Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Lokalizace vláken na mikroskopickém snímku řezu kompozitního materiálu

Plzeň 2016 Jakub Štolle

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 22. června 2016

Jakub Štolle

Poděkování

Tímto bych rád poděkoval Ing. Kamilu Ekšteinovi, Ph.D. za vedení práce, zapůjčení knih a udělování dobrých rad během vývoje softwaru. Dále bych chtěl poděkovat rodině a přátelům za psychickou podporu během tvorby této práce.

Abstract

The purpose of this thesis was to implement an algorithm able to find circles in pictures of composite material taken by an electron microscope. This algorithm can and will be used in the application for computing the area of the matrice and the reinforcement fibres in a cut of a composite material. The application is going to be used by the Department of Mechanics at University of West Bohemia in Pilsen.

Abstrakt

Účelem této práce bylo navržení a implementace algoritmu pro rozpoznávání geometrických obrazců, konkrétně kružnic, ve snímcích kompozitního materiálu získaných pomocí elektronového mikroskopu. Navržený algoritmus bude použit v aplikaci na výpočet procentuálního obsahu vláken a matrice v řezu kompozitního materiálu. Aplikace bude využívána katedrou mechaniky na Západočeské univerzitě.

Obsah

1 Úvod

Zrak je pro přijímání informací z okolí nejvýznamnější a nejdůležitější z lidských smyslů. Už od pradávna lidé využívali různé symboly a obrazce k záznamu informací, protože je lze jednoduše rozpoznat a porovnávat lidským okem. Možnosti pořizování snímků pomocí fotoaparátů, kamer nebo mikroskopů jsou v porovnání s lidským okem rozmanitější. Navzdory tomu izolování a klasifikace objektů není ve většině případů pro stroje vůbec triviální. Už jen kvůli možnému výskytu šumu ve snímku.

Cílem práce bylo vytvořit úzce specializovanou aplikaci, která bude přijímat snímek kolmého řezu kompozitního materiálu a klasifikovat, případně označovat, vlákna sloužící jako výztuha materiálu. Úkol je usnadněn skutečností, že vlákna v tomto typu řezu nabývají podoby kruhů. Získaná informace je použita k výpočtu procentuálního obsahu vláken v daném materiálu, a tudíž i k určení kvality kompozitního materiálu. Aplikace bude sloužit katedře mechaniky, která mi také poskytla snímky pro testování pořízené pomocí elektronového mikroskopu .

Během vývoje byl kladen důraz na robustnost aplikace z důvodu velkého výskytu šumu v datech, na které bude vytvořený algoritmus aplikován. Dále byl kladen důraz na intuitivní ovládání uživatelského rozhraní aplikace, protože ne všichni uživatelé budou zběhlí v používání informačních technologií.

2 Teoretický úvod

Na úvod je potřeba vysvětlit některé pojmy, jako například co jsou kompozitní materiály, a přiblížit problematiku předzpracování a zpracování snímku. Kapitola se bude také věnovat problému tvorby histogramu, jeho různým úpravám a metodám automatického prahování. Závěr kapitoly bude věnován metodě na detekci geometrických obrazců ve snímku. Protože popisované algoritmy jsou relativně složité, předpokládá se, že čtenář má základní matematické znalosti, které budou potřeba v této části dokumentu.

2.1 Kompozitní materiál

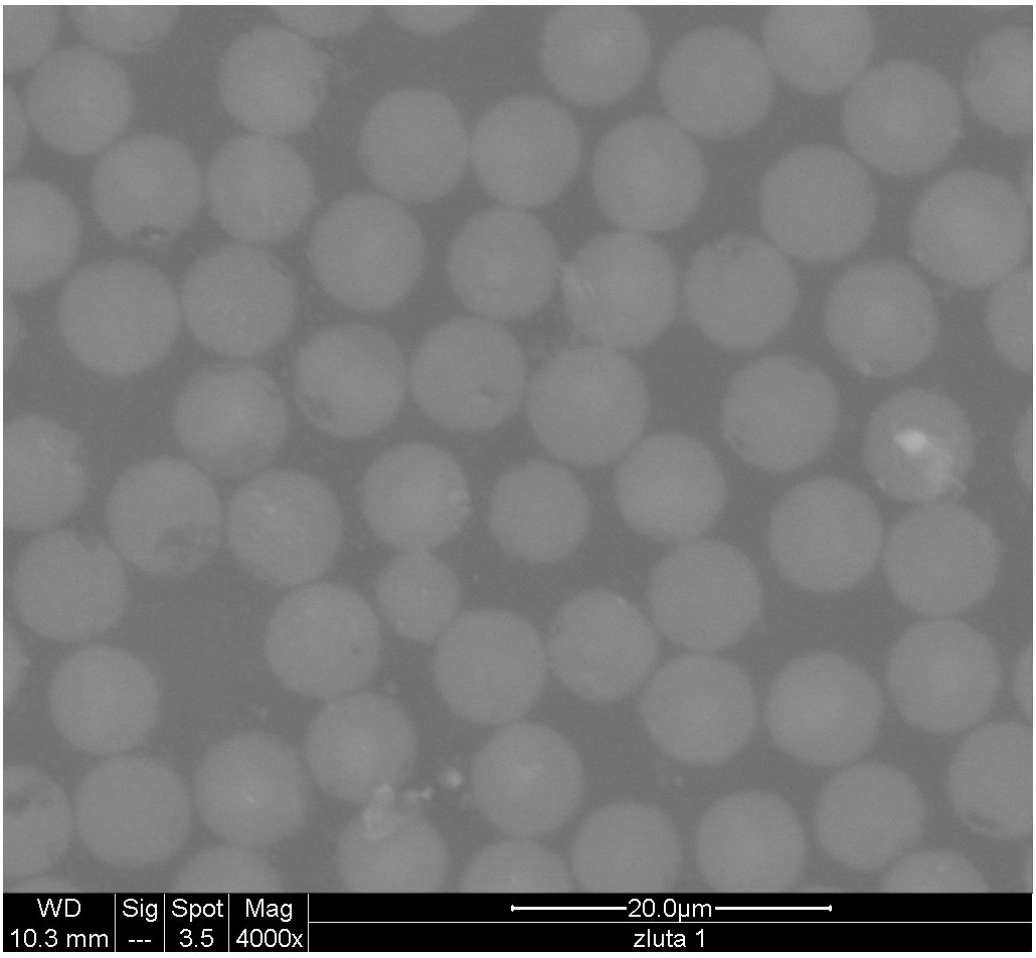
Do kompozitních materiálů řadíme látky, které jsou složeny z několika oddělitelných částí, z nichž alespoň jedna je tuhá. Jedná se tedy o nehomogenní materiál, přestože se jako technický materiál považuje za homogenní[2].
Obvykle se kompozitní materiál skládá z matrice a výztuhy. Za matrici je považována veškerá výplň mezi jednotlivými výztuhami.

2.1.1 Typy kompozitních materiálu

Kompozitní materiály se dají rozdělit podle typu matrice. Jedná se o kompozity s kovovou, keramickou nebo polymerovou matricí. Další typ rozdělení je podle struktury materiálu, kdy se dělí na makrokompozity, mikrokompozity a nanokompozity. Poslední type rozdělení je pro nás nejzajímavější, protože se jedná o rozdělení podle typu výztuh. Takto rozdělené kompozity mají dvě kategorie. Těmi jsou částicové a vláknové kompozitní materiály.

Vlákny vyztužený kompozitní materiál

Jedná se kompozit, jehož výztuží jsou různá umělohmotná nebo přírodní vlákna. Tento typ kompozitu se využívá hlavně ve stavebnictví a jedna z jeho největších výhod vzhledem k ostatním kompozitním materiálům je možnost až dvacetinásobné recyklace. Kvalitu materiálu určuje hustota vláken, jejíž výpočet je náplní práce. Příklad snímku materiálu zobrazuje obrázek 2.1.



