Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Robot pro sledování čáry pro android / Android based Line Following Robot

Plzeň 2020

Lukáš Zahradník

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 20. července 2020

Lukáš Zahradník

Abstract

The purpose of this thesis is to test the abilities of Android smartphones as a central control unit for automatic vehicles and creation of API, with which we could easily implement custom application suitable for this task.
Easiest way to control path of automatic moving vehicle is to let it follow track defined by a line. Automatic moving vehicle have certain tasks to do, when following its track. To control what and when will the vehicle do, we can put signs around the line. Each sign would be bind to a certain task. This behavior can be achieved by processing images from the camera of the Android device.

Abstrakt

Obsahem této práce je otestovat možnosti chytrých telefonů s operačním systémem Android jakožto řídícího zařízení vozidla a vytvoření API, které by umožňovalo snadnou implementaci vlastní aplikace schopnou vykonávat tuto činnost.
Nejjednoduším způsobem kontroly trasy automaticky se pohybujícího vozidla je nechat ho následovat námi vytvořenou čáru. Na trase musí vozidlo vykonávat různé úkony. Aby vozidlo vědělo, kdy a kde dané úkony vykonávat, můžeme kolem čáry umístit různé značky, kdy každá značka bude znamenat jiný úkon ke splnění. Tohoto chování lze docílit pomocí zpracovávání obrazu z fotoaparátu zařízení.

Poděkování

Rád bych poděkoval Ing. Tomáši Mainzerovi, Ph.D. za cenné rady, věcné připomínky, vstřícnost při vypracování bakalářské práce a za vytvoření vozidel k testování aplikace.

Obsah

1 Úvod **8**

2 Sledování čáry pomocí vizuálních dat **9**
2.1 Jasová korekce 10
2.2 Edge detection 13
2.2.1 Sobel Edge Detector 13
2.2.2 LoG detektor hran (Laplacian of Gaussian Edge Detector) 14

3 Detekce, rozpoznávání a analýza jednoduchých geometrických tvarů **15**
3.1 Úvod do problematiky 15
3.2 Boundary Scalar Techniques 16
3.2.1 Zisk jedno-dimenzionální funkce 16
3.2.2 Popis pomocí Fourierových transformací 17
3.2.3 Popis pomocí Stochastických procesů modelování . . 17
3.3 Boundary Space-Domain Techniques 17
3.3.1 Freemanovy řetězové kódy 17
3.3.2 Syntaktické metody 18
3.3.3 Scale-Space techniky 18
3.3.4 Aproximace hranic 18
3.4 Algoritmus Douglas-Peucker 19
3.4.1 Popis 19
3.4.2 Postup 19
3.4.3 Závěr 19
3.5 Sledování hranic tvarů (Border Following) 20
3.5.1 Popis 20
3.5.2 Definice 20
3.5.3 První algoritmus 23
3.5.4 Druhý algoritmus 24

4 Existující řešení **25**
4.1 Klasický přístup 25
4.2 Vizuální přístup 25

5 Analýza a návrh **26**
5.1 Sledování čáry 26
5.2 Extrakce hran čáry 26
5.3 Analýza získané čáry 26
5.4 Rozpoznávání geometrických tvarů a barev 27

6 Implementace **28**
6.1 OpenCV 28
6.1.1 Použité knihovní funkce 28
6.2 API 31
6.2.1 IVehicleController 32
6.2.2 IShapeDetectionReactionController 33
6.2.3 ShapeMessenger 33
6.2.4 ShapeDetection 34
6.2.5 LineFollowing 36
6.3 Aplikace 40
6.3.1 MainActivity 40
6.3.2 Settings 42
6.3.3 SettingsFragment 43
6.3.4 ShapeReactionController 43
6.3.5 WiFiVehicleController 44
6.3.6 USBService 45
6.3.7 USBVehicleController 45

7 Testování a výsledky **46**
7.1 Způsob testování 46
7.2 Detekce geometrických tvarů 46
7.3 Následování čáry 49
7.3.1 Komunikace 49
7.3.2 Detekce čáry 49
7.3.3 Celkové zhodnocení přístupů k detekci a následování čáry 53

8 Závěr **54**

Literatura

Přílohy

1 Úvod

V naší společnosti máme přístup k obrovskému množství zboží. Aby bylo možné odpovídat poptávce, je zboží vyráběno ve velkém množství, které se musí někde uložit, než je prodáno. Díky tomu je mít co nejefektivnější sklady s dobrým managementem, aby bylo možné zboží co nejdříve expedovat.

Problém s managementem zboží řeší komplexní znalostní systémy, umožňující automatické zpracování dat, udržování inventury skladu, zpracování papírování, atd. Ovšem fyzickou manipulaci se zbožím stále provádí především lidé. Zaměstnanci jsou poměrně drazí na provoz, dělají chyby a v mnoho případech jich není dostatek. Nabízí se tedy plná automatizace skladů, např. pomocí autonomních vozítek. Tento přístup by značně snížil počet zaměstnanců a rizik dosavadního zpracování skladů a také podstatně snížil náklady na jeho údržbu.

Jednou z možností realizace plně autonomního skladu je sada autonomních vozítek, které by řešily třídění zboží, skladování a jeho přepravu po skladu. Nejjednodušším řešením pro organizovaný pohyb vozítek po skladu je sledování čáry. Čára může měnit barvy a taky obsahovat různé tvary, na které různá vozítka budou různě reagovat.

Pro tuto funkcionalitu by bylo potřeba značné množství senzorů a specializovaných kontrolerů. Proč tedy nevyužít zařízení, které nosíme každý den u sebe – Smartphone. Smartphony v dnešní době mají poměrně velký výpočetní výkon, obsahují množství senzorů, kvalitní fotoaparát a moduly pro komunikaci, jako Wi-fi a Bluetooth.

V rámci této bakalářské práce implementovat API se základní funkcionalitou, jako sledování čáry a rozpoznávání základních geometrických tvarů. Poté vytvořím demo aplikaci pomocí tohoto API pro demonstraci jeho funkčnosti.

2 Sledování čáry pomocí vizuálních dat

[1] Pro vytvoření robota, který bude úspěšně následovat čáru, je nutné překonat několik překážek. Za prvé je nutné zlepšit získaný obraz. Největším problémem je špatné nasvícení, který by mohl způsobit neschopnost následovat čáru ve špatných světelných podmínkách. Vzhledem k tomu, že námi navrhovaný software je určen pro chytré telefony, můžeme použít blesk u fotoaparátu ke zmírnění tohoto problému, ovšem ne k úplnému řešení. Pro zlepšení kvality obrazu pro naší aplikaci lze použít techniku jasové korekce (Gray Levels Normalization). Poté je nutné získat námi hledanou čáru. Toho docílíme technikou detekce hran (Edge Detection). Tato technika zvýrazní přechod mezi námi hledanou čárou a pozadím. K tomu použijeme konvoluční filtr navržený pro techniku edge detection. Nakonec se musíme nad získanými daty rozhodnout, jak se bude robot pohybovat. Ideálním řešením pro plynulý pohyb robota je určení COG (Center of gravity = těžiště). Pomocí této techniky nalezneme těžiště námi nalezených přechodů. Podle toho, jak nalevo, či napravo se bude těžiště nacházet, můžeme volit směr a rychlost rotace motorů.

[1] Všechny anglické názvy použité v této bakalářské práci byly přeloženy pomocí přednášek předmětu Zpracování digitalizovaného obrazu z Katedry kybernetiky Fakulty aplikovaných věd Západočeské univerzity v Plzni (`http://www.kky.zcu.cz/uploads/courses/zdo/ZDO_aktual_130215.pdf`)

2.1 Jasová korekce

K technice jasové korekce budu používat metodu **Ekvalizace histogramu**.

[3, pp. 91–94] Představme si, že existuje spojitá funkce a proměnná r, reprezentující jas vstupního obrazu. Proměnná r byla normalizována do intervalu [0, 1], kde 0 reprezentuje černou barvu a 1 bílou. Pro r splňující tuto existuje funkce T:

$$s = T\left(r\right), \quad 0 \leq r \leq 1$$

Která vytvoří jas pro každý pixel (s) z hodnoty r vstupního obrazu. Pro $T(r)$ platí:

1. $T(r)$ monotónní prostá rostoucí funkce na intervalu $0 \leq r \leq 1$
2. Funkce nabývá hodnot $0 \leq T(r) \leq 1$ *pro* $0 \leq r \leq 1$

Podmínka 1 zajišťuje, že existuje inverzní transformace a pořadí černé a bílé barvy v obrazu. Podmínka 2 zajišťuje, že výsledný obraz bude mít jasové hodnoty ve stejném intervalu jako vstupní obraz. Opačná funkce pro zpětnou transformaci vypadá následovně:

$$r = T^{-1}\left(s\right), \quad 0 \leq s \leq 1$$

Gray levels v obrazu si můžeme představit jako náhodné veličiny na intervalu [0, 1]. Jednou ze základních deskriptorů náhodné veličiny je její hustota pravděpodobnosti (Probability Density Function, PDF). Nechť $p_r(r)$ *a* $p_s(s)$ jsou hustoty pravděpodobnosti náhodných veličin r *a* s. Ze základní teorie pravděpodobnosti víme, že když $p_r(r)$ *a* $T(r)$ jsou známé a T^{-1} splňuje podmínku a), tak ps (s) lze získat následujícím vzorcem:

$$p_s\left(s\right) = \; p_r\left(r\right)\left|\frac{dr}{ds}\right|$$

Transformační funkce T tedy vypadá následovně:

$$s = T\left(r\right) = \int_0^r p_r\left(w\right)dw$$

kde w je náhrada za derivaci z předešlého vzorce. Pravá strana tohoto vzorce je distribuční funkce náhodné veličiny r. Po aplikování Lebnizova pravidla na předešlou rovnici zjistíme:

$$\frac{ds}{dr} = \frac{dT(r)}{dr} = \; \frac{d}{dr}\left(\int_0^r p_r\left(w\right)dw\right) = p_r(r)$$

Substitucí tohoto poznatku do 3. Rovnice zjistíme:

$$p_s(s) = p_r(r)\left|\frac{1}{p_r(r)}\right| = 1, \quad 0 \leq s \leq 1$$

Jelikož je $p_s(s)$ funkcí hustoty pravděpodobnosti, musí platit, že je mimo interval [0, 1] rovna nule, protože její integrál na intervalu musí být rovný 1. Dozvěděli jsme se tedy, že $T(r)$ je závislé na $p_r(r)$, ale výsledná funkce $p_s(s)$ je vždy uniformní a nezávislá na tvaru $p_r(r)$.

