Detekce jizev a stehů

Katedra kybernetiky

KKY/ZDO

Jakub Rada

FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

Fakulta aplikovaných věd Západočeské univerzity

Květen, 2023

1 Vypracování

Semestrální práce je vypracovávána v programovacím jazyce Python s využitím knihoven Open-CV, NumPy, skimage a math. Veškerý kód použitý v této semestrální práci je možné najít na githubu zde: https://github.com/kubarada/kky_zdo. Celá semestrální práce bude rozdělena do několika experimentů, během nichž vyzkouším několik přístupů a následně vyberu jeden z těchto přístupů a pokusím se s ním danou úlohu vyřešit. Nejdříve popíši využité metody a poté provedené experimenty.

1.1 Prahování

Prahování (thresholding) je jednoduchá a často používaná technika zpracování obrazu, která se používá k převedení obrazu do binární podoby. Cílem prahování je rozdělit obraz na dvě kategorie pixelů - ty, které překračují určitý práh (threshold), a ty, které ho nepřekračují.

$$I_{\text{výstup}}(x,y) = \begin{cases} I_{\text{vstup}}(x,y) & \text{pokud } I_{\text{vstup}}(x,y) > T \\ 0 & \text{jinak} \end{cases}$$

Kde $I_{\text{vstup}}(x,y)$ představuje hodnotu pixelu vstupního obrazu na pozici (x,y),$I_{\text{výstup}}(x,y)$ představuje hodnotu pixelu výstupního obrazu na pozici (x,y) a T je prahová hodnota, kterou určujeme.

Vzorec definuje pravidlo pro přiřazování hodnot pixelů výstupního obrazu. Pokud hodnota pixelu vstupního obrazu je větší než prahová hodnota T, pak je hodnota pixelu výstupního obrazu zachována. Naopak, pokud hodnota pixelu vstupního obrazu není větší než prahová hodnota, pak je přiřazena hodnota 0, čímž se vytváří binární obraz.

Prahování je užitečná technika pro zvýraznění objektů v obraze nebo odstranění šumu. Prakticky existuje několik různých metod prahování, které se liší ve způsobu výpočtu prahové hodnoty nebo adaptivitě prahování na základě okolních pixelů.

1.2 Morfologické operace

Morfologické operace jsou důležitou součástí zpracování obrazu, které se používají pro manipulaci s tvarem, strukturou a topologií objektů v obraze. Tyto operace jsou založeny na teorii matematické morfologie, která se inspirovala biologickými procesy probíhajícími v rostlinách a živočiších. Tyto základní operace mohou být dále kombinovány a aplikovány

za sebou pro dosažení požadovaných efektů. Například eroze následovaná dilatací (nazývaná jako otevření) je užitečná pro odstranění malých objektů a zlepšení tvaru objektů, zatímco dilatace následovaná erozí (nazývaná jako uzavření) je užitečná pro vyplňování děr a spojování objektů.

1.2.1 Eroze

Eroze je základní morfologická operace, která zmenšuje objekty v obraze. Při erozi je každý pixel obrazu zkoumán a nahrazován minimální hodnotou mezi hodnotami pixelů v jeho okolí. Tím dochází k odebírání okrajových pixelů objektů a zmenšování jejich velikosti. Eroze je užitečná pro odstranění malých objektů nebo šumu v obraze.

$$(I \ominus B)(x, y) = \bigwedge_{(s,t)\in B} I(x + s, y + t)$$

Vzorec pro erozi $(I \ominus B)$ vybere nejmenší hodnotu mezi pixely vstupního obrazu I v rámci daného okolí (strukturálního elementu) B a přiřadí ji do výstupního obrazu. Tím se objekty zmenší a šum je odstraněn.

1.2.2 Dilatace

Dilatace je druhá základní morfologická operace, která zvětšuje objekty v obraze. Při dilataci je každý pixel obrazu zkoumán a nahrazován maximální hodnotou mezi hodnotami pixelů v jeho okolí. Tím dochází k rozšiřování objektů a vyplňování děr v jejich struktuře. Dilatace je užitečná pro spojování rozdělených objektů nebo vyplňování děr v obraze.

$$(I \oplus B)(x, y) = \bigvee_{(s,t)\in B} I(x - s, y - t)$$

Vzorec pro dilataci $(I \oplus B)$ vybere největší hodnotu mezi pixely vstupního obrazu I v rámci daného okolí (strukturálního elementu) B a přiřadí ji do výstupního obrazu. Tím se objekty zvětší a díry jsou vyplněny.

1.3 Skeletonizace

Skeletonizace je proces transformace binárního obrazu, který obsahuje objekty, na jejich tenký, středový tvar, nazývaný "skelet". Skelet zachovává topologickou strukturu objektů a slouží k reprezentaci jejich podstatných vlastností, jako jsou tvar, orientace a vzdálenosti.

Při aplikaci kružnicové masky se centrum kružnice přesouvá po pixelech v obraze a pro každý pixel se vyhodnocuje, zda je vhodný k odebrání. Podmínky pro odebrání pixelu závisí na konkrétní implementaci algoritmu. Obvykle se používají pravidla týkající se konfigurace sousedních pixelů a jejich hodnot.

