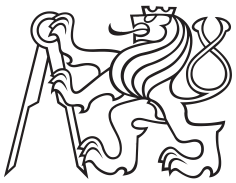


České
vysoké
učení technické
v Praze

Katedra řídicí techniky

Autonomní robot SK8O

Tomáš Bártík

Vedoucí: Ing. Krištof Pučejdl
Květen 2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Bártík** Jméno: **Tomáš** Osobní číslo: **492013**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra řídicí techniky**

Studijní program: **Kybernetika a robotika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Autonomní robot SK8O

Název bakalářské práce anglicky:

Autonomous SK8O robot

Pokyny pro vypracování:

Cílem práce je implantace úlohy SLAM (simultánní lokalizace a mapování) na robotu Sk8o, využívající data z hloubkové kamery Intel Realsense. Daný systém by měl umožňovat autonomní bezkolizní pohyb robota s využitím plánování.
1) Seznamte se s fungováním hloubkové kamery Intel Realsense D455 a propojte ji se stávajícím řídicím systémem robota.
2) Implementujte metody na lokalizaci, mapování a plánování. Výběr algoritmů a jejich implementaci přizpůsobte existujícímu hardware a software v robotu.
3) Otestujte funkčnost vaší implementace na vhodném scénáři projetí naplánované trajektorie.

Seznam doporučené literatury:

[1] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza. "Introduction to Autonomous Mobile Robots (2nd. ed.)," The MIT Press, 2011
[2] C. Cadena et al., "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age," in IEEE Transactions on Robotics, vol. 32, no. 6, pp. 1309-1332, Dec. 2016
[3] Steven M. LaValle, "Planning Algorithms," Cambridge University Press, USA, 2006

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Krištof Pučejdl katedra řídicí techniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **31.01.2022** Termín odevzdání bakalářské práce: **20.05.2022**

Platnost zadání bakalářské práce:
do konce letního semestru 2022/2023

Ing. Krištof Pučejdl	prof. Ing. Michael Šebek, DrSc.	prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta

Poděkování

Děkuji vedoucímu své bakalářské práce Ing. Krištofovi Pučejdlovi za vedení práce a konzultace a za technickou pomoc s provozem robota. Dále děkuji Ing. Janu Bayerovi za konzultace.

Prohlášení

I declare that this work is all my own work and I have cited all sources I have used in the bibliography.

Prague, května 19, 2022

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 19. May 2022

Abstrakt

V této práci používáme hloubkovou kameru Realsense k implementaci úlohy SLAM na dvoukolovém balancujícím robotovi. Jsou prozkoumány různé metody lokalizace, je vybrána metoda ORB-SLAM, dále je implementováno mapování, plánování a řízení pro robota Sk8o. Každá část je následně testována.

Klíčová slova: ORB-SLAM,dvounohý kolový robot, visuální odometrie, mapování,RGB-D,SK8O

Vedoucí: Ing. Krištof Pučejdl

Abstract

This bachelor's thesis has the goal of implementing the SLAM task on a stereo depth camera Realsense from Intel. First we research different localization methods, we integrate the ORB-SLAM method into our pipeline and then we implement mapping, planning and control of the robot Sk8o. Each part is then tested for reliability.

Keywords: ORB-SLAM,bipedal wheeled robot, visual odometry, mapping,RGB-D,SK8O

Title translation: Autonomous SK8O robot

Obsah

Obrázky

Tabulky

Kapitola 1

Úvod

V bakalářské práci navrhujeme systém pro autonomní pohyb dvoukolového balancujícího robota Sk8o, navrženého a postaveného na začátku minulého roku 2021 vedoucím této práce Ing. Krištofem Pučejdlem a Ing. Martinem Gurtnerem. Díky vývoji hloubkových (stereo) kamer z posledních let a s ní spojených algoritmů pro vizuální odometrii jsme schopni připojením této kamery do existujícího robota výrazně rozšířit jeho schopnosti. Robot, který před touto prací byl schopný pouze pohybu na základě referencí z ovladače, bude schopen autonomního pohybu v neznámém prostředí.

Nejprve popíšeme kameru Realsense a další hardware a software, se kterým pracujeme. Následně zjistíme, jaké metody Simultánní lokalizace a mapování (SLAM) jsou nám k dispozici a na jakém principu pracují. Popíšeme souřadné systémy využité v této práci, řešení separace překážek od podložky (podlahy) a jak ovlivní parametry zkreslení naší lokalizaci. Dále přejdeme k samotné lokalizaci. Tato lokalizace je obtížná, především kvůli pohybovým vlastnostem našeho robota, kdy při náklonu nebo otáčení způsobeného regulační smyčkou může docházet k neostrosti a tedy nepřesnosti vytvořeného pointcloudu (soubor datových bodů v určité souřadné soustavě). Díky vytvoření pointcloudu jsme schopni zmapovat své okolí do reprezentace mřížky obsazenosti, ve které lze poté použít plánovací algoritmy na nalezení optimální trasy. Nakonec musíme navrhnout regulátor, který bude schopen řídit Sk8o podél této trasy. Každou z těchto částí práce je potřeba průběžně testovat a na konci provést experimenty na verifikaci.

Kapitola 2

Kamera Realsense D455

V této bakalářské práci využíváme pro lokalizaci a mapování hloubkovou kameru od společnosti Intel typu Realsense D455. [7], volba a nákup kamery proběhla před počátkem mé práce na tématu bakalářské práce. Kamera pracuje v několika módech jako RGB senzor, hloubkový senzor, IR senzor a IMU senzor (Inertial Meassurement Unit). Všechny senzory kamera nabízí v několika různých rozlišeních (od 256×144 až po 1280×800) a různých snímkových frekvencích (od 5 Hz až po 90 Hz). Naše aplikace probíhala na jednodeskovém počítači Odroid s velmi limitovaným výpočetním výkonem, tudíž bylo třeba zvolit dostatečnou frekvenci, aby lokalizace pomocí ORB-SLAMu probíhala spolehlivě, ale zároveň příliš nezatěžovala počítač Odroid. Byla zvolena frekvence 15 Hz a rozlišení hloubkové a RGB kamery jako 640×480. Podrobnější parametry kamery jsou v tabulce:

Tabulka 2.1: Paramtery Realsense D455

hloubkový rozsah	0.4 až 10 m
přesnost	$< 2\%$ ve vzdálenosti 4 m
připojení	USB-C 3.2
zorné pole	$90° \times 65°$
vzdálenost mezi sensory	95 mm

Pro práci s kamerou je také relevantní ohnisková a kardinální vzdálenost:

Tabulka 2.2: Parametry kamery f,c

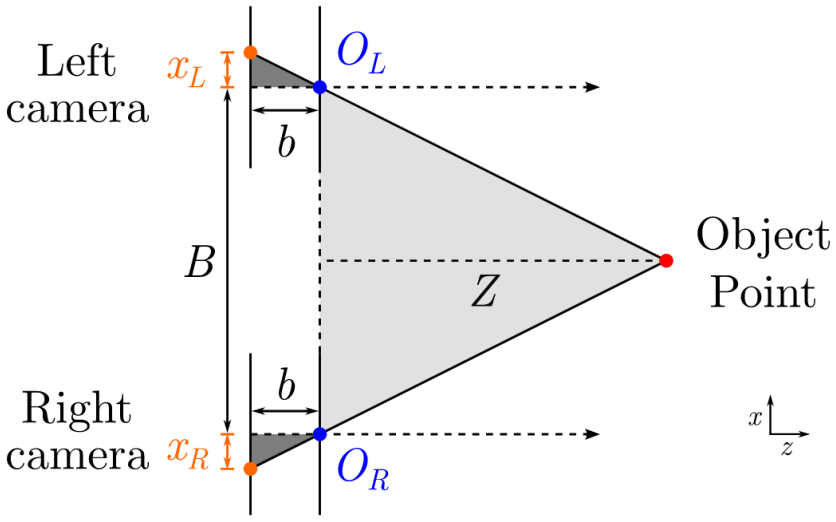
f_x	387.045
f_y	387.045
c_x	321.626
c_y	241.316

Kromě kamer řady D400, kterou používáme v této bakalářské práci, má Intel také zajímavou řadu T265, která má oproti té naší implementovanou metodu SLAM přímo na čipu kamery. Je schopna pomocí dat z kamery na 15 Hz a dat z IMU na 200 Hz počítat vizuální lokalizaci a výsledky pak posílat do počítače robota.

Obrázek 2.1: Kamera Realsense D455[1].

2.1 Princip fungování hloubkové kamery

Hloubkové kamery přidávají k obvyklým 3 kanálům dat RGB čtvrtý kanál, popisující vzdálenost objektu na pixelu od kamery. Naše hloubková kamera má 2 senzory, vzdálené 95 mm od sebe. Realsense zpracuje snímky z obou kamer a rozdíly na nich ji umožní zjistit jak je každá část snímku vzdálená. Kamera defakto napodobuje způsob jakými lidské oči získávají informaci o vzdálenosti objektů. Bližší objekty se mezi očima budou lišit více, oproti tomu velmi vzdálené objekty budou mít jen nepatrný rozdíl ve své pozici mezi dvěma snímky. Kamery Realsense mají navíc infračervená světla, která osvětlí zachycovaný prostor a umožní, že jsou kamery schopny provádět identifikaci hloubky i v neosvětlených prostorech.



