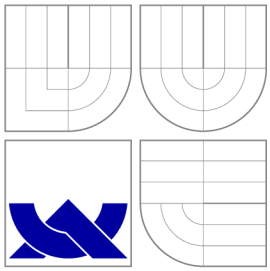
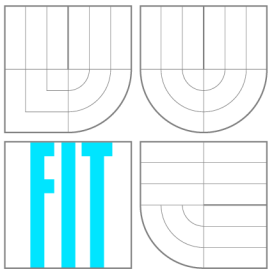


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MODUL PRO LOKALIZACI DVEŘÍ A KLIKY PRO PR2

DOORS AND DOOR HANDLE LOCALIZATION FOR PR2

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

DANIEL BAMBUŠEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MICHAL KAPINUS

BRNO 2016

Zadání bakalářské práce/18349/2015/xbambu04

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Bambušek Daniel**
Obor: Informační technologie
Téma: **Modul pro lokalizaci dveří a kliky pro PR2**
Doors and Door Handle Localization for PR2
Kategorie: Zpracování obrazu

Pokyny:

1. Prostudujte základy zpracování obrazu a problematiku detekce objektů v obraze, případně v hloubkových datech (MS Kinect).
2. Seznamte se s fakultním robotem PR2, platformou ROS (Robotic Operation System) a jejími možnostmi v oblasti zpracování dat ze senzorů a plánování pohybu robota.
3. Vyberte vhodné metody a nástroje a navrhněte modul pro autonomní lokalizaci dveří a kliky pro PR2, který by umožnil následné otevření dveří.
4. Proveďte experimenty, demonstrujte a diskutujte vlastnosti vašeho řešení.
5. Dosažené výsledky zhodnoťte a navrhněte možnosti dalšího vývoje.
6. Vytvořte stručný plakát nebo video prezentující vaši diplomovou práci, její cíle a výsledky.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění prvních dvou bodů zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Kapinus Michal, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Cílem této práce je navrhnout a vytvořit modul pro robotickou platformu PR2, který bude schopen autonomně lokalizovat dveře a kliku a informace o jejich poloze a vlastnostech poskytnout jinému programu, který zajistí jejich otevření. Práce popisuje použité metody detekce, návrh a implementaci zmíněného modulu a na závěr rozebírá experimenty s tímto modulem. Funkčnost metody byla ověřena pomocí testování v prostředí simulátoru.

Abstract

The aim of this thesis is to design and create module for robotic platform PR2, that is able to autonomously locate doors and door handle and provide information about their location and characteristics to another program that ensures their opening. The thesis describes the methods used for detection, design and implementation of this module and finally analyzes performed experiments with this module. The functionality of the method was verified by testing in simulator.

Klíčová slova

Robot, PR2, ROS, Detekce, Detekce dveří, Detekce kliky, PCL, Point Cloud, Kinect

Keywords

Robot, PR2, ROS, Detection, Doors detection, Handle Detection, PCL, Point Cloud, Kinect

Citace

BAMBUŠEK, Daniel. *Modul pro lokalizaci dveří a kliky pro PR2*. Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Kapinus Michal.

Modul pro lokalizaci dveří a kliky pro PR2

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Kapinuse. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Daniel Bambušek
17. května 2016

Poděkování

Tímto bych chtěl poděkovat vedoucímu mé práce, Ing. Michalu Kapinusovi, za jeho ochotu, věnovaný čas a odborné rady, které mě vždy nasměrovaly správným směrem.

Dále bych chtěl poděkovat Danielu Senčuchovi za užitečné rady, které mi poskytl v počátcích vývoje.

Obsah

Kapitola 1

Úvod

Robotika nesporně patří mezi rychle se rozvíjející obory. Stále více robotů se specializuje na interakci s lidmi, prací a pohybem v jejich prostředí. Pro takový pohyb musí být robot schopen samostatné orientace v prostředí, aby se mohl bezproblémově dostat na patřičná místa a splnit zadaný úkol. V uzavřených prostorách pak nepochybně nesmí chybět schopnost robota umět pracovat s dveřmi, čímž se myslí jejich otevření, potažmo zavření. Robot však musí nejprve dveře poznat a najít na nich kliku, aby je mohl následně otevřít. Právě tímto problémem se má práce zabývá.

Detekci dveří, kliky a poskytnutí informací o její poloze, typu dveří, či strany, na kterou se otevírají, zajišťuje tato práce. Na problematiku uchopení kliky, její stisknutí a projetí robota dveřmi se pak soustřeďuje práce Daniela Senčucha [6], jejíž výsledný modul byl tvořen pro spolupráci s mým.

Výstupem práce je modul pro fakultního robota PR2 a na něm běžící Robotický operační systém (dále ROS). V kapitole 2 vysvětlím principy ROSu, popíšu robota PR2 a představím Point Cloud Library – framework používaný pro zpracování point cloudu (tzv. mračna bodů). V kapitole 3 pak blíže specifikuji řešený problém, nastíním možnosti detekce dveří a kliky, rozeberu existující řešení a navrhnu architekturu mého modulu a postup k jeho vývoji. Kapitola 4 se pak bude soustřeďovat hlavně na implementační hledisko této práce a také testování finálního produktu.

Kapitola 2

Teorie

V rámci této kapitoly bude vysvětlen Robotický operační systém a s ním spojené nástroje důležité pro vývoj této práce (sekce 2.1), dále bude představen fakultní robot PR2 (sekce 2.2) a nakonec bude probrán framework PCL a možnosti jeho použití (sekce 2.3).

2.1 Robotický operační systém

Robotický operační systém (ROS) je flexibilní framework pro vývoj robotického softwaru. Jako kolekce nástrojů, knihoven a konvencí si klade za cíl zjednodušit problém vývoje komplexních a robustních chování robotů přes širokou škálu robotických platforem [8].

Systém je volně dostupný, disponuje rozsáhlou komunitou, která vyvíjí nové nástroje, balíčky a knihovny, které zveřejňuje a následně udržuje. Jelikož je stále vyvíjen, jsou postupně nabízeny nové distribuce. V době tvorby této práce fungovala na fakultním robotu PR2 distribuce ROS Hydro Medusa určená pro operační systém Ubuntu 12.04 LTS.

Architektura ROSu spočívá v existenci uzlů (nodes), které spolu komunikují prostřednictvím zpráv a voláním služeb [14].

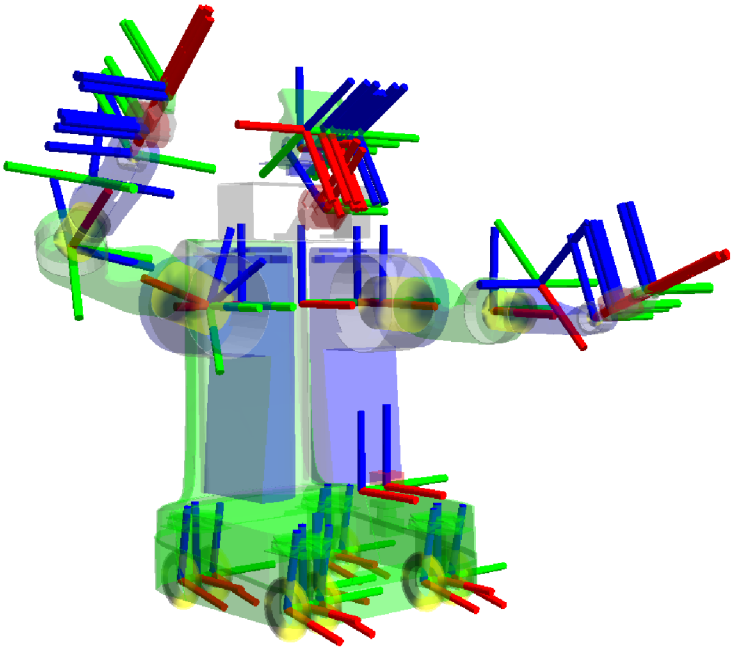
Uzel

Uzly jsou procesy, které provádějí výpočty a akce výsledné aplikace. Aplikace obecně sestává z více uzlů tvořící hierarchickou strukturu, kde uzly mezi sebou jednotlivě komunikují pomocí posílání zpráv nebo voláním služeb. Každý uzel pak zajišťuje specifickou část funkčnosti výsledné aplikace. Například, uzel pro ovládání pohybu základny robota, uzel pro zpracování obrazu z kamery, uzel pro zjištění lokalizace robota atd.

Rozdělení funkčnosti na menší celky nese řadu výhod. Jednou z hlavních výhod je větší odolnost vůči chybám, kdy v případě pádu aplikace jsou chyby izolované v konkrétních uzlech. Takové uzly pak mohou být opětovně spuštěny bez nutnosti restartovat celou aplikaci. Další výhodou je zredukovaná komplexnost zdrojového kódu, dá se tedy vyhnout vytváření velkých monolitických aplikací.

Zprávy a služby

Hlavním prostředkem komunikace mezi uzly jsou zprávy. Zprávy jsou publikovány na určitá témata, z nichž pak mohou další uzly odebírat. Téma funguje jako pojmenovaná sběrnice, skrz kterou se zprávy vyměňují. Komunikace probíhá pouze jednosměrně. Pokud bychom



Obrázek 2.1: Jednotlivé souřadné systémy nacházející se na PR2, jejichž počátky jsou značeny třemi osami: červená pro osu x, zelená pro osu y a modrá pro osu z.

potřebovali obousměrnou komunikaci, je možné využít RPC[1] služeb. Témata mají své přispěvatele (*publisher*) a odběratele (*subscriber*). Díky těmto faktům se dá snadno přenášet například obraz z kamery, kdy uzel zajišťující ovladače kamery publikuje snímaný obraz na své téma, ze kterého pak může daná data odebírat další uzel, pro případné zpracování obrazu.

