VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

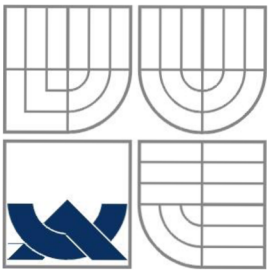
SLEDOVÁNÍ OBJEKTU VE VIDEU

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

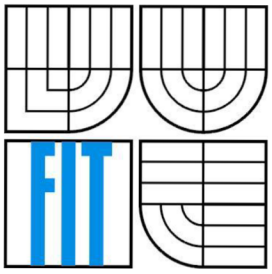
AUTOR PRÁCE
AUTHOR

Bc. ZDENĚK SOJMA

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SLEDOVÁNÍ OBJEKTU VE VIDEU

OBJECT TRACKING IN VIDEO

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. ZDENĚK SOJMA

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. ADAM HEROUT, Ph.D.

BRNO 2011

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2010/2011

Zadání diplomové práce

Řešitel: **Sojma Zdeněk, Bc.**
Obor: Počítačová grafika a multimédia
Téma: **Sledování objektu ve videu**
Object Tracking in Video
Kategorie: Počítačová grafika

Pokyny:

1. Seznamte se s problematikou sledování objektů ve videu.
2. Vyberte, prostudujte a popište nejdůležitější algoritmy sledování objektů v obraze.
3. Pořiďte video-data s pohyblivými objekty pro testování algoritmů sledování. Ujistěte se, že báze dat je dostatečně rozsáhlá a otestujte její vlastnosti s použitím vybraných algoritmů sledování objektů.
4. Implementujte vybraný algoritmus(y).
5. Vytvořte nástroj pro vyhodnocování vlastností sledovacího algoritmu nad pořízenou bází testovacích dat.
6. Vyhodnoťte vlastnosti implementovaného algoritmu a diskutujte dosažené výsledky. Porovnejte zjištěné skutečnosti s dostupnou literaturou.
7. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek pro prezentování projektu.

Literatura:

- dle pokynů vedoucího

Při obhajobě semestrální části diplomového projektu je požadováno:

- Body 1 až 3, dílčí řešení bodů 4 a 5.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Herout Adam, doc. Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 20. září 2010
Datum odevzdání: 25. května 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Diplomová práce popisuje principy činnosti nejvíce používaných druhů systémů pro sledování různých objektů ve videu a poté se především zaměřuje na popis a implementaci algoritmu pro interaktivní offline sledování obecných barevných objektů, jehož přednost spočívá ve vysoké přesnosti výpočtu trajektorie. Systém vytváří trajektorii z dat, jež jsou specifikována uživatelem při startu aplikace a poté interaktivně modifikována či přidávána za účelem zlepšení přesnosti výpočtu. Algoritmus je založen na detektoru, který pracuje na základě barevných příznaků, a na časové koherenci pohybu objektu, pomocí níž se vypočte více pravděpodobných trajektorií objektu. Výsledná optimální trajektorie je poté spočtena pomocí dynamického programování, jehož parametry jsou opět interaktivně upravovány uživatelem. Systém na videu o rozlišení 480×360 bodů pracuje rychlostí v rozmezí 15 až 70 snímků za vteřinu. Práce se dále zabývá vytvořením nástroje na optimální vyhodnocení úspěšnosti sledovacího algoritmu a diskutuje dosažené výsledky.

Abstract

This master's thesis describes principles of the most widely used object tracking systems in video and then mainly focuses on characterization and on implementation of an interactive offline tracking system for generic color objects. The algorithm quality consists in high accuracy evaluation of object trajectory. The system creates the output trajectory from input data specified by user which may be interactively modified and added to improve the system accuracy. The algorithm is based on a detector which uses a color bin features and on the temporal coherence of object motion to generate multiple candidate object trajectories. Optimal output trajectory is then calculated by dynamic programming whose parameters are also interactively modified by user. The system achieves 15-70 fps on a 480×360 video. The thesis describes implementation of an application which purpose is to optimally evaluate the tracker accuracy. The final results are also discussed.

Klíčová slova

Sledování objektu, video, barevný histogram, offline, interaktivní, integrální histogram, adaboost, mean shift, vyhodnocení sledování.

Keywords

Object tracking, video, color histogram, offline, interactive, integral histogram, adaboost, mean shift, tracking evaluation.

Citace

Sojma Zdeněk: Sledování objektu ve videu, diplomová práce, Brno, FIT VUT v Brně, 2011

Sledování objektu ve videu

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením doc. Ing. Adama Herouta, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Bc. Zdeněk Sojma
25.05.2011

Poděkování

Tímto bych chtěl poděkovat svému vedoucímu doc. Adamu Heroutovi, který mi svojí odbornou pomocí objasnil mnohé problémy spojené s tvorbou diplomové práce a různými radami přispěl k lepším výsledkům.

Obsah

1 Úvod

Sledování objektů ve videu postupem času získává stále větší pozornost v poli počítačového vidění, a to především díky zvyšující se dostupnosti a výkonnosti digitálních kamer, jejichž použití spolu s příslušnými algoritmy se v různých technických odvětvích stává finančně dostupným řešením, dříve téměř neřešitelných nebo velmi náročných problémů.

Sledování objektů může být velmi časově nákladný proces, a to hlavně kvůli množství dat, která videa obsahují. Rychlé vyextrahování užitečných a zahození nepotřebných informací je tedy jedním z klíčových problémů sledování, který je řešen širokou škálou různých nástrojů. Práce sledovacích algoritmů nejčastěji vypadá tak, že se z obrazu vyextrahují data, jež jsou specifická právě pro sledovaný objekt, a na jejichž základě se poté provádí detekce v dalších snímcích videa. Cílem sledovacích algoritmů je vytvořit jednu určitou trajektorii pro každý sledovaný objekt, k čemuž samotná detekce nestačí. Proto se současně s ní vytváří různé informativní struktury, které podléhají pravidlům, jež jsou specifická pro daný sledovací algoritmus, a podle kterých se výsledné trajektorie vypočítávají. Příkladem takových aplikací mohou být například dohledové a bezpečnostní systémy, navigační systémy či systémy pro rozpoznávání uživatelských gest.

Diplomová práce v kapitole 2 popisuje nejčastěji používané algoritmy pro sledování objektu ve videu spolu s principem jejich práce a příklady použití. V téže kapitole jsou také přehledně popsány různé způsoby reprezentace objektů a principy jejich detekce, které se nejčastěji ve sledovacích algoritmech využívají. V kapitole 3 je po teoretické stránce detailně charakterizován vybraný algoritmus. Jedná se o systém pro interaktivní offline sledování obecných barevných objektů ve videu a jeho výhoda spočívá v tom, že dokáže poměrně rychle vytvořit velmi přesnou trajektorii za pomocí interakce s uživatelem. Samotný systém není vůbec triviální a kombinuje práci mnoha dalších (nejen sledovacích) algoritmů, kde ty hlavní z nich jsou detailně rozebrány v kapitolách 4, 5 a 6. Kapitola 7 se zabývá návrhem a implementací vybraného systému spolu s implementací vyhodnocovacího algoritmu, jehož úkolem je spočíst dosaženou úspěšnost výsledků sledování. V kapitole 8 je popsán doporučený postup práce se systémem, který by měl uživateli zjednodušit a zefektivnit vytvoření finální trajektorie objektu. Vyhodnocením finálních výsledků se zabývá kapitola 9, která popisuje také výkon implementovaného systému a jeho robustnost. Kapitola 10 se věnuje možnému pokračování diplomové práce.

Text práce navazuje na semestrální projekt, ze kterého si přebírá a upravuje část teorie kapitol 2 a 3, popis algoritmů v kapitolách 4 a 5 a návrh systému popsaný v kapitole 7.

2 Sledování objektu ve videu

Sledování objektu ve videu může být definováno jako proces segmentace obrazu, kde se rozlišuje objekt zájmu od pozadí za účelem extrakce užitečných informací. Specifickým způsobem se sleduje pohyb objektu, jeho orientace, překrývání s dalšími objekty, interakce s těmito objekty a další různé charakteristické situace. V současné době se jedná o rychle se rozvíjející problematiku v oblasti počítačového zpracování obrazu, kde je sledování pohybujících se objektů většinou jedním z prvních kroků k extrakci potřebných informací z videa. Sledování objektu se používá v mnoha reálných aplikacích počítačového vidění, jako jsou například dohledové systémy, systémy pro sledování dopravy a lidí a podobně.

Sledování objektu není triviální problém především kvůli tomu, že se ve videu ztrácí informace reálného 3D světa převedením do 2D formy. Video obraz obsahuje také šum a změny v intenzitě osvětlení, které velkou mírou ovlivňují systémem vyextrahované příznaky, jež upozorňují na přítomnost objektu ve videu. Dalším problémem je to, že jsou sledovány objekty, které mají komplexní tvary a které se mnohdy pohybují po složitých trajektoriích, kde často mění svůj tvar a natočení vůči kameře. Na pozadí scén se také mnohdy vyskytují skupiny dalších objektů, jež mohou mít podobný vzhled, a s těmito objekty se sledovaný objekt může vzájemně vůči kameře (ať částečně nebo plně) překrývat. U mnoha sledovacích systémů je též vyžadován běh v reálném čase [32].

Cílem této kapitoly je rozčlenit nejčastěji používané metody do několika skupin, aby čtenář získal všeobecnou představu o problematice a případně si mohl mezi nimi najít metodu, která se nejlépe hodí k řešení jeho problému. Výběr metod většinou závisí na prostředí, ve kterém je příslušný systém aplikován, a na principu jeho použití.

2.1 Základní přístupy ke sledování objektů

Přístupy ke sledování objektů ve videu se dají rozdělit na tři základní metody [29], jejichž použití v aplikacích převážně závisí na podstatě řešeného problému:

1) **Metody založené na příznacích (*Feature-based methods*)** se zaměřují na extrakci charakteristických příznaků upozorňujících na přítomnost objektů v obrazových sekvencích, jako jsou například různé význačné body, hrany či histogramy. Extrakce příznaků je zpravidla aplikována na každý snímek videa v reálném čase. Rozhodnutí, s jakými typy příznaků pracovat, závisí na typu a budoucím využití vytvářené aplikace (viz dále). Tato metoda je nejvíce náchylná na šum, který v dnešní době ovlivňuje snad každé snímací zařízení. Díky němu bývají často extrahované příznaky nestabilní a sledování se řeší použitím kombinace více příznaků najednou. Příznaky ze sledování objektu využívají například práce [7][26][30].

2) **Diferenční metody (*Differential methods*)** jsou založené na prostorových a časových variacích scény (většinou s využitím jasu obrazu), neboli na výpočtu optického toku v sekvenci videa za pomocí viditelného pohybu mezi sousedními snímky. Naneštěstí tato metoda vyžaduje náročné výpočty s využitím derivování, které mohou být nepraktické ve scénách s velkým šumem [25]. Diferenční metoda je využita například ve článku [28].

3) **Metody využívající korelace (*Correlation*)** se využívají k zisku užitečných informací z videa tak, že počítají závislosti mezi snímky. Například ve článku [14] se porovnává referenční vzor s předpokládanou pozicí objektu ve všech snímcích videa. Korelační metody jsou neefektivní u videí s malým počtem snímků za vteřinu a při velkých změnách natočení objektu vůči kameře.

Sledovací systémy často využívají kombinace těchto tří metod k zvýšení robustnosti své práce, a tudíž se nespoléhají pouze na jednu z nich.

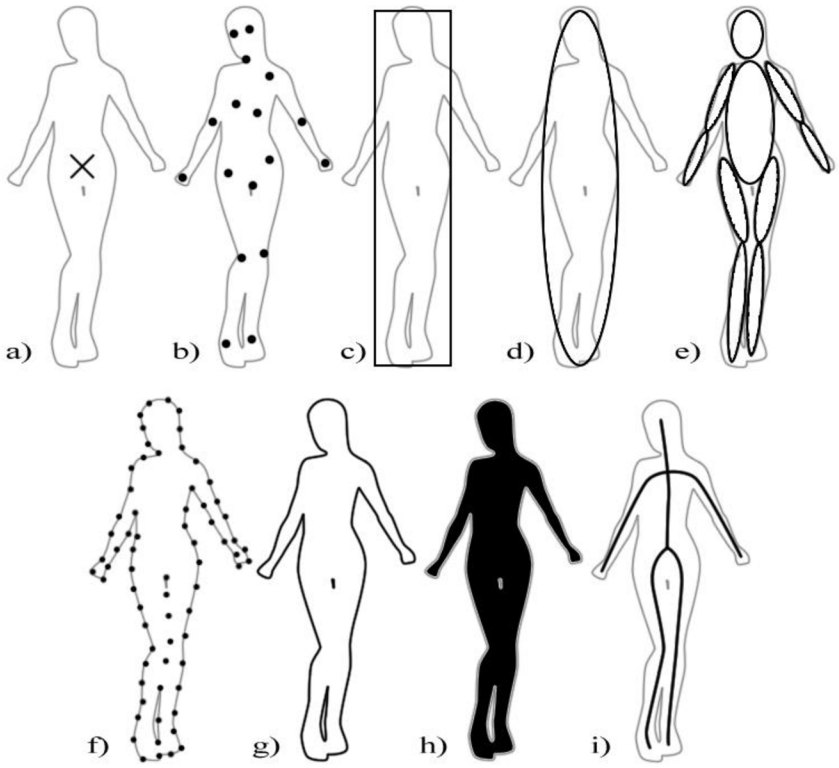
2.2 Reprezentace objektu

Sledovaný objekt může být ve videu reprezentován mnoha různými způsoby. Ty většinou závisí na typu sledovaného objektu. Sledovaným objektem bývají v mnoha případech lidé nebo vozidla, všeobecně ale lze sledovat jakýkoliv objekt zájmu, jehož výsledky jsou relevantní pro zpracování daného úkolu (například ryby v akváriu, bubliny ve vodě, letadla na obloze, atd.). Objekt lze reprezentovat pomocí jeho tvaru, jeho vzhledové reprezentace, či kombinace obou metod. Následující text je inspirován článkem [32].

2.2.1 Reprezentace tvarem

Tvarová reprezentace se provádí jak prostřednictvím nejjednodušších obrazců (bodů, obdélníků, elips, apod.), tak i použitím složitých komplexních tvarů, které přesně popisují vzhled a tvar celého objektu (kostra objektu, obrys objektu, atd.).

- **Bodová reprezentace**: Objekt je sledován pomocí jednoho bodu, který je znázorněn většinou v jeho středu (obrázek 1a). Tento styl reprezentace je vhodný spíše pro malé/vzdálené objekty, které zabírají neveliké části snímku. Na druhou stranu je možné sledovat větší objekty za pomocí série více bodů (obrázek 1b), kde se pro každý z nich individuálně aplikuje sledování (viz sekce 2.5.1). Bodové reprezentace využívá například článek [2].
- **Reprezentace geometrickými primitivy**: Pozice objektu je reprezentována jednoduchými geometrickými tvary, jako například obdélníkem (obrázek 1c) nebo elipsou (obrázek 1d). Pokud se nesledují objekty, které mají vyloženě daný geometrický tvar, využívá se mnohdy vah, které dávají menší význam hodnotám bodům na okraji geometrického primitiva než v jeho středu. Váhy řeší problém způsobený tím, že krajní body bývají hodně ovlivněny prvky na pozadí scény. Sledovaný objekt tudíž nemusí nutně mít pevný tvar odpovídající danému primitivu, a proto tyto reprezentace spolu s bodovými reprezentacemi patří, zvláště díky své jednoduchosti, k jedněm z nejvíce používaných.
- **Reprezentace kloubovou soustavou**: Objekt reprezentovaný kloubově je složen z více částí, které jsou k sobě navzájem připevněny „klouby“ a pohybují se společně pomocí jistých kinematických modelů (například dvě sousední části mají povolené rozmezí natočení jejich společného kloubu, a to nemůže být překročeno). K reprezentaci kloubového modelu v obraze lze použít například soustavu elips (obrázek 1e).
- **Reprezentace siluetou a obrysem**: Silueta (obrázek 1h) je definována jako prostor, jenž vyplňuje obrys (obrázek 1f,g) daného objektu. Tento styl se vyžívá k popisu složitých objektů pro systémy, které si zakládají na někdy i pixelové přesnosti sledování. K zisku obrysu/siluety se využívá například metoda odečítání pozadí [31] (viz dále).
- **Reprezentace kostrou**: Kostra (obrázek 1i) se dá například získat ze siluety provedením příslušené transformace vedoucí k zisku průměrné osy (*medial axis transform*), jako je tomu užito například ve článku [3]. Tato reprezentace je poté obvykle používána pro rozpoznávání objektu na základě tvaru. Reprezentace kostrou může být použita pro sledování objektů, které mají pevný i proměnlivý tvar.



Obrázek 1: Různé reprezentace objektu aplikované při sledování v obraze. Reprezentují se pomocí středového bodu (a), více bodů (b), obdélníku (c), elipsy (d), kloubové soustavy (e), bodů na obryse (f), obrysu (g), siluety (h) a kostry (i). Obrázek je inspirován obrázkem ve článku [32].

2.2.2 Reprezentace vzhledem

Vzhledová reprezentace je styl reprezentace objektu, jež může být získána na základě dat obsažených ve snímku a která úzce souvisí se samotným principem práce sledovacího algoritmu. Vzhledová reprezentace je uskutečněna na základě rozložení pravděpodobnosti (*probability density*) specifikující objekt například pomocí Gaussových funkcí, histogramů či Parzenova okna. Rozložení pravděpodobnosti je založené na příznacích (viz dále) získaných ze specifikované pozice objektu.

Pokud se nevyužívá samotné rozložení pravděpodobnosti, je možné reprezentovat objekt na základě vzorů/modelů (*templates*). Vzor je kombinací rozložení pravděpodobnosti či příznaků spolu s tvarovou reprezentací objektu (nejčastěji za pomocí geometrických primitiv, obrysu či siluety). Vzory samotné uchovávají pouze informaci z jednoho snímku videa, tudíž jsou náchylné na změny osvětlení či variaci natočení objektu vůči kameře. Proto se často využívá vzhledového modelu (*appearance model*), který je kombinací vzorů získaných za pomocí více snímků videa, jež obsahují vždy trochu jinou informaci o sledovaném objektu. Vzhledový model takto získá na robustnosti.

2.3 Výběr příznaků

Výběr správných příznaků hraje také velmi důležitou roli. V podstatě se jedná o výběr specificky měřitelných heuristických prvků vyextrahovaných z obrazu, které nejlépe rozlišují sledovaný objekt od pozadí, za účelem co nejspolehlivějšího a rychlého určení jeho pozice. Výběr příznaků úzce souvisí s výběrem reprezentace objektu. Pokud systém využívá jako příznaky například barvy objektu, můžeme předpokládat, že objekt bude reprezentován histogramem barev. Při použití hranového de-

tektoru bude objekt opět reprezentován například svým obrysem, apod. Mezi nejzákladnější příznaky patří barvy, hrany, textury objektu a optický tok ve videu [32]. Asi nejpoužívanějším z nich je barva. Mnoho sledovacích algoritmů však využívá kombinace více různých příznaků.

K *reprezentaci barvy* se nejčastěji využívá RGB, HSV, *L*u*v** a *L*a*b** prostor barev, kde každý z nich má své specifické vlastnosti, a proto nelze s jistotou určit, zda je jeden lepší než druhý. Všechny prostory barev jsou ovšem hodně náchylné na šum v obraze.

Detekce hran probíhá na základě silných změn intenzit snímku, tudíž není na šum zase tolik náchylná, ale při změně měřítka či tvaru objektu vůči kameře ztrácí na robustnosti.

Texturou je myšlen pravidelný přechod mezi různými hodnotami intenzit na povrchu objektu. K určení textury je potřeba vytvořit deskriptor, který daný povrch popisuje pomocí příslušných pravidel. Mezi nejznámější deskriptory patří například řízené pyramidy (*steerable pyramids*). Ty nejpoužívanější deskriptory jsou sumarizovány ve článku [32].

Optickým tokem je myšleno pole vektorů, které popisuje velikost a směr posunu každého bodu ve sledovaném regionu snímku do korespondujícího bodu v sousedním snímku. Pole se počítá podle jasových hodnot obrazů, a tak se jeho přesnost odvíjí od stálosti jasové složky videa.

2.4 Detekce objektů

Aby mohl jakýkoliv sledovací algoritmus určit polohu objektu, musí jej prvně umět detekovat. Existuje mnoho různých přístupů k detekci. Některé algoritmy se pokoušejí detekovat objekt v každém snímku videa, jiné v časových odstupech ob *n* snímků a dalším algoritmům dokonce stačí zjistit polohu objektu pouze tehdy, když se poprvé ve videu objeví. Detektory by se daly rozdělit na bodové, extraktory popředí a pozadí, segmentační algoritmy a učící se algoritmy.

Bodové detektory většinou pracují tak, že hledají zájmové body v obraze, které mají svoji určitou texturu v dané hledané oblasti snímku (okolí bodu). Body by měly být nejlépe invariantní vůči změně osvětlení, šumu a úhlu pohledu kamery. Běžně používanými bodovými detektory jsou například Moravcův operátor, Harrisův detektor, KLT detektor a SIFT detektor, které jsou podrobně popsány ve článku [17]. Harrisův a KLT detektor je nezávislý na rotaci a posunutí bodů, avšak při afinní a projekční transformaci u nich nastávají problémy. Proto pan Lowe představil SIFT algoritmus (*Scale Invariant Feature Transform*), který se s těmito transformacemi vypořádává. Další rozdíl mezi popsanými detektory je v tom, že generují jinak husté sady bodů. KLT odstraňuje zájmové body, které jsou příliš blízko u sebe, a na druhou stranu SIFT jich generuje daleko více, protože provádí své výpočty ve různých měřítcích a rozlišeních (viz obrázek 2).

Obrázek 2: Detekované zájmové body pomocí (zleva) Harrisova, KLT a SIFT detektoru. Obrázek je převzat ze článku [32].

Extraktory popředí a pozadí rozlišují snímek na dvě množiny bodů, kde popředí obsahuje detekovaný objekt zájmu a pozadí zbytek snímku. Extrakce popředí může být dosaženo například použitím metody rozdílových snímků (*difference images*) [12], metodou propastí a pohoří (*gap-mountain*

method) [27] nebo pomocí Kalmanova filtru [21]. Na určení pozadí se zase využívá algoritmus, který se vzhled scény učí (*background learning*) [29]. Učení pozadí se ovšem používá pouze u stacionárních kamer, kde se prvně nechá algoritmus chvíli běžet, aby zjistil tvar pozadí z neměnících se objektů ve scéně. Jakmile je proces učení dokončen, systém získá většinou kompletní (nebo skoro kompletní) informaci o pozadí. Výhodou tohoto přístupu je to, že jakmile známe pozadí, extrakce popředí je záležitostí jednoduchého odečítání dvou snímků.

Segmentační algoritmy rozdělují obraz do několika částí, které mají přibližně stejný vzhled (například na základě barvy, intenzity apod.). Největším problémem u těchto algoritmů je najít správné nastavení poměru mezi dobrou a efektivní segmentací. Na základě barvy a pozice bodu v prostoru snímku pracuje například mean shift algoritmus [8]. Ten na začátku běhu náhodně vybere velké množství bodů, které iterativně posune pomocí *mean shift vektoru* do lokálních středů příslušných klastrů. Body, které poté leží ve stejném středu, poté náleží jednomu segmentu obrazu. Na obrázku 3 je příklad segmentace, kde každý vyextrahovaný segment má svoji barvu. Mean shift algoritmus se nepoužívá nejen k detekci, ale i ke sledování objektu (viz dále). Dalším druhem algoritmu pro segmentování obrazu je například *Graph-Cuts* metoda neboli metoda aktivních obrysů (*active contours*), jejíž detailnější popis lze nalézt ve článku [32].

Obrázek 3: Segmentace obrazu vlevo pomocí mean shift algoritmu. Výsledek je vpravo, kde vypočtené segmenty jsou vykresleny stejnou barvou. Obrázek je převzat ze článku [8].

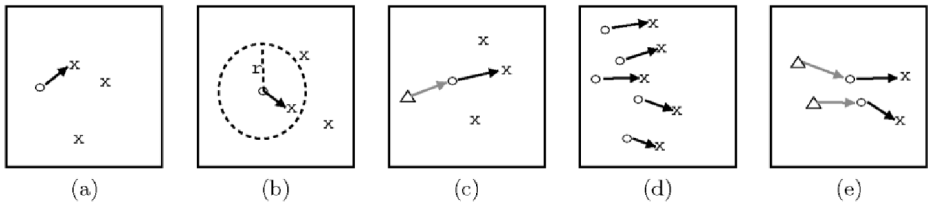
Učící se algoritmy pracují za pomoci učitele a provádí trénování detektoru z množiny vzorků objektu zadaných uživatelem. Za pomocí vstupních dat se vytvoří výstupní příznak, prostřednictvím kterého se detekuje objekt. Jako vstupní data mohou být použity jakékoliv příznaky popsané v sekci 2.3, a to i ve formě rozložení pravděpodobnosti. Vstupní data by měla být vybrána tak, aby zahrnovala všechny možné variace vzhledu objektu. Učící se algoritmy vyžadují velké množství vstupních dat, která musí být uživatelem rozdělena na objekt a pozadí. Efektivní princip označení dat je diskutován ve článku [32]. Pro trénování příznaku použitelného k detekci objektu se využívá například AdaBoost algoritmus (*Adaptive Boosting*) [26] a algoritmus podpůrných vektorů (*Support Vector Machine*, také SVM) [4][33]. AdaBoost algoritmus je iterativní metoda vytvářející přesný klasifikátor pomocí kombinace slabých klasifikátorů. Slabými klasifikátory může být například sada prahů vyextrahovaných ze vstupních dat, které s chybou menší než 50% rozdělují trénovací vzorky na objekt a pozadí. AdaBoost algoritmu je věnována celá kapitola 5. SVM algoritmus se na druhou stranu snaží vstupní data rozdělit jednou lineární hranicí. K nalezení takové hranice převádí data ze zadaného dvou dimenzionálního prostoru do n-dimenzionálního ($n > 2$), kde za pomoci přidaných dimenzí již lze od sebe data oddělit lineárně. Při detekci objektu se každému vzorku přidají příslušné dimenze a klasifikace je poté provedena na základě již vytvořené lineární hranice.

2.5 Sledování objektů

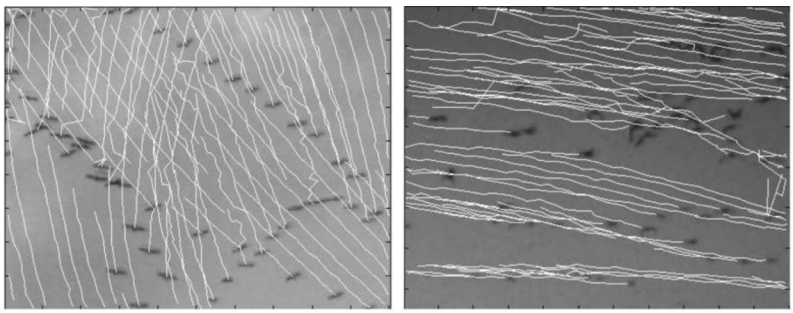
Cílem všech algoritmů zabývajících se sledováním objektu je vygenerovat jeho trajektorii skrz video. Fáze detekce objektu a sestavení jeho dráhy může být spojená v jednom kroku, nebo rozdělena do dvou. V prvním případě systém detekuje objekt v každém snímku videa a poté se různými sledovacími algoritmy snaží tyto detekce správně spojit. V druhém případě je trajektorie získána aktualizováním pozice objektu v aktuálním snímku, na základě pozic v minulých již vyhodnocených snímcích. Sledovací algoritmy se dají podle [32] rozdělit na sledování bodu (*point tracking*), sledování pomocí jádra (*kernel tracking*) a sledování siluety (*silhouette tracking*).

