VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

AUTOMATICKÉ ROZPOZNÁNÍ GEST LIDSKÉ RUKY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

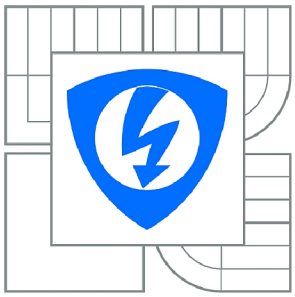
AUTOR PRÁCE
AUTHOR

ADAM OLEJÁR

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

AUTOMATICKÉ ROZPOZNÁNÍ GEST LIDSKÉ RUKY

AUTOMATIC RECOGNITION OF HUMAN HAND GESTURES

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

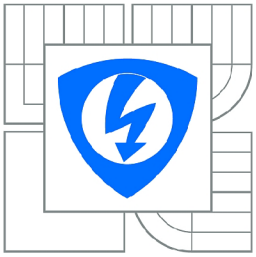
AUTOR PRÁCE — ADAM OLEJÁR

AUTHOR

VEDOUCÍ PRÁCE — Ing. PETR ČÍKA, Ph.D.

SUPERVISOR

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student: Adam Olejár ***ID:*** 134576
Ročník: 3 ***Akademický rok:*** 2012/2013

NÁZEV TÉMATU:

Automatické rozpoznání gest lidské ruky

POKYNY PRO VYPRACOVÁNÍ:

Prostudujte současné algoritmy pro sledování lidské ruky a gest, které mohou být rukou zobrazovány. Vyberte vhodný způsob pro detekci a sledování lidské ruky, který budete implementovat v programovacím jazyce JAVA. Vaši implementaci prakticky ověřte a vhodnou metodikou zhodnoťte výsledky.

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, Rafael C.; WOODS, Richard E. Digital Image Processing. 3. Londýn : Pearson Pentice Hall, 2008. 954 s. ISBN 978-0-13-505267-9.

Termín zadání: 11.2.2013 ***Termín odevzdání:*** 5.6.2013

Vedoucí práce: Ing. Petr Číka, Ph.D.
Konzultanti bakalářské práce:

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Táto bakalárska práca je zameraná na realizáciú aplikácie detekujúcu ruku vo videu z webkamery. V prvej časti sú popísané možnosti vhodného predspracovania obrazu a popísané detektory objektov založené na rôznych metódach. V druhej časti sa projekt okrajovo zaoberá multiplatformovým jazykom Java a knižnicami pomocou ktorých bude projekt realizovaný. V tretej časti je zhrnutá realizácia zadania.

KĽÚČOVÉ SLOVÁ

BLOB, detekcia hrán, detektor, detektor Harris, detektor Hessian, detektor Moravec, gestá ruky, mediánový filter, POI, point of interest, ROI, region of interest, SURF, speeded up robust features

ABSTRACT

This bachelor thesis is aimed at realization application detecting hand in the video from a webcam. The first section describes the pre-processing options and object detectors based on different methods. Second part of the project talks about cross-platform Java language and useful libraries for project realization. The third section is a summary realization of the assignment.

KEYWORDS

BLOB, edge detector, detector, detektor Harris, detektor Hessian, detektor Moravec, hand gestures, median filter, POI, point of interest, ROI, region of interest, SURF, speeded up robust features

OLEJÁR, Adam *Automatické rozpoznání gest lidské ruky*: bakalárska práca. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2013. 45 s. Vedúci práce bol Ing. Petr Číka, Ph.D.

PREHLÁSENIE

Prehlasujem, že som svoju bakalársku prácu na tému „Automatické rozpoznání gest lidské ruky“ vypracoval samostatne pod vedením vedúceho bakalárskej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej bakalárskej práce ďalej prehlasujem, že v súvislosti s vytvorením tejto bakalárskej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/nebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona č. 121/2000 Sb., o právu autorském, o právoch súvisejúcich s právom autorským a o zmeně niektorých zákonov (autorský zákon), vo znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhé, hlavy VI. diel 4 Trestného zákoníka č. 40/2009 Sb.

Brno ....

(podpis autora)

POĎAKOVANIE

Rád bych poděkoval vedoucímu semestrálneho projektu panu Ing. Petru Číkovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

(podpis autora)

OBSAH

ZOZNAM OBRÁZKOV

ZOZNAM TABULIEK

ÚVOD

V dnešnej dobe sa stále častejšie stretávame s potrebou jednoducho a efektívne ovládať zariadenia ktoré používame každodenne v najrôznejších odvetviach ľudskej činnosti. Najrozšírenejšou je technológia diaľkového ovládača ako zariadenia, na ktorom sa pomocou tlačidiel a mikrokontroléru moduluje signál, ktorý je pomocou IR LED prenášaný do zariadenia. Cieľom tejto práce je vytvoriť aplikáciu ktora by dokázala lokalizovať pomocou kamery ľudskú ruku v obraze a v reálnom čase detekovala predom definované gestá ktoré by táto ruka znázorňnovala. Následne by pomocou nich bolo možné ovládať aplikácie príp. zariadenia. Aplikácia bude realizovaná v multiplatformovom jazyku JAVA. Hlavnými výhodami jazyka je univerzálnosť a kompatibilita so zariadeniami kvoli vlastnostiam tohto jazyka. Vďaka týmto špecifikám bude možné aplikáciu implemetnovať do širokého množstva zariadení od televíznych prímačov až po bezkontaktné ovládanie rôzneho softvéru ako sme mohli vidiet napr. v rôznych Sci-Fi filmoch.

Práca je rozdelená na 5 časti. Prvá časť 1 sa zaoberá teóriou detektorov a detekcie objektov v obraze. Druhá časť 2 obsahuje výčet najvýznamnejších knižníc a dôvod ich výberu pre účely tento práce. Tretia časť 3 sa zaoberá realizáciou zadania tejto bakalárskej práce. Štvrtá časť 4 obsahuje zhodnotenie projektu so štatistickým vyhodnotením úspešnosti detektoru. Piata časť 5 je záver.

1 METÓDY DETEKCIE OBJEKTOV

1.1 Predspracovanie obrazu

Pred samotnou aplikáciou detekčných metód je často nutné snímok upraviť. Táto potreba vyplýva z vlastností použitých metód ktoré sú z pravidla citlivé na určité aspekty bežne snímaného obrazu ako je napr. šum, zlé svetelné podmienky alebo farebný model použitý v snímku. Obraz preto musíme optimalizovať na konkrétne vlastnosti a citlivosti použitého detekotru.

1.1.1 Histogram

Histogram je jedna zo základných charakteristík obrazu ktoré berieme do úvahy pri predspracovaní snímku ktorá vyjadruje početné zastúpenie jednotlivých jasových zložiek v obraze od najmenšej po najäčšiu. Vzhladom na kanály možme u viackanálových obrazov určiť niekoľko druhov histogramov [18]:

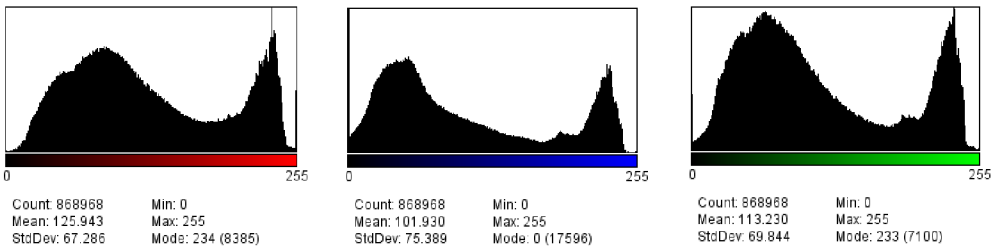
- histogram pre každý kanál zvlášť (napr. pre RGB model, sú to histogramy zvlášť pre červenú, modrú a zelenú) obr. 1.1
- histogram iba pre jasovú zložku obrazu obr. 1.2. Získa sa z váženého priemeru kanálov podľa vzorca:

$$Y(r, g, b) = 0,299\, r + 0,587\, g + 0,114\, b \tag{1.1}$$

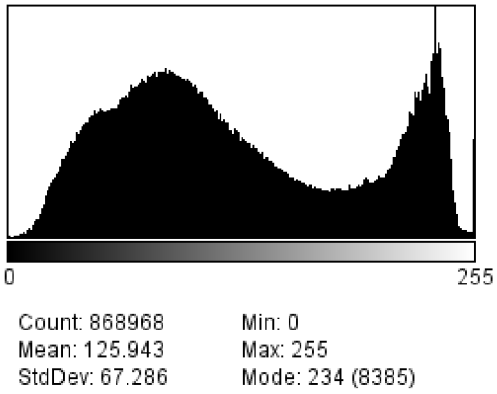
- priemer jednotlivých histogramov

Pri histogramoch obr. 1.1 a 1.2 sú uvedené štatistické hodnoty:

1. Count: celkový počet vzorkov
2. Mean: Priemerná hodnota
3. StdDev: Štandardná odchýlka
4. Min: Minimálna hodnota
5. Max: Maximálna hodnota
6. Mode: Najčastejšie sa vyskytujúca hodnota(počet výskytov)



Obr. 1.1: Histogramy RGB



Obr. 1.2: Histogram jasovej zložky

Obr. 1.3: Zdrojový snímok pre histogramy 1.1 1.2

1.1.2 Ekvalizácia histogramu

Ekvalizáciou sa snažíme dosiahnuť stav, kedy bude histogram obsahovať rovnomerné zastúpenie všetkých jasových zložiek. Tým dôjde aj ku zvýšeniu kontrastu. Ekvalizovaný histogram dostaneme transformáciou pôvodného obrazu $H(p)$ s jasovou stupnicou $p = \langle p_0, p_k \rangle$ na obraz $G(q)$ so stupnicou $q = \langle q_0, q_k \rangle$ pričom chceme aby bolo zobrazenie p a q monotónne:

$$\sum_{i=0}^{k} G(q_i) = \sum_{i=0}^{k} h(p_i) \tag{1.2}$$

Sumy v rovnici 1.2 sú diskrétne distribučné funkcie. Histogram $G(q)$ má rovnomerné rozdelenie f s konštantnou hustotou pravdepodobnosti.

$$f = \frac{N^2}{q_k - q_0} \tag{1.3}$$

Po dosadení do rovnice 1.3 za ľavú stranu rovnice 1.2 dostaneme

$$N^2 \int_{q_0}^{q} \frac{1}{q_k - q_0} d = \frac{N^2(q - q_0)}{q_k - q_0} = \int_{p_0}^{p} H(s)ds \tag{1.4}$$

Transformácia t je

$$q = \tau(p) = \frac{q_k - q_0}{N^2} \int_{p_0}^{p} H(s)ds + q_0 \tag{1.5}$$

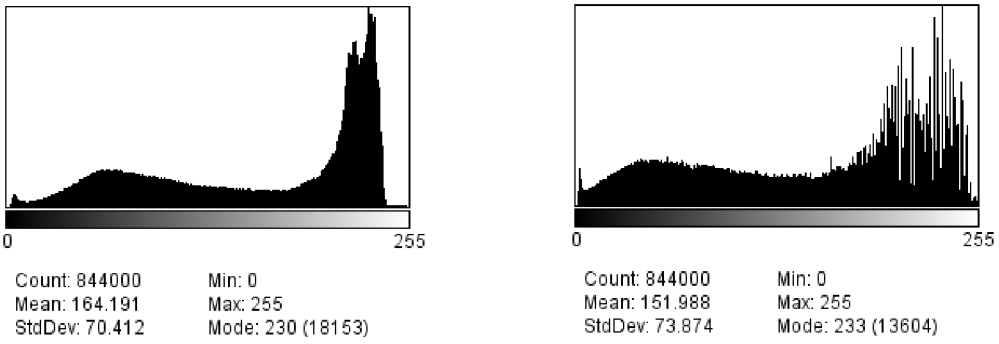
a z nej aproximácia bude

$$q = \tau(p) = \frac{q_k - q_0}{N^2} \sum_{i=0}^{p} H(i) + q_0 \tag{1.6}$$

.

