VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Detekce významné oblasti v obraze

Salient Region Detection in an Image

2015 Pavel Kutáč

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Pavel Kutáč**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Detekce významné oblasti v obraze
Salient Region Detection in an Image

Zásady pro vypracování:

Cílem práce je implementace vybrané metody detekce významné oblasti v obraze. V takovýchto oblastech se pak mohou nalézat objekty, které mohou být zajímavé z pohledu dalšího zpracování.

Ve své práci proveďte:
1. Seznamte se s tématikou detekce významných objektů v obraze.
2. Implementujte jednu metodu pro detekci významných objektů v obraze.
3. Metodu otestujte na vybrané množině dat.
4. Svou implementaci zdokumentujte a zhodnoťte dosažené výsledky.

Seznam doporučené odborné literatury:

[1] Sojka, E.: Digitální zpracování a analýza obrazu, VŠB - TUO, 2001
[2] Fu, K., Gong, C., Gu, I., Y., H., Yang, J.: Geodesic saliency propagation for image salient region detection, ICIP 2013, 2013

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Gaura**

Datum zadání: 01.09.2014
Datum odevzdání: 07.05.2015

VYSOKÁ ŠKOLA BÁŇSKÁ - TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * OSTRAVA *

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 12. dubna 2015 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 12. dubna 2015 .....

Rád bych poděkoval panu Ing. Janu Gaurovi, vedoucímu mé bakalářské práce, za pomocné rady a tipy.

Abstrakt

Tato práce popisuje konkrétní implementaci algoritmů, sloužících k nalezení významné oblasti obrazu. Program zpracuje daný obrazový vstupní soubor, provede jednotlivé operace a poté vytvoří ořezovou masku. V masce je zvýrazněna pouze ta část obrazu, která je pro daný snímek charakteristická.

Ukážeme si postupně vytvoření superpixelů, neboli rozdělení obrazu na atomické části, které vnímáme podobně. Následně vypočteme konvexní obal bodů získaných pomocí detekce rohů, které nám určí hrubou představu o důležitých částech obrazu. Vytvoření masky je dokončeno výpočtem nad jednotlivými superpixely, který kombinuje informace z konvexního obalu a minimální vzdáleností v CIELab prostoru mezi danými superpixely. Pro určení minimální vzdálenosti je použit Dijkstrův algoritmus, kde vrcholy jsou středy jednotlivých superpixelů.

Implementaci jsem prováděl v jazyce C++ za pomocí knihovny OpenCV, která je navíc volně k užití. Tato knihovna obsahuje velké množství funkcí, které implementaci značně zjednodušují a navíc jsou optimalizované pro co nejrychlejší vykonání operací.

Klíčová slova: analýza obrazu, počítačové vidění, superpixel, OpenCV, ořezová maska, CIELab

Abstract

This work describe implementation of algorithms for searching salient object in image. Program analyzes input image, proceed it through algorithms and save clipping mask, where only salient part of image is highlighted.

We describe step by step creating superpixels, which are atomic part of image. Followed by creating convex hull of point found by Harris corner detector. This give us first coarse imagine of salient object. Mask is finished by counting transmitting energy between superpixels, which distance is counted by Dijkstra algorithm to find shortest path.

Implementation I made in C++ using OpenCV library, which is also free for academic and commercial use. This library include lots of functions, which help us with implementation, and there are optimized for best performance.

Keywords: image analysis, computer vision, superpixel, OpenCV, clipping mask, CIELab

Seznam použitých zkratek a symbolů

BSD	–	Berkley Software Distribution
SLIC	–	Simple Linear Iterative clustering
CIE	–	Commission internationale de l'éclairage
Lab	–	Barevný model *Lightness*,*a* a *b*
RGB/BGR	–	Barevný model *Red*,*Green* a *Blue*
GPS	–	Global Positioning System
SIFT	–	Scale-invariant feature transform

Obsah

Seznam obrázků

1 Úvod

Počítačové vidění je v dnešní době velmi důležitou oblastí počítačových věd. Dlouhou dobu bylo kvůli vysokým výpočetním nárokům zapovězenou oblastí a využívaly se tyto techniky pouze ve vojenském a lékařském odvětví. Zvyšující výkon výpočetních jednotek umožňuje obraz analyzovat i na zařízeních každodenního života. V dnešní době i kompaktní fotoaparáty umějí rozpoznávat obličeje a zaostří přímo na ně. Počítačová analýza obrazů již pronikla i do automobilů, které nám hlásí, kolem jakých značek jsme projeli, sledují vodorovná značení a upozorní na změnu jízdního pruhu bez použití směrových blikačů. V mé práci aplikuji algoritmy pro zvýraznění významné oblasti v obrazu. Tato technika by mohla být užitečná pro další analýzu jako je detekce objektů v obraze nebo jeho vyhledávání, segmentace obrazu či zaostření a sledování objektů.

Celá cesta k docílení výsledku se dá rozdělit na několik samostatných bodů, které postupně popíšu. V kapitole 2 si vysvětlíme vytvoření superpixelů. Je to část obrazu, kterou vnímáme podobně a má stejný význam. Pro zjednodušení ji tedy můžeme nahradit homogenní oblastní a vyplníme ji jedinou barvou. Ve zmíněné metodě SLIC [2] navíc všechny superpixely mají podobnou velikost a své středy, které využijeme i v dalším zpracování.

Následně v kapitole 3 spočítáme ořezovou masku, která nám určí důležitost jednotlivých částí. Ořezová maska je obraz stejně velký jako originál, ale pouze ve stupních šedi. Aplikace masky v obrazu zprůhlední části, které jsou černé. Čím je větší intenzita bílé, tím méně průhledný daný pixel bude.

Dopracování k ořezové masce má mnoho kroků. Začneme výpočtem globálního kontrastu v podkapitole 3.1 jednotlivých superpixelů vůči všem ostatním. Již tento krok nám může hodně naznačit, která část obrazu je významná. Pro zlepšení však provedeme detekci rohů pomocí Harrisova detektoru, viz podsekce 3.2, a poté nad těmito body vytvoříme konvexní obal. Jeho využití je popsáno v podkapitole 3.3. Ten nám určí hrubou zájmovou oblast obrazu, zda superpixel je vně či uvnitř obalu.

Protože významná oblast může ale zasahovat i mimo tento obal, mezi superpixely budeme provádět výpočet geodetické vzdálenosti, jejíž nejkratší cestu získáme pomocí Dijkstrova algoritmu. Nalezení sousedních superpixelů jsem prováděl metodou semínkového vyplňování, popsané v kapitole 3.4.

Na závěr si v sekci 5 spustíme náš algoritmus na několika obrázcích, abychom získali větší množství výsledků ke zhodnocení, které nalezneme v sekci 6. Zde si zhodnotíme problémy při implementaci popsaných algoritmů, především chybějící zmínění některých funkcí a částí, ale i rozdíly mezi reálnými a popsanými výsledky.

2 Vytvoření superpixelů

Detekce významné oblasti [1] je založena na obrazu, který již je segmentovaný na jednotlivé superpixely. Z toho důvodu, jak již bylo zmíněno dříve, začneme implementací procesu vytvoření superpixelů. To je popsáno v samostatném článku [2].

Obrázek 1 Vstupní obrázek

2.1 Definice superpixelu

Superpixel je část obrazu, ve které jsou všechny pixely a jejich barva natolik podobné, že je můžeme vyplnit jednotnou barvou, nejčastěji průměrnou hodnotou všech barevných kanálů, čímž vznikne homogenní superpixel. Tato transformace nám zdrojový obraz zjednoduší, protože budeme k většímu množství pixelu přistupovat, jako by se jednalo o jediný pixel. Superpixel by měl být atomický ale také by měl kopírovat kontury obrazce, aby základní rysy zůstaly zachovány.

Ve zmíněné práci [2] se navíc snaží zachovat shodnou velikost superpixelů navzájem mezi všemi ostatními.

Implementace algoritmu SLIC požaduje 2 vstupní parametry:

1. požadovaný počet superpixelů,
2. parametr m v rozmezí 1–40.

Parametr m určuje důležitost dodržení obrazových kontur versus pravidelnost superpixelů. Čím je m větší, tím pravidelnější tvar superpixel bude mít. Čím je hodnota m menší, tím více budou dodrženy obrazové rysy.

Pro svou implementaci jsem použil největší možnou hodnotu, tj. $m = 40$. Přes vstupní parametry lze při spouštění programu pomocí přepínače *-M* konstantu nastavit dle vlastního uvážení, pokud nepřekročíme povolený rozsah hodnot. Více o vstupních parametrech v sekci 4.

Na rozdíl od parametru m je požadovaný počet superpixelů značený K, povinný argument. Zadává se jako celé číslo větší než 0. Přesný počet však nemusí být dodržen, jak se dozvíme níže.

