SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-5384-72737

DETEKCIA ŠPORTOVÝCH AKTIVÍT Z VIDEO SEKVENCIE

DIPLOMOVÁ PRÁCA

2018 **Bc. Marek Hrebík**

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-5384-72737

DETEKCIA ŠPORTOVÝCH AKTIVÍT Z VIDEO SEKVENCIE

DIPLOMOVÁ PRÁCA

Študijný program:	Aplikovaná informatika
Číslo študijného odboru:	2511
Názov študijného odboru:	9.2.9 Aplikovaná informatika
Školiace pracovisko:	Ústav informatiky a matematiky
Vedúci záverečnej práce:	Ing. Dominik Sopiak
Konzultant:	Ing. Jozef Gerát

Bratislava 2018 **Bc. Marek Hrebík**

SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program:	Aplikovaná informatika
Autor:	Bc. Marek Hrebík
Diplomová práca:	Detekcia športových aktivít z video sekvencie
Vedúci záverečnej práce:	Ing. Dominik Sopiak
Konzultant:	Ing. Jozef Gerát
Miesto a rok predloženia práce:	Bratislava 2018

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean et est a dui semper facilisis. Pellentesque placerat elit a nunc. Nullam tortor odio, rutrum quis, egestas ut, posuere sed, felis. Vestibulum placerat feugiat nisl. Suspendisse lacinia, odio non feugiat vestibulum, sem erat blandit metus, ac nonummy magna odio pharetra felis. Vivamus vehicula velit non metus faucibus auctor. Nam sed augue. Donec orci. Cras eget diam et dolor dapibus sollicitudin. In lacinia, tellus vitae laoreet ultrices, lectus ligula dictum dui, eget condimentum velit dui vitae ante. Nulla nonummy augue nec pede. Pellentesque ut nulla. Donec at libero. Pellentesque at nisl ac nisi fermentum viverra. Praesent odio. Phasellus tincidunt diam ut ipsum. Donec eget est. A skúška mäkčeňov a dĺžnov.

Kľúčové slová: detekcia, rozpoznávanie, Motion History Image, Histogram of Oriented Gradients, Support Vector Machine

ABSTRACT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

Study Programme:	Applied Informatics
Author:	Bc. Marek Hrebík
Master's thesis:	Detection of sport activities from video sequency
Supervisor:	Ing. Dominik Sopiak
Consultant:	Ing. Jozef Gerát
Place and year of submission:	Bratislava 2018

On the other hand, we denounce with righteous indignation and dislike men who are so beguiled and demoralized by the charms of pleasure of the moment, so blinded by desire, that they cannot foresee the pain and trouble that are bound to ensue; and equal blame belongs to those who fail in their duty through weakness of will, which is the same as saying through shrinking from toil and pain. These cases are perfectly simple and easy to distinguish. In a free hour, when our power of choice is untrammelled and when nothing prevents our being able to do what we like best, every pleasure is to be welcomed and every pain avoided. But in certain circumstances and owing to the claims of duty or the obligations of business it will frequently occur that pleasures have to be repudiated and annoyances accepted. The wise man therefore always holds in these matters to this principle of selection: he rejects pleasures to secure other greater pleasures, or else he endures pains to avoid worse pains.

Keywords: detection, recognition, Motion History Image, Histogram of Oriented Gradients, Support Vector Machine

Poďakovanie

I would like to express a gratitude to my thesis supervisor.

Obsah

Zoznam obrázkov a tabuliek

Zoznam skratiek

Zoznam algoritmov

Zoznam výpisov

Úvod

Tu bude krasny uvod s diakritikou atd.
A mozno aj viac riadkovy uvod.

1 Použité technológie a prostredie

1.1 OpenCV

Open Source Computer Vision Library (OpenCV) bola pôvodne vydávaná firmou Intel, neskôr výskumným centrom Willow Garage, ktoré vyvíja open source softvér pre aplikácie na poli robotiky. Podporuje frameworky TesorFlow, Torch/PyTorch, Caffe. Knižnica je vydávaná pod BSD licenciou, čo umožňuje jej komerčné používanie. Táto knižnica obsahuje množstvo nástrojov a algoritmov na rozpoznávanie gest, identifikáciu objektov, segmentáciu a rozpoznávanie, spájanie obrazov, sledovanie pohybu, taktiež je vhodná pre implementáciu rozšírenej reality a strojového učenia. OpenCV je podporované v jazykoch C++, Python, Java, C a Matlab. Vývoj je umožnený na operačných systémov Windows, Android, Linux a Mac. [1]

1.2 FFmpeg

Táto knižnica je voľný softvorévý projekt určený na manipulovanie (kódovanie, dekódovanie, multiplexovanie, prehrávanie a streamovanie) s multimediálnymi súbormi. Má voľnú licenciu GNU (General Public License). Výhodou práce s knižnicou FFmpeg je jej rýchlosť a kvalita výstupných dát. Knižnicu využíva aj OpenCV pre spracovanie a distribuovanie obrazovej sekvencie. Obsahuje nástroje na zmenu kvality videa, strihanie, čo napomáha pri kontextovej analýze obrazu.[2]

1.3 Python

Programovací jazyk Python je interpretovaný vysokoúrovňový jazyk, vytvorený v roku 1991. Jeho využitie je široké, od vývoja webových aplikácii, backendu, frontendu cez vedecké a numerické výpočty, tvorbu grafických rozhraní, vývoj softvéru a výučbu a tvorbu biznis aplikácií. V súčasnosti má širokú podporu u programátorov, čo sa odráža aj na prehľadnom zdokumentovaní. Je to open source softvér s množstvom štandardných knižníc, ktoré majú široké využitie v automatizácii, multimédiách, spracovaní obrazu, textu a ďalších sférach IT.[3]

1.4 Scikit a Numpy

1.4.1 Scikit

Scikit-learn je knižnica s otvoreným zdrojovým kódom pre strojové učenie pre programovací jazyk Python. Scikit je napísaný v jazyku Python s niektorými algoritmami písanými v Cythone pre zvýšenie výkonu tejto knižnice. Obsahuje rôzne klasifikačné, regresné a zhlukovacie algoritmy, taktiež na modelovanie dát (krížová validácia, výber príznakov,

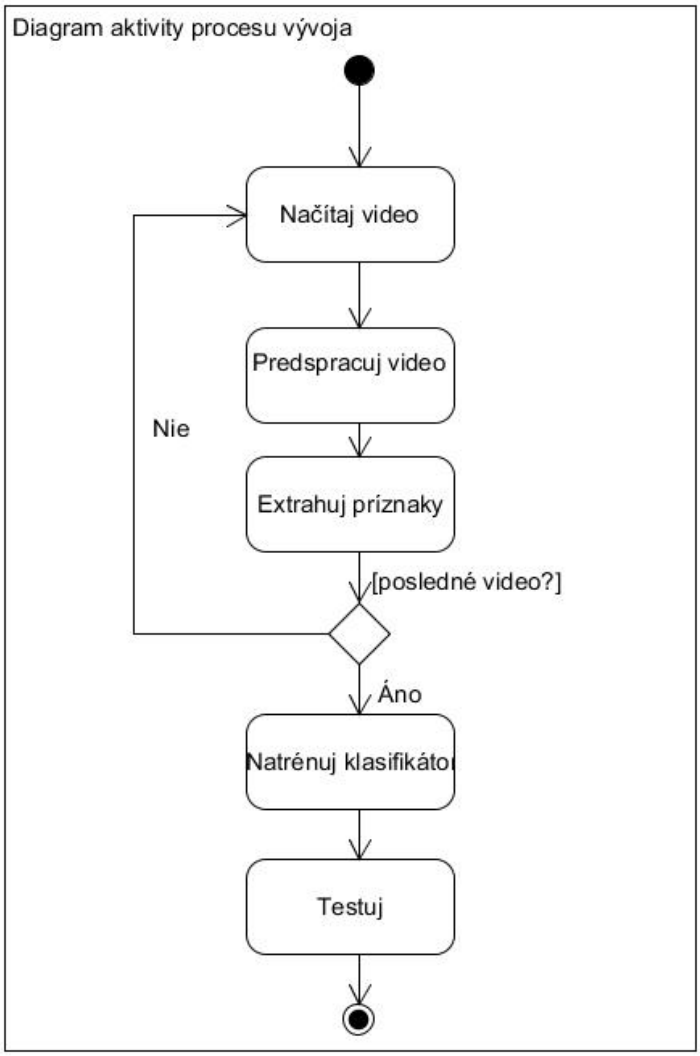
clustering a iné. Je navrhnutá na spoluprácu s numerickou knižnicou NumPy a vedeckou knižnicou SciPy. [4]

1.4.2 NumPy

NumPy je rozšírenie do Pythonu, ktoré nám umožňuje rýchle a efektívne vykonávanie operácií nad poľami homogénnych dát. NumPy pole je multidimenzionálne pole objektov ktoré majú všetky rovnaký typ. V pamäti sa nachádza ako objekt, ktorý smeruje na blok pamäte, ktorý drží informáciu o type údajov v pamäti, počet dimenzií pamäte, veľkosť každej dimenzie a taktiež aj vzdialenosti medzi jednotlivými prvkami pozdĺž každej z osí. Narozdiel od klasického zoznamu, ktorý je implementovaný ako zoznam smerníkov je prehľadávanie NumPy poľa rýchlejšie. klasický zoznam potrebuje na iteráciu poľa prístup cez dva smerníky, čo predĺžuje čas prehľadávania.[5]

2 Princípy riešenia

V prvej časti tejto práce si predstavíme a zároveň porovnáme metódy, ktoré sme zvažovali pri riešení nášho problému. Takisto si zhrnieme aké databázy pohybov máme k dispozícii pre potreby tejto práce, typy databáz, dĺžky jednotlivých videí, vlastnosti a aké druhy aktivít sa v týchto videách nachádzajú. Taktiež je potrebné predstaviť metodológiu spracovania údajov pre zachytenie vhodných vlastností videa na rozpoznanie. Na obrázku 1 je znázornená predstava riešenia problému detekcie a rozpoznávania pohybu. Základom je načítanie videí, následne ich predspracovanie na extrakciu príznakov a samotná extrakcia. Konečným krokom v procese by malo byť testovanie a vyhodnotenie aplikácie.



