VŠB – Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra informatiky

DIPLOMOVÁ PRÁCE

Sledování pohybů objektů v sekvencích získaných kamerou

Object cracking in videosequences captured by a camera

2009 Bc. Roman Musialek

Prohlášení:

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Datum: …… Podpis:……

Rád bych na tomto místě poděkoval panu doc. Dr. Ing. Eduardu Sojkovi za vstřícnost a pomoc při vedení diplomové práce.

Abstrakt:

Tato diplomová práce popisuje vývoj systému na detekování pohybu. Systém na vstupu pracuje se snímky, které mu budou dodány z kamery. Na těchto snímcích je postupně provedena detekce rohů, korespondence mezi body a shlukování. Výsledkem je pak určení pohybu objektů ve snímcích. V práci je proveden i teoretický základ ke všem použitým problematikám. Systém byl otestován na různých vstupních datech. V závěru je pak zhodnocení celého systému a jeho výhod a nevýhod.

Klíčová slova: Optický tok, detekce rohů, shlukování, korespondence bodů, sledování objektů.

Abstract:

This diploma thesis describes the development of a system for motion detection. The system works with images, that are obtained from a camera and used as an input data. Corner detection, correspondence between points, and clustering are subsequently performed on these images. As a result, motion of the objects in the images is determined. Theoretical background of all the considered issues is also presented in this work. The system was tested on various input data. The conclusion contains the assessment of the whole system and its advantages and drawbacks.

Keywords: optical flow, corner detection, clustering, object cracking, correspondence points

Obsah:

Seznam obrázků

Seznam tabulek

1 Úvod

Diplomová práce se zabývá detekci pohybu objektů ve snímcích. Sledování pohybu můžeme využít v mnoha činnostech. Pokud se podíváme na snímky, kde budeme mít automobily, nabízí se možnost měření rychlosti a směru automobilu. Toto se dnes využívá skoro v každé obci, kde se umísťují radary. Tyto radary dokážou s pomoci kamery určovat rychlost projíždějících automobilů. Toto využití má působit hlavně bezpečnostně na řidiče. Pohyb můžeme určovat i na snímcích, kde budeme sledovat pohyb osob. Toto sledování se uplatňuje u zabezpečovacích systémů různých objektů. Určování pohybu umožňuje automatizaci těchto systému, bez nutnosti sledování objektů člověkem. v diplomové práci popisuji systém pro sledování pohybu automobilů. Téma práce jsem si vybral v návaznosti na předměty počítačové grafiky. Tyto předměty mě při studiu zaujaly, a proto jsem si chtěl vyzkoušet určitý problém vyřešit.

Práce je rozčleněná do několika částí. Na začátku je stručně uvedena úvaha nad celou problematikou s jednoduchým nastíněním řešení problému. Čtenář by měl získat určitý přehled o problematice a způsobu řešení. V další části je přehledně popsaná teoretická část. Uvádím zde podrobněji rozebrané algoritmy, které slouží k řešení. Určité algoritmy jsem pak využil i ve své aplikaci. Čtenář by měl pochopit princip každého z uvedených algoritmů. Ve třetí části uvádím výsledky, ke kterým jsem došel při testování své aplikace. Čtenář zde najde i popis vytvořených programátorských tříd, které jsem využil ve své aplikaci. Na závěr jsem zhodnotil celou mou práci a výsledky, ke kterým jsem dospěl.

2 Způsob řešení problémů

Při zpracování obrazu se můžeme setkat s úlohami, které mají dynamický charakter. V takovém případě pak pracujeme nikoli s jedním, ale hned s celou sekvenci obrazů pořízených určitým zařízením, nejčastěji kamerou. Škála úloh, které přicházejí v úvahu je velmi rozmanitá. Úkolem v této práci je ve scéně detekovat pohybující se objekty a určení směru jejich pohybu. Rozšíření můžeme najít ve snaze o rozpoznání, jaké objekty máme ve scéně. Jindy naopak může být scéna statická, ale pohybuje se kamera, která scénu sleduje.

Půjde tedy o sekvenci snímků, které budou dodávané nějakým snímacím zařízením. Toto zařízení může být jakákoli kamera, která dokáže snímat určité objekty. Kamera může být umístěna v různých úhlech pohledu a snímat scénu různě. Bude určitě nutné, aby snímky byly v nějakém stanoveném formátu a v určitých časových intervalech. Tyto snímky bude potřeba zpracovat a určit pohybující se objekty.

2.1 Vstupní data

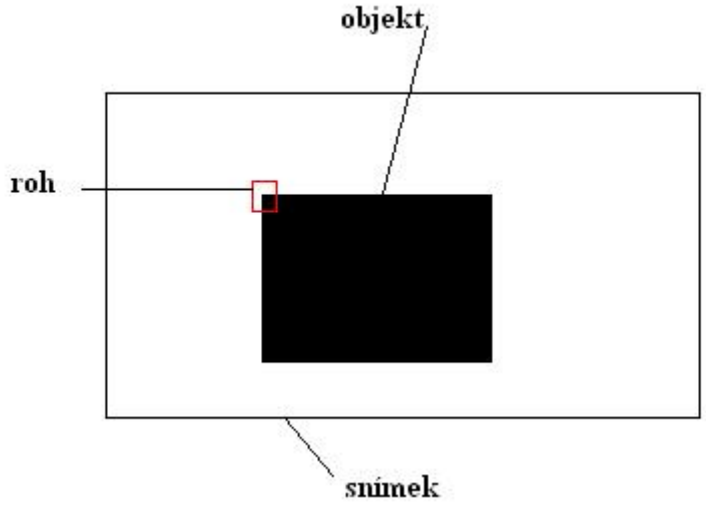
V první fázi jsem si vytvořil představu, jak můžou být potřebná data dodávaná. Existovaly dva formáty, které mohly do mého systému vstupovat. Prvním z nich by mohla být přímo videosekvence, která by byla dodávaná z kamery. Taková to sekvence by byla dodávaná velmi rychle a systém by musel v reálném čase snímky vyhodnocovat. Systém bude potřebovat určitý čas na zpracování a vyhodnocení. Videa dodávají příliš mnoho snímků za jednotku času. Pro můj případ určitě postačí situace, kdy bude do systému dodán jeden snímek za nějaký krátký interval. Vycházím z předpokladu, že nepotřebuji sledovat detailně pohyb, ale postačí mi zjištění pozice objektu v určitém čase. Pro tento případ se naskytovala možnost, aby se do systému dávaly pouze obrázky, které bude kamera snímat v časových intervalech.

Kamera může být umístěna ve vyšších úrovních, než jsou automobily. Budou tedy dodávaný snímky s pohledem na střechu automobilu. Jinou podobu budou mít snímky, když bude kamera umístěna vedle projíždějících automobilů. Automobily budou snímaný z boku. Taktéž je nutné si uvědomit počty automobilů, které se mohou v obrazech vyskytovat. V jednodušším případě bude kamera zabírat jenom jeden pruh a na snímcích by se vyskytoval malý počet automobilů. Nejjednodušší situace nastane v okamžiku, kdy se na obrazech bude vyskytovat pouze jedno auto. Počet automobilů není závislý jenom na počtu jízdních pruhů, ale závisí i na velikosti záběru kamery. Pokud kamera bude zabírat jen velmi blízké vzdálenosti, může často zobrazovat pouze jeden automobil.

Po tomto rozboru bylo zcela jasné, že je potřeba využít metodu výpočtu pomoci obrazových snímku a začít detekci pohybu s jedním objektem, který je základem celého systému.

2.2 Detekce rohů

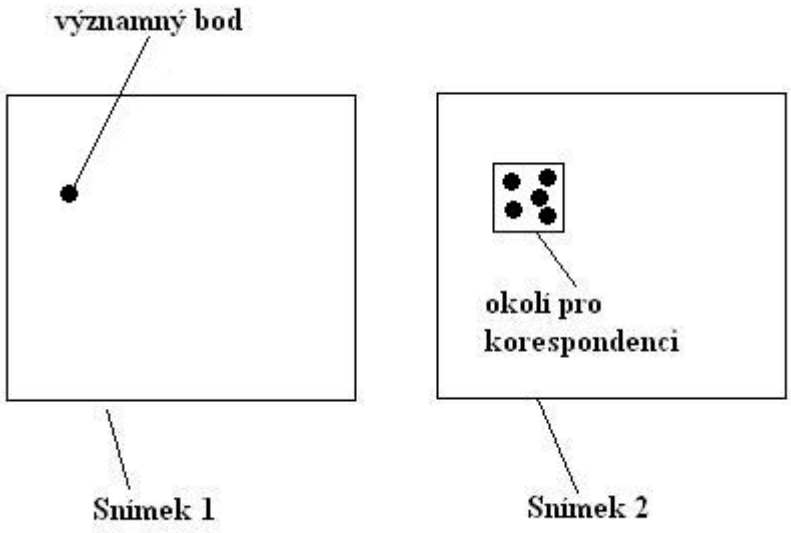
V prvním kroku rozboru problému jsem musel zvážit, jak budu vyhodnocovat objekty na snímku. Bylo tedy nutné rozhodnout se a správně určit významné body na snímku. Problematika umožňuje různé způsoby řešení. V dostupné literatuře je uváděno řešení pomoci jasových oblastí. Toto řešení bylo založené na vytvoření oblastí, které mají stejnou jasovou hodnotu, a dá se předpokládat, že patří jednomu objektu. Tyto oblasti by se hledaly v každém obrázku. Toto řešení je velmi jednoduché ale pro mou práci by se nedalo využít. V mém případě může být v jednom snímku hned několik oblastí, které mohou mít stejnou jasovou složku. Jasová složka navíc může být v různých snímcích odlišná. Odlišnost může být způsobena okolními vlivy. Další možnost se naskytovala v sledování určitých bodů. Otázkou tedy zůstává, jak takovéto body získat v obraze, aby byly jednoznačné. Nabídla se možnost detekce rohových bodů. Tyto body lze získat podle několika algoritmů. Detekce rohů vychází ze skutečných objektů a jejich tvarů. Na následujícím obrázku je znázorněn jeden objekt a ukázka rohu.



Obrázek 1: Ukázka rohu v obraze

Tato zjištění vyhovovala mé problematice, a tedy základním stavebním kamenem mého systému byla detekce rohů v obraze. Existuje několik algoritmů, které řeší tuto problematiku. Já jsem si pro svou práci vybral Harrisův detektor. Jeho popis i s dalšími algoritmy najdete v další části mé práce.

V dalším kroku bylo potřeba tyto body spojovat v jednotlivých obrazech. Bylo nutné nacházet korespondenci bodů. Tato korespondence se děje mezi dvěma po sobě jdoucími snímky. Vlastnost pohybu objektů se nazývá optický tok. Pro jeho výpočet existuje několik metod. Pro mou práci jsem si vybral metodu založenou na detekci změn jasů v obraze. Autory této metody jsou Bruce D. Lucas a Takeo Kanade. Na následujícím obrázku je znázorněn princip tohoto algoritmu.



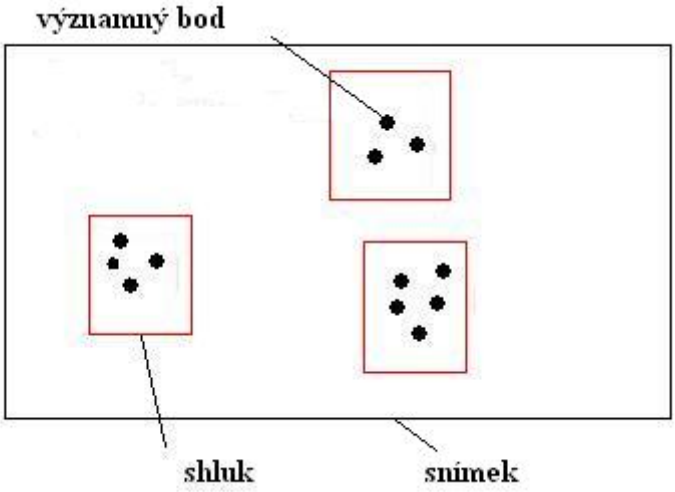
Obrázek 2: Znázornění korespondence bodů

Na prvním snímku je zobrazen detekovaný významný bod. V druhém snímku hledáme jeho korespondující bod, který by se měl nacházet v okolí korespondence. V tomto okolí se může vyskytovat více bodů a v takovém případě musím nalézt ten, který nejvíce odpovídá významnému bodů ze snímku1. V případě, že v okolí je jenom jeden bod, tak korespondence je daná přímo. Pokud v okolí není žádný bod, tak se korespondenci nepovedlo nalézt. Vycházel jsem z předpokladu, že pohyb na dvou po sobě jdoucích snímcích bude malý a tedy korespondující body budou v blízkých pozicích na obou snímcích. Pokud bych tento předpoklad nevyužil a pohyb na snímcích by byl velký, program by měl problémy s nalezením korespondence bodů. Musel bych využít jiný postup. Podrobný teoretický popis najdete v kapitole 3.2, kde bude metoda popsána.

