VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Detekce překážek s využitím hloubkových obrazů

An Obstacle Detection System using Depth Images

2017 Lukáš Odstrčil

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Lukáš Odstrčil**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Detekce překážek s využitím hloubkových obrazů
An Obstacle Detection System Using Depth Images

Jazyk vypracování: čeština

Zásady pro vypracování:

Inteligentní asistenční systémy v dopravě se stávají běžným vybavením moderních vozidel. Jedním z takových asistenčních systémů je detekce překážek před vozidlem. Tento systém může varovat řidiče před nebezpečím nebo může být využit při vývoji autonomních vozidel. Cílem této bakalářské práce je vytvořit program pro detekci překážek na základě analýzy hloubkových dat získaných ze senzoru umístěného na vozidle.

Ve své práci proveďte:
1. Popište zadaný problém.
2. Analyzujte řešení a popište potřebnou teorii.
3. Proveďte vlastní implementaci řešení a její testování.
4. Zhodnoťte dosažené výsledky.

Seznam doporučené odborné literatury:

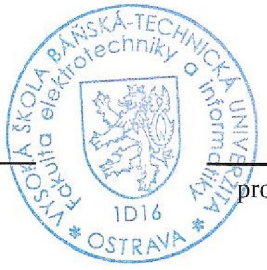
[1] R.C. Gonzalez, R.E. Woods: Digital Image Processing (3rd Edition), 2006, ISBN 013168728X
[2] E. Sojka: Digitální zpracování a analýza obrazů, 2000, ISBN 80-7078-746-5
[3] R. Labayrade, D. Aubert: In-vehicle Obstacles Detection and Characterization by Stereovision, 2003

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

Vedoucí bakalářské práce: **Ing. Michael Holuša**

Datum zadání: 01.09.2015

Datum odevzdání: 28.04.2017



doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

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

V Ostravě 28. dubna 2017 .....

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

V Ostravě 28. dubna 2017

.....

Chtěl bych poděkovat panu Ing. Michaelu Holušovi, za pomoc při vypracovávání této bakalářské práce, bez které by nevznikla.

Abstrakt

Vzhledem k neustálým nehodám způsobených na parkovišti, ale i na cestách, při pohybu ve skladech s vysoko zdvižným vozíkem nebo ovládání robotů, kteří mapují prostředí a nechceme ho poškodit nárazem, je třeba asistenční systém pomáhající odhalit překážku. V této bakalářské práci byl navržen systém, který ze sekvence hloubkových obrazů získaných z kamery (Orbbec Astra) nalezne a zobrazí řidiči překážky před, respektive za vozidlem. Principem řešení problému, je prahování vstupního obrazu, nalezení překážky pomocí změn jasu v hloubkovém obrazu a nakonec zviditelnění překážky, které upozorní řidiče na možnou kolizi. Aplikace využívá knihovnu OpenCV, ze které byly použity některé funkce při detekování objektu. Během návrhu bylo snahou vytvořit systém, který by se dal použít v reálném čase a dalo se tak včas vyhnout překážkám, které stojí vozidlu v cestě.

Klíčová slova: Orbecc Astra, OpenCV, detekce překážek

Abstract

Due to the constant accidents caused in parking lots, on the roads, when moving the high-lift trucks in warehouse or controlling robot which maps surroundings and we don't want to damage this robot, the parking assistance system helping the driver detect obstacles is necessary. In this thesis, we designed the system that finds and displays obstacles in front or behind the car to the driver from sequence of images from camera with depth image (Orbecc Astra). The principle of solving the problem is the thresholding of the input image, finding obstacles by changing the brightness in the depth image and, in the end, making obstacles more visible to warn driver about a possible collision. The application uses the OpenCV library from which we used some features to detect obstacles. During the design we wanted to create a system which could be used in real time to avoid the obstacles standing in the way.

Key Words: Orbecc Astra, OpenCV, obstacle detection

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Poslední dobou jsou všude ve světě hojně zastoupeny asistenční systémy, které pomáhají řidičům, ale i robotům detekovat překážky. Na detekci překážek se používají například infračervené, kamerové nebo laserové senzory nebo ultrazvukové senzory, které objeví překážku na základě vyhodnocení odrazu zvuku od překážky.

V této práci jsme se rozhodli pro detekci pomocí kamery s hloubkovým obrazem, který je nabízen v moderních zařízeních. Alternativou bez hloubkové kamery by byla detekce pomocí dvou normálních kamer, ze kterých se programově dostane hloubkový obraz pomocí stereoprojekce a pak je možné detekovat překážky. Kamery je však potřeba kalibrovat. Při nesprávném nakalibrování kamer může vzniknout nepřesnost v detekci. Na druhou stranu hloubková kamera, jelikož pomocí 3D senzorů vytváří hloubkový obraz a nemusí ho vytvářet program, pracuje rychleji a výsledný obraz je mnohem přesnější a tím i přínosnější pro náš problém, protože se jedná o bezpečnost řidiče.

Z hloubkového obrazu se dá detekovat překážka několika způsoby. Jelikož v hloubkovém obraze používané kamery jsou vzdálenější objekty světlejší a objekty, které jsou blíže kamery jsou tmavší, lze překážku nalézt pomocí aritmetického průměru a odchylky intenzit pixelů od ní.

Návrh tohoto asistenčního systému může mít následující uplatnění, např. automobilový průmysl, systém pro roboty nebo řidiče vysokozdvižných vozíků ve skladech nebo systém pro navigaci robotů, jako jsou třeba průzkumné sondy, nebo domácí spotřebiče (robotický vysavač).

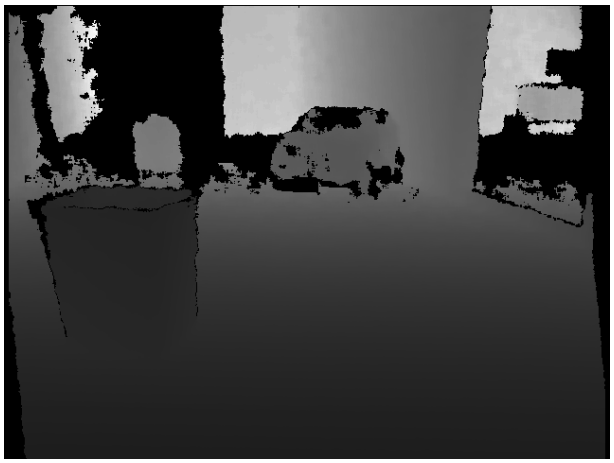
Struktura práce je následující: Kapitola 2 popisuje teoretický postup při řešení problému a popis metod při hledání a vyznačení překážky, Kapitola 3 popisuje průběh vytváření programu, který bude detekovat překážky, v Kapitole 4 budou ukázány jednotlivé experimenty vzniklého programu a popsán hardware a software, který byl na experimenty použit, a nakonec v Kapitole 5 shrneme celou práci, dosažené výsledky a jak by se mohl systém dále rozvinout.

2 Teoretický rozbor práce

V této kapitole jsou zmíněny postupy a algoritmy, které využijeme při řešení problému této práce.