Obr. 1: Diagram plánu vývoja riešenia

2.1 Spôsob spracovania sekvencie

Zo zadania vyplýva, že pre získanie príznakov z videa budeme potrebovať videosekvenciu rozdeliť na menšie časti, respektíve ju spracovať do formátu obrázku (.jpg, .png. alebo iné). Jednotlivé snímky z videa sú potrebné na hlbšiu analýzu. Podľa zvoleného datasetu budeme môcť zvoliť či budeme používať celú videosekvenciu, alebo iba jej časť, v ktorej je zrejmý druh pohybu.

V jednotlivých triedach videí bude potrebné hľadať spoločné a rozdielne znaky na to, aby sme následne vybrali vhodný postup na spracovanie z získanie vhdných príznakov. Knižnica OpenCV neobsahuje metódy ktoré zabezpečia čítanie a prenos obrazu, avšak jej metódy využívajú funkcionalitu ďalšej knižnice FFmpeg, ktorá distribuuje prostredníctvom rozhrania jednotlivé obrazové snímky.

2.1.1 Získanie časovo-priestorových vzťahov

Bude potrebné získať tieto vzťahy na porovnávanie s ďalšími videami, teda určenie vhodného spôsobu na získanie týchto vzťahov. Jedným zo spôsobov je napríklad algoritmus MHI (Motion History Image), ktorý nám môže zabezpečiť tieto príznaky z videa uložiť a ďalej spracovávať. Podľa tohto algoritmu by sme vedeli určiť odkiaľ kam sa aktér z videa pohyboval, akým spôsobom a rýchlosťou, čo môžme považovať za prvé vhodné príznaky pre určenie typu pohybu.[6] Tento algoritmus si bližšie popíšeme ďalej v tejto práci.

2.1.2 Extrakcia príznakov

Extrakcia príznakov je dôležitou časťou pre natrénovanie klasifikátora. Preto je potrebné vybrať príznaky, ktoré nám s čo najvyššou presnosťou zadefinujú konkrétny pohyb, ktorý sa bude vykonávať vo videu. Mali by to byť príznaky, ktoré sú jednoznačné a jedinečné pre každý z druhov pohybu. Preto musíme uvažovať vhodnú metódu extrahovania príznakov, aby sme zaistili diverzitu medzi triedami jednotlivých typov pohybu. K tomu nám bude nápomocná aj kombinácia viacerých spôsobov extrakcie.

Tieto príznaky budeme extrahovať z predspracovaného zdroja a ukladať na disk pre natrénovanie klasifikátora, prípadne viacnásobné trénovanie a testovanie bude vhodné jednotlivé výsledky ukladať pre neskoršie porovnanie.

2.1.3 Návrh klasifikátora

Po úspešnom získaní príznakov z každej videosekvencie bude potrebné natrénovať klasifikátor tak, aby nám každá skupina spracovaných videozáznamov spadala do jednej triedy. Takto natrénovaný klasifikátor budeme potrebovať uložiť pre potreby testovania a porovnávania výsledkov a diverzity jednotlivých skupín. Zároveň budeme potrebovať

zvoliť vhodný pomer dát na testovanie a tréning na základe Databáz, ktoré budeme mať k dispozícii. Z tohto dôvodu budeme potrebovať čo najviac videí, aby bol návrh klasifikačnej metódy čo najpresnejší.

2.2 Databázy videí

Na internete je veľké množstvo video databáz s rôznymi pohybmi objektov, ľudí a zvierat. Výber databázy realizujeme na základe vlastností jednotlivých videí. Videá môžu byť snímané staticky z jedného miesta bez pohybu kamery, staticky s otáčaním kamery, taktiež dynamicky s rôznym pohybom a otáčaním kamery. Výber databázy bude najdôležitejšou súčasťou tejto práce, keďže od nej sa bude odvíjať celý ďalší postup.

2.2.1 Vlastnosti videa

Dôležitými vlastnosťami videí z databáz sú:

- **veľkosť**
- **rozlíšenie**
- **dĺžka**
- **počet videí**
- **spôsob snímania**
- **formát videa**
- **počet objektov**

Veľkosť videa ovplyvňuje vo vysokej miere časový úsek určený na spracovanie videa. Z tohto dôvodu je vhodné hľadať databázy videí, ktorých videá majú menšiu veľkosť, čo priamo súvisí s ich rozlíšením, počtom snímkov za sekundu a dĺžkou. Nesmieme ale zabudnúť na to, že video musí byť zreteľné a pohyb detekovateľný.

Rozlíšenie videa je dôležitým faktorom pre spracovnie z hľadiska kvality príznakov a snímkov. Zároveň jeho odporúčaná veľkosť sa líši od konkrétneho spôsobu extrakcie príznakov. Dôležité je, aby obraz konkrétnej aktivity na videu bol jasný a voľným okom rozpoznateľný.

Dĺžka sekvencie sa líši od rozmanitosti jednotlivých pohybov osoby. Niektoré databázy obsahujú videosekvencie s opakovaním pohybov, teda dĺžka videí v týchto databázach je väčšia ako v tam, kde je daná aktivita alebo pohyb zachovaný bez opakovania.

Počet videí môže ovplyvniť výsledok a efektivitu rozpoznávania aktivít. Niektoré databázy obsahujú malé množstvo videí pre jeden konkrétny pohyb (okolo 10 až 20), iné zdroje videosekvencií ich majú aj viac ako 50 pre jeden typ pohybu.

Spôsob snímania v rozpoznávaní pohybu je dôležitou vlastnosťou, keďže nie všetky spôsoby riešenia je možné aplikovať na videá s pohybujúcou sa kamerou a opačne.

Formát videa súvisí s jeho veľkosťou, avšak formát je možné meniť a konvertovať na nami potrebný rozmer.

Počet objektov, ktoré sa na videu pohybujú a vyhodnocujú vplýva na výkon a rýchlosť rozpoznávania. Tu taktiež nie je možné aplikovať niektoré metódy rozpoznávania.

2.2.2 Databázy ľudského pohybu

Pre potrebu rozpoznávania pohybu osoby existuje viacero databáz, ktoré sme v našej práci uvažovali a testovali:

- **HMDB51**
- **KTH**
- **UCF-Sports**
- **Hollywood**
- **MSR Action I, II**
- **IXMAS**
- **WEIZMANN**

HMDB51 je databáza pohybov v ktorej sa nachádza 51 tried pohybu a gestikulácie ako napríklad česanie vlasov, žuvanie, lezenie, potápanie, chôdza atď. Celkovo obsahuje 6849 videí s rozlíšením 320 x 240 pixelov. Videá sú so statickou aj dynamickou kamerou. Zdrojom videí je YouTube ako aj iné verejne dostupné databázy videí. Videá sú špecifické vysokou diverzitou kvality ako aj pohybu.[7]

KTH pohybová databáza so šiestimi triedami - chôdza, pomalý beh, beh, boxovanie, kývanie rukou, tlieskanie. V každej z týchto tried sa nachádza 100 čiernobielych videí s rozlíšením 160 x 120 pixelov so statickou kamerou. Videá sú natočené v interiéri ako aj exterieri. [7]

UCF-Sports obsahuje 9 tried pohybu so 182 videami so statickou aj dynamickou kamerou. Rozlíšenie videí je 720 x 480 pixelov. Videá sú z rôznych športových podujatí vysielaných v televízii - skok do vody, vzpieranie, jazda na koni a iné.[7]

Hollywood databáza obsahuje 8 tried - vystúpenie z vozidla, bozkávanie, postavenie sa, sadnutie si a iné. Nachádza sa tam 430 videí s rozlíšením 300 - 400 x 200 - 300 pixelov s dynamickou kamerou a dynamickým pozadím.[7]

MSR Action I, II obsahujú 3 triedy pohybu so statickou kamerou a 16-timi videami s rozlíšením 320 x 240 pixelov. Namodelované videá sú určené na detekciu pohybu, nie jeho rozpoznávanie. Vo videách sa nachádza viac ako jeden pohyb. V pozadí sa objavujú aj iné osoby, prípadne objekty ako auto. [7]

IXMAS alebo *INRIA Xmas Motion Acquisition Sequences* obsahuje 13 tried s rozlíšením 390 x 291. Pohyb je zaznamenávaný z viacerých statických kamier. [7]

WEIZMANN obsahuje 10 tried pohybu, celkový počet videí je 90. Obsahujú triedy pohybu ako chôdza, beh, kývanie jednou alebo dvomi rukami, skok na mieste a iné. Videá sú natočené so statickou kamerou a pozadím v interiéri s rozlíšením 180 x 144. [7]

2.3 Možnosti spracovania obrazu

Na získanie príznakov a potrebu klasifikácie pohybu analyzujeme a zvažujeme konkrétne spôsoby spracovania obrazu z videa. Pre tento účel berieme v úvahu algoritmus Motion history image (MHI) na extrakciu prvotných príznakov a následné spracovanie pomocou Histogram of oriented gradients (HOG).

