VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2016 Bc. Luděk Novotný

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

MULTIMEDIÁLNÍ PŘEHRÁVAČ OVLÁDANÝ GESTY LIDSKÉ RUKY
MULTIMEDIA PLAYER CONTROLLED BY HAND GESTURES.

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. Luděk Novotný

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Adam Olejár

BRNO 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**
Ústav telekomunikací

Student: Bc. Luděk Novotný ***ID:*** 146918
Ročník: 2 ***Akademický rok:*** 2015/16

NÁZEV TÉMATU:

Multimediální přehrávač ovládaný gesty lidské ruky

POKYNY PRO VYPRACOVÁNÍ:

Prostudujte současné algoritmy pro detekci lidské ruky a gest, které mohou být takovými algoritmy detekovány a rozlišovány. Na základě tohoto rozboru vyberte a implementujte nejvhodnější způsob detekce gest pro ovládání multimediálního přehrávače.
Vytvořte aplikaci, která umožní přehrávat multimediální obsah nebo prezentace a bude ovládaná definovanými gesty. Doporučeným nástrojem pro implementaci jsou knihovny OpenCV a prostředí MS VisualC ++.

DOPORUČENÁ LITERATURA:

[1] BURGER, Wilhelm; BURGE, Mark J. Principles of Digital Image Processing: Fundamental

Techniques. Londýn : Springer, 2009. 272 s. ISBN 978-1848001909.

[2] GONZALEZ R.C., WOODS R.E.: Digital Image Processing. New Jersey: Prentice-Hall, 2002.

[3] PRATA S.: Mistrovství v C++, Computer Press, Brno 2004, ISBN 80-251-0098-7.

Termín zadání: 1.2.2016 ***Termín odevzdání:*** 25.5.2016

Vedoucí práce: Ing. Adam Olejár
Konzultant diplomové práce:

doc. Ing. Jiří Mišurec, CSc., *předseda oborové rady*

ABSTRAKT

Tato diplomová práce se zabývá návrhem detektoru gest lidské ruky. Teoretická část probírá obvyklé současné metody segmentace a detekce. Byly vybrány ty, které by byly nejvhodnější pro ovládání multimediálního přehrávače. Navržený algoritmus je následně popsán pomocí vývojových diagramů. Detektor je napsán v jazyce Java s využitím knihovny OpenCV. K detekci gest je použita segmentační metoda odečítání pozadí a aproximace konvexním mnohoúhelníkem. Detektor je schopen rozlišit šest gest a sledovat střed dlaně. Nakonec byl detektor propojen s navrženým a implementovaným prohlížečem dokumentů, obrázků a multimediálních souborů.

KLÍČOVÁ SLOVA

OpenCV, Java, detekce gest, počítačové vidění, webová kamera, segmentace, zpracování obrazu, prohlížeč souborů

ABSTRACT

This diploma thesis deals with the hand gesture detector. Theoretical part is devoted to the discussion of current segmentation and detection methods. They were chosen those, which would work best as the controlling methods of multimedia player. The proposed algorithm is described with the help of flowcharts. Detector is written in Java language with the use of OpenCV library. Background subtraction segmentation method and convex hull approximation are used for gesture detection. Detector can distinguish between six gestures and track the center of palm. Finally, the detector was connected with proposed and implemented document, image and multimedia files browser.

KEYWORDS

OpenCV, Java, gesture detection, computer vision, webcam, segmentation, image processing, file browser

NOVOTNÝ, Luděk *Multimediální přehrávač ovládaný gesty lidské ruky*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2016. 44 s. Vedoucí práce byl Ing. Adam Olejár,

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Multimediální přehrávač ovládaný gesty lidské ruky“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a 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/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Adamu Olejárovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

podpis autora

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic

http://www.six.feec.vutbr.cz

PODĚKOVÁNÍ

Výzkum popsaný v této diplomové práci byl realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

Brno . . . . . .

. . . . . .

podpis autora

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OBSAH

Úvod **10**

1 Metody detekce gest **11**
1.1 Získání dat z kamery . . . 11
1.2 Segmentace ruky z obrazu . . . 11
1.2.1 Prahování obrazu . . . 12
1.2.2 Detekce kůže . . . 13
1.2.3 Odečítání pozadí . . . 14
1.3 Morfologické operace . . . 16
1.3.1 Eroze . . . 17
1.3.2 Dilatace . . . 17
1.4 Hledání kontur . . . 18
1.5 Detekce gesta . . . 19
1.5.1 Konvexní mnohoúhelník . . . 19
1.5.2 K zakřivení . . . 20
1.5.3 Porovnání tvarů . . . 20

2 Praktická realizace detektoru **22**
2.1 Implementace detektoru . . . 22
2.1.1 Segmentace . . . 22
2.1.2 Získání kontury . . . 24
2.1.3 Detekce počtu prstů . . . 25

3 Multimediální prohlížeč **30**
3.1 Architektura programu . . . 30
3.1.1 Bridge . . . 30
3.1.2 BridgeComponent . . . 30
3.2 Hlavní okno . . . 31
3.2.1 MainWindowController . . . 31
3.2.2 Ovládací rozhraní hlavního okna . . . 32
3.3 Obrázky . . . 34
3.3.1 ImageTController . . . 34
3.4 Multimédia . . . 35
3.4.1 MediaTController . . . 35
3.4.2 Ovládací rozhraní multimédií . . . 35
3.5 PDF dokumenty . . . 36
3.5.1 PdfTController . . . 36

3.5.2 Ovládací rozhraní prohlížeče dokumentů 36
3.6 Detekce gest 36
3.6.1 Volání setPalmCenter 37
3.6.2 Volání setFingerCount 38
3.6.3 Generování akcí 38
3.6.4 Vybraná gesta 38
3.7 Zhodnocení rozlišovacích schopností detektoru 38
3.7.1 Podmínky 39
3.7.2 Denní světlo 39
3.7.3 Špatné světelné podmínky 40
3.7.4 Proměnlivé světelné podmínky 40
3.7.5 Více kontur v záběru 41

4 Závěr **42**

Literatura **43**

A Obsah přílohy **44**

SEZNAM OBRÁZKŮ

1.1 Fáze metody prahování obrazu 12
1.2 Prahování na intenzitě 127 13
1.3 Graficky znázorněný model HSV 14
1.4 Detekované popředí při změně osvětlení (statický model) 15
1.5 Detekované popředí dynamického modelu 16
1.6 Obraz před a po erozi 17
1.7 Obraz před a po dilataci 18
1.8 Kontura ruky 18
1.9 Konvexní mnohoúhelník a defekty konvexity 19
1.10 K zakřivení 20
1.11 Šablona pro porovnání tvarů 21
2.1 Použití třídy BackgroundSubtractorMOG2 23
2.2 Hledání největší kontury 24
2.3 Nová kontura po aproximace mnohoúhelníkem 25
2.4 Filtrování defektů 26
2.5 Redukce shluků bodů konvexního mnohoúhelníku 27
2.6 Počítání prstů 28
2.7 Pěst 29
3.1 Blokové schéma architektury prohlížeče 31
3.2 Hlavní okno prohlížeče 33
3.3 Otevřený obrázek 34
3.4 Ovládací rozhraní multimédií 35
3.5 Ovládací rozhraní prohlížeče dokumentů 37
3.6 Segmentace a detekce za denního světla 39
3.7 Segmentace a detekce v neosvětlené místnosti 40
3.8 Segmentace a detekce po automatickém přeučení 41
3.9 Hlava v záběru kamery 41

ÚVOD

Nejpoužívanější vstupní periferií počítače je klávesnice a myš. Existují ovšem případy, kdy se nejedná o nejvhodnější zařízení. Uživatel mohl například přijít o končetinu, nebo je neschopen pohybu rukou. V takovém případě je nutné použít úplně jiný způsob ovládání. V budoucnu také můžeme očekávat nástup systémů virtuální reality, kdy helma s displejem zakrývá výhled a tudíž je používání klávesnice a myši nepohodlné a obtížné. Ale i při běžném používání elektronických zařízení je možné využívat výhod alternativních způsobů ovládání.

Jednou z populárních alternativních metod je ovládání zařízení pomocí gest lidské ruky. Webové kamery jsou ve společnosti velmi rozšířenou vstupní periferií. Nacházejí se běžně ve smartphonech, laptopech, dokonce jsou integrovány i do některých monitorů. Samostnatné kamery se prodávají v různých cenových relacích a i ty obyčejné je možné využít k detekci gest.

Tato práce se zabývá návrhem multimediálního přehrávače, který bude ovládán pomocí gest lidské ruky. Vstupním zařízením bude obyčejná webová kamera, použitým programovacícm jazykem pak Java. Využita bude knihovna OpenCV, která obsahuje velké množství algoritmů, co se používají pro počítačové vidění. Diplomová práce je konkrétně zacílená na studii současných algoritmů a vytvoření aplikace, která bude v reálném čase snímat lidskou ruku, detekovat předem definovaná gesta a zobrazovat různé druhy multimediálních souborů.

Práce je rozdělena do třech částí. První popisuje různé metody, které se k dekci gest používají. Diskutuje také jejich výhody, nevýhody a rozlišovací možnosti. Na základě tohoto rozboru je vybrán algoritmus, který bude použit pro praktickou implementaci.

Druhá část se zabývá praktickým návrhem detektoru. Popisuje základní části detektoru, jejich funkce a jaké jsou jejich výstupy.

Třetí část popisuje architekturu prohlížeče, jeho ovládací rozhraní, způsoby načítání různých typů multimediálních souborů a diskutuje rozlišovací možnosti detekoru za různých světělných podmínek.

Spustitelné jar soubory a zdrojový kód (ve formátu projektu NetbBeans) se nacházejí v příloze.

1 METODY DETEKCE GEST

Tato práce se zabývá detekcí gest lidské ruky v reálném čase pomocí standardní webové kamery. Existuje velké množství různých technik, jak z obrazu získat informace o ruce. Některé se zaměřují na získání specifického gesta, jiné jsou více obecné. V této teoretické části bude vyjmenováno několik nejpoužívanějších metod segmentace a detekce. Nakonec budou vybrány ty, které jsou pro ovládání programu nejvhodnější.

Algoritmy získávající informace o provedených gestech se dají velmi obecně rozdělit do těchto tří částí:

1. Získání dat z kamery
2. Segmentace ruky z obrazu
3. Detekce gesta

1.1 Získání dat z kamery

Ještě než je možné započít segmentaci, je nutné z webové kamery získat aktuální snímek. Ve většině případech není třeba komunikovat s kamerou přímo pomocí připojeného rozhraní (nejčastěji USB).

Existuje velké množství různých knihoven, které tuto práci zjednodušují. Výběr té nejvhodnější záleží na použitém programovacím jazyku. Tato diplomová práce využívá knihovnu OpenCV, která už má implementovanou metodu pro získání aktuálního snímku. Detaily o získávání dat z kamery se proto nacházejí v sekci 2.1.

Výstupem je standardně matice vektorů, jejíž velikost odpovídá rozměrům získaného snímku. Každý vektor pak obsahuje tři hodnoty reprezentující červenou, zelenou a modrou barvu barevného modelu RGB. Tyto hodnoty mají většinou 256 úrovní, což odpovídá 24 bitům na celý vektor. Pokud se jedná o šedotónovou kameru, výstupem je matice skalárů. Konkrétní formát hodnot však závisí na použité knihovně.

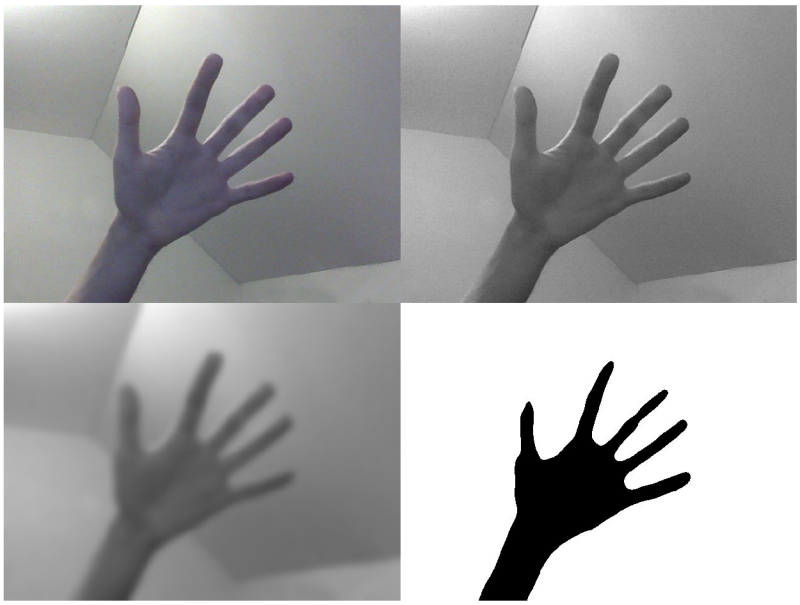
1.2 Segmentace ruky z obrazu

Nikdy nemáme zaručeno, že se na získaném snímku z kamery bude nacházet pouze ruka. V záznamu bude pravděpodobně vidět samotný uživatel či jeho část a za ním pak pozadí. I pozadí může být komplexní, nemusí se jednat o bílou zeď, na které bude lidská ruka vynikat. Nejdříve tedy musíme zjistit, co se na snímku nachází.

