České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídící techniky

DIPLOMOVÁ PRÁCE

Programové vybavení pro řízení průzkumných kamer a zpracování videosignálu

Autor: Bc. Martin Zmrhal

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

Praha, 2014

Název práce: Programové vybavení pro řízení průzkumných kamer a zpracování videosignálu

Autor: Bc. Martin Zmrhal

Katedra: Katedra řídící techniky

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

E-mail vedoucího: krsek@fel.cvut.cz

Anotace: V předložené práci je pojednáváno o vývoji softwarové knihovny funkcí pro práci s digitálními kamerami, kterou je možné používat v koncových aplikacích uživatele, psaných v běžných textových programovacích jazycích. Knihovna byla vyvíjena ve vývojovém prostředí National Instruments LabVIEW. Do knihovny byly implementovány funkce pro automatické zaostřování objektivu kamery, sledování objektu ve videosignálu, digitální stabilizaci obrazu, základní úpravy obrazu, záznam videosignálu a práci s termografickými kamerami. V současné době je vyvinutá knihovna nabízena uživatelům jako komerční produkt s volně přístupnou zkušební verzí.

Klíčová slova: automatické zaostřování, sledování objektu, stabilizace obrazu, zpracování obrazu, GigE Vision, DLL

Title: Software for control of surveillance cameras and video signal processing

Author: Bc. Martin Zmrhal

Department: Department of Control Engineering

Supervisor: Ing. Pavel Krsek, Ph.D.

Supervisor's e-mail address: krsek@fel.cvut.cz

Abstract: In the presented work a development of a software library for working with digital video cameras is described. The library can be used in end user applications written in usual text based programming languages and it was developed using National Instruments LabVIEW development environment. Functions for automatic focusing of the camera lens, object tracking, digital stabilization, basic image processing, video signal recording and work with thermographic cameras have been implemented. At the present time the developed library is being offered as a commercial product with a free trial version.

Keywords: automatic focusing, object tracking, image stabilization, image processing, GigE Vision, DLL

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Martin Zmrhal**

Studijní program: Kybernetika a robotika
Obor: Systémy a řízení

Název tématu: **Programové vybavení pro řízení průzkumných kamer a zpracování video signálu**

Pokyny pro vypracování:

1. Vyberte vhodný algoritmus pro odhadování ostrosti obrazu pořizovaného kamerou a implementujte ho pro ostření kamery vybavené servomechanizmem na objektivu.
2. Vyberte algoritmus vhodný pro sledování objektu a stabilizaci obrazu v reálném čase. Vybraný algoritmus implementujte.
3. Sestavte knihovnu funkcí, která bude implementovat algoritmus automatického ostření a základní algoritmus pro sledování vybraného objektu ve scéně za účelem stabilizace. Knihovna bude připravena ve vývojovém prostředí National Instruments LabView. Knihovna bude prostřednictvím volání funkcí ovladače kamery přímo řídit ostření objektivu.
4. Navrhněte a proveďte experimenty dokumentující funkčnost vámi navržených a implementovaných algoritmů. Experimenty zdokumentujte.
5. Vytvořené algoritmy a software popište v uživatelské a programátorské dokumentaci.

Seznam odborné literatury:

[1] Gamadia, M.; Kehtarnavaz, N.; , "A Real-time Continuous Automatic Focus Algorithm for Digital Cameras", 2006 IEEE Southwest Symposium on Image Analysis and Interpretation, vol., no., pp.163-167, 0-0 0

[2] Yu Sun; Duthaler, S.; Nelson, B.J.; , "Autofocusing algorithm selection in computer microscopy," 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005 (IROS 2005), vol., no., pp. 70- 76, 2-6 Aug. 2005

[3] Handbook of image and video processing. 2nd ed. Amsterdam: Elsevier, 2005. xv, 1372 s. ISBN 0-12-119792-1.

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

[5] D. Comaniciu, V. Ramesh, P. Meer. Kernel-Based Object tracking. IEEE Transaction in Pattern Analysis Machine Intelligence, Vol. 25, No. 5, 564-575, 2003.

Vedoucí: Ing. Pavel Krsek, Ph.D.

Platnost zadání: do konce letního semestru 2013/2014

L.S.

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 2. 1. 2013

Poděkování

Na tomto místě bych rád poděkoval všem, kteří mi pomáhali při tvorbě této diplomové práce. Především pak vedení společnosti Workswell, s. r. o. za možnost účastnit se zajímavého projektu a poskytnutí vývojových prostředků. Dále bych chtěl poděkovat Ing. Pavlovi Krskovi, Ph.D., za vstřícný přístup při vedení této diplomové práce a podnětné konzultace. Také bych rád poděkoval mé rodině a přítelkyni za jejich podporu během celého studia.

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 2. ledna 2014

Martin Zmrhal

OBSAH

Anotace ii

Zadání práce iii

1. Úvod 1

1.1. Historie projektu 1

1.2. Struktura práce 2

2. Princip implementace knihovny 3

2.1. Vývojové prostředí National Instruments LabVIEW™ 3

2.2. Sdílená dynamická knihovna a komunikace mezi procesy 6

2.3. Zobrazování videa v koncové aplikaci 8

3. Ovládání digitální kamery 11

3.1. Otevření komunikace s kamerou 11

3.2. Ovládání hardwarových parametrů kamery 12

4. Automatické zaostřování kamery 15

4.1. Problematika ostrosti obrazu 16

4.2. Měření míry zaostření snímku 17

4.2.1. Gradient snímku 17

4.2.2. Funkce pro vyhodnocení ostrosti snímku 18

4.3. Testování funkcí pro měření ostrosti snímku 20

4.4. Výsledná volba funkce pro měření ostrosti snímku 23

4.5. Filtrace průběhu funkce ostrosti snímku 23

4.6. Detekce růstu a klesání funkce ostrosti 29

4.7. Algoritmus automatického zaostření kamery 30

4.8. Popis hardware pro zaostřování kamery 33

4.9. Ovládání automatického zaostřování z DLL knihovny 34
4.10. Praktické testy automatického zaostřování 35
4.11. Omezení použitého hardware 37

5. Sledování objektu v obraze a digitální stabilizace obrazu 39
5.1. Požadavky na funkce sledování objektu v obraze a digitální stabilizace obrazu . 39
5.2. Problematika hledání vzoru v obraze 39
5.3. Vyhledávání šablony v obraze pomocí NI LabVIEW 41
5.4. Implementace vyhledávaní objektu v obraze a digitální stabilizace v knihovně . 43
5.5. Praktické výsledky algoritmu vyhledávaní objektu v obrazu a digitální stabilizace 45
5.6. Ovládání sledování objektu a digitální stabilizace z DLL knihovny 47

6. Funkce pro zpracování a úpravy snímků 49
6.1. Funkce pro úpravy jasu a kontrastu snímku 49
6.2. Funkce pro úpravu barevnosti snímku 52
6.3. Funkce pro softwarové doostření snímku 53

7. Ukládání snímků a záznam videosignálu 57
7.1. Ukládání snímků videosignálu 57
7.2. Záznam videosignálu 58

8. Podpora termografických kamer 61
8.1. Konverze teplotních dat z termografické kamery 61
8.2. Funkce pro měření teplot v termografickém snímku 62

9. Licencování a distribuce knihovny 65
9.1. Implementace licenčního systému 65
9.1.1. Zkušební verze knihovny 65
9.1.2. Plná verze knihovny 66
9.2. Distribuce knihovny a testovací koncová aplikace 67

10. Závěr 69

Literatura 71

Přílohy 75

A. Fotografie sestavy kamery a mechanismu objektivu 77

B. Funkce pro vyhodnocení ostrosti 83

C. Grafy filtrace funkce pro vyhodnocení ostrosti **93**

D. Grafy průběhů funkce ostrosti při automatickém ostření **97**

E. Ukázky zdrojového kódu **101**

F. Seznam příkazů pro ovládání krokového motoru **105**

G. Blokové schéma knihovny **109**

H. Dokumentace knihovny **111**

H.1. Obecný popis DLL funkcí a záznam chyb v logu 111
H.2. Funkce pro připojení ke kameře, ovládání okna knihovny a nastavení videosignálu 111
H.2.1. Spuštění session a ukončení session 111
H.2.2. Otevření a zavření okna s videem 113
H.2.3. Funkce pro ovládání a čtení parametrů digitální kamery 113
H.2.4. Uložení parametrů a automatické načtení v novém Session 116
H.2.5. Zjištění snímkové frekvence a rozlišení obrazu 116
H.2.6. Přístup k aktuálním snímkům videosignálu 117
H.3. Funkce pro zpracování a úpravy snímků 120
H.4. Funkce pro ovládání automatického zaostřování kamery 122
H.5. Funkce pro sledování objektu v obraze 127
H.6. Funkce pro ovládání digitální stabilizace obrazu 130
H.7. Funkce pro práci s termografickými kamerami 131
H.7.1. Funkce pro nastavení měření, termálního obrazu a zadávání ROI 131
H.7.2. Čtení měřených teplotních dat a statistik 134
H.8. Funkce pro záznam videosignálu 137

SEZNAM OBRÁZKŮ

2.1. Ukázka jednoduchého programu v NI LabVIEW. 4
2.2. Příklad jednoduché funkce se dvěma vstupy a jedním výstupem. 5
2.3. Jednoduchý program pro připojení ke kameře a zobrazení videosignálu vytvořený v NI LabVIEW. 9
2.4. Snímek koncové aplikace, v níž je vložen Front Panel VI zobrazující videosignál z kamery. 10

3.1. Okno pomocné aplikace Camera Explorer. 12
3.2. Textový soubor s výčtem parametrů kamery vygenerovaný aplikací Camera Explorer. 13
3.3. Zdrojový kód funkce `SetAttribute`. 14

4.1. Blokové schéma algoritmu pro zaostření kamery. 15
4.2. Zjednodušená optická cesta kamery. *[Zdroj: wikipedia.org, 2013]* 16
4.3. Ukázka zdrojového kódu s implementací funkce F_{var}. 20
4.4. Ukázka zdrojového kódu s implementací funkce $F_{tenengrad}$. 21
4.5. Výběr scén, na kterých byly testovány funkce pro výpočet ostrosti. 21
4.6. Průběhy funkcí pro vyhodnocení ostrosti pro první a druhý výřez z obrázku 4.5. 22
4.7. Průběhy funkcí pro vyhodnocení ostrosti pro třetí a čtvrtý výřez z obrázku 4.5. . 23
4.8. Ukázka výřezu scény se šumem v obraze. 24
4.9. Ukázka zašuměného průběhu funkce $F_{tenengrad}$. 24
4.10. Průběh funkce $F_{tenengrad}$ použitý pro analýzu frekvenčního spektra. 25
4.11. Frekvenční spektra šumu statické scény a funkce $F_{tenengrad}$ při přejíždění pozice nejlepšího zaostření 26
4.12. Průběhy funkce ostrosti $F_{tenengrad}$ filtrované pěti různými filtry. Vlevo první měření, vpravo druhé měření. 27
4.13. Průběhy funkce ostrosti $F_{tenengrad}$ filtrované pěti různými filtry. Vlevo první měření, vpravo druhé měření. 27

4.14. Průběhy funkce ostrosti $F_{tenengrad}$ filtrované pěti různými filtry. Vlevo první měření, vpravo druhé měření. 28
4.15. Porovnání frekvenčních charakteristik digitálního Butterworthova a Chebyshevova filtru typu dolní propust. 28
4.16. Průběh funkce $F_{tenengrad}$ a její diference při šumu v obraze. 30
4.17. Průběh funkce $F_{tenengrad}$ a její diference při přejetí vrcholu ostrosti. 31
4.18. Vývojový diagram algoritmu pro automatické zaostření kamery. 31
4.19. Schématický průběh hledání polohy objektivu s maximální ostrostí obrazu. . . 32
4.20. Průběh filtrované funkce ostrosti $F_{tenengrad}$ a její diference při testech automatického zaostřování. 36
4.21. Průběh filtrované funkce ostrosti $F_{tenengrad}$ zaostřování bez kompenzace vůle převodovky. 38

5.1. Znázornění polohy a souřadnic šablony vyhledávané ve snímku. 41
5.2. Blokové schéma algoritmu vyhledávaní objektu v obraze. 44
5.3. Vyhledávání objektu v různých snímcích sekvence videosignálu. 45
5.4. Ukázka digitální stabilizace scény v koncové aplikaci. Kladná rotace původního snímku. 46
5.5. Ukázka digitální stabilizace scény v koncové aplikaci. Záporná rotace původního snímku. 46

6.1. Transformační křivka funkce pro úpravu kontrastu za použití různých metod. . 50
6.2. Srovnání metod funkce pro úpravu kontrastu snímku. 51
6.3. Srovnání metod funkce pro úpravu barevnosti snímku. 53
6.4. Schéma obecného algoritmu doostřování jednorozměrném signálu. 53
6.5. Demonstrace funkce doostřování snímku. 54
6.6. Ukázka zdrojového kódu s implementací funkce softwarového doostřování snímku. 55

8.1. Barevné teplotní palety dostupné v knihovně. 63
8.2. Koncová aplikace s měřícími oblastmi vykreslenými do videosignálu z termografické kamery. 64

9.1. Ukázky obrazovek aplikací SmartVision Activator a SmartVision Deactivator. . 66

A.1. Sestava kamery a zaostřovacího mechanismu objektivu. 78
A.2. Sestava kamery a zaostřovacího mechanismu objektivu. 79
A.3. Sestava kamery a mechanismu objektivu. Detail krokového motoru a koncových spínačů. 80
A.4. Sestava kamery a mechanismu objektivu vedle vývojové desky Arduino. 81

B.1. První dvě testovací scény s červeně vyznačenými výřezy, ve kterých byly vyhodnocovány funkce ostrosti 84
B.2. Třetí testovací scéna s červeně vyznačenými výřezy, ve kterých byly vyhodnocovány funkce ostrosti 85
B.3. Výřezy z testovacích scén, na kterých byly testovány funkce pro měření ostrosti snímku. 86
B.4. Grafy průběhů funkcí pro vyhodnocení ostrosti výřezů 1 a 2 87
B.5. Grafy průběhů funkcí pro vyhodnocení ostrosti výřezů 3 a 4 88
B.6. Grafy průběhů funkcí pro vyhodnocení ostrosti výřezů 5 a 6 89
B.7. Grafy průběhů funkcí pro vyhodnocení ostrosti výřezů 7 a 8 90
B.8. Graf průběhů funkcí pro vyhodnocení ostrosti výřezů 9 91

C.1. Měření 1 - průběhy filtrací šumu při konstantní pozici objektivu. 94
C.2. Měření 1 - průběhy filtrací při skokových posunech objektiv. 94
C.3. Měření 1 - průběhy filtrací při přejezdu přes polohu s maximální ostrostí obrazu. 95
C.4. Měření 2 - průběhy filtrací šumu při konstantní pozici objektivu. 95
C.5. Měření 2 - průběhy filtrací při skokových posunech objektiv. 96
C.6. Měření 2 - průběhy filtrací při přejezdu přes polohu s maximální ostrostí obrazu. 96

D.1. Průběh funkce ostrosti při startu objektivu směrem k pozici s menší ostrostí. . . 98
D.2. Průběh funkce ostrosti při startu objektivu z pozice s nejvyšší ostrostí. 98
D.3. Průběh funkce ostrosti při startu objektivu směrem k pozici s nejvyšší ostrostí. . 99
D.4. Průběh funkce ostrosti bez kompenzace vůle převodovky motoru. 99

E.1. Ukázka LabVIEW zdrojového kódu hlavního VI funkce pro automatické zaostřování kamery. 102
E.2. Ukázka LabVIEW zdrojového kódu hlavního VI funkce pro sledování objektu v obraze. 103
E.3. Ukázka LabVIEW zdrojového kódu hlavního VI funkce pro záznam videosignálu. 104

G.1. Blokové schéma průchodu snímku procesem knihovny. 110

KAPITOLA

PRVNÍ

ÚVOD

Tato diplomová práce pojednává o tvorbě softwarové knihovny pro řízení digitálních videokamer a následném zpracování videosignálu z těchto kamer. Pro konkrétní typ kamery byl rovněž implementován algoritmus pro automatické zaostření objektivu vybaveného krokovým motorem. Motivací vytvoření této knihovny byl projekt realizovaný ve společnosti Workswell, s. r. o.

Vyvinutá softwarová knihovna umožňuje pracovat s barevnými, monochromatickými anebo s termografickými kamerami s různými typy digitálních rozhraní. U všech typů kamer je možné nastavovat libovolné hardwarové atributy kamery. Videosignál z barevných a monochromatických kamer lze upravovat funkcemi pro softwarovou filtraci obrazu, je možné aplikovat digitální stabilizaci obrazu, nahrávat záznam apod. V obraze z termografických kamer je navíc možné provádět měření teplot a získávat různé statistiky z měřených oblastí.

Knihovna byla nejprve určena jako řešení na míru pro jediného zákazníka[1], nicméně později byla rozšířena o další funkce pro zpracování videosignálu a v současné době je distribuována jako produkt výše zmíněné společnosti.

V dalším textu bude rozebrán postup při řešení dílčích problémů při vývoji knihovny, zejména pak automatického zaostřování kamery, digitální stabilizace scény, algoritmů pro filtraci snímků videosignálu a dalších funkcí knihovny. Knihovna byla implementována ve vývojovém prostředí National Instruments LabVIEW.

1.1. Historie projektu

Knihovna vznikala původně podle zadání zákazníka firmy Workswell, s. r. o. Jednalo se o firmu konstruující plošiny s digitálními a analogovými kamerami na terénní automobily. K těmto kamerám si zákazník sám vyvíjel i optické členy, například objektivy, mechanismy ostření apod.

[1] Název společnosti zákazníka je po vzájemné dohodě záměrně utajen.

Zobrazování videosignálu z kamery bylo realizováno na dotykové obrazovce počítače uvnitř vozidla, přičemž zákazník sám vytvářel i koncový software pro tento počítač.

Softwarová knihovna vznikla za účelem zobrazování videosignálu z kamer s poměrně novým rozhraním GigE Vision v aplikaci zákazníka. Bylo tedy nutné vymyslet způsob jak zobrazit videosignál v nadřazené aplikaci psané v některém z běžných programovacích jazyků (např. C++, C#). Dále bylo požadováno implementovat automatické zaostřování kamery na základě obrazové informace z videosignálu. Knihovna měla rovněž umožňovat používat další funkce pro zpracování obrazu, zejména úpravy jasové složky, filtrace barev apod.

Kamery byly montovány na plošině vysunuté nad vozidlem, což přinášelo komplikace pokud byl nastartován motor. Přes montáž plošiny se totiž na kameru přenášely vibrace a obraz se tedy při pozorování rychle třásl. Bylo tedy žádoucí implementovat vhodný algoritmus pro stabilizaci takto se třesoucího obrazu, tzv. digitální stabilizaci.

Po dodání hotové softwarové knihovny zákazníkovi bylo rozhodnuto pokračovat ve vývoji a doplnit knihovnu o další funkce, zejména pro práci s termografickými kamerami. Měly být implementovány i funkce pro záznam videosignálu do souborů formátu AVI. V neposlední řadě bylo nutné navrhnout a implementovat způsob licencovaní software.

Z knihovny nakonec vznikl komerční produkt pojmenovaný SmartVision Library, distribuovaný společností Workswell, s. r. o. Ve zkušební, časově omezené verzi je knihovna volně ke stažení na webových stránkách společnosti[2].

1.2. Struktura práce

V kapitole 2 je stručně uvedeno vývojové prostředí National Instruments LabVIEW a nastíněna základní problematika tvorby softwarové knihovny a komunikace koncové aplikace uživatele s knihovnou.

V kapitolách 4 a 5 jsou rozebrány implementované algoritmy pro automatické zaostřování kamery, sledování objektu v obraze a digitální stabilizaci videosignálu. Tyto kapitoly obsahují nejprve studium dané problematiky, je zdůvodněna volba vybraného řešení, popsán způsob implementace řešení a na závěr jsou demonstrovány praktické výsledky vyvinutých algoritmů.

Kapitoly 6 až 8 popisují další funkce pro úpravy a transformace snímků, přístup k těmto snímkům z uživatelské aplikace, nahrávání záznamů videosignálu a práci s termografickými kamerami. V kapitole 9 je vysvětlen způsob licencování a distribuce knihovny.

[2]Zkušební verze knihovny je dostupná volně ke stažení z webových stránek společnosti: `http://www.workswell.cz/smartvision-sdk-library/`.

KAPITOLA

DRUHÁ

PRINCIP IMPLEMENTACE KNIHOVNY

Vzhledem k tomu, že bylo požadováno používat funkce pro ovládání kamer a zpracování obrazu v koncové aplikaci psané v programovacím jazyku C++, resp. C#, pro operační systémy Microsoft Windows, bylo rozhodnuto software distribuovat jako sdílenou dynamickou knihovnu DLL (Dynamic Linked Library). Výhodou tohoto řešení je jak snadná tvorba knihovny v National Instruments LabVIEW, tak snadná implementace funkcí z knihovny v koncové aplikaci uživatele.

2.1. Vývojové prostředí National Instruments LabVIEW™

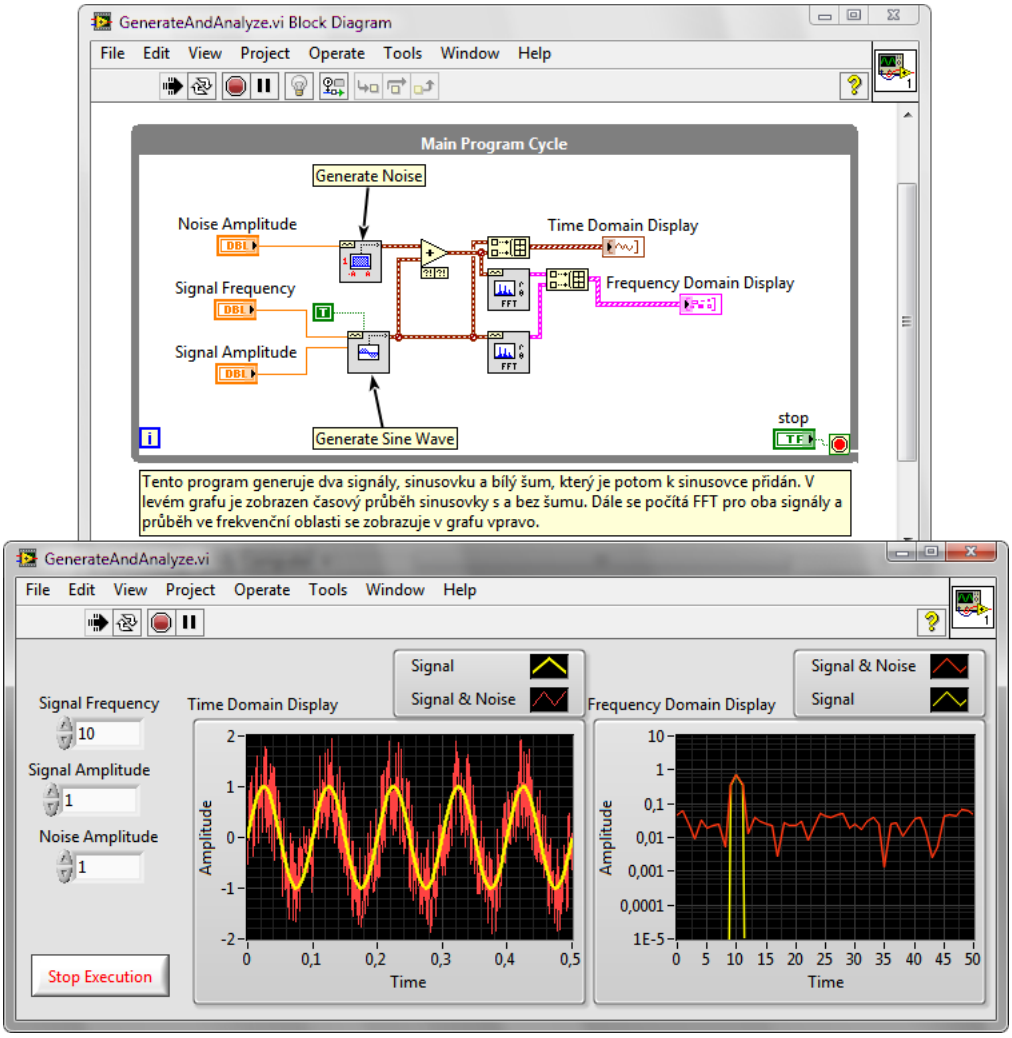
Při sestavování zadání knihovny byla požadována práce především s kamerami s GigE Vision® rozhraním. Výhodou tohoto rozhraní je možnost přenášet vysokorychlostní videosignál o vysokém rozlišení přes standardní gigabit ethernetové (GigE) prvky. To v praxi znamená možnost použití levných a běžně dostupných Cat 5 UTP datových síťových kabelů, standardních přepínačů, síťových karet a dalších síťových prvků nejen pro komunikaci ale i pro PoE napájení kamer[1]. Stručné porovnání rozhraní GigE Vision s ostatními, dnes již také standardními rozhraními pro průmyslové digitální kamery lze najít v [1].

Protože společnost Workswell, s. r. o. vlastnila licenci na vývojové prostředí National Instruments LabVIEW, které práci s GigE Vision kamerami bohatě podporuje a z velké části usnadňuje, bylo rozhodnuto sestavit softwarovou knihovnu právě v tomto prostředí.

National Instruments LabVIEW (NI LabVIEW) je vývojové prostředí využívající grafické programování pro tvorbu programů zejména pro sběr dat, ovládání přístrojů, automatizaci, ale i pro psaní koncových aplikací. Programový kód tedy není zadáván textově jako v klasických programovacích jazycích, ale je vytvářen kladením a spojováním funkčních bloků na tzv. blokovém

[1] Power over Ethernet - napájení zařízení po datovém síťovém kabelu.

schématu (Block Diagram). Každý funkční blok reprezentuje určitou funkci nebo řídící strukturu (cykly, smyčky, podmíněné příkazy atd.). Bloky jsou spojovány spojnicemi, který vedou signál (hodnotu) na vstupy nebo výstupy jiných bloků, čímž je definováno pořadí vykonávání programu (tzv. Data Flow). Ukázka jednoduchého programu pro generování periodického signálu se šumem a pro výpočet jejich spektra je na obrázku 2.1.

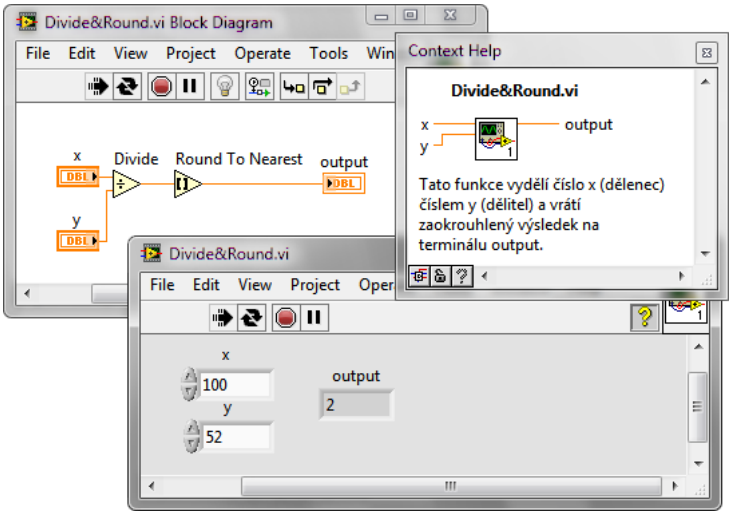


Obrázek 2.1.: Ukázka jednoduchého programu v NI LabVIEW.

Důležitým principem v NI LabVIEW je fakt, že s tvorbou blokového schématu se na tzv. čelní panel (Front Panel) automaticky kladou odpovídající GUI prvky. S každým vstupním a výstupním prvkem (tzv. terminálem) umístěném na blokovém diagramu je automaticky vytvořena odpovídající grafická komponenta pro zadávání vstupní hodnoty, případně pro vizualizaci výstupní hodnoty. Tento přístup velice zrychluje tvorbu grafického rozhraní aplikace, které je tak

tvořeno zcela automaticky. Při běhu zkompilované aplikace je koncovému uživateli zpřístupněn samozřejmě pouze Front Panel.

Blokové schéma a čelní panel dohromady tvoří vlastní program (nebo funkci), který se nazývá VI (Virtual Instrument). Vstupní a výstupní grafické terminály na čelním panelu lze pak snadno nastavit jako vstupy či výstupy daného programu či funkce. VI je potom možné používat jako samostatný funkční blok v blokovém schématu. Příklad funkce, která vydělí a zaokrouhlí dvě čísla je na obrázku 2.2. V okně „Context Help“ je zobrazen náhled funkčního bloku se vstupními a výstupními terminály.



Obrázek 2.2.: Příklad jednoduché funkce se dvěma vstupy a jedním výstupem.

Grafický přístup k programování v NI LabVIEW společně s bohatou paletou funkcí umožňuje velmi rychlý vývoj prototypových zkušebních aplikací a algoritmů i vývojářům bez předchozí větší znalosti algoritmizace či textového programování. Na druhou stranu vývojáři s pokročilými zkušenostmi s vývojem v NI LabVIEW jsou schopni sestavit i aplikace velkého rozsahu, ne zřídka řídící velké průmyslové procesy.

Velkou výhodou NI LabVIEW je vysoce optimalizovaný kompilátor, který umožňuje vykonávat i náročné algoritmy srovnatelně či rychleji než klasické programovací jazyky. Kompilátor se například sám automaticky stará o rozložení paralelních větví programu na více jader procesoru, takže implementace programů využívajících plný výkon moderních procesorů je samozřejmostí.

Pro přístup ke GigE Vision kamerám slouží balík ovladačů NI Vision Acquisition Software. Vytvořená knihovna používá konkrétně NI-IMAQdx ovladač, který umožňuje z NI LabVIEW komunikovat s kamerami s rozhraním GigE Vision, IEEE 1394, IP (Ethernet) a DirectShow (USB kamery). Výhodou tohoto ovladače je jednotné rozhraní pro přístup ke kamerám v NI LabVIEW, je tedy možné používat stále tytéž funkce nezávisle na rozhraní kamery. Ve výsledku to znamená, že vyvíjená knihovna umožňuje pracovat nejen s kamerami s rozhraním GigE Vision,

ale i s libovolnou kamerou s jedním z výše uvedených rozhraní. Je možné dokonce používat i kamery s analogovým výstupem, které jsou k počítači připojeny přes tzv. framegrabber, tedy zařízení digitalizující analogový video signál (framegrabbery typicky mají USB nebo GigE Vision rozhraní).

Dalším důležitým použitým prvkem je NI Vision Development Module, což je samostatný balík funkcí pro zpracování obrazu. Obsahuje širokou paletu funkcí pro komunikaci s kamerami, základní úpravy obrazu a maticové operace i pokročilé algoritmy jako jsou např. rozpoznávání textu ve snímku či detekce objektů ve snímku.

Aby bylo možné vytvořit a zkompilovat vlastní knihovnu, byl použit modul NI LabVIEW Application Builder, který umožňuje z programů vytvořených ve vývojovém prostředí NI LabVIEW sestavit spustitelné .exe aplikace, sdílené knihovny nebo například instalátory pro snadnou distribuci vytvořených aplikací.

2.2. Sdílená dynamická knihovna a komunikace mezi procesy

Každé VI vytvořené v NI LabVIEW je možné zkompilovat do DLL knihovny jako funkci. Vstupní a výstupní terminály definované ve funkčním bloku VI potom slouží jako vstupní a výstupní parametry funkce volané z DLL. Všechny deklarace knihovních funkcí uvedené v dalším textu jsou zapsány ve formátu, v jakém jsou funkce uváděny v NI LabVIEW Application Builder při sestavování DLL knihovny. V různých programovacích jazycích mohou mít zejména názvy datových typů jiný formát.

Funkci zobrazenou na obrázku 2.2 na předchozí straně by bylo možné zkompilovat do knihovní funkce volané v textovém jazyku např. takto:

```
double DivideRound(double x, double y).
```

DLL knihovna vytvořená v NI LabVIEW je potom vlastně soubor takovýchto funkcí. Více vstupních parametrů se v definici funkce odděluje čárkou, nicméně v případě více výstupních parametrů (více výstupů daného VI) je nutné zvolit jiný přístup pomocí ukazatelů na hodnotu do operační paměti (pointerů). Funkce, která zároveň vydělí a vynásobí dvě čísla a vrátí výsledky těchto operací by potom vypadala např. takto:

```
void DivideMultiply(double *divisionValue, double
    *multiplicationValue),
```

kde `double *divisionValue` a `double *multiplicationValue` jsou ukazatele na proměnné, do kterých se zapíší výsledky operací dělení a násobení.

Ve vytvořené knihovně jsou předávány následující datové typy:

- **`int8_t`**, **`int16_t`**, **`int32_t`** – datový typ integer se znaménkem, používán pro celá čísla, číslo ze prefixem „**`int`**“ udává počet bitů, kterými je integer reprezentován,
- **`uint8_t`**, **`uint16_t`**, **`uint32_t`** – datový typ integer bez znaménka, používán pro celá čísla, číslo ze prefixem „`uint`“ udává počet bitů, kterými je integer reprezentován,
- **`double`** – datový typ pro reálná čísla s plovoucí řádovou čárkou, reprezentován 64 bity,
- **`char`** – datový typ reprezentující jeden ASCII znak,
- složené datové typy vytvořené z výše uvedených formou pole, tedy např. **`int32_t`** `Array[]` reprezentující pole datových typů **`int32_t`** nebo **`char`** `String[]` reprezentující textový řetězec,
- protože datový typ pro logickou hodnotu (**`boolean`**) není standardizován, bylo by nutné pro něj v programovacím jazyku koncové aplikace importovat pomocnou knihovnu vytvořenou při kompilaci DLL. Proto je logická hodnota předávána jako datový typ **`int8_t`**. Logickým hodnotám `false` a `true` tedy odpovídají číselné hodnoty 0 a 1.

Výše uvedené volání funkcí z DLL zastupuje komunikaci iniciovanou koncovou aplikací – kód aplikace zavolá funkci a vrátí se mu návratové hodnoty. Pro integraci některých algoritmů knihovny bylo nutné vyřešit ještě odesílání informací o událostech, které nastaly přímo v procesu knihovny. Jedná se například o událost kliknutí kurzorem myši do okna s videosignálem z kamery nebo informaci o změně souřadnic objektu vyhledávaného v obraze.

Jelikož se ve výše uvedených případech jedná o posílání pouze malého množství dat, byl tento způsob komunikace realizován pomocí Windows zpráv (Windows Messages)[2]. Knihovna k tomu využívá funkce `PostMessage(int hWnd, int Msg, int wParam, int lParam)` a `SendMessage(int hWnd, int Msg, int wParam, int lParam)` z Windows API knihovny `User32.dll`[3].

Parametry těchto funkcí jsou reference na hlavní okno nadřazené aplikace (angl. handle), kterému bude zpráva odeslána (`hWnd`), číselný identifikátor zprávy (`Msg`) a doplňkové dva parametry (`wParam` a `lParam`). Identifikátor zprávy umožňuje zadefinovat si vlastní (nesystémové) zprávy. V případě knihovny jsou používány zprávy s identifikátory 9000 až 9002. Doplňkové parametry slouží právě pro přenos informace o události nastalé v knihovně. Tedy např. v případě kliknutí do okna s obrazem z kamery obsahuje první a druhý parametr horizontální a vertikální souřadnici kliknutí. Koncová aplikace potom musí ve zvláštní smyčce kontrolovat, jestli nepřišla některá z těchto zpráv a popřípadě příslušně reagovat.

[2] Standardní způsob, jakým s hlavním procesem aplikace komunikuje operační systém Microsoft Windows nebo ostatní běžící aplikace. Zprávy se v aplikaci řadí do fronty a postupně se zpracovávají. Více viz např. [3].

[3] Windows API je aplikační programovací rozhraní (Application Programming Interface) operačních systémů Windows. Slouží ke komunikaci aplikací s operačním systémem Windows a volání různých jeho funkcí.

2.3. Zobrazování videa v koncové aplikaci

V zadání knihovny bylo požadováno zobrazovat v koncové aplikaci videosignál z kamery. Bylo tedy nutné navrhnout řešení, jak tato objemná data předávat mezi knihovnou a koncovou aplikací.

Předávání obrazových dat pomocí DLL funkcí by bylo z principu neefektivní a nebylo by možné dosáhnout velké snímkové frekvence. Další varianta předávání videa pomocí ukazatelů do operační paměti byla zavržena jako implementačně poměrně složitá a navíc na začátku projektu nebylo jisté, co by se s pamětí dělo, pokud by do jednoho místa paměti jeden proces (knihovna) zapisoval a druhý (koncová aplikace) četl.

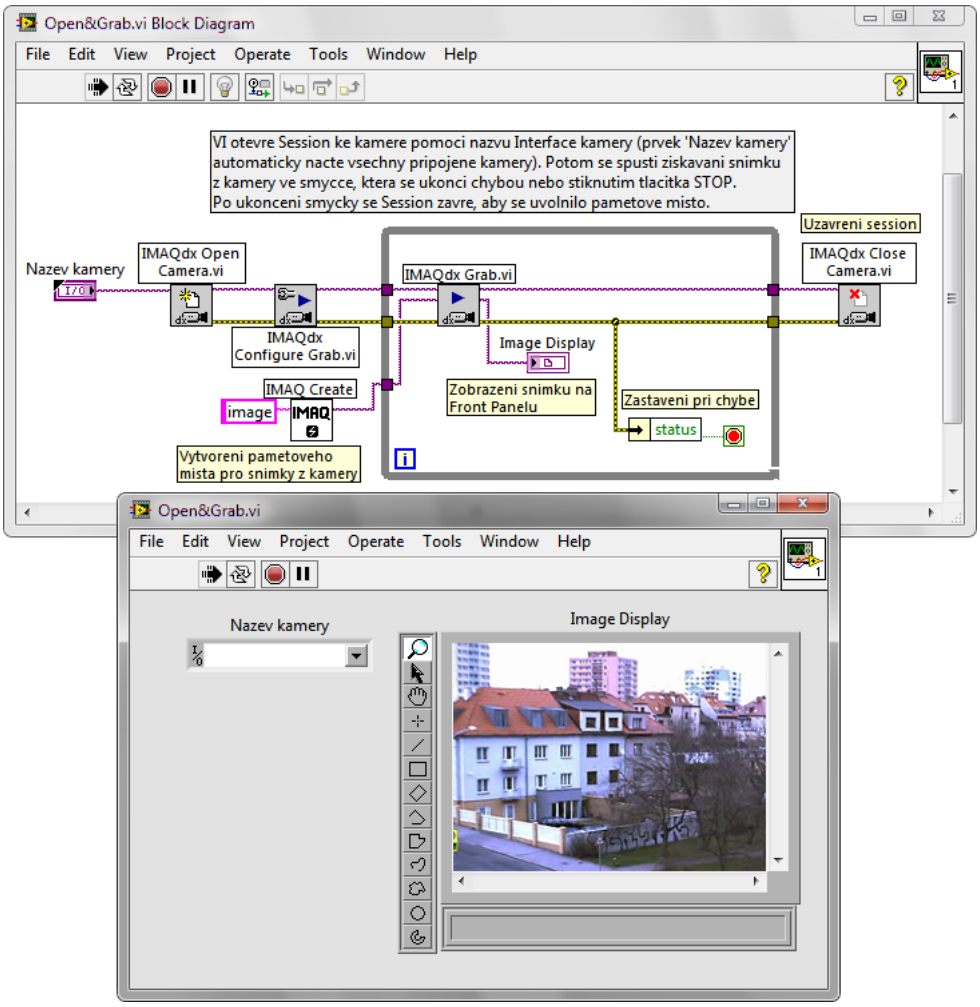
Zobrazování videosignálu z kamery na Front Panelu programu vytvořeného v NI LabVIEW (VI) je naopak poměrně jednoduchý úkol (příklad jednoduchého programu je na obrázku 2.3). Bylo tedy nutné přijít na způsob, jak vložit Front Panel běžícího VI do koncové aplikace zákazníka. Tento neobvyklý úkol byl vyřešen použitím Windows API funkcí [2].

Po zavolání první inicializační funkce z DLL knihovny se spustí VI s hlavním procesem pro získávání videosignálu z kamery. Zároveň se otevře Front Panel, na kterém se tento videosignál zobrazuje.

Dále se pomocí Windows API funkcí zjistí reference (handle) na okno s Front Panelem. Potom se určí souřadnice okna koncové aplikace a podle těchto souřadnic se okno s Front Panelem vloží na pozici v okně koncové aplikace. Nakonec se koncová aplikace nastaví jako tzv. Parent Window a okno s Front Panelem je potom tzv. Child Window. Níže je uvedena ukázka kódu v programovacím jazyku C# .NET, který realizuje celou tuto proceduru.

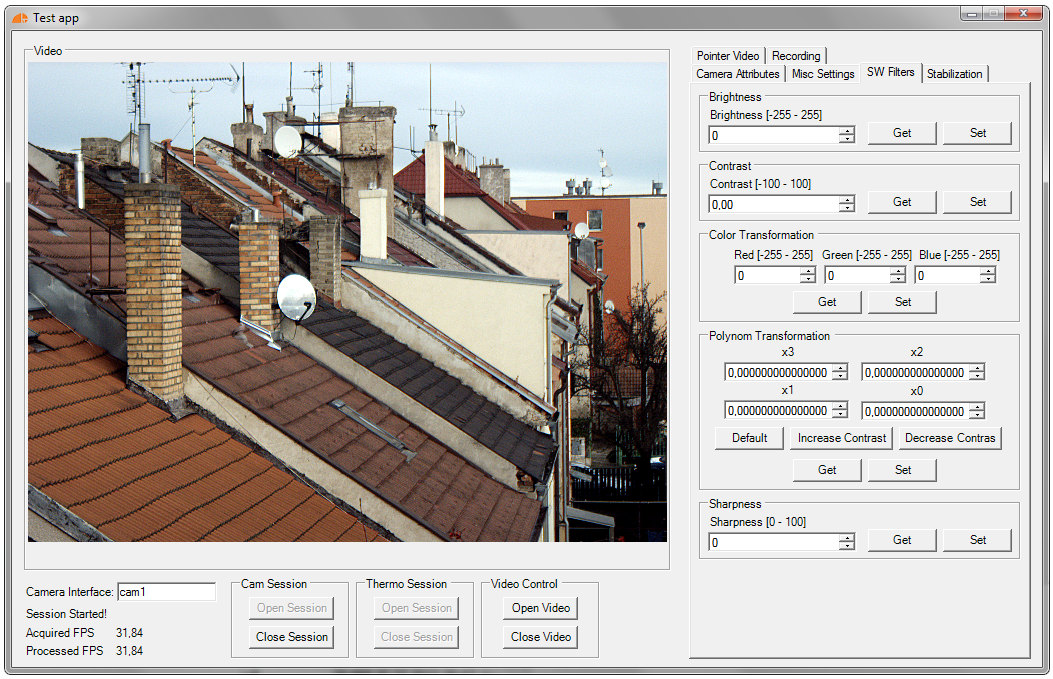
```
// vytvorime  na okno s~Front Panelem knihovny (okno
   ma nazev "SmartVisionVideo")
hWndVI = FindWindow(null, "SmartVisionVideo");
// ziskame obdelnik ohranicujici koncovou aplikaci
   (pomoci WinAPI funkce)
GetClientRect(hWndNET, ref rectNET);
// umistime okno Front Panelu knihovny do okna
   koncove aplikace
SetWindowPos(hWndVI, 0, rectNET.left + 15,
   rectNET.top + 32, 0, 0, 1);
// nastavime rodicovske (parent) okno
SetParent(hWndVI, hWndNET);
```

Výsledkem výše uvedeného procesu je běžící program naprogramovaný v NI LabVIEW, jehož grafické rozhraní je součástí okna libovolné koncové aplikace. Chování koncové aplikace je naprosto standardní, je možné ji minimalizovat, posouvat či měnit velikost jejího okna. Uživatel tedy v aplikaci vidí živý videosignál z kamery tak, jak by tento signál byl zobrazen přímo na



Obrázek 2.3.: Jednoduchý program pro připojení ke kameře a zobrazení videosignálu vytvořený v NI LabVIEW.

Front Panelu VI. Ukázka koncové aplikace napsané v programovacím jazyku C#, v níž je vložen Front Panel zobrazující videosignál z kamery je na obrázku 2.4 na následující straně



Obrázek 2.4.: Snímek koncové aplikace, v níž je vložen Front Panel VI zobrazující videosignál z kamery.

KAPITOLA

TŘETÍ

OVLÁDÁNÍ DIGITÁLNÍ KAMERY

Pro ovládání digitální kamery pomocí NI Vision Acquisition Software je nutné se ke kameře nejprve připojit a vytvořit tzv. Session, čímž se spustí hlavní proces knihovny. Po vytvoření Session se v nekonečné smyčce získávají snímky z kamery a je možné ovládat hardwarové parametry kamery. Videosignál z kamery je poté zobrazován na Front Panelu knihovny v koncové aplikaci uživatele.