2.3 Detekce shlukování

Posledním krokem jsem uspořádal body do celků, abych mohl určovat pohyb jednotlivých oblasti. Tyto celky pak můžeme sledovat a určovat jejich pohyb v obraze. Pro spojování bodů do oblasti se naskýtá možnost shlukování. Shlukování je spojování určitých částí obrazu k sobě podle určitého pravidla.

V mém případě se naskýtá využít možnost spojování těch bodů, které jsou velmi blízko u sebe. Využil jsem tedy metodu podobnou hierarchickému aglomerativnímu shlukování. Na následujícím obrázku je ukázka tří shluků, které jsou zobrazeny červeným rámečkem. Každý shluk obsahuje významné body, které jsou blízko sebe. Jejich počet může být odlišný v každém z vytvořených shluků.



Obrázek 3: Ukázka shluků

Takto vytvořené shluky mají ovšem jednu podstatnou nevýhodu. Shluky jsou závislé na pořadí vkládaných bodů. Toto omezení ovlivňuje konečný výsledek. Takto vytvořím shluky, ve kterých budou body v malých skupinkách. Nabízí se možnost dalšího spojování podle stejného pravidla. Experimentálně je nutné vyzkoušet počet iterací, které by bylo možné využít ke spojování. Hrozí totiž spojení shluků, které se ve skutečnosti vůbec spojit nemají. Toto může nastat v případě, že ve snímku bude více objektů a tyto objekty budou blízko sebe.

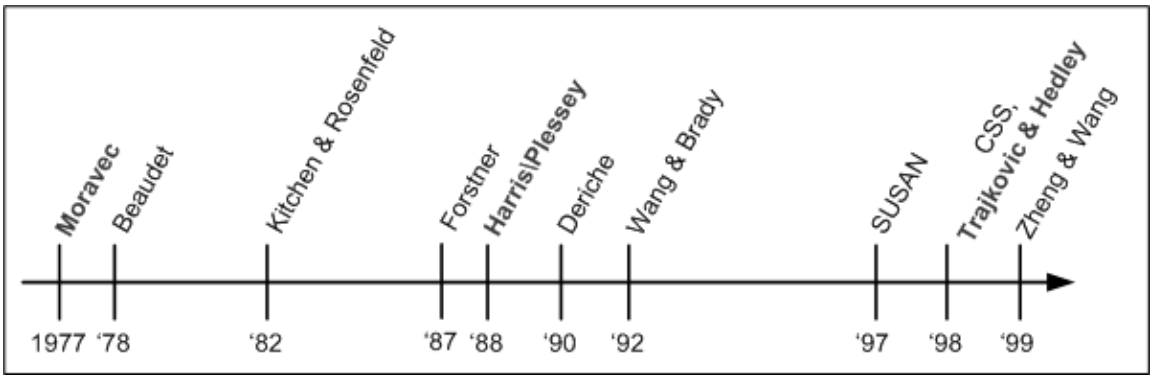
Využitím vícenásobného procházení shluků eliminuji počáteční závislost na pořadí bodů. Ve svém programu jsem tedy využil základní dělení pomoci jednoho cyklu. Program bude detekovat všechny shluky a to i takové, které se ve scéně pohybovat nemusí a jejichž sledování je v našem případě zbytečné. Půjde o detekci statických objektů. V experimentálním zkoušení jsem pak vyzkoušel dvojitý cyklus spojení. Teoretický popis použité metody i s dalšími najdete v sekci 3.3.

3 Teoretická část

Diplomová práce je rozdělená na tři základní úlohy. Je to detekce významných bodů ve scéně, realizace korespondence bodů v jednotlivých snímcích a poslední část je tvorba shluků a jejich sledování. Pro každou tuto část existuje několik způsobu řešení. V každé z těchto úloh jsem si vyzkoušel různé možnosti řešení. Pro detekci významných bodů jsem vyzkoušel detektory rohů. Pro tuto detekci se nabízely algoritmy od Moravce nebo Harrise. Vyzkoušel jsem oba a pro svou aplikaci jsem vybral Harrisův algoritmus. Pro korespondenci se nabízel algoritmus Lucass-Kanade. Vyzkoušel jsem shlukování hierarchické a grafové. Ve své práci jsem použil hierarchické. Všechny tyto algoritmy najdete podrobněji popsány v následující části mé práce.

3.1 Detekce rohů

V současné době je známo více metod, které se zabývají danou problematikou. Metody pracují na odlišných principech. Chronologický vývoj algoritmů je na níže zmíněném obrázku.



Obrázek 4: Historický vývoj algoritmů detekce významných bodů

Principy detekování významných bodů můžeme rozdělit

- **Hranové metody** – Pracují na principu sledování zakřivení hran, které určujeme z významných bodů.
- **Autokoleračních metody** - Pracují na principu výpočtu významných bodů na základě posouvání obrázků.
- **Ostatní metody** - Využití kruhové okolí, zkoumání a porovnávání jasových hodnot na tomto okolí.

3.1.1 Harrisův algoritmus

Harrisův algoritmus slouží pro detekování rohů obrazů. Pracuje na principu posunu zkoumaného snímku v různých směrech a výpočtu gradientu v jednotlivých bodech snímku. Základem metody je výpočet rozdílů čtverců jasů ve všech bodech snímku. Výsledky jsou správné pouze tehdy, jestliže porovnávané obrázky mají stejný jas a kontrast. Citlivost na změny jasu a kontrastu je důležité omezení. Principem algoritmu je výpočet autokorelační matice pro každý bod snímku. Jedná se o matici prvních parciálních derivací. Matice má následující tvar[3]

$$A = \begin{bmatrix} \langle I^2{}_x \rangle & \langle I_x I_y \rangle \\ \langle I_x I_y \rangle & \langle I^2{}_y \rangle \end{bmatrix}.$$

Z této matice se vypočítají vlastní čísla, na jejichž základě se rozhodneme, zda daný bod je pro nás významný. Výpočet provádíme pro každý bod snímku dle následujícího vzorce

$$\lambda_{1,2} = \frac{\langle I^2{}_x \rangle + \langle I^2{}_y \rangle \pm \sqrt{\left(\langle I^2{}_x \rangle - \langle I^2{}_y \rangle\right)^2 + 4 * \langle I_x I_y \rangle * \langle I_x I_y \rangle}}{2}.$$

(1)

Bod bude pro nás významný, pokud vypočtená vlastní čísla splňují následující podmínky. Čísla λ_1, λ_2 musí být zhruba stejně velká a zároveň mnohem větší než nula. Pokud nebude splněna výše uvedená podmínka, bod prohlásím za nevýznamný. V takovém případě se bude jednat o bod určité roviny, který není ve svém okolí odlišný. Může se také jednat o bod, který tvoří hranu objektu a není na začátku nebo konci této hrany. Výpočet vlastních hodnot je časově náročný z důvodu používání odmocnin ve vzorci. Proto autoři tohoto algoritmu použili místo výpočtu vlastních čísel níže uvedený vzorec, který dokáže úspěšně rozhodnout, jestli se v daném bodě nachází roh. Vzorec (2) zahrnuje rozdíl determinantu matice prvních derivací a součtu prvků na hlavní diagonále umocněných na druhou. Součásti vzorce je parametr, který určuje optimalizaci pro detekování významných bodů[3]

$$cor(x,y) = \det(M(x,y)) - k * trace^2(M(x,y)).$$

(2)

Hodnota k byla určena experimentálně a nejlepší výsledky jsou dosaženy pro hodnoty v rozmezí $k = 0.04 \ldots 0.15$. Bod (x,y) považujeme za roh, pokud jsou splněny následující dvě podmínky. Hodnota $cor(x,y)$ musí být větší než předem stanovena prahová hodnota. Druhou podmínkou je, aby hodnota $cor(x,y)$ byla největší v okně o předem definovaných rozměrech. Střed tohoto okna je umístěn právě v bodě o souřadnicích (x,y). V případě, že není splněná některá z uvedených podmínek, bod nepovažujeme na roh a dále ho nezpracováváme.

Nevýhodou této metody je vysoký počet chybných bodů, které jsou určeny nesprávně. Metoda má problémy se šumem v obraze, což se dá odstranit použitím vhodného filtru (Gaussova). Dalším nedostatkem metody je spolehlivá detekce rohů pouze v L-uzlech. V T-uzlech tato metoda velmi selhává. Metoda má i další modifikace, o kterých se krátce zmíním v následující části.

Objevily se i modifikace Harrisova algoritmu, které využívaly jednodušší vlastnosti. Snahou bylo ulehčení výpočtu a dosažení kvalitnějších výsledků, než u základního algoritmu. První takovou modifikaci je metoda Shi Thomasi, kde se využívá k rozhodování pouze menší z vlastních čísel[2]. Pro výpočet se využívá následující vzorec

$$\lambda = \frac{\left\langle I^2{}_x \right\rangle + \left\langle I^2{}_y \right\rangle - \sqrt{\left(\left\langle I^2{}_x \right\rangle - \left\langle I^2{}_y \right\rangle\right)^2 + 4 * \left\langle I_x I_y \right\rangle * \left\langle I_x I_y \right\rangle}}{2}.$$

(3)

Pokud je vlastní číslo větší než prahová hodnota, pak se jedná o významný bod. Výsledky této metody ukazují nejmenší možnou chybu ze všech možných modifikací. V porovnání s klasickým algoritmem tato úprava dosahuje horších výsledků. Ukazuje se, že využití jen jednoho vlastního čísla není dostatečné.

Další možnou úpravou je Förstnerův operátor. V této metodě se využívá lokálního maxima funkce. Operátor pracuje na téměř stejném principu, pouze k porovnání používá odlišné měřítko. Musíme tedy i v tomto případě stanovit předem prahovou hodnotu, s kterou budeme měřítko porovnávat[2]

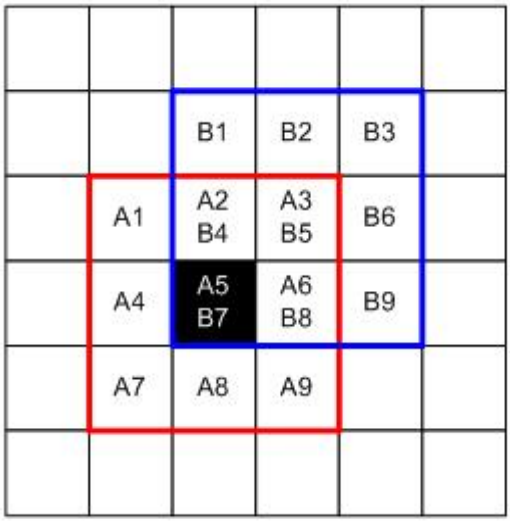
$$M = \frac{\lambda_1 * \lambda_2}{\lambda_1 + \lambda_2} = \frac{Det(A)}{Trace(A)}.$$

(4)

I v případě použití této metody dojdeme k horším výsledkům než u klasického algoritmu. Základní algoritmus tedy dosahuje nejlepších možných výsledků.

3.1.2 Moravcův operátor

Tento algoritmus byl vyvinut v roce 1977 v souvislosti s navigací robota. Moravcův algoritmus detekuje rohy pomoci okolních bodů a jejich podobnosti. Tato detekce je určena kolísáním intenzity jasu. Nad každý detekovaný bod se umístí malý čtverec, s kterým se pohybuje v každém z osmi směrů vždy o jeden pixel. Čtverce bývají velikosti $3x3$, nebo $5x5$ případně $7x7$ pixelů [2]. V následujícím obrázku vidíte čtverce velikosti $3x3$. Červený čtverec zobrazuje základní polohu. Modrý čtverec zobrazuje posunutí ve směru $(1,1)$. Černě je zobrazen bod, pro který se provede patřičný výpočet čtverců. Těchto posunutí je celkem 8.