Obrázek 2 Výstupní obrázek s vygenerovanými superpixely

2.2 Určení středu superpixelů

Vypočteme proměnnou S, viz rovnice (1), kde w odpovídá šířce a h výšce obrazu, K poté požadovaném počtu superpixelů. Hodnota proměnné S je vzdálenost v pixelech mezi jednotlivými středy. První střed umístíme na souřadnice $[S, S]$, další pak v pravidelných krokových intervalech délky S ve vodorovném i svislém směru.

$$S = \sqrt{\frac{wh}{K}} \tag{1}$$

Tímto postupem bychom mohli dosáhnout stavu, kdy střed superpixelu bude ležet velmi blízko okrajům obrazu. Abychom zabránili této situaci, poslední pozice středu může být maximálně $S/2$ px od pravého, resp. spodního okraje pro vodorovný, resp. svislý směr. Z tohoto důvodu nemusíme ve výsledné mřížce mít shodný počet superpixelů jaký jsme požadovali.

$$G(x,y) = \|\mathbf{I}(x+1,y) - \mathbf{I}(x-1,y)\|^2 + \|\mathbf{I}(x,y+1) - \mathbf{I}(x,y-1)\|^2 \tag{2}$$

Pro každý střed ještě musíme vypočítat nejlepší pozici v okolí 3×3 pixely pomocí výběru lokálního minima, který provedeme výpočtem (2). Tento výpočet se musí provést celkem $9\times$ a pixel s nejnižší hodnotou G určíme jako nový střed. Provedením této operace snižujeme pravděpodobnost umístění středu na zašuměný pixel a zvyšujeme účinnost metody.

Ve výpočtu (2) G znamená hodnotu lokálního minima v daném pixelu na souřadnicích $[x,y]$, $\mathbf{I}(x,y)$ značí *Lab* vektor na souřadnicích $[x,y]$. $\|\ldots\|$ je *L2* norma, v našem případě tedy $\sqrt{(l_1-l_2)^2+(a_1-a_2)^2+(b_1-b_2)^2}$

2.3 Výpočet vzdálenosti pixelů

Pro další použití je potřeba si definovat, jak počítat vzdálenost mezi pixely. Důležité je zmínit, že veškeré výpočty probíhají v barevném prostoru CIELab [3], tomu odpovídají ve výpočtech proměnné l, a a b. Proměnné i a j určují indexy superpixelů, mezi kterými vzdálenost počítáme. Zde jsem narazil na problém, protože jsem našel více verzí odborného článku [2] a v každém byl zmíněn jiný vzorec.

$$\begin{aligned} d_{lab} &= \sqrt{(l_j-l_i)^2+(a_j-a_i)^2+(b_j-b_i)^2} \\ d_{xy} &= \sqrt{(x_j-x_i)^2+(y_j-y_i)^2} \\ D &= d_{lab} + \frac{m}{S} d_{xy} \end{aligned} \tag{3}$$

$$\begin{aligned} d_{lab} &= \sqrt{(l_j-l_i)^2+(a_j-a_i)^2+(b_j-b_i)^2} \\ d_{xy} &= \sqrt{(x_j-x_i)^2+(y_j-y_i)^2} \\ D &= \sqrt{d_{lab}^2 + \left(\frac{d_{xy}}{S}\right)^2 m^2} \end{aligned} \tag{4}$$

V těchto výpočtech vidíme, že proměnné d_{lab} a d_{xy} jsou stejné, závěrečný výpočet D se však liší. Při využití postupu v rovnici (3) výsledný obrázek neměl superpixely ani pravidelné, ani homogenní jak lze vidět na obrázku 3.

Pro tuto ukázku jsem využil jiný obrázek, na kterém je problém při počítání vzdálenosti a poté i rozdíl s využitím 32bitového rozsahu kanálu v sekci 2.6 zřetelnější.

Protože jsem stále nenacházel chybu v mém programu, rozhodl jsem se najít již hotová řešení [2]. V nalezené implementaci [7] jsem našel počítání vzdáleností výpočtem (4), ten ale neodpovídal tomu, co bylo zmíněno v článku, který jsem měl stažený. Při zkoumání kódu jsem objevil i další nezmíněné funkce.

Začal jsem tedy pátrat po dalších verzích článku [2], až jsem našel originální článek. V něm na rozdíl od původně nalezeného byl text orientován do 2 sloupců a obsahoval mnoho dalších informací a vysvětlení. I přes aplikaci nového vzorce (4) obrázek 4 stále neodpovídal výsledku ukázaném v odborné práci, přesto lze pozorovat větší pravidelnost superpixelů. Dalšího zlepšení jsem dosáhl pomocí níže uvedených postupů v sekci 2.6 a 2.7.

Obrázek 3 Přiblížený výřez obrázku s počítáním vzdálenosti výpočtem (3)

Obrázek 4 Přiblížený výřez obrázku s počítáním vzdálenosti výpočtem (4), lze pozorovat větší pravidelnost superpixelů

2.4 Výpočet nových středů a oblasti superpixelů

Po rozmístění středů a definici počítání vzdálenosti můžeme začít formovat superpixely. Budeme postupně procházet oblast v okolí středů superpixelů, která má velikost $2S \times 2S$ pixelů. Pozici a barvu středu si značme jako $5D$ vektor $C_k = [l_k, a_k, b_k, x_k, y_k]^T$, kde $k = [1, K_{real}]$ je index středu. K_{real} je skutečný počet superpixelů, protože jak bylo zmíněno v podsekci 2.2, výsledný počet superpixelů může být nižší, než bylo požadováno. Proměnná S zde odpovídá kroku získanému dříve ve výpočtu (1).

Každému pixelu v dané oblasti vypočteme podle rovnice (4) jeho vzdálenost vůči středu a rovněž si u pixelu uložíme index k. Poté se přesuneme k dalšímu středu a postup opakujeme. Pokud vzdálenost u počítaného pixelu je menší, než kterou jsme si uložili v předchozím cyklu, pixel přiřadíme do nového superpixelu. Ten poznáme podle uložených indexů středů k. Všechny pixely, které náleží stejnému středu, tvoří jeden superpixel.

Pokud přiřadíme všechny pixely jejich středům, je potřeba vypočítat každému středu novou pozici a barvu. Novou pozici určíme jako průměr souřadnic x_k a y_k všech pixelů, které náleží do daného superpixelu s indexem k. Obdobný výpočet provedeme pro no-

vou barvu středu, nesmíme však zapomenout na barevný prostor CIELab, proto budeme počítat s proměnnými l_k, a_k a b_k.

Nyní je potřeba celý proces procházení všech středů a jejich okolí opakovat, dokud zbytková chyba počítaná pomocí rovnice (5) bude větší než určitá mez. Hodnotu pro mez ovšem v žádné práci nezmínili, a pravděpodobně by musela být určena pokusem. Naštěstí uvedli výsledky jejich pokusů, při kterých zjistili a stanovili dostatečný počet opakování na 10, tímto výpočet zbytkové chyby můžeme ignorovat.

$$E = \|\mathbf{P}_{\mathbf{new}}(k) - \mathbf{P}_{\mathbf{old}}(k)\| \tag{5}$$

V rovnici (5) značí $\mathbf{P}_{\mathbf{old}}(k)$ starou pozici středu k v prostoru xy, $\mathbf{P}_{\mathbf{new}}(k)$ naopak pozici novou.

2.5 Vykreslení superpixelů ve zdrojovém obrazu

Po dokončení segmentace na jednotlivé superpixely proces končí, samotné vykreslení superpixelů do obrazu je pro nás zbytečné. Důležitá informace je u každého pixelu hodnota k, která nám pixely seskupí do superpixelů.

Přesto jsem zpětné vykreslení implementoval pro jednodušší ladění chyb a získání vizuálního výsledku. Výslednou barvu superpixelu opět počítáme jako průměrnou barevnou hodnotu všech pixelů, můžeme již však počítat v RGB [4] barevném prostoru a nemusíme tedy výsledný obrázek před uložením převádět. Pro každý superpixel sečteme jeho $[B, G, R]^T$ vektory, a poté vydělíme celkovým počtem pixelů v daném superpixelu. Následně budeme procházet zdrojový obrázek pixel po pixelu, získáme si index superpixelu v jakém se nachází, a vyplníme jej barvou, kterou jsem si pro daný superpixel vypočítali a uložili.

Všimněte si, že vektor obsahuje barevné kanály v pořadí *BGR* a ne *RGB* jak jsme zvyklí. To je z důvodu implementaci za pomocí OpenCV, která obsahuje jednotlivé kanály uspořádány takto. Při našem řešení to však není příliš důležité, protože přistupujeme ke všem kanálům stejně, a na které pozici je která barva, je pro nás irelevantní.

Při spuštění programu se zpětné vykreslení neprovádí pro rychlejší běh celého programu. Pokud ovšem požadujeme segmentovaný obrázek uložit, můžeme tak provést přepínačem *-SP jmeno_souboru*. *jmeno_souboru* může být relativní i absolutní adresa, zakončena názvem souboru včetně jeho přípony.

2.6 Využití 32bitové hloubky

Jak lze vidět na obrázcích 3 a 4, superpixely stále nejsou dokonale atomické a jejich tvar není příliš pravidelný. Domníval jsem se, že chyba může být způsobena přepočtem hodnot *RGB* na *Lab*. Protože na každý barevný kanál je vyhrazeno pouze 8 bitů, hodnota l, a a b může nabývat hodnot v rozsahu $[0, 255]$. Barevný prostor *Lab* využívá jiných hodnot, l se pohybuje v rozsahu $[0, 100]$, a a b pak v rozsahu $[-128, 127]$.