Pre zosilnenie našich príznakov a zaručenie vyššej úspešnosti klasifikácie uvažujeme použitie Principal Component Analysis (PCA), ktoré taktiež urýchli trénovanie klasifikátora. Takto získané príznaky by sme následne vedeli natrénovať pomocou Support vector machine (SVM). Z natrénovanej SVM vieme otestovať úspešnosť klasifikátora, ktorý sme získali a tým sa dostať ku výsledkom a porovnávaniam.

2.3.1 Motion History Image

MHI je metódou, ktorá je založená na časovej šablóne. Je to jednoduchá metóda avšak robustná, vhodná na reprezentovanie a analýzu pohybu.[6] Táto metóda bola prvýkrát popísaná v dokumente “An appearance-based representation of action” od Aarona Bobicka a Jamesa W. Davisa.[8] Rozpoznávanie akcie pomocou MHI patrí do skupiny prispôsobovania šablóny. V MHI sa informácia pohybu v čase spája do jedného obrázku, kde intenzita pixelov je funkciou histórie pohybu na danej pozícii pixelu. MHI sa vypočítava pomocou aktualizačnej funkcie.2

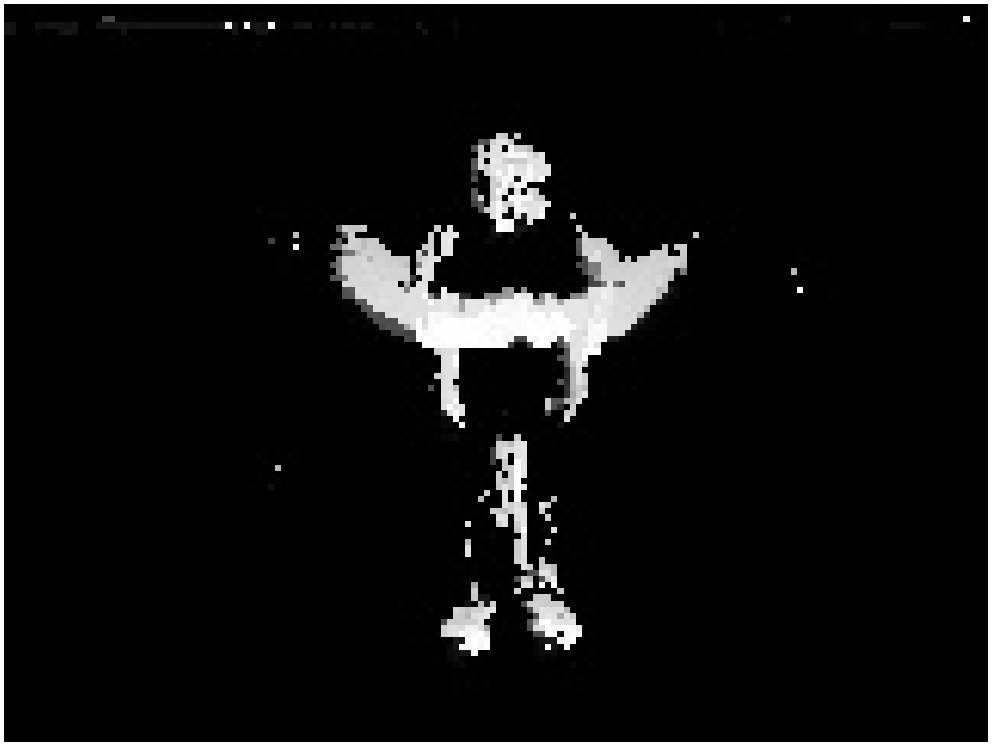
$$H_\tau(x,y,t) = \begin{cases} \tau & ak\ \Psi(x,y,t) = 1 \\ \max(0, H_\tau(x,y,t-1) - \delta & inak \end{cases}$$

$(x,y,t) - pozícia\ pixelu\ a\ čas$
$\Psi(x,y,t) - prítomnosť\ (alebo\ pohyb)\ objektu\ v\ poslednom\ snímku$
$\tau - dĺžka\ určujúca\ časový\ rozsah\ pohybu\ (jednotlivých\ snímkov)$
$\delta\ - parameter\ stmavnutia\ pixelu$

Obr. 2: Funkcia zmeny pixelov v MHI

Táto aktualizačná funkcia je volaná pre každý ďalší snímok, ktorý analyzujeme z videosekvencie. Výsledkom tohto výpočtu je skalárny snímok, v ktorom svetlejšie časti obrázku označujú najnovšie menené pixely a tmavšie sú tie, ktoré boli menené dávnejšie z pohľadu pohybu na videu.

To nám umožňuje zaznamenať pohyb z videa do jediného obrázku, v ktorom je vystihnutý celý priebeh videa alebo jeho časti. Táto technika je vhodná na použitie pri videách so statickou kamerou a pozadím.[9]



Obr. 3: Motion History image tlieskania

Na obrázku 3 je príklad spracovaného pohybu z videa do MHI, na ktorom môžme

vidieť pohyb tela pri akcii tlieskania.

2.3.2 Histogram of Oriented Gradients

HOG je metóda extrakcie príznakov z obrazu. Extrahuje príznaky z každej časti vstupného obrazu. Snaží sa zachytiť tvar štruktúry v regiónoch pomocou prechodov farieb jednotlivých pixelov. Rozdeľuje obraz na menšie spojené časti (napríklad 8 x 8 pixelov)a tie časti do ešte menších blokov z ktorých sa následne zostavuje histogram. Deskriptor je spojenie týchto histogramov.

Každá z týchto častí má rovnaký počet orientácií gradientu. Pre zvýšenie presnosti môžu byť lokálne histogramy normalizované na základe kontrastu a výpočtom miery intenzít naprieč väčšími časťami obrazu, teda bloku. Táto normalizácia vedie k zvýšenej invariantnosti pri zmene osvetlenia alebo tienenia.

HOG Deskriptor má výhody v tom, že pôsobí na lokálne bunky, je invariantný voči geometrickým a fotometrickým transformáciám. N. Dalal a B. Triggs zistili, že hrubé priestorové vzorkovanie, jemné vzorkovanie a silná lokálna fotometrická normalizácia umožňujú HOG deskriptoru vhodne detekovať ľudské bytosti v obrazoch.

Výpočet gradientu Základným krokom pri HOG algoritme je výpočet gradientových hodnôt. Najbežnejšou metódou je aplikácia 1-D centrovanej diskrétnej masky v horizontálnom alebo aj vertikálnom smere. Táto metóda vyžaduje filtrovanie farieb alebo intenzitu obraz s filtračnými jadrami.

Histogram orientovaných gradientov Ďalším krokom je vytvorenie histogramu orientovaných gradientov. Každý pixel v bunke pridáva svoju ováhovanú orientáciu pre orientovaný histogramový kanál, ktorý je vytvorený na základe výpočtu gradientov. Bunky môžu byť obdĺžnikového alebo radiálneho tvaru a histogramové kanály sú rozložené rovnomerne od 0 do 180 alebo 0 - 360 stupňov, záleží od toho, či gradienty môžu nadobúdať aj zápornú hodnotu, alebo nie.

Podľa výskumov N. Dalala a B. Triggsa bolo zistené, že najvhodnejší počet kanálov na detekovanie osoby je 9 pri rozsahu 0 - 360 stupňov.

