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

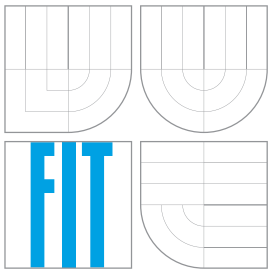
HLASOVÉ OVLÁDÁNÍ PRO ROBOTA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE PAVEL HÁJEK
AUTHOR

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

HLASOVÉ OVLÁDÁNÍ PRO ROBOTA

ROBOT CONTROL WITH VOICE COMMANDS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PAVEL HÁJEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MICHAL ŠPANĚL, PhD.

BRNO 2015

Abstrakt

Tato bakalářská práce se zabývá návrhem a tvorbou aplikace pro hlasové ovládání robota se zaměřením na platformu PR2. Práce popisuje proces rozpoznávání řeči a věnuje se robotickému operačnímu systému pro nějž je určená. Vytvořená aplikace používá knihovnu Sphinx-4 k převodu řeči na text, který dále publikuje v rámci ROSu. Celá aplikace je napsaná v jazyce Java. Následně se práce věnuje otestování aplikace. Ukázalo se, že pro ovládání robota sadou jednoduchých povelů, je nejvhodnější si vytvořit vlastní gramatiku.

Abstract

This bachelor's thesis describes design and development of a robot voice control application aiming at platform PR2. It explains the procecess of voice recognition and describes the robot operating system as a target platform. The application uses library Sphinx-4 to translate speech to text, which is than published in ROS. In the end, the set of tests is described. Tests discovered, that the best way to control robot with a few simple commands is developing own grammar.

Klíčová slova

ROS, PR2, HARK, Sphinx-4, GoogleSpeech API, rozpoznávání hlasu, rosjava

Keywords

ROS, PR2, HARK, Sphinx-4, GoogleSpeech API, voice recognition, rosjava

Citace

Pavel Hájek: Hlasové ovládání pro robota, bakalářská práce, Brno, FIT VUT v Brně, 2015

Hlasové ovládání pro robota

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Španěla PhD. Uvedl jsem všechny literární prameny a publikace, z nichž jsem čerpal.

.....
Pavel Hájek
17. května 2015

Poděkování

Rád bych poděkoval vedoucímu Ing. Michalu Španělovi PhD. za vedení práce, rady, trpělivost a shovívavost.

Obsah

Kapitola 1

Úvod

Rozpoznávání hlasu je jedno z nejaktuálnějších témat nejen robotiky, ale i dalších počítačových systémů, neboť řeč je snad nejzákladnějším lidským dorozumívacím prostředkem. Systémy pro hlasové ovládání musí být velmi pokročilé, neboť samotný převod řeči na text je velmi obtížný proces, stejně jako následující sémantická analýza výstupního textu. Zvláštní důraz se klade na přesnost rozpoznávání a následnou rychlou reakci ovládaného systému. Další požadavky jsou pak nezávislost na mluvčím, filtrování šumu a reakce pouze na určité příkazy.

Hlavním cílem této práce je spojení komponent pro rozpoznávání řeči a ovládání robota a jejich následné otestování simulačními nástroji a aplikace na reálném robotu. Jedná se především o návrh a implementaci komplexního systému, který očekává na vstupu řeč a jeho výstupem je odpovídající reakce na příkaz. Reakcí na příkaz je myšleno buď přímé vykonání jednoduchého příkazu, nebo poskytnutí rozpoznaných dat dalším aplikacím.

Tato práce používá k rozpoznávání řeči knihovnu Sphinx-4, vyvíjenou univerzitou Carnegie Mellon v Pittsburghu. Ta umožňuje vyvíjet a implementovat různé aplikace pro strojové rozpoznávání řeči a navíc je šířena pod svobodnou licencí typu BSD, což umožňuje její modifikaci na míru cílovému systému.

Jako platforma je použit robotický operační systém (ROS) fungující jako middleware prostředí pro vývoj programů pro práci s roboty. Ten nám nabízí mnoho již vytvořených programů a knihoven pro ovládání robota.

Práce se věnuje nejprve teorii z oblasti rozpoznávání řeči, včetně popisu samotného procesu rozpoznávání, robotického operačního systému, popisu jeho nejdůležitějších částí a tvorby softwaru pro něj. Kapitoly 3 a 4.4 popisují současný stav problematiky a jiné nástroje sloužící podobnému účelu. Další kapitoly se pak věnují detailní analýze návrhu řešení a implementace, popisují způsoby spouštění a prezentují dosažené výsledky kvality převodu řeči na text. V kapitole Testování a experimenty jsou popsány způsoby testování aplikace a jejich zhodnocení. Testy dokazují, že je nejvhodnější stát před mikrofonem čelem k němu a pro ovládání robota sadou jednoduchých povelů je vhodné vytvořit si vlastní gramatiku, naopak pro komplexnější například dialogový systém se více hodí použít servery GoogleSpeech. V poslední kapitole je pak celý program zhodnocen a jsou diskutována možná budoucí rozšíření.

Kapitola 2

Rozpoznávání hlasu

Tato kapitola poskytuje stručný přehled o problematice rozpoznávání řeči a věnuje se popisu tohoto procesu.

Většina rozpoznávacích nástrojů je založena na HMM[1], což je typ statistického modelu. V systémech založených na HMM je každý foném reprezentován statistickým modelem obsahujícím údaje o něm. Tyto modely se nazývají akustické.

Rozpoznávání řeči je problém nalezení sekvence slov, které nejlépe odpovídají řeči na vstupu. Jedná se tedy o problém vyhledávání, v případě systémů založených na HMM je to problém vyhledávání v grafu.

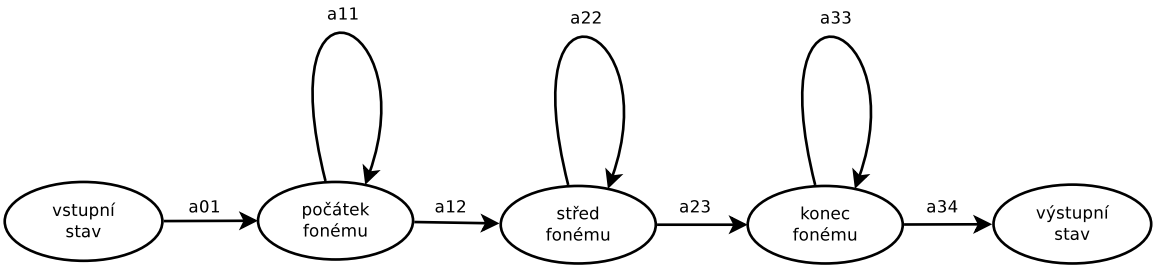
2.1 Skrytý Markovův model

HMM (hidden Markov model) pro obecný foném je zobrazen na obrázku 2.1. Tento model se skládá ze dvou stavů - vstupní a výstupní -, které slouží k vložení do vyhledávací sítě, a z několika dalších stavů sloužících pro popis struktury fonému [10].

Hrany mezi uzly jsou ohodnoceny přechodovými pravděpodobnostmi. Platí, že součet výstupních přechodových pravděpodobností je vždy roven jedné, tedy [10]:

$$\sum_j a_{ij} = 1 \tag{2.1}$$

Hodnota a_{ij} je pravděpodobnost přechodu z i-tého prvku do j-tého prvku.



Obrázek 2.1: HMM obecného fonému.

[1]Hidden Markov models - Skryté Markovovy modely

2.2 Data potřebná pro rozpoznávání

Z pohledu uživatele je nutno knihovně dodat následující položky.

Akustický model

Soubory s akustickými modely obsahují přepisy jednotlivých fonémů na HMM. Vzhledem k tomu, že fonémy jsou specifické pro každý jazyk, tak i akustické modely fungují pouze pro jeden jazyk, pro který byly vytvořeny [6]. Seznam oficiálních akustických modelů pro knihovnu Sphinx-4 lze najít online[2].

Slovník

Slovník obsahuje přepis slov ve formě fonémů. Typicky se jedná o textový dokument ve formátu:

HELLO HH AH L OW
WORLD W AO R L D

tedy slovo následované jeho přepisem na jednotlivé fonémy.

Gramatika

Gramatika může být ve formě jazykového modelu nebo uživatelem definovaná.

Jazykové modely odráží gramatiky jednotlivých jazyků. Zabudování takové gramatiky do modelu je ovšem vzhledem k rozsáhlosti jazyků velmi složité. Tento problém je řešen statisticky. Během vytváření a trénování jazykových modelů se shromáždí až stovky hodin záznamů řeči, ze kterých se získají statistiky o výskytu jednotlivých n-tic slov (typicky dvojic nebo trojic, záleží na stupni jazykového modelu). Ty se pak vloží do modelu [5].

Protiklad tvoří uživatelsky definované gramatiky, které jsou vhodné pro úzké tematické okruhy, jako například hlasové příkazy pro řízení robota. Knihovnou Sphinx-4 podporovaný formát gramatik je například JSGF. Seznam všech formátů je dostupný online[3].

Speciálním případem rozpoznávání řeči je tzv. KWS (key word spotting) - vyhledávání klíčových slov. V tomto případě se místo gramatiky dodá rozpoznávači pouze seznam těchto klíčových slov, které jsou ve zvukovém signálu vyhledávány.

2.3 Proces rozpoznávání řeči

Proces rozpoznávání sestávání z několika fází:

- parametrizace,
- akustické přiřazení,
- dekódování.

[2] http://sourceforge.net/projects/cmusphinx/files/Acoustic%20and%20Language%20Models/

[3] http://cmusphinx.sourceforge.net/doc/sphinx4/edu/cmu/sphinx/linguist/language/grammar/Grammar.html

2.3.1 Parametrizace

Cílem parametrizace je zejména snížení objemu a odfiltrování dat, která nejsou důležitá pro další rozpoznávání. Slouží k vyhledání nejdůležitějších příznaků v řeči vhodných pro další fázi. V této fázi se zvukové signály transformují na sekvenci vektorů, která charakterizuje daný úsek vstupního signálu. Tato sekvence vektorů se nazývá rys (feature).

