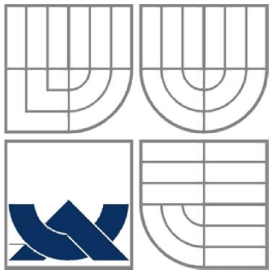
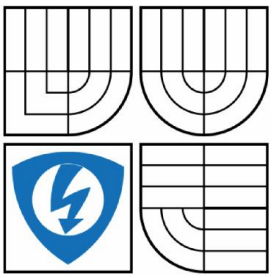


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

ZPRACOVÁNÍ OBRAZU V SYSTÉMU ANDROID - DETEKCE A ROZPOZNÁNÍ OBLIČEJE

IMAGE PROCESSING USING ANDROID DEVICE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
ING. SERGEI KORCHAKOV

VEDOUCÍ PRÁCE
SUPERVISOR
ING. PETER HONEC, PH.D.

BRNO 2014

VYSOKÉ UČENÍ
TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

Ústav automatizace a měřicí techniky

Diplomová práce

magisterský navazující studijní obor
Kybernetika, automatizace a měření

Student: Ing. Sergei Korchakov
Ročník: 2

ID: 141938
Akademický rok: 2013/14

NÁZEV TÉMATU:

Zpracování obrazu v systému Android - detekce a rozpoznání obličeje

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce bude vytvoření GUI a algoritmů detekce a rozpoznání obličejů pro zařízení Android. Zadání lze shrnout do následujících bodů:
1. Zpracujte rešerši týkající se OS Android a základních metod zpracování obrazu se zaměřením na detekci a rozpoznání obličeje..
2. Navrhněte a vytvořte GUI aplikaci, která bude zajišťovat spolupráci s HW prostředky zařízení, pořizovat obraz z kamery zařízení a spouštět algoritmy.
3. Navrhněte a otestujte algoritmy pro detekci a rozpoznání obličeje v reálné scéně.
4. Implementujte do aplikace pro OS Android a otestujte funkcionalitu.

DOPORUČENÁ LITERATURA:

Hlaváč, Šonka, Počítačové vidění.
Šonka, Hlávač, Boyle - IMAGE PROCESSING, ANALYSIS, AND MACHINE VISION,

Termín zadání: 10.2.2014 ***Termín odevzdání:*** 19.5.2014

Vedoucí práce: Ing. Peter Honec, Ph.D.
Konzultanti diplomové práce:

doc. Ing. Václav Jirsík, CSc.
předseda oborové rady

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ Fakulta elektrotechniky a komunikačních technologií Ústav automatizace a měřicí techniky

UPOZORNĚNÍ:

Abstrakt

Tato diplomová práce se zabývá zpracováním obrazu v platformě Android, respektive vývojem mobilní aplikace, která je schopná provádět detekci a rozpoznávání obličejů v reálné scéně.

V rámci projektu byla provedená rešerše současně existujících metod detekce obličejů, byly prozkoumány a porovnány standardní prostředky platformy Android pro detekci tváří (FaceDetector a FaceDetectionListener) a také knihovny JJIL, OpenIMAJ, OpenCV.

Pro rozpoznání obličejů byla použita knihovna OpenCV a vyzkoušeny tři algoritmy identifikace: FisherFaces, EigenFaces a Local Binary Patterns Histograms.

Na základě porovnání nejúspěšnější metody byly uplatněny ve vyvinuté aplikaci.

Klíčová slova

Detekce obličejů, rozpoznání obličejů, Java, Android, OpenCV, FaceDetector, FaceDetectionListener , Algoritmus Viola-Jones, Haarovy vlnky, filtrace obrazu, video stream, kamera, XML, FisherFaces, EigenFaces a Local Binary Patterns Histograms.

Abstract

This master's Thesis focuses on image processing on Android platform and development of an application, that is able to do face detection and recognition in real scene.

Thesis gives highlight of modern algorithms of face detection. It first examines and compares the standard features of Android platform (FaceDetector a FaceDetectionListener) and JJIL, OpenIMAJ, OpenCV libraries experiment, and presents the results.

For purposes of face recognition was selected OpenCV library. Three different algorithms of identification were tested: FisherFaces, EigenFaces a Local Binary Patterns Histograms.

Based on performance comparison best methods were implemented in developed application.

Keywords

Face detection, face recognition, Java, Android, OpenCV, FaceDetector, FaceDetectionListener , Viola-Jones Algorithm, Haar wavelet, image filtration, video stream, camera, XML, FisherFaces, EigenFaces a Local Binary Patterns Histograms.

KORCHAKOV S. Zpracování obrazu v systému Android - detekce a rozpoznání obličeje. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. Ústav automatizace a měřicí techniky, 2014. 48s., 2s. příloh. Vedoucí diplomové práce ING. PETER HONEC, PH.D.

Prohlášení

Prohlašuji, že svou diplomovou práci na téma Zpracování obrazu v systému Android - detekce a rozpoznání obličeje jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

V Brně 18.05.2014 Sergei Korchakov

Poděkování

Děkuji vedoucímu diplomové práce doc. Ing. Petru Honcovi, Ph.D. za účinnou pomoc a cenné rady při zpracování mé diplomové práce.

V Brně 18.05.2014 Sergej Korchakov

OBSAH

Seznam obrázků

Seznam tabulek

1 ÚVOD

Metody pro rozpoznání a detekci obličejů je v poslední době stále více se rozrůstající oblastí výzkumu, neboť tato oblast nachází stále větší uplatnění v různých oborech jako kriminalistika, identifikace, snímací technika. V posledních dvou desetiletích byl v této oblasti proveden velký výzkum a bylo studováno a ohlášeno více než sto strategií.

Cílem diplomové práce bude vytvoření programu pro zařízení Android, který bude využívat integrovanou kameru/fotoaparát k získávání obrazu, rešerše současných způsobů detekce tváři, následné porovnání a výběr knihoven detekce a rozpoznání obličejů, dostupných k implementaci na platformě Android a samotná realizace vybraných algoritmu.

Toto téma jsem si zvolil jako velmi zajímavé spojení dvou oboru – počítačového vidění a vývoje mobilních aplikací. Platforma Android měla v srpnu 2013 bezmála 80% podíl na trhu s chytrými telefony[1]. Podíl smartphonů, neboli „chytrých telefonů“ na trhu stále stoupá a kvůli tomu roste i zájem o mobilní aplikace. Na základě svých zkušeností s programováním v jazyce Java a znalostem moderních technologií vývoje v oblasti Java aplikací jsem považoval tuto práci jako možnost rozšířit své vědomostí v těchto oborech.

V první kapitole budou probrány platforma Android, její vlastnosti a architektura Android aplikací.

Druhá kapitola je věnována rešerši současně dostupných pro platformu Android knihoven, které se používají pro detekcí objektů, respektive obličejů, v obraze. Zde bude provedená analýza úspěšností a porovnání existujících knihoven. Výsledkem kapitoly bude výběr nejúspěšnější metody, která pak bude použita v projektu.

Třetí kapitola popisuje algoritmy rozpoznání lidských tváří, které se dají použit na platformě Android. V této části se probírají vlastnosti algoritmů a problematika jejich implementace. Následně se provádí test kvality predikce v závislostí na velikosti databáze trénovacích obličejů.

Vývoj aplikace a zásadní aspekty implementace zvolených knihoven vzhledem k možnostem platformy Android budou popsány v poslední kapitole. Také zde budou rozebrány problémy detekce a rozpoznání objektů ve video streamu.

Předpokládá se, že v rámci diplomového projektu bude vyvinuta aplikace na platformě Android, která bude schopná detekovat obličeje v reálné scéně, provádět identifikaci tváří a informovat uživatele o možné schode nalezeného obličeje s některým z databázi. Navíc aplikace bude poskytovat možnost rozšíření báze obličejů.

2 PLATFORMA ANDROID

Android je v současnosti nejrozšířenější a zároveň asi nejrychleji rostoucí mobilní open source platforma. Zahrnuje v sobě operační systém (založený na jádru Linux), middleware, uživatelské rozhraní a aplikace. Při vývoji systému byla brána v úvahu omezení, kterými disponují klasické mobilní zařízení jako výdrž baterie, menší výkonnost a málo dostupné paměti. Zároveň bylo jádro Androidu navrženo pro běh na různém hardwaru. Systém tak může být použit bez ohledu na použitý chipset, velikost či rozlišení obrazovky. Samotná platforma Android dává k dispozici nejen operační systém s uživatelským prostředím pro koncové uživatele, ale i kompletní řešení nasazení operačního systému (specifikace ovladačů aj.) pro mobilní operátory a výrobce zařízení a v neposlední řadě pro vývojáře aplikací poskytuje efektivní nástroje pro jejich vývoj – Software Development Kit. Platforma Android dnes (srpen 2013) má na trhu s chytrými telefony 80% podíl.

Architektura operačního systému Android je rozdělena do 5 vrstev. Každá vrstva má svůj účel a nemusí být přímo oddělena od ostatních vrstev.

Nejnižší vrstva architektury je jádro operačního systému, které tvoří abstraktní vrstvu mezi používaným hardwarem a zbytkem softwaru ve vyšších vrstvách. Jádro systému Androidu je postaveno na Linuxu ve verzi 2.6. Je využito jeho mnoha vlastností, jako podpora správy paměti, správa sítí, zabudované ovladače nebo správy procesů, například souběžného běhu aplikací, které běží jako samostatné procesy s oprávněním stanoveným systémem. Tato vlastnost přispívá ke stabilitě a také ochraně systému. Naopak systém nepodporuje grafické uživatelské rozhraní X Window System a ani úplnou sadu GNU knihoven. Důvodem použití jádra Linux byla také vlastnost poměrně snadného sestavení na různých zařízeních a tím zaručená přenositelnost.

Další vrstvou jsou knihovny, které jsou napsány v C nebo C++ kódu a využívají je různé komponenty systému. Tyto funkce jsou vývojářům poskytnuty prostřednictvím Android Application Framework.

Android Runtime vrstva obsahuje aplikační virtuální stroj zvaný Dalvik. Dalvik Virtual Machine (DVM) je registrově orientovaná architektura, využívá základních vlastností Linuxového jádra, jako je správa paměti nebo práce s vlákny. V této vrstvě jsou také obsaženy základní knihovny programovacího jazyka Java. Knihovny se svým obsahem blíží platformě Java Standard Edition. Hlavní rozdíl je v nepřítomnosti knihoven pro uživatelské rozhraní (Abstract Window Toolkit a Swing), které byly nahrazeny knihovnami uživatelského rozhraní pro Android nebo přidání knihovny Apache pro práci se sítí. Překlad aplikace napsané pro Android probíhá zkompilováním zdrojového Java kódu do Java byte kódu pomocí stejného kompilátoru, jako je používán v případě překladu Java aplikací. Poté se překompiluje Java byte kód pomocí Dalvik kompilátoru a výsledný Dalvik byte kód je spuštěn na DVM. Každá spuštěná Android aplikace běží ve svém vlastním procesu, s vlastní instancí DVM.

Vrstva Application framework je pro vývojáře nejdůležitější. Poskytuje přístup k velkému počtu služeb, které mohou být použity přímo v aplikacích. Tyto služby mohou zpřístupňovat data v jiných aplikacích, prvky uživatelského rozhraní, upozorňovací stavový řádek, aplikace běžící na pozadí, hardware používaného zařízení a mnoho dalších služeb a funkcí. Základní sada služeb zahrnuje především:

- Sada prvků View – Tyto prvky jsou použity pro sestavení uživatelského rozhraní jako seznamy, textové pole, tlačítka, checkboxy a jiné.
- Content providers – Umožňuje přístup k obsahu (např. kontakty) jiných aplikací.
- Resource manager – Poskytuje přístup „nekódovým“ zdrojům, jako jsou řetězce, grafika, přidané soubory.
- Notification manager – Umožňuje všem aplikacím zobrazit vlastní upozornění ve stavovém řádku.
- Activity manager – Řídí životní cyklus aplikací a poskytuje orientaci v zásobníku s aplikacemi.

Nejvyšší vrstvu systému tvoří základní aplikace, které využívají běžní uživatelé. Může jít o aplikace předinstalované nebo dodatečně stažené z Android Marketu. Například e-mailový klient, SMS program, kalendář, mapy, prohlížeč, kontakty a další aplikace i od „třetích“ stran. [1]

2.1 Architektura Android aplikací

Architektura Andriod je framework orientovaná (framework-based) na rozdíl od volné (free-style) architektury. Odlišnost je v tom, že se volné aplikace v Java začínají s třídy, která má metodu main() a dále fungují tak, jak je navrhl vývojář. Naopak framework-based aplikace jsou založený na existujícím frameworku. Vývoj se redukuje na rozšíření nějakých tříd nebo realizaci rozhraní poskytovaných touto soustavou. Takováto aplikace se nemůže spustit mimo framework nebo bez něho. Příkladem mohou být webové Java aplikace, ve kterých developer implementuje rozhraní Servlet nebo rozšiřuje jednu z realizací. Také dobrým příkladem je aplikační software Eclipse RCP, kde vývojář rozšiřuje jednu ze tříd Editor či View.

