VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

DETEKCE OBJEKTŮ NA ROBOTICKÉM PRACOVIŠTI POMOCÍ DVOJICE KINECTŮ

OBJECT DETECTION ON ROBOTIC WORKPLACE USING TWO KINECTS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. PETER DRAGÚŇ**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JAROSLAV ROZMAN, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

142882

Ústav:	Ústav inteligentních systémů (UITS)
Student:	**Dragúň Peter, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Vývoj aplikací
Název:	**Detekce objektů na robotickém pracovišti pomocí dvojice Kinectů**
Kategorie:	Umělá inteligence
Akademický rok:	2022/23

Zadání:

1. Nastudujte práci s Kinectem v.2 v Robotickém operačním systému (ROS) a seznamte se s robotickým pracovištěm vybaveným ramenem Melfa.
2. Nastudujte metody detekce polohy a tvaru objektů pomocí kamery a Kinectu a metody pro určení polohy objektů v globálním souřadném systému. Počítejte s takovou přesností, která umožní detekovaný objekt uchopit robotickým ramenem.
3. Navrhněte program vytvořený pomocí ROS, který pomocí dvojice Kinectů detekuje objekty na robotickém pracovišti a určí jejich tvar a polohu v souřadném systému ramene.
4. Navržený program implementujte a vyzkoušejte dosahovanou přesnost v různých místech pracovní plochy včetně oblasti do cca 50cm nad plochou. Dosahovaná přesnost by měla být pod 0,5cm.

Literatura:

- Robotický operační systém, www.ros.org, [online], cit. 20.10.2021

Při obhajobě semestrální části projektu je požadováno:

- bez požadavků

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Rozman Jaroslav, Ing., Ph.D.**
Vedoucí ústavu:	Hanáček Petr, doc. Dr. Ing.
Datum zadání:	1.11.2022
Termín pro odevzdání:	17.5.2023
Datum schválení:	3.11.2022

Abstrakt

Práca sa zaoberá využitím dvojice hĺbkových senzorov Microsoft Kinect V2 na odhad polohy objektov. Hlavný dôraz kladie na kalibráciu, ako vzájomnú tak aj v rámci jedného zariadenia. Tie sú kritické pre správnu transformáciu. V teoretickej časti práca preskúmava metódy využívané na detekciu objektov a odhad ich polohy v priestore s využitím v robotickom operačnom systéme. V praktickej časti je riešený problém saturácie USB zbernice pomocou pridania jednodoskového počítača s architektúrou ARM pre druhý Kinect.

Abstract

This thesis deals with the use of a multi-depth camera Microsoft Kinect V2 for object pose estimation. The main emphasis is on calibration, including mutual and also calibration of a single camera. Which is critical for correct transformation. The theoretical part thesis describes methods for object detection and their pose estimation in space using the robotic operating system. The practical part thesis describes resolving issues with USB saturation by adding a single board computer based on ARM architecture for secondary Kinect.

Kľúčové slová

Mitsubishi, Mefla, ROS, Robotický operačný systém, Kinect, Kinect v2, Microsoft Kinect, robotické rameno, AruCo, stereo kalibrácia

Keywords

Mitsubishi, Mefla, ROS, Robotic operating system, Kinect, Kinect v2, Microsoft Kinect, robotic arm, AruCo, stereo calibration

Citácia

DRAGÚŇ, Peter. *Detekce objektů na robotickém pracovišti pomocí dvojice Kinectů*. Brno, 2023. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jaroslav Rozman, Ph.D.

Detekce objektů na robotickém pracovišti pomocí dvojice Kinectů

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením pána Ing. Jaroslava Rozmana, Ph.D.. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Peter Dragúň
23. mája 2023

Poďakovanie

Týmto by som chcel poďakovať hlavne vedúcemu práce, pánovi Ing. Jaroslavovi Rozmanovi, Ph.D. za jeho odborné vedenie práce, cenné rady a zapožičanie potrebného vybavenia. Zároveň by som chcel poďakovať mojej rodine a priateľom za podporu nielen počas písania práce, ale aj počas celého štúdia.

Obsah

Kapitola 1

Úvod

V súčastnosti sú čoraz viac a viac populárne roboty v rôznych oblastiach. Veľké robotické ramená sa využívajú vo všetkých priemysloch. Či už ide o lakovanie, prepravu alebo montovanie. Stále je potrebné presne definovať kam sa má rameno pohnúť. Jeho pohyby sú definované na základe rôznych senzorov a musia byť čo najpresnejšie na vykonanie požadovaného úkonu. Jedným z najdôležitejších aspektov je pri tom definovanie pozície objektu v priestore, aby bol robot schopný s ním interagovať.

Cieľom práce je určiť súradnice v priestore čo najpresnejšie. Výsledky z tejto práce by mali byť ďalej použiteľné pre navigáciu robotického ramena Melfa na fakulte. Na určenie polohy sú aktuálne používané kamery, ktoré nie sú dostatočne presné, preto bolo potrebné prísť s presnejšou metódou. Práca sa zaoberá návrhom a následnou realizáciou využitia dvojice RGB-D kamier pre určenie polohy objektov. RGB-D kamery oproti klasickým kamerám obsahujú ďalšie senzory, ako napríklad hĺbkový senzor, ktorý bude v tejto aplikácii veľmi nápomocný. Konkrétne sa práca bude zaoberať využitím komerčných senzorov Kinect od spoločnosti Microsoft. Tie už dávno nie sú len nástrojom na hranie hier, ale vďaka svojej rozšírenosti a nízkej cene aj senzorom pre jednoduchšie roboty.

Pre získanie skutočného 3D obrazu je potrebné objekty zachytiť z viacerých uhlov a následne tieto dáta spojiť do jedného obrazu pre rekonštrukciu zaznamenávaného objektu. Existuje viacero spôsobov ako dosiahnuť úplný 3D obraz. Jedným z nich je umiestnenie objektu na otočnú podložku a pomocou jedného senzoru vytvoriť niekoľko snímkov s posunutým objektom. Problémom tohto spôsobu je, že objekt musí byť nehybný voči podložke a zaznamenávanie trvá niekoľko sekúnd.

Využitie viacerých RGB-D kamier sa stáva čoraz populárnejším, prináša benefity v podobe lepšieho pokrytia priestoru, či zabránenie vytvárania prekryvov objektov na scéne. Zároveň dokáže pracovať rýchlejšie ako metódy využívajúce jednu kameru pre získanie 3D obrazu. Existuje viacero štúdií, kde takéto riešenia priniesli mnoho benefitov, či už v oblasti biológie pre detekciu ľudského tela, ale napríklad aj práve v robotike pre zlepšenie detekcie objektov. Tieto metódy však prinášajú svoje výhody za cenu problému kalibrácie a synchronizácie mračien bodov. Tá je jednou z najdôležitejších súčastí celého systému a predstavuje najzraniteľnejší bod riešenia.

Kinecty budú uchytené na stenách miestnosti s robotickým ramenom tak, aby zaistili čo najväčšie a najpresnejšie pokrytie miestnosti. Práca sa zaoberá problematikou skladania týchto dvoch obrazov a zisťovania následnej polohy hľadaného objektu. V prvých kapitolách sa práca zameriava na získanie teoretických znalostí z oblasti robotického operačného systému, opisu senzoru Kinect, práce s ním a jeho spôsobom merania vzdialenosti. Nasleduje naštudovanie spôsobov určovania polohy objektov v priestore. V praktickej časti sa venujem

ako samotnému návrhu riešenia, tak aj implementácii tohto návrhu. Poslednou časťou je overeniu jeho funkcionality, a teda zhodnotenie prípadného zlepšenia prekryvov objektov na scéne.

Kapitola 2

Detekcia objektov v priestore

Vo všeobecnosti môže detekcia objektov znamenať zistenie polohy objektu, jeho veľkosť, tvar alebo dokonca rozpoznanie objektu. Keďže cieľom tejto práce je zamerať objekt pre robotické rameno, ďalej sa v práci zameriam hlavne na pozíciu objektu v priestore, s tým súvisiaci jeho tvar, veľkosť a reprezentáciu týchto vlastností pomocou rôznych spôsobov. Avšak práca nebude ďalej rozoberať možnosti rozpoznania objektu a jeho zaradenia do nejakej kategórie.

Práca s 3D priestorom je zásadná v robotike, keďže poznanie prostredia je nevyhnutné pre komplexné robotické úlohy. Veľa zo spôsobov detekcie objektov využíva kameru alebo laser na detekciu tvarov objektov, tie sú úspešné hlavne pre väčšie a textúrované objekty. Pre detekciu generických objektov je lepšie využiť kombináciu tvaru a farby objektu.

2.1 Metódy využívané s RGB-D kamerami

Vďaka kompaktným RGB-D senzorom sa tento spôsob detekcie stále rozširuje. Presná detekcia a aproximácia polohy v 3D priestore založená na RGB-D dátach je aktuálne stále v aktívnom výskume. V posledných rokoch si získala veľa pozornosti v súťažiach Amazon Picking Challenge. [14]

Na popis objektov nasnímaných pomocou RGB-D kamier sa využíva mračno bodov. Z tohto mračna sa následne získavajú informácie o polohe, tvare objektu a prípadne ďalšie informácie.

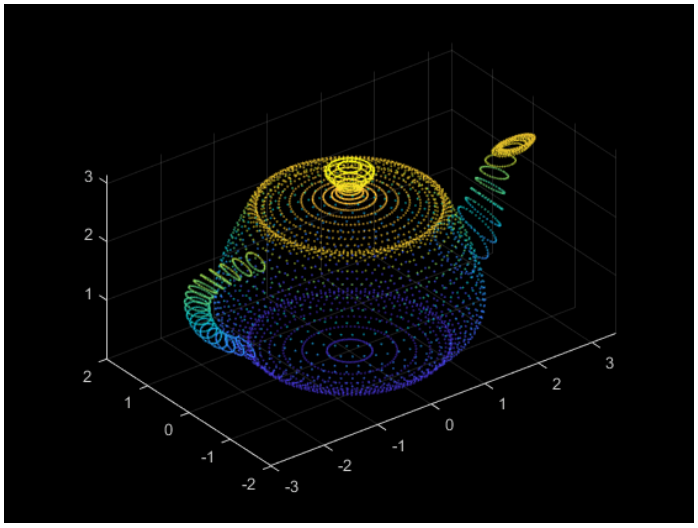
2.1.1 Mračno bodov

Mračno bodov je súbor bodov v priestore, kde každý bod má priradené svoje karteziánske súradnice (X, Y, Z). Typickým využitím je popis povrchu nejakého objektu. Vo všeobecnosti sú vytvorené 3D skenermi, tie môžu vytvoriť rozsiahle a pomerne presné mračná. Samotné body sa prevádzajú na trojuholníkovú polygonálnu sieť modelu, tá sa následne používa na ďalšie spracovanie.

Point Cloud Library (PCL)

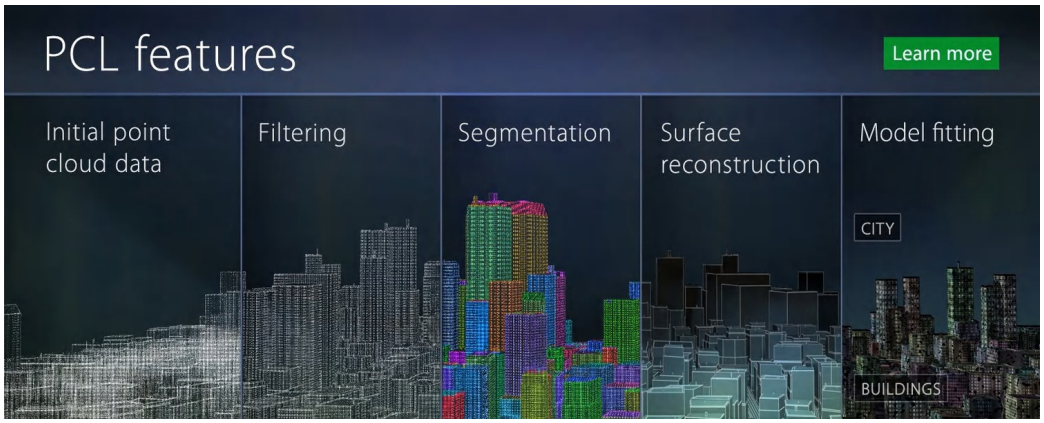
Ako už názov napovedá ide o open-source súbor knižníc na spracovanie n-D mračna bodov a 3D geometrie. Knižnica je plne integrovaná v ROS. Implementačným jazykom je C++, s podporou pre optimalizačné a paralelizačné nástroje ako OpenMP. Podporuje mnoho 3D

[1] Prevzaté z: `https://se.mathworks.com/help/vision/ref/pointcloud.html`



Obr. 2.1: Objekt zachytený pomocou mračna bodov[1]

algoritmov, ktoré pracujú s mračnami bodov, ako sú napríklad filtrovanie, odhad vlastností, rekonštrukcia povrchu, segmentácia a podobne. Každý algoritmus je stavebným blokom, ten môže byť ďalej spojený s ďalšími blokmi, podobne ako uzly v ROS. Využíva sa hlavne vlastný formát PCD, ale knižnica podporuje aj ďalšie formáty. PCL obsahuje vlastnú vizualizačnú knižnicu založenú na VTK.[2] Ide o rozšírenú knižnicu, vďaka tomu existuje množstvo návodov ako ju používať, čo ďalej uľahčuje jej použitie. [13]



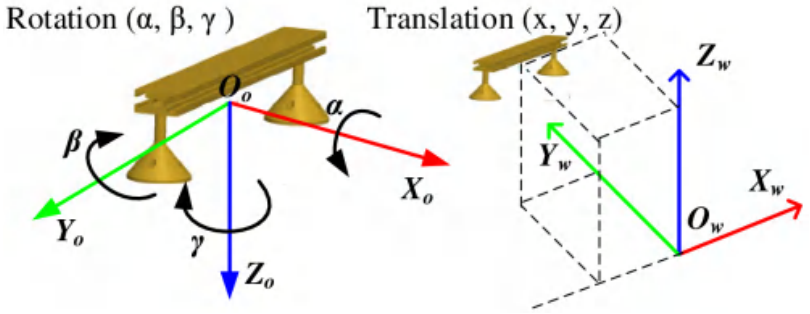
Obr. 2.2: Vizualizácia spracovania mračna bodov pomocou PCL - počiatočné mračno bodov, filtrovanie, segmentácia, rekonštrukcia plôch, detekcia[10]

Knižnicu je možné využiť aj v jazyku python pomocou mapovacích nástrojov akým je napríklad python-pcl[3]. Tento balíček následne priamo spúšťa knižnicu PCL. Napriek tomu, že tento nástroj nie je aktuálne udržiavaný, tak poskytuje rozsiahle nástroje pre spracovanie mračna bodov.

[2]The Visualization Toolkit - `https://vtk.org/`
[3]Dostupné z: `https://github.com/strawlab/python-pcl`

2.1.2 6D odhad polohy objektu

6D poloha je vlastne poloha v 3D priestore a následne aj natočenie objektu vo všetkých troch jeho smeroch. Pre lepšiu vizualizáciu viď obrázok 2.3. Tento spôsob môžeme využiť v prípadoch kedy máme k dispozícii model hľadaného telesa.