Obrázek 2.1: Kolmý řez kompozitního materiálu

2.2 Předzpracování snímku

Většinu algoritmů zpracování snímku nelze z důvodu špatného formátu barev nebo výskytu šumu aplikovat na neupravený snímek získaný z fotoaparátu nebo mikroskopu. Pro odstranění těchto problémů se používají algoritmy předzpracování snímku.

2.2.1 Konverze do tónů šedi

Nejpoužívanější způsob ukládání barev se nazývá **RGB** podle zkratky barevných kanálů, kterými je barva reprezentována v počítači. Zkratka je složena z počátečních písmen použitých barev. To jsou červená(Red), zelená(Green) a modrá(Blue). Tyto tři barvy jsou základním prvkem všech snímků a ostatní barvy vznikají jejich kombinací.
Barevný snímek uchovává barvy ve více hodnotách, které se nazývají ba-

revné kanály. Na druhou stranu snímek šedotónový potřebuje pouze jednu hodnotu a ta se nazývá jas. Hodnota jasu se dá jednoduše vypočítat z **RGB** složek snímku. K tomuto výpočtu se používá následující vzorec

$$Y = 0,2126 \times R + 0,715 \times G + 0,0722 \times B,$$

kde R, G, B jsou hodnoty jednotlivých barevných kanálů pixelu a Y je jeho vypočtený jas.

2.2.2 Rozostření snímku

Rozostření nebo rozmazání se používá k odstranění šumu ve snímku. Existuje velké množství metod pro rozostřování. Většina jich funguje na bázi konvoluce. Protože pracujeme s diskrétními daty, budeme také používat diskretizovanou verzi konvoluce určenou pro 2D snímky, jejíž vzorec vypadá následovně

$$(M * H)(i,j) = \sum_{i=-k}^{i} \sum_{j=-k}^{j} M(x-i, y-j) \cdot H(i,j),$$

kde M je matice obsahující jasové hodnoty snímku a H je konvoluční matice. K zapsání konvoluce slouží operátor $*$. Pro metody rozostření snímku je velmi důležitá velikost konvoluční matice. Větší matice odstraní více šumu za cenu ztráty informace a náročnějšího výpočtu. Na druhou stranu moc malá konvoluční matice zapříčiní nedostatečné odstranění šumu a může způsobit chyby v další práci se snímkem. Pro konvoluční matice většinou platí, že nabývají velikostí $n \times n$, kde n je liché číslo určené velikostí snímku.

Mezi metody rozmazání snímku patří průměrování, kde konvoluční matice je o velikosti $N \times N$ a její složky nabývají hodnot $\frac{1}{N^2}$. Jedná se o jednu z nejjednodušších metod pro rozmazávání snímku, která se v některých případech chová velice agresivně, a tudíž dochází k přílišné ztrátě informací potřebných k další práci.

Další metodou je **Gaussian blur**, která se používá pro jemnější rozmazávání. Metoda funguje na bázi Gaussovské funkce

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

S využitím této funkce se vygeneruje konvoluční matice. Velmi důležitý je parametr σ, který určuje váhu okolních bodů. Pro větší hodnoty parametr σ bude maximum Gaussovy funkce nabývat menších hodnot a okolní body tak budou mít větší vliv na výslednou hodnotu pixelu.

2.2.3 Binarizace snímku

Jak už název napovídá binarizace snímku je algoritmus, který nastaví všechny jeho jasové hodnoty na bílou, která je reprezentována hodnotou jedna, nebo černou, která se rovná nule. Použitím binarizace se zbavíme veškerého šumu ve snímku a při správně zvoleném prahu t rozlišíme pozadí a popředí snímku. Binarizaci lze popsat následujícím vzorcem

$$H(i,j) = \begin{cases} 0, & \text{pro } H(i,j) < t \\ 1, & \text{pro } H(i,j) \geq t \end{cases}$$

kde t je práh, který lze získat z histogramu snímku a $H(i,j)$ je jasová hodnota pixelu na souřadnicích i a j.

Binarizace je velmi jednoduchý proces, jestliže víme, které hodnoty snímku nastavit na nulu a které na maximální hodnotu. O tom rozhoduje práh, který lze získat za pomoci metod prahování.

Binarizací vznikne černobílý snímek, na který je možné aplikovat některé algoritmy pro detekci hran a také velmi usnadňuje provedení Houghovy transformace.

2.3 Histogram

Histogram zobrazuje informace o jasových hodnotách nebo intenzitách barevných kanálů snímku v jednoduchém a snadno pochopitelném grafu.

Struktura grafu je následující: Osa x znázorňuje počet jasových hodnot snímku, které se většinou nabývají hodnot 0 až 255, a na ose y je zobrazen počet pixelů nabývající daného jasu nebo barevné intenzity.

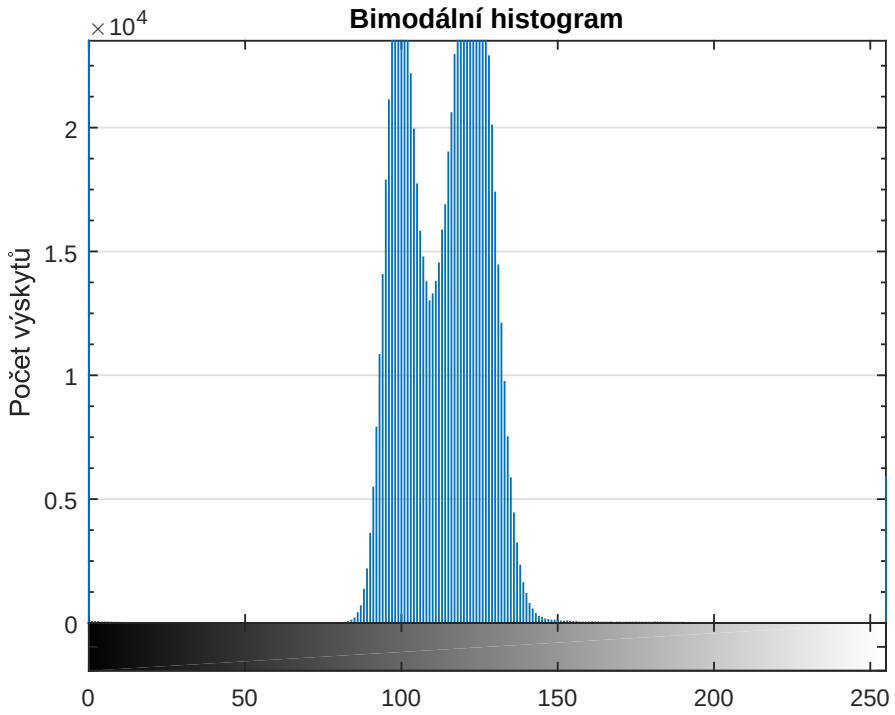
$$n = \sum_{i=1}^{k} m_i.$$

Histogram je funkce, která počítá všechny hodnoty spadající do různých kategorií n, v našem případě jednotlivých jasových hodnot.

2.3.1 Bimodální histogram

Bimodální histogram je speciální typ histogramu. Jelikož většina zpracovávaných snímků má bimodální histogram uvedu zde popis tohoto typu histogramu a jeho výhodu pro binarizaci snímku.

Bimodální histogram se vyznačuje dvěma jasně odlišitelnými maximy a jedním lokálním minimem nacházejícím se mezi nimi. Histogram tohoto typu nám tedy výrazně zjednodušuje automatické určení prahu používaného pro binarizaci snímku.