Framework-based architektura omezuje svobodu programátorů, nařizuje jim co a jak je třeba dělat. Ale ve výsledku mizí opakující se kusy kódu (boilerplate code) a vývojář je nucen se řídit návrhovými vzory.

Jako interakce mezi uživatelským rozhraním a jeho logikou Android používá návrhový vzor «Model-View-ViewModel» (MVVM). V současnosti jedná z nejlepších architektur GUI-aplikací. MVVM byla navržená s cílem rozdělit práce designéru a programátoru, co není možné v běžných frameworcích jako Java Swing nebo Visual C++ MFC.

Architektura MVVM řeší tento problém jasným rozdělením zodpovědností:

- vývojem uživatelského rozhraní se zbývá designér GUI s použitím víceméně přirozené technologie XML,
- logiku GUI realizuje vývojář jak komponentu ViewModel,
- funkční vztahy mezi rozhraním a ViewModel fungují pomocí bindingů, což jsou v podstatě pravidla typu «jestli je zmáčknuto tlačítko A, má se zavolat metoda onButtonAClick() z ViewModel». Bindingy mohou být napsány v kódu nebo deklarovány v XML.

Různé části Android aplikace mohou volat jeden druhého a vzájemně působit jen formálně. Framework Android má několik vzoru interakce:

- výměna zprav pomocí třídy Intent,
- naslouchání s použitím tříd Intent a BroadcastReceiver,
- Late binding a následné volání metod se používá pro přístup ke Content Providerům a lokálním servisům (Services),
- Late binding a meziprocesní kooperace (Inter-process Procedure Communication, IPC) s použitím Android Interface Definition Language (AIDL).[2]

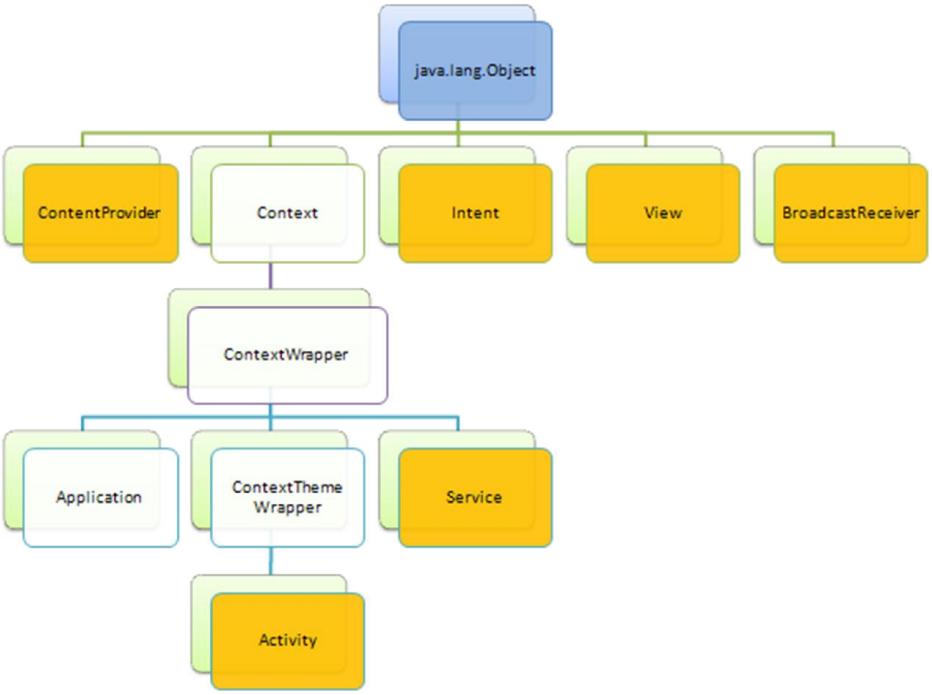
2.2 Základní části aplikace Android

Základní stavební kameny v aplikacích Android jsou komponenty aktivity reprezentující obrazovku, service umožňující provádět akce na pozadí, content providers poskytující přístup k datům a broadcast reciever reagující na příchozí oznámení. Všechny tyto komponenty musí být definovány v souboru AndroidManifest.xml, uloženém v kořenovém adresáři projektu. [1]

Obecně aplikace se skládá z

1. Java tříd, jež jsou potomky základních tříd Android SDK (View, Activity, ContentProvider, Service, BroadcastReciever, Intent), a Java tříd, které nemají rodiče v SDK.
2. Manifestu aplikace.
3. Zdrojů, jako jsou řetězce, grafika.
4. A přidaných souborů.

Na obrázku 1 je představená hierarchie základních tříd Android SDK. Zde žlutým jsou označený nejčastěji používané, respektive ty, se kterými vývojář pracuje bezprostředně.



Obrázek 1. Hierarchie základních tříd v Android SDK[2]

View je rodičovská třída pro všechny widgety uživatelského rozhraní. GUI aplikace je strom instancí této třídy. Rozhraní je definováno pomocí XML (soubory vrstev, layout files) a se za běhu aplikace automaticky mění (inflate) v strom odpovídajících objektů.

Manifest je v projektu soubor AndroidManifest.xml. Obsahuje specifikace aplikace, jako je název nebo verze. Pak také obsahuje specifikace jednotlivých komponent aplikace - aktivit, služeb, content provideru a broadcast receiverů.

Každé moderní GUI používá zdroje. Android není výjimkou. Typy zdrojů jsou:

- obrázky,
- vrstvy GUI (XML soubory),
- deklarace menu (rovněž XML),
- a textové řetězce.

Způsob, kterým se spojují zdroje s Android aplikaci je velmi vzácný. Zpravidla v Java zdroj lze identifikovat textovým řetězcem. Takový řetězec může obsahovat cestu nebo název souboru anebo ID. Problém je v tom, že chyby v takových řetězcích nejde projevit při kompilaci. V Googlu přišli k velmi elegantnímu řešení.

Identifikátory jsou v androidu statickými vlastnostmi třídy R. To je třída, která je generována automaticky na základě obsahu složky res. Například pomocí R.layout.new_layout se můžete odkázat na soubor new_layout.xml ve složce /res/layout.

3 DETEKCE OBJEKTŮ V OBRAZE

V této kapitole bude provedena rešerše současně dostupných pro platformu Android knihoven pro detekcí objektů, respektive obličejů, v obraze. Následně bude vybrána vhodnější pro použití v tomto diplomovém projektu.

Současně existuje spousta algoritmů detekci obličejů. Některé z nich jsou dostupné ve tvaru vědeckých článků s popisem matematických modelů, jiné pouze jako hotové softwarové řešení. Například knihovna OpenCV nebo knihovna android.media.FaceDetector, která je součásti platformy Android.

3.1 Detekce obličejů standardními prostředky platformy Android

Platforma Android poskytuje dva způsoby detekci obličejů. Jeden je starší interface, který byl dostupný již v API verze 1, naopak druhé rozhraní bylo předloženo později v API verze 14 (Android 4.0).

3.1.1 android.media.FaceDetector

Rozhrání bylo dostupné od samých prvních verzí API a poskytuje základní funkčnost pro hledaní tváře v obraze. Algoritmus funguje tak, že přijímá na vstupu objekt typu Bitmap a vrací seznam, každý prvek kterého je unikátní nalezený obličej.

Inicializace algoritmu vypadá následovně.

```
BitmapFactory.Options options = new BitmapFactory.Options();
//provedeme downsampling vstupniho obrazu
options.inSampleSize = SUBSAMPLING_FACTOR;
Bitmap bmp = BitmapFactory.decodeByteArray(imageBytes, 0, imageBytes.length, options);

// je dulezite zkonvertovat obrazek do bitmapu s parametrem RGB_565
// jinak decoder nebude schopen zpracovat vstupni data
Bitmap mFaceBitmap = bmp.copy(Bitmap.Config.RGB_565, true);
int mFaceWidth = mFaceBitmap.getWidth(); // sirska
int mFaceHeight = mFaceBitmap.getHeight(); // vyska

final int MAX_FACES = 4; //maximalni pocet obliceju na snimku
```

Pak stačí vytvořit novou instanci objektu FaceDetector a zavolat metodu hledání.

```
facesFD = new FaceDetector.Face[MAX_FACES]; //nalezene obliceje
FaceDetector fd = new FaceDetector(mFaceWidth, mFaceHeight, MAX_FACES);
int count = fd.findFaces(mFaceBitmap, facesFD); //provest detekci
```

Každý prvek výstupního seznamu obsahuje informaci o středu obličeje a vzdálenosti mezí očima. Kromě toho lze určit pozici, respektive o kolik stupňů je tvař otočená, a pravděpodobnost, že toto je skutečně lidský obličej. Mezní hodnota pro tento algoritmus je 40%. Objekty s menší pravděpodobnosti jsou se odmítají [5].

Nevýhodami tohoto způsobu jsou chybějící informace o tom, jak samotný algoritmus funguje, absence možnosti rozšíření o další funkce, nutnost předem definovat

maximální počet obličejů. A také to, že metoda pracuje se statickými obrazy nikoliv s video potokem, což výrazně degraduje výkon.

3.1.2 android.hardware.Camera.FaceDetectionListener

Od verze Android 4.0 je dostupné nové rozhrání, které umožňuje programátorům vytvářet listener třidu, která bude hledat obličeje, a připojit ji k objektu Camera. Pomocí speciálního callback objektu listener dostává seznam objektů, které reprezentují unikátní obličeje ve snímku při každém obnovení náhledu kamery. Tento interface je dostupný od verze API level 14, avšak kamera zařízení to také musí podporovat. Ověřit zda je tato metoda detekci podporována lze pomocí metody

camera.getParameters().getMaxNumDetectedFaces()

Pokud je výsledek víc než nula, tak podporuje.

Aby FaceDetectionListener byl schopen dostavat informaci, je ovšem potřeba vytvořit SurfaceHolder pro zobrazení náhledu kamery nebo SurfaceTexture, který se používá u OpenGL kreslení [6].

Před spuštěním detektoru obličeje musíme nastartovat náhled kamery, pak samotný detektor.

```
mCamera.startPreview();
mCamera.startFaceDetection();
```

Pro zastavení

```
mCamera.stopFaceDetection();
mCamera.stopPreview();
```

Definice callbacku je následovná

```
FaceDetectionListener faceDetectionListener = new FaceDetectionListener() {
    @Override
    public void onFaceDetection(Face[] faces, Camera camera) {
        if (faces.length == 0) {
            // No Face Detected!
            detectedFaces = null;
        } else {
            // Face Detected
            detectedFaces = faces;
        }
    }
};
```

Připojíme ho ke kameře

```
mCamera.setFaceDetectionListener(context.faceDetectionListener);
```

Informace o každé tváři získaná z callback obsahuje souřadnice očí, úst, obdélníku kolem obličeje a také unikátní identifikátor objektu a pravděpodobnost v procentech.

Metoda je rychlá, dostatečně spolehlivá, avšak algoritmus není otevřený a chybí možnost rozšiřování.

3.2 JJIL

Jon's Java Imaging Library je knihovna s otevřeným kódem, která je napsána ryze v jazyce Java. Knihovna je zamířená na mobilní aplikace, má podporu Android. Autor tvrdí, že kód pro detekcí obličejů dobře funguje s kamerami telefonů. Ale bohužel JJIL není moc spolehlivá, je velmi náchylná k šumu v obrazech. Navíc poslední aktualizace knihovny byla v roce 2008 [7].

3.3 OpenIMAJ

OpenIMAJ je „The Open Intelligent Multimedia Analysis toolkit for Java“ a je open source knihovna, která je trochu podobná OpenCV ale dokáže zpracovávat navíc audio a video informaci. Je kompletně napsána v Javě. Používá algoritmus Viola-Jones k detekci objektů a obsahuje několik předem naučených haarových kaskád, včetně určených k nalezení obličejů a očí.

OpenIMAJ prohlašuje, že bude fungovat na Android zařízeních [8]. Však během testu se zjistilo, že kód pro detekci nejde zkompilovat, neboť knihovna používá k načtení XML souborů třídy z balíku javax.xml.stream, které nejsou v verzi Java pro Android.

