České vysoké učení technické v Praze
Fakulta biomedicínského inženýrství
Katedra biomedicínské informatiky

Knihovna funkcí pro zpracování ultrazvukových vyšetření

Software library for ultrasound image processing

Bakalářská práce

Autor bakalářské práce: Jakub Dušek
Vedoucí bakalářské práce: Mgr. Radim Krupička, Ph.D.

Studijní program: Biomedicínská a klinická technika

Studijní obor: Biomedicínská informatika

18. května 2017

Na tomto místě bude oficiální zadání vaší práce

- Toto zadání je podepsané děkanem a vedoucím katedry,
- musíte si ho vyzvednout na studiijním oddělení,
- v jedné odevzdané práci bude originál tohoto zadání (originál zůstává po obhajobě na katedře),
- ve druhé bude na stejném místě neověřená kopie tohoto dokumentu (tato se vám vrátí po obhajobě).

Prohlášení

Prohlašuji, že jsem bakalářskou práci s názvem „Knihovna funkcí pro zpracování ultrazvukových vyšetření“ vypracoval samostatně. Veškerou použitou literaturu a podkladové materiály uvádím v přiloženém seznamu literatury.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů.

V Kladně 12.5.2017 ..... Jakub Dušek

Poděkování

Rád bych touto cestou vyjádřil poděkování Mgr. Radimu Krupičkovi, Ph.D. za vstřícnost a pomoc při vedení a získávání potřebných informací a podkladů. Dále bych chtěl poděkovat doc. Ing. Zoltánu Szabóvi, Ph.D. za poskytnutí informací, podkladů a uvedení do problematiky.

Abstrakt

Knihovna funkcí pro zpracování ultrazvukových vyšetření

Práce pojednává o návrhu a implementaci knihovny pro zpracování ultrazvukových vyšetření. V úvodu popisuje základní princip a zobrazení ultrazvukových vyšetření, popisem periferního cévního onemocnění a možnou diagnostikou pomocí sledování heterogenity prostředí. Na základě pozorování a vlastností ultrazvukových snímků následuje popis a implementace knihovny pro texturální analýzu a příznaky konstrastu, homogenity, energie, entropie a korelace. Dále jsou popsány možnosti napojení knihovny na prostředí MATLAB. V závěru jsou pak proti sobě otestovány výsledky z prostředí MATLAB a samotné knihovny.

Klíčová slova

Ultrazvuk; Zpracování obrazu; Periferní cévní onemocnění, MATLAB, OpenCV;

Abstract

Software library for ultrasound image processing

The aim of this work is implementation of a library for processing results of medical ultrasound. In the introduction, it describes basic principles of ultrasonic screening, peripheral artery disease and possible diagnosis by monitoring the heterogeneity of surrounding muscles. Based on the observation and properties of ultrasonic images follows the description and implementation of the library for textural analysis and properties of contrast, homogeneity, energy, entropy and correlation. The possibilities of linking the library to the MATLAB environment are described below. At the end, the results from the MATLAB environment and the library itself are tested against each other.

Keywords

Sonography; Image processing; Peripheral Artery Disease, MATLAB, OpenCV;

Obsah

Seznam symbolů a zkratek

PAD	Peripherial Artery Disease
ABI	Ankle-Brachial Index, Index kotník-paže
SDK	Software Development Kit, vývojové prostředí
GLCM	Gray-Level Co-ocurrency matrix, šedotónová ko-okurenční matice

Kapitola 1

Úvod

Ultrazvuková vyšetření jsou v dnešní době oblíbenou a neinvazivní diagnostickou metodou. Nejrozšířenější využití ultrazvuku je asi těhotenské sono, které nám v průběhu těhotenství může prozradit, jestli se plod správně vyvíjí. Na trh se pomalu dostávají produkty, které přinášejí ultrazvuková zařízení ke koncovým uživatelům, například propojení s chytrým telefonem.

Dalo by se tak očekávat, že při takovém rozšíření této diagnostické metody, bude existovat škála otevřených řešení, na kterých může kdokoliv stavět, ale v současnosti není žádná otevřená knihovna, která by se zabývala zpracováním takových snímků. Jednou z možností zpracování ultrazvukových snímků, nikoliv však nejlepší, je použití prostředí MATLAB, který je v akademické sféře velmi rozšířen. Ale prostředí MATLABU není vhodné pro použití v aplikacích třetích stran a jeho kód není tak otevřený, jak by bylo zapotřebí.

Tato bakalářská práce se tak zabývá implementací otevřené knihovny pro zpracování ultrazvukových snímků, konkrétně snímků pacientů postižených periferním cévním onemocněním, zkráceně PAD, kterým se zabývá druhá kapitola.

Kapitola 2

Přehled současného stavu

Ultrazvuk je druh zvuku, jehož frekvence se pohybuje nad hranicí slyšitelnosti, tedy nad 20 kHz. Má široké spektrum použití, od mechanického čištění povrchů, přes vytváření emulzí až po různá diagnostická použití, mezi která patří asi nejznámější použití ultrazvuku a tím je diagnostická sonografie.

Sonografie je druh ultrazvukového vyšetření, při kterém využíváme schopnosti ultrazvuku procházet různými druhy látkového prostředí s odlišnou rychlostí. Na každém rozhraní jednotlivých prostředí tak dojde k částečnému odrazu, při kterém část ultrazvukové vlny přejde do druhého prostředí a část se odrazí zpátky k vysílači. Tato schopnost se nazývá *echogenita* a rozdělujeme ji na několik stupňů, podle toho, jak se zkoumaná oblast liší od svého okolí:

Hyperechogenita - oblast je jasná, tkáň má vysoký stupeň odrazu a jen nepatrná část ultrazvukové vlny projde skrz, jedná se například o kosti nebo konkrementy[1]

Hypoechogenita - oblast s nízkou úrovní jasu, převládající šedé pixely, jedná se o měkké tkáně, skrz které projde velká část ultrazvukové vlny. Jedná se také často o ložiska patologického nálezu.

Anechogenita - oblast je zcela černá, bez odrazu, jedná se například o tekutiny nebo plíce

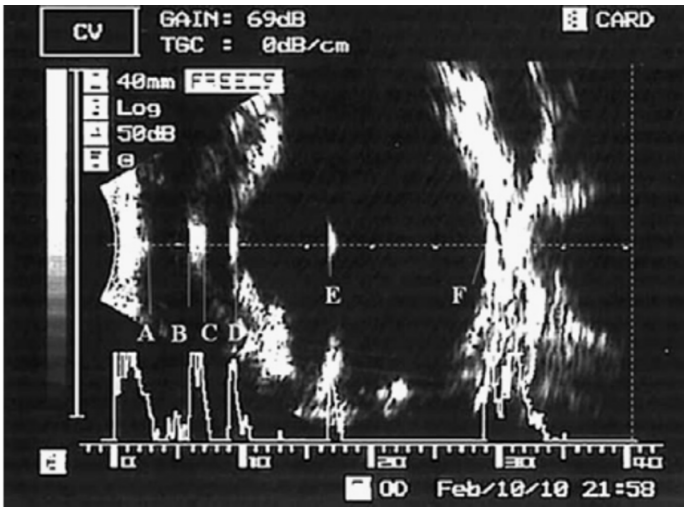
Echogenitu prostředí lze zvýšit aplikací kontrastní látky, často se jedná o mikrobublinky vzduchu uzavřené ve vodiči, biopolymeru.

[1] Kamenům podobné útvary, například ledvinové kameny.

Výsledky ultrazvukových vyšetření lze zobrazit několika způsoby, v tzv. módech:

A-Mode Nejjednodušší zobrazení, na osách x a y je v čase zobrazena intenzita ultrazvukových pulzů, které vycházejí z jednoho zdroje. Možnosti samotného A-Mode zobrazení jsou celkem omezené, ale jsou základem základem pro všechna další zobrazení.

B-Mode V současnosti nejpoužívanější typ zobrazení. Jedná se v podstatě o sérii zobrazení v A-Mode, ale velikost amplitudy je převáděna na šedotónovou hodnotu pixelů. Pro vznik B-Mode je potřeba mnoho snímků z jedné roviny, historicky taková zobrazení vznikala posouváním jednoho zdroje po rovině a výsledné zobrazení se dopočítalo součtem všech měření. V současnosti se snímkování provádí pomocí sondy s velkým množstvím snímačů, výsledný obraz tak vzniká v reálném čase.



