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 OTEVŘENÍ DVEŘÍ PRO PR2

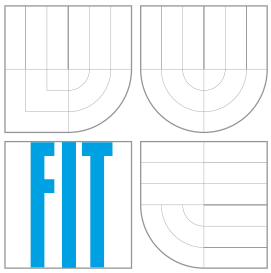
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

DANIEL SENČUCH

BRNO 2015

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 OTEVŘENÍ DVEŘÍ PRO PR2

DOOR OPENING FOR PR2

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — DANIEL SENČUCH
AUTHOR

VEDOUCÍ PRÁCE — Ing. MICHAL KAPINUS
SUPERVISOR

BRNO 2015

Abstrakt

Tato práce si klade za cíl prozkoumat možnosti otevírání dveří robotickou platformou PR2 na základě informací o poloze a vlastnostech dveří. Tyto informace musí být získány od jiného programu, který dodržuje komunikační protokol navržený v této práci. Následně by práce měla vyústit ve funkční program pro Robotický operační systém, který dovede s pomocí PR2 přinejmenším otevřít dveře směrem od robota a dostat robota skrz ně. Proběhlo úspěšné testování na dveřích od fakultní robotické laboratoře.

Abstract

This thesis aims to explore possibilities of door opening by the PR2 robotic platform based on information about the location and properties of the doors. This information must be obtained from another program that follows the protocol designed in this thesis. This thesis' effort should then produce a working program for the Robotic Operating System which – when run on the PR2 – is able to at least open doors in the direction from the robot and get the robot through them. The result has been successfully tested on the doors to the faculty robotic laboratory.

Klíčová slova

PR2, otevřít dveře, ROS, robotický operační systém, moveit, robot, robotika, kinematika

Keywords

PR2, open doors, ROS, robotic operating system, moveit, robot, robotics, kinematics

Citace

Daniel Senčuch: Modul pro otevření dveří pro PR2, bakalářská práce, Brno, FIT VUT v Brně, 2015

Modul pro otevření dveří 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 Senčuch
14. května 2015

Poděkování

Vedoucí mé práce, Ing. Michal Kapinus, ochotně věnoval nemalé množství času a úsilí pro to, aby mi poskytnul cenné rady při tvorbě této práce. Za to mu velice děkuji.

Dále bych chtěl poděkovat Rolandu Botkovi za to, že se rozhodl pro téma bakalářské práce, která podpoří tu moji, za její vypracování a za spolupráci se mnou.

Obsah

1 Úvod **2**

2 Teorie a představení klíčových prvků **3**
2.1 Kinematika robota 3
2.2 PR2 5
2.3 Robotický operační systém 6

3 Návrh řešení **13**
3.1 Specifikace problému a cíle 13
3.2 Existující řešení 13
3.3 Návrh architektury uzlu 14
3.4 Návrh postupu otevření dveří 15
3.5 Návrh postupu řešení 17

4 Realizace **18**
4.1 Obecný průběh realizace 18
4.2 Implementace 20
4.3 Testování 25

5 Závěr **27**

A Obsah CD **30**

Kapitola 1

Úvod

Robotika je prudce se rozvíjející obor a čím dál více robotů nachází aplikace v prostředích, kde musí pracovat v blízkosti lidí. V takových prostředích se robot musí umět orientovat a interagovat s nimi, aby se mohl dostat do cílového místa nebo splnil nějaký úkol. Nezbytnou schopností pro pohyb v uzavřeném prostoru je umět pracovat s dveřmi, tedy umět je alespoň otevřít, projet jimi a následně je zavřít. Právě na tento problém a jeho řešení je má práce zaměřená.

Podobné problémy byly již v minulosti mnohokrát řešeny a vyřešeny, nicméně buď nebyly vytvořeny jako modul pro robota PR2 a Robotický operační systém (dále ROS), nebo byly, ale pro jeho zastaralou verzi (např. modul *pr2_doors*, který byl již z veřejného repozitáře odstraněn), nebo nebyly veřejně dostupné. Můj výsledný modul byl psán pro aktuálně nejnovější distribuci ROSu, která na fakultním robotovi PR2 fungovala (její název je Hydro Medusa), a míněn ke zveřejnění v repozitáři. Co je ROS a proč jsem si ho vybral, se můžete dočíst v sekci 2.3. Podobně se můžete o PR2 dočíst v sekci 2.2.

Aby robot mohl dveřmi projet, musí je nejdřív najít a zjistit, kde je klika, jaký je to typ dveří a kliky, na kterou stranu se otevírají atd. Tímto se zabývá práce Rolanda Botky [1], která vznikala paralelně s touto prací a jejíž výsledný modul byl zamýšlen pro spolupráci s mým modulem. Můj modul pak zajišťuje uchopení kliky, její stisknutí a projetí robota otevřenými dveřmi.

V kapitole 2 vysvětlím problematiku kinematiky robota, popíšu robota PR2, pro nějž budu modul vytvářet, a představím veškerý software, který budu při tvorbě modulu používat a který stojí za zmínku. Ve 3. kapitole rozeberu již existující řešení podobných problémů, budu se zabývat návrhem architektury modulu, celkového rámce jeho vývoje a postupu robota k cíli. Popis celkové realizace, implementace kódu a testování výsledného programu jsou popsány v kapitole 4. Shrnutí úspěšnosti a výhled do budoucna jsou uvedeny v závěrečné kapitole 5.

Kapitola 2

Teorie a představení klíčových prvků

V této kapitole pojednám o základních pojmech a principech z kinematiky robota (sekce 2.1), poté představím fakultního robota PR2 (sekce 2.2) a nakonec vysvětlím Robotický operační systém a s ním spojené nástroje, které při vytváření své aplikace použiji (sekce 2.3).

2.1 Kinematika robota

Informace v této sekci jsem čerpal především z [8]. Robot by se měl být schopen pohybovat ve svém prostředí a případně s ním manipulovat. K tomu potřebuje znát své *manipulátory* (efektory). Příkladem manipulátoru u PR2 je ruka, která je složena z ramene, zápěstí a chapadla. Pohyblivými částmi na ruce jsou *klouby* – osy, kolem kterých se otáčí zbytek ruky a které jsou poháněny (nastavovány) pohony. Význačnou částí manipulátoru je *koncový efektor* (v případě ruky PR2 je to chapadlo). Dosažení nebo vypočítání nějaké cílové polohy koncového efektoru bývají v robotice časté úlohy.

Všechna tělesa, ze kterých se manipulátor skládá, se nazývají *kinematické členy*. Kinematické členy jednoho manipulátoru dohromady tvoří *kinematický řetězec*. Hlavní úlohou kinematiky je zkoumání vzájemného pohybu členů kinematického řetězce, zejména pohybu koncového členu vzhledem k počátečnímu.

> "Z mechaniky je známo, že poloha a orientace tělesa v prostoru je charakterizována šesti údaji. Většinou jsou to 3 hodnoty $[x, y, z]$ souřadnic nějakého referenčního bodu tělesa v základním kartézském souřadném systému a 3 úhly $[\alpha, \beta, \gamma]$ natočení nějakého referenčního systému pevně s tělesem spojeného, vzhledem k tomuto základnímu souřadnému systému. Říkáme, že volné těleso má v prostoru 6 stupňů volnosti." – [8], str. 8.

Z toho plyne, že manipulátor musí mít nejméně 6 vhodně umístěných kloubů – stupňů volnosti – aby uchopený předmět dokázal volně přemisťovat. Výše citovaný odstavec ale neplatí jen pro uchopený předmět, nýbrž také pro koncový efektor.

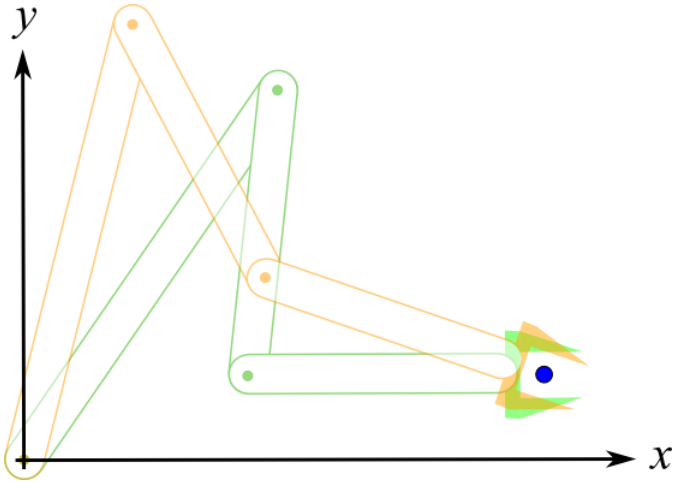
Každý kloub má v každém čase nějaké nastavení, tzv. *kloubovou proměnnou*. Kloubové proměnné bývají označovány symbolem q.

Přímá úloha kinematiky

Označíme-li si vektor kloubových souřadnic $\mathbf{q} = [q_1, q_2, q_3, q_4, q_5, q_6]^T$ (u robota se šesti klouby) a vektor pozice koncového členu robota $\mathbf{P} = [x, y, z, \alpha, \beta, \gamma]$ pak existuje jednoznačné zobrazení z prostoru kloubových souřadnic do prostoru kartézských souřadnic, které zapíšeme ve formě $\mathbf{P} = f(\mathbf{q})$, což představuje 6 rovnic, které jsme u mnoha manipulátorů schopni sestavit běžnými znalostmi geometrie. Nalezení těchto rovnic a vypočítání polohy koncového efektoru na základě hodnot kloubových proměnných se nazývá *přímá úloha kinematiky.*

Inverzní úloha kinematiky

Pro člověka je však přirozenější si určit cílovou polohu a potom k ní koncový efektor dostat, stejně jako je to nezbytné u mnoha algoritmů. Vypočítání hodnot kloubových souřadnic na základě polohy koncového efektoru se nazývá *inverzní úloha kinematiky.* Je oproti přímé kinematice častější, ale zato mnohem složitější mj. proto, že může mít nekonečně mnoho řešení.