Obr. 2.3: 6D poloha objektu [9]

Ide o kombinovaný spôsob, využívajú sa kombinácie klasifikácie objektov a získavania mračna bodov. Pri pokročilejších metódach do neurónovej siete vstupujú nielen informácie z kamery ale aj z hĺbkového senzoru. Následne sa vytvára vzor, oproti ktorému sa porovnávajú nasnímané objekty. Klasifikácia objektov v kombinácii s klasickým procesom získavania mračna bodov prináša presnejšie výsledky, keďže vďaka klasifikácii objektov máme viac informácii ako sme schopní nasnímať v reálnom čase pri detekcii polohy. [14]

Veľa z dostupných algoritmov na určenie polohy spolieha na detekciu objektu, prípadne aspoň na jeho klasifikáciu. Ak poznáme aký objekt je na scéne je jednoduchšie určiť jeho polohu a natočenie keďže poznáme jeho rozmery. Následne je potrebné porovnať nasnímaný objekt so vzorom a zistiť jeho natočenie. Prináša to však nevýhodu, že každý objekt musí byť pred detekciou odfotografovaný a uložený jeho 3D model do databázy. Menším uľahčením môže byť využitie už existujúcich databáz, ktoré ponúkajú už naskenované modely pomocou RGB-D senzorov.

Existuje niekoľko spôsobov implementácie samotnej neuronovej siete v závislosti na rýchlosti, ktorou je potrebné objekt spracovať, prípadne na presnosti spracovania a iných požadovaných parametroch. Skúmané riešenie bolo testované primárne na statickej scéne avšak cielené je aj na pohyblivé objekty. Presnosť tohto konkrétneho riešenia bola overená experimentálne na jednoduchom odhade polohy objektu a následne jeho zdvihnutie pomocou robotického ramena. Z 50 predmetov rôznych tvarov bola úspešnosť zamerania polohy a nastavenia ramena nad predmet približne na úrovni 95 percent. Avšak vďaka testovaciemu podomácky vyrobenému chytaču bola úspešnosť samotného uchopenia len na úrovni 80 percent. Úspešnosť samotného zamerania je pritom takmer rovnaká ako pri podobných algoritmoch. [14]

2.1.3 Dynamický objektovo orientovaný SLAM

Slúži na zaznamenávanie dynamickej polohy objektu oproti statickému pozadiu. Podobne ako v predchádzajúcej metóde sa využívajú neuronové siete na rozpoznanie objektu, avšak modely objektov nie je potrebné poznať už pred samotným spustením aplikácie. Môže využívať niekoľko spôsobov: pravdepodobnostná asociácia dát, rámový model Iterative Closest Point (ICP) kombinovaný s farbou, maximalizácia pravdepodobnosti 3D úrovňovej mno-

žiny. Modely objektov sú rekonštruované priebežne agregáciou dát z RGB-D senzoru s novou zaznamenanou pozíciou.

Spomínané prístupy založené na SLAM môžu čeliť výzvam v robotických aplikáciach, kde sa vyskytujú malé, rovné alebo lesklé objekty bez textúr. To je spôsobené nedostatkom súhlasných bodov medzi snímkami kamery. Na čiastočné zlepšenie tohto problému sa využíva porovnávanie aktuálneho snímku s viacerými historickými snímkami.

Metóda je vhodná na využitie s pohybujúcimi sa objektami, napríklad na posuvnom páse. Presnosť samotného rozpoznania a určenia polohy je na úrovni približne 92 percent, čo je porovnateľné s predchádzajúcou skupinou metód, avšak nie je potrebné vopred získavať modely detekovaných objektov. [18]

2.1.4 3D skenovanie rukou držaných objektov

Objekt je držaný napríklad robotickou rukou, tá ho postupne presúva. Pri tom sa vykonáva rekonštrukcia objektu spolu s jeho sledovaním. Pri tejto metóde nastávajú podobné problémy ako pri dynamickom SLAM, keďže sa súčasne vykonáva rekonštrukcia objektu aj jeho sledovanie. Samotný odhad polohy je súčasťou skenovania, avšak tento spôsob je rozdielny od online 6D sledovania pozície v kľúčových detailoch. [18]

2.1.5 Statické metódy bez využitia modelov objektov

Pred samotným odhadom vzdialenosti je potrebné upraviť zaznamenané dáta. Nasnímané mračná bodov sú často pre aplikácie bežiace v reálnom čase príliš veľké. Preto je nutné najprv znížiť ich rozlíšenie. Následne môžeme ďalej obmedziť veľkosť mračien pomocou definovania oblasti záujmu, teda pomocou filtrovania.

Keďže ide o statický systém je pomerne jednoduché určiť oblasť, v ktorej očakávame objekty. Na ďalšie zmenšenie počtu bodov je možné použiť filtrovanie pomocou *VoxelGrid* algoritmu. Ten redukuje počet bodov konštrukciou 3D poľa voxelov nad vstupným mračnom. Pre každý voxel je vypočítaný stred bodov, ktoré k nemu patria. Následne môžeme predpokladať, že predmety sú položené na nejakej rovine. Tú môžeme odfiltrovať pomocou knižnice PCL, kde pomocou rovinnej segmentácie vieme detekovať najväčšiu plochu.

Následne sa využíva algoritmus *Random Sample Consensus* (RANSAC), pomocou ktorého sa odhadujú body, ktoré patria k ploche. Zvyšné body sú združené a vytvárajú samotné objekty. Využíva sa Euklidovská združovacia extrakcia (*Euclidean Clustering Extraction*) na rozdelenie bodov do zhlukov na základe ich vzájomnej vzdialenosti. Body sú zaradené do zhluku v prípade, že ich polomer vzdialenosti je menší ako r. Polomer pri tom musí byť väčší ako veľkosť voxelu použitá pri *VoxelGrid* algoritme.

Na samotný odhad vzdialenosti je potrebné vypočítať jeho ťažisko. Zo všetkých bodov objektu vieme vypočítať strednú hodnotu vzhľadom na tri koordináty. [14]

Úspešnosť metódy samotného zamerania predmetu by mala byť podobná s metódou spomínanou v podkapitole 2.1.2, keďže ide o podúlohu, ktorú je potrebné riešiť aj v rámci rozpoznávania a odhadu polohy pomocou neuronových sietí.

Kapitola 3

Kinect

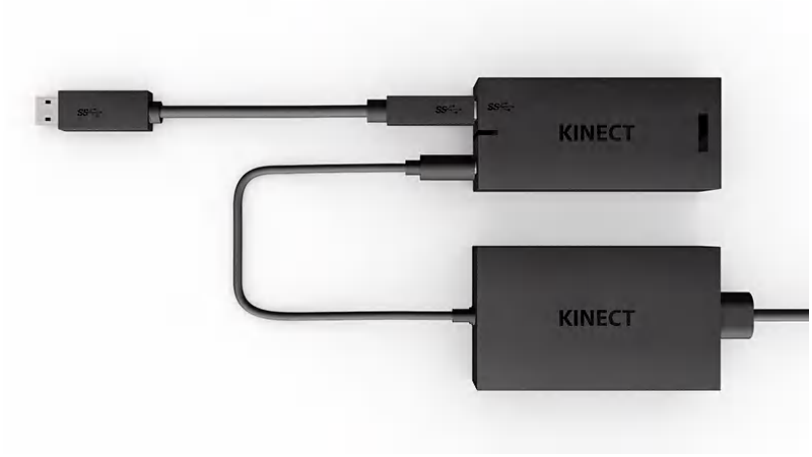
Kinect je zariadenie primárne slúžiace na snímanie pohybu. Bol vyvinutý spoločnosťou Microsoft pôvodne ako doplnok k hracej konzole. Neskôr sa stal populárny aj v iných odvetviach ako je robotika a podobne. Existuje niekoľko verzií tohto zariadenia. K tejto práci som dostal k dispozícii konkrétne druhú verziu, ktorá bola predstavená spolu s konzolou Xbox One v roku 2013. Aktuálne spoločnosť Microsoft už ďalej nevyvíja Kinect ako doplnok k hracím konzolám, ale ako osobitný vývojový modul s využitím v počítačovom videní a spracovaní reči. Tento model je ďalej vyvíjaný pod názvom Azure Kinect.[1] Konkrétne rozdiely medzi jednotlivými verziami sú vysvetlené v podkapitole 3.1.

Obr. 3.1: Kinect v2

Na prednej strane zariadenia sa nachádzajú samotné senzory, vo vrchnej a zadnej časti môžeme nájsť ventilačné otvory pre aktívne chladenie, na spodnej strane snímača sa nachádza ¼ závitu vďaka ktorému je jeho uchytenie jednoduché pomocou bežných stojanov na kamery a podobne. Keďže senzor využíva vlastný konektor, ktorý prenáša potrebné napájanie aj dáta, je potrebné pripojiť redukciu na pripojenie k počítaču, viď obrázok 3.2. Tá okrem iného dodáva napájanie. Pre komunikáciu sa využíva rozhranie USB 3.0. Konkrétnym senzorom a ich využitiu sa budem venovať v nasledujúcich kapitolách.

[1] `https://azure.microsoft.com/services/kinect-dk/`

[2] Prevzaté z: `https://support.xbox.com/help/hardware-network/kinect/kinect-for-windows-v2-setup-with-adapter`



Obr. 3.2: Adaptér senzora Kinect pre počítač[2]

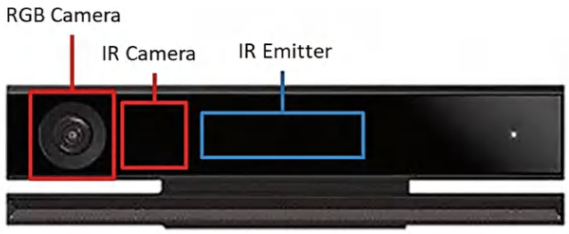
3.1 Verzie

Na obrázku 3.3 môžeme vidieť všetky verzie Kinectov. Pôvodný Kinect 360 (dnes označovaný aj ako Kinect v1) bol revolučným zariadením medzi dostupnými 3D kamerami. Zariadenie si tak ľahko našlo využitie aj v robotike, virtuálnej a zmiešanej realite, detekcii objektov a podobne. Toto zariadenie využívalo princíp projekcie vzoru, kde sa premieta známy vzor pomocou infračerveného žiarenia na scénu a následne sa vypočíta hĺbka pomocou skreslenia tohto vzoru. Kinect v2 ponúkol vyššie rozlíšenie kamery a zároveň sa zmenil spôsob snímania hĺbky obrazu na Time of Flight (ToF), viď podkapitola 3.2.1. Využitie tejto metódy dovolilo zariadeniu dosiahnuť oveľa presnejšie výsledky ako jeho predchodca. Azure Kinect využíva taktiež ToF princíp, avšak naviac podporuje zmenu režimu snímania. Okrem toho je zariadenie menšie, poskytuje vyššie rozlíšenie obrazu a drobné zlepšenie v presnosti a spoľahlivosti. Vďaka úspechu predchádzajúcich verzií aj mimo herného priemyslu je Azure Kinect vyvíjaný primárne na použitie vo výskume a robotike. [17]

Obr. 3.3: Zľava doprava - Kinect v1, Kinect v2, Azure Kinect [17]

3.2 Senzory

Kinect v2 obsahuje niekoľko senzorov: RGB kameru s rozlíšením 1920 x 1080 pixelov, IR kameru s rozlíšením 512 x 424 pixelov, IR vysielač a sadu mikrofónov. Vďaka kombinácii týchto senzorov sme schopní získať ako fotku prostredia, tak aj hĺbkovú mapu. Tú senzor získava na základe princípu dĺžky letu laserového pulzu (Time of Flight - ToF). Vďaka kombinácii týchto senzorov môžeme povedať, že ide o RGB-D senzor, teda kombináciu kamery a hĺbkového senzoru (D - *depth*). Výsledkom merania je teda takzvaný 2,5D obraz priestoru, ktorý oproti 2D obrazu ponúka naviac informáciu o hĺbke avšak nejde o skutočný 3D obraz. Ako môžeme vidieť na obrázku 3.4 senzory sú umiestnené vedľa seba. Zároveň tieto vzdialenosti môžu mať malé odchýlky od zariadenia k zariadeniu. Vďaka tomuto posunu každý senzor vidí scénu z iného pohľadu, čo generuje drobné nepresnosti v meraniach. Je dôležité každý Kinect kalibrovať, aby nedochádzalo k posunom v obraze medzi RGB a hĺbkovým obrazom.



Obr. 3.4: Pozícia senzorov [2]

3.2.1 Time of Flight (ToF)

Metóda dĺžky letu slúži na vypočítanie vzdialenosti. Vo všeobecnosti môže byť metóda založená nielen na svetle ale aj na zvuku či rádiových vlnách a podobne. Na spôsobe vo všeobecnosti nezáleží, ale potrebujeme poznať rýchlosť akou sa signál pohybuje prostredím. Vzdialenosť sa následne počíta ako rýchlosť signálu (c) vynásobená časom (t). Takto dostaneme vzdialenosť, ktorú prekonal signál na ceste k objektu a späť, teda dvojnásobnú vzdialenosť. Pre výpočet vzdialenosti môžeme teda použiť vzorec:

$$h = \frac{ct}{2}$$

V skutočnosti by senzor postavený na tomto princípe potreboval pikosekundové rozlíšenie aby dokázal určiť vzdialenosť objektu na milimetre. To by bolo náročné aj pre zariadenia pracujúce s malým rozlíšením. Preto väčšina kamier využíva fázový posun (*phase shift*). Je postavený na rovnakom princípe, ale využíva sa periodický signál s frekvenciou f a fázový posun θ. Hĺbka môže byť následne spočítaná ako:

$$h = \frac{c}{4\pi f}\theta$$

V prípade Kinectu V2 sa používa infračervené žiarenie a IR kamera. Senzor pomocou infračerveného svetla, ktoré je emitované diódami, opakovane osvetľuje scénu. Následne sa svetlo odráža od objektov a je zaznamenané pomocou IR kamery po jednotlivých pixeloch. Využíva sa zabudovaný generátor modulovaného obdĺžnikového signálu. Ten je použitý ako pri vysielaní, tak pri prijímaní žiarenia. Na základe znalosti modulačnej frekvencie je systém

schopný vypočítať čas letu k objektu a späť. Zaznamenávané sú hodnoty pri rôznych modulačných frekvenciách, čo dovoľuje obmedziť nepresnosti. Vďaka tomuto spôsobu snímania výsledky nie sú ovplyvnené vonkajším osvetlením scény.

Pri snímaní je potrebné myslieť aj na to, že rôzne objekty odrážajú rôzne množstvo svetla. Intenzita sa môže meniť v závislosti na vzdialenosti, ale aj na povrchu telesa a podobne. Objekt môže odraziť 95 % signálu, ale aj 10 %. Senzor preto vytvorí dva obrázky s rozdielnou dĺžkou uzávierky a následne pre každý pixel vyberie vhodnejší. [5]

3.3 Presnosť merania

Podľa špecifikácie senzoru je jeho operačný rozsah 0,5 - 4,5 metra. Experimentálne bola overená presnosť senzoru, kde dosahuje priemernú odchýlku približne 4,8 milimetra a smerodajnou odchýlkou 2.2 milimetra. Pri tom tieto výsledky bol senzor schopný poskytovať v rozsahu 0,56 až 4,4 metra, teda približne v celom operačnom rozsahu. Snímanie hĺbky scény je teda pomerne presné. V porovnaní s podobnými senzormi sa jedná o jeden z najpresnejších. [5] Pri spomínaných odchýlkach uvažujeme použitie Kinectu, ktorý prešiel kalibráciou.

