

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A
INFORMATIKY



AUTOMATICKÁ VIZUÁLNA KONTROLA
VSTUPU

BAKALÁRSKA PRÁCA

2020

Filip Eliaš

UNIVERZITA KOMENSKÉHO V
BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A
INFORMATIKY

AUTOMATICKÁ VIZUÁLNA KONTROLA VSTUPU

BAKALÁRSKA PRÁCA

Študijný program:	Aplikovaná informatika
Študijný odbor:	2511 Aplikovaná informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	doc. RNDr. Milan Ftáčnik, CSc.

Bratislava 2020

Filip Eliaš



Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Filip Eliaš
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Automatická vizuálna kontrola vstupu
The automatic visual control of the entry

Anotácia: Implementačná práca

Cieľ: Preskúmať možnosti automatickej kontroly vstupu do laboratória FTL v pavilóne informatiky a prakticky ich overiť

Vedúci: doc. RNDr. Milan Ftáčnik, CSc.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.
Dátum zadania: 10.10.2017

Dátum schválenia: 07.10.2019

doc. RNDr. Damas Gruska, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Podakovanie

Rád by som sa poďakoval svojmu školiteľovi doc. RNDr. Milanovi Ftáčnikovi, CSc. za ochotu, pomoc, cenné rady a usmerňovanie pri písaní práce.

Abstrakt

Cieľom práce je preskúmať možnosti automatickej vizuálnej kontroly vstupu do laboratória FTL v pavilóne informatiky a prakticky ich overiť. Vzhľadom na malý počet oprávnených osôb sme si zvolili príznakový prístup založený na príznakoch HOG. Kvôli celosvetovej pandémie Covid-19 sme nemali prístup do laboratória a tak sme si vytvorili databázu pozostávajúcu zo snímok reálnych ľudí ako aj umelo získaných obrazov. Riešili sme detekciu tváre vo videu z kamery snímajúcej vstup do miestnosti. Na klasifikáciu sme zvolili lineárny klasifikátor, konkrétne sme trénovali SVM. Testovali sme pomocou vzájomnej validácie a úspešnosť rozpoznávania bola vo všetkých testovacích scenároch na statických snímkach viac ako 94%. Pri testovaní na živom video zázname sme dosiahli výsledky 92,5% pre ideálne zábery a 66,6% pre ostatné.

Abstract

The goal of the thesis is to research options of automatic visual verification of entering in FTL laboratory in pavilion of informatics, and practically certify them. Base on the small number of authorized persons we choose feature method based on HOG features. Because of world wide pandemic of Covid-19 we did not have access to laboratory so we created database containing pictures of real persons and artificially obtained pictures. We solved face detection on live video gained from camera monitoring entrance to room. For classification we choose linear classifier, specifically we trained SVM. We tested using mutual validation and achieved success rate on static images was higher than 94%. Testing on live footage gave us success rate of 92,5% for ideal footage and 66,6% for rest.

Obsah

Úvod	1
1 Prehľad problematiky	2
1.0.1 Spôsoby rozpoznávania tvárí	2
1.0.2 Rozpoznávanie tváre pomocou príznakov	4
1.0.3 Problémy rozpoznávania tvárí	5
1.1 Detekcia tváre	5
1.2 Metódy získavania príznakov	7
1.2.1 Histogram of oriented gradients - HOG	7
1.2.2 SIFT	8
1.2.3 SURF	10
1.3 Klasifikátory	13
1.3.1 Lineárne klasifikátory	14
1.3.2 Support vector machine	15
2 Návrh riešenia	17
2.1 Snímanie obrazu	18
2.2 Detekcia pohybu a tváre	19
2.3 Geometrické zarovnanie tváre	21
2.4 Príznyaky	21
2.5 Rozpoznávanie tváre	22
3 Implementácia	23
3.1 __init__(self)	24
3.2 train_svm(self)	24
3.3 euclidean_distance(self, a, b)	25
3.4 find_eyes(self, img_path)	25
3.5 align_face(self, img_path)	25
3.6 hog_features(self, dat, check = False)	26
3.7 is_human(self, image)	26
3.8 recognize(self, image)	26

3.9	name(self,argument)	26
3.10	detect_face(self,image)	27
3.11	detect(self)	27
3.12	Databáza	28
4	Testy a výsledky	31
4.1	Test rozpoznávania všetkých snímok	31
4.2	Test rozpoznávania reálnych snímok	33
4.3	Test rozpoznávania naživo	34
4.4	Porovnávanie s face_recognition	36
	Záver	38
	Literatúra	40

Úvod

Cieľom práce je preskúmať možnosti automatickej vizuálnej kontroly vstupu do laboratória FTL v pavilóne informatiky a prakticky ich overiť. Hlavným dôvodom je zvýšenie bezpečnosti v laboratóriu.

Vzhľadom na nízky počet osôb oprávnených na vstup sme sa rozhodli použiť na rozpoznávanie príznakové metódy založené na HOG príznakoch a na klasifikáciu sme si zvolili lineárny klasifikátor. Vzhľadom na nepretržité snímanie vstupu do miestnosti kamerou sme riešili detekciu tváre na videu a problémy s tým spojené.

V prvej kapitole sa venujeme prehľadu problematiky kde si podrobnejšie predstavíme základné kroky rozpoznávania založeného na príznakoch. Najprv si uvedieme Viola-Jones algoritmus na detekciu tváre. Následne predstavíme viaceré možnosti na získanie príznakov tváre s využitím detektora a deskriptora. Priblížime si aj klasifikátory, a pozrieme sa špeciálne na lineárny klasifikátor Support Vector Machine.

V druhej kapitole navrhujeme naše riešenie problematiky, a podrobnejšie rozoberáme odlišnosti od generického postupu príznakového rozpoznávania opísaného v prvej kapitole. Riešime aj problematiku detekcie tváre na videu z kamery snímajúcej vstup do miestnosti a uvádzame postupy predspracovania obrazu v takomto prípade.

V tretej kapitole sa venujeme implementácii nášho riešenia v jazyku Python s využitím knižnice OpenCV. Vzhľadom na nemožnosť získať podklady na tréning a testovanie priamo z laboratória pre súčasné obmedzenia, uvádzame postup pri získaní náhradnej databázy obrazov.

V poslednej kapitole uvádzame výsledky testovania, ktoré prebehlo na našej databáze. Uvádzame rôzne prípady testov a ich výsledky. Vo všetkých testovacích scenároch na statických snímkach sme dosiahli úspešnosť viac ako 94%. Pri testovaní na živom video zázname sme dosiahli výsledky 92,5% pre ideálne zábery a 66,6% pre ostatné.

Kapitola 1

Prehľad problematiky

Podľa známeho paradoxu umelej inteligencie niektoré činnosti, ktoré sú ľuďom prirodzené, nemusia sa ich zložito učiť a dokážu ich vykonávať veľmi jednoducho, vyžadujú dlhý čas, kým dokážeme naučiť počítače vykonávať ich na podobnej úrovni, a platí to aj naopak. Človek už od detstva dokáže vnímať čo vidí, rozpoznávať rôzne farby, tvary, tieň a podobne. Počítač to nedokáže, až kým mu na to nenaprogramujeme vhodný softvér.

Problému počítačového videnia sa informatici venujú približne od 70. rokov 20. storočia. Dlhú dobu sa v tomto odvetví nedosahovali výsledky, ktoré by boli porovnateľné s výsledkami ľudského videnia, až v poslednom desaťročí priniesli prelom hlboké konvolučné neurónové siete. Zaujímavé však boli čiastkové výsledky, ktoré dokázali posúvať toto odvetvie vpred. To platí aj pre problém rozpoznávania tváre.

1.0.1 Spôsoby rozpoznávania tváří

Existujú rôzne metódy a spôsoby akými sa rieši problematika rozpoznávania tváří. Dve najznámejšie sú

- (a) Rozpoznávanie tváří pomocou neurónových sietí
- (b) Rozpoznávanie tváří pomocou príznakov (features)

Neurónové siete sú počítačové systémy, ktoré sú súčasťou takzvaného *hlbokého učenia*. Neurónové siete boli do značnej miery inšpirované fungovaním biologických neurónových sietí v mozgu. Základnou jednotkou neurónovej siete sú neuróny. Obráz (napríklad 28x28 pixelov) vstupuje do prvej vstupnej vrstvy, ktorá obsahuje pri plne prepojenej sieti 784 neurónov, kde

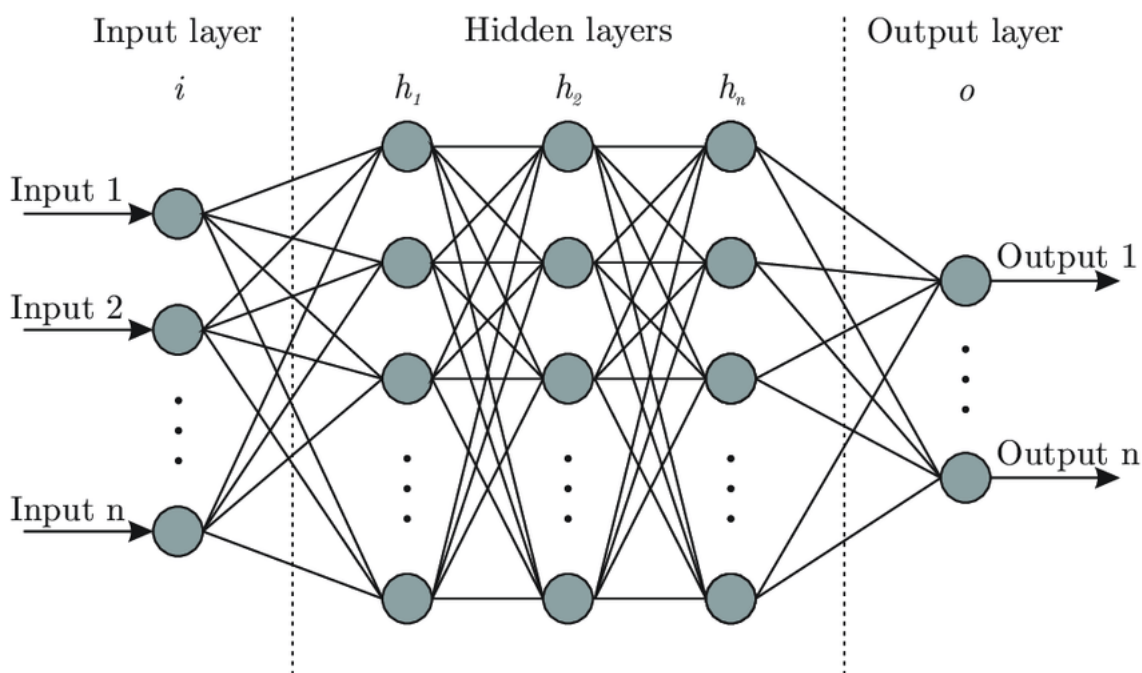
každý z nich obsahuje hodnotu v intervale $< 0, 1 >$. Hodnota daného neurónu určuje jasovú úroveň jedného pixela obrazu.

Posledná (výstupná) vrstva siete obsahuje jej výstup, teda meno triedy, do ktorej zaraďuje neurónová sieť vstupný obraz.

Medzi vstupnou a výstupnou vrstvou sa nachádzajú skryté vrstvy. Výstupné hodnoty jednej vrstvy sú vstupné hodnoty ďalšej vrstvy. Neuróny skrytých vrstiev obsahujú funkcie, ktoré na základe vstupov všetkých neurónov predchádzajúcej vrstvy vynásobených váhami, ktoré sú pridelené jednotlivým hranám (z každého neurónu predchádzajúcej vrstvy ide hrana do každého neurónu nasledujúcej vrstvy) a pripočítania prahovej hodnoty (*bias*) pomocou nelineárnej aktivačnej funkcie vrátia výstupnú hodnotu. Základná štruktúra plne prepojenej siete je ilustrovaná na obrázku 1.2.

To platí pre plne prepojené neurónové siete. Trochu inak sú usporiadané tzv. konvolučné neurónové siete, kde neuróny ďalšej vrstvy nie sú prepojené s každým neurónom predchošej vrstvy.

Neurónové siete sa dokážu *trénovať*. Trénovaním neurónovej siete sa nastavujú váhy jednotlivých hrán a prahové hodnoty (*bias*) pre jednotlivé neuróny.



