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

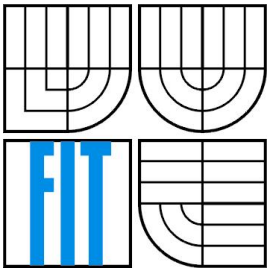
DETEKCE POHYBUJÍCÍCH SE OBJEKTŮ VE VIDEO SEKVENCI

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. TOMÁŠ JELÍNEK
AUTHOR

BRNO 2007

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 POHYBUJÍCÍCH SE OBJEKTŮ VE VIDEO SEKVENCI

MOTION DETECTION IN VIDEO SEQUENCE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ JELÍNEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MICHAL ŠPANĚL

BRNO 2007

Zadání diplomové práce

Řešitel: **Tomáš Jelínek, Bc.**

Obor: Počítačová grafika a multimédia

Téma: **Detekce pohybujících se objektů ve video sekvenci**

Kategorie: Zpracování obrazu

Pokyny:

1. Prostudujte základy zpracování obrazu. Zaměřte se na detekci pohybu v obrazové sekvenci.
2. Zorientujte se v metodách analýzy pohybu v obraze a segmentace pohybujících se objektů.
3. Vyberte vhodnou metodu a navrhněte jednoduchý detektor pohybujících se objektů ve video sekvenci.
4. Experimentujte s vaší implementací.
5. Případně navrhněte vlastní modifikaci metod.
6. Diskutujte dosažené výsledky a možnosti budoucího vývoje.
7. Vytvořte jednoduchý plakát, kterým budete prezentovat dosažené výsledky.

Literatura:

Dle pokynů vedoucího.

Při obhajobě semestrální části diplomového projektu je požadováno:

Splnění prvních tří bodů zadání.

Vedoucí: **Španěl Michal, Ing.**, UPGM FIT VUT

Datum zadání: 28.února 2006

Datum odevzdání: 22. května 2007

Vedoucí ústavu: **doc. Dr. Ing. Pavel Zemčík**

Licenční smlouva

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

Abstrakt

Práce se věnuje metodám pro detekci pohybu ve video sekvenci. Najdete v ní souhrn několika možných přístupů k řešení této problematiky spolu s teoretickým i praktickým náhledem. Detailně jsou popsány principy detekce na základě rozdílů mezi snímky a to z pohledu jejich histogramů nebo světelnosti pixelů. Velká pozornost je věnována popisu obrazu pomocí Local Binary Patterns, na kterém lze mimo jiné postavit rychlý a přesný detektor pohybu. Součástí práce je aplikace pro srovnání úspěšnosti metod v typově různých prostředích. Z výsledků testování jsou odvozeny klady a zápory jednotlivých přístupů, které mohou být čtenáři podkladem pro návrh vlastního detektoru.

Klíčová slova

Detekce pohybu, obrazové filtry, Local Binary Patterns, video sekvence, detektor, Visual Studio .NET, C#, zpracování obrazu, zpracování videa.

Abstract

This thesis is dedicated to techniques for motion detection in video sequences. There is summary of several possible approaches to solve this matter in theoretical and practical point of view. Principles of motion detection based on difference between frames are discussed in details. Great attention is made for image description by new method Local Binary Patterns which allows creating of incredibly fast and accurate motion detector. Vital part of this thesis is application for monitoring and testing motion algorithms on video sequences in different environments. Results of measuring enables to determine best approaches for concrete place and reader might find it useful in case of designing his own motion detectors.

Keywords

Motion detection, image filters, Local Binary Patterns, video sequence, detector, Visual Studio .NET, C#, picture processing, movie processing.

Citace

Tomáš Jelínek: Detekce pohybujících se objektů ve video sekvenci. Brno, 2007, diplomová práce, FIT VUT v Brně.

Detekce pohybujících se objektů ve video sekvenci

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Michala Španěla. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Tomáš Jelínek
22.5.2007

Poděkování

Rád bych poděkoval svému vedoucímu diplomové práce Ing. M. Španělovi za cenné rady a kamarádce MUDr. Janě Jankovičové za výdrž a důslednost při korekturách.
Dále bych chtěl poděkovat firmě Dundas za bezplatné poskytnutí jejich komponenty pro tvorbu grafů. V neposlední řadě pak děkuji všem, kteří mi tuto práci pomohli cennými radami a podporou dostat do konečné podoby.

Obsah

1 Úvod

Hlavním podnětem pro vytvoření této diplomové práce mi byl zajímavý projekt na rozpoznání pohybu ve video sekvenci. V něm jsem poprvé viděl v praxi, jak obtížné je pro počítač získat informaci, nad kterou v našem lidském vnímání okolního světa ani nepřemýšlíme a přijde nám naprosto samozřejmá.

Ve své práci se budu věnovat právě algoritmům, které z rastrové reprezentace obrazu dokáží extrahovat a rozlišit proměnlivé popředí od statického pozadí, na čemž se dá postavit mimo jiné i detektor pohybu. To, že popředí do kterého náleží pohybující se objekty není vždy úplně proměnlivé a pozadí zase vždy úplně statické, je jedním z mnoha problémů, které ovlivňují úspěšnost rozpoznávání. K jejich řešení vzniklo mnoho odlišných přístupů a cílem mé práce je čtenáře nejen s většinou z nich seznámit, ale zejména srovnáním a typovou kategorizací podle jejich pozitiv a negativ prezentovat, které z nich by v konkrétním prostředí fungovaly nejlépe.

Typové kategorizace a srovnání detekčních algoritmů jsou prezentovány na základě výsledků získaných z testovací aplikace. Tato aplikace vznikla jako součást diplomové práce a lze si v ní nejen ověřit výsledky měření, ale slouží i k praktické demonstraci, a tím lepšímu pochopení informací z teoretických kapitol.

1.1 Stručný popis kapitol

Po úvodu, kde prezentuji charakter a cíle celé práce, je teoretická kapitola „Základy zpracování obrazu“. Tato kapitola slouží k bližšímu popisu jednotlivých technik a obecných znalostí, které jsou potřeba k pochopení výkladu dalších částí. Čtenáři, laikovi v oblasti zpracování obrazu, se zde snažím nenáročnou formou vysvětlit konkrétní, dále používané principy a obecnou teorii, zatímco zkušený čtenář ji může přeskočit a vrátit se k ní až při nejasnostech v průběhu dalšího výkladu.

Třetí kapitola je nosnou částí a pojednává o detekčních algoritmech. Je zde postupně probírána každá metoda zvlášť a to vždy nejdříve z obecného pohledu a poté hlubšího na možnosti využití, způsoby alternativních návrhů k jejich vylepšení a nakonec je řečeno jakým způsobem lze na této metodě implementovat konkrétní detektor pohybu. Hlavní pozornost je věnována metodám, které jsou implementovány v testovací aplikaci a lze si je tam prakticky vyzkoušet.

Diskutovány jsou detektory na principu:

- rozdílné světelnosti pixelů mezi snímky
- porovnávání histogramů jasové složky pixelů
- popisu obrazu klasifikátorem Local Binary Patterns
- porovnávání snímku po aplikaci detektoru hran
- zpracování metodou optického toku

Čtvrtá kapitola je věnována testovací aplikaci a to z pohledu jejího návrhu, principů fungování, stručného popisu ovládání (to je detailněji popsáno v příloze) a nakonec náhledu na její implementaci.

Poslední kapitola před závěrem (pátá) se věnuje testování různých typů prostředí ve video sekvenci a prezentaci výsledků spolu s komentáři. Po úvodu je každá podkapitola věnována jednomu prostředí a tabulkou či grafem poskytuji výsledky jednotlivých měření. Z těchto výsledků pak vyplývá, která metoda se pro daný typ prostředí hodí nejlépe a naopak.

V závěru shrnuji poznatky získané v průběhu celé práce, její celkový přínos a v neposlední řadě diskutuji návrhy na další možná pokračování.

1.2 Návaznost na semestrální projekt

Pro navazující diplomovou práci bylo klíčové vyhledat a nastudovat algoritmy, které lze využít pro detekci, sledování a vyhodnocování pohybu v obraze. Tyto algoritmy spadají v počítačové grafice do oblasti zpracování obrazu a k jejich pochopení bylo nutné tuto oblast, ještě před začátkem práce na testovací aplikaci, důkladně prostudovat. Odvětví počítačové grafiky, které se věnuje extrakci informací z obrazu popsaného barevnými body, se v posledních letech výrazně rozvíjí, přesto mnohé přístupy nejsou zatím dostatečně popsány a většinou jsou publikovány jen formou obecných anotací projektů na webu výzkumných pracovišť nebo články ve vědeckých časopisech. Z toho titulu bylo hlavní náplní semestrálního projektu „hromadění“ literatury k dostupným metodám detektorů pohybu, čímž se položily základy pro tuto diplomovou prácí.

1.3 Stanovené cíle práce

Cílem v teoretické části je podat co nejvíce informací o detekčních algoritmech a vysvětlit jejich funkci a možné optimalizace a vylepšení. To je spojeno i s náhledem na použité filtry a další techniky

zpracování obrazu. V praktické části se budu věnovat zejména činnosti a práci v testovací aplikaci. Měla by poskytovat intuitivní rozhranní pro porovnávání úspěšnosti odlišných metod detekce pohybu na rozdílných prostředích ve vstupní video sekvenci. Při tvorbě aplikace jsem si kladl za vedlejší, ale neméně důležitý cíl program strukturovat tak, aby v případě pokračování na tomto projektu někým dalším bylo snadné přidávat další detektory a bez nějakých výrazných zásahů do logiky aplikace s nimi provést totožná měření. Výsledkem těchto měření bude vyhodnocení účinnosti a výkonnosti metody pro určitý typ prostředí a sestavení textových i grafických statistických údajů již v samotné aplikaci. Na základě těchto měření se ověří informace prezentované v teoretické části a zejména si podle nich uživatel může analyzovat vhodnost metody pro svůj vlastní detektor.

2 Základy zpracování obrazu

2.1 Obraz a jeho reprezentace

Pod pojmem obraz si různí lidé mohou představit různou věc a stejně je to i s definicí. Jednoznačná definice neexistuje a vždy bude záležet na oblasti, ve které se s tímto pojmem pohybujeme. V našem případě budeme pracovat s obrazem tak, jak jej definuje **[Žára04]**, tedy jako spojitou funkci dvou proměnných, které se obecně říká obrazová funkce ve tvaru:

$$z = f(x, y) \qquad (1.)$$

Takto definovaný obraz má omezené rozměry, a tedy definiční obor obrazové funkce lze zapsat jako kartézský součin dvou spojitých intervalů z oboru reálných čísel, které vymezují rozsah obrazu:

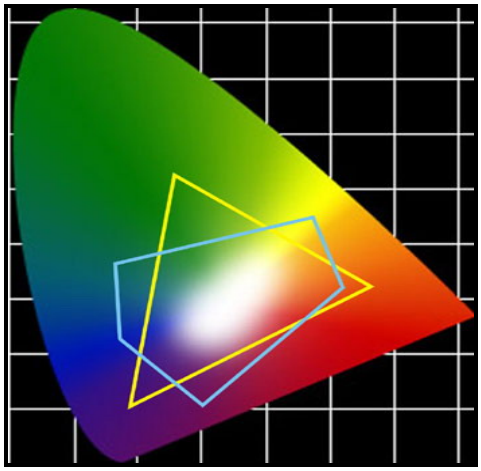
$$H(f) : \langle x_{\min}, x_{\max} \rangle \times \langle y_{\min}, y_{\max} \rangle \qquad (2.)$$

Reálná čísla **x**, **y** jsou souřadnice bodu ve dvourozměrném prostoru, kde funkce nabývá hodnoty **z** v oboru hodnot **H(f)**. Hodnota obrazové funkce může být jediné číslo nebo více hodnot v závislosti na použitém barevném modelu. Ve většině mnou použitých metod se pracuje s tradičním modelem **RGB** (výsledná barva bodu vznikne složením tří barevných složek červená, zelená a modrá), ale například pro algoritmy, které pro zvýšení své efektivity počítají nad nebarevnou informací, je výhodné použít jeden z dále zmíněných modelů **YUV** nebo **HSV**, kde jako hodnota obrazové funkce postačí pouze jediné číslo.

Obraz má nejen omezený definiční obor, ale navíc se s ním pracuje v nějaké diskrétní podobě neboli v rastru. Rastr je složen z obrazových elementů zvaných v počítačové grafice pixely (z anglického picture element).

Proces, při kterém se převádí obrazová funkce **f(x,y)** na diskrétní funkci, a to jak v definičním oboru, tak v oboru hodnot, se nazývá digitalizace. Jednoduše řečeno, reálný obraz, který má teoreticky nekonečný rozsah obrazových hodnot, převádíme na konečné množství pixelů a barev. Zařízení, kterým byl snímek pořízen, již samo většinou redukuje barevnou informaci do nějakého počítačem dobře srozumitelného oboru hodnot. Zpravidla se volí redukce na 16 milionů barev, která dokáže pokrýt prakticky celý rozpoznávací prostor lidského zraku nebo 256 stupňů šedi pro stejně kvalitní, ale nebarevnou informaci.

Na **(Obrázek 1)** je znázorněn rozsah vnímání barev lidským okem (největší část pokrývá zelená barva, na kterou je oko nejcitlivější), proti rozsahům, na které se většinou při digitalizaci obraz převádí (žlutě barevný model RGB a modře CMYK).



(Obrázek 1) Rozsah vnímání barev lidským okem proti rozsahu RGB(žlutě) a CMYK(modře)

2.2 Teorie barevných modelů

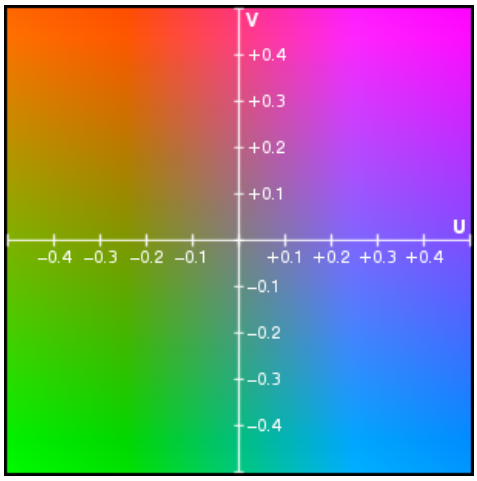
Algoritmy pro detekci obrazu pracují nad velkým objemem obrazových dat. Z toho důvodu jakákoli úspora, byť by se z pohledu jednoho snímku mohla zdát zcela zanedbatelná, na sekvenci tisíců snímků může uspořit velké množství času a systémových prostředků. Algoritmy pro detekci pohybu pro svou funkci většinou nepotřebují tak velký rozsah obrazové informace, kterou nasnímáme i úplně obyčejným zařízením. Často se bez problému spokojí s obrazem v nižším rozlišení nebo i bez barevné informace v pouhých odstínech šedi. Při praktickém využití jsou pořizovací zařízení většinou speciálně upravena tak, aby hardwarově upravila vstupní obraz do co nejvhodnějšího barevného modelu pro detekční algoritmus. V teoretických kapitolách bude uvedeno u konkrétních metod, jak lze redukovat časovou náročnost, právě použitím jiného barevného modelu než tradičního **RGB**.

Je zřejmé, že režie spojená s převodem **RGB** na jiný barevný model nebude zanedbatelná, ale vzhledem k možnosti upravit již vstupní snímací zařízení tak, aby ukládalo obraz v jiném barevném modelu, se tato nevýhoda dá odstranit.

2.2.1 Barevný model YUV

Model **YUV** definuje barevný prostor jako soubor jedné jasové složky **Y** a dvou barevných **U, V**. V praxi je nejčastěji používán ve vizuálních systémech standardů PAL a NTSC, proto získání vstupní sekvence pro detektor přímo v tomto modelu může poskytnout již samotné snímací zařízení. Lidskému vnímání je tento model blízký, ale ne tolik jako následující HSV. Hlavní výhodou, proč se tímto modelem má smysl blíže zabývat je, že prakticky veškerá nebarevná informace se dá uchovat jen v kanálu **Y** a v detektoru pracovat s odstíny šedi bez ztráty barevné informace. Vzorec **(3.)** reprezentuje matici pro převod mezi tímto modelem a **RGB**. Pro ilustraci uvádím **(Obrázek 2),** kde je vidět barevný prostor reprezentovaný složkami **U**, **V** při **Y = 0,5** (poloviční jas, kde lze nejlépe pozorovat sytost barev).

(3.)
$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.515 & -0.100 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$



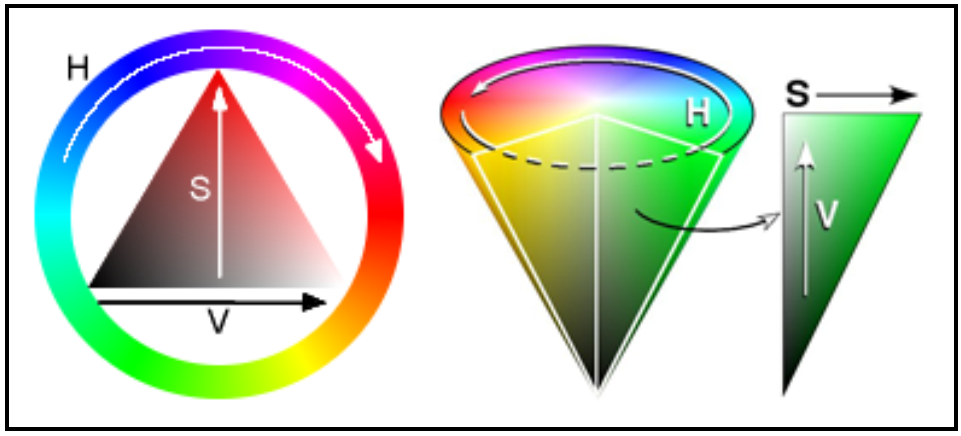
(Obrázek 2) Barevný prostor modelu YUV pro Y = 0,5

2.2.2 Barevný model HSV

Zkratka **HSV** je vytvořena z prvních písmen slov Hue (odstín), Saturation (sytost), Value (jas), což jsou tři složky, ze kterých se skládá barevná informace u tohoto modelu. Hue neboli odstín je barva přesně definovaná na spektrálním kotouči **(Obrázek 3)** hodnotou 0–360, nebo normalizovaně jako 0–100%. Sytost barvy (saturation) je definována v rozmezí 0–100%. Ve své podstatě si barvu v určité hodnotě sytosti můžeme představit, jako bychom do plné hodnoty této barvy přimíchávali šedou barvu a čím víc bychom jí přidali, tím víc vybledlá by se nám jevila. Jasová složka je také v rozmezí 0–100%, ale naopak čím vyšší hodnota, tím více jasná se nám barva bude jevit. Stejně jako jsem

uváděl u modelu **YUV**, je právě jasová složka ta, která v sobě nese spektrum podobné odstínům šedi a proto se dá v detekčních algoritmech s výhodou použít, aniž by se ztratila informace o barvě.

Obecně je model **HSV** v grafice používán proto, že jeho reprezentace je oproti modelům **RGB** či **YUV** mnohem blíže lidskému vnímání barev (lépe se nám popisuje barva způsobem, kdy řekneme název obecné barvy a poté dodáme, zda je světlejší či tmavší). Na barevném kole **(Obrázek 3)** vlevo nebo na spektrálním kuželu vpravo je vidět, jakým způsobem se dá popsat konkrétní barva.



(Obrázek 3) Barevné kolo HSV (vlevo) a barevný kuželový prostor HSV (vpravo)

2.2.3 Převod na 256 stupňů šedi (Grayscale)

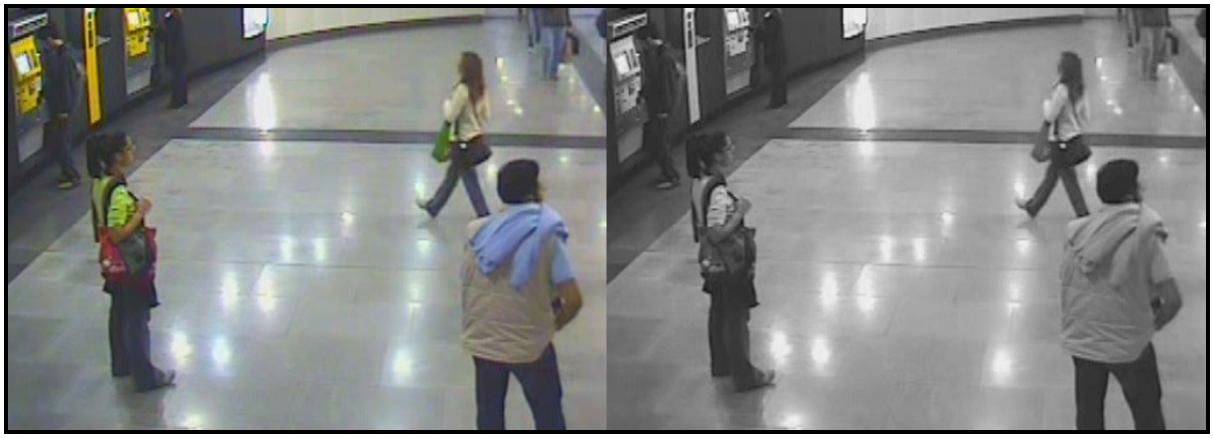
Předchozí modely převedly obraz z tří-bytové reprezentace třech základních barev na opět tříbytovou reprezentaci, čímž se vždy zachovala informace o barvě. Pokud tuto barevnou informaci nepotřebujeme, lze převést obraz na jedno-bytovou reprezentaci a tím ušetřit místo v paměti. Jedno-bytový prostor obsáhne pouhých 256 různých odstínů, ale pokud se chytře zvolí paleta převodu barev na co nejvíce odpovídající šedý odstín, tak zachováme i bez barvy všechny atributy původního obrazu. Jak je vidět na záběru průmyslové kamery **(Obrázek 4)** z projektu **Caretaker**, tak i po redukci barev jsou postavy jasně odlišeny od pozadí a půjdou snadno extrahovat.

Šedá barva se vypočítá na základě hodnot jednotlivých barevných složek **RGB** a rovná se součtu 30% hodnoty červené, 59% zelené a 11% modré barvy podle vzorce **(4.)**.

$$JAS = 0.3R + 0.59G + 0.11B \quad (4.)$$

Tyto procentuální poměry vycházejí z odlišné intenzity vnímání barev lidským zrakem, tak jak bylo dřív ukázáno na barevném prostoru **(Obrázek 1)**, s největším důrazem na zelenou a nejmenším na modrou barvu. Pokud je výsledná šedá barva stanovena právě tímto poměrem, tak se nám obraz jeví

stejný jako původní barevný. Stejným způsobem se získala i hodnota jasové složky **Y** u modelu **YUV**.



(Obrázek 4) Redukce barevné informace ze tří bytů na jeden byte do 256 odstínů šedi

2.3 Filtrování obrazu

Filtrování obrazu je důležité nejen pro snižování výpočetních nároků, ale navíc zlepšuje funkci detektorů tím, že jim obraz nějakým způsobem vhodně předpřipraví. Nejčastěji jde o redukci šumu, který je při detekci jedním z největších nepřátel a jeho bližšímu teoretickému popisu se věnuje kapitola **2.3.1**.

