Gymnázium Myjava
Jablonská 5, 90701 Myjava

STREDOŠKOLSKÁ ODBORNÁ ČINNOSŤ

Číslo odboru: 11 – Informatika

Rozpoznávanie áut a výpočet rýchlosti jazdy pomocou počítačového videnia a knižnice OpenCV

Riešitelia:
Ondrej Škopek
ročník štúdia: **tretí**

2013
Myjava

Gymnázium Myjava
Jablonská 5, 90701 Myjava

STREDOŠKOLSKÁ ODBORNÁ ČINNOSŤ

Číslo odboru: 11 – Informatika

Rozpoznávanie áut a výpočet rýchlosti jazdy pomocou počítačového videnia a knižnice OpenCV

Riešitelia:
Ondrej Škopek
ročník štúdia: **tretí**

konzultant:
Mgr. Peter Škopek

2013
Myjava

Dolupodpísaný Ondrej Škopek vyhlasujem, že túto prácu som vypracoval bez cudzej pomoci, na základe svojich poznatkov a literatúry, ktorá je uvedená na konci práce.

..... 26. januára 2013 v Myjave

Ďakujem konzultantovi Mgr. Petrovi Škopkovi za odbornú pomoc a námety,
ktoré mi poskytol.

Obsah

Úvod

Počítačové videnie je jedna z oblastí informatiky (výpočtovej techniky). Pôvodne sa vyčlenila ako časť umelej inteligencie niekedy v polovici 20. storočia. Odvtedy neustále geometricky narastá obsiahlosť vedomostí v tejto oblasti. Každým rokom sa objavujú nové a revolučné metódy, ktoré sú rýchlejšie, efektívnejšie a stále komplikovanejšie. Taktiež sa využíva v praxi čím ďalej častejšie, a aktívne sa s ňou stretáva dnes takmer každý.

Konkrétna téma praktickej časti tejto práce, teda „Rozpoznávanie áut a výpočet rýchlosti jazdy“, sa dá aplikovať rôznymi spôsobmi. Hlavnou motiváciou je možné využitie ako nízkonákladová rýchlostná „pasca“ proti vodičom prekračujúcim rýchlosť v obciach. Výhodou takejto formy akéhosi „meradla rýchlosti“ je, že sa dá umiestniť hocikam do miniatúrnej krabičky a môže vykonávať svoju funkciu neustále a za minimálne náklady. Cena prístroja by sa dala odhadnúť na približne 25 €, náklady na prevádzku spočívajú iba v napájaní na elektrickú sieť a cene za dátový prenos (ľubovolnou formou). Výpočty by sa diali mimo týchto malých prístrojov, na zostave serverov.

Pre túto tému sme sa rozhodli z dôvodu osobnej fascinácie týmto odborom a jeho ďalekosiahlym použitím okrem už spomenutého, ako napríklad rozpoznávanie tvárí na fotografiach, čítačky QR kódov, rozpoznávanie ŠPZ, textu, či inteligentné domy. V budúcnosti možno aj automatická vizuálna autentifikácia vo verejných budovách, autopilot dopravných prostriedkov a inteligentné bezpečnostné systémy. Počítačové videnie sa dá do istej miery aplikovať na takmer každý problém. Cieľom tejto práce je preto zvýšenie všeobecného povedomia o počítačovom videní a jeho konkrétna aplikácia na rozpoznávanie áut a kalkuláciu ich rýchlosti.

1 Ciele práce

Cieľom tejto práce je rozšíriť povedomie o počítačovom videní a jeho využití v bežnom živote. Ozrejmujeme niektoré základné princípy detekcie objektov a nosné myšlienky najpoužívanejších algoritmov.

Práca sa zaoberá aj presným postupom inštalácie knižnice počítačového videnia OpenCV, pretože o tejto téme existuje veľmi málo informácií a v slovenskom jazyku prakticky neexistujú. Chceme opísať tento postup do detailov, aby bol čo najviac reprodukovateľný.

Vlastná práca je pokus o aplikáciu vedomostí z oblasti počítačového videnia do praxe v oblasti monitorovania dopravy, presnejšie špecifikované ako kontrola dodržiavania povolenej jazdnej rýchlosti. Navrhujeme riešenie tohto problému pomocou detekcie objektov počítačovým videním, na rozdiel od konvenčných radarových metód.

2 Metodika práce

Pri písaní práce sme využili viacero metód. V prípravnej fáze to bolo štúdium odbnornej literatúry. Neskôr sme využili metódu komparácie (porovnania) dokumentácie a dostupných informácií o počítačovom videní a knižnici OpenCV a reálneho stavu vecí pri aplikácii knižnice a počítačového videnia v našom programe CarCV.

Práca pozostáva zo štyroch hlavných kapitol. Úvod vysvetľuje formuláciu témy práce a popisuje, prečo sme si danú tému vybrali.

V prvej číslovanej kapitole vytýčime ciele práce, ku ktorým sa budeme snažiť počas celej práce čo najviac priblížiť.

Druhou kapitolou je Metodika práce. Stručne v nej charakterizujeme postupy a metódy výskumu, opisujeme obsah jednotlivých kapitol.

Teoretickými východiskami sa zaoberá tretia kapitola. Opisuje teóriu počítačového videnia, knižnicu OpenCV a jej základné moduly. Zaoberá sa aj princípom detekcie objektov, ozrejmuje niekoľko metód detekcie.

Vlastná práca sa nachádza v štvrtej kapitole. Detailne a reprodukovateľne opisuje inštaláciu knižnice OpenCV. Trénovanie kaskády pomocou tejto knižnice a teória výpočtu rýchlosti sú ďalšie dve podkapitoly. Na záver kapitoly opisujeme spôsob fungovania programu CarCV a test úspešnosti nášho programu.

V kapitole Záver hodnotíme výsledky práce a ich možnú aplikáciu v praxi. Zamýšľame sa nad úskaliami, ktoré v praxi ohrozujú presnosť navrhovanej metódy výpočtu rýchlosti.

Program CarCV je napísaný pomocou programovacieho prostredia Eclipse, v jazyku C++ s využitím knižnice OpenCV. Sadzba textu bola vytvorená pomocou systému LaTeX. Počas celej práce, vrátane písania tejto práce, pracujeme v operačnom systéme Linux, konrétne Fedora 17.

3 Teória – Počítačové videnie

3.1 Počítačové videnie

Ballard ([1]) definuje počítačové videnie ako konštrukciu jasných, významových opisov fyzických objektov z obrazov. Porozumenie obrazu[1] je veľmi odlišné od spracovania obrazu[2], ktoré skúma premeny obrazu na obraz, nie vytváranie výslovného opisu obrazu. Opis je nevyhnutný predpoklad pre rozpoznávanie, manipuláciu a uvažovanie nad objektom.

3.1.1 História počítačového videnia

Ako uvádza vo svojej knihe Ballard ([1]) v 1982:

„Počítačové videnie je relatívne nová a rýchlo rastúca oblasť. Prvé experimenty boli vykonané v neskorých päťdesiatych rokoch 20. storočia a mnohé fundamentálne koncepty boli vyvinuté počas niekoľkých posledných rokov. S týmto rapídnym rastom vznikli aj viaceré nápady v rôznych súvisiacich oblastiach, ako napríklad umelá inteligencia, psychológia, počítačová grafika a spracovanie obrazu.“

Spočiatku bola táto oblasť veľmi úzko prepojená práve s oblasťou umelej inteligencie. „Sen“ o inteligentných automatoch sa datuje až do stredoveku. Prvýkrát bol vyslovený Turingom vo forme testu, tzv. Turingov test. Test spočíva v tom, že necháme jedného človeka dávať otázky dvom respondentom, ktorých nevidí. Jeden respondent je človek, druhý počítač. Ak pýtajúci sa človek nedokáže na základe odpovedí rozlíšiť, ktorý respondent je človek a ktorý stroj, potom možno tento stroj považovať za inteligentný. V Turingovom teste dodnes žiaden počítač neuspel.

Odvtedy sa o rozvoj počítačového videnia pokúšali hlavne pracovníci v oblasti umelej inteligencie, ktorých cieľom bolo vybaviť počítače schopnosťami spracovať informácie porovnateľné so schopnosťami biologických organizmov. Od začiatku bolo hlavným problémom spracovávanie senzorických vstupov.

Oblasť počítačového videnia je dnes oveľa samostatnejšia, rozšírenejšia a aj častejšie používaná v bežnej praxi. Niektoré bežné využitia uvádza Ftáčnik ([3]) vo svojej

[1] Image understanding
[2] Image processing

zverejnenej prednáške:

- Monitorovanie
- Montáž a vizuálna kontrola
- Rozpoznávanie obrazcov, objektov a udalostí
- Rekonštrukcia scény, vizualizácia a editovanie
- Rozpoznávanie ľudského pohybu, tvárí, gest a ich interpretácia

Prednáška obsahuje aj zoznam oblastí, kde sa počítačové videnie aktívne používa:

- Poľnohospodárstvo a lesné hospodárstvo, rybolov
- Automobilový priemysel – zváranie, sledovanie cesty, automatický pilot
- Automatická predpoveď počasia
- Trojrozmerné modely miest
- Riadenie a monitorovanie dopravy
- Robotika
- DPZ[3] (aj mimoriadne situácie)
- Monitorovanie priestorov – domy, banky
- Medicínske spracovanie obrazu (CT[4], NMR[5], ultrazvuk, kolonoskopia)
- Kontrola pri plnení a balení
- Triedenie potravín podľa kvality, veľkosti – kávové zrnká, ryby
- Autentifikácia
- Kontrola čiarových kódov
- HD-televízia

Príchodom najnovších zariadení (smartfónov, tabletov, . . .) sa otvára úplne nová dimenzia tohto oboru, ktorý ma obrovskú perspektívu a pravdepodobne ešte rozsiahlejšie a intenzívnejšie využitie v budúcnosti.