Algoritmus s využitím kružnic je efektivní a robustní pro skeletonizaci objektů, přičemž zachovává důležité vlastnosti jako tvar, větvení a topologii objektů. Tato metoda je rozšířená a využívá se v mnoha aplikacích, jako je analýza cév, rozpoznávání znaků, detekce hran a dalších oblastech zpracování obrazu a počítačového vidění.

1.4 Canny edge detection

Cannyho detekční algoritmus je jeden z nejpopulárnějších hranových algoritmů. Jedná se o více úrovňový algoritmus, který byl vyvinut Johnem F. Cannym. Jelikož je detekce hran citlivá na šum v obraze, je prvním krokem odstranění šumu v obraze pomocí Gaussova filtru 5x5. Vyhlazený obraz je poté filtrován pomocí Sobelova jádra jak v horizontálním tak i vertikálním směru, abychom se získali první derivace v horizontálním směru G_x a vertikálním směru G_y. Z těchto dvou obrazů můžeme zjistit gradient a směr hrany pro každý pixel následujícím způsobem:

$$G = \sqrt{G_x^2 + G_y^2} \tag{1}$$

$$\theta = tan^{-1}(\frac{G_x}{G_y}) \tag{2}$$

Směr gradientu je vždy kolmý na hrany. Zaokrouhluje se na jeden ze čtyř úhlů představujících svislý, vodorovný a dva úhlopříčné směry.

Po zjištění velikosti a směru gradientu se dojde úplné kontrole obrazu, tak aby došlo k odstranění všech nežádoucích pixelů, které nemusí tvořit hranu. Za tímto účelem se u každého pixelu zkontroluje, zda je ve svém okolí lokálním maximem ve směru gradientu. Výstupem této fáze je binární obraz s tenkými hranami.

V poslední fázi dojde k rozhodnutí toho, které všechny hrany jsou skutečně hranami a které nikoli. K tomu potřebujeme dvě prahové hodnoty, $minVal$ a $maxVal$. Všechny hrany s gradientem intenzity větším než $maxVal$ jsou určeny jakožto hrany a ty, které jsou pod $minVal$, určitě nejsou hrany, takže se vyřadí. Ty, které leží mezi těmito dvěma prahovými hodnotami, jsou klasifikovány jako hrany a "nehrany" na základě jejich konek-

tivity. Pokud jsou spojeny s pixely "jistých hran", jsou považovány za součást hran. V opačném případě jsou vyřazeny.

1.5 Sobelův filtr

Sobelův filtr je založen na konvoluci obrazové matice s dvěma jádry (maskami): jedno pro detekci horizontálních hran a druhé pro vertikální hrany. Tyto jádra jsou malé matice s váhami, které se skládají z kladných a záporných čísel. Konvoluce se provádí tak, že se jádro překrývá s jednotlivými pixely obrazu a výsledkem je nový obraz, ve kterém jsou hrany zvýrazněny. Při použití Sobelova filtru se nejprve obraz převede do šedotónu. Poté se pro každý pixel aplikuje konvoluce s jádrem pro detekci horizontálních hran a jádrem pro detekci vertikálních hran. Výsledky těchto dvou konvolucí se poté kombinují, obvykle pomocí euklidovské vzdálenosti nebo aproximace, aby se získal konečný výsledek. Jedná se o účinný nástroj pro detekci hran v obraze, protože výstupní obraz obsahuje hodnoty gradientu intenzity pixelů, které se výrazně mění na místech, kde se nacházejí hrany. Horizontální hrany jsou zvýrazněny na výstupním obraze, který se získá aplikací jádra pro detekci horizontálních hran, zatímco vertikální hrany jsou zvýrazněny na obraze získaném aplikací jádra pro detekci vertikálních hran.

1.6 Hough transform

Jedná se o transformaci využívanou k detekci přímek. Aby bylo možné tuto transformaci využít, je nutné nejdříve využít hranové detekce během preprocesingu. Jak již víme, tak přímku v prostoru obrazu můžeme reprezentovat pomocí dvou proměnných například s využitím polárních souřadnic následovně.

$$y = (-\frac{\cos\theta}{\sin\theta})x + \frac{r}{\sin\theta} \tag{3}$$

Po vyjádření r vypadá rovnice následovně.

$$r = x\cos\theta + y\sin\theta \tag{4}$$

Obecně můžeme pro každý bod (x_0, y_0) definovat množinu přímek, jež tímto bodem prochází jako:

$$r_\theta = x_0\cos\theta + y_0\sin\theta \tag{5}$$

Což znamená, že každá dvojice r_θ, θ reprezentuje všechny přímky, které procházejí bodem (x_0, y_0) . Pokud pro daný bod (x_0, y_0) sestrojíme množinu přímek , která jím prochází, tak získáme sinusoidu. Tuto operaci provedeme pro všechny body v obraze. Pokud se sinusiody dvou různých bodů protínají v bodě $(\theta, -r)$, tak to znamená, že oba body patří ke stejné přímce. V podstatě to znamená, že přímku lze najít pomocí počtu průsečíků mezi křivkami. Čím více bodů se protíná, tím více bodů má přímka, kterou tento průsečík představuje. Houghova transformace v podstatě sleduje průsečíky křivek každého bodu v obraze. Pokud je počet průsečíků vyšší než určitý práh, prohlásí jej za přímku s parametry $(\theta, r\theta)$ průsečíku bodů.