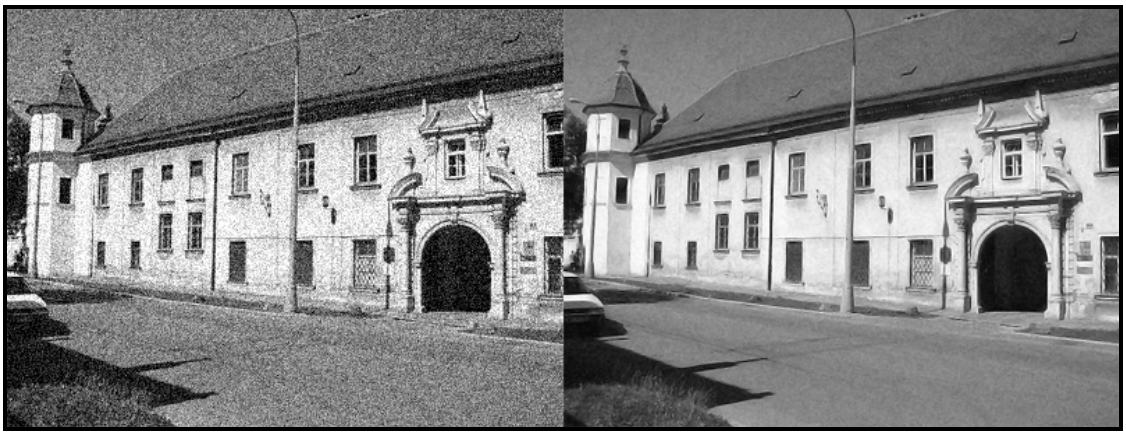
V praxi platí, jestliže bude obraz výrazně zašuměný, tak bude nutné detektoru zvednout hranici, nad kterou teprve začíná o pohybu rozhodovat. Tím se ale zvyšuje pravděpodobnost, že nám nějaký objekt splyne s pozadím a nedojde k jeho správnému vyhodnocení. Vhodně použitým filtrem můžeme šum redukovat a nebo posílit hranici mezi oblastmi popředí a pozadí.

V podstatě potřebujeme poslat detektoru obraz s co nejmenším množstvím pixelů, ale takový, kde informace pozadí a popředí bude od sebe v co největší možné míře odlišena. Když se nám to povede, tak detektor bude vyhodnocovat snímky na základě menšího počtu pixelů mnohem rychleji a pokud informace zůstala stále dostatečně intenzivní, tak redukce množství pixelů nepovede k chybným vyhodnocením.

2.3.1 Definice šumu a jeho odstraňování

Šum se v mnohých publikacích definuje jako nová informace, která byla k původní přidána pořizovacím zařízením či během transportu. Je třeba si uvědomit, že šum je svázán s původní informací, a proto k jeho dokonalému odstranění bychom museli mít k dispozici původní funkci. Náš obraz je však reprezentován pixely, a proto si se šumem musíme poradit jinak a hlavně se smířit s tím, že se jej nikdy nedokážeme zbavit úplně.

Na **(Obrázek 5)** je vidět jak i silný náhodný (zde Gaussovský) šum lze velmi dobře použitím různých filtrů redukovat, což je velice důležité pro zachování co nejméně se měnícího pozadí. Metodám odstraňování šumu by se dala lehce věnovat samostatná diplomová práce, proto se budu ve zkratce věnovat jen těm, které jsem sám někde použil a nebo navrhnul.



(Obrázek 5) Ukázka aplikace filtrů pro redukci silného Gaussovského šumu

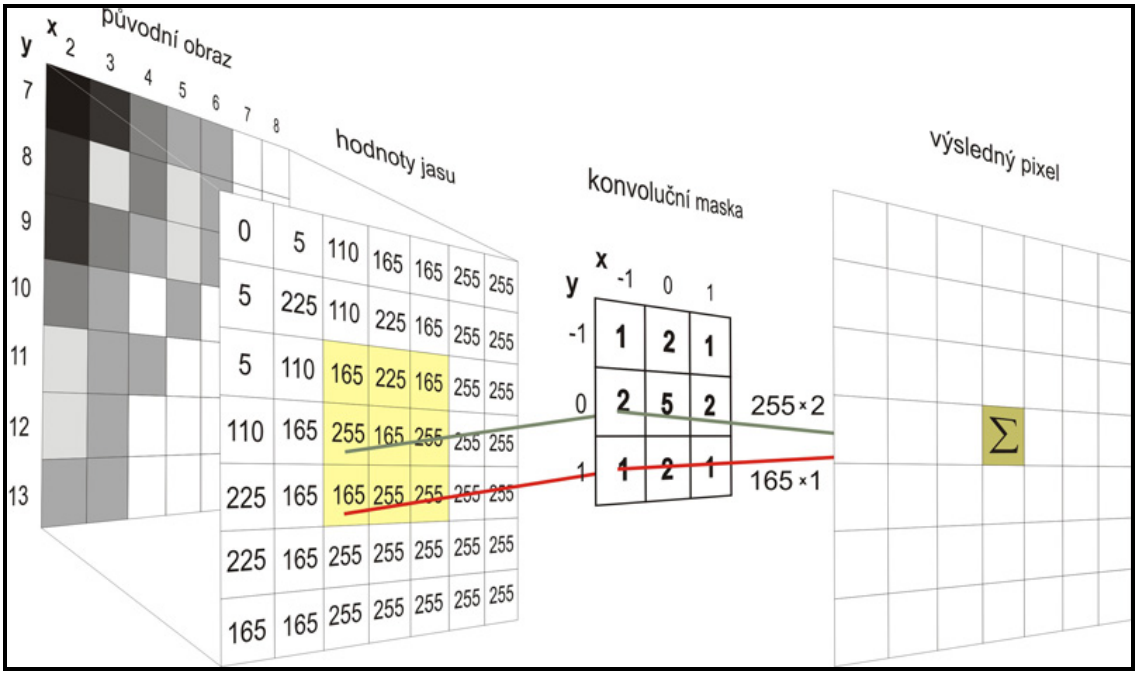
Jak jsem již zmínil, nejlépe bychom odlišili šum tím, že bychom jednoduše znali původní funkci. Toto sice v našem případě možné není, ale můžeme se k původní funkci alespoň přiblížit. Stačí, když budeme mít pořízen obrázek konkrétního místa opakovaně, což v normální fotografii běžné není, ale při nasazování detekčních algoritmů se zpravidla sleduje právě jedno místo a tedy opakovaných snímků máme obrovské množství. K redukci šumu tak lehce postačí porovnat mezi sebou jednotlivé pixely a buď brát výslednou hodnotu jako jejich průměr a nebo lépe nejčastěji opakovanou hodnotu, čímž si výsledek nepokazíme nějakým extrémním výkyvem. Takovéto vylepšení obrazu nemá smysl provádět pro každý snímek vstupující do detektoru. Nejen, že by to bylo výkonově velice nákladné, ale navíc bychom si průměrovali i snímky, kde dochází k nějaké významné změně právě vstupem objektu do popředí. Proto se tato metoda používá k úpravě referenčního statického snímku, který

chceme mít co nejdokonalejší. Při vyladěném referenčním snímku pak lépe fungují detekční metody, které pracují na bázi porovnávání histogramů i metody rozdílů mezi snímky a jejich modifikace.

Referenční snímek se generuje zpravidla při inicializaci a nastavování správných hodnot pro konkrétní prostředí nasazení detektoru. Zde ani v praxi nevadí, že jde o časově náročnější operace, avšak mnohdy bychom potřebovali nějakým způsobem vylepšit každý snímek vstupující do detektoru. Nejčastěji se jedná o redukci šumu pouze v samotném snímku bez kooperace s okolními. Filtr v tomto případě zkoumá okolí jednotlivých pixelů a na jeho základě usuzuje jestli hodnota do okolí přirozeně zapadá, a nebo je natolik odlišná, že jde pravděpodobně o šum. Tyto filtry pracují na principu konvoluce a nebo lokální statistiky okolí. Taková úprava snímku před vstupem do detektoru významně zvyšuje pravděpodobnost správného vyhodnocení, ale ani dobře optimalizovaný algoritmus je nedokáže provádět tak rychle pro všechny pixely obrazu, aby se dali v praxi efektivně použít.V testovací aplikaci jsou tyto filtry zahrnuty, aby se dalo ověřit, jak moc dokáží detekčním algoritmům pomoci.

2.3.2 Ostření a rozmazání obrazu metodou konvoluce

Princip fungování konvoluce je znázorněn na **(Obrázek 6)**, převzato z **[Wiki06]** a bude v dalším textu používán.



(Obrázek 6) Princip diskrétní dvourozměrné konvoluce

Na příslušné místo obrazu „položíme“ konvoluční masku (resp. jádro konvoluce), což je dvourozměrná matice hodnot, která svými koeficienty určuje, jak silnou roli při průměrování bude překrytý pixel mít. Každý pixel překrytý maskou vynásobíme koeficientem z tabulky a provedeme součet všech těchto hodnot. Výslednou hodnotu musíme podělit číslem, které vznikne součtem koeficientů masky tak, aby hodnota konvoluční jádra **h** podle vzorce **(5.)** byla rovna jedné (v našem příkladu tedy podělíme číslem **17** u $\mathbf{h}_1$ a **3** u $\mathbf{h}_2$). Pokud by hodnota byla vyšší, tak by se výsledný obraz zesvětloval, naopak u nižší by se ztmavoval, což nechceme.

$$(5.) \qquad h_1 = \frac{1}{17}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 5 & 2 \\ 1 & 2 & 1 \end{bmatrix} \qquad h_2 = \frac{1}{3}\begin{bmatrix} 0 & -2 & 0 \\ -2 & 11 & -2 \\ 0 & -2 & 0 \end{bmatrix}$$

Jaký efekt bude mít konvoluce na obrázek je dáno koeficienty konvoluční masky. Pokud by byly všechny stejné, tak by se obraz rovnoměrně rozmazával a v případě vícenásobného průchodu bychom mohli dostat až obraz jedné barvy, která by byla průměrem hodnot všech pixelů. Rozmazáním se zbavíme náhodného šumu, který zpravidla vypadá jako různě rozmístěné osamocené pixely s viditelně odlišnou barvou od svého okolí. Bohužel si tak rozmažeme i hrany a zejména můžeme negativně ovlivnit menší objekty v popředí, které by nám mohli splynout s pozadím. Proto maska většinou není takto pravidelná. Například snížením koeficientů směrem od středu tak, jak ukazuje maska $\mathbf{h}_1$ vzorce **(5.)** získáme tzv. Gaussův filtr, který je dobrým kompromisem mezi odstraněním šumu a zachováním hran i u menších objektů.

Po odstranění šumu Gaussovým filtrem lze obraz zase zpětně zaostřit, čímž můžeme vylepšit rozmazáním snížené rozdíly mezi popředím a pozadím. Při ostření je pro nás stěžejní zvýraznění hran. Hranu si můžeme představit jako místo ke dochází k výrazné změně sousedních pixelů.

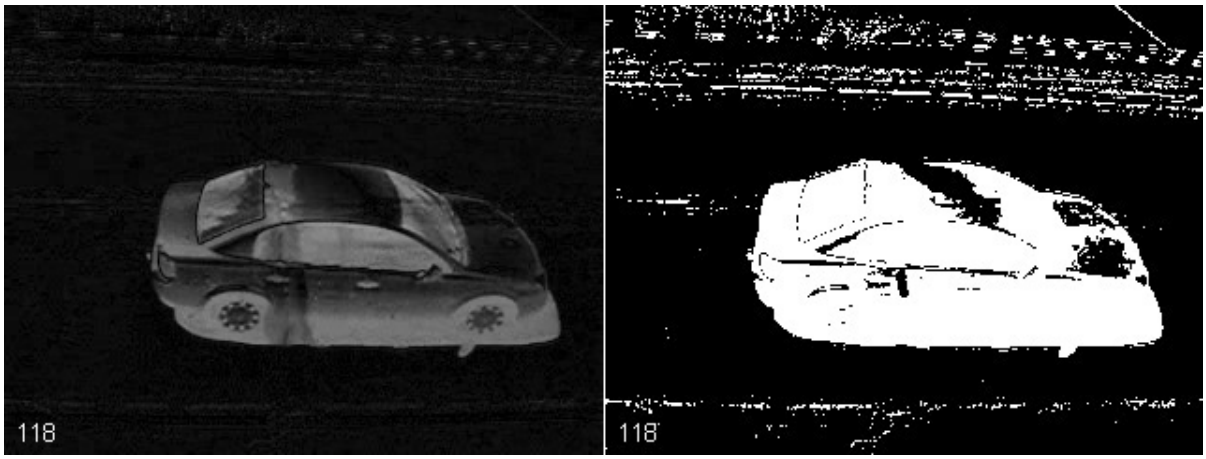
Zaostření lze provádět různými metodami, z čehož nejznámější jsou založeny na použití Sobelova nebo Robertsova operátoru. Pokud však máme hotový algoritmus pro konvoluci použitý při rozmazání, tak je nejjednodušší pouze upravit vhodně koeficienty masky, třeba do podoby koeficientů $\mathbf{h}_2$ u vzorce **(5.)**, který obraz zase zaostří.

2.3.3 Prahování (Thresholding)

Zde budeme diskutovat filtr, který převede obraz do monochromatické, zpravidla černobílé reprezentace.

Rozhodování, zda-li pixel bude mít barvu černou nebo bílou, je stanoveno prahovou hodnotou, kde na základě intenzity jasu pixelu pro hodnotu vyšší než prahová, bude mít výsledně černou a u nižší bílou barvu. Jakým způsobem se získá hodnota jasu pixelu je závislé na použitém barevném modelu. U **RGB** jde o převod do odstínů šedi, u **YUV** se bere složka **Y** a u **HSV** je to složka **V**. Tento filtr se nepoužívá na úpravu snímku před vstupem do detektoru, protože při jeho aplikaci je odstraněna většina informace a výsledkem jsou pouhé siluety objektů.

Významnou roli hraje třeba u detektoru, který pracuje na základě rozdílů mezi dvěma snímky a umožňuje provést finální výpočet, kolik je mezi snímky odlišných pixelů. Výsledek po aplikaci filtru prahování na obrázek, který vznikl odečtením pixelů referenčního snímku od aktuálního, ukazuje **(Obrázek 7)**. Takový detektor bude detailně popsán později v kapitole **3.2**.

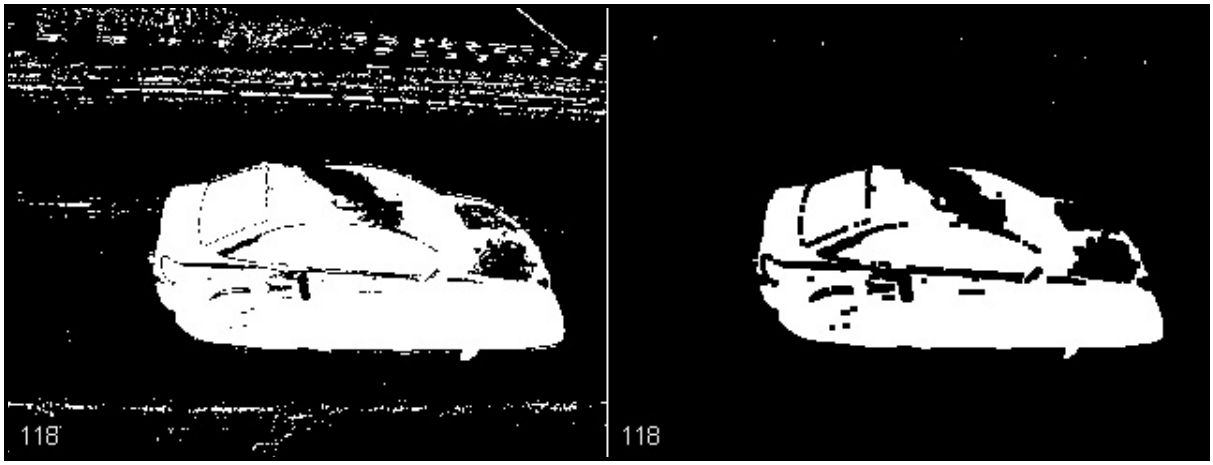


(Obrázek 7) Ukázka původního snímku a jeho podoby po aplikaci filtru prahování

2.3.4 Erosion

Erosion filtr pracuje pouze nad monochromatickým obrazem a používáme jej na místech, kde potřebujeme vyplnit celistvé siluety objektů jednou barvou a odstranit náhodné osamocené pixely. Toto se hodí zejména u detektorů, které na základě odečtení dvou snímků odhalí pixely, ve kterých se mezi sebou liší, ale vlivem šumu nám na místech objektů z popředí vzniknou nespojité siluety a jinde zase osamocené body, které nemají s popředím nic společného.

Princip je jednoduchý a podobně jako u konvolučních filtrů se prochází jednotlivé pixely obrazu s tím, že se vždy vezme nejmenší hodnota celého okolí (zpravidla okolí 3 x 3) a přiřadí se jako nová hodnota pixelu. Tím se dosáhne jednolitého vyplnění siluet na úkor menšího posunu vnější hranice směrem dovnitř. Většina obrazu tedy bude černá, což zahladí osamocené pixely způsobené šumem a větší souvislé oblasti popředí zůstanou bílé. Na **(Obrázek 8)** je pěkně vidět jak erosion filtr po aplikaci na prahovaný obrázek odstranil množství pixelů, které neměli s automobilem v popředí nic společného.



(Obrázek 8) Aplikace filtru erosion (vpravo) na prahovaný obrázek (vlevo)

3 Metody detekce pohybu v obraze

Detekce pohybujících se objektů ve videozáznamu hraje důležitou a často stěžejní roli v mnoha aplikacích počítačového vidění. Tyto aplikace již dávno dosáhly v různých odvětvích lidské činnosti úspěšného praktického využití a je kladen důraz na jejich rozšiřování a vylepšování. Zmiňme například aplikace pro sledování vozidel v autodopravě (monitoring dodržování pravidel silničního provozu, výběr mýtného na zpoplatněných úsecích silnic, hledání odcizených vozidel, atd.), sledování a rozpoznávání lidí (bezpečnostní systémy pro hlídání prostor s omezeným přístupem, monitoring počtu lidí na sledovaném úseku - tvorba statistik, hledání konkrétních osob v záznamu, atd.), rozhranní člověk-stroj (obsluha stroje na základě extrakce určité informace z obrazu, rozpoznání přítomnosti uživatele u počítače, atd.), vojenské aplikace (sledování cíle, detektory pohybu, atd.) a jistě bychom našli mnohé další.

Tradiční cestou k rozpoznání pohybu je oddělení pozadí od popředí, kde jako pozadí bereme tu část obrazu, na které chceme zjišťovat změny a popředí v sobě nese právě ty změny. Pozadí mezi snímky záznamu většinou nebývá statické (úplně beze změny), ale nejčastěji je plné menších i větších změn, které při vyhodnocování detektory chceme stále považovat za pozadí. Může jít o změny vlivem šumu, který produkuje snímací zařízení, změny v osvětlení vlivem denní doby, běžný pohyb venkovního prostředí jako stromy ve větru, apod. Existují mnohé metody, které řeší ten či onen problém, avšak s různými nároky a výpočetní složitostí. To staví vývojáře před otázku: „Která metoda je pro mou aplikaci ta nejvýhodnější?“.

Mým cílem je prověřit obecné implementace několika základních myšlenkových návrhů, ze kterých se velká většina dnešních algoritmů odvozuje. Konkrétně půjde o metody založené na:

- porovnání histogramů jasové charakteristiky obrazu
- počtu rozdílných pixelů mezi snímky na základě hodnoty po jejich odečtení
- porovnání histogramů tvořených koeficienty **LBP** jednotlivých pixelů

V této kapitole budu popisovat princip fungování nejen algoritmů, které jsem použil pro testování ve své aplikaci, ale také jiných přístupů, které lze pro účel detekce využít, ale jejichž implementací jsem se dále ve své práci nezabýval.

3.1 Porovnávání histogramu mezi snímky

V úvodu k této metodě bych rád podotknul, že bychom těžko našli jednodušší metodu pro detekci pohybu ve scéně. Algoritmus pouze porovnává světelnou charakteristiku (histogram hodnot jasu pixelů) aktuálního snímku se stejnou informací u některého jiného, který reprezentuje statickou scénu bez pohybujících se objektů, neboli pozadí. Je zřejmé, že jakákoli změna ve sledované scéně způsobí, že aktuální snímek bude vykazovat v histogramu odlišnou jasovou charakteristiku. Tato metoda je výpočetně málo náročná, protože nejméně jeden histogram pro referenční snímek máme vždy již dopředu vytvořený a ušetříme režii jeho výpočtu při porovnávání. Navíc si můžeme dovolit tento snímek ještě před získáním jasového popisu zbavit šumu i náročnějšími filtry, protože jde o operaci prováděnou jen jednou za čas. Na druhou stranu je tato metoda velice náchylná ke změnám v prostředí, které se ne vždy dají dopředu naplánovat a správná aktualizace referenčního snímku je bezpochyby problematika sama pro sebe. Cílem vývojářů je automatizovat činnost tak, aby i dlouhodobě fungovala bez zásahu člověka. K tomu je nutné důkladně myslet na podmínky reálného světa a samozřejmě i na konkrétní prostory nasazení.

S aktualizací referenčního snímku podle denní doby, a tomu podobným problémům, se lze vypořádat ještě před nasazením detektoru, ale mnohé jiné negativní faktory je třeba řešit přímo za běhu. Jako vždy, nepřítelem číslo jedna je šumová informace, která se k původní informaci dodatečně přidala některým z vlivů popsaných v kapitole o šumu **2.3.1** a nyní je její nedělitelnou (resp. těžko dělitelnou) součástí. Potlačení této negativní složky je do určité míry možné použitím filtrů, ale za cenu zvýšení výpočetních nároků. Protože s jistou dávkou šumu vždy počítáme, tak součástí vyhodnocení této detekční metody je i uživatelem stanovená hranice, nad kterou teprve zjištěné změny mezi obrazy má smysl považovat za pohyb, respektive pod touto hranicí je všechen pohyb brán jako chyba vlivem šumu.

3.1.1 Možnosti využití

Tato metoda má smysl zejména při implementaci programů typu „alarm“, které upozorní, že se na sledovaném místě něco děje, ale ve své základní podobě nedokáže vyznačit místo pohybu. Využití bychom našli třeba při sledování prostor s neproměnlivými světelnými podmínkami, kupříkladu skladiště, které po celou dobu zachovává konstantní světelné podmínky a jediným proměnlivým faktorem zůstává šum. Pokud bychom metodu implementovali do programů pro sledování prostor s proměnlivými podmínkami, například světelnými, v závislosti na denní době, nebo plánovanými

změnami, na které nemá být brán zřetel, není možné pracovat s pouze jedním referenčním snímkem statické scény, ale musíme jej aktualizovat. Výpočetně nejméně náročné je vytvářet průměrný referenční histogram průměrováním se sekvencí snímků, ve kterých detektor neoznačil pohyb. Tím se vylepší nejen schopnost aplikace poradit si se šumem, ale zejména se změnami v osvětlení vlivem počasí, denní doby, apod.

