České vysoké učení technické v Praze

Fakulta elektrotechnická, Katedra kybernetiky

Bakalářská práce

Rozpoznávání gest člověka pro ovládání robotu

Autor práce: Michel Jabali

Vedoucí práce: Ing. Jan Chudoba

Květen 2019

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Jabali** Jméno: **Michel** Osobní číslo: **440934**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra kybernetiky**

Studijní program: **Kybernetika a robotika**

Studijní obor: **Robotika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Rozpoznávání gest člověka pro ovládání robotu

Název bakalářské práce anglicky:

Human Gesture Recognition for a Robot Control

Pokyny pro vypracování:

1. Proveďte rešerši vizuálních metod rozpoznávání člověka a částí jeho těla v obraze z kamery, které umožní detekovat gesta, jimiž by člověk mohl ovládat pohyb mobilního robotu. Zaměřte se na řešení využívající existující dostupné knihovny, vlastní vývoj metody detekce člověka není vzhledem ke komplexnosti problému předmětem práce.
2. Zvolte vhodnou metodu, jejíž implementaci bude možné otestovat při praktických experimentech.
3.Navrhněte sadu gest, kterou bude metoda schopna detekovat a proveďte experimentální měření pro vyhodnocení spolehlivosti detekce. Zmíněné ovládání robotu není předmětem zadání.

Seznam doporučené literatury:

[1] M.Šonka a Václav Hlaváč. Počítačové vidění. Praha: Grada, 1992.
[2] M. Kovalenko, S. Antoshchuk and J. Sieck, "Real-Time Hand Tracking and Gesture Recognition Using Semantic-Probabilistic Network," 2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation, Cambridge, 2014, pp. 269-274.
[3] Nguyen, Dang & Binh, Enokida & Shuichi, Toshiaki & , Ejima. Real-time hand tracking and gesture recognition system. In proceedings of International Conference on Graphics, Vision and Image Processing (GVIP-05). 2019.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Jan Chudoba, inteligentní a mobilní robotika CIIRC

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **22.11.2018** Termín odevzdání bakalářské práce: **27.05.2019**

Platnost zadání bakalářské práce: **20.09.2020**

Ing. Jan Chudoba	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Ing. Pavel Ripka, CSc.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

Datum převzetí zadání Podpis studenta

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, dne 24.5.2019

Poděkování

Rád bych poděkoval panu vedoucímu, ing. Janu Chudobovi, za jeho trpělivost a velmi cenné rady. Také bych rád poděkoval svým přátelům za jejich podporu.

Abstrakt

Tato bakalářská práce se zabývá analýzou vizuálních metod rozpoznávání částí člověka v obraze. Za použití knihovny OpenCV se podaří detekovat obličej, celé tělo a ruce metodou kaskád a samotnou ruku metodou detekce obrysů.

Návrh a implementace se zabývá aplikací umožňující rozpoznávání ruky pro ovládání robotu získáváním obrazu z webkamery. Pro ukázku byla vytvořena a natrénována vlastní kaskáda pro detekci dlaně. Přestože je k dispozici malé množství trénovacích dat, kaskáda detekuje dlaň překvapivě spolehlivě.

Klíčová slova

OpenCV, obličej, detekce, obraz, kaskáda, ruka, dlaň, prst, HSV, kontury.

Abstract

This thesis deals with analysis of visual methods of recognition human's parts in an image. Usining OpenCV library face, full-body and hands are detected by cascade method and hand alone is detected by the contor detection method.

The design and implementation deal with the application which enables the hand recognition for controlling a robot by retrieving the image from the webcam. An own cascade was created and trained to detect the palm. Although just a small amount of training data is available, the cascade detects the palm surprisingly reliably.

Keywords

OpenCV, Face, detection, picture, cascade, hand, palm, finger, HSV, contours.

Seznam obrázků

Obsah

Úvod a cíle práce

S nárůstem technologie roste také zájem o její ovládání mnohými způsoby pro zjednodušení práce - například pohybem části těla (ruky, nohy, hlavy) na zařízení, zvukovým signálem, či jinými způsoby.

Vzhledem k velké dostupnosti digitálních kamer s integrovaným mikrofonem se nabízí možnost práce/implementace mnohými směry.

V rámci výzkumu jsou volně dostupných mnoho open-source projektů, které se již hojně používají v různých aplikacích. Z důvodu oblíbenosti takových aplikací u uživatelů, ale i velkého vojenského využití, jsem se rozhodl podobnému tématu začít věnovat s nadějí vytvořit aplikaci pro ovládání robotů v místnosti.

Cílem této bakalářské práce je vytvořit takovou aplikaci, která umí detekovat části těla z obrazu kamery. Pomocí aplikace bude možné ovládat robot použitím dlaně a prsty ruky. Pro zjednodušení problému je aplikace navržená pro použití v černých rukavicích.

Kapitola 1

Analýza

Vzhledem k existenci velkého množství knihoven, které pracují s detekcí částí lidského těla s předem připravenými klasifikátory, bylo nutné si je podrobně prostudovat k výběru té nejvhodnější pro řešení dané úlohy.

1.1 PoseNet

Používá strojové učení, které umožňuje jen odhad lidské pozice v reálném čase. Pracuje se seznamem svých klíčových slov (keywords). Například *Pose*, která obsahuje seznam klíčových bodů a skóre spolehlivosti na úrovni instance pro každou zjištěnou osobu. Nebo například *Keypoint*, která reprezentuje část lidského těla s informací o vlastní pozici a skóre spolehlivosti klíčových bodů.

Vstupem je barevný obrázek, který je zpracován neuronovou sítí a následně dekódován pomocí dekódovacích algoritmů. Výstupem je pouze množina bodů v podobě lidské postavy, z tohoto důvodu není vhodné k použití pro detekci části těla.

1.2 OpenPose

OpenPose je knihovna pro detekci klíčových bodů na více lidech zároveň v reálném čase. OpenPose dokáže deteovat lidské tělo, obličejové body a také ruce v jednom obraze.

Vyhodou OpenPose je jeho široká detekce lidí a hlavně schopnost zpracovat velké množství bodů na více částech těla zároveň.

1.3 OpenCV

OpenCV je otevřená multiplatformní knihovna pro práci a manipulaci s obrazem v reálném čase. Výhodou této knihovny jsou její vnořené klasifikátory (.xml soubory) pro detekci předem určené části těla.
Pro mě osobně velkou výhodou knihovny OpenCV je její implementace v jazyce Java.

