FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Aplikace pro robota Pepper pro mobilního operátora
Student:	Adam Švehla
Vedoucí:	Ing. Miroslav Skrbek, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Počítačové inženýrství
Katedra:	Katedra číslicového návrhu
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Navrhněte a realizujte aplikaci pro robota Pepper, která bude určená pro podporu péče o zákazníky mobilního operátora. Aplikace musí být interaktivní a zábavnou formou informovat zákazníky. Pro verbální komunikaci integrujte do robota chatbot s kvalitním hlasovým vstupem. Naučte chatbota jak nezávaznou konverzaci, tak poskytování informací od mobilního operátora z jeho interních databází. Využijte také tablet robota a doplňte vhodné pohybové kreace a gestikulaci. Zmapujte možnosti pohybu robota se zákazníky ve formě komentované prohlídky, zhodnoťte realizovatelnost a v přiměřené míře implementujte. Konfiguraci a správu aplikace realizujte přes webové rozhraní. Vhodné technologie včetně využití internetových služeb zvolte na základě rešerše. Zaměřte se hlavně na reálnou použitelnost aplikace a výsledky práce řádně zdokumentujte. Konkrétní rozsah práce upřesněte po dohodě s vedoucím práce.

Elektronicky schválil/a prof. Ing. Hana Kubátová, CSc. dne 5. ledna 2022 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Aplikace pro robota Pepper pro mobilního operátora

Adam Švehla

Katedra číslicového návrhu
Vedoucí práce: Ing. Miroslav Skrbek, Ph.D.

9. května 2022

Poděkování

Děkuji panu Ing. Miroslavu Skrbkovi, Ph.D., za cenné rady, ochotu a vstřícnost při vedení mé bakalářské práce. Rovněž chci poděkovat společnosti T-Mobile, kolektivu Magenta Experience Center a v neposlední řadě také mé rodině a nejbližšímu okolí.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 9. května 2022 . . . . . .

Odkaz na tuto práci

Švehla, Adam. *Aplikace pro robota Pepper pro mobilního operátora.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Abstrakt

Tato bakalářská práce se zabývá návrhem a implementací aplikace pro humanoidního robota Pepper, která má za úkol podporovat péči o zákazníky na pobočce mobilního operátora. Práce analyzuje dostupné technologie pro potřeby aplikace a zároveň se snaží vyvarovat nedostatků a chyb podobných řešení. Cílem práce je vytvořit interaktivní aplikaci, která bude informovat i bavit zákazníky. Výsledkem je robustní řešení, využívající moderní cloudové technologie pro rozpoznávání řeči a umělou inteligenci pro lokalizaci a detekci záměru z přepisu hlasového vstupu, které obstojí v komunikaci, uživatelské přívětivosti a prožitku.

Klíčová slova robot Pepper, NAOqi, interaktivní aplikace, rozpoznávání řeči, lokalizace, mobilní operátor

Abstract

This bachelor thesis covers the design and implementation of an application for the Pepper humanoid robot, which is aimed at supporting customer care at a mobile phone operator's branch office. The work analyzes the available technologies for the needs of the application and at the same time tries to avoid shortcomings and errors of similar solutions. The aim of the work is to implement an interactive application that will inform and entertain customers. The result is a robust solution that uses state-of-the-art cloud technology for speech recognition and artificial intelligence for localization and intent detection, which stands out in terms of communication, user-friendliness and user experience.

Keywords Pepper robot, NAOqi, interactive application, speech recognition, localization, mobile phone operator

Obsah

Seznam obrázků

Seznam tabulek

Úvod

S humanoidními roboty se obvykle setkáváme ve vědeckých časopisech, internetových magazínech nebo například na technologických veletrzích. Na přelomu tisíciletí jsme vídali roboty, kteří sice vzdáleně připomínali člověka, ale často se teprve učili chodit, nebo se stávali spíš portýry, než společníky. Svět humanoidních robotů však kupředu posunula společnost SoftBank Robotics, když v roce 2014 uvedla na trh robota Pepper, jejich slovy „prvního humanoidního robota schopného rozpoznávat obličeje a lidské emoce“.

Jak je na tom populace 27 tisíc vyrobených humanoidů dnes, o 8 let později? Výroba byla v roce 2021 pozastavena na neurčito a zdá se, že již obnovena nebude. Robot Pepper je opředen kritikou, která upozorňuje na některé jeho nedostatky: náročný vývoj, zastaralý hardware i software, nefunkční rozpoznávání řeči, které neobstojí v rušném prostředí. Jedním z cílů této práce je ukázat, že robot Pepper si ještě nezaslouží upadnout v zapomnění a jeho reputaci a nedostatky není pozdě napravit.

Výsledkem této práce bude aplikace s uplatněním v unikátní pobočce mobilního operátora, kde Pepper přijde denně do kontaktu s desítkami zákazníků. Jedná se o jedinou pobočku svého druhu v České republice, která mimo vyřizování běžných požadavků zákazníků operátora slouží i ke konání akcí a představování moderních technologií prostřednictvím vystavených exponátů. Nová aplikace nahradí stávající omezeně interaktivní a konverzačně nevyhovující aplikaci.

Robot Pepper bude vítat zákazníky a interaktivně s nimi konverzovat. Mluvené řeči zákazníků bude rozumět lépe díky cloudovému rozpoznávání řeči a s pomocí umělé inteligence pochopí, co se mu lidé snaží sdělit. Nově se robot Pepper bude moci lokalizovat v prodejně a hovořit o vystavených technologiích. Výsledná aplikace bude přínosem jak pro operátora a jeho zaměstnance, tak pro návštěvníky jeho pobočky.