Bloky deskriptorov Na zohľadnenie zmien v osvetlení a kontraste musí byť intenzita gradientu lokálne normalizovaná, čo vyžaduje spoločné zoskupovanie buniek do väčších priestorovo prepojených blokov. HOG Deskriptor je následne spojený vektor komponentov normalizovaných histogramov buniek zo všetkých oblastí blokov. Tieto bloky sa prekrývajú, z čoho vyplýva, že každý jedna bunka prispieva voac ako raz do vytvorenia konečného deskriptora. Existujú dva hlavné tvary blokov:

- **R-HOG** (Rectangular HOG)
- **C-HOG** (Circular HOG)

R-HOG - sú štvorcové mriežky reprezentované tromi parametrami:

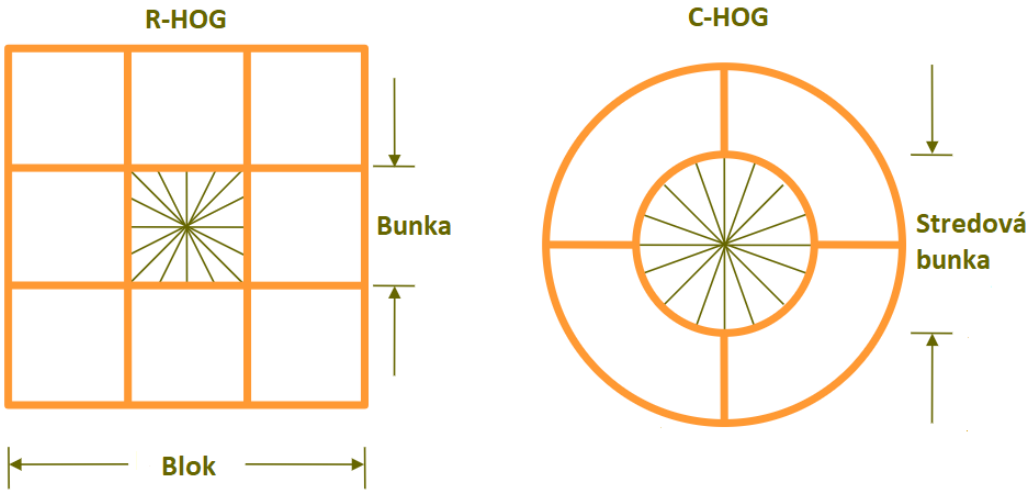
- počet buniek v bloku
- počet pixelov v bunke
- počet kanálov na histogram bunky

V práci N. Dalala a B. Triggsa je uvedené, že optimálne parametre sú štyri pixelové bunky s veľkosťou 8 x 8 na jeden blok (16 x 16 pixelov na blok)s deviatimi histogramovými kanálmi. Bolo taktiež zistené, že menšie vylepšenie vo výkonnosti môže byť nadobudnuté použitím Gaussovho priestorového okna v každom bloku pred zaznamenaním váhovaných histogramov. Lepšiemu výsledku by napomohlo to, že okrajové pixely by boli ováhované nižšími hodnotami.

C-HOG - sú kruhové bloky, ktoré je možné použiť v dvoch variantoch. Jednou variantou je využitie jednej centrálnej bunky, pri druhej variante je táto bunka uhlovo rozdelená do rovnakých častí. Tieto bloky sú popísané štyrmi parametrami:

- počet uhlových buniek
- počet radiálnych buniek
- polomer stredovej bunky
- faktor expanzie pre polomer dodatočných radiálnych buniek

Podľa zistení N. Dalala a B. Triggsa najvhodnejšie zvoleným spôsobom je vytvorenie dvoch radiálnych buniek so štyrmi uhlovými bunkami, polomerom stredovej bunky 4 pixely a faktor expanzie s hodotou 2.



Obr. 4: R-HOG a C-HOG

Na obrázku 4 môžme vidieť ako vyzerá rozdelenie obrazu podľa R-HOG a C-HOG.

Normalizácia blokov Sú štyri rôzne spôsoby normalizácie blokov, ktoré sú popísané na obrázku 5.

$$L2-norm \quad \rightarrow \quad f = \frac{v}{\sqrt{||v||_2^2+e^2}}$$

$$L2-hys \quad \rightarrow \quad L2-norm \;\; s\; hodnotou\; v < 0.2$$

$$L1-norm \quad \rightarrow \quad f = \frac{v}{(||v||_1 + e)}$$

$$L1-sqrt \quad \rightarrow \quad f = \sqrt{\frac{v}{(||v||_1 + e)}}$$

v $-$ *nenormalizovaný vektor obsahujúci histogramy daného bloku*
$||v||_k$ $-$ *je k norma* $(k = 1, 2)$
e $-$ *malá konštanta*

Obr. 5: Metódy normalizácie

Schéma ***L2-hys*** môže byť vypočítaná z ***L2-norm***, kde sa výsledok renormalizuje pre vektor $v < 0.2$. Všetky štyri normy majú výrazne lepšie výsledky ako nenormalizované dáta. Tieto údaje sú následne použiteľné ako vektor príznakov pre SVM. [10]

2.3.3 Principal Component Analysis

Analýza hlavných komponentov (angl. Principal Components Analysis) je metóda štatistiky, ktorá využíva ortogonálnu transformáciu za účelom zmeny prvkov, v ktorých sa vyskytuje korelácia na prvky, kde sa takáto lineárna korelácia nebude existovať. PCA teda spracováva a upravuje jednotlivé komponenty tak, aby sa ich dimenzia zmenšila, alebo bola nanajvýš rovnaká a zároveň sa snaží zachovať čo najväčšie množstvo informácií o pôvodných premenných. Tento prístup pomáha analyzovať dané dáta v podpriestore, čo nám umožňuje ďalšie spracovanie.

Dôležitou súčasťou v PCA sú vlastné vektory. Sú to nenulové vektory, ktorých smer sa po transformácii pomocou lineárneho operátora nemení, pojem vlastný vektor v PCA obmieňame za hlavný komponent.

V Computer Vision má táto metóda veľmi dobrý vplyv na dáta charakterizujúce rôzne špecifické tvary, objekty, keďže nám umožňuje zosilniť význam príznakov, ktoré sú pre daný problém dôležité, a tie, ktoré nie sú potrebné, dokáže eliminovať. Táto transformácia teda zvyšuje varianciu hlavných komponentov naprieč všetkými lineárnymi kombináciami vektora, čo napomáha zvyšovať úspešnosť klasifikátora pri riešení rozpoznávania na extrakciu príznakov. Túto metódu navrhol v roku 1901 matematik anglického pôvodu Karl Pearson. Jej rôzne obmenené a vylepšené verzie sa využívajú dodnes v štatistike, *Computer Vision*, robotike a ďalších smeroch.

Pomocou Jordanovej dekompozičnej vety o symetrických maticiach môžme symetrické štvorcové matice zapísať v nasledovnom tvare, ktorý je na obrázku 6 [11] [12].

$$\boldsymbol{X} = (X_1, \dots\ , X_p)^T$$

$$\boldsymbol{M} = \boldsymbol{OAO^T} = \sum_{j=1}^{p} \alpha_j o_j o_j^T$$

$\boldsymbol{X} - p - rozmerný\ náhodný\ vektor$

$\boldsymbol{p} -\ \ rozmer\ štvorcovej\ matice$

$\boldsymbol{M} - kovariančná\ matica$

$\boldsymbol{A} - diagonálna\ matica\ diag(\alpha_1, \dots, \alpha_p)\ vlastných\ čísel\ matice\ \boldsymbol{M}$

$\boldsymbol{O} - ortogonálna\ matica\ (o_1, \dots\ o_p), kde\ stĺpce\ označujú\ vlastné\ vektory\ matice\ M$

$\boldsymbol{\alpha_j} - vlastné\ číslo\ prislúchajúce\ vlastnému\ vektoru\ o_j, pre\ ktoré\ platí, že\ \alpha_1\ \geq \cdots \geq\ \alpha_p$

$$\boldsymbol{Z} =\ U^T(\boldsymbol{X} -\ \theta)$$

$\boldsymbol{\theta} - stredná\ hodnota\ vektora\ \boldsymbol{X}$

$\boldsymbol{Z} - náhodný\ vektor\ hlavných\ komponentov\ náhodného\ vektora\ \boldsymbol{X}$

Obr. 6: Definícia Jordanovej matice

Z tejto matice následne vieme vypočítať vektor hlavných komponentov, viď obrázok 7.

$$\boldsymbol{Z} =\ U^T(\boldsymbol{X} -\ \theta)$$

$\boldsymbol{\theta} - stredná\ hodnota\ vektora\ \boldsymbol{X}$

$\boldsymbol{Z} - náhodný\ vektor\ hlavných\ komponentov\ náhodného\ vektora\ \boldsymbol{X}$

Obr. 7: Výpočet hlavných komponentov

[13]

Redukciou dimenzie vieme dosiahnúť aj kratšiu dobu trénovania klasifikátora ako aj jeho vyššiu úspešnosť.

2.3.4 Support Vector Machine

SVM je metóda strojového učenia s učiteľom, je vhodná na používanie pri klasifikácii alebo regresnej analýze. Táto metóda na základe trénovacích vzoriek, ktoré sú priradené do jednej z dvoch tried vytvorí model. Tento model je následne schopný priraďovať nové vzorky do jednej z týchto tried, čo z neho robí nepravdepodobnostný binárne lineárny klasifikátor. SVM model reprezentuje vzorky ako body v priestore, tak, že mapuje jednotlivé vstupy do kategórií, ktoré sú v tomto priestore jasne oddelené. Klasifikátor priraďuje nové vzorky podľa toho, ku ktorej z tried majú bližšie v rámci tohto priestoru.

Okrem lineárnej klasifikácie je SVM schopné efektívne riešiť aj nelineárne separovateľné problémy pomocou takzvaného kernel triku. Pokiaľ dáta nie sú pri trénovaní označené, do ktorej kategórie patria, učenie s učiteľom nie je možné, avšak je možné učenie bez učiteľa tak, že algoritmus sa snaží nájsť prirodzené zoskupenie dát a tak ich rozdeliť do kategórií.