Obr. 1.1: Základná štruktúra neurónovej siete.[2]

Príznakové metódy sú metódy, ktoré sa zakladajú na využívaní urči-

tých príznakov (using features). Pod príznakmi si predstavme charakteristiky, ktoré popisujú význačné (zaujímavé) body obrazu, napr. hrany, body zakrivenia a podobné. Na získanie príznakov slúžia detektory a deskriptory. Detektory slúžia na získanie zaujímavých bodov v obraze a deskriptory na ich popis. Príznačky sa zoskupujú do príznakového vektora a ten potom slúži na identifikáciu (rozpoznanie) objektu, napr. porovnávaním s príznakmi objektu v databáze. Problém, s ktorým sa tieto metódy stretávajú, je obnovovanie príznakov, ktoré nie sú na obraze vidno (napríklad pre sklon hlavy). Práca [8] rozdelila metódy extrahovania (získavania) príznakov na:

- (a) Generické metódy založené na hranách, obrysoch a zakriveniach
- (b) Metódy založené na šablónach (template)

1.0.2 Rozpoznávanie tváre pomocou príznakov

Prvou fázou rozpoznávania tváre je detekcia tváre v obraze. Jedná sa o proces, pri ktorom softvér nájde tvár v obraze. Na detegovanie tváre v obraze bol navrhnutý, ako prvý, algoritmus Viola-Jones, ktorý popisujeme v sekcii 1.1.

Po úspešnej detekcii tváre je potrebné v danom obraze nájsť príznaky, ktoré dokážu od seba odlíšiť rozličné tváre a budú následne využité pri rozpoznávaní. Ako fungujú metódy na hľadanie vhodných príznakov tváre, je popísané v sekcii 1.2.

Poslednou fázou je už samotné rozpoznanie, alebo aj identifikácia, teda zistenie identity danej osoby. Je potrebné zistiť, či sa tvár na obraze nachádza niekde v databáze, teda ju vieme spojiť s menom. Teoreticky to nevyzerá zložito, avšak tváre ľudí vo väčšine prípadov nebudú nasnímané ideálnym spôsobom. Za ideálny spôsob považujeme kontrolované podmienky, teda osobu, ktorá sa pozerá priamo do kamery, nie je príliš naklonená, nemá ani čiastočne zahalenú tvár a podobné.

Z reálneho života však vieme, že kamery sa väčšinou nachádzajú nad hlavami ľudí, najčastejšie v rohoch miestností. Dochádza tak ku zakriveniu obrazu, nasnímaný obraz je pod uhlom a preto sa ťažšie identifikuje. Ďalšie problémy nastávajú ak si človek nejakým spôsobom zahaľuje tvár, či už je to šatka cez ústa, slnečné okuliare a podobne. Všetky tieto veci ihneď komplikujú rozpoznávanie tváří.

1.0.3 Problémy rozpoznávania tváří

Existuje mnoho ďalších problémov, ktoré sa týkajú rozpoznávania tváří, jednému z nich sa venovala práca [1]. Jedná sa o problém veľkých tzv. tréovacích údajov (datasetov). Pre upresnenie tejto informácie, nie je problémom, že by neexistovali veľké datasety, ale problémom je, že nie sú verejné. Vo väčšine prípadov ich vlastní internetoví giganti ako napríklad Google alebo Facebook. Ako dobrý príklad využili najmodernejšiu metódu (v roku 2015) na rozpoznávanie tváří od Google, ktorá pracovala na viac ako 200 miliónoch obrazov a s 8 miliónmi unikátnych identít. Toto číslo bolo skoro 3 krát väčšie ako ktorýkoľvek verejný dataset. Samozrejme autori priznávajú, že vytvorenie takejto veľkej databázy je nad sily všetkých akademických výskumných skupín.

Ďalším problémom na rozdiel od predchádzajúceho nie je informatický. Jedná sa o spoločenský problém a tým je ochrana súkromia. V demokraticky fungujúcich štátoch je neprijateľné, aby vláda, alebo súkromné organizácie využívali spôsoby na nasnímanie a následné rozpoznanie tváří bez vedomia danej osoby. Avšak tieto práva sa dajú obísť ak má štát validný dôvod (napríklad pri hrozbe aktu terorizmu). Zároveň, pri veľkom množstve súkromných organizácií, hlavne teda internetových gigantoch, sa ľudia vzdávajú svojho práva na súkromie, väčšinou tak, že odsúhlasia *Podmienky užívania*.

1.1 Detekcia tváre

Existujú rôzne algoritmy na detekciu tváre, ako sú Viola-Jones alebo LBP algoritmus. Vzhľadom na to, že Viola-Jones je považovaný za základný algoritmus na detekciu tváre, sústredíme sa na popis tohto algoritmu.

Algoritmus vznikol v roku 2001. Viola-Jones vyžaduje osobu, ktorá sa pozerá priamo do kamery a jej hlava by sa nemala nakláňať ani veľmi otáčať do strán. Algoritmus teda vyžaduje čiastočne kontrolované podmienky na úspešnú detekciu tváre.

Pri popise algoritmu vychádzame z dokumentu [15]. Algoritmus využíva pri detegovaní tváre Haarove príznaky. Haarove príznaky využívajú vlastnosti svetlosti obrazu, napríklad oblasť očí je tmavšia ako oblasť líc, podobne zas oblasť nosa je svetlejšia ako oblasť očí a podobné. Zaujímavé sú teda lokácie a veľkosti očí, úst a nosa. Môžeme ich vidieť na obrázku 1.2.

HAAR-LIKE FEATURES



This Haar-like feature identifies the eye region



This Haar-like feature identifies the nose

Obr. 1.2: Na obrázku vidíme ako Haarove príznaky identifikujú nos (dole) a oči (hore). [13]

Samotný Viola-Jones algoritmus využíva na natrénovanie algoritmus Ada-Boost, ktorým vyberie najlepšie príznaky a zároveň nimi natrénuje klasifikátor. Algoritmus vytvorí jeden silný klasifikátor skombinovaním jednoduchých klasifikátorov. Tento proces vieme zapísať nasledujúcou sumou:

$$h(x) = \text{sgn}\left(\sum_{j=1}^M a_j h_j(x)\right)$$

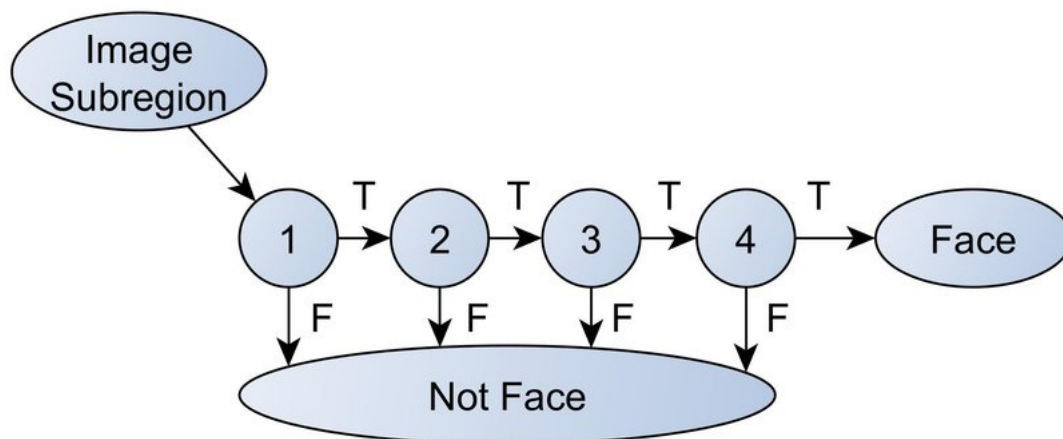
kde a_j reprezentuje váhu slabého klasifikátora h_j . Jednotlivé slabé klasifikátory využívajú prahovú funkciu, založenú na Haarových príznakoch.

$$h_j(x) = \begin{cases} -s_j & \text{ak } f_j < \text{prahová hodnota} \\ s_j & \text{inak} \end{cases}$$

kde f_j je Haarov príznak

Algoritmus zároveň využíva tzv. kaskádový klasifikátor ktorý reprezentuje postupnosť silných klasifikátorov, kde sa časť obrazu posunie do ďalšieho klasifikátora iba vtedy, ak predošlý klasifikátor úspešne klasifikoval, že

je tam tvár. Takto sa rieši aj časová náročnosť, pretože ak klasifikátor neuspeje v niektorej z fáz proces sa ukončí. Ak uspeje prejde na ďalšiu fázu a takto pokračuje až kým nedokončí klasifikáciu alebo neuspeje. Tento proces môžeme vidieť na obrázku 1.3



Obr. 1.3: Na obrázku môžeme vidieť spôsob akým pracuje kaskádový klasifikátor [16].

1.2 Metódy získavania príznakov

V nasledujúcej sekcii si predstavíme niektoré metódy získavania príznakov konkrétne.

1.2.1 Histogram of oriented gradients - HOG

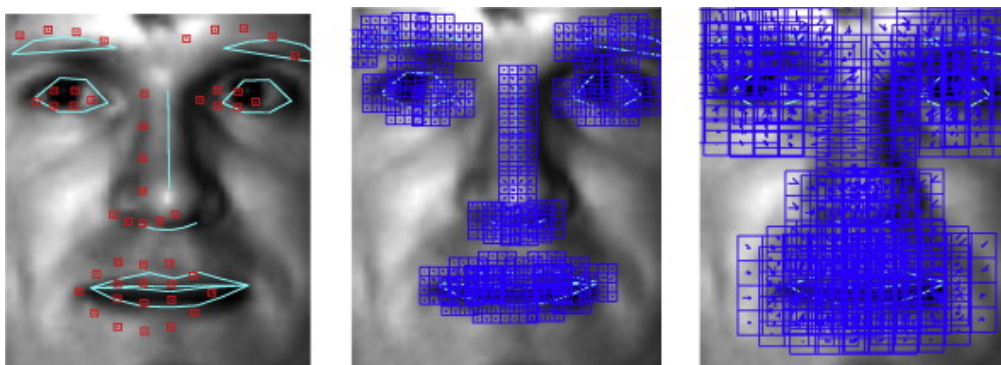
Histogram orientovaných gradientov je vizuálny deskriptor, ktorý sa používa v počítačovom videní na spracovanie obrazu za účelom detegovania objektov. Prvotná myšlienka bola navrhnutá Robertom K. McConnellom v roku 1986 avšak do pozornosti verejnosti sa dostal až v roku 2005, keď Navneet Dalal a Bill Triggs prezentovali svoju prácu, ktorú vykonali na týchto HOG deskriptoroch.

HOG počítá výskyty gradientov vo vybranej časti obrazu, pričom gradient vyjadruje mieru veľkosti zmeny jasu obrazovej funkcie a dokáže určiť aj jej smer. Základná myšlienka tejto metódy je, že výskyt objektu a tvaru v obraze môže byť opísaný gradientami intenzity jasu alebo smerovaním hrán, ktoré sú kolmé na smer gradientu. Obraz sa rozdelí na malé časti, a následne na

pixeloch každej jednej časti sa spustí HOG. Samotný deskriptor je spojenie vzniknutých histogramov.

Praktické využitie histogramu orientovaných gradientov je zaznamenané v práci [3]. Práca bola publikovaná v roku 2009. Táto práca sa zamerala na preskúmanie možností využitia HOG pri rozpoznávaní tvári. Postup, ktorý zvolili začína normalizáciou tváre (teda úprava zaklonenia tváre, odstránenie šumu a podobné) a následne aplikovaním HOG. Táto práca sa zamerala na zlepšenie presnosti rozpoznávania tváre (až o 13 percent) za použitia HOG. Na obrázku 1.4 môžeme vidieť vykreslenie HOG príznakov na tvári. Aby docielili tento výsledok, držali sa troch zásad:

- (a) rovnomerné vzorkovanie HOG príznakov, aby docielili robustnosť pre detekciu príznakov tváre
- (b) využiť dimenzionálnu redukciu v HOG, aby docielili odstránenie zbytočného nadbytku dát a zvýšili efektívnosť počítania
- (c) pri výbere najvhodnejšej veľkosti častí (patch size) využili na porovnanie dáta vytiahnuté za pomoci HOG z obrazov, ktoré mali iný patch size.



Obr. 1.4: Obráz naľavo zobrazuje 49 zaujímavých bodov, ktoré boli lokalizované pomocou AAM (Active Appearance Model). Obráz v strede zobrazuje vybrané HOG deskriptory na veľkosti 24x24 (patch size). Obráz vpravo zobrazuje HOG deskriptory na veľkosti 64x64. [3]

1.2.2 SIFT

Nasledujúca definícia SIFT je motivovaná knihou [4].