Kapitola 1

Cíl práce

Cílem práce je navrhnout a implementovat interaktivní aplikaci pro robota Pepper, která bude sloužit pro podporu péče o zákazníky na pobočce mobilního operátora.

V teoretické části je nejprve krátce představen robot Pepper a jeho hardware a software. Následně dojde na analýzu existujících aplikací, ve kterých robot Pepper komunikuje s člověkem, za účelem vyvarování se úskalí a chyb, na které zmíněné aplikace naráží. Předmětem následné rešerše jsou dostupné technologie pro potřeby aplikace, zejména cloudová řešení rozpoznávání řeči a následné zpracování přirozeného jazyka z přepisu hlasového vstupu, ale i technologie, které umožní lokalizaci a navigaci robota v prostoru.

Cílem praktické části je na základě výstupů teoretické části navrhnout a implementovat aplikaci, díky které by robot Pepper měl být schopen zábavně a interaktivně komunikovat, a to jak nezávazně, tak informativně. Do komunikace by měl být začleněn i robotův tablet a další prostředky pro neverbální komunikaci. Konkrétně by měl robot umět sdělovat informace o pořádaných událostech, které získá z interního systému operátora, spustit krátký kvíz, hovořit o náladě, počasí a podobných tématech, u kterých se předpokládá zájem zákazníka. Zároveň by měl robot dokázat provést zákazníka prostorem a orientovat se ve vystavených technologiích. Důležitou součástí aplikace je také webové administrační rozhraní pro zaměstnance pobočky, pomocí kterého lze aplikaci ovládat a konfigurovat.

Kapitola 2

Robot Pepper

Robot Pepper je humanoidní robot představený v roce 2014 společností SoftBank Robotics. Je následovníkem humanoidního robota NAO, kterého v roce 2006 uvedla na trh společnost Aldebaran Robotics (nyní součástí SoftBank Group). Hlavní využití nalézá Pepper v komerčním, domácím a akademickém prostředí. [4]

Obrázek 2.1: Robot Pepper [1]

2.1 Technické vybavení

Pepper je vysoký 1,2 metru a pohybuje se po třech všesměrových kolech. Jeho 17 kloubů a 20 stupňů volnosti mu umožňují hýbat se v kolenou, v pase, otáčet hlavou, gestikulovat rukami a přiblížit se tak řeči lidského těla. Pro verbální komunikaci je robot vybaven softwarem pro rozpoznávání a syntézu řeči. Mimo gestikulace může neverbálně komunikovat s člověkem i pomocí tabletu a LED diod umístěných v jeho očích, uších a na jeho ramenou. [4]

Důležitou součástí robota jsou senzory. Jmenovitě Pepper disponuje 3D senzorem, dvěma kamerami, čtyřmi mikrofony pro lokalizaci zvuku a dotykovými senzory na hlavě a na rukou. Kromě kamer využívá pro navigaci využívá infračervených, laserových a ultrazvukových senzorů (sonarů). Rovněž je vybaven inerciální měřící jednotkou a tzv. bumpery (dotykovými senzory v náraznících). [4]

V hlavě robota se nachází čtyřjádrový 1,91GHz procesor Intel Atom E3845 a 4 GB operační paměti DDR3. Robot se dokáže připojit k Wi-fi sítím standardu IEEE 802.11 a/b/g/n. [4]

2.2 Programové vybavení

Humanoid je řízen operačním systémem NAOqi OS, jde o distribuci založenou na Gentoo Linux, kterou pro své roboty vyvinula společnost Aldebaran Robotics. Z bezpečnostních důvodů výrobce neumožňuje používat super-user účet *root*, ale pouze účet *nao*, což je pro vývojáře významná překážka v práci. Zásadním omezením je, možnost do robota instalovat software „pouze“ pomocí cross compilingu, což je poměrně náročné [5]. Další způsob instalace softwaru, který jsem při své práci objevil, popisuji v sekci 5.1.1.

K ovládání a správě robota slouží framework NAOqi [6]. Starší verze NAOqi 2.5 poskytuje API pro programovací jazyky Python 2.7, Javascript a C++, zatímco novější verze NAOqi 2.9 umožňuje robota programovat jen pomocí jazyka Java, kontrétně formou Android aplikace. Vzhledem k tomu, že v Laboratoři inteligentních vestavných systémů na ČVUT FIT a stejně tak i u mobilního operátora jsou k dispozici pouze roboti s NAOqi verze 2.5, existenci verze 2.9 pro účely práce vypusťme.

Nutno je ovšem zmínit ještě jeden způsob vývoje aplikací – pomocí programu Choreographe. Jde o grafický program, ve kterém pomocí funkčních bloků a jejich propojování „datovými cestami“ tvoří uživatel bez znalosti programovacího jazyka tzv. behaviour, neboli dílčí aplikaci pro robota [7, 8]. Tato aplikace může být následně manuálně pomocí Choreographe spuštěna na robotovi [7], nebo jí mohou být přiřazeny „triggery“ – spouštěče/stimuly, na jejichž základě bude automaticky spouštěna v průběhu robotova tzv. autonomního života. Takových dílčích aplikací může být do robota nahráno mnoho, aby dohromady tvořily jeden funkční celek [6].

