VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

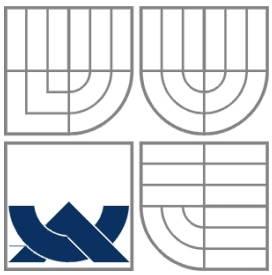
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DETEKCE POHYBU VE VIDEO SEKVENCI

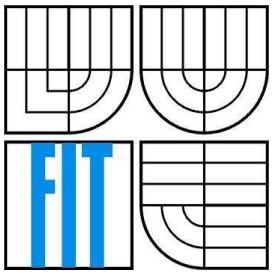
BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE PETR CHADIM
AUTHOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DETEKCE POHYBU VE VIDEO SEKVENCI

MOTION DETECTION IN VIDEO SEQUENCES

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — PETR CHADIM
AUTHOR

VEDOUCÍ PRÁCE — ing. MIROSLAV ŠVUB
SUPERVISOR

BRNO 2008

Detekce pohybu ve video sekvenci

Motion Detection in Video Sequences

Vedoucí:

Švub Miroslav, Ing., UPGM FIT VUT

Oponent:

Štancl Vít, Ing., UPGM FIT VUT

Zadání:

1. Prostudujte základy zpracování obrazu. Zaměřte se na detekci pohybu v obrazové sekvenci.
2. Zorientujte se v metodách analýzy pohybu v obraze a detekce pohybujících se objektů.
3. Vyberte vhodnou metodu a navrhněte postup detekce.
4. Experimentujte s vaší implementací.
5. Případně navrhněte vlastní modifikace metod.
6. Diskutujte dosažené výsledky a možnosti budoucího vývoje.
7. Vytvořte stručný plakát prezentující vaší bakalářskou práci, její cíle a výsledky.

Kategorie:

Počítačová grafika

Implementační jazyk:

C/C++, C# nebo Python

Operační systém:

MS Windows, Linux (pokud možno přenositelný kód)

Literatura:

Dle pokynů vedoucího

Licenční smlouva

Licenční smlouva je uložena v archivu Fakulty informačních technologii Vysokého učení technického v Brně.

Abstrakt

Práce pojednává o problému detekce pohybu ve video sekvenci. Detailně je popsán základní princip metody *Odečítání pozadí* a algoritmus pro hledání rozdílů mezi snímky. Velká pozornost je věnována aktualizaci referenčního snímku. Součástí práce je lokalizace pohybujícího se objektu ve video sekvenci. Výsledkem je aplikace pro otestování práce detektoru v různých prostředích.

Klíčová slova

Detekce pohybu, počítačové vidění, zpracování obrazu, obrazové filtry, video sekvence, zpracování videa, OpenCV, odečítání pozadí, pozadí, popředí, referenční snímek.

Abstract

This thesis deal with a problem of motion detection in video sequence. The basic principle of method *Bacground subtraction* and the algorithm for searching differences between frames were described in details. Great attention is made for actualization of referential frame. Part of this thesis is a localization of motion object. The result of this thesis is an application for testing of motion detector in different environments.

Keywords

Motion detection, computer vision, picture processing, image filters, video sequence, movie processing, OpenCV, background subtraction, background, foreground, referential frame.

Citace

Chadim Petr: Detekce pohybu ve video sekvenci. Brno, 2008, bakalářská práce, FIT VUT v Brně.

Detekce pohybu ve video sekvenci

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Miroslava Švuba. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Petr Chadim
11.5.2008

Poděkování

Chtěl bych na tomto místě poděkovat za cenné rady, ochotu a dobré vedení především svému vedoucímu bakalářské práce Ing. Miroslavu Švubovi.
Dále pak děkuji své rodině a přátelům za velmi důležitou psychickou podporu a zázemí.

Obsah

Úvod

V dnešním sofistikovaném světě je lidská populace víceméně již závislá na informačních technologiích. Dalo by se říci, že běžný, řekněme civilizovaný, člověk využívá nejrůznějších IT produktů denně. Detekce pohybu je rozhodně netriviální problém, jímž se informatici zabývají několik posledních let. Požadavek na schopnost zjišťovat pohyb byl vznesen současně s masovým rozšířením kamerových systémů, ať už bezpečnostních či jakýchkoliv jiných.

V zabezpečené bance bude jistě vhodné nějak reagovat na případný pohyb v noci a spustit alarm, stejně tak v případě kamery provádějící statistické výpočty (např.: dopravní zatíženost pozemních komunikací) je nutné pohybující se objekt započítat. Dále je také užitečné pohybující se objekt v obraze nalézt a vhodným způsobem jej označit. Trajektorii pohybu je pak možné sledovat.

Pokud kamera střeží vyhrazený prostor (dejme tomu parkoviště), vhodnou reakcí na případný pohyb může být spuštění nahrávání. Výsledkem bude video složené z krátkých záznamů, na kterém pak člověk snadno rozpozná, zda se v noci mezi auty procházela kočka nebo zloděj.

Toto vše je jen několik vyjmenovaných oblastí využití detekce pohybu z mnoha dalších. Každá z nich pak klade více či méně odlišné nároky, priority, požadavky na přesnost, komplexnost a rozsáhlost určování pohybu. Detektor pohybu bude správně pracovat jen v podmínkách, pro které byl postaven. Bude se správně chovat pouze v situacích a prostředích, pro něž byli nastaveny všechny potřebné parametry při jeho vývoji, popřípadě při instalaci.

Můžeme si ale položit otázku: Když tedy musíme předvídat situace, do kterých se detektor může dostat, aby byl schopen dobře pracovat, bude umět správně reagovat na situace nové a nepředvídané, které dříve či později zřejmě nastanou?

Člověk si musí uvědomit, že lidské vidění funguje trochu jinak než počítačové. Při lidském vidění po zachycení světelného signálu čidlem zraku je obraz zpracován (ještě před uložením do paměti) jednotkou, která snímaný obraz roztřídí na objekty, nebo podstatné celky. Děje se tak podvědomě a na základě zkušeností a poznatků o okolním světě. Nicméně počítač (v roli detektoru) vnímá obraz jako jeden celek, tudíž musíme nalézt jiný způsob jak rozeznat důležitou informaci od nepodstatné. Bude tedy vždy hlavní snahou obsáhnout co nejvíce situací a „naučit“ detektor na ně správně reagovat. Bohužel ale, stejně jako nikdy nelze uvážit všechny možné případy, nebude detekce pohybu nikdy stoprocentně úspěšná.

Musím ještě upřesnit, že cílem této práce není řešit detekci pohybu s využitím umělé inteligence, která se naopak o napodobení lidského vnímání světa snaží. Tuto oblast tedy nebudu vůbec uvažovat.

Stručný přehled kapitol

Nyní, po jemném zasvěcení do problematiky, bych stručně popsal, čím se v jednotlivých kapitolách budu zabývat.

První kapitola *Základy zpracování obrazu* se věnuje základním technikám a operacím s obrazem v elektronické počítačové reprezentaci. Hlouběji se věnuje filtraci obrazu a pokládá tak teoretický základ pro tuto práci.

Kapitola následující s názvem *Metodika detekce pohybu* pojednává o vybrané metodě detekce pohybu *Odečítání pozadí* a s ní spojené její vzniklá omezení. Vysvětluje základní princip metody a dvě varianty pro samotné odečítání pozadí.

Ve třetí kapitole *Návrh aplikace* je již detailně popsána struktura navrženého detektoru a v jednotlivých krocích vysvětlen postup celé detekce. K názornosti přispívá blokový diagram a obrázek výsledné aplikace. Podrobně jsou rozebrány algoritmy pro odečtení pozadí, aktualizaci referenčního snímku a ohraničovací algoritmus pro lokalizaci pohybu ve snímku. V závěru kapitoly je naznačeno navržené uživatelské rozhraní aplikace.

Čtvrtá kapitola *Implementace* líčí podmínky, ve kterých byla aplikace vyvíjena. Zmiňuje zvolený programovací jazyk, použité knihovny a vývojové prostředí. Velice zhruba je naznačen objektový návrh aplikace.

Testování je nadpis páté kapitoly, která se snaží navržený detektor pohybu vyzkoušet v praxi. Detektor byl nasazen ve dvou diametrálně odlišných prostředích. Jsou zde diskutovány problémy vznikající v testovacích podmínkách a naznačen správný způsob uvažování při nastavování parametrů detekce. Výsledky experimentování doplněné o snímky z jeho průběhu jsou odborně zhodnoceny. Ke konci kapitoly je naznačena správná strategie při aplikaci filtrů na masku a nakonec jsou sepsány hardwarové a softwarové podmínky při testování.

V závěru je shrnut celkový přínos práce, nabyté poznatky při řešení a vlastní navržená vylepšení a rozšíření práce. Pokládám zde návrhy na možná pokračování této práce.

1 Základy zpracování obrazu

Tato kapitola si klade za cíl osvětlit základní metody zpracování obrazu, zmínit několik nejjednodušších operací, které s obrazem můžeme provést, poskytnout teoretický základ pro tuto práci. V kapitolách níže již bude s těmito znalostmi počítáno. Více o této problematice je možno nalézt v [1].

