České vysoké učení technické v Praze

Fakulta elektrotechnická

Katedra kybernetiky

Diplomová práce

Využití principu KLT pro sledování cíle kamerovou hlavicí

Marek Tejc

Květen 2015

Vedoucí práce: Ing. Pavel Krsek, Ph.D.

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Bc. Marek T e j c

Studijní program: Kybernetika a robotika (magisterský)

Obor: Robotika

Název tématu: Využití principu KLT pro sledování cíle kamerovou hlavicí

Pokyny pro vypracování:

1. Seznamte se s principem KLT algoritmu pro sledování objektů.
2. Seznamte se se stávající implementací algoritmu sledování použitému pro kamerovou hlavici.
3. Navrhněte nový algoritmu sledování vybraného objektu založený na principu KLT.
4. Implementujte algoritmus v jazyce C, C++ tak, aby ho bylo možné použít s využitím technických prostředků kamerové hlavice. Kód připravte tak, aby ho bylo možné využít jak v operačním systému Windows tak Linux.
5. Seznamte se s dříve navrženými vylepšeními systému sledování objektů a dle možností je implementujte.
6. Programový kód nezapomeňte dostatečně komentovat.
7. Vše pečlivě zdokumentujte.

Seznam odborné literatury:

[1] Milan Sonka, Vaclav Hlavac, and Roger Boyle. Image Processing, Analysis and Machine Vision. Thomson, 3rd edition, ISBN 978-0-495-08252, 2007.
[2] Simon Baker and Iain Matthews. Lucas-Kanade 20 Years On: A Unifying Framework. International Journal of Computer Vision 56(3), 221–255, 2004.
[3] Baojie Fan, Yingkui Du, Linlin Zhu, Jing Sun, Yandong Tang. A robust template tracking algorithm with weighted active drift correction. Pattern Recognition Letters 32 (2011) 1317–1327.
[4] Iain Matthews, Takahiro Ishikawa, and Simon Baker. The Template Update Problem. In Richard Harvey and Andrew Bangham, editors, Proceedings of the British Machine Conference, pages 65.1-65.10. BMVA Press, September 2003. doi:10.5244/C.17.65.

Vedoucí diplomové práce: Ing. Pavel Krsek, Ph.D.

Platnost zadání: do konce letního semestru 2015/2016

L.S.

doc. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 23. 1. 2015

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 11. 5. 2015

Poděkování

Zde bych chtěl poděkovat všem, kteří mě během tvorby diplomové práce podporovali a motivovali. Především děkuji panu Ing. Pavlu Krskovi Ph.D. za odborné vedení, volný čas ke konzultacím, podporu a vstřícnost.

Dále bych chtěl poděkovat rodině za podporu během dosavadního studia a přátelům za volný čas, rady a podnětné diskuze, které vyřešily spoustu problémů.

Tato práce popisuje výsledky dosažené s finanční podporou TAČR v rámci projektu MAGYSKA (TA02010887).

Abstrakt

Tato diplomová práce se zabývá implementací *KL* sledovacího algoritmu do kamerové hlavice s inerciální stabilizací a obrazovou zpětnou vazbou. *KL* algoritmus byl zvolen pro nahrazení stávajícího algoritmu *SSD* (**S**um of **S**quare **D**ifferences). Algoritmus *KL* má menší výpočetní a časovou náročnost než algoritmus *SSD*, čímž dosáhneme kratšího dopravního zpoždění pro obrazovou zpětnou vazbu.

Implementaci provedeme s využitím funkcí z knihovny *OpenCV* do stávajícího kódu programu systému vizuálního sledování. Testování provedeme na vybraných sekvencích a na reálné kameře.

Stávající algoritmus i implementaci navrženého algoritmu *KL* upravíme tak, aby byla přeložitelná na platformách s operačními systémy Windows i Linux.

Klíčová slova: KL algoritmus, Kanade-Lucas algoritmus, pohyblivá kamera, OpenCV knihovna, knihovna Pthread

Abstract

This thesis propose implementation *KL* tracking algorithm to camera head with inertial stabilization and visual feedback. *KL* algorithm was chosen to replace an existing algorithm *SSD* (**S**um of **S**quare **D**ifferences). The algorithm *KL* has smaller computing and time-consuming requirements than the algorithm *SSD*, thus achieving shorter time delay for visual feedback.

For implementation we will use functions from the library *OpenCV* in the existing program code of visual tracking. We perform testing on selected sequences and a real camera.

Existing algorithms and implementation of the proposed algorithm *KL* adjust so that was translatable platforms with operating systems Windows and Linux.

Keywords: KL algorithm, Kanade-Lucas algorithm, moving camera, OpenCV library, Pthread library

Obsah

Seznam obrázků

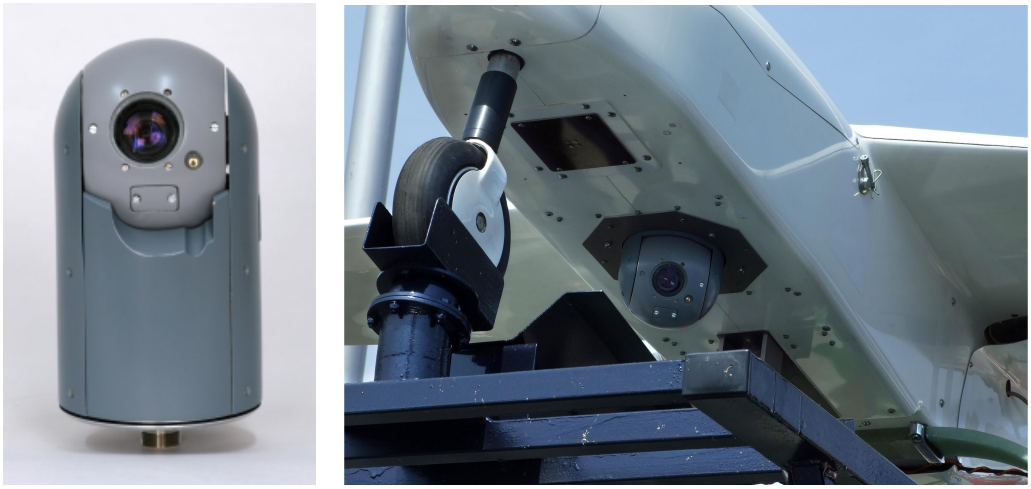
Seznam tabulek

Kapitola 1
Úvod

Pro pozorování objektů z pohybujících se dopravních či bezpilotních prostředků se často používají stabilizované kamery (kamerové hlavice). Příkladem je kamerová hlavice M120 (obrázek 1)[1]. Existuje a používá se několik druhů zavěšení kamer:

- bez stabilizace - otočné zavěšení bez stabilizace kamery (např. v průmyslu)
- mechanická stabilizace - využívá se v případech bez nároku na prostor a vlivy prostředí (letecké snímkování a filmový průmysl)
- elektromechanická stabilizace - používá se v kombinaci s řídícím systémem hlavně ve vojenství a u bezpečnostních složek

Hlavice zajišťuje kameře stabilizaci s využitím údajů z gyroskopických senzorů. Užitečnou a v tomto případě realizovatelnou součástí vybavení kamerové hlavice je vizuální systém pro sledování objektu v obraze. Při pohybu objektu v omezeném zorném poli kamery je těžké pro operátora objekt sledovat. Systém sledování sníží náročnost a umožní operátorovi věnovat pozornost vizuálnímu sledování především systémových údajů. Pro sledování objektu je pak nutné, aby řídící systém kamerové hlavice minimalizoval odchylku ϵ polohy sledovaného objektu ve snímku z kamery od polohy operátorem požadované, v praxi nejčastěji střed snímku.



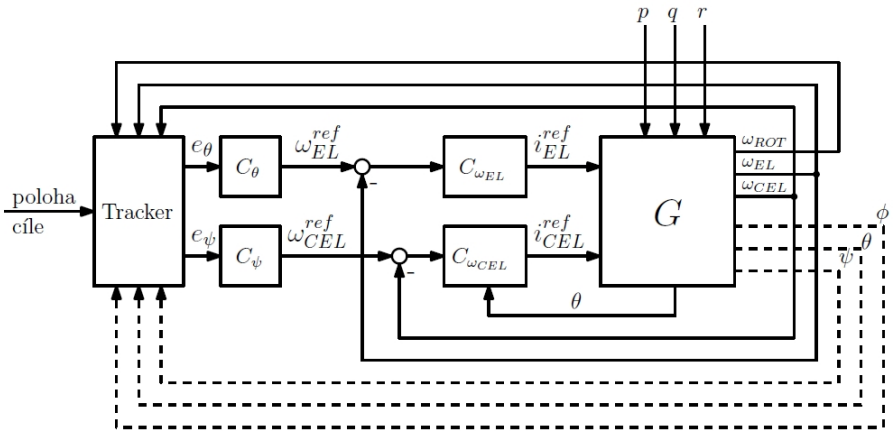
Obrázek 1: Použitá kamerová hlavice a její instalace na podvozku letounu MANTA

Kamera se pak stává součástí zpětnovazební smyčky, kterou můžeme vidět na obrázku (2)[2]. Součástí této smyčky je zmiňovaný sledovací algoritmus (Tracker), který posky-

[1] foto: Ing. Jan Salášek
[2] Schéma použito z projektu MAGYSKA

tuje minimální hodnotu odchylky ϵ. Výpočet této odchylky závisí na použitém algoritmu sledování, jeho náročnosti a také na jeho robustnosti. Pro řízení je důležité dopravní zpoždění. Toto zpoždění je především dáno rychlostí zpracování příchozích obrazů a snažíme se jej proto minimalizovat.

V následující kapitole si popíšeme stávající *SSD* algoritmus a představíme obecný *KL* algoritmus. V kapitole třetí uvedeme a popíšeme navržený algoritmus založený na *KLT*, jeho úpravy a doplnění. Ve čtvrté kapitole si stručně projdeme implementaci navrženého algoritmu do kamerové hlavice a v páté kapitole si porovnáme výsledky původního algoritmu *SSD* s navrženými variantami *KLT* algoritmu sledování.



Obrázek 2: Schéma obrazové zpětné vazby. *Tracker* představuje vizuální sledovací algoritmus a G pak stabilizovanou kamerovou hlavici jako celek. Bloky C_θ, C_ψ jsou polohovými regulátory a bloky $C_{\omega EL}, C_{\omega CEL}$ regulátory rychlostními pro osy azimut a elevace.

Kapitola 2
Stávající algoritmus sledování

2.1 Hlavice

V současné době máme k dispozici kamerovou hlavici z projektu MAGYSKA (číslo projektu TA02010887), v níž je nainstalována RGB kamera s možností její mechanické výměny, např. za termokameru. Samotná hlavice může být nainstalována na nosič (např. na autě, pod vrtulníkem) a má možnost rotačního pohybu v osách *azimut* a *elevace* za pomoci elektromotorů řízených pulzně-šířkovou modulací. Jako výpočetní a komunikační jednotka je v hlavici uložen počítačový modul, který se svými periferiemi komunikuje pomocí sběrnice CAN. Řízení zajišťuje řídící systém, který se skládá ze dvou částí: inerciální stabilizace a obrazové zpětné vazby.

2.1.1 Systém inerciální stabilizace

Cílem inerciální stabilizace optické osy je udržení této osy v klidu, zatímco se nosič, k němuž je kamera připevněna, pohybuje nežádoucím způsobem. Systém řídí polohu kamery vůči tomuto pohybujícímu se nosiči. V hlavici je použita hmotnostní stabilizace[3], rychlostí zpětná vazba (kompenzující rušivý točivý moment přenášený do kamery z nosiče) a proudová zpětná vazba.

2.1.2 Obrazová zpětná vazba

Samotné sledování hlavicí je určeno k tomu, aby se vybraný objekt vždy nacházel uprostřed zorného pole kamery, a my tak měli přehled o jeho okolí. V případě, že se pohybuje cíl, kamera, případně cíl s kamerou zároveň, je třeba mít dostatečně veliký odstup od okrajů snímků videa. Pro kompenzaci těchto pohybů zde slouží již zmíněné motory, které natáčejí kamerovou hlavici ve směru pohybu cíle, či ve směru opačném ke změně polohy nosiče hlavice.

2.1.3 Sběrnice CAN

Hlavní řídicí sběrnicí systému je sběrnice CAN. V hlavici jsou implementovány dva oddělené kanály průmyslové sběrnice CAN 2.0A, *vnitřní* a *vnější*. Sběrnice *vnitřní* je používána pro přenos dat týkajících se všech měření z instalovaných senzorů, akčních zásahů a další informace o chování řídících smyček. Po této sběrnici se posílají zprávy o

[3]Stabilizovaný předmět se snaží setrvat v klidu ve vztahu k inerciální vztažné soustavě, zatímco se nosič pohybuje.

ovládání kamery, nastavení konstant použitých PID regulátorů, atd. Rychlost sběrnice je 1Mb/s. Po *vnější* sběrnici jsou posílány zprávy z / do operátorské konzole, např. přepínání režimů, inicializování a ovládání hlavice. Rychlost sběrnice je 250kbit/s. Oddělení sběrnic je realizováno z důvodu separace dat, které nejsou bezprostředně nutné pro ovládání hlavice a tak nesnižují datovou propustnost. Přenos zpráv mezi sběrnicemi je realizován pomocí mikrokontroléru oddělovače.

Pro správnou velikost natočení motorů v jednotlivých osách, beroucí ohled na pohyb objektu ve videu[4], aktuální zoom čočky kamery a akční zásah motoru, je zde použita explicitně stanovená převodní tabulka s odpovídajícími veličinami (kalibrace kamery, resp. ohniskové vzdálenosti). Tyto stanovené hodnoty je možné v případě změny komponent zařízení upravit kalibrací.

Jako sledovací algoritmus používáme *SSD tracker (**S**um of **S**quare **D**ifferences) - suma rozdílů čtverců*, který podrobněji popíšeme a vysvětlíme v následujícím textu.