Obrázek 2.1: Příklad inverzní úlohy kinematiky s teoreticky nekonečně mnoha řešeními[1]

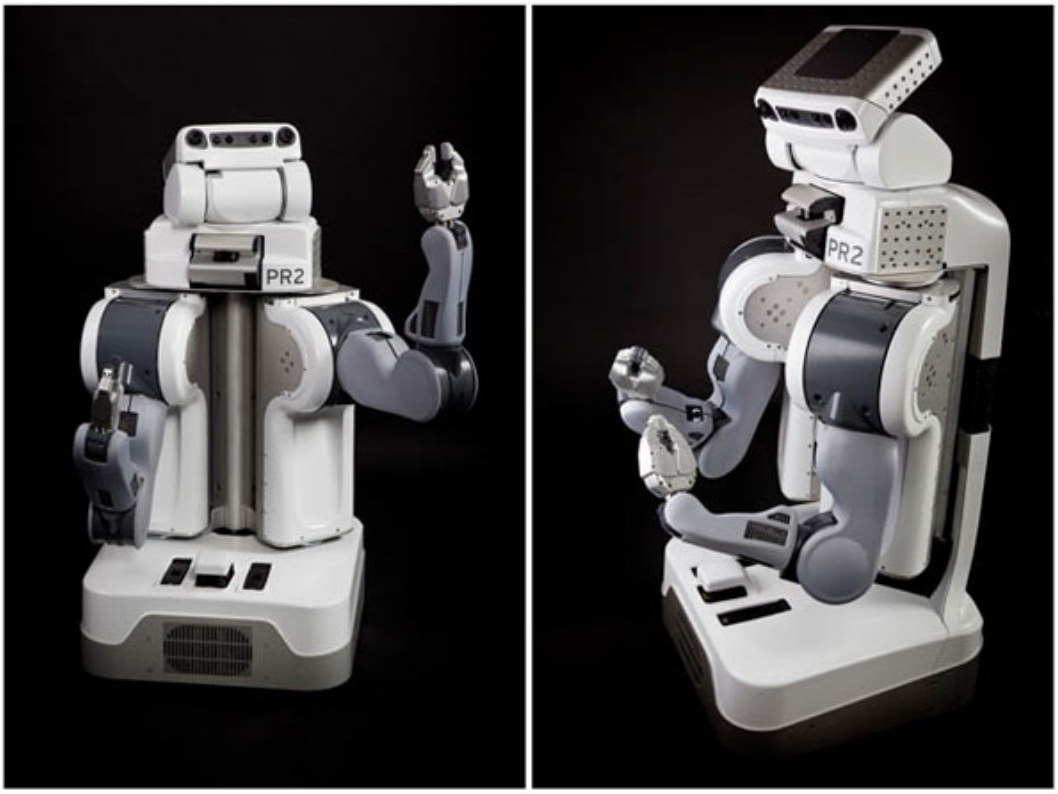
Plánování

Je také potřeba zjistit všechny hodnoty kloubových proměnných na cestě k cíli. Plánovač může používat přímou i inverzní kinematiku k vypočítání dílčích cílů nebo sestavení funkcí kloubových proměnných s časem jako parametrem. Plánovat lze přímo (učením), kdy člověk drží rameno a vede ho po požadované trase. Robot si tuto trasu zapamatuje a v budoucnu ji napodobí (včetně všech případných nedokonalostí, které člověk do trasy zanesl).

Druhou možností je plánovat nepřímo (offline), tedy předem připravit cestu pro manipulátor např. zadáním křivek, po kterých se má rameno pohybovat, a vykonat ji někdy později.

Třetí možností je přímé (online) řešení úloh kinematiky v reálném čase, např. když má robot provádět pohyb na základě údajů ze senzorů v měnícím se prostředí. Pokročilé

[1] Převzato z http://en.bioinformatyk.eu/contest-articles/application-of-the-cyclic-coordinate-descent-algorithm-in-the-protein-loop-closure-problem.html



Obrázek 2.2: PR2.[2]

plánovače tedy dokáží naplánovat pohyb manipulátoru tak, aby se vyhnul objevujícím se překážkám.

2.2 PR2

Robot, kterého jsem naprogramoval k otevírání dveří a který je na naší fakultě přítomný, se jmenuje PR2 (zkratka pro Personal Robot 2) – můžete ho vidět na obrázku 2.2. Byl vyvinut firmou Willow Garage, která také velikou měrou přispěla při vývoji ROSu a vytvořila mnoho modulů pro PR2, na němž je zamýšleno ROS spouštět.

Hardwarová výbava

Výbava efektorů PR2 zahrnuje mj. podvozek, který může robotem hýbat libovolným směrem, výsuvnou páteř, otáčecí a nakláněcí hlavu a dvě ruce s osmi stupni volnosti (pro srovnání lidská ruka jich má přibližně sedm).

Mezi senzory lze najít Microsoft Kinect (na robotovi na obrázku 2.2 není), mono a dva páry stereo kamer na hlavě, naklánějící se laserový hloubkoměr pod hlavou, kamery na předloktí, akcelerometr na chapadle a laserový hloubkoměr na podvozku. Kinect začal být na robota přidáván až jistou dobu po začátku jeho prodeje. Výstup Kinectu je ovšem kvalitnější, než výstup stereo kamer, které má robot zabudované v hlavě.

[2] Převzato z http://www.randomrobotics.com/category/flying/

O náročné výpočty se starají dva čtyřjádrové procesory Intel i7 Xeon. Dále je k dispozici 24 GB paměti RAM, 2 TB úložného prostoru na pevných discích a baterie s výdrží až 2 hodin provozu. S robotem se dá komunikovat pomocí ethernetového i WiFi rozhraní.

Robota je možné ovládat pomocí tzv. Run-Stop ovladačů. Jeden je v podobě velkého tlačítka na zádech robota, druhý je jako dálkové ovládání. Jejich aktivace okamžitě vypne všechny motory, ale nechá vše ostatní (včetně diagnostiky motorů) zaplé. Ruce mají v sobě zabudovaná pružinová protizávaží, která zajišťují, že i s vypnutými motory zachovají ruce svou polohu. Jde s nimi však jednoduše hýbat vnějším zásahem (např. rukami).

Další informace o hardwaru lze nalézt v [12].

Software a práce s robotem

V době psaní této práce byl na PR2 nainstalován operační systém Ubuntu 12.04 LTS a ROS Hydro Medusa včetně všech součástí zmíněných v sekci 2.3. Komunikace s robotem je možná pomocí síťových rozhraní a případně programu `ssh`, ať už z pracovní stanice v robotické laboratoři, nebo z vlastního počítače.

Samotná práce a spouštění uzlů ROSu (vizte 2.3) vypadá následovně: Robot je zapnut a je s ním vytvořeno síťové spojení pomocí `ssh`. Poté se na pracovním počítači změní v terminálu hodnota proměnné prostředí `ROS_MASTER_URI` na síťovou adresu robota. Od této chvíle budou všechny uzly, které budou spuštěné na pracovním počítači, komunikovat s ROSem, který je spuštěný na robotovi, stejně, jako by byly spuštěné na robotovi. Proč je tato skutečnost důležitá, se dozvíte v následující sekci a zejména v její podsekci 2.3. Nyní je možné uzly na pracovním počítači spouštět běžným způsobem.

2.3 Robotický operační systém

Robotický operační systém (zkráceně ROS) je framework podporující vytváření a běh aplikací pro roboty. Jeho hlavní vlastností je otevřenost a podpora širokého spektra robotů. Má rozsáhlou komunitu (jednotlivci i firmy), která zveřejňuje nové nástroje, knihovny a balíčky a následně o nich diskutuje a udržuje je [9]. Balíčky jsou dostupné na webu robotického operačního systému[3]. ROS a mnoho jeho knihoven a uzlů poskytují programátorské rozhraní pro jazyky C++ a Python.

Na fakultním PR2 v době psaní této práce fungovala distribuce ROS Hydro Medusa, která je zaměřená na operační systém Ubuntu 12.04 LTS. Proto tuto distribuci budu také využívat a výsledný modul bude odladěný zejména pro ni.

Architektura ROSu sestává především z *uzlů*, které si posílají *zprávy* na různá *témata*. Princip těchto prvků bude vysvětlen v následujících podsekcích. Informace o nich jsem čerpal z webových stránek ROSu, které o této problematice pojednávají [14].

Balíček

Balíček je elementární modul ROSu. Kromě zdrojových kódů uzlů a knihoven obsahuje taky XML soubor se svým popisem a vyjmenováním všech svých závislostí (zejména na jiných modulech ROSu). Dále obsahuje soubor, který popisuje, jak se mají zdrojové kódy v balíčku přeložit, jaké spustitelné soubory mají vzniknout atd. Na základě tohoto souboru

[3] http://www.ros.org/browse/list.php

pak ROS vytvoří řádný *Makefile*[4]. Balíček může obsahovat více uzlů.

Dále může obsahovat *spouštěcí soubory*, které svou XML syntaxí popisují, které uzly se mají spustit a případně s jakými parametry. Díky nim je tedy možné bezpracně spustit najednou několik uzlů či nástrojů se správným nastavením.

Uzel

Uvnitř balíčku jsou uzly – spustitelné soubory využívající klientskou knihovnu ROSu. Jako uzly se také označují procesy spuštěné z těchto souborů. Tyto procesy jsou při spuštění zaregistrovány Hlavním uzlem (Master) pod požadovaným jménem a mohou si mezi sebou posílat zprávy. Mohou také fungovat jako *služby* a poskytovat možnost vzdáleného volání procedur[5] pro jiné uzly, či za běhu měnit své parametry (a tím svůj stav) na podněty od jiných uzlů.

Určitou nadstavbou nad službami jsou *akční servery* a *akční klienti*[6], které oproti službám navíc umožňují si za běhu vyměňovat zpětnou vazbu o průběhu plnění úkolu a případně úkol na žádost zrušit. Jsou proto vhodné pro úkoly, které zaberou dlouhou dobu (jako například ovládání otevírání a zavírání chapadla robota).

Hlavní uzel mj. má přehled o všech spuštěných uzlech, tématech, jejich jménech. Nově spuštěné uzly a témata jím musí být schválena. Díky němu je možné, aby se dva uzly našly a začaly spolu komunikovat. Vždy to tedy musí být první uzel ROSu, který byl spuštěn. Příklad jeho práce můžete vidět na obrázku 2.3.

Témata a zprávy

Téma je pojmenovaná virtuální sběrnice, skrz kterou si uzly vyměňují zprávy. Může mít teoreticky neomezené množství *přispěvatelů* (publikují zprávy na dané téma) a *odběratelů* (čtou zprávy z daného tématu). Jednotlivé uzly většinou nevědí, jaké jiné uzly na daném tématu komunikují, ani zda nějaký uzel zprávy přijímá. Z tohoto důvodu jsou témata vhodná pro jednosměrnou proudovou komunikaci. Pokud uzly vyžadují i zpětnou vazbu, je pro ně lepší využít zmíněných *služeb* a RPC.

Témata jsou silně typovaná podle zpráv, které se přes ně přenášejí. Tento typ není vynucován u přispěvatelů, ale Hlavní uzel nedovolí jiným uzlům odebírat zprávy z daného tématu, dokud se typy nebudou shodovat.