2.5.1 Sledování bodu

Algoritmy pro sledování bodu se hodí ke sledování objektu, který zabírá malou část snímku videa. Sledování více bodů se poté rovněž používá ke sledování větších objektů (viz obrázek 1b). Při sledování bodu je potřeba provést detekci objektu v každém snímku videa. Současně s ní se vždy provádí hledání detekovaného bodu v aktuálním snímku t, který koresponduje se sledovaným objektem ve snímku $t-1$ na základě jeho pozice a směrového vektoru. Při přiřazení nové pozice se používá měření ceny cesty mezi sledovaným objektem a všemi nově detekovanými body. Vybírá se následně ta cesta, která má cenu nejmenší. Této metodě se říká deterministická.



Obrázek 4: Konstanty omezující pohyb bodu při vytváření trajektorie. Blízkost (a), maximální rychlost (b), malá změna rychlosti (c), společný pohyb (d), tuhost (e). Obrázek je převzat ze článku [32].



Obrázek 5: Sledování hejna ptáků za pomocí deterministické metody. Obrázek je převzat ze článku [22].

Vybírání příslušné nové pozice objektu je limitováno určitými konstantami znázorněnými na obrázku 4. *Blízkostí* (*proximity*, obrázek 4a) je myšlena situace, kdy se pozice objektu nezmění znatelně od předchozí. *Maximální rychlost* (*maximum velocity*, obrázek 4b) omezuje pomocí poloměru r vzdálenost, kterou může objekt urazit. *Malá změna rychlosti* (*small velocity change*, obrázek 4c) zaručuje plynulost pohybu objektu tím, že se mezi snímky rychlost a směr pohybu příliš nemění. Společný *pohyb objektů* (*common motion*, obrázek 4d) zpracovává situaci, ve které se sleduje více objektů najednou, pomocí pravidla, kdy objekty zůstávají v převážně podobném sousedském rozestavení. *Tuhostí* (*rigidity*, obrázek 4e) je myšleno, že objekty reálného světa mají vždy stabilní/tuhý tvar, a tudíž

sousednost dvou sledovaných bodů, které náleží stejnému objektu, zůstává neměnná. Jelikož v prvním snímku ještě není znám směrový vektor pohybu, vypočítává se pomocí optického toku mezi prvními dvěma snímky. Popsaný přístup však nezahrnuje zpracování možnosti, že by objekt mohl vstoupit do zákrytu. Tato situace se většinou řeší přidáním virtuálních bodů, které představují chybějící objekt. S těmito body je poté zacházeno podobně jako se sledovanými. Sledování za pomocí deterministické metody se zabývá například článek [22].

Další možností při sledování bodů je použití statistické metody, která do sledování zanáší šum proto, aby simulovala klasický šum vzniklý snímáním kamery. Základem metody jsou dva modely, kde první popisuje vztah mezi předcházejícím stavem objektu s přidaným bílým šumem a novým stavem. Druhý model zpracovává závislost aktuálního stavu a šumu měření. Klasickým představitelem sledovacího algoritmu využívajícího statistické metody je Kalmanův filtr (*Kalman filter*) a jeho pokračovatel - částicový filtr (*particle filter*) [14]. Rozdíl mezi algoritmy spočívá v tom, že Kalmanův filtr je omezen na sledování pouze jednoho kandidáta na objekt v aktuálním snímku t. Částicový filtr již toto omezení nemá. Práce algoritmu spočívá v tom, že za pomoci šumu vygeneruje soubor částic, jejichž pozice určuje pravděpodobnou pozici objektu v následujícím snímku. Poté prostřednictvím vah, které určují podobnost částice se sledovaným objektem, vyřadí ty nejhorší a zbylé sleduje v dalším snímku. Statistické metody se hodí ke sledování objektu v zašuměných videích, ale jsou náchylné na rychlý pohyb kamery.

2.5.2 Sledování pomocí jádra

Sledování pomocí jádra objektu je většinou prováděno na základě reprezentace objektu prostřednictvím geometrických primitiv. Sledování se uskutečňuje na základě posunutí, rotací či transformace daného primitiva plynule snímek po snímku. Metody pracující s jádrem se podle [32] dají rozdělit na dvě skupiny: sledování pomocí vzorů spolu s rozložením pravděpodobnosti a sledování na základě vzhledového modelu.

Vzory jsou používány hlavně díky jejich relativní jednoduchosti a malé výpočetní náročnosti. Používanou metodou je porovnávání vzorů (*template matching*), která prohledává celý snímek na základě velikosti jádra a vzoru definovaném v předchozím snímku. Hledá se nová pozice vzoru, který má největší podobnost s minulým vzorem na základě jejich vzájemné korelace. Jako příznaku se využívá například intenzity nebo gradientu snímku. Druhou možností sledování, pracující na základě vzorů a rozložení pravděpodobnosti, je mean shift algoritmus [7][9] (viz obrázek 6). Ten oproti předchozí metodě nevyhledává novou pozici v celém snímku, ale pouze v okolí poslední známé pozice objektu. V aplikaci [9] je objekt reprezentován elipsou a histogram barev. Jeho práce spočívá v tom, že se prostřednictvím *mean shift vektoru* (podobně jako při segmentaci obrazu) snaží iterativně najít co největší podobnost histogramu objektu s histogramem na nové pozici na základě Bhattacharyho vzdálenosti. Sledování objektu za pomocí mean shift algoritmu je detailněji popsáno v kapitole 6.

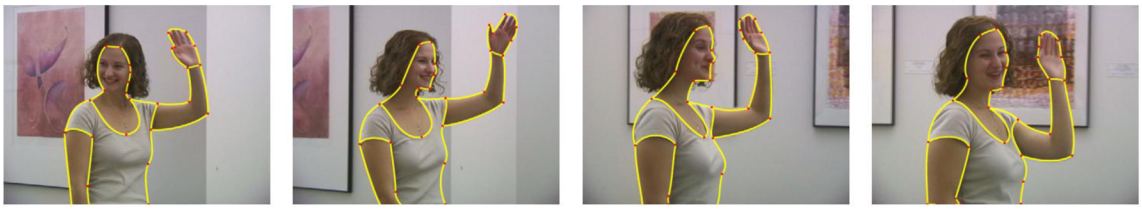
Obrázek 6: Příklad sledování pomocí mean shift algoritmu. Obrázek je převzat ze článku [9].

Sledování na základě vzhledového modelu je v podstatě stejné jako na základě vzoru, jen s tím rozdílem, že vzhledový model není získáván z posledního sledovaného snímku jako vzor, ale z více vzorů získaných z odlehlých snímků offline videa. Takto algoritmus získá robustnější vzor k porovnávání. Typickými detektory využitými k takovému sledování jsou učící se algoritmy [30].

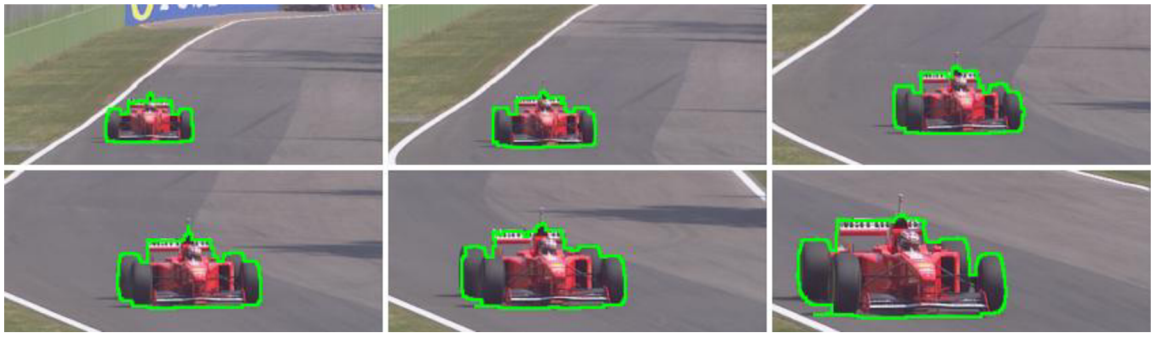
2.5.3 Sledování siluety

Sledování siluety se hodí pro objekty, které nelze popsat jednoduchými geometrickými primitivy, nebo pokud chceme sledovat objekt s velkou přesností. Principem práce algoritmů je najít sledovaný objekt na základě modelu vygenerovaného z předchozího snímku. Model je vygenerován podle tvaru dané siluety a může být například ve formě histogramu barev, hran či obrysů objektu. Sledování siluety lze rozdělit na algoritmy porovnávání tvarů (*shape matching*) a na algoritmy sledování obrysů (*contour tracking*) [32].

Algoritmy porovnávání tvarů (viz obrázek 7) pracují podobně jako metoda porovnávání vzorů popsaná v předchozí sekci. Hledání je provedeno jako výpočet podobnosti siluety objektu a jejího modelu s regiony v aktuálním snímku. Modelem je většinou hranová mapa, která je v každém snímku upravována v závislosti na novém tvaru siluety. K měření podobnosti siluet se využívá například Hausdorffova metrika, která porovnává dvě množiny bodů. Další možností je využití detekce siluety, na základě určení pozadí scény. Ve snímku jsou poté detekce porovnávány s částmi siluety podobně jako při sledování bodů výše, jen s tím rozdílem, že místo porovnávání bodů se porovnávají modely. Modely jsou většinou ve formě rozložení pravděpodobnosti (například s využitím histogramu barev) nebo v různých kombinacích hran, obrysů, apod. Sledováním siluety na základě porovnávání tvarů s využitím hranové mapy se zabývá například článek [1].



Obrázek 7: Příklad sledování siluety na základě porovnávání vzorů. Obrázek je převzat ze článku [1].



Obrázek 8: Příklad sledování obrysu. Obrázek je převzat ze článku [20].

Sledování obrysů (obrázek 8) se od porovnávání vzorů liší velmi razantně. Tato metoda iterativně vyvíjí obrys získaný v předchozím snímku tak, aby nejlépe pasoval na objekt v aktuálním. Podmínkou správné práce algoritmu je, aby se alespoň nějaká část objektu v minulém snímku překrývala s částí objektu v aktuálním. Stav objektu je definován tvarem parametry určující pohyb obrysu. Stav je aktualizován v každém novém snímku, pomocí aktuálně zjištěného tvaru a podobností obrysů.

2.6 Využití sledování objektů v praxi

V současné době se sledování objektů ve videu nejvíce používá v následujících praktických oblastech (čerpáno z [29][30][32]):

- **Automatické dohledové systémy**: Tyto systémy mají za úkol sledovat pohyby objektů ve specifických oblastech permanentně pozorovaných kamerovými systémy. Nejlépe by měly objekty včas identifikovat a případně nahlásit jakékoli jejich podezřelé jednání, aby se dalo zabránit případným následkům. Systém musí umět rozlišovat mezi přírodními jevy a jevy způsobenými lidským faktorem, což vyžaduje velmi spolehlivý a komplikovaný sledovací algoritmus. Dohledovým systémem se zabývají například články [16][28].
- **Robotické vidění**: Využívá se pro navigaci robota v okolním prostředí. Řídicí systém potřebuje umět identifikovat jakoukoli překážku, která přijde robotu do cesty, a přijít s řešením, jak se jí bez zbytečných problémů vyhnout. Pokud se jedná například o pohybující se překážky, je nutné do systému zahrnout sledování objektů v reálném čase.
- **Sledování dopravy**: V této oblasti lze algoritmus pro sledování objektů využít na dopravních komunikacích jako detekce porušení pravidel silničního provozu či jakéhokoliv jiného nelegálního jednání. Poté se často aplikují další algoritmy k identifikaci jakéhokoli vozidla, které tato pravidla poruší.
- **Indexace videa**: Jedná se o automatický popis obsahu videa, který je využíván například k vyhledávání v rozsáhlých multimediálních databázích nebo ke kategorizování videí apod.
- **Interakce člověka s počítačem**: Slouží například k ovládání počítače pomocí pohybu rukou, hlavy, očí či dokonce celého těla. Počítače tak v budoucnu získají schopnosti porozumět lidským gestům, což zmenší komunikační propast člověka s počítačem, kterou v současné době vyplňují pouze jednoduchá uživatelská rozhraní, která jsou ovládaná převážně klávesnicí a myší.

3 Implementovaný algoritmus

Pro svoji diplomovou práci jsem si vybral algoritmus od pánů Weie, Suna, Tanga a Shuma, který je detailněji popsán v jejich článku [30]. Jedná se o systém pro interaktivní offline sledování barevných objektů ve videu.

Offline systémy nacházejí nejlepší využití v případech, kdy je potřeba zajistit vysokou přesnost výpočtu trajektorie, a to i ve složitých scénách videa, kde dochází k rapidním změnám pozadí a kde objekt často mění svůj tvar i natočení vůči kameře. Vysoké přesnosti lze v současné době jen velmi těžko dosáhnout plně automatickými systémy, proto interakce s uživatelem je mnohdy nezbytná. Avšak většinou stačí jen malý zásah od uživatele, aby offline algoritmy zcela napravily špatnou detekci/špatné sledování objektu, nebo aby výrazně vylepšil přesnost svého výpočtu. Na druhou stranu se musí především brát ohled na to, aby interakce s uživatelem byly co nejméně frekventované. Offline systémy mají také proti online systémům zásadní výhodu v tom, že mají celé video předem k dispozici. To znamená, že si mohou řadu věcí předem spočítat či praktikovat výpočty ve více průchodech skrz video, a to jak po směru, tak i v protisměru přehrávání.

Online sledovací algoritmy se dají také docela jednoduše předělat na offline algoritmy principem *trial-and-error* [30]. Neboli tak, že se spustí sledování vybraného objektu, jehož výsledek je ihned zobrazen uživateli. Ten na vypočtenou trajektorii dohlíží, a pokud nastane chyba, výpočet zastaví, vrátí se o několik snímků zpět, kde nastaví správně pozici objektu a spustí sledování dále. Tento přístup ale přestává být použitelný, pokud sledovací algoritmus ztrácí trajektorii objektu příliš často, neboli vyžaduje mnoho externích zásahů. Také tímto principem nelze zajistit vždy plynulý a konzistentní přechod mezi sousedními snímky videa, ve kterých dochází k upravování trajektorie, což je ovšem mnohdy u sledovacích aplikací vyžadováno.

Další offline systémy zabývající se sledováním objektu ve videu jsou popsané například ve článcích [1][5]. Informace popsané dále v této kapitole jsou mnohdy čerpány ze článku [30], pokud tomu není uvedeno jinak.

3.1 Popis algoritmu

Algoritmus pracuje tak, že při jeho spuštění uživatel pomocí obdélníku (podobně jako na obrázku 1c) označí pozici objektu minimálně ve dvou snímcích videa. Tyto snímky jsou nazývané jako klíčové (nebo také k-snímky) a sledování se provede pouze v úseku videa mezi prvním a posledním z nich.

Výpočet probíhá následovně. Ze všech klíčových snímků se vyextrahují dvojice (pozice, histogram) $\mathrm{I} = \{\tilde{x}_k, y(\tilde{x}_k)\}_{k\in\kappa}$, kde $\tilde{x}_k$ je pozice objektu a $y(\tilde{x}_k)$ histogramem. Histogram $y(x) = \{y_1, \ldots, y_B\}$ s B sloupci je vypočten na základě barev objektu. Histogram barev je využit hlavně proto, že je invariantní vůči změně zorného úhlu, či natočení objektu vůči kameře. Výstupem algoritmu jsou vypočtené pozice objektu $\chi = \{x_i\}$ ve všech snímcích vybraného úseku videa (dále jen videa). Ty jsou vypočítané pomocí hlavní a časově nejnáročnější funkce aplikace

$$\mathrm{E}(\chi) = \sum_{i=1}^{N} d(x_i, \mathrm{I}) + \sum_{i=2}^{N} s(x_{i-1}, x_i), \tag{1}$$

kde $x_i = \tilde{x}_i$ pro $i \in \kappa$ a N značí počet snímků videa.

Výraz $d(\cdot)$ ve funkci (1) značí detekovanou pozici objektu v každém snímku a výraz $s(\cdot)$ upravuje pozici tak, aby zaručil plynulost pohybu objektu ve výstupní trajektorii. Pozice objektu je reprezentována pomocí obdélníku $x = \{c, w, h\}$, kde c je jeho středový bod, w je šířka a h je výška. Pro

výpočet výrazu $d(\cdot)$ se využívá histogramu objektu $y(x_i)$, kde pomocí Bhattacharyovy vzdálenosti [7] se spočte výraz

$$d(x_i,1) = BD(y(x_i),\tilde{y}) = 1 - \sum_{j=1}^{B} \sqrt{y_j \cdot \tilde{y}_j}\ , \tag{2}$$

v němž $\tilde{y}$ je histogramem nejbližšího klíčového snímku nebo histogramem lineárně interpolovaným ze dvou sousedních k-snímků. Pro výpočet výrazu $s(\cdot)$ se využívá klasická Euklidovská metrika, která je blíže popsaná v sekci 3.3. Pomocí tohoto výrazu se ve výsledné trajektorii mimo jiné zpracovává také vstoupení a vystoupení objektu ze zákrytu.

Interakce s uživatelem probíhá dvěma způsoby. Po skončení výpočtu funkce (1) se uživatel podívá na výslednou trajektorii a pomocí pár posuvníků zkusí nastavit hodnoty parametrů, které jsou zahrnuty ve výrazu $s(\cdot)$, a jejichž změna se na výsledku projeví takřka okamžitě. Pokud pomocí daných nastavení nelze dosáhnout potřebného výsledku, uživatel přidá nové, upraví či vymaže k-snímky a poté spustí výpočet trajektorie opět od začátku. Cyklus se opakuje tak dlouho, dokud výsledek neodpovídá uživatelovým představám. Jelikož druhý krok interakce vyžaduje nové spuštění celého sledovacího procesu, doba výpočtu (1) je tím pádem pro použitelnost algoritmu kritickým mezníkem a musí být co nejkratší. Výpočet trajektorie probíhá rychlostí 15 až 70 snímků za vteřinu pro video o rozlišení 480×320 pixelů (viz kapitola 9.1), takže s implementovanou aplikací lze pracovat interaktivním způsobem.

V systému je pro dosažení požadované rychlosti výpočtu použit detektor, který je založený na příznaku zvaném *boosted color bin*[1], blíže popsaném v kapitole 3.2.2, a který rychle odmítá majoritní množství regionů neobsahujících objekt. Detekce je provedena ob každých *n* snímků a aplikace poté k nalezení objektu mezi nimi využívá časové koherence jeho pohybu, čím vytváří vícero pravděpodobných trajektorií objektu. Výpočet trajektorií se provádí pomocí algoritmu zvaného *časově závislý růst trajektorie* (*temporal trajectory growing*), který je popsán v sekci 3.3. Na závěr se využívá algoritmu zvaného *dynamické programování* (*dynamic programming*) (viz kapitola 3.3.3), jenž na základě uživatelem interaktivně měněných parametrů kombinuje vypočtené pravděpodobné pozice objektu ve snímcích videa pro zisk nejoptimálnější výsledné trajektorie.

3.2 Detektor

Jak již bylo řečeno dříve, detektor objektu využívá příznak pojmenovaný *boosted color bin*, který byl poprvé představen ve článku [30], a na jehož základě se osvědčil jako velmi robustní pro sledování objektů. Jeho hlavní výhodou je rychlost výpočtu a také to, že je založený na sloupcích histogramů barev získaných z klíčových snímků, čímž získává invarianci vůči rotaci a změně měřítka objektu. Jelikož je výpočet 3D histogramu barev snímku velmi časově náročný[2], což ho činí pro interaktivní sledování takřka nepoužitelným, je pro extrakci histogramů z obrazu raději použita jednodušší a výpočetně rychlá metoda: sloupce vícera 1D histogramů barev získaných z RGB prostoru snímku.

---

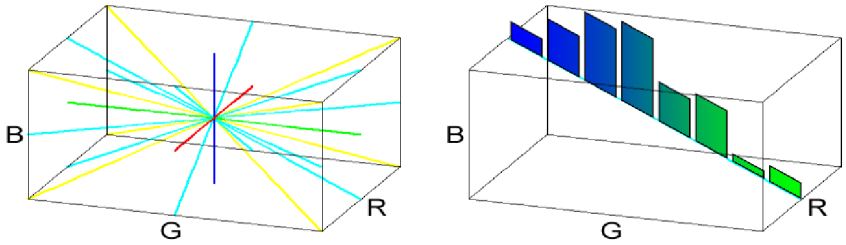
[1] Anglické sousloví *boosted color bin* by se do češtiny dalo přeložit například jako *boostovaný sloupec histogramu barev*, což nezní příliš pěkně. Tudíž jsem se rozhodl v textu dále uvádět pouze daný anglický ekvivalent. Pro sousloví *color histogram bin* budu dále používat český pojem *sloupec histogramu barev* nebo také *barevný sloupec*.

[2] Podle [30] vychází doba výpočtu 3D barevného histogramu pro objekt 80×80 pixelů ve snímku o rozlišení 320×240 na přibližně 20 sekund.

3.2.1 Extrakce histogramu barev

Histogram barev obdélníkového okna objektu, označeného v klíčovém snímku, se vypočítá promítnutím hodnot barev všech jeho pixelů na třináct jednodimenzionálních úseček, které rovnoměrně protínají RGB prostor a které procházejí bodem (127, 127, 127), jak je tomu ukázáno na obrázku 9 vlevo. Poté je na každé z těchto úseček pomocí získaných promítnutých hodnot vypočítán jednodimenzionální histogram (viz obrázek 9 vpravo). 1D histogram je pro robustnost rozdělen pouze na 8 sloupců, aby při budoucím porovnávání vzdáleností histogramů nehrály jemné odchylky v barvě (v osvětlení scény apod.) zásadní roli.

Vypočtené 1D histogramy se znormalizují, a když se všech třináct spojí v jeden, algoritmus získá k dispozici histogram barev $y(x)$ o celkových $13 \cdot 8 = 104$ sloupcích, které jsou v detektoru brané jako příznaky určující objekt (viz obrázek 12). Jelikož se pomocí příznaků detektor trénuje (viz dále), je potřeba vyextrahovat mnoho histogramů ze snímku. Samotné počítání histogramu barev zvlášť z každého nového obdélníkového regionu je zbytečně výpočetně náročné, a tudíž je i zdlouhavé. Proto je v systému využita struktura *integrálního histogramu* [19] (viz také kapitola 4), pomocí níž lze histogram rychle a v konstantním čase vyextrahovat z jakéhokoli obdélníkového regionu. V offline sledovacím systému lze integrální histogramy pro všechny potřebné snímky předem spočítat, a proto je budoucí extrakce histogramů extrémně rychlá.



Obrázek 9: Třináct úseček protínajících RGB prostor (vlevo), vypočtený jednodimenzionální histogram pomocí promítnutých hodnot na úsečce (vpravo). Ve skutečnosti platí, že |R|=|G|=|B|, ale velikost G je zvýšena, aby spolu vzájemně nesplývaly úsečky protínající prostor. Obrázek je inspirován obrázkem ve článku [30].

3.2.2 Boosted color bin

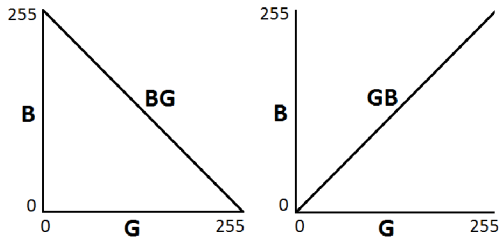
Jednotlivé sloupce histogramu barev jsou pro detekci objektu slabé. Jakmile se použije kombinace více těchto sloupců, lze vytvořit velmi silný a rozlišující detektor. Pro efektivní získání kombinace barevných sloupců je v detektoru použit AdaBoost algoritmus [10][15][26], který je blíže popsán v sekci 5.

Trénování detektoru probíhá tak, že jsou všechny klíčové snímky navzorkovány (principu vzorkování se věnuje další kapitola) a sloupce histogramů pozitivních i negativních vzorků jsou přivedeny na vstup AdaBoost algoritmu, který k nim přistupuje jako ke slabým klasifikátorům. Výsledný silný klasifikátor je poté použit k detekci objektu v dalších snímcích videa. Tento klasifikátor je nazýván *boosted color bin*.

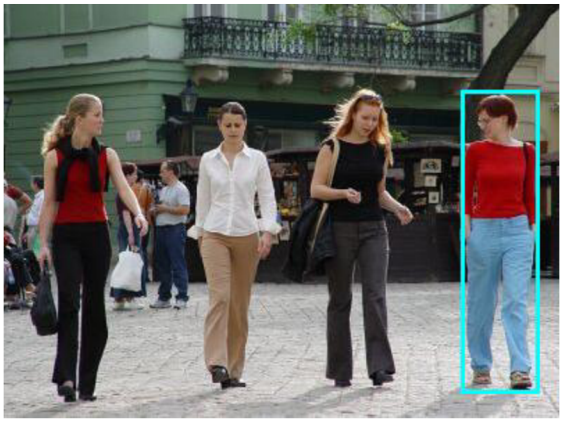
Algoritmus sekvenčně vybírá nejvíce rozlišující slabé klasifikátory a může dosáhnout až stoprocentní úspěšnosti, exponenciálně v závislosti na počtu vybraných klasifikátorů. Výsledný silný klasifikátor obvykle získává potřebnou přesnost již při výběru malého množství slabých klasifikátorů.

Obrázek 12 ukazuje všech 104 sloupců histogramu barev, získaného sloučením třinácti 1D normalizovaných histogramů regionu, jenž je označený na obrázku 11 azurovým obdélníkem. Ve vrchní polovině obrázku 12 je histogram objektu a ve spodní je histogram okolí. Zobrazené barvy sloupců značí barvu úsečky v bodě, do kterého se hodnoty barev pixelů promítly. Popisky značí název 1D

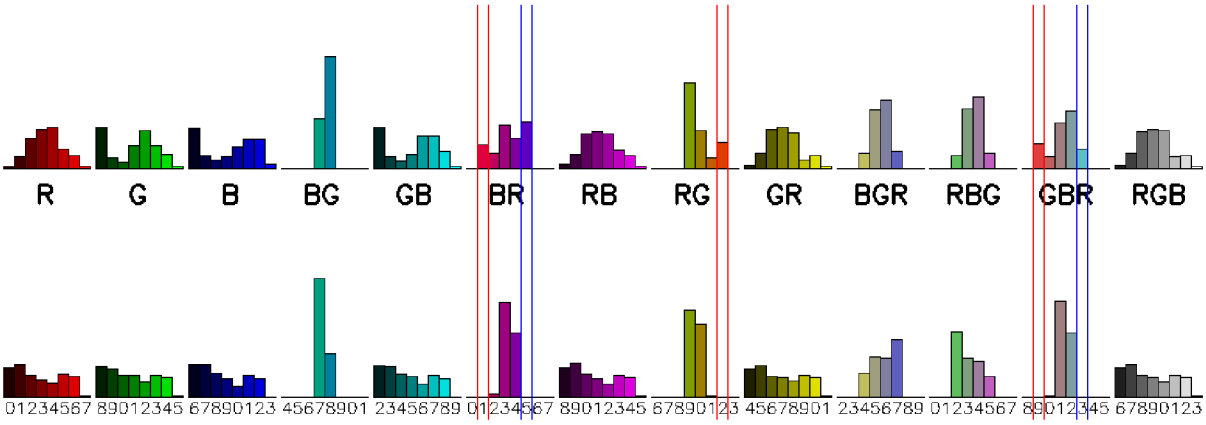
histogramu, neboli úsečky protínající RGB prostor, a to podle pravidla znázorněného následujícím obrázkem:



Obrázek 10: Vysvětlení popisků prostorových úseček protínajících RGB prostor.



