Algoritmus sledování objektů pro funkční vzor sledovacího systému

Technická zpráva - FIT - VG20102015006 – 2014 - 04

Ing. Filip Orság, Ph.D.

Fakulta informačních technologií, Vysoké učení technické v Brně

12. ledna 2015

Abstrakt

Tato technická zpráva shrnuje základní vlastnosti algoritmu použitého v systému pro sledování objektů se zaměřením na aplikaci v bezpečnostních systémech. Konkrétně je uvedeno řešení digitální stabilizace založené na optickém toku a sledování objektu založené na částicových filtrech. Vybrané řešení je optimalizováno z hlediska kvality a rychlosti nasazením grafických akcelerátorů pro účely zrychlení výpočtu algoritmů. Algoritmus je robustní a odolný vůči pohybu kamery. Poradí si dobře s monochromatickými záběry a je odolný vůči částečnému a úplnému zmizení sledovaného objektu.

Obsah

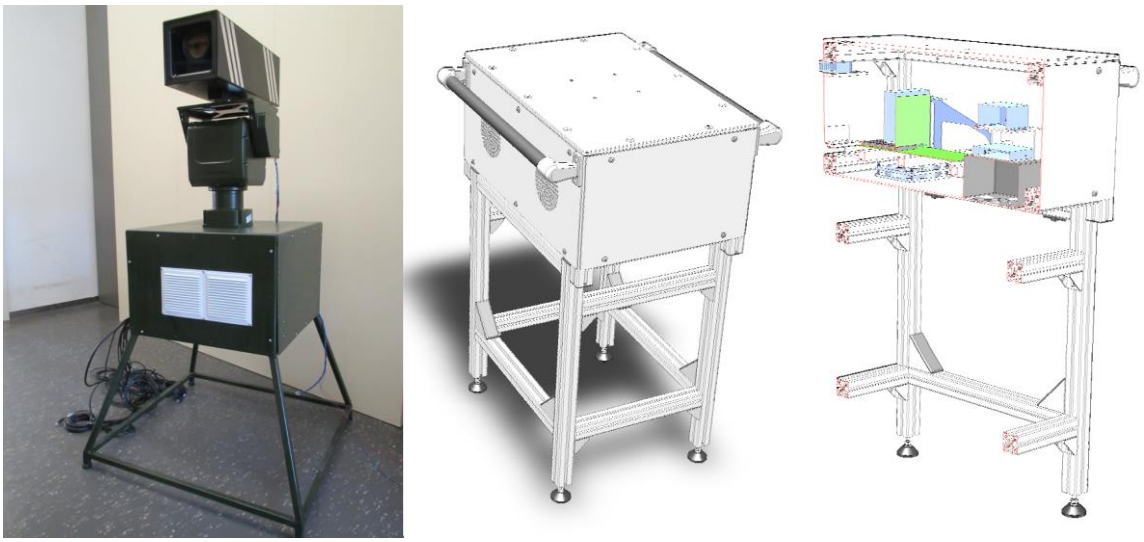
1 Úvod

Cílem této práce je ukázat postup první fáze vývoje funkčního vzoru zařízení pro sledování vzdálených objektů. Cílovým uživatelem v našem případě bude policie nebo armáda, která má za úkol strážit bezpečnost. Požadavkem policie je vlastnit systém, který by byl schopen, po označení podezřelé osoby ve videu, tuto autonomně sledovat prostřednictvím systému PTZ – tedy otočné kamery s proměnnou ohniskovou vzdáleností. Podobně i z řad armády existují požadavky na pasivní systém schopný sledovat vybraný cíl. Výsledkem by měl být návrh sestavy kamery, manipulátoru, výpočetního hardware a software tak, aby byly splněny všechny požadavky na takový systém kladené. Spolu s požadavky postupně vyplynou i omezení daná především úrovní hardware a dostupností algoritmů. Vývoj hardware byl předmětem technické zprávy FIT-VG20102015006-2013-03, tento dokument se zaměřuje na řídicí software, bez něhož by vyvinutý hardware nepřinášel žádné výsledky. Požadavky na algoritmus relativně dobře specifikovatelné na základě analýzy uživatelských scénářů.

Požadavky na software lze specifikovat takto:

- integrované ovládání kamery,
- možnost autonomního sledování vybraného objektu:
    - sledování vybraného pohybujícího se objektu,
    - odolnost vůči překrytí objektu jinými objekty,
    - částečná odolnost vůči úplnému zmizení objektu,
    - odolnost vůči jasovým změnám a absenci barvy (monochromatický záznam),
    - sledování i při pohybu manipulátoru s kamerou,
- integrovaná digitální stabilizace obrazu,
- možnost záznamu videa na disk.

Pouze pro připomenutí uvádím snímek funkčního vzoru ve své první verzi a nový návrh nosoné platformy, který vznikl na konci roku 2014 a nyní je vyráběn (leden 2015).



Obrázek 1.1: Funkční vzor systému pro sledování (vlevo, verze 2), návrh nové nosné platformy (uprostřed a vpravo, verze 3, aktuální pro rok 2015).

2 Digitální stabilizace obrazu a sledování objektů

V této kapitole jsou definovány základní pojmy, s nimiž budeme dále pracovat, popsáno je teoretické pozadí problematiky digitální stabilizace obrazu a sledování objektů.

2.1 Základní definice

Nejdříve definujme základní prvky, s nimiž budeme pracovat (video, sledovaný objekt a také úlohu, kterou řešíme), z pohledu formálního. Kromě toho také popíšeme další teoretické základy (například homografii).

2.1.1 Posloupnost snímků

Posloupnost snímků (*video*) $\mathcal{V}$ lze vnímat jako posloupnost po sobě jdoucích snímků $\boldsymbol{I}_n$ počínaje časem $n = 0$ a konče posledním okamžikem snímání nebo posledním snímkem záznamu $n = N_{max}$ (tato hodnota nebývá často předem definována v případě, kdy zpracováváme kontinuální data – například tok dat z kamer – pro naše další úvahy není tato hodnota podstatná). Definici lze tedy vyjádřit jako

$$\mathcal{V} = \{\boldsymbol{I}_n\}, 0 \leq n \leq N_{max}. \tag{1}$$

Tato definice videa je spíše teoretická, protože v praxi se málokdy pracuje s celým videem. Častější jsou případy, kdy je využit pouze jeden snímek (tzv. aktuální) nebo posloupnost několika snímků předcházejících aktuálnímu snímku především proto, že časově-prostorové nároky na zpracování těchto dat by byly neúměrně velké.

Snímek $\boldsymbol{I}_n$ má konečnou velikost, je tedy velikostně omezen a jakožto popisující vlastnost rozměru snímku $\boldsymbol{I}_n$ definujeme jeho šířku W (maximální počet sloupců) a výšku H (maximální počet řádků). Šířka a výška snímku jsou závislé na vlastnostech čipu snímajícího snímky nebo na vlastnostech vstupního videa, které poskytuje tuto posloupnost. Samotný snímek $\boldsymbol{I}_n$ pak lze definovat několika způsoby, například v závislosti na barevném prostoru zdroje snímků jako množinu prvků:

$$\boldsymbol{I}_n = \{(x, y, \boldsymbol{v}) | x = 0,1, \dots, W-1 \wedge y = 0,1, \dots, H-1 \wedge \boldsymbol{v} \in V^C\}, \tag{2}$$

kde x udává sloupec snímku, y je řádek snímku a oba společně definují bod (x, y) ve snímku. Vektor $\boldsymbol{v}$ udává jas, barvu nebo jiný údaj reprezentující hodnotu daného bodu v podobě prostoru V^C, kde C udává rozměr barevného prostoru a V je interval závislý na rozlišení jednotlivých složek barevného prostoru a celkově na dalším zpracování obrazu. Nejčastěji jde o podmnožinu celých čísel, ale může jít i o podmnožinu reálných čísel a obecně pro každý z C rozměrů může jít o jiný interval, což však nebývá zvykem (například pro monochromatický snímek s rozlišením 8 bitů na jeden bod bude $C = 1$, $V = \langle 0,255 \rangle$ a $\boldsymbol{v}$ nebude vektor, ale skalár, v případě třísložkového 8bitového barevného RGB obrazu bude $C = 3$ a $V = \langle 0,255 \rangle$). Často se prostor hodnot vektoru $\boldsymbol{v}$ různě modifikuje v závislosti na dalším zpracování.

Další možností, jak definovat snímek, je maticový zápis, kdy snímek $\boldsymbol{I}_n$ lze definovat jako 2rozměrnou matici o rozměru $W \times H$, kde jednotlivé prvky této matice jsou vektory $\boldsymbol{v}_{x,y}$ závislé na poloze (x, y) bodu jako

$$\boldsymbol{I}_n = \begin{bmatrix} \boldsymbol{v}_{0,0} & \boldsymbol{v}_{1,0} & \cdots & \boldsymbol{v}_{W-1,0} \\ \boldsymbol{v}_{0,1} & \boldsymbol{v}_{1,1} & \cdots & \boldsymbol{v}_{W-1,1} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{v}_{0,H-1} & \boldsymbol{v}_{1,H-1} & \cdots & \boldsymbol{v}_{W-1,H-1} \end{bmatrix}, \boldsymbol{v}_{x,y} \in V^C. \tag{3}$$

Poslední možností definice snímku, kterou uvádíme, popisuje snímek jako funkci dvou (obvykle diskrétních) proměnných x a y s tím, že definiční obor $\boldsymbol{D}$ této funkce je dán intervaly $\boldsymbol{D} = \langle 0, W-1\rangle \times \langle 0, H-1\rangle$ a obor hodnot je dán jako již zmíněná množina $\boldsymbol{H} = V^C$, tedy

$$\boldsymbol{I}_n(x,y) = \boldsymbol{v}, (x,y) \in \boldsymbol{D}, \boldsymbol{I}_n(x,y) = \boldsymbol{v} \in \boldsymbol{H}. \tag{4}$$

Hodnota funkce $\boldsymbol{I}_n(x,y)$ je označována jako intenzita jasu nebo barva daného bodu (pixelu). Běžně platí, že v závislosti na rozlišení lze přímo definovat obor hodnot v podobě intervalu, obvykle $\langle 0{,}255\rangle$ pro monochromatické zdroje snímků, ale například pro 12bitové A/D převodníky by tento interval byl $\langle 0{,}4096\rangle$. Jsou však i případy, kdy je vhodné, aby byl tento interval normalizován – transformován například na interval $\langle -1{,}1\rangle$. Pro barevné zdroje dat se pak obor hodnot rozšiřuje na více rozměrů s tím, že každý rozměr je definován opět jako interval pro každou barevnou složku zvlášť. Podobně lze upravovat i definiční obor, jehož rozsah je však pevně dán rozměry snímače. Pokud zvýšíme matematickými prostředky počet bodů snímače (tedy definiční obor funkce), pak se bavíme o tzv. *subpixelové přesnosti* a musíme matematicky dopočítat hodnoty barevných složek nebo jasu například interpolací. Pro naše účely budeme téměř výhradně pracovat se snímkem jako s funkcí dvou proměnných s tím, že obor hodnot bude jednorozměrná množina (většinou podmnožina celých čísel) – půjde tedy o monochromatické snímky a budeme pracovat se skalární hodnotou jasu.

Pro zjednodušení dalších zápisů budeme implicitně předpokládat, že snímek je funkcí dvou proměnných a označení $\boldsymbol{I}_n(x,y)$ zjednodušíme na $\boldsymbol{I}_n$ tam, kde to bude přispívat přehlednosti textu. Kromě toho také zavedeme pojem iniciální snímek (*initial image*), který je prvním zpracovávaným snímkem, a bude označován jako $\boldsymbol{I}_0$. Často budeme pracovat také s menší oblastí celého snímku. Tuto oblast definujeme stejně jako snímek s tím rozdílem, že rozměry vybrané oblasti budou jiné.

2.1.2 Sledovaný objekt a sledování změny polohy

Na sledovaný objekt můžeme pohlížet obecně jako na množinu $\mathcal{T}_0$ (*Target*) obsahující M bodů $\boldsymbol{t}_{m,0}$ z iniciálního snímku $\boldsymbol{I}_0$ takových, že o těchto bodech lze říci, že náleží sledovanému objektu a zároveň platí, že libovolný bod snímku $\boldsymbol{I}_0$ je definován souřadnicemi x a y, platí tedy, že $\boldsymbol{t}_{m,0} = (x,y)^T$:

$$\mathcal{T}_0 = \left\{\boldsymbol{t}_{m,0} \mid \boldsymbol{t}_{m,0} = (x,y)^T \wedge \boldsymbol{t}_{m,0} \text{ je součástí sledovaného objektu} \wedge (x,y) \in \boldsymbol{D}\right\}, \tag{5}$$

kde $\boldsymbol{D}$ je definiční obor funkce (snímku) $\boldsymbol{I}_0$ definovaný výše a $m = 1, \dots, M$. Body $(x,y)^T$ budeme uvádět jako sloupcové vektory především proto, že v literatuře tento způsob zápisu nalezneme častěji než zápis v podobě řádkového vektoru. Záměna sloupcového/řádkového vektoru vede k chybným výsledkům, proto je nutné sledovat, jak přesně je v citovaném zdroji definován bod.

Cílem sledování je nalézt všechny body množiny $\mathcal{T}_0$ nebo podmnožinu bodů z této množiny ve snímcích následujících po snímku původním $\boldsymbol{I}_0$ (tedy ve snímcích $\boldsymbol{I}_1, \boldsymbol{I}_2, \dots$), přičemž v tomto místě se budou jednotlivé algoritmy lišit v závislosti na principu, který daný algoritmus využívá. Většina algoritmů je tolerantní a nevyžaduje striktní nalezení všech bodů v následujících snímcích, ale pouze podmnožiny bodů. Zobecněme definici množiny bodů na libovolný snímek $\boldsymbol{I}_n$ takto:

$$\mathcal{T}_n = \left\{\boldsymbol{t}_{m,n} \mid \boldsymbol{t}_{m,n} = (x,y)^T \wedge \boldsymbol{t}_{m,n} \text{ je součástí sledovaného objektu} \wedge (x,y) \in \boldsymbol{D}\right\}, \tag{6}$$
$$m = 1, \dots, M_T \wedge n = 0,1, \dots .$$

Úkol sledování objektu lze zobecnit tak, že hledání rozšíříme na hledání bodů z množiny bodů $\mathcal{T}_i$ ze snímku $\boldsymbol{I}_i$ v libovolném následujícím snímku $\boldsymbol{I}_j$. Formálněji pak lze říci, že hledáme transformaci (matici) $\boldsymbol{H}_{m,i,j}$ ze snímku $\boldsymbol{I}_i$ do snímku $\boldsymbol{I}_j$, pro niž platí, že její aplikace na každý bod $\boldsymbol{t}_{m,i}$ z množiny $\mathcal{T}_i$ (tedy vynásobení bodu $\boldsymbol{t}_{m,i}$ touto maticí) určí místo – bod $\boldsymbol{t}_{m,j}$, kde se původní bod nachází ve snímku $\boldsymbol{I}_j$. Výsledkem transformace

jednotlivých bodů je pak množina $\mathcal{T}_j$, která reprezentuje původní množinu $\mathcal{T}_i$ ze snímku $\boldsymbol{I}_i$ ve snímku $\boldsymbol{I}_j$ a teoreticky je tato transformace pro každý bod jiná, pokud nepůjde například o mezisnímkový rozdíl způsobený takovým pohybem kamery, který způsobí přibližně shodnou změnu polohy všech bodů. Platí tedy:

$$\begin{gathered}\boldsymbol{t}_{m,j} = \boldsymbol{H}_{m,i,j}\boldsymbol{t}_{m,i} \wedge i < j, \\ \boldsymbol{t}_{m,i} \in \mathcal{T}_i, \boldsymbol{t}_{m,i} = (x, y)^T, \\ \boldsymbol{t}_{m,j} \in \mathcal{T}_j, \boldsymbol{t}_{m,j} = (x, y)^T.\end{gathered} \tag{7}$$

Otočení a změnu měřítka lze provést přímo s výše definovanými body a maticí transformace, protože se jedná o lineární transformace. Abychom však mohli obecně ošetřit všechny varianty transformací (tj. především posunutí, které nás z tohoto hlediska zajímá nejvíce), musíme rozšířit body o jednu složku w (tzv. *homogenní složka*, která bude nenulová, nejčastěji se volí $w = 1$) a původní matici nahradit tzv. maticí *afinní transformace* (více viz [1]), která bude mít různý tvar v závislosti na konkrétní transformaci, kterou bude popisovat. Body sledovaného objektu, tedy body množiny $\mathcal{T}_i$, pak budeme zapisovat jako $\boldsymbol{t}_{m,i} = (x, y, w)^T$. Matice $\boldsymbol{H}_{m,i,j}$ bude mít rozměr 3×3 a předchozí rovnici zapíšeme jako

$$\begin{gathered}\boldsymbol{t}_{m,j} = \boldsymbol{H}_{m,i,j}\boldsymbol{t}_{m,i} \wedge i < j, \\ \boldsymbol{t}_{m,i} \in \mathcal{T}_i, \boldsymbol{t}_{m,i} = (x, y, w)^T, \\ \boldsymbol{t}_{m,j} \in \mathcal{T}_j, \boldsymbol{t}_{m,j} = (x, y, w)^T.\end{gathered} \tag{8}$$

Matici $\boldsymbol{H}_{m,i,j}$ lze definovat různými způsoby v závislosti na tom, jaký druh transformace bude popisovat a v závislosti na tom, z jakého zdroje informací budeme vycházet. Body definujeme jako sloupcový vektor. Transformace, které lze popsat maticí afinní transformace, jsou rotace, změna měřítka, posunutí (translace), zkosení a složené transformace (především projekce) [1].

Chceme-li tedy vyjádřit, jakým způsobem se změnila poloha bodu $\boldsymbol{t}_{m,i}$ z množiny bodů $\mathcal{T}_i$ ze snímku $\boldsymbol{I}_i$ v některém z následujících snímků $\boldsymbol{I}_j$, musíme stanovit matici transformace, která tuto změnu popíše. Máme vytyčeny body zájmu (množina $\mathcal{T}_i$) a definován úkol (nalezení matice $\boldsymbol{H}_{m,i,j}$ transformace bodu $\boldsymbol{t}_{m,i} \in \mathcal{T}_i$ ze snímku $\boldsymbol{I}_i$ do snímku $\boldsymbol{I}_j$), který potřebujeme vyřešit. Obecně platí, že každý bod z množiny $\mathcal{T}_i$ může být transformován jinak, proto má každý bod svou „vlastní“ matici. Optimální řešení transformuje body množiny $\mathcal{T}_i$ na body množiny $\mathcal{T}_j$ tak, že chyba určení polohy ε bodu $\boldsymbol{t}_{n,j}$ ve snímku $\boldsymbol{I}_j$ je minimální. Jde tedy o problém optimalizační, kdy se snažíme nalézt parametry matice prostřednictvím měření chyby, která je funkcí všech vstupních parametrů (body objektu i nalezené body, oba obrazy a transformační matice):

$$\varepsilon_n = f\left(\boldsymbol{I}_i, \boldsymbol{I}_j, \boldsymbol{t}_{m,i}, \boldsymbol{t}_{m,j}, \boldsymbol{H}_{m,i,j}\right), m = 1,2, \dots, M_T. \tag{9}$$

Úkolem algoritmu je tedy nalézt takovou metriku (a tím specifikovat chybu ε), která umožní stanovit řešení (tj. nalézt matici transformace), které je pro daný bod nejlepší.

Takto obecně definovaný úkol není snadno řešitelný, proto naprostá většina metod celé zadání zjednodušuje různými způsoby. Například lze omezit transformační matici $\boldsymbol{H}_{m,i,j}$ na translační pohyb, stanovit příslušnost této matice každému bodu náležejícímu objektu (tedy všechny body se transformují stejně), apod. Tato zjednodušení obvykle vnášejí do sledování objektu chyby, které jsou způsobeny tím, že transformace nejsou lineární (posunutí) nebo stejné pro všechny body (přiblížení objektu = změna měřítka = změna polohy každého bodu je rozdílná). Důležité pro danou konkrétní aplikaci je stanovit limity, které omezí úlohu a zároveň umožní získat požadované výsledky.

2.1.3 Homografie

Kromě již zmíněných typů transformací existuje ještě další důležitý typ transformace, jímž se budeme zabývat a tím je projektivní transformace [4]. Projektivní transformace je invertibilní transformace mezi dvěma projektivními perspektivami. Jde o komplexní složenou transformaci a zásadní vlastností je to, že tento typ transformace zachovává přímky – tedy zachovává kolinearitu. Tento typ transformace se také nazývá projektivita nebo homografie. Homografie vyjadřuje, jak se mění vjem pozorovaného předmětu, pokud se mění pozice a/nebo úhel pohledu pozorovatele. Typické využití je například ve stereometrii, kde je projektivní transformace a její vlastnosti využita pro kalibraci kamer. Projektivní transformaci však lze využít například i jako mezisnímkovou analýzu obrazu, jejímž výsledkem může být algoritmus stabilizace obrazu. Matice projektivní transformace (homografie) je definována jako

$$\boldsymbol{H} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}. \tag{10}$$

Matice má rozměr 3×3 a obsahuje tedy 9 prvků, které nejsou blíže specifikovány, avšak existuje vztah mezi touto maticí a transformačními maticemi, které jsme již zmínili v předchozí podkapitole. Zvláštní význam má prvek h_{33}, neboť může měnit globálně měřítko v případě, kdy není roven 1, což ve většině případů není žádoucí. Skládáním transformací – tedy násobením jednotlivých transformačních matic – dosáhneme v podstatě libovolné transformace obrazu a lze si představit, že prvky matice homografie odpovídají prvkům jednotlivých transformačních matic (například bude-li prvek h_{11} větší než 1, bude docházet ke změně měřítka ve směru osy x, prvek h_{13} bude určovat velikost translace ve směru osy x v případě sloupcových vektorů definujících body, apod.). V případech, kdy je homografie využívána, však nejde o konkrétní typ transformace (rotace, posunutí), ale spíše o vyjádření vztahu dvou různých obrazů pomocí této matice.