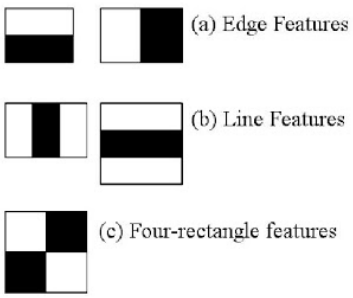
1.3.1 Detekce obličeje - Face Detection

Lze předpokládat, že ruce se pohybují v prostoru do určité maximální vzdálenosti od obličeje, respektive od hrudi. Tato informace pomáhá zúžit pole pro hledání a detekci ruky.

Klasifikátor pro detekci obličeje je nejčastěji použit z důvodu jasného a jednoznačného výsledku, neboť zde dochází k porovnání vybraných rysů obličeje ve zpracovávaném obrazu s obličejem/informací uložené uvnitř databáze. Je možné si vytvořit vlastní databázi, podle které je možné nejen rozpoznat obličeje, ale zároveň identifikovat lidi pomocí rozborů určitých vzorů založené na obličejové textuře a tvaru dané osoby.

Jedná se o strojové učení, kde kaskádová funkce je trénovaná z positivních (obsahují obličeje) a negativních (neobsahují obličeje) obrázků. Pojem kaskádová zde znamená, že funkce iterativně zvětšuje své poznatky.

Zpočátku funkce potřebuje mnoho obrázků, positivních i negativních, aby se učila a trénovala své klasifikátory. Poté je potřeba z databáze načtené rysy obličeje vytáhnout. Každý rys je hodnota získaná odečítáním součtu pixelů pod bílým obdélníkem ze součtu pixelů pod černým, viz. obrázek 1.1. [6]



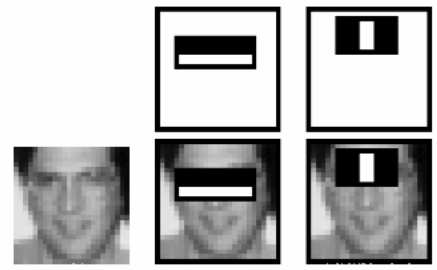
Obrázek 1.1: Zisk hodnot pixelu pro strojové učení. [6]

Pro každý výpočet rysu musíme najít celkový počet pixelů pod bílými a černými obdélníky. To znamená, že pro všechny možné velikosti a umístění každého pixelu musí být použito velké množství rysů. Proto se zde používá Integrální obraz, který je způsob digitální reprezentace obrazu. Hodnota každého pixelu se vypočítá jako součet všech pixelů ve směru doleva a nahoru. S tímto postupem pravý spodní bod obrazu obsahuje součet všech předchozích pixelů obrázku [2]. Tím pádem se veškeré výpočty eliminují na práci se 4 pixely.

Vezme-li v potaz lidský obličej, viz. obrázek 1.2, vidíme, že ze všech výpočtů je pro nás podstatný a zajímavý jen prostor kolem očí a nosu, neboť zde dochází ke kontrastům. V prostorách kolem tváře nedochází k podstatným změnám.

Na obrázku 1.2 je levém sloupci fotka ke zpracování. V prostředním se nachází analýza založená na faktu, že oči budou vždy tmavší něž okolí nosu a tváře. Pravý sloupec spoléhá na vlastnost, že nosní most je světlejší než oči.

Postupem je, že se aplikuje každý rys na všechny tréninkové obrazy pro nález nejlepšího prahu, který klasifikuje tváře na pozitivní a negativní. Cílem je výběr rysu s minimální chybovostí, což znamená, že se jedná o rys, který nejlépe klasifikuje snímky na obličeje a ne-obličeje.



Obrázek 1.2: Kontrasty na lidské tváři.[6]

Na začátku je všem snímkům přiřazena stejná váha a po každé klasifikaci se zvětšuje hmotnost nesprávně klasifikovaných obrazů. Opět se provede stejný proces a tím se vypočtou nové míry chyb a také nové hmotnosti. Proces se opakuje dokud není získána požadovanou přesnost či míru chyb.

Vážený součet těchto malých klasifikátorů se nazývá závěrečný klasifikátor (Někde v literatuře se vyskytuje pojem "slabý"místo malý, protože každý z těchto malých neboli slabých klasifikátorů nemůže rozhodnout o celé klasifikace obrazu).

Stále to není optimální řešení, protože i pro malé obrázky velikosti 32x32 (ikony) napočteme až více než 8000 rysů.

1.3.2 Klasifikátory

Vzhledem k skutečnosti, že většina obrazových oblastí jsou ne-obličejové, pak by bylo efektivněji se pokusit najít způsob kontroly, zda oblast je obličejová či není. V případě že není, pak je okamžitě zahozena a tím nemusí být dále zpracovávána a dojde k ušetření množství testovaných rysů a času.

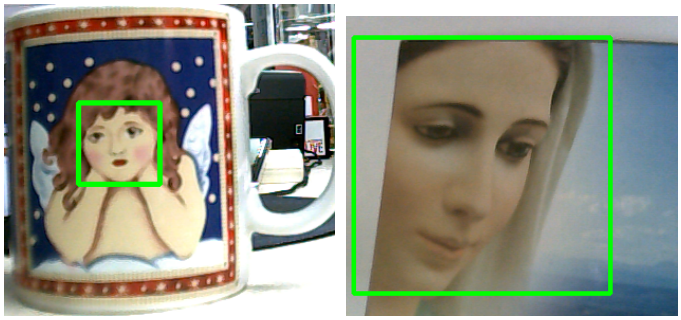
Za tímto účelem byly zavedeny klasifikátory. Namísto použití všech 8000 a více rysů na ikonu velikosti 32x32, je nutné si seskupit rysy do různých fází a aplikovat jednu fázi za druhou. Pro počáteční kontrolu obrazu se běžně v prvních fázích kontroly používá malé množství rysů pro případ, že se v obraze nějaký obličej vůbec vyskytuje. V případě neúspěchu v první fázi, pak není třeba pokračovat, protože obraz nejeví známky obličejových výskytů. Pokud uspějeme v první fázi, pustíme druhou fázi a tak dále. Obraz, který uspěje ve všech fázích považujeme za obličejovou oblast.

V úvodu práce jsem testoval a rozebral veškeré klasifikátory poskytnuté OpenCV knihovnou. Co se obličejových týče, nejlépe se mi pracovalo s klasifikátorem *haarcascade_frontalface_alt.xml*, neboť po jeho aplikaci byla detekce obličeje okamžitá a jednoznačná.