Využitím 32bitové hloubky barevného kanálu bychom tyto ztráty mohli eliminovat. Při převodu *RGB* na *Lab* dojde k přepočítání hodnot, jejichž výsledek jsou desetinná čísla.

Při využití 8bitových barevných kanálů by se hodnoty musely zaokrouhlit na celá čísla. Využitím 32bitové hloubky kanálu však můžeme ukládat desetinná čísla, jediná operace, která se provede je normalizace na rozsah hodnot $[0, 1]$. Také při výpočtu průměrných hodnot barev nebude docházet po dělení k zaokrouhlení, což způsobí větší přesnost a přesnější výpočet vzdáleností.

Zlepšení si můžeme prohlédnout na obrázku 5, pokud jej porovnáme s obrázkem 4 vidíme znatelný rozdíl v celistvosti a pravidelnosti jednotlivých superpixelů. I zde však můžeme nalézt drobné chyby, kdy část superpixelu je separovaná od jeho zbytku, tudíž nejsou atomické.

V práci [2] se o problematice „orphans“, neboli sirotků, zmiňují a upozorňují, že tyto chyby můžou vzniknout. Pro opravení těchto nedokonalostí se musí využít algoritmy vynuceného připojení. Na žádnou konkrétní metodu však neodkázali ani ji neuvedli. Bylo by potřeba navíc dalšího výpočetního výkonu k provedení dané funkce, proto jsem se rozhodl jej ignorovat a využít jinou techniku popsanou v další kapitole 2.7.

Obrázek 5 Segmentace na superpixely s využitím 32bitové hloubky barev

2.7 Zlepšení pomocí Gaussova vyhlazení

Jak bylo zmíněno v předchozí kapitole 2.6, po dokončení segmentace můžou vzniknout „orphans“ segmenty, které patří do jiného superpixelu, než se kterými sousedí. I přes téměř dokonalou segmentaci na obrázku 5 se může u hodně členitých a ostrých obrazů neatomických částí objevit více i s větší plochou. Abychom z části těmto chybám zabránili, po načtení obrázku jsem jej ihned rozostřil pomocí Gaussova vyhlazení, známého také jako Gaussovo rozostření. Tato metoda se hodně využívá pro odstranění zašuměných pixelů, které mohou dělat problémy při následné analýze. Používá se taky pro zjednodušení obrazu, což je přesně náš požadavek.

Operace je poměrně rychlá a účinná, jak se můžeme přesvědčit na obrázcích 6. Zde jsem záměrně zvolil původní obrázek, protože bez Gaussova vyhlazení je v zadní části letadla vidět obrovské roztříštění superpixelů. Provedl jsem také výřez z obrázku, aby vynikla kritická část. U jednotlivých obrázků jsem demonstroval zlepšení při zvyšování průměru rozostření, jejichž hodnotu naleznete v popisku pod obrázkem.

Ani po použití Gaussova rozostření však nejsou superpixely naprosto dokonalé. Nyní bychom mohli provést vynucené spojení, jak jsem se zmínil v podkapitole 2.6. Případně další možností je zvětšit průměr rozostření, což ale nese další drobnou ztrátu výkonu.

i) - Průměr 1x1 - nedochází k rozostření

ii) - Průměr 3x3

iii) - Průměr 5x5

iv) - Průměr 9x9

Obrázek 6: Zlepšení pravidelnosti superpixelů při použití Gaussova vyhlazení

Rychlost Gaussova vyhlazovacího algoritmu se také odvíjí od jeho parametrů, které určují průměr rozostření. Algoritmus si sestaví konvoluční matici, také zvanou jádro, jejíž střed je umístěn nad aktuálně procházený pixel. Do jednotlivých polí umístí konstanty určující podíl sousedních pixelů na výsledném rozostření. Tato síla společně se vzdáleností od středu snižuje svou hodnotu. Protože matice je do všech stran stejně velká, musí být její velikost liché číslo, nesmíme zapomenout započítat střed matice.

Toto může pro některé uživatele působit matoucím dojmem, protože ve většině grafických programů se u Gassova rozostření zadává poloměr, takže není problém zadat například hodnotu 2, nebo 0. Pokud v našem případě zadáme průměr rozostření 1, k žádnému vyhlazení nedojde, neboť matice nesahá nad sousední pixely aktuálně procházeného pixelu, to můžeme vidět na obrázku 6.

Jak bylo vysvětleno, průměr musí být přirozené liché číslo, ale může se pro horizontální a vertikální směr volit různé, Gaussovo jádro nemusí být čtvercového tvaru. Při mé implementaci jsem volil výchozí hodnotu 5 pro oba směry. Nastavení lze však změnit parametrem *-GB*, více o vstupních parametrech v sekci 4.

3 Detekce významné oblasti

Pokud máme obraz segmentovaný na superpixely, jak jsme si ukázali v kapitole 2, můžeme začít s hledáním jeho významné části, jak je popsáno v článku [1]. Celý postup lze rozdělit na dílčí části popsané v následujících podkapitolách. Začneme výpočtem globálního kontrastu jednotlivých superpixelů vůči ostatním (podsekce 3.1), detekcí rohů a následného získání konvexního obalu nad těmito body (podsekce 3.2 a 3.3) a zakončíme vykreslením ořezové masky.

Výsledný obraz bude sloužit jako ořezová maska, ta je pouze ve stupních šedi a při její aplikaci dochází k zprůhledňování jednotlivých částí. Čím více je pixel masky tmavý, tím více je pixel na stejné pozici ve zdrojovém obrazu průhledný. Naopak úplně bílé pixely v masce jsou plně viditelné v původním obraze.

3.1 Výpočet globálního kontrastu

Jedním z důležitých faktorů pro další výpočty je globální kontrast jednotlivých superpixelů vůči ostatním. Tímto způsobem získáme první informaci ohledně významné oblasti. Superpixel, s velkým globálním kontrastem oproti zbylým částem, má velkou pravděpodobnost, že bude součástí významného objektu.

Globální kontrast značný F_k^{global}, kde k je index superpixelu, se vypočítá podle následujícího vzorce (6). c_k^{Lab} respektive c_j^{Lab} značí barvu superpixelu v barevném prostoru Lab na jeho středu s indexem k respektive j.

$$F_k^{global} = \sum_{j=0}^{K_{real}-1} \|c_k^{Lab} - c_j^{Lab}\|_2^2 \tag{6}$$

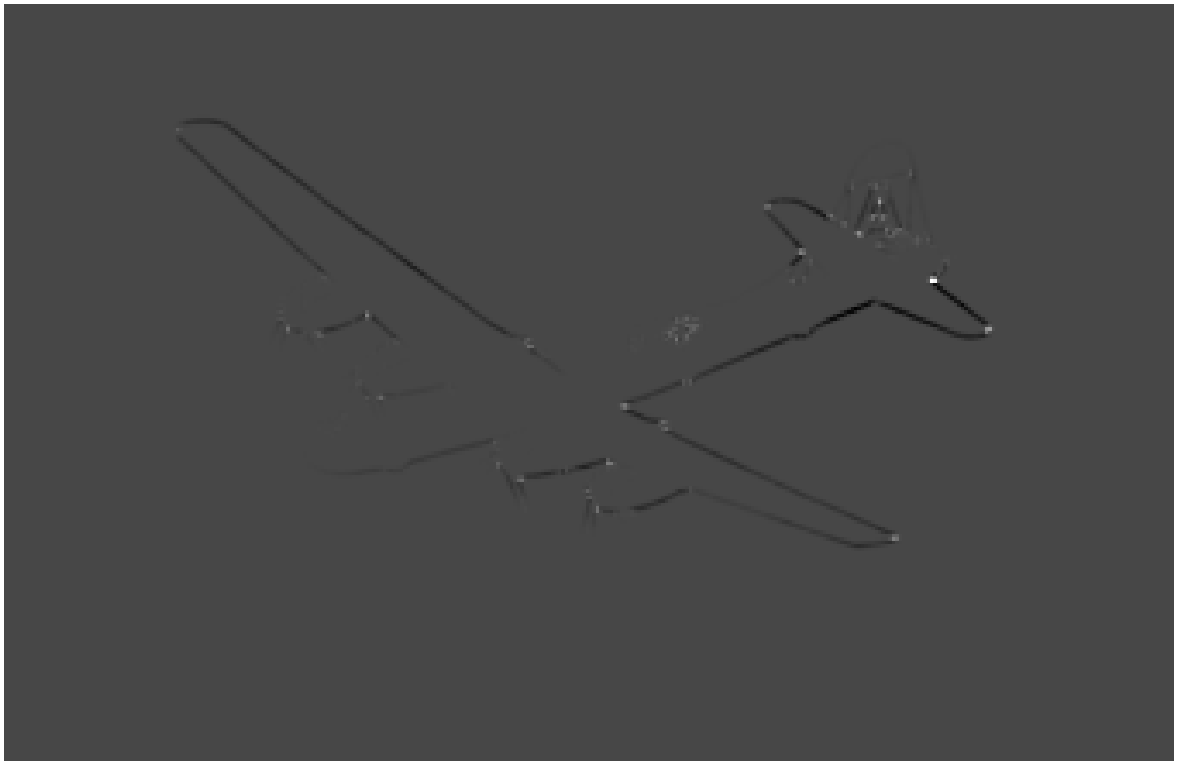
Následně se provede normalizace globálních kontrastů všech superpixelů na hodnoty rozsahu $[0, 1]$ čehož dosáhneme výpočtem (7).