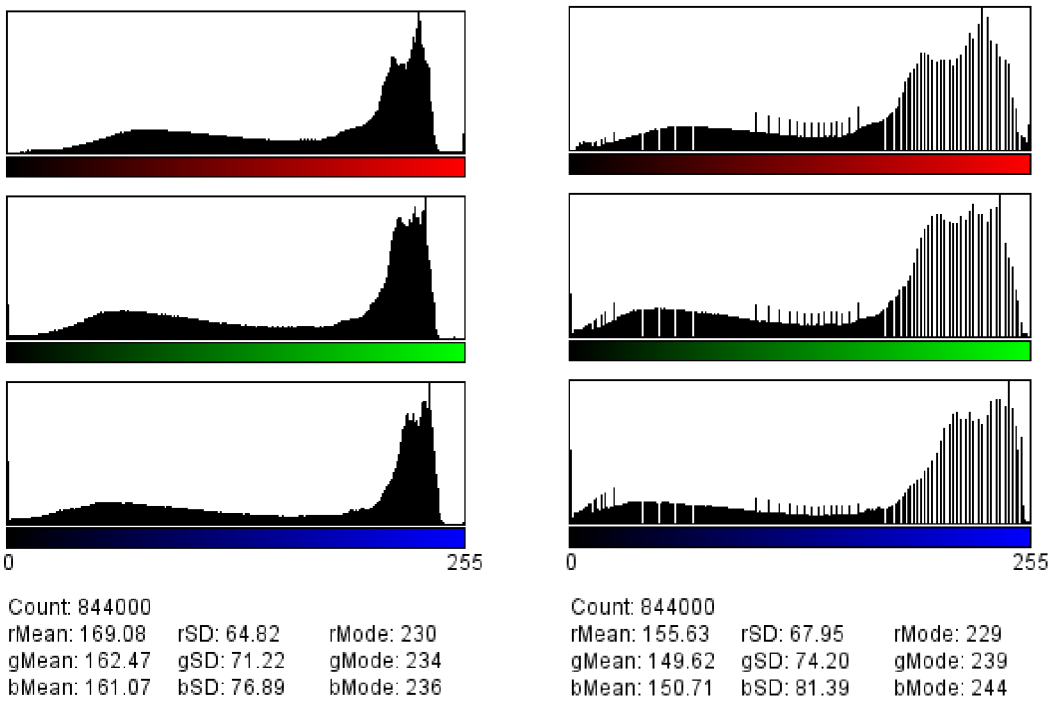
Dôsledky ekvalizácie

Ako sme povedali v časti 1.1.2, úpravou histogramu možme dosiahnuť zlepšenie jasových vlastností obrazu. Ako môžme vidiet na histogramoch obr. 1.4a a 1.5a, a aj zdrojovom obraze obr. 1.6a, je jasové spektrum značne presýtené vo vyšších hodnotách jasu (na obrázku obr. 1.6a je to hlavne oblasť padajúcej vody). Po ekvalizáciou sme matematickou operáciou z kapitoly 1.1.2 dosiahli vyrovnanie priemernej hodnoty jasových zložiek v celom rozsahu histogramu obr. 1.4b a 1.5b a ako môžme vidieť aj na obrázku obr.1.6b, v miestach padajúcej vody je už možné rozoznať štruktúry vody. Na histogramoch po ekvalizácii (hlavne pri jednotlivých farebných zložkách) však môžme pozorovať úplné vymizuntie niektorých hodnôt v dôsledku zaokrúľovania výsledkov matematických operácií.



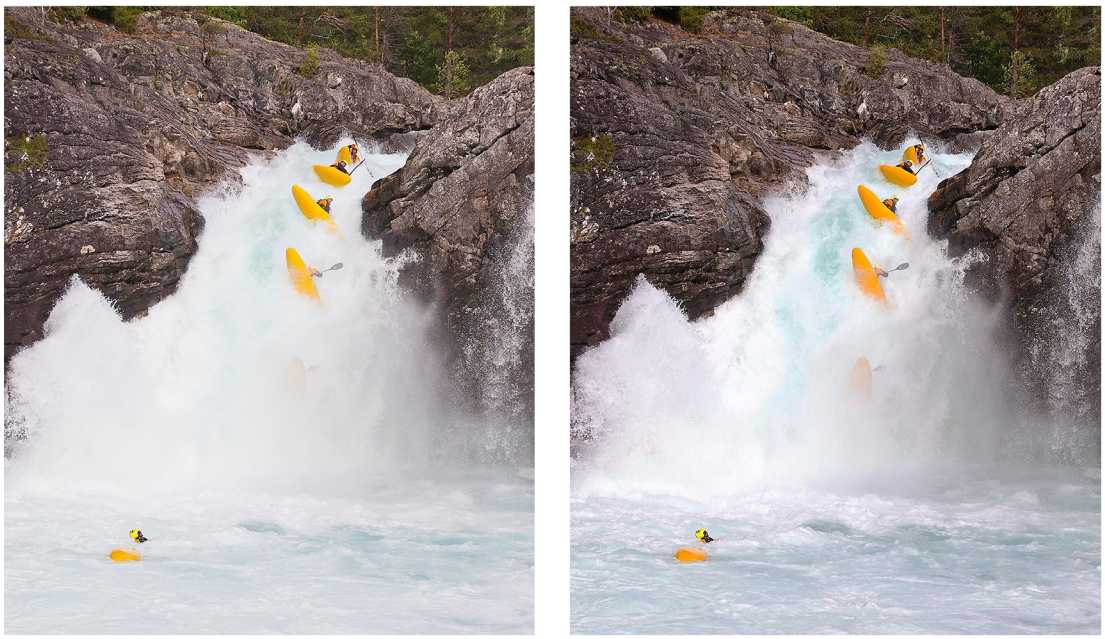
(a) Pred ekvalizáciou (b) Po ekvalizácii

Obr. 1.4: Jasové histogramy



(a) Pred ekvalizáciou (b) Po ekvalizácii

Obr. 1.5: Histogram RGB zložiek



(a) Pred ekvalizáciou

(b) Po ekvalizácii

Obr. 1.6: Pôvodný obraz

1.1.3 Normalizácia histogramu

Normalizáciou histogramu rozumieme matematickú operáciu ktorou dosiahneme lepšiu kvalitu obrazu. Použitím transformácie:

$$q = \tau(p) = (p - p_0)\frac{q_k - q_0}{p_k - p_0} + q_0 \tag{1.7}$$

sa nám podarí roztiahnuť histogram na novú jasovú stupnicu $q = \langle q_0, q_k \rangle$. [5]

1.1.4 Farebné modely

Rôzne detektory vyžadujú rôzne farebné modely. Pre detektory farieb je nutné aby obraz obsahoval všetky farby. Detektory hrán a POI naopak pracujú hlavne s jasovými zložkami a farby sú pre nich nadbytočné. [11]

Rozlišujeme tieto farebné modely [18]:

- RGB – Red, Green, Blue (Jedná sa o aditívny model - predstava celkom zatemnej miestnosti a reflektorov RGB)
- RGBA – model RGB rozšírený o α-kanál ktorý určuje miernu priehľadnosti, teda pomer v akom sa mieša daná hodnota RGB s farbou pozadia.
- CMY – Cyan, Magneta, Yellow (Jedná sa o substraktívny model - predstava celkom svetlej miestnosti a farieb CMY)

- CMYK – CMY rozšírený o farbu Key (blacK) kvoli skutočnosti, že reálne zmiešanie farieb CMY vytvorí špinavo hnedú.
- HLS, HSB - modely obsahujúce trojicu zložiek.
  1. Hue – odtieň (hlavná spektrálna zložka, určuje farebný tón)
  2. Saturation – sýtosť (živosť farby)
  3. Brightnes – svetlosť[1] (jas)
- YC_rC_b – farebný model v ktorom je farba reprezentovaná jedným jasovým (Y) a dvomi chrominančnými parametrami (C_r, C_b). Prepočet z RGB na YC_rC_b je pre 8bitové zobrazenie farieb podľa vzorca:

$$\begin{bmatrix} Y' \\ C_r \\ C_b \end{bmatrix} = \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 0,299 & 0,587 & 0,114 \\ -0,168736 & -0,334264 & 0,5 \\ 0,5 & -0,418688 & -0,081312 \end{bmatrix} \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} \tag{1.8}$$

1.1.5 Metódy na odstránenie šumu

Metóda priemerovania

Postupným prechádzaním obrazu sa dorátavajú nové hodnoty pixelov ktoré sú určené z pixelov zakrytých filtračnou maskou. Hodnota sa získava zo vzorca:

$$f(x,y) = \frac{1}{(2k+1)(2l+1)} \sum_{i=-k}^{k} \sum_{j=-l}^{l} f(k+i, l+j) \tag{1.9}$$

kde f*(x,y)* je jas pre bod o súradniciach *(x,y)*.

Gaussov filter

Má podobné vlastnosti ako vyššie spomenutá metóda prostého priemerovania, ale tvar masky je vypočítavaný z tvaru Gaussovej krivky podľa rovnice [14]

$$g(x,y;\sigma) = \frac{1}{2\pi\sigma^2} \mathrm{e}^{-\frac{x^2+y^2}{2\sigma^2}} \tag{1.10}$$

kde x je horizontálna a y vertikálna vzdialenosť od pôvodného bodu. σ reprezentuje štandardnú odchýlku hodnôt pre gaussovu funkciu.