Dalším způsobem komunikace mezi uzly jsou služby, fungující jako komunikace typu dotaz-odpověď. Pracují na principu RPC. Uzel, který danou službu poskytuje a implementuje, ji tedy provádí pouze na požádání.

O propojení přispěvatelů s odběrateli se stará hlavní uzel (ROS Master). Má přehled o všech běžících uzlech, tématech a jejich názvech. Díky němu je tedy možné, aby se dva uzly našly a mohly spolu komunikovat, ať už pomocí služeb nebo prostřednictvím zpráv.

Bag soubory

Soubory bag, slouží pro ukládání dat z ROS zpráv. Mají koncovku `.bag` a používají se pro záznam surových dat ze senzorů, kterých je poté možné vyvíjet algoritmy bez nutnosti neustálého připojení daného senzoru. Taková data se dají zobrazit pomocí vizualizačního nástroje Rviz. Přehrát je můžeme pomocí nástroje ROSu – *rosbag*.

Rviz

Rviz je vizualizační nástroj, který je schopný prostorově zobrazovat data. Lze si tak zvolit, která témata chceme odebírat a jejich data následně vizualizovat. Uživatel tak může na-

[1] Remote Procedure Call - Vzdálené volání procedur

příklad zobrazovat obraz z kamery, point cloud z Kinectu nebo polohové vlastnosti robota v podobě tf rámců.

Tf

Tf je balíček (knihovna), který pomáhá udržovat přehled o jednotlivých soustavách souřadnic a umožňuje tyto souřadnice mezi sebou libovolně transformovat. Soustavy souřadnic, nazývané rámce, kde osa x směřuje dopředu, osa y doleva a osa z nahoru, jsou hierarchicky uspořádány do stromové struktury. Robot má běžně mnoho takových rámců (viz obrázek 2.1), které se s časem mění. Díky tf je pak možné například zjistit, kde se nacházel rámec hlavy relativně k světu před pěti vteřinami, v jaké pozici je objekt uchopený chapadlem, či jaká je aktuální pozice základny robota vzhledem k rámci mapy[2] [19].

Rámce lze i uměle vytvářet. Stačí zvolit hodnoty souřadnic, jejich rotaci a pak je pod zvolenou transformací[3] publikovat. Využití lze nalézt při detekcích objektů, kde se skrze rámce publikuje poloha nalezeného objektu v reálném světě.

Gazebo

Gazebo je 3D dynamický simulátor schopný simulovat chování robotů v komplexních prostředích. Je řízen fyzikálním enginem, disponuje bohatou knihovnou modelů a prostředí, podporuje širokou variaci senzorů a nabízí pohodlné grafické rozhraní. Používá se tedy pro testování nových algoritmů v robotice a samotný návrh robotů [12].

Je do něj zabudována rovněž podpora ROSu a PR2. Vytváří a napodobuje různá témata ROSu, posílá na ně zprávy, vysílá tf rámce a výstupy ze senzorů. Chová se tak, jak by se choval reálný robot. Z pohledu ROSu a práce s uzly jsou oba roboti totožní.

2.2 PR2

Robot, který je přítomen na naší fakultě a pro kterého je tato práce vyvíjená se jmenuje PR2 (Personal Robot 2). Byl vyvinut firmou Willow Garage. Robot se skládá z všesměrové základny, výsuvného torza, dvou ramen s osmi stupni volnosti a hlavy [13]. Můžete ho vidět na obrázku 2.2.

Základna

Na základně robota můžeme nalézt čtyři kola typu Caster, umožňující pohyb všemi směry. Robot může dosáhnout maximální rychlosti 1 m/s. Z hardwarové výbavy se zde nachází dva procesory Quad-Core i7 Xeon, paměť 24 GB, externí hard disk s kapacitou 1,5 TB a interní hard disk s kapacitou 500 GB. Základna je také osazena lidarem Hokuyo UTM-30LX [13].

Torzo a ramena

Výsuvné torzo robota umístěné na základně lze vysunout z výšky 1330 mm až na 1645 mm – měřeno od podlahy po vrchol hlavy.

K torzu jsou připevněna dvě robotická ramena. Ta se skládají z paže se čtyřmi stupni volnosti, zápěstí se třemi stupni volnosti a chapadla s jedním stupněm volnosti. Rameno je

[2] map frame – nehybná plocha světa, po níž se robot pohybuje

[3] např. `odom_combined` nebo `base_footprint` na PR2

Obrázek 2.2: Robotická platforma PR2

schopno zvednout předmět o váze maximálně do 1,8 kg [13]. Na zápěstí se nachází kamera, chapadlo je pak vybaveno akcelerometrem. Na chapadlo je možné připojit tlakové senzory.

Hlava

Na torzu robota je umístěna hlava, kterou je možné zaklánět a otáčet. Na hlavě je umístěná pěti megapixelová barevná kamera, stereo kamery s širokým a úzkým záběrem a LED projektor textur. Hlava bývá doplněna o Kinect od firmy Microsoft.

2.3 Point Cloud Library

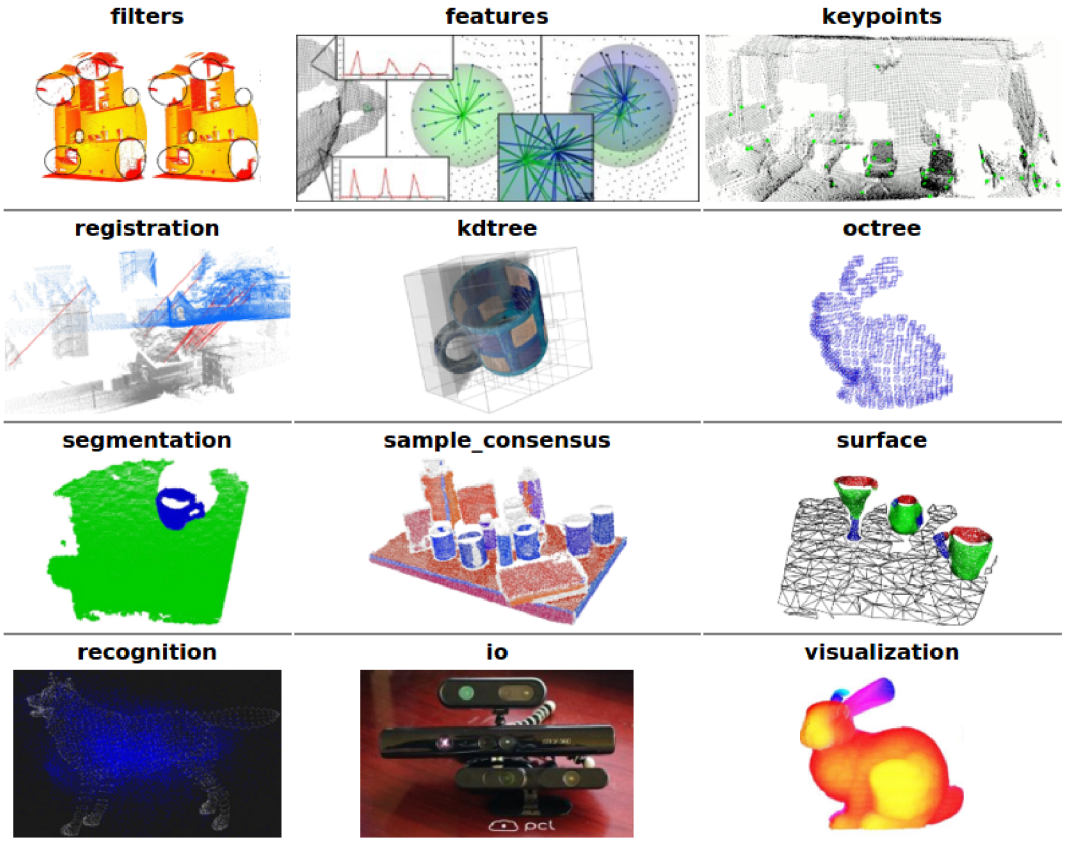
Point Cloud Library (zkráceně PCL) je framework pro 2D/3D zpracování obrazu a point cloudu. Obsahuje řadu algoritmů pro filtrování, rekonstrukci povrchu, detekci objektů, segmentaci atd. PCL je open source software, je multiplatformní, což znamená, že je podporován pro operační systémy Linux, MacOS, Windows a Android/iOS [15]. Pro své široké možnosti zpracování obrazu je hojně podporován v ROSu.

Point Cloud

Point cloud je datová struktura sloužící pro reprezentaci multi-dimenzionálních bodů, běžně ve 3D. Takové body jsou pak reprezentovány x, y a z souřadnicí. 4D cloud pak nese navíc informaci o barvě [15]. Point cloud je tedy složen z mnoha bodů a jako celek má jistou šířku a výšku.

Point cloudy mohou být pořízeny pomocí různých senzorů, jako je například stereokamera, 3D skener (Kinect v1), ToF kamera[4] (Kinect v2), laserový dálkoměr (2D/3D lidar, Velodyne), či mohou být generovány uměle programem.

[4] time-of-flight camera – kamera schopná rozpoznat vzdálenosti



Obrázek 2.3: Přehled nejpoužívanějších modulů PCL a jejich činnosti

Možnosti použití PCL

Framework PCL nalézá využití v mnoha oblastech zpracování obrazu. Framework je rozdělen mezi modulární knihovny, kde každá zajišťuje operace nad specifickou oblastí zpracování obrazu.