Obrázek 2.2: Metoda triangulace pro určení hloubky objektu, převzato [2].

V případě stereoskopických kamer (jako D455) se hloubka obrazu určuje na základě triangulace. Používají se 2 kamery rovnoběžné mezi sebou. Bod v prostoru se promítne do obou kamer pod různým úhlem, což způsobí rozdílnou vzdálenost bodu od středu obrazu x_R, x_L. Vzdálenost bodu je poté nepřímo úměrná tzv. *disparitě*, definovanou jako $\Delta x = x_R - x_L$. Z toho vyplývá, že

hloubkové kamery jsou kvůli měření disparity přesnější blíže ke kameře, s roustoucí vzdáleností přesnost klesá. Pro naši kameru je tento rozsah od 0.4 m až do 10 m. Na obrázku (2.2) máme znázorněný optický střed kamer O_L, O_R, tzv. *baseline* hloubkové kamery B, tedy vzdálenost mezi dvěma kamerami (u naší kamery D455 je to 95 mm), a vzdálenost obrazu od středu kamer b. Z toho vyplývá vztah pro vypočítání vzdálenosti Z použitím podobnosti trojúhelníků.[2]

$$\frac{Z}{B} = \frac{b}{\Delta x} \tag{2.1}$$

Větší detaily o implementaci nejsou společností Intel zveřejňovány.[8]

Tento princip vytváření hloubkového snímku není jediný, který se v podobných aplikacích používá. Nejznámněšním je pravděpodobně LiDAR. Ten tvoří mapu vysíláním laserových paprsků do svého okolí a měří dobu, za kterou se každý z paprsků vrátí zpět. Na základě rychlosti světla lze tedy dopočítat vzdálenost, kterou každý paprsek urazil. Oproti hloubkovým kamerám jsou výrazně dražší a neposkytují takové rozlišení.

Samotný Intel nabízí navíc senzory SR300 založené na principu tzv. *Structured light*, kde na scénu promítá vzory v infračervené oblasti. Poté kamerou detekuje, jak se vzory na objektech deformují a z toho získává informaci o vzdálenosti těchto objektů.

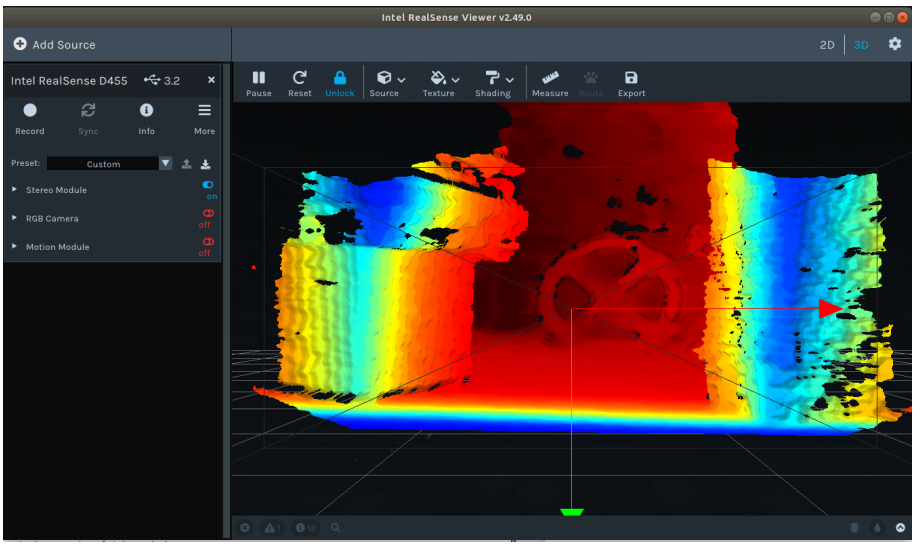
2.2 Jednodeskový počítač Odroid

Obrázek 2.3: Počítač Odroid N2+[3].

Robot Sk8o používá k řízení počítač Odroid N2+ od firmy Hardkernel. Počítač na základě měření senzorů balancuje celého robota pomocí LQR regulátoru, zpracovává reference na pohyb z Xbox kontroléru a řídí drivery motorů. Pro bakalářskou práci potřebujeme na robotovi provádět lokalizaci pomocí ORB-SLAM s frekvencí 15 Hz a s periodou $t = 1.5$ s provádět mapování a plánování. Vše je zároveň třeba provádět bez grafické karty. Na desce je procesor ARM Quad-core Cortex-A73 s 2.4 Ghz. Bootování probíhá z eMMC karty. Pro komunikaci s počítačem jsem na něm nastavil WiFi access point.

2.3 Vývojové prostředí pro kamery Realsense

Společnost Intel poskytuje vývojové nástroje (SDK) pro práci s kamerami Realsense. Pro základní odladění funkce kamery je určen Realsense Viewer. Můžeme zde prohlížet data ze RGB a Stereo kamery a z IMU. Kromě grafického rozhraní je k dispozici knihovna `librealsense2` napsaná v C++, která poskytuje objekty a funkce pro komunikaci s kamerou, přijímání datových toků a úpravu výsledných dat na samotném počítači. Komunikace probíhá přes objekt `rs2::pipeline` s nastavením, které z datových toků mají být posílány. Celkem kamera D455 nabízí 435 datových toků (různá rozlišení, formáty dat, frekvence). Data se převedou do formátu v OpenCV pro další práci. Zároveň má knihovna zabudovanou funkci pro výpočet pointcloudu z hloubkového snímku pomocí `rs2::pointcloud::calculate`. Tyto body jsou pochopitelně vyjádřené v souřadném systému kamery. V předchozí práci jsem



Obrázek 2.4: Grafické prostředí Realsense viewer.

pracoval s implementací této knihovny v Pythonu, ale pro tuto práci by to bylo výpočetně nedostatečné, a proto jsem zvolil vývoj v jazyce C++.

2.4 Sk8o

Robot Sk8o je dvounohý balancující robot, volně inspirovaný robotem Ascento vyvinutým na univerzitě v Curychu. Pro udržení svislé pozice robota byl pro něj navržen LQR regulátor pro motory. Rozhraní robota našemu programu umožňuje nastavovat reference na *velocity* (dopřednou rychlost) a *yaw rate* (úhlovou rychlost natočení). Před touto prací na něm proběhla jedna další bakalářská práce zaměřená na tvorbu matematického modelu pomocí Newton-Eulerovy metody a odvození pohybových rovnic. Při tvorbě tohoto robota na něm nebyl implementován systém ROS, což sebou přináší určitá zjednodušení

v řízení a základní práci s robotem, avšak při práci na této bakalářské práce jsme narazili na několik úskalí, která naopak práci s kamerou a body pointcloudu ztížila a v ROSu by byla řešitelná rychleji.

Obrázek 2.5: Robot Sk8o.

Kapitola 3
Metody SLAM

Základním stavebním blokem pro tvorbu autonomních robotů jsou metody SLAM (metody simultánní lokalizace a mapování). Tyto metody umožňují lokalizovat robota vůči nějaké světové referenci, jejich výstupem je typicky transformační matice v homogenních souřadnicích pro transformaci bodů ze souřadného systému kamery do referenčního/světového souřadného systému. Po získání transformační matice lze tedy všechny body zasadit do světového souřadného systému, což již umožňuje tvorbu mapy, případně mřížku obsazenosti (tzv. occupancy grid). V takové reprezentaci se již dá provádět plánování cesty k požadované souřadnici cíle. [9]

3.1 Algoritmus ORB-SLAM

Algoritmus pro lokalizování může využít řadu senzorů. Používá jednak RGB snímky a jednak hloubkové snímky. [10] Je schopen také pracovat s daty z IMU (akcelerometru a gyroskopu). ORB-SLAM má několik vláken, kde každá plní část úkolu SLAM. V hlavní smyčce algoritmu jsou hledány klíčové objekty (ORBy), které lze mezi jednotlivými snímky korelovat a na základě toho určit translaci a rotaci mezi těmito dvěma snímky (časovými okamžiky). Počet bodů, které se vlákno snaží extrahovat z každého snímku je definován nastavitelnou proměnnou. Typicky se jedná o 1000 ORB features. Pro naše potřeby byla tato hodnota zvýšena na 2000, jelikož blízko u překážek Sk8o často ztrácel lokalizaci, proto nebyl schopen korelovat body na snímku s body v paměti, tedy nenašel 6-DOF pozici (DOF - degrees of freedom, stupně volnosti). Pro vyšší počet ORB features už nastával problém s výpočetním výkonem Odroidu při frekvenci snímků 15 Hz.