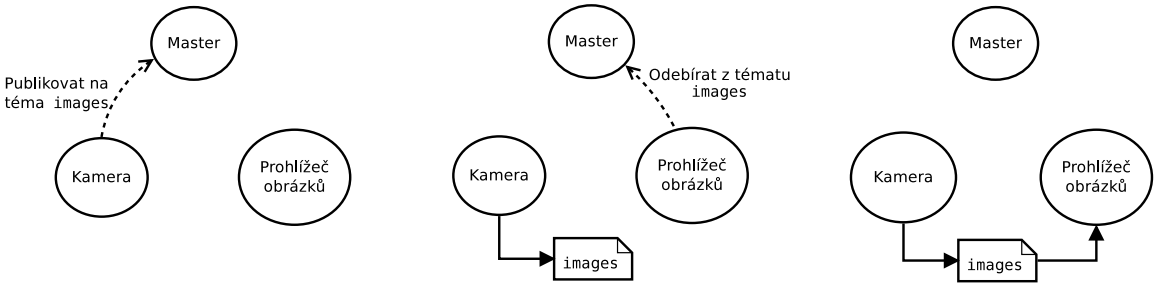
Zpráva je jednoduchá datová struktura podpoující standardní primitivní datové typy (celá čísla, čísla s plovoucí řádovou čárkou atd.), pole a další zanořené struktury a pole. Případné definice zpráv jsou součástí balíčku.

Elegance ROSu spočívá v tom, že senzory a efektory robota mají pro sebe vyhrazená svá témata, přes která se komunikace odehrává. Kamery tak publikují na svá témata své obrazové výstupy a jiný, např. uživatelem vytvořený, uzel může začít z tohoto tématu zprávy odebírat, čímž začne dostávat obraz. Ten může třeba dále zpracovávat. Na druhou stranu také může uzel publikovat zprávy na téma, ze kterého je odebírá např. ovladač podvozku robota. Ovladač tyto zprávy zpracuje a transformuje na primitivní příkazy (změny napětí) pro motory jednotlivých kol podvozku.

[4] V případě uzlů napsaných v jazyce Python není překlad nutný, ale vzhledem k tomu, že má práce je napsaná v jazyce C++, nebudu tento případ uvažovat.

[5] RPC – remote procedure call

[6] Implementované knihovnou actionlib – http://wiki.ros.org/actionlib



Obrázek 2.3: Příklad práce Hlavního uzlu. Uzly zde postupně žádají o publikování a odebírání zpráv z tématu `images`.

Soubory bag

Bag je formát souborů určený pro uchovávání dat o zprávách ROSu. Jejich přípona je stejná, jako název – `.bag`. ROS má nástroj *rosbag*, který umí začít odebírat zprávy z určeného tématu a ukládat je právě do bag souborů společně s informací o čase, kdy zprávy přišly. Tyto soubory se dají později pomocí rosbagu přehrát a tím kdykoliv napodobit proud dat na daném tématu.

tf

TF[7] je jednou z nejdůležitějších a nejpoužívanějších knihoven (balíčků) ROSu. Má proto na stránkách ROSu zevrubnou dokumentaci [15] a byla popsána v konferenčním článku [4].

Mezi běžné nutnosti při práci s robotem je udržovat přehled o jednotlivých soustavách souřadnic a úspěšně mezi nimi souřadnice transformovat. Tato problematika je ovšem velmi složitá a byla často zdrojem chyb v programech. TF nabízí standardizovaný způsob, jak tento problém řešit.

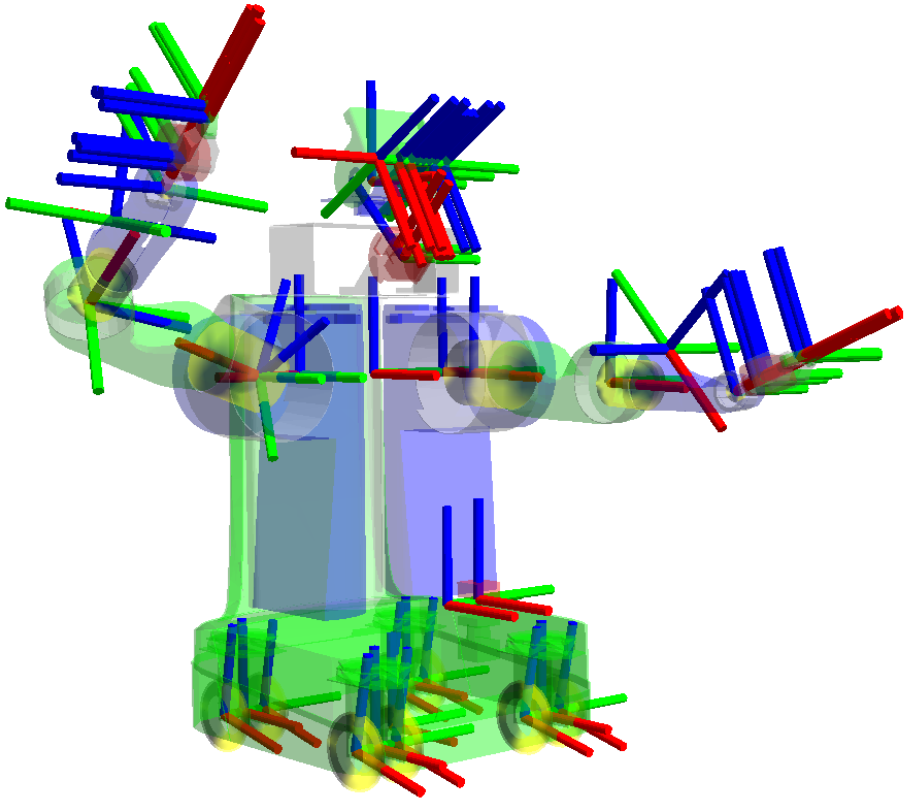
Principy tf

Soustavy souřadnic, nazývané *rámce*, jsou pravotočivé: osa x směřuje dopředu, osa y doleva a osa z nahoru. Rámce jsou hierarchicky uspořádány do stromu (je možné ale také vytvořit les). Kořenem stromu bývá rámec s názvem `map` nebo `odom_combined`. Tyto rámce jsou nehybným počátkem celého světa. Všechny ostatní rámce mají rodiče a jsou definovány vektorem, který značí posunutí počátku soustavy souřadnic potomka oproti rodiči, a kvaternionem, který udává rotaci počátku soustavy souřadnic potomka oproti rodiči. (Mají tedy 6 stupňů volnosti.) Je možné potom jednoduše převést souřadnice bodu, vektoru atd. z jednoho rámce do jiného, který je ve stejném stromu.

Vektory a kvaterniony, které rámce definují, se mohou v čase měnit. TF si udržuje ve vyrovnávací paměti informace o všech rámcích včetně jejich historie. Díky tomu je možné třeba převádět souřadnice do rámce v takovém stavu, v jakém byl před pěti sekundami.

Každý uzel využívající tf má vlastní vyrovnávací paměť – tf je distribuovaný systém a všechny vysílače se podílejí na jeho tvorbě. Vysílače jsou uzly, které vysílají rámce, to znamená, že je definují a informace o nich posílají na společná tf témata. Uzly, které tyto informace zachytávají a využívají, se nazývají přijímače.

[7] TF může být v tomto kontextu zkratkou pro několik slov. Nejběžněji se chápe jako zkratka slova transform.



Obrázek 2.4: Ukázka různých tf rámců, které PR2 implicitně vysílá. Počátky jejich souřadných systémů jsou značeny třemi osami: červená pro souřadnici x, zelená pro y a modrá pro z.[8]

PR2 a tf

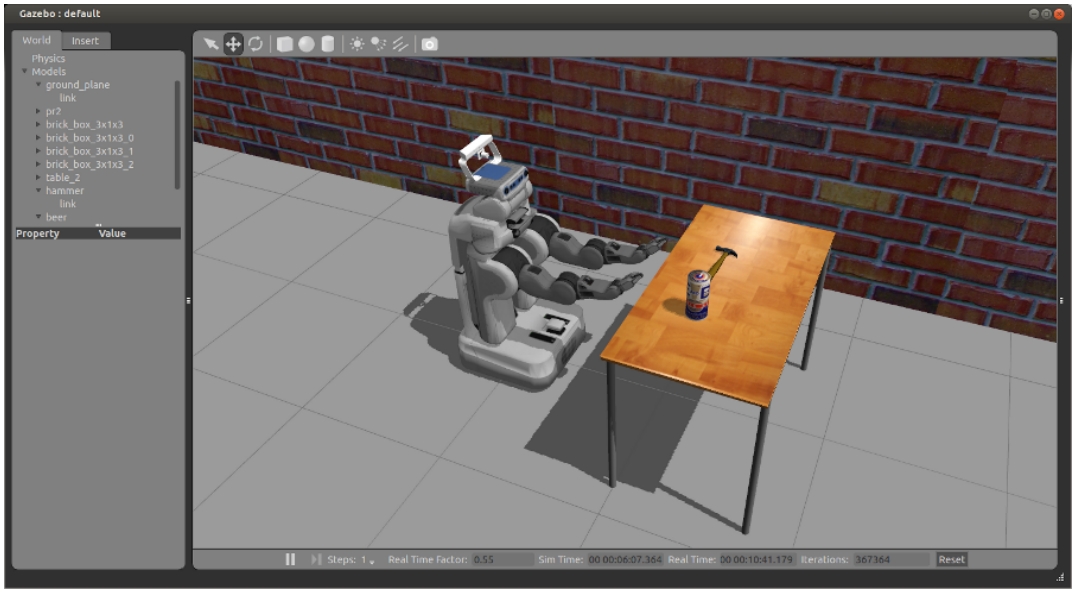
PR2 už při zapnutí automaticky vysílá rámce udávající polohu všech jeho kloubů, a senzorů (vizte obrázek 2.4). Mezi význačné rámce patří:

- `odom_combined`: kořen stromu všech ostatních rámců vysílaných robotem. Nehybný počátek světa, jehož poloha se počítá s pomocí odometrie (odhadnutí změny polohy na základě dat ze senzorů pohybu).
- `base_footprint`: potomek rámce `odom_combined`; značí střed pomyslné spodní stěny základny robota.
- `r_wrist_roll_link` a `l_wrist_roll_link`: koncové efektory, se kterými počítá MoveIt! při plánování. Ve skutečnosti jsou to klouby, které otáčejí pravým a levým chapadlem, takže to nejsou nejzazší body na robotově ruce. (Otevírání a zavírání chapadel se však neovládá pomocí MoveIt!u.)