2.3.2 Úprava histogramu

Před vyžitím histogramu je potřeba ho různě upravovat převážně z důvodu snazšího zobrazení uživateli nebo potřeby eliminace okrajových hodnot.
V neupraveném histogramu se může jedna hodnota jasu pro velké snímky velmi lišit od jiné a tak nastávají obrovské rozdíly mezi jednotlivými jasy. Histogram pak lze jen velmi těžko zobrazit uživateli. K úpravě velkého rozdílu hodnot se používá normalizace. Histogram lze normalizovat podle následujícího vzorce, který aplikujeme pro každou hodnotu jasu,

$$H_n = H \cdot (H_h / H_{max}),$$

v proměnné H jsou předem vypočítané jasové hodnoty, H_h obsahuje požadovanou výšku histogramu a H_{max} je jeho největší jasová hodnota. Pro lepší

čitelnost zobrazovaného histogramu je vhodné do vzorce přidat konstantu $k = 0.9$. Vzorec pak vypadá takto

$$H_n = H \cdot (H_h \cdot k / H_{max}).$$

Další důležitou úpravou je vyhlazení histogramu. Nejlepším způsobem vyhlazení je použití **Gaussova filtru**. Jedná se o konvoluční metodu definovanou takto

$$h_G(i) = (h * G)(i),$$

kde G je definováno jako Gaussova funkce

$$G(x) = \frac{1}{\sqrt{2\pi}} \sigma \cdot e^{-\frac{x^2}{2\sigma^2}}.$$

Protože pracujeme na 1D signálu, budeme používat diskrétní konvoluci pro 1D signál. Její vzorec zní následovně

$$(h * G)(i) = \sum_{j=-k}^{k} (h_j \cdot G_{i-j})$$

Pro optimální vyhlazení histogramu je vhodné zvolit parametry dle článku[3]. Tyto parametry jsou: Velikost konvoluční matice se rovná hornímu zaokrouhlení hodnoty 2.5% počtu **binů** histogramu. Pro histogram nabývající hodnot v rozmezí $0 - 256$ se ke konvoluci použije matice o sedmi prvcích a hodnota $\sigma = 1$.

2.3.3 Hannovo okénko

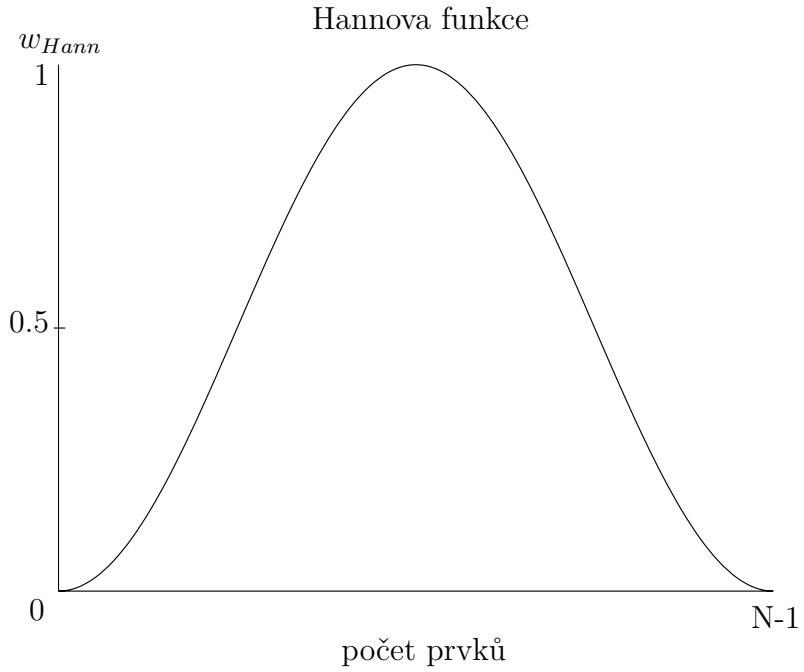
Hannovo okénko je filtr používaný k odstranění okrajových hodnot při měření signálu. Lze také aplikovat na histogramy, u kterých nás více zajímají středové hodnoty než okrajové. Aplikace Hannova okénka je provedena podle následující rovnice

$$h_n(i) = h(i) \cdot w_{Hann}, i \in \langle 0, 255 \rangle,$$

kde Hannova funkce w_{Hann} je definována takto

$$w_{Hann}(n) = 0.5 \left(1 - cos\left(\frac{2\pi n}{N-1}\right)\right), n \in \langle 0, N-1 \rangle.$$

Hannova funkce je znázorněna na obrázku2.2.



Obrázek 2.2: Hannova funkce

2.4 Automatické nalezení prahu

Většina metod pro automatické nalezení prahu používá informace získané z histogramu snímku. Pro plnou automatizaci je potřeba, aby v histogramu byla dvě jasně odlišitelná maxima, viz obrázek 2.3.1. Práh je nejnižší bod mezi těmito maximy. Jedná se tedy o lokální minimum.

2.4.1 Otsuova metoda

Otsuova metoda funguje na principu rozdělení histogramu na dvě křivky reprezentující hustoty pravděpodobnosti popředí a pozadí. Střed křivek je hledaným prahem. Následujíce popisuje histogram jako dvě pravděpodobnostní rozdělení ω_0 a ω_1, kde t je práh a σ_0^2, σ_1^2 je jejich rozptyl,

$$\sigma_w^2(t) = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t).$$

Otsu ve svém článku[4] dokazuje, že minimalizace rozptylu prvků patřících do jedné třídy je stejný proces jako maximalizace rozptylu dvou tříd.

$$\sigma_b^2(t) = \sigma^2 - \sigma_w(t) = \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_T)^2 = \omega_0(t)\omega_1(t)[\mu_0(t) - \mu_1(t)]^2,$$

kde $\omega_{0,1}$ jsou pravděpodobnosti tříd a $\mu_{0,1,T}$ je aritmetický průměr tříd. Pravděpodobnosti lze vypočítat pomocí vzorců

$$\omega_0(t) = \sum_{i=0}^{t-1} p(i),$$

$$\omega_1(t) = \sum_{i=t}^{N} p(i).$$

Aritmetický průměr lze získat vzorci

$$\mu_0(t) = \sum_{i=0}^{t-1} ip(i)/\omega_0,$$

$$\mu_1(t) = \sum_{i=t}^{N} ip(i)/\omega_1,$$

$$\mu_T = \sum_{i=0}^{N} ip(i).$$

Také platí následující rovnost

$$\omega_0\mu_0 + \omega_1\mu_1 = \mu_T,$$

$$\omega_0 + \omega_1 = 1.$$

S využitím následujícího postupu lze implementovat algoritmus pro nalezení optimálního prahu.

1. Výpočet hodnot histogramu.
2. Inicializace $\omega_{0,1}(0)$ a $\mu_{0,1,T}(0)$.
3. Průchod pro všechny t = 1...256.
   (a) Výpočet $\omega_{0,1}$ a $\mu_{0,1,T}$.
   (b) Výpočet $\sigma_b^2(t)$.
4. Hledaný práh t získáme nalezením maximální hodnoty $\sigma_b^2(t)$.

2.5 Detekce geometrických obrazců

Detekcí geometrických obrazců se rozumí nalezení a označení všech objektů rozlišitelných od pozadí snímku lidským okem. Ať už se jedná o body, hrany, přímky nebo jiné, složitější objekty.

2.5.1 Houghova transformace

Houghova transformace se primárně používá pro detekci přímek ve snímku, ale lze ji aplikovat i na detekci kružnic. Nejprve se budeme věnovat základní verzi algoritmu pro detekci přímek.

Každá přímka lze popsat pomocí rovnic

$$\theta = \tan^{-1} n_y,$$

$$\theta = \tan^{-1} n_x,$$

$$d = xn_x + yn_y,$$

kde θ je směrnice přímky a x, y jsou souřadnice bodu, kterým přímka prochází. Algoritmus detekce čar funguje následovně[5].

1. Vytvoříme a vynulujeme si 2D matici, která bude sloužit jako akumulátor pro ukládání hodnot definující potenciální přímky.
2. Procházíme binarizovaný snímek a pro každý pixel s hodnotou 1 vypočteme proměnné θ a d.
3. Inkrementujeme hodnotu v akumulátoru na pozici (θ, d).
4. Lokální maxima v akumulátoru jsou na pozicích, které definují nalezené přímky ve snímku.