Obrázek 11: Snímek použitý pro extrakci 104 sloupců histogramu barev v obrázku níže.



Obrázek 12: Histogramy barev získané z obrázku výše. Histogramy regionu objektu ohraničeného obdélníkem jsou nahoře, histogramy okolí jsou dole.

V histogramech na obrázku 12 je názorně vidět, jak červeně zvýrazněné sloupce dobře specifikují slečninu halenku a modře zvýrazněné její kalhoty. Tyto barevné sloupce jsou také vybrány pomocí AdaBoost algoritmu. Na správnou klasifikaci všech trénovacích vzorků získaných z tohoto snímku stačí zahrnout do trénovacího procesu pouze jeden ze zvýrazněných sloupců.

3.2.3 Trénování detektoru

Pro správné vytvoření silného klasifikátoru pomocí AdaBoost algoritmu je třeba použít několik tisíc pozitivních i negativních vzorků, které určují hodnoty slabých klasifikátorů. Jak již bylo řečeno, slabé klasifikátory jsou sloupce histogramů barev, jež jsou vypočítány z regionů/vzorků všech klíčových snímků.

Pozitivní vzorky se získají variací směru a velikosti posunutí pozice obdélníku objektu o pár pixelů tak, aby se lehce pozměnila jeho pozice i měřítko. Současně s každým takto získaným vzorkem se vytvoří ještě několik dalších, a to takovým způsobem, že se lehce pozmění hodnoty barev jeho pixelů (v rozsahu 0,8 až 1,2 násobku původní hodnoty). Úprava barev se provede především kvůli tomu, aby se předešlo případným změnám osvětlení scény a podobně. Negativní vzorky se získají z pozadí klíčových snímků pouhou variací pozice obdélníku, jenž má vždy stejnou velikost jako sledovaný objekt. V průměru se generuje okolo 1000 pozitivních a 2500-3000 negativních vzorků na jeden klíčový snímek.

Jelikož se vygeneruje poměrně malý počet trénovacích vzorků, musí se vzít v potaz jejich robustnost. Některé sloupce totiž jsou nestálé, a to díky změnám pohledu kamery, pozice objektu nebo díky částečnému zakrytí objektu jiným. Nestálost sloupce značí to, že je u něj velká pravděpodobnost, že bude dávat nesprávné výsledky v průběhu přehrávání videa. Na nestálý sloupec upozorňuje velmi malý práh u vybraného slabého klasifikátoru (menší než je 20% průměrné hodnoty sloupce). Takové sloupce nejsou do trénovacího běhu vůbec zařazeny.

Patřičný počet vybraných slabých klasifikátorů, které nejlépe odlišují objekt od pozadí, záleží na jejich rozlišitelnosti a pohybuje se v rozsahu od dvou do dvanácti na všech dosud testovaných videích. Aby nedošlo k přetrénování, je výpočet silného klasifikátoru zastaven, když dosáhne dostatečně vysoké úspěšnosti na trénovacích datech ($\geq 98\,\%$).

3.3 Časově závislý růst trajektorie

Detektor není dostatečně rychlý na to, aby mohl být aplikován na každý snímek videa hlavně proto, že má vysokou časovou náročnost výpočtu struktury integrálního histogramu. Dále také nelze zaručit, že najde sledovaný objekt ve všech snímcích videa, nebo že pozitivně neklasifikuje regiony, které objekt neobsahují.

Popisovaný systém tyto problémy řeší tak, že provede detekci objektu pouze ob každých n snímků (tyto snímky jsou nazývané i-snímky). Poté se z každého i-snímku pomocí časové koherence pohybu objektu spočítá jeho trajektorie po směru i v protisměru přehrávání videa, čímž se vytvoří množství kandidátních trajektorií (neboli trajektorie roste z každého i-snímku, jak napovídá název kapitoly). Nejoptimálnější výstupní trajektorie objektu je poté nalezena pomocí dynamického programování, které také ošetří případné překrytí objektu druhým, a jež dokáže opravit výslednou trajektorii v nepříliš přesných úsecích.

3.3.1 Extrakce počátečních vzorků

Každý i-snímek se hrubě navzorkuje s krokem o délce $1/4$ velikosti obdélníku sledovaného objektu. Všechny vzorky se klasifikují pomocí detektoru a u pozitivně klasifikovaných se spočte Bhattacharyho vzdálenost jejich histogramu barev oproti referenčnímu histogramu, jenž je získaný interpolací z histogramů barev klíčových snímků. Pokud je vzdálenost příliš velká $(> 0{,}4)$, je vzorek vyřazen.

Jelikož je snímek navzorkován jen velice hrubě, je třeba pozice zbylých vzorků doladit. Každý vzorek je posunut do svého lokálního maxima pomocí mean shift algoritmu [7], a to na základě refe-

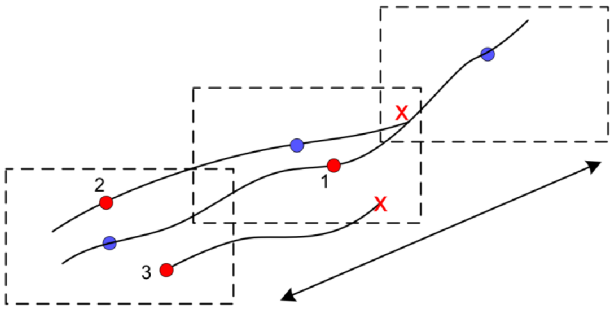
renčního histogramu získaného z k-snímků (mean shift je detailněji popsán v kapitole 6). Pokud některé upravené vzorky jsou příliš blízko u sebe, spojí se v jeden, čímž se vytvoří sada nesousedících vzorků. Sousedící vzorky by s největší pravděpodobností sdílely stejnou trajektorii, což je neefektivní. Nakonec, pokud je počet zbylých pozitivních vzorků větší než *K*, jsou ty, jež mají největší Bhattacharyho vzdálenost, vyřazeny.

3.3.2 Růst trajektorie

Každý i-snímek nyní obsahuje maximálně *K* pozitivně klasifikovaných vzorků. Růst trajektorie se provede paralelně pro každý nejlepší vzorek ve všech i-snímcích, a to jak ve směru, tak i v protisměru přehrávání videa pomocí mean shift sledovacího algoritmu. Nejlepší vzorek je ten, který má nejmenší Bhattacharyho vzdálenost svého histogramu oproti referenčnímu.

Jelikož výpočet modelu[3] je velmi časově náročný, je třeba, aby byl růst trajektorie vzorku zastaven co možná nejdříve. Prvním případem zastavení růstu je, když se počítaná trajektorie dostane příliš blízko k jiné, neboli se s ní sloučí (tím pádem by další pokračování růstu dávalo opět stejné výsledky). Druhý případ nastává, když se výpočet dostane do snímku, ve kterém je již vypočteno *K* lepších trajektorií jiných vzorků (tyto vypočtené vzorky pomocí růstu trajektorie budou dále nazývány *kandidáti na objekt* ve snímku).

Obrázek 13 ukazuje růst trajektorií tří vzorků, kde červenými tečkami jsou označené počáteční vzorky v i-snímcích a modře jsou kandidáti na objekt vypočítaní pomocí růstu trajektorie. Pro vzorek 1 je trajektorie spočtena dopředu i dozadu úspěšně. Trajektorie pro vzorek 2 je přerušena, když se dostane do blízkosti trajektorie vzorku/kandidáta 1 (červený křížek). Růst vzorku 3 byl přerušen ve snímku, kde je již *K* lepších kandidátů.



Obrázek 13: Princip růstu trajektorie, převzato z [30].

3.3.3 Vytvoření finální trajektorie

Algoritmus nazývaný dynamické programování (DP) představili ve svém článku o interaktivním sledovacím systému [5] pánové Buchanan a Fitzgibbon. Jedná se o algoritmus, který prohledává stavový prostor, v němž musí být vždy vybrán jeden stav z po sobě jdoucích množin stavů. Jinými slovy, algoritmus kombinuje kandidáty na objekt ve všech snímcích videa tak, aby našel jejich optimální uspořádání. Práce probíhá na základě několika uživatelem nastavených parametrů. Popisovaný přístup zahrnuje také zpracování překrytí objektu jiným a napravení trajektorie v nepříliš přesných úsecích.

[3] Modelem je nazýván histogram, pomocí kterého mean shift algoritmus sleduje daný objekt. Více v kapitole 6.2.1.

Algoritmus DP v mé aplikaci upravuje funkci (1) v časové složitosti $O(NK^2)$, kde N je počet snímků videa. Jelikož je K zvoleno jako malá konstanta, lze výpočet provést velmi rychle. V tomto algoritmu spočívá hlavní a nejvíce používaná interaktivní složka celého systému.

Po úspěšném růstu trajektorie, popsaném v předchozí kapitole, obsahuje nyní každý snímek maximálně K kandidátů na objekt. Algoritmus vypočítává výraz $s(x_{i-1}, x_i)$ ve funkci (1) pro snímek i pomocí pravidla

$$s(x_{i-1}, x_i) = \begin{cases} \lambda \cdot s'(x_{i-1}, x_i) & \textit{když objekt je stále viditelný} \\ \lambda_o & \textit{když objekt vstoupí/vystoupí ze zákrytu} \\ \lambda_r & \textit{když objekt je stále v zákrytu} \end{cases}, \qquad (3)$$

kde λ_o, λ_r jsou hodnoty penalizací interaktivně nastavitelné uživatelem, výraz $s'(x_{i-1}, x_i)$ značí klasickou Euklidovskou vzdálenost středů c dvou vybraných kandidátů ve snímcích i a $i-1$. λ je další parametr nastavitelný uživatelem, vytvořený hlavně pro to, aby se doladění trajektorie mohlo provádět pomocí dvou parametrů, kde první nastavuje vlastnosti dynamického programování po hrubých krocích a druhý po jemnějších krocích. Na druhou stranu se λ_r vždy určí jako $0{,}4 \cdot \lambda_o$, aby uživatel nemusel opět zbytečně nastavovat příliš mnoho parametrů. Tento poměr vznikl experimentováním autorů článku [5].

Optimální výstupní trajektorie je počítána postupně od druhého snímku videa k poslednímu. Každý kandidátní vzorek ve snímku si uchovává dvě hodnoty, cenu cesty od prvního vzorku v klíčovém snímku až k němu samotnému a index kandidátního vzorku v předchozím snímku, jenž je na téže cestě. Cena cesty ke vzorku se spočte tak, že se určí Euklidovská vzdálenost jeho středu ke středům všech kandidátních vzorků v předchozím snímku, neboli hodnota výrazu $s(x_{i-1}, x_i)$. Vypočtené vzdálenosti se sečtou s cenou cesty daného předchozího vzorku a nejmenší z nich se vybere/uloží jako cena cesty aktuálního vzorku spolu s indexem vybraného předchozího vzorku. Cena cesty vzorku v prvním klíčovém snímku je rovna nule.

Ve všech snímcích videa kromě klíčových snímků se uchovává ještě hodnota virtuálního vzorku, která představuje situaci, kdy by byl objekt aktuálně v zákrytu. Opět se projdou všechny vzorky v předchozím snímku, ale místo vzdálenosti $s(x_{i-1}, x_i)$ se k jejich cestě přičte hodnota penalizace λ_o (neboli se určí cena cesty, v níž by objekt aktuálně vstoupil do zákrytu). Další možností, kterou je třeba ošetřit, je ta, že by objekt zůstal v zákrytu. Neboli k hodnotě virtuálního vzorku v předchozím snímku se přičte penalizace λ_r. Do aktuálního virtuálního vzorku se stejně jako předtím uloží hodnota cesty s nejmenší cenou. Poslední dosud blíže nepopsanou možností je, že objekt vystoupí ze zákrytu. Pro zahrnutí této situace, se vždy počítaná cena cesty kandidátního vzorku (popsaná výše) porovná ještě také s cenou cesty u virtuálního vzorku v předchozím snímku, ke které se přičte hodnota penalizace λ_o.

Nejlepší výsledná trajektorie je poté ta, která má v posledním snímku nejmenší cenu cesty. Jelikož je v dané implementaci poslední snímek vždy klíčový, získáme jednoznačnou identifikaci nejlepší cesty.

číslo snímku:	1		2		3		4		5		6		7		...	
vzorek 1:	**0,0**	**nan**	0,2	1	0,5	3	0,6	2	1,1	1	**1,4**	**2**	**1,5**	**1**	...	...
vzorek 2:	nan	nan	0,2	1	0,4	4	0,7	4	**1,3**	**5**	1,4	1	nan	nan	...	...
vzorek 3:	nan	nan	**0,1**	**1**	0,5	3	nan	nan	0,6	1	1,0	3	nan	nan	...	...
vzorek 4:	nan	nan	0,2	1	0,4	1	nan	nan	nan	nan	1,3	1	nan	nan	...	...
virtuál. vzorek (5):	nan	nan	0,5	1	**0,6**	**3**	**0,8**	**5**	1,0	5	1,1	3	nan	nan	...	...

Tabulka 1: Princip práce algoritmu dynamického programování.

Tabulka 1 znázorňuje princip práce algoritmu. Hodnoty v tabulce jsou postupně vyplňovány po sloupcích zleva doprava, kde sloupce znázorňují snímky videa. Snímky 1 a 7 jsou klíčové, proto u nich není počítán virtuální vzorek. Každý sloupec je následně rozdělen na další dva podsloupce, kde první určuje celkovou cenu cesty a druhý určuje index předchozího vzorku v téže cestě. Hodnoty penalizací jsou nastaveny jako $\lambda_o = 0{,}5$ a $\lambda_r = 0{,}2$. Jak je vidět, nejlepší cestou a také výstupní trajektorií je sekvence vzorků {1, 3, 5, 5, 2, 1, 1}, kde objekt je ve snímcích 3 a 4 v zákrytu. Zisk hodnoty, o kterou byla navýšena cena cesty mezi vzorky, není v tabulce zachycen, protože by se stala poněkud nepřehlednou. Snímek 4 obsahuje pouze dva kandidátní vzorky a snímek 5 pouze tři.

Pozice objektu ve snímcích, v nichž je objekt v zákrytu (v tabulce snímky tři a čtyři), se dopočítá interpolací pozic a velikostí vzorků z okolních dvou snímků (snímků dva a pět).

Pokud se uživateli pomocí změn hodnot penalizací nepodaří finální trajektorii nastavit podle jeho představ, ale přitom vidí, že v potřebném úseku je objekt správně detekován (nechá si ukázat na videu všechny kandidátní vzorky), nemusí ještě definovat nový klíčový snímek a spouštět celý výpočet znovu. Může prvně zkusit, jak dopadne výsledek, když se algoritmu zadá, že má daný vzorek do výpočtu zahrnout jako *pevnou konstantu*. Se snímkem, v němž leží takto označený vzorek, je poté ve výpočtu nakládáno jako s klíčovým, tedy jako by obsahoval pouze tento jeden vzorek, a nepočítá se u něj hodnota virtuálního vzorku. Tudíž všechny cesty vždy povedou skrz pevnou konstantu.

4 Integrální histogram

Histogramy jsou jedním z nejpoužívanějších příznaků v mnoha aplikacích počítačového vidění, zejména kvůli jejich aplikovatelnosti jakýkoliv soubor dat. Asi nejvíce používanými typy histogramů jsou histogramy barev či intenzit v obraze. Pro samotnou detekci objektu jsou využity například ve článcích [18][26] a pro celkové sledování objektu ve článcích [7][30].

Integrální histogram [19] je metoda pro rychlý výpočet histogramů všech možných cílových regionů v obraze v konstantním čase. Pro celý snímek se vypočte potřebná struktura, pomocí metody zvané *propagace* (viz dále), která poté umožňuje s využitím pár aritmetických operací na sloupec vyextrahovat histogram potřebné podoblasti. V algoritmu se využívá prostorové pozice pixelů v kartézském souřadném systému obrazu a po jejich souřadnicích se propaguje součtová funkce. Integrální histogram tedy umožňuje do aplikace zahrnout například i kompletní vyhledávací proces skrz celý snímek v reálném čase, což u jiných běžných metod v dnešní době není takřka možné.

Integrální histogram lze bez problémů rozšířit také do více dimenzí, a to bez obětování žádné z jeho výpočetních výhod. V této kapitole je popsán princip propagace využívající *vlnoplochového snímání* (*wavefront scan*). Další styly propagace, jako je například *řetězcové snímání* (*string scan*), jsou popsány ve článku [19]. Následující text je zaměřen na popis vytvoření integrálního histogramu pro dvou dimenzionální obraz. Formální definici integrálního histogramu pro *d*-dimenzionální prostor je možno nalézt ve [19].

4.1 Propagace

Předpokládejme, že funkce *f* je definovaná ve dvou dimenzionálním kartézském prostoru N^2 jako $p \to f(p)$, kde $p = [x, y]$ je bod obrazu. Tato funkce se mapuje na *k*-dimenzionální tenzor, neboli $f([x, y]) = [z_1, \ldots, z_k]$. Řekneme-li, že dvou dimenzionální prostor obrazu je ohraničen v rozsahu N_1, N_2, pak integrální histogram $H(p_n, b)$ v daném bodě $p_n = [x_n, y_n]$ a sloupci *b* je definován jako

$$H(p_n, b) = \bigcup_{j=0}^{n} Q\left(f\left(p_j\right)\right), \tag{4}$$

kde $Q(\cdot)$ je korespondující sloupec histogramu současného bodu p_j a $\cup$ je operátor sjednocení definovaný následovně: hodnota sloupce *b* integrálního histogramu $H(p_n, b)$ je rovna součtu hodnot onoho sloupce všech předtím navštívených bodů, což je součet všech $Q\left(f\left(p_j\right)\right)$, dokud $j < n$. Jinými slovy $H(p_n, b)$ je histogram regionu obrazu mezi počátečním p_0 a současným bodem p_n ($0 \le x_i \le x_n$, $0 \le y_i \le y_n$). $H(p_N, b)$ je roven histogramu všech bodů v obraze, když $p_N = [N_1, N_2]$ je poslední bod obrazu. Díky tomu může být integrální histogram napsán rekurzivně jako

$$H\left(p_j, b\right) = H(p_{j-1}, b) \cup Q\left(f\left(p_j\right)\right) \tag{5}$$

s použitím počáteční podmínky $H\left(p_0, b\right) = 0$, neboli když jsou všechny sloupce histogramu na počátku propagace nulové.

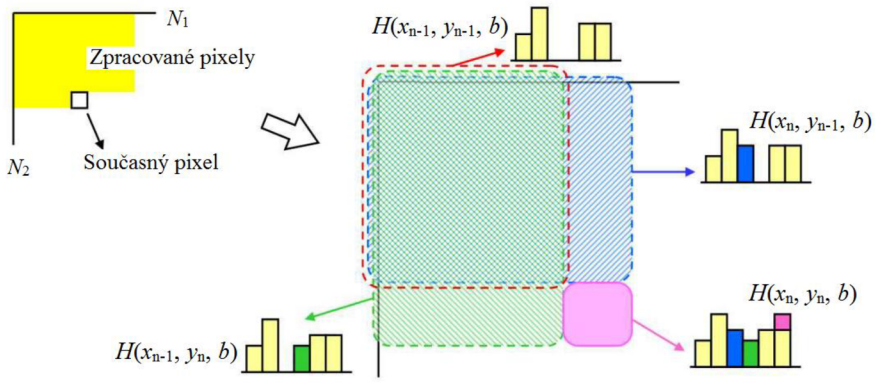
Existuje více přístupů k výpočtu fáze propagace v integrálním histogramu. Jak již bylo zmíněno výše, v implementovaném algoritmu pro sledování objektu jsem použil přístup využívající aktivních sad bodů, neboli *vlnoplochové snímání* [19] (viz obrázek 14). Tento přístup vyžaduje, aby u právě zpracovávaného bodu p_j byly již vypočítány body, které s ním těsně sousedí vlevo, nahoře

a vlevo nahoře. Integrální histogram daného bodu je poté vypočítán pomocí tří aritmetických operací na každém sloupci histogramu s využitím oněch sousedů.

Integrální histogram ve dvou dimenzionálním obraze o velikosti $N_1 \times N_2$ je vypočítán pomocí vlnoplochové propagace pro sloupec b a bod $p_j = [x_j, y_j]$ jako

$$H(x_j, y_j, b) = H(x_{j-1}, y_j, b) + H(x_j, y_{j-1}, b) - H(x_{j-1}, y_{j-1}, b) + Q(f(x_j, y_j)). \quad (6)$$

Následující obrázek znázorňuje princip propagace popsaný rovnicí (6). Integrální histogram v každém kroku propagace je vypočítán z histogramu jeho tří sousedů a sloupec odpovídající hodnotě daného bodu je navýšen o jedničku. Žlutě jsou označeny hodnoty pixelů a sloupců, které byly již vypočítány. Vykreslené histogramy odpovídají regionům, které jsou označené čárkovaným obdélníkem.



Obrázek 14: Propagace integrálního histogramu ve 2D obraze pomocí vlnoplochy, převzato a upraveno z [19].

4.2 Výpočet histogramu pro region obrazu

Pro výpočet sloupce b histogramu daného obdélníkového regionu T v obraze, jenž začíná bodem $p_i = [x_i, y_i]$ a končí bodem $p_j = [x_j, y_j]$, platí rovnice

$$h(T, b) = H(x_j, y_j, b) - H(x_j, y_{i-1}, b) - H(x_{i-1}, y_j, b) + H(x_{i-1}, y_{i-1}, b), \quad (7)$$

která lze použít pro jakýkoliv region v daném obraze bez nutnosti znovu složitého počítání propagace. Rovnice je vždy spočtena v konstantním čase pomocí tří aritmetických operací na sloupec histogramu.

5 AdaBoost algoritmus

AdaBoost je zkratka pro *Adaptive Boosting* a jedná se o samostatně se učící strojový algoritmus. Poprvé byl představen Yoavem Freundem a Robertem Schapirem v roce 1996 ve článku [10]. Ve svém implementovaném systému pro sledování objektu jsem ovšem použil modifikovanou verzi AdaBoostu, představenou Paulem Violou v roce 2000 [26], která je dále zběžně popsána.

Na detekci objektu lze nahlížet jako na problém klasifikace regionů/vzorků v obraze na objekt a pozadí. Pro detekci objektů je potřeba projít velkou řadu vzorků a ty rychle a efektivně klasifikovat. Jelikož takový klasifikátor je velmi obtížné získat, je potřeba využít kombinace více aspektů, které vystihují vlastnosti daného objektu (slabých klasifikátorů). AdaBoost je algoritmus, který kombinuje výsledky více slabých klasifikátorů tak, aby z nich vytvořil právě jeden silný.

Algoritmus přiřadí každému pozitivnímu a negativnímu trénovacímu vzorku určitou váhu, která je na začátku pro všechny stejná. Poté se iterativně na základě těchto vah vyhledá práh, jenž nejlépe rozděluje vzorky. Tento práh se přidá do silného klasifikátoru a provede se pomocí něj klasifikace všech vzorků. Těm vzorkům, které jsou klasifikovány dobře, se váha sníží (čímž se poté méně projeví na výběru dalšího prahu). Výsledný silný klasifikátor je kombinací iterativně získaných slabých klasifikátorů, kde každý má také jistou váhu α.

Slabý klasifikátor $h(x, f, p, \theta)$ se skládá z příznaku *f*, prahu θ a polarity p určující znaménko nerovnice v pravidle

$$h(x, f, p, \theta) = \begin{cases} 1 & pokud \;\; pf(x) < p\theta \\ 0 & jinak \end{cases}, \tag{8}$$

kde *x* označuje vzorky obrazu. Klasifikátor je nazýván slabým, protože se od něj neočekává, že by rozlišoval vzorky s vysokou pravděpodobností správnosti. Do trénovacího procesu lze však zahrnout pouze jakýkoliv slabý klasifikátor, jenž má chybu klasifikace menší než 50%. Při použití klasifikátoru s větší chybou by algoritmus začal divergovat, což je nežádoucí.

Výsledný silný klasifikátor má poté rovnici

$$h^{strong}(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & jinak \end{cases}, \tag{9}$$

kde $\{\alpha_t\}$ jsou lineární váhy slabých klasifikátorů.

Běh algoritmu je následující. Z klíčových snímků se získají trénovací data $(x_1, y_1), \ldots, (x_n, y_n)$, kde $y_i = 0$ pro negativní a $y_i = 1$ pro pozitivní vzorky dat. Inicializují se váhy jednotlivých vzorků *i* tak, že $w_{1,i} = 1/(2m)$ pro pozitivní a $w_{1,i} = 1/(2l)$ pro negativní vzorky, kde *m* je počet pozitivních a *l* je počet negativních vzorků dat. Poté se v cyklu $t = 1 \ldots T$ provádí následující kroky:

1) Normalizují se váhy $w_{t,i} \leftarrow w_{t,i} \Big/ \sum_{k=1}^{n} w_{t,k}$.
2) Pro každý vzorek *k* se vytvoří slabý klasifikátor h_k, u kterého se spočítá chyba ε_k s ohledem na váhy w_t. Efektivní způsob výběru klasifikátoru s nejmenší chybou je popsán níže v sekci 5.1.
3) Definuje se klasifikátor $h_t(x) = h_k(x, f_t, p_t, \theta_t)$, u kterého je chyba $\varepsilon_t = \varepsilon_k$ při daném příznaku f_t, prahu θ_t a polaritě p_t, nejmenší. Pokud je hodnota chyby ε_k větší než 50%, výpočet se zastaví.

4) Aktualizují se váhy vzorků $w_{t+1,i} = w_{t,i}\beta_t^{1-e_i}$, kde $e_i = 0$, pokud je vzorek x_i klasifikován správně, $e_i = 1$ jinak. A $\beta_t = \varepsilon_t / (1 - \varepsilon_t)$.

U silného klasifikátoru se α_t vypočítá následovně: $\alpha_t = \log(1/\beta_t)$.

5.1 Výběr slabého klasifikátoru

Při velkém množství vzorků je potřeba, aby hledání slabých klasifikátorů s nejmenší chybou bylo navrženo co nejefektivněji. Jednou z hlavních výhod AdaBoost algoritmu je, že proces učení probíhá opravdu rychle. Toho je dosaženo tím, že závislost aktuálně vybraného slabého klasifikátoru na předchozím vybraném klasifikátoru je efektivně předávána ve vahách vzorků. Tyto váhy mohou být využity k vyhodnocení chyby slabého klasifikátoru v konstantním čase [26].