3.3.1 Využitie viacerých zariadení

Pre zlepšenie presnosti merania je taktiež možné využiť viacero zariadení. Existuje viacero spôsobov ako dosiahnuť lepšie výsledky pomocou ďalších zariadení:

- **redukcia oklúzie** - objekty, ktoré sa nachádzajú na scéne priamo za inými objektami kamera nie je schopná zachytiť, aj preto hovoríme, že výsledné mračno bodov je takzvané 2.5D, teda neobsahuje úplnú 3D mapu scény. Pridaním druhého zariadenia vieme tento prekryv redukovať, a tak získať informácie aj o objektoch, ktoré na prvom zariadení nie sú viditeľné.

- **spriemerovanie výsledkov** - algoritmy na odhad polohy sú často chvejivé vďaka istej neistote, preto dáta musia byť často filtrované. Toto sa dá redukovať využitím viacerých kamier s rôznym zorným poľom. Odhady polohy zo všetkých kamier sú spriemerované a na základe toho je vytvorený výsledný odhad polohy.

- **rozšírením zorného poľa** - hlavným limitujúcim faktorom viacerých hĺbkových kamier je vzdialenosť od kamery, na ktorú je senzor schopný rozpoznať objekty. Využím viacerých kamier je možné zorné pole detekcie rozširovať.

Využitie viacerých kamier však má svoje obmedzenia. Nastavenie takéhoto systému vie byť pomerne náročné, kalibrácia je netriviálna a nie je jednoduchá adaptácia na rôzne prostredia. Vďaka limitovanému zornému poľu je potrebné kamery namontovať vo vzdialenosti do pár metrov od detekovaných objektov. Ďalšou limitáciou sú hardvérové požiadavky na šírku pásma, keďže každá kamera vyžaduje vlastný USB3.0 ovládač. Pri niektorých druhoch senzorov môže nastať ovplyvnenie merania druhou kamerou, tento problém je hlavne pozorovateľný pri kamerách využívajúcich štruktúrované svetlo, ako Kinect v1. Keďže sa spolieha na identifikovanie skreslenia vzorov, ich prekrývanie môže spôsobiť rušenie. Je dôležité poznamenať, že podľa niektorých štúdií sa celková presnosť riešenia zlepšila len minimálne (prípadne vôbec nezlepšila) pri použití viacerých Kinectov. Jediné preukázateľné zlepšenie bolo zaznamenané v prípade redukcie oklúzie, kde boli výsledky presnejšie. [3]

3.4 Softvér

Microsoft poskytuje na svojich stránkach zdarma na stiahnutie potrebné SDK pre Kinect 2.0.[3] Ide o robustný nástroj, ktorý poskytuje vývojárom možnosti vytvoriť aplikácie na rozpoznávanie reči, gest a podobne. Okrem iného poskytuje aj príklady použitia v rozličných jazykoch, ktoré uľahčia začiatky pri práci s Kinectom. Vyžaduje sa však použitie na operačnom systéme Windows. Ďalším spôsobom ako pracovať s Kinectom je využiť ho napríklad v rámci robotického operačného systému pomocou knižníc *libfreenect2*[4] a *iai_kinect*[5], kde má pomerne veľkú podporu a taktiež množstvo príkladov. Prvá spomínaná knižnica slúži ako ovládač samotného senzoru, v prípade druhej ide o súbor nástrojov, ktoré uľahčujú využívanie Kinectu v2 v prostredí robotického operačného systému. Jedným z nástrojov je práve aj kalibračný program, ktorý je pre správne fungovanie zariadenia veľmi dôležitý. Obe spomínané knižnice sú podporované ako pre zariadenia bežiace na architektúre x86, tak aj pre architektúru ARM (s malými úpravami).

3.5 Alternatívy

Kinect patrí medzi najznámejšie RGB-D kamery, vďaka jeho rozšíreniu primárne ako nový spôsob ovládania hier, avšak nie je to jediná možnosť, pomocou ktorej vieme získať hĺbkové dáta o scéne. Vďaka masovej produkcii však bol schopný dosiahnuť nižšie ceny, a teda dostať sa najviac do povedomia širokej verejnosti. [3] Existujú ale aj ďalšie alternatívy, ktoré ponúkajú podobné vlastnosti avšak často za vyššiu cenu, prípadne s horšou podporou od komunity a vývojárov. Asi najväčším rivalom Kinectu je rodina zariadení Intel Realsense, ktoré poskytuje podobné parametre a pomerne dobrú podporu, v závislosti na konkrétnom zariadení.

[3] Dostupné z: `https://developer.microsoft.com/windows/kinect/`
[4] Dostupné z: `https://github.com/OpenKinect/libfreenect2`
[5] Dostupné z: `https://github.com/code-iai/iai_kinect2`

Kapitola 4

Robotický operačný systém (ROS)

Robotický operačný systém (anglicky: Robotic operating system) je flexibilný aplikačný rámec (framework) na písanie softvéru pre roboty. Ide o kolekciu nástrojov, knižníc a konvencií. Hlavnou snahou je zjednodušiť vytváranie komplexných a robustných aplikácii pre roboty na rôznych platformách. Cieľom je dekomponovať problémy na jednoduchšie, tým vznikajú znovu použiteľné riešenia.

Ide o open-source projekt, ktorý sa v súčastnosti teší veľkej obľube hlavne kvôli jeho všestrannosti. Využíva sa v rôznych odvetviach robotiky ako je poľnohospodárstvo, logistika, autonómne autá, príprava jedla, ťažký priemysel, drony a ďalšie. Podpora rôznych programovacích jazykov medzi balíčkami prináša výhody práce rôznych jazykov, jednotlivé úlohy môžu byť v jednom jazyku jednoduchšie alebo dokonca v iných jazykoch nemožné.

Existuje niekoľko distribúcii tohto frameworku v závislosti na verzii, podpore a stabilite. Na rozdiel od verzie 1 je ROS vo verzii 2 plne podporovaný na všetkých populárnych platformách - okrem Ubuntu Linux aj na Windows 10 a macOS. Ďalším rozdielom medzi verziami sa venujem v podkapitole 4.2.

Aktuálne v čase písania práce sú podporované verzie pre ROS 1 *Melodic Morenia* a *Noetic Ninjemys* (s dlhodobou podporou) a pre ROS 2 distribúcie *Galactic Gechelone*, ktorý prináša najnovšie funkcie, *Foxy Fitzroy* s dlhodobou podporou a aktuálne vo vývoji *Rolling Ridley*. [15]

Navrhovaný systém má spolupracovať s ramenom Melfa a knižnice pre neho sú napísané pre ROS 1, konkrétne pre distribúciu *Kinetic Kame*. Ďalej sa v práci zameriam na prácu s ROS 1 a jeho vlastnosťami, keďže novšia verzia ROS nie je spätne kompatibilná. V nasledujúcich kapitolách sa budem venovať základnej stavbe ROS, jeho aplikáciám a nástrojom, ktoré uľahčujú vývoj robotických aplikácií, ako sú nástroje na vizualizáciu RViz a simulačný nástroj Gazebo.

4.1 Základné koncepty

Pred prácou s ROS je dôležité ujasniť si terminológiu a lepšie pochopiť štruktúry systému cez základné zložky ROS. Definované sú tri úrovne konceptov: úroveň súborového systému, úroveň výpočtového grafu a komunitná úroveň. Okrem toho ROS definuje dva typy mien:

- *Package Resource Names* - využívané na úrovni súborového systému na zjednodušenie odkazov na súbory.

- *Graph Resource Names* - využívané na úrovni výpočtového grafu, na pomenovanie uzlov, parametrov, tém a služieb. Poznáme 4 typy: základné, relatívne, globálne a privátne. Skladajú sa z veľkých a malých písmen abecedy a znakov . (znak bodka) a \.

4.1.1 Úroveň súborového systému

Úroveň súborového systému pokrýva hlavne zdroje ROS, ktoré sa nachádzajú na disku. Patria sem:

- **Balíčky** (Packages) - sú hlavnou zložkou pre organizáciu softvéru v ROS. Balíček môže obsahovať uzly, knižnice, datasety, konfiguračné súbory a podobne. Balíčky sú najmenšou stavebnou jednotkou a vydávanou zložkou v ROS.
- **Metabalíčky** (Metapackages) - špecializované balíčky, ktoré reprezentujú skupinu súvisiacich balíčkov. Využívajú sa najmä na zabezpečenie spätnej kompatibility s *rosbuild*.[1]
- **Manifest k balíčkom** (Package Manifests) - poskytuje metadáta o balíčku, jeho meno, verziu, popis, licenciu, závislosti a ďalšie metadáta. Je to XML súbor s názvom *package.xml*.
- **Repozitáre** (Repositories) - kolekcia balíčkov, ktoré zdieľajú rovnaký systém na správu verzií. Repozitáre môžu obsahovať aj jeden balíček.
- **Typy správ** (Message types) - definujú štruktúru dát, ktoré sú zasielané správami.
- **Typy služieb** (Service types) - definujú štruktúru dát pre požiadavky a odpovede v službách. [6, 12]

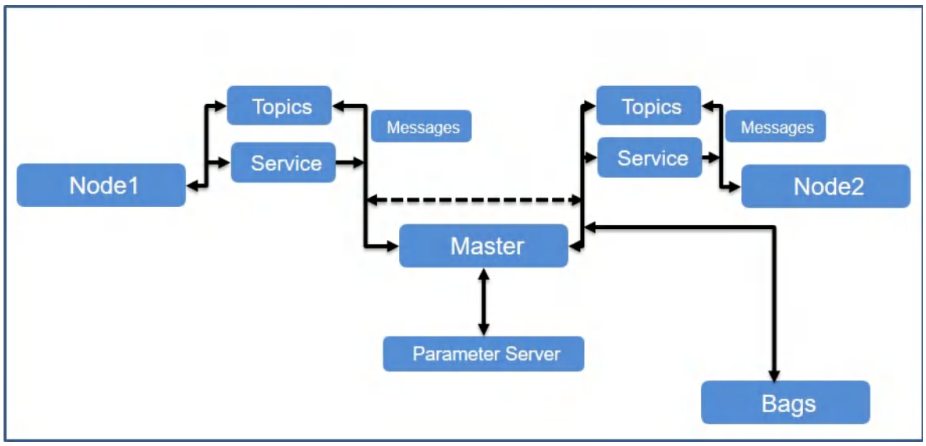
4.1.2 Úroveň výpočtového grafu

Výpočtový graf je *peer-to-peer* sieť ROS procesov, ktoré spracovávajú dáta spoločne. Každý z konceptov poskytuje dáta rozdielnym spôsobom. Medzi jej časti patria:

- **Uzly** (Nodes) - sú procesy, ktoré vykonávajú samotné výpočty. Ovládací systém robota väčšinou pozostáva z mnoho uzlov. Napríklad jeden uzol kontroluje odhad vzdialenosti pomocou laseru, ďalší motory, plánovanie cesty a tak ďalej. Na implementovanie sa využíva klientská knižnica ako napríklad *roscpp* alebo *rospy*.
- **Master** - hlavný uzol, poskytuje registráciu mien a vyhľadávanie pre zvyšok výpočtového grafu. Bez neho by uzly neboli schopné zasielať správy alebo volať služby.
- **Parametrový server** (Parameter server) - je súčasťou hlavného uzla. Dovoľuje uložiť dáta pomocou kľúča na centrálne umiestnenie.
- **Správy** (Messages) - slúžia na komunikáciu medzi uzlami. Správa je jednoducho dátová štruktura, ktorá obsahuje typované hodnoty. Podporované sú primitívne typy (integer, boolean, float atď.) a polia primitívnych typov.

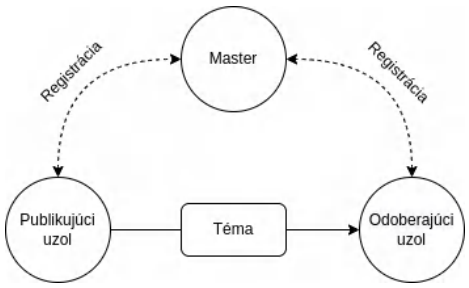
[1] rosbuild: starší spôsob prekladania a inštalácie balíčkov, nahradený nástrojom *catkin*

- **Témy** (Topics) - správy sú zasielané cez systém publikovania a odoberania. Uzol odosiela správu publikovaním na určitú tému. Téma je meno, ktoré slúži na identifikovanie správy. Uzol, ktorý sa zaujíma o správy určitého typu sa prihlási na odpovedajúcu tému. Jednu tému môže odosielať aj prijímať viacero uzlov súčasne. Uzol môže publikovať aj odoberať viacero tém. Vo všeobecnosti publikovatelia a odoberatelia si nie sú vedomí vzájomnej existencie. Cieľom je oddeliť vytvorenie správy od jej konzumácie.
- **Služby** (Services) - systém publikovania a odoberania (*publish/subscribe*) je veľmi flexibilný spôsob komunikácie, avšak nie je vhodný na dotaz a odpoveď. Tie sú často požadované v distribuovaných systémoch. Tento spôsob komunikácie je teda vedený pomocou služieb, tie sú definované ako pár štruktúr - jedna pre dotaz, druhá pre odpoveď. Uzol poskytuje službu pod nejakým menom a klientský uzol využíva službu poslaním požiadavky a čakaním na odpoveď.
- **Bag** - slúži na uchovávanie dát zo senzorov a podobne. [6, 12]



Obr. 4.1: Výpočetný graf [6]

Master slúži ako menná služba vo výpočtovom grafe. Uchováva témy a registrácie služieb pre uzly. Jednotlivé uzly sú navzájom poprepájané, pričom hlavný uzol slúži len na vyhľadanie, podobne ako DNS server. Uzly, ktoré sa chcú prihlásiť na odoberanie témy zažiadajú o spojenie priamo publikujúce uzly. Spojenie sa nadviaže pomocou dohodnutého protokolu, najčastejším je TCPROS, ktorý používa štandardné TCP/IP sokety. Na obrázku 4.2 môžeme vidieť tento koncept graficky znázornený.



