VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

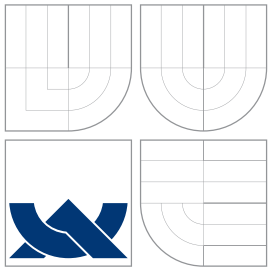
ALGORITMY PRO PODPORU POČÍTAČOVÉHO VIDĚNÍ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

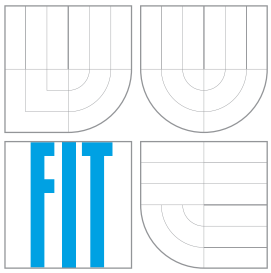
AUTOR PRÁCE
AUTHOR

ANDREJ TRNKÓCI

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

ALGORITMY PRO PODPORU POČÍTAČOVÉHO VIDĚNÍ

ALGORITHMS SUPPORTING COMPUTER VISION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ANDREJ TRNKÓCI

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ZBOŘIL FRANTIŠEK, Ph.D.

BRNO 2008

Abstrakt

Téma tejto práce je implementácia detekujúca narušiteľov v stráženej zóne. Program implementuje počítanie rozdielov v obraze, segmentácia prahovaním a detekcia viaceých objektov.

Klíčová slova

Počítačové videnie, automatický kamerový strážny systém, segmentácia obrazu.

Abstract

The theme of this thesis is implementatation of system detecting intruders in guarded zone. The program implements counting difference frame, segmentating image by thresholding and multiple objects detection.

Keywords

Machine vision, automatic camera guard system, image segmentation.

Citace

Andrej Trnkóci: Algoritmy pro podporu počítačového vidění, bakalářská práce, Brno, FIT VUT v Brně, 2008

Algoritmy pro podporu počítačového vidění

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Františka Zbořila, Ph.D. s použitím uvedenej literatúry.

.....
Andrej Trnkóci
13. května 2008

Poděkování

Touto cestou, by som chcel vyjadriť moje poďakovanie školiteľovi diplomovej práce Ing. Františka Zbořila, Ph.D. za odbornú pomoc a rodine za pomoc pri opravovaní chýb v texte.

Obsah

1 Úvod **2**

2 Použité nástroje **3**
2.1 Inštalácia a príprava nástrojov . . . 3
2.1.1 Ovládače pre webkameru . . . 3
2.1.2 Knižnice OpenCV . . . 4
2.1.3 Preklad programu . . . 4
2.1.4 Alternatíva pre Windows . . . 4

3 Problematika **5**
3.1 Predpríprava obrazu . . . 6
3.2 Segmentácia obrazu . . . 6
3.3 Obnovovanie pozadia . . . 7

4 Implementácia **9**
4.1 Zachytenie a vykreslenie obrazu z kamery. . . . 9
4.2 Funkcie pre ovládanie programu a nastavovanie parametrov . . . 10
4.3 Predpríprava obrazu . . . 10
4.4 Segmentácia obrazu . . . 11
4.4.1 Výpočet obrazu rozdielov . . . 11
4.4.2 Prahovanie rozdielov pre detekciu objektov . . . 12
4.4.3 Rozoznanie oddelených objektov . . . 12
4.4.4 Označenie objektov . . . 14
4.5 Údržba obrazu pozadia . . . 14
4.5.1 Slepé obnovovanie pozadia . . . 14
4.5.2 Metóda čiastočného obnovovania pozadia . . . 14
4.5.3 Obnovovanie pozadia Heikkila and Olli . . . 15

5 Nastavenia programu **16**
5.1 Klávesové skratky . . . 17
5.2 Nastaviteľné premenné programu . . . 17

6 Zhodnotenie výsledkov **20**

7 Námety pre budúcu prácu **22**

8 Záver **23**

Kapitola 1

Úvod

Témou tejto práce je rozpoznanie narušitela v obraze zachytenom z kamery, a sledovanie jeho polohy v obraze. Je to téma typicky patriaca do kategórie počítačové videnie. Táto vedecká disciplína je v súčastnosti veľmi dynamická aj preto, že mnoho problémov je vyriešených len čiastočne keďže naučiť počítač rozumieť obrazu nie je jednoduché aj keď pre človeka je to samozrejmosť. Programi tohto typu nachádzajú v súčastnosti stále širšie uplatnenie v praxi, príkladom môže byť zabezpečenie vonkajších hraníc Európskej únie.

Pri riešení tohoto problému potrebujeme najprv vyriešiť ako získať obraz z webkamery, potom zistiť kde v obraze nastali zmeny dosť významné na to, aby sa dali považovať za nejaký objekt a následné vykreslenie a označenie zisteného objektu vo výstupnom obrázku.

Kapitola 2

Použité nástroje

Kód aplikácie budem písať v jazyku `C`, preklad a testovanie na operačnom systéme `Linux` s prekladačom `gcc`. Tento program by mal byť prenositelný aj na iné operačné systémy keďže som sa rozhodol používať len prenosiťeľné nástroje.

K riešeniu tehto projektu som sa rozhodol použiť open source prenositelnú knižnicu pre počítačové videnie napísanú v jazyku `C++ OpenCV`. Túto knižnicu vyvynula spoločnosť Intel, a obsahuje funkcie pre získanie obrazu z kamery a jeho následné vykreslenie do okna, funkcie pre jednoduché uživateľské rozhranie, okienka so "slidermi" na nastavovanie hodnôt. Pre zapínanie a vypínanie jednotlivých funkcií programu využijeme funkciu zabudovanú v `OpenCV` pre odchytávanie stlačených kláves. Knižnica `OpenCV` obsahuje tiež implementácie rôznych funkcií využívaných v počítačovom videní. Pri riešení tejto úlohy sa však sústredím na vytváranie vlastných funkcií, keďže mi ide aj o hlbšie pochopenie tejto problematiky a funkcie `OpenCV` budem používať v menšej miere.

2.1 Inštalácia a príprava nástrojov

V tejto časti sa budem venovať inštaláci potrebných nástrojov. Distribúcija Linuxu, na ktorej som vyvíjal tento program je Gentoo 2007.0 s kernelom 2.6.20-gentoo-r8.

2.1.1 Ovládače pre webkameru

V prvom rade bolo potrebné nainštalovať ovládače do operačného systému `Linux`. Distribúcia Linuxu, na ktorej som vyvíjal tento program je Gentoo 2007.0 s kernelom 2.6.20-gentoo-r8.

Mal som k dispozícii 3 webkamery. Jedna z nich bola Trust QuickCam Portable s čipom Sonix (hardware id 0c45:6029), druhá Logitech QuickCam Express (hardware id 046d:0928), a tretia webkamera od T-Comm (hardware id 0c45:612a).

Pre kameru Trust Quickcam Portable je podpora ovládača rovno v kerneli Linuxu. Pred použitím bolo treba pridať ovládač do jadra, pretože v bežnom nastavení nie je. To sa dosiahne tak, že sa cez `menuconfig` kernelu nastaví daná vlastnosť (Device Drivers → Multimedia Devices → Video Capture Adapters → V4L USB Devices → USB SN9C10x PC Camera Controller Support), a následne prekompilovaním a nainštalovaním nového kernelu.

Kamera Logitech Quickcam Express nefunguje s ovládačmi v jadre Linuxu, preto bolo potrebné použiť externý modul `gspcav1`. Po nainštalovaní tohto súboru ovládačov pomocov správcu balíčkov portage (príkaz `emerge`) už môžeme používať aj túto kameru.

Pre kameru od T-Commu ešte nie je ovládač pre Linux, takže som ju nemohol použiť.

2.1.2 Knižnice OpenCV

Pre inštaláciu knižníc `OpenCV` do operačného systému Gentoo Linux je potrebné stiahnúť si z internetu zdrojové súbory tejto knižnice. Je to preto že cez systém `Portage` sa nedajú nainštalovať ako balíček. Po stiahnutí je potrebné ich rozbaliť, a preložiť príkazom `make`, následne príkazom `make install` nainštalovať. Môže sa stať, že bude treba ešte v systémových premenných nastaviť cestu ku knižniciam, aby ich prekladač vedel nájsť.