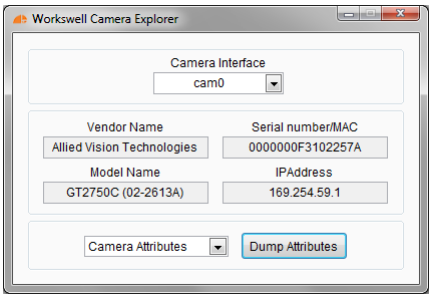
3.1. Otevření komunikace s kamerou

Hlavní proces DLL knihovny se spustí ihned po zavolání knihovní funkce

```
void OpenSessionCam(char cameraInterface[]).
```

Tato funkce otevře komunikaci s kamerou, kterou identifikuje textový parametr `char cameraInterface[]` a následně spustí na Front Panelu obraz z kamery. Identifikátor kamery je automaticky přidělen ovladačem NI-IMAQdx a jednoznačně identifikuje každou kameru připojenou k počítači.

Pro získání seznamu všech dostupných připojených kamer a jejich identifikátorů byl vytvořen samostatný software s grafickým rozhraním nazvaný Camera Explorer. Tento software je distribuován společně s knihovnou. Hlavní okno aplikace Camera Explorer je na obrázku 3.1.



Obrázek 3.1.: Okno pomocné aplikace Camera Explorer.

3.2. Ovládání hardwarových parametrů kamery

Díky ovladači NI-IMAQdx je možné ke všem kamerám s podporovaným rozhraním přistupovat programově stejně. Způsob nastavování hardwarových parametrů[1] je vhodné ilustrovat nejprve na kamerách s rozhraním GigE Vision.

Standard GigE Vision mimo definice transportní a aplikační vrstvy síťové komunikace zavádí softwarové rozhraní GenICam. Toto rozhraní poskytuje unifikovaný přístup k hardwarovým atributům libovolné GigE Vision kamery. Každou kameru provází XML soubor se standardem definovanou strukturou vytvořený výrobcem kamery. V tomto souboru je výčet všech hardwarových atribut dostupných na kameře. Každý atribut je jednoznačně identifikován jeho názvem, obsahuje standardní datový typ, jednotku a doplňkový informační popis funkce tohoto atributu. GenICam XML soubory proto unifikují programátorský přístup k nastavení kamery. Programátor potom jednoduše nahlédne do XML a pomocí GigE Vision ovladače nastaví požadovanou hodnotu hardwarového atributu. K ovládání kamer tedy není zapotřebí používat žádné pomocné SDK[2] ani ovladače od výrobce kamery.

Vyvinutá DLL knihovna řeší ovládání hardwarových parametrů jednoduše pomocí tří funkcí. Nejprve je ovšem nutné pro danou kameru vygenerovat pomocí aplikace Camera Explorer textový soubor obsahující výčet všech parametrů, kterými kamera disponuje. Jedná se v podstatě o přeformátovaný XML soubor, který je výrobcem dodáván s kamerou. Tento textový soubor obsahuje výpis všech atribut ve formátu:

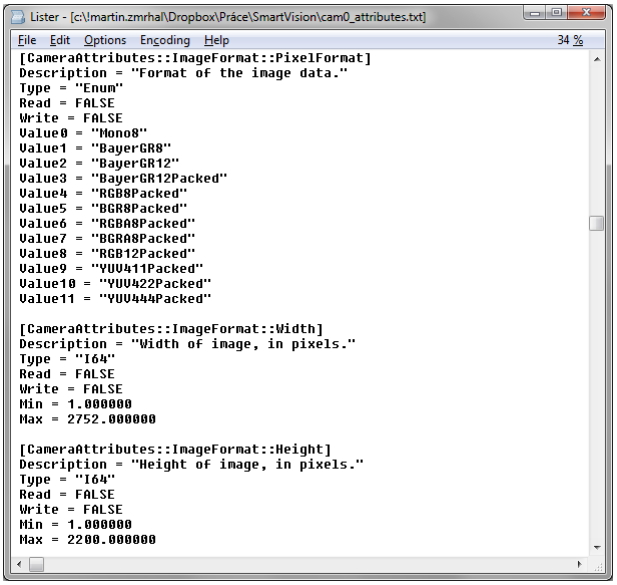
```
[prvniXMLuzel::druhyXMLuzel::n-tyXMLuzel]
Description = "Popis XML atributu ulozeny v~XML
   kamery"
Type = "Datovy typ" // uint32_t, int64_t, double,
   string, enum, boolean
```

[1] Hardwarovými parametry (někdy také atributy) jsou myšleny nastavení různých hardwarových registrů kamery pro ovládání např. výstupního rozlišení kamery, obrazového formátu, doby expozice, clony objektivu, zisku snímače, barevné vyvážení bílé barvy apod.

[2] Software Development Kit, vývojová sada pro určitý programovací jazyk, většinou dodávaná výrobcem, slouží k ovládání jeho kamer.

```
Read = TRUE/FALSE // je mozne atribut cist?)
Write = TRUE/FALSE // je mozne atribut zapisovat?)
Min = "Minimalni hodnota rozsahu atributu"
Max = "Maximalni hodnota rozsahu atributu"
Value0 = "Prvni hodnota datoveho typu enumerator"
Value1 = "Druha hodnota datoveho typu enumerator"
ValueN = "N-ta hodnota datoveho typu enumerator"
```

Ke každému parametru kamery takto nejprve zjistíme a do textového souboru uložíme jeho přesný identifikátor, datový typ, informaci o tom, jestli je parametr možné zapisovat a číst, rozsah platných hodnot parametru a případně, pro datový typ enumerátor, všechny jeho platné hodnoty. Na obrázku 3.2 je jako příklad uvedena část textového souboru s výčtem některých parametrů kamery Allied Vision Technologies Prosilica GT2750.



Obrázek 3.2.: Textový soubor s výčtem parametrů kamery vygenerovaný aplikací Camera Explorer.

K nastavování parametrů kamery potom v knihovně slouží funkce

```
int8_t SetAttribute(char interface[], char
    activeAttribute[], double valueDBL, char
    valueString[]),
```

jejímiž parametry jsou textový identifikátor kamery, název hardwarového atributu (parametru) a jeho číselná, případně textová hodnota. Parametr `valueDBL` nastavuje hodnotu GenICam datových typů **`uint32_t`**, **`int64_t`**, **`double`** a **`boolean`** (hodnoty 0 a 1 odpovídají logickým hodnotám `false` a `true`). Parametr `valueString` nastavuje textovou hodnotu, případně jednu z hodnot enumerátoru. Návratovou hodnotou funkce je číselná hodnota, 1 reprezentuje úspěšné

nastavení atributu a 0 indikuje chybu při jeho nastavení (např. při pokusu o nastavení hodnoty mimo platný rozsah). Jedinou knihovní funkcí je tedy tedy možné ovládat kompletní nastavení kamery s jedním z podporovaných rozhraní.

Pro čtení nastavení daného parametru kamery slouží funkce

```
int8_t GetAttribute(char interface[], char
    activeAttribute[], double *valueDBL, char
    valueString[], int32_t length),
```

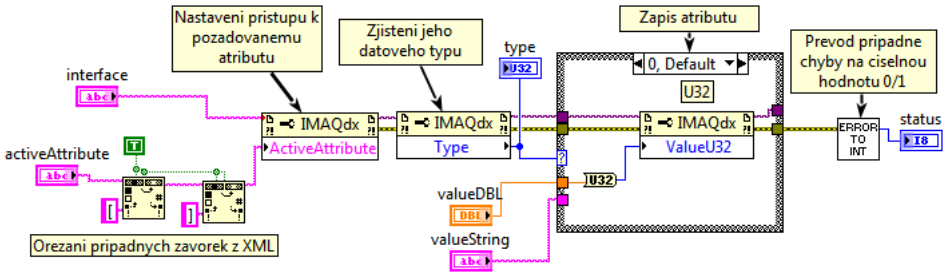
jež se liší jedině v tom, že má více návratových hodnot. První je opět informace o úspěšnosti provedení příkazu (chyba může nastat např. při pokusu o čtení neexistujícího atributu) a další dvě jsou ukazatele na proměnné, do kterých se má zapsat vyčtená číselná, respektive textová hodnota. Pro textovou hodnotu je ještě nutné knihovně zadat její očekávanou délku, aby bylo v operační paměti vyhrazeno dostatek místa pro její uložení.

Poslední z trojice funkcí pro přístup k hardwarovým parametrům je funkce

```
int8_t GetAttributeInfo(char interface[], char
    activeAttribute[], int8_t *type, int8_t
    *readable, int8_t *writable, double *minimum,
    double *maximum, double *valueDBL, char
    valueString[], int32_t length).
```

Princip této funkce je analogický jako u funkce `GetAttribute`, nicméně kromě hodnot požadovaného atributu vyčte i informace o jeho datovém typu, možnosti čtení a zápisu a platný rozsah jeho hodnot.

Pro názornost je zdrojový kód knihovní funkce `SetAttribute` vytvořený v NI LabVIEW zobrazen na obrázku 3.3.



Obrázek 3.3.: Zdrojový kód funkce `SetAttribute`.

Velkou výhodou ovladače NI-IMAQdx je, že stejný přístup jako ke GigE Vision kamerám zavádí i pro další rozhraní. Tedy i pro kamery s dalšími rozhraními lze vygenerovat jistou formu XML souboru definujícího možnosti jejich nastavení. Například pro USB kamery a framegrabbery lze pomocí aplikace Camera Explorer opět vygenerovat textový soubor s výčtem všech dostupných nastavení kamery a poté používat výše uvedené tři funkce pro jejich nastavení.

KAPITOLA

ČTVRTÁ

AUTOMATICKÉ ZAOSTŘOVÁNÍ KAMERY

Jednou z hlavních funkcí knihovny, které byly zákazníkem na začátku projektu požadovány, bylo automatické zaostřování GigE Vision kamery Allied Vision Technologies Prosilica GT2750. K dodané kameře byl namontován objektiv o ohniskové vzdálenosti 100 mm a sestava byla zákazníkem doplněna o krokový motor a elektroniku sloužící k zaostřování objektivu. Pohybem objektivu docházelo ke změně zaostření snímků videosignálu kamery. Celá sestava je zachycena na fotografiích v příloze A. Požadavkem bylo vyvinout funkce pro automatické zaostřování parametrizované pouze na dodanou sestavu.

Jednotlivé dílčí části algoritmu pro automatického zaostření kamery jsou zobrazeny na blokovém schématu na obrázku 4.1.

Obrázek 4.1.: Blokové schéma algoritmu pro zaostření kamery.

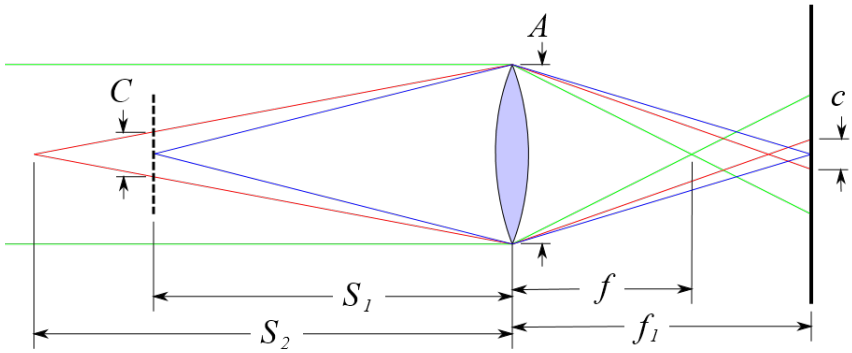
Videosignál z kamery je nejprve redukován na určitou oblast, ve které chceme provést zaostření (výřez). Z výřezu je průběžně v každém příchozím snímku vyhodnocována jeho ostrost (popsáno v oddílech 4.1 až 4.4). Funkci hodnot ostrosti snímků je nutné filtrovat od šumu (viz oddíl 4.5).

Při pohybu objektivu se mění hodnoty funkce ostrosti a je sledován jejich růst a klesání (viz oddíl 4.6) za účelem hledání maxima funkce. Podle vývoje průběhu funkce ostrosti je řízen krokový motor tak, aby bylo dosaženo maximální možné hodnoty funkce a tím i nejlepší polohy zaostření objektivu. Detailně je celý algoritmus popsán v oddíle 4.7.

4.1. Problematika ostrosti obrazu

Lidský mozek rozpozná správně zaostřené snímky podle toho, že obsahují více informace a detailů než snímky rozostřené. Pro rozpoznání množství této informace je nutné nějakým způsobem měřit ostrost snímků[1].

Optická cesta kamery sestává zjednodušeně z objektivu a roviny čipu snímače kamery. Tato cesta reprezentuje přenosovou funkci z bodů vzoru před objektivem na plošky na čipu. Tvar plošek je definován tvarem clony objektivu, v případě ideální kruhové clony se body vzoru zobrazí na kruhy na čipu. Průměr těchto kruhů přímo vyjadřuje míru zaostření s použitím dané optické cesty. Čím bude průměr menší, tím je snímek lépe zaostřen. Situace je zachycena na obrázku 4.2. Správně zaostřený bod je ve vzdálenosti S_1 od objektivu. Rozostřený bod ve vzdálenosti S_2 se zobrazí na kruh s průměrem c. Průměr kruhu dále ovlivňuje ohnisková vzdálenost a clonové číslo objektivu.



Obrázek 4.2.: Zjednodušená optická cesta kamery. *[Zdroj: wikipedia.org, 2013]*

Jinými slovy, rozostřený snímek je výsledkem konvoluce zaostřené scény s rozostřující funkcí optické cesty kamery. Tato přenosová funkce se chová jako filtr typu dolní propust. Frekvence zlomu je maximální, pokud je objektiv správně zaostřen a klesá, pokud se objektiv rozostřuje.

Tím pádem správně zaostřený snímek obsahuje více informace s vysokou prostorovou frekvencí než snímek rozostřený. Pokud bychom nějakým způsobem kvantifikovali množství takové informace, jsme schopni jednoznačně porovnávat ostrost snímků v různých polohách zaostření objektivu. Pro určení množství vysokých prostorových frekvencí se velmi často používají algoritmy založené na výpočtu prostorového gradientu snímků [4, 5].

[1] Měření ostrosti snímků je nutné u metody tzv. pasivního automatického zaostřování. Druhou variantou používanou například u filmových a digitálních zrcadlovek je aktivní metoda, kdy je fotoaparátem vyslán jeden nebo více světelných nebo ultrazvukových svazků. Po odrazu na zaostřovaném předmětu se část svazku vrátí do těla fotoaparátu a pomocí fázového zpoždění odraženého vlnění je určena vzdálenost objektu. Objektiv je potom automaticky zaostřen do polohy odpovídající této vzdálenosti.

4.2. Měření míry zaostření snímku

4.2.1. Gradient snímku

Jednou z častých metod pro vyhodnocení zaostření snímku je výpočet prostorového gradientu snímku. Jeho určení je založeno na výpočtu prvních parciálních derivací sousedních hodnot jasů obrazových bodů, proto má funkce gradientu vlastnosti filtru typu horní propust.

Pokud je $i(x,y)$ dvourozměrný snímek v odstínech šedi (obsahuje pouze jasovou složku) rozměru $M \times N$, gradient na pozici (x,y) tohoto snímku je definován jako dvourozměrný vektor

$$\nabla i(x,y) = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial i}{\partial x} \\ \frac{\partial i}{\partial y} \end{bmatrix}. \tag{4.1}$$

Pro práci s digitálními diskrétními snímky je nutné spojitý gradient aproximovat jeho diskrétní formou. Diskrétní aproximaci je možné provést několika různými filtry, nejjednodušším z nich je prostý rozdíl sousedních hodnot jasu dvou obrazových bodů (pixelů). Horizontální a vertikální složku gradientu lze pak zapsat jako

$$\begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} i(x+1,y) - i(x,y) \\ i(x,y+1) - i(x,y) \end{bmatrix} \tag{4.2}$$

Rovnici (4.2) lze také chápat jako konvoluci snímku s maticemi

$$\triangle_x = \begin{bmatrix} -1 & 1 \\ 0 & 0 \end{bmatrix}, \quad \triangle_y = \begin{bmatrix} -1 & 0 \\ 1 & 0 \end{bmatrix}, \tag{4.3}$$

kde matice $\triangle_x, \triangle_y$ jsou někdy označovány jako tzv. konvoluční jádra. Gradient snímku lze potom vyjádřit pomocí konvoluce jako

$$\nabla i(x,y) = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \triangle_x * i(x,y) \\ \triangle_y * i(x,y) \end{bmatrix}. \tag{4.4}$$

Jeden z problémů metody výpočtu gradientu pomocí první derivace (diference) spočívá v citlivosti této metody na šum v obraze. Tohoto faktu je do jisté míry předcházeno používáním jader s větším rozměrem a složkou ortogonální na směr výpočtu diference. Zapojením okolních pixelů v ortogonálním směru je dosaženo jistého průměrovaní a tím pádem vyhlazení šumu v tomto směru [4, 6, 7].

Obecně lze vlastnosti vyhlazování šumu ovlivnit typem konvolučního jádra reprezentujícího filtr typu dolní propust[2]. Základním takovým jádrem je tzv. Prewittovo jádro

$$\triangle_{p_x} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, \quad \triangle_{p_y} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \tag{4.5}$$

a tzv. Sobelovo jádro

$$\triangle_{s_x} = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, \quad \triangle_{s_y} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}. \tag{4.6}$$

Výsledný gradient snímku má potom tvar

$$\begin{bmatrix} G_{p/s_x} \\ G_{p/s_y} \end{bmatrix} = \begin{bmatrix} \triangle_{p/s_x} * i(x,y) \\ \triangle_{p/s_y} * i(x,y) \end{bmatrix}. \tag{4.7}$$

Sobelovo jádro je často lepší volbou než jádro Prewittovo, protože generovaný filtr má plynulejší frekvenční odezvu [7].

Výše uvedená jádra se používají pro výpočet Prewittova a Sobelova operátoru v algoritmech pro detekci hran v obraze, jedné ze základních obrazových analýz. Hodnota jasu pixelu snímku po zvýraznění hran má potom například se Sobelovým operátorem tvar

$$j(x,y) = \sqrt{G_{s_x}^2 + G_{s_y}^2}. \tag{4.8}$$

4.2.2. Funkce pro vyhodnocení ostrosti snímku

Funkce pro vyhodnocení ostrosti snímku využívající gradient jsou do jisté míry rozšířením operátorů pro detekci hran. Ostrost snímku je však nutné reprezentovat jedinou hodnotou, která je potom zpracovávána při řízení zaostřování objektivu. Průběh této funkce v závislosti na poloze objektivu roste, pokud se blížíme k nejlepšímu zaostření objektivu. V tomto místě dosáhne maxima a při dalším posuvu ostření objektivu opět klesá. Vzdáleně tak připomíná Gaussovu funkci s maximem v poloze správného zaostření objektivu[3].

[2] Suma koeficientů v matici musí být rovna nule, aby byla zachována nulovost při derivaci konstantní funkce (např. osamocený pixel s nenulovou hodnotou jasu a nulovými hodnotami v okolí).

[3] Příklady takových funkcí jsou např. na obrázku 4.6 na straně 22.

Příkladem nejjednodušší funkce pro vyhodnocení ostrosti snímku může být funkce často označována jako F_{grad} (kvadrát gradientu), popsaná rovnicí

$$F_{grad} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [\nabla g(x,y)]^2 = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [G_x^2 + G_y^2]. \tag{4.9}$$

V [8] a [9] bylo popsáno a otestováno celkem osmnáct různých funkcí pro vyhodnocení ostrosti snímků při mikroskopové fotografii. Porovnávány byly nejen funkce založené na výpočtu gradientu snímku, ale i funkce počítající jasové statistiky, absolutní hodnoty jasů, celkový kontrast a korelaci snímků. Jednotlivé funkce byly hodnoceny podle následujících kritérií:

- přesnost – odchylka polohy maximální ostrosti stanovené funkcí od polohy zaostření objektivu ručně nastavené zkušenou obsluhou mikroskopu,
- rozsah – délka intervalu posuvu objektivu, kde funkce ostrosti monotónně klesá,
- šířka – délka intervalu, na kterém průběh funkce dosahuje 50 % maximální hodnoty, popisuje strmost průběhu funkce. Čím je šířka menší (vrchol průběhu je „užší“), tím je snadnější jednoznačně lokalizovat polohu maxima,
- počet falešných maxim – počet maxim funkce vyjma globálního maxima,
- šum – úroveň šumu hodnotící funkce.
- výpočetní čas – čas strávený výpočtem funkce nad jedním snímkem.

Mezi pěti nejlepšími funkcemi pro danou aplikaci se v obou zdrojích objevovaly níže uvedené. Společnou vlastností těchto funkcí je velká odolnost vůči šumu ve snímku a nezávislost na nastavení prahování.

Tenenbaumův gradient (Tenengrad) [10, 11]

Výpočet Tenenbaumova gradientu je založen na konvoluci snímku se Sobelovými jádry $\triangle_{s_x}$ a $\triangle_{s_y}$ (viz (4.6)). Funkce má tvar

$$F_{tenengrad} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [G_{s_x}^2 + G_{s_y}^2]. \tag{4.10}$$

Jedná se tedy o součet druhých mocnin Sobelových gradientů přes všechny pixely snímku.

Rozptyl [10, 12]

Jedná se o funkci založenou na analýze kontrastu snímku. Vyhodnocuje odchylky jasů pixelu od jejich střední hodnoty:

$$F_{var} = \frac{1}{M \cdot N} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [i(x,y) - \mu]^2, \tag{4.11}$$

kde μ je střední hodnota jasu počítaná přes všechny pixely.

Normalizovaný rozptyl [10, 12]

Funkce kompenzuje rozdíly ve střední hodnotě jasu mezi více snímky. Je vyjádřena jako

$$F_{norm_var} = \frac{1}{M \cdot N \cdot \mu} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [i(x,y) - \mu]^2. \tag{4.12}$$

Autokorelace (Vollath F_4) [13, 14]

Vollath v [13, 14] uvádí dvě funkce pro vyhodnocení ostrosti založené na korelaci pixelů snímku. Měly by být velmi odolné vůči šumu. První z nich je popsaná vztahem

$$F_{auto_corr} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} i(x,y) \cdot i(x+1,y) - \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} i(x,y) \cdot i(x+2,y). \tag{4.13}$$

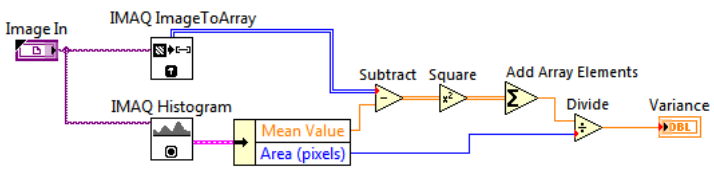
Korelace založená na směrodatné odchylce (Vollath F_5) [13, 14]

Druhá z funkcí založených na korelaci obsahuje rozdíl se směrodatnou odchylkou:

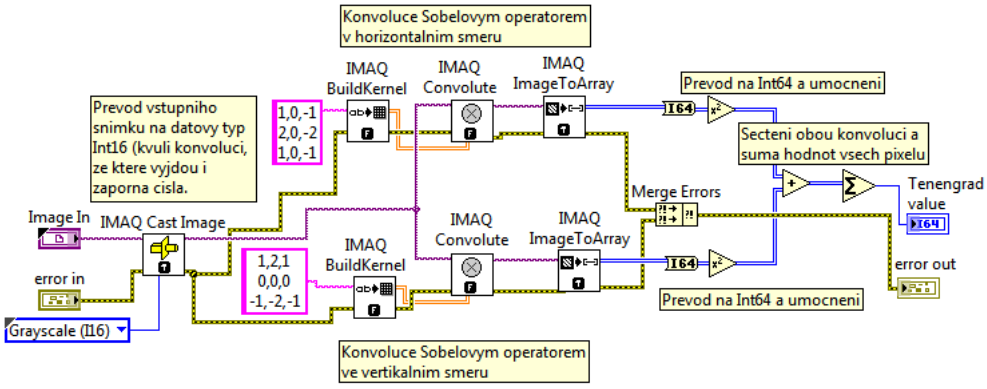
$$F_{corr_stddev} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} i(x,y) \cdot i(x+1,y) - M \cdot N \cdot \mu^2. \tag{4.14}$$

4.3. Testování funkcí pro měření ostrosti snímku

Funkce hodnotící ostrost snímku uvedené v předchozím oddílu byly implementovány za účelem výběru té nejvhodnější pro automatické zaostřování ve vyvinuté DLL knihovně. Na obrázcích 4.3 a 4.4 jsou zobrazeny ukázky implementace funkcí F_{var} a $F_{tenengrad}$ v NI LabVIEW.



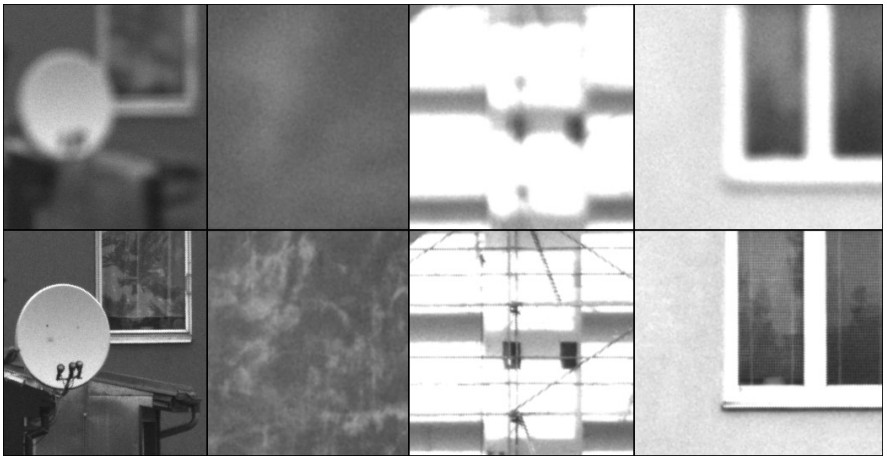
Obrázek 4.3.: Ukázka zdrojového kódu s implementací funkce F_{var}.



Obrázek 4.4.: Ukázka zdrojového kódu s implementací funkce $F_{tenengrad}$.

Funkce byly testovány na videosignálu z kamery AVT Prosilica GT2750 na venkovních scénách. Motoru kamery byl odeslán příkaz pro kontinuální pohyb směrem ke správné poloze zaostření objektivu a byla nahrávána sekvence snímků při tomto pohybu. Pohyb motoru byl zastaven až po posunutí objektivu za polohu nejlepšího zaostření (tato poloha odpovídá vrcholům zobrazených průběhů). V určitém výřezu obrazu z kamery pak byla všemi funkcemi průběžně počítána ostrost každého snímku ve videosignálu. Jelikož funkce ostrosti nabývají řádově jiných hodnot, byly výsledky pro účely porovnání normovány (minimální hodnotě průběhu odpovídá nula a maximální hodnotě jednička).

Na obrázku 4.5 jsou zobrazeny čtyři výřezy testovaných scén. Snímky v horní řadě odpovídají maximálnímu rozostření objektivu, snímky dole byly zaznamenány v poloze nejlepšího zaostření kamery.



Obrázek 4.5.: Výběr scén, na kterých byly testovány funkce pro výpočet ostrosti.

První výřez reprezentuje standardně kontrastní scénu. Průběh jednotlivých funkcí ostrosti je zobrazen vlevo na obrázku 4.6 na následující straně. Funkce $F_{tenengrad}$ a F_{auto_corr} vytvářejí užší a

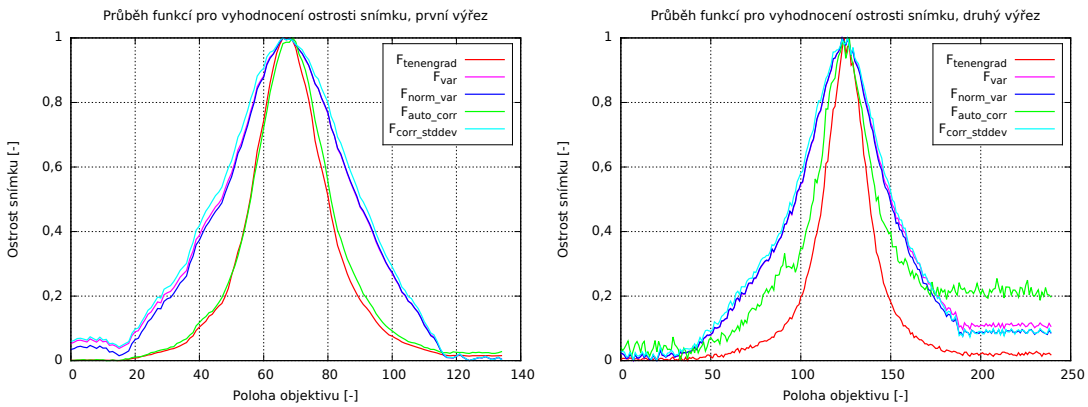
tím pádem i strmější průběh. Obě funkce také obsahují mnohem menší šum než funkce zbývající. V tomto případě by byly vhodnými adepty pro hledání nejostřejší polohy objektivu.

Vpravo na obrázku 4.6 jsou zobrazeny průběhy funkcí pro vyhodnocení ostrosti pro druhý výřez z obrázku 4.5. Výřez zastupuje málo kontrastní scénu s malým počtem detailů. Je zřejmé, že pro tento typ scén je jednoznačným favoritem funkce $F_{tenengrad}$, která oproti ostatním podává nejužší a nejméně zašuměný průběh (způsobeno průměrováním okolních hodnot pixelu, ve kterých se počítá Sobelův gradient). Větší grafy z obrázku 4.6 jsou zobrazeny na obrázcích B.4 a B.5 v příloze B (výřezy 2 a 4).

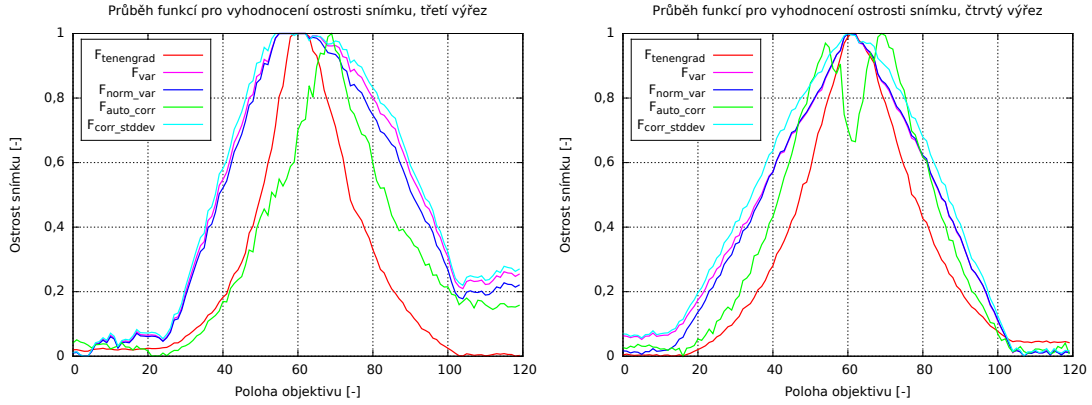
Na obrázku 4.7 vlevo jsou průběhy funkcí odpovídající třetímu výřezu z obrázku 4.5. Výsledkem funkcí F_{var}, F_{norm_var} a F_{corr_stddev} jsou velmi podobné průběhy o největší šířce a s velkým množstvím šumu. Maximum průběhu funkce F_{auto_corr} v případě této scény neodpovídá poloze maximálního zaostření. Pravděpodobným důvodem je skutečnost, že výřez obsahuje velké množství horizontálních čar, které jsou v polohách těsně vedle maximální ostrosti deformovány aliasingem (patrné při zvětšení výřezu) a tím pádem mají ve výpočtu funkce velký příspěvek (počítá se rozdíl součinů sousedních pixelů v horizontálním směru).

Průběhy funkcí hodnotících ostrost čtvrtého výřezu jsou na obrázku 4.7 vpravo. Nejužší a nejméně zašuměný průběh vykreslí opět funkce $F_{tenengrad}$. Funkce F_{auto_corr} by v tomto případě byla pro určení polohy zaostření absolutně nepoužitelná. V okolí polohy maximální ostrosti totiž souměrně nabývá dvou maxim a ve správné poloze prudce klesá do lokálního minima. Důvodem je nejspíše jemná struktura na záclonĕ v okně, která je nejvíce detailní právě v poloze maximální ostrosti. Z podobného principu jako v předchozím případě proto funkce selže. Větší grafy z obrázku 4.7 jsou zobrazeny na obrázcích B.7 a B.8 v příloze B (výřezy 7 a 9).

Další naměřené průběhy a snímky celých testovacích scén je možné nahlédnout v příloze B.



Obrázek 4.6.: Průběhy funkcí pro vyhodnocení ostrosti pro první a druhý výřez z obrázku 4.5.



Obrázek 4.7.: Průběhy funkcí pro vyhodnocení ostrosti pro třetí a čtvrtý výřez z obrázku 4.5.

4.4. Výsledná volba funkce pro měření ostrosti snímku

Podle výše diskutovaných průběhů je jasným kandidátem na nejvhodnější funkci pro vyhodnocení maximální míry zaostření objektivu funkce $F_{tenengrad}$.

Santos et al. v [8] staví na první místo funkci F_{auto_corr} a pokud není uvažována výpočetní náročnost, na druhé místo řadí právě funkci $F_{tenengrad}$[4]. Stejná funkce byla použita i při vývoji algoritmu automatického zaostřování pro kompaktní fotoaparát v [4].

Během výše uvedených testů provedených při vývoji DLL knihovny se funkce F_{auto_corr} projevila jako nevhodný kandidát. Funkce F_{var}, F_{norm_var} a F_{corr_stddev} neprodukovaly dostatečně úzké a strmé průběhy, navíc byly poměrně značně zatíženy šumem.

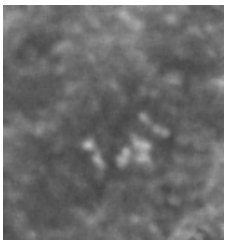
Vzhledem k tomu, že výpočetní výkon použitého PC byl dostatečný a průběžné počítání funkce $F_{tenengrad}$ nezpůsobovalo velké vytížení procesoru ani snížení snímkové frekvence videosignálu, byla pro implementaci automatického zaostřování v knihovně zvolena právě tato funkce.

4.5. Filtrace průběhu funkce ostrosti snímku

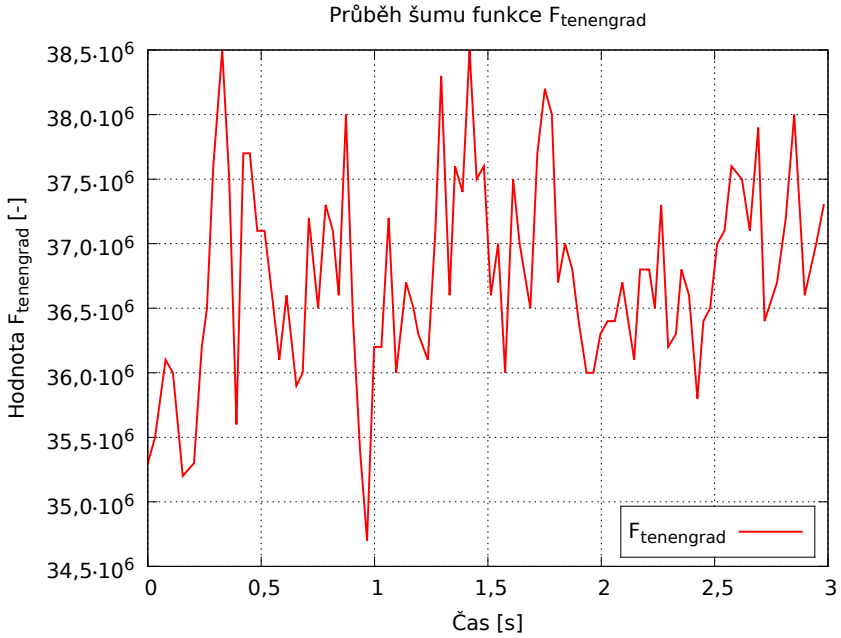
Přestože funkce $F_{tenengrad}$ sama vyhlazuje šum v obraze, ukázala se tato filtrace jako nedostatečná. Zejména na malých výřezech málo kontrastních scén je šum v obraze natolik velký, že by jej mohl algoritmus pro řízení automatického zaostření považovat za skutečné vrcholy ostrosti na scéně.

Ukázka výřezu částečně zaostřené málo kontrastní scény je na obrázku 4.8 na následující straně (celá scéna je zachycena na obrázku B.2 v příloze C). Průběh funkce $F_{tenengrad}$ počítané nad tímto výřezem v čase je na obrázku 4.9 (hodnoty funkce záměrně nebyly normalizovány).

[4] Je ovšem nutné připomenout, že testy funkcí v [8] probíhaly na základě mikroskopických snímků. Venkovní reálné scény snímané kamerami mají jiné charakteristické rysy.



Obrázek 4.8.: Ukázka výřezu scény se šumem v obraze.



Obrázek 4.9.: Ukázka zašuměného průběhu funkce $F_{tenengrad}$.

Na začátku hledání polohy objektivu s největší ostrostí snímků nemá algoritmus ostření představu o tom, jakých maximálních hodnot bude funkce $F_{tenengrad}$ nabývat[5]. Proto by šum funkce mohl být interpretován jako falešný růst či klesání a docházelo by k nesprávné funkci algoritmu. Z tohoto důvodu bylo rozhodnuto průběh funkce $F_{tenengrad}$ před vstupem do algoritmu automatického zaostřování filtrovat.

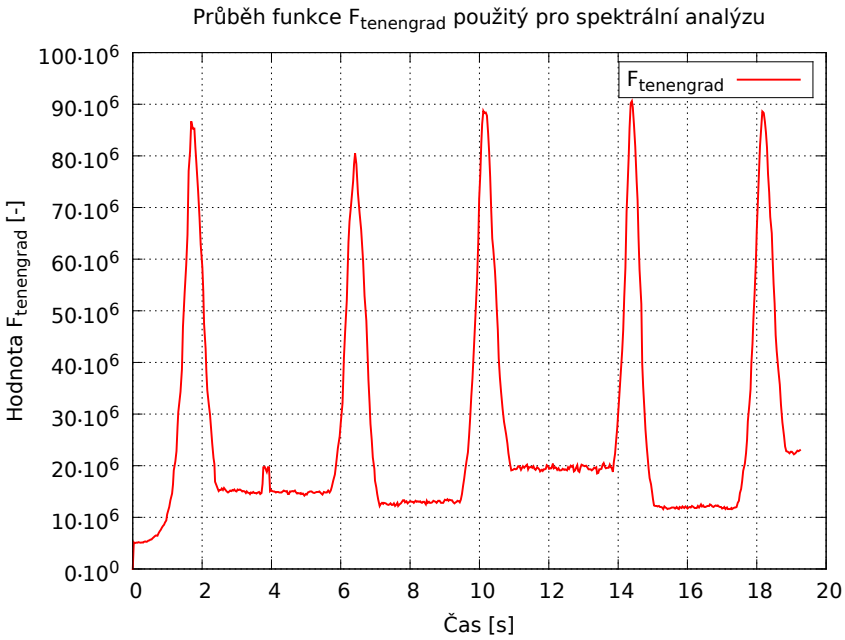
Šum funkce ostrosti má vysokofrekvenční charakter, proto bylo nutné implementovat vhodný filtr typu dolní propust. Tento typ filtru je však charakterizován fázovým zpožděním vstupního a výstupního signálu. Bylo nutné nalézt vhodný kompromis mezi mírou filtrace a zpožděním filtrované funkce $F_{tenengrad}$.

[5] Maximální hodnotu funkce ostrosti bychom získali, pokud bychom projeli celý rozsah zaostření objektivu a průběžně počítali její hodnoty.

Výběr filtru a jeho nastavení byly prováděny na videosignálu z kamery zabírající venkovní scény. Objektivem přitom bylo pohybováno do různých poloh zaostření. Důležité bylo zkoumat průběhy filtrované funkce $F_{tenengrad}$ při přejíždění tam a zpět přes vrcholy průběhu funkce ostrosti tak, aby se projevila jejich dynamika a bylo zřejmé fázové zpoždění filtrovaného signálu.

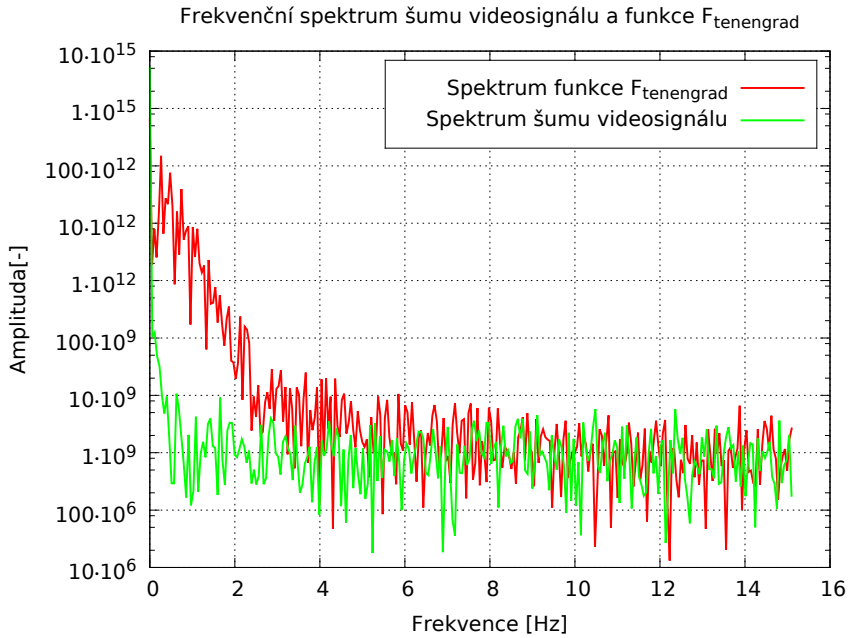
Testovány byly následující filtry: plovoucí průměr, plovoucí medián a dolní propusti typu Butterworth, Bessel a Chebyshev. Pro poslední tři filtry typu dolní propust bylo nutné odhadnout jejich zlomovou frekvenci.

Na obrázku 4.10 je zobrazen průběh, na kterém byla prováděna analýza frekvenčního spektra funkce $F_{tenengrad}$. Objektivem kamery bylo pohybováno vpřed a vzad přes pozici nejlepšího zaostření. Zároveň bylo měřeno frekvenční spektrum šumu videosignálu při jedné poloze zaostření objektivu (průběh šumu byl podobný jako na obrázku 4.9 na předchozí straně). Frekvenční spektra obou signálů jsou na obrázku 4.11 na následující straně (vertikální osa má logaritmické měřítko).



Obrázek 4.10.: Průběh funkce $F_{tenengrad}$ použitý pro analýzu frekvenčního spektra.

Z porovnání frekvenčních spekter je zřejmé, že šum videosignálu má v podstatě charakter bílého šumu (je rozprostřen přes všechny frekvence), zatímco spektrum průběhu funkce $F_{tenengrad}$ obsahuje dominantní frekvence do zhruba $2,5$ Hz. Tato frekvence je samozřejmě závislá na rychlosti pohybu objektivu, nicméně motor na dodané sestavě se pohyboval pouze jednou konstantní rychlostí. Průběhy funkce ostrosti při pohybu objektivu přes pozici s maximální ostrostí snímků jsou tedy pro různé scény víceméně podobné a liší se pouze amplitudou. Zlomová frekvence filtrů typu dolní propust proto byla nastavena na 3 Hz.



Obrázek 4.11.: Frekvenční spektra šumu statické scény a funkce $F_{tenengrad}$ při přejíždění pozice nejlepšího zaostření

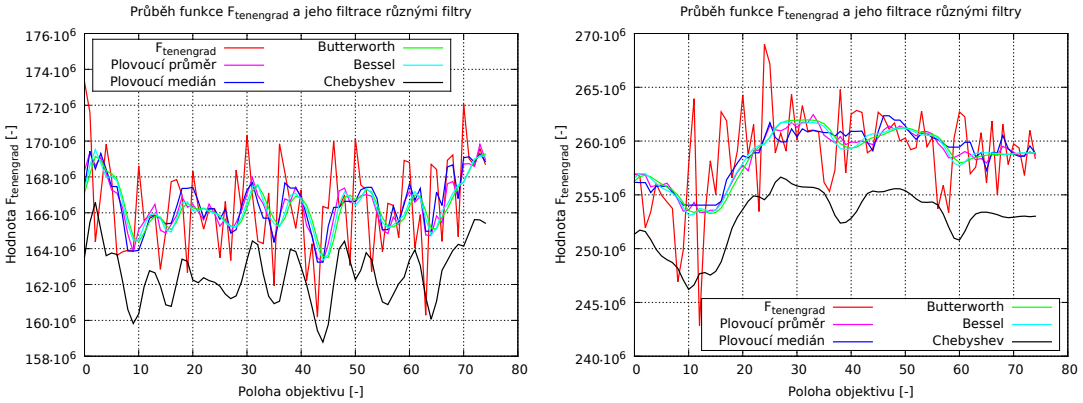
Na obrázcích 4.12, 4.13 a 4.14 jsou zobrazeny průběhy funkce $F_{tenengrad}$ a průběhy filtrací této funkce s použitím různých filtrů. Větší grafy jsou zobrazeny na obrázcích C.1 až C.6 v příloze C.

