VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV MATEMATIKY
FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF MATHEMATICS

NÁVRH SOFTWAROVÉHO MODULU PRO VYHODNOCOVÁNÍ OPTICKÉHO TOKU

OPTIC FLOW SOFTWARE MODULE DESIGN

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAN GRULICH

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. STANISLAV VĚCHET, Ph.D.

BRNO 2014

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav matematiky
Akademický rok: 2013/2014

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

student(ka): Jan Grulich

který/která studuje v **bakalářském studijním programu**

obor: **Matematické inženýrství (3901R021)**

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Návrh softwarového modulu pro vyhodnocování optického toku

v anglickém jazyce:

Optic flow software module design

Stručná charakteristika problematiky úkolu:

Navrhněte softwarový modul pro vyhodnocování optického toku. Tento modul bude použit pro navigaci mobilního robotu v outdoor prostředí. Jako snímací člen bude sloužit běžná webkamera. Pro návrh modulu bude nutné provést rešerši již existujících řešení. Výsledný návrh by měl být použitelný na některé dostupné platformě jednodeskových PC.

Cíle bakalářské práce:

1) Seznamte se s možnostmi využití optického toku pro navigační účely.

2) Navrhněte strukturu modulu pro účely navigace mobilního robotu.

3) Navržený modul otestujte v reálných podmínkách.

Seznam odborné literatury:

1. Sebastian Thrun, Wolfram Burgard, and Dieter Fox. Probabilistic Robotics (Intelligent Robotics and Autonomous Agents series). Intelligent robotics and autonomous agents. The MIT Press, August 2005.

2. Howie Choset, Kevin M. Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. Principles of Robot Motion: Theory, Algorithms, and Implementations (Intelligent Robotics and Autonomous Agents). The MIT Press, June 2005.

Vedoucí bakalářské práce: Ing. Stanislav Věchet, Ph.D.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2013/2014.

V Brně, dne 21.11.2013

L.S.

prof. RNDr. Josef Šlapal, CSc.	doc. Ing. Jaroslav Katolický, Ph.D.
Ředitel ústavu	Děkan fakulty

Abstrakt
Tato bakalářská práce se věnuje analýze optického toku. První část je věnovaná teoretickému rozboru problematiky sledování bodů a jejich pohybu v obraze, zejména nalezení kvalitních význačných bodů a jejich následné sledování ve videosekvenci. Tyto algoritmy jsou detailně vysvětleny na matematickém základě. Druhá část práce popisuje navržený software s implementací algoritmů a jeho funkce. Tou hlavní je schopnost určit vzdálenost snímacího členu od překážky. Dále jsou v práci popsána praktická měření.

Summary
This bachelor thesis is focused on the analysis of optical flow. The first part is dedicated to the theoretical analysis of the motion of image features. Especially important is finding quality features that can be tracked in a videosequence. Detailed mathematical description of the necessary algorithms is presented. The second part describes the software and its functions with the implemented algorithms. The main function is determining the distance of an object from the camera based on its velocity. Empirical tests of this function are presented.

Klíčová slova
Optický tok, obrazová rychlost, Lucas-Kanade, význačné body, výškoměr, pyramidová reprezentace obrazu.

Keywords
Optical flow, image velocity, Lucas-Kanade, features, altimeter, pyramidal image representation.

GRULICH, J. *Návrh softwarového modulu pro vyhodnocování optického toku.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2014. 24 s. Vedoucí bakalářské práce Ing. Stanislav Věchet, Ph.D.

Prohlašuji, že jsem bakalářskou práci *Návrh softwarového modulu pro vyhodnocování optického toku* vypracoval samostatně pod vedením Ing. Stanislava Věcheta, Ph.D. s použitím materiálů uvedených v seznamu literatury.

Jan Grulich

Děkuji vedoucímu bakalářské práce Ing. Stanislavu Věchetovi, Ph.D za odbornou pomoc a cenné rady při psaní této práce. Také děkuji mým rodičům a přátelům za podporu a trpělivost.

Jan Grulich

Obsah

Úvod

Navigace mobilních robotů je komplexní problém, jehož řešení vyžaduje využití informací z několika druhů senzorů. Předkládaná práce se zabývá využitím kamery jako hlavního senzoru. Ta dokáže snímat okolí robota a určovat parametry dalšího pohybu. Dá se takto vyhnout např. kolizím či nesprávnému směru pohybu, ale také lze z těchto kamer získat i další informace. Příkladem může být například výškoměr, kterému pro určení výšky bude stačit právě jen výstup kamerových čidel na robotu. Cílem bude napodobit systém, který při letu využívá hmyz, který výšku letu pozná podle rychlosti míhání povrchu pod sebou. Rychlost míhání ve videosekvenci je nazývána *optický tok*. Tento v přírodě pozorovaný princip bude implementován do softwaru pomocí algoritmů pro vyhledávání a trasování význačných bodů obrazu, tedy pro výpočet optického toku. Různé přístupy pro zpracování dat z kamery a získání optického toku jsou popsány v kapitole 2, kde je představeno několik hlavních metod, což jsou hlavně např. diferenční přístup nebo vyhledávání oblastí. Způsob výpočtu použitý v samotném softwaru je popsán v kapitolách 3 a 4. Provedeme také simulační měření pro zjištění závislosti mezi velikostí optického toku a výškou nad snímaným povrchem. Výsledkem bude návrh systému, na kterém může pracovat reálný výškoměr.

Kapitola 1

Analýza obrazu

1.1 Pohyb kamery

Samotným pohybem kamery se rozumí změna rozmístění pixelů mezi dvěma po sobě jdoucími obrazy. Každý pixel se posune o určitou vzdálenost v určitém směru, což lze vyjádřit vektorem. Jsou uvažovány takové videosekvence, kde se celý obraz bude pohybovat jako celek, tedy všechny vektory budou nenulové. Je možné rozlišit v podstatě dva hlavní typy pohybu - translanční(posuvný) a rotační s osou rotace blízko středu obrazu. To je kvůli faktu, že při rotaci s osou daleko od obrazu je pohyb ve skutečnosti rotační, ale v každém samostatném okamžiku videosekvence se jeví jako translační. Rotace vynikne až teprve po delším sledování videa. Se získanými vektory se dá dále lehce pracovat a poslouží jako vstupní data pro detailnější analýzu pohybu.

1.2 Vyhodnocení pohybu kamery

V zásadě můžeme předpovídat odděleně velikost i směr vektoru. Za zmínku stojí fakt, že videozáznam je pouze dvojrozměrná reprezentace našeho reálného trojrozměrného světa. Jde tedy o určitý průmět všech kamerou viditelných bodů do roviny a body, které od sebe mohou být reálně relativně daleko se k sobě přiblíží.

Nechť A a B jsou dva body, které jsou někde v prostoru a jejich euklidovská vzdálenost je "velká". Jakmile se začne snímat prostor (s oběma body) kamerou, prostor se redukuje na rovinu a na této rovině se k sobě mohou body přiblížit. Při následném pohybu kamery se každý z nich ale bude chovat odlišně. Bod bližší snímacímu členu se bude pohybovat rychleji než bod ve větší vzdálenosti. Čili rychlost pohybu bodu může napovědět něco o jeho vzdálenosti od snímacího členu. Druhou věcí, kterou můžeme využít při vyhodnocování pohybu, je směr(směrnice) vektoru. Pokud se bude snímací člen pohybovat pouze jedním směrem (translačně), budou vektory posunu s malými odchylkami rovnoběžné. V případě rotačního pohybu budou mít směrnice vektorů v jednom okamžiku různé hodnoty. Níže je stručně popsán použitý algoritmus pro výpočet těchto vektorů. Detailněji bude popsán v následujících kapitolách.