1.1 Obraz a jeho reprezentace

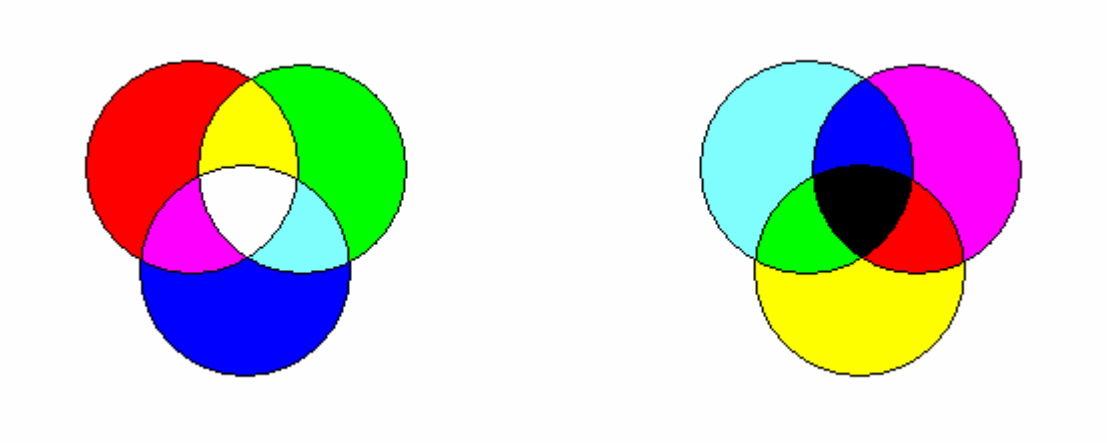
Obraz je obecně definován jako obrazová spojitá funkce dvou proměnných. Tyto fungují jako souřadnice do dvourozměrného prostoru, kde každý bod může nabývat jedné nebo více hodnot. V počítačové problematice má obraz omezený definiční obor a říká se mu **rastr**. Jednotlivé body rastru se pak nazývají **pixely**. Při převodu reálného obrazu do počítačového (při **digitalizaci**) vlastně redukujeme nekonečné rozlišení a nekonečný počet barev na konečné hodnoty dle zvoleného barevného modelu.

1.1.1 RGB model

Každý pixel barevného obrazu je složen ze tří základních barev - červené, zelené a modré. Každá z těchto složek může nabývat různé intenzity a díky aditivnímu součtu těchto složek můžeme získat jakoukoliv barvu. Pro upřesnění - při uložení v paměti po 8 bitech pro jednu složku může barva nabývat 256 hodnot, pro 3 složky tedy 256 * 256 * 256 = 16 milionů barev, což bude i náš případ. Lidské oko tolik barev ani nedokáže rozlišit.

Při této obrazové reprezentaci budeme mít pro každý pixel tedy 3 kanály, tj. celkem 24 bitů. V této práci budeme muset při detekci obraz několikrát procházet po jednotlivých pixelech, proto je pro nás toto uložení dosti nevýhodné.

Obrázek 1 znázorňuje rozdíl mezi aditivním a subtraktivním skládáním barev. Zatímco u prvního způsobu nám při plné intenzitě všech složek vzniká bílá barva, v případě druhém je výsledkem barva černá. Aditivní součet funguje při práci se světlem, tedy ho využíváme v počítačové grafice. Subtraktivní součet lze aplikovat při míchání barev v tiskárnách a jedná se o model CMY (tyrkysová, purpurová, žlutá).



Obrázek 1: **Barevný model RGB (vlevo) a model CMY (vpravo).**

1.1.2 Šedotónový obraz

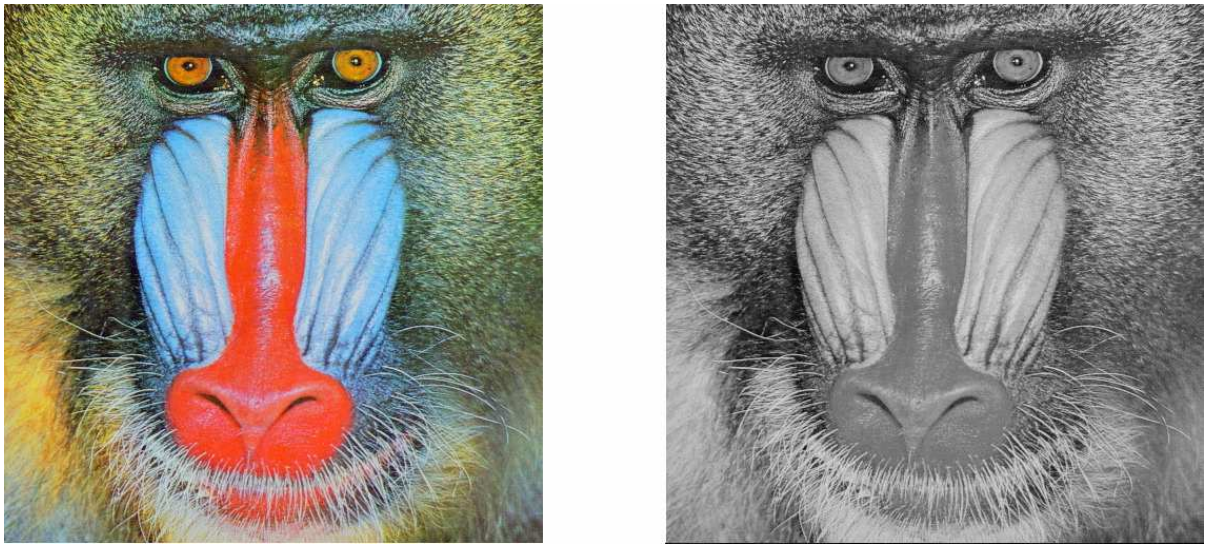
Barevná informace obrazu může být redukována tak, že bude obraz reprezentován pouze v omezeném počtu stupňů šedi. Při této operaci (operace **Grayscale**) se využívá skutečnosti, že jakýkoliv šedivý odstín je výsledkem součtu složek R, G a B, které mají všechny stejnou intenzitu. Při 8-bitovém uložení v paměti máme tedy pouze 256 odstínů šedi.

Pro reprezentaci 256 hodnot nepotřebujeme pro pixel tři kanály, jak tomu bylo u RGB modelu, ale postačí nám kanál pouze jeden. Při hledání pohybu v obraze barevnou informaci nebudeme potřebovat, a tak velice uvítáme, když nebudeme muset procházet 3 barevné složky v každém pixelu.

Šedá barva (neboli intenzita jasu) je vypočtena z původně tří barevných složek podle následujícího vztahu 1, který zohledňuje různou citlivost lidského zraku na jednotlivé barevné složky. Nejvíce je podpořena zelená barva, nejméně pak barva modrá. Výsledný obraz se pak jeví našemu oku naprosto stejně, pouze bez barevné informace (viz obr. 2).

$$I = 0{,}299 * R + 0{,}587 * G + 0{,}144 * B$$

Vztah 1: **Výpočet intenzity jasu pixelu.**



Obrázek 2: Převod 3-kanálového RGB obrázku na 1-kanálový v 256 šedých odstínech.

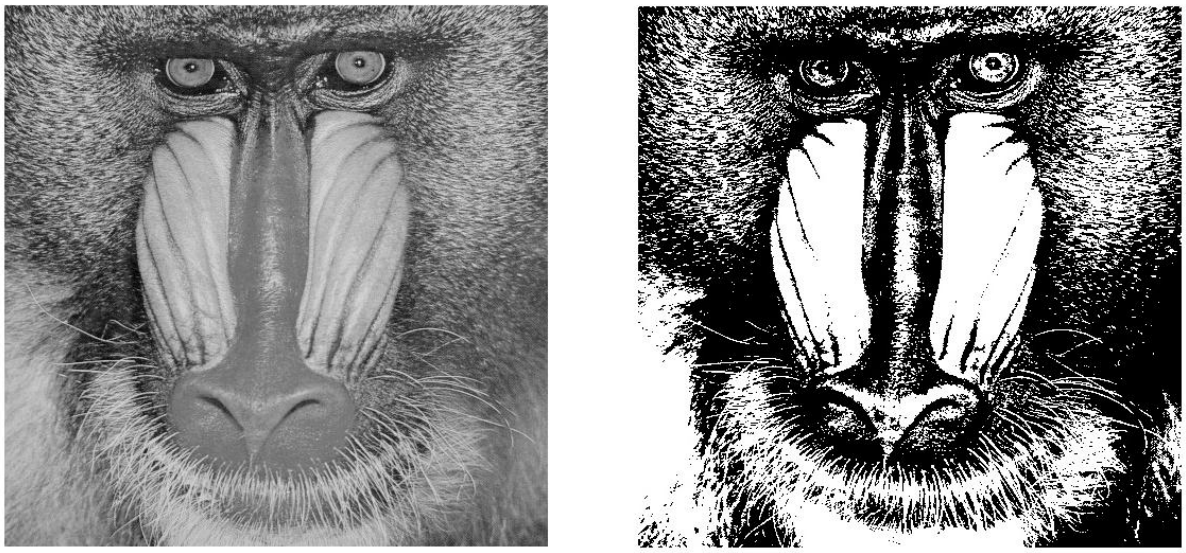
1.2 Filtrace obrazu

Obrazové filtry slouží k úpravě obrázku tím, že redukují nějakou nežádoucí nebo pro danou situaci nepodstatnou obrazovou informaci. Jak jsem již zmínil výše, pro nás je velice důležité, abychom obraz před samotným detekčním algoritmem uvedli do co nejskromnějšího stavu při zachování podstatné informace. Možná bych mohl zařadit mezi filtry i operaci **Grayscale**, která nám tím, že převede obrázek na 256 stupňů šedi, sníží výkonové nároky při detekci asi nejvíce. V této kapitole uvádím také morfologické operátory eroze a dilatace, ačkoli nejsou řazeny mezi obrazové filtry.

My budeme dále uvedené filtry využívat především k odstranění velice nežádoucí informace v obraze a tou je **šum**. Ten je definován jako nová informace, která byla k původní přidána v důsledku nedokonalosti záznamových zařízení, nebo při transportu a digitalizaci. Musíme se bohužel smířit s tím, že šum úplně odstranit nelze, ačkoliv vhodnou aplikací filtrů ho můžeme většinou dostatečně redukovat.

1.2.1 Prahování

Filtr Prahování (angl. *Thresholding*) je většinou aplikován na šedotónový obrázek. Odstraní z něho veškerou informaci o barevných (šedých) odstínech a ponechá jen neúplné obrysy. Výsledkem bude monochromatický, nejčastěji černobílý obrázek. Před samotnou aplikací je třeba zvolit práh, s kterým budeme porovnávat všechny pixely v obrázku. Při jeho překročení nastavíme hodnotu maximální (bílá barva) a v ostatních případech přidělíme hodnotu minimální (černá barva). Abychom co nejvíce zachovali uspořádání scény v obrázku, je třeba pro každý snímek volit vhodný práh (viz obr. 3).



