UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS ASSIGNMENT No. 1979

AN APPLICATION FOR LED PATTERNS RECOGNITION WITH A SMARTPHONE CAMERA

Lovro Pejić

Zagreb, July 2019
MASTER THESIS ASSIGNMENT No. 1979

Student: Lovro Pejić (0036489431)
Study: Computing
Profile: Software Engineering and Information Systems

Title: An Application for LED Patterns Recognition with a Smartphone Camera

Description:

Your task is to design, implement and test an application for LED patterns recognition on Android smartphones. The application should consist of a client and server component. The client component should be able to recognize LED patterns using a mobile device camera. The server component should provide access to data about a recognized pattern from the database. The communication between these components should be based on the REST interface. The format of data to be exchanged in messages should be in the JSON format.

For the detection of LED patterns use appropriate computer vision methods from the OpenCV open source library.

All required literature and working conditions will be provided by the Telecommunications Department.

Issue date: 15 March 2019
Submission date: 28 June 2019

Mentor:
Associate Professor Krešimir Pripužić, PhD

Committee Chair:
Associate Professor Igor Mektarović, PhD

Committee Secretary:
Associate Professor Boris Mladenović, PhD
SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

DIPLOMSKI ZADATAK br. 1979

Pristupnik: Lovro Pejić (0036489431)
Studij: Računarstvo
Profil: Programsko inženjerstvo i informacijski sustavi

Zadatak: Aplikacija za prepoznavanje LED uzoraka korištenjem kamere na pametnim telefonima

Opis zadataka:
Vaš zadatak je osmišljen, izvesti u programskom jeziku Java te testirati aplikaciju za prepoznavanje LED uzoraka na pametnim telefonima s operacijskim sustavom Android.
Aplikacija se treba sastojeći od klijentske i poslužiteljske komponente. Klijentska komponete treba omogućiti prepoznavanje LED uzoraka korištenjem kamere mobilnog uređaja. Poslužiteljska aplikacija treba omogućiti dohvat podataka o prepoznatom uzorku iz baze podataka. Komunikaciju između klijentske i poslužiteljske komponente ostvarite korišćenjem sučelja REST. Format podataka koji će se razmenjivati u porukama neka bude u formatu JSON.
Za prepoznavanje LED uzoraka korišćenjem kamere korišćite metode računalnog vida iz programske knjižnice otvorenom koda OpenCV.
Svu potrebnu literaturu i uvjete za rad osigurat će Vam Zavod za telekomunikacije.

Zadatak uručen pristupniku: 15. ožujka 2019.

Mentor: 
Izv. prof. dr. sc. Krešimir Pripužić

Djelovodstvo: 
Izv. prof. dr. sc. Bora Milešević

Prestdjednik odbora za diplomski rad profila: 
Izv. prof. dr. sc. Igor Mesterović
Contents

1. Introduction ........................................................................................................... 1
2. Android Operating System .................................................................................. 2
   2.1. General .......................................................................................................... 2
   2.2. Android Operating System Architecture ..................................................... 2
       2.2.1. Linux Kernel ............................................................................................. 3
       2.2.2. Libraries .................................................................................................. 4
       2.2.3. Android Runtime ...................................................................................... 4
       2.2.4. Application Framework ........................................................................... 4
       2.2.5. Applications ............................................................................................. 4
2.3. Android Application Structure ......................................................................... 5
       2.3.1. Activities .................................................................................................. 10
       2.3.2. Services ................................................................................................... 12
       2.3.3. Broadcast Receiver .................................................................................. 12
       2.3.4. Content Provider ...................................................................................... 12
       2.3.5. Intents ...................................................................................................... 12
3. Other Technologies Used ..................................................................................... 13
   3.1. REST ............................................................................................................... 13
   3.2. JSON ............................................................................................................. 15
   3.3. Node.js .......................................................................................................... 16
   3.4. MongoDB ...................................................................................................... 17
4. OpenCV Library .................................................................................................. 18
5. LED Pattern Detector Application ....................................................................... 21
   5.1. Prepared Application Purpose ...................................................................... 21
   5.2. General Application Usage ......................................................................... 22
   5.3. Application Implementation Details ............................................................. 33
6. Application Limitations and Improvements ....................................................... 44
7. Installation Instructions ....................................................................................... 46
8. Conclusion ........................................................................................................... 48
9. Literature ............................................................................................................ 49
10. Summary ........................................................................................................... 51
11. Keywords ......................................................................................................... 53
12. Sažetak ............................................................................................................. 54
13. Ključne riječi ..................................................................................................... 56
Thanks to all the people who touched my heart and helped me make my dreams come true.
1. Introduction

Nowadays, when there are mobile applications for almost every purpose, it is hard to be innovative. Most of these applications are based on a client-server architecture and their main task is to manipulate data by exchanging messages between the client and the server. In such an environment, it is difficult to conceive something new and different, what would at the same time attract attention and break the existing monotony.

Developers of modern mobile applications, in order to create space for their ideas, have begun the exploitation of numerous sensors and other devices which have so far been incorporated into an enviable number of smartphones. For example, these are applications that are based on virtual or expanded reality. Given the popularity of such applications, it is very likely that in the future all smartphones will have all the sensors and other devices needed to support this modern technology. The field of computer science that includes the above examples is called "computer vision". This paper will cover the process of creating a mobile application for the Android operating system which uses the built-in camera and library of the field of computer vision called OpenCV, as well as a description of other tools required for its creation.

The following chapters will discuss the Android operating system, followed by description of REST architecture and JSON format that are commonly used when creating the server part of the application. Further, the computer vision’s library OpenCV will be presented, followed by a description of the mobile application that was built for this paper purposes and its most interesting parts.
2. Android Operating System

This chapter will discuss the emergence and development of the Android Operating System. It is separated into three subchapters. First subchapter will generally discuss Android and its beginning. Architecture of the Android Operating System will be discussed in the second subchapter. Finally, the last chapter will present the structure of the Android applications.

2.1. General

The Android Operating System is based on the Linux Operating System, but is primarily intended for touch devices, such as smartphones and other “smart” devices.

Android was originally developed by Android, Inc. Their work was initially funded by Google, who later took over the entire company. From this moment forward, Google began to participate in the global smartphone market.

The first device with the Android Operating System was distributed by Google under the Apache license and was sold in October 2008.

2.2. Android Operating System Architecture

The Android Operating System architecture consists of several layers. Each layer of lower level provides services to layers of higher levels. The following photo shows the Android Operating System architecture (Picture 1), along with the corresponding layers and program components.
Sequentially, from the lowest layer to the highest, these are: Linux Kernel, Libraries, Android Runtime, Application Framework, Applications.