$$\begin{aligned} max_{global} &= \max_{0 \leq i < K_{real}} F_i^{global} \\ min_{global} &= \min_{0 \leq i < K_{real}} F_i^{global} \\ F_k^{global} &= \frac{F_k^{global} - min_{global}}{max_{global} - min_{global}} \end{aligned} \tag{7}$$

3.2 Detekce rohů pomocí Harrisova detektoru

Při hledání významných objektů je často využita metoda detekce rohů, kterých existuje velké množství. Při své implementaci jsem využil Harrisova detektoru rohů [8]. Jeho výsledkem je matice stejně velká jako zdrojový obraz, obsahující desetinná čísla normalizována na rozsah $[0, 255]$. Výstupní obraz z Harrisova detektoru si lze prohlédnout na obrázku 7. Určení samotných rohů probíhá procházením jednotlivých bodů, a pokud hodnota v daném bodě je vyšší než určitá mez, pak se v tomto bodě nachází roh.

Bohužel tuto mez je potřeba nastavit individuálně pro každý obraz a je to čtvrtý a poslední povinný parametr při spouštění mé implementace. Všechny jsou také popsány v kapitole 4. Pokud ovšem zadáme hodnotu parametru příliš nízkou, může nastat zahlcení celého obrazu samotnými rohy. Z toho důvodu jsem v mé implementaci vytvořil kontrolu počtu nalezených rohů, a pokud jich bude více než $PL * 0.4$, ukončím běh programu s chybou. PL určím jako maximální možný počet bodů v obraze, ten se nerovná celkovému počtu pixelů, ale pouze části, ve které se rohy neodfiltrují, viz podkapitola 3.3. Přesto toto chování může být v některých případech nežádoucí a je možné tuto chybu ignorovat zapnutím přepínače *-IPL*.



Obrázek 7 Výřez výstupního obrazu z Harrisova detektoru rohů

Protože se mi nelíbila nutnost udávat při spuštění další parametr, který navíc nemusí být lehké ze začátku určit správně, chtěl jsem zkusit jiný detektor než Harrisův. Zkusil jsem velmi známý SIFT [9], protože tato metoda nevyžaduje žádné vstupní parametry. Po jejím spuštění ovšem dostanu obrovské množství klíčových bodů, a následný konvexní obal, který si popíšeme v podkapitole 3.3, by obsahoval téměř celou plochu obrazu. Toto chování je samozřejmě nežádoucí, protože klíčové body nám mají pomoci s určením významného objektu. Tímto jsem zakončil mé pokusy s metodou SIFT a vrátil se zpět k Harrisovu detektoru.

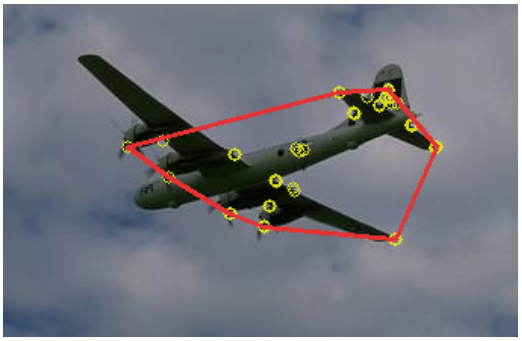
3.3 Využití konvexního obalu a získání hrubé charakteristické mapy

Pokud již máme nalezeny všechny rohy, neboli body, z předchozí podkapitoly 3.2, můžeme nad nimi vytvořit konvexní obal. Dříve než tak učiníme, musíme některé nalezené body odfiltrovat. Získané body mohou ležet velmi blízko okrajům obrazu, kde se významné objekty objevují jen zřídka, z toho důvodu je před další operaci odstraníme. V práci [1] opět nebylo řečeno, v jak velkém prostoru od okrajů nesmí body ležet. Zvolil jsem si konstantu dle svého uvážení na hodnotu $1.5S$. Proměnná S zde odpovídá počáteční vzdálenosti středů superpixelů mezi sebou, získané výpočtem (1).

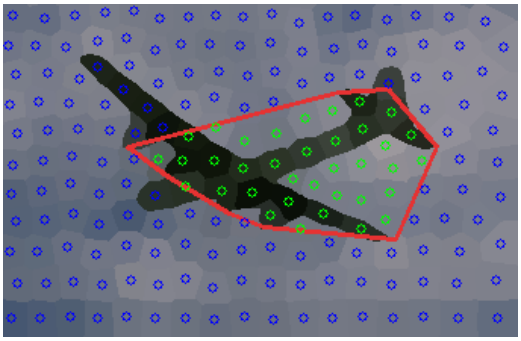
Konvexní obal je mnohoúhelník nad množinou bodů, do kterého všechny body z množiny náleží. Pokud si vybereme libovolné 2 body z množiny a povedeme mezi nimi úsečku, vždy bude ležet uvnitř nebo na okraji získaného mnohoúhelníku. S velkou pravděpodobností bude náš významný objekt, nebo alespoň jeho větší část, ležet právě v tomto konvexním obalu, jak se můžeme přesvědčit na obrázku 8.

Určení, kdy superpixel leží uvnitř či vně konvexního obalu, není v práci blíže specifikováno. Rozhodl jsem se pro nejjednodušší řešení. Pouze určím, jestli v mnohoúhelníku leží střed superpixelu, pak tam leží i samotný superpixel. Tuto informaci si ke každému superpixelu uložíme, jak je popsáno v rovnici (8), vizuálně je řešení znázorněno na obrázku 9.

$$F_k^{information} = \begin{cases} 0 & \text{Superpixel leží vně konvexního obalu} \\ 1 & \text{Superpixel leží v konvexním obalu} \end{cases} \tag{8}$$



Obrázek 8 Detekce bodů a znázornění jejich konvexního obalu



Obrázek 9 Znázornění superpixelů a jejich středů, které patří do konvexního obalu

Vysvětlení barevných útvarů na obrázcích 8 a 9:

Červený mnohoúhelník vykreslený konvexní obal,

Žluté kružnice Body nalezené Harrisovým detektorem rohů,

Zelené kružnice Středy superpixelů ležících v konvexním obalu,

Modré kružnice Středy superpixelů ležících vně konvexního obalu.

Na obrázku 9 lze pozorovat nedokonalost této metody. Plocha některých superpixelů hodně zasahuje do konvexního obalu, ale stále tam nepatří, jiné naopak do obalu patří, i když jejich větší část leží vně obalu. Bylo by možné zkoumat i další parametry, jako je vzdálenost středu od okraje mnohoúhelníku, a při nízké hodnotě superpixel zahrnout. Považuji to ale za zbytečnou komplikaci, která by potřebovala další výpočetní výkon.

Hlavní náplní článku [1] je představení nové metody globálního šíření, založené na počítání geodetické vzdálenosti, kterou si ukážeme v kapitole 3.7. Tato metoda zajistí také detekci superpixelů, které leží mimo konvexní obal, proto i problém některých superpixelů „vypadnutých ze hry“ můžeme ignorovat.

$$S_k^{coarse} = F_k^{global} F_k^{information} \tag{9}$$

Další průkopovou částí práce [1] je použití hrubé charakteristické mapy, která je tvořena výpočtem globálního kontrastu spolu s aplikací konvexního obalu. Tato mapa je dána výpočtem (9), ve kterém je $F_k^{information}$ vysvětleno ve výpočtu (8) a S_k^{coarse} značí úroveň významnosti superpixelu s indexem k.

3.4 Nalezení sousedních superpixelů pomocí semínkového vyplňování

K implementaci Dijkstrova algoritmu, zmíněném v následující kapitole 3.6 je potřeba vytvořit souvislý graf. Vrcholy grafu jsou středy superpixelů. Cesta mezi dvěma vrcholy existuje, pokud spolu superpixely sousedí, tudíž mají společné hranice.

Našim cílem nyní je vybrat vhodnou metodu ke zjištění, zdali superpixely sousedí, či nikoli. Z předchozích obrázků by některé mohlo napadnout vybrat všech osm sousedů ve čtvercové mřížce, protože superpixely jsou téměř čtvercového tvaru. Z následujícího obrázku 10 však jasně vyplývá, že ne vždy jsou superpixely pravidelné, především u složitějších obrazců. Zelené body zde značí středy superpixelů, a čáry mezi nimi cesty. Podle počtu cest a jejich cílů lze vyčíst, které superpixely spolu sousedí. Při pozorování zjistíme, že superpixel sousedí nejčastěji s pěti až šesti dalšími superpixely.