Obrázek 3: Ukázka filtru Prahování s nastaveným prahem 135.

1.2.2 Medián

Mediánový filtr pracuje tak, že vyhledá ve zvoleném okolí (nejčastěji 3x3) střední hodnotu pixelu a na tuto hodnotu aktuální pixel nastaví. V našem případě budeme mediánový filtr aplikovat pouze na monochromatický černobílý obrázek, bude tedy z okolí pixelu vybrána nejčastější hodnota. Filtr poměrně úspěšně redukuje šum, avšak drobně obrázek rozmazává. To ovšem černobílému obrázku vadit nebude a uvítáme odstranění osamocených bílých pixelů způsobených šumem (viz obr. 4).

Obrázek 4: Ukázka filtru Medián s okolím 3x3.

1.2.3 Eroze

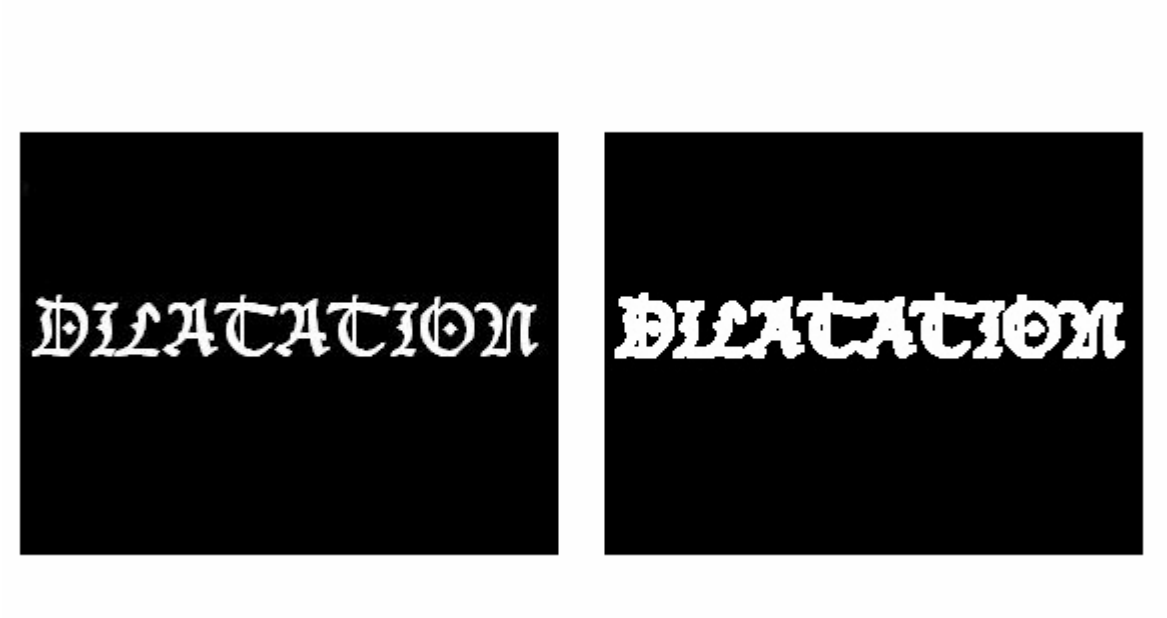
Erozní filtr pracuje podobně jako mediánový, bývá však zpravidla aplikován na obrázek monochromatický, což bude náš případ. Také počítá výslednou barvu pixelu dle jeho okolí. Nehledá však hodnotu střední, nýbrž hodnotu nejmenší. Ve výsledku to vypadá tak, že erodováním obrázek jemně ztmavíme.

V mé aplikaci tedy vedle opět dobré schopnosti odstranit bílé pixely šumu bude kromě zaostřování celistvých bílých oblastí tyto oblasti trochu zmenšovat (viz obr. 5). Tuto drobnou nevýhodu jsme ovšem schopni akceptovat, neboť na druhou stranu uvítáme celkovou konkrétnost a jednoznačnost obrázku, čímž se zvýší úspěšnost správného odlišení pohybu od statického pozadí.

Obrázek 5: Ukázka Eroze.

1.2.4 Dilatace

Dilatace pracuje na naprosto stejném principu jako Eroze. Nevyhledává však nejmenší hodnotu v okolí pixelu, nýbrž hodnotu největší. Bude tedy obrázek naopak trochu zesvětlovat. Hledané objekty v obraze budou poněkud rozostřeny a zvětšeny, jak vidíme na následujícím obrázku 6. Je tedy vhodné aplikovat Dilataci po Erozi nebo Mediánu. Narušený tvar objektů uvede Dilatace v rámci možností zpět do původního tvaru, nicméně šumová informace se již nevrátí.



Obrázek 6: **Ukázka Dilatace.**

2 Metodika detekce pohybu

Způsobů, jak přistupovat k problematice detekce pohybu, je bezpochyby více než mnoho. Nyní již velice záleží na požadavcích, které budou na náš detektor vzneseny, a samozřejmě na samotné funkci, kterou by měl daný detektor být schopen vykonávat. V této části práce si již dovolím toto široké spektrum možností přístupu k problému zúžit a specializovat se pouze na jedno z možných zaměření v rozsáhlé oblasti detekce pohybu.

2.1 Metoda odečítání pozadí

Do této chvíle jsem ještě nenaznačil princip, na jakém by mohla detekce pohybu pracovat. Metoda odečítání pozadí (angl. *Background Subtraction*) jistě patří k nejjednodušším a se svojí efektivitou a úspěšností na tom není přitom vůbec špatně. Informace pro tuto kapitolu jsem čerpal z [2], [3], [4], [5], [6].

2.1.1 Princip metody

Základním principem této metody je pokusit se odlišit od sebe tzv. *popředí* a *pozadí* jednotlivých snímků ve zpracovávané video sekvenci.

Pozadí snímku je definováno jako statická scéna (prostředí), ve které nejsou žádné pohybující se objekty, respektive žádné objekty, které by měli být vnímány jako pohyblivé. Když detektorem projde obrázek, který je složen pouze z pozadí, nebude na něm detekován pohyb.

Popředí je pravý opak pozadí. Je tvořeno všemi pohybujícími se objekty, které netvoří pozadí snímku.

Způsob jak metoda odlišuje popředí a pozadí snímku spočívá, jak už sám název metody naznačuje, v odečtení pozadí od zpracovávaného snímku. Pozadí snímku je pevně určeno již od počátku detekce a tvoří tzv. **referenční snímek**. Referenční snímek je tedy pořízen v okamžiku před spuštěním detekce a to v době kdy je scéna vnímána jako „v klidu“. To znamená, že určuje jak zkoumané prostředí vypadá, když se nic neděje a detektor nic nehlásí. Naopak, jakmile je nalezena na aktuálně zpracovávaném snímku nějaká odlišnost, měl by ji detektor rozpoznat jako pohyb.

Náš algoritmus bude tedy velice zhruba pracovat tak, že bude načítat jednotlivé snímky video sekvence a odečítat od každého referenční snímek, který si vhodným způsobem na začátku pořídí. Jakmile výsledkem po odečtení (uvažujeme ideální případ) nebude celý „černý“ obrázek, bude detekován pohyb. Černý obrázek nám vyjde v případě, že je aktuální snímek shodný s referenčním, tedy že je složen pouze z pozadí. Po odečtení shodných hodnot pixelů vyjde hodnota 0, což se projeví jako černá barva.

Na obrázku 7 vidíme příklad odečtení pozadí (snímek uprostřed) od aktuálního snímku (snímek vlevo). Výsledkem je v ideálních podmínkách černý snímek s popředím (snímek vpravo).

Obrázek 7: Odečtení pozadí od aktuálního snímku - výsledkem je samotné popředí.

2.1.2 Omezení metody

Metoda odečítání pozadí si jako daň za svou jednoduchost bere jisté omezení v její využitelnosti. Její zásadní nevýhoda spočívá v tom, že pro svou správnou funkčnost předpokládá statické pozadí. Skutečnost, že je referenční snímek pevně spjatý se scénou, kterou kamera snímá, zamezuje sebemenší změně zorného úhlu kamery nebo jiného snímacího zařízení. Při případném pohybu kamery bychom museli pořídit nový referenční snímek, který bychom mohli znovu považovat za statické pozadí.

Mluvím zde zatím o této metodě v její základní výchozí podobě. Celá detekce je tak závislá na kvalitě pořízeného referenčního snímku před samotným startem detekčního algoritmu. V takovémto primitivním provedení je metoda náchylná na sebemenší změny v pozadí scény. Mimo uzavřenou místnost se stálým osvětlením a nehybným prostředím téměř nepoužitelná. Tuto podstatnou nevýhodu lze dostatečně zredukovat tím, že budeme referenční snímek nějak vhodně průběžně aktualizovat. Jak k tomuto problému přistupovat samozřejmě opět závisí na prostředí, kde bude kamera umístěna a k jakému účelu bude detekce využívána. Zda bude vevnitř, venku, ve stínu, zda bude snímat část oblohy (plující mraky) nebo park (pohyb stromů), zda bude plnit zabezpečovací funkci nebo provádět statistické výpočty a nesčetně dalších vlivů. Celou záležitostí ohledně problému s referenčním snímkem a možnými přístupy k němu se budu zabývat hlouběji dále ve své práci. Ovšem bohužel nic z toho, co je zde zmíněno nezmění na tom, že vstupem do detektoru pohybu, postavenému na této metodě, musí být signál z **nehybné kamery**, sledující stále stejnou scénu.

2.2 Algoritmy odečítání pozadí

Výše jsem vysvětlil základní princip metody odečítání pozadí. Nyní se stručně podíváme na dva vybrané způsoby, jakými lze samotné odečítání provádět. Každý z nich nabízí jiné možnosti a chová se více či méně úspěšně v různorodých situacích.

