Zpracování digitalizovaného obrazu - Semestrální práce

Jan Vrba a Andrea Kadlecová

Západočeská Univerzita V Plzni
Katedra kybernetiky
Datum: 18. května 2024

Obsah

1 Zadání **2**

2 Použití kódu **2**

3 Vypracování **3**

3.1 Načtení anotací 3
3.2 Načtení obrázků 4
3.3 Práce s jednotlivými obrázky a odstranění stínu 5
3.4 Prahování a odstranění rohů 5
3.5 Morfologie pro oddělení řezu a stehů 6
3.6 Skeletonizace a dilatace skeletonizovaného obrázku 7
3.7 Segmentace 9
3.8 Převedení Boolean obrázku na RGB 9
3.9 Freemanův řetězový kód 9

4 Závěr **12**

1 Zadání

Zadáním této semestrální práce je zhodnotit kvalitu chirurgického šití na poskytnutých obrázcích. Na obrázcích se nachází řezy a stehy, které řez uzavírají. Úkolem je využít metod počítačového vidění k extrakci a segmentaci obrazových dat. Následně využít metod počítačového učení k analyzování dat a určení počtu stehů na jednotlivých obrazech. Všechny obrazy byly předem anotované, podle kterých můžeme kontrolovat přesnost algoritmu.

2 Použití kódu

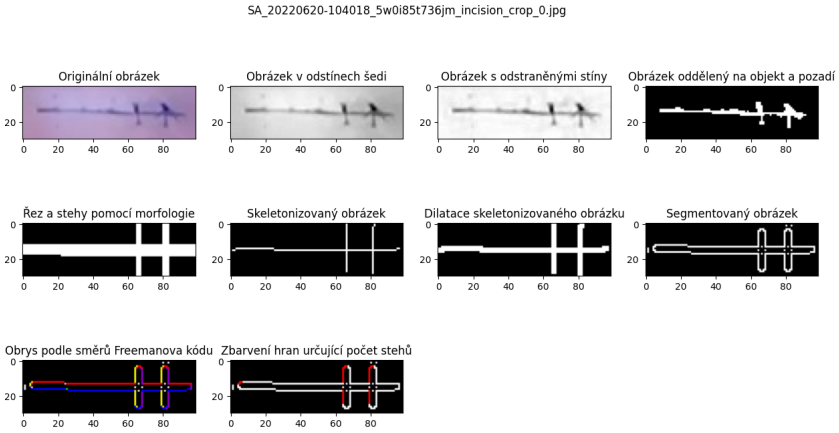
Pro použití a volání kódu musíme v příkazové řádce volat funkci run.py. Za první argument napiště název csv. souboru, kam chcete uložit výsledky. Jako druhý argument můžete napsat buď *-v* a nebo název obrázku, který vás zajímá. Od třetího argumentu můžete psát libovolné množství názvů obrázků, které vás zajímají. Algoritmus rozliší, zda jste napsal *-v* do argumentu a bude a nebo nebude vám podle toho vykreslovat grafické znázornění funkce algoritmu pro ty obrázky, které jste v argumentech uvedl. Program analyzuje všechny obrázky ze složky zdo2024, ale vykresluje a zapisuje do výstupu jen ty, které byly napsány v argumentech.

Příklad volání funkce může být:

```
$ python run.py output.csv -v SA\_20220620-103348\_3imoxskpwsvo\_incision\_crop-
\_0.jpg
```

Soubor requirements.txt obsahuje seznam nainstalovaných externích balíčků, které jsme v programu využili a jejich verze.

Vizualizace ukazuje postup úpravy obrázků a název jednotlivých úprav. Příklad vizualizace pro jeden obrázek můžeme sledovat na obrázku 1. Pro nepřekrývání názvů obrázků je nutné zvětšit si figure na celou obrazovku.