2.2.1. Linux Kernel

Linux Kernel is located at the bottom of the Android architecture hierarchy and represents the basic layer on which all the higher layers in the hierarchy are based. All the drivers are located in this layer. Drivers are programs that communicate and manage device hardware. All higher layers commands pass through this layer to the physical components of the device.
2.2.2. Libraries

The next layer in the hierarchy is Android Libraries. In this layer there are libraries that allow devices to manage different types of data. They are mostly written in C and C++ programming languages, but are easily translated into other languages. Their main purpose is to provide services to other layers at higher levels.

2.2.3. Android Runtime

Android Runtime consists of the core libraries and the Dalvik virtual machine (DVM). The Dalvik virtual machine is actually Java virtual machine (JVM) which is used to launch Android applications, but is specially adapted to mobile devices in order to achieve more efficient resource management.

2.2.4. Application Framework

The next layer in the hierarchy is Application Framework. It contains the applications that manage the basic functions of the device, such as memory resources, voice calls and similar.

2.2.5. Applications

Finally, this is the layer that is located at the top of the Android Operating System architecture hierarchy. It contains the applications that are embedded in the Android Operating System (camera, calculator, phone book) as well as the other applications that the user has decided to install.
2.3. Android Application Structure

From a very basic point of view, it is easy to notice that every single Android application consists of two key parts. The first part includes so-called “Java documents”, which represent the files that contain application logic. These Java documents contain program code that is written in the Java programming language. The second part refers to the design of the graphical user interface (GUI) which is designed in XML (Extensible markup language) language. Popular development environments such as Android Studio offer the ability to design a graphical user interface through special tools.

It is important to emphasize that there are usually some details that are extremely difficult to design precisely through the mentioned tools, so it is still useful to know XML. There are also other interesting files, such as AndroidManifest.xml, that do not belong to any of the above parts. Those will be further explained later.

Android SDK (Software Development Kit) tools compile the code, along with all other application files, into a unique archive with .apk extension. The resulting archive is a file, used by Android devices when installing an application.

The following images illustrate an example of Java code (Picture 2) and an example of XML code (Picture 3) within an Android application.
public int onStartCommand(Intent intent, int flags, int startId) {
    Log.i(TAG, "servicetest", msg, "service started");
    Notification notification = new NotificationCompat.Builder("
            wired connection...")
            .setTicker("wired connection...")
            .setContentTitle("wired connection")
            .setSmallIcon(R.drawable.ic_launcher_background)
            .setOngoing(true)
            .build();
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        String channelId = "default";
        CharSequence channelName = "Some Channel";
        int importance = NotificationManager.IMPORTANCE_HIGH;
        NotificationChannel channel = new NotificationChannel(channelId, channelName, importance);
        channel.setDescription("Wired connection...");
        NotificationManager notificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
        notificationManager.createNotificationChannel(channel);
    }
    startForeground(101, notification);
    enableAsixDriver();
    this.intent = intent;
    return START_STICKY;
}

@Override
public void onDestroy() {
    Log.i(TAG, "servicetest", msg, "service stopped");
    disableAsixDriver();
    super.onDestroy();
}

private void disableAsixDriver() {
    try {
        // Code to disable Asix driver
    }
}
Android application directory has a tree-shaped structure. It contains the components that the developer modifies when creating the application (i.e. when creating a desired graphical user interface).

The following image (Picture 4) shows the Android application directory structure.
Some of the more important directories are:

- **java/** - contains packages with program code

- **res/** - contains parts that are related to the graphical user interface (i.e. images, animation, colors, music, …)

- **Gradle Scripts/** - This directory contains very important build.gradle files. They specify different parameters that apply to the level of individual module or to the level of the entire application. Also, keep a list of external libraries (i.e. Google play services api, Retrofit api, …) which is used within the application. When building an application, the necessary files from these libraries are downloaded from the Internet and embedded in the application.
• Manifests/ - contains the AndroidManifest.xml file, owned by every Android application and is very important for its work. The mentioned file usually includes:

  ▪ List of activities used inside the application
  ▪ Some permissions (i.e. to access the Internet, to use the camera )
  ▪ Some limitations related to individual activity or the entire application
    (i.e. screen rotation lock)

Each Android application is made up of several components that can communicate with other components within the same or other applications as needed.

There are four different groups of components. These are:

• Activities
• Services
• Broadcast receiver
• Content provider

Each of these components has a certain role and life cycle. These are actually the basic building blocks of the Android applications. Thorough knowledge of the working of these components is of great importance given that the developer can perform all the operations that one needs to achieve the desired functionality. In addition, it is known that direct communication between two applications on a device is not possible, although it can still be achieved by using the services provided by the above mentioned components.
2.3.1. Activities

Activity represents one screen along with the corresponding graphical user interface. Each application can consist of one or more activities that work together. They are mutually independent, yet they make a meaningful whole. Each application should have at least one activity so that the user can perform any interaction with the application. Multiple activities can be designed in the application, but simultaneously only one activity can be displayed on the screen. The activity that is triggered when launching an application is most often called the main activity. Each activity can be found in one of the six possible states. These states are:

- Created
- Started
- Resumed
- Paused
- Stopped
- Destroyed

When entering any of the above states, some embedded methods that are closely related to the current state of a particular activity are initiated. These methods can be overridden and modified for the purpose of extra task execution. The next image (Picture 5) shows the life cycle of the activity along with the mentioned embedded methods.
List of embedded methods and their functionality:

- `onCreate()` - called when the activity is first created
- `onStart()` - called when the activity becomes visible
- `onPause()` - called when the current activity is being paused and the previous activity is being resumed
- `onResume()` - called when the user starts interacting with the application
- `onRestart()` - called when the activity restarts after stopping it
- `onStop()` - called when the activity is no longer visible
- `onDestroy()` - called before the activity is destroyed
2.3.2. Services

Service is a component running in the background and has no relevant graphical user interface. It can perform common procedures such as activity, but also very long-lasting and demanding tasks. As it does not work with graphic components, it is never performed on the graphics thread. It is easier to work with services than with activities, but it requires greater caution.

2.3.3. Broadcast Receiver

Broadcast receiver is the component responsible for receiving disclosed intents. When it receives a specific intent, it may initiate notification displays, display a message about a weak battery and similar. Broadcast receiver does not possess its own graphical user interface and never performs a significant amount of work.

2.3.4. Content Provider

Content provider is a component that manages access to different data structures that are permanently stored in the device. It is used to transfer data between applications, to store data in files, to transfer data over the Internet (with OS permission) and such tasks.

2.3.5. Intents

Intent is essentially an event that is intended to activate the appropriate components within the application. The intent is triggered by instantiating the class object Intent that specifies the action to be executed. When initiating the intent, the action that needs to be executed is specified along with the caller of the intent (optionally). If the caller is assigned, the intent can be executed immediately. Otherwise it is determined internally within the application. The best example of this is the code snippet that the developer can launch a new activity with.

```
Intent intent = new Intent(StartUpActivity.this, MainActivity.class);
startActivity(intent);
```
In the enclosed code, the first parameter represents the caller of the intent, while the second represents the action to be executed. In this case that is launching a new activity. In the previous example the activity is initiated by `startActivity()` method, but if it is expected to return a result, then it should be initiated by `startActivityForResult()` method.