Z obrázků 1.3 je patrná aplikace face-klasifikátoru na keramickém (hrnek) a kartonovém povrchu (pohlednice).

Z detekce obličeje na pohledu (obrázek 1.3.b) je patrné, že klasifikátor dokresluje obličej, protože zasahuje i mimo obraz.

Díky strojovému učení klasifikátory pracují s obrysy-contours, což jsou jen křivky spojující všechny spojité body podél určitě hranice na základě jejich společné intenzity. [5].



Obrázek 1.3: Detekce obličejů na různých předmětech.

Mezi známá kritéria pro detekci obličeje patří:

1. Vzdálenost od objektivu - není možné detekovat obličeje, které jsou umístěny dál než 5 metrů.
2. Různá naklonění hlavy - také není možné detekovat obličeje, které nemíří přímo na fotoaparát.
3. Osvětlení - nelze detekovat obličeje v příliš temném prostředí.

1.3.3 Trénování klasifikátorů

Trénování klasifikátorů spočívá ve shromažďování tréninkových dat, přípravě tréninkových dat a provádění vlastního modelového tréninku [4].

Pro trénink zesílené kaskády slabých klasifikátorů je zapotřebí sady pozitivních vzorků (obrázky obsahující skutečné objekty, které chceme detekovat) a sady těch negativních, které obsahující všechno, co nechceme detekovat. Jejich tvorba se velmi liší. Zatímco negativní vzorky se připravují ručně z libovolných obrázků, pro přesnou definici, co se nemá detekovat. Na tvorbu positivních vzorku jsem použil *opencv_createsamples*. Výstupem aplikace je soubor binárního formátu s příponou *.vec*, který obsahuje vektory positivních obrázků.

Na obrázku 1.4. jsou positivní data se potřebnými detaily pro trénování klasifikátoru.

```
<Image path> <number of objects> <x> <y> <width> <height>

C:\Users\Michel\Desktop\pos-0.pgm.pgm 1 0 0 100 40
C:\Users\Michel\Desktop\pos-1.pgm.pgm 1 0 0 100 40
C:\Users\Michel\Desktop\pos-2.pgm.pgm 1 0 0 100 40
C:\Users\Michel\Desktop\pos-3.pgm.pgm 1 0 0 100 40
C:\Users\Michel\Desktop\pos-4.pgm.pgm 1 0 0 100 40
C:\Users\Michel\Desktop\pos-5.pgm.pgm 1 0 0 100 40
C:\Users\Michel\Desktop\pos-6.pgm.pgm 1 0 0 100 40
```

Obrázek 1.4: Příprava dat pro trénování klasifikátoru.

Pozn: $< numberofobjects >$ zastupuje počet objektů, které jsou v obrazu, a pro zjednodušení se u všech fotek vyskytuje jen jeden objekt. $< x >$ a $< y >$ jsou lokace objektů v obrazu a $< width >$ a $< height >$ jsou šířka a výška obrazu.

Je nutné dodržet, aby množství záporných dat bylo alespoň stejné jako počet těch pozitivních. V reálném použití je mnohem větší počet objektů, které má klasifikátor vyhodnotit záporně, než kladně.

Obrázek 1.5: Detekce objektu pomoci trénovaného klasifikátoru.

1.4 Klasifikátor dlaně

Nejprve je nutné nasbírat mnoho obrazů dlaně představující positivní data a ve stejném množství negativní data, obsahující všechno kromě dlaní.

```
<location> <objects' number> <x> <y> <w> <h>
                                                <location>

pos/28_45_Pro1.jpg 1 0 0 138 99
pos/28_45_Pro2.jpg 1 0 0 138 99                 neg/neg-0.pgm
pos/28_50_Pro1.jpg 1 0 0 138 99                 neg/neg-1.pgm
pos/28_50_Pro2.jpg 1 0 0 138 99                 neg/neg-2.pgm
pos/28_51_Pro1.jpg 1 0 0 138 99                 neg/neg-3.pgm
pos/28_51_Pro2.jpg 1 0 0 138 99                 neg/neg-4.pgm
```

Obrázek 1.6: Positivní (ruce.info) a negativní (bg.txt) data pro učení klasifikátoru.

Z obrázku 1.6. je vidět obsah dat. Zatím co u negativních dat je zapotřebí znát jen jejich umístění, kdežto u positivních dat je, krom umístění, nutné znát počet objektů vyskytujících se v obraze, souřadnice x,y pro jejich umístění zkoumaného objektu a také velikost jednotlivých obrazů.

Dále se vytvoří soubor s příponou *.vec* s použitím OpenCV pro pozdější zpracování. Soubor *.vec* je vektorová mapa pro zobrazení tvaru objektů. V tomto případě prsty a dlaně, příkazem *opencv_createsamples*.

V poslední řadě je zapotřebí klasifikátor natrénovat a to použití příkazu *opencv_traincascade*:

```
C:\Users\Michel\Dropbox\ff>opencv_traincascade -data data -vec ruce.vec -bg bg.txt -numPos 108 -numNeg 108 -numStages 2
-w 48 -h 34 -featureType LBP
```

Obrázek 1.7: Příkaz pro trénování klasifikátoru.

Nově vzniklé soubory budou uloženy v adresáři data použitím vektoru *ruce.vec*. -bg reprezentuje pozdí neboli negativní data pro trénování kaskády, následují $-numPos$ a $-numNeg$ což jsou počet pozitivních a negativních obrazů, $-numStages$ je počet etap (čím větší číslo, tím trénování trvá déle, ale také tím přesnější výsledky), $-w$ a $-h$ výška a šířka, a poslední

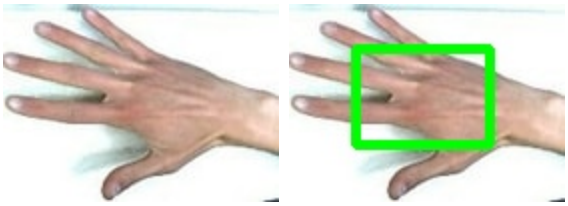
```
===== TRAINING 0-stage =====
<BEGIN
POS count : consumed   108 : 108
NEG count : acceptanceRatio    108 : 1
Precalculation time: 0.125
+----+---------+---------+
|  N |    HR   |    FA   |
+----+---------+---------+
|   1|        1|        1|
+----+---------+---------+
|   2|        1|0.0740741|
+----+---------+---------+
END>
Training until now has taken 0 days 0 hours 0 minutes 1 seconds.

===== TRAINING 1-stage =====
<BEGIN
POS count : consumed   108 : 108
NEG count : acceptanceRatio    108 : 0.484305
Precalculation time: 0.126
+----+---------+---------+
|  N |    HR   |    FA   |
+----+---------+---------+
|   1|        1|        1|
+----+---------+---------+
|   2|        1|0.0740741|
+----+---------+---------+
END>
Training until now has taken 0 days 0 hours 0 minutes 2 seconds.
```

Obrázek 1.8: Průběh vzniku klasifikátoru *cascade.xml* ve dvou etapách.

