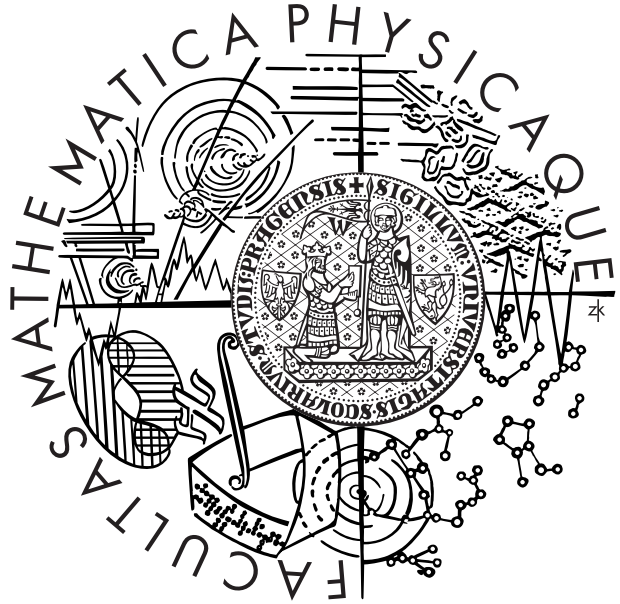
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Michal Wirth

Správce kamerového systému s podporou detekce pohybu

Kabinet software a výuky informatiky

Vedoucí bakalářské práce: RNDr. Jan Horáček

Studijní program: Informatika

Studijní obor: programování

Praha 2011

Tímto bych velmi rád poděkoval mým rodičů za významnou podporu v mém studiu. Velké díky patří rovněž panu RNDr. Janu Horáčkovi za jeho rady a trpělivost při vedení této práce.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V Praze dne 9. prosince 2011

.....

Michal Wirth

Název práce:
Správce kamerového systému s podporou detekce pohybu

Autor:
Michal Wirth

Katedra:
Kabinet software a výuky informatiky

Vedoucí bakalářské práce:
RNDr. Jan Horáček, Kabinet software a výuky informatiky

Abstrakt:
Předmětem této práce je návrh a implementace aplikace pro správu menšího kamerového systému. Jsou zkoumány způsoby uspořádání kamerového systému. Na základě zjištěných skutečností je následně navržena vhodná architektura aplikace. Implementovaný systém je představen jak z pohledu uživatele, tak ve více technickém směru.
Zároveň je také zkoumána i problematika detekce pohybujících se objektů ve statických scénách, a to v reálném čase. Představené algoritmy jsou v aplikaci implementovány, testovány z hlediska jejich účinnosti a následně porovnány.

Klíčová slova:
kamerový systém, detekce pohybu, odčítání pozadí

Title:
CCTV system controller including motion-detection feature

Author:
Michal Wirth

Department:
Department of Software and Computer Science Education

Supervisor:
RNDr. Jan Horáček, Department of Software and Computer Science Education

Abstract:
The object of this work is to design and implement an application for managing a smaller surveillance system. There are analyzed several ways of it's possible arrangement. After then an appropriate application's architecture is designed based on discovered facts. Implemented system is introduced from user's point of view and also in a more technical sense.
There is also a research of algorithms for real-time detection of moving objects within static scenes. Introduced ones are implemented, measured for their efficiency and compared with each other.

Keywords:
surveillance system, motion detection, background subtraction

Obsah

Úvod

V dnešním světě jsou v mnoha budovách, na ulicích a jiných veřejných prostranstvích často instalovány nejrůznější druhy kamer, jejichž úkolem je sledovat a dokumentovat dění v bezprostředním okolí – obvykle za účelem zvýšení bezpečnosti osob nebo majetku. Díky rozmachu digitální techniky se stalo běžné tyto kamery sdružovat v komplexní a sofistikované kamerové systémy s centralizovanou správou. Jejich vybudování však není úplně levnou záležitostí.

Cílem této práce je navržení a implementace softwaru, který by umožnil např. domácnostem, menším podnikům nebo malým muzeím či galeriím podobný kamerový systém realizovat svépomocí, a to pokud možno s využitím již existujících prostředků a s minimem dalších nákladů. Většinou tito potenciální uživatelé totiž již mají k dispozici potřebnou síťovou infrastrukturu, běžně používají osobní počítače a vlastní nějaké webové či poloprofesionální kamery. Jediné, co by jim tak mohlo scházet, je vhodný nástroj v podobě „desktopové“ aplikace, která by dokázala zmíněné vybavení propojit do kompaktního kamerového systému.

1. Analýza a návrh systému

1.1 Uspořádání

Nejprve nahlédněme, na jakém principu by náš kamerový systém, složený z obecně libovolného počtu kamer, mohl pracovat. Byly uvažovány celkem dva protichůdné koncepty.

Prvním z nich je idea, kdy každá kamera by v roli klienta sama neustále zasílala aktuální obrazová data přes síťovou infrastrukturu na vzdálený server. Ten by poté zajišťoval jejich další případné zpracování. Takový systém by jistě mohl fungovat, nicméně je poněkud těžkopádný. Např. v situacích, kdy server data z konkrétní kamery momentálně nepotřebuje, ale přesto by je obdržel. Přenosová síť by se tak mohla snadno zahltit daty, která nikdo nevyžaduje. Aby tomu mohlo být nějak zabráněno, musel by komunikační protokol nabízet možnost i zpětné vazby od serveru ke kameře, a to za účelem jejího řízení. Jak je vidět, takový kamerový systém by již z principu byl příliš složitý a nepružný.

Daleko lepším řešením je koncept přesně opačný. Každá kamera by zde byla v roli serveru a pouze by na požádání nabízela své služby. Zpracování obrazu by tak měl na starosti klient, který by služeb kamer mohl libovolně využívat. Takové jednoduché uspořádání nevyžaduje žádný složitý komunikační protokol. Navíc přináší i řadu dalších možných scénářů použití. Např. umožňuje, aby jedna kamera mohla být sdílena více kamerovými systémy zároveň.

1.2 Komunikace a formát dat

Jedním z problémů, které musíme rovněž uvážit, je forma komunikace systému se vzdálenou síťovou kamerou. Existuje velké množství různých transportních protokolů a snad ještě více použitelných algoritmů pro kompresi obrazových dat. Máme jako transportní vrstvu zvolit spolehlivé TCP? Nebo nám stačí nic nezaručující UDP? Je vhodným kandidátem pro přenos dat např. RTP?

Všechny takové otázku jsou do jisté míry bezpředmětné. Jak již bylo řečeno v úvodu, snažíme se o implementaci kamerového systému určeného pro prostředí domácností apod. Ty většinou ale nedisponují žádnými profesionálními zařízeními s velkou variabilitou možností komunikace. Z tohoto důvodu je třeba zvolit takový protokol a formát dat, který bude jednoduchý a především maximálně univerzální.

Naštěstí existuje řešení, které tyto nároky bezezbytku splňuje. Je jím neoficiální formát MJPEG přenášený pomocí HTTP. Ve zkratce řečeno se jedná o jednoduchý způsob přenosu videa, kdy jednotlivé snímky jsou komprimovány

samostatně pomocí kompresního algoritmu JPEG, a jsou postupně zasílány druhé straně v rámci jediné neustále otevřené HTTP odpovědi. Bližší informace o tomto způsobu přenosu lze nalézt v kapitole 2.3.

Proč právě tato forma komunikace? Většinou je totiž nabízena přímo samotnou kamerou nebo případně dodatečným síťovým softwarem. Důvod je nasnadě. I ty nejlacinější webové kamery mají v rámci svého hardwarového návrhu zabudovanou velmi efektivní implementaci algoritmu JPEG. Dokáží tak bez větších problémů produkovat jednotlivé snímky již ve zkomprimované podobě, a to ve velkém rozlišení s frekvencí i 30 snímků za vteřinu.

1.3 Předešlé návrhy

Tato práce není prvním pokusem o navržení a implementaci podobného software. Jeden již byl učiněn v rámci mého ročníkového projektu, kdy správce kamerového systému byl realizován formou vícevláknové aplikace, ve které každé vlákno mělo za úkol zpracovávat obraz z právě jedné kamery. Jak se později ukázalo, tento klíčový bod návrhu trpěl dvěma závažnými neduhy.

Obecně nelze nikdy zcela vyloučit riziko, že při zpracovávání obrazu některé z kamer nedojde k nějaké neočekávané fatální chybě. V důsledku takové chyby, pak není ukončeno pouze vlákno, ve kterém chyba nastala, nýbrž dojde k nekontrolovanému pádu celého procesu, a tím k odstavení i celého kamerového systému.

V návrhu nebyla ošetřena ani situace, kdy je některé z vláken zahlcováno příliš velkým množstvím dat. Vlákno je pak nestíhá dostatečně rychle odbavovat a ve výsledku tak zpracovává neaktuální obraz.

Protože se jedná o aplikaci bezpečnostního charakteru, jsou oba zmíněné nedostatky velmi závažnými problémy, které nelze jen tak ignorovat.

1.4 Zvolené řešení

Poučeni z předchozích chyb a se znalostí organizace kamerového systému můžeme přistoupit k návrhu finální architektury. Jistě bude výhodné rozdělit práci mezi více samostatných procesů. Snížíme tak riziko ohrožení celého kamerového systému v důsledku vzniku neočekávaných chyb.

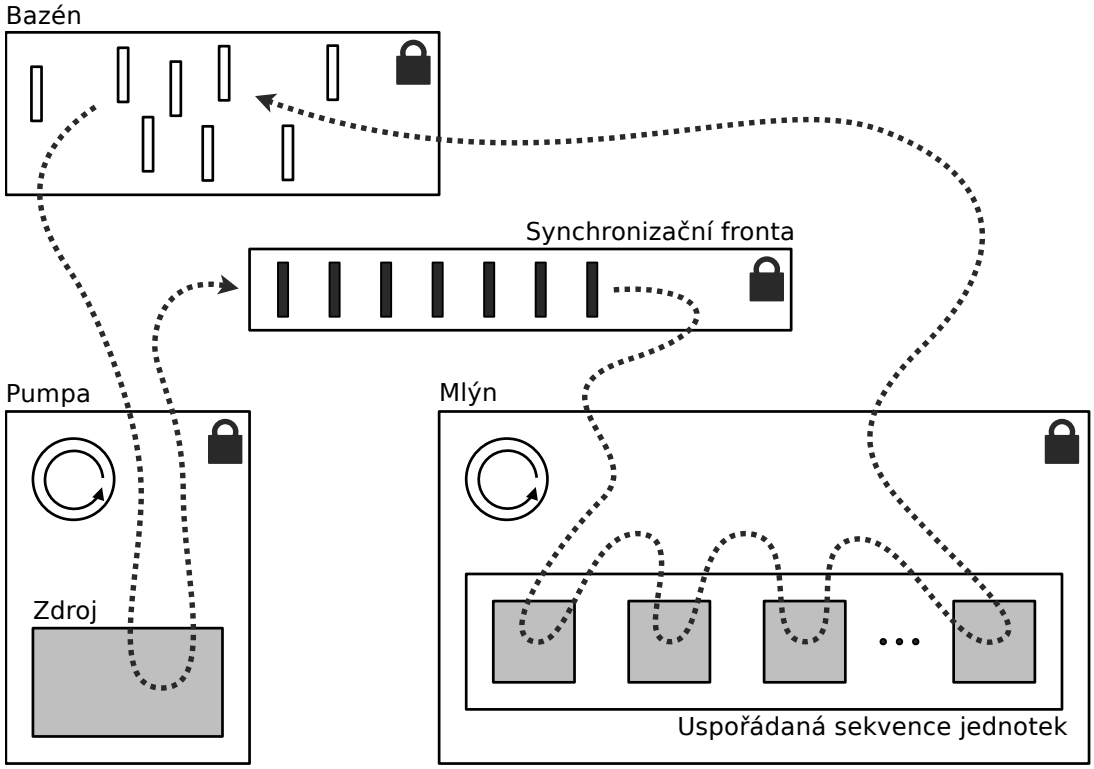
Nabízí se jedno přirozené dělení – obraz každé kamery zpracovávat právě jedním procesem. Program, jenž bude tuto práci vykonávat, nazvěme tedy příznačně Accipiter Camera. Správcem kamerového systému pak nazveme aplikaci Accipiter Manager, která bude jednotlivé výkonné procesy řídit, a bude k nim nabízet jednotné uživatelské rozhraní.