Je potřeba, aby na začátku učení byly vzorky každého příznaku seřazeny na základě jejich hodnoty. Optimální práh každého příznaku lze pak určit v jednom průchodu nad seřazenými trénovacími vzorky dat. Pro každý vzorek v seznamu je poté potřeba uchovávat v paměti čtyři proměnné: celkový součet vah všech pozitivních vzorků T^+, celkový součet vah všech negativních vzorků T^-, součet vah pozitivních vzorků pod současně zpracovávaným vzorkem S^+ a součet vah negativních vzorků pod současně zpracovávaným vzorkem S^-. Poté se celá posloupnost projde a najde se pozice prahu s nejmenší chybou. Chyba příznaku při současném vzorku se vypočítá následovně:

$$\varepsilon = \min\left(S^+ + \left(T^- - S^-\right), S^- + \left(T^+ - S^+\right)\right). \tag{10}$$

Hodnota prahu θ se vypočítá pomocí nalezené pozice prahu jako průměr hodnot vzorku na aktuální pozici a vzorku pod ním v daném seřazeném seznamu. Polarita *p* se určí podle toho, která z dvojice ve funkci $\min(\cdot)$ se vybere (*pozitivní*, *negativní*).

6 Mean shift algoritmus

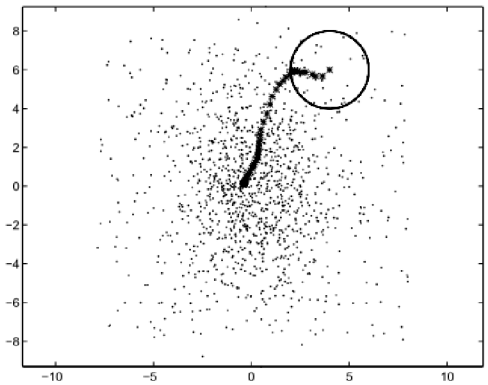
Mean shift je univerzální bezparametrický algoritmus, který se využívá k mnoha účelům, jako například k hledání modusu, ke shlukové analýze nějakého souboru dat a segmentaci obrazu (viz kapitola 2.4). Algoritmus byl poprvé představen pány Fukunagou a Hostetlerem ve článku [11] v roce 1975 a následně byl rozšířen tak, aby byl aplikovatelný i na další různá odvětví vědy, jako například do oboru počítačového vidění.

Následující kapitoly se zabývají prvně intuitivním popsáním algoritmu a poté detailnější charakteristikou celé problematiky zaměřenou na sledování objektu. Úvod do problematiky mean shift algoritmu pro obecné použití lze nalézt také ve článku [24], jeho rozsáhlejší využití pro hledání modusu a ke shlukové analýze je popsáno v [13] a články [7][9] se zabývají více odborným popisem algoritmu, kde jej modifikují tak, aby byl použitelný pro sledování objektu ve videu.

6.1 Idea algoritmu

Máme-li d-dimensionální prostor $\Re^d$ vstupních dat, pro které máme vypočtenou funkci rozložení pravděpodobnosti (*probability density function* neboli pdf), pak mean shift je algoritmus, jež pracuje nad danou pdf a který iterativně najde její lokální maximum. Algoritmus v každém kroku pomocí jádra K a centrálního bodu x spočítá gradient pdf, a následně vždy posune pozici bodu x po směru vektoru daného gradientu (mean shift vektoru). Iterace se zastaví, když algoritmus dosáhne požadované přesnosti (aktuální posun středového bodu x je menší než konstanta ε) a výsledná pozice bodu x poté určuje ono lokální maximum (viz obrázek 15).

Algoritmus tedy každý bod x asociuje s nejbližším vrcholem/lokálním maximem pdf, a pokud spustíme mean shift pro každý bod prostoru, můžeme například určit shluky bodů podle toho, do kterého lokálního maxima konvergují. Algoritmus tedy neprovádí výpočetně nákladné prohledávání celého snímku, ale snaží se iterativně najít nejlepší odpovídající pozici objektu v okolí posledně známé pozice. Mean shift algoritmus je možné dále použít třeba k segmentaci obrazu, detekci hran, apod., jak je tomu ve článku [6].



Obrázek 15: Postupný výpočet lokálního maxima pomocí mean shift algoritmu. Převzato ze [6].

6.2 Sledování objektu pomocí mean shift algoritmu

Tato kapitola podrobněji popisuje princip práce mean shift algoritmu, pomocí kterého lze sledovat pohyb obecného objektu skrz video. Detailnější popis sledovacího nástroje využívajícího mean shift lze nalézt ve článku [7] a text této sekce převážně vychází ze článků [7] a [9].

6.2.1 Reprezentace objektu

K charakterizaci objektu se využívá specifické pdf daného prostoru, ale aby náročnost výpočtu nebyla příliš vysoká, což je potřebné takřka u všech sledovacích systémů, lze místo pdf použít histogram, určující nejpravděpodobnější hodnoty daných sloupců. Toto sice není nejspolehlivější řešení, ale v aktuálním případě dostatečné a hlavně rychlé.

Ve své diplomové práci jsem pro reprezentaci objektu v mean shift algoritmu použil barevných histogramů, které se získají promítnutím na 13 jednodimenzionálních úseček RGB prostoru, stejně jak je tomu u zisku *boosted color bin* příznaku pro detektor v sekci 3.2.2. Každopádně celý následující postup mean shift algoritmu není omezen pouze na práci s barvami objektu, ale lze ho použít například i pro textury či hrany objektu, případně pro jejich různé kombinace.

Pravděpodobnostní histogram sledovaného objektu používaný k výpočtu mean shift algoritmu se nazývá *cílový model*

$$q = \{q_u\}_{u=1\ldots m}, \qquad \sum_1^m q_u = 1, \tag{11}$$

kde středová pozice modelu je bod 0 a m značí počet sloupců histogramu. Histogramy vypočtené algoritmem se nazývají *kandidátní modely*

$$p(y) = \{p_u(y)\}_{u=1\ldots m}, \qquad \sum_1^m p_u = 1, \tag{12}$$

kde y je středová pozice modelů.

Dále se definuje funkce $\rho(y) \equiv \rho[p(y), q]$, která určuje míru podobnosti modelů a její lokální maximum ve snímku videa určuje pozici kandidátního modelu $p(y)$, který je nejvíce podobný cílovému modelu q, definovanému v klíčovém snímku. Pro určení podobnosti se využívá Bhattacharyho koeficientu mezi p a q a jejich vzdálenost je spočtena jako (detailněji viz [9])

$$d(y) = \sqrt{1 - \rho[p(y), q]} = \sqrt{1 - \sum_{u=1}^m \sqrt{p_u(y) q_u}}\ . \tag{13}$$

Model je vypočítán z okolí jeho středového bodu a pomocí jádra K. Budeme předpokládat, že cílový model má střed v bodě 0 a kandidátní v bodě y, jak bylo řečeno výše. Množině bodů v okolí středu modelu $\{x_i\}_{i=1\ldots n}$ se přiřadí váhy pomocí profilu[4] jádra $k(x)$, což zvýší robustnost algoritmu tím, že středové body budou ovlivňovat vlastnosti modelu více než okrajové body, které bývají často nejvíce ovlivněny pozadím snímku (protože hodnota jádra se vzdáleností od středu klesá, viz obrázek 16 vlevo). S použitím funkce $b: \Re^2 \to \{1\ldots m\}$, která přiřazuje hodnotě bodu x_i index sloupce modelu $b(x_i)$, je pravděpodobnostní hodnota sloupce q_u určena pro cílový model vzorcem

[4] Profil jádra K je definován jako funkce $k: [0, \infty) \to \Re$ taková, že $K(x) = k(\|x\|^2)$. K výpočtu se využívá klasická euklidovská $L2$ norma.

$$q_u = C_q \sum_{i=1}^{n} k(\|x\|^2)\delta[b(x_i), u], \tag{14}$$

kde δ je *Kroneckerova delta* funkce (viz dále), C je normalizační konstanta s takovou hodnotou, aby $\sum_1^n q_u = 1$ a $u = 1 \ldots m$. Pravděpodobnostní hodnota sloupce kandidátního modelu se spočte jako

$$p_u(y) = C_p \sum_{i=1}^{n} k\left(\left\|\frac{y - x_i}{h}\right\|^2\right)\delta[b(x_i), u], \tag{15}$$

kde h je poloměr jádra K a C_p je opět normalizační konstanta s takovou hodnotou, aby $\sum_1^n p_u = 1$.

Kroneckerova delta funkce $\delta[i, j]$ vrací hodnotu 1, pokud $i = j$, a jinak vrací 0, což pro výpočet rovnic (14) a (15) znamená, že hodnota profilu je do sloupce histogramu připočtena pouze tehdy, když leží v intervalu jeho hodnot. Neboli hodnota bodu se vloží pouze do sloupce, do kterého náleží.

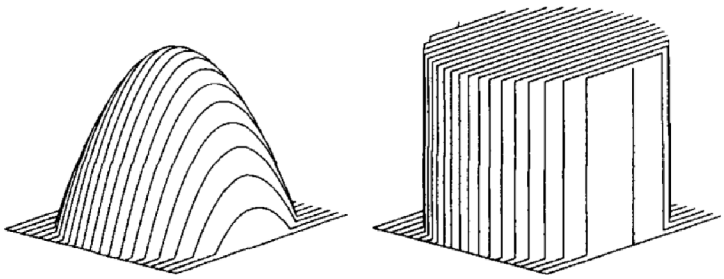
V diplomové práci jsem k výpočtu použil Epanechnikovo jádro (viz obrázek 16 vlevo)

$$K_E(x) = \begin{cases} 1/2 \cdot c_d^{-1}(d+2)(1-\|x\|^2) & pokud\ \|x\| < 1 \\ 0 & jinak \end{cases}, \tag{16}$$

kde c_d je objem d-dimenzionální sféry. Profil Epanechnikova jádra je popsán jako

$$k_E(x) = \begin{cases} 1/2 \cdot c_d^{-1}(d+2)(1-x) & pokud\ x < 1 \\ 0 & jinak \end{cases}. \tag{17}$$

Epanechnikův profil jsem použil, jak je doporučeno v [7], a to především z důvodu, že jeho derivace je rovna profilu uniformního jádra (obrázek 16 vpravo), což výrazně zjednoduší výpočet algoritmu (viz následující kapitola).



Obrázek 16: Epanechnikovo jádro (vlevo) a uniformní jádro (vpravo). Převzato z [13].

6.2.2 Princip práce algoritmu

Pro úspěšný běh mean shift algoritmu se předpokládá, že vždy v aktuálním (případně počátečním) snímku t je specifikována pozice objektu. Tím pádem cílový model q je již vypočítán a probíhá hledání kandidátních modelů v následujícím snímku. Hledání kandidátního modelu ve snímku $t+1$ začíná na pozici y_0, která je rovna středu výchozího modelu v předchozím snímku. Tudíž se musí začít výpočtem modelu $p(y_0) = \{p_u(y_0)\}_{u=1\ldots m}$. S využitím Taylorova rozvoje okolo hodnot $p_u(y_0)$ se lineární aproximace Bhattacharyho koeficientu podle [9] rovná

$$\rho[p(y),q] \approx \frac{1}{2}\sum_{u=1}^{m}\sqrt{p_u(y_0)q_u} + \frac{1}{2}\sum_{u=1}^{m} p_u(y)\sqrt{\frac{q_u}{p_u(y_0)}}. \tag{18}$$

Tato aproximace je dostačující pouze pokud se hodnoty povodního kandidátního modelu $\{p_u(y_0)\}_{u=1..m}$ příliš razantně neliší od následujícího $\{p_u(y)\}_{u=1...m}$, což se dá považovat za pravdivé u dvou po sobě jdoucích snímcích videa (neboli snímky t a $t+1$ obsahují přibližně stejný obraz). Algoritmus tedy neprovádí nákladné prohledávání celého snímku, ale snaží se iterativně najít nejlepší odpovídající pozici objektu v okolí jeho posledně známé pozice. Dosazením rovnice (15) do předchozí rovnice (18) dostaneme

$$\rho[p(y),q] \approx \frac{1}{2}\sum_{u=1}^{m}\sqrt{p_u(y_0)q_u} + \frac{C_p}{2}\sum_{i=1}^{n} w_i k\left(\left\|\frac{y-x_i}{h}\right\|^2\right), \tag{19}$$

kde

$$w_i = \sum_{u=1}^{m}\sqrt{\frac{q_u}{p_u(y_0)}}\delta[b(x_i),u]. \tag{20}$$

Nalezení odpovídajícího kandidátního modelu $p(y)$ odpovídá minimalizaci vzdálenosti (13), což je ekvivalentní s maximalizací hodnoty Bhattacharyho koeficientu $\rho(y)$. Tím pádem chceme získat v druhém výrazu v rovnici (19) maximální hodnotu (první výraz je nezávislý na *y*).

Druhý výraz reprezentuje odhad hustoty a je spočten pomocí profilu $k(x)$ v bodě *y* a zvážen pomocí vah w_i. Modus této hustoty je hledané lokální maximum a může být spočten pomocí *mean shift procedury* (výpočet procedury je detailně popsán ve článku [7]). Procedura rekurzivně posouvá střed jádra *K* z pozice y_0 k nové pozici y_1, a to pomocí rovnice

$$y_1 = \frac{\sum_{i=1}^{n} x_i w_i g\left(\left\|\frac{y_0 - x_i}{h}\right\|^2\right)}{\sum_{i=1}^{n} w_i g\left(\left\|\frac{y_0 - x_i}{h}\right\|^2\right)}, \tag{21}$$

kde $g(x) = -k'(x)$ za předpokladu, že derivace $k(x)$ existuje pro všechny body $x \in [0,\infty)$. Za povšimnutí stojí, že derivace použitého Epanechnikova profilu je uniformní profil (jak bylo zmíněno výše), tudíž je výpočet značně zjednodušen a urychlen.

Výsledný mean shift algoritmus pro výpočet nové pozice y_1 v jednom snímku videa, který na vstupu dostane pozici y_0, lze popsat následovně:

1) Výpočet cílového modelu $q = \{q_u\}_{u=1..m}$ v bodě y_0 a přechod na následující snímek.
2) Výpočet kandidátního modelu $p(y_0) = \{p_u(y_0)\}_{u=1...m}$ a vyhodnocení Bhattacharyho koeficientu $\rho[p(y_0),q] = \sum_{u=1}^{m}\sqrt{p_u(y_0)q_u}$.
3) Zisk vah $\{w_i\}_{i=1...n}$ pomocí rovnice (20).
4) Nalezení pozice kandidátního modelu y_1 pomocí mean shift procedury (21).
5) Výpočet $p(y) = \{p_u(y)\}_{u=1..m}$ a vyhodnocení $\rho[p(y_1),q] = \sum_{u=1}^{m}\sqrt{p_u(y_1)q_u}$.
6) Dokud platí $\rho[p(y_1),q] < \rho[p(y_0),q]$, provede se aktualizace pozice $y_1 = 1/2(y_0 + y_1)$ a vyhodnocení jejího koeficientu $\rho[p(y_1),q]$.

7) Pokud platí $\|y_1 - y_0\| < \varepsilon$, bylo dosaženo požadované přesnosti, a výpočet se pro aktuální snímek videa ukončí. Jinak se provede aktualizace pozice $y_0 \leftarrow y_1$ a pokračuje se krokem 2.

Pokud je velikost konstanty ε menší než jeden pixel, lze výpočet provézt se subpixelovou přesností. Algoritmus většinou konverguje po pár iteracích $(4-8)$, takže jeho výpočetní náročnost není nijak vysoká. Důkaz konvergence je možno nalézt ve článcích [7][24].

7 Návrh a implementace aplikace

Cílem mé práce bylo vybrat si libovolný algoritmus pro sledování objektu ve videu a implementovat jej. K tomuto účelu jsem si vybral systém, jenž je detailně popsán v kapitole 3. Dále jsem měl vytvořit nástroj na vyhodnocení výsledků sledování, který pomocí referenční trajektorie zadané uživatelem a vybrané metriky spočte úspěšnost sledování. Implementace tohoto nástroje je popsána v sekci 7.3.

V následující kapitole je popsán návrh programu spolu s jeho základní funkčnosti, jež jsou potřeba znát pro práci s ním. Úplné ovládání aplikace je popsáno v příloze A. V kapitole 7.2 je detailně popsána implementace mého systému, jehož návrh vychází z teorie popsané v sekci 3, ale mnohdy bylo potřebné pár parametrů upravit a přidat nějakou funkčnost navíc, aby výsledky byly více zajímavé a práce efektivnější.

7.1 Návrh a princip práce aplikace

Po delším zvažování jsem se rozhodl implementovat sledovací systém a algoritmus pro vyhodnocení výsledků sledování do jednoho společného programu, a to jak pro ušetření mé práce jako programátora, tak především pro lepší práci budoucího uživatele, protože přepnutí mezi oběma nástroji je ve výsledku velmi jednoduché, intuitivní a nemusí se zbytečně ukládat na disk výsledná trajektorie objektu a podobně.

Aplikace se spustí z příkazového řádku, kde jejím jediným povinným parametrem je cesta ke vstupnímu videu. Další parametry jsou volitelné a jsou jimi identifikátory `-kf` a `-ef`, které určují, zda se mají načíst klíčové snímky a referenční snímky pro vyhodnocovací nástroj ze souborů. Cesta k souborům se odvodí od cesty ke vstupnímu videu, neboli koncovka *avi*[5] se nahradí koncovkami *kframes* a *eframes*. Tento styl jmen vstupních souborů jsem zvolil hlavně kvůli jednodušší práci uživatele, aby nemusel v názvu uvádět mnohdy složité dlouhé cesty. Ze souboru s klíčovými snímky se načtou také parametry dynamického programování a informace o pozicích pevných konstant. Pozice pevných konstant se použijí, pouze pokud uživatel ponechá nastavení stejné jako při načtení, neboli nezmění žádné klíčové snímky. Při změně k-snímků by pozice pevných konstant nemusely odpovídat, a proto se načtení vynechá. Formát souborů je popsán v příloze B. Dalším volitelným parametrem aplikace je cesta k výstupnímu videu. Pokud není cesta zadaná, a uživatel bude chtít přesto video uložit, aplikace se pokusí výsledek nahrát do souboru, jehož cesta je stejná jako cesta ke vstupnímu videu, jen s tím rozdílem, že k jeho jménu je přidána koncovka `.output`.

Příkaz ke spuštění aplikace zapsaný formálně by mohl vypadat například následovně: `obj_track.exe [-kf] [-ef] <input_video> [output_video]` s tím, že jediná povinná posloupnost parametrů spočívá v tom, že cesta ke vstupnímu videu musí předcházet cestu k výstupnímu.

Uživateli se při spuštění aplikace zobrazí dvě okna, kde v jednom bude první snímek videa a v druhém poslední. Pokud je zadán parametr `-kf`, v oknech bude zobrazen první a poslední klíčový snímek. Uživatel se pomocí posuvníků u oken může libovolně ve videu posunout a zadat, odebrat či modifikovat k-snímky. Zadání pozice objektu klíčového snímku lze provést pomocí myši či klávesnice. Dále pokud uživatel chce, může si nechat zobrazit okno s promítnutými histogramy, které jsou

[5] Koncovka *avi* je u vstupních souborů nutností, protože aplikace je implementovaná pomocí knihovny OpenCV 2.1, která má s jinými multimediálními kontejnery problémy.

spočítané z klíčových snímků a cvičně spustit AdaBoost, aby viděl, kolik sloupců se vybírá a jakou má úspěšnost na trénovacích datech. Poté může bez problémů opět upravit obdélníky objektů.

Spuštění sledování je možné kdykoliv, jakmile je vybrán počáteční a koncový k-snímek. Systém poté vypočte funkci (1) a vrátí výsledek, který si uživatel prohlédne. Po úspěšné kalkulaci trajektorie se uživateli zobrazí další dvě okna se čtyřmi posuvníky, pomocí kterých upraví parametry dynamického programování (plus rozšíření popsaném v kapitole 7.4) tak, aby byl výsledek co nejpřesnější. Tento styl upravování trajektorie je velmi rychlý a hlavní část interaktivnosti systému spočívá právě v něm. Pokud uživatel případně najde chybu ve sledování, jež nemůže být napravena pomocí DP, přidá, ubere nebo modifikuje k-snímky, a spustí sledování znovu od začátku. Až je uživatel s výsledky spokojen, může vypočítané video uložit. Také lze uložit pozice klíčových snímků pro budoucí použití.

Po vypočtení finální trajektorie se uživatel může přepnout do vyhodnocovacího režimu, ve kterém se mu zobrazí vstupní video. Aplikace nyní čeká na zadání několika *referenčních snímků* (e-snímků), v nichž uživatel určí pomocí klávesnice a myši pozici sledovaného objektu a pomocí kterých vyhodnotí výslednou trajektorii. Pokud je zadán parametr `-ef`, e-snímky se načtou z příslušného souboru. Systém nijak nehlídá uživatele, kolik referenčních snímků zadal či s jakými rozestupy, každopádně je doporučeno držet jistý krok (například po deseti snímcích), aby byl výsledek co nejvěrohodnější. Po zadání všech e-snímků uživatel spustí vyhodnocení, jehož výsledek je mu zobrazen na standardní výstup. Opět pozice e-snímků lze kdykoliv uložit pro další použití.

7.2 Implementace sledovacího algoritmu

Aplikace je vytvořena pomocí programovacího jazyka C++, s využitím knihovny OpenCV 2.1 a vývojového prostředí Microsoft Visual Studia 2010. Program je napsán pro 32 bitový systém, především kvůli knihovně OpenCV, která momentálně nedisponuje 64 bitovou verzí. Paradoxně byl program vyvíjen a testován pouze pod operačním systémem Microsoft Windows 7 x64. Aplikace je napsána pomocí objektově orientovaného programování.

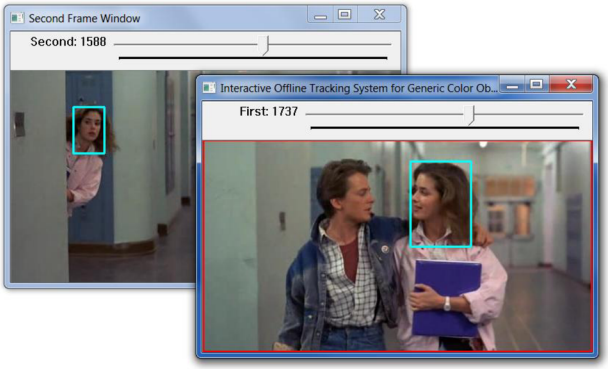
Článek [30], ze kterého implementace vychází, je napsán spíše obecnějším způsobem a mnohdy vynechává důležité aspekty některých částí implementace. Proto bylo třeba mnohá nastavení zjistit pomocí testování nebo z jiné dostupné literatury.

7.2.1 Grafické uživatelské rozhraní

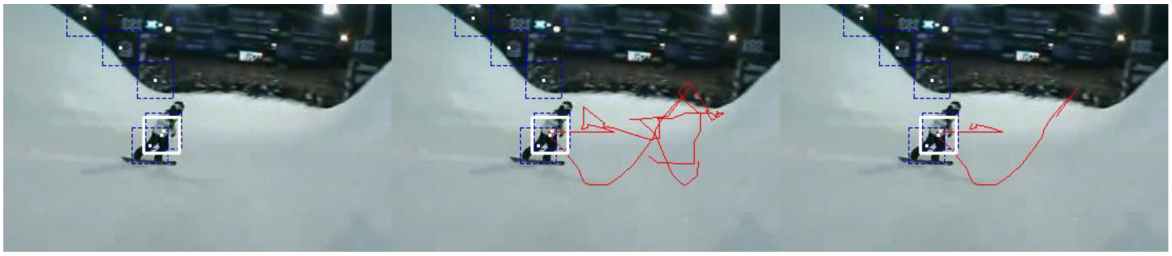
Pro tvorbu grafického uživatelského rozhraní jsem využil pouze modul *highgui* knihovny OpenCV 2.1. Modul byl pro tvorbu dostačující, i když vytvoření GUI s pomocí knihovny Qt, která má podporu pro OpenCV, by nebylo na škodu, protože by aplikace mohla obsahovat jednoduchá ovládací tlačítka a podobně, což by bylo určitě uživatelsky více přívětivé, než aktuální verze, kde se skoro vše ovládá klávesami.

GUI je ve zdrojovém kódu celé reprezentované jedním hlavním nadřazeným objektem, nazývaným `obj_track`, který je třídy `ObjTrack`. Aplikace ke svému ovládání využívá převážně klávesnici a myš (viz příloha A). Aktuálně vybraný region je ve snímku zobrazen červeným obdélníkem, po definování daného regionu jako regionu objektu se vytvoří klíčový snímek a obdélník se zbarví azurově.

Na obrázku 17 je zobrazen příklad spuštěné aplikace s načtenými klíčovými snímky. Jelikož má aplikace více oken, bylo třeba rozlišit, které je právě aktivní neboli které přijímá události od klávesnice. Aktivní okno je tedy označeno červeným obdélníkem okolo snímku. Mezi okny se lze přepínat intuitivně pomocí myši i klávesnice. Za zmínku stojí, že aplikace dále dovoluje uživateli posouvat se jemně po jednom snímku i po skocích (po deseti snímcích) a přeskočit celý úsek videa až na následující nebo předchozí klíčový snímek.



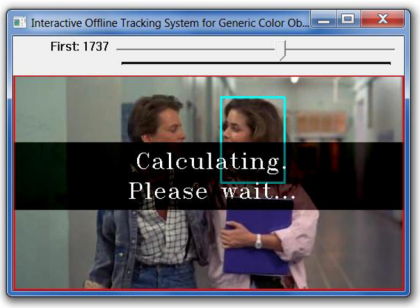
Obrázek 17: Příklad spuštěného programu s označenými klíčovými snímky.



Obrázek 18: Tři způsoby zobrazení spočtené výsledné trajektorie.

Výsledná trajektorie je uživateli zobrazena třemi způsoby (viz obrázek 18, zleva). Prvním je bílý obdélník, který zobrazuje výsledek v aktuálním snímku. Druhým je červená čára, která protíná středy výsledných obdélníků ve všech snímcích videa. Jelikož je ale mnohdy výsledná čára nepřehledná a zamotaná (hlavně u delších videí), lze její vykreslení vypnout, nebo pouze ji zobrazit v rozsahu padesáti snímků okolo aktuálního snímku, což je právě třetím způsobem zobrazení. Způsob zobrazení trajektorie pomocí protnutí středů je nezbytný ke správnému a jednoduchému upravení parametrů dynamického programování.

Měněním parametrů DP uživatel ihned vidí změnu ve tvaru červené čáry výsledné trajektorie. Kandidátní vzorky jsou na snímku zobrazeny pomocí modrých čárkovaných obdélníků a jejich středů. Uživatel může pravým tlačítkem myši označit daný kandidátní vzorek jako pevnou konstantu (kliknutím na jeho střed). Trajektorie se automaticky přepočítá a vzorek je zvýrazněn obdélníkem žluté barvy. Kliknutím na jiného kandidáta či zcela mimo všechny vzorky se snímek odznačí a je s ním zacházeno stejně jako předtím. Pokud algoritmus dopočítá pozici výsledné trajektorie, kdy byl objekt v zákrytu, je výsledek zobrazen pomocí *čárkovaného* bílého obdélníku (místo plného).



Obrázek 19: Příklad spuštěného programu, když se začne počítat trajektorie.

Pro lepší přehled o probíhané práci se uživateli, jakmile spustí výpočet, zobrazí na aktuálním snímku vyčkávací nápis (viz předchozí obrázek) a na standardní výstup se vypisuje postup výpočtu spolu s časem trvání jednotlivých fází a zajímavými údaji o počtu vybraných sloupců pro trénování *boosted color bin* příznaku apod. Po skončení výpočtu se také zobrazí celková doba jeho trvání a informace o počtu zpracovaných snímků za vteřinu.