Na obrázku 4.12 jsou zobrazeny průběhy původní funkce ostrosti a průběhy filtrací této funkce při konstantní pozici objektivu (jedná se tedy o šum videosignálu).

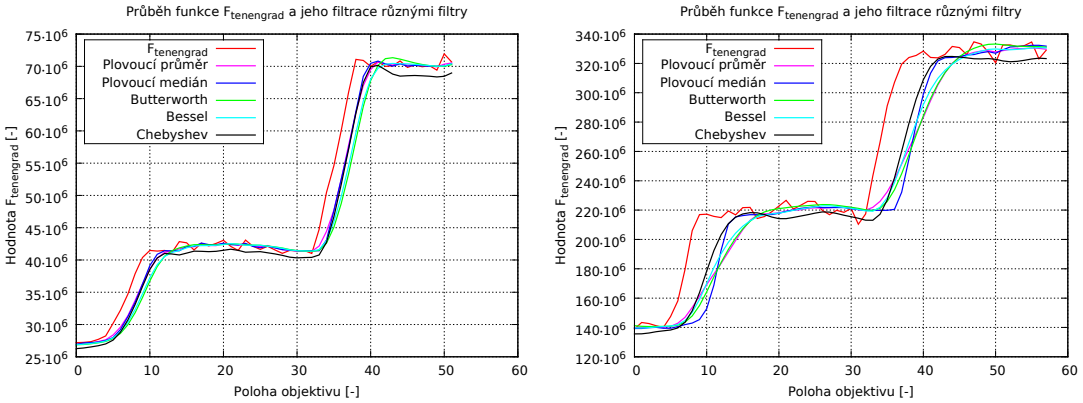
Na obrázku 4.13 jsou průběhy funkce ostrosti a průběhy filtrací při skokových posunech objektivu a na obrázku 4.14 na straně 28 jsou zobrazeny průběhy funkce ostrosti a průběhy filtrací při přejezdu přes polohu s maximální ostrostí obrazu.

Na grafech na obrázcích vlevo jsou zobrazeny průběhy z prvního měření, kdy byl řád filtrů typu dolní propust nastaven na 2, zlomová frekvence na 3 Hz a pro Chebyshevův filtr byl parametr zvlnění v propustném pásmu („ripple“) nastaven na 0,2 dB. Statistické filtry plovoucí průměr a medián byly nastaveny na délku 15 % vstupní snímkové frekvence videosignálu, tedy pokud kamera snímala 30 snímků za vteřinu, byl počítán průměr a medián z hodnot funkce ostrosti za poslední 4 snímky.

Na grafech na obrázcích vpravo je zachyceno druhé měření, při kterém byla zlomová frekvence filtrů typu dolní propust nastavena na 1,5 Hz (řád filtrů a nastavení Chebyshevova filtru bylo stejné jako při prvním měření). Statistické filtry plovoucí průměr a medián byly nastaveny na délku 30 % vstupní snímkové frekvence videosignálu, tedy pokud kamera snímala 30 snímků za vteřinu, byl počítán průměr a medián z hodnot funkce ostrosti za posledních 9 snímků.



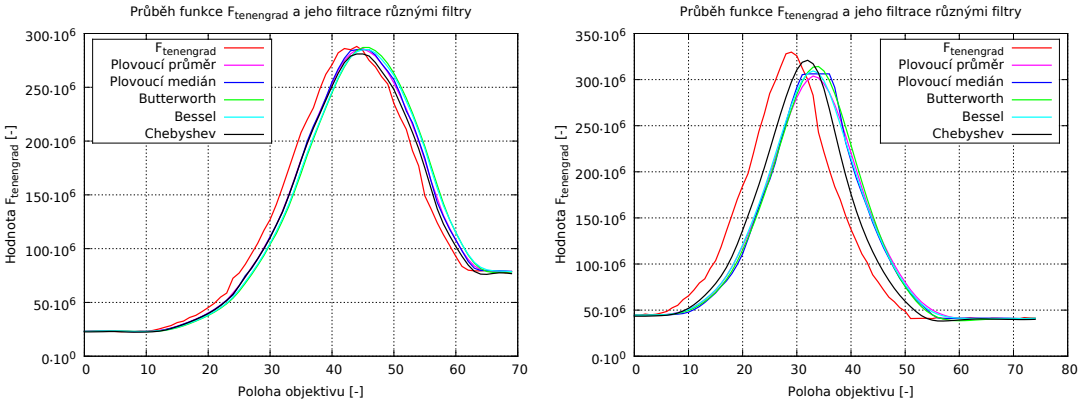
Obrázek 4.12.: Průběhy funkce ostrosti $F_{tenengrad}$ filtrované pěti různými filtry. Vlevo první měření, vpravo druhé měření.



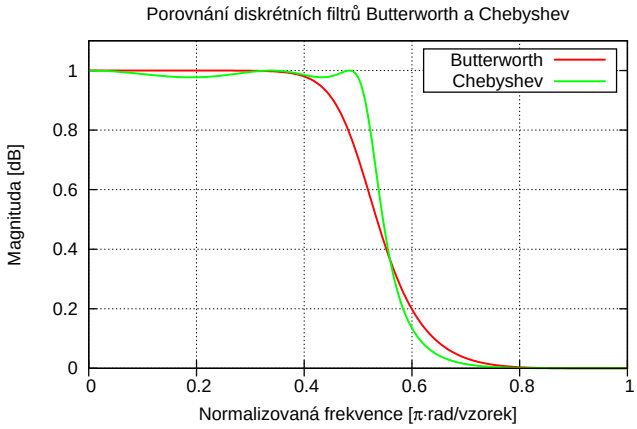
Obrázek 4.13.: Průběhy funkce ostrosti $F_{tenengrad}$ filtrované pěti různými filtry. Vlevo první měření, vpravo druhé měření.

Z grafů průběhů z prvního měření (na obrázcích vlevo) je zřejmé, že všechny filtry vykazují víceméně podobný průběh, míra filtrace i fázové zpoždění jsou na srovnatelné úrovni. Z filtrů typu dolní propust má Chebyshevův filtr nejmenší fázové zpoždění, avšak zároveň snižuje amplitudu. Vysvětlení tohoto jevu ilustruje obrázek 4.15 na následující straně. Řád obou filtrů na obrázku je 5, zlomová frekvence byla nastavena na $0,5\,\pi\cdot\mathrm{rad}/\mathrm{vzorek}$ a parametr zvlnění byl nastaven na $0,2\,\mathrm{dB}$. Frekvenční charakteristika Chebyshevova filtru má na zlomové frekvenci strmější útlum než charakteristika Butterworthova filtru, avšak na propouštěných frekvencích je přítomno zvlnění amplitudy.

Grafy průběhů z druhého měření (na obrázcích vpravo) demonstrují vliv snížení zlomové frekvence za účelem většího vyhlazení signálu. Chebyshevův filtr opět snižuje amplitudu a má nejmenší fázové zpoždění. Z průběhu skoků je patrný drobný překmit Butterworthova filtru.



Obrázek 4.14.: Průběhy funkce ostrosti $F_{tenengrad}$ filtrované pěti různými filtry. Vlevo první měření, vpravo druhé měření.



Obrázek 4.15.: Porovnání frekvenčních charakteristik digitálního Butterworthova a Chebyshevova filtru typu dolní propust.

Filtr typu plovoucí medián je v tomto případě nepoužitelný, protože příliš deformuje vrchol funkce ostrosti.

Je zřejmé, že při druhém měření už byly filtry nastaveny příliš silně a jimi reprezentovaný vrchol ostrosti snímku je díky velkému fázovému zpoždění oproti správné poloze zásadně posunut. Algoritmus automatického ostření by proto zastavil objektiv na pozici mimo skutečný vrchol ostrosti.

Jako optimální kompromis mezi mírou filtrace a fázovým zpožděním se ukázalo takové nastavení filtrů, jaké bylo použito při prvním měření. Jelikož se všechny filtry při zmíněném nastavení chovaly podobně, byl nakonec díky jednoduchosti jeho nastavení a implementace zvolen filtr typu plovoucí průměr.

4.6. Detekce růstu a klesání funkce ostrosti

Aby bylo možné rozhodnout o směru pohybu objektivu za účelem dosažení polohy s nejlepší ostrostí snímků, je nutné po spuštění pohybu detekovat, jestli funkce ostrosti obrazu roste nebo klesá.

K tomu účelu byl vyvinut detektor růstu a klesání. Jde v podstatě o funkci, která diferencuje už zfiltrovanou hodnotu funkce $F_{tenengrad}$. Je používán výpočet zpětné první diference, která je normalizována aktuální hodnotou funkce $F_{tenengrad}$. Hodnota normalizované diference y_i pro současný snímek videosignálu z kamery má tvar

$$y_i = \frac{x_i - x_{i-1}}{x_i},$$

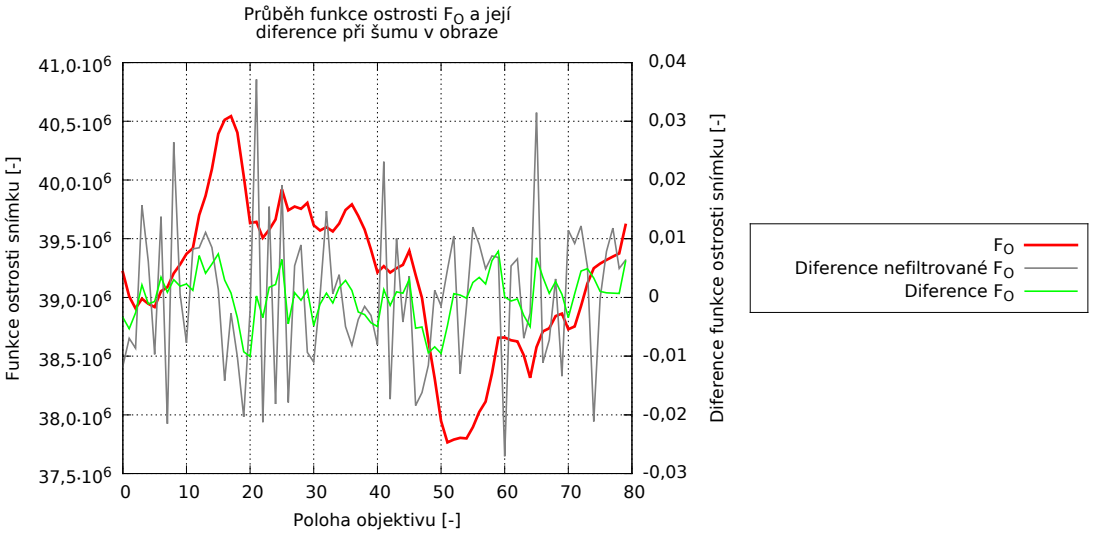
kde je x_i je hodnota funkce$F_{tenengrad}$ v aktuálním snímku. Díky normalizaci nabývá diference porovnatelných hodnot pro všechny hodnoty funkce $F_{tenengrad}$ v různých výřezech. Diference nabývá kladných hodnot, pokud funkce ostrosti roste a záporných hodnot, když funkce ostrosti klesá. Na vrcholu je diference rovna nule.

Detekce růstu a klesání je založena na nastavení prahů pro hodnotu diference funkce ostrosti. Pokud diference v x po sobě jdoucích snímcích překročí nastavený limit y_H, je signalizován růst. Podobně, když diference v x po sobě jdoucích snímcích klesne pod limit y_L, je signalizováno klesání.

Po několika experimentech se pro danou sestavu kamery a objektivu ukázalo jako nejvhodnější nastavit limity $y_H = 0,035$, $y_L = -0,035$ a $x = 4$ (diference musí překročit limit alespoň ve 4 po sobě jdoucích snímcích).

Na obrázku 4.16 na následující straně je zobrazen průběh diference filtrované a nefiltrované funkce $F_{tenengrad}$ při šumu ve videosignálu. K šumu se navíc přidával i pohyb některých objektů ve výřezu scény (listí ve větru). I když je v průběhu funkce ostrosti mnoho růstu a klesání, protože nebyly splněny podmínky překročení limitů, detektor nezahlásí ani růst, ani klesání. Algoritmus automatického zaostřování se tedy v případě šumu v obraze nespustí. Pro porovnání je na obrázku zobrazen také průběh diference nefiltrované funkce $F_{tenengrad}$. Je zřejmé, že průběh diference nefiltrované funkce je dramatičtější a nabývá větší amplitudy než diference filtrovaného signálu.

Na obrázku 4.17 na straně 31 je znázorněn průběh diference funkce $F_{tenengrad}$ při přejezdu objektivu přes pozici maximální ostrosti obrazu. S výše uvedenými parametry nastavení detektoru by byl na pozici 38 detekován růst funkce ostrosti a na pozici 90 její klesání. Limity y_H a y_L jsou znázorněny fialovými přerušovanými čarami a pozice růstu a klesání jsou vyznačeny černými vertikálními šipkami. Pro úplnost je v grafu opět zobrazena i diference nefiltrované funkce $F_{tenengrad}$.



Obrázek 4.16.: Průběh funkce $F_{tenengrad}$ a její diference při šumu v obraze.

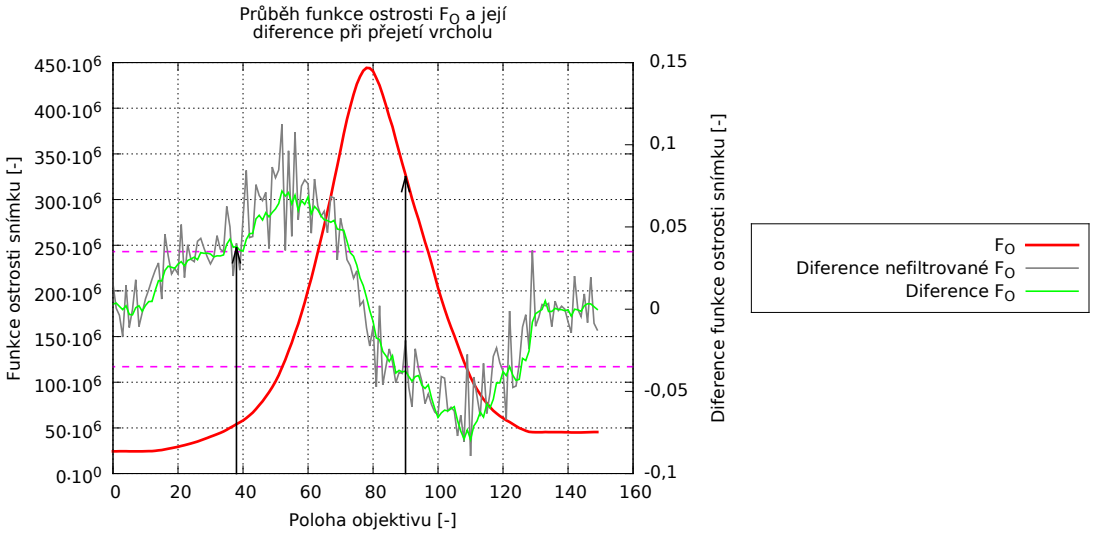
Nastavením parametrů lze ovlivnit rychlost rozhodování detektoru růstu a klesání. Nižší limity zapříčiní rychlejší reakci, ale detektor je potom málo robustní a reaguje falešně i na šum a pohyblivé scény – například pohyb stromů nebo listí ve větru. Nastavení detektoru je také samozřejmě vázáno na použitý motor, respektive jeho dynamiku. Pokud by byl k dispozici rychlejší motor, bylo by nutné nastavit detektor tak, aby reagoval rychleji.

4.7. Algoritmus automatického zaostření kamery

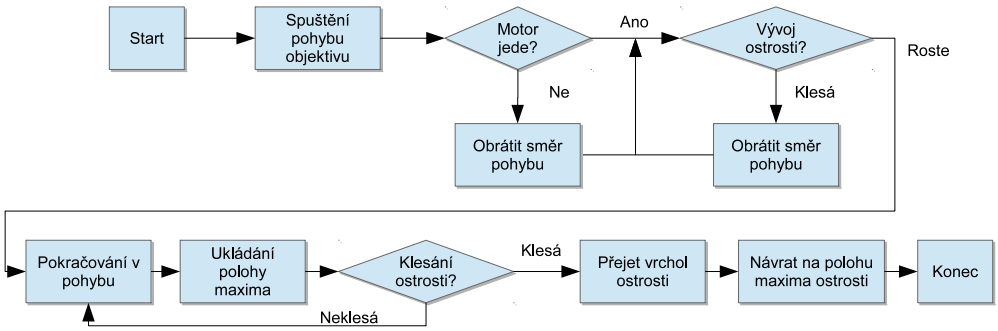
Algoritmus automatického zaostření kamery probíhá podle vývojového diagramu na obrázku 4.18.

Po zavolání knihovní funkce pro spuštění automatického zaostření kamery je nejprve spuštěn pohyb ostření objektivu směrem ke vzdáleným objektům. Pokud se krokový motor nerozběhne, znamená to, že se zdvih objektivu nachází v pozici na koncovém snímači a proto je spuštěn pohyb opačným směrem.

Po spuštění pohybu objektivu algoritmus reaguje na výstup detektoru růstu a klesání. Pokud ostrost snímků s pohybem objektivu klesá, ostření zjevně probíhá špatným směrem a je nutné obrátit směr pohybu. Jakmile začne ostrost obrazu růst je očekáván příchod vrcholu funkce ostrosti. Proto se v každé iteraci kontroluje, jestli je současná hodnota ostrosti vyšší než předchozí a pokud ano, je uložena poloha motoru v této iteraci. Jinými slovy, s růstem funkce se neustále ukládá aktuální pozice motoru. Poslední uložená pozice potom odpovídá pozici maxima funkce ostrosti a tím pádem i pozici nejlepšího zaostření objektivu.



Obrázek 4.17.: Průběh funkce $F_{tenengrad}$ a její diference při přejetí vrcholu ostrosti.



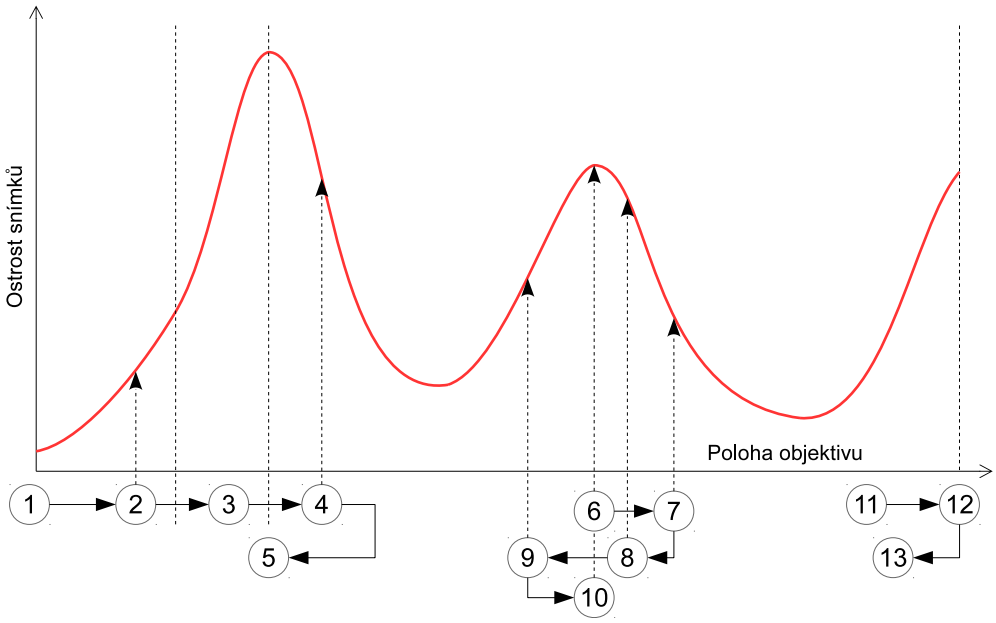
Obrázek 4.18.: Vývojový diagram algoritmu pro automatické zaostření kamery.

Protože funkce ostrosti nyní monotónně roste a očekává se příchod jejího vrcholu, sníží se limity pro hodnotu diference v detektoru růstu a klesání tak, aby bylo klesání detekováno rychleji. Následné klesání funkce ostrosti je považováno za indikaci přejetí vrcholu této funkce. Proto se okamžitě obrátí směr motoru a vyšle se pokyn pro návrat na uloženou polohu maximální ostrosti. Po dokončení polohování motoru je indikováno dokončení automatického zaostření.

Schématicky je výše uvedený postup znázorněn také na obrázku 4.19 na následující straně. Čísla 1 – 5 v kroužcích odpovídají následujícím krokům:

1. start automatického zaostřování,
2. nalezen růst ostrosti,
3. pásmo ukládání polohy objektivu,
4. nalezeno klesání ostrosti,

5. návrat na polohu s maximální ostrostí obrazu.



Obrázek 4.19.: Schématický průběh hledání polohy objektivu s maximální ostrostí obrazu.

Na obrázku 4.19 je znázorněn i další příklad hledání správné pozice zaostření. Pokud by algoritmus hledání začal například přímo na vrcholu ostrosti (bod 6), po spuštění pohybu objektivu by bylo velmi rychle detekováno klesání (v bodě 7) a byl by obrácen směr hledání. Odtud by už chování hledání bylo analogické s předchozím: nejprve by byl ihned detekován růst (bod 8), poté klesání (bod 9) a nakonec by byl spuštěn návrat na pozici uložené maximální hodnoty ostrosti (do bodu 10).

V okolí koncových spínačů je nutné specifické chování algoritmu. Obecně platí, že pokud objektiv narazí na jednu z koncových poloh, je okamžitě obrácen směr pohybu hledání maxima funkce ostrosti. Pokud byl ovšem před dojezdem na koncový spínač detekován růst funkce, ale nikoliv už její klesání (tedy byl očekáván vrchol, který nenastal), je obrácen směr hledání a první klesání v novém směru je ignorováno (jinak by se opět podle standardního chování algoritmu obrátil směr a objektiv by dojel na stejný koncový spínač). Situaci reprezentují body 11, 12 (koncový spínač) a 13 na obrázku 4.19.

Pokud dojde kdykoliv v průběhu hledání pozice s maximální ostrostí snímků k postupnému sepnutí obou koncových snímačů, je indikováno neúspěšné zaostření (v celém rozsahu objektivu nebyl nalezen žádný dominantní vrchol funkce ostrosti).

Rozšířením algoritmu jednorázového zaostření je tzv. průběžné zaostřování. V průběhu hledání vrcholu funkce ostrosti $F_{tenengrad}$ se vedle ukládání polohy objektivu při dosažení maxima ukládá i hodnota maxima funkce při dané poloze. Pokud je aktivní režim průběžného zaost-

řování, po dokončení prvního zaostření se periodicky v uživatelem zadaných intervalech kontroluje hodnota funkce $F_{tenengrad}$. Pokud hodnota funkce klesne pod určenou mez (např. 80 % uloženého maxima), je zahájeno nové zaostření. V tomto případě ovšem při zahájení nového hledání objektiv startuje velice blízko původnímu vrcholu ostrosti, a tudíž je nová pozice maximální ostrosti určena velice rychle (podobně jako v bodech 6 – 10 na obrázku 4.19 na předchozí straně). Tímto postupem je dosaženo průběžného zaostřování i na objekt, který se pohybuje, případně pokud se mění jas a kontrast ve scéně (čímž je druhotně ovlivněna i hodnota funkce $F_{tenengrad}$).

4.8. Popis hardware pro zaostřování kamery

Kamera byla namontována na hliníkový posuv, který byl pohyblivý vzhledem k ohnisku objektivu. Pohyb posuvu s kamerou zajišťoval přes převodovku krokový motor Faulhaber PRECIstep® AM 1020. Pro kontrolu dorazů posuvu byly na koncích rozsahu jeho zdvihu mechanické koncové spínače. Řízení motoru bylo implementováno pomocí vývojové platformy Arduino Micro[6]. Elektronické zapojení prototypu bylo realizováno na nepájivém propojovacím poli.

Deska Arduino překládala příkazy odeslané z PC přes sériové rozhraní[7] na pulzní ovládání krokového motoru. Přes sériové rozhraní byla zpět do PC rovněž odesílána informace o poloze motoru (resp. počet odeslaných pulzů, viz dále), případně o dosažení koncového spínače.

Protože krokový motor neobsahoval enkodér polohy, ovládání motoru bylo zákazníkem implementováno pouze jako přímovazební. Pro pohyb posuvu objektivu na určitou pozici bylo nutné z PC přes sériovou linku poslat příkaz s parametrem počtu pulzů. Deska Arduino spustila odesílání impulzů pro krokový motor a po vygenerování tohoto počtu pulzů odeslala zpět přes sériovou linku informaci o tom, že byl pohyb dokončen. Nebylo tedy možné zjistit, jestli se posuv někde zasekl či selhala převodovka. Jedinou skutečnou zpětnovazební informací bylo odeslání informace o dosažení koncového spínače.

Navíc byla poloha posuvu vždy počítána relativně, tzn. že při spuštění mikroprocesoru desky Arduino se poloha vynulovala a pokud jel motor na jednu stranu, poloha (pulzy motoru) se přičítala a na druhou stranu se poloha odečítala. Rozsah celého zdvihu posuvu kamery odpovídal zhruba 14000 jednotkám (pulzům motoru). Rychlost motoru a převod převodovky byly nastaveny tak, že celý rozsah posuvu objektivu byl překonán za cca 13 vteřin.

Seznam všech příkazů pro ovládání motoru včetně jejich popisu je uveden v příloze F. Pro zaostřování objektivu bylo nutné používat především následující příkazy:

- `MoveHSfwd` / `MoveHSbwd` – spustí pohyb motoru vpřed nebo vzad,

[6] Více informací viz `http://arduino.cc/en/Main/ArduinoBoardMicro`.

[7] Vývojová deska Arduino Micro byla k PC připojena přes USB port. Deska obsahuje čip pro převod USB rozhraní na sériovou linku, ovladač desky tedy zpřístupní v operačním systému Microsoft Windows virtuální sériový port.

- `MoveStop` – zastaví pohyb motoru (zpět je odeslána informace o aktuální poloze),
- `MoveHSfwdStepCount:X` / `MoveHSbwdStepCount:X` – spustí pohyb motoru vpřed nebo vzad o zadaný počet pulzů `X` (po vyslání požadovaného počtu pulzů je odeslána informace o dokončení posuvu),
- `GetPos` – vyčtení aktuální pozici motoru.

Deska Arduino navíc po dosažení koncového spínače odeslala na sériovou linku řetězec `EndSwitchX:ON` (`X` je číselný identifikátor jednoho nebo druhého koncového spínače).

4.9. Ovládání automatického zaostřování z DLL knihovny

Pro spuštění automatického zaostření je nejprve nutné kliknout levým tlačítkem myši do okna s obrazem z kamery. Událost s informací o tomto kliknutí odešle knihovna koncové aplikaci pomocí Windows zprávy systémovou funkcí

```
SendMessage(int hWnd, 0x9000, int wParam, int lParam).
```

Prvním parametrem `hWnd` je reference na hlavní okno nadřazené aplikace (handle), která knihovnu používá, kód zprávy je pevně zvolen jako 0x9000 a parametry `wParam` a `lParam` odpovídají horizontální a vertikální souřadnici v obraze.

Koncová aplikace po přijmutí Windows zprávy zavolá knihovní funkci

```
void StartAF(uint16_t x, uint16_t y)
```

a spustí tak zaostření na zadané souřadnice. Automatické ostření je možné kdykoliv zastavit funkcí `void StopAF()`.

Knihovna o průběhu zaostřování informuje nadřazenou aplikaci opět pomocí Windows zpráv. Informace o událostech zahájení, dokončení či poruše ostření se odesílají systémovou funkcí

```
SendMessage(int hWnd, 0x9002, int wParam, int lParam).
```

Prvním parametrem `hWnd` je opět handle hlavního okna nadřazené aplikace. Parametr `wParam` obsahuje informaci o události zaostřování:

- `wParam = 0` – automatické zaostřování bylo zahájeno,
- `wParam = 1` – automatické zaostřování bylo dokončeno,
- `wParam = 2` – automatické zaostřování selhalo.

Po spuštění automatického zaostřování je do videosignálu z kamery vykreslen žlutý obdélník odpovídající oblasti na kterou je zaostřováno. Po dokončení ostření se změní barva obdélníku na zelenou, případně červenou, pokud se knihovně nepodařilo nepodařilo zaostřit.

Nastavení režimu automatického zaostřování probíhá zavoláním funkce

```
void AFFocusMode(uint8_t nMode),
```

kde číselný parametr `nMode` určuje požadovaný režim:

- `nMode = 0` – manuální ostření,
- `nMode = 1` – jednorázové automatické zaostření,
- `nMode = 2` – průběžné automatické zaostřování.

Průběžné zaostřování je nastavováno funkcemi

```
void AFContFocusInterval(double fTime) a

void AFContFocusTreshold(uint8_t nTresh),
```

kde parametry `fTime` a `nTresh` reprezentují interval kontroly hodnoty funkce ostrosti a práh, při jehož překonání se spustí nové přeostření.

Velikost oblasti, na kterou bude zaostřováno (neboli výřezu obrazu, ve kterém bude vyhodnocována ostrost) je nastavována funkcí

```
void AFAreaSize(uint16_t width, uint16_t height),
```

kde parametry `width` a `height` určují šířku a výšku oblasti.

Knihovna umožňuje kdykoliv se přepnout do režimu manuálního zaostřování, které je ovládáno funkcemi

```
void FocusNear(int8_t nMove) a

void FocusFar(int8_t nMove),
```

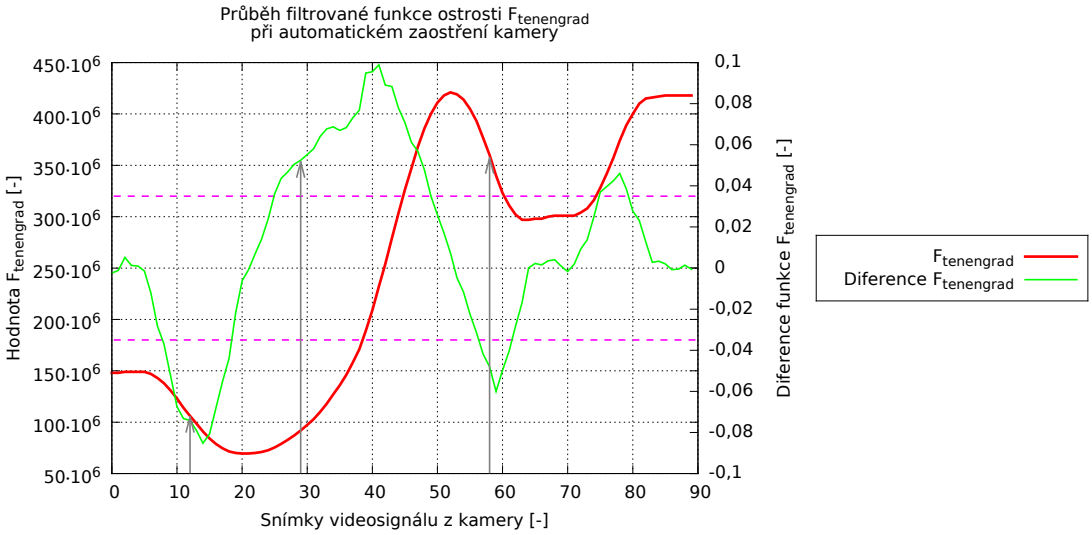
kde parametr `nMove` spouští a zastavuje pohyb motoru (platné hodnoty jsou 1 pro spuštění a 0 pro zastavení pohybu).

Kompletní seznam knihovních funkcí pro ovládání automatického zaostřování je uveden v dokumentaci knihovny v příloze H.

4.10. Praktické testy automatického zaostřování

Již v průběhu vývoje jednotlivých bloků algoritmu automatického zaostřování byly testovány jejich průběžné výsledky a chování. Tomto oddíle bude popsáno výsledné chování implementovaného algoritmu včetně jeho dílčích omezení.

Na obrázku 4.20 na následující straně je zobrazen průběh filtrované funkce ostrosti $F_{tenengrad}$ a její diference při jednom z testů automatického zaostřování. Horizontální osa zastupuje jednotlivé příchozí snímky videosignálu z kamery. V grafu jsou šedými vertikálními šipkami znázorněna místa, kdy byl detektorem rozpoznán růst nebo klesání. Algoritmus automatického ostření



Obrázek 4.20.: Průběh filtrované funkce ostrosti $F_{tenengrad}$ a její diference při testech automatického zaostřování.

začal pohybem objektivu z výchozí pozice směrem k zaostření na nekonečno. Protože bylo rychle detekováno klesání funkce ostrosti (první šedá šipka) a obraz kamery se tedy rozostřoval, byl obrácen směr pohybu.

Následně již byl detekován růst (druhá šipka), tedy příznak toho, že se blíží vrchol funkce ostrosti neboli pozice nejlepšího zaostření. Vrchol byl identifikován po detekci dalšího klesání (třetí šipka), a tudíž se objektiv vrátil zpět na uloženou polohu s maximální ostrostí.

Z principu funkce implementovaného algoritmu je zřejmé, že při automatickém zaostřování zastaví objektiv na nejbližší poloze, ve které se objeví vrchol funkce ostrosti (splňující parametry nastavení detektoru růstu a klesání). Pokud je tedy zaostřovaný výřez scény umístěn na oblast, ve které je v ose objektivu za sebou několik objektů v různých vzdálenostech, automatické zaostřování je dokončeno při zaostření toho objektu, který byl v době spuštění ostření nejblíže výchozí poloze objektivu.

Další omezení použitého principu jsou společná pro všechny systémy spoléhající na tzv. pasivní automatické zaostřování. Všechny tyto systémy totiž ke správnému zaostření využívají pouze obrazovou informaci, která je silně závislá na kontrastu a detailech v obraze. I zde popisovaný algoritmus selže, pokud není výřez, na který se zaostřuje, dostatečně osvětlen, případně obsahuje málo rozlišitelných detailů.

Názornou ukázkou funkce implementovaného automatického zaostřování jsou videonahrávky průběhu zaostřování, které jsou uložené na DVD přiloženém k této diplomové práci.

4.11. Omezení použitého hardware

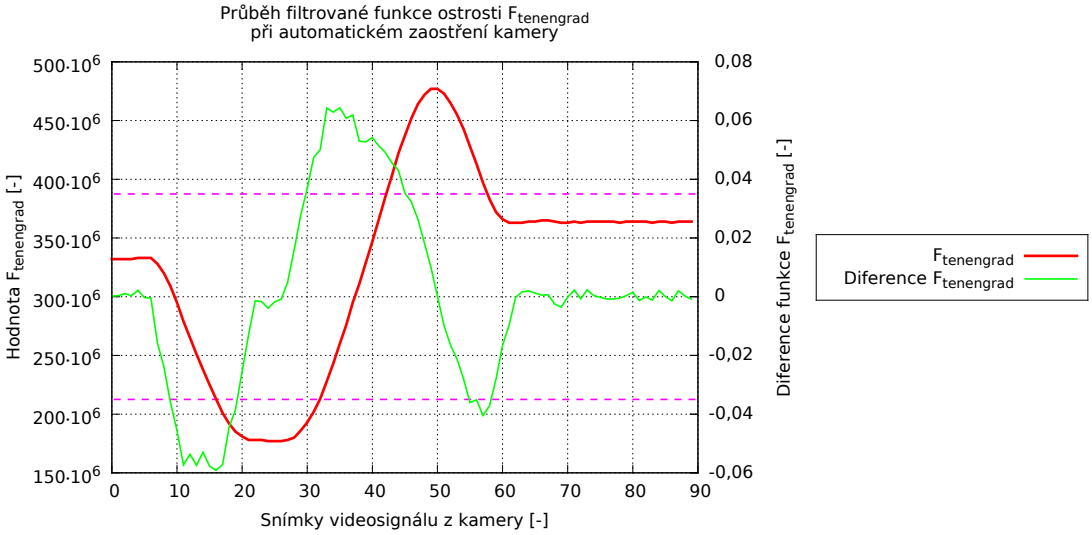
Z grafu na obrázku 4.20 na předchozí straně je zřejmé, že i když detektor zaznamenal obě klesání poměrně rychle, je mezi detekcí klesání a následnou změnou směru motoru určitá prodleva. Důvodem tohoto faktu je pravděpodobně dopravní zpoždění všech prvků systému. Po detekci klesání totiž musí nejprve algoritmus odeslat textový řetězec s příkazem pro motor po sériové lince. Rychlost komunikace byla dodavatelem řídící desky Arduino nastavena na rychlost 57600 baudů. Řídící deska ovšem byla k PC připojena přes USB port a v cestě tedy stál ještě čip pro převod USB rozhraní na sériovou linku. Požadavek na zastavení motoru a obrácení jeho směru potom musela ještě zpracovat deska Arduino, která posléze vyslala motoru příkaz k zastavení. Až poté byl motor skutečně zastaven a byl obrácen jeho směr.

Výkon algoritmu automatického zaostřování je silně závislý na rychlosti pohybu objektivu. Motor sestavy, pro kterou bylo automatické zaostřování implementováno, přesunul objektiv mezi oběma krajními polohami za cca 13 vteřin. Rychlost motoru má na výkon ostření zcela podstatný vliv. Zejména během ostření na scény, ve kterých se objekty pohybují v rovině kolmé na osu objektivu (tedy nikoliv směrem k a od objektivu), se totiž při jejich pohybu mění hodnota jakékoliv funkce pro vyhodnocení ostrosti obrazu. V průběhu funkce tak vznikají nové vrcholy, které může algoritmus automatického ostření mylně považovat za místa s nejlepší ostrostí. Stejně tak se při pohybu v obraze mění průběh funkce ostrosti při již probíhajícím ostření, takže algoritmus kvůli tomuto pohybu může selhat.

Pokud by ovšem byl k dispozici rychlý motor, při pohybu objektivu by se velice rychle nalezl skutečný vrchol funkce ostrosti odpovídající místu s nejlepším zaostřením kamery. Šum vzniklý pomalejším pohybem objektů ve scéně by nehrál tak významnou roli. Byl by totiž jednoduše rozpoznatelný, protože by měl pomalejší dynamiku vrcholů než vrcholy funkce ostrosti vzniklé přejetím objektivu do pozice správného zaostření. Jinými slovy by se dalo říct, že šum vzniklý pohybem objektů by byl oproti rychlosti motoru natolik pomalý, že by během ostření nebyl zaznamenán jeho růst či klesání. Zároveň by bylo možné nastavit detektor tak, aby reagoval pouze na prudký růst funkce ostrosti, odpovídající rychlému pohybu objektivu. U komerčních fotoaparátů využívajících pasivní ostření je běžné automatické zaostření ve zlomcích sekund, což nejen že klade velké nároky na rychlost zpracování, ale také účinně filtruje pohyb objektů v obraze.

Jak už bylo zmíněno, ovládání krokového motoru probíhalo pouze zasíláním požadavku na otočení se o požadovaný počet pulzů. Bez enkodéru polohy tak nebylo možné zjistit, jestli motor na požadovanou pozici skutečně dorazil. Tato absence skutečného zpětnovazebního řízení motoru se při implementaci ukázala jako zásadní problém. Na sestavě objektivu totiž byla použita poměrně slabá převodovka vykazující vůli při otáčení směru pohybu. Tato skutečnost přinášela nejvíce problémů při návratu objektivu na uloženou pozici s maximální ostrostí. Pokud se motor po detekci růstu pohyboval jedním směrem a bylo detekováno klesání, byl okamžitě ob-

rácen směr pohybu a vyslán příkaz pro návrat na polohu s maximální ostrostí. Několik set pulzů motoru však padlo pouze na umoření vůle v převodovce.



Obrázek 4.21.: Průběh filtrované funkce ostrosti $F_{tenengrad}$ zaostřování bez kompenzace vůle převodovky.

Počet pulzů na umoření vůle navíc nebyl v celém rozsahu zdvihu objektivu konstantní a ani lineární. Jako vhodné řešení se nicméně ukázalo implementovat korekční člen, který při volání příkazu pro posun o požadovaný počet pulzů přičetl hodnotu odpovídající střednímu počtu pulzů potřebných na umoření vůle. Tento přístup se ukázal jako dostatečný a ostření potom probíhalo velmi přesně. Hodnotu počtu pulzů, které kompenzují vůli v převodovce lze nastavit zavoláním knihovní funkce. Nejlepším řešením problémů s vůlí by však samozřejmě bylo použití servomotoru se skutečnou zpětnou vazbou, který by umožňoval přesné polohovaní.

Ukázka průběhu funkce ostrosti při ostření bez kompenzace vůle v převodovce je zobrazena na obrázku 4.21. Okolo šedesátého snímku byl algoritmem vyslán krokovému motoru pokyn k obrácení směru pohybu za účelem návratu objektivu na polohu s maximální mírou ostrosti. Počet pulzů potřebný k návratu však byl umořen na překonání vůle převodovky a objektiv se vůbec nepohnul (viz konstantní hodnota průběhu funkce ostrosti od cca šedesátého snímku).

KAPITOLA

PÁTÁ

SLEDOVÁNÍ OBJEKTU V OBRAZE A DIGITÁLNÍ STABILIZACE OBRAZU

5.1. Požadavky na funkce sledování objektu v obraze a digitální stabilizace obrazu

Jedna z dalších zákazníkem požadovaných funkcí knihovny byla možnost sledovat vybraný objekt v obraze. Uživatel koncové aplikace by v živém obraze zprostředkovaném knihovnou označil kliknutím objekt, který chce sledovat a knihovna by dále automaticky vyhledávala a označovala tento objekt ve snímcích videosignálu. Obrazové souřadnice sledovaného objektu měly být z knihovny odesílány do koncové aplikace zákazníka za účelem např. otáčení montáže kamery za sledovaným objektem.

Rozšířením funkce sledování objektu v obraze měla být tzv. digitální stabilizace obrazu. Hlavním účelem této funkce měla být eliminace vysokofrekvenčních vibrací přenášených na montáž pozorovací kamery od motoru nastartovaného vozidla. Na rozdíl od optické stabilizace, kdy je pro eliminaci pohybu objektivu kamery využíváno gyroskopů k určení vychýlení a úhlové rychlosti, je v případě digitální stabilizace obrazu používána pouze informace obsažená ve snímcích videosignálu. Digitální stabilizace tedy z principu není tak robustní a spolehlivá jako stabilizace optická, nicméně lze ji používat s libovolnou kamerou a objektivem a jedná se také o poměrně levnější řešení.

5.2. Problematika hledání vzoru v obraze

Hledání uživatelem označeného objektu v obraze je založeno na vyhledávání tzv. šablony, která reprezentuje část vybraného objektu, v obraze. Daná šablona je tedy určitým malým výřezem z celkového snímku. Pokud je šablona obsažena ve snímku sekvence nebo videosignálu zachy-

cujícího běžné scény, dá se předpokládat, že se objeví (v lehce pozměněné formě) i v některých dalších snímcích. Úlohou algoritmu vyhledávání objektu v obraze je tedy rozpoznávat naučenou šablonu v dalších snímcích videosignálu.

Nejčastější metodou vyhledávání šablony ve snímku je použití vzájemné korelace šablony a daného snímku[1]. V principu jde o procházení snímku po jeho jednotlivých pixelech, ve kterých se vyhodnocuje míra podobnosti výřezu snímku o velikosti hledané šablony s danou šablonou. Místo s největší podobností odpovídá nejpravděpodobnější pozici objektu reprezentovaného naučenou šablonou ve snímku. Hledání pozice šablony ve snímku je založeno na výpočtu druhé mocniny euklidovské vzdálenosti mezi hodnotami jasů pixelů šablony a snímku:

$$d(i,j) = \sum_{x=0}^{K-1} \sum_{y=0}^{L-1} [w(x,y) - f(x+i,y+j)]^2, \tag{5.1}$$

kde $w(x,y)$ je šablona velikosti $K \times L$ a $f(x,y)$ je snímek o velikosti $M \times N$, v němž je šablona vyhledávána. Indexy x,y označují pozici v šabloně a i,j indexují absolutní pozici ve snímku. Když roznásobíme pravou stranu rovnice (5.1) dostaneme vztah

$$d(i,j) = \sum_{x=0}^{K-1} \sum_{y=0}^{L-1} [w^2(x,y) - 2w(x,y)f(x+i,y+j) + f^2(x+i,y+j)], \tag{5.2}$$

kde člen $\sum_{x=0}^{K-1} \sum_{y=0}^{L-1} w^2(x,y)$ je „energie“ šablony a $\sum_{x=0}^{K-1} \sum_{y=0}^{L-1} f^2(x+i,y+j)$ je „energie“ výřezu snímku o velikosti $K \times L$ (stejné velikosti jako šablona) na pozici (i,j). Vzhledem k tomu, že první uvedený člen je konstantní a za předpokladu, že druhý člen je pro všechny pixely snímku přibližně konstantní, lze rovnici (5.2) pro výpočet podobnosti šablony a výřezu snímku zjednodušit na

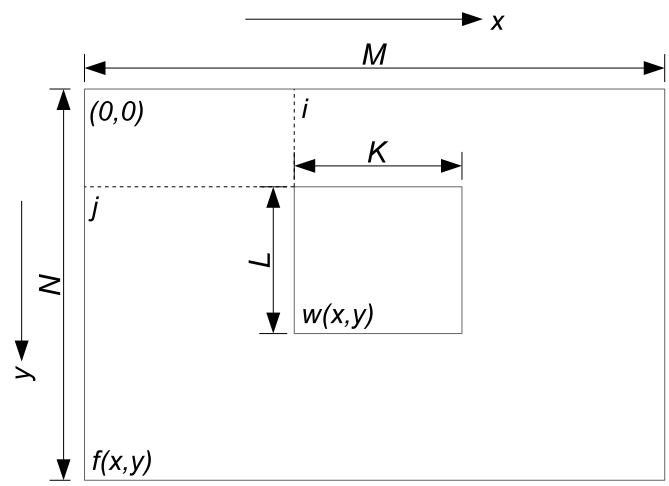
$$c(i,j) = \sum_{x=0}^{K-1} \sum_{y=0}^{L-1} w(x,y)f(x+i,y+j), \tag{5.3}$$

což je vztah vyjadřující vzájemnou korelaci šablony a výřezu snímku na pozici (i,j) [20, 21].