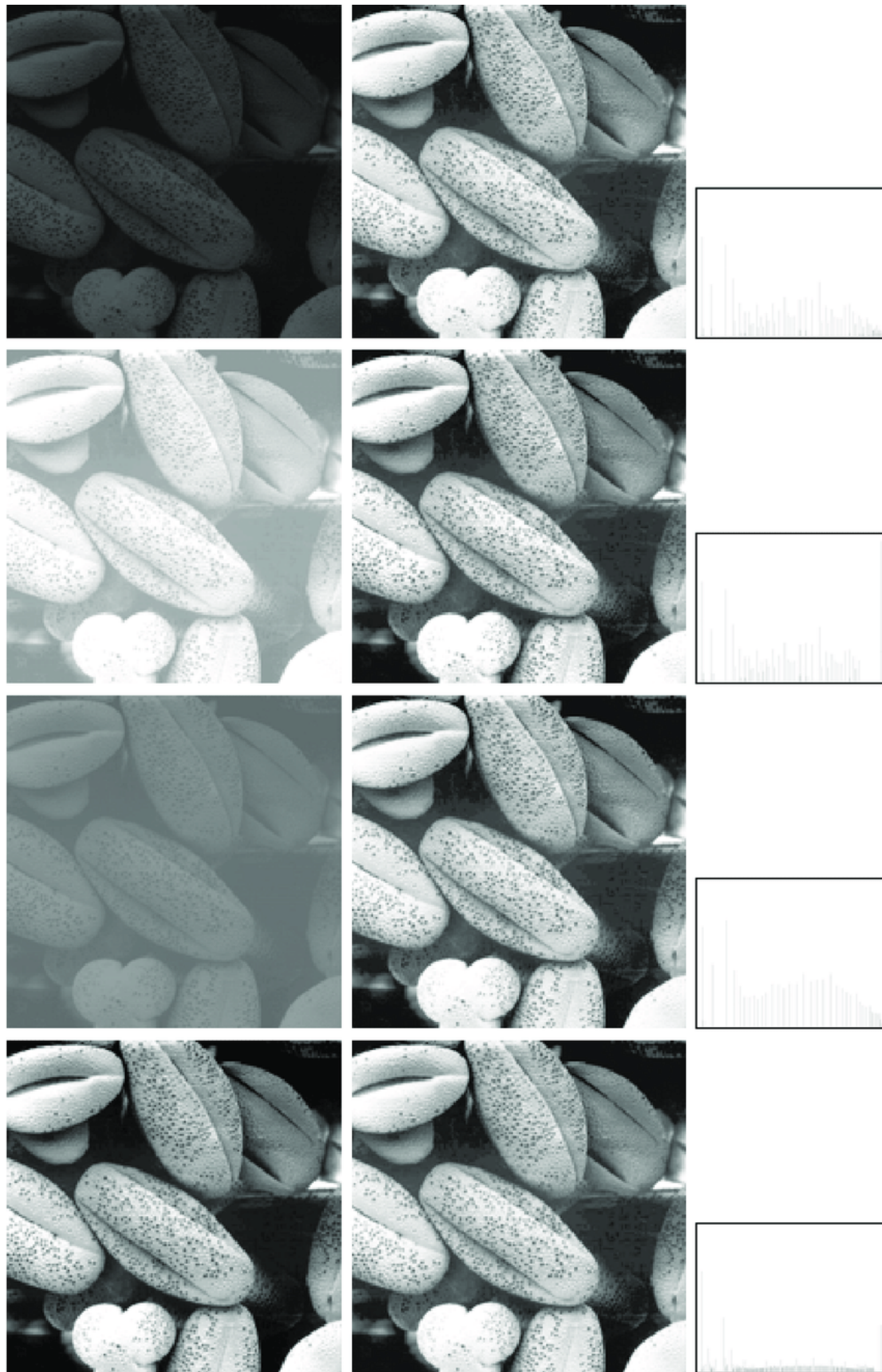
Pro diskrétní hodnoty se zabýváme jejich pravděpodobnostmi a součty namísto fukcemi hustot pravděpodobností a integrály. Pravděpodobnost výskytu gray levelu r_k v obrázku je aproximováno následovně:

$$p_r(r_k) = \frac{n_k}{n}, \quad k = 0, 1, 2, \ldots, L-1$$

kde n je počet pixelů vstupního obrazu, n_k je počet pixelů s gray levelem r_k a L je počet všech možných gray levelů obrazu. Diskrétní verze 4. vzorce:

$$s_k = T(r_k) = \sum_{j=0}^{k} p_r(r_j) = \sum_{j=0}^{k} \frac{n_j}{n}, \quad k = 0, 1, 2, \ldots, L-1$$

Výsledný obraz získáme mapováním každého pixelu s gray levelem r_k vstupního obrazu na korespondující pixel s gray levelem s_k výstupního obrazu. Vztah mezi $p_r(r)$ a p_r se nazývá histogram. Transformace daná předešlým vzorcem se jmenuje Histogram Equalization nebo Histogram Linearization. V obrázku 2.1 lze vidět ukázku této transformace



Obrázek 2.1: Výsledky histogram equalization. 1. sloupec jsou vstupní obrazy, 2. sloupec výsledky a 3.odpovídající histogramy. Obrázek převzán z [3, Figure 3.17]

2.2 Edge detection

[2, 7] Hrany objektů představují jednu z nejdůležitějších vlastností objektů. Jedná se o oblasti s vysokým kontrastem. Nalezením okrajů objektů v obrazech můžeme vyfiltrovat spoustu nepotřebných informací a tím si značně ulehčit práci při rozpoznávání objektů. Existuje mnoho Edge detection technik, dají se však rozdělit na dvě skupiny – Gradient a Laplacian.

Gradient techniky detekují hrany hledáním maxim a minim po první derivaci obrazu. Laplacian metody hledají místa s nulovým křížením (zero crossing) ve druhé derivaci obrazu. V této sekci popíši jednoho zástupce algoritmů z gradientních metod – Sobel a jednoho zástupce Laplaceovských metod – LoG (Laplacian of Gaussian). Tyto algoritmy budou implementovány a otestovány. Na základě výsledků testů bude vybrán nejideálnější, který bude použit ve výsledné podobě aplikace.

2.2.1 Sobel Edge Detector

Měří 2-D gradient vstupního obrazu, čímž zvýrazňuje oblasti s vysokým gradientem, které náleží hranám objektů v obrazu. V algoritmu se používá pár konvolučních masek rozměru 3x3, přičemž masky jsou identické, jen otočené o 90 oproti sobě. Masky jsou navrženy tak, aby jedna co nejvíce korespondovala vertikálním a druhá horizontálním hranám obrazu. Tyto masky mohou být aplikovány separátně k získání výsledků pro obě orientace (nazveme je Gx a Gy). Vidět je můžete v obrázku 2.2

1	0	-1
2	0	-2
1	0	-1

-1	-2	-1
0	0	0
1	2	1

Obrázek 2.2: 3x3 maska pro Sobel Edge Detector. Vpravo Gx, vlevo Gy. Obrázek převzat z [2, Fig. 1]

Pak mohou být zkombinovány pro nalezení absolutní velikosti gradientu pro každý bod a orientaci gradientu. Velikost gradientu lze získat těmito dvěma vzorci (druhý vzorec je aproximace prvního. Výpočet je ale výrazně rychlejší, proto se používá častěji):

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

$$|G| = |G_x| + |G_y|$$

Úhel hrany vzhledem k pixelové mřížce lze získat tímto vzorcem:

$$0 = \arctan\left(\frac{G_x}{G_y}\right) - \frac{3}{4}\pi$$

Tento algoritmus je rychlejší než LoG Edge detector, ale je méně odolný vůči šumu.

2.2.2 LoG detektor hran (Laplacian of Gaussian Edge Detector)

Jádrem tohoto algoritmu je LoG filter. LoG hledá místa v Laplacian obrazu (výsledek po přefiltrování vstupního obrazu LoG filterem), kde Laplacian prochází nulou, nebo–li místa, kde Laplacian mění znaménko. Výsledek je značně ovlivněn velikostí Gaussianu v fázi vyhlazování obrazu (čím větší Gaussian, tím více vyhlazený obraz, tím méně nalezneme hran). Tento algoritmus je pomalejší než předešlá metoda, ale může dosahovat výrazně lepších výsledků.

3 Detekce, rozpoznávání a analýza jednoduchých geometrických tvarů

Tato část bude věnována především detekci jednoduchých geometrických tvarů. Bude probráno několik možných přístupů a jejich klady a zápory. V praktické části této práce pak budou vybrány některé přístupy, implementovány a následně otestovány, obzvláště na časovou náročnost a přesnost detekce. Na základě výsledků testů bude následně zvolen vhodný kandidát pro výslednou implementaci aplikace.

3.1 Úvod do problematiky

[6] Důležitou součástí vnímání v umělé inteligenci je rozpoznávání tvarů ve statických scénách. Při výzkumu této problematiky bylo vždy úzce vycházeno z biologických procesů rozpoznávání nalezených v různých organismech. Z tohoto důvodu existuje několik přístupů k detekci geometrických tvarů:

- Z kontur
- Ze stínů
- Z textur (z povrchů)
- Ze stereografie
- Z fraktální geometrie

Zaměření této části bude především na detekce obrazců z kontur.

Ke stanovení existence nám zajímavému tvaru v obrazu je typicky nutná forma, či představa použitelná k porovnávání objektů. K analýze obrazů z kontur se klasicky přistupuje dvěma různými způsoby – popis tvaru nebo tvarovou reprezentací. Při popisu obrazu se původní tvar transformuje na číselnou reprezentaci nejdůležitějších vlastností. Reprezentace tvarů naopak pro popis používá nečíselné reprezentace klíčových charakteristik. Kromě tohoto rozdělení technik se nabízí ještě jedno – analýza na základě okrajů čí i na základě vnitřních vlastností objektu (globální vlastnosti tvaru). Techniky analýzy okrajů se soustředí především na kontury objektů, zatímco techniky, které berou v potaz i ostatní vlastnosti objektů, se zabývají i vnitřním obsahem a vlastnostmi tvaru. Kombinací tohoto rozdělení získáme čtyři třídy rozdělení přístupů analýzy tvarů:

- Boundary Scalar Techniques - číselný popis okrajů tvarů
- Boundary Space-Domain Techniques - popis okrajů tvaru nenumerickými deskriptory
- Global Scalar Techniques - číselný popis globálních vlastností tvaru
- Global Space-Domain Techniques - popis globálních vlastností tvaru pomocí nenumerických deskriptorů

3.2 Boundary Scalar Techniques

[6] Typicky jsou tvořeny dvěma kroky. Prvním je vytvoření jedno-dimenzionální funkce z okraje tvaru, která je použita v druhém kroku k popisu dvoudimenzionálního okraje tvaru. V druhém kroku jsou použity techniky založené na Fourierově transformaci charakteristické funkce nebo na stochastickém procesu modelování charakteristické funkce.

3.2.1 Zisk jedno-dimenzionální funkce

Pro tento problém existuje několik přístupů.

Vzdálenost centroidu a okraje

Určení centroidu a zaznamenávání vzdálenosti k okrajovým bodům vhledem k úhlu u centroidu. Okrajové body můžeme zvolit, tak, že budou od sebe stejně vzdálené nebo můžeme volit konstantní úhel u centroidu.

Výpočet zakřivení okrajových bodů

Zaznamenávání změn tečných úhlů.

Funkce otáčení

Funkce tečného úhlu vzhledem k délce oblouku.

Komplexní funkce

$x(t) \ + \ j_y(t) \ t...délka \ oblouku.$

3.2.2 Popis pomocí Fourierových transformací

Na charakteristickou funkci se použije Fourierova transformace k převedení frekvenci, která pak charakterizuje daný tvar. Tento přístup ovšem funguje špatně, pokud se v obrazu nachází šum. Výsledek lze zlepšít pomocí zahlazení kontur (odstranění vysokých frekvencí), to má ale ovšem za následek ztrátu komplexnějších tvarů.

3.2.3 Popis pomocí Stochastických procesů modelování

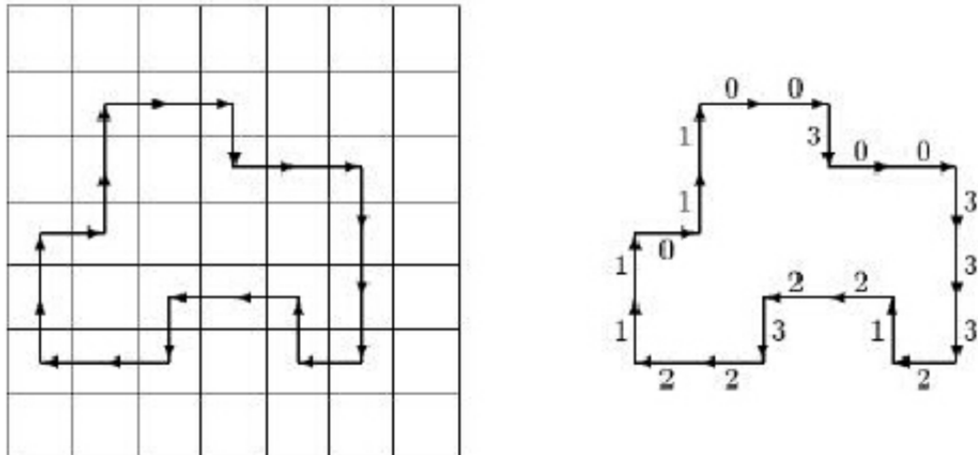
Na charakteristickou funkci jsou použity autoregresivní techniky pro zjištění parametrů modelu, které jsou poté použity pro popis tvaru.

3.3 Boundary Space-Domain Techniques

[6] Cílem těchto technik je vytvoření obrázkových, či grafických reprezentací hranice tvaru. Nejvíce používanými reprezentacemi jsou – řetězové kódy, syntaktické techniky, aproximace hranic, změny velikosti prostoru.