2.5.2 Houghova transformace kružnic

Algoritmus pro detekci kružnic funguje obdobně. Bohužel kružnice nelze popsat pouze dvěma čísly. Proto už se nejedná o triviální prohledávání 2D matice. Kružnice je definována svým středem x_s, y_s a poloměrem R,

$$x = x_s + R\cos(\theta),$$

$$y = y_s + R\sin(\theta),$$

pro $\theta \in \langle 0, 2\pi \rangle$. Kružnice je tedy definována třemi proměnnými, proto také potřebujeme 3D akumulátor. Je sice možné provést algoritmus pouze s 2D akumulátorem, jestliže známe poloměr kružnice, ale takto specializovaný algoritmus ztrácí smysl. Na druhou stranu provedení algoritmu pro všechny možné poloměry kružnic, počínajíce $R = 1$ konče $R = im_{width}$, by bylo velmi náročné na zdroje. Proto se velikost kružnic omezuje.

1. Vytvoříme a vynulujeme si 3D akumulátor.

2. Procházíme binarizovaný snímek a každý pixel rovný 1 použijeme jako střed kružnice, na jejímž obvodu inkrementujeme hodnoty v akumulátoru pro všechna zvolené poloměru velikosti R.

3. Vyhledáme lokální maxima v akumulátoru, která definují nalezené kružnice.

4. Nalezneme a eliminujeme kružnice, jejichž vzdálenost od středu lépe ohodnocené kružnice je menší než parametr zadaný uživatelem.

3 Implementace

3.1 Využité technologie

Program byl vyvíjen v jazyce `C++` v prostředí **QT Creator** a na operačním systému **Linux**. Kompilace a sestavení programu bylo otestováno také na operačním systému **Windows 10**. Pro usnadnění práce s obrázky byla využita knihovna **OpenCV** a pro uživatelské rozhraní knihovna **QT**. Obě využité knihovny jsou legálně volně dostupné na Internetu ke stažení včetně zdrojových kódů.

3.1.1 Knihovna OpenCV

OpenCV se specializuje na problémy počítačové vidění včetně předzpracování a zpracování snímku. Knihovnu lze využít v programech psaných jazyky `C`, `C++`, `Python`, `Java`, `MATLAB` na operačních systémech **Windows**, **Linux**, **Android**, **Mac OS**. Knihovna je napsaná v jazyce `C++`, proto je její integrace do tohoto jazyka jednoduší.

Program využívá knihovnu pro algoritmy zpracování snímku a struktury potřebné pro jejich funkčnost. Dále jsou popsány využité algoritmy a struktury.

`Mat`, struktura lze využít jako matice nebo pro uchovávání dat snímku. Většina algoritmů pro zpracování snímku v knihovně pracuje s touto strukturou jako hlavním zdrojem dat.

`HoughCircles`, funkce najde kružnice v zaslaném snímku. Snímek musí být předzpracován do binární podoby. Algoritmus je velmi náchylný k šumu objevujícího se ve snímku. Z optimalizačních důvodů je ve funkci implementována speciální verze algoritmu Hougovy transformace kružnic, která je popsána níže v sekci **Houghova transformace**.

`cvtColor`, algoritmus k převodu barevného snímku do tónů šedi. Tato funkce je velmi důležitá pro správnou funkčnost ostatních algoritmů knihovny.

`line`, kreslení úseček do snímku, který je, například, uchováván v objektu `Mat`. Funkce byla využita pro vykreslení histogramu.

`circle`, kreslení kružnic do objektu `Mat`. Funkce byla využita pro kontrolu výsledků nalezených kružnic jejich vizualizací.

`morphologyEx`, funkce pro morfologické operace. V programu jsou využity dvě morfologické operace. První je operace otevření, která je složením dvou operací. To je eroze následována dilatací. A druhou operací je nalezení gradientu snímku. Tyto operace jsou použity pro dokončení příprav snímku před spuštěním algoritmu `HoughCircles`.

3.1.2 Knihovna Qt

Knihovna **Qt** se specializuje na tvorbu grafického prostředí. Je dostupná pro operační systémy **Linux**, **Windows** a mnoho dalších systémů včetně mobilních platforem a různých vestavěných systémů. Ke knihovně **Qt** je možné získat mnoho doplňkových nástrojů zdarma, mezi kterými je i vývojářské prostředí **Qt Creator**, jež bylo použito při vývoji aplikace.

Každé okno **Qt** aplikace má vlastní třídu a vlastní **xml** soubor, který definuje vzhled okna. Uživatelské rozhraní obsahuje minimálně jedno okno, které je jejím hlavním oknem a při jeho zavření se ukončí celá aplikace včetně ostatních oken potomků. Každé další okno je potomkem hlavního okna nebo jiného okna, které je potomkem hlavního okna. Kromě vazby potomek, rodič jsou třídy zapouzdřeny.

Qt aplikace mají sofistikovaný způsob předávání informací mezi jednotlivými okny. Ke transferu informací používají signály a sloty pracující na podobném principu jako architektonický vzor klient-server. Signál i slot musí být definovány v hlavičkovém souboru. Signál je definován ve třídě, která posílá informaci nebo volá funkčnost definovanou v jiné třídě. Slot je speciální funkce implementující volanou funkčnost. Slot je definován a implementován ve volané třídě, které reaguje na odeslaný signál.

V aplikaci je využito velké množství objektů knihovny **Qt**, proto zde nebudou popsány všechny, ale jen ty úzce spjaté s vlastní implementací.

`QGraphicsView`, komponenta používaná k zobrazení scény. Z důvodu potřeby přetížení některých událostí a metod této třídy byla založena nová třída s názvem `CustomView`, která dědí od `QGraphicsView`.

`QGraphicsScene`, scéna je objekt podřízený `QGraphicsView`. Do scény se

vkládají objekty, například snímky, které má `QGraphicsView` vykreslit.

`QRubberBand`, komponenta použitá pro selekci části snímku, se kterou bude aplikace dále pracovat. Komponenta je také součástí výše uvedené třídy `CustomView`.

`QImage`, `QPixmap`, `QGraphicsPixmapItem` třídy pracující se snímkem. `QImage` obsahuje `QPixmap`, která je přidávána do scény. Funkce, která přidá `QPixmap` do scény vrací ukazatel na objekt ve scéně v podobě `QGraphicsPixmapItem`.

3.2 Struktura aplikace

Aplikace se skládá ze tří oken. Hlavní okno `MainWindow` zobrazuje aktuálně načtený snímek a jeho aktuální stav po provedených úpravách. Okno **Tools** obsahuje tlačítka pro práci s vykresleným snímkem. Okno **Tools** také zobrazuje výsledek procentuálního zastoupení vláken ve vybrané části snímku. V případě zavření okna, ho lze znovu otevřít z hlavního okna. Posledním je `HistWindow`, které se používá k zobrazení histogramu a umožňuje změnu vypočteného prahu. Okno se objeví po výpočtu histogramu a mizí po provedení operace binarizace.

Kromě tří oken má projekt jednu třídu `CustomView`, která se stará o vykreslení načteného snímku. Třída dědí od `QGraphicsView`. Implementace `CustomView` byla nutná z důvodu potřeby přetížení některých funkcí a událostí třídy
`QGraphicsView`. V třídě jsou implementovány základní operace se snímkem jako je zvětšení, zmenšení snímku nebo výběr jeho části.

Ostatní soubory slouží pouze k oddělení kódu pro lepší čitelnost. Jedná se o následující soubory.

1. `Convert.cpp` - obsahuje kód pro konverzi `QImage` na strukturu `Mat` a zpět,

2. `histogram.cpp` - obsahuje algoritmy pro výpočet a úpravu histogramu. Funkce jsou volány z okna `MainWindow`,

3. `threshold.cpp` - soubor zahrnuje implementované metody pro automatické nalezení prahu a také metodu binarizace snímku.