1.7 Lineární regrese

Lineární regrese je statistická metoda pro modelování vztahu mezi jednou nezávislou proměnnou (x) a závislou proměnnou (y). Cílem je nalézt nejlepší lineární aproximaci tohoto vztahu pomocí přímky. Lineární regresi lze vyjádřit vzorcem:

$$y = \beta_0 + \beta_1 \cdot x + \varepsilon$$

Kde y je závislá proměnná (vysvětlovaná proměnná), x je nezávislá proměnná (vysvětlující proměnná), β_0 je intercept (koeficient přímky pro $x = 0$), β_1 je regresní koeficient (sklon přímky), ε je náhodná chyba.

Cílem lineární regrese je minimalizovat reziduální člen ε, což znamená, že se snažíme nalézt takové hodnoty β_0 a β_1, které nejlépe aproximují vztah mezi x a y.

1.8 DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) je algoritmus pro shlukovou analýzu, který se používá k identifikaci shluků v datovém prostoru na základě hustoty datových bodů. Byl vyvinut v roce 1996 Martinem Esterem, Hans-Peterem Kriegelem, Jörgem Sanderem a Xiaowei Xuem.

Princip DBSCANu spočívá v tom, že vyhledává oblasti ve vstupním prostoru s vysokou hustotou bodů a rozlišuje je od oblastí s nízkou hustotou. Algoritmus nepotřebuje předem určit počet shluků, což ho činí vhodným pro data s neznámým počtem shluků nebo s různou hustotou.

DBSCAN funguje na základě dvou parametrů: epsilon (ε) a minimálního počtu sousedů (MinPts). Epsilon určuje maximální vzdálenost, do které jsou považovány body za sousedy, zatímco MinPts určuje minimální počet bodů, které musí být v blízkosti daného bodu, aby byl považován za jádrový bod shluku.

Proces algoritmu DBSCAN je následující:

- Vyberte náhodný nezpracovaný bod z datového souboru.
- Pokud je tento bod jádrovým bodem, vytvořte nový shluk a přidejte ho do výsledné shlukové struktury.
- Expandujte shluk tím, že přidáte všechny dostupné sousední body, které jsou blíže než epsilon. Pokud je sousední bod také jádrovým bodem, expandujte dál do jeho sousedů.
- Pokud již nejsou žádné nové body k expanzi, označte zpracované body jako šum nebo okrajové body.
- Opakujte kroky 1 až 4, dokud nejsou zpracovány všechny body.

Výstupem algoritmu DBSCAN je kolekce shluků, které se liší v jejich velikosti, tvaru a hustotě. Shluky jsou definovány jako souvislé oblasti s vysokou hustotou, které jsou odděleny od oblastí s nízkou hustotou. Tento algoritmus má několik výhod, jako je schopnost detekovat shluky různých tvarů a velikostí, robustnost vůči šumu a schopnost odhalit shluky s různou hustotou. Nicméně, vyžaduje volbu správných hodnot epsilon a MinPts, což může být obtížné v praxi.

2 První experiment

V prvním experimentu jsem využil kombinace Cannyho hranového detektoru a Houghovi transformace. Využil jsem již implementovaných metod python knihovny OpenCV. Nejdříve jsem načetl vstupní obraz a převedl ho na šedotón.

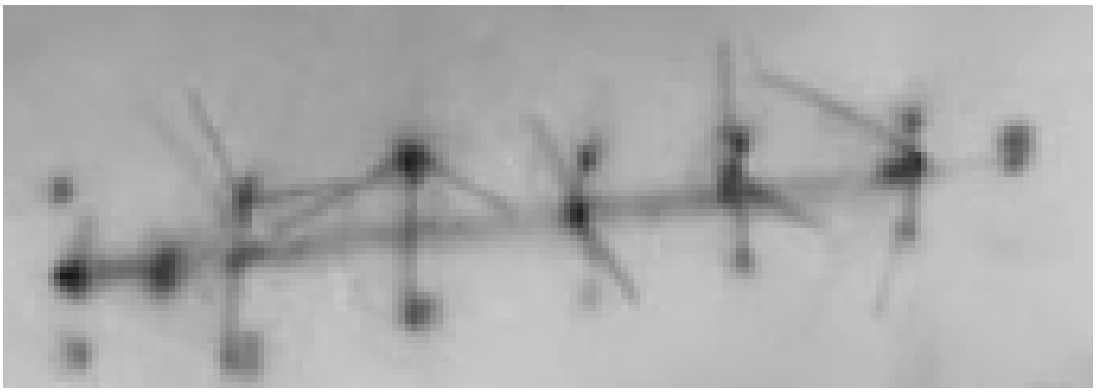

Figure 1: Vstupní šedotónový obraz

Poté jsem využil již implementované funkce Cannyho hranového detektoru a předložil mu vstupní šedotónvý obraz.