Algoritmus hledání největší podobnosti spočívá v posouvání šablony po snímku. V dané pozici se vynásobí hodnota jasu každého pixelu šablony a snímku a všechny součiny se sečtou. Tam kde funkce $c(i,j)$ nabývá maxima, je největší pravděpodobnost výskytu vyhledávaného objektu (reprezentovaného šablonou) ve snímku. Situace je zachycena na obrázku 5.1.

Metoda výpočtu podobnosti šablony a výřezu snímku pomocí (5.3) má však několik zásadních nevýhod. Vzájemná korelace je závislá na předpokladu, že snímky, v nichž je šablona vyhledávána, obsahují pixely s málo se lišícími hodnotami jasů. Proto korelace šablony s velmi světlými místy snímku může nabývat větších hodnot než korelace se skutečnou nejlepší pozicí

[1] V tomto oddíle bude v případě vzájemné korelace vždy řeč o snímcích v odstínech šedi. Vzájemnou korelaci lze totiž počítat jen s hodnotami jasu těchto snímků. Sledování objektu v barevném snímku je realizováno převodem šablony i snímku do odstínů šedi.



Obrázek 5.1.: Znázornění polohy a souřadnic šablony vyhledávané ve snímku.

šablony. Navíc pokud se prodlouží expozice snímků v sekvenci videosignálu, budou snímky světlejší a magnituda hodnot korelace bude jiná než u snímků, které jsou tmavší. Nebylo by tedy možné správně nastavit práh, kterého musí hodnota korelace dosáhnout, aby byla pozice označena jako lokace šablony nalezené ve snímku.

Z výše uvedených důvodů se pro detekci šablon ve snímku používá normalizovaná vzájemná korelace popsaná vztahem

$$c_n(i,j) = \frac{\sum_{x=0}^{K-1}\sum_{y=0}^{L-1}[w(x,y)-\overline{w})][f(x+i,y+j)-\overline{f}(i,j)]}{\sqrt{\sum_{x=0}^{K-1}\sum_{y=0}^{L-1}[w(x,y)-\overline{w})]^2}\cdot\sqrt{\sum_{x=0}^{K-1}\sum_{y=0}^{L-1}[f(x+i,y+j)-\overline{f}(i,j)]^2}}. \tag{5.4}$$

Členy $\overline{w}$ a $\overline{f}(i,j)$ vyjadřují průměrnou intenzitu jasu v šabloně a ve výřezu snímku o velikosti šablony na aktuální pozici (i,j). Hodnota $c_n(i,j)$ leží v intervalu -1 až 1 a nezávisí na absolutních hodnotách jasů ve snímku ani šabloně. Světlé scény ani změny expozice tedy neovlivní výsledek vyhledávání šablony.

Bohužel ani normalizovaná vzájemná korelace sama o sobě není zcela ideální pro vyhledávání šablony ve snímku. Pokud dojde k pootočení objektu vyhledávaného v obraze nebo ke změně jeho velikosti (přiblížení nebo oddálení od objektivu), nebude možné pomocí normalizované korelace šablonu ve snímku spolehlivě nalézt.

5.3. Vyhledávání šablony v obraze pomocí NI LabVIEW

Vývojové prostředí NI LabVIEW použité při vývoji knihovny nabízí v balíku Vision Development Module širokou paletu funkcí pro zpracování a úpravy obrazu. Patří mezi ně i sofistikované funkce např. pro vyhledávání vzorů, geometrických tvarů, barevných objektů v obraze atd.

I když by implementace vzájemné normalizované korelace v její základní podobě nebyla v NI LabVIEW příliš složitá, nebyl by naprogramovaný algoritmus velmi robustní vůči změnám velikosti hledaného vzoru a jeho rotacím. Pro zlepšení chování algoritmu v těchto situacích by bylo nutné vyvinout další mechanismy jako učení se různých velikostí šablony a jejích rotovaných verzí. I s použitím takových mechanismů by však byl algoritmus podstatně pomalejší než značně optimalizované funkce již dostupné ve Vision Development Module.

Ve vyvíjené knihovně byla proto pro naučení šablony použita funkce *IMAQ Learn Pattern* a pro vyhledávání objektu v obraze funkce *IMAQ Match Pattern*. Zmíněné funkce zastupují z velké míry automatizovaný přístup k vyhledávání šablony v obraze. Zavoláním první funkce se vytvoří sada šablon, která je potom používána druhou funkcí k vyhledávání. Výstupem druhé funkce jsou přímo obrazové souřadnice pozice nalezeného objektu. Funkci *IMAQ Match Pattern* je nutné předem zadat minimální požadované „skóre“ v rozsahu $0 - 1000$, které musí nalezený výsledek splňovat, případně další parametry pro nastavení vyhledávání.

Jádro funkce IMAQ Match Pattern je založené na výpočtu normalizované vzájemné korelace vyhledávané šablony a snímku. Bez dalších vylepšení tato metoda obvykle nalezne vzory o nezměněné velikost a pootočené o maximálně 5–10° oproti šabloně. Rotované a přeškálované vzory však vyžadují speciální přístup: např. při jejich vyhledávání v obraze je nutné šablonu nejprve pootočit o úhel (resp. přeškálovat) a poté provést korelaci s touto novou šablonou. Dokud není šablona ve snímku nalezena, je nutné operaci iteračně opakovat.

Problémem výpočtu normalizované korelace je však jeho vysoká výpočetní náročnost daná vysokým počtem operací násobení ve snímku. I když je použito paralelního zpracovávání, vyhledávat přímo naučenou šablonu včetně všech jejích rotovaných variant by bylo výpočetně neúnosné (obzvláště v reálném čase ve videosignálu). Snížení výpočetní náročnosti lze dosáhnout zmenšením velikosti snímku a šablony. Algoritmus proto při učení se funkcí *IMAQ Learn Pattern* uloží šablony převzorkované na několik menších rozlišení (počet nových podvzorkovaných šablon je určen automaticky, lze však parametrem omezit). Menší rozlišení je vždy čtvrtinové oproti předchozímu a velikosti jednotlivých šablon tak tvoří skokově sestupující funkci. Proto je dále popsaný algoritmus hledání označován jako tzv. pyramidové hledání.

Při vlastním vyhledávání šablony se postupuje od hrubého k přesnému hledání. Nejprve se hledá šablona zmenšená na nejmenší rozlišení v poměrově stejně zmenšeném snímku. Protože došlo k výraznému podvzorkování, výpočet normalizované vzájemné korelace není drasticky výpočetně náročný, ale na druhou stranu ani případné nalezené shody nejsou díky ztrátě detailů zcela přesné. Je však sestavena sada oblastí, ve kterých by se mohli nacházet potenciální kandidáti na hledaný objekt. V další iteraci už se prohledávají jen tyto oblasti, nyní s méně zmenšenými šablonami, a pokračuje se, dokud není výsledek hledaní co nejpřesnější. Je zřejmé, že vyloučením zbytečných oblastí již při prvním kroku hledání uvedený algoritmus výrazně šetří čas.

V případě rotovaných šablon je nemyslitelné vyhledávat všechny varianty pokrývající rozsah rotací $0-360°$. Proto se postupuje podobným způsobem jako v předchozím odstavci. Nejprve se hledají šablony otočené o sadu hrubších úhlů a po nalezení oblastí, kde by mohly ležet hledané objekty, se úhel dále iteračně zmenšuje a provádí se nové zpřesňující hledání.

Pro učení se a vyhledávání snímků nepracuje funkce *IMAQ Match Pattern* s celou šablonou a snímkem, jež často obsahují redundantní obrazovou informaci. Místo toho vnitřní algoritmus funkce inteligentně segmentuje šablonu (a při vyhledávání i snímek) a nepravidelně ji vzorkuje. Je kladen důraz především na hrany a spojité oblasti v šabloně.

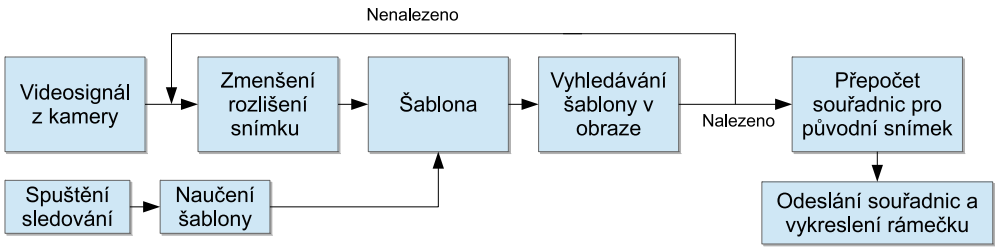
V dokumentaci funkce je udáváno, že algoritmus ve snímku spolehlivě nalezne vzory lišící se velikostí o $\pm 5\%$ a pootočené o $0-360°$ oproti šabloně. Výpočetní náročnost algoritmu silně závisí na omezení vyhledávaní programátorem stanovenými parametry. Důležité je nastavit rozsah úhlů, ve kterém se může pohybovat očekávané pootočení vzoru v obraze. Dále je vhodné omezit počet předpokládaných výskytů vzoru ve snímku a nastavit minimální „skóre“, které nalezený vzor musí získat. Detaily vnitřního principu funkcí *IMAQ Learn Pattern* a *IMAQ Match Pattern* je možné nalézt v [22, 23].

5.4. Implementace vyhledávaní objektu v obraze a digitální stabilizace v knihovně

Protože funkce NI LabVIEW Vision Development Module popsané v předchozím oddílu při testování během vývoje vykazovaly velice spolehlivou detekci šablon v jednotlivých testovacích snímcích, bylo rozhodnuto je implementovat i pro vyhledávání objektů ve videosignálu procházejícím knihovnou.

Sledování objektu v obraze probíhá ve vyvinuté knihovně iterativně v každém příchozím snímku z kamery. Uživatel může pomocí funkcí knihovny provádět nastavení algoritmu vyhledávání. Především je nutné nastavit citlivost algoritmu, která se převádí na minimální požadované „skóre“ vzorů nalezených funkcí *IMAQ Match Pattern* ve snímku. Dále lze nastavit rozlišení, na které se budou snímky videosignálu a učené šablony zmenšovat, čímž dojde k podstatnému snížení výpočetní náročnosti. Velký vliv na dobu strávenou vyhledáváním šablony v jednom snímku má také hledání různě pootočeného objektu. Pokud uživatel dopředu ví, že objekt nebude proti šabloně příliš pootočen, je vhodné omezit rozsah úhlů, s jakými se bude šablona v obraze vyhledávat. Výrazně se tím sníží výpočetní náročnost vyhledávání, která má přímý dopad na snímkovou frekvenci videosignálu procházejícího knihovnou. Detaily funkcí dostupných pro vyhledávání objektu v obraze a nastavení algoritmu jsou v následujícím oddíle.

Blokové schéma algoritmu vyhledávání objektu v obraze je na obrázku 5.2 na následující straně. Po vstupu snímku do algoritmu vyhledávání objektu v obraze je prvním blokem zmenšování příchozích snímku na uživatelem zadané rozlišení. Po spuštění sledování objektu se imple-



Obrázek 5.2.: Blokové schéma algoritmu vyhledávaní objektu v obraze.

mentovaný algoritmus nejprve z aktuálního snímku naučí šablonu odpovídající výřezu snímku na pozici zadané uživatelem (kliknutím do obrazu). Uživatel může zadat jak pozici, tak velikost stran výřezu. Naučená šablona reprezentující vyhledávaný objekt se potom pomocí funkce *IMAQ Match Pattern* vyhledává v následujících snímcích sekvence videosignálu. Výsledkem funkce jsou souřadnice objektu ve zmenšeném snímku, které je nutné v dalším bloku přepočítat na souřadnice původního snímku. Výsledné souřadnice objektu nalezeného v obraze jsou odesílány do koncové aplikace uživatele pomocí Windows zpráv. Po nalezení objektu ve snímku dojde k jeho označení barevným rámečkem[2]. Ukázka vyhledávání objektu ve videosignálu je na obrázku 5.3 na následující straně. Na snímku vlevo nahoře je první snímek, ve kterém byl uživatelem vybrán požadovaný objekt. Šablona reprezentující sledovaný objekt na obrázku uprostřed.

Digitální stabilizace obrazu funguje na podobném principu jako sledování objektu v obraze, je však sledován střed obrazu a na základě jeho vychýlení se vrací obraz zpět na střed okna s videosignálem.

Po spuštění digitální stabilizace se nejprve jako šablona uloží čtvercový výřez středu aktuálního snímku videosignálu o velikosti strany 10 % šířky snímku. Šablona je potom v dalších snímcích videosignálu vyhledávána stejným algoritmem jako při vyhledávání objektu v obraze. Uživatel může nastavit požadované zmenšení rozlišení snímků a šablony pro zmenšení výpočetní náročnosti, stejně tak i předpokládaný úhel, o který se bude scéna při vibracích pootáčet. Při digitální stabilizaci se nevykresluje rámeček kolem nalezeného vzoru, ani se neodesílají souřadnice.

Protože se hledá objekt, který by měl být ve středu snímku, určí se rozdíl souřadnic vzoru nalezeného v aktuálním snímku a středu obrazu. O tento rozdíl se potom posune celý snímek tak, aby nalezený objekt byl opět ve středu snímku. Chybějící část snímku vzniklá vibracemi je doplněna černým okrajem.

Pokud je nalezený vzor ve snímku pootočený, otočí se i výstupní snímek tak, aby pootočení bylo vyrovnáno a chybějící část snímku se opět doplní černým okrajem. Když rotace scény přesáhne uživatelem zadaný maximální předpokládaný úhel rotace, stabilizace přestane být aktivní

[2]Sledování objektu v barevném snímku je realizováno převodem šablony i snímku do odstínů šedi. Vizualizace nalezeného objektu barevným rámečkem je však vykreslována do původního, barevného snímku.

Obrázek 5.3.: Vyhledávání objektu v různých snímcích sekvence videosignálu.

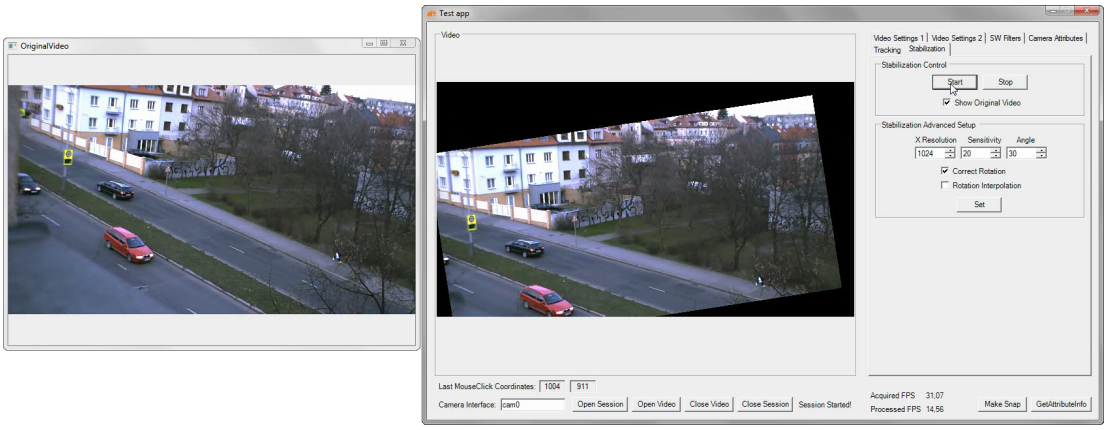
(vzor není ve snímku nalezen) a na výstupu je nestabilizovaný snímek. Protože je stabilizace rotace poměrně dosti výpočetně náročná, je ve výchozím nastavení funkce vypnutá a uživatel ji musí ručně aktivovat. I bez aktivní opravy rotace dokáže algoritmus stabilizovat scény pootočené o zhruba $\pm 5\,^{\circ}$.

Náhled scény, která byla digitálně stabilizována je na obrázcích 5.4 a 5.5 na následující straně. Vlevo je původní nestabilizovaná scéna a vpravo je náhled stabilizovaného snímku videosignálu zobrazovaného v koncové aplikaci používající vyvinutou knihovnu.

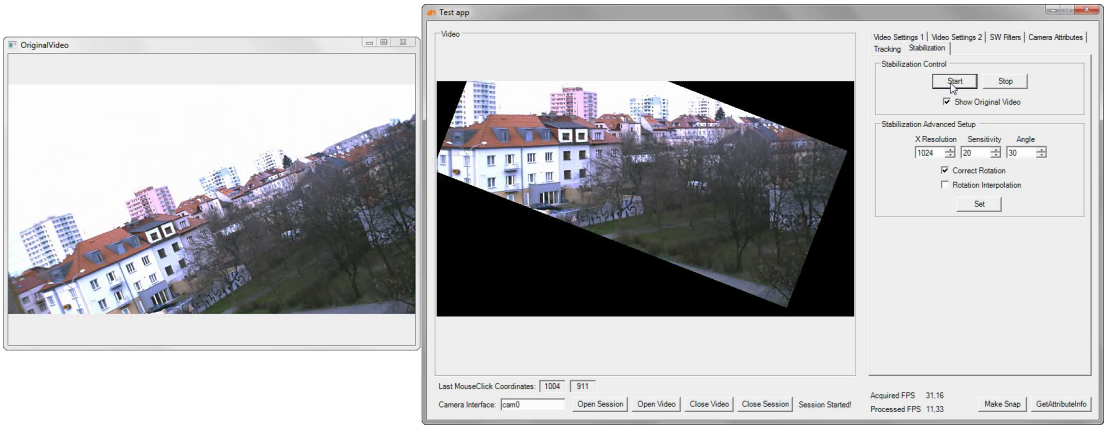
5.5. Praktické výsledky algoritmu vyhledávaní objektu v obrazu a digitální stabilizace

Implementované algoritmy se ukázaly jako vcelku spolehlivá metoda pro základní sledování objektu v obraze. Jejich omezení vyplývají z omezení funkce *IMAQ Match Pattern* a obecně z principu vyhledávání vzoru v obraze pomocí metody vzájemné korelace šablony a částí snímku.

Pro zahájení sledování musí uživatel knihovny ve snímku vybrat oblast reprezentující část sledovaného objektu. Sledování objektu závisí na vyhledávání určené oblasti v dalších snímcích videosignálu. Proto se tato oblast nesmí příliš měnit vlivem pohybu objektu, nesmí pod-



Obrázek 5.4.: Ukázka digitální stabilizace scény v koncové aplikaci. Kladná rotace původního snímku.



Obrázek 5.5.: Ukázka digitální stabilizace scény v koncové aplikaci. Záporná rotace původního snímku.

statně měnit svoji velikost, nesmí být překryta jiným objektem, nesmí obsahovat část pozadí objektu apod.

Změny osvětlení a jasových hodnot snímku nemají na sledování vliv (při zachování rozumného kontrastu a osvětlení objektu). Stejně tak ani rotace sledovaného objektu není problém, pokud se ovšem úhel rotace pohybuje v uživatelem nastavených mezích. Pokud pootočení objektu tyto meze překročí, je sledování přerušeno a začne znovu, pokud se úhel pootočení objektu vrátí do daných mezí. Nastavení velké tolerance pro úhly pootočení vyhledávaného vzoru má přitom zásadní vliv na zvýšení výpočetní náročnosti algoritmu, resp. na snížení snímkové frekvence výstupního videosignálu.

Digitální stabilizace je založena na principu vyhledávání naučeného výřezu ze středu snímku, proto má stejná omezení jako sledování objektu v obraze. Střed snímku, který byl uložen jako referenční poloha, se nesmí pohybovat, měnit svoji velikost, být zakryt jiným objektem, posu-

nout se ven ze snímku atd. Mimo uvedené problémové situace však implementovaná digitální stabilizace funguje až překvapivě dobře. Při pozorování statické scény eliminuje rychlé vibrace s malou amplitudou i větší otřesy kamery a uživateli sledujícímu obraz tak podstatně usnadňuje orientaci v obraze.

Názornou ukázkou funkce sledování objektu v obraze i digitální stabilizace obrazu jsou videonahrávky. Proto je několik nahrávek videosignálu upraveného pomocí implementovaných algoritmů uloženo na DVD přiloženém k této diplomové práci.

5.6. Ovládání sledování objektu a digitální stabilizace z DLL knihovny

Pro spuštění sledování objektu v obraze je nutné kliknout levým tlačítkem myši do okna s obrazem z kamery a vybrat tak část objektu, podle které bude objekt v dalších snímcích sekvence vyhledáván. Událost s informací o tomto kliknutí odešle knihovna koncové aplikaci pomocí Windows zprávy systémovou funkcí

```
SendMessage(int hWnd, 0x9000, int wParam, int lParam).
```

Prvním parametrem `hWnd` je reference na hlavní okno nadřazené aplikace (handle), která knihovnu používá, kód zprávy je pevně zvolen jako 0x9000 a parametry `wParam` a `lParam` odpovídají horizontální a vertikální souřadnici v obraze.

Koncová aplikace po přijmutí Windows zprávy zavolá knihovní funkci

```
void TrackingSetArea(uint16_t x, uint16_t y,
   uint16_t nWidth, uint16_t nHeight),
```

kterou je knihovně zadána oblast, ze které bude vytvořena šablona, jež se bude vyhledávat v dalších snímcích videosignálu. Parametry `x`, `y`, `nWidth` a `nHeight` reprezentují horizontální a vertikální souřadnici středu oblasti a její šířku a výšku (v pixelech). Sledování objektu je poté spuštěno funkcí **`void`** `TrackingON()` a je možné jej kdykoliv zastavit funkcí **`void`** `TrackingOFF()`.

Nastavení rozšířených parametrů algoritmu sledování objektu se provádí funkcí

```
void TrackingSetup(uint16_t xResolution, uint16_t
   sensitivity, uint8_t angle).
```

Parametr `xResolution` nastavuje rozlišení, na které budou snímek a šablona za účelem vyhledávání zmenšovány. Snížení rozlišení šetří výpočetní čas potřebný k prohledání snímku, na druhou stranu přílišné podvzorkování nese riziko ztráty detailů, které mohou být nutné pro úspěšné sledování objektu. Citlivost algoritmu vyhledávání je nastavována parametrem `sensitivity`, který reprezentuje minimální požadované „skóre“, kterého musí vzor nalezený v obraze dosáhnout. Rozsah tohoto parametru je 0–100 a vybraná hodnota je převáděna na „skóre“ 0–1000,

které je v knihovně předáváno funkci *IMAQ Match Pattern*. Parametr `angle` nastavuje předpokládaný maximální úhel, o který může být vzor ve snímcích videosignálu pootočen.

Informace o poloze nalezeného objektu je odesílána koncové aplikaci pomocí Windows zprávy systémovou funkcí

```
SendMessage(int hWnd, 0x9001, int wParam, int lParam).
```

Identifikátor zprávy je pevně zvolen jako 0x9001 a parametry `wParam` a `lParam` odpovídají relativní horizontální a vertikální souřadnici sledovaného objektu (relativní vůči středu obrazu). Pokud se objekt nachází v okně s videosignálem vlevo/vpravo od středu snímku, horizontální souřadnice má záporné/kladné znaménko. Pokud se objekt nachází v okně s videosignálem dole/nahoře od středu snímku v okně s videosignálem, vertikální souřadnice má záporné/kladné znaménko.

Ovládání digitální stabilizace je jednodušší. Stabilizace se aktivuje zavoláním funkce **`void`** `StabON()` a je možné ji zastavit funkcí **`void`** `StabOFF()`. Nastavení digitální stabilizace se provádí funkcí

```
void StabSetup(uint16_t xResolution, uint16_t
   sensitivity, uint8_t angle, uint8_t corrRotation,
   uint8_t interpolate),
```

jejíž první tři parametry jsou stejné jako u funkce pro nastavení sledování objektu. Parametr `corrRotation` aktivuje otáčení výstupního snímku proti úhlu vychýlení kamery a parametr `interpolate` zapíná a vypíná interpolaci výstupního obrazu (při zapnuté interpolaci je otočený výstupní obraz kvalitnější a např. otočené hrany objektů mají plynulejší přechody). Oba poslední parametry nabývají hodnot 1 a 0 pro aktivaci a vypnutí příslušné funkce.

Detaily uvedených funkcí společně s dalšími funkcemi např. pro nastavení barvy a tloušťky čáry obdélníku označujícího nalezený objekt jsou uvedeny v příloze H.

KAPITOLA

ŠESTÁ

FUNKCE PRO ZPRACOVÁNÍ A ÚPRAVY SNÍMKŮ

6.1. Funkce pro úpravy jasu a kontrastu snímku

V knihovně je implementováno rovněž několik funkcí pro základní zpracování obrazu. Jedná se o funkce, které nějakým způsobem změní hodnoty jasu všech třech barevných složek každého snímku videosignálu.

Nejjednodušším příkladem je funkce

```
void SetSwBrightness(int16_t nVal),
```

která k jasovým hodnotám barevných složek přičte hodnotu parametru `nVal`.

Pro úpravu kontrastu snímků byla implementována funkce:

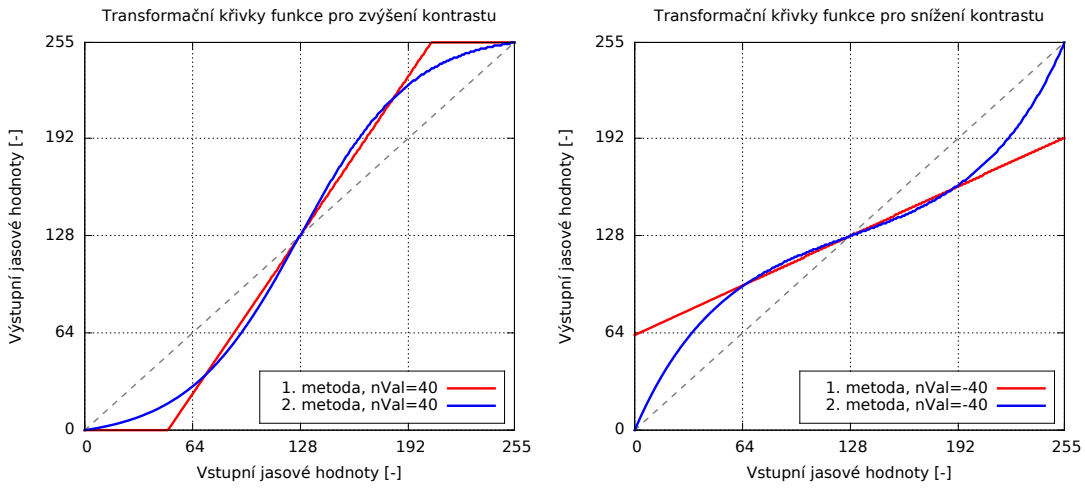
```
void SetSwContrast(uint8_t nMode, int16_t nVal).
```

Parametr `nMode` stanovuje metodu, která se bude při úpravě kontrastu používat.

Transformační křivka operace, kterou funkce provede s jasovými hodnotami snímku v případě použití první metody (`nMode=0`), je vykreslena červenou barvou na obrázku 6.1 na následující straně. Parametr `nVal` nabývá hodnot od -100 do 100 a nastavuje snížení a zvýšení kontrastu snímku. V knihovně je tento rozsah převeden na směrnici transformační přímky. Hodnota 0 odpovídá směrnici $0,5$, hodnota -100 reprezentuje přímku rovnoběžnou s horizontální osou (směrnice je rovna 0) a hodnota 100 reprezentuje přímku rovnoběžnou s vertikální osou.

Z průběhu transformační křivky je zřejmé, že první metoda úpravy kontrastu snižuje dynamický rozsah snímku. Jinými slovy, například při zvýšení kontrastu dochází k saturaci jasových hodnot ve výstupním snímku. Z tohoto důvodu byla implementována druhá, sofistikovanější metoda pro úpravy kontrastu.

Transformační křivka při použití druhé metody (`nMode=1`) má nelineární průběh ve tvaru písmene *S* (někdy je také označována jako tzv. *sigmoid* funkce) [15, 16]. Křivka je zobrazena



Obrázek 6.1.: Transformační křivka funkce pro úpravu kontrastu za použití různých metod.

modře na obrázku 6.1. Parametr `nVal` nabývá hodnot od -100 do 100 a nastavuje snížení a zvýšení kontrastu snímku podobně jako u předchozí metody.

V knihovně jsou obě metody implementovány pomocí vyhledávací transformační tabulky, která obsahuje 256 řádků. Každý řádek odpovídá jedné jasové hodnotě vstupního snímku, kterou hodnota v řádku transformuje do výstupního snímku. Tabulka se vygeneruje se změnou parametru funkce.

V případě první metody slouží pro naplnění tabulky jednoduše trigonometrická funkce, která nejprve vytvoří směrnici pro přímku, jež je dále posunuta ve vertikálním směru. Výpočet je popsán vztahem

$$y(x) = x\tan(\varphi) \tag{6.1}$$

$$z(y) = y - \frac{\max(y) - 255}{2}, \tag{6.2}$$

kde x jsou vstupní jasové hodnoty, $y(x)$ jsou výstupní jasové hodnoty a úhel φ je parametr `nVal` přepočítaný tak, aby nabýval hodnot 0 až $\pi/2$.

Transformační tabulka druhé metody je vygenerována pomocí hyperbolických funkcí vztahem

$$y(x) = \begin{cases} 127,5 \cdot (1 + \tanh(k \cdot (x - 127,5)) & k>0 \\ x & k=0 \\ 127,5 \cdot (1 + \sinh(-k \cdot (x - 127,5)) & k < 0 \end{cases}, \tag{6.3}$$

kde k nastavuje strmost hyperbolické funkce. Parametr `nVal` je v tomto případě přepočítán tak, aby hodnota k nabývala hodnot 0 až $0,05$.

Funkce $y(x)$ je dále ještě normována tak, aby výstupní hodnoty pokrývaly celý rozsah hodnot jasu snímku. Výsledné hodnoty jasů výstupního snímku $z(y)$ jsou pak dány vztahem:

$$z(y) = [y - \min(y)] \cdot \frac{255}{\max(y) - \min(y)}. \tag{6.4}$$

Na obrázku 6.2 je zobrazeno srovnání metod pro úpravu kontrastu obrazu. První fotografie reprezentuje vstupní originální snímek[1]. Na druhém snímku je provedena transformace první metodou pro úpravu kontrastu a třetí snímek je upraven druhou metodou. Na obrázku je zřetelný rozdíl mezi druhým a třetím snímkem, zejména pak ve světlých a tmavých odstínech s jemnou kresbou (kresba na klobouku, odstíny pleti, pravý dolní roh).

Obrázek 6.2.: Srovnání metod funkce pro úpravu kontrastu snímku.

Další funkcí pro úpravu jasu a kontrastu je funkce

```
void SetSwPolynomTransformation(double fConstX3,
   double fConstX2, double fConstX1, double fConstX0),
```

která funguje podobně jako předchozí uvedené s tím rozdílem, že uživatel knihovny může definovat transformační tabulku pomocí polynomiální funkce.

Parametry `fConstX3`, `fConstX2`, `fConstX1` a `fConstX0` zastupují koeficienty polynomiální funkce 3. řádu, kterou budou transformovány vstupní jasové intenzity. Transformační funkce přibližně odpovídající nastavení kontrastu s použitím druhé metody na obrázku 6.1 na předchozí straně vlevo by měla tvar[2]

$$y(x) = -5,056 \cdot 10^{-5} \cdot x^3 + 1,949 \cdot 10^{-2} \cdot x^2 - 0,7517 \cdot x + 9,51. \tag{6.5}$$

Pro všechny výše uvedené funkce byl implementován i jejich protějšek, který vyčte aktuální nastavení daných transformací. Více viz dokumentace knihovny v příloze H.

[1] Záměrně byl zvolen snímek v odstínech šedi, na kterém jde lépe demonstrovat saturaci jasových hodnot. Funkce pro úpravu kontrastu však samozřejmě funguje i na barevných snímcích.

[2] Koeficienty polynomiální funkce lze jednoduše zjistit proložením požadované transformační funkce polynomem 3. řádu například v tabulkovém procesoru.

6.2. Funkce pro úpravu barevnosti snímku

Pro úpravu barvenosti videosignálu byla v knihovně implementována funkce

```
void SetSwColorTransformationValueRGB(uint8_t nMode,
   int16_t nR, int16_t nG, int16_t nB).
```

Parametry `nR`, `nG` a `nB` nastavují hodnotu jasu, která bude přičtena nebo odečtena k červenému, modrému a zelenému barevnému kanálu. Nabývají hodnot -255 až 255.

Výše uvedený režim úpravy barev se nastavuje parametrem `nMode=0`. Protože však zvyšování a snižování hodnot jasů barevných kanálů snímku zvyšuje a snižuje i celkový zdánlivý jas snímku, je možné používat speciální režim filtrace. Parametr `nMode=1` aktivuje úpravu barev se zachováním zdánlivého jasu celého snímku.

V tomto režimu se barvy snímku nejprve upraví stejným způsobem jako v prvním případě, čímž dostaneme tři matice s jasy jednotlivých barevných složek $\mathbf{R}_2$, $\mathbf{G}_2$ a $\mathbf{B}_2$. Po úpravě se však snímek převede na barevný model HSL[3] a spočítají se hodnoty roviny zdánlivého jasu (Luma) $\mathbf{L}_2$ podle vztahu definovaného například v [17] jako

$$\mathbf{L}_2 := 0.299 \cdot \mathbf{R}_2 + 0.587 \cdot \mathbf{G}_2 + 0.114 \cdot \mathbf{B}_2. \tag{6.6}$$

Stejným způsobem se spočítá hodnota zdánlivého jasu netransformovaného snímku jako

$$\mathbf{L}_1 := 0.299 \cdot \mathbf{R}_1 + 0.587 \cdot \mathbf{G}_1 + 0.114 \cdot \mathbf{B}_1, \tag{6.7}$$

kde $\mathbf{R}_1$, $\mathbf{G}_1$ a $\mathbf{B}_1$ jsou hodnoty jasů červené, zelené a modré složky původního snímku. Následně se vyjádří poměr obou složek zdánlivého jasu a tímto poměrem se vynásobí barevné složky transformovaného snímku, čímž vznikne finální upravený snímek [18]:

$$\mathbf{R}_3 = \frac{\mathbf{L}_1}{\mathbf{L}_2} \cdot \mathbf{R}_2 \tag{6.8}$$

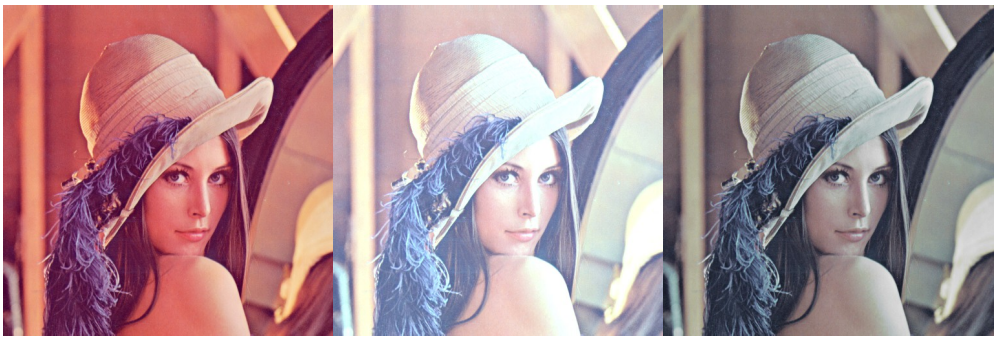
$$\mathbf{G}_3 = \frac{\mathbf{L}_1}{\mathbf{L}_2} \cdot \mathbf{G}_2 \tag{6.9}$$

$$\mathbf{B}_3 = \frac{\mathbf{L}_1}{\mathbf{L}_2} \cdot \mathbf{B}_2. \tag{6.10}$$

Na obrázku 6.3 je zobrazeno srovnání jednotlivých metod pro úpravu barevnosti snímku. Vlevo je originální, neupravený snímek, na druhý snímek byla použita filtrace první metodou a na třetí snímek byla použita druhá metoda úpravy barevnosti. Při transformaci byly k červenému, zelenému a modrém kanálu přičteny hodnoty $\begin{bmatrix} 20 & 100 & 100 \end{bmatrix}$. Je zřejmé, že druhý snímek

[3] HSL je alternativní barevný model sestavený v roce 1978. Tento model nejvíce odpovídá lidskému vnímání barev. Skládá se ze tří barevných složek: Hue – barevný tón, Saturation – sytost barvy a Luma – zdánlivý jas.

je výrazně zesvětlen a v některých místech jsou hodnoty jasu již saturovány. Oproti tomu třetí snímek má zdánlivý jas stejný jako originální snímek.

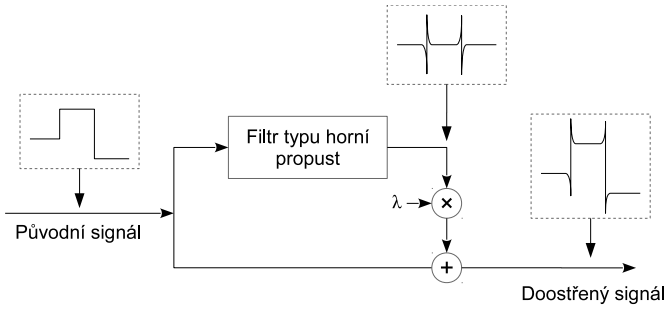


Obrázek 6.3.: Srovnání metod funkce pro úpravu barevnosti snímku.

6.3. Funkce pro softwarové doostření snímku

Lidský zrak je citlivý na ostré a kontrastní detaily, které jsou ve snímku vesměs tvořeny vysokými prostorovými frekvencemi. Zvýraznění vysokých frekvencí neboli doostřování přinese divákovi vylepšení vizuální kvality snímku. Proto byla i pro vyvinutou knihovnu implementována funkce pro softwarové doostření snímku.

Schéma obecného algoritmu doostřování je ilustrováno na jednorozměrném signálu na obrázku 6.4. Z původního snímku (signálu) jsou nejprve filtrem typu horní propust extrahovány vysokofrekvenční komponenty, které jsou po přeškálování opět přičteny k původnímu snímku. Součet tvoří výstupní, doostřený snímek. Oblasti, kde se intenzita jasu skokově neměnila zůstávají procesem nedotčeny.



Obrázek 6.4.: Schéma obecného algoritmu doostřování jednorozměrném signálu.

Vy vyvinuté knihovně byl pro doostřování videosignálu implementován algoritmus, který k získání vysokofrekvenční informace snímku využívá konvoluci snímku s jádrem realizujícím

filtr typu horní propust. Konvoluce je popsána vztahem

$$j(x,y) = \mathbf{K} * i(x,y) \tag{6.11}$$

$$\mathbf{K} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}, \tag{6.12}$$

kde $j(x,y)$ je výstupní snímek obsahující pouze vysokofrekvenční informaci, $i(x,y)$ je původní snímek a $\mathbf{K}$ je konvoluční jádro. Výhoda konvoluce s tímto jádrem spočívá v tom, že využívá i okolních pixelů a tím filtruje šum v obraze (který by ideálně neměl byl doostřen vůbec) [7, 19]. Výsledný doostřený snímek[4] je dán jako

$$k(x,y) = i(x,y) + \lambda\, j(x,y), \tag{6.13}$$

kde $k(x,y)$ je výstupní doostřený snímek a λ je koeficient, kterým je škálován snímek $j(x,y)$ obsahující vysokofrekvenční informaci (viz také schéma na obrázku 6.4). Ukázka softwarového doostřování implementovaného v knihovně je na obrázku 6.5. Vlevo je zobrazen vstupní, záměrně lehce rozostřený snímek. Uprostřed je snímek obsahující vysokofrekvenční informaci z červené barevné složky původního snímku (zobrazeno inverzně, tmavé body reprezentují vyšší jasové hodnoty). A konečně vpravo je výsledný doostřený snímek vzniklý přičtením filtrovaných barevných složek ke složkám snímku původního.

Obrázek 6.5.: Demonstrace funkce doostřování snímku.

Doostřování videosignálu je v knihovně aktivováno funkcí

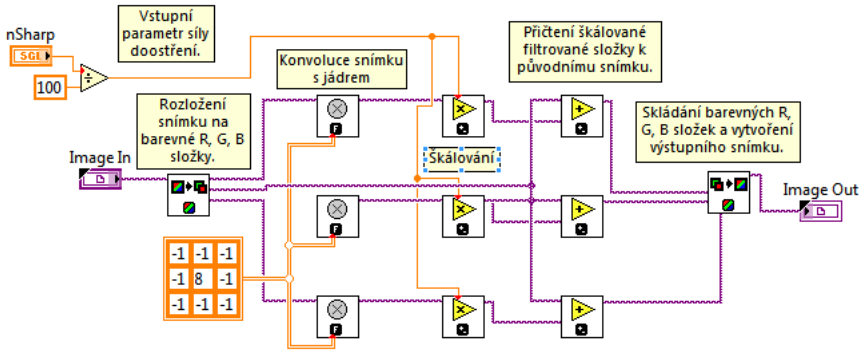
```
void SetSwSharpness(int8_t nSharp),
```

jejíž parametr `nSharp` v rozsahu $0-100$ nastavuje sílu doostření.

Na obrázku 6.6 je zobrazen zjednodušený zdrojový kód s implementací funkce softwarového doostřování snímku vytvořený v NI LabVIEW. Je zřejmé, že snímek se nejprve musí rozdělit

[4] Pro jednoduchost výkladu je uvažován snímek v odstínech šedi. U barevných snímků se stejným způsobem transformují všechny barevné jasové složky.

na barevné složky, jež se filtrují pomocí konvoluce s výše uvedeným jádrem. Filtrované barevné složky se potom přičtou k původním barevným složkám a složí se do výstupního snímku.



Obrázek 6.6.: Ukázka zdrojového kódu s implementací funkce softwarového doostřování snímku.

KAPITOLA

SEDMÁ

UKLÁDÁNÍ SNÍMKŮ A ZÁZNAM VIDEOSIGNÁLU

7.1. Ukládání snímků videosignálu

V knihovně byla za účelem ukládání nezměněných snímků videosignálu z kamery implementována funkce

```
int8_t GetRawImagePointer(uint64_t *PixelPointerOut,
   int32_t *BufferSize, int32_t *PixelSize, int32_t
   *LineWidth, int32_t *XResolution, int32_t
   *YResolution),
```

která je založena na funkci *IMAQ GetImagePixelPtr* z NI Vision Development Module. Pomocí funkce lze zjistit ukazatel na blok paměti (pointer), kde je uložen aktuální nezpracovaný snímek z kamery. Jednorozměrné pole bytů v paměti obsahuje čistá obrazová data snímku. Formát dat odpovídá datovému typu obrazu.

Pro rekonstrukci obrazu je nutné použít další návratové parametry funkce. První řádek snímku získáme z operační paměti takto: začneme na adrese dané parametrem `PixelPointerOut` (pointer na první pixel snímku v paměti, ve snímku vlevo nahoře) a pokračujeme po blocích odpovídajících velikosti `PixelSize` (počet bytů obsahujících hodnotu jasu jednoho pixelu) až do `XResolution` (horizontální rozlišení snímku, odpovídá poslednímu pixelu prvního řádku). První pixel druhého řádku je v paměti uložen na adrese `PixelPointerOut + LineWidth` a opět můžeme postupovat po blocích velikosti `PixelSize`. Takto postupujeme po řádcích až do indexu `YResolution` (vertikální rozlišení snímku = index posledního řádku). Návratový parametr `BufferSize` udává celkový počet bytů snímku v paměti.

Snímek v odstínech šedi má osmibitovou hodnotu jasu každého pixelu uloženou v jednom bytu. Každý byte pole z paměti tedy odpovídá jednomu pixelu. Barevný snímek typu má 32bitovou hodnotu reprezentující tři barvy každého pixelu uloženou ve čtyřech po sobě jdoucích

bytech pole z paměti. První, druhý a třetí byte odpovídá červené, zelené a modré složce pixelu a čtvrtý byte je nulový.

Analogií funkce `GetRawImagePointer` je funkce `GetOutImagePointer` se stejnými parametry. Druhá funkce slouží k vyčtení obrazových dat snímku, který byl upraven všemi aktivními transformačními funkcemi knihovny (stabilizace, softwarové filtry, vyznačení rámečku kolem sledovaného objektu apod.).

