrieval based on color detection begins when SurfaceView and surfaceHolder are initiated. The onPictureTaken method will run the imageProcessing method in the DetectionHelper class. imageProcessing will take the byte that has been converted to the bitmap and then take the value. The value is an array of RGBValue objects taken in accordance with the selected color detection method. The resulting value is mapped to the graph in the recycler view. The camera.startPreview method updates the preview of SurfaceView. The preview must begin before the image is taken in accordance with Android documentation.

3.3.Saving Results

The detection results will be stored in the DataResult class. This class structure is shown in Figure 11. The use of classes is intended to encapsulate the details of the saved observation result. The variables stored are in id, name of detection results, preview picture in the form of a bitmap and its URL, date and time of storage, the color detection method used, the results of graph data, and the results of cloud computing (cloudComputingResult). The methods in this class are constructors, getters, and setters.

Detection results can be stored for future use by using the DataResult class. The main process that occurs when saving detection results is illustrated in Figure 13. The SQLiteAsynTask is an artificial class located in the Server package. This class is derived from Android's AsyncTask. The AsyncTask class is used to perform behind-the-scenes operations and to publish results to UI threads without having to manipulate threads or handlers. AsyncTask should be used for operations that last for only a few seconds.

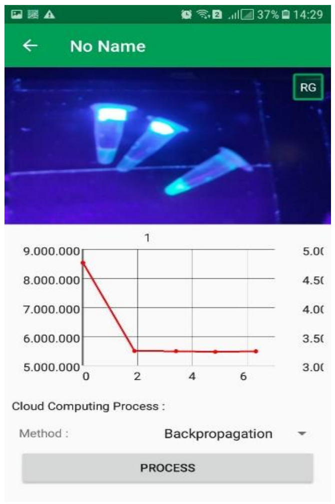


DataResult

-id: int
-name: String
-datetime: Date
-preview_picture_url: String
-is_clouded: boolean
-detectionMethodColors: ArrayList
-graphDatas: ArrayList
-cloudComputingResult: ArrayList
-preview_picture: Bitmap

+DataResult()
+getName()
+setName(name)
+getDetectionMethodColors()
+getDetectionMethodColorId()
+getDetectionMethodColorText()
+getDetectionMethodColorText()
+setDetectionMethodColors(detectionMethodColors)
+getId()
+setId(id)
+getPreview_picture()
+setPreview_picture(preview_picture)
+getPreview_picture_url()
+setPreview_picture_url(preview_picture_url)
+getDateTime()
+isIs_Clouded()
+setIs_Clouded(is_clouded)
+getGraphDatas()
+setGraphDatas(graphDatas)

Figure 11. Class structure for DataResult



Cloud Computing Process :
Method : Backpropagation
PROCESS

Figure 12.DataResult Display Results

Figure 12 displays the data results that have been obtained through detection in CSV form. createFile (dataResult) which is located in the CSVHelper class, participates in CSV creation. The CSVHelper class is located in the package helper. CSV can be created by using the OpenCSV external library, which can be accessed at http://opencsv.sourceforge.net/. The implementation in the CSV Helper class is illustrated in Figure 14.

Detection results can also be processed using several methods. Processing occurs on the server. The transmission of detection results to the server is assisted by a ClassificationRequest class located within the Server component. The Classification Request constructor converts the Data Result class into JSON format, as shown in the schema provided in Figure 15. The JSON scheme format is compiled in reference to Barbaglia et al.[19] The JSON format is selected for the use of REST web services that are highly bandwidth efficient in sending data[20].

The JSON data scheme in MoFlus (Figure 15) consists of a classification_method attribute and an array of data from observations. The classification_method attribute contains a classification method that will be run on the server side for sample classification. The array of observation data is composed of the elements of the observation in the form of the sample name and observation values. JSON schemes are compiled to validate JSON data sent by smartphones. The JSON scheme is shown in Figure 14, and the JSON document sent from the MoFlus application is shown in Figure 16.