Nyní je potřeba rozhodnout, jakým způsobem se má obraz kamery výkonným procesem zpracovávat. Zde můžeme sáhnout k některému z ustálených návrhových vzorů. Vhodným kandidátem je vzor Producent-Konzument [8]. Pro naše potřeby nazvěme producenta pumpou a konzumenta mlýnem.

Idea je taková, že pumpa bude neustále „čerpat“ ze zdroje (reprezentace vzdálené kamery) nové snímky a vkládat je do tzv. synchronizační fronty. Z té budou následně vybírány mlýnem, jehož úkolem bude tyto snímky zpracovat aplikováním tzv. jednotek (provádí nad snímky předem dané operace).

Je důležité, aby pumpa a mlýn pracovali samostatně a nezávisle na sobě. Synchronizační fronta navíc musí v případě přeplnění zahazovat nejstarší, mlýnem ještě nezpracované, snímky. To je ochrana proti zmiňovanému riziku zahlcení.

Celý zde popsaný koncept je znázorněn i graficky na obrázku 1.1.



Obrázek 1.1: Schéma konceptu toku řízení při zpracovávání snímků

1.5 Problematika detekce pohybu

Většina kamerových systémů, které jsou součástí komplexnějšího systému zabezpečení budov a jiných objektů, se skládá především ze sady pevně instalovaných nepohyblivých kamer. V této kapitole se budeme zabývat tím, jak lze takový kamerový systém doplnit o funkci detekce pohybu v reálném čase. To by případně

systému umožnilo na vzniklou situaci ve sledovaném prostoru adekvátně a s jistou mírou autonomie reagovat.

Pro detekci pohybu, v jinak statické scéně, se nabízí jeden zcela přímočarý obecný postup. Z aktuálního obrazu lze segmentovat oblasti s pohybujícími se objekty jednoduchou metodou, která pracuje na bázi samotných pixelů. Jedná se o metodu tzv. odčítání pozadí (angl. background subtraction). Každý pixel s hodnotou x_t (typicky se zde užívá hodnota jasu) se jednoduše odečte od analogického pixelu z referenčního modelu pozadí s hodnotou b_t podle následujícího vztahu:

$$|x_t - b_t| \; > \; T, \tag{1.1}$$

kde parametr T, jak v článku píší např. Šmirg et al. [6], je vhodně nastavený práh, který rozhodne, zda daný pixel bude klasifikován jako popředí (tj. je součástí pohybujícího se objektu) či nikoliv.

Zde však narážíme na problém, jak potřebný referenční model pozadí získat. Algoritmus, který by toto nabízel, se musí umět alespoň částečně vypořádat s postupnými i náhlými změnami jasu celého obrazu kamery (např. jako důsledek pohybu slunce a mraků), s jistou mírou šumu (plynoucím z nedokonalosti snímačů v kamerách), s rychle se měnícími objekty v pozadí (např. třepotání vlajky ve větru nebo listí na stromech), s trvalými změnami pozadí (např. zaparkovaná vozidla) a s mnoha dalšími nepříznivými vlivy [4].

Jak se lze dočíst v článku od Cheunga a Kamath [5], je známa celá řada algoritmů modelujících pozadí. Lze je v zásadě rozdělit na dvě skupiny:

- Nerekurzivní algoritmy pracují na principu fronty, která udržuje několik posledních zachycených snímků. Jelikož je často potřeba mít k dispozici větší výhled do historie, je z kapacitních důvodů do fronty zařazen jen každý k-tý snímek. Nad touto frontou poté operuje např. oblíbený mediánový filtr. V takovém případě se hodnota b_t pro vztah 1.1 jednoduše spočte jako medián všech hodnot $x_{t-1}, x_{t-2}, ..., x_{t-n}$ odpovídajících pixelů u snímků ve frontě.

  Mezi další nerekurzivní algoritmy patří např. Wienerův filtr, metoda odhadování jádrových hustot (angl. Kernel Density Estimation) nebo algoritmus Eigen-background.

- Naopak rekurzivní algoritmy žádnou takovou frontu nepoužívají. Místo toho si model pozadí budují postupně ze všech příchozích snímků. Jako zástupce takového typu algoritmů lze vybrat např. různé druhy klouzavých průměrů. Ty v principu hodnotu b_t pro vztah 1.1 s každým dalším snímkem aktualizují na součet hodnot x_t a b_{t-1} rozdělených v předem daném poměru.

Dále do této kategorie algoritmů patří např. Kalmanův filtr nebo metoda s názvem Směs gaussiánů (angl. Mixture of Gaussians).

Některým z výše uvedených metod se budeme více věnovat v následujících podkapitolách.

1.5.1 Klouzavý Gaussův průměr

Tato rekurzivní metoda, jak ji např. popisuje ve své práci Gallego [3], je postavena na následujícím předpokladu. Na každý pixel je nahlíženo jako na náhodnou veličinu určenou normálním (neboli Gaussovým) rozdělením pravděpodobnosti. To je charakterizováno dvojicí parametrů: střední hodnotou μ a rozptylem σ^2. Celý model pozadí je pak tvořen soustavou takových rozdělení.

S každým dalším snímkem v čase t algoritmus upraví oba parametry jednotlivých rozdělení na základě aktuální hodnoty jasu x_t příslušného pixelu. Úprava je prováděna na principu exponenciálního klouzavého průměru (váha starších hodnot exponenciálně klesá, avšak nikdy nedosáhne nuly), tj. pomocí následujícího předpisu:

$$\begin{aligned} \mu_t &= \alpha \cdot x_t + (1-\alpha) \cdot \mu_{t-1} \\ \sigma_t^2 &= \alpha \cdot (x_t - \mu_t)^2 + (1-\alpha) \cdot \sigma_{t-1}^2, \end{aligned} \tag{1.2}$$

kde α je vstupní parametr algoritmu ovlivňující rychlost adaptace modelu pozadí podle aktuální situace na snímané scéně. Běžné nastavení je $\alpha = 0.01$.

Bezprostředně po aktualizaci obou parametrů konkrétního rozdělení je algoritmem vyhodnocena následující rozlišující podmínka:

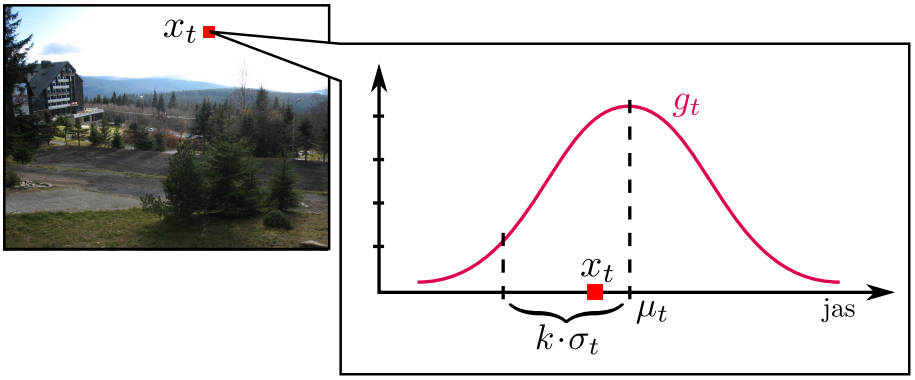
$$|x_t - \mu_t| > k \cdot \sigma_t, \tag{1.3}$$

kde k je předem definovaný koeficient pro určení prahové hodnoty. Obvykle se zde užívá hodnota $k = 2.5$, která plyne z charakteru normálního rozdělení.

V případě, že je podmínka 1.3 splněna, pixel s hodnotou x_t je označen jako součást momentálního popředí. V opačném případě, tj. hodnota x_t náleží příslušnému modelovanému rozdělení, je pixel zahrnut do pozadí. Na obrázku 1.2 je tento koncept lépe graficky znázorněn.

1.5.2 Směs gaussiánů

Metoda klouzavého Gaussova průměru, která byla popisována v předchozí kapitole, selhává mimo jiné i v situacích, kdy hustota rozdělení pravděpodobnosti náhodné veličiny (tj. konkrétního pixelu) nemá pouze jedno lokální maximum.



Obrázek 1.2: Vzorový případ, kdy daný pixel s aktuální hodnotou jasu x_t zapadá do modelovaného pozadí a tudíž nebude algoritmem 1.5.1 klasifikován jako popředí.

Takové tzv. „multimodální“ rozdělení totiž nelze pomocí normálního rozdělení dobře aproximovat a tudíž v daném místě modelu pozadí dochází k chybám. Popisovaný jev bohužel není výjimečným. Typicky nastává v místech obrazu, kde dochází k pohybu objektů s vysokou frekvencí. Příkladem může být již zmíněné třepotání vlajky ve větru nebo listí na stromech apod.

Právě tento nedostatek se snaží odstranit metoda směsi gaussiánů, která byla poprvé prezentována v článku Stauffera a Grimsona [2]. Jak už její název napovídá, konkrétní pixel je místo jednoho rozdělení modelován celou řadou normálních rozdělení, označovaných jako gaussiány. Jedná se tak o svým způsobem přirozené rozšíření metody klouzavého Gaussova průměru. Následující popis algoritmu vychází jak ze zmiňovaného článku [2], tak i z práce Gallega [3].

Celý byl autory navržen tak, aby bylo možné hodnoty pixelů chápat buď jednoduše jako pouhé skalární hodnoty jasu nebo lépe jako vektory s jednotlivými barevnými složkami. My budeme uvažovat pouze druhou variantu, neboť se jedná o rys, který jiné algoritmy většinou nemívají.

Pravděpodobnost pozorování hodnoty x_t konkrétního pixelu v čase t je dána následujícím vztahem:

$$\mathrm{P}(x_t) \;=\; \sum_{i=1}^{m} \omega_{i,t} \cdot \eta\,(x_t,\; \mu_{i,t},\; \Sigma_{i,t}), \tag{1.4}$$

kde konstanta m je počet gaussiánů užitých v každém pixelu modelu (používá se hodnota 3-5, větší počet často již dále nepřináší lepší výsledky), $\omega_{i,t}$ je normalizovaná váha i-tého gaussiánu (udává, jak velkou část informace gaussián udržuje), $\mu_{i,t}$ je jeho střední hodnota a obdobně $\Sigma_{i,t}$ je jeho kovarianční matice. Konečně

η pak značí hustotu pravděpodobnosti ve tvaru n-rozměrné Gaussovy funkce:

$$\eta\left(x,\ \mu,\ \Sigma\right)\ =\ \frac{1}{(2\pi)^{\frac{n}{2}}\left|\Sigma\right|^{\frac{1}{2}}}\cdot e^{-\frac{1}{2}(x-\mu)^{T}\Sigma^{-1}(x-\mu)} \tag{1.5}$$

Ve většině případů uvažujeme hodnoty pixelů z barevného modelu RGB a tedy platí, že dimenze $n = 3$.

Protože zjištění inverzní matice Σ^{-1} je výpočetně náročnou operací, dopouští se zde Stauffer s Grimsonem v zájmu rychlosti záměrné chyby, kdy kovarianční matici zjednoduší do tvaru obyčejné diagonální matice:

$$\Sigma_{i,t}\ =\ \sigma_{i,t}^{2}\cdot \mathrm{I_n} \tag{1.6}$$

Předpokládají tím, že jednotlivé barevné složky jsou navzájem nezávislé a mají stejný rozptyl hodnot. Tento předpoklad je samozřejmě mylný, nicméně z výše uvedeného důvodu je omluvitelný.