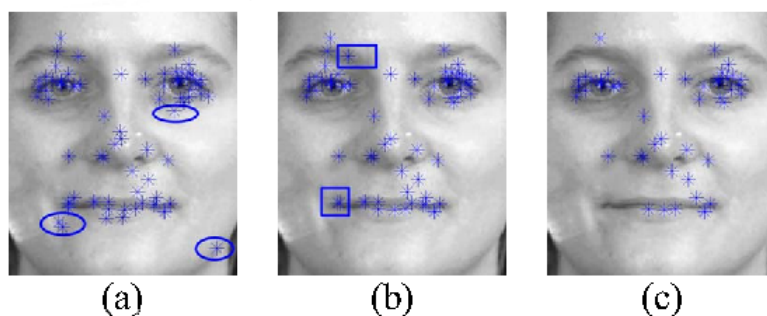
SIFT (The Scale Invariant Feature Transform) je veľmi využívaná metóda pri rozpoznávaní objektov. Jedná sa o tzv. škálovo a rotačne invariantný detektor a deskriptor zaujímavých bodov. SIFT algoritmus pracuje v dvoch krokoch:

- (a) detektor nájde zaujímavé body v obraze
- (b) pre každý bod z prvého kroku deskriptor vytvorí jeho popis

Detektor využíva pri vyhľadávaní záujímavých bodov škálový priestor. Obraz je následne podvzorkovaný do k veľkostí (tzv. oktáv). V každej tejto oktáve je následne vytvorených n ďalších obrazov opätovným filtrovaním pôvodného obrazu Gaussiánom, čím dostaneme rozdiel Gaussiánov. Príklad 1D Gaussiánu:

$$g(x) = \frac{1}{\sqrt{2\pi}\rho} e^{-x^2/2\rho^2}$$

kde ρ označuje písmenom sigma štandardnú odchýlku. Obrazy teda postupne dostávame vyhladzovaním so štandardnou odchýlkou. Pri vyhladzovaní obrazu A pomocou Gaussiánu dostaneme obraz B, ďalší obraz dostaneme vyhladzovaním B atď. V každej oktáve sa následne všetky susedné obrazy odčítajú čím dostaneme $n - 1$ obrazov v každej oktáve. Následne sú lokálne extrémny identifikované ako zaujímavé body. Oktáva, v ktorej bol zaujímavý bod nájdený, bude predstavovať škálu daného zaujímavého bodu a určí veľkosť okolia, ktoré sa použije pri extrakcii deskriptora. Na obrázku 1.5 môžeme vidieť príznaky metódy SIFT.



Obr. 1.5: Na obrázku vľavo vidíme všetky zaujímavé body nájdeme metódou SIFT. V strede sú zobrazené zaujímavé body, ktoré ostali aj po odstránení tých s nízkym kontrastom. Vpravo sú konečné príznaky, ktoré ostali aj po ďalšej filtrácii.[5]

Deskriptor: Hodnota deskriptora sa vypočíta z histogramov orientácií pre 16 štvorcových oblastí v jeho okolí pre každý zaujímavý bod. Dôležité je zistiť orientáciu každého bodu a následne vytvoriť deskriptor na základe tejto orientácie. Každú oblasť následne navzorkujeme 16 bodmi a v každom z nich (označené ako A_{ij}) vypočítame veľkosť (označená ako M_{ij}) a orientáciu (označenú ako R_{ij}) gradientov na základe nasledujúcich vzorcov:

$$M_{ij} = \sqrt{(A_{ij} - A_{i+1,j})^2 + (A_{ij} - A_{i,j+1})^2}$$

$$R_{ij} = \arctan 2((A_{ij} - A_{i+1,j}), (A_{ij} - A_{i,j+1}))$$

kde je hodnota arcus tangens s oborom hodnôt $(-\pi, \pi]$. Týmto hodnotám sú priradené váhy pomocou Gaussovho kruhového okna. Z váhových hodnôt v 16 bodoch sú následne vytvorené histogramy orientácií, pričom každý histogram pokrýva jednu zo šiestnástich oblastí z okolia zaujímavého bodu. Dĺžka šípky v histograme predstavuje sumu veľkostí gradientov v blízkosti daného smeru v oblasti. V každom histograme je 8 smerov, čo pri počte 4x4 oblastí znamená 128 hodnôt deskriptora pre jeden zaujímavý bod.

I napriek výhodám, ktoré SIFT ponúka pri rozpoznávaní objektov a tvárí, metóda nie je dostatočne rýchla, preto nie je ideálna pri rozpoznávaní tvárí.

1.2.3 SURF

Nasledujúca definícia SURF je motivovaná knihou [4].

SURF (Speeded Up Robust Features) je technika, ktorá je považovaná za vylepšenie techniky SIFT a iných podobných techník. Podobne ako SIFT aj SURF obsahuje detektor na nájdenie zaujímavých bodov a deskriptor na ich opísanie.

Detektor SURF metódy podobne ako SIFT deteguje zaujímavé body v škálovej pyramíde, avšak namiesto Gausiánov využíva aproximáciu Gausiánov druhej parciálnej derivácie v smere y a xy pomocou "box filtrov". Pri vytváraní škálového priestoru sa pri tejto metóde nevyužíva zmenšovanie obrazu a následná konvolúcia, ale naopak, konvolúcia pôvodného obrazu s postupne sa zväčšujúcimi filtermi. Na nájdenie zaujímavých bodov je na obraz aplikované potlačenie nemaximálnych hodnôt v okolí 3x3x3.

Deskriptor: Pri vytváraní deskriptora zaujímavých bodov je potrebné si najprv zistiť jeho orientáciu a v druhom kroku vytvoriť štvorcovú oblasť zarovnanú podľa tejto orientácie. Pre určenie orientácie zaujímavého bodu sa najskôr vypočítajú odozvy Haarovho waveletu (jedná sa o postupnosť škáľujúcich funkcií) v x -ovom a y -ovom smere v kruhovom okolí bodu s priemerom $6s$, pričom s je veľkosť aktuálnej škály. Vypočítaným odozvám sú

potom priradené váhy pomocou Gaussovho kruhového okna a sú reprezentované ako vektor v priestore. Dominantná orientácia zaujímavého bodu je najdlhší z vektorov vypočítaných ako suma vertikálnych a horizontálnych odoziev, získaných pomocou otočného okna ako suma všetkých odoziev danej orientácii.

V ďalšom kroku je vytvorená v okolí zaujímavého bodu štvorcová oblasť zarovnaná podľa orientácie. Oblasť je rozdelená na 4x4 podoblastí, pričom v každom sú vypočítané 4 príznaky na 5x5 pravidelne rozmiestnených bodoch. Príznakmi sú:

- (a) suma odoziev Haarovho waveletu v horizontálnom a vertikálnom smere
- (b) suma absolútnych hodnôt odoziev Haarovho waveletu v horizontálnom a vertikálnom smere.

Deskriptor zaujímavého bodu potom obsahuje 4 hodnoty pre každú zo 16 oblastí (teda spolu 64 hodnôt).



Obr. 1.6: Prepojenie bodov metódy SURF aj napriek miernemu otočeniu a zmene výrazu jednej z tvárí.[6]

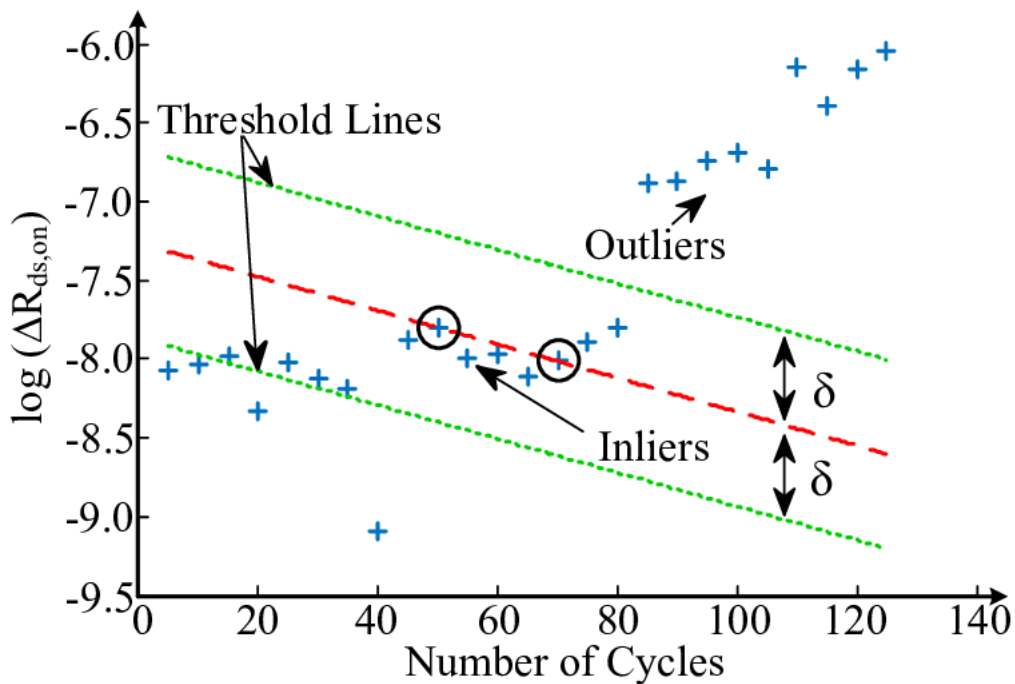
Využitie metódy SURF je popísané v práci [6]. V tejto práci využívajú metódu SURF na rozpoznávanie tvárí, tak ako bolo už v sekcii 1.1.3 vysvetlené. Po vypočítaní príznakového vektora pre tvár, sa následne porovnávajú

všetky príznakové vektory iných obrazov a hľadá sa najbližšia zhoda (najmenšia vzdialenosť dvoch vektorov). Na obrázku 1.6 môžeme vidieť prepojenie SURF príznakov na dvoch obrázkoch, kde sa jedná o najbližšiu zhodu. Navyše sa v tejto práci využíva, konkrétne na porovnávanie obrazov, metóda RANSAC (Random Sample Consensus).

Cieľom tohto algoritmu je odhadnúť matematickú entitu z množiny dát, ktoré obsahujú údaje, ktoré sa veľmi výrazne odlišujú od ostatných (tzv. outliers). Takéto zaradenie môžeme vidieť na obrázku 1.7. Myšlienkou je vybrať niekoľko náhodných bodov zdanej množiny a vykonať odhad len s týmito bodmi. Po tom ako prebehne transformácia, testovacie body budú premietnuté na trénovací obraz. Následne sa body označia podľa toho či sa nachádzajú v danom rádiuse (inliers) alebo nie (outliers).

Po niekoľkých zopakovaných iteráciách predchádzajúceho postupu sa zoberú všetky body, ktoré boli označené ako tie čo sa nachádzali v rámci rádiusu, a využijú sa ako meradlo na zistenie podobnosti medzi testovacím a trénovacím obrazom.

Základnou myšlienkou RANSAC algoritmu je, že čím väčšia je množina dát, tým sa zvyšuje pravdepodobnosť, že vypočítaná matica je správna.

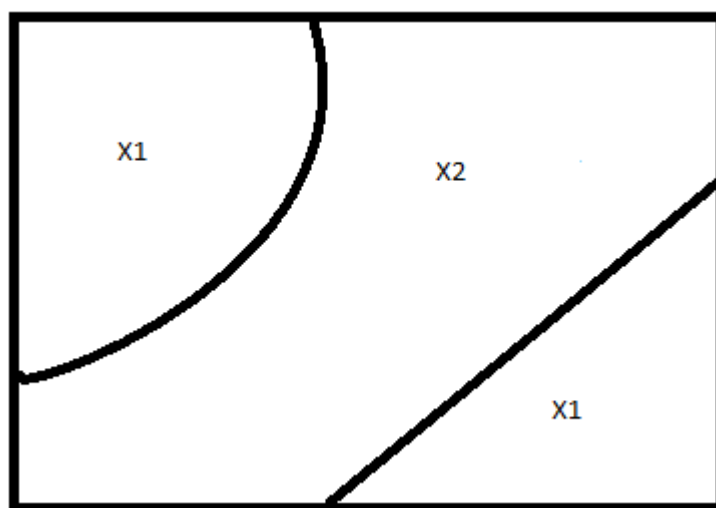


Obr. 1.7: Na obrázku môžeme vidieť body mimo danej hranice (outliers) a body v rámci danej hranice (inliers).[9]

1.3 Klasifikátory

Nasledujúce definície klasifikátorov sú motivované knihou [4]. Na vyriešenie problému rozpoznávania tváre budeme potrebovať klasifikátor. Pod klasifikátorom rozumieme zariadenie, ktoré určí triedu objektu na základe získaných príznakov objektu.

Pri klasifikácii podľa určitého pravidla vieme zaradiť vektor neznámeho objektu do jednej z klasifikačných tried. Toto pravidlo rozdelí priestor na určitý počet rozhodovacích oblastí, kde všetky vektory v danej oblasti opisujú objekty jednej klasifikačnej triedy. Rozhodovacia oblasť môže a nemusí byť súvislá (viď obrázok 1.8). Hranice medzi oblasťami sa nazývajú rozhodovacie hranice.