The Instance generated from MoFlus has been validated by using the JSON Schema Validator[21]. The validation results show the JSON generated by the MoFlus application in accordance with the schema used for validation.

```
new SQLiteAsyncTask(
  context,
  moflusSQLiteHelper,
  dataResult,
  alertDialog).execute();
```

Figure 13. SQLiteAsyncTask implementation

```
public static String
createFile(DataResult dataResult)
throws IOException{
  ...
  CSVWriter writer = new
   CSVWriter(new
   FileWriter(filePath));

  writer.writeAll(text);
  writer.close();
  return filePath;
}
```

Figure 14. MoFlus createFile method

```
{
  "type": "object",
  "$schema": "http://json-
schema.org/draft-07/schema#",
  "properties": {
    "classification_method": {
      "type": "string"
    },
    "data": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "sample_name": {
            "type": "string"
          },
          "value": {
            "type": "integer"
          }
        }
      }
    }
  }
}
```

Figure 15. JSON Schema for MoFlus

```
{
  "classification_method":
"backpropagation",
  "data": [
    {
      "sample_name": "sample-A",
      "value": 250
    },
    {
      "sample_name": "sample-B",
      "value": 250
    }
  ]
}
```

Figure 16. JSON documents sent by moflus

3.5. Batch Process

The batch process does not use the SurfaceView class in image processing. The image has already been taken through the folder on the smartphone or can be taken by using a camera in the MoFluS application. Image processing begins when the user is asked to touch the screen to select the sample position to be observed (sample selection). The process mechanism in the sample selection activity is the same as that illustrated in Figures 5 and 6. The difference in the process in batches of images is taken not in real time but is already stored in the application folder. Thus, processing does not require SurfaceView.

The SurfaceView class is used when the user wants to take pictures through the camera in the MoFluS application. The class responsible for shooting is CameraController and CameraSurfaceView. The CameraController class organizes activities from the view camera, whereas the CameraSurfaceView class manages image processing through SurfaceView.

4. EVALUATION

The evaluation conducted in this work aims to measure the effect of the number of observation samples on memory consumption. Testing is conducted by using Samsung Galaxy J5 Pro, an Android smartphone with an Octa-core 1.6 GHz processor, 32GB internal memory, and API level 21. The phone is set up to work with API level 19 as its minimum level. MoFlus can be deployed on any Android device with API level 19 in this configuration.

Android Profiler is used in Android Studio 3.2 to measure memory usage by MoFlus when run on a smartphone that is connected to a PC. Memory usage is monitored through Android Studio. Tests are carried out for real time and batch processing. A printed microarray is used as an observation source in both processes. Pixel size, sample point number, intervals, and duration are used as variables in real-time testing. Pixel size and the numbers of sample points and images are used as variables in batch testing. Memory changes during observation are observed on Android Profiler.

Real-time processing testing is conducted to identify the correlation of the memory footprint of the MoFlus application with the pixel size of the observation and the number of sample points. The test is conducted by changing the pixel size of the observations (1 pixel, 5 pixels, 10 pixels, 50 pixels, 100 pixels, and 200 pixels). The number of observation sample points is varied for each point (1, 8, 16, 48, and 96 points). The results of this test are illustrated in Figure 17. The size of the memory footprint tends to be constant for the same number of sample points but slightly increases when 200 pixels are used because MoFlus requires additional color processing given that the acquired sample has a large pixel number.

The real-time processing feature is also tested by evaluating the ability of MoFlus to perform sample monitoring for a long duration on the memory used. We have tested the performance of MoFlus in real-time processing for a period of 1 min to 360 min or 6 h with a 15 s delay between acquisitions. The test results are shown in Figure 18. Figure 18 shows the memory usage of MoFlus based on usage time in real-time mode. The results of the experiment show that memory usage is relatively constant. Memory usage negligibly increases when observation is prolonged to 6 h because MoFlus observes samples from the camera that have been converted to the matrix. Thus, the shooting time does not cause changes in memory. DetectionCamera Testing component is crucial for long-term testing.

Figure 19 and Figure 20 shows the results for batch processing testing. Batch processing testing involves using previously acquired photos. The number of sample points used for batch testing is the same as that used for real-time testing.

As shown in Figure 20, testing is performed to determine the effect of the number of images on the application's footprint memory and total time. The number of images used in the test are 10, 50, 100, 500, and 1000. This test uses the same number of sample points for each variation in the number of images, i.e., 96 sample points. The result shows that the same number of sample points for all variations in the number of images tends to produce a constant memory footprint. The total time needed for processing tends to increase when the memory footprint is stable.

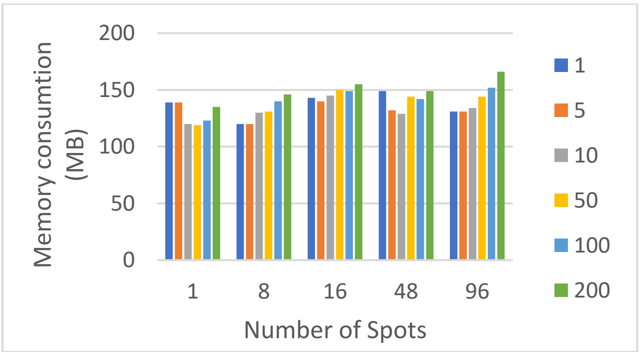


Figure 17. MoFlus real-time test results with variable testing pixel size and number of sample points.