2.1.4 Konzole operátora

Aby mohla hlavice sloužit pro vizuální sledování (trackování) vybraného objektu, je potřeba předat řídícímu systému informaci o tom, jaký cíl sledovat. Tento výběr učiní operátor pomocí svého terminálu, na němž vidí např. informace o GPS poloze vrtulníku, jeho rychlosti a zoom kamery zároveň s aktuálním video výstupem z kamery. Operátor může kontrolovat zda je zvolený objekt sledován správně, případně svoji volbu upřesnit či změnit. Dále může upravit rychlost elektromotorů nebo zoom čočky kamery, atd..

2.2 Algoritmus sledování SSD

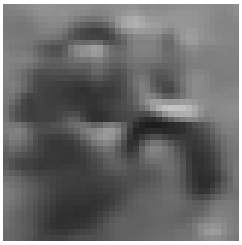
Mějme video pořízené kamerou, která se skládá ze sekvence snímků. Z této sekvence vyberme dvojici po sobě jdoucích snímků a označme je I_{t-1} a I_t, odpovídající pořadí snímku v sekvenci. Mějme dále polohu sledovaného objektu $\mathbf{u} = \begin{pmatrix} u_x & u_y \end{pmatrix}^T$ v souřadném systému snímku I. Definujme si model M sledovaného objektu jako uspořádanou množinu obrazových bodů, definovaných svojí polohou $\mathbf{x}_i$ v souřadném systému modelu vůči poloze objektu $\mathbf{u}$ a intenzitou[5]. Pro přepočet mezi souřadným systémem modelu M a snímku I použijeme vztah $M(\mathbf{x}_i) = I(\mathbf{u} + \mathbf{x}_i)$

Algoritmus SSD sledování pracuje s aktuálním snímkem pořízeným kamerou, v němž vždy probíhá hledání aktuální polohy sledovaného objektu zájmu pomocí modelu tohoto objektu, který jsme vytvořili při inicializaci algoritmu.

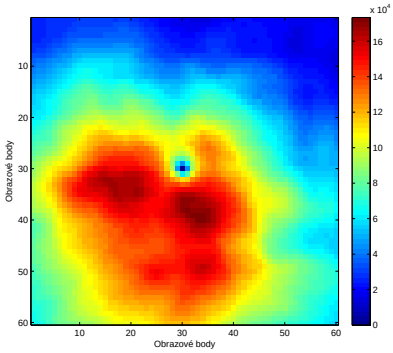
[4] V jednotkách obrazových bodů (pixelů)

[5] Použité snímky pro sledování, vstupní i patřící modelu, jsou uchovávány a zpracovávány jako pole celočíselných intenzit, v rozsahu od 0 do 255 (Případně od 0 do 1, v krocích po $\frac{1}{255}$), při zobrazení se jedná o stupně šedi.

(a) Snímek I_t



(b) Model sledovaného objektu



(c) Kriteriální chybová funkce

Obrázek 3: Ukázka snímku I_t, blízkého okolí sledovaného bodu a kriteriální chybové funkce

2.2.1 Princip SSD

Uvažujte snímek I_t a jemu odpovídající model M_{t-1} a polohu sledovaného bodu $\mathbf{u}$. Můžeme pak obecně definovat chybovou (kriteriální) funkci f jako

$$f(I_t, M_{t-1}, \mathbf{u}) = \frac{1}{n} \sum_{i=1}^{n} \left[I_t(\mathbf{u} + \mathbf{x}_i) - M_{t-1}(\mathbf{x}_i) \right]^2, \tag{1}$$

kde $\mathbf{u}$ je poloha zvoleného bodu ve snímku I a n je počet obrazových bodů modelu. Pokud je objekt ve snímku I_{t-1} na pozici $\mathbf{u}_{t-1}$, pak ve snímku I_t jeho souřadnice $\mathbf{u}_t$ stanoví jako

$$\mathbf{u}_t = \arg\min_{\mathbf{u} \in \Omega_t} f(I_t, M_{t-1}, \mathbf{u}), \tag{2}$$

kde Ω_t představuje množinu všech prohledávaných pozic, např. 2D kruhové či čtvercové okolí bodu $\mathbf{u}_{t-1}$. Příklad objektu, modelu a kriteriální funkce s lokálním minimem je na obrázku (3)

2.2.2 Aktualizace modelu

Objekt se při změně polohy mění. SSD algoritmus takové změny nemůže postihnout bez úpravy modelu a proto jej upravujeme. Tato změna probíhá pomocí exponenciálního

zapomínání:

$$M_t = \alpha I_{input}(\mathbf{u}_t) + (1 - \alpha) M_{t-1}, \tag{3}$$

které vždy aktualizuje model M_t pro použití k dalším kroku sledování $(t + 1)$ tak, že model M_{t-1} upraví váhovým koeficientem α a aktualizuje jej částí snímku I_t v místě lokalizovaného objektu. Velikost α představuje rychlost zapomínání modelu a jeho přizpůsobování se aktuálnímu snímku. Pro příliš vysoké hodnoty to může představovat riziko ztráty sledovaného objektu při špatném určení pozice objektu.

Sledování objektu je možné za předpokladu, že je v kameře dostatečně viditelný, vzhledově či tvarově se mění pouze pomalu a je jednoduše rozlišitelný od zbylých částí snímku. Pokud algoritmus sledování není schopen jednoznačně určit pozici objektu, považujeme jej za ztracený. V takovém případě je sledování přerušeno (stejně tak aktualizování modelu M.

2.3 Algoritmus sledování KLT

Metoda sledování objektu popsané poprvé v článku [1], nazývané *Kanade-Lucas-(Takeo) algoritmus* (zkráceně budeme používat **KL** či **KLT** tracker, algoritmus). Tato metoda se v mnoha ohledech odlišuje od *SSD* algoritmu shrnuté v předchozí části (2.2). Stejně jako zmiňovaný *SSD* tracker pracuje ve své základní a neupravené podobě pouze s obrázky v odstínech šedi.

KLT pracuje také s použitím modelu sledovaného objektu jako *SSD*, tedy přiřazení modelu M vstupnímu snímku I, a $\mathbf{u} = \begin{pmatrix} u_x & u_y \end{pmatrix}^T$ odpovídá souřadnicím sledovaného objektu ve snímku I. *KLT* je definován jako algoritmus pro výpočet optického toku mezi snímky I_{t-1} a I_t, kde modelem je okolí každého bodu $\mathbf{u}$, obvykle definované jako čtvercové okolí 5×5 nebo 10×10 obrazových bodů. Okolí Ω je tvořeno obrazovými body s relativními souřadnicemi $\mathbf{x}_1 \dots \mathbf{x}_n$, kde $\mathbf{x}_i \in \Omega$. Pohyb (uvažujeme pouze posunutí) můžeme parametricky zapsat jako

$$\mathbf{W}(\mathbf{u}, \mathbf{d}) = \begin{pmatrix} u_x + d_1 \\ u_y + d_2 \end{pmatrix}, \tag{4}$$

kde $\mathbf{d}$ představuje posunutí (hledaný opticky tok) v obou osách tak, aby platilo, že $\mathbf{W}(\mathbf{u}, \mathbf{d})$ převezme polohu objektu $\mathbf{u}$ v souřadnicích modelu I_{t-1} a namapuje ji na sub-pixelovou[6] pozici ve snímku I_t. Budeme předpokládat, že model M_{t-1} odpovídá při startu celému snímku I_{t-1} a provádíme jeho aktualizaci dle vztahu (3).

Cílem *KLT* algoritmu je minimalizování sumy kvadrátů chyb mezi modelem M a vstupním snímkem I, při přepočtu souřadnic vzhledem k modelu M:

$$f(I_t, M_{t-1}, \mathbf{u}, \mathbf{d}) = \sum_{\mathbf{x}_i} \left[I_t \left(\mathbf{W}(\mathbf{u} + \mathbf{x}_i, \mathbf{d}) \right) - M_{t-1}(\mathbf{u} + \mathbf{x}_i) \right]^2. \tag{5}$$

[6]Sub-pixel představuje pozici mezi jednotlivými obrazovými body, nejedná se tedy dále pouze o celočíselné souřadnice, ale pro zvýšení přesnosti se využívají hodnoty reálné.

Minimalizace rovnice (5) je prováděna s ohledem na $\mathbf{d}$ a suma je počítána přes všechny body $\mathbf{x}_i$. Samotná minimalizační úloha je nelineární, i když vektor deformace $\mathbf{W}(\mathbf{u}, \mathbf{d})$ je lineárním v $\mathbf{d}$, ale snímek $I(\mathbf{u})$ je nelineárním v $\mathbf{u}$. Hodnoty jednotlivých obrazových bodů, z nichž se skládá $I(\mathbf{u})$ obecně nemají vztah k souřadnicím $\mathbf{u}$. Pro optimalizaci výrazu v rovnici (5), *KL* algoritmus předpokládá, že odhad $\mathbf{d}$ je znám a dále je řešitelný pomocí inkrementace parametrem $\Delta\mathbf{d}$, což můžeme zapsat jako

$$f(I_t, M_{t-1}, \mathbf{u}, \mathbf{d}, \Delta\mathbf{d}) = \sum_{\mathbf{x}_i} \left[I_t \left(\mathbf{W}(\mathbf{u} + \mathbf{x}_i, \mathbf{d} + \Delta\mathbf{d}) \right) - M_{t-1}(\mathbf{u} + \mathbf{x}_i) \right]^2 . \tag{6}$$

Aktualizaci parametru $\mathbf{d}$ zapíšeme

$$\mathbf{d} \leftarrow \mathbf{d} + \Delta\mathbf{d}. \tag{7}$$

Iteraci provádíme přes kroky (6, 7) tak dlouho, dokud parametr $\mathbf{d}$ konverguje. Jako test konvergence je libovolná norma vektoru $\Delta\mathbf{d}$, která je menší než stanovený práh ε, např. $\left\| \Delta\mathbf{d} \right\| \leq \varepsilon$.

2.3.1 Odvození KL algoritmu

KL algoritmus je variantou Gauss-Newtonova gradientní klesající nelineární optimalizační úlohy. Nelineární rovnici (6) linearizujeme Taylorovým polynomem prvního řádu členu $I\left(\mathbf{W}(\mathbf{u}, \mathbf{d} + \Delta\mathbf{d})\right)$ a získáme

$$f(I_t, M_{t-1}, \mathbf{u}, \mathbf{d}, \Delta\mathbf{d}) = \sum_{\mathbf{x}_i} \left[I_t \left(\mathbf{W}(\mathbf{u} + \mathbf{x}_i, \mathbf{d}) \right) + \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{d}} \Delta\mathbf{d} - M_{t-1}(\mathbf{u} + \mathbf{x}_i) \right]^2 . \tag{8}$$

V této rovnici (8) výraz $\nabla I = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right)$ představuje gradient snímku I spočtený v bodě $\mathbf{W}(\mathbf{u}, \mathbf{d})$, tzn. ∇I je spočten v souřadném systému I a poté dosazen zpátky do souřadnic modelu M s použitím aktuálního odhadu toku $\mathbf{W}(\mathbf{u}, \mathbf{d})$. Výraz $\frac{\partial \mathbf{W}}{\partial \mathbf{d}}$ je Jakobián tohoto toku. Předpokládejme $\mathbf{W}(\mathbf{u}, \mathbf{d}) = \frac{(W_x(\mathbf{u}, \mathbf{d})}{W_y(\mathbf{u}, \mathbf{d})}$, pak pro 2D platí

$$\frac{\partial \mathbf{W}}{\partial \mathbf{d}} = \begin{pmatrix} \frac{\partial W_x}{\partial p_1} & \frac{\partial W_x}{\partial p_2} \\ \frac{\partial W_y}{\partial p_1} & \frac{\partial W_y}{\partial p_2}. \end{pmatrix} \tag{9}$$

Minimalizace rovnice (8) je problém nejmenších čtverců a řešení můžeme odvodit následovně. Parciální derivace rovnice (8) vzhledem k $\Delta\mathbf{d}$ je

$$\frac{\partial f}{\partial \Delta\mathbf{d}} = 2 \sum_{\mathbf{x}_i} \left(\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{d}} \right)^T \left(I_t \left(\mathbf{W}(\mathbf{u} + \mathbf{x}_i, \mathbf{d}) \right) + \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{d}} \Delta\mathbf{d} - M_{t-1}(\mathbf{u} + \mathbf{x}_i) \right), \tag{10}$$

kde člen $\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{d}} \Delta\mathbf{d}$ odpovídá směru nejstrmějšího poklesu mezi snímky. Pokud tento vztah položíme roven nule, spočtení nám poskytuje konečné řešení pro minimální hodnotu výrazu v rovnici (8) jako

$$\Delta\mathbf{d} = H^{-1} \sum_{\mathbf{x}_i} \left(\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{d}} \right)^T \left(M_{t-1}(\mathbf{u} + \mathbf{x}_i) - I_t \left(\mathbf{W}(\mathbf{u} + \mathbf{x}_i, \mathbf{d}) \right) \right), \tag{11}$$

kde H je pro 2D Hessova matice

$$H = \sum_{\mathbf{x}_i} \left(\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{d}} \right)^T \left(\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{d}} \right) \tag{12}$$

Člen $\sum_{\mathbf{x}_i} \left(\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{d}} \right)^T \left(M_{t-1}(\mathbf{u}+\mathbf{x}_i) - I_t \left(\mathbf{W}(\mathbf{u}+\mathbf{x}_i, \mathbf{d}) \right) \right)$ odpovídá aktualizaci parametru nejstrmějšího poklesu. Z rovnice (11) můžeme říci, že parametr $\Delta\mathbf{d}$ je právě aktualizace parametru nejstrmějšího poklesu vynásobená inverzní Hessovou maticí. Samotný *KL* algoritmus se skládá z iterativního aplikování rovnic (7) a (11).

