VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

SÉMANTICKÁ SEGMENTACE VOZOVKY S DETEKCÍ JÍZDNÍCH PRUHŮ V OBRAZE
IMAGE BASED ROAD SURFACE SEGMENTATION WITH LANE DETECTION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
Rudolf Turoň

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Tomáš Zemčík

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Automatizační a měřicí technika**

Ústav automatizace a měřicí techniky

Student: Rudolf Turoň ***ID:*** 211188

Ročník: 3 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Sémantická segmentace vozovky s detekcí jízdních pruhů v obraze

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je navrhnout systém pro sémantickou segmentaci vozovky v obraze (tj. segmentace vozovka/nevozovka, horizontální dopravní značení).

1. Proveďte rešerši používaných metod pro obecnou segmentaci a detekci jízdních pruhů.
2. Seznamte se s dostupnými nástroji pro zpracování obrazu (např.: OpenCV, Scikit-image).
3. Nasbírejte reprezentativní obrazový dataset.
4. Navrhněte systém zpracování dat.
5. Navrhovaný systém implementujte pro obraz i video.
6. Systém ověřte na obraze i živém kamerovém vstupu.
7. Funkčnost systému vyhodnoťte s ohledem na přesnost a rychlost.

DOPORUČENÁ LITERATURA:

[1] - HLAVÁČ, Václav a Milan ŠONKA, 1992. Počítačové vidění. Praha: Grada. ISBN

80-854-2467-3.

[2] - JÄHNE, Bernd a Horst HAUSSECKER, 2000. Computer vision and applications:

a guide for students and practitioners. San Diego: Academic Press. ISBN 01-

237-9777-2.

Termín zadání: 8.2.2021 ***Termín odevzdání:*** 24.5.2021

Vedoucí práce: Ing. Tomáš Zemčík

doc. Ing. Václav Jirsík, CSc.
předseda rady studijního programu

Abstrakt

Tato bakalářská práce se zabývá sémantickou segmentací vozovky s detekcí horizontálního dopravního značení. Cílem práce je provedení rešerše používaných metod pro obecnou segmentaci a detekci jízdních pruhů, nasbírání reprezentativního datasetu k testování, návrh systému zpracování dat včetně implementace pro sémantickou segmentaci a detekci jízdních pruhů v obraze, videu i živém kamerovém vstupu. Pro předzpracování obrazu je využito obrazu filtrovaného morfologickými transformacemi. Segmentace se realizuje metodou rozvodí se značkami, přičemž pro hledání značek je navržen adaptivní algoritmus. Jízdní pruhy jsou vyhledány v naprahovaném, projektivně transformovaném obraze pomocí posuvných oken. Ve výsledku se podařilo dosáhnout přesnosti segmentace 88,3 % dle metriky IoU. V závěru práce jsou diskutovány dosažené výsledky a shrnuty možnosti dalšího vylepšení systému.

Klíčová slova

Segmentace, detekce jízdních pruhů, matematická morfologie, rozvodí, posuvná okna, projektivní transformace, adaptivní prahování, barevné modely, histogram

Abstract

This bachelor thesis deals with road semantic segmentation with lane lines detection. The aim of the thesis is to conduct a survey of methods used for general segmentation and lane lines detection, collect a representative dataset for testing, design a data processing system including implementation for semantic segmentation and detection of lanes in image, video and live camera input. In preprocessing phase images are filtered using morphological transformations. The segmentation is performed using the watershed method with labels, and an adaptive algorithm is designed to find the labels. The lanes are searched for in the thresholded projectively transformed image using the sliding window technique. As a result, a segmentation accuracy of 88.3% is achieved based on the IoU metric. The thesis concludes with a discussion of the obtained results and summarizes the possibilities for further improvements of the system.

Keywords

Segmentation, lane lines detection, mathematical morphology, watershed, sliding windows, projective transformation, adaptive thresholding, color models, histogram

Bibliografická citace

TUROŇ, Rudolf. *Sémantická segmentace vozovky s detekcí jízdních pruhů v obraze*. Brno, 2021. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/134715. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Tomáš Zemčík.

Prohlášení autora o původnosti díla

Jméno a příjmení studenta:	Rudolf Turoň
VUT ID studenta:	211188
Typ práce:	Bakalářská práce
Akademický rok:	2020/21
Téma závěrečné práce:	Sémantická segmentace vozovky s detekcí jízdních pruhů v obraze

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: 20. května 2021 ----------

podpis autora

Poděkování

Rád bych poděkoval vedoucímu mé bakalářské práce Ing. Tomáši Zemčíkovi za odborné vedení, cenné rady a věnovaný čas při vypracovávání mé bakalářské práce. Děkuji také přítelkyni a mé rodině za podporu a oporu při studiu.

V Brně dne: 20. května 2021 ----------

podpis autora

Obsah

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

Úvod

Tato bakalářská práce se zabývá návrhem a implementací systému pro sémantickou segmentaci vozovky s detekcí horizontálního dopravního značení v obraze a videu. Počítačové vidění je velmi důležitým prvkem u moderních vozidel. Podle informací o nehodách v USA v letech 2009 až 2015 byla u vozidel vybavených systémem varování před opuštěním jízdního pruhu (angl. *lane departure warning - LDW*) nehodovost nižší o 18 %, při nehodách se zraněním pak o celých 24 % a při nehodách se smrtelným zraněním dokonce až o 86 % nižší [1]. Přesto bylo v roce 2019 pouze asi 13 % automobilů registrovaných v USA vybaveno systémem varování před opuštěním jízdního pruhu [2]. Podle švédské studie se odhaduje, že systémy pro udržování vozidla v jízdních pruzích redukují na švédských silnicích čelní srážky vozidel a nehody jednoho vozidla v rychlostech 70 až 120 km/h, a to až o 53 % [3]. Přínosy počítačového vidění v dopravě jsou tedy jednoznačné a je patrné, že řidiči může tato technologie pomoci zásahem do řízení například pří složité dopravních situaci, při zdravotním problému či únavě řidiče. Systémy najdou dále využití v autonomních vozidlech, která jsou díky němu schopná rozpoznat, kde v obraze je silnice a kde už není. Na základě toho zvládne vozidlo samostatně vyhodnotit, kudy může jet, a udržovat se tak v jízdních pruzích.

K hlavním cílům práce patří nejprve sběr reprezentativního datasetu vhodných obrazových dat ze silnic při jízdě automobilem, a následně pro tato data navrhnout systém jejich zpracování. Úkol systému spočívá v segmentaci vozovky v obraze a v detekci horizontální dopravní značení. Tím jsou myšleny čáry na vozovce, které definují jízdní pruhy. Dalším cílem je teoretický návrh implementovat a ověřit jeho funkčnost na obrazu i na živém kamerovém vstupu. Implementovaný systém by měl být co nejrobustnější, a pokud možno co nejuniverzálnější. Zvlášť důležité je, aby byl celý systém jen minimálně závislý na změně osvětlení scény, počasí, dopravní situaci a dalších aspektech.

K dosažení vytyčených cílů bude využita segmentace vozovky prováděná s pomocí zjednodušeného obrazu segmentovaného metodou rozvodí se značkami, přičemž značky pozadí budou získány adaptivně. Jízdní pruhy se pak detekují v projektivně transformovaném obraze za pomoci metod prahování obrazu a hledání pixelů odpovídajících jízdním pruhům.

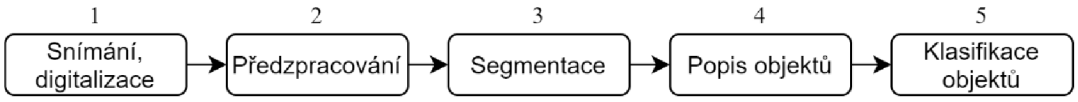
Práce je členěna do pěti základních části. 1. Kapitola se zabývá teoretickým rozborem počítačového vidění a zpracování obrazu, dále rešerší metod používaných pro obecnou segmentaci a detekci jízdních pruhů a také metrikami k vyhodnocení přesnosti sémantické segmentace. V 2. kapitole jsou představeny některé nástroje určené pro zpracování obrazu. 3. kapitola popisuje teoretický návrh systému pro sémantickou segmentaci a detekci jízdních pruhů v obraze. Tato část obsahuje také informace o reprezentativním datasetu. Praktickou implementací teoretického návrhu se následně zabývá kapitola 4. V 5. kapitole jsou implementovaná řešení srovnána a je vyhodnoceno testování navrženého řešení na obrázcích z datasetu a na živém kamerovém vstupu.

1. TEORIE A LITERÁRNÍ REŠERŠE

Tato kapitola se zabývá základy počítačového vidění, předzpracováním obrazu, barevnými modely, matematickou morfologii a geometrickými transformacemi obrazu se zaměřením na principy použité v této práci. Dále je zde provedena rešerše metod používaných pro obecnou segmentaci a detekci jízdních pruhů, včetně využití neuronových sítí. Poslední část této kapitoly se pak zabývá shrnutím nejpoužívanějších metrik pro vyhodnocení přesnosti sémantické segmentace obrazu.

1.1 Počítačové vidění

Cílem počítačového vidění je porozumění obrazu a nalezení vztahu mezi obrazem a reálnými objekty našeho světa. Obvykle je možné jej popsat několika kroky viz obrázek 1.1. Hranice těchto kroků však nejsou přesně vymezeny, a tak se v literatuře často liší. Práce se dále zabývá rozdělením počítačového vidění dle [4].



Obrázek 1.1 Obvyklý řetězec počítačového vidění [4]

Obvykle jsou objekty nebo scéna nejprve nasnímány obrazovým snímačem, poté převedeny na dvourozměrný signál, který je dále digitalizován a kvantován. Takto získáme digitální obraz, a ten je uložen v číselné podobě do počítače.

Následující fází je předzpracování obrazu. Zde se aplikují metody pro kompresi obrazu, filtraci šumu, detekci hran, změnu měřítka, perspektivní transformaci, ostření obrazu, či morfologické transformace. Tyto metody využívají buď přímo nasnímaného digitálního obrazu, nebo obrazu jen lehce upraveného. [4][5]

Segmentace je dalším krokem řetězce počítačového vidění. Její účel spočívá v oddělení objektů od pozadí, a také od sebe navzájem. Tento krok bývá z hlediska návrhu nejnáročnějším, avšak záleží na druhu scény určené pro segmentaci. Objekty v tomto případě rozumíme části obrazu, které nás pro další zpracování zajímají. [4]

Pomocí předchozího kroku byly získány jednotlivé objekty, a nyní je zapotřebí je popsat. Popis objektů může být buď kvantitativní (pomocí číselných charakteristik) nebo kvalitativní (pomocí relací). Příkladem jednoduchého popisu objektů mohou být jejich rozměry, počet, objem a podobně. [4]

Pátým, a zároveň posledním, krokem počítačového vidění je klasifikace objektů. Jejím cílem je popsané objekty určitým způsobem rozdělit do daných tříd, nebo porozumět obsahu obrazu, o kterém se předem nic nepředpokládá. Jednoduchá klasifikace do tříd může představovat například rozdělení objektů podle jejich tvaru, velikosti nebo textury. [4]

1.2 Reprezentace obrazu

Signál je jednorozměrná, dvourozměrná, trojrozměrná, nebo vícerozměrná funkce. Obrazový signál, kterým se práce dále zabývá, představuje funkci závislou na dvou souřadnicích, a jedná se tedy o dvourozměrný signál. K popisu monochromatického obrazu postačí skalární funkce, avšak k popisu barevných obrazu je již potřeba vektorová funkce. Obraz je obvykle definován jako funkce dvou proměnných *f(x, y)*, kde *x* a *y* jsou souřadnice v rovině. Pokud se do obrazové funkce zahrne i čas *t*, definujeme jej jako funkci tří proměnných *f(x, y, t),* a jedná se již o video. [5]

Hodnoty obrazové funkce v odstínech šedé odpovídají jasu v jednotlivých obrazových bodech. Pokud je trojrozměrný prostor mapován pomocí perspektivní projekce do roviny kamery, velké množství informací je ztraceno, což znamená, že tato transformace není vratná. Úlohu rekonstrukce dvojrozměrného obrazu na trojrozměrný lze řešit pouze s další dodatečnou znalostí scény a obrazového snímače. Jiným problémem zpracování obrazu je pochopení jeho jasu. Jediná informace, kterou obraz dává, je hodnota jasu pixelu. Ta však závisí na mnoha okolnostech, jako jsou vlastnosti povrchu snímaného objektu, jeho odrazivost, osvětlení či orientace scény. [4][5]

Digitální obraz je obvykle reprezentován jako matice, kde doménou obrazu je oblast *R* v rovině, která je dána souřadnicemi *x* a *y* a je definována vztahem

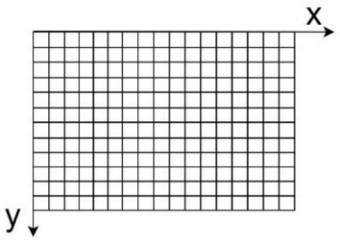
$$R = \{(x, y), 1 \leq x \leq x_m, 1 \leq y \leq y_m\}\,, \tag{1.1}$$

přičemž x_m je maximální souřadnice na ose x a y_m je maximální souřadnice na ose y. Počátek souřadnicového systému může být vlevo dole stejně jako v kartézských souřadnicích, ale častá je také orientace používaná v maticích, kde se počátek nachází vlevo nahoře. [5]

Digitalizace obrazu je proces, při němž je spojitá funkce obrazu *f(x, y)* vzorkována do obdélníkové mřížky (matice) s M řádky a N sloupci viz obrázek 1.2. Pixel je pak jeden bod v této matici obrazu. Kvantování představuje proces, který následně přiřadí vzorku neboli pixelu celočíselnou hodnotu jasu z určeného intervalu. Většinou se při kvantování přidělí bodu hodnota z *k* stejných intervalů a pro reprezentaci této hodnoty se používá *b* bitů. Hodnota jasu pixelu *k* se pak určí pomocí výpočtu

$$k = 2^b\,. \tag{1.2}$$

Pro reprezentaci barevného obrazu se běžně používá osm bitů na jeden kanál, přičemž kanály jsou použity tři - Red, Green a Blue. Každý kanál tedy může nabývat hodnot 0-255. [5]



Obrázek 1.2 Obdélníková obrazová mřížka s počátkem souřadnic vlevo nahoře

1.3 Vlastnosti digitálního obrazu

Digitální obraz má několik důležitých vlastností. Jednou z nich je vzdálenost mezi souřadnicemi *(i, j)* a *(h, k)* v obraze. Vzdálenosti můžeme rozdělit na tři typy. Prvním typem je Euklidovská vzdálenost D_E, která je popsána rovnicí

$$D_E\big((i,j),(h,k)\big) = \sqrt{(i-h)^2 + (j-k)^2}\ . \tag{1.3}$$

Nevýhodou tohoto typu vzdálenosti dvou bodů v obraze je složitý výpočet druhé odmocniny pomocí výpočetní techniky. Další možností, jak vyjádřit vzdálenost dvou bodů, je takzvaná vzdálenost městských bloků, která se označuje jako D_4 a je vyjádřena pomocí rovnice

$$D_4\big((i,j),(h,k)\big) = |i-h| + |j-k|\ . \tag{1.4}$$

V tomto případě je vzdálenost reprezentována nejmenším počtem vodorovných a svislých kroků potřebných pro přesun z výchozího bodu obrazu do bodu koncového. Pokud je možné v obrazové mřížce využít kromě vodorovných a svislých kroků i kroky diagonální, hovoříme o vzdálenosti D_8 definované rovnicí

$$D_8\big((i,j),(h,k)\big) = max\{|i-h|, |j-k|\}\ , \tag{1.5}$$

která vyjadřuje minimální počet kroků mezi dvěma body v obrazové mřížce jako na šachovnici. [5]

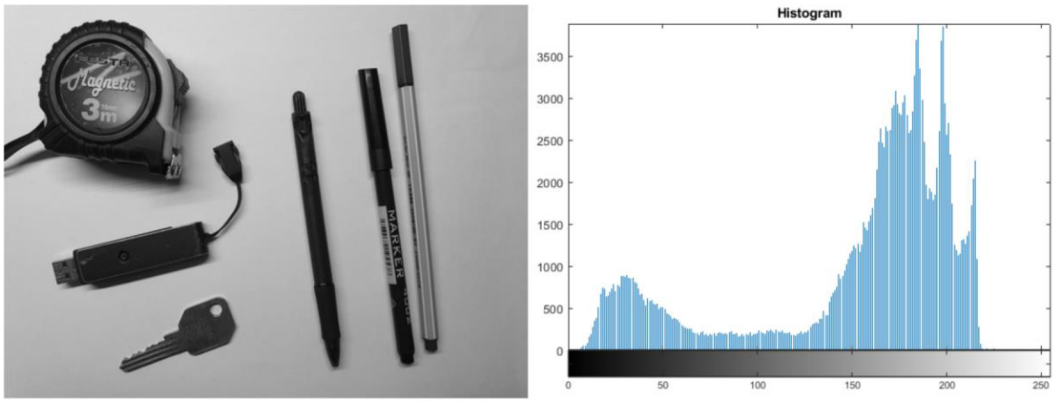
Další vlastností digitálního obrazu je okolí (sousedství) pixelů. To může být definováno například jako 4-sousedství, pokud do okolí bodu patří pixely ve vzdálenosti $D_4=1$, nebo jako 8-sousedství s pixely patřícími do okolí bodu ve vzdálenosti $D_8=1$. Vizualizace sousedství a vzdálenosti je možné vidět na obrázku 1.3. [5]

Obrázek 1.3 Zleva: 4-sousedství, 8-sousedství a vizualizace tří typů vzdálenosti [5]

Pro následné zpracování obrazu je také důležité zmínit pojem oblast. Oblast je množina pixelů, ve které existuje cesta (posloupnost pixelů) mezi jakýmikoli dvěma pixely. Podmínkou je, aby jak pixely, tak cesta, náležely této oblasti. Obvykle pak nazýváme některé oblasti obrazu jako objekty. [5]

1.3.1 Histogramy

Jasový histogram $h_f(z)$ zobrazuje frekvenci hodnoty jasu v obraze. Poskytuje tedy spojení mezi obrazem a pravděpodobností, že určitý pixel obrazu má jas hodnoty z. Histogram se ve většině případů vykresluje jako sloupcový graf a využívá se pro zhodnocení jasu, expozice a kontrastu scény. Dále lze histogram využít k určení hodnoty prahu pro segmentaci pozadí od objektu. Tomuto tématu se více věnuje kapitola 1.8.1. Ukázka histogramu obrazu je na obrázku 1.4 vpravo. [5]



Obrázek 1.4 Histogram (vpravo) vytvořený ze vstupního šedotónového obrazu (vlevo).

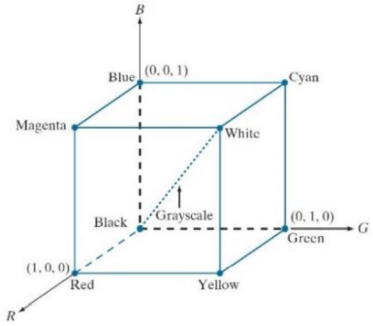
1.4 Barevné modely

Pro zpracování a reprezentaci obrazu existuje mnoho barevných modelů, které lze navzájem transformovat. Tyto modely specifikují standardním způsobem barvy a můžeme je rozdělit do tří skupin. Jedná se jednak o modely závislé na zařízeních a navržené pro co nejefektivnější práci s hardwarem. Druhou skupinou jsou modely, jež jsou orientovány na uživatele, aby pro něj bylo co nejsnazší model pochopit a používat. Posledním typem jsou modely nezávislé na zařízeních, tedy neovlivněné hardwarem. Patří zde modely CIE. CIE je standard definující jednoznačnou reprezentaci barvy nezávisle na vnějších faktorech. V této kapitole jsou dále popsány hlavní barevné modely, jež najdou uplatnění v algoritmech segmentace a v digitálním pořizování obrazu. [5][6][7]

1.4.1 RGB

Barevný prostor RGB má historii v barevném televizním vysílání a zároveň v napodobení lidského zraku, kde oko používá pro barevné vidění základní barvy tohoto modelu. Jedná

se o relativní prostor, který patří do skupiny modelů závislých na zařízeních a používá tři základní barvy nebo také kanály R (červená), G (zelená) a B (modrá). Pro vytvoření žádaného odstínu se využívá aditivní míchaní barev. Konkrétní barva je vyjádřena jako vektor ve trojrozměrném prostoru, který se skládá z intenzit tří základních barev (obrázek 1.5). Pokud jsou hodnoty barev kvantovány dle rovnice (1.2), kde $k = m-1$, pak vektor (0, 0, 0) odpovídá černé barvě, vektor (k, k, k) barvě bílé, vektor (0, k, 0) barvě zelené a tak dále. Obvykle se používá osmibitový barevný prostor na jeden kanál, což znamená, že máme k dispozici $256^3 = 2^{24} = 16\,777\,216$ možných barevných odstínů. V tomto modelu jsou většinou zaznamenávány fotografie a video. [5]

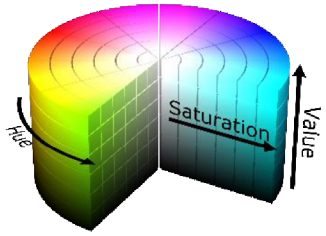


Obrázek 1.5 Barevný model RGB [6]

1.4.2 HSV (HSB)

U barevného modelu HSV (HSB) viz obrázek 1.6, jež patří do skupiny modelů orientovaných na uživatele, je barva vyjádřena pomocí tří složek: H (odstín), S (sytost) a V (hodnota nebo také jas). Model se snaží napodobit míchání barev malíři k dosažení požadovaných barevných odstínů. HSV model tedy odděluje informaci o sytosti barvy a jasu od samotné barvy. Používá se zde válcovitá geometrie, kde se odstín mění od 0° do 360°. Saturace se udává v procentech od 0 (šedá) do 1 (plná sytost barvy). Ve válcovitém prostoru tohoto modelu je sytost barvy reprezentována od středu k okrajům válce. Jas je množství bílé barvy, neboli relativní světlost barvy. To znamená, že jas o hodnotě 0 reprezentuje černou barvu a jas o hodnotě 1 reprezentuje barvu bílou. [5][8]

Na velmi podobném principu válcovité geometrie pracuje i model HSI (odstín, sytost, intenzita), který je známý také jako model HSL (odstín, sytost, světlost). [7]



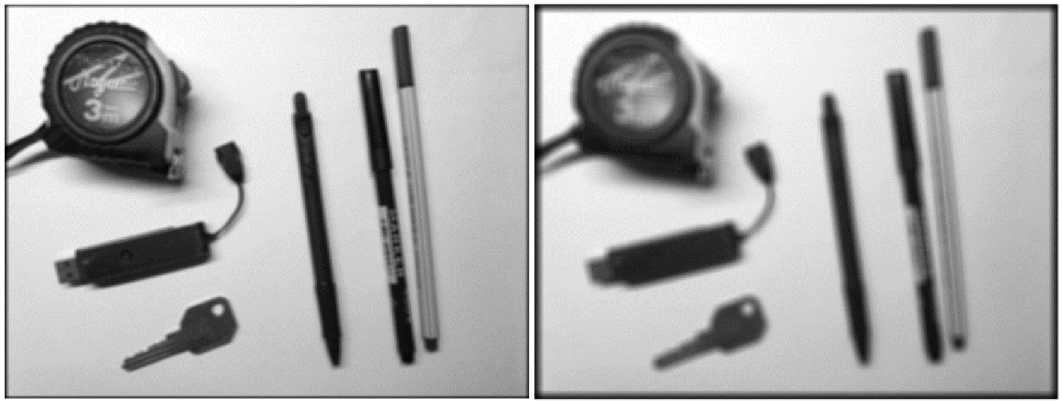
Obrázek 1.6 Barevný model HSV [8]

1.5 Předzpracování obrazu

Fáze předzpracování obrazu zahrnuje vylepšení obrazu, potlačení zkreslení a šumu pro další zpracování. Předzpracování nezvětšuje množství obrazových informací, obvykle ho naopak snižuje. V kapitole 1.5.1 je popsáno vyhlazování obrazu a kapitola 1.5.2 je zaměřena na detekci hran v obraze. Do předzpracování obrazu je možné zahrnout také matematickou morfologii popsanou v kapitole 1.7, která je v této fázi užitečná například pro následnou segmentaci.