Segmentace obrazu je metoda či skupina metod, co slouží k automatickému rozdělení snímku na části, které mají určité společné vlastnosti. V našem případě to

bude rozdělení na část s rukou (popředí) a pozadím. Na segmentaci obrazu leží základy počítačového vidění a v této teoretické sekci budou probrány ty, co jsou vhodné pro následující detekci gest.

1.2.1 Prahování obrazu



Obr. 1.1: Fáze metody prahování obrazu

Základní metodou segmentace je prahování obrazu. Spočívá v tom, že vstupní obraz je nejdříve převeden do odstínů šedi, následně je na něj aplikován Gaussův filtr (čímž se zjemní okraje nalezeného obrysu, ale tento krok není podmínkou) a nakonec dojde k prahování. Práh binárně rozdělí obrázek podle intenzity pixelu na část s rukou a pozadím. Jednotlivé kroky jsou demonstrovány na obrázku 1.1.

Předpokladem pro přijatelný výsledek je, že pixely pozadí musí mít výrazně odlišnou intenzitu jasu než pixely ruky. Také je nutné vhodně nastavit práh. Na obrázku 1.1 je nastaven na intenzitu 105. Na obrázku 1.2 je nastaven na 127, čímž se pod práh dostaly i některé tmavé části snímku. Ruku tak nelze spolehlivě oddělit od pozadí. Úpravami histogramu by bylo možné docílit lepších výsledků, ale to jen pořád za předpokladu, že pozadí bude jednolité.

V praxi nelze počítat s tím, že okolí uživatele bude vždy homogenní. Komplexnost pozadí a změny osvětlení mají tedy velmi silný vliv na kvalitu segmentace. Také je nutné uživatele vyzvat, aby sám nastavil vhodně práh. Tento požadavek je možné vypustit, pokud použijeme automatické hledání prahu. Ten je z počátku nastaven na výchozí vybranou úroveň a následně je postupně zvyšován, dokud v obrazu není nalezena kontura, která vlastnostmi připomíná ruku.

Obr. 1.2: Prahování na intenzitě 127

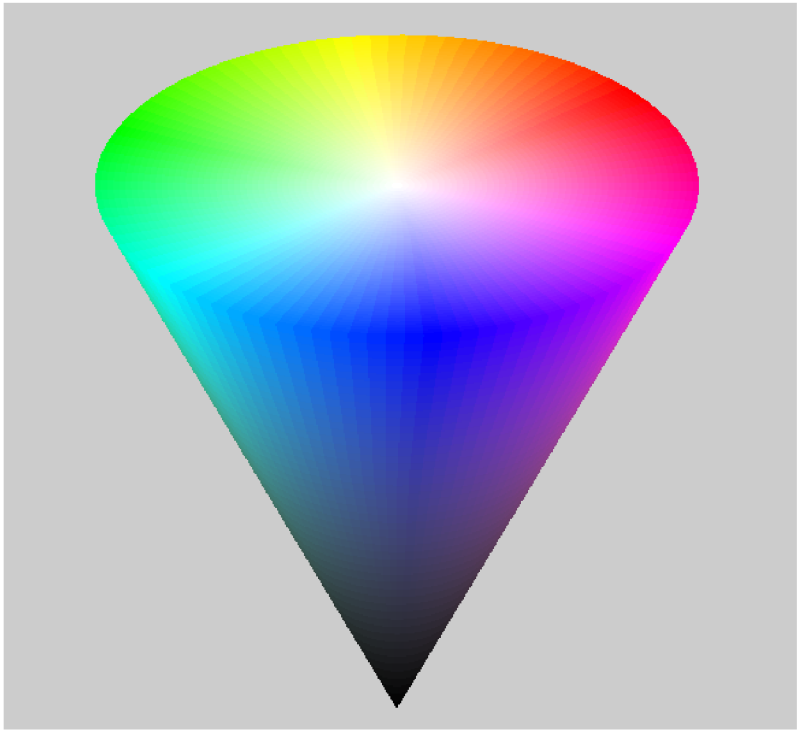
Zautomatizované hledání přenáší zodpovědnost za funkčnost z uživatele na program, ale i přesto si neporadí s pozadím, ve kterém je velké množství detailů. Výhodou metody je jednoduchost její implementace. Prahování obrazu tedy nebude použito.

1.2.2 Detekce kůže

Tento typ segmentace je založen na hledání pixelů, co barvou odpovídají kůži. Metoda je podobná prahování, až na rozdíl, že pracujeme s barevným snímkem v barevném modelu HSV. Ten odpovídá lidskému vnímání barev mnohem více než RGB, takže můžeme přesněji specifikovat meze hledaných barev, které odpovídají kůži.

Barvu v modelu HSV reprezentují tři hodnoty. Odstín (hue), saturace (saturation) a hodnota jasu (value). Odstín popisuje vnímaný barevný tón, saturace čistotu barvy (příměsy jiných barev) a hodnota jasu pak množství bílého světla. Nejčastějším grafickým znázorněním modelu je kužel, kdy úhel na kruhovém dně reprezentuje odstín, vzdálenost od středu kruhu saturaci a vzdálenost od podstavy k vrcholu hodnotu jasu, jak můžeme vidět na obrázku 1.3.

Podobně jako při prahování musíme zvolit hodnoty, při kterých budeme považovat pixel za ruku. Tentokrát ovšem vybereme i horní mez, takže budeme hledat jen ve specifikovaném intervalu barev. Pokud do tohoto intervalu pixel spadne, je považován za kůži. Z toho ovšem vychází několik nevýhod. Falešně mohou být detekovány i jiné části těla, jako například hlava, případně předměty, které barvou připomínají kůži. Do pevně nastavených hranic intervalu se navíc nemusí vejít lidé



Obr. 1.3: Graficky znázorněný model HSV

s odstínem kůže, na který program není připraven. To lze vyřešit zvětšením šířky intervalu, což ale povede k většímu množství falešných detekcí v obrazu.

Řešením by bylo vyzvat uživatele, aby nejdříve ruku umístil do specifikovaného místa před kameru, z něhož by program sejmul několik vzorků a vytvořil z nich vhodné meze sám. Z uživatelovy strany tak vyžadujeme aktivitu, která by užitím jiné metody segmentace nebyla potřeba. I detekce kůže má problémy s proměnlivým nasvícením scény.

1.2.3 Odečítání pozadí

Tato metoda spočívá v rozeznání statického pozadí. Cokoliv, co na něj nepatří, je pak považováno za popředí. V případě detekce gest by byla jako popředí detekována lidská ruka. V nejjednodušší implementaci je první snímek uložen a převeden do odstínů šedi. Je požadováno, aby na tomto modelu pozadí nebylo nic, co by mělo být detekováno jako popředí. Všechny následující snímky jsou též převedeny do odstínu šedi a následně se pixel po pixelu porovnává jejich podobnost. Pokud je podobnost nízká, pixel je považován za popředí.

OpenCV obsahuje několik implementovaných algoritmů pod zkratkami MOG, MOG2, GMG a KNN, přičemž v základu jsou dostupné MOG2 a KNN. Algoritmus MOG2 vychází z práce An Improved Adaptive Background Mixture Model for Re-

altime Tracking with Shadow Detection. [3] Každý pixel je modelován do Gaussova rozdělení. Každý snímek je započítán do modelu a barvy, které na obraze zůstávají nejdéle, mají vyšší pravděpodobnost, že patří do pozadí. MOG2 je také schopen detekovat stíny a oproti ostatním metodám je velmi rychlý. [1]

Oproti segmentacím založených na prahování má odečítání pozadí výhodu v tom, že spolehlivě funguje i s komplexním pozadím, ve kterém je mnoho detailů. Naopak je velmi citlivé na změny osvícení (a obecně změny okolí). Na obrázku 1.4 vidíme, že ruka je detekována i za špatných světelných podmínek, ale jakmile došlo k jejich změně (zapnutí světla), není ji už vůbec možné rozlišit.

Obr. 1.4: Detekované popředí při změně osvětlení (statický model)

Ve výše uvedeném příkladu byl použit statický model, který byl nejdříve natrénován na sekvenci snímků bez popředí. Následně model nebyl obměňován, díky čemuž velmi špatně reaguje na změny osvětlení. Dynamický model se každým snímkem aktualizuje a pružně tak reaguje na změny osvětlení, detekované popředí ovšem obsahuje velké množství duchů a po určité době dané velikostí historie snímků také splyne s pozadím (obr. 1.5).

V praktickém užití nejsou nevýhody statického modelu příliš závažné. Pokud je v místnosti s webovou kamerou okno a mísnost osvětluje přirozené světlo, nemělo by dojít k takové změně podmínek, jako při zapnutí zářivky. Na skokové změny navíc můžeme program adaptovat tak, že jakmile zjistí v obrazu určité procento pixelů s maximální intenzitou, vyzve uživatele, aby ruku stáhl z obrazu a následně si vytvoří nový model pozadí.

Obr. 1.5: Detekované popředí dynamického modelu

Druhou potenciální nevýhodou je fakt, že tato segmentační metoda striktně vyžaduje statické umístění kamery. Jakmile dojde k jejímu přemístění, detekované popředí bude vypadat podobně, jako na obrázku 1.4. Navíc může dojít k nesprávné detekci, když se v záběru kamery objeví nechtěný pohyb. I tyto nevýhody ovšem nejsou příliš vážné. S webovými kamerami se často nepohybuje a pokud není uživatel v nějaké společenské místnosti, i v pozadí se nejspíše nebude nic pohybovat. Menší předměty (jako například větrák) lze eliminovat prahem velikosti nalezených kontur.

Odečítání pozadí je tedy efektivní a za specifikovaných podmínek i spolehlivá metoda, jak z obrazu separovat lidskou ruku. V detektoru bude použita třída BackgroundSubtractorMOG2.

1.3 Morfologické operace

Morfologické operace používáme v počítačovém vidění pro předzpracování obrazu před samotnou detekcí. Dříve se používaly pro zpracování binárních obrazů, dnes však našly uplatnění i pro práci se šedotónovými obrazy. Na obraz se pohlíží jako na množinu bodů, přičemž ho vyšetřujeme pomocí druhé množiny (kernelu). Jelikož má menší velikost, posouvá se v obrazu, takže morfologické operace určitým způsobem připomínají konvoluci. I v kernelu se také nachází referenční prvek, pod kterým zapisujeme výsledek operace. Používáme ovšem množinové operace jako průnik $\cap$ či sjednocení $\cup$. Mezi elementární morfologické operace patří eroze a dilatace.

1.3.1 Eroze

Eroze v obrazu zmenšuje objekty, posouvá jejich hrany směrem dovnitř a je tedy vhodná pro odstranění šumu či separaci tvarů, viz obr. 1.6. Na levé straně vidíme výstup ze segmentace metodou odečítání pozadí. V pozadí je vidět šum a prsty jsou tak blízko u sebe, že skoro splývají v jeden tvar. Po několika aplikacích eroze se separovaly a byl také odstraněn veškerý šum.

Obr. 1.6: Obraz před a po erozi

Při erozi se zjišťuje, zda jsou pod celým kernelem na analyzovaném obrazu jedničky. Pokud ano, na referenční prvek výstupního obrazu se zapíše jedna. Následně se kernel posune. Následující rovnice popisuje tuto situaci. [5] Kernel H se nachází nad souřadnicí obrazu x. X je vstupní a Y výstupní obraz:

$$Y = E_H(X) = \{x | H_x \subseteq X\}\,. \tag{1.1}$$

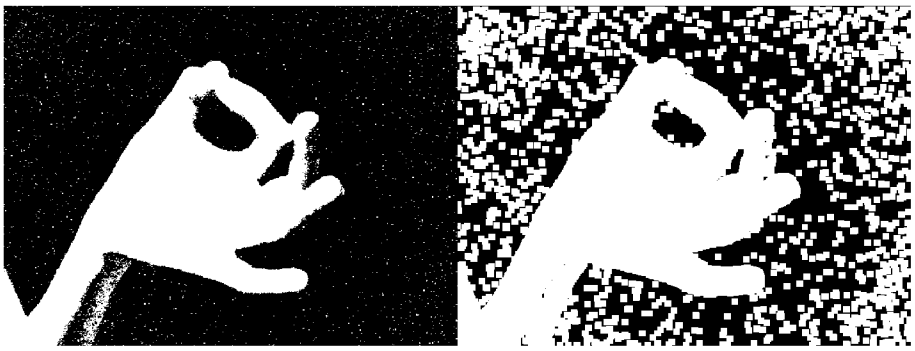
1.3.2 Dilatace

Dilatace má opačnou funkci než eroze. Objekty zvětšuje a roztahuje jejich hrany. Slouží k vyplnění děr a mezer v obrazu. Často ji používáme, abychom navrátili objektům zpátky tvar, který jsme jim ubrali erozí. Této posloupnosti se říká otevírání. Uzavírání je naopak dilatace následovaná erozí. V popředí se uzavřou malé mezery a díry a pak se erozí vrátí tvaru původní velikost.

Obrázek 1.7 ukazuje výstup ze segmentace metodou odečítání po několika dilatacích. Šum byl výrazně zesílen a mezery mezi prsty se začaly uzavírat.