### 3. Other Technologies Used

Backend part of the application usually consists of several technologies. One possible set of them is used for this paper purpose, so they will be described in the following chapter.

#### 3.1. REST

REST is an architectural style for providing standards between computer systems on the web. In the REST architectural style, the implementation of the client and the server can be done independently without each knowing about the other. This means that the code on one side (i.e. client) can be changed without affecting the other side (i.e. server). They can be kept modular and separated as long as each side knows what format of messages to send to the other side. Through the REST interface different clients can perform the same actions to the same REST endpoints and receive the same responses.

A very important feature of the systems that follow the REST paradigm is that they are stateless, meaning that the server does not need to know anything about client state and *vice versa*. This means that both the client and the server can understand any message received, even without seeing previous messages.

When using the REST architecture, clients send requests in order to modify or retrieve resources and servers send responses to those requests. A request generally consists of several parts, such as:

- An HTTP verb (defines the kind of the operation)
- A header (allows client to pass extra information about the request)
• A path to source
• A message body containing data (optional)

There are four basic HTTP verbs that are used in the requests:

• GET – used to retrieve a specific resource or collection
• PUT – used to update a specific resource
• POST – used to create a new resource
• DELETE – used to remove a specific resource

These verbs provide the functions to create, read, update and delete resources. Their popular name is CRUD. These are the fundamental elements of a persistent storage system. The CRUD paradigm is common in constructing web applications, because it provides a memorable framework for reminding developers of how to construct full, usable models. With the appropriate header, the client can suggest to server the type of response expected (i.e. application/json). Requests must contain a path to a resource that the operation should be performed on. The idea of that path is to help the client know what is going on and that is the good practice to design them in that way.

Responses from the server contain status codes to alert the client to information about the success of the operation. The most popular status codes are:

• 200 – OK
• 201 – CREATED
• 400 – BAD REQUEST
• 403 – FORBIDDEN
• 404 – NOT FOUND
• 500 – INTERNAL SERVER ERROR
The next two pictures represent the example of the server request and the appropriate server response.

```json
POST http://fashionboutique.com/customers
Body:
{
  "customer": {
    "name" = "Scylla Buss"
    "email" = "scylla.buss@codecademy.org"
  }
}
```

Picture 6. The client request example [6]

```plaintext
201 (CREATED)
Content-type: application/json
```

Picture 7. The server response example [6]

### 3.2. JSON

JSON stands for JavaScript Object Notation. JSON is a lightweight format for storing and transporting data. JSON is often used when data is sent from the server to the client and *vice versa*. JSON is simple and easily understandable.

There are several rules that must be followed to create a valid JSON object:

- Data is in *name/value* pairs
- Data is separated by *commas*
- Curly braces hold objects
- Square brackets hold arrays
The following picture (Picture 8) shows the structure of the JSON object. It presents an array of three employee records.

```json
{
    "employees":[
        {"firstName":"John", "lastName":"Doe"},
        {"firstName":"Anna", "lastName":"Smith"},
        {"firstName":"Peter", "lastName":"Jones"}
    ]
}
```

Picture 8. JSON example [8]

### 3.3. Node.js

Node.js is an open-source and cross-platform JavaScript runtime environment built on Chrome’s V8 JavaScript engine. Nowadays it has become extremely popular and can be used for almost any kind of project. Node.js uses an event-driven, non-blocking I/O model making it lightweight.

Npm (Node Package Manager) is a package ecosystem for Node.js. Besides, it is the largest ecosystem of open source libraries in the world.

A Node.js application is run in a single process, without creating a new thread for every request. When Node.js needs to perform an I/O operation (i.e. write something to database), instead of blocking the thread and wasting CPU cycles waiting, Node.js will resume the operations when the response comes back. This feature gives the ability to Node.js application to handle thousands of concurrent connections with a single server without introducing a load of managing thread concurrency, which can be a significant source of bugs.

Node.js example application is shown in the following picture (Picture 9).
3.4. MongoDB

"MongoDB is a document database with the scalability and flexibility that you want with the querying and indexing that you need." [13]

MongoDB is a cross-platform document-oriented database program developed by MongoDB, Inc. under SSPL licence. It is classified as a NoSQL database.

MongoDB stores data in flexible, JSON-like documents. This means that stored data structure can be changed over time. The MongoDB document model maps directly to the objects in your application code, so it is always possible to work with predefined object classes. Due to the fact that MongoDB is a distributed database at its core, it has high availability and horizontal scaling possibility.

MongoDB’s document model is simple for developers to use. On the other hand, it is still providing all the capabilities needed to meet the most complex requirements.

Simple example of how to use MongoDB from Node.js application is shown below (picture 10).

```javascript
const http = require('http')
const hostname = '127.0.0.1'
const port = process.env.PORT

const server = http.createServer((req, res) => {
  res.statusCode = 200
  res.setHeader('Content-Type', 'text/plain')
  res.end('Hello World!\n')
})

server.listen(port, hostname, () => {
  console.log('Server running at http://' + hostname + ':' + port + '/')
})
```
4. OpenCV Library

OpenCV (Open Source Computer Vision Library) is an open source and cross-platform computer vision software library. It is originally developed by Intel, but developed by several authors over time. It is released under the BSD 3-Clause License and it is free for commercial use. OpenCV was built to accelerate the use of machine perception in the commercial products and to provide a common infrastructure for computer vision applications. The OpenCV library consists of more than 2500 optimized algorithms, which includes a comprehensive set of computer vision and machine learning algorithms. These algorithms have wide application (Picture 11).

```javascript
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  dbo.collection("customers").find({}).toArray(function(err, result) {
    if (err) throw err;
    console.log(result);
    db.close();
  });
});
```

Picture 10. Example of MongoDB usage from Node.js application [14]
For example, they can be used in following situations:

- To detect and recognize faces
- To identify objects
- To track camera movements
- To track moving objects
- To find similar images from an image database
- To follow eye movements
The OpenCV library is used extensively in companies, research groups and by governmental bodies. The quality of OpenCV was recognized by numerous companies that come from different area. For example, it is known that some software developing companies (i.e. Google, Microsoft), or some companies from car industry (i.e. Honda, Toyota) also use OpenCV to improve their products. Along with mentioned well-established companies, there are many startups that make extensive use of OpenCV. OpenCV is written in C++ and its primary interface is in C++, but it also supports Windows, Linux, Android and Mac OS.

The OpenCV4Android Library is a version of the OpenCV library that is customized and optimized for work on mobile devices with Android Operating System (Picture 12). To start using OpenCV in Android application it is not just enough to add required dependency to build.gradle file, because OpenCV uses native languages, which are not native to Android. To solve this incompatibility issue it is required to install Android native development kit (Android NDK).