Nevýhodou této metody je buď silná reakce na každou změnu v případě použití pouze jednoho statického snímku a nebo naopak slabá reakce na pomalu se pohybující nebo malé objekty, které se kvůli průměrování výchozího snímku velice rychle stanou součástí pozadí a alarm na ně nebude reagovat. Protože tato metoda ve své podstatě reaguje pouze na změnu poměru odstínů mezi aktuálním a referenčním snímkem, tak jak jistě uvidíme v kapitole **5** testování, je pro ní velký problém vyvolat alarm, pokud se v obraze objekt pohybuje, ale celková barevná charakteristika obrazu zůstane stejná. Jako příklad si lze představit video sekvenci, která zabírá točící se vícebarevné kolo na pouťové střelnici. Zde nepochybně dochází k pohybu, ale rozložení barev v obraze se prakticky nemění a obecná verze algoritmu by v tomto případě nezaznamenala vůbec žádný pohyb.

3.1.2 Specializace a optimalizace

Pokud budeme pracovat pouze s nejzákladnější podobou algoritmu, a to jeden histogram pro celý snímek, tak přicházíme o možnost vysledovat, na kterém místě se pohyb udál. Toto ale nemusí být na škodu, protože se tak stává tato metoda asi vůbec nejrychlejší a na druhou stranu řešení existuje. Tím řešením je rozložit snímek na více podprostorů a pro každý pak vytvořit a porovnávat samostatný histogram. Pokud se toto rozložení udělá rozumně a přímo pro konkrétní scénu, tak na místech, kde pohyb očekáváme může být hustší síť menších oblastí, a tím se získá velice přesná hranice pohybujícího objektu. Naopak na místech, kde to není pravděpodobné, bude síť řidší, čímž šetříme výkon. V kombinaci s jednoduchou verzí jednoho či několika málo histogramů, které budou fungovat jako alarm a ve chvíli alarmu se začne počítat histogramů více, dostáváme velice silnou metodu, která je rychlá a zároveň přesná, i když stále je nutno brát v potaz zmíněná další negativa. Rozložení scény na více prostorů řeší dříve zmíněný problém kola na pouťové střelnici, protože pokud se kolo v rozložení promítne do více oblastí, tak se barevná charakteristika bude měnit naopak velice výrazně.

3.1.3 Idea implementace detektoru

Porovnávání dvou snímku v tomto případě spočívá v porovnávání histogramů jejich jasové složky. Histogram se implementuje jako jednorozměrné pole, kde každý prvek reprezentuje určitou hodnotu jasu a jeho hodnota udává počet takových pixelů ve snímku. Abychom vygenerovali histogram, budeme muset spočítat kolik pixelů s touto hodnotou je v něm zastoupeno. Jasová hodnota se vezme buď přímo jako jedna ze složek barevného modelu, nebo převodem na odstíny šedi podle kapitoly **2.2.3**. Často je výhodné snížit rozsah histogramu s tím, že jeden jeho oddíl bude reprezentovat určitý interval jasových hodnot pixelu. Porovnávání tak bude rychlejší a navíc se tak sníží chyba vlivem lehce odlišných odstínů mezi snímky.

Pokud jsou snímky podobné tak i jejich histogramy by měly vykazovat jen malé změny. Proto korespondující oddíly obou histogramů odečteme a výsledný histogram reprezentuje právě odlišné pixely, které by ideálně měli být body popředí. V tomto histogramu se už pouze sečtou oddíly (počty odlišných pixelů) a výsledek se porovná s hranící stanovenou uživatelem, a která při překročení vyvolá alarm detektoru.

3.2 Sledování rozdílných bodů mezi snímky

Když si vzpomeneme na oblíbenou hru „najděte mezi dvěma obrázky deset rozdílů“, tak nám v jisté podobě bude nevědomky hlavou probíhat algoritmus této metody. Naštěstí smysly člověka jsou stále ještě proti počítači lepší, a proto dokáže hledat rozdíly aniž by se trápil tím, že obrázky, které právě sleduje v časopise, vyšly z tiskárny v trošku odlišných odstínech, že si na ně otiskl kolečko ranním šálkem kávy a už vůbec se nebude otravovat tím , že by obrázky porovnával vzájemně po nejmenších viditelných bodech. To vše kolem, co se ani netýká samotného porovnávání a člověk nad tím nepřemýšlí, se stává tím největším problémem k překonání při zpracování počítačem. Tato metoda je tedy sama o sobě velice jednoduchá a v kostce funguje tak, že jen porovnáme korespondující pixely mezi dvěma snímky videa, které v případě výrazně odlišné hodnoty označují pohybující se objekt. Pokud by oba snímky byly ideální a netrpěly by v přeneseném slova smyslu neduhy odlišností při tisku a kolečka od kafe, tak bych pravděpodobně v celé své práci ani jinou metodu uvádět nemusel. Měli bychom metodu, která je rychlá, přesná a navíc nepotřebuje speciální modifikace, aby přesně označila pohybující se objekt. Ideální snímky ale samozřejmě v reálném světě nemáme ...

U metod založených na histogramu je zpravidla jeden z histogramů předpočítán a v každém kroku se tedy vytváří jen histogram aktuálního. Tato metoda bude v každém kroku pracovat v nejhorším případě s každým pixelem obou snímků, což je výpočetně náročné, a proto na redukci počtu zpracovávaných pixelů by se mělo při optimalizaci myslet na prvním místě.

Když budeme hledat rozdíly na dvou barevných snímcích, a pak je zkusíme najít na stejných, ale v černobílé podobě, tak to pro nás nebude pravděpodobně o nic těžší (pokud rozdílem není právě změna barvy). Redukce barevné informace na odstíny šedi neodstraní informaci o celkovém uspořádání scény. U barevných pixelů musíme porovnávat tři složky oproti jedné u odstínů šedi, a proto pouhým použitím jiného barevného modelu můžeme uspořit algoritmu čtyři přístupy k informaci z šesti a výrazně ho tím urychlit. Pro převod snímku do odstínů šedi lze použít dva přístupy. Buď místo tradičního **RGB** modelu budu mít obrazovou informaci uloženu v jednom z modelů **HSV**, **YUV** nebo přímo **RGB** převedu z 24 bitové reprezentace na 8 bitovou podle vzorce **(4.)**. U **HSV** a **YUV** je výhodou, že si stále uchovávám veškerou barevnou informaci snímku, ale pokud mi na ní nezáleží, tak je možno uspořit paměť a převést pouze na 8 bitovou reprezentaci odstínů šedi. Bližšímu popisu modelů a převodům mezi nimi je věnována kapitola **2.2**.

U histogramových metod jsme se snažili ještě před startem detektoru vypořádat se šumem, tady tomu bude přesně naopak a poradíme si s ním v průběhu zpracování snímku. Jednotlivé obrázky v následujícím textu budou znázorňovat postup zpracování obrazu od vstupu snímku do detektoru po jeho označení atributem je/není pohyb.

(Obrázek 9) Aktuální a referenční snímek vstupující do rozdílového detektoru

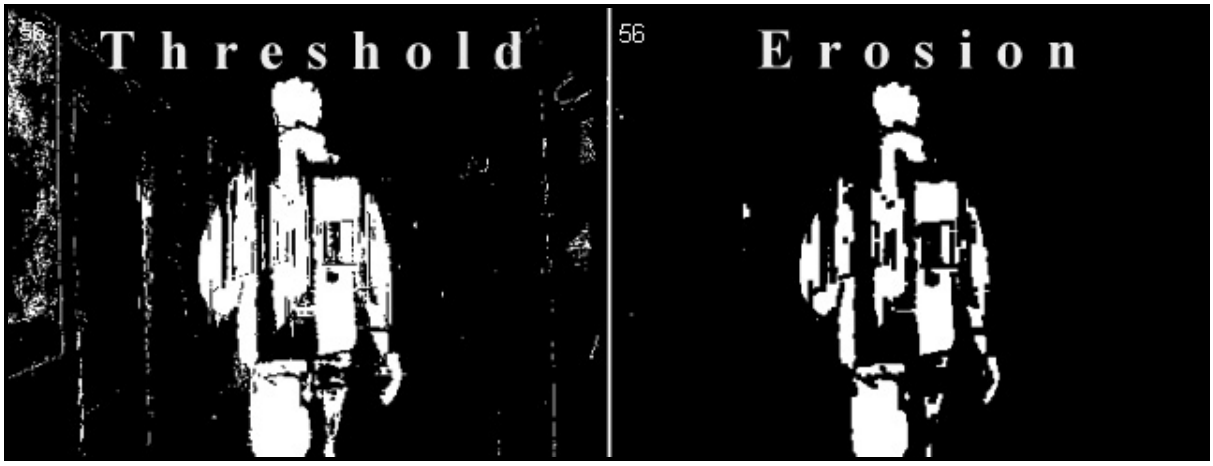
Do detektoru vstupuje aktuální snímek (**Current**), na kterém se snažíme najít objekty v popředí a referenční snímek (**Reference**), který v sobě nese pouze pozadí **(Obrázek 9)**. U obou snímků se bude dále pracovat pouze s nebarevnou informací, kterou lze získat některým z výše popsaných způsobů (na obrázku jde o převod do stupňů šedi **Grayscale**). Oba snímky od sebe v této podobě odečteme a získáme nový snímek, který je na **(Obrázek 10)** nadepsaný jako **Difference**, česky rozdíl. Můžeme si všimnout, že čím méně se od sebe korespondující pixely liší, tím se jejich hodnota bude logicky více blížit nule, tedy černé barvě. Šumová informace se nám nyní promítá do míst, která jsou světlejší, ale nejsou uvnitř obrysu postavy. Zpravidla by mělo platit, že místa, která se mezi snímky lišila mnohem výrazněji, a tedy pravděpodobně budou součástí obrysu postavy, budou mnohem světlejší (hodnota po odečtení pixelů se pohybuje dál od nuly).

(Obrázek 10) Převod snímků z předchozího obrázku na stupně šedi a jejich odečtení (vpravo)

Potřebujeme tedy stejně jako u jiných metod stanovit hranici šumu a spočítat, kolik pixelů se pohybuje nad touto hranicí. V této chvíli by bylo výpočetně nejméně náročné projít postupně všechny pixely a spočítat, kolik z nich se nachází nad uživatelem definovanou hranicí. Pokud si však můžeme dovolit „plýtvat“ výkonem, abychom se dostali k přesnějšímu výsledku, případně chceme v obrazu vyznačit pohybující se objekt, tak před spočtením pixelů aplikujeme pár dalších filtrů. Dále použité filtry jsou podrobněji popsány v kapitole **2.3**.

Cílem následujících kroků je z 8 bitového obrazu ve stupních šedi extrahovat 2 bitovou černobílou masku, ve které budou mít pixely popředí bílou a pozadí černou barvu. Aplikujeme prahovací filtr, který porovná hodnotu jednotlivých pixelů s uživatelem stanovenou hranicí, a pokud je hodnota větší, bude výsledný pixel bílý, jinak černý. Vznikne dvoubitová maska, jak ji vidíme na **(Obrázek 11)** vlevo s popisem **Threshold**. Obrys postavy v popředí je krásně zřetelný, ale i tak je uvnitř stále

spousta pixelů špatně označených, nehledě na množství roztroušených mnohdy osamocených pixelů vlivem šumu. Díky tomu, že roztroušené pixely šumu nevytvářejí jednolité celky, zde bude krásně fungovat filtr **Erosion**, který odstraňuje osamocené pixely a vyplňuje vnitřky spojitých ploch. Takto získaná maska v sobě nese dominantní oblasti popředí s minimem šumové informace a je vhodná jak pro konečné spočtení množství odlišných pixelů, tak pro vizuální zvýraznění odlišných míst.



(Obrázek 11) Aplikace prahovacího (vlevo) a erosion filtru (vpravo)

Ať už s použitím threshold a erosion filtru nebo bez nich, stačí nakonec spočítat množství odlišných pixelů a výsledné číslo detektor porovná se svou hranicí pro sepnutí alarmu.

3.2.1 Možnosti využití

Tato metoda, kvůli nutnosti pracovat vždy s oběma snímky, nebude patřit mezi nejrychlejší, ale její velkou výhodou je, že bez dalších mechanismů nám může ukázat místa, kde přesně k pohybu dochází. Právě snadnost vizualizace výsledků, stejně jako nenáročná implementace, z ní dělá dobrého kandidáta pro demonstrativní účely ve výuce. Praktické využití se nabízí spíše až v kombinaci s nějakým vhodným algoritmem typu alarm, který bude výpočetně méně náročný a ve chvíli, kdy rozpozná pohyb, tak teprve vyvolá rozdílovou metodu a „vytáhne“ z obrazu oblasti popředí. Obraz, který je zbavený pozadí usnadňuje práci rozpoznávacím algoritmům (např. rozpoznávání obličeje, značek na automobilech, apod.), které jsou sami o sobě výpočetně velice náročné.

Tato metoda si dokáže pomocí dodatečných filtrů velice dobře poradit se šumem, i když tím roste počet nutných zásahů uživatele, který mimo nastavení hranice alarmu musí provést nastavení i těchto dalších filtrů. Každé konkrétní nastavení bude dobře fungovat jen v dále neměnných podmínkách a

při vnější změně klesá jeho rozlišovací schopnost. Stejně jako u histogramových metod se tento neduh dá částečně eliminovat rozumnou aktualizací referenčního snímku, nebo jeho průměrováním.

Pokud se mezi snímky pozadí příliš nemění a hladina šumu tímto není příliš velká, tak tato metoda dokáže dobře rozlišit (resp. označit) i malé nebo pomalu se pohybující objekty, na které jsou histogramové metody ve své základní podobě relativně slabé. Ani na problémy s objekty, které zachovávají celkovou barevnou charakteristiku obrazu oproti referenčnímu snímku stejnou, u této metody nemusíme vůbec myslet. V porovnání s histogramovými metodami, když vypustíme společná negativa, dostaneme jako hlavní nevýhodu větší systémovou náročnost, která se dá opět snížit optimalizacemi a nebo přesunutím výpočtu filtrů na speciální hardware.

3.2.2 Specializace a optimalizace

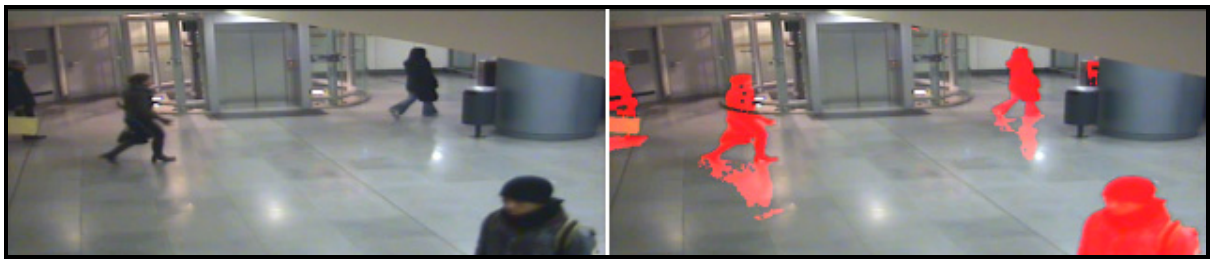
Základní podoba rozdílové metody je sice pomalá, ale už sama o sobě dává dobré výsledky. Optimalizace se tedy budou vztahovat hlavně ke snížení výpočetních nároků. V prvé řadě je třeba redukovat počet zpracovávaných pixelů na hranici, při které se dostaneme k dobrému poměru mezi rychlostí a úspěšností metody. Tato hranice je individuální a její přesné určení závisí na prostředí nasazení algoritmu.

Obecně platí, že pro detektor typu alarm je dostačující i poměrně nízké rozlišení (při této práci byl úspěšně testován i detektor na záběrech z webové kamery v rozlišení 320x240), naopak v situacích, kde chceme vyznačit objekt pro následné rozpoznávání je lepší ponechat vyšší rozlišení a upravit algoritmy tak, aby pracovali jen s některými pixely. Dobrých výsledků lze dosáhnout třeba přeskakováním řádků, kdy například při vynechání každého sudého řádku snížíme počet pixelů na polovinu, ale celkový charakter popředí se nijak významně nenaruší a dokonce ani po aplikaci erosion filtru nedochází k rozkouskování původně celistvých oblastí.

(Obrázek 12) Maska tvořena z každého pixelu obrazu a maska spočtena jen u sudých řádků

Pokud budeme v obraze vyznačovat místa, kde k pohybu došlo, není problém masku, která má kvůli přeskakování řádků nižší rozlišení, pouhým zdvojením řádků dostat zpět do původních rozměrů. Když srovnáme masku na **(Obrázek 12)** vlevo, která vznikla poctivým výpočtem každého pixelu, s naší zdvojenou na obrázku vpravo, tak je vidět, že se od sebe už na první pohled příliš neliší.

Úpravu původního snímku, aby se vizuálně zvýraznily oblasti popředí, lze dosáhnout aplikací masky s tím, že se porovnají korespondující pixely a tam kde je pixel v masce černý ponecháme původní hodnotu, naopak, kde je bílý zvýšíme jednu barevnou složku na maximum. Dostaneme tím podobný obraz jako **(Obrázek 13)**, ve kterém má popředí pouze odlišný barevný nádech (podle zvýrazněné složky) a celá informace scény jinak zůstane zachována.



(Obrázek 13) Demonstrace barevného odlišení míst s pohybem

3.2.3 Idea implementace detektoru

Jak bylo nastíněno v teoretickém náhledu, tak do naší metody budou vstupovat dva sousední snímky (resp. bitmapy) videosignálu. Tyto snímky nejen kvůli vytváření rozdílu, ale hlavně kvůli aplikaci dalších filtrů je třeba převést na některý barevný model, který nám umožní dále pracovat jen s jednou složkou obrazu.

Procházíme jeden i druhý snímek po pixelech na stejných souřadnicích (optimalizace vypouštěním některých pixelů, či rovnou celých řádků jsou na místě), provedeme jejich odečtení a získané číslo v absolutní hodnotě zapíšeme do připravené třetí bitmapy. Jakmile projdeme všechny pixely, tak bitmapa, do které byly zapisovány výsledky, v sobě nese informaci o všech rozdílech.

Tam, kde se mezi snímky nic nemění by měla být nulová hodnota (černá), ale to se kvůli i třeba lehkým změnám v odstínech nebo šumu povede jen málokdy. Proto na tento obrázek aplikujeme filtr „prahování“ a dostaneme binárně reprezentovanou bitmapu, kde například bílé pixely budou určovat místa se změnou (hodnoty vyšší než nastavený práh) a černé místa beze změny. Pokud bychom chtěli pouze vyvolat alarm, tak by stačilo spočítat bílé pixely a v případě, že překročí nějakou mezní hodnotu říct, že došlo k pohybu. Tato metoda, ale jak už jsem říkal dokáže určit i „místo činu“, proto by bylo škoda nějakým vizuálním způsobem toto místo nevyznačit. Je zřejmé, že pohybující se objekt bude reprezentován nějakým větším shlukem bílých pixelů. Proto na obrázek aplikujeme filtr „erosion“, který odstraní osamocené pixely a vyplní souvislé plochy. Získáváme tak masku, kterou vhodně aplikujeme na původní obrázek a místa, kde byl nalezen pohyb třeba barevně zvýrazníme nebo ohraničíme polygonem.

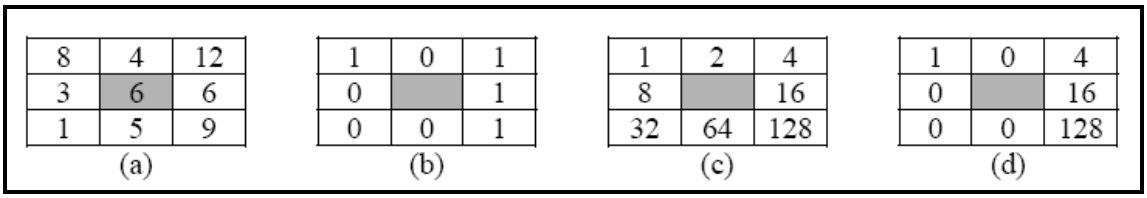
3.3 Algoritmus Local Binary Patterns

Local Binary Patterns (dále jen **LBP**) je oproti ostatním metodám nejmladší, vznikla teprve před pár lety na Finské univerzitě v Oulu a jejímu dalšímu výzkumu se věnuje skupina počítačového vidění na technologické fakultě. Její hlavní využití spadá do oblasti rozpoznávání objektů v obraze a tamní skupina se převážně zabývá algoritmy pro rozpoznání obličeje. Na svých stránkách **[REF01]** uvádí, že jejich **LBP** již byla úspěšně nasazena na několika místech v praxi. Například v průmyslových aplikacích při rozpoznávání defektů v materiálu, při analýze snímků v biomedicínckých aplikacích, pro katalogizaci snímků podle obsahu a dokonce pro svojí vysokou efektivitu i na místech, které vyžadují realtime zpracování obrazu. Vedle nízkých systémových nároků a vysoké rychlosti zpracování dominuje kladům ještě jedna stěžejní vlastnost, a tou je tolerance ke změnám osvětlení. Díky ní je metoda schopná sama korigovat světelné změny, které nastaly ve scéně vlivem postupující denní doby, dokonce si poradí i s náhlými změnami, například při zastínění scény mrakem či snadno vyrovná periodické změny vlivem kolísání proudu v žárovce, apod.

Po kladech rád uvádím zápory, ale u této metody jsem nenarazil na žádné charakteristické negativum, protože neúspěch při detekci je zpravidla dílem nedokonalého algoritmu konečného vyhodnocení scény na základě získaných koeficientů **LBP** jednotlivých pixelů.

Oproti předchozím popsaným algoritmům, které si šlo lehce představit a v základu šlo o jednoduché principy, je **LBP** složitější, ale ne z titulu náročné implementace, spíše tím, že poté co LBP popíše jednotlivé pixely obrazu, nelze jej znovu zobrazit do nám srozumitelné podoby. Nová hodnota pixelu již nereprezentuje jeho barvu, ale spíše popisuje jeho charakter v rámci okolí, což si lze jen těžko představit.

Samotnému algoritmu, který klasifikuje jednotlivé pixely snímku a přiřadí jim hodnotu, která se odvíjí od jeho pozice v okolí, se říká **LBP** operátor. Výpočet této hodnoty je podobný konvoluci popsané blíže v kapitole **2.3.2** s tím, že neexistuje žádná předem daná konvoluční maska, ale stává se jí u každého pixelu jeho vlastní okolí. Výpočet hodnoty **LBP** pixelu si ukážeme na konkrétním příkladu, jednotlivé kroky jsou znázorněny na **(Obrázek 14)** a na konkrétní krok se budu v textu odkazovat podle jeho písmenka v obrázku.