Ke každému `.cpp` - souboru existuje přidružený `.h` soubor, kde jsou definované hlavičky funkcí a globální proměnné. Třídy mají v hlavičkových souborech definovány signály a sloty.

3.3 Konverze QImage na Mat

Qt dokáže v okně zobrazovat pouze objekty tříd `QImage` a `QPixmap`. Zde nastává konflikt s funkcemi knihovny `OpenCV`, která potřebuje k většině operací data uložená v objektu `Mat`. Proto bylo nutné vytvořit konvertor mezi typy `QImage` a `Mat`. Ukázka zdrojového kódu převodu lze vidět v 3.1. Hlavním problémem při vytváření metod pro konverzi mezi `QImage` a `Mat` bylo určit způsob uložení dat v `QImage`. Protože `QImage` může obsahovat data v několika formátech, musel jsem otestovat, který formát bude použit před převodem. V případě jiného formátu funkce vrací prázdný `QImage` nebo `Mat`.

```
1  Mat qImageToMat(const QImage src)
2  {
3    if(src.format() == QImage::Format_RGB32)
4      {
5          Mat mat(src.height(), src.width(), CV_8UC4,
6          const_cast<uchar*>(src.bits()), src.bytesPerLine());
7          return mat.clone();
8      }
9      else
10         return Mat();
11 }
```

Kód 3.1: Ukázka převodu `QImage` na `Mat`.

Inverzní převod z `Mat` na `QImage` je velmi podobný.

3.4 Předzpracování snímku

Po načtení snímku a jeho zobrazení v hlavním okně je třeba ho upravit. V programu je umožněno snímek oříznout a zbavit se tak nežádoucích nebo rušivých okrajových částí. K implementaci jsem využil z velké části knihovnu `Qt`, především objekt `QRubberBand` a události vyvolané myší, `mouseMoveEvent`, `mouseReleaseEvent` a `mousePressEvent`.

Po oříznutí snímku je třeba co nejlépe odstranit šum ze snímku, k tomu je použit algoritmus Gaussova rozmazání pro vyhlazení snímku, který je implementován v knihovně `OpenCV`. Kód můžete nalézt v 3.2

```
// Ziskani snímku
Mat image(qImageToMat(pixIt->pixmap().toImage()));
// Prevod snímku do greyscale formatu
cvtColor( image, image, CV_BGR2GRAY );
// Aplikace Gaussova filtru s konvolucni matici o velikost 9x9
GaussianBlur(image, image, Size(9,9), 0, 0);
```

Kód 3.2: Aplikace Gaussova rozmazání na snímek.

3.5 Výpočet a úprava histogramu

Jak již bylo uvedeno výše, výpočet histogramu je relativně jednoduchý proces. Snímek je funkci předán v podobě matice
Funkce 3.3 spočítá hodnoty histogramu do výstupního pole `hist[]` a vrací maximální hodnotu. Ta je využita k následné úpravě histogramu, díky které je zobrazený histogram čitelnější.

```
int histCompute(Mat image, int hist[])
  {
    int i,j;
    int pos;
    int max = 0;
    for(i = 0; i <  image.size().width; i++)
    {
        for(j = 0; j <  image.size().height; j++)
        {
            pos = (int)image.at<uchar>(j,i);
            (*hist)[pos]++;
            if(max < (*hist)[pos])
            {
                max = (*hist)[pos];
            }
        }
    }
    return max;
}
```

Kód 3.3: Výpočet histogramu.

Výsledné hodnoty histogramu můžou mít mezi sebou velké rozdíly. Tento problém lze vyřešit s použitím výše zmíněného maxima získaného při výpočtu histogramu a to normalizací hodnot do zobrazitelného rozpětí. Kód normalizace naleznete v 3.4.

```
void histNormalize(int max, int histSize, int hist_h, int* hist[])
{
   int i;
   for(i = 0; i < histSize; i++)
       (*hist)[i] = (int)((double)(*hist)[i]*((double)(hist_h*0.9)/max));
}
```

Kód 3.4: Normalizace histogramu.

Dokonce i takto upravený histogram pořád obsahuje šum, které může zabránit úspěšnému nalezení prahu. Z testovacích snímků, které mám k dispozici, znám přibližný vzhled histogramu, a tedy i pozici prahu, který nebude nabývat okrajových hodnot jasu. Proto můžu vyfiltrovat okrajové hodnoty použitím Hannova okénka. Aplikace Hannova okénka v kódu 3.5

```
void hannWindow(int* hist[], int histSize)
{
    double hann;
    int i;
    for(i = 0; i < histSize; i++)
    {
        hann = 0.5*(1 - cos((2*i*M_PI)/(histSize-1)));
        (*hist)[i] = (int)(hann * (double)(*hist)[i]);
    }
}
```

Kód 3.5: Hannovo okénko.

Po aplikaci `Hannova okénka` už jen zbývá vyhladit případné nerovnosti v histogramu. K tomu lze použít `Gaussovu` funkci.

Parametry ω a velikost konvolučního jádra jsou převzaté z článku[3], jenž deklaruje parametr $\omega = 1$ a velikost konvolučního jádra je rovno $2,5\%$ z celkového rozsahu histogramu.

Funkce 3.6 využívá matematických konstant `M_PI` a `M_E`, které jsou obsaženy ve standardní `C++` knihovně `math.h`. Aby byly konstanty použitelné, musí být definováno makro `_USE_MATH_DEFINES`.

```
void histSmoothing(int in[], int* out[], int histSize)
{
    double omega = 1;
    int kernSize = (int)ceil(histSize*0.025);
    int kernHalf = (int)floor(kernSize/2);
    double * kern;
    int i,j, x;
    kern = (double*)malloc(kernSize*sizeof(double));
    for(i = 0; i < kernSize; i++)
    {
       x = i - kernHalf;
       kern[i] = 1.0/(sqrt(2*M_PI)*omega)*pow(M_E,-(x*x)/(2*pow(omega,2)));
    }

    for(i =  kernHalf; i < histSize-(int)floor(kernSize/2); i++)
    {
       for(j = 0; j < kernSize; j++)
           (*out)[i] += (int)(kern[j]*in[i+j-kernHalf]);
    }
    free(kern);
}
```

Kód 3.6: Vyhlazení histogramu.

Upravený histogram bude zobrazen ve zvláštním okně, které také znázorňuje hodnotu aktuálně spočteného prahu.

3.6 Automatické prahování

Po provedení všech uvedených úprav je možné provést automatické prahování za pomoci získaného histogramu. Práh se vyskytuje mezi dvěma maximy, které vzniknou z pozadí a z vláken, která máme za úkol detekovat. Proto můžu použít následující algoritmus. Pro větší přesnost hledám práh dvěma způsoby a pro binarizaci snímku použiji lépe hodnocený práh. Oba způsoby využívají zderivovaných hodnot histogramu.

První způsob vyhledání prahu, jehož kód je uvedený v 3.7, je hledání samotného minima ve zderivovaných hodnotách histogramu. Hledáme tedy průchod nulou. Bohužel to nám nezajistí, že se jedná o minimum. Proto musíme kontrolovat nejen hodnoty s indexem `i` ale i hodnoty s indexem `i-1`.

```
for(i = 1; i < 256; i++)
    {
        if(gradient[i-1] < 0 && gradient[i] > 0 ||
                gradient[i-1] < 0 && gradient[i] == 0 && gradient[i+1] > 0)
        {
            t = i;
            break;
        }
    }
```

Kód 3.7: Hledání lokálního minima.

Takto nalezené minimum může být přese všechny úpravy histogramu ovlivněné případným výskytem šumu obsaženým v histogramu. Druhý způsob, který je uveden v kódu 3.8, taktéž prohledává zderivované hodnoty, ale nehledá minimum. Hledá dvě maxima, která znázorňují jas popředí a pozadí snímku. Podobně jako u předchozího způsobu musíme kontrolovat více hodnot při průchodu nulou k identifikaci maxima.

```
for(i = 1; i < 255; i++){
  if(gradient[i-1] > 0 && gradient[i] < 0 ||
        gradient[i-1] > 0 && gradient[i] == 0 && gradient[i+1] < 0)
    {
      if(max1 == -1)
          max1 = i;
        else{
          max2 = i;
            break;
        }
     }
}
```

Kód 3.8: Hledání lokálních maxim.