7.2.2 Reprezentace snímků

Každý zpracovávaný snímek videa je v aplikaci reprezentován objektem třídy `Frame`. Třída obsahuje vlastnosti společné pro všechny snímky, jako je například jejich číslo, pozice kandidátů objektu, vzniklé pomocí růstu trajektorie, a metody na jejich správu. Objekt `obj_track` obsahuje *vektor ukazatelů* na objekty třídy `Frame`, který je vždy upraven podle potřeby (tento vektor bude dále v textu nazýván vektorem snímků). Na začátku vektor snímků obsahuje pouze klíčové snímky, před výpočtem trajektorie se ovšem rozšíří tak, aby zahrnul celý potřebný úsek videa.

Snímky jsou dále ještě rozděleny pomocí dědičnosti. Od třídy `Frame` dědí třída `SFrame` a třída `TrajectoryFrame`. `SFrame` (neboli *Standard Frame*) je třída vytvořená pro snímky, které nejsou ani klíčovými, ani i-snímky a provádí jejich inicializaci. `TrajectoryFrame` na druhou stranu obsahuje vlastnosti společné pro oba rozšířené typy snímků, jako je třeba objekt pro výpočet integrálního histogramu (třída `IntegralHistogram`) či trajektorie kandidáta pomocí mean shift algoritmu (třída `MeanShift`). Od třídy `TrajectoryFrame` dále dědí třída `IFrame` a `KFrame`. `IFrame` se stará o práci s i-snímky, jako je extrakce počátečních kandidátů, a `KFrame` o práci s k-snímky, jako je například jejich vzorkování. Ve výsledku je každý snímek definován pouze buďto jako `SFrame`, `IFrame` a nebo `KFrame`. `Frame` a `TrajectoryFrame` jsou *abstraktní třídy*.

7.2.3 Detekce objektu

Samotný objekt tvořící detektor v aplikaci není, ale o jeho práci se stará hlavní objekt `obj_track`, který mimo jiné obsahuje objekt třídy `AdaBoost`, pomocí něhož trénink detektoru probíhá. Při přidání každého nového klíčového snímku se automaticky spočte jeho integrální histogram (IH) (popis jeho implementace je níže v kapitole 7.2.6). V prvním kroku trénování detektoru se vytvoří struktura pro uložení pozitivních a negativních vzorků. U každého k-snímku se poté zavolá metoda pro jeho navzorkování, která vloží vzorky do dané struktury. Ukazatel na strukturu je dále předán `AdaBoost` objektu, jenž vypočte *boosted color bin* klasifikátor (detaily výpočtu klasifikátoru jsou popsány níže v sekci 7.2.7).

Klíčové snímky se vzorkují ve více průchodech. Příslušná funkce vyextrahuje jak pozitivní, tak negativní vzorky, kde nejdříve začne nanečisto s hrubým krokem vzorkování snímku, u kterého si počítá množství potenciálně vytvořených vzorků. Pokud je vzorků méně, než jsou nastavené limity, zjemní se krok vzorkování a cyklus se opakuje, dokud se nedosáhne potřebného počtu, nebo dokud již nelze krok dále zmenšovat. Poté se snímek spočteným krokem navzorkuje. Při extrakci negativních vzorků mnohdy nastává problém, že se okno, ze kterého se vzorek získá, nevleze mezi okno objektu a okraj snímku. Byla by škoda do trénovacího procesu tuto část snímku nezahrnout, protože by mohla obsahovat důležité informace. Tudíž jsem situaci vyřešil tak, že jsem okno na problematické ose zúžil a na druhé rozšířil, aby byl zachován jeho původní obsah. Zachování obsahu je potřebné pro vygenerování věrohodného vzorku.

Dalším bodem práce při detekci je vytvoření objektů pro všechny zbývající snímky videa a uložení ukazatelů na ně do vektoru snímků. I-snímky jsou alokovány ob každých 10 snímků videa, ostatní snímky jsou standardní. Pro každý vytvořený snímek se vypočítá jeho referenční velikost obdélníku objektu, jenž se interpoluje z velikosti obdélníků dvou sousedních klíčových snímků. Pomocí

této velikosti se dále vzorkují i-snímky, sleduje objekt a je podle ní také vykreslena finální trajektorie (tento krok by bylo dobré v budoucnu předělat, viz kapitola 10.2).

Sekvenčně pro každý i-snímek videa se spočte integrální histogram, prostřednictvím kterého se získají trénovací vzorky. Vzorkuje se s krokem o délce 1/4 velikosti referenčního obdélníku. Sekvenční zpracování je nutné, protože vytvořená struktura integrálního histogramu je velmi paměťově náročná (okolo 60MB na jeden snímek velikosti 480×320 pixelů), a tudíž musí být zabraná paměť po zisku vzorků uvolněna, aby zbyl dostatek virtuální paměti pro další práci. Což přináší nevýhodu, protože při změně klíčových snímků (a tudíž další nutnosti vypočtení funkce (1)) je třeba IH vypočíst znova (pro klíčové snímky zůstává IH alokován po celou dobu běhu aplikace). Další možností, se kterou jsem experimentoval, bylo uložení integrálního histogramu na pevný disk a následné jeho načtení. Na mém počítači bohužel samotné uložení struktury trvalo přibližně dvakrát déle než její výpočet. Pokud by ale uživatel měl rychlý disk, nebo by se ukládání provedlo paralelně s výpočtem dalších IH, daná možnost by určitě stála za zvážení.

Extrakce počátečních vzorků se provede pro každý i-snímek tak, že je mu objektem `obj_track` předán ukazatel na *boosted color bin* klasifikátor, jenž je vrácen objektem `AdaBoost` třídy. Každý i-snímek obsahuje vlastní objekt třídy `MeanShift`, pomocí něhož jsou pozitivně klasifikované vzorky posunuty do svého lokálního maxima na základě referenčního modelu, získaného buďto z nejbližšího k-snímku, nebo zprůměrováním modelů dvou sousedních k-snímků. Vypočtený kandidátní model je uložen pro další využití při růstu trajektorie. Následně jsou ještě sloučeny ty vzorky, jež jsou od sebe vzdáleny méně než dva pixely, jak je popsáno v teorii v kapitole 3.3.1. Detail práce mean shift algoritmu je popsán níže v sekci 7.2.8.

7.2.4 Růst trajektorie

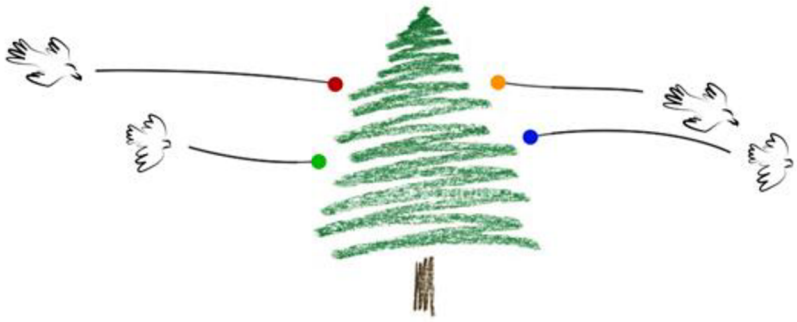
Růst trajektorie se provádí paralelně pro všechny klíčové snímky a i-snímky, jak již bylo řečeno v teoretické kapitole 3.3. Jelikož aplikace nevyužívá vícevláknového zpracování, růst všech trajektorií probíhá tak, že se opakovaně prochází celý vektor snímků a pro všechny klíčové a i-snímky se zavolají metody růstu `growForward()` a `growBackward()`. Metody provedou obousměrný růst trajektorie nejlepšího kandidátního vzorku o jeden následující snímek a určí, zda je daný směr růstu trajektorie dokončen nebo ne. Výpočet se zastaví, když jsou všechny růsty trajektorií dokončené.

Všechny snímky ve vektoru snímků obsahují vektor objektů třídy `TrajectorySample`, ve kterém jsou uloženy kandidáti na objekt, jež jsou vytvořeni růstem trajektorií (červené tečky na obrázku 13). Asi nejzákladnějšími vlastnostmi, které třída `TrajectorySample` obsahuje, je *model vzorku*, *vzdálenost modelu* od referenčního, *ukazatel na trajektorii*, v níž je vzorek obsažen, jeho *pozice v trajektorii* a *pozice obdélníku* vzorku ve snímku. Každý klíčový a i-snímek také obsahuje objekt třídy `Trajectory`. Třída `Trajectory` si udržuje vektor ukazatelů na objekty třídy `TrajectorySample`, které sama svým růstem přidala do snímků, aby bylo možno v dalším cyklu (skrz vektor snímků) v růstu pokračovat tam, kde se skončilo. Jinými slovy, vektor objektů třídy `TrajectorySample` je uložen ve snímcích a v trajektorii jsou pouze ukazatelé na ně. Uložení ve snímcích je důležité pro budoucí práci algoritmu dynamického programování. Důležitými vlastnostmi třídy `Trajectory` je rozsah trajektorie a informace o ukončenosti růstu.

V prvním kroku růstu je do trajektorie přidán nejlepší kandidátní vzorek klíčového snímku či i-snímku. Pomocí *referenčního modelu trajektorie* a mean shift algoritmu (v aplikaci třída s názvem `MeanShift`) se spočte nová pozice vzorku ve vedlejším snímku. Pokud vše projde pravidly popsanými v kapitole 3.3.2, je vzorek do snímku a trajektorie přidán. *Referenčním modelem trajektorie* je model snímku, ve kterém růst začal, nebo model počátečního vzorku v i-snímku, který trajektorie protnula a jenž není obsažen v žádné jiné trajektorii. Dále jsem do výpočtu přidal ještě podmínku, která kontroluje Bhattacharyho vzdálenost každého nového vzorku a ukončí růst trajektorie, pokud je vzdálenost větší než 0,4.

7.2.5 Vytvoření finální trajektorie

Implementace algoritmu dynamického programování se striktně drží teorie popsané v kapitole 3.3.3. Celý algoritmus pracuje rychlostí více než deset tisíc snímků za vteřinu, takže každá změna parametrů se uživateli zobrazuje takřka okamžitě.



Obrázek 20: Trajektorie dvou ptáků, kteří se v průběhu letu schovají za strom.

Na obrázku 20 jsou zobrazeny dvě trajektorie ptáků, kteří se v průběhu letu schovají za strom, neboli se dostanou do zákrytu. Oba ptáci mají přibližně stejnou velikost i barevné rozložení, je tedy velmi pravděpodobné, že detektor detekuje oba jako sledovaný objekt, i když bychom chtěli sledovat pouze jednoho z nich. Pokud bychom sledovali například dravce (výše), pomocí DP by se při jeho vstoupení do zákrytu (červené kolečko) do ceny trajektorie přičítaly penalizace, dokud by opět ze zákrytu nevystoupil. Při vystoupení nastává problém, kdy hodnota penalizace λ_o je vždy stejná, a tudíž je stejný i přírůstek ceny ke všem aktuálním kandidátním vzorkům (oranžové a modré kolečko). Jelikož ale označujeme pozici objektu i na konci video sekvence, a víme, že algoritmus vybírá vždy tu cestu s nejmenší cenou, je trajektorie vybrána správně.

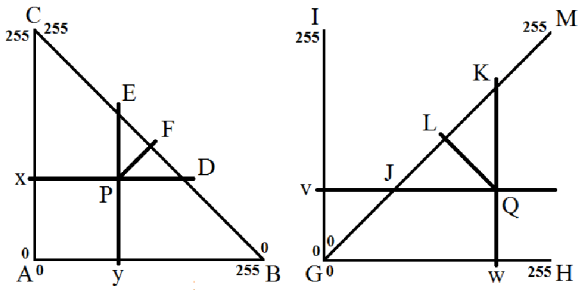
Pokud ale mezi dvěma klíčovými snímky dochází k více zákrytům, pak nevíme, kterou z cest mezi zákryty vybrat, protože obě budou mít přibližně stejnou cenu. Situace se dá upravit zadáním pevné konstanty, ale v testovacích videích mnohdy nastaly případy, kdy bylo třeba zadat pevných konstant opravdu mnoho. Proto jsem při implementaci dynamického programování ještě experimentoval s tím, že do pravidla (3) přidám možnost, která způsobí, že při vystoupení ze zákrytu se k ceně cesty nepřičítá penalizace λ_o, ale vzdálenost $\lambda \cdot s'(x_o, x_i)$, kde o je index snímku, ve kterém objekt vstoupil do zákrytu. Podobný přístup je také popsán ve článku [5]. Tímto způsobem cesta vystupující ze zákrytu v pozici oranžového kolečka má v daném snímku cenu menší než je cena cesty vystupující v pozici modrého kolečka. Bohužel ale implementace nedávala uspokojivé výsledky, protože bylo o dost složitější nastavit správně hodnoty penalizací λ_o a λ. Tudíž jsem se ve výsledku rozhodl skloubit obě možnosti dohromady a implementaci algoritmu jsem provedl dle následujícího pravidla (22), které se podle dosavadních výsledků zdá být dobrým kompromisem obou metod.

$$s(x_{i-1}, x_i) = \begin{cases} \lambda \cdot s'(x_{i-1}, x_i) & \text{když objekt je stále viditelný} \\ \lambda_o & \text{když objekt vstoupí do zákrytu} \\ \lambda_r & \text{když objekt je stále v zákrytu} \\ \frac{1}{2}(\lambda \cdot s'(x_o, x_i) + \lambda_o) & \text{když objekt vystoupí ze zákrytu} \end{cases} . \qquad (22)$$

7.2.6 Integrální histogram

O výpočet integrálního histogramu se stará objekt `i_hist` třídy `IntegralHistogram`. O alokaci a uvolnění paměti pro vypočtená data se stará objekt samotného snímku, který objektu `i_hist` předá pouze ukazatel na data. Výpočet integrálního histogramu se striktně drží teorie kapitoly 4.

Aby byl výpočet co možná nejefektivnější, bylo třeba vymyslet způsob rychlého zisku promítnuté hodnoty na úsečku. Po pár testováních (mnohdy pomocí *sin* a *cos* funkcí) jsem dospěl k následujícímu řešení, které je jednoduché, a přitom velmi rychlé. Základem myšlenky bylo uvědomit si, že hodnoty se nepromítají pomocí výpočtů na pravoúhlém rovnoramenném, ale na obecném rovnostranném trojúhelníku. Výpočet promítnutí barev všech pixelů snímku o velikosti 320x240 px na všech třináct jednodimenzionálních úseček nyní ve výsledku trvá pouze 2ms (u původní verze to bylo okolo 80ms).



Obrázek 21: Výpočet promítnutí hodnoty bodu na úsečku v prostoru.

Určení hodnoty barvy bodu promítnutého na úsečku v RGB prostoru se dá vypočítat pomocí dvou až čtyř sčítání a jednoho dvojkového dělení. obrázek 21 vlevo ukazuje princip promítnutí hodnoty bodu $P=[x,y]$ na úsečku BC, neboli do bodu F (zajímá nás tedy hodnota velikosti $|BF|$). Je nutné si uvědomit, že $|AB|=|AC|=|BC|$ (pro názornost je trojúhelník nakreslen jako pravoúhlý rovnoramenný, neboli tak jak si ho každý napoprvé v prostoru představí, viz obrázek 9a). Díky tomu platí, že $x=|BD|$ a $y=|CE|$. Velikost $|BF|$ se poté vypočítá následovně:

$$|BF|=|BD|+\big(|BC|-|BD|-|CE|\big)/2$$
$$|BF|=x+\big(255-x-y\big)/2.$$

Pokud je promítaný bod P vně trojúhelníku ABC, je potřeba rovnici trochu předělat:

$$|BF|=|BC|-|CE|+\big(|BD|+|CE|-|BC|\big)/2$$
$$|BF|=255-y+\big(x+y-255\big)/2.$$

Pro druhou úsečku (obrázek 21 vpravo) je poté vztah pro promítnutí bodu Q do bodu L ještě jednodušší:

$$|GL|=|GJ|+\big(|GK|-|GJ|\big)/2, \text{ pro } Q\in GHM$$
$$|GL|=|GK|+\big(|GJ|-|GK|\big)/2, \text{ pro } Q\in GIM.$$

Další vymyšlená optimalizace rychlosti práce s integrálním histogramem spočívá ve výpočtu jeho normalizace. Při extrakci histogramu regionu snímku ze struktury integrálního histogramu je třeba vždy získané hodnoty normalizovat. Normalizační konstantu sice lze spočítat součtem všech hodnot

v histogramu, ale jelikož se pro každý pixel regionu navýší hodnota histogramu o jedničku na jednu 1D úsečku protínající RGB prostor, je vše jednodušší. Normalizační konstantu lze poté získat jako $C = 1/(whn)$, kde w a h jsou velikosti regionu a n je počet 1D úseček.

7.2.7 AdaBoost algoritmus

AdaBoost algoritmus dostane trénovací data ve dvou polích struktur. Jedná se o jednorozměrné pole `*samples`, struktury s názvem `Sample`, a dvourozměrné pole `**binSamples`, struktury s názvem `Bin` (viz zdrojový kód 1). Velikost pole `*samples` je rovna množství předaných vzorků, stejně jako druhý rozměr pole `**binSamples`. Jeho první rozměr je roven počtu sloupců výsledného histogramu barev (v tomto případě 104). Pole `**binSamples` je tedy maticí, kde jeden její řádek při předání odpovídá histogramu barev jednoho vzorku, na který se v každé buňce odkazuje ukazatel `*s_id`. Její jeden sloupec odpovídá hodnotám jednoho sloupce histogramů všech trénovacích vzorků. Rozdělení na dvě pole struktur je důležité kvůli tomu, aby se váhy vzorků mohly upravit jen jednou a výsledek se projevil u všech sloupců daného vzorku v `**binSamples`.

```
struct Sample {
  int    label;          //id vzorku - pozitivní/negativní
  double weight;         //váha vzorku
};

struct Bin {
  double val;            //hodnota sloupce daného vzorku
  Sample *s_id;          //ukazatel na vzorek
};
```

Zdrojový kód 1: Struktury použité pro uložení trénovacích vzorků AdaBoost algoritmu.

Kvůli extrakci optimálního prahu daného sloupce v `**binSamples` se musí hodnoty každého řádku individuálně seřadit, jak je popsáno v kapitole 5.1. K řazení je použit *quick sort* algoritmus a hodnoty matice `**binSamples` se před řazením zkopírují do nově alokovaného dvojrozměrného pole, protože se řazením rozhodí struktura matice typu *histogram na řádek*. Zkopírovaná matice s originálními hodnotami se poté využívá k vyhodnocení úspěšnosti výsledného silného klasifikátoru.

V každém kroku t se pro všechny sloupce v matici určí práh. Práh s nejmenší chybou je poté přidán do výstupního klasifikátoru spolu s id sloupce a hodnotou α. Pro daný sloupec se poté již práh opět nepočítá. Někdy stačí na klasifikaci objektu s více jak 98% pravděpodobností na trénovacích datech využít pouze jeden slabý klasifikátor. Poté ale většinou docházelo k mnoha falešným detekcím, a proto jsem se rozhodl omezit minimální počet vybraných sloupců na dva.

Kromě vyřazení nestálených sloupců histogramů barev trénovacích vzorků, popsaných v kapitole 3.2.3, jsem musel ještě z trénovacího procesu vyřadit sloupce, které mají průměrnou hodnotu příliš blízko k nule (menší než 0,05 u normalizovaného histogramu). Toto opatření jsem zavedl proto, že AdaBoost algoritmus velmi často vybíral k trénování sloupce, jež mají práh například o hodnotě 0,005 a průměrná hodnota sloupce byla 0,0026. To znamená, že jak u pozitivních, tak u negativních vzorků byly hodnoty daného sloupce malé a blízko u sebe (i když rozlišitelné oním prahem), což také značí nestálost.

7.2.8 Mean Shift algoritmus

Inicializace mean shift algoritmu se provádí se zadanou výškou a šířkou vyhledávacího okna (obdélníku objektu). Vytvoří se matice hodnot profilu Epanechnikova jádra k_E a jeho derivace g, jež jsou popsané v kapitole 6.2. Zdrojový kód 2 ukazuje výpočet profilů, kde do proměnné `pow_dist` se ukládá výsledek rovnice (17). Jelikož většina termů v rovnici se navzájem vyruší, může být výpočet naprogramován ve zjednodušené verzi. Hodnoty Epanechnikova profilu jsou uložené v matici s názvem `epanech` a jsou primárně využity v rovnicích (14) a (15). Hodnoty uniformního profilu jsou v matici `uniform` a využívají se pro výpočet rovnice (21). V implementované aplikaci se využívá růst pomocí mean shift algoritmu pouze s konstantní velikostí vyhledávacího okna, tudíž se dané hodnoty profilů spočítají pouze jednou pro každý klíčový a i-snímek, čímž je ušetřen potřebný výpočetní výkon.

```
for (x = -half_width; x <= half_width; x++) {
  for (y = -half_height; y <= half_height; y++) {

    pow_dist = pow(x/half_width,2.0) + pow(y/half_height,2.0);

    if (pow_dist > 1) {   //nemusí se odmocňovat, sqrt(1) == 1
      epanech[x+half_width][y+half_height] = 0.0;
      uniform[x+half_width][y+half_height] = 0;
    }
    else {
      epanech[x+half_width][y+half_height] = 1.0 - pow_dist;
      uniform[x+half_width][y+half_height] = 1;
    }
  }
}
```

Zdrojový kód 2: Výpočet Epanechnikova (`epanech`) a uniformního (`uniform`) jádra.

Cílový a kandidátní model je v algoritmu představován vlastním objektem třídy `ColorModel`. Cílový model je spočten také vždy pouze jednou na každou růstovou trajektorii (cílové modely počátečních vzorků i-snímků jsou nyní již spočteny, viz kapitola 7.2.3), ale kandidátní model musí být spočten v každém kroku růstu. Ze všech barev pixelů vyhledávacího okna se vytvoří model stejně jako u extrakce histogramu vzorku pro trénování detektoru, jen s tím rozdílem, že se nyní nezvyšuje hodnota daného sloupce o jedničku, ale o hodnotu Epanechnikova profilu na dané pozici.

Kroneckerova delta funkce δ je ve výpočtu práce mean shift algoritmu využita dvakrát. Poprvé v rovnicích (14), (15) pro extrakci modelů a podruhé v rovnici (20) pro výpočet vah. Jelikož je v obou případech závislá na stejné neznámé x_i, je rychlejší si její výsledek při extrakci kandidátního modelu uložit do pole, než ji opět počítat při určení vah. Objekt třídy `ColorModel` tedy ve výsledku nevrací pouze model regionu snímku, ale také pole hodnot Kroneckerovy delta funkce.

Samotný výpočet jedné iterace posunutí středu popisuje zdrojový kód 3. Výpočet je rychlý hlavně kvůli předem spočtenému uniformnímu profilu. V proměnných `dx` a `dy` je spočten výsledný posun středu na osách x a y. Hodnotou uniformního jádra by se správně měly násobit vypočtené váhy, ale jelikož jádro obsahuje pouze hodnoty 0 a 1, může tak být použito i v podmínce. Za zmínku stojí také to, že výpočet modelu i posun do lokálního maxima jsou závislé na velikosti vyhledávacího okna, tudíž rychlost práce celého mean shift algoritmu je nepřímo úměrně závislá na této velikosti (viz popsané výsledky v kapitole 9.1).

```
w_sum = 0, x_sum = 0, y_sum = 0;

for (x = -half_width; x <= half_width; x++) {
  for (y = -half_height; y <= half_height; y++) {

    if (uniform[x+half_widht][y+half_height] == 1) {
      x_sum += x * weights[x+half_widht][y+half_height];
      y_sum += y * weights[x+half_widht][y+half_height];
      w_sum += weights[x+half_widht][y+half_height];
    }
  }
}

dx = x_sum/w_sum;
dy = y_sum/w_sum;
```

Zdrojový kód 3: Jedna iterace posunutí středu na základě vah a uniformního profilu.

Postup práce mean shift algoritmu popsaný v číslovaném seznamu v kapitole 6.2.2 může být urychlen následujícím způsobem. Krok 5) je ve výpočtu obsažen proto, že nalezení pozice pomocí mean shift procedury (21) nutně nezvýší hodnotu podobnosti modelů $\rho(y) \equiv \rho[p(y), q]$. Avšak z praktického hlediska, zjištěného ve článku [7], se podobnost zvýší v 99,9% případů. Tudíž lze celý iterační postup redukovat na následující posloupnost kroků:

1) Výpočet cílového modelu $q = \{q_u\}_{u=1..m}$ v bodě y_0 a přechod na následující snímek.
2) Výpočet kandidátního modelu $p(y_0) = \{p_u(y_0)\}_{u=1..m}$.
3) Zisk vah $\{w_i\}_{i=1..n}$ pomocí rovnice (20).
4) Nalezení pozice kandidátního modelu y_1 pomocí mean shift procedury (21).
5) Pokud platí $\|y_1 - y_0\| < \varepsilon$, bylo dosaženo požadované přesnosti a výpočet se pro aktuální snímek videa ukončí. Jinak se provede aktualizace pozice $y_0 \leftarrow y_1$ a pokračuje se krokem 2.

Hodnota prahu ε je nastavena tak, aby se výpočet ukončil, když pozice y_0 a y_1 jsou totožné (v pixelech). Dále je ještě dobré pro zrychlení výpočtu nastavit podmínku ukončení cyklu, pokud pozice y_0 a y_1 začnou oscilovat.

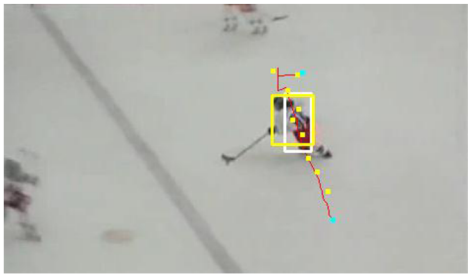
7.3 Implementace nástroje pro vyhodnocování vlastností sledovacího algoritmu

Aby bylo možné číselně posoudit úspěšnost sledování, bylo potřeba vytvořit nástroj, který porovná referenční trajektorii objektu s trajektorií vytvořenou sledovacím algoritmem. Pro jednodušší práci je algoritmus přímo implementován v hlavní aplikaci, což znamená, že se uživatel po vytvoření výsledné trajektorie může bez problémů přepnout do vyhodnocovacího módu.

Jak již bylo zmíněno výše, algoritmus porovnává vypočtenou trajektorii s referenčními snímky (e-snímky), zadanými uživatelem. Výstupem algoritmu několik hodnot vypsaných na standardní výstup. První hodnotou je přesnost trajektorie vypočtená pomocí metriky *F-measure* [23] (viz níže v sekci 7.3.1), druhou je *vzdálenost středů v procentech* (viz sekce 7.3.2). Dále se počítá úplnost a přesnost sledování objektu (sekce 7.3.3). Techniky se použijí na porovnání každého referenčního snímku se vzorkem finální trajektorie nebo kandidáty na objekt a vypočtené hodnoty se zprůměrují.

Aby uživatele při zadávání e-snímků nelákalo si referenční trajektorii přikrášlovat k lepšímu, není při jejich zadávání výsledek sledování zobrazen. Po výpočtu se zobrazí jak výsledná trajektorie, tak e-snímky, a trajektorii lze porovnat i pouhým pohledem (viz obrázek 22). Trajektorie je opět zobrazena pomocí červené čáry a referenční snímky pomocí středů. V aktuálním snímku jsou oba vzorky vykresleny také pomocí obdélníků.