(Obrázek 14) Demonstrace kroků k získání LBP hodnoty pixelu

LBP pracuje se snímkem bez barevné informace ve stupních šedi. Jaké se zde dají uplatnit výhody jsem již popisoval v předchozích kapitolách a bližší popis převodů je uveden v tomu věnované kapitole **2.2**. Snímek se postupně prochází a v základní podobě se používají hodnoty odstínů šedi osmi přiléhajících pixelů, obrázek písmeno **(a)**.

$$(6.) \qquad f(x) = \begin{cases} 0 & x > y \\ 1 & x \leq y \end{cases}$$

Prahováním hodnoty středového pixelu s okolními podle vzorce **(6.)** , kde **x** je hodnota středového a **y** okolního bodu, získáme sled nul a jedniček tak, jak na obrázku u písmene **(b)**. Následně se těmito koeficienty **0** a **1** vynásobí korespondující čísla váhové matice, která v sobě nese prvky rozvoje 2^n kde

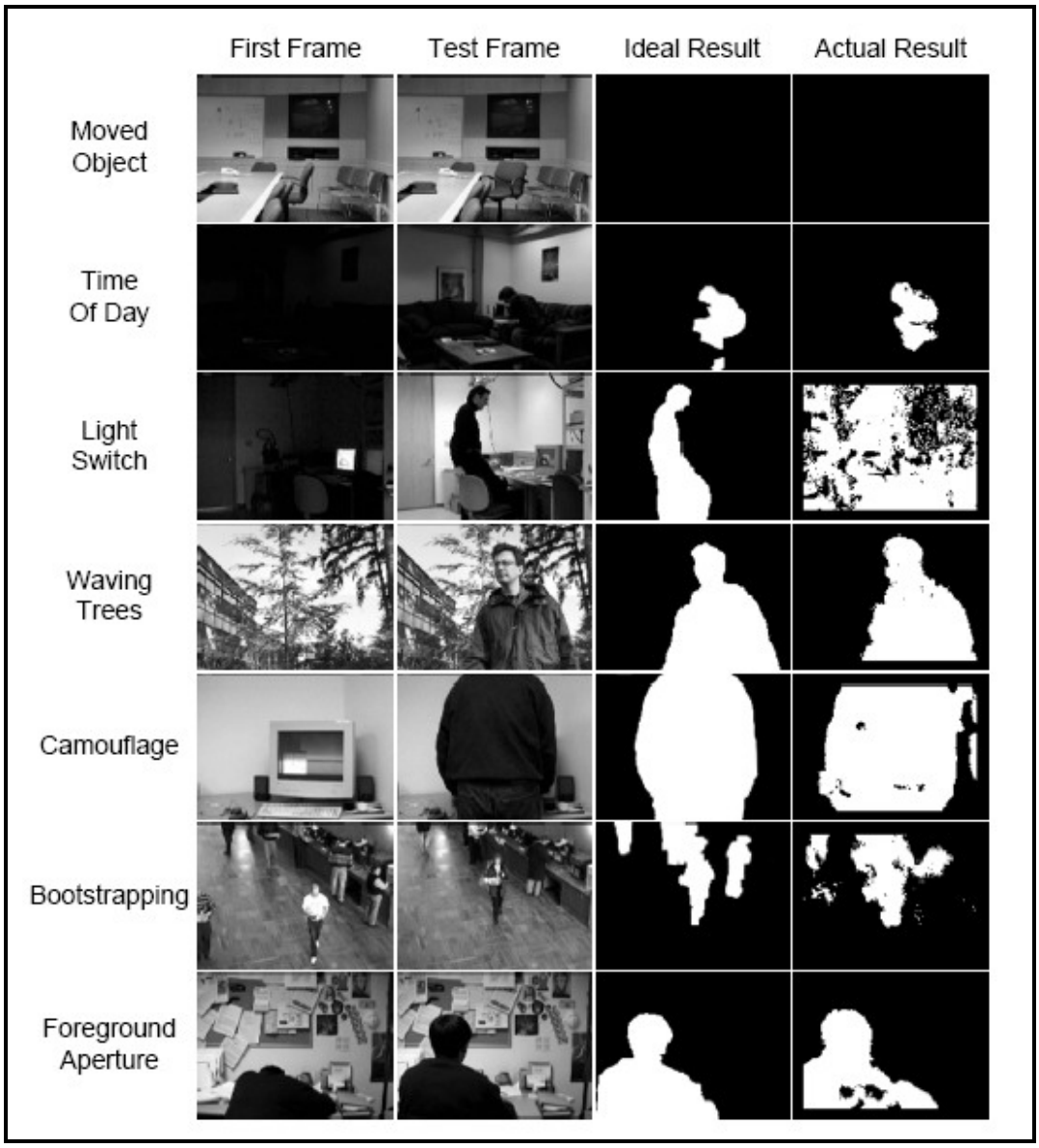
n = <0, 7>, obrázek písmeno **(c)**. Takto získané prvky matice se pouze sečtou a výsledkem je číslo reprezentující hodnotu **LBP** u klasifikovaného pixelu. V daném příkladu byl výsledek **1 + 4 + 16 + 128 = 149**.

LBP operátorem klasifikujeme všechny pixely obrazu s tím, že pokud jde o krajní bod, který nemá úplné okolí, tak jej buď úplně vynecháme a nebo počítáme s hodnotami nula.

Obraz kompletně popsaný místo původních hodnot šedé stupnice koeficienty **LBP** má tu výhodu, že ve chvíli, kdy vypočteme stejnou scénu, ve které se zatím konstantně po celé rozloze změnil jas (vlivem denní doby, apod.), tak nové hodnoty **LBP** se starými si budou velice podobné, ideálně stejné.

Pomocí LBP histogramů, lze pak naprosto stejným způsobem jako jsme tomu dělali v kapitole o porovnávání histogramů jasu pixelů **3.1** vytvořit detektor pohybu, který bude histogramy mezi sebou nějak vhodně porovnávat a na základě velikosti odchylky rozpoznávat pohyb.

Vzhledem k tomu, že **LBP** hodnoty v sobě nesou informaci i o svém okolí, funguje tato metoda dobře i v situacích, kdy je snímáno venkovní prostředí, které nikdy nebude úplně statické. Pokud v něm dochází pouze k malým odchylkám (například pohyb listů na stromech v pozadí, chvění trávy, ...), tak se histogramy mezi snímky nebudou příliš lišit. Jakých zajímavých výsledků v této chvíli dosahují aplikace vytvořené na současných známých variacích **LBP** si lze prohlédnout na **(Obrázek 15)**, který prezentují ve své publikaci **[Oulu06]** přímo členové původní výzkumné skupiny univerzity v Oulu.



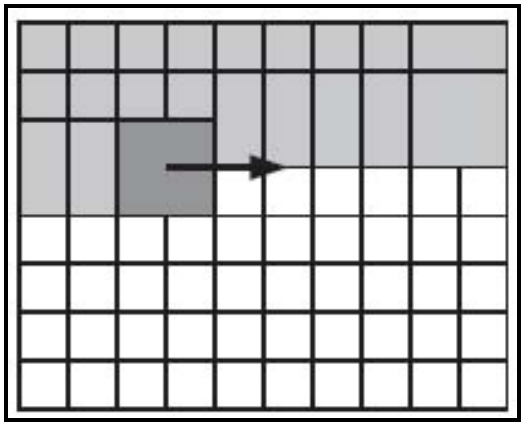
(Obrázek 15) Experimenty s LBP na sadě příkladů v různých podmínkách (převzato z [Oulu06])

3.3.1 Specializace a optimalizace

Narozdíl od jiných metod je **LBP** v základu velice rychlá, proto namísto optimalizací výkonu se můžeme zaměřit na zlepšení její schopnosti rozpoznat pohyb a zejména na vyznačení místa, kde k němu dochází.

Jak již víme, tak z **LBP** koeficientů obrazu se nakonec vytvoří histogram, který se může porovnávat s předchozími snímky. V této podobě samozřejmě nemáme informaci, ze které by šla rozpoznat oblast pohybujícího se objektu. Můžeme však obraz rozdělit na menší bloky a pro ty udělat

samostatný výpočet. Vzhledem k tomu, že výkonově nejnáročnější část při sestavení histogramu je přístup k hodnotám **LBP**, tak i vytvoření většího množství malých bloků obrazu, v porovnání s jedním histogramem celého obrazu, vyjde úplně nastejno. Nabízí se spousta způsobů jak rozestavět bloky. Můžou mít všechny konstantní velikost a nebo na místech, kde je pravděpodobnější výskyt pohybu mít více menších bloků. Nejvýrazněji však ovlivní kvalitu rozpoznání, pokud bloky nebudeme stavět vedle sebe, ale budou se částečně překrývat tak, jak ilustruje **(Obrázek 16)**, převzato z materiálu **[Oulu04]**. Překrývání s sebou ponese nutnost přistoupit k některým hodnotám i vícekrát, ale zato hranice pohybujícího se objektu můžeme velice přesně oddělit od pozadí. Jako příklad, kdy se toto může hodit je třeba rozpoznávání obličeje, kdy získáme velice přesné hranice člověka, ve kterých se dá podle barvy kůže určit místo obličeje a rozpoznávacímu algoritmu se pak předá jen tato konkrétní přesná oblast.



(Obrázek 16) Částečné překrývaní bloků při tvorbě histogramů

3.3.2 Idea implementace detektoru

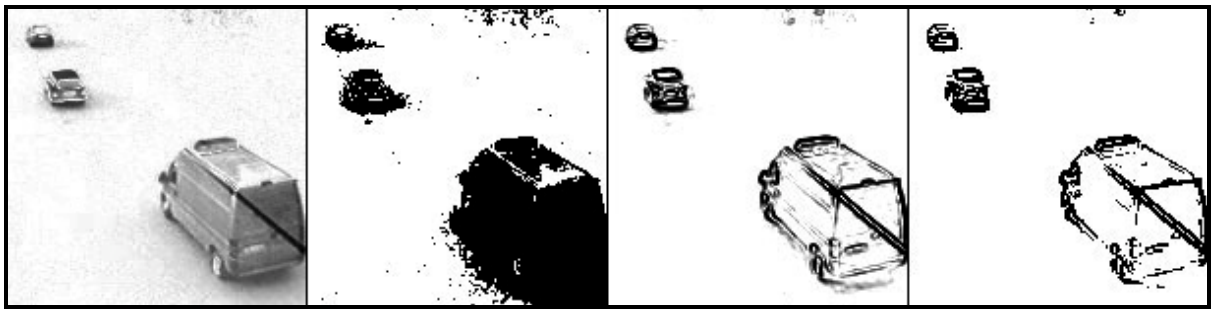
Detektor lze postavit úplně stejným způsobem jako u metody, která porovnávala histogramy jasové složky obrazu. Zde pouze hodnotu světelnosti vystřídá koeficient **LBP**. Histogram **LBP** koeficientů aktuálního snímku by se mohl stejně jako dříve porovnat s některým předchozím referenčním snímkem. U **LBP** se po implementaci porovnávání pouze s jedním snímkem ukázalo, že mnohem lepších výsledků je možné dosáhnout pomocí zachování několika předchozích již vypočtených histogramů, které se zprůměrují a až s takovým se aktuální histogram porovnává. Uživatele tak navíc ušetříme výběru snímku se statickým pozadím někde z video sekvence. Nakonec se provede odečtení korespondujících oddílů od sebe, udělá se suma těchto výsledků a získané číslo se porovná s hranicí, nad kterou se rozlišuje pohyb v obraze.

Díky průměrování se objekty, které do obrazu vstoupily a už v něm dále zůstávají po čase stanou součástí pozadí, což může být někdy výhodné a jindy nežádoucí, proto je nutné jako vždy zvolit správný přístup pro konkrétní místo nasazení.

3.4 Další možné přístupy detekce pohybu

3.4.1 Porovnání snímků zpracovaných detektorem hran

Dalším zajímavým přístupem, který je detailněji popsán v publikaci **[Mason01]**, je aplikace filtru pro zvýraznění hran na obraz před prahováním. Hrany obecně definují okraje objektů, což je informace, která nám pro porovnání snímků bohatě stačí. Výhodou je, že drobný šum, který nepřeroste v souvislé oblasti, se jako hrana nezvýrazní, a tak se jej elegantně zbavíme. **(Obrázek 17)** ukazuje rozdíl mezi výsledkem prahování u obrázku získaného metodou odečtení snímků a po aplikaci hranového filtru. Postupně úplně nalevo je původní rozdílový snímek, další ukazuje výsledek po prahování, na třetím je podoba prvního po aplikaci zvýraznění hran a poslední prezentuje výsledek prahování třetího obrázku. Pro lepší názornost jsou u tohoto obrázku invertovány barvy.



(Obrázek 17) Demonstrace výsledků aplikace prahování na obraz bez a s použitím zvýraznění hran

Ke konečnému vyhodnocení si lze podle situace vybrat jeden z již dříve popsaných přístupů. Prvním může být porovnání na základě histogramů, u kterého by se dál postupovalo stejně jak bylo popsáno v kapitole **3.1**, tedy sestavení histogramů a jejich vzájemné porovnání s možností vyvolat alarm i vyznačit místo objektu. Druhým je princip popsaný v kapitole rozdílové metody **3.2**, kdy jsou od sebe vedlejší snímky odečítány. Zde lze dále postupovat na základě jednoho ze dvou obecných přístupů. Buď dříve popsaným způsobem odečíst snímky, aplikovat prahovací filtr, sečíst bílé pixely (místa kde se výrazně od sebe liší), vyhodnotit množství s danou hranicí a nebo nejprve provést prahování (zůstane pouze černý obraz s bílými oblastmi, kde byly hrany), ten odečíst od stejně upraveného

předchozího snímku a dále stejně sečíst a vyhodnotit počet bílých pixelů. Druhý způsob má výhodu, že zvýrazněné hrany více odliší objekty od šumu a po prahování se ztratí všechen šum, kterému se nepovedlo vytvořit souvislou oblast rozpoznatelnou jako samostatný objekt.

3.4.2 Metoda optického toku (optical flow)

Metoda optického toku dokáže sledovat pohyb pixelu mezi snímky video sekvence a zjistit tak směr a velikost rychlosti pohybu. Toho se může využívat například při předpovědi, na jaké místo by se mohl objekt za pár snímků přesunout nebo pomocí historie přesunu rozlišovat gesta v obličeji, apod. Tato metoda je výpočetně velice náročná a není možné jí sledovat každý pixel v obraze. Založit na ní detektor pohybu by tedy mohlo být jako „jít na komára tankem“, ale pokud budeme mít systém, který sleduje pomocí této metody pohyb objektů a celkově je optimalizován pro její efektivní chod, tak je možné ji v určité podobě použít i jako detektor.

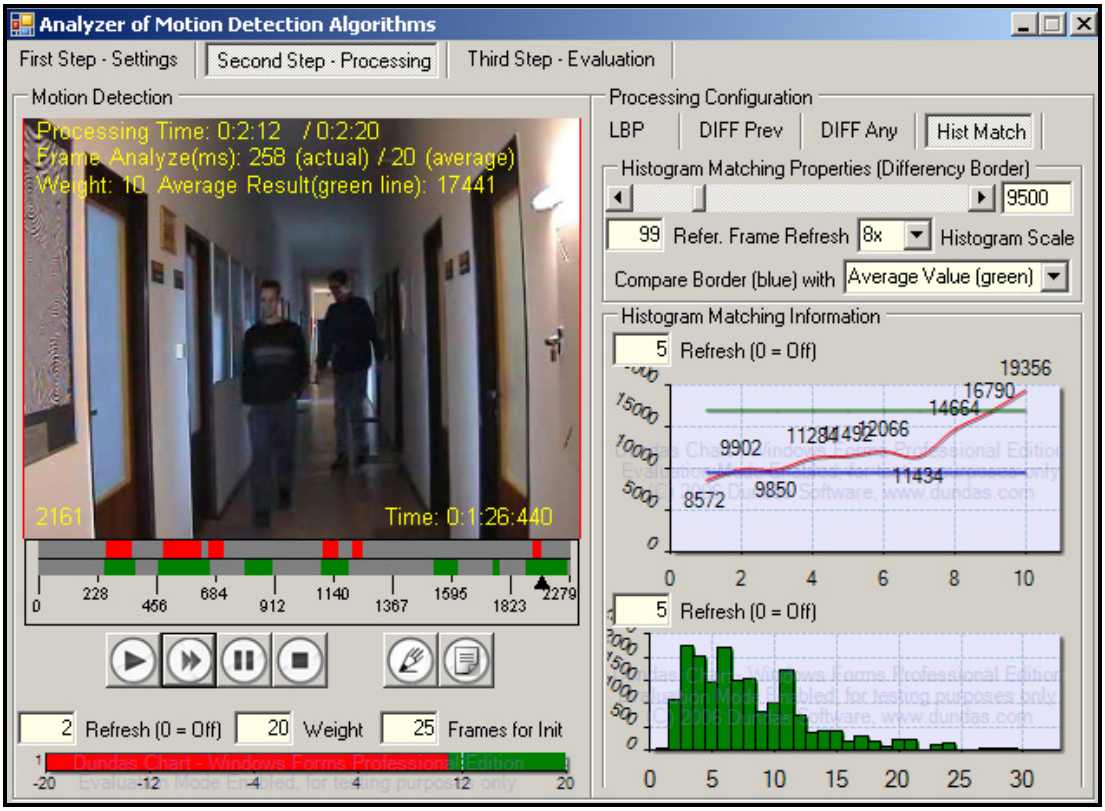
Síť bodů pro kterou by se v tomto případě optický tok počítal, by byla velice řídká, čímž by se jen na pár pixelech porovnávalo, jestli se nějak významně pohnuly z místa a až by se pohnuly, tak by bylo jisté, že se v obraze něco děje. Potom by až mohla přejít do výpočetně náročnějšího stavu, kde by se už přímo sledoval další pohyb objektu.

Optický tok není kvůli své velké výpočetní náročnosti vhodný jako samostatný základ pro detektor pohybu, ale je možné jej i na tuto funkci použít. Další informace a konkrétní užití této metody lze najít třeba ve **[Špan03]**.

4 Návrh testovací aplikace

4.1 Obecná charakteristika a popis fungování

V teoretické části jsem popsal několik metod, které se v praxi používají při implementaci detektorů pohybu. Hlavním cílem celé práce je vyhodnotit, pro které konkrétní situace jsou tyto algoritmy vhodné a jak se chovají v podmínkách reálného užití. Abych mohl tyto testy provést bylo třeba vytvořit aplikaci, která bude schopna změřit výkonnost a účinnost těchto algoritmů a v konečné fázi umožní srovnat výsledky mezi sebou. Jakým způsobem tato měření v aplikaci probíhají, je detailně diskutováno v podkapitolách **4.1.1** o výkonnosti a **4.1.2** o účinnosti detektoru. V aplikaci jsou pro provádění testů implementovány všechny dosud popsané detekční algoritmy, vyjma metod popsaných v kapitole o dalších možných přístupech **3.4**. Některé z těchto algoritmů nejsou implementovány pouze ve své základní podobě a umožňují tedy názorně sledovat, jaké změny ve vyhodnocování může přinést zahrnutí nějakého speciálního principu do obecné podoby algoritmu.



(Obrázek 18) Ukázka zobrazování a nastavování při práci s testovací aplikací

Pro lepší představu při čtení následujících řádků může posloužit **(Obrázek 18)**, kde je podoba testovací aplikace ve chvíli, kdy vyhodnocuje pohyb ve videu metodou rozdílů mezi histogramy.

Aby práce v aplikaci byla co nejintuitivnější, tak většina statistických údajů je mimo textové podoby zobrazena i formou grafů, které jsou pro lidské chápání mnohem přirozenější, byť kvůli omezenému prostoru na obrazovce dochází zpravidla ke zkreslení hodnot. K vizualizaci grafů jsem použil komponentu **Dundas Charts** od firmy **Dundas Data [REF02]**, která je pro výzkumné a nekomerční použití poskytována zdarma.

Aplikace je strukturována do několika sekcí, lépe řečeno kroků, kterými uživatel postupně prochází od načtení hodnoceného videa, přes jeho zpracování, až po vyhodnocení a srovnání s jinými výsledky. Co se odehrává v jednotlivých krocích je dále popsáno v kapitole **4.1.3** o uživatelské struktuře. Při testování se často provádějí stále stejné kroky s minimálním přenastavením, proto lze na více místech aktuální nastavení uložit a v jiném testu jej zase znovu použít.

Po každém dokončeném měření je možno si výsledky uložit do souboru jako hotový úkol (v aplikaci nazváno **job**). Hlavní výhodou ukládání úkolů je vedle zálohy výsledků měření možnost si je do aplikace při vyhodnocování zpětně načíst (až 7 úkolů naráz) a následně z nich tvořit porovnávací grafy. Na těch je pak pěkně vidět, ve kterých aspektech měla použitá metoda před jinou navrch a kde si naopak nevedla moc dobře.

Mimo samotné měření aplikace demonstruje, co se děje v průběhu rozpoznávání pohybu a jakými fázemi při tom algoritmy procházejí. Celou aplikaci proto provází preview okno spolu s jeho ovládacím panelem, které dohromady plní v každém kroku odlišnou ovládací a zobrazovací funkci. Tam, kde se v průběhu algoritmu převádí a filtruje obraz tak, že vzniká jeho nová podoba, si lze mezi těmito podobami přepínat. Toto zobrazování nemá pouze demonstrativní charakter, ale zejména pomáhá při nastavování parametrů detektoru (ukázka různých náhledů je v kapitole o rozdílovém detektoru **3.2**).

Podrobnějšímu popisu aplikace jsou věnovány následující podkapitoly a příloha s ovládáním, ale její ovládání by mělo být dostatečně intuitivní již samo o sobě, a proto slouží zejména pro ilustraci použitých principů. Navíc tam, kde se může uživatel při obsluze dopustit nepovolených operací, se zobrazí informační okno, ve kterém je vždy popsán další korektní postup.

4.1.1 Způsob měření výkonnosti

Výkonnost algoritmu značí dobu, za jakou je metoda schopna vyhodnotit všechny snímky vstupního videa a označit, jestli v daném snímku došlo k pohybu.

Fungování detektorů se od sebe liší zejména tím, jakou režii a tedy i delší čas spotřebují ještě před započetím své vlastní funkce. Některé metody jsou sami o sobě rychlé, ale potřebují mít obraz předem v nějakém formátu, nebo je pro vylepšení rozlišovací schopnosti lepší na snímek předem aplikovat nějaký filtr (zaostření, rozmazání, detekce hran, apod.). Jiné zase můžou být pomalejší, ale na druhou stranu šetří systémové prostředky a režii před započetím vlastní činnosti. S tím vším při měření musí aplikace počítat, jinak by výsledky takto od sebe odlišných algoritmů nebylo možné porovnávat.