Budeme-li vycházet z předchozích definic, pak stále platí uvedené rovnice tak, jak byly uvedeny, tedy transformace bodu $\boldsymbol{t}_{m,i}$ ze snímku $\boldsymbol{I}_i$ na bod $\boldsymbol{t}_{m,j}$ do snímku $\boldsymbol{I}_j$ lze vypočítat jako $\boldsymbol{t}_{m,j} = \boldsymbol{H}_{m,i,j}\boldsymbol{t}_{m,i}$ [5]. Za povšimnutí stojí fakt, že $\boldsymbol{t}_{m,j}$ a $\boldsymbol{H}_{m,i,j}\boldsymbol{t}_{m,i}$ si číselně neodpovídají, neboť se liší v měřítku daném homogenní složkou $w_{m,j}$. Protože platí, že dva body a a b jsou si rovny tehdy, existuje-li nenulový skalár λ tak, že $a = \lambda b$ (tuto rovnost bodů pak označujeme $a \sim b$), využijeme tohoto faktu a odstraníme tak nežádoucí homogenní faktor.

Nechť je bod $\boldsymbol{t}_{m,i} = \left(x_{m,i}, y_{m,i}, 1\right)^T$, bod $\boldsymbol{t}_{m,j} = \left(w_{m,j}x_{m,j}, w_{m,j}y_{m,j}, w_{m,j}\right)^T$, a platí

$$\boldsymbol{t}_{m,j} = \boldsymbol{H}_{m,i,j}\boldsymbol{t}_{m,i}, \tag{11}$$

pak můžeme psát

$$\begin{bmatrix} w_{m,j}x_{m,j} \\ w_{m,j}y_{m,j} \\ w_{m,j} \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_{m,i} \\ y_{m,i} \\ 1 \end{bmatrix}, \tag{12}$$

Vytknutím homogenní složky $w_{m,j}$ a rozepsáním na jednotlivé rovnice získáme soustavu rovnic, kterou dalšími úpravami upravíme na dvojici rovnic

$$\begin{aligned} -h_{11}x_{m,i} - h_{12}y_{m,i} - h_{13} + x_{m,j}(h_{31}x_{m,i} + h_{32}y_{m,i} + h_{33}) = 0, \\ -h_{21}x_{m,i} - h_{22}y_{m,i} - h_{23} + y_{m,j}(h_{31}x_{m,i} + h_{32}y_{m,i} + h_{33}) = 0, \end{aligned} \tag{13}$$

což je soustava dvou rovnic o devíti neznámých, kterou je možné vyřešit, pokud známe minimálně čtveřici korespondujících bodů. Čtveřice korespondujících bodů generuje soustavu 8 lineárně nezávislých rovnic s

9 neznámými, přičemž jde o homogenní soustavu rovnic, lze tedy tvrdit, že má řešení. Podmínkou řešitelnosti je, že žádné 3 body z minimálně 4 bodů neleží na jedné přímce. Výsledkem jsou pak prvky matice homografie $\boldsymbol{H}_{m,i,j}$, která reprezentuje vztah buď mezi dvěma snímky nebo, pro náš případ vhodnější interpretaci, vztah bodů náležejících objektu definovaného ve snímku $\boldsymbol{I}_i$ množinou bodů $\mathcal{T}_i$ a ve snímku $\boldsymbol{I}_j$ množinou bodů $\mathcal{T}_j$. Tímto způsobem jsme schopni vypočítat, jakým způsobem se vizuálně změnil objekt mezi dvěma snímky. Matice homografie nám dává možnost sledování vývoje a případné korekce/reakce na změny, které nejsou přijatelné (například v důsledku chybného nalezení korespondujících bodů bude některá z hodnot matice homografie statisticky mimo běžné meze a to značí buď chybu, nebo například zakrytí objektu).

2.1.4 Sledování změny vzájemné polohy snímků

Analýza změny vzájemné polohy po sobě jdoucích snímků nám poskytuje velmi cennou informaci například v podobě matice homografie, která nás informuje o tom, jak se dva snímky vůči sobě posunuly, pootočily nebo jinak transformovaly. Tato informace je nezbytná v případě, že hodláme využít dostupných informací a aplikovat je například pro účely stabilizace obrazu (viz následující kapitola). Analýza mezisnímkové změny v podstatě spočívá v nalezení nové polohy vybraných bodů ze snímku iniciálního ve snímku aktuálním (nebo jiném, který následuje, ale i předchází, po snímku iniciálním). Řečeno jinak – sledování změny vzájemné polohy snímku je definovatelné stejně jako sledování objektu pouze s tím rozdílem, že množina bodů cíle $\mathcal{T}_i$ ve snímku $\boldsymbol{I}_i$ obsahuje vhodně vybrané body, které musí dostatečně reprezentovat celý obraz, nikoliv pouze jeho část, jak je tomu v případě sledování objektu. Typické je, vybrat například rohy obrazu nebo uniformě rozmístěnou mřížku bodů, apod. Pro různé situace mohou přinášet dobré výsledky různá rozmístění bodů. Změnu vzájemné polohy snímků $\boldsymbol{I}_i$ a $\boldsymbol{I}_j$ pak přesně vyjádříme nalezením matice homografie $\boldsymbol{H}_{m,i,j}$. Samozřejmě k tomu, abychom byli schopni matici homografie číselně vyjádřit, potřebujeme znát polohu dostatečného počtu korespondujících bodů a to jak pro úlohu sledování objektu, tak i pro úlohu sledování změny vzájemné polohy dvou snímků. Známe-li matici mezisnímkové transformace, můžeme eliminovat nežádoucí mezisnímkový pohyb například zapojením transformační matice do řetězce zpracování bodů při sledování objektů nebo i přímo aplikací této matice na jednotlivé body snímku, čímž dosáhneme stabilizace obrazu. Úkolem pak zůstává nalezení korespondujících bodů, čehož lze dosáhnout různými metodami, s nimiž se seznámíme v dalších kapitolách.

2.2 Digitální stabilizace obrazu

Využívání kamerových systémů v různých konkrétních případech užití s sebou přináší mnoho problémů a jedním z nich je i roztřesení, rozhoupání, rotace a jiné nechtěné transformace obrazu. Nechtěné transformace obrazu způsobené mechanickými pohyby celé snímací soustavy nebo jejími částmi lze řešit buď mechanicky, nebo digitálně. Konkrétním příkladem nechtěného pohybu může být například situace, kdy na vysoký stožár umístíme kameru s velkou ohniskovou vzdáleností. V důsledku chvění stožáru, kterému lze těžko zabránit, dochází k malým výkyvům soustavy. V případě, kdy je zorné pole kamery například 1° a dojde-li k výkyvu stožáru o úhel 0,2°, způsobí to nemalé problémy – v podstatě dojde k pohybu v rozsahu 1/5 obrazu, což má za následek téměř nesledovatelný záznam na zobrazovacím zařízení. Řešením může být jiná mechanická konstrukce, což nemusí vždy být realizovatelné, nebo využití mechanické stabilizace s gyroskopy, což je velice drahé řešení, anebo se využije prostředků digitální stabilizace.

Digitální stabilizace obrazu je použitelná pouze v situacích, kdy nedochází k velkým výkyvům, které by měly za následek pohyb o více než polovinu zorného pole, neboť v té chvíli nebude dostatek informací k tomu, abychom jakoukoliv stabilizaci prováděli. Digitální stabilizace spočívá ve zpracování a analýze obrazu a

následné kompenzaci zjištěného pohybu. Výhodou tohoto přístupu je zcela jistě cena, neboť jde o řešení elegantní a levné. Toto řešení je také značně flexibilní a lze ho dále rozšiřovat a modifikovat, případně integrovat s dalšími algoritmy, což z tohoto přístupu činí ideální metodu stabilizace. Nicméně je nutné myslet i na nevýhody, které toto řešení přináší. Během digitální stabilizace dochází ke ztrátě informace, neboť při korekci mezisnímkového pohybu je nutné obraz ořezat, případně i jinak transformovat, což vede k deformaci a ztrátě informace. Dalším negativem je frekvenční omezení. Digitální stabilizace je schopna eliminovat lépe malé, pomalejší změny (nízká frekvence i amplituda) než rychlé změny s velkou dynamikou (vysoká frekvence i amplituda). Snímání obrazu kamerou samo o sobě filtruje obraz a pracuje jako nízkofrekvenční propust, neboť vysoké frekvence pohybu nelze zachytit díky nízké snímkovací frekvenci, což je fakt založený na *vzorkovacím (Shannonova) teorému* [2], který říká, že přesná rekonstrukce spojitého, frekvenčně omezeného signálu z jeho vzorků je možná tehdy, pokud byla vzorkovací frekvence vyšší než dvojnásobek nejvyšší harmonické složky vzorkovaného signálu. Například tedy při využití frekvence snímání ca 30 snímků za sekundu dostáváme vzorkovací frekvenci signálu 30 Hz, chceme-li zaznamenat kmitání kamery s frekvencí vyšší než 15 Hz, nebudeme toho schopni spolehlivě dosáhnout, neboť vzorkovací frekvence 30 Hz neumožní zachycení periodického kmitání o frekvenci vyšší než 15 Hz.

I přes své nevýhody je digitální stabilizace velmi často využívána například v digitálních fotoaparátech, kamerách a dalších zařízeních především pro poměr cena/výkon. Algoritmy řešící problém digitální stabilizace lze rozdělit například podle toho, jaký typ pohybu jsou schopny detekovat a kompenzovat. Obecně mohou činit problém translace, přiblížení/oddálení (změna měřítka), rotace kolem středu, rotace kolem dalších os a jejich kombinace.

Všechny uvedené typy pohybů lze detekovat a kompenzovat, záleží především na tom, kolik výpočetního času máme k dispozici, resp. jak výkonné vybavení máme (maximální přijatelný čas je dán snímkovou frekvencí). Z mnoha možností se zaměříme pouze na dvě jednoduché, ale vcelku dostačující metody stabilizace, které však nejsou obecné, především proto, že máme omezený čas procesoru a nepotřebujeme dosáhnout příliš vysoké přesnosti.

2.2.1 Stabilizace založená na korelaci

Digitální stabilizace obrazu založená na korelaci patří mezi algoritmy založené na hledání posunutí vzorové oblasti (*block matching*). Přesnost algoritmu je na úrovni pixelů [6], ale lze dosáhnout i subpixelové přesnosti [8], a algoritmus je schopen rozeznat pouze translační, případně i rotační pohyb, což nám pro účely stabilizace velmi často zcela postačuje. Později si ukážeme i komplexnější způsob stabilizace, který bude integrován spolu se sledováním objektu do jediného algoritmu.

Algoritmus jednoduchého hledání vzoru (*plain matching algorithm*)[6] je založen na definici korelace, z níž vychází. Principem je tedy nalezení vybraného vzoru v následujícím snímku videa pomocí korelace. Pro tyto účely se využívá funkce pro 2D korelaci definovaná takto:

$$\boldsymbol{F} \circ \boldsymbol{I}(x, y) = \sum_{j=-N}^{N} \sum_{i=-M}^{M} \boldsymbol{F}(i, j)\boldsymbol{I}(x + i, y + j) \tag{14}$$

kde M a N definují rozměr vzoru, který hledáme, F je vzor, který hledáme a $\boldsymbol{I}$ je vstupní obraz, v němž vzor $\boldsymbol{F}$ hledáme. Přístup založený čistě na korelaci má dvě zásadní nevýhody: tento algoritmus je velmi pomalý a nalezení vzoru je indikováno i v místech, kde je intenzita bodu vysoká, což způsobí vysokou hodnotu korelace a tím i chybné nalezení vzoru. Problém chybného nalezení vzoru při vysoké intenzitě řeší normalizovaná korelace, která je definována jako

$$F \circ I(x,y) = \frac{\sum_{j=-N}^{N}\sum_{i=-M}^{M} F(i,j)I(x+i,y+j)}{\sqrt{\sum_{j=-N}^{N}\sum_{i=-M}^{M} I(x+i,y+j)^2}\sqrt{\sum_{j=-N}^{N}\sum_{i=-M}^{M} F(i,j)^2}}. \quad (15)$$

Pokud je výpočet korelace prováděn v původní (časové) doméně, pak je jeho výpočetní složitost enormní (druhý problém korelace). Proto se často využívá vlastnosti korelace, která umožňuje výpočet provádět ve frekvenční doméně, což umožní výrazné zrychlení celého výpočtu díky možnosti využít rychlou Fourierovu transformaci (pro velmi malé oblasti a rozměry vzoru se však nevyplatí). I přes své nevýhody je korelace z hlediska svého využití díky dobré implementovatelnosti ve specializovaném hardware velice zajímavá i pro zpracování v časové doméně. Korelaci lze například vypočítat speciálním technickým zařízením. Problémy, které toto řešení přináší, lze řešit různými způsoby (optimalizace hodnot, velikostí registrů, apod.), blíže se však o tomto řešení však zmiňovat nebudeme.

Problémy algoritmů založených na korelaci byly velmi často řešeny hranovými detektory, které obraz ve většině případů konvertují na binární a mají za úkol zvýraznit hrany. Kromě hran však mnohdy zvýrazní i místa, která mají původ především v šumu v obraze a z našeho podhledu mají nulovou vypovídací hodnotu. Nevýhodou tohoto přístupu je také výpočetní režie, kterou přidáváme do celého řetězce zpracování. Řešením jsou například algoritmy, které jsou založeny na hledání vzoru v obraze, který je binárně kódován tak, že nejužitečnější informace (obvykle vyšší bity v monochromatickém obraze) je zachována. Výsledkem je algoritmus [7], který pracuje pouze s bity, což nám umožňuje využít modifikovanou verzi korelační funkce, kde výsledkem je funkce výpočtu chyby, která vznikne nahrazením operace násobení binárním operátorem exkluzivního součtu. Funkci lze zapsat jako

$$E(x,y) = \sum_{j=-N}^{N}\sum_{i=-M}^{M} \boldsymbol{F}(i,j) \oplus \boldsymbol{I}(x+i,y+j). \quad (16)$$

Na rozdíl od běžné korelace, kdy se snažíme najít maximum, hledáme v případě této funkce minimum. Typicky se pak stabilizace řeší tak, že je definováno více oblastí, v nichž je hledán posun vybraného vzoru a z výsledných vektorů je vybrán jako vítěz medián. Z hlediska programátorského je takovéto řešení velmi výhodné, neboť celý algoritmus se skládá z operací XOR a součtu, což jsou velice rychle proveditelné operace, důsledkem čehož je, že je celý algoritmus dostatečně rychlý pro práci v reálném čase. Přesnost algoritmu již není na tak dobré úrovni.

2.2.2 Stabilizace založená na analýze optického toku

Kromě stabilizace založené na korelaci, bereme ještě v úvahu stabilizace založenou na analýze optického toku [9]. Tento typ stabilizace je velmi výhodný především proto, že se v rámci řetězce zpracování obrazu často optický tok počítá, ať už je využit ke stabilizaci nebo ne.

Optický tok (*optical flow*) je vektorové pole, které pro každý bod daného snímku $\boldsymbol{I}$ v daném čase n určuje směr a velikost pohybu vzhledem k následujícímu nebo předchozímu snímku (zde závisí na konkrétní aplikaci). Je nutné si uvědomit, že optický tok vyjadřuje pohyb v obraze, nikoli přímo pohyb objektu ve scéně (podobně jako metody odčítání pozadí). Jakákoliv korespondence detekovaného pohybu a pohybu sledovaného objektu musí být dodatečně zjištěna, tedy korespondence s vektorovým polem pohybu (*motion field*), které reprezentuje skutečný pohyb ve scéně, závisí nejen na povaze scény, ale i na způsobu snímání.

Z uvedeného plyne, že k aplikaci využívající optického toku existují dva základní přístupy. První skupina algoritmů určují takzvaný hustý optický tok (*dense optical flow)* – optický tok se počítá pro všechny body v obraze. Vezmeme-li v úvahu, že se v obraze vyskytují plochy plné podobných bodů, u kterých není snadné

určit vektor pohybu (například jednobarevná plocha), je velmi obtížné počítat optický tok pro všechny body. V praxi se tento problém řeší interpolací obtížně počitatelných bodů. Což je cesta ke druhému způsobu řešení optického toku a tím je využití algoritmů ze skupiny řídkého optického toku (*sparse optical flow*). Tyto algoritmy vychází z definované množiny bodů, které by měly být snadno detekovatelná a tím i snadno sledovatelné (viz kapitola 2.3.1). Logickou výhodou plynoucí ze snížení počtu sledovaných bodů je zvýšení rychlosti algoritmu. Nejznámějším algoritmem tohoto typu je algoritmus Lucas-Kanade. Jelikož je výpočetní náročnost algoritmů hustých optických toků příliš vysoká, zaměříme se na druhou skupinu algoritmů, konkrétně na algoritmus Lucas-Kanade [20].

Algoritmus Lucas-Kanade je standardním algoritmem výpočtu optického toku. Existují dvě varianty: algoritmus optického toku s rozptýlenými body využívající základní algoritmus Lucas-Kanade nebo iterační pyramidový Lucas-Kanade. První verze algoritmu byla představena v roce 1981 [20]. Problém tohoto řešení je, že počítá optický tok pouze pro malé okolí bodu a tedy je schopen postihnou pouze malé pohyby. Tento problém řeší novější varianta – iterační pyramidový algoritmus, který je schopen zachytit pohyby daleko větší.

Algoritmus Lucas-Kanade patří mezi diferenciální metody výpočtu optického toku. Tento algoritmus předpokládá, že posun mezi dvěma následujícími snímky ve videu (tedy i posun bodu) je relativně malý. Předpoklady pro správnou funkci algoritmu jsou následující: konstantní jas (*Brightness Constancy*), časová stálost (*Temporal Persistence*), prostorová soudržnost (*Spatial Coherence*).

Předpoklad *konstantního jasu* vychází z toho, že mezi dvěma snímky ve videu se okolo daného bodu nemění intenzita jednotlivých bodů, případně se mění velmi málo (=> problém = šum a rychlý pohyb, který sám o sobě může vést na změnu jasu). Pokud je tento předpoklad splněn, pak můžeme uvažovat o použití této metody. Základní rovnicí pro 2rozměrný prostor, z níž vycházíme, je definována takto:

$$0 = \boldsymbol{I}_{t,n}(x_i, y_i) + \nabla \boldsymbol{I}_n(x, y)\boldsymbol{v}, \tag{17}$$

kde $\boldsymbol{I}_{t,n}(x_i, y_i)$ je temporální gradient (tedy mezisnímkový rozdíl jasů) v daném bodě (x_i, y_i), $\boldsymbol{v} = \left(v_x, v_y\right)^T$ je neznámý vektor optického toku a $\nabla \boldsymbol{I}_n(x_i, y_i)$ je gradient ve směru os x a y. Problém této rovnice je zřejmý – máme dvě neznámé a pouze jednu rovnici. Hlavní myšlenkou vedoucí k řešení této situace je zavedení další omezující podmínky – budeme předpokládat, že pohyb v okolí bodu je minimální, resp. všechny body v předem definovaném okolí se pohybují stejným směrem (*prostorová soudržnost*), tedy všechny body budou mít shodný vektor optického toku $\boldsymbol{v}$. Tímto způsobem získáme v podstatě libovolný počet rovnic v závislosti na tom, jak velké okolí bodu budeme považovat za konstantní, například pro okolí 3×3 získáme 9 rovnic:

$$\underbrace{-\begin{bmatrix} \boldsymbol{I}_{t,n}(x_1, y_1) \\ \boldsymbol{I}_{t,n}(x_2, y_2) \\ \vdots \\ \boldsymbol{I}_{t,n}(x_9, x_9) \end{bmatrix}}_{\boldsymbol{b}} = \underbrace{\begin{bmatrix} \frac{\partial \boldsymbol{I}_n}{\partial x_1} & \frac{\partial \boldsymbol{I}_n}{\partial y_1} \\ \frac{\partial \boldsymbol{I}_n}{\partial x_2} & \frac{\partial \boldsymbol{I}_n}{\partial y_2} \\ \vdots & \vdots \\ \frac{\partial \boldsymbol{I}_n}{\partial x_9} & \frac{\partial \boldsymbol{I}_n}{\partial y_9} \end{bmatrix}}_{\boldsymbol{A}} \underbrace{\begin{bmatrix} v_x \\ v_y \end{bmatrix}}_{\boldsymbol{v}}, \tag{18}$$

řešení této soustavy hledáme nejčastěji například metodou nejmenších čtverců. Zapíšeme-li předchozí rovnici ve tvaru

$$\boldsymbol{b} = \boldsymbol{A}\boldsymbol{v}, \tag{19}$$

pak je řešením vektor $\boldsymbol{v}$, který nalezneme tak, že minimalizujeme kvadrát vzdálenosti $\|\boldsymbol{Av} - \boldsymbol{b}\|^2$. Poslední předpoklad – *časová stálost* – se přímo neprojeví ve výpočtu, spíše omezuje možnost nasazení algoritmu. Předpokládá se, že pohyb je tak pomalý, aby se mezi jednotlivými snímky příliš neprojevil.