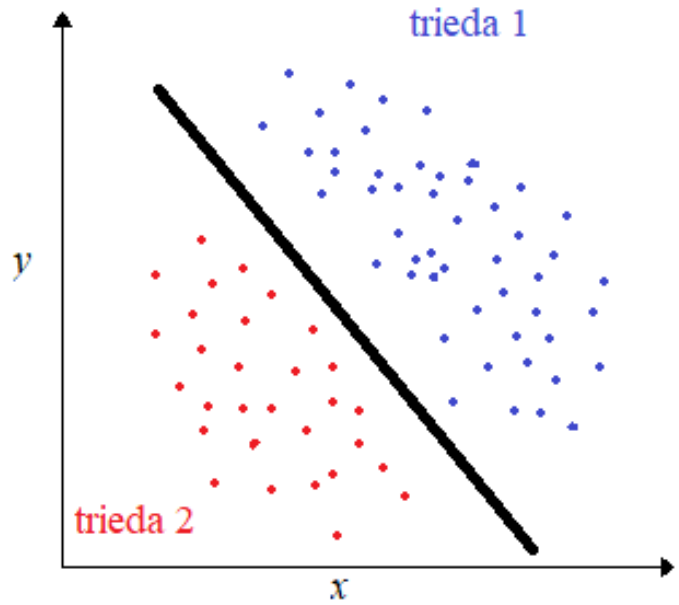
SVM algoritmus vytvorili Hava Siegelmann a Vladimir Vapnik. Tento algoritmus má široké využitie a je používaný v priemyselných aplikáciách, počítačovom videní, umelej inteligencii. [14]

Definícia algoritmu SVM vytvára hyperrovinu [1], alebo viacero hyperrovín, vo vysokorozmernom priestore, ktorý môže byť použitý na klasifikáciu, regresiu alebo iné úlohy. Dobrá separácia sa dosiahne vtedy, keď hyperrovina má veľkú vzdialenosť od najbližšieho trénovacieho bodu akejkoľvek triedy. Teda platí, čím je väčšia vzdialenosť hyperroviny od trénovacích údajov, tým je nižšia chyba klasifikátora pri určovaní triedy testovacej vzorky.[15]

Keďže nie všetky problémy je možné lineárne separovať, bolo navrhnuté, aby sa priestor, ktorý SVM spracováva rozvrhol do viacrozmerného priestoru, čo uľahčí rozdelenie jednotlivých dát do tried. Pre zanechanie výpočtovej rýchlosti je mapovanie v SVM navrhnuté v pôvodnom priestore, kde sa následne zvolí vhodná kernelová funkcia $k(x, y)$. Hyperroviny vo viacrozmernom priestore sú definované ako súbory bodov, kde ich vektory majú konštantnú dĺžku. Následne body z proestoru príznakov sú mapované do hyperroviny.

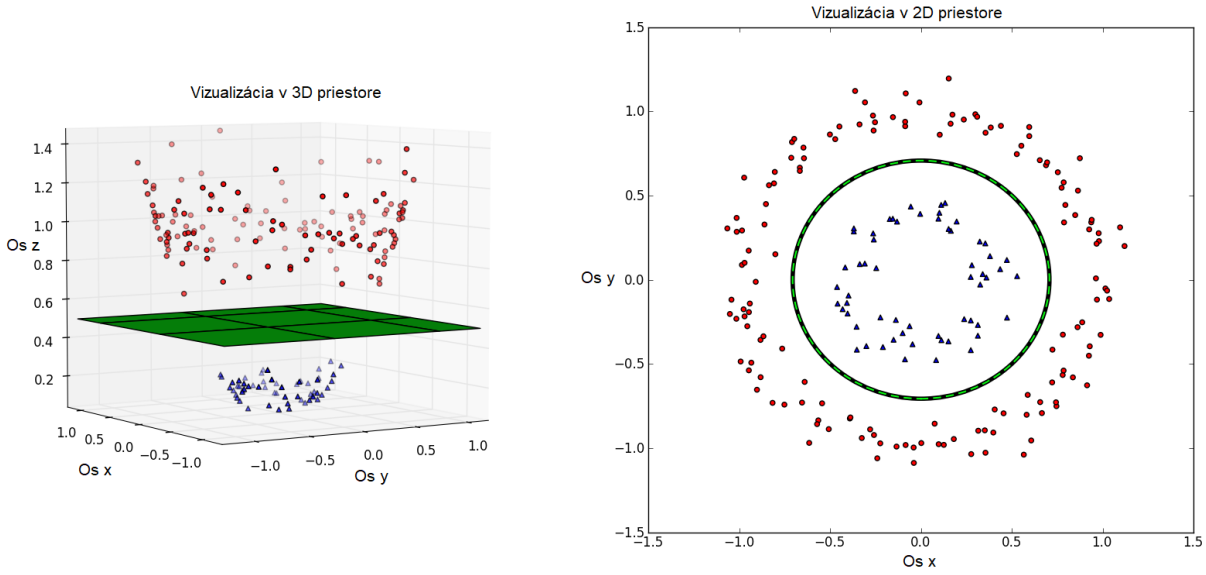
Lineárny klasifikátor Najjednoduchšou možnosťou použitia SVM je prípad lineárneho klasifikátora pre dáta, ktoré je možné lineárne rozdeliť do dvoch tried pomocou hyperroviny. Príklad takéhoto problému je na obrázku 8.

[1] Podpriestor, ktorý má o jednu dimenziu menej ako priestor, v ktorom sa nachádza.



Obr. 8: Príklad lineárne separovateľného problému

Nelineárny klasifikátor Na obrázku č. 9 môžme vidieť lineárne neseparovateľný problém dvoch množín, ktorý sa po pridaní tretej dimenzie stáva lineárne separovateľný. Takýmto spôsobom dokážeme s SVM zatriediť jednotlivé videá z množín do správnych tried. Transformácia do vyššierozmerného priestoru sa nazýva jadrový trik *(angl. kernel trick)*. Pomocou tohto triku vieme lineárne separovať aj lineárne neseparovateľné problémy.[15]



Obr. 9: Rozšírenie dimenzie v SVM

Medzi najčastejšie používané jadrové funkcie patria:

- **Polynóm stupňa p**
- $K(x, y) = (x.y + 1)^p$
- **Radiálne bázové funkcie**
- $K(x, y) = e^{(-\|x-y\|^2/2\sigma^2)}$
- **Dvojvrstvová neurónová sieť**
- $K(x, y) = \tanh(\kappa x.y - \delta)$

3 Návrh riešenia

Po oboznámení s teoretickými piliermi tejto práce prejdeme ku konkrétnemu riešeniu témy a implementácii, ktorú sa nám podarilo vytvoriť. Postupne prejdeme jednotlivé kroky vytvárania programu od analýzy datasetov, načítavania a ukladania videosekvencií do obrazkových formátov. Ďalej si ukážeme metódy a spôsoby, ktorými sme následne vzniknuté dáta spracovali pre potrebu výberu príznakov a zatriedenia do jednotlivých tried. Fungovanie programu priblížime aj časťami kódu s popisom, na čo slúži a ako pracuje. Súčasťou tejto kapitoly bude aj krátky popis zmien, ktoré sme počas návrhu tejto implementácie vykonali a samozrejme aj ich dopad na finálne riešenie.

3.1 KTH Databáza

Pre naše riešenie sme vyberali z viacerých datasetov, ktoré boli popísané v kapitole 2.2.2. Nakoniec sme sa rozhodli pre databázu KTH, v ktorej máme 6 tried pohybu a nachádza sa pri každom z nich 100 videí. Naše rozhodnutie sme učinili takto z viacerých dôvodov. Potrebovali sme dataset, v ktorom sa nachádza veľké množstvo videí, zároveň vo videách sme potrebovali statickú kameru so statickým pozadím, aby sme predišli zvýšenému šumu. V tejto databáze sú pohyby:

- Boxovanie
- Mávanie rukami
- Tlieskanie rukami
- Beh
- Pomalý beh
- Chôdza

Pri analýze snímkov sme zistili, že dĺžka videí je od ***8 do 59 sekúnd***. Čo pre dané typy pohybov je postačujúci čas na to, aby sme z nich dokázali extrahovať dôležité informácie, ktoré sa tohto pohybu týkajú. Počet snímkov za sekundu v každom z videí je 25, z tohto dôvodu sme usúdili, že aj v prípade krátkeho videa s dĺžkou 8 sekúnd budeme mať k dispozícii minimálne ***200*** snímkov. Ďalším dôležitým faktom je, že sa budeme musieť popasovať s podobnosťou pohybov behu, pomalého behu a chôdze, keďže sú veľmi podobné, a ich rozdiel je iba v rýchlosti vykonávania. Je nutné dodať, že v každom z videí sa jednotlivé typy pohybov opakovali viacnásobne, z čoho sme už vopred usudzovali, že

na spracovanie nám bude stačiť iba časť videa, čo nám ušetrí čas spracovania dát počas behu programu.

Obr. 10: Jednotlivé pohyby v datasete KTH [16]

Na obrázku 10 vidíme všetky kategórie pohybov, jednotlivé pohyby vykonáva viacero osôb rôznymi spôsobmi, teda je medzi týmito pohybmi aj diverzita, ktorá je určite vhodná na trénovanie pohybu.

