ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta strojní – Ústav mechaniky, biomechaniky a mechatroniky

DIPLOMOVÁ PRÁCE

Využití obrazové analýzy a jejích algoritmů při defektoskopii nanovlákenných vrstev

Bc. Vojtěch Šípek **2019**

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Šípek** Jméno: **Vojtěch** Osobní číslo: **424925**

Fakulta/ústav: **Fakulta strojní**

Zadávající katedra/ústav: **Ústav mechaniky, biomechaniky a mechatroniky**

Studijní program: **Strojní inženýrství**

Studijní obor: **Mechatronika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Využití obrazové analýzy a jejích algoritmů při defektoskopii nanovlákenných vrstev

Název diplomové práce anglicky:

Use of image analysis and its algorithms for defectoscopy of nanofibrous layers

Pokyny pro vypracování:

1) proveďte rešerši vhodných metod a algoritmů pro defektoskopii z 2-D obrazů podobného typu úloh (např. : Eukl. vzd., Histgr., FFT, PCA, Multi-fraktální míry, Neuronové sítě (SVM, CNN, DNN)
2) Navrhněte a otestujte jednotlivé vhodné metody a vyhodnoťte jejich spolehlivost (např. ROC)
3) Navrhněte co nejspolehlivější řešení detekce (zřejmě jako vhodnou kombinaci více metod) a otestujte, zdokumentujte

Seznam doporučené literatury:

[1] Image Manipulation. The Hitchhiker's Guide to Python [online]. [vid. 2018-04-25]. Dostupné z: http://docs.python-guide.org/en/latest/scenarios/imaging/
[2] VAN DER WALT, Stéfan, Johannes L. et al. scikit-image: image processing in Python. PeerJ [online]. 2014, 2, e453. ISSN 2167-8359. Dostupné z: doi:10.7717/peerj.453
[3] VALÁŠEK, Michael. Mechatronika. Praha: České vysoké učení technické, 1995. ISBN 978-80-01-01276-5.
[4] MALÝ, Vladimír. Metoda PCA a vícerozměrová analýza falešných sousedů pro posouzení neurčitosti redukovaného stavového vektoru provozních veličin, bak. práce (ved. Ivo Bukovský), ČVUT v Praze, FS, 2011.
[5] JENČ, Jiří. Využití jednodeskových počítačů a neuronových sítí pro automatické rozpoznávání RZ vozidel, dipl. práce (ved. Ivo Bukovský), ČVUT v Praze, FS, 2015.

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Ivo Bukovský, Ph.D., U12110.3

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **24.10.2018** Termín odevzdání diplomové práce: **18.01.2019**

Platnost zadání diplomové práce: _____

doc. Ing. Ivo Bukovský, Ph.D. *podpis vedoucí(ho) práce*	prof. Ing. Milan Růžička, CSc. *podpis vedoucí(ho) ústavu/katedry*	prof. Ing. Michael Valášek, DrSc. *podpis děkana(ky)*

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

1.11.2018
Datum převzetí zadání

Podpis studenta

Poděkování

Rád bych poděkoval firmě Contipro a.s., především Ing. Marku Pokornému, Ph.D. za možnost vypracování diplomové práce v jejich laboratořích a za cenné rady v průběhu tvorby. Také bych chtěl poděkovat vedoucímu diplomové práce Doc. Ing. Ivu Bukovskému, Ph.D. za veškerou pomoc při jejím zpracování. Na závěr bych rád poděkoval mé rodině, která mě podporovala v průběhu celého studia.

Prohlášení

Prohlašuji, že jsem diplomovou práci na téma „Využití obrazové analýzy a jejích algoritmů při defektoskopii nanovlákenných vrstev“ vypracoval samostatně a s použitím uvedené literatury a pramenů.

V Praze, dne **Podpis**

Abstrakt

Cílem této práce je nalezení vhodného postupu pro detekci vad v konkrétních vzorcích nanovrstvých materiálů. V teoretické části jsou popsány metody obrazové analýzy, včetně 2D Fourierovy transformace a metody hlavních komponent (PCA). V praktické části je testována funkce jednotlivých metod a je vytvořen program v jazyce Python pro detekci vad z fotografií nanovrstvých materiálů. V práci je otestováno několik různých přístupů k detekci vad, které jsou v závěru porovnány a diskutovány.

Abstract

The aim of this work is to find a suitable procedure for detecting defects in specific samples of nanolayers. In the theoretical part are described the methods of image analysis, including 2D Fourier transform and principal component analysis. In the practical part, the function of individual methods is tested and a program in Python for the detection of defects from photos of nanomaterials is created. Several different approaches to defect detection are tested in the thesis, which are compared and discussed in the end.

Obsah

1 Úvod

Tato diplomová práce je zpracovávána ve spolupráci s farmaceutickou firmou Contipro a.s., která se zabývá výzkumem a výrobou kyseliny hyaluronové. Práce je zde zařazena v projektu pro automatizaci výrobního procesu nanovlákenných vrstev, konkrétně při závěrečné kontrole kvality. Vývoj výroby nanovláken kyseliny hyaluronové proběhl teprve nedávno. Veškeré další zpracování provádí personál laboratoře, což je do budoucna neefektivní. Při neopatrné manipulaci se navíc mohou vzorky nenávratně poškodit. Proto je kladen velký tlak na vývoj automatizovaného post-zpracování, jehož součástí musí být i velmi přesná kontrola vzorků. Z možných přístupů byla k detekci vybrána metoda fungující v závislosti na intenzitě procházejícího světla, které je zachyceno fotoaparátem. Získaná data jsou následně zpracovávána pomocí algoritmů obrazové analýzy. Oproti jiným možným metodám detekce (měření změn procházejícího vzduchu, měření elektrického odporu a další) je tato metoda především hardwarově nenáročná, bezkontaktní a nedestruktivní.

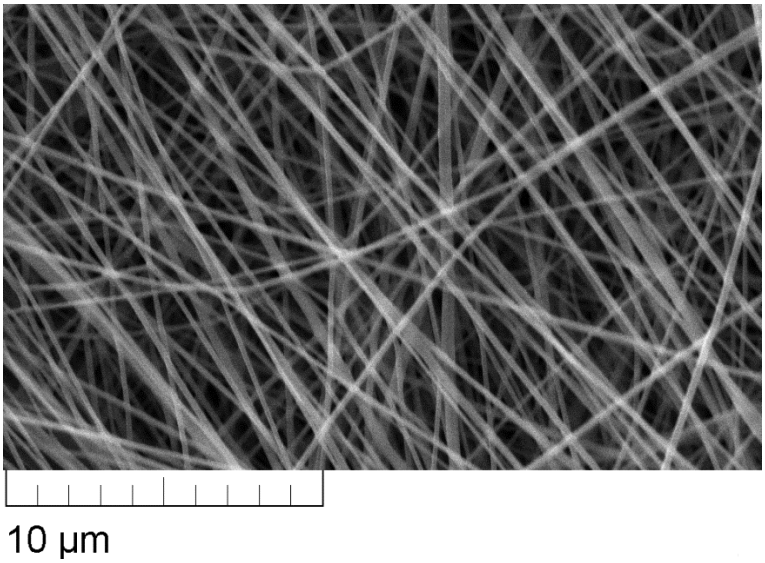
V první, teoretické části práce je nejprve popsán princip tvorby nanovláken. Tato část je zahrnuta především pro pochopení vzniku a významu různých typů vad. Dále jsou postupně popsány jednotlivé metody užívané při zpracování obrazu. Pozornost pak je věnována především 2D Fourierově transformaci a metodě hlavních komponent (PCA). Ty jsou využity na základě potřeby programově odstranit podklad vzorků pravidelné struktury, který komplikuje práci s běžnými metodami a ovlivňuje výsledek vyhodnocení testu.

Součástí práce je vytvořený program v jazyce python, který různými metodami detekuje vzniklé vady. Program zahrnuje dvě funkce pro metodu hlavních komponent, které jsou vytvořeny na základě dvou různých postupů. Obě funkce jsou poté porovnány s veřejně dostupnou funkcí. Praktická část popisuje celý proces od získání dat, přes segmentaci vzorku z fotografie, až po závěrečné nalezení jednotlivých vad. Jsou v ní názorně ukázány funkce jednotlivých metod i jejich kombinací a možnosti jejich nastavení. V závěru pak je navržen postup pro průmyslovou aplikaci, který by měl spolehlivě detekovat většinu možných chyb.

2 Teoretická část

2.1.1 Elektrostatické zvlákňování

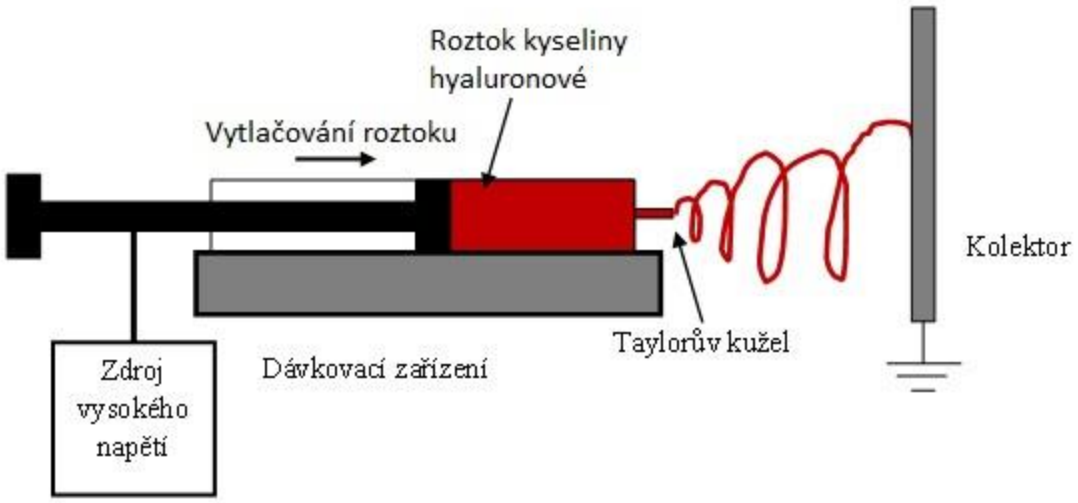
Nanovlákna a z nich vyrobené materiály jsou unikátní nanostruktury s obrovským potenciálem jak v technických oborech, tak i v lékařských aplikacích [1]. Aplikace ve filtrech, funkčních textiliích, vláknitých výztužích, tkáních nebo k distribuci léčiv je jen pár příkladů. K přípravě takových materiálů se primárně využívá elektrostatického zvlákňování (electrospinning).



Obrázek 1 Snímek nanovláken kyseliny hyaluronové pořízený elektronovým mikroskopem [pořízeno v květnu 2018]

2.1.2 Technologie

Electrospinning je metoda produkce vláken, která využívá elektrické síly k tažení vláken polymerních roztoků nebo tavenin až do průměru vláken řádově několika desítek nanometrů. Electrospinning kombinuje charakteristiky sprejování a konvenčního řešení suchého tažení vláken. Tento postup nevyžaduje použití vysokých teplot ani koagulační chemie (srážecí lázně). Mechanismus vzniku vláken je poměrně složitý proces, při němž hraje úlohu řada parametrů, jako je působení vysokého elektrického pole, typ zvlákňovaného materiálu, koncentrace polymeru, tvar a umístění elektrod, viskozita roztoku, povrchové napětí, elektrická vodivost, tenze par rozpouštědla, okolní teplota a vlhkost, rychlost proudění vzduchu a další. Souhrnně je nazveme vláknící parametry roztokové, elektrostatické a okolní.



Obrázek 2 Schéma principu elektrostatického zvlákňování [2]

V procesu elektrostatického zvlákňování je využito vysokého napětí s intenzitou elektrického pole přibližně mezi 100 až 1000 kV/m. Zdroj vysokého napětí je spojen s tryskou a přímo s roztokem, protilehlou elektrodu tvoří uzemněný kolektor. S rostoucí intenzitou elektrického pole vzniká na hrotu trysky (kapiláry) kónický tvar nazývaný Taylorův kužel. Ten vzniká následkem relaxace indukovaného náboje na povrchu kapaliny. Jinými slovy vnější elektrické síly na povrchu polymerního roztoku překročí síly povrchového napětí (kapilární síly) a dojde k erupci roztoku směrem ke kolektoru. V některých odvozených technikách se sekundárně využívá i dalších sil, jako jsou odstředivá nebo gravitační. Během chaotické fáze letu roztoku (bičování) se paprsek enormně napíná, odpařuje se rozpouštědlo a dochází tak k vytvoření ultrajemných vláken. Síla tohoto procesu spočívá v samoorganizaci polymerního materiálu do formy nanovláken jen s pomocí elektrického pole a extrémně rychlého odpaření rozpouštědla. Místa s větší tloušťkou vrstvy již navlákněného materiálu mají větší elektrický odpor, a tedy snižují pravděpodobnost dopadu dalších vláken. Výsledná struktura má pak vlastnosti náhodně uspořádaných vláken - netkaných materiálů. Výsledný poměr objemu vláken a prázdných míst (porozita) hotového materiálu bývá kolem 90 %.

Pro získání vzorků bylo využito stroje 4SPIN vyvinutého firmou Contipro, který využívá výše zmíněný postup výroby. Dále jsou uvedeny dva speciální případy, které by mohly v budoucnu znamenat zvětšení využitelnosti elektrostatického zvlákňování.

Obrázek 3 Laboratorní zařízení 4SPIN pro metodu elektrostatického zvlákňování

Koaxiální zvlákňování je vysoce sofistikovaná metoda umožňující vytvořit nanovlákennou strukturu i u jinak nevláknitelných materiálů [2]. Jde u strukturu jádro-plášť, kde plášť tvoří vláknitelný polymerní materiál, zatímco jádro může být tvořeno rozličnými i nevláknitelnými materiály včetně kapalin. Této metody se používá především k aplikaci léčebných látek, jako jsou například anestetika nebo antibiotika. K takovému vláknění je zapotřebí speciální dvojité kapiláry. Ty jsou umístěny koaxiálně, tedy jedna je uvnitř druhé.

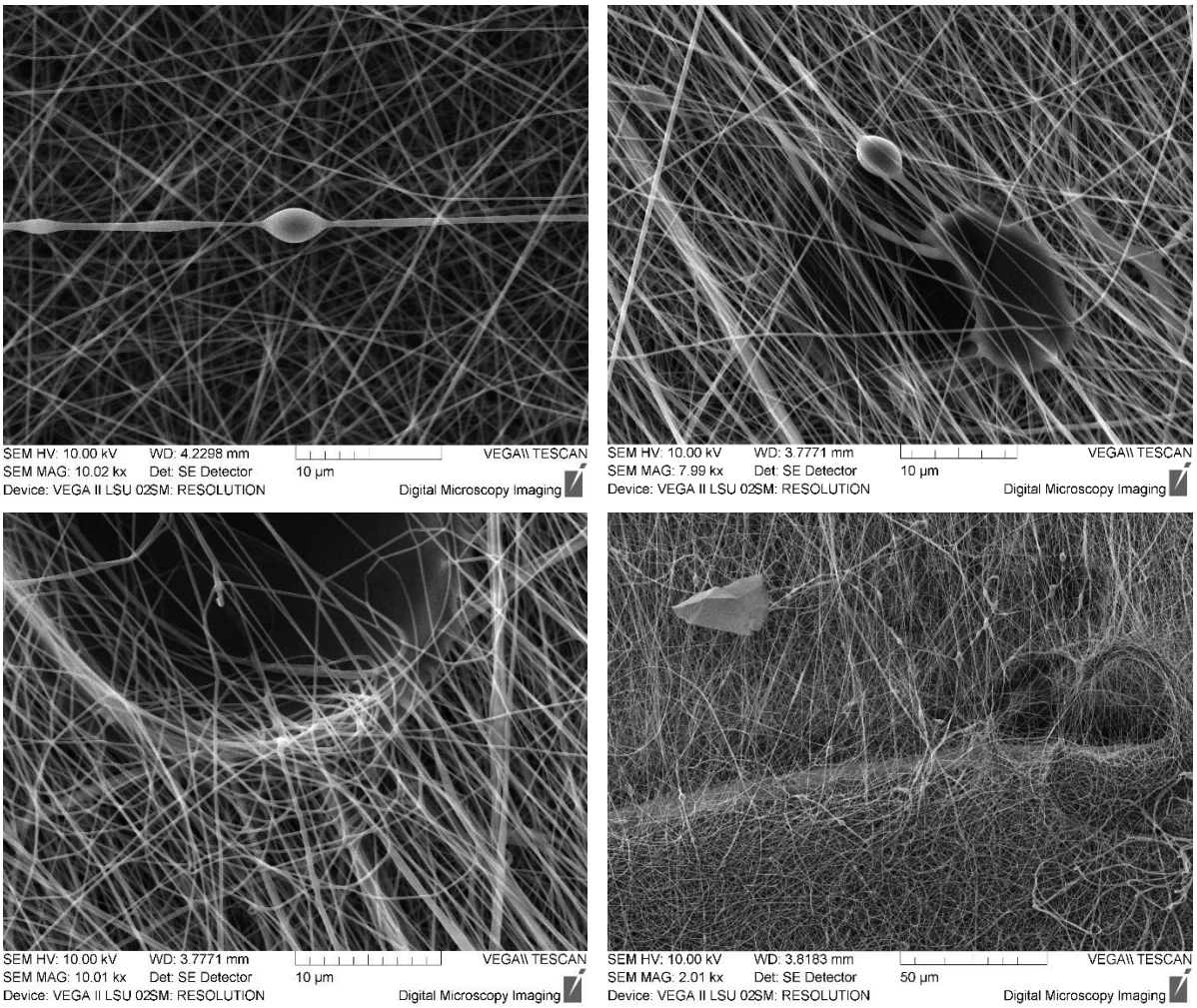
Střídavé elektrostatické zvlákňování je další speciální metoda přípravy nanovláken [2]. Oproti stejnosměrnému elektrostatickému zvlákňování je zde použito střídavé napětí s kmitočtem o desítkách Hz. Díky tomu vznikají vlákna s kladným i záporným nábojem. Jejich vzájemnou interakcí vzniká neutrální svazek, který se do určité míry formuje ještě před dopadem na kolektor. To s sebou přináší mnoho zajímavých změn a úkazů, které u stejnosměrného elektrostatického zvlákňování nenajdeme. Mezi hlavní změny patří zvýšení produktivity výroby a možnost více cíleně uložit vzniklý nanovlákenný svazek. Tato metoda je v současné době úspěšně využívána pro výrobu jádrových i bezjádrových přízí, což představuje zajímavou možnost v oblasti řízeného uvolňování léčiv a jeho cíleného dodávání v potřebném místě.

2.1.3 Strukturní vady

Elektrostatické zvlákňování ze své podstaty (náhodná a složitá trajektorie letu vlákna) produkuje vrstvy, které mohou obsahovat celou řadu nedostatků. Nevhodnou volbou vláknících parametrů nebo použití hůře vláknitelného materiálu může docházet k řadě vad. Z hlediska možnosti pozorování rozdělíme vady na mikroskopické a makroskopické.

Mezi mikroskopické vady patří morfologické vady vlákenné struktury, různé průměry vláken, kuličky na vláknech (korálkování), slepená vlákna, fólie, velké póry, aglomeráty, shluky nerozpuštěného polymeru, zbytky z rozpouštědel, aditiv, krystalky solí použitých při

chemických modifikacích polymerů apod. Několik příkladů vlastních snímků nanovlákenné morfologie je uvedeno na Obrázek 4. Detekce mikroskopických vad však není cílem této práce, jelikož jsou pozorovatelné pouze elektronovým mikroskopem. Ta je proto pro nedestruktivní kontrolu kvality zahrnutou případně i v automatizované výrobě zcela nevhodná.



Obrázek 4 Ukázka mikroskopických vad pořízených elektronovým mikroskopem a) kuličky na vláknech b) různé průměry vláken c) okraj makroskopické kapky d) cizí tělísko, nehomogenita vrstvy [pořízeno v květnu 2018]

Při hodnocení kvality z makroskopického hlediska požadujeme především

pravidelný tvar požadovaných rozměrů,

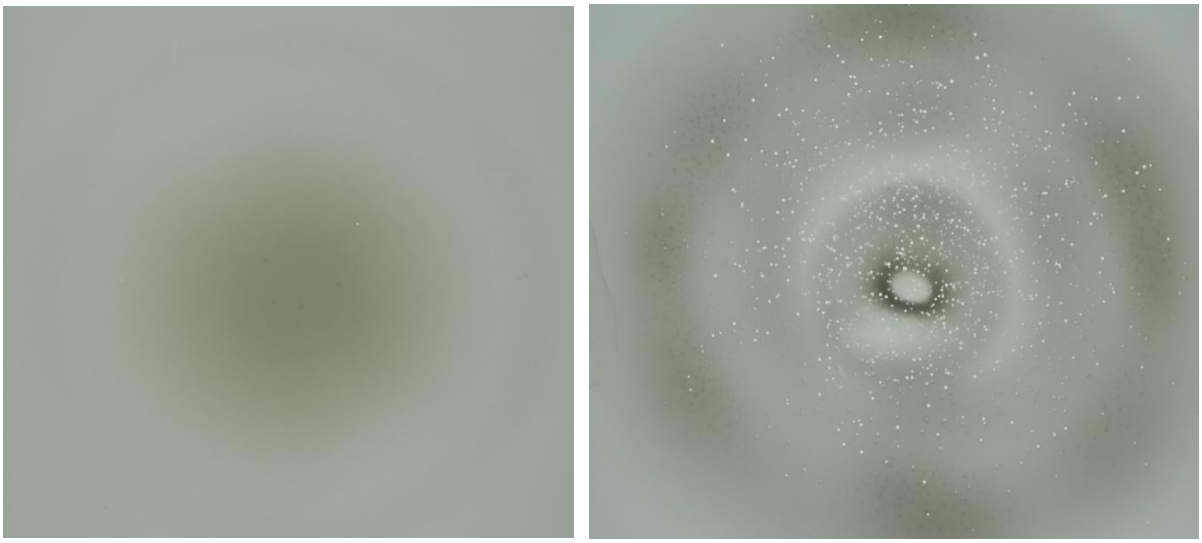
tloušťkově homogenní vrstvu,

plochu bez defektů (kapky, pruhy apod.),

okraje bez defektů (otřepy, překlady nebo shrnutí vrstvy apod).

V ideálním případě při základním uspořádání elektrod (tryska = kapilární jehla, kolektor = rovinná deska) vlákna dopadají na kolektor v kruhovém poli s tloušťkou charakterizovanou gaussovou distribucí (průměr středové oblasti je přibližně 10 cm). Problém nastává, vznikne-li na konci kapiláry dva a více taylorových kuželů. Všechny totiž mají stejný náboj, tudíž se

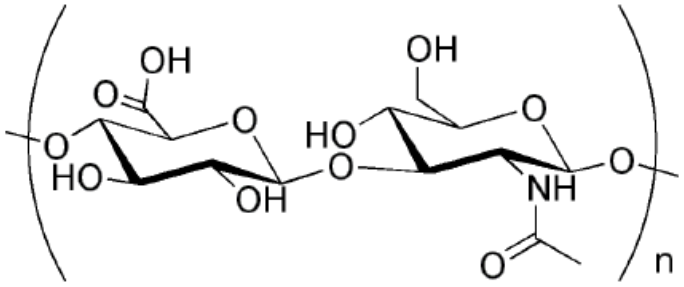
elektricky odpuzují. Prodlužující se vlákna jsou v letu těmito silami odkloněny mimo střed kolektoru. Na kolektoru potom vznikají místa se silnějšími a tenčími vrstvami. Plošné defekty (kapičky) ve vrstvě vznikají vyprsknutím roztoku z kapiláry nebo nedostatečným odváděním rozpouštědla. Výsledné vlastnosti vzorku jsou tímto velmi ovlivněny kvůli enormnímu poměru průměru defektu kapičky a průměru nanovlákna. Obě tyto vady jsou předvedeny na Obrázek 5.



Obrázek 5 Výsledek vláknění na průhlednou fólii a) vhodně nastavené parametry b) nevhodně [pořízeno v květnu 2018]

2.2 Kyselina hyaluronová

Hyaluronát, hyaluronidát, též kyselina hyaluronová, hyaluronan, je nesulfonovaný glykoaminoglykan, který je nejdůležitější složkou mezibuněčné hmoty [3]. Aktivně se účastní imunologických procesů jako signální molekula a ovlivňuje mobilitu, adhezivitu buněk v rámci jejich proliferace a diferenciace. Urychluje hojení a epitelizaci. Ve velkém množství se nachází v očním sklivci, synoviální tekutině a kůži. Rovněž tvoří slizovité obaly vajíček některých organizmů. Po chemické stránce se jedná o nevětvený mukopolysacharid složený z D- glukuronové kyseliny a N- acetylglukoaminu v poměru 1:1.



Obrázek 6 Strukturní vzorec kyseliny hyaluronové [3]

Pro výrobu kyseliny hyaluronové se používá modifikované bakterie streptococcus zooepidemicus [4]. Využívá se vlastnosti této bakterie obalovat se relativně silnou vrstvou

kyseliny hyaluronové. Příprava pak spočívá především v kultivaci (pěstování) této bakterie v živném roztoku a následné několikastupňové filtraci a čištění. Míra čištění pak závisí na konečné implementaci. Méně kvalitní surovina najde využití v kosmetice, kvalitnější se používá ve farmaceutickém průmyslu. Nejkvalitnější produkt se využívá v oční medicíně a plastické chirurgii.

2.3 Metody zpracování obrazu

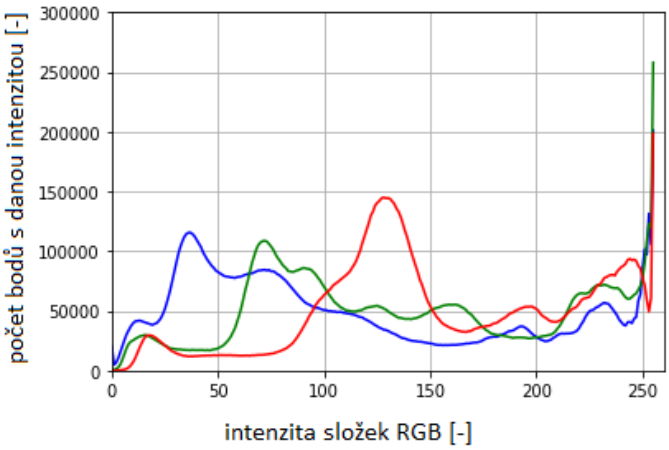
2.3.1 Vlastnosti obrazu

Vstupem pro celou práci uvažujeme fotografii vzorků. Po implementaci do programovacího prostředí Python již s touto fotografií pracujeme jako s maticí o rozměrech (x,y,z), kde x a y jsou hodnoty rozměrů fotografie v pixelech, z pak představuje zastoupení jednotlivých barev. Nejčastěji, a i v našem případě, se jedná o kanály RGB rozložení (červená, zelená, modrá), kde sečtením různé intenzity těchto tří barev dostaneme barvu výslednou [5]. Nulové využití všech tří barev dává barvu černou, maximální barvu bílou. Existují ale i další, například CMY (azurová, purpurová, žlutá), kde se se barvy na rozdíl od RGB odečítají. Tento typ skládání barev je využíván v tiskárnách. Nulové intenzitě barev odpovídá bílá barva (barva prázdného listu). Černá barva se pak skládá z maximálního využití všech 3 barev. Pro zvýšení rychlosti černobílého tisku a úspoře barev je využíváno kanálů CMYB, kde speciální kanál černé barvy nahrazuje kombinaci zbylých 3 barev. Intenzity jednotlivých barev standardně nabývají celočíselných hodnot 0-255 (využívá se 8 bitů na obrazový element). Celkový počet dosažitelných barev je tedy 256^3, tedy přes 16 milionů barevných kombinací.