Metodu Lucas-Kanade pro výpočet optického toku aplikujeme na množině bodů vhodných pro sledování (rohy) a typicky s malým okolím (například 5×5), nicméně z matematického hlediska není problém aplikovat metodu i na větší okolí, tedy na větší objekty. Tato metoda je vcelku spolehlivá, dokud jsou splněna omezení – tedy jasová stálost bodů, malý pohyb bodů a koherence pohybu bodů (tj. všechny body okolí se pohybují stejným směrem).

Řešením problémů základní metody je její varianta – iterativní pyramidální metoda Lucas-Kanade [21]. Tato metoda vychází z myšlenky, že dojde-li k pohybu dostatečně velkého objektu, je tento pohyb patrný i na zmenšeném snímku. Zmenšíme-li snímek, jsme schopni zaznamenat větší pohyb, protože jeden bod snímku reprezentuje dva body původního snímku. Snímek je možné zmenšovat tak dlouho, dokud budou změny v obraze (pohyb) dostatečně znatelné – vytvoříme tak pyramidu snímků poloviční velikosti v obou osách x a y, tedy celkový počet bodů klesne na čtvrtinu původní velikosti na každé další úrovni. Pokud zvolíme vhodný počet pyramid vzhledem k velikosti sledovaného objektu, je možné sledovat také rychlejší pohyby pokrývající větší prostor. Detailnější popis metody viz [21].

Celkově je nevýhodou metody optického toku neschopnost přizpůsobení se tvarovým změnám objektu způsobeným například změnou měřítka nebo rotací (přizpůsobení lze docílit dodatečnou modifikací vzoru, ale to není ideální dlouhodobé řešení).

Metoda stabilizace obrazu využívající optického toku vychází z předpokladu, že mezi jednotlivými snímky ve videu dochází ke změnám intenzity bodů v obraze v závislosti na pohybu kamery nebo objektu. Dojde-li k pohybu bodu, pak lze změnu pozice tohoto bodu vyjádřit vektorem. Takový vektor lze vytvořit pro každý bod obrazu kamery a na základě matice vektorů pohybu lze provádět další analýzu, například analýzu pohybu kamery. Při pohybu kamery dojde k posunu všech bodů a každý z nich tedy bude posunut v určitém směru v závislosti na druhu pohybu. V případě translačního pohybu, který nás především zajímá, dojde k posunu všech bodů ve stejném směru o vzdálenost závisející na vzdálenosti objektu a parametrech objektivu. Tímto způsobem lze zjisti průměrný pohyb každého bodu (nebo lépe střední hodnotu) a tím i pohyb způsobený pohybem kamery. V případě, že je zorné pole kamery z větší části zakryto blízkým pohybujícím se objektem, bývá tento pohybující se objekt chybně rozpoznán jako pohyb kamery, což je problém, který zatím není algoritmicky řešitelný.

Metoda založená na optickém toku tedy spoléhá na to, že jsme schopni ve dvou snímcích po sobě jdoucích $\boldsymbol{I}_1$ a $\boldsymbol{I}_2$ nalézt tentýž bod a spočítat tak vektor pohybu, který bod vykonal. Předpokládáme, že snímky mají stejný rozměr, liší se pouze čas jejich pořízení. Pro daný bod (x, y) v obraze $\boldsymbol{I}_1$ tedy hledáme bod $(x + \delta_x, y + \delta_y)$ v obraze $\boldsymbol{I}_2$ tak, že se snažíme minimalizovat chybu ε která je definována jako

$$\varepsilon(\delta_x, \delta_y) = \sum_{i=x-R_x}^{x+R_x} \sum_{j=y-R_y}^{y+R_y} \left(\boldsymbol{I}_1(i,j) - \boldsymbol{I}_2(i + \delta_x, j + \delta_y)\right), \tag{20}$$

kde R_x je polovina šířky prohledávané oblasti (resp. poloměr prohledávané oblasti, která je však ve výsledku čtvercová, proto je pojem poloměr nevhodný, na druhou stranu je výstižnější) a R_y je polovina výšky prohledávané oblasti (okolí bodu).

Vektor pohybu bodu lze vyhledat pro každý bod obrazu, ale to není ideální řešení, neboť zabere příliš mnoho výpočetního času. Vhodnější je nalézt významné body v obraze, které posléze sledujeme. Podobného přístupu využíváme i při sledování objektu, kdy nalezneme významné body náležející objektu

a snažíme se je nalézt ve snímku, který následuje, proto se jeví využití tohoto přístupu jako vhodnější. Nalezení bodů vhodných pro sledování je důležité například i pro měření vzdálenosti, pokud bychom ji chtěli měřit prostřednictvím dvojice kamer. V této situaci máme k dispozici dva snímky. Pro výpočet disparity bodů je nutné daný bod z prvního snímku nalézt i ve snímku druhém, je tedy účelné použít metodu výběru vhodného bodu, což zvyšuje užitečnost využití metody pro stabilizaci obrazu, protože ji můžeme využít i pro hledání podobných bodů (problém korespondence bodů) pro účely sledování objektu.

Prvním problémem je tedy volba bodů vhodných ke sledování. Pro tyto účely se nejlépe hodí body, které mají ve svém okolí nějakou texturu, nebo jsou to rohy objektu, tedy body, které lze snadno podle nějaké matematicky popsatelné vlastnosti vyhledat. Formálně lze takové body popsat například maticí (viz kapitola 2.3.1).

Jakmile máme k dispozici množinu bodů, které „stojí za sledování“, můžeme přistoupit k vlastnímu výpočtu optického toku tak, že se pokusíme nalézt zvolené body ze snímku $\boldsymbol{I}_1$ ve snímku $\boldsymbol{I}_2$. K tomuto účelu lze využít více algoritmů. Pro naše účely přichází v úvahu především již výše zmiňovaný algoritmus Lucas-Kanade [20][21].

2.3 Sledování objektů ve videu

Algoritmus sledování objektů v posloupnosti po sobě jdoucích snímků (video) lze popsat velice jednoduše ve dvou krocích:

1. výběr bodů v originálním snímku,
2. nalezení bodů v následujícím snímku (snímcích).

Algoritmy sledování je možné rozdělit do různých skupin podle několika kritérií. Jedním z častých přístupů k dělení těchto algoritmů je rozdělení podle přístupu k řešení problému – například takto podle [3]:

- *segmentační algoritmy* – extrahují pohybující se objekty (popředí) na základě metod segmentace obrazu rozdělením na pozadí a popředí, tj. na základě algoritmické separace toho, co nás zajímá od toho, co nás nezajímá,
- *algoritmy využívající reprezentaci objektu* – slouží robustnímu sledování objektu, který je popsán modelem, tj. každý sledovaný objekt musí být možné popsat modelem, což není vždy splnitelné a někdy i nevhodné (v případě, že nevíme, o jaký typ sledovaného objektu půjde),
- *algoritmy využívající obrazové vlastnosti* – hledají objekt podle určitých obrazových vlastností (příznaků), které jsou extrahovány z původní obrazové informace, tj. objekty jsou vyhledávány například podle svého vzhledu – vzoru, který je následně hledán v následujících snímcích videa,
- *predikční metody* – algoritmy zaměřené na modelování pohybu a z něho vycházející nalezení objektu, využívají se pravděpodobnostní přístupy a některé z předchozích metod.

Dalším způsobem dělení, které nalezneme, je například rozdělení podle směru zpracování obrazu na procesy pracující zdola nahoru a algoritmy pracující shora dolu s dalším dělením podle přístupu, který dále využívají.

Procesy zdola nahoru typicky nějakým způsobem uchovávají reprezentaci objektu a snaží se ho nalézt (lokalizovat) v následujících snímcích. Mezi tyto metody patří například:

- *metody sledující malé oblasti* (*blob based*) – objekt je segmentován na dílčí části, jejichž pohyb je pak sledován metodami detekce oblastí, korelací nebo optickým tokem,
- *metody založené na podobnosti vzorů* (*kernel-based*) – iterativní přístup k lokalizaci, kde je cílem maximalizovat míru podobnosti na základě zvolené metriky – vyhledání vzorů,

- *metody založené na sledování kontur* (*contour tracking*) – je detekována kontura oblasti zájmu a ta je pak sledována například kondenzačním algoritmem [3],
- *metody založené na shodě vizuálních vlastností objektu*.

Procesy shora dolu jsou založeny na filtraci a datové asociaci. Tento způsob se zaměřuje na historii pohybu objektu a snaží se brát v potaz i jeho dynamiku a další vývoj. Tyto metody umožňují sledování komplexnějších objektů a patří mezi ně metody využívající:

- *Kalmanův filtr* – optimální Bayesův filtr pro lineární funkce ovlivněné Gaussovým šumem,
- *částicový filtr* – principiálně založené na rozdělení objektu na částice, jejich ohodnocení a sledování v závislosti na kvalitě částice, tento princip umožňuje sledovat i nelineární procesy.

Základní dělení však není tak podstatné. Mnohem důležitější jsou kritéria hodnocení algoritmů. Takovými kritérii může být například dělení podle odolnosti vůči specifickým druhům obrazových transformací (například odolnost vůči translaci, rotaci a podobně) nebo schopnost kvalitně plnit svůj úkol za různých okolností. Kritérii, která nás tedy budou zajímat, jsou například pohyblivost kamery, odolnost vůči transformacím, rychlost pohybu sledovaného objektu, apod.

Prvním kritériem, které nás bude zajímat, je vhodnost algoritmu pro případy, kdy je kamera statická (například metoda odčítání pozadí) nebo pohyblivá (například metoda vyhledávání vzoru, metoda založená na analýze optického toku, Kalmanův filtr, částicový filtr, apod., viz [3]). Kromě mobility kamery je důležitým údajem například schopnost algoritmu adaptovat se na změny tvaru objektu způsobené rotací, změnou měřítka, případně translací. Mezi metodami schopnými adaptace na transformace je například Kalmanův filtr (především jeho rozšířené varianty) nebo částicový filtr. Metody založené na hledání vzoru jsou náchylné k rychlé ztrátě objektu [3]. Velmi důležitým kritériem je i schopnost reagovat správně na situace, kdy dojde k částečnému nebo úplnému zakrytí objektu jiným objektem nebo pozadím. S těmito problematickými situacemi se nejlépe vypořádají například částicové filtry. Posledním velmi důležitým kritériem v případě nasazení systému v praxi je i výpočetní náročnost, která bude také velkou měrou rozhodovat o výsledném výběru vhodné metody.

Pokud shrneme tato kritéria, je zřejmé, že ideálu se nejvíce blíží částicové filtry, které umožňují správně reagovat na mnoho podnětů a jsou schopny se zotavit i z chybových stavů.

V následujících kapitolách budou popsány základní definice problému, metody výběru vhodných bodů (resp. oblastí) pro účely sledování objektu a dále budou rozebrány jednotlivé metody sledování objektů.

Cílem sledování objektu je, abychom nalezli v aktuálním snímku videa objekt (*cíl*), který jsme si zvolili v předchozím snímku (nebo snímcích). Cíl je volen buď manuálně operátorem (například výběrem oblasti, kde se objekt nachází, nebo jiným způsobem popisu případně označení). Také lze využít automatický způsob detekce, který se však hodí spíše pro autonomní dohledové systémy, které kontrolují určitý perimetr a mají za úkol oznámit jeho narušení v případě detekce objektu, který do sledovaného prostoru nepatří. Lze se také setkat s kombinací obou přístupů, kdy operátor (uživatel) vybere oblast, kde se nachází objekt zájmu, který má být dále sledován, a tato oblast je poté analyzována algoritmem, který rozhodne o tom, jaké části z dané oblasti jsou vhodné pro účely sledování. Tato varianta je z mnoha hledisek nejpřínosnější, neboť operátor má možnost zasáhnout do procesu a případně korigovat chyby algoritmu.

Z hlediska uživatelského je tedy cílem objekt, který chceme sledovat. Z pohledu algoritmu je cílem množina bodů, které budou automaticky sledovány = nalezeny v posloupnosti následujících snímků. V následujících kapitolách je popsáno, jakým způsobem lze vybírat body vhodné pro sledování a jak objekty sledovat.

Abychom byli schopni nalézt řešení (například v podobě matic transformace bodů sledovaného objektu), zavádí různé algoritmy různá omezení. Mezi běžná omezení, s nimiž se setkáme, patří například to, že

pohyb objektu musí být plynulý bez náhlých změn, nedochází k náhlým změnám pozadí, nedochází k velkým změnám vzhledu objektu, je kamera statická, omezíme počet a velikost sledovaných objektů, omezíme počet vzájemných interakcí (především překrytí), a mnohá další, které přímo souvisí s konkrétní aplikací i algoritmem.

Pro praktické nasazení sledování v reálných aplikacích je také nutné stanovit hranice, které pro nás budou rozhodující při výběru a implementaci vybraného algoritmu. Naše stanovené cíle určují, že *počet sledovaných objektů musí být větší než jeden*, *algoritmus musí být schopen řešit náhodné zakrytí sledovaného objektu* a důležité také je omezení v podobě nutnosti pracovat pouze s *jasovou informací* – tedy s monochromatickými kamerami, protože pro náš konkrétní případ uvažujeme sledování na velkou vzdálenost, kdy se barevná informace ztrácí. Mnoho algoritmů však vyžaduje pro svou funkci barevný obraz. Takové algoritmy je pak nutné přizpůsobit, je-li to možné. Naším cílem je navrhnout algoritmus tak, aby byl schopen zpracovat jak barevné video (pro účely srovnání s ostatními algoritmy), tak i monochromatické video (pro praktické nasazení).

2.3.1 Výběr bodů pro sledování

Výběr bodů vhodných pro sledování objektu je kritickou částí celého procesu. V případě chybného zvolení bodu v iniciálním snímku dochází k chybnému nalezení bodu v následujících snímcích. Existuje řada vlastností, které by body měly splňovat, aby byly vybrány jako vhodné pro účely sledování.

Náhodný výběr bodů

Triviální metoda výběru bodů pro účely sledování je náhodný výběr. V inicializační fázi algoritmu se náhodně vybere množina bodů z oblasti náležející objektu, které se poté algoritmus snaží nalézt v jednotlivých po sobě jdoucích snímcích. Zřejmým problémem tohoto postupu je fakt, že vybrané body nemusí mít zrovna vhodné vlastnosti, které by bylo možné sledovat. I přesto se někdy tato metoda používá, především pro svoji rychlost.

Histogramy

Definice histogramu říká, že histogram je grafické znázornění distribuce dat pomocí sloupcového grafu, přičemž výška sloupců vyjadřuje četnost sledované veličiny v daném intervalu a jeho šířka vyjadřuje šířku intervalů (tříd). Je důležité zvolit správnou šířku intervalu, neboť nesprávná šířka intervalu může snížit informační hodnotu diagramu [10].

Pro účely zpracování obrazu se používají buď barevné, nebo černobílé histogramy. Vycházíme z faktu, že naprostá většina obrázků má barevnou hloubku 8 bitů, každý pixel tedy lze zařadit do jedné z 256 tříd. V případě barevného obrazu se pak využívají histogramy tři – jeden pro každou barevnou složku RGB.

V aplikaci sledování objektů se histogramy využívají tak, že je definována oblast, která nás zajímá. Tato oblast je rozdělena na menší oblasti, přičemž pro každou z těchto malých oblastí je vytvořen vlastní histogram. Je zřejmé, že takováto množina histogramů vcelku jednoznačně reprezentuje vybranou oblast. Množina histogramů pak slouží jako vektor popisující hledaný objekt. Histogramy lze normalizovat nebo jinak upravovat a lze takto získat vcelku robustní příznak pro další postup algoritmu. V praxi se však ukazuje, že metody založené na histogramech, selhávají na černobílých snímcích.

SURF

Metoda SURF (*Speeded-Up Robust Features*) je metoda, byla představena trojicí Herbert Bay, Tinne Tuytelaars a Luc Van Gool v roce 2006 [10]. Jedná se novější obdobu metody SIFT (*Scale-Invariant Feature Transform*) z roku 1999 a jejímž autorem je David Lowe [12]. Obě metody slouží k nalezení zajímavých (významných) bodů v obraze. Obrázek popisují pomocí takzvaných deskriptorů vypočítaných pro tyto body. SURF využívá výpočtu determinantu z Hessovy matice (hessián, *DoH – Determinant of Hessian*), naproti

tomu SIFT využívá výpočtu rozdílů Gaussiánů (*DoG – Difference of Gaussian*). Takto získané deskriptory jsou invariantní vůči rotaci a vzdálenosti kamery od popisovaného objektu. Algoritmus SURF se využívá v aplikacích počítačového vidění. Používá se pro rekonstrukci 2D a 3D scén, klasifikaci obrázků a rychlý popis obsahu obrázku. Míru podobnosti dvou obrázků lze měřit jako vzdálenost.

Průběh metody SURF lze rozdělit na dvě fáze. V první fázi se hledají klíčové body obrázku, kterými mohou být rohy, skvrny nebo T-spoje. Druhou fází je výpočet deskriptoru z okolí klíčového bodu. Pro detekci klíčových bodů využívá integrální obraz $\boldsymbol{I_\Sigma}$ definovaný jako

$$\boldsymbol{I_\Sigma}(x,y) = \sum_{i=0}^{i\leq x}\sum_{j=0}^{j\leq y} \boldsymbol{I}(i,j), \tag{21}$$

a detektoru založeného na výpočtu determinantu Hessovy matice, která je ve tvaru:

$$\boldsymbol{H}(x,\mathrm{y},\sigma) = \begin{bmatrix} L_{xx}(x,\mathrm{y},\sigma) & L_{yx}(x,\mathrm{y},\sigma) \\ L_{xy}(x,\mathrm{y},\sigma) & L_{yy}(x,\mathrm{y},\sigma) \end{bmatrix}, \tag{22}$$

přičemž (x,y) je bod ve snímku $\boldsymbol{I}$ a L_{xx} je konvoluce druhé derivace Gaussovy funkce $\frac{\partial^2}{\partial x^2} g(\sigma)$ se vstupním obrazem $\boldsymbol{I}$ v bodě (x,y), více viz [11].

Příznaky vhodné pro sledování

Příznaky vhodné pro sledování (tzv. *Good Features to Track*) nemají jiný specifický název a byly představeny v roce 1994 autory J. Shi a C. Tomasi [13]. Pro účely sledování objektu ve videu se nejlépe hodí body, které mají ve svém okolí nějakou texturu, nebo jsou rohy objektu. Formálně lze takové body popsat maticí

$$\boldsymbol{M}(\boldsymbol{I},x,y) = \begin{bmatrix} \sum_{okolí}\left(\frac{\partial \boldsymbol{I}}{\partial x}\right)^2 & \sum_{okolí}\frac{\partial^2 \boldsymbol{I}}{\partial x \partial y} \\ \sum_{okolí}\frac{\partial^2 \boldsymbol{I}}{\partial x \partial y} & \sum_{okolí}\left(\frac{\partial \boldsymbol{I}}{\partial y}\right)^2 \end{bmatrix} \tag{23}$$

přičemž lze říci, že body vhodné pro sledování jsou takové, u nichž je vysoká hodnota vlastního čísla výše uvedené matice. Postup nalezení bodů vhodných pro sledování je následující:

1. Nejdříve jsou pro každý bod snímku $\boldsymbol{I}$ vypočítány hodnoty vlastních čísel λ_1 a λ_2 výše uvedené matice a ty jsou uloženy do matice $\boldsymbol{E}$, přičemž z vlastních čísel je zvolena menší hodnota $\min(\lambda_1,\lambda_2)$.
2. Dále se provede eliminace nevýrazných hodnot v rámci malého okolí každého bodu o velikosti 3×3 body.
3. V dalším kroku jsou z výběru vyřazeny rohy, které nesplňují kvalitativní kritérium definované jako hodnota $Q \cdot \max \boldsymbol{E}(x,y)$, kde Q je zvolená úroveň kvality (*quality level*).
4. Nakonec jsou vyřazeny body, jejichž vzdálenost od každého dalšího zvoleného bodu je větší, než je zvolená minimální vzdálenost, což vede k vyřazení „slabších“ bodů, které jsou v blízkosti „silných“ bodů.