3.3.1 Freemanovy řetězové kódy

Hranice tvaru je kódována jako sekvence spojitých úseček dané délky a jejich směru. Nejčastěji se kóduje ve 4 směrech nebo v 8 směrech. Tato technika má ovšem několik problémů – u komplexních tvarů jejich délka výrazně roste a to obzlášť pokud se v obrazu nachází šum = málo odolný proti šum. Oba tyto problémy lze vyřešit zvětšením délky úseček, což ovšem značně zhorší kvalitu. Dalším problémem je také to, že tvar není detekován, pokud je otočen. To lze ovšem vyřešit tak, že se místo řetězového kódu budou používat rozdíly mezi jednotlivými úsečkami. Příklad v obrázku 3.1.



Obrázek 3.1: Ukázka zakódování hranice obrazu pomocí Chain coding. Algoritmus začal z horní levé části hranice a vytvořil chain code 0 ,0, 3, 0, 0, 3, 3, 3, 2, 1, 2, 2, 3, 2, 2, 1, 1, 0, 1, 1. Obrázek převzat z [6, Figure 6]

3.3.2 Syntaktické metody

Základním principem této metody je rozložení tvarů na co nejmenší součásti, které pak pojmenujeme nějakým řetězcem (slovem). Poté vytvoříme sadu pravidel, jak používat tyto slova (gramatika). Tímto způsobem tedy vytvoříme jakýsi jazyk, pomocí něhož můžeme popsat jakýkoliv tvar. Největším problémem tohoto přístupu je vyextrahování jednotlivých slov, což je dosti náročný proces.

3.3.3 Scale-Space techniky

Základní myšlenkou je, že pokud se na tvar aplikuje dostatečný počet Gaussovských filterů (při zvětšování a zmenšování tvaru) zůstanou jen ty nejvíce podstatné body. Ty se pak použijí na popis tvaru.

3.3.4 Aproximace hranic

Používají se polygoniální a spline aproximace. Cílem je minimalizovat aproximační chybu a maximalizovat obsah aproximované plochy. Příkladem je algoritmus Douglas – Peucker.

3.4 Algoritmus Douglas-Peucker

3.4.1 Popis

[9] Jedná se o algoritmus pro zjednodušování čar, extensivně používaný jak v počítačové grafice, tak v geografických informačních systémech. Je uznáván především za jeho výbornou reprezentaci původních přímek a pro jeho jednoduchou implementaci. Existují 2 varianty – originální, se složitostí $O(nm)$ (n – množství vstupních vrcholů,m – množství výstupních segmentů křivky), který funguje v jakékoliv dimenzi, a jeho rychlejší verze se složitostí $O(n\ log\ m)$, který funguje jen pro jednoduché rovinné křivky ve 2D.

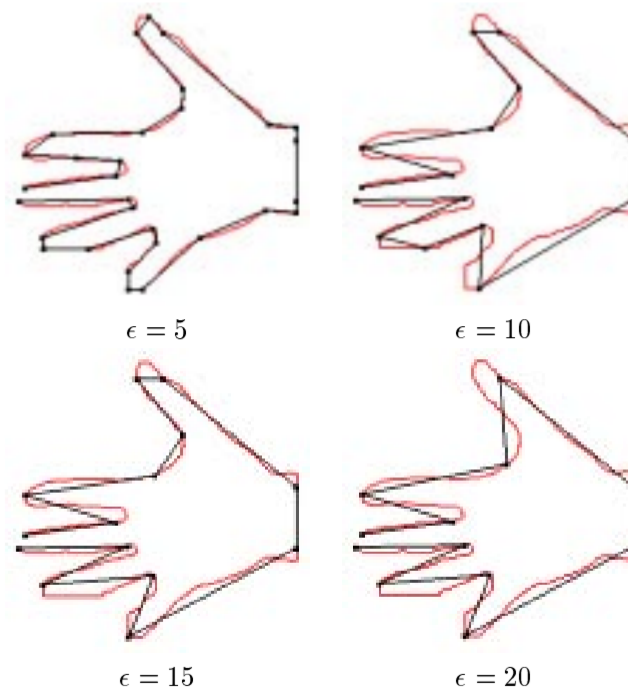
Douglas-Peuckerův algoritmus je schopný operovat ve všech dimenzích, jelikož mu záleží pouze na vzdálenosti jednotlivých bodů a čar. Základním předpokladem je, že aproximace musí obsahovat podmožinu bodů z originálních dat a že všechny originální body musí ležet v předefinované vzdálenosti od aproximace.

3.4.2 Postup

[9] Algoritmus postupuje hiearchicky. Začne spojením prvního a posledního bodu křivky. Poté jsou otestovány zbývající vrcholy podle vzdálenosti ke vzniklé hraně. Pokud se naleznou vrcholy vzdálenější než přednastavená vzdálenost e, nejvzdálenější z nich je přidán do aproximace. Tento postup se rekurzivně opakuje pro všechny strany, dokud se všechny originální body nenachází dál než-li e.

3.4.3 Závěr

[9] Algoritmická složitost je $O(mn)$, tím pádem je tento algoritmus výstupově závislý, tzn. pro hrubé aproximace funguje velice rychle. Na druhou stranu, pokud je e velké může dojít k tomu, že aproximovaná křivka protne sama sebe. Ukázka zjednodušení čar v obrázku 3.2.



Obrázek 3.2: Ukázka výsledku algoritmu Douglas-Peucker s různými tolerancemi e. Obrázek převzat z [9, Figure 8]

3.5 Sledování hranic tvarů (Border Following)

3.5.1 Popis

[8] Sledování hranic tvarů je jedna ze základních technik zpracování binárních obrazů. Extrahuje sekvence souřadnic z okrajů mezi spojenými komponentami námi hledaných objektů (1-pixel komponenty) a okolím (0-pixel komponenty). Tento process má široké využití např. v rozpoznávání a analýze obrazu či obrazové kompresi.

3.5.2 Definice

[8] V tomto textu uvažujeme se souřadnicovým systémem (i, j), kde i představuje číslo řady (z leva do prava) a j představuje číslo sloupce (ze shora dolů). Abychom předešli topologické kontradikci, budeme uvažovat 0-pixely jako osmipropojené (8-connected – tzn. pixely jsou propojeny, pokud se nachází po stranách svých stranách (vpravo, vlevo, nahoře, dole) a nebo na rozích) a 1-pixely jako čtyřpropojené (4-connected – tzn. jsou propojeny, pokud se nachází po svých stranách).

Rám obrazu

Nejvyšší a nejnižší řada a nejvíce vlevo a nejvíce vpravo sloupec obrazu.

Pozadí

0-komponenta, která obsahuje rám obrazu.

Krajní bod

Pokud 1-pixel má ve svém 8-okolí 0-pixel, nazývá se krajním bodem.

Obklopení jedné komponenty druhou

Pro 2 komponenty S_1 a S_2, pokud existuje pixel náležící S_2 ve 4-okolí pixelu náležícímu S_1 a rámu obrazu, tak můžeme říct, že S_2 obklopuje S_1. Pokud S_2 obklopuje S_1 a existuje mezi nimi krajní bod, tak S_2 obklopuje S_1 přímo.

Vnější okraj a okraj díry

Vnější okraj je definován jako množina krajních bodů libovolné 1-komponenty a 0-komponetry, která jí přímo obklopuje. Okraj díry je přímý opak, tedy množina krajních bodů 0-komponentry (díry) a 1-komponentry, která jí přímo obklopuje. Pojmem okraj lze referovat jak na vnější okraj, tak na okraj díry.

Pro okraje a propojené komponenty platí

Pro libovolnou 1-komponentu existuje právě jeden vnější okraj. Tato vlastnost platí I pro libovolnou díru a její okraj. Tato vlastnost platí, protože každá 1 – komponenta nebo díra je přímo obklopena další propojenou komponentou.

Parent border

Parent border vnějšího okraje mezi 1 – komponentou S_1 a 0-komponentou S_2, který přímo obklopuje S1 je definován jako:

- Okraj díry mezi S_2 a 1-komponentou, která přímo obklopuje S_2, pokud je S_2 dírou.
- Rám obrazu, pokud je S_2 pozadí.

Parent border okraje díry mezi dírou S_3 a 1-komponentou S_4, která přímo obklopuje S_3 je definována jako vnější okraj mezi S_4 a 0-komponentou, která přímo obklopuje S_4.

Obklopení jednoho okraje druhým

Pro dva okraje B_0 a B_n z binárního obrazu, můžeme říct, že B_n obklopuje B_0, pokud existuje sekvence okrajů B_0, B1, . . . , B_n pro kterou platí, že B_k^{-1} je rodičovský okraj B_k pro všechna k z intervalu <1, n>

Pro jakýkoliv binární obraz tedy platí, že obě obklopení jsou isomorfní pro následující mapování:

- 1-komponenta <-> její vnější okraj
- Díra <-> její okraj díry mezi dírou a 1-komponentou, která jí přímo obklopuje
- Pozadí <-> rám obrazu

Po ujasnění těchto termínů a vlastností můžeme definovat dva algoritmy využívající techniku border following: 1. Určený pro topologickou analýzu obrazu 2. Pro extrahování nejkrajnějších okrajů

3.5.3 První algoritmus

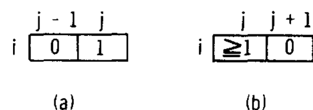
[8] Skenuj vstupní binární obraz pomocí televizního rastrování a přeruš ho, pokud je nalezen pixel (i, j), který splňuje podmínky pro počáteční bod vnějšího okraje (Obrázek 3.3a), či okraje díry(Obrázek 3.3b). Pokud splňuje obě tyto podmínky, jedná se o počáteční bod vnějšího okraje. Přiřaď k okraji unikátní číslo. Toto číslo nazveme sekvenčním číslem tohoto okraje. Poznamenej si jeho okolí (dale NBD).

Zjisti rodičovský okraj nově nalezeného okraje podle následujících instrukcí – Během rasterového skenování si zaznamenáváme také sekvenční číslo LNBD naposledy nalezeného okraje. Tento zapamatovaný okraj by měl být buď rodičovský okraj nově nalezeného okraje a nebo okraj, který sdílí rodičovský okraj s nově nalezeným okrajem. Sekvenční číslo rodičovského okraje tedy rozhodneme podle rozhodovací tabulky (Obrázek 3.4).

Následuj nalezený okraj od počátečního bodu a označuj jeho pixely podle následujících pravidel:

- Pokud je momentálně následovaný okraj mezi 0-komponentou obsahující pixel $(p,\ q\ +\ 1)$ a 1-komponentou obsahující pixel $(p,\ q)$, změň hodnotu pixelu $(p,\ q)$ na –NBD
- Jinak změň hodnotu na NBD, pokud není pixel $(p,\ q)$ již součástí následovaného okraje

Poté co označíš celý okraj, pokračuj s rasterovým skenem.



Obrázek 3.3: Podmínky pro počáteční bod vnějšího okraje a okraje díry. Převzato z [8, Fig. 2]

Decision Rule for the Parent Border of the Newly Found Border B

Type of B \ Type of the border B' with the sequential number LNBD	Outer border	Hole border
Outer border	The parent border of the border B'	The border B'
Hole border	The border B'	The parent border of the border B'

Obrázek 3.4: Rozhodovací tabulka pro Border Following. Převzato z [8, Table 1]

3.5.4 Druhý algoritmus

[8] Druhý algoritmus je modifikovanou verzí algoritmu prvního, tak aby následoval pouze nejkrajnější okraje binárního obrazu (tedy okraje mezi pozadím a 1-komponentami).

Následování okrajů začínáme pouze v bodech, kde je splněna pouze podmínka pro vnější okraj (Fig. 2a) a LNBD ≤ 0.

Značení okrajů je stejné jako při prvním algoritmu s výjimkou, že hodnoty NBD a –NBD jsou nahrazeny hodnotami 2 a -2.

Uchováváme si hodnotu LNBD posledního nenulového pixelu během raster skenu. Pokaždé když začneme skenovat novou řadu, LNBD resetujeme na 0.