2.3.2 Histogram

K prvotnímu získání informací o obraze slouží histogram (stejně jako ve statistice). Graficky zobrazuje zastoupení intenzit jednotlivých složek barev. Je z něj patrné spektrum zastoupení jednotlivých odstínů. V klasické fotografii histogram svojí šíří ukazuje, zda je pořízená fotografie kvalitní, či nikoliv. Zde by měl být histogram co nejširší, neměl by však ve velké míře dosahovat hraničních hodnot. Na Obrázek 7 je vidět špatné nastavení fotoaparátu. Fotografie obsahuje velké množství pixelů bílé barvy, které tak nenesou téměř žádnou informaci o obsahu foceného objektu.

V technické fotografii může být situace opačná, kdy úzký pás v histogramu poukazuje na kvalitní objekt/výrobek bez defektů, které tak nejsou v jiné oblasti histogramu.



Obrázek 7 Příklad histogramu příliš světlé fotografie

2.3.3 Ekvalizace

Zobrazí-li se nám v histogramu pouze úzké pásmo používaných intenzit, je v určitých případech vhodné toto pásmo roztáhnout. Dosáhneme tím zvýraznění jinak málo viditelných detailů. Jinými slovy je potřeba zvýšit kontrast ať už celého obrazu, nebo jen vybrané části. Možnost automatické ekvalizace je znázorněna rovnicí *(2.1)*, kde matice X je vstupní obraz. Často se lze též setkat s ruční ekvalizací několika vybraných pásem, ta je však pro automatizované použití nevhodná.

$$Ekvalizace = \left(\frac{X - \left(\frac{max(X) + min(X)}{2} \right)}{max(X) - min(X)} + 0{,}5 \right) 255. \tag{2.1}$$

2.3.4 Konvoluce

Výpočet diskrétní konvoluce spočívá v součtu součinů jednotlivých koeficientů masky a hodnot jednotlivých bodů (pixelů) obrazu a bodů okolí [5]. Zde hodnota w_5 představuje hodnotu koeficientu masky pro zkoumaný bod, ostatní hodnoty w_i jsou hodnoty masky pro okolní body z_i. Pro obecnou konvoluční masku

$$\begin{bmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{bmatrix}$$

je výpočet dán rovnicí

$$\boldsymbol{R = w_1 z_1 + w_2 z_2 + \ldots + w_9 z_9 = \sum_{i=1}^{9} w_i z_i.} \tag{2.2}$$

Hodnota výstupního parametru R je zapsána do nové matice na stejnou pozici jako zkoumaný bod.

2.3.5 Detekce hran

V této kapitole bylo čerpáno především z [6], [7] a [8]. Detekce hran je postup v digitálním zpracování obrazu sloužící k nalezení oblastí pixelů, ve kterých se podstatně mění jas. Nalezené hrany se však nemusí shodovat s hranicemi skutečného objektu. Mohou vznikat a zanikat v závislosti na úhlu pohledu a kvalitě osvětlení. Hrany tedy nalézáme na rozhraní objektů, ale i na rozhraní světla a stínu. Čáry v objektu (o tloušťce několika pixelů) generují dvě hrany, každou na jedné straně. Reálná hrana však na rozdíl od teoretické bývá zašuměná. K detekci změny jasové funkci se využívá třech hlavních principů.

1. **Hledání maxim prvních derivací pomocí konvolučních masek.** (Roberts, Prewittová, Sobel, Kirsch, Robinson, Canny, …)

Hranu definujeme jako velkou změnu jasové funkce. V místě hrany proto bude velká hodnota první derivace. Směr maxima derivace pak bude kolmo na hranu. Pro jednodušší výpočty se většinou derivace zjišťuje jen v 8 směrech po 45 stupních. Samotná derivace se aproximuje pro každý směr vhodnou konvoluční maskou.

Nejstarší a nejjednodušší je Robertsova metoda. Ta využívá konvoluce s velmi jednoduchými konvolučními maskami

$$h_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad h_2 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}, \quad h_3 = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}, \quad h_4 = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}.$$

Metoda je velmi citlivá na šum, protože okolí použité pro aproximaci je velmi malé (pouze 4 body). Na rozdíl od dalších metod udává směr růstu jen ve čtyřech směrech po 90 stupních.

Konvoluční masky Prewittové, Sobela, Kirsche a Robinsona všechny využívají masku 3x3 a jsou si konstrukčně velmi podobné. Obdobně se vytváří i větší masky s podrobnějším směrovým rozlišením. Sobelovy masky

$$h_1 = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}, \quad h_2 = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix}, \quad h_3 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad h_4 = \begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix},$$

$$h_5 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, \quad h_6 = \begin{bmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{bmatrix}, \quad h_7 = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, \quad h_8 = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{bmatrix}$$

jsou uvedeny jako příklad.

2. **Hledání průchodů druhých derivací nulou.** (Marr-Hildreth)

Výše zmíněné aproximování prvních derivací má několik nevýhod. Konkrétní hodnoty prahu je vhodné nastavit různě pro různé obrázky. Velikost masky by měla odpovídat velikosti detekovaných detailů. Metody jsou též značně citlivé na šum.

Proto byla koncem sedmdesátých let formulována metoda využívající k detekci druhých derivací. Podle ní se hrana nachází v průchodu druhé derivace nulou. Jelikož druhé derivace jsou ještě náchylnější na šum, je zapotřebí využít filtr. Například lineárnímu filtru odpovídá v konvoluční masce 2D Gaussovské rozložení. Po derivování a zavedení normalizačního koeficientu c (zajišťuje, aby součet koeficientů v masce byl 0) dostaneme konvoluční masku. Ta bývá označována jako LoG operátor (Laplacian of Gaussian)

$$h(x,y) = c\left(\frac{x^2+y^2-\sigma^2}{\sigma^4}\right)e^{-\frac{x^2+y^2}{2\sigma^2}}. \tag{2.3}$$

Aproximace operátoru konvoluční maskou 5x5 vypadá následovně

$$\begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix}.$$

Aproximovat LoG lze též pomocí diference dvou obrazů, které vznikly konvolucí s Gaussiánem o různém σ.

Hledání průchodů nulou již nelze jednoduše zjistit prahováním. Možné je například využít jednoduchou masku detekující změnu znaménka (předpoklad pro hranu).

Získané hrany je ještě vhodné filtrovat v závislosti na významnosti první derivace.

Nevýhodou této metody je přílišné vyhlazení ostrých rohů. Metoda se navíc snaží hranové body spojovat do uzavřených smyček. Tvoří pak tzv. talíř špaget.

3. **Lokální aproximaci obrazové funkce parametrickým modelem**, např. polynomem dvou proměnných a následně vypočítáním derivace tohoto parametrického modelu. (V počítačovém vidění zavedl R. Haralick).

V současnosti nejpoužívanějším a nejlepším je **Cannyho hranový detektor**, který patří do první skupiny. Tento algoritmus se skládá z několika kroků.

1. Přibližně najde směry gradientu Sobolovou maskou.
2. Pro každý pixel najde 1D derivaci ve směru gradientu pomocí optimální konvoluční masky. Optimální maska vznikne spojením Gaussovského filtru a první derivace. Díky tomu detektor není tolik náchylný na šum v obraze.
3. Najde lokální maxima derivací
4. Hranové body určí prahování s hysterezí. Prakticky to znamená, že porovnáme všechna lokální maxima s první prahovou hodnotou. Tím získáme několik bodů, které leží na hranicích. Následně na tyto body připojujeme další body kolmo na směr derivace. Tyto bodu již porovnáváme s druhou nižší prahovou hodnotou.

2.3.6 Segmentace

Ve zpracování obrazu není hlavním cílem obraz zpracovávat jako celek [9], [10]. Většinou se totiž soustřeďujeme jen na určitou část. Hlavním cílem je tedy rozpoznat objekty v obraze a rozeznat je od nevýznamného pozadí. Tomuto procesu říkáme segmentace obrazu.

To však obecně bývá velmi složité, většinou je nezbytná předešlá znalost uživatele o řešeném problému. Nejjednodušší případ nastává, když je obraz tvořen konstantními objekty na pozadí konstantního jasu. V obrazu lze sledovat nejen hodnotu jasu, ale také další vlastnosti, jako barvu, texturu a další.

Mezi segmentační techniky patří prahování, hledání hranic, vytváření oblastí (split and merge), srovnávání se vzorem, texturní segmentace a další, případně jejich kombinace.

2.3.7 Prahování

Jednoduchá a často používaná je metoda prahování [9]. Jednotlivé pixely jsou porovnány s prahovou hodnotou. Vyšší hodnoty se přepíší na hodnotu 255, tedy bílou barvu, nižší na hodnotu 0, tedy černou barvu. Hodnotu prahu můžeme vyčíst například z histogramu. Chceme tím oddělit dvě největší zastoupení pixelů v obrázku, tedy objekt a pozadí. Existují i metody, které tento práh najdou automaticky. Otsuova metoda hledá takový práh, který minimalizuje vážený rozptyl σ_w dvou tříd (objektu a pozadí). Známe-li přibližnou velikost objektu (kolik % objektu zabírá), nabízí se využít procentuální prahování. Adaptivní prahování dělí obraz na několik podoblastí, pro každou pak využije jinou hodnotu prahu (např. Otsuovou metodou). Takto je například vhodné prahovat obrazy pořízené v nerovnoměrném osvětlení.

2.3.8 Dilatace

Dilatace se využívá k zaplnění malých mezer a úzkých zálivů v obraze [11]. Tato operace následuje na operaci prahování. Využívá se tzv. strukturních prvků B. Definice je uvedena v rovnici *(2.4)*.

$$X \oplus \mathrm{B} = \{p_d \in \varepsilon^2, p = x + b, x \in X, b \in B\} \tag{2.4}$$

Metoda vytváří novou nulovou matici $\mathbf{p_d}$ v binárním prostoru ε^2, kterou postupně transformuje podle součtu jednotlivých bodů obrazu **X** a strukturního prvku **B**. Vyhovuje-li tento součet (v již prahovaném obraze to znamená: bílá barva v obrazu = 255 + hodnota v masce = 1, tedy výsledek 256), zvětšíme hodnotu v tomto bodě ale i v jeho okolí o 1. Velikost okolí závisí na užitém strukturním prvku. V obecném černobílém obraze lze hodnotu požadovaného součtu volit jinou a užít znaménka ≥ (obdoba prahování). Pro zrychlení výpočtu pro naprahovaný obraz též lze využít binární prostor. Pro vykreslení výsledného obrazu hodnoty 1 (v binárním obraze 1 + 1 = 1) přepíšeme zpět na hodnotu bílé barvy 255, hodnoty 0 ponecháme 0. Tím nám vznikne obraz, který zvýrazňuje bílé objekty na černém pozadí.

Obdobně metoda funguje i pro černý obraz na bílém pozadí. Důležitá vlastnost metody je zvětšení výstupního obrazu právě o velikost strukturního prvku **B**. Přímo se proto nabízí následné použití eroze se stejně velkou maskou, která naopak obraz zmenšuje. Příklady strukturních prvků

$$\begin{matrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{matrix}, \quad \begin{matrix}0&1&1&1&0\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\0&1&1&1&0\end{matrix}$$

představují jádra velikosti 5x5 s čtvercovým a kruhovým základem.

2.3.9 Eroze

Dilatace tedy ke všem bílým oblastem připojí bílé okolí. Díky tomu zvýrazníme objekty, což je naším cílem, ale také velmi zvýrazníme šum. Při použití výše uvedeného strukturního prvku se jeden pixel bílé barvy změní na bílý čtverec 5x5 pixelů. [11] Definice eroze je

$$\mathrm{X\Theta B} = \{p_e \in \varepsilon^2 , \mathrm{p} + \mathrm{b} \in \mathrm{X}, \forall \mathrm{b} \in \mathrm{B}. \tag{2.5}$$

Výstupem je matice **X**, která nabývá hodnot 255, je-li strukturní prvek **B** shodný s okolím daného bodu v matici $\mathbf{p_e}$. V ostatních případech nabývá hodnoty 0.

Pro odstranění šumu je zapotřebí použít větší strukturní element než pro dilataci. Maximální velikost šumu, kterou takto dokážeme potlačit, je rozdíl strukturních prvků $\mathrm{p_e} - \mathrm{p_d}$.

Použití dilatace a následně eroze se souhrnně nazývá uzavření.

Někdy je vhodný i opačný postup, tedy použít nejdříve erozi a následně dilataci, jak je znázorněno na Obrázek 8. Tomuto postupu se říká otevření.

V praxi se pak můžeme setkat s vícenásobným použitím těchto metod.

Obrázek 8 Obrázek se čtverci o velikosti 1, 3, 5, 7, 9, a 15 pixelů. Ukázka eroze a následné dilatace se stejným strukturním elementem. [7]

2.4 Fourierova transformace

Touto metodou se již na začátku 19. století zabýval francouzský fyzik a matematik Jean Baptiste Joseph Fourier, který Fourierových řad využíval k problémům toků tepla. Na jeho počest byla metoda nazvána Fourierova transformace (FT).

Fourierova transformace je matematická metoda, která dovoluje analyzovat průběh libovolného signálu a převést jej na součet (co)sinových signálů vhodných frekvencí a amplitud.

Fourierova transformace je modifikací Fourierovy řady a je užitečná pro řešení mnoha různých problémů [12]. Používá se např. pro převedení řešení diferenciálních rovnic na řešení algebraických rovnic nebo pro frekvenční analýzu časově proměnných signálů. Nejznámějším příkladem využití metody je filtrace časově proměnného zvukového signálu. Velmi snadno lze eliminovat ze signálu rušivé frekvence o stálé hodnotě (v praxi se často jedná o vysoké pískání). V obrazové analýze se využívá analogie, kde namísto časové proměnné je proměnnou poloha, a to hned ve dvou dimenzích. Ta se používá v aplikacích obrazové analýzy, jako jsou filtrování, rekonstrukce, či komprese obrazu.

2.4.1 Princip Fourierovy transformace

Při tvorbě této kapitoly bylo čerpáno především z [5], [13], [14]

Převod mezi časovou a frekvenční oblastí popisují následující rovnice. Rovnice *(2.6)* představuje přímou Fourierovu transformaci, rovnici *(2.7)* pak nazveme zpětnou (inverzní) transformací. FT časové funkce je komplexní funkcí frekvence, jejíž velikost udává množství (amplitudu) té dané frekvence v původní funkci. Její fáze udává posunutí dané cosinusoidy od počátku.

$$X(f) = \int_{-\infty}^{+\infty} x(t) e^{-i2\pi tf} dt \tag{2.6}$$

$$x(t) = \int_{-\infty}^{+\infty} X(f) e^{+i2\pi tf} df \tag{2.7}$$

V případě počítačového zpracování signálů máme k dispozici vždy jen vzorky funkce x(t) v diskrétních časových okamžicích. Zavádíme tedy tzv. diskrétní Fourierovu transformaci, kterou dostaneme formálním nahrazením integrálu integrálním součtem s dělením odpovídajícím periodě vzorkování. Definiční vztah pro diskrétní Fourierovu transformaci je tedy

$$X(k) = \frac{1}{N} \sum_{n=0}^{N-1} x(n) e^{-2\pi \frac{nk}{N}} \tag{2.8}$$

$$x(n) = \sum_{k=0}^{N-1} X(k) e^{+2\pi \frac{nk}{N}} \tag{2.9}$$

Lineární operace v jednom systému (časovém i frekvenčním) mají příslušné operace v systému druhém, které jsou někdy vhodnější pro výpočet. Derivace v časové oblasti koresponduje s násobením danou frekvencí ve frekvenčním spektru, proto jsou některé diferenciální rovnice snadněji analyzovány v oblasti frekvenční. Konvoluce dvou vstupů v časové oblasti odpovídá obyčejnému násobení ve frekvenčním pásmu.

Přímá a inverzní diskrétní 2D Fourierovy transformace jsou napsány v rovnicích *(2.10)* a *(2.11*). Právě této transformace se využívá v obrazové analýze, kde proměnné m,n představují souřadnice původního obrazu, souřadnice k a l jsou potom polohové frekvence ve směrech m a n.

$$X(k,l) = \frac{1}{MN} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} x(n,m) e^{-2\pi(\frac{kn}{N} + \frac{lm}{M})} \tag{2.10}$$

$$x(m,n) = \sum_{k=0}^{N-1} \sum_{l=0}^{M-1} X(k,l) e^{+2\pi(\frac{kn}{N} + \frac{lm}{M})} \tag{2.11}$$

Výstupem FT je obraz komplexních čísel, který můžeme zobrazit ve dvou obrazech, buď jako reálnou a imaginární část, nebo jako velikost (magnitude) a fázi (phase). Ve zpracování obrazu často stačí pracovat pouze s velikostí obrazu, jelikož obsahuje většinu geometrických informací o původním obrazu. Nicméně pro zpětnou transformaci je nutné uchovat obě složky. Hodnoty Fourierova obrazu nabývají na rozdíl od původního (0-255) daleko většího rozpětí hodnot. Proto se hodnoty počítají a zapisují ve formátu float.

$$|X(k,l)| = \sqrt{Re^2\,(k,l) + Im^2(k,l)} \tag{2.12}$$

$$\varphi(k,l) = tan^{-1}\left[\frac{Im(k,l)}{Re(k,l)}\right] \tag{2.13}$$

Provádění výpočtu diskrétní Fourierovy transformace je velmi časově náročné [5], [15]. Je k tomu potřeba N^2 operací. Už pro malý počet vzorků čas výpočtu neúměrně narůstá. Proto je nutno užít k výpočtu jiných algoritmů, které využívají speciálních vlastností definice transformace. Důležitým krokem ve vytváření časové úspory je minimalizace počtu násobení, neboť na všech výpočetních systémech je násobení časově náročnější než sčítání, což platí obzvláště pro komplexní čísla.

Proto bylo vyvinuto několik metod umožňující rychlejší výpočet. Souhrnně je nazýváme FFT (Fast Fourier transform). Jsou to především Cool-Turkeyho algoritmus, který snižuje složitost výpočtu z N^2 na N*log(N). Je zde však nutná podmínka, že $N=2^k$, kde k je přirozené číslo. Good-Thomasův algoritmus předpokládá, že velikost N vstupní DFT je rozložitelná na nesoudělná

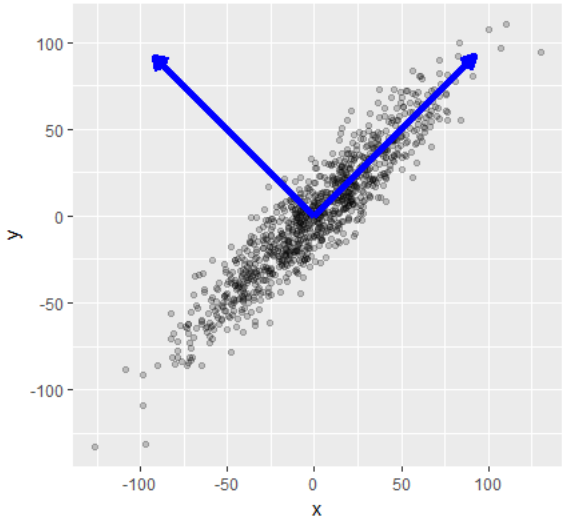
čísla N_1 a N_2. Winogradův algoritmus pracuje pouze s velikostí N, kde N jsou pouze prvočísla nebo mocniny prvočísel.

2.5 Metoda hlavních komponent (PCA)

V této kapitole je popsána metoda hlavních komponent (Principal component analysis, PCA) [17], [18]. Ta byla zavedena roku 1901 anglickým matematikem, filozofem a zakladatelem oboru matematické statistiky Karlem Pearsonem jako popisná statistická metoda, která sloužila zejména k redukci mnoharozměrných dat. Tu v roce 1933 následně rozšířil americký statistik Harold Hotelling. Zobecnil postup aplikací komponentní analýzy na náhodné vektory a navrhl použití metody pro rozbor kovarianční struktury proměnných. V současné době se doporučuje touto metodou začít každou vícerozměrnou úlohu.

Hlavním principem metody je zjednodušení popisu vzájemně korelovaných, neboli lineárně závislých proměnných. Snahou je snížit počet dimenzí úlohy bez ztráty zásadní informace, což umožní rychlejší a efektivnější práci s mnoharozměrnými daty. Jednotlivé měřené veličiny totiž často vykazují silnou závislost (korelaci). Proto jich lze celé skupiny nahradit pouze několika ortogonálními hlavními komponentami, které nesou prakticky stejnou informaci, jako proměnné původní. Tato metoda lze popsat jako lineární transformace (někdy též Hotellingova) původních proměnných na nové, vzájemně nekorelované, které mají vhodnější vlastnosti a je jich výrazně méně.

V grafické podobě si lze tuto metodu představit ve dvou dimenzích jako např. na Obrázek 9. Metoda hledá směr největšího rozptylu. Tento směr pak představuje první hlavní komponentu. Druhou hlavní komponentu představuje směr ortogonální (kolmý) s první hlavní komponentou s největším rozptylem a analogicky až po poslední komponentu. Hlavní problém metody nastává, jsou-li v datech odlehlé body (chyby, způsobené například špatným zápisem). Na
Obrázek 9 by to mohl být například bod [953,2 ; 95,43], kde byla chybně zapsána desetinná čárka. První hlavní komponenta by pak mířila přímo na tento bod, což je v rozporu se správným použitím. K odstranění těchto bodů se používají různé detekční nástroje, případně lze využít algoritmů robustní PCA, která dokáže i takto znehodnocená data správně zpracovat. Vzhledem k povaze této práce, kdy data pochází ze stabilního prostředí a jsou zpracovávána přímo bez možnosti vnesení chyby člověkem by tyto algoritmy byly spíše ke škodě.



Obrázek 9 Grafické znázornění metody PCA [16]

V obrazové analýze se PCA používá hlavně pro odstranění šumu z obrazu. Lze ale využít například i při rozpoznání obličeje, kde porovnáním pouze hlavních komponent jednotlivých obrázků získáváme spolehlivý detekční nástroj.

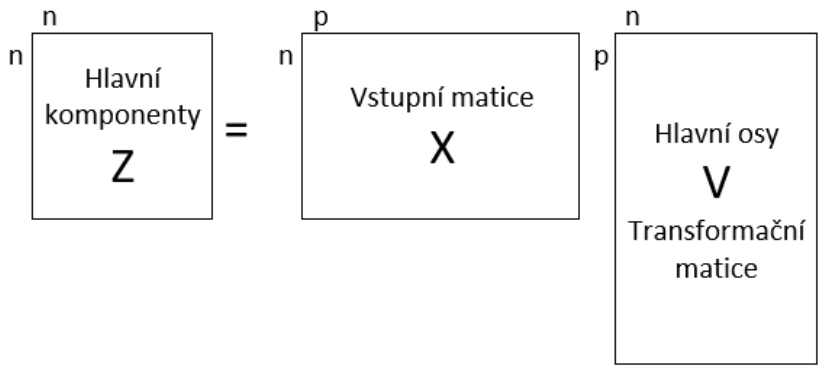
2.5.1 Princip PCA

V této kapitole bylo čerpáno především z [17], [18] a [19] .

Podstatou metody je transformace z jednoho souřadnicového systému do nového souřadnicového systému. Jako nový souřadnicový systém je volen ortogonální prostor. Z definice je pak zaručena vzájemná nezávislost jednotlivých proměnných. Jednotlivé souřadnice pak musí být řazeny sestupně od nejvýznamnější po nejméně významnou. Transformaci provedeme dle rovnice *(2.14)*.

$$Z = X * V \tag{2.14}$$

Vstupem metody PCA je matice **X** (n x p), kde p je počet měření (vzorků) a n počet proměnných. Matice **X** musí být sloupcově centrovaná, tzn. součet prvků v každém sloupci je roven nule. Výstupem je čtvercová ortogonální matice **Z** (n x n), obsahující p hlavních komponent pro p hlavních os. Matice **V** (p x n) je transformační matice.



Obrázek 10 Maticové znázornění metody PCA

Transformační matici **V** získáme jako po řádcích zapsané vlastní vektory kovarianční matice vypočítané ze vstupní matice.

$$C = X^T X/(n-1) \tag{2.15}$$

Kovarianční matici definujeme vztahem *(2.15)*. Ta znázorňuje vzájemné vztahy jednotlivých veličin vstupu X. Je symetrická, proto jsou všechna vlastní čísla reálná, dále je pak pozitivně semidefinitní a tedy platí $\boldsymbol{y^T X X^T y} \geq 0$ pro libovolný nenulový vektor **y**. Vlastní čísla kovarianční matice **C** (p x p) jsou proto nezáporná. Diagonála obsahuje hodnotu rozptylu jednotlivých proměnných. Příklad kovarianční matice je na Obrázek 11 b).

Setkat se lze s obdobou rovnice *(2.15)*, která je však dělena pouze hodnotou n namísto (n-1). To může způsobovat jemné odchylky ve výsledcích při použití předprogramovaných funkcí PCA v různých knihovnách či programovacích jazycích. Na samotnou funkci metody a zpětnou transformaci to nemá žádný vliv.

Korelační matice

	SEPALLEN	SEPALWID	PETALLEN	PETALWID
SEPALLEN	1.000	-0.118	0.872	0.818
SEPALWID	-0.118	1.000	-0.428	-0.366
PETALLEN	0.872	-0.428	1.000	0.963
PETALWID	0.818	-0.366	0.963	1.000

Kovarianční matice

	SEPALLEN	SEPALWID	PETALLEN	PETALWID
SEPALLEN	0.686	-0.042	1.274	0.516
SEPALWID	-0.042	0.190	-0.330	-0.122
PETALLEN	1.274	-0.330	3.116	1.296
PETALWID	0.516	-0.122	1.296	0.581

Obrázek 11 a) korelační matice b) kovarianční matice

Při použití metody PCA je nejprve data vhodné upravit. Nutnou úpravou je sloupcová centralizace. Tu jednoduše dostaneme odečtením průměru daného sloupce matice **X** od hodnot náležící daným sloupcům matice **X**. Při zpětné transformaci je pak třeba tento průměr zpět přičíst. Další úpravy pak záleží na dané úloze.