2.2.1 Porovnávání jasového histogramu

Tento algoritmus bude patrně nejprimitivnější ze všech. Spočívá v porovnávání jasové složky aktuálního snímku s jasovou složkou referenčního snímku. Nejprve se vypočte tzv. **histogram** obrázku. Ten může být implementován jako jednorozměrné pole, kde počet prvků bude dán počtem možných hodnot celkového jasu jednotlivých pixelů (případně můžeme určit intervaly jasových hodnot). Každý prvek pole histogramu nabývá takové hodnoty, kolik je ve snímku pixelů s příslušnou hodnotu jasu. Jasová složka může být vypočtena stejným způsobem, jako když provádíme operaci Grayscale. Nebo pokud pracujeme s šedotónovým snímkem, máme jasovou složku vlastně již vypočtenu.

Nyní budeme pouze od histogramu aktuálního snímku, který vždy pro každý snímek vypočteme, odečítat histogram referenčního snímku. Ten vytvoříme již na začátku. Výsledkem bude histogram, který bude uchovávat pouze počty odlišných pixelů od referenčního snímku, tedy pixelů tvořících případný pohyb (tzn. i šum). Tyto hodnoty sečteme a porovnáme s uživatelem definovanou hranicí. Pokud je hranice překročena, je detekován pohyb a např. spuštěn alarm.

Algoritmus je v jeho základní podobě náchylný na nepatrné změny v pozadí, což by se opět dalo eliminovat vhodnou aktualizací referenčního snímku, respektive jeho jasového histogramu. Můžeme si dovolit plýtvat výkonem i na aplikaci nejrůznějších filtrů pro odstranění šumu, poněvadž je algoritmus velice rychlý.

Je třeba ovšem zmínit ještě jednu podstatnou nevýhodu. Algoritmus v základní podobě neposkytuje možnost, jak pohybující se objekt ve snímku nalézt. Pracuje totiž pouze s informací o počtech pixelů se stejnými intenzitami jasu. Bude tedy vhodný pro využití na místech, kde postačí upozornit, že byl případný pohyb zjištěn.

2.2.2 Porovnávání jednotlivých pixelů

Algoritmus, který porovnává snímky po jednotlivých pixelech, poskytuje možnost bez nutnosti dalších rozšíření pohybující se objekty v obrázku zaměřit. Pracuje totiž po celou dobu se snímky v původním tvaru. Bude tedy klást podstatně větší nároky na výkon než algoritmus předchozí. Je proto namístě zabývat se tím, jak bychom mohli průchod zpracovávaným snímkem co nejvíce urychlit.

Asi základním krokem, který také budu využívat při implementaci, je aplikace operace Grayscale. Jestliže případným pohybem na snímku nebude pouze změna barvy, nebude nám jistě vadit, když o barevnou informaci přijdeme. Jak jsem již uváděl výše, v případě tříkanálové reprezentace snímku bychom museli porovnávat všechny tři barevné složky, nýbrž u šedotónového obrázku nám postačí porovnat v každém pixelu pouze jednu hodnotu.

Kromě tohoto výrazného urychlení můžeme přijít ještě na některá další, která jsem ale ve své práci již nepoužil. Nemusíme například ve většině případů pracovat se snímky v jejich plném rozlišení. Pokud třeba vynecháme každý druhý řádek nebo sloupec v rastru, snížíme tak počet zpracovávaných pixelů na polovinu a základní uspořádání scény se většinou přitom nijak významně nenaruší. Můžeme natvrdo upravit rozlišení obrázku nebo pouze upravit algoritmus tak, aby vybrané řádky nebo sloupce přeskakoval. Zde však samozřejmě záleží na konkrétních případech nasazení detektoru. Vždy je třeba se dostat k výhodnému poměru mezi rychlostí a úspěšností algoritmu.

Tento algoritmus nepatří k nejrychlejším, ale kromě užitečné schopnosti lokalizovat pohyb si dokáže poměrně dobře poradit se i šumem. Zároveň s tím však vzniká nutnost pro každou situaci nastavit vhodný práh pro vytvoření masky a zvolit správnou kombinaci dalších filtrů. Avšak spolu s dobře vyřešenou průběžnou aktualizací referenčního snímku se může chovat tento algoritmus velice účinně a efektivně.

3 Návrh aplikace

Aplikace byla navržena pouze jako ukázka jednoho z mnoha dalších způsobů řešení. Má velice primitivní uživatelské rozhraní. Důraz byl kladen především na funkčnost a efektivnost samotného detekčního algoritmu.

Snažil jsem se o schopnost detektoru úspěšně pracovat v různých prostředích. Proto je uživateli umožněno nastavit detektor sadou přepínačů a hodnot, kterými se algoritmus řídí a jimiž je možno ovlivnit mnoho faktorů - od dovednosti odolávat šumu přes nastavení hranice jemnosti detekce až ke schopnosti vypořádat se s drobnými změnami v pozadí snímku. Tímto nastavením lze obsáhnout relativně široké spektrum situací, jež mohou zpracovávané video sekvence skýtat.

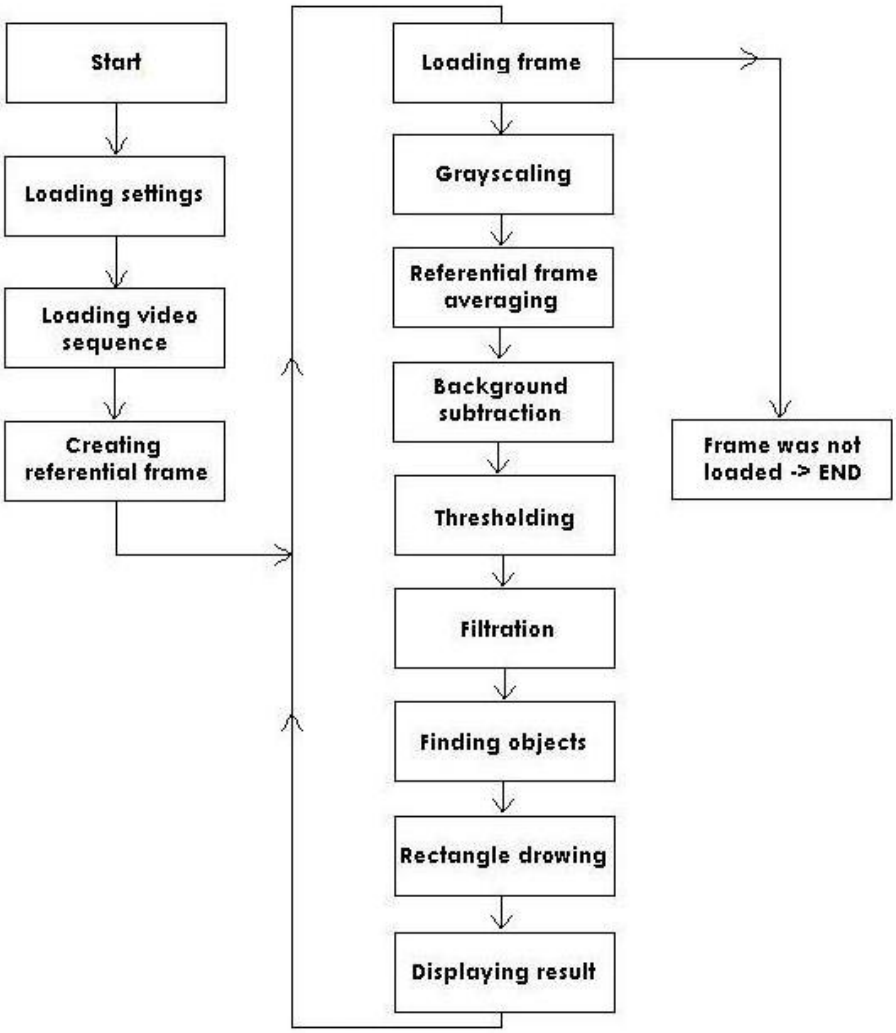
Instinktivně jsem vybral asi nejvíce názornou variantu odečítání pozadí - porovnávání jednotlivých pixelů mezi snímky. Především proto, že bez nutnosti dalších rozšíření dokáže pohyb ve snímku lokalizovat, což je pro účel této práce více než vítané.

3.1 Postup detekce

Zde popíši sled kroků, které jsou prováděny od chvíle spuštění detekce do jejího ukončení. K názornosti přispívá blokový diagram detektoru na obrázku 8.

Prvním krokem je načtení video sekvence. Aplikace dokáže zpracovat video sekvenci získanou z uloženého souboru nebo z připojené kamery. Zdroj je opět možno zvolit, uživatel si tak může vyzkoušet detekci v reálném čase.

V druhém kroku je nutno obstarat referenční snímek pro samotnou detekci. Ve svém řešení jsem tento úkol plně zautomatizoval. Využiji hned jeden z prvních snímků video sekvence. V praxi by asi bylo vhodnější tento krok ponechat v režii uživatele, případně osoby, která detektor instaluje. Referenční snímek je nutno pořídit v situaci, kdy je scéna nejvíce statická. Takovou situaci předpokládám právě v prvních snímcích zpracovávané video sekvence. V případě připojené kamery ještě vynechávám první dvě nebo tři desítky snímků. Počítám tak i s méně kvalitními kamerami, kterým trvá několik prvních sekund činnosti, než se zinicializují (zaostření, vyvážení bílé barvy, atd.). Na načtený referenční snímek aplikuji operaci Grayscale. Celá detekce totiž z úsporných důvodů vysvětlených výše pracuje pouze s šedivými snímky.



Obrázek 8: Blokový diagram detektoru.

Nyní když máme referenční snímek, můžeme spustit samotnou detekci, takže po inicializaci dalších konfiguračních hodnot následuje „nekonečný“ **detekční cyklus**. Cyklus bude probíhat v případě videa z kamery až do ukončení detekce (uzavření aplikace), nebo v případě videa ze souboru do načtení posledního snímku. Může být také ukončen v důsledku případné chyby v průběhu výpočtu.