Obr. 1.8: Rozhodovacie oblasti klasifikačných tried X1 a X2. Rozhodovacia oblasť X1 nie je v tomto prípade súvislá.

Podľa typu rozhodovacieho pravidla môžeme rozdeliť klasifikátory na nasledovné skupiny:

- (a) Lineárne metódy, ktoré využívajú vzdialenosti medzi príznakmi. Jedná sa konkrétne o techniku najbližších susedov, diskriminačnú analýzu, mechanizmy podporných vektorov a iné.
- (b) Štatistické metódy, ktorých cieľom je minimalizácia počtu nesprávne klasifikovaných objektov na základe ich štatistických vlastností.
- (c) Rozhodovacie stromy, ktoré dokážu klasifikovať aj nemetrické príznaky.

Existuje ešte iné delenie klasifikačných metód na riadené a neriadené.

- (a) **Neradené metódy.** Jedná sa o metódy, kde sa určujú klasifikačné triedy len na základe podobnosti údajov, bez toho aby sme vedeli do ktorej triedy popisované objekty skutočne patria. Všetky premenné sú rovnocenné, žiadna neriadi proces učenia. Patria sem metódy ako K priemerov (K určuje počet), hierarchické zhlukovanie, Kohonenove siete (samoorganizujúce mapy) a iné.
- (b) **Riadené metódy.** Existuje jedna cieľová premenná, ktorá riadi proces učenia tak, aby ostatné premenné čo najlepšie povedali hodnotu cieľovej premennej. Medzi riadené klasifikátory patria napríklad Bayesovské modely, techniky najbližších susedov, rozhodovacie stromy, diskriminačná analýza, mechanizmy podporných vektorov a iné.

Pri klasifikácii vyhodnocujeme dosiahnutú chybu. Chybná klasifikácia môže nastať napríklad vtedy, keď sa príznakové vektory rôznych objektov veľmi podobajú, teda vektor x patrí klasifikačnej triede $X1$ ale klasifikátor ho zaradí do klasifikačnej triedy $X2$. Ak sú teda objekty porovnateľné zvyšuje sa neistota pri zaradení objektov do klasifikačných tried. Niektoré klasifikátory v tomto prípade majú zaradenú možnosť odmietnuť klasifikáciu. Alebo majú špeciálnu triedu pre nezaradené. Ovplyvniť počet nezaradených objektov môžeme pomocou nastavenia prahu.

1.3.1 Lineárne klasifikátory

Jedná sa o riadené klasifikátory. Výhodou lineárnych klasifikátorov je ich jednoduchosť a nízka výpočtová náročnosť. Pri iných metódach (ako napríklad metóde najbližších susedov) je trénovacia množina potrebná vždy pri určovaní triedy nového bodu. Pri lineárnom klasifikátore po učení (natréňovaní) už trénovacia množina nie je potrebná.

Predstavme si klasifikáciu do dvoch tried ω_1 a ω_2 . Rovnica nadroviny oddeľujúcej v príznakovom priestore tieto dve triedy je:

$$w^T x + b = 0$$

kde x je vektor príznakov, w je váhový vektor a b je prah (posunutie). Hodnota tejto rovnice je nulová pre body, ktoré ležia na rozdeľujúcej nadrovine. Ak označíme $f(x) = w^T x + b$, tak úlohou lineárneho klasifikátora je nájsť taký váhový vektor w (normálový vektor nadroviny) a b , aby platilo:

$$f(x) \geq 0 \text{ pre } x \in \omega_1$$

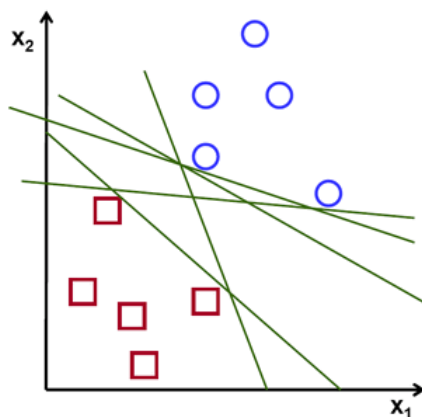
$$f(x) < 0 \text{ pre } x \in \omega_2$$

Pri rozpoznávaní tvári sa však málokedy rozhodujeme len medzi dvoma triedami (tvármi). Lineárny klasifikátor sa dá využiť aj pri klasifikácii do K tried. Môžeme postupovať dvoma spôsobmi:

- (a) Pre každú klasifikačnú triedu vytvoríme rozhodovaciu funkciu, ktorá oddelí vzorky tejto triedy od vzoriek ostatných tried. Budeme teda mať K rozhodovacích funkcií.
- (b) Pre každú dvojicu rôznych klasifikačných tried vytvoríme rozhodovaciu funkciu oddelujúcu vzorky týchto dvoch tried bez ohľadu na vzorky ostatných tried. Takže budeme mať $K(K - 1)/2$ rozhodovacích funkcií.

1.3.2 Support vector machine

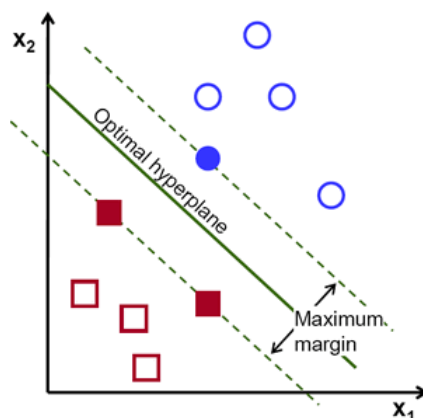
Support vector machine alebo skratene SVM je špeciálny prípad lineárneho klasifikátora. Vychádzame z dokumentácie [14]. Pôvodne SVM vznikol ako ideálny binárny klasifikátor ale neskôr bol rozšírený aby mohol pracovať aj s väčším množstvom tried. SVM mapuje príznakové vektory vo viacdimenzionálnom priestore a vytvára optimálnu diskriminačnú funkciu, ktorá rozdeľuje tréningové dáta. Optimálna diskriminačná funkcia je vytvorená na základe najväčšej minimálnej vzdialenosti od príznakov rôznych tried. Pre lepšiu vizuálnu predstavu si funkčnosť ukážeme v 2D priestore. Na obrázku 1.9 môžeme vidieť rôzne klasifikácie dvoch tried rôznymi rozhodovacími hranicami.



Obr. 1.9: Na obrázku vidíme rôzne predeľovacie polpriamky, ktoré by riešili daný problém klasifikácie.[14]

Polpriamky (vo viacdimenzionálnom priestore ako v 2D sa jedná o nadroviny), ktoré sa nachádzajú blízko pri hociktorej z tried klasifikuje klasifiká-

tor bezchybne na trénovacej množine (na obrázku 1.11 znázornenej modrými kruhmi a červenými štvorcami), ale nemusí tak dobre fungovať na testovacej množine. Teda sa môžu objekty zle klasifikovať. SVM tento problém rieši tak, že si nájde polpriamku, ktorá je v najväčšej minimálnej vzdialenosti od všetkých príznakových vektorov trénovacej množiny. Teda ako môžeme vidieť na nasledujúcom obrázku 1.10.



Obr. 1.10: Na obrázku vidíme optimálnu predeľujúcu polpriamku (v prípade 2D priestoru, inak sa jedná o rovinu, alebo nadrovinu), ktorá sa nachádza v maximálnej minimálnej vzdialenosti od príznakov.[14]

Formálne sa táto nadrovina zapisuje ako

$$f(x) = \beta_0 + \beta^T x$$

kde β reprezentuje váhový vektor a β_0 je bias. Spomedzi všetkých spôsobov ako reprezentovať túto nadrovinu si SVM vyberá nasledujúcu

$$|\beta_0 + \beta^T x| = 1$$

kde x predstavujú trénovacie príznakové vektory vyskytujúce sa najbližšie pri nadrovine. Tieto príznakové vektory (ktoré sa nachádzajú najbližšie pri nadrovine) sa označujú ako podporné vektory (support vectors). Vzdialenosť medzi x a nadrovinou sa dá formalizovať ako

$$vzdialenosť = \frac{|\beta_0 + \beta^T x|}{\|\beta\|}$$

Pomocou vzdialenosti sa dá postupne vypočítať ideálna nadrovina.

Kapitola 2

Návrh riešenia

V prvom rade si vysvetlíme špecifiká nášho problému oproti všeobecnému príznakovému postupu na rozpoznávanie tvárí, ktorý sme uviedli v kapitole 1.

Prvým rozdielom je, že sa nejedná o kontrolované podmienky. Nejedná sa o ne z dôvodu, že snímame obraz živých ľudí pri ich každodennom živote. Môžu nastať prípady, kde bude mať vstupujúca osoba otočenú hlavu do strany, alebo ju bude mať z časti zakrytú a podobné. Takže môže nastávať problém pri detekcii tváre. Šťasti vieme tomuto problému dopomôcť umiestnením kamery.

Druhý rozdiel je ten, že my nepracujeme so statickými snímkami, ale snímame video, a teda musíme pracovať s video záznamom. Začneme teda detekciou tváre a pohybu v zázname. Robíme to pomocou odčítania dvoch nasledujúcich obrazov. Detekcia pohybu je podstatná hlavne preto aby sme vedeli rozlíšiť či sa jedná o pohyb veľkého objektu (napríklad človeka) alebo o pohyb nejakého malého objektu. Keď sme pohyb kategorizovali ako pohyb veľkého objektu, potrebujeme ešte rozhodnúť či sa jedná človeka. V takom prípade nám môže pomôcť rozhodnúť algoritmus na detekciu kože. Tento algoritmus nám pomôže v prípade, keď sme zaznamenali pohyb človeka, ale nedetegovali sme tvár. Na detekciu tváre využívame algoritmus Viola-Jones.

Tretím rozdielom je to, že náš algoritmus musí pracovať v čiastočne kontrolovaných podmienkach. Snímaná tvár môže byť natočená, čiastočne otočená, preto je potrebné ju geometricky zarovnať. Robíme to preto, lebo v prípade laboratória by sme si dopredu nasníмали snímky z priameho pohľadu, teda v kontrolovaných podmienkach. Na takto zarovnanú tvár vypočítame HOG príznaky. Príznakový vektor pošleme do SVM klasifikátora, ktorý robí samotné rozpoznávanie (identifikáciu) osoby z databázy. Na vyhodnotenie úspešnosti klasifikátor robíme testovanie s viacerými testovacími scenármi. Postupnosť jednotlivých krokov môžeme vidieť na obrázku 2.1.

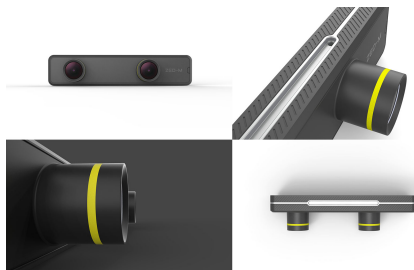


Obr. 2.1: Jednotlivé problémy a postupnosť v akej sa riešia.

Jednotlivé kroky našej postupnosti popíšeme podrobnejšie.

2.1 Snímanie obrazu

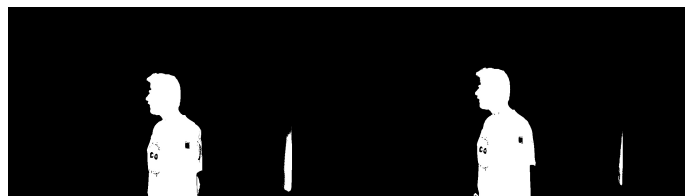
Na snímanie používame kameru ZED mini (obrázok 2.2). Keďže sa jedná o profesionálnu kameru, potrebujeme aj špeciálny softvér na získanie obrazu. Ide o ZED API, ktorý si musí užívateľ nainštalovať pred používaním. Následne si vieme pomocou programovacieho jazyka Python vyžiadať zdieľanie obrazu s Python Idle a teda s ním môžeme pracovať. Obraz budeme snímať z výšky približne jedného metra a osemdesiatich centimetrov. Vzhľadom na vzniknutú situáciu s Covid-19 sme nemohli snímať obraz v podmienkach laboratória, ale museli sme snímať obraz v zmenených priestoroch a teda musíme pracovať aj za zmenených podmienok.



Obr. 2.2: ZED mini kamera. [19]

2.2 Detekcia pohybu a tváre

Po nasnímaní obrazu prichádza na rad detekcia pohybu a tváre. Algoritmus na detekciu pohybu [20] funguje na princípe sledovania obrazu a porovnávaní jednotlivých nasledujúcich snímok. Algoritmus v nich hľadá zmenu, teda pohyb. Pre lepšiu vizualizáciu si algoritmus celý obraz zafarbí na čierne a následne pixely, v ktorých je zaznamenaná zmena označí na bielo. Podľa veľkosti a spojitosti tohto nového čierno-bieleho obrazu sa dá usúdiť či sa jedná o pohyb zapríčinený niečím veľkým teda napríklad človekom. Vizualizácia funkčnosti algoritmu je zachytená na obrázku 2.3. Ak dôjde ku pohybu algoritmus bude očakávať načítanie tváre.