Gazebo

Gazebo (vizte obrázek 2.5) je robotický simulátor, do nějž byla zabudována podpora ROSu a PR2. Je řízen realistickým fyzikálním enginem, podporuje moduly a, podobně jako ROS,

[8] Převzato z http://wiki.ros.org/tf



Obrázek 2.5: Simulátor Gazebo.

má aktivní komunitu. Vytváří a napodobuje v ROSu všechna témata, která by vytvořil i skutečný robot (např. pro ovládání podvozku), poslouchá na nich a podle příchozích zpráv mění i model robota v simulaci. Vytváří také témata pro výstup dat, posílá na ně patřičné zprávy a vysílá tf rámce. Je tak schopen poskytovat ostatním uzlům například informace o stavu kloubů robota nebo generovat data z jeho senzorů [11]. Z pohledu ROSu a práce s uzly a tématy je tedy simulovaný robot naprosto totožný s reálným.

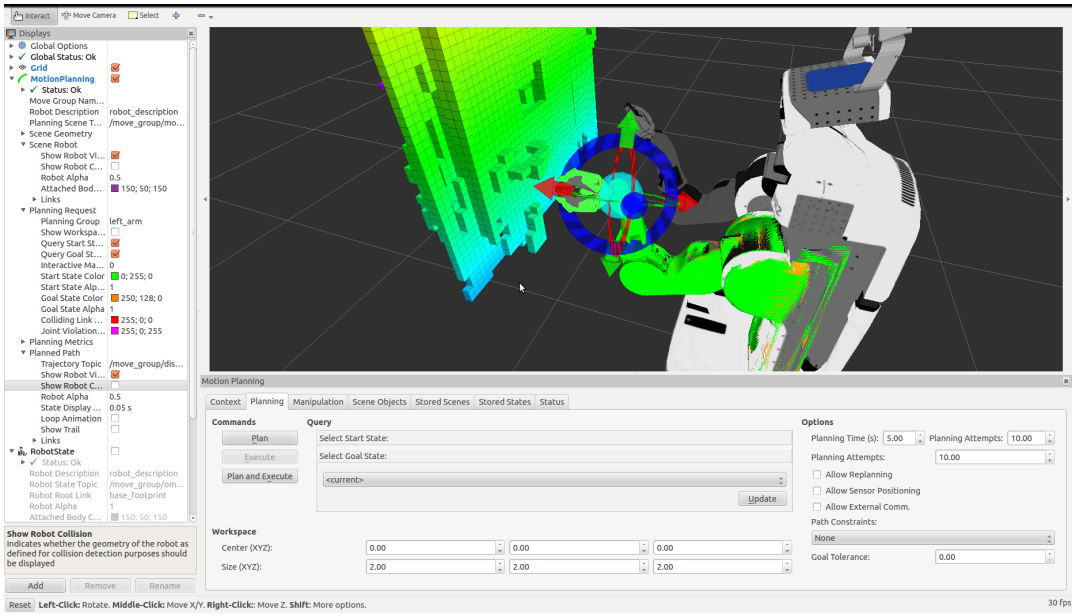
Prostředí se v Gazebu dá budovat pomocí konzolových příkazů, spouštěcích souborů, přímo uvnitř uzlů, nebo pomocí grafického uživatelského rozhraní. Jeho zjevnou výhodou je, že si na něm mohu průběžně zkoušet své algoritmy bez potřeby skutečného robota a riskování, že poškodím jeho nebo okolí.

MoveIt!

MoveIt! je sada knihoven integrovaná s ROSem, která implementuje nejnovější pokroky v plánování pohybu, manipulaci, vnímání prostoru, kinematice, ovládání a navigaci. Dokáže spojovat informace ze stereo kamer, laserových dálkoměrů, kinectu a dalších senzorů, aby vytvořil model prostředí, a v tomto prostředí dokáže rozeznat primitivní geometrické tvary. Výrazně (oproti dřívějším metodám) ulehčuje například počítání přímých a inverzních úloh kinematiky (vizte sekci 2.1) včetně různých omezení cesty nebo dokonce orientace drženého objektu (například držená sklenice s vodou vždy musí být dnem dolů). MoveIt! umí také v reálném čase detekovat kolize předtím, než k nim dojde, a naplánovat cestu kolem blokujícího objektu. [3]

Jádrem je uzel `move_group`. Ten získává informace ze senzorů robota, jeho ovladačů a samozřejmě z různých témat ROSu, aby mohl např. přijímat požadavky na plánování pohybu, vykonat je a poskytnout nějakou zpětnou vazbu. Poskytuje programátorské rozhraní pro jazyky C++ a Python [10].

Modulární architektura umožňuje MoveIt!u používat různé plánovače. V současnosti



Obrázek 2.6: Rviz a jeho modul MotionPlanning.

je jako implicitní plánovač OMPL[9], který je založený na vzorkování prostoru konfigurací efektoru a hledání vhodné mapy cesty mezi jednotlivými vzorky.

Druhy plánování

Dle [13] umožňuje MoveIt! *plánování v kloubovém prostoru*, *v kartézském prostoru* a *do cílové pózy*. Plánování v kloubovém prostoru odpovídá přímé úloze kinematiky – programátor zadá cílové hodnoty kloubových proměnných a plánovač se jich nějakým způsobem snaží dosáhnout.

Plánování do cílové pózy odpovídá inverzní úloze kinematiky – programátor zadá cílové souřadnice a orientaci (ve formě normalizovaného kvaternionu) pro nějakou část manipulátoru (nemusí to být koncový efektor) v libovolném existujícím souřadném systému a plánovač se bude snažit patřičně nastavit kloubové proměnné. Síla MoveIt! spočívá mj. v tom, že zaznamenává informace o souřadném systému každého článku manipulátoru a mezi těmito systémy provádí transformace (díky knihovně TF[10]). Lze tak například je noduše naprogramovat, aby hlava robota byla vždy otočená směrem k pravému chapadlu (souřadnice $[0, 0, 0]$ souřadného systému pravého chapadla). Robot od té chvíle bude hýbat hlavou tak, aby vždy byla otočená k chapadlu, ať už s ním hýbe jiný uzel nebo třeba člověk svýma rukama.

Při plánování do cílové pózy lze ještě specifikovat omezení pro plánovač – vymezit toleranci pro odchylku od počáteční polohy, pro odchylku v orientaci koncového efektoru a pro změnu nastavení vybraných kloubů.

Plánování v kartézském prostoru, podobně jako plánování do cílové pózy, odpovídá inverzní úloze kinematiky. Narozdíl od něj je ale cíl specifikován společně se seznamem souřadnic průběžných poloh na cestě k němu. Plánovač potom tyto průběžné polohy interpoluje. Tento způsob plánování je vhodný pro malé jemné pohyby, které musí proběhnout

[9] The Open Motion Planning Library – http://ompl.kavrakilab.org/
[10] http://wiki.ros.org/tf

po jedné určité dráze (například stisknutí kliky dveří dolů).

Rviz

Velmi užitečným nástrojem je Rviz (zkratka pro ROS 3D Robot Visualizer). Jeho grafické uživatelské rozhraní na první pohled připomíná Gazebo, ve skutečnosti ale jen poslouchá na vybraných tématech a získaná data vizualizuje. Nezáleží, zda data pochází ze simulátoru, reálného robota nebo jiného uzlu, pokud se názvy témat shodují. Uživatel může například přidat mezi témata, která chce mít vizualizována, výstup z kamery robota a Rviz zobrazí přijímaný obraz. Také uživatel může chtít zobrazovat komplexní informace o PR2. Pak přidá modul RobotModel a Rviz bude zobrazovat model robota v trojrozměrném prostředí a případně i do něj promítat výstup ze senzorů. Rviz také umožňuje zobrazovat tf rámce, podobně jako na obrázku 2.4.

Pro mě je nejdůležitější modul MotionPlanning (obrázek 2.6), který zobrazuje stav robota a využívá knihovnu OMPL pro plánování pohybu jeho rukou. Poskytuje informace o stavu jakéhokoliv kloubu a ramene robota (např. souřadnice v prostoru a orientace ve formátu normalizovaného kvaternionu). Také umožňuje pohodlně hýbat s chapadly robota pomocí grafického uživatelského rozhraní.

Kapitola 3

Návrh řešení

V této kapitole nejdříve definuji problém, který se tato práce pokusí vyřešit (sekce 3.1), poté představím dosavadní zajímavá řešení (3.2). Následovat bude návrh architektury uzlu (3.3), který stejně jako sekce o samotném plánu postupu při řešení problému (3.4) byl inspirován existujícími řešeními. Nakonec (3.5) popíšu celkový iterativní proces, kterým se plánuji k výsledku dobrat.

Od tohoto bodu budu používat zkratky UpD (uzel pro detekci) k označení uzlu, na kterém pracoval Roland Botka [1], a UpO (uzel pro otevření) k označení uzlu, který je předmětem této práce.

3.1 Specifikace problému a cíle

Cílem této práce je vytvořit uzel pro fakultního robota PR2, díky kterému bude robot schopen otevřít alespoň dveře od robotické laboratoře směrem od sebe a následně jimi projet. Tento uzel bude spolupracovat s UpD – bude od něj přijímat informace o poloze a druhu dveří a kliky prostřednictvím zpráv a témat ROSu. Tyto informace UpD zjistí díky AR kódu umístěného v určité poloze vůči klice dveří. Samotné otevření dveří (přesné souřadnice pro pohyb ruky atd.) tedy nebude natvrdo zakódováno v programu, ale bude parametrizováno pozicí robota vůči dveřím a polohou kliky vůči ose otáčení dveří a AR kódu. Řešení předpokládá prostředí bez překážek a komplikací (zamčené dveře atd.).

Dle časových možností a obtížnosti se také pokusím implementovat práci s kontrolou kolizí a otevírání dveří směrem k sobě. Dále bych rozšířil repertoár dveří, které robot dokáže otevřít, případně implementoval kontrolu na zamčené dveře a vyrovnávání se s dalšími situacemi, které jsou horší, než ideální.

3.2 Existující řešení

Pokusy otevřít dveře robotem existují již přes 20 let [6] [7] a, ačkoliv předpokládají jiný druh robota a plánování, než budu používat já, jsou inspirující – při návrhu řešení jsem část z nich využil. Dlouho ale bylo vyřešeno pouze otevírání dveří směrem od robota a navíc nikterak robustně.

Řešení Willow Garage

Otevírání směrem k robotovi vyřešili až vývojáři z Willow Garage [5] v roce 2010. Vytvořili k tomuto účelu vlastní plánovač pohybu založený na vyhledávání v grafu [2]. Tento plánovač

však nevyužiji, protože jeho princip není jednoduchý na pochopení. Navíc mám k dispozici novější řešení – MoveIt! a jeho plánovač, jehož vhodnost nemusí být horší, než vhodnost plánovače od Willow Garage. Srovnání plánovačů není jednoduchý úkol a já neaspiruji na co nejrychlejší plánování s nejpřirozenějšími pohyby.