4 Existující řešení

4.1 Klasický přístup

Článek [5] popisuje klasický přístup k našemu problému. Robot sleduje čáru pomocí dvou diod a světelného senzoru. Tmavé barvy pohlcují více světla. Pokud tedy dioda posvítí na černý povrch, světelný senzor zaznamená změnu a určí, že je robot na čáře. Tyto senzory jsou 2, tím pádem, pokud jeden ze senzorů nedetekuje černou barvu, robot se otočí za tím senzorem, který na čáře je.

4.2 Vizuální přístup

Článek [4] přistupuje k našemu problému podobně jako my. Robot dokáže následovat bílou čáru na tmavě zeleném pozadí. K tomu používá web kameru. Obraz si napřed předzpracuje a poté rozdělí na bloky. O pohybu rozhoduje dle poměru bloků, obsahujících bílou barvu, na pravé a levé straně.

5 Analýza a návrh

API bude implementováno v jazyku Java a optimalizován pro platformu Android. API bude rozděleno na dvě, na sobě nezávislé části. Jedna se bude zabývat sledováním čáry, což je proces rozložitelný na 3 části – jasová úprava vstupního obrazu, extrakce hran čáry a analýza čáry pro převedení těchto znalostí na příkazy k pohybu podvozku. Druhá část API se bude zabývat rozpoznáváním jednoduchých geometrických tvarů a barev. Na ty bude možné navázat různé akce, jako změna rychlosti vozítka, změna směru pohybu, atd.

5.1 Sledování čáry

Jasová úprava vstupního obrazu bude vyřešena pomocí techniky ekvalizace histogramu. Ta rozprostře jasové hodnoty vstupního obrazu, čímž zesvětlí tmavá místa v obrazu při zachování kontrastu. Díky tomu v dalších krocích budou lépe rozpoznatelné detaily objektů.

5.2 Extrakce hran čáry

Budou implementovány 3 algoritmy, navržené pro řešení toho problému - Sobelův algoritmus, LoG (Laplacian of Gaussian) algoritmus a vymaskování černé barvy. Ty budou otestovány na přesnost a rychlost detekce čáry a to v různých světelných podmínkách a pro různé druhy čáry. Bude pak vybrán algoritmus, který bude dobrým kompromisem mezi rychlostí a přesností detekce hran čáry.

5.3 Analýza získané čáry

Tento problém má několik řešení. Nejjednodušší by bylo spočítat počet pixelů na levé straně obrazu a na pravé straně. Elegantnějším řešením, které by mělo umožňovat plynulý pohyb podvozku po čáře je vypočítání těžiště získané čáry. Čím více bude těžiště na jedné straně obrazu, tím agresivnější bude podvozek zatáčet.

5.4 Rozpoznávání geometrických tvarů a barev

Pro tento problém budou implementováno řešení založené na kombinaci dvou algoritmů – Border Following algoritm pro získání kontůr a algoritmus Douglas-Peucker pro nalezení počtu hran tvaru, díky čemuž bude možné identifikovat tvar – např. čtverec má 4 strany, trojúhelník 3.

6 Implementace

6.1 OpenCV

Po několika pokusech vlastní implementace různých algoritmů se mi nepodařilo na platformě Android dosáhnout dostatečné rychlosti zpracování obrazu nutnou ke správnému běhu aplikace. Z tohoto důvodu jsem se obrátil na knihovnu OpenCV[1]. Tato knihovna specializující se na zpracování obrazu, obsahuje implementaci většiny nejvíce používaných algoritmů. Knihovna používá paralelizaci a také využívá nativní podpory platformy Android. Při implementaci jsem využíval verzi 3.4.10.

6.1.1 Použité knihovní funkce

Imgproc.equalizeHist()

Metoda se v api využívá k vyrovnání bílé a přitom zachování všech detailů z původního obrazu. Tohoto dosahuje normalizací histogramu jasu obrazu. Algoritmus napřed vytvoří histogram obrazu. Poté dojde k jeho normalizaci a to tak, aby se součet všech intervalů rovnal hodnotě 255. Následovně se spočte integrál histogramu a pomocí něho dojde k transformaci původního obrazu.

Imgproc.Sobel()

Metoda je v api používána k detekci hran v obraze. Tohoto dosahuje konvolučním maskováním Sobelovo operátorem (3x3 konvoluční maska. Viz kapitola 2.2.1) původního obrazu.

Imgproc.Laplacian()

Metoda je také používána k detekci hran. Tohoto výsledku dosahuje součtem 2 druhých derivací původního obrazu získaných pomocí Sobelova operátoru. Viz kapitola 2.2.2.

Imgproc.cvtColor()

Metoda v api slouží k převádění původního obrazu do požadovaného barevného prostoru (např. z BGR na HSV, nebo barevného obrazu na šedý)

Imgproc.GaussianBlur()

Metoda je používána k eliminaci šumu v obrazu. Tohoto dosahuje rozmazáním obrazu pomocí Gaussova filtru – konvoluční maskování Gaussovským operátorem.

Imgproc.threshold()

Metoda je používána k vymaskování černé barvy z obrazu při následování černé čáry. Toho dosahuje vytvořením binárního obrazu z obrazu šedého odstraněním všech pixelů, které jsou menší, než určený práh.

Imgproc.erode()

Slouží v API k odstranění šumu ze zpracovávaného obrazu. Toho dosahuje opakovanou erozí obrazu pomocí určeného členícího elementu.

Imgproc.dilate()

Slouží v API k opravě erodovaného obrazu při odstraňování šumu ze zpracovávaného obrazu.

Core.inRange()

Funkce je používána při maskování určité barvy ze zpracovávaného obrazu při detekci geometrických obrazů. Toho dosahuje vytvořením binárního obrazu z obrazu původního odstraněním všech pixelů, které neodpovídají určenému barevnému intervalu.

Imgproc.findContours()

Funkce slouží k nalezení kontur z binárního obrazu při následování čáry a detekci geometrických tvarů. Toho dosahuje použitím border following algoritmu (viz kapitola 3.5).

Imgproc.approxPolyDP()

Slouží k aproximaci hran geometrických tvarů z kontur. Podle jejich počtu je pak určen typ geometrického tvaru (3 strany = trojúhelník, 4 strany = obdélník atd.). Toho dosahuje pomocí Douglas-Peuckerova algoritmu. Viz kapitola 3.4.

Imgproc.contourArea()

Je používána při detekci geometrických tvarů k odfiltrování příliš malých kontur. Vrací obsah kontury v pixelech.

Imgproc.boundingRect()

Slouží k určení souřadnic zkoumané kontury. Metoda vyhledává nejmenší možný obdélník, který dokáže obklopit zkoumanou konturu. V API se používá pro zisk souřadnic detekovaného geometrického tvaru.

6.2 API

Součástí zadání je vytvoření API, aby šlo snadno vytvořit svojí aplikaci na ovládání vozidla.
API se skládá ze tří tříd – LineFollowing, ShapeDetection a ShapeMessenger, které se starají o zpracování obrazu, a ze dvou rozhraní - IVehicleController a IShapeDetectionReactionController. Uživateli API stačí k funkční aplikaci pouze implementovat tyto dvě rozhraní, které vloží do dvou výše zmiňovaných tříd a implementovat rozhraní
CameraBridgeViewBase.CvCameraViewListener2 z knihovny OpenCV pro ovládání kamery ve své hlavní aktivitě.

6.2.1 IVehicleController

Rozhraní slouží k implementaci komunikace aplikace s vozidlem. Tímto způsobem lze API použít téměř s jakýmkoliv vozidlem. Rozhraní se skládá z 10 metod.

void stop()

Metoda slouží k zastavení vozidla. Pokud je vozidlo zastaveno, nemělo by docházet ke komunikaci aplikace s vozidlem, nehledě na to, co API detekuje.

void start()

Metoda slouží k rozpohybování vozidla

boolean isStopped()

Vrací true, pokud je vozidlo zastaveno metodou stop() nebo false pokud se vozidlo normálně pohybuje.

void turnLeft()/turnLeft(double powerPercentage)

Metoda slouží k zaslání příkazu k odbočení vlevo. Pomocí proměnné powerPercentage lze určit procento výkonu, které chceme k tomuto úkonu použít.

void turnRight()/turnRight(double powerPercentage)

Metoda slouží k zaslání příkazu k odbočení vpravo. Pomocí proměnné powerPercentage lze určit procento výkonu, které chceme k tomuto úkonu použít.

void forward()/void forward(double powerPercentage)

Metoda slouží k zaslání příkazu k pohybu vpřed. Pomocí proměnné powerPercentage lze určit procento výkonu, které chceme k tomuto úkonu použít.

void control(double leftPowerPercentage, double rightPowerPercentage)

Metoda slouží k zaslání příkazu k pohybu vozidla s tím, že si uživatel metody určí procento výkonu pro levý a pravý motor.

6.2.2 IShapeDetectionReactionController

Rozhraní slouží k tomu, aby si uživatel API mohl sám určit reakce na detekci různých geometrických tvarů. Rozhraní se skládá pouze z jedné metody:

void objectsDetected(List<ShapeMessenger> listOfDetectedShapes)

Metoda je zavolána pokaždé, když dojde k dokončení zpracování obrazu v metodě shapeDetection() třídy ShapeDetection. Uživatel si takto může zvolit reakci na základě počtu a pozice detekovaných geometrických tvarů.

6.2.3 ShapeMessenger

Tato třída je implementována dle návrhového vzoru Přepravka. Slouží k přenosu informace o detekovaném objektu třídou ShapeDetection a to konkrétně jeho typ (využívá výčtu Shapes z třídy ShapeDetection) a souřadnice, kde se objekt ve zpracovávaném obraze nachází. Třída se skládá z konstruktoru a tří getterů a setterů.

6.2.4 ShapeDetection

Tato třída obsahuje algoritmus pro detekci geometrických tvarů. Uživatel API si zvolí jednu ze tří základních barev – červená, zelená, modrá. Pokud dojde k detekci některého ze základních geometrických tvarů (trojúhelník, obdélník, kruh), jsou získány jeho souřadnice, je vytvořen objekt třídy ShapeMessenger a je uložen do seznamu detekovaných objektů. Po zpracování celého obrazu je seznam předán ke zpracování třídě implementující rozhraní IShapeDetectionReactionController. Třída se skládá ze dvou výčtů (enum), kontruktoru, dvou metod a tří setterů:

enum Colors

Slouží k výběru jedné ze tří barev – červené, zelené nebo modré.

enum Shapes

Slouží k identifikaci geometrického tvaru - trojúhelník, obdélník a kruh.

ShapeDetection(IShapeDetectionReactionController controller, Colors color)

Uživatel při inicializaci třídy si určí barvu geometrických tvarů k detekci a vloží třídu implementující rozhraní IShapeDetectionReactionController, aby mohlo API reagovat na detekci geometrických tvarů určené barvy. Pro tyto dva parametry ještě existují settery.

void shapeDetection(CameraBridgeViewBase.CvCameraViewFrame inputFrame, boolean lowerResolution)

Uživatel tuto metodu vloží do metody onCameraFrame() z rozhraní CameraBridgeViewBase.CvCameraViewListener2 a předá do ní přijatý obraz. Uživatel si také může zvolit použití nižšího rozlišení pro zvýšení rychlosti zpracování obrazu.
Metoda si napřed získá z obrazu jeho barevnou verzi a pak jí převede do barevného prostoru HSV. Podle určené barvy se pak tento obraz vymaskuje – přemění se na binární obraz, kdy se v obrazu nechají pouze oblasti, které obsahují definovanou barvu.
Z této masky jsou pak vyextrahovány kontury. Každá kontura je pak následovně zpracována. Pokud má kontura menší obsah jak 400 pixelů, předpokládá se, že jde o chybnou detekci a není dále zpracována.
Kontura je aproximována algoritmem Douglas-Peucker. Získá se počet stran potencionálního tvaru. Pokud je počet aproximovaných stran roven 3 je vytvořen objekt třídy ShapeMessenger reprezentující trojúhelník. Pomocí metody z knihovny OpenCV Imgproc.boundingRect() jsou získány souřadnice středu nejmenšího možného obdélníka, který je schopný obklopit konturu detekovaného trojúhelníka. Na tyto souřadnice v původním obraze je nakreslen symbol trojúhelníka a zároveň jsou souřadnice předány do objektu ShapeMessenger. Objekt je pak uložen do seznamu.
Pokud počet aproximovaných stran je roven 4 je vytvořen objekt třídy ShapeMessenger reprezentující obdélník. Pomocí metody z knihovny OpenCV Imgproc.boundingRect() jsou získány souřadnice středu nejmenšího možného obdélníka, který je schopný obklopit konturu detekovaného obdélníka. Na tyto souřadnice v původním obraze je nakreslen symbol obdélníka a zároveň jsou souřadnice předány do objektu ShapeMessenger. Objekt je pak uložen do seznamu.
Pokud je počet aproximovaných stran mezi 10 a 20 je vytvořen objekt třídy ShapeMessenger reprezentující kruh. Pomocí metody z knihovny OpenCV Imgproc.boundingRect() jsou získány souřadnice středu nejmenšího možného obdélníka, který je schopný obklopit konturu detekovaného kruhu. Na tyto souřadnice v původním obraze je nakreslen symbol kruhu a zároveň jsou souřadnice předány do objektu ShapeMessenger. Objekt je pak uložen do seznamu.
Po zpracování všech detekovaných kontur je seznam předán třídě implementující IShapeReactionController. Tímto způsobem může algoritmus reagovat na více objektů v jednom obraze najednou a má k dispozici i jeho souřadnice.