Mediánový filter

Nelineárny filter používaný na vyhladzovanie, vhodný hlavne na elimináciu impulzného šumu. Jeho cieľom je elimonovat veľké jasové rozdiely v okolí zadaného bodu pomocou filtračnej masky. Body ktoré do nej spadajú sa usporiadajú podľa veľkosti. Nová hodnota bude medián tejto postupnosti. [5]

[1] pre model HLS je tento parameter Lightness – svetlosť ktorý vyjadruje polohu medzi bielou a čiernou.

1.2 Detekcia významnej farby

Metóda spočíva v lokalizácii skupín pixelov s vopred definovanou farbou. Farba musí byť jedinečná (v našom prípade to bude farba kože ruky). Táto metóda má však viacero obmedzení ktoré kladú na vybranú farbu tieto podmienky:

- farba by sa mala vyskytovať iba v jednej oblasti snímku,
- farba by mala byť dostatočne odlišná od iných farieb, kvoli možnej tolerancii na chybu v dosledku výskytu skreslenia farby napr. osvetlením,
- jednoznačné umiestnenie v obraze (analýza vzoru musí vždy dôjsť k rovnakému záveru).

Pri aplikácii tejto metódy je treba najprv vyhodnotiť farby referenčného vzoru, z ktorého budeme vychádzať. Najdôležitejším je percentuálne zastúpenie výskytu farby. Ak je farba sústredená v jedinej oblasti, pravdepodobne sa nebude jednať o pozadie, ale o hľadaný objekt.

Metóda nerozlišuje hrany ani rohy objektov a preto by bola pre detekciu ruky a giest nevhodná a neperspektívna. Bolo by však možné ju použiť ako doplnkovú. [11]

1.3 Detekcia hrán a rohov

Rohové a hranové metódy detekcie objektov najčastejšie pracujú na príncípe analýzy rozdielu lokálnych jasových zložiek v jednotlivých pozíciach obrazu a ich blízkom okolí. Tieto detektory sú rýchle, avšak ich presnosť je slabá. Sú vhodné pre zbežnú a nie velmi dôkladnú detekciu.

1.3.1 Moravcov detektor

Pri svojej činnosti používa malé okno ktoré posúva vo všetkých smeroch, a zisťuje zmenu priemernej intenzity jasu podľa vzorca:

$$E(x,y) = \sum_{u,v} [I(x+u, y+v) - I(x,y)]^2 \text{ pre } u,v \in \{-1,0,1\} \tag{1.11}$$

Kde *I(x,y)* je priemerná hodnota prvého rámca a *I(x+u,y+v)* hodnota rámca po posunutí. *E(u,v)* je matica kvadrátov rozdielov týchto dvoch hodôt. Nevýhoda tohto detektoru je citlivosť na smer otočenia. Z rovnice 1.11 vyplýva citlivosť hlavne v násobkoch 45°. [9]

1.3.2 Detektor MMIO

Výlepšený Moravcov detektor obohatený o MMIO operátor. Veľkosť okna je 11 x 11 px a posuv okna je realizovaný v 4 smeroch (diagonálny, antidiagonálny, horizontálny a vertikálny).

1.3.3 Detektor Hessian

Detektor je postavený na symetrickej Hessianovej matici ktorá používa druhé derivácie jednotlivých premenných v diagonále, a zmišané mimo nej.

$$Hf(x,y) \equiv \begin{bmatrix} \frac{\delta^2 f}{\delta x^2} & \frac{\delta^2 f}{\delta x\, \delta y} \\ \frac{\delta^2 f}{\delta x\, \delta y} & \frac{\delta^2 f}{\delta y^2} \end{bmatrix} \tag{1.12}$$

Po prepočítaní tejto matice pre každý bod obrazu je vypočítaný jej determinant a na základe toho sú určené POI. Výsledné body brané do úvahy sú však až tie, ktorých hodnota presiahla danú prahovú hodnotu (treshold) určenú externým vstupom.

1.3.4 Detektor Harris

Harrisov detektor je odvodený od Moravcovho detektoru časť 1.3.1. Jedná sa o rohový detektor ktory vyhľadáva význačné body POI, avšak od Moravcovho sa líši vylepšenou invariantnostou aj o iné uhly ako 45°.

Detektor je však velmi citlivý na šum. Preto je vhodné pred spracovaním týmto detektorom snímok rozmazať – aby sme šum eliminovali – low-pass filtrom napr. konvolúciou Gaussovým filtrom. Musíme však brať v úvahu že prílišné vyhladenie zníži výraznosť hrán, a teda sa zníži schopnosť detektora výhľadávať ich. [7]

1.3.5 Detektor BLOB

Detekory typu Binary Large Object (BLOB) sú zamerané na vyhľadávanie bodov a oblastí obrazu ktoré sa výrazne odlišujú od zvyšku obrazu vlastnostami ako jas alebo farba v porovnaní s pozadím. Realizácia prebieha kombináciou niektorých z predchádzajúcich detektorov. Tento druh detektorov schopný pracovať iba s jednofarebným čierno-bielym obrazom napr. obr. 3.7.

Objekty ktorý tieto detektory vyhľadávajú sa nazývajú BLOBy.

1.4 Detektory SURF

Metódy detektorov SURF vychádzajú pôvodne z detektorov SIFT ktoré sú založené na rovnakých princípoch, avšak používajú trochu iné postupy a sú schopné dosiahnuť

lepšie výsledky rýchlejšie. V súčasnosti existuje niekoľko druhov detektorov pričom sa využívajú rôzne metódy detekcie týchto bodov. Najčastejšie sa stretávame s detektormi používajúcimi Hessianovu maticu a metódu Harris(Laplace).

Detektory SURF sú postavené na vyhľadávani význačných bodov v obraze ako sú napríklad rohy, hrany prípadne prudké zmeny chromatických zložiek ktoré sa dajú jednoducho vektorovo popísať a fungujú v 3 krokoch:

1. Vyhľadanie
2. Popis
3. Zrovnávanie s predlohou

Obrovskou výhodou týchto detektorov je výpočetná rýchlosť ich invariantnosť voči otočeniu alebo priblíženiu / oddialeniu obrazu vďaka vektorovému popisu POI v obraze.

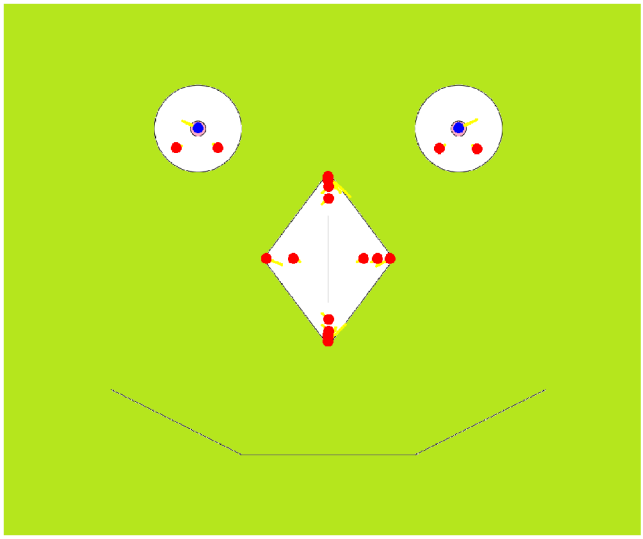
1.4.1 Princíp SURF

Metóda SURF využíva na detekciu POI body s maximálnym determinantom Hessianovej matice 1.12. Aby bola metóda invariantná na rotáciu alebo mierku obrazu, mala by byť detekcia bodov 100% reprodukovatelná. Algoritmus pracuje v 4 krokoch:

1. Výpočet integrálu vstupného obrazu
2. Vyhľadanie POI:
    - Určenie diskrétnych hodnôt Hessianovho operátoru v niekoľkých mierkach použitím filtrov.
    - Výber maximálnych hodnôt determinantov Hessianovej matice 1.12 a následná filtrácia na základne vstupnej úrovne (treshold) zadanej užívateľom.
    - Spresnenie získaných bodov kvadratickou interpoláciou.
    - Uloženie bodov s Laplaceovým operátorom.
3. Výpočet lokálnych deskriptorov:
    - Odhad dominantnej orientácie každého POI

Nižšie sú reálne príklady vizualizácie detekcie POI v obraze.

Aj keď sa javí táto metóda ako najvhodnejšia a najpresnejšia, práve kvôli extrémne vysokej citlivosti na detaily sme po testoch usúdili, že nie je možné zreprodukovať gestá ľudskou rukou natoľko presne aby bola táto metóda použiteľná. [3], [4], [17]



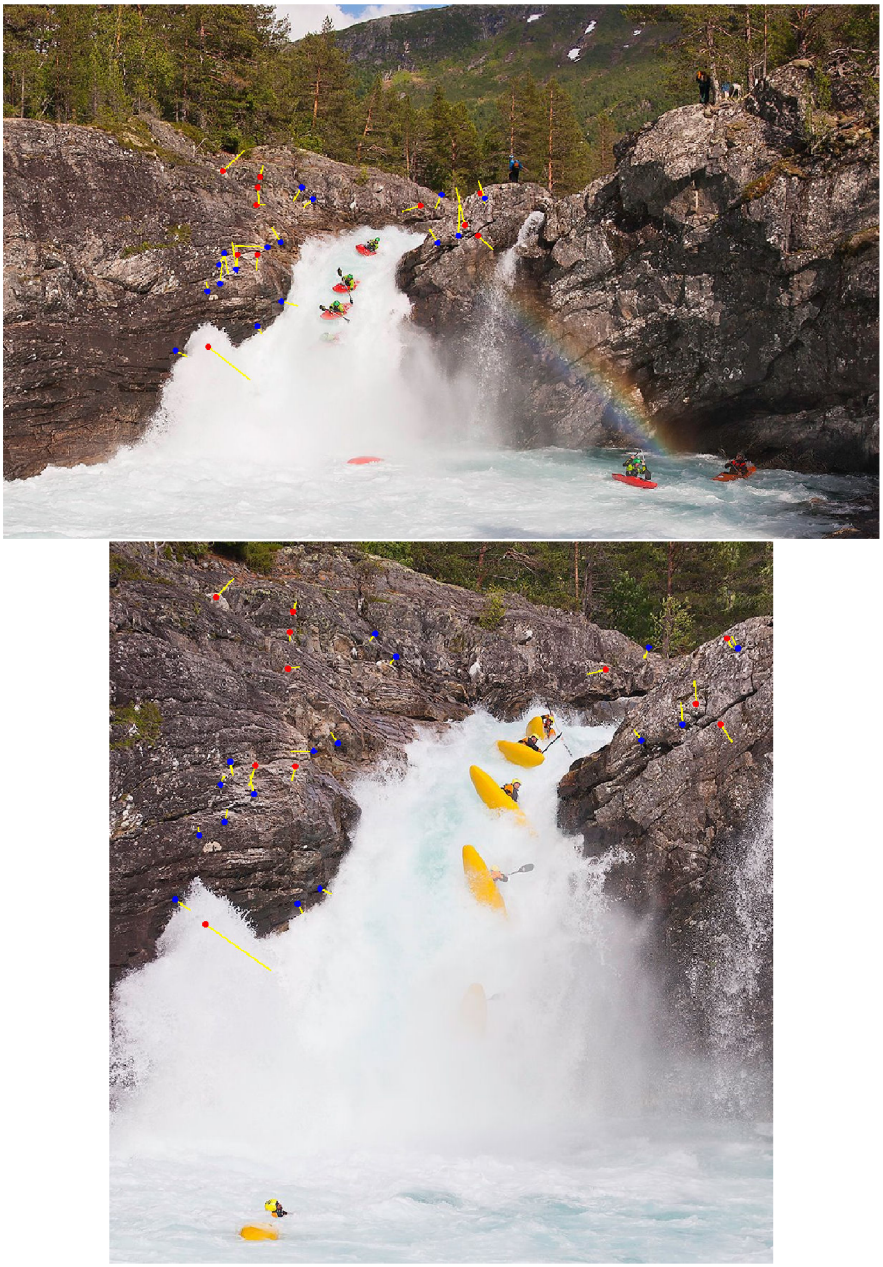
Obr. 1.7: Vizualizácia SURF detektoru Hessian POI pomocou programu ImageJ na obrázku