3.1.2 Princípy počítačového videnia

Szeliski ([5]) uvádza vo svojej knihe viacero základných princípov počítačového videnia.

Väčšina procesov v oblastiach ako fyzika (radiometria, optika, dizajn sensorov) a počítačová grafika fungujú na princípe premeny objektu na model. Modelujú ako sa objekty pohybujú, animujú, ako sa svetlo odráža od ich povrchu, rozptyľuje sa v atmosfére, je lomené objektívom fotoaparátu (kamery, oka) a nakoniec premietaný na plochú (alebo aj zakrivenú) obrazovú rovinu.

[3] Diaľkový prieskum Zeme
[4] Computed (axial) Tomography
[5] Nuclear magnetic resonance

Aj keď počítačová grafika ešte nie je dokonalá (tzn. počítačovo vytvorené modely objektov vo filme sa stále dajú rozoznať od reálnych), v niektorých vytýčených prostrediach, napríklad renderovanie statickej scény zloženej z každodenných objektov alebo animácie vyhynutých dinosaurov (stať 3.1.3, odrážka spájanie pohybu) je ilúzia reality perfektná.

V počítačovom videní sa snažíme dosiahnúť presný opak, teda opísať svet, ktorý vidíme na jednom alebo viacerých obrazoch a rekonštruovať jeho vlastnosti; tvar, osvetlenie, distribúcia farby.

Ľudia a zvieratá majú úžasnú schopnosť vykonávať tieto procesy takmer bez námahy, zatiaľ čo algoritmy počítačového videnia sú veľmi náchylné k chybám. Ľudia, ktorí sa nikdy nestretli s touto problematikou často podceňujú obtiažnosť problému. Pôvod tohto mylného predpokladu, že videnie by malo byť jednoduché sa datuje niekam do obdobia začiatkov umelej inteligencie. Predpokladalo sa, že kognitívne (plánovacie a logiku dokazujúce) časti inteligencie boli interne zložitejšie než percepčné zložky (zložky vnímania).

Ballard ([1]) dodáva krátke vysvetlenie: „Ľudia vnímajú svet zložený z trojrozmerných objektov s veľa invariantnými vlastnosťami. Tieto vstupné vizuálne dáta objektívne nepreukazujú žiadnu zodpovedajúcu súvislosť – obsahujú veľa irelevantných ba dokonca zavádzajúcich sekcií. Nejakým spôsobom však náš vizuálny systém dokáže (od retinálnej až po kognitívnu úroveň) rozumieť, resp. vytvoriť poriadok v, chaotickom vizuálnom vstupe. Robí to pomocou vnútorných informácií, ktoré s konštantnou spoľahlivosťou extrahuje z daného vstupu a tiež pomocou predpokladov a vedomostí, ktoré sú aplikované v rôznych stupňoch vizuálneho spracovania.“

Úloha počítačového videnia je teda relatívne jednoznačná. Aplikáciou rôznych vedomostí a iných filtrov na daný vizuálny vstup sa čo najviac priblížiť vizuálnym schopnostiam biologických organizmov.

Napriek všetkým pokrokom a viac než 50 ročnému výskumu, dvojročné dieťa vie zistiť z daného obrazu rádovo viac informácií a urobiť to rádovo rýchlejšie než najlepší súčasný algoritmus počítačového videnia alebo ich kombinácia (Szeliski ([5])). Prečo je videnie také obťiažné? Čiastočne preto, lebo je to inverzný problém, teda hľadáme riešenie s nedostatočnými informáciami na plné vymedzenie riešenia. Musíme sa preto uchýliť k modelom založeným na pravdepodobnosti a fyzike na rozlíšenie medzi potencionálnymi riešeniami. Avšak modelovanie vizuálneho sveta v celej svojej komplexnosti je oveľa zložitejšie, než povedzme modelovanie vokálnej sústavy produkujúcej hovorené zvuky za cieľom vytvoriť „hovoriaci“ algoritmus.

3.1.3 Počítačové videnie v praxi

Szeliski ([5]) uvádza ďalšie a viac špecifikované príklady použitia počítačového videnia v širokom spektre aplikácií v praxi:

- **Optické rozpoznávanie znakov** (OCR[6]): Čítanie ručne písaných poštových smerovacích čísiel na poštových zásielkach a listoch, digitalizácia starých kníh, automatické rozpoznávanie ŠPZ[7] (skrátene: ANPR[8]).
- **Inšpekcia strojov**: Rýchla inšpekcia súčiastok pre zaručenie kvality použitím priestorového videnia so špeciálnym osvetlením, na meranie tolerancie odchýľok krídiel lietadiel alebo automobilových súčiastok, hľadanie poškodení pri odlievaní ocele pomocou Röntgenového zobrazovania.
- **Obchod**: Rozpoznávanie objektov pre automatizované pokladne.
- **Stavba 3D modelov** (fotogrametria): Plne automatizovaná konštrukcia 3D modelov z leteckých fotografií – v systémoch generovania máp ako napr. Google Maps, Bing Maps, ...
- **Zobrazovanie v medicíne**: Registrácia predoperačných a intraoperačných obrazov alebo dlhodobé štúdie morfológie ľudského mozgu počas starnutia.
- **Bezpečnosť premávky**: Detekcia neočakávaných prekážok (napríklad chodci na ceste), pri podmienkach kde klasické aktívne techniky videnia ako radar alebo lidar nefungujú dobre, alebo pre plne automatizované vedenie vozidla.
- **Spájanie pohybu**[9]: Spájanie počítačom generovaných obrazov (CGI[10]) s naživo natáčanými zábermi na základe sledovania významných bodov – v zdrojovom videu na odhadnutie pohybu 3D kamery a tvaru prostredia. Takéto techniky sa často používajú v Hollywoode (napr. vo filme Jurassic Park). Zároveň vyžadujú precízne použitie tzv. zelených plachiet kvôli presnému vkladaniu nových objektov medzi objekty popredia a objekty pozadia.
- **Snímanie pohybu** (mocap[11]): Použitie retro-reflexných značiek a sústavy kamier alebo inej techniky založenej na počítačovom videní na zaznamenanie pohybu hercov pre účel počítačovej animácie.
- **Stále pozorovanie**: Monitoring neoprávneného vstupu do objektu, analyzovanie premávky na cestách a diaľniciach, monitoring plaveckých bazénov na detekciu topiacich sa ľudí.
- **Biometria a rozpoznávanie otlačkov prstov**: Automatická prístupová autentifikácia (automatizované prihlasovanie) ako aj rôzne forenzné použitia.

[6] Optical character recognition
[7] Štátne poznávacie značky
[8] Automatic Number Plate Recognition
[9] Match move
[10] Computer-Generated Imagery
[11] Motion capture

3.2 OpenCV

OpenCV (Open Source Computer Vision Library, `http://www.opencv.org`) je open-source knižnica, ktorá obsahuje niekoľko stoviek algoritmov týkajúcich sa počítačového videnia. Je napísaná prevažne v jazyku C++ a má modulárnu štruktúru.

3.2.1 Moduly

Zopár základných modulov, ktoré OpenCV obsahuje:

- **core** – Kompaktný modul, ktorý definuje základné dátové štruktúry, vrátane nahusteného viac-dimenzionálneho poľa *Mat* a základných funkcií používaných inými modulmi
- **imgproc** – Modul na spracovanie obrazu, ktorý obsahuje napríklad lineárne aj nelineárne filtrovanie, geometrické premeny obrazu (zmena veľkosti, perspektívy, premapovanie na báze tabuľky), zmena farebného priestoru, histogramy, ...
- **video** – Video-analytický modul, obsahuje predpokladanie pohybu, vynímanie pozadia a sledovanie objektov
- **calib3d** – Základné viac-pohľadové geometrické algoritmy, jedno a dvojkamerová kalibrácia, predpokladanie pózy objektu, priestorová podobnosť a časti 3D rekonštrukcie
- **features2d** – Detektor, deskriptor a komparátor význačných častí obrazu
- **objdetect** – Detekcia objektov a inštancie preddefinovaných tried (napr. tváre, oči, poháre, ľudia, autá, ...)
- **highgui** – Ľahko použiteľný interface na zaznamenávanie videa, obrázové a videové kodeky, jednoduché možnosti grafického zobrazenia
- **gpu** – Algoritmy rôznych modulov urýchlené pomocou GPU

3.3 Detekcia objektov

Detekcia objektov v počítačovom videní je pomerne rozsiahla a komplikovaná téma. Dá sa robiť mnohými metódami, postupmi, existuje veľmi veľa algoritmov spojených s hľadaním objektov v obraze. Niektoré bežne používané, ktoré sme využili aj v praxi, sa nachádzajú v tejto kapitole.

3.4 Metódy a algoritmy detekcie objektov

3.4.1 LBP

LBP (Local Binary Patterns, Lokálne Binárne Vzory) ([4]) je jednoduchý avšak veľmi efektívny textúrový operátor, ktorý označuje pixely obrazu vyhodnocovaním okolitých pixelov a výsledok zhodnotí ako binárne číslo (pozri prílohu C).

