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

INTERAKTIVNÍ ROZHRANÍ PRO VZDÁLENÉHO ROBOTA PRO ANDROID

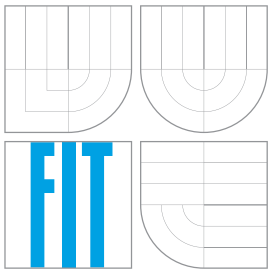
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. VOJTĚCH ROBOTKA

BRNO 2013

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

INTERAKTIVNÍ ROZHRANÍ PRO VZDÁLENÉHO ROBOTA PRO ANDROID

INTERACTIVE INTERFACE FOR ROBOT REMOTE CONTROL FOR ANDROID

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. VOJTĚCH ROBOTKA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MICHAL ŠPANĚL, Ph.D.

BRNO 2013

Abstrakt

Vývoj autonomních robotů pokročil a začínají se objevovat první možnosti nasazení robotů v běžném životě. Spolu s vývojem robotů je však třeba myslet i na vývoj aplikací a uživatelských rozhraní, která budou s roboty vzdáleně interagovat. Cílem tohoto projektu je vytvořit univerzální rozhraní pro vzdálené řízení robotů. Práce vychází z předpokladu použití platformy ROS na ovládaných robotech. Díky tomu se nabízí širší možnost uplatnění výsledné aplikace na dalších robotických projektech založených na této platformě. Navržené vzdálené rozhraní plní dva základní účely pro ovládání robota. Za prvé, umožňuje uživateli zobrazit důležitá data pro orientaci v aktuální situaci robota a tyto data zasadit do kontextu ve 3D scéně. V druhé řadě pak umožňuje vykonávat základní manipulaci s robotem pomocí ovládacích prvků. Výsledná aplikace byla přizůsobena a úspěšně otestována na experimentálních robotech Care-O-Bot a Turtlebot.

Abstract

The development of autonomous robots has made a significant progress and first personal robots for common use start to appear. To use this robots, we need to develop applications and user interfaces to interact with them. The goal of this project is to make a universal interface for robot remote control. This work focuses on robots based on the ROS platform. This gives the final application a potential of use on other robotic projetcs running on ROS. The designed remote interface accomplishes two main purposes. The first is to show important data in a context of a 3D scene to help user understand the state of the controlled robot. And the second goal is to allow the user execute some basic manipulation with the robot. The final application was successfully adapted and tested on experimental robots Care-O-Bot and Turtlebot.

Klíčová slova

robot, vzdálené, řízení, Android, ROS, UI, SRS, Turtlebot, Care-O-Bot

Keywords

robot, remote, control, Android, ROS, UI, SRS, Turtlebot, Care-O-Bot

Citace

Vojtěch Robotka: Interaktivní rozhraní pro vzdáleného
robota pro Android, diplomová práce, Brno, FIT VUT v Brně, 2013

Interaktivní rozhraní pro vzdáleného robota pro Android

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Michala Španěla, PhD.

.....
Vojtěch Robotka
29. května 2013

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Michalu Španělovi, Ph.D. za příkladné vedení a odbornou pomoc při práci na projektu.

Obsah

1 Úvod **4**

2 Robotické systémy a problematika vzdáleného ovládání **5**
2.1 Míra autonomie robota . . . 5
2.2 Mobilní zařízení . . . 5
2.3 Mobilní platformy . . . 6
2.4 Uživatelské rozhraní . . . 6
2.5 3D zobrazení . . . 7
2.6 Prolínání zobrazených dat . . . 10

3 Robotický operační systém (ROS) **12**
3.1 Infratsruktura a komunikace . . . 12
3.2 Rosjava . . . 13
3.3 Simulátor Gazebo . . . 13

4 Využívané experimentální robotické platformy **14**
4.1 Care-O-Bot . . . 14
4.2 TurtleBot . . . 15

5 Projekt SRS **17**
5.1 Popis cílových skupin . . . 17
5.2 Srování s navrhovanou aplikací . . . 18

6 Návrh aplikace pro vzdálené řízení robota **20**
6.1 Univerzální rozhraní . . . 20
6.2 Způsob řízení robota . . . 21
6.3 Uživatelské rozhraní . . . 21
6.4 3D scéna . . . 23

7 Výpočet mapování textury na model **26**

8 Implementace **30**
8.1 Použité vývojové prostředky . . . 30
8.2 Zobrazení 3D scény . . . 31
8.3 Další prvky uživatelského rozhraní . . . 33
8.4 Komunikace . . . 34

9 Dosažené výsledky **36**
9.1 Care-O-Bot 36
9.2 Turtlebot 39
9.3 Nastavení aplikace 41

10 Závěr **44**

A Použití aplikace **47**
A.1 Instalace ROSu 47
A.2 Nastavení proměnných 47
A.3 Spuštění simulace 48
A.4 Spuštění aplikace 48

Seznam obrázků

2.1 Varianty rozložení prvků (zdroj [9]) 8
2.2 Srovnání 3D a 2D rozhraní z experimentu (zdroj [11]) 9
2.3 Ukázka výpočtu hloubkové mapy z několika snímků (zdroj [16]) 11
2.4 Ukázka mapování textury na model (zdroj [14]) 11
2.5 Ukázka mapování (projekce) textury na model (zdroj [10]) 11

3.1 Ukázka simulace v gazebu 13

4.1 Care-O-Bot 3 z Fraunhofer IPA (zdroj [4]) 14
4.2 Nákres komponent robota Care-O-Bot 3 (zdroj [4]) 15
4.3 Technická specifikace robota Care-O-Bot 3 (zdroj [4]) 16
4.4 Schema robota TurtleBot 16

6.1 Koncept výpočtu projekce textury na model 25

7.1 Výpočet projekce textury na model prostředí 27
7.2 Ukázka výpočtu souřadnic textury ve vertex shaderu 29

8.1 Ukázka části souboru ve formátu `world` 32
8.2 Ukázka vykreslení modelu robota 33
8.3 Ukázka vykreslených dat z laserového skeneru 34

9.1 URDF model prostředí a nepřesnost zobrazení vůči mračnu bodů. 37
9.2 Zjednodušený model prostředí a korekce pozice vůči mračnu bodů. 37
9.3 Ukázka modelu robota. 38
9.4 Ukázka z testování aplikace - panel příkazů, manipulátor kamery a joystick. 39
9.5 Robot Turtlebut, použitý pro účely testování aplikace 40
9.6 Ukázka z testování aplikace na Turtlebotovi. 41
9.7 Ukázka dialogu připojení po spuštění aplikace. 42
9.8 Ukázka nastavení aplikace. 42
9.9 Ukázka volby zobrazení jednotlivých prvků. 43

Kapitola 1

Úvod

I když jsou dnešní roboti do určité míry autonomní, velká většina z nich je závislá na pokynech od uživatele, ať už ve formě několika tlačítek, například u robotických vysavačů, nebo na komplexnějším systému ovládání. Ukazuje se, že právě problém vzdáleného ovládání složitějších robotů není triviální. Při návrhu takového ovládání je třeba zohlednit cílového uživatele aplikace (nezaškolený uživatel, pokročilý uživatel, technik), podle toho také nejčastější používané pokyny, ale hlavně způsob přiblížení situace robota uživateli. Tedy jaký zvolit způsob vizualizace jednotlivých dat charakterizujících aktuální prostředí a stav robota.

Jedním z pokročilých robotů, kteří jsou určeni pro použití v běžném prostředí je domácí robot Care-O-Bot. Robot byl vyvinut firmou IPA Fraunhofer ve Stuttgartu. V první fázi je však robot určen pro výzkum a vývoj potřebného software a v budoucnu pak pro přímé použití v domácím prostředí.

Dalším zajímavým a rozšířeným robotem je TurtleBot[1], využívaný převážně v akademické sféře pro výzkum nových technologií a algoritmů. Turtlebot vychází z komerčně úspěšného robotického vysavače Roomba od firmy iRobot.

Tato práce si klade za cíl porovnat aktuálně používané metody při návrhu uživatelských rozhraní pro vzdálené ovládání robotů a na základě nich navrhnout a implementovat vlastní řešení. V rámci návrhu aplikace se zaměříme v prvé řadě na obecné a univerzální řešení ovládání a v druhé řadě pak na vytvoření specifických rozhraní pro použití s roboty Care-O-Bot a Turtlebot.

Jako cílová platforma byl zvolen operační systém Android. Díky implementaci knihovny Rosjava je možné použít komunikační prostředky platformy ROS přímo v Android aplikaci. Tato kombinace se jeví jako ideální řešení pro vzdálené řízení robotů, díky možnosti použití na mobilních zařízeních a spojení s platformou ROS. Ta se stala v posledních letech nejčastěji využívanou platformou v oblasti vývoje robotů. Díky tomu by výsledná aplikace mohla sloužit pro ovládání nejen zmíněných robotů Care-O-Bot a Turtlebot, ale i mnoha dalších robotických projektů založených na platformě ROS.

V práci se nejdříve seznámíme s problematikou vzdáleného ovládání robotů v kapitole 2. V dalších kapitolách bude následovat přiblížení platformy ROS (kapitola 3), zmíněných robotů Care-O-Bot a Turtlebot využívaných při vývoji aplikace (kapitola 4) a souvislost s projektem SRS (kapitola 5). Po teoretickém úvodu přejdeme k samotnému návrhu aplikace v kapitole 6. Kapitola 7 se zabývá návrhem výpočtu mapování textury na model. K závěru práce bude diskutován průběh implementace projektu v kapitole 8 a přehled dosažených výsledků v kapitole 9.

Kapitola 2

Robotické systémy a problematika vzdáleného ovládání

V této kapitole se seznámíme s jednotlivými problémy a odlišnými přístupy při návrhu uživatelského rozhraní pro vzdálené ovládání.

2.1 Míra autonomie robota

I když je robot do značné míry autonomní, je třeba jeho uživatelům poskytnout širokou možnost kontroly. Ta by měla probíhat na různých úrovních řízení robota. Na vyšší úrovni řízení je i vysoká autonomie robota, ale tím jednodušší je kontrola ze strany uživatele. Například úkol "dones mléko"představuje pro robota velmi komplexní problém, zatímco z pohledu uživatele má jednoduché zadání. Naopak v případě manuálního řízení není vyžadována od robota vysoká míra inteligence, zatímco veškerá odpovědnost za pohyb je převedena na uživatele, který robota ovládá. Proto je vhodné rozdělit problém kontroly robota na více jednotlivých aplikací, lišících se právě v rozdělení odpovědnosti mezi robota a uživatele.

2.2 Mobilní zařízení

V posledních letech zaznamenal trh s tablety a tzv. chytrými telefony enormní růst. Spolu s rychlým rozvojem technologií se také rapidně mění parametry těchto zařízení. Každým rokem se zvyšuje výkon a možnosti telefonů a tabletů, zatímco jejich fyzické parametry se zmenšují.

Vzdálené řízení robotů většinou nevychází pouze z potřeby ovládat robota z jednoho konkrétního vzdáleného místa, ale naopak se často žádá, aby bylo možné ovládat jej odkudkoliv. Proto je důležitá přenositelnost daného zařízení. Moderní tablety svými parametry nabízí dokonalou mobilitu a zároveň se svým výkonem blíží klasickým osobním počítačům.

Volba mobilního zařízení také vychází z požadované specifikace pro vzdálené ovládání. Pro jednoduché pokyny dostačuje malá obrazovka s dotykovým ovládáním, zatímco na komplexnější řízení robota a potřebu orientace je třeba kromě větší obrazovky také pokročilejších ovládacích prvků (klávesnice, myši, apod.)

2.3 Mobilní platformy

Budeme-li tedy uvažovat vzdálené ovládání jako mobilní aplikaci, máme na výběr několik různých platforem. Z nejrozšířenějších na trhu jsou to především platformy Android, iOS a Windows mobile. Zatímco při vývoji komerční aplikace musíme při výběru platformy uvažovat cílovou skupinu a uživatelskou základnu jednotlivých platforem, v našem případě se tímto faktorem příliš řídit nemusíme. Předpokládáme totiž, že bude dané zařízení sloužit primárně k ovládání robota a bude tedy za tímto účelem pořízeno.

2.4 Uživatelské rozhraní

Pro pohodlné ovládání robota je důležitá přenositelnost uživatelského rozhraní, tedy možnost ovládat robota odkudkoliv. Zároveň je ale pro vykonávání některých náročnějších operací třeba zobrazení více typů dat a ovládacích prvků (například zobrazení 2D mapy, okolního prostředí robota, pohybového joysticku). V takových případech je pro pohodlné ovládání robota nutné větší zobrazovací zařízení, kde budou tyto prvky přehledně uskupeny. Mobilní telefon je tedy pro tyto účely poměrně nevhodný. Jako ideální zařízení se jeví dotykový tablet s větší úhlopříčkou obrazovky. Z tohoto předpokladu se bude vycházet v dalších úvahách týkajících se návrhu aplikace a uživatelského rozhraní.

2.4.1 Zobrazovaná data

Jednou z hlavních úvah při návrhu rozhraní pro ovládání robota je výběr prezentovaných dat. Ideální rozhraní obsahuje pouze taková data, která jsou uživateli užitečná k provádění definovaných úkonů a tato data jsou přehledně uspořádána dle jejich důležitosti. Rozhodnout, která data jsou právě ta užitečná však není jednoduchý úkol.

Pro efektivní řízení robota je pro uživatele klíčové zorientovat se v aktuálním prostředí okolo něj a správně pochopit jeho stav. To znamená, že uživatel potřebuje znát polohu robota, zda se v jeho okolí nachází nějaké překážky, nebo například jaká je pozice jeho manipulačního ramene. Tyto informace můžeme získat z těchto senzorů robota:

- kamery (reálný obraz prostředí)
- dálkoměry (laserové, ultrazvukové)
- senzor Kinect (hloubková mapa)
- pozice manipulačních serv (aktuální pozice jednotlivých částí)

Jedním z důležitých úkolů zpracování dat je určení pozice robota v prostoru. K tomu slouží senzory odometrie, které v pravidelném časovém intervalu udávají směr a rychlost pohybu robota.