Jak již bylo zmíněno, celý model se v místě konkrétního pixelu skládá ze směsi přesně m gaussiánů. Ne všechny tyto gaussiány však mohou být modelem pozadí. Tam lze vybrat pouze ty, které mají dostatečně velkou váhu a zároveň mají nízkou směrodatnou odchylku. Důvodem je předpoklad, že obraz pozadí scény se objevuje s větší frekvencí a je i více statický [3]. Je tedy výhodné směs gaussiánů v rámci každého pixelu udržovat vždy v setříděném stavu, a to sestupně podle poměru $\frac{\omega_{i,t}}{\sigma_{i,t}}$. Jako modelem pozadí bude tedy po vyhodnocení následující podmínky označeno prvních B gaussiánů:

$$B\ =\ argmin_{b}\left(\sum_{i=1}^{b}\omega_{i,t} > T\right), \tag{1.7}$$

kde T je předem zvolená hranice. Obvykle se používá hodnota $T = 0.6$.

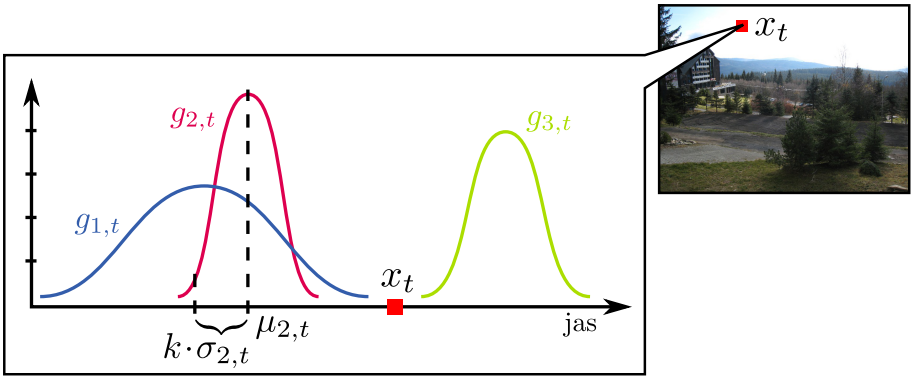
S každou novou hodnotou x_t konkrétního pixelu algoritmus vyhodnotí, zda náleží do některého z příslušných m gaussiánů nebo ne. Rozhodnutí je učiněno na základě této nerovnice:

$$\left|x_t - \mu_{i,t}\right|\ >\ k\cdot\sigma_{i,t}, \tag{1.8}$$

kde k je opět předem definovaná konstanta pro určení prahové hodnoty. Stejně jako v případě klouzavého Gaussova průměru se většinou používá $k = 2.5$.

V případě, že je nerovnice 1.8 splněna pro všechny gaussiány, hodnota x_t nenáleží ani jednomu z nich a tudíž je zřejmé, že daný pixel je součástí momentálního popředí a může tak být označen (viz doplňující obrázek 1.3).

Také je nutné v daném místě příslušně aktualizovat model. Jelikož žádný



Obrázek 1.3: Situace, kdy daný pixel s aktuální hodnotou jasu x_t neodpovídá žádnému z gaussiánů a tudíž bude algoritmem 1.5.2 jednoduše označen jako popředí.

gaussián hodnotě x_t nevyhovoval, je potřeba založit nový – konkrétně s počáteční střední hodnotou nastavenou na x_t, vysokým rozptylem a nízkým váhovým koeficientem [2]. Aby bylo možné gaussián do směsi přidat, musí být z ní jiný, kvůli zachování počtu m, odstraněn. Samozřejmě je třeba vybrat ten, který má ve směsi momentálně nejnižší váhu. Tento způsob aktualizace např. ve výsledku umožňuje, aby do modelu pozadí mohly být časem zahrnuty nové objekty, které již delší dobu „setrvávají na svém místě“ (např. čerstvě zaparkovaná vozidla na ulici a podobně).

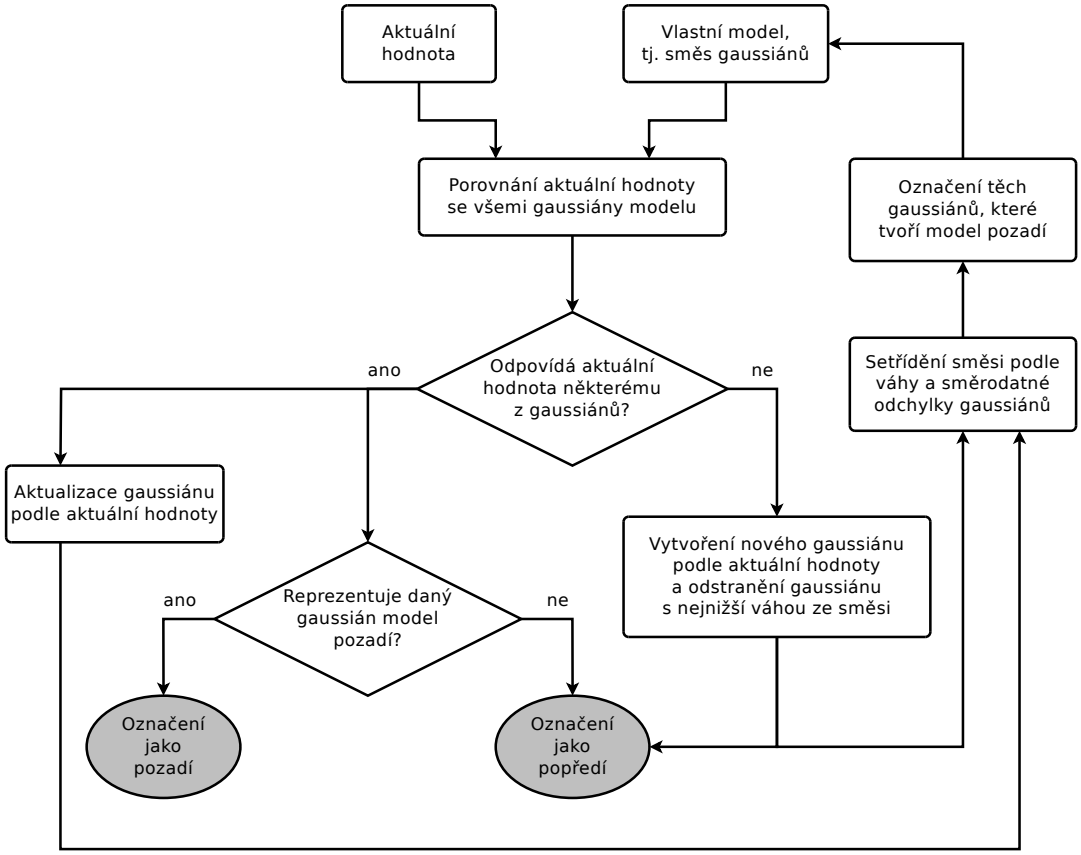
Může nastat i opačný případ, kdy nerovnice 1.8 není splněna pro jeden či více gaussiánů a hodnota x_t tak odpovídá modelu. Pokud gaussián, do kterého x_t náleží, je jedním z prvních B gaussiánů směsi, je daný pixel přirozeně označen jako součást momentálního pozadí. Pokud se nejedná o takový gaussián, pixel je klasifikován jako popředí. V případě, že x_t náleží do více gaussiánů, je vybrán ten s největším váhovým koeficientem a nejmenším rozptylem [3].

Odpovídající gaussián je samozřejmě nutné také aktualizovat. Jeho úprava je prováděna opět na principu exponenciálního klouzavého průměru (viz vztah 1.2), tj. podle tohoto předpisu:

$$\begin{aligned} \mu_{i,t} &= \rho \cdot x_t + (1-\rho) \cdot \mu_{i,t-1} \\ \sigma^2_{i,t} &= \rho \cdot (x_t - \mu_{i,t})^T (x_t - \mu_{i,t}) + (1-\rho) \cdot \sigma^2_{i,t-1}, \end{aligned} \tag{1.9}$$

kde Stauffer s Grimsonem užívají $\rho = \alpha \cdot \eta\,(x_t,\, \mu_{i,t-1},\, \Sigma_{i,t-1})$. Vstupní parametr α určuje rychlost adaptace modelu aktuální situaci na snímané scéně. Jak píše Gallego [3], obvykle se zde užívá hodnota $\alpha = 0.005$.

Ať již nerovnice 1.8 byla či nebyla platná, je ještě potřeba aktualizovat jednotlivé váhové koeficienty celé směsi. Pro jejich úpravu se užívá následující vztah,



Obrázek 1.4: Průběh jednoho kroku algoritmu směsi gaussiánů

který zároveň zachovává jejich normalizovaný charakter:

$$\omega_{i,t} \;=\; (1-\alpha)\cdot\omega_{i,t-1} \;+\; \alpha\cdot M_{i,t}, \tag{1.10}$$

kde koeficient $M_{i,t}$ nabývá hodnoty 1 pro gaussián odpovídající x_t a naopak hodnoty 0 pro ostatní gaussiány.

Celý popsaný krok algoritmu je pro snazší orientaci také znázorněn na diagramu 1.4, který byl převzat z práce Michala Jůzy [7].

2. Systémová architektura

Tato kapitola se zabývá výhradně popisem architektury programu Accipiter Camera. Jak již bylo zmíněno v kapitole 1.4, aplikace Accipiter Manager je spíše pouhou grafickou nadstavbou. Její implementace se skládá převážně z přímočaré definice grafických uživatelských rozhraní a jejich vzájemného provázání. Nejedná se tedy o žádný sofistikovaně uspořádaný program, který by potřeboval nějaký podrobnější popis. Z tohoto důvodu bylo zde od něj upuštěno.

2.1 Struktura projektu

Celá aplikace je již poměrně rozsáhlým projektem. Bylo tedy nutné přistoupit k lepší organizaci jejích zdrojových kódů – jak pomocí jmenných prostorů (viz schéma 2.1), tak oddělením obecných prostředků do separátní knihovny. Hlavní části se nachází v těchto adresářích:

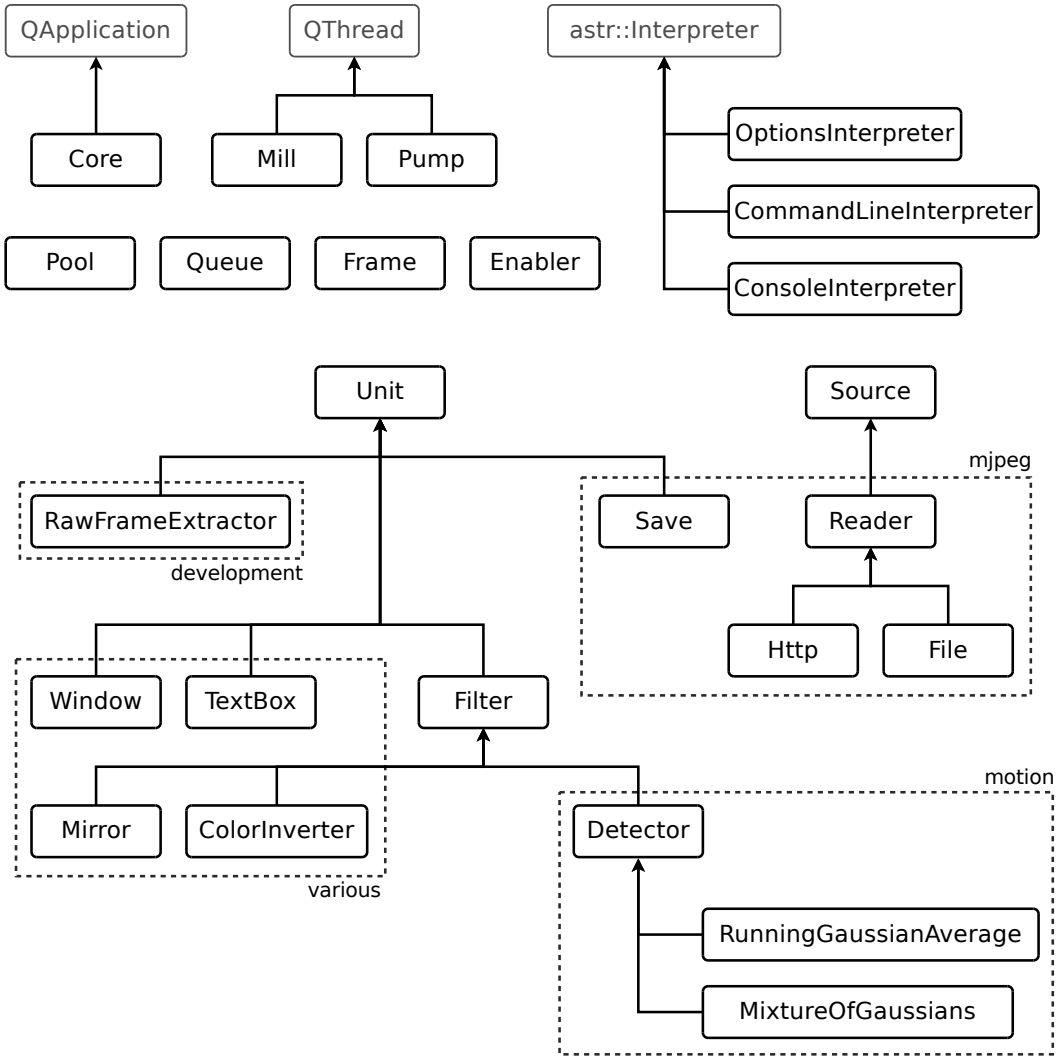
`atoolbox/`
: Zde sídlí zmiňovaná oddělená knihovna. Při překladu aplikace je k ní vždy staticky „přilinkována“. Obsahuje obecně použitelné prostředky, které bylo nutné při vývoji aplikace připravit. Dostupný je např. nástroj pro logování, důmyslný interpret řetězců, prostředky pro správu objektů a jiné. Součástí knihovny je i bohatá referenční dokumentace, která je dostupná na přiloženém DVD – viz příloha A.