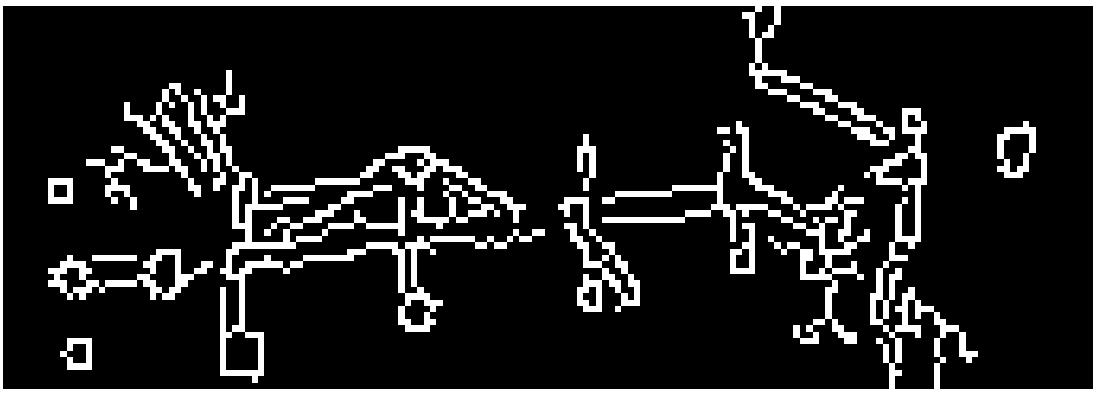

Figure 2: Obraz po aplikaci Cannyho hranového detektoru

Výstup z tohoto detektoru jsem poté ještě upravil pomocí morfologických operací. Začal jsem operací otevření a poté jsem aplikoval erozi pro odstranění zbylého šumu a dilataci pro prodloužení linek. Pro každou operaci bylo využito jiného jádra.

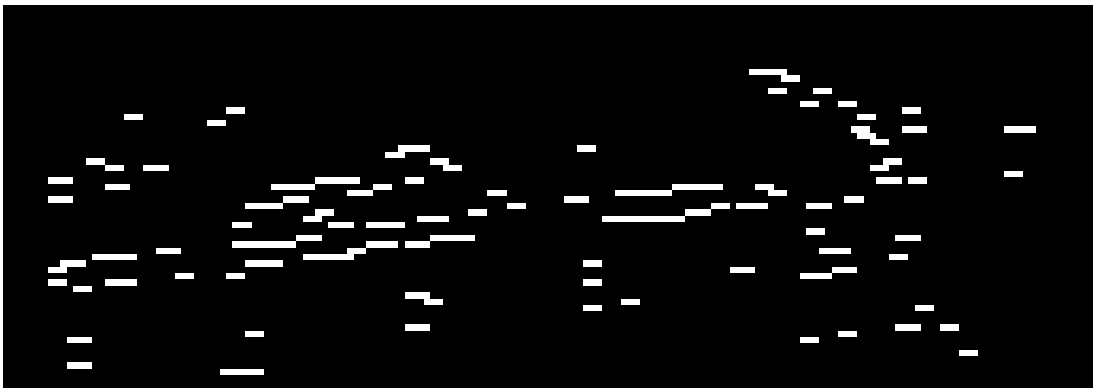

Figure 3: Obraz po aplikaci morfologických operací

Na závěr jsem využil Houghovu transformaci. Konkrétně byla využita probabilistická implementace Houghovi transformace z OpenCV. Vstupní parametry pro hledání úseček byly určeny čistě experimentálně bez jakékoliv závislosti na vstupní obraz. Výstupem této metody byly nalezené úsečky, které jsem následně vykreslil do původního obrazu.

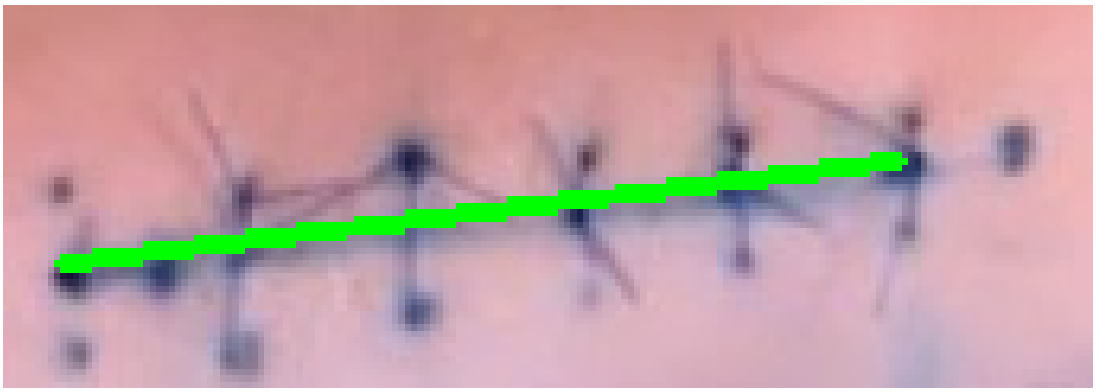

Figure 4: Výstup Houghovi transformace

Na první pohled je vidět, že dochází k velmi přesné detekci jizvy. Bohužel kombinace těchto metod je velmi náchylná na nastavení vstupních parametrů. Na dalším obrázku je vidět, že již při drobné úpravě parametrů dochází k nadbytečné detekci.