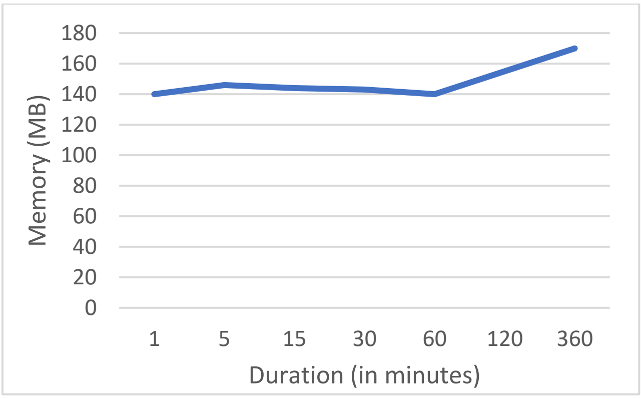


Figure 18. Result of the second real-time processing test.

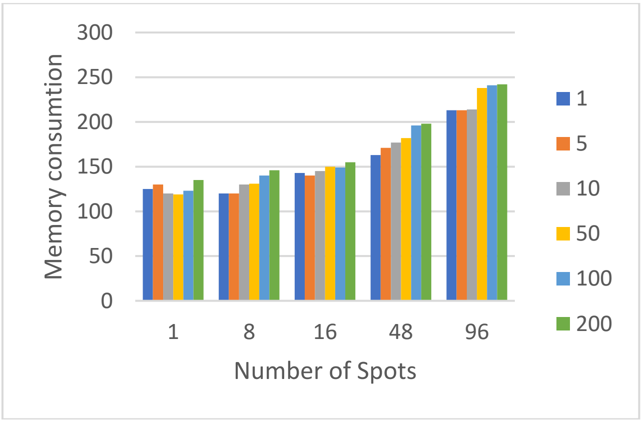


Figure 19. Results of batch testing with variable testing pixel size and number of sample.

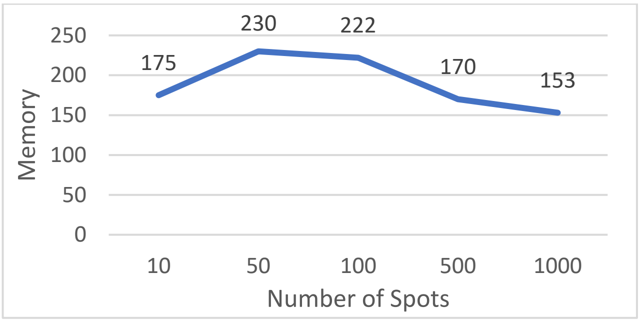


Figure 20. MoFluS batch test results with variable number of images for 96 points

5. CONCLUSION AND FUTURE WORK

This paper discusses MoFlus, an Android-based software project that can be used to observe numerous samples using a smartphone camera. Observations are made by forming observation points on the smartphone preview display. This function is established through extension to the SurfaceView Android library. MoFlus can observe up to 96 samples. The memory used by the application is still acceptable for most current smartphones. MoFlus can be used to observe a number of samples by determining the observation points in accordance with sample position. Observations can be done in real time or in batch. Observations require further processing because processing numerous samples by using a smartphone is impossible. Other services, such as mobile cloud computing, are needed to provide computation service[22]. Several components related to mobile computing and data management are interesting issues for future study. Data management is a particularly interesting topic because the use of MoFlus by many users will involve massive data and processing load. Data and processing management is necessary to prevent mixing among the results of different processes. Thus, studying the appropriate architecture for managing sample data, such as those used in MoFlus, is necessary.