`src/core/`
: Adresář obsahuje tzv. „core“ třídy, které ve výsledku implementují hlavní logiku aplikace. Tyto třídy nejsou součástí vlastního jmenného prostoru.

`src/sources/*/`
: Adresáře s implementacemi dostupných zdrojů. Jsou tématicky odděleny pomocí vlastních jmenných prostorů.

`src/units/*/`
: Analogicky adresář s jednotkami.

2.2 Přehled důležitých tříd



Obrázek 2.1: Přehled tříd s hierarchií dědičnosti v aplikaci Accipiter Camera

Třída `Core`

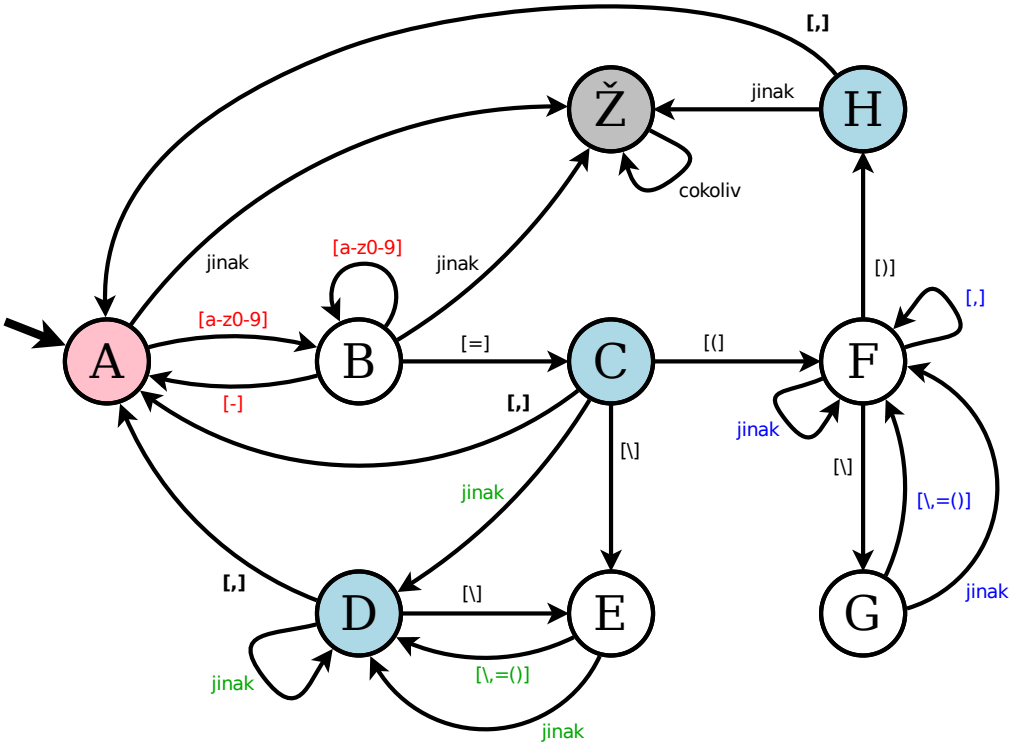
Sestavuje a propojuje hlavní části aplikace a implementuje tak její logiku. Výsledek pak přibližně odpovídá stavu zobrazeném na schématu 1.1.

Třída `Frame`

Instance této třídy přenáší jeden snímek z celkového proudu s videem. Detailnější informace lze nalézt v kapitole 2.4.

Třída `Pool`

Jedná se o implementaci návrhového vzoru Bazén (někdy označovaný jako Fond). Umožňuje aplikaci obejít časté alokace a dealokace instancí třídy `Frame` tím, že nabízí již jednou vytvořené objekty k jejich opětovnému použití. Třída je z pohledu vláken bezpečná.



Obrázek 2.2: Konečný automat rozpoznávající řetězce s nastavením

Třída `Queue`

Z pohledu vláken se jedná o implementaci bezpečné datové struktury typu fronta. Slouží jako synchronizační mezičlánek pro dočasné uchovávání snímků mezi pumpou a mlýnem.

Třída `Pump`

Spravuje zvolený zdroj a pomocí vlastního spuštěného vlákna z něj „čerpá“ další nové snímky.

Třída `Source`

Základní třída pro všechny odvozené zdroje. Především definuje rozhraní, které musí každý z nich implementovat.

Třída `Mill`

Spouští vlastní vlákno, které obstarává dílčí jednotky a řídí jejich aplikování na jednotlivé snímky vyjmuté ze synchronizační fronty. V terminologii aplikace je objekt této třídy nazýván mlýnem.

Třída `Unit`

Tentokrát základní třída pro všechny přítomné jednotky. Definuje povinné rozhraní a implementuje základní schopnosti jednotek.

Třída `Filter`

Jedná se o rozšíření třídy `Unit`, které je vhodné jako základ pro jednotky s konceptem založeným na principu filtru – viz příslušná část kapitoly 3.1.4.

Třída `Enabler`

Každá instance slouží jako kontejner pro jednu jednotku. Manipulací s parametrem `enabled` zjišťuje případné zapnutí jednotky během konstrukce a včasné vypnutí během její destrukce.

Třída `OptionsInterpreter`

Obstarává tokenizaci a interpretaci řetězců s nastavením – viz kapitola 3.1.2. K tomuto úkolu využívá konečný automat, jehož základní kostra je vyobrazena na obrázku 2.2.

Třída `CommandLineInterpreter`

Zajišťuje interpretaci parametrů aplikace zadaných při jejím spuštění na příkazové řádce.

Třída `ConsoleInterpreter`

Stará se o tokenizaci a následnou interpretaci příkazů zadaných na standardním vstupu aplikace.

2.3 Síťový protokol

Jak již bylo zmíněno a zdůvodněno v kapitole 1.2, proud videa ze vzdálené síťové kamery je přenášen ve formátu MJPEG. K jeho získání aplikace zasílá klasický HTTP požadavek na konkrétní adresu a TCP port kamery. Příkladem může být následující dotaz:

```
GET␣/?action=stream␣HTTP/1.1\r\n
Host:␣localhost:8080\r\n
User-Agent:␣kamerovy␣system\r\n
\r\n
```

Je nutné zvlášť zdůraznit, že dle specifikace protokolu musí být jednotlivé „řádky“ dotazu a odpovědí vždy ukončeny posloupností znaků `Carriage Return` a `Line Feed`. Příslušné ASCII hodnoty jsou `0x0d` a `0x0a`. Celý dotaz je navíc nutné vždy ukončit ještě jednou takovou dvojicí.

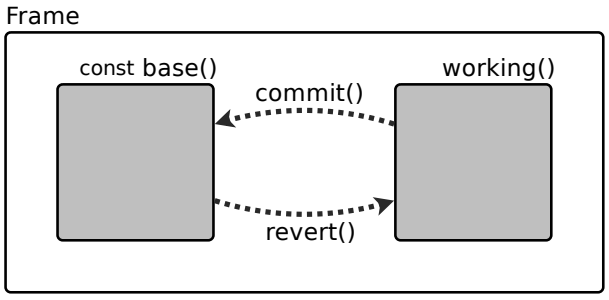
Po zaslání požadavku aplikace čeká, dokud neobdrží validní odpověď obsahující proud videa. Jednotlivé jeho JPEG snímky jsou v ní odděleny speciálním oddělovačem, jenž je specifikován v jejím záhlaví. Hodnota identifikátoru `Content-Type` je navíc nastavena na hodnotu `multipart/x-mixed-replace`. Následující výpis umožňuje získat lepší představu o formátu takové odpovědi:

```
HTTP/1.0␣200␣OK\r\n
Connection:␣close\r\n
Server:␣kamera\r\n
Pragma:␣no-cache\r\n
Expires:␣Mon,␣3␣Jan␣2000␣12:34:56␣GMT\r\n
Content-Type:␣multipart/x-mixed-replace;boundary=donotcross\r\n
\r\n
--donotcross\r\n
Content-Type:␣image/jpeg\r\n
Content-Length:␣10014\r\n
\r\n
01010010010001001000101001001001000100100101001010100
10010101001001001001001001011010101011100101001001011
01101001␣VISUALIZATION␣OF␣BINARY␣JPEG␣DATA␣0010001010
01010010010001001000101001001001000100100101001010100
10010101001001001001001001011100101001001011\r\n
--donotcross\r\n
Content-Type:␣image/jpeg\r\n
Content-Length:␣9959\r\n
\r\n
01010010010001001000101001001001000100100101001010100
10010101001001001001001001011010101011100101001001011
01101001␣VISUALIZATION␣OF␣BINARY␣JPEG␣DATA␣0010001010
01010010010001001000101001001001000100100101001010100
10010101001001001001001001010\r\n
...
```

Zde popsaný protokol je implementován v rámci zdrojového kódu třídy s příznačným názvem `mjpeg::Http`.

2.4 Práce s instancemi třídy `Frame`

Každá z nich, jak již název napovídá, slouží k reprezentaci jednoho snímku. Interně jsou třídou veškerá obrazová data ukládána jako sled hodnot jednotlivých pixelů za použití metody „double bufferingu“ [8].



Obrázek 2.3: Schéma vnitřního uspořádání jednoho snímku

Zjednodušeně řečeno každý snímek je složen ze dvou „podsnímků“ (viz obrázek 2.3). První z nich je přístupný pouze ke čtení, a to pomocí metody `base()`. Ke druhému lze přistupovat metodou `working()`, a to i pro zápis. Mezi nimi

pak probíhá synchronizace vzájemným kopírováním. Metoda `commit()` potvrdí provedené změny, metoda `revert()` je naopak zahodí. Takové uspořádání přináší především větší komfort pro programátora, protože nemálo algoritmů pro zpracování obrazu potřebuje přistupovat k původním hodnotám pixelů a zároveň zapisovat nové.