![OpenCV library and Android OS](image)

Picture 12. OpenCV library and Android OS [15]
5. LED Pattern Detector Application

This chapter consists of three subchapters. First subchapter will present the prepared application and determine its purpose, while the second subchapter will focus on general application usage. Finally, the last chapter will discuss about implementation details of the most interesting parts of the application.

5.1. Prepared Application Purpose

LED Pattern Detector is a mobile application designed for the Android Operating System that is used to determine the device error according to the frequency of blinking of its certain LED light source. The example environment is shown on next image (Picture 13). The entire application has several built-in features that will be described later in this chapter along with the description of the application usage.

![Picture 13. LED Pattern Detector example environment](image-url)
5.2. General Application Usage

When the user starts the application for the first time, a popup dialog that requires the camera usage permission (picture 14) is displayed. After that, as well as every subsequent launch of the application, a short-duration loading screen appears (picture 15), which is necessary to overcome the synchronization process with the remote database, as well as to improve the aesthetics of the main application screen.

Picture 14. Requiring camera permission
Sequence diagram which presents mentioned synchronization process is shown below (Diagram 1).
Diagram 1. Database synchronization process sequence diagram
Immediately after everything is ready, the main application screen appears (picture 16).

![The main application screen](image)

The main application screen consists of several components. There is a small red-colored text in the upper left corner that is dynamically changed depending on the number of available frames per second (FPS).

Right below it is the status bar with a message that tells the user which actions are expected from the user to be taken at a certain point in time. Therefore, there is an initial message that relates with the button that is located at the lower right corner of the screen which initiates the process of finding the LED light source. Before starting the process of detecting the desired light source it is highly recommended to please the initial parameters. This refers to the button that is located at the lower left corner of the screen and its task is to adjust the color range of the light source according to the outside illumination and visual disturbances. The current applied color range can be checked by turning on the slider in the upper right corner of the screen that converts the image to binary content that is used by the application in all future steps.
The following two images demonstrate an example of the color range calibration in the incorrect (picture 17) and then in the correct (picture 18) manner for the light source that is visible on previous image.

Picture 17. Incorrect color range calibration

Picture 18. Correct color range calibration
In the first image the light source is invisible, but it is clear and sharp in the second image. Also, it is easy to notice that there are some other objects which fall into the defined color range, but they will be considered trash, not important in the further processing. There are two available approaches to modify the color range - automatic and manual (picture 19).

![Picture 19. Color calibration options](image)

If the user chooses automatic color calibration option the user will also need to take a frame on which the light source is switched on in the next step (picture 20). It is essential that the light source is switched on because it is very likely that it will be the brightest part of the image and therefore the automatic calibration process will succeed. If it fails, it is possible that the light source was switched off and then it is expected from the user to repeat the automatic color calibration process. Moreover, it is also possible that there is a case where another object is always brighter than the observed light source, rendering the automatic calibration process always a failure because it relies on finding the brightest pixel on the frame.
For this reason, to avoid the impossibility of color range calibration, there is a manual color calibration process which is very similar to automatic one. When the user chooses manual color calibration option one is also expected to take a frame in which the light source is switched on (picture 21).

Picture 20. Auto calibration process – Taking frame

When the frame with light source switched on becomes successfully taken, the user gets the option to tap the middle of the light source (picture 22). After the user has touched the screen, the purple circle appears at the point where the screen has been tapped. Although it is sometimes unlikely to target the exact middle of the source, the user is given a keyboard that helps to get more precise results.

Picture 22. Manual calibration process – Light source targeting

Once the initial parameters have been successfully set, the user is free to proceed to the next step and start the LED light source detecting process (picture 23) by tapping “Start analysis” button on the main application screen.
At this step the user is guided to tap the light source which is expected to be positioned inside the region of interest (ROI). After the light source is detected, the measuring process, lasting 5 seconds, immediately starts (picture 24). The user is recommended to remain as calmer as possible so the focus on the observed source would not be lost. If so, it is necessary to repeat the measurement process to get the precise results. The effect of natural hand movement should be minimized, but it is taken into account and should not affect the measurement results.
Due to some technical requirements, the user is expected to keep the light source inside the green ROI during the measuring process. Therefore, if the light source crosses the border of green ROI, it becomes red (picture 25) just to let the user know that the user is doing something wrong and to suggest the user to put the light source inside the ROI again. If the user refuses, the measuring process may be canceled.

![Picture 25. LED outside the ROI during measuring](image)

When the measuring process finishes, the screen with the measuring results becomes visible (picture 26). The corresponding error code becomes selected from the error codes list, while the description of the error appears on the lower part of the screen. The user can also select any other error from the list to review its description.

Finally, to return to the main screen it is enough to press the back key.
To conclude, all the actions that the user can take are shown on the next use-case diagram (Diagram 2).
5.3. Application Implementation Details

The whole application consists of three activities: StartupActivity, MainActivity and ResultsActivity.

StartupActivity is an activity that handles remote database synchronization process. As mentioned earlier, each time when a user starts the application there is a loading screen necessary to overcome the synchronization process with the remote database. The assumption is that measurements may be performed in an environment where an internet connection is unavailable. Therefore, it is necessary to have a local copy of the database on the mobile device. When the application is launched there are four possible scenarios related to the synchronization process. They are described by following finite state machine diagram (Diagram 3).
So, if internet connection is available, the application will check if the remote database version and local database version are the same. If so, the application synchronization process is finished and main application screen becomes visible. Otherwise, local database will be upgraded to the state of the remote database. On the other hand, if there is no internet connection available and local database exists, it will be assumed that this version of database is valid and will be used in further application usage. Else, if local database version does not exist, further application usage is not allowed and application will terminate. This is the correct behavior because if the database does not exist it is not possible to display information about the sample that is measured.

The method that checks if local database version matches online database is handled by following code.
Server side code which handles that check request is shown below.