2.1.3 Preklad programu

Program využívajúci knižnice OpenCV musí obsahovať hlavičkové súbory cv.h, cxcore.h a highgui.h. Preklad programu sa vykoná príkazom gcc -I/usr/local/include/opencv/ -L/usr/local/lib/ -lcxcore -lcv -lhighgui -lcvaux projekt.c -o projekt.

2.1.4 Alternatíva pre Windows

V prostredí tohto operačného systému som program netestoval, nemal by však byť problém ho tam prekompilovať a spustiť. Ovládače k webkamere sa inštalujú cez jednoduché klikacie inštalácije, pre knižnice OpenCV je tiež podobný inštalátor.

Kapitola 3

Problematika

Problém rozpoznania pohybujúceho sa objektu v scéne nie je vo svojej podstate príliž zložitý v prípade určitej presne danej a stálej scény. V takomto prípade by úplne stačilo pamätať si pozadie (nemuseli by sme sa trápiť s tým, ako ho obnovovať) a porovnávať s každým získaným obrázkom zo scény, pri čom by sme vedeli, že kde sa zachytené body líšia od zapamätaného pozadia sa nachádza objekt (narušitel). V reálnych situáciách sa však s takouto scénou pravdepodobne nestretneme a ak chceme, aby podobný systém bol použitelný aj v exteriéry a nie len v nejakej miestnosti bez okien a so stále zapnutým svetlom, potom musíme rátať s tým, že obraz pozadia sa bude obnovovať počas behu programu.

V exteriéroch sa však môžu nachádzať aj objekty, či povrchy ktoré spôsobia výrazné zmeny aj v rámci dvoch za sebou nasledujúcich zachytených obrazoch. Sú to napríklad stromy s listami, vodná hladina, alebo vlniaca sa zástava. Pri stromoch je problém to, že keď na jednej snímke je napríklad určitý pixel list stromu a na druhej už obloha, môže to byť dosť významná zmena na to, aby to vyvolalo falošnú detekciu objektu. Tento problém bol spomenutý v [2], kde je popísaný aj spôsob riešenia ktorý by stálo za to v budúcnosti vyskúšať. Takýto problém môže spôsobiť aj keď slnko zájde za mračná. V tomto prípade tiež nastane zmena dosť rýchlo na to aby vyvolala falošnú detekciu. Taktiež keď začne pršať sa napríklad zmení farba asfaltových povrchov. Ďalším problémom by mohol byť šum v signále z kamery. Každá kamera má určitú úroveň šumu. Pri dobre osvetlených scénach tento šum zvyčajne nespôsobí väčšie problémy, keďže obraz zachytený v takejto scéne je dostatočne výrazný oproti intenzite šumu. Problém však nastane pri tmavých scénach, kde je už potrebné nejakým spôsobom riešiť filtráciu tohto šumu alebo použiť iný hardware ako napríklad infra kameru.

V obrázku 3.1 je zobrazený postup ktorý využijem na detekciu objektov v tejto práci.

Obrázek 3.1: Postup detekcie objektov

3.1 Predpríprava obrazu

Vo väčšine systémov analyzujúcich obraz sa používajú určité princípy, ktorými sa obraz transformuje do podoby vhodnejšej pre spracovanie. Ide tu o úpravu intenzity pre udržanie citlivosti detekcie, alebo o podvzorkovanie obrazu, aby bola analýza menej náročná na systémové prostriedky. Pri riešení zvážim použitie obidvoch princípov.

3.2 Segmentácia obrazu

Problém, ktorý idem riešiť sa nazýva segmentácia obrazu (image segmentation), čím sa rozumie rozdelenie obrazu na časti, ktoré sú nejakým spôsobom významovo odlišné. Sú to napríklad rozdelenie obrazu na pozadie a popredie, ako v našom prípade, alebo vyhľadávanie rôznych objektov s charakteristickými vlastnosťami v obraze a podobne.

K riešeniu tohto problému sa dá pristupovať rôznym spôsobom ako napríklad:

- Porovnávaním svetlosti saturácie alebo iných vlastností pixelu.
- Segmentácia založená na optických tokoch (optical flow)
- Detekcia zmeny medzi jednotlivými snímkami.

Pri riešení tejto práce sa sústredím na segmentáciu založenú na detekcii zmien, lebo sa mi to zdá ako najschodnejšia metóda pre prvé pokusy. Keďže môžeme rátať s tým, že pri spustení systému nebude nikto v obraze, môžeme tiež rátať s tým, že musí do obrazu prísť a teda musí v ňom vyvolať nejakú zmenu.

Keďže rátame s čistou scénou bez narušiteľov na začiatku, môžeme prvý zachytený obrázok zobrať za obrázok pozadia. Potom potrebujeme zachytávať ďalšie obrázky a ukladať ich do druhej štruktúry a v každom kroku pred zachytením ďalšieho obrázku ich porovnať s obrázkom pozadia. To sa dá dosiahnúť počítaním rozdielov jednotlivých farebných zložiek pixelov a ich následné sčítanie, prípadne pri čiernobielom obraze počítaním rozdielov odtieňov šedej farby medzi pixelmi. Tu je dosť veľká výhoda používať radšej farebný obraz, lebo sa tým dosiahne vyššia citlivosť systému, pretože systém počítajúci v stupňoch šedi nedokáže rozlíšiť napríklad červenú a modrú farbu, ktorým prislúcha rovnaká šedá farba.

Keď vypočítame zmeny medzi pozadím a popredím musíme nejako odlúčiť nevýznamné zmeny, ktoré sú pravdepodobne spôsobené šumom v signáli alebo zmenami v pozadí od zmien dosť významných, aby boli považované za objekt. Toto sa najľahšie dosiahne prahovaním (thresholding), kde sa od určitej hranice rozdielu medzi dvomi obrázkami bude považovať zmena za objekt a pod touto hranicou ako nevýznamná zmena. Tu je dobré uvažovať tiež nad prahovaním väčších regiónov, kde by sa brala súhrnná hodnota pre viac susediacích pixelov, toto by tiež mohlo pomôcť odfiltrovať šum alebo väčšie zmeny osamotených pixelov, ktoré pravdepodobne nebudú reprezentovať objekt v popredí.

Keď už bude obraz rozdelený na pixely pozadia a popredia budeme potrebovať priradiť ich k objektu prípade k viacerým objektom, ak by sa ich tam nachádzalo viac. Tu budem považovať susediace pixely objektov v popredí za jeden objekt, pixely objektov, ktoré sú oddelené od seba pixelmi pozadia budú logicky považované za dva rôzne objekty. Algoritmus, ktorý by nám takto priraďoval pixeli k jednotlivým objektom by mohlo byť známe semienkové vypňovanie, prípade nejaký iný princíp.

3.3 Obnovovanie pozadia

Počas fungovania programu potrebujeme postupne obnovovať obraz pozadia scény, aby sa aspoň pomalé zmeny svetelných podmienok v snímanej oblasti po čase neprejavili ako falošná detekcia objektu. Aj keď to na prvý pohľad tak nevyzerá, riešenie tohto problému nie je triviálne a pri jeho riešení narážam často na nové problémy.

Najjednoduchším riešením je tzv. slepé obnovovanie pozadia. Pri tomto princípe sa po určitom počte snímok skopíruje celý obraz z kamery do obrázku pozadia. Bol to tiež prvý princíp, ktorý sa použil v programovej časti tejto práce. Pri vhodnom nastavení počtu snímok po ktorých sa raz obnovoval obraz boli výsledky celkom dobré, na druhej strane to tiež prinieslo problémy s ktorými som aj do predu počítal a ktoré sa dajú riešiť len nejakým komplikovanejším princípom. V prípade slepého obnovovania sa vyskytli nasledujúce problémy:

- Každý objekt zmizne v obraze v momente, keď sa obnoví pozadie a ak sa odvtedy nebude hýbať, bude aj naďalej považovaný za pozadie a prestane byť viditelný pre program ako objekt narušiteľ.