1.5.1 Vyhlazování obrazu

Cílem vyhlazování obrazu je potlačení šumu a vad v obraze. Konkrétně se jedná o filtraci vysokých frekvencí ve Fourierově transformaci. Obvykle je to postup založený na určité formě průměrování hodnot jasu v okolí, přičemž okolí je podobné bodu, který je zpracováván. Při vyhlazování však dochází k rozmazávání hran, které je pro další zpracování obrazu, obzvlášť segmentaci, nežádoucí. Vybíráme proto primárně takové metody, které hrany v obraze pokud možno zachovávají.



Obrázek 1.7 Rozmazání průměrovacím filtrem velikosti 5x5 (vlevo) a velikosti 10x10 (vpravo).

Pro lineární filtry provádíme lineární transformaci pixelu *f(x, y)* jako lineární kombinaci jasu v okolí *O* pixelu *g(i, j)*. Rovnice

$$f(i,j) = \sum\sum_{(m,n)\in O} h(i-m, j-n)g(m,n) \tag{1.6}$$

je ekvivalentní diskrétní konvolucí obrazu *g* za použití konvoluční masky *h*. V tomto případě se průměrování provádí v okolí za použití čtvercové konvoluční masky, která má ve většině případů liché rozměry. Důvodem je fakt, že takto má maska středový bod a je čtvercová. Ukázka vyhlazování průměrováním různými velikostmi filtru *h* je znázorněna na obrázku 1.7. Konvoluční maska h pro průměrování v okolí 3x3 je definována jako

$$h = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}. \tag{1.7}$$

Význam pixelu ve středu a jeho okolí se někdy také zvětšuje, aby měl filtr větší účinnost. Popsaného principu využívá například aproximace Gaussova vyhlazování pro matici 3×3, kde se hodnoty s rostoucí vzdáleností od středu pixelu zmenšují o 1 viz rovnice

$$h = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}. \tag{1.8}$$

Daný filtr a jeho modifikace se používají pro vyhlazení obrazu v reálných aplikacích poměrně často. Výsledky metody vyhlazování jsou však přijatelné pouze v případě, že je šum menší než velikost objektů na obrázku. Ty jsou důležité pro další zpracování obrazu. [5]

Obrázek 1.8 Ukázka filtrace barevného obrazu mediánovým filtrem velikosti 7x7 (vlevo) a velikosti 19x19 (vpravo)

Mezi nelineární metody vyhlazování patří mediánový filtr. Ve statistice je medián definován jako prostřední hodnota v hodnotách seřazených dle velikosti. Pokud je počet hodnot, ze kterých medián počítáme, sudý, budou střední hodnoty dvě. Abychom tomuto při filtraci obrazu předešli, a byla tak zajištěna jedinečnost středního členu, volíme většinou pro výpočet mediánu čtvercové okolí obrazu sudé velikosti, například 3x3. Tento filtr příliš nerozmazává hrany a velmi dobře eliminuje šum vysoké frekvence. Nevýhodou však je, že čtvercový filtr poškozuje tenké čáry a ostré rohy. Ukázka filtrace barevného obrazu mediánovým filtrem je zachycena na obrázku 1.8 [5]

1.5.2 Detektory hran

Detekce hran v obraze je podstatným prvkem zpracování obrazu a segmentace. Hrana je vlastnost bodu a jeho okolí, ve kterém se rychle mění jas okolních pixelů. Hrany v obraze nejsou tolik závislé na osvětlení scény, a pokud se hledané objekty liší od okolí svým jasem, můžeme podle nich provádět segmentaci. Po aplikaci detekce hran obrazy dále zmenšují svou velikost, a tím také zrychlují další zpracování obrazu. V matematice velikost změny funkce popisuje derivace, v obraze je funkce závislá na dvou proměnných, a to souřadnicích *(i, j)* obrazu. Matematickou operací reprezentující hrany

je tady parciální derivace. Velikost změny obrazové funkce může být popsána také jako gradient, který udává směr největšího růstu obrazové funkce. Hrana se počítá z obrazové funkce v okolí pixelu a výsledkem je vektor se dvěma komponenty – velikostí a směrem. Velikost tohoto vektoru je rovna velikosti gradientu a jeho směr odpovídá směru gradientu ψ - *90°*. Výpočet velikosti gradientu a jeho orientace se provádí pomocí rovnic

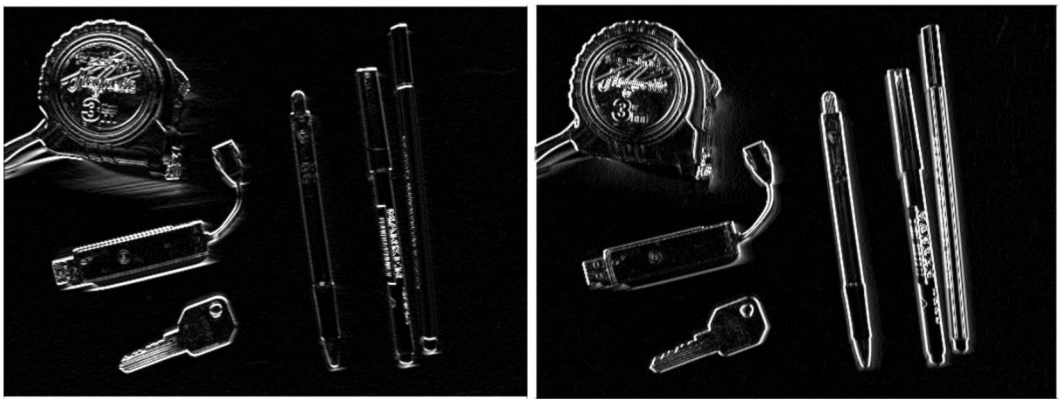
$$|grad\ g(x,y)| = \sqrt{\left(\frac{\partial g}{\partial x}\right)^2 + \left(\frac{\partial g}{\partial y}\right)^2}\,, \tag{1.9}$$

$$\psi = arg\left(\frac{\partial g}{\partial x}, \frac{\partial g}{\partial y}\right), \tag{1.10}$$

kde *g* je vstupní obraz a *(x, y)* jsou souřadnice pixelu [5]. Pokud nás zajímá pouze velikost gradientu bez ohledu na jeho orientaci, lze jej vypočíst také pomocí takzvaného Laplaceova operátoru [5]

$$\nabla^2 g(x,y) = \frac{\partial^2 g(x,y)}{\partial x^2} + \frac{\partial^2 g(x,y)}{\partial y^2}\,. \tag{1.11}$$

Jednoduché operátory aproximují derivace obrazové funkce pomocí rozdílu. Jsou počítány pomocí konvoluce (1.6), což znamená, že masky jsou aplikovány postupně na malé oblasti obrazu.



Obrázek 1.9 Gradientní obraz po aplikaci Sobelova hranového detektoru pro vertikální směr (vlevo) a pro horizontální směr (vpravo)

Sobelův operátor je aproximací první derivace. Jeho konvoluční masky jsou definovány jako h_1 pro vertikální směr hran, h_2 pro směr šikmý a h_3 pro horizontální směr hran, kdy pro masky platí rovnice

$$h_1 = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}, \quad h_2 = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix}, \quad h_3 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}. \tag{1.12}$$

Jako příklad hranového detektoru založeného na druhé derivaci je možné využít například Laplaceova operátoru ∇^2, který je aproximován pro digitální obraz 3x3 konvolučními maskami h_4 a h_5

$$h_4 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad h_5 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}. \tag{1.13}$$

Tento operátor detekuje hrany ve všech směrech. Nevýhodou však je, že na některé hrany v obraze odpoví dvakrát. [4][5]

1.6 Geometrické transformace

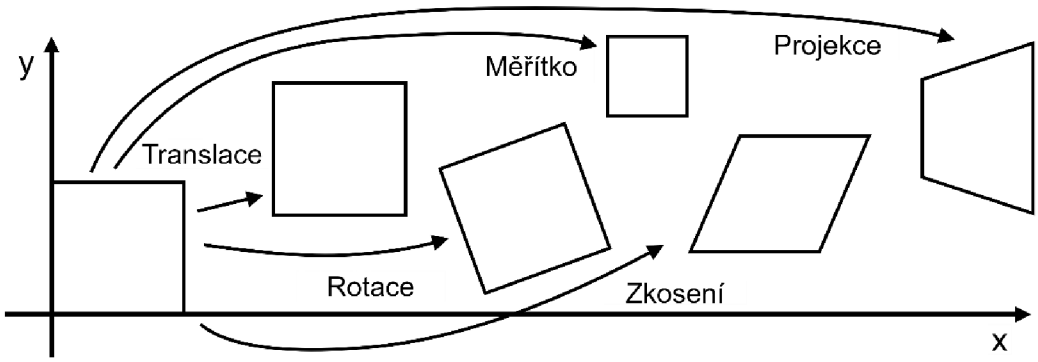
Geometrické transformace obrazu se používají nejčastěji pro eliminaci geometrického zkreslení, ke kterému dochází při pořizování snímku. Jedná se o vektorovou funkci *T*, která mapuje pixel ze souřadnic *(x, y)* na nové souřadnice *(x', y')*. Proces transformace se skládá ze dvou kroků. Nejprve se provede přeměna souřadnic pixelů vstupního obrazu na souřadnice pixelů výstupního obrazu. Výstupní souřadnice však většinou neodpovídají obrazové mřížce. Proto je v dalším kroku ještě potřeba najít bod v obrazové mřížce, který nejlépe odpovídá výstupní souřadnici. Hodnota jasu pixelu je pak často určena interpolací jasu několika okolních pixelů. [5]

Pro transformaci je také důležité zavedení homogenních souřadnic. Reprezentace bodů prostřednictvím homogenních souřadnic se používá primárně z toho důvodu, že je díky nim umožněno vyjádření nejčastěji používaných geometrických transformací s využitím jediné matice. Výhodnou je také to, že inverzní transformace se provede pomocí inverzní matice dané transformace. Skládaní transformací se pak realizuje jednoduše jako maticové násobení, které je snadno algoritmicky implementovatelné. Homogenní dvourozměrné souřadnice $\tilde{\boldsymbol{x}}$ jsou uspořádanou trojicí čísel $[x, y, w]$ s kartézskými souřadnicemi $\boldsymbol{x} = [X, Y]$, pro které platí

$$X = \frac{x}{w}, \; Y = \frac{y}{w}, \; w \neq 0\,, \tag{1.14}$$

přičemž souřadnici *w* nazýváme váhou bodu. [9][10]

1.6.1 Základní geometrické transformace



Obrázek 1.10 Ukázka geometrických transformací dvourozměrného obrazu [9]

Jednotlivé typy základních geometrických transformací dvourozměrného obrazu jsou znázorněny na obrázku 1.10. Patří mezi ně například posunutí, rotace, změna měřítka a zkosení.

Posunutí nebo také translace (angl. *translation*) je maticově definována jako

$$\tilde{\boldsymbol{x}}' = \boldsymbol{T}(t_x, t_y)\tilde{\boldsymbol{x}} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \tilde{\boldsymbol{x}}\,, \tag{1.15}$$

kde t_x je posun ve směru osy x, t_y je posun ve směru osy y, $\tilde{\boldsymbol{x}}$ je homogenní souřadnice bodu, který chceme transformovat na výstupní homogenní souřadnici $\tilde{\boldsymbol{x}}'$, a $\boldsymbol{T}$ je matice translace. [9][10]

Rotace (*angl. rotation*), tedy další často používaná geometrická transformace, je definována jako

$$\tilde{\boldsymbol{x}}' = \boldsymbol{R}(\alpha)\tilde{\boldsymbol{x}} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tilde{\boldsymbol{x}}\,, \tag{1.16}$$

kde α charakterizuje úhel, o který bude výstupní homogenní souřadnice $\tilde{\boldsymbol{x}}'$ natočená vůči vstupní homogenní souřadnici $\tilde{\boldsymbol{x}}$, a $\boldsymbol{R}$ je matice rotace. [9][10]

Transformace změny měřítka (angl. *scale*) je popsána vztahem

$$\tilde{\boldsymbol{x}}' = \boldsymbol{S}(s_x, s_y)\tilde{\boldsymbol{x}} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \tilde{\boldsymbol{x}}\,, \tag{1.17}$$

kde s_x označuje koeficient změny měřítka ve směru osy x, s_y je koeficient změny měřítka ve směru osy y a $\boldsymbol{S}$ je matice měřítka. Změna měřítka ovlivňuje současně polohu i velikost transformovaného obrazu. V měřítku s intervalem (0, 1) dochází ke zmenšení obrazu a přiblížení bodů blíže k počátku souřadnicového systému. V měřítku, které má interval $(1, \infty)$, dochází naopak ke zvětšení obrazu a oddálení bodů od počátku

souřadnicového systému. Transformace formou translace, rotace i změny měřítka zachovává v obraze přímé linie a úhly mezi nimi. [9][10]

Mezi základní geometrické transformace patří také zkosení (angl. *shear*). To je definováno rovnicemi (1.18) a (1.19)

$$\tilde{\boldsymbol{x}}' = \boldsymbol{SH}_x(sh_x)\tilde{\boldsymbol{x}} = \begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tilde{\boldsymbol{x}}\,, \tag{1.18}$$

$$\tilde{\boldsymbol{x}}' = \boldsymbol{SH}_y(sh_y)\tilde{\boldsymbol{x}} = \begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tilde{\boldsymbol{x}}\,, \tag{1.19}$$

kde sh_x je koeficient zkosení ve směru osy x, sh_y je koeficient zkosení ve směru osy y a $\boldsymbol{SH}$ je matice zkosení. Zkosení se vyznačuje důležitou základní vlastností, kterou je, zachovávání rovnoběžných linií v transformovaném obraze. [9][10]

1.6.2 Dvourozměrná projektivní transformace

Dvojrozměrná projektivní transformace je užitečná hlavně tehdy, pokud je potřeba obraz transformovat na pohled kamery z jiného úhlu, než ve kterém je skutečně umístěna. Ukázka je znázorněna na obrázku 1.10. Transformace pracuje v homogenních souřadnicích a její základní vlastností je zachování přímých linií v obraze. Lze ji popsat rovnicí

$$\tilde{\boldsymbol{x}}' = \boldsymbol{H}\tilde{\boldsymbol{x}}\,, \tag{1.20}$$

kde $\boldsymbol{H}$ reprezentuje matici homogenní transformace velikosti 3×3 a $\tilde{\boldsymbol{x}}$ je homogenní souřadnice bodu, který je transformován na výstupní homogenní souřadnice $\tilde{\boldsymbol{x}}'$. Homogenní matice $\boldsymbol{H}$ má tu vlastnost, že dvě homogenní matice, které se od sebe liší pouze měřítkem, jsou totožné. Matici transformace $\boldsymbol{H}$ je možné stanovit tak, že jsou určeny čtyři souřadnice bodů ve vstupním obraze a čtyři souřadnice bodů ve výstupním obraze, na které mají být vstupní souřadnice transformovány. Tyto souřadnice se dosadí do rovnice (1.20) a řešením vzniklé soustavy čtyř rovnic je pak požadovaná transformační matice $\boldsymbol{H}$.

Po projektivní transformaci bodu $\tilde{\boldsymbol{x}}$ pomocí rovnice (1.20) a matice $\boldsymbol{H}$ je nutné výsledné homogenní souřadnice $\tilde{\boldsymbol{x}}'$ ještě normalizovat, aby byl získán nehomogenní výsledek $\boldsymbol{x}' = [X', Y']$. Normalizace je vyjádřena pomocí rovnice

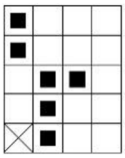
$$X' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}},\ \mathrm{X}' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}}\,, \tag{1.21}$$

kde h_{ij} označují jednotlivé prvky matice homogenní transformace $\boldsymbol{H}$ [9].

1.7 Matematická morfologie

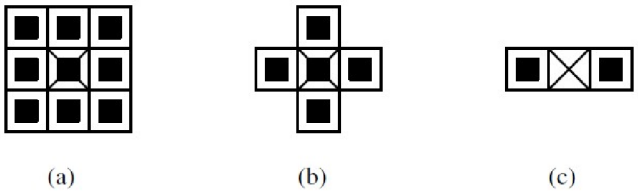
Matematickou morfologii můžeme označit za poměrně samostatnou část zpracování obrazu. Je založena na algebře nelineárních operátorů, které berou v úvahu tvar objektů, jenž se snaží zachovat. Využití matematické morfologie spočívá především ve filtraci šumu, zjednodušení tvaru objektů, zdůraznění struktury objektů a v segmentaci objektů z pozadí. [5]

Předpokládáme, že obrazy lze modelovat pomocí množiny bodů *n* dimenzí. V tomto případě se jedná o Euklidův dvojrozměrný prostor E_2 pro binární obraz nebo trojrozměrný prostor E_3 pro obrazy ve stupních šedi. Dále budeme uvažovat prostor E_2 pro binární obraz. Obraz je zde reprezentován bodovou množinou *X* obsahující body binárního obrazu, jež jsou rovny jedné. Každý bod v této množině je pak charakterizován dvěma čísly, a to souřadnicemi *(x, y)*. Příkladem popsané množiny může být *X= {(1,0), (1,1), (1,2), (2,2), (0,3), (0,4)}* vizualizována na obrázku 1.11.



Obrázek 1.11 Bodová množina X= {(1,0), (1,1), (1,2), (2,2), (0,3), (0,4)} s počátkem v bodě (0,0), který je na obrázku označen křížem [5]

Morfologická transformace se realizuje prostřednictvím relace bodové množiny obrazu *X* a menší bodové množiny *B* nazývanou strukturní element, který je vyjádřen jako okolí *O* počátku souřadnic tohoto elementu. Některé, často používané, strukturní elementy jsou vidět na obrázku 1.12. Symbol kříže v tomto obrázku označuje počátek elementu a černé čtverce charakterizují body, které jsou jeho součástí. Při aplikaci morfologické transformace na obraz *X* pohybujeme systematicky po celém obraze strukturálním elementem *B*. Výsledek relace mezi obrazem *X* a strukturním elementem *B* je pak zapsán do bodu obrazu, který odpovídá počátku souřadnic elementu *B*. [4][5]



Obrázek 1.12 Ukázka některých strukturních elementů [5]

1.7.1 Dilatace a eroze

Dilatace je morfologická transformace vyjádřena jako vektorový součet $\oplus$. Dilatace $X \oplus B$ dle rovnice

$$X \oplus B = \{p \in \varepsilon^2 : p = x + b, x \in X \; a \; b \in B\} \tag{1.22}$$

je pak množina všech možných vektorových součtů prvků obrazu X a strukturního elementu B. Příklad této transformace pomocí čtvercového elementu velikosti 3x3 je uveden na obrázku 1.13. U dilatace za použití strukturního elementu o velikosti 3x3 lze výsledek popsat tak, že se všechny body pozadí, které přímo sousedí s objektem, stanou body objektu. To znamená, že i díry v objektu o velikosti jeden bod se zaplní. Nejčastěji se tato operace používá pro zaplňování malých děr a úzkých mezer v objektu. Nevýhodou popsané metody však je, že objekty zvětšuje, proto se často spojuje s erozí pro zachování původní velikosti. [4][5]

Transformace erozí (2.18) je vektorovým rozdílem $\ominus$ dvou množin X a B viz rovnice

$$X \ominus B = \{p \in \varepsilon^2 : p + b \in X \; pro \; každé \; b \in B\} \tag{1.23}$$

a jedná se o duální operaci k dilataci, nikoli o operaci inverzní k dilataci. Ukázka její aplikace na obraz pomocí čtvercového strukturního elementu velikosti 5x5 je na obrázku 1.13. Operace eroze se často používá pro zjednodušení struktury objektu v obraze a rozdělení spojených objektů. Pomocí ní je navíc možné nalézt obrysy objektů, a to díky odečtení obrazu erodovaného od obrazu originálního. Erozi můžeme interpretovat rovněž jako

$$X \ominus B = \{p \in \varepsilon^2 : B_p \subseteq X\} . \tag{1.24}$$

Lze ji tedy také popsat tak, že se strukturní element posunuje přes obraz X vektorem p. Pokud je element B, který je posunut vektorem p, obsažen v X, pak bod počátku souřadnic strukturního elementu B v obraze X odpovídá erozi $X \ominus B$. [5][6]

Obrázek 1.13 Ukázka dilatace a eroze - vlevo je originální obrázek, uprostřed dilatace a napravo eroze

Dilatace a eroze obrazů v odstínech šedi se dá snadno vyjádřit jako rozšíření binárních operací popsaných výše, a to pomocí operace minimum a maximum. V případě eroze se jedná o přiřazení minimální hodnoty každému bodu z okolí pixelu vstupního obrazu X.

Pokud se jedná o dilataci, používá se operace maximum. Strukturní prvek v tomto případě nedefinuje jen okolí, jak tomu bylo u binárního obrazu, ale specifikuje také požadované lokální vlastnosti úrovní jasu obrazu. Ty jsou pak přičteny nebo odečteny při výpočtu maxima či minima z okolí.