$-featureType$ přestavuje typ trénovací funkci, která v našem případě je LBP (Local Binary Pattern; lokální binární vzor)

Zkoušel jsem také trénovací funkci HOG (histogram of oriented gradients - histogram orientovaných gradientů) ale nepodařilo se mi ho zprovoznit. Případ Haar trénovací funkci by netrvala minuty ani hodiny jako LBP nebo HOG, proto jsem to ani nezkoušel.



Obrázek 1.9: Detekce dlaně pomoci *cascade.xml*.

Z obrázku 1.9. je zřejmé, že klasifikátor nedetekuje celou dlaň. Faktorů je tu více:

1. Počet původních positivních a negativních obrázků není dostatečný.
2. Klasifikátor očekává, že bude detekovat objekty uvnitř obrazu a ne jeden objekt přes celý obraz.
3. Byly provedeny jen 2 etapy trénování, viz obrázek 1.8.

1.5 Hodnocení klasifikátoru

Výsledek byl očekávaný vzhledem k faktu, že klasifikátor umí jen najít dlaň i prsty jako jednotný objekt a ten označí. Přesně splňuje podmínky trénování.

Také jsem se pokusil najít jednotlivé prsty, pomocí který budu moci ovládat robot, ale naneštěstí touto metodou se mi to nepodařilo a tím jsem musel najít úplně jiný způsob, viz. page 9

Kapitola 2

Návrh

V této části mě inspirovalo řešení jednoho z vývojářů, který také řeší zpracování obrazu a rozpoznávání gest. [1]

Úkolem je, aby aplikace našla všechny prsty jedné ruky a označila je jednotlivě od palce až po malíčku.

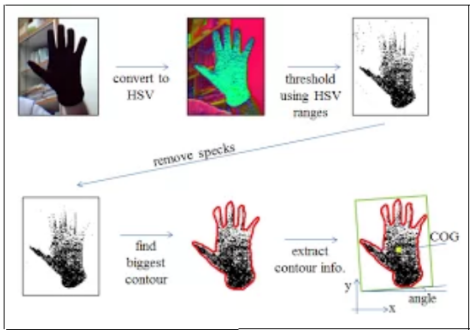
2.1 Aplikace

Na začátek musíme připojit web-kameru k softwaru a přidat OpenCV knihovny do konfigurací.

Je nutné použít textový soubor HSV (Hue - odstín, Saturation - sytost barvy, Value - hodnota jasu) pro určení rozsahu černé barvy kvůli eliminaci všech ostatních barev v pozadí a pro detekci jen ruky v černých rukavicích (podle mně je to nejjednodušší a velmi zřejmá barva, která zřídka mění svůj odstín na rozdíl od ostatních barev).

Pro výběr vhodných hodnot pro černou barvu, pak odstín a sytost nastavujeme na celém rozsahu, tj. od 0-180 pro odstín, respektive 0-240 pro sytost a operujeme jen s hodnotou jasu, což pro černou barvu je 0-40.

Tyto hodnoty nám pomůžou se dostat k centru dlaně (COG - Center Of Gravity), viz. obrázek 2.1.

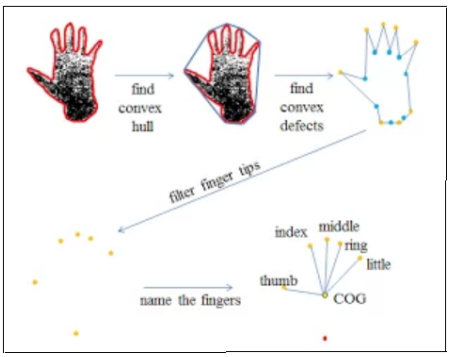


Obrázek 2.1: Cesta konverze z HSV do COG ke hledání kontur. [1]

Nejprve dojde k převedení fotky do HSV prostředí, poté je pomocí hodnot HSV nalezen

práh. Ten je následně zpracován smítkem pro jasnější vidění barev (To je důvod, proč není zapotřebí ideálně kontrastní barvy, která zakryje pozadí, pro čistší detekci černých rukavic). Po smítku dojde ke hledání a nalezení kontur, z jejíchž informací najdeme COG.

V získaném COG, jsou hledány jednotlivé prsty na natažené ruce. Pomocí konvexního obalu ruky jsou nalezeny nejspodnější a nejhornější body, které jsou vrcholy prstů a spodek dlaně. Na závěr jsou jednotlivé vrcholy pojmenovány. Pojmenování prstů zavedené v metodách kapitole Implementace se upřednostňuje pro levou ruku, podrobněji v chapter 3.



Obrázek 2.2: Nalezení konečků prstů uvnitř konvexního obalu ruky. [1]

Nejprve se je nalezen palec a ukazováček vzhledem k jejich polohách vůči středu ruky. Ostatní prsty jsou odvozeny zpětně vzhledem k jejich pozici vůči ukazováčku. Aplikace je navržena tak, že levou ruku projde z pravé strany z pohledu kamery a tím nalezne malíček, který přeskočí. Prsteník a prostředníček jsou také přeskočeny až dojde na známý prst, což je zde ukazováček. Od něho se zpětně pojmenují jednotlivé prsty.

Na závěr jsou nakresleny žluté čáry mezi středem dlaně a jednotlivými identifikovanými prsty, popřípadě neznámé prsty jsou označeny červenými kroužky.

2.2 Gesta

Za základní robotické pohyby jsou považovány pohyb vpřed, stop a otočení doprava či do leva. Z tohoto důvodu je aplikace navržená tak, že v případě rozpoznání palce a ukazovačku (dvou hlavních prstů) robot koná pohyb vpřed. V případě identifikace všech pěti prstů se robot zastaví. Při rozpoznání tří prstů (palec, ukazováček a prostředníček) se robot začne otáčet doprava.

Implementace se zaměřuje na nalezení palce a ukazováčku. Z tohoto důvodu jsou vybrána gesta právě obsahující tyto dva prsty. Detekce jakýchkoli gest bez jejich použití není možná.