Obr. 4.2: Koncept publikácie a odoberania témy

Koncept mien je v ROS dôležitý, mená sa používajú na identifikáciu uzlov, tém, služieb a parametrov. Vďaka tomu dokážeme vytvárať väčšie a komplexnejšie systémy. Každá klientská knižnica poskytuje podporu pre premapovanie mien. To znamená, že v prípade ak sa v systéme nachádzajú napríklad dve senzorické jednotky, ktoré by chceli využívať rovnaké uzly, ale komunikovať so systémom osobitne, stačí jednu z nich premapovať na nové meno. To sa následne bude využívať v ďalšej komunikácii. [12]

4.1.3 Komunitná úroveň

- **Distribúcie** (Distributions) - verzovaná kolekcia ROS balíčkov. Podobne ako pri distribúcii Linuxu ide o snahu uľahčiť inštaláciu kolekcie softvéru.
- **Repozitáre** (Repositories) - ROS sa spolieha na sieť repozitárov, kde rôzne inštitúcie vyvíjajú svoje vlastné časti robotického softvéru.
- **Ros Wiki** - hlavné fórum, ktoré slúži pre všetkých ľudí, môžu sem pridávať a opravovať dokumentáciu, vytvárať tutoriály a podobne.
- **Bug Ticket System** - nahlasovanie chýb v ROS
- **Mailing lists** - slúžia na zasielanie aktualizácií o ROS balíčkoch a pokladanie otázok
- **ROS Answers** - služba na pokladanie otázok o ROS.
- **Blog** - aktualizácie o komunite [12]

4.2 ROS 2

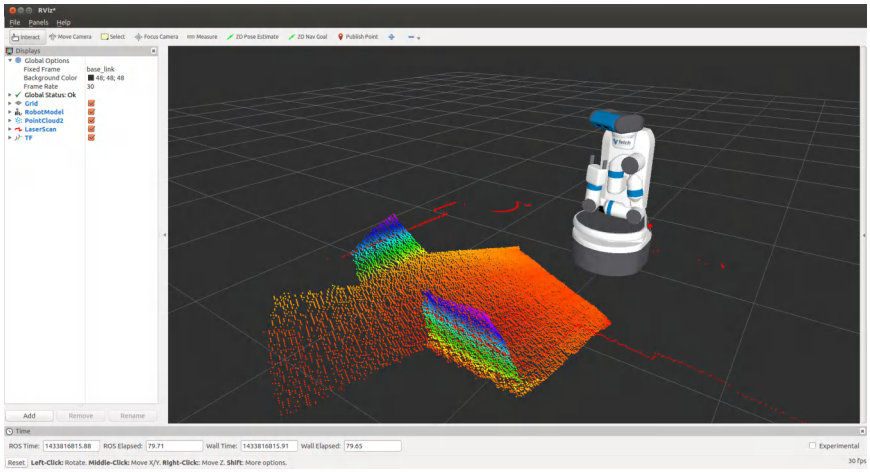
Ide o kompletné prepísanie robotického operačného systému. Táto verzia stavia na pomerne úspešnej prvej verzii systému, avšak prináša niekoľko vylepšení a rozdielov, vďaka ktorým nie je spätne kompatibilná s balíčkami vytvorenými pre ROS 1. Keďže ciele aplikácie sú podobné aj základné princípy, ďalej porovnám hlavné rozdiely medzi verziami.

Okrem už spomínanej natívnej podpory na všetkých platformách ďalšie rozdiely sú v novších verziách pre jazyky, väčšej abstrakcii a podobne. ROS 1 je primárne mierený na využívanie C++03 zatiaľčo ROS 2 prináša podporu pre C++11 a využíva aj niektoré časti z novších verzií. Podobne je podporovaná novšia verzia jazyku Python vo verzii aspoň 3.5 oproti Pythonu 2, ktorý bol využívaný v ROS 1. Zároveň ROS 2 prináša plnú podporu pre súbory ‘setup.py‘, tie boli predtým limitované spracovávaním pomocou CMake.

Medzi ďalšie rozdiely patrí napríklad väčšia miera abstrakcie pre serializáciu, prenos a zisťovanie. ROS 2 prináša mnoho výhod oproti ROS 1, avšak je potrebné všetky balíčky upraviť pre podporu v novej verzii. [16]

4.3 RViz

RViz (skratka z *ROS Virtualization*) je 3D nástroj na vizualizáciu v rámci ROS. Poskytuje grafické rozhranie pre jednoduchšie hľadaní chýb. Vizualizuje dáta zo senzorov, teda dokážeme sledovať čo robot „vidí“ a robí. RViz podporuje pridávanie zásuvných modulov na pridávanie ďalších funkcionalít. Funguje ako jeden z uzlov ROS, dokáže sa teda prihlasovať na odoberanie rôznych tém a takto získavať informácie o systéme. Napríklad ako môžeme

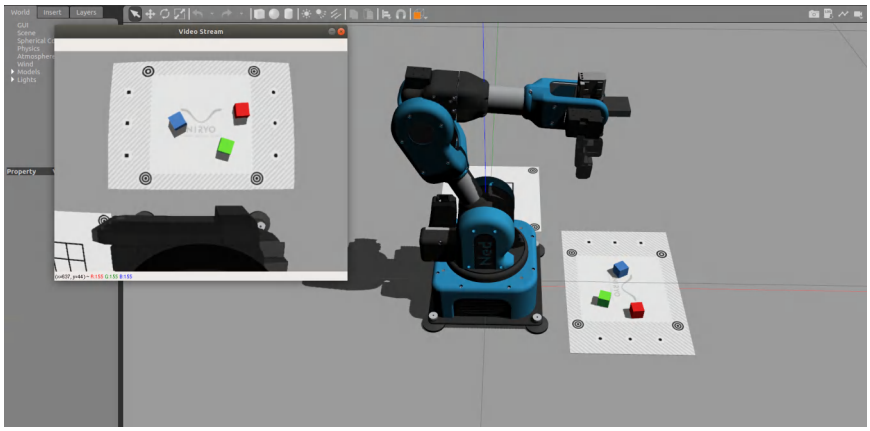


Obr. 4.3: RViz[2]

vidieť na obrázku 4.3 nástroj dokáže vizualizovať namerané hodnoty pred robotom pomocou jeho senzorov a zobraziť ich v priestore.

4.4 Gazebo

Gazebo je nástroj na simuláciu, dokáže presne a efektívne modelovať roboty v exteriéri aj interiéri. Je podobný hracím enginom avšak pracuje s vyššou presnosťou, súborom senzorov a rozhraním pre užívateľa aj programy. Typickým použitím je testovanie robotických algoritmov, návrh robotov a regresné testovanie s realistickými scenármi. Poskytuje viacero fyzikálnych enginov, bohatú knižnicu modelov robotov, prostredí a podobne. Pôvodne išlo o nezávislý open-source projekt Univerzity v Severnej Karolíne, neskôr integrovali ROS. V súčasnosti ide o najpoužívanejší nástroj na simuláciu v rámci Robotického operačného systému. Gazebo je podporovaný pre operačný systém Linux, konkrétne distribúciu Ubuntu. [8]



Obr. 4.4: Simulácia v Gazebo[3]

[2]Prevzaté z: `https://docs.fetchrobotics.com/visualization.html`
[3]Prevzaté z: `https://docs.niryo.com/dev/ros/v3.2.0/en/source/simulation.html`

Kapitola 5

Návrh systému

Pre správny návrh systému je najprv potrebné stanoviť problém, ktorý sa v práci snažíme vyriešiť. Následne rozdeliť problém na menšie časti, ktorých riešenie bude jednoduchšie.

5.1 Stanovenie problému

Cieľom práce je určiť polohu a tvar neznámeho objektu so statickým pozadím. Detekovaný objekt si pre potreby práce môžeme definovať vo všeobecnosti ako akýkoľvek objekt, ktorý sa zobrazí na scéne. Pre zjednodušenie môžeme uvažovať, že tento objekt je kváder bližšie nešpecifikovaných rozmerov (dostatočne veľký pre detekciu). Objekt je na scéne nehybný. Výsledná pozícia by mala byť dostatočne presná pre potreby zdvihnutia pomocou robotického ramena. Rameno je taktiež statickým prvkom na scéne, teda bez pohyblivej základne. Výsledná poloha objektu bude určená relatívne k robotickému ramenu. K dispozícii máme dve RGB-D kamery, ktoré budú upevnené na stenách miestnosti. Využitie dvoch kamier by malo napomôcť detekovaniu objektov, ktoré sú v prekryve.

5.2 Návrh riešenia

Na detekovanie samotných objektov bude využitá dvojica senzorov Kinect v2. Tie sú vhodné vďaka ich multimodálnym spôsobom snímania prostredia, kde dokážeme kombinovať klasickú kameru a hĺbkový senzor. Zariadenia budú statickým prvkom v systéme, teda nebudú meniť svoju polohu.

Cieľ práce môžeme rozložiť na menšie podproblémy a následne nájsť riešenie pre každý z nich. Definoval som si preto nasledujúce podproblémy:

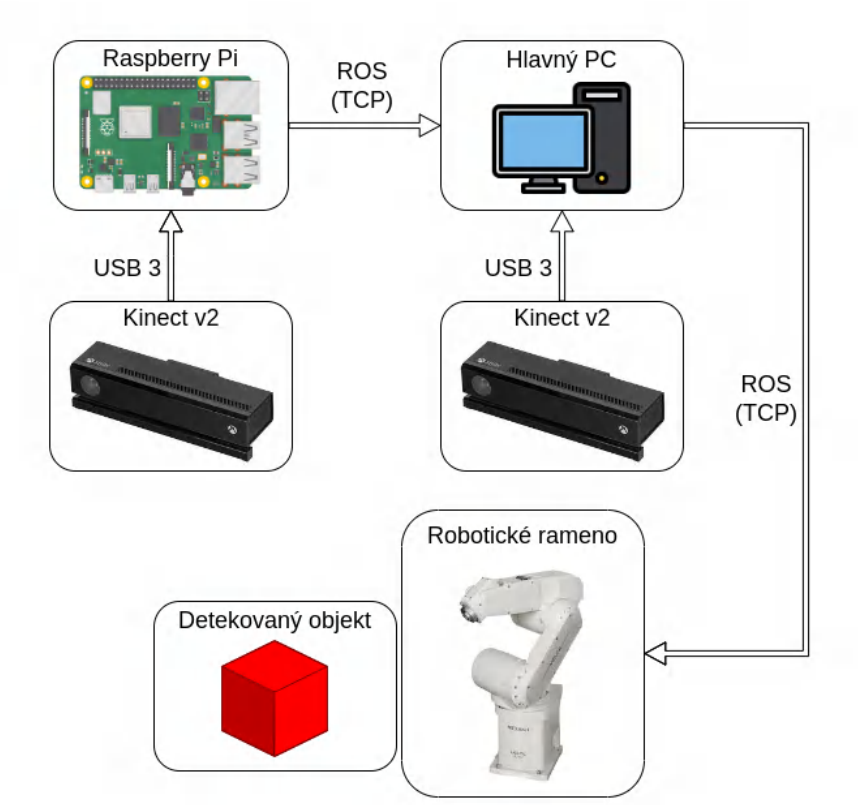
- Umiestnenie dvojice Kinectov na steny
- Zaznamenanie snímkov z Kinectov a predspracovanie údajov
- Spojenie snímkov
- Určenie tvaru objektu
- Určenie polohy objektu

Upevnenie na stenu je pomerne jednoduché vďaka klasickému závitu používanému pre kamery a podobne. Dostatočný bude akýkoľvek stojan, ktorý podporuje následnú úpravu

uhla naklonenia. Drobné zmeny naklonenia môžu byť potrebné pre správnu kalibráciu systému. Senzory budú na stenách umiestnené pod menším uhlom, čo pomôže zväčšiť pokrytie senzoru aj z vrchnej strany v prípade menších objektov.

Pri upevňovaní senzorov treba dbať na ich detekčný rozsah. Kinecty musia byť umiestnené minimálne 0,5 metra od snímaných objektov, avšak nie viac ako 4,5 metra. Senzory musia medzi sebou zvierať dostatočný uhol na to, aby boli schopné zachytiť bočný pohľad pre zamedzenia prekryvu objektov. Ideálne bude umiestnenie pod pravým uhlom. Ostatným problémom sa bližšie budem venovať v nadchádzajúcich podkapitolách.

Na obrázku 5.1 je zobrazený blokový diagram navrhovaného riešenia. V nasledujúcich podkapitolách sa zaoberám konkrétnym popisom návrhu a súčastí systému. Systém bol navrhnutý aj s ohľadom na možné budúce rozšírenie o ďalšie senzory, preto je jeden z Kinectov pripojený pomocou Raspberry Pi. To pomáha rozdeliť záťaž systému na viac zariadení za pomerne malú investíciu do hardvéru systému.



Obr. 5.1: Blokový diagram navrhovaného riešenia

5.3 Zaznamenanie snímkov a predspracovanie údajov

Snímky budú zaznamenané pomocou Kinectov, následne bude výsledok z RGB kamery a hĺbkového senzoru prenesený pomocou USB rozhrania do počítača a spojený do mračna bodov. To bude ideálne na ďalšie spracovanie nasnímaných dát. Na spracovanie snímkov bude využitá knižnica PCL.

Na zaznamenávanie bude využitý balíček v rámci robotického operačného systému. Keďže výsledný systém má byť schopný komunikovať s ramenom Melfa, ktoré má balíčky

len pre ROS 1 budem v nasledujúcom návrhu pracovať s touto verziou. Aktuálne sú stále podporované posledné dve distribúcie a to Melodic a Neotic. Systém by mal byť mierený na najnovšiu distribúciu ROS, avšak ak budú komplikácie s kompatibilitou, distribúcia Melodic je stále podporovaná a overená ďalšími podobnými prácami.

Samotné dáta z kamier bude potrebné spracovať nejakým počítačom. Ako som v teoretickej časti poukázal, keďže objem dát, ktoré dokáže jedna kamera vygenerovať je pomerne veľký, potrebovala by každá kamera v ideálnom prípade svoj vlastný USB radič. Vďaka flexibilite, ktorú ponúka ROS, môžeme pre každý Kinect využiť samostatný počítač a výsledné dáta zasielať pomocou TCP protokolu. Musíme pri tom uvažovať, že potrebujeme jeden stroj na spustenie hlavného procesu pre Robotický operačný systém, ten bude slúžiť ako hlavná jednotka a bude ďalej preposielať dáta medzi jednotlivými uzlami. Tento počítač nemusí byť nutne samostatná jednotka, ale môže obsluhovať priamo jednu z kamier. Pre druhý Kinect nepotrebujeme priveľký výkon a môžeme preto uvažovať nad využitím jednodoskového počítača, akým je napríklad Raspberry Pi 4. Stále je pri tom potrebné myslieť na objem dát, ktoré bude potrebné spracovať, teda odporúčam verziu s minimálne 2 GB RAM. Odosielanie dát na hlavný počítač následne bude prebiehať pomocou samotného robotického operačného systému, teda pomocou správ na určené témy. Tie budú samostatné pre každý Kinect. Hlavný počítač bude v tejto komunikácii vystupovať nie len ako uzol ale aj ako tzv. Master, teda bude sprostredkovávať komunikáciu medzi zariadeniami.

5.4 Spojenie snímkov

Zaznamenávaná oblasť bude tvoriť len polovicu kruhu okolo ramena. Túto časť je možné teoreticky pokryť aj jediným RGB-D senzorom. Avšak v prípade, že sa na scéne nachádza väčšie množstvo objektov, ktoré sú za sebou, teda v prekryve, tento jeden senzor nie je schopný určiť polohu oboch objektov. Preto je potrebné pridať druhý senzor, pomocou ktorého budeme lepšie vedieť rozlíšiť jednotlivé objekty, ich tvary a presnejšie určiť polohu.

Pred správnym spojením snímkov zo senzorov potrebujeme zabezpečiť, že oba senzory sú na správnych pozíciách a sú správne nakalibrované. Na základe toho, že senzory sú statickým prvkom v systéme túto kalibráciu a nastavenie bude potrebné uskutočniť len raz.