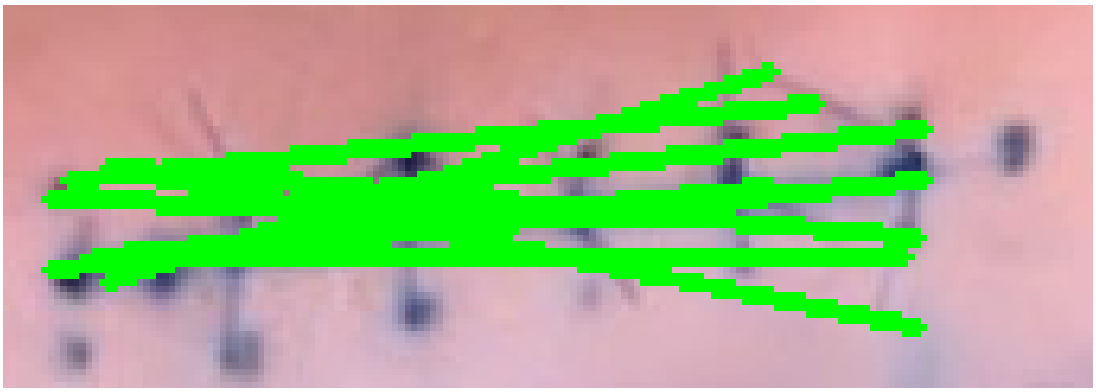

Figure 5: Nadbytečná detekce po úpravě parametrů

Zároveň tato kombinace metod s konstatními parametry Houghovi transformace nebyla příliš robustní nad ostatními vstupními obrazy. Jinak řečeno, metoda byla účinná při ručním přenastavení parametrů pro každý vstupní obraz, ale nedokázala robustně detekovat jizvy ve všech vstupních obrazech, neboť místy docházelo k nadbytečné detekci, jindy k pouze částečné detekci a v určitých případech tato metoda nedokázala detekovat jizvu vůbec. Kombinace těchto metod mi tedy poskytla poměrně dobrý počáteční vhled do úlohy. V dalších experimentech se opět pokusím využít Houghovi transformace s optimálnějším nastavením vstupních parametrů.

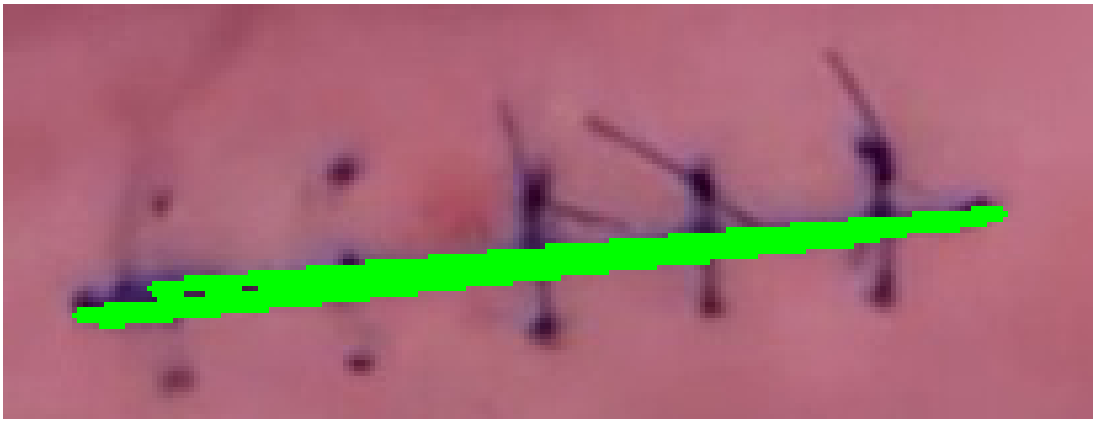

Figure 6: Detekce na různých vstupních obrazech

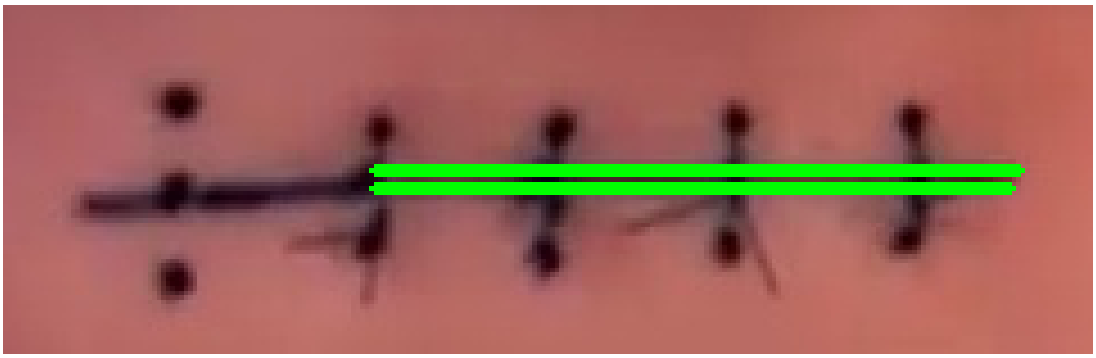

Figure 7: Detekce na různých vstupních obrazech

3 Druhý experiment

Ve druhém experimentu jsem se pokusil využít metodu skeletonizace. Nejdříve jsem načetl vstupní obraz v šedotónu. Zároveň jsem v tomto experimentu využil knihovnu scikit-image. Po načtení vstupního obrazu je možné si všimnout, že knihovna scikit-image využívá jiných barev než knihovna OpenCV, na funkčnosti to však nic nemění.