Pro nalezení sousedních superpixelů jsem se rozhodl využít metodu semínkového vyplňování, která se používá pro vyplnění ohraničené části obrazu. Začneme definováním souřadnic $[x, y]$ počátečního pixelu. Pokud je pixel na souřadnicích $[x, y]$ již vyplněný, další operace s tímto pixelem neprovádíme. Pokud pixel není vyplněn, vyplníme jej a celou operaci opakujeme pro pixely na souřadnicích $[x + 1, y]$, $[x - 1, y]$, $[x, y + 1]$, $[x, y - 1]$ v případě použití 4cestné verze algoritmu. Tato metoda ovšem uvažuje pouze 2 stavy daného pixelu:

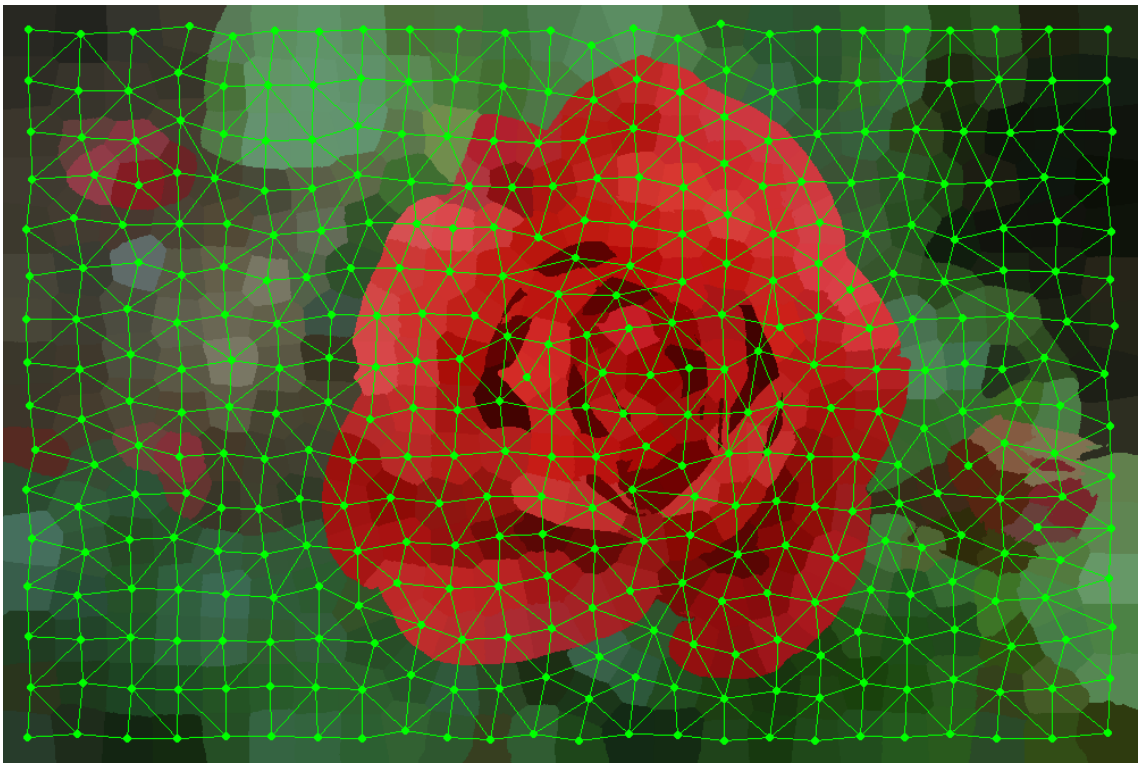
1. pixel není vyplněn - opakujeme operaci pro sousední pixely,
2. pixel je vyplněn - další operace s tímto pixelem neprovádíme.

Pro úsporu paměti využívám matici indexů, která obsahuje u každého pixelu informaci do jakého superpixelu náleží. Indexy jsou samozřejmě vždy celá kladná čísla, proto jsem si stanovil, že záporná čísla detekují vyplnění pixelu. Narazil jsem ale na problém s

indexem 0, protože jeho výsledek po vynásobení -1 je opět 0. Inspiroval jsem se dvojkovým doplňkem, a vytvořil vzorec (10) pro vypočtení záporného indexu. Zde k znamená index pixelu a k_{neg} jeho zápornou hodnotu. Výhoda tohoto výpočtu je symetričnost, tudíž jej můžu použít pro převedení z kladného na záporný index a zpět, aniž bych pro zpětný převod potřeboval jinou funkci. Pokud zjistíme, že index procházeného pixelu je jiný, než index superpixelu který procházíme, narazili jsme na souseda a jeho index si uložíme. Nesmíme však zapomenout, že sousední superpixel jsme již mohli procházet vyplňovacím algoritmem a index může být i záporný. V tom případě musíme získat zpět kladný index použitím výpočtu (10).

$$k_{neg} = (k * -1) - 1 \tag{10}$$

Nejjednodušší implementace algoritmu semínkového vyplňování je pomocí rekurzivní funkce. To sebou nese ale riziko přetečení zásobníku. Pokud jsem program spouštěl na rozměrově malých obrázcích, tento problém nenastal. V momentě, kdy jsem použil obrázek větší, program začal vyhazovat výjimky „Stack overflow“. Bohužel tento druh výjimky v C++ nelze odchytit a po hledání jak problém vyřešit jsem našel pouze rady, že program musí být napsán tak, aby tato výjimka nikdy nemusela být vyhozena. Rozhodl jsem se pro odstranění rekurzivní funkce a využil datové struktury „stack“, do které ukládám souřadnice následujících bodů ke zpracování.



Obrázek 10 Ukázka cest mezi superpixely nalezenými pomocí semínkového vyplňování

Pokud si pozorně prohlédneme obrázek 10, zjistíme, že cesta je někdy nakreslena i mezi superpixely, které navzájem sousední nejsou, což je důsledek dříve zmíněných „orphans“. Náš algoritmus zahájí vyplňování vždy ze středu superpixelu a postupuje k jeho okrajům. Pokud narazí na jinou hodnotu indexu než je index aktuálně procházeného superpixelu, uloží si jej jako nového souseda. Tímto způsobem se do cesty může postavit „orphan“, jehož hlavní část je vzdálená a s procházeným superpixelem vůbec nesousedí. Přesto si jej uložíme jako souseda, a poté vznikají nepřesnosti při hledání nejkratší cesty v grafu.

3.5 Určení vzdálenosti sousedních superpixelů

Pokud již máme zjištěny sousedy jednotlivých superpixelů, je potřeba spočítat cenu cest k těmto sousedním superpixelům. Dijkstrův algoritmus popsaný v kapitole 3.6 se v různých úpravách používá pro plánování tras v GPS navigacích. Cena cesty tak může být určena podle vzdálenosti pro hledání nejkratší cesty, nebo podle rychlosti na daných úsecích pro hledání nejrychlejší cesty.

V našem případě je cena cest určena geodetickou vzdáleností, neboli barevným rozdílem sousedících superpixelů, která se vypočte podle vzorce (11). Parametry i a j označují indexy sousedících superpixelů, mezi kterými chceme cenu cesty, značenou $P(i, j)$, vypočítat.

$$P(i,j) = \sqrt{(l_i - l_j)^2 + (a_i - a_j)^2 + (b_i - b_j)^2} \tag{11}$$

Nesmíme zapomenout, že hodnotu barevného rozdílu je potřeba počítat v Lab barevném spektru. Výslednou hodnotu si uložíme jako délku cesty, kterou poté budeme využívat pro hledání nejvýhodnější cesty mezi superpixely.

3.6 Implementace Dijkstrova algoritmu

Pro další počítání globálního geodetického šíření v kapitole 3.7, je potřeba nalézt mezi superpixely, které nejsou sousední, nejkratší cestu, abychom získali geodetickou vzdálenost. V kapitole 3.4 jsme získali sousedy všech superpixelů a v podkapitole 3.5 jsme spočítali mezi nimi vzdálenost. To dostačuje k vytvoření souvislého grafu, jehož vrcholy jsou středy superpixelů, a nyní můžeme přejít k hledání nejkratší cesty mezi libovolnými superpixely. Autoři využili Dijkstrův algoritmus, který patří mezi nejznámější.

Vytvořil jsem si třídu *SuperPixel*, která obsahuje proměnné pro výpočet hrubé charakteristické mapy. A také seznam cest, které z daného superpixelu vycházejí. Cesta je opět reprezentována třídou s názvem *SPPath*. Zde si potřebuje uchovat informaci o délce, neboli váze, cesty a počátečního i koncového superpipixelu. Všechny proměnné v obou třídách jsou instanční, počáteční a koncová proměnná je pointer na datový typ *SuperPixel*, seznam cest je aplikován pomocí datové struktury *vector*, který uchovává pointery na třídu *SPPath*.

