VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

KAMERA S VÝMĚNOU OBLIČEJE PRO ANDROID

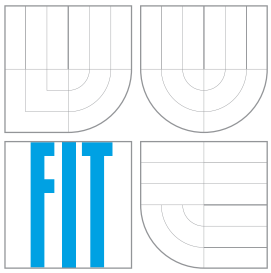
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PETR ŠKORŇOK

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

KAMERA S VÝMĚNOU OBLIČEJE PRO ANDROID

FACE-SWAP CAMERA FOR ANDROID

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PETR ŠKORŇOK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ISTVÁN SZENTANDRÁSI

BRNO 2014

Abstrakt

Cílem této práce je prozkoumat existující možnosti pro detekci obličejů v obraze na mobilních zařízeních s operačním systémem Android a na základě zjištění vytvořit aplikaci podporující výměnu obličeje se vstupem z kamery. Návrh aplikace se drží snahy dosáhnout co možná největší rychlosti při zpracování jednotlivých snímků a výsledné řešení je otestováno z pohledu uživatele i programové funkčnosti a rychlosti.

Abstract

Aim of this thesis is to explore existing possibilities of face detection on mobile devices supporting operating system Android and based on these findings create a face swap application with the input from the camera. The overall application is designed with an effort to reach the highest speed possible while processing the images. Implementation is tested from the view of the user and also from the point of program's speed and functionality.

Klíčová slova

Android, Detekce tváře, Haarovy příznaky, OpenCV, AdaBoost, Integrální obraz, Kalman filter, Maďarská metoda, Camshift, Meanshift, Kamera, Výměna tváře.

Keywords

Android, Face detection, Haar features, OpenCV, AdaBoost, Integral image, Kalman filter, Hungarian method, Camshift, Meanshift, Camera, Faceswap.

Citace

Petr Škorňok: Kamera s výměnou obličeje pro Android, bakalářská práce, Brno, FIT VUT v Brně, 2014

Kamera s výměnou obličeje pro Android

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Istvána Szentandrásiho

.....
Petr Škorňok
21. května 2014

Poděkování

Děkuji panu Ing. Istvánu Szentandrásimu za kvalitní odborné vedení a dostupnost při konzultacích. Dále bych chtěl poděkovat všem lidem, kteří byli ochotni se podílet na testování aplikace a Kateřině Žmolíkové za poskytnutí fotografií z dětství jako podkladového materiálu pro vysvětlení některých algoritmů.

Obsah

Kapitola 1

Úvod

Lidské oko je od přírody velmi citlivé na obličeje. Dokážeme rozpoznat známou tvář ve změti tisíců lidí, se kterými se běžně potkáme. Obličejové rysy, barva kůže, vrásky, mimika, pohlaví, všechny tyto prvky a mnohé další nám pomáhali a pomáhají rozpoznat v přírodě přítele od potenciální hrozby. Možná právě z toho důvodu působí kontrast mezi obličejem staré ženy usazeném na těle batolete tak komicky a záměna obličeje se stala v poslední době velmi rozšířeným způsobem úpravy fotografií za účelem pobavení.

Pro stolní počítače existují aplikace, které tuto záměnu dokáží dělat automaticky v reálném čase, pro mobilní zařízení jsou zase dostupná řešení, kdy je výměna obličejů možná z již vytvořené fotografie.

Tato práce si klade za cíl vytvořit na mobilní zařízení podporující operační systém Android aplikaci, která je schopna v čase blízkém reálnému času, zaměňovat obličeje na snímcích z kamery. Z pohledu uživatele je tedy jejím hlavním cílem pobavit, zatímco na pozadí stojí práce využívající mnoha technologií pro detekci a sledování obličeje.

Nejprve se text zabývá přehledem již existujících algoritmů a postupů při nalezení obličeje v obraze, dále jsou popsány způsoby pro sledování nalezené tváře, následně je představen návrh řešení pro mobilní zařízení s platformou Android, ten je implementován a na závěr je celá aplikace otestována a vyhodnocena.

Kapitola 2

Metody detekce obličeje

Ze začátku je zapotřebí vymezit dva často zaměňované pojmy, které spolu úzce souvisí - detekci a rozpoznávání obličeje. Yang, Kriegman a Ahuja [18] definovali právě první z těchto pojmů tak, že pokud je dán libovolný obraz, cílem detekce obličeje je určit, zda se v něm vyskytují či nevyskytují jakékoliv tváře a pokud ano, vrátit umístění a velikost každého nalezeného obličeje.

Rozpoznávání pak slouží k identifikaci daného obličeje a jeho přiřazení ke specifické osobě. Ze vstupního vzorku několika fotografií jsou analyzovány a uloženy obličejové charakteristiky dané osoby, které jsou poté vyhledávány v dalších fotografiích. Tato technika našla své uplatnění například v automatickém označování lidí v rámci sociální sítě nebo vyhledávání podezřelých osob. Dále se s touto problematikou lze seznámit například v [1]. Tato práce se dále bude zabývat pouze detekcí obličeje.

2.1 Chyby při detekci

Proces samotného nalezení obličeje v obraze není triviální záležitostí. Osoba může být natočena vzhledem k pozorovateli z mnoha různých úhlů, obraz může být rozostřený, přesvícený, obličeje mají různou velikost v závislosti na vzdálenosti od fotoaparátu, lidé nosí brýle, vlasy jim překrývají části tváře nebo se mohou různě šklebit, čímž jsou deformovány různé obličejové proporce.

Někdy dokonce i lidé se nechají snadno ošálit, pokud jde o rozpoznání toho, zda se v daném místě vyskytuje obličej, či nikoliv. Poměrně často se setkáváme s případy, kdy lidé spatřili obličej Panny Marie na stěně kostela nebo v kusu kamene na povrchu Marsu. Stejně tak nejsou neomylné ani algoritmy, které se používají v této oblasti. Obecně se setkáváme s dvěma různými typy chyb podle [15]:

- false positive – rozpoznání obličeje v místě, kde žádná tvář není,
- false negative – nerozpoznání obličeje v místě, kde je tvář

2.2 Základní rozdělení

Existují 4 základní přístupy pro detekci obličejů definované v [18]:

Knowledge-based
Obličej je reprezentován za použití sady pravidel, které vychází z předchozích znalostí

o tom, z jakých částí se skládá obličej a jejich vzájemných vztazích (např. obličej má dvě oči, které jsou vzájemně symetrické, ústa a nos). Nevýhodou je, že je velmi složité vyjemnovat všechny možná pravidla pro obličeje v různých polohách.

Feature invariant
Jsou nalezeny výrazné obličejové rysy (oči, ústa, nos, obočí) a následně jsou ověřeny geometrické vztahy mezi nimi. Tato metoda je méně náchylná na různé polohy, variace.

Template matching
Je uloženo několik základních vzorů pro popis obličeje nebo jednotlivých rysů, ty jsou vytvořeny ručně a ne tréninkem, následuje výpočet korelace mezi vstupním obrazem a vzorem. Umí zpracovat velké množtví úhlů a výrazů, ale za cenu nízké rychlosti.

Appearance-based
Detekci předchází trénování klasifikátoru na dostupném datasetu obsahujícím obrazy s obličeji a bez obličejů. Obraz je skenován pomocí podoken, které jsou vyhodnoceny klasifikátorem na shodu. Umí zpracovat velké množtví úhlů a výrazů.

Příkladem jsou neuronové sítě, eigenfaces, SVM, AdaBoost, Viola Jones,.... Ze všech výše jmenovaných je tento přístup nejúspěšnější.

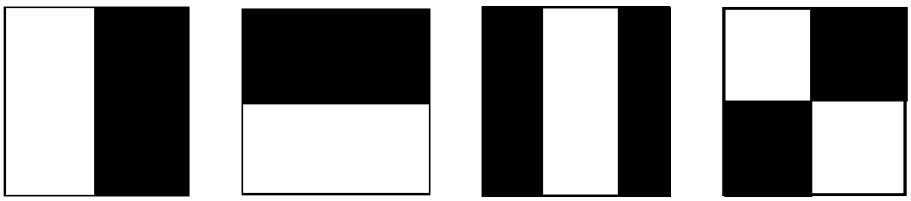
2.3 Viola Jones, AdaBoost

Jedná se o jeden z prvních přístupů pro detekci objektů v reálném čase se zachováním vysoké procentuální úspěšnosti. Podle [16] dosahuje rychlosti až 15 snímků za sekundu při použití na procesoru 700 MHz Intel Pentium III.

Zároveň byl v tomto článku představen pojem integrálního obrazu pro zrychlení výpočtů. Dalším klíčovou součástí je využití algoritmu učení postaveného na AdaBoostu, který může být natrénován pro detekci různorodých objektů, ale primárním účelem je nalezení tváří v obraze.

Posledním důležitým prvkem je využití kaskády klasifikátorů, která umožňuje rychlému průchodu nad daným snímkem bez toho, aniž by bylo třeba dlouze analyzovat části, kde se s vysokou pravděpodobností nevyskytuje obličej.

2.4 Haarovy příznaky



Obrázek 2.1: Základní Haarovy příznaky

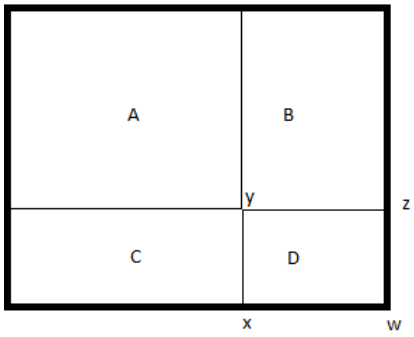
Na obrázku 2.1 jsou čtyři základní typy příznaků. Jejich podoba vychází z předpokladu, že například obličej se skládá ze dvou očí z tmavých pixelů, které jsou odděleny světlým nosem. Stejně tak ústa mají odlišnou barvu, než je barva kůže obličeje. Tento předěl je

vyjádřen hodnotou Haarova příznaku, která se spočítá jako rozdíl mezi intenzitou pixelů uvnitř bílé a černé plochy.

Tyto čtyři základní typy byly později rozšířeny v [11] o další příznaky např. rotované o 45deg, což podle stejné publikace přineslo snížení počtu nesprávných detekcí až o 10%.

Obrázek 2.2: Nicolas Cage 24x24, Haarovy příznaky.
Upraveno ze zdroje [1].

2.5 Integrální obraz



Obrázek 2.3: [autorský obrázek] Integrální obraz

Postup popsaný v 2.4 je ovšem výpočetně náročný. Zrychlení sčítání je možné dosáhnout výpočtem integrálního obrazu v každém bodě. Jedná se o součet pixelů obdélníku od levého horního rohu do daného bodu.