2.3 Framework NAOqi

Framework NAOqi je složen z navzájem síťově komunikujících modulů. Vzhledem k jejich vysokému počtu (celkem jde nejméně o 75 modulů) uvedu jen několik příkladů. Pro přístup k paměti a vyčítání dat ze senzorů slouží nízkoúrovňový modul ALMemory, k ovládání motoriky a dalších periferií moduly ALMotion a DCM. Mezi moduly vyšší úrovně můžeme zařadit například ALFaceDetection pro rozpoznávání obličejů, ALAnimationPlayer pro přehrávání připravených pohybových animací a ALDialog, komplexní modul pro vedení dialogu. Vývojář má rovněž možnost vytvářet moduly vlastní a snadněji tak například reagovat na události, které generují ostatní moduly. [6]

Kapitola 3

Analýza

V analytické části práce nejprve představím podobné aplikace, následně provedu výčet vhodných dostupných technologií pro rozpoznávání řeči a zpracování přirozeného jazyka, konkrétně detekci záměru uživatele. Nakonec se budu zabývat možnostmi lokalizace robota Pepper v prostoru.

3.1 Existující aplikace

Pro úspěšnou implementaci své vlastní aplikace musím nejprve zjistit, jakým úskalím se při návrhu vyvarovat. V následujících čtyřech sekcích nahlédnu na podobné existující aplikace. První z nich je současná aplikace mobilního operátora, se kterou mám vlastní zkušenost a mohu ji proto detailně popsat a identifikovat její nedostatky. Zbývající tři aplikace, o kterých čerpám informace pouze z dostupných zdrojů, krátce popíšu a shrnu závěry jejich vývojářů.

3.1.1 Současná aplikace mobilního operátora

Ve společnosti T-mobile není robot Pepper žádným nováčkem. Operátor robota pokřtil jménem „T-Dee“ a s první verzí aplikace ho představil v prosinci roku 2018. Tehdy robot uměl pouze několik připravených frází, češtině ale zatím nerozuměl a zákazníci s ním museli komunikovat skrze jeho tablet. [9]

V průběhu dalšího roku se díky přispění společnosti Adastra ve spolupráci s ProBoston naučil rozumět mluvené češtině a postupně vystřídal pracovní pozice na několika českých pobočkách [10]. V současné době vítá robot T-Dee zákazníky v Magenta Experience Center, nové netradiční pobočce operátora, která se svým zákazníkům snaží přiblížit svět moderních technologií [11].

Současnou aplikaci pro robota T-Dee vytvořila již zmíněná společnost Adastra. Robot je staticky umístěn uprostřed vstupu do prodejny a po prostoru se v rámci současné aplikace nepohybuje. Při příchodu vítá zákazníky jednou z několika frází a následně sám vede konverzaci. Rychlý sled robotových vtipů a otázek nedává zákazníkovi možnost s robotem promluvit jinak,

Obrázek 3.1: Pepper T-Dee v Magenta Experience Center [2]

než odpovídáním na jeho dotazy. Trefně popsal konverzaci s robotem redaktor magazínu Světchytře.cz Rani Tolimat: „Robot zatím ovládá 250 jednoduchých frází, a když je ze sebe naučeně sype, působí trochu pouťovým dojmem.“ [12]

Pokud se zákazníkovi podaří dostat ke slovu, robot umí zodpovědět tři dotazy: „Jak se máš?“, „Jaké bude počasí?“ a „Máš rodinu?“, přičemž po otázce se tazateli dostane vždy stejné předučené odpovědi. V praxi se však nejspíš tazateli žádné odpovědi nedostane, protože rozpoznávání řeči je nedokonalé a v rušném prostředí prodejny robot člověku nerozumí. Pro tento případ má připravenou speciální vyhýbavou odpověď „Já nevím, asi jo, ale nejsem si jist“.

Robot T-Dee není připojen k žádnému internímu ani externímu systému, proto dotazy týkající se například počasí nebo událostí na pobočce neumí zodpovědět. Přestože sám operátor v tiskové zprávě uvádí, že „...T-Dee je schopný rozeznat obličeje a také až pět základních lidských emocí. Podle toho pak přizpůsobí svou komunikaci ...“ [9], ve skutečnosti aplikace tyto funkce, které poskytuje jeden z modulů frameworku NAOqi, nevyužívá.

Co však nelze robotově aplikaci vytknout je webová aplikace tabletu, která je velmi estetická a vhodně doplňuje robotův dialog. Zároveň robot vhodně využívá zvukové sady od společnosti Aldebaran Robotics a dvou tanečních aplikací stažených z obchodu společnosti SoftBank Robotics. Začátek naslouchání robot indikuje modrou animací v očích a uších a krátkými zvukovými signály, což je výchozí vlastností použitého modulu ALDialog, respek-

tive ALSpeechRecognition[1]. Z mého vlastního pozorování interakcí robota se zákazníky usuzuji, že člověku, který s robotem Pepper komunikuje poprvé, taková indikace není dostatečně srozumitelná.

3.1.2 Robot Pepper v roli recepčního

Návrhem a implementací aplikace pro robota – recepčního ve společnosti Wegree se zabývali na technické univerzitě v polské Opole. Zde autoři článku vyhodnotili jako nedostačující modul pro rozpoznávání řeči a rozpoznávání osob. [13]

V rámci rozšíření kognitivních a interakčních schopností robota se rozhodli připojit externí mikrofon a kamery, jejichž zpracování bude mít navíc na starosti externí server. Mikrofon, který nebude spjat s robotovým tělem, by měl předejít několika problémům. Konkrétně ho na rozdíl od interního mikrofonu v robotově hlavě nebude ovlivňovat ruch od motorů a ventilátorů a změna úrovně signálu podle naklonění hlavy. Přidání dalších kamer potom autoři obhajují tvrzením, že robotovy časté pohyby hlavou spolu s pomalým hardwarem vedou k příliš dlouhé prodlevě při detekci člověka (údajně až 10 sekund) [13]. Nutno však zmínit, že ke zhoršení odezvy modulu pro detekci osob pravděpodobně dochází v důsledku obstrukce senzorů v základně robota, ta je totiž skryta za recepčním stolem.