The method that fetches remote database data and saves it to local database is presented by following code snippet.

```java
private static void checkLocalDbMatchesOnlineDb(final Database db, final DatabaseSyncFinishedListener databaseSyncFinishedListener) {
    Call<OnlineDBVersionModel> onlineDBVersionModelCall = ApiManager.getInstance().getService().getVersionData();
    onlineDBVersionModelCall.enqueue(new Callback<OnlineDBVersionModel>() {
        @Override
        public void onResponse(Call<OnlineDBVersionModel> call, Response<OnlineDBVersionModel> response) {
            if (response.isSuccessful()) {
                OnlineDBVersionModel onlineDBVersionModel = response.body();
                final String onlineDBVersion = onlineDBVersionModel.getOnlineDBVersion();
                if (!onlineDBVersion.equals(db.getDBVersion())) {
                    Log.d("debug", "versions changed.");
                    fetchOnlineDatabaseAndSave(db, databaseSyncFinishedListener);
                } else {
                    Log.d("debug", "versions are the same.");
                    databaseSyncFinishedListener.databaseSyncFinished(true);
                }
            }
        }
        @Override
        public void onFailure(Call<OnlineDBVersionModel> call, Throwable t) {
            databaseSyncFinishedListener.databaseSyncFinished(false);
        }
    });
}
```

var url = 'mongodb://localhost:27017/ErrorsDatabase';
router.get('/', function (req, res, next) {
    mongo.connect(url, function (err, db) {
        assert.equal(null, err);
        db.collection("version_table").find().toArray(function (err, result) {
            if (err) throw err;
            var dbVersion = result[0].version;
            if (result.length !== 0) {
                console.log(dbVersion);
                res.send({"version": dbVersion});
            }
            db.close();
        });
    });
    module.exports = router;

```java
private static void fetchOnlineDatabaseAndSave(final Database db, final DatabaseSyncFinishedListener databaseSyncFinishedListener) {
    db.cleanTables(); // cleans existing local database tables
    Call<OnlineDBVersionModel> onlineDBVersionModelCall = ApiManager.getInstance().getService().getVersionData();
    onlineDBVersionModelCall.enqueue(new Callback<OnlineDBVersionModel>() {
        @Override
        public void onResponse(Call<OnlineDBVersionModel> call, Response<OnlineDBVersionModel> response) {
            if (response.isSuccessful()) {
                OnlineDBVersionModel onlineDBVersionModel = response.body();
                final String onlineDBVersion = onlineDBVersionModel.getOnlineDBVersion();
                Call<ErrorDataListModel> errorDataListModelCall = ApiManager.getInstance().getService().getErrorsDataList();
                errorDataListModelCall.enqueue(new Callback<ErrorDataListModel>() {
                    @Override
                    public void onResponse(Call<ErrorDataListModel> call, Response<ErrorDataListModel> response) {
                        if (response.isSuccessful()) {
                            ErrorDataListModel errorDataListModel = response.body();
                            List<ErrorDataModel> errorDataList = errorDataListModel.getErrorDataList();
                            for (ErrorDataModel errorDataModel : errorDataList) {
                                db.insertErrorData(errorDataModel.getFrequency(), errorDataModel.getMessage());
                            }
                            db.insertDBVersion(onlineDBVersion);
                            databaseSyncFinishedListener.databaseSyncFinished(true);
                        }
                    }
                });

                @Override
                public void onFailure(Call<ErrorDataListModel> call, Throwable t) {
                    databaseSyncFinishedListener.databaseSyncFinished(false);
                }
            }
        }
    });

    @Override
    public void onFailure(Call<OnlineDBVersionModel> call, Throwable t) {
        Log.d("debug", "error occurred 2...");
        databaseSyncFinishedListener.databaseSyncFinished(false);
    }
}
```

The corresponding server side is shown below.

```javascript
var url = 'mongodb://localhost:27017/ErrorsDatabase';

router.get('/', function (req, res, next) {
    console.log("route errors triggered");
    mongo.connect(url, function (err, db) {
        assert.equal(null, err);
        db.collection("errors_table").find().toArray(function (err, result) {
            if (err) throw err;
            var errorsArray = [];
            for (var i = 0; i < result.length; i++) {
                var errordata = {
                    frequency: result[i].frequency,
                    message: result[i].message,
                };
                errorsArray[i] = errordata;
            }
            if (result.length != 0) {
                res.send({'errorDataList': errorsArray});
            }
            db.close();
        });
    });
});
```
Before continuing to the next point, it is very important to emphasize that image processing, as well as other computer vision methods, is extremely demanding for processing, especially in real-time. Therefore, it is advisable to write the programming code in a way that enables the system to maintain acceptable performance.

MainActivity is an activity that implements OpenCV’s CvCameraViewListener2 interface and whose main task is to process each camera frame. The most important method’s declaration from mentioned interface is presented below.

```java
public Mat onCameraFrame(CvCameraViewFrame inputFrame);
```

Due to the fact that the method above is used in various ways depending on the current application state it would not be good practice to encumber the MainActivity’s onCameraFrame method. So, it is a better idea to separate it to different modes which is applied depending on the current application state. That way, onCameraFrame method, whose implementation is presented by next code snippet, should stay as simple as possible.

```java
@Override
public Mat onCameraFrame(CameraBridgeViewBase.CvCameraViewFrame inputFrame) {
    openCVConfig.setRgba(inputFrame.rgba());
    activeModeHolder.getActiveMode().processFrame(this);
    return openCVConfig.getRgba();
}
```

Therefore, each further frame is attached to appropriate mode in which it is processed. There are six possible modes. These modes are:

- InitialMode
- LedDetectingMode
- ManualCalibrationMode
- MeasuringMode
• PreAutoCalibrationMode

• PreManualCalibrationMode

There are two modes that are interesting to analyze more closely. These are LedDetectingMode and MeasuringMode.

When application goes into LedDetectingMode or MeasuringMode each frame should be processed with regard on the central region of interest. This means that user is expected to position the LED source inside mentioned area and keep it inside during detecting and measuring. Honestly, this is an optimization trick that is used to reduce the area of the frame that is currently being processed and thus maintain stable performance. Nevertheless, without using this trick a number of available frames per second may drastically fall and it should be avoided. Increasing the number of available frames increases the upper limit of the source blinking frequency that the application can measure.

LedDetectingMode’s processFrame method implementation is located below.

```java
@Override
public void processFrame(final Context context) {
    MatManager.applyColorSpaceMask(openCVConfig.getTopLeftBorderPoint(),
    openCVConfig.getBottomRightBorderPoint());  // 1
    openCVConfig.getRgba().copyTo(openCVConfig.getTempGrayMat());
    if (!openCVConfig.isFixedPointsThreadStarted()) {  // 2
        openCVConfig.setFixedPointsThreadStarted(true);
        setFixedPointsThread();
    }
    if (openCVConfig.isDetectLedRequestSet()) {  // 3
        processDetectLedRequest();
    }
    guiManager.drawBorder(openCVConfig.getTopLeftBorderPoint(),
    openCVConfig.getBottomRightBorderPoint(), new Scalar(10, 30, 120));
}
```
For easier understanding, the role of this function can be observed separately in three smaller steps. These steps are:

- Change color space range [1]
- Manage thread that searches for fixed points [2]
- Handle LED detect request [3]