1.5 Zrovnávacia detekčná metóda

Patrí medzi najjednoduchšie realizovatelné metódy detekcie známeho objektu. Funguje na princípe porovnávania vhodne upraveného a normalizovaného obrazu snímaného objektu a rovnako upraveného obrazu objektu ktorý chceme detekovať. Je však najviac závislá na kvalite obrazu a nutných grafických a normalizačných upravách obrazu pred samotným porovnaním s predlohou. Takýto detektor funguje v 3 hlavných krokoch:

1. Snímanie objektu: objekt je nasnímaný digitálnou kamerou z ktorej je obraz v podobe snímku odoslaný do samotného detektoru.
2. Normalizácia obrazu:
   (a) Odstránenie šumu
   (b) Odstránenie pozadia
   (c) Odstránenie farieb a stupňov šedi
   (d) Zmena orientácie (vhodné otočenie)
   (e) Zmena veľkosti (zväčšenie/zmenšenie)
3. Detekcia objektu: normalizovaný obraz je porovnávaný s predlohou.

V poslednom kroku je nutné vhodne zvoliť percentuálnu hodnotu zhody, pri ktorej je objekt vyhodnotený ako detekovaný, aby boli čo najviac eliminované chybné pozitívne alebo negatívne detekcie.



Obr. 1.8: Vizualizácia SURF detektoru Hessian POI pomocou programu ImageJ na obraz z jedného miesta v dvoch rôznych mierkach

2 SPRACOVANIE OBRAZU V JAZYKU JAVA

Existuje rada knižníc zaoberajúcimi sa spracovaním obrazu v jazyku JAVA. V tejto kapitole bude popísaný vyber knižníc vhodných pre účely tejto práce. Zoznam najvýznamnejších knižníc pre jednotlivé potreby tejto práce:

1. Snímanie obrazu: Xuggler, LTI-CIVIL, JMF, FMJ, WebcamCapture.
2. Úpravy obrazu: ImageJ, BoofCV, Marvin IPF, JavaCV.
3. Detekcia: ImageJ, BoofCV, JavaCV.

V dalších časiach tejto kapitoly bude odôvodnený výber použitých knižníc.

2.1 Snímanie obrazu

Pre snímanie sme mali na výber knižnice: Xuggler, LTI-CIVIL, JMF, FMJ, WebcamCapture.

	Xuggler	LTI-CIVIL	JMF	FMJ	WebcamCapture
Tutoriály	1	0	0	0	0
Podpora	1	1	0	1	0
Príklady	1	1	1	1	1
API	1	1	1	1	1
Funkčnosť	1	1	0	1	1
Dostupnosť	1	1	1	1	1
URL	[15]	[12]	[16]	[13]	[8]
Hodnotenie	**6**	**5**	**3**	**5**	**4**

Tab. 2.1: Hodnotenie jednotlivých knižníc podľa kritérií

Zo zhodnotenia tab. 2.1 vyplýva, že najlepšia voľba bude pre nás knižnica Xuggler. Táto knižnica ako aj niektoré z vyššie menovaných používa pre záznam, prehrávanie a kódovanie videa kodeky z knižnice FFmpeg[1].

2.2 Úprava obrazu

Pre úpravu obrazu sme mali na výber knižnice: ImageJ, BoofCV, Marvin IPF, JavaCV. Zo zhodnotenia tab. 2.2 vychádza najvhodnejšie knižnica ImageJ ktorá obsahuje všetky potrebné metódy na úpravu obrazu. Knižnica používa svoj vlastný datový typ ImagePlus, ktorý je jednoducho upravovateľný, a je bez problémov konvertovatelný z a do pôvodného obrazového datového typu Javy BufferedImage.

[1] http://ffmpeg.org/

	ImageJ	BoofCV	Marvin IPF	JavaCV
Vhodné metódy	1	1	0	1
Tutoriály	1	0	0	1
Podpora	1	1	0	1
Príklady	1	1	0	1
API	1	1	1	1
Funkčnosť	1	1	1	0
Dostupnosť	1	1	1	1
URL	[10]	[2]	[6]	[1]
Hodnotenie	**7**	**6**	**3**	**6**

Tab. 2.2: Hodnotenie knižníc pre úpravu obrazu

2.3 Detekcia objektu v obraze

Pre detekciu sme mali na výber tieto knižnice:ImageJ, BoofCV, JavaCV. V zhodnotení tab. 2.3 vidíme zhodný počet bodov pre ImageJ[2] a BoofCV. Z dôvodu použitia knižnice ImageJ aj v predchádzajúcej časti programu sme usúdili, že bude najjednoduchšie v jej používaní pokračovať aby sme sa vyhli zbytočnej a výpočtovo náročnej konverzii dátových typov medzi knižnicami.

	ImageJ	BoofCV	JavaCV
Vhodné metódy	1	1	1
Tutoriály	1	1	1
Podpora	1	1	1
Nepotrebuje ext. knižnice	1	1	0
API	1	1	1
Funkčnosť	1	1	0
Dostupnosť	1	1	1
URL	[10]	[2]	[1]
Hodnotenie	**7**	**7**	**5**

Tab. 2.3: Hodnotenie knižníc pre detekciu objektu v obraze

[2]Samotná knižnica neobsahuje metódy na prácu s objektami typu BLOB popísanými v časti 1.3.5, je však dostupný jej plugin IJBlob.

3 DETEKTOR GIEST ĽUDSKEJ RUKY

Zo zadania práce vyplývalo naštudovať a zhrnúť problematiku detekcie objektov (v našom prípade ruky) v obraze čo sme v kapitolách 1 a 2 vysvetlili a popísali. Následne sme vytvorili program ktorý, v krokoch popísaných v tejto kapitole, bude vybraté snímky upravovať a skúmať a vyhodnocovať podľa daných kritérií.

V prvej časti 3.1 je popísané grafické užívateľské rozhranie. V druhej časti 3.2 je popísaný postup snímania, spracovania, zobrazenia a vybratia snímky ktorý bude upravovaný neskor. Tretia časť 3.3 sa zaoberá úpravou obrazu (odstránenie šumu, odstránenie pozadia, prevod do binárnej farebnej schémy. . .) do podoby vhodnej ku ďalšiemu spracovaniu. Vo štvrtej časti 3.4 sa dostaneme ku finálnej normalizácii obrazovej informácie a samotnej analýze obrazu podľa daných podmienok. V poslednej časti 3.5 je popísaná realizácia metódy určenej na počítanie vystretých prstov na ruke.

Program je logicky rozdelený do 4 hlavných tried:

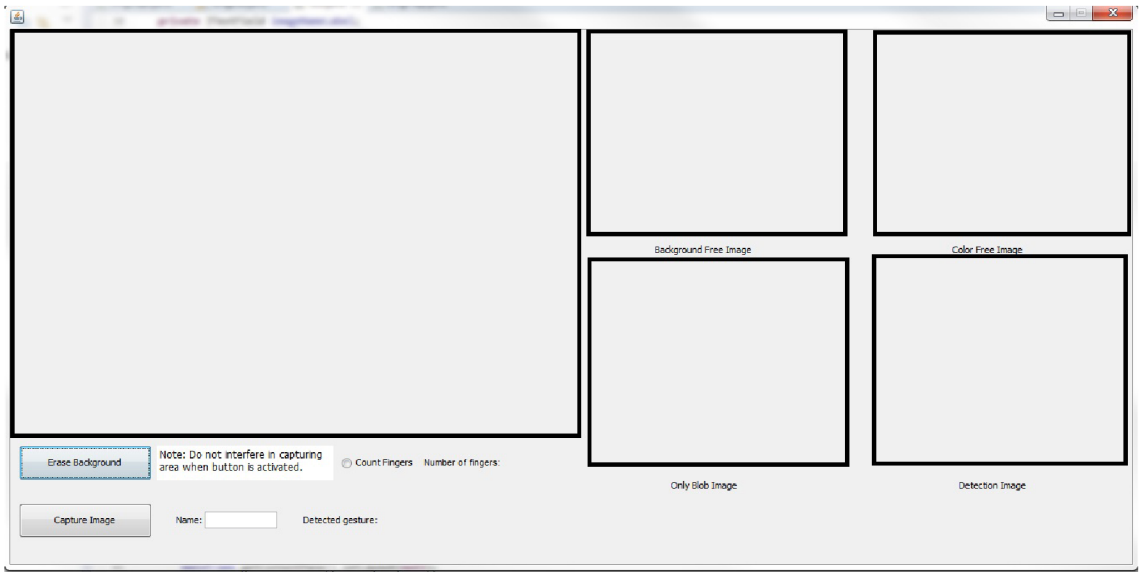
1. GUI - grafické užívatelské rozhranie kapitola 3.1
2. ImgCapt - Trieda na snímanie obrazu kapitola 3.2
3. ImgPrep - Trieda na hlavné úpravy obrazu pred detekciou kapitola 3.3
4. ImgDet - Trieda vykonávajúca samotnú detekciu 3.4

3.1 Graphical User Interface

GUI sme vytvorili pomocou vývojoveho prostredia WindowBuilder. Je tvorené hlavným objektom JFrame do ktorého sú vložené dalšie zobrazovacie a ovládacie prvky prvky:

- JButton
    1. Erase Background - Tlačidlo ktoré vynuluje riadiaci čítač *counter2* a premennú *first* na počiatočné hodnoty. Toto sposobí výmenu referenčného obrazu za aktuálny pri procese odstránenia pozadia z obrazu.
    2. Capture Image - Tlačidlo ktoré volá metódu *imageSaver*. Metóda uloží a pomenuje aktuálne zobrazované gesto v tvare gest.***.bmp - bitmapu.
- JRadioButton - Zaškrtávací button, ktorý nadobúda iba hodnot *true* alebo *false*. Používa sa na aktiváciu metódy *fingerCounter(onlyBlobImage)*.
- JPanel - Multifunkčný objekt grafickej knižnice Swing, do ktorého možu byť vložené zobrazovacie objekty knižnice ImageJ, prípadne íné grafické I/O objekty.
- JLabel - Grafický objekt zobrazujúci iba jeden riadok textu pomocou metódy setText(string s). Neumožňuej však interakciu s užívateľom.