2.4.2 Ovládací prvky

Některé definované úkony vyžadují pokročilejší ovládací prvky. Zatímco zadání bodu na mapě či výběr objektu k manipulaci lze realizovat jednoduchým dotykovým prvkem, řízení robota a kamery vyžaduje specifické vizuální komponenty. V případě řízení robota lze použít 4-směrový joystick pro pohyb dopředu/dozadu či otočení vpravo/vlevo. Pro ovládání kamery je nutné navíc kromě natočení kamery nahoru/dolů a vpravo/vlevo ještě přidat možnost přiblížení a oddálení záběru kamery.

Použití náročnějších vizualizačních prvků může také vyžadovat manipulaci s nimi. Pro zobrazení mapy stačí jednoduchý posun mapy tažením, ale při použití 3D scény je třeba zajistit rotaci, posun a případně přibližování 3D scény. V mobilních aplikací je manipulace s 3D scénou většinou realizována pomocí více-dotykových gest.

2.4.3 Rozložení prvků

Při návrhu uživatelského rozhraní je důležité zvolit správné rozmístění jednotlivých prvků. To by mělo vycházet z priority zobrazovaných dat a četnosti používání jednotlivých ovládacích prvků. Dále je vhodné zohlednit ergonomii ovládání, kdy se snažíme umístit ovládací prvky tak, aby byly jednoduše dostupné při běžné manipulaci s tabletem. Obecně lze říct, že je vhodné umístit ovládací prvky při kraji obrazovky, ideálně k pravému či levému kraji obrazovky. To vyplývá z jednoduchého pozorování - většina uživatelů drží tablet oběma rukama podél stran a ovládají tablet pomocí palců.

Dalším vhodným přístupem je sdružení zobrazovaných dat. Pokud chceme uživateli zobrazit například mapu s pozicí robota, můžeme do stejné mapy zobrazit například i data z dálkoměrů. Zároveň je možné mapu využít k určení cílové pozice pro navigaci robota. Tímto můžeme výrazně zvýšit přehlednost uživatelského rozhraní a zároveň umožnit uživateli vidět související data v kontextu a vytvořit si tak lepší představu o okolním prostředí robota.

Příkladem možných rozložení prvků jsou varianty dle [9]:

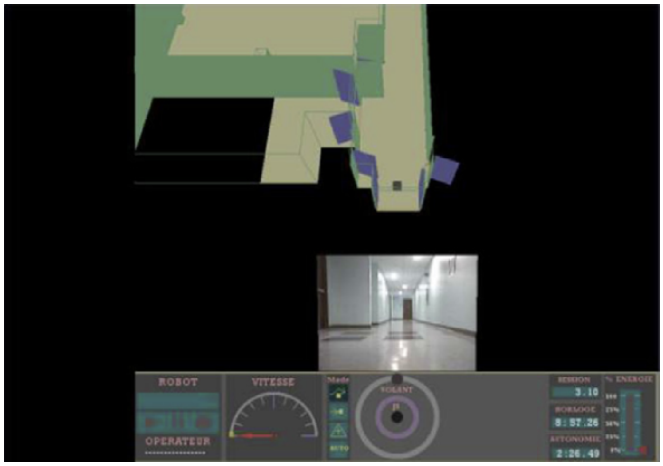
- Video-centrické zobrazení (obrázek 2.1a)
- Smíšená perspektiva s exo-centrickou mapou (obrázek 2.1b)
- Zobrazení rozšířené reality (obrázek 2.1c)

2.5 3D zobrazení

V literatuře se můžeme setkat s odlišnými způsoby zobrazení dat a jejich integrace. Zajímavým a v poslední době také často využívaným přístupem je integrace možných dat do jedné 3D scény [11]. Takový přístup má výhodu v tom, že uživatel vidí veškerá vizualizovaná data v kontextu a je pro něj tedy snažší se v prostoru okolo robota zorientovat. Dle [11] je vhodné v této scéně kombinovat dokonce i data, která jsou reprezentována v 2-dimenzionálním prostoru (například data z dálkových skenerů, 2D mapa prostoru, trajektorie navigace, atd.). Tím ale vzniká netriviální problém tato data začlenit do 3D scény tak, aby zobrazení co nejblíže odpovídalo realitě. Pokud by se ale podařilo většinu těchto dat zobrazit v kontextu a v jedné scéně, velkou mírou by se zvýšila přehlednost rozhraní odebráním samostatných oken se zobrazenými daty a tím by se také zvýšila schopnost uživatele se zorientovat v aktuální situaci robota.

Dle provedeného experimentu [11] se při zobrazní dat (2D mapa a obraz z kamery) v samostaných oknech podařilo pouze 43 % z testovaných uživatelů dokončit zadaný úkol manipulace s robotem. Oproti tomu při zobrazení dat v jedné 3D scéně uspělo 70 %. Zároveň uživatelé, kteří úkol dokončili byli v průměru o 34 % rychlejší než ti, kteří uspěli s 2D zobrazením. Testovaná rozhraní z experimentu jsou na obrázku 2.2 a výsledná data v tabulce 2.1.

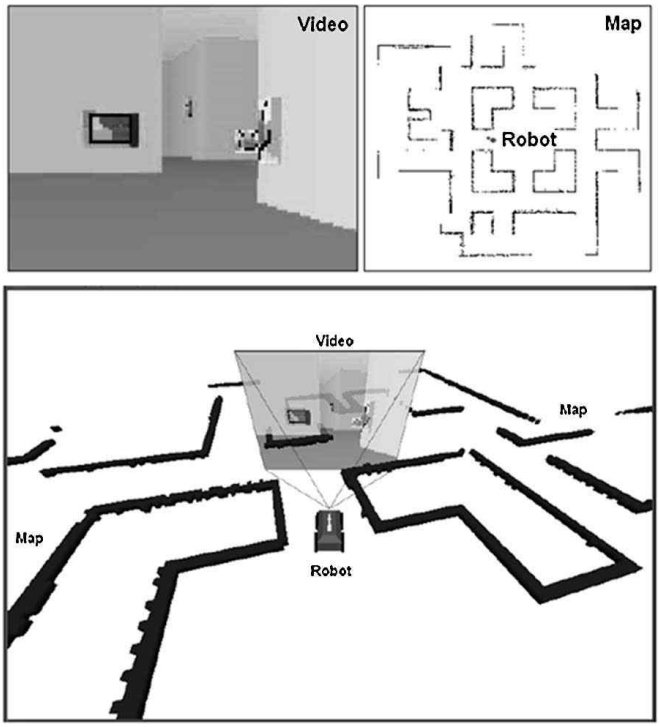
(a) Video-centrické zobrazení



(b) Smíšená perspektiva s mapou

(c) Zobrazení rozšířené reality

Obrázek 2.1: Varianty rozložení prvků (zdroj [9])



Obrázek 2.2: Srovnání 3D a 2D rozhraní z experimentu (zdroj [11])

	Čas splnění (průměr / sm. odchylka)	Kolize (průměr / sm. odchylka)
2D pouze mapa	258 / 57	9,8 / 7,8
2D mapa + video	271 / 55	8,5 / 4,6
2D pouze video	366 / 118	19,1 / 10,2
3D pouze mapa	196 / 28	1,3 / 2,2
3D mapa + video	208 / 34	1,3 / 1,8
3D pouze video	351 / 100	22,7 / 14,4

Tabulka 2.1: Výsledky experimentu srovnání 2D a 3D zobrazení (zdroj [11])

2.6 Prolínání zobrazených dat

V případě použití jednotné 3D scény pro zobrazení co největšího množství dat ze senzorů robota je nutné tato data vhodně transformovat do jednotného 3D prostoru. Zatímco zobrazení některých dat se zdá být poměrně přímočaré (například zobrazení 2D mapy v rovině pohybu robota, jako na obrázku 2.2), u jiných může být komplikované. Zároveň čím více typů dat se ve scéně kombinuje, tím je složitější srovnat souřadnice dat tak, aby na sebe navazovala a odpovídala realitě.

2.6.1 Model prostředí a mapa

Vzhledem k tomu, že robot má ze svých senzorů povědomí pouze o svém blízkém okolí, je často využíváno pro orientaci ve větším prostoru 2D mapy nebo 3D modelu prostředí. To jsou však statická data, která nemusí přesně odpovídat dynamickému prostředí v reálném použití. Tyto modely se tedy většinou používají jako základní vrstva, která je v případě překrytí s reálnými daty nahrazena nebo doplněna aktuálními daty.

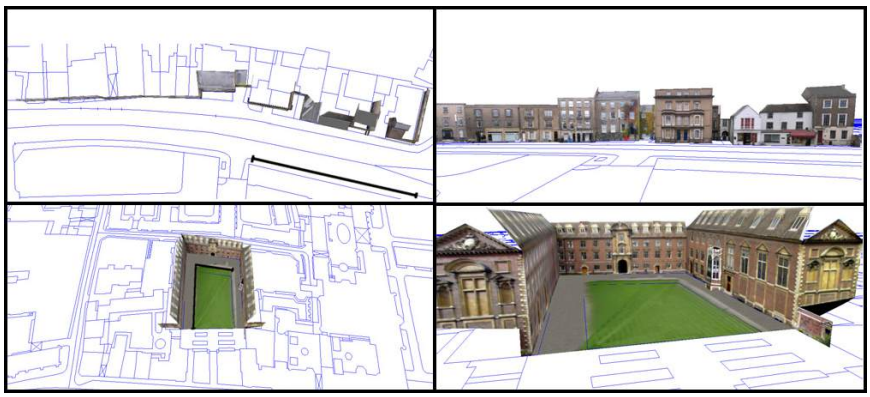
2.6.2 Zobrazení obrazu z kamery

Asi nejsložitějším úkolem je zobrazení obrazu z kamery robota. Zde se většinou používají zjednodušené přístupy transformace do 3D prostředí. Jedním je zobrazení 3D scény z pohledu kamery robota [9]. Díky tomu není nutné provádět transformace obrazu, ale pouze natočení 3D scény. Obrazem z kamery ja pak překryt správně transformovaný pohled na 3D scénu, jako na obrázku 2.1c. Dalším možným přítupem je zobrazení obrazu kamery do plochy umístěné v 3D prostoru, dle orientace robota (obrázek 2.2). Samotnou podstatou záznamu obrazu kamerou je však převod reálného 3D prostředí do 2D obrazu, pro samotnou vizualizaci tedy dává smysl pokusit se obraz zobrazit včetně prostorové informace. Tu však přijímaný obraz z kamery nemá. V odborné literatuře se setkáváme se třemi způsoby řešení tohoto nedostatku:

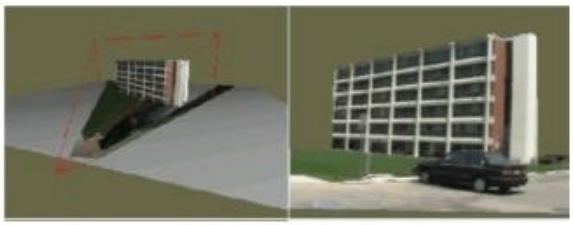
- Při záznamu obrazu snímat hloubkovou mapu (odděleným zařízením, nebo tzv. RGBD záznam - např. Microsoft Kinect)
- Výpočet hloubkové mapy z obrazu (zpravidla z více snímků z různé pozice)
- Mapování obrazu na 3D model

Jako nejvýhodnější způsob se jeví snímání obrazu včetně hloubkové mapy. Aktuální prostředky však v jistých situacích nedávají příliš přesné výsledky a i když je tento způsob v poslední době často využívaný, ne vždy je ideální. Druhý způsob je obdobný prvnímu s tím rozdílem, že hloubkovou mapu je třeba nejdříve vypočítat z obrazu [17] (obrázek 2.3). Tím mohou vznikat další chyby a nepřesnosti. Poslední metodou je mapování obrazu jako textury na 3D model prostředí ([10], [14]). Tento způsob vyžaduje pro dobré výsledky poměrně náročné výpočty a také co nejpřesnější model prostředí, proto se využívá jen ve spefických situacích (příklad na obrázku 2.4 nebo 2.5).

Obrázek 2.3: Ukázka výpočtu hloubkové mapy z několika snímků (zdroj [16])



Obrázek 2.4: Ukázka mapování textury na model (zdroj [14])



Obrázek 2.5: Ukázka mapování (projekce) textury na model (zdroj [10])

Kapitola 3

Robotický operační systém (ROS)

ROS[12] (Robot Operating System) je open-source platforma pro robotické systémy. I když se dle názvu jedná o operační systém, ve skutečnosti jde o sadu balíků a programů primárně určených pro OS Linux. Vzhledem k unikátním mechanismům komunikace v rámci jednotlivých balíků nebo komplexnímu systému překladu se ROS někdy také označuje jako meta-operační systém.

3.1 Infratsruktura a komunikace

Uzly

Uzly (*Nodes*) jsou základními prvky v infrastruktuře ROSu. Uzly jsou procesy provádějící výpočty či operace nad přijímanými daty nebo přímo ovládají činnost robota. Kumunikace mezi uzly je zajištěna datovými proudy v podobě témat (*Topics*), vzdáleným voláním služeb (*Services*), případně sdílením dat přes *Parameter Server*.

Témata

Témata jsou datové proudy s definovaným typem a strukturou přenášených zpráv (*Messages*). Ke každému tématu se může zaregistrovat libovolný počet uzlů jako odběratelů a stejně tak ke každému tématu může libovolný počet témat publikovat svoje data. Tím je zajištěna tzv. many-to-many komunikace, kdy všichni odběratelé přijímají veškerá data od všech publikujících. Každé téma má definovaný typ zpráv, které distribuuje. Díky tomu je zajištěna jednoznačnost formátu dat mezi publikujícími a odebírajícími uzly.