Běžnou oblastí bývá například filtrace, kdy může uživatel chtít odstranit šum, či nežádoucí body z obrazu. Tu pokrývá modul `filters`. Modul `registration` poskytuje možnost složení několika pořízených vzorků dat v jeden velký model. Lze si tak vymodelovat například kompletní scan místnosti. Pro potřeby segmentace existuje modul `segmentation`. Segmentace se používá pro shlukování vzorků, které mají nějaké společné vlastnosti. Shluky je možné oddělit od zbytku obrazu a zpracovávat samostatně. Lze tak například rozlišit desku stolu od podlahy, dveře od zdi, za předpokladu, že nejsou v jedné rovině atd. Modul `sample_consensus` implementuje SAC metody[5] a obsahuje různé modely reálných věcí, které se dají v kombinaci s metodami hojně využít pro detekci specifických modelů v point cloudu.

Grafické znázornění popsaných operací můžete vidět na obrázku 2.3. Informace byly čerpány převážně z tutoriálové stránky [16]. Možnosti použití PCL jsou však mnohem větší. PCL je stále udržována a zlepšována mnoha společnostmi po celém světě.

[5] *SAmple Consensus* – patří mezi ně například iterativní metoda RANSAC

Filtrace

PCL nabízí řadu možností jak vstupní data filtrovat. Základním druhem filtrace je použití *PassThrough* filtru. Tento filtr se používá pro ořezání vstupních dat podle osy x, y nebo z. Je tedy nutné stanovit osu a hranice, odkud kam se má ořez provést. PCL poskytuje možnost vyfiltrovat data jak uvnitř zvolených hranic, tak i vně [11].

Nedílnou součástí filtrací je *Downsampling*, česky – podvzorkování. To se používá pro případy, kdy je potřeba zredukovat počet bodů ve vstupních datech [10]. Využití potom nachází v náročnějších metodách, jako například *Plane model segmentation* nebo *Template alignment*, za účelem jejich zrychlení.

Mezi další typy filtrací patří filtrace za účelem odstranění šumu ze vstupních dat. Šumem se myslí takové body, které nenáleží žádnému objektu ze snímané scény. Na základě analýzy jednotlivých bodů a jejich sousedů se odstraní ty body, které nesplňují daná kritéria [18].

Plane model segmentation

Plane model segmentation je postup, který dokáže ve vstupním cloudu detekovat plochu, která má společné vlastnosti – je v jedné rovině. Pomocí funkcí frameworku PCL tak můžeme najít desku dveří, stolu nebo plochu zdi a případně ji oddělit od zbytku cloudu.

Pro dobrou efektivnost metody je třeba vhodně zvolit hodnotu prahové vzdálenosti, která určuje, jak moc blízko musí body být vzhledem k modelu (rovině), aby mohly být považovány za body jemu náležící [17].

Pro takovou detekci modelu postup používá metodu *Random Sample Consensus*, zkráceně RANSAC. Algoritmus náhodně vzorkuje podmnožinu ze vstupních dat za účelem nalezení shody s hledaným modelem. Iterativně tak vybírá náhodné vzorky dat, které by mohly náležet hledanému modelu, dokud není překročeno ukončovací kritérium [20]. Hledaný model může být cokoliv, co lze parametricky popsat, např. rovina, přímka atd..

Template alignment

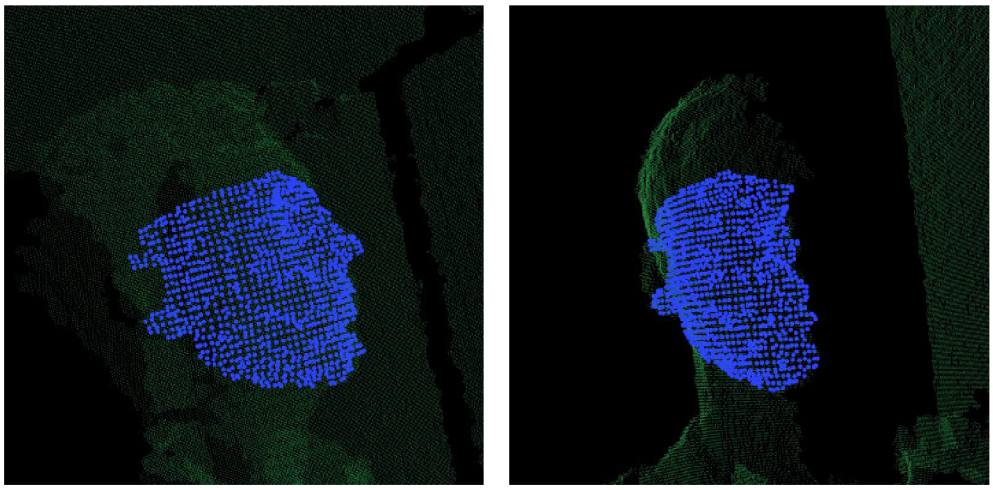
Metoda se používá pro zarovnání dříve pořízených bodů (modelu) na nová data. Na obrázku 2.4 můžete vidět příklad použití metody, kde došlo k zarovnání modelu (modrá část) na nověji pořízený cloud lidského obličeje. Z výsledného zarovnání pak můžeme snadno zjistit pozici a rotaci obličeje ve scéně.

Před aplikováním zarovnání je důležité zjistit povrchové normály a lokální deskriptory jak modelu, tak nově pořízeného vzorku dat. Na základě jejich porovnávání se bude zarovnání provádět. PCL disponuje řadou již implementovaných funkcí pro takové výpočty.

Důležitým výstupem metody je transformační matice, která říká, jak mají být body modelu rotovány a přeneseny, aby byly co možná nejlépe zarovnány se vstupním vzorkem dat. Mimo tuto matici je také důležité „fitness“, což je hodnota, která značí to, jak dobře se povedlo model zarovnat (čím nižší číslo, tím lépe) [9].

Iterative Closest Point

Iterative Closes Point, zkráceně ICP, je algoritmus sloužící pro zarovnání dvou modelů. Zarovnání je založené čistě na jejich geometrii, někdy i barvě. Algoritmus tedy počítá se dvěma modely (vzorky dat) a iterativně se je na sebe pokouší zarovnat a minimalizovat tak rozdíly mezi nimi [7]. Jeden vzorek je zvolen jako výchozí, je fixní. Na druhý vzorek se



Obrázek 2.4: Příklad použití metody *Template alignment*

aplikuje transformace tak, aby se vzorek co nejlépe přiblížil k výchozímu vzorku. S každou iterací se vypočítá nová transformační matice.

Algoritmus tedy očekává dva vzorky dat, které se bude snažit podle zvoleného počtu kroků zarovnat. Jeho výstupem je transformační matice, která stanovuje, jak je nutné daný cloud posunout a otočit, aby co nejlépe seděl na výchozí cloud.

Metoda se často používá pro rekonstrukci 3D scén, protože zařízení snímající komplexní scénu je schopné v jednom okamžiku snímat pouze z jednoho směru [7]. S použitím ICP tak lze nasnímané scény z více úhlů zrekonstruovat do jednoho kompletního modelu. Problém řešící zarovnání více scén v jednu komplexní se nazývá registrace.

Kapitola 3

Návrh řešení

V první části této kapitoly (sekce 3.1 a 3.2) bude představen problém řešený touto prací a nastíněny možnosti detekce, v další části rozeberu existující řešení (sekce 3.3) a na závěr bude popsán návrh architektury vyvíjeného programu a návrh postupu detekce (sekce 3.4 a 3.5)

3.1 Specifikace problému

Cílem této práce je vytvořit uzel ROSu pro fakultního robota PR2, který bude schopen detekovat dveře a jim příslušející kliku. Uzel bude spolupracovat s uzlem pro otevření dveří (dále UpO), který vytvořil Daniel Senčuch [6]. Uzel by měl být schopen detekovat libovolné dveře, u kterých zná jejich rozměry a model kliky. Informace o poloze a druhu dveří a kliky bude publikovat prostřednictvím zpráv ROSu, které pak bude UpO odebírat pro úspěšné otevření.

Činnost uzlu pro detekci (dále UpD) se bude spouštět v případě, že robot narazí na nějakou překážku. Situace se dá tedy popsat tak, že se robot pohybuje určitou předem naplánovanou cestou, jakmile narazí na nějakou překážku a zjistí, že nelze pokračovat dále, začne volat různé uzly zjišťující typ a možnosti řešení překonání překážky. Mezi nimi bude i UpD. Pokud UpD zjistí, že daná překážka jsou dveře, začne s detekcí kliky a publikováním informací o poloze.