REFERENCES

[1] W. F. An, "Fluorescence-based assays," in Cell-Based Assays for High-Throughput Screening, Springer, 2009, pp. 97–107.
[2] A. Hatch et al., "A rapid diffusion immunoassay in a T-sensor," Nat. Biotechnol., vol. 19, no. 5, p. 461, 2001.
[3] A. Bourouis, A. Zerdazi, M. Feham, and A. Bouchachia, "M-health: Skin disease analysis system using smartphone's camera," in Procedia Computer Science, 2013, vol. 19, pp. 1116–1120, doi: 10.1016/j.procs.2013.06.157.
[4] W. K. Tam and H. J. Lee, "Accurate shade image matching by using a smartphone camera," J. Prosthodont. Res., vol. 61, no. 2, pp. 168–176, 2017, doi: 10.1016/j.jpor.2016.07.004.
[5] B. Berg et al., "Cellphone-based hand-held microplate reader for point-of-care testing of enzyme-linked immunosorbent assays," ACS Nano, vol. 9, no. 8, pp. 7857–7866, 2015.
[6] A. Rahangdale and S. Raut, "Machine Learning Methods for Ranking," Int. J. Softw. Eng. Knowl. Eng., vol. 29, no. 06, pp. 729–761, Jun. 2019, doi: 10.1142/S021819401930001X.
[7] D. Kim et al., "Enzyme-Free Nucleic Acid Amplification Assay Using a Cellphone-Based Well Plate Fluorescence Reader," Anal. Chem., vol. 90, no. 1, pp. 690–695, 2017.
[8] A. Priye, S. W. Bird, Y. K. Light, C. S. Ball, O. A. Negrete, and R. J. Meagher, "A smartphone-based diagnostic platform for rapid detection of Zika, chikungunya, and dengue viruses," Sci. Rep., vol. 7, p. 44778, 2017.
[9] W. Chen et al., "Mobile platform for multiplexed detection and differentiation of disease-specific nucleic acid sequences, using microfluidic loop-mediated isothermal amplification and smartphone detection," Anal. Chem., vol. 89, no. 21, pp. 11219–11226, 2017.
[10] U. M. Jalal, G. J. Jin, and J. S. Shim, "Paper--Plastic Hybrid Microfluidic Device for Smartphone-Based Colorimetric Analysis of Urine," Anal. Chem., vol. 89, no. 24, pp. 13160–13166, 2017.
[11] J. Song et al., "Smartphone-Based Mobile Detection Platform for Molecular Diagnostics and Spatiotemporal Disease Mapping," Anal. Chem., vol. 90, no. 7, pp. 4823–4831, 2018.
[12] B. Lin et al., "Point-of-care testing for streptomycin based on aptamer recognizing and digital image colorimetry by smartphone," Biosens. Bioelectron., vol. 100, pp. 482–489, 2018.
[13] K. Chan, P.-Y. Wong, C. Parikh, and S. Wong, "Moving toward rapid and low-cost point-of-care molecular diagnostics with a repurposed 3D printer and RPA," Anal. Biochem., vol. 545, pp. 4–12, 2018.
[14] S. Akraa et al., "A smartphone-based point-of-care quantitative urinalysis device for chronic kidney disease patients," J. Netw. Comput. Appl., vol. 115, pp. 59–69, 2018.
[15] N. Chondros and M. Roussopoulos, "Developing IntegrityCatalog, a software system for managing integrity-related metadata in digital repositories," Softw. Pract. Exp., vol. 48, no. 1, pp. 45–64, Jan. 2018, doi: 10.1002/spe.2515.
[16] M. Ozkaya, "Do the informal & formal software modeling notations satisfy practitioners for software architecture modeling?," Inf. Softw. Technol., vol. 95, pp. 15–33, Mar. 2018, doi: 10.1016/j.infsof.2017.10.008.
[17] "Github." https://github.com.
[18] L. do Nascimento Vale and M. de Almeida Maia, "Key Classes in Object-Oriented Systems: Detection and Assessment," Int. J. Softw. Eng. Knowl. Eng., vol. 29, no. 10, pp. 1439–1463, Oct. 2019, doi: 10.1142/S0218194019500451.
[19] G. Barbaglia, S. Murzilli, and S. Cudini, "Definition of REST web services with JSON schema," Softw. Pract. Exp., vol. 47, no. 6, pp. 907–920, Jun. 2017, doi: 10.1002/spe.2466.
[20] M. A. Paredes-Valverde, G. Alor-Hernández, A. Rodríguez-González, R. Valencia-García, and E. Jiménez-Domingo, "A systematic review of tools, languages, and methodologies for mashup development," Softw. Pract. Exp., vol. 45, no. 3, pp. 365–397, Mar. 2015, doi: 10.1002/spe.2233.
[21] "JSON Schema Validator." https://www.jsonschemavalidator.net/.
[22] T. H. Noor, S. Zeadally, A. Alfazi, and Q. Z. Sheng, "Journal of Network and Computer Applications Mobile cloud computing : Challenges and future research directions," J. Netw. Comput. Appl., vol. 115, no. January, pp. 70–85, 2018, doi: 10.1016/j.jnca.2018.04.018

© 2021. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/)