Zprávy

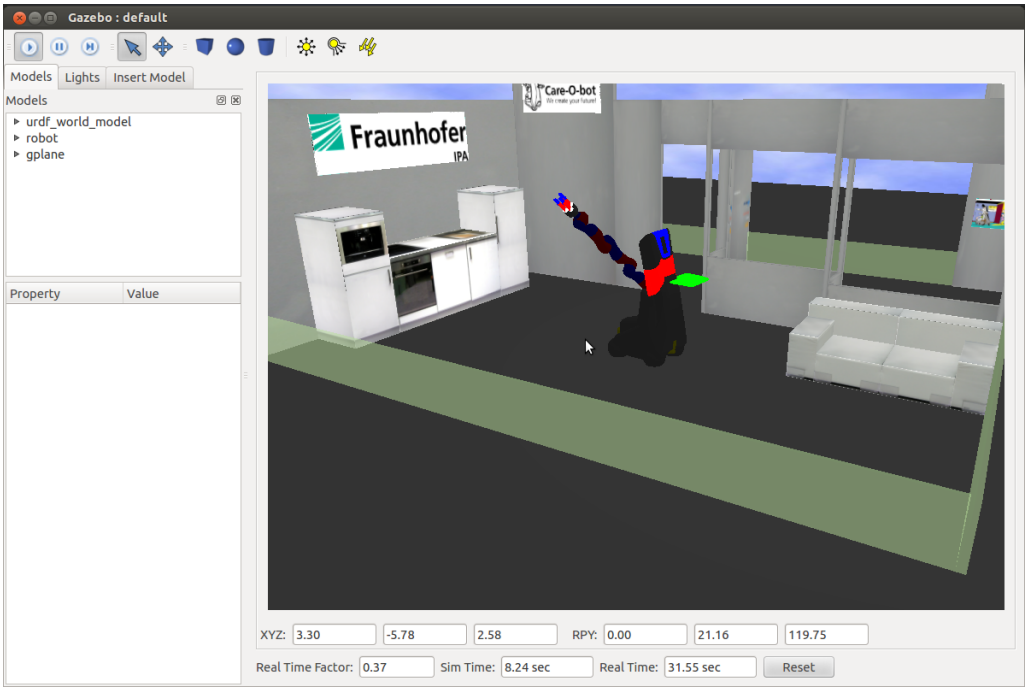
Zatímco témata představují způsob komunikace mezi uzly, samotný přenos dat je realizován pomocí zpráv (*Messages*). Jednotlivé zprávy obsahují strukturovaná data týkající se daného tématu. Struktura dat je dána typem zprávy a definicí tohoto typu v souboru s příponou `msg`. Pro přenos zpráv pomocí témat musí souhlasit typ zprávy s definovaným typem tématu.

3.2 Rosjava

Díky nedávné implementaci knihovny Rosjava[7] je nyní možné psát moduly přímo vytvářející uzly či témata pro ROS v programovacím jazyce Java. Z hlediska ovládací aplikace je tedy možné použití mobilních zařízení s operačním systémem Android, který pro své aplikace využívá výhradně jazyk Java. Rosjava umožňuje buď vytvořit vlastní instanci ROSu, nebo se vzdáleně připojit k již existující instanci. V našem případě nás tedy bude zajímat druhý případ, kdy se budeme chtít připojit přímo na instanci ROSu spuštěné v robotovi.

3.3 Simulátor Gazebo

Vzhledem k tomu, že na světě je zatím pouze několik exemplářů robota Care-O-Bot, není tedy jednoduché testovat vyvíjený software přímo na robotovi. K řešení této překážky slouží simulátor Gazebo[6], který je součástí balíku Care-O-Bota pro ROS. Tento nástroj umožňuje spustit lokálně simulaci robota na PC a případně doplňovat různé simulační podmínky. Simulace dokáže nahradit většinu reálných senzorů, za které generuje virtuální data na základě simulovaného prostředí. Díky tomu je možné poměrně detailně testovat vyvíjenou aplikaci bez nutnosti přítomnosti robota a rizika jeho poškození. Ukázka grafického výstupu ze simulátoru gazebo je na obrázku 3.1.



Obrázek 3.1: Ukázka simulace v gazebu

Kapitola 4

Využívané experimentální robotické platformy

Cílem projektu je vytvořit univerzální aplikaci pro ovládání různých robotů. Při samotném vývoji a testování však bylo žádoucí zaměřit se na konkrétní robotické projekty, které by alespoň rámcově vymezily požadavky na uživatelské rozhraní a zároveň by bylo možné na nich aplikaci otestovat a vyhodnotit její funkčnost. V této kapitole budou blíže popsané dvě vybrané robotické platformy - Care-O-Bot a Turtlebot.

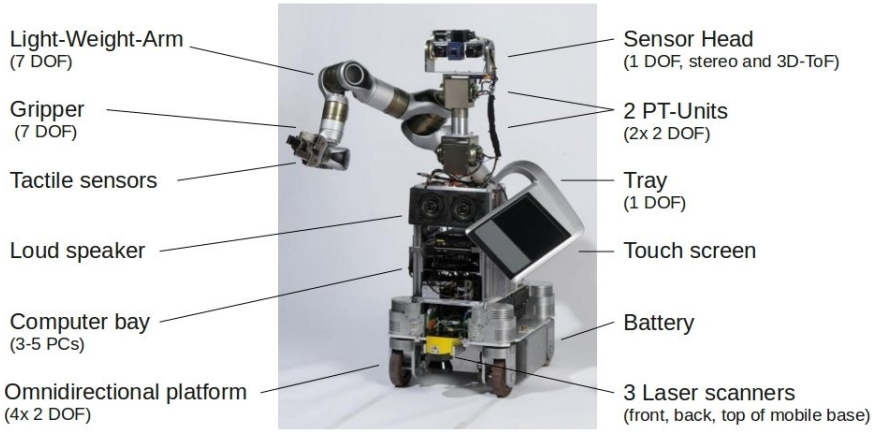
4.1 Care-O-Bot

Jak již bylo řečeno, Care-O-Bot[4] byl vyvinut jako pomocník v domácnosti pro lidi s pohybovým omezením. Robot má k dispozici množství kamer, ultrazvukových senzorů a laserových skenerů, aby byla zajištěna dokonalá orientace i v neznámém prostoru. Kromě klasických kamer využívá i integrovaný senzor kinect, který mu pomáhá vytvořit trojrozměrný model prostředí. K pohybu slouží základna s koly umožňující všesměrový pohyb a rotaci. K manipulaci má robot 3-kloubové rameno zakončené třemi prsty se zpětnou vazbou síly úchopu. Jak je vidět na obrázku 4.1, rameno je možné v případě pohybu robota složit, aby nezavazelo při přesunu.

Obrázek 4.1: Care-O-Bot 3 z Fraunhofer IPA (zdroj [4])

4.1.1 Technické parametry

Technické parametry a použité komponenty popisuje tabulka 4.3. Schematický obraz jednotlivých částí je pak na obrázku 4.2.



Obrázek 4.2: Nákres komponent robota Care-O-Bot 3 (zdroj [4])

4.2 TurtleBot

TurtleBot[1] je otevřená robotická platforma primárně určena pro výzkumné účely. Koncept TurtleBota využívá výhod funkčního a jednoduchého průmyslového podvozku robotického vysavače iRobot, úspěšného kamerového senzoru Kinect od Microsoftu a otevřené platformy ROS. Díky těmto výhodám nabízí TurtleBot ideální a poměrně levnou možnost vývoje robotických systémů. Robot dále nabízí mnoho možností rozšíření a díky široké komunitě a otevřenému vývoji je ideální pro rychlé prototypování a testování nových algoritmů. Na obrázku 4.4 je obrázek TurtleBota spolu s popisem jednotlivých komponent.

Rozměry (ŠxVxH)	75/55/145cm
Váha	180kg
Napájení	Li-ion baterie 60Ah
Všesměrový podvozek	Neobotix MOR[3], 8 motorů 2 SICK S300 skener 1 Hokuyu URG-04LX skener
Rameno	Schunk LWA 3 [15] max. zátěž 3kg
Gripper	Schunk SDH with tactile sensor
Torzo	2 Schunk PW (70 a 90) pan/tilt 1 Nanotec DB42M axis
Obrazové snímače	2 AVT Pike 145 C, 1394b, 1330x1038 MESA Swissranger 4000 Microsoft Kinect

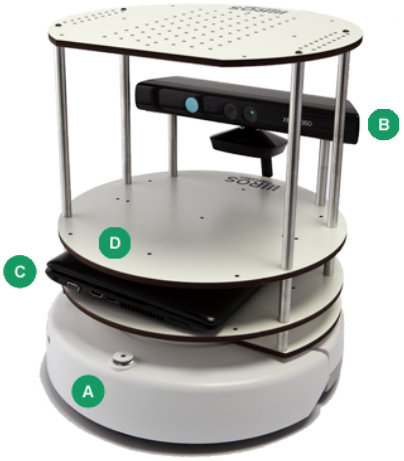
Obrázek 4.3: Technická specifikace robota Care-O-Bot 3 (zdroj [4])

A Základna a napájecí deska
- iRobot Create
- 3000 mAh Ni-MH Baterie
- 150° Gyroskop

B 3D Snímač
- Microsoft Kinect
- Napájení pro Kinect

C Počítač :: ASUS 1215N
- Intel® Atom™ D525 Dual Core
- NVIDIA® ION™ Discrete Graphics

D TurtleBot Hardware
- Patrová konstrukce TurtleBota
- Děrované desky pro montáž modulů



Obrázek 4.4: Schema robota TurtleBot

Kapitola 5

Projekt SRS

Cílem evropského projekts SRS (Multi–role Shadow Robotic System for Independent Living) je vyvinout potřebný software pro robota Care-O-Bot na takovou úroveň, aby jej bylo možné použít v reálném prostředí. Právě v rámci projektu SRS vznikly požadavky na vývoj několika uživatelských rozhraní, která budou poskytovat jejich uživatelům možnost vzdáleného ovládání robota.

5.1 Popis cílových skupin

V případě robota Care-O-Bot byly v rámci projektu SRS[8] definovány požadavky na vytvoření třech různých vzdálených rozhraní, lišící se právě úrovní řízení, respektive cílovou skupinou uživatelů. Pro účely vymezení cílů projektu vycházíme z tohoto rozdělení. V následujících odstavcích budou jednotlivá rozhraní popsána blíže.

5.1.1 Rozhraní pro lokální uživatele

Uživatelské rozhraní pro lokální uživatele má být nejběžněji používaným ovládáním. Je určeno přímo pro uživatele, kterým robot slouží, tedy pro starší či méně pohyblivé osoby. U tohoto rozhraní je zohledněn fakt, že u starších či nemocných lidí nelze počítat s vysokou úrovní zkušeností s moderními prostředky či ochotou se těmto novým technologiím učit. Naopak by toto rozhraní mělo být velmi intuitivní a jednoduché i pro uživatele nesetkávající se s moderními prostředky a technikou.

Obecně definované požadavky pro toto rozhraní:

- Bezdrátová komunikace s robotem
- Malé, lehké a mobilní zařízení
- Intuitivní rozhraní (bez dodatečných zařízení)
- Bezdrátová komunikace s rozhraním pro operátory

Hlavní funkce aplikace:

- Přijmout / odmítnout vzdálené řízení (z UI_PRI nebo UI_PRO)
- Výběr a spuštění úkolu
- Výběr parametrů pro daný úkol
- Akce STOP

- Lokalizace uživatele v prostoru
- Výběr objektů pro úlohy s manipulací
- Zobrazení základních stavových informací
- Zobrazení stavu řízení robota
- Zobrazení následující aktivity robota

5.1.2 Rozhraní pro soukromé operátory

Další definovanou aplikací je uživatelské rozhraní pro soukromého operátora[13]. Tím je myšlena osoba většinou blízká uživateli, která je proškolena k používání robota a řešení netriviálních problémů, se kterými si základní rozhraní neporadí. U tohoto typu ovládání se již tedy předpokládají jisté zkušenosti s moderní technikou, stejně jako se předpokládá, že bude uživatel s aplikací předem detailně seznámen.

Obecně definované požadavky jsou shodné s obecnými požadavky pro lokální rozhraní.

K základním funkcím z lokálního rozhraní jsou navíc definovány tyto funkce:

- Přepínání mezi autonomním a manuálním řízením
- Zobrazení pozice robota v 2D mapě
- Vhodné zobrazení okolního prostředí robota (3D pohled, obraz kamery, apod.)
- Navádění robota do cílové pozice (autonomní mód, zadání bodu na 2D mapě)
- Manuální navádění robota (semi-autonomní mód, pohyb joystickem s využitím senzorů)
- Navádění manipulátoru (ramene) do cílové pozice (autonomní, výběrem objektu z obrazu)
- Řízení kamery robota (úhel natočení, přiblížení)

5.1.3 Rozhraní pro profesionální operátory

Poslední definované ovládání je logicky nejpropracovanější, ale také z hlediska použití nejnáročnější. Je určeno pro profesionální operátory, kteří dobře rozumí fungování robota a jsou schopni také pomoci řešit problémy lokálním uživatelům či soukromým operátorům. Aplikace pro profesionální operátory je také výrazně komplexnější a její použití vyžaduje větší zobrazovací zařízení a pokročilejší ovládací prvky. Proto se předpokládá, že toto rozhraní bude implementováno pro klasické PC.

Obecné požadavky na profesionální rozhraní se tedy liší od předchozích:

- Vzdálená komunikace s robotem
- Plný 3D vstup (se zpětnou vazbou) pro řízení manipulátoru
- Volitelně: 3D výstup s vizualizací dat (rozšířený 3D výstup)

5.2 Srování s navrhovanou aplikací

Lokální rozhraní je většinou velmi svázané s funkcemi robota a zadávání jednotlivých úkolů je tedy pro různé typy robotů odlišné. Zároveň tento typ rozhraní ve většině případů pouze

zpřístupňuje funkce implementované na robotovi a samotná interakce se tak zužuje na jednoduché zadávání pokynů pomocí tlačítek, případně se přenáší na operátory. Účelem tohoto rozhraní tedy není přímé řízení robota, narozdíl od navrhované aplikace.

Uživatelská rozhraní pro operátory naopak poskytují komplexnější řešení pro ovládání robota a většina ovládacích prvků a zobrazovaných dat bude pro různé roboty podobná. Zatímco však pro operační systém Linux je nabídka hotových aplikací a nástrojů poměrně široká, pro ostatní platformy je zatím omezená. Vzhledem k tomu, že jeden z požadavků pro rozhraní soukromých operátorů je mobilita, vznikla tedy poptávka po implementaci tohoto typu rozhraní na některou mobilní platformu.

Návrh aplikace proto z části vychází z funkčních požadavků pro rozhraní pro soukromé operátory z důvodu zasazení aplikace do kontextu dalších ovládacích rozhraní robota Care-O-Bot a možnosti využití projektu pro tohoto nebo podobného robota.