Ve své podstatě se dá rozdělit měření rychlosti na tři části, které vyžadují měření pomocí na sobě nezávislých stopek (budou popsány dále):

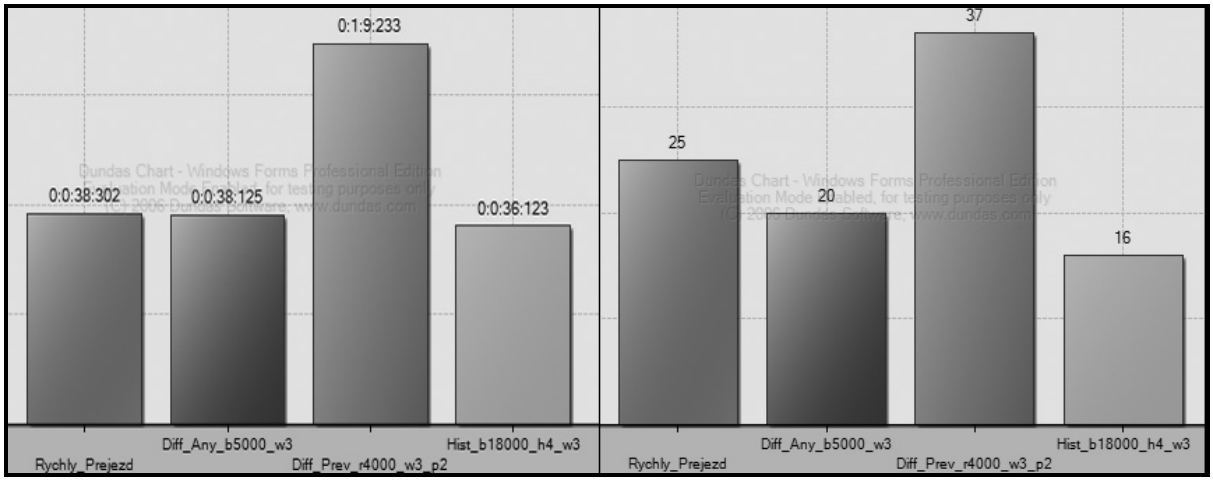
- trvání celkového zpracování od prvního po poslední snímek
- doba potřebná pro označení jednoho snímku i s časem pro aplikaci filtrů, apod.
- čas od vstupu připraveného snímku do detektoru po jeho označení

První je doba za kterou se povedlo zpracovat všechny snímky videa bez ohledu na odlišné vnitřní procesy. Do této doby se započítává při měření veškerá režie spojená s výpočtem snímku, ale i čas, který si vzala samotná aplikace pro svou činnost jako zobrazování videa, statistických údajů, grafů, apod. Pro samotné měření výkonnosti tento čas nemá příliš velkou hodnotu a slouží hlavně pro výpočet přibližné doby dokončení výpočtů a zejména k odhadu přibližné odchylky, která nastane při stejném měření na různých počítačích.

Druhý měřený čas v sobě nese dobu, kterou potřebuje algoritmus od předání barevného snímku videa, až po jeho označení a uložení aktuálních hodnot do historie, která může být potřebná pro výpočet v následujícím snímku. I když se tomu nedá úplně zcela zabránit, tak by v sobě měl nést co nejméně času, který byl spotřebován pro jiné funkce aplikace, než je samotný výpočet metody. V tomto čase je samozřejmě zahrnuta i režie spojená s aplikací obrazových filtrů, převodu na jiný barevný model, apod.

Třetí čas v sobě nese pouze to, co je spojeno s nezbytnými výpočty uvnitř detektoru a jeho stopky se zapínají ve chvíli, kdy snímek již po všech nutných přípravách vstupuje dovnitř a vypínají se až když algoritmus řekne, jestli byl či nebyl nalezen ve snímku pohyb.

Z těchto tří časů lze v konečné podobě odvodit a znázornit téměř jakoukoli časovou informaci. Aplikace je v průběhu zpracování průběžně ukazuje a to od údajů týkajících se celého videa (ty mají smysl zejména po skončení vyhodnocování), přes průměrné hodnoty týkající se dosud zpracovaných snímků, až po historii konkrétních hodnot pro každý snímek (zpravidla je několik posledních průběžně znázorňováno v grafech). Ve fázi vyhodnocení si pak uživatel může nechat zobrazit porovnání s předešlými měřeními formou grafů v podobě, jak jsou vidět na **(Obrázek 19)**.



(Obrázek 19) Ukázka grafů výkonnosti generovaných uvnitř aplikace

4.1.2 Způsob měření účinnosti (resp. úspěšnosti)

Účinnost algoritmů koresponduje s jejich schopností správně označit snímek videa.

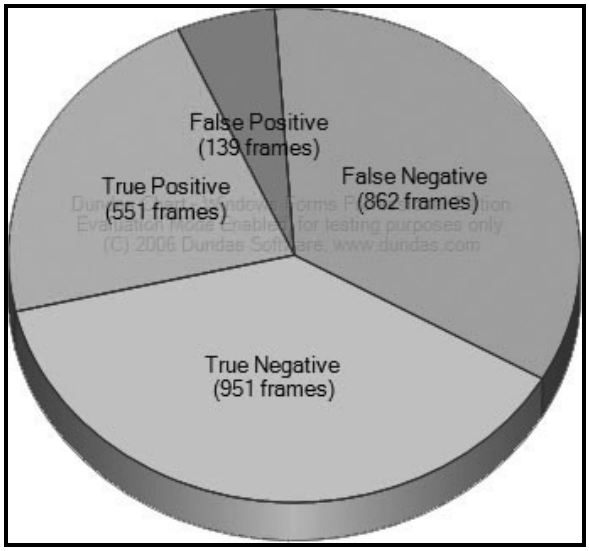
Značkování probíhá na základě boolean hodnoty výstupu detekční metody, tedy „ano v tomto snímku se něco hýbe“ nebo „ne vypadá podobně jako referenční“. Při značkování se vůbec neřeší označení místa pohybujícího se tělesa. V průběhu zpracování videa tedy detektor každému snímku přiřadí atribut pohybu, který se nejen vnitřně ukládá a používá při konečném vyhodnocení účinnosti, ale je uživateli formou barevného označení ovládacího jezdce prezentován po celou dobu značkování. Díky tomu nemusí čekat na vyhodnocení celého videa, aby následně zjistil, že detektor je špatně nastavený a může kdykoli přerušit činnost, přenastavit hodnoty nebo vůbec přepnout na jiný detektor. Jakým způsobem probíhá zjednodušená vizualizace výsledků detektoru je vidět na **(Obrázek 18)**. Při

pozitivním výsledku na pohyb dojde k červenému označení okraje aktuálního snímku v preview okně (jinak žádné okraje nemá) a horní část jezdce pod obrázkem červeně vyznačuje intervaly, ve kterých byl nalezen souvislý pohyb. Je zřejmé, že u delšího videa jsou informace na jezdci značně zkreslené a tedy pouze orientační.

Účinnost algoritmu lze stanovit vždy až při porovnání získaných výsledků s pravdivou informací. Jako nejlepší detektor, se kterým se všechny ostatní ve výsledku porovnávají, slouží sám uživatel. Ten pomocí ovládacích prvků vyznačí ve videu intervaly, kde dochází k pohybu. Aplikace umožňuje kdekoli v rámci videa přesně najít začátek a konec sekvence s pohybem. Tímto jsou vyznačeny intervaly stejně, jako je později vytvoří některý z detektorů. Uživatelem vyznačené části s pohybem jsou následně ve všech krocích zobrazeny zelenou barvou na jezdci, čímž má i v průběhu značkování detektorem orientační přehled, jak si s aktuálním nastavením detekční algoritmus vede. V rámci jednoho videa se bude jistě provádět více testů a proto stačí provést značkování pouze jednou a jeho podobu si uložit.

Ve chvíli, kdy jsou všechny snímky videa jak uživatelem tak detektorem označkovány, nastává vyhodnocení účinnosti metody. Korespondující atributy (je / není ve snímku pohyb) uživatele a detektoru se porovnají a v rámci celého videa se zpracují pomocí tradiční statistické metody na stanovení počtu tzv. „true/false positive“ a „true/false negative“ neboli počtu správně/chybně detekovaných snímků s pohybem a počtu správně/chybně detekovaných snímků bez pohybu.

Ideálně by měl mít detektor ve výsledku co nejvíce správně označených snímků s pohybem i bez pohybu a co nejméně chybně označených. V reálných podmínkách nedokonalého vstupního signálu a omezených možnostech detektorů však musíme dělat kompromisy podle situace. Proto tam, kde je kritické rozpoznat i sebemenší pohyb zvolíme nižší hranice šumu a spokojíme se s velkým množstvím chybně pohybem označených snímků, ale máme víceméně jistotu, že nám i méně výrazný pohybující se objekt takzvaně „neuteče“ a u aplikací, kde je zřejmé, že pohybující se objekt bude ve scéně dominantní, zase právě naopak. Aplikace nám pak ukáže výsledek podobný tomu na **(Obrázek 20)**, který v případě dobře zvoleného a nastaveného detektoru bude odpovídat jednomu z uvedených kompromisů a bude tak pro dané podmínky, alespoň co se týče účinnosti, vhodný.



(Obrázek 20) Ukázka grafu vyhodnocení účinnosti detektoru

4.1.3 Uživatelská struktura aplikace

Aplikace je strukturována do třech logických celků, kterými uživatel od začátku práce do konečného vyhodnocení prochází a jsou jimi:

- část označení intervalů snímků s pohybem ve videu uživatelem
- část označení intervalů snímků s pohybem ve videu detekčním algoritmem
- srovnání označených snímků a vyhotovení výsledků v textové i grafické podobě

Po spuštění programu se uživatel nachází v první části, kde je jeho úkolem zvolit testovací videosoubor a provést vlastnoruční označkování videa. K tomu má k dispozici rozhranní podobné standardním přehrávačům. Video lze nejen spustit, ale libovolně procházet či krokovat po snímcích. Jak kvalitní budou celkové výsledky v této fázi ovlivňuje pečlivost uživatele, který pomocí ovládacích prvků nastavuje ukazatele na začátek a konec pohybu v obraze. Tímto způsobem vytvořené intervaly jsou vizualizovány přímo v oblasti jezdce zelenou čarou na přibližném místě vyznačených hodnot. Značkování lze uložit do souboru a nebo z něj načíst. Po úspěšném vytvoření referenčních intervalů se přechází k dalšímu kroku.

Druhý krok je celý věnován zpracování snímků pomocí implementovaných detekčních metod. Pracovní plocha se v této fázi rozděluje na levou a pravou stranu. V levé části je opět umístěno okno náhledu pro zobrazení aktuálního snímku nebo jeho podoby při aplikaci některého filtru s jeho ovládáním a grafickým znázorněním průběhu značkování na jezdci nebo grafu polohy značkovacích vah (popsány v kapitole **4.1.4**). V pravé části lze vybírat a nastavovat parametry detektorů.

Videem lze „brouzdat“ podobně jako tomu bylo v předchozím kroku pomocí jezdce, což má význam u některých metod pro stanovení referenčního snímku pozadí. Spuštěním přehrávání se začnou snímky zvolenou metodou značkovat. V průběhu značkování lze kdykoli zpracování přerušit nebo pozastavit a změnit parametry nebo samotný detektor. Pod nastavováním parametrů detektorů se pomocí grafů zobrazují užitečné informace, díky kterým se lépe ladí co nejvhodnější nastavení pro dané video. Jakmile jsou označeny všechny snímky videa, tak se objeví popup okno, které nabídne vstup do poslední fáze a nebo vynulování dosavadního měření a vrácení se na začátek kroku.

Vstupem do další fáze se uživatel dostává k vyhodnocení a porovnání metody s dříve dokončenými měřeními. Zase je na levé straně náhledové okno, které tentokrát slouží pouze jako přehrávač. Oproti prvnímu kroku jsou zde však již vyznačeny jak intervaly označené uživatelem, tak detektorem. Smyslem je se podívat, na kterých místech se detekce nezdařila a zkusit se zamyslet, proč tomu tak mohlo být a jaká nastavení zvolit, aby příští měření dopadlo lépe.

Stěžejní je pravá část, kde se v prvé řadě dají získané výsledky uložit do souboru a zejména je zde možno načíst dříve uložené výsledky (říkejme jim úkoly) k jejich porovnání. Porovnávat lze až sedm úkolů mezi sebou. Uživatel má na výběr mezi několika grafickými znázorněními, které se věnují jak porovnání v oblasti výkonnosti metody, tak její účinnosti. Další užitečné statistické informace, které souvisí s aktuálně zpracovaným úkolem, se zobrazují také v textové podobě. Stejné informace, které se týkají některého z načtených starších úkolů, si lze nechat zobrazit do informačního popup okna.

Rozdělení aplikace na tyto logické celky by mělo vést k její přehlednosti a intuitivnímu používání, zejména uživatel nenarazí v jednom místě na ovládací prvky, které v té chvíli ještě nemůže používat.

Ovládání aplikace je podrobně popsáno v příloze 1 a 2.

4.1.4 Princip vah pro sepnutí detektoru

Z důvodu častého kolísání metody kolem hranice, kdy je či není detekován pohyb, je v aplikaci při značkování zvolen přístup, kdy kladná odezva detektoru na pohyb namísto označení snímku pouze posílí jednu ze stran smyšlených vah. K přepnutí režimu značkování mezi scénou s pohybem a bez něj dochází až ve chvíli, kdy hodnota váhy překročí z jedné strany střed na stranu druhou. Každý snímek, který je detektorem označen stejně jako byly předchozí, tímto způsobem posiluje jednu ze stran vah. Proto při náhodném zakolísání v místě, kde se delší dobu značkuje v jednom z režimů nemůže dojít k převážení za střed a tím ve výsledku dostáváme jen souvisle označené intervaly.

V aplikaci jsou váhy vizualizovány grafem, tak jak jej ukazuje **(Obrázek 21)**. S každým snímkem, který detektor označí jako s pohybem narůstá červená oblast směrem doprava (resp. klesá doleva). Pokud ze kterékoli strany přejde přes nulu změní se režim značkování.

(Obrázek 21) Screenshot grafu a ovládání vah v aplikaci

Musíme myslet na to, že v místě, kde se má režim změnit, nastává relativně dlouhá prodleva, než podle snímků s opačným atributem, dojde k posílení druhé strany až po „překlopení“ za střed. Proto má váha volitelnou hodnotu hranice, za kterou už nedochází k posilování té či oné strany. Pokud je tato hodnota vysoká, tak je prodleva mezi přepínáním režimů velká, ale eliminuje chybu vlivem občasného zakolísání detektoru a intervaly jsou mnohem souvislejší. Naopak, pokud je malá, tak je možné pružně reagovat i na rychle se pohybující objekty, které setrvají v záběru pouze několik málo snímků.

S tímto principem na paměti se dá snadno přizpůsobit značkování konkrétním podmínkám a jak bude demonstrováno v části testování dosáhnout mnohem lepších výsledků, než bez použití tohoto systému vah.

4.2 Implementace

Testovací rozhranní i samotné detektory jsou vytvořeny v prostředí **MS Visual Studio .NET 2005** v jazyce **C#**. Spustit se dá na všech systémech, které mají nainstalovánu platformu **.NET Framework** verze dva a vyšší.

Struktura tříd programu je znázorněna na **(Obrázek 22)** a to způsobem, kde třídy zobrazeny ve sloupci (pod sebou na stejné úrovni) tvoří logické celky a v rámci své funkce volají metody těch, co jsou s odsazením ilustrovány pod nimi. Význam jednotlivých celků bude ve stručnosti popsán v následujících kapitolách. Bylo by zbytečné se zdlouhavě věnovat jednotlivým metodám a jejich konkrétní implementaci, proto se budu snažit pouze o popis obecných principů. Zdrojové kódy jsou v kompletní podobě přístupné v elektronické příloze a proto nic nebrání jejich důkladnějšímu prozkoumání. Jednotlivé metody jsou bohatě komentovány a i pro programátora s minimální znalostí

C# by neměl být problém se v nich zorientovat. Všechny komentáře jsou psány v angličtině, aby byla práce přístupná pro co největší okruh zájemců.

4.2.1 Obecné principy tříd a jejich spojitosti

Nejvýše položenou třídou, která v sobě zároveň nese celou podobu hlavního okna aplikace je **MainWindow**. Všechny metody jsou podle jejich příslušnosti do jednotlivých kroků seskupeny do samostatných pojmenovaných regionů, které v **.NET** lze skrýt či zobrazit, čímž lze i rozsáhlý kód udržet přehledný.

Vzhledem k obrovskému množství událostí, které se dějí při interakci s ovládacími prvky nebo překreslování okna, jsou i krátké algoritmy přesunuty do samostatné třídy **AppMethods** a pro přehlednost volány jako metody z ní. V této třídě lze nalézt zejména metody pro značkování a nastavování atributů snímků, načítání a ukládáni různých nastavení, vytváření / aktualizaci informačních grafů (pouze ty, co provázejí druhý krok značkování detektorem), nebo porovnávací metody pro finální vyhodnocení.

Pro snadné sdílení společných proměnných mezi třídami se veškeré globální proměnné nacházejí ve třídě **Settings**, ve které se pomocí virtuálních metod **get** a **set** čtou nebo zapisují. Na tomto místě se vyskytují i všechny komentáře k dotyčným proměnným a také se zde inicializují jejich počáteční hodnoty. Instance této třídy se vytváří pouze v **MainWindow** a do dalších tříd se posílá jen její reference. Tímto přístupem se dá modifikovat konkrétní proměnná i v rámci rozdílných vláken a nebo na jednom místě držet proměnné, které zabezpečí čekání jednoho vlákna na dokončení činnosti druhého.

Ve chvíli, kdy uživatel vyvolá kontextovou nabídku pro otevření videa, se založí instance třídy **VideoStream**, která obsahuje metody pro získání dalšího nebo předchozího snímku videosouboru. Navíc u každého nového videa automaticky načte důležité informace jako délka, rozlišení, počet snímků za sekundu, název, apod., které přímo ukládá do globálních proměnných v **Settings**. Samotnou práci s videem zabezpečuje třída **AVIReader**, která v sobě nese metody na ukládání obrazových dat videa do bitmapy a získávání informací o videu za pomoci třídy **Win32**, která přímo ovládá **API** funkce ze standardní knihovny Windows **avifil32.dll**. Tato třída je převzata z cvičení k předmětu Multimédia a ačkoliv je na Internetu rozšířená a často používaná, bohužel se nepovedlo dohledat původního autora.

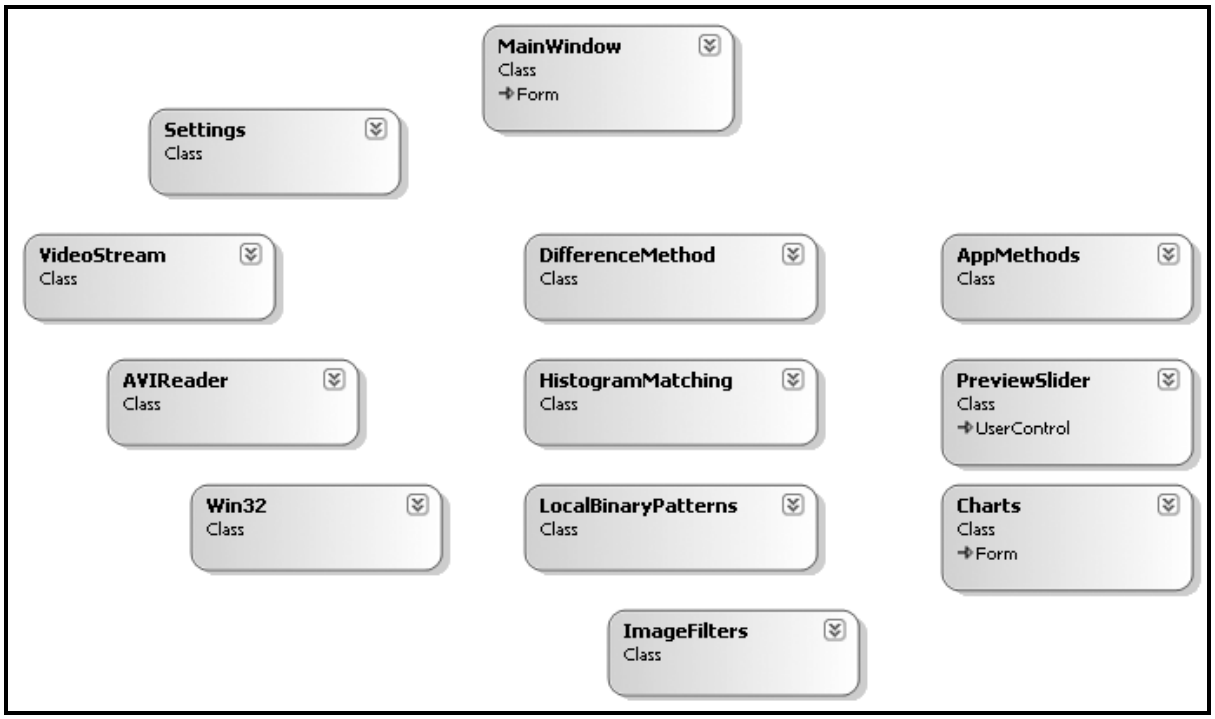
Celou aplikaci pod náhledovým oknem provází posuvník, u kterého bylo třeba, aby nejen hýbal s ukazatelem, ale navíc barevně zobrazoval intervaly pohybu označené uživatelem i detektorem a orientační číselník snímků. Takovou komponentu **.NET** standardně nenabízí, ale umožňuje napsání jakékoli vlastní komponenty, která může zdědit spoustu užitečných vlastností z obecné třídy **UserControl**. Tato komponenta je implementována v třídě **PreviewSlider** a jejímu dalšímu popisu se věnuje kapitola **4.2.3**.

V konstruktoru hlavní třídy **MainWindow** se vytvoří objekty všech metod detektorů. K jejich inicializaci tedy dochází již při spuštění aplikace, a proto v druhém kroku, kde mají detektory provádět značkování videa mezi nimi lze bez problému za chodu přepínat. Jakmile uživatel spustí značkování, tak se spustí komponenta **BackgroundWorker**, která všechen svůj kód provádí na samostatném vláknu a umožňuje tak plynulé, nezávislé překreslování oken aplikace. Tato komponenta v jednom svém cyklu zpracuje zvolenou metodou pouze jeden snímek, který mu dodá výše uvedená instance **VideoStream** a ve chvíli kdy detektor přiřadí snímku atribut pohybu, se vlákno ukončí a vyvolá výjimku. Tu pak odchytne vlákno hlavního okna a následně zobrazí zpracovaný snímek (pokud byl v detektoru změněn, tak v nové podobě), aktualizuje proměnné v **Settings** a zobrazí na příslušných místech statistiky a grafy. Pokud si uživatel nepřál vyhodnotit jen jeden snímek, ale celou sekvenci, tak se po skončení opětovně spustí činnost **BackgroundWorker** a to až do přerušení uživatelem, nebo zpracování posledního snímku, kdy činnost program sám ukončí a nabídne přechod do dalšího kroku.

Ve chvíli, kdy se aplikace přepne do poslední záložky k vyhodnocení (do třetího kroku), se nejdříve ze získaných informací spočítají různé další statistické hodnoty (průměry, celkové časy, apod.), které provádí tomu určené metody z třídy **AppMethods**. Ty se následně uloží do k tomu určeného pole typu „**JobStruct**“ v **Settings**. Tato struktura v sobě zahrnuje položky pro uložení všech užitečných informací získaných v průběhu měření a proměnná je polem proto, aby každý prvek nesl jeden úkol, což již značí, v teoretické části zmiňovanou možnost, otevřít si staré úkoly ze souboru pro jejich následné porovnání.