5.5 Určenie tvaru objektu

Určenie základného tvaru objektu je možné z mračna bodov, keďže pre samotné uchytenie predmetu nie je potrebné vykonávať klasifikáciu objektu. Pomocou knižnice PCL bude mračno bodov najprv filtrované, aby sme mohli izolovať scénu, na ktorej bude prebiehať detekcia. Následne sa využije postup popísaný v podkapitole 2.1.5, teda budeme pomocou RANSAC algoritmu detekovať plochy na scéne. Pomocou zoskupovania bodov na základe ich vzdialeností následne získame jednotlivé objekty a ich normály.

5.6 Určenie polohy objektu

Poloha objektu bude vypočítaná z polohy jednotlivých súčasti systému a nasnímaného mračna bodov. Najprv je potrebné určiť polohu ramena v priestore, to bude umiestnené na statickej základni. Takže pozícia samotného ramena bude nemenná. Následne sa vypočíta

vzdialenosť objektu od senzoru. Z týchto dvoch údajov je potom možné zistiť polohu objektu relatívne k ramenu, čo je hľadanou hodnotou.

Kapitola 6

Inštalácia softvéru a nastavenie prostredia

Inštalácia softvéru a knižníc potrebných pre spracovanie dát zo senzorov sa ukázala ako náročnejšia úloha ako bolo očakávané. Viacero knižníc nie je pravidelne udržiavaných, čo viedlo k viacerým problémom s kompatibilitou. Často boli pri riešení využité komunitou upravené projekty. Išlo hlavne o problémy so spustením na architektúre ARM, ktorú využíva už spomínané Raspberry Pi. Inštalácia predpokladá už nainštalovaný operačný systém Ubuntu 20.04 na oboch zariadeniach. Pre použitie vizualizačného nástroja RViZ je odporúčaná verzia OS s GUI, to však nie je nutné pre spustenie výsledného projektu. V prípade Raspberry je odporúčané používať aspoň pasívne chladenie na zlepšenie výkonu. Nasledujúce kroky budú niekedy rozdelené vzhľadom na to, že rôzne uzly - hlavný počítač a RPi - si vyžiadali upravené balíčky.

6.1 Inštalácia ROS Neotic

Pre inštaláciu ROS môžeme sledovať návod z dokumentácií pre verziu Neotic[1]. Prvé dva príkazy slúžia na pridanie potrebných repozitárov a klúčov. Nasleduje samotná inštalácia robotického operačného systému.

```
sudo apt update && sudo apt install -y curl git python3-rosdep2


sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc)
    main" > /etc/apt/sources.list.d/ros-latest.list'

curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc |
   sudo apt-key add -

sudo apt update && sudo apt install ros-noetic-desktop-full
```

Nastavenie prostredia pre ROS je možné pomocou nasledujúcich príkazov:

```
echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc

source ~/.bashrc
```

[1] Dostupné z `http://wiki.ros.org/noetic/Installation/Ubuntu`

Nasleduje vytvorenie pracovného adresára a zostavenie aplikácie pomocou nástroja *catkin*:

```
mkdir -p ~/catkin_ws/src && cd ~/catkin_ws/

catkin_make
```

6.2 Inštalácia knižníc pre Kinect v2

Ďalším krokom je inštalácia *libfreenect2*[2], teda ovládača pre Kinect v2. Ovládač podporuje rôzne aplikačné rozhrania pre akceleráciu pomocou grafickej karty, či procesora. Tento návod predpokladá inštaláciu OpenCL, ale podporovane sú aj OpenGL, CUDA, VAAPI a OpenNI2. Tie dokážu urýchliť spracovanie obrazu, prípadne presunúť časť spracovania na grafickú kartu. Pre ich inštaláciu odporúčam sledovať návod dostupný v dokumentácií. V prípade tohto ovládača je inštalácia na architektúre ARM odlišná a vyžaduje si špeciálnu verziu knižnice, ktorá bola upravená komunitou. Pred samotnou inštaláciou je však potrebné upraviť konfiguračné súbory pre operačný systém.

Verzia pre RPi: upraviť konfiguračný súbor `/boot/config.txt` a zväčšiť veľkosť pamäte pre integrovanú grafickú jednotku pomocou nastavenia `gpu_mem=64`.

```
git clone https://github.com/Jyurineko/libfreenect2.git
sudo apt install libudev-dev mesa-utils
sudo apt update
sudo apt install libgles2-mesa-dev
```

Verzia pre PC:

```
git clone https://github.com/OpenKinect/libfreenect2.git
```

Pre elimináciu možných problémov s knižnicou PCL odporúčam ešte pridať do súboru `~/catkin_ws/src/CMakeLists.txt` nasledujúce riadky, tie zabezpečia kompatibilitu s novším štandardom jazyka C++ verzie 14:

```
# Set target C++ standard and required compiler features
set(CMAKE_CXX_STANDARD 14)
set(CMAKE_CXX_STANDARD\_REQUIRED ON)
```

```
sudo apt-get install libusb-1.0-0-dev libturbojpeg0-dev libglfw3-dev
cd libfreenect2
mkdir build && cd build
cmake .. -DCMAKE_BUILD_TYPE=Release -DENABLE_CUDA=OFF -DENABLE_OPENCL=OFF -
    DENABLE_OPENGL_AS_ES31=ON -DENABLE_CXX11=ON -DENABLE_VAAPI=OFF
make -j4
sudo make install
sudo ldconfig
sudo cp ../platform/linux/udev/90-kinect2.rules /etc/udev/rules.d/
```

Po tomto kroku by malo byť možné komunikovať s Kinectom a získavať z neho obraz ako aj mračná bodov. S pripojeným zariadením je možné overiť správnosť inštalácie pomocou

[2]Dostupné z: `https://github.com/OpenKinect/libfreenect2`

spustenia nasledujúceho príkazu. Po spustení sa zobrazia okná s infračerveným, farebným a hĺbkovým obrazom.

```
./bin/Protonect gl
```

Druhou dôležitou knižnicou je *iai_kinect2*[3] slúžiaca na integráciu Kinectu do robotického operačného systému. V tomto prípade potrebujeme znovu verziu upravenú komunitou[4], kvôli podpore knižnice OpenCV vo verzii 4, ktorá je predinštalovaná v operačnom systéme Ubuntu 20.04. Druhým spôsobom ako upraviť tento problém by bola inštalácia kompatibilnej verzie OpenCV, avšak to sa ukázalo ako zložitejšie. Keďže použité funkcie sú plne kompatibilné s OpenCV vo verzii 3, bolo len potrebné pridať verziu 4 do matice podporovaných verzií.

```
cd ~/catkin_ws/src/
git clone https://github.com/paul-shuvo/iai_kinect2_opencv4
cd iai_kinect2_opencv4
rosdep update
rosdep install -r --from-paths .
cd ~/catkin_ws
catkin_make -DCMAKE_BUILD_TYPE="Release"
soruce ~/catkin_ws/devel/setup.sh
```

Teraz by malo fungovať prepojenie medzi Kinectom a operačným systémom. Vyskúšať toto spojenie môžeme pomocou nasledujúcich príkazov spustených v osobitných termináloch. Prvý z nich spustí hlavný uzol pre ROS, zvyšné dva sa postarajú o odosielanie respektíve prijímanie a zobrazenie mračna bodov zaznamenaného z Kinectu v reálnom čase.

```
roscore
rosrun kinect2_bridge kinect2_bridge
rosrun kinect2_viewer kinect2_viewer
```

6.3 Nastavenie komunikácie medzi uzlami

Pre zabezpečenie správnej komunikácie medzi uzlami je potrebné definovať IP adresy jednotlivých zariadení. V robotickom operačnom systéme je na to definovaných niekoľko premenných prostredia.

- `ROS_IP` - slúži na definovanie vlastnej IP adresy, teda adresy pod ktorou bude zariadenie v komunikácii vystupovať.
- `ROS_HOSTNAME` - podobne ako `ROS_IP` slúži na identifikáciu uzla, ale využíva pri tom doménove meno namiesto IP adresy. V prípade, že sú definované ako `ROS_IP` tak aj `ROS_HOSTNAME`, využije sa `ROS_HOSTNAME`.
- `ROS_MASTER_URI` - definuje adresu (URL) a port hlavného (master) uzla

Pre nastavenie týchto hodnôt môžeme použiť príkaz `set` alebo `export`. Na všetkých zariadenie je pri tom potrebné nastaviť `ROS_IP` alebo `ROS_HOSTNAME`. Ďalej je potrebné nastaviť `ROS_MASTER_URI` na všetkých zariadeniach okrem hlavného uzla, tam je táto hodnota preddefinovaná na `http://localhost:11311/`.

[3]Dostupné z: `https://github.com/code-iai/iai_kinect2`
[4]Komunitou upravená verzia: `https://github.com/paul-shuvo/iai_kinect2_opencv4`

6.4 Kalibrácia senzorov

Po nastavení prostredia je potrebné vykonať kalibráciu senzorov. Výsledkom kalibrácie je vzájomné posunutie jednotlivých senzorov, ktoré vzniklo nepresnosťami pri výrobe Kinectov. Hlavným cieľom je vylepšiť mapovanie farebného obrazu na hĺbkový obraz v mračnách bodov. Tento proces je potrebné pre každý Kinect vykonať raz, následne je možné výsledky kalibrácie zdieľať medzi zariadeniami, teda z jedného počítača na ďalší.

Kalibračné skripty použité pre túto prácu pochádzajú z nástroja *iai_kinect2*[5]. Pre kalibráciu je potrebné vytlačiť šachovnicu a následne ju uchytiť na pevný povrch. Je dôležité, aby veľkosť mriežky na papieri bola presná, preto je odporúčané skontrolovať vzdialenosti pomocou posuvného meradla. Pred meraniami je potrebné znova zapnúť *kinect2_bridge*, môžeme pri tom obmedziť počet snímkov za sekundu. Pri spúšťaní príkazu uvidíme v konzole vypísané sériové číslo Kinectu, to bude dôležité v nasledujúcich krokoch a je potrebné si ho poznačiť. Zároveň si pripravíme zložku, do ktorej budú ukladané výsledky kalibrácie.

```
rosrun kinect2_bridge kinect2_bridge _fps_limit:=2
mkdir ~/kinect_cal_data; cd ~/kinect_cal_data
```

Následne na scénu pridáme šachovnicu. V tomto prípade bola použitá mriežka veľkosti 6x9. Pri kalibrácií sa vytvorí niekoľko snímkov, na každom musí byť mriežka detekovaná pomocou farebných bodov a následne pomocou klávesy `S` snímok uložíme. Šachovnicu na scéne preložíme a znovu získame snímok. Tento proces opakujeme až kým nemáme dostatok snímkov. Podľa rôznych zdrojov je potrebné zhotoviť až stovky snímkov[4], ale pre začiatok postačí 10-20 pre každý krok. Akonáhle máme dostatok snímkov, ukončíme zaznamenávanie pomocou klávesy `Q`. Prvý z príkazov slúži na spomínané zaznamenávanie snímkov a druhým zo snímkov vypočítame posun.

```
rosrun kinect2_calibration kinect2_calibration chess6x9x0.03 record color
rosrun kinect2_calibration kinect2_calibration chess6x9x0.03 calibrate
    color
```

Po skončení druhého príkazu vznikne v zložke súbor `calib_color.yaml`, v ktorom sú uložené výsledky pre RGB kameru. Tieto dva príkazy opakujeme postupne s obmenou *color* za *ir, sync* a *dept*. Dokopy teda spustíme každý príkaz štyrikrát. Posledným krokom je vytvorenie adresára s názvom podľa sériového čísla Kinectu a presunutie výsledkov do správneho adresára. V príkazoch nahradíme *{seriove cislo}* za sériové číslo, ktoré sme si poznačili. Pomocou tohto adresára bude *kinect2_bridge* vedieť spustiť zaznamenávanie so správnou kalibráciou aj v prípade, že sa na jednom počítači využíva viacero Kinectov.

```
roscd kinect2_bridge/data; mkdir {seriove cislo} && cd {seriove cislo}
cp ~/kinect_cal_data/*.yaml .
```

Na obrázkoch 6.1 môžeme vidieť dôležitosť kalibrácie senzorov. Pre znázornenie poslúžila krabica z Kinectu vďaka jej kontrastnej farbe. Pred kalibráciou (obrázok 6.1a) je jasne viditeľný posun hĺbkového senzoru oproti farebnej kamere. Z tohto dôvodu môžeme pozorovať zlé nanesenie obrazu, celý farebný obraz je posunutý doľava a preteká na objekty za ním, respektíve pred ním. Po kalibrácii (viď obrázok 6.1b) sa nám podarilo tento nežiadúci efekt takmer úplne odstrániť a na objekt so šachovnicou v pozadí preteká len minimum zelených bodov. Zároveň môžeme vidieť, že táto kalibrácia spôsobila v obraze určité artefakty, avšak tie nebudú mať žiaden vplyv na výsledné mračno bodov.

[5]Celý popis kalibrácie a šachovnicu sú dostupné z: `https://github.com/code-iai/iai_kinect2/tree/master/kinect2_calibration#calibrating-the-kinect-one`

(a) Mračno bodov pred kalibráciou

(b) Mračno bodov po kalibrácii

Obr. 6.1: Porovnanie mračien bodov pred a po kalibrácii

Kapitola 7

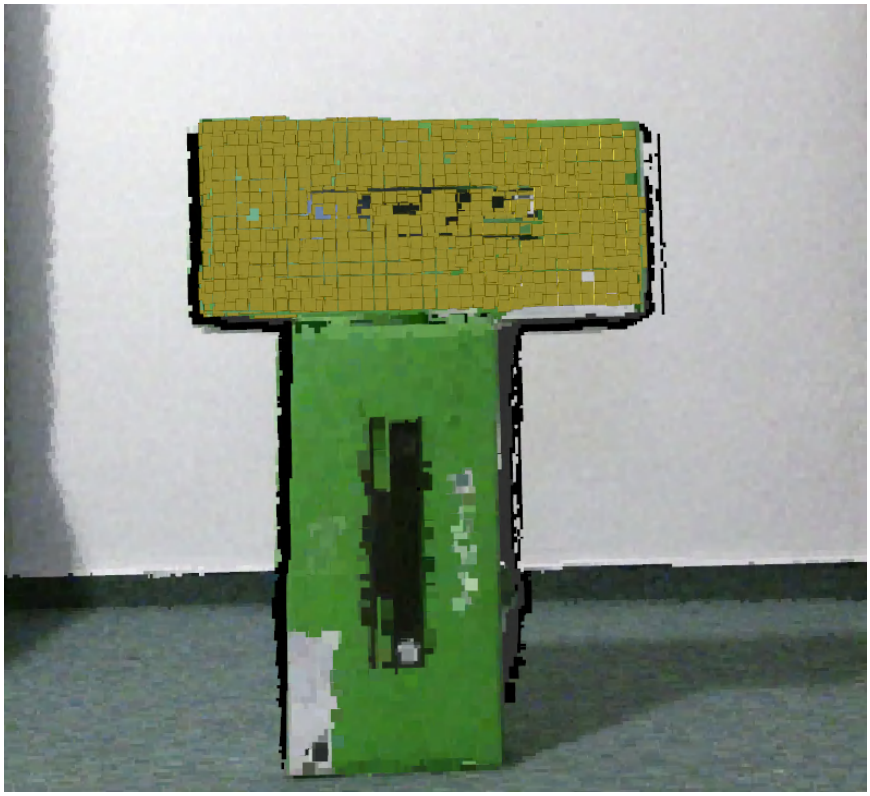
Realizácia

Pre implementáciu bol vybraný jazyk Python3, vďaka jeho podpore v ROS ako aj v PCL pomocou mapovacieho balíčka. Bol preferovanou voľbou hlavne vďaka mojím skúsenostiam s jazykom a jednoduchosti použitia. Okrem iného boli využívané v značnej miere aj knižnice ako OpenCV pre spracovanie obrazu a Numpy pre zjednodušenie, vektorizáciu, a teda urýchlenie výpočtov nad viacrozmernými poľami. V nasledujúcej kapitole bude pre vizualizáciu jednotlivých krokov zobrazený jednoduchý príklad s kvádrovým objektom. Obrázky predstavujú mračná bodov, ktoré boli zachytené a zobrazené pomocou vizualizačného nástroja RViz.