Kapitola 6

Návrh aplikace pro vzdálené řízení robota

Při návrhu aplikace bylo vycházeno z požadavků na mobilitu zařízení a univerzalitu, tedy možnost použití na různých robotech. Zároveň ale bylo přihlédnuto k požadavkům na rozhraní soukromých operátorů definovaném projektem SRS, pro možnost využití aplikace na robotu Care-O-Bot a zasazení do kontextu ostatních rozhraní robota.

Výsledná aplikace má za cíl umožnit uživateli na mobilním zařízení vzdáleně ovládat vybraného robota. K orientaci v prostředí okolo robota bude sloužit zobrazení:

- dat z laserových skenerů (detekce překážek v rovině)
- 3D modelu prostředí
- 3D modelu robota, jeho pozice a orientace v prostředí
- aktuálního snímaného mračna bodů
- obrazu z kamery

K řízení robota budou k dispozici ovládací prvky:

- virtuální joystick pro manuální řízení
- manipulátor pro natáčení kamery (Pan-Tilt)
- ovládací prvky specifické pro dané roboty (tlačítka s různými funkcemi)

Dále je třeba uživateli umožnit nastavit parametry jednotlivých ovládacích prvků a zobrazovaných dat. Aplikace může obsahovat také další možnosti nastavení, jako například způsob komunikace, připojení k robotovi, otáčení obrazovky a další. Zobrazení a rozmístění jednotlivých prvků bude definováno v konfiguračním souboru. Pro jednotlivé roboty bude možné nahrát do aplikace specifický konfigurační soubor.

6.1 Univerzální rozhraní

Aby byla aplikace využitelná na různých robotických projektech, je nutné se zaměřit na vývoj univerzálního rozhraní, které bude možné použít pro libovolné roboty, a to ideálně bez nutnosti zásahu do kódu aplikace. Toho lze dosáhnout parametrizací jednotlivých prvků rozhraní (manuální zadání parametrů specifických pro daného robota) a možností přizpůsobení samotného rozhraní (zobrazených komponent a jejich rozložení).

Univerzálnost rozhraní však s sebou přináší i nutnost konfigurace. Čím větší míra univerzality aplikace, tím složitější může být její správné nastavení. Vzhledem k použití aplikace na konkrétních robotech s předem definovanými požadavky je však žádoucí, aby uživatelé mohli ovládat tyto roboty bez nutnosti nastavení či přizpůsobení aplikace. Nabízí se tedy řešení - vytvořit předdefinované konfigurace pro jednotlivé konkrétní roboty. Univerzálnost rozhraní by pak spočívala v možnosti použití buď předdefinovaných konfigurací, nebo vytvořením nové konfigurace pro daného robota. Tím by byla zajištěna přenositelnost na další roboty a zároveň by byla zachována jednoduchost použití aplikace.

6.2 Způsob řízení robota

Řízení robota bude realizováno pomocí virtuálního joysticku, manipulátoru kamery a panelu příkazů (sada tlačítek). V případě samotné navigace robota by bylo možné využít autonomní řízení pomocí zadání trasy na mapě. Ale vzhledem k požadavku na nízkou úroveň autonomie ovládání robota bylo zvoleno pouze manuální řízení ve formě joysticku. V případě, že by se toto řešení ukázalo jako nevyhovující nebo uživatelsky nepřívětivé, bylo by možné přistoupit k řešení navigace pomocí 2D mapy.

6.3 Uživatelské rozhraní

Uživatelské rozhraní je stěžejní část aplikace. Pokud má uživatel aplikaci dlouhodobě a efektivně používat, je třeba pečlivě volit ovládací prvky a zobrazené komponenty. Pro zpřehlednění celého rozhraní byl pro zobrazení zvolen přístup popsaný v 2.5, tedy zobrazení pomocí jednotné 3D scény, do které budou integrována různá data ze senzorů robota.

Pro ostatní prvky nacházející se na hlavní obrazovce byly pro zvýšení přehlednosti zobrazení navrženy 2 pravidla. Za prvé - pro prvky překrývající 3D scénu využít částečné průhlednosti komponent. Průhlednost je ale třeba použít v rozumné míře tak, aby prvky neztrácely na čitelnosti, ale zároveň umožnily průhled na 3D scénu. A za druhé - umožnit uživateli některé komponenty, které nejsou stěžejní pro řízení robota, dočasně skrýt.

Pro nastavení zobrazení a dalších parametrů aplikace by mělo být využito standartních prvků platformy Android. Vzhledem k tomu, že se předpokládá použití na tabletech, jako hlavní navigační prvek by bylo vhodné využít horní akční lišty (*ActionBar*).

6.3.1 Ovládací prvky

Pro ovládání robota budou sloužit tyto 3 komponenty:

- Virtuální joystick
- Manipulátor kamery
- Panel příkazů

Zatímco virtuální joystick je primárním navigačním prvkem, manipulátor a panel příkazů nejsou pro velkou část vykonávaných úloh stěžejní. Z toho důvodu by bylo vhodné dát uživateli možnost tyto ovládací prvky skrýt.

Virtuální joystick

Navigace pomocí joysticku by měla být jednoduchá a přímočará. Uživateli by mělo být tedy předem jasné, jak bude robot na pohyb joysticku reagovat. Převod pokynů z joysticku na pohyb robota lze realizovat dvěma způsoby. Komponenta virtuálního joysticku obsažená v knihovně Rosjava převádí pokyn z joysticku přímo na požadovanou rychlost, kterou by se měl robot pohybovat. Tento přístup však může způsobit komplikace v případě, že je komunikace mezi tabletem a robotem pomalejší, nebo když se sníží rychlost odezvy aplikace. V takovém případě by robot pokračoval v pohybu zadanou rychlostí a v případě absence či selhání bezpečnostních mechanismů robota proti kolizi by mohl být poškozen nárazem do překážky. Například v případě Care-O-Bota je možné se tomuto riziku vyhnout použitím tzv. bezpečného ovládání, při kterém se robot pohybuje dle zadání pouze v případě, že nehrozí kolize.

Druhým možnýn řešením je z aktuální pozice robota a pokynu z joysticku vypočítat cílovou pozici a tu poslat robotovi. V tom případě nehrozí, že by při chybě komunikace robot zadané místo přejel. Nevýhodou je nutnost získávat zpětně novou aktuální pozici robota.

Manipulátor kamery

I když je komponenta *PanTiltView* z knihovny Rosjava původně určena pro přímou manipulaci s kamerou, robot Care-O-Bot mechanismem natáčení kamery přímo nedisponuje. Místo toho je však schopen natáčet vrchní část torza ve směru horizontálním a vertikálním. Tento mechanismus tedy svojí funkcí zastává přímo pohyb kamery, proto o něm budeme mluvit jako o manipulátoru kamery. Vzhledem k tomu, že robot se může otáčet okolo své osy (vertikální), pravděpodobně bude významnější funkcí natáčení kolem horizontální osy.

Panel příkazů

Panel příkazů by měl sloužit jako sada užitečných funkcí, které manipulují s jednotlivými částmi robota. Protože každý robot je v tomto ohledu jiný, definice těchto příkazů by měla být součástí konfiguračního souboru daného robota. Pro robota Care-O-Bot se nabízí množství užitečných funkcí. Může to být např. změna orientace kamer (vpřed / vzad), změna polohy ramene (složeno, rozvinuto nahoru, do směru, atd.), manipulace s podnosem a další. Panel by měl ale obsahovat i obecnější příkazy, které budou pravděpodobně shodné pro většinu robotů - např. bezpečnostní zastavení, pozastavení robota, otočení na místě (o 90, 180, 270 stupňů).

6.3.2 Zobrazená data

I když je v rámci návrhu aplikace snaha integrovat zobrazovaná data do jednotné 3D scény, pro některé typy dat je tato integrace nevhodná, či dokonce nemožná. Ukazuje se, že i když aplikace poskytuje uživateli komplexní pohled na prostředí ve formě 3D scény, je vhodné v některém případě poskytnout dodatečně i zjednodušené zobrazení 2D dat. Pro zobrazení v aplikaci bylo zvoleno tedy dvou hlavních zobrazovacích prvků - 3D scény a 2D mapy orientované z pohledu shora. V takové mapě se uživatel zorientuje rychleji a bez nutnosti manipulace do vhodné polohy. V 2D pohledu bude znázorněna pozice robota a data z laserových skenerů, které jsou právě přijímány ve formě 2D pole.

6.3.3 Rozmístění prvků

Vzhledem ke snaze integrace většiny dat do centrální 3D scény je logickým krokem zvolit pohled na 3D scénu jako hlavní prvek uživatelského rozhraní. Bylo dokonce zvoleno zobrazení scény na celou plochu okna aplikace a případné další prvky budou tuto scénu překrývat. Předpokládá se, že zobrazení scény bude centrováno a tedy pro další prvky bude využito okrajů obrazovky.

6.4 3D scéna

3D scéna bude stěžejním prvkem uživatelského rozhraní aplikace. Vzhledem k použití na dotykovém tabletu je logické využít multidotykového ovládání pro manipulaci se scénou (pohledem na scénu). Způsob ovládání by měl být intuitivní, proto je vhodné zvolit standardní gesta využívaná na dotykových tabletech, jako:

- Natočení scény tažením (tzv. swipe)
- Přiblížení / oddálení scény dvěma prsty (tzv. pinch-to-zoom)
- Posun scény tažením dvěma prsty (tzv. scroll)

Těmito gesty lze pokrýt manipulaci s kamerou scény všemi směry, uživatel má tedy možnost prohlédnout si i detaily jednotlivých objektů z libovolného úhlu.

6.4.1 Model prostředí

Zobrazení 3D modelu prostředí je velmi důležité pro celkovou orientaci uživatele v prostoru, kde se robot pohybuje. V prostředí ROS je využívaný standard pro ukládání modelů formát *urdf*. Pokud tedy uvažujeme o zobrazení modelu prostředí, je žádoucí, aby aplikace s tímto formátem uměla pracovat a uživatelé nemuseli převádět své stávající modely do jiných formátů. Tyto modely ale mohou být poměrně složité a mohly by způsobit pomalejší vykreslování a odezvu aplikace. Z toho důvodu by bylo vhodné uvažovat i o jednodušším popisu modelu, méně náročném na vykreslení. Takovou jednodušší variantou k formátu `urdf` by mohl sloužit formát `world` určený pro simulátor Gazebo. Tento model většinou obsahuje objekty, které spíše určují fyzické prostředí, než jeho vzhled. Narozdíl od `urdf` nepoužívá pro vykreslování další externí soubory se síťovým modelem objektu (tzv. mesh) a jeho vykreslení je tedy jednodušší z pohledu implementace i samotného vykreslení.

6.4.2 Model robota

Zobrazení modelu robota je obdobné modelu prostředí. Opět je zde využívaný formát `urdf` pro definici modelu robota. Práce s tímto modelem je však komplikovanější. Zatímco u modelu prostředí se předpokládá, že stěny a většina prvků ve scéně jsou statické, model robota se mění. Robot je složen z mnoha prvků a tyto prvky jsou vůči sobě pohyblivé. Model robota popsaný formátem `urdf` však neobsahuje informace o vzájemné poloze a natočení těchto komponent, pouze jejich napojení.

O transformace všech objektů v prostoru se stará knihovna `TF`. Ta vyhodnocuje veškerá data týkající se polohy jednotlivých komponent a rozesílá kompletní seznam transformací v podobě ROS tématu `/tf`. Aplikace tedy musí vyhodnotit transformace pro každý objekt a při vykreslování jej správně posunout a natočit ve scéně.

6.4.3 Mračno bodů (Point-cloud)

Jak bylo zmíněno v odstavci 2.6.2, kromě snímání samotného obrazu je možné pomocí kombinovaných senzorů zaznamenávat tzv. mračno bodů, neboli point-cloud. Zařízení, jako například Microsoft Kinect, která integrují několik různých typů obrazových snímačů, jsou schopna data analyzovat a převést do hloubkové mapy. Ta udává vzdálenost jednotlivých bodů od snímače. V kombinaci s RGB obrazem je tedy možné získat prostorový obraz, neboli mračno bodů.

Care-O-Bot i TurtleBot, stejně jako další vyvíjení roboti, jsou takovým snímačem vybaveni. Protože na rozdíl od modelu prostředí jsou tato data aktuální a zároveň obsahují prostorovou informaci, mohou tak uživateli poskytnout důležité okolnosti či detaily o aktuálním dění. Aplikace by tedy měla být schopna tato data přijímat a zobrazit v 3D scéně.

6.4.4 Prolnutí zobrazených dat

Vzhledem k tomu, že model prostředí i mračno bodů zobrazují ve své podstatě stejné objekty, ale v jiné formě, je zřejmé, že bude docházet ke kolizi či překrývání těchto dat v 3D scéně. Jak bylo zmíněno již dříve, v takovém případě by měla mít prioritu aktuální data, tedy mračno bodů. Ideálním řešení by bylo při vykreslování modelu určit oblast, která bude v kolizi s mračnem bodů a takovou oblast skrýt. Další variantou je v místě kolize (překrytí) zobrazit objekty s určitou průhledností. Zde by se vzhledem k prioritě zobrazení mračna bodů nastavila průhlednost modelu vyšší než průhlednost mračna bodů.