2.3.2 Akustické přiřazení

Cílem této fáze je přiřazení rysu k HMM určitého akustického modelu fonému. Toto přiřazení odstraňuje dva problémy související s mluveným slovem. První z nich je problém trvání fonémů, tedy že stejný foném nikdy netrvá stejně dlouhou dobu. Druhý problém souvisí s různě znějícími lidskými hlasy.

2.3.3 Dekódování

Dekodér vždy na začátku rozpoznávání sestaví rozpoznávací síť - orientovaný graf, je zde tedy řešen problém vyhledávání v grafu. Tento graf reprezentuje všechny možné sekvence zadaného jazyka (určené gramatikou) a sestává z jednotlivých HMM popisujících příslušné fonémy. Grafy dekodérů bývají velmi rozsáhlé. Na CD je vložen obrázek ("linguist.png") zobrazující vygenerovanou rozpoznávací síť ke gramatice z kapitoly 7.5, na kterém je tato rozsáhlost patrná i pro relativně malou gramatiku.

Kapitola 3

Knihovny pro převod řeči na text

Následující dvě kapitoly se věnují knihovnám vyvíjeným na univerzitě Carnegie Mellon v Pittsburgu jako součást projektu zabývajícího se převodem řeči - CMU Sphinx. Další součást projektu - SphinxTrain - se zabývá nástroji pro práci s akustickými modely (jejich vytváření a trénování) atd.

3.1 Pocketsphinx

První z nich je knihovna pocketsphinx. Jedná se o malou nenáročnou knihovnu určenou pro mobilní a vestavěná zařízení, používá se nicméně i na klasických počítačích. Knihovna je z důvodu nenáročnosti a přenositelnosti i na vestavěná zařízení napsaná v jazyce C.

Jejími nedostatky jsou poměrně nízká úspěšnost ve srovnání s druhou zmiňovanou knihovnou[1].

3.2 Sphinx-4

Jedná se o knihovnu napsanou kompletně v Javě, která poskytuje podporu pro veškeré činnosti spojené s převodem řeči na text.

3.2.1 Architektura knihovny Sphinx-4

Celková architektura je zobrazena na obrázku 3.1.

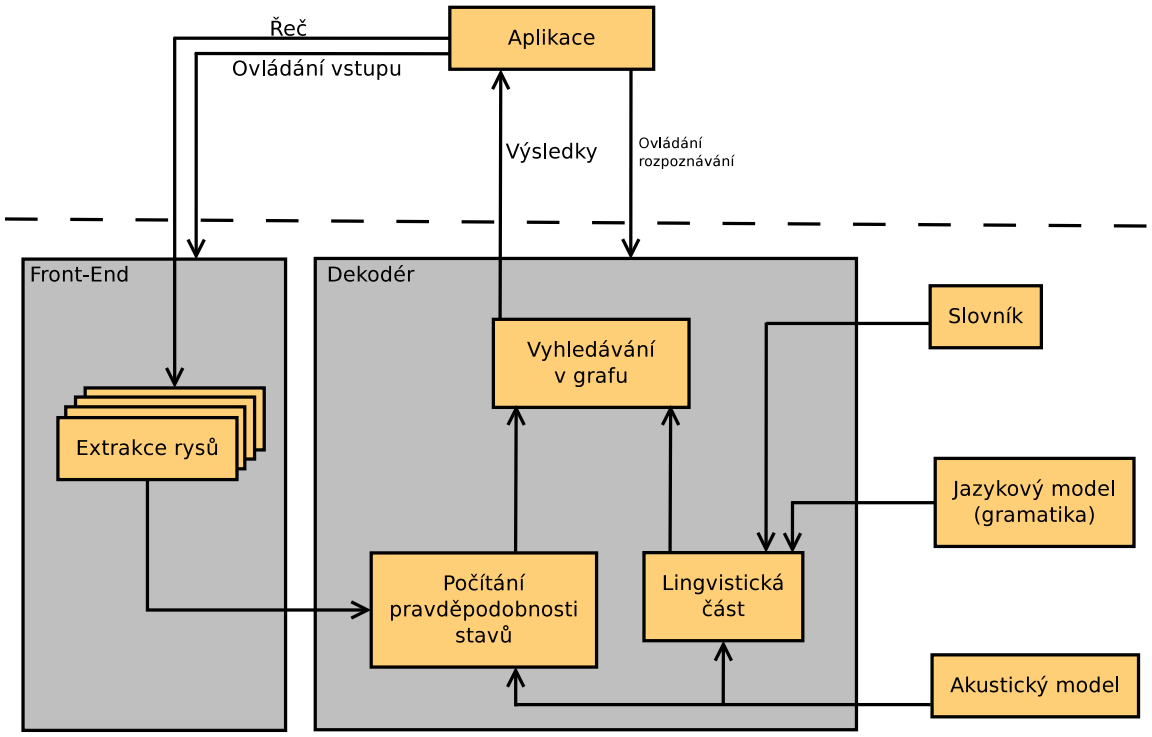
Nejdůležitější části knihovny tvoří Front-End a Dekodér [8]. Front-End přijímá zvuková data (například z mikrofonu), získá z nich rysy signálu a poskytuje je dekodéru. Úkolem dekodéru je vybrat množinu několika pravděpodobných následujících uzlů grafu a následně ohodnotit vstup z Front-Endu s ohledem na ně.

3.2.2 Front-End

Tento modul sestává z několika vzájemně komunikujících bloků. Každý blok má vstup a výstup, přičemž každý vstup je připojen na výstup bloku předešlého. Jednotlivé bloky tohoto modulu označí ve zvukovém signálu části obsahující řeč, vyjmou je a následně je parametrizují [8].

Všechny třídy implementující jednotlivé bloky dědí ze třídy $edu.cmu.sphinx.frontend.BaseDataProcessor$.

[1] http://www.jaivox.com/pocketsphinx.html



Obrázek 3.1: Celková architektura knihovny Sphinx4.

3.2.3 Dekodér

Jak bylo zmíněno výše (2.3), nejprve dekodér sestaví graf. Sestavení se skládá z několika částí. Na začátku se sestaví graf, jehož uzly jsou množiny slov, u kterých se očekává, že budou v určitém kontextu (daném polohou v grafu) vysloveny. Systém dále pokračuje s dekompozicí grafu, dokud uzly netvoří jednotlivé HMM fonémů. Hrany mezi uzly pak obsahují informace o jazykové a akustické pravděpodobnosti přechodů mezi uzly. Při jeho vytváření čerpá postupně informace z gramatiky, ze slovníku a z akustického modelu [9].

Zároveň slouží dekodér knihovny Sphinx-4 i k akustickému přiřazování. Sestává ze třech částí:

- **Lingvistická část** (Linguist) zodpovědná za sestavení grafu,
- **Search Manager**, který vyhledává v grafu,
- **Accoustic Scorer** je část, která komunikuje s Front-Endem. Přijatá data ve formě rysů zvukového signálu ohodnotí a hodnocení pošle Search Manageru.

3.2.4 Konfigurace knihovny

Pro detailní nastavení činnosti knihovny slouží konfigurační soubor ve formátu XML. Tento soubor obsahuje seznam všech součástí knihovny, které se mají podílet na rozpoznávání v definovaném pořadí včetně jejich konfigurace. V tomto souboru tedy uživatel nastavuje mezi jinými i typ gramatiky, cesty k potřebným souborům atd.

Kořenový element konfiguračních souborů nese tag *config*. Jeho potomky tvoří elementy označeny *component*, které definují jednotlivé komponenty knihovny. Tyto elementy

obsahují atributy *name*, které slouží ke vzájemnému odkazování v rámci jednoho souboru, a atribut *type*, který obsahuje název třídy (včetně balíčků), jejíž instancí má komponenta být.

Elementy *component* mohou obsahovat další potomky, typicky se jedná o podelementy s tagem *property*, které mají atributy *name* a *value*. Tyto podelementy slouží k předávání hodnot konstruktorům tříd komponent.

V případě konfigurace front-endu element *component* obsahuje podelement *propertylist*, jehož podelementy s označením *item* obsahují jako hodnotu jméno bloku. Toto jméno musí odpovídat právě jednomu atributu *name* komponenty definované ve stejném souboru.

Ukázka jednoduchého konfiguračního souboru je na obrázku 3.2. V případě tohoto nastavení bude knihovna Sphinx-4 sloužit jako voice-activity detector.

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
        <component name="endpointer" type="edu.cmu.sphinx.frontend.FrontEnd">
                <propertylist name="pipeline">
                        <item>microphone</item>
                        <item>dataBlocker</item>
                        <item>speechClassifier</item>
                        <item>speechMarker</item>
                </propertylist>
        </component>

        <component name="microphone"
                type="edu.cmu.sphinx.frontend.util.Microphone">
                <property name="closeBetweenUtterances" value="false"/>
                <property name="bigEndian" value="false"/>
        </component>

        <component name="dataBlocker" type="edu.cmu.sphinx.frontend.DataBlocker"/>

        <component name="speechClassifier" type="edu.cmu.sphinx.frontend.
                                endpoint.SpeechClassifier"/>