Obrázek 1: Příklad vizualizace

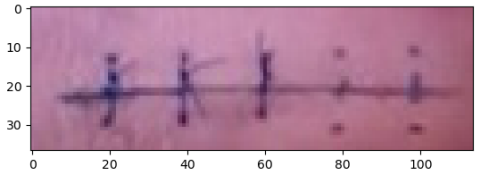
3 Vypracování

3.1 Načtení anotací

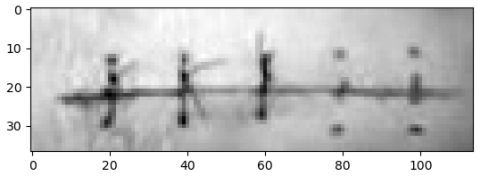
Pro začátek jsme načetli anotace ze souboru *annotations.xml*, který se nachází ve složce *zdo2024*. Soubor jsme rozdělili pomocí funkcí na operace s řetězci. Do slovníku jsme uložili na pozici klíče název obrázku a do jeho hodnoty počet stehů jednotlivých obrázků. Tento slovník jsme později využili pro kontrolu dat, které vyhodnotil náš algoritmus.

3.2 Načtení obrázků

Nyní jsme načetli obrázky s řezy a stehy, se kterýma náš algoritmus pracuje. Obrázky jsme načetli do slovníku, kde za klíč jsme uložili název obrázku a jako hodnotu samotný obrázek. Obrázek byl do slovníku načtený rovnou v odstínech šedi pro lepší práci v budoucnu. Na obrázcíh 2 a 3 můžeme sledovat obrázek v původním zobrazení a v odstínech šedi, se kterým jsme pracovali dál.



Obrázek 2: Originální obrázek

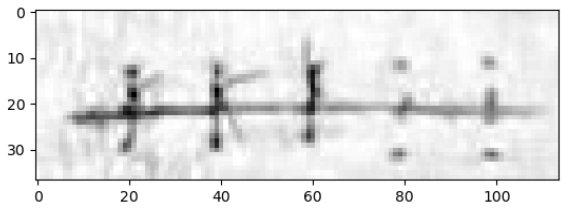


Obrázek 3: Obrázek v odstínech šedi

3.3 Práce s jednotlivými obrázky a odstranění stínu

V této části začínáme pracovat s jednotlivými obrázky pomocí velkého cyklu *for*, kde je morfologicky upravujeme a analyzujeme.

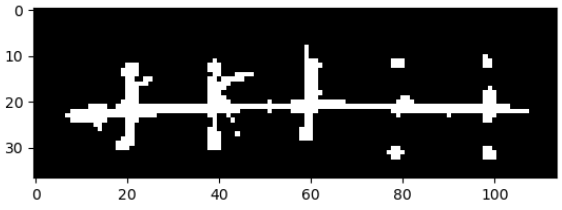
Skoro každý obrázek překrýval v nějaké části stín, jehož jas se často přibližoval k jasu řezu nebo stehů a zhoršoval podmínky pro rozpoznávání objektu a pozadí v další operaci, proto jsme se nejdříve zbavili stínů. Nejdříve jsme obrázek dilatovali s jádrem ve tvaru čtverce 7x7. Poté jsme využili funkce $cv2.medianBlur()$, která nahradí všechny elementu pod jádrem mediánem, vytvořeným z bodů nacházející se také pod jádrem. Následně jsme využili funkce $cv2.absdiff()$, kterou jsme odečetli od 255. Funkce $absdiff()$ zjišťuje absolutní rozdíl mezi dvěma poly. Upravený obrázek jsme poté normalizovali. Tyto všechny úpravy vedli k vyjasnění stínů a udržení řezu a stehů. Všechny tyto funkce jsme použili díky knihovně OpenCV (Viz obrázek 4).



Obrázek 4: Obrázek s odstraněnými stíny

3.4 Prahování a odstranění rohů

Po zesvětlení stínů jsme mohli oddělit objekt od pozadí za pomocí metody prahování Otsu. Tato metoda rozdělí řez a stehy od zbytku obrázku (Viz obrázek 5).