Pro budoucí lokalizaci se poté vybírají pouze některé snímky (*keyframes*), které jsou dále tříděny, aby zůstaly jen ty nejrelevantnější a minimalizovala se paměťová náročnost. Na to je používána strategie tzv. *Survival of the fittest.* [10]

Důležité vlákno ORB-SLAMu je tzv. loop closing, tedy pokud se kamera dostane do stejné pozice jako v některém z předchozích snímku, tak by se měly 6-DOF souřadnice shodovat. Kvůli nepřesnostem v lokalizaci podél trajektorie se souřadnice ale nebudou shodovat. V takovém případě vlákno loop closing rozdistribuuje odchylku rovnoměrně podél celé trajektorie. I když

na první pohled se toto vlákno nezdá být relevantní pro úlohu této bakalářské práce, když je úkolem dostat se na předem definované místo a neočekává se, že Sk8o bude procházet již prošlé místo, přesto je potřeba. Na začátku úlohy, když sk8o teprve vytváří mapu, nalezne cestu přes levou/pravou část mapy, ale po natočení identifikuje překážky, zanese je do mapy a cesta tudy již neexistuje. Proto se vrátí na původní místo, přes které plánuje jinou cestu k cíli. Z toho důvodu je relevantní použít metodu SLAM se spolehlivou metodou loop closingu.

V naší práci používáme variantu algoritmu pro RGB-D. To používá pro určení transformace mezi dvěma po sobě jdoucími snímky algoritmus nazývaný ICP (Iterative Closest Point). Zkráceně řečeno hledá transformaci, která minimalizuje vzdálenost mezi dvěma pointcloudy. [11]

ORB-SLAM nabízí několik parametrů svého algoritmu, které upravují funkčnost algoritmu a náročnost na výpočetní výkon. Mezi ně patří:

Tabulka 3.1: ORB parametry ORB-SLAMu

Number of features per image	2000
scale factor between leves of scale pyramid	1.2
levels in the scale pyramid	8

Hlednání správných kalibračních parametrů a parametrů ORB-SLAMu bylo jedno z věcí, které při tvorbě této práce zabraly poměrně hodně času a zpomalily postup touto prací.

3.2 Další algoritmy

Kromě zde používaného ORB-SLAMu lze pro naši úlohu vybrat i jiné známé algoritmy. Tyto algoritmy se dělí do tří skupin dle způsobu jakým získávají posun robota mezi dvěma snímky (frames). Zde zavedená metoda používá *keyframes*, jejichž porovnáváním s aktuálním snímkem získáme souřadnice robota. ORB-SLAM již byl vytvořen ve 3 verzích, poslední byla publikována v prosinci. Z důvodu větší spolehlivosti jsme zde použili druhou verzi.

Nově vyvíjená metoda SLAM je založena na použití hlubokých konvolučních neuronových sítí (CNN). To jsou neuronové sítě, které používají v některých svých vrstvách konvoluci vstupního obrazu s konvolučním jádrem, což jsou ty trénované parametry. Tyto sítě mají na vstupu 2 po sobě jdoucí snímky, a na výstupu z nich dostáváme posun robota v prostoru. Tyto metody jsou zatím ve vývoji a dle článku porovnávajícího SLAM metody[12] zatím nenahradily jiné metody, které nejsou tvořené jako *Black box* a lze tedy popsat na jakém principu provádí lokalizaci. Nejznámějším takovým algoritmem je RatSLAM.

Třetím typem SLAM metod jsou ty založené na popisování pozice robota jako pravděpodobností funkce. Dále pak využívají Kalmanovy filtery, Rao Blackwellised Particle filtery[12] k získání výstupní lokalizace. Mezi tyto metody patří například EKF SLAM, Graph SLAM nebo Fast SLAM.

Kapitola 4

Souřadné systémy

V této práci používáme několik souřadných systémů a jejich tranformací pro splnění úlohy mapování. Modelem dírkové komory popisujeme práci kamery, dále vytváříme transformační matici pro separaci podlahy od relevantních bodů (překážek) a nakonec popisujeme vliv distorčních parametrů na úlohu lokalizace.

4.1 Homogenní souřadné systémy

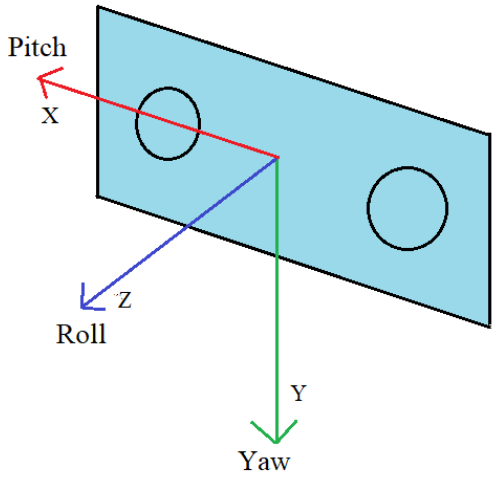
Při práci s pointcloudy pracujeme s homogenními souřadnými systémy. Jejich princip je jednoduchý, pro 3D prostor vypadá převod z Kartézských souřadnic do homogenních a zpět takto:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \rightarrow \begin{pmatrix} x/w \\ y/w \\ z/w \end{pmatrix} \tag{4.1}$$

V našem případě uvažujeme $w = 1$. Tento systém nám umožňuje provádět transformace souřadnic, které obsahují rotace i translace, a následně tyto transformace řetězit za sebe, abychom provedli transformaci z výchozího souřadného systému kamery, ve kterém získáváme body pointcloudu, do souřadného systému výsledné mapy.

4.2 Souřadné systémy použité na robotovi

Orientace souřadnicových systémů v této práci je převzatá ze souřadného systému kamery Realsense, znázorněného na obr. 4.1. Z toho vyplývá, že dopředná osa je osa z, osa doprava je osa x a osa směrem do podložky je osa y. Tento formalismus je zachován pro všechny souřadné systémy zmíněné v této práci. Výslednou souřadnou soustavou je soustava mapy, která se používá pro tvorbu occupancy gridu.



Obrázek 4.1: Souřadný systém kamer Realsense.

4.3 Model dírkové komory

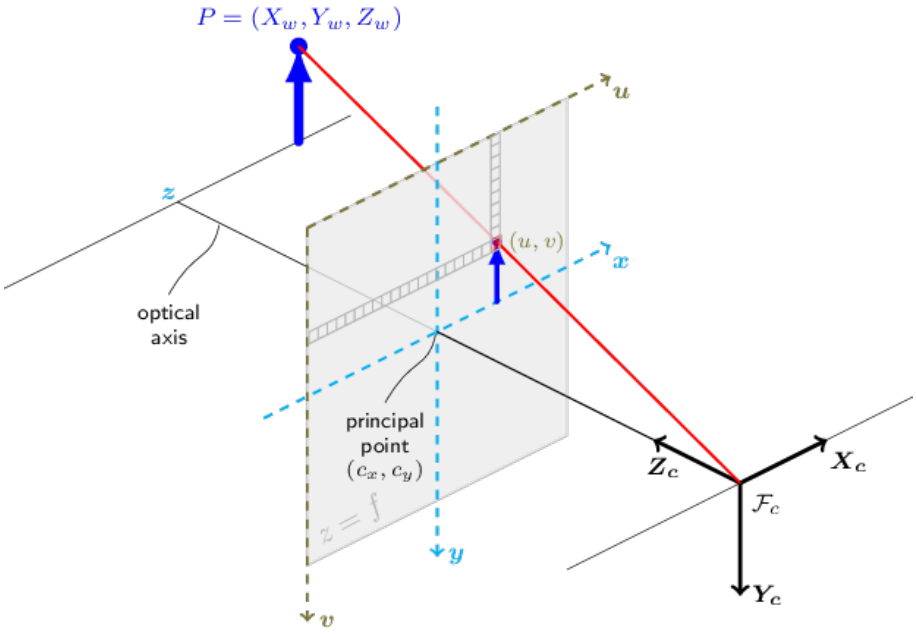
Projekci bodů ve 3D prostoru do roviny obrazu v kameře lze popsat matematickým modelem dírkové komory. Parametry tohoto modelu jsou pak relevantní pro správnou funkci algoritmu lokalizace. Dle obrázku modelu dírkové kamery (obr. 4.2) máme 3 souřadné systémy (kamery, obrazu a světové) a transformace mezi nimi. Mezi souřadnicemi obrazu a kamery je transformační matice A, která popisuje, jak se bod v pozici vůči kameře promítne do souřadnic obrazu. Skládá se z ohniskových vzdáleností kamery f_x, f_y a z hlavního bodu kamery c_x, c_y.

$$p_{obrazu} = A p_{kamery} \tag{4.2}$$

$$A = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{4.3}$$

Hloubková kamera nám umožňuje tento model neuvažovat a dává nám (dle dříve popsané metody) k dispozici body v souřadném systému kamery. Mezi souřadnicemi kamery a světovými/referenčními je pak transformace, kterou získáváme díky lokalizaci za pomoci ORB-SLAMu.