2.1 Řešení problému

Po nahrání vstupních dat (jednotlivých obrazů) je nutné se zbavit pixelů, které představují až příliš vzdálené objekty nebo pozadí, abychom nemuseli zpracovávat zbytečně moc pixelů, které reprezentují body daleko od kamery. K tomuto problému jsme použili prahování, které obraz segmentuje a oddělí okraje bližších předmětů od pozadí. Nyní je třeba zjistit, které pixely patří pozadí a které patří překážce. Jelikož lze tento výsledek vizuálně rozeznat (viz Obrázek 1), může být problém řešen pomocí rozdílu odstínu pixelů (tmavší pixely jsou překážka, světlejší jsou pozadí). Díky tomuto poznatku bylo navrhnuto separovat pixely pomocí aritmetického průměru hodnoty barvy pixelů jednotlivých řádků a následovně je porovnávat s barvou samostatných pixelů. Při razantní odchylce můžeme nalézt hledané pixely překážky a dále s nimi pracovat. Po nalezení překážky je potřeba nějakým způsobem oddělit pozadí od námi nalezené překážky. Pokud pixely objektu separujeme, nemusíme už brát v potaz pixely pozadí což je ve výsledků výhodné například při určování tvaru překážky nebo při jejím zviditelněním. Výsledná překážka je poté zvýrazněna, aby dostal řidič nějaké upozornění o blížící se překážce a možné kolizi.



Obrázek 1: Ukázka hloubkového obrazu z kamery

2.1.1 Prahování (thresholding)

Thresholding nebo také prahování je nejjednodušší metoda jak segmentovat obraz. Pro výsledný prahovaný obraz, který se vytváří z barevného RGB obrazu nebo z obrazu v odstínech šedi se používá metoda, která každý pixel nahradí černým, bílým pixelem a nebo také pixelem v odstínu šedi, podle použitého typu prahování. Jestli bude pixel bílý nebo černý rozhoduje intenzita

pixelu, podle toho jestli je jeho intenzita větší nebo menší než konstanta, tzv. práh (thresh), která je nastavena podle potřeby.

Zde jsou příklady typů prahování:

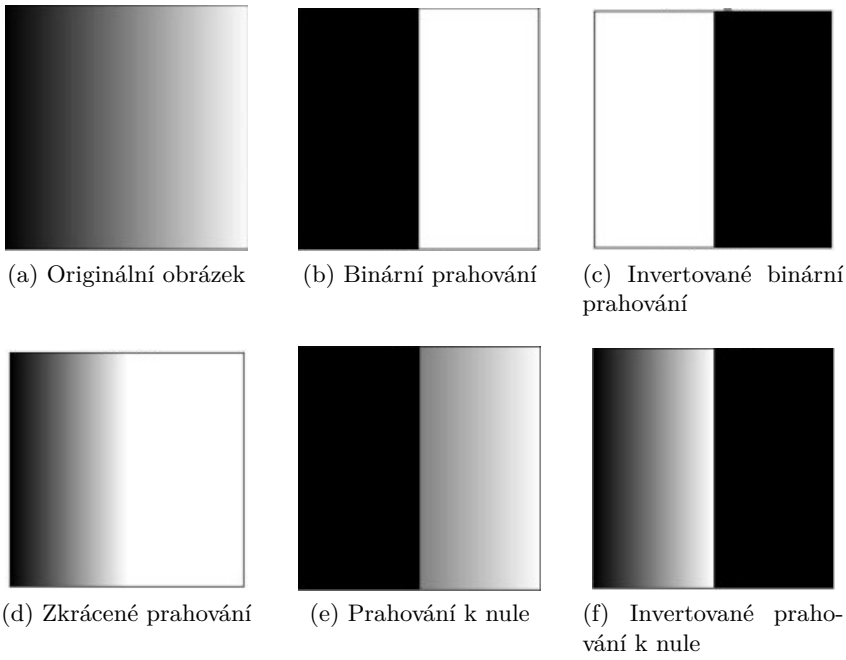
Binární prahování (Obr. 2b): Pokud je intenzita pixelu vetší než nastavený práh, je tento pixel nahrazen bílým pixelem (hodnota intenzity se nastaví na maximální hodnotu), v opačném případě je nahrazen pixelem černým (hodnota intenzity pixelu se nastaví na hodnotu 0).

Invertované binární prahování (Obr. 2c): Opak binárního prahování, pokud je intenzita větší než práh, intenzita pixelu je nastavena na 0 (černý pixel), v opačném případě je intenzita nastavena na maximální hodnotu.

Zkrácené prahování (Obr. 2d): Pokud je hodnota intenzity pixelu větší než maximální hodnota práhu, je pixel nastaven na hodnotu prahu. V opačném případě, pokud je hodnota intenzity pixelu menší než nastavený práh, jsou pixely ponechány bez úpravy.

Prahování k nule (Obr. 2e): Jestliže je hodnota intenzity pixelu menší než nastavený práh, je pixel nastaven na hodnotu 0. Ostatní pixely, které mají intenzitu větší než je hodnota práhu, zůstávají beze změny.

Invertované prahování k nule (Obr. 2f): Opak předešlého prahování. Pokud je hodnota intenzity pixelu větší než hodnota práhu, je hodnota pixelu nastavena na 0 a v opačném případě pokud je menší než hodnota prahu, tento pixel zůstává beze změny.



(a) Originální obrázek (b) Binární prahování (c) Invertované binární prahování

(d) Zkrácené prahování (e) Prahování k nule (f) Invertované prahování k nule

Obrázek 2: Druhy prahování s práhem nastaveným na 50% maximálního jasu [9]

Pro řešení našeho problému je přínosné prahování k nule a invertované prahování k nule. Důvodem je zachování intenzity pixelů, které mají menší, respektive větší hodnotu než práh,

jelikož vyhledávání překážek probíhá hledáním změny intenzity pixelu a toho bychom s ostatními prahováními nedosáhli, viz Obrázek 2.

2.2 Bilaterální filtr

Jedná se o nelineární vyhlazovací filtr, který zachová hrany a snižuje šum v obraze viz Obrázek 3. Toto je výhodou například oproti Gaussova filtru, který sice odstraní šum, ale rozostří informaci o hraně tak, že jsou hrany méně viditelné. To je pro nás značně nevýhodné při detekci objektu, kde potřebujeme zachovat ostré hrany, abychom mohli překážku bezpečně rozeznat. Princip tohoto filtru je nahrazení hodnoty intenzity pixelu, vyváženou hodnotou průměrné intenzity okolních pixelů jedné homogenní oblasti bez větších změn v hranách [2].

Bilaterální filtr je definovám takto:

$$I_p(x) = \frac{1}{W_p} \sum_{q \in S} G_{\delta_s}(||p-q||) G_{\delta_r}(|I_p - I_q|) I_q \tag{1}$$

Normalizovaná rovnice, pro výpočet váhy pixelu, která následně určuje výslednou intenzitu pixelu:

$$W_p = \sum_{q \in S} G_{\delta_s}(||p-q||) G_{\delta_r}(|I_p - I_q|) \tag{2}$$

- **p** je poloha výstupního pixelu,
- **q** je poloha pixelu, který je v blízkosti výstupního pixelu,
- I_p je barva výstupního pixelu,
- I_q je barva pixelu v blízkosti výstupního pixelu,
- **S** je okolí výstupního pixelu,
- G_{δ_s} Gaussův filtr s odchylkou δ_s, která je pro vyhlazení rozdílů intenzity barvy,
- G_{δ_r} Gaussův filtr s odchylkou δ_r, která je pro vyhlazení rozdílů souřadnic
- W_p je váha výstupního pixelu

Pro pixel v obraze , ve kterém je potřeba zbavit se šumu, umístěný na souřadnicích *(x, y)* a jeden z jeho okolních pixelů má souřadnice *(p, q)*, potom je výsledná váha pro pixel *(p, q)* k odstranění šumu pixelu *(x, y)* určena takto:

$$W_{(x,y,p,q)} = e(-\frac{(x-p)^2 + (y-q)^2}{2\delta_s^2} - \frac{||I_{(x,y)} - I_{(p,q)}||^2}{2\delta_r^2}) \tag{3}$$

kde δ_s a δ_r jsou vyhlazovací parametry a $I_{(x,y)}$ a $I_{(p,q)}$ jsou intenzity pixelů na souřadnicích *(x, y)* a *(p, q)*. Po vypočtení váhy pixelu, můžeme vypočítat výslednou intenzitu pixelu *(x, y)* zbaveného šumu I_D:

$$I_{D(x,y)} = \frac{\sum_{p,q} I_{(p,q)} W_{(x,y,p,q)}}{\sum_{p,q} W_{(x,y,p,q)}} \tag{4}$$

(a) Originální obrázek

(b) Obrázek po aplikace bilaterálního filtru

Obrázek 3: Ukázka aplikace bilaterálního filtru

2.3 Nalezení bodů překážky

Překážku značí změna barvy v jednotlivých řádcích, jak je možné vidět na Obrázku 4a. Proto je nutné hledat body překážky pomocí změn odstínu barvy v obraze.