Z předchozích rovnic vidíme, že gradient ∇I musí být spočten pro použití v $\mathbf{W}(\mathbf{u}, \mathbf{d})$ a Jakobián $\frac{\partial \mathbf{W}}{\partial \mathbf{d}}$ v $\mathbf{d}$, obojí obecně závisí na $\mathbf{d}$. V některých případech (pro optické toky složené pouze z translace či toky afinní) může být Jakobián konstantní, nicméně pro úplnost je třeba všechny kroky výpočtu algoritmu v každém kroku opakovat a toto zjednodušení vyloučit. Odhad parametru $\mathbf{d}$ se může lišit mezi jednotlivými iteracemi.

Pro optické toky $\mathbf{W}(\mathbf{u}, \mathbf{d})$ máme pouze požadavek, aby byly diferencovatelné vzhledem k parametru $\mathbf{d}$, což potřebujeme ke spočtení Jakobiánu $\frac{\partial \mathbf{W}}{\partial \mathbf{d}}$. Obecně jsou tyto toky diferencovatelné dle $\mathbf{u}$, ale tato podmínka není nutná.

2.3.2 Varianty KLT

V praxi se používají nejčastěji čtyři varianty algoritmu, které se liší v záměně rolí modelu M a snímku I ve výpočtu a také způsobem aktualizace odhadu pohybu při iteraci. Původní algoritmus a další tři odvozené od tohoto obecného algoritmu. Tyto varianty poskytují jistá zjednodušení pro výpočet optických toků tím, že upravují tvar rovnice pro minimalizaci 6 a krok změnu parametru $\mathbf{d}$ v kroku iterace. Tyto varianty jsou blíže popsané a odvozené v článku ([2]) a poskytují empiricky podložené shodné výsledky (bez přidaného šumu). Pokud zahrneme do procesu vyšší hladinu šumu, zvyšuje se nutný počet iterací k dosáhnutí konvergence a při zvyšujících se hodnotách míra úspěšnosti konvergence klesá. Jednotlivé algoritmy se liší pouze v požadavcích na optické toky a na výpočetní náročnost, 6.

Zrychlením a zjednodušením výpočtu může být aproximace gradientu poklesu hodnoty kriteriální funkce. Většina nelineárních optimalizací a algoritmů pro odhad parametrů pracuje s iterací skládající se ze dvou kroků. Prvním krokem je přibližný lokální odhad kriteriální funkce, nejčastěji pomocí lineární či kvadratické aproximace v okolí aktuálního odhadu parametrů. Druhým krokem je poté aktualizace odhadů těchto parametrů na základě odhadu kriteriální funkce. V základním algoritmu *KLT* je použita Gauss-Newtonova metoda minimalizace, ale můžeme například použít Gauss-Newtonovu minimalizaci s diagonalizací Hessovy matice, Levenberg-Marquardtovu či Newtonovu metodu minimalizace. Mimo těchto postupů existuje nespočet dalších, ne všechny jsou však pro svojí citlivost na šum vhodné pro obecné případy či jsou významně rychlejší než Gauss-Newtonova metoda.

Algoritmy sledování založené na *KLT* jsou obvykle popsány pro intenzitní snímky. Existují však postupy, které používají informaci o barvě či barevnosti obrázku (případně jeho jednotlivých částí), např. ([3]) nebo využití nikoliv jednotlivých barev, ale barevného histogramu, popsaného v ([4]).

Kapitola 3
Navržený algoritmus sledování

V této kapitole podrobněji popíšeme konkrétní použití *KLT* a jaké jsme provedli jeho dodatečné úpravy při řešení naší úlohy.

3.1 Upravený algoritmus KLT

Jak již bylo v předchozí kapitole uvedeno, existují různé varianty KLT algoritmu, které matematicky poskytují shodné či dostatečně blízké výsledky, a které se liší převážně v rozdílných minimalizacích. Našim potřebám nejvíce vyhovoval *KL* algoritmus ve variantě dopředné sčítací(„Forward additive“) s použitím Gauss-Newtonovy metody minimalizace. Tato varianta poskytuje vysokou robustnost při zachování přiměřené výpočetní náročnosti. Vybrali jsme si implementaci metody popsané v článku [5].

3.1.1 Popis algoritmu

Mějme dvojici 2D snímků v odstínech šedi pořízených kamerou, a označme je I_{t-1} a I_t. Jako první snímek budeme považovat snímek I_{t-1} a jako druhý I_t. Intenzitu v libovolném bodě takovýchto šedých snímků můžeme lokalizovat jako $I_{t-1}(x, y)$, případně $I_t(x, y)$, kde souřadnice x představuje vodorovnou osu a y osu svislou, s počátkem v levém horním rohu snímku.

Máme-li náš sledovaný objekt na pozici $\mathbf{u} = \begin{pmatrix} u_x & u_y \end{pmatrix}^T$ ve snímku I_{t-1}, cílem algoritmu je nalézt jeho pozici $\mathbf{v} = \mathbf{u} + \mathbf{d} = \begin{pmatrix} u_x + d_x \\ u_y + d_y \end{pmatrix}$ ve snímku I_t, předpokládáme, že $I_{t-1}(\mathbf{u})$ a $I_t(\mathbf{v})$ jsou „shodné“. Jelikož v praxi nechceme sledovat pouze objekt o velikosti jednoho obrazového bodu, budeme předpokládat, že má určitou velikost, popsanou jako čtverec o hraně q se středem v bodě $\mathbf{u}$, a budeme jej pokládat za model daného objektu, (2.2.2). Pro hledání bodu $\mathbf{u}$, respektive $\mathbf{v}$, použijeme zmenšenou oblast snímku I_t, jejíž rozměry popíšeme pro jednotlivé osy jako $(2 \cdot \omega_x + 1)$ a $(2 \cdot \omega_y + 1)$, a dále ji budeme nazývat prohledávaná oblast a značit jako Ω. Rozměry ω_x a ω_y odpovídají vzdálenosti středového obrazového bodu od hrany této oblasti.

Cílem algoritmu bude minimalizovat chybovou funkci ϵ popsanou jako

$$\epsilon(\mathbf{d}) = \epsilon(d_x, d_y) = \sum_{\mathbf{x_i} \in \Omega} \left[I_{t-1}(\mathbf{u} + \mathbf{x}_i) - I_{t+1}(\mathbf{u} + \mathbf{x}_i + \mathbf{d}) \right]^2 . \tag{13}$$

Obecně předpokládáme, že velikost posunutí $\mathbf{d}$ bude v jednotlivých osách nejhůře shodná jako jsou rozměry prohledávané oblasti, tedy $d_x \leq \omega_x$ a $d_y \leq \omega_y$.

Pro zajištění co největší robustnosti vzhledem k rychlosti algoritmu, je zde použita pyramidová implementace *KLT*.

3.1.2 Sledování objektu s použitím pyramidové reprezentace

Definujme si pyramidovou reprezentaci snímku I o rozměrech $n_x \times n_y$. Mějme $I^0 = I$, jakožto „nultou“ úroveň snímku (v původním, maximálním rozlišení - $n_x^0 = n_x$ a $n_y^0 = n_y$). Pyramidová reprezentace poté spočívá v postupném výpočtu I^1 z I^0, poté I^2 z I^1, I^3 z I^2, atd.. Mějme $L = 1, 2, ...,$ odpovídající úrovním pyramidy, pak I^{L-1} odpovídá snímku na úrovni $L-1$ a n_x^{L-1}, n_y^{L-1} je pak šířka, resp. výška snímku I^{L-1}. Snímek I^{L-1} poté definujeme jako

$$\begin{aligned} I^L(x,y) =& \frac{1}{4} I^{L-1} + \\ & \frac{1}{8}\left[{}^{L-1}(2x-1,2y) + I^{L-1}(2x+1,2y) + I^{L-1}(2x,2y-1) + I^{L-1}(2x,2y+1)\right] + \\ & \frac{1}{16}\left[I^{L-1}(2x-1,2y-1) + I^{L-1}(2x+1,2y+1)\right] + \\ & \frac{1}{16}\left[I^{L-1}(2x-1,2y+1) + I^{L-1}(2x+1,2y-1)\right] \end{aligned} \tag{14}$$

Aby byla dodržena podmínka pro rozměry snímků mezi úrovněmi, musí platit

$$n_x^L \leq \frac{n_x^{L-1}+1}{2} \qquad\qquad n_y^L \leq \frac{n_y^{L-1}+1}{2} \tag{15}$$

Cílem pyramidové reprezentace je zvládnout větší pohyby mezi snímky I_{t-1} a I_t, i pokud by byly větší než je prohledávaná oblast. V praxi není potřeba volit vyšší úroveň než $L = 4$, při němž dochází k 16 násobnému snížení rozlišení snímku.

Vzhledem k tomu, že se mění rozlišení použitých snímků na jednotlivých úrovních pyramidy, je třeba měnit i souřadnice sledovaného bodu $\mathbf{u}$. Mějme $\mathbf{u}^L = \begin{pmatrix} u_x^L & u_y^L \end{pmatrix}^T$ pro danou úroveň pyramidy L, mezi jednotlivými úrovněmi můžeme přecházet pomocí přepočtu $\mathbf{u}^L = \frac{\mathbf{u}}{2^L}$ pro $\mathbf{u}^0 = \mathbf{u}$.

Samotné sledování objektu pomocí pyramid probíhá tak, že se spočte pohyb (posunutí) na nejhlubší úrovni pyramidy L_m, tedy na snímku s nejnižším rozlišením. Poté se výsledek výpočtu propaguje na úroveň vyšší $(L_m - 1)$ jako počáteční odhad posunutí $\mathbf{d}$ pro úroveň $(L_m - 1)$. Tento proces se opakuje až na úroveň $L = 0$, tedy na původní rozlišení vstupního snímku.

Mějme počáteční odhad optického toku na úrovni L, $\mathbf{g}^L = \begin{pmatrix} g_x^L & g_y^L \end{pmatrix}^T$, který jsme si spočetli na úrovni $L+1$. Poté pro spočtení optického toku pro úroveň L musíme určit vektor zbytkového posunutí $\mathbf{d}^L$, pro který je chybová funkce ϵ^L minimální

$$\epsilon^L(\mathbf{d}^L) = \epsilon^L(d_x^L, d_y^L) = \sum_{\mathbf{x_i} \in \Omega} \left[I_{t-1}^L(\mathbf{x}_i) - I_t^L(\mathbf{x}_i + \mathbf{g}^L + \mathbf{d}^L)\right]^2. \tag{16}$$

Velikost integračního okna $(2\omega_x + 1) \times (2\omega_y + 1)$ se mezi jednotlivými úrovněmi nemění a počáteční odhad změny polohy $\mathbf{g}^L$ je použit pro předzpracování snímku I_t. Vektor zbytkového posunutí $\mathbf{d}^L$ je následně malý a je jednodušší jej spočíst skrze standardní krok KL algoritmu.

Optický tok mezi jednotlivými úrovněmi lze přepočíst pomocí vztahu $\mathbf{g}^{L-1} = 2(\mathbf{g}^L + \mathbf{d}^L)$, kde jako počáteční podmínku pro nejhlubší úroveň L_m volíme $\mathbf{g}^{L_m} = \begin{pmatrix} 0 & 0 \end{pmatrix}^T$. Pro spočtení vektoru $\mathbf{d}^{L-1}$ je nutné znovu provést minimalizaci rovnice (16), $\epsilon^{L-1}(\mathbf{d})^{L-1}$. Výsledný vektor zbytkového posunutí $\mathbf{d}$ můžeme určit jako $\mathbf{d} = \sum_{L=0}^{L_m} 2^L \mathbf{d}^L$.

Významnou výhodou pyramidové implementace je při výpočtu vektoru $\mathbf{d}^L$, který může být pro jednotlivé úrovně malý, ale v celkovém součtu $\mathbf{d}$ může dosahovat vysokých hodnot, při zachování relativně malého integračního okna.

3.1.3 Iterativní KLT

Na každé úrovni L je cílem nalézt vektor $\mathbf{d}^L$ minimalizující funkci ϵ^L (16). Z důvodů opakování shodných operací na každé úrovni L nyní provedeme odstranění indexu L a upravíme si použité snímky

$$\forall (x,y) \in [p_x - \omega_x - 1, p_x + \omega_x + 1] \times [p_y - \omega_y - 1, p_y + \omega_y + 1],$$
$$A(x,y) \doteq I^L_{t-1}(x,y) \tag{17}$$
$$\forall (x,y) \in [p_x - \omega_x, p_x + \omega_x] \times [p_y - \omega_y, p_y + \omega_y],$$
$$B(x,y) \doteq I^L_t(x + g^L_x, y + g^L_y) \tag{18}$$

Pro snímek $A(x,y)$ dojde ke zvětšení rozsahu integračního okna na $(2\omega_x+3)\times(2\omega_y+3)$, což využijeme dále v rovnici (19). Abychom zamezili chybám, provedeme změnu značení i pro vektor posunutí $\bar{\nu} = \begin{pmatrix} \nu_x & \nu_y \end{pmatrix}^T = \mathbf{d}^L$ a pro pozici sledovaného objektu $\mathbf{p} = \begin{pmatrix} p_x & p_y \end{pmatrix}^T = \mathbf{u}^L$. S tímto novým značením provedeme úpravu rovnice (16) na

$$\epsilon(\bar{\nu}) = \epsilon(\nu_x, \nu_y) = \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \left(A(x,y) - B(x+\nu_x, y+\nu_y)\right)^2 \tag{19}$$

V optimu této rovnice platí $\left.\frac{\partial \epsilon(\bar{\nu})}{\partial(\bar{\nu})}\right|_{\bar{\nu}=\bar{\nu}_{opt}} = \begin{pmatrix} 0 & 0 \end{pmatrix}^T$, po dosazení a upravení rovnice (19) získáme

$$\left.\frac{\partial \epsilon(\bar{\nu})}{\partial(\bar{\nu})}\right|_{\bar{\nu}=\bar{\nu}_{opt}} = -2 \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \left(A(x,y) - B(x+\nu_x, y+\nu_y)\right) \cdot \begin{pmatrix} \frac{\partial B}{\partial x} & \frac{\partial B}{\partial y} \end{pmatrix} \tag{20}$$