Zvolený algoritmus ORB-SLAM použije souřadný systém kamery na začátku lokalizace jako výchozí a prohlásí ho za světový souřadný systém (tedy transformační matice po lokalizaci na prvním snímku je jednotková matice). Při následném pohybu kamery/robota se ORB-SLAM lokalizuje vůči právě tomuto souřadnému systému. Z algoritmu získáváme transformační matice $T^{\text{kamery}}_{\text{světový}}$, avšak zde potřebujeme tu opačnou/inverzní. Tu získáme ze znalosti



Obrázek 4.2: Model dírkové komory pro kameru[4].

transformací homogenních souřadnic:

$$T^{\text{kamery}}_{\text{světový}} = \begin{pmatrix} \mathbf{R} & \begin{matrix} t_x \\ t_y \\ t_z \end{matrix} \\ \mathbf{0} & 1 \end{pmatrix} \tag{4.4}$$

$$T^{\text{světový}}_{\text{kamery}} = \begin{pmatrix} \mathbf{R}^{-1} & -R^{-1}t \\ 0 & 1 \end{pmatrix} \tag{4.5}$$

4.4 Rovnoběžnost s podložkou

Zde jsem narazil na sice zřejmý, ale podstatný problém. Tím je separace bodů pointcloudu na podlaze od bodů zbytku mapy, které lze použít k navigaci. Úloha by byla jednoduchá pokud by světový referenční souřadný systém byl ve dvou osách rovnoběžný s podlahou. To ovšem nelze zaručit. Světový souřadný systém vychází z pozice/orientace kamery při prvním snímku lokalizace. I když je balancování robota dobré, nelze v žádném případě spoléhat na rovnoběžnost kamery. Řešením je vytvořit nový souřadný systém mapy, kde bude osa x a z rovnoběžná s podlahou. Pro transformaci mezi těmito dvěma systémy potřebujeme znát normálový vektor podlahy ve světovém souřadném systému. Systém pak rotujeme tak, aby osa y byla zarovnána s normálovým vektorem. Tím získáme souřadnicový systém mapy, ve kterém již lze separovat podlahu od bodů použitelných k navigaci. Normálový vektor získáme pomocí 3 bodů na podlaze. Každý ze 3 určujících bodů získáme zprůměrováním několika

bodů pro získání větší robustnosti této metody. Je rozumné předpokládat, že robot nebude začínat čelem ke zdi a bude mít před sebou dostatečné místo kde lze identifikovat body na podlaze. Ze 3 bodů se vytvoří 2 vektory, které už definují rovinu rovnoběžnou s podlahou. Vektorovým součtem získáme kolmý normálový vektor $\vec{a}$. Ten chceme rotovat na bázový vektor osy y, tedy $\vec{b} = \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}^T$. Použijeme Rodriguesové vzorec pro rotační matice:

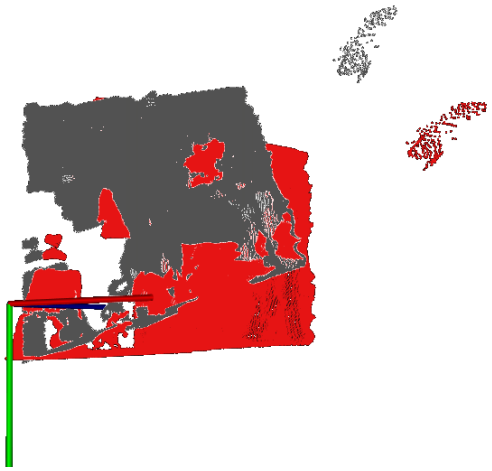
$$n = \vec{a} \times \vec{b} \tag{4.6}$$

$$s = \|n\| \tag{4.7}$$

$$c = a \cdot b \tag{4.8}$$

$$R = I + [n]_\times + [n]_\times^2 \frac{1-c}{s^2} \tag{4.9}$$

Symbol $[n]_\times$ zde značí antisymetrickou matici. Body podlahy, ze kterých tvoříme normálu musejí být dostatečně vzdáleny od okraje hloubkového snímku, jelikož zkreslení na okraji hloubkového snímku způsobovalo výraznou chybu a výsledný souřadný systém mohl být i hůře zarovnán s podlahou než ten bez této korekce. Zvolili jsme 3 body v přibližném trojúhelníku o stranách 10 cm, vzdálené 50 cm od robota, popsané v pixelových souřadnicích. Dle obrázku (4.3) můžeme vidět červený pointcloud zarovnaný s osou x a osou z v bílém pointcloudu. Zároveň můžeme v pointcloudu vidět skupiny bodů, které kamera realsense nedokázala správně identifikovat. Jedná se o lesklý povrch na tabuli.



Obrázek 4.3: Rotace pointcloudu do nové soustavy rovnoběžné s podložkou.

4.5 Závislost na parameterech zkreslení

Parametry zkreslení existují dvojího druhu. Jednak máme radiální zkreslení, které způsobí změnu euklidovské vzdálenosti bodů od středu obrazu. To se projeví tím, že přímky se v obrazu nebudou jevit jako přímky, jak je ilustrováno

na obrázku 4.4. Jednak tangenciální zkreslení, které způsobí natočení bodů o úhel vzhledem ke středu (viz obr. 4.5). Závislost obrazových souřadnic bodů na parametrech zkreslení lze vyjádřit jako:

$$x_d = x + x(k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 xy + p_2(r^2 + 2x^2), \tag{4.10}$$

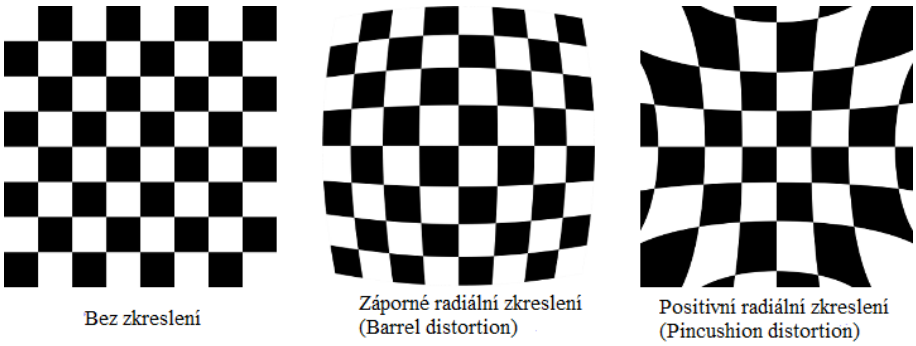
$$y_d = y + y(k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_2 xy + p_1(r^2 + 2y^2), \tag{4.11}$$

,kde (x, y) je bod před korekcí a (x_d, y_d) je bod po ní a r je euklidovská vzdálenost od středu souřadného systému $r = \sqrt{x^2 + y^2}$. První část obsahující k-parametry popisuje radiální zkreslení a druhá část s p-parametry popisuje tangenciální zkreslení. Zpravidla má radiální zkreslení větší efekt než tangenciální. Dohromady tyto parametry způsobí nepřesnou transformaci bodů na okrajích snímku a tedy způsobí nepřesnou identifikaci význačných bodů (ORBů) pro lokalizaci v ORB-SLAMu. Kamery Realsense jsou při dodání již továrně nakalibrovány, ale je doporučeno před používáním kalibraci provést znovu. Použijeme tedy nástroj v kameře Realsense, který toto umožňuje, a naměřené paramtery zkreslení ORB-SLAMu následně předáme.

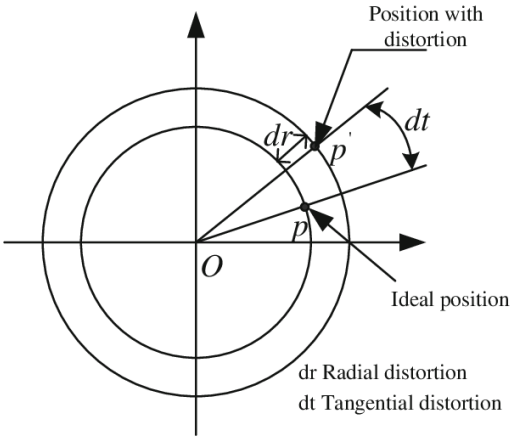
k_1	-0.05456
k_2	0.06447
k_3	-0.02019
p_1	-0.00032
p_2	-0.00068

Z hodnot jsme zjistili, že tangenciální zkreslení je pro tuto kameru o 2 řády nižší než radiální.