Obrázek 5: Obrázek oddělený na objekt a pozadí pomocí metody prahování Otsu

Následně jsme odstranili objekt z rohů obrázku, protože v rohách se nenachází ani řez ani stehy, ale často se tam nachází nežádoucí tvary, které prahování chybně vyhodnotilo jako objekt. Zejména levá strana často obsahuje vertikální čáru, kterou by si mohl algoritmus splést se stehem. Proto z levé strany mažeme 5% z celkové šířky obrázku, zatímco z ostatních stran jen pár pixelů.

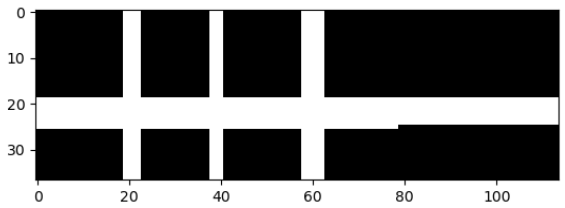
3.5 Morfologie pro oddělení řezu a stehů

Nyní jsme využili morfologickýxh operací pro nalezení a oddělení řezu a stehů. Pro nalezení řezu jsme využili nejdříve eroze s jádrem ve tvaru horizontální čáry dostatečně dlouhé, aby se mohlo předpokládat, že takto dlouhá horizontální čára se může nacházet jen v řezu. Poté jsme použili dilataci s mnohem větším jádrem, abychom nalezený řez roztáhli a spojili tak místa, kde mohlo prahování rozdělit řez na více částí. Takto nalezený řez jsme uložili do samostatného pole.

Stejným způsobem jsme identifikovali stehy. Abychom lépe indentifikovali stehy, museli jsme určit jádro, které se bude škálovat s velikostí obrázku. Některé obrázky mají větší rozlišení a tudíž mají delší stehy a opačně. Proto škálujeme jádro, aby pro méně kvalitní obrázky stačilo méně pixelů pro je-

jich identifikaci a naopak pro kvalitnější obrázky je potřeba větší jádro, aby algoritmus neidentifikoval i jiné dostatečně vysoké části jako stehy. Po erozy jsme obrázek dilatovali, abychom nalezené stehy zvětšili. Nalezené stehy jsme také uložili do samostatného pole.

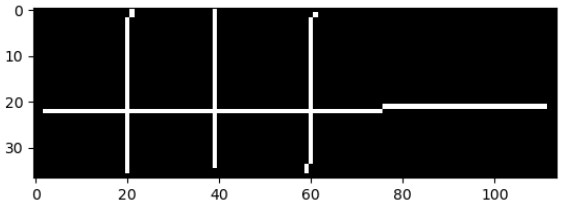
Oba morfologicky rozdělené obrázky jsme spojili dohromady do jednoho, jak můžeme vidět v následujícím obrázku 6.



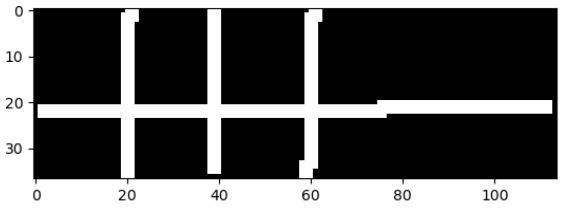
Obrázek 6: Nalezený řez a stehy pomocí morfologických operací

3.6 Skeletonizace a dilatace skeletonizovaného obrázku

Po oddělení stehů a řezu od dalších nežádoucích elementů, které mohlo prahování identifikovat jako objekt, jsme obrázek skeletonizovali (Viz obrázek 7). Skeletonizovaný obrázek jsme dilatovali jádrem ve tvaru 3x3 abychom ho později mohli správně segmentovat (Viz obrázek 8).



Obrázek 7: Skeletonizovaný obrázek

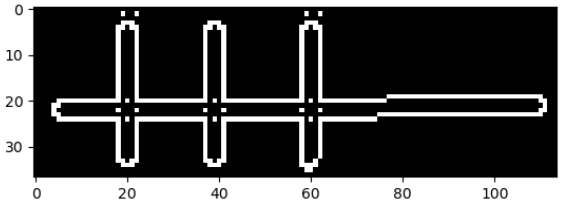


Obrázek 8: Dilatace skeletonizovaného obrázku

3.7 Segmentace

Z upraveného obrázku jsme znovu smazali kraje, aby se obrázek správně segmentoval a nebyl kvůli krajům otevřený.