3.4 OpenCV

OpenCV (Open Computer Vision) je knihovna napsaná v C a C++ původně vyvinutá společnosti Intel. Knihovna poskytuje široký výběr možností v oblasti počítačového vidění. Například detekce obličejů, sledování pohybu, rozpoznávání gest. Kromě toho obsahuje spoustu užitečných funkcí, jako jsou například práce s kamerou a maticová matematika. OpenCV byla portována na platformu Android s použitím Java Native Interface (JNI).

Pro detekci tváří zde je implementace algoritmu Viola-Jones[9], který přijímá jako vstup trénovací soubor pro detekci určitých vlastností. Balík OpenCV také obsahuje předem vygenerované trénovací soubory pro hledání různých objektů: obličejů, očí apod.

3.4.1 Algoritmus Viola-Jones

Objektový detektor Viola-Jones byl poprvé představen P. Violou a M. Jonesem v roce 2001. Jedná se o detektor objektů pracující s šedo-tónovými obrazy, který se skládá ze tří základních částí: integrálního obrazu, Haarovy vlnky a klasifikačního algoritmu AdaBoost. Výhodou tohoto detektoru je rychlost, dostatečná spolehlivost a značná nezávislost na osvětlení a velikosti sledovaného objektu. Z těchto důvodů je v praxi tento detektor často používán například při detekci obličejů a zároveň vzniká velká řada jeho modifikací (např. využití jiné než Haarovi vlnky, nahrazení algoritmu AdaBoost algoritmem GentleBoost a jiné) [11].

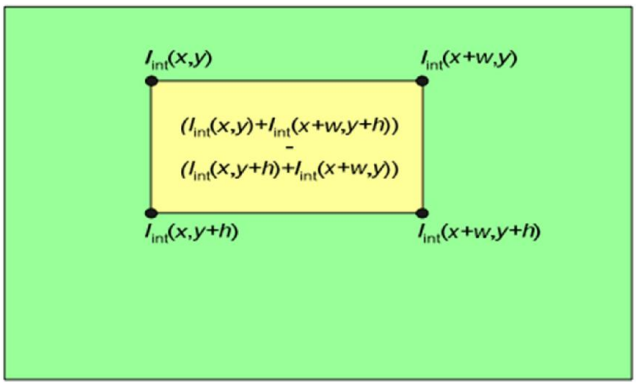
Integrální obraz

Integrální reprezentace obrazu umožňuje rychlé výpočty celkového jasu libovolných obdélníků ve vstupním obrazu, přičemž náročnost této operace je konstantní a nezaleží ani na velikosti obrazu, ani na jeho poloze. Integrální obraz je matice stejné velikosti jako původní obraz. Do každého elementu se ukládá suma intenzit jasu všech pixelů, které se nachází od pozice x,y tohoto elementu k levému hornímu rohu obrazu.

Matematický zápis integrálního obrazu $I_\Sigma(X)$, kde $x = (x,y)^T$ je pozice v integrálním obraze a I(i, j) představuje vstupní obrázek, je uveden na rovnice (3.1).

$$I_\Sigma(X) = \sum_{i=0}^{i \le x} \sum_{j=0}^{j \le y} I(i,j) \tag{3.1}$$

Výpočet hodnot jednotlivých příznaků se pak výrazně zjednoduší, protože na výpočet sumy libovolného obdélníku v obraze postačí dvě operace sčítání a jedna operace odčítání viz. obr. 2, kde x, y jsou počáteční souřadnice a w, h jsou šířka a výška požadovaného obdélníku.



Obrázek 2. Příklad výpočtu sumy libovolného obdélníku pomocí integrálního obrazu [11]

Haarovy vlnky

Snahou detektoru Viola-Jones je získat velkou řadu jednoduchých příznaků s minimálními výpočetními nároky. Takovým typem příznaků jsou příznaky založené na principu podobnému definici Haarově vlnky (tzv. Haar-like features), viz obr. 3. Hodnota takového příznaku se tak vypočítá jako suma pixelů obrazu odpovídající světlé části, od které je odečtena sumy pixelů odpovídacích tmavé části. Tyto vlnky mohou být tvořeny dvěma (hranový příznak), třemi (čárový příznak) či čtyřmi (diagonální příznak) obdélníkovými oblastmi. Jednotlivé příznaky jsou použity na celý vstupní obraz, přičemž zároveň dochází ke změně velikostí jednotlivých příznaků (tj. velikosti jednotlivých obdélníků) z velikosti 1×1 až na velikost odpovídající vstupnímu obrazu. To znamená, že pro vstupní obraz o rozměrech 19×19 dostáváme přibližně 64 tisíc hodnot příznaků, které jsou vstupem učícího procesu klasifikačního algoritmu AdaBoost. Ten z nich potom vybere jen určité malé množství příznaků společněs se

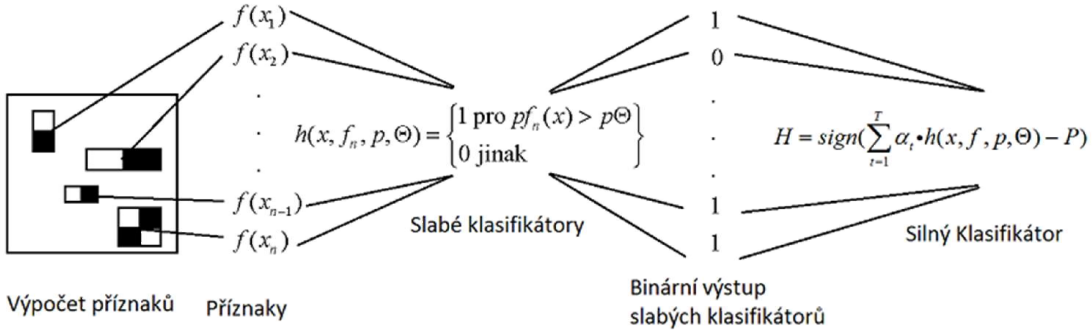
stejným počtem natrénovaných slabých klasifikátorů, pomocí nichž lze vstupní obraz vhodně klasifikovat (stejný příznak se ve výsledném silném klasifikátoru může vyskytovat i několikrát, ovšem pokaždé s jiným nastavením slabého klasifikátoru). Pouze toto malé množství příznaků je pak použito při samotné detekci. Uvedené typy příznaků patří k tzv. základním příznakům, v současnosti se používají i další typy ze základních příznaků odvozené [11].

Obrázek 3. Příznaky podobné Haarově vlnce [11].

AdaBoost

AdaBoost (název je zkratkou pro Adaptive Boosting) je klasifikační algoritmus, který vychází z metody strojového učení zvaného boosting. Cílem metody boosting je zlepšení klasifikační přesnosti libovolného algoritmu strojového učení. [11]

Nechť je informativní příznak jakýkoliv příznak, pomocí něhož je možné rozlišit tvář od pozadí s úspěšností větší než 50%. Klasifikátor, který využívá takové informativní příznaky, je nazýván slabým klasifikátorem. Pokud požijeme více slabých klasifikátorů, pak je možné ve výsledku dosáhnout vysoké klasifikační schopnosti. Slabé klasifikátory jsou spojeny v silném klasifikátoru (perceptronu). Vstupem silného klasifikátoru je váhovaný binární výstup slabých klasifikátorů. Tato skutečnost je vyobrazena na obrázku 4. [12]



Obrázek 4. Kaskádové zapojení klasifikátorů [12]

Výstup slabých klasifikátorů je vypočten dle vztahu:

$$h(x,f,p,\Theta) = 1 \text{ pro } f(x)p > p\Theta \text{ , jinak } 0,$$

kde $h(x,f,p,\Theta)$ je binární výstup slabého klasifikátoru, $f(x)$ je příznak, Θ je práh slabého klasifikátoru a p je parita. Slabé klasifikátory vstupují do silného klasifikátoru - perceptronu; váhy α_t jsou úměrné chybovosti jednotlivých slabých klasifikátorů při trénování. Můžeme tedy říci, že každý slabý klasifikátor má na celkovou klasifikaci vliv úměrný jeho chybovosti.

Silný klasifikátor potom má tvar:

$$H = sign(\sum_{t=1}^{T} \alpha_t \bullet h(x, f, p, \Theta) - P), \tag{3.2}$$

kde α_t jsou váhy slabých klasifikátorů, P je práh silného klasifikátoru vypočtený při trénování a T je celkový počet slabých klasifikátorů, ze kterých je složen silný klasifikátor. [12]

Výhodou Viola-Jones algoritmu je vysoká rychlost detekce, možnost adaptace a nezávislost na velikosti objektu. Nevýhodou je nutnost trénování algoritmu, které je výpočetně a časově náročně, a citlivost vůči rotaci a afinnímu zkreslení, které je u fotografií běžné.

3.4.2 JavaCV

Do jazyka Java knihovna OpenCV byla portována v rocích 2012-1013. I když aktuální verze je prohlášená za stabilní, není stoprocentně funkční. Navíc při portování byla výrazně přepsána. Proto jsem si zvolil JavaCV. Speciální wrapper pro nativní OpenCV, umožňující vývoj v jazyce Java. Nevýhodou tohoto postupu je nedostatek literatury a příkladů, slabé komunity a větší výsledná aplikace kvůli nutností kopírování do apk souboru všech používaných nativních knihoven. Avšak možnost použití metod OpenCV tzv. „napřímo“, jednoduchá integrace a prostor k rozšíření byly rozhodující.

Přesný popis integrace knihovny do projektu je uveden v [9]. Ve stručnosti jde o ryze kopírování jar souboru s kódem knihovny a potřebných nativních knihoven do projektu v Eclipse. Pak je nutné načíst do paměti základní objekt pro detekci.

```
// Preload the opencv_objdetect module to work around a known bug.
Loader.load(opencv_objdetect.class);
```

Dále vytvoříme instanci objektu klasifikátoru, která má odkaz na haarovou kaskádu jako argument.

```
classifier = new CvHaarClassifierCascade(cvLoad(classifierFile.getAbsolutePath()));
if (classifier.isNull()) {
    throw new IOException("Could not load the classifier file.");
}
storage = CvMemStorage.create();
```

Pak při každém obnovení náhledu kamery provádíme detekci. Výstupem této metody bude seznam nalezených obličejů a jejich souřadnic.

```
cvClearMemStorage(storage);
facesCV = cvHaarDetectObjects(grayImage, classifier, storage, 1.1, 3, CV_HAAR_DO_CANNY_PRUNING);
```

Pro detekci je lépe použit převzorkováný obrázek, čím zvýšíme přesnost výsledků. Ovšem obrázek má být šedotonový.

3.5 Porovnání výsledků

Shrnutím této kapitoly bude porovnání a výběr vhodné knihovny pro detekci objektů pro následné použití v rámci diplomového projektu.

OpenIMAJ a JJIL se prokázaly jako nestabilní a velice špatně realizované. I když dokážou fungovat na stolním počítači, nejsou dobré pro mobilní platformy. A proto nebyly do projektu implementovány.

V první fázi se testování a porovnání implementovaných knihoven provádělo na obličejích z databází The MIT-CBCL face recognition database, Cohn-Kanade database, Georgia Tech face database a AT&T Facedatabase (ORL Database of Faces). Detekce probíhala na zvětšených obrázcích tak, že tvář zabírala polovinu místa ve scéně.

Cohn-Kanade database je zamířená na detekci emocí, a proto všechny obličeje jsou s minimálním natočením. Úspěšnost všech metod na této databázi je 100%.

Georgia Tech face database a AT&T Facedatabase jsou primárně určený k trénování rozpoznávání obličejů. Obsahují několik snímků stejné osoby s různým natočením hlavy a tváře. Uhel natočení je relativně malý, a proto úspěšnost detekci pro všechny metody je také 100%.

Databáze MIT-CBCL je určená pro konstrukci 3D modelů hlav, to znamená, že osoby z ní byli vyfocený i z profilu. Výsledky testů pro tuto databázi jsou v tabulce 1. Je pochopitelné, že by se na snímku z profilu neměl nalézt žádný obličej. Na těchto fotografiích budeme testovat falešnou detekci.