Program je implementačne rozdelený na dve základné časti - kalibrácia a samotná detekcia. Ďalej sa v systéme nachádza niekoľko ďalších modulov, ktoré sú zdieľané medzi základnými časťami. Tie zväčša zabezpečujú komunikáciu medzi jednotlivými uzlami a na to potrebné transformácie.

7.1 Detekcia pomocou jedného Kinectu

Implementácia riešenia začala najprv detekovaním objektov s využitím len jedného senzoru pre jednoduchosť. Mračno bodov bolo predspracované pomocou *PointCloudLibrary* (PCL), najprv však bolo nutné previesť štruktúru mračna bodov z formátu využívaného v ROS do formátu PCL. Hlavný rozdiel je v reprezentácii samotných štruktúr. Typ využívaný v ROS správach je reprezentovaný zvyčajne ako jednorozmerné pole bajtov, kde je definované poradie hodnôt a krok, s ktorým prichádza ďalší bod mračna. Narozdiel od ROS, typ v rámci PCL používa na reprezentáciu súradníc desatinné čísla typu *float* a pre každú farbu celé čísla typu *uint8*. Tieto rozdiely prinášajú svoje výhody v efektívnosti zasielania a ukladania dát, respektíve v jednoduchosti spracovania, avšak zároveň prinášajú nevýhodu v podobe potrebnej konverzie medzi typmi. Následne bolo využité filtrovanie pomocou voxelovej mriežky a filtrovanie mračna bodov na oblasť detekcie. Tá bola v tomto prípade vymedzená len pre potreby tohto experimentu a bude v nasledujúcich častiach definovaná pre potreby laboratória, v ktorom bude systém nasadený. Pomocou segmentácie a zhlukovania boli vytvorené mračná bodov detekovaných objektov. Teda bola využitá detekcia plôch, výsledok je zachytený na obrázku 7.1. Na snímke je vidieť zobrazenie mračna bodov získaného z Kinectu, ako aj mračno bodov detekovaného objektu, pre zvýraznenie žltou. Zároveň môžeme vidieť efekt filtrovania, pomocou ktorého sa nám podarilo detekovať len vrchný objekt, keďže spodný objekt sa nachádza mimo detekovanej zóny. Tieto dáta boli zachytené a zobrazené pomocou vizualizačného nástroja RViz.



Obr. 7.1: Detekovaný objekt (zvýraznený žltou) pomocou jedného Kinectu

7.2 Kalibrácia v priestore

Pre správne fungovanie programu potrebujeme definovať, kde v priestore sa nachádzajú kamery, teda ich vzájomnú polohu. Na to môžeme využiť hneď niekoľko spôsobov. V rámci prieskumu podobných prác som zistil, že sa využívajú hlavne tieto tri spôsoby:

- **OpenCV šachovnica**
- **ICP** (Iterative Closest Point), teda iteratívne hľadanie najbližšieho bodu
- **ArUco značka**, na prvý pohľad podobná QR kódu

Rozhodol som sa teda analyzovať tieto spôsoby a nájsť čo najvhodnejší pre túto prácu. V nasledujúcej sekcii sa preto pozrieme na výhody a nevýhody jednotlivých algoritmov a nakoniec na základe nich vyberieme najvhodnejší. Základnou požiadavkou je, aby metóda vedela uložiť kalibráciu a následne podľa nej prepočítavať polohu. Pri tomto spôsobe nie je potrebné kalibrovať každý snímok osobitne. Tento výpočet by bol redundantný nakoľko poloha kamier sa v priestore meniť nebude, prípadne pri zmene je možné znovu spustiť kalibráciu.

7.2.1 OpenCV šachovnica

Ide o jeden z najbežnejších spôsobov kalibrácie kamier. Využíva sa šachovnica, pri ktorej poznáme počet riadkov, stĺpcov a rozmer jednotlivých štvorcov na šachovnici. Pomocou tohto vzoru je následne možne zistiť distorziu (teda skreslenie) obrazu spôsobenú napríklad

zakrivením šošovky kamery. Práve na tento účel bol tento spôsob využitý v podkapitole 6.4, kde sme kalibrovali jednotlivé senzory v rámci jedného Kinectu. Tento systém je však možné využiť aj na stereo kalibráciu, teda pri použití viacerých kamier. Výhodou tejto metódy je, že je pomerne jednoduchá na implementáciu, knižnica OpenCV ponúka všetky potrebné funkcie na výpočet. Ďalšou výhodou je, že samotná šachovnica nemusí byť prítomná na scéne po kalibrácii. Pre kalibráciu je potrebné nahrať až stovky snímkov (v závislosti na požadovanej presnosti) z každého zariadenia a následne spracovať, čo vedie k nevýhode, že samotná kalibrácia môže byť časovo náročná. Pri každom čo i len menšom pohybe senzoru je potrebné túto kalibráciu opakovať.

7.2.2 ICP

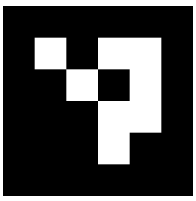
Tento algoritmus postupne opakovane vyberá body v jednom mračne a hľadá odpovedajúci bod v mračne druhom. Následne spočíta odhadovanú chybu riešenia a prípadne opakuje, kým nedosiahne požadovanú presnosť. Výhodu v tomto prípade majú objekty so špecifickými znakmi. Najväčším problémom je, že algoritmus môže ľahko skončiť v lokálnom optimálnom riešení, čo spôsobí nesprávny výsledok kalibrácie. Výhodou je, že táto metóda nepožaduje pridanie ďalších oporných bodov na scénu. Avšak metóda môže byť výpočtovo náročnejšia. Podľa rôznych štúdii (napríklad [7] alebo [11]), ICP je možné použiť teoreticky až do rotácie o 120 stupňov a stále dosiahnuť akceptovateľné výsledky, pričom ale záleží na objekte a ostatných parametroch ako je vzdialenosť jednotlivých kamier od objektu a podobne. Dôležitá pri tom nie je samotná rotácia objektu, ale počet zhodných znakov medzi mračnami teda pomer prekryvu. Rotovaním objektu sa táto podobnosť zmenšuje a teda je náročnejšie nájsť správnu orientáciu medzi kamerami. Ako už bolo spomínané v podkapitole 2.1.3, často je tento spôsob využívaný po orezaní mračna bodov na hľadaný objekt, kde je presnosť výrazne lepšia. Pritom poznáme model hľadaného objektu a pomocou tejto metódy sa snažíme zistiť jeho orientáciu v priestore. V prípade tejto práce ICP algoritmus, teda nie je vhodný na priame zistenie vzájomnej polohy kamier, avšak metóda bude užitočná na doladenie polohy, ktorá bola získaná pomocou inej metódy.

7.2.3 AruCo

AruCo (skratka z názvu *Augmented Reality University of Cordoba*) je skupina značiek (markerov), ktoré sa využívajú v rozšírenej realite na definovanie oporných bodov v obraze. Existuje niekoľko druhov týchto značiek podľa veľkosti mriežky, v ktorej je samotná značka umiestnená. Značky majú priradené svoje jedinečné identifikačné číslo v rámci konkrétnej skupiny. Vo svojej podstate je tento spôsob podobný hľadaniu šachovnice. Na scénu sa umiestni nejaký známy oporný bod, ktorého veľkosť a tvar poznáme, a snažíme sa ho na scéne detekovať.

Podobne ako šachovnica majú AruCo plnú podporu v rámci knižnice OpenCV, teda implementácia tohto riešenia bude pomerne jednoduchá. Výhodou oproti šachovnici je, že dokážeme zistiť presnú orientáciu značky v priestore. Nevýhodou je, že niektoré značky sú v detekovaní menej spoľahlivé ako iné, takže je potrebné nájsť správne vzory, ktoré budú pri detekcii čo najspoľahlivejšie. Zvyčajne ostávajú tieto značky staticky na scéne aj po dokončení kalibrácie a môžu fungovať aj dynamicky, teda výpočet polohy prebieha s každým snímkom znova - výhodnejšie hlavne pre dynamické systémy, kde sa poloha kamery často mení. Značky sa zvyčajne vyžívajú v skupinách, kde sa snažíme vytvoriť čo najväčší počet oporných bodov v obraze pre čo najpresnejšie výsledky. Zvyčajne teda táto metóda potrebuje menej snímok na získanie podobnej presnosti ako v prípade šachovnice.

Kalibrácia pomocou AruCo značiek bola použitá aj v podobnej práci [1], ktorá sa snažila o vzájomnú synchronizáciu správ zo 4 zariadení Kinect v2 s obdivuhodnou presnosťou merania až na 9 metrov. Okrem iného sa táto práca zaoberala zisťovaním spoľahlivosti jednotlivých značiek. Práca pri tom pracovala so značkami 4x4 identifikátormi 1, 13 a 40. Tie boli spoľahlivé a konzistentné pri detekcii, preto som sa rozhodol práve tieto značky použiť aj v mojej práci. Z môjho testovania vychádzajú tieto značky tiež spoľahlivo, zároveň som testoval aj ďalšie značky, keďže plánujem využitie 4 značiek. Značka s identifikátorom 0 je taktiež spoľahlivá a nemal som problémy s jej detekciou, naopak v takmer polovici prípadov som zistil problémy so značkou 30.



Obr. 7.2: Príklad AruCo značky 4x4

7.2.4 Vybrané riešenie

Pôvodné riešenie počítalo s využitím šachovnice. Implementácia bola síce pomerne jednoduchá, avšak zaznamenanie potrebného počtu snímkov bolo relatívne zdĺhavé. Počas implementácie som nedokázal získať spoľahlivé výsledky, odhadovaná vzájomná orientácia zariadení nebola totožná s tou v reálnom priestore. To odpovedalo aj dosiahnutej strednej kvadratickej odchýlke, ktorá dosahovala hodnoty za hranicou akceptovateľných hodnôt. Keďže som nedokázal získať spoľahlivé výsledky pomocou tejto metódy rozhodol som sa pre využitie ďalšieho zo spomínaných riešení - AruCo značiek.

V nasledujúcich častiach bude potrebné rozlíšiť Kinecty na hlavný a vedľajší, keďže počas kalibrácie nebudú rovnocenné. Pre jednoduchosť si teda Kinecty pomenujeme ako *master* a *slave*. V tejto komunikácii pri tom budeme uvažovať master ako hlavný senzor, ktorý bude využívaný ako referenčný pre kalibráciu a zároveň tento bude pripojený k hlavnému počítaču.

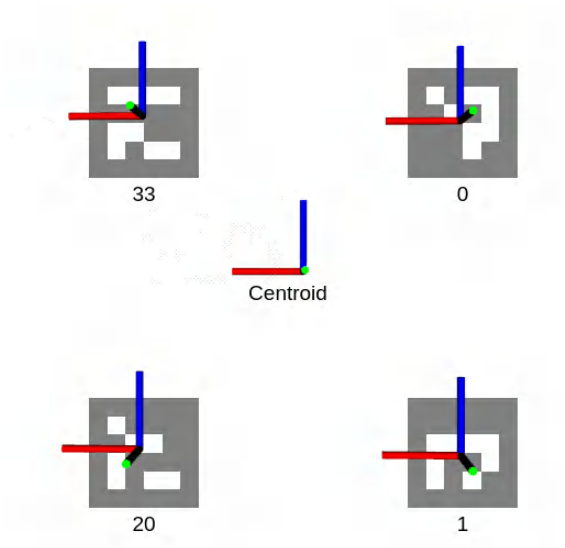
Algoritmy pre vzájomnú kalibráciu Kinectov boli prevzaté z [1] a upravené pre potreby tejto práce. Hlavným rozdielom oproti spomínanej práci je práve využitie master senzoru ako referenčného bodu. V publikácii využívali simulovanie referenčného mračna bodov pomocou OpenCV. To si vyžiadalo presné usporiadanie značiek v priestore a zároveň bolo potrebné poznať ich exaktné umiestnenie. Tým sa systém stal menej flexibilný a v prípade posunutia značky, alebo prípadne jej odstránenia zo scény, bolo potrebné celé meranie a umiestnenie značiek na svoje miesta urobiť znova. V prípade môjho riešenia nepredpokladám konkrétne umiestnenie značiek v priestore. Jediným požiadavkom je prítomnosť aspoň 3 značiek, ktoré budú ohraničovať detektované územie. Využitie viacerých značiek je možné a môže do určitej miery zvýšiť presnosť prvotnej kalibrácie, keďže vzniká viac referenčných bodov. Pre doladenie kalibrácie využijeme algoritmus ICP.

7.2.5 Výpočet vzájomnej polohy kamier

Pred spustením kalibrácie je potrebné v priestore rozmiestniť AruCo značky. Ako bolo spomínané, na ich presnom umiestnení nezáleží avšak snažíme sa pokryť čo najlepšie oblasť,

do ktorej budú v budúcnosti prichádzať objekty na detekciu. Napríklad môžeme využiť umiestnenie podobné tomu z obrázka 7.3. Jedinou značkou, ktorú musíme umiestniť presne je značka s číslom 0. Tú je potrebné umiestniť k robotickému ramenu a bude definovať jeho polohu. Táto časť bude ďalej rozoberaná v kapitole 7.4.1.

Kalibrácia sa skladá z dvoch častí. Prvou je zaznamenávanie údajov a druhou je výpočet polohy na základe týchto údajov. Zaznamenávanie začína získaním hĺbkového a farebného obrazu z každého Kinectu. Vo farebnom obraze hľadáme AruCo značky pomocou knižnice OpenCV. Tá nám vráti zoznam idententifikátorov značiek a pozície jednotlivých rohov každej značky. Následne podľa nich vytvoríme masku, ktorá bude obsahovať len pixely, na ktorých bola nájdená značka. Táto maska je potom aplikovaná na získaný hĺbkový obraz. Týmto spôsobom sme dokázali transformovať pozície značiek z 2D farebného obrazu do 3D priestoru. Pre každú značku je následne vypočítané ťažisko ako priemer jednotlivých bodov mračna, viď osi na jednotlivých značkách na obrázku 7.3. Tieto pozície sa vo formáte *JSON* uložia pre každý Kinect osobitne. Okrem toho sa uložia aj mračná bodov získané spojením farebných a hĺbkových snímkov, tie budú potrebné v ďalšom výpočte. Zaznamenávanie je týmto ukončené.