Snímek $B(x+\nu_x, y+\nu_y)$ nyní rozvineme v Taylorův polynom prvního řádu v bodě $\bar{\nu} = \begin{pmatrix} 0 & 0 \end{pmatrix}^T$, což můžeme provést s vysokou pravděpodobností na správné aproximace, protože očekáváme pouze malý vektor posunutí (kvůli použití pyramid)

$$\frac{\partial \epsilon(\bar{\nu})}{\partial(\bar{\nu})} \approx -2 \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \left(A(x,y) - B(x,y) - \begin{pmatrix} \frac{\partial B}{\partial x} & \frac{\partial B}{\partial y} \end{pmatrix} \bar{\nu}\right) \cdot \begin{pmatrix} \frac{\partial B}{\partial x} & \frac{\partial B}{\partial y} \end{pmatrix}, \tag{21}$$

kde matice $\begin{pmatrix} \frac{\partial B}{\partial x} & \frac{\partial B}{\partial y} \end{pmatrix}$ představuje vektor gradientu snímku. Vztah $(A(x,y) - B(x,y))$ můžeme chápat jako derivaci snímku v čase v bodě $\begin{pmatrix} x & y \end{pmatrix}^T$

$$\forall (x,y) \in (p_x - \omega_x, p_x + \omega_x) \times (p_y - \omega_y, p_y + \omega_y),$$
$$\delta I(x,y) \doteq A(x,y) - B(x,y) \tag{22}$$

Nyní si označme

$$\nabla I = \begin{pmatrix} I_x \\ I_y \end{pmatrix} \doteq \begin{pmatrix} \frac{\partial B}{\partial x} \\ \frac{\partial B}{\partial y} \end{pmatrix} \tag{23}$$

Derivace snímku I_x a I_y můžeme spočíst přímo ze snímku $A(x, y)$, přes okolí $(2\omega_x + 1) \times (2\omega_y + 1)$ bodu $\mathbf{p}$, nezávisle na snímku $B(x, y)$. Přepíšeme rovnici (21) do tvaru

$$\begin{aligned} \frac{1}{2}\frac{\partial \epsilon(\bar{\nu})}{\partial \bar{\nu}} &\approx \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} (\nabla I^T \bar{\nu} - \delta I)\nabla I^T, \\ \frac{1}{2}\left(\frac{\partial \epsilon(\bar{\nu})}{\partial \bar{\nu}}\right)^T &\approx \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \left(\begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix} \bar{\nu} - \begin{pmatrix} \delta I I_x \\ \delta I I_y \end{pmatrix}\right) \end{aligned} \tag{24}$$

Zavedeme si substituci

$$G \doteq \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix} \qquad \bar{b} \doteq \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \begin{pmatrix} \delta I I_x \\ \delta I I_y \end{pmatrix} \tag{25}$$

Rovnici (24) s pomocí této substituce můžeme přepsat do tvaru

$$\frac{1}{2}\left(\frac{\partial \epsilon(\bar{\nu})}{\partial \bar{\nu}}\right)^T \approx G\bar{\nu} - \bar{b}. \tag{26}$$

Z této rovnice pak můžeme určit optimální vektor optického toku jako $\bar{\nu}_{opt} = G^{-1}\bar{b}$. Tento vztah platí pouze za předpokladu, že matice G je invertovatelná, což odpovídá tomu, že snímek $A(x, y)$ obsahuje informace o gradientu v obou osách x, y pro okolí bodu $\mathbf{p}$. Tento výraz je základní rovnice KL algoritmu pro optický tok, platící pro malá posunutí, kvůli použití Taylorova rozvoje prvního řádu. V praxi je pro dosažení výsledku třeba iterovat několikrát (Newton-Raphsonova metoda).

Nyní si popíšeme samotný iterativní algoritmus:

1. Mějme index iterace k s počáteční hodnotou 1. Pro $k \geq 1$ pak předpokládejme počáteční odhad vektoru posunutí $\bar{\nu}^{k-1} = \begin{pmatrix} \nu_x^{k-1} \\ \nu_y^{k-1} \end{pmatrix}$, pro který mějme nový posunutý snímek B_k

$$\begin{aligned} \forall (x, y) \in (p_x - \omega_x, p_x + \omega_x) \times (p_y - \omega_y, p_y + \omega_y), \\ B_k(x, y) = B(x + \nu_x^{k-1}, y + \nu_y^{k-1}). \end{aligned} \tag{27}$$

2. Cílem je určit vektor zbytkového posunutí $\bar{\eta}^k = \begin{pmatrix} \eta_x^k \\ \eta_y^k \end{pmatrix}$ minimalizující chybovou funkci

$$\epsilon^k(\bar{\eta}^k) = \epsilon^k(\eta_x^k, \eta_y^k) = \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \left(A(x, y) - B_k(x + \eta_x^k, y + \eta_y^k)\right)^2. \tag{28}$$

3. Řešením této minimalizace je (podobně jako dříve uvedeno) vztah $\bar{\eta}^k = G^{-1}\bar{b}_k$, kde $\bar{b}_k$ je dvouřádkový sloupcový vektor

$$\bar{b}_k = \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \begin{pmatrix} \delta I_k(x, y) I_x(x, y) \\ \delta I_k(x, y) I_y(x, y) \end{pmatrix}, \tag{29}$$

   kde δI_k představuje k-tý rozdílový snímek definovaný jako

$$\begin{aligned} \forall (x, y) \in (p_x - \omega_x, p_x + \omega_x) \times (p_y - \omega_y, p_y + \omega_y), \\ \delta I_k(x, y) = A(x, y) - B_k(x, y). \end{aligned} \tag{30}$$

4. Před začátkem iterace si můžeme spočíst derivace snímků I_x a I_y (v okolí bodu $\mathbf{p}$) a matici G, která také zůstává konstantní napříč iteracemi. Díky tomuto zjednodušení budeme v průběhu iterování během kroku k přepočítávat pouze vektory $\bar{b}_k$ a $\bar{\nu}^{k-1}$. Po spočtení zbytkového optického toku $\bar{\eta}^k$ provedeme aktualizaci odhadu vektoru posunutí $\bar{\nu}^k$ pro další krok iterace $k + 1$ jako $\bar{\nu}^k = \bar{\nu}^{k-1} + \bar{\eta}^k$.

Iterování provádíme tak dlouho, dokud není zbytkový vektor posunutí menší než stanovená mez, případně není dosaženo určitého počtu kroků iterace. V prvním kroku iterace pro $k = 1$ si stanovíme počáteční odhad $\bar{\nu}^0 = \begin{pmatrix} 0 & 0 \end{pmatrix}^T$. Předpokládejme, že jsme k dosažení konvergence potřebovali celkem K kroků, výsledné řešení můžeme zapsat jako

$$\bar{\nu} = \mathbf{d}^L = \bar{\nu}^K = \sum_{k=1}^{K} \bar{\eta}^k. \tag{31}$$

3.1.4 Sledování objektu poblíž okrajů snímku

Pro sledovací algoritmus je užitečné, aby byl schopný sledovat vybraný objekt v celém vstupním snímku, tedy i v jeho okrajích. Pokud máme L_m jakožto nejhlubší úroveň pyramidy a integrační okno o velikosti $(2\omega_x + 1) \times (2\omega_y + 1)$, pak „zakázaná oblast“ zasahuje $2^{L_m}\omega_x$ (resp. $2^{L_m}\omega_y$)[7] obrazových bodů od okraje minulého snímku. V této oblasti pokračuje algoritmus upraveným způsobem, a to tak, že výpočet probíhá přes obrazové body snímku, které jsou na průniku integračního okna a platných obrazových bodů uvnitř snímku (nepřesáhneme při výpočtu sum přes okraj). Toto se týká snímků $I_x(x, y), I_y(x, y)$ a $\delta I_k(x, y)$. Také při každé iteraci bude potřeba provést přepočtení matice G a následně vektor $\bar{b}_k$, protože již nadále matice G nebude konstantní.

Pokud ovšem sledovaný objekt, představovaný bodem $\mathbf{p} = \begin{pmatrix} p_x & p_y \end{pmatrix}^T$ opustí snímek I^L_{t-1}, případně jemu odpovídající vyhledávaný bod $\begin{pmatrix} p_x + g^L_x + \nu^{k-1}_x \\ p_y + g^L_y + \nu^{k-1}_y \end{pmatrix}$ opustí snímek I^L_t, prohlašujeme objekt za ztracený.

3.1.5 Prohlášení objektu za ztracený

Jak bylo zmíněno v podkapitole výše (3.1.4), objekt prohlásíme za ztracený, pokud opustí jeden ze snímků I^L_{t-1}, I^L_t. Dále můžeme prohlásit objekt za ztracený, pokud chybová funkce $\epsilon(\mathbf{d})$ v rovnici (13) je vyšší, než je definovaná mez. Tuto mez je obtížné pevně stanovit, proto volíme dynamicky stanovenou dle postupu, který je uvedený v bakalářské práci [6]. Ta vychází z posledních 10 hodnot chybové funkce z nichž si vypočítáme průměr $\bar{e}$ a maximální hodnotu z 10 chyb $max_{10}(e)$. Tuto dvojici hodnot dosadíme do vzorce:

$$g_{mez} = \frac{max(e) + \bar{e}}{m}, \tag{32}$$

kde g_{mez} je stanovená mez a m pak empiricky určená hodnota, dle bakalářské práce $m = 2,5$. S touto mezí pak můžeme pro každou sekvenci lépe stanovit, kdy je objekt ztracený a kdy nikoliv.

[7] Pro každou úroveň pyramidy se jedná o ω_x, resp. ω_y obrazových bodů.

Kapitola 4
Implementace

Zvolený algoritmus, popsaný v kapitole 3 (2.3.2), jsme implementovali do stávajícího kódu *SSD* algoritmu tak, aby oba algoritmy byly od sebe oddělené. Tato implementace je realizována s použitím knihovny *OpenCV* pro jazyk *C++*, v němž je napsaný výsledný algoritmus. Z knihovny jsme využili části funkcí, které se týkají *KL* algoritmu.

Implementaci jsme provedli tak, aby výsledný program byl přeložitelný pro dvě požadované platformy, Windows a Linux. V kódu jsme provedli dodatečné změny, např. sledování vybraného objektu na úrovni sub-pixelů, spuštění algoritmu z více bodů a načítání sekvencí ze souboru.

4.1 Implementace navrženého algoritmu

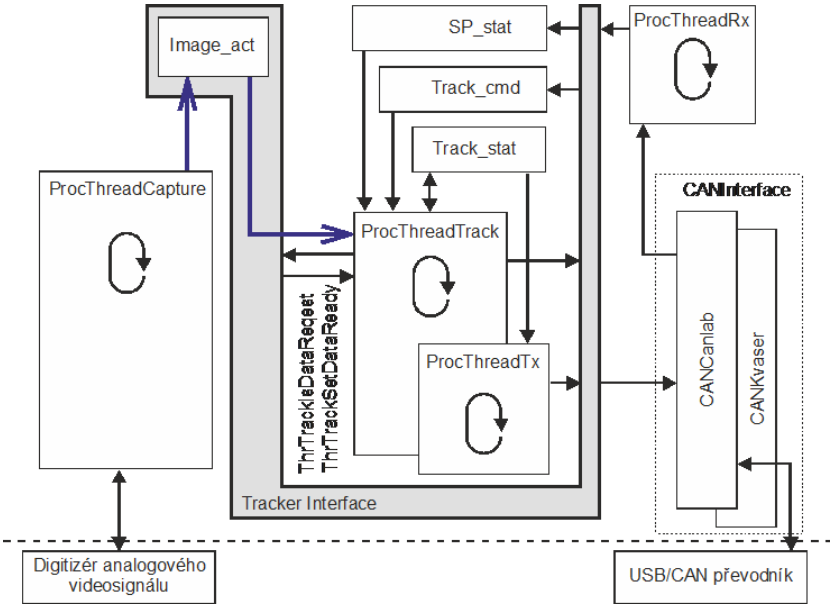
4.1.1 Původní schéma programu

Strukturu programu na obrázku (4) můžeme rozdělit na vnější a vnitřní část. Ve vnější části je převodník USB/CAN sloužící pro komunikaci řídícím systémem kamerové hlavice a digitizér analogového signálu pro převod vstupního videosignálu kamery do digitálního signálu zpracovávaného počítačem. Ve vnitřní části jsou funkce čtveřice hlavních vláken (označené textem „Thread“) a dvojice rozhraní („interface“) zastřešující sledovací algoritmus, resp. dva různé druhy používaných USB/CAN převodníků. V následujících odstavcích si popíšeme čtveřici hlavních vláken programu a používané struktury v původním programu s *SSD* sledovacím algoritmem.

4.1.1.1 Vlákno ThrTracking

Součástí tohoto vlákna je funkce ProcThreadTrack uvedená ve schématu (4), která za svého průběhu volá funkci processTrack. Tato funkce jako vstupní proměnné přijímá rozměry snímku image_act, v němž hledáme sledovaný objekt. ProcessTrack je stavový automat bez návratové hodnoty, který rozlišuje šestici stavů:

1. Začátek sledování
2. Úspěšné nalezení sledovaného objektu ve snímku z kamery
3. Částečná ztráta sledovaného objektu (např. krátkodobý zákryt)
4. Úplná ztráta sledovaného objektu (objekt nebyl nalezen v průběhu 5 snímků, zatímco byl ve stavu 3)
5. Chyba, jejímž důsledkem je sledovaný objekt považován za ztracený



Obrázek 4: Blokové schéma procesů v programu obrazové zpětné vazby, převzato z dokumentace projektu MAGYSKA.

6. Zastavení sledování

Za běhu stavového automatu dochází k volání funkce objectTrack, jejímž vstupem je odkaz do operační paměti počítače s uloženým aktuálním snímkem z kamery (v němž budeme hledat sledovaný objekt) a rozměry tohoto snímku. V této funkci používáme lineárního prediktoru budoucí polohy sledovaného objektu z jeho předchozího pohybu. Pokud je stav automatu 2 nebo 3, probíhá hlavní funkce sledovacího algoritmu *SSD* locateGrayModel. Při stavu automatu 2 probíhá aktualizování modelu sledovaného objektu pomocí vztahu (3). Pokud došlo v průběhu sledování ke změně zoomu kamery, dojde k přepočtení poloh intenzit v modelu s využitím referenční tabulky pro přepočet.