Nejčastější úpravou je normování, kde sjednotíme rozptyl každé proměnné na jednotkovou hodnotu. Poté již ale nemluvíme o kovarianční, nýbrž korelační matici. Využívá se hlavně když každá proměnná je v řádově rozdílných jednotkách. Proměnné s nízkými hodnotami by byly

brány jako nevýznamné a metoda PCA by s nimi pak pracovala jen minimálně. Chybně by se pak ve výsledku mohly promítnout například nezajímavé hodnoty pozadí s velmi malým rozptylem, který bychom touto úpravou zvětšili. Ve zpracování obrazu však jsou všechny hodnoty proměnných v rozmezí 0 – 255 a označují intenzitu světla, tudíž by tato operace byla spíše na škodu a zkomplikovala by zpětnou transformaci. Další možností je použití logaritmu a dalších matematických operací.

Základem konstrukce hlavních komponent je nalezení vlastních čísel a vlastních vektorů kovarianční matice **C** (p x p). Vlastní čísla λ vypočteme dle *(2.16)* a seřadíme je dle velikosti.

$$det(C - \lambda E) = 0 \tag{2.16}$$

Vlastní vektory u_i pak k jednotlivým vlastním číslům získáme z rovnice *(2.17)* a po sloupcích z nich vytvoříme matici **V**.

$$(C - \lambda_i E)u_i = 0 \tag{2.17}$$

Vlastní vektory a vlastní čísla lze též získat diagonalizací matice **C**. Podmínky pro diagonalizaci jsou splněny, jelikož matice **C** je reálná a symetrická. **L** (n x n) je matice vlastních čísel.

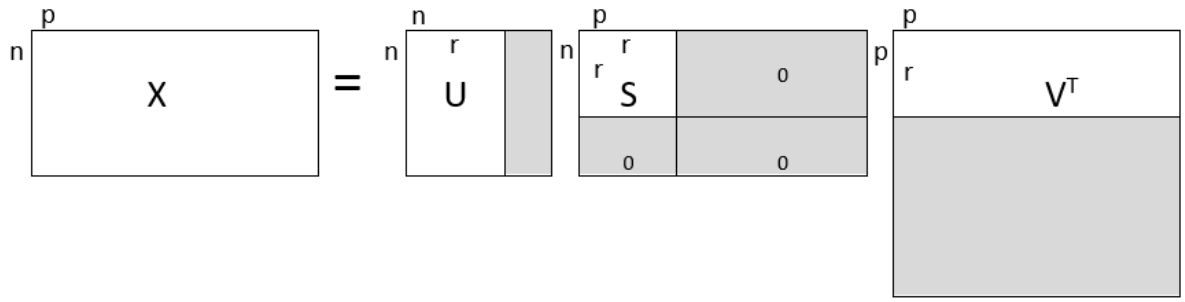
$$C = VLV^T \tag{2.18}$$

Alternativně je možné k získání hlavních os a hlavních komponent použít singulární rozklad (singular value decomposition, SVD). Ta rozkládá libovolnou obdélníkovou matici na tři speciální matice.

$$X = U * S * V^T \tag{2.19}$$

Obecně je matice **X** komplexní proměnnou o velikosti (n x p). Pak i matice **U** a **V** jsou komplexní. Jejich speciální vlastností je, že jsou obě unitární. Unitární matice je taková čtvercová komplexní matice, jejíž hermitovsky sdružená matice je současně maticí inverzní. Ve zpracování obrazu je však matice **X** reálná. Matice **U** (n x n) a **V** (p x p) jsou proto také reálné. Unitární matice v reálném prostoru se rovná matici ortogonální, tedy platí $U^T U = V^T V = E$, kde **E** je jednotková matice. Matice **S** (n x p) je pseudodiagonální matice singulárních reálných čísel matice **X**. Tato singulární čísla jsou seřazena sestupně a je jich právě tolik, jaká je hodnost matice **X** (tento počet označíme r). Pro singulární čísla platí, že jsou to odmocniny z vlastních čísel kovarianční matice *(2.15)*. Sloupce matice **U** jsou vlastní vektory matice XX^T a řádky matice $\mathbf{V^T}$ jsou vlastní vektory matice $X^T X$.

Z praktických důvodů se používá ekonomický tvar singulárního rozkladu. Ten dostaneme z klasického singulárního rozkladu odstraněním částí matic, které jsou při násobení s maticí **S** nulové. Z matice $\mathbf{V^T}$ transpozicí získáme matici **V**, která se shoduje s transformační maticí **V**.



Obrázek 12 Maticové znázornění ekonomického tvaru metody SVD

Z rovnice *(2.20)* lze vidět vzájemnou závislost PCA a SVD. Diagonální matice **S** je rovna své transpozici, $U^T U = E$.

$$C = X^T X/(n-1) = VSU^T USV^T/(n-1) = V\frac{S^2}{n-1}V^T \tag{2.20}$$

Matice vlastních čísel **L** a matice singulárních čísel **S** jsou propojeny rovnicí *(2.21)*

$$L = \frac{S^2}{n-1} \tag{2.21}$$

Hlavní komponenty **Z** jsou dány

$$Z = XV = USV^T V = US \tag{2.22}$$

Zpět do původních souřadnic se dostaneme inverzní metodou PCA. Dle rovnice *(2.14)* je pak inverze daná následující rovnicí.

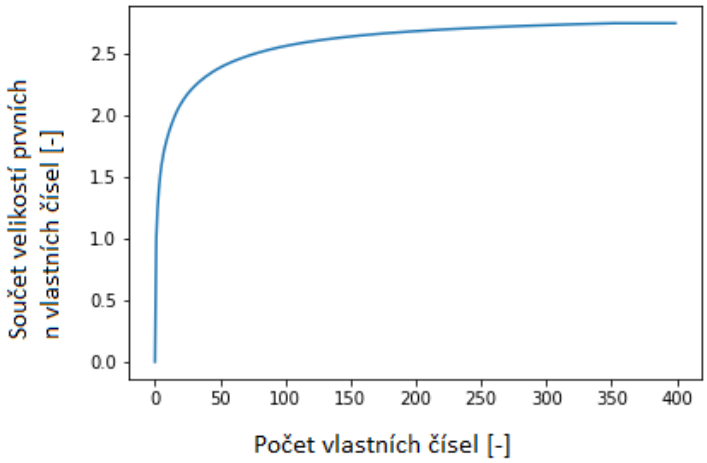
$$X = Z * V^T \tag{2.23}$$

Pokud necháme obě matice beze změny, dostaneme opět původní data. Vynulováním některých sloupců matice **Z**, tzn. odstraněním příslušných hlavních komponent, dostaneme žádané zjednodušení. Stejně je potřeba redukovat i transformační matici $\mathbf{V^T}$ tak, aby byla dodržena pravidla pro násobení matic. Ve statistice je obvykle snaha zanechat maximálně 2 nebo 3 nejhlavnější komponenty, aby bylo snadné zobrazit výstupní hodnoty ve 2D nebo 3D grafu. V obrazové analýze je tento počet zpravidla výrazně vyšší, neboť množství proměnných bývá vysoké. S výhodou se dá též využít opačný postup, tedy odstranit nejvýznamnější komponenty a nevýznamné ponechat. Právě tímto způsobem lze zvýraznit malé chyby obrazu (šum) a odlišit je od opakujících se strukturních prvků obrazu.

K zobrazení výsledků metody hlavních komponent se využívá několik různých grafů. Patří mezi ně

Cattelův indexový graf úpatí vlastních čísel (Screeplot),
rozptylový diagram komponentních skóre (Scatterplot),
dvojný graf (Biplot).

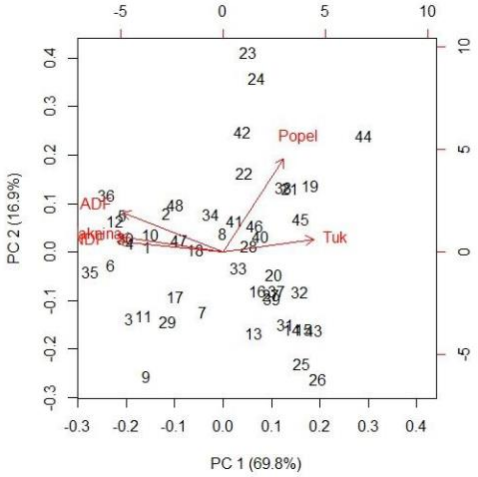
Cattelův indexový graf úpatí vlastních čísel (screeplot) zobrazuje velikosti vlastních čísel náležících jednotlivým vlastním vektorům. Vyskytuje se ve dvou formách. První forma postupně vykresluje pouze vlastní čísla. Graf je tedy klesající a ustálí se na hodnotě 0. Druhá, častěji používaná forma zobrazuje kumulaci vlastních čísel. K-tý prvek grafu se tak zobrazí jako součet prvních K vlastních čísel. Příklad je na Obrázek 13. Z obou grafů pak lze vyčíst významnost daných komponent. Hranici mezi významností a nevýznamností pak určuje zlomový bod, tzv. úpatí.



Obrázek 13 Screeplot

Rozptylový diagram komponentních skóre (Scatterplot) se používá pro zobrazení hodnot v nových komponentních souřadnicích. Většinou se využijí právě první dvě hlavní komponenty jako osy grafu. Scatterplot tak znázorňuje vztahy mezi jednotlivými objekty. Pozorování blízko sebe jsou si navzájem velmi podobná, naopak vzdálená se výrazně odlišují. Tento graf se používá pro identifikaci odlehlých pozorování, nebo k vysvětlení podobnosti objektů.

Dvojný graf (Biplot) je nástroj mnoharozměrné statistické analýzy, který stejně jako scatterplot využívá hodnoty prvních dvou komponent, navíc však zobrazuje jejich vztah k původním znakům. Jednotlivá pozorování jsou zobrazena pomocí bodů, původní proměnné jsou pak znázorněny šipkami vycházejícími ze středu grafu. Úhel mezi šipkami je pak nepřímo úměrný korelaci mezi jednotlivými proměnnými (skrze hodnotu kosinus). Malý úhel znamená vysokou vzájemnou korelaci, a tedy velkou závislost proměnných (hodnota kosinu se blíží 1). Úhel 90° značí nulovou korelaci, tedy příslušné proměnné jsou nezávislé. Úhel 180° pak značí korelaci opačnou, tzn. zvětšení jedné veličiny znamená zmenšení veličiny druhé. Koncový bod šipky navíc ukazuje, jakou měrou se podílí na tvorbě příslušných hlavních komponent.



Obrázek 14 Biplot [20]

Při použití PCA v obrazové analýze se setkáváme se dvěma hlavními přístupy. Pokud máme pouze jeden obraz, nabízí se pouze první varianta, a to určit řádky matice jako jednotlivé vzorky a sloupce jako jejich proměnné (dimenze). Sloupce a řádky jde samozřejmě zaměnit. S druhým přístupem se můžeme setkat, když máme sadu podobných (např. tváře stejného člověka v různých podmínkách) obrazů. V tomto případě řádky představují jednotlivé obrazy (je jich stejně jako počet obrázků). Matice jednotlivých obrazů transformujeme ve vektor, kdy jednotlivé sloupce matice postupně zapisujeme do jediného dlouhého sloupce. Tento vektor pak představuje dimenzionalitu, tedy řádky matice X. Máme-li např. 30 obrázků o rozlišení 400x2000 px, bude matice X mít rozměr (30x800000). Nevýhodou této metody je, že nebere v potaz hodnoty po řádcích sousedících pixelů.

2.6 Programovací jazyk Python

Python je vysokoúrovňový skriptovací jazyk navržený v roce 1991 nizozemským programátorem Guidem van Rossumem [21], [22]. V červnu 2018 van Rossum po 30 letech odstoupil z pozice vedoucího programátora. V současné době se jazyk těší velké popularitě díky své dobré čitelnosti. To je možná právě proto, že byl vyvinut z jazyka ABC, který vznikl pro výuku programování. Právě proto je python často využíván při výuce začínajících programátorů. Zároveň je však vhodný pro využití v praxi. Tomu napomáhá především čistota a jednoduchost syntaxe. K definici bloků se na rozdíl od ostatních jazyků využívá pouze odsazování.

Python nabízí instalační balíčky pro většinu běžných platforem (MS Windows, Linux, macOS, Unix, Android). Ve většině distribucí Linuxu je standardně instalován. Zdrojový kód z jedné platformy není třeba pro jinou upravovat. Je dostupný zcela zdarma (opensource), což je oproti ostatním vhodným jazykům obrovská výhoda. Python podporuje více paradigmat, jako jsou objektově orientovaná, procedurální a funkcionální programování. Díky tomu je označován jako paradigmatický nebo hybridní programovací jazyk.

Python se vyznačuje především produktivitou při psaní kódu. U jednoduchých programů je zapříčeněna především stručností zápisu, u složitějších pak široká škála knihoven nebo prostředky pro psaní testů (unit testing). Výkonově se python řadí mezi pomalejší jazyky. Proto se často setkáváme s knihovnami psanými v jazyce C a následně implementovanými do pythonu. Navíc výkonnost počítačů každým rokem roste, proto nižší rychlost pythonu není velkým problémem. V praxi se setkáme s implementací i na pomalých zařízeních, jako je například Raspberry Pi. Pro velmi výkonově náročné programy se využívají alternativní přístupy, jako PyPy, Cython a další. PyPy je alternativní interpret jazyka. Cython je transkompilátor. Ten převádí kód pythonu do výkonově rychlejšího jazyka C. Programátorovi tak zůstane pohodlí jazyka python, a navíc získá i rychlost jazyka C.

Python je již 10 let (od roku 2008) k dispozici ve dvou vzájemně nekompatibilních verzích. Plynulý přechod z verze 2 na verzi 3 se nezdařil především z důvodů nekompatibility knihoven k nové verzi. Současně proto pokračuje vývoj obou těchto verzí. Tomu by však měl být dle [23] konec do roku 2020. Verze 2.8 nebude nikdy vydána, navíc v roce 2020 python ukončí veškerou podporu, nepůjdou hlásit chyby a nebudou vydány žádné záplaty (patche).

Při vypracování této práce bylo využito několika knihoven. Jejich výčet a zároveň ukázka importu do programu je na Obrázek 15. Použité zkratky (numpy zkráceno na np) jsou programátorské zvyklosti sloužící ke zrychlení psaní a přehlednosti kódu. Importovat lze jak celý balíček knihovny, tak pouze jeho části nebo jednotlivé funkce, jak je vidět na třetím řádku obrázku.

```
 8 import numpy as np
 9 import matplotlib.pyplot as plt
10 from sklearn.decomposition import PCA
11 from numpy import linalg as la
12 import cv2
13 import scipy.signal
14 from mpl_toolkits.mplot3d import Axes3D
```

Obrázek 15 Import knihoven do programu

2.6.1 Knihovny

Numpy je základní knihovna pro vědecké výpočty [24]. Mimo jiné obsahuje funkce pro N-dimenzionální výpočty jako jsou maticové operace a práce s poli, lineární algebru, Fourierovu transformaci a další.

SciPy obsahuje základní algoritmy pro interpolaci, integrální počet, různé optimalizace, statistiku, zpracování signálu a další [25]. Množství funkcí je obsaženo v obou knihovnách SciPy i NumPy, jejich zápis a požadované parametry se však mnohdy liší. Závisí pak pouze na programátorovi, kterou variantu zvolí.

Matplotlib je knihovna pro tvorbu grafů. Spolu s výše uvedenými knihovnami zajišťují podobnou funkcionalitu, jako nabízí prostředí matlab. Na rozdíl od něj jsou však zcela zdarma.

Sklearn neboli scikit-learn je knihovna pro strojové učení a analýzu dat [26]. Je nadstavbou knihoven NumPy a SciPy. Mimo jiné obsahuje funkce pro metodu hlavních komponent (PCA).

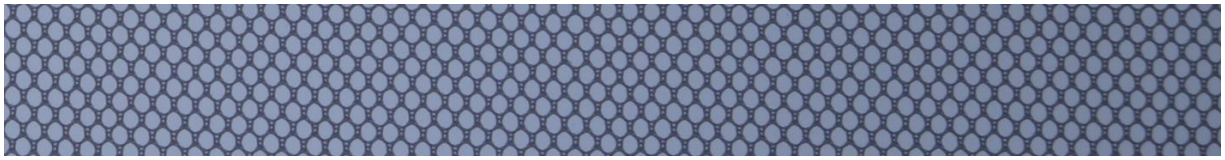
OpenCV je knihovnou navrženou speciálně pro zpracování obrazu v reálném čase [27]. Jako ostatní knihovny je též poskytována zdarma. Obsahuje více než 2500 optimalizovaných algoritmů, které mají základ v jazycích C/C++. Mezi známými firmami OpenCV používá například Google, Microsoft, Intel, Sony, Toyota nebo Honda. Algoritmy z této knihovny jsou využívány při napojování obrazů streetview, sledování důlních zařízení v Číně, pomáhají robotům při navigaci a vyzvedávání předmětů Willow Garage, detekci tonoucích v plaveckých bazénech, kontrolu čistoty přistávacích ranvejí v Turecku nebo rozpoznávání obličeje.

3 Praktická část

3.1 Získání dat

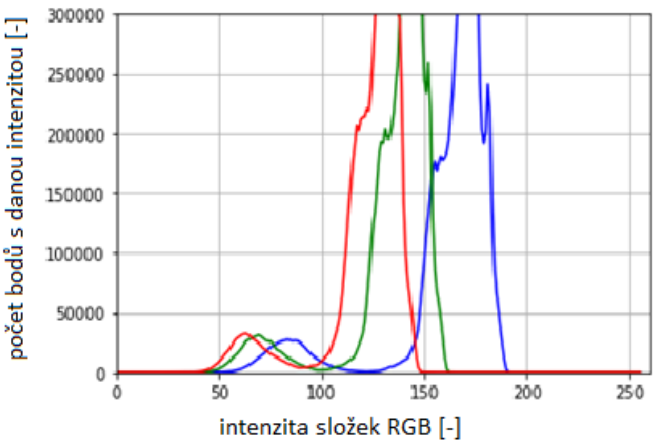
Prvním úkolem bylo provést sérii pokusů pro získání vzorků. Vláknící proces jsem několikrát zopakoval, a to s vhodně i nevhodně zvolenými parametry. Následně jsem vzorky zkoumal na optickém i elektronovém mikroskopu. Některé z nich jsou na Obrázek 1, Obrázek 4 aObrázek 5. Názorně jsem tak mohl vidět příčinu vzniku výše popsaných vad.

Vzhledem k časové náročnosti pokusů a vytíženosti přístrojů jsem poté dostal soubor hotových vzorků, které se staly mým vstupním materiálem. Tyto vzorky byly z různých důvodů vyřazeny jako vadné. Rozdílem oproti předešlým pokusům byl vláknící substrát. Při mých pokusech jsem používal průhlednou fólii. Bylo proto snadné rozpoznat homogenitu materiálu. Tyto vzorky však byly vlákněny na tkaninu jako na Obrázek 16. Dalším rozdílem bylo, že vzorky byly již oříznuté do požadovaného tvaru.



Obrázek 16 Samostatný substrát vzorku

K pořízení vzájemně porovnatelných dat bylo použito speciálního černého boxu, zamezujícího ovlivnit fotografii okolními světelnými podmínkami. Vzorek byl položen na zdroj konstantního světla (nejistota 2%) a následně pořízena fotografie. Místa s větší tloušťkou materiálu propouští méně světla, ve výsledné fotografii tedy náleží tmavším místům. Bylo použito několik nastavení fotoaparátu (ISO, čas, clona). Následně bylo zkoumáno rozložení jednotlivých histogramů. V těchto histogramech byly vždy dva oddělené vrcholy, jeden pro vzorek a druhý pro pozadí (pravítko v Obrázek 19 je jen pro představu velikosti vzorku). Jako nejvhodnější nastavení pak bylo zvoleno to, kde část histogramu představující vzorek byla nejširší.



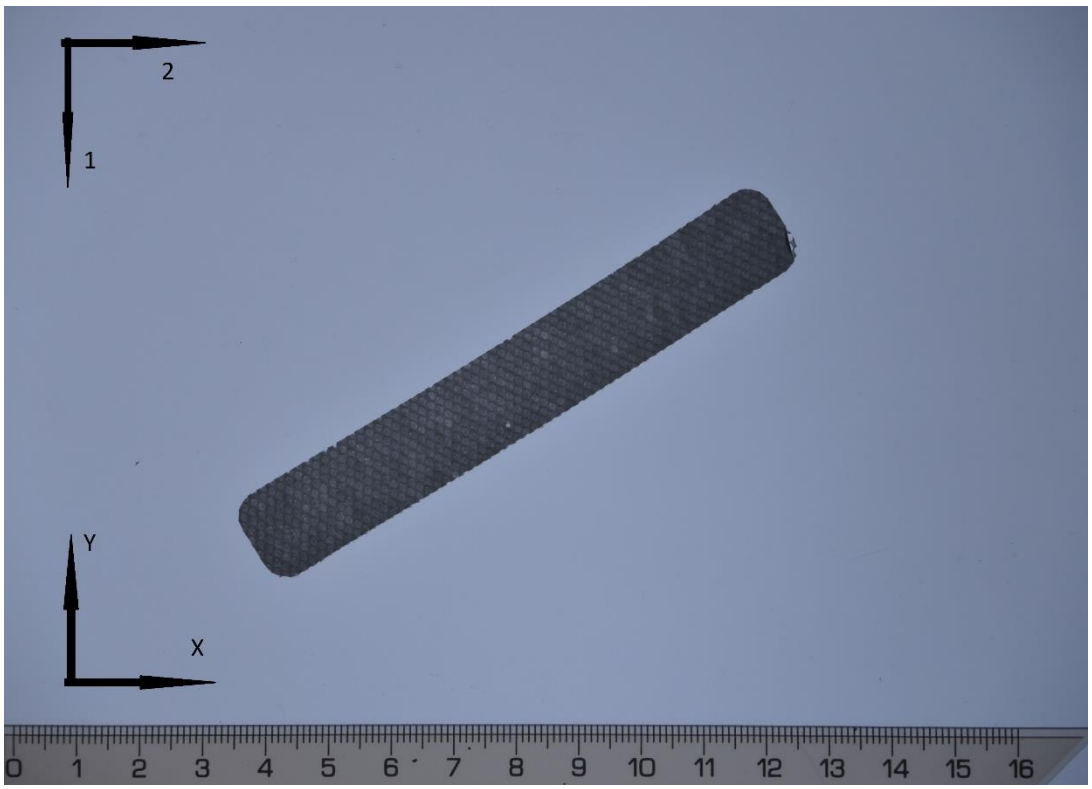
Obrázek 17 Histogram vzorku ve složkách RGB

Histogram získáme v jazykovém prostředí python. Zapotřebí je tedy implementovat fotografii do programu. K tomu využijeme knihovny OpenCV a jejích algoritmů dle Obrázek 18. Vzhledem k nevýznamnosti barev v obraze (hodnotíme pouze intenzitu procházejícího světla vzorkem) postačí zpracovávat fotografii pouze v odstínech šedi. Převod do odstínů šedi průměruje barvy všech tři spekter. Rozptyl jednotlivých spekter se však liší. Ještě lepšího rozlišení, než v odstínech šedi proto dosáhneme použitím pouze modrého spektra obrazu. Jak je patrno z histogramu na Obrázek 17, právě spektrum modré barvy má největší rozsah hodnot. Rozměry matice představující fotografii pak jsou 4000x3000 namísto 4000x3000x3 původního obrazu.

```
591 vstup = cv2.imread('1.jpg')
592
593 def histogram_RGB(image):
594     color = ('b','g','r')
595     for i,col in enumerate(color):
596         hist = cv2.calcHist([np.uint8(image)],[i],None,[256],[0,256])
597         plt.axis([0,260,0,300000]);
598         plt.plot(hist,color = col)
599     plt.grid(True)
600     plt.show()
```

Obrázek 18 Načtení obrázku a vykreslení histogramu všech tří složek

Při práci s obrazem je velmi důležité správné určení souřadného systému. Přirozeně jsou využívány standardní **souřadnice X,Y**, používané v matematice a fyzice. Jejich počátek je v levém dolním rohu, první souřadnice X kladně roste vodorovně směrem doprava, souřadnice Y kladně roste ve svislém směru vzhůru. V kontrastu s tímto zápisem je **maticový zápis**. Zde je počátek souřadnic v levém horním rohu. První souřadnice roste svisle směrem dolů, druhá roste vodorovně směrem vpravo. Neuvážené použití obou souřadnic může vést ke špatnému pochopení a použití metod detekce. Metody založené na výpočtech poloh a vzdáleností bodů jsou na takové chyby náchylné nejvíce. Je zde totiž největší množství ručně psaných výpočtů.



Obrázek 19 Příklad pořízené fotografie s naznačením souřadných systémů

Proto je vhodné (nutné) zavést jednotné použití souřadnic.

1) Využití pouze maticových souřadnic. Jejich názvy pak volit například X1 a X2. Použití názvů x a y je zcela nevhodné (matoucí).

2) Transformace dat z matice do souřadného systému X,Y. Při každém vykreslení dílčích výsledků je pak nutná zpětná transformace. Ta může být zahrnuta přímo do vykreslovací funkce, což zpřehlední celý zápis. Matoucí pak může být akorát zkoumání hodnot přímo v maticích. Nevýhodou této metody je delší čas výpočtu způsobený těmito transformacemi.

3) Využití maticových souřadnic, které jsou pojmenovány Y,X. Touto změnou pořadí se práce s daty výrazně ulehčí. Poloha bodů v obraze odpovídá poloze bodů v matici, kladný směr souřadnice X udává vodorovný směr z leva do prava. Směr a počátek souřadnice Y je opačný oproti standardu, tedy shora dolů. Práce s tímto novým souřadným systémem je velmi podobná standardnímu použití souřadnic X,Y. Oproti přístupu 2 pak není třeba žádných matematických úprav. V programu bylo využíváno právě tohoto přístupu.

3.2 Separace vzorku

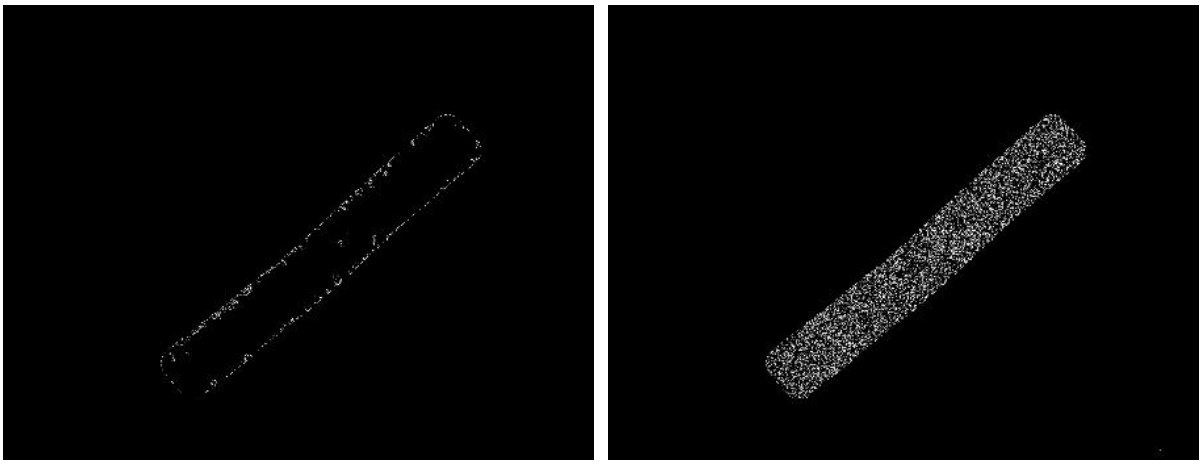
Dalším úkolem je pomocí jazyka python detekovat vzorek a oddělit ho od pozadí. K tomu se z výše uvedených metod nabízí detekce hran a prahování.