First of all, it is necessary to change color space range in order for it to be valid for further processing. That needs to be performed because OpenCV’s methods which are used inside this task can only work with customized color range. Next, when the user selects the LED source that the user wants to track, its location needs to be tracked between the frames while the measuring is in progress. This source could not be directly tracked because it may be visible on one frame, but also it may disappear in the next. The reason why the fixed points are used is because they are always visible. The OpenCV’s method by which suitable fixed points are searched for may be a limiting factor and may slow the frame processing. That is the reason why it is handled by a separate thread. The final step of this task is to handle the LED detect request that user put anywhere inside of the ROI. This step is rather simple because it is only necessary to check if there is a contour under the tapped location that is fully inside of the ROI. If so, the application will detect it and the measuring may be started. Code snippets that present searching for fixed points and LED source detecting process are shown below, sequentially.
private void processDetectLedRequest() {
    List<MatOfPoint> list = new ArrayList<>();
    Imgproc.findContours(openCVConfig.getImgResult(), list, new Mat(), Imgproc.RETR_TREE, Imgproc.CHAIN_APPROX_SIMPLE);
    list = filterListByBorder(list); // remove contours that are not inside of the ROI (fully)
    Point location = openCVConfig.getTransitionROIPoint();
    Point scaledLocation = new Point(location.x - openCVConfig.getTopLeftBorderPoint().x, location.y - openCVConfig.getTopLeftBorderPoint().y);
    MatOfPoint selectedContour = guiManager.getContourOnLocation(scaledLocation, list);
    if (selectedContour != null) {
        Log.i("debug", "Led contour detected on selected location");
        Rect r = Imgproc.boundingRect(selectedContour);
        Point tlLedRoiBorderPoint = guiManager.translatePointToFitScreen(r.tl(), openCVConfig.getTopLeftBorderPoint());
        Point brLedRoiBorderPoint = guiManager.translatePointToFitScreen(r.br(), openCVConfig.getTopLeftBorderPoint());
        if (isFixedPointValid(tlLedRoiBorderPoint, brLedRoiBorderPoint)) {
            prepareAndStartMeasuring(fixedPoint.clone(), selectedContour, tlLedRoiBorderPoint, brLedRoiBorderPoint);
        } else {
            skipFrame();
        }
    } else {
        skipFrame();
    }
    Imgproc.circle(openCVConfig.getRgba(), location, 10, new Scalar(200, 60, 80), 10);
}
MeasuringMode handles the measuring process. It uses some values set in LedDetectingMode such as fixed point initial position. So, it is easy to notice that they are closely related. Moreover, MeasuringMode is very similar to LedDetectingMode, but as its name suggests, it has a different purpose. Implementation of MeasuringMode’s method that processes each frame is shown below.

```java
@Override
public void processFrame(final Context context) {
    prepareErodePoints();

    try {
        calcOpticalFlow(); [1]
        MatManager.applyColorSpaceMask(tlErodePoint, brErodePoint); [2]

        List<MatOfPoint> list = new ArrayList<>();
        Imgproc.findContours(openCVConfig.getImgResult(), list,
                              new Mat(), Imgproc.RETR_TREE, Imgproc.CHAIN_APPROX_SIMPLE);

        Point scaledLedLightCenterPoint = new Point(
                openCVConfig.getLedLightCenterPoint().x - tlErodePoint.x,
                openCVConfig.getLedLightCenterPoint().y - tlErodePoint.y);

        MatOfPoint contour = guiManager.getContourOnLocation(scaledLedLightCenterPoint, list);

        double currentFrameContourArea = 0.0;
        if (contour != null) { [3]
            updateLedLocation(contour);
            currentFrameContourArea = Imgproc.contourArea(contour);
        }

        updateLedStatus(currentFrameContourArea, context); [4]

        guiManager.drawBorder(openCVConfig.getTopLeftBorderPoint(),
                              openCVConfig.getBottomRightBorderPoint(),
                              openCVConfig.getActiveRoiColor());

        openCVConfig.setPreviousFrameContourArea(currentFrameContourArea);
    }
    catch (Exception ex) {
        showMeasuringFailedMessage(context);
        guiManager.resetState();
    }
}
```

This function may also be separated into few smaller steps. These steps are:

- Predict LED source location [1]
- Change color space range [2]
- If LED source exists in the current frame, update its location (to undo zoom change effect) and calculate its area [3]
• Update LED source status (i.e. LED source is turned on) [4]

The first step is to predict LED source location with regard on fixed point’s position on the frame that is currently processed. In the absence of errors, a fixed point is the one that can be tracked between frames. Distance between LED source and the fixed point is always the same. Therefore, if the fixed point position is known, the LED source position is also known, regardless of the LED source current state. OpenCV’s method declaration that is used to determine fixed point position between frames is presented below.

```java
public static void calcOpticalFlowPyrLK(Mat prevImg, Mat nextImg, MatOfPoint2f prevPts, MatOfPoint2f nextPts, MatOfByte status, MatOfFloat err)
```

Simplified, the method receives the previous frame and fixed point’s previous location based on which it computes new location of the fixed point on the current frame.

The second step is exactly the same as the first step that was explained along with LedDetectingMode’s steps.