Nejprve bude nutné nalézt dostatečného množství bodů, které budeme sledovat. Budou to různé šmouhy, rohy, hrany, izolované body apod. Tímto úkolem se zabývají algoritmy hledání význačných bodů (více v kapitole 3). V další fázi nalezneme vektory pohybu jednotlivých bodů, s kterými budeme dále pracovat. Pro nalezení těchto vektorů použiji algoritmus Lucas Kanade (kapitola 4)

Kapitola 2

Optický tok

2.1 Počáteční problém

V této kapitole bude nejprve problém popsán z matematického hlediska a na konci bude definován pojem optický tok. Jak bylo řečeno, video se skládá z množství obrazů, které se promítají za sebou na výstupu a tím vzniká dojem pohybujícího se videa. Prvním krokem bude převedení videa do černobílých odstínů. Je to z praktických důvodů, jelikož je každému pixelu přiřazena pouze jedna číselná hodnota, tedy jeho odstín šedi. Je to mnohem praktičtější, než kdyby byla každému pixelu přiřazena čísla tři, tedy složky RGB. Nechť I a J jsou dva černobílé po sobě jdoucí obrázky z videa. Množiny

$$I(\mathbf{x}) = I(x, y) \qquad J(\mathbf{x}) = J(x, y) \tag{2.1}$$

vyjadřují odstíny šedi na pozici, kam směřuje vektor $\mathbf{x} = [x, y]$, kde x a y jsou souřadnice daného obrazového bodu. Každý obrazový bod si tedy představíme jako polohový vektor směřující z počátku do souřadnic uvažovaného bodu. Obrázek I budu někdy pro zjednodušení nazývat jako první obrázek a J jako druhý. Obrázky se uvažují jako matice, kde a_{ij} vyjadřuje odstín šedi pixelu na pozici $[i, j]$. V použitých algoritmech se ale uvažuje obrázek jako funkce. Ta je ale přirozeně diskrétní a nespojitá, funkční hodnota (odstín šedi) je konstantní pro celý pixel(čtverec). Proto je využita bilineární interpolace, abychom dostali funkci spojitou. Taková funkce obrazu se nazývá jasová funkce nebo funkce intenzity. Dále je důležité uvědomit si, že souřadná soustava pro tuto funkci má počátek $[0, 0]$ v levém horním rohu obrazu. Osa x směřuje doprava a osa y dolů. Je také třeba obecně označit rozměry obrazu. n_x bude představovat šířku a n_y výšku. Pravý dolní roh obrazu má tedy souřadnice $[n_x - 1, n_y - 1]$. Uvažujme obecný obrazový bod $\mathbf{u} = [u_x, u_y]$ na prvním obrazu I. Úkolem celého algoritmu je nalézt obrazový bod $\mathbf{v} = \mathbf{u} + \mathbf{d} = [u_x + d_x, u_y + d_y]$ na druhém obrazu J takový, aby $I(\mathbf{u})$ a $J(\mathbf{v})$ byly maximálně podobné. Použitý vektor $\mathbf{d} = [d_x, d_y]$ je vlastně rozdíl poloh stejného bodu na po sobě jdoucích obrazech. Nazývá se obrazová rychlost, nebo také *optický tok* [3].

2.2 Způsoby určování optického toku

S definovaným optickým tokem je možné představit několik způsobů jeho určení. Jsou čtyři hlavní způsoby podle [5]

- Diferenční přístup
- Vyhledávání oblastí
- Metody založené na energii
- Metody založené na fázi

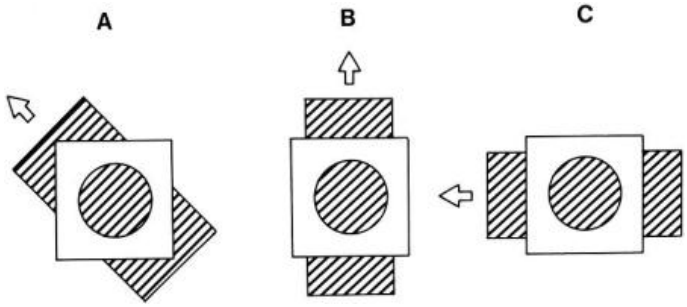
Tyto metody mají společné to, že vycházejí z předpokladu zachování intenzity. Funkce intenzity byla vysvětlena v předchozí části textu. Aby byla obsáhlá celá délka videozáznamu a ne jen dva po sobě jdoucí obrázky, tuto funkci lze rozšířit na obecnější tvar jako proměnlivou v čase, tedy $I(\mathbf{x}, t) = I(x, y, t)$ na dvojrozměrném obrázku, t představuje čas. Předpoklad zachování intenzity lze vyjádřit vztahem

$$I(\mathbf{x}, t) = I(\mathbf{x} - \mathbf{d}\delta t, t - \delta t), \tag{2.2}$$

kde $\mathbf{d} = (d_x, d_y)$ je optický tok patřící bodu x [4,5]. Charakteristiky těchto způsobů jsou vysvětleny v následujících odstavcích.

Diferenční přístup

Diferenční přístupy jsou jedny z prvních přístupů, které se objevily pro výpočet optického toku. Jejich hlavním předpokladem je zachování intenzity (rovnice 2.3).
Využívají diference různých řádů a získané diferenciální rovnice slouží jako aproximace předpokladu zachování intenzity. Odtud ovšem plynou chyby, které se snažíme minimalizovat. Dále se objevuje problém apertury. Ten nastane, pokud sledovaný bod leží např. na hraně. Z důvodu malého integračního okna není možné přesně určit směr pohybu. Z



Obrázek 2.1: Aperturní problém[8]

tohoto důvodu se dodá další chybový člen či další omezující předpoklady. Díky těmto doplňkům se snižuje vliv aperturního problému, ale zároveň se zvyšuje složitost numerického řešení. Diferenční metody mají tedy za úkol najít co nejmenší celkovou chybu takovou, aby byla kompromisem mezi splněním předpokladu zachování intenzity, celkovou výpočtovou náročností a zároveň aby vyhovovala reálným omezením, které na optický tok klademe. Přesnost výpočtu nám také stoupá při použití diferencí vyšších řádů, nicméně to má za následek zvýšenou citlivost např. na obrazový šum, a ne vždy se jejich využití vyplatí. Pro bližší pohled na problematiku je vhodná literatura [5].

Vyhledávání oblastí

Tato metoda využívá korelačních funkcí, které dokáží určit, do jaké míry jsou dva různé objekty podobné. Jde tedy o míru podobnosti mezi obrazci. Optický tok $\mathbf{d} = (d_x, d_y, t)$ si můžeme představit jako nějakou transformační funkci, která určuje pomocí transformace v čase $t - \delta t$ obraz v čase t. Pokud tedy budeme znát odhad optického toku, můžeme zjistit přesnost mezi dvěma obrazy $I(\mathbf{x} - \mathbf{d}\delta t, t - \delta t)$ a $J(\mathbf{x}, t)$. Úkolem tedy zůstává vybrat z prostoru možných optický toků ten, který má největší korelační přesnost. Jako korelační funkce ϵ se používá například součet rozdílů čtverců Nechť ω_x a ω_y jsou celá čísla. Definujeme okolí bodu jako $(2\omega_x + 1) \times (2\omega_y + 1)$. Tento obdélník je někdy nazýván integrační okno. Typické hodnoty pro ω_x a ω_y jsou 2,3,4,5,6,7 pixelů. Nejvhodnějším optickým tokem je vektor, který minimalizuje zadanou korelační funkci ϵ.

$$\epsilon(\mathbf{d}) = \epsilon(d_x, d_y) = \sum_{x=u_x-\omega_x}^{u_x+\omega_x} \sum_{y=u_y-\omega_y}^{u_y+\omega_y} (I(x, y) - J(x + d_x, y + d_y))^2 \tag{2.3}$$

Tato minimalizace se provádí právě na výše definovaném integračním okně a minimalizační vektor je náš hledaný optický tok.
Jak je vidět, tento způsob výpočtu optického toku je řešen "hrubou silou", a jelikož prověřuje všechny možnosti, je velmi numericky náročný, ale na druhou stranu univerzální a přesný. Pokročilejší metody jako třeba iterační Lucas-Kanade algoritmus se již soustředí na vymezené prostory, např. zjemňováním výpočtu. Toto zjemňování je realizováno pomocí tzv. pyramidové reprezentace obrazu a sledovacího algoritmu. Těmto věcem se detailně věnuje kapitola 4.

Další metody budou popsány jen velmi stručně, na konci odstavců se nachází odkazy na detailnější literaturu.

Metody založené na energii

Tyto metody jsou tvořeny sadami filtrů, které určují optický tok pomocí odezvy jednotlivých filtrů pro jednotlivé pixely. Využívají taktéž Fourierovu transformaci. Příklady jsou uvedeny [5].

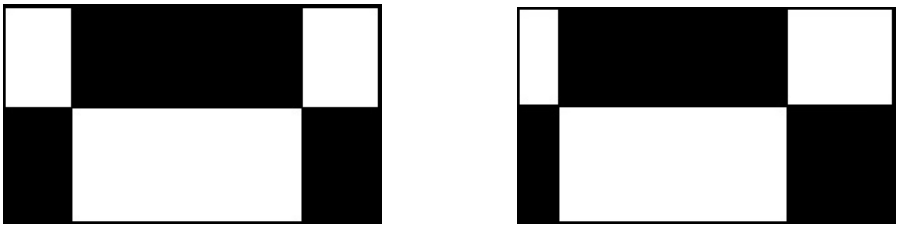
Metody založené na fázi

Stejně jako u předešlé metody jsou využity vlastnosti Fourierova obrazu. Využívá se fakt, že posuvy v doméně prostorové a frekvenční jsou spolu spjaty. Využívá se tedy gradientu fáze, nikoli intenzity, což je v mnoha případech vhodnější a stabilnější způsob výpočtu optického toku. Této metodě je věnována literatura [5].