Snahou je v obraze najít obdélník, který co nejpřesněji bude kopírovat okraje vzorku. Pro testování jsem nejprve použil Cannyho metodu detekce hran. Z detekčních metod patří k nejlepším, navíc je funkce již předprogramovaná v knihovně OpenCV.

```
626
627 Canny = cv2.Canny(vstup, výstup, dolní hranice, horní hranice, velikost jádra)
```

Obrázek 20 Cannyho detekční funkce

Výrazná struktura uvnitř vzorku bránila ideálnímu naladění. Metoda totiž detekovala rychle se měnící jas způsobený substrátem. Při větším potlačení prahové hranice naopak mizely i hledané okraje. Jistý kompromis mezi těmito dvěma extrémy najít šlo. Po otestováním dalších vzorků, kdy se vyhovující hodnoty měnily, se tento přístup ukázal jako nevhodný.



Obrázek 21 a) Cannyho hranový detektor s koeficientem derivace 300 b) Canny s koeficientem 50

Po předešlém nezdaru jsem nastavil prahovou hodnotu na podstatně nižší hodnotu, abych detekoval co nejvíce hran ve vzorku. Monotónní pozadí by dle očekávání nemělo obsahovat hrany. Nicméně v některých fotografiích bylo nalezeno několik pixelů (odpovídající nalezeným hranám) i mimo vzorek. Požadavkem je spojit oblast vzorku v ucelený objekt a odstranění chybných hran v pozadí. To bylo dosaženo pomocí dilatace s jádrem o velikosti 30x30 a následné eroze s jádrem 40x40. V pythonu tento proces zapíšeme následovně:

```
67    dilatace = cv2.dilate(prahovani, cv2.getStructuringElement(cv2.MORPH_RECT, (10, 10)))
68    eroze = cv2.erode(dilatace, cv2.getStructuringElement(cv2.MORPH_RECT, (40, 40)))
```

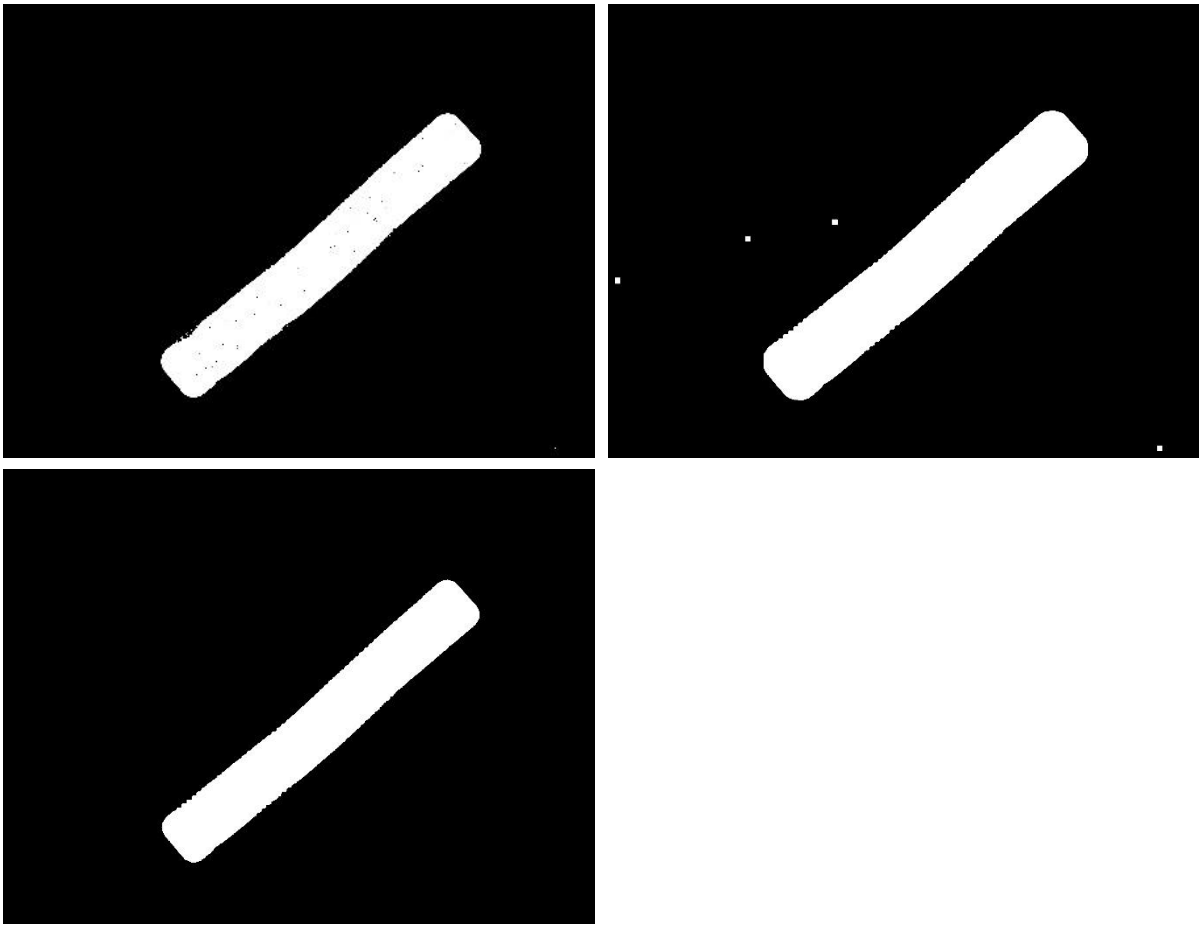
Obrázek 22 Dilatace a eroze

Ve stejné knihovně nalezneme i funkci pro otevření a uzavření viz kapitola 2.3.9. Bohužel však nelze nastavit různé velikosti jader pro každou dílčí část. Pro spolehlivý výsledek by bylo třeba použít metodu uzavření a následně i otevření. Tento postup je oproti kombinaci dilatace – eroze výpočtově dvakrát náročnější a v tomto konkrétním případě neznamená žádné výhody (při jiných aplikacích se tyto postupy využívají i několikanásobně). Navíc algoritmy pro dilataci a erozi v pythonu nemění velikost obrazu, což je sice v rozporu s očekávanou funkcí metody, avšak je to pro toto použití spíše výhodou. Algoritmus zřejmě pracuje jinak na okrajích obrazu.

Vzhledem k tomu, že by se vzorek neměl na okraji nikdy nacházet, nemůže být výsledek nijak ovlivněn.

Dostáváme tedy „binární obraz“ (datový soubor je stále uint8, který ale obsahuje hodnoty pouze 0 a 255) jednoho celistvého bílého obrazce na neporušeném černém pozadí.

Stejného výsledku se pokusím dosáhnout i metodou prahování, následně dilatací a erozí, jako v předchozím případě. Prahování je oproti detekování hran výrazně jednodušší, a proto i rychlejší. Prahovou hodnotu určím na základě histogramu. Je to hodnota ležící mezi vrcholy náležící zastoupení jasové hodnotě vzorku a pozadí. Tato hodnota byla zvolena 180. Jak je vidět na Obrázek 21, takto upravená fotografie obsahuje v oblasti vzorku daleko více bílých pixelů než při předešlé metodě. Pozadí přitom stále zůstává téměř bez chybových bílých bodů. Navíc stačí použít výrazně menší jádra dilatace a eroze (10 a 15), což urychlí výpočet.



Obrázek 23 a) prahování s prahovou hodnotou 240 (z histogramu vyčtená hodnota) b) dilatace – z obrázku je patrné vyplnění černých teček ve vzorku, ale vznik bílých čtverců mimo vzorek c) eroze – odstranění šumu mimo vzorek

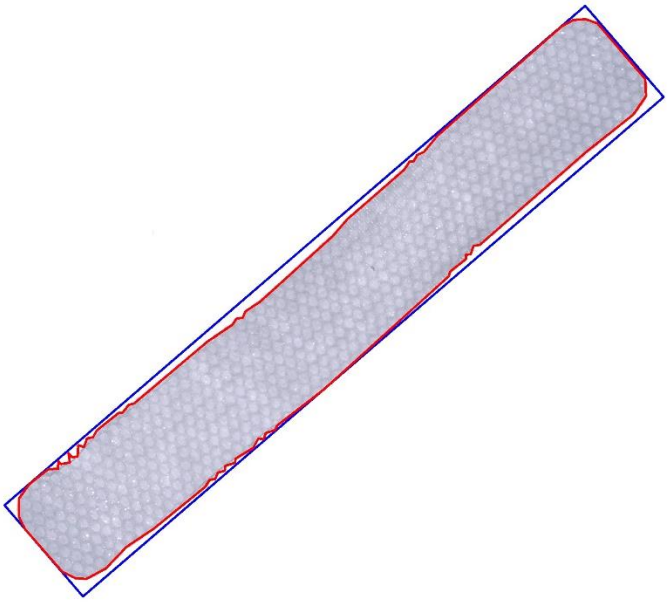
Funkce findContours najde obrys (konturu) obrazu. Konturu jednoduše definujeme jako nepřerušenou křivku spojující body stejné barvy, které jsou na hranici objektu. Tato metoda proto téměř vždy nachází využití pouze po předešlých úpravách, nejlépe na binární obrazy. Na Obrázek 25 je tato kontura vykreslená červenou barvou do původní fotografie pro porovnání funkčnosti algoritmu.

```
70    hranice, contours, hierarchy = cv2.findContours( dilatace, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
71    cv2.drawContours(img,[contours[0]],-1,(0,0,255),5)
72    for cont in contours:
73        rect = cv2.minAreaRect(cont)
74    sizeR = np.int0((rect[(1)]))
75    centerR = np.int0((rect[(0)]))
```

Obrázek 24 Kód pro nalezení obdélníka

Proměnná contours tedy obsahuje souřadnice všech hraničních bodů. Pro zobrazení vzorku v novém obrázku s co nejmenším pozadím potřebujeme bodům z proměnné contours opsat obdélník. Obdélník s nejmenším obsahem pak bude nejlepším řešením. V pythonu toho dosáhneme pomocí funkce minAreaRect a for smyčky. Výstupem funkce je proměnná typu tuple, obsahující souřadnice středu obdélníku, velikost obdélníku a úhel natočení ve stupních.

Tento obdélník je na obrázku vykreslen modrou barvou. Ze známé polohy, velikosti a natočení vypočítáme rohové body obdélníku, které pak funkcí drawContours spojíme a vykreslíme v podobě obdélníka.



Obrázek 25 Červeně vykreslená kontura vzorku v původní fotografii. Modře opsán čtverec s nejmenším obsahem.

Funkce getRotationMatrix2D z těchto informací vytvoří transformační matici. Tuto matici nazývám transformační (nikoliv rotační, jak je v popisku funkce) z jednoho důvodu. Tato matice neobsahuje jen rotační část. Je v ní zahrnuto i posunutí obrazu, které umístí střed obdélníku do středu nového natočeného obrazu. Matice je zobrazena v rovnici *(3.1)*, kde α je úhel natočení a k měřítko transformace.

$$M = k \cdot \begin{bmatrix} cos(\alpha) & sin(\alpha) & (1 - cos(\alpha)) \cdot center(x) - sin(\alpha) \cdot center(y) \\ -sin(\alpha) & cos(\alpha) & sin(\alpha) \cdot center(x) + (1 - cos(\alpha)) \cdot center(y) \end{bmatrix} \quad (3.1)$$

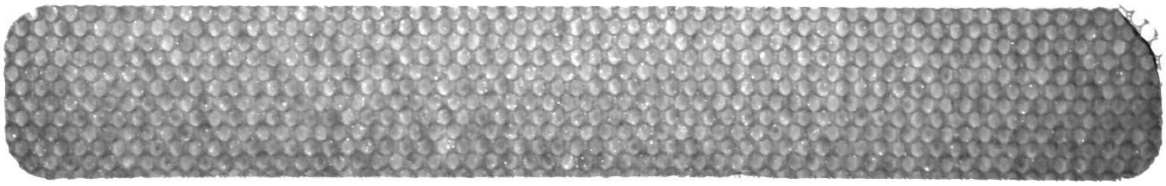
Přidruženou funkcí je warpAffine, která provede výpočet dle *(3.2)*.

$$výstup = vstup\ (M_{11}x + M_{12}y + M_{13}\ , M_{21}x + M_{22}y + M_{23}) \quad (3.2)$$

Následným oříznutím se zbavíme okolí. Ideální by bylo obraz oříznout podle vypočítaného obdélníku. Jeho velikost se však pro každý obraz mírně liší. To by způsobilo obtížné porovnávání a špatnou kalibraci chybovosti. Proto je nutné tyto hodnoty v úvodu programu ručně zadat. Při volbě tohoto parametru je třeba dbát na skutečnou velikost vzorku. Proto se tyto skutečné vypočítané hodnoty vždy v průběhu výpočtu vypisují do dialogového okna. V případě špatné volby se pak objeví chybová hláška.

Posledním krokem předzpracování je ekvalizace. Ta je provedena rovnicí *(2.1)*. Opět je však třeba hodnoty maximálního a minimálního jasu volit manuálně. Vzhledem k dobré odlišitelnosti vzorku a pozadí v histogramu si můžeme dovolit výraznou ekvalizaci vzorku. Hodnoty pozadí přesahující mezní hodnotu intenzity 255 pak jednoduše přepíšeme právě na prahovou hodnotu 255. Roztažením hodnot intenzity pixelů do celého spektra získáme především zvýraznění výstupní fotografie. Následující postupy na rozptylu hodnot nezáleží, jelikož pracují se soubory typu float64 a complex64, tedy s desetinnými, zápornými, i komplexními čísly. Pro zpětnou transformaci do typu uint8 je však větší rozptyl výhodou.

Příkladem výstupu po předzpracování je například Obrázek 26.



Obrázek 26 Výstup po předzpracování

3.3 Filtrace struktury

Nejproblematičtější z vad určených k detekci jsou kapičky nerozpuštěného roztoku ve vzorcích. Kvůli substrátu totiž nelze uplatnit hranové detektory, prahování, ani další klasické metody. Hranové detektory by označily strukturu substrátu, kde je gradient přechodu největší. Kapičky by částečně detekovány byly, ale odlišení od hran struktury by bylo nemožné. Prahování by přineslo o něco lepší výsledky. Stále by však docházelo k častým chybám, a to především ve dvou případech. Buď nedetekování kapičky, která leží na struktuře, nebo detekování míst uvnitř struktury se slabší, ale přesto dostatečnou vrstvou. Proto jsou dále popsány postupy sloužící k odstranění struktury.

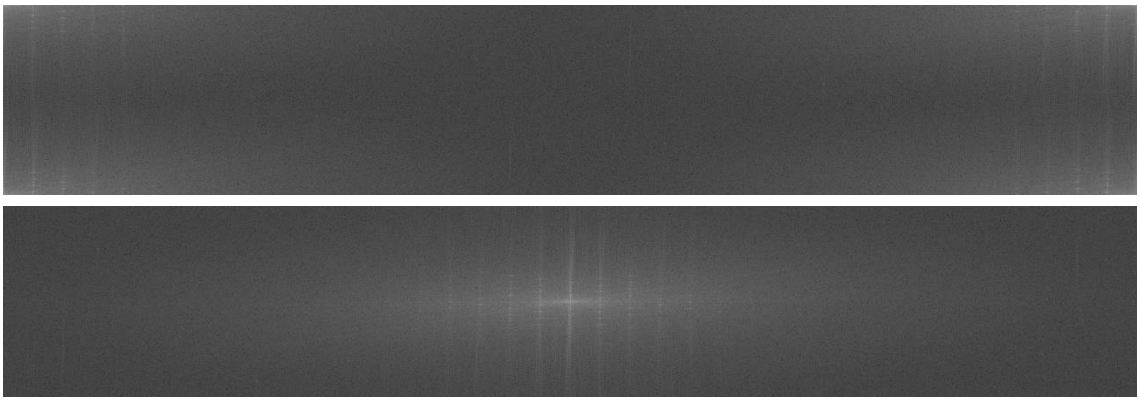
3.3.1 Fourierova transformace

Fourierova transformace je v pythonu zahrnuta v několika knihovnách. Jsou to knihovny numpy, scipy, openCV a další. V programu bylo využito funkce z knihovny openCV. Oproti ostatním totiž zapisuje reálnou a imaginární část komplexního čísla jako dvě různá čísla typu float32 do jedné proměnné. Ta má pak rozměr [x, y, 2]. Další úpravy a výpočty jsou díky tomu jednodušší a přehlednější.

```
258 def fourier2(image):
259     dft = cv2.dft(np.float32(image),flags = cv2.DFT_COMPLEX_OUTPUT)
260     dft_shift = np.fft.fftshift(dft)
261     dft_shift1 = dft_shift.copy()
262 
263     magnitude_spectrum1 = 10*np.log(1+cv2.magnitude(dft_shift1[:,:,0],dft_shift1[:,:,1]))
264     show(magnitude_spectrum1,'Fourierovo spektrum s filtrem1')
265 
266     for (x,y,z), value in np.ndenumerate(dft_shift):
267         if np.abs(dft_shift[x,y,z])>10000000/(1+(18*(x-(size_cut[1])/2)**2 + (y-(size_cut[0])/2)**2)**0.5):
268             if x == (size_cut[1])/2 and y == (size_cut[0])/2:
269                 dft_shift1[x,y,z] = dft_shift[x,y,z]
270             else:
271                 dft_shift1[x,y,z] = 0
272 
273     f_ishift1 = np.fft.ifftshift(dft_shift1)
274     img_back1 = cv2.idft(f_ishift1, flags=cv2.DFT_SCALE | cv2.DFT_REAL_OUTPUT)
275     img_back1 = boundaries(img_back1)
276     show(img_back1,'Filtrovany obraz3')
277     return img_back1
```

Obrázek 27 Kód Fourierovy transformace

Pro lepší analýzu je vhodné výstup z Fourierovy funkce posunout tak, aby bod nulové frekvence ležel uprostřed obrázku, nikoliv v bodě [0, 0]. K tomu slouží funkce np.fft.fftshift(). Výpočtem výkonového spektra dostaneme Obrázek 28. Pro zobrazitelné hodnoty v rozsahu 0 až 255 bylo použito funkce logaritmus. Světlá místa náleží frekvencím s velkou amplitudou, jsou tedy v původním obraze významné.

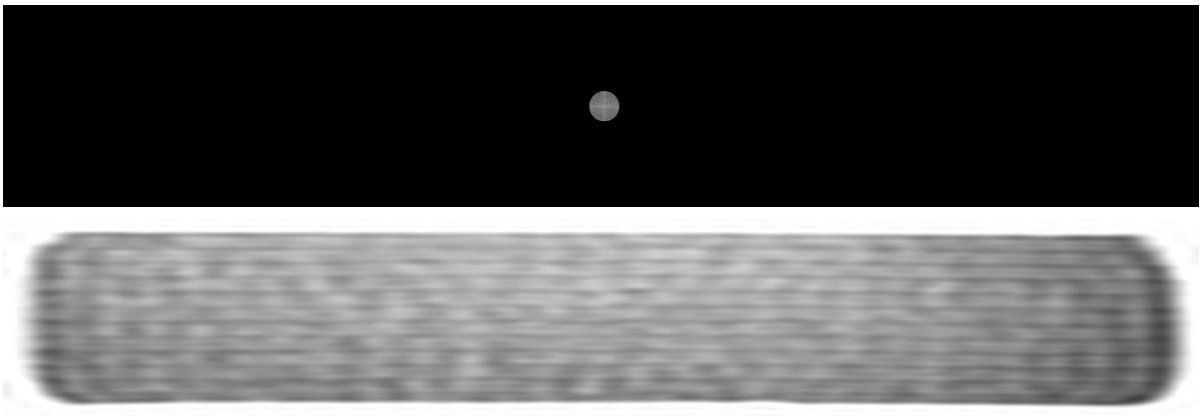


Obrázek 28 Posunutí Fourierova spektra a) původní b) posunuté

Ve Fourierově obraze použitých vzorků je patrná jistá struktura odpovídající struktuře substrátu. Cílem je za pomoci vhodných filtrů odstranit strukturu z původního obrazu, zároveň ale nepoškodit ostatní informace v obrazu.

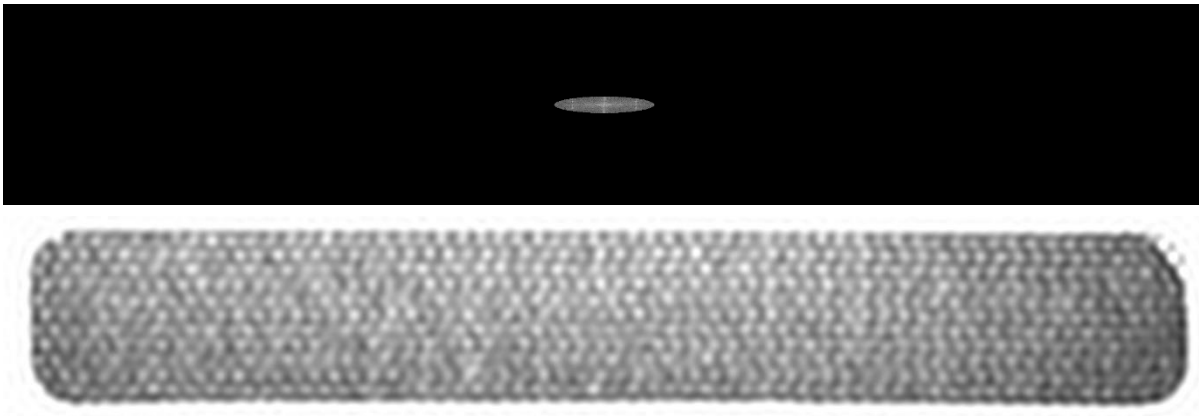
První série pokusů byla provedena s low-pass a high-pass filtrem. Princip těchto filtrů spočívá v potlačení buď středu (high-pass) nebo okolí (low-pass) spektra. Obvykle je pro tento filtr volen kruh. Vzorky však jsou výrazně nesymetrické a při použití kruhového filtru vykazovaly

velké rozdíly filtrace ve směrech x a y. Vysvětlení lze odvodit z analogie použití Fourierovy transformace na časově proměnné signály. Periodě zde přiřadíme vzdálenost. Frekvence je pak analogicky 1/vzdálenost. Při výpočtu diskrétní Fourierovy transformace pak funkci dělíme periodou vzorkování, tedy rozměry obrazu. Ve směru x tedy výsledek dělíme hodnotou 2400 a ve směru y pouze hodnotou 400, která je 6x menší. Výsledek Fourierova spektra si přesto zanechává stejné rozměry jako původní obraz. Aby toto bylo možné, je zapotřebí prohození os. Ose x v původním obrazu tedy odpovídá osa y Fourierova spektra a naopak. Aby filtr fungoval v obou osách stejně, je proto zapotřebí místo kruhu pracovat s elipsou, která má hlavní osy ve stejném poměru jako je velikost vzorku.



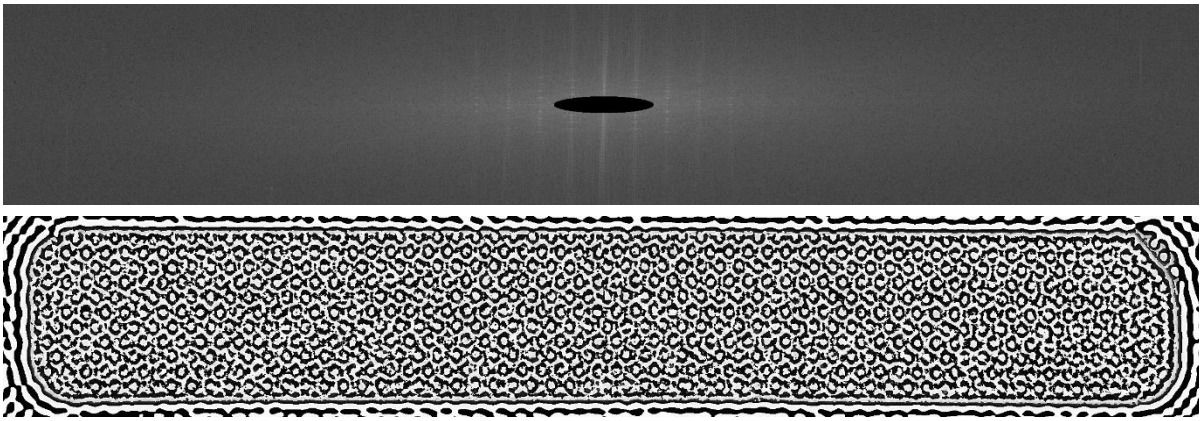
Obrázek 29 kruhový low-pass filter

Low-pass filtr propouští pouze hodnoty nízkých frekvencí. Ty se nacházejí v okolí středu spektra. Ostatní hodnoty jsou tedy přepsány na hodnotu 0. Zpětnou transformací získáme Obrázek 30. Přestože většina spektra byla odstraněna, vzorek je jasně znatelný. Obraz vzorku ztratil ostrost. Je zde dokonce zřetelně vidět základ struktury substrátu. To lze snadno objasnit. Světlá místa poblíž okrajů filtru náleží nejnižší frekvenci struktury. Stejně výrazně (bíle) jako se objevuje ve spektrálním obraze se musí objevit i po zpětné transformaci. Rozmístění struktury ve vzorky se proto musí shodovat s původním obrazem, tvar však tvoří jednoduchá sinusoida. Pro správné zobrazení by bylo zapotřebí vyšších frekvencí.



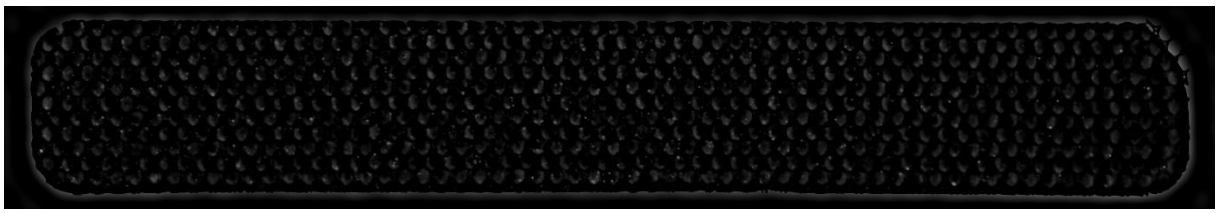
Obrázek 30 eliptický low-pass filtr se stejným poměrem hlavních os, jako poměr rozměru obrázku 6:1

Naopak při použití high-pass filtru chybí nízké frekvence. Zmizet by měl především rozdíl mezi světlostí pozadí a vzorku. Samotná hrana přechodu by však měla zůstat výrazná. Na straně vzorku je okraj tmavší než zbytek vzorku. Po přechodu mimo vzorek velmi rychle přejde na světlý odstín, pak se opět ustálí na průměrné hodnotě. Tohoto efektu se v některých aplikacích používá pro detekci hran.