Jakmile máme k dispozici integrální obraz pro každý bod, jsme schopni spočítat sumu intenzit pro libovolný obdélník ve vstupním obraze. Předpokládejme tedy, že pro body w, x, y, z v nákresu výše známe jejich integrální obraz. Poté suma pixelů uvnitř obdélníku D se dá spočítat jako

$$D = I_o(w) - I_o(x) - I_o(z) + I_o(y) \tag{2.1}$$

kde I_o je integrální obraz.

Ve výsledku nám tedy stačí 3 operace sčítání (resp. odečítání), nalezení 4 hodnot v tabulce a jsme schopní najít sumu pixelů jakéhokoliv obdélníku v obraze v konstantním čase.

Obrázek 2.4: Analýza vstupního obrazu kaskádou klasifikátorů.
Upraveno ze zdroje [2].

2.6 AdaBoost

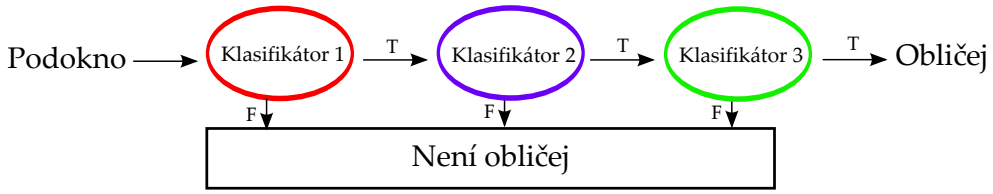
Tento algoritmus poprvé definoval Yoav Freund a Robert Schapire v [5]. Vstupní obraz je skenován pod-oknem o velikosti 24x24 pixelů. Každý z příznaků 2.1 je škálován a posouván přes podokno, což ve výsledků dává zhruba 160 000 různých kombinací příznaků ([16] uvádí 180 000). I přes rychlost výpočtu jednotlivých příznaků potřebujeme celý proces detekce urychlit. K tomuto účelu slouží algoritmus AdaBoost. Využívá více slabých klasifikátorů s horší úspěšností k vytvoření jednoho silného.

$$F(x) = sign[\alpha_1 f_1(x) + \alpha_2 f_2(x) + ... + \alpha_n f_n(x)] \tag{2.2}$$

$$\alpha - \text{váha}$$

Algoritmus Viola Jones využívá kaskádu klasifikátorů, kdy jednotlivé stupně jsou vytvořeny slabými binárními klasifikátory. První klasifikátor pracuje pouze se dvěma základními Haarovými přiznaky pro eliminaci většiny objektů, které nejsou tvář. Touto fází projde 100% obličejů, ale spolu s nimi 40% false positives výsledků. K jejich eliminaci se pokračuje dalšími úrovněmi kaskády.

Pokud neprojde kterýkoliv předchozí test, je klasifikace ukončena a výsledek je zamítnut jako obličej. Pokud podokno projde všemi testy, je vyhodnoceno jako obličej.



Obrázek 2.5: [autorský obrázek] Kaskáda klasifikátorů.

Druhý a každý další klasifikátor nejsou trénovány s kompletním tréninkovým datasetem, ale pouze s obrázky, které prošly přechozími úrovněmi. Typicky bude ve výsledků nalezeno několik shod na jeden obličej, proto se použije metoda non-maximum supression pro sjednocení překrývajících se obdélníků. Výsledná kaskáda vyvinutá Violou a Jonesem měla 32 fázi s využitím 4 297 různých rysů. Trénink trval týdny.

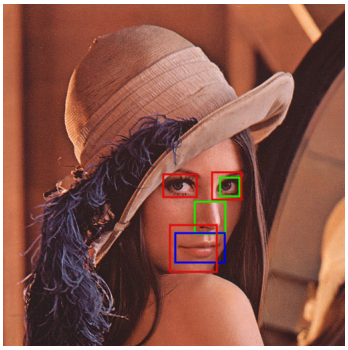
[1] Obrázek ve formátu JPEG. Dostupné z: http://famousface.us/wp-content/gallery/nicolas-cage/nicolas-cage-11.jpg

[2] Obrázek ve formátu BMP. Dostupné z: http://www.hack4fun.org/h4f/sites/default/files/bindump/lena.bmp

2.7 Detekce částí obličeje

Algoritmus 2.3 lze aplikovat i na detekci relevantních obličejových částí jako jsou oči, ústa a nos. Klasifikátory v kaskádě jsou natrénovány dvěma sadami obrázků pro každý detekovaný objekt. První obsahuje scénu, kde se hledaný objekt nevyskytuje jako obrázky běžných předmětů - hory, lesy, papírová sponka a druhá sada obsahuje výskyt jednoho či více instancí hledaného objektu s co možná největší variablitiou (např. při detekci obličejových částí je vhodné pracovat se vzorky lišícími se věkem, pohlavím, barvou pleti).

Ve výsledku jsou vytvořeny celkem odlišné klasifikátory natrénované na různé části obličeje jako oči, ústa a nos. Při testování v [17] s 5000 negativními snímky a 1500 pozitivními pro každou část obličeje tyto klasifikátory stále ale dosahují 23-29% false positive výsledků.

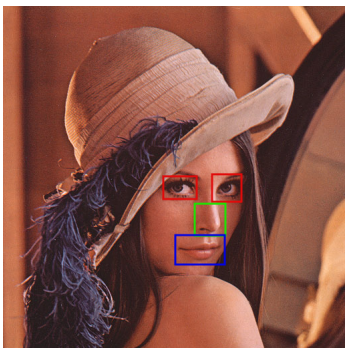


Obrázek 2.6: Ukázka false positive výsledků - oči (červená), nos (zelená), ústa (modrá). Upraveno ze zdroje [2].

Řešení navrhované [17] je minimalizovat prohledávané oblasti v obraze na regiony s vysokou pravděpodobností výskytu hledaných objektů. Analýzou menší části obrazu dochází zároveň ke snížení obsahu integrálního obrazu a tedy nárůstu rychlosti algoritmu.

Vycházíme z předpokladu, že oči, ústa a nos se nachází na obličeji, proto prvním krokem je detekce obličeje. Naneštěstí se false positive výsledky nachází i v této oblasti. Proto je zapotřebí obličej dále rozdělit do regionů s vysokou pravděpodobností výskytu daného objektu.

Předpokládá se, že oči se nachází blízko vrcholu hlavy, nos v blízkosti středu tváře a ústa ve spodní části. Při dalším testování za použití těchto heuristik byly eliminovány všechny false positive výsledky.



Obrázek 2.7: Detekce částí obličeje bez false positive výsledků - oči (červená), nos (zelená), ústa (modrá). Upraveno ze zdroje [2].

Kapitola 3

Sledování detekovaného obličeje

Poté, co je v obrazu nalezen obličej lze předpokládat, že mezi následujícími snímky se bude daný objekt v obraze stále vyskytovat a pouze případně měnit svou pozici. Z toho důvodu není nutné pro každý další snímek aplikovat výpočetně náročnou detekci obličeje, ale nalezený obličej pouze sledovat a o detekci se pokoušet pouze každých N snímků.

3.1 Camshift

Camshift je algoritmus pro sledování obličeje v obraze na základě barevné informace. Tato metoda byla představena v roce 1998 G. Bradskim v publikaci [2] a vznikla úpravou dále popsaného algoritmu mean shift.

Histogram backprojection

Tato metoda je jedním z kroků při použití algoritmu 3.1 a slouží pro nalezení objektu v obraze. Z něj je vytvořen nový obraz stejné velikosti obsahující pouze jeden barevný kanál a v tomto novém obrazu pak každý pixel koresponduje s pravděpodobností jeho výskytu v obraze hledaného objektu.

Barevný histogram je jedním ze způsobů, jak reprezentovat rozložení barev v obraze. Na jeho x-ové ose jsou znázorněny jednotlivé hodnoty, kterých mohou pixely v daném barevném modelu nabývat, na y-ové ose se pak nachází jejich celkový počet.

Nejvhodnějším barevným prostorem pro zobrazení kůže je podle [13] HSV. Ten je dekomponován na jednotlivé složky a pro vytvoření histogramu se použije pouze hodnota odstínu (hue).

Poté dochází k průchodu všech pixelů cílového snímku, kdy každý je opět interpretován ve vybraném barevném modelu a jeho hodnota je nalezena v původním histogramu. Ta je pak uložena do nového snímku. Ve výsledku dochází ke zvýraznění pixelů z původního histogramu a pokud ten obsahoval převážně obličej, na následujícím snímku budou zvýrazněny pixely v barvě kůže. Z výše uvedeného vyplývá, že metoda je náchylná na změnu osvětlení. Formální definici této metody lze najít v [14].

Meanshift

Efektivní algoritmus pro sledování objektů, jejichž vzhled je definován barevnou informací jako například barva kůže, publikovaný již v roce 1975 v [4]. Na jeho základě je pak dále postaven algoritmus 3.1.

Zjednodušený princip je následující. Uvažujme množinu bodů v prostoru (v případě zpracování obraze může jít např. o distribuci pixelů jako 3.1), z nichž část je ohraničena oknem o základním geometrickém tvaru (kruh, obdélník). Úkolem je posunout toto okno tak, aby se v něm vyskytovalo co nejvíce bodů.

Mean-shift algoritmus iterativně vyhledává lokální maxima hustoty bodů uvnitř okna a v každém kroku se okno posunuje ve směru váženého průměru vzorků.

Camshift

Při sledování obličeje často dochází k přesunu i ve směru oddálení/přiblížení, na což je třeba vhodně reagovat změnou velikosti sledovaného okna. Princip je následující. Prvně dochází ke konvergenci okna za pomocí algoritmu mean-shift a je tedy nalezena aktuální pozice obličeje. Podle [2] je pak následně podle vzorce

$$s \ = 2 * \sqrt{\frac{M_{00}}{256}} \tag{3.1}$$

kde M_{00} přestavuje sumu hodnot všech pixelů v okně, velikost okna v následujícím snímku změněna tak, že jeho šířka má velikost s a výška je rovna hodnotě $1.2s$, vzhledem k tomu, že obličeje mají eliptický tvar.

Tracking

Pro uvedení výše zmíněných postupů do praxe je nutné nejprve nalézt okno vymezující obličej. Dále je provedena dekompozice pixelů okna na jednotlivé složky barevného modelu HSV a vytvoření histogramu pro složku odstínu.

Pro každý následující snímek aplikujeme metodu 3.1 a následně je třeba posunout okno ve směru největší hustoty zvýrazněných bodů algoritmem 3.1.