ShapeMessenger drawMarkOfShape(MatOfPoint contour, Mat result, Shapes shape, Colors color, boolean lowerResolution)

Metoda se stará o zisk souřadnic detekovaného objektu, zakreslení symbolu, který ho reprezentuje do předaného obrazu a vytvoření objektu třídy ShapeMessenger s těmito informacemi.

6.2.5 LineFollowing

Třída je zodpovědná za veškerou funkcionalitu týkající se detekce čáry, její analýzy a následné rozhodnutí o pohybu vozidla, aby jí následovalo. K dispozici je několik algoritmů, které lze zkombinovat do několika různých přístupů k problému.

Popis implementovaných algoritmů

Detekce hran
: K dispozici jsou dva algoritmy a to konkrétně Sobelův a Laplacian of Gaussian detektor hran.

Maskování černé barvy
: Jako alternativa k detekci hran je k dispozici algoritmus k maskování černé barvy. Algoritmus vytvoří binární obraz z tmavých pixelů původního obrazu.

Porovnávání jasu
: Jako poslední varianta ke zpracování obrazu je implementován algoritmus porovnávání jasu dvou oblastí na opačných stranách obrazu. Uživatel si zvolí procento šířky obrazu z prostředka, kterou nechce započítávat do detekce (aby odpovídala šířce čáry, kterou chce uživatel následovat) a procento tolerance rozdílu mezi oblastmi. Algoritmus pak uvažuje dvě oblasti v horní polovině obrazovky, v nich spočítá průměrný jas pixelů. Pokud rozdíl přesáhne toleranci, tak na stranu s menším průměrným jasem vozidlo odbočí.

Těžiště detekované čáry
: Tento přístup rozhodování o směru pohybu vozidla lze spojit jak s oběma detektory hran, tak s maskováním černé barvy. Podle toho, na jaké straně osy x se nachází těžiště zpracovávaného obrazu, na té straně poklesne výkon motoru o tolik procent, jak daleko od počátku osy x se těžiště nachází.

Suma pixelů na stranách zpracovávaného obrazu
: Stejně jako při hledání těžiště, lze tento algoritmus spojit s oběma detektory hran i s maskováním černé barvy. Algoritmus si rozdělí obraz do 4 sektorů a začne počítat bílé pixely, přičemž pixely nacházející se v horní polovině obrazu mají váhu 4 a v dolní 1. Vozidlo odbočí na tu stranu, kde se nachází více bílých pixelů.

Popis implementace třídy

Při inicializaci třídy je nutné objektu předat objekt implementující rozhraní IVehicleController, vybrat algoritmy pro detekci čáry a řízení směru vozidla a určit rozměry zpracovávaného obrazu. OpenCV ze základu předává obraz z fotoaparátu otočený na bok. Pokud se uživatel rozhodne toto chování změnit, musí se také při inicializaci nastavit, že obraz je otočený. Na všechny tyto parametry třída obsahuje setry i getry. Třída se dále skládá z 2 výčtů a 7 metod:

enum LineDetectionAlgorithms

Slouží k výběru algoritmu pro detekci čáry. Uživatel si může vybrat mezi následujícími možnostmi – Sobel, Laplacian, Black Color Masking, None (v případě přístupu porovnávání jasu).

enum DirectionDecisionAlgorithms

Slouží k výběru algoritmu pro rozhodování o směru pohybu vozidla. Uživatel si může vybrat mezi následujícími možnostmi – Center of Gravity, Weighted Pixel Counting, Brightness Comparison.

Mat followLine (CameraBridgeViewBase.CvCameraViewFrame inputFrame, boolean edgeDetectionOn, boolean directionDetectionOn, boolean whiteBalanceOn, boolean vehicleControlOn, boolean drawDirectionIndication)

Uživatel tuto metodu vloží do metody onCameraFrame() z rozhraní CameraBridgeViewBase.CvCameraViewListener2 a předá do ní přijatý obraz. Uživatel si může zvolit, pokud chce zapnout/vypnout vyvažování bílé, detekci čáry, směrové rozhodování, kontrolu vozidla a nebo pokud chce, aby na vráceném obrazu byla nakreslena indikace směru pohybu vozidla. Metoda si nejprve získá z předaného obrazu jeho šedou variantu. Poté dojde k vyrovnání jasu. Následně je provedena detekce čáry dle vybraného algoritmu a je rozhodnuto o směru pohybu. Tato informace je pak předána metodě na ovládání vozidla, příslušící vybranému směrovému rozhodovacímu algoritmu. Nakonec je na obraz nakreslena indikace směru pohybu vozidla (šipka pro algoritmy Weighted Pixel Counting a Brightness Comparison, kružnice zobrazující centroid pro algoritmus Center of Gravity).

int weightedPixelCounting(Mat mat)

Metoda si předaný obraz převede na pole bytů k rychlejšímu přístupu k jednotlivým pixelům. Poté metoda postupuje, tak, jak je popsáno výše (viz Suma pixelů na stranách zpracovávaného obrazu).

int[] centerOfGravity(Mat mat)

Metoda si předaný obraz převede na pole bytů k rychlejšímu přístupu k jednotlivým pixelům. Poté provede průměr souřadnic x a y všech pixelů s hodnotou 255. Výsledkem jsou souřadnice těžiště, které metoda vrátí v podobě dvouprvkového pole.

Mat brightnessComparison(boolean vehicleControlOn, boolean drawDirectionIndication, Mat src)

Metoda si předaný obraz převede na pole bytů k rychlejšímu přístupu k jednotlivým pixelům. Metoda poté následuje postup popsaný výše (viz Porovnávání jasu)

void controlVehicleCOG(double x)

Metoda vypočítá jak daleko od středu osy x zpracovávaného obrazu se nachází předané těžiště. Tuto informaci převede na procenta. Metoda pak pomocí třídy implementující rozhraní IVehicleController pak vydá příkaz vozidlu, aby snížilo výkon motoru na té straně, kde se nachází těžiště.

Void controlVehicle(int direction)

Metoda pomocí třídy implementující rozhraní IVehicleController vydá příkaz vozidlu, k pohybu ve směru dle předané hodnoty – 0 toč vpravo, 1 toč vlevo, 2 pohybuj se vpřed.

6.3 Aplikace

Ukázková aplikace byla implementována tak, aby bylo možné ukázat a otestovat všechny aspekty API. Aplikace obsahuje nastavení, kdy si uživatel může zvolit jakýkoliv z implementovaných přístupů k následování čáry, zapnout/vypnout svítilnu na fotoaparátu, zapnout/vypnout vyrovnávání jasu a nebo si zvolit jaký tvar a barvu bude aplikace detekovat. V uživatelském rozhraní se zobrazuje zpracovaný obraz, indikace směru pohybu vozidla, aktuální výkon motorů v procentech a momentálně detekovaný tvar, či pokud byla navolena obecná detekce tvarů počet stran geometrického tvaru.
Aplikace se skládá ze 7 tříd:

6.3.1 MainActivity

Hlavní třída celé aplikace. Jedná se o potomka třídy AppCompatActivity, což je základní třída pro aktivitu z podpůrné android knihovny AndroidX. Třída dále ještě implementuje rozhraní
CameraBridgeViewBase.CvCameraViewListener2, které zajišťuje práci s fotoaparátem.
Jelikož se jedná o potomka třídy AppCompatActivity, je součástí klasického Android životního cyklu aktivity. Tzn. že obsahuje 4 metody, které jsou volány dle stavu, ve kterém se aktivita nachází. Třída také obsahuje 3 metody rozhraní CameraBridgeViewBase.CvCameraViewListener2. Nakonec třída obsahuje 2 metody a Handler. Tato část kódu zajišťuje komunikaci mezi smartphonem a ovládaným vozidlem skrz USB připojení. Tato část kódu a třídy USBService a USBVehicleController byly implementovány s pomocí volně dostupné knihovny UsbSerial verze 4.5 od nezávislého tvůrce felHR85[1].

void onCreate()

Tato metoda je volána při vzniku aktivity. V metodě se vytvoří tlačítko pro otevření aktivity nastavení, a přiřadí se mu listener. Dále se uloží odkazy na textová pole pro zobrazování výkonů motorů a detekovaného tvaru. Nastaví se, aby se na uživatelském rozhraní zobrazovaly námi zpracované obrazy z fotoaparátu a dojde k inicializaci tříd USBVehicleController a ShapeReactionController.

[1] `https://github.com/felHR85/UsbSerial`

void onPause()

Tato metoda je volána při pozastavení aktivity, např. když dojde k přepnutí na aktivitu jinou. Zde akorát zajistíme, aby se pozastavila funkce fotoaparátu a zastavilo se ovládané vozidlo.

void onResume()

Tato metoda je volána při znovuspuštění aktivity. Zde se zajistí načtení knihovny OpenCV a dojde k změně třídních atributů na základě uživatelského nastavení a dojde k nastolení spojení s ovládaným vozidlem pomocí USB. Také dojde k nastartování ovládaného vozidla.

void onRequestPermissionResult()

Metoda je volána, když je jádro operačního systému žádáno o povolení k nějaké akci.

void onCameraViewStarted()

Tato metoda je volána, když začne snímání obrazů z fotoaparátu. V této metodě se nastaví všechny atributy tříd VehicleController a ShapeReactionController dle uživatelského nastavení a dojde k zapnutí/vypnutí svítilny mobilního zařízení.

void onCameraViewStopped()

Tato metoda je volána, když se z jakéhokoliv důvodu zastaví snímaní obrazů z fotoaparátu.

void onCameraFrame()

Tato metoda je zavolána, když je připraven obraz z fotoaparátu. Zde také probíhá veškeré naše zpracování obrazu, tzn. nachází se zde volání metod shapeDetection() a followLine() z API. Metoda vrací buď obraz zpracovaný detekcí čáry anebo obraz zpracovaný detekcí geometrických tvarů (dle volby uživatele). Výsledek této metody je zobrazen na uživatelském rozhraní.

void startService(ServiceConnection serviceConnection)

Metoda slouží ke startu služby zajišťující USB komunikace.

void setFilters()

Metoda slouží k poslechu příchozích notifikací z USB spojení.

static class MyHandler

Třída slouží jako spojení hlavní aktivity a třídy USBService. Pokud třída USBService přijme zprávu z USB, je tomuto Handleru předána a zalogována.

6.3.2 Settings

Jedná se o vedlejší aktivitu, která zajišťuje funkcionalitu uživatelského nastavení. Tato aktivita je vyvolána, když je v uživatelském rozhraní stisknuto tlačítko s opravářským klíčem. Jako MainActivity je potomkem třídy AppCompatActivity.
Obsahuje metodu void onCreate(), ve které akorát dochází k přiřazení a inicializaci třídy SettingsFragment, která zajišťuje veškerou funkcionalitu uživatelského nastavení aplikace.