Následně jsme obrázek segmentovali pomocí Canny algoritmu, který vytvořil obrys upraveného obrázku. Na takto upravený obrázek se můžeme podívat na obrázku 9.



Obrázek 9: Segmentovaný obrázek

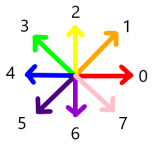
3.8 Převedení Boolean obrázku na RGB

Pro ilustrativní účely jsme vytvořili funkci, která převádí námi upravovaný obrázek na RGB variantu, kde objekt má bílou barvu s hodnotou $[255, 255, 255]$ a pozadí je černé s hodnotou $[0, 0, 0]$. Tuto variantu obrázku později využíváme pro zobrazení hran, které určují počet stehů.

3.9 Freemanův řetězový kód

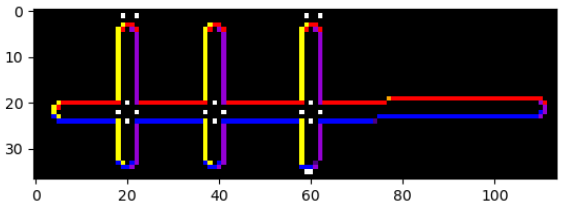
S obrysem objektu jsme mohli vytvořit *Freemanův řetězový kód*. Tento kód nám popisuje, kterým směrem se posouvají jednotlivé pixely a tvoří daný obrys kolem řezu a stehů. Nejprve jsme si navrhli kompas, kterým jsme se řídili. Kompas začíná hodnotou 0 ve směru doprava a pokračuje proti směru

hodinových ručiček. Každá další hodnota je posunutá od té předchozí o 45°. Vizuálně můžeme kompas vidět na nadcházejícím obrázku 10.



Obrázek 10: Kompas pro Freemanův řetězový kód

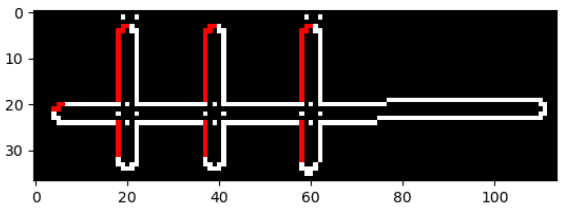
V cyklu jsme našli nejzápadnější pixel objektu, který je ale dále prvních pár pixelů. To z důvodu, kdyby zbyl v obrázku na levém kraji nějaký nesprávný objekt. V tom bodě začíná algoritmus pro řetězový kód. V cyklu prohlídneme každý pixel sousedící s naším aktuálním. Pokud je některý sousední pixel součástí objektu a není již prohlídnutý v předchozích iteracích, zapíše se směr, kterým se posouváme k dalšímu pixelu, aktuální pixel se přepíše do předchozího a nově nalezený pixel do aktuálního. Takto vytvoříme řetězový kód kolem řezu a stehů. Pokud algoritmus již nenajde nový sousední pixel, předpokládá, že jsme objeli objekt a skončí. Pro lepší představu se můžeme podívat na obrázek 11, kde graficky značíme směr každého pixelu od toho předchozího tak, jak je vyhodnotil Freemanův řetezový kód.



Obrázek 11: Obrys obrázku obarvený podle směrů ve Freemanově kódu

Nyní začíná samotné počítání stehů. K určení počtu stehů využíváme Freemanův řetězový kód, ve kterém hledáme určité vlastnosti, které nám dokáží říct, že v tomto místě se nejspíš nachází steh. Pokud se v kódu nachází směry 1,2 nebo speciální kombinace posloupností směrů 0,2 a 3 vícekrát za sebou, stoupáme nyní nejspíš po stehu. Taková posloupnost směrů by se měla pro steh objevit dvakrát. V horní části stehu na řezem a v dolní části po řezem. Proto v cyklu spočítáme, kolikrát se taková posloupnost v kódu objeví a vydělíme ji dvěma.