Tab. 1. Test různých metod detekce na databázi obličejů The MIT-CBCL face recognition database

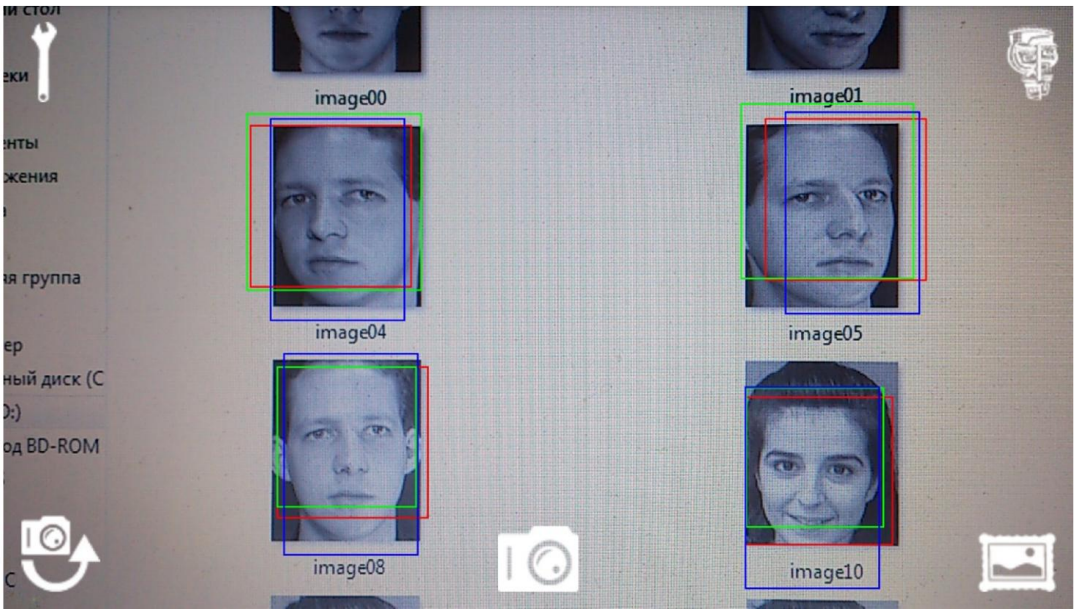
	True Positive, %	True Negative, %	False Positive, %	False Negative, %
FaceDetector	91,9	100	0	8,1
FaceDetectionListener	97,3	81	10,3	2,7
OpenCV	100	95,2	11,4	0

Z tabulky 1 je patrně, že FaceDetector na rozdíl od ostatních knihoven nedetekoval žádné falešně obličeje, ale při tom ukazuje nejmenší přesnost detekce. OpenCV dokázal najít všechny skutečné tváře, ale zároveň našel obličej i tam, kde neměl. Třeba v uších.

FaceDetectionListener má skoro podobnou „úspěšnost“ falešné detekce jako OpenCV, ale ukazuje menší přesnost.

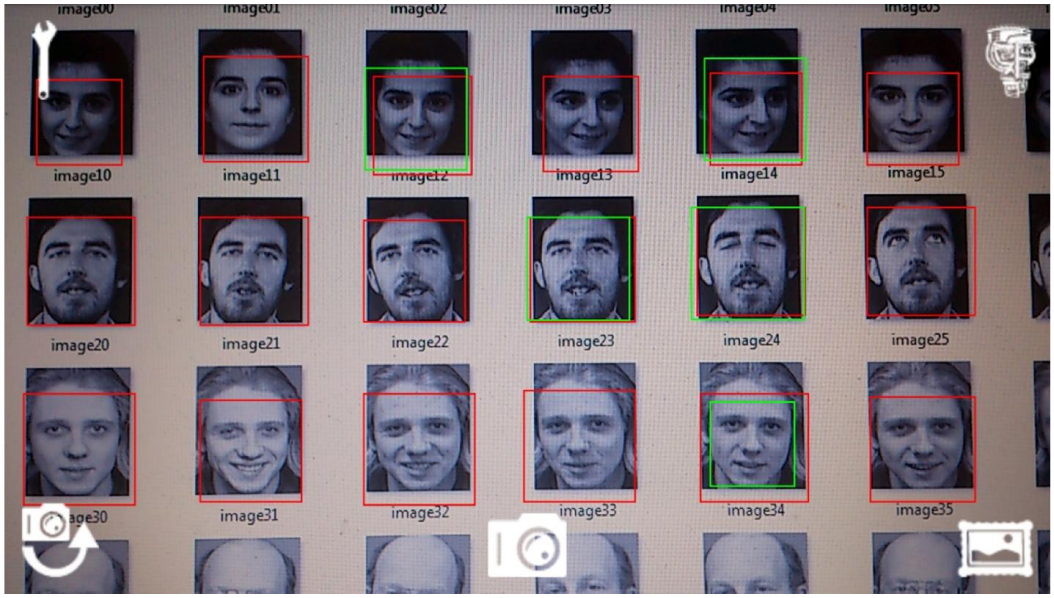
Podle tabulky můžeme určit celkovou úspěšnost detekci velkých a kontrastních obličejů. Však v reálné scéně musíme počítat s různou velikostí tváří, pozadím a počtem osob. Pro test vlivu těchto faktorů na úspěšnost detekce provedeme několik dalších pokusů.

Výsledky experimentu lze vidět na následujících obrázcích. Obrázek 5 obsahuje čtyři velké kontrastní obličeje z databáze AT&T Facedatabase (ORL Database of Faces). Všechny tři metody dokázaly nalézt tváře v této scéně.



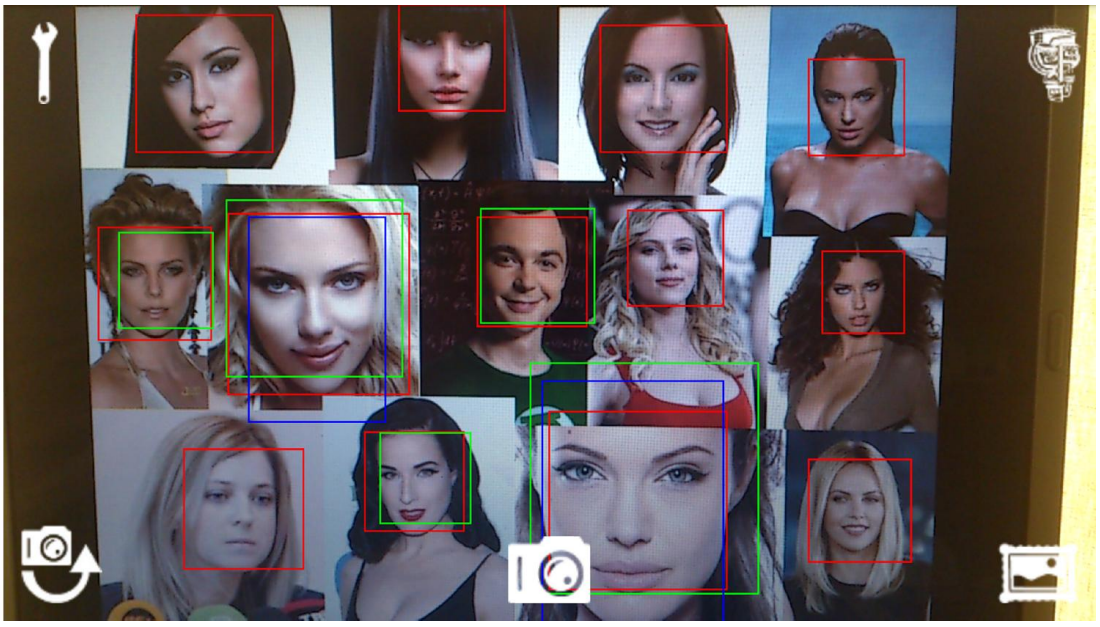
Obrázek 5. Detekce obličejů z databáze AT&T Facedatabase. Velké obličeje. FaceDetector – modrý obdélník, FaceDetectionListener – zelený, OpenCV – červený.

Na šestém obrázku jsou výsledky detekce obličejů ze stejné databáze AT&T Facedatabase, ale menšího rozměru. Je vidět, že FaceDetector nedokázal nalézt žádný obličej, FaceDetectionListener našel pouze pět. Nejlepší výsledek má knihovna OpenCV, která našla úplně všechno. Není možné stanovit příčinu nepřesnosti detekce knihovnou FaceDetectionListener kvůli tomu, že algoritmus detekce není otevřený. Metoda ukazuje různé výsledky na přibližně stejných obrázcích.



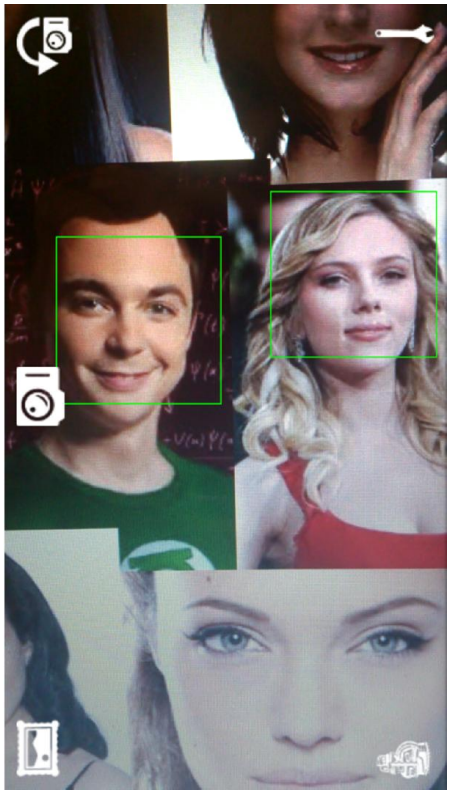
Obrázek 6. Detekce obličejů z databáze AT&T Facedatabase. Malé obličeje. FaceDetector – modrý obdélník, FaceDetectionListener – zelený, OpenCV – červený.

Na obrázku 7 jsou reálné neupravené fotografie osob. Obrázky mají různé velikosti, pozadí, kontrast. Tento test simuluje reálnou scénu. Zde se znovu nejlépe projevila knihovna OpenCV, která dokázala najít všechny obličeje. Výsledky ostatních dvou knihoven nejsou působivé. FaceDetectionListener detekoval pouze část tváří zdánlivě náhodných. FaceDetector zase našel jenom velké obličeje.



Obrázek 7. Simulace reálné scény. FaceDetector – modrý obdélník, FaceDetectionListener – zelený, OpenCV – červený.

Obrázek 8 je ukázkou toho, jak knihovny zvládají otočené obličeje. Evidentně pouze FaceDetectionListener nezávisí na uhlu natočení.



Obrázek 8. Detekce otočených obličejů. FaceDetector – modrý obdelník, FaceDetectionListener – zelený, OpenCV – červený.

Standardní detektory platformy Android jsou relativně stabilní a robustní, integrace do projektu je velmi snadná. Novější rozhrání android.hardware.Camera.FaceDetectionListener navíc nezávisí na uhlu natočení kamery. Avšak úspěšnost detekce v reálné scéně je velmi malá, chybí možnost vylepšení kvality detekce a případného rozšíření. Další nevýhodou je chybějící informace o algoritmu, který používají této knihovny. Po hlubší rešerše dokumentace a SDK Android se zjistilo, že existuje pěvné omezení maximálního počtu tváří pro FaceDetectionListener. Toto číslo se liší pro různé telefony či tablety. Dá se ho zjistit pomocí metody

```
mCamera.getParameters().getMaxNumDetectedFaces();
```

V našem případě se toto číslo rovná pěti.

V podstatě jedinou knihovnou, co nemá pěvné omezení a vyznačující se dobrou úspěšnosti detekce, je OpenCV, respektive JavaCV. A proto tento diplomový projekt bude postaven pravě na ní.

4 ROZPOZNÁVÁNÍ OBLIČEJŮ

V následující kapitole se budeme věnovat rozpoznávání lidských tváří. Knihovna OpenCV, která byla zvolena jako základní pro detekci obličejů, má rovněž i metody pro rozpoznávání vzorů. A to jak jednoduchých geometrických primitiv, tak i komplexních objektů složených z více prvků. Například státních poznávacích značek či obličejů.

OpenCV od verze 2.4 nabízí speciální novou třídu FaceRecognizer pro poznání tváří. V této kapitole probereme jak samotnou implementaci metody, tak i některé faktory, které ovlivňují kvalitu rozpoznávání.

Současně jsou k dispozice implementace tří algoritmů:

- Eigenfaces
- Fisherfaces
- Local Binary Patterns Histograms

4.1 Základy rozpoznávání tváří

Úloha rozpoznávání obličejů je velmi jednoduchá pro člověka. Pokusy ukazují, že dokonce i tři dny staré dítě rozlišuje známé tváře[13]. Ale pro počítač tento úkol již není takový snadný, neboť naše znalosti v oblasti identifikace obličejů jsou velmi malé. Lidský mozek má zvláštní nervové buňky zodpovědné za specifikaci lokálních vlastností scény, takových jak jsou například linie, hrany, úhly a pohyby. Nevšímáme okolní svět jako sadu izolovaných objektů, ale jako celek. Proto mozek musí kombinovat jednotlivé zdroje informace do použitelných šablon. Automatické algoritmy rozpoznávání obličejů by měly extrahovat tyto významné příznaky z obrázku, skládat je do použitelné podoby a pak umět podle nich provádět klasifikaci.