- JTextPane - Grafický objekt umožnujúci zobrazenie textu aj vo viacerých riadkoch, bez možnosti interakcie s užívateľom.
- JTextField - Grafický objekt ktorý umožňuje načítať reťazec znakov zadaných užívateľom.
- ImageCanvas - Objekt triedy ImageCanvas, balíčku ImageJ, ktorý je vložený do JPanelu. Slúži na samotné vykreslenie obrazu.



Obr. 3.1: GUI - Graphical User Interface

3.2 Snímanie, dekódovanie a zobrazenie

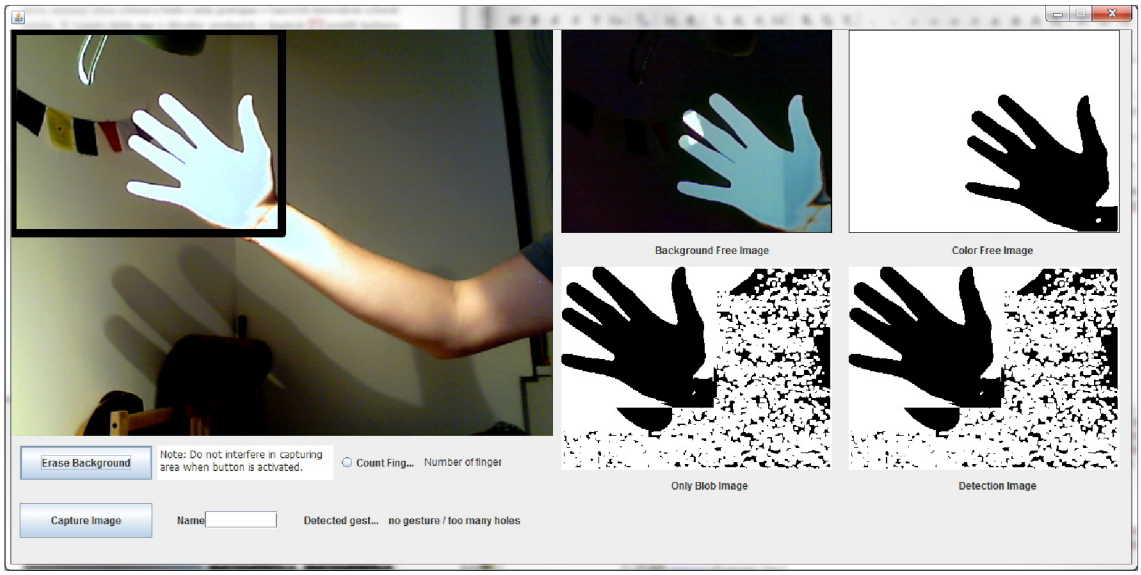
O snímanie a dekódovanie obrazu sa stará trieda *ImgCapt*, ktorá vznikla modifikáciou demonštračnej triedy projektu Xuggler ktorá je šírená pod licenciou General Public License (GPL) 3.

Pri realizácii sme sa stretli s problémami zapríčinenými build-in webkamerou v notebooku. Tieto problémy súviseli s ovládačmi v nami použitých balíčkoch a nebolo možné ich efektívne odstrániť pre dané zariadenie, preto sme museli použiť externú USB webkameru ktorá už fungovala podľa očakávaní. Snímané video by malo mať tieto parametre:

- Rozlíšenie: 640x480 px,
- Framerate: 30 fps

Oba tieto parametre[1] sa dajú nastaviť v programe ale sú limitované vlastnými schopnosťami kamery a dekodéru.

[1]Hodnota 30 fps zadaná ako parameter IMetaData objektu nie je reálne dosahovaná. Nami nameraná hodnota fps dosahovala 5-10 fps.

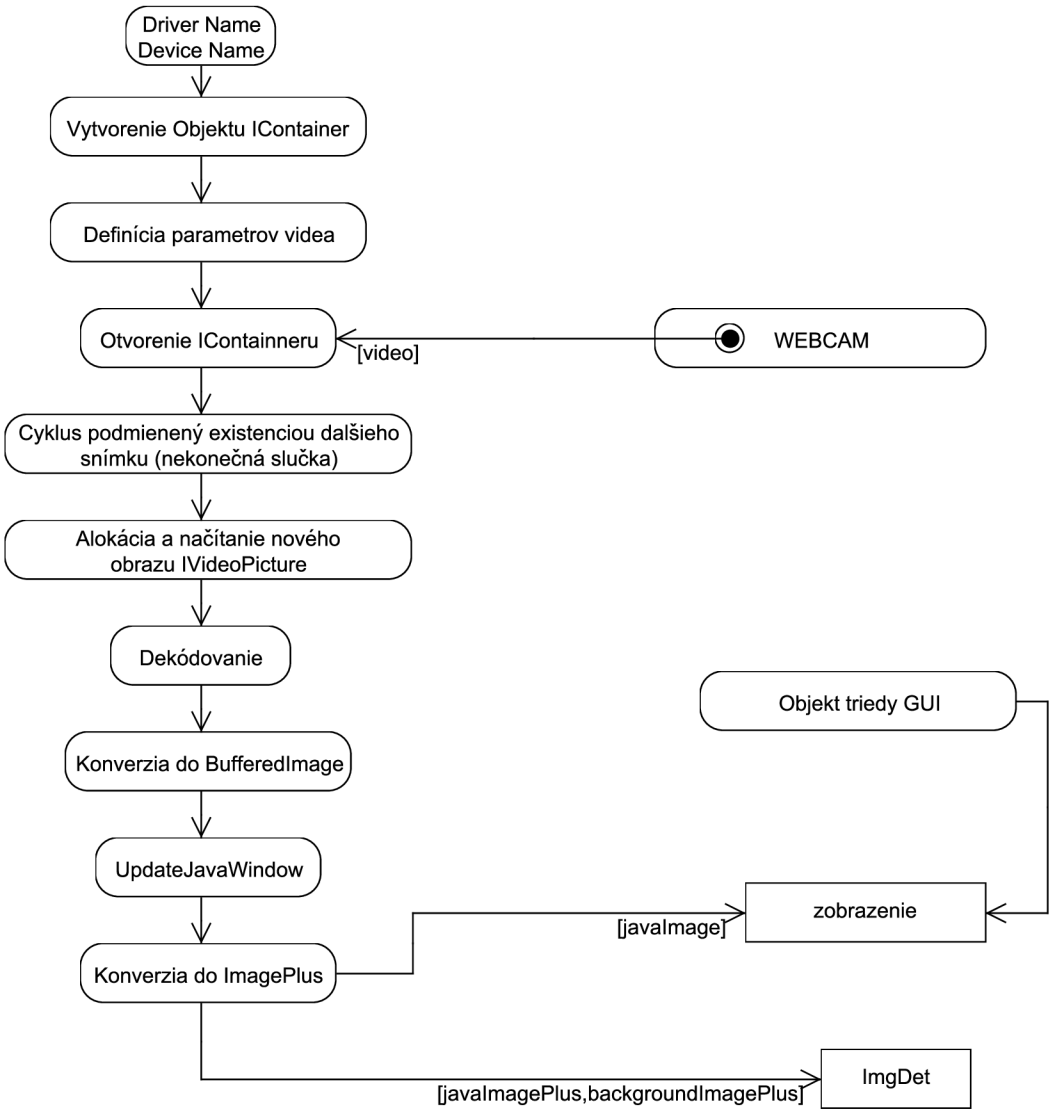


Obr. 3.2: Ľavé okno je live videostream, vpravo vidíme jednotlivé fázy úpravy obrazu

Hlavná časť triedy sa nachádza v metóde *Main* ktorá vykonáva všetky dôležité kroky od samotného snímania obrazu, jeho prekódovaním až po zobrazenie. Metóda *openJavaWindow* slúži na vytvorenie používateľského rozhrania triedy GUI do ktorého je následne obraz nahrávaný metódou *updateJavaWindow*.

Vstupnými premennými do programu je označenie zariadenia z ktorého chceme snímať obraz a meno ovládača k tomuto zariadeniu. Následne sa vytvorí objekt container triedy *IContainer* pomocou ktorého sú definované parametre videa. Z objektu container sa vyberie prvý video stream z ktorého sú v nekonečnom cykle vyberané pakety. Z nich sa v ďalšom kroku dekódujú snímky ktoré sa konvertujú do typu BufferedImage. Obraz je odoslaný do metódy *updateJavaWindow* ktorá ho skonvertuje na ImagePlus , zobrazí , a pošle na dalšie spracovanie.

Čierny rám v snímanom obraze slúži na grafické znázornenie oblasti do ktorej treba gesto umiestniť. Vývojový diagram Triedy *ImgCapt* je na obr. 3.3.



Obr. 3.3: Vývojový diagram

3.3 Spracovanie obrazu

O spracovanie obrazu sa stará trieda *ImgPrep* ktorá implementuje rozhranie Runnable vďaka čomu metódy v nej môžu byť spracovávané paralelne so zobrazovacou triedou bežiacou v pôvodnom hlavnom vlákne. Nové vlákno sme nazvali *editor*. Týmto chceme docieliť rýchlejšieho spracovania obrazu a menšieho oneskorenia detektoru.

Triedu voláme konštruktorom ImgPrep(ImagePlus, ImagePlus) kde prvý argument ImagePlus je aktuálny obraz z kamery a druhý argument ImagePlus je referenčný obraz pozadia.

Spracovanie obrazu prebieha v 3 hlavných krokoch:

1. Odstránenie šumu[2]
2. Odstránenie pozadia
3. Vytvorenie dvojfarebného obrazu

Vývojový diagram triedy *ImgPrep* je na obr. 3.4

Obr. 3.4: Vvojový diagram triedy *ImgPrep*

[2]Šum je odstranňovaný po každej operácii kvoli možnosti vzniku impulzného šumu pri matematických operáciách.

3.3.1 Odstránenie pozadia

Odstránenie pozadia prebieha v metóde *backgroundSubtractor* pomocou objektu triedy *ImageCalculator*. Objekt obsahuje metódu ktorá funguje ako dferenčný filter čo je pre naše potreby ideálne. Filter má dva vstupné obrazy, z ktorých vytvorí výsledný obraz odčítaním hodnot pixelov jedného obrazu od druhého.

```
ImageCalculator iC = new ImageCalculator();
backgroundFreeImage = iC.run("Diference create",captured,background);
```

Výsledný obraz by teda mal obsahovať iba to, čo doňho pridáme.