Obrázek 5: Moravcův operátor

V každém posouvaném směru se počítá suma rozdílů čtverců (základního a posunutého). Z tohoto rozdílů čtverců je určena podobnost. Významný bod se detekuje za předpokladu, že zkoumaný bod není podobný svému okolí. Jinými slovy jeho jas se mění významně ve všech směrech. Na vstupu je obrázek v šedé stupnici. Pro každý pixel (x, y) v obrázku se vypočítá jasová odlišnost při posunutí (u, v) jako

$$V_{u,v}(x, y) = \sum_{\forall a,b} \left(I(x+u+a, y+v+b) - I(x+a, y+b)\right)^2 .$$

(5)

Uvažovaná posunutí (u, v) jsou: $(1,0)$, $(1,1)$, $(0,1)$, $(-1,0)$, $(-1,1)$, $(-1,-1)$, $(0,-1)$, $(1,-1)$. Pro každý pixel (x, y) spočítáme měřítko rohovosti $C(x, y)$. Podle velikosti posuvného čtverce musíme vynechat okrajové částí obrázku, kde by se tento čtverec posouval mimo snímek. Tyto okrajové části hned na začátku prohlásíme za nevýznamné body. Toto měřítko pak dále určuje informaci, jestli se jedná o významný bod či nikoli. Měřítko rohovosti se určí pomoci této rovnice

$$C(x, y) = \min\left(V_{u,v}(x, y)\right).$$

(6)

Porovnáním výsledných hodnot s prahovou hodnotou se určí všechna $C(x, y)$, která jsou menší než T a tyto se změní na nulu. V následujícím kroku vybereme z rohové mapy všechny lokální maxima. Všechny body, které mají nenulovou hodnotu, prohlásíme za rohové body.

Při této detekci je opět velmi těžké určit prahovou hodnotu T. Potřebujeme, aby se detekovaly všechny významné body, které se mají detekovat. Pokud zvolíme nevhodnou prahovou hodnotu, pak algoritmus detekuje špatné body. Tento algoritmus totiž detekuje body, které jsou na

diagonálách objektu a nejsou rohové. Tato detekce je důsledkem kolísaní intenzity při posouvání čtverců svislým a diagonálním směrem. Při posouvání vodorovným směrem pak už ke změně intenzity nedochází a v takovém případě by se tento bod neměl detekovat. Moravcův operátor ale na tyto hrany reaguje a detekuje na nich významné body.

3.2 Korespondence bodů

Druhým naším problémem, když už budeme mít v obraze získané body, bude určení jejích obrazů v následujících snímcích. Budeme se tedy snažit získat tyto údaje:

- Najít odpovídající bod v následujícím snímku
- Určit případnou rychlost pohybu
- Určit směr pohybu

Nejdříve se pár slovy zmíním o optickém toku, který je základním stavebním kamenem při určování pohybu těles v obraze.

Určení optického toku je užitečné při rozpoznávání vzorů a obecně ve zpracování obrazů. Fyzikální pohyb objektů ve snímcích není vždy reprezentován viditelnými změnami jasu. Typicky je pohyb reprezentován pomocí vektorů. Optický tok se používá pro popis rychlostního pole snímků, nebo pro určení pohybu jednotlivých bloků v obrazech. Toto můžeme vidět u videokomprese MPEG[2].

Pro správné výpočty je zapotřebí určitých podmínek. Předpokládá se, že povrchy objektu jsou hladké a ploché. Další důležitou součástí je osvětlení, které by nemělo tvořit stíny na jiných objektech. V neposlední řadě je kladen důraz, aby se objekty v obraze nepřekrývaly. Dodržení všech těchto podmínek vede ke správným výpočtům[2].

Metody optického toku mají za úkol vypočtení pohybu mezi dvěma snímky obrazu, které jsou pořízený v určité časové posloupnosti. První snímek je pořízen v čase t a následující v čase $t + \delta t$. Metody v sobě používají parciální derivace a berou v úvahu časovou složku souřadnic. Pro případ mé práce stačí dvojrozměrný prostor a posun v čase t [2]. Z tohoto důvodu zde více tuto problematiku nebudu rozebírat, jelikož výpočet optického toku není použit v ukázkovém programu.

3.2.1 Lucas-Kanade algoritmus

Tato metoda je založena na detekci změn jasu v obraze. Základní algoritmus se snaží nalézt a správně umístit okolí bodu z prvního obrazu do aktuálního obrazu. Cílem algoritmu je minimalizovat sumu čtverců chyb mezi dvěma obrazy. Z prvního obrazu máme detekovaný významný bod a jeho jasová složka má hodnotu $I(x, y)$. Tento bod chceme umístit v druhém obraze. V tomto obraze postupně procházíme všechny oblasti, kde byly detekované významné body $J(x_1, y_1)$ a mohli

bychom daný vzor zde hledat. Vycházíme z předpokladu, že souřadnice bodu z prvního obrazu a bodu korespondujícího v aktuálním obraze budou posunuty o jednotky pixelů. Proto obraz původního bodu hledáme jenom v nejbližším okolí[5].

Každý takový bod v tomto okolí uvážíme jako možný korespondující bod. Kolem každého takového bodu $J(x_1, y_1)$ vytvoříme okno v našem případě velikosti $3x3$. Stejné okno vytvoříme i u prvního bodu. Body $I(x, y)$ a $J(x_1, y_1)$ jsou umístěny ve středu těchto oken. Z takto definovaných bodů můžeme vypočítat jasový rozdíl. Metoda spolehlivě funguje jen pro hledání korespondence v blízkosti původního obrazu. Tedy korespondující body hledáme v jednotkách pixelů od původního. Metoda by byla po stránce výpočtu velmi náročná, pokud bychom neomezili prohledávanou oblast. Pro výpočet je definován vzorec na určení součtu kvadratických rozdílů jasů. Tento vzorec vypadá následovně

$$Jas = \sum_{a=-1}^{1}\sum_{b=-1}^{1}\left(I(x-a, y-b) - J(x_1-a, y_1-b)\right)^2 .$$

(7)

Uvedený součet určí, který z bodů okolí je korespondujícím bodem. Proto musíme výpočet provést pro všechny body, které jsou v blízkém okolí bodu $I(x, y)$. Korespondující body budou takové, které mají daný součet nejmenší. V ideálním případě je tento součet roven nule.

Pro případ, že bychom toto omezení neudělali, můžeme částečně problém vyřešit úpravou této metody v tzv. pyramidální Lucas-Kanade. Algoritmus spočívá v tom, že se postupně snižuje rozlišení dvou obrazů a počítáme pohyb ve více rozlišeních. Nechť I^0 je původní obraz v největším rozlišení, pak $I^1, I^2, I^3 \ldots$ jsou pyramidálně reprezentované obrazy. Tyto nové obrazy vznikají rekurzivně. Z původního obrazu I vznikne obraz I^1. Z obrazu I^1 vzniká obraz I^2 a tento mechanismus pokračuje až do stanoveného stupně pyramidy. Nechť $L = 1,2,3\ldots$ je úroveň pyramidy a I^{L-1} je obraz na úrovni $L-1$. Obraz I^{L-1} je pak definován takto

$$I^L(x,y) = \frac{1}{4}I^{L-1}(2x,2y) + \frac{1}{8}\left(I^{L-1}(2x-1,2y) + I^{L-1}(2x+1,2y) + I^{L-1}(2x,2y-1) + I^{L-1}(2x,2y+1)\right)$$
$$+ \frac{1}{16}\left(I^{L-1}(2x-1,2y-1) + I^{L-1}(2x+1,2y+1) + I^{L-1}(2x+1,2y-1) + I^{L-1}(2x-1,2y+1)\right) .$$

(8)

V praxi se používají úrovně pyramidy 2 až 4. Většinou nemá smysl jít pod čtvrtou úroveň. Hlavním smyslem této reprezentace je možnost zachytit velké pohyby pixelů. Velký pohyb můžeme počítat v menším rozlišení a naopak pomalý pohyb můžeme počítat ve velkém rozlišení a pokryjeme tím všechny možné směry pohybu. Nevýhodou je nutnost opakovaných výpočtů pro dva obrazy.

Algoritmus řešící výše popsanou problematiku musí mít dvě klíčové vlastnosti. První vlastnosti je přesnost. Jedná se o správné detekování pohybu v obrazech a to za pomocí malých integračních oken, ve kterých budeme daný pohyb vyhledávat. Druhou vlastností je robustnost. Jedná se o schopnost algoritmu reagovat na různé změny osvětlení, nebo zachycení větších pohybů. Pro zachycení větších pohybů by bylo nejjednodušší zvětšit integrační okno. Ne vždy je tato možnost použitelná, a proto existuje určitá závislost mezi přesností a robustností daných algoritmů.

3.3 Shlukování bodů

Shlukování bylo studováno v různých oborech aplikace. Bylo objeveno několik algoritmů, které můžeme najít v literatuře. Shlukovací algoritmy rozdělují datový soubor na množiny x shluků. Shlukovat můžeme objekty, které musejí být popsány pomoci stejné množiny znaků. My budeme porovnávat body (dvojrozměrný prostor), tedy naším hlavním kritériem bude vzdálenost, která se používá nejčastěji[8].

Základní algoritmus shlukování se skládá z těchto kroků:

1. Na vstupu je množina bodů. Z každé se vytvoří samostatný shluk.

2. Postupně procházíme shluky a dáváme k sobě ty dva, které jsou dostatečně blízko.

3. Předcházející krok opakujeme do té doby, dokud se dají spojovat shluky. V případě že nám zůstanou shluky, které už se nedají spojit je naše práce u konce. Algoritmus nám tedy vrátí takto vytvořené shluky.

Základní algoritmus je jen nástřelem. Má však mnoho upravených podob. Úpravy jsou především v jeho optimalizaci při zjišťování vzdálenosti bodů. Jinými slovy řečeno, aby se každé dva body porovnávaly maximálně jednou. Program prochází body shluku. Pro každý takový bod se snaží najít jiný shluk, abychom mohli přes takovýto bod spojit tyto dva shluky v jeden. Dva shluky jde spojit, pokud jsou alespoň dva jejich body dostatečně blízko. Pokud takový bod v jiném shluku nalezneme, jsou všechny body z prvního shluku přiřazeny do shluku druhého.

Shlukovací metody můžeme rozdělit do několika skupin. Uvádím zde alespoň některé z možných rozdělení. Základní metody budou podrobněji vysvětleny.

- **Hierarchické metody**
  - Aglomerativní algoritmy
  - Grafové algoritmy

- **Rozdělovací metody**
  - Pravděpodobnostně shlukovací
  - Shlukování pomocí k-středů

Na shlukovací algoritmy jsou kladeny určité nároky, aby se daly tyto algoritmy využívat. Každý shlukovací algoritmus nemusí řešit všechny níže uvedené požadavky, ale jen ty, které ve zvolených okolnostech mají smysl[8].

Shlukovací algoritmy musí řešit následující věci:

- Rozšiřitelnost – musí být použitelný pro velké množství dat
- Odlehlost – při zpracování musí řešit otázku odlehlých bodů
- Časová složitost – každý algoritmus musí být v použitelné složitosti
- Setřídění dat – algoritmus využije setřídění k urychlení své práce
- Atributy – algoritmus musí poznat, které atributy se použijí
- Tvarovost – algoritmus musí poznat netypické tvary shluků

3.3.1 Hierarchické shlukovací algoritmy

Tyto algoritmy spojují body do jednoduchých shluků. Takto vytvořené shluky se postupně spojují ve větší a tento postup se stále opakuje a tvoří hierarchický postup. Ve výsledku je reprezentace shluku pomoci stromu. Jednotlivé body tvoří listy tohoto stromu a uzly jsou jednotlivé shluky, které byly nalezeny. Takto vytvořený strom umožňuje pozdější procházení a prohlížení výsledků na různých úrovních stromové struktury[2]. Tyto algoritmy se dělí na dvě skupiny, které byly rozděleny již dříve.

- **Aglomerativní** – Tyto metody začínají se shluky, které obsahují pouze jeden bod, a postupně rekurzivně spojují dva a více nejpodobnějších shluků.