3.2 Predspracovanie videa

Na predspracovanie videa a vytiahnutie prvotných príznakov sme sa rozhodli z našich videí vytvoriť obrázky, na ktorých sa bude nachádzať história pohybu. Algoritmus MHI, ktorý sme pri tom použili sme popísali v časti 2.3.1. Tento algoritmus sme zvolili na základe toho, že máme databázu videí bez pohyblivého pozadia, teda predídeme šumom na pozadí a budeme sa môcť venovať iba pixelom, ktoré menia svoju pozíciu v rámci videa. Presne to je postačujúca podmienka na to, aby sme bez väčších problémov mohli implementovať funkcionalitu algoritmu MHI na nami zvolený dátový rozsah. Vytvoriť MHI sme sa rozhodli zo všetkých videí, aby sme následne videli, akým spôsobom prebehlo spracovanie, či je možné z MHI vyčítať pohyb a jeho základné znaky.

```
def getImage(cam, video_src, vidDir):
  while True:
    ret, frame = cam.read()
    h, w = frame.shape[:2]
    prev_frame = frame.copy()
    motion_history = np.zeros((h, w), np.float32)
    timestamp = 0
    while True:
      frame_num = frame_num + 1
      ret, frame = cam.read()
      if not ret:
        break
      frame_diff = cv2.absdiff(frame, prev_frame)
      gray_diff = cv2.cvtColor(frame_diff, cv2.COLOR_BGR2GRAY)
      ret, fgmask = cv2.threshold(gray_diff, DEFAULT_THRESHOLD, 1, cv2.THRESH_BINARY)
      timestamp += 1
      # update motion history
      cv2.motempl.updateMotionHistory(fgmask, motion_history, timestamp, MHI_DURATION)
      # normalize motion history
      mh = np.uint8(np.clip((motion_history-(timestamp-MHI_DURATION)) / MHI_DURATION, 0, 1)*255)
      cv2.imshow('motempl', mh)
      print frame_num
      if frame_num == 160:
          frame_num = 0
          imagename = video_src[0:-4] + '.jpg'
          cv2.imwrite(MHI_dir + '\\' + vidDir + '\\' + imagename, mh)
          return
```

Listing 1: Algoritmus MHI

V tejto ukážke 1 kódu je celý proces čítania videa a jeho jednotlivých snímkov za sebou so spracovaním na MHI. Funkcia *getImage* obsahuje tri vstupné parametre:

- **cam**
- **video_src**
- **vidDir**

cam, čo je objekt typu ***VideoCapture***, ktorý sa ďalej spracováva po jednotlivých snímkoch. Parameter ***video_src*** je názov videosúboru, ktorý sme použili na uloženie obrázku z videa. ***vidDir*** je názov priečinku, v ktorom sa obrázok MHI bude ukladať. Na čítanie snímkov sme použili knižnicu OpenCV, ako aj na spracovanie jednotlivých snímkov do MHI. Dôležitými parametrami, ktoré sme nastavovali boli *frame_num*, ktorý nám označoval počet prehratých snímkov, pri ktorom sme uložili vyhotovené MHI, ďalej to bol *DEFAULT_TRESHOLD*, ktorý nám filtroval nepotrebné pixely s veľmi nízkou hodnotou, pri trénovaní a testovaní sme zistili, že hodnota ***50*** bude dobrá. Akonáhle bol tento parameter vyšší alebo nižší, obrázky už neboli až tak dobre vyhodnocované.

Ďalším dôležitým parametrom je *MHI_DURATION*. Tento parameter nám vymedzuje dĺžku histórie pohybu v jednotke *timestamp*, akú má snímať. čím je hodnota vyššia, tým dlhšia história pohybu je na finálnom MHI zaznamenaná. Pre naše potreby sme zvolili tento parameter na hodnotu ***80***.

Obr. 11: Predspracované videá boxu.

Na obrázku 12 môžme vidieť príklad už predspracovaných MHI prostredníctvom nášho algoritmu. Takto sme si vytvorili všetkých 600 obrázkov z pohybov na ďalší výber príznakov, ktoré sme uložili podľa typu pohybu do jednotlivých priečinkov s ich názvami.

3.3 Výber príznakov

Keďže už máme predspracovaný obraz vo forme MHI, môžme pokračovať v extrahovaní príznakov z týchto obrazov. Na tento účel sme zvolili deskriptor HOG popísaný v časti 2.3.2. Týmto spôsobom vytvoríme vektor príznakov, ktorý budeme môcť následne spracovať prostredníctvom PCA a tak ho dať trénovať do klasifikátora pre každý jeden pohyb z trénovacej množiny. Príznaky z každého MHI obrázku by mali byť v tvare vektora konštantnej dĺžky.

3.3.1 Príznaky HOG

```
image = cv2.imread('image.jpg', 0)

def hogCreate(image):
    winSize = (64,64)
    blockSize = (16,16)
    blockStride = (8,8)
    cellSize = (8,8)
    nbins = 9
    winSigma = 4.
    gammaCorrection = 0
    nlevels = 64
    hog = cv2.HOGDescriptor(winSize, blockSize, blockStride, cellSize, nbins, winSigma,
         gammaCorrection,nlevels)
    winStride = (8,8)
    padding = (8,8)
    locations = ((10,20),)
    hist = hog.compute(image,winStride,padding,locations)
    arrVectorFinal.append(np.concatenate(hist))
```

Listing 2: Algoritmus HOG

V tomto algoritme 2 vstupuje do funkcie *hogCreate* parameter *image* je vlastne obrázkom, ktorý je výstupom funkcie*imread* z knižnice OpenCV. Pre dobré vybratie príznakov je potrebné nastaviť a vytvoriť *HOGDescriptor*, ktorý obsahuje viacero parametrov:

- winSize
- blockSize
- blockStride
- cellSize
- nbins
- gammaCorrection
- nlevels

winSize parameter určuje veľkosť okna na detekciu, musí byť zarovnaný podľa *blockStride* a *blockSize*.

blockSize je veľkosť bloku v pixeloch. Má byť zarovnaný na veľkosť bunky *cellSize*.

blockStride musí byť násobkom veľkosti bunky, znamená veľkosť kroku, ktorým sa blok bude posúvať.

cellSize je veľkosť bunky.

nbins je počet rozdelení uhlov do ktorých jendotlivé gradienty zapadajú.

gammaCorrection hodnota určuje, či sa má ppoužiť predspracovanie gamma úpravy alebo nie.

nlevels určuje maximálny počet detekčných okien, predvolená hodnota je 64.

Po takomto nastavení deskriptora HOG môžme následne spracovať vstupný obraz a vytvoriť tak histogram hodnôt, ktoré už sú reprezentovateľné príznaky, ktoré vložíme do vektora príznakov *arrVectorFinal*. Konečným výstupom je teda pole vektorov, ktoré obsahuje príznaky extrahované zo všetkých testovacích videí. Už tento vektor vieme následne použiť pre trénovanie klasifikátora, avšak pre zlepšenie výsledkov ešte tieto dáta upravíme. Vektor týchto príznakov sme zložili z troch MHI obrázkov z jedného videa, ktoré boli spracované vo štvrtej, šiestej a deviatej sekunde, čo dokáže presnejšie identifikovať pohyb jednotlivých aktérov.

3.3.2 Prídavné príznaky

Ďalšími príznakmi, ktoré sme sa rozhodli vybrať do vektora príznakov sú priemery hodnôt jednotlivých polí pixelov. Týmto algoritmom sme chceli znovu navýšiť úspešnosť, keďže pohyby v jednotlivých triedach sa vykonávajú v rôznych častiach snímaného obrazu. Týmito hodnotami vieme určiť v ktorých častiach obrazu MHI sa vykonával pohyb a v akej intenzite. Pre tento algoritmu sme potrebovali zvoliť vhodné parametre:

- Šírka okna
- Výška okna

Šírka okna nám udáva počet pixelov, ktoré budeme spracovávať v jednom rade pixelov v iterácii. **Výška okna** je potrebná na definovanie počtu pixelov spracovávaných v stĺpci pixelov v iterácii.

Tieto dva parametre je potrebné určiť ako delitele šírky (160px) a delitele výšky (120px) MHI obrázkov. V našej implementácii sme využili šírku a výšku s hodnotou 10.

```
HEIGHT = 10
WIDTH = 10

def processImg(image, boxes):
  count = 0
  rowBoxNum = 120 / HEIGHT
  colBoxNum = 160 / WIDTH
  sumOfBox = 0
  additionalVector = []
  value = 0

  data = np.asarray(image.getdata()).reshape(image.size)

  for i in range(0, rowBoxNum):
```

```
        for j in range(0, colBoxNum):
            for row in range(i*HEIGHT, i*HEIGHT + HEIGHT):
                for column in range(j*WIDTH, j*WIDTH + WIDTH):
                    sumOfBox = sumOfBox + data[column][row]
                    count = count + 1
            value = float(sumOfBox)/(WIDTH*HEIGHT)/255
            additionalVector.append(value)
            sumOfBox = 0
    return additionalVector
]
```

Listing 3: Prídavné príznaky

V algoritme 3 je vo funkcii *processImg* na vstupe obrázok MHI (*image* a počet okien pixelov s rozmermi 10 x 10 pixelov na jeden obrázok. Obraz sme si pre zjednodušenie vložili do *NumPy* poľa a následne iterovali. Aby sme získali normalizovaný príznak *(z rozmedzia <0,1>)*, potrebovali sme vydeliť hodnotu príznaku z jedného okna hodnotou *255*, keďže každý pixel nadobúda hodnoty*<0,255>* podľa svietivosti pixelu.

$$X = \frac{\sum_{i=1}^{pocet\_pixelov} hodnota\_pixelu_i}{255}$$

$X - výsledný\ príznak\ jedného\ okna\ pixelov$

$pocet\_pixelov - počet\ pixelov\ v\ okne$

$hodnota\_pixelu_i - hodnota\ farby\ pixelu\ na\ i - tej\ pozícii$

Obr. 12: Výpočet príznaku okna pixelov

Takto vypočítané príznaky sme pridávali do vektoru príznakov na následné spracovanie a úpravu v PCA.