Při dilataci se zjišťuje, zda se alespoň pod jedním bodem kernelu nachází jednička. Pokud ano, na referenční prvek výstupního obrazu se zapíše jedna. [5] Následně se kernel posune. Následující rovnice popisuje tuto situaci:

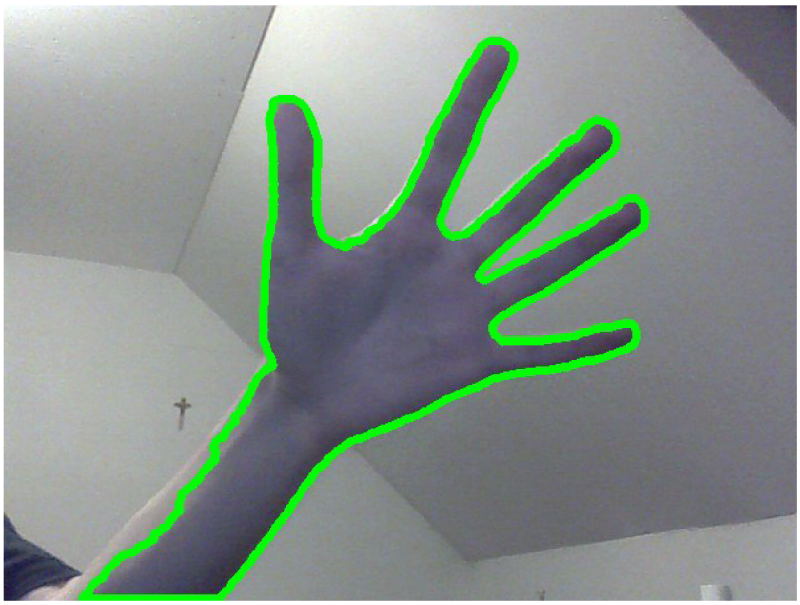
$$Y = D_H(X) = \{x | H_x \cap X \neq \oslash\}\,. \tag{1.2}$$



Obr. 1.7: Obraz před a po dilataci

1.4 Hledání kontur

Kontury nám mohou mnoho vypovědět o obsahu snímku. Můžeme pomocí nich zjistit, kolik objektů na obrazu je, jak jsou velké nebo jaký mají tvar. Jsou reprezentovány řadou seřazených bodů, co se nacházejí na obrysech plochy. Na obrázku 1.8 je vykreslena kontura ruky, která je tvořena 1080 body, takže tvoří takřka souvislou čáru.



Obr. 1.8: Kontura ruky

Algoritmy pro získání kontur jsou už poměrně staré (ne však zastaralé). OpenCV využívá metodu, kterou Suzuki S. a Abe K. popsali v roce 1985 v práci Topological Structural Analysis of Digitized Binary Images by Border Following. [4]

1.5 Detekce gesta

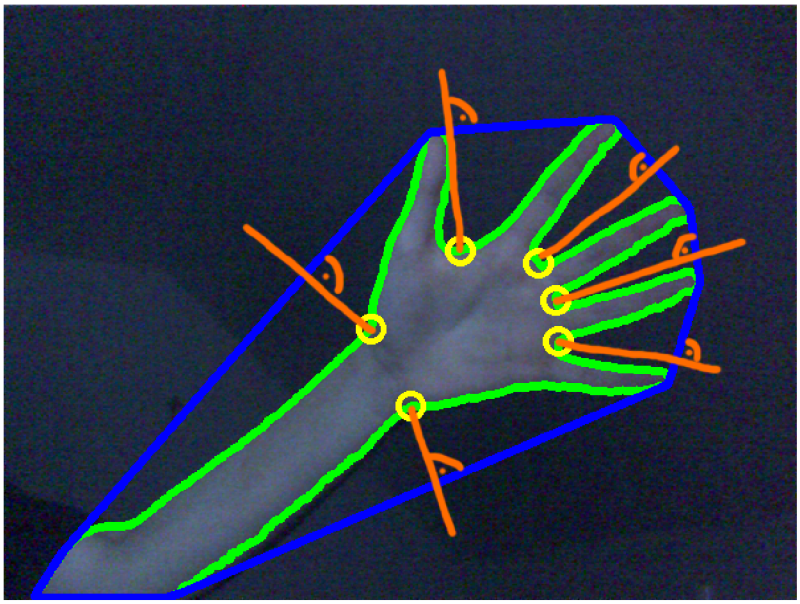
Jakmile je oblast ruky získána, lze z ní různými způsoby získat informace o provedeném gestu či poloze dlaně.

1.5.1 Konvexní mnohoúhelník

Tato metoda je založena na počítání natažených prstů pomocí konvexního mnohoúhelníku, který těsně obklopuje nalezenou konturu. Z toho vyplývá, že metoda dokáže v základu rozeznat 6 gest. S určitými modifikacemi jich ovšem může být více.

Konvexní mnohoúhelník je takový, jehož vnitřní úhly jsou menší než 180 °. V případě kontur hledáme konvexní obal, který pojme všechny body kontury. OpenCV obsahuje metodu convexHull, která implementuje algoritmus popsaný v práci Finding the Convex Hull of a Simple Polygon se složitostí O(N logN). [2]

Z konvexního mnohoúhelníku pak lze zjistit počet natažených prstů (které jsou ve vztahu s počtem vrcholů). Pomocí defektů konvexity je také možné přesněji zjistit, jaké prsty jsou nataženy. Defekty se dají nalézt poměrně lehce. Z každého bodu kontury povedeme kolmici na hranu konvexního mnohoúhelníku, viz obr. 1.9. Ten nejvzdálenější bod označíme za defekt. Nový defekt hledáme vždy, když vzdálenost prověřovaného bodu kontury od hrany mnohoúhelníku přesáhne určitý práh. Takto můžeme vyfiltrovat všechny nechtěné defekty.

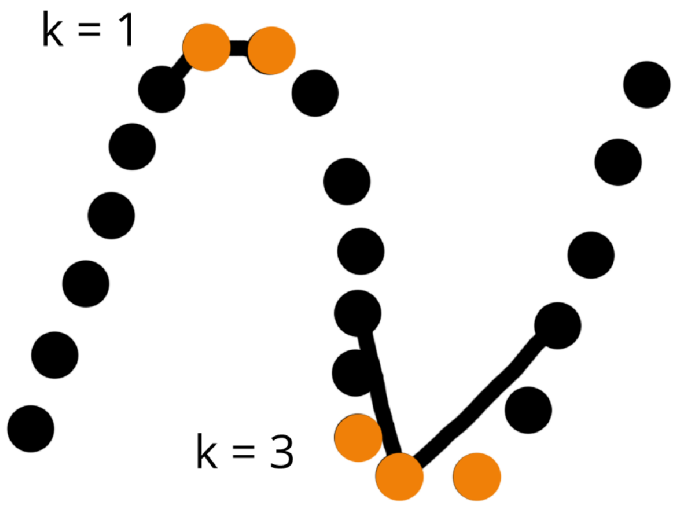


Obr. 1.9: Konvexní mnohoúhelník a defekty konvexity

Metoda počítání prstů s pomocí konvexního mnohoúhelníku je poměrně přesná a umožňuje nám detekovat 6 či více gest. Zároveň je možné pomocí defektů odhadnout střed dlaně a ten sledovat. Pohyb lze následně aplikovat na kurzor a pohybovat tak kurzorem. Z těchto důvodů je metoda pro ovládání programu velmi vhodná a bude do detektoru implementována.

1.5.2 K zakřivení

Jinou možností, jak detekovat pozici špiček prstů, je využití pouhé kontury. Ta se prochází bod po bodu a počítá se úhel mezi přímkami, které vznikou propojením analyzovaného bodu se sousedy, jak je vidět na obrázku 1.10. Hodnota k v tomto případě značí k-tého souseda. Pokud úhel přesáhne námi definovanou hranici, bod označíme. Takto označené body se budou nacházet na špičkách prstů a na defektech konvexity. Na rozdíl od předchozí metody nejsou mezery mezi prsty rozlišeny od špiček, takže je musíme dále filtrovat například na základě vzdálenosti od středu dlaně.



Obr. 1.10: K zakřivení

1.5.3 Porovnání tvarů

OpenCV nám poskytuje funkci matchShapes, která je postavena na Hu momentech. Jedná se o fotometrické příznaky založené na regionech. Jsou invariantní vůči posuvu, změně měřítka a natočení.

Funcke matchShapes na vstup požaduje dvě kontury určené k porovnání. Dodána byla binární maska popředí z webovékamery a předpřipravená šablona 1.11. Zvolena byla druhá metoda pro výpočet podobnosti, která odpovídá rovnici:

$$I_2(A,B) = \sum_{i=1\ldots7} \left| m_i^A - m_i^B \right|, \tag{1.3}$$

kde

$$m_i^A = sign(h_i^A) \cdot log(h_i^A) \tag{1.4}$$

a

$$m_i^B = sign(h_i^B) \cdot log(h_i^B). \tag{1.5}$$

h_i^A a h_i^B jsou Hu momenty náležející objektu A a objektu B.

Jelikož jsou momenty invariantní na posuv, změnu měřítka a natočení, byly užity k detekování gest ze šablony 1.11. Rovnice 1.3 byla vybrána, protože vykazovala nejlepší výsledky. Nejlépe se povedlo detekovat otevřenou dlaň. Bohužel metoda vykazovala velmi velké množství falešných detekcí a zaměňovala vztyčený ukazováček za palec a pěst. Také nebylo možné detekovat polohu dlaně, takže pokud by měla být použita i pro posuv myši, musela by být zkombinována s jinou.

Porovnání tvarů tedy implementováno v detektoru je, kód byl nicméně zakomentován a používá se počítání prstů pomocí konvexního mnohoúhelníku.

Obr. 1.11: Šablona pro porovnání tvarů

2 PRAKTICKÁ REALIZACE DETEKTORU

V následující kapitole je popsán navržený detektor, který jako metodu segmentace využívá odečítání pozadí a gesta detekuje pomocí počítání prstů. Jedná o poměrně rozšířený algoritmus, který například využívá práce [6]. Uvedeny jsou i ukázky kódu, veškerý zdrojový kód se pak nachází v příloze. V závěru jsou diskutovány jeho rozeznávací schopnosti a slabiny.

2.1 Implementace detektoru

Detektor je implementován v programovacím jazyku Java. Využita byla knihovna OpenCV, grafická aplikace je naprogramována s pomocí knihovny JavaFX. Zdrojový kód je ve formátu projektu NetBeansIDE 8.1.

2.1.1 Segmentace

Detektor se nachází ve třídě Detector. Implementuje rozhraní IDetector, které předepisuje, jaké metody si musí přepsat. Jsou to metody pro zapnutí a vypnutí detektoru a getter pro zjištění, zda je detektor aktivní.

Konstruktor třídy obsahuje počáteční inicializace některých objektů a načítají se zde vzorové obrazy vybraných gest, jejichž kontury se později porovnávají s konturou nalezenou v zorném poli webové kamery. Metoda porovnávání tvarů je popsána v kapitole 1.5.3, kde byla označena díky vysokému množství falešných detekcí za nevhodnou. Proto je tato část kódu v konstruktoru zakomentovaná. Dojde k vytvoření objektu typu BackgroundSubtractorMOG2, VideoCapture a několika matic a polí pro ukládání dat.

Voláním metody start() zahájí detektor svou činnost. Pomocí VideoCapture se otevře spojení s webovou kamerou. Nakonec se vytvoří objekt typu ScheduledExecutorService, který slouží ke spouštění úloh v novém vlákně. Detektor je nutné spouštět v jiném vlákně než kde běží samotná aplikace, jinak by byl její běh často blokován. Poslední řádek kódu:

```
executor.scheduleAtFixedRate(detector, 0, 33, TimeUnit.MILLISECONDS);
```

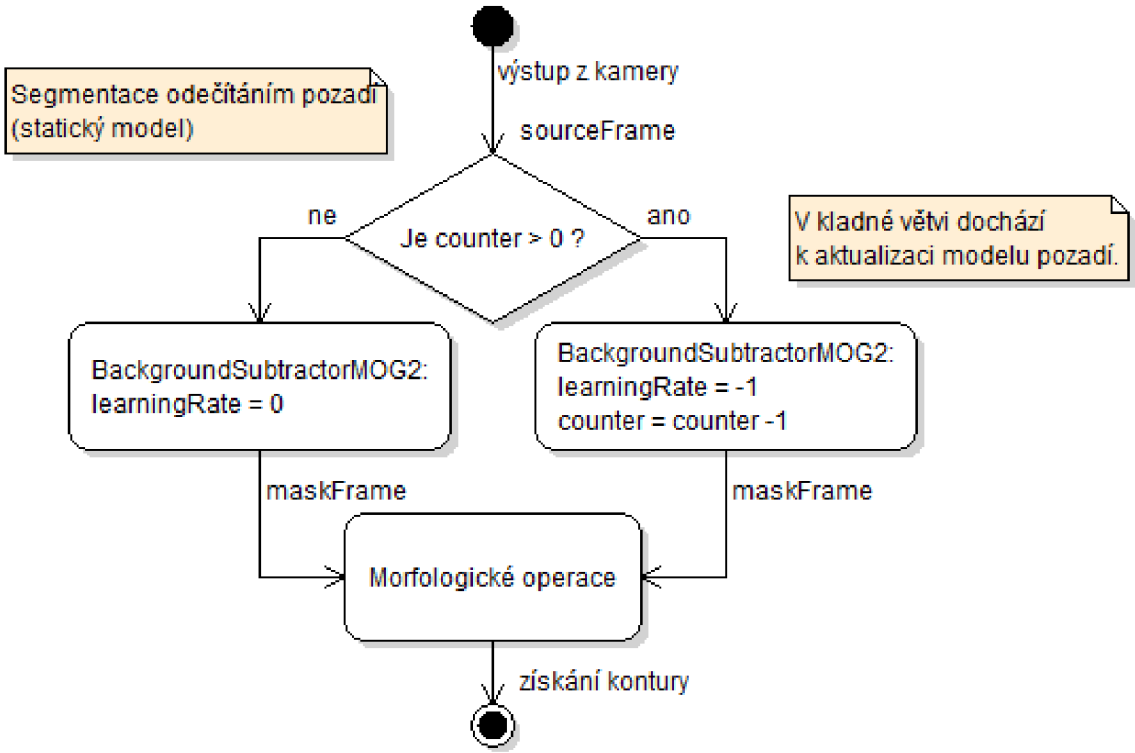
nastaví automatické spouštění úlohy každých 33 milisekund, což zhruba odpovídá frekvenci 30 Hz. Vybraný objekt pro spouštění se nazývá detector.