- **Dělící** – Tyto metody začínají s jedním shlukem, který obsahuje všechny body. Rekurzivně rozděluje tyto shluky na podshluky. Tento proces pokračuje, dokud není dosaženo kritéria pro zastavení. Tímto kritériem může být nedělitelnost již na další shluky, nebo dosažení potřebného počtu shluků.

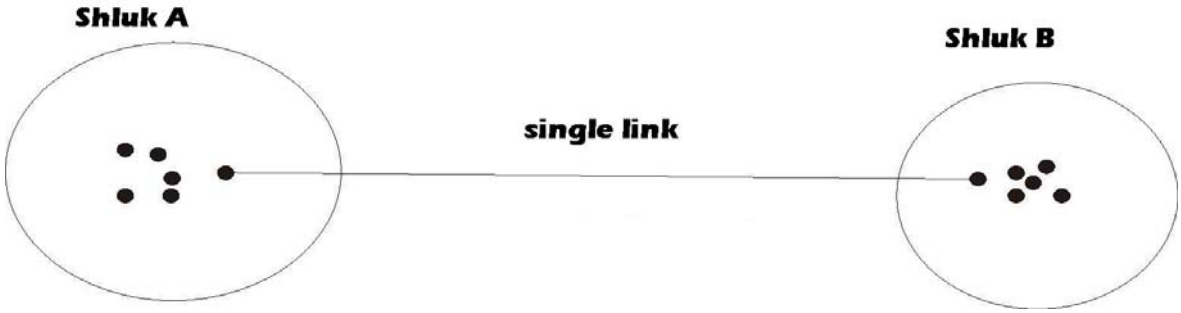
Výhody hierarchických slučovacích algoritmu jsou:

- Realizace pro jakékoli atributy
- Schopnost sledování různých úrovni hierarchického stromu
- Zjištění jakékoli podrobnosti a vzdálenosti

Nevýhody:

- Těžko předem určíme kritérium, aby došlo k zastavení běhu algoritmu
- Neposuzování nově vytvořených shluku.

Pro spojení, nebo rozdělení podmnožin bodů je velmi často vzdálenost mezi jednotlivými body přepočtena na vzdálenost mezi shluky. Použitý typ míry má velký dopad na tvorbu shluku. Míra přesně odráží vzdálenost mezi shluky. Základní používané míry jsou
Single- link, average-link, complete-link. Jednu z nich můžete vidět na následujícím obrázku.

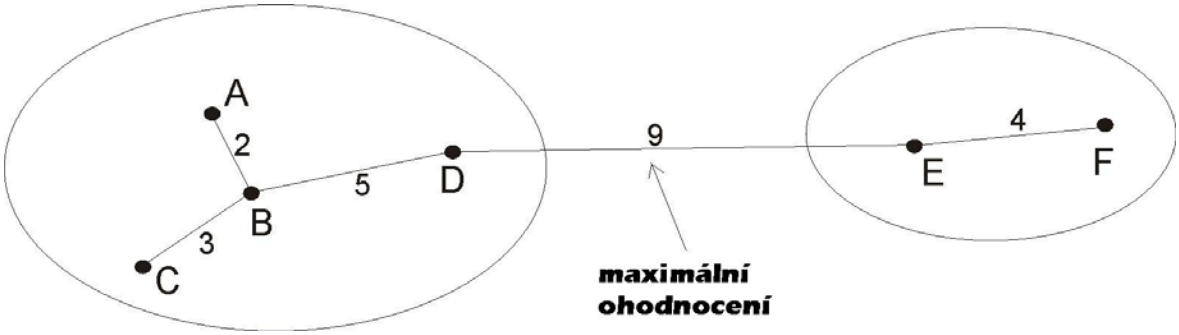


Obrázek 6: Vzdálenost mezi shluky

Na obrázku 5 vidíme dva vytvořené shluky. Mezi těmito shluky byla určená nejkratší vzdálenost dvou bodů. Toto je rozhodovací pravidlo, které určuje dělení shluků.

3.3.2 Grafové shlukovací algoritmy

Nejznámější grafové dělicí algoritmy jsou založené na konstrukci minimální kostry grafu. Kostrou grafu nazýváme libovolný podgraf daného grafu, který obsahuje všechny uzly grafu a tolik hran z původního grafu tak, aby z každého uzlů podgrafu existovala cesta ke všem zbývajícím uzlům. Je jasné, že k jednomu grafu lze nalézt i více koster, pokud existuje dostatečné množství hran a původní graf již například není sám kostrou[7].



Obrázek 7: Kostra grafu.

Z obrázku je patrné, že největší ohodnocení má hrana mezi body DE a proto bude odejmuta v tomto kroku. Vzniknou nám dva oddělené shluky. Bude tedy záležet na nastavených počátečních parametrech, jestli budeme dále prohledávat shluky, nebo již toto dělení bude dostatečné.

3.3.3 Shlukování pomocí k-středu

K-střed je iterativní technika, která začíná s počátečními shluky a během následujících iterací vylepšuje kvalitu nově uspořádávajících se shluku. Tento proces je zastaven, až je dosaženo ukončujícího kritéria (shluky se již nemění ani počtem ani kvalitou). Shlukování rozdělí vstupní vzory do n (uživatelských definovaných) podmnožin. Každý shluk je reprezentován pomocí vzoru neboli jeho středu[6]. Každá iterace průchodu mezi shluky se skládá z následujících kroků:

- **Krok 1** – Změna zařazení ve shluku vylepšuje celkovou kvalitu. Vzorek je přiřazen shluku, pokud je střed shluku danému vzorku nejblíže. Nutnosti je výpočet vzdálenosti mezi každým prototypem a vzorem. Pro výpočet používáme vzdálenosti Euklidovskou metrikou. Tento krok je velmi časově náročný.

- **Krok 2** – Nalezením nového prototypu, tedy nalezení nového středu pro každý shluk.

- **Krok 3** – Vyhodnocení ukončovacího kritéria, tedy zda už je dosaženo původně stanoveného počtu shluků. Často se používá standardní kvadratická chyba.

Počet iterací se může lišit podle složitosti problému od několika málo iterací až k mnoha stovkám i tisícům. Další problém je předem určení hodnoty n, která závisí na okruhu působnosti. Většinou je uživatel nucen zkoušet možnosti, kolik iterací musí udělat, aby byl získán patřičný výsledek. Tyto algoritmy jsou většinou citlivé na počáteční rozdělení shluku. Uživatel tedy je nucen zkoušet nastavení počátečních hodnot. Přímá implementace k-středů může být velmi náročná.

Postupy, které jsou popsané v literatuře, se snaží určitým způsobem snížit nároky na výpočet. Prototypy byly dříve organizovány pomoci vhodné struktury, aby nalezení nejbližšího prototypu pro daný vzor bylo efektivní. Tento postup spolehlivě funguje v okamžiku, kdy prototypy jsou stále a neměnné. V reálné situaci dost často tento předpoklad není možné splnit. Další technika používá informace o členství ve shluku z předchozí iterace a tím redukuje počet výpočtu vzdálenosti. Po několika prvních iteracích dochází k velkému úbytku změn členství ve shluku. Další vlastnost je velmi malá změna pohybu středu v dalších iteracích. Tyto optimalizace jsou mnohem využitelnější.[6]
Pro určení vzdálenosti bodů se používají různé možnosti. Jednou z těchto možností je Euklidovská míra. Tato míra je nejpoužívanější pro určení vzdálenosti mezi body, které jsou spojité. Euklidovská vzdálenost je daná předpisem

$$d_2(x_i, x_j) = \sqrt{\left(\sum_{k=1}^{d} (x_{i,k} - x_{j,k})^2 \right)}.$$

Euklidovská míra je speciální případ Minkowského míry

$$d_p(x_i, x_j) = \sqrt[p]{\left(\sum_{k=1}^{d} \left| x_{i,k} - x_{j,k} \right|^p \right)}.$$

Euklidovská míra je intuitivní a používá se pro ohodnocení vzdálenosti bodů ve dvou nebo trojrozměrném prostoru. Dobré využití má v situacích, kdy množina dat tvoří kompaktní nebo izolované shluky. Nevýhoda je dominance znaků, které by měly největší měřítko. Toto se dá řešit zavedením normalizace neboli zavedením vah. V našem případě ovšem nic takového není nutné.

3.4 Souhrn aplikace

Ve své aplikaci jsem využil algoritmy, které byly výše popsány. Pro detekci rohů jsem využil Harrisův detektor rohů. Tento detektor v mé aplikaci detekuje významné body, které jsou pro práci důležité. Když jsem v obrazech detekoval významné body, bylo potřeba najít korespondenci mezi body v obrazech. Pro korespondenci jsem využil algoritmus Luss-Kanadého. Využil jsem tedy umísťování okna o rozměrech $3x3$ pixelů z původního obrazu. Původní bod je uprostřed tohoto okna. Bod se snažím umístit v aktuálním obraze. Pro umístění vybírám body, které jsou v aktuálním obraze velmi blízké souřadnicím původního bodu. Pro všechny body, které vyhovují blízkosti, vytvářím opět okno o rozměrech $3x3$ pixelů.

Vypočítám rozdíl jasů podle vzorce (7). Bod s nejmenším rozdílem jasů prohlásím za korespondující bod s původním. V případě, že v definovaném okolí nenaleznu žádné body pro korespondenci, původní bod nemá korespondující obraz. Z takto vytvořených korespondujících bodů jsem vytvořil shluky. Shluky jsou v aplikaci tvořený pomoci blízkosti bodů. Procházím jednotlivé body a snažím se je umístit do toho shluku, ve kterém je nějaký blízký bod. Pokud nenajdu žádný bod ve shlucích, vytvořím nový shluk a bod vložím do něj.

Toto jsou tři základní kroky, které moje aplikace provádí na každém snímku. Z vytvořených shluků pak pomoci těžiště určuji pohyb objektů ve snímcích. Pro každý shluk je určen směr pohybu a toto je výsledek, který program vytvoří pro každý dodaný snímek. Tento výsledek je požadavkem diplomové práce.

4 Zhodnocení

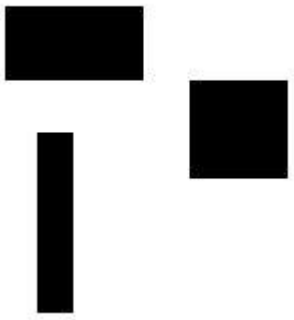
Součástí práce bylo porovnání zvolené metody včetně podrobného popisu výsledku. Prošel jsem tedy celý program a testoval jsem vždy jednotlivé části systému podle toho, jaký měly cíl. Takto jsem zhodnotil detekování rohových bodů, korespondující body, shlukování a na závěr celý program.

4.1 Získání bodů

Detekce bodů byla realizovaná pomoci Harris Stephens algoritmu, jehož popis byl proveden v dřívější části této práce. Pro zhodnocení této metody jsem využil porovnání se softwarovým nástrojem OPEN CV, ve kterém je naimplementováno mnoho metod, které se právě při práci s obrazem využívají. Pro demonstraci jsem vybral několik svých vytvořených obrázků, u kterých jsem zkoumal počet detekovaných bodů a samozřejmě i zda jsou tyto body shodné. Porovnal jsem i časové předpoklady, tedy jak dlouho se patřičný výpočet prováděl. Porovnání a výsledky najdete níže v tabulkách.

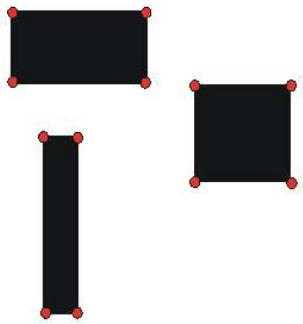
4.1.1 Detekování jednoduchých obrázků

První je jednoduchý obrázek, na kterém jsou nakresleny čtverce a obdélníky. Jedná se o klasicky rohové objekty, u kterých by neměl být problém nalézt rohové body.



Obrázek 8: Základní geometrické obrazy

Z následujícího obrázku je patrné, že program nalezl korektně všechny rohy.



Obrázek 9: Základní geometrické obrazy s detekovanými rohy

Další z možných tvarů je kruh a trojúhelník. Jelikož to už nejsou klasické rohové útvary, tak už zde by mohly nastat komplikace. Objekty již nemají zcela plynulé hrany. Hrany tvoří při větším přiblížení zuby, které detekuje program. Výsledkem jsou detekované body, které lidským okem nejsou brány jako roh, ale při větším zvětšení obrázku je vidět zubovité čary, které vždy v ostrém přechodu tvoří roh.