Willow Garage nahráli zdrojové kódy svého uzlu `pr2_doors` do repozitáře ROSu, aby z nich mohli všichni čerpat. Byly však uzpůsobeny pro starší verzi ROSu a z neznámého důvodu odstraněny z repozitáře dřív, než jsem si je mohl stáhnout. Nakonec jsem je našel trochu upravené v repozitáři nadšence, který je kdysi modifikoval, a podařilo se mi je získat. Tudíž se mohu nechat inspirovat řešením Willow Garage, o kterém také jeho tvůrci napsali alespoň dva články.

3.3 Návrh architektury uzlu

Můj UpO bude v každém případě oddělen od UpD Rolanda Botky. Díky tomu budeme každý moci pracovat na svém uzlu relativně nezávisle. Závazné pro nás bude jen společné rozhraní – struktura zpráv ROSu, které si uzly mezi sebou budou vyměňovat. Za předpokladu dobře navrženého rozhraní bude také zajištěna modularita – UpO nebo UpD bude moci být přepracován nebo vyměněn a funkčnost bude stále zachována.

Nabízí se tedy otázky, jaké zprávy si budou uzly posílat a zda by nebylo vhodné uzly rozštěpit dále. Rozhodnutí o zprávách je jednoznačné: UpD bude UpO posílat informace o dveřích (druh, stav atd.) a TF rámce (osy dveří, kliky atd.). Přesný popis a obsah zpráv je popsán v sekci 4.2.

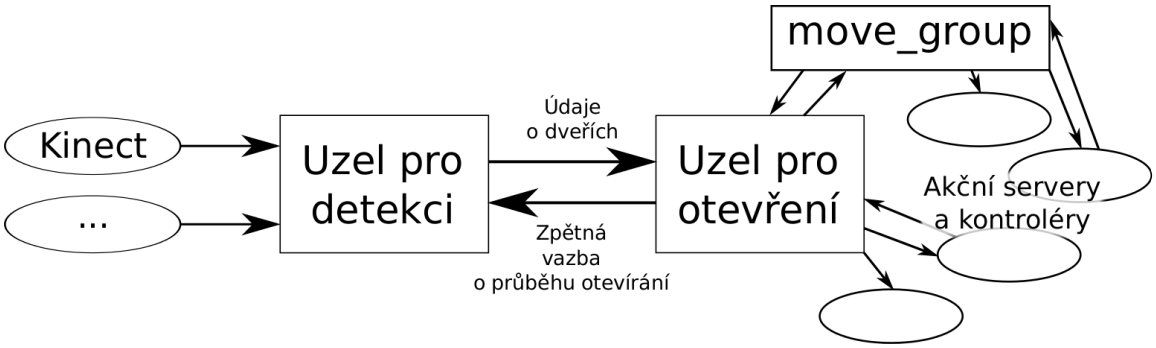
V této sekci je však otázka štěpení mnohem více relevantní. Možnosti pro UpO jsou následující:

1. Vytvořit uzel jako jeden spustitelný celek, který by se staral o celý proces od chycení kliky až po konečnou pozici. Stále by však byl modulární v kontextu programovacího jazyka – výsledný spustitelný soubor by byl složen dohromady z několika knihoven.

2. Rozdělit uzel na části schopné samostatného běhu – akční servery a klienty, které jsem popsal v sekci 2.3. Každý úkon, jako chycení kliky, stlačení kliky, odtlačení dveří atd. by byl implementován jako akční server, který by po spuštění čekal, až bude jeho služba zavolána sjednocujícím akčním klientem, řídícím celkový chod programu.

Možnost 2 je přesně ta, kterou ve svém řešení zvolili vývojáři z Willow Garage. Zajišťuje lepší modularitu a celkovou robustnost uzlu. Dále by například služby jednotlivých akčních serverů mohly být využity i jinými akčními klienty, než tím, který obstarává detekci a otevření dveří. Tato možnost je ovšem složitější na implementaci a vyžaduje znalost rozhraní knihovny `actionlib`. Já se však ve své práci nebudu zaměřovat na robustnost ani rychlost aplikace a proto zvolím možnost číslo 1. Při případném vývoji uzlu i po odevzdání této bakalářské práce by nemělo být složité jednotlivé úkony izolovat a rozdělit na akční servery. Je to však nad rámec mé bakalářské práce.

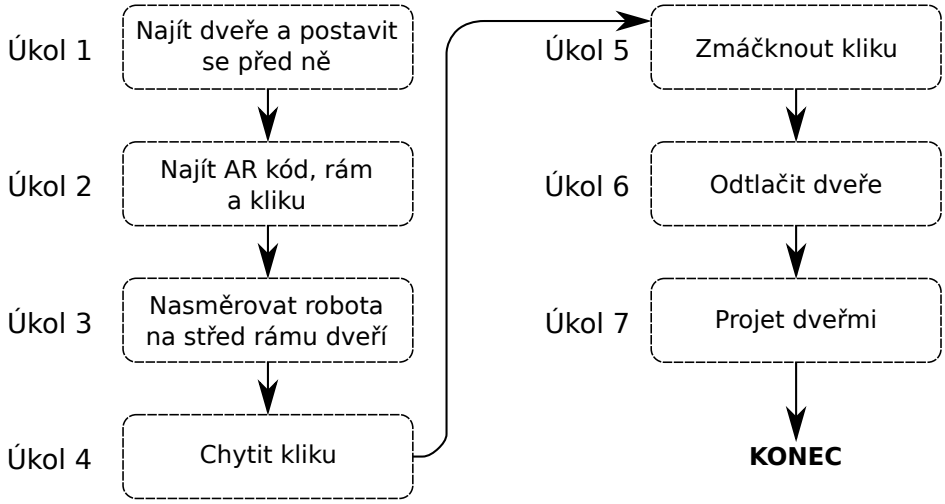
Výsledná architektura je znázorněna na obrázku 3.1. Ačkoliv jsem napsal, že bude podle možnosti 1, stejně se tomu, aby byl uzel akčním klientem, nelze úplně vyhnout, neboť pomocí akčních serverů je potřeba mj. ovládat otevírání a zavírání chapadel robota.



Obrázek 3.1: Ilustrace předpokládaného posílání zpráv mezi uzly. Uzel `move_group` byl zmíněn v sekci 2.3.

3.4 Návrh postupu otevření dveří

Dílčí podproblémy, které robot musí překonat, aby dveře otevřel, jsou znázorněny na obrázku 3.2. Úkoly 1 a 2 má na starosti UpD; na UpO připadá zbytek. V každém z úkolů se mohou vyskytnout různé chybové stavy, obecně řečeno je ale UpO příliš řešit nebude. Pokud chybový stav zdetekuje, ukončí svou činnost.



Obrázek 3.2: Rozložení problému otevření dveří na podproblémy.

Nasměrování robota do středu dveří

Tento úkol je velmi důležitý pro konečné projetí dveřmi. Jde o postavení robota čelem přesně proti dveřím tak, že po rozjetí by projel středem rámu. Je velmi pravděpodobné, že zrovna v takové poloze však po dokončení úkolu 1 nebude, neboť je možné se do ní s jistotou dostat až s pomocí informací získaných v úkolu 2.

V tomto bodě by UpO měl mít od UpD informace o dveřích a TF rámce osy otáčení dveří, AR kódu a kliky. UpO má tedy za úkol všechny tyto informace patřičně zpracovat. Jakmile je s tím hotov, najde s pomocí souřadnic okrajů dveří jejich střed, promítne ho na podlahu a posune o určitou vzdálenost směrem přede dveře. Nyní mám bod, do kterého robot musí dojet, aby byl ve středu rámu.

Orientaci robota můžu nastavit tak, že naprogramuji, aby robot čelil bodu ve středu dveří, který byl získán z procesu najetí před střed dveří.

Chycení a zmáčknutí kliky

V tomto bodě by robot měl stát čelem ke dveřím v dostatečné vzdálenosti, aby na kliku dveří dosáhl. S pomocí informací získaných od UpD by neměl být problém zadat cíl pro plánovač pohybu na souřadnice $[0, 0, 0]$ rámce kliky (nebo případně s nějakou předem danou odchylkou) se správným natočením chapadla.

Následně se chapadlo zavře a pohne na určitou odchylku (opět předem udanou informacemi o dveřích) od počátku souřadného systému kliky tak, aby byla stisknuta.

Odtlačení a projetí dveřmi

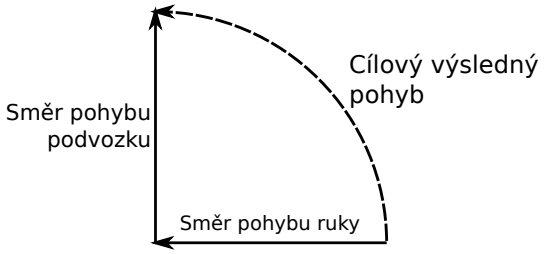
Teoreticky by pro robota stačilo dveře jen trochu odtlačit, pustit kliku, nastavit ruku tak, aby přesahovala podvozek, rozjet se dopředu a ruka by dveře odtlačila. Já bych však rád alespoň pro část dráhy otevírajících se dveří implementoval poněkud sofistikovanější způsob. Se znalostí souřadnic umístění osy otáčení dveří a jejich kliky můžu pomocí analytické rovnice pro kružnici

$$(x - x_0)^2 + (y - y_0)^2 = r^2 \tag{3.1}$$

spočítat souřadnici v ose y pro každou souřadnici v ose x a naopak. Souřadnice v ose x by byla dána pohybem ruky do strany (s konstatní rychlostí) a je potřeba dopočítat změnu vzdálenosti ve směru osy y, kterou podvozek robota musí urazit, aby ve výsledku koncový efektor opsal část dráhy kružnice. Toto je ilustrováno na obrázku 3.3. Musela by stále být kontrolována poloha ruky a podvozku. Rychlost podvozku by bylo nutno řídit, aby opravdu byl tam, kde má. (Je mnohem jednodušší za běhu řídit rychlost podvozku, než rychlost ruky.)

Zároveň se musí měnit natočení koncového efektoru tak, aby bylo vždy kolmé ke dveřím. Knihovna TF ovšem umožňuje spočítat vektory kolmé k ploše nebo k jinému vektoru. Tento vektor dále stačí transformovat na kvaternion a udat ho jako součást cíle pro úlohu inverzní kinematiky.

Tímto by měl být úkol otevření dveří dokončen. V případě, že by se postup zmíněný výše či alternativní postupy projevily jako problematické, např. kvůli selhání pokusu vůbec hýbat rukou a podvozkem najednou, mohu se vrátit k jednoduššímu řešení uvedenému na začátku této podsekce.