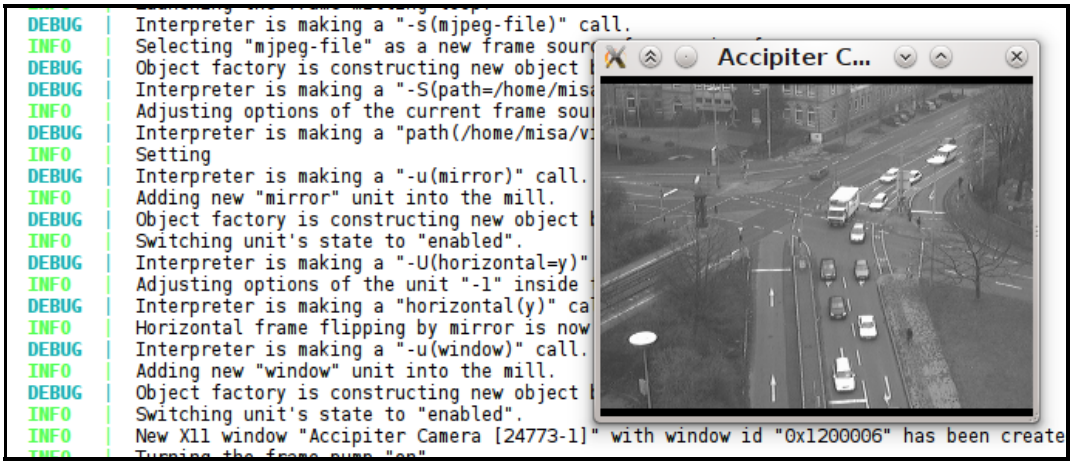
Metoda `commit()` nabízí navíc ještě jednu užitečnou funkcionalitu. Jako parametr totiž přijímá objekt typu `QRegion` [9]. Ten slouží k jednoduchému vymezení oblasti v rámci snímku, ve které mají být změny synchronizovány. Modifikace provedené mimo tuto oblast jsou pak jednoduše zahozeny. S výchozí hodnotou parametru jsou však vždy synchronizovány všechny.

3. Popis uživatelského rozhraní

Vlastní software kamerového systému se skládá ze dvou aplikací. Každé z nich se zde budeme věnovat odděleně.

3.1 Program Accipiter Camera

Tento program slouží jak výkonný prvek softwaru kamerového systému. Každá jeho instance má vždy za úkol zpracovávat v reálném čase aktuální obraz z právě jednoho zdroje. V současné chvíli jím může být buď vzdálená síťová kamera nebo lokální soubor. Program je možné používat samostatně a nebo nepřímo pomocí aplikace Accipiter Manager – viz kapitola 3.2.



Obrázek 3.1: Snímek běžící instance programu Accipiter Camera

3.1.1 Použití a základní volby

Spuštění aplikace

Program lze spustit přes příkazovou řádku zadáním příslušného názvu instalovaného souboru. Spolu s vhodnými volbami lze program konfigurovat k nejrůznějším úkolům. Synopse spouštěcího příkazu je tedy následující:

`accipiter-camera -s <zdroj> <další volby>`

`-s <zdroj>, --source <zdroj>`

Tato povinná volba určuje zdroj, ze kterého mají být jednotlivé snímky pumpou načítány (viz schéma 1.1). Násobné uvedení volby je povoleno, nicméně aplikace použije jen ten zdroj, který byl uveden jako poslední. Momentálně dostupné zdroje jsou tyto: `mjpeg-http`, `mjpeg-file`.

`-S <nastavení>, --source-options <nastavení>`
: Slouží ke změně nastavení aktuálně používaného zdroje.

`-u <jednotka>, --unit <jednotka>`
: Volba vkládá do mlýnu, na konec uspořádané sekvence (opět viz schéma 1.1), novou jednotku s příslušnou funkcionalitou. V současné chvíli je možné vybírat z následujících: `window`, `text-box`, `raw-frame-extractor`, `mjpeg-save`, `mirror`, `color-inverter`, `running-gaussian-average`, `mixture-of-gaussians`.

`-U <nastavení>, --unit-options <nastavení>`
: Použitím této volby lze upravit nastavení poslední vložené jednotky, tj. jednotky přidané na základě nejbližší předchozí volby `-u` nebo `--unit`.

`-Q <nastavení>, --queue-options <nastavení>`
: Pomocí zadaného nastavení lze ovlivnit chování synchronizační fronty mezi pumpou a mlýnem (viz obrázek 1.1).

Konkrétní příklady užití programu se nachází v kapitole 3.1.6.

Konfigurace aplikace za běhu

Úpravy většiny nastavení celé aplikace je možné bezpečně provádět kdykoliv i po jejím spuštění. Není tedy nutné aplikaci restartovat jen např. kvůli změně hodnoty jednoho parametru. Nastavení se provádí zadáváním krátkých příkazů na standardní vstup aplikace. V současné chvíli jsou rozpoznávány tyto příkazy:

`add <jednotka>`
: Přidá do mlýnu novou vybranou jednotku. Jedná se o naprosto stejnou funkcionalitu, jakou poskytuje výše uvedená volba `-u`, resp. `--unit`.

`remove <index>`
: Korektně odstraní z mlýnu jednotku s daným indexem.

`swap <index1> <index2>`
: Umožňuje zaměnit v rámci mlýnu pozice dvou určených jednotek.

`adjust <index> <nastavení>`
: Příslušně v mlýnu upraví nastavení vybrané jednotky.

`list`
: Vypíše na standardní výstup jednoduchý indexovaný seznam s popisy všech jednotek, které jsou v mlýnu momentálně přítomny.

`quit`

Iniciuje proces ukončení celého programu. Jelikož je potřeba korektně uvolnit všechny používané systémové zdroje, nemusí být aplikace ukončena okamžitě a může dojít ke krátké prodlevě. Stejného efektu lze rovněž docílit posláním klasického `SIGINT`, tj. např. stisknutím klávesové zkratky `CTRL+C`.

3.1.2 Formát nastavení

Některé výše uvedené volby a příkazy jsou určeny k úpravě nastavení různých částí aplikace. Jako svůj argument vždy očekávají speciálně formátovaný řetězec, jehož úkolem je popis změn nastavení, které má být provedeno.

Jeho formát je pevně stanoven. Musí být složen z čárkami odděleného seznamu jednotlivých parametrů zapsaných v jednom z následujících tvarů:

- `<parametr>=<hodnota>`
- `<parametr>=(<hodnota1>,<hodnota2>,...)`

V případě, že v rámci hodnoty parametru je potřeba použít některý z řídicích znaků „`\,=()`“, je nutné před ním uvést znak zpětného lomítka „`\`“. Jinak by nastavovací řetězec mohl být aplikací nesprávně interpretován.

Každá hodnota parametru je určitého typu. Jejich závazné formáty jsou uvedeny v následujícím výčtu:

- `řetězec`
  Zapisuje se přirozeně a včetně mezer. Pouze je třeba příslušně ošetřit výskyt zmiňovaných řídicích znaků.
- `přepínač`
  Typ pro určení pravdivostní hodnoty. Možnosti `yes`, `true` nebo `1` jsou interpretovány kladně. Analogicky hodnoty `no`, `false` nebo `0` jsou interpretovány záporně.
- `celé číslo`
  Je možné zapsat jak se znaménkem, tak bez.
- `desetinné číslo`
  Pro oddělení desetinných míst se užívá vždy znaku tečky „`.`“.
- `barva`
  Definici barvy lze zadávat buď v populárním hexadecimálním tvaru `#RRGGBB` nebo přímo pomocí jejího ustáleného anglického názvu – např. `green`, `red`, `pink` atd. Hodnota `transparent` určuje průhlednou barvu.

- `písmo`
  Slouží pro výběr a nastavení písma. V nejjednodušším tvaru lze vybrat pouze jeho rodinu a velikost. Jednotlivé hodnoty se od sebe oddělují znakem svislé čáry „`|`“. Např. hodnota `serif|16` tak značí patkové písmo o velikosti šestnácti typografických bodů.

- `souřadnice`
  Určuje jeden bod v rámci snímku. Souřadnice se zapisují oddělené znakem dvojtečky „`:`“. Hodnota `0:0` náleží jeho levému hornímu rohu.

- `mnohoúhelník`
  Zapisuje se jako seznam `souřadnic` oddělených znakem svislé čáry „`|`“. Např. hodnota `50:30|200:30|200:70|50:70` je obdélníkem.

- `adresa`
  Typ pro určení libovolné URL adresy. Ta je očekávána v obvyklém ustáleném tvaru: `http://host:port/cesta/k/cili?pripadne=parametry`.

- `soubor`
  Umožňuje absolutní nebo relativní určení cesty k jednomu souboru. Pro oddělení adresářů se užívá obvyklého znaku lomítka „`/`“.

- `adresář`
  Obdobně jako `soubor`, ale slouží k určení adresáře.

Konkrétní příklady použití zde popsaných pravidel se nachází v kapitole 3.1.6.

3.1.3 Parametry nastavení zdrojů

Zdroj `mjpeg-http`

- `url` = <`adresa`> výchozí: `http://localhost:80/stream.mjpeg`
  Nastavuje URL adresy vzdálené síťové kamery, ze které mají být jednotlivé snímky čerpány.

Zdroj `mjpeg-file`

- `path` = <`soubor`>
  Definuje cestu k souboru ve formátu MJPEG, ze kterého mají být jednotlivé snímky postupně načítány.

- `loop` = <`přepínač`> výchozí: `no`
  Určuje, zda po načtení posledního snímku má zdroj pokračovat znovu od začátku souboru nebo zda má být uzavřen.

3.1.4 Parametry nastavení jednotek

Obecné parametry všech jednotek

Zcela všechny jednotky, které aplikace může použít, jsou odvozeny od společného základu, jenž poskytuje následující možnosti nastavení:

- `enabled` = <`přepínač`> výchozí: `yes`
  Tento parametr určuje, zda je daná jednotka používána nebo ne. Nepoužívané jednotky jsou mlýnem automaticky vyjmuty z procesu zpracovávání snímků a tedy mají minimální nároky na systémové zdroje.

- `description` = <`řetězec`>
  Parametr je určený k definici krátkého a výstižného popisu funkce dané jednotky. Tento popis může být následně aplikací využit jako součást různých výpisů a hlášení. Všechny odvozené jednotky mají svůj popis již implicitně přednastaven na smysluplnou hodnotu.

Jednotka `window`

Dává aplikaci schopnost okamžitého zobrazení zpracovávaných snímků. V případě zapnutí jednotky parametrem `enabled` je totiž otevřeno jednoduché akcelerované okno s OpenGL kontextem, do kterého jsou jednotlivé snímky postupně vykreslovány. Je samozřejmě možné přidat do mlýnu na různé pozice více těchto jednotek a sledovat tak obraz v jeho různých fázích zpracování. Chování jednotky lze ovlivnit jedním parametrem:

- `keep-aspect-ratio` = <`přepínač`> výchozí: `yes`
  Tento parametr umožňuje určit, zda se vykreslování jednotlivých snímků má řídit poměrem jejich stran nebo ne.

  V případě hodnoty `yes` budou snímky do okna vykreslovány tak, aby nedošlo k jejich deformaci. Tj. pokud to bude nutné, budou k nim nad a pod (resp. nalevo a napravo) přidány černé pruhy vyplňující nevyužité části okna. Naopak při nastavené hodnotě `no` budou snímky škálovány přímo dle rozměrů okna a na vlastní poměr stran snímků nebude brán zřetel.

Jednotka `text-box`

Umožňuje do každého snímku vložit libovolný nápis, ať už statický nebo dynamicky se měnící. V případě, že by při vykreslování nápis přesahoval mimo hranice snímku, bude jeho přesahující část na hranici příslušně oříznuta. Je nutné zmínit, že funkce zalamování řádků není momentálně podporována. Jednotka rozpoznává následující typy parametrů:

- `text` = <`řetězec`>
  Pomocí tohoto parametru je nastavován samotný obsah vykreslovaného textového pole. Mohou se v něm vyskytovat různé substituční řetězce, které budou později na každém snímku nahrazeny příslušnou aktuální hodnotou. Části textu, ve kterých nemá být substituce prováděna, je nutné uzavřít mezi dvojici apostrofů. Přehled všech dostupných substitučních sekvencí spolu s jejich významy lze nalézt v rámci dokumentace funkce `QDateTime::toString()` [9]. Např. hodnota `yyyy-MM-dd hh:mm:ss.zzz` bude do snímků vykreslena jako aktuální datum a čas s přesností na tisíciny sekundy.