O zobrazení grafů na základě údajů obsažených ve strukturách pole úkolů se stará třída **Charts**. Když aplikace založí tuto instanci, tak jí pošle číslo, které udává co za graf si uživatel přeje vytvořit a zároveň změní **boolean** hodnotu u všech dostupných úkolů, která bude značit, jestli se má daný úkol v grafech porovnávat a nebo ne. **Charts** už si sama vezme potřebné údaje o požadovaných úkolech ze **Settings**, vytvoří příslušný graf a zobrazí jej jako pozadí v novém okně.

Toto byl jen obecný náhled na fungování hlavních tříd které jsou ilustrovány na **(Obrázek 22)** a popis jejich vzájemné interakce, která v konečné podobě tvoří testovací aplikaci jako celek. V programu jsou ještě další méně významné třídy, které se zabývají zpravidla kontrolou uživatele a zobrazují mu informační či chybová hlášení. Pro konkrétní podoby a principy algoritmů je nejjednodušší prozkoumat přímo zdrojový kód programu.



(Obrázek 22) Struktura tříd v aplikaci a rozdělení na jednotlivé oblasti odkazované z textu

4.2.2 Třídy detektorů pohybu

Implementované detektory v sobě nesou povinný interface, který umožňuje jejich snadné přepínání za chodu programu. Všechny tyto třídy pracují samostatně a pouze na svém vstupu vyžadují vstupní, případně referenční bitmapu. Jejich činnost se vždy vztahuje pouze na vyhodnocení jednoho snímku a pro zpracování nového jsou volány znovu. Součástí každé třídy jsou i stopky, které sepnou na začátku zpracování a ukončí se na konci. Mezičasy se pak ukládají do pole v **Settings**, ze kterého se nejen v třetím kroku počítají průměrné hodnoty, ale časový průběh napříč celým zpracováním si lze nechat zobrazit v **XY** grafu.

Všechny filtry, které se aplikují na snímek ještě před vyhodnocením pohybu, jsou přímo v detektoru volány z třídy **ImageFilters**. Filtry musí být co nejrychlejší, a proto se u nich veškeré zpracování

provádí s přímým přístupem do paměti, kterou **.NET** umožňuje použitím **unsafe** kódu, nad kterým nechává absolutní kontrolu programátorovi a omezí se na ošetřování minima výjimek. Vstupem filtru je vždy bitmapa a výstupem je nová bitmapa, která zachová původní beze změny.

Třída **DifferenceMethod** v sobě nese rozdílovou metodu, která je nejvíc ze všech ostatních závislá na volání filtrů. Sama pak ve chvíli, kdy dostane masku bílých a černých pixelů, pouze spočte jejich počet a porovná s hranicí, na základě které určí, zda-li došlo k pohybu.

Naopak třída **HistogramMatching**, která rozpoznává pohyb pomocí metody srovnávání histogramů, používá pouze filtr pro převod do stupňů šedi. Podstatou jejího fungování je generování histogramů tak, že prochází jednotlivé pixely bitmapy a zvyšuje hodnotu oddílu histogramu na místě, do kterého velikost jasu tohoto pixelu spadá. Pro vyhodnocení od sebe odečítá histogramy aktuálního a průměru předchozích snímků v absolutní hodnotě, množství těchto rozdílů sečte a tradičně porovná s hranicí, nad kterou rozhodne, jestli došlo k pohybu.

Poslední implementovanou metodou je LBP v třídě **LocalBinaryPatterns**. Na každý snímek je aplikován filtr pro převod do stupňů šedi. Algoritmus pak prochází jednotlivé pixely vstupní bitmapy a pro každý pixel zavolá metodu **getPixelLBPValue**, která pro něj vypočítá hodnotu koeficientu **LBP** a podle výsledku inkrementuje oddíl v generovaném **LBP** histogramu. Metoda pro zjištění **LBP** hodnoty pracuje tak, že pro vstupní pixel prochází jeho 3x3 okolí a hodnotu jasu jednotlivých pixelů porovnává s korespondující hodnotou rozvoje $\mathbf{2^n}$ pro **n = <0,7>**. Výsledkem porovnání pixelu s hodnotou rozvoje může být buď číslo **0,** pokud je menší a nebo číslo rozvoje, pokud větší. Průběžně se výsledky pro každý pixel okolí sčítají a výsledné číslo je hodnota **LBP** koeficientu pro daný pixel.

Dále zpracování **LBP** histogramu probíhá stejně jako vyhodnocení ve třídě **HistogramMatching**. Korespondující oddíly mezi aktuálním a průměrem pěti minulých histogramů se v absolutní hodnotě odečtou, poté už jen všechny výsledky stačí sečíst a porovnat s hranicí šumu. Výsledkem porovnání je **boolean** hodnota, která řekne jestli nastal nebo nenastal pohyb.

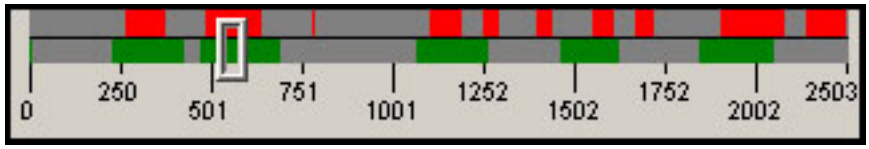
Aplikace je vytvořena s důrazem na možnost snadného začlenění nového detektoru, na kterém by se mohla provést stejná série testů a porovnání jako na stávajících. Nový detektor se do projektu přidá jako další třída, která bude na svém vstupu očekávat aktuální (resp. referenční) bitmapu a jejím výstupem bude **boolean** hodnota, je či není pohyb. Pokud vstupní bitmapu třída nějakým způsobem změní, projeví se tato změna v náhledovém okně, čímž není problém vyznačit pohybující se objekt. Samozřejmostí je možnost použití filtrů implementovaných v **ImageFilters**, nebo doplnění nových.

4.2.3 Komponenta PreviewSlider

Výhoda tvorby samostatných komponent je v jejich dokonalé variabilitě, kdy lze vytvořit prakticky jakýkoli ovládací prvek neomezeného tvaru a schopností, který se pouhým vložením do projektu objeví mezi ostatními komponentami v panelu nástrojů a lze jej libovolněkrát v projektu použít.

Veškerá vizualizace značkování je soustředěna do této komponenty a jako snad jediná třída v projektu, komunikuje se svým okolím skrze načítání a ukládání hodnot do jejich vlastností namísto používání globálních proměnných v **Settings**. Tímto ji lze snadno extrahovat a ve stejné podobě použít kdekoli jinde nebo na ní postavit něco dalšího.

Zobrazované údaje jsou kvůli omezenému prostoru zkreslené a číselník nebo ukazatel se přepočítává na základě počtu pixelů šířky a délky videa. Do **PreviewSlider** lze poslat informace o intervalech značkování, délku videa, aktuální snímek (značí, kde bude ukazatel), pozici levého a pravého ukazatele intervalu, a který proužek s intervaly pohybu se má zobrazovat. Všechny informace jsou vkládány v původních přesných hodnotách a komponenta si je přepočte na zobrazení v pixelech. Naopak lze z komponenty získat přibližnou informaci o pozici ukazatele, která po přesunutí posuvníku určuje hodnotu zobrazeného snímku. Při hýbání ukazatelem se vyvolává výjimka **ValueChanged**, na kterou pak reaguje metoda hlavního okna a mění zobrazovaný snímek.



(Obrázek 23) Komponenta PreviewSlider

5 Testování detektorů pohybu

V kapitolách o detekčních algoritmech jsem psal o různých kladech a záporech, které upřednostňují nebo naopak vylučují použití detektoru v té či oné situaci. V této kapitole vyzkouším na video sekvenci z určitého typového prostředí chování implementovaných algoritmů. Zaměřím se na porovnání podle časového trvání výpočtu a správnosti vyhodnocení. Hranici, nad kterou je detekován pohyb budu u každého testu nastavovat tak, aby bylo co nejvíce snímků s pohybem označeno stejně i detektorem i na úkor detekce pohybu tam, kde žádný není.

Každá podkapitola se bude věnovat jedné typové situaci a poskytne nejen přehledovou tabulku, ale i slovní komentář k výsledkům měření. Výstupem mé práce je zejména samotná testovací aplikace, která umožňuje i vkládání dalších tříd detektorů a testování na jakémkoli videu. Proto si nekladu za cíl udělat co největší množství podrobných testů, ale chtěl bych motivovat čtenáře k vlastnímu testování a ověření teoretických poznatků z této oblasti, které může uplatnit ve vlastním mnohem sofistikovanějším detektoru.

Výsledky testu jsou po stránce účinnosti jednotlivých algoritmů vždy shrnuty v tabulce. Řádky reprezentují jednotlivé detektory a jsou označeny zkratkou stejně jako v testovací aplikaci:

- **LBP** pro metodu Local Binary Patterns
- **Diff Prev** pro rozdílovou metodu
- **Diff Any** je také rozdílová metoda, ale referenční snímek uživatel označí sám, případně se může jednou za čas automaticky obnovit
- **Hist** označuje metodu porovnání histogramů

Občas je potřeba dále upřesnit speciální nastavení detektoru. K tomu používám v tabulkách následující značky (čísla se mohou lišit):

- **8x** u porovnání histogramů znamená osmi násobné zmenšení počtu oddílů histogramu
- **r99** prezentuje nastavení obnovování referenčního obrazu každých 99 snímků
- **p5** u rozdílové metody říká, že jako referenční snímek se vždy bral pátý před aktuálním
- **Act / Aver** značí, jestli k vyhodnocení pohybu podle uživatelem stanovené hranice se použila aktuálně vypočtená hodnota **Act** a nebo průměr hodnot **Aver**
- **w3** jedná se o hodnotu nastavení hranice váhy, zde je hranice **<-3 ; +3>**, ke změně značkovacího režimu dochází vždy při překročení hodnoty **0**

- **t13** značí hranici pro prahovací (**threshold**) filtr, v tomto případě bude každý pixel s hodnotou jasu nad **13** bílý a pod ní černý

V druhé příloze jsou všechny použité parametry detektorů pro testované video sekvence přehledně sepsány v **(Tabulka 6)**. Použitá typová videa jsou dostupná v elektronické příloze.

5.1 Objekt se od kamery vzdaluje / přibližuje

Tato situace je v reálných nasazeních velice běžná. V záběru se objeví objekt, který v první chvíli dominuje popředí a jen těžko jej lze zaměnit se šumem. Postupně se vzdaluje od kamery (resp. přibližuje ke kameře) a tím se vzhledem k perspektivě zmenšuje (resp. zvětšuje). Problémem takového objektu je, že se v záběru může pohybovat tak dlouho, že jej některé detektory mohou postupem času zprůměrovat s pozadím a dále na něj nereagovat, nebo se od kamery natolik vzdálí, že se změna, kterou působí v obraze, může dostat pod hranici, kdy ho detektor bude považovat za šum.

(Obrázek 24) Ukázka z video sekvence „Chodba.avi“

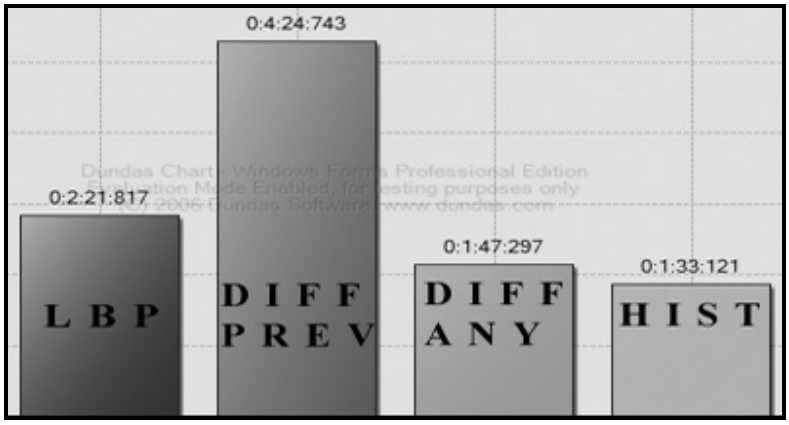
Tento test pracuje s videem chodby, kterou z jedné strany snímá kamera a napříč ní se pohybují postavy, které v určité chvíli vystoupí z obrazu do postranního pokoje **(Obrázek 24)**. Video se jmenuje **Chodba** a má délku 91 sekund při **25fps**.

	True Positive	True Negative	False Positive	False Negative
LBP (Act)	28.6%	51.4%	10.7%	9.3%
LBP (Aver)	31.4% J	51.4% J	8%	9.2%
Diff Prev p5 (Act)	34.1% J	58.9% J	5.3% J	1.7% J
Diff Prev p5 (Aver)	34.1% J	58.9% J	5.3% J	1.7% J
Diff Any (Act)	31.4 %	43.2%	8%	17.4%
Diff Any (Aver)	31.2%	41.5%	8.3%	19%
Hist 1x r0 (Act)	29.7% L	20.7% L	9.7% L	39.9% L
Hist 1x r0 (Aver)	28% L	22.1% L	11.4% L	38.5% L
Hist 8x r99 (Act)	21.7%	54.9%	17.7%	5.7%
Hist 8x r99 (Aver)	21.7%	54.9%	17.7%	5.7%

(Tabulka 1) Měření účinnosti (resp. úspěšnosti) detektorů u videa „Chodba“

Při tomto testu se ukázalo, jak dobře si detektory poradí s objektem, který výrazně mění svojí velikost, čímž se mění množství zjištěného rozdílu mezi snímky. Procentuální vyhodnocení účinnosti detekčních metod jsou v **(Tabulka 1)** a podle ní lze vydedukovat následující skutečnosti.

Nejúspěšnější metodou, co se týče množství správně rozlišených snímku pohybu i bez něj, byla rozdílová metoda (**Diff Prev**), která provádí odečítání s vždy o několik snímků starším obrázkem. Tato modifikace rozdílové metody je ve všech aspektech lepší než standardní přístup, který má předem vybrán referenční snímek (**Diff Any**). Naopak nejhůře dopadlo porovnávání na základě histogramů (**Hist**) a ani její modifikace na snížení počtu oddílů histogramu nebo periodická obnova referenčního snímku jí příliš nepomohla. Z toho by se dalo usuzovat, že tvorba histogramu pro celý snímek je všeobecně špatným přístupem. Čím hůře histogramové metody dopadly, tím zajímavější je fakt, že Local Binary Patterns se téměř ve všech aspektech drží ve výsledcích těsně pod rozdílovou metodou. Dokonce v **True Positive** neboli správně označených snímcích s pohybem byla lepší. Přitom histogram **LBP** koeficientů, který tato metoda generuje, se porovnává stejně jako to dělá metoda označená **Hist**. Navíc jak ukazuje **(Obrázek 25)**, na kterém je délka trvání výpočtu jednotlivých detektorů, tak **Diff Prev** video vyhodnocovala více jak dvakrát déle než **LBP**.

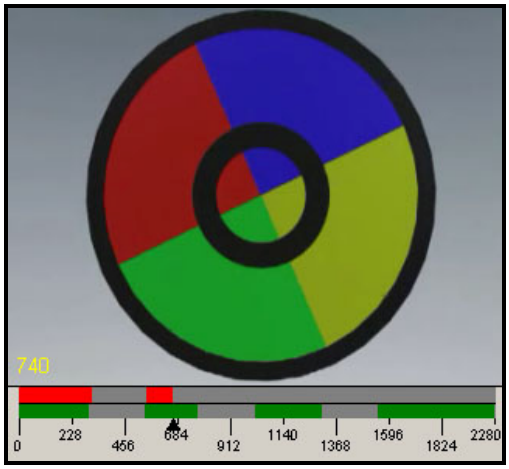


(Obrázek 25) Porovnání časové náročnosti detektorů u videa „Chodba“

V tomto testu značkování pomocí Local Binary Patterns poskytuje nejlepší poměr mezi rychlostí zpracování videa a úspěšností správného označení jednotlivých snímků. Navíc, oproti rozdílové metodě, vyžaduje **LBP** od uživatele pouze zadání hranice, nad kterou se značkuje atributem pohyb.

5.2 Objekt nemění charakteristiku obrazu

Kamera zabírá místo, kde dochází k pohybu, ale objekty neopouští zorné pole kamery. V takovém případě, pokud je zachována světelná charakteristika scény, nedochází k výrazné změně v jejím histogramu. Pixely s různou světelností se pouze přemisťují na jiné místo, ale stále jsou v obraze. Předpokladem je, že histogramové metody nebudou mít prakticky žádnou šanci pohyb rozpoznat, naproti tomu rozdílové by měli fungovat bez problémů. Zajímavé bude sledovat jak si v tomto prostředí poradí **LBP**.



(Obrázek 26) Ukázka z video sekvence „Terc.avi“

Test pracuje s videem počítačové animace, ve které se otáčí na stejném místě barevný terč, který se občas na nějakou dobu zastaví **(Obrázek 26)**. Dále se ve videu náhle konstantně zvyšuje a snižuje světlost, což by mělo ukázat výhody **LBP** oproti rozdílové metodě bez aktualizace referenčního snímku. Aby se metody museli vypořádat i se šumem, je na celou animaci v průběhu aplikován animovaný Gaussovský šum. Video má stejnou délku, rozměry i počet snímků za sekundu jako **Chodba**, aby bylo možné mezi nimi srovnat časovou náročnost.

	True Positive	True Negative	False Positive	False Negative
LBP	63% J	33.8% J	1.2% J	2% J
Diff Prev	63%	34.2%	1.3%	1.5%
Diff Any r0	64.2% L	0% L	0% L	35.8% L
Diff Any r30	61.9%	33.7%	2.3%	2.1%
Hist 2x r0	64.2%	20.1%	0%	15.7%

(Tabulka 2) Měření účinnosti (resp. úspěšnosti) detektorů u videa „Terc“

Z výsledků měření v **(Tabulka 2)** je patrné, že nejlépe z hlediska úspěšnosti si opět vedla rozdílová metoda v modifikaci, kdy jejím referenčním snímkem je několikátý předchozí. Avšak kvůli velké časové náročnosti patrné z **(Obrázek 27)** bychom ji jen těžko mohli označit za nejlepší volbu. Naproti tomu i přesto, že se popředí obrazu výrazně měnilo a rozdílová metoda by obecně měla dávat dobré výsledky, tak verze, kdy se porovnává pouze s jedním zvoleným snímkem, byla vůbec nejhorší. Je to způsobeno absencí stabilního pozadí, se kterým by se dalo porovnávat. Navíc video v průběhu mění osvětlení scény, na což je tato modifikace obzvláště citlivá. Řešením je aktualizovat snímek, což jak je patrné z tabulky pomohlo, ale musela se vybrat správná frekvence, která se bez znalosti celé video sekvence prakticky nedá odhadnout. Histogramová metoda si kupodivu nevedla tak špatně, jak by se dalo čekat, ale kvůli nepatrným změnám v barevné charakteristice scény mezi pohybem a bez pohybu bylo nutné hranici nastavit velmi přesně, což v reálném nasazení není možné a proto pro podobné situace také není vhodná.

(Obrázek 27) Porovnání časové náročnosti detektorů u videa „Terc“

Vítězem je opět **LBP**, jejíž koeficienty snímky popsaly velice přesně a při porovnání aktuálního histogramu s průměrem předchozích bylo patrné velké rozmezí mezi pohybem a scénou beze změny. Kvůli tomu nebylo nutné hranici nastavovat příliš přesně a díky její vysoké rychlosti by i v tomto případě zastala svou funkci nejlépe.

5.3 Objekt rychle vstoupí a vystoupí ze záběru

V tomto testu se objekt v záběru kamery zdrží jen několik snímků a proto bude kladen důraz na rychlé sepnutí detektoru. Pokud detektor nebude rychle reagovat, nemusí být schopen objekt v takové situaci vůbec zachytit. Spínání detektoru ovlivňuje zejména způsob vyhodnocování pomocí systému vah. Váhy budou v testovací aplikaci nastaveny na nízkou hodnotu, což sice umožňuje algoritmu rychlejší reakci, ale na druhou stranu se budou vlivem šumu častěji označovat místa bez pohybu jako s pohybem.

(Obrázek 28) Ukázka z video sekvence „Rychly_Prejezd.avi“

Video sekvence je z reálného prostředí, kde kamera snímá rychlostní silnici a napříč záběru projíždějí auta **(Obrázek 28)**. Auto se vyskytne v záběru na pouhých 10-20 snímků. Navíc kamera nebyla umístěná na stativu a proto se musí detekční algoritmy vypořádat s lehkým chvěním obrazu. Video se jmenuje **Rychly_Prejezd** a má délku 36 sekund při **25fps**.

	True Positive	True Negative	False Positive	False Negative
LBP w3	31.8% **J**	56.4% **J**	5.3% **J**	6.5% **J**
Diff Prev p2	31.2%	61.2%	5.9%	1.7%
Diff Any w3	32.6% **L**	27.3% **L**	4.4% **L**	35.7% **L**
Hist 4x w3	28.9%	57.9%	8.2%	5%

(Tabulka 3) Měření účinnosti (resp. úspěšnosti) detektorů u videa „Rychly_Prejezd“

Z výsledků **(Tabulka 3)** je patrné, že si většina metod vedla velice dobře. Bylo pouze nutné snížit hranice vah aby detektor spínal rychleji. **Diff Any** (rozdílová metoda s předem určeným pozadím) propadla jen proto, že je video natočeno bez stativu a referenční snímek se od jiných snímků bez aut podstatně lišil. Naopak ostatní metody i přes časté chvění obrazu značkovaly téměř bezchybně. To dokazuje, že pokud je popředí dostatečně dominantní (každé auto zabíralo velkou část obrazu), nezáleží příliš na použité metodě a dokonce zde lze použít i nejzákladnější verzi histogramové metody, která obvykle nedávala u jiných testů dobré výsledky.

Grafy časové závislosti vycházejí v poměru velice podobně, a proto není třeba je dále u testů uvádět.

5.4 Objekt je malý, rychlý a ve směru záběru

V tomto testu jsou podrobeny detektory zatěžkávací zkoušce, která ukáže jejich schopnosti v reálném prostředí s téměř všemi negativními vlivy. Objekty se pohybují ve směru snímání kamery, podobně jako tomu bylo u testu v chodbě. Zde ale půjde o venkovní prostředí, kde pozadí není statické, ale je v něm spousta rušivých prvků jako pohyb stromů ve větru, chvění kamery, šum, apod. Navíc pohybující se objekty jsou relativně malé a ještě se pohybují rychle. Můžeme položit předpoklad, že zde budou mít problémy jak rozdílové, tak histogramové metody, ale LBP o které její autoři tvrdí, že si právě v tomto proměnlivém prostředí vede dobře, by to mohla ukázat.

(Obrázek 29) Ukázka z video sekvence „Silnice_z_Mostu“

Video sekvence v tomto testu zabírá z mostu přes rychlostní silnici auta, která tentokrát nebudou dominantní složkou obrazu. Auta se pohybují ve směru snímání kamery a velice rychle se perspektivně zmenšují **(Obrázek 29)**. V pozadí se pohybují stromy ve větru a navíc kamera není na stativu, takže dochází k lehkému chvění obrazu. Video se jmenuje **Silnice_z_Mostu** a má délku 90 sekund při **25 fps**.