- Ak bol objekt v snímku, ktorý sa skopíroval do obrázku pozadia a ďalej sa pohybuje, bude aj miesto, na ktorom sa v tej chvíli nachádzal, považované za objekt na popredí z opačného dôvodu ako obyčajne - nie preto, že by bol objekt v snímke zachytenej z kameri a pozadie ktoré sa s ním nezhoduje v obrázku pozadia, ale naopak. Toto by spôsobovalo buď predĺženie detekovaného objektu, alebo pri dostatočne rýchlom pohybe objektu aj detekciu dvoch objektov miesto jedného. Toto by nastalo ak by objekt stihol celý odísť z miesta, kde sa skopíroval do pozadia.

- V prípade pohybu narušiteľa, ktorý by po zosnímaní kamerou bol zložený s pixelov s dostatočne podobnou farbou, bol by jeho pohym pri pomalých rýchlostiach zachytený ako pohyb dvoch objektov. Tento problém bol opísaný v [4] pod kapitolou 15.

Z toho vyplýva, že potrebujeme zabrániť tomu, aby sa detekovaný objekt v popredí skopíroval do pozadia. Riešením tohto problému by mohlo byť to, že by sme do pozadia kopírovali len tú časť obrázku, ktorá nie je považovaná za objekt v popredí. Tu ale narážam na ďalšie problémy. Jeden je spôsobený tým, že pixely na okraji objektu sa aj tak dostávajú do pozadia, v dôsledku toho, že objekt zachytený kamerou s nízkym rozlíšením má zrejme na okrajoch pixely ktoré sú z časti ovplyvnené pozadím a s časti objektom, a tak vytvárajú zrejme postupne plynulejší prechod medzi pozadím a popredím.

Tento problém sa dá vyriešiť ďalšou úpravou princípu obnovovania pozadia, a to tak, že by sa pixely v blízkosti objektu tiež nekopírovali do pozadia. Išlo by o akýsi okraj okolo detekovaného objektu, ktorý by sa neobnovoval do pozadia. Stále je tu ale problém s objektami ktoré pomaly prichádzajú do scény cez okraj. Takto sa môže pár pixelov objektu objaviť v obraze bez toho aby prekročili hranicu pre detekciu objektu, tieto pixely sa takto dostnú do pozadia scény. Ak sa potom objekt pohybuje dosť pomaly môže sa takto za určitých podmienok dostať celý do obrázku pozadia.

Lepšie výsledky by bolo možné dosiahnúť, ak by si program pamätal na koľkých obrázkoch zachytených z webkamery sa daný pixel neodlišoval o viac ako o určitú zadanú hranicu a len ak by tento údaj bol väčší ako prednastavená hodnota by sa pixel updatoval do pozadia. Bolo by tiež vhodné tento princíp skombinovať s vyššie spomenutým pravidlom nekopírovať do pozadia pixely, ktoré sú považované za objekt v popredí alebo okolie objektu

v popredí. Tu ale stále nie je isté či sa nevyskytnú vyššie spomenuté problémy s objektom prichádzajúcim na scénu.

Podobných princípov s rôznymi pravidlami pre obnovovanie pozadia existuje veľa. Pár známejších je spomenutých v [5]. Pri implementácii zvážim použitie niektorých z týchto princípov.

Kapitola 4

Implementácia

Ako som už spomenul v predchádzajúcich kapitolách, na tvorbu aplikácie som použil jazyk C a knižnicu OpenCV. V tejto kapitole budem rozoberať ako som naimplementoval jednotlivé funkcie. Spomeniem aj funkcie OpenCV ktoré som použil a stručne popíšem ako fungujú. Detailnejší opis všetkých funkcií sa dá nájsť v [1].

4.1 Zachytenie a vykreslenie obrazu z kamery.

Získavanie obrazu pomocou knižnice OpenCV je jednoduché. Najprv je treba vytvoriť si akýsi ukazateľ na zariadenie zachytávajúce opraz. Tento ukazateľ je typu `CvCapture*` a inicializuje sa hodnotou, ktorú vráti funkcia `cvCaptureFromCAM`. Táto funkcia má jeden parameter určujúci, ktoré zariadenie sa má použiť. V prípade zadania hodnoty `CV_CAP_ANY`, čo je vlastne definovaná konštanta OpenCV s hodnotou -1, sa vyberie prvé zariadenie v zozname.

OpenCV používa pre ukladanie obrázkov dátovú štruktúru `IplImage`, ktorá obsahuje rôzne informácie o formáte uloženého obrazu, ako je počet kanálov, farebná škála, výška a šírka v pixeloch a tak ďalej. Potom sa tu nachádza odkaz na pole premenných typu `integer`, ktoré reprezentujú informácie o farbe jednotlivých pixelov.

Obraz z kamery získame príkazom `cvQueryFrame`. Ako parameter sa zadáva spomínaný ukazateľ na zariadenie zachytávajúce obraz a vracia štruktúru `IplImage` so zachyteným obrázkom.

Pre následné vykreslenie obrazu použijeme okno vytvorené implementáciou grafického užívateľského rozrania v OpenCV. OpenCV okno sa vytvára príkazom `cvNamedWindow`. Táto funkcia má dva parametre, jeden je meno okna ktoré ho jednoznačne identifikuje a druhý parameter sú "flags", ktoré upravujú správanie sa okna. V druhom parametri budeme využívať len vlastnosť `CV_WINDOW_AUTOSIZE`, ktorá spôsobí že rozmer okna sa automaticky prispôsobí obrázku, ktorý bude doňho vykreslený.

Do vytvoreného okna vykreslíme obrázok pomocou funkcie `cvShowImage`, kde prvý parameter bude meno okna, do ktorého chceme vykresliť obrázok a druhý parameter ukazateľ na štruktúru typu `IplImage` s obrázkom, ktorý chceme vykresliť.

Pre stálé zobrazovanie z kamery sa v nekonečnom cykle musí opakovať volanie funkcie `cvQueryFrame` a `cvShowImage`.

4.2 Funkcie pre ovládanie programu a nastavovanie parametrov

Aplikácia sa ukončí po stlačení klávesi `esc`. Je to zariadené funkciou `cvWaitKey`, ktorá vracia kód stlačenej klávesy, ak bola nejaká stlačená. Funkcia má jeden parameter a to čas, aký má funkcia čakať v milisekundách.

Po získaní kódu stlačenej klávesy, program kontroluje, či kód nezodpovedá niektorej aktívnej klávese v tomto programe. Tieto klávesy sa používajú pre zapínanie a vypínanie rôznych podsystémov v programe a tým sa upravuje jeho správanie. Po stlačení takejto klávesy sa zavolá funkcia `check_keys`, ktorej prvý parameter je kód klávesy a druhý je ukazateľ na riadiacu premennú typu integer `flags`, kde jednotlivé bity reprezentujú stav aktivácie jednotlivých funkcií. Funkcia `check_keys` prepne bit asociovaný s kódom stlačenej klávesy volaním funkcie `toggle_flag`. Funkcia `toggle_flag` má dva parametre pričom prvý z nich je ukazateľ na premennú s riadiacimi bitmi a druhý premenná typu integer identifikujúca príznak, ktorý treba prepnúť. Tieto príznaky sú definované v hlavičkovom súbore `kamerka.h`.

V programe sa ďalej nastavujú veľkosti hodnôt premenných ovplyvňujúcich funkcie programu. Tu sa zas využívajú možnosti, ktoré nám poskytuje jednoduché uživateľské rozhranie v OpenCV. Ide o ovládacie prvky typu "slider". Ten sa vytvára v programe volaním funkcie `cvCreateTrackbar`, ktorá pridá do daného okna posúvací ovládací prvok, ktorý bude mať popis zadaný reťazcom v prvom parametre funkcie, bude sa zobrazovať v okne, ktoré má meno podľa druhého parametru funkcie, bude nastavovať premennú, ktorú jej zadáme v treťom parametre jej odkazom. Maximálnu hodnotu nastaviteľnú v tomto riadiacom prvku udáva štvrtý parameter funkcie. Posledný parameter tejto funkcie sme nevyužili a je to ukazateľ na funkciu, ktorá sa má volať pri zmene hodnoty. Ak nieje potrebná žiadna akcia pri zmene hodnoty, čo bol náš prípad, potom je posledný parameter `NULL`.