Obrázek 2.1: B-Mode zobrazení oka v kombinaci s A-Mode zobrazením.
Zdroj: [4]

M-Mode Obdobný princip zobrazení jako B-Mode, využívá se k zobrazení pohyblivých struktur, například v echokardiografii k zobrazení srdce. Série A-Mode snímků je seřazena za sebe a zobrazena v čase.

Dopplerovské zobrazení Speciální zobrazení, které nám umožňuje zobrazení toku krve cévami pomocí Dopplerova jevu, který popisuje změnu frekvence a vlnové délky v závislosti na pohybu vysílače a přijímače.

2.1 Současné možnosti zpracování UZ snímků

2.1.1 Systém PACS a formát DICOM

V oblasti ukládání a distribuce ultrazvukových snímků v současnosti jasně dominuje systém PACS a obrazový formát DICOM. Systém PACS se skládá ze čtyř hlavních komponent:

- Obrazová dokumentace
- Úložiště dat
- Síť
- Klientské stanice

Obrazová dokumentace představuje samotné snímky, uložené ve formátu DICOM, který umožňuje ukládání nejen obrazových dat, ale také neobrazových dat, jako například informace o pacientovi, původ snímku nebo okolnosti, za kterých byl snímek pořízen. Samotná obrazová část je pak uložena v bezztrátovém formátu, nebo běžně dostupném formátu pro kompresi JPEG.

DICOM ale není jen formát pro ukládání dat, ale hlavně datový standard, který popisuje způsob manipulace s daty, způsob jakým jsou distribuovány po síti a možnosti zapojení i ostatních přístrojů, jako například tiskáren, skenerů a serverů, právě do systému PACS.

Bohužel, jak systém PACS tak DICOM, jsou oba v současnosti zatíženy patenty, proto je v jistých oblastech obtížné jejich použití, například pro komerční využití.

2.1.2 Ultrazvuková řešení

Na trhu se dnes pohybuje spousta firem, které se zabývají vývojem a výrovou komplexních ultrazvukových přístrojů. Ať už se jedná o hotové ultrazvukové přístroje i s počítačem, nebo čistě jen hardware používaný v takových přístrojích. Na poli hotových ultrazvukových přístrojů hrají prim především Toshiba, GE, Samsung a Phillips. Všechny zmíněné poskytují možnost vývoje dodatečného software přes nějaký typ SDK. Nutno podotknout, že v tomto případě nejsou svazováni standardy, jako v případě DICOMu, nejsou tak nijak standardizovány a záleží čistě na výrobci, co do SDK zahrne a co ne. A i když tato SDK mohou být otevřená, jsou chráněna licenčním ujednáním, což opět znesnadňuje jejich další použití[1].

[1] Často jsou tato SDK dostupná pouze pro nekomerční použití

2.2 Periferní cévní onemocnění

Periferní cévní onemocnění, zkráceně PAD[1], je onemocnění cév, při kterém dochází k zužování tepen a sníženému přísunu kyslíku do svalů. Projevuje se bolestivými křečemi, úbytkem svalové, které může vést až k gangréně, a obtížným hojením ran.

Rizikovými faktory pro vznik PAD jsou:

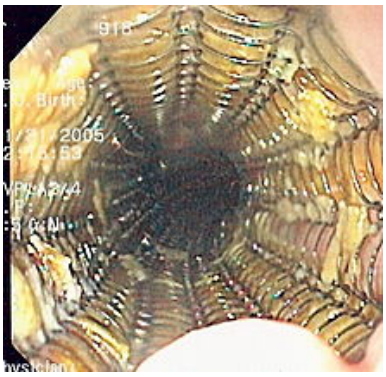
- Cukrovka
- Kouření
- Obezita
- Hypertenze
- Ateroskleróza

Současná léčba pro pacientů zahrnuje snížení počtu rizikových faktorů a užívání léku na ředění krve a vazodilataci, popřípadě chirurgická léčba, angioplastika.

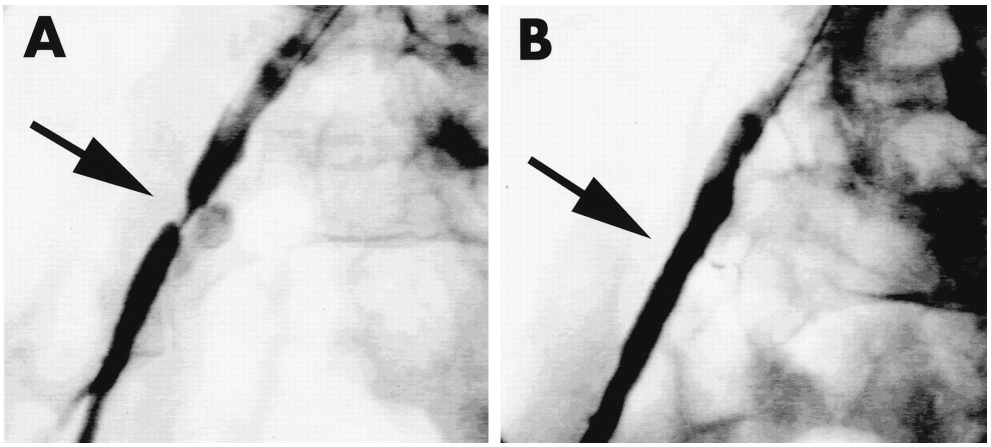
[1]Peripheral Artery Disease

2.2.1 Angioplastika

Jednou z možností léčby PAD je i angioplastika. Jedná se o chirurgický zákrok, při kterém je do zúženého místa v tepně zaveden katétr se stentem, kovovou výztuží, která je pomocí balónkového katétru roztažena a zůžení je opět průchodné. Úspěšnost toho zákroku je téměř 100%.



Obrázek 2.2: Ukázka stentu zavedeného v jícnu.
Zdroj: [5]

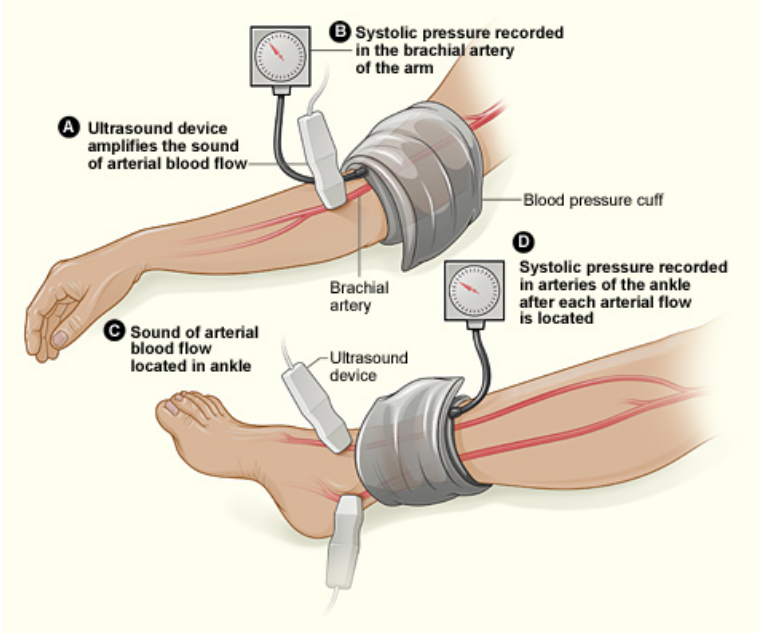


Obrázek 2.3: Angiogram tepny před a po zavedení stentu
Zdroj: [1]

2.2.2 Současné metody diagnostiky

Diagnostikovat PAD lze několika způsoby. Často po jednom vyšetření následuje druhé, které má jen diagnózu potvrdit nebo vyvrátit.

První vyšetření, které pacient podstupuje, je *"Index kotník-paže"*. Vyšetření spočívá v měření krevního tlaku na horních končetinách a kotníku. Volitelně se ještě používá měření dopplerovské sonografie, ale moderní přístroje na měření tlaku pracují i bez něj. Výsledkem je pak rozdíl v tlacích, z jejichž podílu můžeme usoudit, zdali pacient s určitou pravděpodobností trpí PAD nebo ne a přistoupit k dalšímu vyšetření.