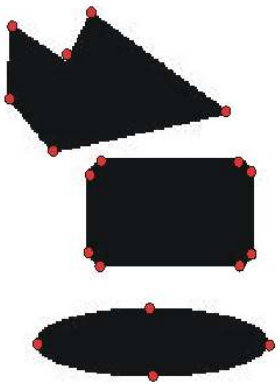
Pokud tedy zvolíme jako porovnávací hodnotu malou hodnotu pro $cor(x, y)$, budeme detekovat tyto zuby jako naše rohové body. Tento jev je patrný hlavně u kruhu, kdy celý obvod je tvořen těmito zuby. Zuby se tvoří i na trojúhelníku. Tento jev se dá odstranit pokusným zjištěním, jakých hodnot nabývá $cor(x, y)$ v daných zubech. Pokud zvolíme pro $cor(x, y)$ hodnoty dostatečně velké, tak eliminujeme detekci těchto zubů na minimum.

Na následujícím obrázku jsou geometrické útvary, které již nemají strany v úhlech 90°, a proto je potřeba zjistit jak se program vyrovná s takovýmito tvary. Jedná se vlastně o spojení vlastností z předchozích objektů.

Obrázek 10: Geometrické útvary

Program detekoval body v ostrých přechodech jednotlivých obrazců. Detekoval ale body i na roztřepených hranách jednotlivých tvarů. Tyto body se daly eliminovat již dříve zmíněným způ-

sobem. Byly detekovány body na hranách těchto obrazců, které jsem zde nezvýrazňoval. Tato detekce je způsobená zubovitým obrysem.



Obrázek 11: Detekování bodů u geometrických útvarů

Prozkoumal jsem tedy detekci rohových bodů na základních geometrických tvarech, které by se měly objevovat při použití samotného programu. Úvodní testování proběhlo na zcela uměle nakreslených snímcích. Tyto snímky měly dobře nakreslené rohy bez větších šumů. Program bez problému našel rohové body v těchto obrazech. V případě kruhových útvaru nacházel i rohy na obvodu. Příčinou tohoto jevu bylo zubovité kreslení kruhových části, kdy program nacházel rohy i na těchto zubovitých částech. Odstranění tohoto jevu se dá provést pomocí vhodně zvolené porovnávací hodnoty proměnné $cor(x, y)$. Nástroj OPEN CV má toto ošetřeno a snaží se pouze detekovat rohové body.

Pro zhodnocení jsem vytvořil jednoduchou scénu, Na ni postupně hodnotím všechny tři části mého programu. Tato scéna je jednoduchý obrázek autíčka.

Obrázek 12: Ukázkový obrázek autíčka

Od tohoto základního obrazu byly vytvořeny další čtyři, které se odlišují umístěním autíčka. Takto vytvořená sekvence byla vložena do programu. Při testování byla postupně měněná prahová hodnota, která určovala, kdy prohlašujeme konkrétní pixel za významný bod. Nejdříve si ukážeme počty detekovaných bodu při různých prahových hodnotách. Na obrázku je zobrazení detekce autíčka při prahové hodnotě 1000.

Obrázek 13: Významné body na obrázku

Tabulka 1: Počty detekovaných bodů:

Název	**Prahová hodnota**				
	Cor= 1000	Cor= 5000	Cor=8000	Cor=10000	Cor=25000
Auto1.jpg	197	118	91	86	65
Auto2.jpg	189	124	104	94	71
Auto3.jpg	252	161	136	127	96
Auto4.jpg	236	166	145	138	107
Auto5.jpg	252	179	155	147	107

4.1.2 Dosažené výsledky

Z výše uvedené tabulky vyplývá, že při detekci bodů dochází k různým změnám, a že počet detekovaných bodů je rozdílný v každém ze snímků. Počet detekovaných bodů je taktéž závislý na prahové hodnotě. Tento testovací případ ve scéně obsahuje jenom jeden objekt. Objekt sledujeme a nedetekujeme žádné nežádoucí body, které by byly detekované zbytečně. Už v takto jednoduché scéně ale můžeme vidět, že v určité pozici nám vznikají nové body, které třeba v minulé scéně nebyly. Na druhé straně máme také body, které v minulé scéně ještě byly, ale v následující vůbec

detekované nejsou. Toto bude patrnější u dalších sekvencí, které budou reálněji zobrazovat scénu pohybu. Ve výsledku tedy můžu říct, že počet detekovaných bodů je ovlivněn zvolenou prahovou hodnotou. Tato hodnota se musí určit předem a to je menší úskalí. Očekáván byl i různý počet detekovaných bodů v jednotlivých snímcích. Při této detekci tedy zůstává otázkou, zda zvolit menší prahovou hodnotu a očekávat větší počet detekovaných bodů nebo zvolit větší prahovou hodnotu a vystačit si s menším počtem detekovaných bodů. Toto je otázka, která se musí řešit individuálně podle potřeby.

4.2 Hodnocení korespondence bodů

Korespondence bodů probíhá podle dříve popsaného algoritmu. I zde je závislost na zvolené prahové hodnotě. Z nižší prahovou hodnotou bylo detekováno více významných bodů. Proto při hledání korespondence pro konkrétní bod se objevovalo více bodů, které by se mohly za korespondující prohlásit. Nastávaly situace, kdy pro určitý bod nebyla nalezena korespondence. Toto je zcela realistický obraz reálných scén, kdy se tyto situace zcela normálně vyskytují. Hodnocení korespondence je následující.

4.2.1 Korespondence jednoduché sekvence

Opět se podíváme na jednoduchý příklad pohybujícího se autíčka. Byla vytvořena sekvence pohybu autíčka. Základní postavení bylo zobrazeno na předešlém obrázku. Vytvořil jsem celkem pět pozic. Poslední pozice vypadá následovně.

Obrázek 14: Poslední pozice autíčka

Nejdříve se zaměřím na počty korespondujících bodů v jednotlivých obrázcích v závislosti na zvolené prahové hodnotě. Hodnoty uvádím v následující tabulce.

Tabulka 2: Počet korespondujících bodů

Název	Prahová hodnota				
	Cor= 1000	Cor= 5000	Cor=8000	Cor=10000	Cor=25000
Auto1.jpg	182	106	83	78	54
Auto2.jpg	186	119	99	89	67
Auto3.jpg	192	113	94	87	66
Auto4.jpg	198	131	111	103	78
Auto5.jpg	200	136	116	107	75

4.2.2 Dosažené výsledky

Už z předešlé tabulky vyplynulo, že čím menší byla prahová hodnota, tím více bylo detekovaných významných bodů. Proto při prahových hodnotách 10000 a více se bod korespondence hledal maximálně mezi třemi až pěti body, které přicházely v úvahu. Při prahových hodnotách nižších se stává, že pro daný bod se nachází až 20 bodů, které by mohly být korespondující. Můžeme tedy říci, že nižší prahová hodnota má za následek větší výběr korespondujících bodů. Otázkou opět zůstává, zda mít větší počet bodů, které přicházejí v úvahu a mít tím větší výběr, nebo si vystačit s menším počtem bodů kdy hrozí, že korespondence nemusí být vůbec nalezena.

4.3 Hodnocení shlukování

Pokud se podívám na výsledky shlukování v mé diplomové práci, tak vidím, že se tvořily různé počty shluků. Byly odlišné počtem bodů ve shluku. Toto bylo ovlivněno pravidlem přiřazování bodů do shluku. V této problematice můžeme využít celou škálu možnosti, jak shluky tvořit. Tím jsem se zabýval v teoretické části této práce.

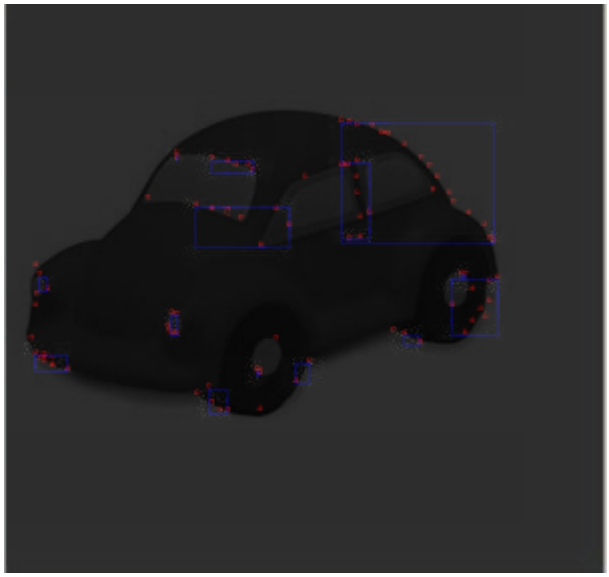
4.3.1 Shlukování jednoduché sekvence

I pro tuto část zvolím sekvenci pohybujícího se autíčka. Nejdříve jsem vyzkoušel pravidlo blízkosti bodů. Do shluku jsem přidával body, které byly velmi blízko sebe. Řešení velmi jednoduché, ale závislé na pořadí bodů. Vytvářely se shluky, které v sobě měly třeba jen dva nebo tři body. Tento shluk by se mohl spojit s již vytvořeným shlukem. K tomuto kroku bych využil vícenásobné procházení vytvořených shluků, a snahu o spojení jednotlivých shluků k sobě. Toto vylepšení by samozřejmě sebou neslo větší časovou náročnost při tvorbě shluků a tedy i při chodu celého mého programu. Program tvoří shluky v jednom cyklu. Řešení tvoření shluku pro jeden cyklus přineslo tyto výsledky:

Tabulka 3: Počty shluků v obrazech

Název	Prahová hodnota				
	Cor= 1000	Cor= 5000	Cor=8000	Cor=10000	Cor=25000
Auto1.jpg	17	21	19	19	15
Auto2.jpg	12	19	19	19	18
Auto3.jpg	9	13	15	17	15
Auto4.jpg	10	13	14	15	18
Auto5.jpg	12	11	11	10	12

Z výše uvedené tabulky můžeme vyčíst, že i když se na obrázku jedná o jeden objekt, bylo by zapotřebí mít toto auto jako jeden shluk. Program auto rozdělí na menší shluky, které už nespojuje do větších celků. Na následujícím obrázku vidíte rozdělení automobilu.



Obrázek 15: Detekování shluků

Z počtu shluků je patrné, že při jemnější detekci jsou body velmi blízko sebe a pak je program schopen detekovat menší počet shluků. Při hrubší detekci je počet detekovaných bodů menší. Od toho se odvíjí i jejich tvorba. Body jsou od sebe ve větší vzdálenosti, a proto je není možné dát do menšího počtu shluků. Při tomto detekování se velmi často stává, že se tvoří shluky, které mají

pouze jeden bod. Počty shluků ukazují, že v každém obraze je detekován rozdílný počet bodů a to má za následek že i počty shluků se různě mění.

4.3.2 Určení pohybu shluků

Pro zjištěné shluky bylo nutné určit směr pohybu. Pohyb se určoval z těžiště každého shluků. Na základě souřadnic těžiště shluku a těžiště korespondujícího shluku bylo možné určit směr pohybu. Program zobrazuje pro každý vytvořený shluk směr pohybu. Určení směru pohybu je vlastnost, která se při pohybu sleduje. Tento údaj by bylo možné využít i při spojování jednotlivých shluků. Každé shluky, které by se pohybovaly stejnou rychlosti a směrem a zároveň jejich těžiště by byly dostatečně blízko, mohly by se spojit. Vycházel bych z předpokladu, že se jedná o jeden objekt, který se pohybuje určitým směrem. Experimentálně jsem to na několika snímcích vyzkoušel a o výsledku se zmíním v další sekci.

4.4 Hodnocení reálné scény

Na mou aplikaci jsem zkusil poslat i obrázky z reálných scén, kdy už je ve scéně více objektů. Jsou zde i objekty, které se nepohybují a nejsou předmětem mého zájmu. Takové objekty bychom potřebovali eliminovat, jelikož detekování bodů na takových objektech je nežádoucí. Bohužel toto není reálně proveditelné, jelikož program nedokáže již při detekci říct, jestli je tohle bod, který nás zajímá, nebo je to bod statický a tudíž pro naše využití je nepotřebný.