4.3 Predpríprava obrazu

Na predprípravu obrazu máme implementované dve funkcie ktoré, upravujú jas a kontrast zachytených obrázkov.

Prvou z týchto funkcií je funkcia `adjust_brightness`. Táto funkcia zvyšuje prípadne znižuje celkovú svetlosť obrázku tak, aby sa dostala na nejakú rozumnú hodnotu. Jednoducho povedané, aby obraz nebol veľmi tmavý alebo veľmi svetlý. Funkcia má dva vstupné parametre, kde prvý je odkaz na obrázok uložený v štruktúre typu `IplImage` a druhým hranica priemernej hodnoty pixelu, za ktorou sa má obraz upravovať. Algoritmus môže obraz zosvetliť aj stmaviť, preto táto hranica bude aj dolná (ak je priemer menší tak sa obraz upraví), ale aj horná hranica, teda ak bude priemer väčší ako maximálna možná hodnota mínus hranica. V prvom kroku tohto algoritmu si spočítame priemerú hodnotu stupnov šedi pre všetky pixely, prípadne priemernú hodnotu všetkých farebných zložiek všetkých pixelov. Tu využijeme funkciu implementovanú v OpenCV a to `cvAvg`, ktorej prvý parameter je odkaz na `IplImage` a druhý nevyužívame a môže byť kľudne `NULL`. Funkcia vráti štruktúru `CvScalar`. Ide vlastne o štruktúru typu pole o štyroch premenných typu `integer`, kde každá prislúcha jednej farebnej zložke red green blue, prípadne alpha pri obrázkoch s priesvitnosťou. Funkcia vráti pri farebnom obrázku priemer zvlášť pre každú z troch základných farieb. My ale potrebujeme spoločný priemer pre všetky a preto treba ešte vypočítať priemer z týchto troch farieb. Vo funkcii `adjust_brightness` sa potom vypočíta násobiteľ

tak, aby sa po vynásobení priemeru s týmto násobitelom priemer dostal na zadanú hranicu ak je mimo nej. V poslednom kroku sa s týmto násobitelom vynásobí každá farebná zložka všetkých pixelov.

Druhou použitou funkciou, ktorú som implementoval je `histogram_equalization`. Tento algoritmus som implementoval podľa algoritmu histogram equalization popísanom v [4] a má za cieľ zvýšiť kontrast v obraze. Funguje na jednoduchom princípe a to tak, že najprv si vytvorí histogram podľa hodnôt všetkých pixelov. Z tohto histogramu si potom vytvorí kumulatívny histogram a to tak že sa prebehne histogram a do kumulatívného histogramu sa bude ukladať vždy hodnota v predchádzajúcom histograme plus predchádzajúca hodnota v kumulatívinom histograme. Na koniec sa podľa vzorca uvedeného v [4] vypočíta z hodnôt v kumulatívnom histograme tabuľka pre prevod pôvodných hodnôt pixelov na novú hodnotu. Pre túto tabuľku môžeme využiť tú istú dátovú štruktúru ako pre histogrami, tj. poľe premenných typu integer o šírke rovnajúcej sa počtu hodnôt akých môže pixel v našom obraze nadobudnúť. Na základe tejto tabuľky sa nakoniec zmení každý pixel v obraze tak, že index v tabuľke je jeho pôvodná hodnota a hodnota pod týmto indexom nová hodnota pre pixel. Tento algoritmus funguje správne len pri čiernobielych obrázkoch, pri farebných spôsobí, že sa v ňom objavia veľmi neprirodzené farby.

Tieto algoritmi pre prípravu obrazu sú síce dobré na to, aby človek lepšie videl objekty v obraze, ale pre zlepšenie citlivosti strojového videnia zjavne nepomáhajú. To je spôsobené tým, že pri zvýraznení kontrastu prípadne intenzity v obraze sa tak isto zvyšuje aj intenzita šumu, ktorý sa v obraze nachádza. Rovnakého výsledku pri detekcii objektov musíme dosiahnúť aj znížením prahu detekcie, ak je obraz tmavý. Tým sa zas zvýši aj množstvo falošných detekcií. Tieto poznatky nás zas vedú k záveru, že pre dobré výsledky je vhodné mať dostatočne osvetlenú scénu.

4.4 Segmentácia obrazu

Pre rozdelenie obrazu v scéne som použil jednoduchý princíp na základe prahovania medzi obrázkom pozadia a novým zachyteným obrázkom z kamery. Obraz pozadia sa obnovuje podľa určitých pravidiel, algoritmov, ktoré sú popísané v nasledujúcej kapitole.

4.4.1 Výpočet obrazu rozdielov

V každom kroku sa vypočíta obraz, kde hodnoty pixelov reprezentujú rozdiely medzi aktuálnym obrázkom a obrázkom pozadia. Jedná sa o čiernobiely obraz kde intenzita pixelu reprezentuje odlišnosť pixelu v zachytenom obrázku od prislúchajúceho pixelu v obrázku pozadia. O výpočet obrázku zmien sa stará funkcia `count_diff_frame`, ktorej sa parametrami predá odkaz na aktuálny obrázok, obrázok pozadia a odkaz na datovú štruktúru pre obraz zmien a premennú typu `integer`, ktorá slúži na určenie, či sa má pracovať s farebným alebo čiernobielim obrazom. Ak sa jedná o farebný obraz, potom táto funkcia počíta rozdiel ako súčet všetkých farebných zložiek v tomto obraze. Ak je súčet väčší ako 255 (čo je hranica pre pixel v obraze zmien, ktorý som použil), tak sa hodnota určí na túto maximálnu hodnotu. Pri čiernobielom obraze sa rozdiely pixelov počítajú jednoducho ako absolutná hodnota rozdielu pixelov medzi obrazom pozadia a aktuálnym.

4.4.2 Prahovanie rozdielov pre detekciu objektov

Pre detekciu objektu som sa rozhodol umožniť aby sa prah určoval nie pre samostatné pixely, ale skupiny pixelov v štvorcoch s nastavitelnými rozmermi, vďaka čomu sa môže zamedziť tomu, aby sa jeden objekt detekoval ako viac menších objektov. Tiež sa s tým odfiltrujú detekcie malých regiónov, ktoré sú pravdepodobne len zmeny v pozadí spôsobené pohybom lístia a to preto, že pri takomto prahovaní súčtu viacerých pixelov vedľa seba môžeme dosiahnúť, aby bola potrebná zmena viacerých pixelov v jednej kope pre dosiahnutie detekcie objektu. Pri tomto princípe nejaké chaotické zmeny jednotlivých pixelov vzdialených od seba nespôsobia falošné detekcie, ak je ich hustota dosť nízka.

V podstate sa dá povedať, že sa takto znižuje rozlíšenie v obraze zmien pri zvýšení škáli hodnôt pre jeden pixel a následné prahovanie v tomto obraze. O segmentáciu obrazu sa stará funkcia `simple_segmentation`. Tejto funkcii sa predá obraz zmien, čo je štruktúra `IplImage`, vlastne definovanú štruktúru `segmentPict`, rozmeri štvorcov zhlukov pixelov, ktoré sa budú sčítavať pred prahovaním v dvoch premenných typu `integer` a prah detekcie objektov v premennej typu `integer`.

Do štruktúri `segmentPict` sa nám vlastne uloží segmentovaný obraz, kde nula reprezentuje pozadie a väčšie ako nula reprezentujú detekovaný objekt. V tejto štruktúre sú uložené počet riadkov a stĺpcov obrazu, šírka a výška štvorcov, aká bola nastavená vo funkcii `simple_segmentation` (tieto údaje budú potrebné pre ďalšie využitie tohto obrazu pre označenie detekovaných objektov) a odkaz na pole s premennými typu `integer` pre dáta obrazu.

4.4.3 Rozoznanie oddelených objektov

Ďalším krokom detekcie je priradenie jednotlivých bodov označujúcich detekcie v štruktúre `segmentPict` k jednotlivým objektom. Aby systém zistil, či je v obraze jeden alebo viac objektov a ktoré pixely patria ku ktorému objektu. To sa dosiahne tak, že sa zistí, ktoré body v obraze sú susediace. Všetky susediace body teda budú považované za jeden objekt. Táto metóda síce nerieši prípady, keď sa v obraze prekrývajú dva objekty, ale pre túto prácu nám to zatiaľ nevadí. Spôsobovalo by to problémy, ak by sme tento systém chceli využívať na účel sledovania frekventovanosti dopravy na ceste alebo počítanie ľudí čo prešli cez dvere.