Jelikož ScheduledExecutorService požaduje objekt implementující rozhraní Runnable, aby nad ním v určitých časových intervalech mohl volat metodu run, je nutné takovou třídu vytvořit. Jelikož se jedná o jádro, které je pevně spjato se třídou Detector, je přímo v metodě start() vytvořena anonymní třída implementující Runnable.

Metoda run() se tedy bude spouštět každých 33 milisekund a obsahuje samotnou segmentaci a detekci.

```
Runnable detector = new Runnable() {
...
```

Výstupní obraz z kamery je uložen do matice sourceFrame. Následně je použit objekt typu BackgroundSubtractorMOG2 (obr. 2.1). Metoda apply vyžaduje vstupní matici a výstupní matici (maskFrame), kam je uloženo separované popředí. Volitelným parametrem je integer learningRate. Pokud není nastaven, implicitně se použije hodnota -1. V takovém případě objekt každým přijatým snímkem aktualizuje model pozadí a uloží si ho do historie (dynamický model).



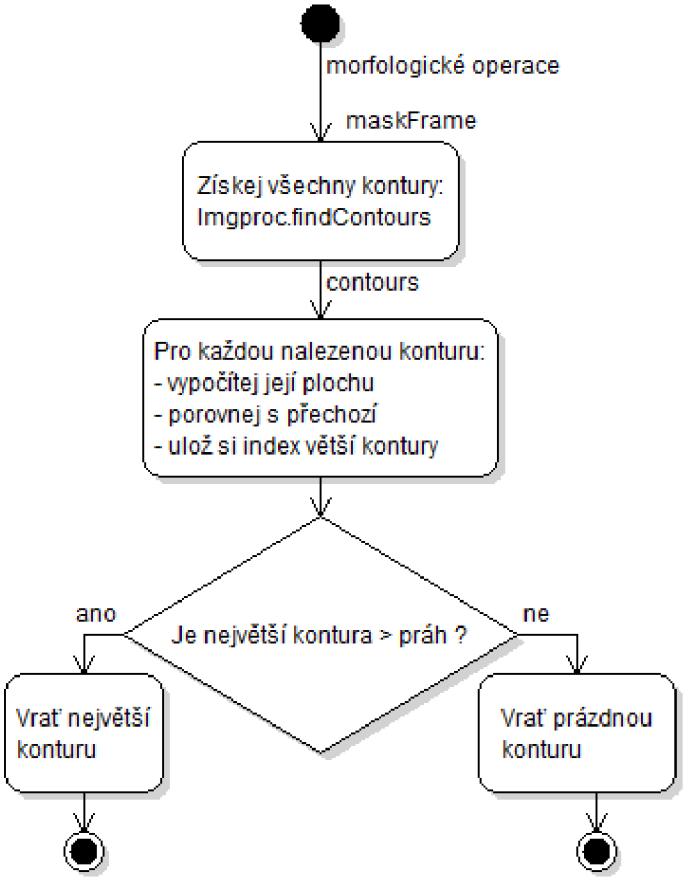
Obr. 2.1: Použití třídy BackgroundSubtractorMOG2

Jak už bylo zmíněno v teorii, tento model má závažné nevýhody a produkuje duchy, které jsou vidět na obrázku 1.5. Použije se tedy statický model. Nejdříve program vyzve uživatele, aby před kamerou ničím nehýbal a nevkládal před ní ruku. BackgroundSubtractorMOG2 si z prvních padesáti snímků vytvoří model pozadí a ten při každé další separaci popředí nijak neaktualizuje. To je provedeno nastavením parametru learningRate na nulu v metodě apply(). Přeučení lze kdykoliv aktivovat nastavením proměnné learningCounter.

Po provedení segmentace se nachází popředí v matici maskFrame. Obsahuje ovšem nežádoucí šum. Ten lze do určité míry potlačit vhodně zvoleným prahem, který se nastavuje při vytváření objektu BackgroundSubtractorMOG2. Vyšší hodnoty ovšem až příliš ořezávají okraje segmentované ruky, takže má detektor problémy se zachycením gesta. Proto se použijí morfologické operace eroze a dilatace, které jsou popsané v kapitole 1.3. Nejdříve se eliminuje pomocí eroze šum, pak se vrátí okraje detekovaného tvaru zpět do původní podoby dilatací.

```
Imgproc.erode(maskFrame, maskFrame, kernel);
Imgproc.dilate(maskFrame, maskFrame, kernel);
```

2.1.2 Získání kontury



Obr. 2.2: Hledání největší kontury

Následně dojde k testování podmínky, zda nedošlo ke změně světelných podmínek. Pokud je třeba model pozadí přeučit, nastaví se znovu learningCounter. Pokud ne, pokračuje se získáním kontur (obr. 2.2) pomocí metody Imgproc.findContours.

Výstupem je řada matic bodů, přičemž každá položka řady odpovídá nalezené kontuře. Tu tvoří body ve sloupcové matici. V obrazu bude pravděpodobně více než jedna kontura. Některé mohou náležet obličeji uživatele, jiné se mohou vztahovat k pozadí (puštěný větrák), další třeba artefaktům vzniklým změnou osvětlení v místnosti. Můžeme předpokládat, že lidská ruka bude v obrazu tvořit největší konturu, tudíž ji vyseparujeme z řady. Jednoduše hledáme konturu, co zabírá na snímku největší plochu. Také ji porovnáme s definovaným prahem, čímž eliminujeme falešné detekce, kdy v obrazu není ruka a v pozadí se něco malého pohybuje. Pomocná funkce findBiggestContour se spolu s dalšími nachází ve třídě Helper. Tímto krokem je segmentace ukončena.

2.1.3 Detekce počtu prstů

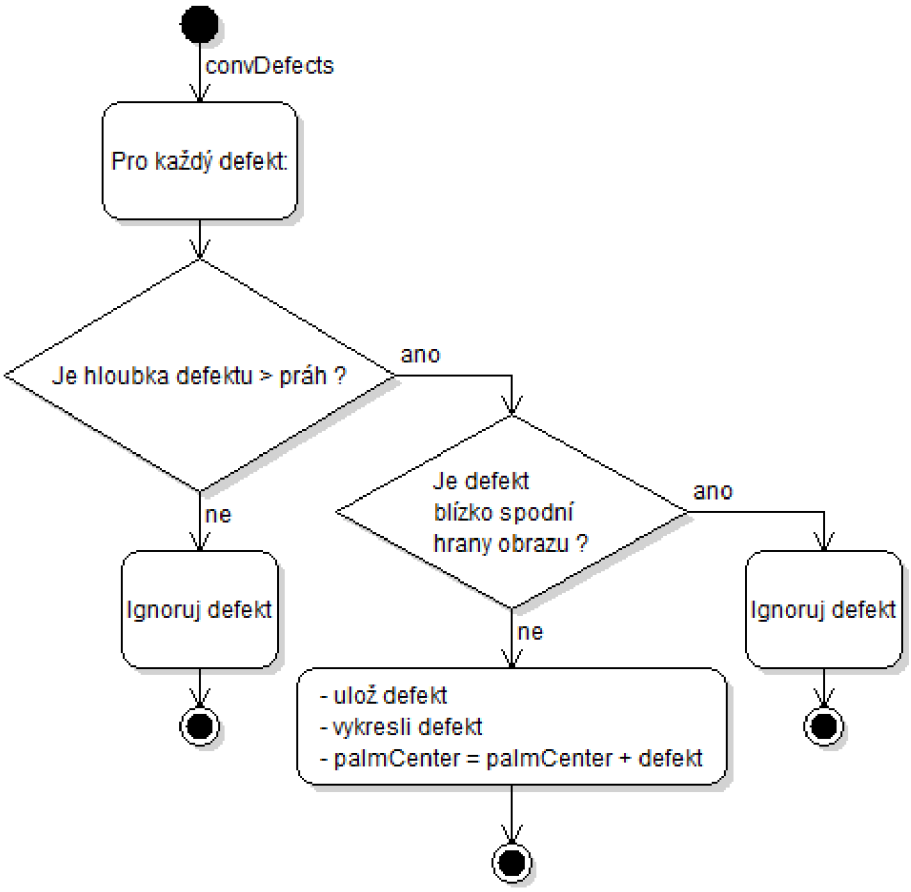
Následující kód pokračuje, jen pokud je nalezena alespoň malá kontura. Jelikož je příliš detailní, hledaný konvexní polygon obsahoval na každý prst více než jeden bod. Logickým krokem tedy bylo snížit počet bodů v kontuře tím, že ji budeme aproximovat novým polygonem s menším množstvím vrcholů, jak je vidět na obrázku 2.3. Tato metoda se ovšem neosvědčila. Každé špičce prstu nyní náležel jen jeden bod konvexního polygonu, některé z nich byly ale velmi nestálé, takže počet nalezených prstů neustále kolísal. Proto je metoda approxPolyDPC zakomentovaná. Zakomentovaný je i kód porovnávající nalezenou konturu s předpřipravenými šablonami. V teoretické části je vysvětleno, proč tato metoda nakonec nebyla použita.

Obr. 2.3: Nová kontura po aproximace mnohoúhelníkem

Dojde k získání konvexního mnohoúhelníku a defektů konvexity pomocí těchto volání:

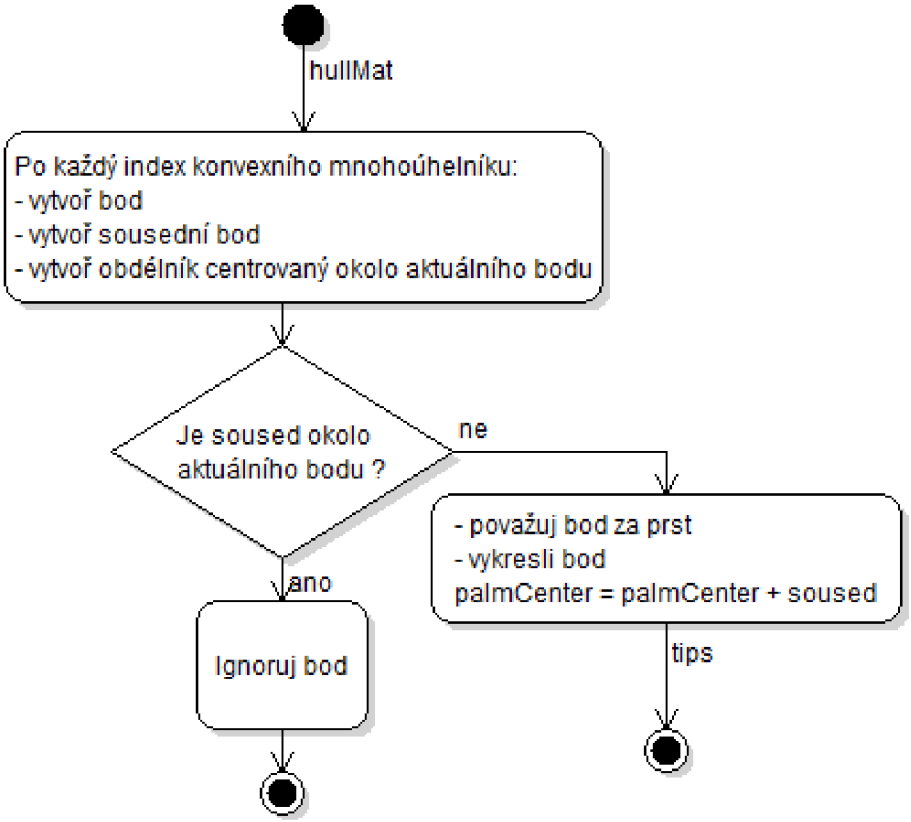
```
Imgproc.convexHull(contour, hull);
Imgproc.convexityDefects(contour, hull, convDefects);
```

Ve smyčce for se prochází všemi defekty a podle nastaveného prahu jsou ty s malou hloubkou (obr. 1.9) zahozeny. Dva defekty se také nacházejí u kraje obrazovky a budou také zahozeny. Zbývající body jsou vykresleny a přičteny k bodu palmCenter, který reprezentuje předpokládaný střed dlaně, viz obr. 2.4.