Přepnutí mezi vyhodnocovacím a normálním módem aplikace se provádí pomocí jednoho tlačítka a uživatel změnu pozná tak, že se zavřou všechna nastavovací okna pro dynamické programování a také druhé okno s videem. Ovládání vyhodnocovacího algoritmu je popsáno v příloze A.



Obrázek 22: Zobrazená výsledná trajektorie (červená čára a bílý obdélník) spolu s e-snímky (žluté tečky a obdélník).

7.3.1 F-measure metrika

Metrika *F-measure* (také F-míra) je definována jako harmonický průměr přesnosti (*precision*) a úplnosti (*recall*) a má vzorec

$$F = \frac{2PR}{P+R}, \tag{23}$$

kde P značí přesnost a R úplnost, pro jejichž výpočet se musí prvně určit obsah I průniku dvou porovnávaných obdélníků/vzorků. Poté platí, že $P = I/S_T$ a $R = I/S_R$, kde S_T je obsah vzorku T ve spočtené trajektorii a S_R je obsah daného referenčního vzorku R.

Harmonický průměr se k měření přesnosti hodí daleko víc než aritmetický, jenž je definován jako $A = 1/2 \cdot (P+R)$. Máme-li dva obdélníky, kde $S_R = 1{,}0$, $S_T = 0{,}2$ a jejich průnik $I = 0{,}2$ (neboli T je uvnitř S), pak intuitivně by měla být přesnost velmi malá. Po potřebných výpočtech se $A = 0{,}6$ a

$$F = \frac{2 \cdot \frac{0{,}2}{0{,}2} \cdot \frac{0{,}2}{1{,}0}}{\frac{0{,}2}{0{,}2} + \frac{0{,}2}{1{,}0}} = \frac{0{,}4}{1{,}2} = \frac{1}{3} = 0{,}333,$$

což je daleko rozumnější výsledek. Obecnou definici aritmetického a harmonického průměru je možno nalézt ve článku [23].

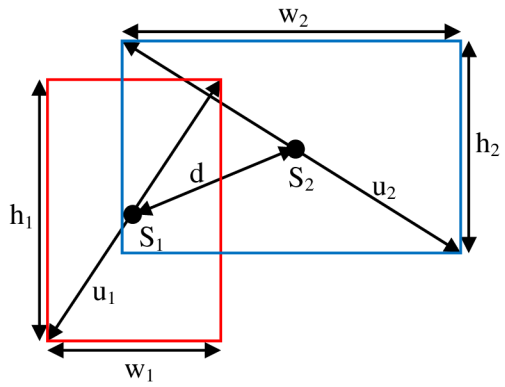
7.3.2 Vzdálenost středů

Do vyhodnocení jsem přidal ještě míru vzdálenosti středů hlavně proto, že implementovaný algoritmus nikterak nezahrnuje sledování objektu za pomocí změny měřítka vyhledávacího okna (tudíž je odkázán pouze na velikosti zadané uživatelem), a i když je objekt detekován docela přesně, rozměry obdélníků si nemusí vždy odpovídat (viz obrázek 22). Procentuální vzdálenost středů se od klasické

euklidovské vzdálenosti liší v tom, že do svého výpočtu zahrnuje velikosti obdélníků a to podle aritmetického průměru

$$D = \frac{1}{2} \cdot \left(\frac{d}{u_1} + \frac{d}{u_2} \right), \tag{24}$$

kde hodnoty d, u_1 a u_2 odpovídají hodnotám na následujícím obrázku 23. Musí se ovšem ošetřit situace, kdy zlomky v rovnici budou mít výsledek větší než 1. Přičtení více jak stoprocentního rozdílu je nesmyslné a značně zkresluje výsledky, tudíž kdykoli hodnota zlomku překročí 1, je vždy započítána jako 1. Tímto způsobem je zaručeno, že výsledek stejně jako u metriky *F-measure* nezávisí na velikosti obdélníků, jen s tím rozdílem, že nyní je nejlepší výsledek 0%, u *F-measure* metriky to bylo 100%.



Obrázek 23: Pomocný obrázek pro určení vzdálenosti středů dvou obdélníků.

7.3.3 Úplnost a přesnost sledování

Ve článku [30] jsou použita dvě různá měření úspěšnosti sledování objektu. Jmenují se stejně jako vlastnosti používané k výpočtu metriky *F-measure* a to proto, že mají podobné vlastnosti. K jejich výpočtu je potřeba určit, kdy je kandidátní vzorek sledován správně a kdy ne. Správně sledovaný vzorek je každý vzorek, který má hodnotu metriky *F-measure* od referenčního vzorku větší než 50%. Přesnost sledování je poté poměr správně sledovaných vzorků oproti všem kandidátním vzorkům a úplnost značí procentuální počet snímků, ve kterých je alespoň jeden vzorek sledován správně. Dále se k vyhodnocení práce detektoru ve zmíněném článku využívá ještě jedno měření, které určuje procentuální míru negativně klasifikovaných vzorků v i-snímcích, neboli udává procento ušetřeného času oproti nákladnému prohledávání celého snímku.

Všechna tato měření jsou ve vyhodnocovacím nástroji také implementována.

7.4 Rozšíření dynamického programování

Při testování algoritmu pro vytvoření finální trajektorie vše pracovalo bez problémů, ale v některých případech algoritmus preferoval cestu, kde byly v několika po sobě jdoucích snímcích vzorky na stejné pozici a poté skočil na některý vzdálenější snímek, než aby postupně šel po snímcích ve směru pohybu objektu. Situace by se dala popsat následující tabulkou.

číslo snímku:	1		2		3		4		5		6		7		8	
vzorek 1:	0,0	nan	0,0	1	0,0	1	0,0	1	0,0	1	0,8	1	1,2	1	**1,6**	**?**
vzorek 2:	nan	nan	0,2	1	0,4	2	0,6	2	0,8	2	1,0	2	1,2	2	nan	nan

Tabulka 2: Ilustrativní případ, kdy skoková cesta může mít přednost před plynulou.

Vezmeme-li v úvahu, že vzorky 1 a 2 ve snímku 7 mají stejnou Euklidovskou vzdálenost od vzorku 1 ve snímku 8, pak podle algoritmu dynamického programování nastává situace, kdy nevíme, kterou cestu vybrat (cestu o posloupnost vzorků {1, 1, 1, 1, 1, 1, 1, 1} budeme dále v textu nazývat první cestou a cestu o posloupnosti {1, 2, 2, 2, 2, 2, 2, 1} druhou). Vzdálenosti jsou sice počítány na desetinná místa, tudíž přesná shoda nastane málokdy, ale i kdyby byla cena druhé cesty horší jen o pouhých několik setin, stejně bychom ji raději chtěli preferovat před první cestou.

Hlavně pro tento případ, jsem do své aplikace přidal mnou vymyšlený interaktivní algoritmus, rozšiřující algoritmus DP, který se snaží danou situaci vyřešit tím, že stagnující cestě přičítá uživatelem modifikovatelnou penalizaci a pohybující se cestě ji naopak od ceny odečítá. Největším problémem bylo určit, co je to stagnující a co pohybující se cesta. Danou situaci jsem se pokusil vyřešit tak, že jsem spočítal průměrnou pozici středových bodů všech kandidátních vzorků v n-tém následujícím snímku videa, čímž jsem získal přibližný směr pohybu objektu a pozici každého vzorku v předchozím snímku, k němuž se aktuálně počítá vzdálenost $s(\cdot)$, jsem trochu posunul směrem ke spočtenému průměru. Nastavení nešlo nechat na nějakých konstantních hodnotách, protože se jedná o poměrně velký zásah do vypočtené trajektorie, který se na každém videu projeví trochu jinak, a někdy i negativně. Tudíž uživatel ve výsledku musí určit, zda chce danou metodu použít, a s jakým nastavením.

Nastavují se dva parametry, o kolik snímků dále se má počítat průměrný středový bod a jaká míra spočtené vzdálenosti se má ke vzorkům přičíst ($0-100\%$). Počet snímků je omezen na rozsah $0-30$, kde 0 značí situaci, kdy se metoda použít nemá. Větší rozsah jak $15-20$ již mnohdy není použitelný, ale ponechal jsem uživateli jistou rezervu. Případ popsaný v tabulce 2 je pomocí daného rozšíření upraven například následovně:

číslo snímku:	1		2		3		4		5		6		7		8	
vzorek 1:	**0,00**	**nan**	0,05	1	0,10	1	0,15	1	0,20	1	0,95	1	1,30	1	**1,25**	**2**
vzorek 2:	nan	nan	**0,15**	**1**	**0,30**	**2**	**0,45**	**2**	**0,60**	**2**	**0,75**	**2**	**0,90**	**2**	nan	nan

Tabulka 3: Ilustrativní případ, cesty upravené rozšiřujícím algoritmem.

Ke každému stagnujícímu kroku cesty se přičetla cena 0,05 a od každého pohybujícího se kroku se stejná cena odečetla (u druhé cesty se ve všech snímcích hodnota odečetla a u první se ve snímcích 2 až 5 přičetla a ve snímcích 6 a 7 se odečetla). Ve výsledku je vybranou cestou námi preferovaná cesta druhá. Rozšíření nemá na časovou náročnost algoritmu dynamického programování takřka žádný vliv. Dalo by se říct, že rozšíření ušetří uživateli práci tím, že nemusí nastavovat tolik pevných konstant.

Dalším problémem, který při testování DP nastal, bylo to, že sousední vzorky se stejnými pozicemi měly od sebe vzdálenost $s(\cdot)$ nulovou, a tudíž bylo nemožné přerušit cestu mezi nimi pomocí zadání nízkých hodnot penalizací. Nejmenší vzdálenost $s(\cdot)$, která může nastat kromě nuly, je 1,0, a proto, aby nebyla přerušena priorita vzorků se stejnými pozicemi nad vzorky s jinými, ale i tak mohla být cesta přerušena pomocí DP, jsem vzorkům se vzdáleností nula zvětšil hodnotu cesty na $0{,}9\lambda$ (0,9 jsem zvolil proto, že ve většině videí se sleduje pohyblivý objekt, a proto stagnující trajektorie není příliš žádaná). Tímto způsobem získává uživatel v nastavení hodnot dynamického programování daleko větší volnost.

8 Doporučený postup práce uživatele

Jakmile se uživatel blíže seznámí s aplikací, zjistí, že postup práce, jenž vede k vytvoření finální trajektorie, je vždy velmi podobný. Základem je vždy vybrat klíčové snímky nejlépe tak, aby každý obsahoval trochu jiné pozadí, které se ve videu vyskytuje. Pokud objekt mění razantně svoji velikost nebo pozici vůči kameře v některém úseku videa, je také dobré do nich přidat k-snímky, aby interpolovaná velikost vyhledávacího okna pro růst trajektorie přibližně odpovídala tvaru sledovaného objektu v daném úseku (jak je tomu na následujícím obrázku 24). Když se sleduje například člověk, který často mění pozici svých končetin, je užitečné nastavit klíčové snímky tak, aby zahrnovaly zejména trup člověka, čímž se zaručí, že trénování detektoru bude méně ovlivněno prvky na pozadí.

Obrázek 24: Ilustrace principu zadávání k-snímků. Postupně zleva: Původní k-snímek, v němž začíná sledování. K-snímek přidaný kvůli velké změně barev na pozadí. K-snímek určující změnu natočení objektu vůči kameře. K-snímek určující vzdálení se objektu od kamery a opětovnou změnu pozadí.

V dalším kroku, předtím než se začnou jakkoli nastavovat parametry dynamického programování, by si měl uživatel prohlédnout celý úsek videa a přidat, pokud je to možné, k-snímky tam, kde není objekt správně sledován (na jeho pozici nejsou zobrazeny žádné kandidátní vzorky), a opět spustit výpočet možných trajektorií.

Až je vše v pořádku, je dobré zkusit najít snímky, kde dochází ke vstupu a výstupu objektu ze zákrytu, a označit jeho pozici pevnou konstantou, i když je v daném úseku pozice objektu detekována správně (například v místech zelené a modré tečky v obrázku 20). Díky tomu je poté jednodušší nastavit parametry DP. U parametrů dynamického programování by se měla prvně měnit hodnota penalizace λ_o tak, aby výsledná trajektorie dávala při vstupech a výstupech objektu ze zákrytu schopné výsledky, a vše by se poté mělo jemněji doladit pomocí nastavení hodnoty penalizace λ.

Nyní nachází okamžik využití rozšíření DP, u kterého se celkem okamžitě dá zjistit, jestli je použitelné nebo ne. Rozšíření je dobré používat jen v případě, pokud po zadání malé míry přičítané vzdálenosti a použití výpočtu středu o několik snímků dopředu dává dobré výsledky (mnohdy jsou lepší výsledky při dívání se dále dopředu než blíže - více než 7-8 snímků). Poté by se měla trochu upravit nastavená míru přičítání spolu s hodnotami penalizací.

Když se daným způsobem nepodaří vytvořit trajektorii správně, měly by se přidat nové pevné konstanty v potřebných úsecích a opět pozměnit hodnoty penalizací a rozšíření. Případně přidat nové klíčové snímky. U přidávání pevných konstant je dobré plně vypnout zobrazení výsledné trajektorie a vykreslení obdélníku kandidátních vzorků (aby zůstaly vykresleny pouze jejich středy). Scéna je poté přehlednější.

Všechny tyto kroky by se poté měly opakovat, dokud není uživatel spokojen s výsledkem. Dle mého testování, zkušený uživatel dokáže získat optimální trajektorii objektu již v několika málo iteracích.

9 Finální výsledky a vyhodnocení

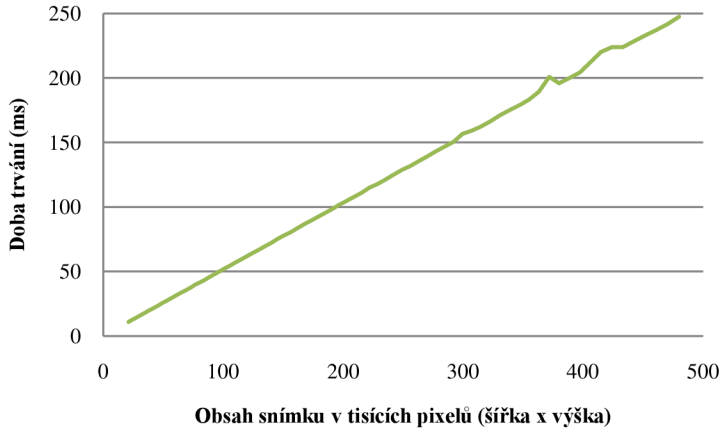
V této kapitole jsou popsány některé závěry a výsledky vytvořené aplikace spolu s jejich vyhodnocením pomocí příslušného implementovaného nástroje. První sekce se zabývá popsáním výkonnosti implementovaného systému. Druhá charakterizuje robustnost detekčního algoritmu a ve třetí je diskutována úspěšnost detekce objektu v jednoduchém videu. Poté následuje prezentace finálních výsledků spolu s jejich vyhodnocením na několika testovacích videích. Video jsou stažena ze serveru youtube.com, pokud tomu není uvedeno jinak.

Jelikož se jedná o interaktivní algoritmus, ve kterém všechna důležitá nastavení provádí uživatel, nebylo možné vytvořit skript, který určitým způsobem vyhodnotí úspěšnost sledování na základě různých variací hodnot parametrů. Proto jsem automaticky, prostřednictvím dočasného upravení kódu, určil pouze časovou náročnost algoritmu v závislosti na velikosti objektu a rozlišení videa. Vyhodnocení úspěšnosti sledování je poté prováděno vždy ručně za pomoci implementovaného nástroje.

9.1 Výkon aplikace

Pro vyhodnocení výkonnosti vytvořené aplikace jsem vytvořil několik grafů, které se snaží vystihnout klíčové závislosti doby běhu programu na jednotlivých fázích výpočtu. Veškeré testy jsou prováděné na notebooku s 2,00GHz procesorem a se 4,00GB pamětí. V prvních dvou případech jsou měření opakována stokrát při stejném nastavení a poté průměrována. Ve třetím případě bylo nutno algoritmus spouštět ručně, proto jsou výsledky získány zprůměrováním pouze pěti měření na několika testovacích videích (výsledné hodnoty byly vždy velmi podobné, tedy lze předpokládat vysokou přesnost měření).

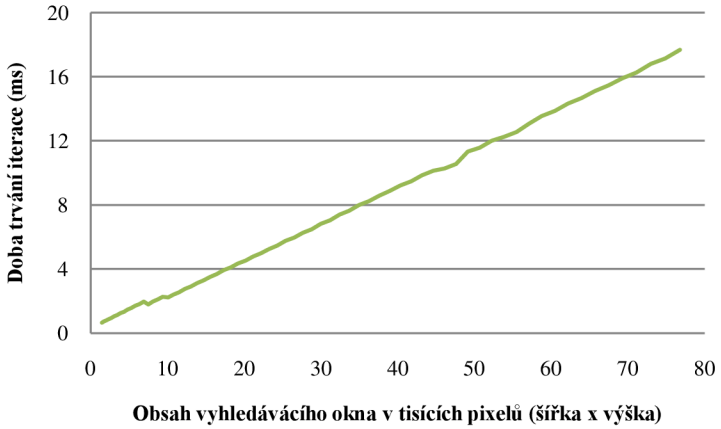
Na grafu 1 je ukázána závislost doby výpočtu integrálního histogramu na rozlišení snímku videa. Jak je vidět, výpočetní doba roste lineárně s velikostí, kde pro snímek s rozlišením $800 \times 600 = 480k$ pixelů je doba výpočtu okolo 250ms a pro rozlišení $480 \times 360 = 172{,}8k$, které je v následujících testech a vyhodnoceních často používané, je to přibližně 90ms.



Graf 1: Doba trvání výpočtu integrálního histogramu v závislosti na velikosti snímku.

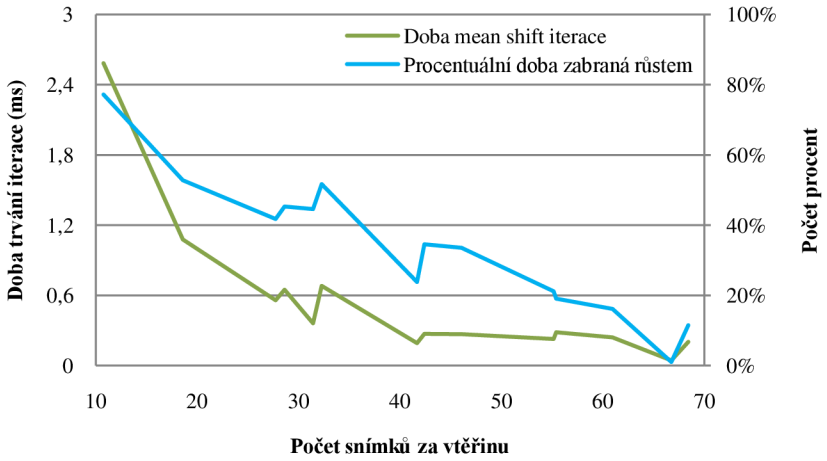
Graf 2 ukazuje dobu výpočtu jednoho kroku iterace mean shift algoritmu v závislosti na velikosti vyhledávacího okna. Opět je vidět, že doba lineárně roste s velikostí, jak je očekáváno. Pro okno o velikosti $320 \times 240 = 76{,}8k$ vychází doba výpočtu přibližně na 17,5ms, avšak v dalších testech

a vyhodnoceních, popsaných níže, se objekty větší jak $150 \times 100 = 15k$, u kterých je doba výpočtu pod 3,5ms, nesledují. Je však málo pravděpodobné, že algoritmus nalezne lokální maximum pouze v jedné iteraci, proto se doba jeho práce liší od vzorku ke vzorku.



Graf 2: Doba trvání výpočtu jedné iterace mean shift algoritmu v závislosti na velikosti vyhledávacího okna.

Následující vyhodnocení jsou prováděna na čtrnácti testovacích videích o rozlišení 480×360 pixelů, která lze nalézt na přiloženém DVD (viz příloha C). Implementovaný sledovací systém dokáže běžet rychlostí 15-70 snímků za vteřinu v závislosti na velikosti sledovaného objektu (viz dále). Trénování detektoru trvá průměrně 70ms na jeden klíčový snímek a vybírá se v rozmezí 2-11 slabých klasifikátorů (průměrně 4,1). Extrakce vzorků ze struktury integrálního histogramu zabírá průměrně 40ms a samotná klasifikace vzorků je poté již velmi rychlá (0,5ms na i-snímek). Průměrný počet iterací mean shift algoritmu, po kterém se nalezne lokální maximum, je 3,75 a rychlost práce dynamického programování je pouhých 0,1ms na snímek.



Graf 3: Doba trvání kalkulace jedné iterace mean shift algoritmu (zelená čára, levá svislá osa) a procentuální část trvání výpočtu trajektorie zabraná časově závislým růstem trajektorie (modrá čára, pravá svislá osa) v závislosti na době práce aplikace vyjádřené v množství spočtených snímků za vteřinu.

Zdá se, že časy výpočtu jednoho růstu mean shift algoritmu jsou velmi malé, ale ve výsledku kalkulace časově závislého růstů trajektorií nejvíce ovlivňuje kolísání výpočetní doby celého sledování. Na grafu 3 je znázorněna celková doba výpočtu finální trajektorie ve snímcích za vteřinu, která

je vykreslena v závislosti na době výpočtu jedné iterace mean shift algoritmu a na době zabrané růstem trajektorie v procentech celkového času výpočtu. Data znázorněná v grafu jsou vytvořena pomocí oněch čtrnácti testovacích videí. U každého videa je pokaždé použita jiná velikost vyhledávacího okna a podmínky výpočtu jsou vždy lehce změněny různým datovým obsahem videí. Výpočetní doba integrálního histogramu je konstantní, jelikož je použito vždy stejné rozlišení videa a doba trénování detektoru nemá na výsledný počet snímků za vteřinu moc velký vliv. Je zřetelné, že čím je doba trvání jedné iterace mean shift algoritmu menší (tím pádem i velikost vyhledávacího okna objektu), tím roste celková rychlost výpočtu. Mohlo by se zdát, že rychlost výpočtu v grafu roste takřka exponenciálně, ale graf 2 potvrzuje lineární růst, tudíž je křivka ovlivněna pouze různými vlastnostmi videí. Procentuální doba znázorněná na grafu 3, jež je zabraná růstem, také odpovídá velikosti vyhledávacího okna, což dokazuje, že časově závislý růst trajektorie nejvíce ovlivňuje kolísání rychlosti výpočtu celého programu.

Paměťová náročnost aplikace závisí hlavně na rozlišení videa a počtu klíčových snímků, které zabírají majoritní množství paměti hlavně proto, že se pro každý z nich uchovává vypočtený integrální histogram, jehož struktura zabírá 416 bajtů na jeden pixel snímku[6]. Neboli pro jeden snímek s rozlišením 480×320 pixelů je potřebná paměť pro uchování IH 60MB. U i-snímků se po navzorkování paměť zabraná strukturou vždy uvolní.

Hlavní rozdíl mezi implementovaným systémem a systémem popsaným ve článku [30] (referenčním) spočívá v tom, že momentálně velkou procentuální dobu kalkulace trajektorie zabírá výpočet struktury integrálního histogramu (optimalizace je diskutována v kapitole 10.1). V referenčním systému růst trajektorie zabírá přibližně 84% celkového času, přitom v implementovaném to je průměrně 33,7%. Výpočet integrálních histogramů zabírá průměrně 44% výpočetního času, vzorkování a klasifikace cca 11% a zbytek další nezbytná režie. Referenční systém dokáže na videích o rozlišení 320×240 pixelů pracovat rychlostí 60-100 snímků za vteřinu [30]. Implementovaný systém při tomto rozlišení pracuje rychlostí přibližně 40-100 fps.

9.2 Robustnost boosted color bin příznaku

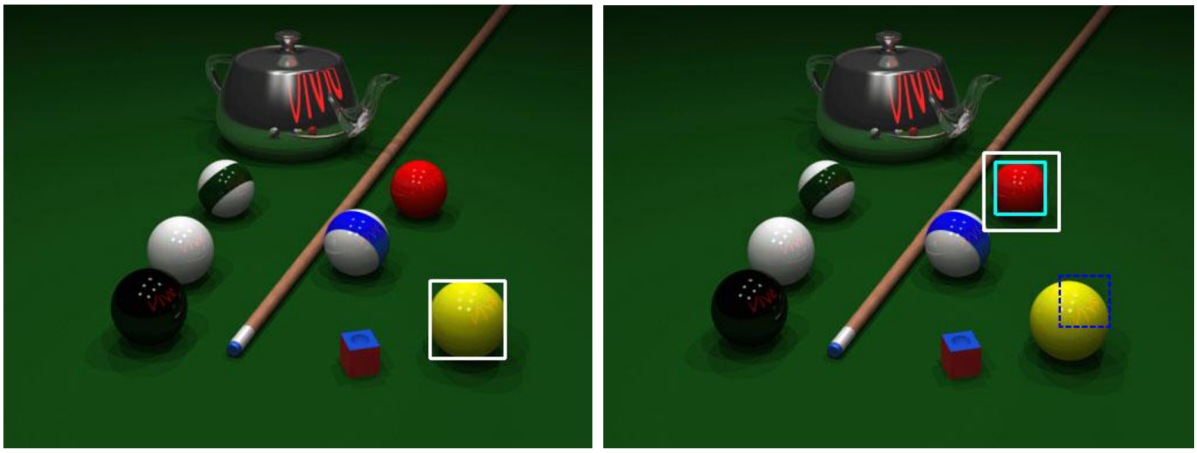
Při sledování objektu, jenž se svou barvou velmi liší od ostatních ve scéně, má AdaBoost na výběr mnoho barevných sloupců, které samy o sobě s dostatečnou úspěšností klasifikují objekt. Obrázek 26 znázorňuje histogram žluté kulečníkové koule zobrazené na obrázku 25 vlevo, kde každý z patnácti červeně a modře zvýrazněných barevných sloupců (slabých klasifikátorů) rozlišuje kouli od pozadí se 100% úspěšností. Pokud se trénovací data získají pouze z tohoto klíčového snímku, AdaBoost vybere jako silný klasifikátor jakýkoliv jeden z nich[7].