        <component name="speechMarker" type="edu.cmu.sphinx.frontend.
                                endpoint.SpeechMarker">
                <property name="speechLeader" value="200"/>
                <property name="speechTrailer" value="200"/>
        </component>
</config>
```

Obrázek 3.2: Jednoduchý konfigurační soubor pro VAD.

3.2.5 Knihovna Sphinx-4 jako voice activity detector

Knihovna Sphinx-4 může při vhodné konfiguraci sloužit i jako VAD. V tomto případě bude konfigurační soubor obsahovat informace jen o front-endu. Zapojením pouze určitých bloků

může uživatel dosáhnout pouhé extrakce řečového signálu a nikoliv i jeho následné parametrizace. V programu potom uživatel získává řečová data z posledního bloku front-endu.

Tuto vlastnost knihovny je vhodné využít při rozpoznávání pomocí Google Speech API, kdy by bylo nevhodné posílat na server všechna zachycená data z mikrofonu.

3.3 Google Speech API

Při používání této metody jsou požadavky posílány na adresu $https://www.google.com/speech-api/v2/recognize$. Pomocí GET parametrů je potřeba zaslat několik dalších informací:

- **output** - nastavuje formát výstupu, používá se json,
- **lang** - označuje jazyk řeči, pokud je tento parametr vynechán, očekává se angličtina,
- **key** - slouží k zaslání klíče.

Klíč je jediný povinný parametr, slouží k omezení přístupu na padesát požadavků denně. Klíč si je potřeba nechat vygenerovat, návody na to jsou dostupné online[2].

Metodou POST jsou pak poslány zvuková data. Pro jejich správnou interpretaci je potřeba vhodně nastavit identifikátor souboru Content-Type. V případě běžných PCM 16-bitových dat z mikrofonu o frekvenci 16kHz je to $audio/l16;rate = 16000;$.

Bohužel se mi nepodařilo najít, jak je rozpoznávání servery implementováno, nicméně nějaké kusé informace lze najít online[3].

Velkou nevýhodou tohoto řešení je omezení na padesát požadavků denně, což je pro jakoukoliv interakci (nejen s robotem) silně nedostačující. Nicméně tento nedostatek lze obejít vygenerováním více klíčů a jejich střídáním.

[2] např.: http://www.chromium.org/developers/how-tos/api-keys
[3] http://research.google.com/pubs/SpeechProcessing.html

Kapitola 4

Robot Operating System

ROS je soubor knihoven a nástrojů fungující jako middleware prostředí pro vývoj programů pro práci s roboty. K tomuto účelu poskytuje mj. hardwarovou abstrakci, ovladače zařízení, podporu pro meziprocesovou komunikaci, simulační nástroje atd [1]. Umožňuje tedy roboty ovládat, číst data ze senzorů a ihned je zpracovávat, plánovat veškeré akce a mnoho dalšího. Programy pro ROS je možné psát v několika jazycích - např. C++, python, lisp, Java.

Pro práci s tímto middlewarem je nutné, aby běžel program *roscore*, který tvoří jeho jádro. Jednou z jeho nejdůležitějších funkcí je zajištění meziprocesové komunikace, na které je závislá většina programů.

Program využívající služeb ROSu se nazývá uzel (node) [3]. Tyto programy jsou součástí balíčků, s jejichž pomocí se i spouští. Jednotlivé uzly spolu pak komunikují. V ROSu existují dva mechanismy pro vzájemnou komunikaci procesů - topicy a služby. Topicy slouží pro jednosměrnou komunikaci, služby pak umožňují komunikaci typu dotaz-odpověď. V této práci jsou využity pouze topicy.

4.1 ROS Topics

Topic je kanál mezi publisherem (tím, co na kanál zapisuje) a subscriberem (tím, co z kanálu čte). Topicy tedy slouží k jednosměrné komunikaci mezi několika uzly, kdy každý z nich před připojením deklaruje, jestli bude z topicu číst, nebo na něj zapisovat [4]. Komunikace probíhá formou výměny zpráv. Zpráva je jednoduchá datová struktura obsahující prvky různých datových typů, případně pole s nimi. Navíc mohou obsahovat i jednoduché datové struktury, vycházející z jazyka C[1] [2].

Při připojování na topic je třeba specifikovat požadovaný typ zpráv. Ten je možné zvolit z předem definovaných[2] nebo si vytvořit nový pomocí utility *rosmsg*.

4.2 ROS balíčky

I když lze programy pro ROS distribuovat samostatně, například jako pouze zdrojový text, doporučuje se používat systém balíčků *catkin*, který nahrazuje předchozí systém *rosbuild*. Tomu se ale v tomto textu nebudu věnovat. Systém *catkin* je sada nástrojů sloužící pro vytváření, překlad a instalování balíčků, využívající nástroj pro automatický překlad CMake.

[1] http://wiki.ros.org/Messages
[2] http://wiki.ros.org/std_msgs

Základním prvkem balíčků je pracovní adresář (catkin workspace). Jedná se o adresář, který má v základní podobě pouze podadresář *src*, ve kterém jsou samotné balíčky.

Balíček je adresář obsahující implementaci uzlů, hlavičkové a konfigurační soubory, knihovny a definice zpráv. Každý balíček musí obsahovat nejméně dva soubory:

- **manifest.xml** obsahující informace o balíčku - název, autor, dotum, verze, případné závislosti na dalších balíčcích a další,
- **CMakeLists.txt** obsahující informace nutné k sestavení balíčku. Dále obsahuje informace potřebné k nainstalování balíčku - soubory k překopírování s jejich cílovým umístěním a informace k dalším činnostem.

Dále pak často obsahuje složku *src*, která obsahuje zdrojové soubory. Překlad balíčků se provádí programem *catkin_make* zavolaným z adresáře pracovní plochy. Program sám vyhledá všechny dostupné balíčky a následně je postupně zkompiluje, resp. vykoná činnosti definované v jednotlivých CMakeLists.txt souborech.

4.3 ROS Java

Balíček rosjava obsahuje kompletní implementaci služeb ROSu dostupné v jazyce Java.

4.3.1 rosjava balíčky

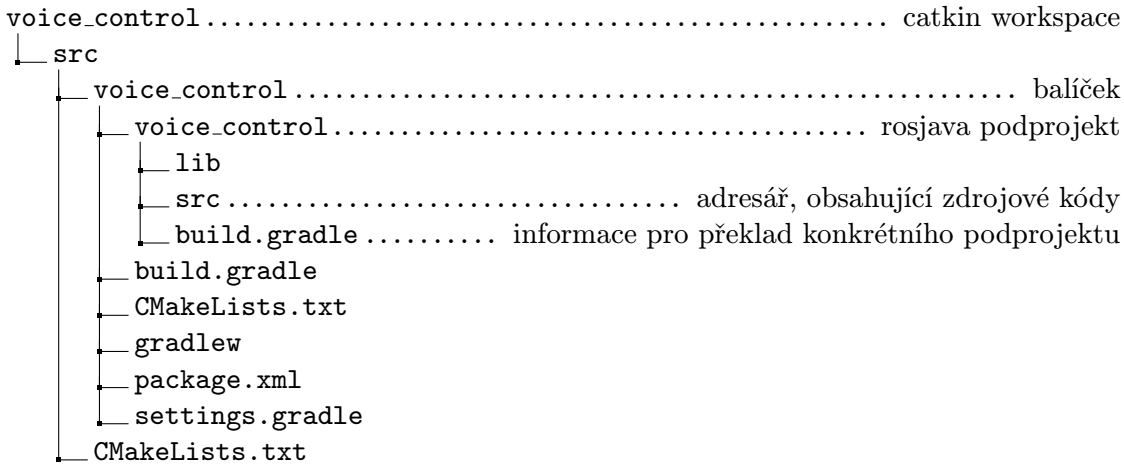
Adresářová struktura balíčků rosjava přejímá strukturu obyčejných ROS balíčků. Navíc však tato složka obsahuje soubory utility Gradle, která slouží k automatizaci překladu a dalších úkonů s ním spojených. Jedná se o tyto soubory:

- **gradle** - adresář obsahující jar archiv - hlavní program utility,
- **gradlew** - spouštěcí skript napsaný v jazyce bash,
- **build.gradle** - soubor obsahující informace potřebné k sestavení balíčků,
- **settings.gradle** - soubor s informacemi o podprojektech ke kompilaci (viz dále).

Ve většině případů nemusí uživatel tyto soubory jakkoliv upravovat s výjimkou vyplnění hlavní třídy obsahující metodu *main*, které je předáno řízení programu po spuštění aplikace.

Další rozdíl oproti klasickým balíčkům je ten, že neobsahuje přímo složku src se zdrojovými kódy. Místo ní obsahuje adresáře, které odpovídají podprojektům. Tyto složky s podprojekty už obsahují složku src se zdrojovými kódy, navíc obsahují soubor build.gradle, ve kterém je definovaná hlavní třída pro spuštění zkompilovaného archivu, a také knihovny potřebné pro překlad a běh programu. Tyto knihovny se typicky vkládají do složky lib. Příklad takové hierarchie balíčku, který je založen na výsledku této práce, je zobrazen na obrázku 4.1.

Překlad těchto balíčků probíhá stejně jako překlad balíčků klasických, utilita cmake sama spouští program Gradle.



Obrázek 4.1: Diagram s adresářovou strukturou balíčku.

4.4 Nástroje pro ovládání robota hlasem

Následující kapitoly se věnují již existujícím řešením pro platformu ROS. Ani jeden z těchto programů nedokáže přímo vykonat příkaz, pouze zapisovat výsledky na definovaný topic. Zároveň i názvy topiců jsou předem definovány autory těchto programů. Oba dva tyto nedostatky jdou zásahy do zdrojových kódů odstranit. U těchto aplikací jsou zhodnoceny jejich pozitiva a negativa.

4.4.1 gspeech

Tento program umožňuje zachycení řeči a její následné odeslání na servery Google Speech k rozpoznání, které po zpracování odeslané nahrávky vrátí výsledek a jeho pravděpodobnost. V současné době program nefunguje, neboť využívá služeb Google Speech API v1, které firma Google v dubnu roku 2014 odstavila.

Jeho největší nevýhoda je závislost na internetovém připojení, které musí být zároveň dostatečně rychlé, aby časový rozptyl mezi zachycením zvuku a obdržením odpovědi byl co nejmenší.

Jeho výhodou je pak vysoká úspěšnost rozpoznávání, která je daná úspěšností serverů Google Speech.

4.4.2 pocketsphinx

Balíček *pocketsphinx* funguje jako wrapper nad knihovnou *pocketsphinx*. Funguje podobně jako předešlá aplikace s rozdílem, že nikam neposílá zachycená audio data, ale předává je knihovně pocketsphinx, která je rozpozná a vrátí výsledek. Rozpoznaná data jsou pak opět publikována na definovaný topic.

Výhoda tohoto programu je nezávislost na internetovém připojení, může tedy pracovat i na systémech offline. Nevýhodou je pak poměrně nízká úspěšnost rozpoznávání daná úspěšností knihovny pocketsphinx.

Kapitola 5

Robot PR2

PR2 (zkratka pro Personal Robot 2) je humanoidní robot určený pro výzkumné účely.

Tento robot se skládá ze základny, která umožňuje pohyb všemi směry, výsuvného torza, ramen a hlavy. Školní verze robota má na sobě navíc připojený Kinect, jehož mikrofony jsou v této práci použity. O provoz robota se starají dva počítače, oba s procesory Quad-Core i7 Xeon a 24 GB RAM. Navíc je na základně umístěn lidar Hokuyo UTM-30LX.

Torzo nesoucí ramena a hlavu je výsuvné v rozsahu 315 mm. Výška robota se tedy pohybuje mezi 133 a 164.5 cm. Na vrchní část torza je umístěn stejný lidar, jako na základně.