6.4.5 Mapování textury na model

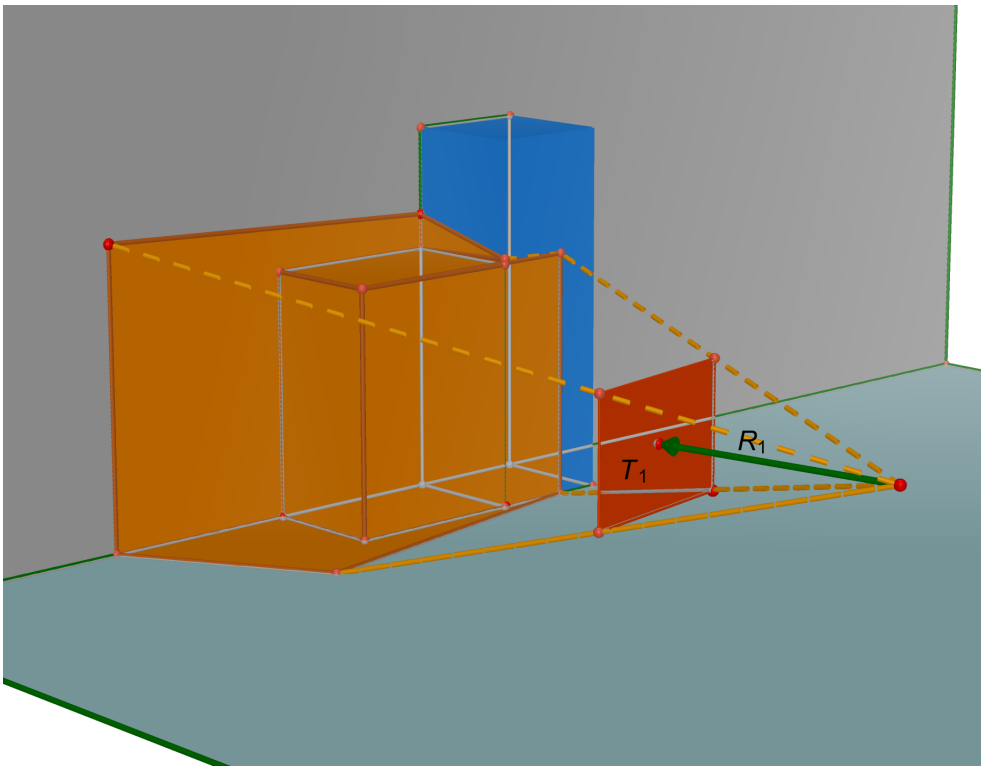
Pro zobrazení obrazu z kamery v 3D scéně neexistuje přímočaré řešení. Jak také vyplývá z odstavce 2.6.2, přístupů k řešení je několik. Nejvhodnějším a nejrealističtějším přístupem se zdá být mapování textury na model. Díky tomu lze propojit nebo doplnit model o aktuální data. Zmíněný přístup mapování textury se však zabýval pouze zpracováním na statických datech. V našem případě je však nutné toto mapování provádět v reálném čase. Z tohoto pohledu se jeví jako možné řešení zpracovat výpočet přímo při vykreslování modelu. Výhodou takového řešení by byla také rychlost zpracování, protože vykreslovací výpočty jsou prováděny převážně na grafickém procesoru, který je na takové výpočty optimalizován.

Vzhledem k tomu, že přístup mapování textury z obrazu na model je poměrně ojedinělý a nebylo možné použít žádné stávající řešení, bylo nutné přijít s vlastním řešením výpočtu. Na obrázku 6.1 je naznačen výpočet výsledných souřadnic textury ze zadaného bodu v prostoru.

Vstupními parametry pro výpočet je:

- Směrový vektor kamery ($R1$)
- Pozice kamery robota
- Textura - obraz z kamery ($T1$)

Samotný výpočet bude detailněji popsán v kapitole 7.



Obrázek 6.1: Koncept výpočtu projekce textury na model

Kapitola 7

Výpočet mapování textury na model

Pravděpodobně nejnáročnější částí při vývoji aplikace byl výpočet mapování (projekce) obrazu na model prostředí. K výpočtu lze přistoupit dvěma způsoby. Prvním je pro každý bod z obrazu spočítat průsečík s modelem prostředí a tento bod následně obarvit barvou bodu v obrazu. Tento způsob se zdá být poměrně přímočarý, ale nese s sebou několik komplikací. Obraz a výsledný model nemají stejný počet bodů na stejné ploše, navíc tento rozdíl se ještě mění se vzdáleností kamery robota od modelu. Jinak řečeno, pro 2 sousedící body obrazu by byly obarveny 2 body modelu, které spolu však nesousedí. Tím by došlo ke fragmentaci obrazu na modelu a vznikly by neobarvená místa, která však jsou v záběru kamery a měla by být obarvena.

Lepším řešením je tedy druhý způsob - vypočítat pro každý bod modelu odpovídající bod v obraze (pokud je v záběru kamery). Tento výpočet je sice náročnější, ale je konzistentní se způsobem výpočtu pozice a barvy pixelů v OpenGL, a navíc nedojde k fragmentaci obarvených ploch modelu.

Na obrázku 7.1 je naznačen výpočet souřadnic textury ze zadaného bodu v prostoru a směrového vektoru a pozice kamery robota. Dále bude popsán samotný výpočet.

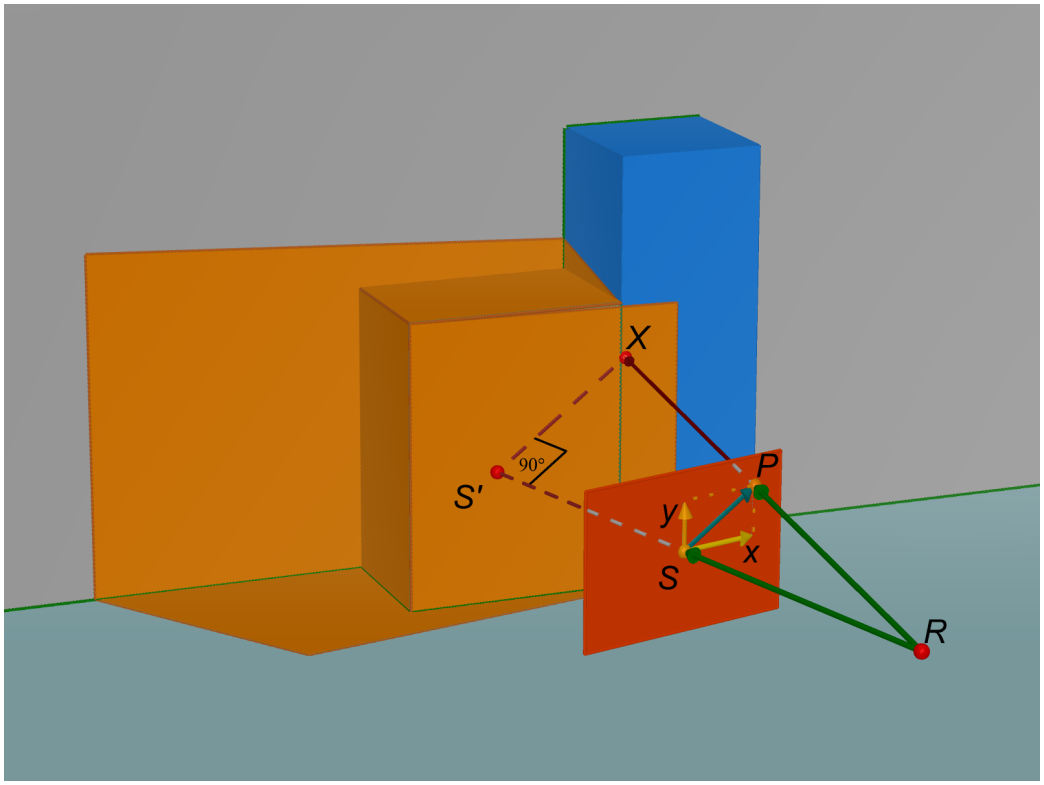
X	...	bod modelu, pro který hledáme odpovídající bod textury
R	...	pozice kamery robota
S	...	střed textury
$\vec{RS}$	...	směrový vektor kamery robota
P	...	hledaný bod průniku vektoru $\vec{RX}$ s texturou
S'	...	pata kolmice spuštěné z bodu X na přímku RS

Nejdříve vypočítáme polohu bodu P:

- Pomocí vzorců pro výpočet skalárního součinu vektorů určíme kosinus úhlu svíraném vektory RX a RS:

$$\vec{a} \cdot \vec{b} = a_1 b_1 + a_2 b_2 + a_3 b_3 \tag{7.1}$$

$$\vec{a} \cdot \vec{b} = |\vec{a}| \cdot |\vec{b}| \cdot \cos \alpha \tag{7.2}$$



Obrázek 7.1: Výpočet projekce textury na model prostředí

Pak

$$\cos\alpha = \frac{a_1b_1 + a_2b_2 + a_3b_3}{|\vec{a}| \cdot |\vec{b}|} \tag{7.3}$$

Tedy pro vektory RX a RS

$$\cos\angle XRS = \frac{RX_1RS_1 + RX_2RS_2 + RX_3RS_3}{|\overrightarrow{RX}| \cdot |\overrightarrow{RS}|} \tag{7.4}$$

- Z kosinu úhlu svíraném vektory RX a RS a velikosti vektoru RS můžeme vypočítat velikost vektoru RP:

$$\cos\angle XRS = \frac{|\overrightarrow{RS}|}{|\overrightarrow{RP}|} \tag{7.5}$$

A tedy

$$|\overrightarrow{RP}| = \frac{|\overrightarrow{RS}|}{\cos\angle XRS} \tag{7.6}$$

- Z poměru velikostí vektorů RX a RP, pak můžeme vypočítat samotný vektor RP a bod P

$$\overrightarrow{RP} = RX \cdot \frac{|\overrightarrow{RP}|}{|\overrightarrow{RX}|} \tag{7.7}$$

$$P = R + \overrightarrow{RP} \tag{7.8}$$

Po výpočtu polohy bodu P dostaneme vektor SP, který leží v rovině textury. Další částí bude vypočítat x a y souřadnice bodu textury. K nalezení směrového vektoru $\vec{x}$ se pokusíme najít vektor kolmý k $\vec{RS}$. Takových vektorů je ale nekonečně mnoho. Využijeme skutečnosti, že z-souřadnice výsledného vektoru má být nulová (robot nedokáže rotovat kamerou kolem osy pohledu).

- Využijeme pravidla pro kolmost vektorů v prostoru:

$$\vec{a} \cdot \vec{b} = a_1 b_1 + a_2 b_2 + a_3 b_3 = 0 \tag{7.9}$$

  Pokud je $b_3 = 0$, pak stačí zvolit $b_1 = a_2$ a $b_2 = -a_1$, případně $b_1 = -a_2$ a $b_2 = a_1$ a vektory budou kolmé:

$$\vec{a} \cdot (a_2, -a_1, 0) = a_1 a_2 - a_2 a_1 + a_3 \cdot 0 = 0 \tag{7.10}$$

  Směrový vektor $\vec{x}$ tedy získáme jako:

$$\vec{x} = (RS_y, -RS_x, 0) \tag{7.11}$$

- Směrový vektor $\vec{x}$ však nemá správnou délku. Pro výpočet jeho správné velikosti (souřadnice textury t_x opět využijeme vzorce pro skalární součin (7.2 a 7.2) a postup jako při výpočtu velikosti vektoru $\overrightarrow{RP}$ v 7.6:

$$t_x = \frac{\vec{x} \cdot \overrightarrow{SP}}{|\vec{x}|} \tag{7.12}$$

- Pro výpočet souřadnice využijeme vektorového součinu. Výsledkem vektorového součinu dvou kolmých vektorů je vektor kolmý k oběma původním. V našem případě tedy vypočítáním vektorového součinu vektorů $\overrightarrow{RS}$ a $\vec{x}$ získáme směrový vektor $\vec{y}$, na kterém bude ležet souřadnice tesxtury t_y:

$$\vec{y} = \vec{x} \times \overrightarrow{RS} \tag{7.13}$$

  A stejně jako v 7.12 vypočítáme souřadnici textury t_y:

$$t_y = \frac{\vec{y} \cdot \overrightarrow{SP}}{|\vec{y}|} \tag{7.14}$$

Nyní souřadnice t_x a t_y určují příslušný bod v obraze (souřadnice jsou ze středu obrazu) v případě, že je bod v záběru kamery (absolutní hodnota obou souřadnic je menší než 0,5). Na obrázku 7.2 je ukázka výpočtu souřadnic ve vertex shaderu.

```
void main() {
    vec4 vertexPosCam = u_MMatrix * a_Position;
    vec3 point_position = vertexPosCam.xyz;
    vec3 point_vector = point_position - u_RobotPos;
    float l_coef = dot(point_vector, normalize(u_RobotCam))
      / length(u_RobotCam);
    vec3 texture_vector = point_vector / l_coef;
    vec3 Z~= u_RobotPos + texture_vector;
    vec3 X = u_RobotPos + u_RobotCam;
    vec3 texture_coord_vect = Z~- X;
    vec3 texture_x_vect =
      normalize(vec3(-u_RobotCam.y, u_RobotCam.x, 0.0));
    float x_coef = dot(texture_x_vect, texture_coord_vect);
    vec3 texture_y_vect =
      normalize(cross(texture_x_vect, u_RobotCam));
    float y_coef = dot(texture_y_vect, texture_coord_vect);
    if((dot(point_vector, u_RobotCam) >= 0.0)) {
        v_TextureCoord = vec2( x_coef, y_coef);
    }
    vec3 modelViewNormal = normalize(u_NormMatrix * a_Normal);
    float diffuse = max(dot(modelViewNormal, u_lightVector), 0.4);
    v_Color = vec4(diffuse*u_Color.xyz, u_Color[3]);
    gl_Position = u_MVPMatrix * a_Position;
}
```

Obrázek 7.2: Ukázka výpočtu souřadnic textury ve vertex shaderu

Kapitola 8

Implementace

V této kapitole bude popsána hlavní fáze vývoje projektu - samotná realizace. Vývoj aplikace byl limitován zejména absencí fyzického robota Care-O-Bot, který měl sloužit jako jeden z experimentálních robotů. Místo testování na reálném zařízení však bylo využito simulátoru Gazebo, který dokáže navodit alespoň přibližné podmínky, jako při použití robota.

8.1 Použité vývojové prostředky

Při samotné implementaci aplikace bylo využito stávajících prostředků, které splňovaly požadavky návrhu aplikace a zároveň byly využívané i v jiných projektech. Toto kritérium bylo zvoleno z důvodu možného pokračování projektu. Pokud by byla aplikace závislá na projektu, který nemá dostatečnou uživatelskou základnu nebo podporu ze strany vývojářů, znemožňovala by tato závislost další rozvoj aplikace.

8.1.1 ROS

Platforma ROS je v poslední době velmi oblíbena u široké řady dalších robotických projektů. Použití ROSu v aplikaci tedy umožní přenositelnost na další roboty využívající ROS. Zároveň nabízí vysokou modularitu a řadu komunikačních knihoven a rozšíření. Jedním ze zajímavých rozšíření je knihovna Rosjava, umožňující použití ROSu v prostředí jazyka Java.