Pre spájanie susediacich bodov do jedného objektu by sa dal využiť algoritmus semienkového vyplňovania. Ten by sa spustil vždy keď by sme pri prechode struktúrou `segmentPict` narazili na nejaký bod detekcie, a všetky body, do ktorých by sa algoritmus dostal by boli priradené k jednému objektu a vyradené z ďalšieho spracovania. Ja som sa rozhodol vyskúšať iný princíp. Implementoval som ho vo funkcii `cluster_detection`. Dalo by sa síce pochybovať, či je tento algoritmus efektívnejší, ako keby som využil semienkové vyplňovanie pretože, síce mu stačí jediný prechod obrazom, v niektorých krokoch vyžaduje zjavne zložitejšie rozhodovanie a úkony ktoré sa musia vykonať, ale je to určite zaujímavý spôsob, ako riešiť tento problém. Bolo by zaujímavé v budúcnosti implementovať aj algoritmus so semienkovým vyplňovaním a porovnať výkon týchto dvoch algoritmov.

Vstupom do tohto algoritmu, ktorý je implementovaný vo funkcii `cluster_detection` je štruktúra `segmentPict`. Jeho produktom je zas štruktúra `ObjTable`, v ktorej sú uložené informácie o každom detekovanom objekte a všetky potrebné údaje pre určenie jeho polohy v obraze a veľkosti. Jedná sa vlastne o jednosmerne viazaný zoznam. V programe sú implementované funkcie pre prácu s týmto zoznamom a to:

- `ob_tab_init` - počiatočná inicializácia tabuľky.
- `ob_add` - pridanie nového objektu do tabuľky. Funkcia vracia priradené poradové číslo v tabuľke. Parametrami sa mu predáva okrem tabuľky, do ktorej sa má objekt vložiť ešte aj jeho poloha v x-ovej a y-ovej súradnici, podľa čoho sa nastavia okraje objektu, podľa ktorých vieme ďalej nakresliť rámik okolo pohybujúceho sa objektu.
- `ob_tab_clear` - vymazanie všetkých objektov z tabuľky.
- `ob_merge` - spojí dva objekty a nastaví jeho okraje tak, že v obidvoch osiach x aj y nastaví jeho minimum na menšiu hodnotu z oboch objektov a maximum na tú väčšiu. Táto funkica je potrebná pre prácu algoritmu, lebo ten pri prechode môže v jednom riadku vytvoriť jeden objekt a potom pri pokračovaním v riadku môže zistiť, že nadväzuje na objekt v predchádzajúcom riadku a vtedy sa musia spojiť.
- `ob_find` funkcia vracajúca odkaz na objekt s daným indexom. Je potrebná pre algoritmus, aby sa stále nemuselo prechádzať v lineárnom zozname, keď sa niečo v ňom má zmeniť. Takto sa jednoducho vyhľadá objekt raz a potom sa s ním cez vrátený ukazateľ môže pracovať bez ďalšieho vyhľadávania.

Algoritmus ešte využíva štruktúru `ObjConnections`, ktorá vlastne reprezentuje, ktoré body v riadku nad riadkom, ktorý práve prechádzame, a v tomto poli je označené číslom 0 kde sa nenachádzal žiaden detekovaný objekt, a číslom objektu zo štruktúry `ObjTable` pre body v ktorom objekty sú. Táto štruktúra obsahuje jednu premennú typu `integer`, ktorá reprezentuje šírku poľa pre tieto údaje, potom odkazy na dve polia s premennými typu `integer`, kde jedna je pre predchádzajúci riadok v ktorej sa kontroluje či nad bodmi nejakého objektu niesu body iného objektu, druhá sa zatiaľ napĺňa informáciami o riadku ktorý práve prechádzame. Po prechode na ďalší riadok sa tieto zoznamy vymenia. Pre štruktúru `ObjConnections` sú v programe implementované nasledujúce funkcie:

- `conn_init` - inicializácia štruktúry.
- `conn_clr_next` - vyprázdni zoznam s informáciami pre ďalší riadok.
- `conn_next_line` - používa sa pri prechode na ďalší riadok. Najprv sa prehodia zoznami pre aktuálny a nový riadok a potom sa zavolá funkcia `conn_clr_next`.
- `conn_free` - uvoľní z pamäte obe polia.
- `is_conn` - zisťuje či nad bodom v aktuálnom riadku a v zadanej x-ovej súradnici je objekt s ktorým by sa ten čo sme našli spájal.
- `insert_conn` - vkladá do poľa informácie o objekte v riadku.
- `change_cThisLine_objNum` - zmení číslo objektu v terajšom riadku z čísla zadaného v druhom parametri funkcie na číslo zadané v tretom parametri funkcie.
- `change_cNextLine_objNum` - to isté ako funkcia `change_cThisLine_objNum` len edituje poľe pre ďalší riadok.

Ako už bolo spomenuté, funkcia `cluster_detection` prechádza raz celé poľe v štruktúre `segmentPict`, a kým nenarazí na žiaden bod, ktorý by bol označený ako bod objektu len pokračuje ďalej a nerobí v podstate nič. V okamihu, keď narazí na bod detekovaného objektu spraví nasledujúce akcie:

- ak nad ním nie je bod s detekovanom objekte na spojenie (teda funkcia `is_conn` vráti 0) sa vytvorí nový objekt a údaje o ňom sa následne upravujú.
- Ak je nad ním detekovaný objekt na spojenie (teda funkcia `is_conn` nevráti 0) budeme len upravovať údaje o tomto detekovanom objekte v štruktúre `ObjTable`, a to dolný okraj (`maxY`).

Ak sme už predtým našli bod detekovaného objektu a pokračujeme v prechádzaní jeho bodou v aktuálnom riadku:

- Ak je nad ním detekovaný objekt na spojenie (teda funkcia `is_conn` nevráti 0), spojí aktuálny objekt s objektom, ktorý je nad ním volaním funkcie `obj_merge`. Potom upraví záznam v štruktúre `ObjConnections` podľa toho, ktorý objekt zanikne sa upraví buď v riadku pre aktuálny, alebo pre ďalší riadok volaním funkcie `change_cThisLine_objNum` prípadne `change_cNextLine_objNum`.
- S funkciou `obj_find` sa zistí ukazateľ na objekt, ktorý ostal a ďalej sa upravuje.
- Upravujú sa okraje objektu.

Výsledkom po ukončení tejto funkcie je štruktúra `ObjTable` so záznamom pre každý pohybujúci sa objekt v popredí.

4.4.4 Označenie objektov

V obraze sa detekované objekty v popredí označia funkciou `oznac_objekty`, ktorej predáme obraz, do ktorého chceme vyznačovať objekty, odkaz na štruktúru `segmentPict` a štruktúru `ObjTable` vytvorenú funkciou `cluster_detection`.

Vykreslenie obdĺžniku okolo objektu sa uskutoční pomocou funkcie z knižnice OpenCV `cvRectangle`, ktorej sa zadá obraz, do ktorého sa má kresliť, súradnice ľavého horného rohu, pravého horného rohu a farba.

4.5 Údržba obrazu pozadia

Tento problém bol rozvinutý v programe tromi rôznymi metódami, pri čom každá z nich má svoje výhody aj nevýhody, každá z nich zlyháva v iných prípadoch.

4.5.1 Slepé obnovovanie pozadia

Prvá z metód, ktorú som implementoval bolo slepé obnovovanie pozadia. Pri tomto obnovovaní sa inkrementuje počítadlo a porovnáva sa s nastavenou hodnotou. Ak uplynie zvolený počet snímok sa aktuálny obrázok skopíruje do pozadia.