- `position` = <`souřadnice`> výchozí: `25:40`
  Slouží k nastavení pozice v rámci jednoho snímku, na které se má při vykreslování nacházet bod účaří písma textového pole, který je nejvíce vlevo. Zjednodušeně řečeno je to přibližně jeho levý dolní roh.

- `font` = <`písmo`> výchozí: `sans-serif|14`
  Parametr umožňuje vybrat a nastavit písmo, kterým bude textové pole do snímků vykresleno.

- `foreground` = <`barva`> výchozí: `white`
  Nastavuje barvu podkladu celého textového pole. Hodnota `transparent` samozřejmě způsobí, že textové pole bude vykresleno bez podkladu.

- `background` = <`barva`> výchozí: `black`
  Obdobně lze zvolit barvu samotného písma.

Jednotka `raw-frame-extractor`

Tato jednotka najde uplatnění především při experimentech s algoritmy detekujících pohyb. Dokáže totiž extrahovat z proudu vybrané snímky a uložit je po jednotlivých souborech v nekomprimovaném formátu PPM. Vzniklé soubory lze poté dále zpracovávat libovolným externím softwarem. Aby jednotka poskytovala korektní výsledky, je většinou nutné upravit v nastavení synchronizační fronty parametr `brutal` na hodnotu `no`. Podrobnosti lze nalézt v kapitole 3.1.5. Chování vlastní jednotky lze ovlivnit takto:

- `at` = <`celé číslo`>
  Naplánuje extrakci snímku se zadaným pořadovým číslem (číslováno od nuly). Parametr je možné použít vícekrát, a tak extrahovat více snímků během jediného spuštění aplikace.

- `output-directory` = <adresář> výchozí: `./`
  Určuje cestu k adresáři, do kterého mají být jednotlivé snímky postupně ukládány. Je zde možné zadat jak absolutní, tak i relativní cestu.

- `filename-template` = <řetězec> výchozí: `%1.ppm`
  Parametr nastavuje šablonu pro názvy vznikajících souborů. Zadaná šablona musí obsahovat speciální žolík `%1`, který bude nahrazen příslušným pořadovým číslem snímku. V případě, že soubor s daným názvem již existuje, bude bez upozornění přepsán.

Jednotka `mjpeg-save`

Jednotka slouží k tvorbě záznamů z proudu snímků. Umožňuje nechat některé snímky ukládat do souborů v MJPEG formátu. Soubory jsou jednotkou vytvářeny automaticky a každý z nich obsahuje vždy jednu hodinu záznamu.

- `save-interval` = <celé číslo> výchozí: `2000`
  Parametr určuje, s jaký časový rozestupem zadaným v milisekundách mají být snímky zaznamenávány. Při hodnotě `0` budou zaznamenány všechny snímky.

- `save-directory` = <adresář> výchozí: `./`
  Nastavuje cestu k adresáři, do kterého mají být všechny soubory se záznamy postupně ukládány.

- `erase-interval` = <celé číslo> výchozí: `24`
  Záznamy starší než zde zadaný počet hodin jsou jednotkou automaticky průběžně mazány. Hodnota `0` tuto funkcionalitu vypne.

Rozšíření jednotky na typ filtr

U některých druhů jednotek má smysl uvažovat nad možností omezení jejich činnosti výlučně jen na předem určený region v rámci jednotlivých snímků. V terminologii aplikace je každá taková jednotka s omezujícím regionem nazývána filtrem. Ve výchozím nastavení je region vždy prázdný. V takovém případě filtr na region nebere ohled a změny jsou provedeny vždy v rámci celého snímku. Konkrétní region je možné nastavit pomocí následujících parametrů:

- `region-union` = <mnohoúhelník>
  Poskytuje možnost, jak zvětšit již definovaný region skrz jeho sjednocení se zadaným mnohoúhelníkem. Je dovoleno násobného použití tohoto parametru. Tak je možné definovat i komplexní nesouvislé regiony, které ale již mohou mít nezanedbatelný dopad na výkon filtru.

- `region-subtraction` = <`mnohoúhelník`>
  Doplňkový parametr jenž naopak umožňuje definovaný region zmenšit a sice odečtením zadaného mnohoúhelníku. Násobné použití tohoto parametru je rovněž povoleno.

Filtr `mirror`

Tento filtr je užitečný např. v situacích, kdy vlastní kamera byla fyzicky instalována jiným způsobem, než bylo při její konstrukci předpokládáno, a poskytuje tak převrácený obraz snímané scény. Filtr umožňuje jednotlivé snímky převrátit zpět do přirozeného stavu. Jsou dostupné dva různé směry převrácení:

- `horizontal` = <`přepínač`> výchozí: `no`
  Tento parametr určuje, zda se jednotlivé snímky mají či nemají „zrcadlově“ převrátit (tj. podle svislé osy, ve směru zleva doprava a naopak).

- `vertical` = <`přepínač`> výchozí: `no`
  Analogický parametr, tentokrát ovládající převracení snímků „vzhůru nohama“ (tj. podle vodorovné osy, ve směru shora dolů a naopak).

Filtr `color-inverter`

Pokud kamera snímá převážně tmavé scény, může být výhodné provést pomocí tohoto filtru barevnou inverzi jednotlivých snímků. Jak píší Gonzalez s Woodsem [1], může tato operace opticky zvýraznit drobné světlé detaily v rámci tmavých oblastí. Filtr neposkytuje žádné další možnosti nastavení.

Rozšíření filtru na typ detektor

Všechny jednotky určené pro detekci pohybu mají některé parametry společné:

- `marker` = <`barva`> výchozí: `magenta`
  Tímto parametrem lze nastavit i barvu, kterou budou, po algoritmem provedené segmentaci snímku, zvýrazněny všechny pixely odpovídající detekovanému popředí. Hodnota `transparent` tuto funkci deaktivuje.

- `eraser` = <`barva`> výchozí: `transparent`
  Obdobně lze nastavit barvu, která bude použita pro odstranění detekovaného pozadí. Použitím hodnoty `transparent` lze toto chování potlačit.

Detektor `running-gaussian-average`

Jedná se o implementaci algoritmu podrobněji popsaného v kapitole 1.5.1. Tento detektor je ovládán následující sadou parametrů:

- `learning-rate` = <`desetinné číslo`> výchozí: `0.01`
  Parametr je definicí konstanty α ve vztahu 1.2.
- `threshold-factor` = <`desetinné číslo`> výchozí: `2.5`
  Tímto parametrem je obdobně definována konstanta k v nerovnici 1.3.

Detektor `mixture-of-gaussians`

Obdobně je v kapitole 1.5.2 popsán robustní algoritmus směsi gaussiánů. Tento detektor je jeho implementací a lze jej konfigurovat v několika ohledech:

- `gaussian-count` = <`celé číslo`> výchozí: `3`
  Stanovuje počet užitých gaussiánů, tj. hodnotu meze m v rámci sumy 1.4.
- `learning-rate` = <`desetinné číslo`> výchozí: `0.005`
  Slouží k úpravě konstanty α užité ve vztazích 1.9 a 1.10.
- `background-boundary` = <`desetinné číslo`> výchozí: `0.6`
  Ovládá nastavení konstanty T ve vztahu 1.7.
- `threshold-factor` = <`desetinné číslo`> výchozí: `2.5`
  Rovněž lze přenastavit hodnotu k užitou v nerovnici 1.8.

3.1.5 Parametry nastavení synchronizační fronty

Chování fronty synchronizující komunikaci mezi pumpou a mlýnem (viz obrázek 1.1) je také konfigurovatelné:

- `capacity` = <`celé číslo`> výchozí: `5`
  Omezuje maximální počet snímků ve frontě. S rostoucí hodnotou může stoupat paměťová náročnost aplikace a případně i míra latence obrazu oproti aktuální situaci na scéně. Nižší počet naopak neumožňuje aplikaci se lépe vypořádat s nerovnoměrným výkonem systémových prostředků.
- `brutal` = <`přepínač`> výchozí: `yes`
  Udává, zda aplikace může v případě přeplnění fronty zahazovat nejstarší nezpracované snímky. Nastavením hodnoty `no` lze toto chování vypnout a vynutit tak zpracování všech snímků. Nicméně v takovém případě nelze zaručit, že aplikace bude operovat vždy s aktuálním obrazem s minimální latencí.

3.1.6 Ukázky použití

Představme si následující ukázkový scénář. K domácímu routeru s IP adresou `192.168.1.120` je připojena obyčejná webová kamera snímající např. prostor před vjezdem do garáže. Dále na routeru běží jednoduchý „streamovací“ server, který naslouchá na portu `8080`. Následujícím příkazem se můžeme ke kameře připojit a nechat obraz zobrazovat do jednoduchého okna:

```
$ accipiter-camera \
   -s mjpeg-http \
   -S 'url=http://192.168.1.120:8080/' \
   -u window
```

Chceme vyzkoušet, jak by pro tuto scénu fungoval některý z algoritmů detekce pohybu. Za tímto účelem tak na standardní vstup spuštěné aplikace zadáme jednoduchý příkaz:

```
add mixture-of-gaussians
```

Vypadá to, že se ale nic nestalo. Nová jednotka totiž byla přidána na úplný konec mlýnu, tj. až za jednotku `window`. Abychom výsledek viděli, musíme ještě tedy změnit jejich pořadí:

```
swap 0 1
list
```

To, že byly jednotky zaměněny můžeme zkontrolovat na výstupu aplikace. Nyní již tedy můžeme vidět, jak algoritmus na dané scéně pracuje. Pokud nejsme s výsledkem zcela spokojeni, můžeme zkusit detektor trochu „poštelovat“:

```
adjust 0 "threshold-factor=3.1,gaussian-count=5"
```

Výsledky detekce se tak mohou o něco vylepšit. Ještě nás může zajímat, jak obraz vlastně vypadá před provedení detekce pohybu. Není žádný problém přidat další okno:

```
add window
swap 1 0
list
```

Nyní jsme s výsledkem již spokojeni. Aplikaci tedy vypneme:

```
quit
```

Výhodou je, že celé nastavení lze při případném opětovném spuštění aplikace přímo zrekonstruovat. Není tak nutné vše znovu manuálně upravovat. V našem případě bychom toho docílili tímto příkazem:

```
$ accipiter-camera \
   -s mjpeg-http \
   -S 'url=http://192.168.1.120:8080/' \
   -u window \
   -u mixture-of-gaussians \
   -U 'threshold-factor=3.1,gaussian-count=5' \
   -u window
```

3.1.7 Instalace aplikace

Program je určen výhradně pro běh pod operačním systémem GNU/Linux. Jeho distribuce probíhá formou zabaleného balíčku se zdrojovými kódy, které je nutné před spuštěním aplikace na daném systému zkompilovat do binární podoby.

K tomu je potřeba mít v systému nainstalovány všechny knihovny, na kterých aplikace závisí. Jedná se především o knihovny Qt ve verzi alespoň 4.7, knihovny Boost ve verzi 1.46, knihovnu SFML v posledním stabilním vydání 1.6 a konečně knihovnu libjpeg od skupiny IJG, a to nejlépe ve verzi 8. Všechny zmíněné závislosti by mělo být snadné získat za pomoci příslušného správce balíčků používané linuxové distribuce. V případě např. distribuce Debian lze k instalaci závislostí využít následující příkaz spuštěný pod uživatelem root:

```
# aptitude install build-essential qt4-qmake libqt4-dev \
libboost-dev libsfml-dev libjpeg-dev
```

Poté by již mělo být možné provést jako běžný uživatel vlastní kompilaci programu. Nejprve je potřeba rozbalit archiv se zdrojovými kódy a v nově vzniklém adresáři vykonat potřebné kompilační úkony. Celý proces lze popsat následujícím sledem příkazů:

```
$ cd /misto/ulozeni/archivu/se/zdrojovymi/kody/
$ tar -xzvf accipiter-camera.tar.gz
$ cd accipiter-camera/
$ qmake
$ make -j 3
```

Pokud sestavení aplikace proběhne v pořádku, vznikne v adresáři `build/` nový spustitelný soubor. Ten je vhodné ručně umístit do některého z adresářů nastavených v proměnné `PATH`. Její hodnotu lze zjistit příkazem:

```
$ echo $PATH
```

Nyní by již mělo být možné aplikaci spustit a používat. Motivační ukázky použití lze nalézt v kapitole 3.1.6.