Obr. 2.3: Detekcia pohybu pomocou odstráneného pozadia.

Následne budeme na detekciu tváre využívať algoritmus Viola-Jones. Algoritmus bude neprestajne hľadať v obraze tváre. V momente keď načíta tvár uloží snímku. Takto uloží niekoľko snímok, z ktorých sa bude neskôr vyberať tá najlepšia. Samozrejme však môžu nastať prípady kedy sa tvár nepodarí načítať. Môže sa napríklad jednať o prípady, kedy je tvár vstupujúcej osoby výrazne natočená a Viola-Jones ju nedokáže identifikovať, alebo má zakrytú časť tváre, napríklad okuliarmi, fúzami, bradou apod.

Môžu však nastať aj prípady, kedy algoritmus na detekciu pohybu detegoval pohyb a nejednalo sa oň ľudským pričinením. Napríklad ak spadne objekt pred kameru, alebo prievan pohne dverami. Aby sme vedeli rozpoznať

prípady, kedy sa jednalo o pohyb človeka, využijeme algoritmus na detekciu kože.

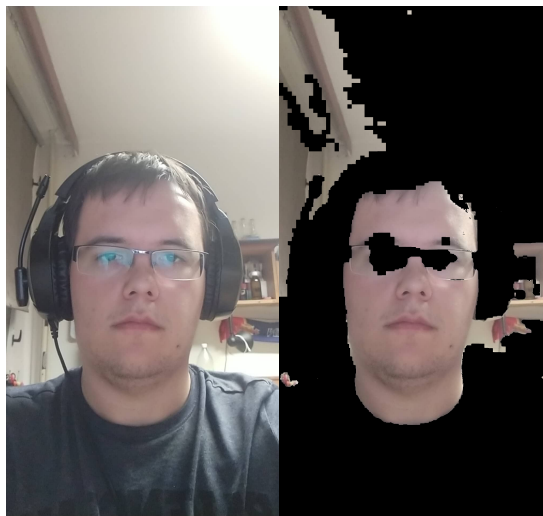
Algoritmus na detekciu kože [12] funguje priamočiara a to tak, že z nájdennej snímky vytiahne všetky pixely spektra farby kože. Využíva pri tom vnímanie kože podobné tomu ako ju vnímajú ľudia. Využívajú sa tri základné zložky a to odtieň (H), sýtosť (S) a hodnota (V), ktorá popisuje svetlosť (alebo intenzitu). Výpočtové vzorke pre jednotlivé zložky sú nasledovné:

$$H = \arccos \frac{0.5 * ((R - G) + (R - B))}{\sqrt{(R - G)^2 + (R - B) + (G - B)}}$$

$$S = 1 - 3 \frac{\min(R, G, B)}{R + G + B}$$

$$V = \frac{1}{3}(R + G + B)$$

Nejedná sa o bezchybný algoritmus, pretože sa môžu v miestnosti nachádzať objekty, ktoré majú farbu podobnú farbe kože. Rovnako môže človek prísť zahalený. Pre lepšiu vizualizáciu algoritmus ukáže vyextrahovanú "kožu" z obrazu. Funkčnosť algoritmu môžeme vidieť na obrázku 2.4.



Obr. 2.4: Na obrázku môžeme vidieť funkčnosť algoritmu na detekciu kože. V tomto konkrétnom prípade zaznamenal aj časť pozadia.

Ak sa jedná o človeka (algoritmus na detekciu kože tak rozhodne) a nepodari sa nám nasnímať jeho tvár v určitom časom intervale, bude určená situácia klasifikovaná tak, že sa v miestnosti nachádza neidentifikovaná osoba.

Ak však algoritmus na rozpoznanie kože nepotvrdí, že sa jedná o človeka, algoritmus na pohyb nebude reagovať.

2.3 Geometrické zarovnanie tváre

Po úspešnom načítaní tváre, vyberáme z načítaných snímok najlepšiu. Hlavná podmienka pri výbere snímky je detegovanie oboch očí na snímke. Detekcia očí je dôležitá pri otáčaní tváre. Zarovnať tvár je potrebné pre ľahší a presnejší výpočet príznakov na snímke. Po nájdení vhodnej snímky spustíme algoritmus na zarovnanie tváre.

Existujú rôzne algoritmy, ktoré dokážu docieľiť otáčanie tváre. Jeden z algoritmov si detekuje tvár a v nej potrebuje nájsť obidve oči. Algoritmus pochádza zo zdroja [11]. Určite môžeme považovať za slabinu algoritmu potrebu obidvoch očí, ale pre princíp akým funguje je nevyhnutná. Algoritmus nájde obidve oči a v nich si nájde ich centrá. Pokračuje vykreslením úsečky, ktorá spája tieto centrá, a na jej základe zistí, ktoré oko je vyššie. Ak je vyššie ľavé oko tvár otáča proti smeru hodinových ručičiek, ak pravé tak v smere. Následne sa tento nový obraz upravenej tváre uloží a môžeme s ním ďalej pracovať. Výsledok algoritmu vidíme na obrázku 2.5.



Obr. 2.5: Naľavo je pôvodný obrázok.[10] Vpravo je obrázok po vykonaní algoritmu.

Ak však nemáme snímku, na ktorej by sme vedeli úspešne spustiť tento algoritmus, použijeme iné snímky bez zarovnaní.

2.4 Príznaky

Existujú rôzne príznakové metódy, ktoré sa využívajú a niektoré z nich (konkrétne SIFT, SURF a HOG) sme si už priblížili v predchádzajúcej kapitole (v sekcii Metódy). V našom algoritme sme sa rozhodli využívať metódy HOG teda histogram orientovaných gradientov. Rozhodli sme tak kvôli malému množstvu testovacích dát.

Vzhľadom na funkcionálnosť kamery ZED Mini sme zvažovali aj ďalšie prístupy. Konkrétne išlo o metódy, ktoré sa zakladajú na RGB-D, čiže snímkach, ktoré okrem farebného spektra vedia snímať aj hĺbku v obraze. Avšak po vykonaní prieskumu sme sa rozhodli, že sa tieto prístupy využívať nebudú a to z dôvodu ich komplikovanosti. Samotné metódy sú zbytočne komplikované pre problém, ktorý riešime. Metódy, ktoré fungujú na princípe RGB-D sa používajú napríklad pri rozpoznávaní výrazu tváre.

2.5 Rozpoznávanie tváre

Z nájdených príznakov metódy HOG sa vypočíta príznakový vektor, ktorým sa následne pošle do klasifikátora. Ten na základe dát z databázy porovná vektory ľudí, ktorí majú prístup do laboratória a vráti zhodu respektíve nezhodu.

Vzhľadom na to, že množstvo dát, teda osôb, ktoré majú prístup do laboratória je malé, potrebujeme klasifikátory a metódy, ktoré pracujú na malom množstve dát (rádovo na desiatkach). Vzhľadom na vzniknutú celosvetovú pandémiu sa nám množina testovacích dát ešte zmenšilo resp. upravilo. Za klasifikátor sme si zvolili Support Vector Machine (SVM). Hlavným dôvodom pri rozhodovaní bola podpora OpenCV knižnice pre SVM, jednoduchosť a to, že SVM dokáže pracovať na malej množine dát.

Kapitola 3

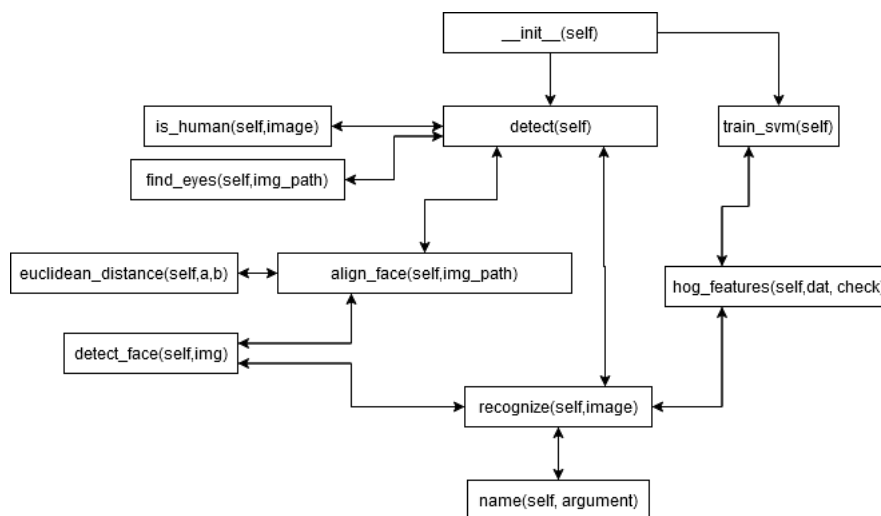
Implementácia

Na implementáciu sme si zvolili jazyk Python, konkrétne verziu 3.7. Vzhľadom na malý počet dát, s ktorým sme mali pôvodne pracovať, nebol potrebný efektívnejší jazyk. Databáza samotného laboratória by nebola o veľa väčšia ako databáza, ktorú sme si vyrobili v zmenených podmienkach. A teda pri takomto počte snímok (zhruba 100) je efektívnosť jazyka zanedbateľná.

Databáza, ktorú sme si vytvorili, je popísaná v sekcii 3.12.

Na implementáciu sme si zároveň zvolili knižnicu OpenCV, ktorú Python implementuje. OpenCV je knižnica, ktorá sa zameriava primárne na počítačové videnie. Okrem nej využívame aj Numpy knižnicu, ktorá sprostredkovoáva prácu s veľkými maticami a multidimenziálnymi poliami. V implementácii využívame metódy, ktoré sú zadané v tejto knižnici a teda nie sme autormi niektorých algoritmov.

Keďže sa jedná o prácu s rozvinutou knižnicou implementácia nie je veľmi dlhá, keďže využívame už implementované metódy. Preto sme rozhodli implementáciu vložiť do jednej triedy *Face_recognition*, ktorá neprijíma žiadny vstup pri inicializácii. Trieda obsahuje okrem svojej inicializácii 10 ďalších funkcií, ktorých funkčnosť si v nasledujúcich sekciách popíšeme. Využívanie funkcií, teda ktoré funkcie sú využívané inými môžeme vidieť na obrázku 3.1.



Obr. 3.1: Smerovanie šípky určuje smer komunikácie medzi funkciami. Ak je šípka jednosmerná, neočakáva sa odpoveď.

3.1 `__init__(self)`

Jedná sa o inicializačnú funkciu, ktorá sa automaticky zavolá pri vytvorení inštancie triedy. Funkcia v sebe zahŕňa vytvorenie premenných, ktoré sa budú neskôr využívať. Konkrétne sa jedná o polia (listy) obsahujúce trénovacie snímky pre jednotlivé triedy klasifikácie. Zároveň vytvorí klasifikátory na detekciu tváre a očí. Klasifikátory pracujú s pomocou Haarových príznakov (kapitola 1. sekcia 1.1). Inicializácia zavolá funkciu 3.2 a následne funkciu 3.11. Tu končí funkčnosť tejto funkcie a ďalej sa už nevyužíva.

3.2 `train_svm(self)`

Funkcia si na základe zoznamov z inicializácie vytvorí trénovaciu množinu pre SVM. Zároveň si vytvorí aj pole, ktoré slúži na označenie tried jednotlivých obrázkov, aby si klasifikátor (SVM) vedel priradiť obrázky ku triedam. Hodnoty na označenie používa číselné, pretože to SVM vyžaduje. Z tohto dôvodu závisí na poradí v akom sa obrázky nahrávajú. Ak by sa to nejakým spôsobom poprehadzovalo, došlo by s najvyššou pravdepodobnosťou ku zlej klasifikácii.

Následne prechádza každý prvok z trénovacej množiny a pokúša sa zarovnať tvár v danom obraze (volaním 3.5). Ak sa to podarí táto zarovnaná tvár sa uloží, ak nie (dôvodom, je že neboli detegované obidve oči) tak sa

samotná tvár len deteguje a uloží sa nezarovnaná. Pred uložením obrázku oboch prípadov sa ich veľkosť zmení na jednu konkrétnu (128x64).

Obrázky, ktoré sa nachádzajú v testovacej množine teraz zmení na čierne-biele a pošle ich ako parameter do funkcie 3.6 (ako parametre pošle testovaciu množinu), ktorý vypočíta príznakový vektor všetkých obrázkov. Pomocou numpy knižnice si pretransformuje tento príznakový vektor na maticu.