7.2. Záznam videosignálu

Jedním z dalších požadavků na funkce vyvíjené knihovny byl i záznam videosignálu do některého z běžně rozšířených formátu při zachování co největšího rozlišení a snímkové frekvence sekvence. Balík NI Vision Development použitý k vývoji knihovny obsahuje i funkce pro vytváření videonahrávek ve formátu AVI. Aby bylo možné zaznamenávat videosignál procházející knihovnou, byl vyvinut algoritmus, který umožňuje nahrávat sekvence v maximální možné snímkové frekvenci (omezené výpočetním výkonem a hardwarem PC - např. rychlostí pevného disku).

Pro zápis jednoho snímku do AVI souboru je v NI Vision Development Module dostupná funkce *IMAQ AVI Write Frame*. Dalšími funkcemi je možné nastavit použitý kodek, míru komprese videosignálu, omezení snímkové frekvence apod. Základní algoritmus pro záznam videosignálu by každý aktuální snímek příchozí z kamery předal funkci *IMAQ AVI Write Frame* a po jeho konverzi a uložení by se pokračovalo další iterací smyčky pro sběr a zobrazování snímků z kamery. Tento postup je však použitelný pouze při menších rozlišeních videosignálu, protože konverze a ukládání snímku je náročné na výpočetní čas. Při rozlišeních od 640×480 pixelů výše znatelně klesá počet snímků zobrazovaných za vteřinu (snímková frekvence).

Z důvodu zachování maximální možné snímkové frekvence byl v knihovně implementován algoritmus, který při nahrávání sekvence snižuje čas strávený v jedné iteraci smyčky pro sběr a zobrazování snímku z kamery. Algoritmus sestává ze dvou bloků běžících ve dvou paralelních vláknech procesoru.

Po spuštění nahrávání ukládá první blok, označený jako rekordér, snímky videosignálu do dočasného adresáře na pevném disku počítače v binárním, nekomprimovaném formátu. Protože se data snímku nijak nekonvertují a pouze se kopírují z operační paměti na disk počítače, je minimalizován čas strávený jejich ukládáním. Smyčka pro sběr a zobrazování snímků tak není nikde zdržována a prakticky nedochází k poklesu snímkové frekvence[1]. Na pevném disku počítače se tak tvoří jakýsi zásobník, obsahující snímky sekvence v nekomprimovaném formátu, které posléze zpracovává druhý blok. Po sobě jdoucí snímky sekvence jsou ukládány do binár-

[1] Jediným omezením je maximální rychlost zápisu na pevný disk použitý v PC, na kterém je spuštěna knihovna. Pokud je rozlišení videosignálu tak velké, že pevný disk počítače nestihne v jedné iteraci smyčky zapsat objemná data snímku, dojde k poklesu snímkové frekvence.

ních souborů, jejichž velikost může uživatel nastavit jednou z funkcí knihovny. Po překročení zadané velikosti prvního souboru je vytvořen druhý soubor atd.

Druhý blok, označený jako „enkodér“, postupně čte nekomprimované snímky sekvence uložené v zásobníku tvořeném soubory uloženými v dočasném adresáři. Tyto snímky konvertuje do AVI formátu pomocí funkce *IMAQ AVI Write Frame* s použitím uživatelem zvoleného kodeku a dalších nastavení. Jakmile jsou převedeny všechny snímky z jednoho souboru, je tento soubor vymazán a pokračuje se čtením snímků z dalšího souboru. Sada souborů obsahujících snímky je tedy postupně zmenšována a po zkonvertování celé sekvence nezbude v dočasném adresáři žádný soubor. Pokud na pevném disku zůstávají soubory s nekomprimovanými snímky i po skončení nahrávání (zastavení rekordéru), enkodér pracuje dokud nejsou všechny snímky sekvence zpracovány. Protože je výpočetně náročný převod snímků do sekvence ve formátu AVI prováděn ve vlákně odděleném od procesu záznamu a zobrazování snímků, není konverzí ovlivněna snímková frekvence videosignálu.

Výše uvedeným postupem je možné dosáhnout nahrávání videa ve vysokém rozlišení při zachování plné snímkové frekvence kamery. S rozlišením a snímkovou frekvencí ovšem roste i objem dat zapisovaných konvertorem na pevný disk a stejně tak i čas strávený převodem snímků do AVI sekvence blokem enkodéru. Protože enkodér konvertuje snímky pomaleji než jsou zapisovány rekordérem, zásobník souborů obsahující nekomprimované snímky postupně roste a může dojít až k zaplnění pevného disku počítače.

S kapacitou pevného disku souvisí i maximální rychlost zápisu disku, která může rekordér při nahrávání nekomprimovaných dat omezovat a snižovat tak snímkovou frekvenci videosignálu (FPS). Je nutné počítat s tím, že při rozlišení videosignálu 800×600 a 30 FPS se na disk zapisuje rychlostí cca 41 MB/s, při rozlišení 1280×1024 a 30 FPS cca 113 MB/s a při 1920×1080 a 30 FPS cca 180 MB/s. Rychlost zápisu pevného disku je přitom u klasického pevného disku s rotačními plotnami cca $50 - 100$ MB/s a u SSD disku cca $200 - 300$ MB/s. Během vývoje knihovny bylo otestováno, že pokud je použit dostatečně rychlý SSD disk, je možné bez problémů nahrávat video v rozlišení 1920×1080 při 30 FPS.

Nahrávání sekvence je v knihovně zahájeno zavoláním funkce

```
void RecRecordStart(char directory[], char
    filename[]),
```

kde parametry `directory[]` a `filename[]` jsou řetězce obsahující cestu k adresáři a název AVI souboru. Pokud jsou řetězce prázdné, videonahrávky se ukládají do výchozí cesty „`C:\`“ a jsou pojmenovány podle vzoru „`dd.m.rrrr_hh-mm-ss.avi`“ (den, měsíc, rok, hodina, minuta a sekunda uložení nahrávky). Nahrávání je ukončeno po zavolání funkce **`void`** `RecRecordStop()`.

Důležitou funkcí pro nastavení nahrávání je funkce

```
void RecSetEncoderSettings(uint16_t xResolution,
   uint16_t yResolution, uint16_t FPSLimit, char
   codec[]).
```

Parametry `xResolution`, `yResolution` a `FPSLimit` nastavují výstupní rozlišení videonahrávky a její snímkovou frekvenci. Řetězec `codec[]` obsahuje textový identifikátor kodeku, který bude při konverzi nekomprimovaných dat použit. Seznam kodeků instalovaných na počítači, kde je knihovna spuštěna, je možné získat funkcí `RecGetCodecs` (detaily této a ostatních funkcí pro nahrávání videosignálu jsou uvedeny v příloze H). Nahrávání v knihovně bylo testováno s běžně dostupnými kodeky, např. s volně dostupným kodekem Xvid[2]. Protože knihovna samotná neobsahuje kromě okna s videosignálem žádné jiné grafické rozhraní, je nutné použitý kodek nastavovat v některé z aplikací třetích stran (např. v aplikaci VirtualDub[3]).

[2] Kodek Xvid je dostupný volně ke stažení na `http://www.xvid.org/`.

[3] Aplikace VirtualDub pro úpravy videonahrávek je dostupná volně ke stažení na `http://www.virtualdub.org/`.

KAPITOLA

OSMÁ

PODPORA TERMOGRAFICKÝCH KAMER

Přestože byla knihovna původně vyvíjena pro práci s barevnými pozorovacími kamerami, v průběhu vývoje byla implementována i podpora termografických kamer.

Termografické kamery snímají infračervené záření dopadající na detektor kamery a měří tak povrchovou teplotu objektů ve snímané scéně. Snímek z kamery (tzv. termogram) je uložen jako dvourozměrné pole, jehož každý prvek obsahuje informaci o teplotě v daném pixelu. Termografické kamery s rozhraním GigE Vision odesílají snímky jako videosignál naprosto stejně jako klasické barevné kamery. Nastavení kamer se provádí shodně jako u barevných kamer přes XML atributy (viz oddíl 3.2 na straně 12). Liší se jen reprezentace dat snímků videosignálu. Teploty v jednotlivých pixelech jsou ukládány každým výrobcem kamery různě (často je hodnota odpovídající teplotě v každém pixelu uložena v 16 bitech). Teplotní data lze v zobrazovacím software vizualizovat například pomocí odstínů šedi, případně i barevnou paletou, kdy se snímaný rozsah teplot převede na odstíny barev v použité paletě.

8.1. Konverze teplotních dat z termografické kamery

Připojení k termografické kameře a snímání videosignálu obsahujícího teplotní data probíhá stejně jako v případě barevných kamer. Standard GigE Vision však neumožňuje reprezentovat hodnoty pixelů ve snímku desetinným číslem (detaily viz definice platných formátů pixelu v GigE Vision standardu v [24]), a proto jsou teploty ve snímku uloženy ve formě dvourozměrného pole celočíselných hodnot, ze kterých je nutné pomocí vzorců dodaných výrobcem kamery vypočítat konkrétní teplotu daného pixelu.

V knihovně byly implementovány dvě metody převodu teplotních dat z termografické kamery. První z nich je automatický převod dat z kamer výrobce FLIR Systems, Inc. Podporované jsou

GigE Vision termografické kamery typu Ax5 a Ax15[1]. Například výpočet teploty T v kelvinech v jednom pixelu kamer typu Ax5 je popsán vztahem

$$T = \frac{B}{\ln(\frac{R}{S-O} + F)}, \tag{8.1}$$

přičemž hodnoty proměnných B, R, S, O a F jsou před konverzí vyčteny z odpovídajících XML atributů kamery [25].

Druhou metodou pro konverzi teplotních dat je převod pomocí tzv. M skriptu. Jedná se o textový soubor psaný programovacím jazykem MATLAB®, ve kterém je uživatelem zapsána funkce pro výpočet teplot z dat příchozích z termografické kamery. Aby bylo možné data převádět je nutné v M skriptu definovat funkci s názvem „*thermo_conversion*" a soubor se skriptem se musí jmenovat *„thermo_conversion.m*".

Níže je uveden příklad M skriptu, který převede teplotní data z kamery tak, že dvourozměrné pole (matici) dat z kamery `in` vynásobí číslem 10 a výsledek uloží do výstupní proměnné `out` :

```
function out = thermo_conversion(in) out = in*10;.
```

Pro nastavení způsobu konverze dat je nutné zavolat knihovní funkci

```
int8_t SetConversionMethod(int32_t conversionMethod),
```

jejíž parametr `conversionMethod` je číselný identifikátor metody konverze. Návratová hodnota funkce indikuje, zda nastavení konverze proběhlo úspěšně.

V knihovně je možné funkcí

```
void SetThermoPalette(int32_t palette)
```

nastavit barevnou paletu, pomocí které budou vizualizovány teploty v termografickém videosignálu. Parametr `palette` je číselný identifikátor požadované palety. Ukázka termografického snímku zobrazeného pomocí různých barevných palet, které jsou dostupné v knihovně, je na obrázku 8.1.

8.2. Funkce pro měření teplot v termografickém snímku

Pro přístup k aktuální teplotní matici snímků byla v knihovně implementována funkce

```
int8_t GetThermoImagePointer(uint64_t
    *PixelPointerOut, int32_t *BufferSize, int32_t
    *PixelSize, int32_t *LineWidth, int32_t
    *XResolution, int32_t *YResolution),
```

[1] Specifikace podporovaných termografických kamer viz `http://www.flir.com/cs/emea/en/view/?id=56341` a `http://www.flir.com/cs/emea/en/view/?id=41330`.

kterou lze získat ukazatel na adresu v paměti, kde je uloženo dvourozměrné pole teplotních dat datového typu **double**. Bližší význam dalších návratových hodnot je stejný jako u funkcí popsaných v oddíle 7.1 na straně 57.

Obrázek 8.1.: Barevné teplotní palety dostupné v knihovně.

Aby uživatel knihovny nemusel za účelem měření teplot vyčítat celou teplotní matici snímku, byly v knihovně implementovány funkce pro pokročilá měření teplot ve videosignálu. Po správném nastavení převodu termografických dat je možné používat knihovní funkce pro měření teplot a teplotních statistik přímo v okně s videosignálem z termografické kamery.

Nejprve je nutné v okně s obrazem zadefinovat oblast zájmu, tzv. ROI (Region of Interest), ve které se bude provádět měření. Zadávání a vykreslování ROI do obrazu bylo implementováno pomocí nativních nástrojů grafického terminálu, který zobrazuje videosignál na Front Panelu hlavního VI knihovny. ROI je možné ve snímku definovat buď kreslením myší do okna s obrazem, nebo programově, zadáním pole hraničních bodů pomocí funkce

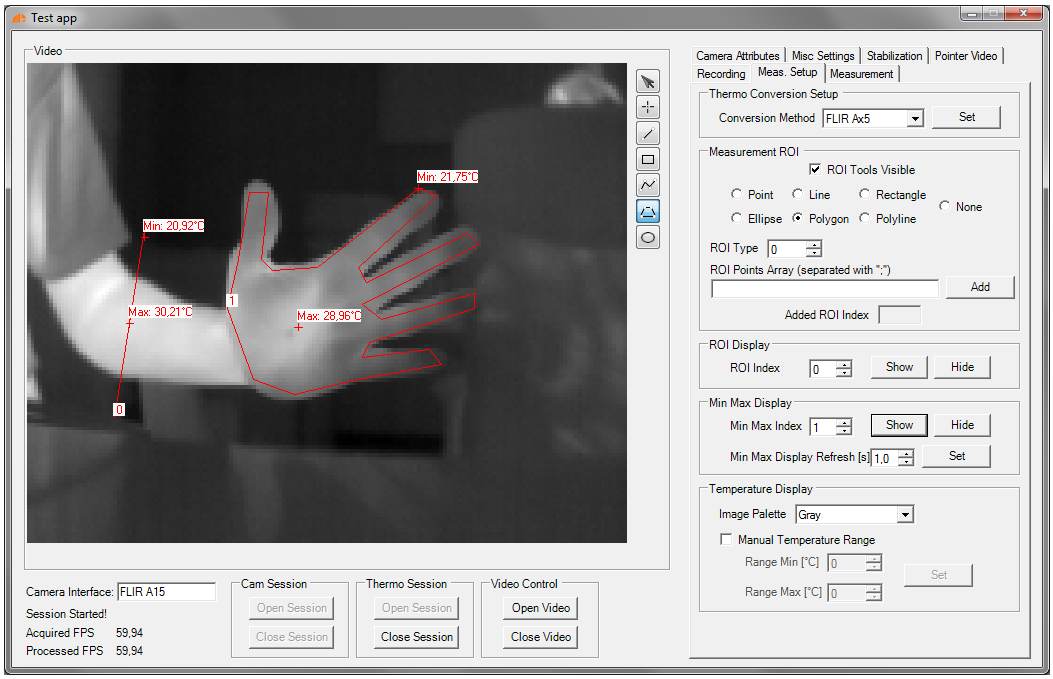
```
int8_t SetMeasROI(int32_t ROIType, int32_t points[],
    int32_t len).
```

Parametrem `points[]` se zadává pole hraničních bodů oblasti a parametr `ROIType` definuje, o jaký typ oblasti se bude jednat. K měření jsou dostupné následující typy ROI oblasti:

- bod,
- úsečka,
- obdélník,
- elipsa,

- polygon,
- lomená čára.

Na obrázku 8.2 je zobrazena koncová testovací aplikace, ve které byly pomocí kreslících nástrojů vykresleny měřící oblasti do okna s videosignálem. Na obrázku stojí za povšimnutí grafické rozhraní aplikace využívající všechny funkce knihovny pro práci s termografickými kamerami: nastavení konverze teplotních dat, přepínaní barevné teplotní palety, výběr a zadávání měřícího nástroje, nastavení zobrazování teplot v ROI atd.



Obrázek 8.2.: Koncová aplikace s měřícími oblastmi vykreslenými do videosignálu z termografické kamery.

Po zadefinování měřící oblasti je jí automaticky přiřazen pořadový identifikátor, který se v obraze zobrazuje na jejím okraji. Pomocí identifikátoru je poté možné z konkrétních ROI měřit funkcemi knihovny teplotní statistiky jako např. minimum, maximum, průměr, rozptyl nebo směrodatná odchylka teplot v oblasti. Případně je možné získat kompletní vektor teplot (jednorozměrné pole) pouze z vybrané oblasti. V každé oblasti je možné aktivovat vykreslování aktuální minimální a maximální teploty. Detaily zmíněných funkcí je možné nalézt v dokumentaci knihovny v příloze H.

KAPITOLA

DEVÁTÁ

LICENCOVÁNÍ A DISTRIBUCE KNIHOVNY

9.1. Implementace licenčního systému

Aby bylo možné knihovnu prodávat po jednotlivých kopiích, byl vyvinut licenční systém zajišťující ochranu proti zneužití zakoupené licence a zároveň umožňující knihovnu používat v bezplatné, zkušební verzi. Licenční systém byl kromě aplikací na straně serveru vyvinut společně s knihovnou v rámci práce na této diplomové práci.

Aplikace vytvořené pomocí softwarového balíku NI Vision Development Module vyžadují ke spuštění a běhu tzv. běhové prostředí (angl. runtime) NI Vision Run-Time Engine. Proto i vyvinutá knihovna musí být distribuována s tímto běhovým prostředím. NI Vision Run-Time Engine však není bezplatný software a pro jeho používání je nutné si zakoupit licenci od společnosti National Instruments, případně je možné používat bezplatnou, zkušební verzi po dobu sedmi dní. V případě zakoupení licence pro běh knihovny je společností Workswell, s. r. o. (dále jen „společnost“) zákazníkovi přiděleno licenční číslo pro NI Vision Run-Time Engine, které je používáno nejen pro aktivaci běhového prostředí, ale i samotné knihovny.

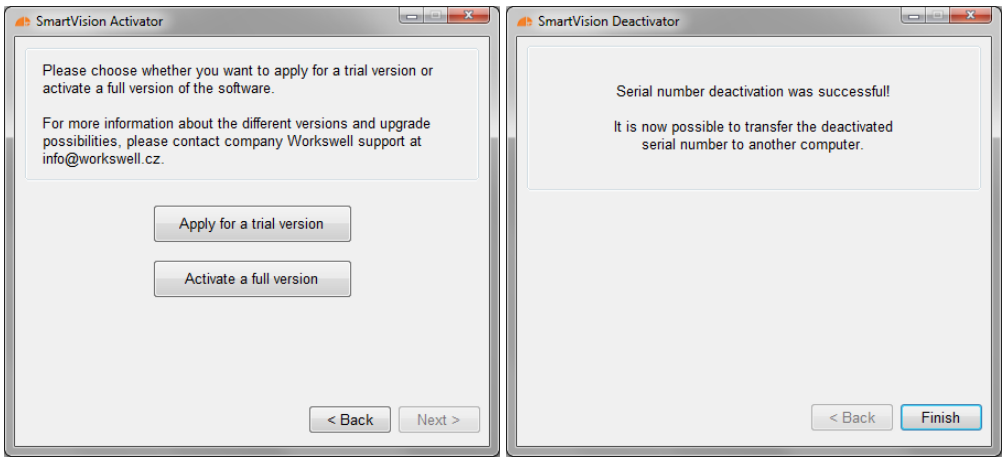
Kolegy ze společnosti byla vyvinuta webová databáze shromažďující licenční čísla a zajišťující správu aktivací knihovny. Po instalaci knihovny je automaticky spuštěn software SmartVision Activator, který byl vyvinut společně s knihovnou za účelem aktivace licencí pro její spuštění. Tento software aktivuje licenci buď pro plnou nebo pro zkušební verzi knihovny. Bez úspěšné aktivace jedné z verzí nelze knihovnu spustit. Ukázka obrazovky aplikace SmartVision Activator je na obrázku 9.1 na následující straně vlevo.

9.1.1. Zkušební verze knihovny

Pro aktivaci zkušební verze knihovny musí uživatel v aplikaci SmartVision Activator aktivovat žádost o zkušební verzi a vyplnit svoji e-mailovou adresu. Aplikace vygeneruje řetězec unikátně identifikující počítač, na kterém je software spuštěn. Pro vytvoření tohoto řetězce byla v aplikaci

implementována funkce, která přes příkazovou řádku spustí jednoduchý program od společnosti National Instruments. Tento program je součástí software NI License Manager a slouží ke generování jedinečného řetězce založeného na MAC adrese síťového adaptéru a sériovém čísle pevného disku daného počítače [26].

E-mailová adresa zákazníka společně s identifikačním řetězcem jsou zašifrovány algoritmem AES[1] a pomocí HTTP protokolu odeslána příkazem POST na webovou stránku společnosti. Stránka údaje zpracuje a zapíše do databáze. Uživateli je na e-mailovou adresu odeslán e-mail s odkazem, který vede na jinou webovou stránku, kde je nutné potvrdit aktivaci zkušební verze. Až po potvrzení aktivace je v databázi k zákazníkovu identifikačnímu řetězci a e-mailu uložen příznak aktivní zkušební verze knihovny.



Obrázek 9.1.: Ukázky obrazovek aplikací SmartVision Activator a SmartVision Deactivator.

Aplikace SmartVision Activator mezitím na daném počítači vygeneruje záznam o tom, že knihovna byla zaregistrována pro spuštění ve zkušební verzi[2]. Při spuštění knihovna vždy nejprve kontaktuje server společnosti (opět pomocí HTTP protokolu a příkazu POST) a odešle identifikační řetězec a dotaz, jestli lze software spustit. Server zkontroluje, jestli pro daný počítač ještě nevypršela zkušební lhůta a v odpovědi knihovně povolí nebo zakáže spuštění. Zkušební verzi je možné kdykoliv po zakoupení licence převést na plnou verzi.

9.1.2. Plná verze knihovny

Průběh aktivace plné verze je do jisté míry podobný aktivaci verze zkušební. Uživatel v aplikaci SmartVision Activator zvolí aktivaci plné verze, vyplní kontaktní údaje a sériové licenční číslo. Toto sériové číslo reprezentuje licenci pro spuštění knihovny a je nutné jej zakoupit od společnosti. Při prodeji zákazníkovi je toto číslo uloženo v databázi na serveru společnosti.

[1] Advanced Encryption Standard (AES) – symetrická bloková šifra. Pro šifrování i dešifrování využívá stejný klíč.
[2] Detaily generování záznamu nejsou záměrně uváděny.

SmartVision Activator vygeneruje řetězec unikátně identifikující počítač, na kterém je software spuštěn. Kontaktní údaje, sériové číslo a identifikační řetězec jsou zašifrovány a pomocí HTTP protokolu (příkazem POST) odeslány na server společnosti. Na serveru je v databázi zkontrolováno, zda je sériové číslo platné a pomocí identifikačního řetězce je rozhodnuto, jestli už nebylo použito pro aktivaci knihovny na jiném PC.

Výsledek kontroly je odeslán odpovědí na POST příkaz zpět aplikaci SmartVision Activator. Pokud je sériové číslo platné, aplikace přes rozhraní příkazového řádku aktivuje pomocí programu NI License Manager licenci pro běhové prostředí NI Vision Run-Time Engine. Dále je na počítači, podobně jako u zkušební verze, vygenerován záznam o tom, že knihovna byla zaregistrována v plné verzi.

Při spuštění knihovny nemusí být počítač připojen k internetu, pouze se pomocí aktivačního záznamu v počítači zkontroluje, jestli je aktivovaná plná licence platná. Pokud ano, knihovna je spuštěna, v opačném případě se knihovna nespustí.

Pokud je nutné zakoupenou plnou verzi knihovny přenést na jiný počítač, je nutné licenci na prvním počítači nejprve deaktivovat pomocí aplikace SmartVision Deactivator. Tato aplikace kontaktuje server společnosti, kde je v databázi vymazán uložený identifikační řetězec prvního počítače. Na druhém počítači je pak možné provést standardní novou aktivaci. Ukázka obrazovky aplikace SmartVision Deactivator je na obrázku 9.1 vpravo.

9.2. Distribuce knihovny a testovací koncová aplikace

Knihovna byla sestavena jako DLL pomocí nástroje NI LabVIEW Application Builder. V tomto nástroji byl vytvořen i spustitelný instalátor, který knihovnu nainstaluje do uživatelem vybrané cesty. S knihovnou je nainstalován i všechen podpůrný software (Camera Explorer, SmartVision Activator, SmartVision Deactivator). Jako samostatný soubor se s knihovnou distribuuje i instalátor ovladačů a běhových prostředí, která jsou potřebná pro připojení se k podporovaným digitálním kamerám a spouštění aplikací vyvinutých v programovacím prostředí NI LabVIEW.

Knihovnu provází rozsáhlý uživatelský manuál, obsahující obecné informace o knihovně a kompletní dokumentaci ke všem implementovaným funkcím. Tato dokumentace je uvedena v příloze H.

Jako součást softwarového balíku knihovny byla naprogramována i vzorová testovací koncová aplikace psaná v programovacím jazyku C#. Jedná se o příklad aplikace, která pomocí knihovny zobrazuje videosignál ve svém hlavním okně a pomocí jejíhož grafického rozhraní jsou volány téměř všechny funkce dostupné v knihovně. Uživatel tak může okamžitě vyzkoušet funkce, které jsou v knihovně dostupné. S testovací aplikací je dodáván i její okomentovaný zdrojový kód, který obsahuje názorné příklady způsobu používání jednotlivých funkcí knihovny.

Oba instalátory, uživatelský manuál a testovací aplikace jsou dostupné ke stažení na webových stránkách společnosti Workswell, s. r. o.[3] a jsou dispozici i na DVD přiloženém k této diplomové práci.

[3]Distribuovaný software je dostupný z webových stránek: `http://www.workswell.cz/smartvision-sdk-library/`.

KAPITOLA

DESÁTÁ

ZÁVĚR

Hlavním cílem předložené diplomové práce byl vývoj softwarové DLL knihovny pro ovládání digitálních videokamer a následném zpracování videosignálu z těchto kamer. Motivací byl projekt realizovaný v rámci společnosti Workswell, s. r. o.

Projekt byl nejprve zpracováván na míru konkrétnímu zákazníkovi, ovšem po jeho dokončení byla knihovna doplněna o další funkce (zejména práce s termografickými kamerami) a stala se produktem zmíněné společnosti určeným pro další zákazníky. V listopadu roku 2013 bylo řešení knihovny prezentováno autorem diplomové práce na mezinárodní technické konferenci NIDays pořádané společností National Instruments v Praze. Zkušební, časově omezená verze vyvinuté knihovny je v současné době veřejně dostupná na webových stránkách společnosti Workswell, s. r. o.

V rámci této diplomové práce byla autorem vyvinuta kompletní DLL knihovna včetně veškerého doplňkového softwaru, v programovacím jazyku C# byla naprogramována testovací koncová aplikace demonstrující většinu funkcí knihovny a byla sepsána dokumentace ke všem funkcím. Výběr a testování všech algoritmů použitých v knihovně, volba způsobu implementace knihovny a zobrazování videosignálu v koncové aplikaci byly rovněž součástí této práce.

Vyvinutá knihovna umožňuje uživateli zobrazovat videosignál z digitálních videokamer ve své aplikaci psané v libovolném programovacím jazyce podporujícím DLL knihovny. Díky využití ovladače NI-IMAQdx knihovna podporuje velké množství typů kamer s různými rozhraními: GigE Vision, IEEE 1394, IP (Ethernet) a DirectShow (USB kamery). Kamery se všemi zmíněnými rozhraními se knihovnou ovládají stejným způsobem a je tedy snadné v koncové aplikaci vyměnit jednu kameru za jinou.

Knihovna byla naprogramována v programovacím prostředí National Instruments LabVIEW metodou grafického programování. Ačkoliv se nejedná o úplně tradiční metodu programování pokročilých algoritmů, ukázalo se, že zejména použití funkcí dostupných v komponentě NI Vision Development Module značně napomáhá při vývoji algoritmů pro zpracování obrazu a tím

velmi urychluje jejich vývoj. Používané funkce jsou navíc silně optimalizovány, čímž se snižuje výpočetní náročnost programů, které je využívají.

Prvním algoritmem vyvinutým v rámci této diplomové práce je automatické zaostřování objektivu digitální kamery dodané původním zákazníkem. Objektiv byl vyroben zákazníkem a bylo možné jej posouvat pomocí krokového motoru. Z literatury byly nastudovány některé z metod pro vyhodnocení ostrosti snímků z kamery a vybrané funkce byly implementovány a otestovány. Pro řídící blok algoritmu automatického zaostřování byla zvolena funkce $F_{tenengrad}$. Byl vybrán vhodný způsob filtrace průběhu této funkce a navržen způsob detekce růstu a klesání průběhu funkce. Byl implementován algoritmus, který řídí krokový motor posouvající objektiv a polohuje ho do pozice nejlepšího zaostření, jež odpovídá vrcholu funkce ostrosti snímků.

V průběhu testů automatického zaostřování bylo zjištěno, že použitý mechanismus ostření vykazuje vůli převodovky, která zabraňovala přesnému zaostření. Problém byl vyřešen implementací parametru zohledňujícího vůli při obracení směru pohybu motoru. Rychlost motoru dodaného zákazníkem ovlivňovala odolnost automatického zaostřování vůči pohybu zaostřovaného výřezu scény. V případě použití rychlejšího motoru by byl vliv pohyblivé scény menší. Vyvinutý algoritmus automatického zaostřování funguje spolehlivě i při poměrně málo kontrastních a detailních scénách. Vzhledem k ceně kamer již z výroby vybavených automatickým zaostřováním byla knihovna s algoritmem pro automatické zaostřování pro zákazníka velkým přínosem.

Dalším implementovaným algoritmem bylo sledování objektu ve videosignálu a digitální stabilizace obrazu. Princip těchto funkcí byl založen na metodě vzájemné korelace šablony reprezentující vyhledávaný objekt a snímků videosignálu. Pro implementaci byly použity funkce dostupné v NI Vision Development Module. Tyto funkce rozšiřující metodu vzájemné korelace o doplňkové algoritmy za účelem spolehlivějšího vyhledávání šablony ve snímcích. Implementované sledování objektu v obraze funguje spolehlivě pro objekty, které nemění zásadně svoji velikost. Rotace, posun objektů ve snímku či změna světelných podmínek ve snímané scéně nedělají algoritmu větší problémy. Implementovaná digitální stabilizace je vhodná především pro eliminaci otřesů scény, ve které se nepohybuje příliš mnoho objektů, protože použitý algoritmus závisí na vyhledávání naučeného středu obrazu ve snímcích videosignálu.

V knihovně byly dále implementovány funkce pro úpravy obrazu (změna jasu, kontrastu a barevnosti snímků) a softwarové doostřování snímků. Byla vyvinuta metoda pro přístup k jednotlivým snímkům videosignálu a implementována možnost záznamu videosignálu formou nahrávek ve formátu AVI. Pro práci s termografickými kamerami byly implementovány nástroje pro měření teplot a statistik v konvertovaném termografickém snímku.

Protože většina vyvinutých algoritmů, o kterých se pojednává v této práci, pracuje s videosignálem měnícím se v čase, obrázky uvedené v textu poskytují jen částečnou ilustraci funkce těchto algoritmů. Názorným shrnutím výsledků implementace knihovny jsou tak videonahrávky uložené na přiloženém DVD disku.

LITERATURA

[1] Choosing the Right Camera Bus. *National Instruments* [online]. I 07, 2013 [cit. 2013-12-01]. Dostupné z: http://www.ni.com/white-paper/5386/en/

[2] Making a LabVIEW VI an Embedded Child Window of a Visual C# User Interface Window. *National Instruments* [online]. IX 20, 2006 [cit. 2013-12-02]. Dostupné z: http://www.ni.com/example/29338/en/

[3] About Messages and Message Queues. *Windows Desktop Development: Windows Dev Center* [online]. 11/16/2013 [cit. 2013-12-02]. Dostupné z: http://msdn.microsoft.com/en-us/library/windows/desktop/ms644927%28v=vs.85%29.aspx

[4] LEE, Je-Ho *et al.* Implementation of a passive automatic focusing algorithm for digital still camera. *IEEE Transactions on Consumer Electronics*. AUGUST 1995, vol. 41, issue 3, s. 449-454. DOI: 10.1109/30.468047. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=468047

[5] KEHTARNAVAZ, N. a H.-J. OH. Development and real-time implementation of a rule-based auto-focus algorithm. *Real-Time Imaging*. 2003, vol. 9, issue 3, s. 197-203. DOI: 10.1016/S1077-2014(03)00037-8. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S1077201403000378

[6] Gradient based methods. MARSHALL, David, Dr. *Vision Systems* [online]. [8. září 1997] [cit. 2013-12-06]. Dostupné z: http://www.cs.cf.ac.uk/Dave/Vision_lecture/node28.html

[7] MLSNA, Phillip A. a Jeffrey J. RODRÍGUEZ. Gradient and Laplacian Edge Detection. BOVIK, Al. *Handbook of image and video processing*. 2nd ed. Burlington: Elsevier Academic Press, 2005, s. 540-543. ISBN 0-12-119792-1.

[8] SANTOS, A., C. ORTIZ DE SOLÓRZANO, J. J. VAQUERO, J. M. PEÑA, N. MALPICA a F. DEL POZO. Evaluation of autofocus functions in molecular cyto-

genetic analysis. *Journal of Microscopy*. 1997, vol. 188, issue 03, s. 264-272. DOI: 10.1046/j.1365-2818.1997.2630819.x. Dostupné z: http://doi.wiley.com/10.1046/j.1365-2818.1997.2630819.x

[9] YU SUN, S. DUTHALER a B.J. NELSON. Autofocusing algorithm selection in computer microscopy. In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems: Edmonton, AB, Canada, 2-6 August 2005*. Piscataway, N.J.: IEEE Operations Center, c2005, s. 70-76. ISBN 0780389123. DOI: 10.1109/IROS.2005.1545017. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1545017

[10] YEO, T., S.O. JAYASOORIAH a R. SINNIAH. Autofocusing for tissue microscopy. *Image and Vision Computing*. 1993, vol. 11, issue 10, s. 629-639. DOI: 10.1016/0262-8856(93)90059-P. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/026288569390059P

[11] KROTKOV, Eric. Focusing. *International Journal of Computer Vision*. 1988, vol. 1, issue 3, s. 223-237. DOI: 10.1007/BF00127822. Dostupné z: http://link.springer.com/10.1007/BF00127822

[12] GROEN, Frans C. A., Ian T. YOUNG a Guido LIGTHART. A comparison of different focus functions for use in autofocus algorithms. *Cytometry*. 1985, vol. 6, issue 2, s. 81-91. DOI: 10.1002/cyto.990060202. Dostupné z: http://doi.wiley.com/10.1002/cyto.990060202

[13] VOLLATH, D. Automatic focusing by correlative methods. *Journal of Microscopy*. 1987, vol. 147, issue 3, s. 279-288. DOI: 10.1111/j.1365-2818.1987.tb02839.x. Dostupné z: http://doi.wiley.com/10.1111/j.1365-2818.1987.tb02839.x

[14] VOLLATH, D. The influence of the scene parameters and of noise on the behaviour of automatic focusing algorithms. *Journal of Microscopy*. 1988, vol. 151, issue 2, s. 133-146. DOI: 10.1111/j.1365-2818.1988.tb04620.x. Dostupné z: http://doi.wiley.com/10.1111/j.1365-2818.1988.tb04620.x

[15] SARUCHI. Adaptive Sigmoid Function to Enhance Low Contrast Images. *International Journal of Computer Applications* 2012-10-20, vol. 55, issue 4, s. 45-49. DOI: 10.5120/8747-2634. Dostupné z: http://research.ijcaonline.org/volume55/number4/pxc3882634.pdf

[16] Non Linear Mappings. IBÁÑEZ, Luis et al. *The ITK software guide: updated for ITK version 2.4* [online]. 2nd ed. Clifton Park, N.Y.: Kitware, 2003, s. 149-151 [cit. 2013-12-

15]. ISBN 1-930934-15-7. Dostupné z: http://www.itk.org/ItkSoftwareGuide.pdf

[17] Recommendation ITU-R BT.601-7: Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios. INTERNATIONAL TELECOMMUNICATION UNION. *Recommendation BT.601* [online]. 03/2011 [cit. 2013-12-16]. Dostupné z: http://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.601-7-201103-I!!PDF-E.pdf

[18] Any ideas how 'Preserve Luminosity' works in Photoshop's Pho. In: *BorlandTalk.com: Borland discussion newsgroups* [online]. Apr 06, 2006 [cit. 2013-12-16]. Dostupné z: http://www.borlandtalk.com/any-ideas-how-preserve-luminosity-works-in-photoshops-pho-vt94669.html

[19] Unsharp Filter. FISHER, Robert. *The Hypermedia Image Processing Reference* [online]. 2004 [cit. 2013-12-17]. Dostupné z: http://homepages.inf.ed.ac.uk/rbf/HIPR2/unsharp.htm

[20] LEWIS, J. P. Fast Normalized Cross-Correlation. *Vision Interface* [online]. 1995, Vol. 95, s. 15-19 [cit. 2013-12-22]. Dostupné z: http://www.scribblethink.org/Work/nvisionInterface/vi95_lewis.pdf

[21] MARTIN, Jérôme a James L. CROWLEY. Experimental Comparison of Correlation Techniques. *IAS-4, International Conference on Intelligent Autonomous Systems* [online]. 1995 [cit. 2013-12-22]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.50.8468

[22] NI Vision 2013 Concepts Help: Pattern Matching Techniques. NATIONAL INSTRUMENTS. *NI Product Manuals* [online]. June 2013 [cit. 2013-12-22]. Dostupné z: http://zone.ni.com/reference/en-XX/help/372916P-01/nivisionconcepts/pattern_matching_techniques/

[23] IMAQ Match Pattern 4 VI. NATIONAL INSTRUMENTS. *NI Vision 2013 for LabVIEW Help* [online]. June 2013 [cit. 2013-12-22]. Dostupné z: http://zone.ni.com/reference/en-XX/help/370281U-01/imaqvision/imaq_match_pattern_4/

[24] AUTOMATED IMAGING ASSOCIATION. *Pixel Format Naming Convention*. Version 1.1. Ann Arbor, February 1, 2013 [cit. 2013-12-27]. Dostupné z: http://www.visiononline.org/userassets/aiauploads/file/Pixel_Format_Naming_Convention_1-1-00.pdf

[25] FLIR SYSTEMS, Inc. *FLIR Ax5 series: User's manual*. Wilsonville, USA, 2013-10-18 [cit. 2013-12-27]. Dostupné z: `http://support.flir.com/DocDownload/Assets/86/English/T559770-en-US$AB.pdf`

[26] Activation FAQ. NATIONAL INSTRUMENTS. *Products and Services: Product Activation* [online]. 2012 [cit. 2014-01-01]. Dostupné z: `http://www.ni.com/license/activation_faq.htm`

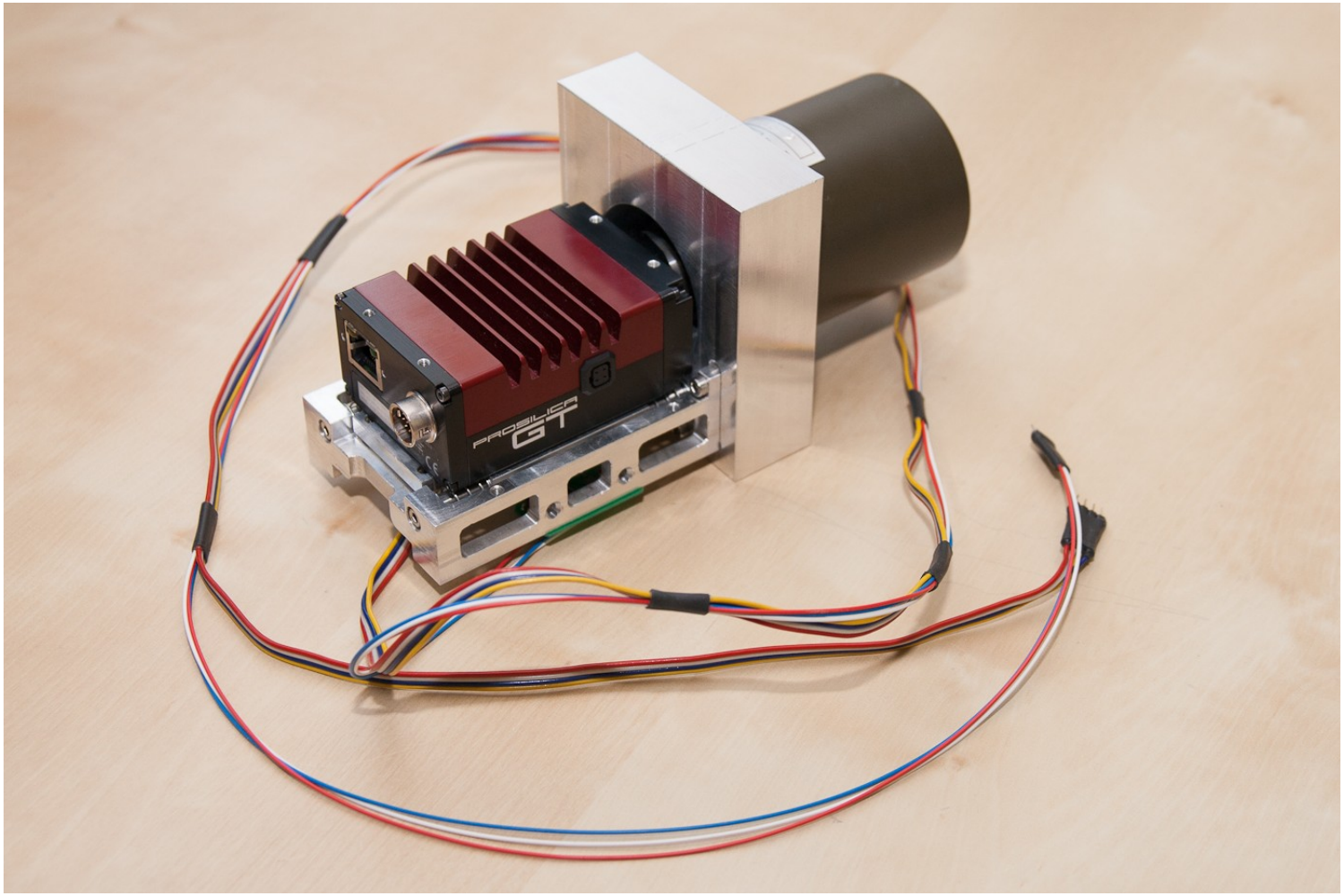
Přílohy

PŘÍLOHA

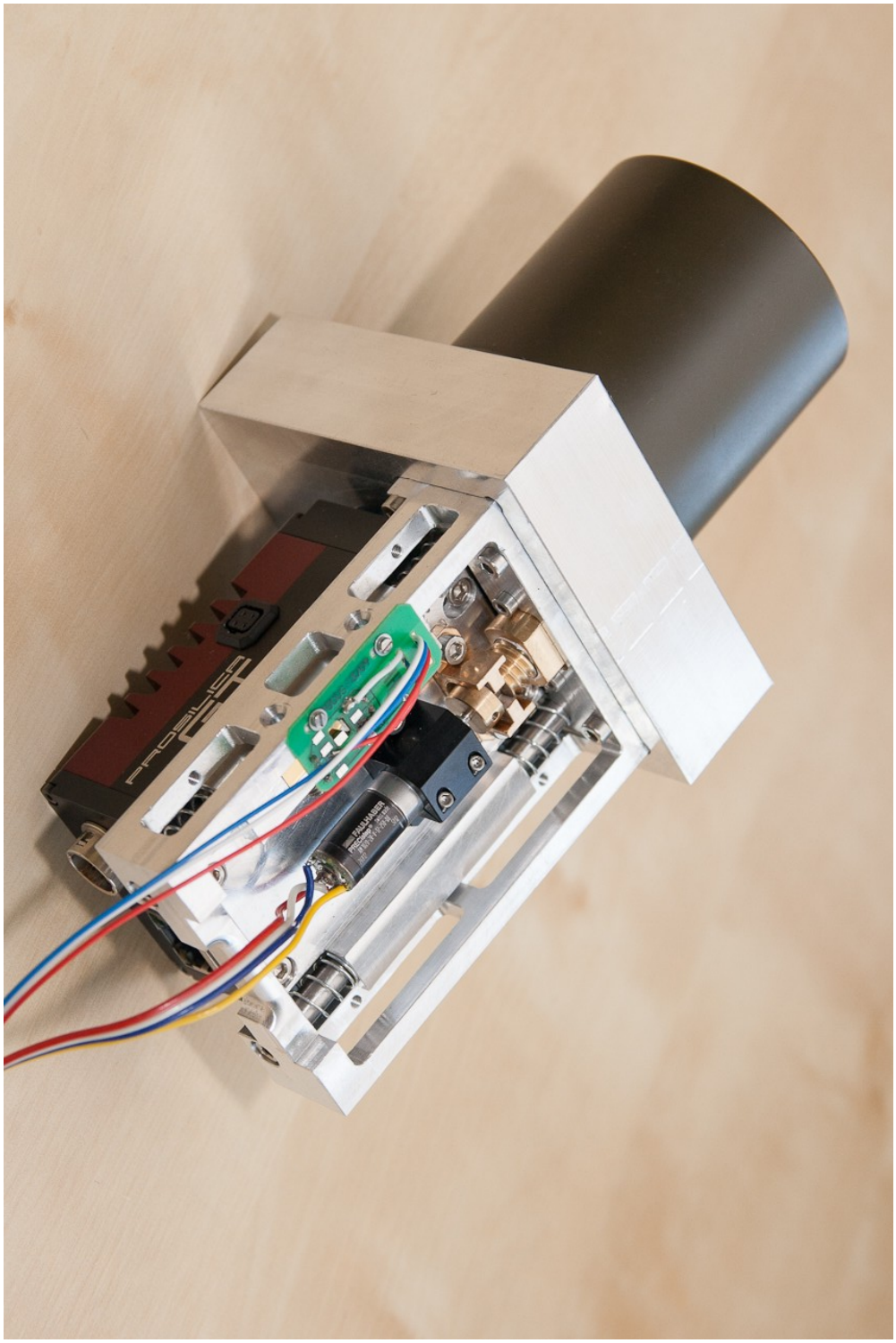
A

FOTOGRAFIE SESTAVY KAMERY A MECHANISMU OBJEKTIVU

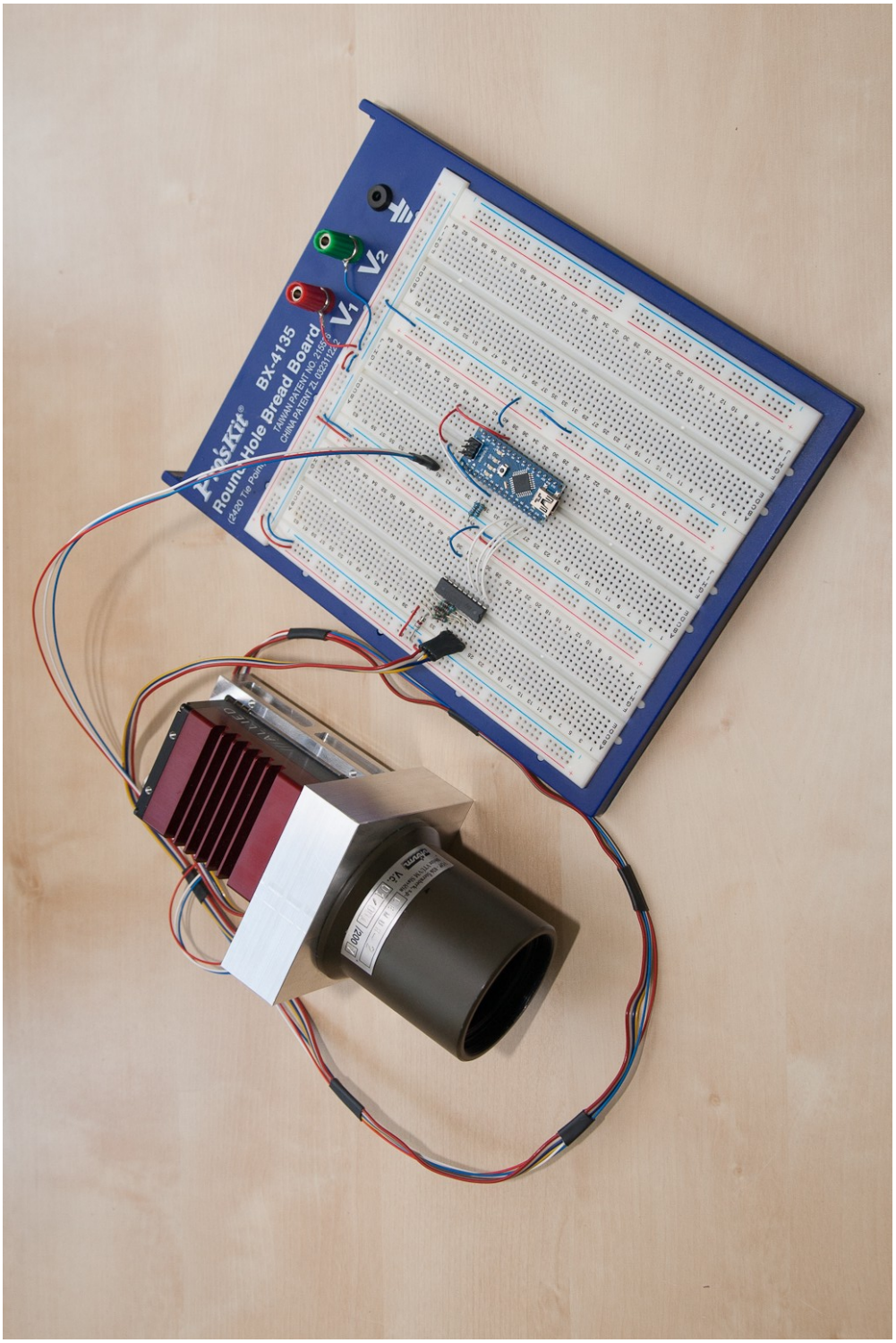
Obrázek A.1.: Sestava kamery a zaostřovacího mechanismu objektivu.