Výše popsaný postup umožnuje vytvořit takzvané topografické zobrazení obrazu v odstínech šedi. Odstín následně reflektuje nadmořskou výšku. Světlé odstíny obrazu reprezentují výše položená místa v krajině a tmavé tóny charakterizují údolí a prohlubně. Tohoto topologického obrazu využívá segmentační metoda pomocí rozvodí. [5]

S využitím morfologických operací eroze a dilatace lze také výpočetně jednoduše aproximovat gradient pomocí takzvaného Beucherùva gradientu

$$grad(X) = (X \oplus B) - (X \ominus B) , \tag{1.25}$$

který se dá spočítat jako algebraicky rozdíl dilatace vstupního obrazu X strukturním elementem B jednotkové velikosti a eroze obrazu X stejným strukturním elementem. [5]

1.7.2 Otevření a uzavření

Otevření a uzavření jsou morfologické operace, jež vznikají spojením dvou základních transformací - dilatace a eroze. Používají se především k odstranění detailů v obraze, které jsou menší než strukturní element B, přičemž celkový tvar objektu zůstane zachován. Nejprve je popsána eroze následovaná dilatací zvanou otevření. Ta odděluje objekty, jež jsou spojeny úzkým spojem. Otevření obrazu X pomocí strukturního elementu B je definováno pomocí rovnice

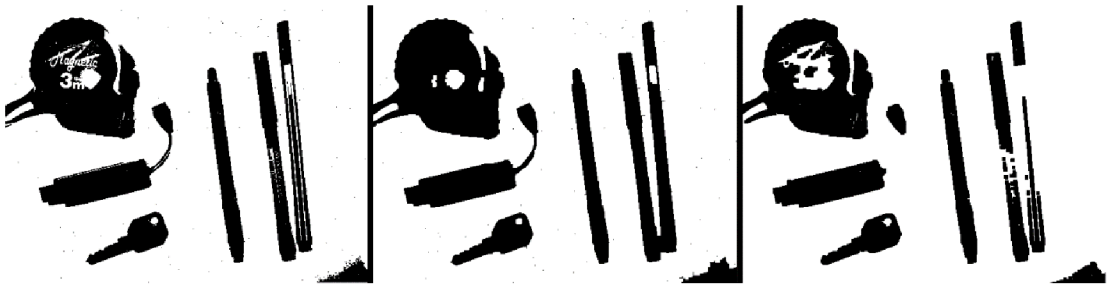
$$X \circ B = (X \ominus B) \oplus B . \tag{1.26}$$

Uzavření obrazu X odpovídá dilataci následované otevřením pomocí strukturního elementu B a je definováno rovnicí

$$X \bullet B = (X \oplus B) \ominus B . \tag{1.27}$$

Většinou se využívá ke spojení objektů, které leží blízko sebe, a dále pro vyplnění malých a tenkých otvorů v nich. Velikost těchto otvorů, které jsou zaplněny, je závislá na velikosti strukturního elementu.

Otevření a uzavření obrazu jsou operace idempotentní, což znamená, že vícenásobná aplikace těchto dvou operací již výsledný obraz dále nezmění. Ukázku morfologického otevření a uzavření ilustruje obrázek 1.14. [4][5]



Obrázek 1.14 Ukázka otevření a uzavření – vlevo originální obrázek, uprostřed otevření a vpravo uzavření.

1.7.3 Morfologická rekonstrukce

Morfologická rekonstrukce je založena na geodetické transformaci, která upravuje morfologické operace tak, aby pracovaly pouze na určité části obrazu. Základními operacemi jsou geodetická dilatace a geodetická eroze. Geodetická dilatace velikosti *n* množiny *Y* uvnitř množiny *X* má označení $\delta_X^{(n)}$. V případě, že se její velikost rovná 1, lze dilataci popsat rovnicí

$$\delta_X^{(1)} = (Y \oplus B) \cap X \,, \tag{1.28}$$

tedy jako průnik obrazu *X* s dilatací množiny *Y* pomocí strukturního prvku *B*. [5]

Při rekonstrukci máme množinu obrazu *X*, která obsahuje objekty. Některé z těchto objektů jsou označeny značkami, které jsou v množině *Y*. Následně provádíme postupnou geodetickou dilataci množiny *Y* v množině *X*. Ta je dokončena, až se při opakující aplikaci operace přestane měnit výstupní obraz. Rekonstrukci binárního obrazu [5] je možné matematicky vyjádřit rovnicí

$$\rho_X(Y) = \lim_{n\to\infty} \delta_X^{(n)}(Y) \,. \tag{1.29}$$

Pro šedotónový obraz se rekonstrukce provádí tak, že z obrazu ve stupních šedi získáme množinu naprahovaných binárních obrazů pomocí dekompozice $T_k(I)$ obrazu *I* uvedené v rovnici

$$T_k(I) = \{p \in D_I,\ I(P) \leq k\},\ k = 0 \ldots N, \tag{1.30}$$

kde D_I je definičním oborem obrazu a *N* je maximální hodnota jasu v obraze. Dekompozice se provádí postupným prahováním obrazu zvyšující se hodnotou prahu. Prahování je podrobně popsáno v kapitole 1.8.1. [5]

Morfologickou rekonstrukci v odstínech šedi $\rho_I(J)$ je pak možné matematicky popsat za použití dekompozice pomocí rovnice

$$\forall p \in D,\ \rho_I(J)(p) = max\{k \in [0, N],\ p \in \rho_{T_k}(T_K(J))\}, \tag{1.31}$$

kde J a I jsou obrazy v odstínech šedí s definičním oborem D, a zároveň hodnotami jasu v intervalu $[0, 1, \ldots, N]$. I je vstupní obraz, J je obraz odpovídající značkám, $T_k(J)$ je dekompozice obrazu J a ρ_{T_k} je binární rekonstrukce. Pro každý pixel p také musí platit, že $J(p) \leq I(p)$. [5]

1.8 Segmentace

Segmentace je nejpodstatnější částí při zpracovávání obrazu pro detekci silnice a dopravního značení. Jejím cílem je sjednotit části obrazu, které odpovídají hledaným objektům v obraze. Jeden z největších problému při segmentaci je nejednoznačnost obrazových dat doprovázená šumem. Segmentaci lze rozdělit do tří základních skupin podle toho, jakou metodu zpracování obrazu využívají. První skupina zahrnuje metody využívající globální znalost obrazu (histogramu), u druhé skupiny segmentace probíhá na základě hran v obraze a poslední skupinu tvoří segmentace založená na regionech. [5]

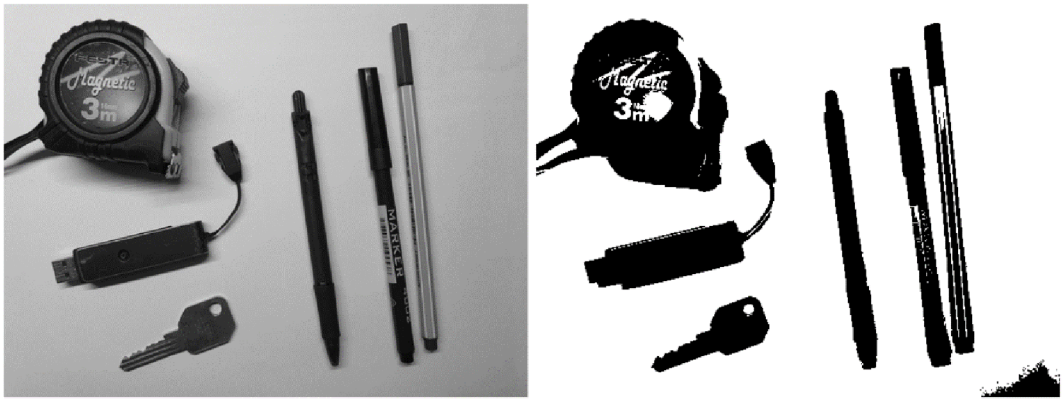
1.8.1 Segmentace prahováním

Cílem prahování obrazu je určit oblasti v obraze skládající se z podobné hodnoty jasu v odstínech šedi. Jedná se tedy o oblasti se stejnou nebo velmi podobnou barvou, odrazivostí a pohltivostí světla, které odpovídají zadaným kritériím. Oblasti se musí lišit od pozadí a neměly by se při tom výrazně překrývat, aby bylo možné jednotlivé objekty oddělit. Prahování představuje nejjednodušší a nejrychlejší metodu segmentace, kdy je potřeba pouze určit prahovou hodnotu T. Jedná se o transformaci vstupního obrazu $f(i, j)$ na výstupní obraz $g(i, j)$, která je definována jako

$$g(i,j) = \begin{cases} 1\ pro\ f(i,j) > T \\ 0\ pro\ f(i,j) \leq T \end{cases}. \tag{1.32}$$

Ukázka naprahovaného obrazu je vyobrazena na obrázku 1.15. Výsledkem je binární obraz, kde $g(i, j) = 255$ jsou oblasti objektu a $g(i, j) = 0$ je pozadí obrazu (nebo také naopak). [5][6]

Prahování je ale velmi náchylné na nerovnoměrné osvětlení nebo změnu světelných podmínek scény. V případě, že má obraz nerovnoměrně rozložený jas, je možné použít takzvané adaptivní prahování. Tato metoda rozdělí obraz do konečného počtu menších částí obrazu a pro každou oblast se určí individuální prahová hodnota T. Pokud v oblasti výrazně převažují buď pixely oblasti, nebo pixely pozadí, a zároveň hodnotu prahu nelze určit z histogramu, je možné použít interpolaci sousedních prahů. [11]



Obrázek 1.15 Ukázka naprahovaného šedotónového obrazu (vpravo), kde práh je zvolen pomocí histogramu

Velmi důležitým faktorem pro správné použití metody segmentace prahováním je správná volba prahové hodnoty, která se většinou provádí pomocí analýzy tvaru histogramu. V ideálním případě se jedná o bimodální histogram, který obsahuje dva vrcholy. Mezi nimi se pak nachází minimum, které zvolíme jako práh. Tato metoda je založena na skutečnosti, že se v obraze nachází pixely podobných jasových hodnot, jež náleží pozadí a hledaným oblastem. Rozhodnutí, zda je histogram bimodální nebo multimodální, není vždy zcela triviální. Pokud je obrázek tvořen černou a bílou polovinou, je jeho histogram k nerozeznání od histogramu obrazu, na kterém je šum typu pepř a sůl (náhodně rozmístěné černé a bílé pixely). [5][11]

1.8.2 Segmentace založena na regionech

Cílem segmentace založené na regionech je rozdělení obrazu do požadovaných oblastí. Základem metody je rozčlenění obrazu do maximálních souvislých oblastí, a to tak, aby byly podle zadaného kritéria homogenní. Kritérium homogenity oblastí může být založeno na jasu, barvě, textuře, tvaru nebo modelu. Kompletní segmentace obrazu R do konečného množství oblastí $R_1, \ldots, R_s$ je definováno pomocí rovnice

$$R = \bigcup_{i=1}^{S} R_i \, , R_i \cap R_j = \emptyset, i \neq j \, . \tag{1.33}$$

Dané oblasti musí být nejen homogenní, ale současně také maximální, což znamená, že po spojení s jakoukoli oblastí sousední se kritérium homogenity poruší. Mezi metody segmentace založené na regionech patří metoda spojovaní oblastí, metoda štěpení oblastí, kombinace zmíněných dvou metod a dále metoda rozvodí. [4][5]

Spojování oblastí je založeno na principu počátečního rozdělení obrazu do určeného počtu oblastí. Ty se následně spojují s okolními oblastmi tak dlouho, dokud není dodrženo kritérium homogenity. Výsledek zmíněné metody je ale závislý na tom, v jakém pořadí budeme oblasti testovat, zda splňují kritérium homogenity, a poté tyto oblasti spojovat. To znamená, že pokud začneme spojovat oblasti obrazu vpravo nahoře a budeme postupovat k levému dolnímu okraji, získáme jiný výsledek, než kdybychom

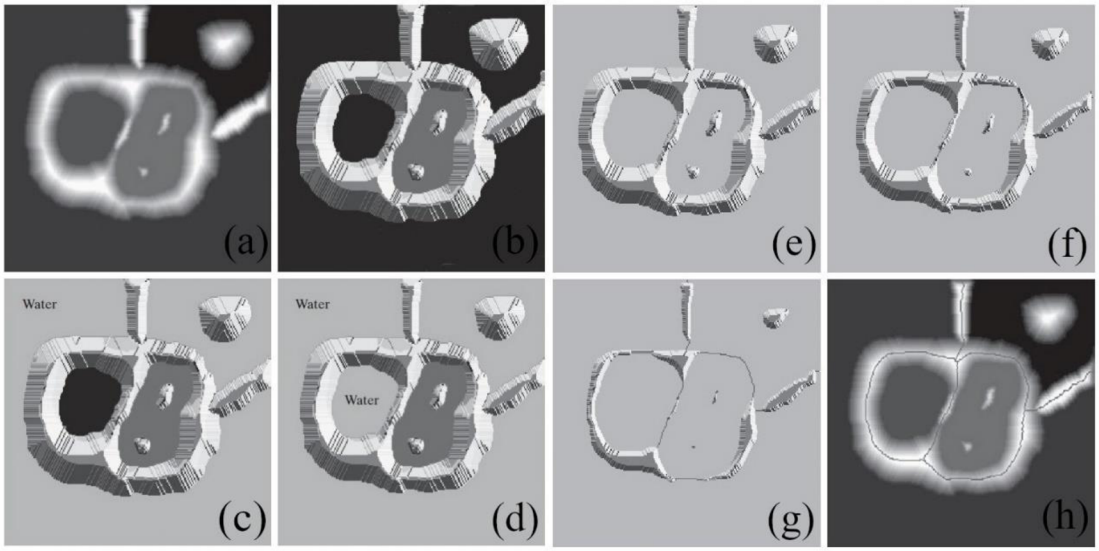
postupovali opačným směrem. Nejjednodušší metody popisují oblasti pomocí jejich statistických vlastností, což může být například maximální, minimální a průměrná hodnota jasu. Na základě tohoto popisu je rozhodnuto, zda se oblasti mají spojit, a pro nově vzniklou oblast se následně určí popis nový. Pokud již oblast nelze spojit s žádnou jinou okolní oblastí, jedná se o oblast konečnou. [4][5]

Štěpení oblastí je opakem metody spojování oblastí. Metoda na začátku reprezentuje obraz jako jednu velkou oblast, která je rozdělena do menších oblastí tak, aby bylo splněno kritérium homogenity. Většinou využívá podobná kritéria homogenity, jako metoda spojování oblastí popsaná výše. Přestože se při segmentaci štěpením jedná o postup opačný vzhledem k segmentaci spojováním, ani zde při použití stejného kritéria není dosaženo shodného výsledku segmentace. [5]

1.8.3 Rozvodí

Segmentace obrazu pomocí rozvodí (angl. *watershed*) je založena na principu interpretace vstupního obrazu ve stupních šedi jako topografická plocha, kde tmavé odstíny reprezentují údolí a světlé odstíny vrcholy [5]. V tomto zobrazení uvažujeme tři typy bodů obrazu. První skupinou jsou body patřící regionálnímu minimu (tmavší odstíny šedé). Dále jsou to body nazývané rozvodí minima. Pokud je v nich umístěna kapka vody, klesne tato kapka do regionálního minima. Posledním typem jsou body, ve kterých by kapka klesla se stejnou pravděpodobností do dvou nebo více regionálních minim, a ty pak tvoří vrcholy topografického zobrazení obrazu zvané hranice rozvodí (světlejší odstíny šedé). [6]

Obrazy však ve většině případů nejsou tvořeny tmavými oblastmi, jež jsou odděleny světlejšími hřebeny, a proto se rozvodí aplikuje nejčastěji na vyhlazený gradientní obraz, nebo na obraz upravený pomocí morfologických transformací. [9]

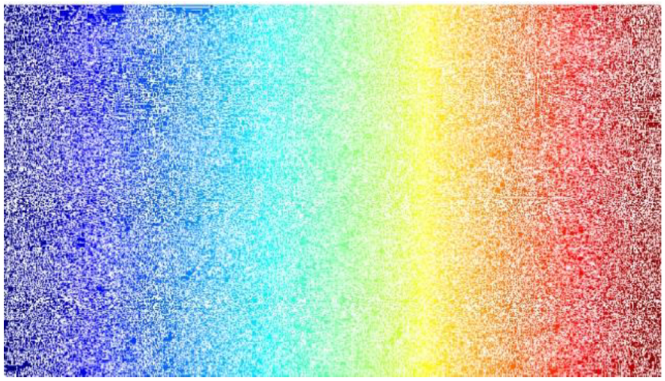


Obrázek 1.16 Postup segmentace pomocí metody rozvodí [6]

Princip metody rozvodí můžeme popsat s využitím obrázku 1.16. Na dílčím obrázku 1.16 (a) můžeme vidět vstupní obraz pro tuto segmentační metodu. Obrázek 1.16 (b) pak znázorňuje zmíněný vstupní obraz v topografickém zobrazení, kde výška pohoří odpovídá úrovni jasu vstupního obrazu. Lokální minima topografického zobrazení jsou vizualizována černou barvou. Následně začne v obrazu z lokálních minim stoupat rovnoměrnou rychlostí voda a obraz je postupně zaplavován. První krok zaplavování je vidět na dílčím obrázku 1.16 (c), kde je stoupající voda vykreslena světle šedou barvou. Obrázky 1.16 (d) a (e) znázorňují postupné stoupání vody v povodích, a také stav, kdy voda již dosáhla takové výšky, že zalila také další dvě lokální minima obrazu. Posupné zvyšování hladiny je vidět na obrázku 1.16 (f). Důsledkem pak je, že jednotlivá povodí přetečou z jednoho do druhého. Na obrázku 1.16 (f) je patrné, že levé prostřední povodí přeteklo do sousedního pravého povodí. V tomto místě se postaví vysoká hráz o velikosti jednoho pixelu, který reprezentuje maximální hodnotu jasu, a tím zabráníme mísení vody z daných dvou povodí. Se stále stoupající hladinou vody se mezi jednotlivými povodími jednak hráze prodlužují, a jednak se tvoří nové hráze v místě jejich spojení. Tuto situaci znázorňuje dílčí obrázek 1.16 (g). Zvyšování hladiny vody pokračuje až do dosažení úrovně nejvyšší intenzity jasu bodu, kterou obraz obsahuje. V tuto chvíli jsou v obraze hráze, které tvoří hranice oblastí výsledné segmentace. Čáry reprezentující hráze poskytují kontinuální hranice mezi jednotlivými oblastmi, jež odpovídají jednotlivým objektům. Výsledek segmentace metodou rozvodí je na obrázku 1.16 (h) zobrazen pomocí tmavých čár tloušťky jednoho pixelu překrývajících vstupní obrázek. [6]

Pokud je pro segmentaci použit obraz převedený přímo do odstínů šedi, výsledkem je ve většině případů přesegmentovaný obraz (obrázek 1.17), který je způsoben především šumem obrazu. Abychom zmíněnému problému předešli, používají se při segmentaci pomocí rozvodí značky (angl. *markers*). Ty můžeme rozdělit do dvou skupin: interní značky, které označují objekty v obraze, a značky externí označující pozadí. Značky se většinou vybírají buď pomocí metod předzpracování obrazu, morfologie, nebo pomocí definovaných kritérií, která musí značky splňovat. Objekt, jenž chceme na obrázku detekovat, by měl být označen interní značkou, která je obvykle umístěna v jeho prostřední části. Při následné aplikaci algoritmu povodí se značkami jsou interní značky jedinými regionálními minimy, ze kterých hladina vody stoupá. Pokud se spojí dvě povodí se stejnou značkou, nevytvoří se mezi nimi hráz. Díky popsanému postupu je možné předejít problému s přesegmentováním obrazu, a dosáhnout tak dobrých výsledků segmentace. [5][6]

Transformaci rozvodí lze také prezentovat v oblasti matematické morfologie, avšak v tomto případě je segmentace výpočetně i časově náročnější. [5]



Obrázek 1.17 Výsledek aplikace povodí bez značek na gradientní obraz, kde je zřejmé přesegmentování obrazu

1.9 Neuronové sítě

Neuronové sítě, nazývané také dopředné sítě, se využívají k aproximaci dané funkce, a to tak, aby ve výsledku bylo dosaženo určitého zobecnění. Název neuronové sítě odráží skutečnost, že neurony v těchto sítích jsou inspirovány biologickým neurony. Označení sítí jako dopředné pak vychází z toho, že při vybavování sítě signály proudí od vstupu přes jednotlivé neurony až na výstup sítě bez zpětných vazeb. Neuronová síť se skládá nejčastěji ze vstupní vrstvy, skrytých vrstev a výstupní vrstvy neuronů. Neurony v jednotlivých vrstvách přijímají vstup od všech neuronů v předchozí vrstvě a počítají svou aktivační hodnotu. Správnou aproximaci námi požadované funkce sítí zajišťuje učící algoritmus. Ten systematicky předkládá tréninkové vzory síti a upravuje váhy vstupů jednotlivých neuronů tak, aby byla funkce co nejlépe aproximována. [12]