4.5.2 Metóda čiastočného obnovovania pozadia

Táto metóda je založená na myšlienke, že keď vieme spoľahlivo detekovať objekt, vieme zamedziť aj jeho skopírovani do pozadia. Je implementovaná vo funkcii `update_background`. Teda môžeme kopírovať len časti obrazu, v ktorých nebol detekovaný objekt. Logicky teda použijeme štruktúru s detekovanými objektami `objTable` a rovnakým spôsobom, ako sa označovali objekty v obraze teraz vybrať regióny obrazu pozadia, ktoré sa majú obnoviť. Na takéto kopírovanie častí obrazu je v OpenCV vhodná funkcia `cvCopy`. Táto funkcia má

tri vstupné parametre, kde každý z nich je štruktúra `IplImage`, prvý je zdrojový obrázok, druhý cieľový obrázok a posledný je maska. Práve maska určuje, ktoré regióny obrázku sa majú skopírovať, kde nulová hodnota označuje pixely, ktoré sa nemajú skopírovať a nenulová hodnota pixely, ktoré sa majú skopírovať. Teda nám stačí vytvoriť si obrázok s maskou, ktorú si najprv zinicializujeme na nenulovú hodnotu a potom funkciou `cvRectangle` vykreslíme čierne obdĺžniky v miestach kde sme detekovali objekty.

Pre zníženie rizika, že sa nejaké okrajové pixely objektu dostanú do pozadia som v implementácii ešte rozšíril oblasť okolo objektu, ktorá sa neobnovuje o okraj. Týmto sa znížilo riziko že sa poškodí obrázok pozadia.

Ďalším vylepšujúcim pravidlom pre tento algoritmus je že sa obnovujú len tie pixely, ktoré neboli označené za objekt v popredí za určitý počet snímkov. Potom som ešte pridal do algoritmu pravidlo, že sa neobnovujú pixely, ktoré sa za určitý počet posledných snímkov odlišovali od pozadia o viac ako určená hranica. Týmto som sa snažil vyriešiť problém s poškodením pozadia pri pomalom príchode objektov na scénu.

Aby sa pozadie obnovovalo až po určitej dobe stálosti, potrebujeme nejakú štruktúru s informáciami o tom, ako dlho určitý pixel bol dostatočne stály. Keďže najjednoduchšie riešenie bolo použiť už definovanú štruktúru, rozhodol som sa použiť aj pre tento účel štruktúru `IplImage`. Funkcia `rstFTUBgFromDiff` sa stará o spravovanie údajov v tejto štruktúre. Nastavuje pixeli ktoré majú v `diffFrame` (obrázok zmien) hodnotu vyššiu ako nastavená hranica vo `FTUBgFrame` (frames to update background frame) na nastavenú hodnotu, ktorá určuje koľko obrázkov sa má čakať, kým sa pixel zas začne kopírovať do pozadia. Túto štruktúru potom ešte upravuje aj funkcia `update_background` a to na základe štruktúry `objTable`. Samozrejme sa všetky pixely v `FTUBgFrame` dekremetujú vždy raz pri jednom prechode programu hlavným cyklom.

4.5.3 Obnovovanie pozadia Heikkila and Olli

Túto metódu obnovovania pozadia som implementoval podľa popisu v [5]. Bola podrobnejšie popísaná a použitá v práci [3]. Ide o princíp, kde pixely z každého zachyteného obrázku ovplyvňujú malou mierou farbu pixelov v pozadí.

Algoritmus funguju tak, že sa zoberie obraz pozadia a popredia a všetky pixeli nového obrazu pozadia sa vypočítajú na základe zadaného čísla alfa, ktoré môže nadobúdať hodnoty v definičnom obore $< 0, 1 >$. Toto číslo určuje akú váhu bude mať pixel v zachytenom obraze v novom obraze pozadia oproti starému obrazu pozadia. Ak bude mať alfa hodnotu 0, pozadie sa nebude vôbec obnovovať. Naopak v prípade ak alfa nadobudne hodnotu 1, bude sa systém chovať ako slepé obnovovanie pozadia. Ako bolo spomenuté v [5], alfa musí byť malá aby sa zabránilo tvoreniu "chvostov" v detekcii pohybujúceho sa objektu.

Pri implementácii som nastavovanie hodnoty premennej vyriešil tak, že sa nastavujú "sliderom" tisíciny hodnoty alfa. Spravil som to tak preto, lebo ovládací prvok v jednoduchom uživateľskom rozhraní pre `OpenCV` nepočíta s možnosťou nastavovať reálne čísla.

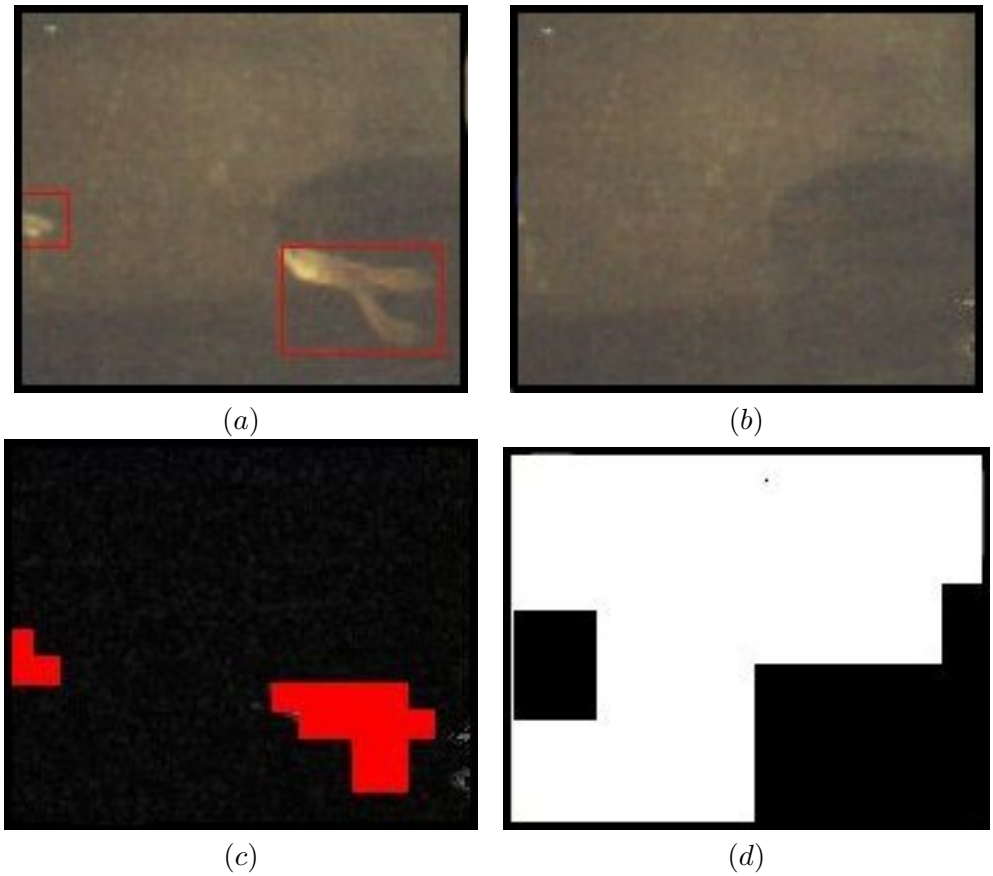
Aby bolo možné tento systém používať aj pri malých hodnotách `alfa`, musí si pamätať systém aj malé zmeny v pozadí (menšie ako 1). Ak by si nepamätal tieto malé zmeny, potom by sa obraz pozadia pri malom rozdieli s popredím už prestal prispôsobovať. Keďže obraz `IplImage` používa celé čísla, musel som skopírovať body pozadia do poľa premenných typu `long double` a upravať ich tu a v každom kroku ich potom zaokrúhliť a skopírovať do prvého obrazu pozadia.

Kapitola 5

Nastavenia programu

Ako som spomínal v predchádzajúcich kapitolách, správanie aplikácie sa dá upravovať za behu pomocou klávesnici a okna s ovládacími prvkami.

V nasledujúcom obrázku 5.1 je ukázaný príklad ako môžu vyzerať okná v programe pri práci programu.