Obrázek A.2.: Sestava kamery a zaostřovacího mechanismu objektivu.



Obrázek A.3.: Sestava kamery a mechanismu objektivu. Detail krokového motoru a koncových spínačů.



Obrázek A.4.: Sestava kamery a mechanismu objektivu vedle vývojové desky Arduino.

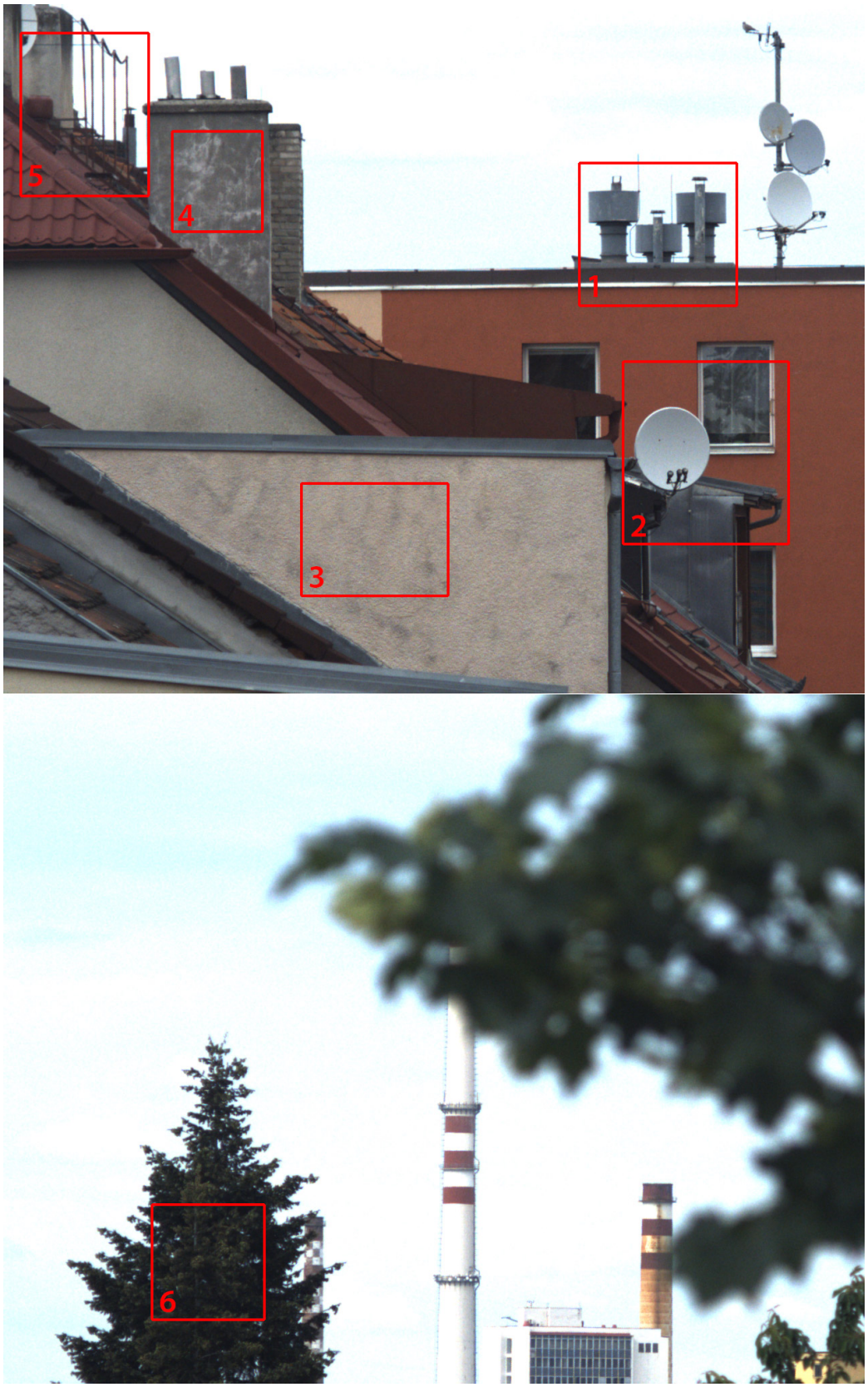
PŘÍLOHA

B

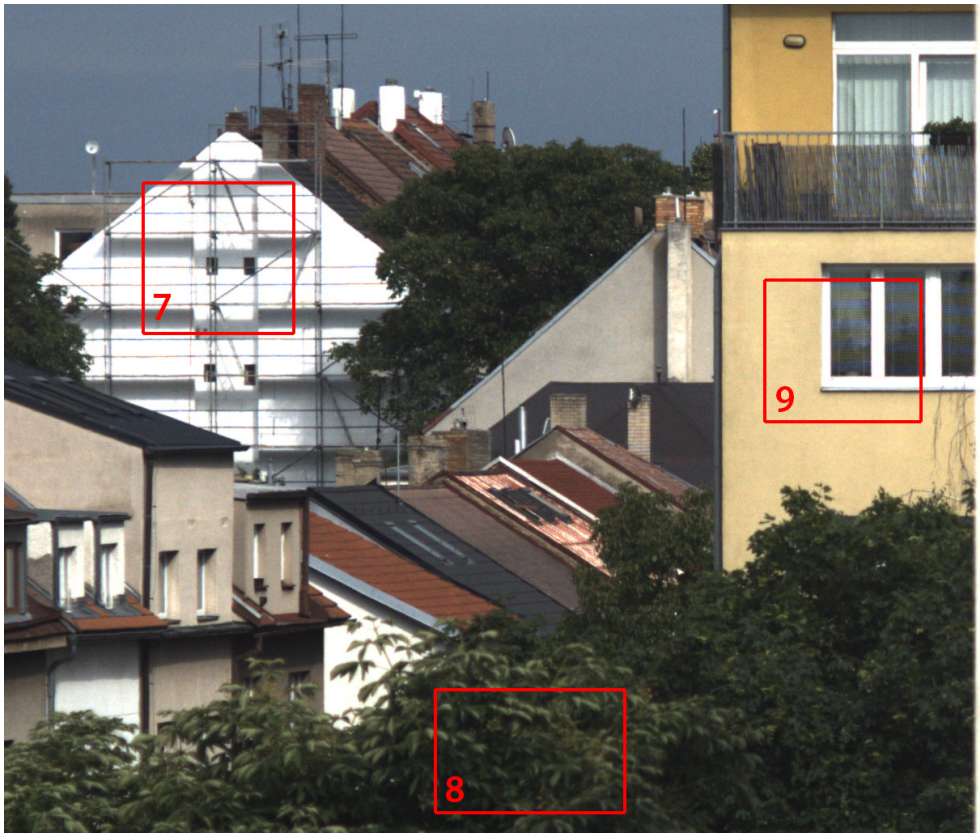
FUNKCE PRO VYHODNOCENÍ OSTROSTI

Na obrázcích B.1 a B.2 jsou zobrazeny tři scény, na kterých byly porovnávány funkce pro hodnocení ostrosti snímku. Výřezy použité k porovnávání jsou vyznačeny červeným rámečkem a pořadovým číslem. Tyto výřezy jsou zachyceny na obrázku B.3 při minimálním a maximálním zaostření objektivu.

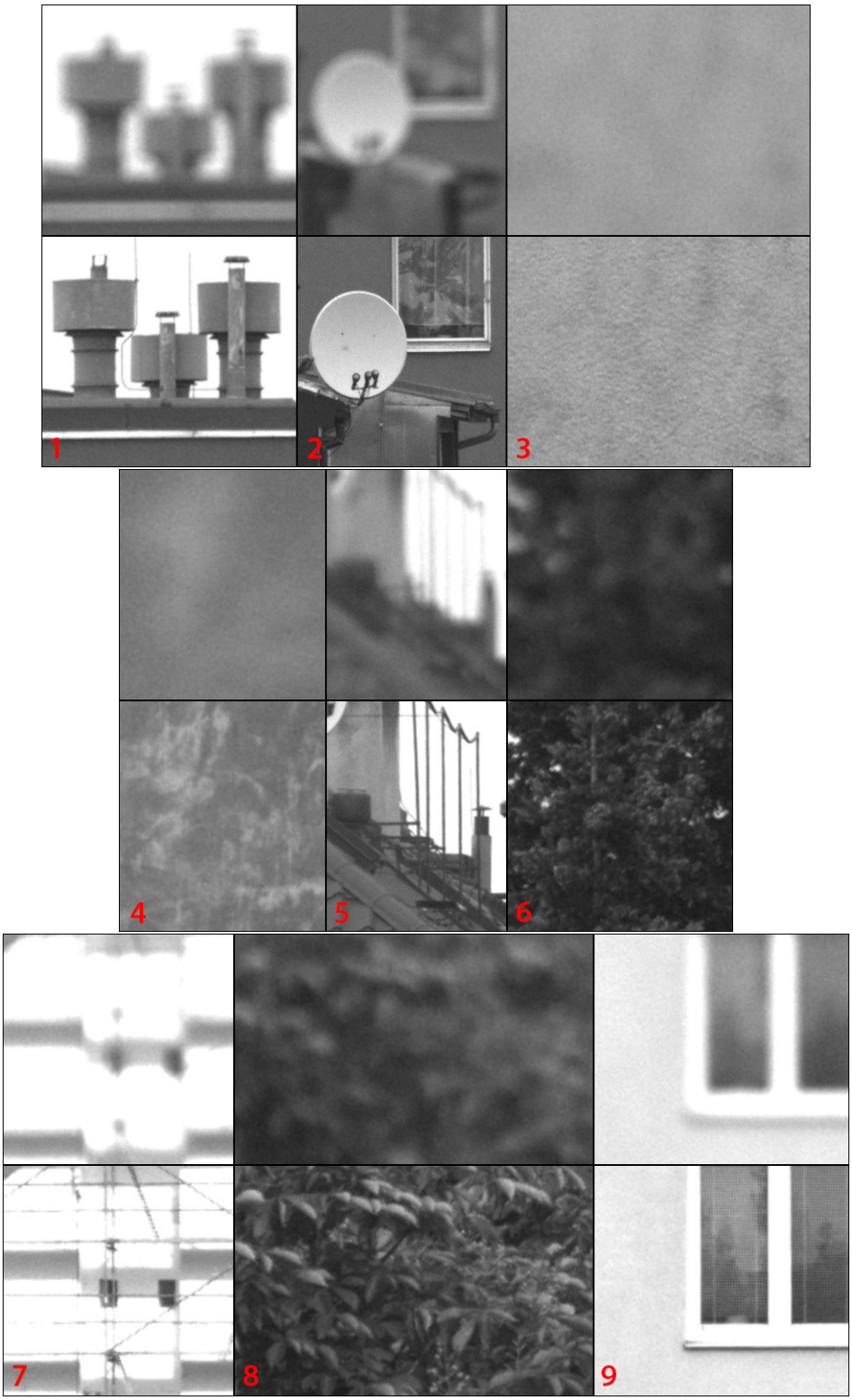
Grafy na obrázcích B.4 až B.8 znázorňují průběhy různých funkcí pro vyhodnocení ostrosti snímku. Při měření hodnot funkcí byl motoru kamery byl odeslán příkaz pro kontinuální pohyb směrem ke správné poloze zaostření objektivu a byla nahrávána sekvence snímků při tomto pohybu. Pohyb motoru byl zastaven až po posunutí objektivu za polohu nejlepšího zaostření (poloha odpovídá vrcholům zobrazených průběhů). V daném výřezu obrazu z kamery pak byla všemi funkcemi průběžně počítána ostrost každého snímku ve videosignálu. Jelikož funkce ostrosti nabývají řádově jiných hodnot, byly výsledky pro účely porovnání normovány (minimální hodnotě průběhu odpovídá nula a maximální hodnotě jednička).



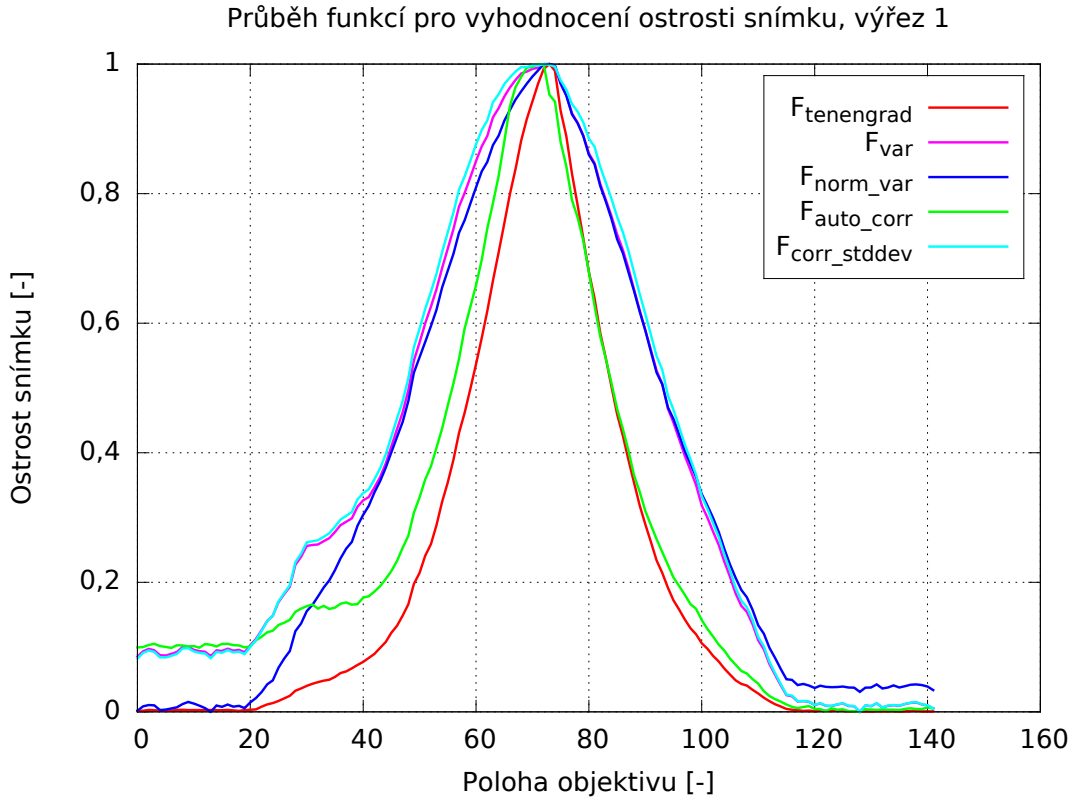
Obrázek B.1.: První dvě testovací scény s červeně vyznačenými výřezy, ve kterých byly vyhodnocovány funkce ostrosti

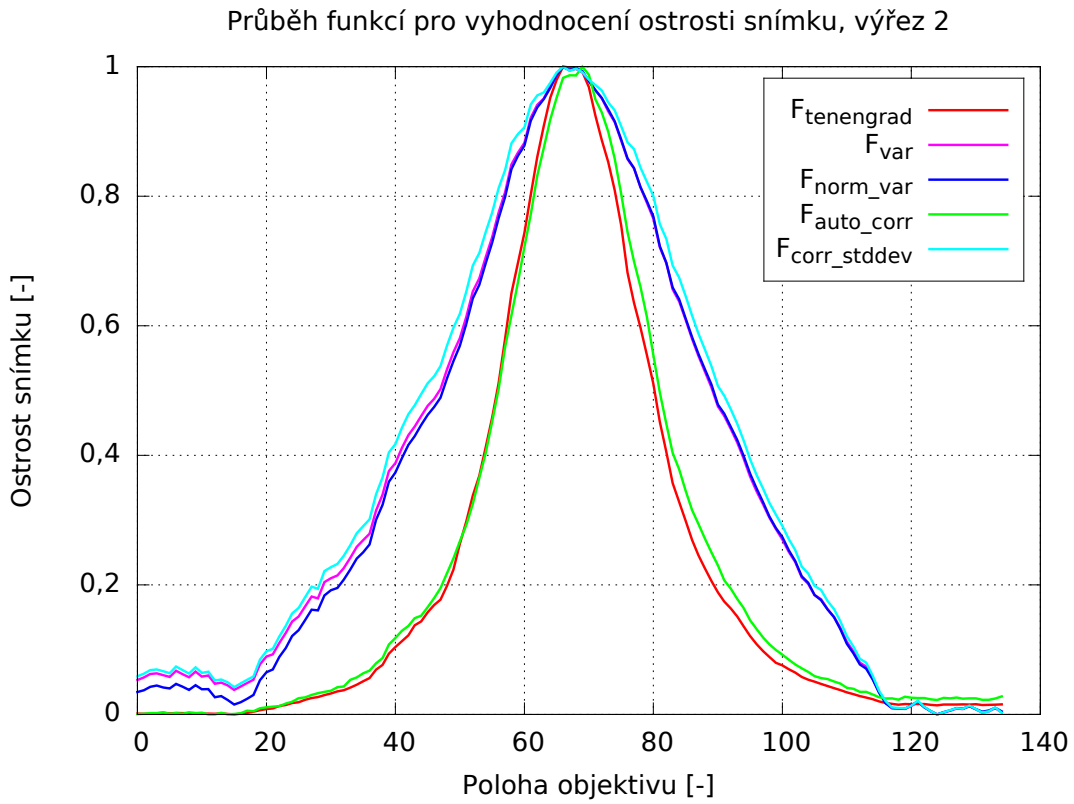


Obrázek B.2.: Třetí testovací scéna s červeně vyznačenými výřezy, ve kterých byly vyhodnocovány funkce ostrosti

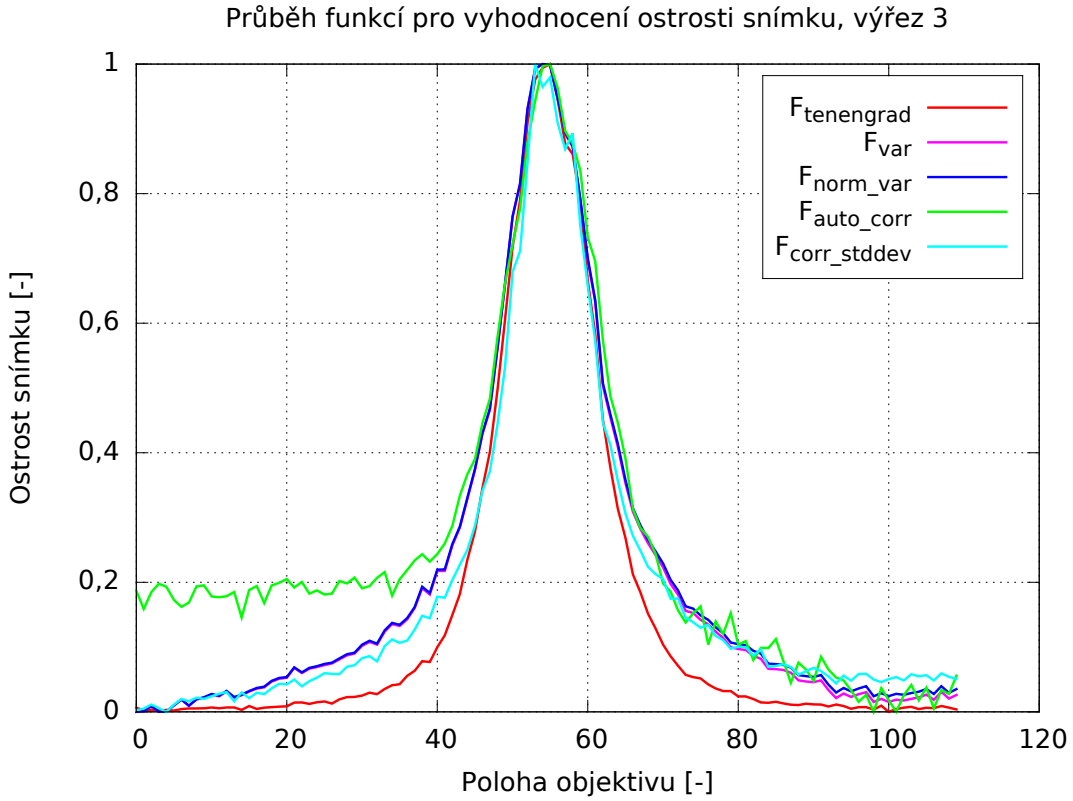


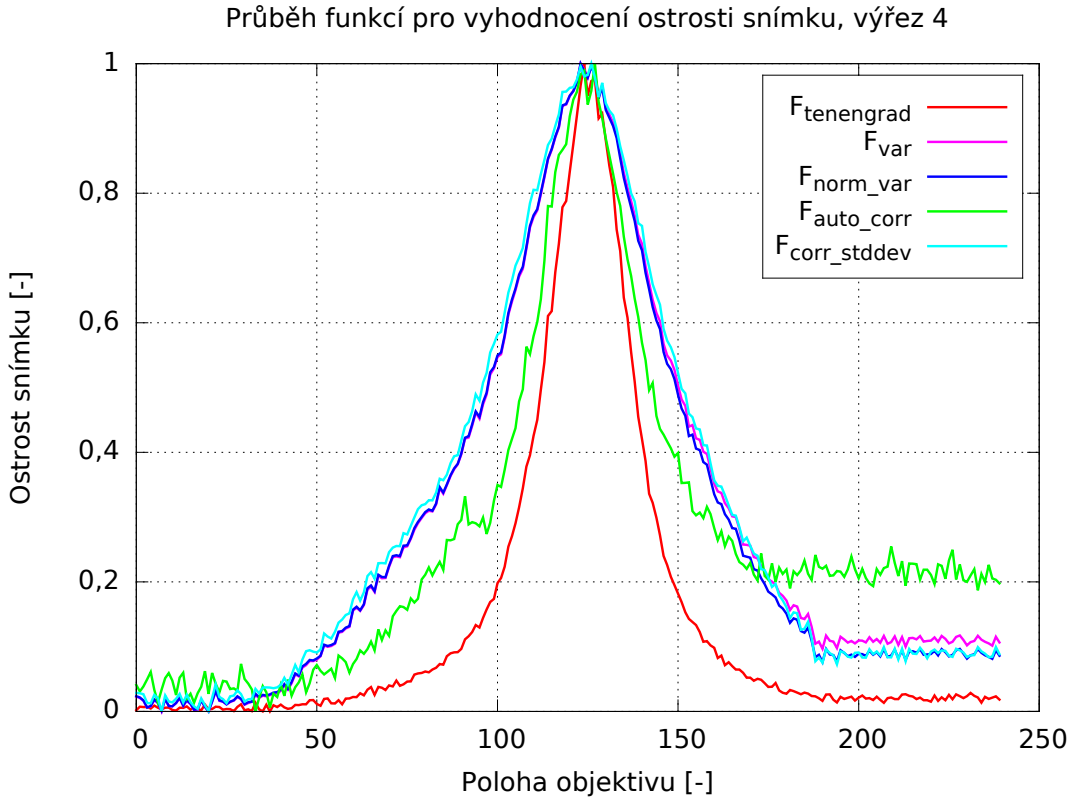
Obrázek B.3.: Výřezy z testovacích scén, na kterých byly testovány funkce pro měření ostrosti snímku.



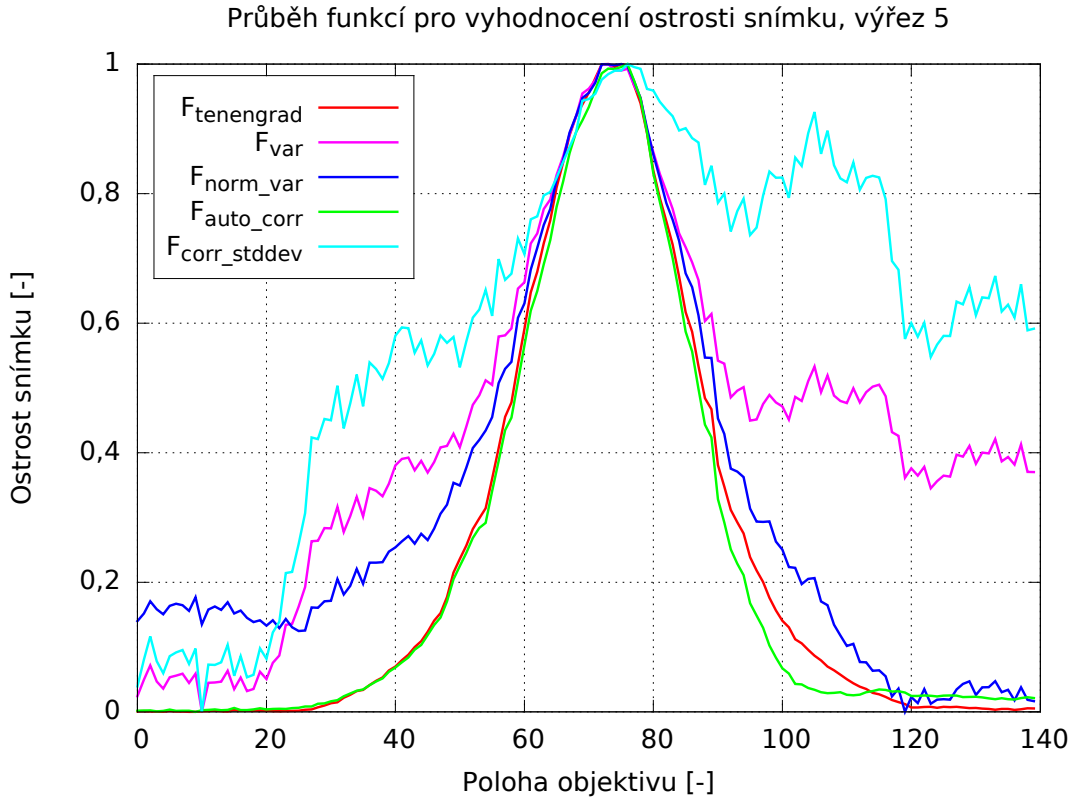


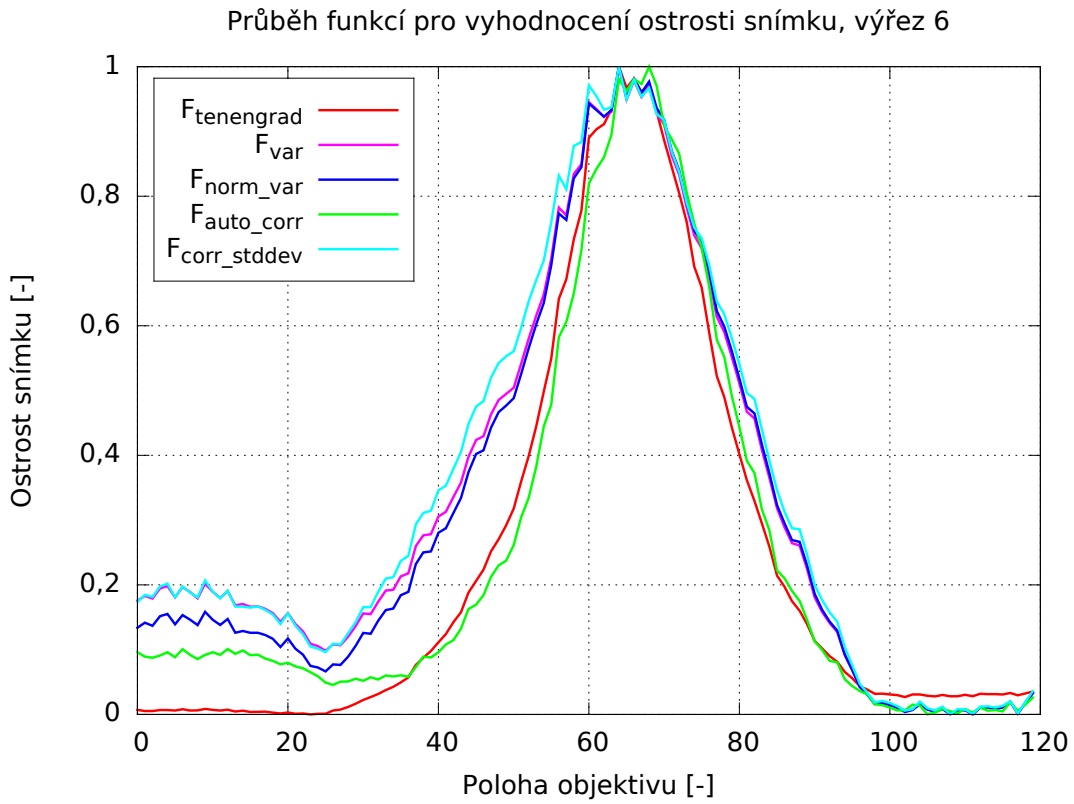
Obrázek B.4.: Grafy průběhů funkcí pro vyhodnocení ostrosti výřezů 1 a 2



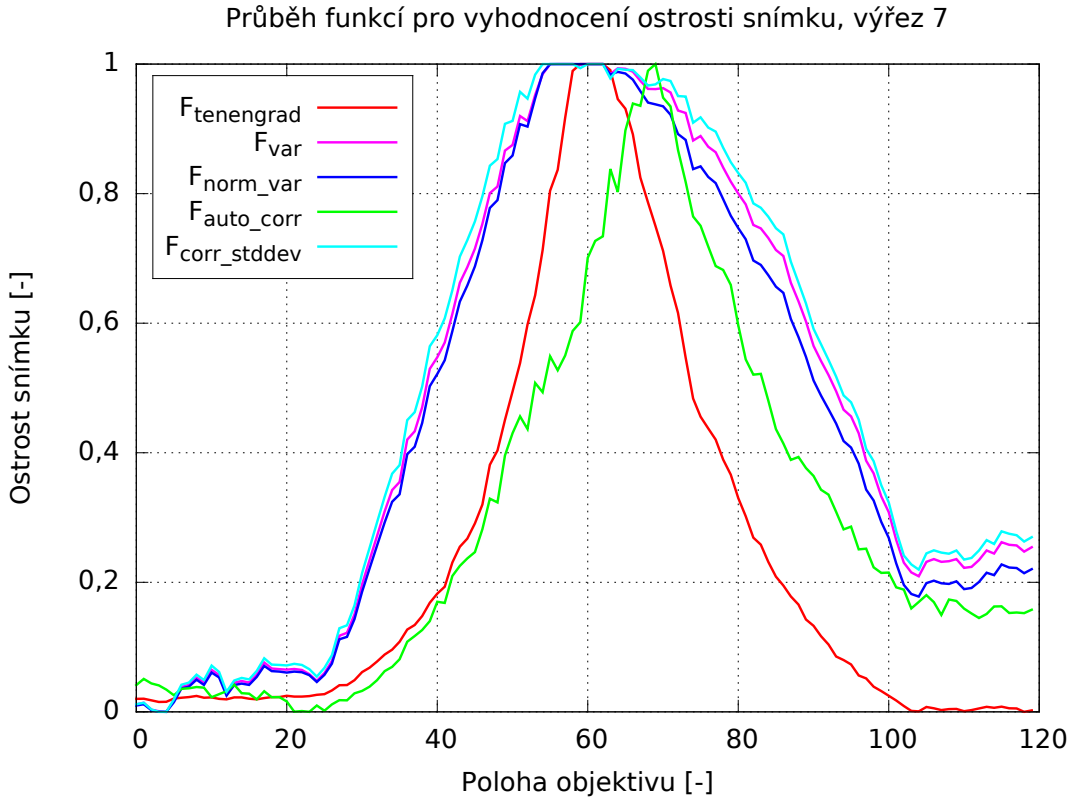


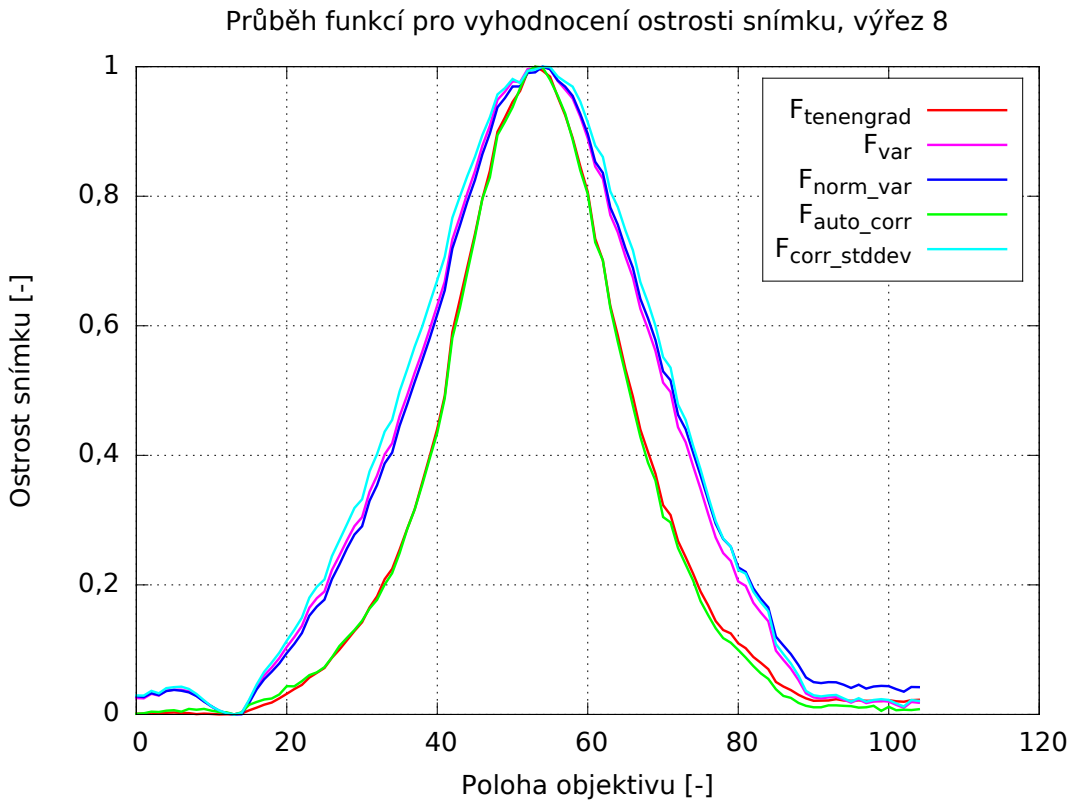
Obrázek B.5.: Grafy průběhů funkcí pro vyhodnocení ostrosti výřezů 3 a 4



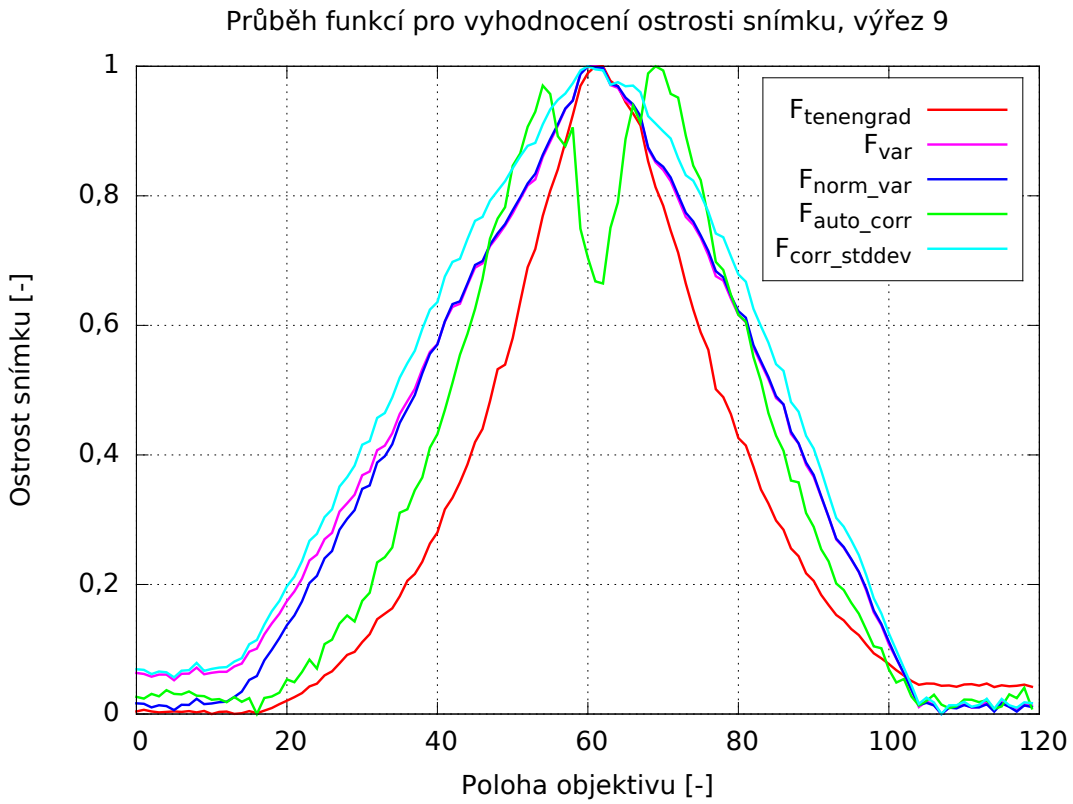


Obrázek B.6.: Grafy průběhů funkcí pro vyhodnocení ostrosti výřezů 5 a 6





Obrázek B.7.: Grafy průběhů funkcí pro vyhodnocení ostrosti výřezů 7 a 8



Obrázek B.8.: Graf průběhů funkcí pro vyhodnocení ostrosti výřezů 9

PŘÍLOHA

C

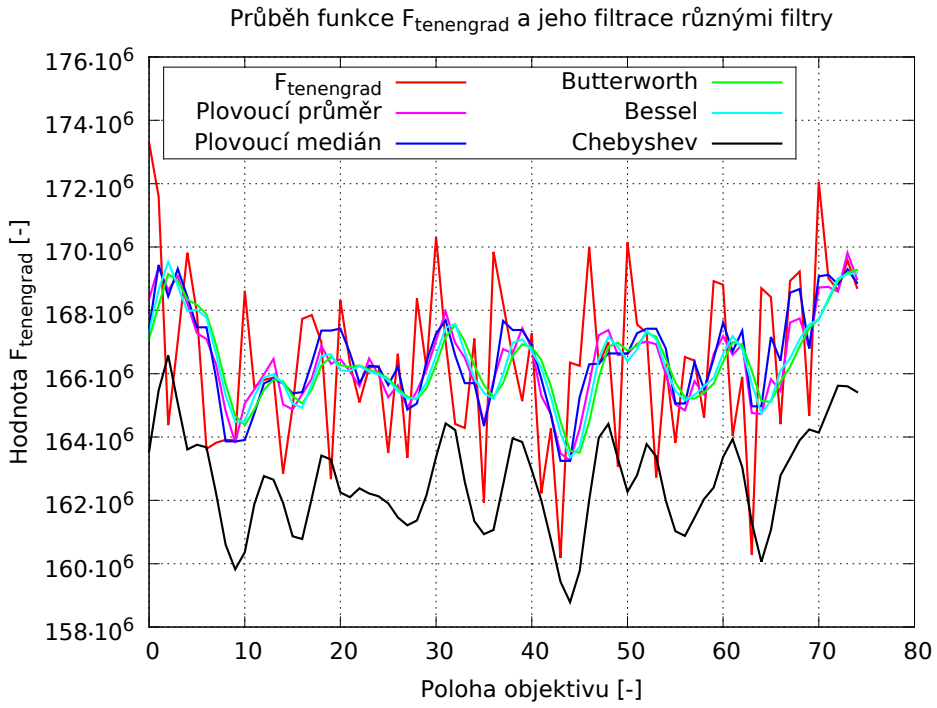
GRAFY FILTRACE FUNKCE PRO VYHODNOCENÍ OSTROSTI

Měření 1

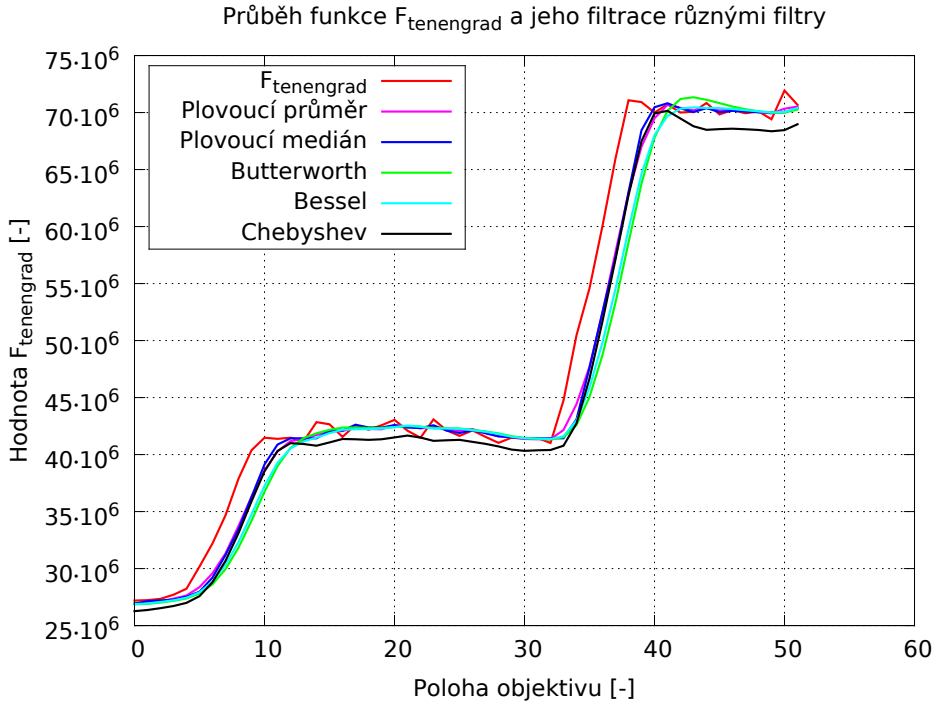
Řád filtrů typu dolní propust byl nastaven na 2, zlomová frekvence na 3 Hz a pro Chebyshevův filtr byl parametr zvlnění v propustném pásmu („ripple“) nastaven na $0,2$ dB. Filtry plovoucí průměr a medián byly nastaveny na délku 15 % snímkové frekvence videosignálu (tedy při 30 snímcích za vteřinu byl počítán průměr a medián z posledních 4 hodnot).