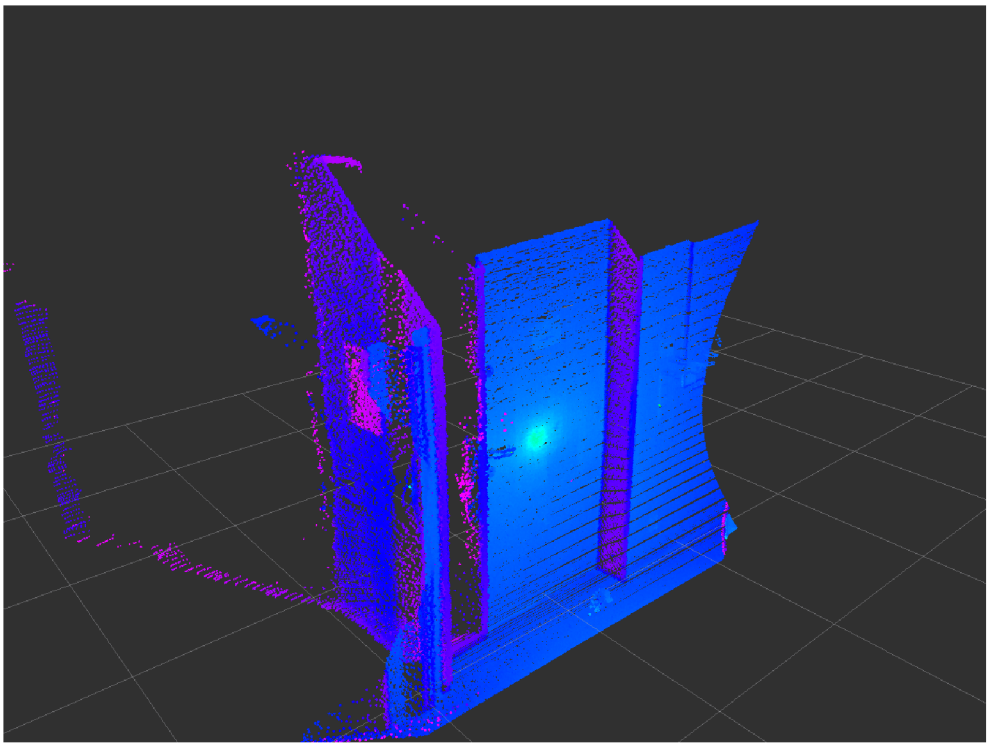
Obrázek 3.1: Kinect a Tilting Hokuyo UTM-30LX Laser Scanner

3.2 Možnosti detekce

Pro detekci je možné využít různých přístupů podle typu použitého senzoru: (1) laserový dálkoměr (LIDAR), (2) kamera, (3) Kinect nebo stereo-kamera, (4) ostatní [21].

Detekce s použitím laseru

V případě robota PR2 lze pro tento typ detekce využít naklápěcí Hokuyo laser skener, umístěný nad rameny robota (viz obrázek 3.1). Skener snímá vše před robotem v podobě laserového paprsku, jehož údaje pak publikuje na tématu `tilt_scan`. Aby se dalo s daty pracovat, je důležité je převést na point cloud a ten následně složit v komplexní scan prostředí. K převodu poskytuje ROS všechny potřebné metody, k složení scanu má pak zabudovány nástroje, které scan načtou do bufferu, s nímž je možno dále pracovat. Takto složený scan vizualizovaný pomocí nástroje Rviz můžete vidět na obrázku 3.2.



Obrázek 3.2: Scan fakultních dveří pořízený laserovým skenerem

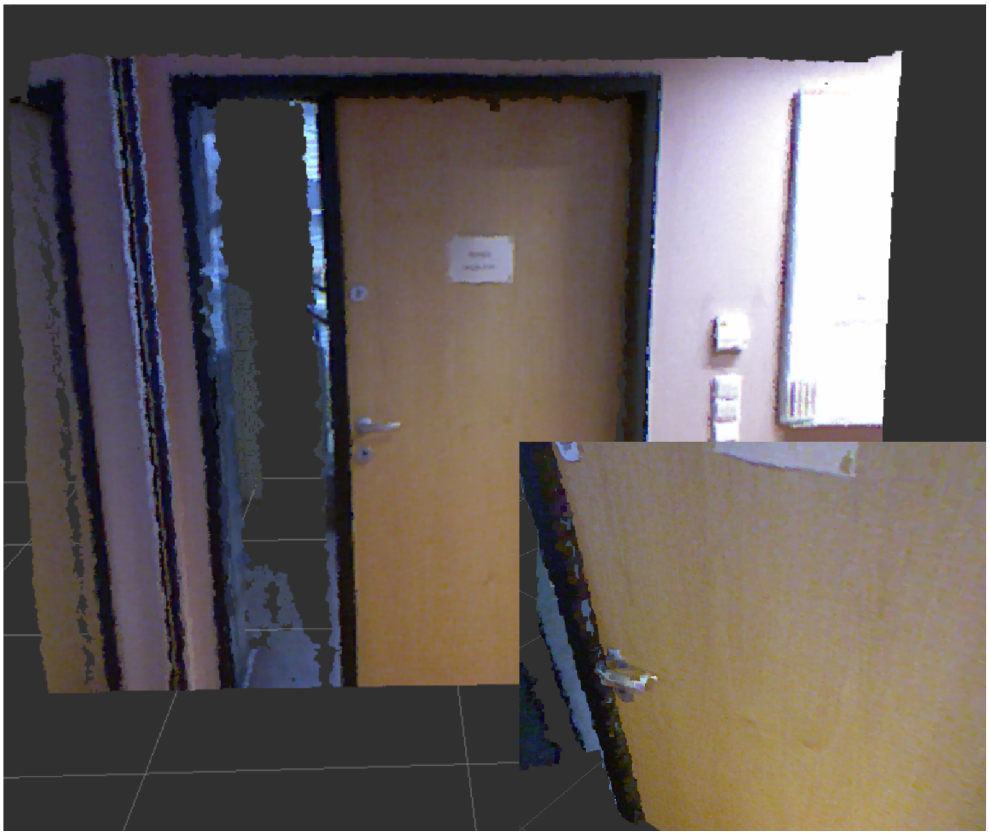
Detekce s použitím Kinectu

Kinect (viz obrázek 3.1) je zařízení od firmy Microsoft původně určené pro ovládání her snímáním pohybu uživatelů konzole Xbox 360. Je složen z infračerveného projektoru, infračervené kamery a RGB kamery. K funkčnosti pak využívá princip techniky structured light[1], podle něhož počítá hloubkovou mapu [2].

Protože je Kinect levný a má dobré vlastnosti (je lehký, není nijak velký, atd.), je hojně využíván na robotech, díky čemuž má dobrou podporu v ROSu. V případě robota PR2 je

[1] proces vysílání známého vzoru pixelů na scénu

umístěn na hlavě. Snímaná data jsou již publikována v point cloud struktuře. Je tak možné rovnou aplikovat segmentační, detekční a různé další metody. Point cloud fakultních dveří pořízený Kinectem je možné vidět na obrázku 3.3.



Obrázek 3.3: Scan fakultních dveří pořízený Kinectem

3.3 Existující řešení

Problém detekce dveří a jejich následného otevření byl již řešen firmou Willow Garage [3], popsaný v subsekci 3.3. Obecným problémem detekce rukojetí se zabývají ve své práci autoři Andreas ten Pas a Robert Platt [1], popsaným v subsekci 3.3.

Řešení detekce rukojetí

Řešení se zabývá lokalizací jakýchkoliv uchopitelných rukojetí v 3D point cloudu. Hlavní myšlenkou je určit sadu geometrických podmínek pro možnost existence bodu úchopu a ve vstupním cloudu pak najít takové body, které podmínkám vyhovují [1].

V práci se nesoustřeďují na lokalizaci objektu jako celku, ale pouze na lokalizaci jeho uchopitelných částí. Algoritmus běží na základě implicitního zarovnávání kvadratických křivek.

Metoda má poměrně dobrou přesnost, běží v reálném čase a lze ji snadno adaptovat na různé roboty a situace. Je zveřejněna formou ROS balíčku, dostupná z GitHub[2].

[2]https://github.com/atenpas/handle_detector

Řešení Willow Garage

Autoři se v jejich práci se soustředili na autonomní pohybování robota PR2 v kancelářském prostředí, schopnosti otevírat dveře a v případě potřeby, autonomního zapojení do elektrické zásuvky za účelem dobití baterie.

Pro detekci dveří využili 3D point cloud pořízený naklápěcím laserovým skenerem. Samotný algoritmus je pak založen na segmentaci a shlukování. Tak je generována množina možných kandidátů, kde jsou eliminovány ty, které nesplňují požadavky ADA[3]. Pro správnou funkčnost algoritmu je důležité, aby nebyly dveře v jedné rovině se zdí.

Detekce kliky pak využívá paralelně kombinaci dvou odlišných přístupů. Jednak metoda využívající k zpracování obraz z kamery a metoda využívající laserový skener. Obě metody se pak musejí shodnout na lokaci kliky. V metodě využívající laserový skener je segmentace bodů kliky z point cloudu založena na shlukování, které využívá rozdílů intenzit mezi klikou a deskou dveří a na přístupu, který bere v úvahu rozdíly v zakřivení dveří v oblasti kliky. Detailněji popsáno v [5]. Druhá metoda naopak využívá *object classifier cascade* od autorů P. Viola a M. Jones [4].

Willow Garage poskytlo zdrojové kódy volně komunitě ROSu. Avšak bylo uzpůsobeno pro starší verzi ROSu a z neznámých důvodů již odstraněno.

3.4 Návrh architektury uzlu

Uzel pro detekci bude moci pracovat nezávisle, bude tedy oddělen od UpO Daniela Senčucha. Ke správnému propojení těchto dvou uzlů bude potřeba dodržet společné rozhraní – strukturu zpráv ROSu, které si uzly budou mezi sebou vyměňovat. Díky dodržení tohoto rozhraní bude zajištěna modularita, kde UpD nebo UpO bude moci být přepracován a funkčnost zůstane stále zachována.

UpD bude posílat UpO informace o dveřích (rozměry, stav atd.) a tf rámce (osy dveří, kliky atd.) pro určení přesné polohy dveří a kliky. Přesný popis zpráv je v sekci 4.5. UpO pak bude posílat zpětnou vazbu o průběhu otevírání dveří.

UpD bude fungovat jako uzel poskytující službu, což znamená, že bude publikovat služby s možností spuštění a zastavení detekce. Jeho činnost tak bude spouštěna na zavolání.

3.5 Návrh postupu detekce dveří

Uzel bude předpokládat, že robot již před nějakou překážkou stojí a dívá se přímo na ni. Jeho činnost se dá rozdělit na dva podproblémy - detekovat dveře (zjistit rozměry, stav, tf rámce) a následně najít kliku. Pokud dveře nedetekuje, nebude se ani zahajovat detekce kliky. Pro detekci bude využita varianta s použitím Kinectu popsaná v sekci 3.2.