Iterativně se vypočítává nový střed okna, dokud nedojde ke konvergenci. Pro výpočet nového středu okna (x_c, y_c) vycházíme ze vzorců pro obecný moment nultého a prvního řádu.

$$M_{00} = \sum_x \sum_y I(x, y) \tag{3.2}$$

$$M_{10} = \sum_x \sum_y x \times I(x, y) \tag{3.3}$$

$$M_{01} = \sum_x \sum_y y \times I(x, y) \tag{3.4}$$

$$x_c = \frac{M_{10}}{M_{00}} \tag{3.5}$$

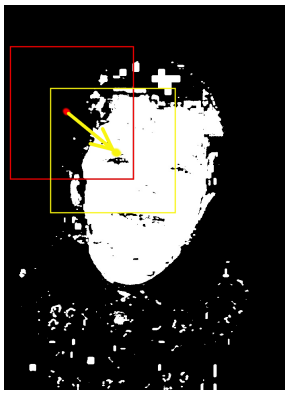
$$y_c = \frac{M_{01}}{M_{00}} \tag{3.6}$$

x a y jsou souřadnice pixelu okna ve snímku. M_{00} představuje obecný moment nultého řádu, tedy sumu všech pixelů daného okna. M_{10} a M_{01} jsou obecné momenty prvního řádu.

Jakmile metoda konverguje, je aplikován vzorec 3.1 pro změnu velikosti okna.

Nový snímek s posunutým obličejem.

Histogram backprojection.



Posunutí okna.

Cílová pozice okna.

Obrázek 3.1: Znázornění algoritmu mean-shift.

3.2 Kalman filter

Tento algoritmus byl představen v roce 1960 maďarským matematikem Rudolfem E. Kálmánem. Zajímavostí a zároveň i potvrzením významu této metody je fakt, že podle [6] byl např. použit pro navigační systém v raketoplánu Apollo, který dokázal přepravit Neila Armstronga ze Země na Měsíc a zase zpátky. To ovšem neznamená, že ani dnes si tato metoda nenajde uplatnění. Naopak je hojně využívána v satelitech, navigačních zařízeních či počítačových hrách.

Cílem algoritmu je filtrace šumu ze vstupního signálu, který je zatížen chybou. Je zapotřebí znát matematický model, který popisuje systém. Metoda pak rekurzivně zpracovává data a generuje optimální odhad při zadané množině měření. Díky tomu, že je rekurzivní, tak nemusí uchovávat hodnoty všech předchozích měření a znovu zpracovávat data v každém kroku.

Podstatné je, že tato metoda pracuje s diskrétním modelem, což znamená, že záleží na měřeních mezi pravidelnými úseky v čase, kdy není známo, co se děje v mezičase.

Algoritmus pracuje na základě predikce budoucího stavu, kterou vypočítává z hodnoty předchozího stavu, směru pohybu a odhadu chyby stavu. Chyba je dynamicky přepočítávána na základě rozdílu mezi predikcí a měřením v současnosti.

Formální definici algoritmu včetně odvození jednotlivých rovnic lze najít popsané v původním díle R.E.Kálmána [8]. Kalman filter byl dále upraven na např. rozšířený Kalman

filter [7] pro výpočty nelineárních modelů a některé další varianty.

3.3 Maďarská metoda

Používá se pro řešení přiřazovacího problému, jehož matematickou formulaci lze najít v [3]. Tu lze zjednodušeně parafrázovat takto:

> Předpokládejme, že existuje n zdrojů, kterým je potřeba přiřadit n úkolů. Je také známá cena přiřazení každého zdroje ke každému úkolu. Je zapotřebí přiřadit všechny úkoly tak, že každý ze zdrojů má přiřazen právě jeden a právě jeden zdroj je přiřazen ke zdroji, tak aby celková hodnota všech přiřazení byla minimální.

Řešení tohoto problém našel H.W.Kuhn [9] a vznikla tak tzv. Maďarská metoda. V rámci příkladu pro vysvětlení jednotlivých kroků při výpočtu se vychází z předpokladu, že existují 4 zdroje a 4 úkoly. Pokud by tvar matice nevycházel jako čtverec, pak je zapotřebí doplnit matici o patřičný počet řádků příp. sloupců obsahujících n všech místech nejvyšší hodnotu matice.

Krok 1: Je zadána matice

$$\begin{pmatrix} 82 & 83 & 69 & 92 \\ 77 & 37 & 49 & 92 \\ 11 & 69 & 5 & 86 \\ 8 & 9 & 98 & 23 \end{pmatrix}$$

Krok 2: Odečtení minima na řádku

$$\begin{matrix} -69 \\ -37 \\ -5 \\ -8 \end{matrix} \begin{pmatrix} 13 & 14 & 0 & 23 \\ 40 & 0 & 12 & 55 \\ 6 & 64 & 0 & 81 \\ 0 & 1 & 90 & 15 \end{pmatrix}$$

Krok 3: Odečtení minima ve sloupci

$$\begin{matrix} -0 & -0 & -0 & -15 \end{matrix} \begin{pmatrix} 13 & 14 & 0 & 8 \\ 40 & 0 & 12 & 40 \\ 6 & 64 & 0 & 66 \\ 0 & 1 & 90 & 0 \end{pmatrix}$$

Krok 4: Překrytí všech nul minimálním počtem vertikálních a horizontálních čar. Pokud by existovaly 4 (v obecné matici n) čáry, pak algoritmus tímto krokem končí, jinak pokračuje krokem 5.

$$\begin{pmatrix} 13 & 14 & 0 & 8 \\ 40 & 0 & 12 & 40 \\ 6 & 64 & 0 & 66 \\ 0 & 1 & 90 & 0 \end{pmatrix}$$

Krok 5: Nalezení minimální hodnoty mezi nepřekrytými prvky. Odečtení tohoto čísla od všech nepřekrytých prvků a součet s čísly, které jsou překryty dvakrát (jak horizontální, tak vertikální čarou). Pokračuje se krokem 4.

$$\begin{pmatrix} 7 & 8 & 0 & 2 \\ 40 & 0 & 12 & 40 \\ 0 & 58 & 0 & 60 \\ 0 & 1 & 90 & 0 \end{pmatrix}$$

Krok 6: Překrytí minimálním počtem čar. Počet čar je roven počtu řádků/sloupců, konec algoritmu.

$$\begin{pmatrix} 7 & 8 & 0 & 2 \\ 40 & 0 & 12 & 40 \\ 0 & 58 & 0 & 60 \\ 0 & 1 & 90 & 0 \end{pmatrix}$$

Krok 7: Zvýrazněné nuly označují pozice prvků optimálního přiřazení.

$$\begin{pmatrix} 7 & 8 & 0 & 2 \\ 40 & 0 & 12 & 40 \\ 0 & 58 & 0 & 60 \\ 0 & 1 & 90 & 0 \end{pmatrix} \longrightarrow \begin{pmatrix} 82 & 83 & 69 & 92 \\ 77 & 37 & 49 & 92 \\ 11 & 69 & 5 & 86 \\ 8 & 9 & 98 & 23 \end{pmatrix}$$

Kapitola 4

Návrh řešení

Cílem aplikace je záměna obličejů (dále také jako tzv. faceswap) na vstupním snímku z fotoaparátu mobilního zařízení. Příklad podobné záměny je na vidění na obrázku 4.1.

Standardní výbavou dnešních mobilů a tabletů je jak přední, tak i zadní kamera, s čímž pracuje i výsledná aplikace a uživateli je umožněno mezi nimi přepínat.

Obrázek 4.1: Ukázka záměny obličeje.
Pochází ze zdroje [1].

[1] Obrázek ve formátu JPEG. Dostupné z: http://farm5.staticflickr.com/4144/5096533113_caa92a3895_o.jpg. Chráněno licencí Creative Commons: http://creativecommons.org/licenses/by-nc/2.0/

Při průzkumu již existujících aplikací na Google Play, tedy oficiálním marketu s aplikacemi pro Android, se ukázalo, že bez výjimky všechny volně dostupné aplikace pracují tímto způsobem:

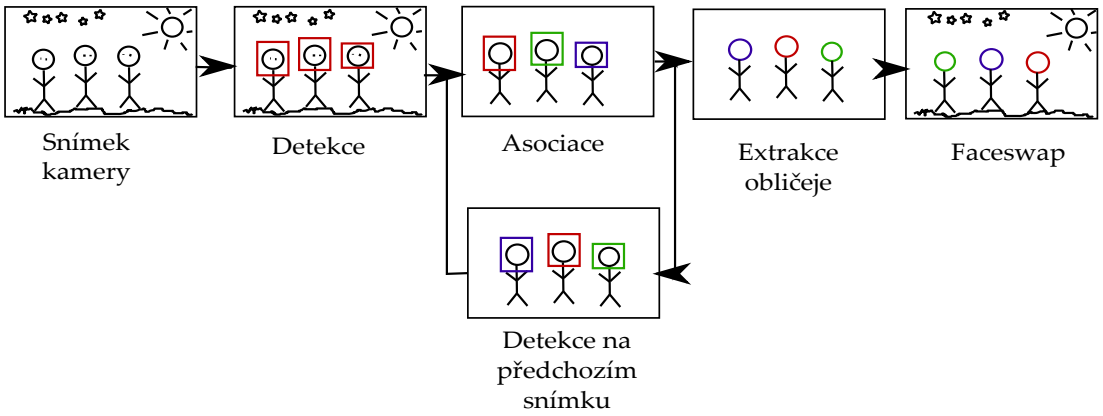
1. Pořízení snímku z fotoaparátu nebo nahrání obrázku z galerie.
2. Manuální, v lepším případě automatická detekce obličeje.
3. Záměna tváří.

Tato aplikace tedy byla vyvíjena se snahou odlišit se od tohoto zajetého principu, přistoupit k problému odlišně a pokusit se o to, aby výsledek působil pro uživatele stejným dojmem jako při použití kamery.

Není tedy zapotřebí pořizovat každý snímek zvlášť a čekat na jeho zpracování, ale důraz je převážně kladen na běh v reálném čase. Po spuštění tedy k faceswapu dochází automaticky, jakmile jsou detekovány obličeje ve vstupním obraze.

Je proto nežádoucí, aby uživatel zaznamenal zpomalení aplikace např. při detekci obličejů, která je výpočetně nejnáročnější částí, a proto se stala středobodem při návrhu řešení koncové aplikace.

4.1 Princip aplikace



Obrázek 4.2: [autorský obrázek] Pozice detekovaných obličejů na aktuálním

V každém snímku z fotoaparátu je detekován obličej a tato oblast zájmu (často se uvádí jako tzv. ROI[2]) má tvar čtverce. Nalezené oblasti jsou asociovány vzhledem k již dříve nalezeným obličejům, což umožňuje rozlišit tváře. Po tomto kroku jsou tedy známy spojitosti mezi dvěma snímky.