Měření 2

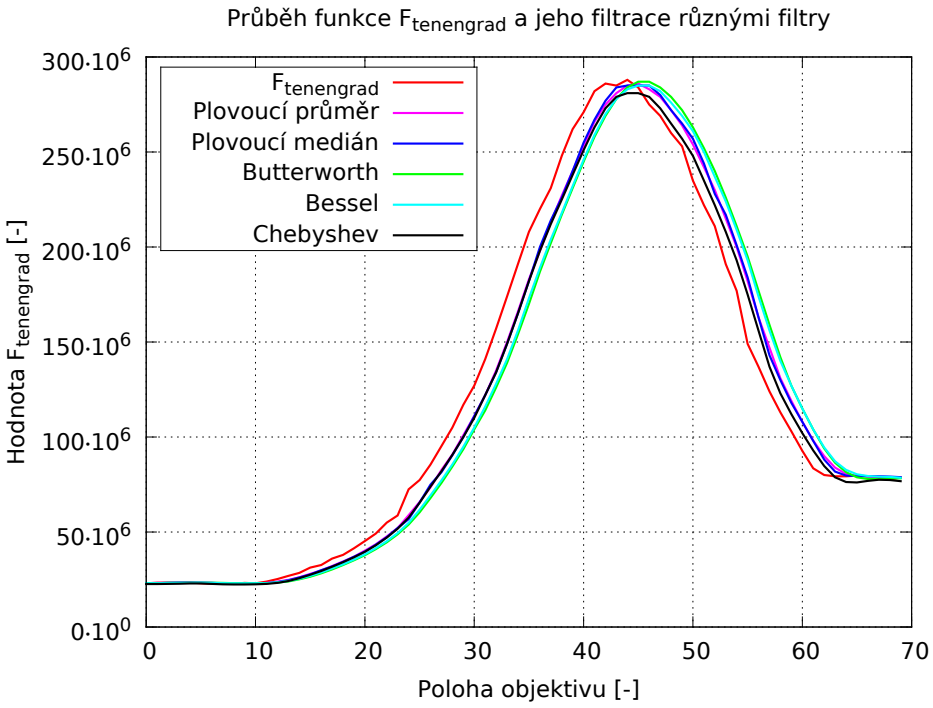
Zlomová frekvence filtrů typu dolní propust nastavena na $1,5$ Hz (řád filtrů a nastavení Chebyshevova filtru bylo stejné jako při měření 1). Filtry plovoucí průměr a medián byly nastaveny na délku 30 % snímkové frekvence videosignálu (při 30 snímcích za vteřinu byl počítán průměr a medián z posledních 9 hodnot).



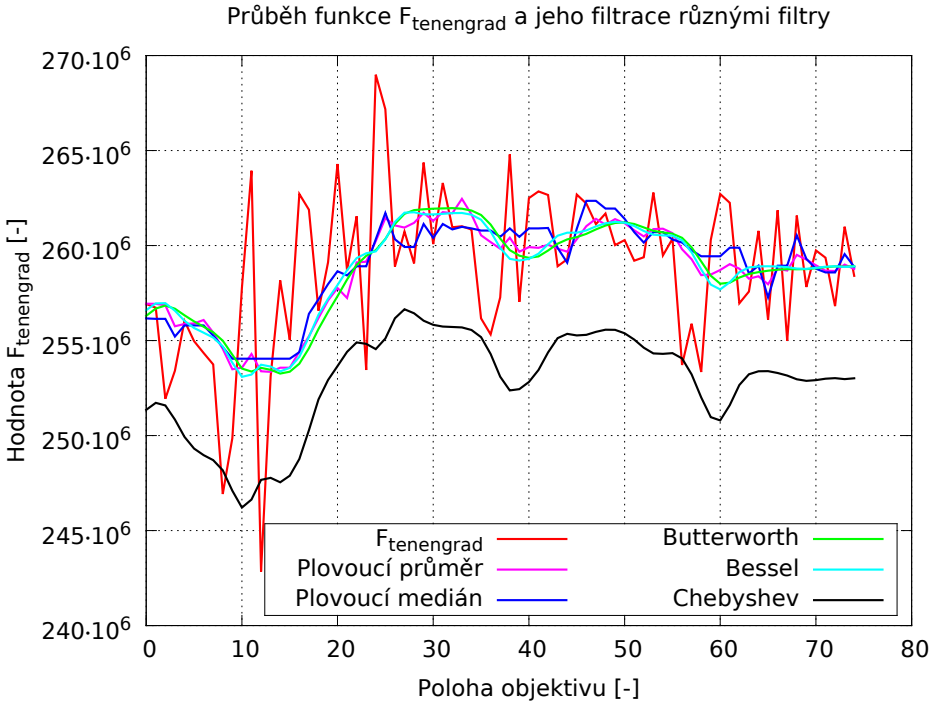
Obrázek C.1.: Měření 1 - průběhy filtrací šumu při konstantní pozici objektivu.



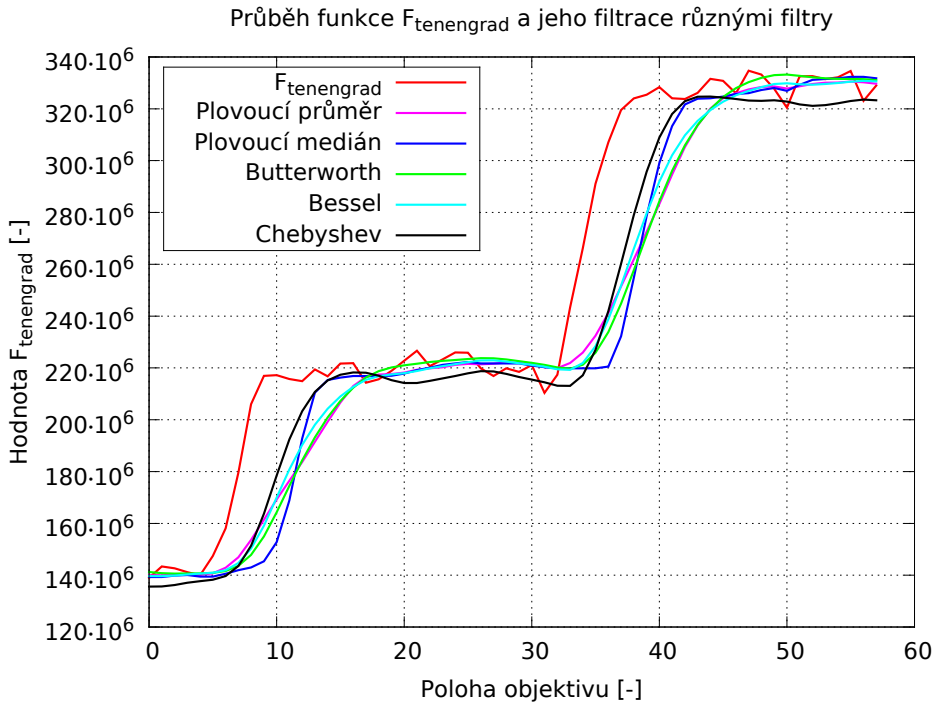
Obrázek C.2.: Měření 1 - průběhy filtrací při skokových posunech objektiv.



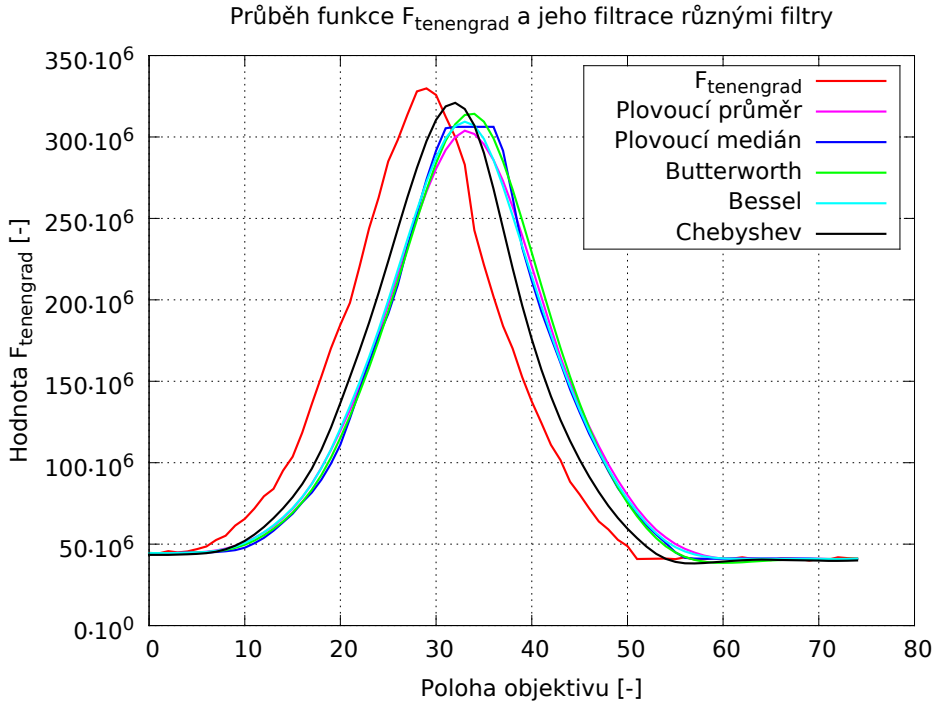
Obrázek C.3.: Měření 1 - průběhy filtrací při přejezdu přes polohu s maximální ostrostí obrazu.



Obrázek C.4.: Měření 2 - průběhy filtrací šumu při konstantní pozici objektivu.



Obrázek C.5.: Měření 2 - průběhy filtrací při skokových posunech objektiv.



Obrázek C.6.: Měření 2 - průběhy filtrací při přejezdu přes polohu s maximální ostrostí obrazu.

PŘÍLOHA

D

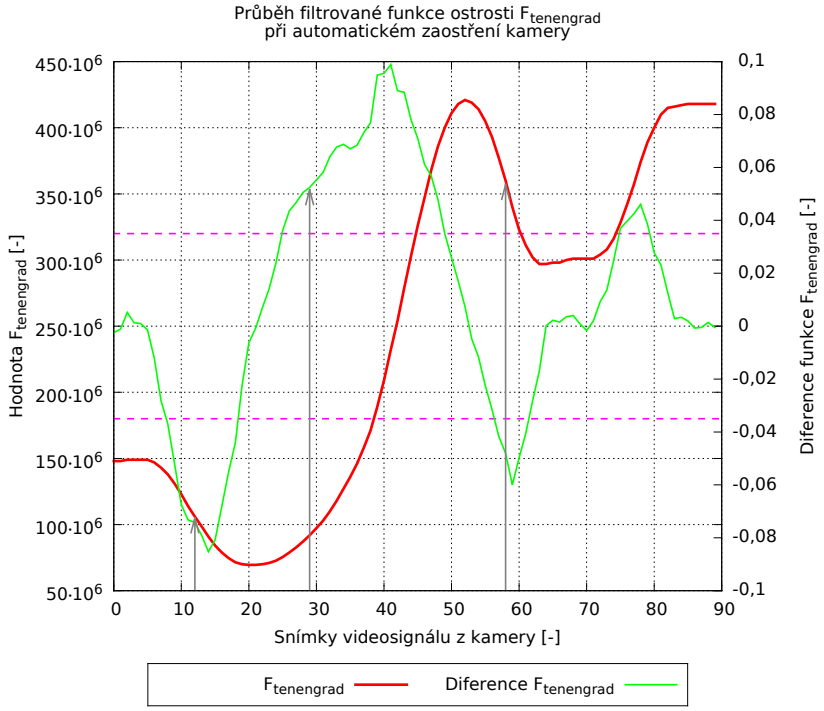
GRAFY PRŮBĚHŮ FUNKCE OSTROSTI PŘI AUTOMATICKÉM OSTŘENÍ

V této příloze jsou zobrazeny průběhy filtrované funkce ostrosti a její diference při testech automatického zaostřování. Situace, kterým znázorněné průběhy odpovídají, jsou uvedeny v popisku příslušných obrázků. Svislé šedé šipky označují místa, kde algoritmus automatického zaostřování detekoval klesání nebo růst funkce ostrosti.

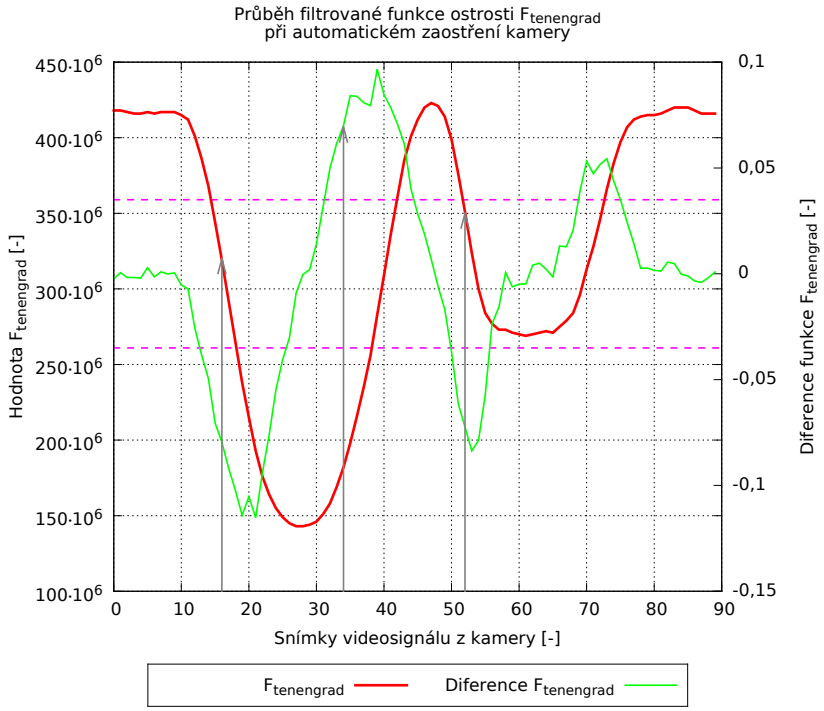
Na obrázku D.1 začal krokový motor posouvat objektiv směrem od místa s největší mírou ostrosti. Proto byl obrácen směr hledání a po nalezení vrcholu funkce ostrosti motor vrátil objektiv na pozici s nejlepším zaostřením.

Obrázek D.2 ilustruje situaci, při níž se již objektiv nacházel v pozici s nejvyšší ostrostí obrazu a bylo spuštěno zaostření. Po první detekci klesání funkce ostrosti bylo doostřeno stejně jako v předchozím uvedeném případě.

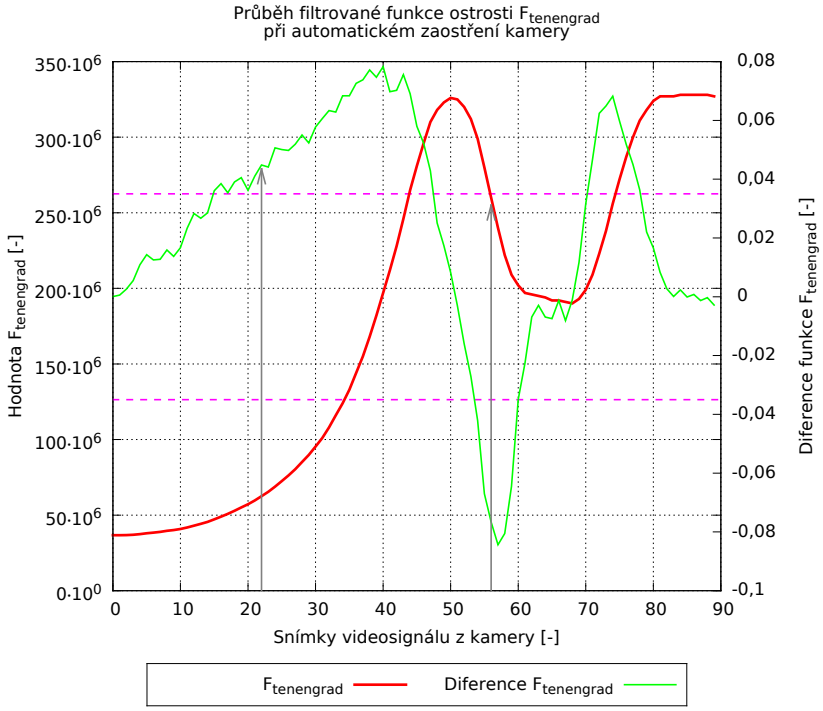
Na obrázku D.4 je zobrazen průběh funkce ostrosti při automatickém zaostření bez kompenzace vůle v převodovce. Okolo šedesátého snímku byl algoritmem vyslán krokovému motoru pokyn k obrácení směru pohybu za účelem návratu objektivu na polohu s maximální mírou ostrosti. Počet pulzů potřebný k návratu však byl umořen na překonání vůle převodovky a objektiv se vůbec nepohnul (konstantní hodnota průběhu funkce ostrosti od cca šedesátého snímku).



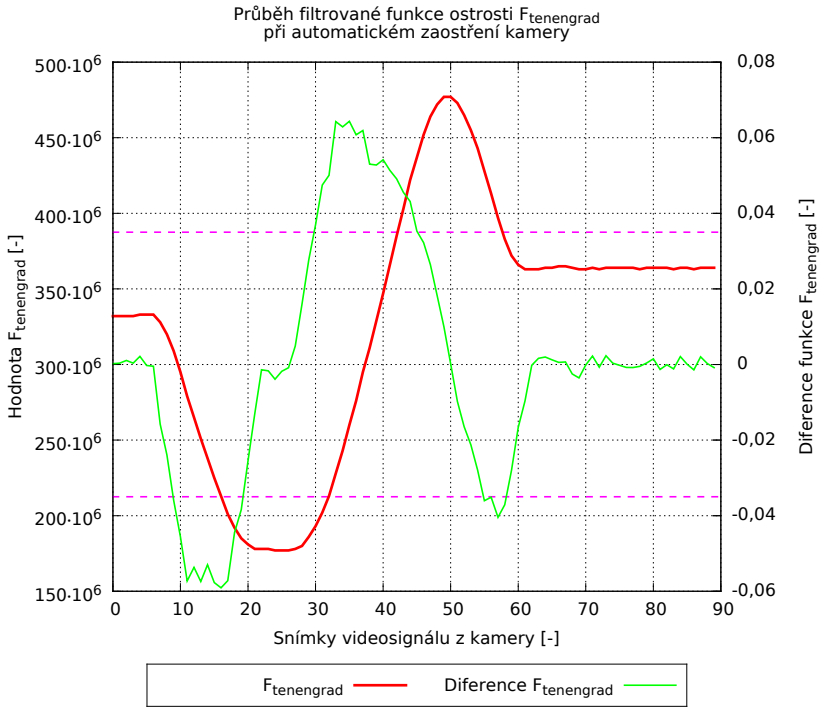
Obrázek D.1.: Průběh funkce ostrosti při startu objektivu směrem k pozici s menší ostrostí.



Obrázek D.2.: Průběh funkce ostrosti při startu objektivu z pozice s nejvyšší ostrostí.



Obrázek D.3.: Průběh funkce ostrosti při startu objektivu směrem k pozici s nejvyšší ostrostí.



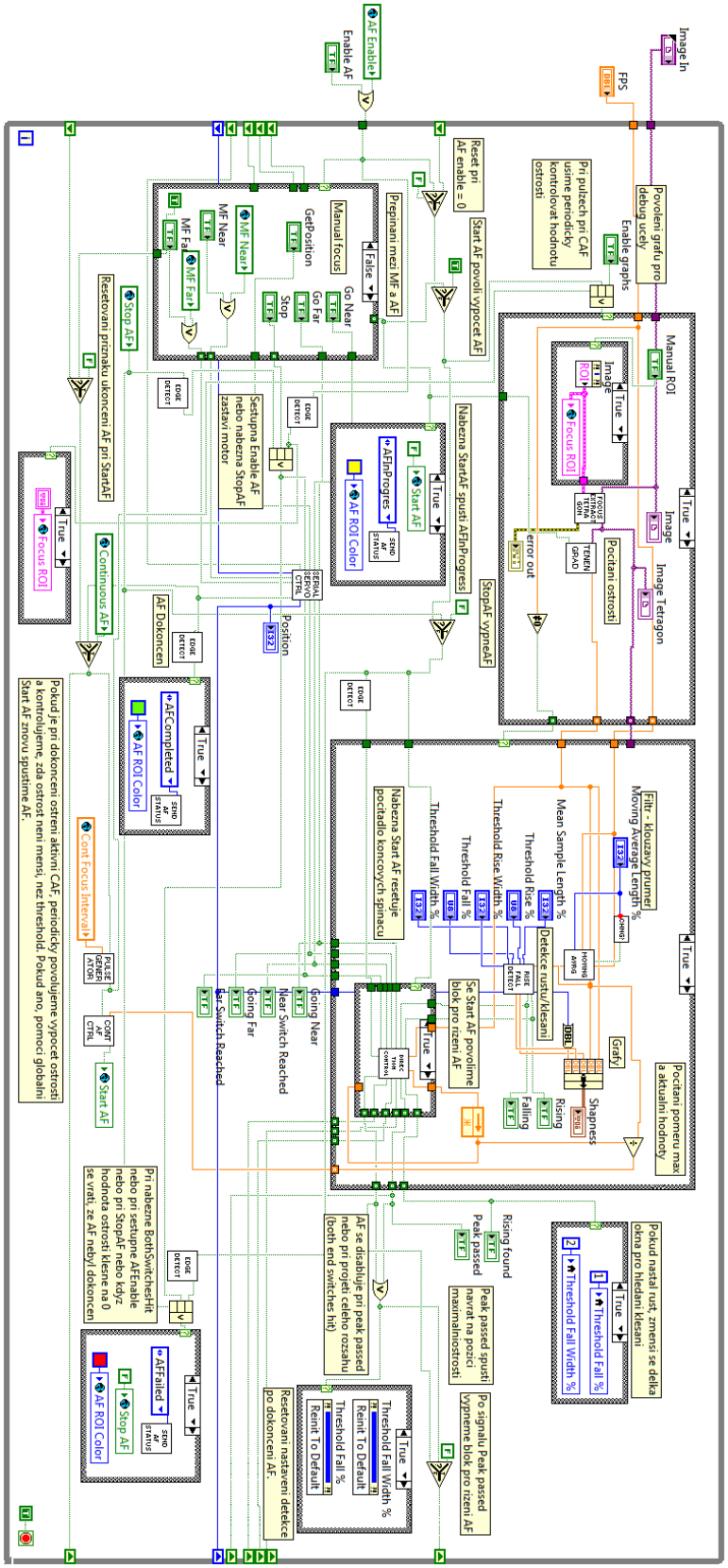
Obrázek D.4.: Průběh funkce ostrosti bez kompenzace vůle převodovky motoru.

PŘÍLOHA

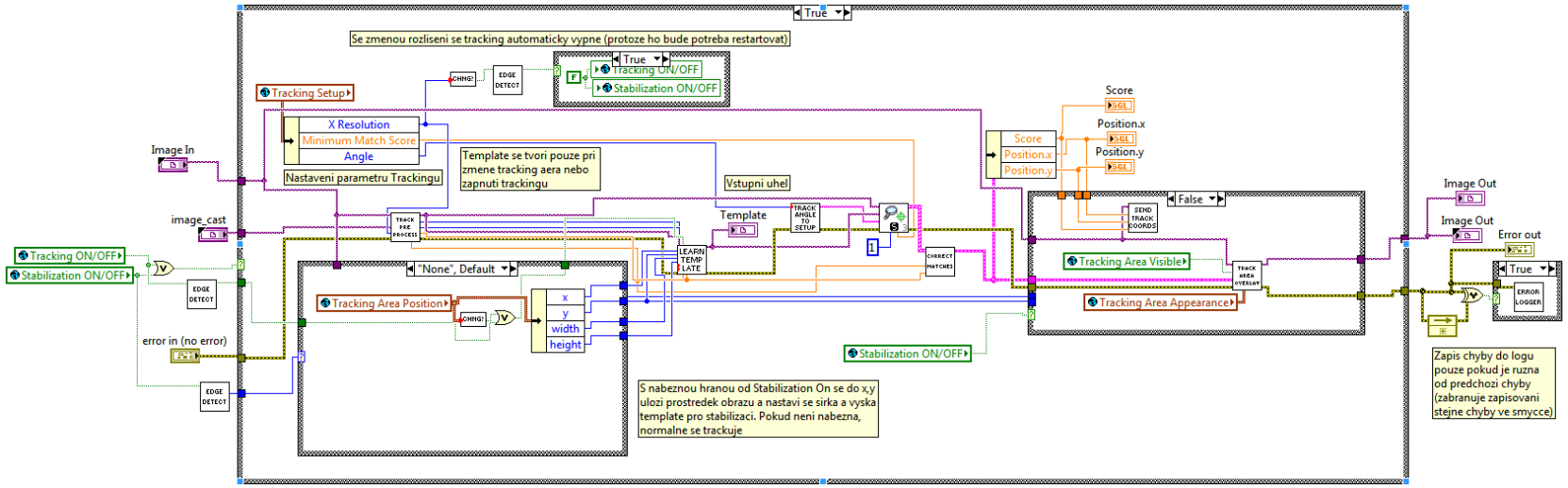
E

UKÁZKY ZDROJOVÉHO KÓDU

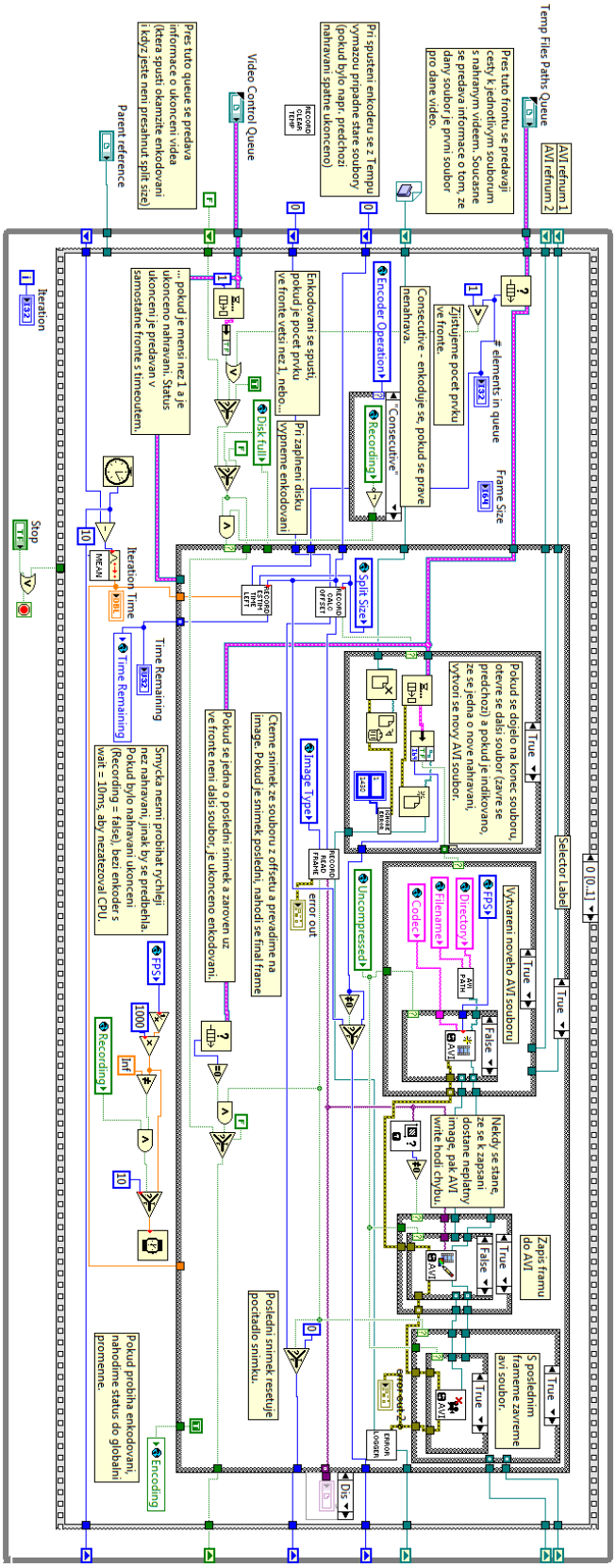
Na obrázcích E.1, E.2 aE.3 jsou pro ilustraci znázorněny zdrojové kódy některých implementovaných funkcí v programovacím prostředí LabVIEW. Uvedené zdrojové kódy obsahují vyšší, zastřešující funkce daného algoritmu – v jednotlivých blocích obsahují mnoho dalších funkcí.



Obrázek E.1.: Ukázka LabVIEW zdrojového kódu hlavního VI funkce pro automatické zaostřování kamery.



Obrázek E.2.: Ukázka LabVIEW zdrojového kódu hlavního VI funkce pro sledování objektu v obraze.



Obrázek E.3.: Ukázka LabVIEW zdrojového kódu hlavního VI funkce pro záznam videosignálu.

PŘÍLOHA

F

SEZNAM PŘÍKAZŮ PRO OVLÁDÁNÍ KROKOVÉHO MOTORU

Každý příkaz odeslaný vývojové desce Arduino musí být ukončen znakem `0x0a`, stejně tak každý příkaz odeslaný vývojovou deskou je ukončen znakem `0x0a`. V případě požadavku na pohyb, který nelze vykonat, Arduino odešle zprávu `Povel_k_pohybu:ERR`.

Povely odesílané z PC na vývojovou desku Arduino

`Ver`

- Vrátí verzi programu pro řízení krokového motoru.
- Odpověď: `Ver:cislo_verze`.

`MoveHSfwd`

- Kontinuální pohyb dopředu.
- Odpověď: `MoveHSfwd:OK`.

`MoveHSbwd`

- Kontinuální pohyb vzad.
- Odpověď: `MoveHSbwd:OK`.

`MoveStop`

- Zastavení kontinuálního pohybu.

- Odpověď: `Pos:soucasna_poloha_motoru`.

`MoveHSfwdStep`

- Pohyb motoru vpřed o 1 pulz.
- Odpověď: `Pos:soucasna_poloha_motoru`.

`MoveHSbwdStep`

- Pohyb motoru vzad o 1 pulz.
- Odpověď: `Pos:soucasna_poloha_motoru`.

`MoveHSfwdStepCount:X`

- Pohyb motoru o X pulzů vpřed.
- Odpověď: `Pos:soucasna_poloha_motoru`.

`MoveHSbwdStepCount:X`

- Pohyb motoru o X pulzů vzad.
- Odpověď: `Pos:soucasna_poloha_motoru`.

`GetPos`

- Vyčtení aktuální pozice motoru.
- Odpověď:
  - ◇ `Pos:soucasna_poloha_motoru`.
  - ◇ V případě, že posun objektivu dosáhl pozice koncového spínače: `EndSwitchX:ON`, kde `X` je číslo koncového spínače.

`ZeroPos`

- Nulování pozice motoru.
- Odpověď: `ZeroPos:OK`.

`GetEndSwitch`

- Vyčtení stavu koncového spínače.
- Odpověď:
  - ◇ `EndSwitchX:ON`, kde `X` je číslo koncového spínače.
  - ◇ `EndSwitch:NONE`, v případě, že se posun objektivu nenachází na pozici žádného z koncových spínačů.

Zprávy samostatně odesílané vývojovou deskou Arduino do PC

`EndSwitch1:ON`

- Zpráva je odeslána při najetí na koncový spínač číslo 1.

`EndSwitch2:ON`

- Zpráva je odeslána při najetí na koncový spínač číslo 2.

`Err:BadCommand`

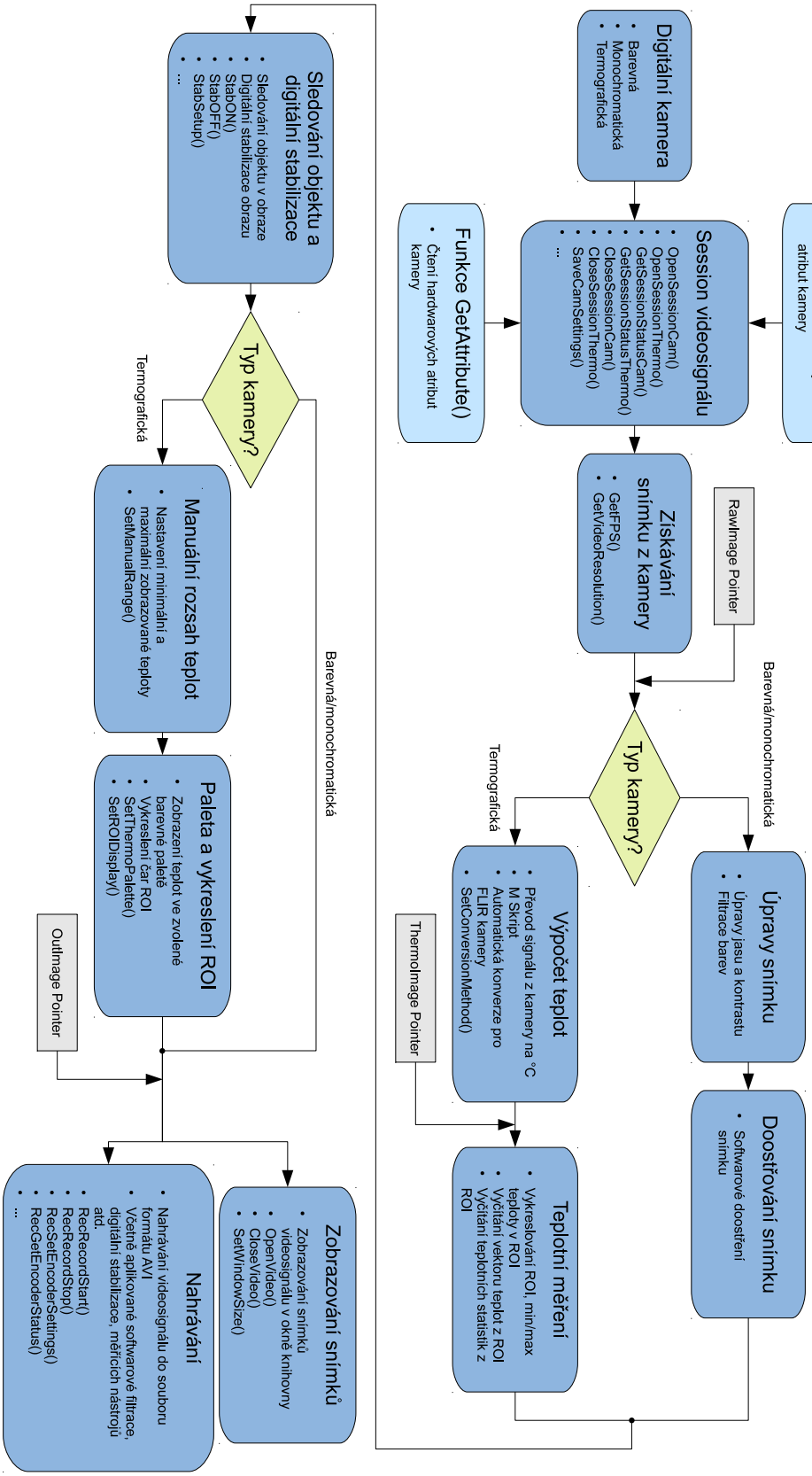
- Odpověď při nesprávném povelu.

PŘÍLOHA

G

BLOKOVÉ SCHÉMA KNIHOVNY

Na obrázku G.1 je znázorněno blokové schéma průchodu snímku procesem knihovny s příklady funkcí, které jednotlivé bloky realizují. Detaily uvedených funkcí jsou v příloze H.



Obrázek G.1.: Blokové schéma průchodu snímku procesem knihovny.

PŘÍLOHA

H

DOKUMENTACE KNIHOVNY

V této příloze je uveden výňatek z manuálu k DLL knihovně obsahující podrobný popis všech implementovaných funkcí. Všechny níže uvedené deklarace funkcí jsou zapsány ve formátu, v jakém jsou funkce uváděny v NI LabVIEW Application Builder při sestavování DLL knihovny. V různých programovacích jazycích mohou mít zejména názvy datových typů jiný formát.

Kompletní manuál ke knihovně obsahující další informace o instalaci, licencování, a obecném používání knihovny je ke dostupný ke stažení na webové stránce `http://www.workswell.cz/smartvision-sdk-library/`.

H.1. Obecný popis DLL funkcí a záznam chyb v logu

Všechny DLL funkce kromě **`void`** funkcí vrací hodnotu 1 (v dokumentaci označena jako „Return“), pokud jsou úspěšně dokončeny. Pokud však nastala chyba, vrací hodnotu 0 nebo –1 (typicky u getter funkcí), dle typu chyby. Veškeré chyby, které při běhu knihovny nastaly, jsou zapsány do tzv. error logu, uloženého v textovém souboru „`error_log.txt`“, který je vytvořen ve stejné cestě jako spuštěná DLL knihovna.

H.2. Funkce pro připojení ke kameře, ovládání okna knihovny a nastavení videosignálu

H.2.1. Spuštění session a ukončení session

Pro ovládání barevné (monochromatické) kamery je nutné nejprve spustit tzv. Session pomocí funkce `OpenSessionCam`. V případě termografické kamery je možné Session spustit pomocí funkce `OpenSessionThermo`. Parametrem těchto funkcí je textový identifikátor rozhraní kamery, který lze zjistit v poli „Camera Interface“ v pomocném programu Camera Explorer. Ote-

vření Session chvíli trvá, a proto je nutné před další prací s kamerou periodicky provádět dotazování na stav Session pomocí funkcí `GetSessionStatusCam`, případně `GetSessionStatusThermo`. Session se nepodaří spustit, pokud již ke kameře přistupuje jiný proces.

Popis příslušných DLL funkcí:

`void` `OpenSessionCam(`**`char`** `cameraInterface[])`

- Otevírá Session k barevné/monochromatické kameře.
- **`char`** `cameraInterface[]` – řetězec obsahující název rozhraní kamery získaný pomocí aplikace Camera Explorer.

`void` `OpenSessionThermo(`**`char`** `thermoInterface[])`

- Otevírá Session k termografické kameře.
- **`char`** `thermoInterface[]` – řetězec obsahující název rozhraní kamery získaný pomocí aplikace Camera Explorer.

`void` `CloseSessionCam()`

- Ukončuje Session barevné/monochromatické kamery.

`void` `CloseSessionThermo()`

- Ukončuje Session termografické kamery.

`int8_t` `GetSessionStatusCam()`

- Zjišťuje, jestli běží Session barevné/monochromatické kamery.
- Return **`int8_t`**:
  - ◇ 0 = Session není spuštěná.
  - ◇ 1 = Session je spuštěná.

`int8_t` `GetSessionStatusThermo()`

- Zjišťuje, jestli běží Session termografické kamery.
- Return **`int8_t`**:
  - ◇ 0 = Session není spuštěná.
  - ◇ 1 = Session je spuštěná.

H.2.2. Otevření a zavření okna s videem

Po spuštění Session je možné otevřít okno s videosignálem pomocí funkce `OpenVideo`. Dále už lze libovolně volat ostatní funkce z knihovny. Po ukončení práce je nutné zavřít okno s videosignálem funkce `CloseVideo` a ukončit Session pomocí funkce `CloseSessionCam`, případně `CloseSessionThermo`. Session je nutné ukončit před ukončením koncové aplikace, ze které se volá DLL, jinak v operační paměti zůstane bežící proces příslušící spuštěné Session.

Popis příslušných DLL funkcí:

`void` `OpenVideo()`

- Otevírá okno s videosignálem. Okno je potom nutné přesunout na obrazovce pomocí systémových Windows API funkcí (knihovna „`User32.dll`“) na souřadnice v koncové aplikaci.

`void` `CloseVideo()`

- Zavírá okno s videosignálem.

`int8_t` `SetWindowSize(`**`int16_t`** `width,` **`int16_t`** `height)`

- Nastavuje velikost okna s videosignálem. Okno je vždy umístěno na pozici (0,0).
- **`int16_t`** `width` – horizontální velikost okna v px.
- **`int16_t`** `height` – vertikální velikost okna v px.
- Return **`int8_t`**:
  - ◇ 1 = nastavení úspěšné.
  - ◇ 0 = nastavení neúspěšné/chyba.

H.2.3. Funkce pro ovládání a čtení parametrů digitální kamery

Veškeré hardwarové parametry (atributy) digitální kamery nebo framegrabberu je možné ovládat pomocí funkce `SetAttribute`. Tyto parametry jsou reprezentovány názvem XML uzlu, který používá ovladač GigE rozhraní, případně dalších rozhraní (DirectShow apod.). Čtení hardwarových nastavení kamery je realizováno funkcí `GetAttribute`, která z parametru vyčte jeho aktuální hodnotu. Rozšířenou verzí funkce `GetAttribute` je funkce `GetAttributeInfo`, která z parametru vyčte další, doplňkové informace.

Popis příslušných DLL funkcí:

```
int8_t SetAttribute(char interface[], char
   activeAttribute[], double valueDBL, char
   valueString[])
```

- Funkce pro nastavování hardwarových parametrů kamery.
- **char** `interface[]` – řetězec obsahující název rozhraní kamery získaný pomocí aplikace Camera Explorer.
- **char** `activeAttribute[]` – název XML uzlu vyčtený ze souboru vygenerovaného pomocí aplikace Camera Explorer (bez složených závorek, ovšem včetně oddělovacích dvojteček).
- **double** `valueDBL` – číselná hodnota atributu. GiGE GeICam standard specifikuje tyto datové typy: **uint32_t**, **int64_t**, **double**, **string**, **enum**, **boolean**, **command**. Parametr `valueDBL` nastavuje hodnotu datových typů **uint32_t**, **int64_t**, **double**, **boolean** a **command** (hodnoty 0 a 1 odpovídají logickým hodnotám `false` a `true`). Je nutné si pohlídat platný rozsah atributu, uložený opět v textovém souboru vygenerovaném aplikací Camera Explorer.
- **char** `valueString[]` – hodnota řetězce atributu. Zahrnuje datové typy **string** a **enum**. Pro enumerátory jsou v textovém souboru uloženy všechny platné hodnoty daného enumerátoru a tyto se předávají funkci jako jeden řetězec.
- Return **int8_t**:
  - ◇ 1 = nastavení úspěšné.
  - ◇ 0 = nastavení neúspěšné/chyba.

```
int8_t GetAttribute(char interface[], char
   activeAttribute[], double *valueDBL, char
   valueString[], Int32 lengthIn)
```

- Vyčte hodnotu hardwarového atributu kamery.
- **char** `interface[]` – řetězec obsahující název rozhraní kamery získaný pomocí aplikace Camera Explorer.
- **char** `activeAttribute[]` – název XML uzlu vyčtený ze souboru vygenerovaného pomocí aplikace Camera Explorer (bez složených závorek, ovšem včetně oddělovacích dvojteček).