Obr. 2.4: Filtrování defektů

Z konvexního mnohoúhelníku získáme informace o špičkách prstů. Díky velkému rozlišení kontury ovšem získáme více než jeden bod na prst, takže ostatní musíme redukovat (obr. 2.5). Jelikož jsou v kontuře seřazeny za sebe, testujeme, zda se v blízkém okolí testovaného bodu nachází soused. Pokud ne, bod se považuje za potenciální špičku prstu. Blízké okolí je definováno čtvercem, v jehož středu leží testovaný bod. Kruhové okolí by bylo vhodnější, ale třída Point už obsahuje metodu inside, která pracuje s obdélníkem. Ke středu dlaně se dále připočtou polohy prstů.



Obr. 2.5: Redukce shluků bodů konvexního mnohoúhelníku

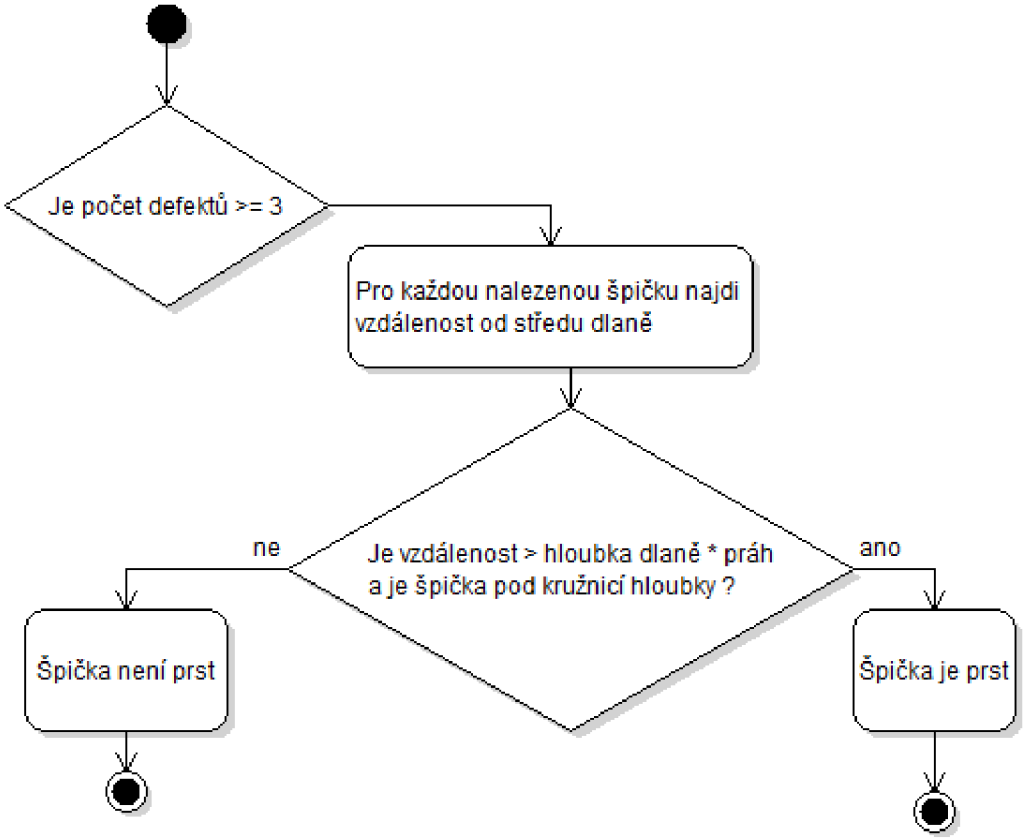
Střed dlaně už můžeme získat podělením počtu přičtených bodů.

```
palmCenter.x /= defPointsFar.size() + tips.size();
palmCenter.y /= defPointsFar.size() + tips.size();
```

Hloubka dlaně je znázorněna kruhem s poloměrem palmRadius, který je spočítán jako průměrná vzdálenost od středu dlaně ke všem defektům. Vzdálenost každého se tedy připočte do proměnné palmRadius a podělí se počtem defektů.

```
//vzdalenost dvou bodu
palmRadius += Math.sqrt(((defPointsFar.get(i).x - palmCenter.x) *
    (defPointsFar.get(i).x - palmCenter.x)) + ((defPointsFar.get(i).y -
    palmCenter.y) * (defPointsFar.get(i).y - palmCenter.y)));
```

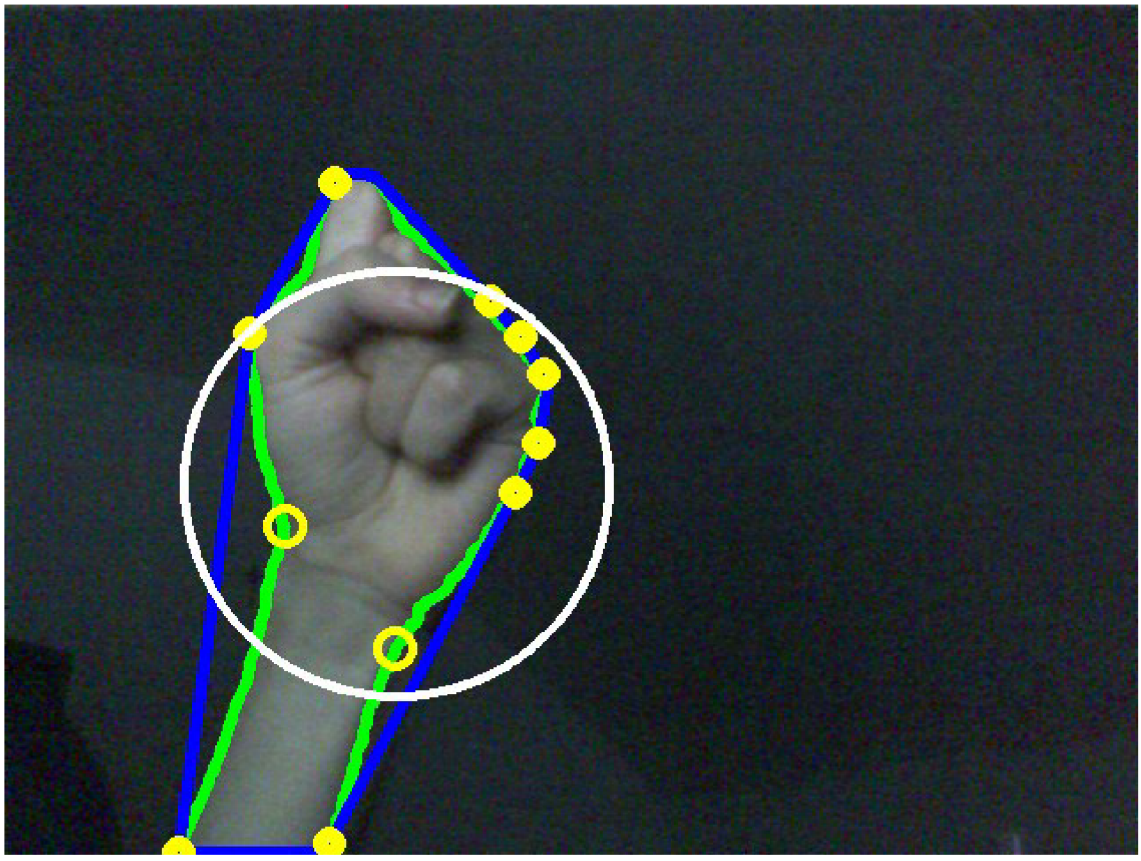
Zbývá už jen spočítat prsty. Je vytvořen další nový kruh, jehož střed leží na detekované dlani. Jeho poloměr je vypočítán násobením poloměru dlaně s hodnotou pod názvem fingerDetectionThreshold. Zjistí se vzdálenost špiček prstů od středu dlaně a pokud se nacházejí za tímto prahem, jsou považovány za natažené prsty. Neuvažují se také body, které leží pod touto kružnicí. Jak ale můžeme vidět na obrázku 2.7, pěst vyvolá falešnou detekci, protože jeden bod konvexního mnohoúhelníku je mimo



Obr. 2.6: Počítání prstů

tuto kružnici. Jak si ale můžeme všimnout, pro pěst je charakteristická přítomnost dvou defektů okolo zápěstí. Nejdříve tedy testujeme podmínku, zda jsou v obrazu tři či více defektů a pokud ano, teprve tehdy prsty počítáme (obr. 2.6).

Počet prstů a poloha středu dlaně se odesílají do objektu typu DetectorController statickými metodami setFingerCount(int fingerCount) a setPalmCenter(double palmCenterX, double palmCenterY), kde jsou dále zpracovány.



Obr. 2.7: Pěst

3 MULTIMEDIÁLNÍ PROHLÍŽEČ

Prohlížeč je napsán v jazyce Java, proto je využita knihovna JavaFX pro implementaci ovládacího rozhraní. Její předností je, že je úzce svázána s jazykem a jejím použitím nevznikají žádné další závislosti na externích knihovnách. Při spouštění programu je však nezbytné na užívané platformě mít nainstalován JRE alespoň ve verzi 7u6.

JavaFX využívá pro implementaci ovládacího rozhraní návrhový vzor MVC. Písmeno M v této zkratce znamená model, V view a C pak controller, tedy ovladač. V praxi to znamená, že data aplikace (model) jsou oddělena od definice ovládacího rozhraní (view) a tyto dvě části propojuje ovladač (controller). Architektura prohlížeče tento návrhový vzor respektuje a přichází s jednoduchým, avšak robustním systémem pro správu ovladačů.

3.1 Architektura programu

3.1.1 Bridge

Obecným požadavkem na takový systém je, aby byly různé části ovládacího rozhraní, tedy ovladače, spolu schopny komunikovat, případně komunikovat se sdílenými objekty. Oba tyto požadavky v tomto případě řeší příhodě pojmenovaná třída Bridge, která sdružuje ovladače a provádí jejich načítání pomocí funkce loadViewController(View view, String pathToView).

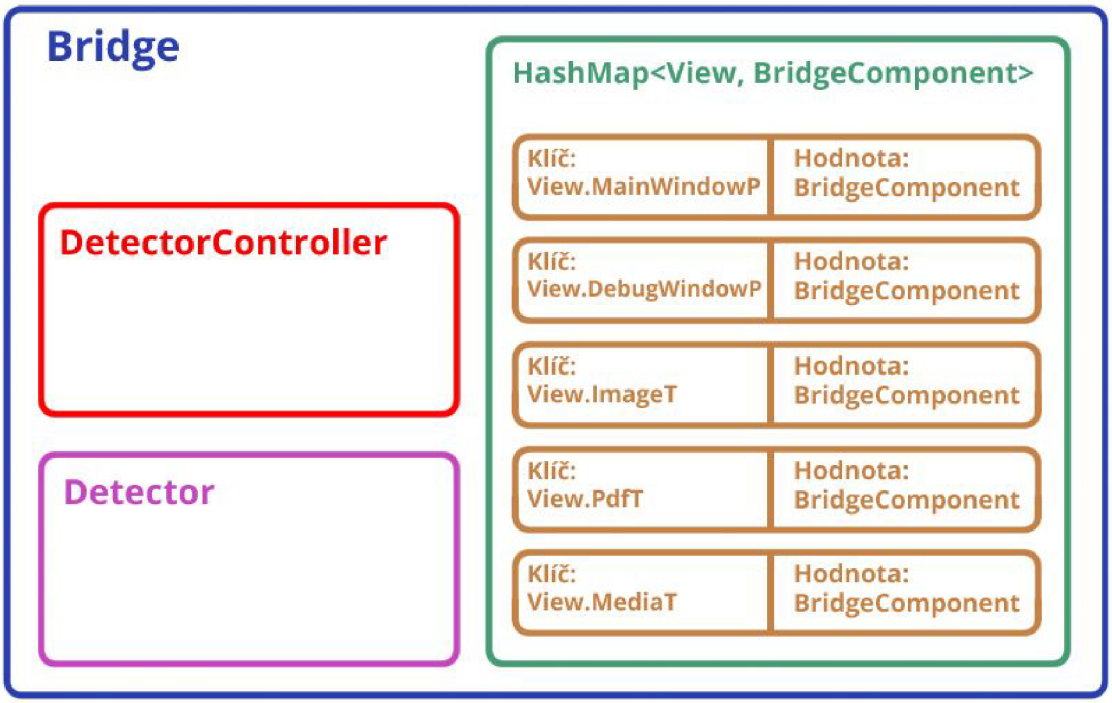
Ovladače jsou uchovávány v hešovací mapě, kde klíč zastupuje výčtový typ View. Ve třídě Bridge se také nachází samotný objekt detektoru, který už byl popsán v sekci 2.1. Posledním členem je pak objekt typu DetectorController. Tento název je poněkud zavádějící, protože se nejedná o ovladač, jak ho chápe JavaFX. Jedná se o třídu, pomocí které detektor gest ovládá prohlížeč. Tato třída bude popsána později.

3.1.2 BridgeComponent

Aby bylo možné s ovladači jednotně pracovat, byla definována abstraktní třída BridgeComponent. Ta definuje pouze funkce pro nastavení a získání reference na Bridge a na kořen stromu uzlů, který je načten a vytvořen z fxml souboru.

Abstraktní třídu ready() si definuje každý ovladač sám podle potřeby a je volána z funkce loadViewController(View view, String pathToView), jakmile jsou nastaveny výše zmíněné reference. Slouží tedy vlastně jako inicializační funkce pro případ, že

ovladač při vytvoření potřebuje přistupovat na most. Zjednodušené blokové schéma architektury ukazuje obrázek 3.1.