Ze čtvercové oblasti je následně třeba vymaskovat skutečný tvar obličeje, který bude zaměněn. Tyto jednotlivé tváře jsou poté následně popřeházeny.

4.2 Metody pro detekci obličeje

Pro nalezení tváře ze vstupního snímku mobilního zařízení nabízí platforma Android několik řešení, které budou následně popsány a srovnány.

[2]ROI - region of interest

API 1

První myšlená varianta byla detekce pomocí metod vestavěných v rámci samotného Android Software Development Kit (dále jen SDK). Ten již od API 1, které odpovídá Androidu 1.0, poskytuje metodu `FaceDetector.findFaces()`. Nalezené tváře jsou pak instancemi třídy `FaceDetector.Face`, kdy každý z těchto objektů poskytuje pouze informaci o pozici nalezeného obličeje a i přestože podle oficiální dokumentace je jedním z atributů i informace o natočení tváře, tak je tato hodnota vždy nastavena na 0.

Dalším důležitým aspektem tohoto detektoru je, že vací i informaci o detekovaných očích. I přes snahu minimalizovat výpočetní čas této metody přepsáním částí kódu např. pro převod formátu obrazu se naneštěstí doba výpočtu metody `findFaces()` sama o sobě pohybovala v rozmezí 500-1000ms při rozlišení obrázku 640x480px.

Výsledná rychlost aplikace se poté tedy pohybuje na testovacím zařízení v rozmezí 1-2 snímků za sekundu (dále jen FPS) v závislosti na počtu nalezených obličejů v obraze, což je nedostačující pro běh v reálném čase, pokud by měly být tváře detekovány v každém vstupním snímku.

API 14

Pro zařízení s patřičnou hardwarovou podporou (neexistuje jejich oficiální seznam, ale jedná se o převážnou většinu mobilních telefonů vydaných po zveřejnění Androidu 4.0 ICS) a API verze 14 a vyšší a je k dispozici třída `Camera.FaceDetectionListener`.

Nalezené obličeje jsou pak instancemi třídy `Camera.Face`, jejíž atributy poskytují informace o pozici očí, úst i natočení obličeje, což je dostačující pro následné zpracování.

Jako další nespornou výhodou je i rychlost, která odpovídá požadavkům na zpracování obrazu v reálném čase a dosahuje tedy 10-12 FPS.

Na druhou stranu kvůli absenci seznamu zařízení, na kterých je tato metoda podporována, minimálnímu povědomí toho, co se děje pod pokličkou samotné funkce a tudíž nulové šanci zasáhnout do zpracování samotné detekce a v neposlední řadě faktu, že testované zařízení nepatří mezi podporované telefony, padlo výsledné řešení na použití knihovny OpenCV.

OpenCV

Dalším možným způsobem, jak detekovat obličeje na mobilním zařízení je za použití volně dostupné knihovny OpenCV. Ta je šířena pod BSD licencí a je tedy zdarma pro akademické a komerční použití. Mimojiné nabízí široké spektrum dalších metod pro zpracování obrazu včetně implementace algoritmů popsaných v kapitole 2, díky čemuž jsem získal větší svobodu při implementaci detektoru a následně trackeru.

Pro detekci objektů tato knihovna implementuje kaskádový klasifikátor, který vychází z Viola-Jones algoritmu 2.3. Výsledná funkčnost detektoru je pak odvislá od vstupní kaskády, která určuje, co by měl detektor v obraze hledat. K dispozici je poměrně rozsáhlé množství kaskád natrénovaných pro detekci tváří s rozdílnou úspěšností.

Důležité je rozlišovat Haarovy a LBP[3] kaskády. LBP je na tom v otázce rychlosti podstatně lépe než Haarovy kaskády díky použití celočíselné aritmetiky (Haarova kaskáda pracuje s plovoucí desetinnou čárkou) za cenu ztráty přesnosti. To způsobuje poměrně často se vyskytující false positive výsledky. Více informací o LBP v [10].

Pro detekci obličejových částí je zapotřebí použít další natrénovanou kaskádu.

[3]LBP - Local Binary Patterns

4.3 Asociace obličejů

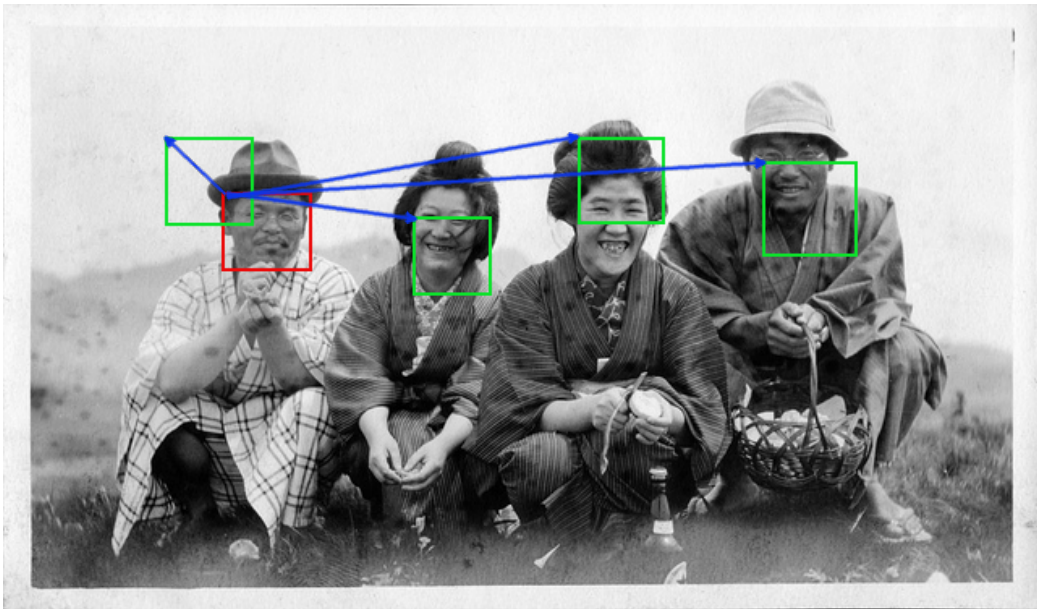
Předpokládejme, že na snímku F_n byly nalezeny 3 obličeje A, B a C. Na následujícím snímku F_{n+1} byly opět nezávisle na předchozím výsledku získány souřadnice 3 hledaných objektů např. X, Y a Z.

Momentálně jsme tedy postaveni před otázku, jak spolu tyto tři výskyty hledaných objektů v různých časech souvisí. Nejpřesnější by v tomto ohledu byla metoda rozpoznávání tváří, což je ovšem pro účely této aplikace zároveň nejpomalejší způsob.

Jako řešení, které je dostatečně výpočetně nenáročné bylo tedy zvoleno využít Maďarskou metodu 3.3 a hodnoty vstupní matice určit jako euklidovskou vzdálenost mezi středy detekovaných oblastí zájmů.

$$\begin{array}{c} \\ C_1 \\ C_2 \\ C_3 \\ C_4 \end{array} \begin{array}{c} \begin{array}{cccc} P_1 & P_2 & P_3 & P_4 \end{array} \\ \begin{pmatrix} d_{11} & d_{21} & d_{31} & d_{41} \\ d_{12} & d_{22} & d_{32} & d_{42} \\ d_{13} & d_{23} & d_{33} & d_{43} \\ d_{14} & d_{24} & d_{34} & d_{44} \end{pmatrix} \end{array} \tag{4.1}$$

Sloupce v matici 4.1 představují tváře detekované na předchozím snímku, řádky jsou nalezené objekty na aktuálním a hodnoty d_{xy} v matici pak reprezentují danou euklidovskou vzdálenost.

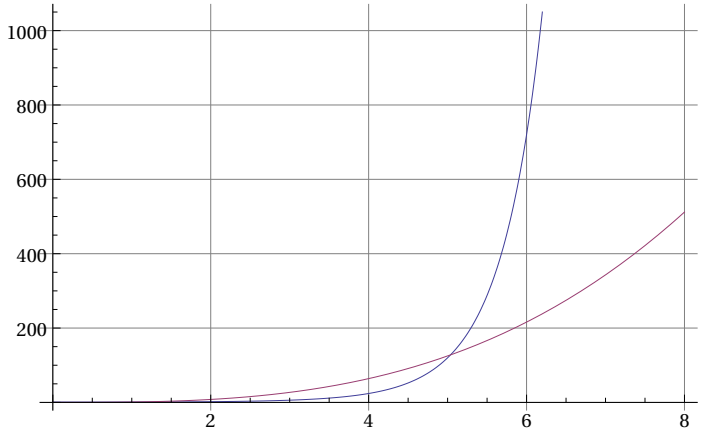


Obrázek 4.3: Pozice detekovaných obličejů na aktuálním (červeně) a na předchozím (zeleně) snímku. Modře je znázorněna euklidovská vzdálenost mezi levými horními rohy detekovaných čtverců. Pochází ze zdroje [4].

Při nalezení n obličejů v aktuálním a předchozím obraze existuje celkem $n!$ možností, jak je možné dané obličeje přiřadit. Při použití Maďarské metody je časová náročnost algo-

[4] Davey, A.: Group Photo. 1915,
Dostupné z: https://www.flickr.com/photos/adavey/5099892792
chráněno licencí Creative Commons: http://creativecommons.org/licenses/by-nc/2.0/.

ritmu n^3, takže se její použití vyplatí již při 5 detekovaných obličejích. Rostoucí výpočetní náročnost je znázorněna v grafu 4.4.



Obrázek 4.4: Grafy funkcí $f(x) = x!$ (modře) a $f(x) = x^3$ (červeně).

4.4 Filtrace výsledků

OpenCV detektor vrací poměrně velké množství false positives výsledků, které se vyskytují zcela nahodile a přerušovaně. K eliminaci těchto chyb je použita heuristika, kdy ke každému detekovanému obličeji, reprezentovanému jako pozice v poli nalezených detekcí, je přiřazen čítač, jehož hodnota odpovídá počtu instancí tohoto obličeje nalezených v po sobě jdoucích snímcích.

Pokud překročí nastavenou mezní hodnotu (např. 5 po sobě jdoucích snímků), pak je výsledek zpracováván dále a vykreslen. To vše za cenu prvotního zpomalení v rozmězí 0.5s - 1s.