Prvním krokem Dijkstrova algoritmu je nastavení vzdáleností do všech vrcholů v grafu na nekonečnou hodnotu. V našem případě je vzdálenost reálné číslo, a maximální

hodnotu získáme konstantou *DBL_MAX*, výjimkou je vrchol, který je výchozí, tomu nastavíme vzdálenost na hodnotu 0. Nyní začíná cyklus, který bude probíhat do doby, než projdeme všechny vrcholy grafu:

- Jako aktuální vrchol označíme ten, který ještě nebyl zpracován a má nejkratší vzdálenost od výchozího vrcholu.
- Aktuální vrchol označíme jako zpracovaný. To znamená, že již z něj počítáme vzdálenosti k sousedním vrcholům.
- Spočteme vzdálenosti do sousedních vrcholů, které ještě nebyly zpracovány.
- Vzdálenost k sousednímu vrcholu připočteme ke vzdálenosti aktuálně procházeného vrcholu a uložíme. Pokud sousední vrchol nemá uloženou kratší vzdálenost z předchozích výpočtů.

Z výše uvedeného postupu je patrné, že Dijkstrův algoritmus při svém dokončení získá z výchozího vrcholu cesty ke všem ostatním vrcholům. Toto chování může být mnohdy přítěží, pro následující výpočty je to ale obrovskou výhodou, protože při počítání geodetického šíření budeme od jednoho superpixelu potřebovat vzdálenosti cest ke všem ostatním. Z toho důvodu si uložíme všechny cesty, ne jen tu kterou jsme potřebovali.

V naší implementaci je také lehce upraven krok č. 4. Dříve než vypočtenou vzdálenost přičteme k celkové vzdálenosti aktuálně procházeného vrcholu, je potřeba provést výpočet (12). D_s označuje výslednou vzdálenost k připočtení, d značí délku cesty a $u(\cdot)$ je kroková funkce popsána v rovnici (13). Proměnná d_0 je mez pro vyvážení malých barevných rozdílů, tuto hodnotu lze měnit spouštěcími parametry viz sekce 4.

$$D_s(d) = u(d - d_0)d \tag{12}$$

Stejně jako proměnná d_0, ani kroková funkce $u(\cdot)$ nebyla blíže specifikována a musel jsem opět vymyslet vlastní řešení. Pro větší možnost přizpůsobení jsem se rozhodl místo ostré krokové funkce použít Logistickou funkci, která je popsána vzorcem (13). Jednotlivé proměnné jsem určil experimentálně, ale pomocí spouštěcích parametrů popsaných v kapitole 4 je lze měnit a zadávat můžeme reálná čísla v celém rozsahu. Na obrázku 11 je znázorněn graf pro výchozí hodnoty následně vysvětlených proměnných:

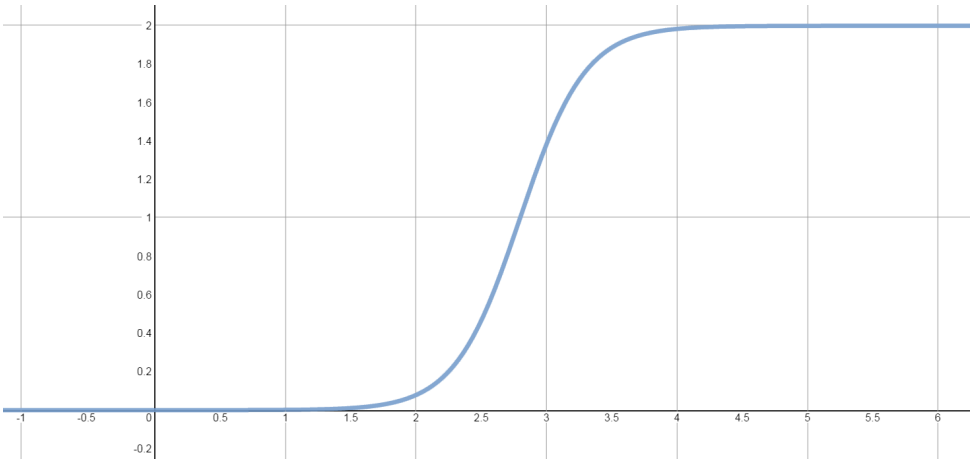
r_y Rozsah hodnot na ose y. Výchozí hodnota je 2,

o_x Posun středu Logistické funkce na ose x. Výchozí hodnota je 2.8,

s Strmost přechodu z hodnoty 0 na hodnotu r_y. Výchozí hodnota je 4,

d_0 Mez pro vyvážení malých barevných rozdílů, výchozí hodnota je 0.005.

$$u(x) = \frac{r_y}{1 + e^{(x-o_x)s}} \tag{13}$$



Obrázek 11 Graf Logistické funkce

3.7 Výpočet globálního geodetického šíření

Jak již bylo zmíněno, hlavní náplní článku [1] je představení nové metody globálního geodetického šíření, která přesněji určí náš významný objekt. Využijeme hrubou charakteristickou mapu a budeme šířit geodetickou energii jednotlivých superpixelů ke všem zbylým v závislosti na geodetické vzdálenosti. Toho dosáhneme výpočtem (14), který provedeme pro všechny superpixely.

$$S_k^{propagation} = \sum_{j}^{K_{real}} f_{j\to k} S_j^{coarse} \tag{14}$$

Výsledkem $S_k^{propagation}$ je šířená energie superpixelu s indexem k, který nabývá hodnot $[0, K_{real} - 1]$, vůči všem ostatním superpixelům s indexem j. Hodnota S_j^{coarse} je charakteristická energie superpixelu získaná v sekci 3.3 a $f_{j\to k}$ je intenzita šíření mezi superpixely s indexy j a k získaná výpočtem (15).

$$f_{j\to k} = \frac{1}{N} e^{-\beta d(R_k, R_j)} \tag{15}$$

$$N = \sum_{j}^{K_{real}} e^{-\beta d(R_k, R_j)} \tag{16}$$

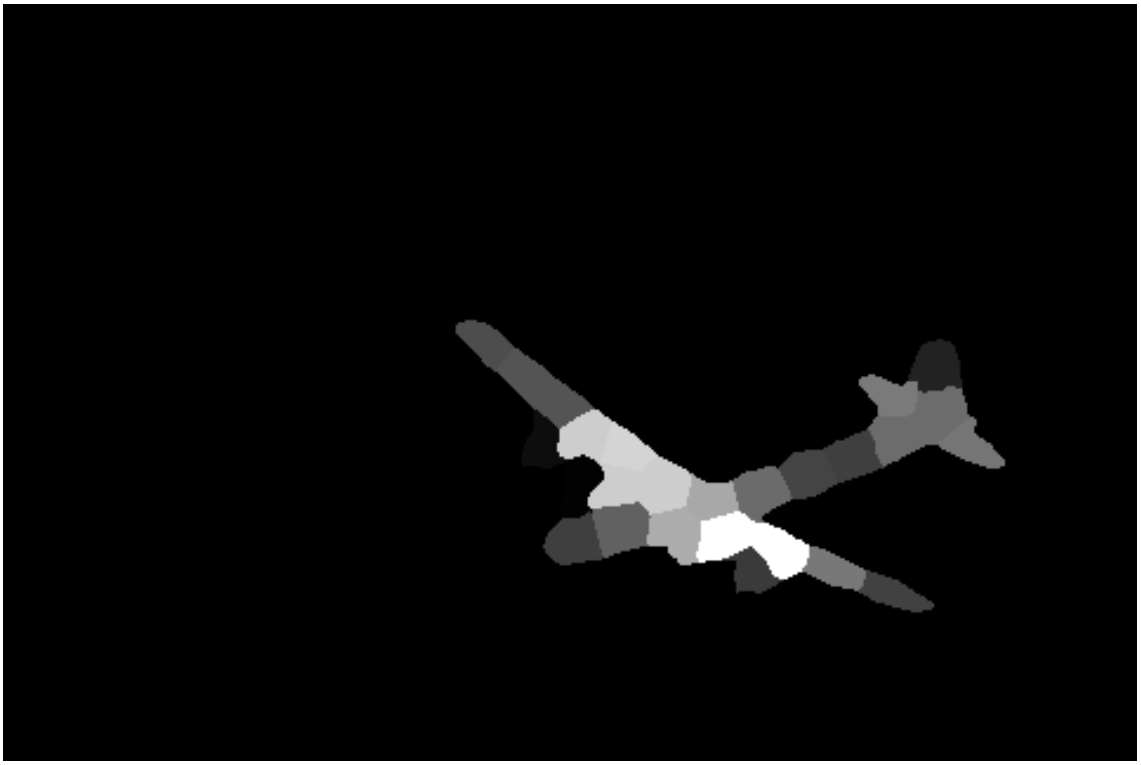
Konstanta β byla pro nenormalizované hodnoty barevného prostoru *Lab* stanovena na hodnotu 0.05. Pro normalizování charakteristické energie je N definováno výpočtem (16). Vzdálenost mezi superpixely s indexy k a j označena $d(R_k, R_j)$ musí být vypočtena Dijkstrovým algoritmem, protože dané superpixely nemusí být sousední.

3.8 Vykreslení ořezové masky

Závěrečným úkolem je vytvoření ořezové masky, ve které jsou černé části nedůležité, naopak bílé či šedé jsou součástí našeho významného objektu. Vykreslení masky je shodné s procesem vykreslování superpixelů, popsaným v kapitole 2.5. Nyní však budeme využívat pro barevné hodnoty údaje získané geodetickým šířením popsané v kapitole 3.7.