6.3.3 SettingsFragment

Třída je potomkem abstraktní třídy PreferenceFragmentCompat z podpůrné android knihovny AndroidX zajišťující funkcionalitu nastavení aplikace. Třída se skládá ze tří metod:

void onCreatePreferences(Bundle savedInstanceState, String rootKey)

Tato metoda je volána při vzniku aktivity nastavení. V metodě dojde ke konfiguraci nastavení dle xml dokumentu preference.xml. Dále zde dojde k inicializaci listeneru změn nastavení. V něm dochází ke kontrole uživatele, jestli zadává povolené hodnoty a taky změny popisu jednotlivých nastavení dle zvolených hodnot.

void onResume()

Tato metoda je volána při spuštění aktivity. V této metodě dojde k registraci listeneru změn nastavení a taky změny popisu jednotlivých nastavení dle nastavení předešlého (nastavení uloženého v zařízení).

void onPause()

Tato metoda je volána při pozastavení aktivity. V této metodě dojde ke zrušení registrace listeneru změn nastavení.

6.3.4 ShapeReactionController

Třída implementuje rozhraní IShapeDetectionReactionController z API. Třída spočítá počet detekovaných objektů dle jejich typu a zobrazí tento počet na uživatelském rozhraní.

6.3.5 WiFiVehicleController

Třída implementuje rozhraní IVehicleController z API a zajišťuje komunikaci smartphonu s ovládaným vozidlem skrz Wi - Fi. Při inicializaci si uživatel zvolí maximální procentuální hodnotu výkonu, kterým chce, aby se vozidlo pohybovalo; vyvažovací koeficient v intervalu [-1, 1], který zajistí pohyb vozidla po rovné čáře, pokud je jeden z motorů silnější, než ten druhý; IP adresu vozidla a textová pole pro zobrazení procent výkonu motorů na uživatelském rozhraní. Při zavolání jakékoliv z metod z rozhraní, zobrazí výkon v procentech na textová pole uživatelského rozhraní a zavolá metodu sendCommandToVehicle().

void sendCommandToVehicle(int powerLeft, int powerRight)

Pomocí knihovny OkHttp verze 4.7.2[2] vytvoří a pošle POST request do vozidla ve formátu:

```
http://<ip_adresa>/drv?w1=<vykon_leveho_motoru>&w2=<vykon_praveho_motoru>
```

void equalizePower(double steeringEqualizer)

Tato metoda je volána v konstruktoru této třídy a zajišťuje vyrovnání výkonu motorů, aby se vozidlo mohlo pohybovat rovně, na základě určeného vyrovnávacího koeficientu. Pokud je koeficient záporný, vynásobí se maximální hodnota výkonu levého motoru. Pokud je koeficient kladný dojde k vynásobení maximálního výkonu pravého motoru.

void showPowerPercentages(final double leftPowerPercentage, final double rightPowerPercentage)

Metoda nastaví text textových polí zobrazující aktuální výkon motorů v procentech dle předaných hodnot.

[2] https://square.github.io/okhttp/

6.3.6 USBService

Třída je potomkem třídy Service. Třída zajišťuje komunikaci na USB portu smartphonu. Třída po vzniku prohledá všechna připojená USB zařízení a při nalezení prvního kompatibilního zařízení se zeptá uživatele na práva k jeho manipulaci. Pokud uživatel udělí aplikaci potřebná práva, třída naváže spojení. Třída byla implementována dle ukázkového kódu od autora knihovny UsbSerial[3].

6.3.7 USBVehicleController

Třída implementuje rozhraní IVehicleController z API a zajišťuje zasílání příkazů ovládanému vozidlu skrz USB spojení. Při inicializaci si uživatel zvolí maximální procentuální hodnotu výkonu, kterým chce, aby se vozidlo pohybovalo; vyvažovací koeficient (-1 – 1), který zajistí pohyb vozidla po rovné čáře, pokud je jeden z motorů silnější, než ten druhý a textová pole pro zobrazení procent výkonu motorů na uživatelském rozhraní. Při zavolání jakékoliv z metod z rozhraní, zobrazí výkon v procentech na textová pole uživatelského rozhraní a zavolá metodu sendCommandToVehicle().

void sendCommandToVehicle(int powerLeft, int powerRight)

Pomocí třídy USBService zašle skrz USB spojení příkaz k pohybu vpřed a nastaví výkon jednotlivých motorů dle předaných hodnot.

void equalizePower(double steeringEqualizer)

Tato metoda je volána v konstruktoru této třídy a zajišťuje vyrovnání výkonu motorů, aby se vozidlo mohlo pohybovat rovně, na základě určeného vyrovnávacího koeficientu. Pokud je koeficient záporný, vynásobí se maximální hodnota výkonu levého motoru. Pokud je koeficient kladný dojde k vynásobení maximálního výkonu pravého motoru.

void showPowerPercentages(final double leftPowerPercentage, final double rightPowerPercentage)

Metoda nastaví text textových polí zobrazující aktuální výkon motorů v procentech dle předaných hodnot.

[3] `https://github.com/felHR85/UsbSerial/tree/master/example`

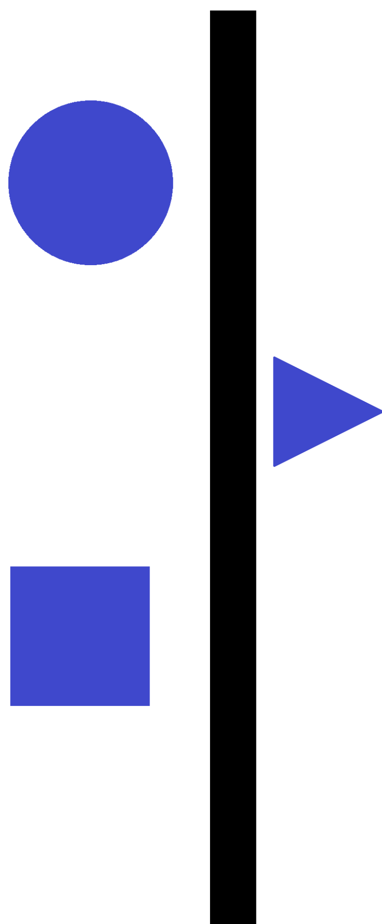
7 Testování a výsledky

7.1 Způsob testování

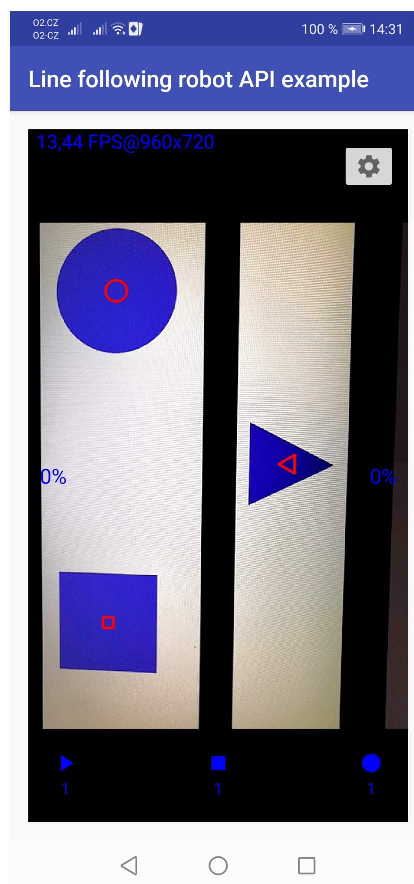
Testování probíhalo pomocí 2 vozidel - jedno ovládané přes Wi-Fi, druhé pomocí USB a dvou Android zařízení různé cenové třídy - Xiaomi Mi A1 (zařízení s nižším výkonem a výrazně horším fotoaparátem) a Honor 10. Aplikace byla testována v různých světelných podmínkách (přirozené denní světlo, umělé osvětlení, osvětlení vlastní svítilnou smarthphonu). Co se týče samotného následování čáry, byla aplikace testována na dráze vytisknuté na papír. Dráha obsahovala sadu ostrých 90-stupňových zatáček a sadu méně prudkých zatáček.

7.2 Detekce geometrických tvarů

Pomocí knihovny OpenCV se mi podařilo implementovat kód schopný detekovat tří základní geometrické tvary – trojúhelník, obdélník a kruh. Aby nedocházelo k falešné detekci tvarů, které by ve skutečnosti byli čárou, kterou má vozidlo následovat, musí být nejprve zvolena barva tvarů, které bude algoritmus detekovat. Místa obsahující tuto barvu jsou vymaskovány a následně klasifikovány jako jednotlivé tvary dle počtu jejich aproximovaných stran. Pokoušel jsem se zlepšit rychlost zpracování zmenšením rozlišení zpracovávaného obrazu. Ovšem bottleneck tohoto přístupu je algoritmus Douglas-Peucker, který je nezávislý na rozlišení, ale na počtu aproximovaných hran (viz kapitola 3.4). Díky tomuto faktu bylo zlepšení výkonu nepatrné (např. při zpracování obrázku 7.2 při 4x zmenšeném rozlišení došlo k změně snímků za sekundu ze 14.5 na 15.6)
Tento přístup se ukázal jako efektivní, a to jak z pohledu rychlosti, tak i z pohledu schopnosti detekce tvarů (viz tabulka 7.1 a obrázky 7.1 a 7.2). Největší nevýhodou tohoto přístupu je především jeho závislost na kvalitě fotoaparátu zařízení. Fotoaparáty nižší kvality mohou totiž při zhoršených světelných podmínkách chybně snímat barvy, díky čemuž dojde k vytvoření špatné barevné masky (viz obrázky 7.3 a 7.4). Další nevýhodou je závislost rychlosti zpracování na počtu aproximovaných hran objektů, tím pádem i na počtu objektů v obraze. Tato nevýhoda se však projevuje až u opravdu velkého množství objektů (20+), které by se však za normálních podmínek ani nevešly do viditelného prostoru fotoaparátu zařízení.



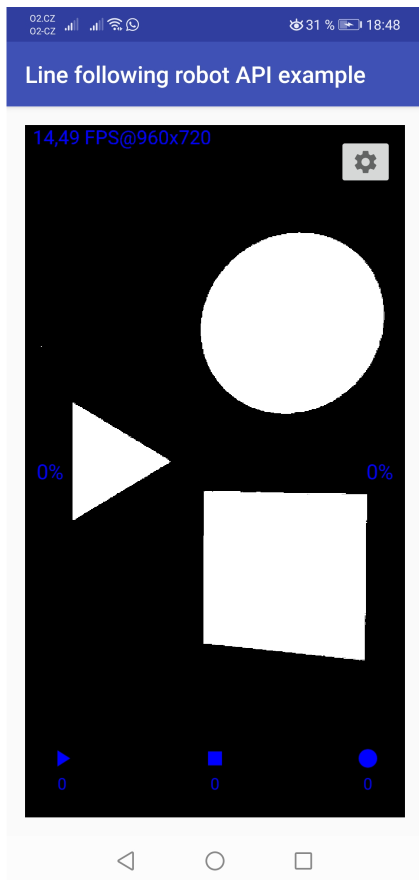
Obrázek 7.1: Původní obraz před zpracováním detekcí tvarů



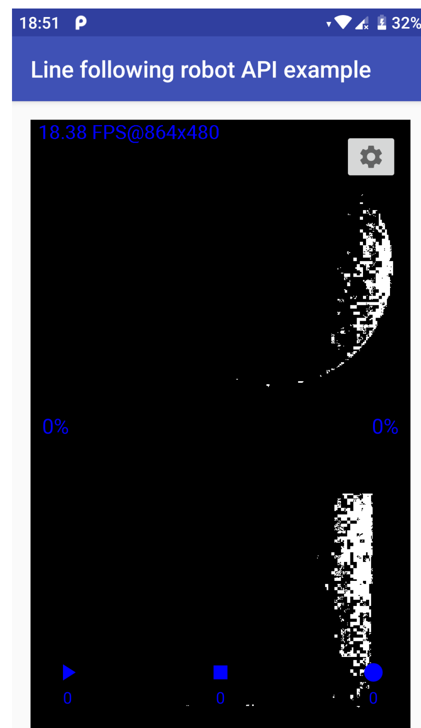
Obrázek 7.2: Obraz po zpracování detekcí tvarů

	Vypnutá detekce	Zapnutá detekce
Xiaomi Mi A1	21.2	8.8
Honor 10	29.9	11.3

Tabulka 7.1: Průměrný počet snímků za sekundu v závislosti na zařízení



Obrázek 7.3: Maska vytvořená zařízením Honor 10



Obrázek 7.4: Maska vytvořená zařízením Xiaomi Mi A1

7.3 Následování čáry

7.3.1 Komunikace

Původním plánem bylo implementovat komunikaci skrz technologii Wi-Fi. Tato technologie se díky vysoké datové propustnosti a nízké latenci zdála jako ideálním kandidátem. Ovšem opak je pravdou. Vysílač vozidla a vysílač smartphonu se díky jejich velice krátké vzájemné vzdálenosti začali rušit, což způsobilo kompletní selhání schopnosti vozidla přijímat příkazy k pohybu. Z tohoto důvodu jsem nakonec implementoval komunikaci zkrze technologii OTG (On-The-Go) USB.