Poslednou funkčnosťou tejto funkcie je vytvorenie SVM klasifikátora. Ten teda zadefinuje, nastaví ako lineárny klasifikátor a vytrénuje ho pomocou matice, ktorú sme si vytvorili.

3.3 euclidean_distance(self, a, b)

Priamočiara metóda, ktorá dostane dva body a a b , a vypočíta ich euklidovskú vzdialenosť pomocou vzorca

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

kde x_1 a y_1 sú súradnice bodu a , a x_2 , a y_2 sú súradnice bodu b .

3.4 find_eyes(self, img_path)

Metóda si pomocou cesty otvorí obrázok, pomocou metódy z OpenCV knižnice cv2.cvtColor ho pretransformuje na čierne-biele a následne pomocou Haarových príznakov na detekciu tváre a očí, najprv deteguje tvár a následne oči v rámci tejto tváre. Vrátí počet očí, ktoré boli detegované.

3.5 align_face(self, img_path)

Metóda slúži na geometrické zarovnanie tváre. Najprv si algoritmus pomocou funkcie detect_face nájde tvár v obraze, následne v tomto obraze deteguje oči. Ak sa mu nepodarí detegovať oči, funkcia vráti UnboundLocalError. Ak sa podarí nájsť aspoň dve oči tak pokračuje ďalej. Algoritmus si zoradí oči podľa ich veľkosti a vyberá dve najväčšie. Ďalej pracuje už len s nimi.

V tomto bode má dve oči. Teraz si algoritmus porovná ich veľkosť a určí, ktoré je ľavé a ktoré pravé oko. Následne si vypočíta súradnice centra ľavého aj pravého oka. Algoritmus si určí, ktoré oko je väčšie respektíve, ktoré oko je vyššie. Táto informácia mu určí smer, akým bude danú tvár otáčať a taktiež mu určí pomocný bod. Ak je vyššie ľavé oko otáča proti smere hodinových

ručičiek a súradnice pomocného bodu sú (*x_pravého*, *y_lavého*), ak práve tak v smere, súradnice sú opačné. Následne vypočíta 3.3 pre všetky tri dvojice bodov. Pomocou nich vypočíta uhol, pod akým musí obraz otočiť. Následne už len použije metódu *rotate* otočí obraz o daný počet stupňov. Výstupom funkcie je tento zarovnaný obrázok.

3.6 `hog_features(self, dat, check = False)`

Funkcia využíva metódu *hog* z knižnice *skimage*. Využívajú sa tu základné odporúčané parametre. Parameter funkcie *hog_features* *dat* je, alebo sú, obrázky, ktorých príznakový vektor chceme vypočítať. Parameter *check* určuje či sa jedna o jeden obrázok (*True*) alebo viac ako jeden (*False*). Výsledkom je teda pole príznakových vektorov.

3.7 `is_human(self, image)`

Algoritmus si tak ako bolo popísané v sekcii 2.2 vypočíta maximálny a minimálnu farbu kože, a následne deteguje všetky farebné pixely v tomto intervale. Výstupom je booleanovská premenná, závislá od toho či je počet nenulových pixelov v obrázku, ktorý je zložený len z vyextrahovanej kože, vyšší (*True*) alebo nižší (*False*) ako 10000.

3.8 `recognize(self, image)`

Funkcia dostane na vstupe obrázok, ktorý ide rozpoznávať. Pomocou 3.10 nájde tvár a používa ďalej už len túto tvár. Zmení jej veľkosť na rovnakú ako v metóde 3.2. Pomocou HOG vypočíta príznakový vektor, z ktorého následne spraví maticu. Následne zavolá metódu *SVM.predict*, ktorá vráti array hodnotu triedy, na ktorú sa vstup podobal najviac. Funkcia vracia výstup funkcie 3.9 s parametrom výsledku porovnávania.

3.9 `name(self, argument)`

Jedná sa algoritmus, ktorý je podobný svojou myšlienkou metóde *SWITCHER* ale tú sme nemohli použiť, keďže neakceptuje array ako argument. Takže sa jedná len o podmienky, ktoré na základe čísel vrátia meno osoby.

3.10 detect_face(self,image)

Funkcia nájde tvár v obraze a vráti nový obraz, ktorý obsahuje iba danú tvár. Na detekciu tváre sa využívajú Haarove príznaky. Pri metóde na detekciu tváre, teda konkrétne detectMultiScale() sa ako parametre vkladajú obraz, na ktorom chcete nájsť tvár, škálujúci faktor, ktorý určuje ako je obraz zmenšený na každej jeho škále (hodnota, ktorú využívame je 1.3, jedná sa o štandardnú) a minimálny počet susedov, ktorý určuje koľko susedov by mala tvár mať na to aby sme ju za tvár mohli považovať. Tento parameter je najpodstatnejší pri zabráňovaní detekcie falošne pozitívnych tvárí (teda takých, kde sa reálne nejedná o tvár ale algoritmus ju identifikoval), čím vyšší je tento parameter, tým menej falošne pozitívnych tvárí. Ale má to aj svoju nevýhodu a to hlavne pri nie ideálnych záberoch, môže nastať prípad, že tvár nebude rozpoznaná. Táto funkcia začína na počte susedov 10 a postupne ho znižuje až kým nenájde aspoň jednu tvár. Keď je nájde vyselektuje ju z obrázka a vráti ju ako nový obrázok. Ak žiadnu nenájde vráti pôvodný obrázok.

3.11 detect(self)

Posledná funkcia je spustená počas celého behu programu. Pomocou implementácie zo zdroja [17] na prácu so ZED MINI kamerou, dokážeme vyselektovať obraz z kamery a pracovať s ním. Nastaví si teda rozšírenie 2560x1080 a zadeklaruje si prázdne pomocné polia na jednoduché obrázky a už otočené. Zároveň si zadeklaruje premennú *človek* na False (premenná hovorí o tom či sa v zábere nachádza človek). Vytvorí si ešte premennú *start* na 0, ktorá bude slúžiť na meranie času. V nekonečnom While cykle si funkcia sníma obraz pomocou kamery, pomocou algoritmu, ktorý bol popísaný v sekcii 2.2 deteguje pohyb a zároveň sa snaží detegovať tvár pomocou Viola-Jones algoritmu zo sekcie 1.1. Ak je počet pixelov pohybu väčší ako 15000 a teda došlo ku nejakému väčšiemu pohybu algoritmus si pomocou metódy 3.7 overí či sa jedná o človeka a do premennej *človek* priradí odpoveď (True/False). Ak sa teda jedná o človeka (True) a bola nájdená tvár, a ešte nemáme žiadnu zarovnanú tvár v zozname, algoritmus uloží snímku na disk a pridá jej názov do zoznamu jednoduchých snímok. Zároveň, ak sa premenná *start* rovná 0, tak sa do tejto premennej zapíše pomocou metódy *time.time()* čas. Následne (ak teda je nejaká snímka v zozname jednoduchých snímok) prejde všetky tieto snímky a pre každú zistí počet očí, ktoré vie detegovať pomocou 3.4. Ak je tento počet väčší ako 1 pokúsi sa túto tvár zarovnať. Ak sa to podarí zarovnanú tvár uloží do zoznamu zarovnaných tvár, ak nie tak pokračuje na

ďalšiu.

Ak má algoritmus aspoň jednu zarovnanú tvár tak ju rozpozná pomocou 3.8 a vypíše výsledok. A nastaví si pomocnú premennú *recognized* ako True.

Ak však algoritmus nasnímal nejakú tvár (v zozname jednoduchých tvárí sa niečo nachádza) a za 15 sekúnd od prvého zaznamenania človeka (získame to tak, že od spustenia `time.time()` odrátame *start*) , algoritmus nenašiel žiadnu ideálnu tvár na zarovnanie, tak algoritmus vyberie prvú snímku zo zoznamu jednoduchých tvárí a spustí 3.8 na tejto snímke. Opäť si pomocnú premennú *recognized* nastaví ako True.

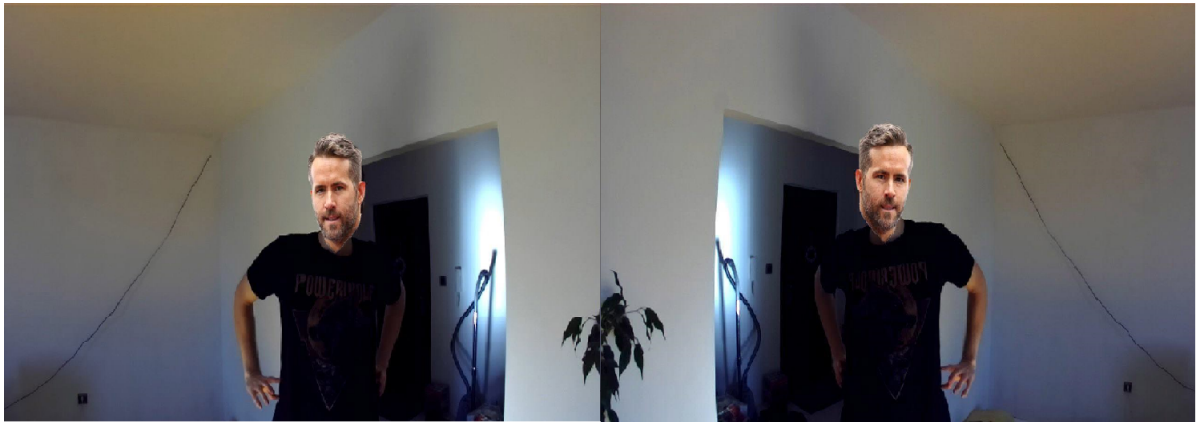
Ak rozpoznávanie prebehlo, algoritmus ešte prejde všetky snímky v zozname jednoduchých a vymaže ich z disku (aby sa zbytočne nenapĺňal), následne nastaví obidva zoznamy na prázdne, *človek* na False a *recognized* ako False, a *start* na 0.

Môže ešte nastať prípad, kedy algoritmus detegoval pohyb a našiel v obraze človeka ale nepodarilo sa nám zaznamenať tvár. Algoritmus teda počká 30 sekúnd a ak sa v tomto čase nedeteguje tvár, vyhlási nájdenie neznámej osoby.

3.12 Databáza

Pri rozpoznávaní tvárí je databáza jednou z najdôležitejších častí. Pôvodne mala naša databáza vyzeráť tak, že by obsahovala určitý počet snímok ľudí s prístupom do laboratória. Zároveň by sme si nachystali aj nejaké zábery týchto ľudí v kontrolovaných podmienkach. V obidvoch prípadoch by sa teda jednalo o prirodzené zábery, nijak neupravované iným softvérom. Vzhľadom na vzniknutú celosvetovú pandémiu vírusu COVID-19 sme však museli zvoliť alternatívny prístup.

Hlavným problémom bol nedostatok ľudí na vytvorenie databázy. Databáza mala obsahovať aspoň desať ľudí, a každý človek mal mať okolo desať snímok. Nepodarilo sa nám však získať viac ako šesť ľudí na snímanie. Rozhodli sme sa preto doplniť databázu o snímky celebrít, ktorých tváre sme pomocou photoshopu prilepili do prostredia, v ktorom snímame. Využívali sme aj augmentáciu obrazu, konkrétne stranové otočenie obrazu. Naša databáza obsahuje teda šesť reálnych ľudí a štyri celebrity. Photoshopované tváre a augmentáciu môžeme vidieť na obrázku 3.2.



Obr. 3.2: Na obrázku môžeme vidieť ukážku úpravy obrázku za pomoci dosadenia tváre a následnú augmentáciu obrázku.

Naša databáza teda obsahuje okrem autora tejto práce aj ďalších 5 reálnych osôb, spolu 2 ženy a 4 mužov. Na obrázku 3.3 môžeme vidieť jednu snímku reálnej osoby. Celebrity, ktorých zábery sme využívali, sú Ryan Reynolds, Arnold Schwarzenegger, Sylvester Stallone a Jennifer Aniston.



Obr. 3.3: Na obrázku môžeme vidieť ukážku záberu reálnej osoby.

Jednotlivé zdroje pre tváre celebrit, ktoré sme použili sú nasledujúce:

(a) **Ryan Reynolds**

- i. <https://bit.ly/2ZOftQN>
- ii. <https://bit.ly/2AncmOI>
- iii. <https://bit.ly/2BbQ4Qc>
- iv. <https://bit.ly/3gAjqYJ>
- v. <https://bit.ly/36IYtpW>

(b) **Sylvester Stallone**

- i. <https://bit.ly/2XOdRgI>
- ii. <https://bit.ly/2TPWg6Y>
- iii. <https://bit.ly/2ZT0wXm>
- iv. <https://abcn.ws/3diIH09>
- v. <https://bit.ly/2MfRmMw>

(c) **Arnold Schwarzenegger**

- i. <https://bit.ly/2MdOtvB>
- ii. <https://bit.ly/2ZQJbhy>
- iii. <https://bit.ly/36OCSN8>
- iv. <https://bit.ly/2ZSndus>
- v. <https://bit.ly/2XHPJwp>

(d) **Jenifer Aniston**

- i. <https://bit.ly/36IZw9D>
- ii. <https://imdb.to/2TRtgM0>
- iii. <https://bit.ly/3dhNNBn>
- iv. <https://bit.ly/2XjSogS>
- v. <https://bit.ly/36IZsXr>

Všetky zdroje sme skrátili pomocou *bitly*, kvôli prehľadnosti.