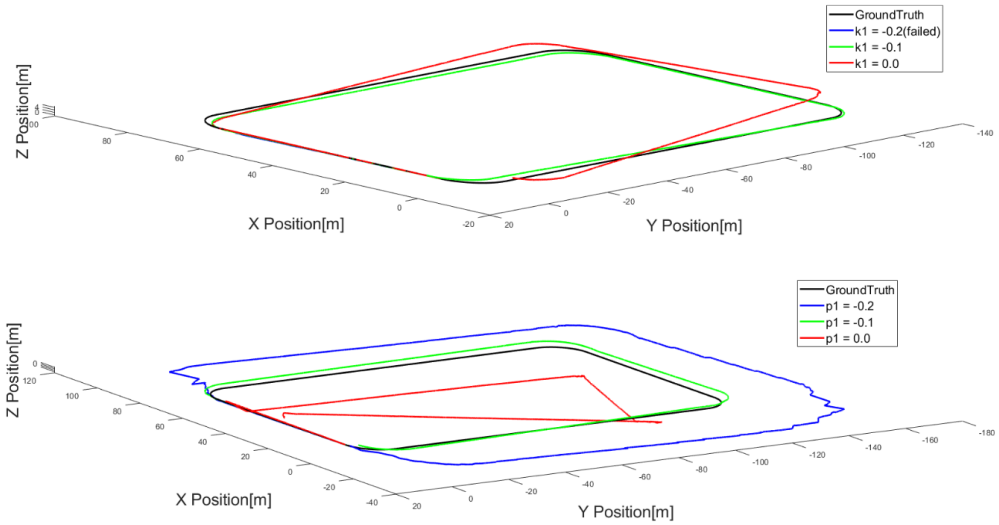
To jak výrazně se mění lokalizace systému ORB-SLAM v závislosti na velikosti parametrů zkreslení bylo analyzováno v práci z Univerzity v Shanghai [6], jejich výsledky jsou na obr. 4.6. Experiment provedli taktéž na ORB-SLAM pro různě chybné parametry zkreslení a zjistili, že čím horší je odhad parametrů, tím hůře probíhá lokalizace a může vést až k selhání ORB-SLAMu. I když rozdíly v tangenciálních parametrech vedly k větším chybám, rozhodně lze tangenciální zkreslení naší kamery prohlásit za zanedbatelné, jelikož jsme identifikovali parametry o 2 řády nižší. Kamera Realsense je tedy převážně ovlivněna radiálním zkreslením.



Obrázek 4.4: Radiální zkreslení, převzato od [4] a upraveno.



Obrázek 4.5: Radiální a tangenciální zkreslení, převzato od [5].



Obrázek 4.6: Analýza závislosti parametrů zkreslení na lokalizaci, převzato od [6].

Kapitola 5
Metoda ORB-SLAM

5.1 Řešení limitů metody

Systém ORB-SLAM má zásadní limit při vysoké rychlosti otáčení. Kamera na robotu Sk8o je při normálním pohybu zpravidla otáčena ve dvou směrech. Jednak při natáčení směru jízdy (*yaw*) v ose y (rovnoběžně s podložkou). Zároveň, jelikož se jedná o dvoukolového balancujícího robota, tak se pro vykonání dopředného pohybu Sk8o natáčí v ose x (tedy se natáčí dopředu pro vykonání dopředného pohybu). ORB-SLAM při vysoké úhlové rychlosti ztratí lokalizaci a přestane určovat transformační matici, ale ještě předtím vznikne problém pro mapování. Při vyšší úhlové rychlosti vlivem odchylky lokalizace kamery a vlivem nepřesného určení bodů v hloubkovém snímku (zřejmě vlivem pohybové neostrosti (motion blur)) budou body v pointcloudu transformované do světového souřadnicového systému úhlově posunuté, což způsobí že v occupancy gridu budou některé objekty zaneseny na více místech a tím přestane být grid pro plánování použitelný. Z toho důvodu ze znalosti rotační matice (vzaté z transformace kamerového souřadného systému na ten světový) získáme úhly natočení kamery ve všech třech osách obecně známé jako *yaw, pitch, roll*.

$$R = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \tag{5.1}$$

$$\alpha_{\text{yaw}} = \arctan(r_{21}/r_{11}) \tag{5.2}$$

$$\alpha_{\text{pitch}} = \arctan(-r_{31}/\sqrt{r_{32}^2 + r_{33}^2}) \tag{5.3}$$

$$\alpha_{\text{roll}} = \arctan(r_{32}/r_{33}) \tag{5.4}$$

Tyto úhly poté použijeme k vypočítání úhlové rychlosti v těchto osách. V případě překročení limitů na některé z os (především na prvních dvou, osa *roll* by se při současné konstrukci robota neměla výrazně měnit) poté odložíme algoritmus mapování a plánování (probíhající periodou $t = 1.5\,\text{s}$) do doby než úhlové rychlosti opět vyhovují limitům. Pro omezení otáčení *yaw* sice saturujeme výstup regulátoru pro motory na *yaw rate* (viz kapitola Sledování reference robotem), ale jedná se pouze o referenci, tedy překmit může snadno dostat úhlovou rychlost přes limit.

5.2 Test spolehlivosti ORB-SLAM

Pro porovnání přesnosti a spolehlivosti lokalizace systému ORB-SLAM potřebujeme nějakou referenci. V prvním pokusu bude touto referencí poloha pomocí kombinace odometrie a IMU. Toto zatím nebylo na robotovi vytvořeno, proto to nyní implementujeme zde. Pro referenční sledování polohy použijeme průměr dat na odometrech obou kol $h(t)$. Z IMU použijeme hodnotu Eulerova úhlu α_{yaw}, který měří úhel v osách rovnoběžných s podložkou. V diskrétních bodech měření odometru zjistíme změnu $h(t)$, tuto změnu polohy potom integrujeme ve směru α_{yaw}. Tuto metodu lze spojitě zapsat, pro dopřednou osu z a kolmou osu x:

$$z(t) = \int_0^t \frac{dh(t)}{dt} \cos(\alpha_{yaw}(t))dt \tag{5.5}$$

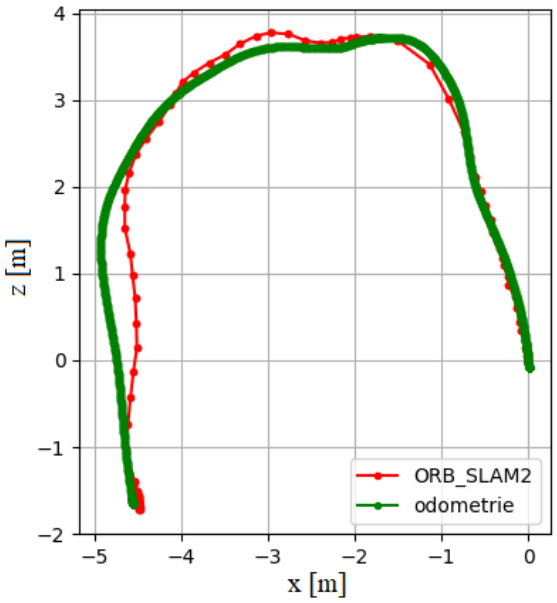
$$x(t) = \int_0^t -\frac{dh(t)}{dt} \sin(\alpha_{yaw}(t))dt \tag{5.6}$$

Tato metoda se ukázala jako spolehlivá pro krátká měření a porovnávání s metodou ORB-SLAM, avšak pro delší měření se začal projevovat drift hodnot z IMU, tedy α_{yaw}, a lokalizace pomocí odometrie přestala být vypovídající.

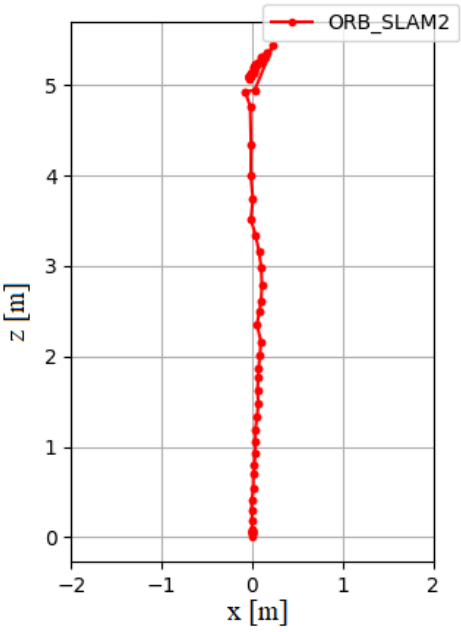
Dalším problémem naší implementace odometrie bylo právě průměrování hodnot na kolech. Tím jsme ztratili část informace. Pokud bychom uvažovali každé kolo zvlášť mohli jsme implementovat spolehlivou metodu odometrie, jak je například prezentována zde [13]. Avšak pro naše potřeby porovnání s ORB-SLAMem nám implementace vystačila.

První experiment, který jsme provedli bylo objetí místnosti laboratoře. Robota jsem kontroloval přes xbox ovladač a snažil jsme se projet co nejhladší cestu okolo místnosti. Cílem experimentu bylo porovnat obě metody lokalizace podél trajektorie. Tento experiment (obr. 5.1) ukázal relativní přesnost ORB-SLAMu v kancelářském prostředí se spoustou příznaků (ORB features), které metoda může použít ke zpřesnění lokalizace.

Na dalším experimentu jsme se pokusili ověřit, že jednotky v systému ORB-SLAM odpovídají metrům. To jsme provedli řízením po rovné čáře dlouhé 5 m. Řízení proběhlo opět manuálně. Jelikož je obtížné startovat robota natočeného přesně po čáře, aby souřadný systém z ORB-SLAMU měl osu z rovnoběžnou s čárou, byla provedena korekce bodů na trajektorii (pouze) rotační maticí. Dle výsledku (obr. 5.2) můžeme prohlásit, že systém ORB-SLAM je spolehlivý, co se týče jednotek v souřadném systému. Jelikož byl robot během experimentu řízen manuálně, došlo na konci k překmitu z cílového bodu.