Obrázek 3.3: Způsob složení pohybu ruky a podvozku v zamýšlený výsledný pohyb. Ruka se pohybuje pouze v ose x a podvozek pouze v ose y. Ruka i podvozek urazí každý vzdálenost rovou poloměru kružnice (od osy otáčení ke klice dveří).

3.5 Návrh postupu řešení

Nové postupy a poznatky není moudré rovnou zkoušet na opravdovém robotovi a dveřích. Mnohem bezpečnější a úspornější je, jak jsem v sekci 2.3 naznačil, zkoušet je nejdříve v simulátoru Gazebo. Možnosti MoveIt!u a tf jsou rozsáhlé a je potřeba testovat, kam až sahají. Pro Gazebo jsem stáhl a pro své potřeby upravil model dveří – stačí vytvořit spouštěcí soubor spouštějící Gazebo s modelem PR2 a dveří ve správné poloze a můžu začít testovat uzel.

Poté, co budu mít hotový uzel, který umožní robotovi aspoň nějaké dveře otevřít, mohu se přesunout k reálnému robotovi. Nejdříve budu ale používat falešné dveře vyrobené z lepenky. Je to pro PR2 bezpečnější – kdyby se mu musely mezi rámem opravdových dveří nouzově vypnout motory, bylo by pracné ho z rámu odtlačit. Dveře z lepenky také umožňují narozdíl od simulace testovat i UpD a jeho spolupráci s UpO prostřednictvím AR kódu nalepeného na dveřích.

Až bude řešení uspokojivé a fáze řešení pokročí, začnu uzel testovat i na reálných dveřích. Pokud bude do této fáze implementováno pouze zmáčknutí kliky a odstrčení dveří, v této fázi se to změní – dopíšu i část programu pro projetí dveřmi a jejich postupné odstrkování. Jakmile bude spolehlivost dostatečně velká, začnu se dle časových možností věnovat dalším cílům, jako rozšíření repertoáru dveří, kontrola na zamčené dveře, otevírání dveří směrem k sobě a vyrovnávání se s dalšími situacemi, které jsou horší, než ideální.

Kapitola 4

Realizace

UpO byl úspěšně implementován a spolupracuje s UpD na otevření dveří. Narazil jsem však na několik problémů, kvůli kterým jsem své cíle musel uskromnit. Ty jsou probrány v sekci 4.1. V sekci 4.2 popisuji kód výsledného programu a jeho průběh při spuštění, stejně jako komunikaci mezi UpD a UpO. Výsledný program jsem otestoval – průběh a výsledky testování uvádím v sekci 4.3.

4.1 Obecný průběh realizace

Vytvořil jsem spouštěcí soubor, který spouští Gazebo s připraveným robotem a dveřmi před ním. Kromě toho je ještě třeba spustit uzel `move_group`, který se nachází v balíčku nastavení MoveIt!u pro PR2 (`pr2_moveit_config`). Nyní jsem mohl zkoušet práci s MoveIt!em a vhodnost jeho různých metod k dosažení různých cílů, abych našel tu nejvhodnější.

Vhodnost plánování v kartézkém prostoru

Jedním z nejzákladnějších úkonů z hlediska práce s programátorským rozhraním uzlu (či třídy) `move_group` je plánování do cílové pózy. Metodou pokus-omyl jsem nastavil ruku simulovaného robota kolem kliky dveří a jednoduchým pohybem dolů ji stiskl.

Pro takové jemné pohyby se však podle popisu hodí spíše plánování v kartézkém prostoru, tak jsem ho zkusil. Při stisknutí ruky však plánoval nesmyslnou trajektorii ruky a, jak jsem později zjistil, nerespektuje omezení orientace a polohy, která nastavím s nadějí, že to trajektorii zlepší. Toto je známý problém. Byl jsem tedy nucen používat pouze plánování v kloubovém prostoru a do cílové pózy. V tomto případě ani nebylo omezení pro orientaci a polohu ruky třeba.

Detekce kolizí

Veliký problém nastal, měl-li robot rukou, kterou tiskne kliku dolů, zatlačit proti dveřím. Plánovač takový pohyb nedokázal naplánovat. Přitom byl cíl v dosahu ruky robota. Zjistil jsem, že uzel `move_group` přijímá mračna bodů ze stereo kamery (či z Kinectu) a převádí je na vnitřní reprezentaci volného a obsazeného prostoru – oktomapy. Plánovač potom nedovolí žádné kolize robota s oktomapou. Chycení a stlačení kliky, přestože to kolize je, bylo možné proto, že robot na kliku přes své chapadlo neviděl, takže místo kolem ruky považoval za prázdné.

Nenašel jsem žádný způsob, jak kolize určité části robota s oktomapou povolit, kromě funkcí `pick()` a `place()` uzlu `move_group`. Použití těchto funkcí však shledávám příliš složitým – ani po značném úsilí se mi nepodařilo s pomocí funkce `pick()` chytit kliku. Potřebuje totiž, aby byl vytvořen virtuální objekt, který nahradí oktomapu. I manuální pokusy o chycení a pohnutí s tímto objektem selhaly. Plánovač stále detekoval kolize.

Jediný způsob, jak bych mohl pokračovat, je tedy vypnout kolize s prostředím úplně. Toto samo o sobě provést nelze, nicméně jde změnit konfigurace balíčku `pr2_moveit_config` tak, aby uzel `move_group` přijímal mračna bodů ze špatného (neexistujícího) tématu. Výsledný efekt je stejný: Robot nebude kolidovat sám se sebou, protože si udržuje vnitřní reprezentaci své polohy, ale bude kolidovat s prostředím, protože uzel `move_group` žádné nevidí.

Místo změny původního balíčku jsem ho zkopíroval, změnil výše popsaným způsobem a přejmenoval na `blind_pr2_moveit_config`. Školní PR2 má však ještě na pravé ruce F/T senzor a vyžaduje svoji vlastní konfiguraci. Na ni jsem tedy aplikoval obdobný proces a vytvořil balíček `blind_but_pr2_moveit_config`. Uzel `move_group` se nyní podle situace musí spouštět z jednoho z těchto dvou balíčků.

Dveře z lepenky a tf rámce

Vyhotovil jsem dveře s klikou v životní velikosti z lepenky a začal zkoušet svou implementaci na nich. Byla však primitivní – chtěl jsem ji udělat flexibilnější, aby robot dokázal chytit dveře, o kterých mu je sdělena jejich poloha a vlastnosti, a otáčet klikou a dveřmi po kruhové dráze. Použití tf rámců začalo být podmínkou. Došel jsem k závěru, že budu potřebovat následující tf rámce:

1. `doors`: Potomek rámce `odom_combined`; značí nejspodnější část osy otáčení dveří.
2. `handle_axis`: Potomek rámce `doors`; značí osu otáčení kliky dveří. Je umístěn tak, aby rámec `handle_grip` oproti němu mohl být posunut jen v souřadnici y.
3. `handle_grip`: Potomek rámce `handle_axis`; značí místo na klice, kde by robot měl kliku chytit.

K vysílání těchto rámců bylo potřeba vytvořit nový uzel (dále popsán v sekci 4.2). Příkaz k jeho spuštění jsem přidal do spouštěcího souboru, takže se automaticky spouští zároveň s hlavním uzlem. Polohu rámců jsem nastavoval podle simulovaných dveří (a údajů ze simulátoru). Vzhledem k tomu, že dveře z lepenky jsem stavěl podle simulovaných dveří, tak jediný rozdíl ve vysílání rámců byl, že rodič rámce `doors` byl `base_footprint`, takže poloha dveří byla definována odchylkou od polohy robota, nikoliv počátku světa.

Dále jsem pracoval na třídě Circle, která mi pomůže vypočítat bod na kružnici definované středem a poloměrem na základě dodaného úhlu. Ve výsledku dokáže robot otočit klikou a následně dveřmi s inteligentnější trajektorií ruky a otočení je parametrizováno úhlem, o který má otočení proběhnout.

Reálné dveře

Po přeměření reálných dveří a implementaci naměřených hodnot do vysílače tf rámce dokázal robot stejně dobře otevřít i reálné dveře. Můj program ale ještě nezahrnoval ježdění. Kromě toho jsem při každém pokusu o otevření musel změřit relativní polohu dveří vůči robotovi a tyto údaje vložit do kódu tf vysílače. Tento problém ale byl odstraněn po tom,

co začala být funkční komunikace mezi UpD a UpO. Nyní jsem tedy potřeboval, aby robot dokázal dojet na souřadnice v rámci, který mu zadám a aby se za nimi uměl otočit. Tuto funkčnost sice nabízí balíček ROSu `pr2_navigation`[1], ten je ale pro mé potřeby zbytečně složitý a navíc se vyskytly problémy při práci s ním na fakultní PR2. Raději jsem si tuto fukčnost s pomocí tutoriálů ROSu implementoval sám; více podrobností uvádím v sekci 4.2.

Spolupráce uzlu pro detekci a uzlu pro otevření

S pomocí mnou navrženého a jasně definovaného protokolu se mně a Rolandovi podařilo zajistit správnou komunikaci mezi UpD a UpO. UpO dokáže zpracovávat zprávy přijaté z UpD a poskytovat zpětnou vazbu tak, jak je to uvedeno dále v sekci 4.2.

Z časových důvodů jsem však už nestihl implementovat složitější způsob otevírání dveří, kdy robot po celou dobu projíždění dveřmi má ruku na klice, jak je uvedeno v sekci 3.4. Robot dveře odtlačí jen trochu – bez pohnutí základnou – poté ji z kliky sundá, nastaví před sebe a proti dveřím se rozjede.

4.2 Implementace

Tato sekce bude pojednávat o zdrojovém kódu a jeho rozmístění v souborech výsledného uzlu, průběhu spuštění uzlu a testování uzlu.

Komunikace mezi uzlem pro detekci a uzlem pro otevření

Uzly komunikují dvěma konceptuálně odlišnými způsoby: tf rámci a zprávami zasílanými na témata ROSu.

Vysílané tf rámce

1. `ar_code`: Potomek rámce `odom_combined`; značí prostředek AR kódu, který naskenoval UpD.
2. `doors`: Potomek rámce `ar_code`; značí nejspodnější část osy otáčení dveří.
3. `handle_axis`: Potomek rámce `doors`; značí osu otáčení kliky dveří. Je umístěn tak, aby rámec `handle_grip` oproti němu mohl být posunut jen v souřadnici y.
4. `handle_grip`: Potomek rámce `handle_axis`; značí místo na klice, kde by robot měl kliku chytit.

Zprávy ROSu

Definice zpráv je popsána jednoduchým jazykem pro popis zpráv[2]. Nachází se v souborech s příponou `.msg`, které jsou uloženy ve složce `msg` ve složce balíčku.