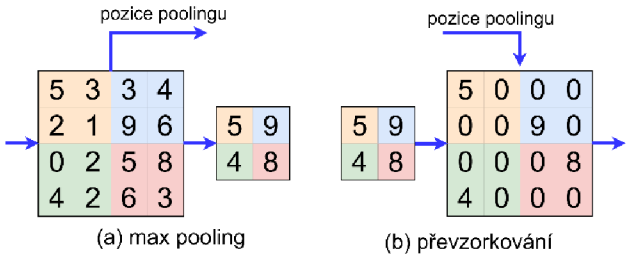
1.9.1 Konvoluční neuronové sítě

Speciálním případem dopředné sítě je konvoluční neuronová síť pro zpracování dat, která mají topologii podobnou mřížce. Mřížkové topologii odpovídají kupříkladu data obrazová, a proto je tato síť hojně využívána ke klasifikaci objektů v obraze nebo pro sémantickou segmentaci. Jak již napovídá název, konvoluční neuronovou síť lze charakterizovat jako síť, která používá alespoň v jedné ze svých vrstev operaci konvoluce (1.6) nebo korelace definované vztahem

$$f(i,j) = \sum\sum_{(m,n)\in O} g(i+m, j+n)h(m,n)\ , \tag{1.34}$$

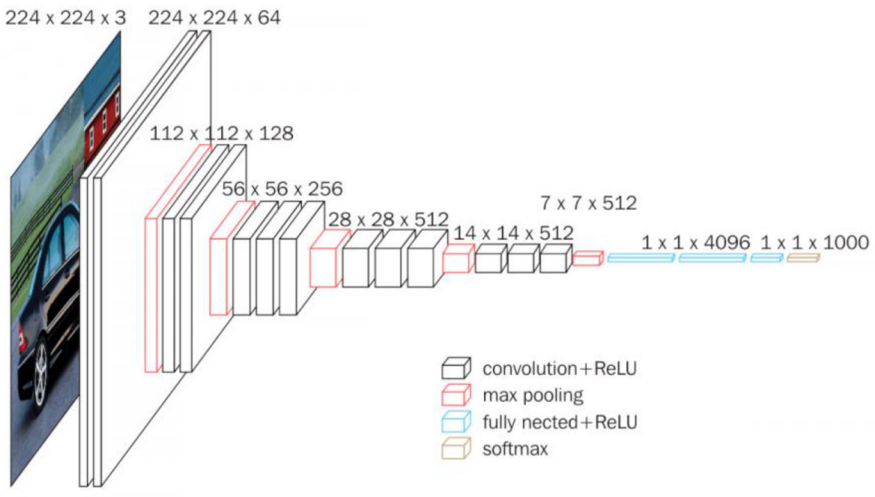
kde, g je obraz a h je maska. Mnoho sítí implementuje místo operace konvoluce operaci korelace, avšak nazývá jí také konvolucí. Dále v této kapitole tedy budou obě operace označovány termínem konvoluce. Výhodou konvolučních sítí je, že vstupní obraz může mít tisíce řádků i sloupců, ale konvoluční masky mají mnohonásobně menší velikost, například jen 9x9 pixelů. Pomocí masek jsou v obraze nalezeny výrazné rysy, například hrany či geometrické útvary. Pokud bychom použili dopřednou neuronovou síť,

u obrazu o velikosti 1 000x1 000 pixelů, by bylo potřeba uložit ve vstupní vrstvě 1 000 000 neuronů a jejich parametrů. Při konvoluci se však ukládájí pouze masky a obraz, na který byly aplikovány, což razantně snižuje paměťové i výpočetní nároky. [12]



Obrázek 1.18 Princip max poolingu (a) a převzorkování (b) v konvoluční síti SegNet [13]

Další podstatnou operací v konvolučních neuronových sítích je takzvaný *pooling*, který používají téměř všechny konvoluční sítě. Typická vrstva takové sítě se skládá ze tří fází zpracování obrazu. Nejprve se provede několik konvolucí s využitím konvolučních masek. Následně je na obrazy získané konvolucí, také zvané aktivace, aplikována nelineární funkce. V poslední fázi přichází na řadu pooling. Pooling označuje funkci, která provede nahrazení oblasti pixelů statistickou funkcí. Hojně používaný je takzvaný *max pooling*, který provede substituci oblasti pixelů, povětšinou čtvercového tvaru, maximální hodnotou pixelu z dané oblasti. Ukázka max poolingu je zachycena na obrázku 1.18 (a). V praxi se dále často využívá pooling s průměrováním hodnot. Pokud je tedy vstupní oblast pixelů obrazu pro pooling o velikosti 4x4, výsledkem je jeden pixel. Pooling je považován za velmi užitečnou operací, zejména z toho důvodu, že přináší invarianci vůči posunutí. Jeho další přínos spočívá ve zmenšení aktivací, a to minimálně na polovinu původní velikosti. [12]

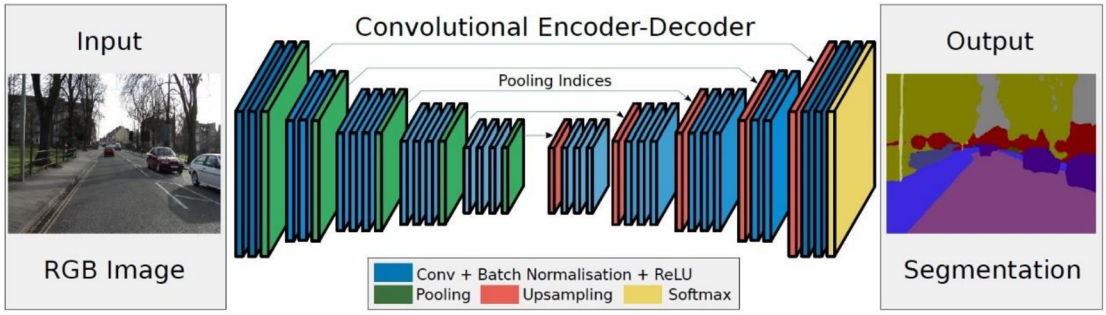


Obrázek 1.19 Architektura konvoluční neuronové sítě VGG-16 [14]

Po několika vrstvách konvoluce a poolingu je výstup přiveden na plně propojenou dopřednou neuronovou síť, která zajistí klasifikaci. Ukázka architektury konvoluční neuronové sítě VGG-16 určené pro klasifikační účely je zobrazena na obrázku 1.19.

1.9.2 Enkodéry-dekodéry

Po účely sémantické segmentace se často používají plně konvoluční sítě typu enkodér-dekodér bez plně propojené vrstvy sítě. Příkladem takové sítě je síť SegNet [13] navržená na Cambridgeské univerzitě. Její architektura je znázorněna na obrázku 1.20. První polovina sítě (enkodéry) se skládá z 13 konvolučních a max poolingových vrstev, které odpovídají prvním 13 předtrénovaným konvolučním vrstvám ze sítě VGG-16 [15], přičemž plně propojená vrstva zde není použita. Při aplikaci max poolingu však dochází ke ztrátě informací o původní aktivaci, a proto je pokaždé uchována v paměti pozice prvku při poolingu s maximální hodnotou. V druhé polovině sítě (dekodérech) je výstup z první poloviny sítě převzorkován (angl. *upsamling*) za pomoci zaznamenané pozice poolingu, jak je vidět na obrázku 1.18 (b). Na takto převzorkovaný obraz jsou následně aplikovány trénovatelné masky určené pro dekódování. Poslední vrstva dekodéru, na obrázku 1.20 vyznačená žlutou barvou, je trénovatelný klasifikátor, který rozdělí pixely obrazu do předem daných tříd. [13]



Obrázek 1.20 Architektura konvoluční sítě typu enkodér-dekodér SegNet [13]

Jinou velmi podobnou sítí s architekturou enkodér-dekodér je například DeconvNet [16]. DeconvNet má však mnohem větší možnosti parametrizace, je tudíž složitější, a zároveň potřebuje větší výpočetní výkon než síť SegNet [13]. Dále je vhodné zmínit síť ENet [17], která má mnohem menší hardwarové nároky než SegNet nebo DeconvNet. Při porovnání rychlosti segmentace na videu v rozlišení 1280x720 a grafické kartě NVIDIA Titan X potřebuje síť ENet průměrně pouze 21 ms na zpracování jednoho snímku, zatímco SegNet celých 289 ms. [17]

1.10 Metriky pro určení přesnosti sémantické segmentace

Cílem sémantické segmentace je přiřazení každého pixelu obrazu k jedné z předem daných tříd, například k vozovce a pozadí, nebo k žádné z tříd. Pro zhodnocení úspěšnosti segmentace nebo klasifikace je potřeba zavést metriku, která je schopna určit, jak dobře algoritmus segmentaci provedl. Nejčastěji se pro vyhodnocení sémantické segmentace používá porovnání výsledku segmentace získaného pomocí testovaného algoritmu a obrazu pravdivě segmentovaného (angl. *ground truth*) na úrovni pixelů. Metody pro hodnocení přesnosti segmentace můžeme rozdělit na dva typy, a to přesnost založenou na oblastech a přesnost založenou na obrysech. [18]

Tabulka 1.1 Matice záměn [19]

třída	klasifikováno jako pozitivní	klasifikováno jako negativní
skutečně pozitivní	správně pozitivní (tp)	nesprávně negativní (fn)
skutečně negativní	nesprávně pozitivní (fp)	správně negativní (tn)

Pro definici metrik sémantické segmentace zavedeme takzvanou matici záměn (angl. *confusion matrix*), která je uvedena v tabulce 1.1. Tato matice definuje čtyři různé typy klasifikovaných pixelů v obraze. Prvním typem pixelů je správně rozpoznaná třída (správně pozitivní - tp). Druhým typem jsou správně rozpoznané pixely, které do třídy nepatří (správně negativní - tn). Třetím typem jsou pixely, které byly rozpoznány jako třída, ale nepatří do ní (nesprávně pozitivní - fp). A posledním typem jsou pixely, které byly nesprávně rozpoznány tak, že do třídy nepatří (nesprávně negativní - fn). [19]

Mezi základní metriky segmentace a klasifikace patří přesnost definována vztahem

$$přesnost = \frac{tp}{tp+fp}, \tag{1.35}$$

kde *tp* a *fp* jsou počty pixelů spadající do jednotlivých tříd matice záměn uvedené v tabulce 1.1. Přesnost určuje, jak moc můžeme věřit tomu, že jsou pixely rozpoznané algoritmem správně jako hledaná třída. Opakem je citlivost vyjádřena pomocí rovnice

$$citlivost = \frac{tp}{tp+fn}, \tag{1.36}$$

kde *tp* a *fn* jsou počty pixelů spadající do jednotlivých tříd matice záměn (tabulka 1.1). Citlivost vyjadřuje, jak dobře je algoritmus schopen identifikovat pixely hledané třídy. [19]

Jednou z nejpoužívanějších metrik, kterou lze nazvat defacto standardem, je podíl průniku a sjednocení (angl. *intersection over union, IoU*), také známý jako Jaccardův index (JI). Metrika IoU se vypočítá jako

$$IoU = \frac{tp}{tp+fn+fp}, \tag{1.37}$$

kde *tp*, *fn* a *fp* jsou počty pixelů spadající do jednotlivých tříd matice záměn (tabulka 1.1). Tato metoda spadá do metrik založených na oblastech a kombinuje přesnost (1.35) a citlivost (1.36) popsanou výše. [18][20]

F-skóre (angl. *F-score*) je další používanou metrikou pro určení přesnosti sémantické segmentace. Jedná se o harmonický průměr přesnosti (1.35) a citlivosti (1.36). F-skoré se obecně vypočítá jako

$$F_{skóre} = \frac{(\beta^2+1)tp}{(\beta^2+1)tp+\beta^2 fn+fp}, \tag{1.38}$$

kde β je koeficient pro škálování mezi přesností a citlivostí, *tp*, *fn* a *fp* jsou počty pixelů spadající do jednotlivých tříd matice záměn (tabulka 1.1). Speciálním případem je velmi často používané F1-skóre, kdy v rovnici (1.38) je $\beta = 1$. Toto skóre je pak definováno jako

$$F_1 = \frac{2tp}{2tp+fn+fp} \tag{1.39}$$

a podstatně více zohledňuje správně identifikované pixely než pixely, které byly identifikovány nesprávně. [19][20]

2. NÁSTROJE PRO ZPRACOVÁNÍ OBRAZU

Tato kapitola popisuje nástroje určené pro zpracování obrazu se zaměřením na nástroje použité v této práci. Nejdříve je zde popsán programovací jazyk Python. Následně jsou uvedeny knihovny určené pro Python, a zároveň sloužící ke zpracování obrazu (OpenCV a scikit-image). Podstatnou roli v této práci má také knihovna NumPy, sloužící pro práci s maticemi, k vědeckým výpočtům a pro zpracování dat.

2.1 Programovací jazyk Python

Python je univerzální programovací jazyk s podporou objektově orientovaného programování, který se často používá pro tvorbu skriptů. Tento jazyk je zaměřen na co největší čitelnost, pochopitelnost a snadnou udržovatelnost kódu. Další jeho výhodou je vysoká přenositelnost programů, kdy většinu programů v jazyce Python lze spustit beze změny na téměř všech dnešních hlavních počítačových platformách. Python je dodáván s rozsáhlou standardní knihovnou pro široké spektrum úloh, a kromě toho je do něj možné přidat nepřeberné množství knihoven třetích stran, například pro zpracování obrazu, vykreslování grafů, neuronové sítě, nebo práci s daty. Tento programovací jazyk navíc umožňuje volání knihoven jazyka C a C++, a také podporuje použití komponent v jazycích Java a .NET. Díky tomu mohou být složité operace v knihovnách předkompilovány, a poté prováděny ve velmi rychlém jazyce C nebo C++. [21]

Python je využíván mnoha známými společnostmi po celém světě. Příkladem může být NASA, která jej používá pro vědecké účely, nebo také společnosti Intel, Cisco a Qualcomm s využitím pro testování hardwaru. [21]

2.2 Knihovny pro programovací jazyk Python

Pro programovací jazyk Python existuje velké množství knihoven určených pro zpracování obrazu a jeho vizualizaci. V této kapitole jsou popsány nejznámější knihovny a knihovny použité v praktické části práce.

2.2.1 OpenCV

OpenCV je open source knihovna učená pro zpracování obrazu, videa a pro strojové učení. Je napsána v jazyce C++ a má rozhraní pro programovací jazyky C++, Java, Python a MATLAB. Operační systémy, které knihovna podporuje, jsou Windows, Linux, Android a Mac OS. Máme zde k dispozici více než 2500 optimalizovaných algoritmů pro aplikaci počítačového vidění a strojového učení. Knihovna byla poprvé vydána po několika beta verzích v roce 2006, a to ve verzi 1.0. OpenCV 2 pak byla vydána roku 2009, přičemž tato verze je stále vyvíjena, podporována a jsou do ní přidávány nové a optimalizované algoritmy. Velká část komunity odborníků a vývojářů v oblasti počítačového vidění používá knihovnu OpenCV jako primární vývojový nástroj. [22][23]

Nejzajímavějšími algoritmy, které najdou uplatnění v této práci, jsou funkce a třídy pro prahování, rozmazání obrazu a detekci hran. Práce obsahuje také základní segmentační metody, včetně metody segmentace rozvodím či základní morfologické operace a převody mezi barevnými modely.

Součástí knihovny je také podpora OpenCL, díky které může kód běžet jak v procesoru, tak na grafické kartě počítače. Tato skutečnost přináší možnost velkého paralelismu výpočtů na GPU, s čímž souvisí velké zrychlení mnoha algoritmů zpracování obrazu, zejména těch, které používají maticové výpočty. [22]

2.2.2 Scikit-image

Scikit-image je vědecká knihovna pro zpracování obrazu. Ta obsahuje sadu algoritmů pro programovací jazyk Python, které jsou k dispozici bez omezení zdarma. Knihovna je snadno použitelná, velmi dobře dokumentovaná a pro práci s obrazy a maticemi využívá známou knihovnu NumPy, díky čemuž je mezi vývojáři velice oblíbená. [24]

Knihovna obsahuje některé algoritmy, jež v OpenCV implementovány nejsou. Jedná se například o morfologickou rekonstrukci. Scikit-image však bohužel nepodporuje běh algoritmů na grafické kartě. Pro účely zpracování dat v praktické části práce bude tedy nejlepší knihovny OpenCV a scikit-image vhodně kombinovat.

2.2.3 NumPy

NumPy je open source knihovna určená pro vědecké výpočty založené na polích v programovacím jazyce Python. Knihovna poskytuje velké množství funkcí, které jsou implementovány v softwaru Matlab nebo Mathematica. Podstatná část této knihovny je napsána v jazyce C, přičemž operace se provádí v již předem zkompilovaném kódu. Proto je práce s poli a maticemi NumPy rychlejší než v případě, že by byla použita ekvivalentní datová struktura čistě v programu jazyka Python. [25]

Základem knihovny je objekt *ndarray*, který zapouzdřuje vícerozměrné pole jednoho datového typu. To pak reprezentuje pole, matici, vektor nebo n-dimenzionální pole. Základní vlastností polí NumPy je jejich pevná velikost, a pokud je tedy potřeba do pole například přidat prvek, musíme vytvořit nové pole a původní smazat. Navíc dané pole musí vždy obsahovat prvky jen jednoho datového typu se stejnou velikostí v paměti. [26]

Pole NumPy používají obě výše zmíněné knihovny pro zpracování obrazu OpenCV i scikit-image jako výchozí datový typ pro práci s obrazem. Dá se říci, že většina dnes využívaného matematického a vědeckého softwaru určeného pro Python pracuje s knihovnou NumPy. V této práci budou tedy primárně uplatněny matice a vektory právě knihovny NumPy. [26]

2.2.4 Matplotlib

Matplotlib je velmi široce používaná knihovna pro vykreslování 2D i 3D grafů v jazyce Python. Je inspirována MATLABem a používá se téměř stejně. Knihovna je napsána v jazyce Python a využívá pole NumPy. Podporuje mimo jiné zobrazení obrazu. [27]

3. TEORETICKÝ NÁVRH SYSTÉMU ZPRACOVÁNÍ DAT

V této kapitole je rozebrán návrh systému zpracování obrazu nasnímaného z automobilu. Navržený systém segmentuje vozovku a detekuje horizontální dopravní značení (jízdní pruhy) za pomocí teorie a principů popsaných výše. V návrhu je kladen důraz na co nejkvalitnější výsledky zpracování obrazu z reprezentativního datasetu a to tak, aby celý systém bylo možné implementovat v programovacím jazyce Python za použití algoritmů obsažených v knihovnách OpenCV a scikit-image (popsaných v kapitole 2).

3.1 Reprezentativní dataset

Reprezentativní dataset pro testování a návrh systému segmentace a detekce jízdních pruhů v obraze byl sbírán na silnicích v Moravskoslezském kraji, přesněji v okolí měst Frýdek-Místek a Třinec. Nahrávaní vzorků videa bylo provedeno mobilním telefonem Samsung Galaxy S10+ uchyceném pomocí držáku v horní části středu čelního skla automobilu. Telefon disponuje hlavním obrazovým snímačem s rozlišením 12 MP, světelností f/1.5-2.4 a ohniskovou vzdáleností 26 mm. Druhý použitý snímač má rozlišení taktéž 12 MP, světelnost f/2.4 a ohnisková vzdálenost je 52 mm. Oba snímače obsahují také optickou stabilizaci obrazu. Video bylo nahráváno v rozlišení 1280x720 pixelů s třiceti snímky za sekundu, přičemž obrázky mají rozlišení totožné s videem, tedy 1280x720 pixelů.

V rámci tvorby datasetu byly nahrány celkem 3 hodiny a 35 minut záznamů. Z toho bylo pro reprezentativní dataset v příloze A.2 na DVD vybráno 30 minut videí a 389 obrázků.

3.1.1 Vlastnosti datasetu

Reprezentativní vzorek videí a obrázků byl sbírán ve dne a je v něm zahrnuto mnoho typů silnic. Obsahuje silnice jak s horizontálním dopravním značením (čáry, které definují jízdní pruhy) na obou stranách jízdního pruhu, tak se značením pouze uprostřed silnice. Dále jsou zde zastoupeny i vzorky silnic bez horizontálního dopravního značení. Několik vzorků dat také obsahuje dálnice a silnice vyšších tříd s více jízdními pruhy v jednom směru. Ukázka z datasetu je zobrazena na obrázku 3.1.

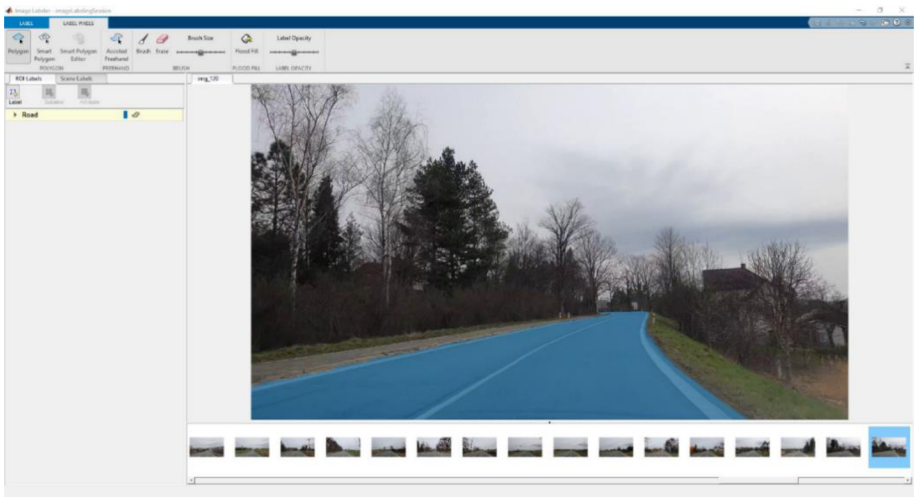
Obrázek 3.1 Ukázka různých scén z reprezentativního datasetu

Dataset a navržený algoritmus jsou omezeny na videa pořízena ve dne za běžného provozu, aby silnice nebyla příliš zastíněna projíždějícími vozidly. Snímky neobsahují záběry, na kterých je slunce těsně na obzorem (tedy při východu a západu slunce). Důvodem je skutečnost, že by se vytvářely odlesky na čelním skle, které by segmentaci a detekci jízdních pruhů velmi ztěžovaly. Součástí datasetu nejsou lesní, polní nebo jiné nezpevněné komunikace, avšak pouze silnice s asfaltovým nebo betonovým povrchem.

3.1.2 Anotace snímků

Pro otestování přesnosti segmentace bylo náhodně vybráno 171 snímků v rozlišení 1280x720 pixelů z videí datasetu, a následně v nich bylo ručně vyznačeno, které pixely naleží vozovce a které pozadí. Pro účely anotace byly testovány dva nástroje, jež jsou popsány v následujících odstavcích.