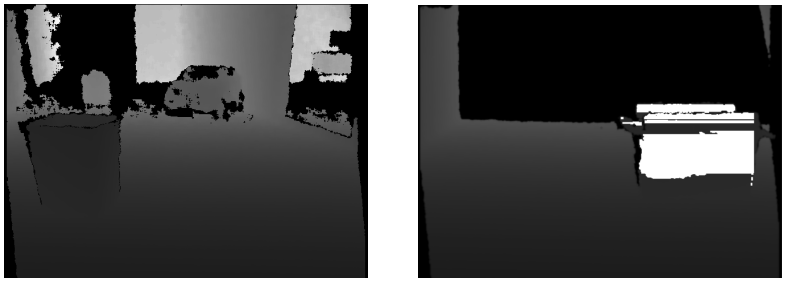
2.3.1 Výpočet průměrné hodnoty barvy

Pro vyhledání rozdílů mezi barvami, byl použit výpočet aritmetické hodnoty:

$$\oslash = \frac{\sum_{i=0}^{P} I_{x_i}}{P} \tag{5}$$

kde I_{x_i} značí intenzitu pixelu v daném řádku obrazu a P počet pouze barevných pixelů v řádku. Do celkového počtu pixelů na jednotlivých řádcích nebyly započteny černé pixely, tj. pixely, které mají malou intenzitu a jedná se o plochu, kde kamera nedohlédne nebo je naopak až moc blízko kamery a tedy není známá hloubka obrazu. Poté co vypočítáme průměrnou hodnotu intenzity pixelu na řádku, budeme hledat odchylku od jednotlivých pixelů v řádku. Z experimentálního měření jsme zjistili, že pokud je tato odchylka v určitém rozmezí jedná se o body překážky, tyto body mají menší intenzitu, jsou tmavší a tím pádem blíže kameře než okolní pozadí. Na rozdíl od odchylky, která je mimo toto rozmezí, pak tyto body naopak patří zemi, pozadí nebo stěně. Dále bylo zjištěno, že jsou rozmezí rozdílné a to podle počtu černých (tudíž nezapočtených) pixelů.

K odchylce je taky potřeba ještě brát v potaz intenzitu samotného pixelu. Pokud bychom nebrali ohled na intenzitu pixelu, stalo by se, že bychom vybírali pixely v pozadí, jelikož tyto pixely nemají intenzitu nulovou, ale intenzitu, která se k nule blíží.



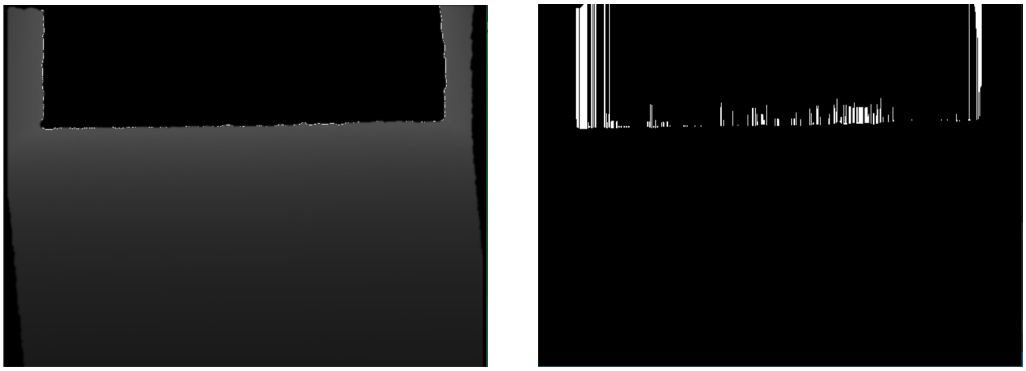
(a) Originální obrázek

(b) Obrázek po vyhledání pixelů překážky

Obrázek 4: Vzniklý obraz po vyznačení překážky (aplikaci průměrné hodnoty)

2.4 Odstranění nežádoucích pixelů

V průběhu vyhledávání překážek se stává, že jsou některé pixely započítány omylem a do překážky nepatří (Obr. 5a). Tak je tomu například u přechodu mezi zdí a pozadím nebo zemí a pozadím (černou plochou). Tyto body mají dostatečně velkou intenzitu na to, aby splňovaly kritéria pro vyhledávací algoritmus, ale o překážku se nejedná. Je tedy nutné se těchto pixelů zbavit, aby nám v pozdějším zpracování nevytvářely falešné detekce (Obr. 5b). S tímto problémem nám mohou pomoci morfologické operace, které nám odstraní malé samostatné body, které nejsou překážkou.



(a) Obrázek s nežádoucími pixely (b) Obrázek po zpracování nežádoucích pixelů

Obrázek 5: Ukázka práce s nežádoucími pixely

2.4.1 Morfologické operace

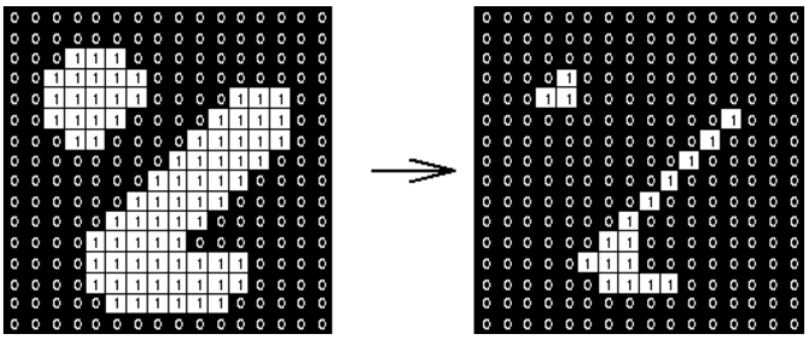
Jedná se o operace, které analyzují a zpracovávají geometrické struktury, na základě teorie množin, mřížek a topologie. Základními morfologickými operacemi jsou eroze, dilatace, otevírání a uzavírání.

- **Eroze** (znak eroze: $\ominus$)

  U operace eroze jsou vždy dva vstupy, první je obraz, který má být erodován a druhý je malá množina bodů známá jako strukturální prvek nebo také jádro. Tento strukturální prvek určuje jaký účinek bude mít eroze na vstupní obraz. Erozí se rozumí průnik všech posunů vstupního obrazu X o vektory $-b \in B$ tak, že se původní obraz zmenší o množinu B, respektive vektor -b, který náleží množině B, viz Obrázek 6.

  Vzorec: $X \ominus B = \bigcap_{b \in B} X_{-b}$

  Eroze se používá k rozložení objektu na jednodušší části. Objekty jsou tenčí a ty, které jsou menší než je strukturální prvek zaniknou.