Kapitola 4

Testy a výsledky

V tejto kapitole si priblížime testy a aké výsledky sme pri nich docielili. Pri testovaní využívame snímky ľudí, ktoré boli nasnímané pomocou implementácie popísanej v kapitole 3. Pri prvom testovaní sme využili všetky snímky, teda snímky reálnych ľudí, ale použili sme aj snímky celebrit.

V druhom testovacom scenári testujeme len snímky reálnych osôb.

V treťom testovacom scenári testujeme kompletnú funkčnosť algoritmu.

Vo štvrtom testovacom scenári porovnávame naše výsledky z prvého testu, s výsledkami implementácie knižnice `face_recognition` na rovnakých snímkach.

4.1 Test rozpoznávania všetkých snímok

Na testovanie využívame 101 snímok 10 osôb. 6 osôb je v tejto množine reálnych (teda snímky neboli photoshopované) a 4 sú celebrity. Pri testovaní používame metódu vzájomnej validácie, kde testujeme 10 krát a vždy si vyberieme z fotiek danej osoby jednu náhodnú, ktorá bude slúžiť na testovanie a zvyšné zaradíme do testovacej množiny. Metóda na validáciu je na obrázku 4.1.

```

def validation(zoz):
    test = []
    res = []
    index = 0
    index = round(random.randint(0, len(zoz)-1))
    res.append(zoz[index])
    for x in zoz:
        if x != zoz[index]:
            test.append(x)
    return test, res

```

Obr. 4.1: Jedná sa o jednoduchý algoritmus, ktorý vytiahne jednu náhodnú snímku zo zoznamu, ktorý dostal na vstupe a zvyšné vráti v samostatnom zozname.

Pri prvom pokuse o testovanie sme dostávali rôzne chyby založené na zlej detekcii tváre. Stávali sa prípady, kde algoritmus detegoval falošne pozitívne tváre, teda detegoval tváre tam kde reálne nie sú. Problém sme vyriešili modifikáciou kódu, kde sme nastavili vyšší počet susedov (parameter pre detekciu tváre, ktorý určuje počet minimálnych susedov, ktorí sú potrební na to aby sa potenciálna tvár mohla považovať za tvár) pri detegovaní tváre.

Následne sa objavil problém, kde algoritmus na zarovnanie tváre (2.3) otáčal tváre, ale ich nezarovnával. Táto chyba bola spôsobená podobným problémom ako prechádzajúca (falošne pozitívne tváre), zistili sme, že algoritmus detegoval falošné oči a na základe nich zle zarovnával tvár (teda ju otáčal). Riešenie bolo obdobné, nastavili sme vyšší počet susedov pri detegovaní očí.

Po oprave sme algoritmus rozpoznávanie spustili 10x a výsledky sú zachytené v nasledujúcej tabuľke.

Výsledky										
Číslo testu	1	2	3	4	5	6	7	8	9	10
Výsledok	90%	90%	100%	100%	100%	100%	90%	100%	90%	90%

- (a) **Test 1:** V testovacej množine boli obrázky: ['rr7.jpg', 'ss1.jpg', 'as7.jpg', 'mb1.jpg', 'ke7.jpg', 'jk4.jpg', 'tk10.jpg', 'ie1.jpg', 'fe6.jpg', 'jea7.jpg']. Zle klasifikovalo 'mb1.jpg' do triedy 8 (ie).
- (b) **Test 2:** V testovacej množine boli obrázky: ['rr7.jpg', 'ss6.jpg', 'as4.jpg', 'mb2.jpg', 'ke3.jpg', 'jk8.jpg', 'tk10.jpg', 'ie8.jpg', 'fe3.jpg', 'jea4.jpg']. Zle klasifikovalo 'fe3.jpg' do triedy 7 (tk).
- (c) **Test 3:** V testovacej množine boli obrázky: ['rr5.jpg', 'ss8.jpg', 'as5.jpg', 'mb3.jpg', 'ke8.jpg', 'jk8.jpg', 'tk2.jpg', 'ie8.jpg', 'fe4.jpg', 'jea5.jpg'].

- (d) **Test 4:** V testovacej množine boli obrázky: ['rr5.jpg', 'ss6.jpg', 'as5.jpg', 'mb4.jpg', 'ke1.jpg', 'jk7.jpg', 'tk3.jpg', 'ie3.jpg', 'fe11.jpg', 'jea3.jpg'].
- (e) **Test 5:** V testovacej množine boli obrázky: ['rr5.jpg', 'ss6.jpg', 'as6.jpg', 'mb9.jpg', 'ke3.jpg', 'jk8.jpg', 'tk8.jpg', 'ie2.jpg', 'fe7.jpg', 'jea6.jpg'].
- (f) **Test 6:** V testovacej množine boli obrázky: ['rr10.jpg', 'ss5.jpg', 'as4.jpg', 'mb5.jpg', 'ke5.jpg', 'jk4.jpg', 'tk7.jpg', 'ie4.jpg', 'fe6.jpg', 'jea2.jpg'].
- (g) **Test 7:** V testovacej množine boli obrázky: ['rr9.jpg', 'ss9.jpg', 'as8.jpg', 'mb9.jpg', 'ke4.jpg', 'jk7.jpg', 'tk12.jpg', 'ie7.jpg', 'fe9.jpg', 'jea6.jpg']. Zle klasifikovalo 'as8.jpg' do triedy 2 (ss).
- (h) **Test 8:** V testovacej množine boli obrázky: ['rr6.jpg', 'ss4.jpg', 'as1.jpg', 'mb4.jpg', 'ke1.jpg', 'jk7.jpg', 'tk2.jpg', 'ie6.jpg', 'fe4.jpg', 'jea9.jpg'].
- (i) **Test 9:** V testovacej množine boli obrázky: ['rr3.jpg', 'ss7.jpg', 'as1.jpg', 'mb2.jpg', 'ke2.jpg', 'jk6.jpg', 'tk10.jpg', 'ie9.jpg', 'fe7.jpg', 'jea3.jpg']. Zle klasifikovalo 'ke2.jpg' do triedy 7 (tk).
- (j) **Test 10:** V testovacej množine boli obrázky: ['rr6.jpg', 'ss9.jpg', 'as4.jpg', 'mb5.jpg', 'ke7.jpg', 'jk9.jpg', 'tk11.jpg', 'ie3.jpg', 'fe9.jpg', 'jea2.jpg']. Zle klasifikovalo 'jk9.jpg' do triedy 5 (ke).

Priemerný výsledok testu je teda 95%. Ku zlej klasifikácii došlo hlavne v prípadoch reálnych snímok (iba v jednom prípade (7) došlo ku chybe pri snímkach celebrit). Ku chybe došlo pravdepodobne z dôvodu veľkej podobnosti testovacej snímky so snímkou z inej klasifikačnej triedy a zároveň chýbajúcou podobnou tréningovou snímkou vo svojej vlastnej klasifikačnej triede. Za ďalšie dôvody môžeme považovať dopad svetla / tieňa na tvár v testovacej snímke ako aj výraz tváre.

4.2 Test rozpoznávania reálnych snímok

Využívame rovnaký algoritmus na validáciu z obrázku 4.1. Tento krát ho spúšťame 6 krát pretože máme iba 6 tried. Výsledky sú nasledovné:

Výsledky						
Číslo testu	1	2	3	4	5	6
Výsledok	100%	100%	83,3%	100%	100%	83,3%

- (a) **Test 1:** V testovacej množine boli obrázky: ['mb4.jpg', 'ke5.jpg', 'jk1.jpg', 'tk7.jpg', 'ie5.jpg', 'fe6.jpg'].

- (b) **Test 2:** V testovacej množine boli obrázky: ['mb9.jpg', 'ke1.jpg', 'jk3.jpg', 'tk3.jpg', 'ie7.jpg', 'fe7.jpg']. .
- (c) **Test 3:** V testovacej množine boli obrázky: ['mb2.jpg', 'ke2.jpg', 'jk5.jpg', 'tk5.jpg', 'ie1.jpg', 'fe5.jpg']. Zle klasifikovalo 'ke2.jpg' do triedy 4 (tk).
- (d) **Test 4:** V testovacej množine boli obrázky: ['mb2.jpg', 'ke4.jpg', 'jk2.jpg', 'tk5.jpg', 'ie1.jpg', 'fe5.jpg'].
- (e) **Test 5:** V testovacej množine boli obrázky: ['mb5.jpg', 'ke1.jpg', 'jk3.jpg', 'tk4.jpg', 'ie1.jpg', 'fe6.jpg'].
- (f) **Test 6:** V testovacej množine boli obrázky: ['mb8.jpg', 'ke1.jpg', 'jk9.jpg', 'tk11.jpg', 'ie9.jpg', 'fe12.jpg']. Zle klasifikovalo 'jk9.jpg' do triedy 2 (ke).

Priemerný výsledok testu je teda 94, 44%. Snímky, ktoré boli v tomto prípade zle klasifikované, boli rovnako zle klasifikované aj v predchádzajúcom prípade 4.1.

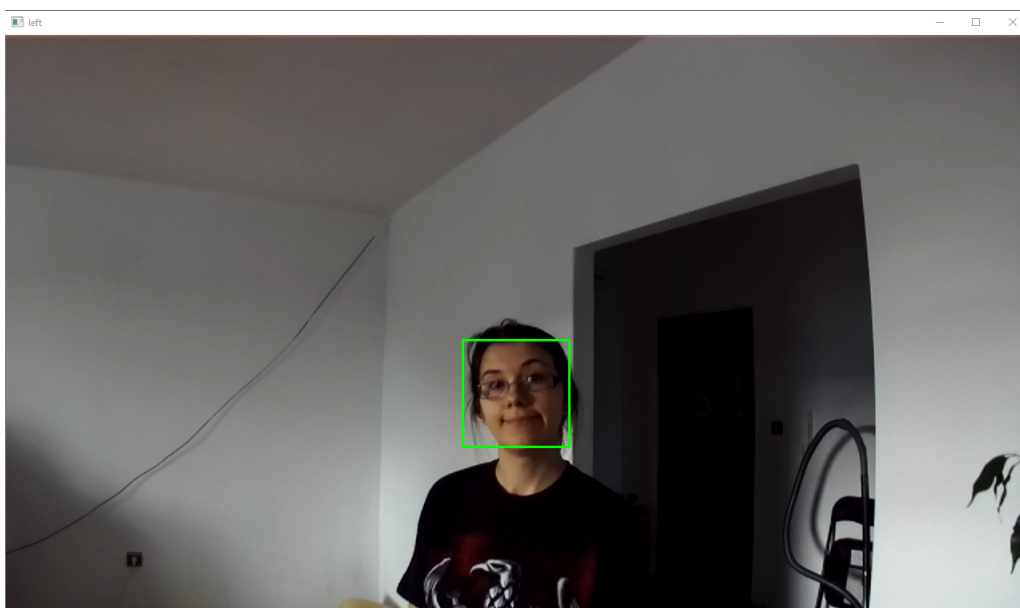
4.3 Test rozpoznávania naživo

V tomto teste s testovali funkčnosť nášho celého algoritmu, teda nie iba rozpoznávacej časti. Testovali sme teda kompletnú funkčnosť algoritmu.