Činnost detekčního cyklu bych shrnul v následujících osmi krocích:

1) načtení dalšího snímku z video sekvence

2) operace Grayscale na načtený snímek

3) případná aktualizace referenčního snímku zvolenou metodou

4) odečtení pozadí (referenčního snímku) od aktuálního - získání masky

5) filtr prahování - získání černobílé masky

6) aplikace dalších filtrů na černobílou masku ve zvoleném pořadí

7) vyhledání spojitých bílých oblastí v masce (pohybujících se objektů)

8) vyznačení pohybujících se objektů opsaným polygonem

K prvním dvěma bodům není třeba se více rozepisovat. Získaný snímek z video sekvence je ihned zkonvertován do šedých odstínů. Zbylým šesti bodům se podrobněji věnují následující podkapitoly.

3.2 Aktualizace referenčního snímku

Jak jsem již zmínil, vhodnou aktualizací referenčního snímku můžeme omezit náchylnost na drobné změny v pozadí, nejčastěji způsobené přírodními vlivy. Navrhnul jsem několik způsobů, jakými je možno aktualizaci provádět. Všechny jsou založeny na stejném principu, a to na průměrování několika po sobě jdoucích snímků. Ve své aplikaci poskytuji možnost volby mezi třemi způsoby průměrování. Každý z nich je vhodný jen pro některé účely nasazení detektoru.

3.2.1 Průměrování dvou po sobě následujících snímků

Toto řešení, jak už sám název napovídá, spočívá v získání nového referenčního snímku jako průměru mezi dvěma po sobě následujícími snímky. Oba snímky jsou procházeny po jednotlivých pixelech a z těch na stejných souřadnicích je počítán průměr. Uživateli je poskytnuta možnost ovlivnit v jak velkých periodách bude aktualizace uskutečňována. Perioda bude nastavena počtem snímků mezi jednotlivými aktualizacemi.

Tento způsob je vhodný pro detektory v prostředích, kde jsou běžné rychlé a větší nebo skokové změny v pozadí. Dokáže se s nimi poměrně svižně vyrovnat, poněvadž si vždy vytvoří zcela nový referenční snímek. Jeho použití je však omezeno na stále se pohybující objekty. Jakmile objekt přestane vykazovat pohyb, zanedlouho se stává součástí pozadí. Samozřejmě ne vždy to musí být nevýhodou. Hodí se tedy na detekci stálého a rychlejšího pohybu, čehož může být využito např. při sledování provozu na pozemních komunikacích.

3.2.2 Průměrování více po sobě následujících snímků

Toto řešení nebudu příliš rozvádět. Jedná se jen o vylepšenou verzi předchozího způsobu. Kromě frekvence (periody) jednotlivých aktualizací, je možno stanovit i počet průměrovaných, po sobě jdoucích snímků. Lze tedy lépe přizpůsobit konkrétní situaci. Použití zůstává stejné.

3.2.3 Chytré průměrování více po sobě následujících snímků

Zde uvádím jistou chytřejší variaci na předchozí dva způsoby. Opět se jedná o průměrování daného počtu snímků v daných periodách. Chytrost spočívá ve vynechání pixelů, které se účastní pohybu, z průměrování. Tím zamezíme splývání pohybujících se objektů s pozadím scény, a to i když se zastaví. Skutečnost, zda se pixel účastní či neúčastní pohybu zjišťujeme porovnáváním s úplně původním referenčním snímkem, který jsme si zálohovali ještě před samotným detekčním cyklem.

Pokud se pixel liší od původního na stejných souřadnicích o více než zadanou hodnotu (práh detekce - viz násl. kapitola), je místo něho započtena dosud průměrná hodnota pixelu.

Toto řešení bude mít dobré využití naopak v prostředích, které nevykazují markantní změny v pozadí. Jakákoliv drobná neplynulá změna bude chybně vyhodnocena jako popředí a vyvolá falešný poplach. Z této chyby se nejspíš detektor již sám nezotaví. Bude ale dobře pracovat v uzavřených místnostech nebo venkovních prostorech s velice pozvolnou změnou osvětlení. Navíc je schopen sledovat stále i objekty, které se při pohybu scénou na jakoukoliv dobu zastaví.

3.3 Odečtení pozadí a prahování

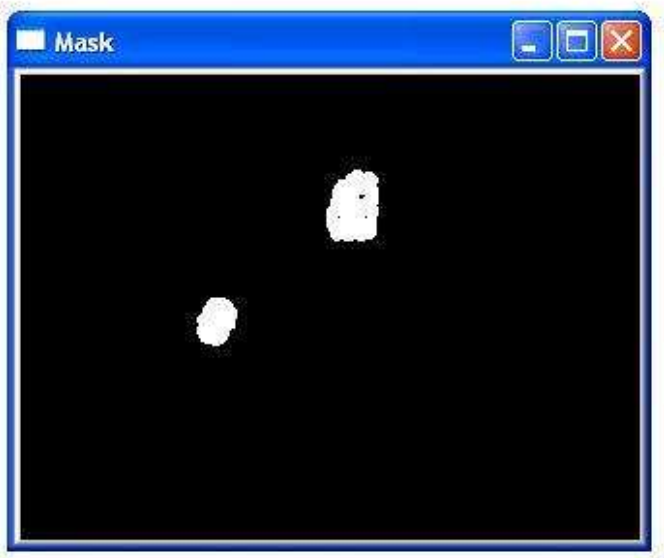
Odečítání pozadí funguje následovně. Procházíme aktuální snímek po pixelech a odečítáme od nich hodnoty pixelů referenčního snímku na stejných souřadnicích. Výsledky zapisujeme do třetího předem připraveného obrázku. Musíme si dát pozor, aby nám při odečítání nepřetekl rozsah, a ošetřit to třeba tak, že budeme brát absolutní hodnotu výsledku. Tímto postupem získáme nový obrázek, který může představovat **masku**, jejíž světlejší místa značí pohyb a tmavší naopak klid. V ideálním případě by vše ostatní kromě pohybu bylo černé, tedy s hodnotou 0. Ve skutečnosti však budou po celé masce neuspořádaně rozmístěny jednotlivé světlejší pixely (viz obr. 9). To je způsobeno kvůli neúplně statickému pozadí scény a také kvůli šumu. Nyní na vzniklou masku aplikujeme filtr prahování, čímž získáme masku černobílou. Práh, kterým se bude prahování řídit je zároveň **prahem celé detekce**. Uživatel jím bude moci nastavit, o jak velkou hodnotu se musí lišit porovnávané pixely, aby byl vyhodnocen pohyb. Při vhodně zvoleném prahu se můžeme zbavit i části šumové informace.

Obrázek 9: Maska po odečtení pozadí a prahování.

3.4 Aplikace dalších filtrů

Po prahování masky jsou v ní většinou ještě stále rozptýleny osamocené bílé pixely. Podstatně lepších výsledků dosáhneme aplikací dalších filtrů jako je Medián, Eroze a Dilatace. Jejich vhodnou kombinací můžeme získat masku, kde nehybná místa budou opravdu téměř celá černá a místa značící pohyb budou vypadat jako větší bílé „fleky“ (viz obrázek 10). Záleží však opět na konkrétní situaci. Vždy je třeba vyzkoušet více různých kombinací.

Moje aplikace umožňuje ve svém jednoduchém uživatelském rozhraní tuto filtraci řídit. Uživatel může dle svého uvážení zvolit aplikaci třeba pouze jednoho filtru, dvou různých filtrů, nebo všech tří filtrů a to vždy v jakémkoliv pořadí.



Obrázek 10: Maska po aplikaci dalších filtrů - konkrétně Medián a Dilatace.

3.5 Lokalizace pohybu

V této fázi by stačilo spočítat bílé pixely, porovnat je s uživatelem definovanou hranicí a ohlásit případný pohyb. Můžeme ale ještě vyznačit větší bílé oblasti a tím pohyb lokalizovat. Aplikujeme masku na původní snímek a pohybující se objekty ohraničíme třeba obdélníkem. Nejedná se o zcela triviální problém, proto se jeho řešení pokusím více rozepsat.

3.5.1 Rozbor a nástin řešení

Abychom mohli vykreslit obdélník do snímku, potřebujeme znát souřadnice dvou protilehlých bodů. Musíme tedy nejprve najít všechny bílé oblasti v masce (ty které stojí za povšimnutí) a pro každou vypočíst oba vrcholy pro opsaný obdélník. Nabízí se zde více možných způsobů řešení.

Například můžeme snímek rozdělit na zvolený počet menších oblastí a v každé spočítat bílé pixely. Když jejich počet přesáhne určitou hranici, tuto oblast označíme obdélníkem.

Dalším řešením může být postupné rekurzivní dělení napůl. Nejprve bychom rozdělili snímek na dvě oblasti a zjistili, zda se v některé z nich něco děje. Oblast, ve které by přesahoval počet bílých pixelů stanovenou hranici, bychom rozdělili opět na dvě části a znovu spočítali bílé pixely. Tak bychom postupovali do chvíle, kdy budou bílé oblasti lokalizovány s dostatečnou přesností, a vykreslili obdélníky.

3.5.2 Ohraničovací algoritmus

Můj algoritmus pro hledání bílých oblastí pracuje tak, že prochází černobílou masku po jednotlivých pixelech a v momentě, kdy narazí na bílý pixel, provede následující kroky. Od aktuálního pixelu pokračuje horizontálně vpravo dokud nenarazí na černý pixel. Když se tak stane, vrátí se na původní pozici a stejný postup opakuje vertikálně dolů. Tímto jsme získali pomyslný obdélník, jehož souřadnice si uložíme, pokud stojí za povšimnutí. To znamená pokud má větší rozměry, než je uživatelem stanovená hranice - **jemnost detekce**. Nyní přesuneme pozornost na první černý pixel, na nějž jsme narazili při horizontálním postupu a celý cyklus se opakuje - hledá se další bílý pixel.