Rozpoznávání řeči autoři implementovali pomocí služby Google Cloud Speech-to-Text. Zároveň na základě ankety mezi testovacími subjekty zjistili, že prodleva mezi vyslovení věty uživatelem a robotovou odpovědí nesmí přesáhnout tři sekundy, jinak konverzace není dostatečně plynulá. Nakonec provedli analýzu průměrného času odpovědi z cloudové služby. Z naměřených dat zjistili vztah k vzdálenosti hovořícího od mikrofonu – čím dál je hovořící od mikrofonu, tím déle trvá rozpoznání řeči a tím pádem i navrácení výsledku. Nicméně ani při vzdálenosti 1,2 metru nepřesáhla tato doba v průměru 1,5 sekundy [13]. V publikaci bohužel autoři dále nespecifikují, jak s přepisem konkrétně nakládají a jakým způsobem robot zvolí svou odpověď.

3.1.3 Robot Pepper na zubní klinice

Vývojem aplikace pro zubní kliniku v USA se zabývala lotyšská společnost Diatom. V článku na jejich webovém blogu krátce shrnují problémy, na které při vývoji narazili a jejich řešení.

Cílem jejich práce bylo vytvořit aplikaci, která bude využívat data ze vzdálené databáze a zobrazovat informace na robotově tabletu. Od této inovace si klinika slibovala především zpříjemnění návštěvy současným klientům, ale také potenciální zvýšení přitažlivosti pro nové klienty. [14]

Vývojáři měli za úkol integrovat Pepper se současnou aplikací, která již měla data o klientech kliniky. Rozhodli se proto rozdělit logickou a zobra-

[1] `http://doc.aldebaran.com/2-5/family/pepper_user_guide/interacting_pep.html`

zovací část, přičemž úlohu zobrazování dat převezme webová stránka na robotově tabletu a potřebná logika se bude vyhodnocovat zcela mimo robotův systém. Zde ovšem narazili na první problém – prohlížeč v tabletu (WebView) je zastaralý a nepodporuje současné vymoženosti jazyka Javascript. Museli se proto uchýlit k používání pouze základních funkcí a syntaxe jazyka. Dalším problémem, který se objevil, se ukázalo načítání objemných dat, konkrétně obrázků a videí, které trvalo tabletu příliš dlouho. Řešením bylo tento obsah při spouštění dopředu stáhnout, aby se za běhu aplikace předešlo zdlouhavému načítání. [14]

3.1.4 Robot Pepper jako průvodce muzeem

Návrhem nové architektury a „proof of concept“ aplikace pro robota Pepper pracujícího v roli průvodce muzeem se zabývali na fakultě University of Catania. Aplikace, kterou autoři pojmenovali CUMA (Cultural hUManoid Assistant) si klade vysoké cíle – mimo jiné má robot Pepper naplánovat a provést návštěvníky vybranou trasou v muzeu.

Pro aplikaci zvolili vývojáři architekturu sestávající se z dílčích aplikací – tzv. behaviours, které jsem popsal na konci sekce 2.2. Plánování cesty a samotnou jízdu mají obstarat aplikace „Tour Planner“, „Path Control“ a „Navigation“. Navigaci robota po muzeu má umožnit mapa prostředí a tzv. Naomarks – orientační body, jejichž rozpoznávání nabízí jeden z modulů frameworku. Na základě těchto orientačních bodů a s pomocí behaviour „Work Explanation“ má Pepper rozeznat a začít hovořit o vystavených dílech. Nakonec by měl robot od návštěvníků získat zpětnou vazbu. [15]

V implementační části se autoři rozhodli zaměřit na sledování tváří, konverzační schopnosti a tabletovou aplikaci robota. Po pilotním testování aplikace shledali, že robotovy systémy pro detekci tváří, věku a nálady splňují jejich očekávání. Naproti tomu byli zklamáni kvalitou rozpoznávání řeči, která je pro italský jazyk slabá a silně závisí na dikci hovořícího a okolním hluku. Autoři rovněž došli k závěru, že současné robotovo programové vybavení pro lokalizaci je pro pohyb v rozsáhlých prostorech muzea nedostačující. [15]

3.1.5 Shrnutí

Na základě analýzy výše představených projektů jsem došel k několika indiciím pro vývoj vlastní aplikace:

(a) robot nesmí člověka zahltit nepřerušitelnou smyčkou vlastní řeči (3.1.1)
(b) tabletová aplikace má na celkový dojem zásadní vliv (3.1.1)
(c) je třeba zlepšit indikaci, že robot právě naslouchá (3.1.1)
(d) je možné, že robot bude mít problém s detekcí osob (3.1.2)
(e) hluk v okolí mikrofonů pravděpodobně ztíží rozpoznávání řeči (3.1.2)
(f) prodleva mezi hlasovým příkazem zákazníka a robotovou odpovědí by neměla přesáhnout tři sekundy (3.1.2)

(g) je možné, že výpočetně náročné úlohy bude nutné přesunout mimo robotův systém (3.1.2)
(h) při vývoji tabletové aplikace bude třeba dbát na zpětnou kompatibilitu kódu v jazyce Javascript (3.1.3)
(i) tablet bude mít problém načítat objemná média (3.1.3)
(j) v aplikaci bude pravděpodobně možné využít funkce frameworku pro detekci tváří, věku a nálady (3.1.4)
(k) je potřeba zdokonalit robota v rozpoznávání řeči (3.1.1, 3.1.4)
(l) robotovo programové vybavení pro lokalizaci je nedostačující (3.1.4)

