VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

SOFTWARE PRO METODU SPEKTROSKOPIE LIBS

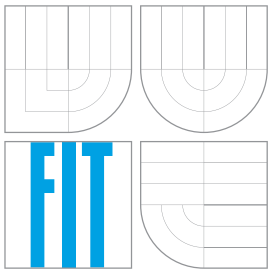
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JANA VYROUBALOVÁ

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

SOFTWARE PRO METODU SPEKTROSKOPIE LIBS

SOFTWARE FOR LIBS SPECTROSCOPY METHOD

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JANA VYROUBALOVÁ
AUTHOR

VEDOUCÍ PRÁCE — Ing. FILIP ORSÁG, Ph.D.
SUPERVISOR

BRNO 2014

Abstrakt

Práce se zabývá tvorbou softwaru pro metodu LIBS (pro elektronickou komoru). V první části se čtenář seznámí se samotnou metodou LIBS a se stručným popisem pracoviště, kde se tento projekt vyvíjí. Dále se věnuji návrhu aplikace pro laboratoře využívající tuto metodu, pak pokračuji hlubším popis implementace softwaru, kde může čtenář například zjistit, jaké byly použity algoritmy pro automatické zaostřování(vyhodnocení ostrosti obrazu a iterativní stanovení polohy kamery). Závěr je soustředěn na zhodnocení návrhu GUI a testování aplikace.

Abstract

This thesis discusses creating of software for the LIBS spectroscopy method (for an electronic chamber). In the first part the reader will learn about the LIBS method and about a short description of a workplace, where the project is being created. Then I write about an application design for laboratories, which are going to use this method. Next I describe an implementation of software, where the reader can find some details about used algorithms for autofocus (an evaluation of image sharpness and an iterative determination of the position of the camera). The end of this work is focused on the assessment of GUI and testing the application.

Klíčová slova

Windows, Qt, Spektroskopie laserem buzeného plazmatu, Autofokus, Q-switch, RGB, Fourierova transformace, RS232

Keywords

Windows, Qt, Laser Induced Breakdown Spectroscopy, Autofocus, Q-switch, RGB, Fourier Transformation, RS232

Citace

Jana Vyroubalová: Software pro metodu spektroskopie LIBS, bakalářská práce, Brno, FIT VUT v Brně, 2014

Software pro metodu spektroskopie LIBS

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením vedoucího bakalářské práce. Veškerá použitá literatura a informační zdroje jsou v práci uvedeny v seznamu literatury, popřípadě citovány s co nejpodrobnějším uvedením zdroje.

.....
Jana Vyroubalová
13. května 2014

Poděkování

V první řadě bych chtěla poděkovat mému vedoucímu za příjemné a pohodové vedení práce a hlavně za to, že se mnou jednal jako s člověkem. Dále děkuji klukům z FSI, kteří mě zasvětili a poskytli mi prostředky pro tvorbu aplikace. A obrovské díky také patří mému příteli Štěpánu Křehlíkovi za obrovskou podporu a pochopení.

Obsah

Kapitola 1

Úvod

Ať už se pohybujeme ve vesmíru, ve zdravotnictví anebo se pohybujeme například v odvětví strojírenství nebo biologie, analýza materiálu je v těchto oblastech nedílnou součástí. Nejenže zjistíme složení takových materiálů, ale dokážeme z toho také odvodit jejich vlastnosti, což se dále využíva vlastně po celém světě. Ovšem ne všude se pro analýzu hodí stejné metody.

Máme tedy několik možností, jak zjistit parametry různých materiálů. Metodou APM (Atomic Probe Microscopy), která má destruktivní charakter a pozoruje jednotlivé atomy, jak jsou z povrchu odstraňovány anebo další XPS (X-Ray Photoelectron Spectroscopy), LEED (Low Energy Electron Difraction) a EELS (Electron Energy Loss Spectroscopy). Detail těchto metod můžete zhlédnout v [2, 4]. Metoda LIBS[1] má výhody v tom, že díky ní dokážeme velmi rychle interpretovat výsledky, příprava vzorku na měření je minimální, finanční náklady jsou také minimální, a dokonce si tahle metoda sama vytváří ideální podmínky pro měření. Naopak těžko takovou metodu můžeme používat například při zkoumání lidského těla, protože se v ní střílí laserem a má omezený prostor pro vzorek. Podrobnosti jsou popsány v kapitole 2.1.

Cílem práce je navrhnout a implementovat aplikaci pro cílovou platformu Windows, která má pracovat s metodou LIBS. Jedná se tedy o synchronizaci několika hardwarových prvků v multivláknové aplikaci. Práce se pak zaměřuje na algoritmus vyhodnocení relativní ostrosti obrazu a algoritmus pro rychlé iterativní stanovení polohy snímače, tedy autofokus. Důležitým cílem je rovněž návrh a vytvoření grafického uživatelského rozhraní se zaměřením na měření již zmíněnou metodou. Avšak rozhodně se v ní nesoustředím na optimalizace jakýchkoliv algoritmů.

V první kapitole se čtenář seznámí s detaily metody LIBS, popisem pracovního prostředí a trochou teorie k autofokusu. Druhá kapitola pak poskytne informace o Qt frameworku a návrhu aplikace. Dále je zde lehce popsána implementace některých částí aplikace a jádrem této kapitoly je implementace algoritmu autofokusu. Poslední kapitola je zaměřená na testování. V první řadě je zde předvedeno měření ostrostí jednotlivých vzorků. Poté zde porovnávám na stejných vzorcích implementovaný algoritmus autofokusu a v závěru kapitoly předvádím výsledky testů při hodnocení ergonomie grafického návrhu.

[1] Laser Induced Breakdown Spectroscopy

Kapitola 2

Použité metody a popis pracoviště

2.1 Laserová spektroskopie (LIBS)

„Laserová spektroskopie (LIBS) patří mezi moderní metody analýzy chemického složení materiálu. K detekci jednotlivých chemických prvků využívá spektrální charakteristiky záření emitované plazmatem, které na povrchu zkoumaného vzorku vytvoří pulz fokusovaného laserového svazku.“ [10]

Výhodou této metody je, že jde o vyhodnocování dat v reálném čase, příprava materiálů je minimální a dále našla metoda uplatnění i v těžko přístupných a nebezpečných místech (metodu lze implementovat pro vzdálené ovládání, tzv. rLIBS[2]). My se však v této práci omezíme na lokální ovládání z vlastního počítače.

2.1.1 Popis metody z fyzikálního hlediska

Základní princip: Paprsek laseru nasměrujeme na vzorek. Při dopadu paprsku na vzorek se odpaří nepatrné množství materiálu a pomocí spektroskopu zachytíme požadované spektrum látek.

Měření se tedy skládá z několika navazujících částí:

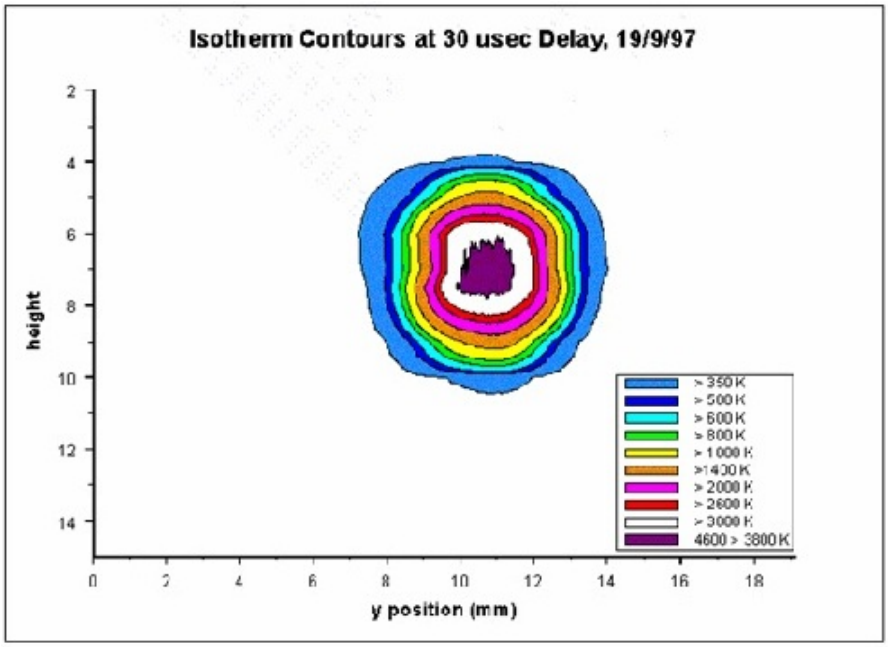
- vytvoření mikroplazmatu,
- časový rozvoj plazmatu a sběr záření,
- kalibrace a vyhodnocení.

Nejprve vyšleme laserový pulz. Pro metodu LIBS se používají ablační pevnolátkové lasery s délkou trvání pulzu v jednotkách nanosekund. Pro zvýšení intenzity a zmenšení stopy svazku na vzorku je tento svazek fokusován. Metoda dále využívá Q-switch[3] režimu, díky kterému lze vyvolat intenzitu o velikosti několika GW/cm^2. Při dopadu laserového fokusovaného paprsku na vzorek vznikají jevy závislé na délce trvání pulzu, vlnové délce laserového záření a na chemických a fyzikálních vlastnostech vzorku. Dochází především k intenzivnímu ohřevu vzorku na velmi malé ploše, a tím vzniká tenká roztavená vrstva. Tepelná energie na povrchu vzorku dále narůstá (dodané teplo dosáhne skupenského tepla varu daného vzorku a ten se začne odpařovat). Vznikající plazma se rozpíná rychlostí až 10km/s, a to po celou dobu trvání pulzu.

[2] remote LIBS

[3] Prvek uvnitř laseru, který pohlcuje energii od jiného prvku (FLASH lampa) a až jí bude mít požadované množství, vyšle se laserový pulz

Po skončení doby trvání pulzu je v prvních desítkách nanosekund emitováno kontinuální brzdné záření. V této fázi tedy nelze provádět spektroskopické měření. V prvních stovkách nanosekund se plazma ochlazuje a začínají být patrné ostré spektrální čáry. Ještě však není vhodné vzorek měřit. Po řádově tisících nanosekundách zůstávají viditelné pouze ostré emisní čáry ionizovaných atomů, které jsou zachycovány spektroskopem právě pro metodu LIBS. Časový rozvoj plazmatu vidíme na následujícím obrázku.



Obrázek 2.1: Časový rozvoj plazmatu [5]

Po detekci emisních čar ionizovaných prvků lze vyhodnotit chemické složení vzorku. Jelikož ale LIBS není absolutní metoda, vyžaduje standardní vzorky o známém složení, které budou sloužit pro srovnání a pozdější analýzu vzorku s prozatím neznámým složením. Z emisních čar se tedy sestrojí kalibrační křivky, podle kterých je možné nalézt obsah prvku v neznámém vzorku. Proces vyhodnocení se automatizuje počítačem a je velmi rychlý (vyhodnocení v reálném čase) [12].

2.1.2 Popis aparatury

Aparatura je složena ze tří základních částí: **laser**, **systém optických vláken** a **sestava pro spektrální analýzu**. Dalšími důležitými prvky, kterými se budeme zabývat především při implementaci, jsou: fokusační kamera (včetně závěrky kamery), motory, pulzní generátor, modul vzduchotechniky, zdroj světla, interakční komora, popřípadě další laser.

Konkrétní typ **laseru**, který v této aparatuře použijeme, se nazývá *Nd:YAG laser*. Jedná se o ablační pevnolátkový laser s velmi vysokým výkonem. Při připojení druhého laseru bude spuštěný mód tzv. *double-pulse*[4]. Lasery používají již zmíněnou techniku Q-switch. Laser je neustále na pevném místě.

Pro pohyb se vzorkem použijeme šestiosý (využijeme však jen 3 osy) **motor** *TMCM-6110* od firmy Trinamic. Vzorek budeme posouvat pod fokusovaným laserem ve 3 osách:

[4]Dvojitý výstřel laserů (2 po sobě jdoucí pulzy laserů s velmi malým zpožděním) do jednoho místa ve vzorku

X, Y, Z.

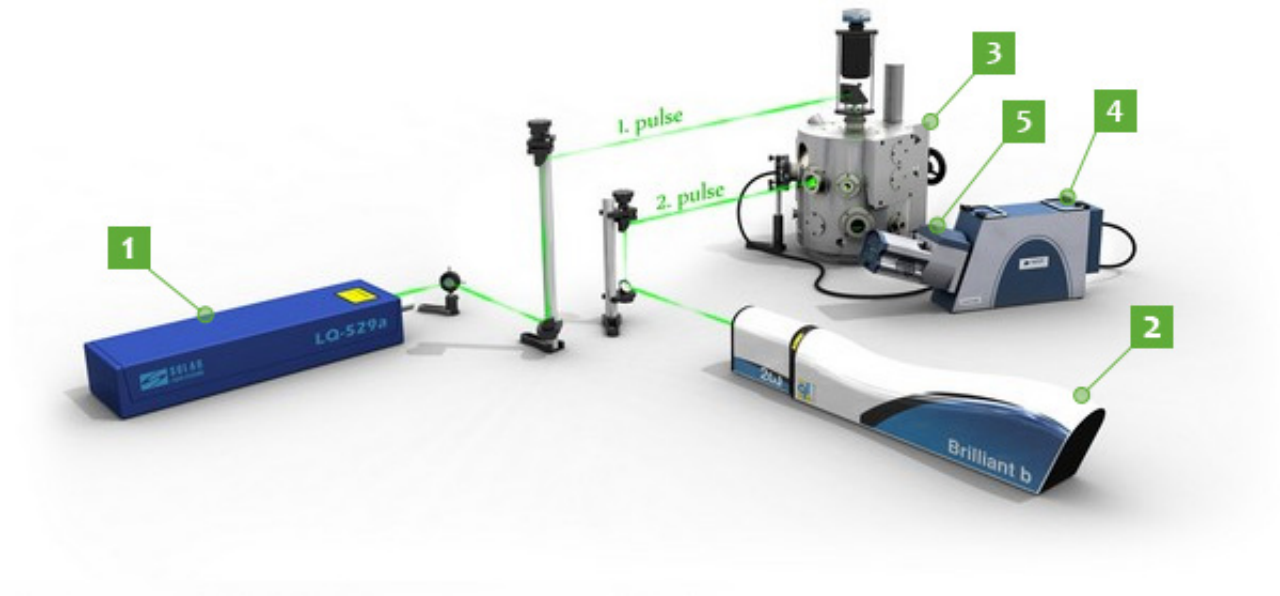
Fokusační kamera má kromě funkce sledování vzorku také funkci zaostřování (na tentýž vzorek). A to nejen pro ostrý obraz při náhledu na vzorek, ale také bude možné změnit přiblížení kamery do 2 vzdáleností pomocí dvou sad optických čoček. Dále bude řízena závěrka kamery při výstřelu laseru, aby se čip kamery nespálil.

Do **pulzního generátoru** jsou vedeny vodiče pro generování pulzů kvůli synchronizaci dvou laserů a u každého laseru správného načasování Q-switche. Dále je zde připojen vodič pro vygenerování signalizačního pulzu do detektoru, kdy může začít spektroskop sbírat data.

Modul vzduchotechniky bude sloužit pro zvyšování či snižování tlaku v komoře pomocí ventilace různých plynů, a tím zlepšování měřících schopností komory. O modulu nemáme žádné podrobné informace, neboť ještě nebyl definitivně navržen.

Zdroj světla má jednoduchou funkci – zajistit osvětlení uvnitř komory.

Většina z těchto prvků je uzavřena do **interakční komory**. Při proběhnutí výstřelu laseru jsou data směrována přes optická vlákna do sestavy pro spektrální analýzu, kde se analyzují a poté se na počítači vyhodnotí. Na následujícím obrázku je vidět schéma naší konkrétní aparatury (obecné aparatury jsou podobné, mají malé odchylky – například absence interakční komory, obsah komory však zůstává).



Obrázek 2.2: Schéma aparatury: 1. laser Sollar, 2. laser Brilliant, 3. Interakční komora, 4. Spektrometr, 5. ICCD kamera [9]

2.1.3 Popis pracoviště

Laboratoř s popisovanou LIBS komorou se nachází na **Ústavu fyzikálního inženýrství** na **Fakultě strojního inženýrství** školy **Vysokého učení technického v Brně**. Vedoucím laboratoře je **doc. Ing. Jozef Kaiser, Ph.D.**, který tuto pracovnu buduje už od roku 1997 [11]. Areál se skládá z mnoha budov. Naše pracovna se nachází v budově

*A*2 **v 5. patře** tohoto areálu. Číslo dveří je 519 a jsou to první dveře vpravo. Přístup do pracovny je zprovozněn přes čipovou kartu zaměstnance. Nachází se zde 2 stoly, z nichž na jednom je složena komora LIBS s ostatními komponentami a druhý stůl slouží pro odkládání/vytváření/testování dalších potřebných modulů do LIBS aparatur. Kromě této LIBS komory se zde totiž nachází i remote LIBS, na které pracuje můj kolega. Dále je ke komoře připojen PC, přes který je možné aparaturu testovat. Lepší je ale přijít se sestaveným programem na vlastním notebooku. Nejen že v laboratoři najdeme obě tyto aparatury, ale dále je laboratoř vybavena řadou periferních zařízení:

- HeNe laser Metra Blansko LA121,
- HeNe laser Melles Griot 05-LH-925,
- Polovodičový laser Andor ME-OPT-0007,
- Polovodičové lasery Thorlabs CPS532, CPS180,
- pulzní generátor Standford DG535,
- rotační vývěva Pfeiffer Vacuum Duo 5 a další...