Kapitola 3

Vyhledávání význačných bodů

Tato kapitola je věnována vyhledávání význačných bodů. Při vyhledávání optického toku je na vstupu vyžadována množina pixelů jednoznačně definovaných dvěma souřadnicemi. První přirozenou možností je hledání optického toku pro všechny body obrazu. Představme si ale obraz velikosti n_x a n_y. Celkový počet pixelů je součin těchto čísel, a jelikož dnešní kamery pracují již ve vysokých rozlišeních, dostáváme řádově statisíce až miliony bodů. To je pro numerický výpočet velmi náročné a proto se tento způsob většinou nepoužívá. Místo toho vznikly metody, které dokáží na daném obraze vybrat nižší počet pixelů, který je dostačující pro další práci s obrazem. Tyto body se nevybírají nahodile, ale naopak se vyhledávají sofistikovanými algorimy, aby jejich kvalita byla co nejvyšší. Pro trasovací algoritmy jsou totiž nejatraktivnější právě ty body, které lze lehce identifikovat. Představme si situaci zobrazenou na obrázku 3.1.



(a) (b)

Obrázek 3.1: Dva po sobě jdoucí obrázky videa

Pokud by tyto obrázky byly po sobě jdoucí ve zjednodušené videosekvenci, lze na nich ukázat, jaké body algoritmus vyhledává. Pro jednoduchost bude sledován jeden bod z celého obrázku. Pokud by byl vybrán jeden čistě náhodný, může se stát, že to bude jeden z bodů uprostřed černé plochy. Tento bod je ale poté zpětně po posunu obrazu neidentifikovatelný. Mnohem zajímavější je bod na špici černého obdélníku. Po posuvu jej lze opět lehce identifikovat a lze tedy vypočítat jeho optický tok. Mezi algoritmy, které se pro hledání těchto bodů používají, se řadí následující.

3.1 Harrisův operátor

Harrisův operátor vychází z Moravcova detektoru a cílem je odstranit jeho nedostatky. Dokáže na obrázku rozlišit body na hranách a rohy, což jsou body, kde se spojují dvě hrany, tedy směr hrany se v podstatě mění.

Obrázek 3.2: Typický testovací obrázek s hranami a rohy[7]

Nechť I je jasová funkce, dále část obrazová funkce $w(u,v)$ nazývané okno, ležící na souřadnicích (u,v) s posuvy x v x-ové souřadnici a y v y-ové souřadnici. Součet rozdílů čtverců je

$$S(x,y) = \sum_u \sum_v w(u,v)(I(u+x, v+y) - I(u,v))^2 \tag{3.1}$$

Tento součet lze aproximovat $I(u+x, v+y)$ Taylorovým rozvojem. Parciální derivace jsou značeny jako I_x a I_y. Výsledkem je

$$I(u+x, v+y) \approx I(u,v) + I_x(u,v)x + I_y(u,v)y \tag{3.2}$$

a dosazením do původní rovnice se tvar změní na

$$S(x,y) \approx \sum_u \sum_v w(u,v)(I_x(u,v)x + I_y(u,v)y)^2 \tag{3.3}$$

což je v maticovém zápisu

$$S(x,y) \approx \begin{pmatrix} x & y \end{pmatrix} A \begin{pmatrix} x \\ y \end{pmatrix}$$

kde matice A je

$$A = \sum_u \sum_v w(u,v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}.$$

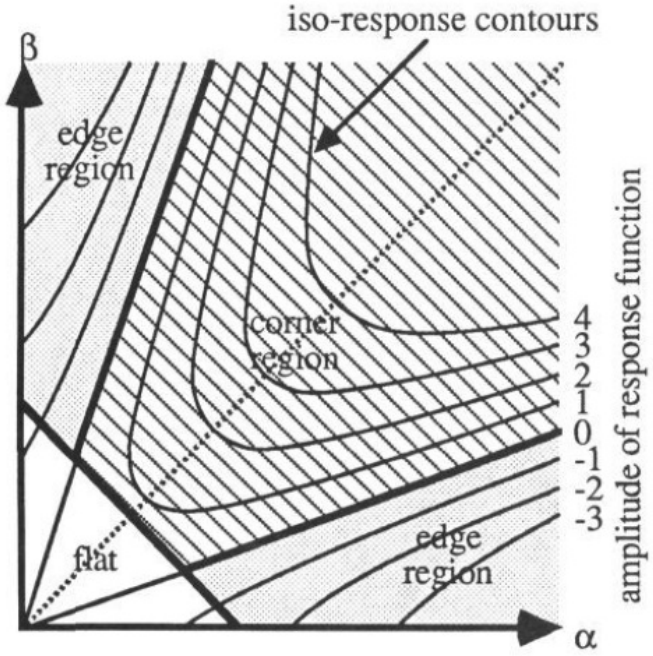
Vlastní čísla této matice λ_1 , λ_2 určí typ význačného bodu a jeho významnost. Mohou nastat tři možnosti podle obrázku 3.3.

- pokud $\lambda_1 \approx 0$ a $\lambda_2 \approx 0$, pak pixel (x,y) není význačným bodem
- pokud $\lambda_1 \approx 0$ a λ_2 má velkou kladnou hodnotu, leží pixel na hraně
- pokud λ_1 i λ_2 mají velkou kladnou hodnotu, pixel je rohem.

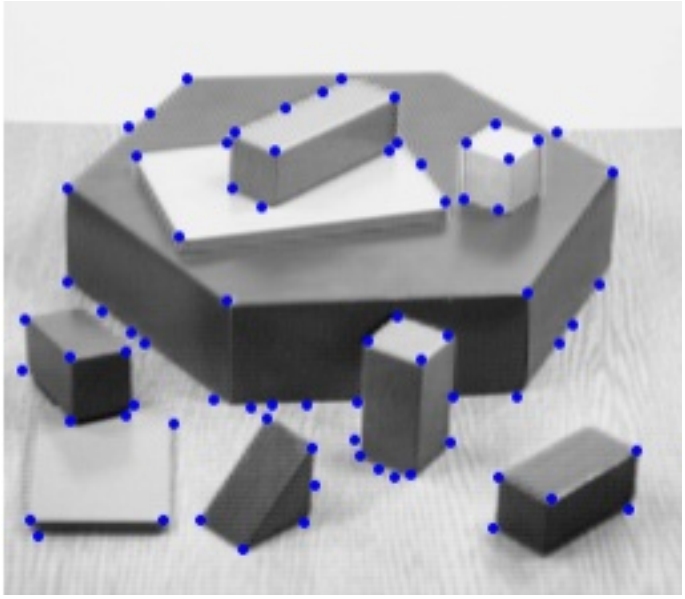
Jako kritérium pro rozhodnutí mezi těmito třemi podmínkami se počítá indikátor přítomnosti rohu R:

$$R = det(A) - k * tr^2(A)$$

kde funkce *det* značí determinant matice, funkce *tr* je stopa matice, tedy suma hodnot na diagonále, a k je empiricky získaná hodnota pohybující se podle literatury mezi 0,04-0,15, přičemž nejčastěji používaná hodnota je 0,04.
Pokud je R kladné, jedná se o roh, záporná hodnota ukazuje na hranu a hodnoty blízké nule značí plochu o přibližně konstantní intenzitě[6].