3.2 Rozpoznávání řeči

Schopnost rozumět přirozené řeči je pro humanoidní roboty naprosto zásadní [16, 17, 18]. Přestože robot Pepper je softwarem pro rozpoznávání řeči vybaven, jedná se o offline rozpoznávání omezené slovníkem[2] (alespoň tak tomu je v případě českého jazyka). To znamená, že robot musí mít připravené velké množství možných frází, které by člověk mohl vyslovit, a pokud se hovořící do jedné z frází nestrefí, robot mu nebude rozumět. Vzhledem k tomu, že v přirozeném jazyce existuje nespočet variant vyjádření stejného záměru, pro složitější a robustnější aplikace tato technologie není vhodná [18, s. 2–3]. Zároveň dále nezvažuji použití offline služeb vzhledem k zastaralému hardwarovému vybavení robota a jeho omezenému výkonu [17].

V následujících sekcích představím tři populární cloudové služby pro rozpoznávání řeči, které zmíněnými neduhy netrpí, zato jejich provoz zpravidla vyžaduje vynaložení nezanedbatelných finančních prostředků a spolehlivé připojení k internetu. V závěru této sekce zhodnotím jejich vlastnosti a na základě těchto zjištění vyberu nejlepšího kandidáta pro svou aplikaci.

3.2.1 Azure Speech to Text

Společnost Microsoft nabízí cloudové řešení pro rozpoznávání řeči v rámci její platformy Azure. Nabízí transkripci v reálném čase, ale i zpracování audio souborů pro více než 100 světových jazyků. Zároveň je možné přizpůsobit model vlastnímu odvětví nebo prostředí. Pro komunikaci s koncovým softwarem nabízí REST API a SDK pro několik populárních programovacích jazyků. Požadavky na REST API jsou však omezeny maximální délkou 60 sekund a neposkytují mezivýsledky převodu. [19]

Cena za hodinu převodu řeči na text (bez dodatečných služeb) je stanovena na 1 USD za hodinu převedeného zvuku, přičemž prvních pět hodin v měsíci je zdarma. Jako další služby nabízí Azure v rámci „Speech Services“ identifikaci mluvčího, klíčových slov nebo například okamžitý překlad. [19]

3.2.2 Watson Speech to Text

Cloudové řešení Watson Speech to Text od společnosti IBM se mimo možnosti využití několika rozhraní chlubí také mnoha doplňkovými funkcemi. Jedná se například o časové značky a konfidenční skóre pro jednotlivá slova nebo filtrování vulgárních výrazů a identifikace mluvčího. Zásadním rozdílem je, že tyto funkce oproti některým jiným službám zvlášť nezpoplatňuje. Watson Speech to Text nabízí WebSocket rozhraní a synchronní a asynchronní HTTP rozhraní. Služba se rovněž pyšní možností dotrénování rozpoznávacího modelu pro specifické prostředí zákazníka. [20]

[2] `http://doc.aldebaran.com/2-5/naoqi/audio/alspeechrecognition.html`

Cena za transkripci se odvíjí od frekvence využívání služby. Pokud délka převedeného audia přesáhne v měsíci milion minut, cena za hodinu audia je 0,6 USD, v případě menšího využití je cena dvojnásobná, tedy 1,2 USD za hodinu přepisu. [20]

3.2.3 Google Cloud Speech-to-Text

Rozpoznávání řeči od společnosti Google slibuje podobné výhody jako konkurence. Navíc ovšem nabízí automatické doplňování interpunkce a jazykovou podporu více než 125 světových jazyků a jejich variant. Samozřejmostí je streamování audia k přepisu, přizpůsobení modelu podle potřeb zákazníka a automatické odfiltrování šumu z dodaných nahrávek. Mimo jiné Google nabízí rozhraní REST API a knihovny pro jazyky Go, Java, Node.js a Python. Velkou výhodou je možnost získat volný kredit ve výši 300 USD na vyzkoušení služby při prvotní registraci. [21]

Účtování probíhá na rozdíl od ostatních služeb po 15 vteřinách, cena se navíc dále odvíjí od vybraného modelu a souhlasu s využitím dat pro zlepšování služeb. Po přepočtu se cena za hodinu rozpoznávání řeči vyšplhá na 0,96 až 2,16 USD za hodinu. První hodina přepisu je každý měsíc zdarma. [21]

3.2.4 Shrnutí

Všechny tři představené cloudové služby splňují požadavky pro nasazení v navrhované aplikaci. Ve stručné srovnávací tabulce 3.1 lze vidět, že služby bez výjimky podporují češtinu i streamování audia. Azure Speech to Text nicméně nemá SDK pro jazyk Python ve verzi 2.7, který jsem nucen použít, a přestože lze komunikaci se službou implementovat i bez knihovny, bylo by to zbytečně pracné. V posledním sloupci sleduji nejnižší dosažitelnou cenu za hodinu transkripce, přičemž u služby Watson Speech to Text nepovažuji za možné s jediným robotem dosáhnout milionu odstreamovaných minut měsíčně, což by umožnilo dosáhnout na nižší hodinovou sazbu 0,6 USD.