Prvním testovaným nástrojem byl open source *VGG Image Annotator* vytvořený na univerzitě v Oxfordu. Tento nástroj pro označení silnice umožnuje anotaci polygonem nebo obdélníkem. Avšak poté, co proběhlo testování pro vytvoření značek silnice, bylo zjištěno, že tento typ manuální anotace není nejrychlejší a nejpřesnější. Proto bylo potřeba najít jiný, propracovanější nástroj pro anotaci. [28]

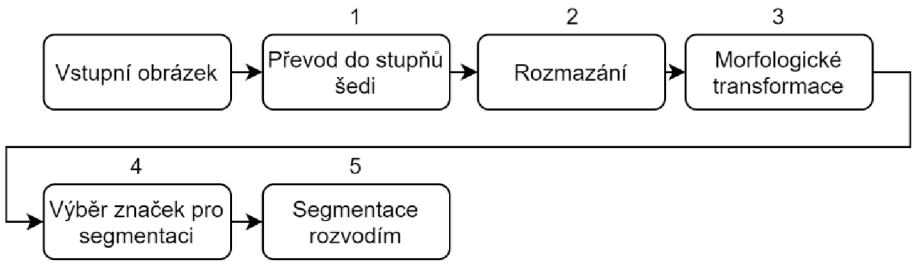


Obrázek 3.2 Program Matlab Image Labeler pro ruční označení silnice

Druhým nástrojem, který se jevil jako vhodnější varianta, a také byl nakonec pro vytvoření značek použit, byl MATLAB a jeho *Image Labeler* v toolboxu *Computer Vision Toolbox*. Jeho nesporná výhoda spočívá v tom, že pro označení obrázků můžeme využít kromě použití polygonů a obdélníků, také možnost vybarvení oblasti pixelů náležící určité třídě jednoduše pomocí štětce viz obrázek 3.2. Ruční vyznačování oblastí pomocí štětce se pro tento projekt ukázalo být nejjednodušší a nejrychlejší variantou, a nebylo tedy pochyb o vhodnosti využití programu Matlab Image Laberer. Z tohoto nástroje lze označené obrázky exportovat ve formátu PNG, a následně je rovnou srovnat s výsledkem sémantické segmentace pomocí navrženého testovacího algoritmu. [29]

3.2 Segmentace vozovky

První částí navrženého postupu je segmentace vozovky od pozadí obrazu za použití segmentační metody rozvodí. Diagram návrhu je zaznamenán na obrázku 3.3. Charakteristický je pro něj nejprve převod vstupního obrazu vozovky z datasetu na šedotónový, neboť navržené zjednodušení a filtrace v dalších krocích zpracování vyžaduje vstupní obraz v odstínech šedi.



Obrázek 3.3 Blokový diagram navrženého postupu segmentace vozovky pomocí metody rozvodí

Následně je na obraz pomocí konvoluce (1.6) aplikován Gaussův vyhlazovací filtr velikosti 3×3 uvedený v rovnici (1.8). Velikost Gaussova filtru může být v pozdějších fázích návrhu ještě upravena na velikost 5×5 v závislosti na šumu vstupního obrazu. Tato Filtrace zajistí odstranění šumu z obrazu, a bude tak dosaženo lepší segmentace pomocí rozvodí. Ta je na šum velmi citlivá, jelikož způsobuje přesegmentování obrazu popsané v kapitole 1.8.3.

Na vyhlazený šedotónový obraz je poté použita morfologická eroze, a dále je obraz morfologicky rekonstruován. Dalším krokem je aplikace morfologické dilatace obrazu, a následuje jeho morfologická rekonstrukce, čímž je dosaženo zjednodušeného obrazu, který obsahuje mnohem méně detailů. Ukázka takto upraveného obrazu je vidět na obrázku 3.4. Daný postup zajistí ještě větší omezení nežádoucího přesegmentování výsledného obrazu. Velikost a tvar použitého strukturního elementu budou vybrány při implementaci a testování algoritmu v navazující fázi práce. Použité morfologické operace je potřeba důkladně otestovat, a případně navrhnout jejich vylepšení nebo změnu. [30]

Obrázek 3.4 Ukázka zjednodušeného obrazu pomocí morfologických transformací

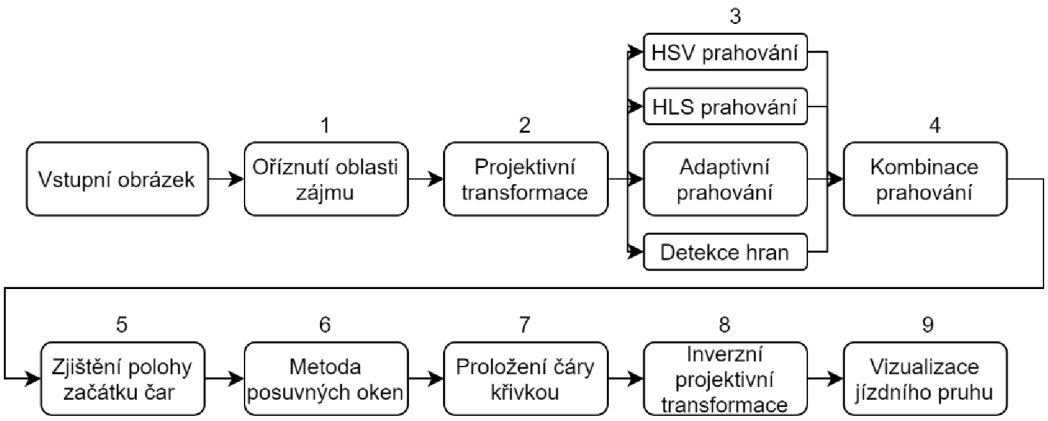
Značky označující silnici a pozadí v obrazu budou zvoleny na základě znalosti scény. Předpokládá se, že vozidlo je vždy na silnici. Z toho lze heuristicky usoudit, že ve středu dolní části obrazu je vždy silnice. Do tohoto místa je tedy umístěna značka silnice. Pozadí, do kterého patří obloha, stromy, hory a budovy se bude s jistotou nacházet v celé horní části obrazu. Tam je umístěna značka, která označuje pozadí obrazu.

Na obraz upravený pomocí morfologických operací se aplikuje algoritmus povodí se značkami definovanými výše. Ten ve výsledku z obrazu segmentuje vozovku. Metoda segmentace pomocí povodí byla zvolena primárně z toho důvodu, že přináší stabilní výsledky, a také je méně citlivá na změny osvětlení scény. [6]

Část návrhu, ve které se obraz filtruje, a dále kde počítá se gradient a povodí, může být realizována pomocí knihovny OpenCV. Morfologické transformace pak budou provedeny pomocí scikit-image, jelikož tato knihovna obsahuje i potřebnou morfologickou rekonstrukci.

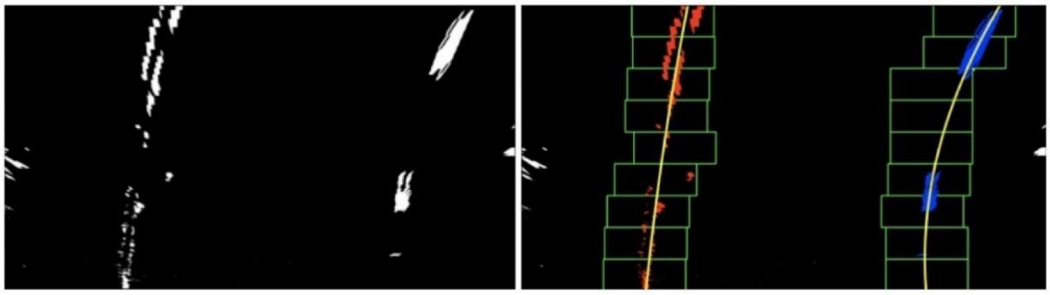
3.3 Detekce horizontálního dopravního značení

Na část popisující segmentaci silnice navazuje část, jejímž cílem je detekovat v obraze jízdní pruhy. Blokový diagram tohoto postupu je znázorněn na obrázku 3.5. Nejprve je uplatněn výsledek segmentace pomocí povodí pro určení oblasti zájmu (tou je myšlena silnice), ve které je potřeba jízdní pruhy detekovat. Oblast je tedy obdélníkově oříznuta tak, aby se v zde nacházela celá detekovaná vozovka. Díky tomu nebude potřeba zpracovávat celý obraz. V důsledku dojde k žádoucímu zvýšení rychlosti i přesnosti detekce. Dále je oříznutý obraz perspektivně transformován statickou maticí transformace tak, aby byla silnice zobrazena pohledem shora. To je důležité pro mnohem jednodušší detekci pruhů v následující části práce, a zároveň jejich aproximaci pomocí křivky.



Obrázek 3.5 Blokový diagram navrženého postupu detekce horizontálního dopravního znační v obraze [31]

Na obraz je poté aplikováno několik metod, díky kterým bude možné jízdní pruhy detekovat. První možnost segmentace pruhů využívá prahování barevného modelu HSV, kdy jsou nastaveny prahy (omezen rozsah) jednotlivých složek H, S a V, a to tak, aby bylo v obraze získáno jen horizontální dopravní značení. Totéž je možné udělat s modelem HLS. Nastavení prahových hodnot barevných modelů se dá provést manuálně na základě nejlepších výsledků testování vzorků z datasetu. Další metodou je získání obrysu pruhů pomocí gradientního obrazu, například po aplikaci konvoluce (1.6) horizontálním Sobolovým filtrem (1.12). Zde lze vycházet z poznatku, že světlé čáry na tmavé silnici mají na rozhraní s vozovkou velký gradient v horizontálním směru. Tento gradientní obraz je pak naprahován vhodným prahem, čímž se získá binární obraz obsahující hrany jízdních pruhů. Poslední uplatněnou metodou je adaptivní prahování s poměrně vysoko nastaveným prahem přímo na šedotónový obraz vozovky. [31]



Obrázek 3.6 Naprahovaný obraz obsahující čáry značení (vlevo) a aplikace metody posuvných oken a proložení křivkou (vpravo) [32]

Na základě metod popsaných výše jsou získány čtyři binární obrazy, které je možné zkombinovat tak, že ve výsledném obraze bude mít daný bod se souřadnicemi *(i, j)* hodnotu 255, a to za předpokladu, že se bod o této hodnotě jasu nachází na stejných souřadnicích *(i, j)* alespoň ve dvou ze čtyř vstupních binárních obrazů. Tím bude dosaženo zpřesnění detekce pruhů, jelikož se omezí body, které byly jednotlivými metodami chybně detekovány jako čára na silnici. Takový binární obraz ilustruje obrázek 3.6 (vlevo).

Dále je nutné horizontální silniční značení aproximovat křivkou. Pro zjištění počáteční polohy čar jízdních pruhů ve spodní části obrazu lze použít postup, při kterém se sečtou hodnoty bodů v každém sloupci spodní třetiny obrazu a uloží se do vektoru. Spodní třetina obrazu je vybrána proto, že zásadní roli hraje pouze poloha značení ve spodní části obrazu, a dále z důvodu zajištění, že na určení počáteční polohy značení nemá vliv směr nebo průběh čáry ve větších vzdálenostech od vozidla. V takto získaném vektoru pak lze najít dvě maxima, jedno v levé a druhé v pravé polovině vektoru. Maxima pak umožní určení polohy jízdních pruhů u spodní hrany obrazu. V této fázi je velkým přínosem, že je obraz projektivně transformován, díky tomu lze mnohem snadněji nalézt počátky čar na vozovce, i když se jedná o přerušované čáry. [32]

Na předchozí postup navazuje další krok, ve kterém se do počátku čar na vozovce umístí takzvaná posuvná okna. Okna se budou v obraze posouvat směrem vzhůru a sledovat jízdní pruhy (zelené obdélníky na obrázku 3.6 vpravo). Princip sledování značení na vozovce spočívá v tom, že nad výchozí okno se umístí další okno tak, aby jeho střed ležel v horizontální poloze odpovídající poloze s největším výskytem bílých pixelů v předchozím okně. Tímto způsobem se okna skládají na sebe až dosáhnou horního okraje obrazu. Poté jsou uloženy souřadnice všech pixelů nacházejících se uvnitř posuvných oken pro každou z obou čár jízdního pruhu zvlášť. [31]

Body nalezené popsaným postupem jsou proloženy křivkou, tedy polynomem druhého řádu s předpisem v rovnici

$$f(y) = ay^2 + by + c = 0\ . \tag{3.1}$$

Polynom druhého řádu by měl postačovat k aproximaci většiny tvarů silnic. Funkce je zvolena jako závislost na souřadnici *y*, jelikož čára na silnici má v obraze vertikální směr. [31]

Nakonec jsou tyto dvě křivky vykresleny provedením jejich inverzní projektivní transformace, pomocí které budou křivky odpovídat horizontálnímu dopravnímu značení jízdních pruhů na vstupním obrazu. [31][32]

4. PRAKTICKÁ IMPLEMENTACE

Tato kapitola se zabývá praktickou realizací a implementací návrhu systému zpracování dat pro segmentaci vozovky a detekcí jízdních pruhů popsaného výše. Realizace byla provedena v programovacím jazyce Python 3.8 [33] a zdrojový kód byl napsán v programovacím prostředí PyCharm 2021.1.1 (Professional Edition) [34]. Při realizaci se vycházelo z návrhu popsaného v kapitole 3, kdy byly postupně implementovány jednotlivé kroky zpracování dat. Zdrojové kódy všech navržených a implementovaných řešení jsou uvedeny v příloze A (A.1) umístěné na DVD.

4.1 Sémantická segmentace vozovky

V této kapitole je objasněn způsob implementace sémantické segmentace vozovky pomocí metody rozvodí. Dále kapitola pojednává o tom, jak byl návrh při realizaci upraven a vylepšen ve srovnání s návrhem teoretickým, který je uveden v kapitole 3.2.

4.1.1 Předzpracování obrazu

První část implementovaného algoritmu zajišťuje předzpracování dat pro odstranění šumu a zjednodušení obrazu. To je potřeba provést proto, že ve vstupním obraze z kamery je poměrně velké množství šumu. Pro jeho odstranění a zjednodušení obrazu bylo implementováno, a následně otestováno několik různých postupů. Jednotlivé postupy jsou vysvětleny v následujících odstavcích.

První implementovaný typ zjednodušení a filtrace šumu v obraze je využití šedotónových morfologických transformací podle teoretického návrhu uvedeného v předchozí části práce (kapitola 3.2). Při testování se nejvíce osvědčila kombinace morfologických operací eroze, rekonstrukce a dilatace. Jako strukturní element byl zvolen tvar disku o poloměru 7 pixelů. Toto zjednodušení obrazu bylo nazváno jako „erdr“. K provedení morfologické eroze a dilatace byla použita knihovna OpenCV a pro morfologickou rekonstrukci byla, jakožto nejvýhodnější, zvolena knihovna scikit-image, která má funkci *reconstruction()* implementovánu pomocí „downhill filtering“ algoritmu popsaného v [35][24]. Mimo to byla morfologická rekonstrukce implementována vlastním algoritmem vycházejícím z rovnic (1.28) a (1.29), a to za pomoci knihovny OpenCV. Vlastní implementace však nedosahuje takové rychlosti, jako implementace v knihovně scikit-image viz tabulka 4.1. Srovnání rychlosti vykonání jednotlivých morfologických transformací je možné nalézt v tabulce 4.1. Na základě této tabulky byla vybrána knihovna, která bude pro danou operaci použita. Po implementaci a testování na reprezentativním datasetu však nebyly výsledky uspokojivé, a proto bylo navíc doimplementováno převedení zjednodušeného šedotónového obrazu do jeho gradientu. Převod byl realizován pomocí Sobelova horizontálního a vertikálního filtru uvedeného v rovnici (1.12) za použití funkce *Sobel()* knihovny OpenCV a OpenCL [22]. Algoritmus

pro tuto detekci spočívá v postupné aplikaci nejprve Sobelova horizontálního filtru a následně vertikálního filtru. Výstupními hodnotami z aplikace filtrů jsou pole datového typu 64-bitový float, ve kterém jsou kladné i záporné hrany. Vstupní obraz do rozvodí však musí být pole 8-bitového *unsigned integer* (celočíselný datový typ bez znaménka). Z toho důvodu byla nejdříve u obou filtrů vypočítána absolutní hodnota, a poté provedena normalizace do rozsahu 0-255 pomocí rovnice

$$\mathrm{N(i,j)} = 255 \cdot \frac{|S(i,j)|}{max(|S|)}, \tag{4.1}$$

kde S je gradientní obraz získaný pomocí Sobelovy konvoluční masky a N je získaný obraz normalizovaný do rozsahu hodnot 0-255. Takto normalizované obrazy jsou převedeny do 8-bitového celočíselného datového typu a je mezi nimi provedena bitová disjunkce. Následně je ještě realizováno morfologické uzavření gradientního obrazu strukturním elementem tvaru disku o poloměru 5 pixelů, které zajistí zacelení malých mezer v gradientním obrazu.

Další možností zjednodušení a filtrace obrazu před segmentací je aplikace takzvaného „occo“ algoritmu. Ten využívá morfologického otevření a uzavření a je popsán následující posloupností kroků: [36]

- O = OPEN(I, se)
- OC = CLOSE(O, se)
- C = CLOSE(I, de)
- CO = OPEN(C, se)
- RES = $S(i,j) = \frac{1}{2}[OC(i,j) + CO(i,j)]$.

Kde *I* je vstupní obraz, *se* je strukturní elementem tvaru disku a *RES* je výsledný filtrovaný obraz. Výsledek takto zjednodušeného barevného obrazu, jenž připomíná mozaiku, dobře filtruje šum a nerozmazává hrany, což je vidět na obrázku 4.1. Implementace byla provedena pomocí knihovny OpenCV a její funkce *morphologyEx()* [22]. Jako strukturní element byl na základě výsledků při testování vybrán disk o poloměru 3 pixely.

Posledními testovanými zjednodušeními obrazu jsou filtrace pomocí různých filtrů. Prvním filtrem je mediánový filtr implementovaný pomocí knihovny OpenCV a funkce *medianBlur()* [22]. Velikost okolí byla zvolena jako čtverec o hraně 9 pixelů. Dalšími filtry jsou bilaterální filtr [9] a mean shift filtr [5]. Oba filtry jsou implementovány jako funkce knihovny OpenCV, a to *bilateralFilter()* a *pyrMeanShiftFiltering()* [22]. Parametry těchto filtrů byly laděny také při testování.

Tabulka 4.1 Srovnání průměrné doby vykonání využitých operací nad obrazem

	Knihovna / způsob implementace			
Operace	**OpenCV [ms]**	**OpenCV s GPU [ms]**	**scikit-image [ms]**	**Vlastní implementace [ms]**
Eroze	1,546	5,967	166,0	-
Dilatace	1,524	5,698	165,0	-
Rekonstrukce	-	-	285,6	500,4
Otevření	3,022	7,016	331,21	-
Uzavření	2,995	7,052	331,16	-
Watershed	15,71	11,88	382,3	-
Sobelův hranový detektor V+H 3x3	7,656	1,478	32,44	-
Gaussův vyhlazovací filtr 5x5	1,114	5,234	62,57	-
Mediánový filtr 9x9	102,7	109,4	631,7	-
Bilaterální filtr	245,3	5,265	5188	-
Mean shift filtr	211,5	218,2	218,2	-

Data v tabulce 4.1 byla vytvořena pomocí programovacího jazyka Python verze 3.8, knihoven OpenCV verze 4.5.1.48 a scikit-image verze 0.18.1. Algoritmus běžel na procesoru Intel Core i58250U a grafickém adaptéru AMD Radeon RX 560. Hodnoty uvedené v tabulce 4.1 jsou aritmetickým průměrem časů, jež byly získány při měření na 30 náhodně vybraných obrazech silnic z reprezentativního datasetu v rozlišení 1280x720 pixelů.

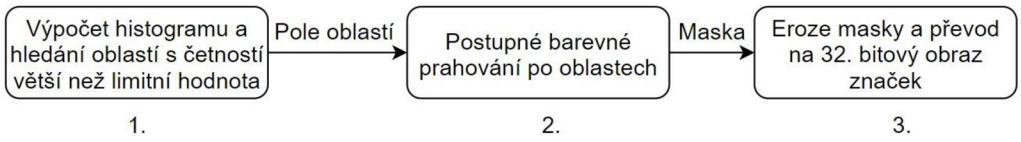
Obrázek 4.1 Obraz filtrovaný pomocí „occo“ algoritmu

4.1.2 Hledání značek pozadí a silnice v obraze

Pro segmentaci je třeba nalézt značky, které budou označovat pozadí a silnici. Návrh v teoretické části práce (kapitola 3.2) obsahuje informaci, že značky pro silnici a pozadí budou statické, určeny heuristicky. V horní časti obrazu měla být umístěna značka pro

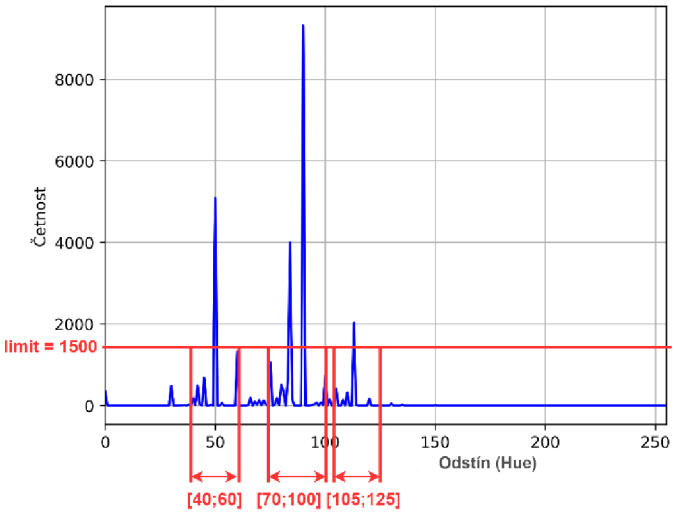
pozadí a uprostřed spodní části obrazu značka silnice. Při testování tohoto postupu však bylo zjištěno, že výsledky segmentace metodou rozvodí jsou nevyhovující a velmi často je za silnici označeno i její okolí, jako jsou lesy, louky, pole nebo budovy. Tento problém lze vyřešit tím, že bude v obraze nalezeno více značek odpovídajících pozadí scény.

Proto byl navržen a implementován algoritmus pro adaptivní hledání značek pozadí v obraze za pomocí histogramu. Vstupem algoritmu je výřez vstupního obrazu v barevném modelu HSV v místě statické značky silnice. Výstupem je pole udávající, jaké rozsahy hodnot H, S a V tvoří vstupní obraz.



Obrázek 4.2 Diagram algoritmu adaptivního prahování

První část adaptivního hledání značek spočívá v postupném výpočtu histogramu složek H, S a V vstupního obrazu. V těchto třech histogramech jsou pak hledány oblasti, ve kterých je četnost hodnot jednotlivých složek H, S a V vyšší než nastavená limitní hodnota. Nalezené rozsahy jsou nakonec uloženy do pole hodnot. Vstupními parametry této funkce jsou jednak již zmíněná limitní hodnota četnosti, a dále parametr, kterým je možné nastavit šířku okolí, o niž budou nalezené oblasti rozšířeny. Zvětšení šířky oblastí hraje důležitou roli zvláště proto, aby bylo dosaženo co největší obecnosti při hledání značek pozadí. Vizualizace funkce algoritmu pro hledání rozsahů odstínů silnice je zobrazena na obrázku 4.3. Tato část algoritmu je implementovaná jako funkce v jazyce Python a její definice je uvedena v příloze A.4.