Obr. 7.3: Ilustrované značky s ťažiskami a stredom

Samotný výpočet je následne vykonaný bez interakcie s kamerami a môže trvať až niekoľko sekúnd. Na začiatku sa načítajú zaznamenané ťažiská značiek. Tie využijeme na výpočet rotácie a posuvu (translácie) medzi súradnicovými systémami Kinectov pomocou zhodného zobrazenia. Je dôležité, aby tieto body boli zoradené pomocou identifikátorov, a teda body postupne za sebou korešpondovali medzi zoznamami. Pre nájdenie polohy potrebujeme vyriešiť nasledujúcu rovnicu:

$$R \times A + t = B \tag{7.1}$$

kde A je zoznam bodov master senzoru, B je zoznam bodov slave senzoru, R je rotačná matica a t je translačný vektor. Začneme hľadaním optimálnej rotácie, kde využijeme metódu singulárneho rozkladu (SVD). Pred nájdením rotácie potrebujeme dostať oba súradnicové systémy do spoločného stredu, ako je zobrazené na obrázku 7.4. Tým vieme

odstrániť transláciu a izolovať len rotáciu. Najprv je potrebné nájsť stred každej množiny bodov - viď *centroid* na obrázku 7.3. Tie vypočítame ako priemer bodov:

$$\begin{aligned} centroid_A &= \frac{1}{N}\sum_{i=1}^{N} A^i \\ centroid_B &= \frac{1}{N}\sum_{i=1}^{N} B^i \end{aligned} \tag{7.2}$$

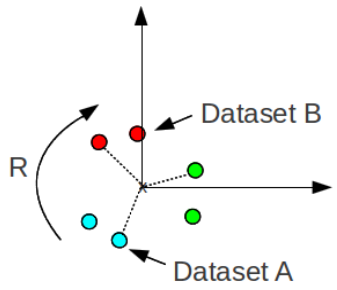
kde A^i a B^i sú vektory 3x1 a predstavujú ťažiská jednotlivých značiek. Ďalším krokom je výpočet kovariančnej matice H a využitie singulárneho rozkladu, čím dostaneme hľadanú rotáciu R:

$$\begin{aligned} H &= (A - centroid_A) \times (B - centroid_B)^T \\ [U, S, V] &= SVD(H) \\ R &= VU^T \end{aligned} \tag{7.3}$$

Výraz $A - centroid_A$ je operácia, kde od každého bodu odpočítame súradnice stredu a tak efektívne presunieme počiatok súradnicovej sústavy do bodu $centroid_A$.

Teraz, keď už poznáme R, môžeme vypočítať t využitím pôvodnej rovnice 7.1 a dosadím priemeru stredov značiek z 7.2 za jednotlivé matice:

$$\begin{aligned} R \times A + t &= B \\ R \times centroid_A + t &= centroid_B \\ t &= centroid_A - R \times centroid_A \end{aligned} \tag{7.4}$$



Obr. 7.4: Hľadanie optimálnej rotácie[1]

Takto získaná poloha môže mať drobné odchýlky od skutočného usporiadania, a teda by spôsobovala mnoho nepresností. Preto je potrebné polohu doladiť pomocou algoritmu ICP. Ten zohľadňuje pôvodne vypočítané usporiadanie a vypočíta odhadovanú chybu riešenia, ktorú sa snaží znížiť. Pre výpočet bola využitá knižnica Open3D[2], ktorá poskytuje aj vlastné nástroje na vizualizáciu. [19]

Výsledok kalibrácie je následne publikovaný do ROS ako transformácia slave Kinectu voči referenčnému master Kinectu. Zároveň sú tieto dáta uložené vo formáte *JSON* pre potreby využitia v hlavnej časti programu.

[1] Prevzaté z: `https://nghiaho.com/?attachment_id=807`
[2] Dostupné z: `http://www.open3d.org/`

7.3 Spojenie mračien

Pred samotným spájaním mračien potrebujeme vyriešiť problém časovej synchronizácie. Je dôležité aby obe mračná boli zaznamenané v rovnakom momente inak by poloha prípadne hľadaného objektu mohla byť medzi mračnami odlišná. ROS poskytuje nástroje na synchronizáciu správ, tie zahŕňajú synchronizáciu podľa presného času, ale aj aproximačné riešenia. Keďže naše uzly nebežia na rovnakom zariadení a ich hodiny môžu byť mierne posunuté, budeme využívať aproximačný synchronizátor. Ten páruje snímky s najpodobnejšou časovou známkou a odosiela na ďalšie spracovanie do špecifikovanej funkcie.

Samotné spojenie mračien je potom pomerne jednoduché vďaka správnej synchronizácii a transformácii. Teda vďaka správnej kalibrácii môžeme mračná spojiť do jedného poľa bez potreby ďalších výpočtov. Je potrebné publikovať vzájomnú polohu Kinectov a o transformáciu mračna sa postará robotický operačný systém.

Kvôli chybe pri zostavení jedného z balíčkov pre ROS som však nemohol využiť vstavané metódy pre transformácie mračien bodov. Bolo teda potrebné túto časť kódu pribaliť do výsledného riešenia. Kód sa nachádza v súbore `tf2_sensor_msgs.py` a oproti pôvodnému zdroju[3] obsahuje mierne úpravy pre opravu chýb v transformácii mračien.

7.4 Odhad polohy objektu

Základný odhad polohy k objektu môžeme uskutočniť na základe detekovaného mračna bodov, ktorého získavanie bolo opísané v podkapitole 7.1. Z tohto mračna bodov vieme získať základnú predstavu o objekte, jeho tvare a pozícii relatívne ku Kinectu. Jeho polohu v priestore vieme získať jednoducho z mračna bodov, teda urobíme priemer bodov a dostaneme stred objektu. Rovnako aj tvar objektu vieme už zo získaného mračna. Posledným hľadaným znakom, ktorý nie je jednoducho odvoditeľný z mračna bodov je jeho rotácia v priestore. Na to využijeme podobný spôsob ako pri kalibrácii senzorov v podkapitole 7.1. Presnejšie vypočítame stred mračna, na to využijeme rovnicu 7.2 a následne odpočítame od každého bodu hodnotu stredu. Týmto sme presunuli počiatok sústavy do stredu mračna. To nám pomôže pri počítaní rotácie objektu, kde využijeme SVD, podobne ako v rovnici 7.3. Výsledný normálny vektor je posledný stĺpec matice V.

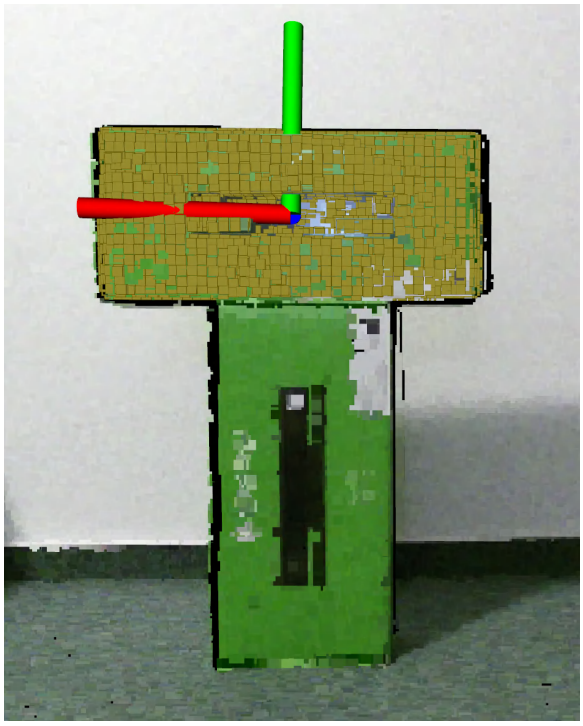
Výslednú polohu môžeme vidieť zobrazenú na obrázku 7.5, tá je vyznačená pomocou osi, ktorá je umiestnená v strede mračna bodov detekovaného objektu.

7.4.1 Prepočet polohy relatívne k ramenu

Po získaní polohy relatívne ku Kinectu je potrebné túto polohu prepočítať relatívne k pomyselnému bodu 0 v súradnicovej sústave, ktorým je v našom prípade robotické rameno. Keďže jeho poloha je v priestore nemenná rozhodol som sa využiť statickú polohu, ktorú sme definovali v podkapitole 7.2.5, pomocou AruCo značky s číslom 0. Tá bude predstavovať základňu ramena.

$$\left(\begin{array}{ccc|c} & \mathbf{R} & & t \\ \hline 0 & 0 & 0 & 1 \end{array}\right)^{-1} = \left(\begin{array}{ccc|c} & \mathbf{R}^T & & -\mathbf{R}^T \times t \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \tag{7.5}$$

[3] `https://github.com/ros/geometry2/blob/noetic-devel/tf2_sensor_msgs/src/tf2_sensor_msgs/tf2_sensor_msgs.py`



Obr. 7.5: Odhadovaná poloha objektu (žltou) zobrazená pomocou osi

Keďže z kalibrácie poznáme umiestnenie značky a teda jej rotačný a translačný vektor vzhľadom k master Kinectu, môžeme využiť inverznú homogénnu transformáciu. Pomocou tej vypočítame opačnú transformáciu, teda polohu master Kinectu ku značke a efektívne presunieme počiatok celej súradnicovej sústavy do tohto počiatku pre všetky body na scéne. Začneme prekladom rotačného vektoru na rotačnú maticu **R**, na to využijeme Rodriguesov rotačný vzorec z knižnice OpenCV. Keď už poznáme rotačnú maticu **R** a translačný vektor t môžeme tieto dosadiť do rovnice 7.5 a tak získať inverznú transformačnú maticu. Týmto sme si definovali polohu ramena ako počiatku sústavy a preto môžeme získané polohy z Kinectov prepočítať pomocou ROS transformácií. Výstupom programu je zoznam polôh objektov, ktorý je publikovaný na témy s predponou *obj_estimation/poseX*, kde X značí poradie objektu, v ktorom boli objekty na scéne nájdené. Spolu s polohou sú taktiež publikované aj mračná bodov jednotlivých objektov pre prípadné ďalšie spracovanie. Tie boli zároveň využívané pri ladení riešenia a vizualizované v RViZ.

7.5 Spustenie programu

Program je jednoduché spustiť pomocou príkazu `roslaunch` v príkazovom riadku. Celý balíček pozostáva z dvoch častí a to kalibrácie a samotného uzla pre odhad polohy.

Prvým krokom kalibrácie je zaznamenanie polohy značiek v priestore. Pre zahájenie kalibrácie využijeme nasledujúci príkaz:

```
roslaunch object_pose_estimation calibration.launch action:=record sensor:=
    master serial_num:=009842564647
```

Príkaz zahŕňa spustenie uzla pre zaznamenávanie obrazu z Kinectu na lokálnom stroji ako aj zaznamenávanie údajov pre kalibráciu. V prípade problémov je však možné spustiť

kalibráciu osobitne od samotného Kinectu pomocou argumentu `run_bridge`, to sa môže hodiť na sprehľadnenie výpisov programu. Príkaz je potrebné spustiť pre každý Kinect osobitne. Pre definovanie, či sa jedná o master alebo slave môžeme využiť argument `sensor`. Zároveň je potrebné špecifikovať sériové číslo Kinectu pomocou `serial_num`, keďže pre správnu kalibráciu medzi zariadeniami je najprv potrebné zohľadniť lokálnu kalibráciu každého zariadenia. Zaznamenávanie prebieha v nekonečnej slučke, je potrebné sledovať výstup programu a overiť, že všetky značky na scéne boli označené ako nájdené pre obe zariadenia. Po zaznamenaní prostredia prejdeme k samotnej kalibrácií pomocou nastavenia argumentu `action` na *calibrate*. Pre podrobnejší popis všetkých argumentov je možné využiť argument `-ros-arg`.

Pre spustenie samotnej aplikácie na detekciu objektov môžeme použiť nasledujúci príkaz:

```
roslaunch object_pose_estimation object_pose_estimation.launch
```

Program podporuje limitáciu vzdialenosti od master Kinectu vo všetkých osiach. To je hlavne využiteľné v prípade, že chceme obmedziť a koncentrovať územie, v ktorom prebieha detekcia. Limitácie je možné nastaviť pomocou parametrov programu.

Pre spustenie slave Kinectu je potrebné na Raspberry Pi nastaviť spojenie medzi zariadeniami, ako bolo vysvetlené v podkapitole 6.3. Následne je možné spustiť samotné zaznamenávanie pomocou:

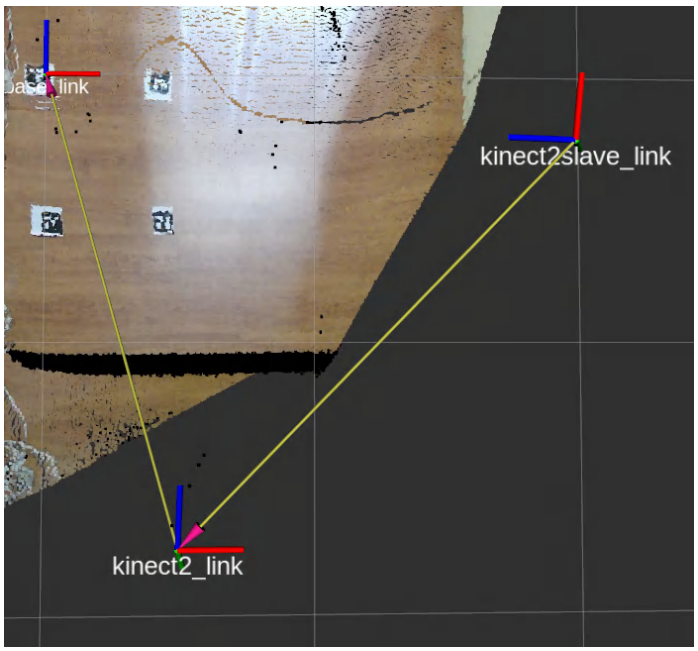
```
roslaunch kinect2_bridge kinect2_bridge.launch fps_limit:=1 publish_tf:=
    true base_name:=kinect2slave
```

Kapitola 8

Experimenty

Ako bolo poznamenané aj v úvode a rozbore, cieľom práce nebolo zlepšiť presnosť odhadu polohy objektov pomocou pridania ďalšieho Kinectu, ale redukovať možné prekrytia medzi objektami. Teda v tejto časti sa zameriam na presnosť riešenia ako takého oproti skutočným vzdialenostiam, ale hlavne na schopnosť systému nájsť a identifikovať objekty, ktoré sa nachádzajú priamo za sebou a zakrývajú výhľad jednej z kamier na druhý objekt.

Experimenty boli prevedené v domácom prostredí. Kinecty boli umiestnené približne vo vzdialenosti 2 metre od detekovaného územia, to bolo vytýčené štyrmi AruCo značkami. Detekované územie malo pôdorys v tvare štvorca so stranou 50 centimetrov. Na spracovanie sa využívali mračná bodov v SD kvalite (512 x 424) pre odľahčenie záťaže pre RPi. Pri experimentoch boli využívané rôzne štvorcové a kvádrové objekty s veľkosťami jednej strany od 10 do 45 cm. Tieto objekty boli postupne umiestňované na rôzne miesta na scéne. Podobne ako v predchádzajúcej kapitole budú aj v tejto vizualizované jednotlivé kroky pomocou nástroja RViZ.