Obrázek 2.4: Schéma vyšetření *index kotník-paže*.
Zdroj: [6]

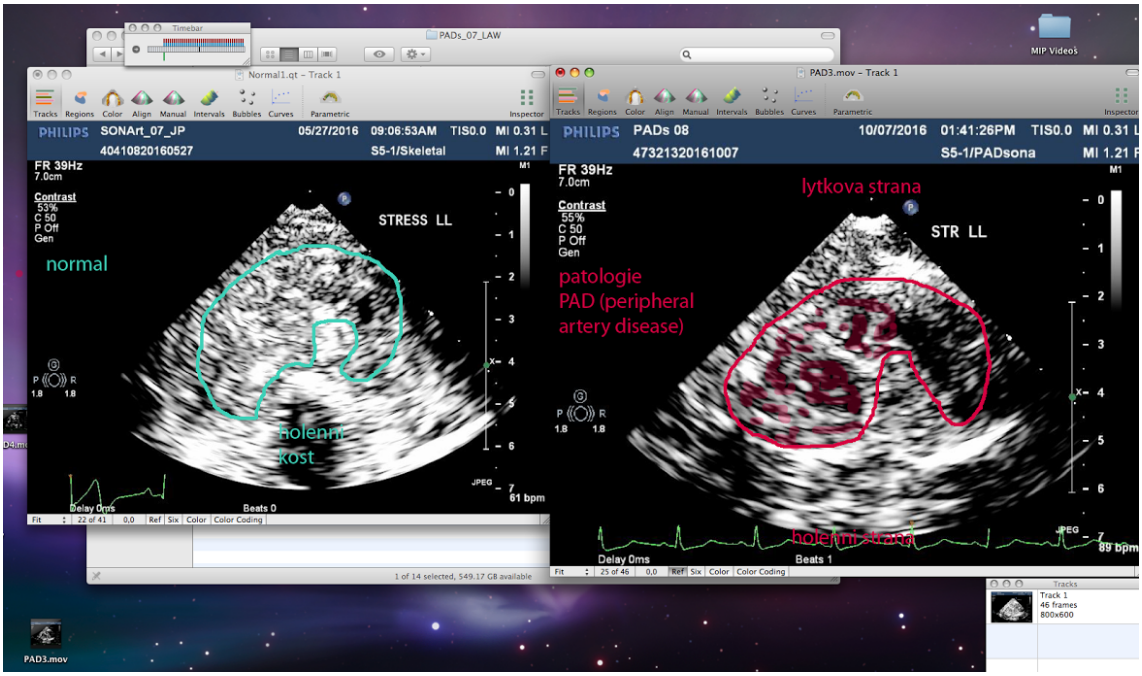
Mezi další vyšetření pak patří různé metody angiografie a to pomocí magnetické rezonance nebo počítačové tomografie.

2.2.3 Index heterogenity

PAD může způsobovat lokální heterogenitu proudění krve, kterou můžeme po aplikaci kontrastní látky sledovat pomocí echokardiografie. Ze vzniklých ultrazvukových snímků tak můžeme vypočítat heterogenitu, nebo homogenitu, ve svalu[1].

Pacientovi je při vyšetření zavedena infúze s kontrastní látkou, vzduchové mikrobublinky, a ultrazvukovou sondou je snímán sval v jedné rovině. Výstupem je série snímků, na jejímž začátku se nachází jeden jasný snímek, který je důsledkem rozpadu vzduchových bublinek vlivem ultrazvuku. Všechny další snímky jsou už tmavší a intenzita jasu se opět zvyšuje postupně s přibývající kontrastní látkou, která do oblasti přitéká, až do chvíle, kdy se úroveň jasu vyrovná. Rychlost, jakou je dosaženo rovnováhy odráží rychlost proudění krve.

Na obrázku 5.1 můžeme pozorovat heterogenitu v tkáni v porovnání se zdravým svalem.



Obrázek 2.5: Vlevo: zdravý sval, Vpravo: sval postižený PAD

Toto vyšetření je nejen rychlé a levné, ale pokud by se prokázala jeho správnost, mohlo by se upustit od snímkování pomocí počítačové tomografie, které produkuje škodlivé rentgenové záření.

[1]Bylo zjištěno, že sval postižený PAD vypadá víc heterogenně než zdravý sval

Kapitola 3

Cíle práce

Cílem bakalářské práce je vytvořit univerzální, na prostředí nezávislou, knihovnu funkcí pro zpracovaní ultrazvukových snímků. Knihovna musí splňovat následující požadavky:

Multi-platformnost - knihovnu bude možné použít nezávisle na platformě, jedinou podmínkou je přítomnost kompilátoru jazyka C++[1] a knihovny *OpenCV*

Kompatibilitu s nějakou permisivní licencí - knihovna musí být licencována silně permisivní licencí, bez závislostí na komponentách licencovaných nekompatibilními licencemi, například některou verzí *GNU GPL*

Texturní analýza obrazu - nad vybranou oblastí obrazu se provede texturní analýza pomocí redukované ko-okurenční matice.

Výpočet texturních příznaku - z vypočítané ko-okurenční matice budou vypočítány texturní příznaky, konkrétně homogenita, energie, korelace a kontrast.

Demonstrační aplikace

Předvést funkčnost všech funkcí - demonstrační aplikace bude předvádět možnosti knihovny a poskytovat funkční příklad pro další použití

Možnost vybrat nepravidelnou oblast - nad obrazem půjde vybrat nepravidelná oblast, z níž se vytvoří maska pro další zpracování

Obarvit vybranou oblast - vybraná oblast bude barevně označena předem definovanou paletou barev, *color map*

[1] Doporučuje se *clang*

MATLAB

Otestované a správné výsledky - v aplikačním prostředí MATLAB ověřit správnost postupů a výsledků funkcí texturální analýzy obrazu.

Kapitola 4

Návrh aplikace

Knihovna by měla být co nejvíce přímočará, jak na implementaci, tak použití. Pro výpočet všech texturních příznaků je nejvhodnější si vytvořit tzv. ko-okurenční matici, nebo-li GLCM, která nám poskytuje přehled o poloze pixelu a jeho souseda v daném směru. Danou vybranou oblast je také vhodné nějak barevně obarvit na základě hodnot pixelů. Zároveň by měl být všechen kód správně licencovaný, pro jeho další možné použití.

4.1 Ko-okurenční matice, GLCM

Ko-okurenční matice, nebo také GLCM (gray-level co-ocurrency matrix), je matice vytvořená z šedotónového obrazu. Vyjadřuje distribuci společně se vyskytujících pixelů v daném směru.

Pro vytvoření ko-okurenční matice je nutné znát obor hodnot pixelů v obrázku, v případě šedotónových pixelů jsou to hodnoty od 0 do 255 pro každý pixel. Vytvoření takové matice by znamenalo vytvoření matice o velikosti 256×256, jejíž zpracování by v některých případech mohlo značně zpomalovat běh programu. Proto se často obor hodnot zmenšuje, výsledná matice má mnohem menší rozměry a operace nad ní jsou mnohem rychlejší.

$$A = \begin{bmatrix} 10 & 10 & 20 \\ 40 & 30 & 70 \\ 20 & 50 & 80 \end{bmatrix} \tag{4.1}$$

$$B = \begin{bmatrix} 1 & 1 & 2 \\ 4 & 3 & 7 \\ 2 & 5 & 8 \end{bmatrix} \tag{4.2}$$

Postup vytvoření GLCM je velmi jednoduchý. V matici 4.1 se nachází hodnoty od 10-80. Kdyby nedošlo k zmenšení oboru hodnot, musela by ko-okurenční matice mít rozměry 81×81. Pokud ale zmenšíme obor hodnot o desítku, získáme matici 4.2, jejíž odpovídající ko-okurenční matice by měla rozměry pouhých 9×9.

Pro vytvoření ko-okurenční matice je ještě potřeba znát směr, ve kterém budeme kontrolovat všechny sousedy daného pixelu a kontrolovat. Do výsledné matice pak zapisujeme počet sousedů, kteří sdílejí stejnou hodnotu pixelů. Pro matici 4.2 je to tedy matice:

$$GLCM = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.3}$$

Na matici 4.3 tak můžeme vidět, že ve směru zprava doleva se nachází právě jednou hodnota 1 vedle další hodnoty 1 a jednou se též nachází hodnota 1 vedle hodnoty 2. Pokud by se podobných dvojic v matici B nacházelo více, byl by jejich počet zaznamenán na stejné pozici $i = 1$ a $j = 2$.

4.2 Texturní analýza pomocí GLCM

Ko-okurenční matice je v mnoha ohledech velmi užitečná, protože je schopna nám poskytnout mnoho informací o zpracovávaném obrazu, aniž bychom jej museli opakovaně procházet pixel po pixelu. Výpočetně by to bylo náročné a i největší ko-okurenční matice, která by obsahovala 65536 buněk, by byla pro zpracování ve většině případů vhodnější a rychlejší.