8.1.2 Android

Hlavním faktorem při výběru mobilní platformy byla především možnost integrace s nejpoužívanější robotickou platformou ROS[12] (popsanou dříve). Tomuto požadavku nejvíce vyhovuje platforma Android, pro kterou vznikla knihovna Rosjava[7] umožňující integraci s ROSem. Android podporuje přímo vývoj aplikací v jazyce Java, navíc knihovna Rosjava obsahuje některé základní komponenty pro komunikaci s ROSem a manipulaci a zobrazení dat.

Volba vývojových nástrojů byla v případě Androidu poměrně jednoduchá. V době vzniku projektu bylo prakticky jediným podporovaným nástrojem vývojové prostředí Eclipse [2]. Dnes už společnost Google nabízí i alternativu v podobě vlastního vývojového prostředí Android Studio, které je založené na oblíbeném prostředí IntelliJ IDEA od spo-

lečnosti JetBrains [5]. Eclipse však už nyní nabízí možnost exportu celého projektu do Android Studia, v případě dalšího vývoje projektu je tedy možné využít obě varianty.

8.1.3 Rosjava

Projekt Rosjava byl založen právě především kvůli chybějící podpoře platformy ROS pro mobilní zařízení. Při začátku vývoje aplikace byl však projekt Rosjava teprve po prvním roce svého vývoje a v průběhu implementace aplikace se postupně vyvíjela i Rosjava. Pro zachování kompatibility a možnosti dalšího rozvoje projektu byla postupně aktualizována i verze Rosjavy v aplikaci. Díky tomu také bylo možné průběžně využívat nové funkce vyvíjené v Rosjavě. Na druhou stranu takový vývoj přináší i nemalé komplikace spojené se změnami API a nutností přizpůsobit se novým změnám.

8.1.4 Rviz pro Android

Po vývoji projektu Rosjava do poměrně stabilní verze přistoupili autoři k vývoji nového projektu Rviz pro Android. Tento projekt stavěl na základních funkcích a třídách Rosjavy. Hlavním rozšířením však byla poměrně rozsáhlá implementace 3D zobrazení různých typů dat. V době, kdy se Rviz pro Android objevil však už byla v aplikaci 3D scéna implementována. Bylo nutné se rozhodnout, zda pokračovat ve vývoji vlastního řešení, nebo jej přizpůsobit objektovému návrhu Rvizu. Nakonec bylo zvoleno přepsání stávající implementace a integrace s vykreslováním Rvizu pro Android, a to hlavně kvůli pozdější využitelnosti aplikace v dalších podobných projektech.

8.2 Zobrazení 3D scény

3D scéna je dominantním prvkem celé aplikace, proto byl při implementaci kladen důraz na jednoduchost jejího ovládání a rychlou odezvu při manipulaci. Pro 3D výstup je využito standardu OpenGL ES ve verzi 2.0 pro jeho pokročilejší funkce a možnost přeprogramování shaderů (programovatelné části vykreslovacího řetězce). I když některá starší zařízení podporují pouze OpenGL ES ve verzi 1.1, k 1. květnu 2013 je zařízení nepodporujících verzi 2.0 pouze 0.2 % z celkového počtu používaných Android zařízení.

8.2.1 Model prostředí

Model prostředí, narozdíl od ostatních prvků ve scéně, není přenášen z robota prostřednictvím témat. Uživatel či operátor musí sám určit prostředí, ve kterém se robot bude pohybovat a zvolit příslušný soubor obsahující popis modelu. Jako podporované formáty popisu modelu by bylo vhodné zvolit *urdf* a případně i formát `world` (viz. 6.4.1 . Oba formáty vychází z *xml* standardu a je tedy možné pro jejich načtení využít principy shodné jako pro jiné xml soubory. Pro samotné zpracování souboru s modelem bylo nutné nastudovat strukturu obou formátů a jména jednotlivých objektů (xml tagů).

`World`

V první fázi vývoje byla zvolena implementace formátu `world`. Hlavními prvky tohoto formátu, ze kterých je možné vytvořit jednotlivé komponenty modelu jsou tagy:

`model:physical` definující objekt v prostoru, uvnitř kterého jsou:

```
<model:physical name="wall_3_model">
  <xyz>-0.08 2.36 1.25</xyz>
  <rpy>0.0 0.0 0.0</rpy>
  <static>true</static>
  <body:box name="wall_3_body">
    <geom:box name="wall_3_geom">
      <mesh>default</mesh>
      <size>0.16 0.058 2.5</size>
      <visual>
        <size>0.16 0.058 2.5</size>
        <material>Gazebo/Green</material>
      </visual>
    </geom:box>
  </body:box>
</model:physical>
```

Obrázek 8.1: Ukázka části souboru ve formátu `world`

`xyz` s údaji o posunutí objektu

`rpy` s údaji o rotaci objektu

`geom:box`, `geom:plane` definice geometrie objektu

- `size` - parametry objektu (šířka, výška, hloubka)
- `material` - povrch objektu (textura, barva)

Na obrázku 8.1 je ukázka části souboru ve formátu `world`.

8.2.2 Model robota

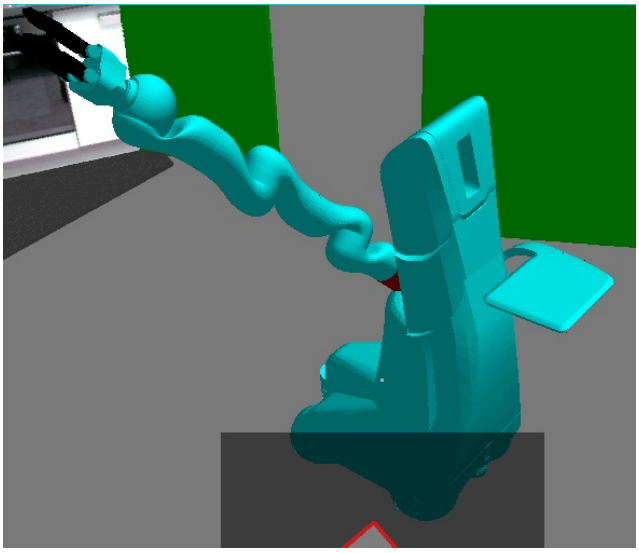
Model robota je na platformě ROS standartně reprezentován ve formátu urdf. Výhodou při práci s modelem robota je možnost distribuce přes ROS infrastrukturu. Data s definicí modelu jsou přenášena tématem `/robot_description` odebírajícím uzlům. Problém však může nastat v případě použití externích `stl` souborů v rámci definice modelu. Ty jsou totiž v případě vykreslování modelu načítána z disku a nejsou distribuovány po síti. V případě vykreslení na vzdáleném zařízení tyto soubory nejsou k dispozici. Z toho důvodu byl vytvořen jednoduchý program `mesh_server`, který je schopen přijímat požadavky na `stl` soubory a odeslat je po síti.

Model robota je popsán jednotlivými pohyblivými částmi, ale bez popisu jejich vzájemné polohy. Je tedy nutné z přijímaných transformací určit polohu a orientaci každé části robota, aby byla vykreslena korektně a model robota byl konzistentní. Zprávy s transformacemi jsou ukládány do objektu `FrameTransformTree`, který obsahuje všechny známé transformace mezi objekty a vytváří z nich stromovou strukturu. Z této struktury je pak možné zjistit rotaci a posunutí části robota vůči libovolnému jinému objektu ve scéně.

Po zjištění globální pozice a natočení části robota ve scéně je vykreslena pomocí funkcí OpenGL do 3D scény. Ukázka vykreslení robota v 3D scéně je na obrázku 8.2.

8.2.3 Mračno bodů

Mračno bodů je zpracováno na straně robota a distribuováno po síti ROS zprávami s typem `PointCloud`, nebo `PointCloud2`. `PointCloud2` je novější standard pro přenos mračna



Obrázek 8.2: Ukázka vykreslení modelu robota

bodů a oproti staršímu typu `PointCloud` umožňuje mimo jiné přenášet data reprezentována libovolným datovým typem. Zprávy mohou obsahovat různá data - zpravidla však obsahují kanály `x, y` a `z` pro určení souřadnic bodů. Dále mohou obsahovat kanály jako například `rgb, rgba` (barva bodu), `normal_x, normal_y, normal_z` a mnoho dalších. Robot Care-O-Bot poskytuje kromě souřadnic jednolivých bodů v mračnu i jejich barvu (`rgb` kanál). Barva bodu je reprezentována složkami r, g a b, zakódovaných do datového typu float (na 24 bitech). Pro zpětný převod zakódovaného kanálu `rgb` je třeba binárními operacemi získat hodnotu osmi bitů jednotlivých složek barvy. Druhou a jednodušší variantou je předání parametru barvy přímo ke zpracování v OpenGL jako pole tří hodnot typu `int`. Tím dojde k automatickému převodu 24-bitového čísla na tři 8-bitové celočíselné hodnoty. V OpenGL shaderu je pak třeba celočíselnou hodnotu v rozsahu 0 až 255 převést na typ `float` v rozsahu 0 až 1.

8.3 Další prvky uživatelského rozhraní

Jak bylo navrženo v kapitole 6, 3D scéna je hlavním prvkem uživatelského rozhraní a bude umístěna přes celou plochu uživatelského rozhraní. Ostatní prvky budou vykresleny přes 3D scénu po okrajích obrazovky. U všech prvků bylo využito průhlednosti, aby nebyla příliš omezena přehlednost 3D scény.

8.3.1 Zobrazení dat z laserových skenerů

Pro doplnění orientace v prostoru okolo robota je využito dat z laserových skenerů robota. Pro vykreslení dat je v knihovně Rosjava přítomna komponenta `DistanceView`. Ta byla upravena pro použití v aplikaci. Jako zdroj dat je použit přední skener robota. Aplikace odebírá data z tématu */front_scan* a vykresluje je do jednoduché bitové mapy se středem v pozici robota. Celá komponenta je částečně průhledná, v místě zobrazených dat je průhlednost nižší než v okolí. Zaznamenaná data jsou vykreslena červenou barvou, pro znázornění volného prostoru je oblast mezi robotem a překážkou znázorněna světle šedou barvou. Ukázka zobrazení dat z laseru je na obrázku 8.3.

Obrázek 8.3: Ukázka vykreslených dat z laserového skeneru

8.3.2 Virtuální joystick

Pro ovládání pohybu robota byla využita komponenta virtuálního joysticku z ukázkových aplikací knihovny Rosjava. Tato komponenta však implementuje jednoduché pokyny pro směr pohybu robota, zatímco Care-O-Bot využívá pro ovládání určení nové pozice v čase. Pro zjištění aktuálního systémového času slouží v ROSu téma */clock*. Poté jsou podle zadané pozice joysticku a zjištěného aktuálního času vygenerovány zprávy pro pohyb robota a publikovány na téma */torso_base/command*. To v případě nové zprávy pohybu už zajistí další kroky k tomu, aby se robot přesunul na požadovanou pozici.

8.4 Komunikace

Základem platformy ROS je mimo jiné i unikátní způsob komunikace mezi procesy, popsaný v kapitole 3.1. Tento model je zachován i pro knihovnu *Rosjava*, která bude v projektu použita. Samotná komunikace mezi uzly však předpokládá síťové spojení (TCP/IP) mezi aplikací a robotem (přes Wifi, Bluetooth,...).

8.4.1 Přenos dat

Veškerý přenos mezi robotem a vzdáleným rozhraním je realizován pomocí témat a zpráv. Tento přístup má některé výhody i nevýhody. Umožňuje vybrat konkrétní témata, která nás v rámci aplikace zajímají a ostatní data se nepřenáší. Na druhou stranu knihovna Rosjava v implementaci této komunikace při každém novém přijetí zpráv vyvolá novou událost a vynutí zpracování dat v aplikaci. Díky tomu může v případě přenosu velkého množství zpráv dojít k zahlcení aplikace tak, že nestačí zpracovávat veškerá příchozí data, což může vést ke zpomalení odezvy uživatelského rozhraní. Proto je vhodné množství zpráv a celkový přenos dat omezit na množství nezbytné k fungování aplikace, případně najít vhodný kompromis mezi množstvím přenášených zpráv a rychlostí jejich zpracování.

8.4.2 Omezení toku dat

Obecně je možné omezit tok dat dvěma způsoby. Za prvé, snížením frekvence publikace dat v daném tématu. Tím dojde i ke snížení počtu volání procesu zpracování dat v aplikaci. Druhou možností je pokusit se omezit množství dat přenášených v každé zprávě. Tím by se proces zpracování mohl urychlit a aplikace by tak byla schopna zpracovat vyšší počet příchozích zpráv.

Komprese obrazu

Jedna z možností výrazného omezení množství dat v jedné zprávě je komprese přenášeného obrazu. ROS přímo podporuje vytváření nových témat na základě stávajících a publikujících stejný obraz v komprimované či jinak upravené variantě. V případě aplikace se počítá s využitím obrazu pro mapování textury na model a v tomto případě je výhodné snížit rozlišení obrazu na mocniny čísla 2, se kterými standartně OpenGL pracuje. Jiné rozměry buď vykreslovací řetězec sníží, nebo je ve většině případů vůbec nevykreslí. Vzhledem k tomu, že obraz kamery je v poměru obvykle větším než 1:1 (většinou 4:3, 16:9, 3:2), je nutné obraz převést na čtverec a následně v aplikaci při vykreslení převést zpět na původní poměr stran.

Mračno bodů

Samotná podstata mračna bodů určuje velikost jeho dat při přenosu. Pro každý bod je třeba zaznamenat polohu (x, y, z) a barvu (RGB nebo RGBA), tedy 6-7 údajů na každý obrazový bod. Výpočet mračna bodů přitom vychází pouze z RGB obrazu a hloubkové mapy (tzn. 4 údaje na obrazový bod). Nabízí se tedy možnost přenosu jednotlivých obrazů (RGB obrazu a hloubkové mapy) a následný výpočet mračna bodů. Tím by se sice snížilo množství přenášených dat, ale na druhou stranu by se zvýšily výpočetní požadavky na samotnou aplikaci.

Model robota

Model robota je popsaný souborem ve formátu URDF. Tento formát však pro popis jednotlivých komponent může využívat dalších souborů ve formátu `stl` (tzv. mesh). Přenášení těchto souborů představuje další riziko zvýšení přenášených dat mezi robotem a aplikací. Řešením by bylo uložení modelu robota včetně dodatečných souborů přímo v aplikaci, nebo průběžné ukládání těchto souborů v datovém úložišti uživatele na tabletu.