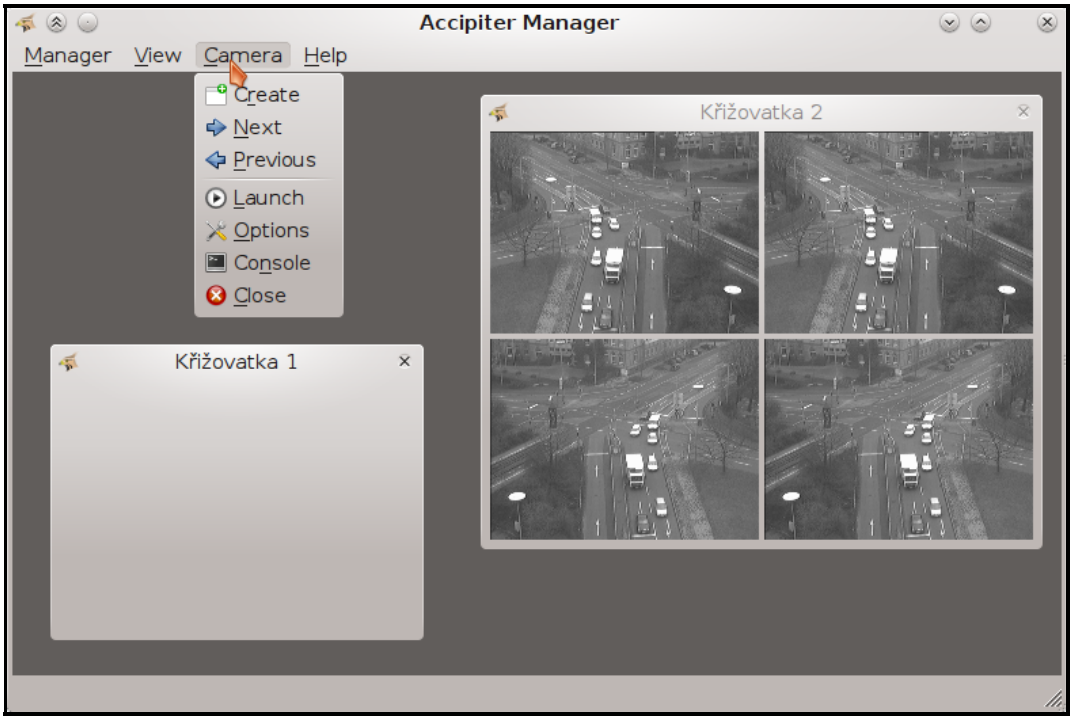
3.2 Program Accipiter Manager

Tento program je určen k organizaci a konfiguraci jednotlivých instancí aplikace Accipiter Camera. Lze jej tak považovat za hlavní uživatelské rozhraní pro správu kamerového systému.

3.2.1 První spuštění

Po prvním spuštění programu se zobrazí prázdné hlavní okno aplikace, protože úvodní nastavení nedefinuje žádné kamery. Veškeré nastavené hodnoty a další konfigurace, které jsou v aplikaci uživatelem v průběhu práce provedeny, jsou při jejím zavření ukládány do konfiguračního souboru. Tím je zajištěno kompletní obnovení stavu aplikace při jejím opětovném spuštění. Relativní cesta jeho umístění vzhledem k domovskému adresáři uživatele, pod kterým je aplikace provozována, je `.config/alcor/accipiter-manager.conf`. V případě, že bude aplikaci nutné přenést a provozovat na jiném systému, stačí pro úplnou obnovu nastavení zazálohovat tento jediný soubor.

3.2.2 Hlavní okno



Obrázek 3.2: Hlavní okno programu Accipiter Manager

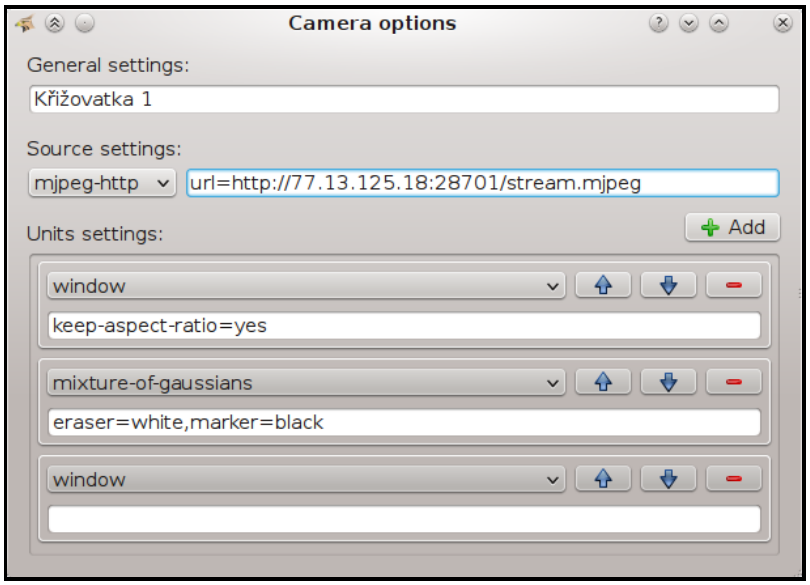
Je jedním z nejdůležitějších částí uživatelského rozhraní. Právě zde jsou zobrazovány přehledně náhledy na všechny nastavené kamery. V rámci hlavního okna

je možné kamery přesouvat nebo měnit jejich velikost. Z technických důvodů však není dovoleno je přes sebe překrývat. Pro lepší pocit uživatele je v aplikaci implementována funkcionalita jejich vzájemného přichytávání.

V horní části okna se nachází menu, které zpřístupňuje všechny dostupné volby. Pravděpodobně nejdůležitější z nich jsou volby v menu `Camera` – viz snímek 3.2. Ty umožňují do systému přidat novou kameru nebo naopak označenou z nich odebrat. Kameru je samozřejmě také možné spustit či vypnout. A konečně jsou zde volby i pro otevření dialogu nastavení či ovládací konzole kamery.

3.2.3 Dialog s nastavením

Podstatnou částí uživatelského rozhraní je právě okno, které umožňuje nastavování jednotlivých parametrů vybrané kamery. Lze jej vyvolat pomocí volby `Options` v menu hlavního okna. Výběr zdroje dat, konfigurace přítomných jednotek či úprava pořadí jejich aplikování je prováděna právě zde. Pro použití nově nastavených voleb může být nutné kameru restartovat.



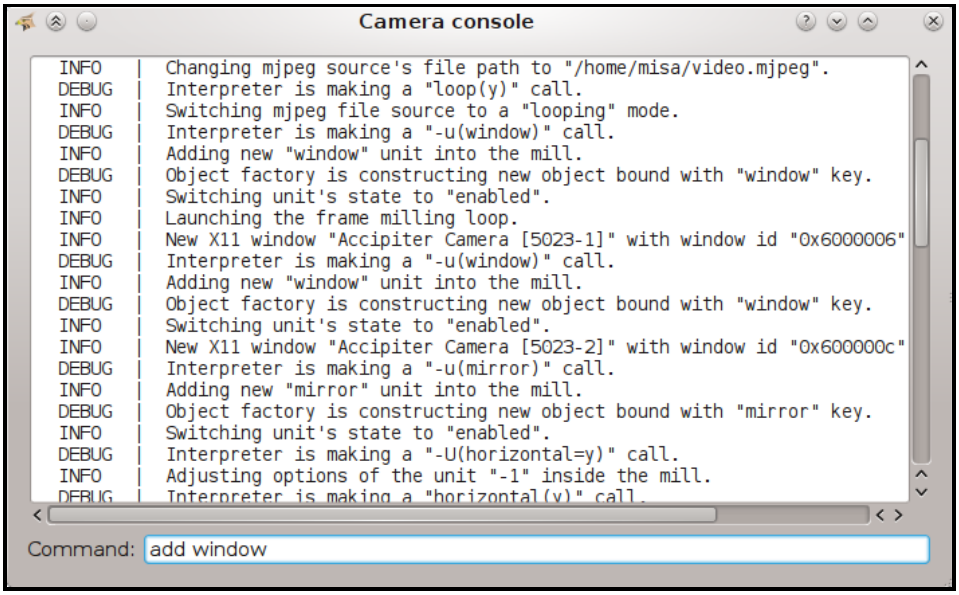
Obrázek 3.3: Pohled na dialog s nastavením kamery

Jak je ze snímku okna 3.3 vidět, ovládání koresponduje přesně schopnostem a možnostem konfigurace programu Accipiter Camera. Jsou používány naprosto stejné parametry a formáty nastavení jako v příslušných částech kapitoly 3.1.

3.2.4 Dialog s konzolí

Okno lze vyvolat pomocí příslušné volby `Console`. Svým charakterem se jedná o obdobu standardního vstupu a standardního výstupu aplikace Accipiter Camera. Lze jej tedy použít i ke stejnému účelu – tj. ke spouštění příkazů měnících

nastavení kamery. Viz příslušná část kapitoly 3.1.1. Na obrázku 3.4 je okno zachyceno v průběhu přidávání nové jednotky `window`.



Obrázek 3.4: Pohled na dialog s konzolí ovládající kameru

3.2.5 Instalace aplikace

Stejně, jako v případě aplikace Accipiter Camera, je i tento program určen pro platformu GNU/Linux, také je distribuován ve formě zabaleného archivu se zdrojovými kódy a rovněž závisí na knihovnách Qt ve verzi alespoň 4.7. Postup instalace je tak analogický tomu, jenž je uveden v části 3.1.7. Z tohoto důvodu je zde detailnější popis postupu vynechán.

Pouze je třeba zdůraznit, že pro správnou funkci programu je z pochopitelných důvodů nezbytné, aby spustitelný soubor `accipiter-camera` byl skutečně umístěn v některé z cest uvedených v proměnné `PATH`.

4. Výsledky

V rámci této kapitoly se budeme zabývat porovnáním představených algoritmů detekce pohybu, a to především z hlediska jejich účinnosti. Za tímto účelem byla s nimi provedena série několika pokusů na předem zvolených testovacích sekvencích, které zachycují různé pohybující se objekty.

Kamerový systém je typicky nasazen v situacích, kdy je třeba sledovat pohyb osob v interiérech budov nebo na venkovních prostranstvích. Ve městech pak může být kamerový systém používán např. k monitorování aktuální dopravní situace a podobně. Z tohoto důvodu byly pro pokusy vybrány dvě příznačné testovací sekvence: pohled na křižovatku s projíždějícími vozidly[1] (viz obrázek 4.2a) a osoba procházející chodbou (obdobně viz obrázek 4.3a).

Při hodnocení účinnosti algoritmů byl využit stejný postup, jaký použili Cheung a Kamath při experimentech s jejich vlastním algoritmem [5]. Z každé testovací sekvence bylo vybráno 5 rovnoměrně rozložených snímků. V nich byly poté ručně označeny všechny oblasti, které zobrazují právě nebo bezprostředně předtím se pohybující objekty. Přitom nebyly brány v úvahu ty, které by ideální algoritmus měl ignorovat. Jedná se např. o ohýbající se větve stromů ve větru, měnící se reklamní plochy apod. Naopak případné vrhané stíny zahrnuty byly, neboť postupy pro potlačení stínů nejsou předmětem tohoto experimentu.

K poměřování algoritmů (resp. jejich nastavení) byla použita následující dvě hlediska, jenž charakterizují, jak moc se výsledek algoritmu na daném snímku přiblížil ručnímu výběru:

$$\text{úspěšnost} = \frac{\text{počet algoritmem správně detekovaných pixelů popředí}}{\text{počet všech pixelů ručně vybraného popředí}}$$

$$\text{přesnost} = \frac{\text{počet algoritmem správně detekovaných pixelů popředí}}{\text{počet všech pixelů detekovaného popředí}}$$

Celková účinnost algoritmu s konkrétním nastavením byla poté vypočtena jako dvojice aritmetických průměrů naměřených úspěšností a přesností napříč všemi vybranými snímky dané testovací sekvence. Dobrý algoritmus (resp. jeho správné nastavení) by měl poskytovat co možná největší úspěšnost, a to bez velkého negativního dopadu na jeho přesnost.