3.3.3 Spracovanie pomocou PCA

V algoritme 4 na extrakciu komponentov, ktorú sme si popísali v časti 2.3.3 sme použili metódy triedy ***PCA*** knižnice *sklearn*. Pre vytvorenie objektu PCA, sme použili parameter *n_components*, ktorý udáva počet komponentov, koľko chceme vybrať z vektora príznakov *arrVectorFinal*. Tento počet komponentov musí byť nižší alebo nanajvýš rovný počtu vektorov príznakov. Natrénovaný model PCA s vektorom príznakov *arrVectorFinal* sme uložili pre ďalšie spracovanie v SVM.

```
pca = PCA(n_components)
pca.fit(arrVectorFinal)
X_t_train = pca.transform(arrVectorFinal)
joblib.dump(pca, 'ModelPCA.pkl')
```

Listing 4: Trénovanie PCA

Keďže v trénovacej množine máme iba 200 videí a tým by sme vedeli získať iba 200 hlavných komponentov, rozhodli sme sa v jednotlivých trénovaniach a testoch multiplikovať pole vektorov *(2x, 4x)* pred výpočtom PCA, aby sme dosiahli vyšší počet komponentov a tým aj lepší výsledok pri testoch.

3.4 Klasifikácia

Pre natrénovanie klasifikátora sme sa rozhodli použiť SVM, ktorý sme popísali v časti 2.3.4. Na implementovanie SVM sme použili triedu *svm* z knižnice *sklearn*. K dátam, ktoré máme uložené v *arrVectorFinal* potrebujeme ešte označenie, ktorý vektor do akej kategórie patrí. Preto sme vytvorili ďalší vektor, v ktorom je pre každé video označenie kategórie, kam vektor patrí. Teraz už môžme natrénovať SVM.

```
clf = svm.SVC(decision_function_shape='ovr')
clf.fit(arrVectorFinal, classVector)
joblib.dump(clf, 'Model.pkl')
```

Listing 5: Trénovanie klasifikátora

Algoritmus 5 využíva volanie funkcií *svm.SVC*, ktoré nám vytvorí klasifikátor (angl. Support Vector Classifier), rozhodovaciu funkciu sme zvolili *ovr* (angl. one versus rest), keďže máme viac ako dve triedy pohybov. Následne prostredníctvom funkcie *fit* sme natrénovali klasifikátor *clf*, kde boli vložené vektory príznakov *arrVectorFinal* a vektor tried *classVector*. Takto natrénovaný model bolo potrebné uložiť pre ďalšie použitie a testovanie.

3.5 Priebežné testovania a ladenie

Po naprogramovaní všetkých základných častí nášho projektu sme potrebovali otestovať naše riešenie, či náš klasifikátor funguje dobre a či môžme prejsť ku finalizácii výsledkov.

3.5.1 t-SNE

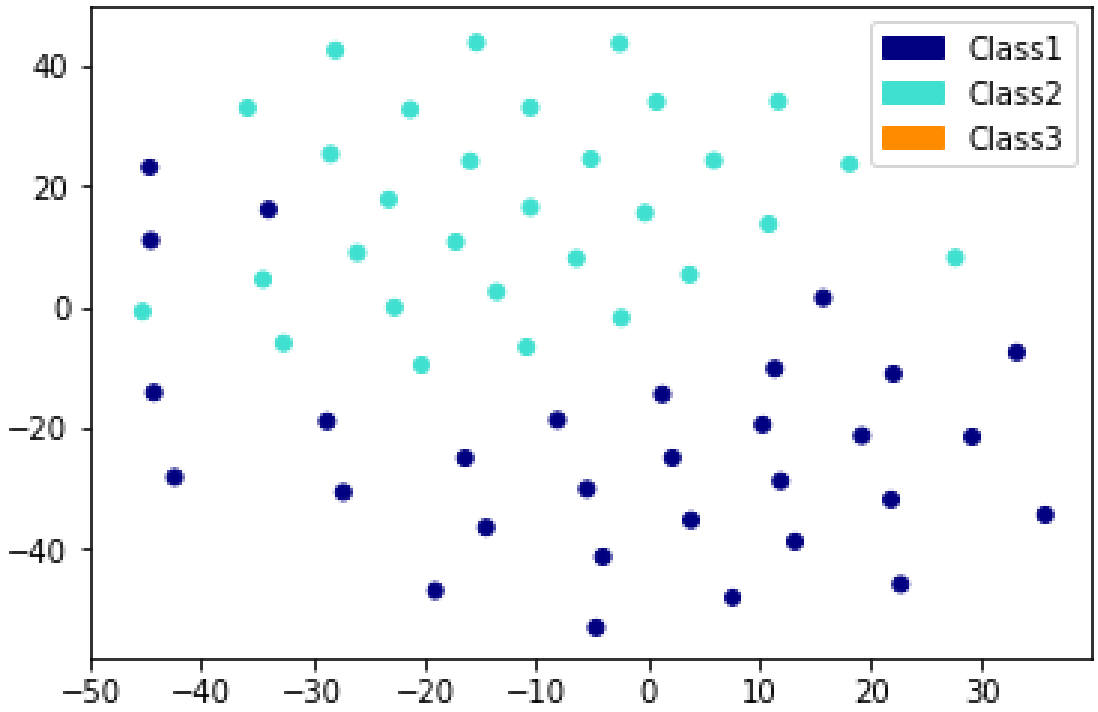
Aby sme vedeli výsledok trénovania našej SVM, potrebovali sme výsledky vizualizovať, ako prebehlo rozdelenie do tried. Na to sme využili testovací nástroj *t-Distributed Stochastic Neighbor Embedding (t-SNE)*, ktorého algoritmus nájdete v prílohe B. Tento algoritmus umožňuje vizualizáciu viacrozmerných dát do 2D priestoru. Túto techniku je možné implementovať pomocou aproximácií, ktoré môžu byť aplikované na datasety obrovských rozmerov. Autorom tohto algoritmu je ***Laurens van der Maaten***, ktorý pôsobí ako výskumný pracovník na poli strojového učenia a počítačového videnia. t-SNE implementoval pre množstvo jazykov vrátane jazyka Python, takže sme nemali problém

tento nástroj využívať. [17]

3.5.2 Prvotné testy

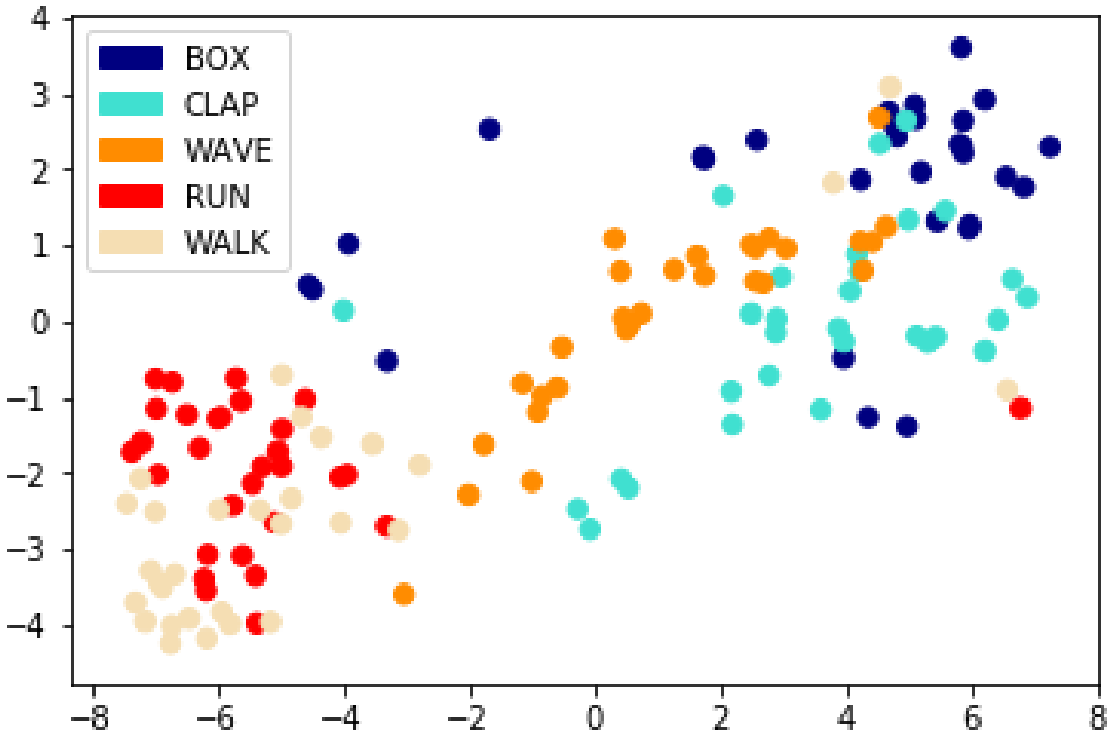
Ako prvý krok sme zvolili testovanie a tréning dvoch množín a teda beh a boxovanie. SVM sme natrénovali na tridsiatich videách z triedy behu a tridsiatich videách z triedy boxovania.