LocateGrayModel jako vstupní parametry přijímá aktuální snímek, model sledovaného objektu a polohu sledovaného objektu z předchozího kroku. V této funkci implementujeme rovnici vyjádřenou matematicky v (1), její návratová hodnota je nalezená pozice sledovaného objektu v aktuálním snímku a jí odpovídající hodnota kriteriální chybové funkce. Pokud hodnota chybové funkce je vyšší než stanovená v konfiguračním souboru, či nedošlo k nalezení sledovaného objektu, přepneme stav stavového automatu z 2 na 3 za předpokladu, že ve stavu 3 již není.

Za běhu tohoto vlákna dochází ke přepisování parametrů struktur track_cmd a track_stat, které v sobě mají uloženou konfiguraci sledovacího algoritmu, polohu sledovaného objektu, atd.

4.1.1.2 Vlákno ThrCapture

Toto vlákno má na starost zpracování příchozích dat z digitizéru analogového signálu, do nějž vstupují analogová video data pořizovaná kamerou. Základní funkce ProcThreadCapture kontroluje aktuální stav komunikace s digitizérem a konzistenci včetně platnosti

přijímaných dat. Přijímaná data jsou ukládána do zásobníku dat, z nějž se při ověření kompletnosti a správných rozměrů uloží do datového pole act_image.

Pořízená data jsou uložená jako třírozměrná matice o rozměrech šířka × výška × 3, kde každá ze tří vrstev obsahuje intenzity v obrazových bodech pro základní barvy RGB[8]. Tuto třívrstvou matici převedeme do matice jednovrstvé obsahující pouze odstíny šedi. Přepočet provedeme pomocí vzorce

$$H_{\text{šedá}} = 0,2989 \cdot R + 0,5870 \cdot G + 0,1140 \cdot B, \tag{33}$$

kde hodnota šedého obrazového bodu $H_{\text{šedá}}$ odpovídá postupnému sečtení jednotlivých hodnot obrazového bodu z jeho tří hodnot vyjadřujících barvu. Tímto výpočtem dostaneme matici obsahující intenzity obrazových bodů z načteného snímku v rozsahu 0 až 255, které uložíme do proměnné plmgDir.

Při použití programu pod operačním systémem Windows je zde k dispozici grafické uživatelské rozhraní, v němž je zobrazeno video pořizované kamerou (aktuální snímek act_image). Přes toto video (5) je zobrazena současná poloha sledovaného objektu (čtverec odpovídající velikosti sledovaného objektu) a stav stavového automatu (vyjádřeno barvou čtverce). V grafickém rozhraní jsou dále kontrolní tlačítka pro ovládání sledovacího algoritmu:

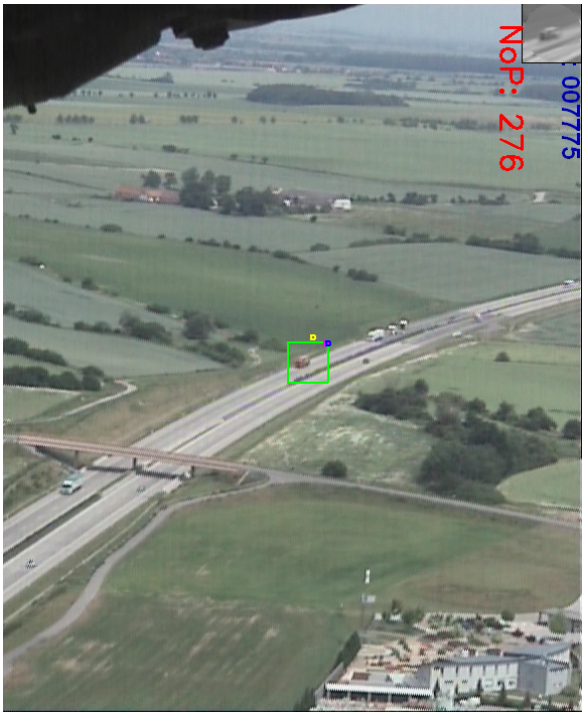
- Volba velikosti sledovaného objektu (velikost okolí)
- Zapnutí / vypnutí sledovacího algoritmu
- Inicializování kalibračního pohybu kamerové hlavice

4.1.1.3 Vlákna ThrCommTx a ThrCommRx

Tato dvojice vláken zajišťuje komunikaci sledujícího algoritmu pomocí rozhraní TrackerInterface s rozhraním CANInterface. Ve schématu (4) jim odpovídají hlavní funkce ProcThreadTx a ProcThreadRx. Rozhraní CANInterface obsahuje funkce pro obousměrnou komunikaci sledovacího algoritmu s řídícím systémem kamerové hlavice pomocí specifických zpráv pro čtení a zápis hodnot z CAN sběrnice. Toto čtení a zápis probíhá pomocí funkcí v souborech CANCanlab.cpp a CANKvaser.cpp pro použité převodníky USB/CAN. Funkce v těchto souborech obsahují upravené zprávy pro použití ovladačů třetích stran pro každý použitý převodník. Konkrétní převodník USB/CAN a jeho parametry nastavujeme v konfiguračním souboru pomocí proměnných:

- CAN_device - převodník *Kvaser* nebo *Canlab*
- CAN_port - přednastavené číslo komunikačního portu
- CAN_speed - nastavení komunikační rychlosti použité CAN sběrnice v KB/s

[8] R - červená (red), G - zelená (green) a B - modrá (blue)



Obrázek 5: Grafické uživatelské rozhraní se zobrazením sledovaného objektu. Zelená barva čtverce odpovídá stavu automatu 1 a 2, žlutá by odpovídala stavu 6, fialová 3 a červená stavům 4 a 5.

4.1.2 Využití knihovny OpenCV

Nově jsme pro práci s obrazem použili knihovnu *OpenCV*. Knihovnu *OpenCV* jsme zvolili z důvodu velkého množství implementovaných funkcí pro práci se snímky a kvůli přenositelnosti kódu mezi platformami (v našem případě mezi platformou využívající operační systém Windows a Linux).

Naším požadavkem byla možnost volby přenášet tento výsledný soubor mezi zařízeními bez nutnosti instalovat *OpenCV* knihovnu, používáme nikoliv dynamických, ale statických knihoven. Nutnost instalace *OpenCV* do počítače v kamerové hlavici by byla omezující, protože se jedná o nepříliš výkonné zařízení s malou úložnou kapacitou. Statické knihovny se při překladu kódu zahrnují do programu již při kompilování do spustitelného souboru, nikoliv jako knihovny dynamické, které jsou volány spuštěným souborem až při jejich prvním použití. Statické knihovny zvětší velikost výsledného souboru, ale zajistí, že pro spouštění programu se sledovacím algoritmem nebude třeba dodatečných souborů.

Pro tuto aplikaci jsme použili vícevláknovou realizaci, které spolu sdílí několik proměnných. Tyto proměnné používáme pro definování stavu algoritmu, komunikace s řídícím systémem a operátorem. Abychom zamezili vzájemnému přepisování těchto hodnot a problémům, používáme zamykání vybraných úseků kódu pro to vlákno, které bude přistupovat ke sdíleným proměnným a číst či měnit jejich hodnoty.

Pro naše řešení je použita verze *OpenCV* ve verzi 2.4.10 z října roku 2014.

4.1.3 Upravené schéma programu

Vycházíme z původní implementace *SSD* algoritmu a programu, do které zakomponováváme naší implementaci *KL* algoritmu. Z původního kódu *SSD* algoritmu používáme pouze funkce pracující s modelem *M* (natáčení, aktualizace při změně zoomu). Veškeré proměnné a datové typy jsme přepsali do takových, se kterými pracují funkce *OpenCV*. Do vnitřních funkcí této knihovny jsme nezasahovali, pouze jsme lokálně upravili funkce sledování objektu tak, aby splňovali naše požadavky.

V následujících odstavcích si porovnáme, jak se liší implementace původního *SSD* a *KLT* algoritmů sledování. Původní schéma programu (4 zůstává nezměněné. Vstupní proměnné upravených funkcí zůstávají shodné s původní implementací, liší se pouze v použitých datových typech.

4.1.3.1 Vlákno ThrTracking

Implementace *KLT* algoritmu se v tomto vlákně a vnořených funkcích liší v použitých datových strukturách pro uložení a práci se snímky, nepoužíváme tedy dynamicky alokovaná pole 8 bitových hodnot, ale struktury *Mat* z knihovny *OpenCV*. Přínosem této struktury jsou přidané obslužné funkce, sloužící mimo jiné k jednoduchému přístupu k rozměru uloženého snímku, kopírování dat tohoto snímku či jednoduchý výběr podoblasti snímku.

Funkce objectTrack zůstala prakticky nezměněná, liší se pouze tím, jak jsme použili model pro *KL* algoritmus. Předpokládali jsme, že pro sledování používáme z dvojice snímků jeden jako model a druhý pro vyhledání nové polohy sledovaného objektu. Při implementaci neprovádíme aktualizaci modelu přes celou jeho velikost, ale pouze přes okolí polohy sledovaného objektu v předchozím snímku. Tímto si ušetříme výpočetní kapacitu a zamezíme přepisování celé matice intenzit modelu.

Funkci LocateGrayModel jsme nahradili funkcí KLDiffImg_cv, která obsahuje námi upravenou implementací *KLT* algoritmu (funkce calcOpticalFlowPyrLK), která je součástí knihovny *OpenCV* (4.1.3.4).

4.1.3.2 Vlákno ThrCapture

V tomto vlákně používáme stejně jako v ostatních částech kódu strukturu *Mat* pro ukládání a práci se snímky. Vstupní snímek act_image je třírozměrná matice *Mat*, kterou pomocí funkce ImportRGB24_cv naplníme. Tuto matici pak pomocí funkce cvtColor z knihovny *OpenCV* převedeme do matice intenzit se stejnou šířkou a délkou.

Při použití na počítači s operačním systémem Windows zobrazujeme při zachování původní funkcionality také aktuální model sledovaného objektu v levém horním rohu grafického uživatelského rozhraní, obrázek (4).

4.1.3.3 Vlákno ThrCommTx a ThrCommRx

Komunikační vlákna zůstala shodná s původní implementací, pouze byla doplněna a přepsána tak, aby používala na platformě nezávislou knihovnu *Pthread* pro správu vláken a jejich komunikaci.

4.1.3.4 Funkce KLDiffImg_cv a calcOpticalFlowPyrLK

Implementace funkce KLDiffImg_cv nabízí možnost výběru typu sledovacího algoritmu *KL* dle počtu startovních pozic (4.2.1), které jsme zvolili v hlavičkovém souboru ConfigDef.h(4.4). Spuštění algoritmu z více startovních poloh (4.2.1) je implementováno pro hodnoty 1,9,25,36,49 a 81, ze kterých používáme nejjednodušší start z jedné polohy (shodné s původní implementací calcOpticalFlowPyrLK) a z 25 bodů. Tyto body jsou vybírány tak, aby byly rovnoměrně rozmístěny po celé prohledávané oblasti. Prohledávaná oblast je obdélník v konfiguračním souboru nastavitelná hodnotami proměnných search_height pro výšku a search_width pro šířku oblasti. Při spuštění algoritmu z jedné startovní polohy vrátíme souřadnici sledovaného objektu s minimální hodnotou chybové funkce (6). Pokud jsme vybrali start z více než jedné pozice, provádíme výběr takové nalezené souřadnice sledovaného objektu z uložených poloh, pro něž hodnota kriteriální funkce (6) minimální ze všech uložených hodnot pro jednotlivé souřadnice bodů. Jako návratovou hodnotu funkce KLDiffImg_cv pak nastavujeme polohu, pro níž byla chybová funkce minimální.

Upravená funkce calcOpticalFlowPyrKL umožňuje upravení zvlášť velikosti okolí sledovaného objektu a prohledávané oblasti pro nalezení souřadnic objektu. V průběhu funkce dojde k vypočítání pyramidové konstrukce pro oba vstupní snímky, po které dojde ke spuštění gradientního algoritmu. Algoritmus je spouštěn ze zadaného počtu bodů rozmístěných v prohledávané oblasti. Pro každý bod proběhne minimalizační gradientní algoritmus, který vrátí souřadnici objektu s minimální hodnotou chybové funkce (6) po skončení iterování. Pro všechny body si uložíme hodnotu chybové funkce a nalezené souřadnice.

4.1.4 Načítání ze souboru

Pro potřeby testování algoritmu *KLT* i *SSD* jsme přidali možnost načítání sekvencí snímků ze složky s lokálně uloženými daty. Při inicializaci programu je potřeba zvolit pouze cílovou složku v počítači, počáteční snímek sekvence a polohu sledovaného objektu v souřadnicích vzhledem k levému hornímu rohu snímku. Tyto parametry nastavíme v konfiguračním souboru Ini_parameter.par pomocí proměnných data_path, tracking_col a tracking_row. Proměnné data_path předáme cestu k prvnímu snímku načítané sekvence v absolutní podobě (data_path = C:\Data\Sekvence\01-00%003d.png) či v relativní (data_path = ..\.\Sekvence\01-00%003d.png). Zadání prvního snímku sekvence probíhá pomocí konstrukce 01-00%003d.png, která se skládá ze začátku názvu (01-00, který se u každého snímku opakuje), z pořadového čísla snímku sekvence (%003d)[9] a dále datového typu snímku (.png). U sekvencí předpokládáme, že jsou snímky číslované od čísla 1 a v číselné sekvenci jejich pořadí nejsou mezery. Zadání počáteční polohy sledovaného objektu provádíme vzhledem k prvnímu snímku sekvence. Proměnná tracking_row (resp. tracking_col) obsahuje řádkovou (resp. sloupcovou) souřadnici sledovaného objektu v celých číslech. Oba algoritmy poté od prvního snímku sledují vybraný

[9] Zadáním této konstrukce říkáme, že chceme, aby program dále načítal snímky s číslem od 1(resp. 0) do 999. Tato konstrukce lze dle potřeby upravit, tvoří se podobně jako formátovaný výstup v programovacím jazyce C++.

objekt napříč celou zvolenou sekvencí, dokud sekvence neskončí či nedojde ke ztrátě sledovaného objektu. Načítání snímků probíhá pomocí funkce (*sequence*) z knihovny *OpenCV*, která načte celou sekvenci snímků a z níž postupně vyčítáme snímky do struktury *Mat* patřící knihovně *OpenCV*.