Obr. 3.5: Obraz u ktorého bolo odobraté pozadie diferenčným filtrom.

3.3.2 Odstránenie šumu

Odstránenie náhodného šumu je realizované mediánovým filtrom v metóde *medianCutter* objektom *ImageProcessor* ktorý je obsiahnutý priamo v datovom type ImagePlus. *ImageProcessor* obsahuje metódu *medianFilter()* ktorou docielime potrebný efekt. Princíp mediánového filtru ktorý je popísaný v kapitole 1.1.5

3.3.3 Vytvorenie dvojfarebného obrazu

Vytváranie dvojfarebného obrazu prebieha v metóde *biColorConvertor* v dvoch krokoch:

Konverzia farebného obrazu do odtieňov šedi

Konverzie farebného obrazu do odteňov šedi je docielené pomocou objektu *ImageConverter* metódou *convertToGray8*, ktorá vypočíta z farieb jednotlivých pixelov RGB ich výslednú jasovú zložku Y′ pomocou rovnice 1.1

Obr. 3.6: Obraz po prevode na stupne šedi

Pri pozornejšom preskúmaní je v obraze vidieť obrysy objektov z pozadia ktoré sú však veľmi slabo viditeľné, a budú odstránené vhodne zvolenou prahovacou hodnotou v nasledujúcom kroku.

Prahovaním na čierno-biely obraz

Rozdelením odtieňov šedi prahovacou hodnotou na dva extrémy(0 - čierna, 255 - biela) podľa jasu. Toto je realizované objektom *ImageConverter* metódou *autoTreshold()*, ktorá automaticky mení prahovaciu hodnotu podľa jasových pomerov v obraze. Všetky jasové hodnoty pod prahovacou hodnotou nadobudnú hodnotu 0 a všetky jasové hodnoty nad prahovacou hodnotou nadobudnú hodnotu 255. Týmto docielime dvojfarebný čierno-biely obraz napr. obr. 3.7.

Následne je ešte obraz invertovaný do stavu biele-pozadie, čierny-objekt aby ho nebolo nutné invertovať v triede *ImgDet*.

Obr. 3.7: Dvojfarebný obraz.

3.4 Detekcia

O detekciu správneho gesta sa stará trieda *ImgDet* ktorá implementuje rozhranie Runnable vďaka čomu beží súbežne s ostatnými vláknami. Nové vlákno sme nazvali *detector*.

Triedu voláme v triede *ImgPrep* konštruktorom ImgPrep(ImagePlus) kde argument je predspracovaný čierno-biely obraz obr. 3.7, ktorý je návratovou hodnotou metódy *biColorConvertor* triedy *ImgPrep*.

Pre správnu detekciu objektu je žiadúce predpripravený obraz z kapitoly 3.3 normalizovať na vhodný formát tak, aby ho bolo možné spoľahlivo porovnať s predlohou. Detekcia prebieha v 2 krokoch :

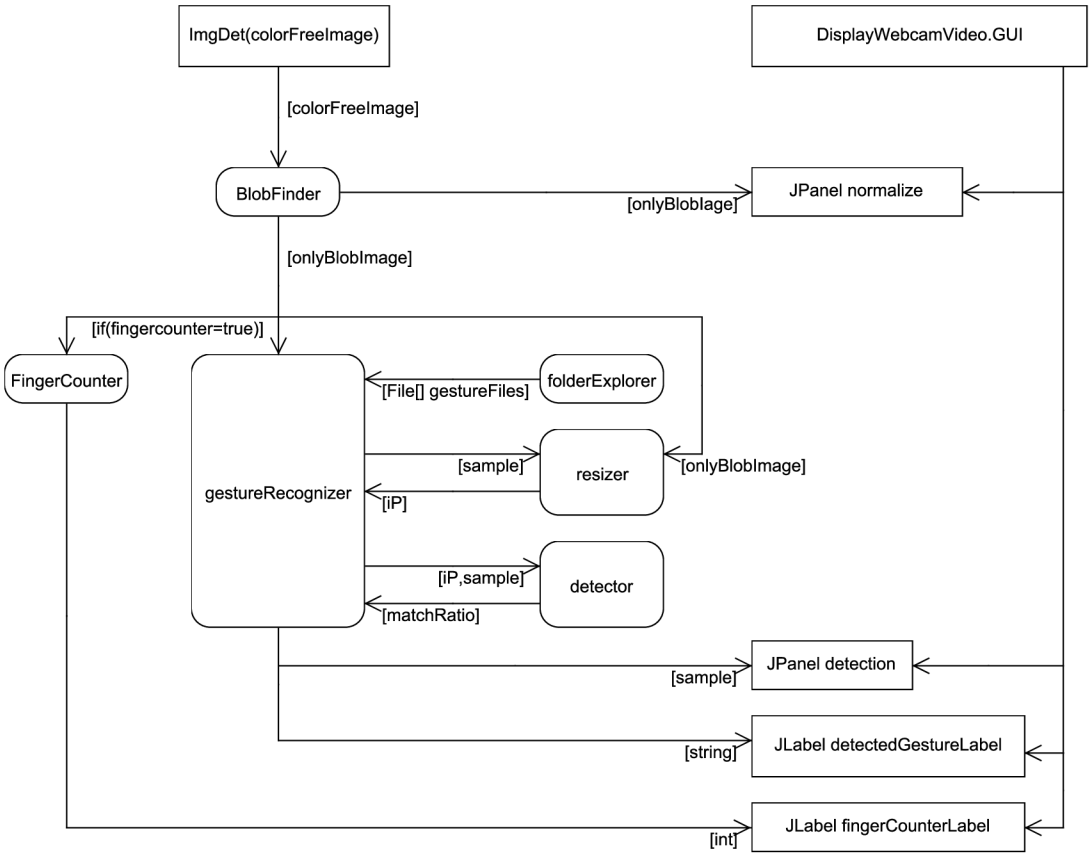
1. Vyhľadanie objektu metódou BLOB 1.3.5
2. Detekcia porovnaním s predlohou

V našom prípade dostávame z triedy *ImgPrep* čierno-biely obraz ktorý by mal obsahovať iba žiadúci objekt - ruku.

Vývojový diagram triedy *ImgDet* je na obr. 3.8

3.4.1 Vyhľadanie objektu metódou BLOB

Na manipuláciu s objektami BLOB sme použili knižnicu IJBlob ktorá obsahuje všetky potrebné metódy potrebné na vyhľadanie a manipuláciu.



Obr. 3.8: Vývojový diagram triedy *ImgDet*.

Na vyhľadanie a výber najvhodnejšieho (najväčšieho) BLOBu slúži metóda *blobFinder* ktorá pozostáva z 3 hlavných krokov:

1. Vyhľadanie objektov BLOB
2. Vyfiltrovanie najväčšieho
3. Vystrihnutie najväčšieho BLOBu

Vyhľadanie vhodných objektov

Vyhľadanie všetkých BLOBov v obraze je realizované pomocou metódy *findConectedComponents()* na objekt typu ManyBlobs ktorá dedí z triedy triedu Array.

```
ManyBlobs allBlobs = new ManyBlobs(iP);
allBlobs.findConnectedComponents();
```

Vyfiltrovanie najväčšieho

Filtrácia metódou *filterBlobs()* podľa veľkosti nám zaručí zmenšenie celkového počtu objektov iba na tie najväčšie podľa plochy. Prvý argument je spodná hranica

plochy BLOBu, hlavne kvôli častému výskytu drobných chýb v obraze. Druhý je horná hranica, ktorá je nutná pre vylúčenie pozadia ako najväčšieho BLOBu [3]. Posledný argument slúži na presnejší popis filtrovaných objektov, v našom prípade vyhľadávame uzavretú plochu.

```
allBlobs.filterBlobs(20000,280000, Blob.GETENCLOSEDAREA);
```

Následne pomocou iteračného cyklu `for` prehľadávame objekt ManyBlobs na najväčší pomocou metódy *getEnclosedArea()*

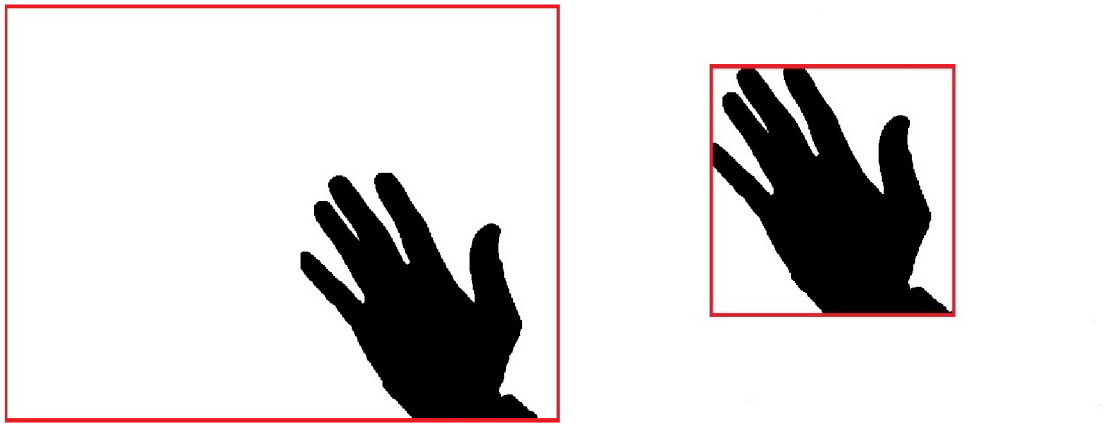
Vystrihnutie najväčšieho BLOBu

Vystrihnutie najväčšieho BLOBu prebieha získaním vonkajších obrysov blobu metódou *getOuterContours()* v podobe objektu typu polygon z ktorého následne pomocou metódy *getBounds()* dostaneme jeho ohraničenie v tvare rovnobežníka triedy Rectangle. Ten použijeme na vytvorenie oblasti Region of Interest (ROI).

```
iProcessor.setRoi(biggestBlob.getOuterContour().getBounds());
```

Oblasť ROI je možné následne pomocou objektu typu ImageProcessor a metódy *crop()* vystrihnúť a vytvoriť z nej nový objekt ImagePlus.

```
croppedBlobImage = new ImagePlus("croped", iProcessor.crop());
```



(a) Vstupný obraz z metódy ImgPrep (b) Výstupný obraz metódy blobFinder()

Obr. 3.9: Obraz pred a po orezaní BLOBu (červenou)

Tento obraz obr. 3.9b je zároveň aj návratovou hodnotou.

[3]Podľa autora pluginu by tento problém nemal nastávať, v našom prípade sa ináč vylúčiť nedal.