Prvním souborem je `Doors.msg`; zprávu, která je v něm popsaná, zasílá UpD na téma `open_doors/doors`. Její zaslání pro UpO znamená:

- UpD má o dveřích všechny potřebné informace.

[1] http://wiki.ros.org/pr2_navigation
[2] http://wiki.ros.org/msg

- UpD začal úspěšně vysílat TF rámce.
- UpD se vzdává kontroly nad robotem až do okamžiku projetí robota dveřmi.
- UpO může začít otevírat dveře.

Ve struktuře zprávy jsem se inspiroval řešením Willow Garage. Zpráva obsahuje také pojmenované konstanty, zde psané velkými písmeny. Obsah samotné zprávy je zde uveden ve formátu:

`datový_typ identifikátor`: Popis proměnné.

- `float32 width`: Šířka dveří v metrech.
- `float32 height`: Výška dveří v metrech.
- `int32 latch_state`: Stav dveří; může nabývat hodnot `LOCKED` (zamčené), `LATCHED` (zavřené) a `UNLATCHED` (otevřené). UpO umí zatím otevřít jen zavřené dveře. Ostatní hodnoty tam jsou jako ambice do budoucna.
- `float32 handle_rot_angle`: Úhel v radiánech, o který robot musí otočit klikou po směru hodinových ručiček, aby je otevřel.
- `int32 rot_dir`: Směr, kterým se dveře mají otáčet. Může nabýt hodnot `CLOCKWISE` a `COUNTERCLOCKWISE`. Opět spíše ambice do budoucna.
- `int32 push_pull`: Značí, za má robot zatlačit nebo zatáhnout za dveře. Nabývá hodnot `PUSH` a `PULL`. Také ambice do budoucna.

Popis zprávy v souboru `Opening_progress.msg` obsahuje jen jednu celočíselnou proměnnou `progress`, která může nabývat čtyř různých hodnot (opět konstant):

- `BAD_INFO`: Poskytnuté informace jsou nedostatečné nebo neplatné.
- `STARTING`: UpO dostal veškeré potřebné informace a začíná proceduru otevírání dveří.
- `SUCCESS`: Dveře jsou otevřené a robot jimi projel. UpO se vzdává kontroly nad robotem.
- `FAILURE`: V průběhu otevírání dveří se vyskytla chyba. UpO se vzdává kontroly nad robotem.

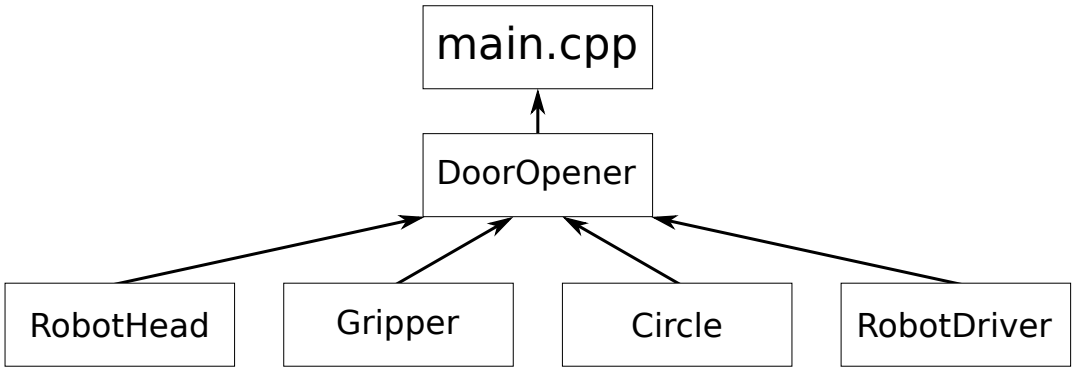
UpO tedy před začátkem otevírání (po přijetí zprávy Doors) pošle zprávu na téma `open_doors/progress` s hodnotou `BAD_INFO` nebo `STARTING`. Pokud otevírání započal, na konci vykonávání své činnosti pošle `SUCCESS` nebo `FAILURE`. Mezi začátkem a koncem si vyhrazuje právo na kontrolu robota (nikdo jiný ho kontrolovat nebude). UpO v průběhu také sám nastaví robota do potřebné polohy proti dveřím.

Popis tříd a souborů

Pokud nebude uvedeno jinak, předpokládejte, že každá třída má svůj hlavičkový soubor ve složce `složka_balíčku/include/pr2_open_doors/` a soubor s implementací ve složce `složka_balíčku/src/`.

Všechny hlavičkové soubory jsou komentované pro program Doxygen; dokumentaci lze vygenerovat pomocí příkazu `doxygen doxy.cfg` ve složce balíčku. Výsledek se bude nalézat ve složce `doc`.

Diagram, který zobrazuje, která mnou vytvořená třída využívá kterou, je na obrázku 4.1.



Obrázek 4.1: Diagram znázorňující, která třída instanciuje a využívá pro práci kterou. Pozor, nejedná se o graf dědičnosti. Zobrazuje pouze mnou vytvořené třídy.

Vysílač tf rámců

Je implementován pouze v souboru `doors_tf_broadcaster.cpp`. Vysílání tf rámců by měla být práce UpD, ale tuto funkčnost jsem potřeboval již v rané fázi vývoje UpO. Proto jsem vytvořil vlastní vysílač, který nyní umožňuje samostatné fungování UpO za předpokladu, že se vysílači předají správné argumenty.

Kód vysílače je relativně jednoduchý, neboť vysílat tf rámce není složitá záležitost. Nastavení všech rámců je však v kódu ve třech variantách. Volba varianty je zajištěna globální proměnnou `doors`, která nabývá jedné z hodnot výčtového typu, které značí, zda má vysílač vysílat rámce pro simulované dveře, lepenkové dveře, nebo reálné dveře. Hodnota této proměnné se dá, jak jsem zmínil, změnit pomocí parametrů příkazového řádku. Mají-li se otevřít reálné dveře, dají se pomocí parametrů předat také souřadnice a rotace polohy dveří (rámce `doors`) vůči robotovi (rámci `base_footprint`). Tyto parametry se ale v praxi předají spouštěcímu souboru, který je dále předá vysílači.

Kromě vysílání tf rámců vysílač také pošle jednu zprávu na téma `open_doors/doors`, čímž spustí proces otevírání dveří v hlavním uzlu.

Circle

Velmi jednoduchá třída, která má pomoci s počítáním bodu na kružnici podle zadaného úhlu. Kružnice je definovaná středem a poloměrem a toto jsou také parametry, kterými se nový objekt této třídy inicializuje.

RobotHead

Implementuje funkci `lookAt()`, která otočí hlavu směrem k libovolnému bodu v daném tf rámci, která byla převzata z tutoriálu na stránkách ROSu[3]. Tato funkce ve skutečnosti jen dá dohromady a pošle žádost akčnímu serveru, který hlavu ovládá.

Gripper

Implementuje funkce pro otevření a zavření chapadla podle tutoriálu ROSu[4]. Při instanciaci je zvoleno buď levé, nebo pravé chapadlo. Opět vlastně jen posílají žádost akčním serverům.

RobotDriver

Tato třída umožňuje ovládání základny robota pomocí čtyř funkcí. Funkce `turnOdom()` a `driveOmnidirOdom()` jsou inspirované tutoriálem na stránkách ROSu[5]. První funkce zajišťuje otáčení základny o daný úhel, druhá pohyb základny o danou vzdálenost libovolným směrem. Čím více se u obou funkcí robot blíží k cíli, tím více zpomaluje v pohybu, aby dosáhl větší přesnosti.

Dále funkce `faceTarget()` otočí robotem tak, aby čelil zadaným souřadnicím v daném rámci. Nejdříve spočítá úhel mezi aktuálním směrem kupředu a směrem k cílovému bodu, poté volá funkci `turnOdom()`.

Poslední funkce `driveToTarget()` doveze robota přímou trasou na dané souřadnice v daném rámci. Pokud je cíl blíže než půl metru, dojede tam robot bez otáčení díky funkci `driveOmnidirOdom()`. V opačném případě se robot nejdříve otočí směrem k cíli pomocí `faceTarget()` a potom se rozjede kupředu.

Hlavní soubor

Soubor `main.cpp` moc kódu neobsahuje. Provádí akorát standardní inicializaci uzlu ROSu, instanciuje třídu DoorOpener, přihlásí se k odběru z tématu `open_doors/doors` a jako callback při přijetí zprávy uvede funkci `open_doors_callback()` ze třídy DoorOpener. Nakonec bude čekat, dokud nepřijme zprávu nebo dokud nebude vypnut. Po přijetí zprávy je zavolána callback funkce, ta vykoná otevření dveří a vrátí se zpět do hlavního programu, kde se bude znovu čekat.

DoorOpener

Nejdůležitější třída celého uzlu. Řídí celý proces otevírání dveří a kontroly přijaté zprávy, která je přijata funkcí `open_doors_callback()`. Při své instanciaci instanciuje mj. také třídy uvedené na obrázku 4.1 a rozhraní pro ovládání rukou robota. Popis procesu otevření dveří je uveden v následující podsekci (4.2).

Průběh programu

Nejprve proběhne v hlavním souboru inicializace uzlu a objektu třídy DoorOpener, jak je uvedeno v předchozí podsekci (4.2). Při přijetí zprávy je volán `open_doors_callback()`

[3] http://wiki.ros.org/pr2_controllers/Tutorials/Moving%20the%20Head

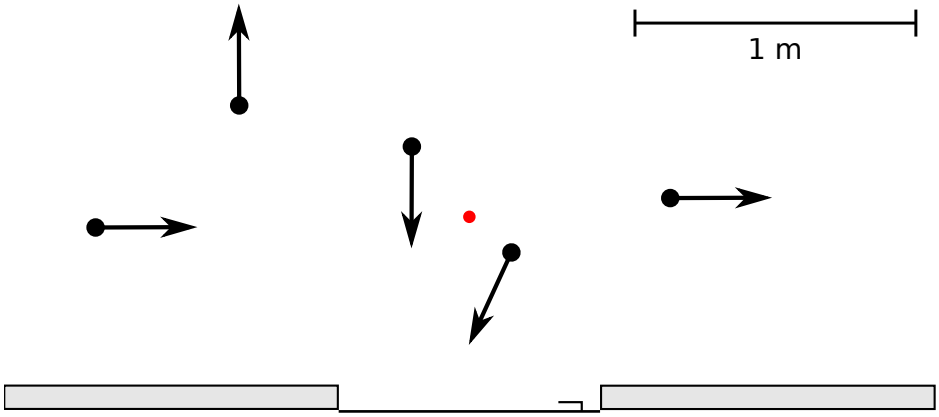
[4] http://wiki.ros.org/pr2_controllers/Tutorials/Moving%20the%20gripper

[5] http://wiki.ros.org/pr2_controllers/Tutorials/Using%20the%20base%20controller%20with%20odometry%20and%20transform%20information

Obrázek 4.2: Průběh otevírání dveří.

a začíná proces otevírání dveří. Jeho prvním krokem je kontrola správných údajů v přijaté zprávě. Na základě výsledku pošle UpO zprávu na téma `open_doors/progress` s hodnotou buď `BAD_INFO`, nebo `STARTING`, je-li zpráva v pořádku.