Výčet několika příznaků, které můžeme snadno vypočítat pomocí následujících rovnic z ko-okurenční matice:

- **Kontrast**

$$\sum_{i,j=0}^{N-1} P_{ij}(i-j)^2 \tag{4.4}$$

- **Homogenita**

$$\sum_{i,j=0}^{N-1} \frac{P_{ij}}{1+|i-j|} \tag{4.5}$$

- **Korelace**

$$\sum_{i,j=0}^{N-1} \frac{(i-\mu)(j-\mu)}{\sigma^2} \tag{4.6}$$

- **Energie**

$$\sum_{i,j=0}^{N-1} (P_{ij})^2 \tag{4.7}$$

- **Entropie**

$$\sum_{i,j=0}^{N-1} P_{ij} \ln P_{ij} \tag{4.8}$$

Kde i, j jsou souřadnice hodnoty v matici, P_{ij} je samotná hodnota, μ je střední hodnota a σ^2 je odchylka jasu.

4.3 Demonstrační aplikace

Jako ukázku použití funkcí z knihovny slouží jednoduchá demonstrační aplikace *demo*. Jejím hlavním účelem je načtení ultrazvukových snímků a výpočet příznaků *GLCM* z vybrané oblasti a jejich následný výpis na obrazovku.

4.4 Licence

Jedním z požadavků na knihovnu je použití silně permisivní open-source licence, která by usnadňovala další použití. Zároveň musí použité komponenty splňovat vzájemnou licenční kompatibilitu. Kompatibilní licence jsou licence, jejichž podmínky nejsou vzájemném rozporu. Příkladem takových licencí je například licence *GNU GPL* a *MIT*.

Pro kód bakalářské je nutno brát v úvahu tři licence: GNU GPL, BSD a MIT.

4.4.1 Licence GNU GPL

Licence GNU General Public License, česky všeobecně veřejná licence, je licence vytvořená Richardem Stallmanem pro projekt GNU, jako copyleftová[1] licence pro vznikající open-source projekty. GNU GPL se silně vymezuje proti uzavřenosti kódu a nařizuje autorům kódu odvozeného od kódu pod touto licencí, použití stejné, nebo kompatibilní licence. V praxi to znamená, že veškerý kód vytvořený pod licencí GNU GPL je možné volně editovat a šířit, za předpokladu, že zůstane otevřený a dostupný.

Postupem času vznikly dvě revize, které různě upravují vztahy autora odvozeného projektu, ale základní požadavky zůstali stejné. Ač je licence v současné době celkem oblíbená u autorů větších open-source projektů, vývojářům znesnadňuje použití takového kódu v projektech s nekompatibilními a více permisivními licencemi. Tento problém je znát hlavně u různých knihoven, z toho důvodu vznikla odvozená licence *LGPL*, která na původním kódu ponechává restrikce GPL licence, ale nevymáhá je na kódu odvozeném. To znamená, že programátor může použít jinou, běžně nekompatibilní licenci, za předpokladu, že neupraví původní kód.

[1] antonymum k výrazu copyright, vyjadřuje vztah autora software k jeho šíření a nevymáhání autorských práv

4.4.2 Licence BSD

Silně permisivní licence, na rozdíl od licencí GNU GPL se vyznačuje značnou svobodou pro autora kódu i kódu odvozeného. Vyžaduje pouze uvedení autora použitého kódu a zřeknutí se odpovědnosti. Hlavním důvodem nekompatibility s jinými copyleftovými licencemi je hlavně fakt, že umožňuje použití otevřeného kódu pro komerční účely.

Od vytvoření původní, nebo-li 4-bodové, licence vznikly dvě revize. První 3-bodová, někdy též "nová"BSD licence, a silně zjednodušená 2-bodová BSD licence. Původní licence byla oboustranně nekompatibilní s GNU GPL licencemi, revidovaná 3-bodová BSD licence už je dopředně kompatibilní s revidovanou GPLv3 licencí. To znamená, že projekt licencovaný pod 3-bodovou BSD licencí lze začlenit do projektu pod licencí GNU GPL, ale ne naopak, protože by docházelo k porušení klauzule o komerčním využití v GPL licenci.

Tato licence je vhodná pro vytváření knihoven a podpůrného software, v praxi je jedinou podmínkou nutnost zahrnout text licence BSD a autora kódu do závěrečných licenčních ujednání.

4.4.3 Licence MIT

Ještě o něco permisivnější licence než v případě BSD. Jediným nařízením, které autorovi MIT licence ukládá, je přiložení jejího textu k licenčním ujednáním. Ač je svým textem přímo v konfliktu s restrikcemi licence GPL, lze použít jejich kombinace, protože licence GPL explicitně povoluje její použití.

Na rozdíl od licencí GPL a BSD neprošla od svého vzniku žádnou revizí a je v současnosti jednou z nejoblíbenějších licencí.

4.4.4 Licence WTFPL

Za zmínku ale stojí asi nejpermisivnější open-source licence WTFPL[1], která autorovi nenařizuje takřka nic. Je založená na jednoduchém principu, kdy si autor odvozeného projektu může s původním kódem dělat co chce, za předpokladu, že si na případné nedostatky nebude stěžovat jeho autorovi.

[1] What The F*ck Public License

4.4.5 Výběr správně licence

Kvůli striktním omezením, není licence GPL vhodná pro použití v kódu bakalářské práce. S čímž se ale přichází nutnost opatrného výběru podpůrného kódu, který je pod touto licencí zveřejněn.

Ač se jako nejvhodnější licence jeví právě licence WTFPL, nepřijde mi vhodné ji použít. Rozhodl jsem se proto veškerý kód licencovat pod licencí MIT. Nejen z osobních preferencí, ale i z možnosti pozdějšího přelicencování a možnosti použití s restriktivnějšími licencemi.

4.5 Programové prostředí

Zatížení formátu DICOM patenty a množství zbytečných informací, které nejsou pro zpracování obrazu nutně, jsou důvody, proč jsou snímky z DICOM převáděny do běžnějších obrazových formátů. Ať už ve formě videa nebo obrázků ve formátu JPEG. Pro zpracování obrazových dat a počítačové vidění existuje pokročilá knihovna OpenCV, která je schopna se postarat o nezbytné úkony načtení a zobrazení obrazů a videí. Poskytuje též oficiálně podporované prostředí pro několik jazyků. Mezi nimi jsou asi nejrelevantnější jazyky C++ a Python.

4.5.1 C++

Kompilovaný jazyk, navržený původně jen jako objektové rozšíření jazyka C. Na rozdíl od jazyka C je však stále aktivně a celkem agresivně vyvíjen. Jeho hlavní výhodou rychlost, která je ale kompenzována relativně složitou kompilací. Od svého vzniku totiž prošel několika revizemi, ta poslední je z tohoto roku, 2017, proto se v něm nashromáždila spousta věcí, které jsou udržované už jen čistě z důvodu zpětné kompatibility.

V jazyce C++ lze ale uplatňovat nejen objektově orientovaný přístup k programování. Lze psát čistě procedurální a generický kód, který bude srovnatelný s kódem v C.

4.5.2 Python

Python je interpretovaný programovací jazyk, který byl navržený pro rychlé prototypování a nízkou náročnost na udržování kódu. V porovnání s C++ je kód mnohem čitelnější a udržitelný, ale rychlost vývoje je vykoupená samotnou rychlostí běhu programu, vzhledem k tomu, že se zpracování kódu provádí až těsně před spuštěním, proto není vhodný pro nasazení na místa, kde je potřeba rychlost.

V současnosti existují dvě verze, Python 2 a Python 3, které nejsou vzájemně kompatibilní. Verze 3 vznikla aby vyřešila historické problémy, které nelze jednoduše odstranit bez ztráty kompatibility, na rozdíl od verze 2, která už se téměř nemění, probíhá na třetí verzi aktivní vývoj.

Jazyk	Sekundy
Python 3	2.82
C++	0.60

Tabulka 4.1: Porovnání implementace algoritmu na vyhledávání v sekvenci DNA.
Zdroj: [2]

4.6 MATLAB a MEX

MATLAB je komerční programové prostředí a skriptovací jazyk, navržený jako prostředí pro vědeckotechnické účely, bez nutnosti používat nižší programovací jazyky. Z uživatele tak odpadá nutnost se takový jazyk učit, ale zároveň má k dispozici nástroje, které umožňují psát normální programy, které využívají integrované prostředí MATLABu, které zahrnuje vizualizaci dat, toolkit pro zpracování obrazových dat a rozšíření Simulink, pro vytváření interaktivních modelů.