Vďaka jeho diskriminačnej úrovni a jednoduchej vypočítateľnosti sa tento textúrový operátor stal rozšíreným riešením rôznych problémov. Jeho najväčšou dominantou pri využití v praxi je jeho odolnosť voči zmenám grayscale hodnôt pixelov (pri prevedení obrazu do farebnej škály grayscale), ktoré môže byť podmienené napríklad zmenou svetla v danej scéne. Ďalšou dôležitou vlastnosťou je jednoduchosť jeho výpočtu, ktorá umožňuje analýzu obrazov v náročných podmienkach, ako sú napr. operácie v reálnom čase.

Na opis daného pixelu používa tento operátor binárne číslo, ktoré pre daný výpočet $LBP_{P,R}$ skúma počet pixelov P v okruhu R (rádius). Teda pre danú prílohu Cplatí $LBP_{8,1} = 15 = 00001111$, čo znamená že počet prechodov (z 0 na 1 alebo naopak) tohto LBP kódu je 1. Kódy, ktorých počet prechodov je menší než 2 sa nazývajú uniformné, tie čo majú tri a viac prechodov sa nazývajú neuniformné (01010010 – 6 prechodov).

Pri skúmaní operátora pre (8,R) je týchto kódov 256, z ktorých je uniformných len 59 rôznych. Avšak pri skúmaní vzorov pre obrazy sa zistilo, že až 90% z celkového počtu vzorov v obraze je uniformných. Pri operátore pre (16,R) je to asi 70%. Hlavná výhoda skúmania rozdielov medzi počtom uniformných a celkových prechodov je v tom, že uniformné kódy su invariantné k rotácií. Teda pokiaľ strata vzniknutá ignorovaním neuniformných vzorov nie je priveľká, môžeme použiť iba uniformné a získame tak rotačnú invariantnosť.

V prístupe LBP ku klasifikácii textúr sa výskyt LBP kódov v obraze ukladá do histogramu. Klasifikácia sa potom prevádza jednoduchým počítaním podobností histogramov. Avšak takýto prístup pri komplexnejších obrazoch vedie k strate priestorových informácií. Preto sa pri problémoch tohto druhu snaží dosiahnúť kodifikácia textúrovej informácie pri uchovaní jej miesta. Jedným zo spôsobov ako je to možné dosiahnuť je použitie LBP textúrových deskriptorov na opísanie viacerých lokálnych častí a ich následné spojenie do jedného globálneho opisu. Tento prístup je taktiež viac odolný voči zmene pozorovacieho uhla alebo osvetlenia.
Základný dvojkrokový postup opisu objektov:

1. Obraz je rozdelený do lokálnych oblastí a z každej je vypočítaný LBP deskriptor (de facto histogram) individuálne
2. Deskriptory sú poskladané do formy jedného globálneho deskriptora

Tento globálny deskriptor má tri úrovne lokality: LBP textúrové deskriptory na úrovni pixelov, regionálne súčty týchto histogramov a nakoniec globálny súčet všetkých regionálnych histogramov. Pri používaní spájacích metód založených na spájaní histogramov tieto regionálne oblasti nemusia byť obdĺžniky. Nemusia byť dokonca ani rovnakej veľkosti či tvaru, nemusia pokryť celý obraz, ba môžu sa aj prekrývať.

3.4.2 Viola–Jones metóda

Viola–Jones metóda (bežne nazývaná HAAR metóda) bola vyvynutá Paulom Violom a Michaelom Jonesom v roku 2001, pôvodne nazvaná „Rapid Object Detection using a Boosted Cascade of Simple Features“ ([6]). K problému detekcie objektov pristupuje spôsobom strojového učenia. Pôvodné využitie tejto metódy bola rýchla a spoľahlivá detekcia tvárí, no neskôr bola aplikovaná aj na iné scenáre. Dosahuje rýchlosť 15 obrazov za sekundu na bežnom 700 MHz Intel Pentium III (samozrejme, dnes sú už oveľa rýchlejšie procesory).
Vyznačuje sa tromi črtami, ktoré priniesla ako úplne nové koncepty:

1. Integrálny obraz ako spôsob reprezentácie obrazu, ktorý umožňuje detektora takú rýchlosť. Pracuje na princípe, že každý bod obrazu je súčtom pixelov nad ním a vľavo od neho. Teda bod, ktorý je pravým dolným rohom obrazu je súčtom všetkých pixelov obrazu. Teda ak daný bod (x, y) je aktuálna pozícia v obraze I, platí:
$$I(x, y) = \sum_{\substack{x' \leq x \\ y' \leq y}} i(x', y')$$
2. Rýchly učiaci sa algoritmus založený na algoritme AdaBoost, ktorý vyberá malý počet kritických vizuálnych čŕt z väčšej vzorky a vkladá ich do extrémne efektívnych „classifiers“ (de facto jednoduchých filtrov), ktoré o danom regióne pixelov hovoria, či sa daný objekt v ňom nachádza.
3. Metóda na skladanie „classifiers“ do väčšieho celku, tzv. cascade. Je to akoby rad filtrov (pozri prílohu A), kde ak čo len jeden z nich vypočíta negatívny nález, automaticky je celý región pixlov negatívny a nepokračuje sa ďalej v rade filtrov na ďalší. To znamená, že venuje málo času nezaujímavým oblastiam a viac času strávi pri sľubných regiónoch.

Pri výbere kritických čŕt používa matematický pojem „Haar features“ (Haar črty). Je to istá forma kombinácií obdĺžnikov, ktoré symbolizujú dané reálne črty objetku na obraze. Na účel rozoznávania sa viac používajú v mierne upravenej forme. Nazývajú sa „Haar-like“ črty, teda črty podobné Haar črtám (pozri prílohu B).

Výhoda Haar-like čŕt je v tom, že pri využití princípu integrálneho obrazu ich dokážeme veľmi rýchlo vypočítať, a keď už sú raz vypočítané, vieme ich prepočítať pri ľubovolnej mierke či pozícii na obraze v konštantnom čase.

3.4.3 SURF

SURF (Speeded Up Robust Features) bol pôvodne vyvynutý na *Computer Vision Laboratory, Department of Information Technology and Electrical Engineering, ETH Zürich* a zverejnený na ECCV 2006[12], neskôr upravený a vydaný v žurnáli CVIU ([2]). Skladá sa z detektora a deskriptoru daného obrazu, samotné porovnávanie priamo nešpecifikuje.

Výsledok tejto metódy, ktorý je nezávislý od mierky či rotácie daného opisovaného objektu alebo scény, je údajne ešte aj oveľa rýchlejšie vypočítaný než iné používané metódy.

Úlohou tejto metódy je detekcia zhodných bodov dvoch obrazov danej scény alebo objektu. SURF Detektor na to využíva tri hlavné kroky:

1. „Významné body“ (napr. roh, blob[13], T-križovatka (bod spájajúci 3 hrany), ...) sú zvolené v rôznych miestach na obraze. Najváženejšia vlastnosť detektora je jeho reprodukovateľnosť. Tá vyjadruje spoľahlivosť detektora nájsť tie isté fyzické významné body pri rôznych podmienkach zobrazovania.
2. Okolie každého významného bodu je reprezentované funkčným vektorom[14]. Deskriptor musí byť charakterizujúci a zároveň odolný voči tzv. obrazovému šumu, detekčným chybám, geometrickým a fotometrickým deformáciam.
3. Nakoniec sú tieto opisné vektory z dvoch rôznych obrazov navzájom porovnávané. Porovnávanie je založené na vzdialenosti medzi vektormi, napr. Euklidovou vzdialenosťou. Dimenzia deskriptora má priamy dopad na čas, ktorý trvá jeho exekúcia. Na porovnávanie významných bodov sa teda preferujú detektory s menej dimenziami. Avšak málodimenzionálne funkčné vektory sú vo všeobecnosti menej rozlíšeniaschopné než vysokodimenzionálne.

V niektorých prípadoch sa môže vynechať nezávislosť na rotácii objektu alebo scény, a teda dosiahnuť iba mierkovo nezávislú verziu deskriptora, ktorý sa označuje U-SURF (Upright SURF). To sa prejaví nielen vo zvýšenej rýchlosti, ale aj vo zvýšenej rozlišovacej schopnosti. Bežne sa táto úprava využíva napríklad v navigácii robotov alebo vo vizuálnej navigácii turistov, kedy sa kamera otáča takmer výlučne iba na zvislej osi. SURF nevyužíva farbu obrazu (pracuje s nimi vo farebnej škále grayscale) na svoje výpočty, čo taktiež zvyšuje jeho rýchlosť.

SURF má teda jednoznačnú výhodu oproti iným metódam: je rýchlejší, kompaktnejší a nezávislý od otočenia objektu alebo scény. Avšak ako jeho nevýhoda sa môže prejaviť inkonzistentnosť a nespoľahlivosť.

[12] 9th European Conference on Computer Vision; http://eccv2006.tugraz.at/
[13] http://en.wikipedia.org/wiki/Blob_detection
[14] http://en.wikipedia.org/wiki/Feature_vector

4 Vlastná práca

V tejto kapitole sa zaoberáme vlastnou prácou týkajúcou sa rozpoznávania áut a výpočtu ich rýchlosti. Pracujeme s knižnicou OpenCV a jej základnými modulmi na vytváranie vzoriek a trénovania kaskády. Taktiež ju využívame vo vlastných programoch, hlavne pri detekcii áut, triedení testových obrazov na pozitívne a negatívne, nakoniec aj pri roztriedení jednotlivých áut. Výpočet rýchlosti vykonávame jednoduchými operáciami s výsledkami predchádzajúcich operácii. Kvôli malému počtu dostupných informácií o tejto téme a ich ešte menšom počte v slovenskom jazyku, opíšeme detailne celý postup od kompilácie OpenCV až po výsledky testov vykonaných našimi programami.