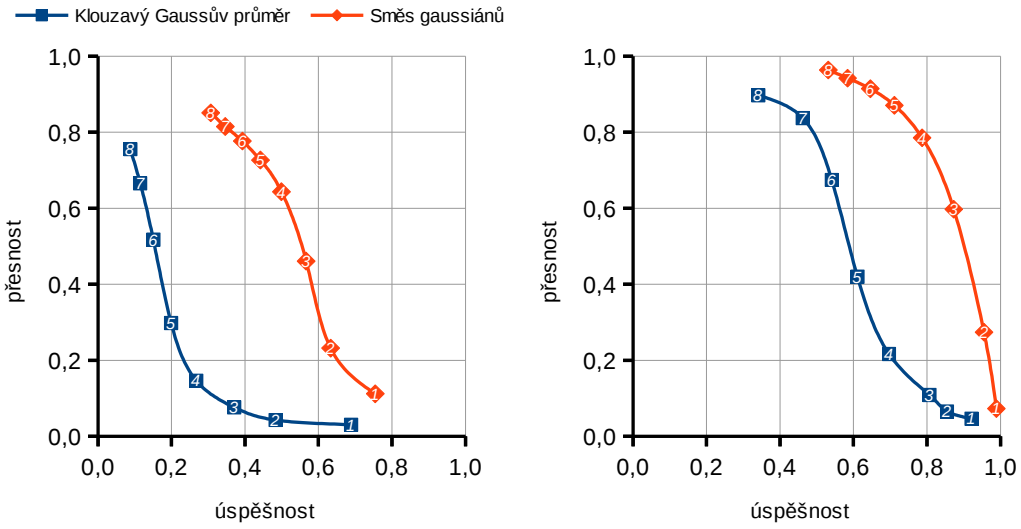
Pro každý algoritmus byla v rámci jedné testovací sekvence provedena série 8 pokusů, kdy v každém z nich byla pozměněna hodnota vstupního parametru, který má na výsledek algoritmu největší vliv. Hodnoty ostatních parametrů zůsta-

[1] Tato sekvence není vlastní výroby, je ale volně dostupná ke stažení pro potřeby výzkumu. Lze ji nalézt na webových stránkách Institutu pro algoritmy a kognitivní systémy na Univerzitě v Karlsruhe, a to konkrétně na adrese http://i21www.ira.uka.de/image_sequences/.

	Sekvence s exteriérem		Sekvence s interiérem	
Klouzavý Gaussův průměr	$\alpha = 0.01$	k	$\alpha = 0.05$	k
Směs gaussiánů	$m = 3$ $\alpha = 0.005$ $T = 0.6$	k	$m = 5$ $\alpha = 0.01$ $T = 0.6$	k

Tabulka 4.1: Udává parametry nastavení algoritmů u provedených pokusů. Hodnota $k = 0.5 \cdot i$, kde $i \in \{1, 2, ..., 8\}$ je pořadové číslo pokusu.

ly pro všechny provedené pokusy konstantní. Podrobnější informace o použitém nastavení lze vyčíst v tabulce 4.1. Naměřené výsledky byly následně zaznamenány do jednoduchého grafu 4.1, kde tak tvoří křivky charakterizující závislost mezi úspěšností a přesností algoritmů na konkrétní testovací sekvenci.



(a) Sekvence s exteriérem

(b) Sekvence s interiérem

Obrázek 4.1: Graf s naměřenými výsledky všech provedených pokusů

Na první pohled je z vykreslených grafů patrné, že oba testované algoritmy podávají u druhé testovací sekvence o něco lepší výsledky. To není překvapivé zjištění. Ve venkovním prostředí panuje řada nepříznivých vlivů, kterých jsme u scén z místností víceméně ušetřeni. Navíc venkovní kamery jsou většinou instalovány tak, že snímají daleko větší oblast. Důsledkem je, že pohybující se objekty jsou v obraze spíše menších rozměrů a algoritmy tak mají o poznání těžší úkol.

To je pravděpodobně i důvodem, proč algoritmus klouzavého Gaussova průměru u sekvence s exteriérem de facto selhal. Naměřená křivka víceméně leží pouze v těch partiích grafu, které reprezentují velmi malou účinnost detekce. V tomto případě směs gaussiánů poskytuje výrazně lepší výsledky.

Výstup klouzavého Gaussova průměru u sekvence s interiérem dopadl o pozná-

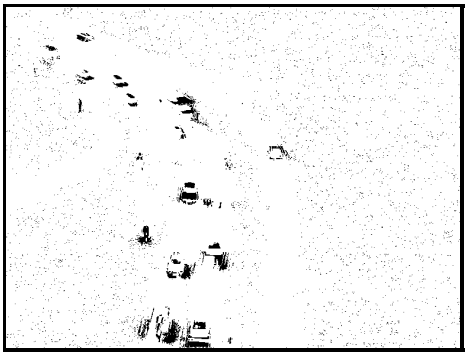
ní lépe. Dalo by se tak uvažovat o jeho případném reálném nasazení. Je vidět, že zatímco v běžných situacích nemusí jednoduchý algoritmus dosahovat požadovaných výsledků, ve specifických případech může být dostatečně účinný. Podstatnou výhodou těchto algoritmů může být obvykle i jejich vyšší rychlost.

Metoda směsi gaussiánů se poté osvědčila i u sekvence s interiérem. Naměřené hodnoty by se s trochou nadsázky daly již považovat za téměř ideální.

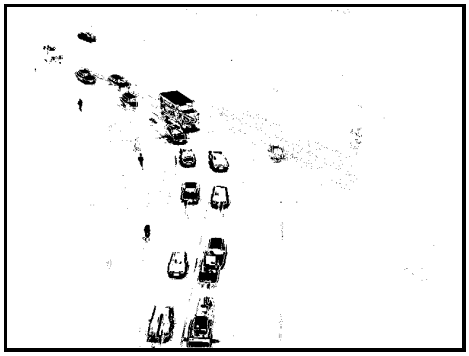
Na obrázcích 4.2 a 4.3 si lze prohlédnout konkrétní výstupy testovaných algoritmů a porovnat je s očekávanými výsledky.

(a) Originální snímek

(b) Ruční vyznačení popředí



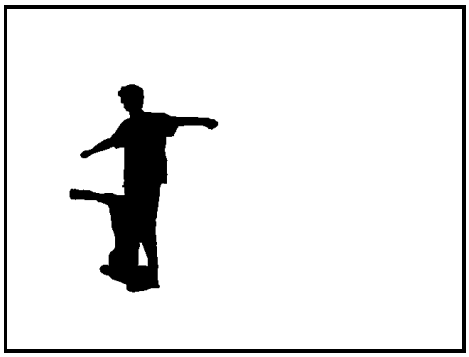
(c) Klouzavý Gaussův průměr



(d) Směs gaussiánů

Obrázek 4.2: Výstupy algoritmů spuštěných na testovací sekvenci s exteriérem

(a) Originální snímek



(b) Ruční vyznačení popředí

(c) Klouzavý Gaussův průměr

(d) Směs gaussiánů

Obrázek 4.3: Obdobně výstupy algoritmů u testovací sekvence s interiérem

Závěr

Cíl, který jsem si předsevzal, se do jisté míry podařilo splnit. Byl připraven software, tvořený dvojicí programů Accipiter Camera a Accipiter Manager, jenž umožňuje realizovat správu kamerového systému v prostředích, pro která byl zamýšlen.

Za poměrně povedený považuji zvlášť návrh aplikace Accipiter Camera. Zdá se být výkonný, lze díky němu aplikaci snadno rozšiřovat o nové funkce a navíc, přestože používá paralelně spouštěná vlákna, umožňuje uživateli v aplikaci za běhu pracovat se sadou plně konfigurovatelných filtrů. Lze tak operativně řešit situace, kdy např. v důsledku malého kontrastu obrazu kamery nepracuje některý z použitých filtrů správně. Díky zvolenému návrhu aplikace, je však snadné připravit filtr pro zvýšení kontrastu, přímo za běhu jej předřadit před postižený filtr a provést případné změny nastavení.

Rovněž se mi podařilo vyzkoušet dvě metody pro detekci pohybujících se objektů ve statické scéně – klouzavý Gaussův průměr a algoritmus směsi gaussiánů. Druhý z nich se, dle provedených experimentů, ukázal být funkčním robustním řešením, které si poradí i z řadou překážek.

Bohužel z nedostatku dalšího času se již nedostalo na implementaci a experimentování s dalšími známými metodami. Dle dostupné literatury může být zajímavým např. nerekurzivní algoritmus Eigen-background či populární Kalmanův filtr. To tak může být alespoň námětem do budoucna. V práci bych rád dál pokračoval, neboť jsem do ní investoval ne málo času a stále v ní cítím další potenciál.

Seznam použité literatury

[1] Gonzalez, Rafael C. a Woods, Richard E. *Digital Image Processing*. Third Edition. Upper Saddle River (New Jersey): Pearson Prentice Hall, 2008. xxii, 954 s. ISBN 978-0-13-505267-9, ISBN 0-13-505267-X.

[2] Stauffer, Chris a Grimson, W.E.L. *Adaptive background mixture models for real-time tracking*. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. 1999, č. 2, s. 246-252.

[3] Gallego Vila, Jaime. *Foreground segmentation and tracking based on foreground and background modeling techniques*. [s.l.]: Universitat Politècnica de Catalunya, 2009. iii, 91 s. Master thesis. Universitat Politècnica de Catalunya, Departament de Teoria del Senyal i Comunicacions.

[4] Piccardi, Massimo. *Background subtraction techniques: a review* [online] [prezentace]. 2004-04-15, [cit. 2011-11-20]. Dostupné z: http://www-staff.it.uts.edu.au/~massimo/BackgroundSubtractionReview-Piccardi.pdf.

[5] Cheung, Sen-Ching S. a Kamath, Chandrika. *Robust Background Subtraction with Foreground Validation for Urban Traffic Video*. EURASIP Journal on Applied Signal Processing. 2005, č. 14, s. 2330-2340.

[6] Šmirg, Ondřej, et al. *A methods for recognition and separation human body*. Elektrorevue [online]. 2010-11-17, [cit. 2011-11-19]. Dostupné z: http://www.elektrorevue.cz/en/articles/analogue-technics/0/a-methods-for-recognition-and-separation-human-body-1/. ISSN 1213-1539.

[7] Jůza, Michal. *Detection of the Unusual Behavior from Video Sequences*. Praha, 2004. 57 s. Diploma thesis. Czech Technical University in Prague, Faculty of Electrical Engineering, Department of Cybernetics.

[8] Molkentin, Daniel. *The Book of Qt 4: The Art of Building Qt Applications*. San Francisco: No Starch Press, 2007. 440 s. ISBN 978-1-59327-147-3.

[9] *Qt Reference Documentation*. Qt 4.7 [online]. Nokia Corporation, 2011 [cit. 2011-11-23]. Dostupné z: http://doc.qt.nokia.com/4.7/index.html.

Seznam obrázků

A. Obsah přiloženého DVD

- text práce v elektronické podobě
- balíček se zdrojovým kódem aplikace Accipiter Camera
- balíček se zdrojovým kódem aplikace Accipiter Manager
- generovaná referenční dokumentace k oddělené knihovně
- přesné výsledky měření provedených experimentů,
  spolu s použitými testovacími sekvencemi
- jednoduchá aplikace pro streamování obrazu webových kamer,
  pro platformu GNU/Linux, spolu s uživatelskou dokumentací

B. Seznam zkratek

CCTV	Closed-circuit television
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
RTP	Real-time Transport Protocol
HTTP	Hypertext Transfer Protocol
JPEG	Joint Photographic Experts Group
MJPEG	Motion JPEG
IJG	Independent JPEG Group
SFML	Simple and Fast Multimedia Library
ASCII	American Standard Code for Information Interchange
GNU	GNU's Not Unix!