3.4.2 Detekcia porovnaním s predlohou

O samotnú detekciu objektu sa stará metóda *gestureRecognizer(ImagePlus)* ktorá používa 3 hlavné metódy:

1. folderExplorer()
2. imageResizer(onlyBlobImage, sample)
3. detector(onlyBlobImage, sample)

Metóda folderExplorer()

Umožňuje ľubovoľne meniť počet definovaných giest, bez nutnosti úpravy zdrojového kódu. Gestá sú ukladané ako rastrove mapy pomenované vo formtáte gest.***.bmp.

Metóda folderExplorer() prehľadáva zadaný priečinok a vyhľadáva súbory ktoré majú názov v zadanom tvare. Návratová hodnota je objekt File[] ktorý obsahuje podmienkam vyhovujúce položky vrátane absolútnej cesty k daným súborom. Z objektu listOfGestures[] typu File[] je následne pomocou metódy *getPath()* vytvorený nový objekt ImagePlus nazvaný sample ktorý obsauje vzor definovaného gesta.

```
sample = new ImagePlus(listOfGestures[i].getPath());
```

Metóda imageResizer(onlyBlobImage, sample)

Jednoduchá metóda ktorá slúži na prisposobenie rozmerov získaného objektu na rozmery vzoru. Na to používa metódu *resize()* objektu ImageProcessor.

Metóda detector(onlyBlobImage, sample)

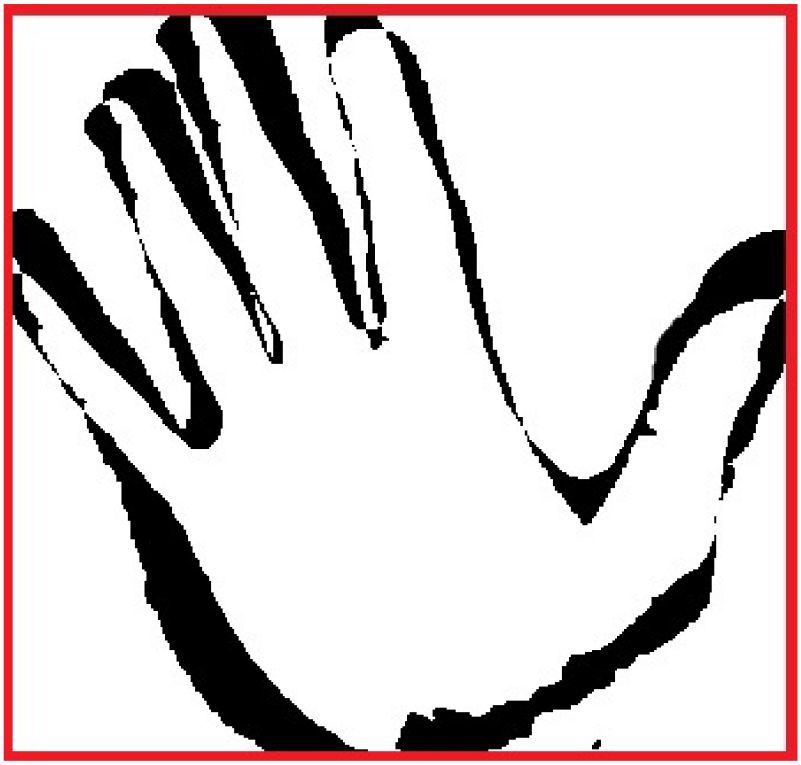
Hlavná porovnávacia metóda využíva na zisťovanie rozdielu obrazov na základe ich vzájomného odčítania číselný koeficient. Myšlienka porovnávacej metódy spočíva predpoklade, že po odčítaní 2 rovnakých obrazov od seba by výsledkom mal byť prázdny obraz. Keďže je tento stav ťažko dosiahnuteľný, uvažujeme presnosť vyjadrenú v % zhodných pixelov.

Na realizáciu tohto princípu metóda využíva differenčný filter triedy *ImageCalculator* použitý aj v metóde *backgroundSubtractor* triedy *ImgPrep*. Výsledný obraz po odčítaní je následne v iteračnom cykle kontrolovaný na pozostatky po odčítaní, ktoré vyjadrujú mieru zhody/nezhody s predlohou. Tie sa prejavia ako čierne oblasti. Výsledný koeficient zhody x_{match} je vyrátaný pomocou rovnice

$$x_{match} = \frac{count_{black} \times 100}{count_{total}}[\%] \tag{3.1}$$

kde $count_{black}$ je počet čiernych pixelov v obraze a $count_{total}$ je celkový počet pixelov obrazu.

Na základe koeficientu x_{match} z rovnice 3.1 sa určuje, či je dané gesto detekované. Príklad takéhoto obrazu[4] môžme vidieť na obr. 3.10.



Obr. 3.10: Výsledný rozdiel dvoch obrazov (čierna)

3.5 Metóda na počítanie prstov

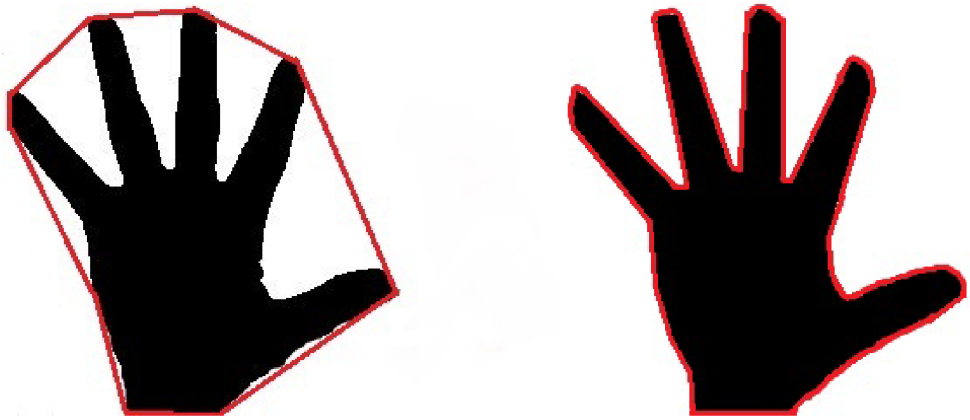
V triede detektoru *ImgDet* sme vytvorili metódu *fingerCounter(onlyBlobImage)* ktorá, za predpokladu že je v obraze ruka napr. obr. 3.7 je schopná spočítať vystreté prsty na ruke. Na aktiváciu slúži objekt JRadioButton umiestnený v ovládacej časti GUI.

Princíp metódy spočíva v spočítaní rozdielových oblastí medzi konvexnou a konkávnou krivkou blobu ruky ktoré môžeme vidieť na obr. 3.11b a 3.11a.

Vstupným argumentom do metódy je obraz ImagePlus *onlyBlobImage* ktorý je produktom metódy *blobFinder*. Metóda pracuje v 4 hlavných krokoch:

1. Vyhľadanie BLOBu
2. Vytvorenie oblastí ROI a ich vyplnenie farbou
3. Odčítanie obrazov
4. Vyhľadanie a spočítanie zvyšných BLOBov.

[4]Červené orámovanie je prítomné iba kvoli vizuálnemu rozlíšeniu okrajov.



(a) Konvexná krivka (b) Konkávna krivka.

Obr. 3.11: Krivky konkávnosti a konvexnosti (červenou)

Vyhľadanie BLOBu

Na vyhľadanie BLOBu sme opäť použili knižnicu IJBlob. Z Vytvorili sme objekt typu ManyBlobs a vyhľadali všetky BLOBy.

```
ManyBlobs allBlobs = new ManyBlobs(onlyBlobImage);
allBlobs.findConnectedComponents();
```

V dalšom kroku otestujeme objekt allBlobs, aby sme sa uistili, že nie je prázdny. Vyberieme prvý blob.

Vytvorenie oblastí ROI a ich výplň farbou

K tomuto účelu si vytvoríme dve kópie vstupného obrazu *convexIP* a *concaveIP*. Z blobu vytvoríme oblasti typu PolygonROI *concaveRoi* a *convexRoi*. *ConvexRoi* získame pomocou metódy *getConvexHull* ktorá vracia konvexnú krivku. *ConcaveRoi* zasa získame pomocou *getOuterContour()* ktorá vracia vonkajšie obrysy blobu, čo je vo svojej podstate viac kriviek konkávnosti spojených dohromady.

```
convexRoi = new PolygonRoi(blob.getConvexHull(),Roi.POLYGON);
concaveRoi = new PolygonRoi(blob.getOuterContour(),Roi.POLYGON);
```

Oblasti *convexRoi* pomocou metódy *setFillColor(java.awt.Color c)* nastavíme čiernu farbu výplne

```
convexRoi.setFillColor(java.awt.Color.BLACK);
```

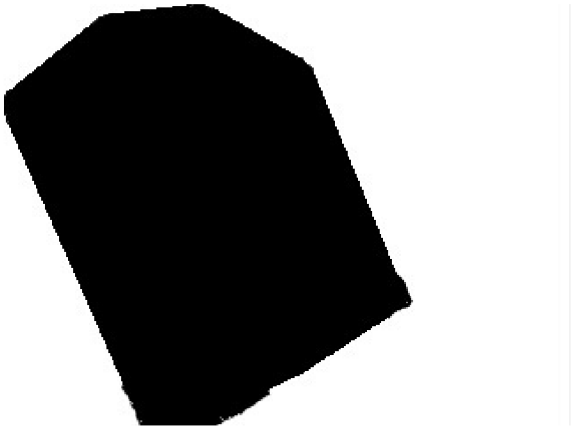
Obraz convexIP premaľujeme oblasťou ROI nastavenou farbou z predchádzajúceho kroku

```
convexIP.setOverlay(new Overlay(convexRoi));
convexIP = convexIP.flatten();
```

následne ešte obraz invertujeme aby po odčítaní boli rozdielové oblasti v požadovanej čiernej farbe

```
convexIP.getProcessor().invert();
```

Výsledok je obraz ako napr. na obr. 3.12



Obr. 3.12: Obraz po vyplnení konvexnej krivky

Odčítanie obrazov

Odčítanie je realizované objektom triedy *ImageCalculator* funkciou differenčného filtru rovnako ako v ďalších metódach tohoto projektu (napr. *ImgPrep.backgroundSubtractor(ImagePlus, ImagePlus)*, *ImgDet.detector(ImagePlus, ImagePlus)*). Výsledný obraz je na obr. 3.13.