Obrázek 4.3 Ukázka principu hledání rozsahu odstínů odpovídajících silnici pomocí implementovaného algoritmu

Druhou částí tohoto algoritmu je funkce pro postupné prahování po jednotlivých složkách barevného modelu. Prahování je implementováno pomocí vzorce

$$T(i,j) = \begin{cases} 255\, pro\ \ l \leq I(i,j) \leq u \\ 0\ \ pro\ \ I(i,j) \leq l \vee I(i,j) \geq u \end{cases}, \tag{4.2}$$

kde I je vstupní obraz, l je spodní limit, u je horní limit a T je výstupní naprahovaný obraz. Prahování dle rovnice (4.2) se realizuje postupně pro všechny vstupní rozsahy. Poté je provedena disjunkce všech naprahovaných obrazů dle vzorce

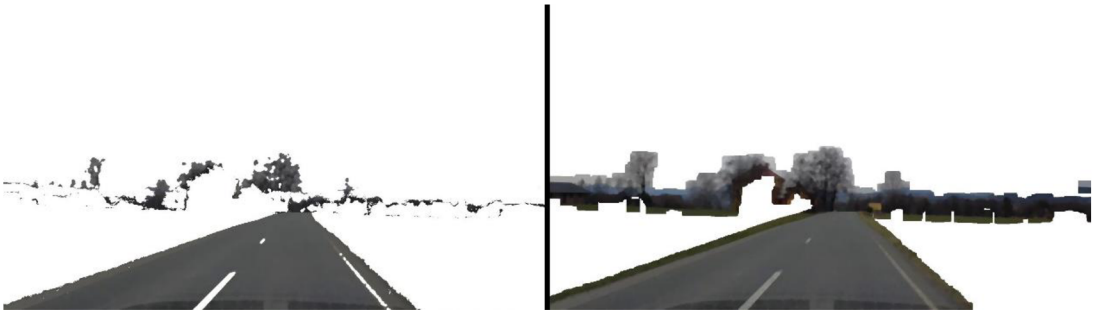
$$H(i,j) = T_1(i,j) \vee T_2(i,j) \vee \ldots \vee T_N(i,j) \quad N = 1,2,3 \ldots n, \tag{4.3}$$

kde T je vstupní naprahovaný obraz, n je počet vstupních naprahovaných obrazů, nebo také počet rozsahů, a H je v tomto případě binární výstupní maska pro složku „hue“ modelu HSV. Daný postup se dále aplikuje i na zbylé složky barevného modelu (S a V). Posledním krokem je provedení konjunkce zmíněných tří binárních masek

$$M(i,j) = H(i,j) \wedge S(i,j) \wedge V(i,j)\,, \tag{4.4}$$

kde H, S a V jsou masky jednotlivých barevných složek a M je konečná výstupní maska. Výsledná maska označuje části obrazu, ve kterých by měl obraz odpovídat barvě silnice. Algoritmus funkce postupného prahování je uveden v příloze A.5 (pojmenování proměnných v rovnicích výše a ve funkci v příloze se liší).

Ve třetí části (příloha A.6) je potřeba konečnou výstupní masku M z algoritmu v příloze A.5 invertovat, aby místo silnice bylo maskou označeno pozadí. Posledním krokem je pak eroze masky, která je podstatná z toho důvodu, aby byl odstraněn šum a nebylo v obraze velké množství malých značek. Eroze je provedena za pomoci strukturního elementu tvaru disku o poloměru 30 pixelů. Výsledek adaptivního prahování jak bez využití morfologické eroze, tak s jejím využitím, je zobrazen na obrázku 4.4.



Obrázek 4.4 Výsledky algoritmu adaptivního prahování bez morfologické eroze (vlevo) a po erozi strukturním elementem velikosti 30x30 (vpravo)

4.1.3 Segmentace za pomoci rozvodí

Implementace segmentace metodou rozvodí se značkami byla provedena s pomocí funkce *watershed()* knihovny OpenCV za použití OpenCL akcelerace na grafické kartě [22]. Výběr použitých nástrojů byl uskutečněn na základě srovnání rychlostí algoritmů

v tabulce 4.1, ze které je zřejmé, že průměrná doba vykonání rozvodí na testovacím hardwaru je 11,88 ms. Vstupem pro metodu rozvodí je zjednodušený, filtrovaný nebo gradientní obraz a 32-bitový obraz značek. Ve zmíněném obrazu značek je každá značka (region) reprezentována hodnotou 1, 2, 3, atd. Hodnota 0 označuje části, které nejsou přiřazeny žádné značce. Výsledkem rozvodí je segmentovaný obraz, který je rozdělený do stejného počtu oblastí, jako je počet značek. Každá z oblastí je pak označena stejnou hodnotou jako značka, ze které byla vytvořena. Hodnota -1 v segmentovaném obrazu označuje hranice mezi jednotlivými oblastmi.

Pro účely vizualizace a dalšího zpracování je z výsledku segmentace extrahována oblast silnice na základě znalosti polohy její statické značky. Výstupem implementovaného algoritmu sémantické segmentace rozvodím je pak binární obraz, kde hodnota 0 odpovídá pozadí a hodnota 255 silnici. Vizualizace výsledků segmentace vozovky s filtrací za pomocí „occo“ algoritmu je na obrázku 4.5 a v příloze E (A.10).

Obrázek 4.5 Ukázka výsledků segmentace vozovky s „occo“ filtrací obrazu

4.2 Detekce jízdních pruhů

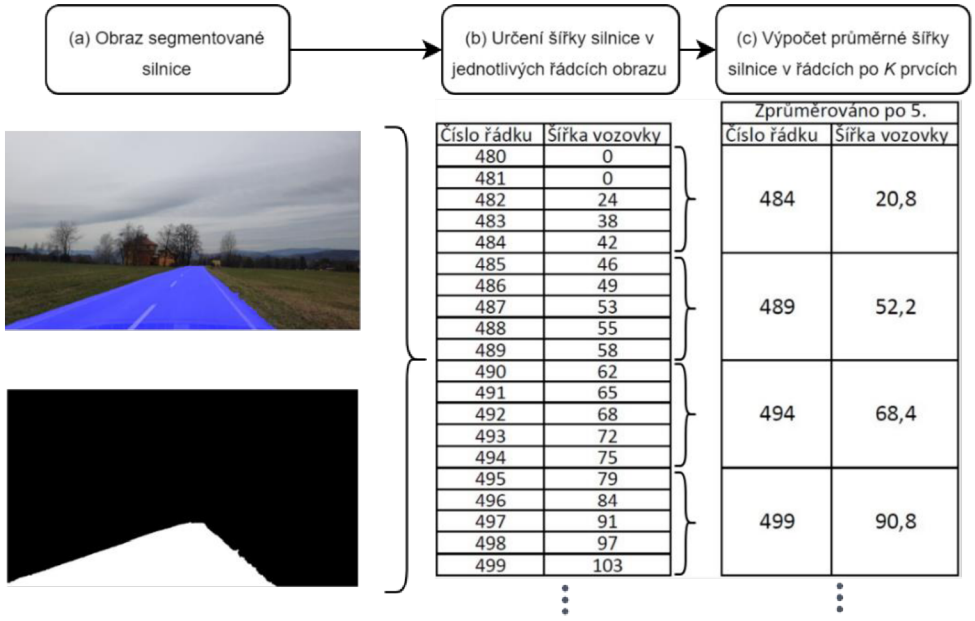
Implementace detekce jízdních pruhů navazuje na sémantickou segmentaci vozovky popsanou výše. V této kapitole je rozebrána implementace projektivní transformace, prahování obrazu pro získaní binárního obrazu čar, a také hledání čar jízdních pruhů pomocí posuvných oken.

4.2.1 Výběr oblasti zájmu a projektivní transformace

Po segmentaci silnice a pozadí výše popsaným postupem je nutné určit oblast zájmu (silnici), ve které se budou detekovat jízdní pruhy, a poté silnici projektivně transformovat. V teoretickém návrhu postupu segmentace jízdních pruhů v kapitole 3.3

je uvedeno, že projektivní transformace obrazu vozovky bude provedena tak, aby byla vidět pohledem shora. Dále bylo předpokládáno, že na základě testování v reprezentativním datasetu se určí statické body v obraze pro zjištění matice projektivní transformace $\boldsymbol{H}$. Z výsledků zmíněného postupu však vyplynulo, že fixní oblast pro projektivní transformaci je vyhovující pouze na zcela rovných silnicích. Pokud se ale vozovka před vozidlem stoupá vzhůru, nebo se svažuje dolů, není již výsledek projektivní transformace správný. Při pohledu shora se pak objevuje kromě silnice často i oblast za silnicí, například obloha, stromy atd., což není žádoucí.

Z důvodů vysvětlených v předchozím odstavci byl navržen algoritmus pro adaptivní hledání bodů k výpočtu matice projektivní transformace. Tento algoritmus využívá skutečnosti, že čím více se v obraze silnice od vozidla vzdaluje, tím více se zužuje. V dolní části obrazu je silnice nejširší a postupem od spodního okraje obrazu nahoru je silnice stále užší. Na základě postupného zužování je pak vypočtena matice projektivní transformace. Schéma algoritmu je znázorněno na obrázcích 4.6 a 4.7.



Obrázek 4.6 Schéma algoritmu pro adaptivní hledání bodů k výpočtu matice projektivní transformace (první část).

Vstupem daného algoritmu je výsledek sémantické segmentace, jak lze vidět na obrázku 4.6 (a) a výstupem je matice projektivní transformace. První krok algoritmu spočívá v tom, že je v každém řádku matice obrazu segmentované silnice zjištěn a uložen do pole počet pixelů, které odpovídají silnici (počet prvků tohoto pole odpovídá počtu řádků obrazu) viz obrázek 4.6 (b). Poté se pole, v němž je uložena šířka silnice, rozdělí na části po K prvcích (v implementaci je použitá hodnota $K = 30$) a vypočítá se průměrná šířka silnice v jednotlivých částech. Výsledek je uložen do nového pole viz obrázek 4.6 (c). Průměrování je nezbytný krok proto, aby zvolený postup určení matice projektivní transformace nebyl příliš náchylný na chyby ve výsledku segmentace

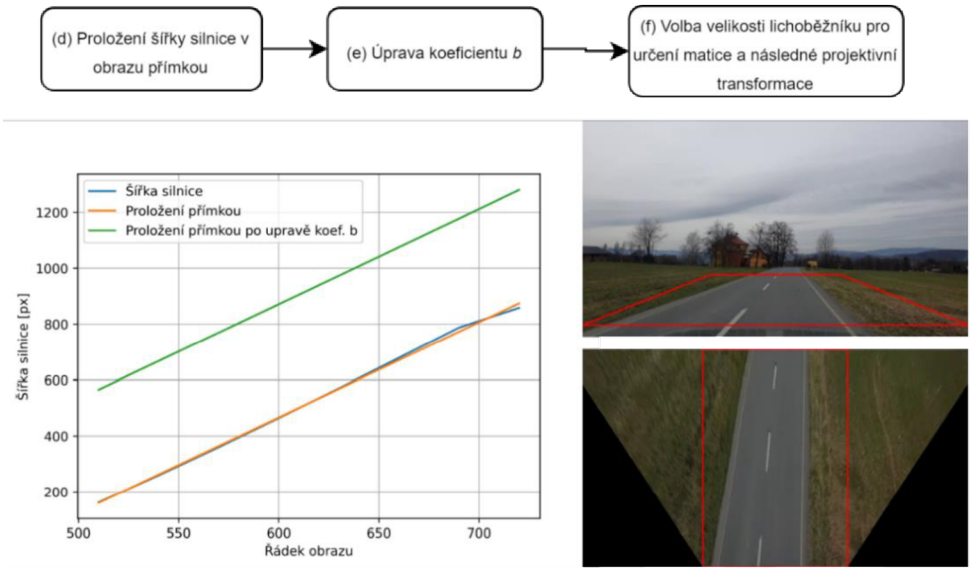
vozovky. Z pole s průměrnou šířkou vozovky v částech obrazu je možno určit dvě důležité vlastnosti silnice v obraze. První vlastností je pozice, kde silnice končí, a druhou, jak rychle se silnice zužuje. Pro zjištění rychlosti zužování se průměrná šířka silnice proloží přímkou pomocí metody nejmenších čtverců [37] implementované ve funkci *polyfit()* v knihovně NumPy [26], a tím jsou získány koeficienty parametrické rovnice přímky a a b. Parametrická rovnice přímky je definována jako

$$y = ax + b\,, \tag{4.5}$$

kde a vyjadřuje rychlost zužovaní vzdalující se silnice v obraze a b její šířku. Ukázka proložení šířky silnice přímkou je znázorněna pomocí grafu na obrázku 4.7 (d), kde modrá křivka označuje vstupní šířku silnice a oranžová je proložení přímkou. Koeficient b následně upravíme podle následujícího vztahu

$$b = w - ha\,, \tag{4.6}$$

kde w je počet sloupců obrazu, h je počet řádků obrazu a a je koeficient získaný při proložení metodou nejmenších čtverců. Úprava koeficientu se provádí proto, aby po dosazení počtu řádků obrazu za x v rovnici (4.5), byl výsledkem y počet sloupců obrazu. Tato modifikace koeficientu b je znázorněna jako zelená přímka v grafu na obrázku 4.7 (e), si lze všimnout, že je šířka silnice zvětšena a posunuta v grafu nahoru. Po dosazení čísla řádku ve spodní části obrazu do parametrické rovnice přímky (4.5) upravené pomocí vztahu (4.6) bude výsledkem číslo odpovídající šířce obrazu viz šířka spodní hrany lichoběžníku na obrázku 4.7 (f) nahoře. Doplněním čísla řádku obrazu, kde silnice končí, a zároveň je dosti vzdálená od vozidla, do rovnic získáme odpovídající šířku v daném místě viz horní hrana lichoběžníku na obrázku 4.7 (f) nahoře.



Obrázek 4.7 Schéma algoritmu pro adaptivní hledání bodů k výpočtu matice projektivní transformace (druhá část).

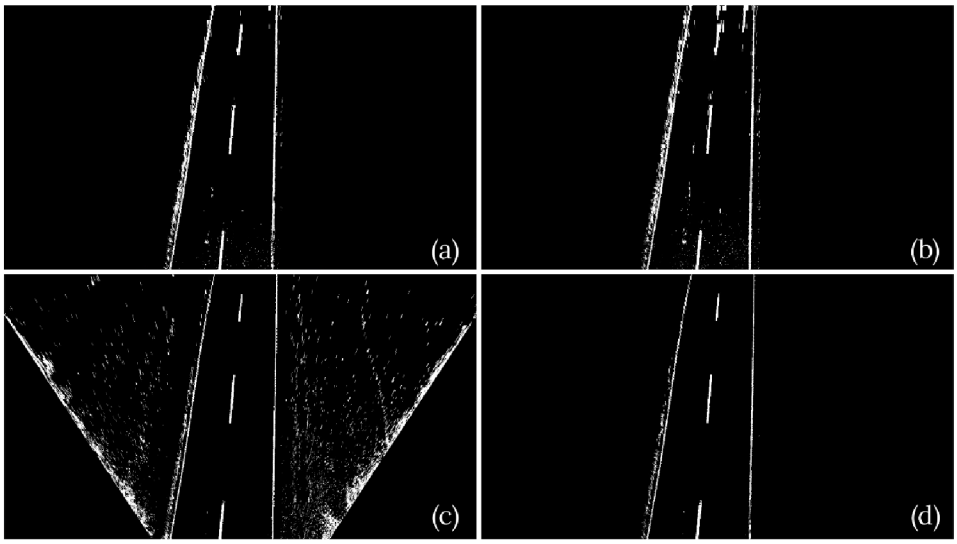
Číslo řádku konce silnice pro dosazení do rovnice (4.5) upravené pomocí vztahu (4.6) lze určit tak, že se v obraze vyhledá řádek, kde je průměrná šířka silnice užší než N pixelů (v implementaci $N = 60$). Na základě získaných hodnot se poté vykreslí lichoběžník viz obrázek 4.7 (f) nahoře. Ze souřadnic vrcholů lichoběžníku je pomocí knihovny OpenCV a funkce *getPerspectiveTransform()* získána matice projektivní transformace pro pohled shora. Vstupem funkce *getPerspectiveTransform()* jsou souřadnice vrcholů čtyřúhelníku ve vstupním obraze a výstupní souřadnice vrcholů čtyřúhelníku, na které mají být vstupní souřadnice transformovány [22]. V tomto případě, kdy je žádoucí transformovat silnici na pohled shora, je využit ve vstupním obraze lichoběžník viz obrázek 4.7 (f) nahoře, který se transformuje na obdélník viz obrázek 4.7 (f) dole. Výstupem je pak homogenní matice projektivní transformace.

Po otestování popsaného algoritmu na reprezentativním datasetu bylo ještě implementováno jedno vylepšení. To spočívá v ukládání do fronty 100 posledních souřadnic místa, kde silnice končí, a 100 posledních šířek silnice ve stejném místě. Vstupem funkce *getPerspectiveTransform()* je pak průměr z uložených dat ve frontě, což zásadně omezuje špatný výběr lichoběžníku pro projektivní transformaci způsobený například nepřesnou segmentací vozovky nebo atypickým tvarem vozovky v určitém místě.

Posledním krokem již je jen projektivní transformace obrazu pomocí knihovny OpenCV a její funkce warpPerspective*()* [22], jejímž vstupem je obraz, který chceme transformovat a matice projektivní transformace. Výstupem je projektivně transformovaný obraz. Ukázka zdrojového kódu funkce pro získání matice projektivní transformace je uvedena v přílohách A.7 a A.8.

4.2.2 Prahování obrazu pro detekci jízdních pruhů

Prahování obrazu pro detekci jízdních pruhů bylo implementováno podle návrhu v kapitole 3.3. Na obraz transformovaný na pohled shora bylo tedy nejdříve aplikováno prahování s využitím barevných modelů HSV a HLS viz obrázek 4.8 (a) a (b). Parametry maximálních i minimálních hodnot odstínu, sytosti a hodnoty jasu či světlosti byly určeny fixně při testování. Pouze minimální hodnota jasu u barevného modelu HSV a minimální hodnota světlosti u modelu HLS jsou určeny adaptivně pomocí algoritmu. Důvod spočívá v tom, že pokud by minimální hodnoty byly konstantní, jak bylo navrženo v teoretické části práce, při změně světelných podmínek scény by prahování selhalo. Adaptivní algoritmus byl navržen a implementován tak, že se nejprve vypočítá histogram požadované barevné složky transformovaného obrazu silnice. Dále se z histogramu určí maximální světlost, v případě barevného modelu HLS, či jas, v případě modelu HSV, s požadovanou minimální četností. Odečtením konstanty (v implementaci hodnota 35) od určené hodnoty je zvolen práh.



Obrázek 4.8 Ukázka výsledku prahování obrazu pomocí barevných modelů HSV (a), HLS (b), adaptivního prahování (c) a jejich kombinace (d)

Dalším krokem bylo adaptivní šedotónové prahování, které se aplikovalo na transformovaný šedotónový obraz pomocí funkce *adaptiveThreshold()* knihovny OpenCV. Ta provádí prahování prahem *T(x, y),* který je stanoven jako průměrná hodnota pixelů ve čtvercovém okolí velikosti *N* x *N* mínus *C* [22]. Parametry funkce prahování *N* a *C* byly určeny jako konstanty na základě testování na videích z reprezentativního datasetu. Ukázka výsledku adaptivního šedotonového prahování je na obrázku 4.8 (c).

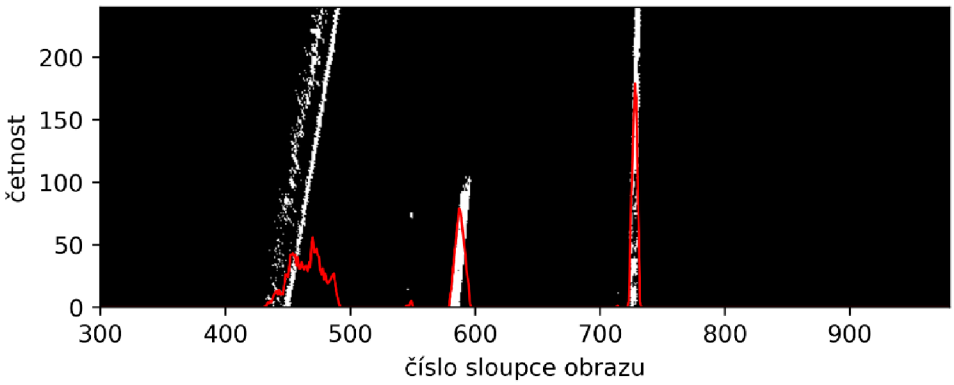
Prahování gradientního obrazu dle teoretického návrhu bylo po otestování na datasetu odebráno, jelikož výsledkem byly pouze obrysy jízdních pruhů. Navíc se zde vyskytovalo mnoho prvků, které čárám jízdních pruhů na silnici nenáleží, kupříkladu krajnice vozovky, svodidla a další objekty, které způsobovaly nesprávnou detekci pruhů.

Mezi binárními obrazy získanými pomocí HSV, HLS a adaptivního šedotónového prahování byla nakonec provedena logická operace *and*, takže výsledný binární obraz jízdních pruhů obsahuje pouze ty pixely, které se vyskytují ve všech třech vstupních obrazech. Výsledek lze vidět na obrázku 4.8 (d). Oproti teoretickému návrhu tedy došlo v posledním kroku ke zjednodušení postupu při získání binárního obrazu s jízdními pruhy.

4.2.3 Detekce jízdních pruhů pomocí posuvných oken

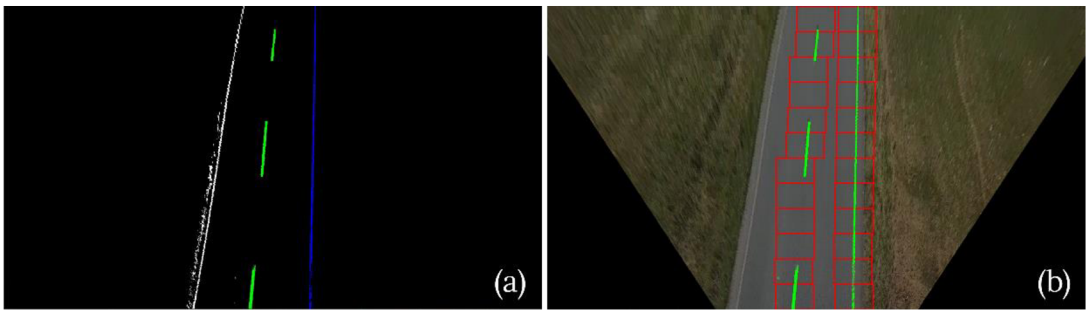
Zjištění pozice začátku jízdních pruhů bylo implementováno stejnou metodou, jaká je uvedena v teoretickém návrhu v kapitole 3.3. Nejprve byl spočítán počet pixelů, které odpovídají jízdním pruhům ve sloupcích spodní třetiny binárního naprahovaného obrazu. Tato operace je znázorněna na obrázku 4.9, kde červená křivka představuje četnost pixelů silnice v jednotlivých sloupcích obrazu. Poté byla vyhledána dvě maxima, jedno napravo od středu obrazu a druhé nalevo od jeho středu. Tato dvě místa s největším počtem pixelů odpovídají počátku pravé a levé čáry jízdního pruhu. V implementaci bylo nutné ještě

ošetřit možnou situaci, že na silnici jízdní pruhy nejsou, případně je čára jen uprostřed silnice, nebo pouze u krajnice. Z tohoto důvodu bylo navíc implementováno rozhodování, zda v obraze pruhy jsou, či ne. Pokud tedy nalezené maximum obsahovalo méně bodů, než je stanovená minimální hranice, byl učiněn závěr, že čára jízdního pruhu nebyla nalezena.