The third step is specific to MeasuringMode and it is very important to explain it well. If the contour at the given location is not found, the assumption is that the LED source is completely extinguished and further processing will take this into account. Otherwise, if the contour is found, there are two actions to be performed before continuing. Therefore, it is necessary to update the information about the found contour area which will be used in the last step and it is necessary to update the distance between fixed point and LED source. As mentioned before, that distance is actually always the same, but the camera may consider it is changed if the zoom is changed. In order to neutralize zoom changed effect, the source location is updated once again each time when LED source is found. The visual explanation is presented by following image (Picture 27).
Finally, the last step is to update the LED source status, which is to determine if LED source is turned on or off and thus count total blinks number. The part of source code that handles that is shown below.

```java
private void updateLedStatus(double currentFrameContourArea, Context context) {
    if (isLedTurnedOn(currentFrameContourArea)) {
        lightOnSignal();
    } else {
        lightOffSignal(context);
    }
}

private void lightOnSignal() {
    if (!openCVConfig.getLightStatus()) {
        openCVConfig.setTotalBlinks(openCVConfig.getTotalBlinks() + 1);
        openCVConfig.setLightStatus(true);
    }
}

private void lightOffSignal(Context context) {
    if (openCVConfig.getLightStatus()) {
        openCVConfig.setLightStatus(false);
        checkTimerSet(context); // if not, set it
    }
}
While the measuring, when the LED source becomes turned off for the first time, measuring countdown timer may be started. When it counts down to zero, the measuring process is finished and the final activity that presents measuring results may be presented to the user. That activity name is ResultsActivity.

ResultsActivity’s task is quite simple. It calculates LED source blinking frequency based on the total blinks number and the measuring duration, after which it displays the corresponding description of the calculated frequency which was obtained from the local database.

### 6. Application Limitations and Improvements

Before continuing on, it is suitable to show and explain the graph depicting a sample of LED source behavior (Picture 28).

![Picture 28. The graph depicting a sample of LED source behavior](image)
The graph above represents a possible sample of the LED source frequency that may appear during the measuring. When the state equals one, the LED source is turned on. On the other hand, if it equals zero, the LED source is turned off. The slopes in the graph represent a moment in which the state of the LED source changes its state. Red circles refer to moments that the camera managed to catch. So, it is obvious that the camera could not obtain every single moment, but only some, depending on the number of available frames per second. If the LED source blinking frequency is too high or the number of available frames per second is too low, it is possible that the camera could miss the frames that carry information about LED source state change and thus skip to count some blinks.

Current implementation takes the following into account:

- If the LED source state is equals zero, the LED source is turned off
- Everything else implies that the LED source is turned on, including the moments in which LED source state is changing its state

To conclude, each blinking frequency of the LED source is valid as long as the current implementation is able to catch the moments in which the LED source is completely turned off.

Prepared application was tested on Samsung Galaxy S6 edge+ mobile device with Android Operating System. Its number of frames per second was stable and it was around 30 FPS. The maximum frequency that the application managed to work with was 6.6 Hz. With too high frequencies, the application skipped some blinks.

As in many other cases, there is also a possibility for some improvements here. For example, it is possible to increase the maximum supported frequency by modifying the way in which the LED source states are interpreted. Moreover, changing the LED source state could be detected by tracking the area of the LED source. Therefore, if it increases twice in a row, it means that the LED source is turned on and vice versa. Using this solution, it is necessary not to forget to neutralize the zoom effect by scaling the area of the LED source.

Finally, it is inevitable to admit that previous statement is just one of the possible improvements, but still not the solution for the frequencies that are too high.
Installation of the application is pretty simple and it takes just a few steps. It could be performed in two different ways.

The first way is automatically through the development environment. This method is more convenient (usually used by developers) and will be described first below.

For the example of installation approach that is mentioned above, the development environment Android Studio will be used. The installation steps are as follows:

- The first step is to open the device settings and find the “Unknown sources” option in the security section and enable it. This option is used to allow installation of the applications that do not come directly from the Google Play Store.

- Connect your Android device to your computer using a USB cable and allow your computer to access your mobile device data. The dialog that requests this permission appears automatically and only needs to be accepted.

- Click on the green button “play” inside the Android Studio, select the connected device from the devices list and confirm the selection.

- When installation becomes finished, it is possible to disconnect the device.

The second way implies exporting the application to the .apk archive using the capabilities of the development environment, after which it needs to be transferred to the internal memory of the phone and installed manually.

The alternate way of installation steps are quite similar. Moreover, first two steps are completely the same.
All the steps are:

- The first step is to open the device settings and find the “Unknown sources” option in the security section and enable it. This option is used to allow installation of the applications that do not come directly from the Google Play Store.

- Connect your Android device to your computer using a USB cable and allow your computer to access your mobile device data. The dialog that requests this permission appears automatically and only needs to be accepted.

- Transfer the exported .apk archive from your computer to your Android device.

- Disconnect the USB cable.

- Look up the application icon in the device’s memory and open it to start the installation process.
8. Conclusion

In today's world, due to a great number of standard applications that are present, it is hard to be innovative. Despite that, developers of modern mobile applications have found solution in the use of numerous sensors and other devices that are present in a number of smartphones. Such a modern application has been discussed in this paper. The task of this paper was to design, implement and test an application for LED patterns recognition on Android smartphones. The built application consists of two components, the client and the server. The process of communication between the mentioned components was explained along with the OpenCV's most interesting parts which were used for this paper purposes.

Making this paper truly impressed me because it has enriched my knowledge for a great new design experience that consists of two large parts, the client and the server. More importantly, I have learned a lot about using modern technologies from the domain of computer vision that is anticipated even greater growth and the prospective future. Finally, I would dare to conclude that this topic gave me a glimpse of the great and interesting world of the modern mobile applications development. If I could go back to choose a topic once again, I definitely would not change my mind.
9. Literature

https://hr.wikipedia.org/wiki/Android_(operacijski_sustav)


https://androidsvet.wordpress.com/2013/04/18/struktura-android-aplikacija/


https://www.raywenderlich.com/2705552-introduction-to-android-activities-with-kotlin

https://www.codecademy.com/articles/what-is-rest

https://www.w3schools.com/js/js_json_intro.asp

https://www.w3schools.com/whatis/whatis_json.asp

https://www.freecodecamp.org/news/what-exactly-is-node-js-ae36e97449f5/

https://www.w3schools.com/nodejs/

https://nodejs.dev/
https://en.wikipedia.org/wiki/MongoDB

https://www.mongodb.com/what-is-mongodb

https://www.w3schools.com/nodejs/nodejs_mongodb_find.asp

http://www.discoversdk.com/blog/opencv-with-android-studio

[16] Dissolve, Computer and server room rack panel, June 2019

[17] Photopos, Tutorial – change photo depth, June 2019
https://www.photopos.com/PhotoPosPro_FreePhotoEditor_v3/Help%20Pages/Tutorials/ChangePhotoDepthZoomIn.aspx

[18] Opencv.org, OpenCV, June 2019
https://opencv.org/

https://en.wikipedia.org/wiki/OpenCV

https://opencv.org/about/

[21] Kickstarter, AI courses by OpenCV.org, June 2019