Kapitola 3

Implementace

Testovací aplikace je napsaná v jazyce Java. Pro zpracování obrazu se používá knihovna OpenCV, konkrétně Java wrapper nazvaný JavaCV. Aplikace se připojí k webové kameře počítače a jednotlivé snímky zpracuje. Na každém provede detekci ruky a poté jej zobrazí v okně. Výsledek vypadá jako živý přenos z webové kamery o frekvenci 20-30 snímků za sekundu dle možností web kamery.

Hlavní třída *Handy* řídí celý proces: načítání nativních OpenCV knihoven a zpracování pracovního okna. Úkolem další třídy *HandPanel* je zobrazovat zpracovaný obraz s průměrným časem zpracování snímku, popřípadě na výstup oznámit, že se nelze připojit ke kameře. Poslední třída *HandDetector* má za úkol zachycený obraz zpracovat a detekovat objekty. V aplikaci se dále vyskytuje enum, ve kterém jsou uloženy názvy všech jednotlivých prstů a také proměnná UNKNOWN.

3.1 Obraz

3.1.1 Převod do HSV

Hned začátkem procesu je zachycený obraz konvertovaný do HSV formátu, neboť v tomto barevném prostoru jsou barevné odchylky výraznější, což usnadňuje filtraci obrazu.

setHSVRanges() je metoda používaná pro zisk tří prahových hodnot HSV, respektive šest, protože každý z parametrů má nižší a vyšší mez. *fnm*-filename je parametr metody s cestou, odkazující na textový soubor s hodnotami HSV. Celá metoda je zabalená v try-catch bloku, pro případ nezachycení definované barvy.

V metodě *update*() se volá metoda *cvResize*() pro změnu velikosti původního obrazu web kamery na žádoucí procento, přesněji z *image* na *scaleImg*. Zde se také volá metoda *cvCvtColor*() z OpenCV balíčku, jejíž úkolem je převést *scaleImg* do *hsvImg*, čili z jednoho barevného prostoru na jiný s konstantou *CV_BGR2HSV*, která je rovná 40.

3.1.2 Práh obrazu

Nyní po převodu do HSV formátu, potřebujeme najít práh obrazu neboli image threshold vyfiltrováním barev mimo meze z již načteného HSV souboru. Opět přes metody OpenCV, tentokrát z *opencv_core* metodu *cvInRangeS*(), přesněji převést *hsvImg* na *imgThreshed* s parametry nižších a vyšších mezí pro HSV. Ve výsledném obrázku zůstanou pouze pixely barev v povolených mezích HSV.

Na závěr, před hledáním kontur, jsou odstraněny všechny "skvrny"pro čistší a jasnější detekci největší kontury. Je nutné zachovat velikost původního *imgThreshed*. Pro tento účel se používá metoda *cvMorphologyEx*() také z OpenCV balíčku.

3.1.3 Nalezení obrysu/kontury

Poté, co je obraz vyčištěný od skvrn pro lepší detekci ruky, je na řadě nalezení velké kontury v něm, jinými slovy, oblast s velkou hustotou pixelů.

Vstupním parametrem metody *findBiggestContour* je obraz *imgThreshed* tj. prahový obraz po čištění, aby zbytečně nedošlo k detekci nechtěných bodů v pozadí.

Postup metody *findBiggestContour* je jednoduchý. Nejdříve se alokuje místo v paměti pro kontury, které se budou generovat a pak jsou všechny obrysy nalezeny metodou *cvFindContours*(). Ze všech nalezených kontur jsou vybrány pouze ty, jejichž opisující obdélník má největší plochu. Je předem definovaná minimální plocha, kterou kontura musí převýšit, aby se brala v potaz. Předpokladem je, že výsledná největší kontura reprezentuje ruku.

Nebude-li výstup metody žádný tj. *null*, pak se předpokládá, že na snímku ruka není, zpracování snímku končí a pokračuje se s dalším.

3.2 Obrysy

Jako poslední část obrázku 3.2.a je nalezením COG a hlavní osu.

3.2.1 COG

Metoda *extractContourInfo*() spočítá a najde umístění COG a úhel hlavní osy kontur vzhledem k vodorovné přímce. I když na první pohled se zdá být uhel pravý, záleží jen na pozici a naklonění ruky.

Pro výpočet COG je použita funkce *cvGetSpatialMoment*() z OpenCV balíčku, která načítá prostorové momenty potřebné pro výpočet centra v $[x,y]$ rovině, vzorec 3.1.

$$m(p,q) = \sum_{i=1}^{n} I(x,y)x^p y^q \tag{3.1}$$

p,q jsou argumenty funkce momentu (vzorec 3.1) a zde představují mocniny pro x,y a musí platit $p+q \leq 3$. Použitá funkce $I(x,y)$ reprezentuje intenzitu jednotlivých pixelů (x,y). Parametr n je počet pixelů, které tvoří tvar obrazu.

3.2.2 Hlavní osa kontur

Pro nalezení hlavní osy obrysů je použita funkce *cvGetCentralMoment*(), jejímž úkolem je najít centrální momenty, které později v kódu budou použity v metodě *calculateTilt*(), která vrací úhel hlavní osy obrysů s vodorovnou osou s předpokladem, že svislá y-ova osa míří dolu, viz. obrázek 3.1.

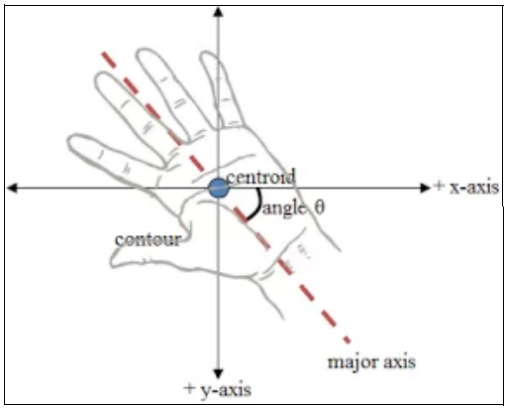
Centrální moment je definován:

$$m(p,q) = \sum_{i=1}^{n} I(x,y)(x-x_c)^p(y-y_c)^q \tag{3.2}$$

kde

$$x_c = \frac{m_{(1,0)}}{m_{(0,0)}} \tag{3.3}$$

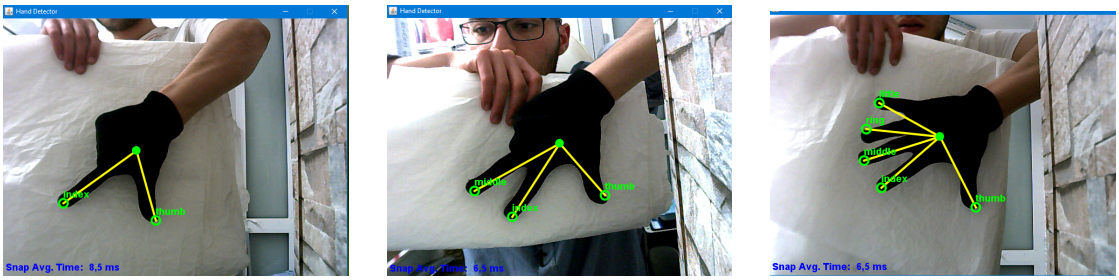
$$y_c = \frac{m_{(0,1)}}{m_{(0,0)}} \tag{3.4}$$