Detekce dveří

Jelikož zorné pole Kinectu nebude robotu umožňovat náhled na celé dveře, za předpokladu, že bude stát přímo před nimi na vzdálenost zhruba $1m$, bude nutné hýbat hlavou robota nahoru a dolů, aby mohla být detekována jak šířka, tak výška dveří. Ty se poté porovnají s referenčními rozměry dveří, které budou uloženy v databázi. Při shodě se bude detekovaná

[3] Americans with Disabilities Act (američané s postižením) - rozsáhlý občanský zákoník, který je určen k ochraně proti diskriminaci na základě zdravotního postižení

deska považovat za dveře, čímž je možno pokračovat v hledání kliky. V opačném případě se činnost uzlu ukončí.

Detekce kliky

Před začátkem hledání kliky by se měl robot dívat na bod detekovaných dveří vysoký $1m$ od země, aby Kinect pokryl celé zorné pole s nejvyšší pravděpodobností výskytu kliky. Pro kliku bude dopředu vytvořen model z dříve pořízeného point cloudu Kinectu, který bude uložen společně s rozměry dveří v databázi. Ten si pak uzel načte a pomocí detekčních metod jej bude hledat na aktuálně dostupných datech.

3.6 Návrh postupu řešení

Pro začátek práce bude nejlepší pořídit záznamy formou bag souborů, do nichž se nahraje snímaný point cloud dveří a tf rámce vysílané robotem. Bag soubory se pak dají přehrávat a simulovat tím reálnou scénu, kdy se robot dívá na dveře. Zmíněným postupem se však nedají aplikovat autonomní pohyby hlavy za účelem zjištění výšky dveří. To se však dá pro počáteční verzi uzlu zanedbat a lze tedy předpokládat, že po zjištění patřičné šířky jsou dveře detekovány.

Jakmile bude takový uzel implementován, lze ho vyzkoušet na reálném robotovi. Jelikož aplikace zatím nijak nepohybuje robotem, nemělo by zde být ani žádné riziko poškození robota nebo jeho okolí.

V další fázi vývoje lze zahrnout i zmíněné pohyby hlavou pro zjištění výšky dveří. K tomu už bag soubory stačit nebudou. Do úlohy tedy nastupuje simulátor Gazebo, kde stačí spustit simulovaného robota a vložit model dveří. Gazebo je schopné plně simulovat reálného robota, proto budou výstupy ze senzorů stejné jako z pořízených bag záznamů reálného PR2.

Po zapojení detekce výšky dveří a odzkoušení v Gazebu lze jít opět testovat na reálném robotu. Ve finální fázi vývoje pak bude potřeba zjistit požadované tf rámce a ty pak vysílat společně s informacemi o dveřích pro UpO. Pokud bude poloha rámců kliky a dveří odpovídat realitě, lze pokračovat napojením UpD na UpO a otestovat, zda bude robot schopen na základě získaných informací otevřít a projet dveřmi.

Kapitola 4

Implementace a testování

Při realizaci jsem využíval možností ROSu a PR2, kdy jsem mohl aplikaci vyvíjet z domu za pomocí bag souborů a simulačního nástroje Gazebo. Díky tomu jsem nemusel být tolik vázaný na robota.

Pro implementaci jsem využíval již existujících kódů, zejména z tutoriálů PCL. Samotná aplikace je pak realizována jako jeden uzel ROSu, který přijímá data z Kinectu, ty zpracovává a výsledky publikuje tf rámci a zprávami ROSu. Pro překlad jsem použil prostředí `catkin`.

V počátcích vývoje, kdy nebyla nutnost, aby robot hýbal hlavou, jsem testoval aplikaci na reálném PR2 a jeho výstupu z Kinectu v robotické laboratoři. S přibývajícími fázemi vývoje, po zapojení pohybů hlavy robota, jsem testoval hlavně v simulátoru Gazebo. Na obrázku 4.1 můžete vidět náhled na robota v simulátoru Gazebo a dveře, na nichž jsem testoval. K finálnímu testování na reálném PR2 nedošlo, neboť s ním byly v té době technické problémy.

V této kapitole bude popsána implementace uzlu, potřebná vstupní data, celkový postup detekce a výstupní data publikovaná uzlem UpD. Na závěr se budu věnovat testovaní a problémům, které jsem musel při vývoji řešit.

4.1 Vstupní data

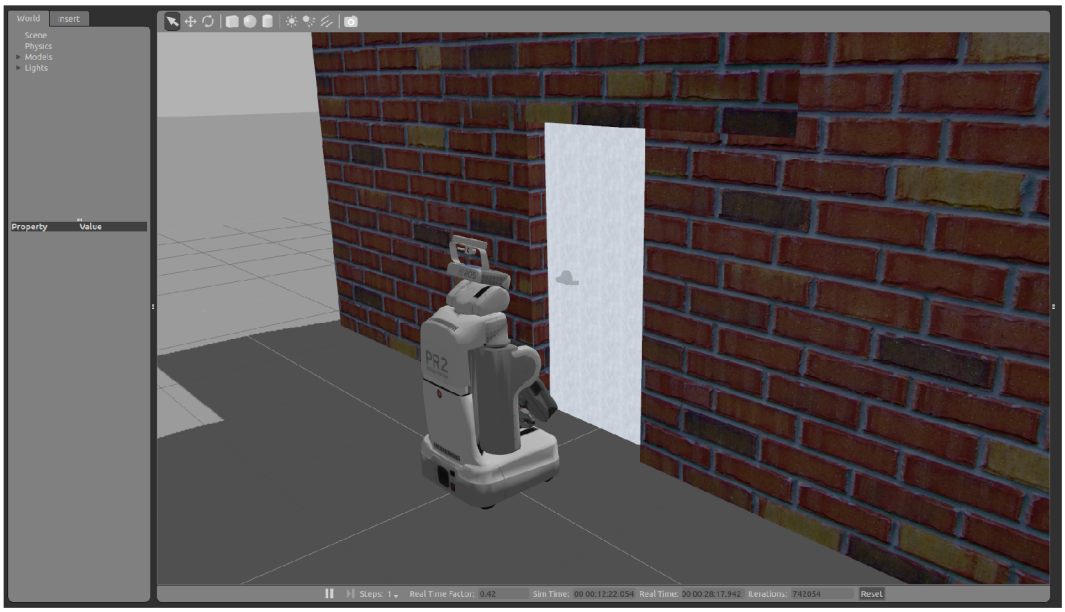
Uzel pracuje se třemi druhy vstupních dat. Ke své činnosti potřebuje data z Kinectu, dále JSON soubor se specifikací jednotlivých dveří, včetně názvů modelů klik a v neposlední řadě samotné modely klik.

Data z Kinectu

Aplikace pracuje se dvěma vstupními daty. Jedná se o výstup z Kinectu – PointCloud2 a vstupní JSON soubor s databází známých dveří, o němž je více popsáno níže. Kinect vysílá na dané téma, z něhož UpD získává nová data. Tato data se zpracovávají pomocí callback funkce, která je hlavní řídící konstrukcí celé aplikace. Vstup je nutné převést na datovou strukturu PointCloud, se kterým umí pracovat knihovny PCL.

Vstupní JSON soubor

Soubor obsahuje specifikace dveří, které je aplikace schopna detekovat. Jednotlivé záznamy obsahují informaci o šířce a výšce daných dveří a názvy souborů s jednotlivými modely



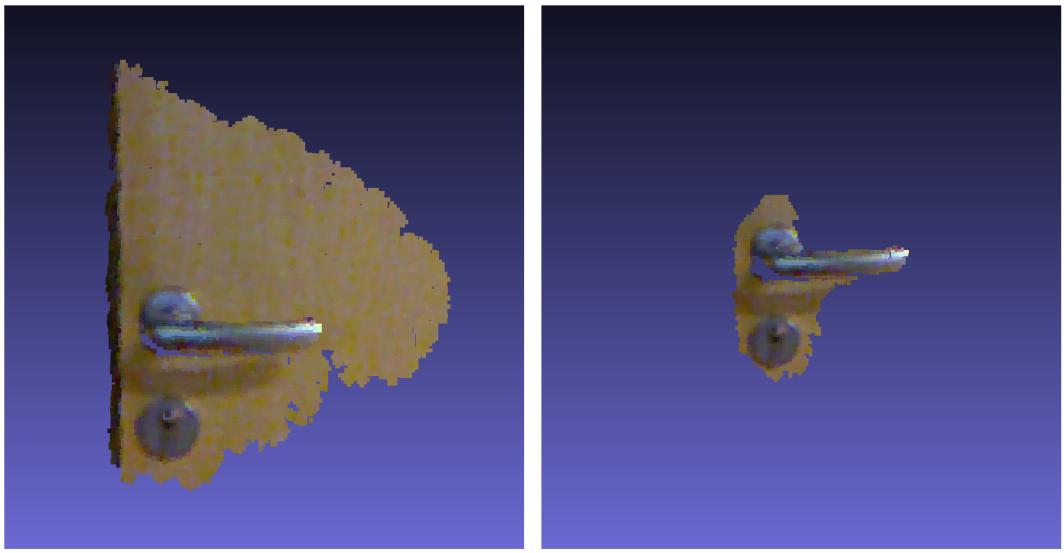
Obrázek 4.1: PR2 po spuštění aplikace v simulátoru Gazebo

klik korespondujících k daným dveřím. Mimo modely je potřeba uchovávat i údaj o šířce kliky, který se bude používat pro větší spolehlivost detekce. Veškeré rozměry jsou uváděné v metrech.