Robotická ramena na PR2 se skládají z paže se čtyřmi stupni volnosti, zápěstí se třemi stupni volnosti a chapadla s jedním stupněm volnosti. Rameno obsahuje systém protivah, díky čemuž je možné využít motorů s menším příkonem. Navíc pokud odpojíme napájení z motorů nezatíženého ramene, to zůstane na místě a nespadne. Na "zápěstí"ramene se nachází barevná kamera, která nám umožňuje pohled na práci chapadla. Chapadlo dále obsahuje tří-osý akcelerometr a na jeho dva prsty je možné připojit tlakové senzory. Poslední části manipulátoru, jeho zápěstí a chapadlo, jsou připojeny pomocí kloubu, který umožňuje jejich nekonečné otáčení. To zvyšuje možnosti práce s těmito manipulátory, neboť je možné otáčením například zašroubovat šroub nebo skládat Rubikovu kostku.

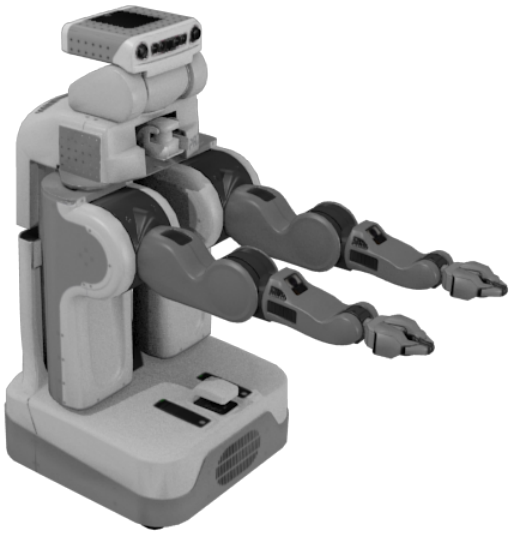
Na vrcholu celého robota je umístěna hlava. S tou je možné otáčet v rozsahu až 350° a naklánět dopředu a dozadu v rozsahu až 115°. V hlavě jsou zabudované 2 páry stereokamer s úzkým a širokým záběrem, dále pěti megapixelová barevná kamera a projektor textury. K hlavě je navíc připojený kinect [7].

5.1 Ovládání pohybu robota PR2

Řízení pohybu nejen platformy PR2, ale i dalších robotů používajících ROS, je řešeno zapisováním příkazů na topic */cmd_vel*. Na tomto topicu zveřejňovaná data mají formát *Twist*. Jedná se o strukturu složenou ze dvou tříprvkových vektorů s názvy *linear* a *angular*. Prvky těchto vektorů se nazývají x, y, z.

První z těchto vektorů - linear - obsahuje informace o rychlosti a směru přímé jízdy. Například při zadané hodnotě $y = 2$, pojede robot do boku rychlostí $2m/s$. Je zřejmé, že robot PR2 nebude využívat prvek vektoru z.

Vektor *angular* nese informace o rychlosti a směru otáčení, údaje se poskytují v radiánech za sekundu. Robot PR2 se dokáže otáčet pouze po vodorovné rovině z. Pokud nese parametr z hodnotu $\pi/2$, robot se během jedné vteřiny otočí o 180°.



Obrázek 5.1: Robot PR2 [a].

[a] Zdroj: http://www.openrobots.org/morse/doc/1.2/_images/pr2.png

5.2 Kinect

Kinect je zařízení vytvořené firmou Microsoft primárně určené k ovládání herní konzole Xbox 360 pomocí pohybu těla. Od jeho uvedení na trh v roce 2010 si toto zařízení nachází cestu i do jiných odvětví informatiky jako například virtuální reality a robotiky. Vzhledem k tomu, že chybí oficiální podpora jiných operačních systémů než MS Windows, začaly vznikat knihovny vyvíjené komunitou, které umožňují použití Kinectu i v jiných operačních systémech. Příkladem takových knihoven můžou být libfreenect a OpenNI.

Zařízení obsahuje sadu několika kamer umožňující prostorové vidění, sledování pohybů a jejich 3D zaznamenávání, dokáže také rozpoznávat obličeje a gesta.

Dále Kinect obsahuje pole čtyř mikrofonů, které je kromě klasického nahrávání zvuku schopné například lokalizace mluvčího v prostoru a filtrování šumu. Tyto mikrofony zaznamenávají 16 bitové vzorky zvuku s vzorkovací frekvencí 16kHz. Tři z těchto mikrofonů jsou umístěny na pravé straně pod nápisem XBOX360, čtvrtý je na levém kraji.

5.2.1 HARK-Kinect

Software HARK-Kinect slouží jako ovladač pro Kinect na Linuxová zařízení. Díky němu se po připojení Kinect objeví jako obyčejný mikrofon zapojený přes USB a je tak možné jej ovládat jako ALSA zařízení pomocí systémových funkcí a přistupovat k němu jako k obyčejnému mikrofonu.

Obrázek 5.2: Zařízení Kinect[a].

[a]Zdroj: http://upload.wikimedia.org/wikipedia/commons/6/67/Xbox-360-Kinect-Standalone.png

Kapitola 6

Návrh aplikace pro rozpoznávání řeči v ROSu

Cílem práce je vytvořit komplexní program, který za pomocí externích nástrojů dokáže převést mluvené slovo na text, který následně zpracuje. Aplikace se tedy zabývá získáním zvukových dat, jejich předáním rozpoznávacímu systému a následným zpracováním rozpoznaných dat. Způsob zpracování těchto dat bude záležet na uživateli. Po dohodě s vedoucím práce budou v programu zahrnuty možnosti pro zapisování rozpoznaného textu na definovaný ROS topic a pro jeho zpracování.

Zpracováním je myšlena analýza textu a následné vykonání odpovídající činnosti. Tato možnost slouží pouze pro vykonávání jednoduchých příkazů, například otočení se o devadesát stupňů doleva či doprava a jízda rychlostí 1m/s dopředu a dozadu. Pro obsluhu složitějších příkazů doporučuji vytvořit uzel, který bude číst rozpoznaný text z ROS topicu a následně vykoná požadovanou reakci. Tyto příkazy budou definovány gramatikou, která bude součástí aplikace a nastavená jako výchozí gramatika.

Výsledný systém se bude skládat z několika částí:

- **Modul ovládání knihovny Sphinx-4**, který se bude starat o veškeré činnosti spojené s rozpoznáváním pomocí knihovny Sphinx-4. Tento modul bude také sloužit k nastavování parametrů knihovny, jako například cesta k souboru s gramatikou.
- **Modul ovládání serverů GoogleSpeech**, sloužící ke komunikaci se servery GoogleSpeech. Tento modul bude také využívat knihovnu Sphinx-4 z důvodu extrakce řečového signálu ze vstupu mikrofonu. Jednomu z těchto dvou modulů bude předáno řízení po startu aplikace.
- **Část pro interakci v rámci ROSu**, která má na starost publikování výsledků ve formě textu, případně zpracování výsledku.

Rozpoznávaná data

Co se týče druhu očekávaných vstupních dat, uživatel si bude moci vybrat mezi jazykovým modelem, který rozpoznává smysluplné, logicky řazené větné konstrukce, a JSGF gramatikou, kdy rozpoznávač očekává fráze patřící do této gramatiky. Ukázková gramatika bude součástí aplikace, uživatel si může vytvořit jakékoliv další. Je třeba počítat s omezením, že jiné gramatiky nebudou pravděpodobně fungovat s částí ROS modulu zodpovědnou za analýzu při vykonávání příkazu.

6.1 Popis plánované činnosti aplikace

Příklad plánované činnosti aplikace je ilustrován na modelových situacích popsaných v následujících kapitolách.

6.1.1 Rozpoznávání s pomocí knihovny Sphinx-4

Během tohoto druhu rozpoznávání jsou všechny činnosti řízeny knihovnou Sphinx. Jediná potřebná činnost je zapnutí mikrofonu na začátku a spuštění samotného rozpoznávání. Vzhledem k tomu, že proces rozpoznávání běží během nahrávání mikrofonem a nově příchozí data jsou knihovnou ihned analyzována, není potřeba vytvářet žádná nová vlákna. Po rozpoznání příkazu je aplikaci vrácen výsledek, který je následně předán do hlavního balíčku k zapsaní na ROS topic, a následně je opět zavolána funkce pro začátek rozpoznávání. Pokud uživatel během zápisu na ROS topic začne říkat nový povel, tato data nejsou ztracena - knihovna Sphinx si je ukládá do vnitřních bufferů. Tato data jsou ale analyzována až po zavolání funkce pro započetí rozpoznávání.

Zpoždění tohoto rozpoznávání (doba od konce řeči po vrácení výsledku) lze očekávat v intervalu od půl do jedné vteřiny. Toto zpoždění je dáno blokem front-endu knihovny, který je zodpovědný za extrakci řečového signálu. Zjednodušeně řečeno, pokud po dobu půl vteřiny po ukončení řeči nepřijde další řečový signál, předpokládá se, že uživatel domluvil. Tuto dobu lze v konfiguračním souboru knihovny měnit, nicméně její výchozí hodnota je vyhovující.

6.1.2 Rozpoznávání s pomocí serverů GoogleSpeech

Při tomto online rozpoznávání je třeba využít více komponent. První z nich slouží jako voice-activity detector. Ten bude extrahovat řečový signál, který předá další komponentě k odeslání. Následně počká na odpověď, kterou zapíše na topic. Jakmile dojde k zachycení signálu, vytvoří se nové vlákno, které má na starost právě komunikaci se servery Google. To proto, aby systém nemusel čekat na příjem odpovědi, což by v některých případech mohlo trvat nepřípustně dlouho (například i přes půl minuty, skutečný čas záleží na nastavení operačním systému, příp. běhového prostředí).

Oproti předchozí modelové situaci je v tomto problému potřeba počítat s větším zpožděním systému. K půl vteřině voice-activity detectoru se přidává doba nutná k poslání dat na server, zpracování dat servery Google a následné stáhnutí odpovědi.

6.2 Ovládání aplikace

Aplikace bude běžet jako ROS balíček, spouštět se bude utilitou *roslaunch*. Tato utilita očekává dva parametry - jméno balíčku a jméno launch souboru. Launch soubory se nachází v podadresáři *launch* kořenového adresáře specifikovaného balíčku a mají syntaxi XML. Předávání případných parametrů se bude dít právě pomocí těchto souborů.

Vzhledem k výsledkům testů (viz 8.3) byla do programu přidána možnost zastavení zpracovávání dat. To znamená, že se rozpoznaná data nebudou zapisovat na topic, případně text nebude vůbec předán k analýze pro vykonání příkazu, ale pouze se vypíší na standardní výstup. Tuto funkci doporučuji použít hned po spuštění aplikace za účelem kalibrace knihovny Sphinx-4. Tato vlastnost se aktivuje během rozpoznávání zmáčknutím klávesy *p*, deaktivuje se stejným tlačítkem.

Aplikace se ukončuje stisknutím kombinace kláves ctrl+c.

Všechny parametry aplikace, jako například klíč serverů GoogleSpeech, cesta k vlastní gramatice atd., se budou nastavovat editací launch souborů. Pouze parametry knihovny Sphinx-4, u kterých lze očekávat, že se nebudou měnit často, bude potřeba nastavit přímo v konfiguračním souboru aplikace. Příkladem těchto parametrů může být akustický a jazykový model a slovník.

Kapitola 7

Implementace

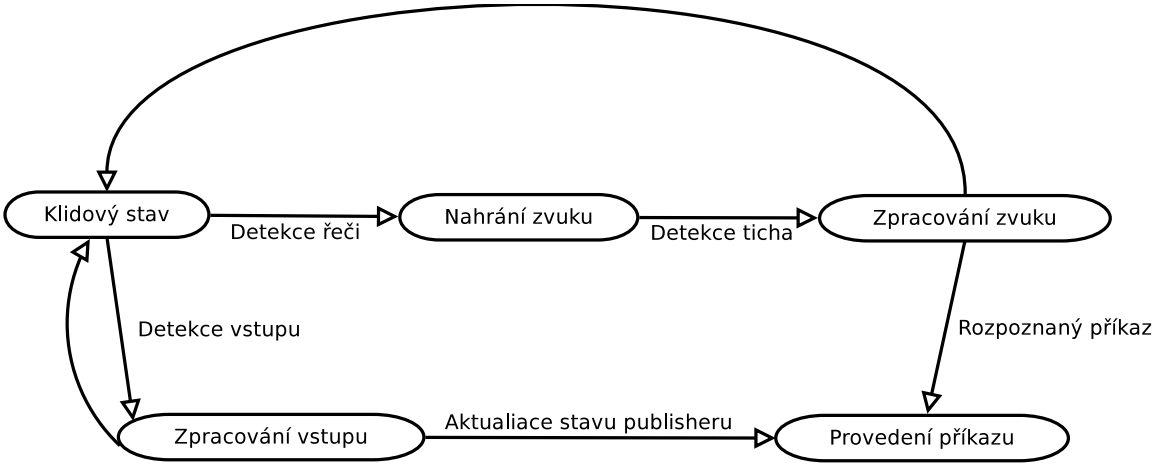