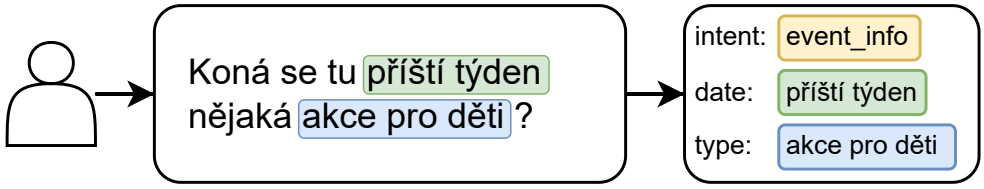
	čeština	streamování	Python 2.7	cena/hod.
Google Cloud STT	ano	ano	ano	0,96 USD
Azure STT	ano	ano	ne	1,00 USD
Watson STT	ano	ano	ano	1,20 USD

Tabulka 3.1: Srovnávací tabulka služeb pro rozpoznávání řeči

Vítězným kandidátem se stala služba Google Cloud Speech-to-Text, nejen díky nejnižší ceně po vyřazení Azure Speech to Text, ale také kvůli bonusovému kreditu pro nově registrované uživatele, který přijde vhod při implementaci a testování aplikace.

3.3 Zpracování přirozeného jazyka

Po rozpoznání řeči následuje další klíčová úloha – zpracování přirozeného jazyka. Už víme, co robotovi uživatel říká, teď ale ještě potřebujeme vědět, co od robota skutečně chce. Cílem takového zpracování je v případě mé aplikace zjištění „záměru uživatele“ (z anglického „intent detection“). Mimo záměru mohou být při zpracování vyhledány také „entity“ – části analyzované řeči nesoucí význam a „sentiment“ – emoce mluvčího [22, 23]. Na obrázku 3.2 uvádím příklad rozpoznání záměru a entit z dotazu „Koná se tu příští týden nějaká akce pro děti?“ – záměr je získání informací o událostech a extrahované entity jsou datum v lidském formátu (příští týden) a typ akce (pro děti).



Obrázek 3.2: Příklad zpracování přirozeného jazyka

Taková analýza může být provedena algoritmicky vyhledáváním vzorů, ale dnes již díky rychlému rozvoji technologií strojového učení probíhá spíše pomocí neuronových sítí [23, 24]. Cloudových služeb, knihoven a programů, které umožňují zpracování přirozeného jazyka je celá řada. V další sekci uvedu opět vzhledem k hardwarovým omezením pouze několik cloudových řešení a na závěr vyhledám nejvhodnější alternativu pro svou aplikaci.

3.3.1 Google Dialogflow

Velmi známé řešení nabízí společnost Google. Původní platforma API.AI, přejmenovaná na Dialogflow nabízí zpracování přirozeného jazyka a tvorbu „agentů“ – chatbotů. Služba nabízí několik „edicí“, které se liší v ceně za relaci a požadavek. Výhodou je možnost využít nejen textový ale i hlasový vstup. Zajímavá je dále možnost využít edici „ES Agent“, která ve zkušebním provozu dovoluje do dosažení určitého objemu požadavků službu (včetně rozpoznávání řeči) užívat zcela zdarma. Trénování probíhá ve webovém grafickém rozhraní. Služba Google Dialogflow je čistě cloudová, neumožňuje lokální nasazení. Zásadním nedostatkem je absence podpory českého jazyka. [25]

3.3.2 Microsoft LUIS

Cloudové řešení Language Understanding (LUIS) od společnosti Microsoft v rámci platformy Azure umožňuje zpracovávat přirozený jazyk. Služba nabízí REST API a SDK pro C#, Javascript a Python. Cena se odvíjí od zvoleného

režimu, v případě nízkého využití s pouze textovým vstupem lze využít až milionu požadavku měsíčně zdarma. Opět bohužel chybí podpora češtiny. [26]

3.3.3 Wit.ai

Další cloudové řešení nabízí společnost Facebook pod názvem Wit.ai. Na rozdíl od ostatních služeb neumožňuje vytvářet konverzační boty, zato její používání není zpoplatněno, a to včetně možnosti využít hlasový vstup. Rozpoznávání řeči je bohužel bez podpory češtiny, textový vstup si však s češtinou poradí. Služba nabízí HTTP API a několik SDK pro programovací jazyky, Python nevyjímaje. Trénování opět probíhá pomocí intuitivního webového rozhraní, případně pomocí REST API. [27]

3.3.4 RASA

Společnost RASA nabízí open source framework pro zpracování přirozeného jazyka a budování chatbotů. Na rozdíl od podobných řešení funguje lokálně. Framework je naprogramovaný v jazyce Python 3 a může být jednoduše nainstalován pomocí příkazu pip. Před natrénováním umělé inteligence je potřeba vyplnit textové konfigurační soubory, po dokončení trénování a otestování je možné spustit lokální server a skrze HTTP API odesílat požadavky na zpracování řeči. V případě frameworku RASA je možné využít pouze textových vstupů, pro zpracování hlasového vstupu je zapotřebí externího rozpoznávání. [28]

3.3.5 Shrnutí

Cloudových služeb pro zpracování přirozeného jazyka, které podporují češtinu, existuje jen hrstka. Vzhledem k tomu, že pro svou aplikaci plánuji využít konverzační modul frameworku NAOqi ALDialog, nepotřebuji službu s vlastním řešením vytváření chatbotů. Jako nejvhodnější službu pro svou aplikaci jsem vyhodnotil Wit.ai a to z důvodu, že její provoz je zdarma, podporuje češtinu a ke komunikaci s ní mohu využít jednoduchého HTTP API.

3.4 Lokalizace a navigace v prostoru

Úloha lokalizace a navigace robota Pepper v prostoru není snadná. Robotův slabý výpočetní hardware, laserové senzory s pomalým přístupem a nízkým rozlišením [29, 30, 31, 32] a omezené možnosti instalace software dále komplikují její řešení. Zároveň však pro mou aplikaci nepřipadá v úvahu dovybavení robota Pepper externími senzory, jelikož taková modifikace by znemožnila nasazení na libovolném robotovi. V této části práce představím techniku SLAM, algoritmus particle filter localization, alternativní metodu lokalizace pomocí UWB technologie a ve shrnutí nastíním návrh vlastního přístupu k lokalizaci.