Ukázka kódu 4.1: Ukázka JSON souboru se záznamem testovaných dveří v laboratoři

```
{
  "VUT-FIT-cvt_doors":{
    "width": 0.90,
    "height": 2.10,
    "handle_models": [
      {
        "handle_vutfit1.pcd": 0.135
      },
      {
        "handle_vutfit2.pcd": 0.135
      }
    ],
    "handle_rot_angle":0.536
  }
}
```

Původně jsem měl v plánu kategorizovat dveře podle jejich rozměrů a modelu kliky. Od toho jsem však upustil, protože by se pro rozměrově stejné dveře s rozdílnou klikou, prováděla zbytečně vícekrát detekce dveří samotných. Kategorizoval jsem tak dveře pouze podle jejich rozměrů, přičemž modely klik jsou vloženy do pole, ze kterého se pak zvolí nejvhodnější z nich.



Obrázek 4.2: Testované modely klik fakultních dveří

Soubor jsem vytvořil proto, aby bylo možné aplikaci libovolně rozšiřovat o další typy dveří, bez nutnosti zasahovat do kódu. Pro zpracování JSON formátu jsem použil nástroj *RapidJSON* dostupný na GitHub[1].

Modely klik

Modely klik jsem vytvářel z pořízených bag záznamů. V nástroji MeshLab[2] jsem pak pořízený cloud ořezal pouze na kliku, což jsem použil jako výsledný model.

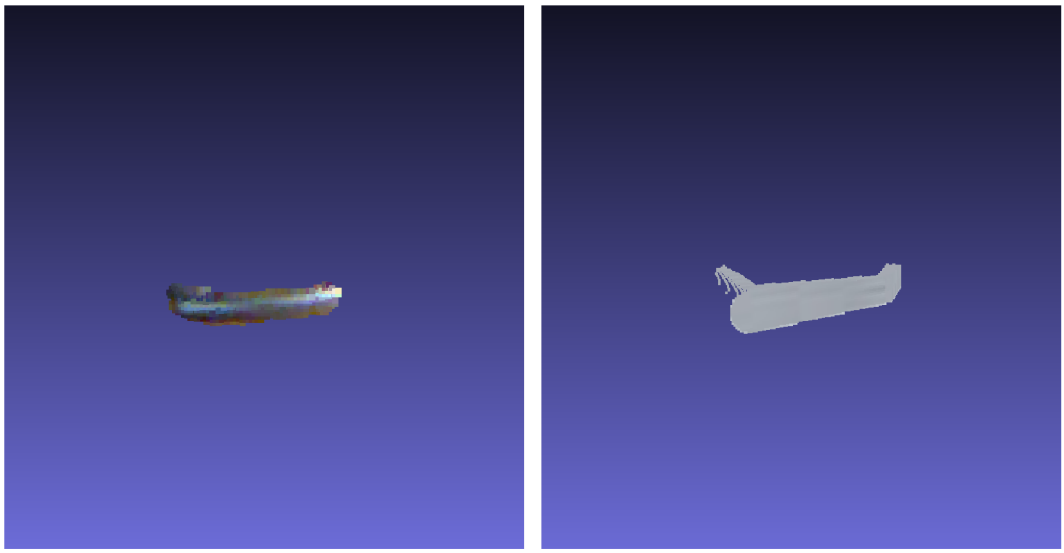
Zkoušel jsem vytvořit a používat více variací modelů té samé kliky. Vytvořil jsem tak model samotné kliky, model kliky s kouskem dveří a klíčovou dírkou a model kliky s větším kusem dveří. Ve výsledné fázi vývoje se jako nejspolehlivější ukázal model samotné kliky, protože použitá metoda pro detekci kliky *Template alignment*, popsaná v 2.3, v mnoha případech zarovnávala zbylé dva modely zcela nepřesně.

4.2 Průběh programu

Uzel po spuštění načte vstupní data o známých typech dveří z JSON souboru, inicializuje ovládání hlavy pomocí třídy RobotHead a publikuje službu `/detection_start`, po jejímž zavolání dojde k zahájení činnosti jádra aplikace, tedy detekce. Služba může být zavolána z jakéhokoliv uzlu, pro testování jsem používal volání nástrojem `rosservice`. Kromě toho publikuje také službu `/detection_end`, která detekci ukončuje.

Samotný algoritmus detekce začíná převedením vstupních dat z Kinectu na strukturu PointCloud frameworku PCL. Ten je pak transformován do souřadného systému nejspodnějšího bodu základny robota, a to `base_footprint`. Díky transformaci do tohoto souřadného systému budou jednotlivé osy odpovídat realitě – tzn. osa z bude kolmá na podlahu, osy x a y budou paralelní s podlahou s tím, že hodnota osa x bude vzrůstat směrem dopředu od

[1] https://github.com/miloyip/rapidjson
[2] http://meshlab.sourceforge.net/



Obrázek 4.3: Finální model kliky fakultních dveří a dveří v Gazebu

robota a osa y poroste od robota směrem doleva. Poté se takto transformovaný cloud posílá do funkce zajišťující detekci samotných dveří, detailněji popsané v podsekci 4.3. Detekce může selhat pokud nejsou k dispozici informace o dveřích ve vstupním souboru nebo pokud robot před žádnými dveřmi nestojí. Pokud však robot před dveřmi opravdu stojí a nepovede se mu je najít, je nutné uzel restartovat a celý proces tak opakovat.

Pokud budou dveře úspěšně detekovány, spustí se hledání kliky na již upraveném cloudu. Klika se bude hledat za pomoci načteného modelu metodou *Template alignment*, více viz. 4.4. Jakmile se povede kliku nalézt, uzel začne vysílat tf rámce s polohou kliky a dveří. Dále vysílá zprávu na téma `open_doors/doors` s informacemi o stavu a rozměrech dveří, více v podsekci 4.5.

Rámce a zprávy publikuje do doby, než bude UpD vypnut nebo zastaven pomocí služby `/detection_end`. Průběh programu znázorněného v programu Rviz můžete vidět na obrázku 4.4.

4.3 Detekce dveří

Na vstupní cloud jsem aplikoval podvzorkování filtrovacím objektem, aby se pracovalo s menším množstvím bodů a ušetřil se tak čas následujících výpočtů. Potom jsem použil metodu *Plane Segmentation*, popsanou v sekci 2.3, za účelem nalezení desky dveří. Na detekovanou desku jsem ještě aplikoval metodu *Euclidean Cluster Extraction*, kterou zajišťuje funkce `removeNoise()`. Učinil jsem tak proto, že ne vždy dokázala samotná segmentace dokonale oddělit desku od zbytku cloudu. Euklidovskou extrakcí jsem tak odstranil nežádoucí body, které nepatřily k desce dveří.

V cyklu se aplikují tyto metody, dokud ve vstupním cloudu pořád zbývají nějaké body. V průběhu tohoto cyklu se vždy kontroluje šířka aktuálně nalezené desky, která se porovnává s načtenými rozměry vstupního JSON souboru. Pokud šířka sedí na nějaký typ známých dveří, je pravděpodobné, že robot detekoval dveře. Pro zjištěné rozměry z cloudu jsem zvolil toleranci $5cm$.

Po zjištění a ověření správnosti detekované šířky je třeba zjistit také výšku, aby šlo jednoznačně říct, že se jedná o dveře. K tomu jsem využil třídy RobotHead a její funkce `lookAt()`, která zajistí, že se robot podívá na zadaný bod v souřadnicích xyz. Pro bod x volím vzdálenost robota od dveří, bod y zůstává 0, neboť předpokládám, že robot už stojí uprostřed dveří. Mění se tedy pouze bod z, který se vždy nastaví na maximální výšku aktuálně detekované desky.

Robot se tak začne dívat směrem nahoru, čímž bude hledat maximum výšky dveří. Je tak nutné nechat načíst do callback funkce nová data z Kinectu a znovu provést podvzorkování a segmentaci desky, aby se ověřilo, že se pořád jedná o ty samé dveře.

Až najde maximální výšku desky, začne obdobným způsobem hledat minimum. Do z souřadnice funkce `lookAt()` se bude tedy vkládat minimum výšky aktuálně detekované desky.

Pokud se ji povede úspěšně najít, odečtením minima od maxima se získá výsledná výška. Ta se pak porovná společně s detekovanou šířkou s rozměry z načteného JSON souboru. Pokud nalezený typ dveří existuje, víme dál jaké modely klik použít a lze tedy prohlásit první část detekce za úspěšnou.

4.4 Detekce kliky

Do funkce `detect_handle()` pro detekci kliky vstupuje upravený cloud, který již prošel úsekem pro detekci samotných dveří. Takto upravený cloud je zbaven detekované desky dveří. Zbývá tedy vše, co se nacházelo před a po stranách dveří. K odstranění bodů, které se nacházejí po stranách jsem použil *PassThrough filter*, popsaný v kapitole 2.3, kterému jsem jako vstupní parametry vložil hodnoty krajních bodů dveří tvořících jejich šířku a ořezal tak cloud v ose y, čili podélně. Z původního cloudu tak zbyly body, které jsou výhradně před deskou dveří. V ideálních případech to budou pouze body náležící samotné klice.

Nyní už robot ví, jaké dveře detekuje, proto si načte odpovídající modely klik do třídy `FeatureCloud`. Ta pro každý model spočítá povrchové normály a deskriptory, které jsou pro použitou metodu *Template alignment*, více v podsekci 2.3, nezbytné. Pomocí třídy `TemplateAlignment` se metoda pokusí každý model zarovnat na předzpracovaný cloud. Model s nejlepším *fitness* skórem je vybrán jako kandidát na hledanou kliku.