TF data

Transformace jednotlivých částí robota a modelu je zásadní pro korektní zobrazení modelů. Při větším množtví objektů v modelu však může narůstat i množství přenášených transformací. Vzhledem k tomu, že téma publikující tyto transformace nemá definované žádné omezení frekvence publikace těchto dat, může rychle dojít k zahlcení zpracovávání jednotlivých zpráv. Pro omezení množství příchozích zpráv lze využít existující implementace uzlu `tf_throttle_node`, který odebírá zprávy z hlavního tématu `tf` a publikuje je v tématu `/tf_throttled` se zadanou frekvencí. Zároveň kontroluje změny v transformacích a přeposílá pouze změněné transformace. To přináší další výrazné snížení počtu přenášených zpráv.

Kapitola 9

Dosažené výsledky

Výsledná aplikace byla testována s robotem Care-O-Bot simulovaným v prostředí Gazebo a s robotem Turtlebot v reálném prostředí. V případě simulace robota Care-O-Bot však kvůli výpočetní náročnosti probíhala simulace chování robota s časovým koeficientem 0,2, tedy simulace byla výrazně pomalejší než robot v reálném použití. Z toho důvodu nebylo možné otestovat reálnou odezvu robota na ovládací prvky. Z funkčního hlediska však byly provedeny testy zobrazovaných dat i ovládacích prvků. Zobrazení 3D scény s různými typy vizualizovaných dat výrazně napomáhá k orientaci v okolí robota. Použití virtuálního joysticku je intuitivní a funguje dle předpokladů uživatele. Aplikace umožňuje při spuštění zvolit typ robota a později přizpůsobit nastavení pro zvoleného robota. Při používání aplikace je možné všechny prvky dočasně skrýt. Tím dojde zároveň k jejich deaktivaci (v případě přijímaných nebo odesílaných dat).

9.1 Care-O-Bot

Robot Care-O-Bot má oproti Turtlebotovi více senzorů a tedy i více možností zobrazených dat. Zároveň se u Care-O-Bota předpokládá použití ve známém prostředí, a proto je zobrazen model prostředí. Soubor s modelem je možné změnit v nastavení aplikace. Pro ovládání robota je použit virtuální joystick i manipulátor kamery. Ze přijímaných dat je možné zobrazit detekované překážky ze skeneru, model robota, mračno bodů, obraz z kamery (samostatně nebo jako texturu mapovanou na model prostředí).

9.1.1 Model prostředí

Při zobrazení většího množství dat představuje model prostředí ve formě `URDF` nezanedbatelnou zátěž pro výpočetní i paměťové prostředky tabletu. V kombinaci s dalšími zobrazenými daty je tedy vhodnější použití jednoduššího modelu, který má transformace jednotlivých komponent definovány staticky přímo v souboru. Pokud se však předpokládá model prostředí s dynamickými prvky, jednodušší formát je nevyhovující.

Jak je vidět na obrázku 9.1, zobrazení modelu `urdf` spolu s mračnem bodů bez další korekce je někdy nepřesné. To je způsobené nesprávnou lokalizací robota v simulaci. Pro účel testování byla implementována možnost korekce modelu (ukázka na obrázku 9.2.

Obrázek 9.1: URDF model prostředí a nepřesnost zobrazení vůči mračnu bodů.

Obrázek 9.2: Zjednodušený model prostředí a korekce pozice vůči mračnu bodů.

Obrázek 9.3: Ukázka modelu robota.

9.1.2 Model robota

Model robota představuje pravděpodobně nejnáročnější prvek 3D scény. V případě Care-O-Bota je model poměrně náročný (obrázek 9.3) a po načtení všech komponent do paměti tabletu může zabrat i 150MB z paměti RAM. Může se tak stát, že v průběhu používání aplikaci dojde vyhrazená paměť a nebude schopna přijímat další zprávy s daty. Toto je poměrně velký nedostatek, který byl při testování odhalen. Řešením může být použití tabletu s větší operační pamětí, nebo zjednodušení modelu robota.

9.1.3 Mračno bodů

Vykreslování mračna bodů představuje další výpočetně i paměťově náročnou operaci. Z testování vyplývá, že zobrazení modelu robota, modelu prostředí (URDF) a mračna bodů dohromady představuje přes 90 % z využitých prostředků aplikace. Mračno bodů je specifické ve vyšším nároku na procesor zařízení. To je způsobené jednak poměrně velkým objemem zpracovaných dat a také jejich častým překreslováním. Pro zvýšení odezvy aplikace by bylo vhodné omezit přenášená data, ať už snížením počtu přenášených bodů nebo frekvencí přijímaných dat. Ukázky vykreslení mračna bodů jsou na obrázcích 9.1 a 9.2.

9.1.4 Zobrazení dat ze skenerů

Komponenta pro zobrazení dat ze skenerů je nenáročná na paměť i procesor, zároveň je ale užitečná pro orientaci a zobrazení překážek. Podklad komponenty je průhledný a nepřekrývá tedy 3D scénu. Díky jednoduchosti zobrazení a přenášených dat je rychlost vykreslování poměrně vysoká.

Obrázek 9.4: Ukázka z testování aplikace - panel příkazů, manipulátor kamery a joystick.

9.1.5 Virtuální joystick a manipulátor kamery

Testování na Care-O-Botovi probíhalo pouze v simulaci a s nízkou simulovanou rychlostí (přibližně s násobkem 0,2x). Proto bylo použití ovládacích prvků poměrně těžkopádné a odezva odpovídala nízké rychlosti simulace. I když robot reagoval na pokyny z manipulátoru a joysticku správně, naměřená odezva v případě těchto ovládacích prvků nebyla příliš průkazná. Joystick i manipulátor kamery je znázorněn na obrázku 9.4.

9.1.6 Panel s příkazy

Pro Care-O-Bota existují předdefinované příkazy pro základní manipulaci s podnosem, ramenem, kamerami a dalšími jeho částmi. Tyto jednoduché příkazy jsou přístupné v panelu `Commands` (na obrázku 9.4). Robot na zadané příkazy reaguje téměř okamžitě a zadaná změna se brzy projeví i na modelu robota v aplikaci.

9.2 Turtlebot

Testování proběhlo na upravené variantě Turtlebota na Fakultě Informačních Technologií VUT v Brně (na obrázku 9.5) i na simulovaném robotovi v prostředí Gazebo. Pro Turtlebota se předpokládá zobrazení mračna bodů ze senzoru Kinect, modelu robota a obrazu kamery. Pro ovládání pohybu pak virtuální joystick. U robota Turtlebot se nepočítá s použitím modelu prostředí, i když by v případě potřeby bylo možné jej přidat v nastavení aplikace.

Na obrázku 9.6 je ukázka z testování aplikace na simulovaném robotovi Turtlebot.

9.2.1 Model robota

Model Turtlebota je výrazně jednodušší oproti Care-O-Botovi a jeho načítání a vykreslování je tedy podstatně méně náročné na paměť i procesor. Některé části modelu robota jsou však

Obrázek 9.5: Robot Turtlebut, použitý pro účely testování aplikace

stále třeba předem načítat z externích souborů a načtení může tedy trvat 30 - 60 sekund. Po načtení už se přenáší pouze transformace.

9.2.2 Mračno bodů

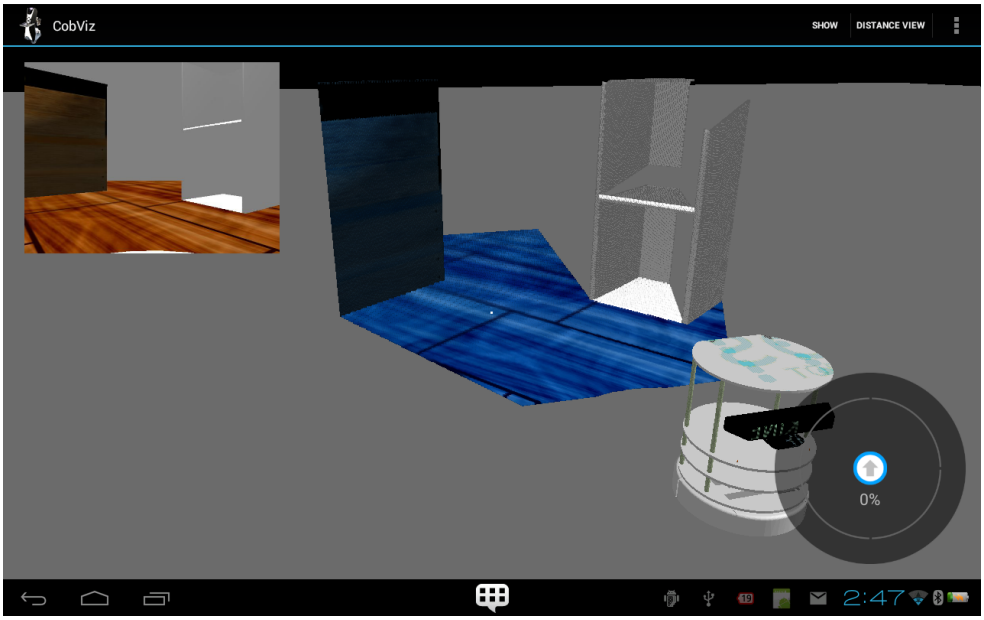
Podobně jako v případě Care-O-Bota, zobrazení mračna bodů je náročné a zpomaluje rychlost vykreslování 3D scény. V případě Turtlebota se však nezobrazuje model prostředí a jednodušší je i model robota. Celková odezva a rychlost vykreslování aplikace je velmi dobrá i v případě zobrazení mračna bodů.

9.2.3 Virtuální joystick

Díky celkově jednoduššímu nastavení a množství zobrazovaných dat je i odezva virtuálního joysticku velmi příznivá. Při testování robot reagoval téměř okamžitě na pokyn pohybu, stejně tak na pokyn zastavení. Vzhledem k tomu, že veškerá komunikace i zpracování přijímaných dat probíhá asynchronně, můžeme předpokládat, že srovnatelné odezvy dosáhneme i v případě použití na reálném robotovi Care-O-Bot nebo na dalších robotech.

9.2.4 Obraz z kamery

I když mračno bodů obsahuje data ve své podstatě shodná s obrazem kamery (pravděpodobně pochází i ze stejného zařízení - senzoru Kinect), při testování se ukázalo, že samotné mračno bodů v reálném prostředí může být poměrně fragmentované a nesouvislé. V případě robota Turtlebota je tedy užitečné i zobrazit živý náhled z kamery. Při testování byl obraz



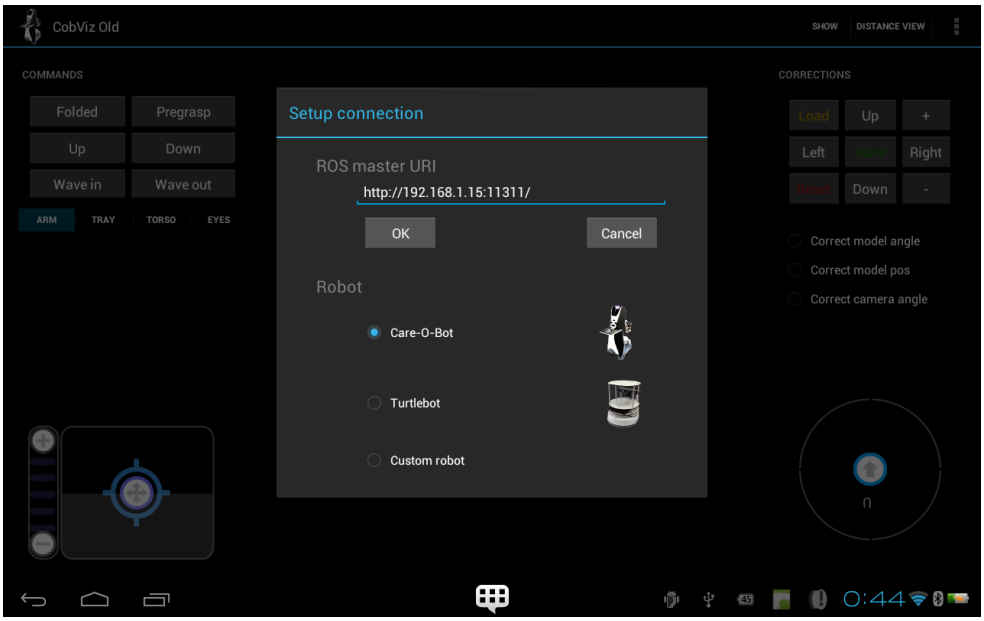
Obrázek 9.6: Ukázka z testování aplikace na Turtlebotovi.

přenášen v komprimované i nekomprimované variantě. I když je komprimovaná varianta méně náročná na množství přenášených dat, dekomprese a převod do formátu zobrazitelném v Android aplikaci představuje mírné navýšení nároků na výpočetní výkon. Ani jedna varianta však nemá na celkovou rychlost aplikace příliš velký vliv - v porovnání s dalšími prvky a zobrazenými daty je její vytížení poměrně zanedbatelné.

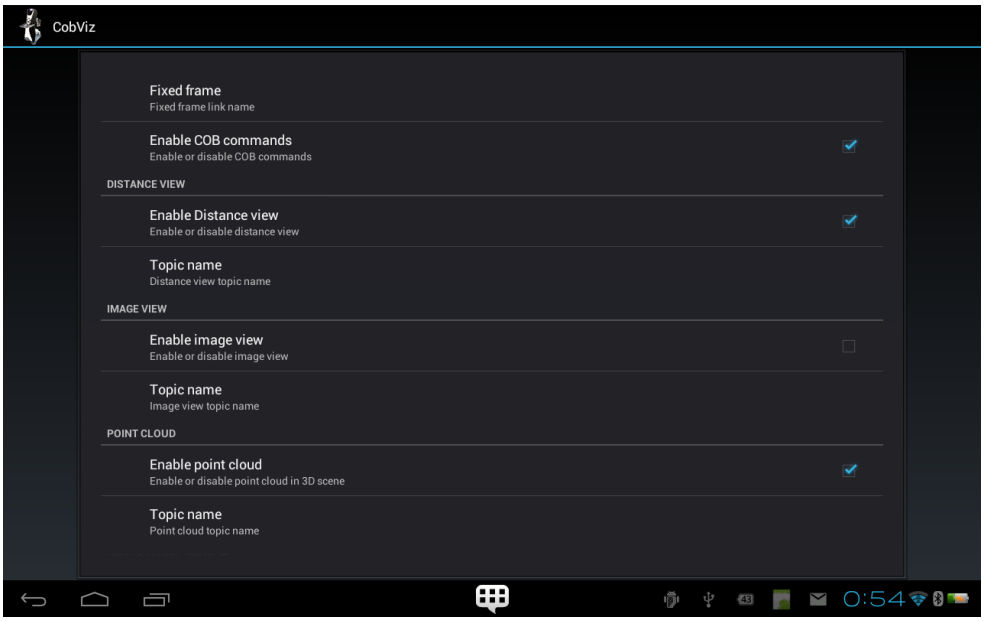
9.3 Nastavení aplikace

Nastavení aplikace umožňuje konfiguraci předdefinovaných robotů i použití odlišného robota. Po spuštění aplikace se zobrazí dialog s připojením k robotovi a s výběrem typu cílového robota (obrázek 9.7). Po připojení je načtena konfigurace pro daného robota, kterou je možné v nastavení změnit (obrázek 9.8). Pro konfiguraci jiného robota je třeba aplikaci spustit znovu a vybrat požadovaného robota v dialogu připojení.