Hned vedle laboratoře se nachází kancelář, kde probíhají různé porady a schůzky týmu.

2.1.4 Tým

Pro takový složitý a rozsáhlý projekt nestačí jen jeden člověk. Proto se mnou na tomto projektu spolupracuje spousta lidí (vyjmenuju jen ty, se kterými spolupracuji):

- Jan novotný – můj osobní vedoucí, specifikace SW,
- David Procházka – zaučení do problematiky, vývoj SW (přede mnou),
- Michal Petrilak – správa HW.

2.2 Autofokus

Jedním ze stěžejních problémů této aplikace je autofokus. Kamera nahlížející na vzorek je v pevném bodě a vzorek by měl být v jejím předmětovém ohnisku[5]. Toho dosáhneme pomocí posuvu motoru představujícího zetovou osu ve směrech nahoru a dolů. Nejen že je autofokus důležitý pro náhled z kamery, ale také musí být velmi přesný, protože paprsky laserů se budou sbíhat ve stejné ohniskové vzdálenosti, kde se nachází vzorek, který je detekován skrze čočky na čip kamery. Paprsky totiž procházejí tou samou optickou soustavou. Pokud tedy dostatečně přesně nezaměříme vzorek, paprsek laseru neprovede výbuch v jednom bodě vzorku, ale kvůli rozostření budou paprsky mírně rozptýleny. Nejprve je potřeba obraz zpracovat a vyhodnotit jeho ostrost. Dále musíme určit algoritmus, který vyhodnotí nejostřejší snímek v reálném čase. Postup tedy rozdělíme do několika podúkolů:

- Vyhodnocení ostrosti obrazu
  - Digitalizace obrazu

[5]Předmětové ohnisko je bod, do kterého se sbíhají pomyslné paprsky procházející optickou čočkou. V tomto bodě je obraz nejostřejší.

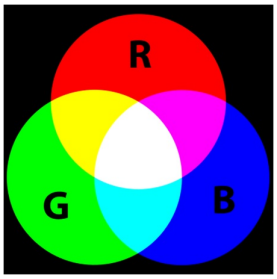
  - 2D diskrétní Fourierova transformace
  - Amplitudové spektrum
  - Gaussův váhový filtr a vyhodnocení
- Nalezení nejostřejšího snímku

2.2.1 Digitalizace obrazu

V digitálních technologiích můžeme obraz reprezentovat jako **dvojrozměrnou matici**, v níž každý bod představuje určitou hodnotu (barevnost, jas...). V naší kameře DCC1645C je záznamový čip (podobně jako u ostatních dnešních digitálních kamer a fotoaparátů), který působí jako vzorkovací funkce. Na vzorek dopadají fotony[6] a my pomocí **expoziční doby**[7] (parametr kamery) nastavíme vzorkovací frekvenci. Po každém posbírání množství světla ve snímaných bodech nám vzniká již zmíněná dvojrozměrná diskretizovaná matice.

Důležité je však zmínit to, že kvůli vadě optických soustav, takzvané **vinětaci**, kdy je na okrajích a nejvíce v rozích nižší jas než ve středu obrázku, je potřeba vybrat oblast ze snímku, které se vinětace netýká. Proto při každém získání snímku z kamery ořízneme obraz tak, aby nám zůstal jen čtverec, jehož střed bude ve středu původního snímku. Kromě toho, že tedy výsledná matice bude čtverec, je ještě nutné, aby byla jeho strana rovna mocnině čísla 2. To vše kvůli dalším úpravám a modifikacím tohoto snímku.

Rozlišujeme několik systémů barev: **RGB** (red-green-blue), **CMY** (cyan-magenta-yellow), **HSB** (hue-saturation-brightness), **YCbCr**... V mém programu je použit systém barev RGB s využitím aditivního míchání, viz následující obrázek 2.3, kde si můžete všimnout, že při maximální intenzitě všech tří barev vzniká bílá barva, naopak když bude intenzita všech barev nulová, vzniká nám černá.



Obrázek 2.3: Aditivní míchání RGB barev

Jak už název napovídá, model RGB má 3 základní složky: Red, Green, Blue. Každá z těchto složek je reprezentována na 8 bitech. Může tedy nabývat $2^8 = 256$ hodnot, tedy hodnot $0 - 255$. Pro jeden pixel (jednu složku RGB barvy) tedy čítáme $3 \cdot 8 = 24$ bitů, což odpovídá 256^3 různým barevným odstínům. Podrobnější rozbor najdete například v manuálu kamery [17]. Lidské oko nejvíce reaguje na zelenou barvu, potom na červenou a pak na modrou. Proto je vhodné optimalizovat barevné složky RGB do jedné hodnoty intenzity, a to pomocí vzorečku

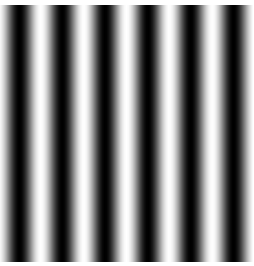
$$Y = 0,299R + 0,587G + 0,114B, \tag{2.1}$$

[6] částice světla
[7] doba, po kterou dopadá světlo na čip kamery, než se sejme snímek

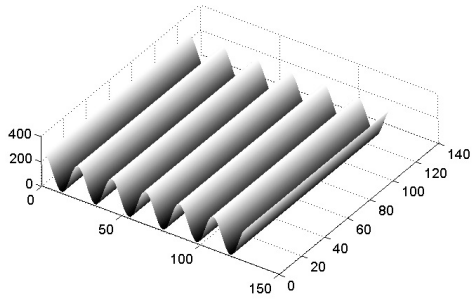
kde vidíme, že složka zelené barvy nabývá nejvyšší hodnoty právě kvůli citlivosti lidského oka na tuto barvu, naopak modrá složka je skoro zanedbatelná. **Y** je kromě naší vypočítané intenzity také složka systému barev YCbCR, ale tím se v této práci nebudeme zabývat.

2.2.2 2D diskrétní Fourierova transformace

Diskrétní Fourierova transformace, respektive rychlá Fourierova transformace (FFT[8]) je jedna z nejznámějších metod pro zpracování signálů. V našem případě se tedy jedná o diskrétní obrazový signál. Pro podrobnější vysvětlení použijeme 2 následující obrázky převzaté z literatury [14]:



Obrázek 2.4: Ukázkový snímek



Obrázek 2.5: Průběh jasu ve 3D

Na obrázku 2.4 vidíme jednoduchý snímek ve dvou rozměrech. Pro lepší představu si ho můžeme ukázat ve třech rozměrech, viz 2.5. Na obrázku vpravo tedy vidíme, že se signál skládá z jakýchsi sinusoid/kosinusoid. A to je právě podstata Fourierovy transformace. Na vstupu dostane nějaký signál, který přetransformuje do několika různých sinusoid/kosinusoid lišících se v amplitudě, fázi a prostorové frekvenci (respektive periodě). **Prostorová frekvence** je právě ta hodnota, která nás zajímá... Její hodnota je v případě digitalizovaného obrazu závislá na rozlišení snímku. Poskytuje o obraze velice cenné informace a právě s touto hodnotou se při různých editacích snímků manipuluje. U většiny snímků převažují nízké frekvence. Pro nás jsou ale důležité **vysoké frekvence**, protože právě ty mají na svědomí přítomnost a podobu hran v obraze [14].

Nyní si ještě připomeňme, jak se Fourierova transformace počítá. Vycházet budeme z následujícího vzorce pro diskrétní signály:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk} \qquad k = 0, \ldots, N-1. \tag{2.2}$$

Tento vzorec je aplikovatelný pro jednorozměrný diskrétní signál. Vstupem jsou komplexní čísla (v našem případě je na vstupu imaginární složka nula a reálná je právě hodnota jasu Y) a výstupem jsou také komplexní čísla, ale ve tvaru jakýchsi koeficientů, které nám představují již zmíněné sinusovky/kosinusovky. Po aplikaci tohoto vzorce se nám přepočítají hodnoty po řádcích (1D). My ale máme matici (2D), na kterou se tedy aplikuje 1D FFT po řádcích a poté 1D FFT po sloupcích. Tím dosáhneme kompletní dvojrozměrné diskrétní Fourierovy transformace.

[8] Fast Fourier Transform

2.2.3 Amplitudové spektrum

Fourierova transformace nám tedy vypočítá prostorové frekvence, ze kterých se vypočítá amplitudové spektrum. Jak už název napovídá, budeme počítat největší výchylku. Bude zde tedy hrát roli absolutní hodnota, protože výchylka může být i záporná. Každý pixel naší matice tedy přepočítámé podle vztahu:

$$P = \sqrt{Re^2 + Im^2}, \tag{2.3}$$

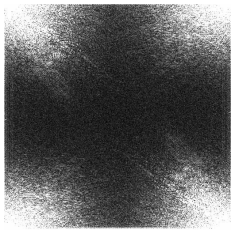
kde P představuje novou hodnotu pixelu, která se vypočítá jako odmocnina ze součtu reálné a imaginární části prostorové frekvence.

Pro ukázku výstupu použijeme jako vstup známý obrázek Leny

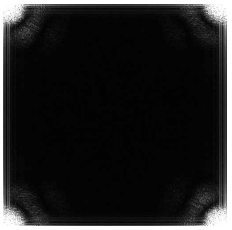
Obrázek 2.6: Ostrý snímek

Obrázek 2.7: Rozostřený snímek

a k nim odpovídající výstupy amplitudových spekter:

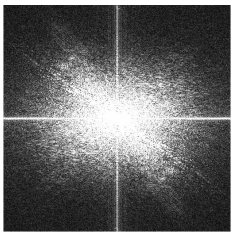


Obrázek 2.8: Ostrý snímek

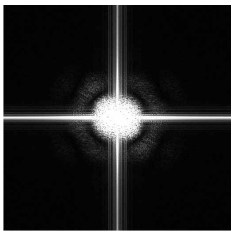


Obrázek 2.9: Rozostřený snímek

Z obrázků vyplývá, že v rozích jsou nižší frekvence než u středu. Nejen kvůli budoucímu použití Gaussova filtru by však bylo vhodné upravit takovou matici do **konvenčního tvaru**, viz [19]. V praxi to znamená, že musíme přehodit první se třetím kvadrantem v matici a druhý se čtvrtým (také můžeme kvadranty rotovat dvakrát na tutéž stranu). Tím se dostáváme do stavu, kdy nízké frekvence leží u středu matice a ty se zvyšují směrem ke krajům. Přímo ve středu je pak minimální hodnota amplitudy a na kraji maximální hodnota, jak je ukázáno na následujících obrázcích:



Obrázek 2.10: Přeskládání kvadrantů ostrého snímku



Obrázek 2.11: Přeskládání kvadrantů rozostřeného snímku

2.2.4 Gaussův váhový filtr a vyhodnocení ostrosti

V posledně zmíněných obrázcích si také můžeme všimnout, že frekvence blízko středů mezi oběma obrázky se příliš neliší. Odlišnosti jsou patrné až v určité vzdálenosti od středu ke kraji. Jedním z nejjednodušších způsobů, jak vyjádřit hodnotu ostrosti, by byl například výpočet průměrné amplitudy přes celé spektrum podle:

$$O = \frac{\sum_{x=1}^{X}\sum_{y=1}^{Y} M(x,y)}{XY}, \tag{2.4}$$

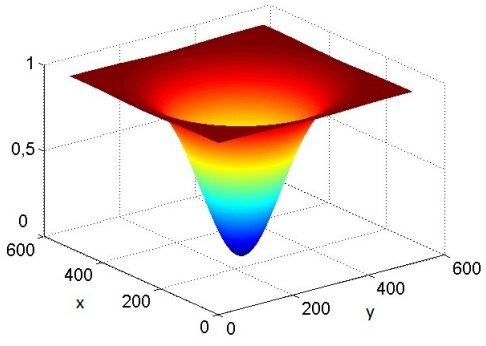
kde O odpovídá hledané hodnotě ostrosti, X a Y jsou počty řádků a sloupců (v našem případě čtvercová matice, tedy $X = Y$), x, y jsou indexy řádků a sloupců a $M(x,y)$ je hodnota naší matice M s danými indexy. To celé dělíme počtem prvků v matici, tedy $X \cdot Y$.

V praxi je ale vhodné aplikovat jeden z filtrů. Pomocí Gaussova filtru můžeme například odstranit šum z obrazu. To ale není náš případ. My tento filtr použijeme pro odstranění nízkých frekvencí v obrazové matici, protože ty nám, jak jsem již zmínila, rozdíl v ostrosti neprovádí (viz středy u snímků 2.10 a 2.11). Po použití takového filtru se tedy změní výpočet ostrosti O matice M takto:

$$O = \frac{\sum_{x=1}^{X}\sum_{y=1}^{Y} M(x,y)G(x,y)}{XY}, \tag{2.5}$$

kde přibyl Gaussův váhový filtr G s indexy x, y, tedy $G(x,y)$. Tento filtr si lze ve skutečnosti představit opět jako matici, v našem případě čtvercovou, o stejných rozměrech jako je naše obrazová matice, ze které chceme získat požadovanou ostrost. Váhování tímto filtrem se potom provádí tak, že se vynásobí každý člen obrazové matice M se členem matice filtru G na stejných indexech. Neprovádí se zde tedy žádná konvoluce (tu by bylo vhodné použít u vyhlazení obrazu od šumu), nýbrž je zde aplikováno pouhé jakoby překrytí stejně velkých matic. Kdyby tedy obrazová matice byla velká 3x3 pixelů, příklad aplikovatelného filtru by byl na obrázku 2.13.

Jak už název napovídá, Gaussův filtr bude mít tvar tzv. „gaussovky“. Pro lepší představu se podívejme na následující obrázek 2.12 ve 3D:



Obrázek 2.12: Gaussův filtr 3D

1	0,5	1
0,5	0	0,5
1	0,5	1

Obrázek 2.13: Ukázka hodnot v matici 3x3 Gaussova filtru

Na první pohled si lze povšimnout hned několika důležitých věcí. Filtr je **invertovaný**. To je právě kvůli nízkým frekvencím uprostřed snímků amplitudového spektra. Zde bude

hodnota Gaussovy funkce (váhy) nejnižší. Dále z obrázku vyplívá, že hodnoty ve filtru budou nabývat hodnot **0 až 1**. To nemusí být pravidlem, ale pro náš případ se omezíme právě na váhování těmito hodnotami. Nyní si vysvětleme, jak vypočítáme požadovanou matici Gaussova filtru. Používá se pro to již zmíněná **Gaussova funkce** [16, 14, 8], jejíž znění je:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}. \tag{2.6}$$

V této funkci se skrývá několik vlastností Gaussova filtru. Hned první člen

$$\frac{1}{2\pi\sigma^2} \tag{2.7}$$

nám říká, jak bude maximálně „gaussovka“ vysoká. Když se podíváme na obrázek 2.12, tak vidíme, že hodnoty dosahují maximálně hodnoty 1. Po dosazení do 2.6 dostáváme

$$G(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}}. \tag{2.8}$$

Pro invertovaný filtr odečteme vypočtenou hodnotu od maximální hodnoty, tedy

$$G(x,y) = 1 - e^{-\frac{x^2+y^2}{2\sigma^2}}. \tag{2.9}$$

Čitatel v exponentu exponenciály nám říká, kam bude „gaussovka“ posunutá. Jmenovatel určuje její šířku. Proměnné x a y tedy posuneme o polovinu rozměru matice v obou směrech, abychom dostali vrchol „gaussovky“ do středu matice. Po dosazení získáváme tedy

$$G(x,y) = 1 - e^{-\frac{\left(x-\left(\frac{X}{2}\right)\right)^2+\left(y-\left(\frac{Y}{2}\right)\right)^2}{2\sigma^2}}. \tag{2.10}$$

A konečně σ, která je mimo jiné i směrodatná odchylka, ale nám v praxi určuje, jak bude „gaussovka“ široká. Označme tedy r jako efektivní poloměr[9] (na obrázku 2.12 je zobrazen filtr s efektivním poloměrem 100). Pak platí:

$$r = \sqrt{2\ln(2)} \cdot \sigma, \tag{2.11}$$

viz [8]. Efektivní poloměr r je volitelná proměnná, díky které tedy vypočítáme σ:

$$\sigma = \frac{r}{\sqrt{2\ln(2)}}, \tag{2.12}$$

kde ještě můžeme odstranit dvojku pod odmocninou, protože se vykrátí po dosazení do výsledného vzorce:

$$\sigma = \frac{r}{\sqrt{\ln(2)}} \tag{2.13}$$

a po dosazení do 2.10 dostáváme výslednou funkci:

$$G(x,y) = 1 - e^{-\frac{\left(x-\left(\frac{X}{2}\right)\right)^2+\left(y-\left(\frac{Y}{2}\right)\right)^2}{\sigma^2}}. \tag{2.14}$$

Když potom proženeme matici získaných amplitudových spekter tímto filtrem, sečteme všechny hodnoty a zprůměrujeme (podle 2.5), dostáváme požadovanou ostrost snímku.