Po nalezení dvou maxim už pouze stačí najít minimum, které se nachází mezi nimi. K tomu jsou použity nezderivované hodnoty histogramu.

Výběr lepšího z nalezených prahů uvedený v 3.9, je jednoduchý. Lepší práh je vždy ten menší, protože se jedná o lokální minimum. Nicméně toto minimum nebude nikdy rovno nule, proto vezmeme menší práh, který se nerovná nule.

```
1 //Vezme menší threshold, pokud je větší než 0
2 tRet = hist[t1] > hist[t2] ? t1 > 0 ? t1 : t2 : t2 > 0 ? t2 : t1;
3 return tRet;
```

Kód 3.9: Klasifikace lepšího prahu.

Nalezený práh bude zobrazen v okně, které vykresluje histogram. Zde také může být upraven v případě jeho nepřesného určení.

3.7 Binarizace snímku a další úpravy

Za pomoci získaného prahu může aplikace provést binarizaci snímku, jejíž kód je zobrazen v 3.10.

```
1  void threshold_img(int threshold, Mat img)
2  {
3      for (int i = 0; i < img.size().width; i++)
4      {
5          for (int j = 0; j < img.size().height; j++)
6          {
7              if (img.at<uchar>(j, i) < threshold)
8                  img.at<uchar>(j, i) = (uchar) 0;
9              else
10                 img.at<uchar>(j, i) = (uchar) 255;
11         }
12     }
13 }
```

Kód 3.10: Binarizace snímku.

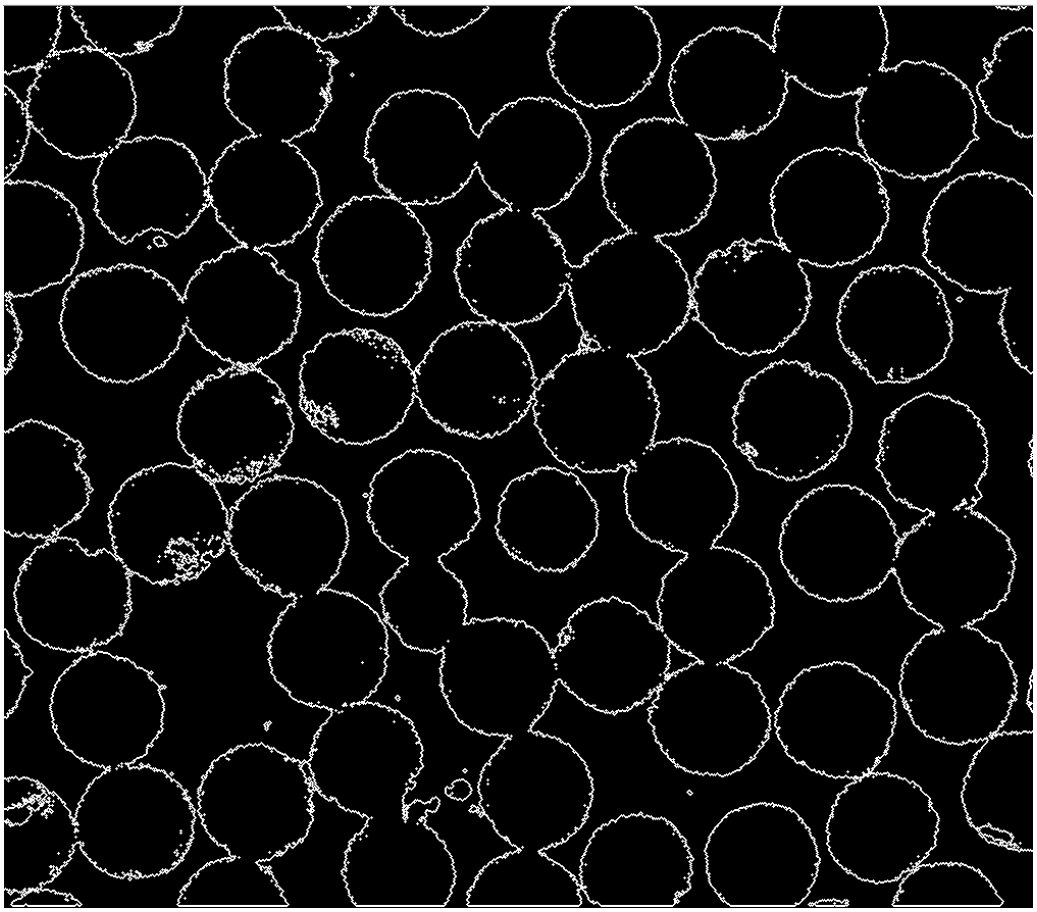
Protože binarizovaný snímek bude obsahovat celé kruhy a algoritmus pro detekci kružnic funguje lépe na snímku obsahující pouze hrany, bude po binarizaci provedeno několik dalších úprav, které lze nalézt v kódu 3.11.

Pro nalezení hran kruhů v binarizovaném snímku, používá aplikace funkci knihovny `OpenCV`. Jedná se o morfologickou operaci gradient. Před použitím této operace je provedena ještě jiná operace, která se nazývá otevřen. Tato operace vyplní a spojí některé mezery v kruzích a usnadní tak následující operaci gradientu. Obě operace používají kruhovou konvoluční matici o velikosti 3×3.

```
1 Mat kernel = getStructuringElement(MORPH_ELLIPSE, Size(3, 3), Point(1, 1));
2 morphologyEx(in, in, MORPH_OPEN, kernel);
3 morphologyEx(in, out, MORPH_GRADIENT, kernel);
```

Kód 3.11: Úprava binarizovaného snímku.

Výsledný snímek zobrazuje obrázek 3.1.



Obrázek 3.1: Předzpracovaný snímek

3.8 Houghova transformace

K vyhledání a označení kružnic ve snímku je používán algoritmus Hougovy transformace kružnic. Funkce, která je implementována v knihovně **OpenCV**, vrací seznam středů a poloměrů nalezených kružnic. Nalezené středy a poloměry se použijí nejen k výpočtu plochy, kterou vlákna vyplňují ve snímku, ale i k jejich označení.

Z důvodu časové náročnosti algoritmu je v **OpenCV** vyžita tzv. Houghova gradientní metoda, jež je popsána níže.

1. Detekce hran ve snímku pomocí `Cannyho` algoritmu.
2. Pro každý nenulový bod ve snímku vzniklém po aplikaci detekce hran je pomocí `Sobelova` operátoru spočítán lokální gradient.
3. Každý nenulový bod je uložen v paměti pro pozdější použití v algoritmu.
4. Podle gradientu jsou inkrementovány hodnoty v matici akumulátoru, který je omezený zadanými hodnotami minimální a maximální velikostí kružnic.
5. Středy kružnic jsou identifikovány jako lokální maxima matice akumulátoru přesahující zadaný práh.
6. Takto nalezené středy jsou seřazeny podle hodnocení v matici akumulátoru v sestupném pořadí.
7. Pro každý střed jsou seřazeny uložené nenulové pixely podle vzdálenosti od středu, kdy první pixel je ten nejbližší.
8. Je vybrán nejlepší poloměr podle vzdálenosti nenulových pixelů od středu.

Tento algoritmus nabízí možnost využít `2D` akumulátor místo obvykle vyžadovaného `3D` akumulátoru a snižuje tím využité zdroje[1].

Aplikace byla testována na dvou skupinách snímků. První skupina byly snímky focené s přiblížením $\times 4000$ a druhá $\times 1000$. Protože algoritmus **Houghovy** transformace je upřesněn předpokládanou velikostí poloměrů, je logické, že aplikace volá funkci algoritmu s jinými parametry pro každou skupinu. Tyto parametry byly určeny experimentálně.