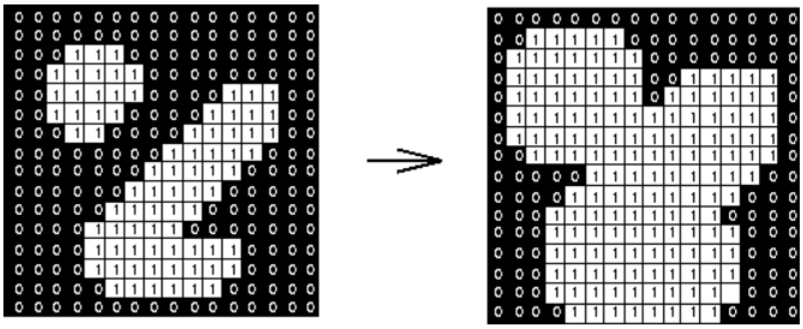
Obrázek 6: Ukázka principu fungování eroze [4]

- **Dilatace** (znak dilatace: $\oplus$)

  Tak jako u eroze i u dilatace jsou nutné dva vstupy, obraz a strukturální prvek, který určuje účinek dilatace na vstupním orazu. Dilataci se rozumí sjednocení množiny vstupní obrazu X s posunutou množinou B tak, že se původní obraz X rozšíří o množinu B viz Obrázek 7.

  Vzorec: $X \oplus B = \bigcup_{x \in B} X_x$

  Dilatace se používá na zaplnění malých děr. Zvětší se také velikost původního objektu.



Obrázek 7: Ukázka principu fungování dilatace [5]

- **Otevírání** (znak otevírání: $\circ$)

  Otevírání je kombinací předešlých dvou morfologických operací, konkrétně se jedná o erozi, po které následuje ihned dilatace. Pokud je obraz větší než nastavený parametr eroze tak se velikost obrazu nezmění. Je nutné aby obě operace měly stejný parametr, množinu podle které se eroduje nebo dilatuje, protože jinak by došlo ke změně velikosti původního obrazu. Operace otevírání se používá k odstranění menších objektů viz Obrázek 8.

Obrázek 8: Ukázka principu fungování otevírání; zleva: před, po [6]

- **Uzavírání** (znak uzavírání: •)

  Operace uzavírání je opakem otevírání a jedná se o dilataci, po které ihned následuje eroze. Opět je nutné zachování stejného parametru. Uzavírání se používá k vyplnění malých děr v obraze a tudíž dokáže odstranit nežádoucí šum z obrazu, například zrnění viz Obrázek 9.

Obrázek 9: Ukázka principu fungování uzavírání; zleva: před, po [6]

2.5 Vyplnění „děr“ v překážce

Po aplikaci výpočtu průměrné hodnoty a tedy i nalezení překážky, lze na Obrázku 10a vidět, že překážku netvoří celistvý objekt. Objekt naopak obsahuje mezery, což by ve výsledném vyznačení nebylo úplně přínosné a je potřeba tyto mezery zaplnit. K tomuto problému byla vytvořena funkce na vyplňování těchto mezer. Funkce byla inspirována funkcí semínkového vyplňování. Důvod vlastní funkce byl ten, že semínkové vyplňování pracuje s výchozím bodem tzv. semínkovým bodem, ze kterého se následně vyplňování šíří, uživatelem zvolenou barvou. Jelikož však pracujeme s pohyblivým obrazem, je pro nás náročné tento bod najít v každém snímku. Dalším důvodem je, že některé objekty nejsou spojité, tzn. hrany obsahují mezery a těmito mezerami by algoritmus semínkového vyplňování zaplnil celý obraz. Proto byla zvolena jednodušší alternativa pro vyplnění vzniklých mezer.

2.5.1 Algoritmus zaplnění mezer

Algoritmus byl navržen tak, aby nalezl bod patřící překážce a následně vyplnil mezery v překážce. Tento algoritmus pracuje na základě ověřování intenzit pixelu a jeho okolí v hloubkové obraze. Pokud je intenzita okolního pixelu podobná jako intenzita počátečního pixelu, jedná se o jeden objekt, pokud se intenzita liší, jde o hranu, což znamená ukončení překážky. Jak je možné vidět na Obrázku 10 tento algoritmus efektivně zaplní mezery v objektu a přitom nesloučí dva za sebou jdoucí objekty dohromady. Výhodou tohoto algoritmu je, že pracuje s intenzitami pixelů, tzn. že pokud bude ve hraně mezera, nezaplní se celý obraz, jelikož pozadí za objektem už má jinou intenzitu nežli objekt.

(a) Před vyplněním mezer (b) Po vyplnění mezer

Obrázek 10: Rozdíl: Před a po vyplnění mezer

2.6 Nalezení obrysů

Abychom mohli nějakým způsobem zvýraznit překážku, a tím i předejít kolizi, je nutné najít v obraze obrysy požadovaného předmětu. Obrysy, neboli také kontury jsou křivky, které propojují všechny spojité body, se stejnou barvou nebo intenzitou. Nalezení těchto kontur je užitečně zejména pro analýzu tvaru, ale také pro detekci a rozpoznání objektů.

Algoritmus Suzuki85 [7]

Tento algoritmus se používá k vyhledání obrysů objektů v obraze. Než se dostaneme k samotnému algoritmu, je potřeba nejdříve zavést několik pojmů:

1. **Hraniční bod** - Bod, který leží v jednom objektu a okolní bod, který leží v objektu druhém se nazývá hraniční bod.

2. **Okolí mezi spojenými komponenty** - Pokud existuje v jenom objektu takové čtyři pixely, abychom na ně narazili ve všech 4 směrech z druhého pixelu druhého objektu,

potom první objekt obklopuje druhý objekt. Pokud mezi nimi nalezneme hraniční bod obklopují se přímo.

3. **Vnější hranice a hranice mezery** - Vnější hranice je množina hraničních bodů, mezi dvěma přímo se obklopujícími objekty. Podobně hranice mezery je množina hraničních bodů mezi objektem a mezerou.

4. **Mateřská hranice** - Vnější hranice mezi dvěma objekty, přičemž jeden obklopuje přímo druhý, je mateřskou. Pokud je vnější objekt mezera, je mateřskou hranicí hranice mezery mezi objekty, přímo se obklopujícími.

Algoritmus sledující hranici pro topologickou analýzu

Algoritmus rastruje vstupní binární obraz a přeruší rastrování, pokud nalezne pixel splňující podmínku pro hranici sledující výchozí bod buď vnější hranice nebo hranice mezery. Jestliže pixel splňuje obě, výše uvedené podmínky, je považován za výchozí bod vnější hranice. Tuto nově objevenou hranici označíme unikátním identifikační číslem. Během rastrování si také ukládáme číslo pořadí hranice mezery nebo vnitřní, které jsme použili naposledy. Po uložení hranice se rastr opět spustí a pokračuje dál, až nakonec dojde do dolního levého rohu, což značí konec obrazu. Výsledné kontury jsou uloženy jako pole hranic (čar), se kterými můžeme následně pracovat. Tento algoritmus využívá funkce *findContours* z knihovny OpenCV , která s jeho pomocí nalezne obrysy objektů v obraze a uloží je jako vektory (v jakém bodě obrys začíná a v jakém končí).

2.7 Zvýraznění překážky

Aby byl řidič nějakým způsobem informován o blížící se překážce, je nutné tuto překážku vyznačit. Zvolili jsme ohraničení obdélníkem, aby bylo možné v horizontální rovině jasně rozeznat kde překážka začíná a předešlo se tak následné kolizi. K tomu využijeme nalezené hranice z předešlého algoritmu vyhledávajícího obrysy. Z těchto hranic si vezme první a poslední souřadnici jednoho spojitého objektu a následně poskládá obdélník, který objekt ohraničí a tím tak upozorní na překážku.