https://www.kickstarter.com/projects/satyamallick/ai-courses-by-opencvorg
10. Summary

Title: An Application for LED Patterns Recognition with a Smartphone Camera

The Android Operating System is based on the Linux Operating System. It was originally developed by Android, Inc. The Android Operating System architecture consists of several layers. These are: Linux Kernel, Libraries, Android Runtime, Application Framework, Applications. From a very basic point of view, it is easy to notice that every single Android application consists of two key parts. The first part includes so-called “Java documents”, while the second part refers to the design of the graphical user interface (GUI) which is designed in XML (Extensible markup language) language. Android application directory has a tree-shaped structure. Some of the more important directories are: java/, res/, Gradle Scripts/, Manifests/. Each Android application is made up of several components that can communicate with other components within the same or other applications as needed. These are: Activities, Services, Broadcast receiver, Content provider.

REST is an architectural style for providing standards between computer systems on the web. A very important feature of the systems that follow the REST paradigm is that they are stateless which means that the server does not need to know anything about the client state and vice versa. When using the REST architecture, clients send requests in order to modify or retrieve resources and servers send responses to those requests. A request generally consists of several parts, such as: An HTTP verb, a header, a path to source, a message body containing data. There are four basic HTTP verbs that are used in the requests: GET, PUT, POST, DELETE. Responses from the server contain status codes to alert the client to information about the success of the operation. The most popular status codes are: 200 – OK, 201 – CREATED, 400 – BAD REQUEST, 403 – FORBIDDEN, 404 – NOT FOUND, 500 – INTERNAL SERVER ERROR.

JSON is a lightweight format for storing and transporting data. JSON is often used when data is sent from the server to the client and vice versa. JSON is simple and easy to understand.
Node.js is an open-source and cross-platform JavaScript runtime environment built on Chrome’s V8 JavaScript engine. It can be used for almost any kind of project. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight.

MongoDB is a cross-platform document-oriented database program developed by MongoDB, inc. under SSPL licence. It is classified as a NoSQL database. MongoDB stores data in flexible, JSON-like documents. The MongoDB document model maps directly to the objects in your application code, so it is always possible to work with predefined object classes. Due to the fact that MongoDB is a distributed database at its core, it has high availability and horizontal scaling possibility.

OpenCV (Open Source Computer Vision Library) is an open source and cross-platform computer vision software library. It is originally developed by Intel and released under the BSD 3-Clause License. OpenCV was built to accelerate the use of machine perception in the commercial products and to provide a common infrastructure for computer vision applications. The OpenCV library consists of more than 2500 optimized algorithms, which includes a comprehensive set of computer vision and machine learning algorithms. The OpenCV4Android Library is a version of the OpenCV library that is customized and optimized for work on mobile devices with Android Operating System.

LED Pattern Detector is a mobile application designed for the Android Operating System that is used to determine the device error according to the frequency of blinking of its certain LED light source. The whole application consists of three activities. That is StartupActivity, MainActivity and ResultsActivity. StartupActivity is an activity that handles remote database synchronization process. MainActivity is an activity that implements OpenCV’s CvCameraViewListener2 interface and whose main task is to process each camera frame. It is separated to six different modes. These are: InitialMode, LedDetectingMode, ManualCalibrationMode, MeasuringMode, PreAutoCalibrationMode, PreManualCalibrationMode. ResultsActivity’s task is quite simple. It calculates LED source blinking frequency based on the total blinks number and the measuring duration, after which it displays the corresponding description of the calculated frequency which was obtained from the local database.
Making this paper truly impressed me because it has enriched my knowledge for a great new design experience that consists of two large parts, the client and the server. More importantly, I have learned a lot about using modern technologies from the domain of computer vision that is anticipated even greater growth and the prospective future.

11. Keywords

Java, JavaScript, LED source, Pattern, Pattern detector, Android, Android history, Android architecture, Linux, Application, Mobile application, Android Studio, OpenCV, Computer vision, Client, Server, Database, Local database, Remote database, JSON, Node.js, MongoDB, REST, Real-time processing
12. Sažetak

Naslov: Aplikacija za prepoznavanje LED uzoraka korištenjem kamere na pametnim telefonima


REST je arhitekturni stil koji nudi standarde za komunikaciju između računala na internetu. Važno svojstvo sustava koji prate REST standard jest da oni ne ovise o stanju komponenti s kojima komuniciraju, (engl. “stateless”), što znači da poslužitelj ne mora znati ništa o stanju klijenta i obrnuto. Pri korištenju REST arhitekturnog stila klijenti šalju zahtjeve poslužitelju za dohvaćanjem nekog resursa, a poslužitelj odgovara na te zahtjeve. Zahtjev se obično sastoji od nekoliko dijelova. To su: HTTP operacija, zaglavlje, putanja do resursa i tijelo.

Postoje četiri glavne HTTP operacije. To su: GET, PUT, POST, DELETE. Odgovori koji su pristižu od poslužitelja sadrže statusni kod, koji se koristi kako bi se klijenta obavijestilo o uspješnosti izvršavanja operacije. Najpopularniji statusni kodovi su: 200 – OK, 201 – CREATED, 400 – BAD REQUEST, 403 – FORBIDDEN, 404 – NOT FOUND, 500 – INTERNAL SERVER ERROR.

JSON je jednostavni format za prijenost podataka. Obično se koristi kako bi se podatci prenjeli s klijenta na poslužitelj i obrnuto. JSON je poprilično jednostavan i lako razumljiv.
Node.js je JavaScript radno okruženje otvorenog koda koje se može koristiti za gotovo bilo koju vrstu projekata. Node.js se temelji na događajima (engl. event-driven), neblokirajući je i stoga ne preoprećuje sustav na kojem se izvodi (engl. lightweight).

MongoDB je multiplatformska baza podataka orijentirana na rad s dokumentima koju je prvobitno razvila tvrtka MongoDB, Inc. pod SSPL licencom. MongoDB se kvalificira kao NoSQL baza podataka koja pohranjuje podatke u neformalnom obliku sličnom JSON formatu. Objekti koji su pohranjeni u bazi podataka mapiraju se direktno na objekte u programskom kodu, stoga je uvijek moguće raditi samo s predefiniranim razredima objekata. MongoDB je baza podataka za koju je karakteristična visoka dostupnost, te mogućnost horizontalnog skaliranja.

OpenCV (Open Source Computer Vision Library) je multiplatformska knjižnica otvorenog koda iz domene računalnog vida. Prvu inačicu izradio je Intel, a izdana je pod BSD 3-Clause licencom. Knjižnica OpenCV sadrži više od 2500 optimiranih algoritama, što uključuje i određen dio namijenjen za računalni vid i strojno učenje. OpenCV4Android knjižnica je verzija OpenCV knjižnice koja je prilagođena i optimirana za rad na mobilnim uređajima s operacijskim sustavom Android.

LED Pattern Detector je mobilna aplikacija za operacijski sustav Android koja se koristi za određivanje pogreške uređaja prema frekvenciji treptanja ugrađenih LED izvora. Aplikacija se sastoji od tri aktivnosti. To su: StartupActivity, MainActivity i ResultsActivity. StartupActivity je aktivnost zadužena za proces sinhronizacije s udaljenom bazom podataka. MainActivity je aktivnost koja implementira sučelje CvCameraViewListener2 iz programске knjižnice OpenCV, čiji je zadatak da obrađuje okvire koje je kamera zabilježila. Podijenjena je u šest različitih modova. To su: InitialMode, LedDetectingMode, ManualCalibrationMode, MeasuringMode, PreAutoCalibrationMode, PreManualCalibrationMode. Zadatak aktivnosti ResultsActivity poprilično je jednostavan, a to je da izračuna frekvenciju treptanja LED izvora na temelju izmjerenog broja treptanja i duljine trajanja mjerenja, te da potom dohvati (iz lokalne baze podataka) i prikaže opis odgovarajuće pogreške.
Izrada ovog rada veoma me se dojmila prvenstveno zbog toga što je obogatila moje znanje za jedno novo, veliko iskustvo dizajniranja aplikacije koja se sastoji od dva ključna dijela (korisničkog i poslužiteljskog), te ono još važnije, izrada ovog rada omogućila mi je susret s korištenjem modernih i općeprihvaćenih tehnologija kojima se predviđa još veći rast i svjetla budućnost.

13. Ključne riječi

Java, JavaScript, LED izvor, Uzorak, Detektor uzoraka, Android, Android povijest, Android arhitektura, Linux, Aplikacija, Mobilna aplikacija, Android Studio, OpenCV, Računalni vid, Klijent, Poslužitelj, Baza podataka, Lokalna baza podataka, Udaljena baza podataka, JSON, Node.js, MongoDB, REST, Obrada u stvarnom vremenu.