2.3.2 Metoda odčítání pozadí

Metoda odčítání pozadí [15][16] je jedna z metod počítačového vidění pro segmentaci obrazu a především pak pro detekci pohybujících se objektů umístěných v popředí scény při statickém snímání obrazu. Uplatní se tedy především v situacích, kdy se kamera snímající scénu nepohybuje, neboť pohyb kamery by vnesl do snímku falešné detekce pohybu. Základní metoda vychází z toho, že máme k dispozici model pozadí

scény (tedy snímek objektů, které do scény patří, a které nazýváme souhrnně pozadím), od kterého odčítáme aktuální snímek. Odčítají se vždy jednotlivé body (například jejich jasové hodnoty) na korespondujících pozicích a výsledný rozdíl se porovná s prahovou hodnotou s tím, že hodnoty rozdílu větší, než je prahová hodnota, jsou oceněny hodnotou 1 (popředí, objekt) a ostatní body hodnotou 0 (pozadí). Hodnota prahu je v tomto případě kritická a musí být stanovena tak, aby byl správně zachycen objekt, nikoliv však minoritní změny ve scéně. Kromě pevného stanovení hodnoty prahu existuje i algoritmický přístup ke stanovení této hodnoty dynamicky v závislosti na celkovém stavu scény (například jasové podmínky) – automatické prahování. V závislosti na stanovené prahové hodnotě T lze tedy definovat tuto metodu pro aktuální snímek $\boldsymbol{I}_n$ a daný model pozadí $\boldsymbol{I}_{B,n}$ v čase n jako

$$\boldsymbol{I}_{F,n}(x,y) = \begin{cases} 1, \text{ pro } \left|\boldsymbol{I}_n(x,y) - \boldsymbol{I}_{B,n}(x,y)\right| > T \\ 0, \text{ pro } \left|\boldsymbol{I}_n(x,y) - \boldsymbol{I}_{B,n}(x,y)\right| \leq T, \end{cases} \tag{24}$$

přičemž $\boldsymbol{I}_{F,n}$ je binárním výstupem pro aktuální snímek v čase n s tím, že tento výstup je potřeba dále zpracovat (případné další zpracování – eroze, dilatace, nalezení hran – a pak samotné nalezení objektů). Jelikož výsledkem prahování je pozitivní maska vymezující jednotlivé objekty v obraze, je nutné nějakým způsobem reagovat na změny intenzity osvětlení scény, pohyb kamery a změnu geometrie pozadí, aby bylo možné tuto metodu prakticky použít.

Kromě základní metody odčítání pozadí existují i různé varianty, které se snaží řešit některá problematická místa základní metody, jako je například malá odolnost vůči šumu. Jedna z takových metod modifikuje základní algoritmus tak, že nahrazuje prosté odčítání jasové hodnoty odčítáním hodnoty filtru. Tím může být například jednoduchý průměr nebo medián, jejichž hodnoty získáme z posloupnosti několika předchozích snímků. Tyto metody nutně potřebují určitý „startovací“ čas pro vytvoření dostatečné historie snímků, tedy pro vytvoření kvalitního modelu (tento čas však potřebuje i metoda základní), lze ho však v extrémním případě zkrátit na jediný snímek.

2.3.3 Adaptivní metoda odčítání pozadí

Řešením největších úskalí základní a modifikovaných verzí odčítání pozadí jsou metody adaptivní, které se snaží přizpůsobit jak lokálním podmínkám osvětlení v dané oblasti scény, tak průběžné změně globálního osvětlení. Algoritmus by tedy měl být schopen vyřešit změny osvětlení, reagovat správně na pohyb pozadí (např. tráva, listí, …), a měl by být schopen reagovat na dlouhodobé změny scény (např. změna osvětlení v důsledku počasí).

Takovým algoritmem je například adaptivní metoda odčítání pozadí využívající směs vícerozměrných Gaussových rozdělení, přičemž vícerozměrné Gaussovo (normální) rozdělení [14] je definováno pro D-dimenzionální vektor $\boldsymbol{X}$, vektor středních hodnot $\boldsymbol{\mu}$ a varianční matici $\boldsymbol{\Sigma}$ jako:

$$\mathcal{N}(\boldsymbol{X}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{\frac{D}{2}}|\boldsymbol{\Sigma}|^{\frac{1}{2}}} e^{-\frac{1}{2}(\boldsymbol{X}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{X}-\boldsymbol{\mu})}, \tag{25}$$

kde $|\boldsymbol{\Sigma}|$ je determinant varianční matice $\boldsymbol{\Sigma}$. Varianční matice vektoru $\boldsymbol{X}$ je maticí složenou z kovariancí mezi jeho složkami a je definována jako

$$\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1D} \\ \sigma_{21} & \sigma_2^2 & & \sigma_{2D} \\ \vdots & & \ddots & \vdots \\ \sigma_{D1} & \sigma_{D2} & \cdots & \sigma_D^2 \end{pmatrix}, \tag{26}$$

kde σ_d^2 jsou rozptyly složek (na diagonále matice), jednotlivé prvky jsou kovariancemi složek

$$\sigma_{ij} = \mathrm{cov}\left(X_i, X_j\right), \sigma_{ij} = \sigma_{ji} \tag{27}$$

a matice sigma $\boldsymbol{\Sigma}$ je symetrická a pozitivně semidefinitní (více viz [14]). Takto definovaná Gaussova funkce slouží jako základ směsi adaptivních Gaussových funkcí. Směs se využívá proto, že lépe modeluje několik různých povrchů bodů a adaptace je nutná z důvodu přizpůsobování okolnímu prostředí a změnám osvětlení. V každé iteraci metody jsou hodnoty přepočítány jednoduchou heuristikou s tím, že se odhadne, které patří s největší pravděpodobností pozadí. Body, které pozadí nenáleží, jsou hledaným popředím a jsou spojovány dalšími metodami zpracování obrazu do ucelených oblastí. Detailnější rozbor není potřebný a další informace o adaptivních metodách odčítání pozadí lze nalézt v literatuře (například [15][16]).

2.3.4 Metoda založená na sledování optického toku

Metoda sledování objektu založená na optickém toku vychází z principu výpočtu optického toku jednotlivých bodů (viz kapitola 2.2.2). Nosnou myšlenkou této metody je opět výběr bodů pro sledování a následné sledování jejich pohybu. Vektor pohybu daného bodu se vypočítá některou metodou výpočtu optického toku. Tímto způsobem lze vypočítat vektor pohybu všech sledovaných bodů. Nevýhodou této metody je její velká chybovost [22].

2.3.5 Metoda založená na částicových filtrech

V několika předcházejících letech se stal velmi populárním přístup založený na různých modifikacích a variantách rekurzivního Bayesovského filtrování, nazývaného také částicové filtry (*particle filters*), někdy také označované jako sekvenční metody Monte Carlo (SMC). Částicové filtry poskytují robustní základ pro sledování objektů, který se neomezuje pouze na lineární dynamické systémy a Gaussovo rozložení šumu. Určitou formu částicového filtru použili poprvé na sledování objektů ve videu Isard a Blake a nazvali ho *Condensation* [24], což je zkratka z *Conditional Density Propagation* a v komunitě zaměřené na počítačové vidění je tento algoritmus typickým představitelem částicových filtrů. Tento algoritmus byl použit na sledování objektů v obrazech upravených na konturovou reprezentaci scény [25][26].

Konturové modely jsou relativně robustní vůči změnám osvětlení, ale mohou být výpočetně náročné a náchylné k chybám způsobeným členitostí pozadí scény [27]. Využití informace o barvě je další z mnoha velmi často využívaných technik. Mnohdy je využívána informace ve formě histogramu [23][28]. Částicové filtry využívající barevné informace v podobě histogramů jsou při sledování objektu velmi odolné vůči rigiditě pohybujících se objektů, částečnému zakrytí objektu a šumu. Nicméně tento způsob popisu je omezen tím, že ignoruje prostorovou polohu bodů, což znemožňuje rozeznání objektů s podobným barevným rozložením. Tento problém je pak znásoben v případě nutnosti práce s monochromatickým videem [29].

Jiným jednoduchým a velmi často používaným přístupem k popisu pohybujícího se objektu je využití vzoru objektu (*template*) – tedy malé části snímku, který přímo reprezentuje vzhled daného objektu. Výhodou tohoto přístupu je, že tento popis v sobě nese informaci o barvě i prostorovém vzhledu. Tento přístup je vhodný pro barevné i monochromatické modely a jeho výpočet bývá velmi rychlý. Na druhou stranu může dojít (a dochází) k tomu, že vybraný vzor se stane nereprezentativní pro sledovaný objekt především z důvodu změny jeho vzhledu v důsledku pohybu (změna měřítka, rotace) a šumu. Aby se předešlo tomuto problému, využívá se například techniky pomalé aktualizace vzoru v průběhu času [30][31][32]. V průběhu aktualizace však hrozí nebezpečí, že vzor postupně začne reprezentovat zcela jiný objekt v důsledku nesprávného určení polohy. Velká pozornost musí být také věnována i (částečnému) zakrytí objektu, které může také vést k selhání algoritmu.

Nelineární rekurzivní Bayesovské sledování

Nelineární stochastický systém v doméně diskrétního času lze popsat následujícími způsobem [11]. Rovnice

$$x_{n+1} = f(x_n, d_n), \tag{28}$$

se nazývá stavovou (nebo přechodovou) rovnicí a x_{n+1} popisuje stav systému v čase $n+1$, x_n popisuje stav systému v čase n a d_n lze považovat za náhodné sekvence v podobě bílého šumu s neznámou statistikou v diskrétní časové doméně. Následující rovnice se nazývá rovnice měření (nebo pozorování), kde pozorování v čase n je označeno jako y_n, x_n je stav systému ve stejném čase a v_n je, podobně jako u předchozí rovnice, šum:

$$y_n = h(x_n, v_n). \tag{29}$$

Za povšimnutí stojí fakt, že aktuální stav systému závisí pouze na předchozím stavu, což je situace, kterou lze popsat Markovskými modely, a označujeme ji jako Markovský proces. Za určitých okolností lze tyto rovnice redukovat na lineární Markovský model. Teorie zabývající se touto problematikou je velmi dobře popsána například v [11]. Cílem sledování objektu (filtrování) je tedy určení stavu x_n, který je skrytý, na základě předchozích pozorování $y_{0:n} = (y_0, y_1, \dots, y_n)$.

Pokud se na celý problém podíváme z Bayesovského úhlu pohledu, pak je problém sledování objektu vytvoření posteriorní hustoty pravděpodobnosti $p(x_n|y_{0:n})$ aktuálního stavu x_n, za předpokladu, že je dána posloupnost pozorování od počátečního stavu 0 do času n a za dané počáteční hustoty $p(x_0)$, přechodové hustoty $p(x_n|x_{n-1})$ a pravděpodobnosti $p(y_n|x_n)$. V principu lze tuto posteriorní hustotu vypočítat rekurzivně ve dvou krocích [11] výpočtem Chapman-Kolmogorovovy rovnice predikčního kroku, která slouží k určení předchozí hustoty $p(x_n|y_{0:n-1})$:

$$p(x_n|y_{0:n-1}) = \int p(x_n|x_{n-1})\, p(x_{n-1}|y_{0:n-1}) dx_{n-1} \tag{30}$$

a rovnice aktualizační, která vychází z úměrnosti:

$$p(x_n|y_{0:n}) \propto p(y_n|x_n) p(x_n|y_{0:n-1}). \tag{31}$$

Rekurzivní vztah obou rovnic nelze obecně určit analyticky kvůli náročnosti, téměř nemožnosti, vyhodnocení obvykle komplikovaného integrálu (kromě speciálních případů, kdy jde o lineární a Gaussovský stavový prostorový model).

Pomocí systémového modelu $p(x_n|x_{n-1})$ je odhadnuta apriorní funkce hustoty pravděpodobnosti (*pdf*) na základě předcházejícího stavu systému $p(x_n|y_{0:n-1})$ v čase $n-1$. S dalším měřením (v čase n) se na základě Bayesova pravidla vypočítá aposteriorní *pdf* (fáze aktualizace) podle rovnice

$$p(x_n|y_{0:n}) = \frac{p(y_n|x_n) p(x_n|y_{0:n-1})}{p(y_n|y_{0:n-1})}, \tag{32}$$

kde pravděpodobnost $p(y_n|x_n)$ zastupuje model měření a představuje sadu naměřených hodnot v čase n, filtr modifikuje v této fázi nepřesnou apriorní *pdf* měřením y_n, a $p(y_n|y_{0:n-1})$ je normalizační konstanta.

Rekurentní vztahy mezi fázemi vytváří optimální Bayesovské řešení. Abychom se vyhnuli integraci v rekurzivní Bayesovské rovnici, je posteriorní hustota aproximována množinou vzorků, které mají přiřazeny pravděpodobnostní váhy. Nicméně, posteriorní hustota může být nestandardní, a proto může být značně obtížné vygenerovat vzorky z posteriorní hustoty přímo. Abychom tomuto problému předešli, používá se další, tzv. navrhovaná hustota, z níž lze snadněji generovat vzorky, které jsou použity namísto

hodnoty pravděpodobnosti posteriorní hustoty. Tento postup je označován jako *Importance Sampling* [11][17]. Aposteriorní *pdf* tedy vyjádříme například náhodně generovanými váženými vzorky – *částicemi*. S rostoucím počtem částic se přibližujeme skutečnému aposteriornímu *pdf*. Množinu částic vyjádříme jako:

$$\left\{\forall i = 1, \dots, N: X_{n-1}^{(i)}, w_{n-1}^{(i)}\right\} \tag{33}$$

kde $w_{n-1}^{(i)}$ představuje váhu konkrétní částice. Tuto množinu částic poté používáme při výpočtu v jednotlivých krocích algoritmu.

Bayesovský bootstrap filtr

Bayesovský bootstrap částicový filtr (převzorkovávací filtr) je variantou filtru převzorkování na základě významnosti (*Sequential Importance Resampling – SIR*)[18], kde je hustota přechodu využita jako navrhovaná hustota, a kde je krok převzorkování prováděn při každé iteraci. Obecný algoritmus Bayesovského převzorkovávacího filtru je složen z následujících kroků:

1. Predikce: každá částice $\check{\chi}_{n-1}^{i}, i \in\{1, \dots, N\}$ je využita v modelu systému k výpočtu diskrétní aproximace hustoty pravděpodobnosti
$$\chi_n^i = f\left(\check{\chi}_{n-1}^{i}, d_{n-1}\right). \tag{34}$$

2. Aktualizace: je vyhodnocena pravděpodobnost všech pozorování, každé částici je přiřazena nenormalizovaná váha významnosti:
$$\varpi_n^i = p\left(y_n \middle| \chi_n^i\right). \tag{35}$$

3. Normalizace vah:
$$\omega_n^i = \frac{\varpi_n^i}{\sum_1^N \varpi_n^i}. \tag{36}$$

4. Převzorkování: ze stávající množiny částic $\{\chi_n^1, \dots, \chi_n^N\}$ je vybráno N částic $\check{\chi}_n^i$ na základě jejich významnosti, tedy na základě jejich vah ω_n^i.

Inicializace filtru vyžaduje N částic, které jsou vybrány ze známé hustoty pravděpodobnosti, tyto částice pokračují přímo krokem 2, jsou tedy aktualizovány přímo. Jako jednotlivé měření může být odhadovaný stav vypočítán jako střední hodnota nejvýznamnějších částic nebo může být reprezentován přímo nejvýznamnější částicí (tj. jako *Maximum a-posteriori* – MAP).

2.4 Shrnutí

V této kapitole jsme shrnuli teoretické poznatky, které mají vztah k problematice digitální stabilizace obrazu a sledování objektu. Kromě základních definic problému byly představeny metody pro výběr bodů vhodných pro sledování a metody sledování objektů. Vybrané metody jsme také implementovali a porovnávali jsme vlastnosti, které dané řešení má, především s přihlédnutím na požadavky aplikací v reálném světě.

Porovnání metod bylo založeno čistě na subjektivním hodnocení kvality výstupu, rychlosti a potřeb pro zvolenou úlohu. Zvážení výhod a nevýhod nás vedlo k tomu, že jako vítězný kandidát pro implementaci byl zvolen algoritmus založený na sledování částic, konkrétně pak varianta Bayesovský bootstrap filtr s tím, že budou provedeny specifické úpravy pro zlepšení vlastností tohoto algoritmu.

V následujících kapitolách bude ukázán postup optimalizace software pro plynulý chod aplikace integrující automatické sledování objektů dle stanovených cílů.

3 Implementace a optimalizace algoritmů

V této kapitole se zaměříme na implementaci algoritmů uvedených v kapitole 2 a na optimalizace kódu za účelem dosažení dostatečné rychlosti zpracování. Popíšeme implementaci aplikace pro kamerový systém (návrh GUI), algoritmů (sledování cílů a digitální stabilizace) a jejich optimalizaci prostřednictvím paralelizace (GP-GPU a SIMD).

Na začátku práce jsme stanovili podmínky, kdy sledovaný objekt může měnit své měřítko, otáčet se, pohybovat, být překryt jinými objekty a může být podobný jiným objektům a jako nejvhodnější algoritmus založený na částicovém filtru, respektive na jeho variantě – Bootstrap filtr (viz kapitola 2.3.5), který šíří částice podle prvního řádu adaptivního dynamického modelu.

Dále budeme vycházet z následujících předpokladů. Sledovaný objekt je reprezentován pevně daným obdélníkovým vzorem s automaticky vytvořenou maskou objektu. Abychom předešli kumulaci rušivých pohybů kamery při určování rychlosti objektu, která je velmi významná pro správnou predikci, je souřadnicový systém částic odlišný od souřadnicového systému bodů snímku (pixelů). Velká pozornost je věnována možnému výskytu zákrytu vzoru nebo objektu. Tento problém je řešen kombinací metody odčítání pozadí pro detekci pohybu a masky objektu.

3.1 Programátorské prostředky

V následujících dvou kapitolách budou popsány dva hlavní vývojářské prostředky využité při vývoji aplikace kamerového systému – knihovny Qt a OpenCV – a prostředky pro heterogenní programování výpočetních zařízení sloužící k akceleraci algoritmů. Aplikace je naprogramována v jazyce C++, který zde rozebírat nebudeme, krátce se však zmíníme o dalších technologiích. Knihovna Qt poskytne prostředky pro návrh a provoz GUI a knihovna OpenCV poskytne mnoho užitečných funkcí pro zpracování obrazu s návazností například na algoritmy sledování objektu nebo stabilizaci. Veškeré informace o knihovnách jsou dostupné na oficiálních internetových stránkách: Qt [41] a OpenCV [42]. Dále následuje popis heterogenního přístupu k návrhu a vývoji aplikací spolu se dvěma nejčastěji používanými řešeními v podobě knihovny OpenCL [43] a technologie CUDA [44][45].

3.1.1 Heterogenní přístup k programování

Velká poptávka po systémech schopných zpracovat v reálném čase velké množství dat (vysoké rozlišení kamer, 3D grafika) způsobila rozvoj v oblasti grafických karet (GPU – *Graphics Processing Unit*) a jejich nasazení právě v těchto oblastech. Vzniklo mnoho prostředků pro využití možností, které současné výpočetní systémy nabízí. Ačkoliv je vývoj procesorů pro obecné aplikace (CPU – *Central Processing Unit*) velmi rychlý a jejich úroveň nikterak nezaostává za procesory grafických karet, v mnoha oblastech nedostačuje svým výkonem. Těmito oblastmi je již zmiňovaná 3D grafika, nebo zpracování dat z kamer s vysokým rozlišením, které se velmi často projevují společným rysem – tyto úlohy je možné velmi dobře paralelizovat. Výkon CPU není malý, ale pro tyto typy úloh se mnohem více hodí masivně paralelní přístup, který právě například řešení založená na využití GPU nabízí. A nejde pouze o využití GPU, ale i jiných přídavných karet, které nabízí dodatečný výkon ve formě určitého „koprocesingu“ (například dříve známé koprocesory 80387 pro 80386). Tyto systémy jsou heterogenní – tedy výpočetní prostředky poskytují různé způsoby zpracování dat.

Heterogenními hardwarovými (výpočetními) prostředky tedy rozumíme přídavná zařízení nebo i externí zařízení, která poskytují výpočetní výkon pro akceleraci algoritmů. V našem případě půjde o hardwarovou heterogenitu lokální, tedy přídavná zařízení přímo v počítači nebo lokálně blízko něho. Hardwarovou heterogenitou rozumíme využití různých hardwarových prostředků v jednotném prostředí (heterogenní

hardware vs. homogenní programátorské prostředí). Heterogenita se může projevit například na úrovni jader, procesorů nebo výpočetních uzlů, kde dostáváme na výběr několik úrovní členění (jeden procesor nabízí několik jader = příliš homogenní případ, grafická karta pro výpočty = vhodný příklad heterogenního prostředí – vazba CPU+GPU). Další aspekty, které ovlivňují tento druh využití výpočetních prostředků, jsou například výpočetní heterogenita (hrubý výkon, skoky), komunikační heterogenita (latence a šířka pásma, sdílená paměť) a paralelismus a škálovatelnost.

Problémem v současné době je především to, že máme k dispozici výkonný hardware, ale neumíme ho dostatečně využít, neboť programátorské prostředky, které by to umožnily, se omezují buď na přímou znalost hardware, nebo na obecné modely, které je potřeba znát velmi dobře, aby bylo možné získat výkon, který hardware nabízí.