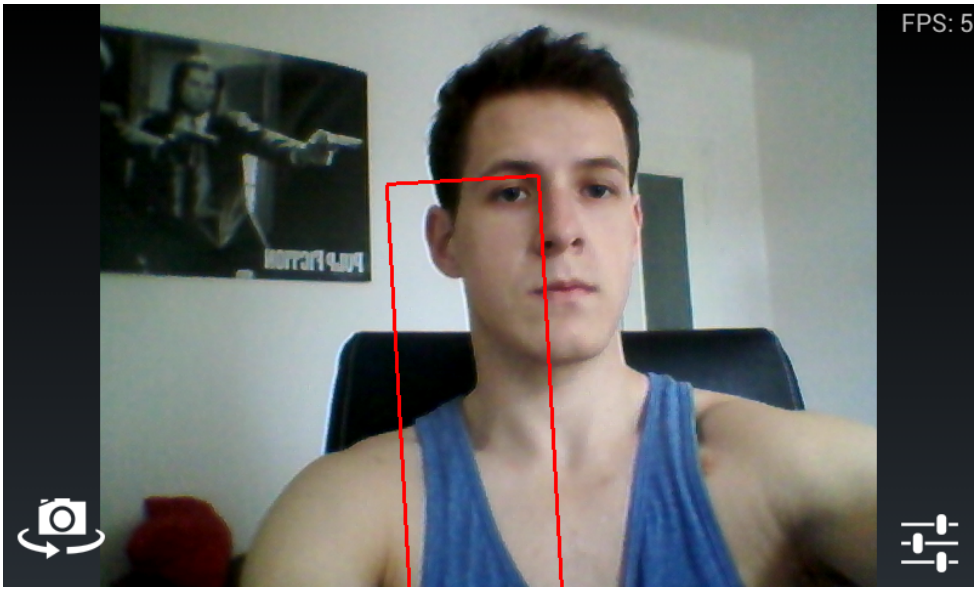
4.5 Tracking

Z hlediska výkonu je výhodné jednou nalezený obličej dále sledovat. K tomu poslouží implementace algoritmů popsaných v kapitole 3 z knihovny OpenCV.

Camshift

Tento postup byl zvolen jakožto prvotní varianta především z důvodu vysoké rychlosti vzhledem k tomu, že vychází pouze z barevného histogramu dané ROI. Jeho další nespornou výhodou je to, že dokáže reagovat na natočení hlavy do stran.

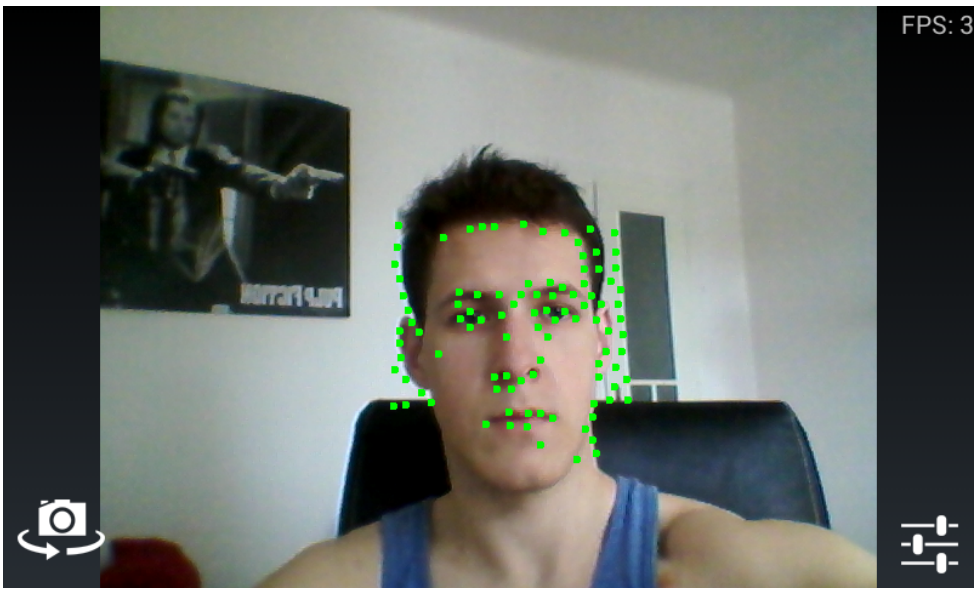
Při experimentech se naneštěstí ukázalo, že pro účely aplikace je naprosto nevhodný protože již při malé změně osvětlení vznikaly velké nepřesnosti při trasování kůže a dalším nešvarem je zkreslení detekovaného čtverce, pokud se k obličeji přiblížil další objekt s podobnou barevnou informací (např. ruka), což způsobuje až moc nepředvídatelné chování.



Obrázek 4.5: Screenshot z aplikace při použití Camshift trackeru.

Kanada-Lucas tracker

Dalším použitým algoritmem pro trasování obličeje je Kanade-Lucas tracker, který dokáže sledovat určené body, v tomto případě např. charakteristické obličejové rysy. Při testování bohužel nedosahoval svou rychlostí požadavkům pro zpracování obrazu v reálném čase.



Obrázek 4.6: Screenshot z aplikace při použití Kanada-lucas trackeru.

Kalman filter

Algoritmus popsaný v sekci 3.2, slouží k vyhlazení přechodů (šumu) mezi detekovanými snímky. Čtvercové okno se poté pohybuje plynule, jsou eliminovány skoky mezi

snímky.

Matice filteru je sestavená podle pohybových rovnic.

Jeho implementace není součástí Java nastavby knihovny OpenCV, proto bylo zapotřebí jej implementovat v nativním kódu a C++ OpenCV.

4.6 Extrakce obličeje

Informace o nalezených obličejích je dána jako souřadnice a rozměry čtverce ve snímku. Pokud tedy člověk nemá čtvercovou hlavu, je následně zapotřebí odlišit popředí (tvář) od pozadí (např. okolní prostředí).

Grabcut

Jedná se o algoritmus implementovaný v rámci knihovny OpenCV, který slouží pro extrakci popředí od pozadí. Jediná režie z pohledu uživatele/programátora je v tomto případě v podobě vykreslení obdélníku okolo oblasti, kde je zapotřebí oddělit popředí od pozadí a iterativním voláním metody jsou získávány přesnější výsledky. Podstatným problémem je rychlost této metody, která trvá v rámci sekund a neodpovídá tedy požadavkům pro běh v reálném čase.

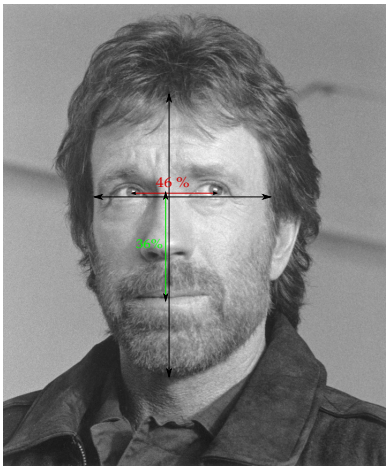
Detekce očí

Dalším způsobem, jak aproximovat tvar obličeje je pomocí detekovaných očí nebo úst. Podle [12] je vertikální vzdálenost mezi očima a ústy přibližně 36% svojí délky a vodorovná vzdálenost mezi očima je 46% šířky tváře.

Výhodou je, že pro tyto účely není zapotřebí implementovat žádné další třídy/metody, protože o samotnou detekci se postará stejný detektor jako při hledání tváře, pouze s odlišnou natrénovanou kaskádou.

Tyto kaskády existují v mnoha různých podobách, jsou např. samostatně natrénované pro levé a pravé oko, pro obě oči, oči s brýlema,... Bohužel všechny jsou Haarovy a pracují tedy s plovoucí desetinnou čárkou, což způsobuje i jejich výraznou pomalost a nepodařilo se najít LBP kaskádu. Nejrychlejší se ukázala být `haarcascade\_eye.xml`, ale ta obsahovala tak velké množství false positives výsledků, že byla pro práci nevhodná.

Vychází se navíc z předpokladu, že obličej musí být natočen zepředu, jinak nebudou nalezeny všechny relevantní části.

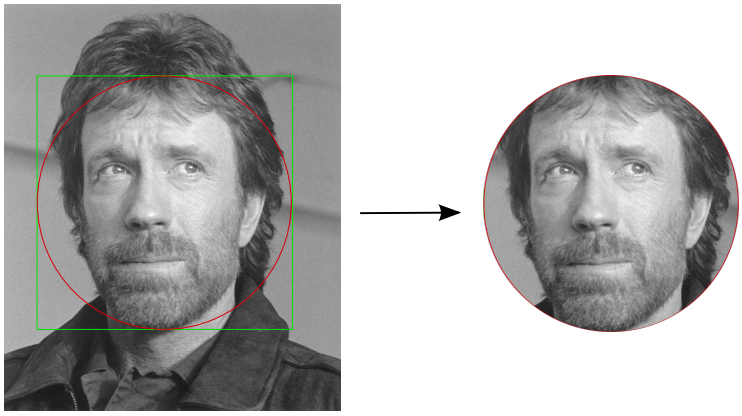


Obrázek 4.7: Poměr stran obličeje.
Upraveno ze zdroje [5].

elipsa, kruh

Jako nejrychlejší způsob se jeví použití základního geometrického tvaru pro vymaskování hledaného obličeje ve čtverci.

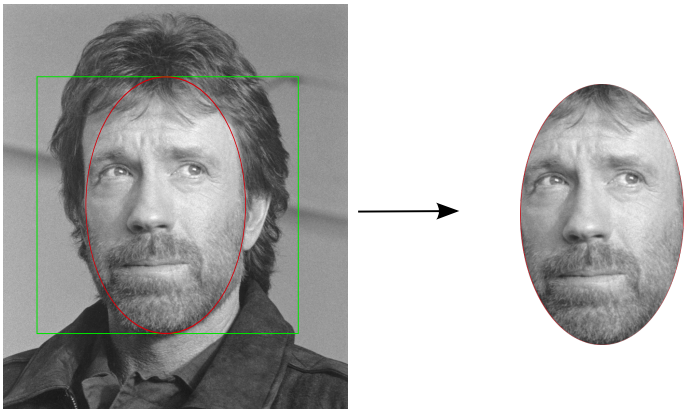
Prvním vyzkoušeným tvarem byla kružnice. Ve většině případů ovšem i tento kruh obsahoval části pozadí.



Obrázek 4.8: Vyříznutí kruhové masky.
Upraveno ze zdroje [5].