4.4.1 Detekce bodů na reálném obrázku

Proces detekování byl již několikrát zmíněn, a proto zde ještě uvádím výsledky, jak se program zachová při detekci reálných obrázků. Určitě je potřeba uvést detekci v reálných obrázcích, pro které by měl být celý program použit. Pro ukázku jsem si nejdříve vybral jednoduchou fotografii automobilu, který můžete vidět na následujícím obrázku.

Obrázek 16: Fotografie auta

Již zde neuvádím tabulku detekovaných bodů, ale stručně uvádím výsledek detekce. Program na základě této fotografie detekoval obrys automobilu, kde jsou nalezeny významné body. Detekoval také okno, které se odlišuje od karoserie a je samostatně detekovatelné. Již na ukázce jednoduchého obrázku bylo patrné, že počet detekovaných bodů je závislý na zvolené prahové hodnotě. Stejné to je i v tomto případě. Program detekoval tento výsledek při prahové hodnotě $cor = 20000$.

Obrázek 17: Dekování bodů na reálném obrázku

Když zvolíme malou prahovou hodnotu, vidíme velmi jemnou detekci celého automobilů. Pokud budeme tuto hodnotu zvyšovat, bude se počet detekovaných bodů zmenšovat a i detekování bude hrubší. Snažil jsem se vyzkoušet detekci na fotografiích aut, které jsou focené z různých stran, abych vyzkoušel a pokryl co možná největší rozsah. Fotogalerie snímků a detekovaných bodů je uložena na CD.

4.4.2 Detekování reálné sekvence

V poslední části hodnocení, jsem využil kamerový systém společnosti OVANET a nahrál jsem si určité sekvence z křižovatek, které jsem pak vyzkoušel v mém programu. Snímky byly vytvořeny z videosekvencí, které jsou pořizovány z kamer zabudovaných na křižovatkách. Tyto sekvence nejsou přenášeny v nejlepší kvalitě a proto i snímky pro program jsou v horší kvalitě. Navíc kamery jsou v určité vzdálenosti od křižovatek a automobily na snímcích jsou v menší velikosti. Program tedy pracoval s obrázky, které vypadaly následovně.

Obrázek 18: Reálná podoba křižovatky

Pro porovnání se můžete podívat na tabulku, ve které vidíme množství významných bodů, které se v takovéto scéně mohou objevovat.

Tabulka 4: Počty detekovaných bodů v reálné scéně

Název	**Prahová hodnota**		
	Cor= 1000	Cor=10000	Cor=25000
kriz.jpg	580	300	210
kriz1.jpg	610	351	247

Z tabulky jasně vyplývá, že pokud budeme chtít detekovat reálné scény, budeme muset počítat s velkým množstvím nalezených bodů. Tato testovací data byla pořízena v menší kvalitě, což se odrazilo na výsledku. Program dokázal sledovat sekvence pohybů, ale nebylo to tak ideální, jak v předcházejících ukázkách. Při těchto testovacích datech se již objevila i časová složka programu. Program musel pracovat s větším počtem bodů, větší možnosti korespondence a větším počtem shluků. Na to všechno bylo zapotřebí určitého časového úseku.

Proto by v reálném čase tyto snímky nebyly možno zpracovat pomoci programu. Bylo by zapotřebí programu posílat snímky v určitých časových intervalech, aby mohly být zpracované. Program by taktéž měl po určité řadě snímků poznat oblasti, které se jakoby nepohybují. Tyto oblasti by pak v dalších sekvencích mohl vypustit a tím by získal na rychlosti. Toto není jednoduché provést, jelikož by hrozila ztráta informací. Pokud by se v těchto oblastech náhle objevil objekt našeho zájmu, byl by vypuštěn.

Součásti práce bylo i prověření jak celý systém funguje při své činnosti. Proto jsem se snažil co možná nejvíce zjistit účinnost mé práce. Program byl zkoušen na výše zmíněných testovacích datech. Běžel na klasickém stolním PC s procesorem Intel Core Duo T5500, s paměti 1024 MB. Jelikož tato testovací data nebyla nijak náročná, program nevykazoval závažné problémy a nedostatky.

4.5 Výsledek sledování pohybu

Nyní se podívám na zhodnocení všech již dříve vyhodnocovaných úkolů v celku. Systém měl umět detekovat pohyb v sekvenci snímků. Otestoval jsem tedy systém pro sekvenci pohybu autíčka v obraze. Snímek pro testování byl zobrazen výše na obrázku11. Autíčko jsem rozmístil různě na obrázcích a tímto způsobem jsem vytvořil kolekci snímků. Tyto snímky jsem postupně posílal do aplikace a vyhodnocoval je. Program reagoval na pohyb detekovaných shluků a zobrazoval jejich směr. Pro každý snímek byl vypsán seznam shluků. U každého shluku byl určen i směr pohybu. Tento směr je i graficky znázorněn. Na snímcích je vždy jenom jedno auto, které je vždy posunuté nějakým směrem. Na snímku by se tedy všechny shluky měly pohybovat stejným směrem. Výsledky programu jsou ve většině případů předpokládané. Našly se snímky se shluky, které měly trochu odlišný směr. Toto je způsobeno několika faktory. Směr pohybu určuji na základě rozdílů souřadnic těžišť shluků z předešlého a aktuálního snímků.

Na základě vypočteného rozdílů souřadnic x a y bude rozhodnuto o směru pohybu. Určení velikosti x a y bylo provedeno experimentálně podle testovacích dat. Toto rozdělení je pro určitý typ testovacích dat použitelné, ale v jiných testovacích datech už může nastat chybné určení. Určování směrů souvisí i se vzdáleností pohybu objektů v jednotlivých snímcích. Program vychází z předpokladů velmi malých pohybů v jednotlivých snímcích. V opačném případě budou velikosti souřadnic příliš velké a detekování směrů bude nepřesné. Ideální by bylo nejdříve vyzkoušet a určit jednotlivé směry a pak používat. Z tohoto důvodu jsou některé shluky směřovány trochu odlišným směrem, jelikož velikost rozdílových souřadnic x a y zapadají do odlišných směrů. Tyto rozdíly jsou většinou minimální.

Detekování bylo testováno na snímcích menší velikosti a program pracoval plynule bez větších problému. Počty významných bodů byly malé, a tedy nebylo nutné tolik času pro korespondenci a shlukování. V případě zvětšování počtu detekovaných bodů se bude prodlužovat doba, kterou program bude potřebovat na korespondenci a shlukování. Program je tedy schopen pracovat plynule za předpokladu menšího počtu detekovaných bodů. V opačném případě bude program potřebovat určitý čas na zpracování. Nasazení v reálném světě je závislé na potřebách aplikace a faktorech, které ovlivňují chod programu.

4.6 Experimenty se shluky

Program tvoří shluky v nejjednodušší podobě. Proto jsem pouze zkusil experimentálně, co by se stalo, kdyby se tyto shluky tvořily trochu jinak. Zkusil jsem při tvorbě shluků vícenásobné procházení shluků se snahou o spojení dvou do jednoho. Při tomto spojení jsem použil pravidlo

blízkosti dvou bodů z těchto shluků. Pokud jsem takové body našel, prohlásil jsem je za blízké a tyto shluky jsem spojil. Ve výsledku toto vylepšení mělo za následek zmenšení počtu shluků na úkor vícenásobného procházení shluků. Toto vylepšení bylo dobře použitelné na snímcích, kde bylo pouze jedno auto a případné další objekty na snímku byly dostatečně daleko od tohoto auta. V takovém případě se jednalo o spojování shluků, které tvořily určité části tohoto automobilu. V ideálním případě by mohl vzniknout jeden velký shluk, který by reprezentoval celý automobil.

Pokud se na snímku vyskytovalo více automobilů, nebo na snímku byl jenom jeden automobil, ale v jeho blízkosti se vyskytovaly další detekované oblasti, vznikl problém se spojováním. Takto vzniklé shluky nebylo možné touto jednoduchou úpravou spojovat. Nemohl jsem tedy toto jednoduché vylepšení použít.

Další vylepšení se naskytlo ve spojování shluků na základě blízkosti těžišť jednotlivých shluků a směru pohybu. Tato možnost přinesla větší rozsah použití. Pohyb každého shluku byl vypočten ze dvou snímků. Tato metoda se dala použít jak u jednoho automobilu na snímcích tak i při větším počtu. Otázkou ale zůstává určení parametru blízkosti těžišť shluků a směru pohybu. Jelikož shluky sice mohou být od stejného objektu, ale těžiště nemusí být blízko a směr pohybu obou těchto shluků nemusí být stejný.

Tyto možnosti byly prověřený pouze experimentálně a zdálo se, že by mohly přinést vylepšení při nasazení v reálných případech. Byly pouze orientační, a proto je více zde neuvádím a ani ve výsledném programu nejsou zakomponovány.

Zajímavý výsledek přinesl detekování blížícího se automobilu přímo proti nám. Tento výsledek je zcela očekáván. Při detekci směru pohybu shluků bylo patrné, že každý shluk má svůj směr. Jednoduše řečeno každý shluk se pohybuje jinam. Když se nad tímto případem trochu zamyslíme, dojdeme k závěru, že je to zcela správné. Automobil se blíží k nám a na snímku se zvětšuje. Jednotlivá těžiště korespondujících shluků se posouvají k okraji snímku. Každý shluk se pohybuje k okraji snímku. Tento případ je zcela výjimečný a program není schopen určit směr pohybu automobilů. Program správně určí směry pohybujících se shluků, ale výsledný pohyb není správně určen.

5 Programová část

V této části uvedu popis programu, který by měl demonstrovat dříve zmíněnou teoretickou část. Celý program byl naimplementován ve Visual studiu 2005 a byl využit programovací jazyk C a C++. Program je rozdělen do několika částí, kdy se každá z částí zaobírala určitým problémem, který byl teoreticky popsán. Celý program obsahuje několik tříd.

Všechny třídy budou v této části popsány a bude vysvětleno jejich praktické využití. Při tvorbě celé aplikace jsem postupoval po jednotlivých dílčích krocích. Prvním krokem jsem vytvořil program, který dokáže vyhledat významné body. V další části jsem přidal korespondenci bodů ve dvou snímcích. Na závěr přišlo shlukování a určení pohybu objektů v obraze.

5.1 Programové balíky

Celý program můžeme rozdělit do 3 stěžejních úkolů, které byly postupně implementovány. Detailnější popis tříd je v sekci Přílohy v tomto dokumentu.

5.1.1 Praktická detekce bodů

Dříve zmíněnou teoretickou část jsem využil a prakticky naimplementoval. Zmíněný obraz jsem načetl do proměnné, se kterou jsem dál pracoval. Vytvořil jsem dvě třídy a to třídu Bod a Matice (popis tříd najdete v sekci 5.2). V nich jsem pracoval s jasovou složkou každého pixelů v obraze a snažil jsem se určit, zda je konkrétní bod významný či nikoli. Získal jsem kolekci významných bodů, s kterou jsem dále pracoval. Celá tato část byla poté odzkoušená na různých obrazcích od velmi jednoduchých, které tvořili základní geometrické útvary až ke složitějším.

Tato detekce se řídí předem stanovenou prahovou hodnotou. Hodnota určuje, které body jsou důležité. Proto je zapotřebí na začátku tuto hodnotu stanovit a v případě, že je v programu nastavena špatně, může docházet k chybným detekcím. Nyní uvedu jednoduchý popis algoritmu pseudokódem.

Vstup: Vstupní obrázek.

```
For 1 to výška obrázku
      For 1 to šířka obrázku
            získej hodnotu pixelu

for 1 to weight
      for 1 to height
            spočítej cor(x, y)

for 1 to weight
```

```
for 1 to height
      porovnej cor(x,y) s konstantou
      rozhodování
      je hodnota cor (x,y)>konstanta
      {
      porovnej všechny cor(x,y)v okolí
      pokud je maximální tak jde o vyznamný bod
      }
```

Výstup: pole bodů, které byly určené za významné.