Hodnotu $S_k^{propagation}$ pro všechny pixely normalizujeme na rozsah $[0, 255]$ podle vzorce (17). To nám umožní normalizované údaje přímo ukládat jako barevnou hodnotu do výstupního souboru. Obraz pro výstup musí být definován pouze s jedním 8bitovým kanálem, abychom nemuseli nastavovat všechny *RGB* kanály, čímž vytvoříme obraz ve stupních šedi. Výslednou ořezovou masku si můžete prohlédnout na obrázku 12.

$$\begin{aligned} max_{propagation} &= \max_{0 \leq i < K_{real}} S_k^{propagation} \\ min_{propagation} &= \min_{0 \leq i < K_{real}} S_k^{propagation} \\ F_k^{global} &= \frac{S_k^{propagation} - min_{propagation}}{max_{propagation} - min_{propagation}} 255 \end{aligned} \tag{17}$$



Obrázek 12 Výsledná ořezová maska

4 Parametry programu

Jak již bylo zmíněno v dřívějších kapitolách, hodně funkcí a proměnných vůbec nebylo v pracích zdokumentováno ani specifikováno. Často jediné vysvětlení bylo frázemi podobnými, jako je tato: „Where $u(\cdot)$ is step function“. Proto jsem většinu proměnných umožnil zadávat jako parametry při spouštění programu. Některé z nich jsou povinné a je nutné, aby je uživatel zadal. Zvolit jejich správné hodnoty může být ze začátku dosti náročné, ale nelze je univerzálně určit pro všechny případy. Ostatní proměnné, které nebyly vysvětleny, jsem určil metodou „pokus omyl“. Pokud ale nebudou vyhovovat, můžeme je pomocí spouštěcích parametrů libovolně měnit.

Spouštěcí parametry napsány mezi hranatými závorkami jsou nepovinné, a nemusí se dodržet jejich pořadí, jak je uvedeno v předpise. Pořadí se musí dodržet pouze u povinných parametrů. Předpis:
zdrojovy cilovy K CT [-M pravidelnost] [-GB rozsahX [rozsahY]] [-SP cesta] [-IPL] [-CH cesta] [-SF prah rozsahY posunX strmost]

zdrojovy Název obrázku, který chceme zpracovat.

cilovy Název souboru, do kterého uložíme výslednou masku.

K Počet superpixelů, které se vygenerují. (Více v kapitole 2.2)

CT Práh Harrisova detektoru rohu, reálné číslo v rozsahu $0 - 255$. (Více v kapitole 3.2)

-M Úroveň dodržování kontur vs. pravidelnost superpixelů. Zadává se celé číslo v rozsahu $0 - 40$. Výchozí hodnota je 40. (Více v kapitole 2.2)

-GB Průměr Gaussova vyhlazení. RozsahX i RozsahY je přirozené liché číslo. RozsahY je nepovinný, a bude mít shodnou hodnotu jako RozsahX, pokud nebude zadán. Výchozí hodnota pro oba rozsahy je 5. (Více v kapitole 2.7)

-SP Název souboru, do kterého uložíme obrázek s vygenerovanými superpixely.

-IPL Při zadání se program neukončí při zaplnění více než 40% obrázku klíčovými body z Harrisova detektoru. (Více v kapitole 3.2)

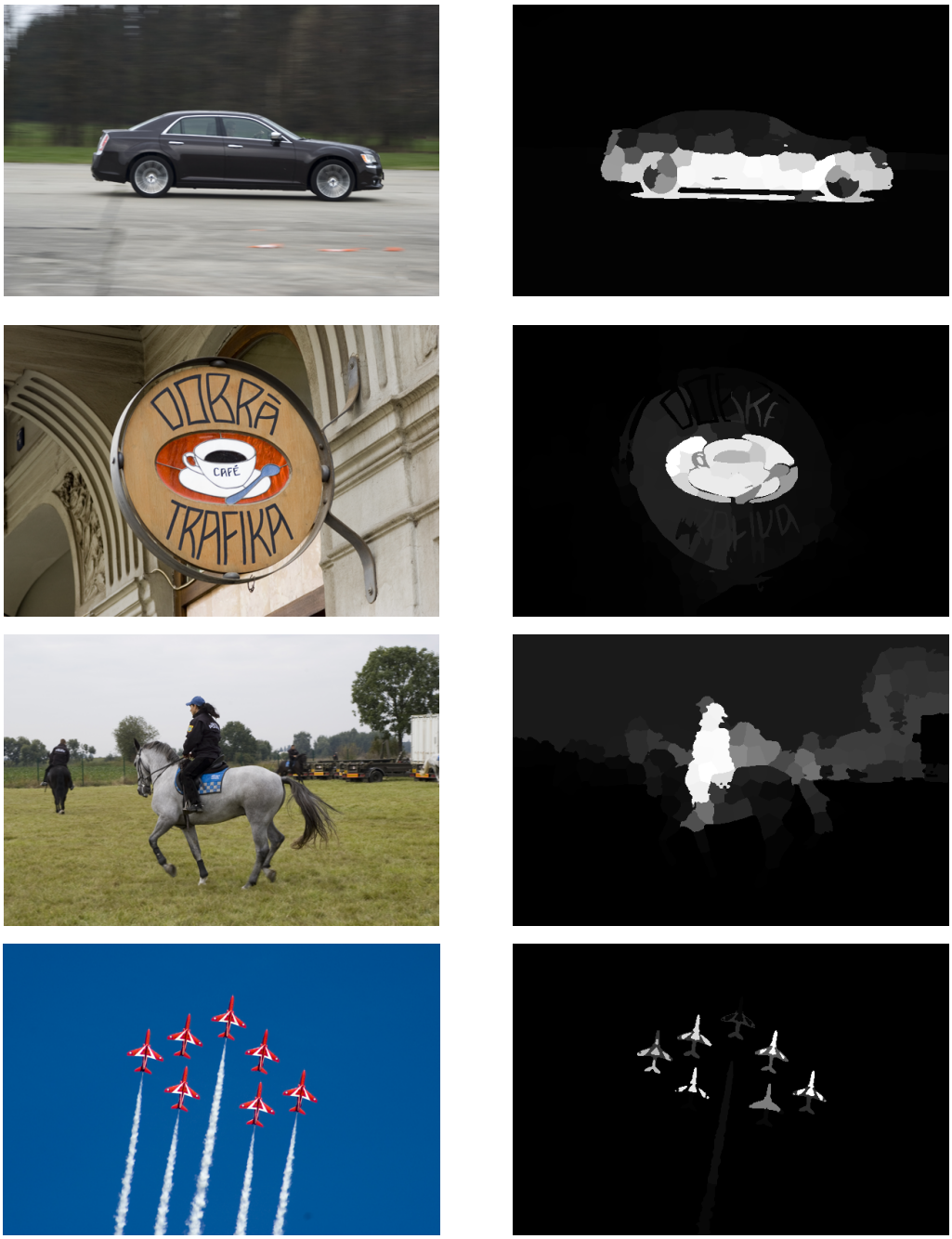
-CH Název souboru, do kterého uložíme obrázek s vykresleným konvexním obalem.

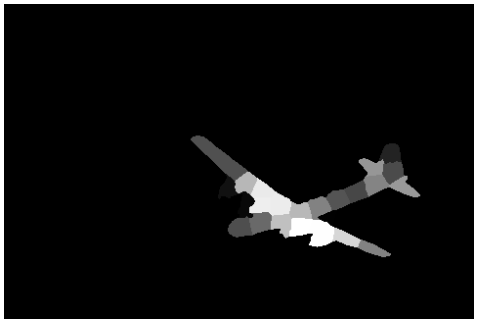
-SF Parametry krokové funkce viz rovnice (13). Všechny hodnoty jsou reálná čísla bez omezení. (Více v kapitole 3.6)

- **prah** Malá hodnota, která se odečítá od vypočtené délky mezi superpixely. Výchozí hodnota je 0.005.
- **rozsahY** Rozsah hodnot pro osu y, počátek je v 0. Výchozí hodnota je 2.
- **posunX** Posun středu logistické funkce po ose x. Výchozí hodnota je 2.8.
- **strmost** Určuje strmost přechodu logistické funkce mezi hodnotou 0 − rozsahY. Výchozí hodnota je 4.