Obrázek 3.3: Oblasti hran a rohů s vlastními čísly α a β[10]



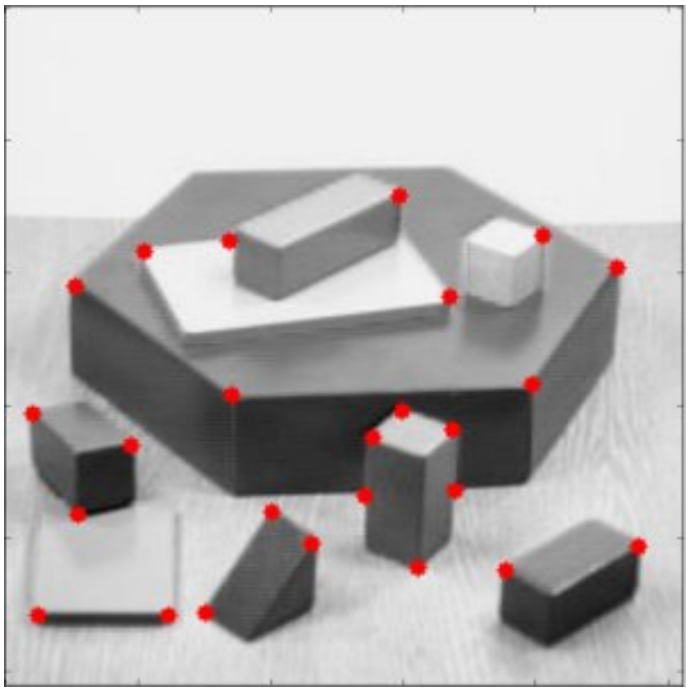
Obrázek 3.4: Body nalezené Harrisovým operátorem[6]

3.2 Good Features to track

Algoritmus Good Features To Track byl představen v roce 1994, přičemž si kladl za cíl nalezení dostatečného množství co nejvýraznějších bodů vhodných ke sledování pohybu mezi dvěma obrázky. Výhodou algoritmu je jeho schopnost nalezení těchto bodů bez předchozích znalostí jejich vlastností. Jeho základem je Harrisův operátor (viz 3.1), ale na rozdíl od něj dokáže určit pořadí vhodnosti ke sledování. Tedy seřadí nalezené body podle výraznosti a výstupem je pouze námi daný počet nejlepších bodů. Zjednodušuje také oproti Harrisovu operátoru vzorec pro výpočet indikátoru přítomnosti rohů. V tomto případě platí

$$R = min(\lambda_1, \lambda_2)$$

Tedy hodnota indikátoru je rovna menšímu z vlastních čísel. Důvod je jednoduchý. Pokud je menší z hodnot dostatečně velká, abychom daný bod mohli označit za význačný bod, vyšší hodnota tuto podmínku musí splňovat také. Takto se postupuje po celém obrázku a každému pixelu je přiřazena hodnota R. Poté se podle koeficientu kvality vyberou body s vyšší hodnotou R, než je hodnota mezní. Ta se získá poměrem k nejlepšímu bodu (tedy s nejvyšší hodnotou R). Pokud má koeficient hodnotu c, pak mezní hodnota $R_{mezni} = c * R_{max}$. Dalším omezením je zadaný počet bodů N_{max}. Odstraní se tedy případné přebytečné body a posledním krokem je odstranění jednoho z bodů, jejichž vzájemná vzdálenost je menší než zadaná minimální vzdálenost. V tomto případě se samozřejmě odstraní bod s nižší hodnotou R. Výstupem je tedy množina bodů s nejvyššími hodnotami R, přičemž splňuje námi zadané parametry [3].



Obrázek 3.5: Body nalezené metodou Good Features To Track[9]

Kapitola 4

Algoritmus Lucas - Kanade

V této kapitole bude detailně rozebrán algoritmus Lucas Kanade. Nejdříve bude vysvětleno, jak lze pyramidově chápat obraz a jak toho lze využít v algoritmu Lucas Kanade.

4.1 Pyramidová reprezentace obrazu

Nyní je tedy třeba říci, co je myšleno pyramidovou reprezentací obrazu. Nechť I je obraz velikosti $n_x \times n_y$. $I^0 = I$ značí "nultou" úroveň obrazu, má tedy maximální rozlišení. Obrazová šířka je rovna $n_x^0 = n_x$ a výška je rovna $n_y^0 = n_y$. Další úrovně se získávají rekurzivně. Tedy získáme postupně I^0, I^1, I^2, I^3... Nechť tedy $L = 1, 2, 3...$ je pyramidová úroveň a I^{L-1} je obraz úrovně $L-1$. Šířku a výšku označíme n_x^{L-1} a n_y^{L-1}. Obraz I^{L-1} je poté definován jako

$$I^L(x,y) = \frac{1}{4}I^{L-1}(2x,2y)$$
$$+\frac{1}{8}[I^{L-1}(2x-1,2y) + I^{L-1}(2x+1,2y) + I^{L-1}(2x,2y-1) + I^{L-1}(2x,2y+1)]$$
$$+\frac{1}{16}[I^{L-1}(2x-1,2-1) + I^{L-1}(2x+1,2y+1) + I^{L-1}(2x+1,2y-1) + I^{L-1}(2x-1,2y+1)] \tag{4.1}$$

Odstín každého bodu obrazu I^L je vypočten z obrazu I^{L-1} podle vzorce 4.1. Může ale nastat problém. Pro krajní body obrazu neexistují všechny okolní body potřebné pro výpočet. Proto se těmto neexistujícím bodům přiřadí následující hodnoty

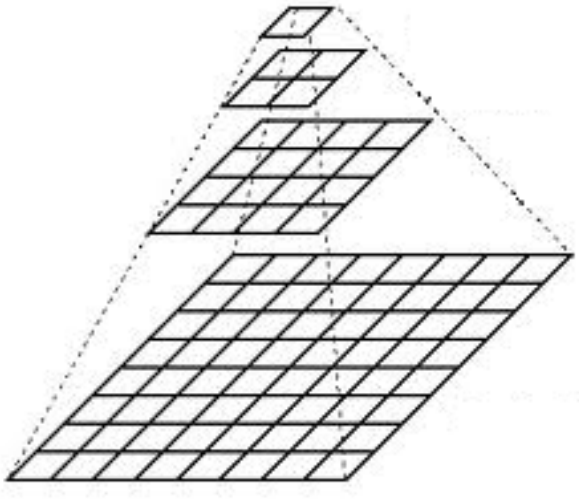
$$I^{L-1}(-1,y) \cong I^{L-1}(0,y)$$
$$I^{L-1}(x,-1) \cong I^{L-1}(x,0)$$
$$I^{L-1}(n_x^{L-1},y) \cong I^{L-1}(n_x^{L-1}-1,y)$$
$$I^{L-1}(x,n_y^{L-1}) \cong I^{L-1}(x,n_y^{L-1}-1)$$
$$I^{L-1}(n_x^{L-1},n_y^{L-1}) \cong I^{L-1}(n_x^{L-1}-1,n_y^{L-1}-1)$$

Takto se získá rovnice správně definovaná pro všechny body $0 \leq 2x \leq n_x^{L-1}-1$, $0 \leq 2y \leq n_y^{L-1}-1$. Z těchto vzorců lze mimo jiné vyčíst fakt, že šířka n_x^L a výška n_y^L obrazu I^L jsou největší celá čísla splňující

$$n_x^L \leq \frac{n_x^{L-1}+1}{2} \tag{4.2}$$

$$n_y^L \leq \frac{n_y^{L-1} + 1}{2} \tag{4.3}$$

Rovnice 4.1,4.2 a 4.3 jsou dostačující je kompletní rekurzivní konstrukci obrazů $\{I^L\}_{L=0,1,...,L_m}$ a $\{J^L\}_{L=0,1,...,L_m}$. Hodnota L_m je nazývána hloubkou pyramidy a číselně je definována dle potřeby, přičemž při jejím definování je nutné brát ohled na předpokládanou velikost optického toku. V praxi se většinou využívají pouze hodnoty 2,3,4. Pro obrazy typických



Obrázek 4.1: Pyramidová reprezentace obrazu

velikostí nemá smysl jít níže než je hodnota 4. Například pro obraz I velikosti 640x480 obrazových bodů mají obrazy I^1, I^2, I^3 a I^4 velikosti 320x240, 160x120, 80x60 a 40x30. Pokračovat na úroveň 5 již nemá smysl, jelikož rozlišení by bylo velmi nízké. Velkou výhodou pyramidové reprezentace obrazu je to, že můžeme zachytit velké posuvy pixelu, a to mnohem větší, než je velikost definovaného integračního okna. Výsledkem je mnohem menší časová náročnost výpočtu[4].