Obrázek 5.1: Lokalizace podél trajektorie, použitím systému ORB-SLAM a odometrie.



Obrázek 5.2: Lokalizace podél rovné čáry, pro porovnání jednotek.

Kapitola 6

Mapování okolí

6.1 Tvorba mřížky obsazenosti

Z kamery Realsense dostáváme hloubkový snímek 640×480, tedy 370200 bodů. Pro některé z těchto bodů se kameře nepodaří určit hloubku, takže reálně bude bodů méně. Pro plánování trasy potřebujeme okolí robota označit na buď volné body nebo na body s překážkou. K tomu si budeme vytvářet occupancy grid (mřížku obsazenosti), tedy diskrétní reprezentace okolí na tzv. buňky (*cells*) s informací zda se v ní nachází překážka nebo nikoli (v příští sekci přidáme informaci zda již s jistotou víme, že tam není překážka nebo jen jsme zatím v tom směru kamerou neprohledávali). Pravděpodobnost rozdělení na gridu lze vyjádřit jako $p(g)$. Provádíme předpoklad (a zjednodušení), že pravděpodobnost každé buňky je nezávislá na ostatních (což reálně neplatí, jelikož překážky obvykle zabírají více než jednu buňku). Z toho:

$$p(g) = \prod_{i,j} p(g_{i,j}) \tag{6.1}$$

V této práci jsme zvolili 2D diskretizaci prostoru, jelikož nepracujeme s robotem schopným překonávat výškové překážky a tudíž není třeba rozdělovat prostor do buněk ve 3D. Zároveň uvažujeme body pouze ve výškách (y souřadnicích) relevantních pro plánování robota (0.1 m*až* − 1 m (y souřadnice míří dolů od kamery)). Při každé periodě mapování použijeme pointcloud v souřadné soustavě mapy (viz kapitola 4) a do každé buňky započítáme body pointcloudu, které do ní patří. Uvažujeme pouze body nad podlahou a do 1 m nad robotem, jelikož to jsou jediné relevantní.

Jelikož u kamery Realsense je potřeba počítat s šumem, musíme nastavit práh počtu bodů, od kterého se bude uvažovat buňka za obsazenou. Toto je dobře vidět na obr. 4.3, kde vidíme skupinu bodů mimo obrysy místnosti, způsobené lesklou tabulí. Jak poté vidíme na obr. 6.1, tyto body se nepropíší do occupancy gridu. Pro naše potřeby jsme zvolili buňky o rozměrech 10 cm × 10 cm. Vzhledem k velikosti místností pro pokusy jsme používali grid/mapu o celkovém rozměru 20 m × 20 m. Přesnější grid nebyl zvolen z důvodu nepřesnosti ORB-SLAMu na našem robotovi.

Celkově mapování zabere $t \approx 80$ ms na Odroidu. Nejdelší čas trvá výpočet pointcloudu ze snímku použitím knihovny `librealsense` a funkce

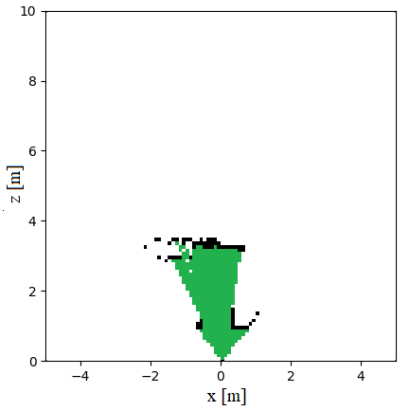
`calculate_depth()`. Transformace do souřadnic mapy a vytvoření gridu již nemá pro výpočet větší význam. Z tohoto důvodu probíhá smyčka pouze s periodou $t = 1.5\,\text{s}$.

Tabulka 6.1: Časy částí programu na Odroidu

Výpočet pointcloudu z hloubkového snímku	67 ms
Transformace souřadnic	10 ms
Zasazení bodů do gridu	1 ms

Jelikož Sk8o má nenulové rozměry, je potřeba s tím počítat pro plánování trasy. Proto z occupancy gridu vytváříme nafouknutý grid. Zde okolo každé obsazené buňky vytvoříme 2 buňky široké pole, které označíme stejně jako obsazené buňky. Okolo každé obsazené buňky tedy vytvoříme 5×5 pole obsazených buněk. Díky tomu bude okolo každé překážky alespoň 20 cm velká mezera, kde plánovací algoritmus nebude plánovat trasu. Vzhledem k šířce robota 32 cm (poloměr 16 cm) je toto dostatečné k zajištění bezkoliznosti s překážkami. Je možné neobsahovat rohové body, ale potom bychom mohli garantovat jen 14 cm vzdálenost od překážky, což je nedostatečné.

6.2 Bresenhamův algoritmus



Obrázek 6.1: Mapování míst bez překážky (znázorněno zeleně).

Pro tvorbu occupancy gridu potřebujeme označit všechny buňky, o kterých s jistotou víme, že jsme na nich nedetekovali překážku. To můžeme říci o buňkách nacházejících se mezi robotem a detekovanou překážkou. K tomu použijeme Bresenhamův algoritmus pro přímky, který se v grafice používá pro vykreslení přímky mezi dvěma pixely (v našem případě buňkami). Zajistíme, aby algoritmus začínal kreslit na buňce, kde se nachází robot (a kamera), a skončil na místě překážky. V případě, že během vykreslování narazí na překážku, tak je vykreslení zastaveno. Toto provedeme pro každou nově přidanou překážku do occupancy gridu. Výsledné přímky kreslíme do separátního gridu, který nám říká, přes které buňky na mapě je možné plánovat trasu.

Jak je patrné z obrázku 6.1 (použit pouze jeden frame pro tvorbu gridu), tak algoritmus správně vyplňuje všechen prostor mezi kamerou a překážkou jako volný. Poté při plánování se nejprve pokusíme nalézt cestu přes buňky označené jako volné a pouze pokud žádnou nenalezneme, tak spustíme plánování přes všechny buňky, kde není překážka.

6.3 Tvorba mapy na základě Bayesova filtru

Přestože předchozí metoda mapování vytváří occupancy grid správně, u některých překážek označovala okolní buňky jako obsazené. V této části vytvoříme robustnější metodu odvozenou od Bayesovy věty, popsanou v knize Probabilistic Robotics [14], v kapitole 4. Mapa pracuje s pravděpodobnostním modelem, kde místo s překážkou značíme $p(g_{i,j}) = 1$, 0 znamená buňka bez překážky, 0.5 je neprozkoumaná. Pokud máme pozorování mapy v časech od 1 až po $t-1$ značeno jako $z_{1:t-1}$, a chceme znát pravděpodobnostní rozdělení naší mapy za předpokladu pozorování $z = z_{1:t}$, tak pravděpodobnostní rozdělení můžeme vyjádřit rovnicí

$$p(g_{i,j}|z_{1:t}) = \frac{p(g_{i,j}|z_t)p(z_t)p(g_{i,j}|z_{1:t-1})}{p(g_{i,j})p(z_t|z_{1:t-1})}. \tag{6.2}$$

Jelikož předpokládáme, že je měření časově nezávislé, můžeme zjednodušit na $p(z_t|z_{1:t-1}) = p(z_t)$.

Pravděpodobnost opačného jevu potom bude

$$p(\neg g_{i,j}|z_{1:t}) = \frac{\neg p(g_{i,j}|z_t)p(z_t)p(\neg g_{i,j}|z_{1:t-1})}{\neg p(g_{i,j})p(z_t)}. \tag{6.3}$$

Tyto dvě rovnice poté lze vydělit a zlogaritmovat, tím získáme tzv. *log-odds ratio*

$$\log \frac{p(g_{i,j}|z_{1:t})}{\neg p(g_{i,j}|z_{1:t})} = \log \frac{p(g_{i,j}|z_t)}{\neg p(g_{i,j}|z_t)} + \log \frac{p(g_{i,j}|z_{1:t-1})}{\neg p(g_{i,j}|z_{1:t-1})} + \log \frac{\neg p(g_{i,j})}{p(g_{i,j})}. \tag{6.4}$$

Když logaritmické poměry pravděpodobností označíme jako l:

$$l(g_{i,j}|z_{1:t}) = l(g_{i,j}|z_t) + l(g_{i,j}|z_{1:t-1}) - l(g_{i,j}) \tag{6.5}$$

První člen napravo označuje nové informace, které jsme získali o mapě v čase t. Druhý popisuje informace, které jsme měli k dispozici do $t-1$. A poslední člen je počáteční odhad pravděpodobnosti $p(g_{i,j}) = 0.5$, tedy log-odds ratio bude rovno 0. Součtem předchozího odhadu a nové informace získáme aktualizovaný odhad pravděpodobnosti výskytu překážky.

$$l_{\text{nový odhad}} = l_{\text{nová informace}} + l_{\text{původní odhad}} \tag{6.6}$$

Pokud je buňka mimo dohled kamery (neprotala jí žádná Bresenhamova přímka a není na ní žádný bod z pointcloudu), žádné nové informace jsme nezískali a $l = 0$. Pokud detekujeme na buňce skupinu bodů, přičítáme k l hodnotu proporcionální k počtu bodů. Pokud buňku protne Bresenhamova přímka, vracíme zápornou hodnotu l.