	True Positive	True Negative	False Positive	False Negative
LBP w5	20.5%	58.1%	14%	7.4%
Diff Prev w5	30.5% **J**	62% **J**	4% **J**	3.5% **J**
Diff Any t13	19.8%	63.4%	14.8%	2%
Hist 4x w3	17.3% **L**	64% **L**	17.2% **L**	1.5% **L**

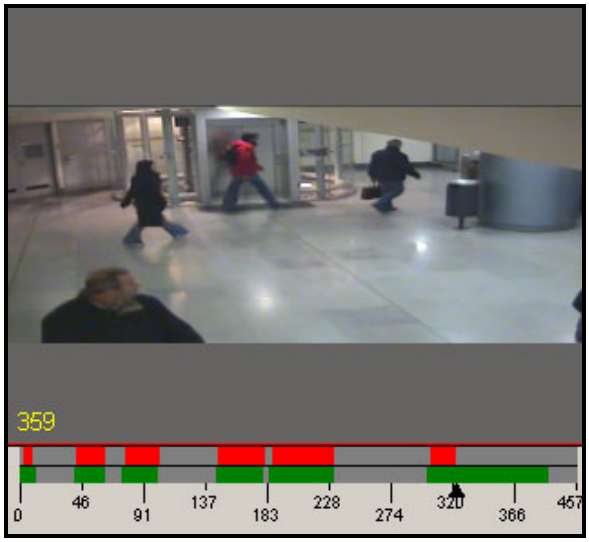
(Tabulka 4) Měření účinnosti (resp. úspěšnosti) detektorů u videa „Silnice_z_Mostu“

Podle výsledků měření **(Tabulka 4)** je patrné, že i v nepříznivých podmínkách reálného exteriéru jsou schopny algoritmy i v obecné podobě docela dobře fungovat. Bezkonkurenčně značkovala scénu časově nejnáročnější rozdílová metoda, u které ani nebylo nutné příliš přesně odhadovat hranici pohybu. Narozdíl od předchozích měření nebyla nejhorší základní podoba rozdílové metody, ale algoritmus porovnávání na základě histogramů. Důvodem byla jen malá změna barevného charakteru scény při vjezdu automobilu. Tato metoda by si vedla lépe pouze v případě většího přiblížení, kdyby se popředí stalo proti pozadí dominantnější. **LBP** i v tomto nadmíru nepříznivém prostředí

vykazovala dobré výsledky, ale hranici pohybu bylo nutné nastavit velmi přesně, a proto by se v reálném nasazení do exteriéru ve své základní podobě pravděpodobně využít nedala.

5.5 Prostředí snímané průmyslovou kamerou

Tento test je typově podobný prvnímu testu s chodbou. Avšak narozdíl od chodby, která byla natočena jako modelový příklad, zde se jedná o reálné prostředí snímané průmyslovou kamerou. V tomto prostředí se objekty pohybují pro změnu napříč zorného pole a nemění tak svou velikost. Dochází k lehkým změnám v osvětlení, ale kvůli dominantnímu postavení popředí s velkými objekty vzhledem k pozadí by toto nemělo mít vliv. Předpokladem je, že žádný z algoritmů nebude mít se značkováním problémy.



(Obrázek 30) Ukázka z video sekvence „Metro“

Video sekvence pochází z projektu **Caretaker [REF03]** vedeném na **FIT** skupinou zpracování řeči. Jsou na ní reálné záběry z průmyslové kamery umístěné v metru **(Obrázek 30)**. Záběrem procházejí různou rychlostí lidé, kteří vždy tvoří dominantní část v obrazu. Jediný problém, se kterým si musí detektory poradit, je lesklá podlaha osvícená zářivkami periodicky měnícími jasový charakter scény. Video se jmenuje **Metro**, má délku 91 sekund při **5fps**.

	True Positive	True Negative	False Positive	False Negative
LBP w3	46.6%	40%	10.3%	3.1%
Diff Prev pw1	53% J	42.5% J	3.9% J	0.6% J
Diff Any w3	52%	39.4%	4.8%	3.8%
Hist 4x r0 w4	54.5% L	19.5% L	2.4% L	23.6% L
Hist 4x r9 w4	53.4%	34.8%	3.5%	8.3%

(Tabulka 5) Měření účinnosti (resp. úspěšnosti) detektorů u videa „Metro“

Výsledky **(Tabulka 5)** korespondují s předpokladem, že tato scéna se statickým pozadím a velkými objekty nebude detektorům činit velké problémy. Výjimkou byla základní podoba histogramové metody bez obnovy referenčního snímku. Tato fungovala na začátku dobře, ale vlivem změny barevné charakteristiky scény v průběhu zpracování se brzy dostala do stavu, kdy i místa bez pohybu byla od snímku natolik odlišná, že v nich pohyb detekovala. Toto se lehce vyřešilo periodickým obnovováním referenčního snímku, který už značkoval podobně dobře jako ostatní metody.

6 Závěr

V této práci jsem si kladl za cíl nejen seznámit čtenáře s různými principy řešení problematiky detekce pohybu v obraze, ale hlavně mu pomoci při rozhodování, jaký přístup by měl ve své aplikaci zvolit, aby dosáhl co nejlepšího poměru mezi účinností a výkonností detektoru.

V teoretické části jsem nabídl mnohé rozdílné přístupy, které lze dále rozvíjet a zdokonalovat. Na základě implementace těchto metod v testovací aplikaci jsem u každého detektoru mohl zmínit, jakým směrem by se měla ubírat jejich vylepšení, aby se rozdíly mezi schopností správně rozpoznávat pohyb a časové náročnosti vyvážily a nebo se staly v určité oblasti mnohem robustnější. Implementaci těchto a jim podobných „zlepšováků“ jsem se věnoval jen okrajově, ale nabízím je k dalšímu řešení v budoucích projektech.

Praktickým výstupem této práce je aplikace pro testování a vzájemné porovnání detekčních metod. Program plní funkci testovací, kdy lze jednoduše vyzkoušet chování detekčních algoritmů na různých prostředích v záznamu, ale i funkci demonstrativní, kdy je formou grafů a náhledů na ještě rozpracované snímky názorně vidět, jakými kroky detektor (resp. snímek) prochází od přečtení snímku, až po jeho konečné vyhodnocení. Do aplikace jsem implementoval čtyři detektory, nad kterými lze provést měření a získat výsledky v číselné i grafické podobě. Navíc umožňuje snadno vložit novou třídu s detekční metodou a provést nad ní stejná měření, čímž se může stát užitečnou pomůckou při tvorbě a optimalizaci detektorů.

V neposlední řadě výsledky provedených měření ukázaly, že detektor založený na metodě Local Binary Patterns, vyvinuté relativně nedávno na univerzitě v Oulu, se chová velmi dobře prakticky v každém testovaném prostředí. Jedná se tedy o velice rychlý univerzální algoritmus, na který by se při návrhu systémů pro detekci či sledování objektů mělo myslet a bezpochyby najde v této oblasti ještě silné uplatnění. Mnohem dokonalejší detektor na základě této metody, který by mohl kromě detekce pohybu i objekt sledovat nebo rozpoznat, by bylo zajímavé rozpracovat v dalších projektech.

Celkově získávání různých informací z rastrové reprezentace obrazu je téma, které v dnešní době hraje významnou roli a každý dobrý nápad se téměř okamžitě dočká praktického užití. Této oblasti by se i nadále měla věnovat náležitá pozornost, ať už formou výuky nebo návrhem dalších prací.

Literatura

[Žára04] ŽÁRA, J., BENEŠ, B., FELKEL, P.: Moderní počítačová grafika. ComputerPress, 2004, 448 s., ISBN: 80-7226-049-9.

[Wiki06] GRT: Princip výpočtu dvourozměrné diskrétní konvoluce. http://cs.wikipedia.org/wiki/Konvoluce, (19.7.2006).

[Oulu06] 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).

[MVG05] PIETIKÄINEN, M,: Image analysis with local binary patterns. http://www.ee.oulu.fi/mvg/publications/show_pdf.php?ID=638, (2005).

[Oulu04] HEIKKILÄ, M., PIETIKÄINEN, M., HEIKKILÄ, J.: A texture-based method for detecting moving objects. http://www.ee.oulu.fi/mvg/publications/show_pdf.php?ID=533, (2004).

[Oulu96] OJALA, T.,PIETIKÄINEN, M., HARWOOD, D.: A comparative study of texture measures with classification based on feature distributions. http://www.ee.oulu.fi/research/imag/texture/publications/show_pdf.php?ID=336, (1996).

[FIT06] ŠPANĚL, M., BERAN, V.: Obrazové segmentační techniky. http://www.fit.vutbr.cz/~spanel/segmentace/.en.iso-8859-2, (19.1.2006).

[Kirill06] KIRILLOV, A.: Motion Detection Algorithms. http://www.codeproject.com/cs/media/Motion_Detection.asp, (11.4.2006).

[Mason01] MASON, M., DURIC, Z.: Using histograms to detect and track objectss in color video. In 30th Applied Imagery Pattern Recognition Workshop, (2001).

[Špan03] ŠPANĚL, M.: Rozpoznávání gest ve video sekvencích. http://www.fit.vutbr.cz/~spanel/dp/spanel_dp_2003.pdf, (2003).

[REF01] MACHINE VISION GROUP: Texture-based computer vision. http://www.ee.oulu.fi/research/imag/texture/texture.php, (2007).

[REF02] DUNDAS DATA: Dundas Chart for .NET. http://www.dundas.com/Products/Chart/NET/index.aspx, (2007).

[REF03] SPEECH PROCESSING GROUP: Caretaker. http://www.fit.vutbr.cz/research/view_project.php.cs?id=342&notitle=1&format=0&shortname=0, (2007).

Seznam příloh

Příloha 1: Popis ovládacích prvků aplikace

Ovládací prvky aplikace se ve všech krocích logicky rozdělují podle své funkce na dvě skupiny, které jsou umístěny na levou a pravou stranu. Na levé straně je především náhled na aktuální snímek videa a ovládací prvky k jeho prohlížení či zpracování. Pravá strana dává prostor grafům, ovládacím prvkům pro nastavování parametrů a ikonám pro vyvolání kontextových nabídek (ukládání, nahrávání) nebo pro vyvolání popup oken.

Ačkoli jsou jednotlivé kroky součástí samostatné záložky, není možné jimi procházet kliknutím na záložku, ale je nutné nejdříve splnit podmínky pro přechod do dalšího kroku a přejít dále uvedeným způsobem. Program v sobě nese i stručnou nápovědu, která se vyvolává ikonou modrého otazníku v pravé dolní části aplikace.

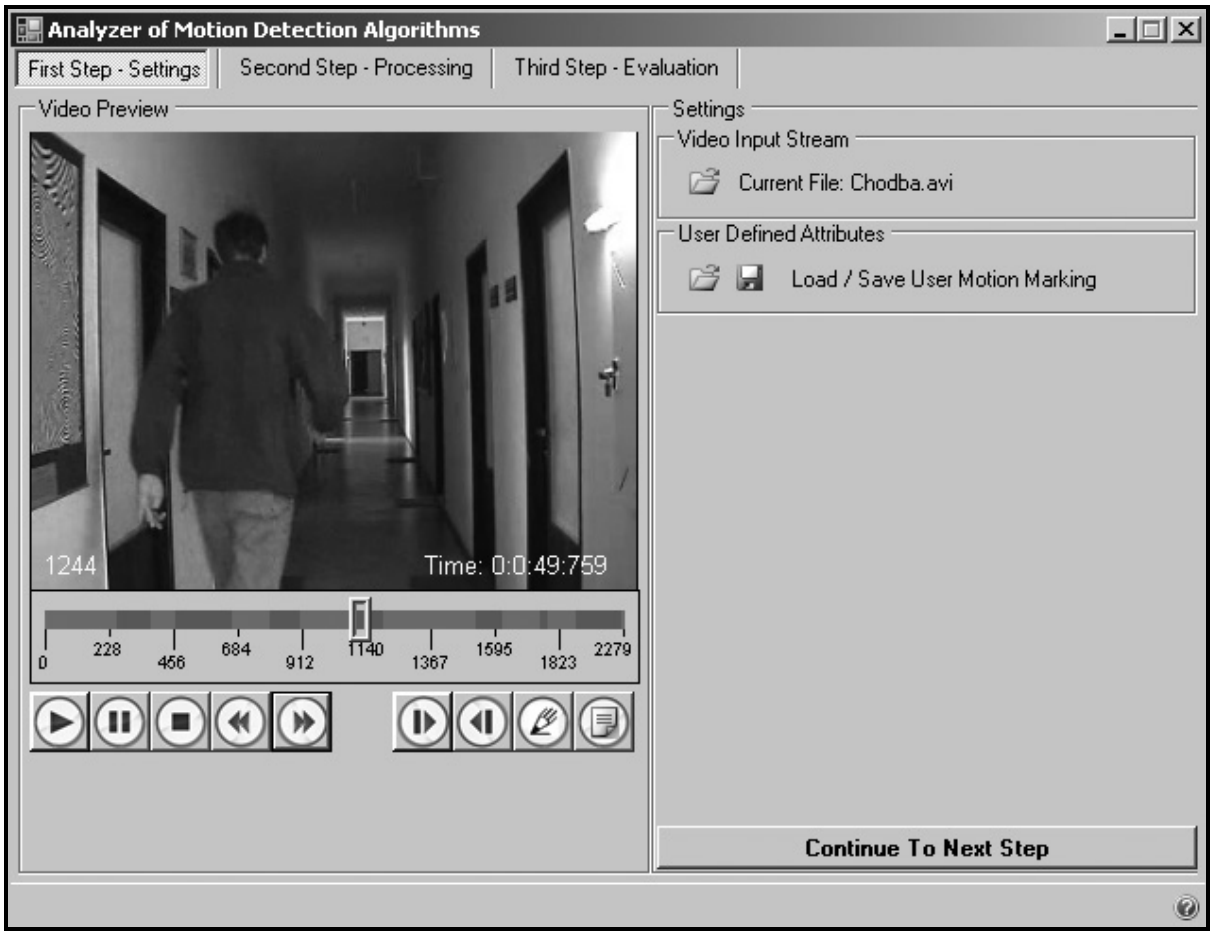
Není-li uvedeno jinak, tak se dále popsané kroky budou pro lepší orientaci vždy vztahovat k uvedenému screenshotu aplikace.

První krok – značkování uživatelem

Nejprve je nutné zvolit video ke zpracování pomocí tradiční kontextové nabídky stiskem ikony vedle **Load Video File**. Pokud zvolíte soubor, ke kterému bude existovat pod stejným názvem a příponou **txt** soubor s již uloženými značkovacími daty, budete dotázáni na jeho automatické otevření. Soubor se značkovacími informacemi lze samostatně nahrát či uložit kontextovými nabídkami pod ikonami vedle **Load / Save User Motion Marking**. Po načtení a označkování videa se do dalšího kroku přechází tlačítkem **Continue To Next Step**.

Na levé straně se po načtení videa zobrazí jeho první snímek. Přímo do obrazu se žlutým písmem zobrazují užitečné informace, v tomto kroku konkrétně číslo snímku.

Pod obrazem snímku je komponenta jezdce, který ukazuje přibližné umístění zobrazeného snímku v rámci videa pomocí spodní orientační stupnice. Navíc zeleně od původní šedé barvy označuje místa, kde je vyznačen pohyb. Prohlížení různých částí videa je umožněno pohybem ukazatele. Dvouklikem do prostoru jezdce se objeví popup okno umožňující vymazat dosud provedené značkování.



(Obrázek 31) Ukázka aplikace v prvním kroku – značkování uživatelem

Pod jezdcem jsou tlačítka obsluhy přehrávání (v tradiční podobě a se stejným významem jako u přehrávačů multimédií) a značkování, jejich význam popiši zleva doprava podle **(Obrázek 31)**:

- Spustit přehrávání.
- Zastavit přehrávání a zůstat na aktuálním snímku.
- Zastavit přehrávání a přesunout na první snímek.
- Posun o jeden snímek vlevo.
- Posun o jeden snímek vpravo.

To byla tlačítka pro přehrávání, a teď pro speciální obsluhu:

- Umístění ukazatele pro levou stranu intervalu pohybu.
- Umístění ukazatele pro pravou stranu intervalu pohybu (jakmile se umístí oba, tak se místo nich zeleně označí interval).
- Přepínací tlačítko mezi režimem označování pohybu (symbol tužky).

- Přepínací tlačítko režimem mazání již označeného pohybu (symbol gumy).
- Zobrazení informací do obrazu (zde jen vypne či zapne číslo snímku).

Druhý krok – značkování detektorem

K popisu tohoto kroku bude dobře sloužit již uvedený **(Obrázek 18)** z kapitoly obecného popisu aplikace. Na levé straně se opět zobrazuje náhled na video a jezdec pod ním je nyní rozdělen na dvě části. V horní se po zahájení detekce budou zobrazovat červenou barvou místa s pohybem a v dolní jsou vyznačeny intervaly vybrané v minulém kroku uživatelem. Nadále je zachována možnost si videem libovolně procházet a děje se tak posunem teď již malého ukazatele (malý, aby nepřekážel v náhledu na vybarvované intervaly) po oblasti číselníku.Ovládací tlačítka pod jezdcem nejen trošku prořídla, ale budou nyní plnit odlišnou funkci. Z leva doprava mají následující funkci:

- Spuštění zvoleného detektoru (zpracovává jeden snímek po druhém tak rychle, jak jej dokáže počítač vyhodnotit).
- Spočtení pouze jednoho následujícího snímku (hodí se pro ladění hodnot parametrů na konkrétním místě).
- Zastavení funkce detektoru na posledním dokončeném snímku.
- Zastavení práce detektoru s přesunem na první snímek plus kompletní zrušení dosud získaných výsledků.
- Zapínání a vypínání statistických informací zobrazovaných přímo do obrazu náhledu.
- Výběr konkrétního referenčního snímku (symbol tužky, zobrazuje se pouze u rozdílové a histogramové metody).

Pod ovládacími tlačítky je zobrazen graf vah a jeho možná nastavení. Stejně jako u všech jiných grafů se v prvním boxu nastavuje frekvence překreslování (nula znamená vypnuto), v druhém hranice vah (číslo znamená plusovou i mínusovou stranu se středem vždy v nule) a nakonec třetí box nastavuje počet snímků, které se provedou pro inicializaci proměnných a historie detektorů (až po uplynutí se spustí samotné značkování od prvního snímku).

Prává strana je tentokrát plná ovládacích prvků a zejména vizuálních zobrazení průběhu zpracování. Každý z nabízených algoritmů má svá odlišná nastavení a proto jsou jednotlivě rozděleny pro zvýšení přehlednosti do samostatných záložek. Zvolení konkrétní záložky (i v průběhu zpracování) zapříčiní, že další snímek po aktuálním, bude již vyhodnocen touto zvolenou metodou (vyjma přepínání záložek ještě před prvním zahájením, kdy se pak zvolená metoda použije už od prvního snímku a ne až na následující). Obsah jednotlivých záložek je kvůli snazšímu popisování zobrazen na **(Obrázek 32)** v pořadí postupně z leva doprava a na další řádek.

Prvky které jsou společné pro všechny detekční algoritmy a najdeme je na každá záložce jsou:

- Horní jezdec, který určuje hranici zjištěných rozdílů mezi snímky a jehož hodnota se dá i vepsat číslem do vedlejšího boxu (značkovací algoritmus porovnává výstupní hodnotu velikosti rozdílu mezi snímky z detektoru s touto nastavenou hranicí a pokud zjistil hodnotu nižší vyhodnotí snímek jako bez pohybu jinak s pohybem).
- Nabídka pro výběr, jestli bude značkovací algoritmus k porovnání se stanovenou hranicí používat aktuální výstupní hodnotu detektoru (červená) a nebo průměrnou hodnotu z desíti posledních výsledků (zelená).
- Graf aktuálního výsledku detektoru a historie devíti posledních hodnot (červenou čárou ukazuje spojnicí aktuální výsledek reprezentující hodnotu množství rozdílu mezi snímky, zelenou vodorovnou čarou průměrnou hodnotu na základě červeně znázorněných výsledků a modrou vodorovnou čarou uživatelem stanovenou hranici nastavenou jezdcem nahoře).
- Box nad každým grafem, který určuje po kolika snímcích se bude graf překreslovat, hodnotou 0 se překreslování zastaví.

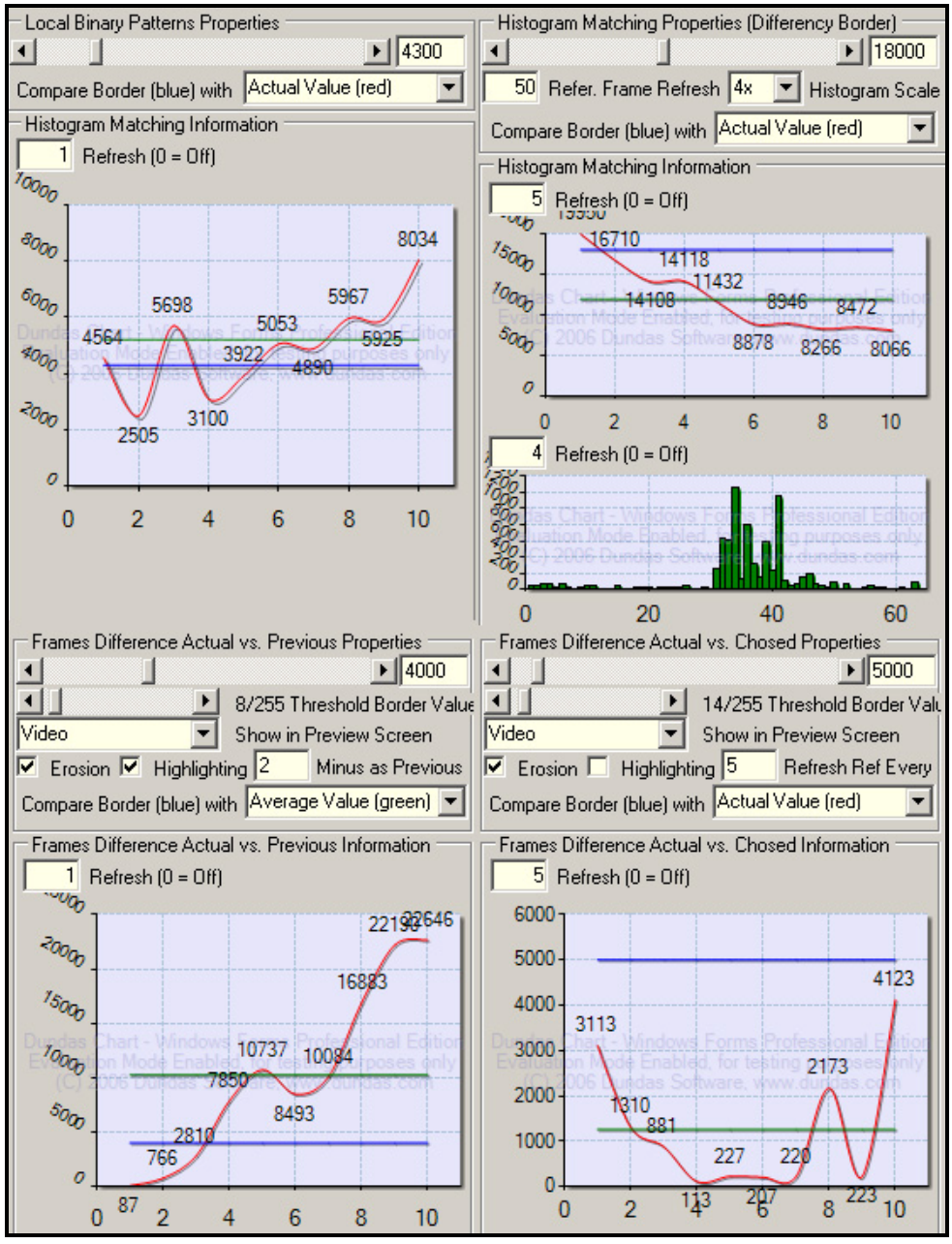
První záložka je věnována asi nejzajímavější metodě Local Binary Patterns blíže popsané v kapitole **3.3**. Tato metoda nepotřebuje krom společných nastavení popsaných výše žádná další.