Potrebné programy na uskutočnenie všetkých operácií v tejto práci sa dajú nainštalovať pomocou príkazu:

```
yum install make cmake gcc gcc-c++ libpng* libjpeg* libtiff* jasper
jasper-* tbb* gtk2 gtk2-devel numpy* pkgconfig python python-devel
python-libs ffmpeg* libavc1394* libdc1394*
```

Na niektoré z daných programov je treba mať nastavený aj repozitár RPMFusion[1].

4.1 Použitie OpenCV

V prvom rade treba nainštalovať knižnicu OpenCV. Ideálny spôsob ako to spraviť je stiahnuť najnovšiu stabilnú verziu zo stránky `http://opencv.org/downloads.html`.

4.1.1 Verzia

Pre účely tejto práce sme použili verziu knižnice OpenCV 2.4.3, skompilovanú pre 32-bitový procesor Intel Core2Duo, na operačnom systéme Linux, distribúcia Fedora 17 Beefy Miracle. Všetky kompilačné nastavenia a parametre sa zhodujú s nasledujúcim postupom kompilácia.

[1] http://rpmfusion.org/

4.1.2 Postup kompilácie

V stiahnutom balíku sú zdrojové kódy všetkých modulov OpenCV a všetko ostatné, čo ku kompilácii treba.
Postup sa dá rozložiť na jednotlivé príkazy a zhrnúť do týchto bodov:

1. Vytvoríme niekde na disku adresár s názvom `opencv`. Do nej umiestnime stiahnutý súbor a rozbalíme pomocou príkazu: `tar xvfj OpenCV*`
2. Po rozbalení prejdeme do nového adresára a vytvoríme potrebné miesto na kompiláciu:

   ```
   cd OpenCV*
   mkdir debug
   cd debug
   ```

3. Nachádzame sa v adresári, kam budeme kompilovať zdrojový kód. Kompilačné súbory pripravíme pomocou príkazu `cmake`:

   ```
   cmake -D WITH_TBB=ON -D BUILD_EXAMPLES=ON -D
   CMAKE_BUILD_TYPE=DEBUG -D CMAKE_INSTALL_PREFIX=/usr/local ..
   ```

   Pripravíme týmto kompiláciu s použitím balíka `TBB`, skompilujeme aj vzorové programy, v štýle `DEBUG` a všetko nainštalujeme do adresára `/usr/local`.
4. Spustíme kompiláciu (pozri prílohu D). Proces kompilácie potrvá relatívne dlho, v závislosti od výkonnosti počítača:

   ```
   make
   ```

5. Nakoniec prekopírujeme skompilovanú knižnicu do inštalačných adresárov. Na túto operáciu treba práva systémového administrátora, teda `root`. Dá sa to spraviť dvoma spôsobmi:
   - `sudo make install` – Pokiaľ máme nainštalovaný a správne nakonfigurovaný balík `sudo`
   - `su -c "make install"`
6. Správnu inštaláciu overíme príkazom:

   ```
   which opencv_traincascade
   ```

   Ten by mal vyprodukovať následujúci výstup:

   ```
   /usr/local/bin/opencv_traincascade
   ```

Ak všetko prebehlo v poriadku, máme úspešne nainštalovanú knižnicu OpenCV.

4.2 Tréning kaskády

Na detekciu objektov sme používali kaskádu klasifikátorov, ktorú je treba najprv pomocou vzoriek vytrénovať. Postup je podobný aj pre HAAR aj LBP. Celý tento tréning je zľahčený vďaka programom, ktoré spravila komunita OpenCV na zjednodušenie trénovania – `opencv_createsamples` a `opencv_traincascade`. V tejto časti by sme chceli opísať celý postup trénovania, od prípravy až po samotné parametre tréningu.

4.2.1 Príprava vzoriek

Zjednodušene, vzorka v oblasti detekcií objektov znamená obrázok (nemýliť s abstraktnejším pojmom „obraz“ znamenajúci obrázok, ale bez závislosti na formáte alebo kódovaní). OpenCV dokáže pracovať so širokou škálou formátov a kódovaní obrázkov. Vzorky sa pri tréningu delia na dva druhy: pozitívne a negatívne. Negatívne neobsahujú daný detekovaný predmet, pozitívne ho obsahujú.

Pozitívne vzorky sme získali pomocou natáčania premávky na náhodne zvolenej ceste. Všetky sú z 35 minútového videa nafilmovaného na digitálnej zrkadlovke Nikon D3100 dňa 8. decembra 2012 v rozlíšení 1920×1080 a s počtom snímok za sekundu 30. Z týchto záberov sme pomocou programu OpenShot[2] a Kdenlive[3] vystrihli tie časti, kde autá neboli, alebo kde išli v opačnom smere. Tie, ktoré ostali, sme previedli do formátu mp4 a rozlíšenia 1280×720 s rovnakým počtom snímok za sekundu, 30.

Pomocou programu `ffmpeg` sme video rozstrihali na jednotlivé obrázky vo formáte bmp (bitmap) a rozlíšení 852×480. Bitmapový format volíme kvôli jeho bezstratovosti a nekomprimovanosti. Na prevod použijeme príkaz:

```
ffmpeg -i video_pos.mp4 -r 10 -s hd480 pos-%d.bmp
```

Parameter `-r 10` udáva, že program vyberie z daného videa iba 10 snímok za sekundu aj keď vstupné video má 30. Výstup tohto programu je n bitmapových obrázkov zo vstupného videa s dĺžkou v sekundách $d = 100s$ pri parametri $r = 10$ platí:

$$n = d \cdot r$$
$$n = 100 \cdot 10$$
$$n = 1000$$

Z daného videa sme teda získali 1000 obrázkov, ktoré sme použili ako pozitívne vzorky pre tréning.

Negatívne vzorky sme získali od autora `sonots@gmail.com`, ktorý ich uverejnil na stránke `https://code.google.com/p/tutorial-haartraining/`. Negatívnych obrázkov

[2] http://www.openshotvideo.com/
[3] http://www.kdenlive.org/

tu bolo viac než 3000, no my sme odhadom potrebovali len 2000. Vytriedili a vymazali sme preto všetky fotografie, na ktorých by sa len vzdialene mohli nachádzať objekty podobné autám. Pomer pozitívnych ku negatívnym vzorkám by mal byť asi 1:2.

4.2.2 Opisovanie vzoriek

Vytvoríme opisný súbor `positive_info.txt`. Obe metódy HAAR aj LBP potrebujú na tréning označenie pozitívnych vzoriek – aby vedeli kde na obraze sa hľadaný objekt nachádza. Premiestnime opisný súbor do adresára s pozitívnymi vzorkami. Na označovanie vzoriek použijeme program ObjMarker[4]:

```
./ObjMarker positive_info.txt .
```

Autá sa označujú myšou do červeného obdĺžnika, výber potvrdíme stlačením medzerníka (pozri prílohu E). Toto opakujeme pre všetky autá na obraze. Pokiaľ chceme trénovať autá iba z protismeru, tak označujeme samozrejme len tie, kde je auto otočené smerom k nám. Po označení všetkých áut potvrdíme stlačením klávesy „B“, parametre označenia aktuálneho obrazu sa uložia do opisného súboru a načíta sa ďalší obraz. Toto opakujeme pre všetky pozitívne vzorky.

Opisný súbor má nasledujúcu syntax:

```
img.bmp n x y šírka výška
```

Každý riadok opisného súboru začína názvom súboru. Prvé číslo značí počet označených objektov. Nasledujúce série štyroch čísel sú parametre daného označeného objektu. Prvé dve čísla (x, y) sú súradnice ľavého horného rohu obdĺžnika označujúceho objekt. Ďalšie dve čísla sú šírka a výška daného obdĺžnika. Ak je obdĺžnikov viac, pridá sa ďalšia séria štyroch čísel a počet označených objektov sa zvýši o 1.

Všetky pozitívne vzorky, spolu s opisným textovým súborom, sme premiestnili do adresára `pos`, všetky negatívne do adresára `neg`. V adresári `neg` bolo potrebné vytvoriť zoznam negatívnych vzoriek príkazom `ls > negative.txt` a vymazať z neho záznam s menom daného zoznamu.

4.2.3 opencv_createsamples

Označené pozitívne vzorky zadáme ako vstup do programu `opencv_createsamples`, ktorý je štandartnou súčasťou knižnice OpenCV. Spustíme ho s nasledujúcimi parametrami:

```
opencv_createsamples -info pos/positive_info.txt -vec vecfile.vec
-num 1000 -w 24 -h 16
```

[4] Autor: Achu Wilson – achu_wilson@rediffmail.com, `http://achuwilson.wordpress.com`

Program si zoberie vstupné informácie z opisného súboru `pos/positive_info.txt` a vytvorí 1000 vzoriek so šírkou 24 pixelov a výškou 16 pixelov. Vytvorené vzorky zapíše do „vektorového“ súboru `vecfile.vec`. Tento program má mnoho ďalších vstupných parametrov a možností, o ktorých sa dozviete, ak ho spustíte bez parametrov.