4.2 Další úpravy algoritmu

4.2.1 Vyhledávání z více pozic

Při rychlém pohybu kamery i sledovaného objektu může objekt mezi snímky vykonat velký pohyb (posun) a *KL* algoritmus sledování by mohl objekt prohlásit za ztracený. Dále ke ztrátě může dojít, pokud se sledovaný objekt překrývá s aktuálním modelem z menší části jak 50 procent délky modelu, pokud je v dostatečné blízkosti aktuální polohy dostatečně podobný objekt námi sledovanému. Abychom takto způsobenou ztrátou sledovaného objektu zabránili, rozšířili jsme *KLT* algoritmus tak, aby predikci počáteční polohy ve snímku nebral pouze polohu sledovaného objektu ve snímku předchozím. Rozšíření spočívá v tom, že jako predikovanou polohu volíme postupně z více bodů rovnoměrně rozmístěných po celé prohledávané oblasti v předchozím snímku. Každá tato startovní predikovaná poloha po dokončení iterování (3.1.3) jako výsledek vrátí polohu s minimální hodnotou kriteriální chybové funkce (13). Z těchto všech vrácených minim, která ve většině případů odpovídají stejné poloze, vybereme to s nejnižší hodnotou. Díky zvětšení počtu startovních poloh pro predikci můžeme předpokládat, že sledovací algoritmus neskončí v lokálním minimu, ale v minimu globálním.

4.2.2 Použití pohyblivé desetinné čárky

Oproti implementaci *SSD* používáme při sledování a práci se snímky souřadnic s pohyblivou desetinnou čárkou, namísto původních celočíselných souřadnic. Toto zvýšení přesnosti zlepšilo chování sledovacího algoritmu při velmi malých posunech mezi snímky, což způsobovalo opakované změny polohy v rozsahu jednoho obrazového bodu. Takovéto časté a malé změny vyvolávaly zbytečné odchylky pro řídící systém.

4.3 Implementace pro Linux

V rámci práce bylo také cílem upravit algoritmus i samotný program tak, aby byl spustitelný na řídící jednotce kamerové hlavice, která funguje s operačním systémem Linux. Došlo tedy k oddělení původních vláken aplikace a vytvoření vláken nových, s využitím knihovny *Pthread* [7], poskytující Linuxovou správu vláken do operačního systému Windows. Dále došlo k úpravě některých datových typů a funkcí, které jsou vázané na operační systém Windows tak, aby výsledný kód mohl být přeložitelný na zařízení s OS Linux. Pro následné spuštění algoritmu přímo na kamerové hlavici je třeba vždy kód upravit tak, aby odpovídal potřebám použitého HW vybavení (komunikace programu a HW, atd.).

Verze překládaná pod operačním systémem nemá, na rozdíl od verze pro OS Windows, grafické uživatelské rozhraní a program běží pouze v příkazové řádce (terminálové okno pro OS Linux). Toto snižuje výpočetní a grafické nároky na HW vybavení v kamerové hlavici, která pak může komunikovat pouze pomocí textového výpisu. Veškeré nastavování probíhá čtením dat z konfiguračního souboru nebo konfiguračními zprávami z terminálu operátora.

4.4 Překlad

Implementaci navrženého algoritmu a změn jsme provedli tak, aby byla od původního kódu kompletně oddělena a my jsme si mohli při kompilování zvolit, který algoritmus a jaké jeho varianty si přejeme použít. Tato možnost výběru je umožněna díky hlavičkovému souboru ConfigDef.h. Tento výběr můžeme učinit vybráním proměnné klíčovým slovem #define:

- *OFFLINE_TRACK* - slouží k načítání připravených sekvencí z uložených dat na disku počítače specifikovaných v souboru Ini_parameters.par (4.1.4)
- *KL_TRACKER* - použití navrženého *KL* algoritmu namísto původního *SSD*
- *TIME_MEASURE* - měření doby průběhu algoritmu sledování a uložení naměřených hodnot do souboru timemeasure.txt
- *ALLOWED_ROTATE_MODEL* - povolení dodatečného natáčení modelu po nalezení polohy sledovaného objektu pro přesnější aktualizování modelu objektu (použitelné pro algoritmus *SSD*)
- *KL_TRACKER_i* - zvolení počtu startovacích pozic algoritmu pro nalezení souřadnic objektu ve snímku (4.2.1). Implementováno pro hodnoty $i = 1, 9, 25, 36, 49, 81$, používáme pro hodnoty 1 a 25.
- *OPEN_CV_TRACKER* - pro použití knihovny *OpenCV*, pro *KL* algoritmus je nutné použít. Slouží k přepínání původní implementace *SSD* algoritmu v jazyce C/C++ a implementace s použitím knihovny *OpenCV*.

Využitím direktiv preprocesoru v jazyce C++ (#define, #ifdef, atd.) pro tvorbu podmíněných překladů (v závislosti na volbě parametru z výše uvedeného seznamu) snížíme velikost výsledného programu po kompilace zdrojových kódu s více algoritmy.

4.5 Výběr vhodného objektu ke sledování

Jako sledovaný objekt nejčastěji chceme zvolit takový, který se dostatečně odlišuje od svého okolí a díky čemuž bude dobře sledovatelný. Při souběžném pohybu kamerové hlavice i objektu pro operátora u terminálu není jednoduché přesně zvolit objekt, co chce sledovat[10]. Aby algoritmus sledoval opravdu objekt, který chtěl operátor a ne

[10]Objekt ke sledování operátor vybírá postupným natáčením kamery tak, aby byl objekt uprostřed zorného pole kamery, nebo přímou volbou objektu na dotykovém displeji terminálu.

pouze jeho část s okolím, použijeme metodu popsanou v článku [8]. Článek vychází z toho, že použití pouze hranového detektoru pro nalezení nejlépe sledovatelného objektu není dostatečné, protože např. objekt s pouze svislými hranami nemusí být dobře sledovatelný. V článku je tato metoda rozšířená o výpočet „rozdílnosti“ bodů zájmu mezi dvojicí snímků, zkouší tedy sledovat detekované rohy z prvního snímku do snímku druhého. Tato metoda vybere nejlépe sledovatelnou oblast ve snímku, tedy takovou, kterou bychom chtěli pravděpodobně sledovat. Taková oblast nemusí být pouze jedna, ale ve snímku jich může být několik.

Pro náš kód jsme zvolili implementaci v knihovně *OpenCV*, kterou jsme upravili tak, aby nám vyhovovala. Tato upravená metoda při zadání počáteční polohy sledovaného objektu prohledá jeho nejbližší okolí (10×10 obrazových bodů od startovní polohy) a vybere takovou oblast, která bude nejlépe sledovatelná v algoritmu. Poloha této oblasti s vysokou pravděpodobností bude představovat náš zvolený objekt ke sledování.

Na obrázku (6) porovnáváme volbu sledovaného objektu operátorem (červený čtverec) a nejlépe sledovatelnou oblast zvolenou algoritmem (zelený čtverec).

Obrázek 6: Volba sledovaného objektu pomocí funkce goodFeaturesToTrack, červený čtverec představuje volbu operátora a zelený čtverec upravené souřadnice objektu.

Kapitola 5
Experimentální výsledky (testování algoritmu)

Implementovaný algoritmus jsme testovali na 16 sekvencích snímků pořízených z videozáznamů, které byly pořízeny při testovacích letech kamerové hlavice při zavěšení na spodní části vrtulníku. V těchto sekvencích je vždy sledován objekt zájmu, u nějž předpokládáme dostatečnou odlišitelnost od pozadí a který se pohybuje poblíž středu snímku.

Použité snímky jsou uloženy ve formátu PNG pro zachování vyšší kvality snímků. Rozlišení snímků je dáno použitou kamerou 704×576 obrazových bodů. Sekvence jsou uložené v oddělené složce na počítači Data\Sekvence a jsou číslovány ve tvaru *číslo sekvence-pořadí snímku*. Pro každou sekvenci je důležité dodržení inkrementace pořadí snímku o 1 a čísla (názvu) sekvence.

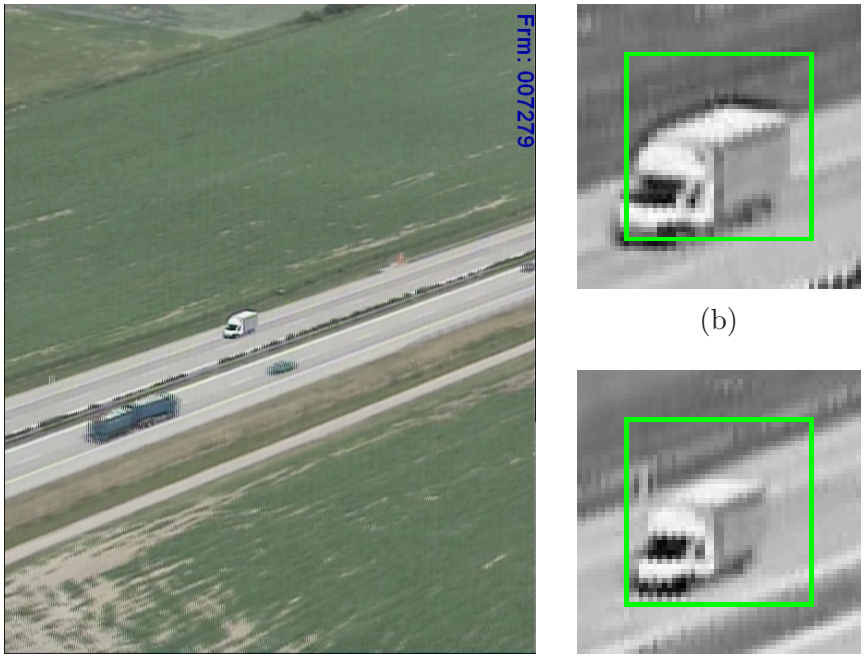
Algoritmus pro porovnání algoritmů *SSD* a *KLT* načítá sekvence po jednom snímku. Z každé sekvence je na obrázku (např. 7) zobrazen sledovaný objekt (resp. jeho model), který jsme inicializovali na počátku sekvence. Sledovali jsme např. jedoucí automobil, část tepelné elektrárny, dům na samotě, hrad a osoby. Kromě sekvencí s jedoucími automobily jsou všechny objekty stacionární a k pohybu dochází pouze vlivem pohybující se kamery. U sekvencí s jedoucími automobily dochází k pohybu sledovaného objektu i kamery.

Velikost sledovaného objektu používáme čtverec 40×40 obrazových bodů, model sledovaného objektu (okolí polohy objektu, které aktualizujeme) používáme 60×60 obrazových bodů a velikost prohledávané oblasti, v níž hledáme sledovaný objekt je čtverec o rozměrech 50×50 obrazových bodů. S rostoucí velikostí modelu a prohledávané oblasti kvadraticky stoupá náročnost algoritmu a tím i doba běhu algoritmu. Při zvolení menší velikosti sledovaného objektu, dosáhneme výrazné časové i výpočetní úspory, ale jsme schopni pomocí algoritmu sledovat pouze objekty srovnatelně veliké jako je nastavená velikost okolí polohy objektu.

Pro aktualizování modelu sledovaného objektu dle rovnice (3) používáme hodnotu konstanty $\alpha = 0,01$. Tato hodnota odpovídá velmi pomalému zapomínání původního modelu (mění se pouze 1% modelu). Pro případy námi sledovaných objektů předpokládáme, že jejich velikost, struktura a tvar se nebudou v čase rychle měnit.

5.1 Sekvence

V sekvencích snímků je sledován různý druh zvolených objektů pro ověření kvality algoritmu. Pro testování algoritmu bylo použit 7 videozáznamů ze zkušebních letů vrtulníku s kamerovou hlavicí. Záznamy jsme rozdělili do sekvencí snímků, které obsahují všechny



(a) (b) (c)

Obrázek 7: Sekvence číslo 1: (a) úvodní snímek, (b) sledovaný objekt na začátku sekvence, (c) sledovaný objekt na konci sekvence (zeleně označena sledovaná část obrazu)

snímky. Z těchto sekvencí jsme poté použili kratší úseky (porovnávané sekvence s čísly 1 až 10) a delší úseky (sekvence s čísly 11 až 16).

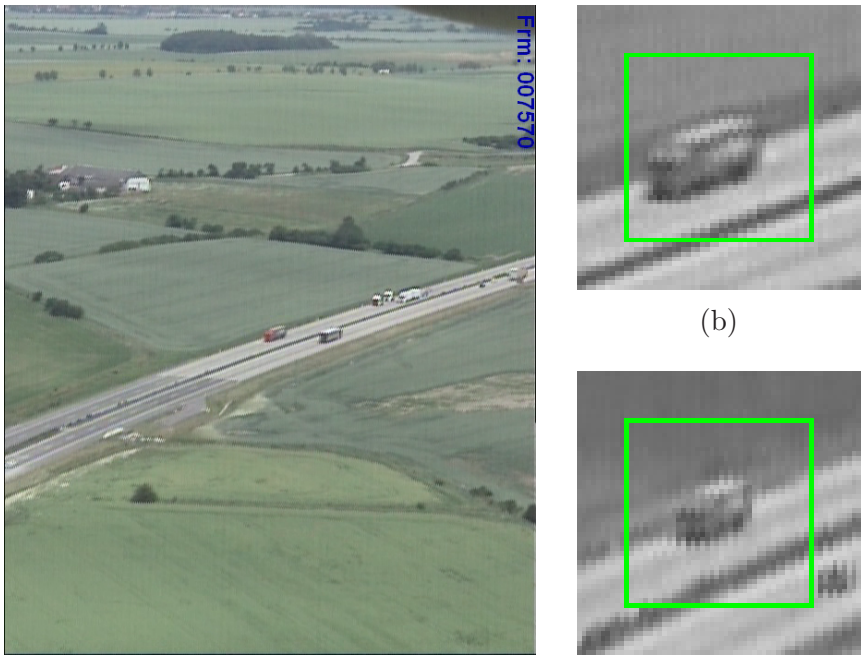
Sekvence s čísly 1,2 a 12 jsou vybrány jako úseky shodného videozáznamu. V těchto sekvencích sledujeme automobil (pro sekvence 2 a 12 shodný) jedoucí po rychlostní komunikaci. V průběhu sekvencí kamera sleduje směr jedoucího automobilu, vzdálenost se ale kvůli vyšší rychlosti pohybu kamery vůči automobilu postupně zvětšuje. V sekvenci (obrázek 7) sledujeme bílý nákladní automobil, v sekvencích 2 a 12 sledujeme vozidlo dálkové přepravy (obrázek 8).