- **double** *valueDBL – pointer na proměnnou typu **double**, do které se zapíše číselná hodnota atributu. Proměnná zastupuje datové typy **uint32_t**, **int64_t**, **double**, **boolean** a **command**. Hodnoty 0 a 1 odpovídají logickým hodnotám false a true.
- **char** valueString[] – hodnota řetězce atributu. Zahrnuje datové typy **string** a **enum**.
- **int32_t** lengthIn – očekávaná délka obdrženého řetězce (kvůli vyhrazení paměti pro výstupní řetězec valueString[]) .
- Return **int8_t**:
  - ◇ 1 = čtení úspěšné.
  - ◇ 0 = čtení neúspěšné/chyba.

```
int8_t GetAttributeInfo(char interface[], char
   activeAttribute[], int8_t *type, int8_t *readable,
   int8_t *writable, double *minimum, double *maximum,
   double *valueDBL, char valueString[], int32_t len)
```

- Vyčte hodnotu hardwarového atributu kamery a další doplňkové informace o atributu.
- **char** interface[] – řetězec obsahující název rozhraní kamery získaný pomocí aplikace Camera Explorer.
- **char** activeAttribute[] – název XML uzlu vyčtený ze souboru vygenerovaného pomocí aplikace Camera Explorer (bez složených závorek, ovšem včetně oddělovacích dvojteček).
- **int8_t** *type – pointer na proměnnou, do které se zapíše datový typ daného atributu (0 = **uint32_t**, 1 = **int64_t**, 2 = **double**, 3 = **string**, 4 = **enum**, 5 = **boolean**, 6 = **command**).
- **int8_t** *readable – je možné atribut číst? Hodnoty 0 a 1 odpovídají hodnotám false a true.
- **int8_t** *writable – je možné atribut zapisovat?
- **double** *minimum – pointer na proměnnou, do které se zapíše minimální hodnota rozsahu atributu.
- **double** *maximum – pointer na proměnnou, do které se zapíše maximální hodnota rozsahu atributu.

- **double** `*valueDBL` – pointer na proměnnou typu **double**, do které se zapíše číselná hodnota atributu. Proměnná zastupuje datové typy **uint32_t**, **int64_t**, **double**, **boolean** a **command**. Hodnoty 0 a 1 odpovídají logickým hodnotám `false` a `true`.
- **char** `valueString[]` – hodnota řetězce atributu. Zahrnuje datové typy **string** a **enum**.
- **int32_t** `lengthIn` – očekávaná délka obdrženého řetězce (kvůli vyhrazení paměti pro výstupní řetězec `valueString[]`).
- Return **int8_t**:
  - ◇ 1 = čtení úspěšné.
  - ◇ 0 = čtení neúspěšné/chyba.

H.2.4. Uložení parametrů a automatické načtení v novém Session

V některých případech je nutné přenastavit výchozí parametry digitální kamery (např. rozlišení videosignálu). Funkcí `SetAttribute` se příslušné parametry nastaví pro aktuální Session, ale v případě vypnutí a opětovného zapnutí napájení kamery dojde k načtení výchozích hodnot. Funkce `SaveCamSettings` uloží aktuálně nastavené hodnoty jako výchozí na pevný disk počítače. Tyto hodnoty jsou následně načteny při opětovném připojení kamery.

```
int8_t SaveCamSettings(char camInterface[])
```

- Uloží aktuální nastavení všech atributů kamery jako výchozí. Při zavření a novém spuštění session jsou načtena tato nastavení.
- **char** `camInterface[]` – řetězec obsahující název rozhraní kamery získaný pomocí aplikace Camera Explorer.
- Return **int8_t**:
  - ◇ 1 = nastavení úspěšné.
  - ◇ 0 = nastavení neúspěšné/chyba.

H.2.5. Zjištění snímkové frekvence a rozlišení obrazu

Následující funkce jsou vhodné k získání přehledu o snímkové frekvenci a rozlišení videosignálu. Tyto informace jsou potřeba například k volání funkce `MakeSnap` (viz níže).

```
void GetFPS(double *acquiredFPS, double *processedFPS)
```

- Zjistí počet snímků za vteřinu videosignálu.

- **double** `*acquiredFPS` – počet snímků za sekundu získaných z kamery před zpracováním videosignálu.
- **double** `*processedFPS` – počet snímků za sekundu po zpracování videosignálu.

```
void GetVideoResolution(int16_t *resolutionX, int16_t
   *resolutionY)
```

- Zjistí aktuální rozlišení videa.
- **int16_t** `*resolutionX` – pointer na proměnnou datového typu **int16_t**, do které se zapíše horizontální rozlišení videosignálu.
- **int16_t** `*resolutionY` – pointer na proměnnou datového typu **int16_t**, do které se zapíše vertikální rozlišení videosignálu.

H.2.6. Přístup k aktuálním snímkům videosignálu

Prvním způsobem zachycení aktuálního snímku z videosignálu je funkce `MakeSnap`.

```
void MakeSnap(uint8_t buffer[], int32_t length)
```

- Odešle aktuální snímek videosignálu do bufferu (pole). Při volání funkce je nutné parametrem `length` zadat požadovanou délku čteného bufferu.
- **int32_t** `length` – délka čteného bufferu (vypočítá se jako horizontální × vertikální rozlišení × 3 (3 barvy), rozlišení videosignálu lze rychle získat funkcí `GetVideoResolution`).
- **uint8_t** `buffer` – pointer na jednorozměrné pole datového typu **uint8_t** délky `length`, do kterého budou uložena data posledního snímku. Data jsou do pole uložena po jednotlivých pixelech zleva doprava a dále po jednotlivých řádcích snímku. Každé po sobě jdoucí tři prvky pole odpovídají barevným složkám každého pixelu (R, G, B). Tedy např. pro matici pixelů $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ bude mít pole tvar: {R1, G1, B1, R2, G2, B2, R3, G3, B3, R4, G4, B4}.

Pro přístup k aktuálním nezměněným snímkům z kamery lze použít i funkci `GetRawImagePointer`, která vrátí pointer na blok operační paměti, kde je uložen aktuální nezpracovaný snímek z kamery. Jednorozměrné pole bytů v paměti obsahuje čistá obrazová data snímku. Formát dat odpovídá datovému typu pixelů snímku. Pro rekonstrukci obrazu je nutné znát i další návratové parametry funkce `GetRawImagePointer`.

První řádek snímku zrekonstruujeme takto: začneme u `pixelPointerOut` a pokračujeme po blocích odpovídajících velikosti `pixelSize` až do `xResolution` (poslední pixel prvního řádku). První pixel druhého řádku získáme takto: `pixelPointerOut + lineWidth` a opět můžeme postupovat po blocích velikosti `pixelSize`. Tímto způsobem postupujeme po řádcích až do `yResolution` (vertikální rozlišení = index posledního řádku).

Černobílý snímek má 8bitovou hodnotu každého pixelu uloženou v jednom bytu. Každý byte pole z operační paměti tedy odpovídá jednomu pixelu. Radiometrický snímek má 16bitovou hodnotu každého pixelu uloženou ve dvou bytech. Hodnota pixelu potom odpovídá dvěma po sobě jdoucím bytům pole z paměti. Barevný snímek má 32bitovou hodnotu každého pixelu uloženou ve čtyřech bytech. Hodnota pixelu potom odpovídá čtyřem po sobě jdoucím bytům pole z paměti, přičemž první, druhý a třetí odpovídají červené, zelené a modré složce pixelu a čtvrtý byte je nulový.

```
int8_t GetRawImagePointer(uint64_t *pixelPointerOut,
   int32_t *bufferSize, int32_t *pixelSize, int32_t
   *lineWidth, int32_t *xResolution, int32_t
   *yResolution)
```

- **`uint64_t`** `pixelPointerOut` – pointer na první pixel snímku v operační paměti (ve snímku vlevo nahoře).
- **`int32_t`** `pixelSize` – počet bytů obsahujících hodnotu jednoho pixelu.
- **`int32_t`** `bufferSize` – celkový počet bytů snímku v paměti.
- **`int32_t`** `lineWidth` – délka jednoho řádku snímku v paměti (včetně pomocných obrazových pixelů).
- **`int32_t`** `xResolution` – skutečné horizontální rozlišení snímku.
- **`int32_t`** `yResolution` – skutečné vertikální rozlišení snímku.

Pro přístup k aktuálním zpracovaným obrazovým datům z kamery lze použít funkci `GetOutImagePointer`. Tato funkce vrátí pointer na blok paměti, kde je uložen aktuální zpracovaný snímek z kamery (tedy snímek po průchodu softwarovými filtry, digitální stabilizací apod.). Funkce funguje totožně jako funkce `GetRawImagePointer`, bližší popis je tedy stejný (viz výše).

```
int8_t GetOutImagePointer(uint64_t *pixelPointerOut,
   int32_t *bufferSize, int32_t *pixelSize, int32_t
   *lineWidth, int32_t *xResolution, int32_t
   *yResolution)
```

- **uint64_t** pixelPointerOut – pointer na první pixel snímku v operační paměti (ve snímku vlevo nahoře).
- **int32_t** pixelSize – počet bytů obsahujících hodnotu jednoho pixelu.
- **int32_t** bufferSize – celkový počet bytů snímku v paměti.
- **int32_t** lineWidth – délka jednoho řádku snímku v paměti (včetně pomocných obrazových pixelů).
- **int32_t** xResolution – skutečné horizontální rozlišení snímku.
- **int32_t** yResolution – skutečné vertikální rozlišení snímku.

Pro přístup k aktuální matici teplotních dat z termografické kamery slouží funkce GetThermoImagePointer. Tato funkce vrátí pointer na blok paměti, kde jsou uložena aktuální data z termografické kamery převedená teplotní konverzí na stupně Celsia. Data jsou uložena jako buffer dat reprezentující dvourozměrné pole. Teplotní data jsou datového typu **float** a každý pixel je tedy uložen jako 32bitová hodnota ve čtyřech bytech. Teplotní data nejsou ovlivněna nakreslenými měřícími nástroji, digitální stabilizací atd. Funkce GetThermoImagePointer funguje totožně jako funkce GetRawImagePointer, bližší popis je tedy stejný (viz výše).

```
int8_t GetThermoImagePointer(uint64_t *pixelPointerOut,
   int32_t *bufferSize, int32_t *pixelSize, int32_t
   *lineWidth, int32_t *xResolution, int32_t
   *yResolution)
```

- **uint64_t** pixelPointerOut – pointer na první pixel teplotního snímku v operační paměti (ve snímku vlevo nahoře).
- **int32_t** pixelSize – počet bytů obsahujících hodnotu jednoho pixelu.
- **int32_t** bufferSize – celkový počet bytů snímku v paměti.
- **int32_t** lineWidth – délka jednoho řádku snímku v paměti (včetně pomocných obrazových pixelů).
- **int32_t** xResolution – skutečné horizontální rozlišení teplotního snímku.
- **int32_t** yResolution – skutečné vertikální rozlišení teplotního snímku.

H.3. Funkce pro zpracování a úpravy snímků

`void SetSwBrightness(int16_t nVal)`

- Nastavení hodnoty jasu, která bude přičtena ke všem pixelům snímku.
- **`int16_t`** `nVal` – hodnota jasu.
  - Rozsah –255 až 255.

`int16_t GetSwBrightness()`

- Získání aktuální hodnoty jasu, který je přičítán k pixelům snímku.
- Return **`int16_t`**: hodnota jasu
  - Rozsah –255 až 255, výchozí hodnota 0.

`void SetSwContrast(uint8_t nMode, int16_t nVal)`

- Funkce pro zvyšování a snižování kontrastu snímků na požadovanou úroveň.
- **`uint8_t`** `nMode` – metoda úpravy kontrastu. Hodnota 0 nastavuje transformaci pomocí lineární křivky, hodnota 1 nastavuje transformaci pomocí nelineární sigmoid funkce.
- **`int16_t`** `nVal` – hodnota kontrastu.
  - rozsah –100 až 100. Záporné hodnoty snižují a kladné zvyšují kontrast.

`int16_t GetSwContrast(uint8_t *nMode)`

- Získání aktuální hodnoty kontrastu.
- **`uint8_t`** `*nMode` – pointer na proměnnou, do které se zapíše metoda úpravy kontrastu.
- Return **`int16_t`**: hodnota jasu
  - Rozsah –100 až 100, výchozí hodnota 0.

`void SetSwSharpness(int8_t nSharp)`

- Nastavení doostření obrazu na požadovanou úroveň, rozsah 0 až 100
- **`int8_t`** `nSharp` – hodnota doostření.

- ◇ Rozsah 0 až 100, výchozí hodnota 0 = doostřování vypnuto.

```
int8_t GetSwSharpness(void)
```

- Získání aktuální hodnoty doostření obrazu.
- Return **int8_t**: hodnota doostření obrazu
  - ◇ Rozsah 0 až 100, výchozí hodnota 0 = doostřování vypnuto.

```
void SetSwColorTransformationValueRGB(uint8_t nMode,
   int16_t nR, int16_t nG, int16_t nB)
```

- Nastavení hodnot jasu, které budou přičteny k barevným složkám všech pixelů snímku.
- **uint8_t** nMode – metoda úpravy barevnosti snímku. Hodnoty 0 a 1 nastavují úpravu barev bez a se zachováním zdánlivého jasu celého snímku.
- **int16_t** nR – hodnota, která bude přičtena k červené složce snímku.
- **int16_t** nG – hodnota, která bude přičtena k zelené složce snímku.
- **int16_t** nB – hodnota, která bude přičtena k modré složce snímku.
- Rozsah parametrů jasu je –255 až 255.

```
void GetSwColorTransformationValueRGB(uint8_t *nMode,
   int16_t *nR, int16_t *nG, int16_t *nB)
```

- Získání hodnot jasu, které jsou přičítány k barevným složkám všech pixelů snímku. Poslední tři parametry funkce jsou pointery na proměnné typu **int16_t**, do kterých budou uloženy hodnoty přičítaných jasů.
- **uint8_t** nMode – pointer na proměnnou, do které se zapíše metoda úpravy barevnosti snímku.
- **int16_t** nR – pointer na proměnnou, do které se zapíše hodnota jasu, která je přičítána k červené složce snímku.
- **int16_t** nG – pointer na proměnnou, do které se zapíše hodnota jasu, která je přičítána k zelené složce snímku.
- **int16_t** nB – pointer na proměnnou, do které se zapíše hodnota jasu, která je přičítána k modré složce snímku.

```
void SetSwPolynomTransformation(double fConstX3, double
   fConstX2, double fConstX1, double fConstX0)
```

- Nastavení polynomu pro transformaci jasu barevných složek pixelů snímku. Parametry zastupují jednotlivé koeficienty polynomu třetího řádu.
- **double** `fConstX3` – hodnota koeficientu třetího řádu.
- **double** `fConstX2` – hodnota koeficientu druhého řádu.
- **double** `fConstX1` – hodnota koeficientu prvního řádu.
- **double** `fConstX0` – hodnota koeficientu nultého řádu.
- Rozsah všech parametrů – plný rozsah proměnné datového typu **double**.

```
void GetSwPolynomTransformation(double *fConstX3, double
   *fConstX2, double *fConstX1, double *fConstX0)
```

- Získání koeficientů polynomu pro transformaci jasu barevných složek pixelů snímku. Parametry funkce jsou pointery na proměnné datového typu **double**, do kterých budou uloženy hodnoty koeficientů polynomu 3. řádu.
- **double** `fConstX3` – pointer na proměnnou, do které se zapíše hodnota koeficientu třetího řádu, výchozí hodnota = 0.
- **double** `fConstX2` – pointer na proměnnou, do které se zapíše hodnota koeficientu druhého řádu, výchozí hodnota = 0.
- **double** `fConstX1` – pointer na proměnnou, do které se zapíše hodnota koeficientu prvního řádu, výchozí hodnota = 1.
- **double** `fConstX0` – pointer na proměnnou, do které se zapíše hodnota koeficientu nultého řádu, výchozí hodnota = 0.
- Rozsah všech parametrů – plný rozsah proměnné datového typu **double**.

H.4. Funkce pro ovládání automatického zaostřování kamery

Pro komunikaci s krokovým motorem je nutné nejprve otevřít a nastavit sériový port pomocí funkce `SerialInit`. Knihovna potom ovládá přes sériový port elektroniku pro řízení motoru.

Po ukončení práce se servomotorem nebo pro přístup jiného procesu na sériový port je nutné jej uvolnit pomocí funkce `SerialClose`. Před zahájením automatického ostření je jej nutné nakonfigurovat pomocí funkcí `AFFocusMode`, `AFAreaSize`, `AFAreaWidth`, `AFAreaVisible`, `AFContFocusInterval` a `AFContFocusTreshold`, jinak budou použita výchozí nastavení.

V případě kliknutí uživatele do okna s videosignálem provede knihovna předání souřadnic o této události koncové aplikaci pomocí Windows zprávy. Událost s informací o tomto kliknutí odešle knihovna koncové aplikaci pomocí Windows zprávy systémovou funkcí

```
SendMessage(int hWnd, 0x9000, int wParam, int lParam).
```

Prvním parametrem `hWnd` je reference na hlavní okno nadřazené aplikace (angl. handle), která knihovnu používá, kód zprávy je pevně zvolen jako 0x9000 a parametry `wParam` a `lParam` odpovídají horizontální a vertikální souřadnici v obraze.

Při startu aplikace je nutné knihovně předat handle hlavního okna koncové aplikace pomocí funkce `SetMessageReceiverHandle`, jinak posílání zpráv nebude funkční.

Aplikace spustí zaostřování zavoláním funkce `StartAF` s příslušnými souřadnicemi pro zaostření. Knihovna se snaží ve scéně najít místo s nejvyšší ostrosti, na které potom pomocí krokového motoru zaostří objektiv. Pokud je scéna málo kontrastní, nebo neobsahuje dostatečně ostrý objekt, zaostřování selže – typicky pokud se prohledá celý rozsah objektivu (sepnou postupně oba koncové spínače mechanismu objektivu), je signalizováno, že se zaostření nezdařilo.

Knihovna o průběhu zaostřování informuje nadřazenou aplikaci opět pomocí Windows zpráv. Informace o událostech zahájení, dokončení či poruše ostření se odesílají systémovou funkcí

```
SendMessage(int hWnd, 0x9002, int wParam, int lParam).
```

Prvním parametrem `hWnd` je opět handle hlavního okna nadřazené aplikace. Parametr `wParam` obsahuje informaci o události zaostřování:

- `wParam = 0` – automatické zaostřování bylo zahájeno,
- `wParam = 1` – automatické zaostřování bylo dokončeno,
- `wParam = 2` – automatické zaostřování selhalo.

Popis příslušných DLL funkcí:

```
void SetMessageReceiverHandle(int32_t hWnd)
```

- Předání reference (handle) na okno koncové aplikace knihovně.
- **`int32_t`** `hWnd` – handle na hlavní okno koncové aplikace.

```
uint8_t SerialInit(char COMPort[], uint32_t baudRate,
   uint16_t dataBits, uint16_t stopBits, uint16_t parity)
```

- Provede inicializaci sériového portu.
- **`char`** `COMPort[]` – řetězec obsahující název sériového portu (např. „COM1“).
- **`uint32_t`** `baudRate` – rychlost sériového portu.
- **`uint16_t`** `dataBits` – počet datových bitů.
- **`uint16_t`** `stopBits` – počet stop bitů.
  - ◇ 10 = 1 stop bit.
  - ◇ 15 = 1,5 stop bit.
  - ◇ 20 = 2 stop bity.
- **`uint16_t`** `parity` – parita.
  - ◇ 0 = žádná parita.
  - ◇ 1 = lichá.
  - ◇ 2 = sudá.
- Return **`uint8_t`**:
  - ◇ 1 = otevření úspěšné.
  - ◇ 0 = otevření neúspěšné/chyba.

```
uint8_t GetSerialSetup(uint32_t *baudRate, uint16_t
   *dataBits, uint16_t *stopBits, Enum *parity)
```

- Zjistí aktuální nastavení sériového portu.
- **`uint32_t`** `*baudRate` – pointer na proměnnou, do které se zapíše rychlost sériového portu.
- **`uint16_t`** `*dataBits` – pointer na proměnnou, do které se zapíše počet datových bitů.
- **`uint16_t`** `*stopBits` – pointer na proměnnou, do které se zapíše počet stop bitů.
  - ◇ 10 = 1 stop bit.
  - ◇ 15 = 1,5 stop bit.
  - ◇ 20 = 2 stop bity.
- **`uint16_t`** `*parity` – pointer na proměnnou, do které se zapíše parita.
  - ◇ 0 = žádná parita.

  - 1 = lichá.
  - 2 = sudá.

- Return **`uint8_t`** :
  - 1 = vyčtení úspěšné.
  - 0 = vyčtení neúspěšné/chyba.

`void` `AFBackslashCorrection(`**`int32_t`** `correctionValue)`

- Nastavuje parametr pro korekci vůle převodovky.
- **`int32_t`** `correctionValue` – korekce vůle převodovky v pulzech motoru umořených na vymezení vůle.
  - Výchozí hodnota = 450.

`void` `AFFocusMode(`**`uint8_t`** `nMode)`

- Volba režimu ostření.
- **`uint8_t`** `nMode` – Režim ostření.
  - 0 = manuální ostření.
  - 1 = automatický jednorázový.
  - 2 = automatický kontinuální (výchozí hodnota).

`void` `AFAreaSize(`**`uint16_t`** `width,` **`uint16_t`** `height)`

- Nastavuje rozměr oblasti, na kterou bude zaostřováno.
- **`uint16_t`** `width` – šířka oblasti, na kterou bude zaostřováno, v pixelech.
- **`uint16_t`** `height` – výška oblasti, na kterou bude zaostřováno, v pixelech.
- Rozsah obou parametrů = 0 až 65535, výchozí hodnoty: `width` = 100, `height` = 75.

`void` `AFAreaWidth(`**`uint8_t`** `nWidth)`

- Nastavení tloušťky rámečku oblasti automatického zaostřování.
- **`uint8_t`** `nWidth` – tloušťka rámečku.

- ◇ Rozsah = 0 – 255, výchozí hodnota = 1.

```
void AFAreaVisible(uint8_t nVisible)
```

- Zapnutí/vypnutí označení zaostřované oblasti (rámeček vybrané oblasti).
- **uint8_t** `nVisible` – zapnutí/vypnutí označení.
  - ◇ 1 = označení zapnuto (výchozí hodnota).
  - ◇ 0 = označení vypnuto.

```
void AFContFocusInterval(double fTime)
```

- Nastavuje interval mezi jednotlivými analýzami pro ostření v režimu kontinuálního ostření.
- **double** `fTime` – čas v sekundách mezi jednotlivými analýzami ostrosti.
  - ◇ Výchozí hodnota = 5,0.

```
void AFContFocusTreshold(uint8_t nTresh)
```

- Nastavuje práh pro spuštění nového zaostření v kontinuálním režimu ostření.
- **uint8_t** `nTresh` – práh pro spuštění nového zaostření v % ostrosti původního zaostření.
  - ◇ Výchozí hodnota = 40.

```
void StartAF(uint16_t x, uint16_t y)
```

- Spustí zaostření na zadané souřadnice.
- **uint16_t** `x` – horizontální souřadnice videa.
- **uint16_t** `y` – vertikální souřadnice videa.

```
void StopAF()
```

- Provede okamžité zastavení zaostřování.

```
void FocusNear(uint8_t nMove)
```

- Manuální ostření – směr k bližším fokálním vzdálenostem.

- **uint8_t** nMove – spuštění/zastavení pohybu.
  - ◇ 1 = spustit pohyb.
  - ◇ 0 = zastavit pohyb.

```
void FocusFar(uint8_t nMove)
```

- Manuální ostření – směr k vzdálenějším fokálním vzdálenostem.
- **uint8_t** nMove – spuštění/zastavení pohybu.
  - ◇ 1 = spustit pohyb.
  - ◇ 0 = zastavit pohyb.

H.5. Funkce pro sledování objektu v obraze

V případě kliknutí uživatele do okna s videosignálem provede knihovna předání souřadnic do koncové aplikace pomocí Windows zprávy systémovou funkcí

```
SendMessage(int hWnd, 0x9000, int wParam, int lParam).
```

Prvním parametrem hWnd je reference na hlavní okno nadřazené aplikace (angl. handle), která knihovnu používá, kód zprávy je pevně zvolen jako 0x9000 a parametry wParam a lParam odpovídají horizontální a vertikální souřadnici v obraze.

Při startu aplikace je nutné knihovně předat handle hlavního okna koncové aplikace pomocí funkce SetMessageReceiverHandle, jinak posílání zpráv nebude funkční.

Pro spuštění sledování objektu musí koncová aplikace zavolat knihovní funkce SetArea (pro nastavení souřadnic a velikosti sledované oblasti) a TrackON. Sledování se vypne funkcí TrackOFF.

Informace o poloze nalezeného objektu je odesílána koncové aplikaci pomocí Windows zprávy systémovou funkcí

```
SendMessage(int hWnd, 0x9001, int wParam, int lParam).
```

Identifikátor zprávy je pevně zvolen jako 0x9001 a parametry wParam a lParam odpovídají horizontální a vertikální souřadnici sledovaného objektu (relativní vůči středu obrazu). Pokud je nalezený objekt vlevo/vpravo od středu obrazu, horizontální souřadnice má záporné/kladné znaménko. Pokud je objekt dole/nahoře od středu obrazu, vertikální souřadnice má záporné/kladné znaménko.

Sledování je možné před spuštěním nakonfigurovat funkcemi TrackingSetup, TrackedAreaSetColor, TrackingAreaSetWidth, TrackedAreaVisible, TrackingAreaSetWidth, jinak budou použita výchozí nastavení. Zmíněné funkce je možné volat i za běhu programu.

```
void SetMessageReceiverHandle(int32_t hWnd)
```

- Předání reference (handle) na okno koncové aplikace knihovně.
- **int32_t** hWnd – handle na hlavní okno koncové aplikace.

```
void TrackingSetArea(uint16_t x, uint16_t y, uint16_t
   nWidth, uint16_t nHeight)
```

- Nastavení pozice a velikosti oblasti pro sledování. Poloha x = 0, y = 0 odpovídá levému hornímu rohu snímku.
- **uint16_t** x – horizontální souřadnice středu sledované oblasti.
- **uint16_t** y – vertikální souřadnice středu sledované oblasti.
- **uint16_t** nWidth – šířka sledované oblasti (v pixelech).
- **uint16_t** nHeight – výška sledované oblasti (v pixelech).

```
void TrackingSetup(uint16_t xResolution, uint16_t
   sensitivity, uint8_t angle)
```

- Nastavení pokročilých parametrů sledování objektu.
- **uint16_t** xResolution – rozlišení použité pro zmenšení (podvzorkování) snímku, ve kterém se vyhledává sledovaná oblast. Nižší xResolution znamená zrychlení vyhledávání a snížení výpočetní náročnosti za cenu ztráty detailů použitých k rozlišení sledované oblasti. Zadává se pouze horizontální rozlišení, vertikální je dopočítáno tak, aby byl dodržen poměr stran vstupního videosignálu. Při změně rozlišení je automaticky sledování oblasti vypnuto a je nutné jej znovu zapnout funkcí TrackingON().
  - ◇ Rozsah = 0 – 65536, výchozí hodnota = 1024.
- **uint16_t** sensitivity – citlivost vyhledávání sledované oblasti v obrazu. Vyšší citlivost znamená více tolerantní algoritmus sledování, ovšem za cenu možnosti zachytit falešnou oblast. Nižší citlivost je přísnější ve vyhledávání vzoru ve snímku.
  - ◇ Rozsah = 0 – 100, výchozí hodnota = 20.
- **uint8_t** angle – nastavení maximálního úhlu rotace sledované oblasti v prohledávaném obraze. Zásadně ovlivňuje výpočetní náročnost sledování.
  - ◇ Rozsah = 0 – 180, výchozí hodnota = 4.

```
void TrackingON()
```

- Zapne sledování objektu.

```
void TrackingOFF()
```

- Vypne sledování objektu.

```
void TrackingAreaVisible(uint8_t nVisible)
```

- Zapnutí/vypnutí označení sledovaného objektu (rámeček sledované oblasti).
- `uint8_t nVisible` – zapnutí/vypnutí označení
  - ◇ 1 … označení zapnuto (výchozí hodnota).
  - ◇ 0 … označení vypnuto.

```
void TrackingAreaSetColor(uint8_t nR, uint8_t nG,
   uint8_t nB)
```

- Nastavení barvy rámečku.
- `uint8_t nR` – intenzita jasu červené složky barvy rámečku.
- `uint8_t nG` – intenzita jasu zelené složky barvy rámečku.
- `uint8_t nB` – intenzita jasu modré složky barvy rámečku.
  - ◇ Rozsah = 0 – 255, výchozí hodnoty `nR` = 255, `nG` = 0, `nB` = 0.

```
void TrackingAreaSetWidth(uint8_t nWidth)
```

- Nastavení tloušťky rámečku.
- `uint8_t nWidth` – tloušťka rámečku.
  - ◇ Rozsah = 0 – 255, výchozí hodnota = 1.

H.6. Funkce pro ovládání digitální stabilizace obrazu

```
void StabON()
```

- Zapnutí stabilizace.

```
void StabOFF()
```

- Vypnutí stabilizace.

```
void StabSetup(uint16_t xResolution, uint16_t
   sensitivity, uint8_t angle, uint8_t corrRotation,
   uint8_t interpolate)
```

- Nastavení pokročilých parametrů stabilizace.
- **uint16_t** `xResolution` – rozlišení použité pro zmenšení (podvzorkování) snímku, který je použit pro stabilizaci. Nižší `xResolution` znamená zrychlení algoritmu stabilizace a snížení výpočetní náročnosti za cenu ztráty detailů použitých k rozlišení oblasti pro stabilizaci. Zadává se pouze horizontální rozlišení, vertikální je dopočítáno tak, aby byl dodržen poměr stran vstupního obrazu. Při změně rozlišení je automaticky stabilizace vypnuta a je nutné ji znovu zapnout funkcí `StabON()`.
  - ◇ Rozsah = 0 – 65536, výchozí hodnota = 1024.
- **uint16_t** `sensitivity` – citlivost vyhledávání oblasti pro stabilizaci v obrazu. Vyšší citlivost znamená více tolerantní algoritmus stabilizace, ovšem za cenu možnosti zachytit falešnou oblast používanou pro stabilizaci. Nižší citlivost je přísnější ve vyhledávání oblasti v obraze.
  - ◇ Rozsah = 0 – 100, výchozí hodnota = 20.
- **uint8_t** `angle` – nastavení maximálního úhlu rotace oblasti používané pro stabilizaci. Zásadně ovlivňuje výpočetní náročnost digitální stabilizace.
  - ◇ Rozsah = 0 – 180, výchozí hodnota = 4.
- **uint8_t** `corrRotation` – zapnutí/vypnutí stabilizace rotace kamery. Má zásadní vliv na výpočetní náročnost digitální stabilizace.
  - ◇ Hodnota 1 = zapnuto, 0 = vypnuto, výchozí hodnota = 0.
- **uint8_t** `interpolate` – zapnutí/vypnutí interpolace výstupního obrazu při zapnuté stabilizaci rotace kamery. Opět má velký vliv na výpočetní náročnost digitální stabilizace.

- ◇ Hodnota 1 = zapnuto, 0 = vypnuto, výchozí hodnota = 0.

```
void StabShowOrigVideo(uint8_t showOrigVideo)
```

- Demonstrační funkce, která otevře přídavné okno s nestabilizovaným obrazem (pro porovnání stabilizovaného a nestabilizovaného videosignálu).
- **`uint8_t`** `showOrigVideo` – otevře/zavře okno s nestabilizovaným obrazem.
- Hodnota 1 = okno zobrazeno, 0 = okno zavřeno, výchozí hodnota = 0.

H.7. Funkce pro práci s termografickými kamerami

H.7.1. Funkce pro nastavení měření, termálního obrazu a zadávání ROI

```
int8_t SetConversionMethod(int32_t conversionMethod)
```

- Nastavuje způsob převodu výstupního signálu z termografické kamery na stupně Celsia.
- **`int32_t`** `conversionMethod` – metoda převodu signálu.
  - ◇ `0` = žádný převod (výchozí)
  - ◇ `1` = M Skript
  - ◇ `2` = FLIR A5
  - ◇ `3` = FLIR A15
  - ◇ `4` = FLIR A35
  - ◇ `5` = FLIR A65
  - ◇ `6` = FLIR A315
  - ◇ `7` = FLIR A615
- Return **`int8_t`**:
  - ◇ 1 = nastavení způsobu převodu proběhlo úspěšně.
  - ◇ 0 = nastavení způsobu převodu proběhlo neúspěšně.

```
void SetROIToolsVis(int8_t enable)
```

- Nastavuje zobrazení palety s nástroji pro kreslení ROI vedle okna s videosignálem.
- **`int8_t`** `enable` – aktivuje a deaktivuje zobrazení nástrojů pro kreslení ROI.

  - ◇ 0 = paleta vypnuta.
  - ◇ 1 = paleta zapnuta.

```
void SetROITool(int32_t ROITool)
```

- Vybere konkrétní nástroj pro kreslení ROI. Vhodné, pokud není žádoucí zobrazovat paletu s nástroji pro kreslení ROI.
- **`int32_t`** `ROITool` – vybraný nástroj pro kreslení ROI.
  - ◇ 0 = kreslení ROI vypnuto
  - ◇ 1 = bod
  - ◇ 2 = úsečka
  - ◇ 3 = obdélník
  - ◇ 4 = elipsa
  - ◇ 5 = polygon
  - ◇ 6 = lomená čára

```
int8_t SetMeasROI(int32_t ROIType, int32_t points[],
   int32_t len)
```

- Umožňuje zadat ROI programově (bez kreslení) pomocí hraničních bodů oblasti.
- **`int32_t`** `ROIType` – typ ROI.
  - ◇ 0 = ROI vypnuto
  - ◇ 1 = bod
  - ◇ 2 = úsečka
  - ◇ 3 = obdélník
  - ◇ 4 = elipsa
  - ◇ 5 = polygon
  - ◇ 6 = lomená čára
- **`int32_t`** `points[]` – pole souřadnic jednotlivých bodů oblasti. Jednotlivé souřadnice jsou zadávány za sebou. Podle typu oblasti má pole následující formáty:
  - ◇ Bod – souřadnice [x, y].

  - ◇ Úsečka – souřadnice [x,y] počátečního bodu, souřadnice [x, y] koncového bodu (tedy pole tvaru [x1, y1, x2, y2]).
  - ◇ Obdélník – souřadnice [x, y] levého horního rohu, souřadnice [x, y] pravého spodního rohu (tedy pole tvaru [xLH, yLH, xPS, yPS]).
  - ◇ Elipsa – souřadnice přímek tvořících strany obdélníku obklopujícího elipsu [levá strana (horizontální souřadnice), horní strana (vertikální souřadnice), pravá strana (horizontální souřadnice), spodní strana (vertikální souřadnice)], tedy pole tvaru [xL, yH, xP, yS].
  - ◇ Polygon – souřadnice [x, y] všech po sobě jdoucích uzlů. První a poslední bod musejí mít stejné souřadnice. Tedy pole tvaru [x1, y1, x2, y2, . . . , xN, yN, x1, y1]).
  - ◇ Lomená čára – souřadnice [x, y] všech po sobě jdoucích uzlů.

- **`int32_t`** `len` – délka pole bodů.

`void` `SetManualRange(`**`int8_t`** `enable,` **`float`** `min,` **`float`** `max)`

- Nastavuje manuální rozsah zobrazovaných teplot. Ve výchozím stavu je zobrazení termálního obrazu nastaveno na automatický rozsah teplot, takže pokud je pro zobrazení použita např. paleta odstínů šedé (Gray), odpovídá minimální teplotě v obraze černá barva a maximální teplotě bílá barva.
- **`int8_t`** `enable` – vypnout/zapnout manuální rozsah.
- **`float`** `min` – minimální zobrazovaná teplota. Tato teplota bude reprezentována nejnižší teplotou (barvou) použité palety a všechny nižší teploty v obraze budou mít stejnou barvu.
- **`float`** `max` – maximální zobrazovaná teplota. Tato teplota bude reprezentována nejvyšší teplotou (barvou) použité palety a všechny vyšší teploty v obraze budou mít stejnou barvu.

`void` `SetThermoPalette(`**`int8_t`** `palette)`

- Nastavuje paletu pro reprezentaci termografického snímku.
- **`int8_t`** `palette` – typ palety
  - ◇ 0 = Gray
  - ◇ 1 = Gradient
  - ◇ 2 = Rainbow
  - ◇ 3 = Temperature

  - ◇ 4 = Iron

```
void SetMinMaxDiplay(int32_t index, int8_t enable)
```

- Vypne/zapne zobrazování minima a maxima v dané ROI.
- **`int32_t`** `index` – pořadový identifikátor odkazující na požadovanou ROI.
- **`int8_t`** `enable` – vypnout/zapnout zobrazování minima a maxima.
  - ◇ 0 = zobrazení minima a maxima vypnuto.
  - ◇ 1 = zobrazení minima a maxima zapnuto.

```
void SetMinMaxRefresh(float refresh)
```

- Nastaví periodu obnovení výpočtu a zobrazení minima a maxima.
- **`float`** `refresh` – perioda obnovení minima a maxima v sekundách (umožňuje zadat i desetinné číslo).

```
void SetROIDisplay(int32_t index, int8_t enable)
```

- Vypne/zapne zobrazení dané ROI ve snímcích videosignálu.
- **`int32_t`** `index` – pořadový identifikátor odkazující na požadovanou ROI.
- **`int8_t`** `enable` – vypnout/zapnout zobrazování ROI ve snímku.
  - ◇ 0 = zobrazení ROI vypnuto.
  - ◇ 1 = zobrazení ROI zapnuto.

H.7.2. Čtení měřených teplotních dat a statistik

```
int32_t GetMeasDataLength(int32_t index)
```

- Získá délku dat v měřící ROI oblasti.
- **`int32_t`** `index` – pořadový identifikátor odkazující na požadovanou ROI.
- Return **`int32_t`** – délka dat v ROI.

```
void GetMeasData(int32_t index, float appendedArray[],
   int32_t len)
```

- Vyčte vektor teplotních dat z ROI oblasti. Data jsou z oblasti čtena po řádcích zleva doprava. V závislosti na velikosti a typu oblasti může vyčítání trvat několik milisekund. Pro vyčtení dat je nutné předem znát délku dat získanou funkcí `GetMeasDataLength`.
- **`int32_t`** `index` – pořadový identifikátor odkazující na požadovanou ROI.
- **`float`** `appendedArray[]` – pointer na jednorozměrné pole datového typu **`float`** délky parametru `len`, do kterého bude uložen vektor teplotních dat. Data jsou v poli uložena po jednotlivých pixelech zleva doprava a dále po jednotlivých řádcích snímku.
- **`int32_t`** `len` – délka čtených dat získaná funkcí `GetMeasDataLength`.

```
void GetMinMaxAvg(int32_t index, float *maxValue, float
   *minValue, float *avg)
```

- Vyčte minimální, maximální a průměrnou teplotu v ROI.
- **`int32_t`** `index` – pořadový identifikátor odkazující na požadovanou ROI.
- **`float`** `*maxValue` – pointer na proměnnou typu **`float`**, do které se zapíše maximální teplota v ROI.
- **`float`** `*minValue` – pointer na proměnnou typu **`float`**, do které se zapíše minimální teplota v ROI.
- **`float`** `*avg` – pointer na proměnnou typu **`float`**, do které se zapíše průměrná teplota v ROI.

```
void GetMinMaxCoords(int32_t index, float *maxValue,
   float *minValue, int32_t *maxX, int32_t *maxY,
   int32_t *minX, int32_t *minY)
```

- Vyčte minimální a maximální teplotu v ROI včetně souřadnic pixelů s těmito extrémy.
- **`int32_t`** `index` – pořadový identifikátor odkazující na požadovanou ROI.
- **`float`** `*maxVale` – pointer na proměnnou typu **`float`**, do které se zapíše maximální teplota v ROI.
- **`float`** `*minValue` – pointer na proměnnou typu **`float`**, do které se zapíše minimální teplota v ROI.
- **`int32_t`** `*maxX` – pointer na proměnnou typu **`float`**, do které se zapíše horizontální souřadnice maximální teploty v ROI.

- **int32_t** *maxY – pointer na proměnnou typu **float**, do které se zapíše vertikální souřadnice maximální teploty v ROI.
- **int32_t** *minX – pointer na proměnnou typu **float**, do které se zapíše horizontální souřadnice minimální teploty v ROI.
- **int32_t** *minY – pointer na proměnnou typu **float**, do které se zapíše vertikální souřadnice minimální teploty v ROI.

```
void GetAvgVarStd(int32_t index, float *avg, float *var,
    float *std)
```

- Vyčte průměrnou teplotu, rozptyl a směrodatnou odchylku teplot v ROI.
- **int32_t** index – pořadový identifikátor odkazující na požadovanou ROI.
- **float** *avg – pointer na proměnnou typu **float**, do které se zapíše průměrná teplota v ROI.
- **float** *var – pointer na proměnnou typu **float**, do které se zapíše rozptyl teplot v ROI.
- **float** *std – pointer na proměnnou typu **float**, do které se zapíše směrodatná odchylka teplot v ROI.

```
float GetMedian(int32_t index)
```

- Vyčte medián teplot v ROI.
- **int32_t** index – pořadový identifikátor odkazující na požadovanou ROI.
- Return **float** – medián teplot v ROI.

```
float GetRMS(int32_t index)
```

- Vyčte kvadratický průměr teplot v ROI.
- **int32_t** index – pořadový identifikátor odkazující na požadovanou ROI.
- Return **float** – kvadratický průměr teplot v ROI.

H.8. Funkce pro záznam videosignálu

```
void RecGetCodecs(char codecNames[], int32_t len)
```

- Získá seznam kodeků pro konverzi videosignálu dostupných v OS Windows. Je podporován libovolný kodek s rozhraním Video for Windows.
- **`char`** `codecNames[]` – řetězec obsahující dostupné a použitelné kodeky oddělené středníkem. Pro nastavení komprese při nahrávání je nutné zadat jeden z kodeků.
- **`int32_t`** `len` – očekávaná délka obdrženého řetězce (kvůli vyhrazení paměti pro výstupní řetězec **`char`** `codecNames[]`).

```
int8_t RecGetEncoderStatus(uint32_t *timeRemaining)
```

- Získá status enkodéru videa.
- **`uint32_t`** `*timeRemaining` – pointer na proměnnou typu **`uint32_t`**, do které se zapíše odhadovaný zbývající čas pro kompresi videa. Odhad je pouze přibližný.
- Return **`int8_t`**:
  - ◇ 1 = enkódování videosignálu probíhá.
  - ◇ 0 = enkódování videosignálu dokončeno/neprobíhá.

```
void RecSetEncoderSettings(uint16_t xResolution,
   uint16_t yResolution, uint16_t FPSLimit, char codec[])
```

- Nastavení parametrů enkodéru videosignálu. Nastavování parametrů enkodéru nelze měnit v průběhu nahrávání videosignálu a nastavení kodeku ani v průběhu komprese videosignálu.
- **`uint16_t`** `xResolution` – nastavení horizontálního rozlišení nahrávky.
  - ◇ Výchozí hodnota 0 = horizontální rozlišení nahrávky je stejné jako rozlišení vstupního videosignálu.
  - ◇ Pokud je nastaveno `xResolution` $\neq 0$ a `yResolution` $= 0$, vertikální rozlišení nahrávky se automaticky dopočítá tak, aby měla stejný poměr stran jako vstupní videosignál.
- **`uint16_t`** `yResolution` – nastavení vertikálního rozlišení nahrávky.

  - ◇ Výchozí hodnota 0 = vertikální rozlišení nahrávky je stejné jako rozlišení vstupního videosignálu.

- **`uint32_t`** `FPSLimit` – omezení snímkové frekvence nahrávky.
  - ◇ Výchozí hodnota 0 = počet snímků za vteřinu nahrávky je stejný jako aktuální snímaný počet snímků za vteřinu (resp. maximální možný).

- **`char`** `codec[]` – řetězec obsahující název kodeku pro kompresi videosignálu. Názvy dostupných kodeků lze získat funkcí `RecGetCodecs`.
  - ◇ Výchozí hodnota: prázdný řetězec = nahrávání v nekomprimovaném bezztrátovém formátu (ARGB32).

`void` `RecSetEncoderOperation(`**`uint8_t`** `operation)`

- Nastavení chování enkodéru videosignálu. Je možné zvolit mezi průběžnou a postupnou kompresí videosignálu. Při průběžné kompresi enkodér zpracovává videosignál ještě v průběhu nahrávání a šetří tak místo na disku, protože průběžně maže již zpracované dočasné soubory. Při postupné kompresi enkodér začne zpracovávat videosignál až po ukončení nahrávání. Snižuje se tak zátěž pevného disku (není nutné zároveň zapisovat a číst jako v případě průběžné komprese) a tím i případný pokles FPS při nahrávání. Pokud se při postupné kompresi opět spustí nahrávání, je enkodér dočasně pozastaven. Nastavení nelze měnit v průběhu nahrávání videosignálu.

- **`uint8_t`** `operation` – nastavení způsobu komprese.
  - ◇ 1 = postupná komprese (výchozí hodnota).
  - ◇ 0 = průběžná komprese.

`void` `RecRecordStart(`**`char`** `directory[],` **`char`** `filename[])`

- Spuštění nahrávání videosignálu.

- **`char`** `directory[]` – řetězec obsahující cestu k nahrávanému souboru.
  - ◇ Výchozí hodnota: prázdný řetězec = nahrávky se uloží do cesty „`C:\`“.

- **`char`** `filename[]` – řetězec obsahující název nahrávky.
  - ◇ Výchozí hodnota: prázdný řetězec = název souboru je automaticky vygenerován ve tvaru „`dd.MM.rrrr_hh-mm-ss.avi`“ (d = den, M = měsíc, r = rok, h = hodina, m = minuta, s = sekunda).

```
void RecRecordStop()
```

- Ukončení nahrávání.