V této kapitole je popsána celková architektura aplikace a jsou zde popsány nejdůležitější části programu.

Během implementace a testování se ukázalo, že ne každé zařízení Kinect, byť stejného typu, dokáže nahrávat zvuk. Práce byla testována na Kinectu s označením "DEMO"s číslem "SAP:001000206497-0000", který to umožňuje.

7.1 Koncept systému

Vzhledem k tomu, že jako implementační jazyk byla vybrána Java, byl použit objektový návrh. Všechny třídy budou rozděleny do několika logických celků - balíčků. Hlavní balíček se bude starat o interakci se systémem ROS - vytvoření uzlu a zapisování dat na požadovaný ROS topic. Druhý balíček bude obsahovat implementaci objektů obsluhujících proces převodu řeči na text. Poslední balíček je pomocný, bude obsahovat objekty potřebné pro rozpoznávání pomocí serverů GoogleSpeech.

Celá aplikace poběží jako ROS uzel a ke svému běhu bude využívat knihovny balíčku *rosjava*. Dále bude zapojena knihovna Sphinx-4 a to pro rozpoznávání řeči. Právě využití knihovny Sphinx-4, která je kompletně napsaná v Javě, je důvod, proč byla jako implementační jazyk zvolena Java.



Obrázek 7.1: Zobrazení principu činnosti aplikace.

7.2 Konfigurace knihovny Sphinx-4

Jak bylo zmíněno výše (3.2.4), činnost knihovny Sphinx-4 se specifikuje pomocí konfiguračního souboru. Tyto soubory jsou rozsáhlé a pro běžného uživatele zbytečně složité, proto jsem přistoupil k možnosti nastavení nejčastějších parametrů v launch souborech. Aplikace tyto údaje předá knihovně.

V rámci této práce byly vytvořeny dva konfigurační soubory.

full_recognition.xml

První z těchto souborů je použit při rozpoznávání pouze s pomocí knihovny Sphinx. Vzhledem k faktu, že tento soubor popisuje každou komponentu (včetně jejích parametrů) použitou během rozpoznávání, je velmi složitý. Jako základ byl použit vzorový výchozí konfigurační soubor knihovny dostupný online[1].

Ze vzorového souboru byly odstraněny nepoužívané části a část front-endu *Microphone* byla nahrazena blokem *Kinect*, popsaným níže. Důležitou částí tohoto souboru je část popisující lingvistickou část knihovny *flatLinguist*, v jehož podelementu s atributem *grammar* se nastavuje typ gramatiky (resp. ID elementu popisujícího požadovaný typ gramatiky).

vad.xml

Druhý z těchto souborů nastavuje knihovnu Sphinx-4 k tomu, aby sloužila jako voice-activity detector. Z toho důvodu obsahuje pouze komponenty front-end. V tomto případě navíc obsahuje pouze bloky pro zachycení zvuku (entita microphone), jeho rozčlenění na stejně dlouhé úseky (entita dataBlocker) a následné označení řečového signálu (entity speechClassifier a speechMarker). Označení řečového signálu se provádí přidáním signálu *SpeechStartSignal* pro začátek řeči a *SpeechEndSignal* pro její konec.

Označením zvukového signálu těmito signály je myšleno to, že zvuková data jsou uložena do instance právě tříd *SpeechStartSignal*, resp. *SpeechEndSignal*.

7.3 Hierarchie souborů

Všechny soubory jsou součástí balíčku *edu.vutbr.xhajek*31, případně jeho podbalíčků. Celková hierarchie je vidět v diagramu tříd na obrázku 7.2.

RecognizerNode.java

Soubor obsahuje implementaci zpracování parametrů a spuštění samotného hlavního uzlu. Následně předá řízení jedné ze tříd z podbalíčku *loop*.

ROS.java

Třída, která se stará o zápis na ROS topic. V konstruktoru přijímá parametr *execute* určující, jestli se má rozpoznaný příkaz vykonat. V závislosti na tomto parametru vytvoří vhodný publisher (textových příkazů, nebo zpráv o poloze). V případě, že hodnota parametru *execute* byla *true*, třída ROS pomocí jednoduché sekvence if-else if rozpoznaný

[1] https://github.com/cmusphinx/sphinx4/blob/master/sphinx4-core/src/main/resources/edu/cmu/sphinx/api/default.config.xml

text analyzuje a v případě nalezení vhodné reakce vykoná požadovaný příkaz, v opačném případě vypíše přijatá data na topic.

Pokud byla zvolena možnost analýzy příkazu a jeho vykonání, program očekává příkazy korespondující s gramatikou na obrázku 7.5.

Následující dva soubory patří do balíčku *loop*. Obě tyto třídy dědí z abstraktní třídy *CancellableLoop*, která je součástí balíčku rosjava a slouží pro implementaci nekonečných smyček, které jsou v rámci ROSu snadno přerušitelné. Pro tento účel obsahují metodu *setup*, která se volá na počátku před první iterací a metodu *loop*, která obsahuje implementaci cyklicky vykonávané činnosti. Tato metoda je volána stále dokola knihovnou *rosjava*, dokud nedojde k přerušení zavoláním funkce *cancel*.

Těmto třídám, které slouží k samotnému řízení rozpoznávání, je předáno řízení programu.

GrammarLoop.java

Tato třída komunikuje s knihovnou Sphinx-4. Používá se při offline rozpoznávání na základě uživatelem definované gramatiky a jazykového modelu.

GSpeechLoop.java

Tato třída je použita při rozpoznávání pomocí serverů GoogleSpeech. Využívá pro svůj chod balíček *gspeech*. Nejdříve si z voice activity detectoru načte pole bajtů obsahující řečová data a to následně předá k odeslání na servery Google. Tím jeho úloha končí a začne nová iterace čekáním na data z VAD.

Následující dvě třídy z balíčku *gspeech* jsou využity při rozpoznávání se servery GoogleSpeech.

GRecognizer.java

Této třídě jsou předána zvuková data obsahující pouze řeč ve formě pole bajtů. Následně vytvoří nové vlákno, které nahraje zvuk na servery a čeká na odpověď. Zároveň je novému vláknu předána předem vytvořená instance třídy ROS, které jsou předána příchozí rozpoznaná data. Nové vlákno je vytvořeno z toho důvodu, aby se mohlo vrátit řízení třídě *GSpeechLoop*, v opačném případě by mohlo docházet ke zbytečnému blokování činnosti aplikace, například v případě chyby během přenosu.

Kinect.java

Kvůli nekompatibilitě knihovny Sphinx-4 a zařízení Kinect bylo potřeba vytvořit třídu, která bude knihovně poskytovat zvuková data. Tato třída je napsaná jako blok front-endu knihovny, dědí tedy ze třídy $edu.cmu.sphinx.frontend.BaseDataProcessor$.

VAD.java

Obsahuje implementaci voice-activity detectoru. Pro tento účel využívá služeb knihovny Sphinx-4.

Metoda *nextSpeech* této třídy čte příchozí data z front-endu knihovny, které jsou již označená signály *SpeechStartSignal* a *SpeechEndSignal*. Po tom, co dostane data označená jako začátek řeči, začne všechna následující příchozí data zapisovat do pole bajtů až po příchod signálu *SpeechEndSignal*. Front-end knihovny ale vrací zvuková data v bytovém uspořádání big-endian, proto je ve třídě implementovaná funkce *toLittleEndian*, která pomocí cyklu *for* pole přeuspořádá na little-endian, tedy formát přijímaný servery Google-Speech. Tato metoda je zavolána pro nashromážděná data po zaznamenání konce řečového signálu, následně metoda *nextSpeech* vrátí data, již ve správném pořadí bajtů.

7.4 Spouštěcí soubory

Hlavní spouštěcí soubor aplikace je zobrazen na obrázku 7.3. Hned z jeho začátku je patrné, že jméno topicu, na který se případně budou zapisovat rozpoznaná data je implicitně nastaveno na *voice_commands*. V následující části je patrné nastavení několika parametrů, kde jediný nastavený na *true* je parametr *grammar*. To znamená, že se ve výchozím nastavení používá uživatelem definovaná gramatika. Nastavení parametru *execute* na *false* znamená, že se příkazy nebudou zpracovávat a vykonávat, ale text příkazu se zveřejní na daném topicu. Další části launch souboru slouží ke zpracování těchto parametrů a jejich následnému předání spouštěného programu.

V adresáři *launch* balíčku této aplikace se nachází i další soubory. Tyto jednotlivé soubory vypadají velmi podobně. Nejdůležitější část je zobrazena na obrázku 7.4. První řádek říká, že se má použít soubor *voice_control.launch*, následujících jsou pak věnovány nastavení argumentů programu. Ukázka na obrázku 7.4 (soubor *voice_control_execute.launch*) tedy způsobí, že se opět použije uživatelem definovaná gramatika, ale dojde k vykonání příkazů.

7.5 Vlastní gramatika formátu JSGF

Pro ovládání pohybů robota doporučuji využít vlastní gramatiku ve formátu JSGF. Tento formát je velmi jednoduchý a umožňuje vytvořit komplexní gramatiky popisující očekávané fráze.

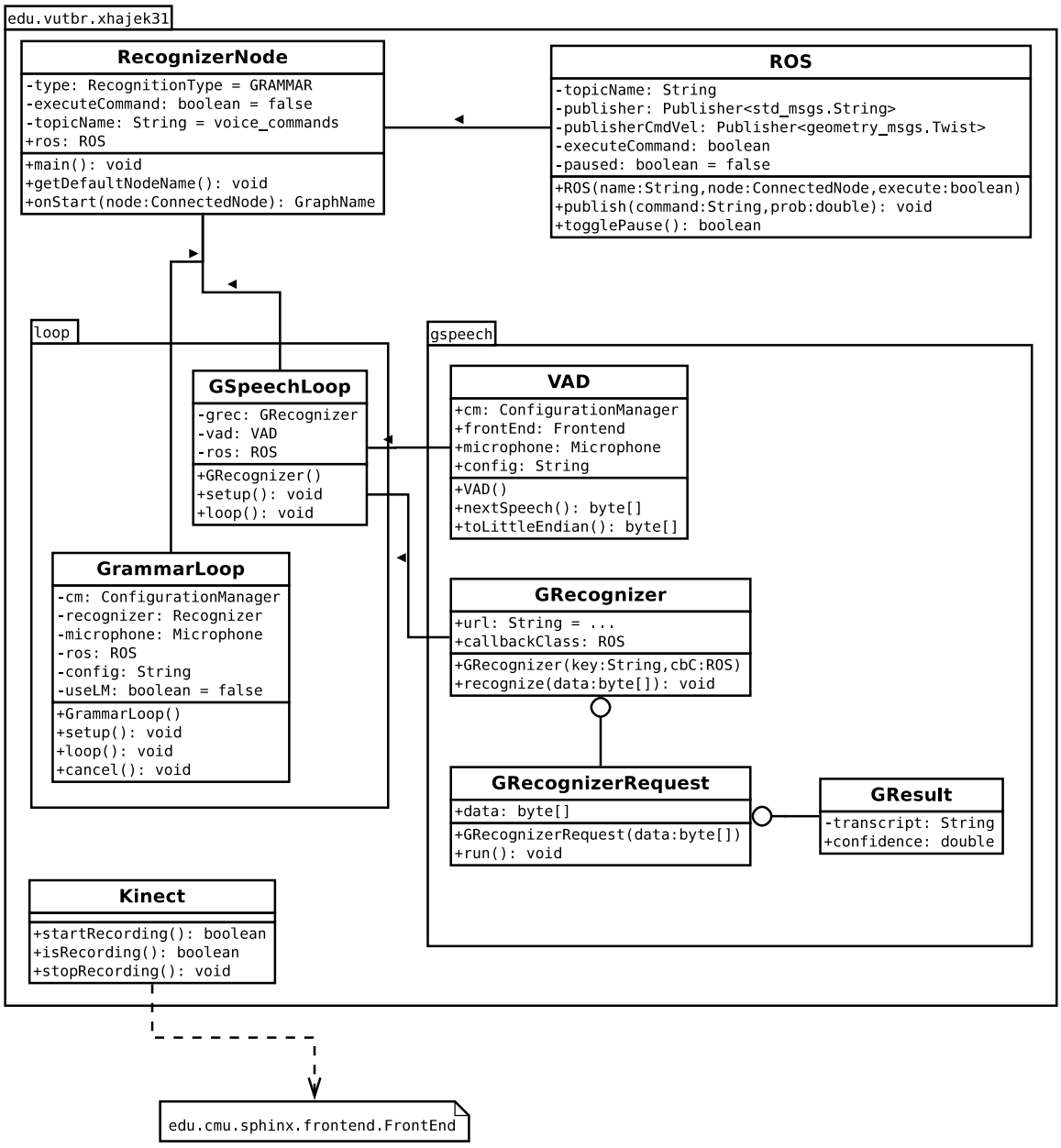
Z obrázku 7.5 je patrné, že každý příkaz musí začínat oslovením robota "pee are two", tedy přepisem jména PR2. Po jménu následuje jeden ze sady příkazů určující činnost. Ve finále můžou být příkazy následující:

"PR2 go forward"
"PR2 go back"
"PR2 rotate right"
"PR2 stop"

7.6 Ukázkový python skript

Pro účely ilustrace byl vytvořen v novém catkin balíčku *voice_control_examples* jednoduchý skript v jazyce *python*. Tento skript se po spuštění připojí na topic, na který jsou zapisována rozpoznaná data, a následně pomocí jednoduché sekvence if-elseif rozpoznaný příkaz zpracuje a vykoná odpovídající činnost. V podstatě dělá tento skript to samé, co třída ROS s parametrem *execute* nastaveným na *true*(viz 7.3), pouze jinak a snáze rozšiřitelněji.

Skript se opět spouští pomocí vytvořeného launch souboru, který nejprve spustí hlavní rozpoznávací aplikaci v režimu rozpoznávání pomocí knihovny Sphinx-4 s uživatelskou JSGF gramatikou a zapisování rozpoznaných dat na topic a následně spustí samotný příkladový python skript.



Obrázek 7.2: Diagram tříd a balíčků.