Druhá záložka aktivuje rozdílový detektor blíže popsaný v kapitole **3.2**, který si jako referenční snímek bere několikátý předchozí. Mimo společné ovládací prvky se zde pomocí jezdce nastavuje hranice pro prahovací filtr. Další je vysouvací nabídka, kterou lze určit podobu v jaké se bude snímek zobrazovat v náhledu (**preview**). Díky možnosti zobrazit si snímek po aplikaci jednotlivých filtrů, lze lehce nastavit správnou hranici prahovacího filtru nebo sledovat hranici šumu. Pod vysouvací nabídkou se na levé straně zatrhne, jestli se má aplikovat na obraz i erosion filtr. Vedle toho je k zatrhnutí možnost zobrazovat pohybující se objekt přímo červenou barvou v obraze (vysouvací nabídka musí být nastavena na „video“) a nakonec na pravé straně box jehož hodnota stanoví kolikátý předchozí snímek se v každém kroku bude brát jako referenční.

Pod třetí záložkou se opět skrývá rozdílový detektor, který ale neporovnává aktuální s předchozím snímkem, ale umožňuje uživateli kdykoli zvolit nový referenční snímek (vždy musí být nějaký zvolen) pouhým nastavením ukazatele na vybraný snímek a stiskem tlačítka se symbolem tužky. Dále je tu box jehož hodnota říká po kolika snímcích dojde k aktualizaci referenčního (standardně má hodnotu nula neboli neobnovovat). S ostatními ovládacími prvky se pracuje stejně jako v druhé záložce.

Poslední záložka je pro detektor založený na porovnávání histogramů blíže popsaný v kapitole **3.1**. Tato metoda stejně jako předchozí vyžaduje zvolení referenčního snímku pomocí tlačítka se

symbolem tužky. Pro aktualizaci referenčního snímku po určité době je tu opět box označený **Refer. Frame Refresh**. Vedle něj se nastavuje poměr histogramu, který při jiné hodnotě než 1x pokrývá jedním oddílem celý interval jasových hodnot pixelů. V této záložce je navíc graf, který zobrazuje histogram hodnot po odečtení histogramu aktuálního a referenčního snímku.



(Obrázek 32) Záložky nastavení jednotlivých detektorů pohybu

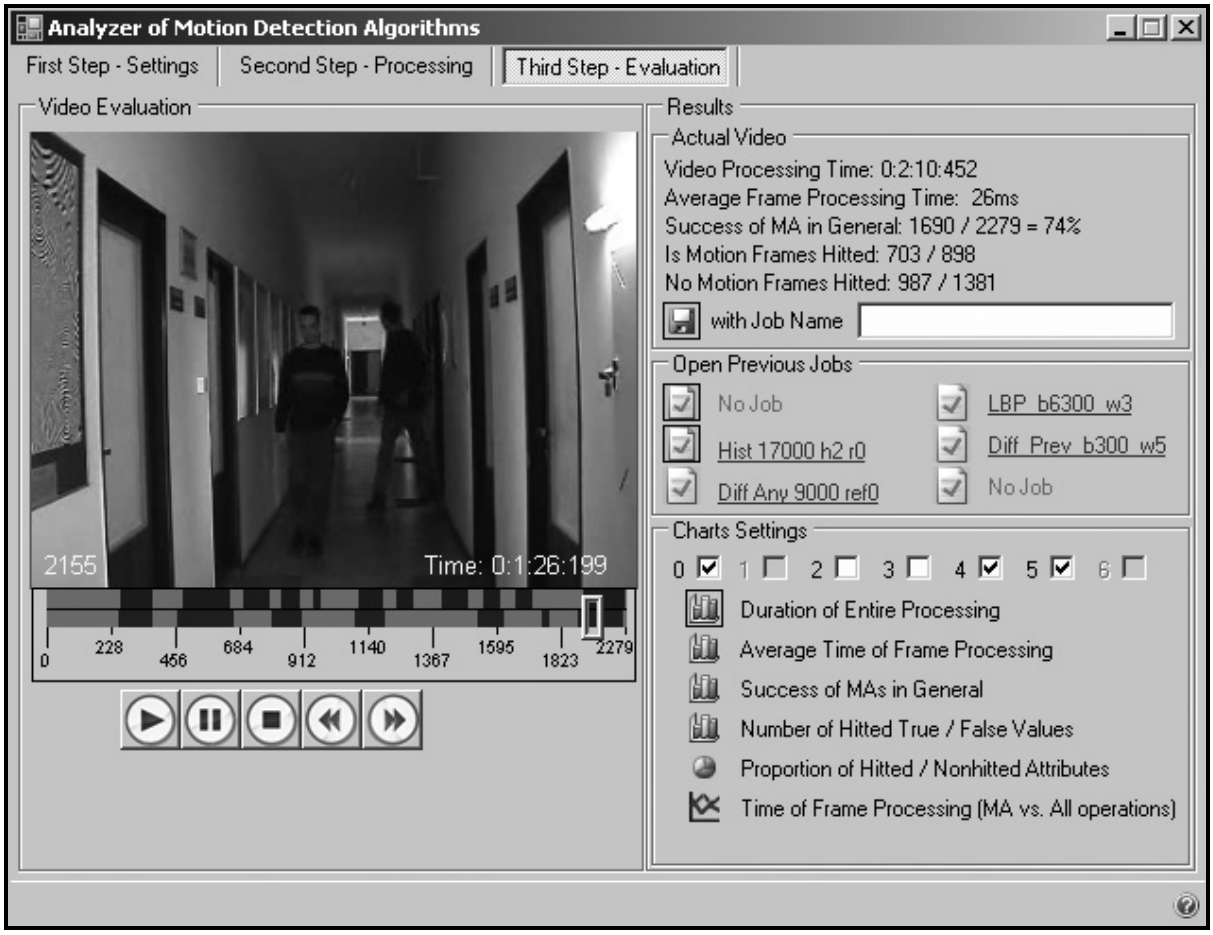
Třetí krok – vyhodnocení a porovnání

V závěrečné fázi jsou ovládací prvky zejména pro vyvolání statistik, grafů, prohlížení a porovnávání získaných výsledků při tomto nebo jakémkoli minulém měření. Levá strana má opět funkci pouhého přehrávače videa, ale smyslem je podívat se, na kterých místech se liší intervaly označené uživatelem s vypočtenými detektorem a případně rovnou odhalit, čím by to mohlo být.

Pravá strana nahoře ukazuje statistické údaje k aktuálně vyhodnocenému úkolu a pod ní je ikona pro uložení úkolu do souboru. Do připraveného boxu vedle lze vepsat identifikační název, který se pak ukazuje v grafech a slouží ke snadnému odlišení měření. Pokud zůstane box při uložení nevyplněný, bude název úkolu totožný se jménem ukládaného souboru. Oblast pod statistikou aktuálního úkolu je věnována možnosti načíst si statistiku až šesti dříve uložených měření. Po načtení ze souboru se zobrazí název úkolu formou hypertextového odkazu, který po kliknutí otevře okno a v něm zobrazí načtené statistické údaje.

Spodní část je věnována ikonám pro vyvolávání grafů (zobrazují se v samostatném okně a lze jich otevřít více naráz). Vstupními hodnotami pro výpočet grafu jsou jednotlivé úkoly. Které z nich do grafu budou vstupovat rozhoduje zatržení boxu s číslem úkolu s tím, že 0 je aktuální úkol a další čísla reprezentují načtené úkoly 1-6 (číslováno postupně shora dolů, přechod na další sloupec a zase shora dolů). Graf s ikonou tradičního „koláče“ a křivkový **XY** vyžadují, aby byl zatržen pouze jeden z úkolů. Který ze statistických údajů do grafu vstupuje a je porovnáván je stručně popsáno vedle ikony. Aplikace umožňuje zobrazit tyto grafy:

- sloupcový graf porovnání délky celkového průběhu zpracování
- sloupcový graf průměrné doby zpracování snímku
- sloupcový graf celkové úspěšnosti metod v počtu správně označených snímku (**true positive** a **true negative**)
- sloupcový graf porovnání úspěšně a neúspěšně označených snímků v rámci sloupcových grafů pro více úkolů zaráz
- graf typu „koláč“ reprezentující účinnost (resp. úspěšnosti) metody
- **XY** graf znázorňující časové trvání vyhodnocení jednotlivých snímků v porovnání mezi časem výpočtu celého snímku a časem pouze samostatné práce detektoru



(Obrázek 33) Ukázka aplikace ve třetím kroku - vyhodnocení

Příloha 2: Tutoriál měření v aplikaci

V této příloze bude na příkladech prakticky ukázáno jak lze provést měření od otevření videa po získání statistických výsledků a grafů. Na přiloženém elektronickém nosiči jsou videa na kterých byly provedeny testy jejichž výsledky diskutuje kapitola **5**. Součástí tohoto tutoriálu bude tabulka nastavení jednotlivých detektorů pro získání totožných výsledků. Reprodukcí měření podle těchto parametrů lze snadno pochopit princip ovládání.

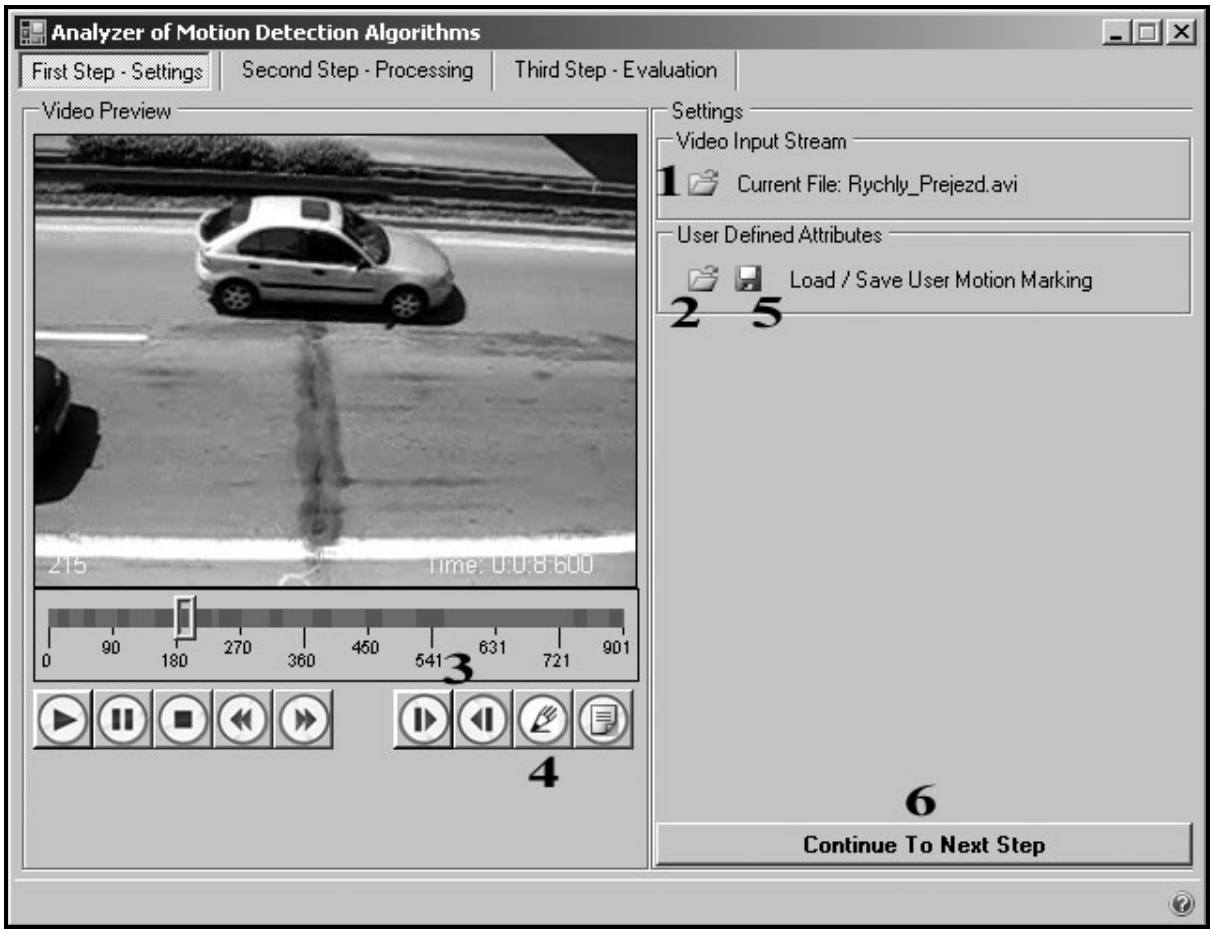
Instalace a spuštění aplikace

8. Aplikace vyžaduje v systému nainstalovaný **.NET Framework 2.0** a vyšší. Verzi pro **Windows XP** naleznete na přiloženém **CD** ve složce: **Testovaci Aplikace\.NET Framework 2\ dotnetfx.exe.**
9. Video sekvence se kterými se bude v aplikaci pracovat vyžadují v systému instalovaný příslušný video kodek. Pokud aplikace zobrazí chybové hlášení **Failed to open video**, je třeba příslušný kodek nainstalovat a nebo použít nekomprimované video. Videa pro testování jsou všechna ve formátu **DivX 4** se kterým si poradí i každá vyšší verze.
10. Testovací aplikace je na **CD** umístěna ve složce **Testovaci Aplikace** a spouští se souborem **Diplomka_xjelin19.exe**.

První krok: otevření videa a značení pohybu

Jednotlivé položky číslovaného seznamu odpovídají značení na **(Obrázek 34)**.

1. Otevřít testovací video. Pokud u sebe video má soubor s uloženým značkováním pod stejným názvem objeví se nabídka jestli se má automaticky načíst.
2. Otevřít soubor s uloženým značkováním, pokud nechceme značkovat znovu.
3. Pomocí pohybu jezdce ve video sekvenci nastavit levou a pravou hranici intervalu snímků s pohybem. Intervaly se nastavují kliknutím na tlačítko levé / pravé hranice (na obrázku pod číslem 3).
4. Přepínání mezi módem, kdy se označují intervaly s pohybem (symbol tužky) a mazáním těchto intervalů (symbol guma). Dvojklik na jezdec umožní smazat celé značkování.
5. Uložení vyznačených intervalů pro budoucí použití. Uložení se stejným jménem jako video soubor při načítání videa vyvolá nabídku pro automatické načtení.
6. Přechod do dalšího kroku

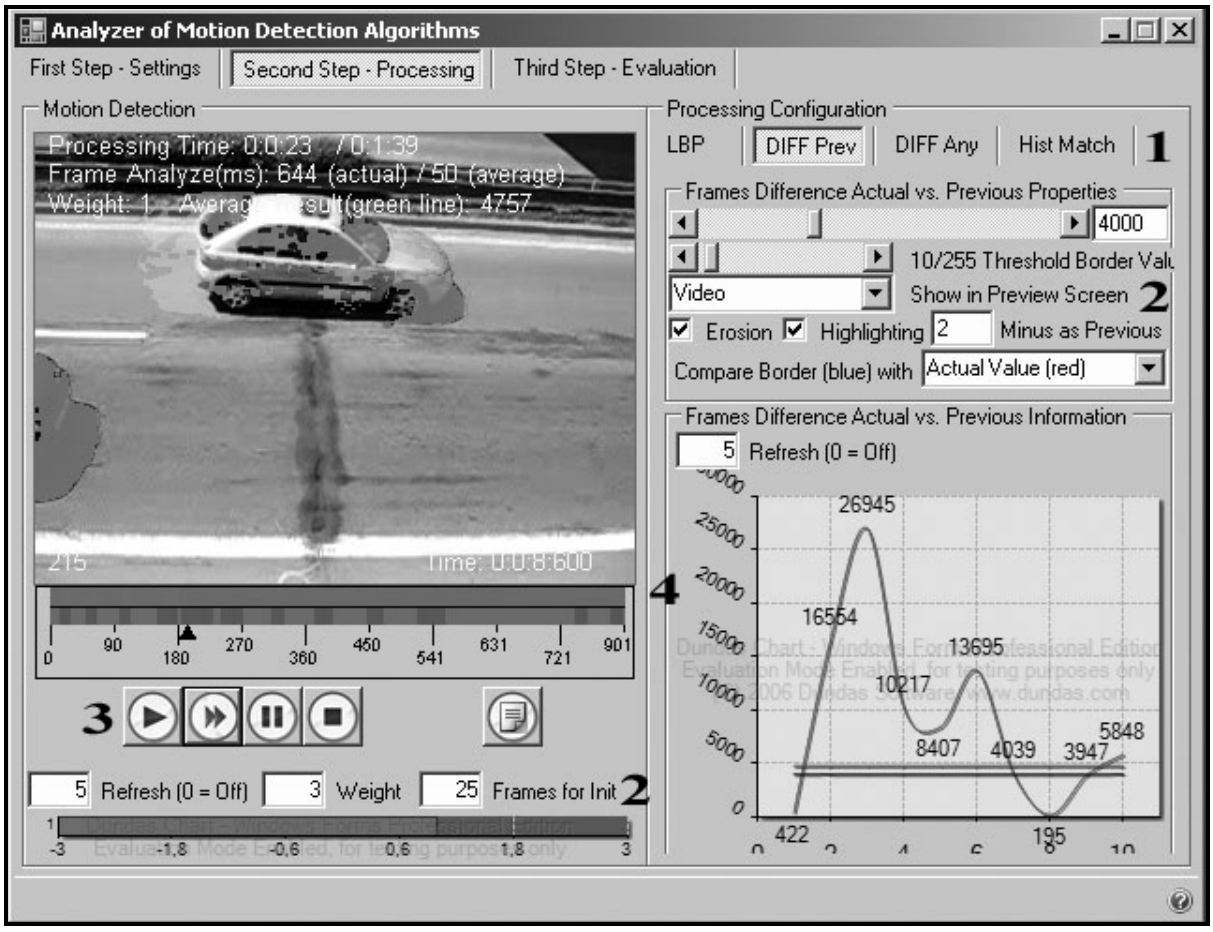


(Obrázek 34) Postup průchodu prvním krokem (značkování uživatelem)

Druhý krok: značení pohybu detektory

Jednotlivé položky číslovaného seznamu odpovídají značení na **(Obrázek 35)**.

1. Každá záložka reprezentuje jeden detektor pohybu. Vybrání záložky zobrazí její možná nastavení a pro dále hodnocené snímky se bude takto zvolený detektor používat.
2. Nastavení parametrů vybraného detektoru. Význam jednotlivých položek nastavení je popsán v příloze 1. Nastavovat lze i za chodu a nové parametry se promítnou při zpracování dalšího snímku. Jaké hodnoty oproti přednastaveným zvolit u přiložených video sekvencí prezentuje **(Tabulka 6)**.
3. Ovládání chodu detektoru. Význam jednotlivých tlačítek je popsán v příloze 1, ale jedná se o spuštění detekce, pozastavení, zpracování pouze po jednom snímku a resetování s přesunem na začátek. Ikonou napravo lze zapnout / vypnout zobrazení údajů v obraze.
4. Podle grafů a průběhu značkování, které se zobrazuje v místech nalezeného pohybu červenou čarou na jezdci je třeba upravovat parametry detektoru. Jakmile jsme s nimi spokojeni, tak je dobré dosud označenou část resetovat a nechat proběhnout od prvního snímku po konec. Jakmile je zpracován poslední snímek, objeví se nabídka pro přesun do dalšího kroku.



(Obrázek 35) Postup průchodu druhým krokem (značkování detektorem)

V následujících tabulkách jsou uvedeny hodnoty parametrů detektorů tak, jak byly při testování v kapitole **5** nastaveny u měřených video sekvencí. S těmito hodnotami lze ověřit dříve prezentované výsledky, ale zejména by měli sloužit pro snadné nastavení detektorů při prvních pokusech s aplikací.

Chodba	**Border**	**Weight**	**Threshold**	**Ref / Prev**	**H. Scale**
LBP	2150	10	x	x	x
Diff Prev	160	10	10	5	x
Diff Any	1200	10	8	0	x
Hist Match	9500	10	x	99	8
Terc	**Border**	**Weight**	**Threshold**	**Ref / Prev**	**H. Scale**
LBP	2500	10	x	x	x
Diff Prev	500	10	10	5	x
Diff Any	9000	10	10	30	x
Hist Match	17000	10	x	0	2
Rychly_Prejezd	**Border**	**Weight**	**Threshold**	**Ref / Prev**	**H. Scale**
LBP	4300	3	x	x	x
Diff Prev	4000	3	10	2	x
Diff Any	5000	3	10	0	x
Hist Match	18000	3	x	0	4
Silnice_z_Mostu	**Border**	**Weight**	**Threshold**	**Ref / Prev**	**H. Scale**
LBP	1200	5	x	x	x
Diff Prev	300	5	10	0	x
Diff Any	3000	3	13	0	x
Hist Match	4500	3	x	0	4
Metro	**Border**	**Weight**	**Threshold**	**Ref / Prev**	**H. Scale**
LBP	6300	3	x	x	x
Diff Prev	200	2	10	1	x
Diff Any	2500	3	10	0	x
Hist Match	7000	4	x	10	4

(Tabulka 6) Tabulka nastavených parametrů detektorů při testování v kapitole 5

Třetí krok: vyhodnocení a výsledky

1. Prohlédnout si textově zobrazené výsledky měření a případně si je uložit pro porovnávání s předchozími projekty.
2. Načtení dříve uložených výsledků měření. Po kliknutí na název načteného projektu se zobrazí okno s textovým výčtem naměřených hodnot. Načíst lze až šest dříve uložených výsledků.
3. Vizualizace výsledků formou různých grafů. Sloupcové grafy umožňují porovnat více projektů najednou a které z nich do grafu vstoupí se řídí zatrhnutím příslušného čísla v horní části. Do jednoho koláčového grafu se promítnou pouze výsledky jednoho měření. Poslední **XY** graf zobrazuje časový průběh pouze aktuálního projektu.

(Obrázek 36) Zobrazení výsledků ve třetím kroku