3 Technické řešení problému

V této kapitole bude ukázáno vyřešení výsledného programu detekce překážek v praxi. Dále tato kapitola obsahuje rozdíl mezi vstupními daty kamer, které byly v průběhu vytváření programu ověřovány.

3.1 Nahrávání dat

Proces nahrávání dat k experimentu probíhal následovně. Na pojízdný vozík byl umístěn notebook, ke kterému byla připojena kamera, snímající prostor před vozíkem. Na chodbě byly poté rozmístěny různé předměty, které měly představovat překážky. Po rozmístění „scény“, bylo zapnuto nahrávání snímků kamerou a vozík touto „scénou“ projížděl. Kamera byla nainstalována na vozík tak, aby mohla snímat jak podlahu, tak i velkou část prostoru před vozíkem. Důvodem proč byla data pořizována uvnitř budovy je samotná kamera, která se používá k nahrávání dat uvnitř, jelikož její obraz zkreslují odlesky slunečních paprsků.

3.2 Vstupní data programu

Jako vstupní data byla použita sekvence obrazů z kamery, které byly uloženy do počítače a poté teprve nahrány do programu na zpracování. Nejedná se tudíž o aplikaci bežící v reálném čase. Jednotlivé obrazy byly ukládány v rozlišení 640x480 pixelů.

3.2.1 Data z kamery Orbecc Astra (Obrázek 11a)

Kamera Orbecc Astra, respektive její data, byla testována jako první. Kamera poskytuje obrazy v dobrém rozlišení, tzn. obraz s ostrými hranami, ve kterém se dá dobře vyhledat objekt i pouhým okem. Nevýhodou této kamery, kterou jsme měli k dispozici, je ten, že dokáže nahrávat v jednu dobu buď barevný RGB obraz a nebo hloubkový obraz. Tudíž nebylo možné výslednou překážku přenést do barevného obrazu, pro lepši představu a vizualizaci řidiči. Hloubkový obraz kamery má dosah 0.6 - 8.0 metrů. Využití kamery je hlavně pro vnitřní účely, jelikož její obraz zkreslují odlesky, např. slunečních paprsků [10].

3.2.2 Data z kamery Intel RealSense R200 (Obrázek 11b)

Tato kamera společnosti Intel má, oproti kameře Orbecc Astra, tu výhodu, že dokáže nahrávat zároveň barevný RGB obraz a hloubkový obraz, což je pro nás výhodou, jelikož můžeme vyobrazovat výslednou překážku v barevném obrazu a tím tak zpřesnit polohu překážky (z pohledu řidiče). Nevýhodou této kamery byl rozostřený obraz, který se ukázal být nepoužitelný. Důvodem byl šum, který byl po celém obraze a náš program nedokázal v takovémto obrazu detekovat žádnou překážku. Kamera poskytuje hloubkový obraz v rozlišení buď 320x240, nebo 480x360 a používá se hlavně pro venkovní účely. Dosah hloubkového obrazu kamery uvnitř budovy 0.5 - 3.5 metrů, dosah hloubkového obrazu kamery venku až 10 metrů [11].

3.2.3 Data z kamery Xbox One Kinect (Obrázek 11c)

Data z této kamery byla, co se týče šumu, oproti předešlé kameře RealSense R200, lepší. Obraz byl v celku bez šumu, ale hloubka nebyla tak dobrá jako z kamery Orbecc Astra. Tím je myšleno, že hrany nebyly výrazné a tudíž předměty mnohdy splývaly s pozadím. To mělo za následek neúplnou detekci, kdy kamera detekovala jenom polovinu překážky a druhou polovinu bral program jako pozadí, což by nejspíš vedlo ke kolizi, při řízení vozidla s touhle kamerou. Kamera Kinect a námi vytvořený program tedy nejsou kompatibilní. Hloubkový obraz kamery má dosah: 0,5 - 4,5 metrů. Kamera poskytuje obraz v rozlišení 512x424 [12].

Shrnutí

Rozhodli jsme se v konečném důsledku pro kameru Orbecc Astra. Důvodem je její kvalitní obraz, ostré hrany, ze kterých se dá snadno vyčíst překážka a oproti kameře Kinect také lepší hloubkový obraz, tj. větší kontrast mezi pozadím, popředím a překážkou, jak je možné vidět na rozdílů mezi Obrázky 11a a 11c. V důsledku využití kamery Orbecc Astra, která má lepší využití uvnitř budovy než vně, z důvodů odlesků slunečních paprsků, jsme se rozhodli testovat obraz na vozíku uvnitř budovy.

(a) Obraz kamery Orbecc Astra (b) Obraz kamery Intel RealSense R200 (c) Obraz kamery kinect

Obrázek 11: Ukázky vstupních obrazů z rozdílných kamer

3.3 Vyhledávání překážky

Jak již bylo zmíněno v teoretické části na tuto problematiku byl použit výpočet aritmetické hodnoty. Algoritmus funguje tak, že spočte sumu všech intenzit pixelů na jednom řádku, které mají intenzitu větší než 5. Pixely s intenzitou menší než 5, jsou považovány za černé, respektive moc tmavé pixely, které by naše měření zkreslilo a překážka by tak nebyla úplná nebo by nebyla nalezena vůbec. Pokud algoritmus najde pixel s intenzitou menší než 5 tak tento pixel odečte od celkového počtu pixelů, které jsou na řádku. Jelikož má obraz rozlišení šířky 640 pixelů, tak od čísla 640 odečteme počet černých pixelů.

V dalším kroku algoritmus vypočítá aritmetickou hodnotu intenzity pixelů na jednom řádku jako poměr součtu hodnot intenzit na jednom řádku ku počtu „nečerných“ pixelů tohoto řádku.

Algoritmus poté porovnává jednotlivé intenzity pixelů daného řádku s jeho aritmetickou hodnotou intenzity. Algoritmus pracuje tak, že zatímco se na jednom řádku vypočítává aritmetická hodnota, na předchozím řádku se porovnávají pixely s již vypočítanou aritmetickou hodnotou tohoto předešlého řádku.

Z experimentálního měření jsme zjistili, že překážky se nacházejí v určité rozmezí tohoto rozdílu. Dále jsme zjistili, že se toto rozmezí liší podle toho kolik bylo na řádku černých pixelů. Pokud bylo na řádku černých pixelů méně než polovina šířky obrazu (menší než 320) je toto rozmezí od 5 do 17. Naopak pokud je počet černých pixelů více než je šířka obrazu (větší než 320 pixelů) rozmezí je od 2 do 8. V ukázce kódu 1 lze vidět podmínku, která rozhoduje jestli je daný pixel překážkou či nikoli.