Jak jsem již zmínil v kapitole 2, robot Pepper disponuje vybavením pro lokalizaci, a to jak softwarovým (modul ALNavigation), tak hardwarovým (laserové senzory, kamery, 3D senzor). Bohužel, přestože v minimalistických scénářích lze s modulem ALNavigation dosáhnout zdánlivého úspěchu [32], pro navigaci v rozsáhlých prostředích se ukazuje jako nevhodný. I proto volí autoři řešení pro navigaci a lokalizaci robota v prostoru jiné přístupy, konkrétně „visual SLAM“ [29, 30, 31]. Všichni zmínění autoři zároveň využívají výhod frameworku ROS, který poskytuje softwarové nástroje pro usnadnění vývoje programů pro širokou škálu nejen humanoidních robotů [33]. Klíčem k usnadnění tohoto vývoje je především hardwarová abstrakce a množství dostupných balíčků a nástrojů [33].

3.4.1 SLAM

Simultaneous Localization and Mapping (SLAM) je technika, která se používá k mapování a navigaci v původně neznámém prostředí [34]. Pokud se jako vstup použijí pouze obrazové informace z kamer, hovoříme o visual SLAM (vSLAM) [34]. Důležité je poznamenat, že u vSLAM se bude ve většině případů jednat o 3D SLAM, zatímco u technik založených například na 360° laserových senzorech půjde o 2D SLAM.

Existuje několik přístupů k vSLAM. Nejranějším z nich můžeme označit „feature-based“ přístup, který je založen na detekci tzv. feature points – význačných bodů z obrazu kamer. Pro holá prostředí, ve kterých bychom tyto „feature points“ hledali marně, nebo jen velmi stěží, existuje „direct“ přístup, který pro lokalizaci a mapování používá celý snímek. S příchodem levných RGB-D senzorů vyvstal ještě jeden přístup, který v potaz mimo snímek jako celek bere i zjištěné vzdálenosti. [34]

Algoritmus SLAM lze zjednodušeně popsat na příkladu feature-based implementace. Pro inicializaci algoritmu je zpravidla vytvořen nový souřadnicový systém, v jehož středu se robot nachází. V dalších iteracích je již nová robotova poloha zjišťována pomocí odometrie. Následně jsou získána data z vybraných senzorů pro lokalizaci a pomocí algoritmu (pro obrazová data například Scale-Invariant Feature Transform) jsou nalezeny vhodné význačné body. Ty jsou poté porovnány s dříve pozorovanými body a na základě analýzy posunu těchto

bodů je zjištěna změna v poloze robota. Dalším krokem je sloučení nových bodů s těmi dříve uloženými a proces se může zopakovat. Pokud dojde k takovému pohybu, po kterém změna v poloze nemůže být zjištěna, musí dojít k „relokalizaci“. [35]

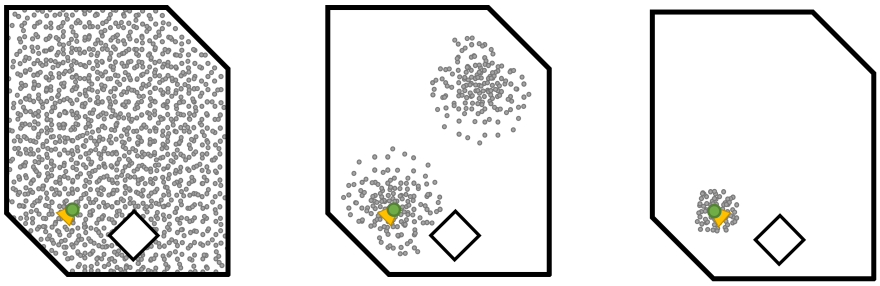
Detailní popis „feature-based“ SLAM algoritmů nabízí [35]. Mimo jiné uvádí i rozlišení význačných bodů podle jejich škály. Přestože zpravidla jsou pro SLAM využívány miniaturní body označující například ohraničení kontury, lze využít i ploch v obrazu – plochy stěn, stolů, případně i skutečné objekty v prostředí – židle, stoly, klávesnice, lahve...

3.4.1.1 Particle filter localization

Jeden z algoritmů, který lze v technice SLAM využít, se nazývá particle filter localization (do češtiny particle filter řídce překládáno jako částicový filtr), případně synonymně Monte Carlo localization [35, 36]. Algoritmus spoléhá na principy Bayesovské statistiky, především podmíněné pravděpodobnosti.

Použití pro lokalizaci robota v prostoru vypadá následovně. Jelikož se jedná o algoritmus typu Monte Carlo, velkou roli hraje v algoritmu i náhoda. Na počátku je vygenerováno velké množství náhodně rozmístěných částic (particles), které reprezentují domnělou pozici robota. Každé částici je přiřazena souřadnice v připravené mapě, úhel a pravděpodobnost, která určuje, jak dobře daná částice robotovu polohu reprezentuje. Následně je na základě senzorických dat robota (například vzdáleností od překážek, které dodají laserové senzory) pro každou částici vyhodnocena míra shody jejího pozorování ve vytvořené mapě s pozorováním robota v reálném světě. Díky následnému procesu „resampling“ je eliminována podmnožina částic s extrémně malou pravděpodobností a tyto částice jsou náhodně přemístěny do okolí částic s vyšší pravděpodobností. Postup odhadu polohy pomocí algoritmu particle filter localization je demonstrován na obrázku 3.3.