4.2 Pyramidová implementace sledovacího algoritmu

Cílem sledování je pro daný vektor $\mathbf{u}$ na obrázku I najít související vektor $\mathbf{v} = \mathbf{u} + \mathbf{d}$ na obrázku J. Pro $L = 0, ..., L_m$ platí $\mathbf{u}^L = (u_x^L, u_y^L)$, což jsou upravené souřadnice vektoru $\mathbf{u}$ na pyramidových obrázcích I^L. Z rovnic 4.1,4.2 a 4.3 vyplývá vztah pro výpočet vektoru

$$\mathbf{u}^L = \frac{\mathbf{u}}{2^L} \tag{4.4}$$

Operace dělení je v tomto vztahu aplikována nezávisle pro obě souřadnice, tedy po složkách vektoru. Algoritmus probíhá následovně. Nejprve je optický tok vypočítán pro nejnižší úroveň pyramidy L_m. Výsledek je využit na úrovni $L_m - 1$ jako aktuální odhad posunu. Poté je optický tok vypočítán na úrovni $L_m - 1$ a výsledek je opět použit jako odhad na vyšší úrovni. Tímto způsobem se pokračuje až k $L = 0$, tedy k originálnímu obrázku. Samotnou rekurzivní operace probíhá následovně. Nechť existuje odhad na úrovni L, který označíme $\mathbf{g}^L = (g_x^L, g_y^L)$ a je získán z výpočtu optického toku na úrovni $L + 1$. Nyní je třeba vypočítat optický tok na úrovni L. Výpočet se provádí nalezením vektoru

$\mathbf{d}^L = (d_x^L, d_y^L)$, který minimalizuje chybovou funkci ϵ^L

$$\epsilon^L(\mathbf{d}^L) = \epsilon^L(d_x^L, d_y^L) = \sum_{x=u_x^L-\omega_x}^{u_x^L+\omega_x} \sum_{y=u_y^L-\omega_y}^{u_y^L+\omega_y} (I^L(x,y) - J^L(x+g_x^L+d_x^L, y+g_y^L+d_y^L))^2 \quad (4.5)$$

Vidíme také, že integrační okno zůstává stejně velké pro všechny úrovně L. Získaný optický tok $\mathbf{d}^L$ se využije pro vytvoření odhadu pro úroveň $L-1$ vztahem

$$\mathbf{g}^{L-1} = 2(\mathbf{g}^L + \mathbf{d}^L)$$

Tento odhad se využije pro nalezení vektoru $\mathbf{d}^{L-1}$, který minimalizuje funkce ϵ^{L-1}, což se opakuje až po dosažení úrovně $L = 0$. Poznamenejme, že počáteční odhad $\mathbf{g}^{L_m} = (0,0)$. Vztah pro výpočet výsledného optického toku je tedy

$$\mathbf{d} = \mathbf{g}^0 + \mathbf{d}^0, \quad (4.6)$$

což lze také vyjádřit součtem optických toků na všech úrovních pyramidového obrazu, ovšem s koeficientem převodu 2^L. Výsledkem je tedy

$$\mathbf{d} = \sum_{L=0}^{L_m} 2^L \mathbf{d}^L \quad (4.7)$$

Velkou výhodou pyramidové implementace je fakt, že každý reziduální vektor $\mathbf{d}^L$ je ve výsledku velmi malý v porovnání s výsledným optickým tokem, který může být obecně velmi velký. Pokud by byl každý elementární krok výpočtu optického toku omezen velikostí d_max, pak celkový pohyb pixelu, který je dosažitelný pyramidovou implementací sledovacího algoritmu, je

$$d_{max,final} = (2^{L_m+1} - 1)d_{max} \quad (4.8)$$

Z této rovnice vyplývá, že např. již pro hloubku $L_m = 2$ dostaneme posun sedmkrát větší než při použití klasického ("nepyramidového") algoritmu Lucas Kanade[4].

4.3 Matematický popis iteračního algoritmu

Stále zde chybí popis jádra algoritmu, tedy samotného iteračního výpočtu optického toku.
Jelikož pro každou úroveň je proces stejný, v tomto odstavci bude vynechán index L. Definujme si nové obrázky A a B.

$$\forall(x,y) \in (p_x - \omega_x - 1, p_x + \omega_x + 1) \times (p_y - \omega_y - 1, p_y + \omega_y + 1) \quad (4.9)$$

$$, A(x,y) \doteq I^L(x,y)$$

$$\forall(x,y) \in (p_x - \omega_x, p_x + \omega_x) \times (p_y - \omega_y, p_y + \omega_y) \quad (4.10)$$

$$, B(x,y) \doteq J^L(x + g_x^L, y + g_y^L)$$

Pro výpočet vektoru $\mathbf{d}$ platí, že v ideálním případě bude derivace ϵ nulová, tedy

$$\frac{\partial \epsilon(\mathbf{d})}{\partial(\mathbf{d})} = (0,0), \quad (4.11)$$

jejíž rozepsání je

$$\frac{\partial \epsilon(\mathbf{d})}{\partial \mathbf{d}} = -2 \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} (A(x,y) - B(x+d_x, y+d_y)) \left(\frac{\partial B}{\partial x}, \frac{\partial B}{\partial y} \right)$$

Další úpravou (odkaz literatura.....) se tvar mění na

$$\frac{1}{2} \left[\frac{\partial \epsilon(\mathbf{d})}{\partial (\mathbf{d})} \right]^T \approx G\mathbf{d} - \mathbf{b} \tag{4.12}$$

z čehož lze s ohledem na (4.11) vyčíst

$$\mathbf{d}_{opt} = G^{-1}\mathbf{b}, \tag{4.13}$$

kde

$$G \doteq \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{4.14}$$

$$\mathbf{b} \doteq \sum_{\mathbf{x}=\mathbf{p_x}-\omega_\mathbf{x}}^{\mathbf{p_x}+\omega_\mathbf{x}} \sum_{\mathbf{y}=\mathbf{p_y}-\omega_\mathbf{y}}^{\mathbf{p_y}+\omega_\mathbf{y}} \begin{bmatrix} \delta I I_x \\ \delta I I_y \end{bmatrix} \tag{4.15}$$

přičemž δI a parciální derivace I_x, I_y jsou definovány takto:

$$\delta I = A(x,y) - B(x,y) \tag{4.16}$$

$$I_x(x,y) = \frac{\partial A(x,y)}{\partial x} = \frac{A(x+1,y) - A(x-1,y)}{2} \tag{4.17}$$

$$I_y(x,y) = \frac{\partial A(x,y)}{\partial y} = \frac{A(x,y+1) - A(x,y-1)}{2} \tag{4.18}$$

Rovnice (4.13) je standardní Lucas-Kanadeho rovnice pro výpočet optického toku. Je ale platná pouze pro malé posuvy, proto je nezbytné v praxi tento postup několikrát iterovat. Iterační index bude značen k, na počátku má hodnotu 1. Nechť proběhly předchozí výpočty a z iterací 1,2,...,k - 1 získaly odhad $\mathbf{d}^{k-1} = (d_x^{k-1}, d_y^{k-1})$ pro posun $\mathbf{d}$. Nově získaný obraz s přihlédnutím na odhad $\mathbf{g}^{k-1}$ je značem B_k a platí:

$$\forall (x,y) \in (p_x - \omega_x, p_x + \omega_x) \times (p_y - \omega_y, p_y + \omega_y), B_k(x,y) = B(x + g_x^{k-1}, y + g_y^{k-1}) \tag{4.19}$$

Cílem je tedy vypočítat vektor posunu $\mathbf{d}^k$, který minimalizuje chybovou funkci

$$\epsilon^k(\mathbf{d}^k) = \epsilon^k(d_x^k, d_y^k) \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} (A(x,y) - B_k(x + d_x^k, y + d_y^k))^2 \tag{4.20}$$

Tato minimalizace může být vypočítána pomocí standardní Lucas-Kanadeho rovnice (4.13), do které přidáme iterační index k

$$\mathbf{d}^k = G^{-1}\mathbf{b}^k \tag{4.21}$$

,kde je vektor $\mathbf{b}_k$ definován jako

$$\mathbf{b}_k \doteq \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \begin{bmatrix} \delta I_k I_x \\ \delta I_k I_y \end{bmatrix} \tag{4.22}$$

a diference k-tého obrázku δI_k je rovna

$$\delta I_k(x,y) = A(x,y) - B_k(x,y), \forall (x,y) \in (p_x - \omega_x, p_x + \omega_x) \times (p_y - \omega_y, p_y + \omega_y) \quad (4.23)$$

Důležité je poznamenat, že parciální derivace I_x a I_y i matice G se počítají pouze jednou na počátku algoritmu a přes celý proces se nemění. Je tedy nutné při každé iteraci k přepočítat pouze vektor $\mathbf{b}^k$ a obrazovou diferenci δI^k, což velmi šetří výpočetní čas.
Jakmile je vypočítán pomocí rovnice (4.21) optický tok $\mathbf{d}^k$, vypočítá se aktuální odhad pro další iteraci $k+1$

$$\mathbf{g}^k = \mathbf{g}^{k-1} + \mathbf{d}^k \quad (4.24)$$

Iterační proces pokračuje, dokud nedosáhne zadaného počtu iterací, nebo vypočítaný optický tok $\mathbf{d}^k$ je menší než tolerance. Nechť celý proces proběhl v K iteracích. Pak výsledný optický tok je roven

$$\mathbf{d} = \mathbf{g}^K = \sum_{k=1}^{K} \mathbf{d}^K \quad (4.25)$$

Tento vektor minimalizuje chybovou funkci ϵ popsanou již mnohokrát výše[4].