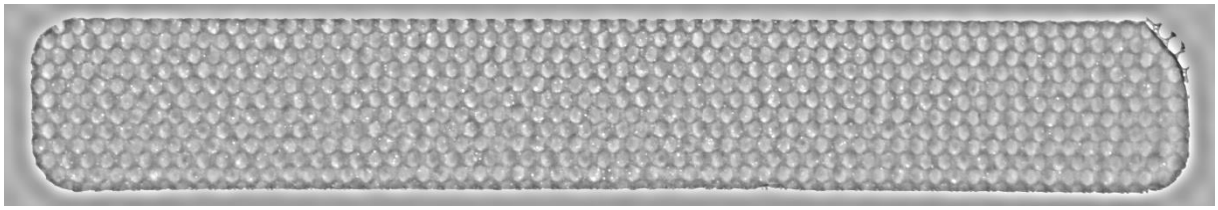
Obrázek 31 eliptický high-pass filtr

Výsledek na Obrázek 31 se však zásadně lišil od očekávání. Při číselném zobrazení matice se ukázalo, že matice obsahuje záporná čísla. Funkce cv2.imshow(), která očekává pouze hodnoty 0 až 255, tato čísla sice dokázala zpracovat, zobrazila je ovšem velmi nečekaně. Například hodnotu -1, tedy těsně za limitem černé barvy, zobrazila jako hodnotu 254. Hodnotě -5 pak připadla hodnota 250. Z nejtmavších míst se tak rázem stala místa bílá. Po zjištění příčiny problému špatného zobrazení byla vytvořena funkce zaručující zobrazení hodnot v definičním oboru obrazu. Ta záporné hodnoty přepisuje na 0, hodnoty větší než 255 přepisuje na 255. Výsledek po použití filtru je na dalším obrázku.



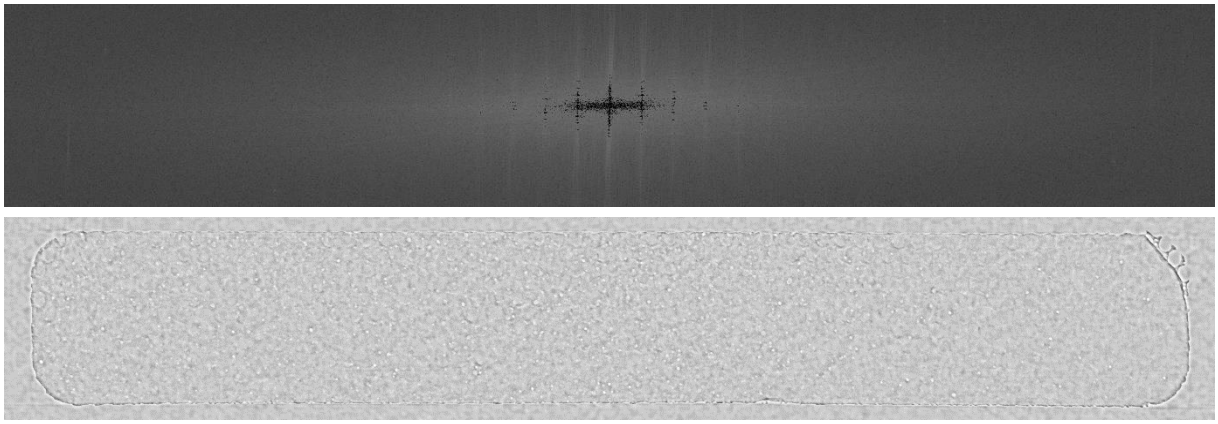
Obrázek 32 Obrázek 31 po ošetření hodnot mimo definiční obor

Získáváme výstupní obrázek, který z velké části tvoří černá barva. Světlejší místa se objevují na okrajích vzorku a v místech nepravidelností a kapiček ve vzorku. Důvodem je vynulování středového bodu (bod nulové frekvence). Ten totiž nabývá vysoké hodnoty, a proto neprojde prahovým filtrem. Tato hodnota v sobě ukrývá průměrnou hodnotu intenzity obrazu. Normalizace tak je ukryta přímo ve funkci samotné (na rozdíl od metody PCA, kdy je nutno vstup nejprve normalizovat na průměrnou hodnotu 0).



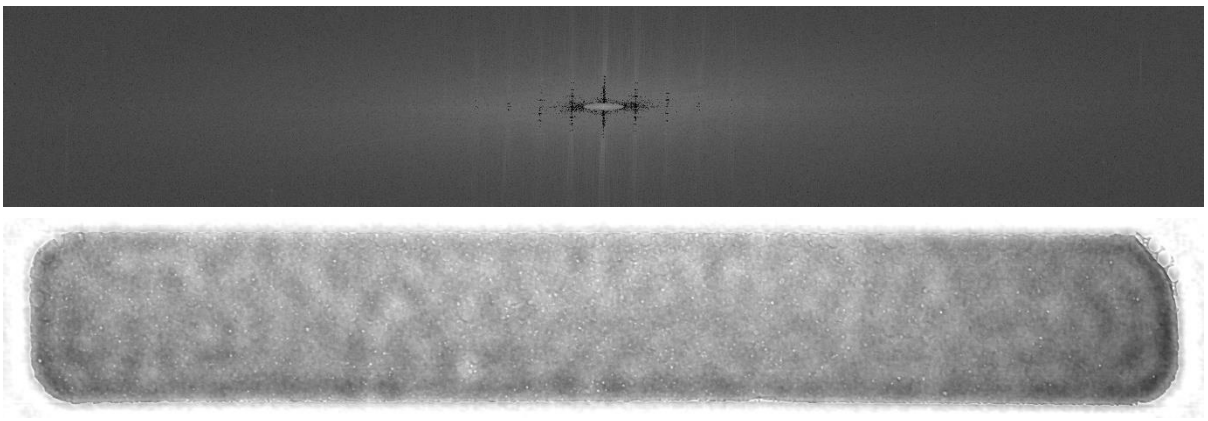
Obrázek 33 high-pass filtr s ošetřením střední hodnoty

Následuje pokus s využitím prahování. Všechny body s vyšší hodnotou, než je ta prahová, jsou nahrazeny nulou. Tím bychom se měli zbavit opakující se struktury, ostatní nepravidelné jevy by však měly zůstat zachovány. Výsledek pokusu je na Obrázek 34. Ve spektrálním obrazu můžeme vidět hlavní nedostatek této metody. Amplitudy malých frekvencí totiž mívají mnohem větší amplitudy. Z obrázku tak odstraňujeme spíše rozdílnou tloušťku vrstvy (kterou chceme detekovat), než abychom odstranili strukturu.



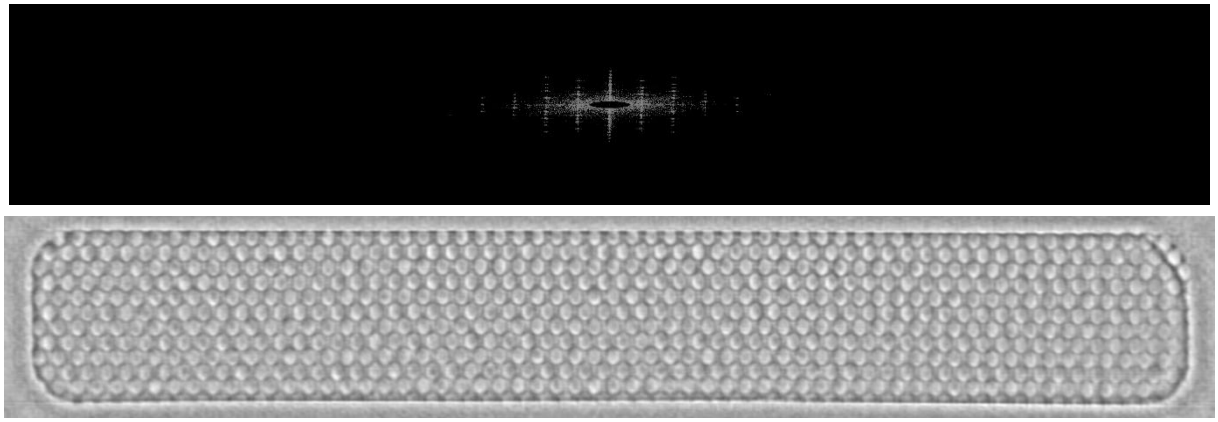
Obrázek 34 Filtrace pomocí prahování

Následující pokus vychází z kombinace výše zmíněných pokusů. Je využito prahování pro nalezení významných frekvencí. Zároveň je využit obdoby low-pass filtru, pomocí něhož zabráníme odstranění velmi nízkých frekvencí. Metoda přináší uspokojivé řešení. Ze vzorku zmizela struktura, přitom chybové kapičky jsou v obraze stále znatelné. Znatelná je i tenčí vrstva nahoře uprostřed vzorku. Tato metoda však do vzorku vnáší systematickou chybu. Na okraji low-pass filtru se nachází největší hodnoty, které vždy nemusí znamenat opakující se prvky struktury. Právě tyto vymazané frekvence pak v určité vzdálenosti od okraje ve výsledném obrázku tvoří tmavší pruh. Výpočtem lze ověřit, že právě tato vzdálenost od kraje přísluší frekvencím na okraji filtru. Pro lepší výsledky by bylo vhodné použít více pásem pro filtraci.



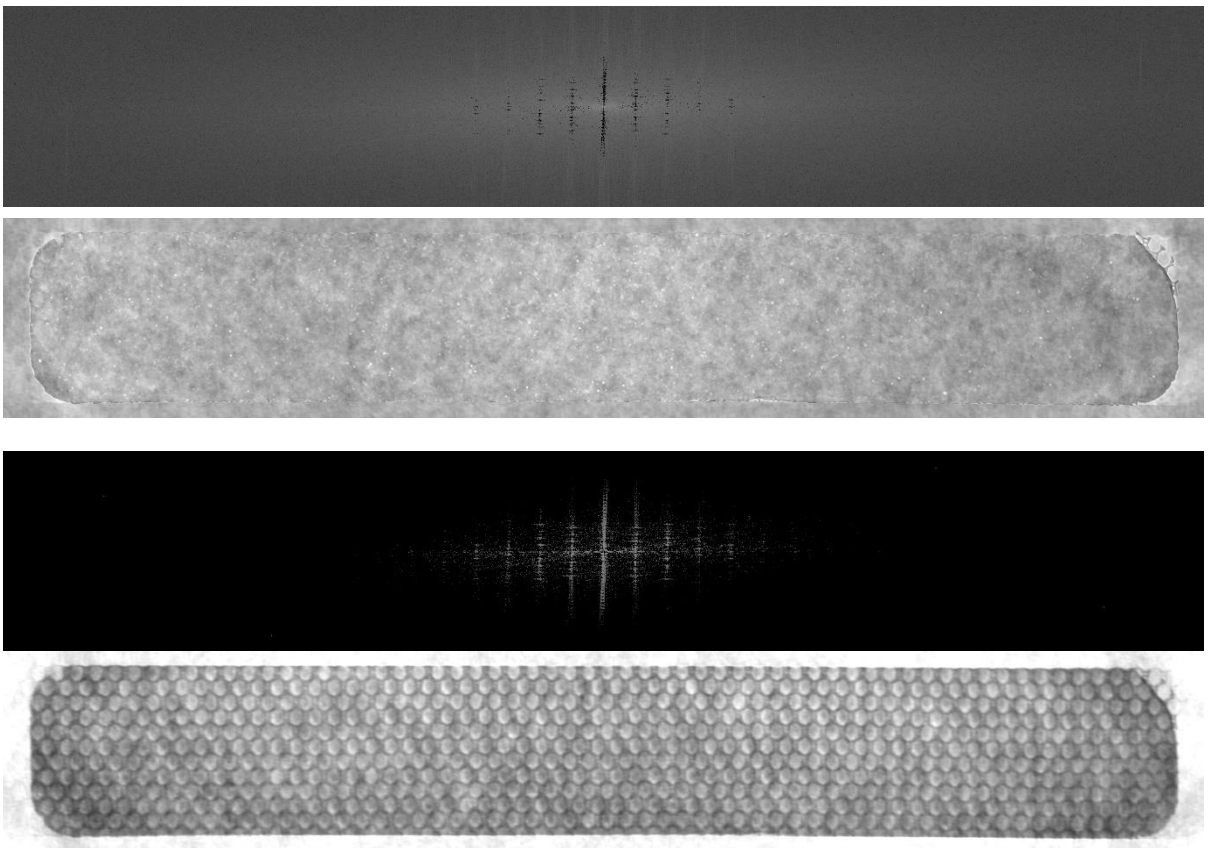
Obrázek 35 Filtrace pomocí prahování a low-pass filtru

Zajímavé výsledky poskytuje „inverzní“ postup k předešlému. Hodnoty prve ponechané odstraníme a ponecháme odstraněné. Dostáváme téměř čistou strukturu substrátu. Zcela zmizely malé chyby (kapičky), nerovnoměrnost vrstvy. Došlo k narovnání okrajů. Velká chyba okraje (pravý horní roh) zůstala zachována. Opět se zde vyskytuje sinusový přechod na hranici vzorku. Ve struktuře jsou také znatelné chyby, které mají tutéž příčinu.



Obrázek 36 Prahování + low-pass opačně

Další myšlenkou je využít adaptivního prahování. Práh nebude mít všude konstantní hodnotu, ale bude se měnit dle potřeby. Obrázek 37 ukazuje příklad použití. Prahová hodnota v každém bodě je dělena vzdáleností od středového bodu (+1, aby nedocházelo k dělení nulou). Tato metoda funguje natolik dobře, že ponecháním pouze několika frekvencí (Obrázek 37 d)) dokážeme rekonstruovat původní obraz, kde chybí pouze drobné vady. Když zobrazíme obraz bez těchto nejdůležitějších frekvencí, dostaneme Obrázek 37 b). V obraze se prolíná různá tloušťka vrstvy, která však neodpovídá tloušťce vrstvy vzorku.



Obrázek 37 a) frekvenční spektrum po odečtení nejvyšších frekvencí adaptivním prahováním b) obraz po zpětné transformaci c) ponechání nejvyšších frekvencí d) obraz po zpětné transformaci

3.3.2 Metoda hlavních komponent (PCA)

Předprogramovanou metodu hlavních komponent nalezneme například v knihovně sklearn [28]. Takto byla provedena série pokusů. Pro základní filtraci ponecháním několika hlavních komponent byla dostačující, pro další testování však obsahovala příliš mnoho neznámých výpočtových fází. Z tohoto důvodu jsem se proto rozhodl vytvořit vlastní funkce dle postupu v teoretické části, a to jak pomocí výpočtu přes kovarianční matici, tak i metodou singulárního rozkladu (SVD). PCA z knihovny sklearn pak posloužila pro ověření správnosti obou funkcí.

```
323 def pca(image):
324     pca = PCA()
325     pca.fit(image)
326     X1 = pca.transform(image)[:,pocet_komponent:] @ pca.components_[pocet_komponent:,:]
327     X1 += + np.mean(image)
328     return X1
```

Obrázek 38 PCA z knihovny sklearn

První postup je uveden na

Obrázek 39. Prvním krokem je sloupcová centralizace. Tu provedeme funkcí knihovny numpy.mean, kde jako druhý argument uvedeme axis=0. Právě ten udává, že se jedná o centralizaci po sloupcích. Kovarianční matici vypočítáme funkcí numpy.cov(). Funkce je standartně nastavená tak, že řádky reprezentují proměnné a sloupce jednotlivá měření. To je

však opačně oproti postupu popsanému výše, proto je nutno nastavit argument rowvar jako False.

Funkcí eig() z knihovny linalg nalezneme vlastní čísla a vlastní vektory. Vlastní čísla mohou být z definice komplexní čísla. Kovarianční matice **C** je však symetrická a pozitivně semidefinitní, proto jsou vlastní čísla reálná a kladná. Můžeme proto pracovat pouze s reálnou částí vlastních čísel a vlastních vektorů a uložit je do proměnné typu float místo complex.

```
336 def pca2(image):
337     X = image - np.mean(image, axis=0)
338     C = np.cov(X, rowvar=False)
339     L, V1 = la.eig(C)
340     V1 = V1.real
341     poradi = L.argsort()[::-1]
342     L, V1 = L[poradi], V1[:, poradi]
343     V = V1[0:size_cut[0], 0:size_cut[1]]
344     Z = X @ V
345     X1 = Z[:,:pocet_komponent] @ V.T[:pocet_komponent,:]
346     X1 += + np.mean(image)
347     X1 = boundaries(X1)
348 #     show(X1,'PCA2')
349     return Z,V1,L,X1,C
```

Obrázek 39 PCA

Dále je třeba vlastní vektory seřadit podle velikostí příslušných vlastních čísel. K tomu je využito funkce argsort(). Jejím výstupem je pole hodnot zobrazující pořadí seřazených vlastních čísel. Hodnota -1 pak určuje sestupné řazení. Dalším příkazem pak vlastní vektory a vlastní čísla seřadíme.

Matice **C** (p x p) obsahuje p vlastních čísel a p vlastních vektorů. Matice **V1** je proto také (p x p). Velká část však obsahuje pouze nuly, proto je vhodné tuto matici zmenšit. Největší možné zmenšení bez ztráty nenulových vlastních čísel a vlastních vektorů je právě na rozměr (p x n).

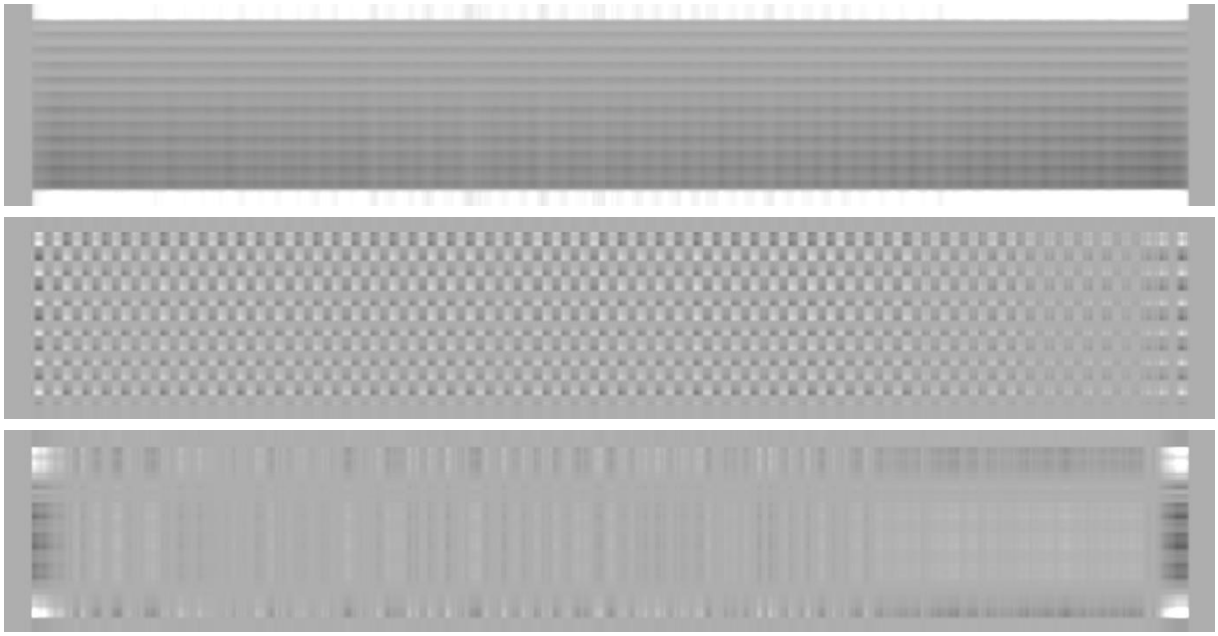
Vynásobením vstupní matice **X** a matice vlastních vektorů **V** dostaneme matici **Z**, neboli matici hlavních komponent. Ta se shoduje s výstupem předprogramované funkce. Zpětnou transformací získáme opět původní data. Právě při zpětné transformaci lze některé hlavní komponenty odstranit a tím získat potřebné údaje. Na závěr už je potřeba jenom zpět přičíst průměrnou hodnotu původního obrazu (zde už ne sloupcově centrovanou).

Druhý postup využívá k získání transformační matice singulárního rozkladu SVD. Knihovna linalg obsahuje již funkci svd. Funkce již obsahuje volitelnou redukci nulových prvků matice. Tu nastavíme parametrem full_matrices = False. Matici hlavních komponent **Z** získáme opět násobením matice **X** a **V**.

```
295 def svd(image):
296     X = image - np.mean(image, axis=0)
297     U, s, Vt = la.svd(X, full_matrices=False)
298     V = Vt.T
299     Z = X @ V
300     X1 = Z[:,:pocet_komponent] @ Vt[:pocet_komponent,:]
301     X1 += + np.mean(image)
302     X1 = boundaries(X1)
303     cv2.imwrite( "pca.jpg", X1 );
304     show(X1,'SVD1')
305     screeplot(s)
306     return X1,V,Z,X,U,s
```

Obrázek 40 SVD

K hlubšímu pochopení byly postupně vykresleny vždy pouze jednotlivé komponenty. První tři pak jsou na Obrázek 41. Dohromady totiž tvoří více jak 50% informace uložené v obrazu. V první hlavní komponentě je zachycen obdélníkový tvar vzorku. V žádné jiné komponentě se tímto způsobem již neobjeví. Tvar je v nich rozpoznatelný pouze chybějící strukturou mimo vzorek. Tvar obdélníku je dokonalý. Veškeré vady se projeví až v méně významných komponentách. To platí dokonce i pro zaoblené rohy. Zajímavé je chování levého a pravého okraje. Téměř konstantní hodnoty pozadí nejsou v souladu s převládajícím průběhem uvnitř vzorku. Proto jsou výstupem z metody PCA v těchto místech nuly. Závěrečné přičtení průměrné hodnoty pak vytvoří šedou barvu. V první komponentě je též viditelný základ struktury. Zobrazuje se zatím pouze ve vodorovných pruzích. Částečně se zobrazuje i tloušťka vrstvy. Světlejší část v horní prostřední části, a naopak tmavší místa v dolních rozích jsou v souladu se vstupním vzorkem.



Obrázek 41 a) první hlavní komponenta b) druhá hlavní komponenta c) třetí hlavní komponenta

Druhá hlavní komponenta zobrazuje především hrubý základ struktury. Tvoří jej pravidelně střídající se čtverce připomínající šachovnici. Místa s tenkou vrstvou mají strukturu zřetelně viditelnou. Proto se i v této komponentě zobrazují výrazněji. Místa s větší vrstvou materiálu,

jako je na pravém okraji, mají tuto strukturu výraznou méně. Okolí prvku zůstává téměř konstantní.

Třetí hlavní komponenta je zásadní pro tvar vzorku. Konkrétně zaoblení rohů. Podobný význam má i šestá hlavní komponenta.

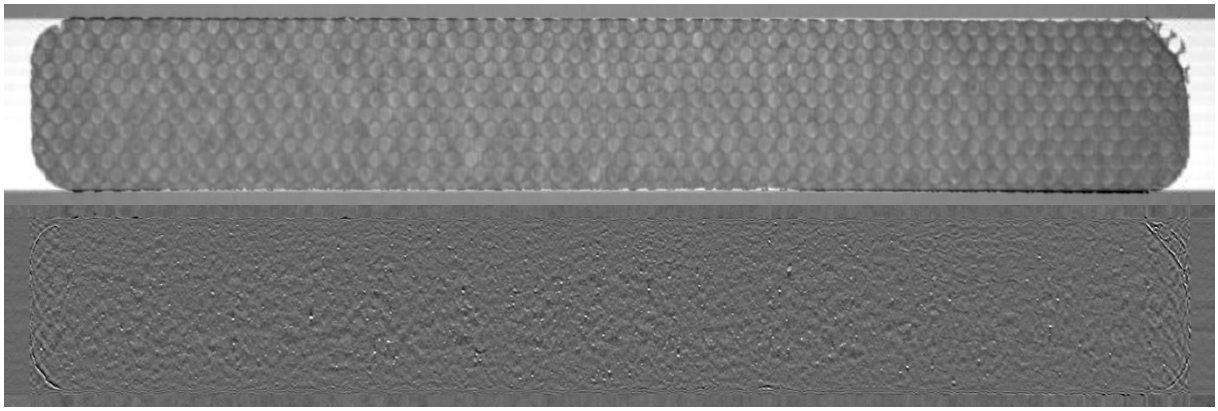
V dalších komponentách se postupně vyskytují další zpřesnění struktury. Jsou však stále méně výrazná. Velmi rychle též mizí i jejich důležitost v rámci celého obrazu. Struktura přestává být viditelná přibližně u padesáté komponenty. Stá komponenta již nenese prakticky žádnou informaci.

Pořadí komponent náležící příslušným strukturám se navíc u každého obrázku liší. Např. pro jiný vzorek se rohy promítnou ve druhé a osmé komponentě, namísto třetí a šesté. Původní předpoklad, že se struktura bude dát jednoduše odstranit odstraněním pouze několika komponent, byl tedy chybný.

Je zapotřebí odstranit všechny komponenty obsahující strukturu. Při testování se osvědčilo odstranění prvních 50 komponent. Ze vzorku bohužel neodstraníme pouze nežádanou strukturu, ale i informace o tvaru vzorku a rozložení (tloušťce) vrstvy materiálu. Malé náhodné vady však jako jediné zůstanou. Toho bude později využito při detekci kapiček. (Pro lepší zobrazení bylo využito ekvalizace).

Naopak při zachování prvních 50 komponent dostáváme původní obraz bez malých vad. Navíc metoda zvýraznila odchylku tvaru. Jasně lze vidět černá místa tam, kde vzorek přesahuje hranici obdélníku, bílá místa jsou naopak tam, kde materiál chybí. Tento způsob pro určení správnosti rozměrů je však nevhodný, jelikož hodnota hrany obdélníku je určena metodou, nikoliv uživatelem.

Obrázek 42 je navíc získán z opačného postupu než Obrázek 41. Zde jsou jako proměnné brány hodnoty v řádcích, jednotlivá měření ve sloupcích. Této změny ve výpočtu nejsnadněji dosáhneme transponováním vstupní matice, poté provedením celého výpočtu a na závěr opět transponováním výsledku.



Obrázek 42 a) PCA bez 50 prvních hlavních komponent b) 50 hlavních komponent PCA

3.4 Detekce vad

3.4.1 Pravidelný tvar požadovaných rozměrů

Pro porovnání velikosti a tvaru je nutné nejprve detekovat hrany vzorku. To provedeme analogicky jako při detekci vzorku ze vstupní fotografie, tedy kombinací prahování, dilatace, eroze a Cannyho hranového detektoru. Výstupem je spojitá bílá čára na bílém pozadí. Tu je potřeba porovnat s ideálními rozměry vzorku, určenými obsluhou programu pomocí 3 parametrů. Rozměrem X, Y a poloměrem zaoblení rohů R.