Pokud je, pokračuje se otevřením levého chapadla, otočením hlavy směrem ke klice dveří (tf rámce se již vysílají z UpD) a nastavením rukou do iniciální polohy. Pravá ruka není při procesu vůbec používaná a je sbalená do předpřipravené polohy zadané jako cíl



Obrázek 4.3: Ilustrace počátečních stavů při pokusech v testu. Černé body značí počáteční pozici robota, šipky značí jeho počáteční orientaci a červený bod značí optimální polohu vůči dveřím, na kterou se robot vždy snaží dojet.

v kloubovém prostoru. Poté je do cílové pózy nastavena levá ruka. Po každém úkonu rukou je kontrolována úspěšnost dokončení pohybu. V případě neúspěchu se proces přeruší a vrací se zpět do hlavní funkce, kde čeká na další zprávu o dveřích.

Dále je robot navigován do optimální vzdálenosti před střed dveří a otočen směrem k němu. To vše s pomocí funkcí třídy RobotDriver. Proces otevření dveří, který bude popsán od této chvíle, je zobrazen na obrázku 4.2. Levá ruka je nastavena do pozice asi 20 cm před kliku, kolmo ke dveřím, poté navedena na kliku a chapadlo je zavřeno.

Před robotem nyní leží úkol stisknout kliku po kruhové dráze o daný úhel. K výpočtu cíle dráhy je použita třída Circle. Její výsledek je předán funkci pro plánování do cílové pózy a rukou je pohnuto. Obdobným způsobem je zatlačeno na dveře a následně klika vrácena do vodorovné polohy. Chapadlo se otevírá, ruka couvá od kliky a nastavuje se před robota, aby byla připravena pro odstrčení dveří.

Robot je již natočen směrem ke středu dveří, takže se stačí rozjet kupředu, čímž se odtlačí polootevřené dveře. Robot je nyní na opačné straně dveří; program hlásí úspěch zprávou `SUCCESS` a vrací se do smyčky, kde vyčkává na další zprávu.

4.3 Testování

Testoval jsem pouze samotný UpO ve spojení se svým vysílačem tf rámců, kterému jsem vždy zadal správnou polohu dveří. Zaměřoval jsem se pouze na úspěšnost dojetí na optimální polohu vůči dveřím a otevření dveří z této polohy. Všechny testované počáteční stavy jsou ilustrovány na obrázku 4.3.

Robot dokázal ve všech případech úspěšně dojet na cílové místo, z tohoto místa dveře otevřít a projet jimi. Komplikací ovšem jsou časté pády uzlu `move_group`. Knihovna OMPL, kterou `move_group` používá, se i při validních požadavcích často dopouští chyby segmentace a uzel je následně ukončen. UpO po chvíli čekání na odezvu nahlásí selhání a přeruší proces otevírání dveří. Žádné řešení tohoto problému jsem nenašel; je potřeba pokusy o otevření zkoušet, dokud se nepodaří dveře otevřít, jelikož uzel `move_group` ne při každém pokusu spadne.

Je také třeba mít na paměti, že robot ignoruje veškeré překážky při cestě na cílové místo a následném otočení. Maximální rychlost jízdy, kterou robotovi můj program dovoluje, je

15 cm/s a maximální rychlost otáčení je 0,75 rad/s, což je hluboko pod schopnostmi robota. Doprava robota na cílové místo je trochu zdlouhavá, také proto, že postupně s blízkostí cíle zpomaluje. Toto jsem však implementoval proto, aby robot dosáhl co nejvyšší přesnosti.

Je-li robot v optimální poloze přede dveřmi a otočený čelem k nim, trvá mu otevření a projetí dveřmi přibližně 54 sekund. Bylo by možné zlepšit tento čas i čas dojetí ke dveřím a otočení, nicméně toto považuji za záležitost, která je nad rámec této práce.

Kapitola 5

Závěr

Cílem této práce bylo vytvořit uzel pro framework ROS a fakultního robota PR2, který robotovi umožní na základě dat přijatých ze senzorů a zpracovaných uzlem, který byl přemětem jiné bakalářské práce [1], otevřít dveře od robotické laboratoře směrem od sebe.

Tento cíl byl splněn a řešení bylo v tomto dokumentu dostatečně vysvětleno, nicméně nepodařilo se splnit pokročilé cíle, jako projetí dveřmi, zatímco má robot celou dobu chapadlo na klice, nebo zavření dveří za robotem. Detekce kolizí robota s prostředím musela být pro zjednodušení vypnuta. Robot zvládá otevření dveří směrem od sebe, s klikou na levé straně. Při tvorbě uzlu bylo myšleno na budoucí vývoj a program je jednoduše rozšiřitelný, aby dovolil otevírání dveří s klikou na pravé straně. Otevírání dveří směrem k sobě by však vyžadovalo rozdílný postup a vyžadovalo by velké zásahy do programu.

Můj uzel není závislý na konkrétním uzlu, který posílá údaje o poloze a vlastnostech dveří, dokud splňuje komunikační protokol. Ten byl navržen s ohledem na budoucí vývoj, takže by nebylo nutné ho hned při dalším vývoji přepracovávat. Oba uzly jsou dva nezávislé celky, které se mohou samostaně vyvíjet či vyměnit.

Veškeré výše uvedené nesplněné pokročilé cíle, stejně jako celkové zvýšení rychlosti dojetí do cíle a otevření dveří, můžou být předmětem dalšího vývoje a možná také mé diplomové práce. Dosáhnu-li při budoucím vývoji uzlu dostatečné kvality, zveřejním ho na webu ROSu.

Literatura

[1] Botka, R.: *Modul pro detekci dveří pro PR2*. Bakalářská práce, FIT VUT v Brně, 2015.

[2] Chitta, S., Cohen, B., Likhachev, M.: Planning for autonomous door opening with a mobile manipulator. In *IEEE International Conference on Robotics and Automation*, IEEE, 2010, ISBN 978-1-4244-5038-1, ISSN 1050-4729, s. 1799 – 1806.

[3] Chitta, S., Sucan, I., Cousins, S.: MoveIt! *IEEE Robotics & Automation Magazine*, ročník 19, č. 1, 3 2012: s. 18 – 19, ISSN 1070-9932.

[4] Foote, T.: tf: The transform library. In *IEEE International Conference on Technologies for Practical Robot Applications (TePRA)*, 4 2013, ISSN 2325-0526, s. 1 – 6.

[5] Meeussen, W., et al.: Autonomous Door Opening and Plugging In with a Personal Robot. In *IEEE International Conference on Robotics and Automation*, IEEE, 2010, ISBN 978-1-4244-5038-1, ISSN 1050-4729, s. 729 – 736.

[6] Nagatani, K., Yuta, S.: An experiment on opening-door-behavior by an autonomous mobile robot with a manipulator. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 1995, ISBN 0-8186-7108-4, s. 45 – 50.

[7] Niemeyer, G., Slotine, J.-J. E.: A Simple Strategy for Opening an Unknown Door. In *IEEE International Conference on Robotics and Automation*, IEEE, 1997, ISBN 0-7803-3612-7, s. 1448 – 1453.

[8] Orság, F.: *Robotika – studijní opora*. FIT VUT v Brně, 2006.

[9] WWW stránky: About ROS [online]. http://www.ros.org/about-ros/, [cit. 2015-02-02].

[10] WWW stránky: Concepts | MoveIt! [online]. http://moveit.ros.org/documentation/concepts/, [cit. 2015-02-03].

[11] WWW stránky: Gazebo [online]. http://gazebosim.org/, [cit. 2015-02-02].

[12] WWW stránky: Hardware specs | Willow Garage [online]. http://www.willowgarage.com/pages/pr2/specs, [cit. 2015-03-14].

[13] WWW stránky: Move Group Interface/C++ API – pr2_moveit_tutorials documentation [online]. http://docs.ros.org/hydro/api/pr2_moveit_tutorials/html/planning/src/doc/move_group_interface_tutorial.html, [cit. 2015-02-05].

[14] WWW stránky: Nodes [online]. http://wiki.ros.org/Nodes/, [cit. 2015-02-02].

[15] WWW stránky: tf - ROS Wiki [online]. http://wiki.ros.org/tf, [cit. 2015-05-06].

Příloha A

Obsah CD

Složka blind_pr2_moveit_config

Obsahuje nakonfigurovaný uzel `move_group`. Konfigurace byla od původní verze upravena tak, aby uzel nedostával informace z Kinectu (či stereo kamery) a nevytvářel si tak mapu obsazenosti prostoru kolem sebe, čímž se ve výsledku vypne detekce kolizí mezi robotem a prostředím. Tento uzel by měl být spouštěn při práci s robotem v simulátoru.

Složka blind_but_pr2_moveit_config

Oproti konfiguraci uzlu `move_group`, která je ve složce `blind_pr2_moveit_config`, je tato vytvořena pro fakultního robota PR2, který se od toho běžného liší tím, že má na pravé ruce F/T senzor, který mu ruku trochu prodlužuje. Uzel s tím musí při plánování pohybu počítat.

Složka models

Obsahuje modely dveří a cihlové stěny pro simulátor Gazebo.

Složka detect_doors

Obsahuje uzel Rolanda Botky, který vytvořil ve své bakalářské práci [1]. Je sem přidaný, aby se mohla demonstrovat schopnost spolupráce mého a Rolandova uzlu.

Složka pr2_open_doors

Obsahuje uzel, který byl předmětem této práce a který v ní je zevrubně popsán. Také obsahuje soubor s licencí.

Složka tex

Obsahuje zdrojové kódy a grafický materiál pro vytvoření tohoto dokumentu.

Soubor Detekce a otevření dveří pro PR2.avi

Toto video popisuje a demonstruje výsledek práce Rolanda Botky a mě.

Soubor Modul pro otevření dveří pro PR2.pdf

Tento dokument.

Soubor README.txt

Obsahuje návody, jak nainstalovat veškerý software, který je prerekvizitou pro spuštění uzlů `detect_doors` a `pr2_open_doors`, jak tyto uzly přeložit a jak je spustit v jejich různých variantách.