Navíc je potřeba také zohledňovat samotnou intenzitu pixelu. Pokud by byl pixel až příliš tmavý, jednalo by se o pixel, který patří pozadí. Proto byla intenzita pixelu omezena na hodnotu větší než 25. Pixely, které měly menší intenzitu než je 25 byly zobrazovány v pozadí.

```
if((((AverageValue - delayedValueOfPixel >= 5
     &&
     AverageValue - delayedValueOfPixel <= 17)
     &&
     delayedValueOfPixel >= 25
     &&
     (delayedTotalAmount >= image.cols / 2))
     ||
     ((delayedTotalAmount <= image.cols / 2)
     &&
     ((AverageValue - delayedValueOfPixel >= 2)
     &&
     (AverageValue - delayedValueOfPixel <= 8))
     &&
     delayedValueOfPixel >= 25))
      &&
      (j >= 50 && j <= image.cols - 50)
     )
     { image.at<uchar>(i, j) = 255;
     }
```

Výpis 1: Podmínka rozhodující o přítomnosti překážky

3.4 Algoritmus vyplnění mezer

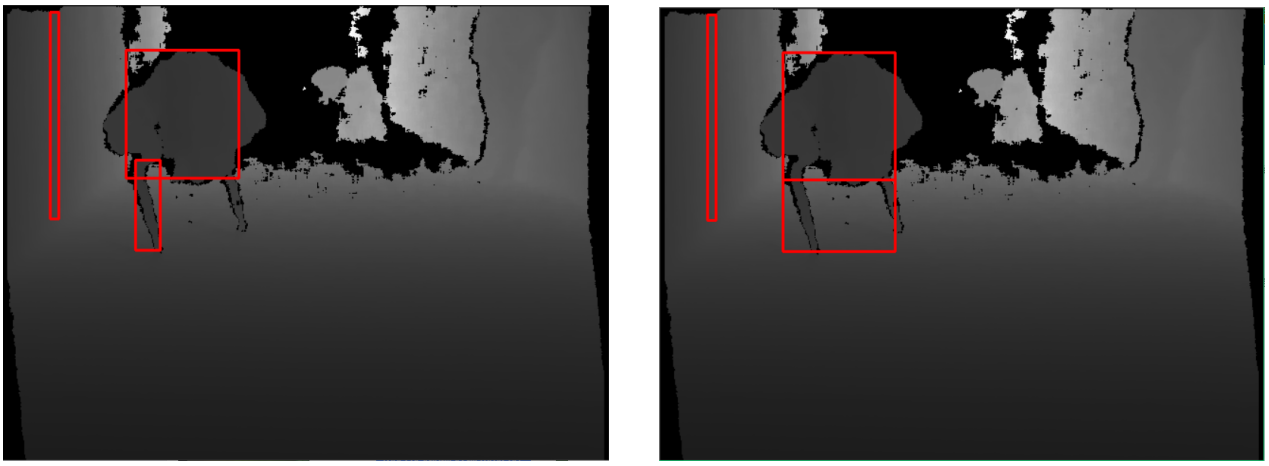
Po nalezení pixelů překážky se některé pixely nevykreslily a je tedy nutné mezery v překážkách vyplnit. Na tuto problematiku byl navržen vlastní algoritmus, který byl inspirován z funkce semínkového vyplňování (floodFill). Tento algoritmus pracuje ve dvou částech. První část má zjistit rozmezí, které bude vybarveno a druhá má za úkol vybarvit dané rozmezí. Rozmezí je nalezeno následujícím způsobem. V obrazu se hledá pixel, který náleží překážce. Po nalezení tohoto pixelu algoritmus nadále hledá, jakou hodnotu intenzity má pixel v původním obrazu. Dále hledá po vertikální ose pixely, které mají rozdíl v intenzitě maximálně jedna. To znamená, že takovýto pixel patří k překážce a můžeme jej započítat do našeho rozmezí, aby byl následně pixel vybarven. Takto pokračuje algoritmus dál, až narazí na pixel, který nebude odpovídat, tj. jeho hodnota intenzity se bude lišit o více než jedna. V takovém případě se jedná o hranu, která ukončuje překážku a algoritmus ukončí hledání dalších pixelů (viz ukázka kódu 2). Dále předá souřadnice pixelů, které se mají vybarvit vykreslovacímu algoritmu, který následně pixely vykreslí a zcelí tak překážku jak je možné vidět na Obrázku 10 v teoretické části.

```
for (int j = 0; j < floodFillingImage.cols; j++)
{  for (int i = 0; i < floodFillingImage.rows; i++)
   {  row = i;
      colorFromFinal = floodFillingImage.at<uchar>(i, j);
      if (colorFromFinal == 255 && i >=1)
      {  finalRow = i;
         colorFromOriginal = grayImage.at<uchar>(finalRow, j);
         stillFinding = true;
         while (finalRow > 0 && stillFinding)
         {  finalRow -= 1;
            colorFromOriginal2 = grayImage.at<uchar>(finalRow, j);
            if (colorFromOriginal - colorFromOriginal2 <=1
            &&
            colorFromOriginal - colorFromOriginal2 >=-1)
            {  colorFromOriginal = colorFromOriginal2;
            }
            else
            {  stillFinding = false;
               break;
            }
            fillDraw(row, finalRow, j, floodFillingImage);
      }}}
```

Výpis 2: Algoritmus zaplnění mezer v objektu

3.5 Nalezení obrysů a zvýraznění překážky

Jakmile máme vyznačenou celou překážku, je potřeba ji pro řidiče zviditelnit. Na tento problém byla použita funkce *findContours*, která vyhledá v obraze vyhledá obrysy a následně je uloží jako vektory. Tyto vektory se používají ve funkci, která je ohraničí obdélníkem. Abychom zamezili zvýraznění příliš malých překážek vypočteme si ze souřadnic obdélníků jejich obsahy a pokud tyto obsahy budou menší než 200 pixelů^2, tyto obdélníky se nevykreslí. Dále je nutné sloučit dohromady obdélníky, které se zobrazují uvnitř nějakého jiného obdélníku. Nejprve jsme si obdélníky seřadili podle velikosti. Poté jsme zkoumali jestli některý z obdélníků nemá společný průnik s některým jiným obdélníkem. Nakonec byly obdélníky, které měly společný průnik sloučeny do jednoho obdélníku, který ohraničuje překážku. Rozdíl v aplikaci tohoto algoritmu je možné vidět na následujícím Obrázku 12



(a) Obraz bez algoritmu na sloučení

(b) Obraz s algoritmem na sloučení

Obrázek 12: Rozdíl v aplikování algoritmu na sloučení obdélníků

4 Experimentální ověření programu

V této kapitole budou popsány jednotlivé experimenty a také software a hardware, na kterých byl program testován. Poté zde budou zhodnoceny výsledky prováděných experimentů a porovnání. Všechna data, na kterých se prováděly experimenty jsou k nahlédnutí na přiloženém DVD.

4.1 Hardware

Při řešení byl použit notebook ASUS GL552VX-CN146T se čtyř jádrovým procesorem Intel i5 o frekvenci 2,30 GHz, RAM pamětí 8 GB a grafickou kartou NVIDIA GeForce GTX 950M s velikostí paměti 2 GB. Dále pak kamera s hloubkovým obrazem Orbecc Astra, ze které byly použity sekvence obrazů na detekci překážek zachycených kamerou.

4.2 Software

Software, který byl použit na vypracovaní systému, byl Visual Studio 2015 v jazyce C++, byly také použity funkce knihovny OpenCV, které pomohly při hledání objektů ze sekvence obrazů z kamery.

4.3 Experimenty

V následující části bude popsáno 5 jednotlivých experimentů, na kterých jsme testovali úspěšnost námi navrhnutého systému.

4.3.1 První experiment

V prvním experimentu jsme použili data z kamery Orbecc Astra. Jedná se o průjezd chodbou mezi několika překážkami. V průběhu videa lze vidět pixely (Obr. 13), které nepatří překážce, ale podlaze. Následně algoritmus vyplnění mezer tyto pixely spojil a zobrazil na stěně falešnou překážku.