Z hlediska programátorského se nabízí několik variant řešení přístupu k homogenizaci heterogenních výpočetních prostředků (tj. jednotné programátorské prostředí nezávislé na hardware, který může být různorodý). V současné době se nabízí řešení, která lze rozdělit na závislá a nezávislá na konkrétní hardwarové platformě. Následuje výčet nejznámějších dostupných řešení s tím, že existuje i mnoho variant, především v akademické sféře, které jsou zatím spíše teoretické:

- Řešení nezávislé na hardware:
    - OpenMP (vícejádrové, rozšíření jazyka direktivami, nevhodné pro heterogenní systémy),
    - OpenHMPP (hybridní, vhodné pro heterogenní systémy, rozšíření jazyka direktivami),
    - OpenCL (hybridní, multiplatformní jazyk).
- Řešení orientované na GPU – GP-GPU technologie:
    - Nezávislé na konkrétním hardware:
        - Direct Compute (Microsoft, API),
        - C++ AMP (akcelerace kódu v jazyce C++, knihovna založená na DirectX 11),
        - Lib Sh (knihovna pro C++).
    - Závislé na konkrétním hardware (výrobce, typ, apod.):
        - StreamSDK (technologie firmy AMD, podpora je však směřována spíše k OpenCL a DirectCompute),
        - CUDA (technologie firmy nVidia, karty však podporují i OpenCL a DirectCompute).

Z uvedených technologií jsou zřejmě nejznámější OpenCL, CUDA a DirectCompute. Pro naši práci vybereme řešení využívající buď OpenCL nebo CUDA.

OpenCL

OpenCL je Framework původně vytvořený firmou Apple a jeho hlavní výhodou je možnost nasazení v heterogenním prostředí. Nyní se o specifikaci stará skupina Khronos Group a aktuálně je ve verzi 1.1 a připravuje se verze 1.2 [43]. Cílem je, aby programátoři mohli napsat jeden přenositelný (paralelní) program, který bude schopen využít všechny dostupné prostředky (přídavné procesory, výpočetní karty apod.) nezávisle na výrobci a typu karty (podmínkou je dostupnost ovladače pro konkrétní platformu).

Celý princip spočívá v rozdělení výpočetních prostředků na hlavní řídicí systém a na jedeno nebo více OpenCL zařízení, přičemž každé zařízení obsahuje jednu nebo více výpočetních jednotek a každá výpočetní jednotka je dále dělena na jeden nebo více výpočetních elementů. OpenCL je implementováno jako knihovna (ovladač) výrobce konkrétního hardware. Definuje platformu, které umožní interakci systému s kartami schopnými poskytovat OpenCL funkcionalitu. Nyní podporuje každý výrobce pouze jednu platformu i v případě více zařízení stejného výrobce (například dvě grafické karty v systému) – zde pak rozlišujeme jednotlivá zařízení.

CUDA

CUDA (*Compute Unified Device Architecture*) je hardwarově-softwarová architektura umožňující, podobně jako OpenCL, na GPU spouštět programy napsané v jazycích C/C++, FORTRAN nebo programy postavené na technologiích OpenCL, DirectCompute a jiných. Tato architektura je však dostupná pouze na grafických kartách společnosti nVidia, která ji vyvinula.

Architektura čipu GPU firmy nVidia, je založena na velkém množství relativně jednoduchých skalárních procesorů, což je rozdíl od klasického CPU, který bývá velmi komplexní. Čipy této firmy se liší například i od architektury konkurenční firmy AMD, jejíž multiprocesory jsou tvořeny komplexnějšími VLIW – *Very Long Instruction Word* – jednotkami, které jsou organizovány do celků nazývaných streaming multiprocesory. Vzhledem k tomu, že se jedná o SIMT (*Single Instruction Multiple Threads*) architekturu, je řízení jednotek a plánování instrukcí vcelku jednoduché a spolu s malou vyrovnávací pamětí zabírá malou část plochy čipu, což má však za následek právě omezení v predikci skoků a časté výpadky cache. Poslední významnou částí, která je rozměrově velice podobná CPU, je řadič paměti, který zabírá přibližně stejnou plochu. Výše zmíněný streaming multiprocesor se obecně skládá z několika (v současnosti až z 32) procesorů, pole registrů, sdílené paměti, několika jednotek pro ukládání a čtení a speciální funkční jednotky pro výpočet složitějších funkcí jako je sinus, kosinus nebo logaritmus. Výpočetní možnosti zařízení jsou dány množinou instrukcí, které jsou podporovány, konkrétní vlastnosti však pro nás v tomto případě nejsou rozhodující.

Z hlediska programátorského je způsob programování CUDA podobný OpenCL s tím rozdílem, že je poněkud jednodušší v inicializační fázi, neboť nemusíme řešit tu skutečnost, že nevíme, s jakým hardware a s jakými prostředky přijdeme do kontaktu, což snižuje složitost programátorského modelu. Tento fakt a skutečnost, že CUDA byla představena dříve než OpenCL, je také důvodem pro větší rozšířenost a oblíbenost využití CUDA pro různé výpočetně náročné úkoly.

3.2 Implementace algoritmu sledování objektu

Sledování začíná výběrem objektu (cíle), který bude sledován (výběr provádí operátor prostřednictvím vstupních zařízení – myši nebo dotekového monitoru), v aktuálním snímku videa. Výběr je obdélníkový a celý sledovaný objekt by měl být ohraničen. Takto vybraný objekt zájmu bude sloužit jako referenční vzor. K referenčnímu vzoru je vypočítána automaticky maska vzoru metodou odčítání pozadí. Jako pozadí slouží iniciální snímek, případně více snímků. Vzor i maska jsou pak normalizovány na velikost 32 × 32 obrazových bodů. Vzor (spolu s maskou objektu) je aktualizován jednou a pak zůstává beze změn. Ačkoliv naše řešení poskytuje robustní způsob sledování na základě stanoveného vzoru bez kontinuální aktualizace/adaptace vzoru objektu, je možné použít i některou techniku aktualizace vzoru (viz například [31][32][33]). V našem řešení byly snahy o adaptaci vzoru spíš kontraproduktivní, proto je nevyužíváme.

3.2.1 Stav cíle a pohybový model

Aby bylo možné pokrýt všechny možnosti pohybu objektu, je pro částicové filtry často využíván „slabý“ pohybový model [34]. Tento přístup umožňuje překonávat vznik odchylek způsobených například pohybem kamery, a tím posiluje kvalitu chování systému sledování v situacích, kdy je přesné modelování pohybu nesnadné nebo nemožné. Nicméně je rozumné předpokládat, že schopnost objektu manévrovat je omezena fyzikálními zákony a využít toho k maximálnímu omezení prohledávaného prostoru. Jinak řečeno – vhodný model pohybu může dramatickým způsobem snížit pravděpodobnost selhání algoritmu sledování a také výpočetní nároky.

Navržené řešení je popsáno stavovým vektorem systému $\boldsymbol{s}_n = (P_n, \boldsymbol{v}_n, S_n)$, kde $P_n = (x, y)$ je bod, který udává polohu cíle v souřadném systému částic, $\boldsymbol{v}$ je vektor rychlosti objektu a $S_n = (w, h)$ udává rozměr

ohraničující obdélníkové oblasti (šířku w a výšku h) v čase n. Abychom zmírnili potíže v předvídatelnosti složení pohybů, modelujeme pohyb kamery a objektu separátně, což je důvodem pro popis pohybu pohybovými rovnicemi prvního řádu dynamického modelu s adaptivním normálním (Gaussovým) šumem:

$$\begin{aligned} P_{n+1} &= P_n + \boldsymbol{v}_n + \mathcal{N}_2\big(0,\, \boldsymbol{\Sigma}_P\big), \\ \boldsymbol{v}_{n+1} &= \boldsymbol{v}_n + \mathcal{N}_2(0, \boldsymbol{\Sigma}_v), \\ S_{n+1} &= S_n + \mathcal{N}_2(0, \boldsymbol{\Sigma}_s), \end{aligned} \tag{37}$$

kde $\mathcal{N}_2\big(0,\, \boldsymbol{\Sigma}\big)$ je normální (Gaussovo), 2rozměrné rozložení se střední hodnotou $(0{,}0)$ a varianční maticí $\boldsymbol{\Sigma}$. Varianční matice pro jednotlivé rovnice jsou:

$$\boldsymbol{\Sigma}_P = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}, \boldsymbol{\Sigma}_v = \begin{bmatrix} \sigma_{vx}^2 & 0 \\ 0 & \sigma_{vy}^2 \end{bmatrix}, \boldsymbol{\Sigma}_S = \begin{bmatrix} \sigma_w^2 & 0 \\ 0 & \sigma_h^2 \end{bmatrix}, \tag{38}$$

přičemž hodnoty rozptylu byly stanoveny experimentálně: $\boldsymbol{\sigma}_x^2 = \boldsymbol{\sigma}_y^2 = 5$, $\boldsymbol{\sigma}_{vx}^2 = \boldsymbol{\sigma}_{vy}^2 = 0{,}05$ a $\boldsymbol{\sigma}_w^2 = \boldsymbol{\sigma}_h^2 = 0{,}2$. Rychlost nejlépe ohodnocené částice je stanovena jako

$$\boldsymbol{v}_{n+1} = \frac{1}{N} \sum_{i=0}^{N} \bar{P}_{n-i} - \bar{P}_{n-i-1}, \tag{39}$$

kde $\overline{P}_n$ je střední hodnota polohy v čase n a N je počet předchozích stavů, které jsou vzaty v potaz pro stanovení aktuální hodnoty (experimentálně byla zvolena hodnota $N = 25$). V našem případě je stanovený stav v čase n reprezentován právě nejlépe ohodnocenou částicí.

3.2.2 Míra podobnosti vzoru a obrazu

Funkce pravděpodobnosti pozorování je založena na míře podobnosti (resp. na vzdálenosti) mezi vzorem a oblastí aktuálního snímku reprezentovaného částicí. Vzorem rozumíme čtvercovou oblast obrazu

Jednoduchá míra (například ve formě vzdálenosti), která je běžně používána pro porovnávání shody vzoru a obrazu (*template matching*), bývá založena chybě shody intenzit vzoru a obrazu například v podobě součtu rozdílů čtverců (*Sum of Squared Differences – SSD*) obou oblastí snímku (tedy vzoru z iniciálního snímku a obrazu z aktuálního snímku). Nicméně tato míra je značně nepraktická pro účely sledování objektů, protože není dostatečně odolná proti úplnému ale i částečnému zakrytí sledovaného objektu, k němuž může běžně docházet například zmizením za jiný objekt. Odolnost vůči zakrytí je velmi důležitá vzhledem k přesnosti předchozích určení polohy, které pak mají vliv na každé následující určení polohy a predikci pohybu. Velmi často částečné zakrytí objektu předchází úplnému zmizení, a proto je nutné této situací věnovat pozornost.

Mnoho pozornosti v literatuře je věnováno detekci a zacházením s částečným i úplným zakrytím objektu [30][32] (viz následující kapitola). Přístupy založené na robustním statistickém zpracování je efektivní v případě, že statistický popis překrytí koresponduje s definovanými předpoklady. Nicméně tyto algoritmy nepostihují situace, v nichž jsou si objekt, za který se sledovaný objekt schová, a sledovaný objekt příliš podobné. Náš přístup se na tento problém zaměřuje a přináší řešení v podobě integrace modelu pohybu objektu, který získáme odčítáním pozadí, do měření míry podobnosti takovým způsobem, který zvýší odolnost algoritmu vůči částečnému zakrytí a zároveň zvýší přesnost sledování objektu. Navrhovanou míru podobnosti lze definovat jako:

$$d(\boldsymbol{I}_a, \boldsymbol{M}_a, \boldsymbol{I}_b, \boldsymbol{M}_b) = \sum_{(x,y)\in I} \underbrace{\overbrace{e^{\omega \min(\boldsymbol{M_a}, \boldsymbol{M_b})}}^{\text{pohyb}} \overbrace{(1 - |\boldsymbol{I}_a - \boldsymbol{I}_b|)^2}^{\text{vzhled}}}_{\text{operace prováděné nad body}}, \tag{40}$$

kde ω je vážený koeficient pohybové složky, $\boldsymbol{I}_a$ je snímek s odpovídající maskou objektu (popředí) $\boldsymbol{M}_a$, $\boldsymbol{I}_b$ je snímek s odpovídající maskou objektu $\boldsymbol{M}_b$. Hodnoty jasu bodů snímků i masek jsou normalizovány do intervalu $< 0,1 >$. Operátor $\min$ si lze představit jako průnik masek, což zvýší významnost částí společných oběma maskám. Pokud je jedna z masek nulová, degraduje rovnice na sumu rozdílů čtverců, což značí, že rozdílný vzhled objektu a vzoru nadále způsobuje penalizaci. Tato vlastnost je také využita k inicializaci algoritmu sledování, kdy je maska objektu neinicializovaná.

Pro účely měření podobnosti vzoru a obrazu slouží tři hlavní parametry – vzor sledovaného objektu, jeho maska a oblast v aktuálním snímku reprezentovaná částicí s přidruženou maskou pohybu. Trojice $(\boldsymbol{s}_n, \boldsymbol{I}_n, \boldsymbol{H}_n)$ popisuje pozorování v časovém okamžiku n, přičemž $\boldsymbol{s}_n$ je stavový vektor systému (definovaný dříve), $\boldsymbol{I}_n$ je snímek a $\boldsymbol{H}_n$ je matice transformace mezi dvěma v čase po sobě jdoucími snímky $\boldsymbol{I}_{n-1}$ a $\boldsymbol{I}_n$, přičemž $\boldsymbol{H}_0$ je jednotková matice. Označme tuto trojici $\boldsymbol{C}_n$, tedy

$$\boldsymbol{C}_n = (\boldsymbol{s}_n, \boldsymbol{I}_n, \boldsymbol{H_n}) \tag{41}$$

Označme posloupnost

$$\mathcal{H} = (\boldsymbol{C}_n, \boldsymbol{C}_{n-1}, \dots, \boldsymbol{C}_{n-N}) \tag{42}$$

historií procesu sledování, kde $\boldsymbol{C}_n$ je pozorování v časovém okamžiku n a N udává délku celé uchované historie. Dále definujeme funkci $\varphi(\boldsymbol{s}, \boldsymbol{I}, \boldsymbol{F})$, jejímž výstupem je oblast ve snímku $\boldsymbol{I}$, parametrizovaná stavem $\boldsymbol{s}$ a maticí transformace $\boldsymbol{F}$, která popisuje převod souřadnicového systému částic do souřadnicového systému snímku (ta vychází ze vzájemného vztahu snímků a z polohy cíle v daném snímku). Pravděpodobnost p shody vzoru a obrazu v aktuálním snímku je pak dána jako

$$p \propto e^{d(\varphi(\boldsymbol{s}_n, \boldsymbol{I}_n, \boldsymbol{F}), \varphi(\boldsymbol{s}_n, |\boldsymbol{I}_n - \boldsymbol{I}_b|, \boldsymbol{F}), \boldsymbol{T}, \boldsymbol{M}_0)}, \tag{43}$$
$$\boldsymbol{F} = \prod_{i=0}^{n} \boldsymbol{H}_{n-i},$$

kde d je dříve definovaná vzdálenost, $\boldsymbol{T}$ je vzor (sledovaný objekt) se svojí maskou $\boldsymbol{M}_0$ (maska popředí), $\boldsymbol{I}_n$ je aktuální snímek, $\boldsymbol{I}_b$ je snímek z historické posloupnosti snímků $\mathcal{H}$, kde obdélníkové ohraničující oblasti definované stavovými vektory $\boldsymbol{s}_{n-1}$ a $\boldsymbol{s}_b$ nemají žádný nebo jen minimální průnik s ohledem na délku celé historie, $\boldsymbol{F}$ je matice transformace ze souřadnicového systému částic do souřadnicového systému snímku, přičemž je brána v potaz celá historie sledování od počátku, $\boldsymbol{H}_n$ jsou matice transformace mezi po sobě jdoucími snímky a $\boldsymbol{s}_n$ je aktuální stavový vektor.

3.2.3 Zakrytí a zmizení objektu, pohyb kamery

Dalšími problémy, jimiž se musíme zabývat, je mizení objektu za jinými objekty (částečné, ale i úplné) a pohyb kamery.

Částečné zakrytí objektu může být pro mnohé metody stejně závažný problém, jako úplné zmizení, především z důvodu toho, že nelze přesně určit, jak se bude chovat objekt, který zmizí za překážku. Existují dvě základní situace – cílené a necílené zmizení za objekt. Necíleným zmizením rozumíme situaci, kdy je sledovaný objekt částečně nebo úplně zakryt objekty v popředí (listy, stromy, sloupy, různé objekty) s tím, že sledovaný objekt je takto zakryt, aniž by se o to cíleně snažil. Na rozdíl od necíleného zakrytí je u cíleného zřejmý úmysl skrýt se a v tomto případě nelze dále objekt sledovat, neboť jeho chování se stane

algoritmicky nepředvídatelné. U necíleného zakrytí můžeme předpokládat, že se objekt nadále pohybuje stejným směrem a rychlostí, jako dříve a je vysoká pravděpodobnost, že se opět objeví v předpokládané trajektorii. Tuto situace lze vyřešit prediktorem pohybu, definicí a číselnou kvantifikací pojmu „zakrytí objektu“. Pokud se objekt do určité doby neobjeví, je nutné celý algoritmus restartovat a operátor musí objekt opět sám vybrat, je-li to nutné nebo žádoucí.

Pohyb kamery má v důsledku negativní vliv na celý proces sledování. Z tohoto důvodu je nutné s pohybem kamery počítat a kompenzovat ho vhodnými prostředky. Větší výkyvy je možné kompenzovat mechanicky pomocí manipulátoru a menší pak digitálně. Chybná korekce pohybu kamery může vést i k chybnému určení polohy sledovaného objektu, proto je potřeba se tímto problémem zabývat.

Nejdříve je nutné definovat „zakrytí“ nebo „zmizení“ matematicky. Pravděpodobnostní funkce (43) definovaná v předchozí kapitole je odolná vůči částečnému zakrytí, ale úplné zmizení objektu neřeší, proto se budeme zabývat především problémem úplného zmizení, který je potřeba ošetřit jiným způsobem. Důležitým okamžikem, který je potřeba vyřešit, je detekce počátku mizení objektu. V našem případě lze říci, že objekt je plně zakryt, pokud pro nejlépe ohodnocenou částici platí

$$\sum_{(x,y)\in M} \min(\boldsymbol{M}_n, \boldsymbol{M}_0) < \gamma \sum_{(x,y)\in M} \boldsymbol{M}_0, \tag{44}$$

kde $\boldsymbol{M}_n$ je aktuální maska objektu (popředí), $\boldsymbol{M}_0$ je iniciální maska objektu (popředí), γ je předem definovaná prahová hodnota (prakticky volíme hodnotu $\gamma = 1$) a min lze popsat, podobně jako v předchozí kapitole, jako průnik obou masek. Konec úplného zakrytí objektu nastane tehdy, když je aktuální nejvyšší hodnota váhy větší než váha posledního pozorování, kdy objekt ještě nebyl zakryt.

V případě detekce zakrytí objektu se mění chování Bootstrap filtru tak, že se provádí pouze predikční krok s následujícími hodnotami parametrů pro model pohybu v čase n:

$$\sigma_x^2 = \frac{1}{N}\sum_{i=0}^{N} \bar{x}_{n-i} - \bar{x}_{n-i-1}, \tag{45}$$

$$\sigma_y^2 = \frac{1}{N}\sum_{i=0}^{N} \bar{y}_{n-i} - \bar{y}_{n-i-1},$$

$$\sigma_{vx}^2 = \sigma_{vy}^2 = 0.$$

Takovéto nastavení parametrů zapříčiní rozšiřování částic do prostoru od poslední pozitivní nalezené pozice objektu ve směru určeném predikcí v závislosti na stanovené rychlosti. Tento způsob detekce a zacházení s mizením objektů se v praxi jeví jako nejvhodnější řešení, které jsme byli schopni nalézt.

3.2.4 Digitální stabilizace pohybu kamery

V mnoha případech se bude kamera (tedy manipulátor) pohybovat. V takových případech je při současném sledování objektů nutné kompenzovat drobné chvění nebo i větší výkyvy v obraze způsobené pohybem kamery. Stabilizace je důležitým krokem, neboť jakýkoliv posun obrazů vůči sobě může znamenat změnu intenzity a tím i chybnou aktivaci adaptace modelu odčítání pozadí v závislosti na velikosti změny, což může mít nežádoucí vliv především v souvislosti s následným sledováním objektu. V našem systému je pro popis geometrického vztahu dvou po sobě jdoucích snímků model založený na afinní transformaci. Je tedy potřeba nalézt 6 parametrů, které jsou součástí matice afinní transformace $\boldsymbol{H}$.