4.2.4 opencv_traincascade

Po úspešnom vytvorení vektorového súboru začíname trénovanie. Tento proces zabezpečuje program `opencv_traincascade`, taktiež súčasť OpenCV. Zoznam parametrov je tu relatívne dlhý:

```
opencv_traincascade -data haar -vec vecfile.vec -numPos 1000
-numNeg 2000 -numStages 30 -precalcValBufSize=1024
-precalcIdxBufSize=1024 -featureType HAAR -mode ALL -w 24 -h 16
```

Definície parametrov trénovacieho programu:

- `-data haar` – Definuje adresár `haar` ako miesto, kam sa ukladajú všetky výsledky tréningu
- `-vec vecfile.vec` – Vzorky čerpá z vektorového súboru `vecfile.vec`
- `-numPos 1000 -numNeg 2000` – Počet pozitívnych a negatívnych vzoriek
- `-numStages 30` – Počet etáp trénovania kaskády (stupeň trénovania)
- `-precalcValBufSize=1024` – Veľkosť vyrovnávacej pamäte pre predkalkulované hodnoty čŕt
- `-precalcIdxBufSize=1024` – Veľkosť vyrovnávacej pamäte pre predkalkulované indexy čŕt – Čím viac pamäte, tým rýchlejší je tréningový proces
- `-featureType HAAR` – Spôsob detekcie objektov je HAAR (v druhej kaskáde sem nastavíme hodnotu LBP)
- `-mode ALL` – Množina, z ktorej tréning vyberá detekované črty – BASIC sú len vzpriamené črty, ALL obsahuje okrem vzpriamených aj otočené o 45°
- `-w 24 -h 16` – Poukazujú na to, že vzorky vo vektorovom súbore majú šírku 24px a výšku 16px

Pri trénovaní našich kaskád `t6` a `t7` sme použili horeuvedené parametre, no pre krátkosť času sme počet etáp znížili na 20.

4.3 Výpočet rýchlosti

Náš návrh pracuje na jednoduchom princípe detekcie objektu v danej meracej oblasti. Vstupné testované video rozstriháme na jednotlivé snímky, podobne ako v kapitole 4.2.1. Zvolíme číslo f ako počet vybratých snímok za sekundu (dôležité je, aby sme

toto číslo používali neskôr vo všetkých výpočtoch, nie počet snímok za sekundu vstupného videa). Uložíme ich ako bitmapové obrázky do testového adresára `test`. Zvolíme meraciu oblasť, ktorá môže byt lubovoľný rovinný útvar nachádzajúci sa na všetkých obrazoch a musíme poznať jeho reálnu dĺžku.

Obrázky zadáme ako vstup do nášho programu na triedenie obrazov na pozitívne a negatívne, potom do programu na triedenie obrazov na jednotlivé „unikátne“ autá. Pre každé „unikátne“ auto potom počítame rýchlosť individuálne. Obrázky auta porovnávame s meracou oblasťou. Zhodné dáme do adresára `test/pos/cars/CarXX/inside/`. Samotný výpočet robíme dvoma spôsobmi, s ktorými sa oboznámime v ďalšej kapitole.

4.3.1 Teória výpočtu

Na výpočet používame dve jednoduché metódy:

1. Uvažujme počet obrázkov n, na ktorých sa auto nachádza v meracej oblasti, počet vybratých snímok za sekundu f a reálna dĺžka meracej oblasti l (v metroch). Počítame rýchlosť v v kilometroch za hodinu (konštanta 3,6 označuje prevod medzi metrami za sekundu a kilometrami za sekundu):

$$v = l \cdot \frac{n}{f} \cdot 3{,}6$$

2. Obrázky auta, kde sa auto nachádza v meracej oblasti, majú názov v tvare `test-x.bmp`, kde `x` reprezentuje kladné celé číslo. Zoberieme z daného adresára obrázok s najmenším (n_{min}) a obrázok s najväčším (n_{max}) číslom. Počítame rýchlosť v v kilometroch za hodinu (konštanta 3,6 označuje prevod medzi metrami za sekundu a kilometrami za sekundu):

$$v = l \cdot \frac{n_{max} - n_{min}}{f} \cdot 3{,}6$$

4.4 Náš program CarCV

Na uskutočnenie rozpoznávania áut a výpočtu ich rýchlosti sme naprogramovali program nazývaný CarCV. Program je v programovacom jazyku C++, využíva OpenCV a Boost Filesystem knižnice. Skladá sa zo štyroch základných „metód“, ktoré sú opísané v nasledujúcich podkapitolách.

Program berie pri spúšťaní nasledovné argumenty:

- `--cascade` – Zadáme xml súbor, v ktorom sme zapísali vytrénovanú kaskádu.
- `--method` – Jedna, popr. viac z nasledovných: `DETECTSORT`, `SORTUNIQUE`, `INSIDE` a `SPEED`. Každú jednotlivú metódu musíme ukončiť znakom `+`, metódy nemá zmysel opakovať. Napríklad:

`--method=prva_metoda+druha_metoda+tretia_metoda+`

- `--speedbox` – Zadáme štyri čísla (`x+y+width+height+`) opisujúce obdĺžnik na meranie rýchlosti. Súradnice polohy ľavého horného rohu `x`, `y` a šírku a výšku obdĺžnika `width` a `height`. Dodržiavame správne poradie týchto čísel.
- `--list` – Poloha zoznamu so vstupnými obrázkami.
- `--posdir` – Adresár, kam program uloží pozitívne obrazy.
- `--negdir` – Adresár, kam program uloží negatívne obrazy.
- `--cardir` – Adresár, kam program uloží obrazy detekovaných unikátnych áut.
- `--insidedir` – Adresár, kam program uloží obrazy unikátnych áut, ktoré sa nachádzajú vnútri obdĺžnika na meranie rýchlosti.

4.4.1 Detekcia a roztriedenie

Názov metódy: **DETECTSORT**

Táto metóda má na starosti roztriedenie vstupných obrázkov na pozitívne a negatívne. Pomocou knižnice OpenCV a kaskády zistí, či sa na danom obrázku nachádza auto. Ak áno, zaradí ho medzi pozitívne obrazy, ak nie, tak medzi negatívne. Obidve skupiny nakoniec uloží do ich zložiek.

4.4.2 Triedenie unikátnych áut

Názov metódy: **SORTUNIQUE**

Metóda triedi a kopíruje unikátne autá do adresára `cardir`. Pod pojmom unikátne auto sa rozumie zoznam obrazov fyzicky existujúceho auta. Pre všetky obrazy daného zoznamu platí, že sa na nich dané auto nachádza.

Vstupné parametre tejto metódy sú zoznam obrazov, na ktorých sa nachádza aspoň jedno auto, a minimálnu pravdepodobnostnú konštantu (kedy považuje auto na triedenom obraze za totožné s autom na obraze už zatriedenom).

Prvý obrázok v zozname je zaradený ako unikátne auto `Auto0` (pri ukladaní sa vytvára rovnomenný adresár). Unikátne auto môže obsahovať viacero obrazov. Každý ďalší obrázok prechádza algoritmom triedenia. Algoritmus sa dá popísať takto:

1. Pre každý obraz v `Auto0` vypočítame pravdepodobnosť či sa auto na triedenom obraze zhoduje s autom na danom obraze.
2. Vypočítame aritmetický priemer pravdepodobností všetkých obrazov v `Auto0`. Túto hodnotu považujeme za pravdepodobnosť, že sa triedený obraz zhoduje s fyzickým autom, ktoré sa nachádza na obrazoch v `Auto0`.
3. Body 1-2 opakujeme pre všetky už zatriedené unikátne autá `Auto0, Auto1, Auto2,...,AutoN`

4. Pre triedený obraz máme voči každému už zatriedenému unikátnemu autu vypočítanú pravdepodobnosť zhodnosti. Nájdeme najväčšiu hodnotu pravdepodobnosti, a ak je väčšia, nanajvýš rovná, minimálnej pravdepodobnostnej konštante (vstupný parameter metódy triedenia), tak zaradíme daný triedený obraz k unikátnemu autu, ktorého hodnota pravdepodobnosti to je. Ak táto hodnota pravdepodobnosti je menšia ako minimálna pravdepodobnostná konštanta, vytvoríme nové unikátne auto a zaradíme doň triedený obraz.
5. Body 1-4 opakujeme, kým nie sú zatriedené všetky obrazy.

Výpočet pravdepodobnosti zhodnosti pre dané dva obrazy A,B prebieha takto:

1. Detekujeme auto na oboch obrazoch. Dostaneme body, ktorých spojením získame hranicu (obrys) auta.
2. Obraz A zostrihneme podľa bodov hranice auta a zmenšíme na $x\%$. Označíme ho C.
3. Zostrihnutý a zmenšený obraz pomocou algoritmu na výpočet zaujímavých bodov SURF a porovnávača bodov FLANN porovnáme s obrazom B. Dostaneme zoznam dvojrozmerných bodov.
4. Ak je počet bodov menší ako 4, pripočítame 1 iba k počtu všetkých a preskakujeme až na ďalšie opakovanie (bod 6). Ak nie je menší, vypočítame z daných bodov obdĺžnik.
5. Vypočítame obsah obrazu C (S_C) a obsah obdĺžnika vytvoreného v bode 4 (S_B). Tieto dva obsahy porovnáme s toleranciou t.

   $$S_C \in \langle S_B \pm t \rangle \Rightarrow C \equiv B$$

   Toto ohodnotenie má výstup buď zhodnosť alebo nezhodnosť. Ak sú zhodné zvýšime počet zhodných o 1 a počet všetkých o 1. Ak nie sú zhodné zvýšime o 1 iba počet všetkých.
6. Tento postup (body 2-5) opakujeme pre všetky $x\%$ v danom rozsahu. Experimentálne sme zistili, že najvhodnejší rozsah je interval $\langle 85, 90 \rangle$.
7. Potom rovnakým spôsobom zostrihávame a porovnávame obraz B s obrazom A (vymenili sme poradie) a tiež opakujeme pre všetky hodnoty $x\%$ v danom rozsahu.
8. Nakoniec ako pravdepodobnosť zhodnosti obrazov A,B vyhlásime:

   $$P_{A,B} = \frac{\textit{počet zhodných}}{\textit{počet všetkých}}$$

4.4.3 Poloha auta v meracom obdĺžniku

Názov metódy: **INSIDE**

Metóda porovnáva obdĺžnik ohraničujúci detekované auto a obdĺžnik na meranie rýchlosti. Ak existuje bod patriaci obdĺžniku s autom, ktorý zároveň patrí obdĺžniku na meranie rýchlosti, je tento obraz auta zaradený medzi obrazy, ktoré budú použité na výpočet rýchlosti.