(a) (b) (c) (d)

Obrázek 5.1: Okná programu: (a) okno s aktuálnym obrázkom scény (b) okno s pozadím scény (c) okno s obrazom zmien (d) okno s maskou obnovovania

Po spustení programu sa otvoria štyri okná, a to:

- okno `popredie` v ktorom sa zobrazuje vždy aktuálny obraz z webkamery.
- okno `zmeny` v ktorom sa zobrazuje obráz zmien a vyznačujú sa červenými štvorčekmi. Čierna farba znamená že nebola žiadna zmena a biela že bola maximálna možná zmena.
- okno `pozadie` v ktorom sa zobrazuje obraz pozadia objektu.
- okno `UpdateMask` v ktorom sa zobrazuje maska pre obnovovanie pozadia. Toto okno sa prekresluje len ak je zapnutý systém čiastočného obnovovania pozadia. Biela farba označuje pixeli ktoré sa kopírujú do pozadia a čierna ktoré nie.
- okno `nastavenia` v ktorom sú "slideri" pre nastavovanie riadiacich premenných programu.

5.1 Klávesové skratky

Aktívne klávesy ktoré sa dajú použiť pri behu programu sú:

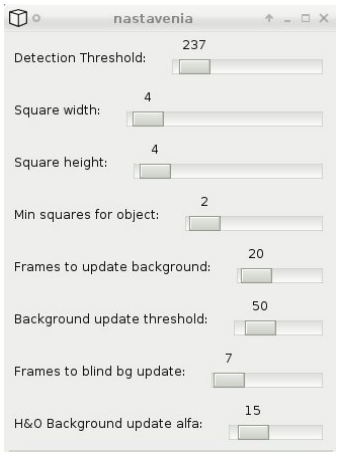
- `S` - zapne funkciu `swap red blue`, ktorá prehadzuje červený a modrý kanál v obraze zachytenom z kamery. Táto funkcia nieje nijak podstatná pre detekciu objektov v obraze, len upravuje problém s niektorými ovládačmi webkamery, ktoré popletú tieto kanáli a potom má obraz neprirodzené farby.
- `G` - zapína a vypína redukciu farebného priestoru v zachytenom obraze na stupne šedi.
- `H` - zapína a vypína funkciu `histogram_equalization`, ktorá zvyšuje kontrast v zachytenom obraze.
- `B` - zapína a vypína upravovanie svetlosti obrazu (funkciu `adjust_brightness`).
- `U` - skopíruje celý aktuálne zachytený obraz do pozadia.
- `L` - zapína a vypína slepé obnovovanie pozadia po určitom počte snímok.
- `P` - zapína a vypína čiastočné obnovovanie pozadia (funkcia `update_background`).
- `O` - zapína a vypína metódu obnovovania pozadia Heikkilä Olli (funkcia `Heikkila_and_Olli_bg_update`).
- `W` - zapína a vypína zápis videa do súboru video.mpeg.
- `I` - vypíše do konzoli informácie o zapnutí jednotlivých systémov programu.

Všetky tieto systémy sú pro spustení programu vypnuté.

5.2 Nastaviteľné premenné programu

Pre hladanie optimálneho nastavenia je potrebné aby sa dali niektoré premenné meniť počas behu programu. Toto je umožnené cez okno `nastavenia`, kde sa tieto premenné dajú nastaviť. V tejto časti vysvetlím čo ktorá premenná znamená.

Na obrázku 5.2 vidíme ako vyzerá okno na nastavovanie premenných v programe. Každý "slider" nastavuje jednu premennú a to konkrétne:

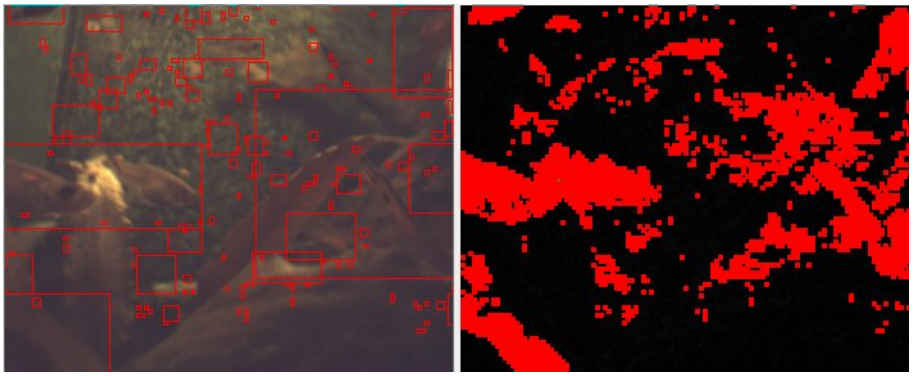


Obrázek 5.2: Okno nastavenia

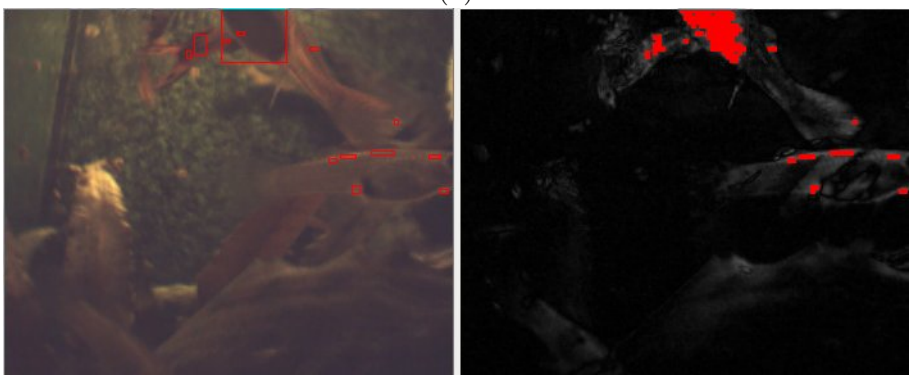
- `Detection threshold` - nastavuje prah pre funkciu `simple_segmentation` Tento prah sa porovnáva so súhrnnými zmenami v oblastiach v `diffFrame`. Jednoducho povedané čím nižšia bude táto hodnota, tým bude systém citlivejší na zmeny v obraze.
- `Square width` a `Square height` - nastavujú sa rozmery štvorca ktorý sa bude brať pri prahovaní ako jeden pixel vo funkcii `simple_segmentation`.
- `Min squares for object` - nastavuje sa minimálny počet susediacich štvorcov, ktoré boli vyhodnotené ako objekt, aby sa objekt v obraze označil. Takto môžeme testovať aký vplyv by malo na obraz obmedzenie malých regiónov zmien.
- `Frames to update background` - nastavuje počet snímkov, počas ktorých musí byť pixel dostatočne stály, aby sa začal kopírovať do pozadia pri čiastočnom obnovovaní pozadia.
- `Background update threshold` - určuje hranicu pod ktorou musí byť rozdiel medzi pixelom a pozadím počas určeného počtu snímkov, aby sa pixel začal kopírovať do pozadia pri čiastočnom obnovovaní pozadia.
- `Frames to blind update bg` - určuje po koľkých snímkoch sa má zachytený obraz skopírovať do pozadia, ak je zapnuté slepé obnovovanie pozadia.
- `H&O Background update alfa` - určuje hodnotu alfa pre obnovovanie pozadia metódou Heikkila Olli. Toto číslo sa v programe potom delí tisícimi, takže vlastne pri hodnote 100 bude alfa rovná 0.1.

Pre správnu detekciu objektov je potrebné doladiť najmä `Detection threshold`, `Square width` a `Square height`. Pri príliž nízkom prahu pre detekciu sa budú objavovať falošné detekcie náhodne v celom obraze, alebo v miestach obrazu ktoré sú výrazne lepšie osvetlené. Vysoké hodnoty `Sqare width` a `Sqare height` spôsobia, že malé objekty nebudú detekované, na druhej strane je však menšia šanca že systém označí jeden veľký objekt za viac menších.