Dalším testovaným útvarem byla elipsa o rozměrech . . . dovnitř detekovaného čtverce. Jedná s nejběžnější tvar obličeje podle

[5] Obrázek ve formátu JPEG. Dostupné z:
http://www.americaremembers.com/wp-content/uploads/2013/08/Chuck-Norris-BW.jpg



Obrázek 4.9: Vyříznutí eliptické masky.
Upraveno ze zdroje [5].

4.7 FaceSwap na základě porovnání histogramu

Poslední otázka, která zůstává je, v jakém pořadí by měly být výsledné obličeje při automatické záměně vyměněny. Původní myšlenka byla jednoduchá záměna každého obličeje v poli za obličej na následujícím indexu (první byl tedy vyměněn za druhý, druhý za třetí,... až poslední za první). Vzhledem k faktu, že jednotlivé tváře se liší rozdílnou barvou a odstínem kůže, byly výsledky těchto nahodilých záměn neuspokojující, přestože tedy byla tato metoda rychlá.

Obrázek 4.10: Záměna obličejů při použití kruhového indexu.
Upraveno ze zdroje [6].

Barevný tón kůže lze nejpřesněji popsat pomocí hodnot hue a saturation z barevného modelu HSV. Pro tyto dva kanály jsou vytvořeny histogramy pro každý obličej a ty jsou následně porovnány. Pro porovnani histogramu jsou v rámci knihovny OpenCV implementovány 4 různé metriky pro měření podobnosti histogramů. Výsledek porovnání leží v intervalu $\langle -1, 1 \rangle$, kde 1 odpovídá naprosté shodě a -1 neshodě.

[6] Obrázek ve formátu JPEG. Dostupné z: http://static2.businessinsider.com/image/51f288dd6bb3f72041000006-1200/there-we-go-cranstons-entire-breaking-bad-family-plus-pinkman-including-rj-mitte.jpg

Obrázek 4.11: Záměna obličejů při použití porovnání histogramu.
Upraveno ze zdroje [6].

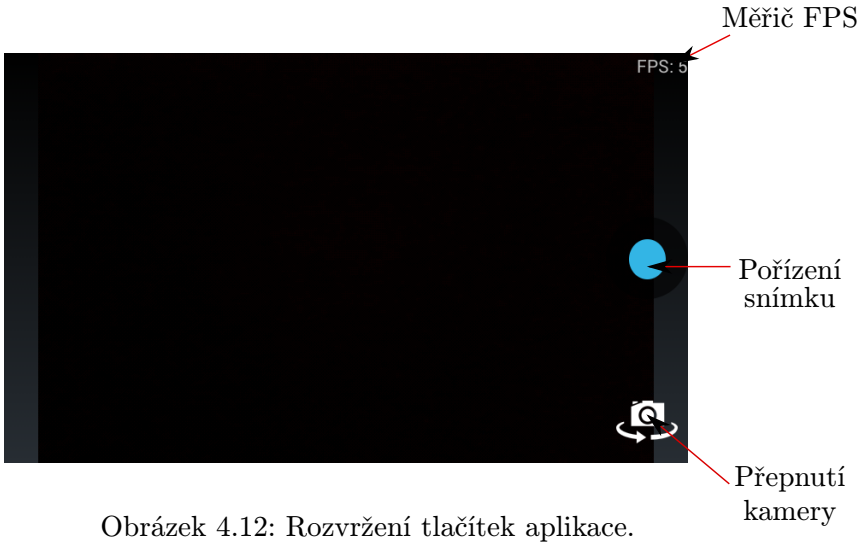
Ve výsledku tedy mohou nastat případy, kdy jeden histogram je podobný více histogramům, ne všechny obličeje musí být tedy při záměně vykresleny. To na druhou stranu s sebou nese věrohodněji vypadající záměny, než kdyby musely být nutně použity všechny obličeje.

4.8 Grafické uživatelské rozhraní

Z pohledu uživatele je účelem aplikace pouze pobavení. Výsledný návrh tedy vychází z předpokladu, že se uživatel nebude zdržovat nastavováním mnoha různých možností a dostane se k samotné funkčnosti hned po spuštění.

Grafické prvky se drží standardů prostředí operačního systému Android a jejich rozvržení vychází z aplikace `Fotoaparát`, která je jedna z aplikací dodávaných se samotným OS, díky čemuž se uživatel při prvním použití rychleji zorientuje.

Po spuštění aplikace je jako primární obrazový vstup nastaven zadní fotoaparát a všechny nalezené obličeje jsou okamžitě automaticky zaměňovány. Pokud fotoaparát zařízení navíc disponuje funkcí stálého autofocusu, tak je obraz tvrale zaostřován.



Obrázek 4.12: Rozvržení tlačítek aplikace.

Základními ovládacími prvky aplikace jsou

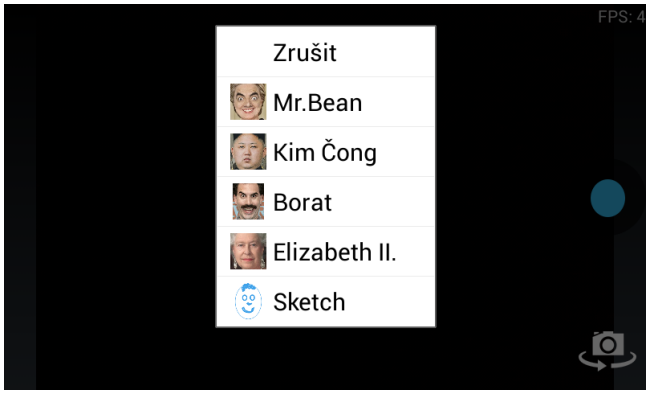
- tlačítko pro otočení kamery (přepnutí ze zadní na přední a naopak)
- pořízení snímku obrazovky a uložení na interní úložiště do složky Faceswap, předchází mu zaostření pomocí autofocusu, pokud ho daný fotoaparát podporuje (přední kamera málokdy)

Dále se zobrazuje měřič FPS, který pouze informuje uživatele o aktuální rychlosti aplikace.

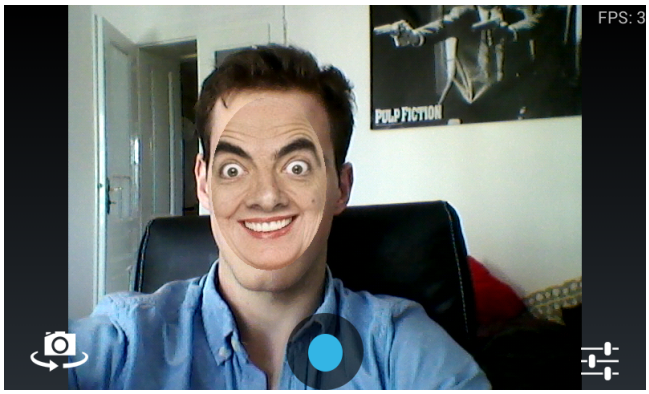
Kromě automatické záměny obličejů na základě podobnosti histogramu, aplikace dále nabízí manuální výměnu detekované tváře za jeden z přednastavených obličejů. Jakmile je nalezena tvář v obraze, stačí na ni kliknout a objeví se dialogové okno nabízející možnost výběru.

Při ztrátě obličeje v obraze, respektive objektu na dané pozici v poli objektů, nedochází ke ztrátě informace o nastavení předdefinované tváře. Tato funkčnost byla implementována na základě reakcí uživatelů vzhledem k tomu, že detektor byl velmi citlivý na chvilkové ztráty a bylo pak nutné předdefinovaný obličej opět inicializovat.

Oba dva tyto módy je možné libovolně kombinovat, polovina tváří může být tedy přednastavena a druhá polovina zaměněna na základě histogramu.



Obrázek 4.13: Dialogové okno po kliku na detekovaný obličej.



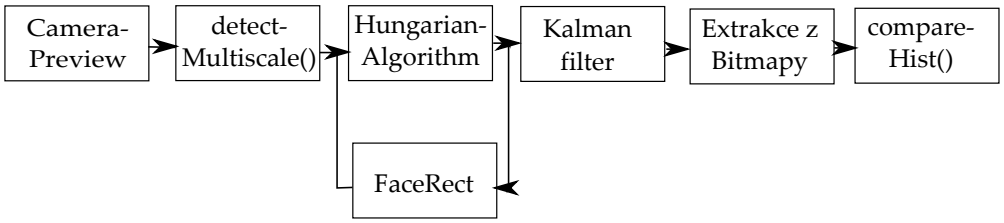
Obrázek 4.14: Faceswap s použitím přední kamery a předdefinovaného obrázku po výběru z dialogového okna.

Další snímky pořízené z aplikace jsou k nalezení v příloze B.

Kapitola 5

Implementace

Návrh na obrázku 4.2 v konkrétní implementaci má pak následující podobu.



Obrázek 5.1: [autorský obrázek] Znázornění jednotlivých kroků při zpracování obrazu.

Algoritmy pro zpracování obrazu byly implementovány pomocí knihovny `OpenCV` verze 2.4.8 pro Android SDK[1].

Vývojovým prostředím bylo IDE Eclipse ve verzi Android Development Tools 22.2.6 na operačním systému Windows 7. K aplikaci byla přidána nativní podpora pomocí Android NDK pro implementaci některých dále popsaných algoritmů. Minimální nastavená verze SDK je 9, což odpovídá operačnímu systému Android 2.3 a výše.

Potřebná práva, která aplikace při instalaci vyžaduje, zahrnují použití fotoaparátu a navíc i Internetu z důvodu zapnuté podpory odesílání logů při pádu aplikace díky integraci knihovny `BugSense`[2].

5.1 CameraActivity

```
public class CameraActivity extends Activity
```

Hlavní a zároveň jediná aktivita. Aby uživatel nebyl nucen instalovat OpenCV Manager na svoje zařízení, je knihovna OpenCV staticky inicializována a dodávaná přímo s FaceSwap aplikací.

Přepínání mezi přední a zadní kamerou je implementováno pomocí restartování celé aktivity, z toho důvodu není zapotřebí hlídat správnou posloupnost přerušení běhu kamery,

[1] Zdroj: http://docs.opencv.org/doc/tutorials/introduction/android_binary_package/O4A_SDK.html
[2] Knihovna pochází ze zdroje: https://www.bugsense.com

reinicializace SurfaceView,... Zároveň je zapotřebí předávat parametr informující o tom, kterou kameru instanciovat.

Pokud je zapnuto internetové připojení, pak na začátku dochází k vytvoření sezení se vzdáleným serverem `BugSense` pro ladění aplikace při jejím nečekaném pádu.

Dále v této třídě dochází k nastavení základního rozhraní, tlačítek a menu, inicializaci kamery a vytvoření instance třídy Preview.