Obrázek 4.9 Ukázka nalezení počátku čar jízdních pruhů na silnici

Také implementace algoritmu posuvných oken byla provedena téměř stejně, a to jako funkce dle teoretického návrhu v kapitole 3.3. Vstupem této funkce je výška a šířka posuvných oken, nalezená počáteční pozice čáry a naprahovaný obraz silnice. Okna jsou postupně posouvána pomocí cyklu *for* od spodní části obrazu nahoru. Narozdíl od teoretického návrhu se však souřadnice pixelů odpovídajících čáře ukládají do pole průběžně s posouváním oken. Navíc se kontroluje, zda je nalezený počet pixelů odpovídajících čáře v posuvném okně větší než daná konstanta (v implementaci je to 50 pixelů). Pokud ano, je okno vychýleno doprava či doleva na průměrnou horizontální pozici všech nalezených pixelů v okně. Tato kontrola se provádí proto, že v místě, kde není na vozovce čára, by šum mohl okno prudce vychýlit. Ukázka nalezených pixelů, které odpovídají čáře je na obrázku 4.10 (a), kde modrá barva reprezentuje pravou čáru a zelená barva levou čáru pruhu. Na obrázku 4.10 (b), který znázorňuje transformovanou silnici, jsou čáry jízdních pruhů znázorněny zelenou barvou, a posuvná okna jsou vykreslena červenou barvou.



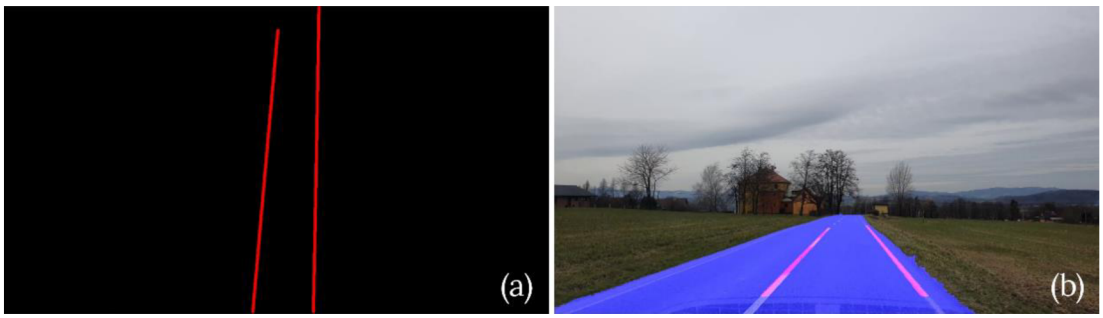
Obrázek 4.10 Ukázka detekce jízdních pruhů pomocí posuvných oken

V praxi byla také testována varianta, kdy místo projektivní transformace silnice bylo použito postupné zmenšování posuvných oken na základě toho, jak se silnice do dálky zužuje. Zde se však objevil problém s nalezením počátku čar jízdních pruhů, jelikož čáry ve spodní části obrazu nejsou svislé, nýbrž šikmé. V případě projektivně transformované vozovky na pohled shora se daný problém neobjevuje a čáry označující pruhy jsou téměř svislé.

Algoritmus proložení nalezených čar na silnici polynomem druhého řádu odpovídá popisu v kapitole 3.3. Proložení bylo implementováno pomocí knihovny NumPy a její funkce *polyfit()* využívající metodu nejmenších čtverců [26]. Výsledek proložení je vizualizován na obrázku 4.11 (a).

4.2.4 Vizualizace

Pro účely vizualizace detekovaných čar jízdních pruhů a výsledku sémantické segmentace vozovky byl obraz s jízdními pruhy viz obrázek 4.11 (a) zpětně projektivně transformován pomocí knihovny OpenCV do původního obrazu. To je zobrazeno na obrázku 4.11 (b). Konečné zobrazení segmentované vozovky je pak vizualizováno pomocí modré barvy a čáry jízdních pruhů pomocí barvy červené, jak je vidět na obrázku 4.11 (b).



Obrázek 4.11 Proložené jízdní pruhy (a) a výsledná vizualizace segmentace včetně jízdních pruhů (b)

4.3 Struktura zdrojových kódů programu

Zdrojové kódy systému zpracování obrazových dat jsou uloženy na DVD (příloha A.1). Zmíněná příloha obsahuje také popis struktury složek a souborů. Zdrojové kódy programu jsou rozděleny do dvou složek. První z nich je *složka modules*, v níž jsou uloženy moduly vytvořené pro zpracování obrazu, pro detekci jízdních pruhů a segmentaci. Modul *preprocess_before_segmentation.py* obsahuje funkce pro předzpracování obrazu před segmentací, *segmentation_markers.py* zajišťuje hledání značek pro segmentaci, *segmentation_watershead.py* provádí výslednou sémantickou segmentaci vozovky v obraze, *perspective_transform.py* obsahuje funkce pro adaptivnímu určení matice projektivní transformace, *lanes_thresholding.py* slouží

k prahování transformovaného obrazu jízdních pruhů a *lanes_sliding_window.py* zajišťuje hledání počátku jízdních pruhů a hledání jízdních pruhů pomocí posuvných oken. Ve druhé složce, pojmenované *segmentation_methods,* jsou pak funkce s implementací segmentace vozovky za pomoci různých předzpracování obrazu (occo, erdr, mean-shift a bilaterálního filtrování).

V kořenu složky „program“ na DVD (příloha A.1) jsou pak uloženy čtyři skripty určené pro spuštění implementovaných řešení. Jedná se o skripty určené ke spuštění sémantické segmentace a detekce jízdních pruhů. Pro zpracování obrazu z videa je určena funkce *viedo_segmentation_with_lane_detection.py* a pro živý kamerový vstup skript *live_video_segmentation_with_lane_detection.py*. Dále se zde nachází skript pro testování rychlosti a přesnosti sémantické segmentace nazvaný *segmentatin_accuracy_test.py,* a také skript pro testování segmentace s možností výběru metody předzpracování obrazu *segmentation_methods_test.py*.

Pro spuštění skriptu je nutné mít nainstalovaný programovací jazyk Python 3.8 a následující knihovny:

- opencv-python 4.5.2.52
- numpy 1.20.3
- matplotlib 3.4.2
- scikit-image 0.18.1

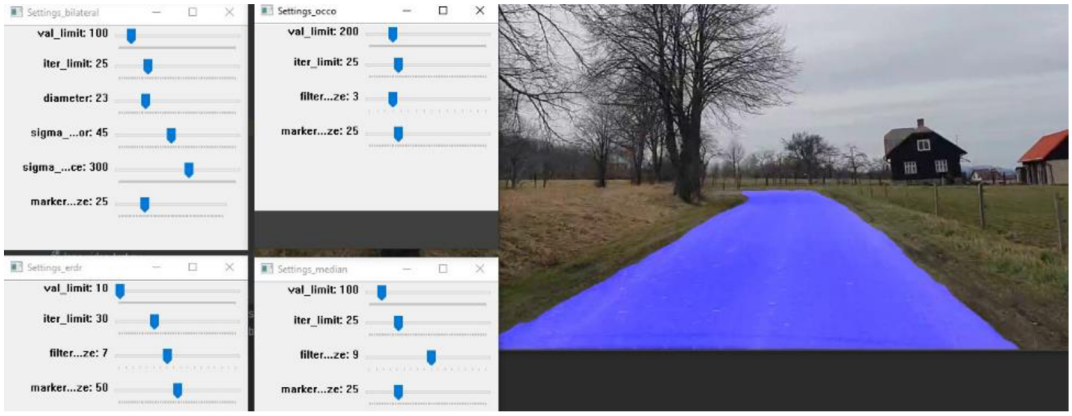
Dále je nezbytné mít správně nastavenou cestu k zdrojovému videu v rozlišení 1280x720 pixelů ve spouštěném skriptu na následujícím řádku:

```
patch = '../dataset/videos/dataset_video_1.mp4'
```

5. TESTOVÁNÍ A SROVNÁNÍ IMPLEMENTOVANÝCH ŘEŠENÍ

Pro implementované algoritmy bylo potřeba navrhnout a implementovat systém testování, který ověří, zda jsou algoritmy implementovány správné. Výsledkem testování je stanovení přesnosti, jaké se podařilo dosáhnout. Díky tomu lze určit, která kombinace implementovaných metod, popsaných v kapitole 4, je pro konečnou aplikaci a implementaci nejvhodnější.

Pro získaní co nejlepších výsledků a pro testování byla implementována také funkcionalita pro ladění parametrů filtrů předzpracování obrazu, segmentace a detekce jízdních pruhů. Tento krok byl nezbytný, protože v implementovaném řešení je mnoho parametrů, kterými lze program ladit. Efektivním nástrojem, který byl v praxi použit, jsou posuvníky pro výběr hodnot knihovny OpenCV [22]. Ukázku ladících posuvníku je možné vidět na obrázku 5.1.



Obrázek 5.1 Ukázka posuvníků pro ladění parametrů segmentace a detekce čar jízdních pruhů

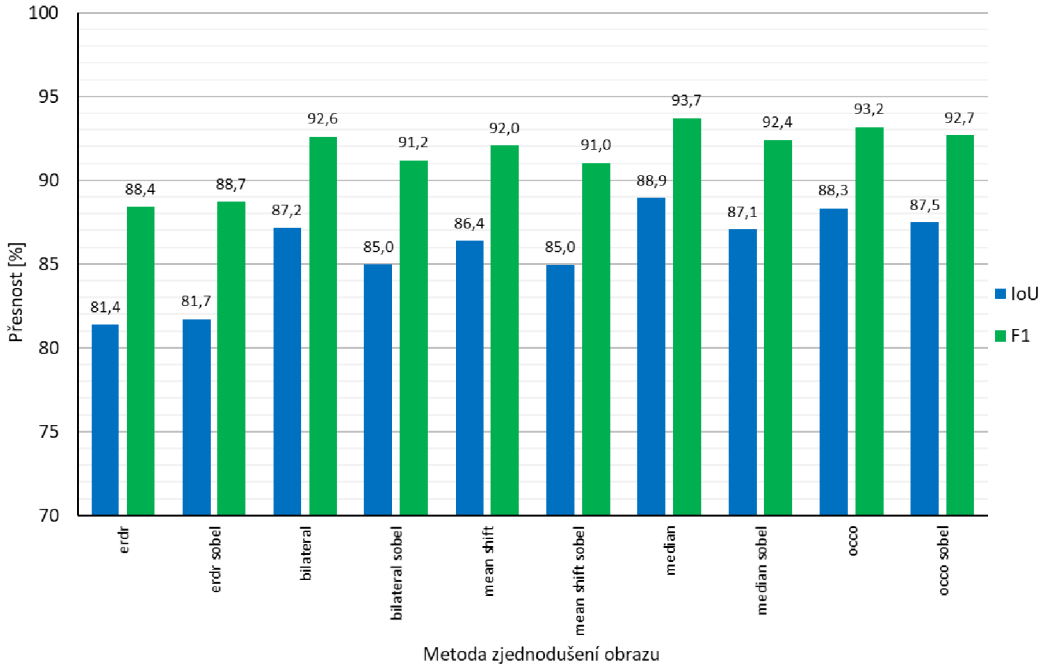
Mnoho implementovaných funkcí obsahuje také parametr „visualize“, který v případě nastavení na hodnotu „True“ způsobí, že se na obrazovce zobrazí okna s vizualizací mezivýsledků a výsledků dané funkce. Některé funkce po nastavení tohoto parametru na „True“ generují také grafy pomocí knihovny Matplotlib určené pro snadné vytváření grafů [27]. Pro korektní zobrazení grafů knihovny Matplotlib při zpracování videa je však potřeba spouštět program pomocí vývojového prostředí *PyCharm Professional Edition* [34] se zapnutým „Scientific Mode“. Je potřeba zdůraznit, že toto platí pouze u funkcí, které vykreslují grafy pomocí knihovny Matplotlib, a mají nastavený parametr „visualize“ na hodnotu „True“.

5.1 Testování sémantické segmentace vozovky

Tato kapitola se zabývá vyhodnocením přesnosti sémantické segmentace vozovky na vytvořeném reprezentativním datasetu, a také srovnáním rychlosti vykonání všech implementovaných řešení.

5.1.1 Testování přesnosti

Pro otestování přesnosti sémantické segmentace vozovky byly využito 171 obrázků silnic s ručně označenou vozovkou pomocí programu Matlab Image Labeler, jak je popsáno v kapitole 3.1. Přesnost byla testována pomocí hojně využívaných metrik IoU a F1 vysvětlených v kapitole 1.10. Pro účely testování byl navržen algoritmus, který postupně projde všechny implementované metody předzpracování obrazu pro segmentaci pomocí rozvodí, a následně vypočítá přesnost sémantické segmentace dle metrik IoU a F1. Testováno bylo celkem 5 metod zjednodušení, a to „erdr“ (morfologická eroze následována rekonstrukcí, dilatací a znovu rekonstrukcí), bilaterální filtr, mean shift filtr, mediánový filtr a „occo“ zjednodušení. Další testovanou záležitosti bylo, jaký vliv na výsledky má převedení filtrovaného obrazu do jeho gradientu před segmentací.



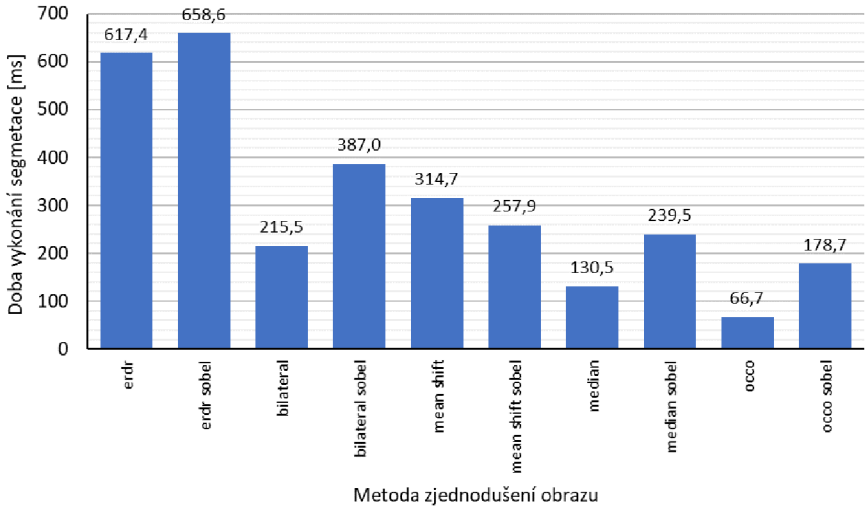
Obrázek 5.2 Srovnání přesnosti výsledků sémantické segmentace vozovky pomocí metrik IoU a F1 pro různé metody předzpracování obrazu

Výsledky testování všech metod předzpracování obrazu pro segmentaci jsou přehledně zobrazeny v grafu na obrázku 5.2. Z těchto výsledků je patrné, že nejlepších výsledků se povedlo dosáhnout pomocí zjednodušení obrazu mediánovým filtrem, který vykazoval přesnost segmentace silnice podle mteriky IoU 88,9 %. Druhou nejpřesnější metodou zjednodušení byl pak „occo“ algoritmus (kapitola 4.1.1) s výslednou přesností

88,3 % a třetího nejlepšího výsledku dosáhl bilaterální filtr s přesností 87,2 %. Nejhoršího výsledku přesnosti segmentace (88,4 %) bylo dosaženo při zjednodušení pomocí „erdr“ algoritmu (kapitola 4.1.1), který pracuje pouze s šedotónovým obrazem. Převedení obrazu do gradientu před aplikací rozvodí vedlo k lepšímu výsledku segmentace pouze u šedotónového „erdr“ filtru, a to jen o 0,3 %. U ostatních metod zjednodušení, které pracují s barevným obrazem, způsobil převod do gradientu horší výsledky, a proto se nejeví jako užitečné tuto úpravu provádět.

5.1.2 Testování rychlosti

Srovnání všech implementovaných algoritmů segmentace dle časového hlediska je vidět v grafu na obrázku 5.3. Do doby trvaní segmentace viz graf na obrázku 5.3 bylo započítáno zjednodušení obrazu, hledání značek i výsledná segmentace pomocí rozvodí. Je zde patrné, že nejvyšší průměrné rychlosti segmentace 66,7 ms dosáhl implementovaný „occo“ algoritmus využívající morfologických transformací otevření a uzavření. Druhý nejrychlejší byl mediánový filtr s dobou vykonání segmentace 130,5 ms a třetího nejlepšího času bylo dosaženo pomocí bilaterálního filtru s dobou trvání 215,5 ms. Naopak nejpomaleji proběhla segmentace pomocí „erdr“ algoritmu prvotně navrženého v teoretického návrhu v kapitole 3.2.



Obrázek 5.3 Srovnání rychlosti vykonání sémantické segmentace vozovky pro různé metody předzpracování obrazu

Rychlost vykonávání segmentace byla testována na stejném hardwaru, který byl použit k získaní dat pro tabulku 4.1. Jeho popis je uveden na stránce 48.

5.1.3 Zhodnocení

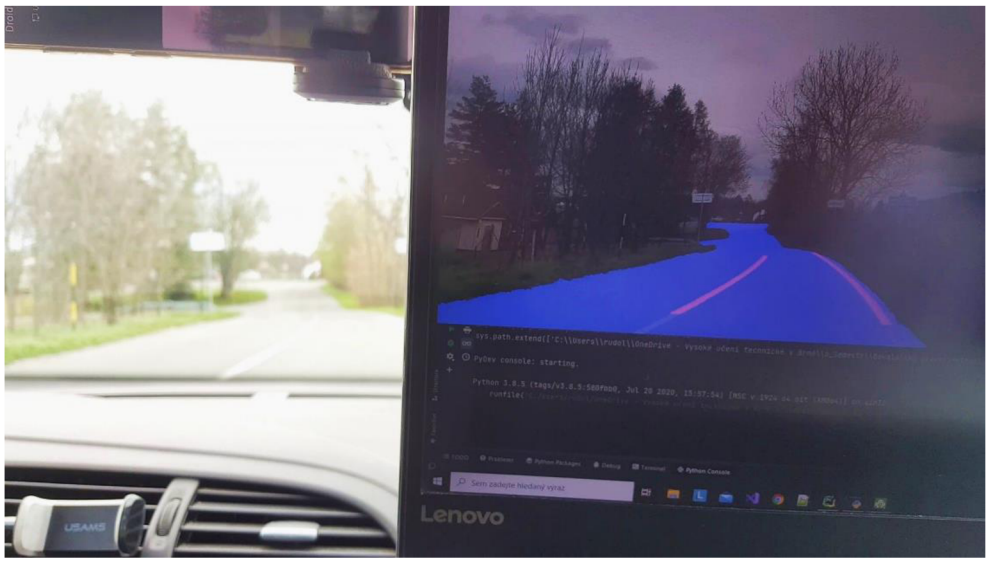
Na základě informací získaných z testování je patrné, že různé metody předzpracování barevného obrazu vykazují podobné výsledky segmentace při testování na reprezentativním datasetu a přesnost podle metriky IoU se pohybuje od 85 % do 88,9 %.

Jako nejvhodnější s ohledem na přesnost (graf na obrázku 5.2) i rychlost (graf na obrázku 5.3) byla zvolena metoda zjednodušení obrazu pomocí morfologického „occo“ algoritmu. Výsledky testování přesnosti a rychlosti implementovaných metod jsou přehledně uvedeny také v tabulce v příloze A.9.

Při testování metod segmentace na datech, pro které daný algoritmus nebyl navržen, tedy například na nočních záběrech, bylo zjištěno několik poznatků. Prvním je, že segmentace snímků pořízených v noci je značně nepřesná a často selhává. Dalším zásadní zjištěním je, že pokud jsou pro vstup do algoritmu použita data z jiné kamery, nebo je kamera odlišně umístěna, je nezbytné zkontrolovat, a případně upravit pozici statické značky silnice.

5.2 Testování na živém kamerovém vstupu

Testování na živém kamerovém vstupu bylo potřeba provést z toho důvodu, aby se mohla ověřit funkčnost navrženého řešení segmentace vozovky a detekce jízdních pruhů při reálném nasazení ve vozidle za běžného silničního provozu. Pro testování na živém kamerovém vstupu ve vozidle byl použit telefon Samsung Galaxy S10+ umístěný ve středu horní části čelního skla automobilu. Informace o optice fotoaparátu tohoto telefonu lze najít v kapitole 3.1. Pro živé streamování obrazu z telefonu s operační systémem Android do notebooku s operačním systémem Windows 10 pomocí USB kabelu byla využita mobilní aplikace *DroidCamX* pro Android. V notebooku pak byla nainstalována desktopová aplikace *DroidCam Client,* která umožnuje živý přenos videa z chytrého mobilního telefonu do počítače maximálně v rozlišení 1920x1080 pixelů. [38]



Obrázek 5.4 Ukázka z testování na živém kamerovém vstupu

Při testování bylo experimentálně ověřeno, že navržené a implementované řešení je možné aplikovat i na živý kamerový vstup, přičemž výsledky segmentace i detekce jízdních pruhů jsou totožné s výsledky testování na videích a fotografiích z reprezentativního datasetu. Navržené řešení segmentace vozovky lze tedy využít i pro nasazení v reálném čase. Průměrná doba zpracování segmentace včetně detekce jízdních pruhů u jednoho snímku byla 136,5 ms, což odpovídá 7,3 snímku za vteřinu. Ukázku z testování na živém kamerovém vstupu je možné vidět na obrázku 5.4. Videa zachycující testování segmentace včetně detekce jízdních pruhů jsou uložena v příloze A.3.

6. ZÁVĚR

Výsledkem bakalářské práce je teoretický návrh a implementace efektivního systému pro segmentaci vozovky s detekcí vodorovného dopravního značení na základě literární rešerše. Systém je koncipován tak, aby byl co nejpřesnější, nejuniverzálnější a nejrychlejší. Důraz byl kladen také na možnost realizace systému za pomocí dostupných knihoven pro zpracování obrazu OpenCV a scikit-image v programovacím jazyce Python včetně akcelerace výpočtů na grafické kartě.