[9] Efektivní poloměr je vzdálenost od středu Gaussovy funkce do bodu, ve kterém nabývá funkční hodnota poloviny maximální hodnoty, tedy 0,5

2.2.5 Nalezení nejostřejšího snímku

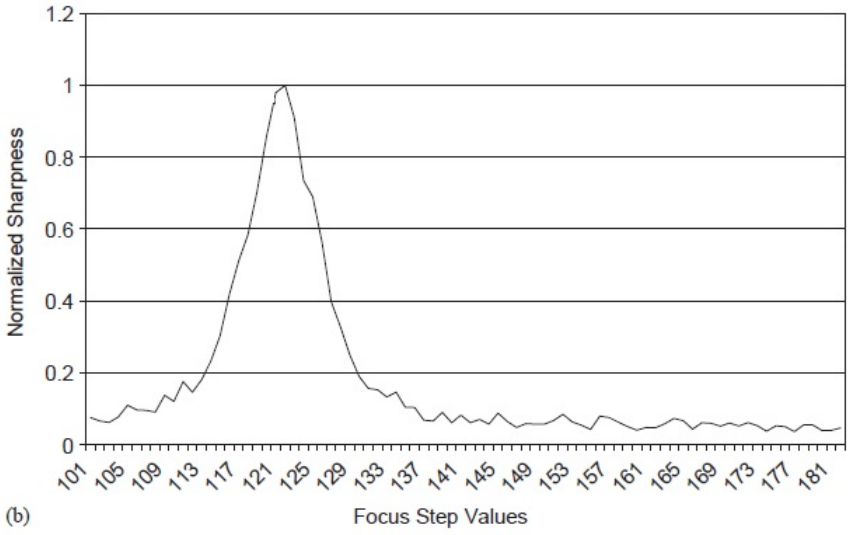
Autofokus je jedním z nejdůležitějších softwarových prvků kamer. My ale máme k dispozici pouze čip digitální kamery, díky kterému jsme schopni vyhodnotit ostrost obrazu. Proto zde zbývá otázka, který je ten správný algoritmus pro autofokus, tedy vyhodnocení nejostřejšího snímku. Jelikož se bavíme o real-time aplikaci, požadujeme po takové operaci, aby byla co nejrychlejší a nejefektivnější. Vycházíme z následujícího stavu:

- kamera (čip) je neustále v pevném bodě, nahlíží shora na vzorek,
- vzorek je položen na posuvném modulu, kterým posouváme pomocí motorů v osách X a Y,
- umíme vyhodnotit ostrost snímku v libovolné vzdálenosti vzorku od kamery.

Zde je důležité uvědomit si parametry, které ovlivňují rychlost autofokusu. Jsou to především:

- rychlost posuvu motoru X,
- rychlost posuvu motoru Y,
- doba trvání vyhodnocení ostrosti obrazu,
- rychlost posuvu motoru v ose Y (tento motor má největší rychlost právě kvůli autofokusu, protože slouží pro přibližování a oddalování vzorku od čipu kamery),
- počet kroků motoru Z.

Obecná charakteristika křivky vyhodnocování ostrosti obrazu vypadá asi takto:



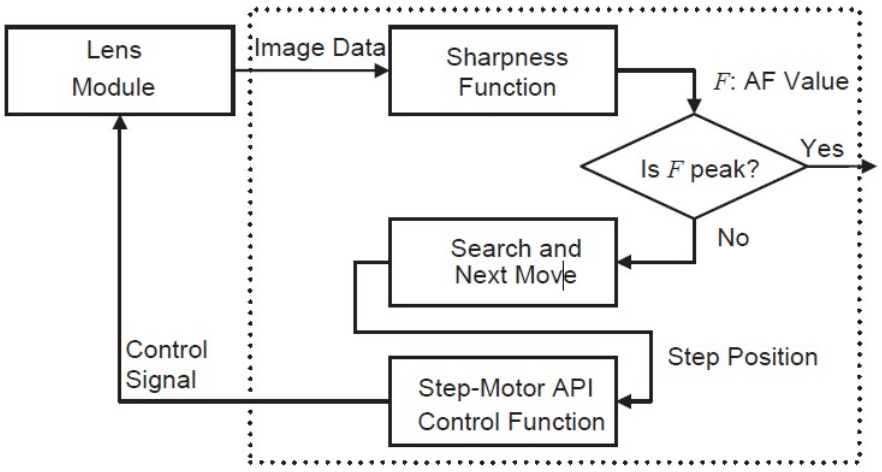
Obrázek 2.14: Křivka ostrosti [7]

Na vodorovné ose jsou vyčísleny kroky motoru (neboli pozice vzorku v ose Z) a na svislé ose vidíme hodnoty ostrosti. Ovšem takovéto hodnoty by pro nás byly ideálním případem. Na svislé ose totiž vidíme normalizované hodnoty, tudíž by se dalo počítat s rozsahem hodnot 0 až 1. Tady ale nastává náš první problém. A to ten, že u každého vzorku se pohybujeme v jiném rozsahu relativních čísel. Pro jednoduchost si ještě rozdělme graf na dvě oblasti: **střední oblast** (tam, kde hodnoty kolísají pouze šumem a jsou mimo nejvyšší vrchol) a **vrcholová oblast** (tam, kde vidíme prudké stoupání a klesání kolem nejvyššího vrcholu). Z obrázku tedy můžeme předpokládat, že:

- mezi ostrým a neostrým obrazem je velký skok,
- ve vrcholové oblasti začíná rapidní nárůst a pokles hodnot ostrosti,
- pokud se blížíme k vrcholové oblasti z jakékoliv strany, hodnoty budou narůstat.

Vidíme ale, že když se přibližujeme k vrcholové oblasti, tak hodnoty sice globálně rostou, ale lokálně mohou kolísat. To je způsobeno právě šumem, ve speciálních případech pak osvětlením. Šum je možné eliminovat filtrem, kvalitnějším čipem v kameře, zlepšením okolního prostředí... Při nedostatku, nebo naopak přebytku osvětlení můžou hodnoty různě kolísat, protože nedetekujeme skoro žádné hrany, ale pouze šum. V našem případě nebyl použit žádný filtr na vyrušení šumu. Děláme dostatečně velké kroky, díky kterým kolísání hodnot skoro vymizí a čip kamery se také jeví býti velmi kvalitním.

Nyní se podívejme na základní všeobecný **iterační proces** algoritmu celého autofokusu, který jsem převzala z [7]:



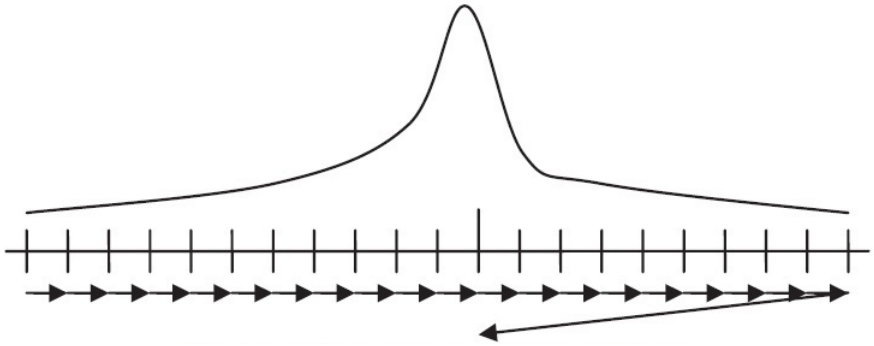
Obrázek 2.15: Iterační proces

Na obrázku 2.15 tedy vidíme jednoduchý průběh algoritmu nalezení peaku. Z optického modulu nám do programu vstupují data o obrazu. Funkcí ostrosti zjistíme jeho ostrost. Pokud jsme našli peak, ukončíme vyhledávání. Pokud ale ještě nejsme v maximu, vyhledávání pokračuje – provedeme posun motoru. Tím získáváme nový stav a opět opakujeme akci zaslání dat z kamery do programu a vyhodnocování ostrostí až do té doby, dokud nenalezneme peak.

Kterým směrem provedeme posuv motoru a o kolik, je různé. Pro základní rozdělení a jednodušší představu si uveďme 3 základní algoritmy, viz [7]. Pro jednoduchost uvažujme

start na jednom konci křivky (přiblížení kamery je maximální, nebo minimální – je tedy jasné, kterým směrem se při startu vydáme).

Nejjednodušší je **globální vyhledávání**. To probíhá tak, že si zvolíme konstantní velikost kroku. Hned na začátku, tedy na startovní pozici, zjistíme ostrost prvního snímku. Poté motor posuneme o zvolený konstantní krok a opět zjistíme ostrost snímku. Takto naměříme po jednotlivých krocích všechny vzorky (celou osu motoru Z), přičemž si po celou dobu ukládáme vždy maximum. Až detekujeme poslední vzorek, vyhodnotíme maximum jako peak a motor posuneme právě na tuto pozici, viz ukázka 2.16:

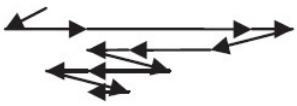


Obrázek 2.16: Globální vyhledávání

Výhody a nevýhody globálního vyhledávání:

- při zvolení rozumného kroku nikdy nemineme peak (třeba s drobnou odchylkou),
- nezajímá nás žádný šum (zda hodnoty klesají či rostou), vždy jdeme přímo jedním směrem,
- byly by pro nás jednoduché i vícehladinové vzorky viz 4.2,
- těžko určíme, jaký je ideální krok (při velmi malém bude růst doba trvání vyhledávání, při velkém se může stát, že přeskočíme peak, nebo ho najdeme s nechtěnou odchylkou),
- zbytečně se provádí velké množství kroků motoru (vždy se musí změřit snímky na celé ose)...

Další vyhledávací přístup pro autofokus může být **binární vyhledávání**. U tohoto přístupu musíme umět detekovat situaci, kdy přeskočíme peak. Začneme se tedy pohybovat ze startu opět jedním směrem. Jakmile zjistíme, že jsme přeskočili peak, vracíme se zpět (třeba s menším krokem). Pokud při navracení se opět zjistíme, že jsme přeskočili peak, změníme směr. Je to vlastně jakási forma algoritmu s navracením. Ukončit vyhledávání můžeme například splněním podmínky, kdy se 2 hodnoty (předchozí a aktuální) od sebe liší o dopředu známou hodnotu (odchylku). Detail tohoto algoritmu vidíte na následujícím obrázku 2.17:



Obrázek 2.17: Binární vyhledávání

Výhody a nevýhody binárního vyhledávání:

- pokud je peak blízko startu, metoda je rychlejší než globální vyhledávání (neprocházíme celou osou),
- největší výhodou je její přesnost, která je navíc volitelná,
- první nevýhodou je, že navracením se neustále přejíždíme motorem po stejné oblasti vzorku,
- dále již zde musíme brát v úvahu šum, díky kterému nám mohou hodnoty ostrosti kolísat,
- a pokud měříme na vícehladinovém vzorku, může se stát, že mineme peak (za hledaný peak budeme považovat první nalezený), viz 4.2 při vyhledávání z druhé strany (zprava grafu)...

Jako poslední ze základního rozdělení vyhledávacích přístupů představím **vyhledávání založené na pravidlech**. Celý rozsah osy posuvu si rozdělíme na několik oblastí: počáteční, vrcholovou, střední a obyčejnou. Pro každou tuto oblast si definujeme jinou velikost kroku či jiný počet kroků v dané oblasti. V počáteční oblasti tedy děláme docela jemné kroky, kdybychom náhodou už byli na peaku, ať ho nemineme. Poté se dostáváme do střední oblasti, kde můžeme kroky mírně zvětšit. Jakmile detekujeme vrcholovou oblast, děláme nejjemnější kroky proto, že víme, že jsme blízko lokálního maxima[10]. To můžeme považovat za globální maximum[11], tedy peak, nebo můžeme nechat s dostatečně velkým krokem prozkoumat zbytek osy, zda se tam nenachází ještě větší maximum (což může opět nastat u výjimečných vzorků 4.2). Po projetí celé osy se opět, jako u globálního vyhledávání, vrátíme do globálního maxima. Detail je znázorněn na následujícím obrázku 2.18, podrobnější obrázek již byl zmíněn zde 2.14.

Obrázek 2.18: Vyhledávání založené na pravidlech

Výhody a nevýhody vyhledávání založeného na pravidlech:

- děláme přiměřený počet kroků na každou oblast, tím algoritmus zrychlujeme a zpřesňujeme,
- na tento algoritmus je potřeba nejméně kroků ze všech zmíněných přístupů,
- ale ztrácíme přesnost, kterou bychom získali u binárního vyhledávání...

V praxi je možné použít jednu určitou metodu z těchto zmíněných, záleží však na situaci a požadavcích. Já ve své práci implementuji algoritmus, který je modifikací všech tří, viz následující kapitola Implementace (3).

[10] maximum, která platí pouze pro blízkou oblast
[11] maximum vzhledem k celé ose

Kapitola 3

Implementace

3.1 API, cílová platforma

Požadavky od uživatelů byly takové, aby se program jednoduše ovládal a aby aplikace fungovala na operačních systémech Windows. Požadavky od zaměstnavatele byly, aby aplikace byla psána stejnými prostředky a ve stejném jazyce, jako jsou jejich ostatní programy. Proto tedy cílovými platformami budou různé verze systému Windows (prioritně Windows 7, dále Windows Vista, Windows 8 a nejnovější Windows 8.1). V aplikaci tedy používám konkrétní knihovny na cílovou platformu, jako je například knihovna ⟨ `windows.h` ⟩, dále například knihovnu „`vld.h`“, která patří k programu Visual Leak Detector[12]. Jako knihovnu pro GUI[13] používám QT knihovny. Program je tedy vyvíjen na platformě Windows 7 ve vývojovém prostředí Microsoft Visual Studio 2010 v jazyce `C++` a GUI vytvářím pomocí QT frameworku (QT 4.8).

3.1.1 QT framework

QT je API[14] nejen pro GUI. Používá se například na těchto platformách:

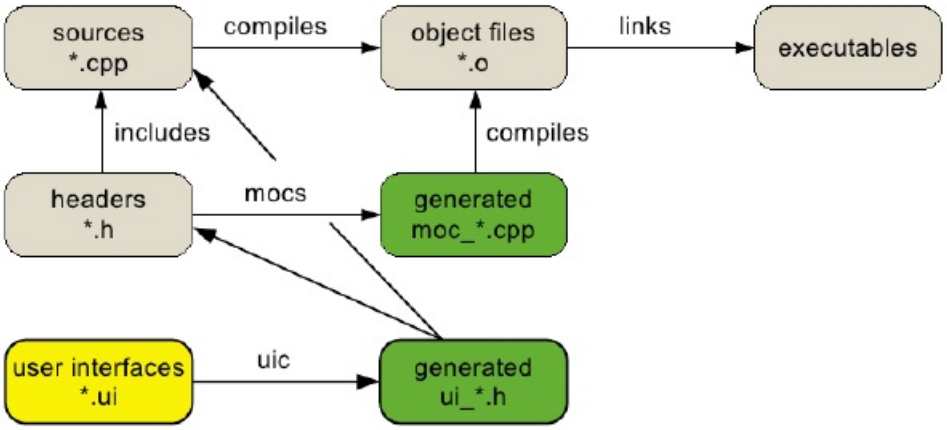
- Windows,
- Mac,
- Linux,
- vestavěné systémy (pomocí speciálních technik),
- mobilní telefony (Android, Symbian...)

Další výhodou tohoto frameworku je, že se s ním dá pracovat v **několika programovacích jazycích**, například `C++` (nejpoužívanější), Python, C♯. Pro zjednodušení programování (například sázení konkrétních grafických prvků) existují programy, které tuto práci ulehčují (tahání prvků myší přímo na plochu). Mezi tyto programy se řadí **Qt Creator**, **Qt Designer**...

QT má docela **složitý překlad**, neboť při překladu generuje a spojuje různé druhy souborů. Uživatelem vytvořené soubory mohou mít koncovky `*.cpp` a `*.h`, což jsou klasické

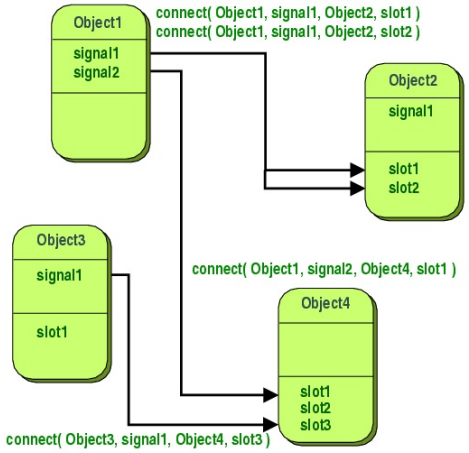
[12] Detektor úniků paměti a dalších chyb při programování – knihovna přímo pro Visual Studio 2010
[13] Graphis User Interface – grafické uživatelské rozhraní
[14] Application Programming Interface – rozhraní pro programování aplikací

zdrojové soubory známé z jazyka `C++`. Pokud pracujeme s jedním z výše uvedených SDK[15], přibývá zde soubor s příponou `*.ui`. Tento soubor (`*.ui`) přeloží UIC[16] na soubor `ui_*.h`, který je nutno inkludovat do hlavičkových souborů `*.h`, popřípadě do souborů `*.cpp`. Ze souborů `*.h` se pomocí MOC[17] vytvoří soubory `moc_*.cpp` a ty se společně se soubory `*.cpp` zkompilují do `*.o`, tzv. objektového souboru. Poté lze sestavit program, viz následující obrázek:



Obrázek 3.1: Překlad a nástroje QT [15, 13]

Správa paměti je velmi podobná objektově orientovanému programování v `C++`. Pomocí příkazů `new` a `delete` se vytváří a ruší objekty, dále viz [6, 15]. Co je ale velmi typické a identifické pro QT, jsou signály a sloty. Tato technika slouží pro komunikaci mezi objekty, například při vyvolání události jedním objektem (kliknutí na tlačítko) pro reakci jiného objektu. Výhodou je, že je to jednoduché propojení existujících tříd, které je oproti ostatním mechanismům (tzv. „Callback“[18]) bezpečné[1].