Hlavní výhodou, jak již bylo řečeno, je relativně rychlá možnost začít pracovat na skutečném problému, než strávit větší množství času nastavováním pomocných komponent.

Ale MATLAB není a nikdy nebyl navrhován jako jazyk, nebo prostředí, pro produkční nasazení, proto je nutné případné postupy, které uživatel vytvořil v MATLABu, přepsat do nějakého nižšího jazyka.

Prostředí MATLABu umožňuje vytvářet rozšíření i v nižších jazycích, skrz abstraktní vrstvu MEX. MEX soubory se pak chovají jako srandardní funkce, které je možné volat a nechat si od nich poskytovat výstup. MATLAB poskytuje API pro vytváření MEX souborů v jazycích C, C++ a FORTRAN, ale neposkytuje univerzální kompatibilitu napříč platformami. Zkompilovaný MEX soubor tak nelze spustit na jiné platformě, než na jaké byl zkompilován.

Kapitola 5

Implementace

Jako jazyk implementace bylo zvoleno C++, protože knihovna OpenCV poskytuje nativní prostředí právě jen v C++. Ostatní podporované jazyky jsou z něj exportovány. Kapitola implementace je rozdělena do čtyř částí:

- Implementace knihovny SMOOTH
- Implementace demonstrační aplikace
- Vytvoření MEX souborů

5.1 Implementace knihovny SMOOTH

Knihovnu jsem pracovně pojmenoval jako SMOOTH. Všechny funkce, které jsem vytvořil, rozšiřují namespace[1] *cv*.

Funkce se dají rozdělit do dvou částí:

- **Utility** - funkce, které se přímo nezabývají požadavkům ze zadání, ale jsou pro jeho vypracování potřeba
- **Processing** - funkce, které se zabývají zpracováním obrazu podle zadání

Všechny matice se kterými knihovna pracuje jsou typu *Mat*. Jedná se o základní formát reprezentace matice v knihovně OpenCV. Oproti obyčejnému vícerozměrnému poli má hned několik výhod, mezi tu hlavní asi patří to, že každá nově vytvořená matice má všechny prvky nastavené na nulu.

5.1.1 Funkce pro vytváření GLCM

Vstupním bodem pro vytvoření ko-okurenční matice a později i příznaků je funkce *GLCM*, případně *GLCMWMask*, která poskytuje jako volitelný parametr masku. I v případě volání funkce *GLCM* je volána funkce *GLCMWMask* s prázdnou maskou.

Popis vstupních parametrů:

Mat image - dvourozměrná šedotónová matice, ze které se má GLCM vytvořit.

Mat mask - maska oblasti, nad kterou se bude vytvářet GLCM

bool downscale - implicitně nastaveny na *False*, pokyn, zda provádět downscaling[2] vstupní matice, nebo ne.

Direction dir - směr ve kterém se mají hledat sousedé

Dircetion je výčtovým typem a vyjadřuje úhel 0°, 45°, 90° a 135°.

```
enum Direction {
        A0, A45, A90, A135
};
```

[1] Jmenný prostor
[2] Zmenšení oboru hodnot

Samotné vyhledávání a počítání je přímočaré, jediné co je nutné hlídat je, aby vyhledávání souseda nepřeteklo mimo hranice obrazu, nebo mimo hranice masky. Pokud si uživatel vybere, že chce nad maticí nejdřív provést downscaling, zavolá se ulitity funkce *MatTo4*, jejíž návratovou hodnotou je nová matice, jejímž obsahem jsou hodnoty z původní matice převedené do 4 bitového oboru hodnot. Tím se sníží i velikost výsledné ko-okurenční matice na velikost 16×16.

Výsledná GLCM matici, která není normalizovaná, je před výpočtem texturálních příznaků nutné normalizovat. Normalizovanou maticí se rozumí matice, ve které je součet všech prvků roven jedné. Pro normalizaci GLCM matice se v knihovně nachází funkce *GLCMNorm* jejíž jediným argumentem je samotná matice.

Pro výpočet texturálních příznaků pak slouží následující funkce:

- GLCMMean - pro výpočet střední hodnoty podle rovnice:

$$\mu = \sum_{i,j=0}^{N-1} iP_{ij} \tag{5.1}$$

- GLCMVar - pro výpočet variance matice podle rovnice:

$$\sigma^2 = \sum_{i,j=0}^{N-1} P_{ij}(i-\mu)^2 \tag{5.2}$$

- GLCMCorr - pro výpočet korelace
- GLCMProps - sdružující funkce, která zároveň počítá energii, entropii, homogenitu a kontrast

Všechny mají stejný a jediný argument, normalizovanou GLCM matici. Výstupem funkce *GLCMProps* je jednorozměrné pole čísel typu *double*[1] s hodnotami jednotlivých příznaků v následujícím pořadí:

- props[0] - kontrast
- props[1] - homogenita
- props[2] - energie
- props[3] - korelace
- props[4] - entropie

Funkce *GLCMProps* je navržena tak, aby bylo snadné do ní kdykoliv přidat některý z dalších příznaků. Výpočet by se měl držet existujícího schématu a vyvarovat se zbytečným průchodům celé matice, aby nedošlo ke zbytečnému zpomalení.

5.1.2 Utility funkce

Při implementaci demonstrační aplikace i knihovny samotné, vyvstaly požadavky na vytvoření pomocných funkcí.

loadFrames(char* filename) - Funkce, která se stará o načtení video souboru a uložení jednotlivých snímků do *vector*u[2] matic. Jediným vstupem je název video souboru. Pro načtení video souboru je využito třídy *VideoCapture*. Při použití na jiných platformách než Linux ne potřeba myslet na to, že podpora různých video formátů je závislá čistě na jejich podpoře na dané platformě.

MatTo4(Mat image) - Funkce pro downscaling matice. Matice musí být dvourozměrná, šedotónový obrázek. Návratovým typem je nová matice.

MatToBGRGRAY(Mat image) - Funkce pro převedení obrázku reprezentovaného BGR modelem, na šedotónový obrázek, který je též reprezentovaný BGR modelem. Vstupem a výstupem je BGR matice.

[1] 64-bitový datový typ s desetinnou čárkou

[2] Datový typ vector implementuje spojový seznam, proto není potřeba definovat jeho velikost.

maskOnRoi(...) - Funkce pro aplikaci zvoleného barevného modelu, *color map*, na vybranou oblast. Jedním ze vstupních parametrů je i volitelný ukazatel na funkci, která se nad obrazem zavolá těsně před tím, než je na vybranou oblast aplikován barevný model. Dalšími parametry jsou maska, *boundingBox*, který představuje obdélníkový výřez oblasti a nastavení průhlednosti.

Ukázka funkce, která je použita pro předzpracování obrázku před aplikací barevné palety:

```
void replaceBlue(Mat image, Mat mask) {
        Mat m;
        threshold(image, m, 12, 127, 0);
        m.copyTo(image);
}

int main(int argc, char** argv) {
        .
        .
        .
        maskOnRoi(v, COLORMAP_HSV, bbox, mask, 1, replaceBlue);
        .
        .
        .
}
```

5.2 Bresenhamův algoritmus a calcLine

Při psaní demonstrační aplikace jsem narazil na omezení event systému OpenCV, které znemožňovalo vytvoření nepřerušované úsečky z bodů pomocí kurzoru myši. Interní event systém grafického rozhraní knihovny OpenCV má nastavený časovač na relativně vysokou hodnotu, proto se při pokusu zaznamenat všechny body, kterými kurzor prochází, nezaznamenají všechny body, ale jen některé. Počet zaznamenaných bodů je závislý na rychlosti kurzoru. Protože OpenCV nenabízí moc možností nastavení interního event systému, tak se jako nejlepší řešení jeví ty body dopočítat.