Na obrázku 5.3 môžeme vidieť ako sa prejaví nesprávne nastavenie prahu detekcie. V (a) môžeme vidieť čo sa stane keď je prah príliž nízky. Tu sa v obrázku každá malá zmena



(a)



(b)

Obrázek 5.3: Zlé nastavenia prahu detekcie - (a) príliž nízky prah, (b) príliž vysoký prah

považuje za objekt a preto je skoro celý obraz posiaty náhodnými falošnými detekciami. V druhom prípade (b) zas môžeme vidieť ako sa prejaví nastavenie príliž vysokého prahu. Z objektov v scéne sa detekujú len najvýraznejšie časti a pri menších objektoch sa nezistí ich prítomnosť vôbec. Teda pri nastavení prahu treba hľadať niečo medzi, nejakú optimálnu hodnotu. Ak sa detekujú len časti objektov a len s nastavením prahu detekcie je problém dosiahnúť rozumný výsledok, potom sa treba pohrať aj s nastavením `Square width` a `Square height`, treba ich skúsiť zvýšiť.

Kapitola 6

Zhodnotenie výsledkov

Pri implementácii programu som využíval najjednoduchšie princípy aké mi prišli na rozum prípadne ktoré som našiel v literatúre. Tieto princípy majú ale veľa problémov v určitých prípadoch.

Detekcia objektov je závislá od zvoleného prahu. Preto systém s jednou hodnotou zvoleného prahu môže fungovať dobre pri silnejšom osvetlení scény, ale nemusí už fungovať správne pri slabšom osvetlení a naopak. Záleži tu tiež od toho aký veľký rozdiel je medzi farbami objektu a pozadia za ním.

Na druhej strane, ak chceme len zistiť, že do stráženeho priestoru vstúpila nejaká osoba nemusí byť detekcia až tak dokonalá a preto prah nemusí byť nutne nastavený na najcitlivejšiu detekciu. Pri takom nastavení je ešte priestor aby systém fungoval aj pri silnejšom osvetlení. Ťažko si však možno predstaviť, aby bol použiteľný program, ktorý by bolo stále treba nastavovať aby fungoval. Preto ak by som mal pokračovať v používaní tohto princípu založenom na zmenách, tak by bolo zrejme vhodné spraviť nejaký systém, ktorý by automaticky nastavoval tento prah. Dalo by sa to dosiahnúť tak že by sa vypočítal priemer intenzity pixelov a na základe tohto štatistického údaju upravovať prah.

Z metód údržbi pozadia fungovala najspoľahlivejšie metóda Heikkila Olli. Táto metóda sa dokázala vyrovnať po krátkom čase aj so zmenou svetlosti obrazu, prípadne s pootočením kamery. V takomto prípade sa pozadie pomaly prispôsobilo. Pri takto udržiavanom pozadí by nebol problém východ a západ slnka. Lenže v prípade ak by bola zmena dostatočne rýchla ako napríklad zájdenie či výjdenie slnka spoza mračien, potom by systém vyvolal falošný poplach. Preto som ani takto nedosiahol systém plne využiteľný v praxi.

Ďalší problém tejto metódy spočíva v tom, že ak je objekt výrazne odlišný od pozadia, potom sa pozadie mení rýchlo. Pri malých odlyšnostiach sa mení pomaly. To znamená že ak máme objekt výrazne odlyšný, ktorý prechádza cez scénu, nemusí v nej ani stáť na mieste, a aj tak spôsobí, že sa pozadie odlýši od skutočnosti dostatočne na to, aby sa aj po odchode objektu ešte detekoval na tomto mieste objekt. Ale keďže odlyšnosť od pozadia tu nieje taká veľká, ako keď na tom mieste bol v obraze objekt, vracia sa pozadie do pôvodného stavu pomaly. Teda sa na tomto mieste udrží falošná detekcia relatívne dlho.

Metóda čiastočného obnovovania pozadia fungovala dobre, pokiaľ bol obraz segmentovaný správne. Ale v prípade že sa len pár pixelov z nejakého objektu dostali do pozadia, systém sa s tým už nevedel vyrovnať. Je to kôli tomu že potom už miesto kde sa tak stalo bolo stále nesprávne detekované ako objekt a teda sa už nikdy nekopírovalo. Problém s prenikaním pixelov pohybujúceho sa objektu do pozadia nastával pri pomalom príchode objektu na scénu. V reálnych scénach by bol tento princíp úplne nevhodný, pretože by sa v ňom pri každej zmene osvetlenia spustil poplach a falšná detekcia by sa odstránila až

slepým skopírovaním aktuálneho obrazu do pozadia. V podstate môžeme povedať že pre kamerový bezpečnostný systém je už lepšie aj slepé obnovovanie pozadia ako tento princíp. Nebolo by však dobré podobné myšlienky zavrhnúť preto že nie je vylúčené, že by takýto princíp mohol fungovať s určitými úpravami či vylepšeniami dobre.

Pre použiteľnosť tejto aplikácie musia byť vyriešené nasledujúce problémy:

- Vyriešiť problém falošných poplachov pri rýchlich zmenách osvetlenia scény.
- Ako zaručiť aby bol objekt detekovaný s rovnakou citlivosťou pri rôznych osvetleniach.
- Oslobodiť aplikáciu od nutnosti nastavovať ju manuálne pre rôzne podmienky aké nastávajú počas dňa.

Kapitola 7

Námety pre budúcu prácu

Ako som už spomenul v predchádzajúcej kapitole, táto aplikácia v podobe do akej som ju zatial dotiahol nieje vôbec vhodná pre použitie v praxi. Ukazuje sa, že reálne situácie v sebe skrývajú mnoho problematyckých situácií pre podobné aplikácie. Preto bude potrebné vykonať veľa práce v budúcnosti aby sa dosiahli adekvátne výsledky v praktickom použití.

Bolo by možné pokračovať v upravovaní už implementovaných metód, ale niesom presvedčený, že by to niekam viedlo.

Ako sľubná metóda sa zdá byť metóda spomenutá v [6]. V článku je uvedené že systém založený na tomto princípe bežal takmer nepretržite počas 16-tich mesiacov a vyrovnal sa aj s dažďom a snežením. Jedná sa o princíp pri ktorom sa pixeli modelujú pomocou gausiánov a používa sa "on-line" aproximácia na obnovovanie modelu.

Ďalšia metóda ktorú by som v budúcnosti rád vyskúšal je metóda "Non-parametric background subtraction", ktorá bola publikovaná v [2]. Jedná sa o metódu ktorá je schopná odfiltrovať prudké zmeny jednotlivých pixelov spôsobené napríklad pohybom lístia na stromoch.

Kapitola 8

Záver

V tejto práci som implementoval systém pre sledovanie objektu či areálu kamerou pripojenou k počítaču. Účelom je spustiť alarm keď sa narušiteľ dostane do obrazu a označiť ho v obraze. Dosiahnuté výsledky zatiaľ nie sú uspokojivé a teda sa tu otvára veľký priestor pre budúcu prácu. Je ešte veľa iných prístupou ktoré sa dajú využiť. Je tu aj veľký priestor pre vylepšovanie pretože dokonalý systém zvládajúci všetky situácie čo môžu nastať ešte zrejme neexistuje. Okrem toho téma tejto práce je v podstate základ aj pre väčšinu komplikovanejších systémov strojového videnia.

Literatura

[1] Opencv reference manuals. http://public.cranfield.ac.uk/soe/c5354/teaching/dip/opencv/manual/.

[2] Larry Davis Ahmed Elgammal, David Harwood. Non parametric model for background subtraction. Technical report, Computer Vision Laboratory, University Of Maryland.

[3] Janne Heikkilä and Olli Silvén. A real-time system for monitoring of cyclists and pedestrians. www.ee.oulu.fi/mvg/files/pdf/pdf_156.pdf, 2003.

[4] R. Boyle. M. Sonka, V. Hlavac. *Image processing analyzis and machine vision.* Brooks/Cole Publishing Company, 1999. ISBN 0-534-95393-X.

[5] Alan M. McIvor. Background subtraction techniques. www.mcs.csuhayward.edu/~tebo/Classes/6825/ivcnz00.pdf.

[6] Chris Stauffer and W.E.L Grimson. Adaptive background mixture models for real-time tracking. Technical report, The Artificial Intelligence Laboratory, Massachusetts Institute Of Technology.

Zoznam príloh

1. CD s programom a ukážkovým videom.