Obr. 3.1: Blokové schéma architektury prohlížeče

3.2 Hlavní okno

V minulosti měla JavaFX problémy s úniky paměti a obecně je výhodnější objekty recyklovat, což platí zvláště pro jazyky využívající garbage collector. Nemůžeme totiž dopředu odhadnout, kdy bude spuštěn a jaký bude mít vliv na plynulost aplikace. Proto prohlížeč načte strom GUI prvků z fxml souboru pouze jednou. To znamená, že v tomto konkrétním případě není možné vedle sebe zobrazit dva obrázky či přehrávat dvě videa. Jedná se o nevýhodu specifickou pro architekturu tohoto programu, z hlediska návrhu ovládacího rozhraní však požadavek na zobrazení dvou stejných souborů neexistuje. Ovládací rozhraní je inspirováno internetovými prohlížeči a používá systém záložek, tudíž je viditelný vždy jen jeden dokument. Při přepnutí záložky se v případě potřeby strom GUI prvků jednoduše přesune do aktivní záložky.

3.2.1 MainWindowController

O správu záložek se stará ovladač MainWindowController, který zároveň obsluhuje hlavní okno. Každý nově otevřený soubor se zobrazí ve své záložce. Zároveň se na-

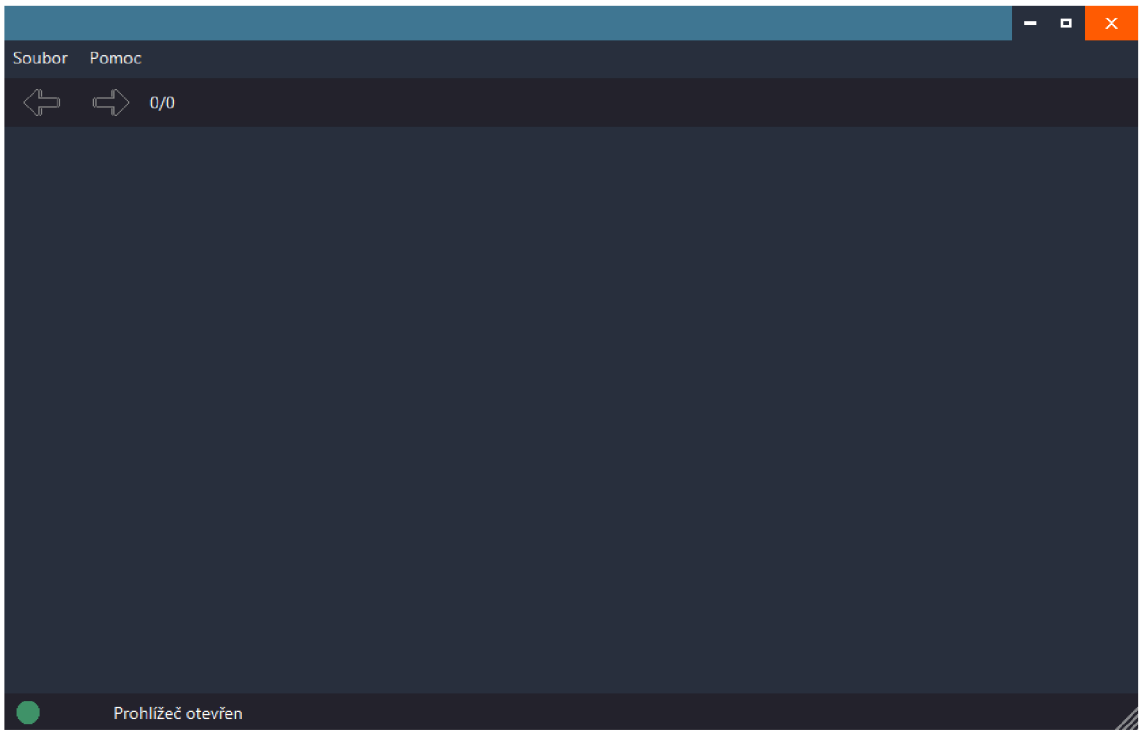
čtou všechny další podporované soubory z adresáře, aby mezi nimi mohl uživatel jednoduše listovat pomocí šipek. Jak už ale naznačil předchozí odstavec, může docházet k případům, kdy má uživatel otevřené dva soubory stejného typu, strom GUI prvků a přiřazený ovladač je však načten pouze jedenkrát. Proto si každá záložka ukládá ještě tzv. stav.

Stavy se nacházejí v hešovací mapě pojmenované ve zdrojovém kódu tabStates, kde klíčem je reference na záložku a hodnotou objekt typu VariantState. Tento objekt slouží jako struktura, která může obsahovat data vázající se ke stavu jakéhokoliv ovladače. Při přepnutí záložky uživatelem je možné stav uložit ve zpětném volání saveState(Tab sourceTab) a nahrát ve loadState(Tab sourceTab), které se spolu s dalšími metodami nacházejí v rozhraní IControllerViewState. Pokud chce být jakýkoliv ovladač recyklován a umět tak přepínat stavy, musí si toto rozhraní implementovat. VariantState v tomto případě obsahuje zarovnané pole objektů typu File s názvem filesInDirectory, což jsou všechny soubory ve vybraném adresáři včetně nepodporovaných. Podporované jsou zarovnány a část pole s nepodporovanými soubory je odsazena hodnotou offset. Integer fileIndex pak ukazuje na aktuálně otevřený soubor. Proměnné currentPage a vValue se pak týkají stavu ovladače PdfTController.

Otevírání souborů provádí metoda openFile(). Nejdříve se vytvoří okno pro výběr souboru, což provádí objekt typu FileChooser. Z načteného souboru se pomocí volání statické metody FilenameUtils.getExtension(String cesta) získá koncovka a následně adresář parentDirectory, ve kterém se nachází. Z adresáře se získají všechny soubory voláním listFiles(). Ve smyčce for proběhne jejich zarovnání a také se hledá index vybraného souboru. Po dokončení smyčky získáme tento index a také odsazení validních dat. Dle dříve získané koncovky se následně vytvoří nová záložka s vhodným uživatelským rozhraním. Logika listování mezi podporovanými soubory v adresáři se nachází v metodě switchFile(boolean isNextFile), která posune index fileIndex vhodným směrem a podle potřeby změní v záložce strom uzlů uživatelského rozhraní. Nad přiřazeným ovladačem následně zavolá metoda loadState, kde proběhne samotné načtení a zobrazení dat.

3.2.2 Ovládací rozhraní hlavního okna

Logika okna je od základu napsána a nachází se z velké části ve třídě Popup a to z toho důvodu, aby byl vzhled konzistentní skrze všechny operační systémy. Přesto je s ním možné pracovat tak, jako s klasickým oknem v operačním systému Windows. Je možné ho posouvat a přychytit k okrajům obrazovky, minimalizovat ho a maximalizovat. Jeden z rozdílů je, že druhé použití maximalizace okno vykreslí přes celou obrazovku a překryje tak všechny panely operačního systému. Druhý rozdíl je, že měnit velikost okna je možné jen pomocí pravého dolního rohu aplikace.



Obr. 3.2: Hlavní okno prohlížeče

Nabídka Soubor obsahuje tři položky. Otevřít soubor otevře FileChooser, tedy výzvu k výběru souboru. Je možné vybrat i ty nepodporované, případně si vyfiltrovat podporované. Položka Zavřít soubor zavře aktuálně otevřenou záložku a Vypnout vypne celou aplikaci. Nabídka Pomoc obsahuje položku Nápověda prohlížeče, která otevře nápovědu, jak aplikaci ovládat pomocí webové kamery. Otevří debug. okno pak otevře nové okno zobrazující výstup segmentace a vstupní obraz z webové kamery, do kterého jsou vykresleny informace o poloze dlaně a prstů. Pomocí tohoto okna je možné nastavit kameru tak, aby byla kvalita detekce co nejlepší.

Pomocí šipek lze listovat soubory, přičemž vedle nich se nacházejí informace o počtu souborů a cestě k nim. V levém dolním rohu se nachází ikona pro zapnutí a vypnutí ovládání pomocí gest, vedle se pak zobrazují zprávy týkající se činnosti prohlížeče.

Prohlížeč je schopen otevírat typy dat z tabulky 3.1. Každý typ načítá, zpracovává a zobrazuje jiná třída.

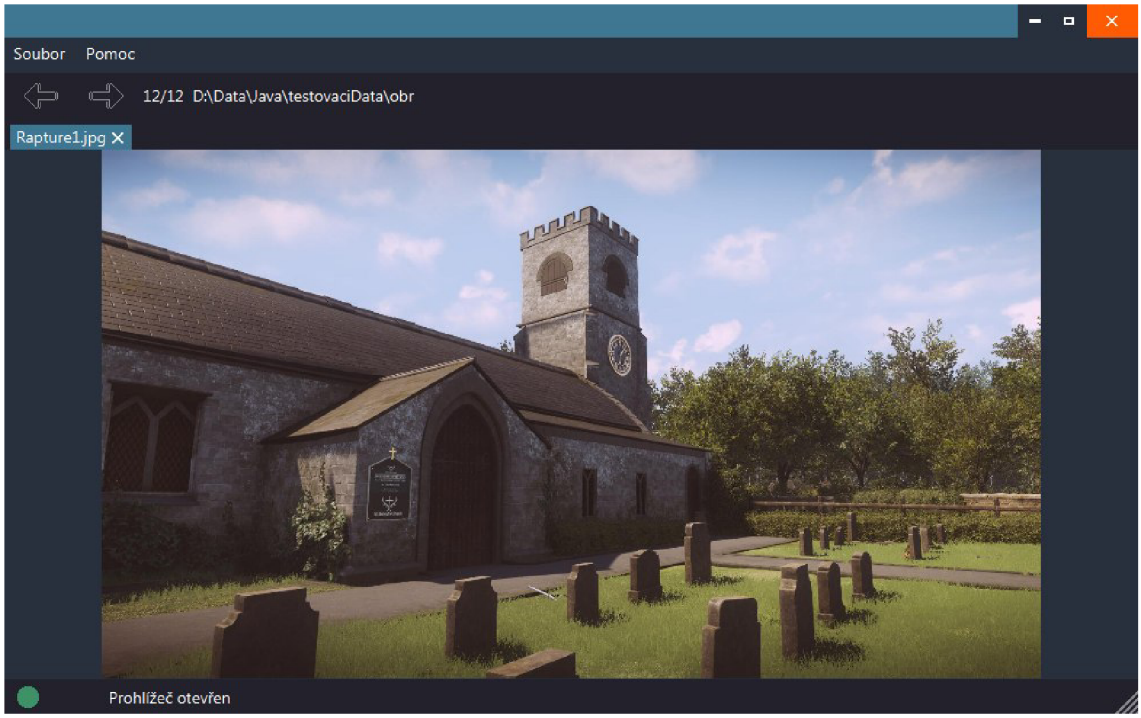
Typ	Kontejner
Obrázky	.jpg, .jpeg, .bmp, .gif, .png
Multimédia	.mp3, .waw, .mp4, .m4a, .m4v, .flv, .fxm, .aiff, .aif
Dokumenty	.pdf

Tab. 3.1: Podporované typy dat

3.3 Obrázky

3.3.1 ImageTController

Vybraný soubor je nejdříve načten objektem typu FileInputStream ze standardního balíčku java.io. Obrázek je reprezentován objektem Image z balíku javafx.scene.image, jehož konstruktor přijímá právě objekt typu FileInputStream. Třída Image je schopna dekódovat obrazová data ve formátu JPEG, GIF, BMP a PNG. Jakmile je obrázek úspěšně načten, je zobrazen v GUI uzlu ImageView. V metodě ready() je velikost tohoto uzlu svázána metodou bind s velikostí okna prohlížeče.