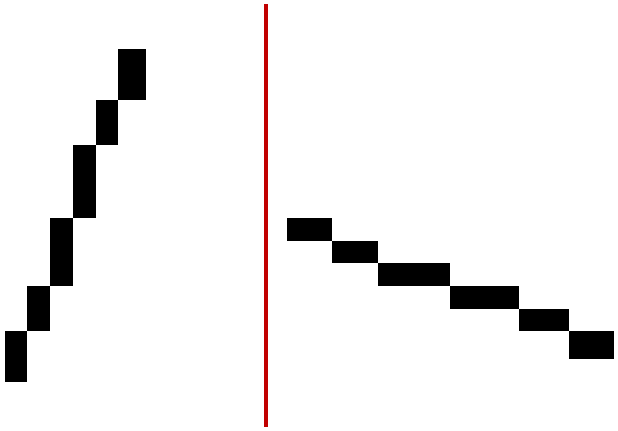
OpenCV má iterátor *LineIterator*, který takové body umí dopočítat. Ale použití takového iterátoru není triviální a vzhledem k tomu, že samotný algoritmus není složitý, implementoval

jsem ho samostatně ve funkci *calcLine*, jejíž výstupem je *vector* bodů. Vstupními hodnotami jsou počáteční a cílový bod.

Pro výpočet rastru linky existuje několik algoritmů, já jsem zvolil Bresenhamův algoritmus, protože jej používá i OpenCV a od jiných algoritmů, například DDA algoritmu, má výhodu v tom, že pracuje s celými čísly.

Bresenhamův algoritmus v mnohém funguje jako jednoduchý klasifikátor. V každém kroku se rozhodujeme, mezi dvěma směry, kde umístit další bod přímky.

Vstupními argumenty funkce jsou dva body typu *Point*, mezi kterými chceme vytvořit úsečku. Na jejich vzájemné poloze v prostoru nastavíme směr, kterým se budou vytvářet body, *stepx* a *stepy*. Poté je nutné si zjistit vzdálenost na osách dx, dy a vypočítat odchylku err. Při vypočítání odchylky je nutné brát v úvahu sklon požadované úsečky.



Obrázek 5.1: Dva sklony úsečky. Vlevo: $dx < dy$ Vpravo: $dy < dx$

Podle sklonu úsečky lze vypočítat odchylku podle rovnice:

$$err = \begin{cases} \frac{dx}{2}, & dx > dy \\ \frac{-dy}{2}, & \end{cases} \tag{5.3}$$

S vypočítanou odchylkou lze začít vyhledávat body. Jednotlivé body vyhledáme postupným přičítáním hodnoty kroku v daném směru a kontrolou velikosti odchylky. S každým krokem periodicky měníme směr odchylky a kontrolujeme, v jakém bodě má nejmenší vzdálenost k cíli. Pokud takový bod nalezneme, uložíme jej do seznamu bodů a pokračujeme dalším krokem. Algoritmus končí v momentě, kdy najdeme bod, který odpovídá cíli.

5.3 Implementace demonstrační aplikace

Demonstrační aplikace slouží k předvedení možností knihovny SMOOTH a OpenCV. Prostředí aplikace je velmi jednoduché, skládá se jen z okna a po výběru oblasti i z výpisu jednotlivých příznaků. Sestavení aplikace probíhá přes *Makefile*[1]

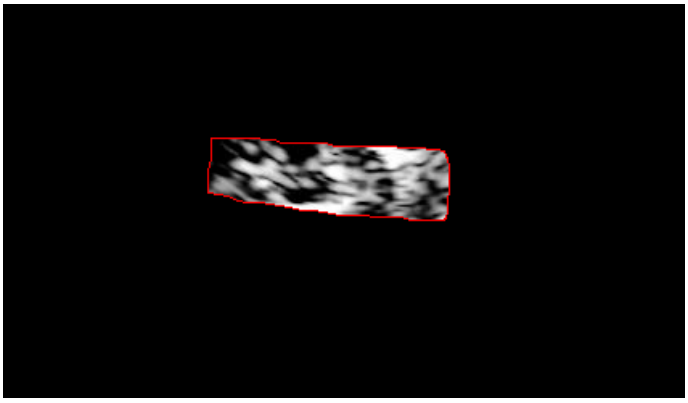
Obrázek 5.2: Hlavní okno aplikace

Vstupním argumentem aplikace je video soubor s ultrazvukovými snímky, který je po spuštění načten a převeden do *vector*u matic. Nad oknem je registrován *event handler*[2] pro zachytávání pohybu myši a detekci stisknutí tlačítek.

Po načtení souboru a registraci handleru dochází k vytvoření nekonečné smyčky, v které je do okna zobrazován aktuální snímek ultrazvuku. V tento okamžik je možné začít nad snímkem vytvářet masku jednoduše tím, že se nakreslí požadovaný tvar. Handler pro zachytávání myši si poznačí, kde uživatel vytvořil začátek oblasti, kudy se pohyboval a kde skončil.

[1] Krátký soubor, který obsahuje návody pro program *make*, který podle nich sestaví daný cíl
[2] Handler je funkce, která se stará o zpracování vyvolané události, třeba kliknutí myši

K detekci bodů uvnitř vybrané je potřeba znát průsečík hranice, kterou uživatel kreslí. Pokud známe body, kterými již uživatel prošel, stačí během kreslení oblasti kontrolovat, zda-li uživatel nenarazil na bod, který už je uložený. Pokud na takový bod narazí, jsou všechny další body zahozeny. Zahozeny jsou i body, které byly nakresleny před bodem průsečíku, protože nejsou součástí hranice. Z vyznačené oblasti pak pomocí funkce *createMask* vytvořena binární maska.

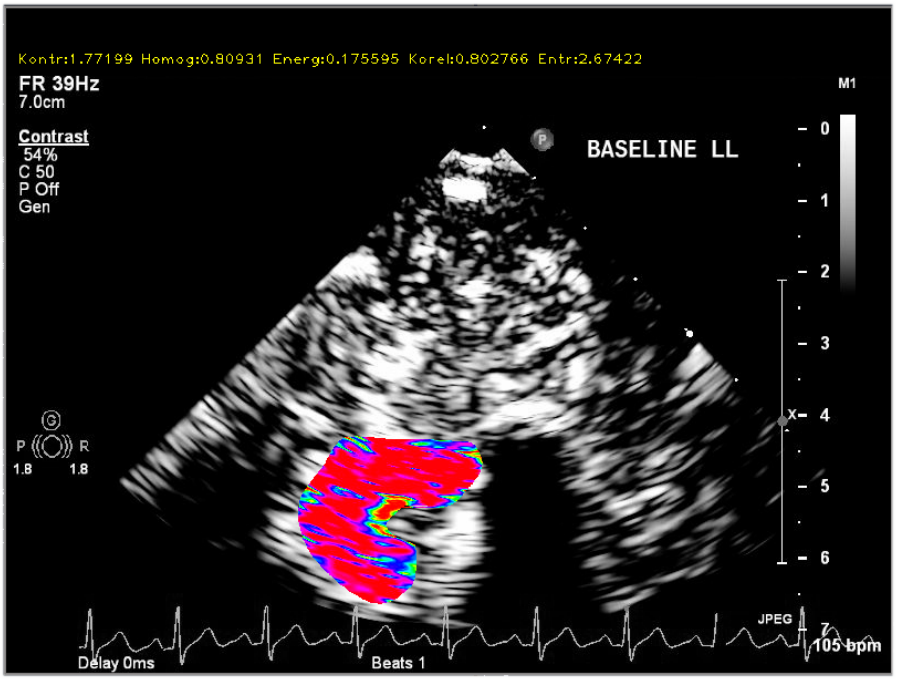


Obrázek 5.3: Výřez snímku po vytvoření a aplikaci masky

Samotná maska má ale nepravidelný tvar, proto je potřeba kolem ní vytvořit opsaný obdélník pomocí funkce *cv::boundingRect*. Tím vznikne obdélníková oblast, nad kterou se mohou provádět běžné maticové operace, ale zároveň si zachová informace o masce, jak můžeme vidět na matici 5.4.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix} \tag{5.4}$$

Samotný výřez snímku opatřený maskou lze jednoduše poslat jako argument do funkce *GLCMWMask* a uložit výslednou GLCM matici. Dalším krokem je výpočet příznaků pomocí funkce *GLCMProps* a jejich zobrazení v horní části okna. Na snímku je překryta část grafického rozhraní ultrazvuku černou oblastí, do které jsou vypsány jednotlivé příznaky. Jednotlivé snímky lze posouvat vpřed, nebo zpět, klávesami *'n'* a *'p'*.



Obrázek 5.4: Ultrazvukový snímek s obarvenou oblastí a zobrazenými příznaky.

5.4 Implementace MEX souborů

Před samotnou implementací MEX souborů je nutné nastavit prostředí, ve kterém se budou MEX soubory sestavovat.