```
<launch>
      <arg name="voice_commands_topic" default="voice_commands"/>

      <arg name="execute" default="false"/>
      <arg name="google" default="false"/>
      <arg name="grammar" default="true"/>
      <arg name="lm" default="false"/>

      <arg unless="$(arg execute)" name="exec" value=""/>
      <arg if="$(arg execute)" name="exec" value="--execute"/>

      <arg if="$(arg google)" name="type" value="--google"/>
      <arg if="$(arg grammar)" name="type" value="--grammar"/>
      <arg if="$(arg lm)" name="type" value="--lm"/>

      <node type="voice_control" name="voice_control"
           pkg="voice_control" output="screen"
           args="$(arg type) -n $(arg voice_commands_topic) $(arg exec)" />
</launch>
```

Obrázek 7.3: Základní launch soubor aplikace.

```
<include file="$(find voice_control)/launch/voice_control.launch">
       <arg name="execute" default="true"/>
       <arg name="google" default="false"/>
       <arg name="grammar" default="true"/>
       <arg name="lm" default="false"/>
</include>
```

Obrázek 7.4: Část launch souboru.

```
public <robot_gram> = <name> <command>;
<name> = pee are two;
<command> = <go> | <turn> | stop;

<go> = go <direction>;
<turn> = rotate <dir_side>;