Jelikož se *Template alignment* ukázal v mnohých případech jako pouhý „nástřel“ toho, kde by klika mohla být, bylo ještě potřeba použít *Iterative Closest Point* algoritmu, popsaného v podsekci 2.3, pro úplné dorovnání.

Poslední kontrolou toho, že se opravdu jedná o kliku, je ověření její délky. Jak bylo výše zmíněno, mimo názvy modelů klik se ve vstupním JSON souboru uchovává také jejich délka. Ta se nyní porovná s délkou modelu kliky, který už je po výše popsaných metodách v nějakém zarovnání a rotaci v prostoru. Pokud rozměry s rozumnou tolerancí nebudou sedět, znamená to, že klika byla detekována v neočekávaném natočení, čímž lze prohlásit, že se kliku nalézt nepodařilo.

Pokud získaná hodnota *fitness* skóre z obou provedených metod prošla empiricky zjištěným limitem přijatelnosti a pokud délka kliky koresponduje k jejímu modelu, je vysoce pravděpodobné, že byla klika úspěšně nalezena.

V případě úspěšného nalezení kliky jsou zjištěny souřadnice osy, podle které se klika otáčí a bodu úchopu kliky. Pokud byla klika nalezena na levé straně dveří, bude souřadnici osy tvořit nejlevější bod kliky. Pokud byla nalezena na pravé straně, bude osa v nejpravějším bodě. Výpočet bodu úchopu jsem stanovil jako posunutí bodu osy kliky v nutném směru, podle polohy kliky, buď doprava nebo doleva, o dvě třetiny celkové délky kliky. Takto

zjištěné souřadnice bodů tvoří základ pro vysílané tf rámce pro uzel UpO. K těmto bodům je třeba ještě doplnit rotaci kliky vůči robotu.

4.5 Výstupní data

Výstupem uzlu je dvojí druh dat – tf rámce a ROS zpráva. Oba výstupy jsou důležité pro UpO, neboť díky nim zjistí informace o dveřích a jejich poloze.

Vysílané tf rámce

Rámce slouží jako jedna z hlavních komunikací mezi uzly UpD a UpO. Uzel UpD přes ně vysílá polohové informace o dveřích a klice. Jsou vysílány následující tf rámce:

- `doors` – značí nejspodnější část osy otáčení dveří.
- `handle_axis` – značí osu otáčení kliky dveří.
- `handle_grip` – značí místo na klice, kde má robot kliku chytit.

ROS zpráva

Dalším použitým způsobem komunikace mezi uzly UpD a UpO je komunikace prostřednictvím zpráv ROSu. Definice zpráv je popsána jazykem pro popis zpráv[3] a nachází se ve složce `msg` uvnitř balíčku UpD.

Prvním souborem s popisem zprávy je `Doors.msg`. Zprávu, která je v něm popsaná zasílá UpD na téma `open_doors/doors`. Zaslání zprávy pro oba uzly znamená:

- UpD má o dveřích všechny potřebné informace
- UpD začal úspěšně vysílat tf rámce
- UpO může začít otevírat dveře

Samotná zpráva obsahuje také pojmenované konstanty, zde psané velkými písmeny. Její struktura je následující:

- `float32 width`: Šířka dveří v metrech.
- `float32 height`: Výška dveří v metrech.
- `int32 latch_state`: Stav dveří nabývající hodnot `LOCKED` (zamčené), `LATCHED` (zavřené) nebo `UNLATCHED` (otevřené). UpD zatím předpokládá, že jsou dveře vždy zavřené, proto vždy položka nabývá `LATCHED` hodnoty. Ostatní volby jsou zde jako ambice do budoucna.
- `float32 handle_rot_angle`: Úhel v radiánech, o který robot musí otočit klikou po směru hodinových ručiček, aby dveře otevřel, pokud je klika nalevo. Pokud je klika napravo, bude se otáčet proti směru hodinových ručiček. UpD hodnotu úhlu zjistí ze vstupního JSON souboru podle detekovaného typu dveří.
- `int32 rot_dir`: Směr, kterým se dveře mají otáčet. Nabývá hodnot `CLOCKWISE` nebo `COUNTERCLOCKWISE`. Opět ambice do budoucna.

[3] http://wiki.ros.org/msg

- `int32 push_pull`: Značí, zda má robot na dveře zatlačit nebo zatáhnout. Nabývá hodnot `PUSH` nebo `PULL`. Opět ambice do budoucna.

		Aritmet. průměr	**Rozptyl**	**Směrodatná odchylka**	**Očekávaná hodnota**
Rozměry dveří	šířka	1.05971	$9{,}521 \times 10^{-7}$	$9{,}758 \times 10^{-4}$	1.060
	výška	2.11100	$3{,}030 \times 10^{-7}$	$5{,}505 \times 10^{-4}$	2.100
Poloha osy dveří	x	0.94410	$3{,}358 \times 10^{-7}$	$5{,}795 \times 10^{-4}$	0.940
	y	-0.52079	1×10^{-10}	$7{,}215 \times 10^{-6}$	-0.520
	z	0.00336	$3{,}034 \times 10^{-7}$	$5{,}508 \times 10^{-4}$	0.003
Poloha osy kliky	x	0.87515	$1{,}448 \times 10^{-5}$	$3{,}805 \times 10^{-3}$	0.875
	y	0.35385	$2{,}049 \times 10^{-5}$	$4{,}527 \times 10^{-3}$	0.350
	z	1.06048	$7{,}84 \times 10^{-8}$	$2{,}799 \times 10^{-4}$	1.060
Poloha bodu úchopu	x	0.87515	$1{,}448 \times 10^{-5}$	$3{,}805 \times 10^{-3}$	0.875
	y	0.22945	$1{,}271 \times 10^{-5}$	$3{,}564 \times 10^{-3}$	0.230
	z	1.06048	$7{,}84 \times 10^{-8}$	$2{,}799 \times 10^{-4}$	1.060
Doba detekce		27.34773	5.39673	2.32309	—

Tabulka 4.1: Tabulka se souhrnem statistik k deseti provedeným testům

4.6 Testování

Ke konci vývoje aplikace došlo bohužel k problémům s funkčností fakultního robota PR2. Nemohl jsem tak na něm výslednou aplikaci řádně otestovat.

Stihl jsem však průběžně testovat během vývoje, kdy robot ještě fungoval. Aplikace měla v té době ještě omezenou funkčnost, kde nebylo implementováno zjišťování výšky dveří, s čímž souvisí naklápění robotovy hlavy a nebyla zavedena kontrola délky kliky po provedení *Template alignment* metody. Takže detekce kliky zatím neměla tak vysokou spolehlivost.

Základní kostra programu však fungovala a kliku se až na pár výjimek detekovat podařilo úspěšně. Díky těmto pár výjimkám jsem mimo zmíněnou kontrolu délky kliky přidal také její dorovnání *ICP* algoritmem.

Finální podobu aplikace jsem testoval pouze v simulátoru Gazebo. V něm jsem převzal spouštěcí soubor Gazeba od Daniela Senčucha, který spustí PR2 a umístí před něj dveře.

Byl problém s rameny robota, které jsou ve výchozím nastavení předpažené a brání tak robotu v hledání spodní hrany dveří. Obecně se nelze spoléhat na to, že bude mít robot vždy čistý výhled před sebe a nebudou mu v něm bránit vlastní paže, proto je jako první krok nutné zavolat uzel `but_pr2_tuck_arm`[4], který paže složí k jeho tělu.

[4] dostupný z https://github.com/robofit/but_pr2

Robot byl pak schopen bez problému detekovat dveře, najít na nich kliku a publikovat informace. Správné polohy tf rámců jsem ověřoval pomocí ROS programu Rviz.