Obrázek 3.2: Ukázka komunikace signálů a slotů [1]

[15] Software Development Kit – vývojový nástroj
[16] User Interface Compiler – přeloží vytvořené grafické prvky
[17] Meta Object Compiler – pomocí maker přidává funkcionalitu do `C++`
[18] Vytvoříme ukazatel na funkci a při vyvolání události se zavolá odpovídající funkce přes ukazatel

Základní grafický objekt reprezentuje **Widget**, od kterého ostatní třídy dědí. Objekty jsou poskládány do **stromové hierarchie** (hlavní okno, tlačítka...), přičemž pokud dealokujeme rodičovský prvek, tak všechny prvky, které byly tomuto objektu přimknuty (děti), se automaticky dealokují. Jedním ze způsobů jak rozmístit objekty jsou **Layouty**. Pokud vkládáme všechny prvky do těchno Layoutů, získáme tím správné (rozumné a korektní) změny velikostí při změnách velikostí grafických prvků za běhu programu. Dále můžeme u QT využívat jeho vlastní jazyk QML. Pokud potřebujeme v programu pokročilé grafické efekty, nabízí nám tzv. **Shadery** [13].

QT se tedy řadí mezi multifunkční API, jehož podrobnější informace najdete například v knihách [1, 3].

3.2 Návrh aplikace

Chceme tedy napsat software pro ovládání LIBS komory. Ta se skládá z jednotlivých částí zmíněných v kapitole 2.1.2. Známe tedy většinu[19] HW komponent, které bude potřeba synchronizovat a řídit.

3.2.1 Návrh GUI

Aplikace je určena především pro badatele zabývající se oblastí výzkumu laserovou spektroskopií. Program se tedy může zdát velice podrobný, co se týče různého nastavování parametrů. V tomto případě je to ale opravdu nutné, uživatelé jsou profesionálové a veškeré tyto parametry budou velice často měnit.

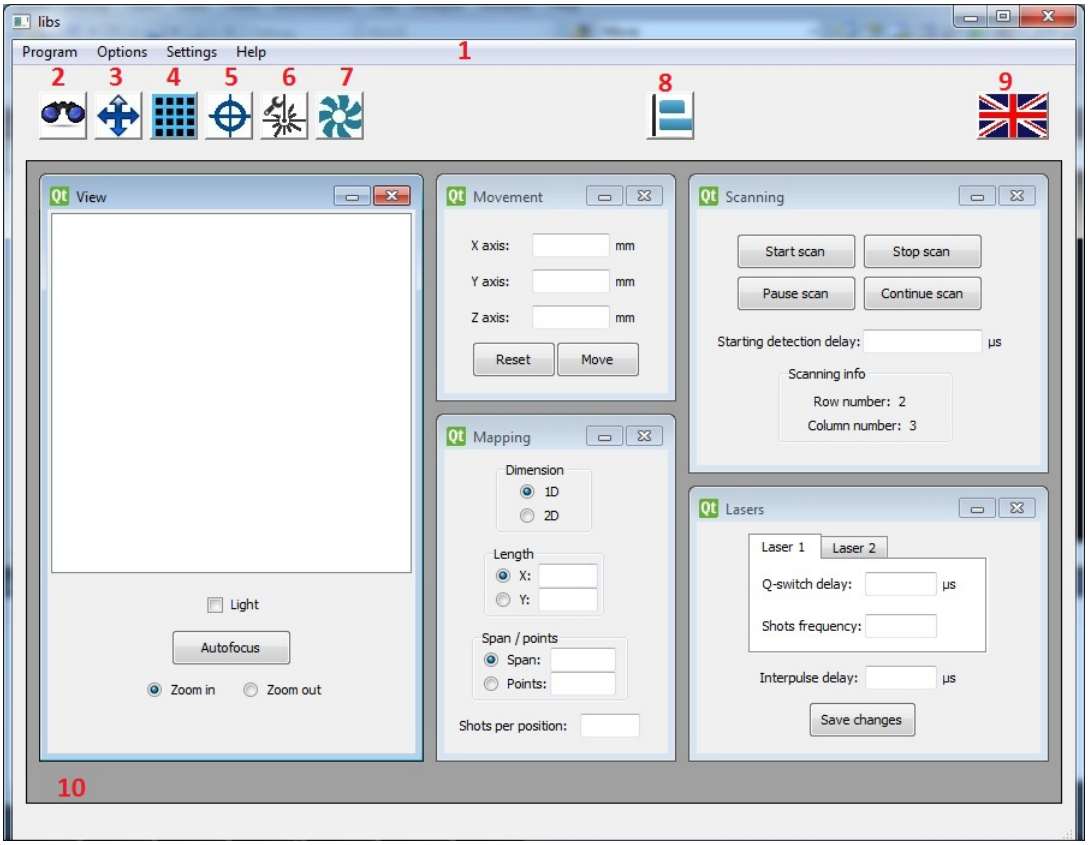
Původní návrh byl takový, že celá aplikace poběží v jednom hlavním okně a všechny potřebné prvky, které by pracovaly s komorou v reálném čase, by byly vidět přímo v tomto hlavním okně. Ostatní nastavení (různá zpoždění a detaily) by bylo skryté a v případě potřeby by se zobrazovala dialogová okna. Tento návrh by byl vhodný, kdyby se aplikace prodávala jako celek. Realita je však jiná. Klient, který si aplikaci koupí, nemusí mít zaplacenou 100% funkčnost. Může mít například koupenou jen kameru (HW) zároveň s modulem Náhled (SW).

Zásadní tedy je, aby aplikace splňovala maximální modularitu. To znamená, že ovládání každého HW prvku bude umožněno nezávazně na ostatních prvcích. Dále si klient nepřeje, aby byla aplikace roztahaná v několika oknech na hlavním panelu. Proto jsem se rozhodla použít techniku podoken, která mi umožní vytvářet okna jako potomky hlavního okna. Aplikace bude ovládána myší. Návrh můžete vidět na obrázku 3.3 s následujícím popisem (čísla odpovídají číslování na obrázku):

1. Hlavní panel s nabídkou – **Program, Options, Setting, Help** – možnosti uložení dat, načtení dat, možnosti skrytého nastavení...
2. Tlačítko **„View“** – otevře/zavře okno modulu Náhled
3. Tlačítko **„Movement“** – otevře/zavře okno modulu Posunu
4. Tlačítko **„Mapping“** – otevře/zavře okno modulu Mapování
5. Tlačítko **„Scanning“** – otevře/zavře okno modulu Skenování
6. Tlačítko **„Laser module“** – otevře/zavře okno modulu Laserů

[19] většinu proto, že je aplikace ve vývoji a požadavky se ještě mohou v zájmu klientů měnit

7. Tlačítko **„Ventilation“** – otevře/zavře okno modulu Ventilace (zatím se přesně neví, jak tento modul bude vypadat, proto je skrytý)
8. Tlačítko **„Reset windows position“** – upraví pozice podoken do původního stavu
9. Tlačítko **„Language“** – přepíná český a anglický jazyk
10. **Plocha pro podokna** – zobrazování a minimalizace podoken, s podokny hýbat pouze v rámci této plochy



Obrázek 3.3: GUI návrh

Když uživatel spustí program, bude moci zobrazovat pouze ty moduly, za které zaplatil. Moduly zobrazené na obrázku 3.3 obsahují tyto části:

- **modul Náhled** – prostor pro zobrazení náhledu z kamery, přepínač pro manipulaci s osvětlením, tlačítko pro zaostření a výběr ze dvou možných vzdáleností sledování vzorku,
- **modul Posunu** – formulář pro zadávání souřadnic X, Y, Z, tlačítko na reset souřadnic do původního stavu a tlačítko pro odeslání příkazu k posunutí,
- **modul Mapování** – výběr rozměru ke skenování (1D či 2D), podle toho vyplnění délky skenování v jedné či dvou osách, zda se bude skenovat určitý počet bodů nebo rozsah na vzorku a počet výstřelů na jeden skenovaný bod,

- **modul Skenování** – tlačítka pro běh/zastavení/přerušení/pokračování skenování, zpoždění kdy má začít detekce emisních čar a zobrazení informací o aktuální pozici skenovacího zařízení,
- **modul Laserů** – pokud budou připojeny 2 lasery s technikou double-pulse, zobrazuje přepínaní mezi těmito lasery, u každého je možné nastavit Q-switch delay a frekvenci výstřelů. Mezi oběma lasery se dále nastavuje zpoždění a všechny změny potvrdíme tlačítkem „Save changes“.

Modul Ventilace bude předběžně obsahovat výběr z několika druhů plynů, kterými se bude plnit komora. Tímto se dosáhne změny tlaku v komoře a zároveň se změní prostředí pro vytváření mikroplazmatu. Tento modul zde bude za účelem přesnějšího měření. . . Momentálně však jako jediný není k dispozici.

3.2.2 Funkce programu

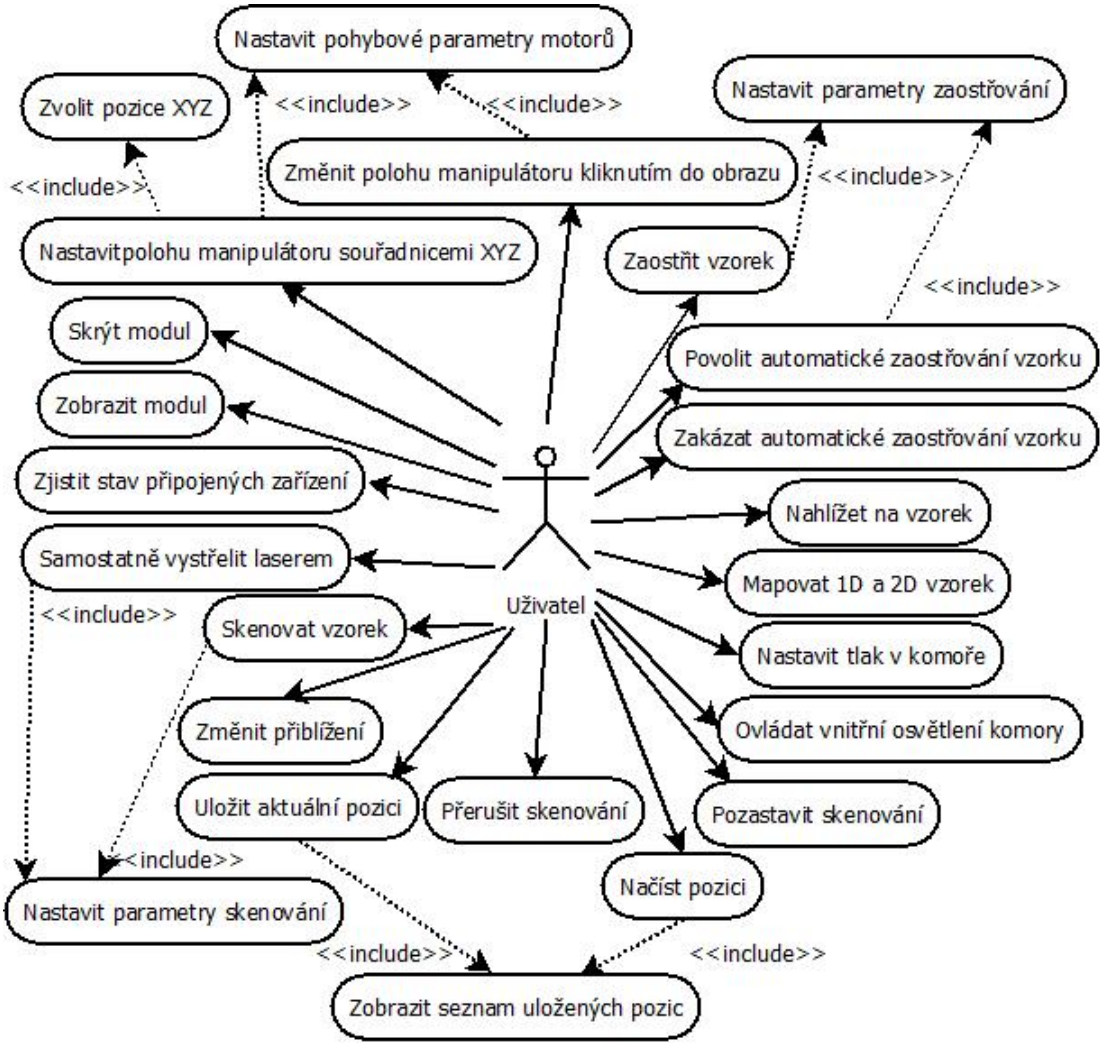
Program lze spustit pouze pod jednou rolí, a to je uživatel aplikace, který má možnost s programem pracovat takto:

- Může měnit polohu manipulátoru kliknutím do obrazu, nebo nastavením souřadnic X, Y, Z (čemuž předchází zvolení pozic X, Y, Z). U obou možností je potřeba mít nastavené požadované pohybové parametry motorů (možná budou nastaveny implicitně a nepůjdou změnit).
- Dále si může uživatel nastavit automatické zaostřování vzorku nebo si ho může zaostřovat sám. Opět oběma funkcím předchází nastavení parametrů pro zaostřování.
- Může si namapovat 1D nebo 2D vzorek pro skenování, má možnost nahlížet na vzorek, nastavovat tlak v komoře pro přesnější rozpoznání zkoumané látky a intenzitu osvětlení uvnitř komory je také možné měnit v zájmu uživatele.
- Co se týče vlastního skenování, lze sken pozastavit či úplně zrušit. Dále je možné si kdykoliv uložit pozici aktuálního skenovaného bodu, nebo si takovou pozici načíst ze seznamu uložených. Kromě skenování si uživatel může vystřelovat laserem samostatně pouze do jednoho bodu.
- Zaplacené moduly může skrývat a znovu zobrazovat a pokud bude chtít, zobrazí si stav připojených zařízení.

Po každém dokončeném skenování (ať už je to skenování 2D matice či 1D úsečky nebo jednotlivý vystřel laseru do jednoho bodu) se spustí detekce spektroskopem a vzorek je možné vyhodnotit.

Use Case diagram[20] můžete vidět na obrázku 3.4.

[20] Diagram případů užití



Obrázek 3.4: Use Case diagram

3.2.3 Návrh tříd

Hlavní třídou bude `Libs`, která bude dědit od třídy knihovny QT, `QMainWIndow`. Je to hlavní okno, které se zobrazí při startu aplikace a veškeré další akce v aplikaci se přesouvají právě do vlákna s tímto oknem. Jelikož aplikace musí být schopna multitaskingu[21], v této třídě vytvořím několik objektů třídy `QThread`, do kterých budu později vytvářet procesy (za běhu aplikace). Tato vlákna budou například pro načítání hardwarových komponent – `loading_thread` a `loading_worker`, pro průběh samostatného skenovacího procesu – `scanning_thread` a `scanning_worker`, dále pro vykreslování obrazu z kamery – `rendering_thread`, a kvůli tomu, že si uživatel může sám zaostřovat vzorek, tu bude i vlákno pro zaostřování – `focus_thread` a `focus_worker`. Vlákno časovače – `timer_thread`, do kterého se přesune například `loading_timer`, bude také nutnou součástí této třídy.

Hlavní třída `Libs` se bude skládat ze tříd `Pulse_generator` a `Libs_modules`. Třída `Pulse_generator` obsahuje atributy, které vyjadřují zpoždění, za jaké bude vysílán impulz na rozhraní laserů (například `t_q_switch_sollar`) nebo na rozhraní detektoru – `t_detector`. `Libs_modules` může obsahovat hardwarové komponenty, které vyjadřují třídy `Motor, Camera, Laser_aparature` a mapovací modul třídy `Mapping`. O HW zařízeních budeme moci zjistit informace pomocí metody `get_hw_info()` nebo můžeme zařízení připojit či odpojit...

Třída `Motor` sestává ze třídy `Serial` a `Actual_position`. Pro svoji komunikaci s driverem bude využívat parametry `command` – příkaz k odeslání po sériovém portu `comm` a `response` pro příjem odpovědi na stejném portu. Dále zde vytvořím privátní metodu `execute()`, která bude mít za úkol veškerou práci s odesláním a zpracováním dat na sériovém portu a metody pro práci s motorem, jako jsou například `rotate(), stop()`...