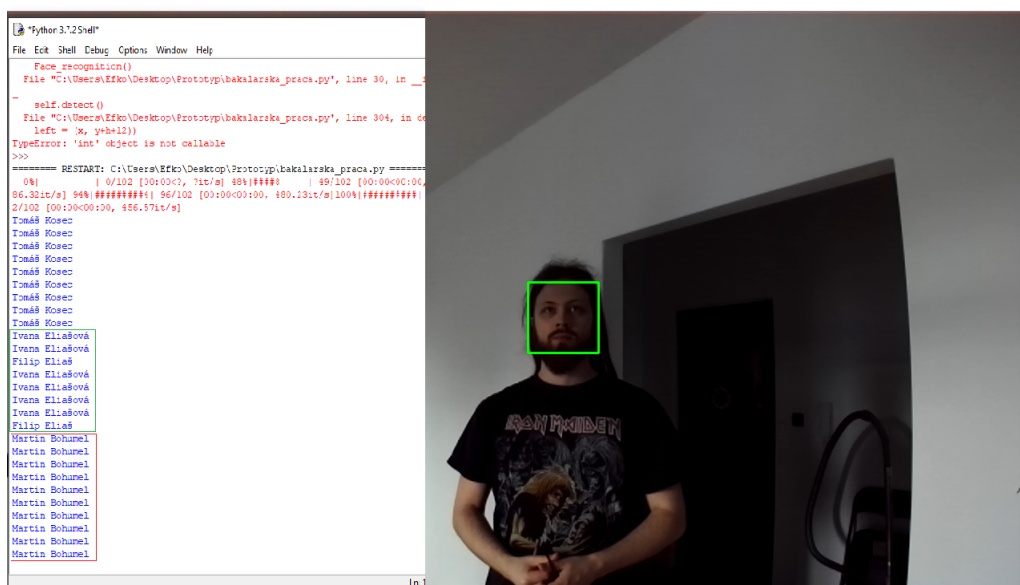
Ludia prechádzali popri kamere a tá sa ich pokúsila rozpoznať. Ak sa tvár osoby nepodarilo zaznamenať a prešiel určitý čas, algoritmus začal hlásiť detekciu neznámej osoby. Takto sme teda otestovali funkčnosť detekcie kože, pohyb a kombináciu týchto dvoch detekcií.

Následne sme testovali samotné prípady rozpoznávania tváre. Jednalo sa o dva prípady kedy sa osoby pokúšali vytvoriť ideálny záber, teda taký obraz v ktorom sa podarí detegovať obidve oči a bude tak môcť prebehnúť zarovnanie tváre. Tieto prípady môžeme vidieť na obrázkoch 4.2, 4.3. Jedná sa len o ukážkové screenshoty, ktoré sme vytvorili počas tohto testovania. V iných prípadoch, kde sa nepodarilo detegovať obe oči, k zarovnaniu snímky nedošlo.

Algoritmus rozpoznával tváre vcelku dobre, ale konkrétne pri iv.jpg nastali 2 chyby, kedy ju zle klasifikovalo. Klasifikovalo ju do triedy, ktorá má značnú podobnosť s autorovou vlastnou klasifikačnou triedou a teda sme predpokladali, že sa môže pri klasifikácii pomýliť. Celkovo teda pri tomto testovaní algoritmus rozpoznával s priemernou úspešnosťou 92, 5%.



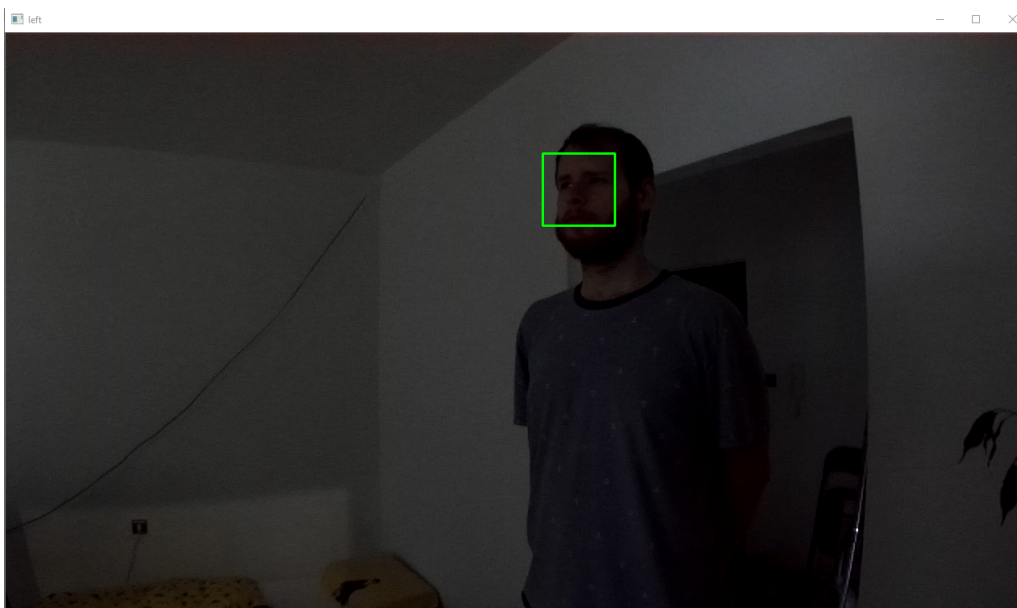
Obr. 4.2: V rozpoznávaní tu nastali dve chyby, kedy zle klasifikovalo tvár.



Obr. 4.3: Rozpoznávanie tejto snímky prebehlo bez problémov (červený rámček). Môžeme však vidieť chyby, ktorých sa dopustil algoritmus pri snímke 4.2 (zelený rámček)

Testovali sme ešte prípad, kedy sa nepodarí zaznamenať ideálnu snímku

(vhodnú na zarovnanie), ale algoritmus dokáže detegovať tvár aspoň raz. Algoritmus v tomto prípade fungoval, ale zároveň aj pochybil. Úspešnú klasifikáciu sme v týchto prípadoch dostali v 66,6%. Tu je pravdepodobným dôvodom pochybenia nedostatočná kvalita záberu.



Obr. 4.4: Záber neideálnej snímky, ktorá nie je vhodná na zarovnávanie.

4.4 Porovnávanie s `face_recognition`

Keďže nemáme veľa možností ako sa porovnávať s inými prácami budeme sa porovnávať s inými implementáciami klasifikátorov, ktoré si môžeme sami spustiť na našej databáze. Dôvod prečo sa nemôžeme porovnávať s inými prácami je v tom, že algoritmy neprebehli na rovnakých datasetoch, teda nemôžeme ich percentuálnu úspešnosť považovať za porovnateľnú s našimi výsledkami.

Využijeme teda na porovnanie knižnicu `face_recognition` [18]. Implementácia tejto knižnice je na pár riadkov, teda si ju vieme jednoducho zreplikovať. Jej úspešnosť budeme porovnávať s úspešnosťou nášho algoritmu.

Porovnávanie sme viedli spôsobom, kde sme zisťovali, či knižnica `face_recognition` spraví rovnaké chyby ako náš test 4.1. Knižnica `face_recognition` rozpozná všetky tváre správne, ale má však iný problém. Konkrétne sa jedná o problém, kde nerozpozná niektoré obrázky z našej databázy, teda v nich nedokáže nájsť tvár a teda zlyháva pri výpočte príznakov.

Takže sa jedná o lepšie rozpoznávanie avšak o horšiu detekciu tváre, predpokladom teda je, že face_recognition potrebuje kontrolované podmienky.

Jedna zo snímok, ktorú nedokázal face_recognition rozpoznať je snímka na obrázku 4.4.



Obr. 4.5: V tomto prípade algoritmus skončil errorom, teda nedokázal pravdepodobne detegovať tvár.

Záver

V tejto práci sme sa venovali preskúmaniu možností automatickej vizuálnej kontroly vstupu do laboratória FTL v pavilóne informatiky a ich praktickému overeniu. V úvodnej kapitole sme preskúmali možnosti riešenia problematiky rozpoznávania tvárí. Priblížili sme si postupy, akými sa rieši táto problematika a jednotlivé jej podčasti.

Všeobecné postupy sme však museli upraviť pre naše konkrétne podmienky. Na to sme využili algoritmy na predspracovanie obrazu, ktoré nám pomohli docieľiť potrebnú funkčnosť. Zároveň sa nám zmenilo aj prostredie, v ktorom sme mali pracovať. Vzhľadom na celosvetovú pandémiu, sme nemali prístup do laboratória a preto sme museli náš postup prispôbiť tejto novovzniknutej situácii.

Hlavnou zmenou, ktorá nastala, bola zmena databázy snímok, na ktorej sme plánovali trénovať a testovať náš algoritmus. Pomocou reálne dostupných osôb a fotografií celebrit, a úpravy získaných záberov, sme dosiahli požadované minimum snímok na prácu.

Implementovali sme získavanie obrazu pomocou kamery ZED Mini, kde sme dokázali úspešne detegovať pohyb, následne detegovať kožu a rozhodnúť tak či sa jedná o človeka. Prípade, že sa jedná o človeka, algoritmus pracuje so snímkami tváre, ktoré nasnímal a pokúsi sa v nich nájsť ideálnu na zarovnanie a následnú klasifikáciu. Rovnako algoritmus môže pri veľmi dlhom hľadaní tejto ideálnej (alebo pri zistení, že takúto snímku ani nemá) zvoliť neideálnu snímku nasnímanej tváre a tú bude rozpoznávať.

Ak nastane prípad, že algoritmus zaznamená pohyb, jedná sa o človeka, ale algoritmus nezaznamenal žiadnu tvár, algoritmus úspešne vyhlási po prejdenej 15 sekundách neznámu osobu.

Na statických snímkach dokázal náš algoritmus na rozpoznávanie tváre, rozpoznať tvár s úspešnosťou vyššou ako 94%.

Počas vývoja sme však narazili na problém pri klasifikácii pomocou Support Vector Machine. Zistili sme, že tento lineárny klasifikátor nedokáže klasifikovať objekt ako neznámu triedu. Klasifikátor vždy priradí tvár niektorej z tried. Teda sa jedná o celkom závažný problém pri našej práci, keďže podsta-

tou bolo rozpoznávať známych ľudí od neznámych. Tento problém pri SVM sa ešte stále rieši a zatiaľ len prebiehajú experimenty, ako ho vyriešiť. Vzhľadom však na našu podmienku voľby klasifikátora (to jest funkčnosť na malých dátach) sme nemali veľa možností pri výbere.

Do budúcnosti by sa dalo nadviazať na teoretický základ tejto práce a vybrať sa buď smerom iného klasifikátora, teda takého, ktorý dokáže pracovať s neznámou triedou, alebo celkovo prehodnotiť prístup a vybrať sa cestou neurónových sietí.

Literatúra

- [1] Omkar M. Parkhi, Andrea Vedaldi, Andrew Zisserman *Deep Face Recognition*, 2015
- [2] Basic Neural Network Structure, <https://towardsdatascience.com/designing-your-neural-networks-a5e4617027ed>, Towards data science
- [3] O. Déniz, G. Bueno, J. Salido, F. De la Torre, *Face recognition using Histograms of Oriented Gradients*, ScienceDirect, 2009
- [4] Elena Šikudová, Zuzana Černeková, Wanda Benešová, Zuzana Haladová a Júlia Kučerová, *Počítačové videnie detekcia a rozpoznávanie objektov*, Wikina, 2013
- [5] Cong Geng, Xudong Jiang: *SIFT features for face recognition*, 2nd IEEE International Conference on Computer Science and Information Technology, pp. 598-602, 2009.
- [6] Carro R.C., Larios JM.A., Huerta E.B., Caporal R.M., Cruz F.R. (2015) *Face Recognition Using SURF*. In: Huang DS., Bevilacqua V., Premaratne P. (eds) *Intelligent Computing Theories and Methodologies*. ICIC 2015. Lecture Notes in Computer Science, vol 9225. Springer, Cham
- [7] John Hilvert: *Real time crowd scans still a challenge for Face Recognition tech*, Australian Defence Magazine, October 2017, Yaffa Media.
- [8] Divyarajsinh N. Parmar, Brijesh B. Mehta: *Face Recognition Methods & Applications*, International Journal of Computer Applications in Technology , Január 2014
- [9] Remaining Useful Lifetime Estimation for Power MOSFETs under Thermal Stress with RANSAC Outlier Removal https://www.researchgate.net/figure/Illustration-of-the-threshold-value-determined-by-RANSAC-algorithm-to-detect-outliers_fig3_313472923.

- [10] Body language: Head and neck gestures *https :
//www.psychmechanics.com/body – language
– gestures – of – head – and – neck/*.
- [11] Face alignment for face recognition in python within opencv *https :
//sefiks.com/2020/02/23/
face-alignment-for-face-recognition-in-python-within-opencv/*
- [12] Skin detection using python opencv
https : //nalinc.github.io/blog/2018/skin – detection – python – opencv/
- [13] Haar-Like Features in Face Detection With Python
https : //www.youtube.com/watch?v = RPoUdDGonWc/
- [14] Introduction to Support Vector Machines
https : //docs.opencv.org/3.4/d1/d73/tutorial _introduction _to _svm.html/
- [15] Face Detection using Haar Cascades *https : //bit.ly/2MaLLAJ*
- [16] Cascade classifier illustration *https : //bit.ly/3ew1HQv*
- [17] How to Use OpenCV with ZED in Python
https : //www.stereolabs.com/docs/opencv/python//
- [18] Face Recognition *https : //pypi.org/project/face – recognition/*
- [19] ZED Mini *https : //www.stereolabs.com/zed – mini/*
- [20] Background Subtraction *https : //bit.ly/302yPuW*