Figure 8: Obraz načtený pomocí knihovny scikit-image

Po načtení vstupního obrazu jsem využil již implementovanou metodu skeletonize(). Po vykreslení výsledku skeletonizace je možné vidět, že tato metoda nefunguje příliš spolehlivě. Tuto metodu jsem ozkoušel na většině obrazů ze vstupního datasetu, ale ve velké většině případů metoda nedokázala vydetekovat vůbec nic, a proto jsem se rozhodl v tomto experimentu nepokračovat.

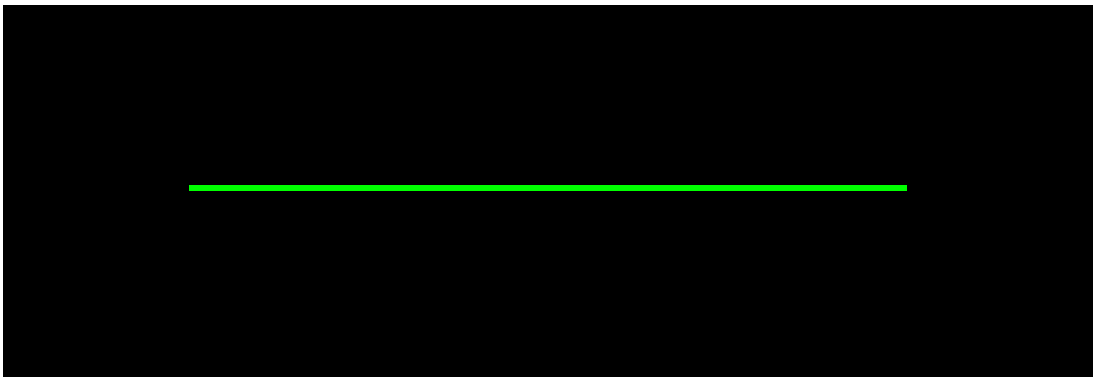

Figure 9: Detekovaný skeleton

4 Třetí experiment

Ve třetím experimentu jsem využil kombinace prahování a Houghovi transformace pro detekci jizev a stehů. Opět jsem začal načtením vstupního obrazu v šedotónu, který jsem pomocí metody adaptivního prahování z knihovny OpenCV naprahoval. Ozkoušel jsem celkem tři metody prahování. Nejdříve jsem použil klasické prahování s experimentálním určením vhodného prahu.

Figure 10: Prahování s experimentální určením prahu

Jednalo se ovšem o velmi zdlouhavý proces, který vyžadoval velké úsilí pro určení optimálního prahu. Pokračoval jsem tedy využitím metodou adaptivního prahování. Tato metoda využívá dvou různých postupů pro určení optimálního prahu. Prvním postupem je 'cv.ADAPTIVE_THRESH_MEAN_C', který počítá prahovou hodnotu, jako střední hodnotu sousedního okolí mínus konstanta, která do metody vstupuje jako další parametr.

Figure 11: Adaptivní prahování s využitím'cv.ADAPTIVE_THRESH_MEAN_C'

Na první pohled je vidět, že s minimální úsilím byl obraz naprahován velmi uspokojivě pro další využití. Druhým postupem pro určení optimálního prahu byl 'cv.ADAPTIVE_THRESH_GAU který počítá optimální práh jako gaussovsky vážený součet hodnot sousedních hodnot mínus konstanta, která opět vstupuje do metody jako další parametr. Zároveň jsem se pro další zpracování rozhodl využít inverzního prahování.

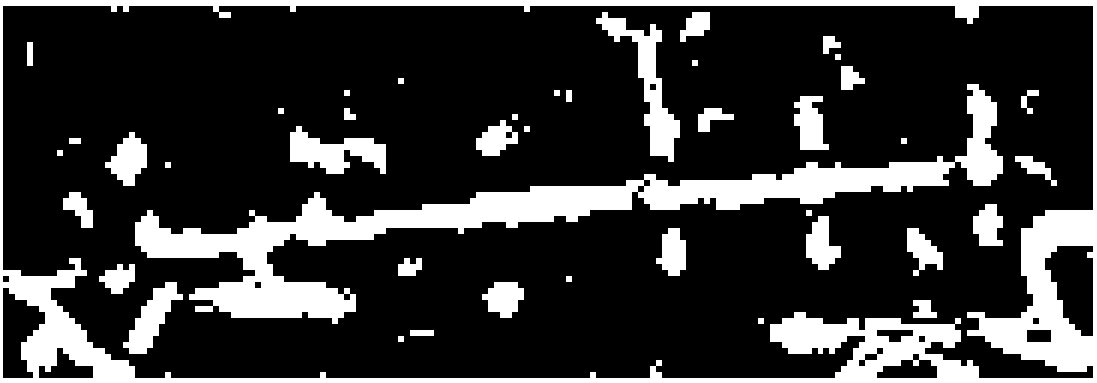

Figure 12: Adaptivní prahování s využitím'cv.ADAPTIVE_THRESH_GAUSSIAN_C'

Na posledním výsledném obraze je vidět, že tato metoda funguje nejlépe, neboť dochází k největšímu odstranění šumu. Výstup tohoto prahování proto použiji pro další zpracování. Tento experiment jsem rozdělil do dvou částí - na detekci jizvy a detekci stehů. Začal jsem detekcí jizev. Experimentálně jsem poté přišel na to, že u menších obrazů dochází k horší detekci, při aplikaci Houghovi transformace. Proto jsem menší vstupní obrazy zvětšoval pomocí upscalingu. Tento zvětšený obraz jsem poté předložil probabilistické Houghově transformaci. Tímto způsobem jsem získal koncové a počáteční body detekovaných úseček . Z těchto bodů jsem poté pomocí lineární regrese aproximoval výslednou úsečku a získal tak detekovanou jizvu. Možno vidět níže.