Když tento cyklus skončí, zjistíme, že máme uloženy souřadnice třeba několika set obdélníků. Drtivá většina z nich však sdílí část nebo dokonce celou svoji plochu s ostatními. Je to způsobeno tím, že jsme hledali možné obdélníky na každém řádku v rastru snímku. Pro jednu bílou oblast bylo tedy vygenerováno více obdélníků. Velice přibližně by se dalo říct, že každému řádku, přes který bílá oblast sahá, přísluší jeden obdélník. Ve skutečnosti jich bude zpravidla více a to v závislosti na míře členitosti a nedokonalosti obrysu bílé oblasti v masce.

Musíme se tedy postarat o redukci zbytečných obdélníků a vyhledání, případně vypočítání, těch, které opíší celou bílou oblast. Budeme tedy procházet všechny uložené obdélníky a po dvou mezi sebou porovnávat jejich pozice. Jestliže zjistíme, že se obdélníky z této dvojice protínají, vnořují, nebo spolu blízko sousedí, ponecháme z nich pouze jeden a jeho rozměry upravíme tak, aby zahrnoval plochu obou původních obdélníků. Takto upravené „velké“ obdélníky si budeme pamatovat. Další dvojici utvoříme vždy z jednoho velkého (již upraveného) obdélníku a z nějakého dalšího. Každý následující obdélník se musí ocitnout ve dvojici vždy se všemi dosud vytvořenými velkými obdélníky. V případě, že jsou obdélníky z dané dvojice od sebe ve snímku vzdáleny více než

je uživatelem stanovená hranice, ponecháme je oba jako velké, poněvadž se nejspíše jedná o dva různé pohybující se objekty.

Když tedy proběhla redukce, počet obdélníků se nám rapidně snížil. Nicméně, vlivem postupného zvětšování obdélníků se mohlo stát, že se nám překryly i některé velké obdélníky. Celou redukci tedy provedeme ještě jednou. Po dvounásobném procezení s největší pravděpodobností zůstane už jen tolik obdélníků, kolik je v masce oddělených bílých oblastí.

Nyní tedy pro každý fiktivní obdélník dle jeho souřadnic vykreslíme skutečný obdélník do aktuálně zpracovávaného snímku (viz obr. 11). Uživateli ponechávám možnost nastavit barvu a tloušťku obvodové hrany kreslených obdélníků.

Obrázek 11: Vyhledání bílých oblastí v masce a jejich ohraničení v aktuálním snímku.

3.6 Uživatelské rozhraní aplikace

Jak jsem již zmínil v úvodu této kapitoly, uživatelské rozhraní je na velice primitivní úrovni, jelikož byl kladen důraz především na samotnou podstatu detekce. Návod na spuštění aplikace je uveden v příloze 1.

3.6.1 Konfigurační soubor

K zadání vstupních parametrů a k řízení celé detekce jsem navrhnul jednoduchý konfigurační soubor. V kapitolách výše jsem občas zmínil jisté atributy detekce, které byli přizpůsobeny k tomu, aby mohl uživatel nastavováním jejich hodnot více či méně ovlivňovat chování detektoru v různých situacích.

V konfiguračním souboru lze nastavit:

- zdroj pro načtení video sekvence
- název souboru pro případný zdroj ze souboru
- práh detekce - čím větší hodnota, tím více se musí popředí lišit od pozadí
- kombinaci filtrů aplikovaných na masku - Medián, Eroze, Dilatace, žádný filtr
- metodu aktualizace referenčního snímku
- jemnost detekce - velikost nejmenšího objektu, který má být ještě detekován
- minimální vzdálenost mezi kreslenými obdélníky - jinak jsou sloučeny do jednoho
- barvu a tloušťku obvodové hrany kreslených obdélníků
- zda má být zobrazován referenční snímek a maska mezivýsledku
- rychlost zpracovávání video sekvence
- zda má být zobrazen posuvník pro regulaci rychlosti za běhu detekce

Podrobný popis syntaxe konfiguračního souboru je uveden v příloze 2.

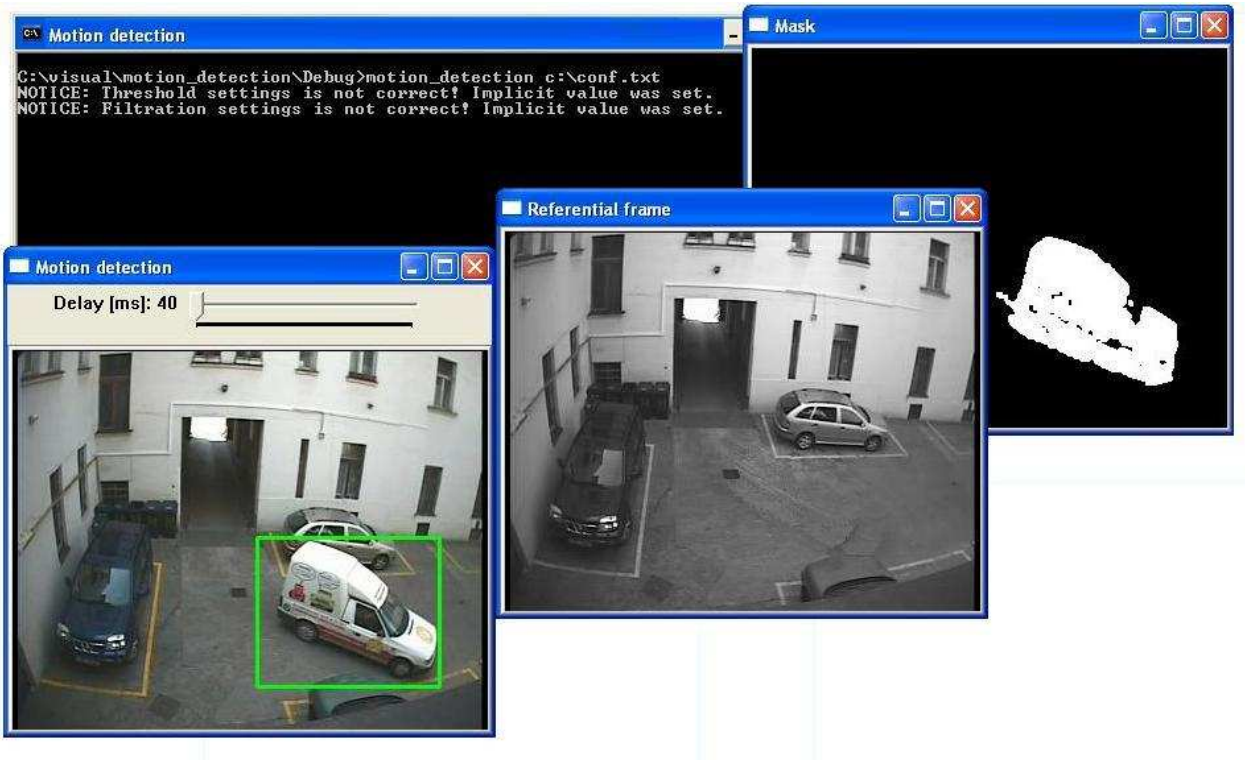
3.6.2 Výstup aplikace

Pro zobrazení výsledku jsem použil funkci knihovny OpenCV, která vytvoří jednoduché okno a do něho vykreslí zvolený obrázek. Výstupem mé aplikace je tedy přehrávání vstupní video sekvence, ve které jsou vyznačovány pohybující se objekty. Je možno požadovat zobrazení masky vzniklé po odečtení referenčního snímku a po aplikaci zvolených filtrů a také samotný referenční snímek v dalších oknech.

V průběhu detekce mohou nastat různé nežádoucí situace. Nepovede se třeba nalézt konfigurační soubor, nebo hodnoty v něm zapsané budou v chybně syntakticky zapsány. V těchto případech byl kladen důraz na to, aby těchto drobných uživatelských chyb detektor vždy nějak zahájil svou činnost. Pouze bude vypsáno upozornění na standardní chybový výstup a detektor použije implicitní hodnoty.

Může nastat samozřejmě i závažnější chyba, kdy detekci nebudeme moci spustit nebo dokončit. V těchto případech je aplikace ukončena a na textový terminál sdělena příčina tohoto počínání.

Na následujícím obrázku 12 je zaznamenán výstup aplikace v průběhu detekce pohybu.



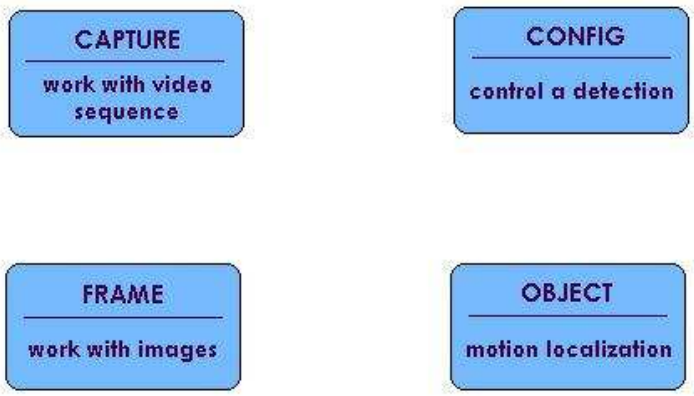
Obrázek 12: Celkový vzhled aplikace v průběhu detekce pohybu.

4 Implementace

Celá aplikace byla implementována v prostředí Microsoft Visual Studio 2005 Professional Edition v objektově orientovaném jazyce **C++**. Byl využit vestavěný kompilátor. Pro zpracování videa byla použita volně dostupná knihovna **OpenCV**. Pro spuštění aplikace je nutné mít knihovnu OpenCV nainstalovánu.

4.1 Objektový návrh

Ve snaze co nejlépe zpřehlednit kód byla aplikace navržena s využitím možností objektově orientovaného programování. Na obrázku 13 je znázorněn diagram navržených tříd.



Obrázek 13: Objektový návrh aplikace.