6.4 BFS

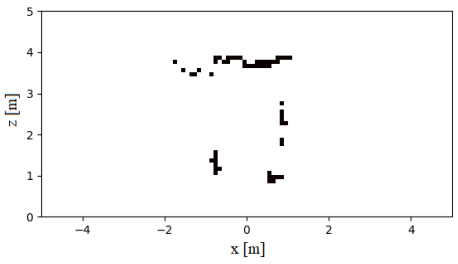
Pro plánování trasy přes occupancy grid používáme Breadth First Search (BFS). Ten nám zaručí vždy nalezení optimální trasy (ve smyslu vzdálenosti v occupancy gridu) z aktuální pozice robota, do předem zadaného cíle, popsaného souřadnicemi v occupancy gridu. Časová složitost algoritmu je $O(|V| + |E|)$, kde $|V|$ je počet vrcholů v grafu (v tomto případě počet buněk m^2) a $|E|$ je počet hran (zde jde o $8|V|$ jelikož používáme osmiokolí). Celkem je tedy časová složitost $O(m^2 + 8m^2) \approx O(m^2)$. Pro náš grid trval výpočet na Odroidu $t \approx 1\,\mathrm{s}$.

V případě, že by trvalo plánování příliš dlouho, případně pokud bychom používali menší velikost buněk, lze nahradit BFS algoritmem RRT.

6.5 Verifikace mapování

Dále potřebujeme ověřit, že naše tvorba mřížky obsazenosti je dostatečně přesná, aby v ní bylo možné plánovat trasu a tu pak sledovat. Předpokladem již zde je správně pracující a ověřená lokalizace. Opět jsme manuálně kontrolovali robota, podél trajektorie podobné jako na obr. 5.1, a očekávali jsme, že získáme odpovídající půdorys laboratoře.

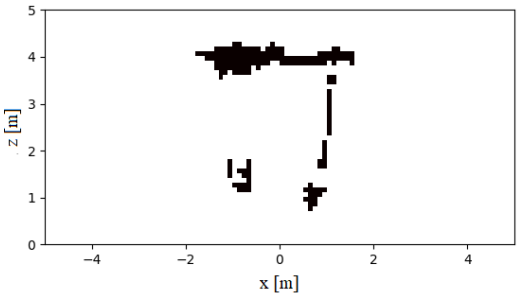
Nejprve jsme vytvořili occupancy grid na základě pouze jediného hloubkového snímku. Na obrázku lze vidět základní obrysy laboratoře (obr. 5.1). Pokud necháme mapování probíhat po několik period, uvidíme, že se do gridu po 5 snímcích (7.5 s) zasadí i úzké a malé objekty. Každopádně pro plánování je použitelný již první snímek, kde jsou patrné obrysy všech předmětů.



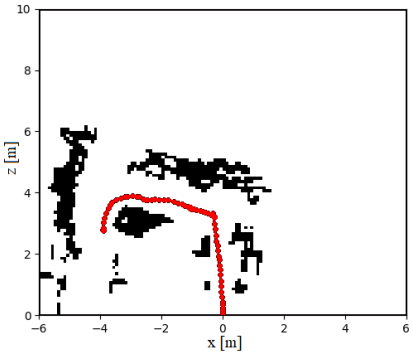
Obrázek 6.2: Mapování okolí při stacionární pozici (balancování na místě), použit jediný snímek.

Tento pokus nám dokázal, že je Sk8o schopen namapovat své okolí dostatečně přesně, aby byl schopen nalézt v dané mapě trasu, kterou pak dokáže Sk8o sledovat. Zároveň lze na výsledku vidět, že objekty, které byly během trasy zakryty, nebo na které nebyl Sk8o natočený, nejsou na mapě zobrazeny.

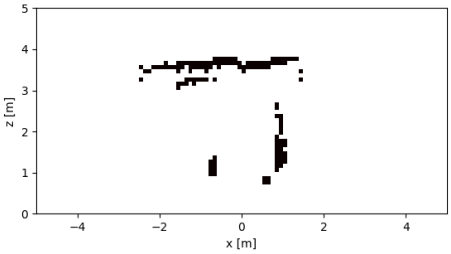
Nakonec otestujeme naši novou metodu mapování založenou na Bayesově



Obrázek 6.3: Mapování okolí při stacionární pozici (balancování na místě), 7.5 s mapování (více snímků).



Obrázek 6.4: Mapování okolí robota podél trajektorie (znázorněná červeně).



Obrázek 6.5: Mapování okolí robota Bayesovou metodou.

filtru. Z experimentu (obr. 6.5) je vidět, že metoda mapuje stejně dobře jako naše původní, ale jsou na ni jednoznačněji vidět hrany překážek.

Kapitola 7

Sledování reference robotem

7.1 Návrh řízení

Z plánovacího algoritmu dostáváme posloupnost bodů v occupancy gridu začínající na místě kamery v momentě provedení plánovacího algoritmu a vedoucí k cílovému bodu. Náš algoritmus je založen na metodě nazývané *carrot following/carrot tracking* [15]. Aktuální poloha kamery (a tedy i robota) je odlišná od polohy kamery v době plánování, jelikož plánování (a tvorba occupancy gridu) probíhá s periodou $t = 1.5\,\mathrm{s}$ (dané omezením výpočetního výkonu na Odroidu), zatímco řízení probíhá ve stejné smyčce, a tedy i stejné frekvenci, jako lokalizace pomocí ORB-SLAM, tedy na $f = 15\,\mathrm{Hz}$. Zároveň plánování probíhá pouze pokud nejsou překročeny limity rychlosti rotace kamery (viz kapitola 6), takže je třeba, aby řízení fungovalo nezávisle na plánování. Pouze pokud ORB-SLAM ztratí lokalizaci, tak je třeba přerušit řízení a na motory posílat nulové reference.

Řízení nejprve najde referenční bod v posloupnosti cesty, který je na kružnici 50 cm nebo blíže od kamery. V případě dvou průsečíků vybíráme ten pozdější v posloupnosti (tedy blíže k cíli). Tento bod používáme jako referenci k řízení natočení robota. Porovnáváme úhel natočení kamery/robota v ose x a ose z s úhlem natočení referenčního bodu a regulační odchylka je potom použitá v regulátoru, jehož výstupem je akční zásah, což je posíláno jako reference úhlové rychlosti (v řídícím systému značeno jako *yaw rate*) na motory v řídícím systému robota. Regulátor je saturován na hodnotu ± 1.0, což pro robota znamená maximální rychlost otáčení $30°/\,\mathrm{s}$, děláno z důvodu lokalizace na ORB-SLAM, která při vysoké rychlosti otáčení lokalizaci ztratí.

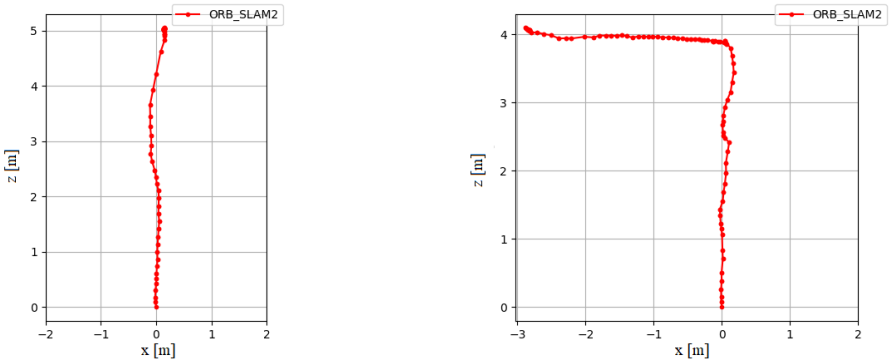
Regulátor na reference dopředné rychlosti na motorech (v systému značeno *velocity*) je závislý na velikosti odchylky úhlu natočení. Pokud je odchylka úhlu nulová, tak se saturuje dopředná rychlost na hodnotě 8, což odpovídá 25 cm/s, opět kvůli limitům lokalizace a mapování. Za regulátor je zařazený filtr prvního řádu k vyhlazení referencí posílaných na drivery motorů. Jedná se o průměrování přes třetinu sekundy. Sk8o poté nereaguje příliš rychlými změnami náklonu (*pitch*).

7.2 Verifikace řízení

Řízení je zcela závislé na lokalizačních datech poskytnutých systémem ORB-SLAM, zde testujeme pouze smyčku řízení, proto porovnáváme zadanou referenční trasu (v souřadnicích mapy) se skutečnou trasou dle lokalizace.