Figure 13: Aproximovaná výsledná úsečka - jizva

Pokračoval jsem detekcí stehů. Zde jsem stejně jako výše nejdříve použil prahování a rescalingu. Pokračoval jsem však morfologickými operacemi erozí a dilatací, tak abych co nejvíce zdůraznil vertikální linky v obraze pro co nejlepší následnou detekci.

Figure 14: Zvýrazněné vertikální linky v obraze

Následně jsem aplikoval probabilistickou Houghovu transformaci, tentokrát implementovanou knihovnou scikit-image, která mi vrátila koncové a počáteční body detekovaných vertikálních úseček. Nad získanými body jsme zavolal algoritmus DBSCAN, který mi na základě vstupních parametrů poshlukoval získané úsečky do několika shluků. Jako parametr epsilon jsem zvolil 20% šířky vstupního obrazu. Po rozdělení úseček do jednotlivých shluků jsem je pomocí knihovny NumPy zprůměroval tak, abych pro každý shluk získal právě jednu úsečku. Níže jsou vidět příklady kvalitně detekovaných jizev a stehů.

Figure 15: Kvalitně detekované stehy + jizva

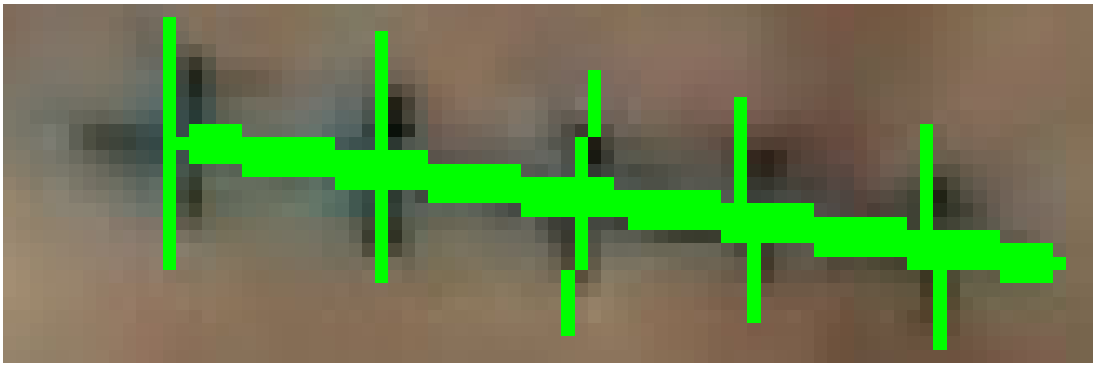

Figure 16: Kvalitně detekované stehy + jizva

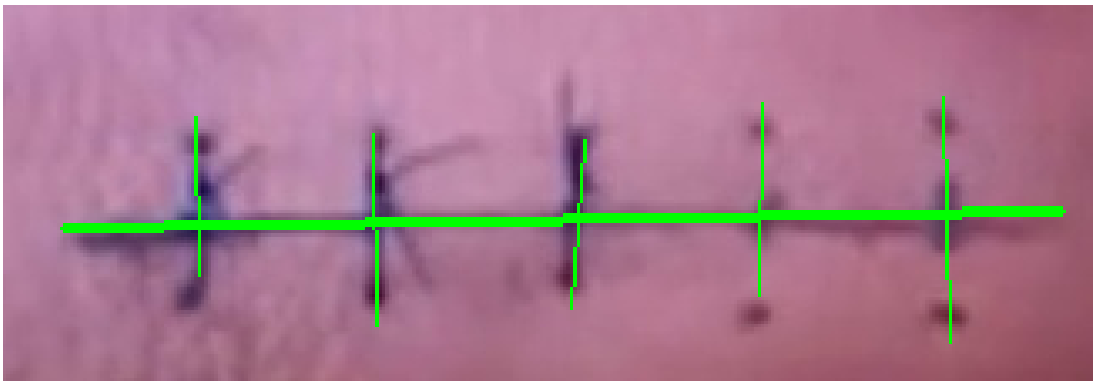

Figure 17: Kvalitně detekované stehy + jizva

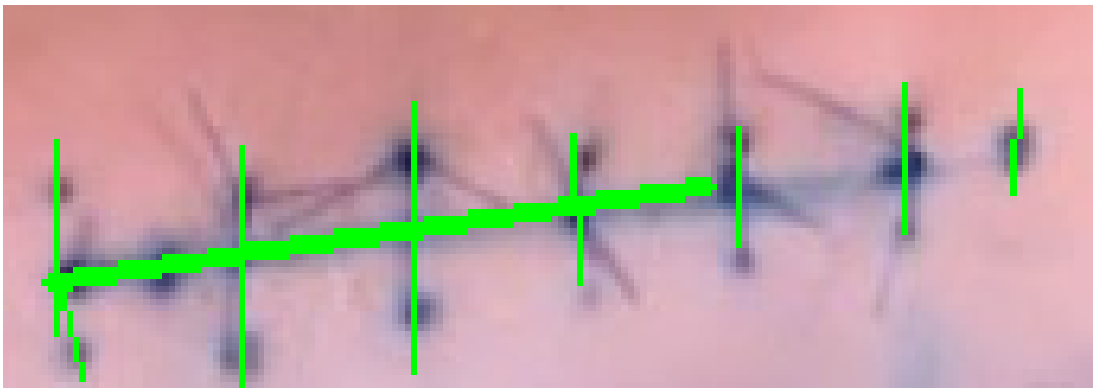

Figure 18: Nekvalitně detekované stehy + jizva

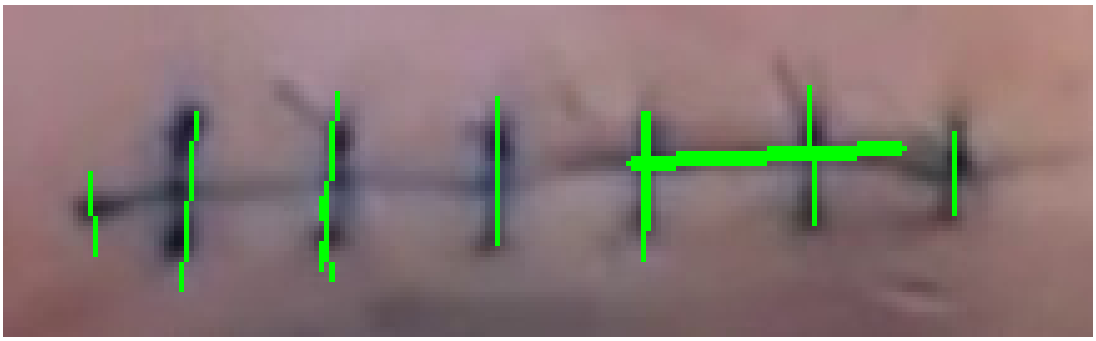

Figure 19: Nekvalitně detekované stehy + jizva

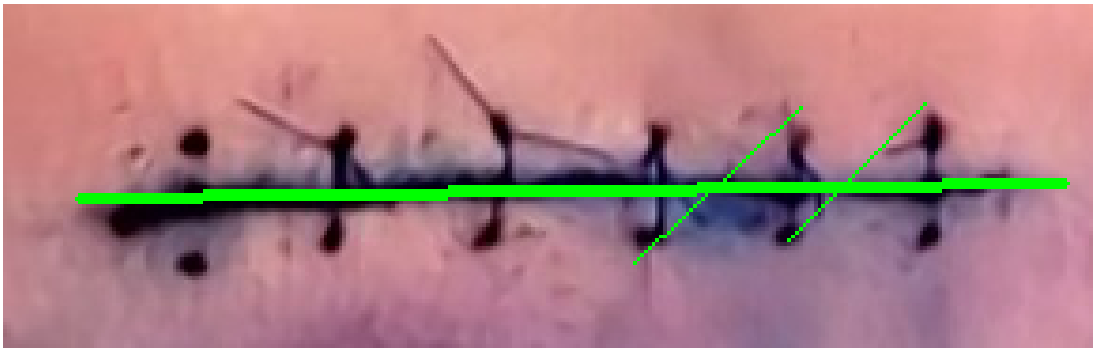

Figure 20: Nekvalitně detekované stehy + jizva