5.2 Preview

```
class Preview extends ViewGroup implements SurfaceHolder.Callback
```

Třída `Preview` zapouzdřuje `CameraPreview` implementuje `SurfaceHolder callback` pro obdržení informací o změně stavu `CameraPreview` a dědí od třídy `ViewGroup`, díky tomu slouží pro vytvoření vlastního layoutu vlastních rozměrů.

Její implementace vychází ze vzorových příkladů dodávaných s Android SDK[3]. Funkcí této třídy je zarovnání kamery na střed. Vzhledem k tomu, že existuje rozličné množství zařízení podporujících operační systém Android, existuje také velké množství rozlišení kamer a ne vždy odpovídají rozlišení kamery rozlišením obrazovky a to samé nemusí platit ani pro poměr stran. O získání nejlepšího možného rozměru se stará funkce `getOptimalPreviewSize()`.

Ta prochází seznam jednotlivých podporovaných rozlišení fotoaparátu a snaží se najít takové rozlišení, aby poměr stran nepřesáhl hodnotu `ASPECT_TOLERANCE` definovanou na 0.1. To znamená, aby šířka/výška Preview - šířka/výška Camera nepřesáhla toto číslo.

Pokud není nalezeno takové rozlišení, tak se cyklus opakuje znovu bez nutnosti tohoto požadavku. Podle nalezeného optimálního rozměru se následně v metodách `onMeasure` a `onLayout` vytvoří layout daných rozměrů.

5.3 CameraPreview

```
public class CameraPreview extends SurfaceView implements Camera.PreviewCallback
```

Dědí od třídy SurfaceView pro vykreslení snímků z kamery na obrazovku a implemetuje `PreviewCallback` pro obdržení jednotlivých snímků z kamery pomocí metody `onPreviewFrame()`.

Je zde vytvořen klasifikátor s daným kaskádovým souborem. Srdcem třídy je funkce `onPreviewFrame()`, jejímž vstupním parametrem je pole Bytů, která reprezentuje snímek z kamery.Tato funkce implementuje také řídící logiku detektoru.

O vše podstatné v oblasti vykreslování se pak stará metoda `onDraw()`.

Existují různé formáty, ve kterých může být vstupní snímek pořízen, nejrozšířenější je YUV420 NV21. Ten je následně zapotřebí převést na bitmapu, ze které je vyříznut obličej.

[3]Cesta k původní vzorové třídě ve složce SDK:
Samples/android-14/ApiDemos/src/com/example/android/apis/graphics/CameraPreview.java

5.4 detectMultiscale

```
public void detectMultiScale(Mat image, MatOfRect objects, double scaleFactor,
                             int minNeighbors, int flags, Size minSize, Size
                                 maxSize)
```

Tuto funkci implemetuje knihovna OpenCV a slouží pro detekci objektů různých velikostí v obraze v závislosti na vstupní natrénované kaskádě. Návratovou hodnotou je poté pole obdélníků (resp. čtverců v případě detekce obličeje).

Podle jednotlivých parametrů při volání funkce lze zásadně měnit její funkčnost, přesnost a rychlost.

scaleFactor – určuje, jak moc je obraz redukován při každém škálování. Zvýšením hodnoty lze výpčoetní rychlost funkce urychlit za cenu ztráty přesnosti detektoru.

minNeighbors – každý detekovaný obličej je nalezen v obraze vícekrát (viz. 2.4), tento parametr pak určuje minimální počet takovýchto detekcí nutných proto, aby byl výsledek uznán jako tvář. Zvýšením hodnoty dojde k eliminaci false positive výsledků, ale zároveň i pravých tváří

minSize – objekty menší než tato hodnota budou ignorovány. Vzhledem k tomu, že se vychází z předpokladu, že detekované obličeje mají nějakou minimální hodnotu, lze tento parametr nastavit a urychlit tak dobu vykonávání funkce (oproti tomu např. při detekci očí, mohou být objekty tohoto typu menší a není ted vhodné parametr nastavovat)

maxSize – objekty větší než tato hodnota budou ignorovány

Z výše zmíněného lze odvodit, že je velmi důležité nastavit správné hodnoty funkce tak, aby se minimalizoval počet navrácených výsledků s false positives a přitom bylo dosaženo co největší rychlosti. Nalezením odpovídajících hodnot se zabývá kapitola 6. Taktéž pro zvýšení rychlosti lze zavést heuristiku, že jsou adaptivně upravovány parametry minSize a maxSize za běhu aplikace podle již detekovaných tváří.

5.5 NDK

Některé algoritmy byly implementovány v nativní vrstvě platformy Android za pomocí NDK pluginu a rozhraní JNI pro kompilaci a propojení kódu virtuálního stroje Javy a nativních souborů napsaných v jazyce C++.

To umožnilo zvýšit rychlost některých výpočetně náročných funkcí (datové struktury jsou předávány odkazem) a zároveň použití některých funkcí knihovny OpenCV, které nebyly v době vývoje implementovány v Java verzi jako např. Kalman filteru.

Kapitola 6

Testování a vyhodnocení

Platforma Android se vyskytuje na velmi rozmanitém množství zařízení s nejrůznější velikostí displeje, výkonem a vlastnostmi fotoaparátu. Proto je zapotřebí konečnou aplikaci vhodně otestovat a zhodnotit, jak z pohledu uživatele, pro které je aplikace cílená, tak i z hlediska programové funkčnosti.

6.1 Uživatelské testy

V rámci těchto testů byly vyhodnoceny názory koncových uživatelů na danou aplikaci. Testu se zúčastnilo 24 lidí ve věkové kategorii 15-35 let, kteří představují demografickou skupinu, pro kterou byla aplikace cílena.

Důležité je zhodnotit, jak snadno uživatel dokáže s programem zacházet, zda jsou mu některé pokyny nejasné a co by se dalo případně zlepšit. U uživatelů nebyla vyžadována předchozí znalost operačního systému Android.

Samotný test pak měl následující podobu. Každý uživatel byl obeznámen s účelem aplikace a ta mu byla následně nainstalována na jeho vlastní zařízení. Pokud jím nedisponoval, tak mu byl poskytnut testovací přístroj Xiaomi Mi2 s předinstalovanou aplikací.

Po spuštění dostal uživatel zadáno několik úkonů k vykonání:

- vyzkoušet automatickou záměnu obličejů ze zadní kamery pro pochopení základní funkčnosti
- přepnout na přední kameru
- nastavit detekovaný obličej na jeden z předdefinovaných
- pořídit snímek

Poté byl uživateli dán prostor pro vlastní použití pro získání vlastního dojmu z aplikace. Následně uživatel ohodnotil jednotlivé aspekty aplikace v dotazníku C.1, který shrnoval jeho dojmy.

Každou vlastnost bylo možné ohodnotit na celočíselné stupnici 6.1, kde hodnota 1 představuje nejhorší a 5 nejlepší možnou známku. Výsledky pak byly zprůměrovány a tato hodnota je znázorněna u každé odpovědi v dotazníku v příloze C. Poslední otázky pak dávají uživateli prostor se k aplikaci vyjádřit vlastními slovy, případně shrnout možná vylepšení.

Obrázek 6.1: Stupnice

Z výsledků testů je vidět, že uživatelé byli spokojení s jednoduchostí aplikace a ovládání. Výraznou výhodu měli v tomto případě uživatelé, kteří měli předchozí znalost znalost operačního systému Android vzhledem k tomu, že ovládání aplikace vychází z rozložení prvků aplikace `Fotoaparát` dodávané k OS Android od vydání verze Jelly Bean (4.1 - 4.3). Jediná věc, která občas dělala problém bylo odhalit možnost kliknout na detekovaný obličej.

Rychlost a dojem z výsledné záměny jsou dva aspekty, které spolu úzce souvisí a zlepšení jednoho z nich se nese ruku v ruce se zhoršením druhého, proto je důležité, že ač není výsledný dojem ani rychlost perfektní, tak je mezi těmito prvky rovnováha.

Znovupoužitelnost výsledné aplikace je ovlivněna, jak spokojeností s aplikací tak i faktem, zda by uživatel podobnou aplikaci vůbec vyhledával.

Zároveň během testu bylo podstatné sledovat uživatele při použití aplikace, přičemž stojí za povšimnutí, že uživatelé si často neuvědomují natočení hlavy do všech možných úhlů, přičemž pro správnou detekci je zapotřebí pohled čelem ze předu bez naklánění hlavy.

Připomínky uživatelů k chybějící funkčnosti navíc dávají podněty pro rozšíření stávající verze aplikace před jejím umístěním na Google Play.

6.2 Programové testování

Tato část testování se zabývala programovou funkčností aplikace a měřením rychlosti případně přesnosti detektoru a zároveň i jednotlivých částí celku. Tato analýza pak slouží k nalezení úzkého hrdla[1] při zpracování obrazu v reálném čase.

Pro testování byly využity 3 různá zařízení s rozdílnými parametry fotoaparátu a verzí operačního systému Android znázorněné v tabulce 6.1.

První test se zabývá nalezením optimálního nastavení detektoru a druhý slouží pro zobrazení rychlosti jednotlivých částí aplikace.

název	verze OS	rozlišení přední kamery	rozlišení zadní kamery
Xiaomi Mi2	4.1.1	720x1280	720x1280
Sony Ericsson Xperia Ray	2.3.3	480x640	480x864
Samsung Nexus S	4.2.2	480x640	480x720

Tabulka 6.1: Zařízení použita pro testování.

Parametry detektoru

Vstupní argumenty funkce `detectMultiscale()` popsané v sekci 5.4 podstatně ovlivňují funkčnost detektoru, ať už s ohledem na rychlost, tak přesnost. Opět je zapotřebí najít takové nastavení, aby poměr mezi těmito hodnotami byl vybalancovaný, a tedy aby aplikace například nebyla zbytečně přesná na úkor rychlosti.

Pro tyto účely byly na testovacích zařízeních 6.1 provedeny experimenty s třemi odlišnými konfiguracemi popsanými v tabulce 6.2, které se nejvíce blížili požadavkům.

[1] úzké hrdlo – nebo-li zpracování je tak rychlé, jako je rychlost nejpomalejší části

	scaleFactor	minNeighbors	minSize
Konfigurace 1	1.1	3	$48 * 48$ px
Konfigurace 2	1.2	4	$96 * 96$ px
Konfigurace 3	1.2	4	$72 * 72$ px

Tabulka 6.2: Vstupní parametry metody `detectMultiscale()`.

Samotný test byl proveden na 687 obličejích patřících 90 individuálním lidem[2].

U každého se předpokládalo natočení hlavy přímo proti kameře s co možná nejmenším nakloněním do strany nebo rotací po ose procházející hlavou a středem těla. Dále pak byly zahrnuty testovací subjekty s různou barvou kůže, za odlišného osvětlení.