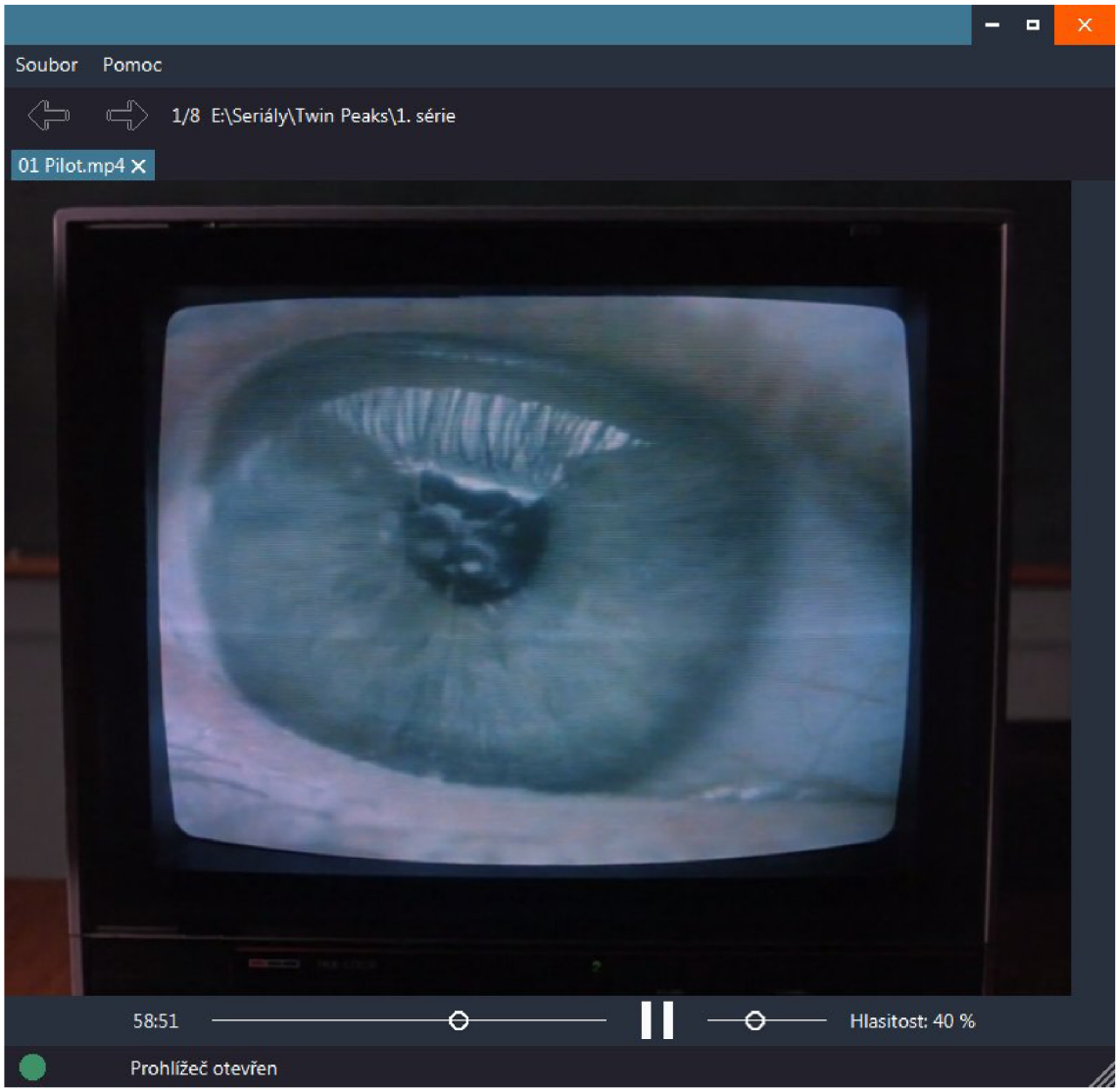
Obr. 3.3: Otevřený obrázek

3.4 Multimédia

3.4.1 MediaTController

Všechny multimediální soubory jsou načítány do objektu jednotného typu Media. Pokud je soubor úspěšně načten a nedojde k vyvolání výjimky, vytvoří se objekt typu MediaPlayer, přičemž do konstruktoru vstupuje médium. Následně je přehrávač zobrazen v uzlu typu MediaView. Třída Media dekóduje standardy AAC, MP3, PCM, H.264, AVC a VP6.

3.4.2 Ovládací rozhraní multimédií



Obr. 3.4: Ovládací rozhraní multimédií

Ovladač MediaTController dále obsluhuje dodatečné ovládací rozhraní přehrávače. Přehrávání je možné pozastavit, znovu spustit a měnit aktuální pozici přehrávání pomocí posuvníku, vedle něhož je zobrazen aktuální čas přehrávání. Dále je možné regulovat hlasitost přehrávače pomocí posuvníku hlasitosti. V metodě ready() je velikost uzlu MediaView svázána metodou bind s velikostí okna prohlížeče.

3.5 PDF dokumenty

3.5.1 PdfTController

JavaFX nepodporuje dekódování pdf dokumentů, proto bylo nutné tuto funkčnost prohlížeči implementovat. Pro práci s pdf dokumenty existuje velké množství bezplatných i komerčních knihoven. Mezi těmi bezplatnými je ovšem malé množství kvalitních a aktualizovaných knihoven. Takovou je například Apache PDFBox, která je dostupná pod licencí Apache License v2.0 a která je v tomto prohlížeči použita.

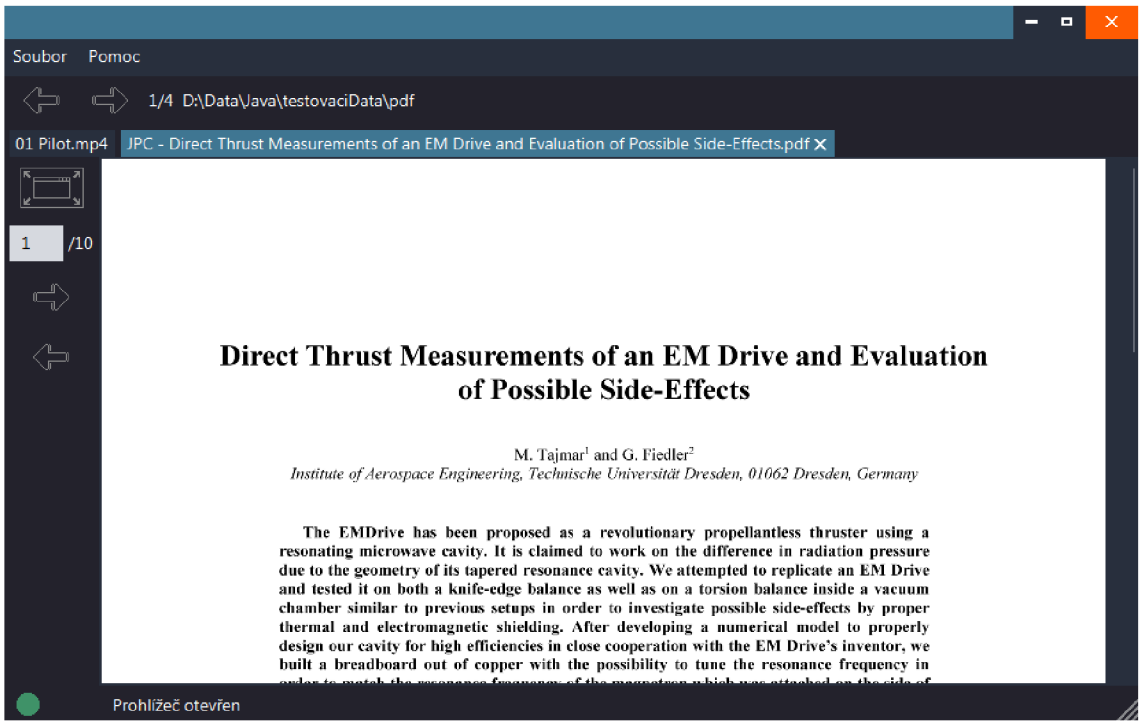
Nejdříve je soubor načten pomocí statické metody load ze třídy PDDocument. Load pak vrací objekt třídy PDDocument, který pdf dokument reprezentuje. Pokud je objekt načten a není vyvolána výjimka, vytvoří se objekt typu PDFRenderer, jehož konstruktor přijímá objekt dokumentu. Vykreslování probíhá v metodě renderCurrentPage(), kde se nejdříve obrázek vykreslí pomocí metody renderImageWithDPI do objektu typu BufferedImage. Ten je převeden statickou metodou toFXImage ze třídy SwingFXUtils do obrázku typu Image z balíčku javafx.scene.image. Tento obrázek vykresluje uzel ImageView.

3.5.2 Ovládací rozhraní prohlížeče dokumentů

V praxi se ukázalo, že v případě velkých dokumentů o stovkách stran vykreslovaných ve vysoké kvalitě může dojít haldě volná paměť a dojde k vyvolání výjimky OutOfMemoryError. Nejjednodušším způsobem je proto vykreslovat vždy jednu stranu. Ta se roztáhne do šířky okna. Dokumenty je možné listovat pomocí šipek, nalézt konkrétní stranu přímým zadáním čísla, přibližovat a oddalovat pohled kolečkem myši a držením levého tlačítka myši posouvat pohled do stran. Také lze dokument zobrazit přes celou obrazovku, což je provedeno pomocí nového okna (třída Stage), do kterého je zobrazen strom uzlů náležející tomuto ovladači.

3.6 Detekce gest

Implementovaný detektor je schopen počítat natažené prsty a sledovat pozici dlaně. Tato data jsou předávána objektu typu DetectorController v nezpracované formě



Obr. 3.5: Ovládací rozhraní prohlížeče dokumentů

pomocí volání statických funkcí setFingerCount(int fingerCount) a setPalmCenter(double palmCenterX, double palmCenterY). Z nich je dále nutné odvodit ovládací gesta a definovat ovládací akce.

Nejdříve jsou vytvořeny dva buffery pro pozici dlaně s názvy palmCenterXBuffer a palmCenterYBuffer, což jsou pole typu double, jejichž velikost definuje konstanta BUFFER_SIZE. Jeden buffer s názvem fingerCountBuffer je také vytvořen pro počet prstů, jehož velikost definuje stejná konstanta. Jedná se o rolovací buffery, tudíž se nikdy nevyprazdňují a nejstarší hodnoty jsou přepisovány novými.

3.6.1 Volání setPalmCenter

V případě volání setPalmCenter se do bufferu přidají nové hodnoty a z celého bufferu je vypočítáná průměrná pozice dlaně. To je tak učiněno proto, aby byl posun myši co nejhladší a nejstabilnější. Z průměrné pozice z předchozího volání a z aktuální pozice se vypočítá rozdíl deltaX a deltaY. Pokud je dlaň rozevřená a je detekováno všech pět prstů, jsou tyto rozdíly připočítávány k pozici myši. Rozevřená dlaň je pro posun myši nejvhodnější, protože ji detektor dokáže nejlépe detekovat a pozice dlaně je v tomto případě nejstabilnější. Pokud dojde ke změně gesta, buffer a hodnoty prevAveragePalmCenterX a prevAveragePalmCenterY jsou vynulovány.

3.6.2 Volání setFingerCount

Průběh tohoto volání začíná podobně jako setPalmCenter. Počet prstů se uloží do rolovacího bufferu, tentokrát však průměr počítat nelze. Proto se zjistí četnost jednotlivých hodnot v bufferu a pokud je tato četnost vyšší či stejná jako práh získaný výpočtem BUFFER _SIZE · FINGER _COUNT_THRESHOLD, je toto gesto převedeno na akci. Využitím bufferu se lehce zpomalí provádění akcí, výhodou je však odfiltrování šumu, kdy detektor v krátkém časovém okamžiku vyhodnotí chybné gesto.

Pokud je aktuální gesto jeden natažený prst a předtím bylo jiné, vygeneruje se akce držení levého tlačítka myši. Změnou gesta dojde k puštění tlačítka. Dva natažené prsty vygenerují kliknutí pravým tlačítkem. Posledními dvěma gesty je možné listovat soubory v adresáři a stránkami v dokumentu. Dochází k testování podmínky na natažené prsty a zároveň musí deltaX přesáhnout konstantu MOUSE _DELTA.

3.6.3 Generování akcí

V případě výše zmíněného listování soubory stačí testovat podmínky a volat už definované funkce, jejichž volání je také svázáno s ovládacím rozhraním aplikace. Pro generování akcí spjatých s myší je nutné využít třídu Robot. V případě Javy jich existuje rovnou několik a každá se váže k jiné knihovně pro tvorbu ovládacího rozhraní. V prohlížeči je implementována ta z balíčku com.sun.glass.ui. Je také nutné se rozhodnout, jakým způsobem budou akce generovány.

Pokud by byl použit FXRobot, bylo by možné vytvářet události přímo pro ovládací rozhraní. Výhodou je, že akce nikdy nepřekročí práh aplikace a proto nemůže dojít k nechtěné manipulaci s operačním systémem, když například dojde ke změně světelných podmínek a detektor začne klikat na ikony na ploše. Nevýhodou je, že tímto způsobem nelze ovládat výzvu k výběru souboru. Proto je v tomto prohlížeči použit druhý způsob, kdy akce vychází z operačního systému.

3.6.4 Vybraná gesta

V tabulce 3.2 se nachází shrnutí všech podporovaných gest.

3.7 Zhodnocení rozlišovacích schopností detektoru

Zhodnotit rozlišovací schopnosti detektoru nelze úplně exaktně, neboť kvalita detekce závisí na mnoha faktorech, jako například na použité webové kameře, na světelných podmínkách v místnosti či na velikosti rukou. Prohlížeč nemusí být používán

Gesto	Akce
Rozevřená dlaň	Posun kurzoru
Natažení jednoho prstu	Držení levého tlačítka myši
Změna výše uvedeného gesta	Puštění levého tlačítka myši
Natažení dvou prstů	Kliknutí levým tlačítkem myši
Tři prsty + rychlý pohyb doleva	Předchozí soubor
Tři prsty + rychlý pohyb doprava	Další soubor
Čtyři prsty + rychlý pohyb doleva	Předchozí list dokumentu
Čtyři prsty + rychlý pohyb doprava	Další list dokumentu
Zatnutá pěst	Žádná akce, zablokovaný posun kurzoru

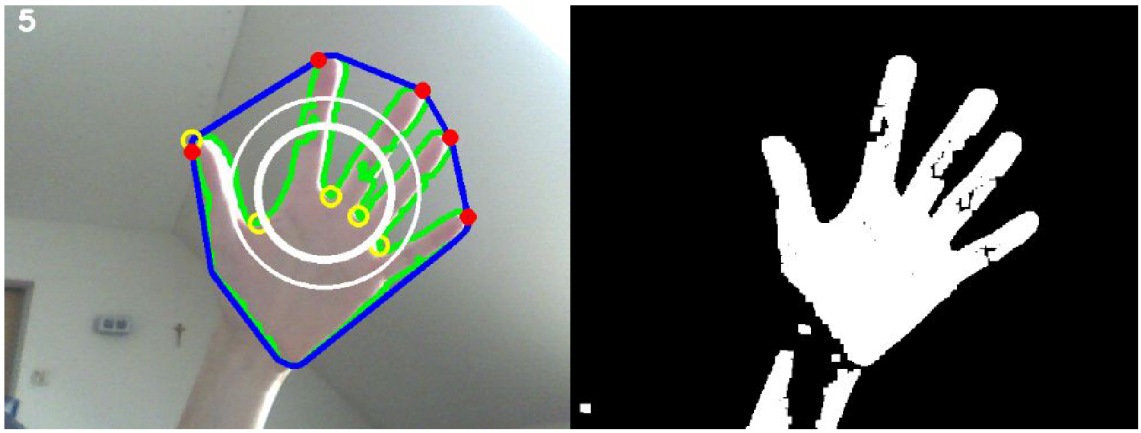
Tab. 3.2: Přehled podporovaných gest

vždy za stejných podmínek, proto se mohou subjektivní dojmy z používání lišit.