4.1 Spuštění programu

Program pro detekci jizev a stehů se dá spustit z příkazové řádky a to pomocí následujícího příkazu.

```
cd kubarada/kky_zdo/src
python run.py
```

V takovémto případě se spustí demo programu s jedním obrazem. Pro spuštění programu s vlastními argumenty bez vizualizace se dá využít následující příkaz.

```
cd kubarada/kky_zdo/src
python run.py output.json incision001.jpg incision005.png incision010.JPEG
```

Kde argument na prvním místě je název výstupní .json soubor a zbylé argumenty oddělené mezerou jsou cesty ke vstupním obrazům. Pro vizualizace poté slouží následující příkazy.

```
cd kubarada/kky_zdo/src
python run.py output.json -v incision001.jpg incision005.png incision010.JPEG
```

5 Závěr

V semestrální práci jsem si ozkoušel různé metody počítačového vidění a umělé inteligence. Mým úkolem bylo detekovat jizvy a stehy ve vstupním obraze. V celé práci bylo využito pouze klasických metod počítačového vidění. Neuronových síti jsem nevyužil, neboť vstupní data byla anotována různými lidmi, což bohužel nezaručilo konzistenci anotací, neboť v datasetu bylo více typů stehů a každý jednotlivec tyto stehy anotoval jinak. Jako hlavním komponentem pro detekci byla Houghova transformace a prahování v kombinaci s různými metodami pro úpravu binárního obrazu a následného postprocessingu. Získané výsledky hodnotím pozitivně, neboť pro dostatečně kvalitní vstupní obrazy je detekce téměř bezchybná.