(a) Obraz upravený bilaterálním filtrem

(b) Obraz po aplikaci algoritmu vyplnění mezer

(c) Originální obraz s vyznačenou překážkou

Obrázek 13: První experiment: Aplikace vyplňovacího algoritmu na pixely mimo překážku

Na konci videa jsou možné vidět odlesky od podlahy, které jsou způsobeny sluncem. Tyto odlesky vedou k falešným detekcím. Možným řešením je zvýšit parametr u morfologické operace uzavírání (kapitola 2.4.1), abychom se zbavili nežádoucích malých plošek, jak je možné vidět na Obrázku 14a.

(a) Obraz upravený bilaterálním filtrem

(b) Obraz po aplikaci algoritmu vyplnění mezer

(c) Originální obraz s vyznačenou překážkou

Obrázek 14: První experiment: Falešná detekce

Následující skupina obrázků (Obr. 15), ukazuje výsledný obraz s vyznačenou překážkou, kterou program detekuje.

(a) Obraz upravený bilaterálním filtrem

(b) Obraz po aplikaci algoritmu vyplnění mezer

(c) Originální obraz s vyznačenou překážkou

Obrázek 15: První experiment: Detekce celého objektu

4.3.2 Druhý experiment

V tomto experimentu jsme se pokoušeli o detekci bez naklonění kamery, abychom zjistili jak bude program detekovat překážky pokud budeme snímat pouze okolí před vozíkem bez podlahy (Obr. 16).

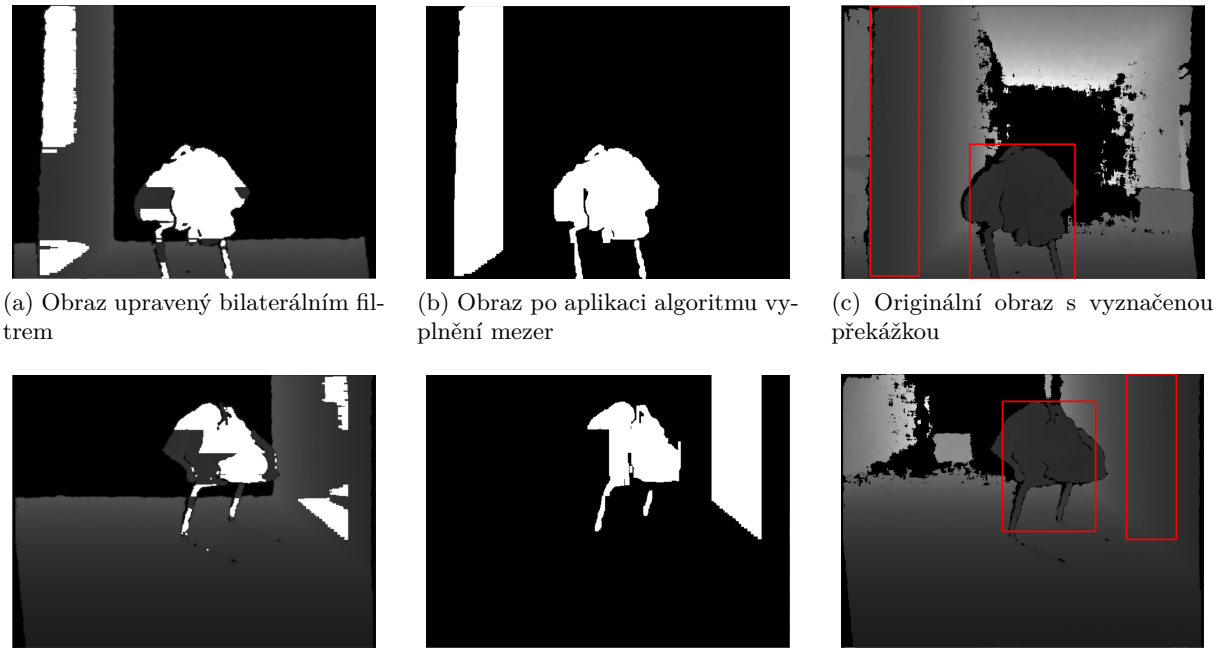
(a) Obraz upravený bilaterálním filtrem

(b) Obraz po aplikaci algoritmu vyplnění mezer

(c) Originální obraz s vyznačenou překážkou

Obrázek 16: Druhý experiment: Detekce bez snímání podlahy

Jak je možné vidět na předchozím Obrázku 16, nebyl tento experiment zcela úspěšný. Objekty nejsou detekovány tak přesně jako s nakloněnou kamerou, tzn. že neznáme velikost objektu u země a to může způsobit kolizi. Ve výsledku jsme dospěli k závěru, že naklonění kamery je nutné, aby měl řidič větší rozhled okolo vozidla. Rozdíl je možné vidět na následujícím Obrázku 17.



(a) Obraz upravený bilaterálním filtrem

(b) Obraz po aplikaci algoritmu vyplnění mezer

(c) Originální obraz s vyznačenou překážkou

(d) Obraz upravený bilaterálním filtrem

(e) Obraz po aplikaci algoritmu vyplnění mezer

(f) Originální obraz s vyznačenou překážkou

Obrázek 17: Druhý experiment: Rozdíl mezi detekcí bez nakloněné a s nakloněnou kamerou

4.3.3 Třetí experiment

U třetího experimentu byla zjištěna falešná detekce, respektive „zvětšení“ překážky. Jedná se o situaci, kdy kamera byla nasměrována šikmo ke zdi a mezi kamerou a zdí, ležela překážka. Výsledkem bylo, že program nalezl jako překážku jak objekt, tak i zeď (Obr. 18). To mělo za následek sloučení obou překážek do jedné a tím i následné „zvětšení“ objektu.

(a) Obraz upravený bilaterálním filtrem

(b) Obraz po aplikaci algoritmu vyplnění mezer

(c) Originální obraz s vyznačenou překážkou

Obrázek 18: Třetí experiment: Sloučení dvou překážek dohromady

Na následujícím Obrázku 19 lze vidět detekci židle s kabátem při průjezdu chodbou.

(a) Obraz upravený bilaterálním filtrem

(b) Obraz po aplikaci algoritmu vyplnění mezer

(c) Originální obraz s vyznačenou překážkou

Obrázek 19: Třetí experiment: Ukázka fungování detekce

4.3.4 Čtvrtý experiment

Při čtvrtém experimentu bylo zjištěno, že algoritmus špatně detekoval překážku, která byla šikmo natočena na kameru. Toto je důsledkem jiné hloubky v objektu, čím je daná část objektu dále od kamery tím je vyšší intenzita pixelu a algoritmus vyhledání překážky tento pixel nezapočítá jako překážku. Toto se však děje pokud je objekt daleko (Obr. 20). Když jsme přijeli s vozíkem blíž, objekt byl detekován celý jak je možné vidět na Obrázku 21.

(a) Obraz upravený bilaterálním filtrem

(b) Obraz po aplikaci algoritmu vyplnění mezer

(c) Originální obraz s vyznačenou překážkou

Obrázek 20: Čtvrtý experiment: Neúplná detekce natočeného objektu

(a) Obraz upravený bilaterálním filtrem

(b) Obraz po aplikaci algoritmu vyplnění mezer

(c) Originální obraz s vyznačenou překážkou

Obrázek 21: Čtvrtý experiment: Detekce blíže k překážce

4.3.5 Pátý experiment

Na pátém a také posledním experimentu, jsme testovali, jak bude vypadat detekce s pohyblivým objektem. Pohyblivý objekt představoval chodec, který procházel „scénou“ před vozíkem (Obr. 22 a 23). Program dokonale zachytil chodce jako překážku, tudíž je možné vyhnout se i pohyblivé překážce.