V sekvencích 3,4 a 14 je použit videozáznam průletu vrtulníku okolo tepelné elektrárny (obrázek 9). Jako sledovaný objekt jsme zvolili patu chladících komínů u okraje budovy. Sekvence 3 a 4 obsahují část průletu okolo elektrárny, v sekvenci 14 je opakovaný průlet se změnou zoomu.

Sekvence 5 a 15 sledují dům obklopený loukami (obrázek 10), který je velmi dobře odlišitelný od pozadí snímku. Sekvence 5 je kratší část sekvence 15, v níž dochází k postupnému oddalování sledovaného objektu.

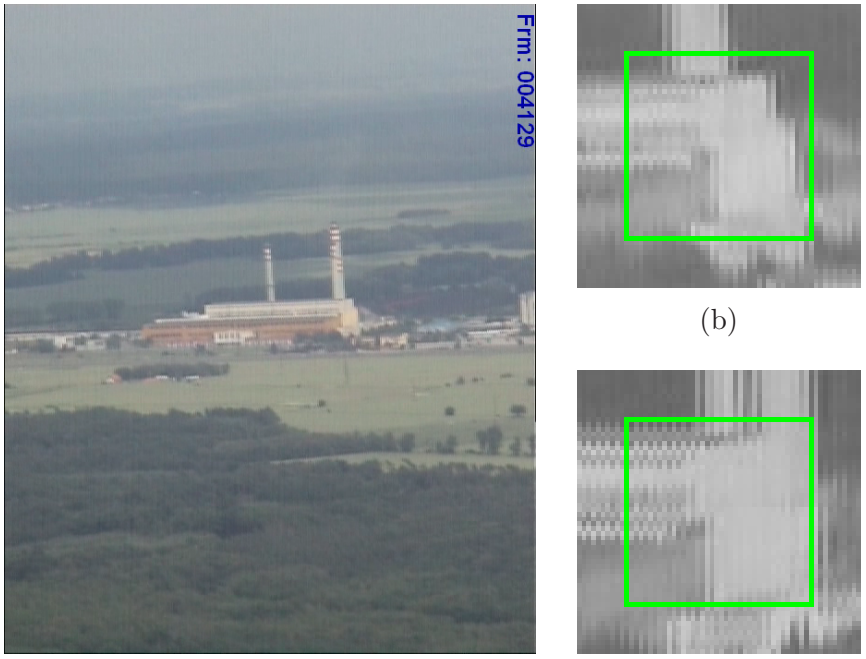
V sekvencích 6 a 16 kamera prolétá okolo hradu na kopci (obrázek 11), který nesledujeme celý kvůli jeho velikosti, ale pouze levou stěnu. V sekvenci 16 dochází ke změně použitého zoomu a tím ke změně sledovaného velikosti objektu.

Sekvence 7,8,10 a 11 sledují osoby, které jsou velmi dobře odlišitelné od okolního prostředí. Sekvence 7 a 8 jsou úseky shodného videozáznamu (obrázek 12), v němž došlo k významné změně intenzity osvětlení scény. Tato změna se vyskytuje právě mezi těmito sekvencemi a přes tuto skokovou změnu nešlo provádět sledování vybraného objektu.



(a) (b) (c)

Obrázek 8: Sekvence číslo 2 a 12: (a) úvodní snímek, (b) sledovaný objekt na začátku sekvence, (c) sledovaný objekt na konci sekvence (zeleně označena sledovaná část obrazu)

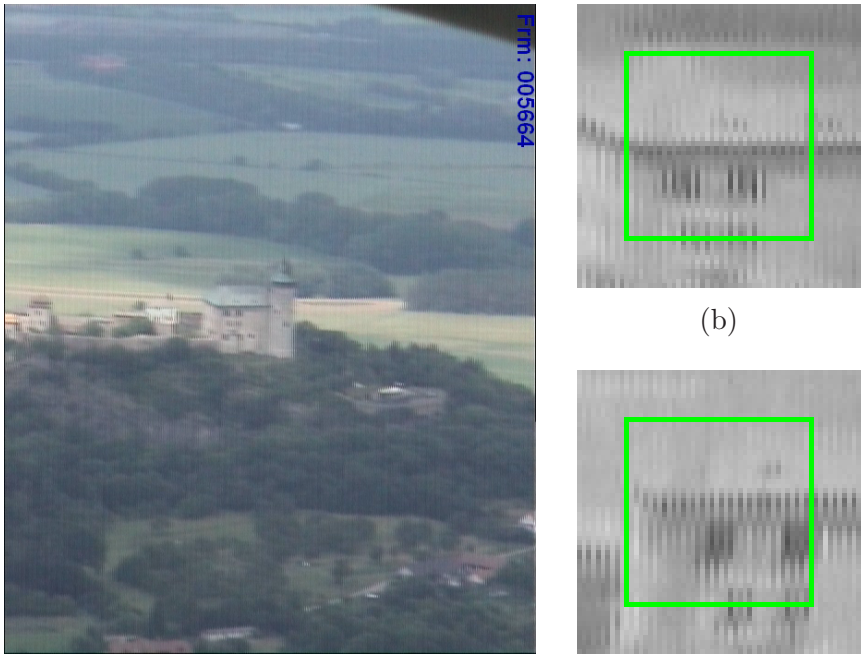


(a) (b) (c)

Obrázek 9: Sekvence číslo 3 a 14: (a) úvodní snímek, (b) sledovaný objekt na začátku sekvence, (c) sledovaný objekt na konci sekvence (zeleně označena sledovaná část obrazu)

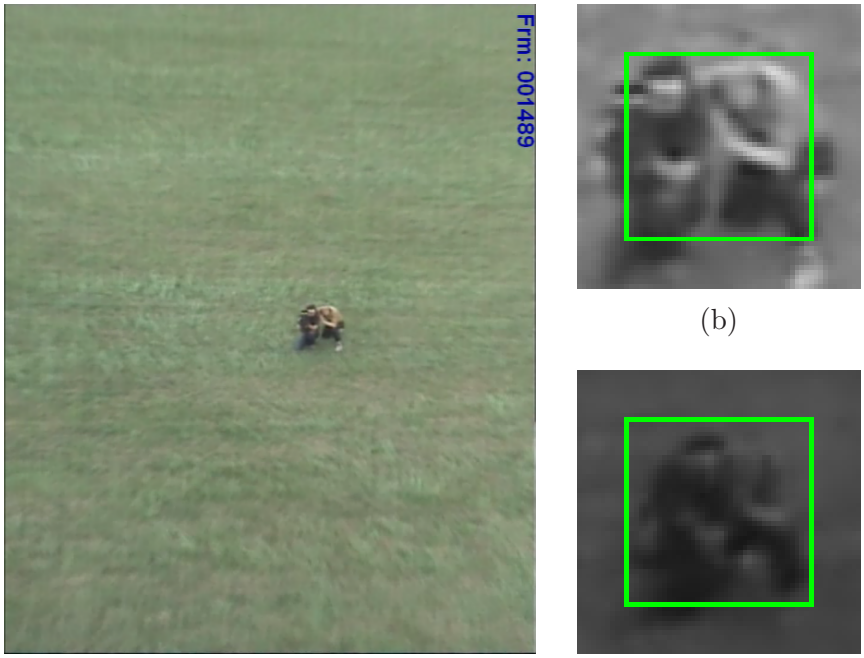
(a) (b) (c)

Obrázek 10: Sekvence číslo 5 a 15: (a) úvodní snímek, (b) sledovaný objekt na začátku sekvence, (c) sledovaný objekt na konci sekvence (zeleně označena sledovaná část obrazu)



(a) (b) (c)

Obrázek 11: Sekvence číslo 6 a 16: (a) úvodní snímek, (b) sledovaný objekt na začátku sekvence, (c) sledovaný objekt na konci sekvence (zeleně označena sledovaná část obrazu)



(a) (b) (c)

Obrázek 12: Sekvence číslo 7 a 8: (a) úvodní snímek, (b) sledovaný objekt na začátku sekvence, (c) sledovaný objekt na konci sekvence (zeleně označena sledovaná část obrazu)

Sekvence 10 (obrázek 13) je vybrána jako stabilní část sekvence 11, v níž dochází ke změně zoomu kamery.

V sekvenci 9 (obrázek 14) sledujeme budovu v prostorách letiště, přes níž je prováděn přelet vrtulníku s kamerou. Díky průletu dochází ke změně velikosti sledovaného objektu a kvůli rychlosti letu i k chvění pořízeného videozáznamu.

Sekvence s číslem 13 (obrázek 15) je odlišná od všech ostatních, sledujeme v ní vysokou chladící věž elektrárny, která je obklopena dalšími věžemi. V průběhu sekvence dochází k opakované změně zoomu kamery a pohybu sledovaného objektu. Jako sledovaný objekt volíme hranu vrcholu chladící věže v jejímž okolí se nacházejí vizuálně podobné objekty.

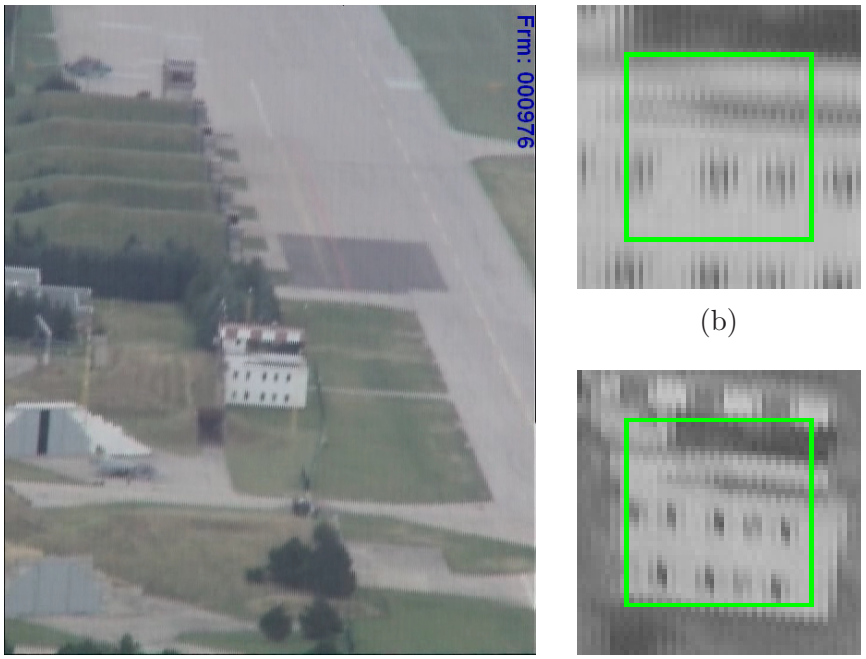
5.2 Porovnání algoritmů

Porovnání funkčnosti algoritmu *SSD* a *KL* je vidět v tabulce (1). *KL* algoritmus sledování je porovnáván při spouštění z jedné startovní pozice a z 25 pozic. Těchto 25 pozic je rovnoměrně rozmístěno v prohledávané oblasti. Tato prohledávaná oblast pro *KL* je shodná s prohledávanou oblastí *KL* algoritmu.

U *KL* algoritmu v sekvenci 1 na obrázku (7) došlo ke ztrátě sledovaného cíle tím, že v jeho těsné blízkosti projel automobil s velmi podobnou strukturou jako sledovaný objekt. V sekvenci číslo 7 došlo ke ztrátě sledovaného objektu v důsledku výrazné

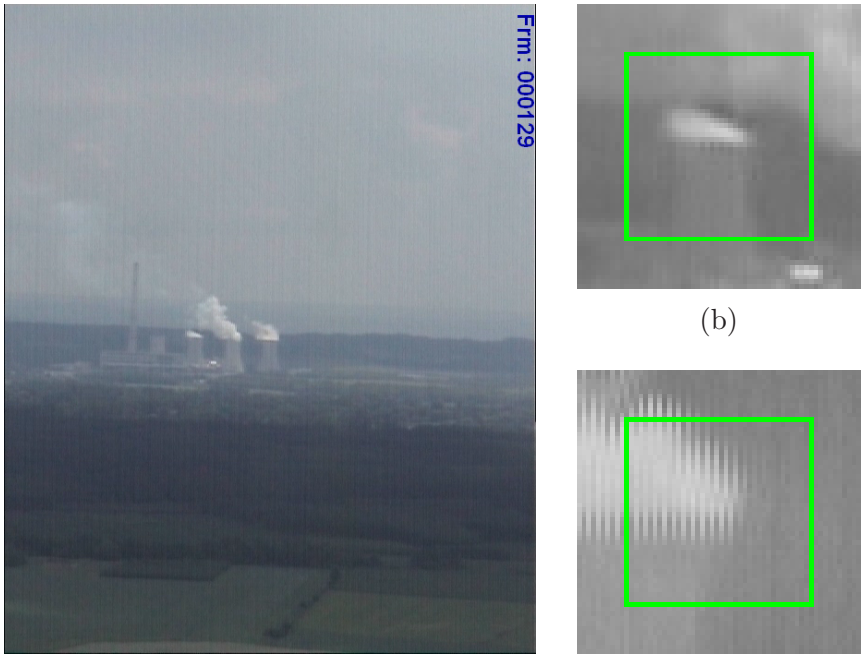
(a) (b) (c)

Obrázek 13: Sekvence číslo 10 a 11: (a) úvodní snímek, (b) sledovaný objekt na začátku sekvence, (c) sledovaný objekt na konci sekvence (zeleně označena sledovaná část obrazu)



(a) (b) (c)

Obrázek 14: Sekvence číslo 9: (a) úvodní snímek, (b) sledovaný objekt na začátku sekvence, (c) sledovaný objekt na konci sekvence (zeleně označena sledovaná část obrazu)



(a) (b) (c)

Obrázek 15: Sekvence číslo 13: (a) úvodní snímek, (b) sledovaný objekt na začátku sekvence, (c) sledovaný objekt na konci sekvence (zeleně označena sledovaná část obrazu)

skokové změny scény ve snímku mezi snímky. Pro sekvence $10 - 16$ poskytuje *KL* algoritmus s 25 startovními polohami lepší výsledky než *SSD* algoritmus, dochází zde ke kmitavým (opakujícím se tam a zpět) změnám polohy objektu a zkreslení celého snímku, oproti čemuž je *KLT* algoritmus robustnější. Pro sekvenci 14 je algoritmus *KL* s 25 startovními polohami lepší a s jednou startovní polohou horší než algoritmus *SSD*. Tento rozdíl je způsoben změnou zoomu kamery a tím i velikosti sledovaného objektu.