V rámci práce byl vytvořen reprezentativní dataset se záběry silnic pohledem z vozidla. Dataset obsahuje 389 snímků vozovek, z toho 171 ručně anotovaných, a také 35 minut videí. Je v něm zastoupeno mnoho druhů scén ze silnic s betonovým a asfaltovým povrchem.

Po otestování a zhodnocení možných řešení byl pro sémantickou segmentaci vozovky implementován následující postup. V prvním kroku je barevný obraz vyfiltrován pomocí „occo“ algoritmu založeného na morfologickém otevření a uzavření popsaném v kapitole 4.1.1. Dále byl navržen a implementován algoritmus pro adaptivní hledání značek pozadí pro segmentaci na základě histogramu obrazu v barevném modelu HSV. Díky zmíněnému algoritmu se podařilo výsledky segmentace vozovky značně vylepšit. Závěrečná segmentace vozovky je provedena aplikaci metody rozvodí na filtrovaný obraz s pomocí nalezených značek. Ukázku výsledků lze vidět na obrázku 4.5 a v příloze E (A.10).

Implementovaná detekce čar jízdních pruhů využívá pro jejich nalezení výsledků sémantické segmentace. Vozovka je nejprve projektivně transformována na pohled shora, přičemž matice projektivní transformace je určována adaptivně na základě zužování silnice v obraze. Transformovaný obraz je adaptivně prahován za pomoci několika metod. Následnou aplikací metody posuvných oken [31] jsou v obraze detekovány čáry, které se nakonec proloží křivkou a vizualizují do původního obrazu. Blíže je celá implementace popsána v kapitole 4. Ukázku výsledků sémantické segmentace včetně detekce jízdních pruhů lze vidět na obrázku 4.11 (b) a ve videích v příloze A (A.3).

Všechny implementované algoritmy byly otestovány z hlediska rychlosti. To znamená, že byla vyhodnocena doba vykonání jednotlivých operací nad obrazem a na základě výsledků byla vybrána nejoptimálnější implementace. Doba trvání jednotlivých operací je shrnuta v tabulce 4.1 na straně 48. Dalším otestovaným parametrem byla přesnost sémantické segmentace za použití různých metod předzpracování obrazu. Z výsledků vyplývá, že nejefektivnější segmentace z hlediska přesnosti i času vykonání byla dosažena za použití „occo“ filtrace obrazu. Algoritmus s danou filtrací dosáhl přesnosti segmentace na vytvořeném datasetu dle metriky IoU 88,3 % s průměrným časem segmentace jednoho snímku 66,7 ms. Srovnání přesnosti segmentace u všech testovaných metod je uvedeno v grafu na obrázku 5.2 a porovnání rychlostí znázorňuje graf na obrázku 5.3. Navíc bylo experimentálně ověřeno, že systém je plně funkční i na

živém vstupu z kamery ve vozidle. Při testování na nočních snímcích, pro které nebyl tento systém navržen, bylo zjištěno, že systém selhává. Dalším nalezeným omezením z hlediska korektního fungování segmentace a detekce jízdních pruhů je nutnost správného umístění statické značky silnice pro segmentaci. Z tohoto důvodu je důležité danou značku kalibrovat při využití jiné kamery, či při změně umístění kamery. Limitující vlastností detekce jízdních pruhů pak je schopnost detekovat pouze čáry jízdních pruhů, které nejsou orientované k vozidlu kolmým směrem. Na některých typech silnic se také vyskytuje problém s chybnou detekcí čáry jízdního pruhu v místech, kde čára není, nebo je mimo vozovku. To je však možné řešit například tím, že se provede bitový součin eroze výsledku segmentace vozovky a masky s naprahovaným obrazem jízdních pruhů.

Dalším postupem, navazujícím na tuto práci, může být integrace systému do vozidla a na základě získaných dat realizace asistence udržování vozidla na vozovce a v jízdních pruzích. Systém by také bylo možné vylepšit nalezením vhodných parametrů adaptivního hledání značek a segmentace i pro noční scénu. Dále, pokud je na vozovce více jízdních pruhů, bylo by možné jednoduchou úpravou algoritmu hledání počátku čar detekovat i více čar, což by umožnilo sledovat i vedlejší jízdní pruhy.

Ze statistik nehodovosti uvedených v úvodu této práce je patrné, že přínosy práce zaměřené na segmentaci vozovky a detekci jízdních pruhů horizontálního značení v dopravě jsou jednoznačné. Je proto důležitý další vývoj systémů, optimalizace, stejně jako rozvoj nových přístupů k této problematice. V budoucnu bude možné díky tomuto zdokonalování dosahovat stále větší přesnosti i rychlosti těchto systémů, a s tím související bezpečnosti silničního provozu.

LITERATURA

[1] CICCHINO, Jessica B. Effects of lane departure warning on police-reported crash rates. *Journal of Safety Research* [online]. 2018, **66**, 61-70 [cit. 2021-5-14]. ISSN 00224375. Dostupné z: doi:10.1016/j.jsr.2018.05.006

[2] HIGHWAY LOSS DATA INSTITUTE. *Bulletin: Compendium of HLDI collision avoidance research* [online]. Arlington, December 2020, **2020**(Vol. 37, 12) [cit. 2021-5-14]. Dostupné z: https://www.iihs.org/media/e635cc76-b9bc-4bad-a30a-5d7b78791df2/vxeQ3A/HLDI%20Research/Collisions%20avoidance%20features/37-12-compendium.pdf

[3] STERNLUND, Simon, Johan STRANDROTH, Matteo RIZZI, Anders LIE a Claes TINGVALL. The effectiveness of lane departure warning systems—A reduction in real-world passenger car injury crashes. *Traffic Injury Prevention* [online]. 2017, **18**(2), 225-229 [cit. 2021-5-14]. ISSN 1538-9588. Dostupné z: doi:10.1080/15389588.2016.1230672

[4] HLAVÁČ, Václav a Milan ŠONKA. *Počítačové vidění*. Praha: Grada, 1992. ISBN isbn80-85424-67-3.

[5] ŠONKA, Milan, Václav HLAVÁČ a Roger BOYLE. *Image Processing, analysis, and machine vision*. 4th ed. Stamford: Cengage Learning, 2015. ISBN isbn978-1-133-59360-7.

[6] GONZALEZ, Rafael C. a Richard E. WOODS. *Digital image processing*. Fourth edition. New York: Pearson, [2018]. ISBN isbn978-1-292-22304-9.

[7] IBRAHEEM, Noor A., et al. Understanding color models: a review. *ARPN Journal of science and technology*, 2012, 2.3: 265-275.

[8] HSL and HSV. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2020 [cit. 2020-11-22]. Dostupné z: https://en.wikipedia.org/wiki/HSL_and_HSV

[9] SZELISKI, Richard. *Computer Vision: Algorithms and Applications*. London: Springer, 2010. Texts in computer science. ISBN isbn9781848829350.

[10] ŽÁRA, Jiří. *Moderní počítačová grafika*. 2., přeprac. a rozš. vyd. Brno: Computer Press, 2004. ISBN 80-251-0454-0.

[11] SOJKA, Eduard. *Digitální zpracování a analýza obrazů*. Ostrava: VŠB-Technická univerzita, 2000. ISBN isbn80-7078-746-5.

[12] GOODFELLOW, Ian, Yoshua BENGIO a Aaron COURVILLE. *Deep learning*. Cambridge, MA: MIT press, [2016]. Adaptive computation and machine learning series. ISBN 978-0262035613.

[13] BADRINARAYANAN, Vijay, Alex KENDALL a Roberto CIPOLLA. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* [online]. 2017, **39**(12), 2481-2495 [cit. 2021-5-5]. ISSN 0162-8828. Dostupné z: doi:10.1109/TPAMI.2016.2644615

[14] HASSAN, Muneeb UL. *VGG16 – Convolutional Network for Classification and Detection* [online]. 20 November 2018 [cit. 2021-5-7]. Dostupné z: https://neurohive.io/en/popular-networks/vgg16/

[15] SIMONYAN, Karen; ZISSERMAN, Andrew. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[16] NOH, Hyeonwoo, Seunghoon HONG a Bohyung HAN. Learning Deconvolution Network for Semantic Segmentation. In: *2015 IEEE International Conference on Computer Vision (ICCV)* [online]. IEEE, 2015, 2015, s. 1520-1528 [cit. 2021-5-7]. ISBN 978-1-4673-8391-2. Dostupné z: doi:10.1109/ICCV.2015.178

[17] PASZKE, Adam, et al. Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147*, 2016.

[18] CSURKA, Gabriela, Diane LARLUS a Florent PERRONNIN. What is a good evaluation measure for semantic segmentation? In: *Procedings of the British Machine Vision Conference 2013* [online]. British Machine Vision Association, 2013, 2013, 32.1-32.11 [cit. 2021-04-05]. ISBN 1-901725-49-9. Dostupné z: doi:10.5244/C.27.32.

[19] SOKOLOVA, Marina a Guy LAPALME. A systematic analysis of performance measures for classification tasks. *Information Processing & Management* [online]. 2009, **45**(4), 427-437 [cit. 2021-04-05]. ISSN 03064573. Dostupné z: doi:10.1016/j.ipm.2009.03.002.

[20] FERNANDEZ-MORAL, Eduardo, Renato MARTINS, Denis WOLF a Patrick RIVES. A New Metric for Evaluating Semantic Segmentation: Leveraging Global and Contour Accuracy. In: *2018 IEEE Intelligent Vehicles Symposium (IV)* [online]. IEEE, 2018, 2018, s. 1051-1056 [cit. 2021-04-06]. ISBN 978-1-5386-4452-2. Dostupné z: doi:10.1109/IVS.2018.8500497

[21] LUTZ, Mark. *Learning Python*. 5th ed. Sebastopol: O´Reilly, 2013. ISBN 978-1-449-35573-9.

[22] BRADSKI, Gary. *The OpenCV Library*. Dr. Dobb's Journal of Software Tools, 2000.

[23] LAGANIÉRE, Robert. *OpenCV 2 computer vision application programming cookbook: over 50 recepies to master this library of programming functions for real-time computer vision*. Brimmingham: Packt Publishing, 2011. Quick Answers to Common Problems. ISBN isbn978-1-849513-24-1.

[24] VAN DER WALT, Stéfan, Johannes L. SCHÖNBERGER, Juan NUNEZ-IGLESIAS, François BOULOGNE, Joshua D. WARNER, Neil YAGER, Emmanuelle GOUILLART a Tony YU. Scikit-image: image processing in Python. *PeerJ* [online]. 2014, **2** [cit. 2021-03-26]. ISSN 2167-8359. Dostupné z: doi:10.7717/peerj.453.

[25] IDRIS, Ivan. *Numpy Beginner's Guide*. Second Edition. Birmingham: Packt Publishing, c2013. ISBN 978-1-78216-608-5.

[26] HARRIS, Charles R., K. Jarrod MILLMAN, Stéfan J. VAN DER WALT, et al. Array programming with NumPy. *Nature* [online]. 2020, **585**(7825), 357-362 [cit. 2021-03-26]. ISSN 0028-0836. Dostupné z: doi:10.1038/s41586-020-2649-2.

[27] HUNTER, John D. Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering* [online]. 2007, **9**(3), 90-95 [cit. 2021-5-4]. ISSN 1521-9615. Dostupné z: doi:10.1109/MCSE.2007.55

[28] DUTTA, Abhishek a Andrew ZISSERMAN. The VIA Annotation Software for Images, Audio and Video. In: *Proceedings of the 27th ACM International Conference on Multimedia* [online]. New York, NY, USA: ACM, 2019, 2019-10-15, s. 2276-2279 [cit. 2021-04-14]. ISBN 9781450368896. Dostupné z: doi:10.1145/3343031.3350535.

[29] MATLAB. *Image Labeler* [online]. The MathWorks, c1994-2021 [cit. 2021-04-14]. Dostupné z: https://www.mathworks.com/help/vision/ref/imagelabeler-app.html

[30] BEUCHER, Serge; BILODEAU, M.; YU, X. Road segmentation by watershed algorithms. In: *PROMETHEUS Workshop, Sophia Antipolis, France*. 1990.

[31] REZWANUL HAQUE, Md., Md. MILON ISLAM, Kazi SAEED ALAM a Hasib IQBAL. A Computer Vision based Lane Detection Approach. *International Journal of Image, Graphics and Signal Processing* [online]. 2019, **11**(3), 27-34 [cit. 2021-03-10]. ISSN 20749074. Dostupné z: doi:10.5815/ijigsp.2019.03.04.

[32] SUN, Ziqiang. Vision Based Lane Detection for Self-Driving Car. *2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications(AEECA)* [online]. IEEE, 2020, 2020, , 635-638 [cit. 2021-03-10]. ISBN 978-1-7281-6521-9. Dostupné z: doi:10.1109/AEECA49918.2020.9213624

[33] THE PYTHON SOFTWARE FOUNDATION. *The Python Language Reference* [online]. c2001-2021 [cit. 2021-4-16]. Dostupné z: https://docs.python.org/3.8/reference/index.html

[34] JETBRAINS. *PyCharm 2021.1.1 (Professional Edition)* [software]. c2000-2021 [cit. 2021-5-4]. Dostupné z: https://www.jetbrains.com/pycharm/

[35] ROBINSON, Kevin a Paul F. WHELAN. Efficient morphological reconstruction: a downhill filter. *Pattern Recognition Letters* [online]. 2004, **25**(15), 1759-1767 [cit. 2021-03-18]. ISSN 01678655. Dostupné z: doi:10.1016/j.patrec.2004.07.002.

[36] PETERS, R.A. A new algorithm for image noise reduction using mathematical morphology. *IEEE Transactions on Image Processing* [online]. 1995, **4**(5), 554-568 [cit. 2021-03-21]. ISSN 1057-7149. Dostupné z: doi:10.1109/83.382491.

[37] ČERMÁK, Libor a Rudolf HLAVIČKA. *Numerické metody*. Vydání třetí. Brno: Akademické nakladatelství CERM, 2016. ISBN 978-80-214-5437-8.

[38] DEV47 APPS. *DroidCam* [online]. Vancouver [cit. 2021-5-3]. Dostupné z: https://www.dev47apps.com/

SEZNAM PŘÍLOH

Příloha A - Zdrojový kód programu, dataset a ukázková videa jsou uložena na přiloženém DVD

A.1 Struktura složky program

- modules - zde jsou uloženy moduly obsahující funkce určené pro detekci jízdních pruhů a segmentaci vozovky
  - lanes_sliding_window.py - modul pro detekci jízdních pruhů pomocí posuvných oken.
  - lanes_thresholding.py - modul pro prahování obrazu obsahující jízdní pruhy.
  - perspective_transform.py - modul pro adaptivní projektivní transformaci.
  - preprocess_before_segmentation.py - modul pro předzpracování obrazu před segmentací.
  - segmentation_markers.py - modul pro adaptivní hledání značek pozadí v obraze.
  - segmentation_watershead.py - modul pro segmentaci obrazu metodou watershead.

- segmentation_methods - zde jsou implantace segmentace vozovky pomocí různých předzpracování obrazu.
  - seg_bilateral.py
  - seg_erdr.py
  - seg_meanshift.py
  - seg_occo.py

- live_video_segmentation_with_lane_detection.py - Program pro sémantickou segmentaci vozovky s živým vstupem videa z kamery.
- viedo_segmentation_with_lane_detection.py- Program pro sémantickou segmentaci vozovky videa.
- segmentatin_accuracy_test.py - Program pro otestování přesnosti sémantické segmentace na reprezentativním datasetu.
- segmentation_methods_test.py - Program pro segmentaci vozovky s možností výběru metody segmentace.

A.2 Struktura složky dataset

- all_dataset_images - všechny vytvořené obrázky datasetu (389 snímků)
- dataset_images - obrázky pro testování sémantické segmentace (171 snímků)
- ground_truth - obrázky pro testování obsahující ručně označenou silnici (171 snímků)
- videos – videa určená pro testování segmentace a detekce jízdních pruhů (30minut)

A.3 Struktura složky s video ukázkami

- live_video_input - ukázkové video z testování na živém kamerovém vstupu
- segmentation_with_lane_detection_1 – ukázka č.1 z testování na datasetu
- segmentation_with_lane_detection_2 – ukázka č.2 z testování na datasetu

Příloha B - Ukázka zdrojového kódu algoritmu pro adaptivní hledání značek pozadí

A.4 Adaptivní prohledávání histogramu

```
import numpy as np

def adaptive_histogram(img_hsv, val_limit: np.uint8,
                       iter_limit: np.uint8):
    limits = [], [], []
    for i in range(3):
        histogram, bins = np.histogram(img_hsv[:, :, i].ravel(),
                                       256, [0, 256])
        over_limit = False
        iterating = False
        right_limits, left_limits = [], []
        iteration = 0

        for j, x in np.ndenumerate(histogram):
            if not over_limit and x > val_limit:
                val = max(j[0] - iter_limit, 0)

                if not iterating and (not right_limits or
                                      right_limits[-1] < val):
                    if len(left_limits) == len(right_limits):
                        left_limits.append(val)

                elif right_limits and right_limits[-1] >= val:
                    right_limits.pop()

                over_limit = True
                iterating = False

            if over_limit and x < val_limit:
                over_limit = False
                iterating = True
                iteration = 0

            if iterating and not over_limit:
                iteration += 1

            if iteration >= iter_limit:
                iterating = False
                iteration = 0
                if left_limits[-1] < j[0] and
                          len(left_limits) - 1 == len(right_limits):
                    right_limits.append(j[0])

        limits[i].append(left_limits)
        if len(right_limits) == len(left_limits) - 1:
            right_limits.append(255)
        limits[i].append(right_limits)

    return limits
```

A.5 Prahování obrazu pro získaní značek

```
import cv2 as cv

def color_thresholding(img, limits):
    mask = 255 * np.ones((img.shape[0], img.shape[1]), np.uint8)

    for i in range(len(limits)):
        mask1 = np.zeros((img.shape[0], img.shape[1]), np.uint8)

        for j in range(len(limits[i][0])):
            l = limits[i][0][j]
            u = limits[i][1][j]
            l_b, u_b = np.array([]), np.array([])

            if i == 0:
                l_b = np.array([l, 0, 0])
                u_b = np.array([u, 255, 255])
            elif i == 1:
                l_b = np.array([0, l, 0])
                u_b = np.array([180, u, 255])
            else:
                l_b = np.array([0, 0, l])
                u_b = np.array([180, 255, u])

            mask1 = cv.bitwise_or(mask1, cv.inRange(img, l_b, u_b))

        mask = cv.bitwise_and(mask, mask1)

    return mask
```

A.6 Funkce adaptivního prahování

```
hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)
hsv_road = hsv[590:710, 520:700, :]

histogram_limits = adaptive_histogram(hsv_road, val_limit, iter_limit)
mask = color_thresholding(hsv, histogram_limits)

kernel_morph = np.ones((markers_ksize, markers_ksize), np.uint8)
mask_morph = cv.erode(~mask, kernel_morph, iterations=1)
```

Příloha C - Ukázka zdrojového kódu algoritmu adaptivního určení matice projektivní transformace

A.7 Funkce pro získaní matice transformace

```
import cv2 as cv; import numpy as np
roi_widths_history = []
roi_heights_history = []
Def get_transformation_matrices(img, segmentation_overlay,
                                visualize=False):
    segmentation_overlay_road_width_arr = np.sum(
                                  segmentation_overlay, axis=1) / 255

    road_heights = roi_from_segmented_img(30,
                                  segmentation_overlay_road_width_arr)

    poly = np.polyfit(road_heights[0], road_heights[1], deg=1)
    poly_edited = np.array([poly[0],
                            img.shape[1]-poly[0]*img.shape[0]])
    poly_1d_1 = np.poly1d(poly_edited)

    roi_widths_history.append(max(int(poly_1d_1(
                                     road_heights[0][-1]) / 2), 30))

    roi_heights_history.append(road_heights[0][-1]
                      if road_heights[0][-1] > img.shape[0] / 2
                      else int(img.shape[0] / 2))

    if len(roi_widths_history) > 10:
        roi_widths_history.pop(0)
        roi_heights_history.pop(0)

    roi_top_width = int(np.mean(roi_widths_history))
    roi_top_center = int(img.shape[1] / 2) + int(img.shape[1] * 0.00)
    roi_r = roi_top_center + roi_top_width
    roi_l = roi_top_center - roi_top_width
    roi_height = int(np.mean(roi_heights_history))

    points_roi = np.float32([[roi_l, roi_height],
                             [roi_r, roi_height],
                             [0, img.shape[0] - 40],
                             [img.shape[1], img.shape[0] - 40]])

    points_result = np.float32([[400, 0],
                                [img.shape[1] - 400, 0],
                                [400, img.shape[0]],
                                [img.shape[1] - 400, img.shape[0]]])

    transformation_matrix = cv.getPerspectiveTransform(points_roi,
                                                       points_result)
    transformation_matrix_inv = cv.getPerspectiveTransform(
                                            points_result, points_roi)

    return transformation_matrix, transformation_matrix_inv
```

A.8 Funkce pro zjištění rychlosti zužování silnice

```
Import numpy as np

def roi_info_from_segmented_img(row_height, road_width_array):
    MIN_ROAD_POINTS = 60

    row_pos = len(road_width_array)
    road_avg_heights = []
    row_positions = []

    for x in range(0, int(len(road_width_array) / row_height)):

        avg_road_points = int(np.mean(
                        road_width_array[row_pos - row_height:row_pos]))
        if x == int(len(road_width_array) / row_height) or
                    avg_road_points <= MIN_ROAD_POINTS:
            break

        road_avg_heights.append(avg_road_points)
        row_positions.append(row_pos)

        row_pos -= row_height

    return [row_positions, road_avg_heights]
```

Příloha D - Naměřené hodnoty

A.9 Tabulka naměřených přesností a doby vykonání sémantické segmentace

Metoda předzpracování	**IoU [%]**	**F1 [%]**	**Čas [ms]**
erdr	81,4	88,4	617,4
erdr sobel	81,7	88,7	658,6
bilateral	87,2	92,6	215,5
bilateral sobel	85,0	91,2	387,0
mean shift	86,4	92,0	314,7
mean shift sobel	85,0	91,0	257,9
median	88,9	93,7	130,5
median sobel	87,1	92,4	239,5
occo	88,3	93,2	66,7
occo sobel	87,5	92,7	178,7

Příloha E - Ukázka výsledků segmentace vozovky

A.10 Obrázky s výsledky sémantické segmentace