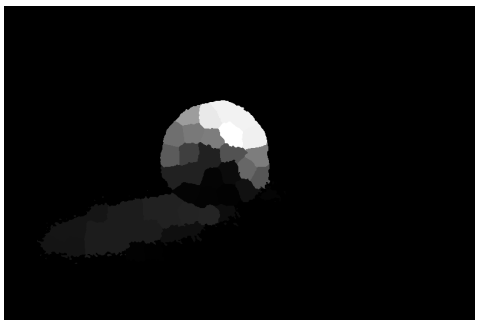
5 Výsledné vygenerované masky

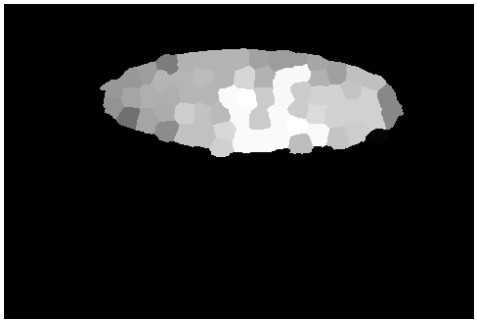
Obrázek 13: Zdrojový obrázek nalevo, a vygenerovaná maska programem napravo

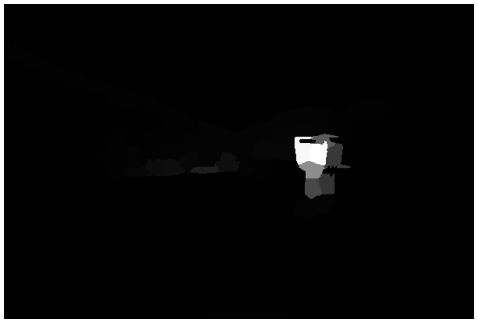


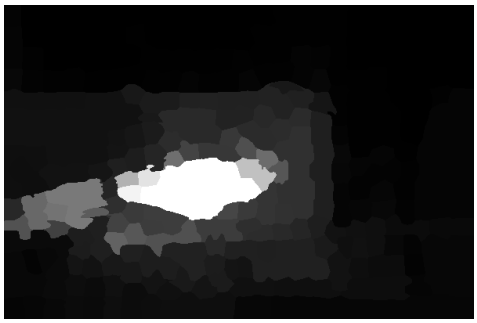







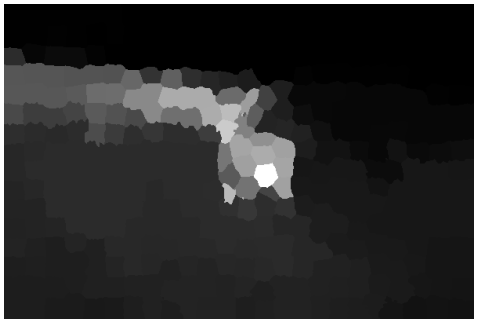




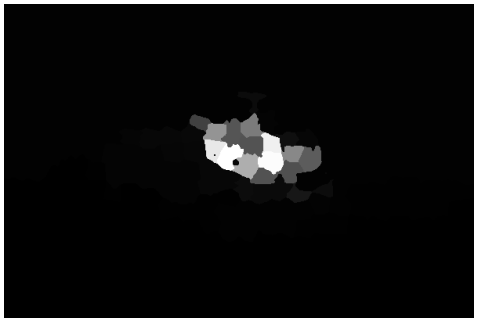












6 Zhodnocení práce

V práci jsme si popsali implementaci mnoha algoritmů, které nás postupně dovedly k výslednému detekování významných částí v obraze. Závěrečný výstup programu je ořezová maska, jejíž světlé části by měly být součástí významného objektu, zatímco tmavé části by neměly být důležité vůbec. Ne vždy je však detekce úspěšná, a jsou znázorněny pouze zájmové části obrazu, jak jsme se mohli přesvědčit.

Započali jsme tvorbou superpixelů, které by měly náš obraz rozdělit na atomické přibližně stejně velké části. Tato operace je klíčová pro další využití a musela být provedena kvalitně, abychom následným používáním nenásobili chyby vzniklé ihned na začátku. První problém nastal, když jsem našel více verzí článku, jak superpixely generovat, a v obou byl zmíněn jiný výpočet vzdálenosti pixelů. Použití různých vzorců ale nepřineslo tak velké zlepšení, jakého jsem dosáhl použitím 32bitové hloubky každého barevného kanálu. Zvětšení barevné hloubky je čistě implementační záležitost, kterou jsem se rozhodl využít. Nemá nic společného s odborným článkem, protože v něm o implementaci programu není vůbec zmínka, popisuje pouze postup. I při využití větší barevné hloubky jsem nebyl s výsledky spokojen. Mnohdy vznikaly superpixely s velmi nepravidelným tvarem, případně obsahovaly odtržené části zvané „orphans“, které nebyly součástí celého superpixelu. Na vyřešení těchto problémů v článku upozorňovali a navrhovali řešení *vynuceného spojení*, jehož algoritmus však nepopsali. Já se místo toho rozhodl využít Gaussovo rozostření, které dle mého názoru bude rychlejší a zlepšení je velmi zřetelné, přesto u složitějších obrázků „orphans“ části mohou vznikat.

Další algoritmy se týkají již samotné problematiky detekování významný částí, které na závěr vykreslí zmíněnou ořezovou masku. Bezproblémové bylo získání globálního kontrastu jednotlivých superpixelů vůči všem ostatním, což se ovšem nedá říci o získání konvexního obalu, který by nám měl přiblížit významné části. K vytvoření obalu je zapotřebí získat body, které jsme detekovali pomocí Harrisova detektoru rohů, a zde nastaly hned 2 problémy. První problém byl určení meze pro hledání bodů, což je však nutné zadat uživatelem a nemusí být jednoduché mez zvolit správně. Pokud zvolíme špatnou hodnotu, získáme body téměř v každém pixelu, nebo naopak dostaneme malý počet bodů. Snažil jsem se problém vyřešit použitím jiného detektoru rohů, konkrétně metody SIFT, v tomto případě jsem ale dostal vždy obrovské množství bodů. Jako další krok bylo odebrání klíčových bodů v určité vzdálenosti od okraje, ale nebylo specifikováno jak velká tato oblast má být, proto jsem si hodnotu určil dle svého uvážení. Poté jsme již mohli vytvořit konvexní obal nad body, které zůstaly.

Dále bylo potřeba zjistit, které superpixely leží v získaném konvexním obale a které mimo něj a zde vznikl další problém. Nebylo specifikováno jak určit jejich vzájemnou polohu, z toho důvodu jsem se rozhodl pro nejlehčí a nejrychlejší řešení, určení zda v konvexním obale leží střed daného superpixelu. Nyní následuje vypočtení globálního šíření geodetické energie superpixelů, kde se využívá pro získání vzdáleností Dijkstrův algoritmus. Článek popisující detekování významných částí také neřeší implementaci jednotlivých částí, takže způsob implementace byl plně v mých rukou. Abychom mohli implementovat algoritmus pro hledání nejkratší cesty v grafu, musíme tento graf prvně vytvořit. Pro získání cest mezi superpixely jsme museli zjistit, které jsou navzájem sou-

sední. Implementoval jsem jednoduchý algoritmus semínkového vyplňování, který bylo potřeba lehce upravit našim potřebám. Při hledání nejkratší cesty Dijkstrovým algoritmem jsem narazil na další problém. Na délky jednotlivých cest při počítání se aplikuje výpočet, ve kterém od délky odečítáme mez, a také obsahuje krokovou funkci. Bohužel ani mez, ani kroková funkce nebyla specifikována a určil jsem všechny proměnné pokusem. Z důvodu velkého počtu proměnných určených metodou „pokus omyl“ jsem se rozhodl využívat spouštěcí parametry, pomocí kterých můžeme všechny proměnné měnit. Opět to ale ztěžuje využití algoritmu, protože nastavení nemusí být jednoduché.

Program by určitě bylo možné zlepšit testováním jednotlivých funkcí a jejich optimalizací. Také by se mohl navrhnout algoritmus pro správné zjištění proměnných, které jsou nyní určeny pokusem, nebo je musí zadávat uživatel při spouštění programu. Přestože podle autorů, je SLIC nejrychlejší algoritmus pro získání superpixelů, dle mého názoru je pomalý. Asi jsem měl zkreslenou představu o rychlosti programu, ale jeho využití v reálném čase, například zpracování videa, mi přijde nemožné.

7 Reference

[1] Fu, K., Gong, C., Gu, I. Y., & Yang, J. (2013). *Geodesic saliency propagation for image salient region detection*. In *ICIP* (pp. 3278-3282).

[2] Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., & Susstrunk, S. (2012). *SLIC superpixels compared to state-of-the-art superpixel methods. Pattern Analysis and Machine Intelligence*, IEEE Transactions on, 34(11), pp. 2274-2282.

[3] Weatherall, I. L., & Coombs, B. D. (1992). *Skin color measurements in terms of CIELAB color space values*. Journal of investigative dermatology, 99(4), pp. 468-473.

[4] Süsstrunk, S., Buckley, R., & Swen, S. (1999, January). *Standard RGB color spaces*. In *Color and Imaging Conference* (Vol. 1999, No. 1, pp. 127-134). Society for Imaging Science and Technology.

[5] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). *Introduction to algorithms* (Vol. 2, pp. 531-549). Cambridge: MIT press.

[6] E. Davies *Machine Vision: Theory, Algorithms and Practicalities*, Academic Press, 1990, pp. 42-44.

[7] Pascal Mettes *https://github.com/PSMM/SLIC-Superpixels*

[8] Harris, C., & Stephens, M. (1988, August). *A combined corner and edge detector*. In *Alvey vision conference* (Vol. 15, pp. 50).

[9] Lowe, D. G. (2004). *Distinctive image features from scale-invariant keypoints*. *International journal of computer vision*, 60(2), pp. 91-110.