5.1.2 Praktická korespondence bodů

Korespondenci bodů jsem naprogramoval pomocí již dříve zmíněné metody Lucas-Kanade. Program hledá korespondující body v jednom cyklu. Nevyužil jsem možnost pyramidálního sledování. Program postupně prochází všechny detekované významné body z dřívějšího obrazu a snaží se najít jeho obraz v aktuálním snímku. Hledání probíhá vždy mezi všemi body v aktuálním obraze. Z těchto bodů beru v úvahu jen ty, které jsou v těsné blízkosti bodu předešlého obrazu. Pro všechny body, které jsou v této blízkosti je spočítány rozdíl jasů podle vzorce (7). Podle nejmenší hodnoty určuji, které body mezi sebou korespondují. V reálných scénách často dochází ke změně počtu bodů. Počet detekovaných bodů v obrazech je odlišný. Proto v mém programu nastávají situace, kdy jeden bod v aktuálním obraze může mít více korespondujících bodů z předešlého obrazu. Vycházím z předpokladů, že se body z předešlého obrazu ztratily a tím nebyly v aktuálním obraze detekovány. Nyní naznačím jednoduchý popis algoritmu pseudokódem.

Vstup: Dvě kolekce významných bodů z aktuálního a předešlého obrázku.

```
For 1 do (počet bodu v minulem obrázku)
{
      For 1 to (počet bodu v aktuálním obraze)
            {
            Najdi blízké body
            Spočítej rozdíl jasu
            }
Porovnej všechny vypočtené rozdíly
Ten s nejmenší hodnotou je korespondujícím bodem
}
```

Výstup: Množina korespondujících dvojic bodu.

5.1.3 Praktické shlukování bodů

Shlukování bodu jsem naimplementoval pomocí vzdálenosti bodů. V prvním kroku jsem vytvořil první shluk a do něho jsem vložil první bod, který jsem při procházení obrazu našel. V další části jsem procházel jednotlivé body a snažil jsem se je umístit do patřičného shluku.

Pro určení vzdálenosti bodu jsem opět využil souřadnice bodů. Porovnával jsem tedy souřadnice bodu s body ve shlucích. Snažil jsem se najít blízky bod. Za blízký jsem prohlásil takový, jehož x-ová a y-nová souřadnice se lišily maximálně o předem stanovené konstantní hodnoty.
Při takto zvolené implementaci mohly nastat dvě možnosti.

a) Konkrétní shluk – Při prohledávání už vytvořených shluků jsem v určitém shluku našel bod, který jsem prohlásil za blízký. V takovém to případě se daný bod přidal do tohoto shluku.

b) Není shluk – Při prohledávání všech již vytvořených shluků jsem nenašel žádný, který by byl blízký. V takovémto případě bylo potřeba vygenerovat nový prázdný shluk. Do tohoto shluku se pak tento bod vložil.

Takto vytvořený algoritmus vytvořil shluky, jejichž body byly blízko sebe. Samozřejmě že jsou zde i nevýhody. Takto popsaný algoritmus prochází celou kolekci bodů jenom jednou a je závislý na pořadí detekovaných bodů. Proto se stává v programu, že jsou vytvořeny shluky, které by se daly spojit, ale muselo by se toto spojení provádět v dalších krocích. Znamenalo by to časové prodloužení celého procesu shlukování. Nyní uvedu jednoduchý popis algoritmu pseudokódem.

Vstup: Seznam významných bodů

```
For 1 to počet detekovaných bodů
      {
            For 1 to počet vytvořených shluku
            {
            Najdi místo v nějakém shluku
            }

      Rozhodování
      Našli jsme shluk, pak tam vlož tento bod
      Nenašli jsme shluk, pak vytvoř nový a vlož tam tento bod
      }
```

Výstup: Kolekce shluku a v nich jednotlivé body.

Pro každý takto vytvořený shluk jsem určil těžiště. Toto těžiště bylo udáno souřadnicemi $T(x, y)$ .Následují rovnice výpočtů souřadnic těžiště.

$$T_x = \left(\sum_{i=1}^{n} A_{ix} \right) / n$$

$$T_y = \left(\sum_{i=1}^{n} A_{iy} \right) / n$$

n …je počet bodu ve shluku
A_{ix} … x -sová souřadnice i …tého bodu ve shluku
A_{iy} … y -sová souřadnice i …tého bodu ve shluku

Určení směru pohybu jsem tedy realizoval pomoci rozdílů souřadnic těžišť, které byly ve dvou po sobě jdoucích snímcích nalezeny a určeny jako korespondující. Tato informace je základním stavebním kamenem pro určování směru pohybu v mé práci.

5.2 Popis tříd

V programu jsem vytvořil pro své potřeby níže uvedené třídy. Každá třída je přizpůsobena pro řešení určitého problému v systému. Třídy byly tvořené podle postupného vývoje systému. Většina z tříd se skládá ze dvou souborů. První z nich je hlavičkový soubor, kde je definice všech proměnných a metod, které bude daná třída využívat. Druhý soubor obsahuje deklarace jednotlivých tříd, neboli vlastní tělo každé z metod.

5.2.1 Třída Bod

Tato třída simuluje jeden pixel v obraze, tedy uchovává patřičné souřadnice konkrétního bodu. U významných bodů je zapotřebí vědět jejich souřadnice. S tím souvisí i následná korespondence, kdy využívám právě tyto instance.

Proměnné

Pro tuto třídu jsem použil jenom dvě proměnné, které určovaly souřadnice v obraze.

- x – určuje souřadnici na ose x
- y – určuje souřadnici na ose y

Metody

V této třídě nalezneme pouze metody, které naplňují patřičnou proměnnou, nebo naopak z proměnné hodnotu získávají.

- **getx()** – metoda získá hodnotu z proměnné x
- **gety()** - metoda získá hodnotu z proměnné y
- **setx()** – metoda naplní proměnnou x

- **sety()** – metoda naplní proměnnou y

Ukázka zdrojového kódu z třídy Bod. h

```
class bod
{
public:
//x-sová souřadnice
int x;
//y-nová souřadnice
int y;
// ostatni funkce
public:
      // ziskání proměnné x
      int getx(void);
      // ziskání proměnné y
      int gety(void);
      // nastaveni proměnné x
      void setx(int cislo);
      //nastavení proměnné y
      void sety(int cislo);
};
```

5.2.2 Třída Matice

Tato třída má za úkol uchovávat důležité informace o daném bodě. Na základě těchto informací se určí, zda daný bod je bodem našeho zájmu, tedy pro naši aplikaci důležitý. V opačném případě tento bod již dále nevyužíváme. Tato třída byla vytvořena za účelem realizace Harrisova algoritmu a všech výpočtů potřebných k získání informací o daném bodě. Obsahuje čtyři proměnné, které reprezentují matici o rozměrech $2x2$ a na kterých je celý algoritmus založen.

Proměnné

- pole11 – prvek pole
- pole12 – prvek pole
- pole21 – prvek pole
- pole22 – prvek pole
- lambda1 – výsledný výpočet
- lambda2 – výsledný výpočet
- cor – hodnota pro určení důležitosti bodů

Metody

Pro každou dříve zmíněnou proměnnou jsou vytvořený dvě metody a to set+”název proměnné ” a get+” název proměnné”. První metody slouží pro naplnění proměnné patřičnou hodnotou a druhé metody jsou pro získání hodnoty z konkrétní proměnné. Detailnější popis najdete v sekci Přílohy, kde jsou celé třídy vloženy. Další metody, které jsou v této třídě využity.

- **determinant** () - metoda pro spočítání determinantu matice
- **trace** () - metoda pro vracení součtu na hlavní diagonále

Ukázka zdrojového kódu ze souboru matice. h

```
class mat
{
public:
// prvek  matice
int pole11;
// prvek  matice
int pole12;
// prvek  matice
int pole21;
// prvek  matice
int pole22;
// kriterium pro významný bod
double lambda1;
// kriterium pro významný bod
double lambda2;
// kriterium pro významný bod
double cor;



//ostatní funkce
public:
// nastaveni promměnné
void setpole11(int cislo);
// nastaveni promměnné
void setpole12(int cislo);
// nastaveni promměnné
void setpole21(int cislo);
// nastaveni promměnné
void setpole22(int cislo);
// nastaveni promměnné
void setlambda1(double cislo);
```

…….celá třída je zobrazena v příloze

5.2.3 Třída Shluk

Tato třída má za úkol v sobě uchovávat body, které jsou si blízké a tvoří společný celek. Takto vytvořené celky pak určují pohyb na snímcích. V programu je zjišťován pohyb jednotlivých bodů ve shluku a tím sleduju pohyb ve snímcích.

Proměnné

- počet - určuje počet bodu v tomto shluku
- pole[] - slouží k uložení jednotlivých bodů korespondence do shluku
- xmin - souřadnice pro ohraničení shluku
- xmax - souřadnice pro ohraničení shluku
- ymin - souřadnice pro ohraničení shluku
- ymax - souřadnice pro ohraničení shluku

Metody

- **setpocet () –** Nastavuje počet bodů na požadované číslo
- **getpocet () –** Funkce vrátí aktuální počet bodů ve shluku
- **setbod ()** – Funkce uloží bod do shluku
- **getbod() –** funkce která je implementována ve dvou podobách. Funkce buď vrací korespondující body na poslední pozici, nebo v druhém případě je funkce implementována s parametrem a tato funkce vrací body z pozice, kterou určuje parametr
- **kontrolabodu() –** funkce má za úkol zkontrolovat, zda se bod z původního obrazu se svým korespondujícím bodem mají vložit do nějakého již vytvořeného shluku, nebo je zapotřebí vytvořit shluk nový

Ukázka třídy shluk.h

```
class shluk
{
public:
      // počet bodu ve shluku
      int pocet;
      // seznam uložených bodů
      bodk body[700];
      // souřadnice pro ohraničení shluku
      int xmin;
      int xmax;
      int ymin;
      int ymax;
```

```
public:
      // nastaveni počtu bodu
      void setpocet(int cislo);
      // ziskání hodnoty počtu
      int getpocet(void);
      //nastavení korespondujících bodů
      void setbod(bodk roh);
      //ziskání konkretní korespondence na
poslední pozici
      bodk getbod(void);
      //ziskání konkretní korespondence na určené
pozici
      bodk getbod(int cislo);
      //metoda pro určení zařazení bodu do shluku
      bool kontrolabodu(bod roh);
};
```

5.2.4 Třída Bodk

Tato třída se stará o spojování korespondujících bodů, které se nacházejí při porovnávání bodů. Když je nalezena korespondence bodů, tak se oba dva body zařazují do této třídy. Tato třída se dále využívá při zjišťování pohybu jednotlivých shluků.

Proměnné

Jak již bylo zmíněno, třída se stará o korespondenci a tedy jsou v ní definované dvě proměnné pro body. První je určena bodu, který byl detekován v minulé scéně. V druhé proměnné je pak korespondující bod. Tento bod je nalezen v aktuálním obraze.

- bod puvod – proměnná pro původní bod
- bod obraz – proměnná pro korespondující bod

Metody

Tato třída obsahuje pouze metody, které slouží k práci s proměnnýma.

- **getPuvod** () – ziskaní původního bodu
- **getObraz** () – získání korespondujícího bodu
- **setPuvod** () – nastavení původního bodu
- **setObraz** () – nastavení korespondujícího bodu

Ukázka třídy Bodk.h

```
class bodk
{
```

```
public:
      //proměnná pro vychozí bod
      bod puvod;
      //proměnná pro korespondující bod
      bod obraz;
public:
      //metody pro ziskavání hodnot z proměnných
      bod getPuvod();
      bod getObraz();
      //metody pro nastavování proměnných
      void setPuvod(bod bodik);
      void setObraz(bod bodik);

};
```

5.2.5 Třída Metody

Tato třída se stará o fungování celého programu. V ní jsou naimplementované metody, které se starají o správné fungování. Jsou tady metody, které se starají o všechny tři části aplikace a to o detekci významných bodů, korespondenci bodů i tvorbu jednotlivých shluků.

Proměnné

Tato třída nemá v sobě žádné proměnné, protože slouží k chodu celého programu. Jsou v ní pouze metody, které se starají o všechny již dříve zmiňované úkoly.

Metody

- **gaussova** () – Funkce realizující Gaussovou funkci u hledání významných bodů.
- **naplneni_maticek**() – Funkce se stará o prvotní naplnění matic pro výpočet významných bodů. Toto prvotní naplnění je základ pro konvoluci a získání významných bodů.
- **konvolucni_maska** () – Funkce, která inicializuje pole s hodnotami pro konvoluci.
- **korespondence() -** Metoda pro nalezení korespondujicich bodů.
- **posunbodu() –** Metoda pro přesun hodnot obrazu pro další zpracování při posunu obrazu z pozice aktuální do pozice předchazející.
- **hledanirohu() –** Metoda, která hleda významné body v obraze.