Každou nalezenou hranu algoritmus zároveň vykreslí do RGB obrázku červenou barvou, abychom viděli, podle kterých hran se algoritmus rozhoduje. Vykreslení můžeme pozorovat na obrázku 12.



Obrázek 12: Zobrazení hran, podle kterých algoritmus určuje počet stehů

Pokud je Freemanův řetězový kód natolik krátký, že na obrázku nejšpíš není řez správně viditelný nebo se programu nepovedlo ho správně objet, vyhodnotí ho jako neplatný obrázek a do počtu stehů zapíše -1

4 Závěr

Tato metoda pro analýzu obrázku a určení počtu stehů je hodně závislá na nastavení parametrů. Jak velkou posloupnost správných směrů potřebuje, aby vyhodnotila, že se nachází na stehu. Jak velké jádro použít při morfologických erozních operacích, aby algoritmus správně našel v obrázku stehy a řez a jak velké jádro použít při dilatačních operacích, aby správně spojil řez a všechny stehy do jednoho přehledného objektu. Které posloupnosti směrů povolit, aby algoritmus spravně identifikoval stoupající hranu na stehu. Obrázky mají navíc každý trochu jiné rozlišení. Některý je kvalitnější více a jiný méně. Proto některé parametry upravujeme dynamicky v závislosti na kvalitě obrázku. Například při vytváření jádra pro erozi stehů měníme délku jádra dynamicky s přihlédnutím na kvalitu obrázku. Kvalitnější obrázky obsahují více pixelů a proto potřebujeme pro správnou identifikaci delší jádro. Parametry jsme nastavili optimálně tak, aby každý parametr přinesl nejlepší výsledky oproti všem okolním.

Jeden problém nastal i při výběru, kterým směrem se algoritmus vydal při vytváření Freemanova řetězce. Často se stalo, že algoritmus si v zatáčkách vybral diagonální validní směr, ale poté se jakoby vrátil vertikálně nebo horizontálně zpět, místo toho aby pokračoval dál a v tom bodě se zasekl protože se v okolí nenacházel žádný dříve neprojetý pixel. Proto jsme určili, že algoritmus nejdříve hledá horizontální a vertikální směry a až poté diagonální. To zamezilo častému zasekávání algoritmu.

Na obrázku 12 můžeme vidět, že algoritmus vyhodnotil hranu řezu na levé straně jako hranu stehu. To jsme vyřešili jednoduchým zaokrouhlením dolů ve výsledném dělení počtu nalezených hran. Nezkracovali jsme potřebnou velikost posloupnosti směrů pro identifikaci hrany, protože to poté zhoršilo kvalitu výsledků. V pár případech se stalo, že algorimus v nějakém stehu našel pouze jednu hranu kvůli chybné úpravě v předešlých krocích. Vyhodnocená hrana řezu je takový opatření pro tyto případy.

V několika případech jsme objevili, že algoritmus určil podle nás počet stehů správně, ale obrázek byl chybně anotovaný a nepočítal se tedy do správných výsledků. Názvy těchto obrázků, jejich anotaci a správný výsledek podle našeho úsudku uvidíte v tabulce 1.

Název obrázku	Anotace	Správně
SA_20220620-105545_tlmnkcr9sjlm_incision_crop_0_start.jpg	5	−1
SA_20220620-110036_vp424rn48961_incision_crop_0.jpg	4	5
SA_20220707-185800_qz8chub67fgk_incision_crop_0.jpg	3	4
SA_20231011-104231_e3sdqld32ut6_incision_crop_0_start.jpg	−1	2
SA_20240221-124113_bsvosrv74isf_incision_crop_0_start.jpg	3	−1

Tabulka 1: Chybné anotace

I přes vysokou závislost na parametrech a řadě námi usouzených chybných anotací dokázal náš algoritmus správně identifikovat počet stehů na 69 obrázcích z celkových 134, což dělá úspěšnost 51.49253731343284 %. Tedy více něž polovinu všech obrázků dokázal algoritmus určit správně.