Na výpočet či existuje aspoň jeden bod patriaci obom obdĺžnikom je použitý jednoduchý algoritmus, založený na úvahe pretínania sa okrajov obdĺžnikov. Vypočítame súradnice bodov, ktoré tvoria prvý a posledný riadok a prvý a posledný stĺpec, pre oba obdĺžniky. Ak sa aspoň jeden takto vypočítaný bod zhoduje s bodom takto vypočítaným pre druhý obdĺžnik, je vyslovený záver, že sa dané auto nachádza v obdĺžniku na meranie rýchlosti.

4.4.4 Výpočet rýchlosti

Názov metódy: **SPEED**

Metóda výpočtu rýchlosti sa zaoberá praktickým prevedením dvoch uvažovaných výpočtových metód z kapitoly 4.3.1. Na výpočet potrebuje špecifikovať výpočtovú metódu, počet snímok za sekundu vstupného videa a reálnu dĺžku obdĺžnika na meranie rýchlosti. Operuje na obrázkoch v adresári `insidedir` a podľa zadaného typu výpočtovej metódy počíta buď metódou rozdielu extrémov poradových čísel obrázkov alebo metódou založenou na počte obrázkov v adresári. Obidve tieto metódy už boli definované v kapitole 4.3.1. Nakoniec vypíše rýchlosti jednotlivých áut v kilometroch za hodinu.

4.5 Testovanie úspešnosti CarCV

Program CarCV by sa dal v praxi použiť ako statický detektor rýchlostí áut. V tejto kapitole skúsime aplikovať našu prácu presne na tento scenár. Budeme sa zaoberať aj vyhodnotením výsledkov dosiahnutých v týchto testoch. CarCV je však momentálne stále vo vývoji, preto sa obmedzíme len na testovanie metódy DETECTSORT.

4.5.1 Testové vzorky

Na testovanie použijeme dve vzorky. Sú to videá (resp. jednotlivé snímky videa) natočené rovnakým spôsobom ako vzorky v kapitole 4.2.1. Konkrétne testové vzorky v našich testoch sú:

- **Trénovacie obrazy** – Video zostrihané z videí použitých ako trénovacie vzorky na vytrénovanie kaskád.
- **Porovnávacia vzorka** – Video natočené 20. januára 2013 na rovnakom mieste ako trénovacie vzorky, avšak autá na ňom sú iné ako tie na trénovacích vzorkách.

Taktiež prostredie: pribudol sneh, zmena farby aj intenzity svetla.

4.5.2 Testové kaskády

Pre účely tohoto testu sme vytrénovali 3 kaskády: *t5*, *t6* a *t7*. Ich parametre sú zosumarizované v priloženej tabuľke (príloha H). Spôsobom trénovania kaskád sa zaoberá kapitola 4.2.

4.5.3 Postup testov

Pre každú vzorku prebiehajú testy následovne:

1. Použitím metódy DETECTSORT roztriedime všetky snímky na pozitívne alebo negatívne. Vypočítame efektivitu rozlišovania danej kaskády.
2. Spočítame, koľkokrát program zle zaradil vzorky (pozitívne ako negatívne a negatívne ako pozitívne) a koľkokrát dobre (pozitívne ako pozitívne, negatívne ako negatívne).
3. Vyhodnotíme pomocou štatistických postupov.

Všetky kaskády otestujeme individuálne.

4.5.4 Štatistické vyhodnotenie

Výsledky testov pre trénovaciu vzorku a porovnávaciu vzorku sú priložené v tabuľkách (prílohy I a J).

Pri skúmaní výsledkov individuálne v rámci vzorky vidíme, že počet vstupných vzoriek je priamo úmerný úspešnosti danej kaskády. V prípade porovnávacej vzorky je to nárast o 28,5% pri zvýšení počtu vzoriek 10×. Kaskáda *t6* bola lepšia asi o 8% aj od kaskády *t7*, ktorá mala rovnaký počet vstupných vzoriek. Kaskáda *t7* však bola vytrénovaná pomocou LBP, zatiaľ čo *t5* a *t6* pomocou HAAR. Všetky údaje o trénovaní kaskád nájdete v priloženej tabuľke (príloha H). Môžeme z toho vyvodiť záver, že aj keď LBP je rýchlejší pri trénovaní ako HAAR, jeho výsledky sú menej spoľahlivé.

Tieto fakty podčiarkujú aj výsledky pre trénovaciu vzorku. Kaskáda *t6* bola zase suverénne najlepšia, o 10% lepšia ako *t7* a o 31% lepšia ako *t5*.

Ako pozitívna správa sa javí výkonnosť kaskád pri zaraďovaní negatívnych vzoriek. Všetky tri kaskády mali v tomto obore úplne rovnaké výsledky. 100% úspešnosť pri zaraďovaní negatívnych obrazov ako negatívne a 0% úspešnosť pri zaraďovaní negatívnych do pozitívnych. Obidva výsledky sú ideálne pre ďalšie fungovanie metód programu CarCV.

Ak porovnáme výsledky medzi porovnávacou a trénovacou vzorkou, zistíme, že kaskáda *t6* mala oveľa menšie ťažkosti rozoznať „známe“ testovacie vzorky (72,5%), zatiaľ čo porovnávacia vzorka predstavovala pre ňu veľký problém (iba 34,5%).

Na základe týchto údajov by sa dala vysloviť hypotéza ideálnej kaskády: Na jej vytvorenie použijeme čo najviac a čo najrôznorodejšie vzorky, použijeme trénovaciu metódu HAAR a necháme bežať na čo najviac etáp.

5 Záver

Na záver by sme chceli podotknúť, že sme si vedomí istých limitov našej implentácie a hlavne jej nedokončenosti. Avšak, pri dodatočnej optimalizácii a lepšej prepracovanosti si myslíme, že má potenciál.

Najväčšie obmedzenie bola úroveň natrénovania detekčného algoritmu, respektíve jeho (ne)schopnosť rozoznávať autá. Aj pri našom najlepšom pokuse dosiahol úspešnosť iba 34,5%. To znamenalo, že sa kvôli tomu nedali ďalej vykonať metódy, ktoré by postupne viedli až ku výpočtu rýchlosti. Ak by sme vyvinuli algoritmus, ktorý by eliminoval tieto chyby a zároveň vytrénovali lepšiu kaskádu, vedeli by sme úspešnosť dramaticky zvýšiť.

Postupnou evolúciou technológie by sa dali porovnateľné, ba aj lepšie výsledky dosiahnuť aj pomocou úplne triviálnych kamier a prístrojov. Raz bude možné spraviť rovnaký rozbor ako v tejto práci pomocou bežného mobilného telefónu so zabudovaným fotoaparátom.

Ďalšia vec, ktorá samozrejme obmedzuje výkonnosť implentácie, je hardvérová výkonnosť počítača, ktorý ju vykonáva. Program aj napriek žiadnej optimalizácii nemá vysoké nároky na pamäť, zato oveľa vyššie na výkon procesora. Myslíme si, že pri detailne prepracovanej optimalizácii algoritmov a správy pamäte, by sa dali tieto požiadavky znížiť minimálne o 50%. Taktiež sa dá tento problém potlačovať využitím moderných technológií cloud computingu.

Pri aplikácii v praxi vzniká predbežným odhadom intrigujúca hypotéza: Keby sme do každého mesta v Slovenskej republike dali dve krabičky zostrojené na detekciu rýchlosti vozidiel týmto spôsobom a prenajali zostavu serverov, dokázal by tento systém pokryť svoje náklady už pri *dvoch* vydaných pokutách s najnižšou sadzbou ročne vzhľadom na jednu krabičku. Všetky ostatné pokuty už by boli ziskové a skončili by v štátnej pokladnici. Ani nehovoriac o tom, že by sa eventuálne dali pridať aj schopnosti rozpoznávania iných priestupkov, ako napríklad nebezpečná jazda, jazda v protismere, atď. Pridanie funkcionality, úpravy alebo aktualizácie by sa dali samozrejme robiť bez fyzickej prítomnosti a synchronizovane.

Zhrnutie

Táto práca sa zaoberá problematikou počítačového videnia, rozpoznávaním objektov a výpočtom rýchlosti ich pohybu. V našom prípade sa jedná o rozpoznávanie áut. Práca pozostáva z dvoch väčších častí.

V teoretickej časti opisujeme oblasť počítačového videnia a knižnicu OpenCV, ktorú ďalej často využívame. Metódy a algoritmy detekcie objektov sú detailne a jednoducho rozobrané.

V praktickej časti využívame poznatky z teoretickej časti a aplikujeme ich na problém výpočtu rýchlosti áut. Pomocou vlastného programu CarCV uskutočňujeme sériu testov, kde namiesto konvenčných metód na zistenie rýchlosti využívame počítačové videnie. Na záver navrhujeme lacné a jednoduché riešenie problému kontroly dodržiavania maximálnej povolenej rýchlosti.