Další součástí třídy `Libs_modules` může být třída `Camera`. Ta bude provádět komunikaci se svým driverem pomocí `cam_handler`u a video zobrazí do okna `display`. Parametry kamery vyjadřují atributy `height, width` a posloupnost načítání obrazů ukládá do `image_memory`. `Kamera` zná metody `start_camera` pro inicializaci kamery, `render_bitmap` pro vykreslování videa a další...

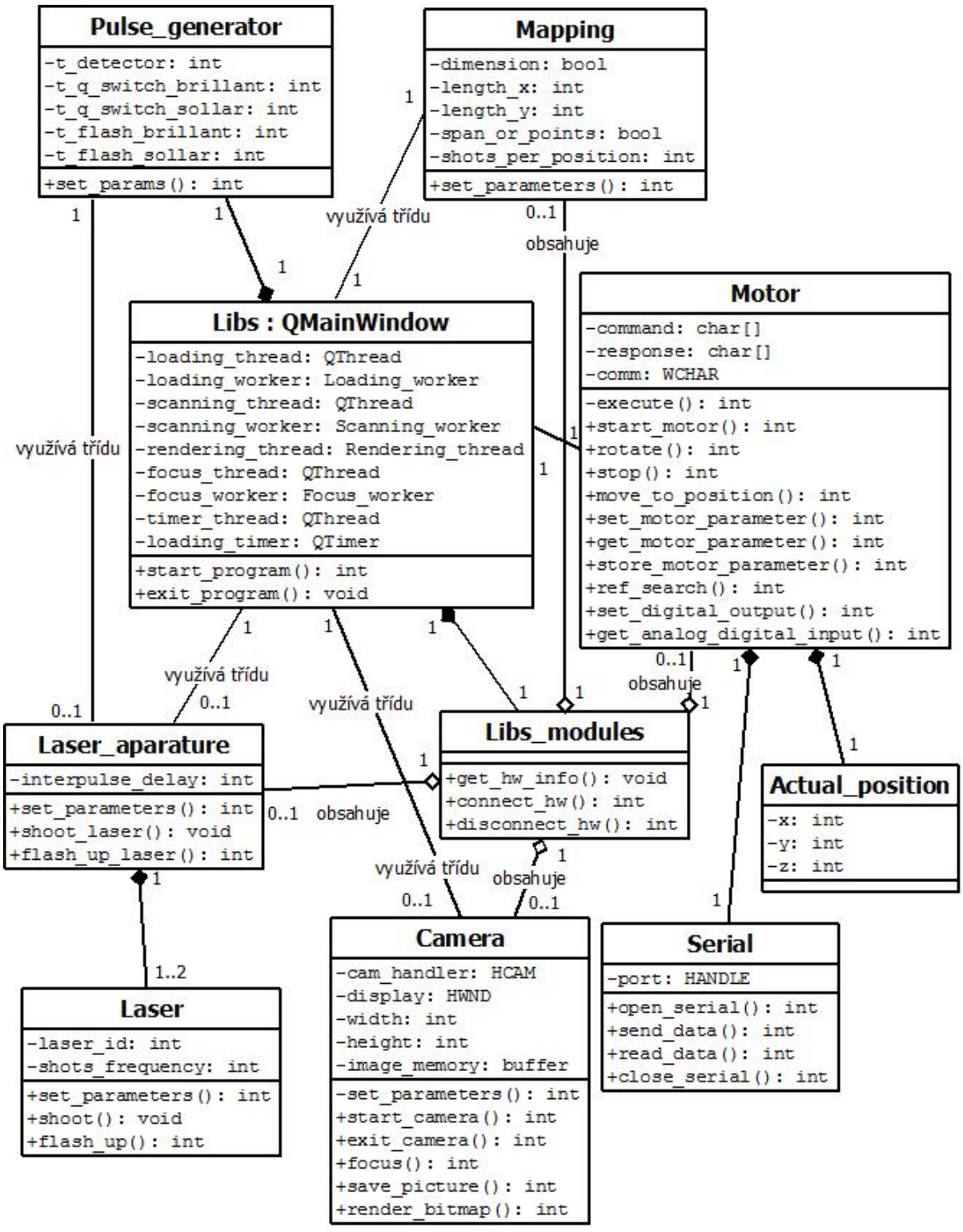
`Laser_aparature` se skládá z jednoho či ze dvou laserů. Třída `Laser` obsahuje atributy `laser_id, shots_frequency` a dále zná metody například pro výstřel laseru `shoot()` a pro nabití flash lampy `flash_up()`. Samotná aparatura pak bude nastavovat `interpulse_delay` a volat metody laserů pomocí metod `shoot_laser()` a `flash_up_laser()`.

`Mapping` je modul nezbytný pro chod skenování. Můžeme zde nastavit parametry jako je například typ dimenze v atributu `dimension`, délku skenování v ose X a v ose Y – `length_x, length_y`, zda se bude skenovat podle rozteče nebo počtu bodů – `span_or_points` a další...

Modul Ventilace pro nastavování tlaku v komoře zatím neuvažujeme. Podrobnější informace o návrhu tříd můžete vidět na následujícím obrázku 3.5.

Poznámka: Při navrhování aplikace (jak GUI, tak logiky) bylo zadání aplikace velmi volné a nepřesné. V průběhu vývoje se zadání dynamicky měnilo a bylo potřeba návrh těmto změnám přizpůsobovat (původní návrh je také slabý, protože nebylo mnoho z čeho čerpat). Proto se většina souborů nyní jmenuje jinak a aplikace se rapidně rozšířila. Logika a podstata však zůstává stále stejná.

[21] Schopnost provádět více procesů současně



Obrázek 3.5: Diagram tříd

3.3 Algoritmus pro autofokusu

V této sekci představím mnou implementovaný algoritmus pro nalezení nejostřejšího snímku. Nejvíce jsem čerpala z materiálů, které jsou popsány v části 2.2.5 této práce. Před přímým vysvětlením algoritmu ještě zmíním několik důležitých informací.

Moje implementace nejvíce odpovída vyhledávání založenému na pravidlech, viz 2.2.5. Celý algoritmus je implementovaný v nekonečné smyčce `while()`, která má ovšem svůj timeout. Jelikož mi odpadla starost s vícehladinovými vzorky (není ani jisté, zda tento problém bude potřeba řešit), nemusím po nalezení peaku prohledávat zbytek osy. Záchytným bodem je tedy nalezení vrcholové oblasti a poté nalezení peaku. Je nutné si však uvědomit, že při spuštění autofokusu nikdy nevíme, kde se vzhledem ke křivce ostrosti nacházíme (zda jsme na jednom z okrajů, zda jsme těsně před vrcholovou oblastí či přímo v ní)... Dále je ještě vhodné hned na začátku zmínit, že jsem si průběh rozdělila do několika stavů: `initial` – stav, ve kterém se nacházím hned při startu autofokusu, `middle` – stav kdy nejsem ani na startu, ani ve vrcholové oblasti, `fine` – našli jsme vrcholovou oblast, `closeSampling` – vzorkujeme ve vrcholové oblasti s docela malým krokem a konečně `finalPhase`, kdy děláme nejdrobnější kroky za celé zaměřování a jsme nejblíže peaku.

Nyní přejděme k samotnému algoritmu. Opakuj do nekonečna (respektive do vypršení timeoutu):

1. Získej hodnotu ostrosti aktuálního snímku
2. Pokud jsi ve stavu `initial`, velikost kroku posuvu bude 6000 a
   - pokud jsi zde poprvé, zjisti směr, kterým se vydáš (tam, ke kterému okraji je to dál) – `up`/`down`,
   - pokud jsi zde podruhé, kontroluj, zda hned nejsi na vrcholové oblasti (předchozí hodnota ostrosti je několikanásobně[22] menší než aktuální),pokud ano, změň stav na `fine`,
   - pokud jsi zde potřetí, zkontroluj, zda nejdeme na špatnou stranu (předpředposlední[23] získaná hodnota ostrosti je větší jak poslední získaná == aktuální), pokud ano, změň směr a proveď rollback[24], jinak změň stav na `middle`
3. Pokud jsi ve stavu `middle`, velikost kroku je 9000 a
   - kontroluj, zda nebyla nalezena vrcholová oblast, pokud ano, změň stav na `fine`,
   - kontroluj, zda jsme nepřeskočili vrcholovou oblast (předpředposlední hodnota ostrosti je větší jak poslední), pokud ano, proveď rollback do prozatím pozice, kde byla nalezena největší ostrost a změň stav na `fine`,
   - pokud jsme se ocitli na jedné z okrajových pozic, vrať se na startovní pozici a změň směr (může nastat při situaci,kdy máme špatné světelné podmínky, ostrost tedy počítáme pouze z šumu)
4. Pokud jsi ve stavu `fine`

[22] konstantu je nutné vhodně zvolit vzhledem ke všem možným vzorkům

[23] porovnávám s hodnotou ob jedno proto, abych se vyhla chybě šumu, kde by při příliš drobných krocích hodnoty ostrosti kvůli šumu mohly kolísat

[24] vrať všechny hodnoty i pozici motoru do původního stavu

- pokud jsi zde poprvé, nejdřiv zjisti, kterým směrem hledat peak (nastav velikost kroku na minikrok[25]),
- pokud jsi zde podruhé a předchozí hodnota byla větší, jdeš špatným směrem (změna směru a rollback), ale za každou cenu změň stav na `closeSampling` a nastav velikost kroku na 1600

5. Pokud jsi ve stavu `closeSampling`
   - pokud je hodnota aktuálního snímku menší než předchozí, zmenši krok na 80-90% a posuň motor do protisměru, nastav krok na minikrok, změň stav na `finalPhase` a proveď opakování cyklu příkazem `continue` (provedení minikroku nám umožní ihned v první návštěvě posledního stavu zjistit, jestli jdeme správným směrem, tedy blíže k peaku)
6. Pokud jsi ve stavu `finalPhase`
   - a jsi zde poprvé, změň velikost kroku na minikrok, zkontroluj, zda se pohybuješ správným směrem, popřípadě ho změň
   - jinak nastav velikost kroku na minikrok,
   - pokud předchozí hodnota ostrosti byla větší jak aktuální, našli jsme peak, ukončujeme `while()` cyklus příkazem `break`
7. Vždy ulož maximum ze dvou hodnot (předchozí a aktuální hodnota ostrosti)
8. Podle směru nastav správné znaménko u posuvu motoru
9. Posuň motor o požadovaný krok
10. Vrať se na bod 1

Tímto cyklem získáme hodnotu a pozici peaku. Po ukončení cyklu tedy provedeme poslední krok – návrat na pozici, kde je uložena maximální hodnota ostrosti. Veškeré poměry a velikosti kroků jsou zatím nastavovány ručně v kódu a mohou se jednoduše měnit. Ještě nebyly provedeny žádné optimalizace. Není ani jisté, zda se tento algoritmus ponechá, je to zatím jen alfa verze. Do budoucna se zvažuje domyslet algoritmus s navracením se nebo aproximace maxima pomocí prokládání fukncí jistými body. Detail implementace celého algoritmu můžete vidět v souboru `autofocusworker.cpp` v metodě `doWork()`.

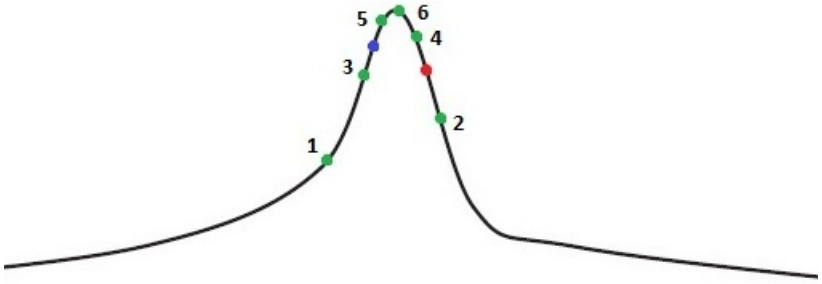
Výhody a nevýhody implementovaného algoritmu:

- algoritmus je rychlý (provádíme málo kroků) a zatím i dostatečně přesný,
- rychlé nalezení peaku i pokud startuji z vrcholové oblasti (žádné zbytečné krokování pryč od peaku),
- je těžké zvolit kompromis pro nastavování některých konstant (například pro detekci vrcholové oblasti, kde porovnávám kolikrát musí být aktuální hodnota větší než předposlední získaná hodnota),

[25] nejmenší prováděný krok který provádím pouze za účelem zjistit směr nebo při závěrečném vzorkování, abychom získali co možná nejlepší přesnost

- zatím není vyřešena detekce vícehladinových vzorků (nebylo v popisu práce, ale jestli bude, nejspíše by se za vrcholovou oblastí zvětšil krok a dovzorkoval by se zbytek celé osy),
- při široké vrcholové oblasti provádí algoritmus ve finální fázi mnoho kroků.

Nyní se ještě vraťmě k otázce, proč jsem nepoužila algoritmus s navracením se (binární vyhledávání) ve vrcholové oblasti. Počítáme s tím, že jsme tedy našli vrcholovou oblast a nacházíme se například na pozici 1 v zeleném bodě na následujícím obrázku 3.6 . Zároveň jsme zvolili velikost kroků takovou, abychom opravdu kroky nedrobili (třeba 75% předchozího kroku) a tím jakoby chceme zmenšit počet kroků pro nalezení peaku. Dále máme zvolenou nějakou odchylku, které když se rovná rozdíl 2 posledních hodnot, tak vzorkování končí.



Obrázek 3.6: Problémy při vyhledávání peaku s navracením se

Na obrázku si můžeme povšimnout hned několika problémů. Zelené body s čísly nám říkají, v jakém pořadí bychom vzorkovali vrcholovou oblast v ideálním stavu. S tím ovšem nemůžeme počítat. Všechny vrcholové oblasti jsou u různých vzorků různě široké. Při každém následujícím kroku se nám může stát, že už nepřeskočíme peak. Museli bychom tedy ke každému kroku připočítat jeden další, který zjistí, ze které strany se k peaku blížíme (na obrázku bychom například hned při druhém kroku zasáhli modrý bod místo zeleného s číslem 2). Dále by mohl nastat problém s odchylkou. Té totiž můžeme dosáhnout i tehdy, pokud bude vrcholová oblast příliš úzká a přeskočíme jedním krokem celý vrchol a zasáhneme do bodu se stejnou (podobnou) hodnotou, jen z druhé strany peaku. Na obrázku je tento problém naznačen červenou tečkou, kdy provádíme čtvrtý krok (místo zeleného bodu s číslem 4 bychom trefili červenou tečku). V tu chvíli by se nám podmínka odchylky vyhodnotila jako splněna a ukončili bychom cyklus. Ovšem peak by byl naprosto netknutý. Částečné východisko z tohoto problému by bylo například vyhodnotit peak uprostřed mezi posledními 2 body. Tím bychom ovšem ztráceli výhodu binárního vyhledávání, kterou je právě možnost získání nejpřesnějšího výsledku dle požadované odchylky. Přesto jsem tuto metodu implementovala, ale při nutném krokování navíc kvůli zjišťování, na které straně peaku se nacházím, se provádělo velké množství zbytečných kroků. Navíc se často stávalo, že jsem narazila na druhý zmiňovaný problém a aplikovalo se půlení vzdálenosti mezi posledními 2 body a výsledek byl tedy příliš nepřesný. Původní algoritmus předvedl mnohem lepší výsledky, a proto byl prozatím zvolen. Část tohoto algoritmu je ponechána v komentáři také v souboru `autofocusworker.cpp` v metodě `doWork()`.

Poznámka: Algoritmus určitě není ve finální fázi. Jedním z nevyřešených problémů

zustává šum v nesprávně osvětleném snímku, ze kterého se momentálně implementovaný algoritmus dostane jen výjimečně.

3.4 Ostatní důležité části programu

Tato práce je soustředěna především na zpracování obrazu a autofokus. To ale není vše, co tato aplikace obsahuje. V krátkosti zde ještě představím, jaká byla náplň programování mimo autofokus.

3.4.1 GUI

Náhledy GUI jsou k dispozici v přílohách, viz A.3 a A.4. Co se týče implementace, tak zde jen zmíním pár zajímavých použitých tříd a událostí. Samozřejmostí jsou třídy jako `QPushButton`, `QMainWindow`, `QProgressBar`, `QActionGroup`, `QDockWidget`... Pro vykreslování do scény byly použity například třídy `QPixMap`, `QPointF`, `QGraphicsScene`, `QGraphicsView`, `QGraphicsRectItem`. Pro vytvoření více vláken jsem použila `QThread`. Pro práci s registry jsem využila knihovnu `QSettings`. Jelikož je program velmi reaktivní na kurzor, implementovala jsem také několik událostí, například `mousePressEvent()` a `mouseMoveEvent()`. Zajímavým spojením nejen GUI prvků je například hlavní funkce programu, a tou je mapování[26]. Uživatel si zvolí body (zatím je funkční jen matice). Mapa bodů se vykreslí do scény a body pro střelbu laseru se vypíšou do tabulky, viz A.4. Při spuštění mapování se přepočítávají souřadnice a překresluje se mapa. Z důvodů několika systémů souřadnic (pro náhled zrovna systémy pixelů a milimetrů) je také zajímavé, kolik bylo potřeba implementovat metod na pouhé přepočítávání mezi systémy...