Když se podíváme na červenou kouli na obrázku 25 vpravo (bílý obdélník), hodnoty modře zvýrazněných sloupců jejího histogramu znázorněného na obrázku 27 se výrazně liší od korespondujících sloupců histogramu žluté koule. Vše by proběhlo bez problémů až na případ, když by AdaBoost algoritmus pro detekci žluté koule vybral jeden z pěti červeně označených sloupců. Výběr těchto sloupců sice v daném případě nijak neovlivní detekci žluté koule, protože normalizované hodnoty červeně označených barevných sloupců získaných z histogramu červené koule nejsou pro pozitivní klasifikaci dostatečně vysoké. Pokud se ale červená koule přiblíží více ke kameře, neboli zvětší se její měřítko

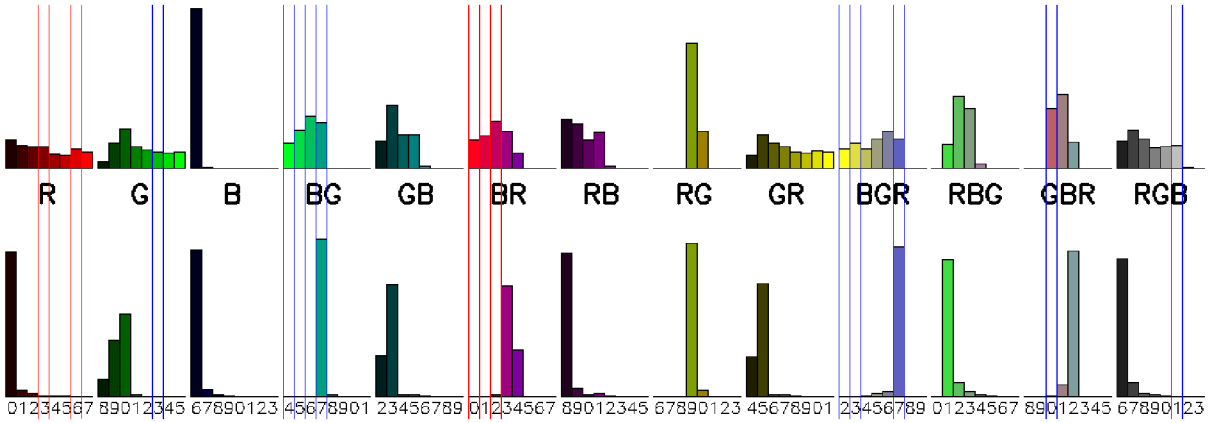
[6] Každá hodnota sloupce histogramu je uložena v proměnné typu `unsigned int`, která ve 32 bitovém operačním systému zabírá 4 bajty. Pro každý jeden pixel se uchovává 104 sloupců histogramu, což ve výsledku dává zmíněných 416 bajtů.

[7] Přesněji, AdaBoost algoritmus vybere jakoukoli kombinaci dvou označených sloupců, kvůli minimu popsanému v kapitole 7.2.7, ale pro zisk silného klasifikátoru s více jak 98% úspěšnosti na trénovacích datech opravdu stačí vybrat pouze jeden z nich.

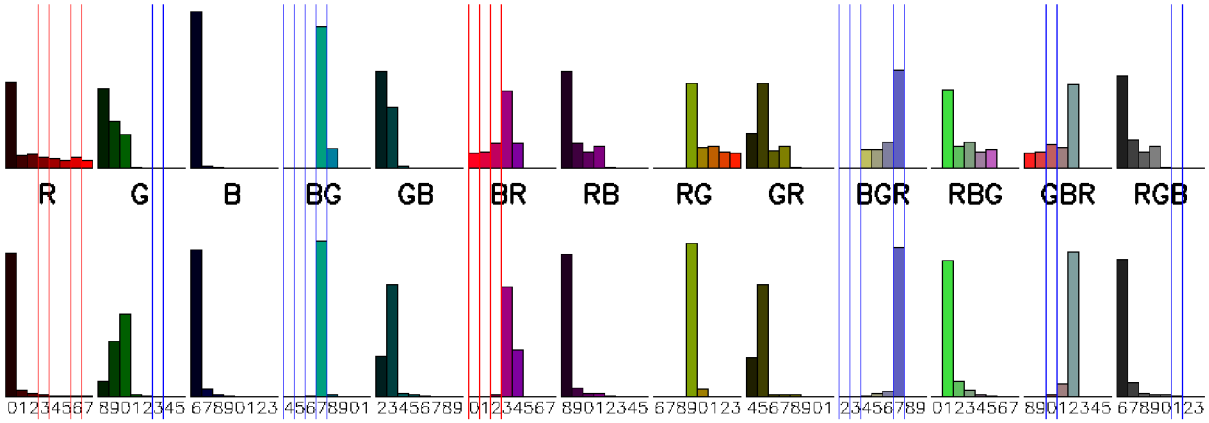
(což odpovídá také normalizovanému histogramu vypočteného z azurového obdélníku na obrázku 25 vpravo), hodnoty těchto sloupců se zvětší na přibližně stejnou hodnotu, jakou mají sloupce histogramu žluté koule, a dochází k falešné detekci.



Obrázek 25: Snímek kulečníkového stolu s vybranou žlutou koulí vlevo a červenou vpravo.



Obrázek 26: Histogram žluté kulečníkové koule (obrázek 25 vlevo) se zvýrazněnými nejvíce rozlišujícími sloupci (červeně i modře).



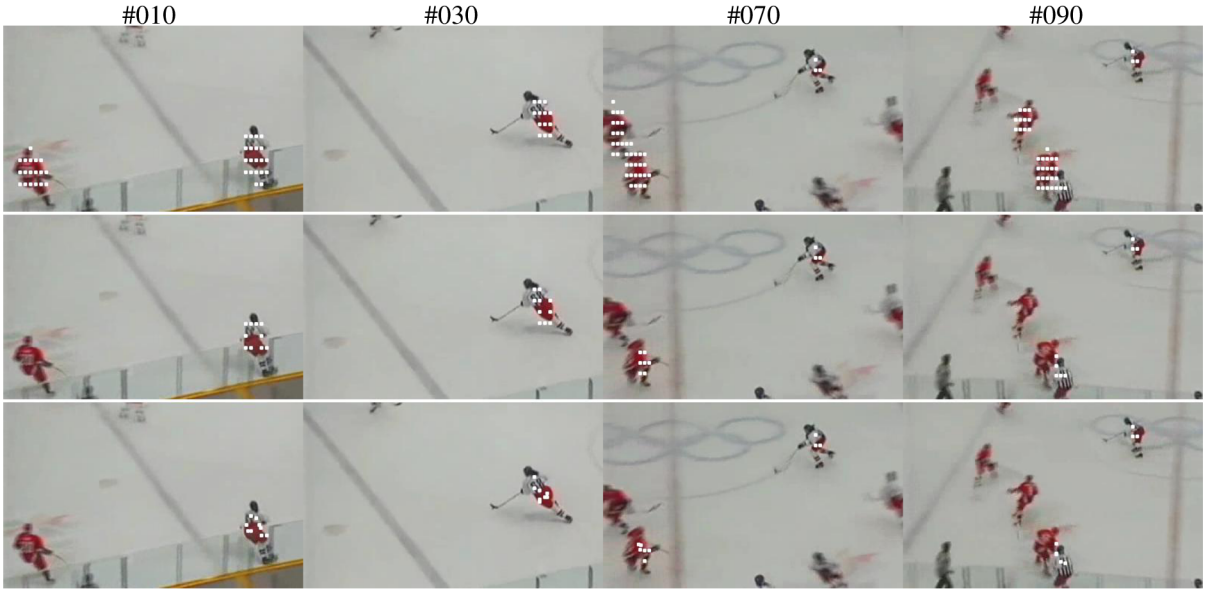
Obrázek 27: Histogram červené kulečníkové koule (obrázek 25 vpravo, bílý obdélník) se zvýrazněnými problematickými sloupci pro detekci žluté koule.

Pokud potom nastane například problém, že sledování přeskočí na červenou kouli a situace nepůjde napravit pomocí dynamického programování (což je velmi nepravděpodobné), uživatel může do problematického snímku vložit nový klíčový snímek, což způsobí, že s přidáním nových vzorků do trénovacího procesu poklesne u těchto pěti problematických barevných sloupců úspěšnost a AdaBoost nově vybere jiný sloupec, který klasifikuje žlutou kouli nejlépe (modře označený).

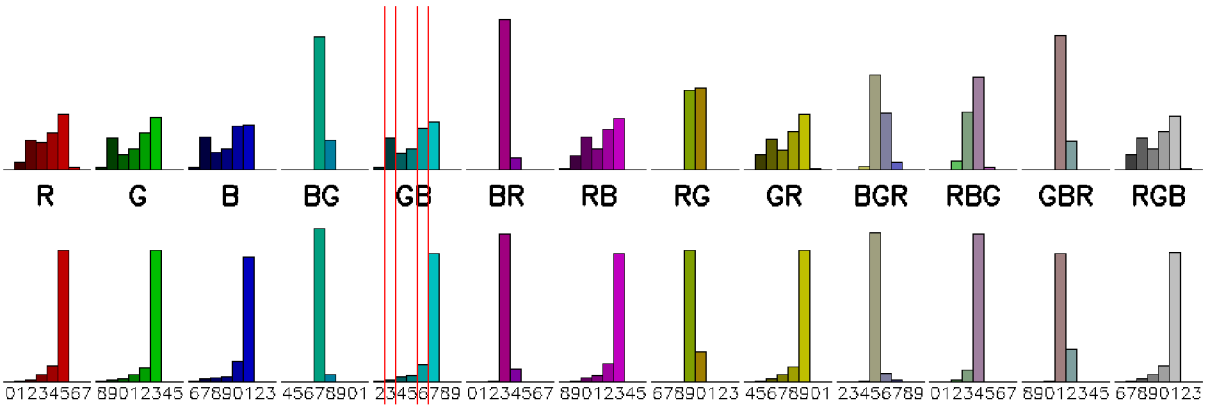
Popsaný problém nenastává, pokud uživatel bude chtít sledovat červenou kouli (azurový obdélník), protože AdaBoost červeně zvýrazněné sloupce nevybere kvůli negativním vzorkům získaným z pozice žluté koule (jako je například znázorněn modře čárkovaný obdélník na obrázku 25 vpravo).

9.3 Analýza práce detektoru

Na obrázku 28 je znázorněna detekce českého červenobílého hokejisty ve videu. Bílé tečky na obrázku znázorňují středy pozitivně klasifikovaných vzorků. Klíčovými snímky jsou snímky #000 a #099 a silný klasifikátor je sestaven ze dvou sloupců zvýrazněných na obrázku 29, které dávají přesně 99,0% úspěšnost na trénovacích datech.



Obrázek 28: „Hokejista**“**: Detekce pohybu hokejisty s pukem. První řádek obsahuje pouhou klasifikaci vzorků. Na druhém řádku jsou odstraněny vzorky s největší vzdáleností histogramů a na třetím řádku jsou vzorky posunuty do svých lokálních maxim pomocí mean shift algoritmu.



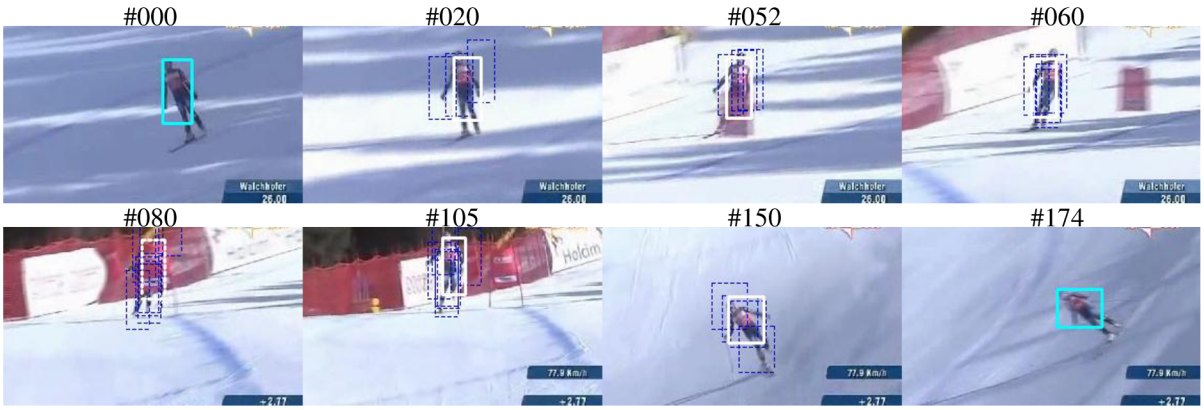
Obrázek 29: Histogram hokejisty v obrázku 28 zprůměrovaný z klíčových snímků #000 a #099.

Na prvním řádku v obrázku 28 je na vzorcích provedena pouhá klasifikace, a jak je vidět, dochází k mnoha falešným detekcím protihráčů v červeném, protože v k-snímcích byli protihráči většinou víceméně rozmazáni. Situace se razantně zlepší na druhém řádku, kde jsou vyřazeny vzorky, které mají největší Bhattacharyho vzdálenost histogramů. Jak je vidět, vzorky jsou primárně odstraněny z chybně klasifikovaných pozic. Na třetím řádku jsou zbylé vzorky posunuty do svých lokálních maxim. Obzvlášť na snímku #030 je zřetelně vidět, že posunem se přesnost detekce zvýšila. U zbylých špatně klasifikovaných vzorků je sice počítán růst trajektorie, ale ten je ve výsledku zastaven poměrně brzo, protože se po pár růstech zvýší vzdálenost jeho histogramu za povolenou mez. Detekce je ve výsledku úspěšná ve všech snímcích videa a algoritmus dynamického programování do své práce falešné detekce nezahrne.

9.4 Finální výsledky sledování

Tato kapitola obsahuje některé finální výsledky sledování vytvořené za pomocí implementovaného systému. Všechna videa lze nalézt na přiloženém DVD spolu s několika dalšími, které zde nejsou prezentovány. Videa jsou zobrazena pomocí posloupnosti osmi nebo čtyř snímků, kde prvním a posledním bývají klíčové snímky určující rozsah sledování. Další klíčové snímky zobrazeny nebývají, ale v textu k obrázkům jsou vždy jejich pozice uvedeny. Snímky jsou očíslovány podle jejich relativní pozice k prvnímu k-snímku. Mnohdy na obrázcích jsou zobrazeny okraje snímku obsahující pozadí, aby nebylo zabráno mnoho prostoru a objekt byl lépe vidět. Obdélníkem azurové barvy jsou zobrazeny klíčové snímky a čárkovaným modrým obdélníkem kandidáti na objekt. Kandidát vybraný pomocí dynamického programování je zvýrazněn bílým obdélníkem, a pokud se musela pozice objektu dopočítat, je vykreslena bílým čárkovaným obdélníkem.

Na obrázku 30 je zobrazeno poměrně jednoduché video, ve kterém se lyžař dvakrát dostane do zákrytu (ve snímku #052 do částečného a ve snímku #081 do plného). Sledování je provedeno úspěšně za pomocí tří klíčových snímků (#000, #116, #174), kde prostřední snímek je přidán proto, že v daném úseku videa se vyskytuje pozadí, které má podobné barvy jako samotný lyžař. Jak je vidět, oba zákryty jsou zpracovány správně, stejně jako i sledování v problematickém úseku (#105).

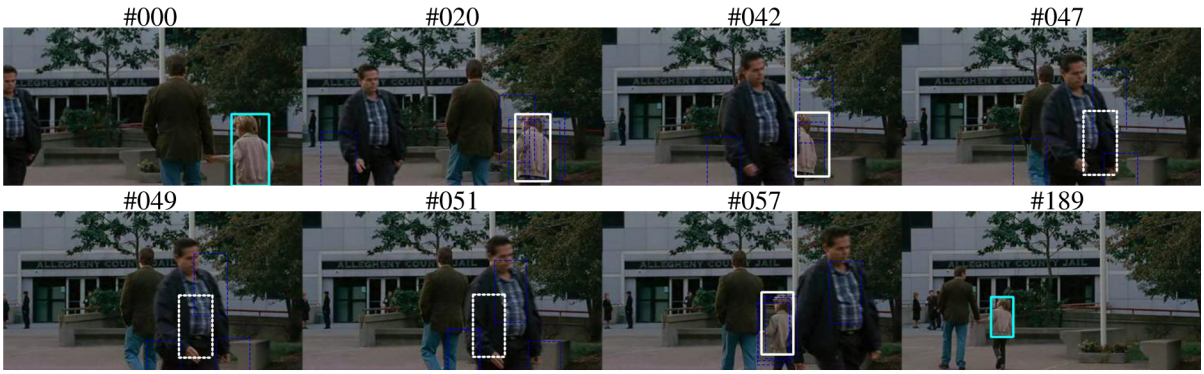


Obrázek 30: „Lyžař**“**: Sledování jízdy lyžaře, který se dostane do zákrytu vůči kameře.

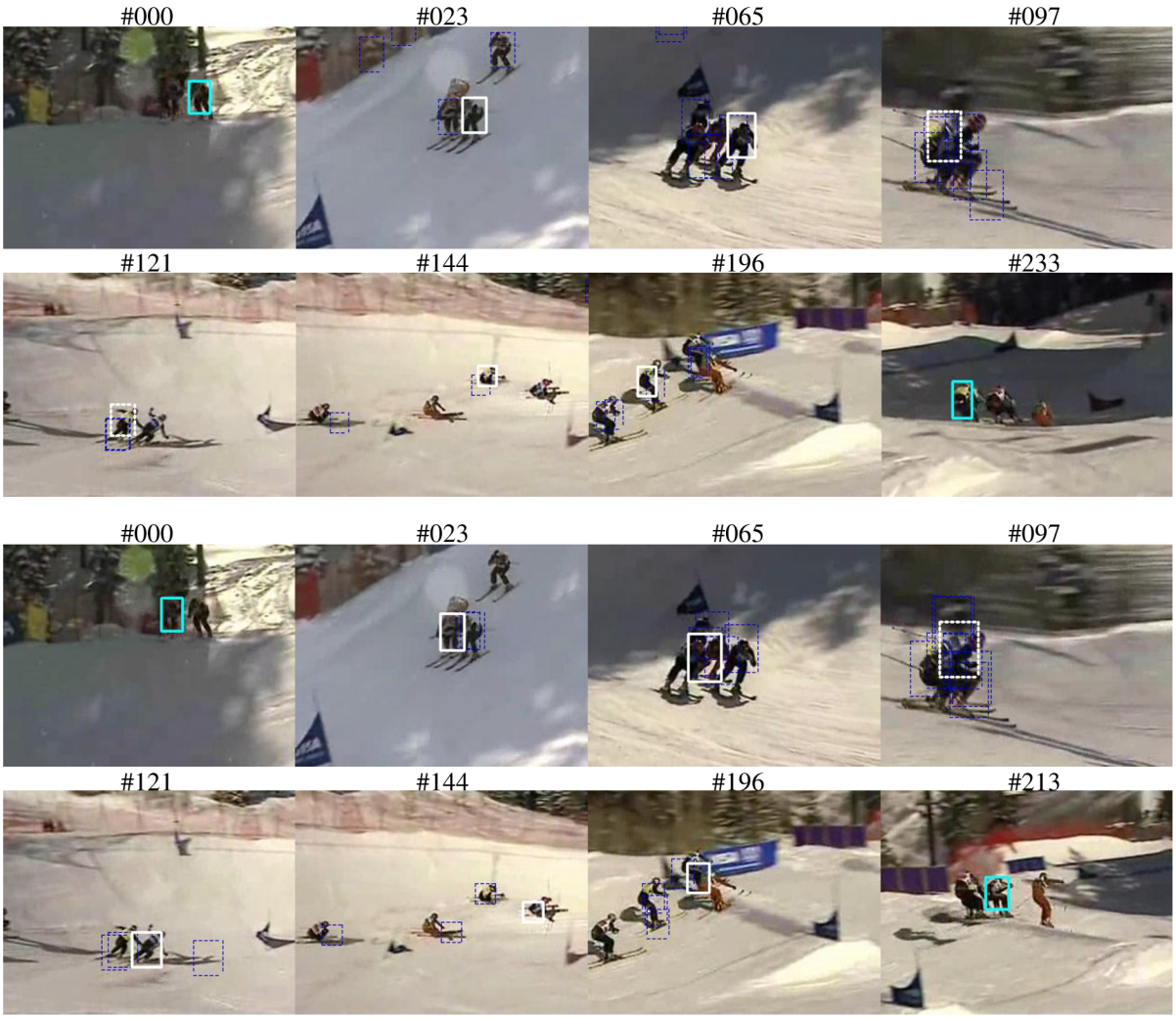
Oba zákryty na obrázku 30 trvaly pouze dva až tři snímky. Obrázek 31 ukazuje korektní zvládnutí patnácti snímkového zákrytu (od snímku #042 až po #057), kde sledovaný hoch není po celou dobu vůbec vidět. Ke sledování bylo potřeba použít pouze dva k-snímky a nastavit parametry DP.

Na následujícím obrázku 32 jsou ukázány snímky ze dvou nezávislých sledování lyžařů ve stejném úseku videa. K trénování detektoru jsou v obou případech použity čtyři klíčové snímky, které jsou v obou videích také umístěny ve stejných místech kromě posledního, protože v daném úseku

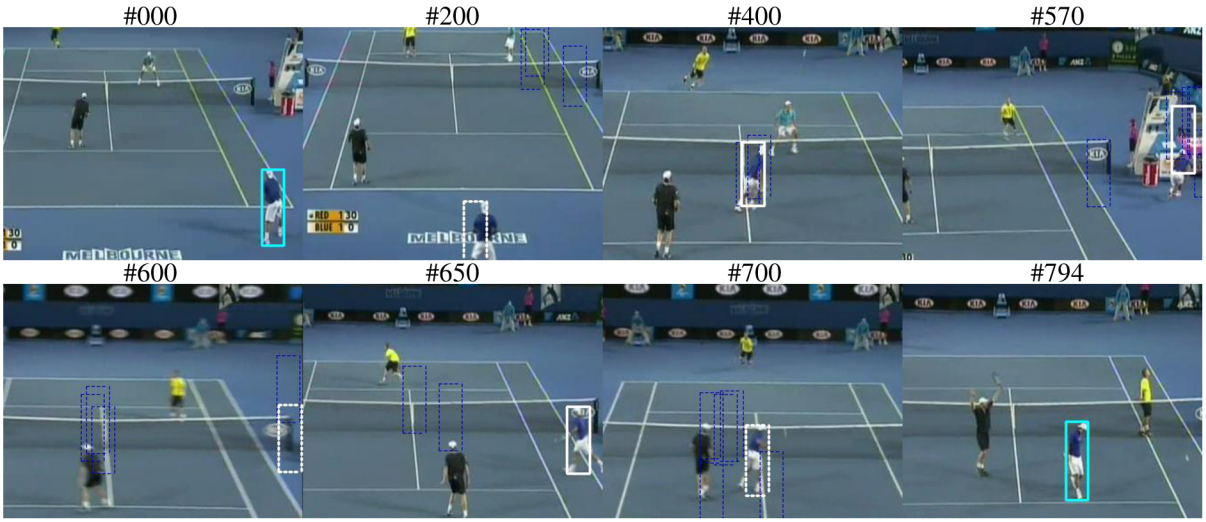
dochází k mnohým překrýváním a oba lyžaři nebyli vidět najednou. Sledování je obtížné, protože se všichni čtyři lyžaři na videu navzájem překrývají, mění se osvětlení scény a pozadí obsahuje vzorky, které jsou objektu podobné. Zbylé dva lyžaře nelze příliš dobře sledovat, protože v úseku okolo snímku #100 se oba dostanou mimo záběr kamery. Ve výsledku je sledování úspěšné v obou případech v celém úseku videa.



Obrázek 31: „Chlapec**“**: Sledování pohybu chlapce, který se v průběhu své chůze dostane do plného zákrytu vůči kameře.



Obrázek 32: „Lyžaři 1 a 2**“**: Nezávislé sledování dvou lyžařů ve stejném úseku videa. V prvních dvou řádcích jsou snímky ze sledování žlutého lyžaře (k-snímky: #000, #102, #135, #233). Ve spodních dvou řádcích jsou snímky ze sledování modrého lyžaře (k-snímky: #000, #102, #135, #213).



Obrázek 33: „Tenista 1**“**: Sledování tenisty, který se na chvíli dostane mimo záběr kamery.

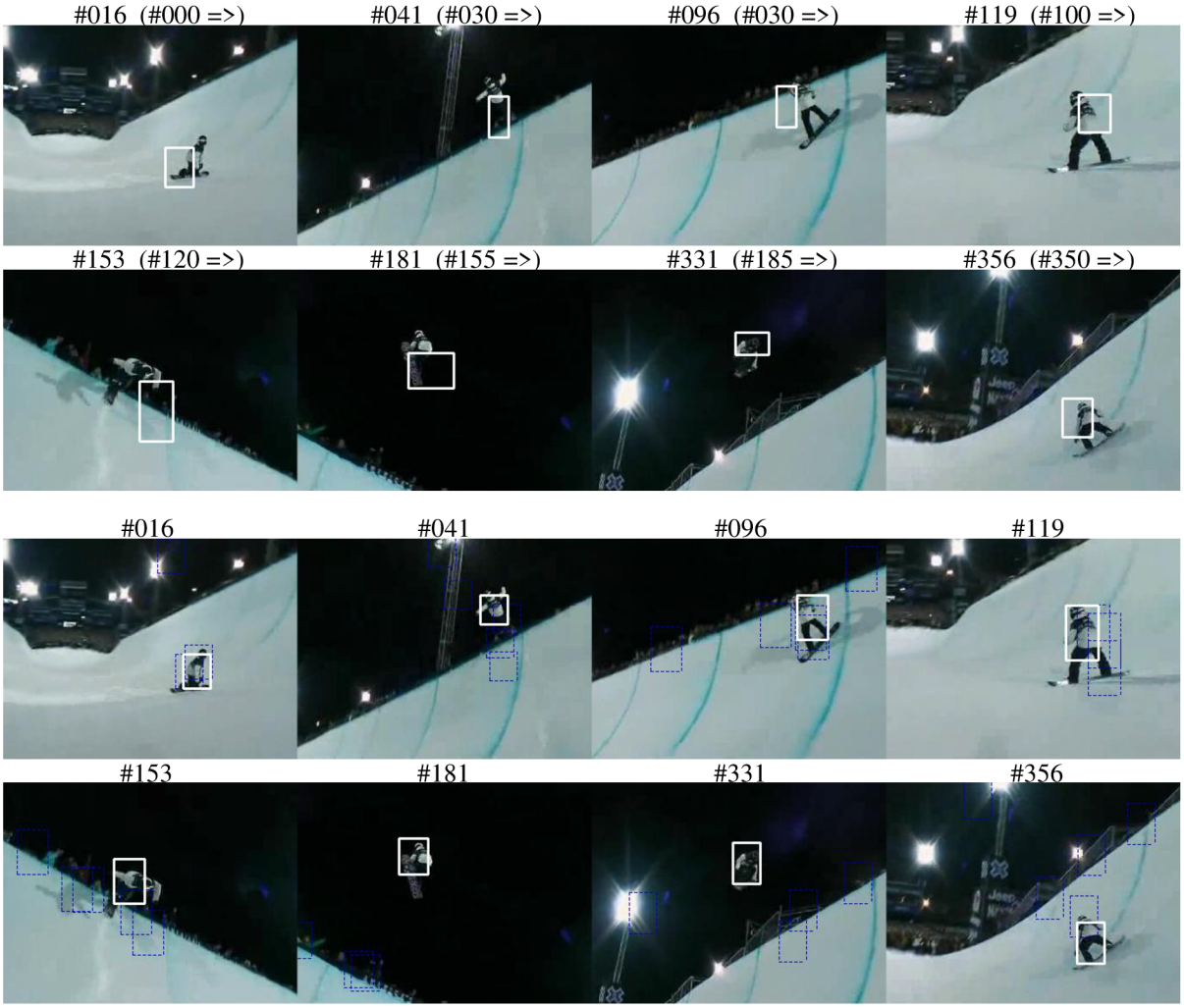
Obrázek 34: „Tenista 2**“**: Sledování tenisty při prudkých pohybech kamery.