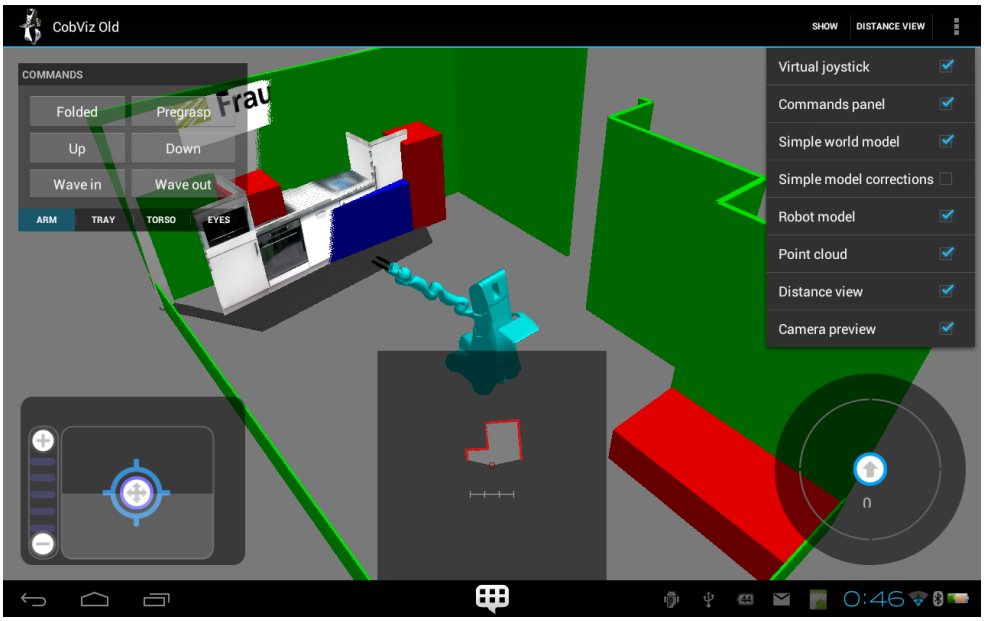
Jednotlivé komponenty je možné průběžně skrývat pomocí nabídky `show` na horním panelu aplikace (obrázek 9.9).



Obrázek 9.7: Ukázka dialogu připojení po spuštění aplikace.



Obrázek 9.8: Ukázka nastavení aplikace.



Obrázek 9.9: Ukázka volby zobrazení jednotlivých prvků.

Kapitola 10

Závěr

V počátku projektu jsem se seznámil s problematikou vzdáleného ovládání robotů a na základě zjištěných poznatků definoval cíle projektu. Hlavním cílem bylo vytvoření univerzálního rozhraní pro vzdálené řízení robotů využívajících platformu ROS. Dalším dílčím cílem bylo na základě univerzálního rozhraní vytvořit specifické rozhraní pro robota Care-O-Bot.

Návrh aplikace vycházel ze zjištěných poznatků v oblasti vzdáleného řízení s přihlédnutím ke specifickým podmínkám ovládání robota Care-O-Bot. Jako hlavní prvek uživatelského rozhraní byla zvolena 3D scéna kombinující zobrazení modelu robota, modelu prostředí, aktuálního mračna bodů a obrazu z kamery robota. Byl navržen vlastní způsob mapování obrazu na model prostředí a úspěšně implementován. Jako další užitečný prvek zobrazení slouží 2D pohled na data z laserových skenerů. K samotnému ovládání robota je využito virtuálního joysticku a komponenty pro natáčení torza robota.

Průběh implementace aplikace byl provázen několika komplexnějšími problémy, které bylo třeba vyřešit. Jako netriviální se ukázala kombinace různých dat do 3D scény. Bylo třeba získat z robota údaje o transformacích jednotlivých prvků a následně je aplikovat v 3D scéně, aby byly prvky vůči sobě korektně zobrazeny. Pro zobrazení dat bylo využito netradičního řešení - mapování obrazu na model prostředí. Byl navržen výpočet projekce obrazu na plochy modelu dle aktuální pozice robota a směru kamery. Tento výpočet byl implementován v jazyce GLSL pro vykreslení pomocí OpenGL. Realizovaná implementace je plně funkční a výpočet probíhá v reálném čase.

Výsledná aplikace byla úspěšně otestována s robotem Turtlebot v reálném prostředí a robotem Care-O-Bot simulovaným v prostředí Gazebo. Testy ukázaly, že v reálném použití je odezva ovládacích prvků velmi dobrá, ale zobrazení některých typů dat klade vysoké nároky na paměť a výpočetní výkon zařízení, což snižuje celkovou odezvu aplikace a rychlost vykreslování dat. Navržené uživatelské rozhraní je intuitivní a díky možnosti konfigurace umožňuje použití na různých robotických projektech.

Literatura

[1] Casado, F.: Development of control programs for the Turtlebot robot using ROS (Robot Operating System). Technická zpráva, Grupo de Robótica. Escuela de Ingenierías Industrial e Informática, 2012.

[2] open source community, T. E. F.: Eclipse Developer IDE. 2013.
URL http://www.eclipse.org

[3] GmbH, N.: Neobotix - Robotics & Automation. 2012.
URL http://www.neobotix-roboter.de/

[4] Graf, B.; Reiser, U.; Hagele, M.; aj.: Robotic home assistant Care-O-bot 3 - product vision and innovation platform. In *Advanced Robotics and its Social Impacts (ARSO), 2009 IEEE Workshop on*, nov. 2009, s. 139 –144, doi:10.1109/ARSO.2009.5587059.

[5] JetBrains: IntelliJ IDEA. 2013.
URL http://www.jetbrains.com/idea/

[6] Koenig, N.; Howard, A.: Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, ročník 3, sept.-2 oct. 2004, s. 2149 – 2154 vol.3, doi:10.1109/IROS.2004.1389727.

[7] Kohler, D.: An implementation of ROS in pure Java with Android support. Květen 2011.
URL http://code.google.com/p/rosjava/

[8] Maple, C.; Yue, Y.; Li, D.; aj.: Case study: Multi-role shadow robotic system for independent living. In *High Performance Computing and Simulation (HPCS), 2012 International Conference on*, july 2012, str. 415, doi:10.1109/HPCSim.2012.6266950.

[9] Michaud, F.; Boissy, P.; LabontĂ©, D.; aj.: Exploratory design and evaluation of a homecare teleassistive mobile robotic system. *Mechatronics*, ročník 20, č. 7, 2010: s. 751 – 766, ISSN 0957-4158, doi:10.1016/j.mechatronics.2010.01.010, ¡ce:title¿Special Issue on Design and Control Methodologies in Telerobotics¡/ce:title¿.
URL http://www.sciencedirect.com/science/article/pii/S0957415810000280

[10] Neumann, U.; You, S.; Hu, J.; aj.: Augmented virtual environments (AVE): dynamic fusion of imagery and 3D models. In *Virtual Reality, 2003. Proceedings. IEEE*, 2003, ISSN 1087-8270, s. 61–67, doi:10.1109/VR.2003.1191122.

[11] Nielsen, C.; Goodrich, M.; Ricks, R.: Ecological Interfaces for Improving Mobile Robot Teleoperation. *Robotics, IEEE Transactions on*, ročník 23, č. 5, 2007: s. 927–941, ISSN 1552-3098, doi:10.1109/TRO.2007.907479.

[12] Quigley, M.; Conley, K.; Gerkey, B. P.; aj.: ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*, 2009.

[13] Radev, D.: UI_PRI for the SRS Project. In *Twenty-First International Conference "Robotics and Mechatronics"*, september 2011.

[14] Reitmayr, G.; Drummond, T.: Going out: robust model-based tracking for outdoor augmented reality. In *Mixed and Augmented Reality, 2006. ISMAR 2006. IEEE/ACM International Symposium on*, 2006, s. 109–118, doi:10.1109/ISMAR.2006.297801.

[15] Schunk: SCHUNK Mobile Greifsysteme GmbH. 2012.
URL http://www.schunk-modular-robotics.com/

[16] Strecha, C.; Fransens, R.; Van Gool, L.: Combined Depth and Outlier Estimation in Multi-View Stereo. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, ročník 2, 2006, ISSN 1063-6919, s. 2394–2401, doi:10.1109/CVPR.2006.78.

[17] Zhang, L.; Curless, B.; Seitz, S.: Spacetime stereo: shape recovery for dynamic scenes. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, ročník 2, 2003, ISSN 1063-6919, s. II–367–74 vol.2, doi:10.1109/CVPR.2003.1211492.

Příloha A

Použití aplikace

I když samotná aplikace je poměrně jednoduchá a její ovládání nevyžaduje hlubších odborných znalostí, pro použití s platformou ROS a správné nastavení prostředí je zde uvedena tato kapitola, jako jednoduchý návod a vysvětlení souvisejících pojmů. Dále bude vysvětleno vše nutné k instalaci a spuštění simulace robota a vzdálené připojení aplikace.

A.1 Instalace ROSu

Instalace ROSu je možná provést dvěma způsoby. První, jednodušší varianta, je použít již zkompilované balíky pro linuxovou distribuci Ubuntu. Ty je možné nainstalovat běžným nástrojem *apt*:

```
sudo apt-get install ros-fuerte-desktop-full
```

Instalalce ROSu je poměrně náročná a může tedy trvat i několik hodin, dle rychlosti připojení k internetu. Po dokončení instalace samotného ROSu je třeba nainstalovat potřebné balíky pro Care-O-Bota. Opět lze využít nástroje *apt*.

```
sudo apt-get install ros-fuerte-care-o-bot
```

Druhá alternativa instalace je kompilace přímo ze zdrojových kódů. To dává vývojáři větší kontrolu nad všemi balíky s možností úpravy libovolného balíku. Tato varianta je však časově velmi náročná, protože je nutné stáhnout velké množství různých balíků, kdy každý balík může používat další rozdílné nástroje. I samotná kompilace všech balíků a samotného jádra ROSu může zabrat několik hodin. Poslední možností je obě varianty zkombinovat. Díky nástroji *rosws* je možné do funkční instalace ROSu přidávat balíky nové, či použít již stávající balíky, ale zkompilované ze zdrojových kódů. Tento nástroj pak nejdříve zjistí, zda balík existuje v uživatelském adresáři a pak teprve prohledává standardní instalační adresář ROSu. Tato varianta je velmi výhodná, pokud vývojář chce provést pouze menší úpravy v již existujících balících, aniž by musel kompilovat kompletní zdrojové kódy.

A.2 Nastavení proměnných

Pro zajištění správné komunikace po síti a spuštění funkční simulace Care-O-Bota je nutné nastavit několik proměnných prostředí. Pro nastavení všech základních proměnných lze použít výchozí skript:

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

V případě více použitých distribucí pak lze tyto proměnné načítat zvlášť při každém spuštění terminálu:

```
source /opt/ros/fuerte/setup.bash
```

Dále je nutné pro správné spuštění simulace nastavit proměnné *ROBOT* a *ROBOT_ENV*:

```
export ROBOT=cob3-3
export ROBOT_ENV=ipa-kitchen
```

Protože se běžně instance ROSu spouští lokálně, v případě vzdáleného přístupu je třeba přenastavit výchozí parametr *ROS_MASTER_URI* z původní podoby

```
export ROS_MASTER_URI=http://localhost:11311
```

na IP adresu stroje přístupnou ze sítě, např.

```
export ROS_MASTER_URI=http://192.168.43.101:11311
```

Port lze nastavit libovolně, ale standardně se používá port *11311*. V případě přímé komunikace mezi PC a Android zařízením bohužel není samotné Android zařízení schopné přeložit doménové jméno (hostname), které mu počítač nabízí. Proto je nutné ještě dále použít oba parametry:

```
export ROS_IP=192.168.43.101
export ROS_HOSTNAME=http://192.168.43.101
```

A.3 Spuštění simulace

Po správném nastavení všech proměnných můžeme spustit instanci ROSu

```
roscore
```

V dalším okně pak můžeme spustit samotnou simulaci Care-O-Bota

```
roslaunch cob_bringup_sim robot.launch
```

Výsledkem by mělo být otevřené okno se simulátorem Gazebo a zobrazeným robotem v prostředí *ipa_kitchen*. Dále je možné spustit nástroje pro ovládání robota

```
roslaunch cob_bringup dashboard.launch
```

A.4 Spuštění aplikace

Pro úspěšnou kompilaci aplikace je nutné:

- Vývojové prostředí pro Android (Eclipse IDE + Android SDK)
- Zkompilovaná knihovna Rosjava
- Všechny potřebné balíky v ROSu, na kterých je Rosjava závislá

Po úspěšné kompilaci v prostředí Eclipse s ADT pluginem je aplikace automaticky nainstalována k připojenému zařízení a spuštěna. Po spuštění aplikace na Android zařízení se zobrazí editační pole. Do něj je třeba zadat správnou IP adresu PC, na kterém je spuštěna simulace, včetně zadaného portu. Pokud po spuštění nejsou zobrazována žádná data, je vhodné prověřit např. nástrojem PING, zda zařízení jsou schopna navázat komunikaci.