Matice transformace $\boldsymbol{H}$ se počítá mezi dvěma po sobě jdoucími snímky, předpokládá se, že jde o homografii, tedy matice $\boldsymbol{H}$ má rozměr 3×3. Postup je následující:

1. Zmenšení snímku z původní velikosti $W \cdot H$ na velikost $rw \cdot rh$ ($rw < w$ a $rh < h$) z důvodu urychlení nalezení transformace $\boldsymbol{T}$ mezi snímky. Pro plnou velikost pak platí $\boldsymbol{H} = \boldsymbol{S}_2\boldsymbol{T}\boldsymbol{S}_1$, kde $\boldsymbol{T}$ je transformace zjištěná na zmenšeném obrázku a matice $\boldsymbol{S}_1$ a $\boldsymbol{S}_2$ jsou definovány jako

$$\boldsymbol{S}_1 = \begin{bmatrix} \frac{1}{rw} & 0 & 0 \\ 0 & \frac{1}{rh} & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad (46)$$

$$\boldsymbol{S}_2 = \begin{bmatrix} rw & 0 & 0 \\ 0 & rh & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

2. Následuje nalezení množiny význačných bodů T (viz [19] nebo kapitola 2.3.1) ve snímku $\boldsymbol{I}_n$ – jedná se především o rohy. Hrany nejsou pro účely sledování vhodné, neboť způsobují problém v lokálním měřítku. Jejich poloha se jeví stále stejná u některých druhů pohybu, především translační pohyb ve směru hrany není možné detekovat. Používá se funkce knihovny OpenCV `cvGoodFeaturesToTrack`.
3. Dále pomocí pyramidální verze algoritmu sledování optického toku Lucas-Kanade [20] nalezneme odpovídající body z aktuálního snímku ve snímku předchozím (viz kapitola 2.2.2).
4. Algoritmem RANSAC [40] stanovena matice afinní transformace (viz kapitola 2.1.3) mezi jednotlivými korespondujícími body. Pokud je dostatek reprezentativních bodů, pak tímto způsobem zaručíme odolnost vůči výběru nevhodného bodu (například bodu náležejícího pohybujícímu se objektu), který by mohl nevhodně ovlivnit výslednou matici, což zajistí odolnost vůči extrémním hodnotám. Používá se funkce OpenCV `cvFindHomography`.
5. Může se stát, že vypočítaná afinní transformace bude nepřijatelná (může způsobit například rozostření obrazu). Abychom takovým případům předešli, uplatníme jednoduchou penalizační funkci, která bere v úvahu polohu rohů snímku po transformaci a změnu velikosti diagonály s tím, že předpokládáme, že pohyb mezi následujícími snímky je pouze minimální. Výpočet penalizace získané transformace s využitím čtyř rohů snímku lze popsat takto: $\boldsymbol{a} = (0,1,0)^T$ – levý horní roh, $\boldsymbol{b} = (1,1,0)^T$ – pravý horní roh, $\boldsymbol{c} = (0,0,0)^T$ – levý spodní roh, $\boldsymbol{d} = (1,0,0)^T$ – pravý spodní roh. Změna jejich polohy je úměrná penalizaci:

$$f(\boldsymbol{H}) = f_t(\boldsymbol{H}) + \gamma f_z(\boldsymbol{H}), \qquad (47)$$

kde $f(\boldsymbol{H})$ je penalizační faktor, $f_t(\boldsymbol{H})$ je penalizační faktor translačního pohybu a $f_z(\boldsymbol{H})$ je penalizační faktor změny měřítka s vhodně zvolenou váhou γ, vše v závislosti na vypočítané matici transformace $\boldsymbol{H}$. Experimentálně byla určena hodnota váhy $\gamma = 20$. Penalizační faktor translační je definován jako

$$f_t(\boldsymbol{H}) = \|\boldsymbol{H}\boldsymbol{a} - \boldsymbol{a}\| + \|\boldsymbol{H}\boldsymbol{b} - \boldsymbol{b}\| + \|\boldsymbol{H}\boldsymbol{c} - \boldsymbol{c}\| + \|\boldsymbol{H}\boldsymbol{d} - \boldsymbol{d}\|, \qquad (48)$$

kde $\|\cdot\|$ je Euklidovská norma výsledného vektoru. Penalizační faktor změny měřítka je definován jako součet absolutních hodnot rozdílu původní a transformované Euklidovské normy diagonál:

$$f_z(\boldsymbol{H}) = \left|\sqrt{2} - \|\boldsymbol{H}\boldsymbol{a} - \boldsymbol{H}\boldsymbol{d}\|\right| + \left|\sqrt{2} - \|\boldsymbol{H}\boldsymbol{c} - \boldsymbol{H}\boldsymbol{b}\|\right|. \qquad (49)$$

Pokud je penalizace vyšší než předem stanovená hodnota prahu, je nový snímek zahozen, tj. neprobíhá následující sledování objektu a měření dálky, pouze se navýší počet nepodařených po sobě jdoucích odhadů transformace. A opět, je-li tento počet nepoužitelných odhadů transformace vyšší než další předem stanovená hodnota, dojde ke kompletnímu restartu modelu pozadí a navazujících algoritmů.

Pokud je penalizace nižší než předem stanové konstanta, tak se transformuje model pozadí podle zjištěné transformace $\boldsymbol{H}$ na aktuální snímek a provede se krok aktualizace modelu pozadí, přičemž případné nově přidané oblasti jsou inicializovány stejně jako v kroku inicializace modelu pozadí.

3.2.5 Model pozadí a jeho použití

Metoda odčítání pozadí je ideální prostředek pro detekci pohybu v obraze, má však nevýhody, které vyřadily tuto metodu z výběru vhodných kandidátů na sledování objektů. Nicméně i pro algoritmus založený na částicových filtrech, který byl vybrán jako nejvhodnější, se pozadí a jeho model uplatní v lokálním měřítku jako detektor pohybu. Pro tyto účely je potřeba navrhnout takový model pozadí, který bude splňovat náročné podmínky zadání.

Ideální model pozadí by měl mapovat pouze „statické“ prvky scény. Ovšem rozpoznání scény a objektů v ní náležících patří stále k otevřeným problémům počítačového vidění. Proto se tento problém řeší přístupy, které nejsou strukturálně/objektově orientované, ale pracují pouze s jasovou funkcí jednotlivých obrazových elementů. Velmi častý je tak pravděpodobnostní přístup, přičemž se uplatňují principy přejaté z teorie filtrace. Obecně platí, že model pozadí by měl být adaptabilní, robustní vůči jasovým změnám (mrak/slunce), opakujícím se změnám (vlnky na vodě, listí stromů ve větru), výpočetně nenáročný a invariantní vůči pohybu kamery. Důraz na jednotlivé vlastnosti použitého modelu se odvíjí od konkrétní aplikace.

Mezi často používané a ne příliš robustní modely patří průměr z plovoucího okna, medián z plovoucího okna a dříve přijatý snímek. Nevýhoda těchto modelů je jednak ve výpočetní náročnosti (medián, průměr) a jednak neschopnosti adaptace (dříve přijatý snímek). Velice oblíbeným přístupem je rovněž model Gaussovský, kde je každý obrazový element modelu reprezentován Gaussovou křivkou, tj. střední hodnotou a variancí.

Na model pozadí a jeho funkčnost jsou kladeny základní požadavky vyplývající z globálního zadání, ale i z požadavků, které vyplynuly z řešení problému. Požadavky lze shrnout do následujících bodů:

1. Je potřeba zajistit maximální rychlost výpočtu a minimální paměťové nároky, neboť paměť GPU, které bude využito pro akceleraci algoritmu, je v tomto směru silně omezující, což je také hlavní důvod proč se nehodí konvenční modely pozadí.
2. Algoritmus vytvářející model pozadí musí být schopen rychlé adaptace na malé změny způsobené změnou osvětlení scény, musí tedy být schopen reagovat na globální jasové změny.
3. Algoritmus musí být schopen pomalé adaptace na velké změny způsobené pohyblivými objekty, protože změny způsobené pohyblivými objekty jsou krátkodobé a neměly by žádným způsobem ovlivnit model.
4. Je nutné brát v úvahu fakt, že kamera může být pohyblivá, což je v principu velký problém pro vytvoření vhodného modelu.

Pro model pozadí je potřeba stanovit vhodnou reprezentaci oblasti. Každá sledovaná oblast (objekt) je množinou obrazových bodů (pixelů), což není dostatečná informace pro účely vytváření modelu pozadí. Z tohoto důvodu bude každý bod reprezentován normálním rozložením, tedy kromě jasové složky bude obsahovat informaci o střední hodnotě a varianci. Body jsou lokálně izolovány, platí tedy, že při aktualizaci se nebere v úvahu okolí bodu. Z hlediska implementace půjde o dvourozměrné pole o stejných rozměrech,

jako má snímek z kamery (uložený v GPU). Hodnoty pozorování, především tedy jasová informace, jsou normalizovány do intervalu $< 0{,}1 >$. Původní snímek $\boldsymbol{I}_n(x, y)$ je tedy transformován na modelový prostor snímku $\boldsymbol{I}_{B,n}(x, y)$ takto:

$$\boldsymbol{I}_{B,n}(x, y) = (\mu_{n-1}, \sigma_{n-1}, \mu_n, \sigma_n), \tag{50}$$

kde μ_{n-1} je střední hodnota a σ_{n-1} je rozptyl modelu pozadí platného do času $n-1$, μ_n je střední hodnota aktuálního pozorování (snímku, typicky přímo hodnota jasu daného bodu v čase n, tedy $\mu_n = \boldsymbol{I}_n(x, y)$) a σ_n je variance aktuálního pozorování definována jako

$$\sigma_n = \mathrm{e}^{\alpha(|\mu_{n-1} - \mu_n| - \sigma_{n-1})}. \tag{51}$$

Prvním krokem získání modelu pozadí je inicializace modelu. Prvotní informace pocházejí z prvního (inicializačního) snímku $\boldsymbol{I}_0$ tak, že střední hodnota μ_{n-1} bodu (x, y) modelu v čase $n = 0$ (tedy pro μ_{-1}) je dána přímo jasovou složkou snímku $\boldsymbol{I}_0(x, y)$ a jeho variance je rovna konstantě β, která byla experimentálně stanovena na hodnotu $\beta = 0{,}05$. Pro iniciální stav tedy platí:

$$\boldsymbol{I}_{B,0}(x, y) = \left(\mu_{-1} = \boldsymbol{I}_0(x, y), \sigma_{-1} = \beta, \mu_0 = \boldsymbol{I}_0(x, y), \sigma_0 = \mathrm{e}^{\alpha(|\mu_{-1} - \mu_0| - \sigma_{-1})}\right). \tag{52}$$

S každým novým snímkem je potřeba aktualizovat model pozadí. Možností pro aktualizaci pozadí je několik, například standardním postupem pro aktualizace střední hodnoty pro nový snímek v čase n reprezentovaný novým pozorováním popsaným hodnotami μ_{new} a σ_{new} lze psát:

$$\mu_n = \frac{\mu_{n-1}\sigma_n + \mu_{new}\sigma_{n-1}}{\sigma_n + \sigma_{n-1}}, \tag{53}$$

$$\sigma_n = \frac{1}{\frac{1}{\sigma_{n-1}} + \frac{1}{\sigma_{new}}}.$$

Hodnota μ_{new} bývá přímo rovna intenzitě bodu aktuálního snímku v čase n, tedy $\mu_{new} = \boldsymbol{I}_n(x, y)$, a hodnota variance je dána vztahem $\sigma_{new} = \mathrm{e}^{\alpha(|\mu_{n-1} - \mu_{new}| - \sigma_{n-1})}$ pro daný čas n. Tyto hodnoty pak slouží pro aktualizaci parametrů modelu.

Problémem klasického způsobu úpravy střední hodnoty a variance je, že se pro aplikaci sledování objektu a metodu stanovení modelu pozadí příliš nehodí. Abychom dosáhli prakticky použitelných výsledků, je nutné postup úpravy (především variance) modifikovat. Postup lze shrnout do následujících kroků:

2. Střední hodnota pozorování bodu aktuálního snímku je rovna přímo jeho intenzitě v aktuálním snímku $\mu_{new} = \boldsymbol{I}_n(x, y)$.
3. Variance pozorování je závislá na vzdálenosti jasových složek vzoru (modelu) a aktuálního snímku $\sigma_{new} = \mathrm{e}^{\alpha(|\mu_{n-1} - \mu_{new}| - \sigma_{n-1})}$.
4. Na základě těchto hodnot pozorování je potřeba aktualizovat střední hodnotu modelu pozadí klasickým způsobem výpočtu aktualizace střední hodnoty u_n podle rovnice (53).
5. Aktualizace variance se liší od klasické formy v závislosti na tom, je-li vzdálenost jasových hodnot vzoru a obrazu větší než variance modelu. Na základě vzdálenosti pak dochází k aktualizaci popsanou jako:

$$\sigma_n = \begin{cases} \kappa\sigma_n, & |\, \mu_n - \mu_{new} \,| > \sigma_n \\ \dfrac{\sigma_{new}\sigma_n}{\sigma_{new} + \sigma_n}, & |\, \mu_n - \mu_{new} \,| \leq \sigma_n \end{cases} \tag{54}$$

$$\sigma_n = \sigma_n + \delta$$

kde κ je multiplikativní a δ je aditivní šum. Tyto hodnoty volíme $\kappa = 1{,}02$ a $\delta = 25 \cdot 10^{-5}$ a jejich důsledkem je mírné zvýšení hodnoty variance pro daný bod v každém případě a znatelnější nárůst v případě, že je vzdálenost jasů obou bodů větší, než je původní hodnota variance.

Důsledkem tohoto způsobu úpravy adaptace modelu pozadí je snížená citlivost na skokové změny, které jsou vzaty v úvahu až v případě dlouhodobého setrvání v daném bodě, což zamezuje chybné adaptaci modelu na změny, které nejsou dlouhodobé (například pohyb objektu, který nesouvisí s pozadím). Naproti tomu pomalé jasové změny jsou zaznamenány téměř okamžitě, protože jsou vyhodnocovány jako změny světelných podmínek, které je potřeba brát v potaz při aktualizaci modelu pozadí.

3.3 Paralelizace algoritmu sledování objektu

Aplikace pro kamerový systém je vyvíjena jako vícevláknová aplikace. Synchronizace probíhá na datové úrovni pomocí časových známek, které slouží jako jedinečné identifikátory dat. To znamená, že každý modul/knihovna si musí zajistit svoji vnitřní konzistenci nezávisle na pořadí příchozích informací. Obecně platí, že aplikace nemůže zajistit v sekvenci volání neklesající monotónnost těchto identifikátorů (uživatel může požadovat korekci v minulosti, tj. ne jen korekci posledního stavu apod.).

Komunikaci v systému dále přirovnáme ke dvojici senzor-pozorování. Senzor zde představuje objekt, který může být generátorem pozorování (kamera), nebo filtrem/agregátorem jiných pozorování (lokalizační jednotka – generuje pozorování o pozici). V naší terminologii představuje pozorování výstupní informaci ze senzoru. Každé pozorování sebou nese veškerou informaci platnou v době pozorování (např. pozici kamery v čase sejmutí snímku, nastavení optické soustavy, apod.) a případně další informace tak, aby veškerá data byla v pozorování dostupná a nebyla vyžadována komunikace se samotným senzorem. Tím je dosaženo nezávislosti mezi odesílatelem a příjemcem, generičnosti a možnosti záznamu. Každé pozorování navíc obsahuje identifikátor, který je zároveň časovou známkou nesoucí informaci o době jeho vzniku (v případě agregovaného pozorování pak časovou známku platnosti těchto dat, nikoliv čas vytvoření instance třídy) a jméno senzoru, který toto pozorování vytvořil. Pozorování je obecně struktura určená ostatním příjemcům pouze ke čtení (*read only*). Na jednotlivá pozorování ukazují takzvané „chytré ukazatele“, které zajistí dealokaci paměti v případě potřeby a příjemci se tak o tento problém nemusí starat.

3.3.1 Distribuce výpočtů mezi procesor a GPU

Jednotlivé výpočty jsou rozmístěny mezi procesor a grafický akcelerátor. Některé výpočty lépe řeší sériový algoritmus běžící na procesoru, některé lze s výhodou paralelizovat. Algoritmy, které je vhodné implementovat na GPU, jsou výpočet obrazové transformace H (střední hodnota a variance) a aktualizace modelu pozadí algoritmu. Naopak, na CPU je vhodné ponechat například výpočet optického toku a další algoritmy, které výrazně netěží z paralelního přístupu a cena za složitost implementace je zbytečně vysoká v porovnání s výkonnostním ziskem takového řešení.

Všechny konstanty byly zjištěny experimentálně. Jejich změnou je možné nastavit rychlost adaptace respektive její průběh (například délku „ploché“, necitlivé fáze, rychlost adaptace ve fázi učení nebo citlivost modelu).

3.3.2 Modul sledování objektů a jeho optimalizace

Algoritmus sledování objektů tvoří samostatný modul, který běží ve vlastním vlákně. Komunikace s modulem probíhá pomocí signálů a slotů, je tedy bufferovaná a vláknově bezpečná. Modul přijímá

obrazová pozorování, požadavky na přidání cílů, jejich korekci a odstranění. Sám generuje pozice cílů, jejich stav ke zpracovanému pozorování a informuje o korekci, přidání či odstranění cíle. Každý cíl má jedinečný identifikátor a jeho stav sestává z kombinace identifikátoru cíle a identifikátoru pozorování, ke kterému jsou data platná.

Modul sledování objektů si uchovává data o posledních 100 příchozích pozorování v plném rozsahu (tj. samotný snímek a příslušné pozadí) a umožňuje tedy provádět korekce v minulosti, přičemž v případě korekce dojde k automatickému dohledání cíle zpět do přítomnosti. Velikost historie je omezena velikostí paměti GPU, kde jsou data uložena (nový přesun mezi CPU-GPU je nemyslitelný, protože je časově velmi náročný a aplikace by tak velmi utrpěla z hlediska reálného času). Pokud je identifikátor obrazového pozorování korekčních dat již mimo historii, je tato korekce odmítnuta.

Pro práci s modulem sledování objektů je vytvořena třída, která definuje rozhraní pro komunikaci. Komunikace je asynchronní, datová přes signály a sloty. Po příchodu nového snímku do slotu `imageObservation` (tedy po volání této funkce), provede modul následující kroky:

1) aktualizace vnitřního modelu pozadí,
2) sledování všech cílů v seznamu cílů v novém snímku,
3) emise nových stavů cílů (tedy generování signálu `targetTracked`) v případě, že je cíl úspěšně sledován.

V případě, že dojde ke korekci (především polohy) cíle uživatelem, dojde k volání funkce `correctTarget` (aktivace slotu)., která vykoná následující:

1) opraví historii cíle,
2) znovu inicializuje vnitřní struktury (například částicový filtr),
3) zpětně dosleduje cíl do přítomnosti,
4) emituje aktualizovaný stav cíle generováním příslušného signálu.

Ostatní funkce – tedy sloty přidání cíle a odstranění cíle – a příslušné zpětnovazební signály – tedy potvrzení přidání cíle, odebrání cíle a sledování cíle – nepotřebují další vysvětlení.

Algoritmus jako celek není možné implementovat na GPU, nicméně vlastnosti dílčích částí algoritmu umožňují akceleraci těch částí, které v principu mohou pracovat paralelně. Tyto dílčí algoritmy jsou tedy paralelizované na GPU s tím, že modul sledování objektu sám využívá pouze jedno vlákno, které se stará o obslužné rutiny a předává data GPU. Pro další popis platí, že vstupní snímky jsou uloženy v paměti GPU.

Aktualizace vnitřního modelu

Aktualizace vnitřního modelu pozadí musí předcházet samotnému procesu sledování objektu. Nejdříve je vypočítána transformace mezi posledním a aktuálním snímkem, jak bylo popsáno v kapitole 3.2.5 zabývající se teorií modelu pozadí. Dále se provede mapování modelu pozadí na nový snímek na základě zjištěné mezisnímkové transformace, čímž je eliminován případný vliv pohybu kamery, a následně se provede samotná aktualizace modelu.

Výpočet transformace obou snímků probíhá na CPU, neboť bylo empiricky zjištěno, že je tento výpočet na CPU při vhodné velikosti obrázku rychlejší než na GPU. Abychom dodrželi zmíněnou podmínku „vhodná velikost obrázku“, musíme změnit rozlišení vstupního snímku na akceptovatelnou hodnotu. Změna velikosti probíhá na GPU, kde jsou již připraveny rychlé algoritmy, které tuto změnu provedou. Výsledný snímek je pak uložen do paměti CPU a volá se funkce pro zjištění transformace – tedy funkce, která vypočítá příslušnou matici homografie. Algoritmus lze v pseudokódu zapsat takto:

```
cv::Mat current;
cv::gpu::resize(obs_F, res_current_F_,  hSize_);
res_current_F_.convertTo(res_current_U_, CV_8UC1, 255.0f);
```

```
res_current_U_.download(current);
cv::Mat T = homography(previous_, current);
```

Výpočet transformace (tedy matice homografie) probíhá v následujícím sledu kroků na CPU:

1) detekují se význačné body v předchozím snímku,
2) tyto body jsou sledovány do nového snímku pomocí algoritmu optického toku,
3) mezi takto určenými množinami bodů se vypočítá matice homografie.

Pokud se podařilo transformaci (matici homografie) najít, provede se kompenzace změny rozlišení obrázku a získá se výsledná transformace pro plné rozlišení:

```
      cv::Mat H = r2_ * T * r1_;
```

Po zjištění transformační matice dojde k mapování předešlého stavu do aktuálního snímku pomocí GPU. Původně zamýšlené využití funkce `warp_perspective` z knihovny OpenCV bylo znemožněno chybami, které tato funkce v knihovně má, proto bylo potřeba tuto funkci znovu implementovat. Algoritmus pro vzájemné mapování obou stavů pak vypadá takto:

```
    Img *img_in = img_create_header(
        gmean_.cols, gmean_.rows, gmean_.step, (float *) gmean_.data);
    Img *img_out = img_create_header(
        tmp.cols, tmp.rows, tmp.step, (float *) tmp.data);
    // Warp gmean.
    warp_perspective(img_in, img_out, m_device_->data_);
    tmp.copyTo(gmean_);
    // Warp gvar.
    img_in->data_ = (float *) gvar_.data;
    warp_perspective(img_in, img_out, m_device_->data_);
    tmp.copyTo(gvar_);
    // Warp mask.
    warp_perspective_get_mask(
        gmask_.data, gmask_.cols, gmask_.rows, gmask_.step,
        m_device_->data_, 0, 255);
    // Release allocated resources.
    img_free_header(&img_in);
    img_free_header(&img_out);
```

Samotná aktualizace modelu se provádí na GPU pomocí elementárních funkcí OpenCV, které jsou optimalizovány pro použití s grafickým akcelerátorem. Díky chybné implementaci zmiňované funkce `warp_perspective` knihovny OpenCV jsme byli nuceni tuto funkci implementovat. Pro svou činnost využívá funkce texturovací jednotku, neboť pro svoji činnost potřebuje data, která získáme interpolací hodnot bodů ve snímcích. Za normálních okolností bychom interpolaci museli vypočítávat vlastními silami, ale vlastností jádra GPU nám umožní využít faktu, že textury (tedy snímky v paměti textur) jsou interpolovány automaticky, což opět zrychlí celý výpočet.