Kapitola 5

Výškoměr

Hlavním úkolem navrhovaného softwaru je pomocí výše zmíněných metod zjistit výšku snímacího členu nad povrchem. Jako snímací člen slouží běžná kamera nebo webkamera. V prvním, jednodušším, případě se natočí video a zpětně se analyzuje na počítači. Snímání webkamerou je mnohem složitější, jelikož software musí zpracovávat videosekvenci v reálném čase. Jsou zde tedy mnohem větší nároky na výpočtovou náročnost, aby se obraz analyzoval plynule a bez zpoždění. Výškoměr využívá výše zmíněné metody pro analýzu obrazu.

5.1 Princip funkce

Hlavní myšlenkou je rozdílnost rychlosti pohybu bodu při různých vzdálenostech od snímacího členu. Je to stejné, jako když jede člověk v autě a dívá se bočním okénkem ven. Stromy kolem silnice ubíhají velmi rychle, že je ani téměř nepostřehne. Naopak stromy ve větší vzdálenosti se pohybují o poznání pomaleji. Vše je samozřejmě relativní k rychlosti, pokud se bude auto pohybovat minimální rychlosti, i stromy u silnice se budou pohybovat pomalu. Je tedy zřejmé, že bude potřeba znát nějakou výchozí podmínku pro určení zbylých veličin, což bude v tomto případě rychlost pohybující se kamery.

5.2 Využití výškoměru

Výsledný software se dá použít pro určení vzdálenosti od libovolné ploché překážky, pokud se snímací člen pohybuje podél této překážky. Lze to tedy použít jako výškoměr pro létající modely nebo na určování vzdálenosti pozemního modelu ode zdi, pro vyhnutí se srážce apod. Důležitou podmínkou je ale, aby model ,který nese snímací člen, byl schopen určit svoji okamžitou rychlost, popřípadě aby dokázal udržovat předem zadanou konstantní rychlost.

Kapitola 6

Realizace softwaru

6.1 EmguCV a jeho metody

Pomůckou při tvorbě softwaru bude knihovna EmguCV. Je to volně šiřitelná knihovna pro práci s videem. Vychází z knihovny OpenCV, jejíž je wrapper pro programovací jazyk C# .OpenCV je totiž naprogramované pro C++ a Javu. EmguCV obsahuje všechny dříve zmíněné algoritmy a metody v optimalizovaném tvaru, proto jsou ideální pro využití v navrhovaném softwaru, a díky tomu tyto metody spotřebují minimum výpočetní kapacity procesoru. V celém textu jsou využity pojmy z prostředí objektového programování. Pro nalezení význačných bodů je použit již výše popsaný algoritmus, v EmguCV je definován pod jménem GoodFeaturesToTrack. Parametry jsou vysvětleny níže a jejich použití je podle kapitoly 3.2

Good Features To Track

public Point2D<float>[ ][ ] **GoodFeaturesToTrack**(
int **maxFeaturesPerChannel**,
double **qualityLevel**,
double **minDistance**,
int **blockSize**)

maxFeaturesPerChannel	maximální počet význačných bodů
qualityLevel	specifikuje nejnižší povolenou kvalitu bodů
minDistance	nejmenší vzájemná euklidovská vzdálenost mezi jednotlivými body
blockSize	velikost porovnávacího okna

Dále je využit Lucas Kanadeho pyramidový interační algoritmus pro nalezení posunu význačného bodu. Ten je v EmguCV naprogramován jako součást třídy OpticalFlow. Parametry jsou použity podle kapitoly 4.

Pyramidal Lucas Kanade algorithm

public static void **PyrLK**(
Image<Gray, byte> **prev**,
Image<Gray, byte> **curr**,
PointF[ ] **prevFeatures**,
Size **winSize**,
int **level**,
MCvTermCriteria **criteria**,
out PointF[ ] **currFeatures**,
out byte[ ] **status**,
out float[ ] **trackError**)

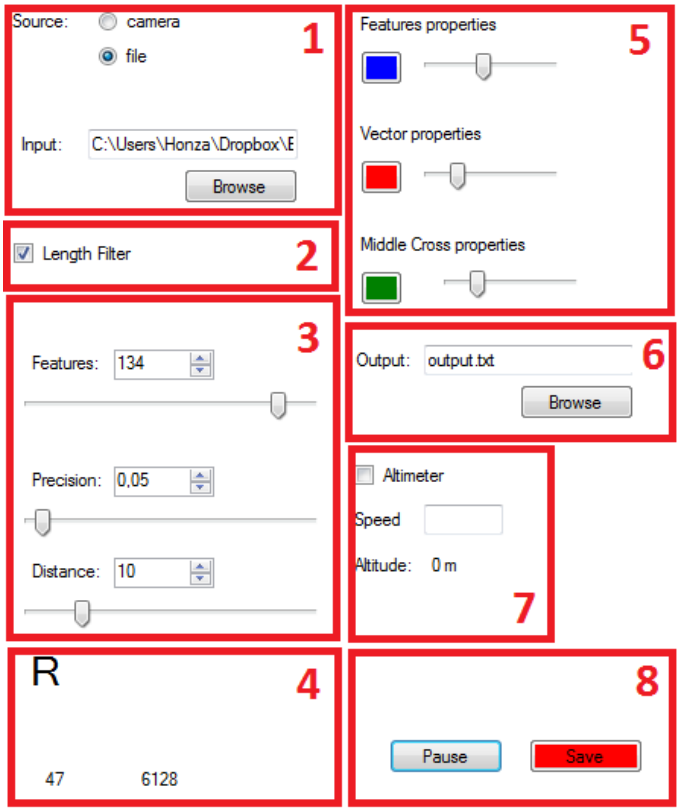
prev	první šedotónový obrázek (v čase t)
curr	druhý šedotónový obrázek (v čase t + δt)
prevFeatures	pole bodů, které chceme sledovat
winSize	velikost integračního okna
level	max. pyramidová úroveň (0 značí nevyužití pyramid)
criteria	specifikuje, za jakých podmínek se ukončí iterační proces (počet iterací, přesnost)
currFeatures	výstup, nové body nalezené trasovacím algoritmem
status	výstup, pole jedniček a nul, obsahuje informace o tom, které body se podařilo trasovat
trackError	čísla pole vyjadřují změnu okolí mezi originálními a posunutými body

6.2 Popis grafického rozhraní

Software je naprogramován v programovacím jazyce C# s využitím knihovny EmguCV a .NET frameworku. Maximálně je při tom využit objektový potenciál jazyka, proto každá významná část programu má svoji třídu. Jako vývojové prostředí posloužilo *Microsoft Visual Studio*, které je v distribuci *Express* poskytováno zdarma. Vizuálně je software řešen v rozhraní Windows Forms.

Na obrázku 6.1 je zobrazeno ovládací rozhraní softwaru. Na obrázku je označeno několik důležitých oblastí a zde je jejich popis.

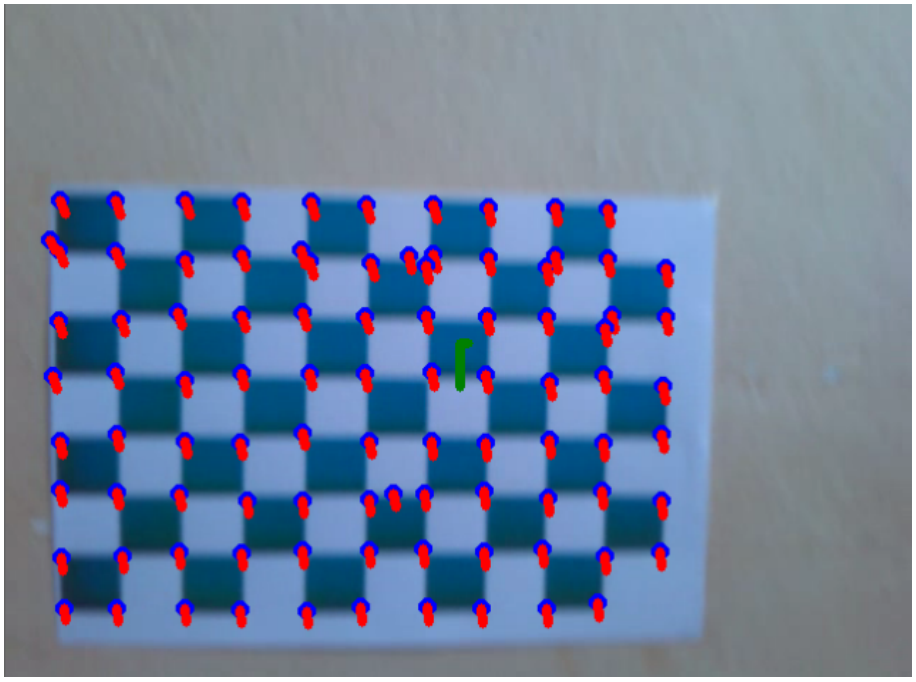
1) Sekce upravuje zdroj videa. Jsou zde na výběr dvě možnosti, webkamera a soubor na disku. V kolonce Input je aktuální cesta k souboru, tlačítkem Browse můžeme soubor lehce vybrat. Pokud je aktivní režim webkamery, objeví se výběr čísla kamery. Například u notebooků s vestavěnou kamerou platí, že tato kamera má číslo 0, jiné kamery, např. připojené přes USB mají čísla vyšší.
2) Zde se zatrhnutím aktivuje délkový filtr. Více v sekci 6.4
3) Zde lze měnit parametry metody GoodFeaturesToTrack, přesněji počet bodů, kvalitu a minimální vzdálenost. Vše lze měnit i při probíhajícím zpracovávání videa.
4) Velké písmeno R představuje odhad typu pohybu. Dokáže rozlišit mezi translačním(T) a rotačním(R). Čísla níže představují aktuální počet sledovaných bodů a celkový