Obr. 3.13: Obraz po odčítaní

Vyhľadanie a spočítanie zvyšných BLOBov

Vyhľadanie je realizované opäť metódou *findConectedComponents()* ktorej výsledok ešte vyfiltrujeme metódou *filterBlobs(min, type)*

```
ManyBlobs numberOfFingersBlobs = new ManyBlobs(numberOfFingersImage);
numberOfFingersBlobs.findConnectedComponents();
numberOfFingersBlobs.filterBlobs(200, Blob.GETENCLOSEDAREA);
```

kvoli menším blobom ktoré by vznikaju pri zakrivení ruky, avšak nie sú to medzery medzi prstami. Po tejto operácii by už mal počet BLOBov odpovedať počtu prstov. Počet blobov získame následne pomocou metódy *size()*

```
int numOfFingers = numberOfFingersBlobs.size();
```

4 VYHODNOTENIE

V tejto kapitole zhrnieme hlavné vlastnosti detektoru, plusy a mínusy daného spôsobu detekcie a štatisticky vyhodnotíme výsledky detekcie giest podľa rôznych kritérii. Definované gestá sú na obr. 4.1.

(a) Metal

(b) OK

(c) Peace

(d) Ruka

(e) ThumbUP

(f) Stop

Obr. 4.1: Definované gestá

4.1 Test úspešnosti detektoru pre jednotlivé gestá

Hlavným kritériom tohoho testu bol časový interval, za ktorý detektor pozitívne identifikuje zobrazované gesto z 20 pokusov. Definované časové intervaly sú 1, 3, 5, 7 a 10 sekúnd. Dalšie možnosti sú „falošne identifikované“ a všetky väčšie intervaly považujeme za neidentifikované. Výsledky sme vyniesli do tab. 4.1

Obrázok č.	4.1d	4.1b	4.1c	4.1a	4.1e	4.1f
Čas [s]\Gesto	Ruka	OK	Peace	Metal	ThumbUP	Stop
1	12	5	15	14	4	16
3	4	5	4	3	5	3
5	4	2	1	1	2	0
7	0	1	0	1	2	0
10	0	2	0	1	3	0
Falošne detekované	0	3	0	0	4	1
Nedetekované	0	2	0	0	0	0
Úspešnosť [%]	100	75	100	100	80	95
Priemerný čas [s]	2,2	2,85	1,6	2,25	3,65	1,25

Tab. 4.1: Výsledky detekcie jednotlivých giest.

Z tab.4.1 vyplýva veľmi dobrá presnosť a spoľahlivosť detektoru. Celková úšpešnosť detekcie dosiahla hodnotu 91,6 % a priemerný čas detekcie bol 2,03 s.

4.2 Test úspešnosti detektoru počítania prstov

Hlavným kritériom tohto testu bolo počet správne detekovaných vystretých prstov na ruke z 10 pokusov. Výsledky sme zahrnuli do tab. 4.2.

Počet Prstov	Správne detekované	Nesprávne detekované	Úspešnosť [%]
1	6	4	60
2	6	4	60
3	5	4	50
4	6	4	60
5	8	2	80

Tab. 4.2: Výsledky detekcie počtu prstov.

Zo zhodnotenia vyplýva podstatne nižšia spoľahlivosť oproti detektoru giest. Celková úspešnosť dosiahla 62 %.

4.3 Zhodnotenie schopností a obmedzení

Najväčším obmedzením oboch detektorov bola webkamera s nízkym rozlíšením ktorej automatické mechanizmy aj pri menších zmenách jasu, napr. vložením nasvieteného objektu, výrazne ovplyvňovali hlavne mechanizmus odčítania pozadia.

Detektor giest ľudskej ruky

Vzhľadom na metódu detekcie ktorá je použitá v detektore vyplývajú isté obmedzenia, ktorých nedodržanie môže značne znížiť úspešnosť a presnosť detekcie. Pri testovaní boli najviac citeľné tieto stavy:

1. Zmena svetelných podmienok v snímanej scéne napr. odostrením závesov v miestnosti a pod..
2. Členité pozadie v mieste detekcie spôsobí, že v sa daná textúra pozadia pretlačí na vložený objekt, prahovaním sa dostane až na čierno-biely obraz ktorý tým znehodnotí.
3. Výrazne jasovo sa odlišujúce plochy v mieste detekcie spôsobia rovnaký problém ako bod 2.
4. Tiene ktoré občas vytvárala sama ruka v dôsledku silného bočného nasvietenia môžu čierno-biely obraz znehodnotiť.

Tieto problémy najviac ovplyvňovali vhodné vytvorenie finálneho obrazu. Na správnu detekciu treba zreprodukovať gesto tak, ako bolo zadefinované (nie je možné ruku otáčať a pod.). Keďže každý človek ukazuje gestá iným spôsobom, je najvhodnejšie aby si každý užívateľ definoval gestá sám.

Detektor počítania prstov

Vzhľadom na spôsob detekcie je nutné pri tejto metóde dodržať body z predchádzajúcej kapitoly a tieto pravidlá:

1. úplne vystrieť prsty,
2. čo najviac rozťiahnuť prsty od seba.

V dôsledku toho že prsty sú rôzne zakryvené táto metóda nemá až takú veľkosť a chvíľu trvá než sa človek naučí daný počet prstov detekovatelne znázorniť.

5 ZÁVER

Zadaním práce bolo naštudovať súčasné algoritmy pre sledovanie ľudskej ruky a giest, ktoré môžu byť rukou znázorňované. Následne bol vybraný vhodný sposob na detekciu a sledovanie, ktorý bol implementovaný v programovacom jazyku JAVA.

V prvej časti bola teoreticky objasnená problematika predspracovania, spracovania, úpravy a analýzy digitálneho obrazu. Boli popísané jednotlivé metódy existujúcich detektorov založené na detekcii farby, hrán, známeho objektu prípadne POI a uviedli ich príklady.

V druhej časti boli vytvorený prehľad o knižniciach v jazyku Java vhodných na snímanie, úpravu, spracovanie a porovnanie obrazu kde bol následne aj zdôvodnený ich výber.

Tretia časť sa zaoberá praktickou implementáciou zadania do programovacieho jazyku JAVA. Bol vytvorený program umožnujúci detekciu giest ľudskej ruky pozostávajúci zo 4 tried:

1. GUI (Graphical User Interface),
2. ImgCapt (snímanie obrazu),
3. ImgPrep (predspracovanie obrazu),
4. ImgDet (detekcia obrazu)

ktorými ho logický členený podľa funkcií. Obraz je najprv zosnímaný a dekódovaný triedou *ImgCapt*. Následne je odoslaný do triedy *ImgPrep* kde sa z neho odráta pozadie a je prevedený do stupňov šedi. Takýto obraz je prahovaný, automaticky volenou hodnotou podľa jasových pomerov v obraze, na čierno-biely. Takýto obraz je predaný do triedy *ImgDet* kde je pomocou naštudovaných algoritmov analyzovaný.

Vo štvrtej časti je štatistické zhrnutie testov uskutočnených na detektore. Ako hlavné kritérium bol braný čas za ktorý je detektor schopný gesto spoľahlivo detekovať prípádne nedetekovať. Pre každé gesto bol uskutočnený test 20 krát. Výsledná úspešnosť predstavovala 91,6 % a priemerný čas detekcie bol 2,03 s.

Následne bola otestovaná metóda počítania vystretých prstov ktorá však nedosiahla výsledky detektoru giest. Z 10 pokusov pre každý počet prstov bola celková úspešnosť 62 %.

Testami bolo overené že pri dodržaní správnych podmienok a presnému zobrazeniu gesta je možné daným detektorom dosiahnuť výsokú úspešnosť detekcie.

LITERATÚRA

[1] A, S.: javacv - Java interface to OpenCV and more - Google Project Hosting. 2013.
URL http://code.google.com/p/javacv/

[2] Abeles, P.: Hybrid Detect-Track and Detect-Describe-Associate Tracker. http://boofcv.org/techreports/report20120001v1.pdf, 2012.

[3] Bay, H.; Ess, A.; Tuytelaars, T.; aj.: Speeded-UpRobustFeatures(SURF). Technická zpráva, ETH Zurich and K.U.Leuven, 2008.

[4] Bay, H.; Tuytelaars, T.; Gool, L. V.: SURF: Speeded Up Robust Features. Technická zpráva, ETH Zurich and Katholieke Universiteit Leuven, 2006.

[5] Blázsovits, G.: DIP - Digital Image Processing, Interaktívna učebnica spracovania obrazu. http://dip.sccg.sk/, 2006.
URL http://dip.sccg.sk/predspra/predspra.htm

[6] Danilo Munoz, G. A. A., Fabio Andrijauskas: Marvin Image Processing Framework. 2013.
URL http://marvinproject.sourceforge.net/en/index.html

[7] Derpanis, K. G.: The Harris Corner Detector. Technická zpráva, York University, 2004.

[8] Firyn, B.: Webcam Capture in Java. 2012.
URL http://webcam-capture.sarxos.pl/

[9] Harris, C.; Stephens, M.: A Combined Corner and Edge Detector. Technická zpráva, Plessy Research Roke Manor, 1988.
URL http://www.ic.unicamp.br/~rocha/teaching/2012s1/mc949/aulas/additional-material-harris-detector.pdf

[10] of Health, N. I.: ImageJ. 2013.
URL http://rsbweb.nih.gov/ij/

[11] IASTED: *Visualization, Imaging, and Image Processing*, ročník 1. ACTA Press, 2003, ISBN 1482-7921, 351 s.

[12] Larson, K.: Home - LTI-CIVIL. 2007.
URL http://lti-civil.org/

[13] Larson, K.; Bloomer, W.; Rowley, A.; aj.: Home - FMJ. 2007, freedom for Media in Java.
URL http://fmj-sf.net/index.php

[14] Lee-kang, L.; Nguyen, T.; Chan, S.: Do we really need Gaussian filters for feature point detection? In *Signal Processing Conference (EUSIPCO), 2012 Proceedings of the 20th European*, aug. 2012, ISSN 2219-5491, s. 131 –135.

[15] LLC, C.: Xuggle. 2011.
URL http://www.xuggle.com/

[16] Microsystems, S.: 1999.
URL http://www.oracle.com/technetwork/java/javase/download-142937.html

[17] Pedersen, J. T.: SURF: Feature detection & description. Technická zpráva, AARHUS University, 2011.

[18] Rajmic, P.: *Základy počítačové sazby a grafiky.* Vysoké učení technické v Brně Fakulta elektrotechniky a komunikačních technologií Ústav telekomunikací, 2012, ISBN 978-80-214-4451-5.

ZOZNAM SYMBOLOV, VELIČÍN A SKRATIEK

API Application Programming Interface

BLOB Binary Large Object

FMJ Freedom to Media in Java

fps Frame Per Second

GPL General Public License

GUI Graphical User Interface

IR infrared

JMF Java Media Framework

LED Light Emmiting Diode

LTI-CIVIL Larson Technologies Inc. Capturing Images and Video In a Library

POI Points of Interest

px Pixel

ROI Region of Interest

SIFT Scale-Invariant Feature Transform

SURF Speeded-Up Robust Features

USB Universal Serial Bus