Sledování cíle

Algoritmus sledování cíle založený na částicových filtrech můžeme přepsat do podoby pseudokódu takto:

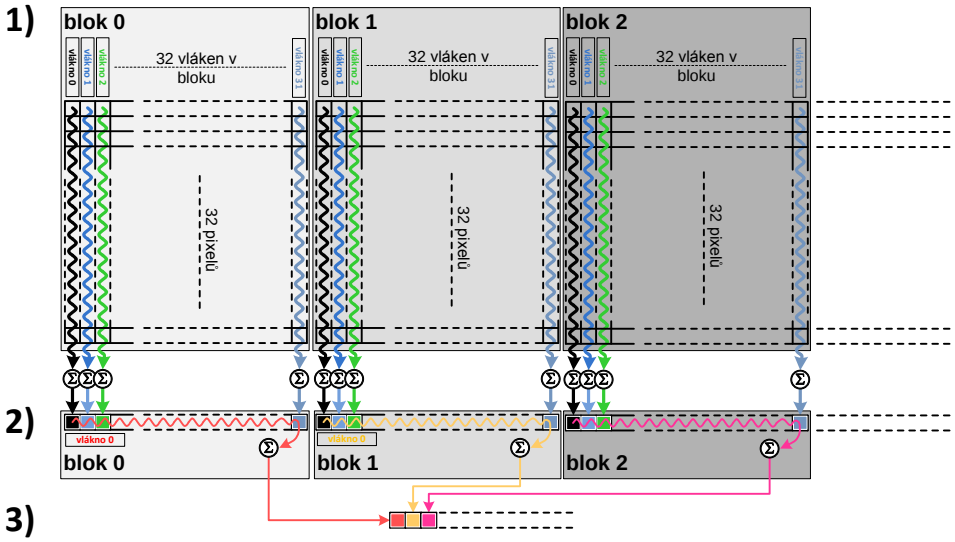
```
Vytvoření částice
Nastavení parametrů částice
Inicializace částice

Pro každé pozorování {
    Pro každý cíl {
        particleTransition;
        particleEvaluation;     // Výpočet na GPU
        particleNormalize;
        particleResample;
    }
}

Uvolnění částic
```

Na GPU běží nejnáročnější část celého výpočtu – stanovení ohodnocení částice – tedy výpočet *fitness* funkce. Kroky predikce, normalizace a převzorkování, tj. všechny ostatní operace týkající se částic, mimo evaluaci, jsou spouštěny sériově na CPU. Experimentálně jsme zjistili, že při počtu částic pod cca 10 000 nemá význam implementovat celý částicový filtr na GPU.

Ponechejme stranou veškeré doprovodné algoritmy, které jsou implementovány na CPU, a zaměřme se pouze na samotnou paralelizaci evaluace částic. K tomu, abych mohli provést evaluaci částic na GPU, musíme do paměti GPU nahrát aktuální stav částicového systému, aktuální snímek, aktualizovaný model pozadí, reprezentaci cíle a transformační matici popisující transformaci ze souřadného systému cíle (částicového filtru) do souřadného systému aktuálního snímku (souřadný systém cíle je nezávislý na souřadném systému snímku, abychom oddělili pohyb kamery od pohybu cíle). Funkce, která nachystá data pro volání jádra funkce GPU, má tvar:

```
void PFTracker::evaluateParticles(
    cv::gpu::GpuMat img, cv::gpu::GpuMat fg,
    cv::gpu::GpuMat target, cv::gpu::GpuMat mask,
    CvParticle *p, cv::Mat H)
```

Aby byl výpočetní čas vyhodnocení částice každého jádra konstantní stejně jako i velikost, má každá reprezentace cíle předem danou velikost (v našem případě je to 32×32 pixelů). Průchod touto maticí je paralelizován. Ke každému pixelu v této matici je nalezen odpovídající pixel v aktuálním snímku a je proveden pixelově orientovaný výpočet ohodnocení. Abychom mohli ukázat samotnou paralelizaci, musíme nejdříve objasnit pohled na předaná data, situace je graficky znázorněna na následujícím obrázku.



Obrázek 3.1: Rozdělení výpočtu vyhodnocování částic do bloků a vláken. Krok 1: rozměr obrazového vzoru každé částice je 32×32 pixelů, každé vlákno zpracovává jeden sloupec a vláken je 32. Krok 2: částečné součty je potřeba agregovat (dokončuje další kód jádra). Krok 3: výsledek je uložen v paměti a nahrán zpět do CPU.

Každá možná konfigurace, tj. částice, představuje jeden blok. Počet bloků ve směru osy x je roven počtu částic, rozměr ve směru osy y a z jsou rovny jedné. Počet vláken v jednom bloku je roven šířce obrázku cíle (ve skutečnosti je to definováno obecněji, aby maximální velikost reprezentace nebyla omezena, ale kvůli zjednodušení popisu budeme uvažovat tuto reálnou variantu) – v našem případě je tedy počet vláken 32 ve směru osy x (ostatní dimenze jsou rovny 1). Každé vlákno v bloku zpracovává právě jeden sloupec. Abychom mohli adresovat odpovídající si pixely cíle a možného výskytu cíle v aktuálním snímku, musíme nejdříve vypočítat transformační matici $\boldsymbol{F}$. Transformační matice $\boldsymbol{F}$ je dána kompozicí transformace mezi souřadným systémem snímku a souřadným systémem cíle (matice $\boldsymbol{H}$, která je pro všechny možné

varianty/částice shodná) a pak transformační maticí vzniklou vlastní konfigurací částice (matice $\boldsymbol{T}$), tj. $\boldsymbol{F} = \boldsymbol{H} \cdot \boldsymbol{T}$. Protože hodnoty v této matici jsou využívány při každé adresaci pixelu v aktuálním snímku, je tato matice uložena ve sdílené paměti (modifikátor `__shared__`). Následně každé vlákno provede pixelově orientovaný výpočet dle popsaného algoritmu a uchovává průběžný výsledek (je využita asociativita operátoru sčítání), čímž získáme pole částečného ohodnocení, a to pro jednotlivé sloupce (viz první část předchozího obrázku). Tato částečná řešení je potřeba znovu agregovat. Implementačně se jedná o volání dalšího jádra, které tuto operaci provede.

Aby byly využity teselovací jednotky a tím zajištěno využití hardwarové bilineární interpolace při přístupu k obrazovým datům, je aktuální snímek považován za texturu. Další výhodou je rychlejší přístup k datům. Výsledek procesu je nahrán zpět do paměti CPU, kde se provede převzorkování částic na základě jejich ohodnocení.

3.4 Výsledky experimentů

S hotovou aplikací jsme provedli několik experimentů. Prvním z nich bylo ověření funkce modifikovaného algoritmu aktualizace modelu pozadí pro účely odčítání pozadí při sledování pohybu objektu ve videu, kdy jsme ji porovnali s metodami *Static Frame Difference*, *Weighted Moving Mean* [35] a *Simple Gaussian* [36], které jsou reprezentativními zástupci běžně využívaných metod.

Nastavení parametrů metod bylo ponecháno na referenčních hodnotách (hledání a nastavení jiných hodnot by vyžadovalo velmi detailní studium těchto metod a nastavené parametry jsou v drtivé většině případů optimální pro daný algoritmus – ale nemusí být optimální pro daný způsob využití), kromě metody *Simple Gaussian*, kde byl změněn koeficient „*Rating rate*“ z hodnoty 18 na 50, čímž bylo dosaženo rychlejší adaptace pro objektivnější porovnatelnost výsledků.

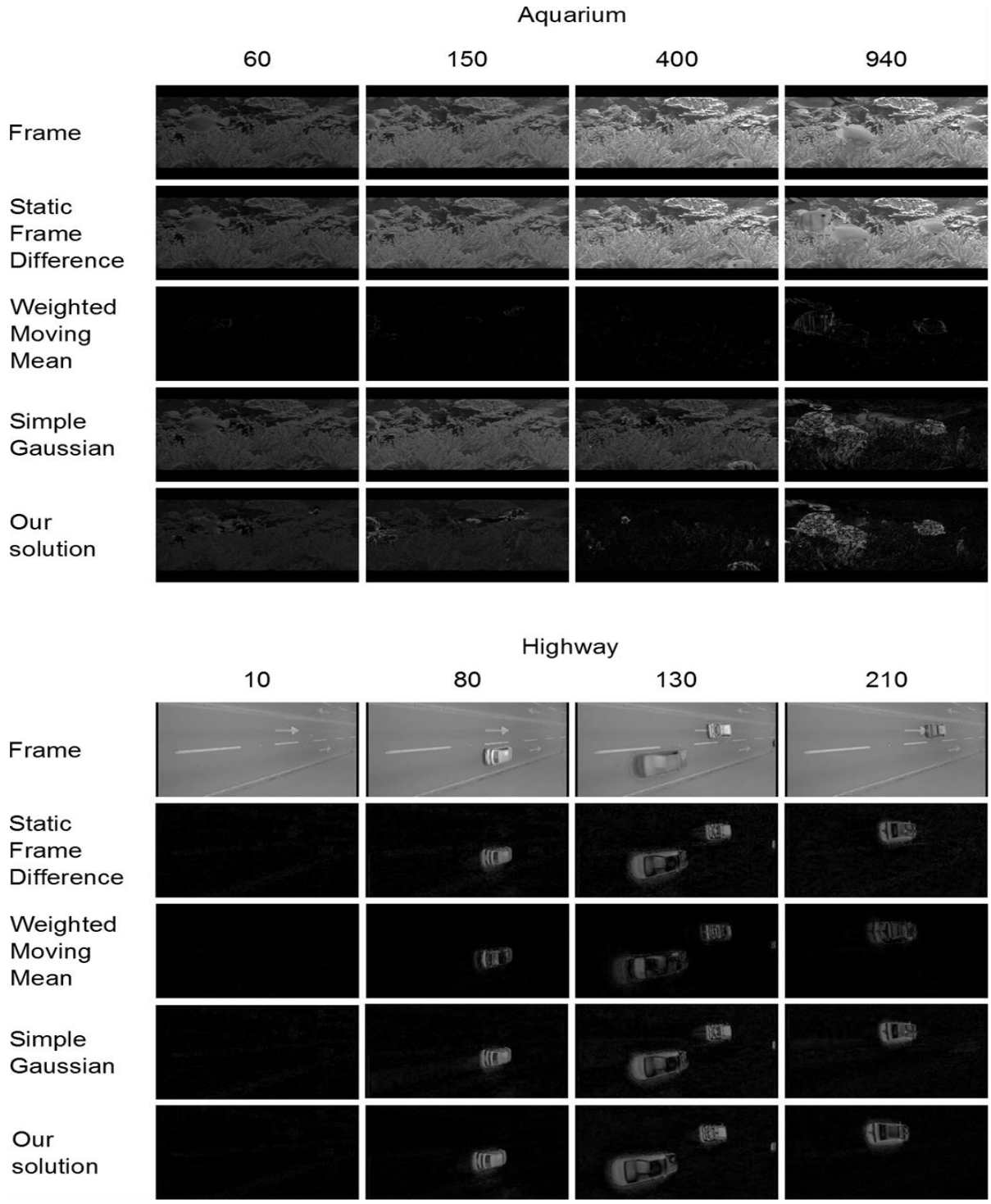
Porovnání metod bylo provedeno na dvou testovacích množinách videí – na datových sadách *dálniční provoz* a *podmořský svět*. Obě tyto datové sady byly pořízeny statickou kamerou, neuplatní se tedy jedna z nejdůležitějších výhod našeho přístupu – schopnost adaptace i v případě snímání pohybující se kamerou. Důvodem výběru záběru ze statické kamery je ten, že metody pracující s odčítáním pozadí se velmi často nezabývají situací, kdy se kamera pohybuje (což byl také hlavní důvod, proč nebyly tyto metody primárně použity pro účely sledování objektů, ačkoliv vykazují velmi dobré výsledky v případě statických kamer).

Obě datové sady lze charakterizovat stručně takto:

- Datová sada *dálniční provoz* je pořízena statickou kamerou, na záběrech jsou rychle se pohybující objekty (vzhledem ke snímkovací frekvenci kamery), scéna je relativně stabilně osvětlena (nedochází tedy k výrazným změnám jasu celé scény) a objekty vrhají obrysové stíny.
- Datová sada *podmořský svět* je také pořízena statickou kamerou, na záběrech jsou rychle i pomalu se pohybující objekty (vzhledem ke snímkovací frekvenci kamery), vyskytují se zde chaoticky oscilující pohybující drobné objekty, světelné podmínky v záběrech se dynamicky mění s lineárním průběhem změn. Stíny nejsou viditelné.

Úkolem algoritmu je detekce pohybujících se objektů. Výstupem většiny netriviálních modelů by měla být maska objektu (tj. totéž co objekt popředí, maska popředí) a to v binární reprezentaci s tím, že maska nabývá hodnoty 0 pro body náležející pozadí a hodnoty 1 pro body náležející detekovanému objektu (popředí). K získání této masky popředí využívá většina metod zpravidla porovnání aktuálního jasového rozdílu s předem stanovenou (ale i dynamicky určovanou) prahovou hodnotou. To však nemusí být aplikováno striktně jen na absolutní rozdíl aktuálního snímku od modelu pozadí v jasové složce, ale mohou využívat i dalších spočtených příznaků (např. variance v případě pravděpodobnostního přístupu). Z toho

plyne, že porovnání absolutní diference v jasové složce mezi modelem pozadí a aktuálním snímkem není určujícím faktorem kvality modelu pro detekci pohyblivých objektů, avšak vlastní algoritmus sledování dále pracuje právě s vahami získanými absolutní jasovou diferencí modelu a aktuálního snímku, proto je pro nás toto srovnání rozhodující.



Obrázek 3.2: Vizuální srovnání s jinými konvenčními modely pozadí na dvou datových sadách – *podmořský svět* a *dálniční provoz*. Výhody našeho řešení jsou patrné zejména na datové sadě *podmořský svět*, kde si lépe poradí jak se změnou světelných podmínek scény, tak i s pohyblivými objekty.

Vizuální srovnání s jinými konvenčními modely pozadí na obou datových sadách ukazují výhody našeho řešení zejména na datové sadě *podmořský svět*, kde si lépe poradí jak se změnou světelných podmínek

scény (snímek 150 a 400), tak i s pohyblivými objekty (nevznikají nežádoucí artefakty, kontury jsou zřetelné, snímek 940).

Celý navržený systém sledování objektu byl testován a hodnocen na třech velice odlišných videosekvencích, které označíme jako A, B a C. Tato videa byla pořízena statickou i pohyblivou barevnou kamerou (barevnou kameru jsme zvolili proto, abychom mohli algoritmus porovnat i s algoritmy, které využívají barevnou informaci). Odlišnosti jednotlivých videí spočívají v kompozici scény, typu pohybu a počtu možných zmizení sledovaných objektů (viz následující tabulka). Ve všech experimentech byla na počátku definována pouze poloha objektu, který chceme sledovat, a jeho velikost.

Charakter sekvencí použitých pro účely testování algoritmů sledování pohybu					
	Pohybující se kamera	Změny měřítka	Zakrytí objektu	Rigidní	Počet snímků
A	Ano	Ne	Ano	Ne	437
B	Ano	Částečně	Ne	Ne	370
C	Ne	Ne	Ano	Ano	200

Abychom mohli naši metodu porovnávat, implementovali jsme další tři algoritmy:

- *Částicový filtr* založený na *monochromatickém* a *barevném* histogramu se slabým modelem pohybu a s mírou vzdálenosti pro hodnotící funkci pravděpodobnosti definovanou jako průnik histogramů. Barevná verze je označena jako PC a monochromatická jako PG.
- *Korelační* metoda založená na relativně jednoduchém principu porovnávání vzoru a obrazu jejich vzájemnou korelací. Tento algoritmus je označen jako TC a je převzat z *VIVID Tracker Testbed Package* [37][38].

Částicové filtry byly shodně inicializovány s počtem 100 částic a se stejnou počáteční variancí. Jako míru chyby sledování objektu v čase n byla použita Euklidovská vzdálenost mezi algoritmem určenou pozicí cíle $(\bar{x}_n, \bar{y}_n)$ a předem stanovenou (požadovanou, skutečnou) polohou objektu (x_n^G, y_n^G) normalizovánou vzhledem k šířce snímku w (ve směru osy x) a jeho výšce h (ve směru osy y). Chyba je označena ε a definována jako

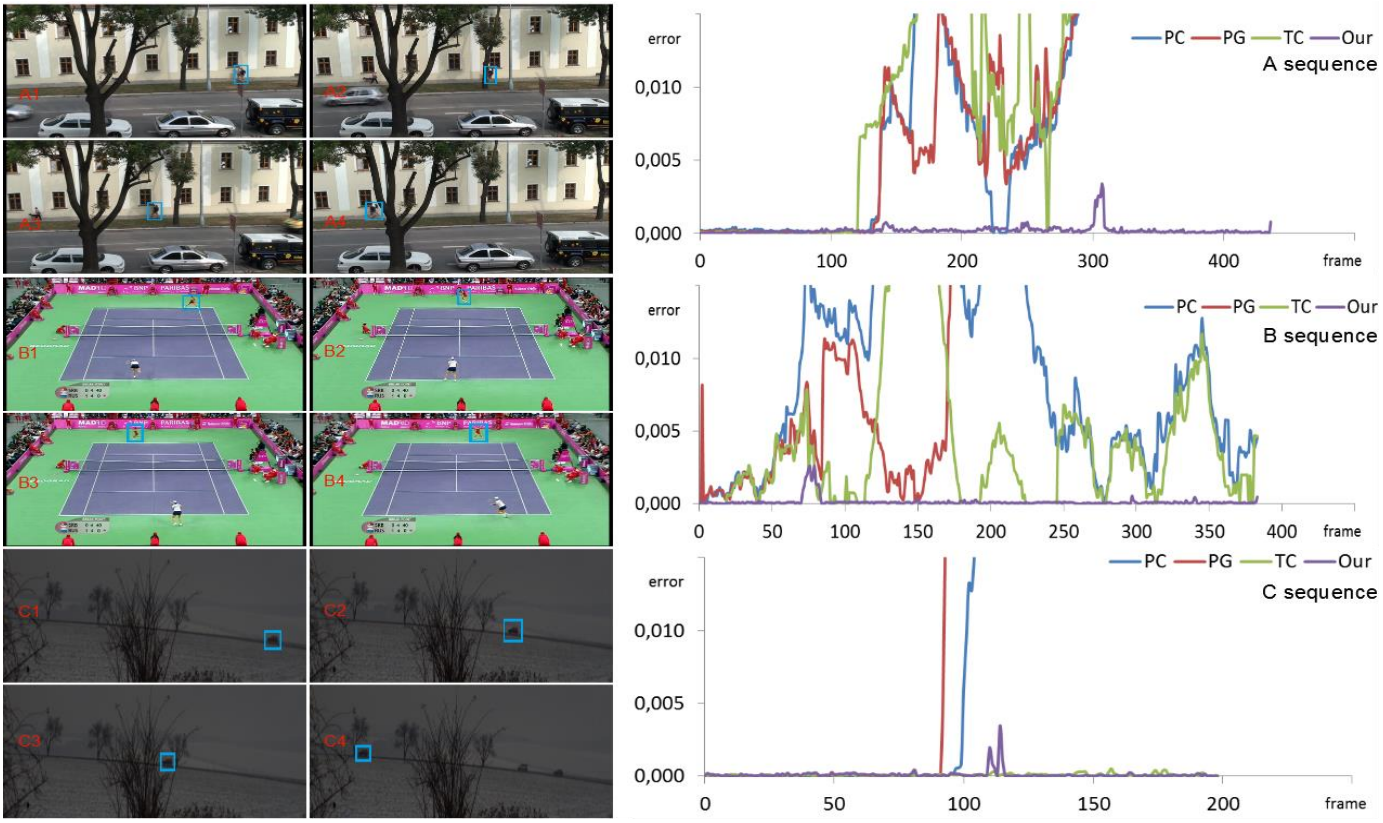
$$\varepsilon(\bar{x}_n, \bar{y}_n) = \left(\frac{\bar{x}_n - x_n^G}{w}\right)^2 + \left(\frac{\bar{y}_n - y_n^G}{h}\right)^2. \tag{55}$$

Výsledky měření shrnuje následující tabulka a chování chyby v průběhu jednotlivých sekvencí je zřetelné z grafu na následujícím. Náš algoritmus se chová lépe především v situacích, kdy dochází buď k pohybu kamery, k zakrytí objektu nebo k oběma jevům zároveň. Chyba stanovení pozice se pohybuje blízko nuly a jen v případě zmizení objektu se chyba chvilkově zvýší, ale jakmile se objekt znovu objeví na scéně, vrátí se algoritmus do normálních hodnot. Důležitá je především stabilita algoritmu v různých situacích. Ostatní algoritmy se ze ztráty objektu většinou nevzpamatují (především na videu A), algoritmus TC se při ztrátě viditelnosti objektu na videu C dokáže vzpamatovat. Z hlediska výběru vhodného algoritmu nás zajímají především videa A a C, protože jsou snímána za reálných podmínek a představují skutečnou ukázku nasazení systému, tedy v podstatě vcelku přesně simulují využití sledovacího systému v praxi. Důležitou výhodou našeho algoritmu v porovnání s algoritmem TC je především schopnost práce s černobílými snímky a schopnost řešit sledování objektu i v případech, kdy se pohybuje kamera a když objekt zmizí za podobně barevným, větším objektem, což tomuto algoritmu problémy činí (viz grafy analýzy videí A a B).