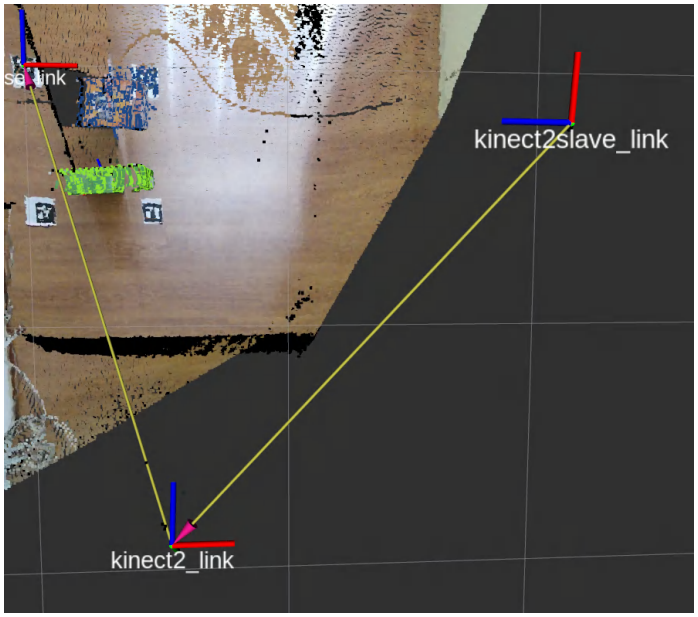


Obr. 8.1: Scéna pre experimenty zachytená po kalibrácii

Na obrázku 8.1 môžeme vidieť scénu, na ktorej prebiehali experimenty. Približne v strede ľavej časti obrázku je zároveň pozorovateľný prechod medzi mračnami v podobe čierneho zhluku bodov. To na zaznamenávané objekty nemá vplyv, keďže tie sa na budú nachádzať len v časti vyznačenej pomocou značiek. Avšak treba mať na pamäti, že oblasť, do ktorej umiestňujeme objekty, musí byť v detekčnom rozsahu oboch Kinectov, inak tento šum spôsobí zlé odhady. Vizualizácia taktiež zachytáva polohy a transformácie jednotlivých Kinectov v priestore pomocou ROS správy typu `TF2`.

Počas prípravy experimentov bolo zistené, že lesklý podklad môže skresľovať ako kalibráciu tak aj detekciu objektov, tento odlesk je možné vidieť aj na obrázku 8.1. Rozdiely boli malé, avšak niekedy znamenali, že Kinecty neboli schopné identifikovať AruCo značky, čo viedlo k opakovaniu časti kalibrácie. Preto som sa rozhodol presunúť detekovanú oblasť mimo tohto odrazu. Pre prípadné nasadenie to môže byť taktiež jeden z rozhodujúcich faktorov kam umiestniť jednotlivé zariadenia.

Pri samotnom meraní vzdialenosti bol systém pomerne presný a dokázal určiť polohu objektu s odchýlkou do 1 cm, čo je pomerne dobré vzhľadom na využitie nižšieho rozlíšenia. Túto odchýlku by bolo teda možné ďalej zlepšiť s použitím výkonnejšieho hardvéru a využívaním mračna s vyšším rozlíšením. Najväčšie problémy nastali pri meraní vzdialenosti v smere k slave Kinectu. Kvôli mierne nesprávnemu zloženiu objektu z obrazov niekedy vznikli problémy s posunom strany bližšie ku Kinectu aj napriek tomu, že všetky kontrolné značky sa zhodovali medzi senzormi. Najväčší problém pri tom nebol v nameraných hodnotách. Tie boli podobné tým získanými pomocou metódy s jedným Kinectom, avšak mali pomerne veľký rozptyl medzi meraniami. Rovnaká poloha objektu dokázala na jednom snímku byť úplne presná a na ďalšom mať rozdiel až do 5 centimetrov. Teda samotným problémom bol rozptyl medzi jednotlivými meraniami.



Obr. 8.2: Scéna s detekovanými objektami (vyznačené modrou a zelenou) v prekryve

Nasledovalo overenie redukcie oklúzie. Na scénu boli preto umiestnené dva objekty, ktoré boli z pohľadu master Kinectu priamo za sebou. Príklad takého usporiadania môžeme vidieť na obrázku 8.2. Počas tohto experimentu bolo zároveň zistené, že minimálna vzdialenosť medzi objektami, predtým než sa spoja do jedného mračna bodov, je na úrovni približne

2 centimetre. Pod touto úrovňou dokázal systém objekty ešte niekedy rozlíšiť, avšak nie spoľahlivo. Išlo o pomerne jednoduchý experiment, v ktorom som postupne posúval objekty bližšie k sebe až sa vo vizualizácii mračna nezobrazili ako jeden objekt. Po každom posune som sledoval mračná po dobu asi 20 sekúnd aby som overil, že riešenie je stabilné. Je dôležité poznamenať, že v tomto experimente boli úmyselne využité objekty s rôznou výškou pre čo najväčšie zakrytie objektu. To však mohlo mať výhodu pri nájdení roviny v inej výške a teda jasnejšie rozdelenie medzi objektami. Obrázok 8.2 zobrazuje práve tieto dva objekty vo vzdialenosti 15 centimetrov.

Pre ďalší experiment som sa preto rozhodol vziať dva objekty rovnakého tvaru a podobným spôsobom ako v predchádzajúcom prípade zistiť ich minimálnu vzdialenosť. Posledná poloha, v ktorej bolo mračno rozdelené bolo vo vzdialenosti približne 3 centimetre. Avšak táto poloha už bola hraničná a zaznamenal som počas pozorovania jedno spojenie.

Aj napriek správnej kalibrácii je takmer nemožné úplne synchronizovať obrazy z oboch kamier. Problémy sú pozorovateľné hlavne na hranách objektov a vo väčšej vzdialenosti od samotného senzoru. Tento jav je viditeľný napríklad pri zobrazení oboch mračien v RViZ. Oblasť, v ktorej sa nachádzali kalibračné značky sa presne prekrýva, avšak pri vyšších objektoch vieme pozorovať skreslenie aj v tejto oblasti. Metóda akou sa získavajú mračná bodov spôsobuje, že aj rovné povrchy majú mierne zvlnenie. To v prípadne jedného zariadenia neprináša veľké problémy. Ak sa však snažíme spojiť dve takéto mračná, ktoré sú orientované kolmo na seba, vzniká veľké množstvo šumu. To ďalej spôsobuje nepresnosti v odhade polohy objektov. V porovnaní s riešením len s jedným Kinectom sme dosiahli mierne zhoršenie v presnosti merania a to hlavne kvôli nekonzistentným výsledkom.

Kapitola 9

Záver

Výsledkom práce je balíček pre Robotický operačný systém, ktorý je schopný synchronizovať dáta z viacerých RGB-D kamier a následne na scéne nájsť objekty, zistiť ich polohu a základný tvar. Ako sa počas písania práce ukázalo, jednou z najzložitejších a zároveň najdôležitejších častí bola správna kalibrácia senzorov. Tá si vyžiadala hľadanie rôznych riešení. Bez správnej kalibrácie systém nemôže fungovať a poskytované výsledky budú nezmyselné. Preto je dôležité dbať na správne nastavenia a pri čo i len menších posunoch kamier opakovať vzájomnú, teda stereo kalibráciu.

Napriek tomu, že počas prvotného testovania sa použitie Raspberry Pi 4B ukazovalo ako dostatočné, avšak pri pokuse nasadiť celý systém sa niekedy ukázali drobné nedostatky vo výkone. Tie spôsobili občas drobné komplikácie s komunikáciou s Kinectom, ktoré boli väčšinou vyriešené pri reštarte operačného systému. Aj keď prezentované riešenie je funkčné, pre prípadné finálne nasadenie by som v budúcnosti odporučil využiť výkonnejšie zariadenie ako je napríklad Intel NUC, či iné výkonnejšie zariadenie pre zvýšenie celkovej stability systému.

Jednou z limitácií riešenia je, že systém nie je schopný zistiť skutočne natočenie objektu, keďže nepozná objekt samotný ani jeho prednú stranu. To je ale zároveň aj výhodou riešenia, pretože prináša určitú flexibilitu pri detekovaní, a teda vie nájsť a určiť polohu aj jemu neznámych objektov. Prípadné rozšírenie tejto práce vidím v pridaní neurónovej siete, ktorá bude rozpoznávať predmety a na základe nej sa bude rameno rozhodovať, čo sa s daným predmetom vykoná. Tento systém by mohol slúžiť napríklad na triedenie objektov a podobne. Zároveň by vedela vyriešiť možné problémy so skutočným natočením objektov.

Ďalším možným vylepšením systému by bolo využitie novších senzorov Kinect Azure, ktoré prinášajú vylepšenia v kvalite obrazu, ale aj podporovaných knižniciach. Nevýhodou je však, že kód medzi knižnicami nie je prenositeľný a nasadenie nového hardvéru by si vyžiadalo pomerne veľké zásahy do implementácie.

Práca bola navrhovaná so statickými prvkami v podobe robotického ramena a kamier, keďže tie sa v priestore nebudú pohybovať. Teda aktuálne riešenie prepočítava vzdialenosť objektu ku základni robotického ramena. Avšak bolo by možné pridať ďalšiu časť kalibrácie, a tak dynamicky počítať polohu objektu k ruke ramena. Tým by výsledkom programu boli priamo kroky, ktoré musí rameno uskutočniť na to, aby sa presunulo ku objektu.

Ako je jasné z implementačnej časti a experimentov, pridanie ďalšieho zariadenia do systému spôsobilo rapídne zvýšenie komplexity celého riešenia. To malo za následok trochu menej spoľahlivé výsledky pri zaznamenávaní polohy objektov. Zároveň prináša do systému ďalší bod zlyhania, ktorým je kalibrácia. Tá je pomerne krehká a každý posun kamier znamená potrebu opakovania kalibrácie. Využitie podobného systému v praxi teda odporú-

čam, len ak je to nevyhnutné. Je potrebné zvážiť, či prípadne zlepšenie pokrytia priestoru je dostatočným faktorom. Ak je tento faktor pre nasadenie kritický, tak takýto systém prináša svoje výhody, ktorými sú práve limitácie prekryvov za cenu zníženia konzistentnosti nameraných hodnôt.

Literatúra

[1] Aalerud, A., Dybedal, J. a Hovland, G. Automatic Calibration of an Industrial RGB-D Camera Network Using Retroreflective Fiducial Markers. *Sensors*. 2019, zv. 19, č. 7. DOI: 10.3390/s19071561. ISSN 1424-8220. Dostupné z: https://www.mdpi.com/1424-8220/19/7/1561.

[2] Brancati, R., Cosenza, C., Niola, V. a Savino, S. Experimental Measurement of Underactuated Robotic Finger Configurations via RGB-D Sensor. In: Aspragathos, N. A., Koustoumpardis, P. N. a Moulianitis, V. C., ed. *Advances in Service and Industrial Robotics*. Springer International Publishing, 2019, s. 531–537. DOI: 10.1007/978-3-030-00232-9_56. ISBN 978-3-030-00232-9.

[3] Clark, R. A., Mentiplay, B. F., Hough, E. a Pua, Y. H. Three-dimensional cameras and skeleton pose tracking for physical function assessment: A review of uses, validity, current developments and Kinect alternatives. *Gait & Posture*. 2019, zv. 68, s. 193–200. DOI: https://doi.org/10.1016/j.gaitpost.2018.11.029. ISSN 0966-6362. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0966636218311913.

[4] Crimmins, R. *Using Xbox Kinect V2 Within Robot Operating System for Outdoors Obstacle Detection*. 2018. Dizertačná práca. Worcester Polytechnic Institute.

[5] Heredia Conde, M. *Phase-Shift-Based Time-of-Flight Imaging Systems*. 1. vyd. Wiesbaden: Springer Fachmedien Wiesbaden, 2017 [cit. 2021-12-20]. ISBN 978-3-658-18057-7. Dostupné z: https://doi.org/10.1007/978-3-658-18057-7_2.

[6] Joseph, L. *ROS Robotics Projects*. 1. vyd. Packt Publishing, 2017 [cit. 2021-12-21]. ISBN 978-1-78355-471-3. Dostupné z: https://books.google.sk/books?id=rLkrDwAAQBAJ.

[7] Li, P., Wang, R., Wang, Y. a Tao, W. Evaluation of the ICP Algorithm in 3D Point Cloud Registration. *IEEE Access*. 2020, zv. 8, s. 68030–68048. DOI: 10.1109/ACCESS.2020.2986470.

[8] Osrf. *Beginner: Overview*. 2014. Dostupné z: https://gazebosim.org/tutorials?cat=guided_b&tut=guided_b1.

[9] Pan, W., Zhu, F., Hao, Y. a Zhang, L. Fast and precise 6D pose estimation of textureless objects using the point cloud and gray image. *Appl. Opt.* OSA. Oct 2018, zv. 57, č. 28, s. 8154–8165. DOI: 10.1364/AO.57.008154. Dostupné z: http://www.osapublishing.org/ao/abstract.cfm?URI=ao-57-28-8154.

[10] Point Cloud Library. *Point Cloud Library* [online]. [cit. 2023-02-10]. Dostupné z: https://pointclouds.org.

[11] Pomerleau, F., Colas, F., Siegwart, R. a Magnenat, S. Comparing ICP variants on real-world data sets. *Autonomous Robots*. Apr 2013, zv. 34, č. 3, s. 133–148. DOI: 10.1007/s10514-013-9327-2. ISSN 1573-7527. Dostupné z: https://doi.org/10.1007/s10514-013-9327-2.

[12] ROS Wiki contributors. *Concepts* [online]. Jun 2014 [cit. 2021-12-21]. Dostupné z: http://wiki.ros.org/ROS/Concepts.

[13] Rusu, R. B. a Cousins, S. 3D is here: Point Cloud Library (PCL). In: Bicchi, A., ed. *2011 IEEE International Conference on Robotics and Automation*. 2011. DOI: 10.1109/ICRA.2011.5980567. ISBN 978-1-61284-385-8.

[14] Soltan, S., Oleinikov, A., Demirci, M. F. a Shintemirov, A. Deep Learning-Based Object Classification and Position Estimation Pipeline for Potential Use in Robotized Pick-and-Place Operations. *Robotics*. 1. vyd. 2020, zv. 9, č. 3. DOI: 10.3390/robotics9030063. ISSN 2218-6581. Dostupné z: https://www.mdpi.com/2218-6581/9/3/63.

[15] Stanford Artificial Intelligence Laboratory et al.. *Robotic Operating System* [online]. 2021 [cit. 2021-10-22]. Dostupné z: https://www.ros.org/.

[16] Thomas, D. *Changes between ROS 1 and ROS 2*. Jun 2017. Dostupné z: http://design.ros2.org/articles/changes.html.

[17] Tölgyessy, M., Dekan, M., Chovanec, L. a Hubinský, P. Evaluation of the Azure Kinect and Its Comparison to Kinect V1 and Kinect V2. *Sensors*. 2021, zv. 21, č. 2. DOI: 10.3390/s21020413. ISSN 1424-8220. Dostupné z: https://www.mdpi.com/1424-8220/21/2/413.

[18] Wen, B. a Bekris, K. BundleTrack: 6D Pose Tracking for Novel Objects without Instance or Category-Level 3D Models. In: IEEE. *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021, s. 8067–8074. DOI: 10.1109/IROS51168.2021.9635991. ISBN 978-1-6654-1714-3.

[19] Zhou, Q.-Y., Park, J. a Koltun, V. Open3D: A Modern Library for 3D Data Processing. *ArXiv:1801.09847*. 2018.