Obrázek 6.1: Ovládací rozhraní softwaru

počet bodů odstraněných délkovým filtrem.
5) Dále lze také měnit barvu a velikost vykreslovaných bodů, vektorů a hlavních výchylek.
6) Toto textové pole je pro zadání cesty k výstupnímu textovému souboru pro výpis zpracovaných dat. Tomuto je věnována sekce 6.5. Tlačítkem Browse se dá opět lehce soubor vybrat.
7) Zde je možné zapnout výškoměr. jak bylo řečeno v kapitole 5, je nutné znát aktuální rychlost snímacího členu. Jelikož je předpokladem konstantní rychlost, zde je nutné tuto rychlost uvést.
8) V poslední části je tlačítko Start/Pause pro spuštění/zastavení práce softwaru. Nakonec po stisknutí červeného tlačítka začne ukládání dat do výstupního textového souboru.

Další částí okenní aplikace je výstup pro video. Ten je zobrazen na obrázku 4.2. Zobrazují se zde tři výstupy. Modrou barvou jsou vyznačeny nalezené význačné body (viz kap. 3). Červenou barvou jsou vyznačeny optické toky těchto bodů, přičemž výchozí význačný bod je počátečním bodem vektoru. Uprostřed se zobrazuje hlavní (průměrná) výchylka a je odděleně v x-ové a y-ové ose. Platí, že tato výchylka určuje reálný posun kamery, ne obrazu. Tedy ve vyobrazeném případě se kamera posunuje dolů a vzorová šachovnice vůči kameře nahoru.



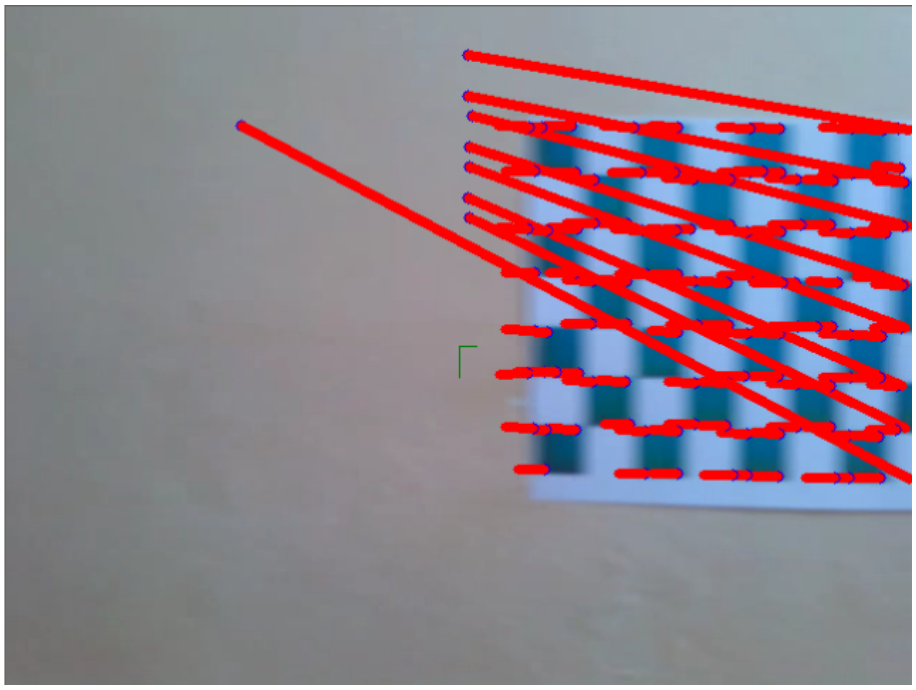
Obrázek 6.2: Videovýstup softwaru

6.3 Vstupy

Jak už bylo řečeno, vstupní video je možné získat ze dvou zdrojů. O načtení videosouborů či videa z realtimové kamery se stará třída *Capture*, která je součástí knihovny EmguCV. Ta poté pomocí metody *QueryFrame* získá jeden ze snímků videa. Pokud používáme jako zdroj webkameru, pak načte obraz, který kamera vidí v okamžiku odeslání požadavku. V opačném případě načte snímek, který následuje v pořadí.

6.4 Zpracování videa, filtry a ukládání

V následujícím odstavci bude vysvětleno jádro softwaru, tedy zpracování videa. To je řešeno pomocí vytvořené třídy *ImgProcess* a její metody *ProcessImage*. Tato metoda je volána pokaždé, když se načte nový obrázek videa. Nejprve nalezne pomocí metody *GoodFeaturesToTrack()* význačné body na předchozím obrázku (s parametry zadanými v grafickém rozhraní) a jejich posun metodou *OpticalFlow.PyrLK()*. Hlavními výstupy těchto metod jsou tři pole (podle odstavce 6.1 - *prevFeatures*, *currFeatures* a *status*). Pro jednodušší práci jsou tato pole převedena na generické listy. Dalším krokem je odstranění bodů, které se nepovedlo sledovat. Tato informace je v poli *status*, kde se na problematických indexech vyskytují nuly. Body z těmito indexy jsou tedy odstraněny z obou kolekcí bodů. I když se některé body povedlo "spárovat", mohou nastat určité problémy a výsledné vektory jsou nesmyslně velké či nesmyslně umístěné. Taková situace je zobrazena na obr 6.2 Jde například o situace, kdy význačný bod zmizí ze zabíraného okna. Je samozřejmě nežádoucí ukládat tyto vektory, navíc by zkreslovaly průměrnou délku pohybového vektoru. Pro účely filtrace byla vytvořena třída *Filter*. Předpokladem je fakt, že pohybové vektory budou při translaci přibližně stejně velké, proto stačí vypočítat medián ze všech velikostí a odstranit vektory, které se od mediánu výrazně odlišují, nebo přesněji, které jsou mnohem delší. Medián je použit kvůli tomu, že jeho hodnota není ovlivněna dlouhými chybnými vektory. Oproti tomu aritmetický průměr by byl značně deformován.



Obrázek 6.3: Nalezené posuvy bez délkové filtrace

Jelikož je filtr aktivní i při rotačním pohybu, je nutné ponechat vektory, které jsou výrazně menší než medián(obr. 6.1). Důvodem je fakt, že při rotaci se objevuje široká škála délek vektorů s minimem ve středu otáčení. Výrazně velké vektory se odstraní a malé ponechají. Tyto vektory totiž nejsou zpravidla chybné a při jejich odstranění by se výrazně snížil počet ponechaných vektorů, což není žádoucí.