Obrázek 33 a 34 zobrazují opět dvě nezávislá sledování objektu v jednom stejném videu. Hráči se nijak nepřekrývají, ovšem video obsahuje dva zajímavé okamžiky. Tenista se v obrázku 33 dostane na 58 snímků úplně mimo záběr kamery (přibližně mezi snímky #570 a #650). Algoritmus dynamického programování tedy v daném úseku pracuje s objektem, jako by byl v zákrytu, a díky přidání pevných konstant na pozice #584 a #642 nebylo nutné nastavovat penalizaci λ_o na velmi nízké hodnotu, a zbylá trajektorie nebyla DP negativně ovlivněna. Celé sledování je provedeno za pomocí čtyř k-snímků (#000, #101, #660, #794), protože pozadí scény obsahuje mnoho vzorků, které jsou barevně velmi podobné s objektem. Bylo tedy nutné natrénovat detektor i pomocí dvou dalších klíčových snímků, v nichž je tenista aktuálně v pohybu (trochu rozmazaný). I tak je přesnost detekce pouhých 17,9%, což nasvědčuje velké složitosti sledování daného objektu v tomto videu.

Na obrázku 34 zase nastává situace, kde dochází k rychlému pohybu kamery (tenista na předchozím obrázku je v daný moment mimo záběr), což způsobí nejen to, že se celá scéna poněkud rozmaže (viz snímky #597 a #615), ale mean shift algoritmus daný objekt ani nenajde, protože je jeho pozice mimo vyhledávací okno. Situace je vyřešena přidáním nového klíčového snímku na pozici #597, díky čemuž se detektor naučí detekovat i „rozmazaného“ tenistu. Sledování je ve výsledku provedeno pomocí tří k-snímků.

9.4.1 Porovnání implementované aplikace s online mean shift algoritmem

Na následujícím obrázku 35 jsou v několika snímcích ukázány dvě různá sledování snowboardisty na u-rampě. Ve vrchních snímcích je vyobrazeno sledování pomocí samotného online mean shift algoritmu pracujícího stylem *trial-and-error*, kde uživatel zastaví jeho běh, když je velká nepřesnost při sledování, vrátí se tedy o několik snímků zpět, určí pozici objektu, a opět spustí sledování od daného bodu. V závorkách u každého vyobrazeného snímku je napsána pozice jeho startovního snímku. Například ve snímku #153 je zobrazen výsledek po opětovném spuštění sledování ze snímku #120. K úspěšnému vytvoření trajektorie trvající celých 365 snímků videa bylo potřeba sedmkrát běh algoritmu přerušit, i přesto je výsledek pořád nepříliš přesný (hlavně kvůli tomu, že nelze zajistit plynulý přechod mezi poslední detekovanou a aktuální upravenou pozicí objektu).



Obrázek 35: „Snowboard**“**: V prvních dvou řádcích je zobrazeno sledování snowboardisty na u-rampě pomocí online mean shift algoritmu. Ve spodních dvou je totéž sledování pomocí implementovaného offline systému.

Níže na stejném obrázku jsou ty samé snímky videa, kde je nyní objekt sledován prostřednictvím implementovaného offline systému. K výpočtu bylo použito pět klíčových snímků na pozicích #000, #058, #120, #185 a #365. S použitím několik pevných konstant a pár iterací nastavení parametrů dynamického programování je výsledek daleko přesnější (viz tabulka 4). Práce se systémem je přehlednější, pohodlnější a dá se říct, že k výpočtu bylo potřeba také méně lidského úsilí a soustředění.

9.4.2 Vyhodnocení výsledků sledování

Následující tabulka ukazuje některé výsledky sledování objektů ve videích použitých v této diplomové práci. Řádky určují jednotlivá sledování, kde se název odvíjí od názvů napsaných v popiscích příslušných obrázků. Sloupce ukazují postupně počet snímků sledovaného úseku, kde „k“ značí počet klíčových snímků. Velikost vyhledávacího okna v pixelech, čas spotřebovaný na růst trajektorie, celkový čas výpočtu trajektorie. Dále následují hodnoty popsané v kapitole 7.3, které jsou získané pomocí vyhodnocovacího nástroje.

	snímků/k	okno (px)	růst(s)	čas(s)	F-measure	vzd. středů	detekce	úplnost	přesnost
Hokejista:	99 / 2	28 × 60	0,54	2,30	76,97%	13,31%	98,64%	100.0%	47,27%
Lyžař:	171 / 3	46 × 95	4,36	8,46	79,61%	9,12%	92,00%	89,47%	50,00%
Chlapec:	189 / 2	38 × 70	2,13	4,95	78,40%	9,12%	99,07%	100,0%	62,61%
Lyžaři 1:	233 / 4	26 × 38	1,32	5,36	67,45%	18,96%	96,54%	76,92%	18,18%
Lyžaři 2:	213 / 4	30 × 40	1,66	5,57	64,16%	24,49%	97,26%	66,67%	21,93%
Tenista 1:	794 / 4	27 × 75	9,49	24,09	61,61%	15,33%	94,55%	68,35%	17,85%
Tenista 2:	794 / 3	24 × 72	9,67	25,16	68,15%	10,99%	98,19%	95,35%	28,57%
Snowboard:	365 / 5	34 × 46	2,71	9,87	66,05%	20,39%	97,49%	87,18%	34,90%
Slečna:	665 / 3	74 × 115	36,3	70,90	74,69%	13,49%	91,90%	100,0%	78,67%
Motocykl:	265 / 3	57 × 44	3,18	7,92	67,57%	20,24%	98,74%	100,0%	46,59%
Průměr:					70,47%	15,54%	96,44%	78,39%	40,65%

Tabulka 4: Vyhodnocení sledování ukázkových testovacích videí zobrazených v textu.

Výsledky metriky *F-measure* se pohybují stabilně někde v rozmezí 60 až 80%, což značí velmi slušnou přesnost implementovaného algoritmu, i přesto že ve své práci nepoužívá adaptivní změnu měřítka vyhledávacího okna, čímž by se určitě výsledky této metriky ještě vylepšily (viz kapitola 10.2). Na druhou stranu procentuální vzdálenost středů nezávisí na velikosti vypočteného okna a dává výsledky někde kolem 9 až 25%. Procento odmítnutých regionů obrazu detektorem se pohybuje kolem 96,5%, což přesně odpovídá výsledkům popsaným ve článku [30], na jehož základě je systém implementován. Hodnoty úplnosti a přesnosti se také téměř neliší.

Musí se vzít také v úvahu, že všechny výsledky porovnávající vypočtenou trajektorii s referenční se odkazují na data zadaná ručně, která nemusí být zcela spolehlivá. Mnohdy by se dalo diskutovat, zda daný objekt leží přesně na zvolené pozici.

Obrázek 36: „Slečna**“**: Sledování slečny v dlouhém úseku videa pomocí tří k-snímků (#000, #129, #665) s použitím poměrně velkého vyhledávacího okna.

Na obrázku 36 je ukázka 665 snímkového videa, kde je největším problémem při sledování natočení hlavy. Jelikož první i poslední klíčový snímek zabírá slečnu zepředu, bylo třeba přidat třetí, kde jsou vidět pouze vlasy (např. #060). Jak je v tabulce 4 vidět, f-míra vytvořené trajektorie je takřka 75%, kdežto bez prostředního k-snímku klesne na pouhých 64%. Jelikož je v tomto videu vysoká přesnost sledování objektu, lze místo zadávání více pevných konstant využít implementované rozšíření dynamického programování. Výsledek je pořízen s využitím rozšíření a se čtyřmi zadanými pev-

nými konstantami (bez rozšíření bylo potřeba zadat konstant okolo deseti, aby se dosáhlo stejné úspěšnosti měření).

Obrázek 37 ukazuje příklad dalšího sledování, kdy se objekt dostává do zákrytu, tentokrát se jedná o motocyklistu, který jede ve skupině s ostatními. Sledování je úspěšné za pomocí tří klíčových snímků.

Obrázek 37: „Motocykl**“**: Sledování motocyklisty ve videu pomocí tří k-snímků (#000, #215, #265).

10 Možné pokračování práce

Pokračování diplomové práce by mohlo být zaměřeno na snížení potřebné výpočetní doby celého systému, zahrnutí sledování objektu za pomoci několika různých změn rozlišení vyhledávacího okna, na vylepšení uživatelského rozhraní, které by umožňovalo větší volnost při interaktivním nastavování parametrů, nebo na způsobu, jak prostřednictvím daného systému sledovat více objektů najednou.

10.1 Zrychlení výpočtu algoritmu

Jak již bylo řečeno, nejvíce zpomalujícím faktorem celého systému je výpočet struktury integrálního histogramu, který zabírá na testovacích videích průměrně 44% celkového času. Výpočet IH lze ve výsledku poměrně pěkně paralelizovat a to následujícím, mnou navrženým způsobem propagace[8], který obsahuje mnoho paralelních zpracování, a proto je vhodný například k výpočtu na procesoru grafické karty. Ve své práci jsem s tímto výpočtem experimentoval a pokusil se jej implementovat pomocí knihovny OpenCL, ale bohužel časová tíseň a nedostatek zkušeností hrály velkou roli a ve výsledku nebylo rozšíření implementováno.

Máme-li zadána vstupní data ve formě binární matice (černobílého obrazu), která vypadá například následovně,

0	0	1	0	0	1
1	0	0	1	1	0
0	0	0	0	0	0
1	0	1	0	1	0
0	0	0	0	1	0

je její propagace provedena pomocí tří kroků. V prvním kroku se paralelně pro každou hodnotu bodu matice určí její korespondující sloupec histogramu (neboli výsledek výrazu $Q(\cdot)$). Binární matici jsem zvolil jako příklad proto, že pokud hodnota 1 odpovídá bílé barvě a hodnota 0 černé, je korespondující navýšení sloupce bílé barvy o jedničku stejné, jako jsou hodnoty ve výše znázorněné vstupní matici (pro černý sloupec se pouze přehodí hodnoty 0 a 1).

Další krok propagace se provádí paralelně pro každý sloupec histogramu u všech řádků matice a poté paralelně pro každý sloupec histogramu u všech sloupců matice. Pro přehlednost je znázorněn výpočet pouze pro sloupec histogramu odpovídající bílé barvě.

0	0	1	0	0	1
1	0	0	1	1	0
0	0	0	0	0	0
1	0	1	0	1	0
0	0	0	0	1	0

⇒

0	0	1	1	1	2
1	1	1	2	3	3
0	0	0	0	0	0
1	1	2	2	3	3
0	0	0	0	1	1

⇒

0	0	1	1	1	2
1	1	2	3	4	5
1	1	2	3	4	5
2	2	4	5	7	8
2	2	4	5	8	9

[8] Tento způsob byl nejspíše již publikován, protože je celkem jednoduchý. Nikde jsem jej bohužel nenašel.

Druhý krok tedy spočívá v tom, že se matice projde postupně po řádcích a sčítá se hodnota aktuálního sloupce histogramu s hodnotou korespondujícího sloupce v buňce matice vlevo a výsledek se uloží jako nová hodnota aktuálního sloupce. Ve třetím kroku se provede to samé, ale tentokrát se postupuje po sloupcích matice.

Výsledek je, jak je vidět, správný, protože každá buňka v matici se nyní rovná součtu všech hodnot buněk, které původně ležely vlevo nahoře od ní, včetně hodnoty jí samotné. Výpočet histogramu pro region matice/obrazu se vypočítá úplně stejně jako při klasické propagaci za pomoci vlnoplochy.

Pro snímek o rozlišení 480×360 pixelů, kde každý histogram má 104 sloupců, by se výpočet mohl v každém kroku postupně zpracovat pomocí $480 \cdot 360 = 172800$, $360 \cdot 104 = 37440$ a $480 \cdot 104 = 49920$ paralelních procesů, což zcela vyhovuje principu práce GPU.

Zrychlení výpočtu integrálního histogramu by se dalo provést také zmenšením rozlišení daného snímku. Podle [30] robustnost detekce při nižších rozlišeních příliš razantně neklesne.

10.2 Změna velikosti vyhledávacího okna

Implementovaný systém sleduje objekt pomocí obdélníku, jehož velikost je získána interpolací velikostí obdélníků ze dvou sousedních k-snímků. Jako rozšíření práce by bylo dobré implementovat možnost, kdy by uživatel zapnul vyhledávání za pomoci adaptivní změny velikosti vyhledávacího okna.

Ve výsledku by to znamenalo, že se i-snímky navzorkují pod několika různými změnami rozlišení, což by nebyl zase moc velký problém. Obtížnější by bylo tuto změnu provézt u sledování kandidátního vzorku pomocí mean shift algoritmu, protože se při zahájení jeho práce předem počítají hodnoty profilů (viz kapitola 6.2.1), které jsou závislé právě na rozlišení vyhledávacího okna. Nejoptimálnější by asi bylo, globálně předem spočítat hodnoty jader pro všechna možná rozlišení (například pro původní rozlišení $\pm 10\%$) a mean shift algoritmu předávat ukazatel na ně. V každém kroku růstu by se musel také provést mean shift posun za pomocí každého jednoho nového rozlišení, kde by se jako nová pozice a velikost vybrala ta kombinace, která má nejmenší Bhattacharyho vzdálenost.

10.3 Zvýšení robustnosti detektoru

Detektor je založen pouze na barevných příznacích, což jej činí více náchylným na variace osvětlení scény a šum v obraze. Proto by bylo dobré zkusit zkombinovat jeho trénování s dalšími příznaky, jako jsou například Haarovy vlnky apod.

10.4 Vylepšení uživatelského rozhraní

Vylepšením uživatelského rozhraní není myšlena jeho grafická stránka (i když ta by také potřebovala vylepšit), ale spíše jeho funkčnost. Po testování nastavení parametrů dynamického programování jsem zjistil, že by bylo mnohdy dobré jedny hodnoty penalizací nastavit pro jeden úsek videa a jiné pro další. Proto bych jako uživatel jistě uvítal možnost nastavit ve videu pozici různých separátorů, které by rozdělovaly video na více úseků, v nichž by se mohly penalizace individuálně měnit.

Tato změna ovšem vyžaduje využití jiné knihovny pro vytváření GUI, která nabízí více možností.

10.5 Sledování více objektů

Sledování více objektů by bylo asi nejvíce ošemetné v principu implementace GUI, které by mělo být hlavně přehledné a názorně by mělo zobrazovat každou trajektorii, aby šla lehce upravit. Z hlediska výpočtu trajektorií by se musel pro každý objekt vytvořit nový *boosted color bin* příznak a z každého klíčového a i-snímku by se individuálně provedl růst jejich kandidátních vzorků, čímž by byla razantně ovlivněna výpočetní doba celého systému (doba výpočtu IH by se nezměnila).

Celkové úpravy by nemusely být příliš složité, protože je celá aplikace psána pomocí objektově orientovaného programování, kde je většina hlavních částí programu od sebe logicky oddělena a pracuje se s nimi pouze na základě příslušných objektů jejich tříd.

11 Závěr

Diplomová práce ve svém úvodu přehledně popisuje algoritmy, které se nejvíce používají pro sledování objektů ve videu, spolu s jejich principy práce, příklady použití a odkazy na články, v nichž jsou aplikovány. Detailněji jsou také diskutovány různé způsoby samotné reprezentace objektů ve videu, spolu s nejpoužívanějšími přístupy k jejich detekci.

Práce se převážně zaměřuje na teoretický popis a implementaci vybraného sledovacího algoritmu, kterým je interaktivní offline systém pro sledování obecných barevných objektů ve videu. Systém k detekci objektu využívá kombinaci barevných příznaků a pomocí časové koherence vytváří více jeho pravděpodobných trajektorií, na jejichž základě se pomocí uživatelem měněných parametrů vytváří optimální výsledná trajektorie objektu. Pomocí těchto nastavitelných parametrů se dopočítají také situace, kde objekt není zcela správně sledován nebo kde se dostal mimo záběr kamery.

Systém na videu o rozlišení 480×320 pixelů dosahuje rychlosti okolo 15 až 70 snímků za vteřinu v závislosti na velikosti sledovaného objektu. Aplikace dosahuje na doposud testovaných videích velmi dobrých výsledků, kde úspěšnost sledování objektu skrz celý úsek videa činí takřka 80% (bez dopočítaných pozic) a přesnost určení výsledné pozice objektu je v průměru více než 70%. Tyto údaje byly naměřeny za pomoci implementovaného nástroje na vyhodnocení výsledků sledování, který je v práci také detailně popsán.

Přínos práce spočívá především v analýze a ve vyhodnocení přesnosti vybraného sledovacího systému a ve způsobu jeho implementace, kde díky přidání různých konstant a podružných algoritmů se zvýšila robustnost a uživatelská přívětivost celé aplikace. Bezpochyby největším přínosem je navržené rozšíření, které usnadňuje nalezení optimální trajektorie objektu.

Další vývoj projektu by se mohl odebírat směrem zvýšení robustnosti detektoru či zvýšení přesnosti napasování vyhledávacího okna na tvar objektu ve snímku. Rychlost práce aplikace lze zvýšit například paralelizováním výpočtu integrálního histogramu. Upravení aplikace pro sledování více objektů najednou by také nemuselo obsahovat velké překážky.

Literatura

[1] Agarwala, A., Hertzmann, A., Salesin, D., Seitz, S.: *Keyframes-Based Tracking for Rotoscoping and Animation*. In SIGGRAPH, 2004

[2] Agbinya, J. I., Ress, D.: *Multi-Object Tracking in Video*. CSIRO Telecommunication and Industrial Physics, 1999

[3] Ali, A., Aggarwal, J.: *Segmentation and Recognition of Continuous Human Activity*. In IEEE Work-Shop on Detection and Recognition of Events in Video, pages 28-35, 2001

[4] Avidan, S.: *Support Vector Tracking*. In CVPR, 2001

[5] Buchanan, A., Fitzgibbon, A.: *Interactive Feature Tracking using K-D Trees and Dynamic Programming*. In CVPR, 2006

[6] Comaniciu, D., Meer, P.: *Mean Shift Analysis and Applications*. The Proceedings of the Seventh IEEE International Conference, 1999

[7] Comaniciu, D., Ramesh, V., Meer, P.: *Real-time tracking of non-rigid objects using mean shift*. In CVPR, 2000

[8] Comaniciu, D., Meer, P.: *Mean Shift: A Robust Approach toward Feature Space Analysis*. In IEEE Transaction on Pattern Analysis and Machine Intelligence, vol. 24, no. 5, May 2002

[9] Comaniciu, D., Ramesh, V., Meer, P.: *Kernel-Based Object Tracking*. IEEE Transaction on Pattern Analysis and Machine Intelligence, vol. 25, no. 5, May 2003

[10] Freund Y., Schapire R., *Experiments with a new boosting algorithm***.** In the Proceedings of the Thirteenth International Conference on Machine Learning, pages 148-156, 1996

[11] Fukunaga, K., Hostetler, L.: *The estimation of the gradient of a density function, with applications in pattern recognition*. IEEE Transaction on Information Theory, vol. 21, no. 1, pp. 32-40, January 1975

[12] Gonzalez, R., Woods, R.: *Digital Image Processing*. Second edition, Prentice Hall, ISBN 978-0201180756

[13] Cheng, Y.: *Mean Shift, Mode Seeking, and Clustering*. IEEE Transaction on Pattern and Machine Intelligence, vol. 17, No. 8, August 1995

[14] Islam, Z., Oh, Ch., Lee, Ch.: *Video Based Moving Object Tracking by Particle Filter*. Chonnam National University, Gwangju, 2009

[15] Li, F.: *AdaBoost Algorithm and its Application on Object Detection*. [cit. 2011-01-02]. Dostupné na URL <http://cs.gmu.edu/~fli/presentation/AdaBoost And Detection.ppt>

[16] Lipton, A., Fujiyoshi, H., Patil, R.: *Moving Target Classification and Tracking from Real-time Video*. Carnegie Mellon University, Pittsburgh, 1998

[17] Mikolajczyk, K., Schmid, C.: *A Performance Evaluation of Local Descriptors*. In CVPR, pp. 1615-1630, 2003

[18] Papageorgiou, C., Oren, M., Poggio, T.: *A General Framework for Object Detection*. In ICCV, Bombay, India, pp. 555-562, 1998

[19] Porikli, F.: Integral *Histogram: A Fast Way to Extract Histograms in Cartesian Spaces*. In CVPR, vol. 1, pp. 829-836, Volume 1, 2005

[20] Quast, C., Kaup, A.: *Shape Adaptive Mean Shift Object Tracking in Video Sequences*. University of Erlangen-Nuremberg, December 2009, [cit. 2011-01-05]. Dostupné na URL <http://itg32.hhi.de/docs/ITG32_UER_09_2_226.pdf>

[21] Ridder, C., Munkelt, O., Kirchner, H.: *Adaptive Background Estimation and Foreground Detection using Kalman-Filtering*. Proceedings of International Conference on recent Advances in Mechatronics (ICRAM), pp. 193-199, 1995

[22] Shafique, K., Shah, M.: *A Noniterative Greedy Algorithm for Multiframe Point Correspondence*. In ICCV, pp. 110–115, 2003

[23] Susaki, Y.: *The truth of the F-measure*. Poslední modifikace 26.10.2007 [cit. 2011-05-12]. Dostupné na URL <http://www.flowdx.com/F-measure-YS-26Oct07.pdf>

[24] Thirumuruganathan, S.: *Introduction To Mean Shift Algorithm*. Poslední modifikace 1.4.2010 [cit. 2011-04-11]. Dostupné na URL <http://saravananthirumuruganathan.wordpress.com/2010/04/01/introduction-to-mean-shift-algorithm/>

[25] Vidal, F. B., Alcalde, V. H.: *Object Visual Tracking using Window-Matching Techniques and Kalman Filtering*. ISBN 978-953-307-094-0, 2010

[26] Viola, P., Jones, M.: *Robust Real-time Face Detection.* In ICCV, vol. 2, pp. 747, Volume 2, 2001

[27] Wang, Y., Doherty, J., Dyck, R.: *Moving Object Tracking in Video*. Proceedings Conference on Information Sciences and Systems, Princeton, 2000

[28] Wang, F., Yang, J., He, X., Loza, A.: *Surveillance Video Object Tracking with Differential SSIM*. Institute of Image Processing and Pattern Recognition, Shanghai, 2010

[29] Watve, K.: *A Seminar on Object Tracking in Video Scenes*. Indian Institue of Technology, Kharagpur, 2004

[30] Wei, Y., Sun, J., Tang, X., Shum, H.: *Interactive Offline Tracking for Color Objects.* In ICCV, pp.1-8, 2007

[31] Yilmaz, A.: *Contour-Based Object Tracking with Occlusion Handling in Video Acquired Using Mobile Cameras*. In IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 26, no. 11, November 2004

[32] Yilmaz, A., Javed, O., Shah, M.: *Object Tracking: A Survey*. ACM Computing Surveys 38, 4, Article 13., 2006

[33] *Support Vector Machine (SVM) - Algoritmy podpůrných vektorů.* Strojové učení. Faculty of Information Technology, Masaryk University Brno. [cit. 2011-04-22]. Dostupné na URL <http://is.muni.cz/el/1433/podzim2006/PA034/09_SVM.pdf>

Seznam příloh

Příloha A: Ovládání aplikace

Sledovací aplikace:

Klávesnice:

`Up, 8`	- posun obdélníku objektu o pixel nahoru.
`Down, 2`	- posun obdélníku objektu o pixel dolů.
`Left, 4`	- posun obdélníku objektu o pixel doleva.
`Right, 6`	- posun obdélníku objektu o pixel doprava.
`Home, 7`	- zvětšení obdélníku objektu v ose x.
`End, 1`	- zmenšení obdélníku objektu v ose x.
`PgUp, 9`	- zvětšení obdélníku objektu v ose y.
`PgDn, 3`	- zmenšení obdélníku objektu v ose y.
`+`	- zvětšení obdélníku objektu v ose z.
`-`	- zmenšení obdélníku objektu v ose z.
`i`	- vložení k-snímku.
`e`	- vymazání k-snímku.
`c`	- spuštění výpočtu trajektorie.
`p`	- uložení trajektorie do výstupního video souboru.
`s`	- uložení pozic k-snímků, parametrů dynamického programování a pozic pevných konstant.
`b`	- spuštění samotného AdaBoost algoritmu.
`l`	- posunutí se o jeden snímek po směru videa.
`k`	- posunutí se o jeden snímek v protisměru videa.
`j`	- posunutí se o deset snímků po směru videa.
`h`	- posunutí se o deset snímků v protisměru videa.
`m`	- posunutí se na následující k-snímek nebo pevnou konstantu po směru videa.
`n`	- posunutí se na následující k-snímek nebo pevnou konstantu v protisměru videa.
`f`	- přepnutí způsobu zobrazení finální trajektorie.
`g`	- přepnutí způsobu zobrazení kandidátních vzorků.
`TAB`	- přepnutí se do vedlejšího okna (jeho aktivace).
`x`	- zobrazení okna s histogramy.
`v`	- přepnutí se do vyhodnocovacího módu.
`ESC, q`	- ukončení aplikace.

Myš:

stisk levého tlačítka + posun + puštění	- označení pozice obdélníku objektu
pravé tlačítko	- označení okna jako aktivního
pravé tlačítko ve středu vzorku	- označení vzorku jako pevné konstanty

Vyhodnocovací aplikace:

Klávesnice:

`Up, 8`	- posun obdélníku referenčního vzorku o pixel nahoru.
`Down, 2`	- posun obdélníku referenčního vzorku o pixel dolů.
`Left, 4`	- posun obdélníku referenčního vzorku o pixel doleva.
`Right, 6`	- posun obdélníku referenčního vzorku o pixel doprava.
`Home, 7`	- zvětšení referenčního vzorku v ose x.
`End, 1`	- zmenšení referenčního vzorku v ose x.
`PgUp, 9`	- zvětšení referenčního vzorku v ose y.
`PgDn, 3`	- zmenšení referenčního vzorku v ose y.
`+`	- zvětšení referenčního vzorku v ose z.
`-`	- zmenšení referenčního vzorku v ose z.
`i`	- vložení referenčního vzorku.
`e`	- vymazání referenčního vzorku.
`c`	- spuštění vyhodnocení.
`s`	- uložení pozic referenčních vzorků.
`l`	- posunutí se o jeden snímek po směru videa.
`k`	- posunutí se o jeden snímek v protisměru videa.
`j`	- posunutí se o deset snímků po směru videa.
`h`	- posunutí se o deset snímků v protisměru videa.
`m`	- posunutí se na následující k-snímek nebo referenční vzorek po směru videa.
`n`	- posunutí se na následující k-snímek nebo referenční vzorek v protisměru videa.
`v`	- přepnutí se do sledovacího módu.
`ESC, q`	- ukončení aplikace.

Myš:

stisk levého tlačítka + posun + puštění	- označení pozice referenčního vzorku

Příloha B: Formát vstupních souborů

Hodnoty se načítají ze souborů, kde jsou od sebe odděleny bílými znaky. Formát souboru s klíčovými snímky je následující:

```
(číslo_snímku
x1 y1
x2 y2)*
#dp
lambda_penalizace
lambda_o_penalizace
vzdálenost
míra
#hc
(číslo_snímku
x1 y1
x2 y2)*
```

Kde hodnoty `#dp` a `#hc` jsou pouze identifikační a rozdělují oblast pro uložení parametrů dynamického programování a pozic pevných konstant. Hvězdička značí, že obsah závorky může být opakován nula až *n*-krát. Hodnoty x_n a y_n označují pozice, kde 1 je levý horní a 2 je pravý dolní roh ukládaného obdélníku.

Formát vstupního souboru s referenčními snímky pro vyhodnocení úspěšnosti sledování je následující:

```
(číslo_snímku
x1 y1
x2 y2)*
```

Příloha C: Adresářová struktura DVD

- Aplikace
  - Dokumentace
  - Binární soubory
  - Zdrojové soubory
  - instalace.txt
- Písemná zpráva
- Videa
  - Vzorová videa
  - Ukázková videa
- plakát.pdf