Hraniční body jsou rozděleny dle polohy do 8 kategorií (horní, dolní, pravá a levá strana a 4 rohy). Pro každý bod v těchto kategoriích je pak počítána chybovost. V okolí ideální křivky je toleranční pásmo. Výskyt hrany uvnitř tohoto pásma se považuje za správný a do celkové odchylky přičítá hodnotu 0. Velikost tohoto pásma je nastavitelná parametrem t a defaultně je volena ± 3 pixely. Body mimo toleranční pásmo do celkové odchylky přičítají druhou mocninu své vzdálenosti od okraje tolerančního pásma. Druhé mocniny je voleno pro její vlastnosti, kdy pro malé vzdálenosti není příliš významná, pro odlehlé body však velmi rychle narůstá. Konečný součet odchylek dělíme počtem bodů příslušného pásma. Tato hodnota je poté porovnávána s mezní přípustnou hodnotou. Pro vodorovné hrany je tato přípustná hodnota nastavena na 15, pro svislé 50 a pro rohy 100.

Z různých testovaných výpočtů se právě tento osvědčil jako nejspolehlivější. Přesto se zde setkáváme s několika problémy. Hlavním problémem je neschopnost odlišit tvarově poškozený vzorek (natržení, ohnutý roh, …) od nepoškozeného vzorku špatných rozměrů. K přesnému zjištění velikosti bylo použito několik algoritmů, jako jsou průměrná vzdálenost, střední vzdálenost nebo proložení křivkou a spočítání odchylek. Zásadním problémem se ukázaly již předchozí úpravy, především detekce samotného vzorku a následné oříznutí. Například pravá strana vykazovala odchylku +10 pixelů (přibližně 0,5 mm), zatímco levá -8 pixelů. Kontrola velikosti na obou stranách nevyšla, velikost vzorku je přitom správná. Ještě větší problém nastává v případě otřepu. Vzorku opsaný obdélník je výrazně větší a může být i natočený.

3.4.2 Tloušťkově homogenní vrstva

Kyselina hyaluronová je vlákněna na substrát výrazně větší, než je velikost vzorků. Ten je po skončení vláknícího procesu různými způsoby (stříhání, řezání, laserový paprsek a další) dělen na vzorky požadovaných velikostí. Vrstva vláken proto nezávisí pouze na průběhu vláknícího procesu, ale i na následném zpracování (umístění substrátu v řezacím stroji, umístění vzorku v substrátu). Každý vzorek v jednom cyklu proto může mít vrstvu různě velkou.

Pro určení tloušťky použijeme jako vstupní obraz výstup z Fourierovy transformace, při které bylo využito adaptivního filtru v kombinaci s low-pass filtrem. Právě tímto způsobem se totiž

podařilo nejlépe odstranit strukturu a zároveň uchovat původní homogenitu vrstvy. Jedinou nevýhodou této metody je tmavší pás při okrajích vzorku.

Při použití prahování bychom detekovali především světlé kapičky vzorku, proto je zapotřebí obraz nejprve rozmazat. K tomu využijeme funkci knihovny openCV blur. Metoda využívá konvoluce. Jelikož požadujeme velké plošné rozmazání, volíme velikost konvolučního jádra 13x13. Body, které jsou nižší než prahová hodnota, vyhodnotíme jako místa s tenkou vrstvou. Jejich součet by pak měl být nižší než hodnota tolerance. Takto porovnáváme jen body uvnitř vzorku, nikoliv body pozadí. Tyto body odlišíme úpravou předzpracovaného ořízntého obrázku kombinací funkcí dilatace, eroze a prahování. Vznikne obraz obsahující hodnoty 255 pro vnitřek vzorku a s hodnotami 0 mimo vzorek. Jednoduchou podmínkou if pak výpočet provádíme pouze tam, kde obraz obsahuje hodnoty 255.

Na Obrázek 43 je zobrazen výstup z detekce tloušťky vrstvy. Bílé pozadí představuje body, kde tloušťka vrstvy nebyla zkoumána. Šedé body náleží místům, kde není dostatečná vrstva materiálu. Program obsahuje pouze detekci tenké vrstvy. Hledání míst s příliš velkou vrstvou by se provedlo obdobně.

Vlivem konvolučního rozmazání zmizely ostré hrany vzorku. To z počátku působilo problémy chybně detekované tenké vrstvy na okrajích. Odstranění těchto okrajů bylo provedeno s využitím vlastnosti eroze zmenšovat objekty. Pro erozi proto bylo využito podstatně většího jádra (40), než pro dilataci (10).

Obrázek 43 Výstup z detekce tenké vrstvy

3.4.3 Plocha bez defektů (kapky, pruhy apod.)

Vznik kapiček byl popsán již v teoretické části.

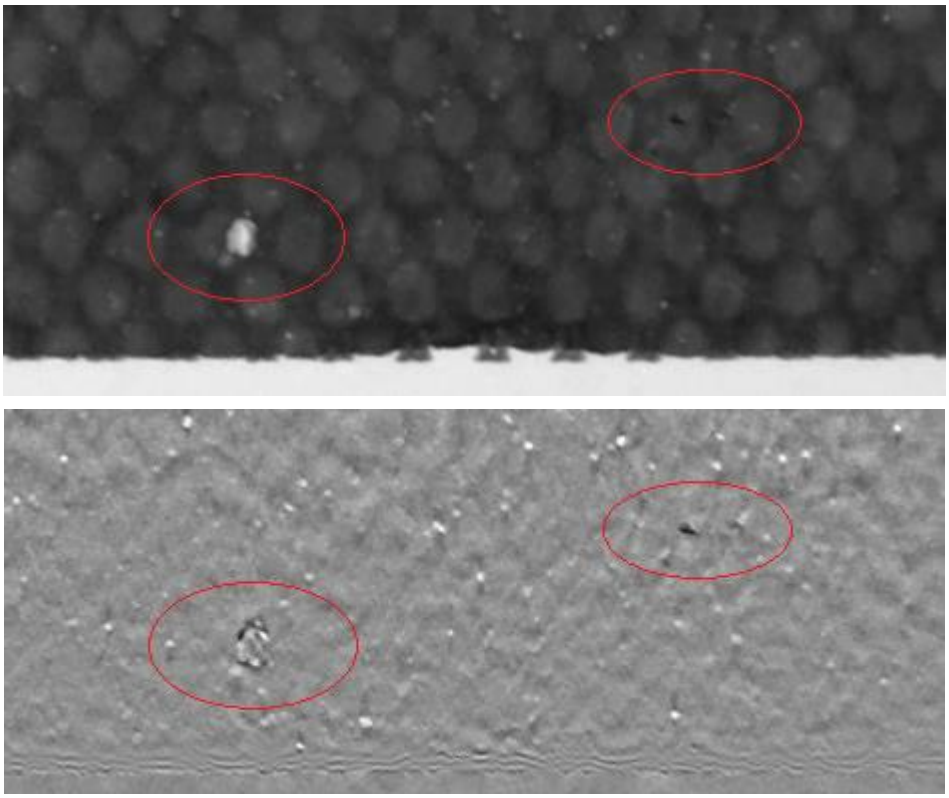
K detekci kapiček bylo využito třech různých způsobů, které zde budou postupně ukázány a v další kapitole porovnány. Bohužel spolehlivý způsob vhodný k detekci pruhů a dalších vnitřních plošných vad nebyl nalezen. Je to způsobeno především neschopností spolehlivě oddělit strukturu vzorku při zanechání proměnné tloušťky vrstvy v celé ploše vzorku.

Výstup z metody PCA a následná úprava

Z metody PCA jako výstup dostáváme vzorek, který má již odfiltrovanou strukturu, odfiltrovanou měnící se tloušťku vrstvy a okolí vzorku nabývá přibližně stejných hodnot jako vnitřek vzorku. Výraznější odchylky se objevují na hranách vzorku, především pak v rozích. Při

detekci kapiček však nejsou zajímavé, proto je do výpočtu nezahrneme obdobně jako při detekci tenké vrstvy.

Malé kapičky jsou detekovány spolehlivě, jak je ukázáno na Obrázek 44. Tmavá zrnka (shluky nerozpuštěného polymeru, krystalky solí a podobné) jsou též detekovány bez větších problémů. Zásadní nedostatek této metody se objevuje při detekci velkých kapek. Metoda PCA takové kapky vůbec nedetekuje, případně je detekuje pouze jako několik malých kapek. I jediná taková kapka by přitom měla znamenat vyřazení vzorku. Z tohoto důvodu je tato metoda nevhodná.



Obrázek 44 Detekce kapiček PCA

Výstup z Fourierovy transformace a následná úprava

Výstup Fourierovy transformace zřetelně zobrazuje kapičky všech velikostí. Světlost i velikost kapiček se přesně shoduje se vstupním obrázkem, proto je tento výstup pro jejich detekci vhodný. Další zpracování mírně komplikuje pomalu měnící se pozadí uvnitř vzorku. To částečně představuje skutečnou tloušťku vrstvy vzorku. Fourierova transformace však do obrazu vnesla zkreslení v určitých vzdálenostech od okrajů.

Zásadním problémem je správné určení přípustné a nepřípustné chybovosti. Kapičky je nutné rozdělit do kategorií podle velikosti a světlosti. Počet nalezených kapiček v příslušné kategorii rozhodne o případném vyřazení.

Světlost detekovaných kapiček udává nastavená prahová hodnota. Pro určení velikosti kapiček využijeme část kódu, kterým kapičky zobrazujeme ve vstupním obrazu. Stejně jako v části 3.2 Separace vzorku využijeme funkcí cv2.findContours a cv2.minAreaRect k detekci okrajů kapiček z předem upraveného binárního obrazu. Druhou funkcí pak získáme polohu a rozměry jednotlivých čtverců opsaných jednotlivým kapičkám. Pak už stačí jen vypočítat obsah jednotlivých čtverců a porovnáním rozřadit do kategorií.

Konvoluce

Základní myšlenkou pro použití této metody je skutečnost, že kapičky mají ve většině případů stejný tvar i velikost. Ten je navíc výrazně odlišný (velikostně) od zbytku vzorku.

Je tedy potřeba najít vhodnou konvoluční masku, která po konvoluci se vzorkem bude dávat následující výsledky. Body nalézající se uprostřed kapky by měly nabývat vysokých hodnot, zatímco body v neměnném prostředí by měly nabývat nulových hodnot. Příklad masky nejlépe detekující kruhové kapičky o velikosti 7x7 pixelů je na Obrázek 45.

```
[[-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],
 [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],
 [-1, -1, -1,  0,  2,  2,  2,  0, -1, -1, -1],
 [-1, -1,  0,  2,  2,  2,  2,  2,  0, -1, -1],
 [-1, -1,  2,  2,  2,  2,  2,  2,  2, -1, -1],
 [-1, -1,  2,  2,  2,  2,  2,  2,  2, -1, -1],
 [-1, -1,  2,  2,  2,  2,  2,  2,  2, -1, -1],
 [-1, -1,  0,  2,  2,  2,  2,  2,  0, -1, -1],
 [-1, -1, -1,  0,  2,  2,  2,  0, -1, -1, -1],
 [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],
 [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],]
```

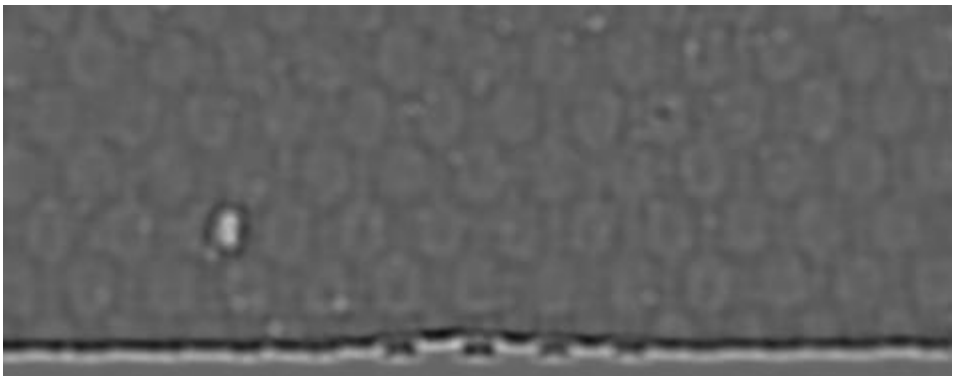
Obrázek 45 Příklad konvoluční masky pro detekci kapiček 7x7 pixelů

K využití právě této velikosti masky bylo rozhodnuto na základě velikosti kapiček. Ta může být přibližně zjištěna přepočtem milimetrů na pixely (1 mm = cca 20px) nebo odečtena při zobrazení hodnot matice obrázku. Hodnoty masky jsou pak voleny tak, aby celkový součet byl roven 0. Právě tato vlastnost zaručuje pro neměnnou část vzorku výstupní hodnoty přibližně rovny nule.

Na Obrázek 46 je zobrazen výstup z konvoluce. Zřetelně je vidět detekovaná velká kapka, což bylo hlavním cílem (velikost kapičky přibližně odpovídá velikosti jádra masky). Menší kapičky stále vidět jsou, ale došlo spíše ke zhoršení viditelnosti. Viditelné jsou též pozůstaky struktury. Ta zde nebyla nijak filtrována, proto je tento výsledek pochopitelný. Oproti světlosti velké kapky je však málo výrazná (tmavší), proto nebude problém ji při detekci oddělit. Zajímavé je výrazné zviditelnění hrany vzorku. Na vnitřní hraně tmavá barva skokově přejde v barvu bílou. Následně se opět ústálí na průměrné hodnotě. Tato průměrná hodnota by měla být dle výše

vysvětlené fukce nulová, tedy černá. Nulová také skutečně je. V postupu je však pro zobrazení i záporných hodnot zahrnuta adaptivní ekvalizace.

Testovány byly i další masky pro různé velikosti kapiček (5,7,9,11). Zvýrazněny pak byly nejvíce vždy kapičky příslušící velikosti jádra masky. Rušivé detekování hran struktury však rychle rostlo při využití menších masek. Maska se totiž čím dál více podobala základním maskám využívaných při detekci hran, jež jsou zmíněny v teoretické části. Nejvhodnějším kompromisem se pak stala maska pro kapičky 7x7 pixelů, která bez problémů detekovala velké a střední kapky. Vymizení malých kapek bych označil spíše za výhodu, která usnadňuje další zpracování.



Obrázek 46 Detekce kapek konvolucí

3.4.4 Okraje bez defektů (otřepy, překlady nebo shrnutí vrstvy)

Poškození okrajů vzniká především při řezání na požadovaný tvar (otřepy) a při manipulaci (shrnutí vrstvy, překlady). Kyselina hyaluronová totiž reaguje s vlhkostí. Při neopatrné manipulaci se může přilepit k prstům, nebo k částem automatických manipulátorů (proces v současné době ještě není automatizován). Výskyt otřepů je v určité míře tolerovatelný (případně ručně opravitelný), překlady nebo chybějící vrstva jsou netolerovatelné.

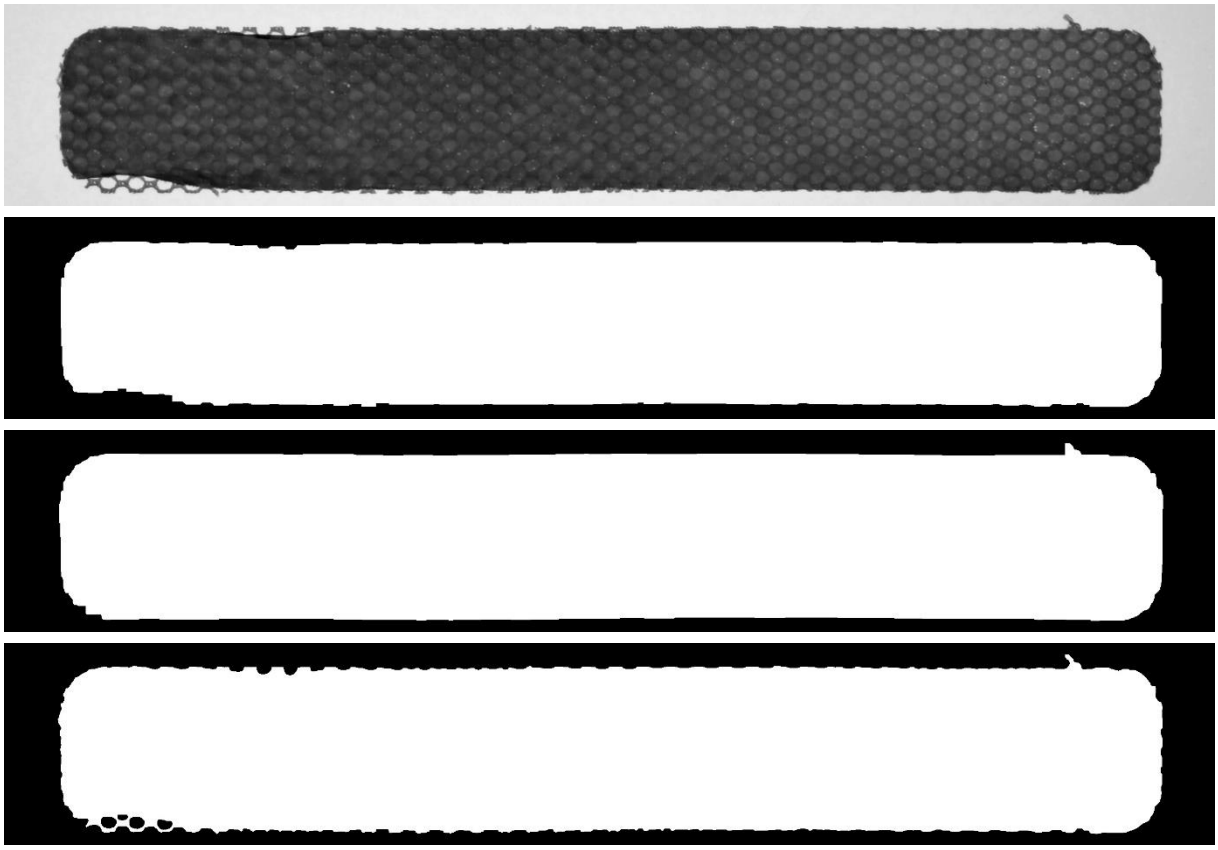
Poškození okrajů je částečně zahrnuto již v kapitole 3.4.1, kde počítáním odchylky od ideální křivky ověřujeme nezávadnost vzorku. V této části se proto zaměříme na přímé detekování a označení místa, kde se některá z vad okrajů nalézá.

Obrázek 47 Detekce přeložení (dvojité vrstvy)

Překlad vrstvy by měl být detekovatelný nejlépe. Je zde totiž dvakrát větší vrstva než kdekoliv jinde ve vzorku. Jediný problém je tedy odlišit dva extrémní případy. Přeložená hraničně tenká vrstva (dvojitá vrstva) by mohla nabývat stejných hodnot, jako jiný vzorek s hraničně tlustou

vrstvou. K řešení bylo využito adaptivní nastavení prahové hodnoty. Prahová hodnota je určena odečtením konstantní hodnoty (50) od mediánu vzorku. Tím zajistíme funkčnost pro vzorky s různou vrstvou materiálu.

Otřepy a chybějící vrstva jsou detekovány v rámci jedné metody. Ta je postavena na různé přesnosti detekování objektů. K detekci je opět použito kombinace funkcí prahování, dilatace a eroze. Tentokrát však nedostáváme výsledek přímo z těchto metod, ale z porovnání, kdy byl tento proces proveden s různě nastavenými parametry.



Obrázek 48 Detekce otřepů a shrnutí a) vstupní poškozený vzorek b) hrubá detekce – odebrání c) hrubá detekce – doplnění d) přesná detekce

Obrázek 48 b) zobrazuje hrubý tvar vzorku. Místa obsahující pouze strukturu nejsou zahrnuta. Otřepy jsou také vyjmuty. Toho je docíleno nejdříve prahováním (jinak by funkce dilatace a eroze nebylo možné použít). Následuje jemná dilatace k ucelení vnitřku vzorku (kapičky a podobně). Jádro dilatace je voleno 10x10. Následuje výrazná eroze s jádrem 40x40. Ta odstraní vše kromě ucelené vrstvy vzorku. Odchylky, jako jsou vnější otřepy nebo samostatný substrát, jsou odstraněny. Porovnatelná data získáme pomocí dilatace s jádrem 30x30. Ta data opět zvětší, odstraněné části struktury a otřepy se zpět už neukáží.

Opačný přístup je na Obrázek 48 c). Zde je nejdříve použita výrazná dilatace (40x40), která spojí jak malé chybějící kapičky, tak i větší mezery ve struktuře bez vláken. Ze vzorku je tak velký pravidelný obrazec. I jediný bod v pozadí vzorku (který prošel prahováním) se však zvětšil

na čtverec 40x40 pixelů. Proto je třeba použít ještě větší erozi (50x50) a následně dilatací (10x10) upravit na původní rozměry. Tímto postupem získáme pravidelný tvar bez poškození. Otřep je připojen ke vzorku. Jeho vnější velikost se však nezměnila.

Oba tyto postupy jsou porovnány s jemnou detekcí hran, tedy prahování, dilatace (10x10), eroze (14x14) a dilatace (4x4). Ta je zobrazena na Obrázek 48 d).

Na Obrázek 48 byla využita vždy čtvercová konvoluční jádra. Testována byla také jádra kruhová. Čtvercová jádra lépe fungují na svislých a vodorovných hranách, kde se jim daleko lépe daří udržet rovinu. Rohy, obzvláště pak ty poškozené, jsou při hrubé detekci zobrazeny skokově. To může mít za následek chybnou detekci vady. V rozích je vhodnější využití kruhových jader.

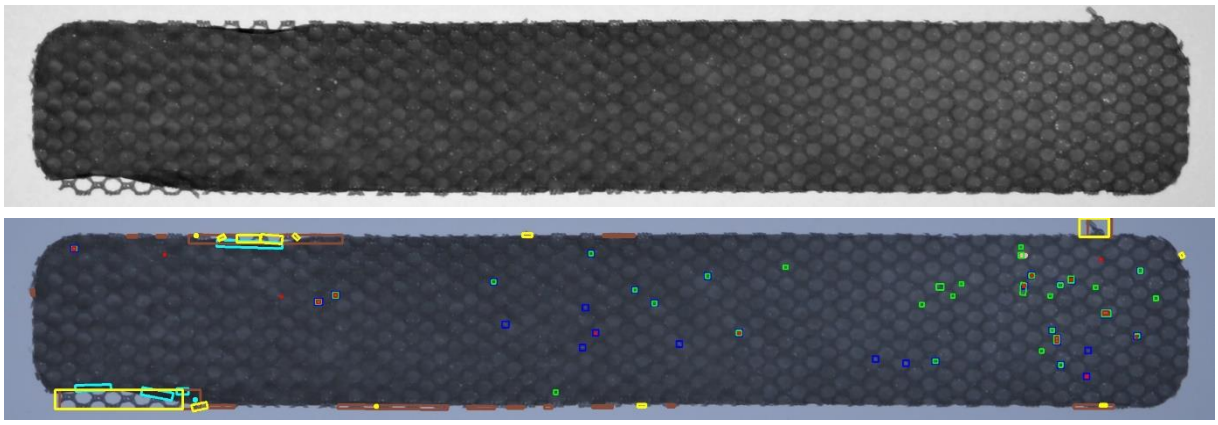
Konečná detekce je pak určena odchylkou hrubé a jemné detekce, případně lze použít rozdíl hrubých detekcí (doplnění a odebrání). Data je samozřejmě ještě nutné upravit a správně zobrazit. Hlavní princip metody však již byl ukázán.

3.5 Testování funkce

Pro grafické znázornění byly vybrány 3 vzorky (č.1, č.5, č.10), na kterých bude ukázána a diskutována funkčnost jednotlivých metod. Vždy je pro porovnání ukázán i ekvalizovaný černobílý vstupní obraz. Jednotlivé metody jsou vykresleny příslušnými barvami dle Tabulka 1.

Kapičky PCA	Kapičky Fourier	Kapičky Konvoluce	Dvojitá vrstva	Tenká vrstva	Okraje kruh	Okraje čtverec

Tabulka 1 Barvy označující typ detekované vady



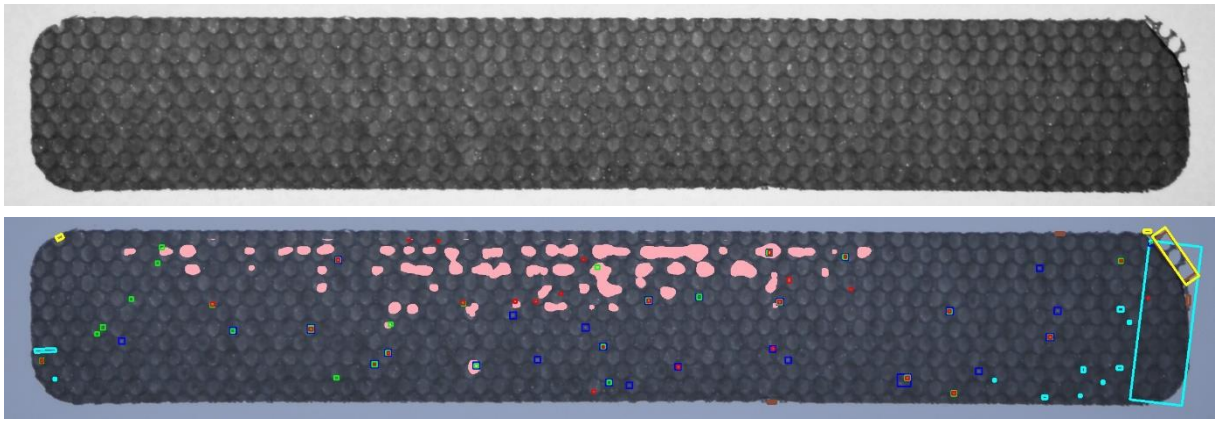
Obrázek 49 Vzorek č.1

Tento vzorek byl zahrnut především jako ukázka detekce hran. Otřep byl spolehlivě detekován za použití obou jader dilatace a eroze. Stejně tak i velké shrnutí vrstvy v levé části. Rozdílně jsou detekovány jemné výstupky struktury. Metoda se čtvercovým jádrem detekuje výrazně

větší množství těchto výstupků. Metoda s kruhovým jádrem naopak detekuje vadu v pravém horním rohu. Podobný (výraznější) případ je i u vzorku č.5 v levém horním rohu.