4.1.1 Třída CAPTURE

Třída CAPTURE reprezentuje zpracovávanou video sekvenci. Má přetížený konstruktor pro načtení video sekvence ze souboru a z připojené kamery. Obsahuje metodu pro získání dalšího snímku z video sekvence.

4.1.2 Třída FRAME

Třída FRAME reprezentuje samostatný snímek, popřípadě jakýkoliv obrázek. Obsahuje konstruktory pro vytvoření obrázku z existujícího snímku (načteného z video sekvence), pro vytvoření nového obrázku zadaných vlastností a pro načtení obrázku ze souboru.

Metody této třídy zastupují všechny operace, které jsou na snímek aplikovány. Zmíním pouze ty podstatnější z nich - aktualizace, kopie a zobrazení snímku; filtry grayscale, prahování, medián, eroze a dilatace; odečtení dvou snímků; průměrování dvou a více snímků; vykreslení obdélníku; ohraničení pohybujících se objektů.

4.1.3 Třída OBJECT

Třída OBJECT reprezentuje polygon, který ve snímku ohraničuje oblast, kde se odehrává pohyb. Polygon má tvar obdélníku a je vždy rovnoběžný s okraji snímku. Je určen 4 atributy: levý okraj (left), pravý okraj (right), horní okraj (top) a dolní okraj (bottom). Jejich hodnoty jsou souřadnice v rámci snímku, přičemž počátek souřadnic je v levém horním rohu. Pro vykreslení obdélníku potřebujeme znát jeho 2 protilehlé body, které získáme například touto kombinací: `[left, top]` a `[right, bottom]`.

Tato třída obsahuje jedinou metodu, která je volána z metody třídy FRAME pro ohraničení pohybujících se objektů. Metoda porovnává dva obdélníky, tedy dvě instance třídy OBJECT, a zjistí jejich vzájemný vztah. Princip je podrobněji popsán v kapitole *Lokalizace pohybu* v *Návrhu aplikace*.

4.1.4 Třída CONFIG

Třída CONFIG reprezentuje sadu konfiguračních hodnot, podle kterých se řídí celá detekce. Obsahuje metodu, která načte veškeré nastavení z konfiguračního souboru. Přetížený konstruktor naplní všechny atributy implicitními hodnotami pro případ, kdy některá hodnota nebude v konfiguračním souboru uvedena.

4.1.5 Rozdělení do modulů

Zdrojový kód aplikace jsem rozdělil dle objektového návrhu do 3 modulů. Implementace tříd s prototypy jejich metod se nachází v samostatném hlavičkovém modulu **classes.h**. Definice metod je pak oddělena v samostatném modulu **methods.cpp**. Samotný algoritmus detekce pohybu je implementován s využitím navržených tříd a jejich metod v modulu **motion_detection.cpp**.

Snažil jsem se rozdělit celou detekci do více či méně dílčích problémů. Nejrozsáhlejší je metoda pro nalezení všech bílých oblastí v masce a jejich ohraničení. Metoda využívá pro ukládání nalezených obdélníků (instancí třídy OBJECT) „nafukovací“ pole - typ `<vector>`.

4.2 OpenCV

Knihovna OpenCV byla vyvinuta speciálně ke komerčnímu využití počítačového vidění pro interakci člověk-stroj, robotiku, různé monitorovací a bezpečnostní systémy. Je open-source a nabízí otevřenou infrastrukturu zpracování obrazu.

Pro naše účely poskytuje široké spektrum funkcí pro práci s obrázkem a videem. Umožňuje načíst uloženou video sekvenci, připojit se na kameru, zpracovávat jednotlivé snímky video sekvence, uložit ji do souboru, nebo vytvořit zcela novou. Nabízí celou řadu operací s obrázky od načtení, uložení nebo vytvoření nového, přes vykreslení základních entit (úsečka, polygony, atd.), až k nejrůznějším operacím, které obrázek nějak transformují (převody mezi barevnými modely, morfologické operace, obrazové filtry a další transformace).

Rozsáhlé možnosti OpenCV mi umožnily se ve své práci zcela zaměřit na samotný algoritmus detekce. Nemusel jsem se zabývat implementací základních operací s obrazem.

Při práci s OpneCV jsem čerpal z [7].

5 Testování

Tato kapitola bude věnována několika testům. Pokusím se postavit svůj detektor do několika situací, jež budou reprezentovat různorodost prostředí, v kterých může být detekce pohybu vyžadována. Budu zde hodnotit úspěšnost správného odlišení popředí snímku od jeho pozadí, účinnost odolávání drobným změnám v nedokonale statickém pozadí snímku, schopnost přizpůsobovat se změnám v osvětlení scény a další aspekty detekce.

5.1 Detekce pohybu v uzavřené místnosti

Tato varianta představuje situaci, kdy kamera snímá prostor v uzavřené místnosti. Nejpodstatnějším faktorem tohoto prostředí je jednoznačně téměř stoprocentní absence veškerých přírodních vlivů, jež mohou detekci jakkoliv stěžovat. V uzavřené místnosti panují většinou stále stejné podmínky (nefouká vítr, nesněží, nelétají ptáci, atd.). Celkové osvětlení scény se také nijak výrazně rychle nemění. Teď nebudeme uvažovat případy, kdy se například v místnosti s okny najednou téměř setmí z důvodu náhlého příchodu bouřky, nebo když někdo v místnosti zhasne či rozsvítí světlo. Ve většině případů se tedy můžeme spolehnout na skutečnost, že je-li detekován pohyb, nejedná se zpravidla o planý poplach.

Při volbě metody aktualizace referenčního snímku tedy není nutno docílit schopnosti čelit rychlým a skokovým změnám v pozadí. Můžeme využít **chytré průměrování** referenčního snímku, které nedovolí, aby pohybující se objekt splynul s pozadím. Bude tedy detekován i extrémně pomalý pohyb. Naopak drobné diference ve světlosti v různých částech scény budou chytrým průměrováním kompenzovány.

V tomto případě mnoho záleží na zvoleném **prahu detekce**. Ten je totiž využíván i k vynechání pohybu z průměrování. Čím větší práh zvolíme, tím větším změnám v pozadí bude schopen detektor odolávat.

V uzavřené místnosti se spolu s chytrým průměrováním chová detektor velice úspěšně. Je schopen detekovat extrémně pomalý pohyb, ale poradí si i s rychlým pohybem. Je ale bohužel náchylný na sebemenší skokové změny v pozadí snímku.

Spuštěním dávky **indoor1.bat**, **indoor2.bat** (pomalý pohyb), nebo **indoor3.bat** (rychlý pohyb) na přiloženém mediu ve složce „demo“ bude spuštěna detekce s nastavenými parametry pro detekci v těchto podmínkách (popř. viz obrázky 14 a 15).

Obrázek 14: Detekce pohybu v uzavřené místnosti s chytrým průměrováním - pomalý pohyb.

Obrázek 15: Detekce pohybu v uzavřené místnosti s chytrým průměrováním - rychlý pohyb.

5.2 Detekce pohybu venku

Když bude kamera umístěna venku, budou nám detekci ztěžovat přírodní vlivy. Těch může být nepřeberné množství. Už jsem je ve své práci několikrát jmenoval. Mezi nejpodstatnější patří změna osvětlení scény střídáním noci a dne nebo náhlým zatažením oblohy a především vítr, který nám tím, že hýbe se stromy a dalšími předměty, ztíží detekci asi nejvíce. Opět samozřejmě záleží na konkrétní situaci. Při snímání prostoru parkoviště nám jistě vítr nebude vadit tolik, jako při snímání městského parku s desítkami stromů.

Poněvadž tedy musíme počítat s náhlými a často relativně velkými skokovými změnami, nemůžeme zde použít chytré průměrování. Zde se naopak velice osvědčí klasická varianta **průměrování dvou** a **více** po sobě jdoucích **snímků**. Každá náhlá změna bude tímto vždy v krátké době kompenzována. Tato doba bude záviset na zvolené frekvenci průměrování a zvoleném počtu průměrovaných snímků. Když bude třeba čelit méně statickému pozadí snímku, snížíme počet průměrovaných snímků a zmenšíme periodu mezi jednotlivými průměrováními. V případě více statického pozadí tyto hodnoty můžeme zvětšit.

Při detekci ve venkovním prostředí se velice osvědčilo obyčejné průměrování 2 po sobě jdoucích snímků v krátkých periodách. Abychom zcela eliminovali například pohyb stromů, je výhodné nastavit neustálé průměrování, tedy s periodou 1. Toto řešení ovšem způsobí, že objekty, které jsou pomalé, nebo se zastaví, nebudou detekovány, protože budou ihned promítnuty do referenčního snímku a splynou tak s pozadím. Musíme proto pro každou situaci zvolit vhodný poměr mezi eliminací nestatičnosti pozadí a úspěšností detekce.

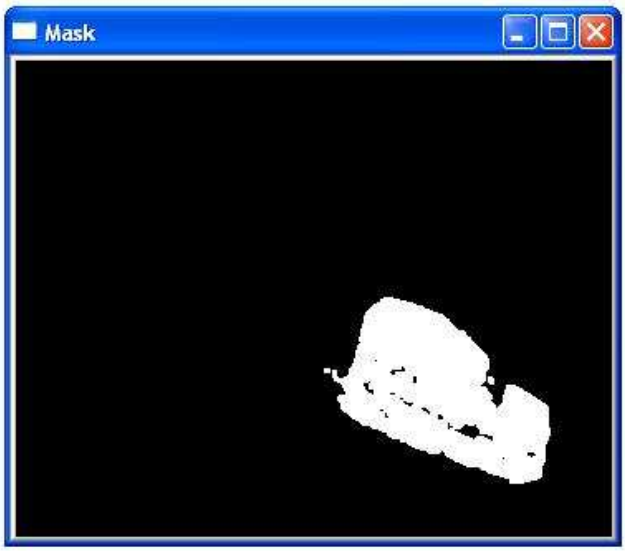
Spuštěním dávky **outdoor.bat** na přiloženém mediu ve složce „demo“ se rozběhne detekce venku s pohybujícími se stromy v pozadí. Díky neustálému průměrování stromy nepůsobí problémy, nicméně pomalejší objekty nejsou ideálně nalezeny. Myslím ale, že zde vidíme relativně dobrý kompromis (obr. 16).