3.4.2 Motory

Pro motory bylo nutné implementovat komunikaci přes sběrnici RS232. Zde jsem tedy implementovala nejprve sériovou komunikaci pro port COM[27], viz soubory `serial.h` a `serial.cpp`. Poté přišly na řadu metody (příkazy), které jsou popsány C v manuálu [18]. Ty můžete zhlédnout ve zdrojových kódech `tmcmdriver.h` a `tmcmdriver.cpp`. Jsou zde metody jako například: `moveMotorToPosition()` – relativní/absolutní pozicování motorů, `setMotorParameter()` – nastavení parametrů motorů a další...

3.4.3 Kamera

Pro kameru jsem pouze zprovoznila ovladače a přidala cestu ke knihovnám, které jsou přístupné od výrobců kamery [17]. K implementaci kamery tedy bylo jen potřeba volat několik inicializačních funkcí, jako například `is_InitCamera()`, `is_AllocImageMem()`, `is_SetImageMem()`... Poté je možné za běhu programu používat `is_CopyImageMem()` pro zkopírování získaných dat do paměti a další. Podrobnosti naleznete v souborech `thorlabscaleradriver.h` a `thorlabscameradriver.cpp`.

[26] Funkce, kdy program pohybuje s motory po zvolených bodech uživatelem, střílí do nich laser a probíhá detekce materiálů (par), jednoduše se dá říct, že laser vytváří mapy do vzorku

[27] Komunikační port

Kapitola 4

Testování

Testování samozřejmě probíhalo v celém průběhu programování aplikace a mezivýsledky byly potom použity pro nastavování některých konstant (například při aplikaci Gaussova filtru, viz kapitola 2.2.4 – efektivní poloměr). Dále bylo potřeba rozsáhlého testování při nastavování kamery, kde byly i potíže s tím, že při předvádění programu sponzorovi bylo nutné hodnoty opět měnit, protože cílové zařízení nebylo příliš výkonné. A vlastně při každém dokončení menšího celku (ať už je to kominukace po sériovém portu, modul kamery či jakákoliv část autofokusu) bylo nutné testy provádět a sjednocovat různá nastavení a parametry. V této kapitole představím testy především pro **vyhodnocení ostrosti snímku** a poté testy funkčnosti celého **autofokusu**. Ke konci kapitoly se podíváme, jak dopadly testy k hodnocení GUI.

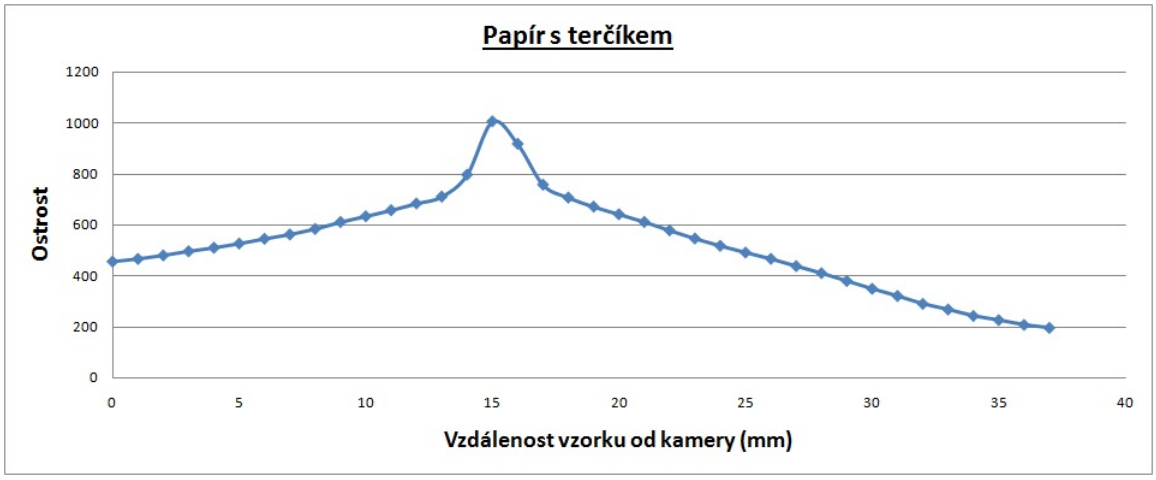
4.1 Vyhodnocení ostrosti snímku

Podívejme se tedy, jak vypadají grafy ostrosti pro různé vzorky (měření hodnot ostrosti probíhalo po 1mm posuvu v ose Z).Rozmezí výsledných hodnot ostrosti je relativní vzhledem k velikosti snímané matice a také k typu použitého filtru, nastavení expozice kamery... Počítejme tedy, že pro toto testování byly použity tyto důležité parametry:

- velikost matice pro výpočet byla 512x512 pixelů,
- velikost filtru také 512x512 bodů,
- hustota[28] pixelů pro výpočty byla maximální (nastaveno zatím implicitně, do budoucna bude možnost nastavení),
- efektivní poloměr 128 pixelů.

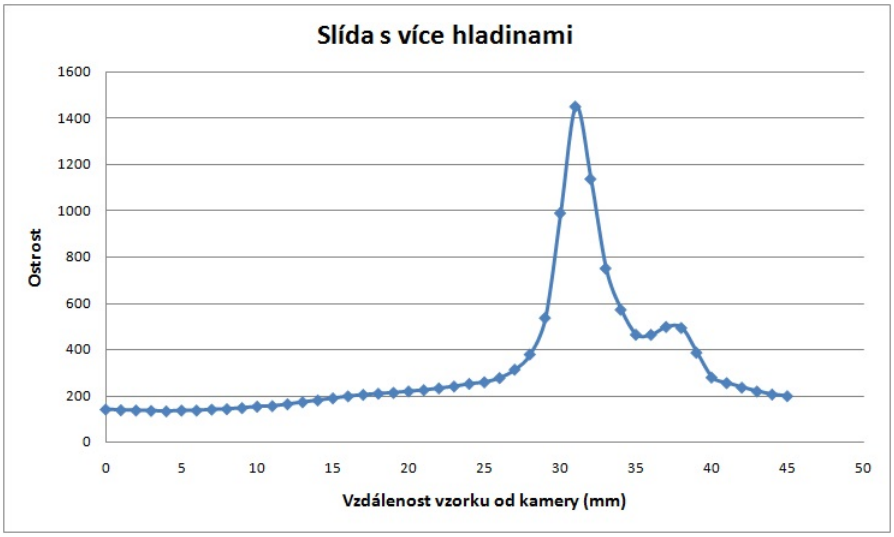
Na prvním vzorku (což byl obyčejný **papír s potiskem terčíku**), viz 4.1, vidíme, že hodnoty mimo vrcholovou oblast docela prudce rostou a klesají. Navíc mezi nejvyšší hodnotou ve vrcholové oblasti (tzv. **peaku**) a hodnotou mimo tuto oblast není tak rapidní rozdíl, jako tomu je na obecné křivce na obrázku 2.14. Hodnoty ostrosti se pohybují zhruba v rozmezí 200 až 1000.

[28] hustota pixelů říká, zda budou při zpracování matice některé pixely vynechány pro urychlení výpočtu – například každý druhý pixel může být nezapočten



Obrázek 4.1: Papír

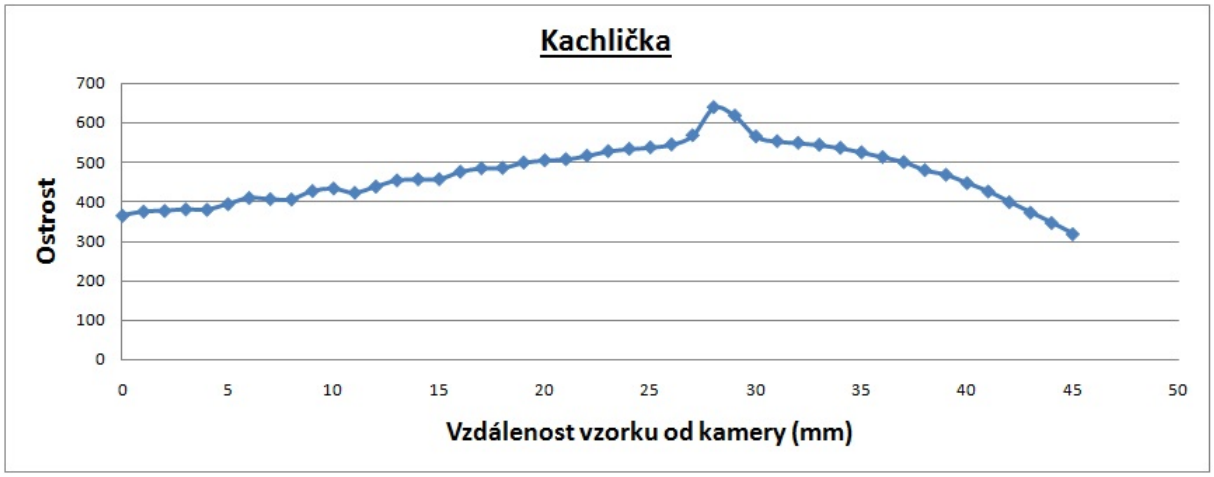
Dalším vzorkem byla **slída**.Ta byla speciální v tom, že ve vzorku byla vydlabaná díra. Musela jsem tedy zaostřovat přes 2 hladiny hloubky, což je vidět i v grafu 4.2. Vpravo od vrcholové oblasti je totiž detekována právě druhá hloubka. Takovýto problém byl ale odložen do budoucna, tím se tedy v této práci nebudu zabývat. Co je ale ještě zvláštní v tomto případě, jsou hodnoty v rozmezí 0 až 10 milimetrů. Zde nastala situace, kdy byl vzorek velmi málo osvětlen. Ostrost snímku tedy byla vyhodnocována pouze z černého obrazu a šumu. Proto hodnoty zhruba od $0mm$ do $5mm$ klesají i přes to, že se blížíme k vrcholové oblasti. Hodnoty ostrosti se pohybují mezi čísly 133 až 1451.



Obrázek 4.2: Slída

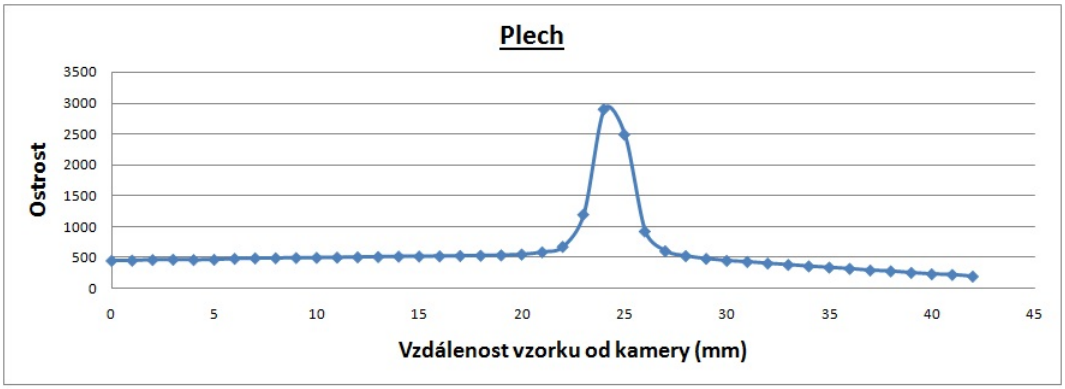
Úlomek kachličky je pro nás jeden z nejhorších a zároveň nejdůležitějších (pro algoritmus autofokusu) vzorků. A to proto, že právě u tohoto vzorku 4.3 vidíme, že může nastat situace, kdy předchozí hodnota bude větší než následující, a přitom se budeme stále přibližovat k peaku. Tento jev můžeme vidět nejlépe na obloučku nad desátým milimetrem. Z hlediska rychlosti a přesnosti zaostření to pro nás bude nejhorší vzorek. Dále vidíme, že peak vyčnívá velice málo. To vše je způsobeno tím, že má kachlička lesklý povrch – odráží

tedy světlo skoro jako zrcadlo a zároveň je tak hladká, že se v ní těžko hledá nějaká hrubší struktura povrchu. Za peakem už není struktura kopečků tak zřejmá nejspíše proto, že už tam není k dispozici tolik množství světla k osvětlení vzorku. Hodnoty ostrosti se pohybují ve velmi malém rozmezí, a to zhruba od 316 do 640.



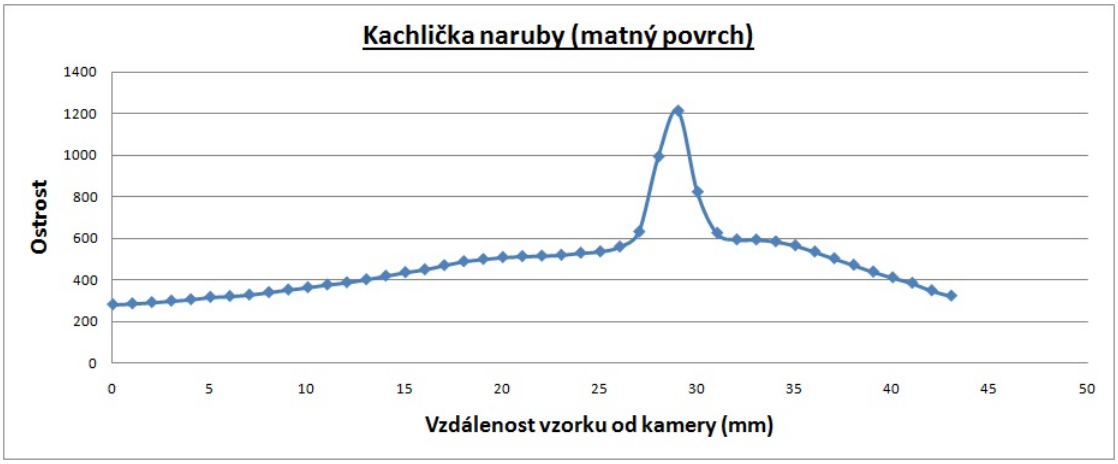
Obrázek 4.3: Úlomek kachličky

Nejideálnější chování podle očekávání z obrázku 2.14 má obyčejný hnědo-stříbrný **plech**. Ačkoliv se může zdát, že by mohl být moc hladký, kamera snímá takový detail, že najde v plechu jemně hrubou strukturu (v kachličce ne,ta je úplný extrém). Na obrázku 4.4 vidíme, že po celou cestu k peaku hodnoty mírně rostou. Poté hodnoty vyšplhají do extrému, a to až do **pětinásobku** svých hodnot ve střední oblasti. Za vrcholovou oblastí hodnoty opět klesnou na ty velmi nízké a pomalu zase klesají. Ve střední oblasti se hodnoty pohybují kolem 500 a peak je nalezen na hodnotě skoro 3000.



Obrázek 4.4: Plech

Poslední vzorek, který zde zmíním, je opět **úlomek kachličky**, ale otočená **naruby**. Tam už má pěkný zachytitelný matný vzorek. Jeho průběh ostrosti je zobrazen na následujícím obrázku 4.5. Křivka má velmi podobný charakter jako vzorek slídy (když nepočítáme vetší množství hladin), viz 4.2. Hodnoty ostrosti tedy zvolna narůstají a až dosáhnou peaku, tak zase pozvolna klesají. Rozpětí hodnot je mezi 300 a 1200.



Obrázek 4.5: Matná kachlička (naruby)

Další testované vzorky měly velmi podobný charakter těm, které jsem již představila, a proto jsem je zde podrobněji nerozváděla. Většina vzorků je podobna obecné křivce ostrosti, viz 2.14.

Závěr měření: Ze všech výše zmíněných měření tedy vyplívá již zmíněná teorie, že jsou hodnoty relativní. Nevíme tedy, jaké bude jejich maximum ani minimum. Dále víme, že není vzorek jako vzorek a při hledání peaku je potřeba zohlednit několik zjištěných skutečností a problémů, které mohou nastat.

4.2 Testování algoritmu autofokusu

Jak už jsem zmínila v sekci Algoritmus pro autofokus 3.3, rychlejší nalezení peaku předvedl můj implementovaný algoritmus před algoritmem s navracením se (binární vyhledávání). Pro následující měření jsem použila stejné vzorky jako v kapitole Testování – Vyhodnocení ostrosti snímku 4.1. Měřila jsem **počet kroků**, kolik bylo potřeba k nalezení peaku z různých pozic rozostření. Pro každý vzorek jsem provedla 10 měření na každou vzdálenost. V tabulkách jsou v záhlaví označeny jednotlivé pokusy měření, v nejlevějším sloupci jsou typy vzorků a tabulka je potom v každé datové buňce vyplněna právě počtem potřebných kroků k nalezení peaku.

	1.	**2.**	**3.**	**4.**	**5.**	**6.**	**7.**	**8.**	**9.**	**10.**
Papír a terčíkem	16	16	16	16	16	16	16	16	16	16
Slída s více hladinami	23	23	23	23	23	23	23	23	23	23
Kachlička	14	15	14	15	14	14	14	15	14	14
Plech	13	12	12	12	13	12	12	12	12	13
Kachlička naruby	12	11	10	11	11	11	11	12	11	11

Tabulka 4.1: Počet kroků pro zaostření z co nejvíce rozostřené startovní pozice

První tabulka 4.1 nám ukazuje, kolik bylo potřeba kroků pro zaostření z co nejvíce rozostřené startovní pozice (z kraje). Aby byly výsledky měření rovnocenné, byly nastaveny

startovní pozice tak, že jsem u každého typu vzorku našla peak, a poté jsem posunula motor osy Z o určitou konstantu, která byla u všech vzorků stejná.