<direction> = front | back | forward | backward;
<dir_side> = left | right;
```

Obrázek 7.5: Vytvořená gramatika.

Kapitola 8

Testování a experimenty

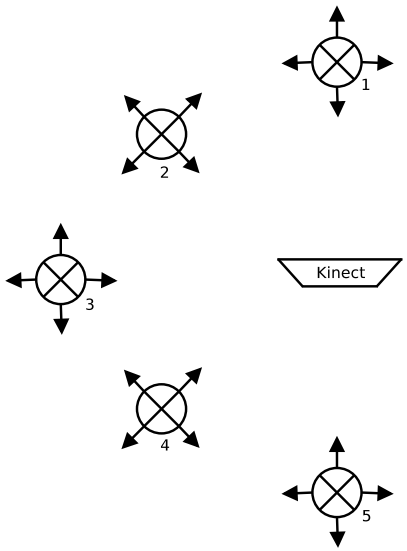
V následující kapitole jsou popsány jednotlivé testy, kterým byla vytvořená aplikace podrobena. Tyto testy mají za úkol srovnat rychlost a úspěšnost různých rozpoznávání v závislosti na použité metodě rozpoznávání a postavení mluvčího v prostoru.

Testy probíhaly v anglickém jazyce a účastnilo se jich devatenáct uživatelů (jedenáct mužů a osm žen) ve věku dvacet až třicet let, všichni s úrovní angličtiny alespoň B1, většina (čtrnáct) s úrovní B2.

8.1 Experimenty s polohou mluvčího

Tyto experimenty mají za úkol zjistit vliv polohy mluvčího vůči zařízení Kinect. Jejich cílem je stanovit, při kterých umístěních mluvčího ještě lze rozpoznávání označit za spolehlivé.

Během testu bylo stanoveno pět umístění mluvčího v prostoru okolo Kinectu. Dále byly vybrány pro každé umístění mluvčího čtyři jeho polohy vůči zařízení, tedy otočení čelem, zády a boky. Všechny polohy jsou ilustrovány na obrázku 8.1. Vzdálenost mluvčího od zařízení se pokaždé pohybuje okolo třech metrů a mluvčí vždy mluví jasně, nahlas a zřetelně.



Obrázek 8.1: Testované polohy mluvčího vůči Kinectu (pohled shora).

Jako rozpoznávací metoda bylo vybráno rozpoznávání se servery GoogleSpeech, neboť při dostatečném opakování si dokáže knihovna Sphinx-4 zvyknout na uživatele, čímž by mohlo docházet ke zkreslování výsledků. Testovací sada příkazů obsahuje výrazy popsané v gramatice z kapitoly 7.5. Jedinou změnou je vynechávání sousloví "PR2"na začátku.

umístění v prostoru / poloha mluvčího	1	2	3	4	5
čelem	0.86	0.85	0.79	0.52	0.47
zády	0.73	0.68	0.61	0.4	0.36
levým bokem	0.8	0.7	0.51	0.47	0.44
pravým bokem	0.81	0.63	0.5	0.46	0.4

Tabulka 8.1: Průměrné pravděpodobnosti úspěšného rozpoznání v závislosti na poloze mluvčího vůči kinectu.

Prezentované výsledky v tabulce 8.1 jsou aritmetickými průměry pravděpodobností úspěšného rozpoznání jednotlivých pokusů, vrácenými servery GoogleSpeech. Jinými slovy se jedná o hodnotu, jak moc si je rozpoznávač jistý, že je výsledek správný. Tabulka ukazuje, že dle očekávání jsou výsledky nejlepší, pokud stojí mluvčí před robotem čelem k němu. Dále je patrné, že uspokojivých výsledků je dosaženo i v ostatních umístěních mluvčího okolo robota, pokud je k němu čelem, stejně jako v případě, že mluvčí stojí před robotem (polohy 1, 2 a 3) bokem k němu. Pokud stojí mluvčí před robotem (polohy 1, 2, a 3) zády k němu nebo za robotem (polohy 4 a 5) bokem, stále se dají výsledky označit za přijatelné, nicméně občas už dochází k přeslechům rozpoznávacího systému.

Naopak při umístění mluvčího za robotem (4 a 5) zády k němu výsledky jasně dokazují, že je systém velmi náchylný na přeslechy a vrací také nízké hodnoty pravděpodobnosti úspěšného rozpoznání.

8.2 Základní srovnání metod rozpoznávání

Cílem následujících experimentů je popsat rozdíly rozpoznávacích metod, co se týče rychlosti a spolehlivosti. Během experimentů stojí mluvčí čelem k zařízení Kinect ve vzdálenosti dvou metrů.

Testovací sada se skládala z pěti příkazů patřících do gramatiky popsané v kapitole 7.5. Jelikož se dá očekávat, že sousloví "pee are two"bude v jazykovém modelu ohodnoceno pravděpodobností blížící se nule, bylo toto sousloví vyměněno za "robot".

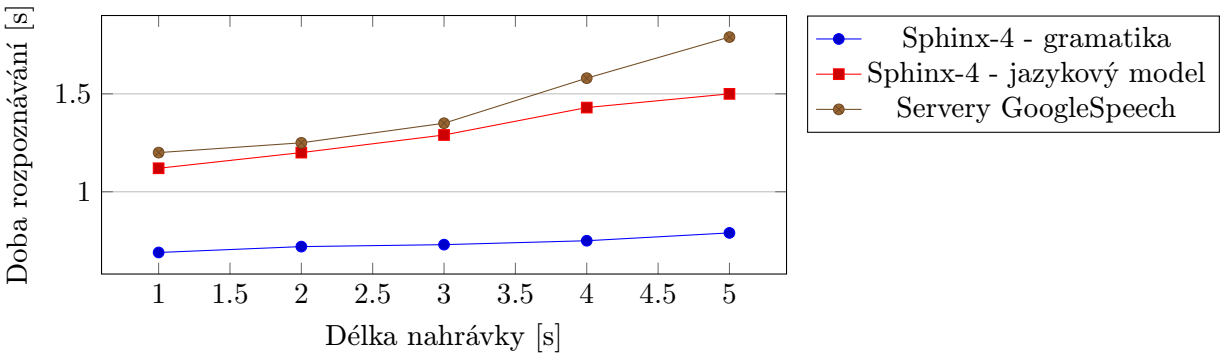
Na počátku testu byla vytvořena sada nahrávek, na kterých budou v následujících kapitolách testovány různé vlastnosti rozpoznávačů.

8.2.1 Srovnání rychlosti

V tomto případě je měřena doba trvání rozpoznávání od zjištění konce řeči po obdržení rozpoznaného textu. Experimenty jsou prováděny na spolehlivém internetovém připojení[1].

Z grafu 8.2 je patrné, že při rozpoznávání pomocí knihovny Sphinx-4 se hodnoty trvání pohybují stále na podobné úrovni, což je dáno průběžnou analýzou řečového signálu a při

[1]Rychlost připojení: download - 93.66Mbps, upload - 94.08Mbps. Testováno pomocí serveru http://www.speedtest.net/



Tabulka 8.2: Graf zobrazující rychlost rozpoznávání.

jeho nahrávání. Naopak serverům GoogleSpeech rozpoznání trvá déle s delšími nahrávkami. To je způsobeno potřebou nejprve nahrát zvukový příkaz a pak jej jako celek odeslat k analýze.

8.2.2 Srovnání v úspěšnosti

Tento experiment má za cíl zjistit, která metoda rozpoznávání je nejlepší z hlediska úspěšnosti. Experimenty jsou prováděny na robotu PR2.

rozpoznavač / testovací fráze	Sphinx-4 - gramatika	Sphinx-4 - jazykový model	Servery GoogleSpeech
”robot go forward”	**0.91**	0.44	0.88
”robot go backward”	**0.85**	0.37	**0.85**
”robot move forward”	**0.87**	0.53	0.85
”robot move backward”	**0.89**	0.32	0.86
”robot go front”	0.76	0.26	**0.83**
”robot go back”	**0.81**	0.44	**0.81**
”robot turn left”	**0.86**	0.37	0.84
”robot turn right”	0.70	0.35	**0.76**
”robot rotate left”	**0.98**	0.58	0.87
”robot rotate right”	**0.87**	0.35	0.85
”robot stop”	**0.93**	0.51	0.4

Tabulka 8.3: Úspěšnost správného rozpoznání textu rozpoznávači.

Tabulka 8.3 ukazuje, že nejvyšší úspěšnosti dosahuje metoda s vlastní JSGF gramatikou. To je dáno tím, že knihovna Sphinx se rozhoduje mezi mnohonásobně menším počtem možností, dá se tedy říci, že to má mnohem lehčí. Tato metoda je následována servery GoogleSpeech, které také dosahují velmi kvalitních výsledků. Jako nejhorší z tohoto testu vyšla metoda založená na jazykovém modelu.

Výsledky tohoto experimentu ukazují, že pro běžné ovládání robota předem stanovenými příkazy je nejlepší použít knihovnu Sphinx-4 s vlastní JSGF gramatikou. Naopak při

potřebě širší slovní zásoby, například pro vedení dialogu s robotem, je lepší použít servery GoogleSpeech. Pokud však nebude mít robot přístup k internetu, zbývá použít knihovnu Sphinx-4 s jazykovým modelem.

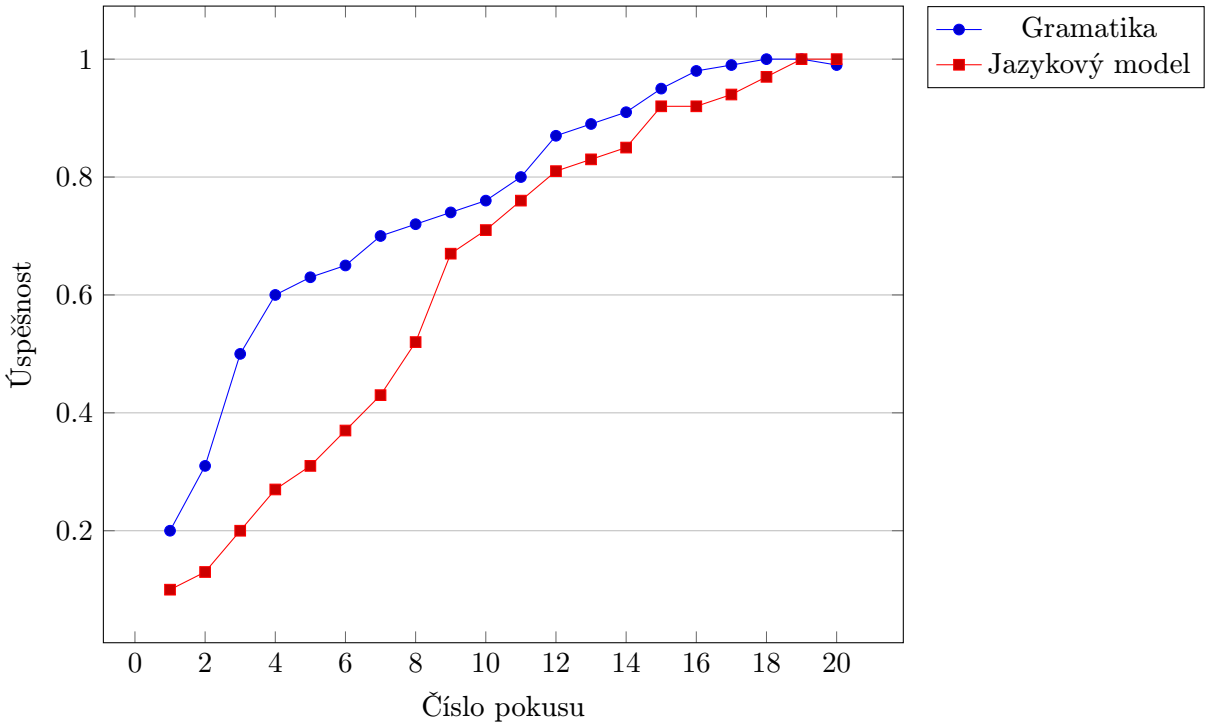
Dále výsledky ukazují, že co se týče příkazu pro otáčení, má mnohem lepší výsledky sloveso *rotate* než *turn*. Podobný rozdíl je vidět i na dalších dvojicích, například v případě směru vpřed *forward* a *front*. Tento rozdíl je dán složitostí a variabilitou znělých hlásek těchto slov. Proto mají fráze se slovy jako *right*, *turn* nebo *front* v případě rozpoznávání pomocí knihovny Sphinx-4 menší úspěšnost než ostatní.

8.3 Experimenty s knihovnou Sphinx-4

V této kapitole jsou popsány různé experimenty s metodami rozpoznávání využívající knihovnu Sphinx-4, jejichž cílem je popsat různé způsoby pro zvýšení efektivity rozpoznávání.

8.3.1 Srovnání úspěšnosti metod

Během těchto experimentů se mluvčí nachází stejně jako při předchozím experimentu dva metry od Kinectu v poloze čelem k zařízení.



Tabulka 8.4: Graf s výsledky jednotlivých pokusů, který ilustruje adaptaci knihovny Sphinx-4 na mluvčího.

Graf 8.4 ukazuje, že počáteční pokusy mají nižší úspěšnost, která se ale s přibývajícím počtem povelů zvyšuje, až se dostane k hranici stoprocentní úspěšnosti. Z toho je patrné, že by bylo vhodné přidat do programu možnost jakési kalibrace, které by bylo doporučeno využít po startu aplikace. Tato možnost bude nepovinná a bude ji možné spustit kdykoliv za běhu aplikace.