Výsledné hodnoty testování algoritmů sledování pohybu – součet chyby pro všechny snímky

video	PC	PG	TC	Náš algoritmus
A	11,761	11,791	13,133	**0,001**
B	0,049	0,052	0,022	**0,003**
C	9,041	12,762	**0,019**	0,023

Algoritmus sledování objektů jsme porovnávali s ostatními algoritmy na barevných záznamech především proto, že ostatní algoritmy pro svoji činnost vyžadují barevnou informaci. Náš algoritmus si byl schopen poradit i v situacích, kdy barva nebyla k dispozici, což bylo naším cílem. Kvantitativní vyjádření kvality algoritmu sledování není příliš možné ani vhodné především proto, že do celého procesu vstupuje příliš mnoho parametrů a není snadné vytvořit dostatečně obecný model, který by celou situaci kvalitně popsal (například jen velmi těžko vyzkoušíme dva algoritmy na jednom systému v terénu a změříme spravedlivě jejich kvalitu, neboť se neustále mění především světelné podmínky, obsluha systému reaguje různě, poloha manipulátoru, i další parametry nemusí být vždy nastaveny shodně – to vše znesnadňuje kvantitativní vyjádření kvality řešení). Z tohoto důvodu jsme neprováděli další rozsáhlejší měření výsledného řešení, místo toho jsme se spokojili s testováním v terénu a stanovením funkčnosti algoritmu spolu s vyjádřením slabých a silných míst algoritmu.



Obrázek 3.3: Vlevo je ukázka z testovaných videí A, B, a C. Vpravo jsou znázorněny průběhy hodnoty odchylky stanovení polohy objektu jednotlivými algoritmy pro jednotlivá videa.

3.5 Shrnutí

Algoritmus se chová dle našich požadavků ve většině situací, před něž byl postaven. Pracuje dobře i na monochromatických záběrech, při pohybu velmi dobře stabilizuje snímky z kamery a ke „ztrátě“ sledovaného objektu dochází v předem očekávaných situacích. Slabá místa, resp. situace, kdy algoritmus selhává, jsou předvídatelná už z principu fungování algoritmu.

Odolnost vůči zakrytí nebo zmizení objektu

Algoritmus velmi dobře reaguje na částečné zakrytí nebo úplné zmizení objektu. V praktických testech bylo jasně vidět, že situace, kdy část objektu zmizí, není žádným problémem a i situace, kdy byl objekt celý zakrytý, byly pro algoritmus řešitelné s určitými omezeními. Algoritmus v sobě integruje určitý druh extrapolace dráhy a předpokládá, že se objekt bude pohybovat stejným směrem, jako v nedávné minulosti. Na základě této informace pak může objekt dohledat (pokud zmizí na přechodnou dobu). Pokud není objekt nalezen po předem danou dobu, pak je prohlášen za ztracený a sledování končí.

Odolnost vůči pohybu kamery

Zabudovaná digitální stabilizace je schopna eliminovat translační pohyb kamery a v malém rozsahu i rotační pohyb. Pokud je pohyb kamery do rozsahu ca 30% rozměru obrazu, je ještě algoritmus schopen objekt nalézt (za předpokladu, že pohybem kamery nedojde k úplnému zmizení objektu ze zorného pole kamery). Velmi rychlé kmitání je filtrováno již principem snímání a relativně pomalý pohyb s frekvencí méně než 15 Hz je kompenzován algoritmem digitálně.

Schopnost práce s monochromatickým obrazem

Dle požadavku zadání byl algoritmus optimalizován tak, aby nezaostával za konkurenčními algoritmy v případě, kdy není k dispozici barevná informace. Ve chvíli, kdy se barva ztrácí v atmosféře nebo kdy nemáme k dispozici barevnou kameru, selhává velké množství algoritmů. Naše řešení je v tomto ohledu velmi robustní a sledování probíhá i bez barevné informace. Algoritmus je tedy možné použít i v obou případech (s barvou i bez ní).

Míjení dvou objektů

V případě, že se míjí dva objekty pohybující se stejným směrem podobného tvaru, dochází často k „přeskočení“ a algoritmus sleduje jiný cíl, než původně zamýšlený. Je to způsobeno především tím, že algoritmus nemá k dispozici dostatek vhodných informací na to, aby rozlišil, který z objektů má být sledován (příkladem může být situace, kdy jeden automobil předjíždí jiný – není-li k dispozici barva a v případě podobnosti vozů není možné, aby algoritmus vždy reagoval správně). Při míjení objektů pohybem proti sobě k tomuto problému nedochází, protože jednou z vlastností algoritmu je i integrace pohybového modelu, který penalizuje prudké změny směru i rychlosti. To ve svém důsledku může samozřejmě vést k chybě jiné – objekt měnící rychle směr a rychlost nebude možné sledovat právě díky těmto omezením. Zde je nutné se zamyslet nad tím, k jakému účelu bude algoritmus použit.

Nevýrazný objekt

Stává se, že v případě, kdy objekt není dostatečně výrazný (bez hran, rohů, výrazné textury nebo jiných jasově významných změn), není algoritmus schopen nalézt takové body, které by zaručily kontinuální sledování objektu. Může pak docházet k podobnému problému, jako u míjení objektů – algoritmus „přeskočí“ a sleduje jiný objekt, který ani nemusí být v pohybu – typicky se sledování zastaví.

Tvarová změna objektu

Objekt, který mění svůj tvar z důvodu přibližování se, rotace nebo jiného druhu pohybu, může měnit výrazně polohu bodů, které byly vybrány pro účely sledování. Jelikož jsme si pro účely sledování nevybrali algoritmus, který by přímo počítal s takovými změnami, mohou se tyto změny ukázat jako problematické, především ve chvíli, kdy nějakým pohybem dojde k velké změně jasu objektu. Algoritmus je schopen reagovat dobře na změnu měřítka, ale rotace objektu mu činí problémy.

Chybně označený objekt

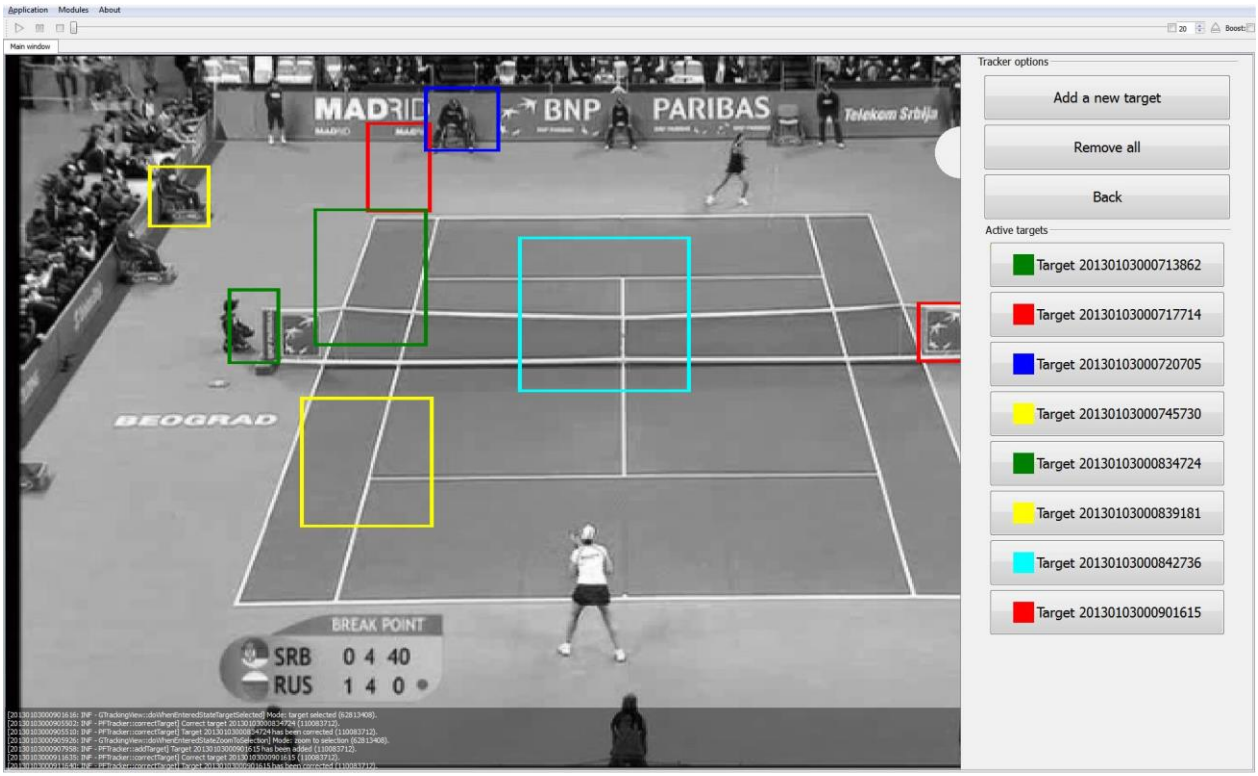
Mnohdy se stává, že uživatel označí objekt tak, že nejsou zachyceny výrazné body objektu a algoritmus pak již z některého z výše uvedených důvodů selže.

4 Závěr

Tato práce shrnuje teoretické poznatky, které mají vztah k problematice digitální stabilizace obrazu a sledování objektu především pro účely bezpečnostních aplikací. Kromě základních definic problému byly představeny metody pro výběr bodů vhodných pro sledování a metody sledování objektů. Vybrané metody byly implementovány a porovnány jejich vlastnosti především s přihlédnutím na požadavky aplikací v reálném světě. Porovnání metod bylo založeno především na subjektivním hodnocení kvality výstupu, rychlosti a potřeb pro zvolenou úlohu. Zvážení výhod a nevýhod nás vedlo k tomu, že mezi mnoha dostupnými řešeními jsme pro účely sledování objektu zvolili algoritmus založený na sledování částic, konkrétně pak varianta Bayesovský bootstrap filtr s tím, že dále byly provedeny optimalizace algoritmu tak, aby splňoval požadavky, které jsou stanoveny v úvodní kapitole.

Původní algoritmus byl do detailu rozebrán a implementován. Na základě znalostí z oblasti paralelizace výpočtů na grafických akcelerátorech, byly provedeny úpravy vedoucí ke značnému zrychlení algoritmu. Kromě toho jsme také navrhli mnohá vylepšení, která měla zamezit nežádoucímu chování algoritmu v kritických situacích. Silnými stránkami algoritmu je především odolnost vůči částečnému zakrytí (ale i úplnému zmizení) objektu, odolnost vůči pohybu kamery během sledování a schopnost práce s monochromatickým (ale i barevným) obrazem. Naproti tomu slabými stránkami jsou například problém ztráty sledovaného objektu v případě míjení dvou objektů, sledování nevýrazného objektu, v případě tvarové změny objektu nebo chybně označeného objektu.

Experimenty ukazují, že se algoritmus chová dle našich požadavků ve většině situací, před něž byl postaven. Pracuje dobře na monochromatických záběrech, při pohybu velmi dobře stabilizuje snímky z kamery a ke „ztrátě“ sledovaného objektu dochází v předem očekávaných situacích. Na práci lze navázat především v oblasti zlepšení chování algoritmu v situacích, které byly identifikovány jako problematické. Na závěr ještě následuje ukázka z aplikace pro řízení systému pro sledování s výběrem objektů pro sledování. Je zde naznačena i možnost sledování více objektů zároveň.



Obrázek 4.1: Screenshot z aplikace pro systém sledování objektů.

5 Literatura

[1] Olšák, P. Úvod do algebry, zejména lineární. Praha: Skriptum FEL ČVUT, 2007.

[2] Jan, J. Číslicová filtrace, analýza a restaurace signálů. 1. vydání. Brno: VUT v Brně, 1997. ISBN: 80-214-0816-2.

[3] Jalal, A.S. et al. The State-of-the-Art in Visual Object Tracking [online, cit. 2012]. Publikováno 09-2012. *Informatica*, svazek 36, číslo 3. ISSN: 1854-3871. Dostupné z <http://www.informatica.si/PDF/Informatica_2012_3.pdf>.

[4] Beneda, M. Homografie a epipolární geometrie [online, cit. 2012], publikováno 31-10-2010. In: *Trilobot*. Zlín: Univerzita Tomáše Bati ve Zlíně, číslo 2, 2010. ISSN: 1804-1795. Dostupné z <http://trilobit.fai.utb.cz/homografie-a-epipolarni-geometrie>.

[5] Dubrofsky, E. Homography Estimation. Diplomová práce. Vancouver: Univerzita Britské Kolumbie, 2009.

[6] Brooks, A. C. Real-Time Digital Image Stabilization, EE 420 Image Processing Computer Project Final Paper, EED Northwestern University, USA, March 2003, p. 10.

[7] Ko, S. J., Lee, S. H., Jeon, S. W. Fast Digital Image Stabilizer Based on Gray-Coded Bit-Plane Matching, IEEE, USA, 1999, pp. 90-91, ISBN 0-7803-5123-1.

[8] Vella, F., Castorina, A., Mancuso, M., Messina, G. Robust Digital Image Stabilization Algorithm Using Block Motion Vectors, IEEE, USA, 2002, pp. 234-235, ISBN 0-7803-7300-6.

[9] Chang, J. Y., Hu, W. F., Cheng. M. H., Chang, B. S. Digital image translational and rotational motion stabilization using optical flow technique. In: *IEEE Transactions on Consumer Electronics*, svazek 48, číslo 1, s. 108-115, únor 2002. ISSN 0098-3063. doi: 10.1109/TCE.2002.1010098.

[10] Kupka, K. Statistické řízení jakosti: interaktivní analýza a interpretace dat pro řízení jakosti a ekonomiku. Pardubice : TriloByte, 1997. ISBN: 80-238-1818-X.

[11] Bay, H., Tuytelaars, T. a Gool, L. SURF: Speeded Up Robust Features. In: *Computer Vision – ECCV 2006*. Berlín: LNCS Springer, s. 404-417, 2006. doi 10.1007/11744023_32. ISBN 978-3-540-33832-1.

[12] Brown, M. a Lowe, D. Invariant Features from Interest Point Groups. In: *British Machine Vision Conference 2002*. Cardiff, Wales. 09-2002.

[13] Shi, J., Tomasi, C. Good Features to Track. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, s. 593-600, 06-1994. ISBN: 0-8186-5825-8.

[14] DeGroot, M.H., Schervish, M.J. *Probability and Statistics*. Pearson, 4. edice, 1. 1. 2011, s. 912. ISBN: 978-0321709707.

[15] Stauffer, C. „Adaptive background mixture models for real-time tracking“. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Fort Colins, CO, USA, svazek 2, 1999. ISBN: 0-7695-0149-4.

[16] Piccardi, M. „Background subtraction techniques: a review“. In: *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*. Sydney, Austrálie, svazek 4, s. 3099-3104, 2004. ISBN: 0-7803-8566-7.

[17] Rubinstein, R.Y., Dirk, P. K., *Simulation and the Monte Carlo Method*, Wiley & Sons, New York, 2008, ISBN: 978-0-470-17794-5.

[18] Gordon, N., J., Salmond, D., J., Smith, A., F., M., *Novel approach to nonlinear/non-Gaussian Bayesian state estimation*, Radar and Signal Processing, IEE Proceedings F In Radar and Signal Processing, 1993

[19] Shi, J., Tomasi, C., *Good Features to Track*, In: Computer Vision and Pattern Recognition, 1994, ISBN:0-8186-5825-8

[20] Lucas, B. D., Kanade, T. „An iterative image registration technique with an application to stereo vision“. In: *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI '81)*, s. 674-679.

[21] Bouguet, J., Y., *Pyramidal Implementation of the Lucas Kanade Feature Tracker*, Intel Corporation, Microprocessor Research Labs, 2000.

[22] Craciunescu, T. et al. Application of optical flow method for imaging diagnostic in JET. In: *Journal of Nuclear Materials*. Elsevier, svazek 400, vydání 3, 2010, s. 205-212. ISSN: 0022-3115.

[23] Nummiaro, K., Koller-Meier, E., Gool, V., L., *An Adaptive Color-Based Particle Filter*, 2002

[24] Isard, M., Blake, A., *Condensation – Conditional Density Propagation for Visual Tracking*, International Journal of Computer Vision, 1997

[25] Isard, M., Blake, A., *Contour Tracking by Stochastic Propagation of Conditional Density*, European Conference on Computer Vision, 1996

[26] Heap, T., Hogg, D. *Wormholes in Shape Spaces: Tracking through Discontinuous Changes in Shape, International Conference on Computer Vision*, 1998

[27] Luo, X., Huang, Y. *Visual Tracking With Singular Value Particle Filter*, International Workshop on Machine Learning for Signal Processing, 2010

[28] Pérez, P., Hue, C., Vermaak, J., Gangnet, M., *Color-Based Probabilistic Tracking*, ECCV 2002

[29] Khalid, S., M., Ilyas, U., M., Sarfaraz, S., M., Ajaz, A., M. *Bhattacharyya Coefficient in Correlation of Gray-Scale Objects*, Journal Of Multimedia, 2006

[30] Nguyen, H., T., Worring, M., Boomgaard, R., *Occlusion Robust Adaptive Template Tracking*, 2001

[31] Matthews, I., Ishikawa, T., Baker, S. *The Template Update Problem*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2003

[32] Jurie, F., Dhome, M. *Real Time Robust Template Matching*, BMVC, 2002

[33] Mei, X., Zhou, K, S., Porikli, F., *Probabilistic Visual Tracking Via Robust Template Matching and Incremental Subspace Update*, 2008

[34] Barrera, P., Canas, J., M., Matellan, V. *Visual object tracking in 3D with color based particle filter*, 2005.

[35] Sobral, A.C. *BGSLibrary: A OpenCV C++ Background Subtraction Library* [online, cit. 2012]. Software, 2012, dostupné z <http://code.google.com/p/bgslibrary/>

[36] Y. Benezeth, Y., Jodoin, P.M., Emile, B., Laurent, H., Rosenberger, C. „Review and Evaluation of Commonly-Implemented Background Subtraction Algorithms“. In: *Proceedings of the 19th International Conference on Pattern Recognition (ICPR 2008)*, 8-11. 12. 2008, Tampa, Florida, USA. IEEE 2008, 2008, ISBN: 978-1-4244-2175-6.

[37] Collins, R.T., Zhou, X., Teh, S.K. „An Open Source Tracking Testbed and Evaluation Web Site“. In: *IEEE International Workshop on Performance Evaluation of Tracking and Surveillance (PETS 2005)*, leden, 2005.

[38] Collins, R.T. et al. *VIVID Tracking Evaluation Web Site* [online, cit. 2012]. Software Dostupné z <http://vision.cse.psu.edu/data/vividEval/software.html>

[39] Corradi, A., Bellavista, P., Giannelli, C.: *Mobility and Handover Prediction mechanism: a performance comparison exploiting several filters* [online, cit. 2012]. Publikováno 04-05-

2010. Dostupné z
<http://lia.deis.unibo.it/Research/SOMA/MobilityPrediction/filters.shtml>

[40] Fischler, M. A., Bolles, R. C. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. In: *Comm. of the ACM* 24 (6): s. 381–395, červen 1981. doi:10.1145/358669.358692.

[41] Digia: *oficiální stránky knihovny Qt* [online, cit. 2012]. Dostupné z <http://qt.digia.com>.

[42] *Welcome - OpenCV Wiki*. 2010-04-06 [cit. 2011-10-11]. Dostupné z <http://opencv.willowgarage.com/wiki>.

[43] Munshi, A.: *The OpenCL Specification* [online]. Publikováno 6. 1. 2011 [cit. 2012]. Technická specifikace, verze 1.1, revize 44, Khronos OpenCL Working Group. Dostupné z <http://www.khronos.org/registry/cl/>.

[44] nVidia Corporation: *CUDA C Programming Guide* [online]. Publikováno 10-2012 [cit. 2012], nVidia. Dostupné z <http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf>.

[45] Triolet, D.: *Nvidia Fermi: the GPU Computing revolution* [online]. Publikováno 09-10-2009 [cit. 2012]. Dostupné z <http://www.behardware.com/articles/772-6/nvidia-fermi-the-gpu-computing-revolution.html>.