Geometrické metody

Metody identifikace obličejů založené na geometrických příznacích jsou asi nejvíc intuitivní. Příznaky jsou pozice očí, uší, nosu atd. Rozpoznávání se provádí jako výpočet euklidovské vzdálenosti mezí vektorem zkoumané a známé tváři. Podobné metody jsou robustní vůči světelným podmínkám, ale mají velkou nevýhodu v tom, že i malé afinní zkreslení ve vstupním obraze výrazně zhoršují přesnost identifikace. Některá studia geometrických metod rozpoznávání říká, že dokonce i dvaadvaceti dimenzionální vektor příznaku nenese dostatečné množství informace pro kvalitní identifikaci[14].

Metoda Eigenfaces

Metoda Eigenfaces je soubor vektorů používaných v počítačovém vidění k řešení problému rozpoznávání lidských obličejů. Eigenface byl vyvinut již v roce 1987. Soubor eigenface může být získán pomocí matematického procesu PCA (principal component analysis) na velkém souboru obrazů zobrazujících různé lidské tváře. Pro

zjednodušení je možné si eigenface představit jako soubor standardizovaných částí obličeje, odvozený od statistického modelu různých obrazů lidských obličejů.

Právě z těchto standardizovaných částí se skládá jakákoli jiná lidská tvář. Zajímavé je, že k aproximaci obličeje není nutné mít mnoho eigengface tváří. Je to rovněž způsobeno tím, že každá tvář je ukládána jako soubor hodnot, nikoli samostatný obraz. To umožňuje efektivní porovnávání snímků s mnoha již uloženými obličeji. [15]

Pro trénování tohoto algoritmu musíme získat vhodnou databázi obrazů lidských tváří. Tyto obrazy mají být pořízeny za stejných světelných podmínek, musí být normalizovány a musí mít sjednocené oči a ústa ve všech obrazech. Také snímky musí být navzorkovány na stejné rozlišení. Bez splnění těchto podmínek nejde dosáhnout akceptovatelné úspěšnosti identifikace.

Metoda Fisherfaces

Další metodou využívanou při rozpoznání obličeje je diskriminační analýza (Linear Discriminant Analysis, LDA), což je technika související s mnohorozměrnou statistickou analýzou. Filozofie je taková, že lze roztřídit objekty do tříd (kategorií) na základě rozhodovacího pravidla, které je sestaveno podle objektů stejného typu ve známé tréninkové množině. Tato technika pracuje s velkými rozdíly mezi jednotlivými třídami, ale pouze s minimálními odlišnostmi v rámci jedné třídy.

V oblasti výpočetní techniky je každý obličej reprezentován množstvím obrazových bodů, jejichž počet je příliš velký a jejich kombinace „příliš jedinečná“ pro snadné porovnání nebo určení identifikačních znaků. Diskriminační analýza je tedy (podobně jako dříve zmíněná analýza hlavních komponent) využita pro snížení objemu dat, se kterými pracujeme, a nalezení vhodných lineárních kombinací jednotlivých bodů. Tím zformujeme jakýsi vzor, ne nepodobný modelu vlastních tváří. Název je v tomto případě odvozen od autora použité metody a vzorům se říká Fisher faces, Fisherovy tváře.

V porovnání s metodou analýzy hlavních komponent není technika Fisherových tváří tolik závislá na shodném nasvětlení předloženého obličeje a rovněž se lépe vypořádává se změnami mimiky. [16]

Local Binary Patterns Histograms

V reálných aplikacích nemůžeme zaručit stejné dobré osvětlení všech obličejů. Občas není možné pořídit postačující počet snímků. V takových případech výsledky trénování a pak i identifikace budou příliš špatné. Metoda LBPH je pravě určená pro takovéto podmínky. Základní myšlenkou je sumarizace lokálních vlastností obrázku. Lokální binární šablona je popisem okolí pixelu, které jednoznačně určuje jeho tvar.

Algoritmus dokáže vypracovat model i podle jednoho snímku, a proto se používá tam, kde se vyžaduje zpracování obrazu v reálném čase. Vedle nízkých systémových nároků na vysoké rychlosti zpracování má velmi dobrou toleranci ke změnám osvětlení.

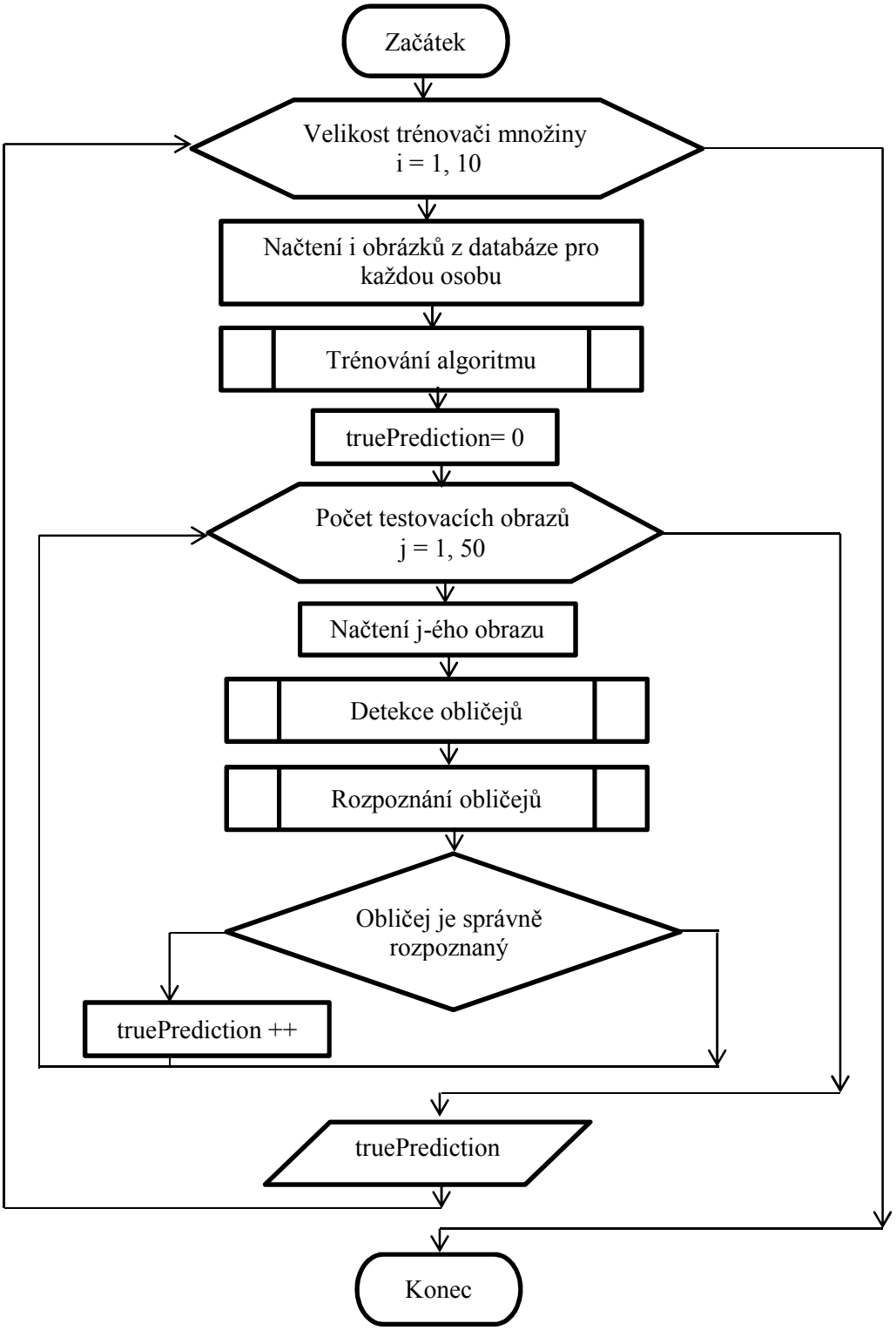
4.2 Srovnání úspěšnosti metod

Dalším krokem je implementace výše uvedených metod a výběr algoritmu s největší úspěšností rozpoznávání obličejů. Porovnání úspěšnosti metod se provádělo na počítači, neboť to umožňovalo automatizovat analýzu velkého počtu testovacích dat.

Snad nejdůležitějším aspektem identifikace tváří je databáze obrázků. Na špatně sestavené bázi i teoreticky dobrá metoda může selhat. Testovací databáze se skládá z fotek Angeliny Jolieové, Nicole Kidmanové a George Clooneyho. Pro každou osobu bylo vytvořeno deset snímku. Obrázky byly normalizovány: obličej se vyřezával, pak se otáčel tak, že přímka mezí centry očí byla horizontální. Nakonec se obrázek zmenšoval do velikosti 68x68 pixelů. Kromě toho, z důvodů snížení šumu ve snímcích se aplikovaly Gaussův a mediánový filtry. Ukázka toho jak vypadají snímky po těchto opravách je na obrázku 9.

Obrázek 9. Vzor z databáze obličejů

Za účelem porovnání metod byla vytvořena aplikace na počítači. Tím jsme zrychlili etapu výběru vhodného algoritmu. Vývojový diagram je uveden na obrázku 10. Aplikace byla napsána v jazyce Java s využitím knihovny JavaCV. Program postupně zvětšuje počet trénovačích vzorků od 1 až po 10 a počítá úspěšnost predikce. Na testovacím obrázku na začátku se pomocí haarových klasifikátorů hledá obličej. Pak se tento obličej normalizuje a vstupuje do funkce rozpoznávání.



Obrázek 10. Vývojový diagram

Inicializace algoritmu identifikace se provádí následující m způsobem:

```
FaceRecognizer faceRecognizer = null;
try{
    faceRecognizer = createFisherFaceRecognizer();

    //faceRecognizer = createEigenFaceRecognizer();

    //faceRecognizer = createLBPHFaceRecognizer();

}catch(Exception ex) { //catch possible exceptions
    System.out.println(ex.getMessage());
}
```

Každý algoritmus rozpoznávání je implementován do OpenCV jako samostatná třída implementující rozhrání FaceRecognizer.

Trénování proběhne při zavolání metody train():

```
faceRecognizer.train(images, labels);
```

Metoda přijímá na vstupu dvě pole: pole trénovačích obrázku a pole označení, které říká, jaké osobě patří tento obrázek.

Pak predikci dostaneme pomocí metody predict():

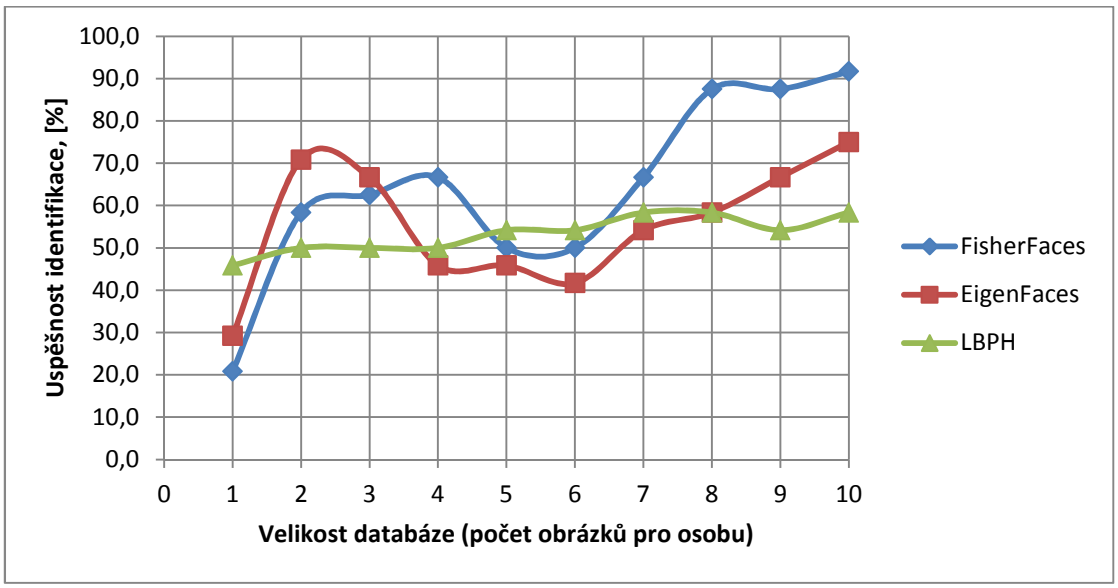
```
int[] predicted_label = {-1};
double[] predicted_confidence = {0.0};
faceRecognizer.predict(imgSmall, predicted_label, predicted_confidence);
```

Zde výstupem nebude pouze označení osoby, ale i euklidovská vzdálenost tohoto obličeje od nejbližšího z databáze. Pro nepřesnou predikci tato vzdálenost bude větší, a proto můžeme stanovit určitý práh, od kterého bude obličej považován za neznámý. Velikost prahu stanovíme později experimentálně podle výsledku rozpoznávání v reálné scéně.