Kalibrací je myšleno zastavení zpracovávání dat třídou ROS. Se zapnutou kalibrací jsou data zapisována na standardní výstup a uživatel může kontrolovat, zdali je rozpoznávání dostatečně přesné.

8.3.2 JSGF gramatika a špatná data

Další část testů se věnuje testování úspěšnosti rozpoznávání pomocí JSGF gramatiky. Během tohoto testu knihovna očekává vstup splňující pravidla vytvořené gramatiky popsané v kapitole 7.5, nicméně jsou jí předkládána data jiná, často ale velmi podobná správným frázím. V ideálním případě by tyto fráze neměly být vůbec rozpoznány a knihovnou ignorovány[2].

Testovací nahrávky byly pořízeny dvakrát, pro každou frázi v tabulce tedy vzniklo třicet osm nahrávek.

testovaná fráze nepatřící do gramatiky	nejčastěji přiřazený text (počet)	počet odmítnutých
PR2 turn back	PR2 turn back (19)	17
PR2 go right	PR2 go front (23)	15
PR2 backward backward	PR2 go backward (9)	25
turn left	PR2 stop (8)	16
go forward PR2	PR2 go front (9)	17
go somewhere	PR2 go front (4)	33
turn up	PR2 stop (12)	21
bring me a beer	PR2 go back (6)	16
shut down	-	38
je m'appelle Pavel	PR2 go back (17)	13
máma mele maso	PR2 rotate left (14)	19
- odkašlání si -	PR2 stop (11)	23
- bouchnutí dveří -	-	38
- tleskání jedné osoby -	PR2 stop (1)	37

Tabulka 8.5: Tabulka s výsledky experimentu rozpoznání frází "podobných"definovaným příkazům.

Tabulka 8.5 nicméně dokazuje, že k požadovanému odmítnutí došlo pouze v 62% případů. Toto číslo není moc přesvědčivé, zvláště při ovládání pohybu robota, kdy každý chybně rozpoznaný příkaz může způsobit nehodu. Na druhou stranu je třeba si uvědomit fakt, že aby k těmto případům docházelo, musí být v prostoru s robotem alespoň dva komunikující lidé (případně člověk trpící samomluvou), což se nedá očekávat vždy. Pokud by ale tyto případy nastaly a osoby by potřebovaly komunikovat, jako nejlepší se jeví využití možnosti pozastavení zpracovávání dat, přidané do programu na základě předchozích experimentů. Další možností je využití systému push-to-talk a pomocí bezdrátového ovladače umožňovat zpracování dat (resp. aktivovat mikrofon) na dálku. To už ale přesahuje rámec této práce.

Poslední dvě sekce tabulky mapují reakce robota na úplně neznámé zvuky (cizí jazyky a hluk), kdy už míra úspěšnosti záleží zejména na akustickém modelu tím, jaké fonémy

[2]knihovna Sphinx-4 ve skutečnosti reaguje vracením prázdného řetězce

těmto zvukům přiřadí. Pokud se bude sekvence těchto přiřazených fonémů dostatečně blížit nějaké frázi v gramatice, dojde ke špatnému přiřazení a nechtěnému rozpoznání.

Míru odmítnutí lze nastavit v konfiguračním souboru nastavením atributu *outOfGrammarProbability*, který značí, s jakou pravděpodobností budou přicházet data nepatřící do gramatiky. Tuto hodnotu jsem po experimentování s nahrávkami nastavil na $1 * 10^{-71}$. S touto hodnotou nikdy nedocházelo k odmítnutí fráze, která do gramatiky patří, nicméně občas dochází k přeslechům.

8.4 Shrnutí poznatků

Z výsledků jednotlivých experimentů je patrné, že nejvyšší úspěšnosti dosahuje aplikace při použití metody založené na JSGF gramatice, zvláště pak po počáteční kalibraci knihovny. Tato metoda je ale kvůli omezení gramatiky, co se týče rozsahu, vhodná především pro sadu několika příkazů k ovládání robota.

Pro interakci s robotem bez předem stanovených povolených frází se nejlépe hodí rozpoznávání s pomocí serverů GoogleSpeech. Tato metoda je také velmi spolehlivá a při kvalitním internetovém připojení a krátkých frázích (cca do 3s) i srovnatelně rychlá s rozpoznáváním pomocí knihovny Sphinx-4.

Co se týče umístění mluvčího v prostoru, tak větší benevolenci dovoluje rozpoznávání se servery *GoogleSpeech*, nicméně při jakémkoliv druhu rozpoznávání je nejvhodnější stát co nejvíce před kinectem čelem k němu. Naopak při polohách za kinectem je úspěšnost rozpoznání menší než padesát procent. Tato úspěšnost se ještě razantně zmenšuje, pokud mluvčí stojí zády ke knihovně.

Kapitola 9

Závěr

Cílem této práce bylo vytvořit a implementovat komplexní systém umožňující ovládání robota hlasovými příkazy. Implementace aplikace proběhla s využitím middlewaru ROS v jazyce Java. Jako rozpoznávací nástroje byly použity knihovna Sphinx-4 a servery GoogleSpeech API. Výsledkem této práce je aplikace, která poslouchá uživatele a následně zpracovává zvuk. Jejím největším přínosem je fakt, že zaplňuje prostor mezi existujícími aplikacemi pro ROS. V současné době totiž není zveřejněná žádná další aplikace, která by sloužila ke stejnému účelu jako tato práce.

Tuto aplikaci jsem na závěr otestoval na několika uživatelích v různých podmínkách a zhodnotil závěry. V jednom případě jsem se pak přiklonil k drobné úpravě aplikace. Experimenty ukázaly, že pro ovládání robota předem definovanou množinou příkazů je nejvhodnější použít rozpoznávání s pomocí JSGF gramatiky, naopak pro obecnou interakci s robotem (například dialogový systém) je vhodné použít rozpoznávání se servery GoogleSpeech.

Co se týče budoucího rozvoje aplikace, systém by šel zdokonalit přidáním podpory pro více knihoven, například pro možnost ovládání českým jazykem. Jednou z těchto knihoven může být například BSAPI[1], vyvíjená výzkumnou skupinou Speech@FIT na Fakultě informačních technologií Vysokého učení technického v Brně. Dále je možné do aplikace přidat podporu dalších operačních systému pro práci s roboty, což by ale ve většině případů vyžadovalo kompletní přepsání aplikace.

[1] http://www.phonexia.com/docs/bsapi/index.html

Literatura

[1] WWW stránky: About ROS. online, [cit. 5.3.2015].
URL http://www.ros.org/about-ros/

[2] WWW stránky: Messages. online, [cit. 6.3.2015].
URL http://wiki.ros.org/Messages

[3] WWW stránky: Nodes. online, [cit. 5.3.2015].
URL http://wiki.ros.org/Nodes/

[4] WWW stránky: Topics. online, [cit. 5.3.2015].
URL http://wiki.ros.org/Topics

[5] Adami, A. G.: Automatic Speech Recognition: From the Beginning to the Portuguese Language. Technická zpráva, Faculdade de Informática, Pontifícia Universidade do Rio Grande do Sul, 2010.
URL https://www.inf.pucrs.br/~propor2010/proceedings/tutorials/Adami.pdf

[6] Gruhn, R.; Minker, W.; Nakamura, S.: *Statistical Pronunciation Modeling for Non-Native Speech Processing*. Springer, 2011, ISBN 978-3-642-19585-3.
URL http://www.springer.com/cda/content/document/cda_downloaddocument/9783642195853-c2.pdf?SGWID=0-0-45-1136143-p174105861

[7] Kapinus, M.: *Uživatelské rozhraní pro řízení servisního robota*. Diplomová práce, FIT VUT v Brně, 2014.
URL https://www.fit.vutbr.cz/study/DP/DP.php?id=16784&file=t

[8] Lamere, P.; Kwok, P.; Gouvêa, E.; aj.: The CMU Sphinx-4 Speech Recognition System.
URL http://www.cs.cmu.edu/~rsingh/homepage/papers/icassp03-sphinx4_2.pdf

[9] Lamere, P.; Kwok, P.; Walker, W.; aj.: Design of the CMU Sphinx-4 Decoder. Technická zpráva, Mitsubishi Electric Research Laboratories, 2003.
URL http://www.merl.com/publications/docs/TR2003-110.pdf

[10] Černocký, J.: *Zpracování řečových signálů — studijní opora*. FIT VUT v Brně, 2006.
URL http://www.fit.vutbr.cz/study/courses/ZRE/public/opora/zre_opora.pdf