Nejvyššího počtu kroků nabývá zaostřování na vzorku slídy. Je to způsobené tím, že křivka ostrosti u tohoto vzorku má velmi širokou vrcholovou oblast. Proto, když se dostaneme do stavu `finalPhase`, je zde zapotřebí mnoho drobných kroků.

Hned na druhém a třetím místě, co se týče nejvyššího počtu kroků k nalezení peaku, se dostaly vzorky papíru a lesklé kachličky. A to proto, že zvolená konstanta, která nám detekuje několikanásobný rozdíl hodnot ostrosti při přechodu ze stavu `middle` do stavu `fine`, je příliš velká (u těchto vzorků nenabývá peak tak vysokých hodnot vzhledem k hodnotám ve střední oblasti, viz 4.3 a 4.1). Algoritmus tedy nezaregistruje vrcholovou oblast, nýbrž ji ignoruje a teprve po detekci klesání hodnot ostrosti se do vrcholové oblasti vrátí.

Plech a kachlička naruby nám předvedly nejlepší výsledky. A to proto, že detekce vrcholové oblasti byla provedena okamžitě v prvním průchodu a vrcholové oblasti u těchto vzorků nejsou natolik široké, aby se provádělo příliš drobných kroků.

Dále mezi hodnotami u některých vzorků můžeme sledovat kolísání kroků kolem střední hodnoty o $+-1$, což souvisí i s odchylkou měření. Může to být způsobeno například šumem kolem peaku, kde jsou si u absolutního vrcholu hodnoty velmi blízké.

	1.	**2.**	**3.**	**4.**	**5.**	**6.**	**7.**	**8.**	**9.**	**10.**
Papír a terčíkem	12	12	7	12	12	11	12	12	12	12
Slída s více hladinami	12	12	12	12	12	12	12	12	12	12
Kachlička	9	8	10	9	9	8	9	8	8	10
Plech	11	12	11	11	12	11	11	11	12	11
Kachlička naruby	11	11	12	12	11	11	11	12	11	12

Tabulka 4.2: Počet kroků pro zaostření ze startovní pozice velmi blízko peaku

Druhá tabulka 4.2 nám zobrazuje počty kroků k nalezení peaku, přičemž startovní pozice byla velmi blízko peaku. Vidíme, že počty kroků už jsou u některých vzorků výrazně nižší. Kromě stejných jevů, jako jsem popisovala u první tabulky, zde můžeme však vidět jeden extrém. Byl nalezen u vzorku papíru s terčíkem a je to počet kroků 7. Toto číslo je naprosto v pořádku, je to právě nejlepší výsledek ze všech naměřených hodnot u tohoto vzorku. Konkrétně v tomto případě totiž byla odhalena vrcholová oblast hned ve druhém kroce (vlivem šumu se hodnota těsně vešla do detekovaného intervalu). U ostatních případů odhalena nebyla a byla zaregistrována až při klesání hodnot, navrácení se do maxima a opětovném klesání hodnot na druhou stranu.

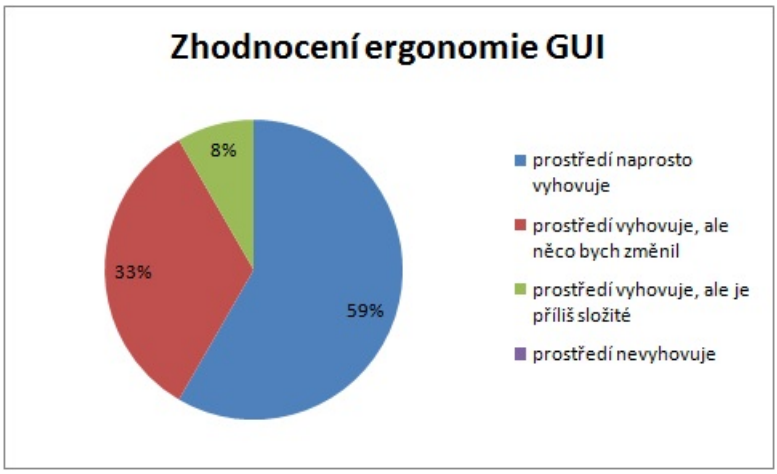
Je ale nutné podotknout, že podmínky měření nebyly úplně ideální. V místnosti se pohybovalo více lidí a jejich chůze kolem komory občas vyvolávala u komory menší otřesy, které mají na měření vliv (vzorek je v neklidném stavu). Další ovlivňující fakt je ten, že vzorek papíru je jen kus papíru, do kterého stačí mírně fouknout a už je také v neklidném stavu (komora není úplně izolovaná, viz A.2, kde vidíme otvor v komoře, kudy se do ní dávají vzorky). Snažila jsem se však tyto ovlivňující faktory maximálně eliminovat.

4.3 Zhodnocení ergonomie návrhu GUI

Ergonomii GUI jsem nechala testovat 12 osob. 4 z nich byli budoucí uživatelé. Ti měli možnost si program vyzkoušet v dosavadní funkčnosti. Ostatních 8 osob zhodnotilo GUI pouze v offline režimu, protože nebylo možné je přímo ke komoře připustit (měli k tomu ode mě navíc ústní projev). Testování probíhalo u budoucích uživatelů konzultacemi a u ostatních testerů dotazníkovou formou (i s otevřenými otázkami). Hodnocení jsem shrnula do několika bodů:

- základní ovládání je opravdu intuitivní, srozumitelné a jednoduché (logické rozmístění tlačítek a jiných objektů),
- pokročilé ovládání je také intuitivní, ale je k tomu potřeba trochu vysvětlení (budoucí manuál),
- estetické zpracování velmi jednoduché, ale příjemné (estetika přijde na řadu později),
- GUI je velmi adekvátní a účelné,
- jednoduchá editace hodnot/objektů v prostředí, ale jsou zde i velké rezervy,
- rychlá adaptace uživatelů.

GUI je také neustále ve vývoji a na složitosti teprve nabude, až bude zprovozněné veškeré nastavování. Je opravdu vhodné především pro uživatele z oboru, jinak se může zdát velmi složité. Budoucí pracovníci s tímto prostředím byli velice spokojeni. Méně informovaní měli trochu problémy, ale po řádném vysvětlení hodnotili ergonomii návrhu GUI také velmi kladně, viz následující graf 4.6.



Obrázek 4.6: Zhodnocení ergonomie GUI

Kapitola 5

Závěr

Cílem práce bylo navrhnout a implementovat aplikaci se zaměřením na autofokus. Návrh probíhal v docela těžkých podmínkách, neboť jsem měla velmi málo informací, požadavků a specifikací k provedení tohoto úkonu. Nicméně provedla jsem návrh jak logiky, tak GUI. Při programování se aplikace dynamicky měnila, dokonce jsem ji v jedné fázi pak sepsala celou znovu i v jiném prostředí, abych měla lepší podmínky pro verzování. Výsledná aplikace oproti návrhu se liší především ve vzhledu, ale návrh modulů a dalších důležitých prvků zůstal stejný. Logika programu se pouze rozšířila.

Při autofokusu už od zpracování obrazu bylo v možnostech vždy několik metod, co a jak implementovat. Rozhodování probíhalo na základě mých návrhů a požadavků od zaměstnavatele, který už to měl částečně naplánované. Implementované zpracování obrazu samozřejmě není dokonalé, jsou zde určité rezervy, které se budou měnit později v průběhu optimalizací. Samotný algoritmus autofokusu zabral nejvíce času, neboť jeho výsledky jsou v mnoha případech nevyzpytatelné, v našem smyslu slova tedy relativní. Probíhalo velmi mnoho měření na různých vzorcích, které jsme našli v laboratořích. Navíc nikdy nevíte, jaký vzorek (s jakými parametry) uživatel do komory vloží. Vymýšlení algoritmu autofokusu je tedy velmi náročné a v některých případech šlo spíše o jakousi eliminaci chybujících algoritmů, než vymýšlení toho nejefektivnějšího a nejrychlejšího. Je zde stále co zlepšovat.

Hlavní funkce programu je nyní taková: uživatel si může pohybovat vzorkem ve třech směrech. Dále si ho může zaostřovat (ručně nebo algoritmem autofokusu, který ovšem bude potřebovat do budoucna optimalizace), může měnit nastavení kamery a má možnost si zvolit a spustit mapování, které ovšem není stoprocentně dokončené. Funkční část mapování je, že se pohybují motory a celé to sleduje kamera. Chybí zde ale výstřel laseru, místo kterého se pouze vypíše do informačního okna zpráva o výstřelu a odsimuluje se jedna sekunda čekání. Stejně tak není zatím zprovozněna detekce materiálu. Další část, která chybí k dokončení aplikace, je modul vzduchotechniky, ale práce na tomto modulu je zatím nejvzdálenější. V programu zatím nejsou odchytávány žádné výjimky, protože pro funkční stránku to není důležité. Nejprve musím dostat program do fáze, kdy bude fungovat alespoň jako prototyp. Navíc jsme se ani s týmovými kolegy nedostali k tomu, na jaké úrovni by výjimky bylo potřeba ošetřovat. Implementace výjimek je tedy také odsunuta na později, spolu se spoustou detailnějších rozšíření.

Obrovskou zkušeností pro mě bylo to, že jsem si zkusila reálné pracovní prostředí. Je to úplně něco jiného, když pracujete a víte, že je ve vašich rukou technika, která má cenu několikaset tisíc. Dále jste ve všech školních projektech vždy věděli, že problémy v projektu je možné vyřešit a naimplementovat (jinak by vám ho nezadávali). V reálné praktické aplikaci toto u některých problémů zůstává stále otázkou...

Literatura

[1] BLANCHETTE, J.; SUMMERFIELD, M.: *C++ GUI Programming with Qt4*. Prentice Hall, únor 2008, ISBN 978-0132354165, 2. vydání.

[2] ECKERTOVÁ, L.; KOLAŘÍK, V.; FRANK, L.: *Metody analýzy povrchů: elektronová mikroskopie a difrakce*. Academia, 1996, ISBN 80-200-0329-0, 1. vydání.

[3] EZUST, A.; EZUST, P.: *An Introduction to Design Patterns in C++ with Qt*. Prentice Hall, září 2011, ISBN 978-0132826457, 2. vydání.

[4] FRANK, L.; KRÁL, J.: *Metody analýzy povrchů: iontové, sondové a speciální metody*. Academia, 2002, ISBN 80-200-0594-3.

[5] HÜWEL, L.: Isotherm Contours. 2011.
URL http://www.wesleyan.edu/physics/labs/mol2.html

[6] JONES, L., B.; LIBERTY, J.: *Naučte se C++ za 21 dní*. Computer press, červen 2007, ISBN 9788025115831, 2. vydání.

[7] KEHTARNAVAZ, N.: Development and real-time implementation of a rule-based auto-focus algorithm. [online]. 2003 [cit. 2014-04-22].
URL http://www.sciencedirect.com/science/article/pii/S1077201403000378

[8] Kolektiv autorů: Gaussian filter. [online]. 2013 [cit. 2014-03-27].
URL http://en.wikipedia.org/wiki/Gaussian_filter

[9] LIBS Lab, VUT Brně: LIBS and DB LIBS aparatura. [online]. 2011 [cit. 2014-1-8].
URL
http://libs.fme.vutbr.cz/index.php/en/lab-setup/libs-a-db-libs-setup

[10] LIBS Lab, VUT Brně: Metody LIBS na VUT v Brně. [online]. 2011 [cit. 2013-12-26].
URL http://libs.fme.vutbr.cz

[11] LIBS Lab, VUT Brně: Metody LIBS na VUT v Brně. [online]. 2011 [cit. 2014-01-17].
URL http://libs.fme.vutbr.cz/index.php/cs/kontakt

[12] MIZIOLEK, A.; PALLESCHI, V.; SCHECTER, I.: *Laser-Induced Breakdown Spectroscopy (LIBS) Fundamentals and Applications*. Cambridge University Press, 2006.

[13] MLÍCH, J.: Základní principy a využití knihovny Qt. [online]. 2013 [cit. 2014-1-8].
URL https:
//www.fit.vutbr.cz/study/courses/ITU/private/lectures/itu-qt-cs.pdf

[14] NOVOTNÝ, J.: *Návrh systému pro automatické nastavení vzorku do ohniskové vzdálenosti objektivu v sestavě laserové spektroskopie*. Diplomová práce, VUT Fakulta strojního inženýrství, 2007.

[15] STROUSTRUP, B.: *C++ programming language*. Addison-Wesley, květen 2013, ISBN 978-0321563842, 4. vydání.

[16] SUBEDI, B.: Gaussian Filter generation using C/C++. [online]. 2013 [cit. 2014-03-27].
URL http://www.programming-techniques.com/2013/02/gaussian-filter-generation-using-cc.html

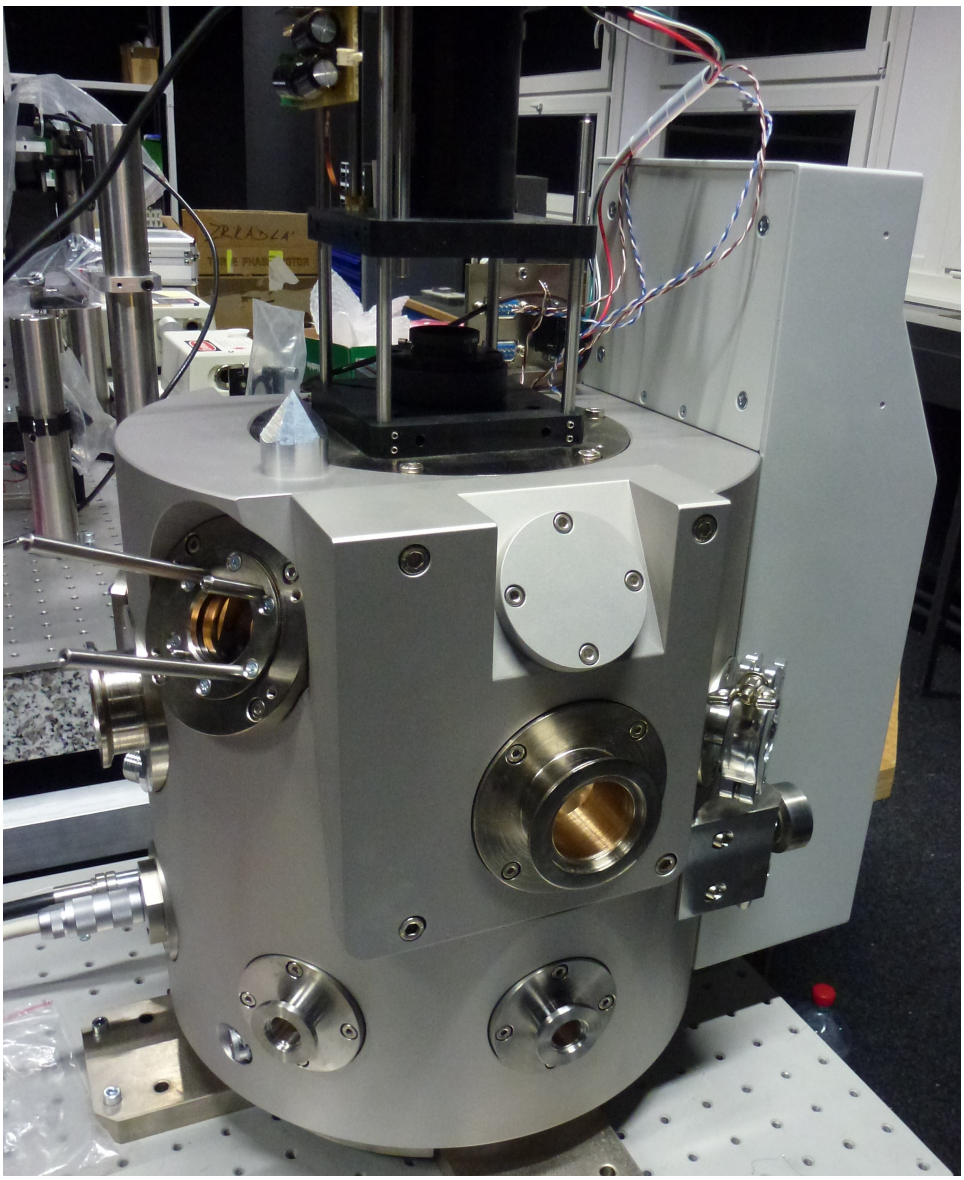
[17] THORLABS: *Thorlabs CDD and CMOS cameras*. Thorlabs, Čtvrté vydání, březen 2013.

[18] TRINAMIC: *TMCM-6110 TMCL firmware manual*. Trinamic, první vydání, prosinec 2012.