Výsledky porovnávání zcela odpovídají teoretickému popisu algoritmu. Nejlepší přesnosti dosáhla metoda Fisherfaces. Což je pochopitelné, protože ani obrázky v databázi, ani testovací obrázky neměly stejné světelné podmínky. Algoritmus lokálních binárních histogramů je relativně úspěšný na malém počtu trénovacích dat, ale nedokázal zlepšit přesnost predikce s růstem databáze. Výsledky lze vidět v tabulce 4.1, grafické znázornění je na grafu na obrázku 11.

Tab. 2. Výsledky predikce různými metodami v závislosti na velikosti databáze

Počet trénovacích snímků, [-]	Fisherfaces, [%]	Eigenfaces, [%]	LBPH, [%]
1	20,8	29,2	45,8
2	58,3	70,8	50,0
3	62,5	66,7	50,0
4	66,7	45,8	50,0
5	50,0	45,8	54,2
6	50,0	41,7	54,2
7	66,7	54,2	58,3
8	87,5	58,3	58,3
9	87,5	66,7	54,2
10	91,7	75,0	58,3



Obrázek 11. Graf výsledků predikce různými metodami v závislosti na velikosti databáze

Poněvadž rychlost výpočtu není pro naši aplikaci kritická, můžeme si zvolit nejúspěšnější algoritmus rozpoznávání obličejů. To jest Fisherfaces. Ten se bude používat v rámci vyvinuté mobilní aplikace. I přes možné snížení výkonu, respektive klesání hodnoty FPS (frames per second), aplikace bude schopná detekovat a identifikovat tváře v reálném čase.

5 VÝVOJ APLIKACÍ

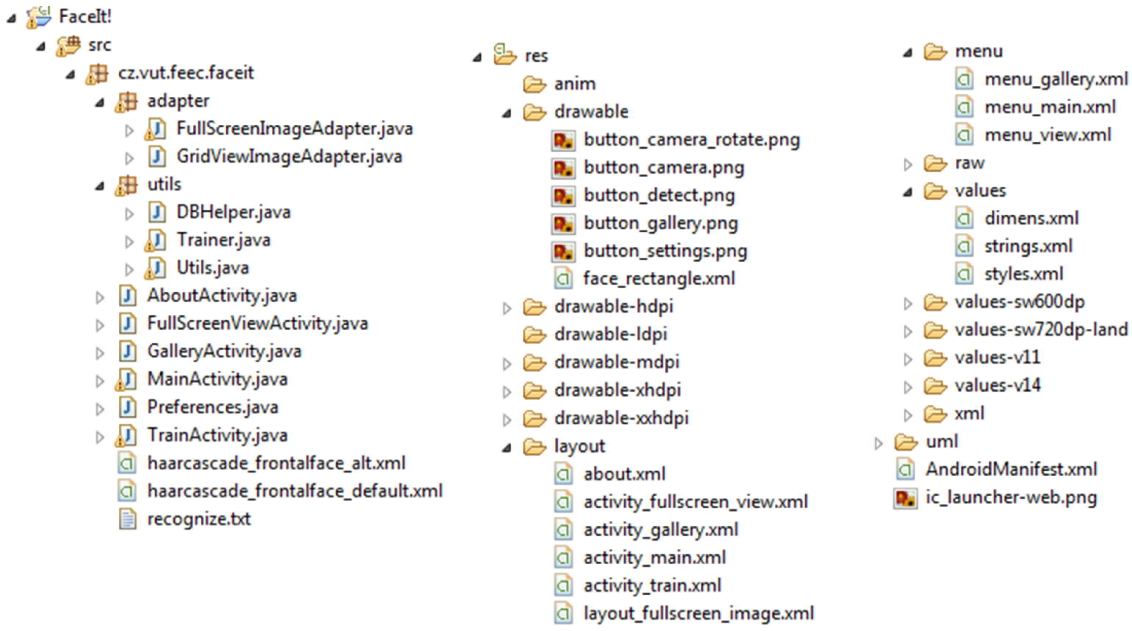
Vzhledem k tomu, že oficiálně podporované vývojové prostředí pro aplikace Android je Eclipse, většina vývojářů používá pravě Eclipse. Nicméně nikdo nebrání vyvíjet i v jiném prostředí. Dokonce stačí textový editor a ke kompilaci příkazový řádek.

Aby bylo možné vyvíjet aplikace je potřeba nainstalovat Java Development Kit (JDK), Eclipse Android Development Tools (ADT) plugin, Android Software Development Kit (SDK). V této diplomové práci jsem používal speciální ADT Bundle, který je doporučen pro nové android developery v [4]. Obsahuje všechno potřebné pro vývoj za výjimkou JDK. Samotný postup vývoje a příklady kódu jsou popsány dále v této kapitole.

Jelikož se jedná o hotový software, má tento produkt mít vlastní unikátní název a ikonu. Název by měl naznačovat hlavní funkcionalitu aplikace, být stručným a lehce pamatovatelným. V tomto projektu se zabývám detekci a rozpoznáním obličeje, tak vznikl název FaceIt!. Ikonu lze vidět na obrázku 12. Ona byla vybrána z online kolekce bezplatných clipartu [3].

Obrázek 12. Ikona aplikace[3]

Dále budeme probírat jednotlivé části aplikace, a proto na začátku uvedeme strukturu projektu v Eclipce. Ve stromě jsou vyjmenovány zdrojové kódy (adresář src/), zdroje aplikace (adresář res/) a soubor AndroidManifest.xml. Viz obrázek 13.



Obrázek 13. Struktura projektu v Eclipse

5.1 Manifest

Manifest je základním souborem každé aplikace. Zde se uvádí veškerá informace o projektu, aj. Název, cílová a minimálně nutná pro spuštění verze API, verze aplikace, seznam povolení přístupu k systémovým zdrojům. Manifest je místem, kde jsou popsány jednotlivé aktivity, ze kterých je složená aplikace.

Soubor AndroidManifest.xml musí být umístěn v kořenovém adresáři. Představuje běžný XML dokument. Výňatek z manifestu je uveden níže:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="cz.vut.feec.faceit"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="cz.vut.feec.faceit.MainActivity"
            android:screenOrientation="portrait"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Na obrázku 3 uvádím pouze část zdrojového kódu, kompletní soubor viz v příloze 1. Je důležité zmínit význam některých sekcí a to:

- `uses-permission` popisuje jaké systémové zdroje budou použité, respektive kamera a přístup k vnějšímu úložišti (paměťové kartě).
- Prvek `application` je jedním ze základních elementů manifestu, který obsahuje popis komponent aplikace. V každé aplikaci musí být pouze jeden element `application`.
- `Activity` deklaruje aktivitu čili jednotlivou obrazovku aplikace. Na obrázku 3. je úvodní aktivita aplikace MainActivity. Pro tento prvek může existovat filtr záměrů (intent-filter), určující aktivitu, která se má provést. Speciálním typem je akce `android:name=«android.intent.action.MAIN»`, umožňující označit aktivitu, kterou může spustit uživatel například ze seznamu aplikací. Element `category` (`android:name=«android.intent.category.LAUNCHER»`) ukazuje na to, že aplikace bude přidána do seznamu všech android aplikací zařízení.
- Hodnoty atributů začínající se symbolem zavináč @ odkazují na zdroje, které obsahují aktuální hodnoty. Například lokalizace nebo grafické elementy. Víc o zdrojích viz kapitola 3.4

5.2 Uživatelské rozhraní

Aktivita poskytuje uživatelské rozhraní aplikace. Aplikace může obsahovat více aktivit, mezi kterými přechází, a jedna aktivita většinou znamená jednu obrazovku s konkrétním účelem. Jak jíž bylo zmíněno v kapitole 1.1 jednou ze vlastností platformy Android je rozdělení GUI a logiky aplikace. Všechny aktivity definované v manifestu mají jako název třídu, ve které se zpracovávají akce. Stačí zavolat metodu `setContentView`, která nastaví xml maketu pro aktuální obrazovku. Dále budeme makety jmenovat Layout na rozdíl od aktivit obrazovek.

Hlavními součástmi aplikace jsou 5 aktivit. Každá z nich bude popsána v jednotlivých částech této podkapitoly.

5.2.1 Úvodní obrazovka aplikací

Pozadí úvodní obrazovky programu je tvořeno náhledem kamery. Jsou zde 5 tlačítek: tlačítko k zachycení snímku, odkaz na galerii a nastavení, tlačítko, které zapíná detekcí obličejů, a také odkaz na aktivitu přidávání obličejů do databáze. V layoutu se používá frameové rozložení. Jedná se o nejjednodušší a zároveň velmi efektivní z pohledu designéru typ rozmístění prvků. Ve většině je to prázdný prostor na obrazovce, který lze naplnit dceřinými objekty. Layout sestavíme v kódu z toho důvodu omezených možnosti platformy Android. Tedy musíme mít možnost kreslit na obrazovce. Při xml rozložení pro každý prvek, respektive View, bude vytvořeno vlastní vlákno a kreslení nebude vidět.

```
// Create our Preview view and set it as the content of our activity.
try {
    layout = new FrameLayout(this);
    faceView = new FaceView(this);
    mPreview = new Preview(this, faceView);
    layout.addView(mPreview);
    layout.addView(faceView);

    drawImageButtons();

    setContentView(layout);
} catch (IOException e) {
    Log.e(TAG, e.getMessage());
    new AlertDialog.Builder(this).setMessage(e.getMessage()).create().show();
}
```

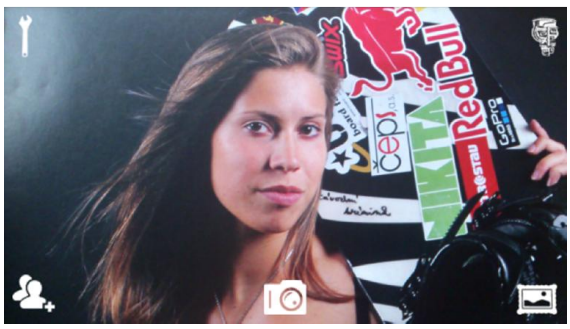
Objekty FrameLayout, FaceView a Preview jsou definovány jako vnitřní třídy objektu MainActivity.

Layout je roztažen na celou obrazovku. Každý potomek je umístěn v horním levém rohu obrazovky. Oblast SurfaceView je určená pro zobrazení náhledu z kamery. Roztažení je nastavěno tak, aby se zachovaly poměry stran náhledu.

Aby bylo vidět všechna tlačítka, pro každé z nich byl nastaven atribut layout_gravity, který ukazuje roh (respektive bod) kam prvek má být umístěn. Příklad generace tlačítka je vidět tady:

```
//capture btn
 ImageView imgCapture = new ImageView(this);
 imgCapture.setImageResource(R.drawable.button_camera);
 LayoutParams captureParams = new FrameLayout.LayoutParams(ViewGroup.LayoutParams.WRAP_CONTENT,
         ViewGroup.LayoutParams.WRAP_CONTENT,
         Gravity.BOTTOM|Gravity.CENTER_HORIZONTAL);
 captureParams.setMargins(0, 0, 0, 20);
 imgCapture.setLayoutParams(captureParams);
 imgCapture.setOnClickListener(mPreview);
 layout.addView(imgCapture);
```

Ukázku GUI lze vidět na obrázku 15.



Obrázek 14. Úvodní obrazovka aplikace

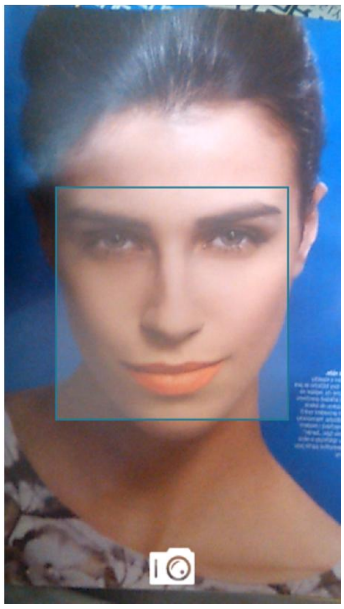
Taková maketa bude vypadat naprosto stejně na různě velikých obrazovkách. Jak je také vidět na úvodní obrazovce není standardní hlavička z důvodu úspory místa.