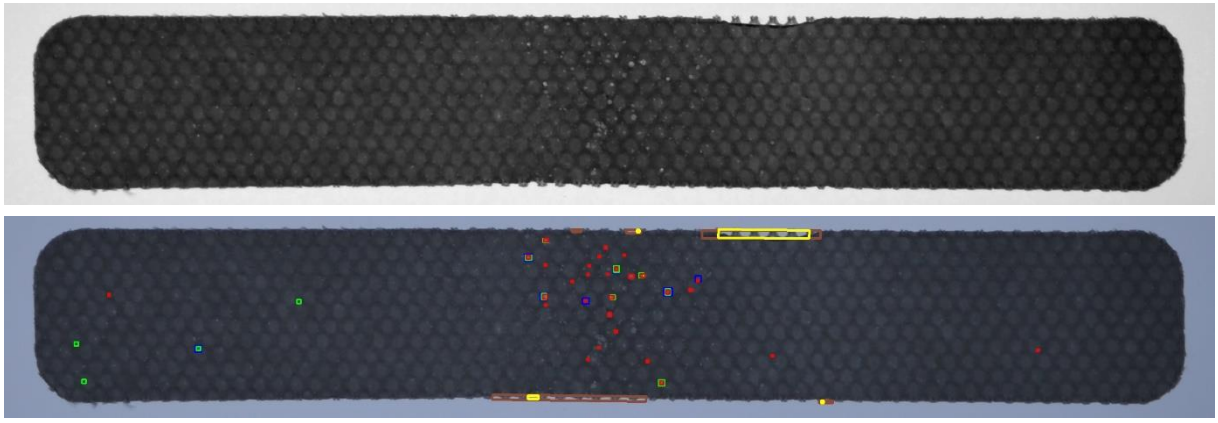
Detekce dvojité vrstvy dopadla také dle očekávání. Ty se objevují právě na hranicích, kde byla vrstva shrnuta.

Detekce kapiček bude popsána na detailnějších obrázcích. Již zde je však vidět, že každá z metod detekuje vady odlišně.



Obrázek 50 Vzorek č.5

Vzorek č.5 byl ukázán z důvodu detekování tenké vrstvy. Ta byla v detekovaných místech vykreslena růžovou barvou. Důležité je však diskutovat spíše chování detekce dvojité vrstvy. Ta na několika místech, a především na celém pravém okraji, detekuje dvojitou vrstvu. Otázkou je, zda se špatná detekce ovlivněná chybou, která je již detekována jinou metodou, dá brát jako chyba této metody. Vzhledem k faktu, že dvojitá vrstva vzniká shrnutím okraje, což je spolehlivě detekované jinou metodou, nebyla tato metoda doporučena pro využití ve výrobě. Další otázkou je, zda by vzorky s velkou proměnlivostí tloušťky vrstvy (i když jsou z obou stran v toleranci) neměly být označeny za chybné.



Obrázek 51 Vzorek č.10

Na vzorku č.10 jsou demonstrovány dva případy. Protože k testování byly poskytnuty pouze vyřazené (vadné) vzorky, bylo nutné pro porovnání použít kvalitní části jinak nekvalitních

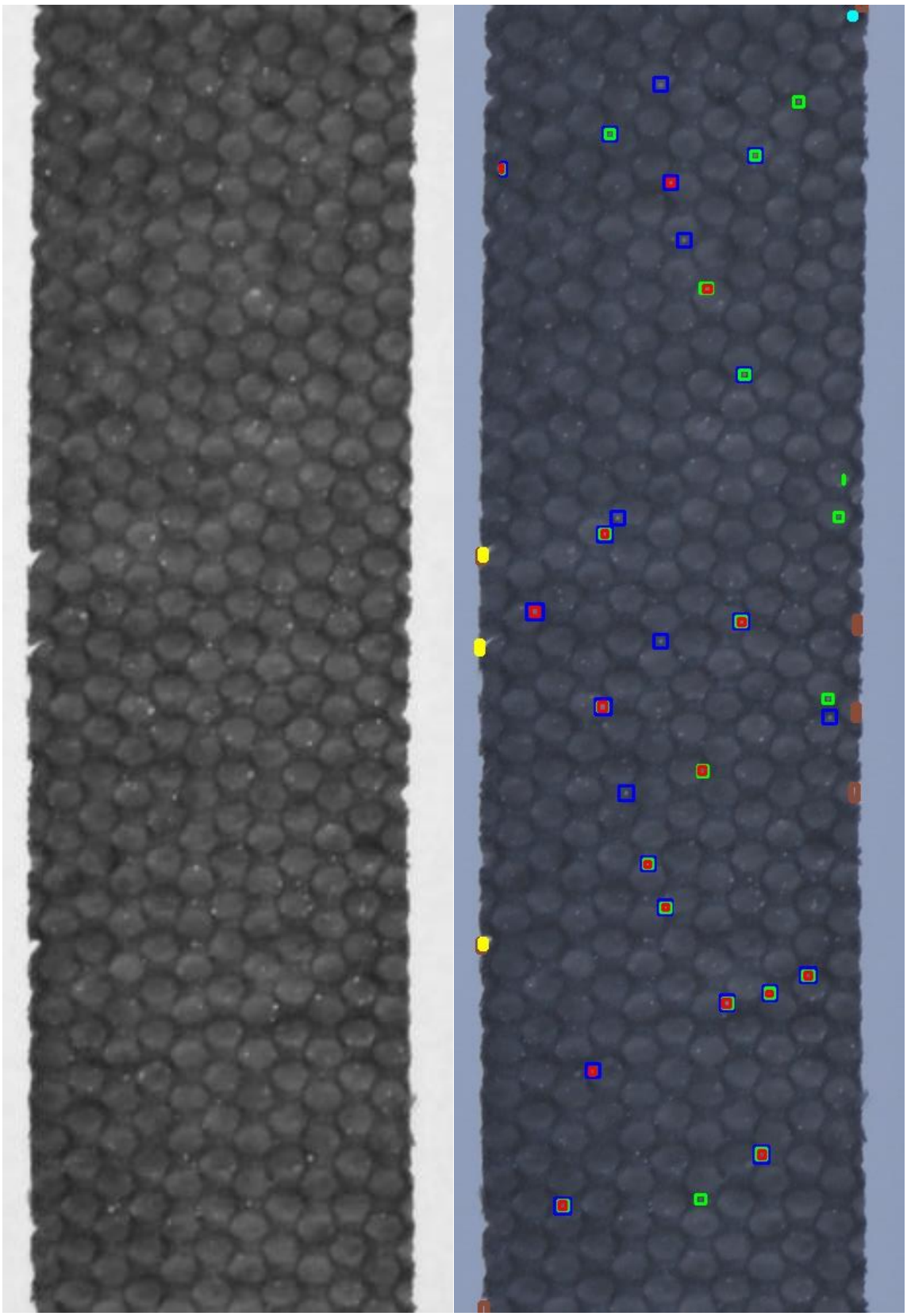
vzorků. K tomuto účelu sloužil právě tento a několik dalších vzorků. V pravé části je detekována pouze jedna kapička metodou konvoluce, v levé pak kapiček 5, z toho 4 Fourierovou transformací.

Uprostřed vzorku je detekováno velké množství kapiček. Zajímavé na těchto kapičkách je především jejich umístění v jednom místě a jejich velikost. Vše naznačuje tomu, že tyto vady nevznikly náhodně při vláknícím procesu, ale až po jeho skončení dotykem prstu. Tento typ kapiček je spolehlivě detekován právě konvolucí, která má jádro nastavené na velikost těchto kapek.

Nalezené kapičky jsou vykreslovány v různých velikostech. Je tak učiněno pro správné zobrazení případů, kdy je daná kapička nalezena více metodami. Jejich velikost proto nehraje roli. Na Obrázek 52 je deteilní pohled na detekované chyby ve vzorku. Z něj lze částečně vyčíst výhody a nevýhody všech tří metod.

PCA	
Výhody	Malé ovlivnění na okrajích vzorku Nejlépe detekuje malé kapičky
Nevýhody	Mírné ovlivnění strukturou (větší pravděpodobnost výskytu na struktuře) Nedetekuje velké kapky
Fourier	
Výhody	Není ovlivněna strukturou Detekuje malé i velké kapky
Nevýhody	Závislosti na poloze ve vzorku (obsahuje světlejší místa) Na okrajích (v rozích) vzorku nelze detekovat
Konvoluce	
Výhody	Nastavitelná velikost detekovaných kapek
Nevýhody	Ovlivněna strukturou (větší pravděpodobnost výskytu chyby na hranicích struktury) Na okrajích (celý obvod) vzorku nelze detekovat

Tabulka 2 Výhody a nevýhody metod detekce kapiček



Obrázek 52 Detekce kapiček

3.6 Návrh řešení

Pro průmyslové použití bych navrhl sekvenci nejspolehlivějších metod v pořadí

1) segmentace,
2) okraje (shrnutí, otřepy),
3) tvar (velikost),
4) tloušťka vrstvy,
5) kapičky – Konvoluce (velikost jádra 5 a 9).

Segmentace musí být první krok za všech okolností. Kontrola okrajů následuje hned po ní, jelikož závažnost takových vad znamená okamžité vyřazení. Spolu s kontrolou tvaru poskytnou dostatečnou informaci o možných vnějších (okrajových) vadách vzorku. Kontrola tloušťky vrstvy využívající Fourierovu transformaci je na místě třetím a sekvenci končí kontrola kapiček využívající konvoluci. Nejprve je otestována metoda s jádrem velikosti 9x9 a vyšší prahovou hodnotou. Nalezení takové chyby znamená vadný celý vzorek. Poté je provedena jemnější konvoluce s jádrem 5x5, nižší prahovou hodnotou a tolerancí výskytu dvaceti kapiček.

Dalším vhodným řešením by mohl být následující postup. V tomto řešení by se pořídily 2 fotografie. První by byla pořízena hned po zvlákňování. Provedeny by byly metody

1) kapičky – Fourierova transformace,
2) tloušťka vrstvy – Fourierova transformace.

Teprve poté by byl navlákněný materiál dělen na jednotlivé vzorky. Následně by byla pořízena druhá fotografie. Zde by byly provedeny pouze funkce

1) segmentace,
2) okraje,
3) tvar,
4) kontrola dalšího mechanického poškození.

Tento navrhovaný postup by představoval mírnou komplikaci způsobenou nutností provádět kontrolu dvakrát, mohl by však přinést výrazné zkvalitnění kontroly. Obraz by totiž neobsahoval žádné okraje, tedy výrazné vysoké frekvence. Jediné výrazné vysoké frekvence by náležely právě struktuře substrátu. Ve Fourierově spektru by je bylo snadné detekovat a následně odstranit. Jejich odstraněním bychom dostali původní obraz s odstraněnou strukturou. Dalším zpracováním bychom předem dostali místa, která jsou nevhodná pro další

zpracování. Těchto dat by bylo využito k určení míst řezů k získání výchozích vzorků. Snížila by se proto zmetkovitost, naopak vytěžitelnost by vzrostla.

Ze získaných zkušeností se domnívám, že při obdobném využití metody PCA bychom výrazného zlepšení nedosáhli. Metoda totiž nerozpoznává strukturu jako celek, ale jako kmitání jednotlivých sloupců (řádků), které mění svůj počátek a díky proměnné vrstvě i intenzitu. Struktura tohoto typu se proto postupně prolíná téměř všemi důležitými hlavními komponentami a nelze ji jednoduše odstranit.

4 Závěr

Cílem této práce bylo seznámit se s výrobou a vlastnostmi nanovlákenných materiálů, získat potřebná data a vytvořit spolehlivý nástroj k detekci jejich vad. V první části diplomové práce byly zkoumány principy funkcí, které by k tomuto účelu byly vhodné. Získané znalosti poté byly využity při testování jednotlivých funkcí v jazyce Python. Významnou částí práce bylo zkoumání funkcí metody hlavních komponent (PCA) a 2D Fourierovy transformace a možnosti filtrace při použití těchto metod. Tyto metody se ukázaly jako užitečné především pro odstranění struktury substrátu, která komplikovala správnou detekci. Lepších výsledků bylo získáno Fourierovou transformací, a to s dvěma různými filtry. První filtr ponechává pouze vady v obraze, druhý ponechá vady i tloušťku vrstvy. Ve druhém případě se však v obraze objevuje několik špatně detekovaných frekvencí, které výstup značnou měrou ovlivňují. Metodou PCA je možné detekovat pouze malé vady, a proto se ukázalo pro toto použití jako nejméně vhodná.

Následně byly testovány funkce navržené pro detekci vad. S velkou úspěšností byly detekovány okrajové vady, jako jsou otřepy, shrnutí vrstvy nebo špatný rozměr. Úspěšně také byla detekována místa s příliš tenkou vrstvou. Při detekci strukturních vad byla úspěšnost menší. Jedním z důvodu je fakt, že doposud byly vzorky hodnoceny subjektivním pohledem pracovníka. Nalezení přesné definice vady proto bylo v některých případech obtížné. Nejtěžší pak bylo definování přípustné hodnoty kapiček. Ty se ve vzorku objevují v množství několika desítek až stovek, v různých velikostech a intenzitách. Byly proto navrženy, popsány a názorně ukázány 3 detekční funkce, jejich výhody a jejich nedostatky. Na základě výsledků pak byla navržena dvě možná řešení pro aplikaci ve výrobě.

Uživatelská část programu začíná na řádku č. 575, kde je třeba zvolit uživatelské parametry, jako jsou například přibližná velikost vzorku pro segmentaci, manuálně volené parametry pro ekvalizaci (automatická by ovlivnila výstupní hodnocení) nebo ideální rozměry pro výpočet odchylky velikosti při výstupní kontrole. Dále je vlastní tělo programu, které volá jednotlivé funkce definované výše. Dbáno je především na přehlednost a dostatečnou dokumentaci kódu.

Seznam použité literatury

[1] WENDORFF, Joachim H., Seema AGARWAL, Andreas GREINER a Seema AGARWAL. *Electrospinning: Materials, Processing, and Applications* [online]. Hoboken, GERMANY: John Wiley & Sons, Incorporated, 2012 [vid. 2018-10-11]. ISBN 978-3-527-64773-6. Dostupné z: http://ebookcentral.proquest.com/lib/cvut/detail.action?docID=865188

[2] *Nanopharma - Technologie* [online]. [vid. 2018-11-20]. Dostupné z: http://www.nanopharma.cz/cs/produkty-a-technologie/technologie

[3] *Základy anatomie - nervový systém a čivy | Fakulta sportovních studií Masarykovy univerzity* [online]. [vid. 2018-11-28]. Dostupné z: https://is.muni.cz/do/fsps/e-learning/zaklady_anatomie/zakl_anatomie_IV/pages/civy.html

[4] Jak vyrábíme kyselinu hyaluronovou. *Continoviny*. Contipro a.s., 2018.

[5] JIŘÍ ŽÁRA, BEDŘICH BENEŠ, JIŘÍ SOCHOR a PETR FELKEL. *Moderní počítačová grafika*. 2004. ISBN 80-251-0454-0.

[6] HLAVÁČ, Václav. Hledání hran. In: . České vysoké učení technické v Praze. nedatováno

[7] RADOVAN HÁJOVSKÝ, RADKA PUSTKOVÁ a FRANTIŠEK KUTÁLEK. *Zpracovani obrazu v merici a ridici technice* [online]. [vid. 2018-11-14]. Dostupné z: http://www.person.vsb.cz/archivcd/FEI/ZOMRT/Zpracovani%20obrazu%20v%20merici%20a%20ridici%20technice.pdf

[8] *Canny Edge Detection — OpenCV 3.0.0-dev documentation* [online]. [vid. 2018-11-07]. Dostupné z: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_canny/py_canny.html

[9] ZDENĚK KRŇOUL. *Zpracování digitalizovaného obrazu - Segmentace* [online]. B.m.: Západočeská univerzita v Plzni. [vid. 2018-11-14]. Dostupné z: http://www.kky.zcu.cz/uploads/courses/zdo/prezentace/zdo-05.pdf

[10] JIŘÍ JENČ. *Využití jednodeskových počítačů a neuronových sítí pro automatické rozpoznání RZ vozidel*. B.m., 2015. České vysoké učení technické v Praze.

[11] *Morphological Transformations — OpenCV 3.0.0-dev documentation* [online]. [vid. 2018-12-01]. Dostupné z: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html

[12] DOC. ING. MICHAEL VALÁŠEK, DRSC A KOLEKTIV. *Mechatronika*. B.m.: ČVUT, 1996. ISBN 80-01-01276-X.

[13] *Image Transforms - Fourier Transform* [online]. [vid. 2018-11-07]. Dostupné z: https://homepages.inf.ed.ac.uk/rbf/HIPR2/fourier.htm

[14] BATARD, Thomas a Michel BERTHIER. Spinor Fourier Transform for Image Processing. *IEEE Journal of Selected Topics in Signal Processing* [online]. 2013, **7**(4), 605–613. ISSN 1932-4553, 1941-0484. Dostupné z: doi:10.1109/JSTSP.2013.2259796

[15] *Rychlá Fourierova transformace* [online]. Dostupné z: http://apfyz.upol.cz/ucebnice/down/mini/fourtrans.pdf

[16] ENHANCEDATASCIENCE. R Basics: PCA with R. *Enhance Data Science* [online]. 7. květen 2017 [vid. 2018-11-06]. Dostupné z: http://enhancedatascience.com/2017/05/07/r-basics-pca-r/

[17] MILITKÝ, Í a Milan MELOUN. METODA HLAVNÍCH KOMPONENT A EXPLORATORNÍ ANALÝZA VÍCEROZMĚRNÝCH DAT. nedatováno, 11.

[18] RICHARD A. JOHNSON a DEAN W. WICHERN. *Applied Multivariate Statistical Analysis* [online]. nedatováno [vid. 2018-10-29]. 6. Dostupné z: https://www.pearson.com/us/higher-education/program/Johnson-Applied-Multivariate-Statistical-Analysis-6th-Edition/PGM274834.html

[19] dimensionality reduction - Relationship between SVD and PCA. How to use SVD to perform PCA? *Cross Validated* [online]. [vid. 2018-11-06]. Dostupné z: https://stats.stackexchange.com/questions/134282/relationship-between-svd-and-pca-how-to-use-svd-to-perform-pca

[20] ZUZANA_TONHAUSEROV. *Metoda hlavních komponent a její aplikace* [online]. B.m., 2013 [vid. 2019-01-04]. Univerzita Palackého v Olomouci, Přírodovědecká fakulta, katedra matematické analýzy a aplikací matematiky. Dostupné z: https://theses.cz/id/iwan2b/Zuzana_Tonhauserov_-_Metoda_hlavnch_komponent.pdf

[21] Welcome to Python.org. *Python.org* [online]. [vid. 2018-11-30]. Dostupné z: https://www.python.org/psf-landing/

[22] Python (programming language). *Wikipedia* [online]. 21. listopad 2018 [vid. 2018-11-30]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Python_(programming_language)&oldid=870005904

[23] PEP 404 -- Python 2.8 Un-release Schedule. *Python.org* [online]. [vid. 2018-11-30]. Dostupné z: https://www.python.org/dev/peps/pep-0404/

[24] *NumPy — NumPy* [online]. [vid. 2018-11-30]. Dostupné z: http://www.numpy.org/

[25] *Scientific Computing Tools for Python — SciPy.org* [online]. [vid. 2018-11-30]. Dostupné z: https://www.scipy.org/about.html

[26] *scikit-learn: machine learning in Python — scikit-learn 0.20.1 documentation* [online]. [vid. 2018-11-30]. Dostupné z: https://scikit-learn.org/stable/

[27] *About - OpenCV library* [online]. [vid. 2018-11-30]. Dostupné z: https://opencv.org/about.html

[28] dimensionality reduction - How to reverse PCA and reconstruct original variables from several principal components? *Cross Validated* [online]. [vid. 2018-10-23]. Dostupné z: https://stats.stackexchange.com/questions/229092/how-to-reverse-pca-and-reconstruct-original-variables-from-several-principal-com

Přílohy

Pro kompletnost práce je uveden vytvořený program. Na přiloženém CD disku je pak jeho elektronická verze, která je připravena k použití. Elektronická verze obsahuje navíc komentáře popisující funkci. Ty jsou v tištěné verzi odstraněny kvůli přehlednosti a omezením tisku. Přiložené CD dále obsahuje soubor pořízených a testovaných fotografií.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from numpy import linalg as la
import cv2
import scipy.signal

def show1(image,name):
    cv2.namedWindow(name, cv2.WINDOW_NORMAL)
    cv2.resizeWindow(name,1500,1000)
    cv2.imshow(name,np.uint8(image))

def show(image,name):
    cv2.namedWindow(name, cv2.WINDOW_NORMAL)
    cv2.resizeWindow(name,size_cut[0],size_cut[1])
    cv2.imshow(name,np.uint8(image))

def histogram_RGB(hist):
    color = ('b','g','r')
    for i,col in enumerate(color):
        histr = cv2.calcHist([np.uint8(hist)],[i],None,[256],[0,256])
        plt.axis([0,260,0,300000]);
        plt.plot(histr,color = col)
    plt.grid(True)
    plt.show()

def histogram(hist):
    histr = cv2.calcHist([np.uint8(hist)],[0],None,[256],[0,256])
    plt.axis([0,250,0,30000]);
    plt.plot(histr)
    plt.grid(True)
```

```
    plt.show()

def screeplot(s):
    screeplot = np.zeros(size_cut[1])
    sumS=0
    for i in range(size_cut[1]-1):
        sumS = sumS+s[i]
        screeplot[i+1] = s[i]/sumS+screeplot[i]
    plt.plot(screeplot);plt.show()

def boundaries(image):
    for (y,x), value in np.ndenumerate(image):
        if image[y,x] > 255:
            image[y,x] = 255
        elif image[y,x] < 0:
            image[y,x] = 0
    return image

def segmentace(image):
    size = np.int0([4000,2666])
    img = image[0:size[1], 0:size[0]]
#    show1(img,'Vstupni obraz')
#    grey = cv2.cvtColor( img, cv2.COLOR_RGB2GRAY )
    blue = img[:,:,0].copy()
    histogram_RGB(img)
#    histogram(blue)
    prahovani = cv2.threshold(blue, 120, 255, cv2.THRESH_BINARY_INV)[1].astype(np.uint8)
    dilatace = cv2.dilate(prahovani, cv2.getStructuringElement(cv2.MORPH_RECT, (10, 10)))
    eroze = cv2.erode(dilatace, cv2.getStructuringElement(cv2.MORPH_RECT, (40, 40)))
    dilatace = cv2.dilate(eroze, cv2.getStructuringElement(cv2.MORPH_RECT, (30, 30)))
    eroze, contours, hierarchy = cv2.findContours( dilatace, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
#    cv2.drawContours(img,[contours[0]],-1,(0,0,255),5)
    for cont in contours:
        rect = cv2.minAreaRect(cont)
    sizeR = np.int0((rect[(1)]))
    if sizeR[0] > size_cut[0] or sizeR[1] > size_cut[1]:
        print('nutno zvetsit parametr size_cut, sizeR = ',sizeR)
    elif size_cut[0] > 1.3*sizeR[0] or size_cut[1] > 1.3*sizeR[1]:
```

```
        print('nutno zmensit parametr size_cut, sizeR = ',sizeR)
    else:
        print(sizeR)
    centerR = np.int0((rect[(0)]))
#   box = np.int0(cv2.boxPoints(rect))
#   cv2.drawContours(img,[box],-1 ,(255,0,0),5)
#   show1(eroze,'Binarni obraz')
#   show1(img,'Vykreslene hrany')
    M = cv2.getRotationMatrix2D((rect[(0)]),(rect[(2)]),1)
    rotate = cv2.warpAffine(np.float64(blue),M,(size[0],size[1]))
    rotate_rgb = cv2.warpAffine(np.float64(img),M,(size[0],size[1]))
#   show1(rotate,'Rotace')
    cut = rotate[np.int0((centerR[1])-(size_cut[1])/2):np.int0((centerR[1])+(size_cut[1])/2),\
            np.int0((centerR[0])-(size_cut[0])/2):np.int0((centerR[0])+(size_cut[0])/2)]
    vzorek_rgb = rotate_rgb[np.int0((centerR[1])-(size_cut[1])/2):np.int0((centerR[1])+(size_cut[1])/2),\
                    np.int0((centerR[0])-(size_cut[0])/2):np.int0((centerR[0])+(size_cut[0])/2)]
    if jasmax < np.float64(np.max(cut)) :
        print('nutno zvetsit jasmax, jasMAX = ',np.float64(np.max(cut)))
        print('jasmin',np.float64(np.min(cut)))
    elif jasmin > np.float64(np.min(cut)):
        print('nutno zmensit parametr jasMIN, jasMIN = ',np.float64(np.min(cut)))
        print('jasmax',np.float64(np.max(cut)))
    else:
        print('jasmax',np.float64(np.max(cut)))
        print('jasmin',np.float64(np.min(cut)))
    rozptyl = (jasmax-jasmin)
    prumer = np.float64((jasmax+jasmin))/2
    equalized = ((((np.float64(cut)-prumer)/rozptyl)+0.5)*255)
    equalized = boundaries(equalized)
#   histogram(equalized)
    return  equalized,vzorek_rgb


def vnitrek_vzorku(image):
    prahovani = cv2.threshold(image, 150, 255, cv2.THRESH_BINARY_INV)[1].astype(np.uint8)
    dilatace = cv2.dilate(prahovani, cv2.getStructuringElement(cv2.MORPH_RECT, (10, 10)))
    eroze = cv2.erode(dilatace, cv2.getStructuringElement(cv2.MORPH_RECT, (40, 40)))
    return eroze
```

```
def tvar(image1,rozmer,r):
    image=image1+0.1
    prahovani = cv2.threshold(image, 150, 255, cv2.THRESH_BINARY_INV)[1].astype(np.uint8)
    dilatace = cv2.dilate(prahovani, cv2.getStructuringElement(cv2.MORPH_RECT, (10, 10)))
    eroze = cv2.erode(dilatace, cv2.getStructuringElement(cv2.MORPH_RECT, (13, 13)))
    canny = cv2.Canny(eroze,1,20,5)
    T = 40
    t = 3
    counter = [0,0,0,0,0,0,0,0]
    odchylka = [0,0,0,0,0,0,0,0]
    error = [0,0,0,0,0,0,0,0]
    for (y,x), value in np.ndenumerate(image):
        if canny[y,x] == 255:
            if x > size_cut[0]/2 - rozmer[0]/2 + r and x < size_cut[0]/2 + rozmer[0]/2 - r  \
                    and y > size_cut[1]/2 - rozmer[1]/2 - T and y < size_cut[1]/2 - rozmer[1]/2 + T:
                if np.abs(y - (size_cut[1]/2 - rozmer[1]/2)) > t:
                    odchylka[0] = odchylka[0] + (y - (size_cut[1]/2 - rozmer[1]/2))**2 - t**2
                    image[y,x] = 0
                counter[0] = counter[0] + 1
            elif x > size_cut[0]/2 - rozmer[0]/2 + r and x < size_cut[0]/2 + rozmer[0]/2 - r \
                    and y > size_cut[1]/2 + rozmer[1]/2 - T and y < size_cut[1]/2 + rozmer[1]/2 + T:
                if np.abs(y - (size_cut[1]/2 + rozmer[1]/2)) > t:
                    odchylka[1] = odchylka[1] + (y - (size_cut[1]/2 + rozmer[1]/2))**2 - t**2
                    image[y,x] = 0
                counter[1] = counter[1] + 1
            elif y > size_cut[1]/2 - rozmer[1]/2 + r and y < size_cut[1]/2 + rozmer[1]/2 - r \
                    and x > size_cut[0]/2 - rozmer[0]/2 - T and x < size_cut[0]/2 - rozmer[0]/2 + T:
                if np.abs(x - (size_cut[0]/2 - rozmer[0]/2)) > t:
                    odchylka[2] = odchylka[2] + (x - (size_cut[0]/2 - rozmer[0]/2))**2 - t**2
                    image[y,x] = 0
                counter[2] = counter[2] + 1
            elif y > size_cut[1]/2 - rozmer[1]/2 + r and y < size_cut[1]/2 + rozmer[1]/2 - r \
                    and x > size_cut[0]/2 + rozmer[0]/2 - T and x < size_cut[0]/2 + rozmer[0]/2 + T:
                if np.abs(x - (size_cut[0]/2 + rozmer[0]/2)) > t:
                    odchylka[3] = odchylka[3] + (x - (size_cut[0]/2 + rozmer[0]/2))**2 - t**2
                    image[y,x] = 0
                counter[3] = counter[3] + 1
            elif ((x - (size_cut[0]/2 - rozmer[0]/2 + r))**2 + (y - (size_cut[1]/2 - rozmer[1]/2 + r))**2)**0.5 < r+T \
```