V současnosti existují dvě možnosti, jak sestavit MEX soubor s hlavičkovými soubory OpenCV. Tou první je použití standardního MEX kompilátoru a hlavičkové soubory přibalit standardním způsobem, přes argument *-I*. Druhou možností, která ale není dostupná pro instalace hromadné licence, je použití rozšiřujícího balíku *Computer Vision System Toolbox*, která si nainstaluje, pomocí příkazu *visionSupportPackages* všechny potřebné knihovny a hlavičkové soubory. Nevýhodou už tak jen zůstává celková netransparentnost celého procesu. Při použití standardního postupu, je potřeba při kompilaci zahrnou všechny potřebné hlavičkové soubory OpenCV.

Před samotnou kompilací MEX souboru je potřeba nastavit verzi kompilátoru, protože MATLAB, konkrétně verze 2015b podporuje pouze kompilátor *g++* do verze 4.7, nejnovější MATLAB R2017a pak podporuje až to veze 4.9. Pokud se tak v systému nenachází potřebná verze, je potřeba ji doinstalovat, případně sestavit. MATLAB je distribuován s vlastními knihovnami, které ale překrývají systémové knihovny. To je problém hlavně v případě knihovny *libstdc++*, protože poskytuje starší *cross-vendor application binary interface*, než které vyžaduje aktuální OpenCV ve verzi 3.2.

Struktura samotného MEX souboru je jednoduchá. Jeden MEX soubor pak po kompilaci odpovídá jedné volatelné funkci v MATLABu. Z programového hlediska se jedná o C++ kód rozšířený knihovnou *"mex.h"*, který obsahuje definice datových typů kompatibilních s MATLABem. Důležité je to hlavně při práci s maticemi, protože MATLAB reprezentuje matice v tzv. *column-major* řazení, kdežto C++ a obecně většina programovacích jazyků v *row-major* řazení.

$$M = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \tag{5.5}$$

Mějme ukázkovou matici 5.5. Při *row-major* řazení jsou v paměti uloženy ukazatelé na začátek řádku, pokud bychom chtěli vybrat prvek a_{11}, tak se nejprve v paměti dotážeme na na řádek 1, tj. $M[1]$, a poté na sloupec 1, tj. $M[1][1]$. Ale v případě *column-major* řazení, se postupuje přesně obráceně, pro výběr prvku a_{12} vybereme celý sloupec 2 a poté řádek 1.

Tento způsob řazení odpovídá více matematickému přístupu k matici v reálném světě, kde nejdřív hledáme hodnotu x a k ní hodnotu y, tedy sloupec a řádek. Dalším jazykem, který je také často používán k vědeckotechnickým účelům a má *column-major* řazení, je FORTRAN.

Základní struktura MEX souboru vypadá následovně:

```
#include "mex.h"

void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{
        ...
}
```

Vstupním bodem je funkce *mexFunction*, která je volána pokaždé, když je z MATLABu zavolán název MEX souboru. Výstup však funkce neposkytuje přes standardní návratovou hodnotu, ale před pole typu *mxArray *plhs*, která je předána jako vstupní parametr. Všechny vstupní parametry od uživatele jsou uloženy v poli *mxArray *prhs*[1].

Při zakomponování funkce pro výpočet GLCM matice a jejích příznaků, je nejprve nutné transponovat vstupní obrazovou matici na *row-major* řazení. Poté je vypočítána GLCM matice, příznaky jsou vypsány na obrazovku a do prostředí je vrácena samotná GLCM matice, která prošla zpětnou transpozicí na *column-major* řazení. Z výpisu příznaků je vynechána hodnota entropie, MATLAB nemá funkci pro výpočet entropie z GLCM matice, není tedy s čím porovnávat.

[1] plhs a prhs jsou zkratky pro "Parameters Left-Hand Side"a "Parameters Right-Hand Side", nlhs a nrhs zase pro "Number of Left-Hand Side"a "Number of Right-Hand Side"

Kapitola 6

Uživatelská dokumentace

Pro úspěšné sestavení je nutné mít v systému přítomný kompilátor jazyka C++ a knihovnu OpenCV alespoň ve verzi 3.2. K sestavení demonstrační aplikace slouží soubor *Makefile* a příkaz:

make demo

Vstupním argumentem pro zkompilovaný program *demo* je video soubor s ultrazvukovými snímky. Po spuštění se v okně zobrazuje aktuální, první, snímek z video souboru. Držením levého tlačítka myši a vzájemným tažením po snímku, lze vytvořit nepravidelný tvar, zvýrazněný červenou dráhou. Kreslení je ukončeno v případě, že uživatel zvedl prst z tlačítka, nebo protnul již existující dráhu.

Zvolená oblast je obarvena podle hodnot, které v ní leží a v horní části okna se zobrazí vypočtené příznaky. Pomocí tlačítek *'n'* a *'p'* se lze mezi snímky přesouvat vpřed nebo zpět.

Kapitola 7

Testování

Knihovna byla před demonstrační aplikaci otestována na reálných ultrazvukových záznamech. Video soubory byly uloženy v různých formátech kontejneru.

Výsledná GLCM matice z knihovny SMOOTH se od matice vytvořené v MATLABu, ze stejného testovacího obrazu, zanedbatelně lišila v několika bodech. Důvodem je odlišný přístup k mezním hodnotám na okraji pole. Hodnoty výsledných příznaků:

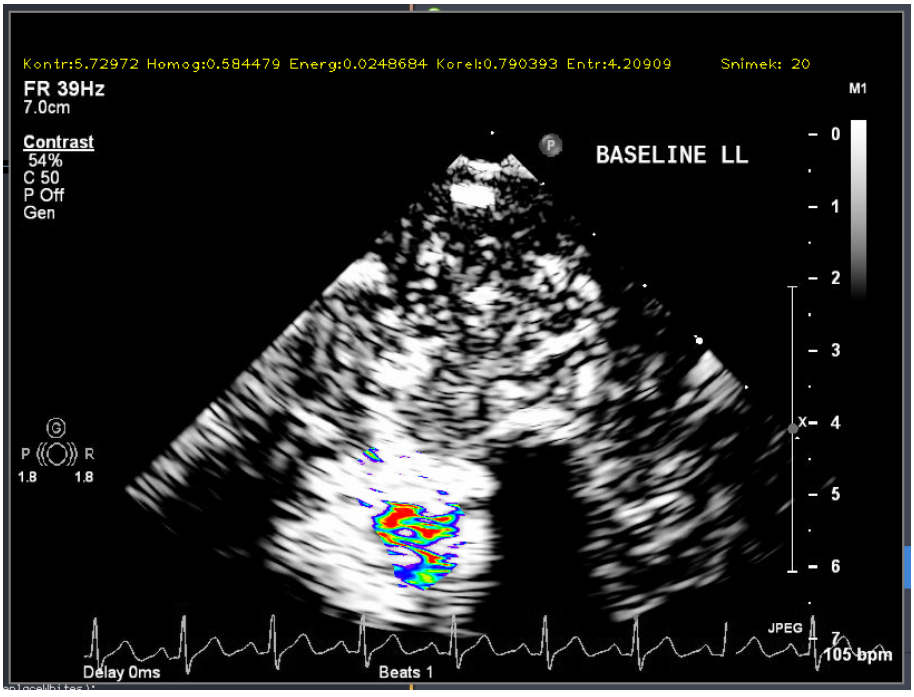
Zdroj	Kontrast	Homogenita	Korelace	Energie	Entropie
MATLAB	2.0317	0.8937	0.9048	0.3974	-
SMOOTH	2.09232	0.893743	0.953501	0.397347	2.1393

Tabulka 7.1: Porovnání texturálních příznaků z nativní MATLAB funkce a z SMOOTH knihovny

Pro účely testování funkčnosti masky jsem vytvořil funkci *replaceWhite*, která ve vybrané oblasti barevně obarví jen tmavé hodnoty:

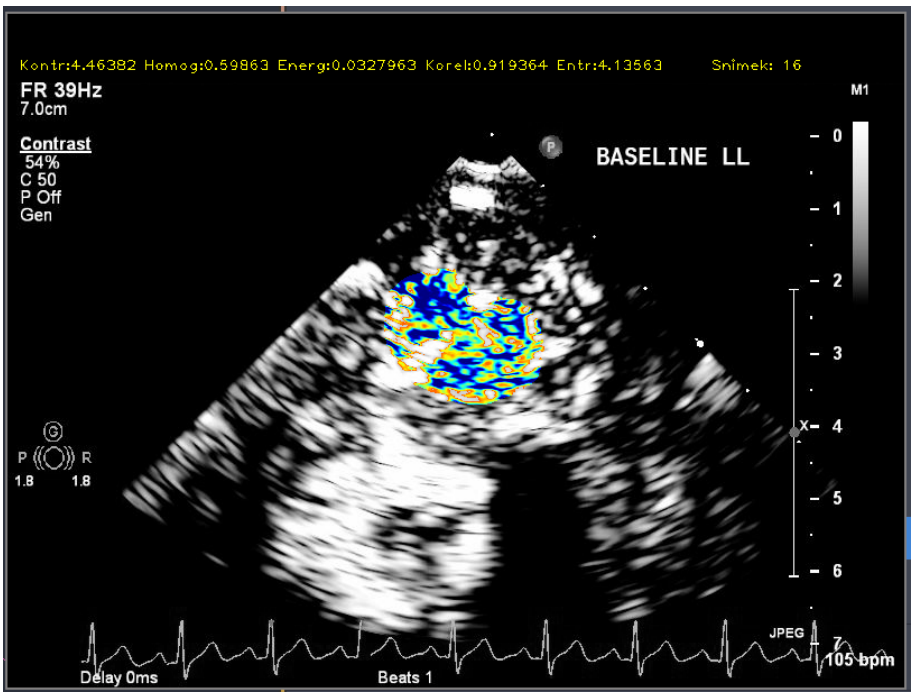
```
void replaceWhites(Mat image, Mat mask) {
        Mat g;
        cvtColor(image, g, COLOR_BGR2GRAY);
        Size size = image.size();

        for(int y=0; y < size.height; y++) {
                for(int x=0; x < size.width; x++) {
                        if ((int)g.at<uchar>(y, x) > 200) {
                                mask.at<uchar>(y, x) = 0;
                        }
                }
        }
}
```

Funkce upravuje použitou masku, takže je vždy vhodné volat funkci *maskOnRoi* až po provedení matematických operací. Obarvený snímek pak vypadá takto:



Obrázek 7.1: Ultrazvukový snímek obarvený jen v nízkých oblastech, HSV paleta

Zvýraznění tmavých oblastí může později pomoct při pochopení echogenity okolní tkáně.



Obrázek 7.2: Ultrazvukový snímek obarvený jen v nízkých oblastech, RGB paleta

Kapitola 8

Diskuze

Knihovna SMOOTH je v použitelném stavu, bez závislostí na relativně špatně dostupném prostředí MATLAB. Většina funkcí, které byly použity, se nachází v knihovně OpenCV, až na funkce pro výpočet GLCM matice. Bylo by vhodné se správci knihovny diskutovat případné rozšíření knihovny o tyto funkce, v případě pokračování práce, pak celý toolkit pro zpracování snímků z formátu DICOM, jehož podpora teď v OpenCV není.

Jedním z požadavků bylo i otestování výsledků proti výsledkům MATLABu, které byly zvoleny jako referenční, v prostředí MEX. MATLAB je ohromná aplikace, se spoustou modulů, které mají různé závislosti. To jej činí nejen mocným, ale také pomalým a těžkopádným. Programátor, nebo výzkumník, je často odkázán na netransparentní práci někoho jiného, bez hlubšího pochopení, proč se něco děje právě tak jak se to děje. Výzkumník si s tím nemusí moc lámat hlavu, ale programátor, který se snaží MATLAB rozšířit tak neodvratně spěje do bodu, kdy vytváří jen další kus těžkopádného kód.

Rozdíl ve výsledcích příznaků vypočítaných MATLABem a knihovnou SMOOTH může být do jisté míry ovlivněn způsobem převodu obrazu z barevné palety do šedotónové palety. Způsobů pro převod sice není tolik, ale silně záleží na vnitřním fungování jednotlivých funkcí, případně verzi OpenCV nebo MATLABu. Výsledné příznaky se od sebe liší zanedbatelně, kromě hodnot kontrastu a korelace, které se liší o něco více než zbylé příznaky.

Knihovnu SMOOTH by do budoucna bylo rozhodně možné jednoduše rozšířit o výpočet jiných příznaků, prostor a prostředí je na to připraveno. Pro účely zpracování ultrazvukových snímků by bylo vhodné ji rozšířit o další možnosti texturální analýzy, případně přidat podporu pro detekci podobných struktur. Určitě by bylo užitečné mít funkce pro stanovení

echogenity napříč snímkem, nebo detekci pevných, případně patogenních, struktur. Jednou z dalších možných aplikací knihovny by bylo začlenění podpory pro mobilní zařízení a možnosti použití v telemedicíně.

Kapitola 9

Závěr

Cílem bakalářské práce bylo navrhnout a implementovat otevřenou knihovnu pro zpracování ultrazvukových snímků.

V první části je popsán princip ultrazvuku, jaké existují možnosti jeho zobrazení a pokryl jsem i nemoc PAD, včetně probíhajícího výzkumu, zabývajícího se možností diagnostiky na základě homogenity okolí.

V části návrhu aplikace jsem se zabýval pozadím operací nad obrazem pro výpočet texturálních příznaků, ale také možnostmi licencování a použití podpůrného software v této práci a možnostmi napojení knihovny na MATLAB, pro ověření správnosti výsledků. Popsána byla i základní funkčnost demonstrační aplikace.

Implementace probíhala přímočaře, ale s přibývajícím kódem se knihovna SMOOTH rozšířila i o funkce, které nemají se zpracováním ultrazvukových snímků nic společného. Tyto funkce jsem ale považoval za důležité a užitečné pro případný další vývoj, případně jejich existující implementace byla svazující nebo nepřehledná.

Jako podpůrná obrazová knihovna byla zvoleno OpenCV, které se zprvu zdálo jako ideální volba. Až po nějakém čase se ukázalo, že některé její části nedávají zrovna smysl a jejich použití je značně nepohodlné. Pořád se ale jedná o nejvyspělejší knihovnu pro počítačové vidění, která má za sebou spoustu let intenzivního vývoje a spousta věcí a problémů byla nějakým způsobem vyřešena, z programátora pak odpadá řešení problémů v samotné knihovně, pokud se přizpůsobí jejímu použití.

Práce s prostředím MATLAB na úrovni MEX a samotného rozšiřujícího prostředí byla už od začátku problémová, vzhledem k rozdílnostem mezi verzemi MATLABU jsem neustále

narážel na problémy, které plynuly z rozdílných verzí různých knihoven, které byly dodávané s MATLABem, OpenCV nebo standardní knihovnou C++.

Moje práce by se dala považovat za základ pro další rozvoj knihovny a prostředí pro zpracování ultrazvukových snímků na půdě ČVUT a budu rád a nápomocen komukoliv, kdo se rozhodne na práci navázat. Všechen kód a vlastní grafika je dostupná pod licencí MIT.

Reference

[1] John A. Bittl and Alan T. Hirsch. Concomitant peripheral arterial disease and coronary artery disease. http://circ.ahajournals.org/content/circulationaha/109/25/3136/F6.large.jpg, 2004. ISSN 0009-7322. [Online].

[2] Isac Gouy. The computer language benchmarks game. https://benchmarksgame.alioth.debian.org/u64q/revcomp.html. [Online].

[3] Robert M. Haralick and Linda G. Shapiro. *Computer and Robot Vision, Vol. 1.* Addison-Wesley, 1991. ISBN 0201108771.

[4] Seyed-Farzad Mohammadi. Sheep practice eye for ophthalmic surgery training in skills laboratory. https://www.researchgate.net/profile/Seyed-Farzad_Mohammadi/publication/51150099/figure/fig1/AS:305718277230592@1449900325702/Figure-1-A-and-B-mode-ultrasonogram-of-a-1-year-old-Afshar-sheep-eye-Interfaces-A-Z.png. [Online].

[5] Wikipedia. Stent — Wikipedia, the free encyclopedia. https://upload.wikimedia.org/wikipedia/commons/6/6a/SEMS_endo.jpg. [Online].

[6] WikiSkripta. Index kotníkových tlaků. https://upload.wikimedia.org/wikipedia/commons/9/91/Pad_abi_ENG.svg. [Online].

Seznam obrázků

Seznam tabulek

Seznam příloh

Přílohy na CD

Příloha 1 Abstrakt a klíčová slova česky (abstrakt.pdf)

Příloha 2 Abstrakt a klíčová slova anglicky (abstract.pdf)

Příloha 3 Naskenované zadání BP (zadani.jpg)

Příloha 4 Kompletní bakalářská práce (BP_Dusek.pdf)

Příloha 5 Zdrojové kódy, jako git repozitář (src.zip)

Příloha 6 Testovací video soubory (vids.zip)