Obrázek 3.1: Nalezení hlavní osy kontur vůči vodorovné ose. [1]

$$\tan(2\theta) = \frac{2.m_{(1,1)}}{m_{(2,0)} - m_{(0,2)}} \tag{3.5}$$

kde θ je uhel hlavní osy kontur s x-ovou přímkou. Momenty $m_{(1,1)}$, $m_{(2,0)}$ a $m_{(0,2)}$ jsou jednotlivé centrální momenty.

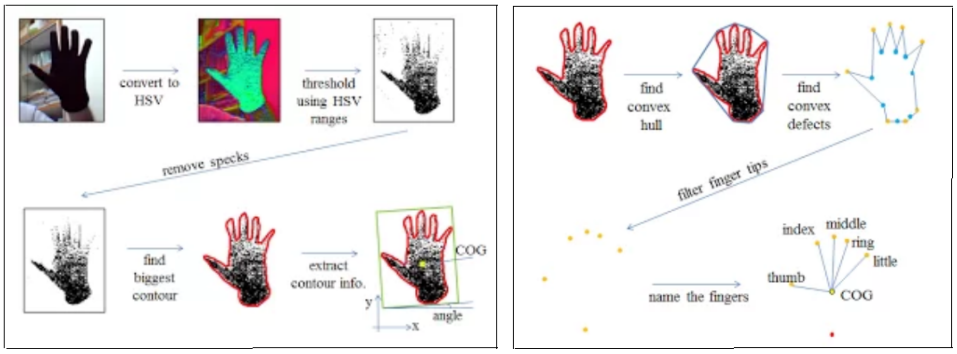


Obrázek 3.2: Záběry ze vzdálenosti 50cm.

Obrázky 3.2. potvrzují, že kartézský systém souřadnic ”míří” dolů a nikoli nahoru jak je běžně.

3.3 Analýza Prstů

Nyní, po nalezení COG a určením hlavní osy kontur, viz. obrázek 3.3.a, se přejde k obrázku 3.3.b. Nejdříve se ruka obalí konvexním obalem, z něhož jsou pak odvozeny vrcholy prstů. Na závěr jsou jednotlivé prsty pojmenovány.

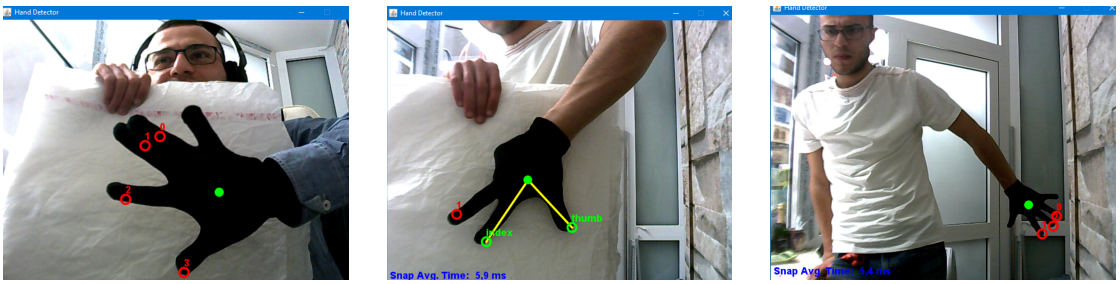


Obrázek 3.3: Návaznost jednotlivých procesů z návrhu.[1]

3.3.1 Naleznutí vrcholů

Metoda *findFingerTips*() začíná s již získanými obrysy, kde spočte jejich konvexní obal a najde jeho defekty neboli vady. Parametr *bigContour* je již vypočtená dlaň bez skvrn v pozadí.

Defektem je zde označováno všechno, co se dá identifikovat za prst. Jsou to případy neznámých prstů zmíněné už v enumu za UNKNOWN a které se budou zobrazovat jako červená kolečka na obrazovce, viz. obrázky 3.2.



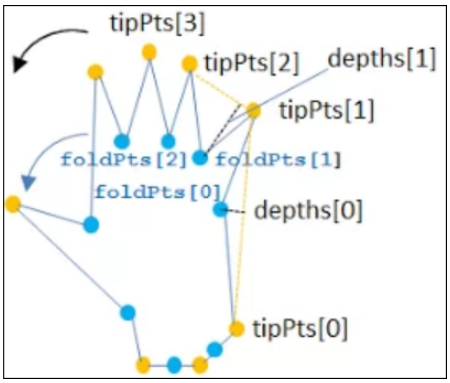
Obrázek 3.4: Případy identifikace dočasně neznámých prstů.

Vzhledem k velkému počtu vrcholů na identifikované kontuře, je použitá funkce *cvApproxPoly*() pro aproximaci křivky dlaně jinou křivkou s méně vrcholy. Výsledkem je, že rozdíl mezi porovnávanými křivkami je menší nebo roven zadané přesnosti.

$$cvApproxPoly(bigContour, Loader.sizeof(CvContour.class), \\ approxStorage, CV\_POLY\_APPROX\_DP, 3, 1)$$

Přesnost je rovná 3 (předposlední parametr metody), tj. maximální vzdálenost mezi skutečnou křivkou dlaně a její aproximací. Parametr *CV_POLY_APPROX_DP* reprezentuje aproximační algoritmus kontur, nehledě na faktu, že je to konstanta rovné nule.

$cvConvexHull2(approxContour, hullStorage, CV\_COUNTER\_CLOCKWISE, 0)$
je metoda pro nalezení konvexního obalu kolem kontur, kde jedním z parametrů musí být výsledek již aproximované funkce. Další z parametrů je *CV_COUNTER_CLOCKWISE*, což je směr orientace právě proti směru hodinových ručiček, dále viz. obrázek 3.5.
Parametr *hullStorage* je posloupnost obsahující konvexní body.