Resumé

This work deals with the computer science field of computer vision, object detection and calculating the speed of objects' movement. In our case the objects we deal with are cars. Our work consists of two major parts.

In the theoretical part we describe the field of computer vision and a computer vision library, OpenCV, which we use quite often throughout our work. Methods and algorithms related to object detection are simplistically explained, but into relative detail.

In the practical part we use our knowledge from the theoretical part and apply it onto the problem of calculating speed of cars. With the help of our program CarCV, we issue a series of tests, where instead of conventional methods for calculating the speed of a car we use computer vision. Finally, we introduce a simple and cheap solution for enforcing the obeying of speed limits based on the results of our work.

Literatúra

[1] Ballard, D. H. - Brown, C. M.: Computer Vision. Englewood Cliffs, New Jersey: Prentice-Hall, 1982, ISBN 0-13-165316-4.

[2] Bay, H. - Ess, A. - Tuytelaars, T. - Goo, L. V.: SURF: Speeded Up Robust Features. Computer Vision and Image Understanding (CVIU), zv. 110(3); s. 346–359, 2008, ISSN 1077-3142, `http://www.vision.ee.ethz.ch/~surf/`.

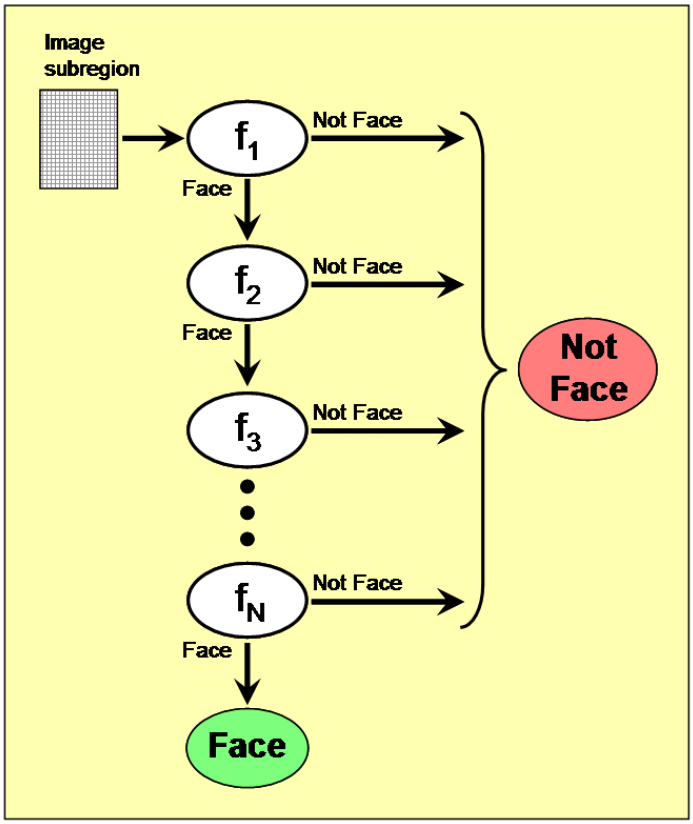
[3] Ftáčnik, M.: Počítačové videnie - Úvod do problematiky. Department of Applied Informatics, Faculty of Mathematics, Physics and Informatics, Comenius University, 2011, `http://fractal.dam.fmph.uniba.sk/~ftacnik/compvis.htm`.

[4] Pietikäinen, M.: Local Binary Patterns. Scholarpedia, zv. 5(3); s. 9775, 2010, `http://www.scholarpedia.org/article/Local_Binary_Patterns`.

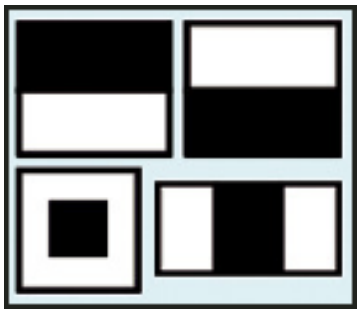
[5] Szeliski, R.: Computer Vision: Algorithms and Applications. None: Springer, 2010, ISBN 978-1-84882-934-3.

[6] Viola, P. - Jones, M.: Rapid Object Detection using a Boosted Cascade of Simple Features. Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), zv. 1; s. 511–519, 2001, ISSN 1063-6919.

Prílohy



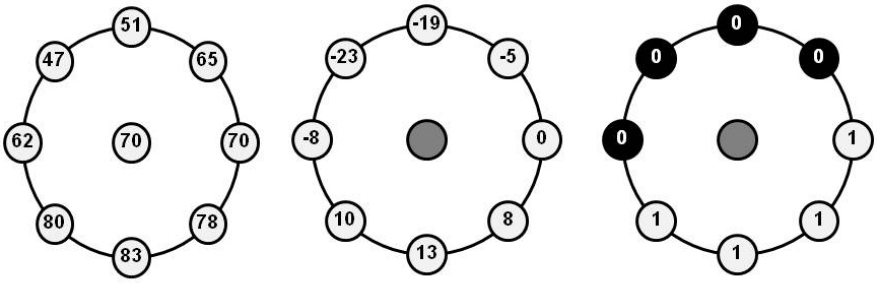
Príloha A: Diagram kaskády klasifikátorov



Príloha B: Haar-like črty

The value of the LBP code of a pixel (x_c, y_c) is given by:

$$LBP_{P,R} = \sum_{p=0}^{P-1} s(g_p - g_c)2^p \qquad s(x) = \begin{cases} 1, if\ x \geq 0; \\ 0, otherwise. \end{cases}$$



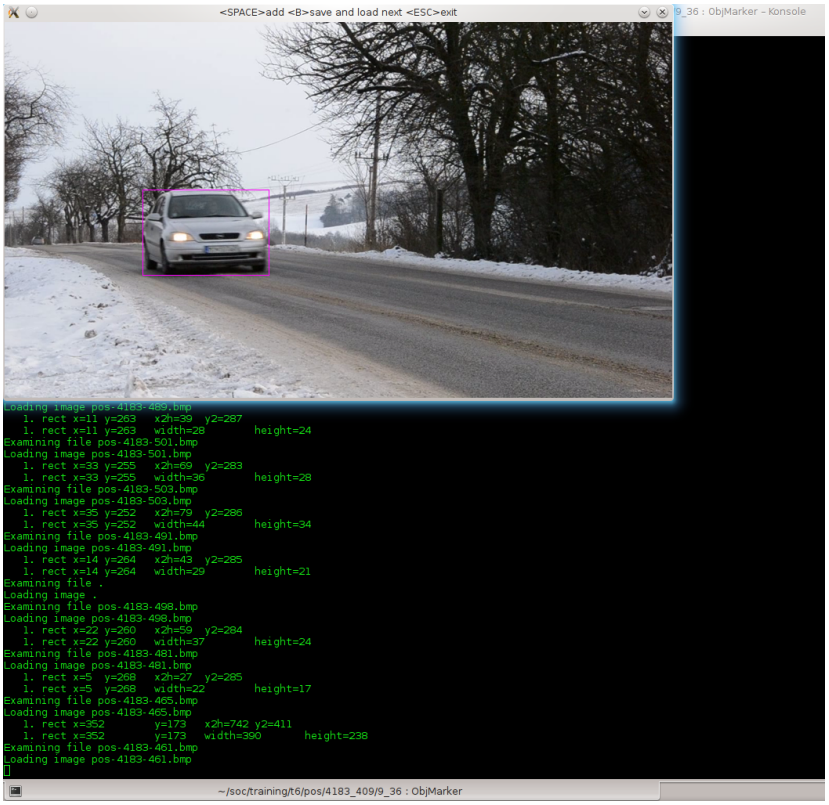
1*1 + 1*2 + 1*4 + 1*8 + 0*16 + 0*32 + 0*64 + 0*128 = 15