Obrázek 16: Detekce pohybu v nestatickém prostředí s průměrováním 2 snímků s periodou 1.

5.3 Strategie aplikace filtrů

Aplikace umožňuje zvolit jednoduché, dvojité nebo trojité filtrování. Lze přitom použít jakoukoliv kombinaci filtrů Medián, Eroze a Dilatace.

Po odzkoušení všech kombinací se ve většině případů nejlépe osvědčilo filtrování MD - tedy Medián a Dilatace. Medián zařídí odstranění šumové informace a Dilatace zrekonstruuje a vyhladí bílé oblasti v masce (viz obr. 17), takže jsou pak lépe lokalizovány.



Obrázek 17: Maska po aplikaci filtrů Medián a Dilatace (pohybující se objekt - auto).

Pokud bychom ale například očekávali detekci několika desítek objektů ve snímku, určitě by se vyplatilo využít filtru Eroze. Buď místo Mediánu nebo v kombinaci s ním. Eroze, mimo opět dobré schopnosti odstranit šum, bílé oblasti v masce nahuštěné na sobě od sebe lépe oddělí a jejich hranice posune směrem dovnitř. Opět může být ještě aplikována Dilatace. Vždy však opět záleží na konkrétní situaci a uživateli, popř. osobě, která detektor instaluje a uvádí do provozu, nezbývá nic jiného, než strávit nějaký čas nad nalezením nejvhodnější kombinace.

5.4 Podmínky testování

Testování probíhalo na operačním systému Windows XP Professional SP2 na stroji značky Asus s procesorem Intel Celeron 1,5 GHz a 760 MB paměti. Testovací video sekvence obstarávala webová kamera Labtec v rozlišení 320x240 pixelů.

Závěr

Cílem této práce bylo rozebrat problematiku detekce pohybu. Po nastudování několika principů jsem zvolil jako výchozí metodu odečítání pozadí a algoritmus porovnávání jednotlivých pixelů mezi snímky. Na tomto principu byl postaven detektor pohybu, použitelný pouze pro nehybnou kameru.

Výsledná aplikace může dobře sloužit jako ukázka detekce pohybu v různých prostředích. Široké možnosti nastavení parametrů detekce v konfiguračním souboru umožňují dosáhnout vždy poměrně dobrých výsledků.

Detektor byl úspěšně testován v několika diametrálně odlišných prostředích. Několik navržených metod aktualizace referenčního snímku, zvláště chytré průměrování, zajišťuje dobrou stabilitu detektoru a schopnost zotavit se z drobných změn v pozadí snímku.

Navíc jako vítané rozšíření byla implementována lokalizace pohybu. Pohybující se objekty jsou ve snímku opsány obdélníkem a tím sledovány. Byl proto navržen speciální algoritmus pro hledání a ohraničování bílých oblastí v masce.

Na této práci by bylo rozhodně možno pokračovat dále, a to bez obav z nedostatku nápadů a motivace. Celý detekční algoritmus byl navržen s důrazem na funkčnost. Není příliš výkonově šetrný. Optimalizace tohoto detektoru by mohla s přehledem být novým tématem pro celou další práci. Algoritmy odečítání pozadí, průměrování, hledání bílých oblastí by jistě mohli projít hlubokou korekturou, případně přeměnou do jiných jejich varant či variací.

Detekce pohybu a z globálního pohledu samotné počítačové vidění je velice rozsáhlá oblast vědy. Získávání informací z rastrové reprezentace obrazu je pro mě velice zajímavým tématem a pro rozvoj informačních technik snažících se co nejlépe komunikovat s okolním světem nesmírně důležitým bodem.

Literatura

[1] Hlaváč, V., Šonka, M. *Počítačové vidění*, GRADA 1993.

[2] Jelínek, T. *Detekce pohybujících se objektů ve video sekvenci*. Diplomová práce (2007).

[3] Heikkilä, M., Pietikäinen, M. *A texture-based method for modeling the background and detecting moving objects*. http://www.ee.oulu.fi/mvg/publications/show_pdf.php?ID=662 (2006).

[4] Kirillov, A. *Motion Detection Algorithms*. http://www.codeproject.com/cs/media/Motion_Detection.asp (2006).

[5] Kamath, Ch. *Detection of tracking of moving objects in video*. https://computation.llnl.gov/casc/sapphire/video/video.html (2005).

[6] Kamath, Ch. *Backround subtraction for detection of moving objects*. https://computation.llnl.gov/casc/sapphire/background/background.html (2005).

[7] *OpenCV library Wiki*. http://opencvlibrary.sourceforge.net/

[8] *Wikipedie, otevřená encyklopedie*. http://cs.wikipedia.org

[9] *Wikipedia, the free encyklopedia*. http://en.wikipedia.org

Seznam příloh

Příloha 1: Manuál k ovládání aplikace.

Příloha 2: Syntaxe konfiguračního souboru.

Příloha 3: DVD s aplikací, demonstračními testy a knihovnou OpenCV.

Příloha 1: Manuál k ovládání aplikace

Spuštění aplikace

Předpokládejme, že je aplikace zkompilována a sestavena do souboru **`motion_detection.exe`**.

Když nyní spustíme `motion_detection.exe`, aplikace se pokusí najít konfigurační soubor s implicitním názvem `config.txt` v aktuálním adresáři. Pokud konfigurační soubor nenalezne, spustí detekci s přednastavenými hodnotami.

Druhou možností je zvolit jiný konfigurační soubor, který si uživatel připraví. Ten může mít jakýkoliv název a na jeho umístění nezáleží. Při spouštění aplikace uvedeme název konfiguračního souboru s kompletní cestou k němu jako další parametr v příkazové řádce. V tomto případě nesmí být zadán žádný další parametr. Příklad spuštění:

```
> motion_detection dir\my_config_file.txt
```

Konfigurační soubor musí být vždy textový. Vždy v případě nenalezení nebo neúspěchu otevření souboru v textovém režimu bude hledán implicitní konfigurační soubor.

Na přiloženém mediu v adresáři „demo“ je mimo jiné předpřipraveno několik spustitelných dávek, které demonstrují různé varianty detekce. Každá z nich používá svůj konfigurační soubor.

Ukončení aplikace

Běžící aplikace ukončí svoji činnost sama po zpracování celé video sekvence. Pokud ale načítáme video sekvenci přímo z připojené kamery, nebo když chceme aplikaci ukončit dříve, postačí uzavřít textový terminál nebo všechna okna aplikace a ta se pak sama ukončí.

Příloha 2: Syntaxe konfiguračního souboru

Konfigurační soubor musí být **textový** a mít následující **syntaxi:**

```
input:          [FILE/STREAM]
video_file:     (adresar\soubor)
threshold:      [0-255]
filtration:     [ANY/M/E/D/ME/MD/EM/ED/DM/DE/MED/MDE/EMD/EDM/DME/DEM]
avgset:         [ANY/TWO/MORE/MORE2] [1-(max_int)] [2-(max_int)]
nicety:         [0-(max_int)] [0-(max_int)]
max_distance:   [1-(max_int)]
brush:          [R/G/B] [1-(max_int)]
windows:        [ANY/MASK/REF/ALL]
speed:          [1-(max_int)]
speed_control:  [YES/NO]
```

Všechny položky jsou case-sensitive, takže je třeba striktně dodržovat velká a malá písmena. Každý atribut se svou hodnotou musí být na novém řádku tak jak je uvedeno výše. Pořadí atributů (řádků) je možno libovolně zaměnit. Pokud bude atribut uveden vícekrát, bude platit poslední jeho výskyt. Atributy nemusí být uvedeny všechny. Při absenci atributu bude nastavena implicitní hodnota a vypsáno upozornění na terminál. Tak se stane i v případě chybně zapsané hodnoty atributu.

Atributy:

input	- typ zdroje zpracovávané video sekvence
video_file	- název video souboru
threshold	- práh detekce
filtration	- kombinace filtrů
avgset	- metoda průměrování
nicety	- jemnost detekce
max_distance	- min. vzdálenost mezi obdélníky
brush	- barva a tloušťka hrany obdélníku
windows	- co vše má být zobrazeno
speed	- zpoždění mezi snímky
speed_control	- zobrazení regulátoru zpoždění

Sémantika:

`FILE/STREAM`	- video soubor / signál z kamery
`ANY/M/E/D`	- bez filtrace / Medián / Eroze / Dilatace
`ANY/TWO/MORE/MORE2`	- bez prům. / prům. dvou snímků / více snímků / chytré prům.
`R/G/B`	- červená / zelená / modrá
`ANY/MASK/REF/ALL`	- pouze akt. snímek / zobrazení masky / ref. snímku / všeho
`YES/NO`	- zobrazit regulátor / nezobrazit regulátor

Příklad konfiguračního souboru:

`input:`	`FILE`	- video sekvence ze souboru
`video_file:`	`c:\\test8.avi`	- video soubor
`threshold:`	`25`	- když se pixely liší o 25 => detekován pohyb
`filtration:`	`MED`	- filtrace - Medián + Eroze + Dilatace
`avgset:`	`MORE2 30 10`	- průměruje se 10 snímků v periodách 30ti snímků
`nicety:`	`2 3`	- objekty užší než 2 a nižší než 3 pixely nebereme
`max_distance:`	`15`	- mezery mezi obdélníky budou min. 15 pixelů
`brush:`	`B 3`	- modrý obdélník, tloušťka hrany = 3 pixely
`windows:`	`ANY`	- zobrazuj jen aktuální snímek
`speed:`	`40`	- zpoždění mezi snímky cca 40 ms
`speed_control:`	`YES`	- zobraz trackbar pro regulaci zpoždění