Zvolené testovací sekvence mají odlišné délky (počty snímků), sekvence s čísly 1-10 jsou kratší části vybrané z videí, sekvence s čísly 11-16 obsahují delší úseky vybrané z videí pořízených kamerou.

V tabulce (1) porovnáváme číslo snímku, v němž algoritmus sledovaný objekt ztratil a čas, který v průměru potřeboval na jeden průběh sledovacího algoritmu. Jako jeden průběh uvažujeme načtení dvojice po sobě jdoucích snímků, inicializace algoritmu ze startovní polohy, výpočet polohy ve snímku z minimální hodnoty (minimálních hodnot) kriteriální chybové funkce (6). Pokud algoritmus skončil dříve, než je délka sekvence, průměrná doba běhu je počítána z časů do doby ztracení objektu.

Objekt považujeme za ztracený, pokud při jeho sledování došlo ke vzdálení se predikované polohy od sledovaného objektu o více než 20 obrazových bodů bez návratu na správnou souřadnici ve snímku v průběhu následujících 3 snímků. Pokud algoritmus oznámí ztrátu sledovaného objektu, dochází k pozastavení aktualizace modelu objektu a po 5 sekundách bez opětovného nalezení objektu se algoritmus přepne do stavu 5. Ověření, že je v testovaných sekvencích stále sledován vybraný objekt na správných souřadnicích, provádí člověk.

Podstatným parametrem pro porovnání algoritmů je rychlost algoritmu při výpočtu polohy sledovaného objektu v novém snímku. Měření doby průběhu algoritmu jsme prováděli pomocí funkcí QueryPerformanceCounter a QueryPerformanceFrequency. Funkce QueryPerformanceCounter jako návratovou hodnotu vrací obsah čítače výkonu, kterou si poprvé uložíme při startu algoritmu a podruhé při navrácení predikované polohy sledovaného objektu. Pokud rozdíl těchto dvou hodnot vydělíme frekvencí procesoru, na němž probíhal výpočet[11](kterou vrací funkce QueryPerformanceFrequency), získáme dobu běhu algoritmu v mikrosekundách. Přesnost funkce QueryPerformanceCounter je v dokumentaci uváděna pod $1\mu s$, díky čemuž můžeme považovat naměřené hodnoty za dostatečně přesné. Dvojice použitých funkcí je použitelná pouze při běhu programu na operačním systému Windows (využívají soubor windows.h). Pro operační systém nepředpokládáme výrazný rozdíl měřených časů, používáme minimum funkcí závislých na konkrétním operačním systému a výsledky v tabulce (1) považujeme za platné pro Windows i Linux.

Výpočet naměřených hodnot jsme prováděli tak, že jsme měřili pro každý snímek dobu průběhu algoritmu zvlášť a ze všech hodnot pro snímky, v nichž byl objekt úspěšně sledován, jsme vypočítali průměr. Očekáváme, že při zakomponovávání *KL* algoritmu do původního programu se doba běhu zbylých částí programu výrazně neměnila a tudíž pro porovnávání neměla vypovídající hodnotu.

5.3 Zhodnocení výsledků

Porovnáme-li dobu jednoho průběhu *SSD* a *KLT* algoritmu s jedním startovním bodem, pak je *KLT* algoritmus výrazně rychlejší. Výrazná rozdílnost je způsobena tím, že *SSD* algoritmus je kompletní a *KL* je algoritmem gradientním. Při startu z 25 bodů je doba běhu algoritmu *KL* srovnatelná s *SSD* algoritmem. *SSD* algoritmus používá při výpočtu minimální hodnoty chybové funkce poloviční rozlišení snímku i modelu. Tohoto snížení velikosti snímku a modelu na čtvrtinu originální velikosti je použito pro zvýšení rychlosti algoritmu s tím, že počítáme se snížením přesnosti algoritmu pro určení nové polohy objektu na polovinu. Pokud bychom používali pro *SSD* algoritmus snímky v plném rozlišení, došlo by ke znatelnému zpomalení a snížení počtu snímků, v nichž jsme za vteřinu schopni nalézt sledovaný objekt. Pro algoritmus *KLT* používáme snímky v plném rozlišení (704×576). Průměrné doby běhu jednoho kroku uvedené v tabulce jsou uvedeny pro *SSD* se zmenšenými snímky a pro *KLT* s originální velikostí snímků.

Při porovnání *KL* algoritmu s 1 a 25 startovními polohami se čas průběhu navýšil pouze dvakrát. Nejvyšší výpočetní náročnost představuje předběžné vypočítání pyramidové konstrukce pro dvojici snímků. Následné vyhledání minimální hodnoty kriteriální chybové funkce pro jednotlivý startovní bod trvá přibližně $2ms$.

Z tabulky (1) můžeme vidět, že délka úspěšného sledování objektu pomocí algoritmu *KLT* je srovnatelná s *SSD* algoritmem. V 6 sekvencích z 15 je *KL* algoritmus (1 a 25 startovních pozic) stejně úspěšný při sledování vybraného objektu jako algoritmus *SSD*.

[11]Pro více jádrové procesory funkce vrací shodné hodnoty. Při testování jsme používali procesor Intel Core i3 s taktovací frekvencí 2,13GHz.

Pro 2 sekvence z 15 je *KL* algoritmus horší než *SSD* algoritmus sledování. *KL* algoritmus v 7 sekvencích sleduje objekt déle než *SSD* algoritmus, z nichž ve 3 sekvencích je rozdíl v délce úspěšnosti sledování jednoznačný (sekvence 11, 13 a 16).

Testování probíhalo nejenom na datech uložených v počítači, ale ověřili jsme funkčnost i na externí kameře. Použitá kamera je *Mintron OS-45D*, jejíž analogový obraz prochází do počítače skrze digitizér videosignálu *Sensoray 2255S* v rozlišením 704 × 576 obrazových bodů. Ověřování probíhalo sledováním shodných objektů v reálném prostředí, v němž výsledky byly srovnatelné pro trojici porovnávaných algoritmů. Doby průběhů algoritmů zachovávají shodné poměry jako v tabulce (1).

číslo	délka	SSD		KL(1)		KL(25)	
sekvence	sekvence	snímků	[ms]	snímků	[ms]	snímků	[ms]
1	142	142	69,71	133	31,74	94	77.74
2	191	191	69,62	191	31,63	191	67,56
3	151	135	71,11	151	32,06	151	65,27
4	191	191	73,58	191	32,20	191	63,19
5	151	151	69,84	151	31,45	151	63,28
6	81	81	69,23	81	32,38	81	65,12
7	128	116	70,07	117	31,44	76	62,76
8	130	130	71,19	130	31,63	130	64,77
9	112	102	72,53	112	32,23	112	69,70
10	171	70	73,80	171	31,99	171	66,25
11	108	40	70,96	75	31,64	108	66,05
12	276	85	69,85	276	31,87	276	70,25
13	679	289	70,86	355	31,84	679	69,55
14	659	560	69,17	480	32,03	640	70,47
15	321	321	69,93	321	31,93	321	65,39
16	329	15	72,47	90	32,48	329	73,10

Tabulka 1: Porovnání funkčnosti algoritmů *SSD* a *KL* (start z 1 a 25 bodů). Počet snímků odpovídá, jak dlouho algoritmus v dané sekvenci sledoval objekt. Čas v $[ms]$ je průměrná doba běhu algoritmu na jeden snímek.

Kapitola 6
Závěr

V průběhu práce jsme navrhli algoritmus založený na metodě popsané [1], který vybrali kvůli nižším výpočetním nárokům oproti stávajícímu a používanému algoritmu založeném na metodě *SSD*. Při návrhu jsme vycházeli z dalších článků ([2], [9], [10], [11] a [8]), návrh jsme upravili tak, aby vyhovoval požadavkům pro použití na kamerové hlavici *M120* projektu *MAGYSKA*.

Implementaci navrženého algoritmu jsme provedli v programovacím jazyce C++ s využitím knihovny *OpenCV* a jejích datových typů a struktur, z nichž jsme využili části některých funkcí, které jsme si upravili. Mezi tyto úpravy patří například upravení souřadnic sledovaného objektu po zadání operátora (dle metody [8]), zpřesnění výsledků algoritmu použitím reálných čísel (namísto celých) v souřadném systému snímku a možnost načítat snímky z připravených sekvencí. Původní algoritmus *SSD* jsme přepsali tak, aby využíval knihovny *OpenCV*.

Implementaci navrženého i původního algoritmu jsme předělali tak, aby programový kód byl přeložitelný na platformách operačních systémů Windows a Linux. Operační systém Windows byl použit pro testování algoritmu na stolním počítači s použitím připravených sekvencí snímků. Upravený kód a výsledný program pro operační systém Linux bude použit pro instalaci na výpočetní jednotku kamerové hlavice. Pro tuto úpravu jsme použili knihovnu *Pthread* [7] pro správu vláken a mezivláknovou komunikaci.

Testování algoritmu jsme prováděli na připravených sekvencích snímků z videozáznamů a externí kameře připojené k počítači. Při porovnání algoritmů *SSD* a *KL* jsme ověřili, že *KL* algoritmus je stejně rychlý jako *SSD* (případně rychlejší v závislosti na počtu startovních poloh). Délka sledování objektu v testovacích sekvencích byla pro algoritmus *KL* shodná či delší ve většině případů v porovnání s *SSD*. Dodržením požadavků na zvýšení rychlosti sledovacího algoritmu a zachování shodné (či delší) doby sledování jako při použití algoritmu *SSD* bylo dle našeho názoru splněno zadání této práce.

Ověření funkčnosti a použitelnosti navrženého algoritmu proběhne v blízké době na kamerové hlavici M120 po dodatečné úpravě kódu pro konkrétní použité technické vybavení hlavice.

Literatura

[1] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, pages 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc.

[2] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifying framework. *International Journal of Computer Vision*, 56(3):221–255, 2004.

[3] P. Montesinos, V. Gouet, and R. Deriche. Differential invariants for color images. In *Pattern Recognition, 1998. Proceedings. Fourteenth International Conference on*, volume 1, pages 838–840 vol.1, Aug 1998.

[4] T.S.Subashini S.Bhuvaneswari. Tracking manually selected object in videos using color histogram matching. *Journal of Theoretical and Applied Information Technology*, 67(3), 30 September 2014.

[5] Jean-Yves Bouguet. Pyramidal implementation of the lucas kanade feature tracker description of the algorithm. *OpenCV Document, Intel Microprocessor Research Labs*, 1999. Technical report.

[6] Petr Vávra. Detekce zákrytu při vizuálním sledování algoritmem SSD, 2012, Praha. Bakalářská práce, ČVUT v Praze, Fakulta elektrotechnická, Katedra kybernetiky.

[7] J. Bossom, A. Terekhov, L. Thomas, T. Pfaff, and R. Johnson. Open Source POSIX Threads for Win32, 5 2012. https://sourceware.org/pthreads-win32/.

[8] Jianbo Shi and Carlo Tomasi. Good features to track. In *1994 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, pages 593 – 600, 1994.

[9] Baojie Fan, Yingkui Du, Linlin Zhu, Jing Sun, and Yandong Tang. A robust template tracking algorithm with weighted active drift correction. *Pattern Recognition Letters*, 32(9):1317 – 1327, 2011.

[10] I. Matthews, T. Ishikawa, and S. Baker. The template update problem. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(6):810–815, June 2004.

[11] Vaclav Hlavac Milan Sonka and Roger Boyle. *Image Processing, Analysis and Machine Vision*. Thomson, 3rd edition, 2007. ISBN 978-0-495-08252.

Obsah DVD

K této diplomové práci je přiloženo DVD, na kterém je elektronická verze odevzdané práce ve formátu PDF, zdrojové kódy programu v jazyce C++, sekvence snímků, videozáznamy z nichž byly vytvořeny sekvence snímků a spustitelný program pro prostředí Windows s konfiguračním souborem.

Složky na přiloženém médiu s obsahem:

- PDF - diplomová práce ve formátu PDF
- Program - 2 spustitelné soubory SPTtracker_online.exe a SPTracker_offline.exe zkompilované pro start s 25 body pro spuštění s připojenou kamerou, resp. s uloženými sekvencemi. Konfigurační soubor Ini_parameters.par je přiložen.
- Sekvence - sekvence snímků ve formátu PNG použité pro testování
- Videa - videozáznamy pořízené kamerou, z nichž byly sekvence snímků vytvořeny
- Zdrojové kódy - *.cpp a *.h zdrojové kódy programu a projektové soubory pro vývojové prostředí *Visual Studio 2013*