4. Multiply by powers of two and sum

Príloha C: Princíp kalkulácie hodnoty pixelu pomocou LBP

```
[  6%] Building CXX object 3rdparty/openexr/CMakeFiles/IlmImf.dir/IlmImf/ImfStringAttribute.cpp.o
[  6%] Building CXX object 3rdparty/openexr/CMakeFiles/IlmImf.dir/IlmImf/ImfStdIO.cpp.o
[  6%] Building CXX object 3rdparty/openexr/CMakeFiles/IlmImf.dir/IlmImf/ImfIntAttribute.cpp.o
[  6%] Building CXX object 3rdparty/openexr/CMakeFiles/IlmImf.dir/IlmImf/ImfHeader.cpp.o
[  6%] Building CXX object 3rdparty/openexr/CMakeFiles/IlmImf.dir/IlmImf/ImfTileDescriptionAttribute.cpp.o
[  6%] Building CXX object 3rdparty/openexr/CMakeFiles/IlmImf.dir/IlmImf/ImfChromaticitiesAttribute.cpp.o
[  6%] Building CXX object 3rdparty/openexr/CMakeFiles/IlmImf.dir/IlmImf/ImfStandardAttributes.cpp.o
[  7%] Building CXX object 3rdparty/openexr/CMakeFiles/IlmImf.dir/IlmImf/ImfRational.cpp.o
[  7%] Building CXX object 3rdparty/openexr/CMakeFiles/IlmImf.dir/IlmImf/ImfPreviewImage.cpp.o
[  7%] Building CXX object 3rdparty/openexr/CMakeFiles/IlmImf.dir/IlmImf/ImfKeyCodeAttribute.cpp.o
[  7%] Building CXX object 3rdparty/openexr/CMakeFiles/IlmImf.dir/IlmImf/ImfRationalAttribute.cpp.o
[  7%] Building CXX object 3rdparty/openexr/CMakeFiles/IlmImf.dir/IlmImf/ImfEnvmap.cpp.o
[  7%] Building CXX object 3rdparty/openexr/CMakeFiles/IlmImf.dir/IlmImf/ImfPizCompressor.cpp.o
[  7%] Building CXX object 3rdparty/openexr/CMakeFiles/IlmImf.dir/IlmImf/ImfThreading.cpp.o
[  7%] Building CXX object 3rdparty/openexr/CMakeFiles/IlmImf.dir/IlmImf/ImfHuf.cpp.o
Linking CXX static library ../lib/libIlmImf.a
[  7%] Built target IlmImf
[  7%] Generating opencv_core_pch_dephelp.cxx
Scanning dependencies of target opencv_core_pch_dephelp
[  8%] Building CXX object modules/core/CMakeFiles/opencv_core_pch_dephelp.dir/opencv_core_pch_dephelp.cxx.o
Linking CXX static library ../../lib/libopencv_core_pch_dephelp.a
[  8%] Built target opencv_core_pch_dephelp
Scanning dependencies of target pch_Generate_opencv_core
[  8%] Generating precomp.hpp
[  9%] Generating precomp.hpp.gch/opencv_core_DEBUG.gch
[  9%] Built target pch_Generate_opencv_core
Scanning dependencies of target opencv_core
[  9%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/matrix.cpp.o
[  9%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/stat.cpp.o
[  9%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/persistence.cpp.o
[  9%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/parallel.cpp.o
[  9%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/arithm.cpp.o
[  9%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/precomp.cpp.o
[ 10%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/array.cpp.o
[ 10%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/out.cpp.o
[ 10%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/copy.cpp.o
[ 10%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/rand.cpp.o
[ 10%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/tables.cpp.o
[ 10%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/system.cpp.o
[ 10%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/datastructs.cpp.o
[ 10%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/convert.cpp.o
[ 10%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/matop.cpp.o
[ 10%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/lapack.cpp.o
[ 11%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/alloc.cpp.o
[ 11%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/algorithm.cpp.o
[ 11%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/opengl_interop.cpp.o
[ 11%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/matmul.cpp.o
[ 11%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/mathfuncs.cpp.o
[ 11%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/gpumat.cpp.o
[ 11%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/dxt.cpp.o
[ 11%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/cmdparser.cpp.o
[ 11%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/drawing.cpp.o
Linking CXX shared library ../../lib/libopencv_core.so
[ 11%] Built target opencv_core
[ 11%] Generating opencv_ts_pch_dephelp.cxx
Scanning dependencies of target opencv_ts_pch_dephelp
[ 11%] Building CXX object modules/ts/CMakeFiles/opencv_ts_pch_dephelp.dir/opencv_ts_pch_dephelp.cxx.o
Linking CXX static library ../../lib/libopencv_ts_pch_dephelp.a
[ 11%] Built target opencv_ts_pch_dephelp
Scanning dependencies of target pch_Generate_opencv_ts
[ 11%] Generating precomp.hpp
[ 11%] Generating precomp.hpp.gch/opencv_ts_DEBUG.gch
```

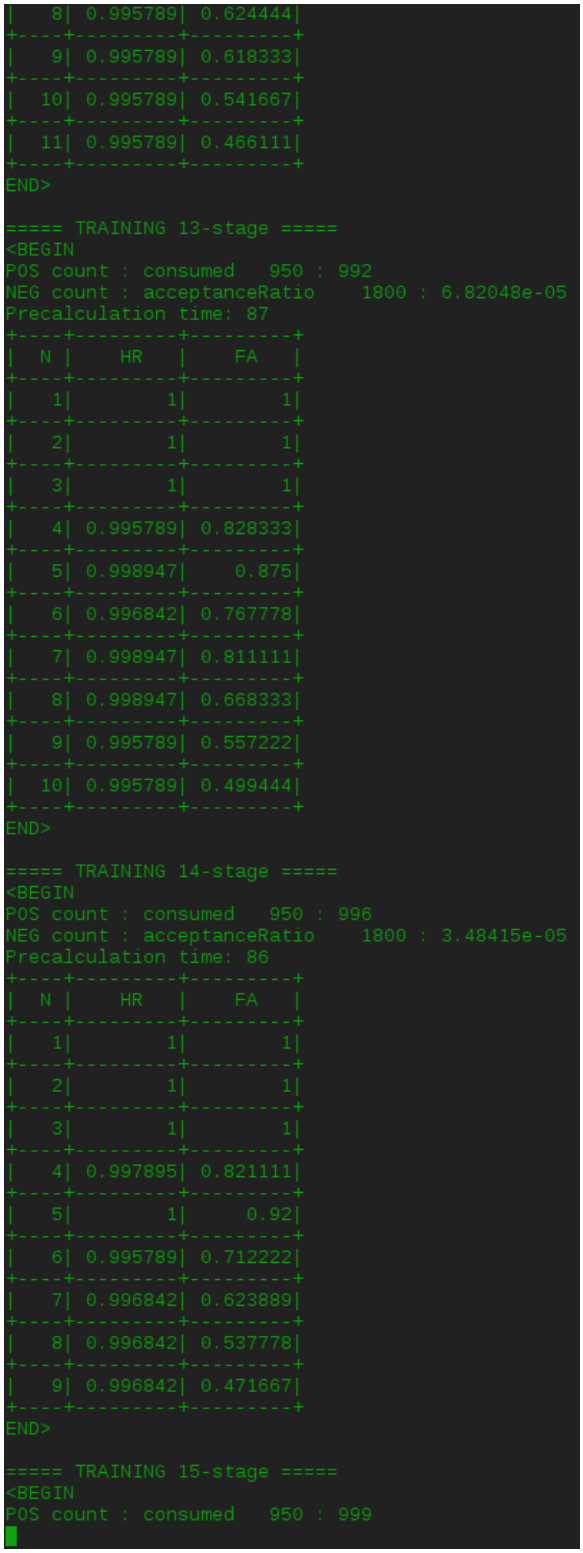
Príloha D: Kompilácia OpenCV



Príloha E: Označovanie objektov

```
-------------------------------------------------------
mkdir: created directory `build'



-------------------------------------------------------
-------COPYING SOURCE FILES OVER
-------------------------------------------------------
`/home/odenkos/c_workspace/CarProject/cardetect/carcv.cpp' -> `./carcv.cpp'
`/home/odenkos/c_workspace/CarProject/cardetect/carcv.hpp' -> `./carcv.hpp'
`/home/odenkos/c_workspace/CarProject/cardetect/carimg.cpp' -> `./carimg.cpp'
`/home/odenkos/c_workspace/CarProject/cardetect/carimg.hpp' -> `./carimg.hpp'
`/home/odenkos/c_workspace/CarProject/cardetect/det.cpp' -> `./det.cpp'
`/home/odenkos/c_workspace/CarProject/cardetect/det.hpp' -> `./det.hpp'
`/home/odenkos/c_workspace/CarProject/cardetect/match.cpp' -> `./match.cpp'
`/home/odenkos/c_workspace/CarProject/cardetect/match.hpp' -> `./match.hpp'



-------------------------------------------------------
-------RUNNING CMAKE
-------------------------------------------------------
-- The C compiler identification is GNU 4.7.2
-- The CXX compiler identification is GNU 4.7.2
-- Check for working C compiler: /usr/bin/gcc
-- Check for working C compiler: /usr/bin/gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/odenkos/soc/car_project/build



-------------------------------------------------------
-------RUNNING MAKE
-------------------------------------------------------
Scanning dependencies of target carcv
[ 25%] Building CXX object cardetect/CMakeFiles/carcv.dir/carcv.cpp.o
[ 50%] Building CXX object cardetect/CMakeFiles/carcv.dir/match.cpp.o
[ 75%] Building CXX object cardetect/CMakeFiles/carcv.dir/carimg.cpp.o
[100%] Building CXX object cardetect/CMakeFiles/carcv.dir/det.cpp.o
Linking CXX executable carcv
[100%] Built target carcv



-------------------------------------------------------
------DISTRIBUTING
-------------------------------------------------------
`/home/odenkos/soc/car_project/build/cardetect/carcv' -> `/home/odenkos/soc/car_project/test/all/carcv'
`/home/odenkos/soc/car_project/build/cardetect/carcv' -> `/home/odenkos/soc/car_project/test/all/test/carcv'



-------------------------------------------------------
-------FINISHED
-------------------------------------------------------
[odenkos@sauron car_project]$
```

Príloha F: Kompilácia CarCV



Príloha G: Tréning kaskády pomocou a) HAAR b) LBP

Kaskáda	Algoritmus	Pozitívne vz.	Negatívne vz.	Etapy
t5	HAAR	113	200	25
t6	HAAR	1000	2000	20
t7	LBP	1000	2000	19

Príloha H: Tabuľka parametrov kaskád

Kaskáda	DETECTSORT			
	POS	FPOS	NEG	FNEG
t5	41,39%	0,00%	100,00%	67,90%
t6	72,72%	0,00%	100,00%	31,60%
t7	62,64%	0,00%	100,00%	43,28%

Príloha I: Tabuľka štatistického vyhodnotenia pre trénovaciu vzorku

Kaskáda	DETECTSORT			
	POS	FPOS	NEG	FNEG
t5	6,04%	0,00%	100,00%	93,96%
t6	34,51%	0,00%	100,00%	65,49%
t7	26,57%	0,00%	100,00%	73,43%

Príloha J: Tabuľka štatistického vyhodnotenia pre porovnávaciu vzorku