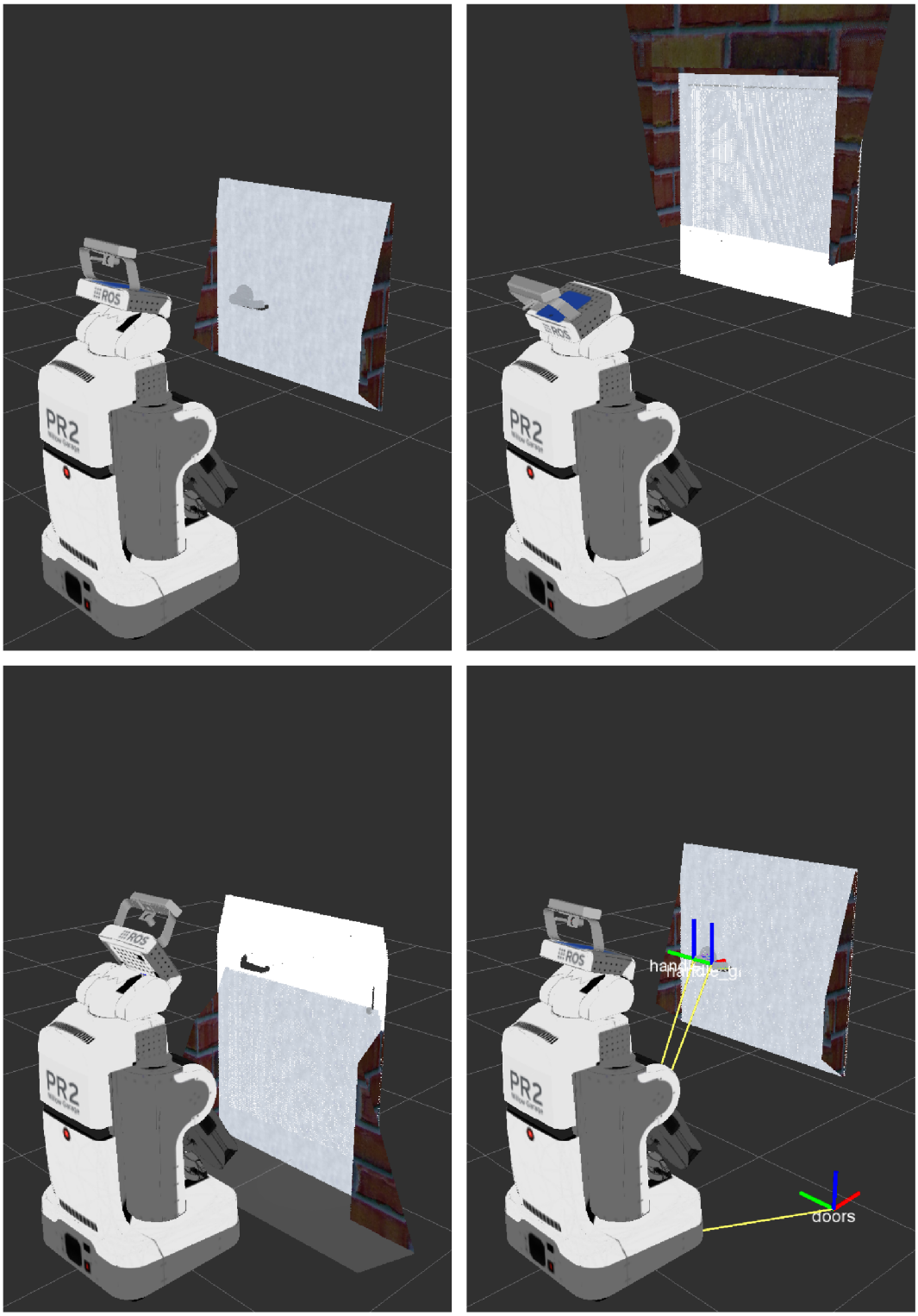
Provedené testy a jejich výsledky

Testy byly prováděné v simulátoru Gazebo, vždy za stejných podmínek, tzn. stejné dveře a stejná výchozí pozice robota. Pozice byla nastavena tak, aby robot stál přímo před dveřmi, tedy svíral s nimi pravý úhel a byl od nich vzdálen cca $0.94m$. Samozřejmostí byl předpoklad, že robotu nebrání ruce ve výhledu.

Jeden samotný test pak spočíval ve spuštění UpD, zavolání služby `/detection_start` a ukončení UpD po úspěšném či neúspěšném pokusu o detekci kliky. Bylo provedeno celkem deset takových testů.

Vždy byly měřeny rozměry detekovaných dveří, polohy jednotlivých rámců, které UpD vysílá a také reálný čas, jak dlouho nalezení kliky trvalo (doba od spuštění detekce po publikování prvního rámce a ROS zprávy).

Z výsledků všech deseti testů byl spočten aritmetický průměr, rozptyl a směrodatná odchylka. Výsledky můžete vidět v tabulce 4.1. Tabulka také obsahuje očekávané hodnoty k jednotlivým měřeným položkám. Hodnoty k rozměrům a polohám jsou uváděny v metrech, doba detekce pak v sekundách. Žádná z dílčích hodnot v rámci jednoho testu nijak výrazně nevybočovala.



Obrázek 4.4: Průběh detekce; znázorněno v programu Rviz

Kapitola 5

Závěr

Cílem této práce bylo vytvořit uzel pro framework ROS a fakultního robota PR2, který bude schopen detekovat dveře, na nich najít kliku a poskytnout informace o její poloze dalšímu uzlu, který zajistí její otevření [6].

Tento cíl byl splněn, postup ke zdárnému řešení byl v tomto dokumentu dostatečně vysvětlen. Přístupem, který jsem použil, by měl robot být schopen detekovat a najít kliku na jakýchkoliv dveřích, ke kterým má jejich rozměry a model kliky.

Výslednou aplikaci jsem bohužel nemohl otestovat na fakultním PR2, protože s ním byly v době tvorby této práce technické problémy. Testoval jsem tak pouze v simulátoru Gazebo, kde se detekce kliky téměř vždy úspěšně zdařila. Bohužel se mi nepodařilo můj uzel propojit s uzlem pro otevření dveří a v simulaci tak dveře otevřít a projet jimi.

Ačkoliv byla práce vyvíjena a testována na PR2, není na něm zcela závislá. S jistými modifikacemi, například vypuštěním zjišťování výšky a s tím související hýbání hlavou, je možné práci použít na jakéhokoliv jiného robota, který má senzor co poskytuje point cloud.

Při vývoji jsem zanedbával různá výchozí natočení robota vzhledem ke dveřím, předpokládal jsem, že robot k nim bude stát vždy čelem a bude od nich vzdálen zhruba jeden metr. Testoval jsem detekci kliky převážně na dveřích, které se otevírají směrem od sebe a mají kliku nalevo. Obohacení aplikace o schopnost robota lépe se vypořádat s nestandardním výchozím postavením vůči dveřím a větší spolehlivost při detekci jiných variací dveří by mohlo být předmětem dalšího vývoje nebo mé diplomové práce.

Literatura

[1] Andreas ten Pas, Robert Platt: Localizing Handle-Like Grasp Affordances in 3D Point Clouds. *International Symposium on Experimental Robotics (ISER)*, Červen 2014, Maroko.

[2] MacCormick, J.: How does the Kinect work? [navštíveno 26.1.2016].
URL http://users.dickinson.edu/~jmac/selected-talks/kinect.pdf

[3] Meeussen W., et al: Autonomous Door Opening and Plugging In with a Personal Robot. *IEEE International Conference on Robotics and Automation*, 2010: s. 729–736, ISBN 0-7695-0984-3, ISSN 1050-4729.

[4] P. Viola, M. Jones: Rapid Object Detection using a Boosted Cascade of Simple Features. *Proc. of Computer Vision and Pattern Recognition Conf. (CVPR)*, 2001: s. 511–517.

[5] R. B. Rusu, W. Meeussen, S. Chitta, M. Beetz: Laser-based perception for door and handle identification. *Proc. of the Intl. Conf. on Advanced Robotics (ICAR)*, 2009.

[6] Senčuch, D.: *Modul pro otevření dveří pro PR2*. Bakalářská práce, FIT VUT v Brně, 2015.

[7] Szymon Rusinkiewicz, Marc Levoy: Efficient Variants of the ICP Algorithm. *IEEE 3-D Digital Imaging and Modeling*, 2001: s. 145–152, ISBN 0-7695-0984-3.

[8] WWW stránky: About ROS. [Online; navštíveno 14.12.2015].
URL http://www.ros.org/about-ros/

[9] WWW stránky: Aligning object templates to a point cloud. [Online; navštíveno 11.5.2016].
URL http://www.pointclouds.org/documentation/tutorials/template_alignment.php

[10] WWW stránky: Downsampling a PointCloud using a VoxelGrid filter. [Online; navštíveno 12.5.2016].
URL http://www.pointclouds.org/documentation/tutorials/voxel_grid.php

[11] WWW stránky: Filtering a PointCloud using a PassThrough filter. [Online; navštíveno 12.5.2016].
URL http://www.pointclouds.org/documentation/tutorials/passthrough.php

[12] WWW stránky: Gazebo. [Online; navštíveno 2.5.2016].
URL http://gazebosim.org/

[13] WWW stránky: Hardware Specs. [Online; navštíveno 18.12.2015].
URL https://www.willowgarage.com/pages/pr2/specs

[14] WWW stránky: Nodes. [Online; navštíveno 15.12.2015].
URL http://wiki.ros.org/Nodes

[15] WWW stránky: PCL About. [Online; navštíveno 21.12.2015].
URL http://pointclouds.org/about/

[16] WWW stránky: PCL Walkthrough. [Online; navštíveno 21.12.2015].
URL http://pointclouds.org/documentation/tutorials/walkthrough.php

[17] WWW stránky: Plane model segmentation. [Online; navštíveno 10.5.2016].
URL http:
//www.pointclouds.org/documentation/tutorials/planar_segmentation.php

[18] WWW stránky: Removing outliers using a StatisticalOutlierRemoval filter. [Online; navštíveno 12.5.2016].
URL http://www.pointclouds.org/documentation/tutorials/passthrough.php

[19] WWW stránky: tf - ROS Wiki. [Online; navštíveno 25.1.2016].
URL http://wiki.ros.org/tf

[20] Xiaoyan Wang, Hui Zhang, Sheng Liu: Reliable RANSAC Using a Novel Preprocessing Model. *Computational and Mathematical Methods in Medicine*, 2013.

[21] Yangsheng Xu, Huihuan Qian, Xinyu Wu: *Household Service Robotics*. Academic Press Inc, 2014, ISBN-13: 9780128009437.

Přílohy

Příloha A

Obsah CD

K bakalářské práci je přiloženo CD s touto adresářovou strukturou:

- `pr2_detect_doors/` – Balíček pro ROS s výslednou aplikací.
- `tex/` – Zdrojové soubory pro LaTeX.
- `other/` – Obsah složky není výsledkem mé práce, je zde pouze pro usnadnění testování výsledné aplikace v simulátoru Gazebo. Složka obsahuje uzel `but_pr2_tuck_arm`, pro složení robotových paží a soubor `doors.launch`, pro spuštění simulátoru Gazebo a v něm umístění robota PR2 společně s dveřmi a zdmi, vytvořený Danielem Senčuchem.
- soubor `BP.pdf` – Tento dokument.
- soubor `poster.pdf` – Plakát prezentující tuto práci.
- soubor `README.txt` – Stručný návod k použití aplikace.