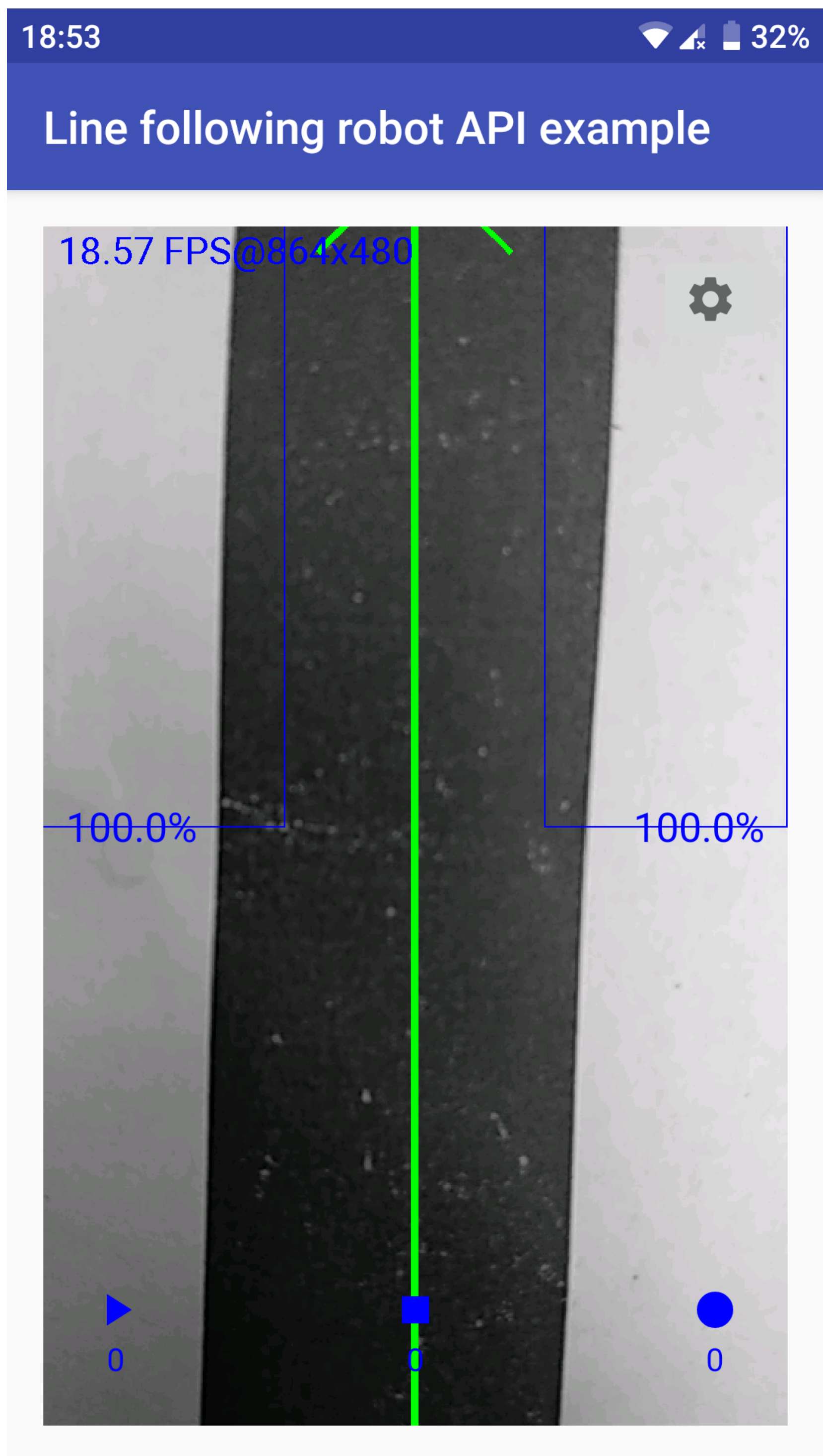
7.3.2 Detekce čáry

Porovnávání jasu

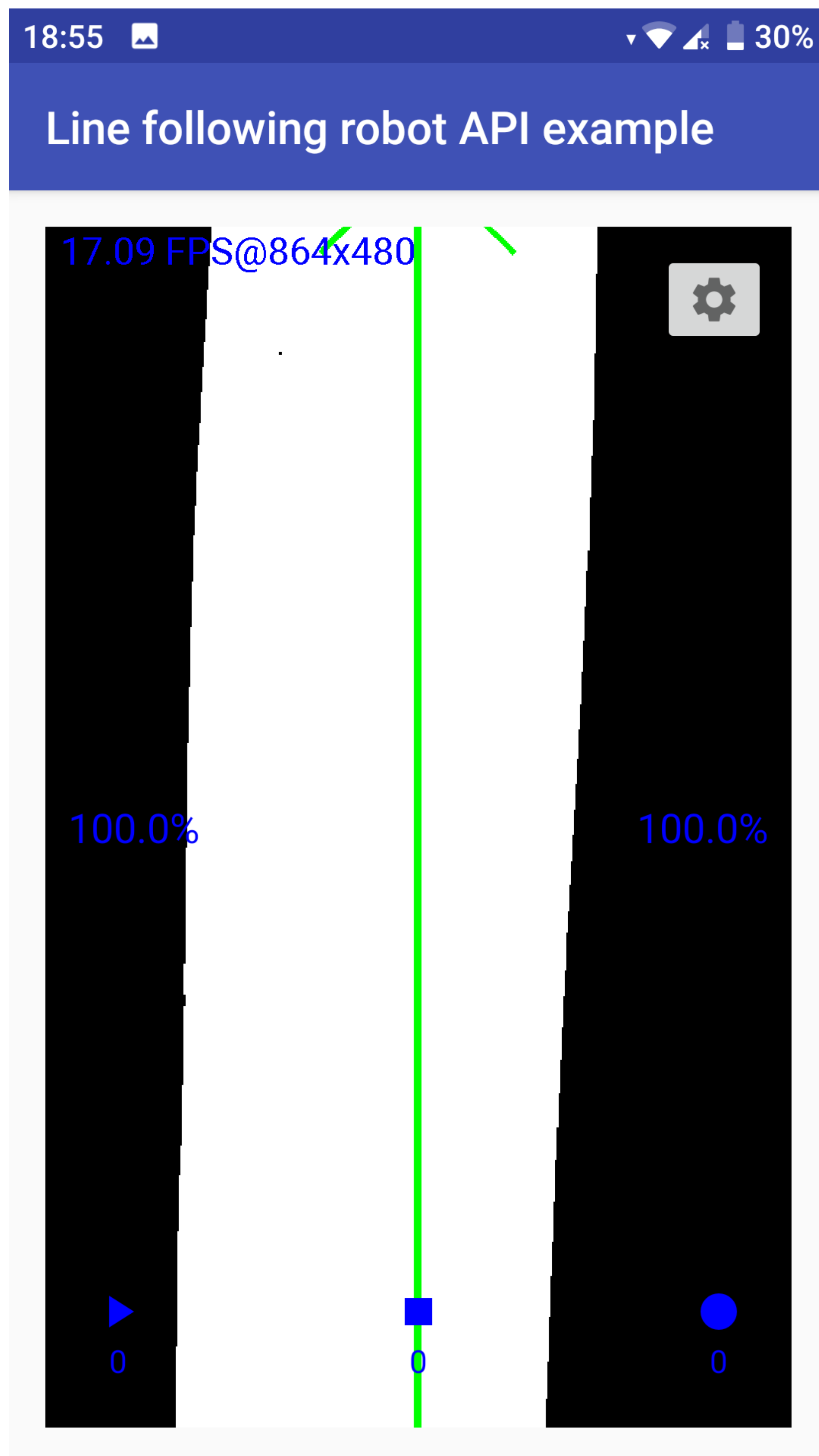
Tento přístup se ukázal jako nejvíce efektivní a to především díky jeho robustnosti vůči špatným světelným podmínkám a rušivým elementům (např. smítka prachu v okolí čáry, přerušení čáry). Další výhodou je, že je schopný detekovat jakoukoliv tmavou čáru. Jedná se také o jediný přístup, který lze použít za tmy. Svítilna totiž neovlivňuje výsledky tohoto algoritmu. Jedná se také o nejrychlejší z přístupů (viz tabulka 7.2 a obrázek 7.5).

Maskování černé

Tento přístup se ukázal jako druhé funkční řešení tohoto problému. Ve spojení se sumou pixelů na stranách obrazu vytváří poměrně robustní řešení (viz obrázek 7.6). Mezi jeho problémy patří především ovlivnitelnost vnějšími světelnými podmínkami a jeho rychlost – jde o nejpomalejší přístup (viz tabulka 7.2). Pokud se spojí s počítáním těžiště obrazu, můžeme pozorovat značné zlepšení plynulosti jízdy vozidla. Ovšem pouze do té doby, dokud nenarazíme na prudší zatáčku. V tomto případě totiž není těžiště dostatečně daleko od středu osy x a nedojde k dostatečnému zatočení. Pokoušel jsem se tento problem vyřešit s pomocí hranice na ose x zpracovávaného obrazu. Pokud se těžiště dostane za ní, je motor na dané straně úplně vypnut namísto snížení jeho výkonu. To ovšem vyústilo pouze ve ztrátu jediné výhody tohoto přístupu – plynulosti pohybu vozidla a zisku méně spolehlivého přístupu, než oba výše zmíněné.



Obrázek 7.5: Výsledek porovnávání jasu na stranách obrazu



Obrázek 7.6: Výsledek maskování černé s kombinací s počítáním pixelů na stranách obrazu

Obrázek 7.7: Sobel detektor s výpočtem těžiště

Obrázek 7.8: LoG detektor s výpočtem těžiště

Obrázek 7.9: Sobel detektor s Hough transformacemi

Detekce hran

Ačkoliv byl tento přístup určen při analýze jako hlavní a to především k jeho univerzálnosti v barvě čáry, tak se ukázal jako v praxi nepoužitelný a to u obou implementovaných algoritmů (Sobelův detektor i detektor Laplacian of Gaussian). Sobelův algoritmus selhává v tom, že jako hranu detekuje jakoukoliv nedokonalost čáry a i artefakty vzniklé snímáním obrazu za špatných světelných podmínek. Tento efekt se projevuje ještě víc po zapnutí vyrovnávání jasu. Laplacian of Gaussian kromě problémů zmíněných u Sobelova detektoru trpí ještě tím, že čím méně je světla o to méně výrazné jsou detekované hrany. Ať ve spojení se sumou pixelů na stranách obrazu nebo ve spojení s výpočtem těžiště obrazu nebylo možné dojít k dostatečně konzistentním výsledkům, aby šlo o tomto přístupu uvažovat jako o plně funkčním řešení. Tomu nedopomohlo ani implementace Houghovo algoritmu pro detekci čar. Nejen že snímky za sekundu spadly pod hranici jednoho snímku za sekundu, ale výsledek se nijak nezlepšil, ba naopak. (viz obrázky 7.7, 7.8 a 7.9).