3.9 Problémy při implementaci

Největším problémem bylo samotné zprovoznění prostředí pro implementaci. První verze projektu byla bez uživatelského rozhraní a tudíž bez knihovny **Qt**, proto mohla být vyvíjena v operačním systému **Windows** v prostředí **Microsoft Visual Studio**. Po dokončení funkčního prototypu bylo potřeba integrovat do prostředí knihovnu **Qt**, což se projevilo jako časově náročný

problém. Řešením bylo využití jiného vývojového prostředí **Qt Creator**. Zde bohužel nastal problém integrace knihovny **OpenCV**. Vše bylo vyřešeno kompilací vlastní knihovny **OpenCV** a dočasným přenesením vývoje aplikace na systém **Linux**. Ve finální fázi vývoje byla provedena integrace knihovny **OpenCV** do prostředí **Qt Creator** na systému **Windows 10**. Využité části knihovny **OpenCV** jsou dodané společně s aplikací.

Dalším problémem byl nevhodný výběr algoritmu pro automatické prahování snímku. Prvotně byl zvolen Otsuovo algoritmus pro nalezení prahu. Tento algoritmus bohužel pro většinu dat nefungoval a nacházel špatné prahy.
Špatně zvolený algoritmus byl také algoritmus pro hledání hran ve snímku, čímž by se výrazně usnadnila detekce kružnic. K nalezení hran měl být použit algoritmus **SUSAN - Smallest Univalue Segment Assimilating Nucleus**, který používá kruhovou masku ke konvoluci. Bohužel z důvodu nedostatečných vědomostí k implementaci a z nedostatku času byl algoritmus vyřazen z projektu.

4 Výsledky

4.1 Testovací metody

Způsob testování výsledků získaných pomocí aplikace by byl velmi jednoduchý, kdyby bylo třeba zaznamenávat pouze počet nalezených vláken. Cílem aplikace je vypočítat procentuální obsah vláken. Z důvodu rozdílné velikosti vláken, nebylo možné určit jejich přesný obsah ve snímku pouhým okem. To bylo vyřešeno následujícím způsobem.

1. Získání počtu nalezených vláken a jejich obsah ve snímku aplikací.
2. Výpočet průměrného procentuálního obsahu jednoho vlákna.
3. Identifikace špatně určených a neurčených vláken.
4. Výpočet obsahu vláken ve snímku pomocí získaných informací a vzorce

$$S_{akt} = (S_{app} \cdot n_{akt})/n_{app},$$

   kde S je obsah a n je počet vláken. Dále dolní index znázorňuje zda se jedná o data získaná aplikací (app) nebo lidským okem (akt).

Další problém, který bylo třeba vyřešit při ověřování výsledků aplikace, byla nutnost otestovat i snímky s přiblížením $\times 1000$. Tyto snímky obsahují přibližně 500 až 800 vláken, proto bylo u nich velice obtížné spočítat skutečný obsah vláken pouhým okem. Z tohoto důvodu bylo využití vzorků s malým přiblížením omezeno.

4.2 Testování výsledků

Snímků použitých pro testovací účely bylo osmnáct. Každý snímek byl focen třikrát, pokaždé s jiným filtrem. Z celkové množiny čítající 54 snímků jsem vybral 9 snímků s přiblížením 1000 a 21 snímků s přiblížením 4000.

K otestování správnosti výsledků aplikace jsem se rozhodl použít test významnosti rozdílu dvou výběrových průměrů, známý jako `t-test`. Mějme hypotézu H_0, která praví $m_1 = m_2$, kde m jsou střední hodnoty množin získaných výsledků. Tuto hypotézu přijmeme, pakliže bude platit

$$T < t_{0.05},$$

$t_{0.05}$ je kritická hodnota hladiny významnosti 5% a T je testovací kritérium definované jedním z následujících vzorců

$$T = \frac{m_1 - m_2}{\sqrt{n_1 \cdot s_1^2 + n_2 \cdot s_2^2}} \cdot \sqrt{\frac{n_1 \cdot n_2 \cdot (n_1 + n_2 - 2)}{n_1 + n_2}},$$

nebo

$$T = \frac{m_1 - m_2}{(n_2 - 1) \cdot s_1^2 + (n_1 - 1) \cdot s_2^2} \cdot \sqrt{(n_1 - 1) \cdot (n_2 - 1)},$$

kde m jsou střední hodnoty množin, n jsou velikosti množin a s^2 jsou jejich rozptyly (disperze). Tyto dva vzorce potřebujeme, protože v různých případech se kritérium T počítá jiným způsobem.

První vzorec použijeme v případě, že platí následující rovnost

$$s_1^2 = s_2^2,$$

kterou určíme pomocí kritéria F-testu definovaného

$$F = \frac{n_1(n_2 - 1) \cdot s_1^2}{n_2(n_1 - 1) \cdot s_2^2}.$$

Získanou hodnotu F porovnáme s kritickou hodnotou $f_{0,025}$, jestliže platí

$$F < f_{0,025}$$

přijímáme hypotézu $s_1^2 = s_2^2$ a použijeme první vzorec pro výpočet testovacího kritéria T. V opačném případě použijeme druhý vzorec.

Naměřené hodnoty lze nalézt v přiloženém souboru `Vysledky.ods`, z důvodu velkého obsahu zde uvedu data potřebná pouze k samotným výpočtům. Nejprve je třeba získat střední hodnoty měření.

$$m = \frac{1}{N} \sum_{i=1}^{N} x_i,$$

kde x jsou jednotlivé hodnoty a N je jejich celkový počet.
Nejprve dosadíme hodnoty určené lidským okem a získáme hodnotu

$$m_1 = 0.49890.$$

A dosazením aplikací spočítaných obsahů vláken získáme hodnotu

$$m_2 = 0.46800.$$

Pro výpočet rozptylů s_1^2 a s_2^2 jsem použil vzorec

$$s^2 = \frac{1}{N-1}\left(\sum_{i=1}^{N} x_i^2 - N \cdot m\right).$$

Tento vzorec nám poskytne následující hodnoty

$$s_1^2 = 0.00711,$$

$$s_2^2 = 0.00714.$$

Teď už jen zbývá, s užitím získaných hodnot, vypočítat testovací kritérium T. Nejdříve ale potřebujeme zjistit jaký vzorec k výpočtu použijeme.

To zjistíme provedením jednoduchého F-testu popsaného výše.

$$F = \frac{30 \cdot 29 \cdot 0.00711}{30 \cdot 29 \cdot 0.00714} = 0.995165.$$

Získaný výsledek porovnáme s hodnotou významnosti 0.025, kterou získáme funkcí `FINV`, která je součástí `MS excelu` nebo `LibreOffice Calc`. Kritická hodnota se rovná

$$F_{0.025} = 2.10100.$$

Můžeme tedy říct, že platí

$$F < F_{0.025},$$

a použít vzorec

$$T = \frac{m_1 - m_2}{\sqrt{n_1 \cdot s_1^2 + n_2 \cdot s_2^2}} \cdot \sqrt{\frac{n_1 \cdot n_2 \cdot (n_1 + n_2 - 2)}{n_1 + n_2}},$$

pro výpočet testovacího kritéria T. Po dosazení získáme rovnici

$$T = \frac{0.4989 - 0.4680}{\sqrt{30 \cdot 0.00711 + 30 \cdot 0.00714}} \cdot \sqrt{\frac{30 \cdot 30 \cdot 58}{60}} = 1.394110.$$

Získáme kritickou hodnotu $t_{0.05}$ pomocí funkce `TINV`, která je také součástí `MS excelu` nebo `LibreOffice Calc`.

$$t_{0.05} = 2.001717.$$

Můžeme tedy říct, že platí

$$T < t_{0.05}.$$

Z toho vyplývá, že přijímáme hypotézu $s_1 = s_2$, a tudíž obě množiny nabývají stejného pravděpodobnostního rozdělení na hladině významnosti 5%.