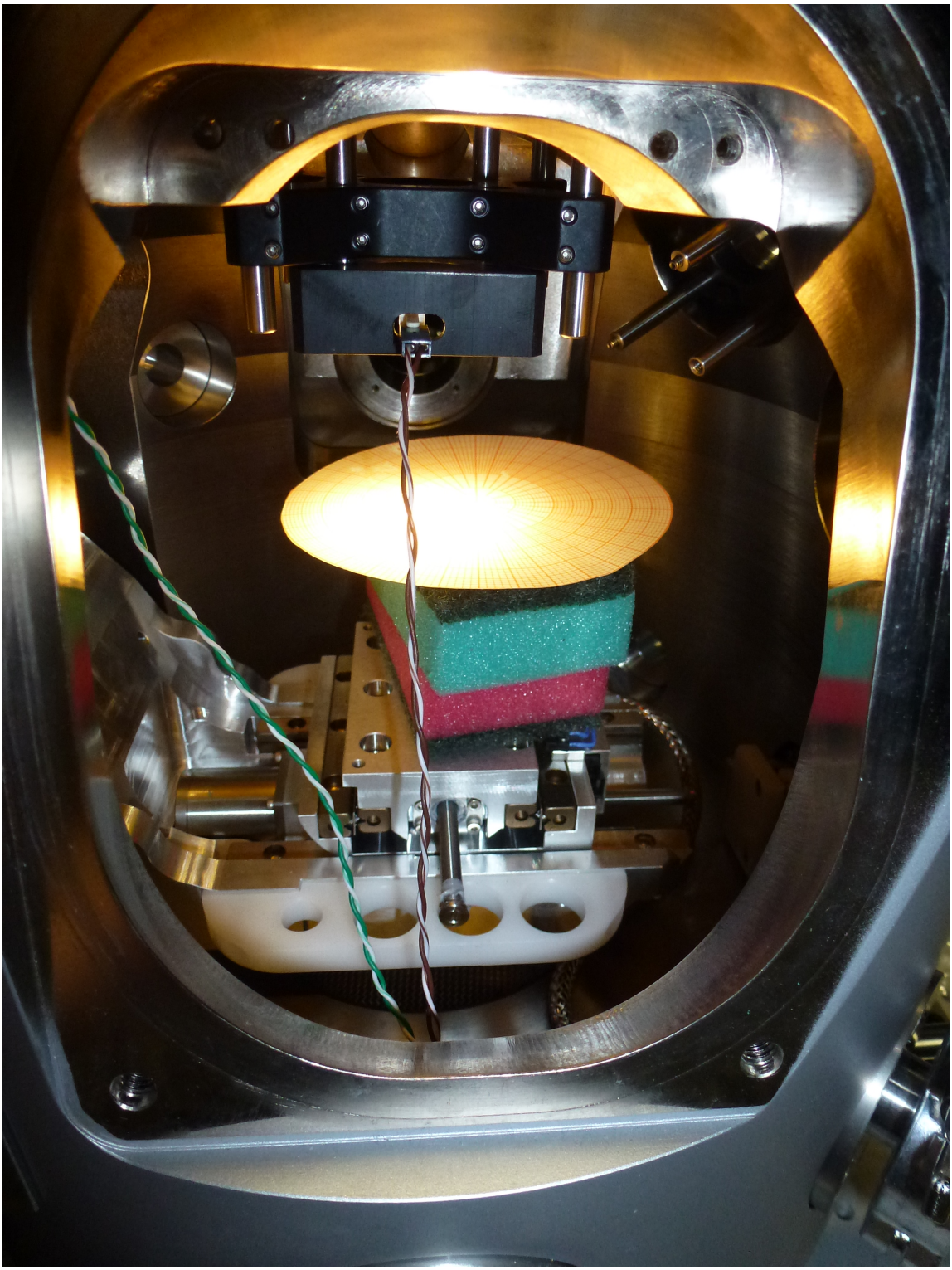
[19] ZÁMEČNÍK, J.: *Netradiční metody výpočtu difrakčních jevů v optice*. Diplomová práce, VUT Fakulta strojního inženýrství, 2004.

Dodatek A

Fotodokumentace



Obrázek A.1: Pohled na LIBS komoru



Obrázek A.2: Pohled dovnitr LIBS komory

Obrázek A.3: Ukázka GUI

Obrázek A.4: Ukázka GUI – režim mapování

Dodatek B

Návod na překlad a spuštění

Před stahováním programů si zkopírujeme celou složku `src` (ta obsahuje veškeré soubory pro překlad a spuštění aplikace) nejlépe do další složky, například `LIBS`. Budeme tedy mít hierarchii `.../LIBS/src/vsechny_potrebne_soubory`. Novou složku `LIBS` je vhodné vytvořit proto, že až sestavíme a přeložíme tuto Qt aplikace podle návodu, vytváří se spustitelné soubory a další objekty. Ty se automaticky ukládají do složky, která se vytvoří na stejné úrovni, jako je naše složka `src`.

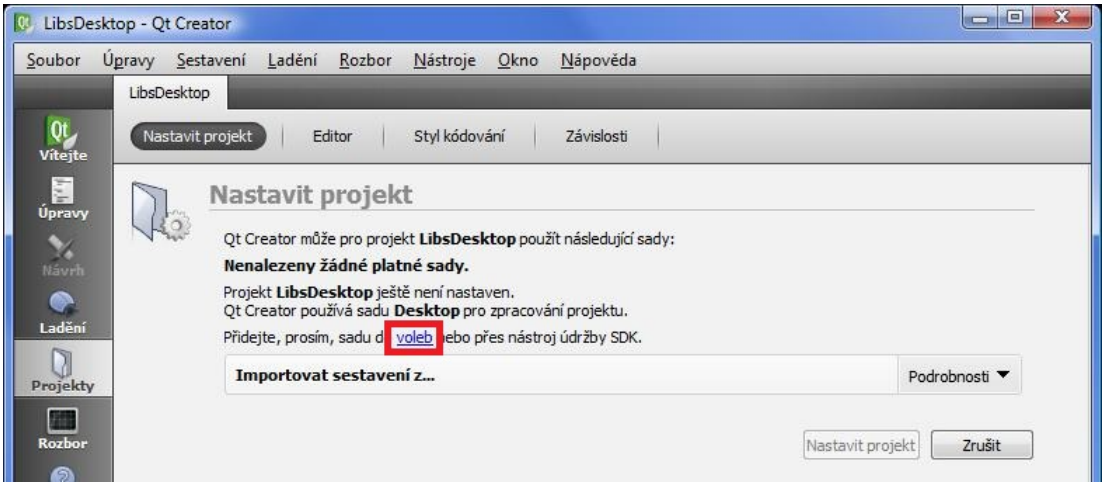
Překlad budeme provádět na operačním systému Windows, konkrétně mám otestované Windows 7 Professional, Windows Vista a Windows 7 Enterprise. K tomu budeme potřebovat několik programů.

1. Zaprvé je to QtCreator, který je potřeba kvůli souboru `LibsDesktop.pro`, díky kterému pak program sestavíme. Některé verze jsou dostupné například zde http://qt-project.org/downloads#qt-creator (po rozkliknutí „Show Downloads“)... Testováno konkrétně na verzi **Qt Creator 3.1.0 for Windows (68 MB)**.

2. Dále potřebujeme **Microsoft Visual Studio 2010 (Ultimate)**, ze kterého jsem používala překladač. Opět stačí pouze nainstalovat a zatím nejsou potřeba žádné změny.

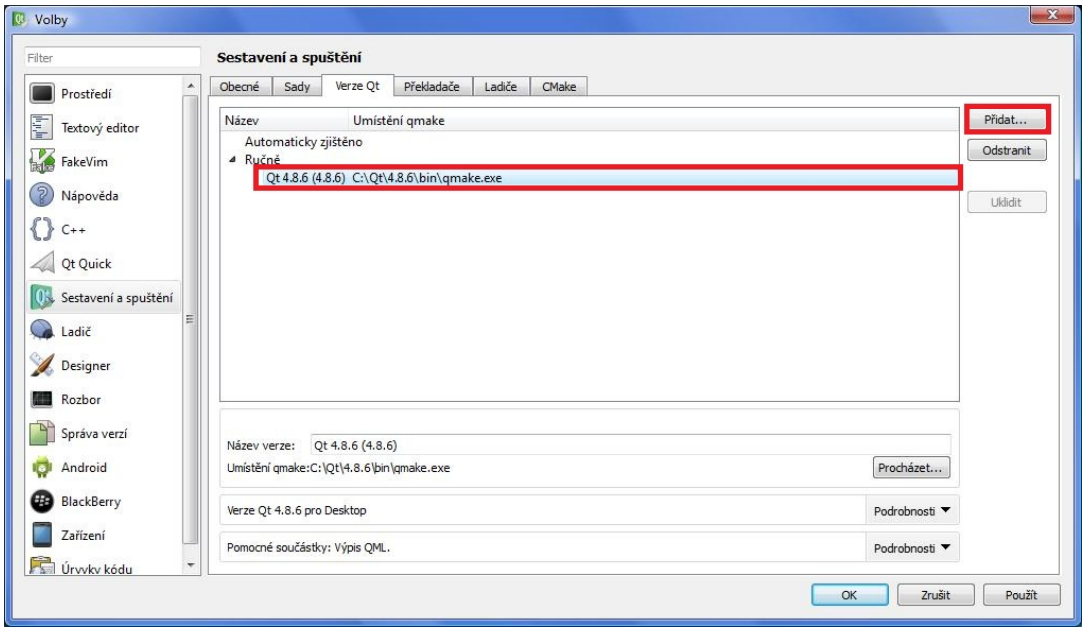
3. Jako poslední budeme potřebovat Qt knihovny. Ty jsou opět ke stažení na http://qt-project.org/downloads#qt-creator . Použijte prosím verzi knihoven **Qt libraries 4.8.6 for Windows (VS 2010, 236 MB)**, ostatní by mohly být nekompatibilní. Je vhodné neměnit cesty při instalacích (především u knihoven), aby nebyly příliš hluboko v adresářovém stromu. Knihovny by se měly automaticky nainstalovat na primární disk (například `C:\Qt`).

4. Po instalaci všech těchto tří komponent můžeme otevřít soubor `LibsDesktop.pro` (ten najdeme ve zkopírovaném adresáři `src`). Měl by se nám otevřít v Qt Creatoru. Pokud se tak nestane, otevřete prosím nejprve Qt Creator a v nabídce `Soubor -> Otevřít soubor` jej otevřete. Objeví se nám konfikurace projektu pro sestavení. Nejprve přidáme sadu do voleb, viz následující ukázka B.1. Otevře se nám okno pro editaci nastavení sestavení a spuštění.



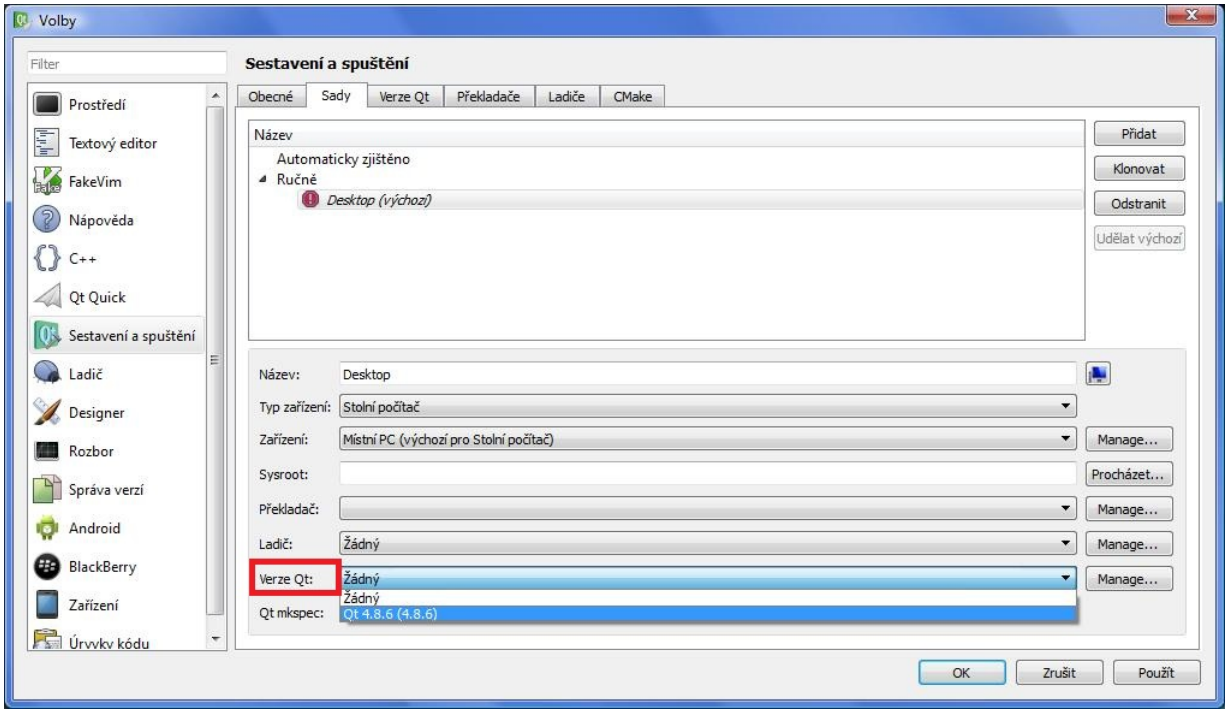
Obrázek B.1: Volby

5. V záložce Qt Verze klikneme vpravo na tlačítko Přidat a přidáme naši nainstalovanou Qt verzi. Konkrétně hledáme soubor `qmake.exe`, který by měl být přesně tam, kam jsme instalovali knihovny Qt. Cesta bude stejná nebo podobná, jako vidíme na obrázku B.2.



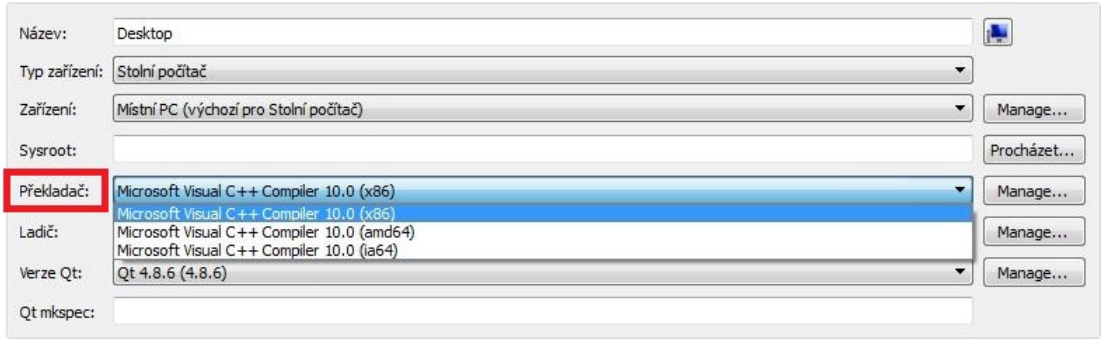
Obrázek B.2: Přidání verze Qt

6. Vrátíme se na záložku Sady a po kliknutí na `Desktop(výchozí)` vybereme ve verzích Qt (dole) naši přidanou verzi, viz další obrázek B.3.



Obrázek B.3: Výběr verze Qt

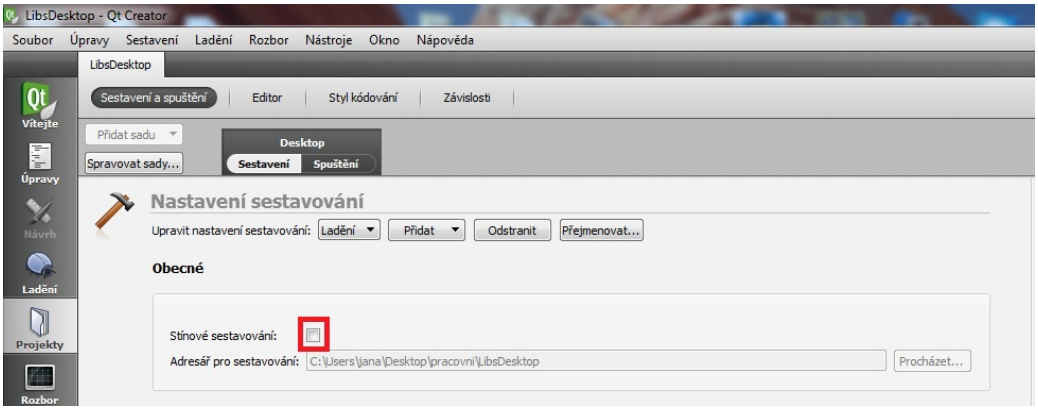
7. Hned 2 řádky nad tímto nastavením nastavíme překladač, viz B.4. Poté potvrdíme



Obrázek B.4: Výběr překladače

veškeré změny tlačítkem Ok, zaškrtneme námi nově vytvořenou sadu `Desktop` a konečně sestavíme projekt (tlačítko `Nastavit projekt`).

8. Na závěr vypneme stínové sestavování (nezatrhnuto) v záložce `Projekty -> Sestavení a spuštění`, viz B.5.



Obrázek B.5: Stínové sestavování

Projekt přeložíme kladívkem (vlevo dole) nebo klávesovou zkratkou `Ctrl + B`. Spustíme ho pak tlačítkem play či klávesovou zkratkou `Ctrl + R`.

Poznámka: Aplikace pracuje s registry v `HKEY_CURRENT_USER/Software/vutbr`