```
            and ((x - (size_cut[0]/2 - rozmer[0]/2 + r))**2 + (y - (size_cut[1]/2 - rozmer[1]/2 + r))**2)**0.5 > r-T:
            if np.abs((((x - (size_cut[0]/2 - rozmer[0]/2 + r))**2 + (y - (size_cut[1]/2 - rozmer[1]/2 + r))**2)**0.5 - r)) > t:
                odchylka[4] = odchylka[4] + ((((x - (size_cut[0]/2 - rozmer[0]/2 + r))**2 + \
                    (y - (size_cut[1]/2 - rozmer[1]/2 + r))**2)**0.5 - r))**2 - t**2
                image[y,x] = 0
            counter[4] = counter[4] + 1
        elif ((x - (size_cut[0]/2 + rozmer[0]/2 - r))**2 + (y - (size_cut[1]/2 - rozmer[1]/2 + r))**2)**0.5 < r+T \
            and ((x - (size_cut[0]/2 + rozmer[0]/2 - r))**2 + (y - (size_cut[1]/2 - rozmer[1]/2 + r))**2)**0.5 > r-T:
            if np.abs((((x - (size_cut[0]/2 + rozmer[0]/2 - r))**2 + (y - (size_cut[1]/2 - rozmer[1]/2 + r))**2)**0.5 - r)) > t:
                odchylka[5] = odchylka[5] + ((((x - (size_cut[0]/2 + rozmer[0]/2 - r))**2 + \
                    (y - (size_cut[1]/2 - rozmer[1]/2 + r))**2)**0.5 - r))**2 - t**2
                image[y,x] = 0
            counter[5] = counter[5] + 1
        elif ((x - (size_cut[0]/2 + rozmer[0]/2 - r))**2 + (y - (size_cut[1]/2 + rozmer[1]/2 - r))**2)**0.5 < r+T \
            and ((x - (size_cut[0]/2 + rozmer[0]/2 - r))**2 + (y - (size_cut[1]/2 + rozmer[1]/2 - r))**2)**0.5 > r-T:
            if np.abs((((x - (size_cut[0]/2 + rozmer[0]/2 - r))**2 + (y - (size_cut[1]/2 + rozmer[1]/2 - r))**2)**0.5 - r)) > t:
                odchylka[6] = odchylka[6] + ((((x - (size_cut[0]/2 + rozmer[0]/2 - r))**2 + \
                    (y - (size_cut[1]/2 + rozmer[1]/2 - r))**2)**0.5 - r))**2 - t**2
                image[y,x] = 0
            counter[6] = counter[6] + 1
        elif ((x - (size_cut[0]/2 - rozmer[0]/2 + r))**2 + (y - (size_cut[1]/2 + rozmer[1]/2 - r))**2)**0.5 < r+T \
            and ((x - (size_cut[0]/2 - rozmer[0]/2 + r))**2 + (y - (size_cut[1]/2 + rozmer[1]/2 - r))**2)**0.5 > r-T:
            if np.abs((((x - (size_cut[0]/2 - rozmer[0]/2 + r))**2 + (y - (size_cut[1]/2 + rozmer[1]/2 - r))**2)**0.5 - r)) > t:
                odchylka[7] = odchylka[7] + ((((x - (size_cut[0]/2 - rozmer[0]/2 + r))**2 + \
                    (y - (size_cut[1]/2 + rozmer[1]/2 - r))**2)**0.5 - r))**2 - t**2
                image[y,x] = 0
            counter[7] = counter[7] + 1
        else:
            print('hrana mimo okraj')
            image[y,x] = 255
for i in range(0,2):
    error[i] = odchylka[i] / counter[i]
    if error[i] > 15:
        print('neproslo kontrolou tvaru ',[i],'odchylka', error[i])
    else:
        print('kontrola tvaru OK',[i],'odchylka', error[i])

for i in range(2,4):
```

```
        error[i] = odchylka[i] / counter[i]
        if error[i] > 50:
            print('neproslo kontrolou tvaru ',[i],'odchylka', error[i])
        else:
            print('kontrola tvaru OK',[i],'odchylka', error[i])
    show(image,'image')


    for i in range(4,8):
        error[i] = odchylka[i] / counter[i]
        if error[i] > 100:
            print('neproslo kontrolou tvaru ',[i],'odchylka', error[i])
        else:
            print('kontrola tvaru OK',[i],'odchylka', error[i])
    show(image,'image')


#
def fourier(image):
    dft = cv2.dft(np.float32(image),flags = cv2.DFT_COMPLEX_OUTPUT)
    dft_shift = np.fft.fftshift(dft)
#   show(image,'vzorek')
    dft_shift1 = dft_shift+0
    dft_shift2 = dft_shift+0
#   cv2.imwrite("vzorek.jpg",vzorek);
    for (x,y,z), value in np.ndenumerate(dft_shift):
        if np.abs(dft_shift[x,y,z])>10000000/(1+(36*(x-(size_cut[1])/2)**2 + (y-(size_cut[0])/2)**2)**0.5)\
            and (36*(x-(size_cut[1])/2)**2 + (y-(size_cut[0])/2)**2)**0.5>60:
            if x == (size_cut[1])/2 and y == (size_cut[0])/2:
                dft_shift1[x,y,z] = dft_shift[x,y,z]
            else:
                dft_shift1[x,y,z] = 0
        else:
            if x == (size_cut[1])/2 and y == (size_cut[0])/2:
                dft_shift2[x,y,z] = dft_shift[x,y,z]
            else:
                dft_shift2[x,y,z] = 0
    magnitude_spectrum1 = 10*np.log(1+cv2.magnitude(dft_shift1[:,:,0],dft_shift1[:,:,1]))
    show(magnitude_spectrum1,'Fourierovo spektrum s filtrem1')
#   cv2.imwrite( "spectrum21.jpg", magnitude_spectrum1);
```

```
    f_ishift1 = np.fft.ifftshift(dft_shift1)
    img_back1 = cv2.idft(f_ishift1, flags=cv2.DFT_SCALE | cv2.DFT_REAL_OUTPUT)
    img_back1 = boundaries(img_back1)
    show(img_back1,'Filtrovany obraz1')
#    cv2.imwrite( "FFT21.jpg", img_back1 );

#    magnitude_spectrum2 = 10*np.log(1+cv2.magnitude(dft_shift2[:,:,0],dft_shift2[:,:,1]))
#    show(magnitude_spectrum2,'Fourierovo spektrum s filtrem2')
#    cv2.imwrite( "spectrum22.jpg", magnitude_spectrum2);
    f_ishift2 = np.fft.ifftshift(dft_shift2)
    img_back2 = cv2.idft(f_ishift2, flags=cv2.DFT_SCALE | cv2.DFT_REAL_OUTPUT)
    img_back2 = boundaries(img_back2)
    show(img_back2,'Filtrovany obraz2')
#    cv2.imwrite( "FFT22.jpg", img_back2 );
    return img_back1

def fourier2(image):
    dft = cv2.dft(np.float32(image),flags = cv2.DFT_COMPLEX_OUTPUT)
    dft_shift = np.fft.fftshift(dft)
#    show(image,'vzorek')
    dft_shift1 = dft_shift.copy()
    dft_shift2 = dft_shift.copy()
#    cv2.imwrite("vzorek.jpg",vzorek);
    for (x,y,z), value in np.ndenumerate(dft_shift):
        if np.abs(dft_shift[x,y,z])>10000000/(1+(18*(x-(size_cut[1])/2)**2 + (y-(size_cut[0])/2)**2)**0.5):
#            if x == (size_cut[1])/2 and y == (size_cut[0])/2:
#                dft_shift1[x,y,z] = dft_shift[x,y,z]
#            else:
                dft_shift1[x,y,z] = 0
        else:
            if x == (size_cut[1])/2 and y == (size_cut[0])/2:
                dft_shift2[x,y,z] = dft_shift[x,y,z]
            else:
                dft_shift2[x,y,z] = 0
#    magnitude_spectrum1 = 10*np.log(1+cv2.magnitude(dft_shift1[:,:,0],dft_shift1[:,:,1]))
#    show(magnitude_spectrum1,'Fourierovo spektrum s filtrem1')
#    cv2.imwrite( "spectrum21.jpg", magnitude_spectrum1);
    f_ishift1 = np.fft.ifftshift(dft_shift1)
```

```
    img_back1 = cv2.idft(f_ishift1, flags=cv2.DFT_SCALE | cv2.DFT_REAL_OUTPUT)
    img_back1 = boundaries(img_back1)
    show(img_back1,'Filtrovany obraz3')
#    cv2.imwrite( "FFT21.jpg", img_back1 );

#    magnitude_spectrum2 = 10*np.log(1+cv2.magnitude(dft_shift2[:,:,0],dft_shift2[:,:,1]))
#    show(magnitude_spectrum2,'Fourierovo spektrum s filtrem2')
#    cv2.imwrite( "spectrum22.jpg", magnitude_spectrum2);
    f_ishift2 = np.fft.ifftshift(dft_shift2)
    img_back2 = cv2.idft(f_ishift2, flags=cv2.DFT_SCALE | cv2.DFT_REAL_OUTPUT)
    img_back2 = boundaries(img_back2)
#    show(img_back2,'Filtrovany obraz2')
#    cv2.imwrite( "FFT22.jpg", img_back2 );
    return img_back1

#def pca(image):
#    pca = PCA()
#    pca.fit(image)
#    axes = pca.components_[:,:]
#    comp = pca.transform(image)[:,:]
#    X1 = comp[:,:pocet_komponent] @ axes[:pocet_komponent,:]
#    X1 += + np.mean(image)
#    X1 = boundaries(X1)
#    show(X1,'PCA')
#    return X1,comp,axes


def pca(image):
    pca = PCA()
    pca.fit(image.T)
    X1 = pca.transform(image.T)[:,pocet_komponent:] @ pca.components_[pocet_komponent:,:]
    X1 += + np.mean(image.T)
    rozptyl = (np.max(X1)-np.min(X1))-50
    prumer = np.float64((np.max(X1)+np.min(X1)))/2
    X2 = ((((X1-prumer)/rozptyl)+0.5)*255)
    X2 = boundaries(X2)
#    show(X1.T,'PCA')
#    cv2.imwrite( "pcaT50_s.jpg", X2.T );
```

```
    return X2.T

def pca2(image):
    X = image - np.mean(image, axis=0)
    C = np.cov(X, rowvar=False)
    L, V1 = la.eig(C)
    V1 = V1.real
    poradi = L.argsort()[::-1]
    L, V1 = L[poradi], V1[:, poradi]
    V = V1[0:size_cut[0], 0:size_cut[1]]
    Z = X @ V
    X1 = Z[:,:pocet_komponent] @ V.T[:pocet_komponent,:]
    X1 += + np.mean(image)
    X1 = boundaries(X1)
#    show(X1,'PCA2')
    return Z,V1,L,X1,C

def svd(image):
    X = image - np.mean(image, axis=0)
    U, s, Vt = la.svd(X, full_matrices=False)
    V = Vt.T
    Z = X @ V
    X1 = Z[:,:pocet_komponent] @ Vt[:pocet_komponent,:]
    X1 += + np.mean(image)
    rozptyl = (np.max(X1)-np.min(X1))-50
    prumer = np.float64((np.max(X1)+np.min(X1)))/2
    X2 = ((((X1-prumer)/rozptyl)+0.5)*255)
    X2 = boundaries(X2)
#    cv2.imwrite( "pca_bezchyb.jpg", X2 );
#    show(X2,'SVD1')
    screeplot(s)
    return X2,V,Z,X,U,s

def konvoluce(image):
#    filter_kernel_5 = [[-1, -1, -1, -1, -1, -1, -1, -1, -1],
#                       [-1, -1, -1,  0,  0,  0, -1, -1, -1],
#                       [-1, -1,  0,  2,  2,  2,  0, -1, -1],
#                       [-1,  0,  2,  2,  2,  2,  2,  0, -1],
```

```
#                 [-1,  0,  2,  2,  2,  2,  2,  0, -1],
#                 [-1,  0,  2,  2,  2,  2,  2,  0, -1],
#                 [-1, -1,  0,  2,  2,  2,  0, -1, -1],
#                 [-1, -1, -1,  0,  0,  0, -1, -1, -1],
#                 [-1, -1, -1, -1, -1, -1, -1, -1, -1],]


  filter_kernel_7 = [[-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],
                     [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],
                     [-1, -1, -1,  0,  2,  2,  2,  0, -1, -1, -1],
                     [-1, -1,  0,  2,  2,  2,  2,  2,  0, -1, -1],
                     [-1, -1,  2,  2,  2,  2,  2,  2,  2, -1, -1],
                     [-1, -1,  2,  2,  2,  2,  2,  2,  2, -1, -1],
                     [-1, -1,  2,  2,  2,  2,  2,  2,  2, -1, -1],
                     [-1, -1,  0,  2,  2,  2,  2,  2,  0, -1, -1],
                     [-1, -1, -1,  0,  2,  2,  2,  0, -1, -1, -1],
                     [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],
                     [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],]
#
#   filter_kernel_9 = [[-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],
#                      [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],
#                      [-1, -1, -1, -1, -1,  2,  2,  2, -1, -1, -1, -1, -1],
#                      [-1, -1, -1, -1,  2,  2,  2,  2,  2, -1, -1, -1, -1],
#                      [-1, -1, -1,  2,  2,  2,  2,  2,  2,  2, -1, -1, -1],
#                      [-1, -1,  2,  2,  2,  2,  2,  2,  2,  2,  2, -1, -1],
#                      [-1, -1,  2,  2,  2,  2,  2,  2,  2,  2,  2, -1, -1],
#                      [-1, -1,  2,  2,  2,  2,  2,  2,  2,  2,  2, -1, -1],
#                      [-1, -1, -1,  2,  2,  2,  2,  2,  2,  2, -1, -1, -1],
#                      [-1, -1, -1, -1,  2,  2,  2,  2,  2, -1, -1, -1, -1],
#                      [-1, -1, -1, -1, -1,  2,  2,  2, -1, -1, -1, -1, -1],
#                      [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],
#                      [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],]


#   kapky5 = scipy.signal.convolve2d(image, filter_kernel_5,mode='same', boundary='fill', fillvalue=0)
#   kapky5 = ((((kapky5-(np.float64((np.max(kapky5)+np.min(kapky5)))/2))/(np.max(kapky5)-np.min(kapky5)) )+0.5)*255)
#   kapky5 = boundaries(kapky5)
#   show(kapky5,'kapky5')
```

```
    kapky7 = scipy.signal.convolve2d(image, filter_kernel_7,mode='same', boundary='fill', fillvalue=0)
#    kapky7 = ((((kapky7-(np.float64((np.max(kapky7)+np.min(kapky7)))/2))/(np.max(kapky7)-np.min(kapky7)) )+0.5)*255)
    kapky7 = 0.05*kapky7
    kapky7 = boundaries(kapky7)
#    show(kapky7,'kapky7')

#    kapky9 = scipy.signal.convolve2d(image, filter_kernel_9,mode='same', boundary='fill', fillvalue=0)
#    kapky9 = ((((kapky9-(np.float64((np.max(kapky9)+np.min(kapky9)))/2))/(np.max(kapky9)-np.min(kapky9)) )+0.5)*255)
#    kapky9 = boundaries(kapky9)
#    show(kapky9,'kapky9')
#    cv2.imwrite( "konvoluce9.jpg", kapky9 );
    return kapky7

def tenka_vrstva(fourier,image,rgb):
    rozmazani = cv2.blur(np.uint8(fourier), (20,20))
#    show(rozmazani,'rozmazani')
    vnitrek = vnitrek_vzorku(image)
    prah = cv2.threshold(rozmazani, 100, 255, cv2.THRESH_BINARY)[1].astype(np.uint8)
    prah = cv2.erode(prah, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5)))
    prah = cv2.dilate(prah, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (10, 10)))
#    show(vnitrek,'vnitrek')
#    show(prah,'prah')
    counter = 0
    for (y,x), value in np.ndenumerate(image):
        if vnitrek[y,x] == 255:
            if prah[y , x] == 255:
                vnitrek[y,x] = 255
                rgb[y,x,:] = (185,174,255)
                counter = counter + 1
            else:
                vnitrek[y,x] = 0
#    show(vnitrek,'tenka vrstva')
#    cv2.imwrite( "tenka_vrstva.jpg", vnitrek );
    if counter > 500:
        print('neproslo kontrolou tloustky vrstvy', counter)
    else:
        print('tloustka vrstvy OK', counter)
    return rgb
```

```
def dvojita_vrstva(image,rgb):
    stred_histogramu = np.median(image)
    print(stred_histogramu)
    prahovani = cv2.threshold(image, stred_histogramu - 50 , 255, cv2.THRESH_BINARY_INV)[1].astype(np.uint8)
    dilatace = cv2.dilate(prahovani, cv2.getStructuringElement(cv2.MORPH_RECT, (40, 40)))
    eroze = cv2.erode(dilatace, cv2.getStructuringElement(cv2.MORPH_RECT, (35, 35)))
#    show(eroze,'eroze')
#    show(prahovani,'eroze1')
    prahovani, contours, hierarchy = cv2.findContours(eroze, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    for cont in contours:
        rect = cv2.minAreaRect(cont)
        box = np.int0(cv2.boxPoints(rect))
        cv2.drawContours(rgb,[box],-1 ,(255,255,0),3)
#    counter = 0
#    for (y,x), value in np.ndenumerate(image):
#        if prahovani[y,x] == 255:
#            counter = counter + 1
#    if counter > 100:
#        print('neproslo kontrolou dvojite vrstvy', counter)
#    else:
#        print('kontrola dvojite vrstvy OK', counter)
    return rgb


def kapicky(vzorek,pca,fourier,konvoluce,rgb):
    counter = 0
    prahovani = cv2.threshold(pca,  30 , 255, cv2.THRESH_BINARY)[1].astype(np.uint8)
    prahovani = cv2.erode(prahovani, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3)))
    prahovani = cv2.dilate(prahovani, cv2.getStructuringElement(cv2.MORPH_RECT, (13, 13)))
    vnitrek = vnitrek_vzorku(vzorek)
    for (y,x), value in np.ndenumerate(pca):
        if vnitrek[y,x] == 0:
            prahovani[y,x] = 0
#    show(prahovani,'prahovani1')
    prahovani, contours, hierarchy = cv2.findContours( prahovani, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    for cont in contours:
        rect = cv2.minAreaRect(cont)
        box = np.int0(cv2.boxPoints(rect))
```

```
        cv2.drawContours(rgb,[box],-1 ,(255,0,0),2)
        counter = counter + 1
    if counter > 20:
        print('neproslo kontrolou kapicek (PCA,modra) ', counter)
    else:
        print('kontrola kapicek OK PCA,modra) ', counter)
    counter = 0

    prahovani = cv2.threshold(fourier,  40 , 255, cv2.THRESH_BINARY)[1].astype(np.uint8)
    prahovani = cv2.erode(prahovani, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3)))
    prahovani = cv2.dilate(prahovani, cv2.getStructuringElement(cv2.MORPH_RECT, (9, 9)))
    for (y,x), value in np.ndenumerate(pca):
        if vnitrek[y,x] == 0:
            prahovani[y,x] = 0
#   show(prahovani,'prahovani2')
    prahovani, contours, hierarchy = cv2.findContours( prahovani, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    for cont in contours:
        rect = cv2.minAreaRect(cont)
        box = np.int0(cv2.boxPoints(rect))
        cv2.drawContours(rgb,[box],-1 ,(0,255,0),2)
        counter = counter + 1
    if counter > 20:
        print('neproslo kontrolou kapicek (Fourier,zelena) ', counter)
    else:
        print('kontrola kapicek OK (Fourier,zelena) ', counter)
    counter = 0

    prahovani = cv2.threshold(konvoluce,  100 , 255, cv2.THRESH_BINARY)[1].astype(np.uint8)
    prahovani = cv2.dilate(prahovani, cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5)))
    for (y,x), value in np.ndenumerate(pca):
        if vnitrek[y,x] == 0:
            prahovani[y,x] = 0
#   show(prahovani,'prahovani3')
    prahovani, contours, hierarchy = cv2.findContours( prahovani, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    for cont in contours:
        rect = cv2.minAreaRect(cont)
        box = np.int0(cv2.boxPoints(rect))
        cv2.drawContours(rgb,[box],-1 ,(0,0,255),2)
```

```
        counter = counter + 1
    if counter > 20:
        print('neproslo kontrolou kapicek (Konvoluce,cervena) ', counter)
    else:
        print('kontrola kapicek OK (Konvoluce,cervena) ', counter)
    return rgb


def okraje(image,rgb):
    prahovani = cv2.threshold(image, 150, 255, cv2.THRESH_BINARY_INV)[1].astype(np.uint8)
    odebraniR = cv2.dilate(prahovani, cv2.getStructuringElement(cv2.MORPH_RECT, (8, 8)))
    odebraniR = cv2.erode(odebraniR, cv2.getStructuringElement(cv2.MORPH_RECT, (40, 40)))
    odebraniR = cv2.dilate(odebraniR, cv2.getStructuringElement(cv2.MORPH_RECT, (32, 32)))
#    cv2.imwrite("hrubo1.jpg",odebraniR);
    pridaniR = cv2.dilate(prahovani, cv2.getStructuringElement(cv2.MORPH_RECT, (40, 40)))
    pridaniR = cv2.erode(pridaniR, cv2.getStructuringElement(cv2.MORPH_RECT, (50, 50)))
    pridaniR = cv2.dilate(pridaniR, cv2.getStructuringElement(cv2.MORPH_RECT, (10, 10)))
#    cv2.imwrite("hrubo2.jpg",pridaniR);
    odebraniC = cv2.dilate(prahovani, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (8, 8)))
    odebraniC = cv2.erode(odebraniC, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (40, 40)))
    odebraniC = cv2.dilate(odebraniC, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (32, 32)))
#    cv2.imwrite("hrubo12.jpg",odebraniC);
    pridaniC = cv2.dilate(prahovani, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (40, 40)))
    pridaniC = cv2.erode(pridaniC, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (50, 50)))
    pridaniC = cv2.dilate(pridaniC, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (10, 10)))
#    cv2.imwrite("hrubo3.jpg",pridaniC);


    detekceR =  np.abs(np.float32(odebraniR) - np.float32(pridaniR))
    detekceC =  np.abs(np.float32(odebraniC) - np.float32(pridaniC))
#    show(detekce1,'detekce1')
#    show(detekce2,'detekce2')


    detekceR = cv2.threshold(detekceR,  120 , 255, cv2.THRESH_BINARY)[1].astype(np.uint8)
    detekceR = cv2.erode(detekceR, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5)))
    detekceR = cv2.dilate(detekceR, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (40, 40)))
    detekceR = cv2.erode(detekceR, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (35, 35)))
    detekceR, contours, hierarchy = cv2.findContours(detekceR, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    for cont in contours:
        rect = cv2.minAreaRect(cont)
```

```
        box = np.int0(cv2.boxPoints(rect))
        cv2.drawContours(rgb,[box],-1 ,(57,76,139),4)
    error = False
    counter1 = 0
    for (y,x), value in np.ndenumerate(image):
        if detekceR[y , x] == 255:
            counter1 = counter1 + 1
    if counter1 > 2000:
        print('neproslo kontrolou okraju (ctvercove jadro, hneda) ', counter1)
        error = True
    else:
        print('kontrola okraju OK (ctvercove jadro, hneda) ', counter1)

    detekceC = cv2.threshold(detekceC,  120 , 255, cv2.THRESH_BINARY)[1].astype(np.uint8)
    detekceC = cv2.erode(detekceC, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (4, 4)))
    detekceC = cv2.dilate(detekceC, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (40, 40)))
    detekceC = cv2.erode(detekceC, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (35, 35)))
    detekceC, contours, hierarchy = cv2.findContours(detekceC, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    for cont in contours:
        rect = cv2.minAreaRect(cont)
        box = np.int0(cv2.boxPoints(rect))
        cv2.drawContours(rgb,[box],-1 ,(0,255,255),4)

    counter2 = 0
    for (y,x), value in np.ndenumerate(image):
        if detekceC[y , x] == 255:
            counter2 = counter2 + 1
    if counter2 > 1000:
        print('neproslo kontrolou okraju (kruhové jadro, zluta) ', counter2)
        error = True
    else:
        print('kontrola okraju OK (kruhové jadro, zluta) ', counter2)
#   show(rgb,'rgb')
    return rgb,error

# ======================================================================
# KONSTANTY VÝPOČTU
# ======================================================================
```

```
size_cut = [2400,400]
jasmax = 200
jasmin = 30
pocet_komponent = 30
rozmer = [2287,338]
polomer_rohu = 80

# ================================================================================
# ANALYZA OBRAZU - vlastní tělo programu
# ================================================================================

vstup = cv2.imread('1.jpg')
vzorek,vzorek_rgb = segmentace(vstup)

fourier = fourier(vzorek)
fourier2 = fourier2(vzorek)
##svd,V,Z,X,U,s = svd(vzorek)
pca = pca(vzorek)
##principal_components, principal_axes,l,pca2,C = pca2(vzorek)
#
konvoluce = konvoluce(vzorek)

vzorek_rgb = tenka_vrstva(fourier,vzorek,vzorek_rgb)
vzorek_rgb = dvojita_vrstva(vzorek,vzorek_rgb)
vzorek_rgb = kapicky(vzorek,pca,fourier2,konvoluce,vzorek_rgb)
vzorek_rgb,error = okraje(vzorek,vzorek_rgb)
if error == False:
   tvar(vzorek,rozmer,polomer_rohu)

show(vzorek,'Vzorek')
show(vzorek_rgb,'vzorek_rgb')
#cv2.imwrite("vzorek.jpg",vzorek);
#cv2.imwrite("rgb.jpg",vzorek_rgb);

cv2.waitKey(0)
cv2.destroyAllWindows()
```