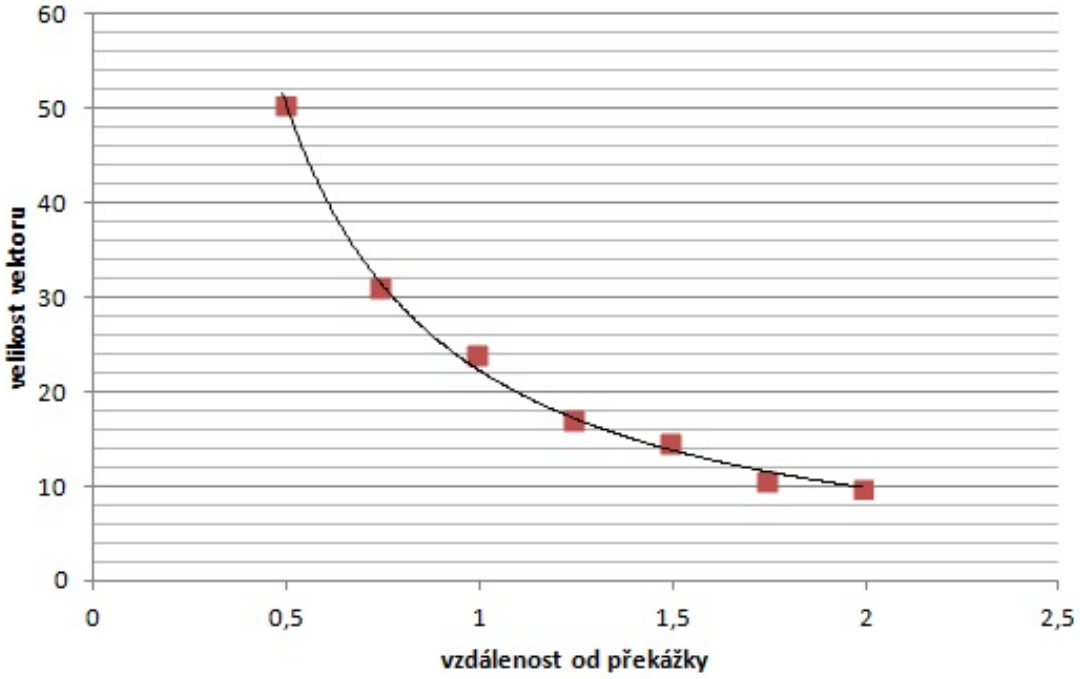
Ukládání dat je řešeno třídou *Data*. Výstupním souborem je textový soubor, odkud se dají hodnoty lehce načíst z libovolného softwaru pro práci s daty, např. Excel nebo Matlab. Přímo v kódu lze nastavit v jakém tvaru se budou vektory ukládat.

6.5 Kalibrace výškoměru

Pro jednoznačné určení výšky je nutné znát dva parametry pohybu. V první řadě aktuální, přesnou rychlost pohybu kamery. Tato nutná znalost je velmi problematická, jelikož je obtížné jednoduchým způsobem určit rychlost pohybu. Plyne to z použití výškoměru pro mobilní roboty, které v naprosté většině případů nenesou zařízení pro zjištění rychlosti. Dalším nutným vstupem je velikost vektorů optického toku získaných z navrhovaného softwaru. Ze znalosti těchto dvou parametrů je možné získat aktuální výšku/vzdálenost od povrchu.
Nejpřesnějším způsobem řešení je simulace snímání plochy při známé rychlosti. Z naměřených dat lze získat data o optickém toku v obrazu. Při dostatečném množství měření je poté možné získat grafy pro konstantní rychlosti a z nich určovat výšky pro dané velikosti optického toku.

Simulace

Pro simulaci měření výšky bylo nutné použít nosič kamery, u kterého je možné co nejpřesnější určení rychlosti. Pro tento účel je využit model buginy, který dokáže udržovat téměř konstantní rychlost a směr jízdy. Na tento model byla přidána kamera pro snímání povrchu s členitým šachovnicovým vzorem pro různé vzdálenosti od překážky. Po sérii měření byla natočená videa zpracována vytvořeným softwarem, získaná data zprůměrována a zakreslena do grafů. Příkladem je graf na obrázku 6.4, kde jsou hodnoty naměřeny při rychlosti 0,668 m/s. Při větším množství naměřených grafů a jejich proložením vhodnou křivkou lze výšku vypočítat pro libovolný bod v naměřené oblasti interpolací. Pro



Obrázek 6.4: Graf závislosti velikosti optického toku na vzdálenosti od překážky při konstantní rychlosti

nízké rychlosti (přibližně do 0,8 m/s) byla získaná data poměrně přesná, ale pro vyšší rychlosti se do výsledku promítaly nežádoucí okolnosti. Hlavním problémem byla doba, než model dosáhl konstantní rychlosti. Po dobu měření se rychlost pohybu lehce zvyšovala. Druhým problémem byly vibrace přenášené z odpružení podvozku přes držák kamery. Tím bylo video také dost negativně ovlivněno.

Pro využití softwaru jako přesného a univerzálního výškoměru je nutná kalibrace pro odpovídající rychlosti a odpovídající výšky. S tím souvisí zajištění dostatečně přesného a stabilního nosiče kamery při vyšších rychlostech pohybu.

Závěr

Předkládaná práce je věnována návrhu výškoměru, který je založen na analýze optického toku v obraze snímaném kamerou. Začátek práce je věnován teoretickému popisu algoritmů. V kapitole 2 jsou popsány metody vhodné pro zpracování optického toku, detailněji jsou popsány metody diferenční a vyhledávání oblastí. Kapitola 3 pojednává o vyhledávání význačných bodů přes Harrisův operátor a metodu GoodFeaturesToTrack. Jádro určování optického toku je Lucas-Kanadeho iterační algoritmus, který je popsán v kapitole 4. Všechny tyto oddíly byly matematicky popsány pro lepší pochopení celé problematiky. Dále byl vytvořen software v programovacím jazyce C# s využitím knihovny EmguCV, který využíval všechny popsané poznatky. Jeho hlavní funkcí bylo nalezení optického toku pro vstupní videosekvenci. Dále dokázal určit směr translačního pohybu kamery a vykreslit jej do grafického výstupu. Ve vstupní videosekvenci také dokázal určit typ pohybu, rotaci a translaci. Výstup softwaru ve tvaru vektorů optického toku byl poté využit pro návrh funkce výškoměru. Byla provedena měření, která měla určit závislost velikosti optického toku a vzdálenosti snímacího členu od překážky. O výsledcích měření pojednává kapitola 6. Pro nízké rychlosti byly nalezeny poměrně přesné mocninné závislostní grafy, ale pro vyšší rychlosti se ukázalo velmi problematické vlastní mechanické připevnění kamery k bugině. Výsledkem práce je tedy návrh systému, na jehož principu může pracovat reálný výškoměr.

Další vývoj softwaru by měl jít směrem rozšíření pracovní oblasti, v ideálním případě získat data pro libovolnou rozumnou rychlost a vzdálenost. Pro tato měření je ovšem nutné obstarat dostatečně přesný a stabilní snímací člen a jeho upevnění na pojezdu.

Literatura

[1] THRUN, Sebastian. Probabilistic robotics. Massachusetts: MIT Press, c2006. ISBN 02-622-0162-3.

[2] CHOSET, Howie. Principles of robot motion: theory, algorithms and implementations. Massachusetts: MIT Press, 2005, 603 s. ISBN 02-620-3327-5.

[3] SHI, Jianbo, Carlo THOMASI. Good Features To Track. In: [online]. [cit. 2014-05-25]. Dostupné z: http://w3.inf.fu-berlin.de/lehre/SS06/SeminarComputerVision/origReport_von_Carlo_Tomasi.pdf

[4] BOUGUET, Jean-Yves. Pyramidal Implementation of the Lucas Kanade Feature Tracker Description of the algorithm. [online]. [cit. 2014-05-25]. Dostupné z: http://robots.stanford.edu/cs223b04/algo_tracking.pdf

[5] BARRON, J. L., D. J. FLEET, S. S. BEAUCHEMIN. Performance of optical flow techniques. International Journal of Computer Vision [online]. 1994, s. 43-77 [cit. 2014-05-25]. DOI: 10.1007/BF01420984. Dostupné z: http://link.springer.com/10.1007/BF01420984

[6] OpenCV Documentation [online]. [cit. 2014-05-26]. Dostupné z: http://docs.opencv.org/

[7] PARKS, Donovan a Jean-Philippe GRAVEL. Corner Detectors. Corner Detectors [online]. [cit. 2014-05-26]. Dostupné z: http://kiwi.cs.dal.ca/ dparks/CornerDetection/index.htm

[8] The Aperture Problem. [online]. [cit. 2014-05-26]. Dostupné z: http://stoomey.wordpress.com/2008/04/18/20/

[9] Shi-Tomasi Corner Detector & Good Features to Track. [online]. [cit. 2014-05-26]. Dostupné z: http://docs.opencv.org/trunk/doc/py_tutorials/py_feature2d/py_shi_tomasi/py_shi_tomasi.html

[10] Interesting windows in the Harris Corner Detector. [online]. [cit. 2014-05-26]. Dostupné z: http://www.aishack.in/2010/04/interesting-windows-in-the-harris-corner-detector/