5 Závěr

Cílem této práce bylo vytvořit software využívaný zaměstnanci katedry mechaniky při zkoumání kvality kompozitních materiálů, která se, mimo jiné, určuje také podle hustoty výztužných vláken obsažených v materiálu. Zejména kvůli potřebě takto zpracovat velké množství dat vznikl požadavek vytvořit aplikaci pro automatickou detekci.

Práce se skládá ze tří významových kapitol. V první kapitole jsou detailně vysvětleny teoretické znalosti, které byly potřeba k implementaci aplikace. Patří mezi ně například informace o kompozitních materiálech, ale také metody předzpracování snímku a jiných úprav používaných v oboru počítačového vidění. Druhá kapitola se věnuje samotné implementaci a nabývá tak více praktického rázu. Zde je detailně popsána struktura aplikace, využité technologie a také uvedeny programové struktury použité k řešení některých problémů. Kapitola je obohacena o ukázky zdrojového kódu. Třetí kapitola shrnuje dosažené výsledky a aplikuje statistické metody pro jejich interpretaci.

Statistické hodnocení výsledků ukázalo jejich správnost, a proto aplikaci považuji za odpovídající zadání. Pro usnadnění ovládání je aplikace obohacena o intuitivní uživatelské rozhraní. Dále byla vytvořena uživatelská příručka, která je součástí této práce.

Přestože aplikace splnila zadání a dosahuje dobrých výsledků, existuje prostor pro její zlepšení, zejména se jedná o rozšíření uživatelského prostředí o možnost zpět nebo také o doplnění více možností detekce vláken. Aplikace by mohla také zahrnovat možnost hromadného zpracování snímků a výstupu výsledků do souboru.

Literatura

[1] Bradski, G. – Kaehler, A. *Learning OpenCV*. O'Reilly Media, 2008. ISBN 978-0-596-51613-0.

[2] Daďourek, D. I. K. *Kompozitní materiály - definice a rozdělení* [online]. Technická univerzita v Liberci, 2008. [cit. 2016/04/16]. Kompozit - definice. Dostupné z: `http://www.kmt.tul.cz/edu/podklady_kmt_magistri/KM/Kompozity%20Dad/02defrozd.pdf`.

[3] Ekštein, K. Simple and Efficient Method of Low-Contrast Grayscale Image Binarization. 2015.

[4] Otsu, N. A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics*. Jan 1979, 9, 1, s. 62–66. ISSN 0018-9472.

[5] Szeliski, R. *Computer Vision: Algorithms and Applications*. Springer, 2010. ISBN 978-1-84882-935-0.

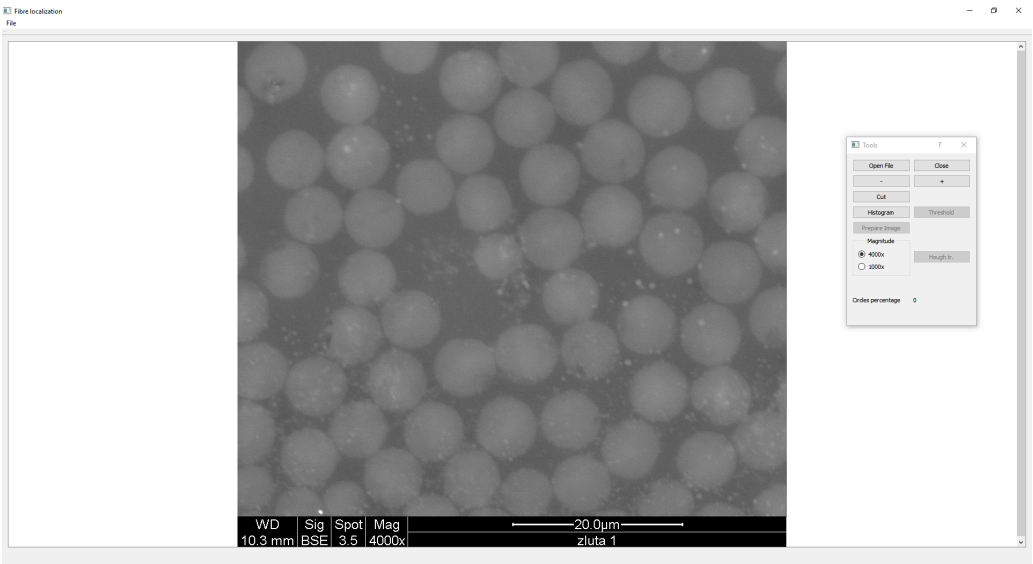
A Uživatelská příručka k programu Fibres

A.1 Sestavení programu

Aplikace je napsaná v jazyce **C++**, a protože využívá různých nestandardních knihoven, je její kompilace trochu odlišná. Pro kompilaci aplikace je potřeba mít nainstalovanou knihovnu **OpenCV 3.1.0 32bit** nebo vyšší a knihovnu **Qt 5.6.0 MSVC2015 32bit** nebo vyšší. Pro operační systém **Windows** musejí být cesty knihovnám v proměnném prostředí systému. Pro sestavení aplikace je doporučeno použít programovací prostředí **Qt Creator**.

A.2 Ovládání programu

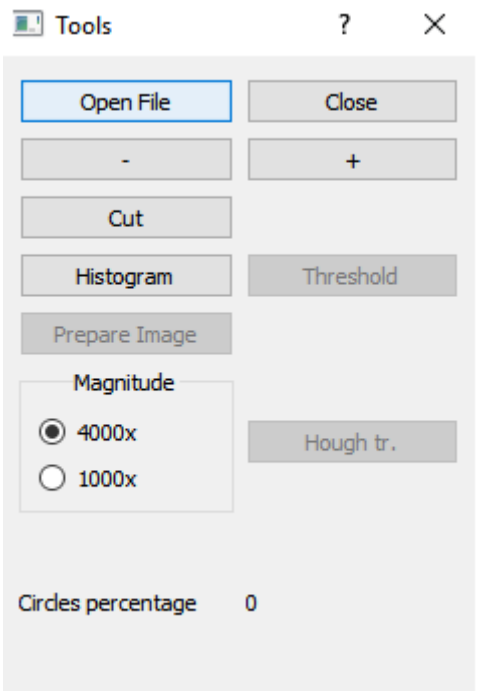
Aplikace je vybavená jednoduchým uživatelským rozhraním, které se skládá ze dvou oken. Hlavní okno, **Fibres localization**, vykresluje aktuální stav načteného snímku. Druhé okno, **Tools**, poskytuje přístup k nástrojům pro práci se snímkem. Nástrojové okno obsahuje následující možnosti práce se



Obrázek A.1: Aplikace s otevřeným snímkem

snímkem.

1. 'Open File' - Otevře souborový průzkumník pro načtení snímku. Aplikace pracuje se snímky typu **.tif** nebo **.jpg**.
2. 'Close' - Zavře otevřený snímek.
3. '-' a '+' - Zmenší/Zvětší načtený snímek.
4. 'Cut' - Vyřízne označenou část snímku. Část snímku lze označit držením levého tlačítka a tahem požadovaným směrem.
5. 'Histogram' - Vypočte histogram a zobrazí ho v novém okně.
6. 'Threshold' - Vypočte práh a jeho hodnotu zobrazí v okně histogramu, kde je možné práh ručně upravovat.
7. 'Magnitude' - Zde je potřeba určit přiblížení zpracovávaného snímku.
8. 'Hough tr.' - Provede Houghovu transformaci kružnic.
9. 'Circles percentage' - Zobrazuje vypočtený procentuální výskyt vláken ve snímku.



Obrázek A.2: Detail okna **Tools**

Aktivita jednotlivých tlačítek se mění v závislosti na aktuální fázi procesu přípravy snímku a umožňují tak intuitivní práci s programem.