5.2.2 Aktivita preferences

Také do aplikace byla implementována možnost zapínat a vypínat různé detektory a samotné rozpoznávání. Pro tyto účely byla vytvořena aktivita Preferences. Její třída je potomkem speciálního typu aktivit `android.preference.PreferenceActivity`. Xml soubor nastavení `settings.xml` popisuje parametry, které se budou ukládat do systému a jejich strukturu.

5.2.3 Trénovací aktivita

V aplikaci je realizována možnost rozšíření databáze známých tváři. Za tuto činnost je odpovědná aktivita trénovací. Její obrazovka je uvedená na obrázku 16.



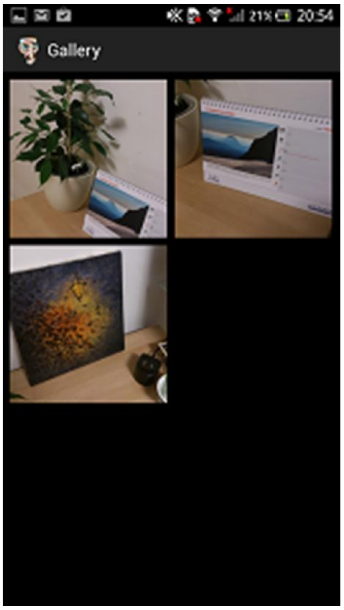
Obrázek 15. Obrazovka trénovací aktivity

Při vstupu do této uživatel musí zadat jméno nové osoby. Pak udělat pomocí tlačítka sérii z deseti snímku. Jak již bylo zmíněno v kapitole 4, všechny pořízené obrázky musí být normalizovány. Proto na úvodní obrazovce této aktivity je nakreslen čtverec, do kterého má být umístěn obličej. Aplikace obnoví algoritmus identifikace, upozorní uživatele a vrátí se na úvodní obrazovku.

5.2.4 Aktivita Galerie

Představuje sebou tag `GridView`, sloužící jako kontejner pro adaptéry. Adaptéry jsou třídy, jejichž instance se používá jako prostředník mezi kolekcí položek a UI komponentou reprezentující seznam. Tímto zajistíme vykreslování každého náhledů zvlášť. Následně v kódu bude pomocí adaptéru `GridViewImageAdapter` naplněn

objekty typu `Bitmap` z balíčku `android.graphics`. Sceenshot obrazovky je na obrázku 17. Po kliku do obrázku se otevře aktivita s jeho náhledem.



Obrázek 16. Obrazovka Galerie

Obrázek 17. Obrazovka náhled

5.2.5 Náhled obrázků

Slouží pro zobrazení snímku na celou obrazovku. Jedná se o LinearLayout, který má v sobě tag ViewPager z Adroid SDK. Zde se používá adaptér `FullScreenImageAdapter` potomek `android.support.v4.view.PagerAdapter`. Adapter se pak naplňuje pomocí speciálního layoutu `layout_fullscreen_image.xml` pro zobrazení snímku na celou velikost obrazovky. Pohybem prstu doleva či doprava lze skrolovat seznam obrázku. Sceenshot obrazovky je na obrázku 18.

5.2.6 Menu

Každá obrazovka programu má vlastní menu, které se zobrazí po stisknutí hardwarového tlačítka Menu. Obsah nabídek se trochu liší. Na úvodní jsou to položky nastavení, resetování databáze, trénování databáze na již pořízených fotkách a informace o aplikace. V galerii jsou k dispozici nastavení a volba Smazat všechno. U náhledu je také možné smazat snímek a popřípadě sdělit ho pomocí všech dostupných na tomto zařízení aplikací. Například odeslat obrázek mailem nebo sdělit ho na Facebook.

5.3 Detekce a rozpoznáváni obličejů ve video streamu

Aplikace algoritmů detekce či rozpoznání obličejů pro statické obrázky nenese nic nového, je to pouhé opakování existujících řešení. V podstatě rozdíl mezí aplikaci podobných algoritmu pro platformu Android nebo jakoukoliv jinou platformu či jazyk není patrný. Aby tento projekt přinášel přidanou hodnotu a byl inovativní, rozhodl jsem implementovat metody detekce a identifikace tváři ve video streamu.

Jak již bylo řečeno výše, pro zobrazení náhledu kamery se použil speciální prvek rozložení obrazovky SurfaceView. Tento objekt zobrazuje upravený signál z kamery telefonu. Abychom byli schopní tento signál zpracovávat, musíme upravit třídu FaceView tak, aby implementovala rozhrání Camera.PreviewCallback. Tím pádem máme k dispozici novou metodu

```
public void onPreviewFrame(final byte[] data, final Camera camera)
```

Argumenty jsou objekt typu Camera, reprezentující aktuální kameru a pole bytů. Pravě tento parametr bude pro nás důležitým.

Jako každá digitální kamera, kamera android telefonu snímá scénu s určitou periodou vzorkování. To jest vývojář pracuje s číslicovým výstupem zařízení nikoliv s analogovým. Video potok tedy se rozbije na jednotlivé snímky a vzorkovací, respektive zobrazovací frekvence, se jmenuje FPS (frames per second). Při každém obnovení náhledu kamery platforma Android volá tuto metodu.

Obsahem výše zmíněného parametru jsou bity snímku ve formátu YUV. Je to barevný model používaný v televizním vysílání v normě PAL i HDTV. Model k popisu barvy používá tříprvkový vektor [Y,U,V], kde Y' je jasová složka a U a V jsou barevné složky. U je také někdy označováno jako B-Y a V odpovídá R-Y. Barevné složky se používají v rozsahu od -0.5 do +0.5, jasová složka má rozsah od 0 do 1. Výhodou YUV je oddělení jasové složky, kterou člověk přesněji vnímá. Pak je možné vyhradit pro chromatickou složku menší šířku přenosového pásma.[17]

Z pohledu knihovny OpenCV není potřeba měnit barevný model. Knihovna dokáže sama detekovat formát. Pro účely detekce vytvoříme z pole podvzorkováný šedotónový obrázek.

```
IplImage grayImage = IplImage.create(width/f, height/f, IPL_DEPTH_8U, 1);
```

Rozdělíme obsah pole na jednotlivé řádky snímku a zapíšeme je ve smyčce do bufferu obrázku:

```
ByteBuffer imageBuffer = grayImage.getByteBuffer();
for (int y = 0; y < imageHeight; y++) {
    int dataLine = y*dataStride;
    int imageLine = y*imageStride;
    for (int x = 0; x < imageWidth; x++) {
        imageBuffer.put(imageLine + x, data[dataLine + f*x]);
    }
}
```

Pak je možné použit již probranou v kapitole 3 metodu detekce pomocí Haarových příznaků.

Naopak knihovna FaceDetector vyžaduje úpravy barevného schématu. Musíme konvertovat snímek do bitmapu s parametrem RGB565. Následující kód je příkladem konverze:

```
YuvImage image = new YuvImage(data, format, width, height, null);
ByteArrayOutputStream out = new ByteArrayOutputStream();

 image.compressToJpeg(
            new Rect(0, 0, image.getWidth(), image.getHeight()), 90,
            out);
byte[] imageBytes = out.toByteArray();

BitmapFactory.Options options = new BitmapFactory.Options();
//provedeme downsampling vstupniho obrazu
options.inSampleSize = SUBSAMPLING_FACTOR;
Bitmap bmp = BitmapFactory.decodeByteArray(imageBytes, 0, imageBytes.length, options);

// je dulezite zkonvertovat obrazek do bitmapu s parametrem RGB_565
// jinak decoder nebude schopen zpracovat vstupni data
Bitmap mFaceBitmap = bmp.copy(Bitmap.Config.RGB_565, true);
```

Ve výsledném bitmapu lze detekovat obličeje.

Po té, co byly detekovány obličeje, následuje krok zobrazení průběžných výsledku pro uživatele. Kolem každé tváře se kreslí obdélník příslušné barvy:

- čevená (OpenCV),
- zelená (FaceDetectionListener),
- modrá (FaceDetector),
- žlutá (známý obličej)

To, jak vypadají výsledky detekce lze vidět na obrázku XXX v kapitole 3. Kreslení se provádí v metodě onDraw(Canvas canvas), kde parametr canvas je plátno poskytující možností zobrazení geometrických primitiv na náhledu kamery.

Před samotným rozpoznáním obličej se musí správně ořezat a zmenšit, aby plnil podmínky knihovny OpenCV. Podle známých souřadnic levého horního bodu obličeje a jeho šířky a výšky vyřežeme z celého šedotónového snímku čtverec s tváři.

```
CvRect cut = new CvRect(centerX - Math.round(w*0.35f),
        centerY - Math.round(h*0.3f),
        Math.round(w*0.7f),
        Math.round(h*0.7f));

cvSetImageROI(grayImage, cut);
IplImage cropped = cvCreateImage(cvGetSize(grayImage),
                            grayImage.depth(),
                            grayImage.nChannels());
cvCopy(grayImage, cropped);
cvResetImageROI(grayImage);
```

Pak po zmenšení můžeme aplikovat algoritmus identifikace.

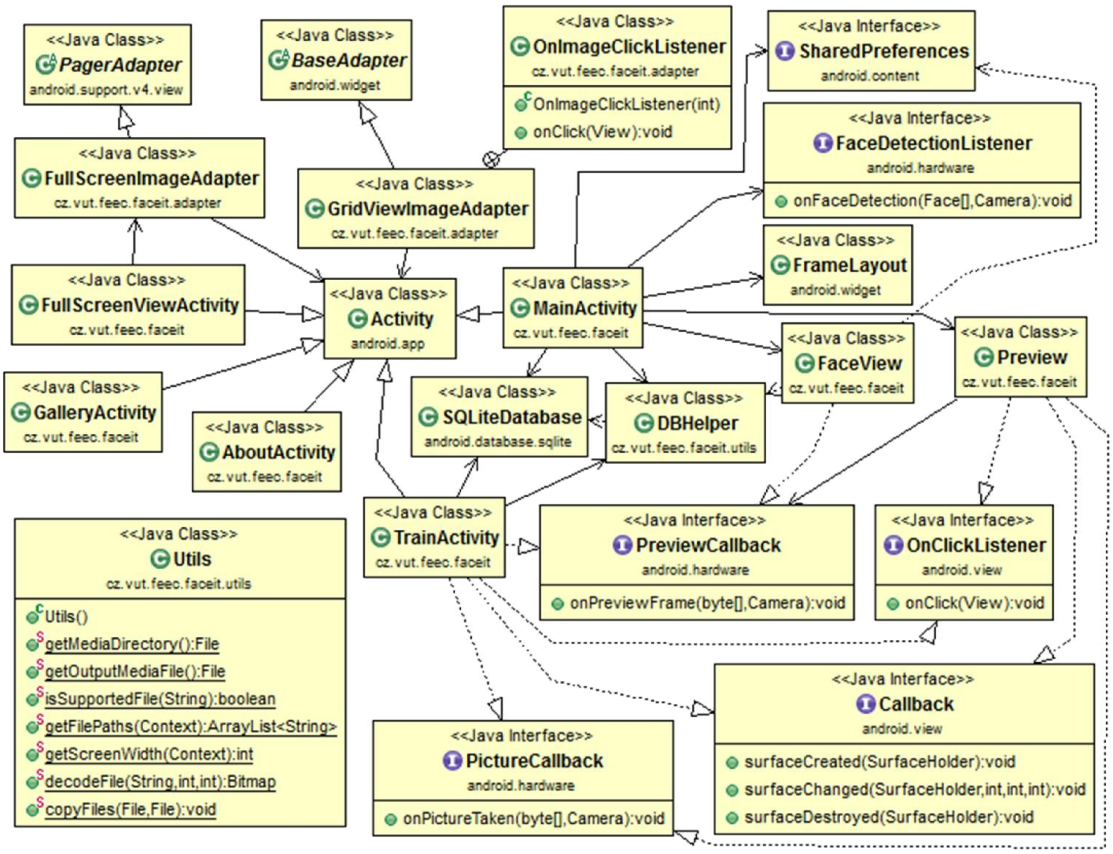
```
IplImage img = IplImage.create(Utils.SAMPLE_SIZE,
                               Utils.SAMPLE_SIZE,
                               cropped.depth(),
                               cropped.nChannels());
cvResize(cropped, img, CV_INTER_CUBIC);
```

Lze říct, že stabilita detekce a rozpoznání obličejů ve video potoku je o něco horší, než při práci se statickými obrázky, poněvadž nemůžeme zaručit stabilní polohu telefotu. kamera běžného telefonu těžko snímá rychlé děje ve scéně. Každý pohyb jednak rozostří snímek, jednak vyvolá funkci autoFocus, která změní ohnisko čočky kamery. Na kvalitu algoritmu působí také digitální šum. I když ho snažíme potlačit mediánovým filtrem, jeho vliv je stále patrný. Akceptovatelných výsledků dosáhneme až po ustálení pohybu ve snímané scéně.