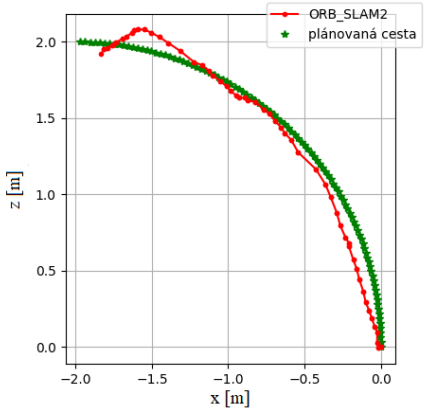
První zkouškou bylo prosté projetí rovné trasy dlouhé 5m. Největší odchylka podél této trasy byla 10 cm. Důležité je, že řízení nemá žádnou ustálenou regulační odchylku. Druhou zkouškou byla trasa 4 m dopředu a 3 m doleva. Zde během zatáčení řízení provedlo největší chybu, konkrétně 17 cm.

Jelikož se jedná o balancujícího robota, který kromě sledování referencí dopředné a úhlové rychlosti zároveň vyrovnával náklon robota přes regulátor LQR, vždy bude vznikat podél trasy chyba daná akčním zásahem z druhého regulátoru.



(a) : Řízení robota podél 5 m rovně. **(b) :** Řízení robota podél 4 m rovně, 3 m doleva.

Obrázek 7.1: Test řízení



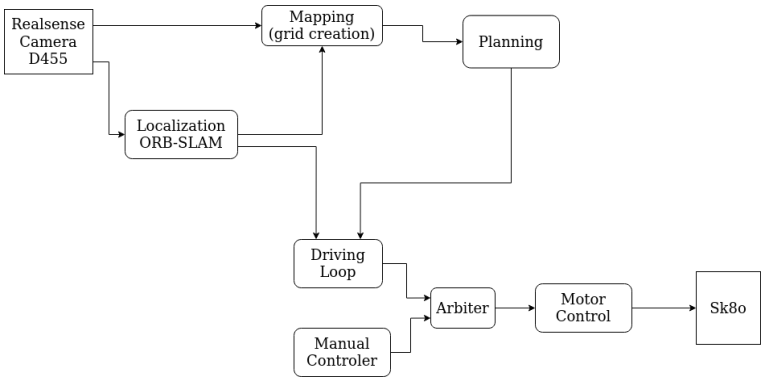
Obrázek 7.2: Řízení robota podél kružnice 2 m poloměrem.

Posledním experimentem byla kružnice s poloměrem 2 m. Zde se již více projevovaly nepřesnosti řízení robota paralelně s balancováním. Přestože měl

Sk8o sledovat kružnici, tak překmit při balancování způsobil odchylku od trasy, kterou pak náš regulátor musel vyrovnat. Přesto zde byla největší odchylka 13 cm.

Kapitola 8

Implementace



Obrázek 8.1: Ilustrace architektury našeho řídícího algoritmu

V předchozích kapitolách jsme podrobně popsali vývoj a teorii za jednotlivými částmi funkčních bloků naší bakalářské práce. V této kapitole shrneme implementaci částí do celkového programu řídícího robota Sk8o.

Základem práce je vyčítání snímků z kamery Intel Realsense D455, konkrétně vyčítáme obrazové a hloubkové snímky na frekvenci 15 Hz. Do lokalizace ORB-SLAM, která je typu RGB-D, posíláme oba snímky a na stejné frekvenci z ní dostáváme transformační matici $T_{\text{kamery}}^{\text{světový}}$, kterou posíláme jednak do řídicí smyčky, kde je používána k nalezení polohy a orientace robota (souřadnice x, z, α_{yaw}). V mapovací části přijímáme hloubkové snímky, které knihovna *librealsense* transformuje na pointcloud, a zároveň matici transformace z lokalizace, abychom mohli body převést do souřadné soustavy mapy a provést aktualizaci mřížky obsazenosti. Mřížka obsazenosti se poté použije pro naplánování trasy, popsané jako posloupnost bodů v mřížce. Toto společně s lokalizací nám dá dostatek informací pro naši řídicí smyčku k posílání referencí na motory (na stejné frekvenci jako přicházejí snímky). Pro umožnění řízení vybraných reference buď z ovladače manuálně nebo z našeho autonomního řízení jsme předřadili Arbiter, který tuto funkci plní. Motor kontroléry (implementovány nezávisle na této práci) pak kombinují reference na pohyb s požadavkem balancování a řídí otáčky motorů.

Kód této práce je k nahlédnutí v repositáři na mém školním gitlabu https://gitlab.fel.cvut.cz/bartito2/orb-slam2_for_sk8o. Práci jsem imple-

mentoval dovnitř naklonované knihovny ORB-SLAM2 [16]. Zároveň jsou vytvořeny nejrůznější skripty na tvorbu grafů a posílání *lcm* komunikačních zpráv.

Kapitola 9

Závěr

Cílem této bakalářské práce bylo použít RGB-D kameru Intel Realsense na implementaci úlohy SLAM a následné řízení robota.

Zvolili jsme tedy metodu ORB-SLAM pro úlohu lokalizace a vyhodnotili její přesnost na robotovi Sk8o v laboratorním prostředí. Zde se nám podařilo verifikovat přesnost systému, přestože bylo potřeba provést několik úprav pro zajištění větší spolehlivosti na balancujícím robotovi. Dále jsme úspěšně implementovali algoritmus mapující okolí robota, dostatečně robustně, aby bylo možné ve výsledné reprezentaci provádět bezkolizní plánování. Zde jsme zvolili dva přístupy, kde oba se ukázaly jako použitelné pro naši úlohu, avšak ten druhý se ukázal jako robustnější. Nakonec jsme vytvořili řídicí smyčku pro robota Sk8o pro sledování trasy naplánované na mapě. Toto řízení se ukázalo jako robustní, jelikož maximální odchylka během experimentů byla 17 cm.

Poslední součást bakalářské práce byla otestování našeho systému na projetí neznámé trasy. V posledním týdnu, kdy jsme prováděli experimenty se objevil problém na motorech a Sk8o nebyl schopný správně balancovat. Zda je to problém s drivery motorů či nějaký jiný, není doposud zřejmé, každopádně poslední sadu experimentů jsme bohužel nebyli schopni provést. V momentě, kdy bude závada vyřešena, bych chtěl tyto experimenty rozhodně dokončit.

Literatura

[1] Introducing the intel realsense depth camera d455. https://www.intelrealsense.com/depth-camera-d455/, Nov 2021.

[2] Christopher Hahne, Amar Aggoun, Vladan Velisavljevic, Susanne Fiebig, and Matthias Pesch. Baseline and triangulation geometry in a standard plenoptic camera. *International Journal of Computer Vision*, 126:1–15, 01 2018.

[3] Odroid n2+. https://wiki.odroid.com/odroid-n2/hardware.

[4] Camera calibration and 3d reconstruction. https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html.

[5] Yunting Li, Jun Zhang, Wenwen Hu, and Jinwen Tian. Laboratory calibration of star sensor with installation error using a nonlinear distortion model. *Applied Physics B: Lasers and Optics*, 115:561–570, 2014, 09 2013.

[6] Yifan Zhu, Ling Pei, Zou Danping, Wenxian Yu, Tao Li, Qi Wu, and Songpengcheng Xia. *A Geometric Consistency Model of Virtual Camera for Vision-Based SLAM Simulation*, pages 271–280. 02 2021.

[7] Intel. *Intel Realsense Product Family D400 Series*, 06 2020. Rev. 9.

[8] Beginner's guide to depth (updated). https://www.intelrealsense.com/beginners-guide-to-depth/, May 2020.

[9] Arthur Huletski, Dmitriy Kartashov, and Kirill Krinkin. Evaluation of the modern visual slam methods. In *2015 Artificial Intelligence and Natural Language and Information Extraction, Social Media and Web Search FRUCT Conference (AINL-ISMW FRUCT)*, pages 19–25, 2015.

[10] Raul Mur-Artal, J. Montiel, and Juan Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31:1147 – 1163, 10 2015.

[11] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Mobile Robot Localization*, pages 265–367. 2011.

[12] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, Jose Neira, Ian Reid, and John Leonard. Simultaneous localization and mapping: Present, future, and the robust-perception age. *IEEE Transactions on Robotics*, 32, 06 2016.

[13] Abhishek Jha and Manoj Kumar. Two wheels differential type odometry for mobile robots. In *Proceedings of 3rd International Conference on Reliability, Infocom Technologies and Optimization*, pages 1–5, 2014.

[14] S. Thrun and American Association for Artificial Intelligence. *Probabilistic Robotics*. American Association for Artificial Intelligence, 2000.

[15] Robert Hogg, Arturo Rankin, Stergios Roumeliotis, Michael McHenry, Daniel Helmick, Charles Bergh, and Larry Matthies. Algorithms and sensors for small robot path following. volume 4, pages 3850 – 3857 vol.4, 02 2002.

[16] Raúl Mur-Artal and Juan D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.