(a) Obraz upravený bilaterálním filtrem

(b) Obraz po aplikaci algoritmu vyplnění mezer

(c) Originální obraz s vyznačenou překážkou

Obrázek 22: Pátý experiment: Detekce pohyblivého objektu

(a) Obraz upravený bilaterálním filtrem

(b) Obraz po aplikaci algoritmu vyplnění mezer

(c) Originální obraz s vyznačenou překážkou

Obrázek 23: Pátý experiment: Detekce po průchodu chodce

Zhodnocení experimentů

S výslednými experimenty jsme byli spokojeni. V experimentech nebyly nalezeny větší problémy, se kterými by byla spojena kolize vozidla. Naše experimenty byly simulací např. skladiště, ve kterém jsou po zemi různé objekty a do kterých řidič vozidla nesmí narazit. Z důvodu malého rozsahu hloubkového obrazu, tento program není vhodný například pro běžnou jízdu v autě. Program je vhodný využít při malých rychlostech např. ve skladech, při parkování nebo navádění robotů terénem.

V následující části budeme hodnotit přesnost jednotlivých experimentů pomocí statistického měření citlivosti a F1-score.

Experiment	**První**	**Druhý**	**Třetí**	**Čtvrtý**	**Pátý**
Počet správných detekcí	9	16	20	17	13
Počet falešných detekcí	3	2	2	2	3
Počet nesprávných detekcí	0	3	2	8	3
Počet snímků	503	660	832	930	727
Doba zpracování všech snímků	63,86 s	96,21 s	107,77 s	120,65 s	96,17 s
Průměrná doba zpracování jednoho snímku	127 ms	146 ms	130 ms	130 ms	132 ms
Citlivost	75%	89%	91%	89%	81%
F1-score	86%	86%	91%	77%	81%

- **Správná detekce:** Je předpokládána překážka a program ji zobrazil
- **Falešná detekce:** Není předpokládána překážka, ale program ji zobrazil
- **Nesprávná detekce:** Je předpokládána překážka, ale program ji nezobrazil[1]
- **Citlivost** poměr správných detekcí : falešným detekcím
- **F1-score** vypočítáme jako: $\frac{2*SD}{2*SD+FD+ND}$

  SD, je počet správných detekcí; FD, je počet falešných detekcí; ND, je počet nesprávných detekcí

[1] Překážky se v programu nezobrazují hned, ale až po pár sekundách, kdy je hloubka viditelnější. Za nesprávnou detekci považujeme takový případ, kdy i při jasně viditelné hloubce program překážku zobrazí částečně cca z 50% nebo ji nezobrazí vůbec

Z tabulky můžeme vyčíst, že nejpřesnější byl třetí experiment a naopak nejméně přesným byl experiment první. Toto je způsobeno počtem překážek v jednotlivých experimentech a také samotným rozestavěním překážek. Pokud je překážka blízko zdi, tak se výsledná detekce spojí v jeden objekt, místo toho aby byly zobrazeny objekty dva, zeď a objekt. Dále předměty, které jsou natočeny šikmo ke kameře byly detekovány méně než z 50% a byly započteny jako nesprávná detekce a až při bližším snímání kamerou byly objekty detekovány. Co se týče průměrné délky zpracování jednoho snímku, záleží na tom, kolik pixelů musí program zpracovávat. V našem případě to znamená rozdíl mezi průjezdem chodbou rovně, kdy obraz, který se bude detekovat obsahuje hodně černých pixelů, které program nezpracovává a nebo natáčet kameru na zeď a tudíž program bude zpracovávat více pixelů a tím bude delší i čas zpracování.

5 Závěr

V této práci jsme vytvořili program, který detekuje a vyznačuje překážky za pomocí kamery s hloubkovým obrazem. Tento program z jednotlivých obrázků vypočítá odchylku od aritmetické hodnoty v daném řádku a nalezne body překážky. Po nalezení překážky zaplní mezery vzniklé nepřesným měřením odchylek a nakonec celou překážku vyznačí.

Vzniklý program byl poté testován na různých „scénách“, kde byly různě rozmístěny překážky a v posledním experimentu jsme vyzkoušeli i pohyblivý objekt. Ze statistiky úspěšnosti detekce jsme se dozvěděli, že program na detekování překážek funguje nejhůře na 75%, což bylo způsobeno i tvarem a rozmístěním jednotlivých překážek. Z praktického využití kamery jsme také zjistili, že je pro řidiče výhodnější a hlavně bezpečnější, pokud vidí, kde překážka leží na zemi a má o ní celkový přehled. Překážka by mohla být u země široká a nahoře úzká a došlo by ke kolizi.

Využití tohoto systému v praxi by mohla být nejen jízda ve skladu s různými vozíky, které přepravují nějaký obsah, který nechceme poškodit, což jsme se v našich experimentech snažili nasimulovat, ale také jízda v malých rychlostech po parkovišti a nebo také navádění robotů, kteří třeba prozkoumávají a skenují okolí a nechceme je poškodit nárazem. Důvodem, proč není doporučena velká rychlost, je ten, že námi využívané kamery mají malý dosah a ve vyšší rychlosti by řidič, spoléhající se na náš systém neměl tolik času na zareagování na následnou překážku.

Možným vylepšením by byla kamera, která má dobrý hloubkový obraz a zároveň ještě dokáže nahrávat i barevný RGB obraz. Toho by se dalo využít pro lepší představu řidiče o překážce před vozidlem. Pro tento případ by bylo potřeba zjistit o kolik pixelů v obraze se od sebe jednotlivé kamery liší a po vypočítání této hodnoty by se souřadnice obdélníku, který vyznačuje překážku, přenesly do RGB obrazu.

Literatura

[1] Gonzalez, Rafael C. and Woods, Richard E. (2002) - Digital Image Processing

[2] Carlo Tomasi and Roberto Manduchi, “Bilateral filtering for gray and color images,” in Computer Vision, 1998. Sixth International Conference on . IEEE, 1998, pp. 839– 846.

[3] Václav Hlaváč - Matematická morfologie, ČVUT v Praze, Fakulta elektrotechnická, katedra kybernetiky

`http://cmp.felk.cvut.cz/~hlavac/TeachPresCz/11DigZprObr/71-3MatMorpholBinCz.pdf`

[4] Robert Fisher, Simon Perkins, Ashley Walker and Erik Wolfart - Erosion

`http://homepages.inf.ed.ac.uk/rbf/HIPR2/erode.htm`

[5] Robert Fisher, Simon Perkins, Ashley Walker and Erik Wolfart - Dilate

`http://homepages.inf.ed.ac.uk/rbf/HIPR2/dilate.htm`

[6] Opencv Closing, Opening documentation

`http://docs.opencv.org/2.4/doc/tutorials/imgproc/opening_closing_hats/opening_closing_hats.html`

[7] Satoshi Suzuki and Keiichi Abe, "Topological structural analysis of digitized binary images by border following", In Computer Vision, Graphics, and Image Processing, Volume 30, Issue 1, 1985, pp. 32-46

[8] OpenCV documentation page

`http://docs.opencv.org/trunk/index.html`

[9] OpenCV Thresholding documentation

`http://docs.opencv.org/trunk/d7/d4d/tutorial_py_thresholding.html`

[10] Orbecc Astra documentation

`https://orbbec3d.com/product-astra/`

[11] Intel RealSense r200 documentation

`https://software.intel.com/en-us/articles/realsense-r200-camera`

[12] Kinect Xbox One documentation

`https://developer.microsoft.com/en-us/windows/kinect/hardware`

A Příloha

Obsah přiloženého DVD

Součástí této práce je také přiložené DVD

- tato bakalářská práce v Elektronické podobě
- testovací vstupní data z experimentů
- zdrojový kód pro aplikaci Visual Studio 2015 v jazyce C++
- knihovna OpenCV, kterou jsme v programu využili