5.4 Vnitřní struktura

Standardem programování v jazyce Java je rozdělení tříd do tzv. balíky (packages), jejichž jména jsou tvořena hierarchicky identifikátory oddělenými tečkami. Každý balíček představuje logickou část aplikace a shromažďuje významově podobné třídy. V tomto projektu budeme používat základní balík `cz.vut.feec.faceit`, ve kterém jsou aktivity. Balík `cz.vut.feec.faceit.adapter`, jak je zjevné z názvu, bude obsahovat adaptéry. Posledním balíčkem bude `cz.vut.feec.faceit.utils`, kde budou umístěny pomocné třídy.

Diagram tříd přehledně reprezentuje strukturu aplikace. Vstupním bodem, který je definován i v manifestu, je třída MainActivity v kořenovém balíčku. Viz obrázek 19.



Obrázek 18. UML diagram tříd

Objekt `MainActivity` obsahuje tři vnitřní třídy:

1. `FrameLayout` (definuje rozložení aktivity)
2. `FaceView` (implementuje algoritmy detekce a rozpoznávání a zároveň zobrazuje detekované obličeje)
3. `Preview` (obstará inicializaci kamery a pořízení snímků)

Důležitou metodou tady je setCameraParams() v objektu Preview, kde se nastavují parametry kamery. Tyto modifikace se provádí proto, aby na preview nebyla žádná zkreslení v geometrii, aby náhled byl natočen správně. Hledání správných hodnot těchto parametrů zabralo přibližně třetinu času, co jsem strávil nad projektem. Ačkoli platforma Android je velice standardizována, každý výrobce zařízení má vlastní pohled na to, jak by měl být vyřešen hardware. U některých chytrých telefonů, zejména starších generací, naprosto nešlo otočit kameru.

Třída Utils obsahuje několik pomocných statických metod, jež se volají z různých částí projektu. Například metoda decodeFile() mění velikost obrázku tak, aby se zachoval poměr stran a obrázek se hodil do galerie nebo náhledu. Nebo metoda getScreenWidth(), která zjišťuje šířku obrazovky v pixelech.

Při rozpoznávání obličejů je třeba říct uživateli jméno nalezené osoby, poněvadž pouhé pořadové číslo osoby v bázi nenese pro něj žádnou informaci. Proto bylo implementováno uložení řetězců jmen do SQL databázi. Není nutné používat externí knihovnu. Platforma Android poskytuje speciální pomocnou třídu SQLiteOpenHelper, která umožňuje přístup k databázi SQLite. Objekt DBHelper realizuje základní metody a obsahuje popis struktury DB.

Kompletní zdrojové kódy jsou v příloze 2.

5.5 Zdroje

Kromě kódu patří k aplikaci další zdroje (resources), jako jsou obrázky, definice vzhledu nebo nabídek menu, atd. Ty jsou umístěny v adresáři res/. Ve zdrojovém kódu lze ke všem přistupovat pomocí identifikátorů, jež jsou v androidu statickými vlastnostmi třídy R. V XML se pro referenci používá syntaxe se znakem zavináč, typem a názvem zdroje.

Podadresáře mají vlastní strukturu a umožňují ukládat odlišné objekty do různých složek. Například obrázky v tomto projektu jsou uložený do drawable/, vzhledy uživatelského prostředí do layout/, nabídky dostupné v aplikaci leží v menu/, ve složce values/ jsou definice řetězců, polí hodnot, barev, rozměrů atd.

6 ZÁVĚR

Výsledkem této práce je aplikace pro zařízení Android, která umožňuje uživateli detekovat obličeje ve video streamu z integrované kamery/fotoaparátu a rozpoznávat osoby, jejichž fotografie byly použity při trénování aplikace. Aplikace také může fungovat jako obyčejný fotoaparát: pomocí ní se dá pořídit fotky a pak je zhlédnout v galerii.

V programu je realizována možnost rozšíření databáze známých tváří, a to prostřednictvím frontální kamery. V tomto režimu je nutné udělat 10 snímků. Po uložení snímků do databáze se algoritmus trénování spustí automaticky. V souvislosti s problematikou detekce objektů ve video streamu, která je popsaná v páté kapitole, snímky by měly být normalizovány, mít dobré osvětlení a relativně malou úroveň šumu, kterou nám například nezaručí kamera levného mobilního telefonu.

Kromě požadavků na kvalitu kamery je také požadavek na výpočetní výkon mobilu. Kvůli velké výpočetní náročnosti používaných algoritmů minimální verze platformy Android, kterou by zařízení mělo podporovat, má být 4.0. U starších generací mobilních telefonů při spouštění algoritmu detekce obličejů hodnota FPS (frames per second) klesá do nepřijatelné úrovni.

V rámci diplomové práce byly prozkoumány standardní prostředky platformy Android pro detekci obličejů (FaceDetector a FaceDetectionListener) a také knihovny JJIL, OpenIMAJ, OpenCV. V režimu detekce obličejů vyvinutá aplikace umožňuje použit pouze tři metody: knihovnu OpenCV, FaceDetector a FaceDetectionListener, neboť knihovny JJIL a OpenIMAJ se prokázaly jako nestabilní a velice špatně realizované. Uživatel si může zvolit, jakými metodami bude detekovat obličeje, a to může byt jedna, dvě nebo všechny tři metody.

Po porovnání výsledků detekce tváří v různých scénách jako nejúspěšnější se projevila knihovna OpenCV, která pak byla použita ve výsledné aplikaci jako základní.

Další funkcí aplikace je rozpoznání obličejů, jež se dělá prostřednictvím knihovny OpenCV. Byly vyzkoušeny tři algoritmy identifikace (FisherFaces, EigenFaces a Local Binary Patterns Histograms) a byla stanovena nejlepší metoda, a to FisherFaces.

Na úspěšnost rozpoznání obličejů rozhodující vliv měla kvalita databáze, co bylo experimentálně ověřeno na několika bázích. Pro lepší výsledky identifikace tváří byla vytvořena vlastní databáze, která se skládá z obličejů významných lidí a autora této práce. Kvalita rozpoznávání na této bázi je poměrně dobrá.

Tento projekt byl pro mě velkým přínosem. Rozšířil jsem své znalosti v oblasti vývoje aplikací pro mobilní zařízení, seznámil jsem se s principy fungování různých hardwarů, možnostmi jejich nastavení a ovládání. Poznal jsem několik nových koncepcí a návrhových vzorů.

Výsledky této práce můžou být použity například ve vývoji mobilních biometrických aplikací pro policii jako způsob okamžité identifikace osob bez průkazu totožnosti.Nebo v Google Glass pro hledání známých či důležitých osob v davu nebo na společenské akci.

LITERATURA

[1] Android (operační systém) [online], poslední aktualizace 3. 1. 2014 v 10:19 [cit. 3.1.2014], Wikipedie. Dostupné na URL: http://cs.wikipedia.org/wiki/Android_(operační_systém)

[2] Архитектура Android-приложений. Часть III — основные части приложения [online], poslední aktualizace 1.4.2012 v 19:03 [cit. 3.1.2014], Habrahabr – Otevřený společný blog o IT, počítačové vědě a spojených tématech. Dostupné na URL: http://habrahabr.ru/post/141201/

[3] Abstract Face [online], poslední aktualizace 24.3.2012 v 19:29 [cit. 3.1.2014], Free clipart collection. Dostupné na URL: https://openclipart.org/detail/169106/abstract-face-by-viscious-speed

[4] Android Developer Documentation [online] [cit. 3.1.2014]. Dostupné na URL: http://developer.android.com/

[5] FaceDetector.Face [online], poslední aktualizace 13.05.2014 [cit. 10.4.2014], Android API Reference. Dostupné na URL: http://developer.android.com/reference/android/media/FaceDetector.Face.html

[6] Camera [online], poslední aktualizace 13.05.2014 [cit. 10.4.2014], Android API Reference. Dostupné na URL: http://developer.android.com/reference/android/hardware/Camera.html

[7] jjil - jon's java imaging library, for mobile image processing [online], poslední aktualizace 15.5.2008 [cit. 10.4.2014], google project hosting. Dostupné na URL: http://code.google.com/p/jjil.

[8] Hare J., Samangooei S. OpenIMAJ on Android [online], poslední aktualizace 18.11.2013 [cit. 10.4.2014]. Dostupné na URL: http://sourceforge.net/p/openimaj/wiki/OpenIMAJ%20on%20Android

[9] Michael J. Jones Paul Viola. Robust real-time face detection[online], 2004. Dostupné na URL: http://www.vision.caltech.edu/html-files/EE148-2005-Spring/pprs/viola04ijcv.pdf

[10] Java interface to OpenCV and more [online]. [cit. 10.4.2014], google project hosting. Dostupné na URL: https://code.google.com/p/javacv/

[11] Jiří Přinosil, Martin Krolikowski. Využití detektoru Viola-Jones pro lokalizaci obličeje a očí v barevných obrazech [online]. 21.08.2008 [cit. 30.04.2014]. Dostupné na URL: http://www.elektrorevue.cz/cz/clanky/zpracovani-signalu/0/vyuziti-detektoru-viola-jones-pro-lokalizaci-obliceje-a-oci-v-barevnych-obrazech/

[12] T. Malach, P. Bambuch, J. Malach Detekce obličeje v obraze s využitím prostředí MATLAB [online]. [cit. 30.04.2014]. Dostupné na URL: http://dsp.vscht.cz/konference_matlab/MATLAB11/prispevky/078_malach.pdf

[13] Chiara Turati, Viola Macchi Cassia, F. S., and Leo, I. Newborns face recognition: Role of inner and outer facial features. Child Development 77, 2 (2006), 297–311.

[14] Brunelli, R., Poggio, T. Face Recognition through Geometrical Features. European Conference on Computer Vision (ECCV) 1992, S. 792–800.

[15]Petr Vychodil, Michal Vymazal. Zpracování obrazových informací získaných z policejních radarů pomocí systému FaDeR [online], 31.08.2010 [cit. 30.04.2014]. Dostupné na URL: http://www.elektrorevue.cz/cz/clanky/zpracovani-signalu/20/zpracovani-obrazovych-informaci-ziskanych-se-znimku-policejnich-radaru-pomoci-systemu-fader/

[16] Daniel Karchňák. ROZPOZNÁVÁNÍ OBLIČEJŮ. Časopis Computer, říjen 2011, s. 72-75

[17] YUV [online] , poslední aktualizace 14. 5. 2014 v 13:25 [cit. 30.04.2014], Wikipedie. Dostupné na URL: http://cs.wikipedia.org/wiki/YUV

SEZNAM PŘÍLOH

PŘÍLOHA 1. SOUBOR ANDROIDMANIFEST.XML

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
 package="cz.vut.feec.faceit"
 android:versionCode="1"
 android:versionName="1.0" >

 <uses-sdk
 android:minSdkVersion="13"
 android:targetSdkVersion="13" />

 <uses-permission android:name="android.permission.CAMERA" />
 <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

 <application
 android:allowBackup="true"
 android:icon="@drawable/ic_launcher"
 android:label="@string/app_name"
 android:theme="@style/AppTheme" >
 <activity
 android:name="cz.vut.feec.faceit.MainActivity"
 android:screenOrientation="portrait"
 android:label="@string/app_name" >
 <intent-filter>
 <action android:name="android.intent.action.MAIN" />
 <category android:name="android.intent.category.LAUNCHER" />
 </intent-filter>
 </activity>
 <activity
 android:name="cz.vut.feec.faceit.GalleryActivity"
 android:label="@string/title_activity_gallery" >
 </activity>
 <activity
 android:name="cz.vut.feec.faceit.FullScreenViewActivity"
 android:theme="@android:style/Theme.Holo.NoActionBar">
 </activity>
 <activity
 android:name="cz.vut.feec.faceit.AboutActivity"
 android:label="@string/title_about" >
 </activity>
 </application>

</manifest>
```

PŘÍLOHA 2. ZDROJOVÉ KÓDY A APLIKACE

Součástí elektronického archivu práce jsou zdrojové kódy a výsledná aplikace. Projekt s kompletním kódem je umístěn v komprimovaném souboru faceit-project.zip, aplikace připravená k instalaci do zařízení je uložena pod názvem faceit.apk, testovací aplikace v komprimovaném souboru test-recognition.zip