Obrázek 3.5: Pořadí detekce bodů na dlani. [1]

Momentální stav je, že jsou vypočteny konvexní obal a aproximovaná křivka ruky. Funkcí *cvConvexityDefects* se zjistí jejich rozdíl, tj. body, které leží na aproximované křivce ruky, ale ne na křivce konvexního obalu. Případ, že počet těchto bodů přesáhne nějaké maximum (zde 20, stačilo by i 15, ale je to rizikové), značí, že se pravděpodobně nejedná o ruku nebo je ruka nesprávně nakloněná. V ten okamžik není nic detekováno ani nakresleno na výstupním obraze.

Pro zpracování jednotlivých vrcholů jsou zkopírovány informace o vadách ze sekvence defektů do pole obsahující souřadnice vrcholů prstů.
Pomocí metody $CvConvexityDefect.start()$ jsou nalezeny body kontury ($tipPts$), kde začíná závada. Pomocí metody $CvConvexityDefect.depth\_point()$ jsou nalezeny nejvzdálenější body ($foldPts$) od konvexního trupu, které jsou stále uvnitř defektu, viz. obrázek 3.5.
Body *depths*, které se objevují mezi vrcholy a začátky prstů jsou nalezeny pomocí metody $depth()$.

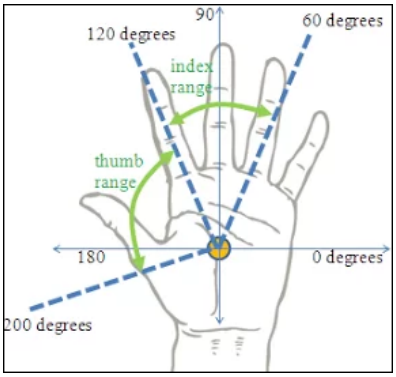
Na závěr je pro hledání hrotů prstů zavolaná metoda $reduceTips()$, jejíž parametry jsou počet nalezených bodů - $tipPts$, $foldPts$ a *depths*. Úkolem metody je eliminovat "falešné prsty", které mohou vzniknout nerovnou posloupností bodů, například na hranách dlaně. Tyto "falešné prsty"se vyznačují tím, že sedlo mezi nimi je příliš mělké, či naopak úhel (sedlo-prst-sedlo) je tupý, viz obrázek 3.6.

Obrázek 3.6: Případy falešných prstů. [1]

3.4 Identifikace prstů

Identifikace probíhá ve dvou krocích. Nejprve se naleznou prsty nacházející se v levé polorovině od COG, tj. palec a ukazováček, viz. obrázek 3.7. Tyto prsty jsou referenčními body pro identifikaci dalších prstů, proto se předpokládá jejich přítomnost.

V druhém kroku se pomocí referenčních prstů pojmenují ostatní zbylé prsty.



Obrázek 3.7: Nalezení prstů vůči středu dlaně. [1]

3.4.1 Nalezení palce a ukazováčku

Na začátku se musí nalézt palec a ukazováček. Pro každý je definovaný úhlový rozsah, tj. rozsah úhlu svíraný vodorovnou přímkou a špičkou prstu kolem bodu COG. Budou s největší pravděpodobností uloženy na konci seznamu, protože obal kontur je vytvořen v proti směru hodinových ručiček. Všechno se provede v metodě $labelThumbIndex()$ včetně volání metody $angleToCOG()$, kde dojde k aplikaci vzorečků 3.5, respektive 3.6.

$$\tan(2\theta) = \frac{2.m_{(1,1)}}{m_{(2,0)} - m_{(0,2)}} \tag{3.6}$$

3.4.2 Nalezení zbylých prstů

Po nalezení dvou hlavních prstů, pak projdeme ruku zleva doprava dokud nenarazíme na již známý prst a od tohoto momentu opět procházíme zpět a postupně pojmenováváme prsty.

Je zde použitá metoda $labelUnknowns()$, neboli ”pojmenování neznámých prstů” doposud. V této metodě voláme metody $labelPrev()$ a $labelFwd()$, které pohybuji vzad respektive vpřed

přes seznam prstů a označují neznámé prsty. V nich je použitá metoda *usedName*(), jejíž úkolem je kontrola seznamu prstů zda obsahuje název, který metoda potřebuje použít.

3.5 Nakreslit prsty

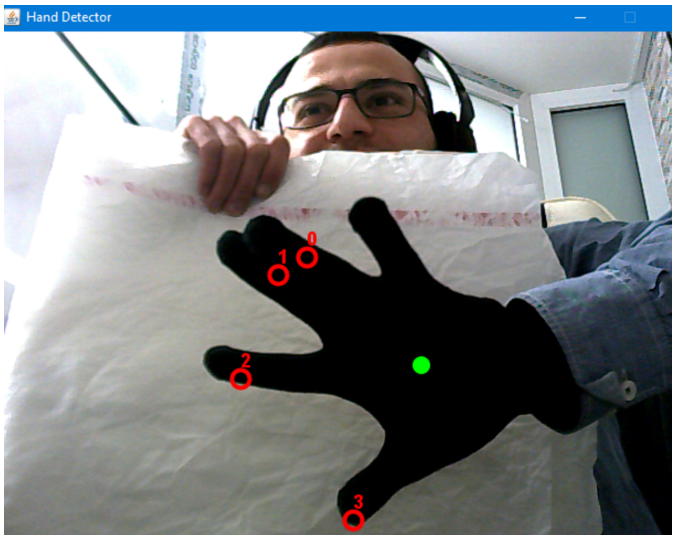
Na závěr zbývá jen namalovat čáry mířící ze středu dlaně k identifikovaným prstům. UNKONWN prsty jsou označeny červenými kroužky, ostatní jsou identifikované zelenými kroužky. Z COG k identifikovaným prstům vždy míří čára, prsty jsou značeny žlutými čarami od COG, viz. obrázek 3.8.

Obrázek 3.8: Správná identifikace všech prstů.

Hned na začátku metody je podmínka, zda seznam prstů něco obsahuje. V případě že seznam je prázdný, pak dojde k opuštění této metody a začne se zpracovávat další snímek.

Metoda *draw*() projde cyklem přes seznam prstů a ptá se, jestli daný prst je známý či není. Pokud je neznámy-*UNKNOWN*, pak se vytvoří červený kroužek s číslicí kolem vrcholu daného prstu voláním metody *setPaint*(*Color.RED*) ze třídy *Graphics2D*, viz. obrázek 3.9. Číslice reprezentují počet neznámých prstů.