3.7.1 Podmínky

Testování probíhalo v místnosti s jedním zdrojem přirozeného světla (okno) a dvěma zdroji umělého světla (monitor, úsporná žárovka). Byla použita levná webová kamera Genius iSlim 320. Testování probíhalo jak za dobrých, tak za špatných světelných podmínek (v noci). Dále bylo vyzkoušeno, jak si detektor poradí s rušivým elementem v záběru kamery.

3.7.2 Denní světlo



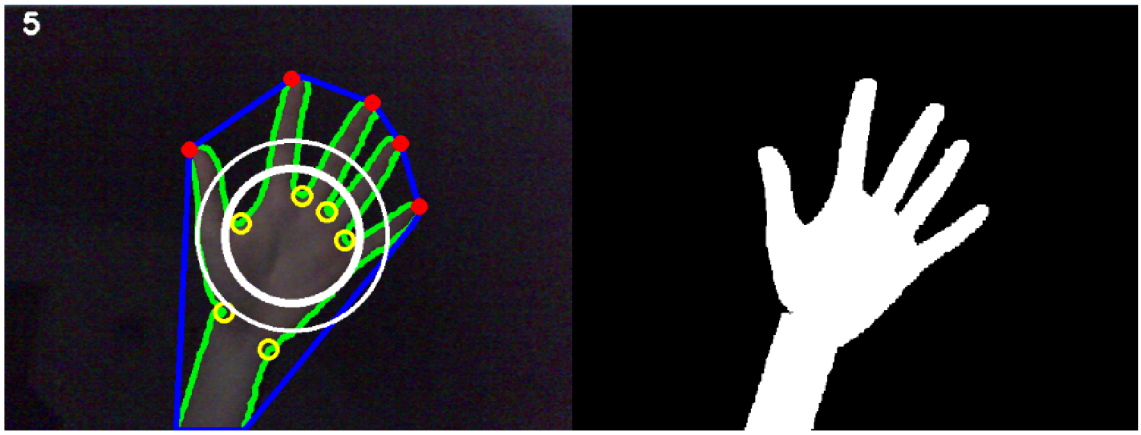
Obr. 3.6: Segmentace a detekce za denního světla

Za denního světla je rozlišovací schopnost detektoru ovlivněna hlavně intenzitou nasvětlení dlaně a pozadí. Pokud barva pozadí splývá s barvou dlaně, je pro detektor

obížné určit popředí, což vyústí v nekompletní konturu, jak je ukázáno na obrázku 3.6. Také dochází ke změnám tvaru kontury v čase, což způsobuje nestabilitu detekce středu dlaně. Proto je ovládání kurzoru nepřesné, detekce počtu prstů až na výjimky funguje bezproblémově.

Další problémy segmentace mohou způsobit samotné vlastnosti webové kamery. Některé jsou vybaveny funkcí automatického ostření či automatické změny kontrastu, čímž kvalitu segmentace podstatně snižují. Proto je vhodné podobné funkce vypnout, pokud je to možné.

3.7.3 Špatné světelné podmínky



Obr. 3.7: Segmentace a detekce v neosvětlené místnosti

Ve tmě překvapivě detektor podával nejlepší výsledky. V praxi totiž uživatel sedí před monitorem v takové vzdálenosti, kdy je ruka nasvětlena, pozadí však ne. Kontrast mezi pozadím a rukou je tak velký, že v segmentovaném obraze se neobjevují prakticky žádné jiné kontury.

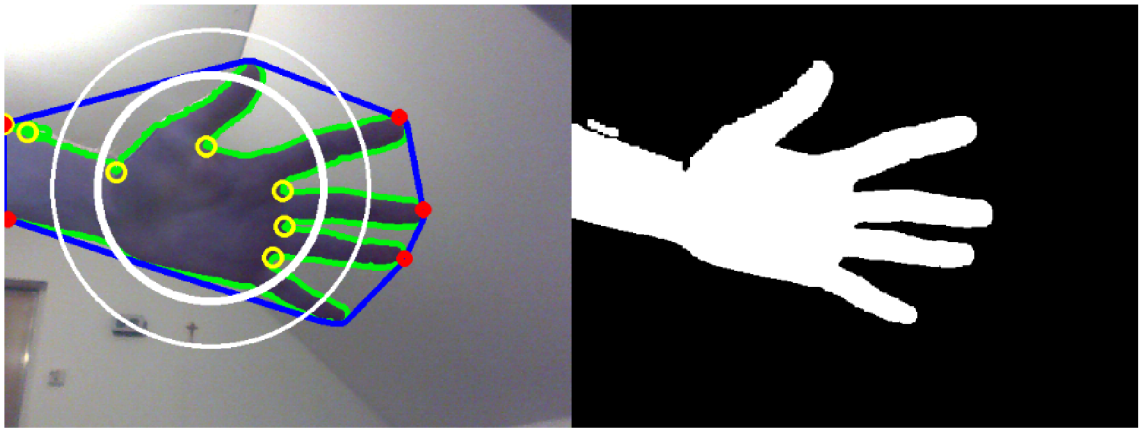
V takovém případě i detekce gest metodou počítání prstů funguje bez problémů a je tedy možné využívat všechna nadefinovaná gesta, včetně přesného ovládání kurzoru.

3.7.4 Proměnlivé světelné podmínky

Použitá metoda segmentace je velmi citlivá na proměnlivé světelné podmínky. V případě i méně výrazných změn dochází k chybné detekci popředí, tudíž není možné prohlížeč ovládat.

Detektor je však schopen podle nadefinovaného prahu poznat, kdy je šum v obrazu příliš vysoký a podle potřeby je schopen sám přeučit svůj model pozadí bez uživatelovy intervence. Toto přeučení proběhne i v případě, kdy se ruka nachází v příliš

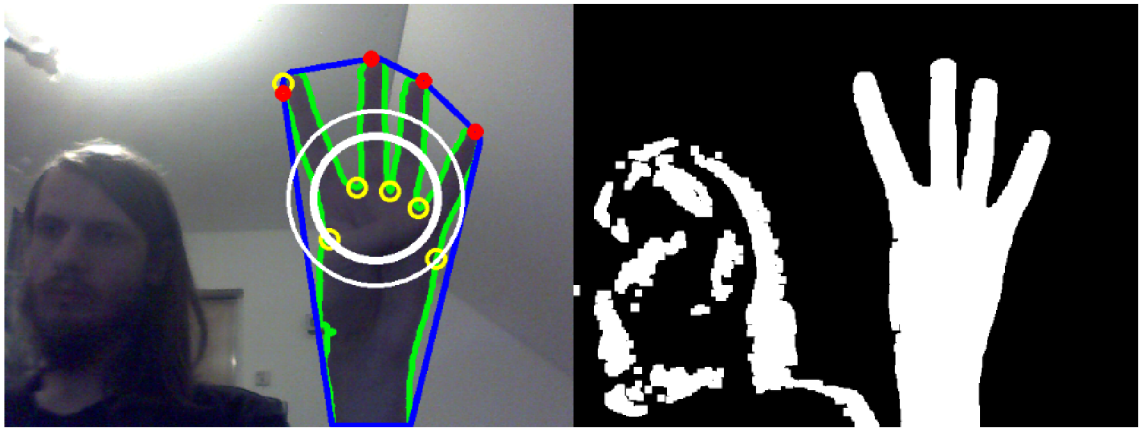
blízké vzdálenosti od kamery. Tuto vzdálenost by uživatel neměl překročit. Obrázek 3.8 ukazuje, že po automatickém přeučení je program znovu schopen ruku čistě segmentovat.



Obr. 3.8: Segmentace a detekce po automatickém přeučení

3.7.5 Více kontur v záběru

V praxi často není možné docílit toho, aby se v záběru kamery kromě natažené ruky nenacházel jiný pohybující se objekt. Jelikož detektor hledá největší konturu a zároveň pracuje s definovaným prahem, je možné program ovládat i za předpokladu, že se bude v záběru vyskytovat celý uživatel. Je ovšem nutné zajistit, aby nedošlo ke spojení kontur, aby byla kontura ruky dominantní a aby plocha minoritních kontur byla co nejmenší.



Obr. 3.9: Hlava v záběru kamery

4 ZÁVĚR

Teoretická část práce se zabývala možnostmi detekce lidské ruky v obrazu získaného pomocí webové kamery. Vybrána byla metoda segmentace pomocí odečítání pozadí a metoda detekce gest pomocí počítání natažených prstů. Implementace detektoru byla provedena v jazyku Java s pomocí knihovny OpenCV.

Navržený detektor dokáže poměrně spolehlivě detekovat 6 gest a to otevřenou dlaň, pěst, jeden, dva, tři a čtyři natažené prsty. Dále je schopen detekovat pozici středu dlaně a tuto informaci využít k posuvu kurzoru, listování podporovanými typy souborů a listování dokumentem. Práce se též zabývala tím, jak tyto nezpracované informace z detektoru interpretovat, filtrovat a přeložit na akce.

Problémy detektoru plynou hlavně z použité metody segmentace, která je citlivá na změny osvětlení. Robustnost by bylo možné zvýšit použitím několika různých typů metod segmentace, přičemž výsledná maska symbolizující ruku v obrazu by byla vytvořena jejich vhodnou kombinací. Navržený detektor používá pouze jednu metodu segmentace, přičemž nevýhody se pokouší potlačit jinými způsoby, jako například automatickým přeučením pozadí.

Detektor byl následně integrován do prohlížeče souborů, který je schopen načíst a zobrazit vybrané typy multimediálních souborů, obrázků a dokumenty ve formátu PDF. Prohlížeč je možné ovládat standardně pomocí kurzoru a také pomocí webové kamery, přičemž je možné nechat si zobrazit výstup segmentace a detekce do debugovacího okna.

LITERATURA

[1] BURDÍK, V. *Detekce ochranných pomůcek v obrazovém signálu.* Brno: Vysoké učení technické v Brně, 2014. 40 s.

[2] GRAHAM, R. L.; YAO, F. F. *Finding the Convex Hull of a Simple Polygon* [online]. 1982 [cit. 9.12.2015]. Dostupné z URL: `http://www.math.ucsd.edu/~ronspubs/83_09_convex_hull.pdf`

[3] KAEWTRAKULPONG, P.; BOWDEN, R. *An Improved Adaptive Background Mixture Model for Realtime Tracking with Shadow Detection* [online]. 2001 [cit. 9.12.2015]. Dostupné z URL: `http://www.ee.surrey.ac.uk/CVSSP/Publications/papers/KaewTraKulPong-AVBS01.pdf`

[4] SUZUKI, S.; ABE, K. *Topological Structural Analysis of Digitized Binary Images by Border Following* [online]. 1985 [cit. 9.12.2015]. Dostupné z URL: `http://tpf-robotica.googlecode.com/svn-history/r397/trunk/Vision/papers/SA-CVGIP.PDF`

[5] WALEK, P.; LAMOŠ, M.; JAN, J. *Analýza biomedicínských obrazů.* 1. vyd. Brno: Vysoké učení technické v Brně, 2013. 138 s. ISBN 978-80-214-4792-9.

[6] NAYANA, P. B.; KUBAKADDI, S. *Implentation of Hand Gesture Recognition Technique for HCI Using Open CV* [online]. 2014 [cit. 19.5.2016]. Dostupné z URL: `http://www.ijrdet.com/files/Volume2Issue5/IJRDET_0514_04.pdf`

A OBSAH PŘÍLOHY

Ve fyzické příloze na CD se nachází zdrojový kód strukturovaný do formátu NetBeans IDE projektu (složka source). Je použita verze Javy 1.8.0_60 a NetBeans IDE 8.1. Aby bylo možné projekt zkompilovat, je nutné dodat projektu přístup ke knihovnám OpenCV 3.0 pro Javu, Apache Commons IO 2.4 a PDFBox ve verzi 2.0. Ty se nacházejí ve složce Gestures\dist\lib. Zdrojové soubory se nacházejí ve složce Gestures\src.

Ve složce binaries jsou dva spustitelné jar soubory pro 32 bitový a 64 bitový operační systém Windows. Testována byla pouze 64 bitová verze na systému Windows 7.

Elektronická verze přílohy je kvůli požadavku na maximální velikost 10 MB omezena pouze na zdrojové soubory. Spustitelné soubory nebylo možné zkomprimovat tak, aby se pod tuto hranici vlezly. Binární verze jsou proto k nalezení na GitHubu z odkazu https://github.com/GlaDOSik/GestureViewer/releases.