Všechny metody jsou ukázaný v sekci Přílohy.

Ukázka třídy Metody.h

```
class metody
{
```

```
public:
      metody(void);

public:
      ~metody(void);

public :
//    metoda pro výpočet gausianu
float gaussova(int hodnota,float pi);

//    metoda pro nalezeni korespondujicich bodu
void korespondence(int pocetbodu, int pocetbodu1,int
&pocetkores, bod body[], bod body1[], int **obraz,int
**obraz1, bodk korespon[]);

//metody pro výpočet těžiště shluku
int souradnice_shluk_puvod_x(shluk sh );
int souradnice_shluk_obraz_x(shluk sh );
int souradnice_shluk_puvod_y(shluk sh );
int souradnice_shluk_obraz_y(shluk sh );

//metoda pro přesun hodnot při načitání dalšího snímku
void posunbodu(IplImage *inpImage,int &pocetbodu, int
&pocetbodu1, bod body[], bod body1[],int **&obraz,int
**obraz1 );

//metoda pro výpočet hodnot v druhém obraze
void druhybod(IplImage *inpImage,int **obraz1,mat
**maticka1,bod body1[],int &pocetbodu1,double
**m,double **konvoluce);

//metoda pro načtení matice bodu před konvoluci
void uvod_zavorky(IplImage *inpImage,double **m,mat
**maticka,double **konvoluce);

//metoda pro výpočet hodnot matice
void naplneni_maticet(IplImage *inpImage,int
**obraz1,int **obraz,mat **maticka,mat **maticka1);

//načteni hodnot konvolučni masky 3x3
void konvolucni_maska(double **&konvoluce);
//metoda pro výpočet hodnoty určující významné body

void vypocet_cor(mat **&maticka ,IplImage *inpImage);
//metoda, která hledá významné body

void hledanirohu(mat **maticka ,IplImage *inpImage, int
&pocetbodu, bod body[]);
};
```

6 Závěr

V diplomové práci jsem se zabýval detekcí pohybu ve snímcích. V první kapitole jsem stručně nastínil, čím se ve své práci budu zabývat. V dalších částech jsem se už zabýval jednotlivými algoritmy, které se využívají. Zhodnotil jsem vlastní aplikaci, která měla realizovat detekci pohybu v praxi.

Vytvořil jsem program, který dokáže rozeznat a detekovat významné body v obrazech. S těmito body dokáže dále pracovat. V určité sekvenci umí najít korespondující body, mezi sebou spojit patřičné body do shluku a určit směr pohybu těchto shluků. Program správně pracuje na jednodušších snímcích, kde určí pohyb všech shluků. Dokáže pracovat i se snímky reálnými. Zde je výskyt většího počtu detekovaných bodů, větší množství shluků a program potřebuje větší časovou prodlevu. Dokáže vytvářet základní shluky, které jsou závislé na pořadí bodů.

Nasazení pro reálnou aplikaci je tedy možné, ale výsledky budou v omezené míře. Kritériem pro práci určitě bude, jak rychle by měl systém pohyb určovat a v jaké kvalitě budou dodávány patřičné snímky. Závěrem bych řekl, že jsem si vyzkoušel řešení problematiky detekce pohybu a vytvořil jsem jednoduchý systém, který dokáže tento úkol vyřešit.

Literatura

[1] *Jiří Žára, Bedřich Beneš, Jiří Sochor, Petr Feikel* ,Moderní *počítačová grafika - 2 vydání ,ISBN – 80-251-0454-0*

[2] Internetové stránky *http://en.wikipedia.org/wiki/Main_Page*

[3] Eduard Sojka ,Skripta *Digitální zpracování obraz,* FEI-VSB

[4] Bruce D. Lucas, Takeo Kanade, *An Iterative Image Registration Technique with an Applicationto Stereo Vision*, Computer Science Department, Carnegie-Mellon University,Pittsburg, Pensylvania 1981.

[5] Konstantinos G. Derpanis, *The Harris Corner Detector*, 2004.

[6] Skripta *Rozdělující algoritmus shlukování pomocí k-středů* autor: Lumír Návrat

[7] Berkhin. P, Survey of Clustering Data Mining Techniques

[8] L.Lorenc ,Shlukování, http://www.fit.vutbr.cz/study/courses/ZZD/public/seminar0304/Shlukovani2.pdf

[9] Další internetové stránky naleze ve vyhledávači www.google.cz

7 Přílohy

I. Popis CD

/program-exe/	Složka se spustitelnou aplikaci
/program-visual/	Složka s aplikaci určenou pro vývojové prostředí Visual Studia 2005
Diplomova_prace.pdf	Diplomová práce
/napoveda	Nápověda ovládání programu

II. Třídy programu

Soubor bod.h

```
pragma once
#include <stdio.h>
#include <tchar.h>
class bod
{
public:
//x-sová souřadnice
int x;
//y-nová souřadnice
int y;

// konstruktor
public:
      bod(void);
// destruktor
public:
      ~bod(void);
// ostatni funkce
public:
      // ziskání proměnné x
      int getx(void);
      // ziskání proměnné y
      int gety(void);
      // nastaveni proměnné x
      void setx(int cislo);
      //nastavení proměnné y
      void sety(int cislo);
};
```

Soubor bod.cpp

```
#include "StdAfx.h"
#include "bod.h"

bod::bod(void)
{
      x=0;
      y=0;
}

bod::~bod(void)
{
}
void bod::setx(int cislo){
x=cislo;
}

void bod::sety(int cislo){
y=cislo;
}

int bod::getx(){

      return x;
}

int bod::gety(){
      return y;
}
```

Soubor mat.h

```
#pragma once
#include <stdio.h>
#include <tchar.h>
class mat
{
public:
// prvek  matice
int pole11;
// prvek  matice
int pole12;
// prvek  matice
int pole21;
// prvek  matice
int pole22;
// kriterium pro vyynamny bod
double lambda1;
// kriterium pro vyynamny bod
double lambda2;
// kriterium pro vyynamny bod
double cor;

// konstruktor
public:
      mat(void);
// destkruktor
public:
      ~mat(void);
      //ostatní funkce
public:
      // nastaveni promměnné
      void setpole11(int cislo);
      // nastaveni promměnné
      void setpole12(int cislo);
      // nastaveni promměnné
      void setpole21(int cislo);
      // nastaveni promměnné
      void setpole22(int cislo);
      // nastaveni promměnné
      void setlambda1(double cislo);
      // nastaveni promměnné
      void setlambda2(double cislo);
      // nastaveni promměnné
      void setcor(double cislo);
      // získání hodnoty z proměnné
      int getpole11(void);
      // získání hodnoty z proměnné
      int getpole12(void);
      // získání hodnoty z proměnné
      int getpole21(void);
      // získání hodnoty z proměnné
      int getpole22(void);
      // získání hodnoty z proměnné
      double getlambda1(void);
      // získání hodnoty z proměnné
```

```
        double getlambda2(void);
        // získání hodnoty z proměnné
        double getcor(void);
        // funkce na výpočet determinantu
        int Determinant();
        // funkce pro výpočet součtu na hlavní diagonále
        int Trace();
};
```

Soubor mat.cpp

```
#include "StdAfx.h"
#include "mat.h"

mat::mat(void)
{
      //hodnota=new int [2][2];
      pole11=0;
      pole12=0;
      pole21=0;
      pole22=0;
      lambda1=0;
      lambda2=0;
      cor=0;
}

mat::~mat(void)
{
}

// nastaveni prvku matice na pozici 11

void mat::setpole11(int cislo)
{
pole11=cislo;
}

// nastaveni prvku matice na pozici 12
void mat::setpole12(int cislo){
pole12=cislo;
}

// nastaveni pozice matice na pozici 21
void mat::setpole21(int cislo){
pole21=cislo;
}


// nastaveni pozice matice na pozici 22
void mat::setpole22(int cislo){
pole22=cislo;
}

// nastaveni parametru lambda1 pro urceni rohu
void mat::setlambda1(double cislo){
lambda1=cislo;
}

// nastaveni parametru lambda2 pro urceni rohu
void mat::setlambda2(double cislo){
      lambda2=cislo;
}


// nastaveni parametru cor pro urceni rohu
```

```
void mat::setcor(double cislo){
      cor=cislo;
}


// vrati prvek matice na pozici 11
int mat::getpole11(void ){
return pole11;
}


// vrati prvek matice na pozici 12
int mat::getpole12(){
return pole12;
}

// vrati prvek matice na pozici 21
int mat::getpole21(){
return pole21;
}

// vrati prvek matice na pozici 22
int mat::getpole22(){
return pole22;
}

// spocitani determinantu matice
int mat::Determinant(){
return ((pole11*pole22)-(pole12*pole21));
}


int mat::Trace(){
      return(pole11+pole22);
}

// vraci promenou lambda1
double mat::getlambda1(){
      return lambda1;
}

//vraci promenou lambda2
double mat::getlambda2(){
      return lambda2;
}

// vraci promennou cor
double mat::getcor(){
      return cor;
      }
```

Soubor shluk.h

```
#pragma once
#include "bodk.h"
class shluk
{
public:
	// pocet bodu ve shluku
	int pocet;
	// seznam uložených bodů
	bodk body[700];
	// souřadnice pro ohraničení shluku
	int xmin;
	int xmax;
	int ymin;
	int ymax;

public:
	shluk(void);
public:
	~shluk(void);
public:
	// nastaveni počtu bodu
	void setpocet(int cislo);
	// získání hodnoty počtu
	int getpocet(void);
	//nastavení korespondujích bodů
	void setbod(bodk roh);
	//získání konkretní korespondence na poslední
pozici
	bodk getbod(void);
	//získání konkretní korespondence na určené
pozici
	bodk getbod(int cislo);
	//metoda pro určení zařazení bodu do shluku
	bool kontrolabodu(bod roh);
};
```

Soubor shluk.cpp

```
#include "StdAfx.h"
#include "shluk.h"
#include <math.h>
shluk::shluk(void)
{
      pocet=0;
      xmin=2000;
      xmax=0;
      ymin=2000;
      ymax=0;
}

shluk::~shluk(void)
{
}
int shluk::getpocet(){
      return pocet;
}
void shluk::setpocet(int cislo){
      pocet=cislo;
}
bodk shluk::getbod(){
      return body[pocet-1];
}
void shluk::setbod(bodk roh){
      body[pocet]=roh;
      pocet++;
      if(roh.getPuvod().getx()>xmax)
            xmax=roh.getPuvod().getx();
      if(roh.getPuvod().getx()<xmin)
            xmin=roh.getPuvod().getx();
      if(roh.getPuvod().gety()>ymax)
            ymax=roh.getPuvod().gety();
      if(roh.getPuvod().gety()<ymin)
            ymin=roh.getPuvod().gety();

}
bodk shluk::getbod(int cislo){
      return body[cislo];
}
bool shluk::kontrolabodu(bod roh){
 int pomoc=0;
      for (int i=0;i<pocet;i++)
 {
       if((abs(body[i].getPuvod().gety() -
roh.gety())<15)&&(abs(body[i].getPuvod().getx()-
roh.getx())<15))
       {
      pomoc=1;
      //printf("\n Hodnota i je %d",i);
      break;
       }
 }
      if(pomoc==0)
```

```
 return false;
      else
            return true;
}
```

Soubor bodk.h

```
#pragma once

#include "bod.h"
class bodk
{
public:
	//proměnná pro vychozí bod
	bod puvod;
	//proměnná pro korespondující bod
	bod obraz;
public:
	bodk(void);
public:
	~bodk(void);
public:
	//metody pro ziskavání hodnot z proměnných
	bod getPuvod();
	bod getObraz();
	//metody pro nastavování proměnných
	void setPuvod(bod bodik);
	void setObraz(bod bodik);

};
```

Soubor bodk.cpp

```
#include "StdAfx.h"
#include "bodk.h"

bodk::bodk(void)
{

}

bodk::~bodk(void)
{
}
bod bodk::getObraz(){
return obraz;
}
void bodk::setObraz(bod bodik)
{
obraz=bodik;
//pocet++;
}


bod bodk::getPuvod(){
return puvod;
}
void bodk::setPuvod(bod bodik)
{
puvod=bodik;
//pocet++;
}
```