V opačném případě, tj. daný prst je identifikovaný, pak je kolem vrcholu prstu zelený kroužek s jeho názvem a čárou ze středu dlaně k prstu, viz. obrázek 3.8.



Obrázek 3.9: Případy neznámých prstů.

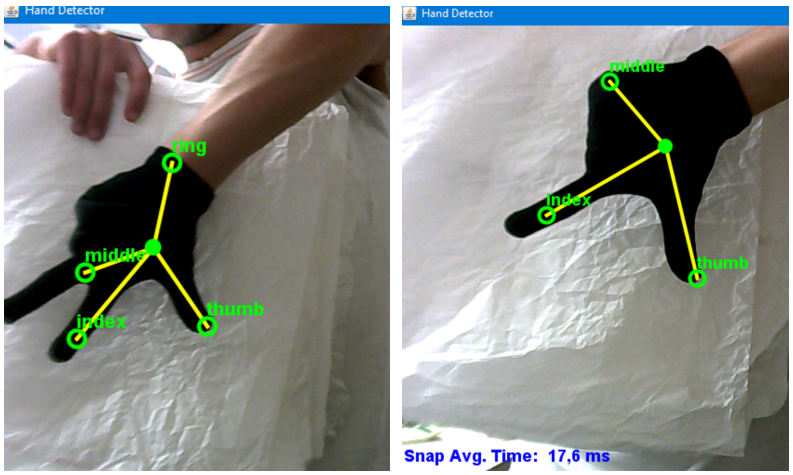
Kapitola 4

Testování

Test je pomocí CANYON Webkamera CNR s rozlišením 1,3MP 4:3 (640x480) a zorným úhlem 70 stupňů. Maximální frekvence snímání snímků je 30 fps (640x480).

Aplikace je velmi citlivá na osvětlení v místnosti, které je stěžejní pro správnou detekci černé barvy rukavic (při velkém osvětlení webkamera nedokázala zaznamenat jasnou černou barvu, viz. hodnoty HSV souboru a tím pádem nedošlo k žádné detekci). Dále je citlivá na naklonění a otevření dlaně.

Na obrázku 4.1 je vidět křehkost metody a její závislost na správné pozici dlaně.



Obrázek 4.1: Špatná identifikace některých prstů.

Špatná detekce prstů má dvojí vysvětlení:

1. Obrázek 4.1.a: Nesprávná identifikace středu dlaně (COG) má velký dopad na celou dlaň, protože zde už neplatí pravidlo o uhlech, jež bylo zmíněno v kapitole Návrh. Z tohoto důvodu došlo k identifikaci prsteníku na nemožném místě.

2. Obrázek 4.1.b: Software hledal nejvyšší blízký vrchol s očekáváním, že najde další prst. Hlavním problémem zde je pozice dlaně, protože při jejím posunutí došlo ke správné

identifikaci.

4.1 Statistika

Provedl jsem 4 měření abych zjistil, jak dobře dojde ke správné detekci v případě posunutí od kamery. Od určité vzdálenosti (zde 50*cm*) platí pravidlo: Čím vzdálenější je ruka od zdroje/kamery, tím hůř se detekuje. Před vzdálenosti 50*cm* je ruka příliš blízko k kameře, proto nedochází ke 100% detekci, jak jsem očekával.

	2 prsty	3 prsty	5 prstů
40*cm*	7/25	19/23	3/20
50*cm*	12/28	40/44	7/39
80*cm*	23/60	59/96	5/40
100*cm*	10/22	9/23	2/22

Tabulka 1: Přesnost detekce jednotlivých prstů.

Hodnoty v tabulce 1 definují počet úspěšně povedených pokusů vůči celkovému provedenému měření.

Kapitola 5

Závěr

Knihovna OpenCV byla vybrána pro práci, protože je nejrozšířenější a nejvíce podporovaná, včetně jazyka Java.

Na začátku práce byly otestovány OpenCV klasifikátory pro detekci částí těla, na kterých jsem pochopil jejich funkčnost.

Byl vytvořen vlastní klasifikátor dlaně pomocí metod z OpenCV, který detekuje dlaň spolehlivě i přes malé množství vstupních dat.

Pro detekci gest byla použitá jiná metoda, která v reálném čase nejen detekuje, ale i identifikuje jednotlivé prsty. Pomocí této metody se mi podařilo vytvořit pár základních gest pro ovládání robotu. Metoda je nespolehlivá, protože občas nedojde k správné detekci v závislosti na úhlu nakloněné ruky nebo správné identifikaci prstu.

Na práci je možné navázat. Kombinací obou zmíněných metod, detekce ruky v obraze a rozpoznávání gest, je potenciálně možné dosáhnout o mnoho lepších výsledků ve zpracování obrazu.

Detekce nerovných gest, například uzavřená pěst nebo gesto s jedním prstem, je určitě zajímavým tématem k pokračování v práci.

Literatura

[1] Attila Mihaly Balazs (2012). Hand and Finger Detection using JavaCV. https://www.javacodegeeks.com/2012/12/hand-and-finger-detection-using-javacv.html. Online; accessed 8 May 2019.

[2] BADGERATI (2019). Computer Vision – The Integral Image. https://computersciencesource.wordpress.com/2010/09/03/computer-vision-the-integral-image/. Online; accessed 6 May 2019.

[3] Flintbox (2016). OpenPose - Realtime Multiperson 2D Keypoint Detection from Video. https://flintbox.com/public/project/47343/. Online; accessed 24 May 2019.

[4] OpenCV (2017a). Cascade Classifier Training. https://docs.opencv.org/3.3.0/dc/d88/tutorial_traincascade.html. Online; accessed 29 April 2019.

[5] OpenCV (2017b). Contours : Getting Started. https://docs.opencv.org/3.3.1/d4/d73/tutorialpycontours$begin.html. Online; accessed 30 April 2019.

[6] OpenCV (2017c). Face Detection using Haar Cascades. https://docs.opencv.org/3.3.0/d7/d8b/tutorial_py_face_detection.html. Online; accessed 29 April 2019.

[7] OpenCV (2019). Structural Analysis and Shape Descriptors. https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html. Online; accessed 9 May 2019.

[8] TensorFlow (2018). Real-time Human Pose Estimation in the Browser with TensorFlow.js. https://medium.com/tensorflow/real-time-human-pose-estimation-in-the-browser-with-tensorflow-js-7dd0bc881cd5. Online; accessed 24 May 2019.