Výsledky tohoto experimentu jsou pak znázorněny v tabulce 6.3.

	Úspěšnost [%]			**Rychlost [ms]**		
	Xiaomi Mi2	Xperia Ray	Nexus S	Xiaomi Mi2	Xperia Ray	Nexus S
Konfigurace 1	92.8	93.4	91.3	251	550	523
Konfigurace 2	75.1	77.2	77.6	49	66	59
Konfigurace 3	82.3	84	81.5	101	146	156

Tabulka 6.3: Rychlost jednotlivých částí aplikace.

Zatímco v případě konfigurace 1 byla detekována převážná většina obličejů v dalších konfiguracích tato úspěšnost silně klesala se zhoršujícími se světelnými podmínkami. Některé obličeje nebyly detekovány téměř nikdy ani při správných podmínkách a úhlu natočení.

S ohledem na rychlost není překvapivým výsledkem, že nejpomalejší byla nejpřesnější metoda a naopak. Bylo ovšem potřeba najít takové nastavení, které by pracovalo přiměřený čas na většině zařízení a jehož úspěšnost detektoru výrazně neklesá. Tomuto požadavku pak vyhovovala konfigurace 2.

Rychlost aplikace

Při zpracování každého snímku kamery prochází tyto obrazové data stále stejným koloběhem metod a pokud bude jedna z nich pomalá, poté bude zdržovat i každé další zpracování. Proto cílem této části programového testování bylo najít tu část aplikace, která zabírá největší množství výpočetního času.

Měření bylo provedeno na celkem 50-ti snímcích, při kterých bylo vždy detekováno různé množství obličejů. Test byl proveden na všech zařízeních 6.1 a přední i zadní kameře. Výsledná rychlost pak byla zprůměrována a znázorněná v tabulce 6.4.

Jednotlivé heslovité názvy metod poté znamenají následující.

Bitmap – převod vstupních dat kamery do RGB, vytvoření dvou bitmapových obrázků

Převod – vytvoření OpenCV matic ve formátu grayscale a HSV

Detekce – detekce obličejů

Celkem – celková rychlost zpracování jednoho snímku zahrnující všechny výše zmíněné části včetně dalších méně výpočetně náročných (uvolnění paměti a další)

[2] Zdroj: http://pics.psych.stir.ac.uk/2D_face_sets.htm

	Přední kamera [ms]			**Zadní kamera [ms]**		
	Xiaomi Mi2	Xperia Ray	Nexus S	Xiaomi Mi2	Xperia Ray	Nexus S
Bitmap	60	38	53	75	57	80
Převod	50	39	62	45	57	59
Detekce	45	35	67	57	68	75
Celkem	**170**	**125**	**190**	**185**	**185**	**220**

Tabulka 6.4: Rychlost jednotlivých částí aplikace.

Z výsledků testů je patrné, že žádná z částí nemá výrazně nižší rychlost než ostatní, proto i s vyšším výpočetním výkonem mobilního zařízení případně nižším rozlišením snímku je snazší dosáhnout vyšší rychlosti výsledné aplikace. To lze vidět i z toho, že pokud byl na zařízení použit přední fotoaparát, pak se zvedla i celková rychlost. Výjimkou je v tomto případě Xioami Mi2, které používalo stejné rozlišení pro přední i zadní fotoaparát. Tím byl tedy potvrzen původní návrh.

Průměrný čas při pořízení snímku není část, která by ovlivňovala chod aplikace pravidelně, proto je zde uvedena spíše pro úplnost v samostatné tabulce 6.5.

	Přední kamera [ms]			**Zadní kamera [ms]**		
	Xiaomi Mi2	Xperia Ray	Nexus S	Xiaomi Mi2	Xperia Ray	Nexus S
Pořízení snímku	350	329	187	230	242	356

Tabulka 6.5: Rychlost pořízení snímku.

Kapitola 7

Závěr

Cílem aplikace bylo vyzkoušet možnosti detekce obličejů na platformě Android v reálném čase a odlišit se tak od již existujících řešení pro záměnu tváří na mobilní platformy.

Na základě popsaných algoritmů pro detekci a sledování obličeje a několika dalších heuristik tak byla ve výsledku vytvořena vlastní metoda, která vyhovuje původním požadavkům. Samotná výsledná rychlost se na dostupných zařízeních různí v závislosti na výpočetním výkonu a použitém rozlišení fotoaparátu, ale obecně požadované rychlosti bylo dosaženo za cenu ztráty přesnosti detektoru.

Aplikace nabízí prostor pro další vylepšení v podobě úspěšnějšího detektoru či nalezení lepší metody pro extrakci tváře (např. pomocí detekovaných očí a úst navržených v kapitole 4.6). Zde byla největším problémem právě dosažená rychlost, což by bylo možné snížit vytvořením a natrénováním vlastní LBP kaskády. Dalším možným pokračováním v případě detekce je práce s natočeným obličejem z profilu nebo pod určitým úhlem vzhledem ke kameře.

Stejně tak lze dále pokračovat v práci na zlepšení samotné záměny obličejů, kdy v momentální verzi jsou znatelné barevné přechody mezi zaměněným obličejem a původní vrstvou kůže. Zde se nabízí, otestovat možnosti metod jako např. Poisson blending a zda jejich rychlost odpovídá požadavkům aplikace.

Literatura

[1] Bishop, C. M.: *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[2] Bradski, G. R.: Real Time Face and Object Tracking as a Component of a Perceptual User Interface. *Applications of Computer Vision, IEEE Workshop on*, ročník 0, 1998: str. 214, doi:http://doi.ieeecomputersociety.org/10.1109/ACV.1998.732882.

[3] Burkard, R.; Dell'Amico, M.; Martello, S.: *Assignment Problems*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2009, ISBN 0898716632, 9780898716634.

[4] Cheng, Y.: Mean Shift, Mode Seeking, and Clustering. *IEEE Trans. Pattern Anal. Mach. Intell.*, ročník 17, č. 8, 1995: s. 790–799.
URL http://dblp.uni-trier.de/db/journals/pami/pami17.html#Cheng95

[5] Freund, Y.; Schapire, R. E.: A Short Introduction to Boosting. In *In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, 1999, s. 1401–1406.

[6] Grewal, M.; Andrews, A.: Applications of Kalman Filtering in Aerospace 1960 to the Present [Historical Perspectives. *IEEE Control Systems Magazine*, ročník 30, č. 3, Červen 2010: s. 69–78, ISSN 0272-1708, doi:10.1109/mcs.2010.936465.
URL http://dx.doi.org/10.1109/mcs.2010.936465

[7] Julier, S. J.; Jeffrey; Uhlmann, K.: Unscented Filtering and Nonlinear Estimation. In *Proceedings of the IEEE*, 2004, s. 401–422.

[8] Kalman, R. E.: A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME ? Journal of Basic Engineering*, ročník 82 (Series D), 1960: s. 35–45.
URL http://www.cs.unc.edu/~{}welch/kalman/media/pdf/Kalman1960.pdf

[9] Kuhn, H. W.: The Hungarian Method for the Assignment Problem. *Naval Research Logistics Quarterly*, ročník 2, č. 1–2, March 1955: s. 83–97, doi:10.1002/nav.3800020109.

[10] Liao, S.; Zhu, X.; Lei, Z.; aj.: Learning Multi-scale Block Local Binary Patterns for Face Recognition. In *ICB*, *Lecture Notes in Computer Science*, ročník 4642, editace S.-W. Lee; S. Z. Li, Springer, 2007, ISBN 978-3-540-74548-8, s. 828–837.
URL http://dblp.uni-trier.de/db/conf/icb/icb2007.html#LiaoZLZL07

[11] Lienhart, R.; Maydt, J.: An Extended Set of Haar-Like Features for Rapid Object Detection. In *IEEE ICIP 2002*, 2002, s. 900–903.

[12] Pallett, P.; Link, S.; Lee, K.: New "golden"ratios for facial beauty. *Vision Research*, ročník 50, 2010: s. 149–154.

[13] Sedláček, M.: Evaluation of RGB and HSV Models in Human Faces Detection. Central European Seminar on Computer Graphics, Budmerice. In *IIIA.1-5 - Conference on Computer Systems and Technologies - CompSysTech?2004*, 2004, str. 125131.

[14] Swain, M.; Ballard, D.: Indexing via Color Histograms. In *Proc. Third International Conference on Computer Vision*, 1990, s. 390–393.

[15] Szeliski, R.: *Computer Vision: Algorithms and Applications*. New York, NY, USA: Springer-Verlag New York, Inc., první vydání, 2010, ISBN 1848829345, 9781848829343.

[16] Viola, P.; Jones, M.: Rapid object detection using a boosted cascade of simple features. 2001.

[17] Wilson, P. I.; Fernandez, D. J.: Facial feature detection using haar classifiers. *Journal of Computing Sciences in Colleges*, 2006: s. 127–133.

[18] Yang, M.; Kriegman, D. J.; Ahuja, N.: Detecting faces in images: A survey. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, ročník 24, č. 1, 2002.

Příloha A

Obsah CD

- README.txt – návod k instalaci
- BP-xskorn01.pdf – text práce
- `LaTex/`
  – zdrojové kódy pro překlad textu práce
- `Android/`
  – zdrojové kódy Android projektu
- `docs/`
  – dokumentace programové části
- `OpenCV2.4.8/`
  – knihovna OpenCV verze 2.4.8 použita při vývoji a překladu aplikace

Příloha B

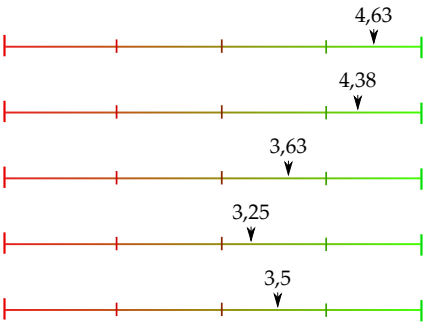
Snímky z aplikace

Obrázek B.1: Faceswap s použitím zadní kamery a detekovaných obličejů.

Obrázek B.2: Faceswap s použitím zadní kamery a detekovaných obličejů.

Příloha C

Dotazník



Co Vás nejvíce zaujalo?	Předdefinované obličeje, Originalita, Faceswap pěti obličejů zároveň.
Co se nejvíce nelíbilo?	8FPS, Citlivost detektoru.
Jaká funkce Vám chybí?	Galerie, Sdílení fotky na sociální síti.

Tabulka C.1: Dotazník k uživatelským testům s vyhodnocením.