	Xiaomi Mi A1	Honor 10
Sobel + COG	16.7	21.1
LoG + COG	18.5	22.4
Maskování černé + COG	17.1	18.6
Sobel + Suma pixelů na stranách	16.9	20.1
Laplacian + Suma pixelů na stranách	18.1	22.1
Maskování černé + Suma pixelů na stranách	14.8	19.8
Porovnání jasu na stranách	18.2	24.2
Bez detekce	21.2	29.9

Tabulka 7.2: Průměrný počet snímků za sekundu v závislosti na zvoleném přístupu následování čáry a zařízení

7.3.3 Celkové zhodnocení přístupů k detekci a následování čáry

Podařilo se implementovat dvě funkční řešení k následování čáry – Porovnávání jasu a Maskování černé ve spojení se sumou pixelů na stranách obrazu. Obě tyto řešení jsou schopné následovat jednoduchou čáru. Oba přístupy fungují nejlépe, pokud je fotoaparát umístěn co nejblíže k ose otáčení a zároveň vidí co největší množství z následované čáry, tzn. čím větší vzdálenost fotoaparátu od čáry, tím lépe. Ani jeden z přístupů si však nedokáže poradit s komplexnějším obrazem, tzn. v obraze je více čar, na čáře je křižovatka, v okolí čáry jsou tmavé tvary atd. Při implementaci jsem experimentoval se zmenšením rozlišení zpracovávaného obrazu. To ovšem způsobovalo to, že výsledky zpracování byli podstatně horší. Navíc rozdíl ve snímcích za sekundu nebyl větší než 2 - 4 fps, což při pohybu vozidla nezpůsobí žádný rozdíl. Viz tabulka 7.2.

8 Závěr

Podařilo se mi implementovat funkční API, s pomocí kterého lze snadno implementovat vlastní aplikaci k následování čáry, nehledě na způsob komunikace s vozidlem. Uživatel API si také může sám určit akce prováděné při detekci různých geometrických tvarů.
Nad tímto API jsem také implementoval demo aplikaci, která demonstruje všechny funkce API, tzn. uživatel aplikace si může zvolit mezi všemi implementovanými algoritmy a vyzkoušet si, jak se API chová v různých situacích.
Co se týče samotné implementace, byl jsem překvapen rozdílem mezi praxí a teorií. Odkazuji se teď především na výslednou nepoužitelnost původně navrhovaného přístupu následování čáry - kombinace detekce hran a výpočtu jejich těžiště. Nejlepším řešením se nakonec ukázalo nejjednodušší přístup k problému a to porovnávání jasu na opačných stranách zpracovávaného obrazu. Detekce geometrických tvarů se mi podařilo implementovat dle původního návrhu. Jeho funkčnost je však omezená schopností fotoaparátu zařízení Android správně rozeznávat barvy i za zhoršených světelných podmínek.
Tímto jsem splnil všechny body zadání. Navrhuji však pokračování v průzkumu dalších možností, jak doplnit funkcionalitu. Příkladem vhodné nástavby by byla logika pro orientaci se na křižovatkách nebo zpětná vazba o skutečném pohybu vozidla přes optický tok.
Při implementaci této práce jsem se dozvěděl mnoho nových informací o zpracování obrazu a taktéž jsem získal nové zkušenosti ve vytváření aplikací pro mobilní zařízení na platformě Android.

Literatura

[1] OpenCV documentation. Dostupné z: https://docs.opencv.org/3.4.10/.

[2] Chandwadkar, R. Comparison of Edge Detection Techniques. 08 2013. doi: 10.13140/RG.2.1.5036.7123.

[3] Gonzalez, R. C. – Woods, R. E. *Digital image processing.* Addison-Wesley, 2nd edition, 1993.

[4] Ismail, A. H. et al. Vision-based system for line following mobile robot. 2, s. 642 – 645, 11 2009. doi: 10.1109/ISIEA.2009.5356366.

[5] Kazi, M. – Nahid, A. – Mamun, A. Implementation Of Autonomous Line Follower Robot. 04 2012.

[6] Levner, I. Shape Detection, Analysis and Recognition. 2002. doi: 10.7939/R3N29P60T. Dostupné z: https://era.library.ualberta.ca/items/6e675137-56d5-4c53-9061-c943cfc7c034.

[7] Shrivakshan, G. – Chandrasekar, C. A Comparison of various Edge Detection Techniques used in Image Processing. *International Journal of Computer Science Issues*. 09 2012, 9, s. 269–276.

[8] Suzuki, S. – be, K. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*. April 1985, 30, 1, s. 32–46. doi: 10.1016/0734-189x(85)90016-7. Dostupné z: https://doi.org/10.1016/0734-189x(85)90016-7.

[9] Wu, S.-T. – Marquez, M. A non-self-intersection Douglas-Peucker algorithm. In *16th Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI 2003)*. IEEE Comput. Soc. doi: 10.1109/sibgra.2003.1240992. Dostupné z: https://doi.org/10.1109/sibgra.2003.1240992.

Přílohy

Uživatelská příručka

Překlad a instalace

Pro překlad a vytvoření balíčku pro instalaci na vašem zařízení je potřeba mít nainstalovanou Javu (verze 1.8 a vyšší) a build utilitu Gradle (verze 6.1 a vyšší).

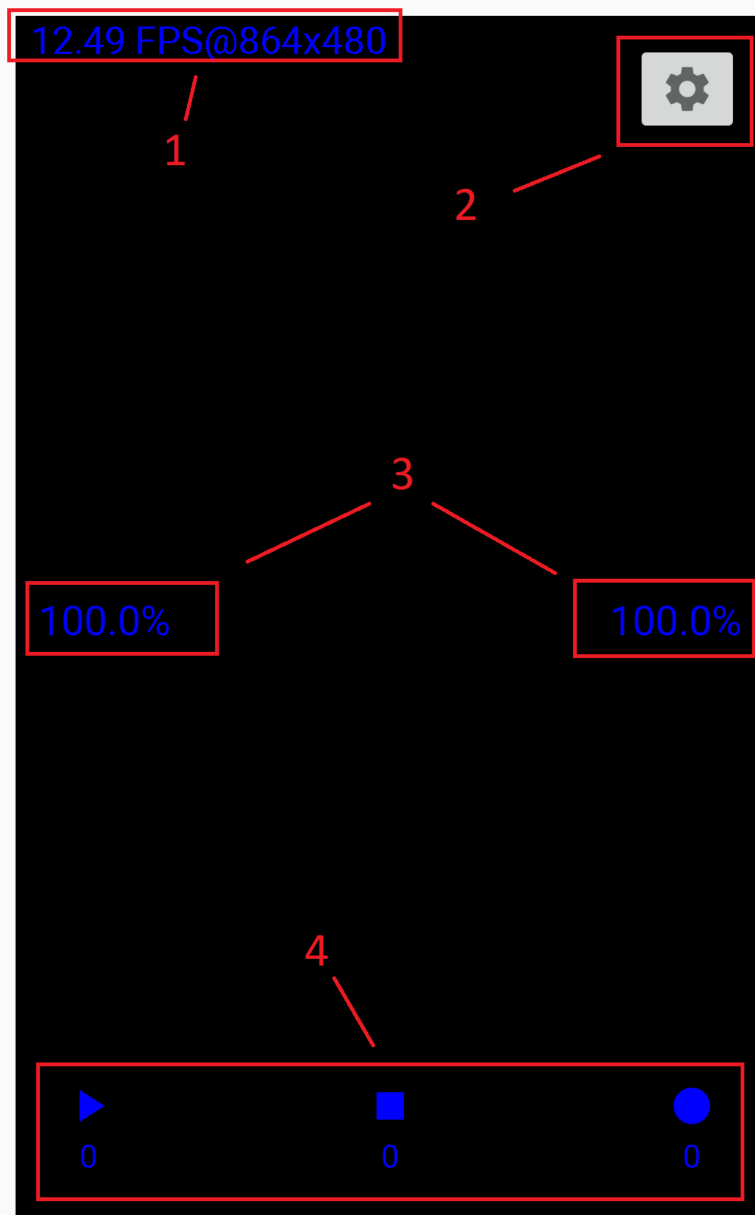
Pokud máte výše zmíněné programy nainstalované, stačí přejít do složky `...\LineFollowingRobotAPI\app\` a zadat příkaz:

```
> gradle assembleDebug
```

Po proběhnutí celého procesu se ve složce `...\LineFollowingRobotAPI\app\build\outputs\apk\debug\` vytvoří soubor `app-debug.apk`. Ten si stačí přesunout do svého Android zařízení a nainstalovat. Pozor! V nastavení svého zařízení si musíte povolit instalaci aplikací z cizích zdrojů.

Popis uživatelského rozhraní

Aplikace má velice jednoduché uživatelské rozhraní (dále UI) sestávající se ze dvou obrazovek – základní UI a nastavení. Na hlavní obrazovce se zobrazuje zpracovaný obraz v reálném čase. Nastavení lze otevřít stiskem tlačítka s ozubeným kolečkem (viz obrázek 8.1) - bod č. 2. Stiskem tohoto tlačítka dojde také k zastavení ovládaného vozidla. Z nastavení se lze dostat zpět na hlavní obrazovku stisknutím tlačítka zpět na vašem zařízení Android Hlavní obrazovka dále obsahuje čítač snímků za sekundu (viz obrázek 8.1) - bod č. 1, procentuálně vyjádřený výkon obou motorů (viz obrázek 8.1) - bod č. 3 a čítače detekovaných geometrických tvarů.



Obrázek 8.1: Popis uživatelského rozhraní.
1 - čítač snímků za sekundu
2 - tlačítko pro otevření nastavení aplikace
3 - ukazatele momentálního výkonu motorů v procentech
4 - čítače na momentální počet detekovaných geometrických tvarů

Popis nastavení

Flashlight

Zapnutí/vypnutí svítilny zařízení Android.

Lower the resolution of processed images for better performance

Snížení rozlišení zpracovávaného obrazu při detekci geometrických tvarů.

Show images from Line following/Shape detection

Volba mezi zobrazování výsledků z algoritmu na následování čáry nebo z algoritmu na rozpoznávání tvarů na hlavní obrazovce.

Line Following

Zapnutí/vypnutí algoritmů pro následování čáry.

White Balance

Zapnutí/vypnutí vyrovnávání jasu při následování čáry.

Line detection algorithm

Volba algoritmu na detekování čáry. Je na výběr mezi Sobelovo detektoru hran, detektoru hran Laplacian of Gaussian, Maskování černé a žádného algoritmu (pokud chceme použít algoritmus porovnávání jasu na stranách obrazu).

Turn on/off Hough transformation

Zapnutí/vypnutí vylepšení detekce hran houghovo transformacemi.

Shape Detection

Zapnutí/vypnutí detekce tvarů v obraze.

Color of Shapes

Volba barvy geometrických tvarů k detekci. Je na výběr mezi základními barvami – červenou, zelenou a modrou.

Direction Detection

Zapnutí/vypnutí detekce směru pohybu ovládaného vozidla.

Forward Direction Toleration

Nastavení tolerance rozdílu v sumách při detekci směru algoritmem suma pixelů na stranách zpracovávaného obrazu pro pohyb dopředu. Pokud je hodnota nastavena na 0, vozidlo se nikdy nebude pohybovat dopředu. Pokud je hodnota nastavena na 1, vozidlo se bude pohybovat jen dopředu.

Draw direction indication on screen

Zapnutí/vypnutí zakreslování směru pohybu na hlavní obrazovku. Při algoritmech Porovnávání jasu a Suma pixelů na stranách zpracovávaného obrazu je zakreslován směr šipkami, při počítání těžiště je zakreslována do obrazu kružnice na souřadnice těžiště. *Direction Detection Algorithm Volba algoritmu pro detekci směru pohybu ovládaného vozidla. Je na výběr mezi Sumou pixelů na stranách zpracovávaného obrazu, výpočtem těžiště a porovnáváním jasu na stranách zpracovávaného obrazu.

Center of screen ignorance

Nastavení procenta středu obrazovky při algoritmu porovnávání jasu, který algoritmus ignoruje. Algoritmus zakresluje do obrazu dva obdélníky reprezentující oblasti, které algoritmus bere v potaz. Tuto hodnotu nastavte tak, aby při pohybu vozidla po rovné čáře byla čára mimo tyto dvě zmíněné oblasti.

Brightness tolerance

Nastavení tolerance hodnoty jasu (0 – 255) pro pohyb vozidla dopředu.

Vehicle Control

Zapnutí/vypnutí ovládání vozidla.

Wheel Power Calibration

Nastavení procent vyrovnávání výkonu motorů, pokud je jeden silnější, než ten druhý. Záporné hodnoty zvyšují výkon motoru na levé straně vozidla, kladné hodnoty zvyšují výkon motoru na pravé straně vozidla.

Power Level

Nastavení procent výkonu vozidla, který chceme maximálně použít.