Výsledok trénovania bol pre nás dobrý, keďže už vizuálne bolo vidno rozdelenie týchto pohybov do dvoch tried. Výsledok je možné vidieť na obrázku 13.



Obr. 13: Test dvoch tried pohybu

Ako príklad uvedieme taktiež pokus trénovania s jedným MHI, ktorý bol vytvorený vo 4. sekunde priebehu pohybu, avšak bez získania prídavných príznakov 3.3.2 a využitia metódy PCA. Do tohto testu sme zapojili 5 tried pohybov, teda dataset KTH bez triedy pomalého behu. Takto natrénované SVM sme znovu podrobili testu *t-SNE*, kde výstup vyzeral ako na obrázku 14, výsledky síce dávali zmysel, avšak rozdelenie jednotlivých tried nebolo až tak prehľadné ako sme si predstavovali.



Obr. 14: Prvý test piatich tried pohybu

Z tohto dôvodu sme sa rozhodli analyzovať náš postup a pristúpiť k optimalizácii riešenia, zbieraniu a ladeniu príznakov za pomoci viacnásobného využitia MHI, prídavných príznakov 3.3.2 a aplikovania PCA, ako bolo spomenuté v častiach 3.3.1, 3.3.3, 3.3.2.

4 Porovnanie riešení

Záver

Conclusion is going to be where?
Here.

Zoznam použitej literatúry

1. *About - OpenCV library* [https://opencv.org/about.html]. (Accessed on 04/23/2018).
2. *Developer Documentation* [http://ffmpeg.org/developer.html]. (Accessed on 04/28/2018).
3. *Applications for Python | Python.org* [https://www.python.org/about/apps/]. (Accessed on 04/23/2018).
4. PEDREGOSA, F. et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*. 2011, roč. 12, s. 2825–2830.
5. *Frequently Asked Questions — SciPy.org* [https://www.scipy.org/scipylib/faq.html#what-is-numpy]. (Accessed on 04/28/2018).
6. AHAD, Md. Atiqur Rahman, TAN, Joo Kooi, KIM, Hyoungseop a ISHIKAWA, Seiji. Motion history image: its variants and applications. *Machine Vision and Applications*. 2010, roč. 23, s. 255–281.
7. *https://www.cs.utexas.edu/~chaoyeh/web_action_data/dataset_list.html* [https://www.cs.utexas.edu/~chaoyeh/web_action_data/dataset_list.html]. (Accessed on 04/22/2018).
8. A. BOBICK, J. Davis. *An appearance-based representation of action - IEEE Conference Publication* [https://ieeexplore.ieee.org/document/546039/?denied]. 1996. (Accessed on 04/23/2018).
9. AHAD, Md. Atiqur Rahman, TAN, Joo Kooi, KIM, Hyoungseop a ISHIKAWA, Seiji. Motion history image: its variants and applications. *Machine Vision and Applications*. 2010, roč. 23, s. 255–281.
10. DALAL, Navneet a TRIGGS, Bill. *Histograms of Oriented Gradients for Human Detection* [http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf]. 2008. (Accessed on 04/28/2018).
11. PEARSON, K. On Lines and Planes of Closest Fit to Systems of Points in Space. *Philosophical Magazine*. 1901, roč. 2, s. 559–572.
12. JOLLIFFE, I.T. *Principal Component Analysis*. Springer Verlag, 1986.
13. HALES, A. W. a PASSI, I. B. S. Jordan Decomposition. In: *Algebra: Some Recent Advances*. Ed. PASSI, I. B. S. Basel: Birkhäuser Basel, 1999, s. 75–87. ISBN 978-3-0348-9996-3. Dostupné z DOI: 10.1007/978-3-0348-9996-3_5.

14. SCHOLKOPF, Bernhard a SMOLA, Alexander J. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press, 2001. ISBN 0262194759.

15. HASTIE, Trevor, TIBSHIRANI, Robert a FRIEDMAN, Jerome. *https://web.stanford.edu/~hast* [`https://web.stanford.edu/~hastie/Papers/ESLII.pdf`]. (Accessed on 04/28/2018).

16. YOUSEFI, B a CHU KIONG, Loo. Comparative Study on Interaction of Form and Motion Processing Streams by Applying Two Different Classifiers in Mechanism for Recognition of Biological Movement. 2014, roč. 2014, s. 723213.

17. MAATEN, Laurens van der a HINTON, Geoffrey. Visualizing Data using t-SNE. *Journal of Machine Learning Research*. 2008, roč. 9, s. 2579–2605. Dostupné tiež z: `http://www.jmlr.org/papers/v9/vandermaaten08a.html`.

Prílohy

A Štruktúra elektronického nosiča

/CHANGELOG.md

· file describing changes made to FEIstyle

/example.tex

· main example *.tex* file for diploma thesis

/example_paper.tex

· example *.tex* file for seminar paper

/Makefile

· simply Makefile – build system

/fei.sublime-project

· is project file with build in Build System for Sublime Text 3

/img

· folder with images

/includes

· files with content

/bibliography.bib

· bibliography file

/attachmentA.tex

· this very file

B Algoritmy

```
import numpy as np
from sklearn.decomposition import PCA, IncrementalPCA
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from sklearn import manifold
trainData = np.load('arrVectorFinal.npy')
bag_of_labels = np.load('classVector.npy')

print(trainData.shape)

n_components = 200
ipca = IncrementalPCA(n_components=n_components, batch_size=200)
X_ipca = ipca.fit_transform(trainData)

#print 'Explained variation per principal component: {}'.format(ipca.explained_variance_ratio_)
print("Sum {0}".format(np.sum(ipca.explained_variance_ratio_)))

colors = ['navy', 'turquoise', 'darkorange', 'red', 'wheat', 'yellow']
     #navy-vojnove_lodstvo,turquoise-tyrkysova(modro-zelena)


bag_of_labels = np.squeeze(bag_of_labels)

if False:
    area = np.full(X_ipca.shape, 25)
    plt.scatter(X_ipca[bag_of_labels == 1,0], X_ipca[bag_of_labels == 1,1], s=area, c=colors[0])
    plt.scatter(X_ipca[bag_of_labels == 2,0], X_ipca[bag_of_labels == 2,1], s=area, c=colors[1])
    plt.scatter(X_ipca[bag_of_labels == 3,0], X_ipca[bag_of_labels == 3,1], s=area, c=colors[2])
    plt.scatter(X_ipca[bag_of_labels == 4,0], X_ipca[bag_of_labels == 4,1], s=area, c=colors[3])
    plt.scatter(X_ipca[bag_of_labels == 5,0], X_ipca[bag_of_labels == 5,1], s=area, c=colors[4])
    plt.scatter(X_ipca[bag_of_labels == 6,0], X_ipca[bag_of_labels == 6,1], s=area, c=colors[5])

    plt.show()
    plt.gcf().clear()

tsne = manifold.TSNE(n_components=2, init='random',
                     random_state=0, perplexity=50)
Y = tsne.fit_transform(X_ipca)

area = np.full(Y.shape, 40)
plt.scatter(Y[bag_of_labels == 1,0], Y[bag_of_labels == 1,1], s=area, c=colors[0])
plt.scatter(Y[bag_of_labels == 2,0], Y[bag_of_labels == 2,1], s=area, c=colors[1])
plt.scatter(Y[bag_of_labels == 3,0], Y[bag_of_labels == 3,1], s=area, c=colors[2])
plt.scatter(Y[bag_of_labels == 4,0], Y[bag_of_labels == 4,1], s=area, c=colors[3])
plt.scatter(Y[bag_of_labels == 5,0], Y[bag_of_labels == 5,1], s=area, c=colors[4])
plt.scatter(Y[bag_of_labels == 6,0], Y[bag_of_labels == 6,1], s=area, c=colors[5])

navy_patch = mpatches.Patch(color='navy', label='BOX')
turquoise_patch = mpatches.Patch(color='turquoise', label='CLAP')
```

```
dark_orange_patch = mpatches.Patch(color='darkorange', label='WAVE')
red_patch = mpatches.Patch(color='red', label='RUN')
gold_patch = mpatches.Patch(color='wheat', label='WALK')
#yellow_patch = mpatches.Patch(color='yellow', label='JOGG')
plt.legend(handles=[navy_patch, turquoise_patch, dark_orange_patch, red_patch, gold_patch])

plt.show()
```

Listing B.1: Test TSNE

C Výpis sublime

```
../../ fei .sublime-project
```

Listing C.1: Ukážka sublime-project