Formální vysvětlení principu částicového filtru nabízí [37].



Obrázek 3.3: Postup odhadu pomocí algoritmu Monte Carlo localization

3.4.2 Alternativní způsob lokalizace

V testovacím režimu je na pobočce operátora nasazen systém pro lokalizaci zaměstnanců pomocí čipových karet, který s největší pravděpodobností využívá technologii UWB, neboli ultraširoké pásmo. Tento systém by robotovi mohl umožnit lokalizaci bez nutnosti spolehnutí na vlastní senzory a ušetřil by drahocenný výkon. Praktické nasazení prozatím není možné a to především z důvodu nedostatku času a testovací povaze věci, nicméně v budoucnosti by robot mohl tohoto systému využít.

3.4.3 Shrnutí

Na základě analýzy předchozích řešení lokalizace robota Pepper jsem se rozhodl navrhnout vlastní přístup. Především kvůli pesimistickému závěru diplomové práce [31], která se detailně zabývala lokalizací Pepper v prostoru, jsem zavrhl využití SLAM a místo toho jsem navrhl řešení využívající orientační body velkého měřítka, které zmiňuje [35]. Konkrétní vybrané orientační body v pobočce operátora naučím rozpoznávat neuronovou síť, mapu prostoru se zanesenými body vytvořím manuálně a pro následnou lokalizaci využiji algoritmus particle filter localization. Návrh své metody popisuji v sekci 4.3.

Kapitola 4

Návrh

4.1 Požadavky

Před zahájením návrhu je nutné provést rozbor zadání a stanovit požadavky pro mou aplikaci. Je vhodné poznamenat, že jednotlivé požadavky, které se do zadání, ale i práce jako celku, promítly, vznikaly na základě konzultací se zaměstnanci a vedením prodejny. Na základě těchto požadavků lze následně navrhnout architekturu aplikace a schematicky znázornit propojení systému s externími službami. U každého požadavku v následujícím seznamu uvádím, jakým způsobem bude naplněn.

- **Samostatný provoz na prodejně mobilního operátora**
  Jelikož navrhuji aplikaci, která musí být funkční v reálném světě a na jakémkoli robotovi, nepřipadá v úvahu upravovat hardware konkrétního robota, například přidáním externích senzorů. Nelze počítat ani s nasazením výkonného serveru, na kterém by aplikace byla spuštěna a proto se veškeré výpočetní úlohy (s výjimkou rozpoznávání objektů) budou spouštět přímo na robotovi. Zároveň se nelze spolehnout na možnost výrazně upravit prostředí prodejny, například vylepením značek pro účely usnadnění robotovy lokalizace.

- **Implementace chatbota s kvalitním hlasovým vstupem**
  Na základě analýzy v teoretické části bude pro účely zkvalitnění hlasového vstupu využita cloudová služba Google Cloud Speech-to-Text. Zpracování přirozeného jazyka bude probíhat díky službě Wit.ai. Pro vedení dialogu bude přizpůsoben modul robotova frameworku ALDialog, který je založen na předem definovaných konverzacích. Zatímco doteď byl modul řízen přímo přepisem hlasového vstupu, nyní bude řízen především detekovaným záměrem a entitami.

- **Poskytování informací z interních databází operátora**
  Pro účely podávání informací je po domluvě s operátorem možné využít

rozhraní do systému pro správu konaných událostí. Do této databáze se robot bude připojovat skrze klíčem zabezpečené webové API a vytěžené informace bude následně podávat při konverzaci zákazníkům.

- **Využití robotova tabletu**
  S ohledem na analýzu stávající aplikace bude návrhu a implementaci webové aplikace věnována nezanedbatelná pozornost. Tabletová aplikace bude zobrazovat robotem vyslovovaný text mimo jiné pro zvýšení přístupnosti robota nedoslýchavým lidem. Rovněž bude během robotova naslouchání zobrazovat transkripci hlasového vstupu za účelem dosažení vyšší transparentnosti aplikace – člověk si bude vědom, co mu robot rozuměl a případně bude moci svou mluvu přizpůsobit tak, aby mu robot rozuměl lépe. Pro stydlivé zákazníky, kteří na robota nechtějí hovořit, budou na tabletu rovněž k dispozici připravené „rychlé odpovědi“, které lze vybrat pouhým dotykem. Vzhled aplikace a jeho vlastnosti budou inspirované současnými hlasovými asistenty.

- **Vhodné pohybové kreace a gestikulace**
  Gestikulace bude zajištěna použitím modulu frameworku, který během syntézy řeči spouští náhodné i kontextuální animace hlavy, těla a rukou. Pohybové kreace ve formě tanců s hudebním doprovodem budou staženy z obchodu s aplikacemi SoftBank Robotics a vhodně zakomponovány.

- **Zmapování možnosti pohybu robota se zákazníky ve formě komentované prohlídky**
  Pro pohyb robota se zákazníky je zásadní možnost robota Pepper zjistit svoji polohu v prostoru. Na základě analýzy několika prací na téma lokalizace a navigace robota Pepper jsem se rozhodl navrhnout vlastní systém pro lokalizaci. Ten díky identifikaci předučených orientačních bodů a zjištění jejich vzdálenosti od robota na základě známé velikosti bude odhadovat pozici robota v prodejně. Identifikaci orientačních bodů z obrázků kamer umožní neuronová síť, vyhodnocování bude pro nedostatečný výkon hardwaru přesunuto na externí server. Funkcionalita komentované prohlídky bude vzhledem ke komplikacím implementována jen zčásti. Robot bude umět odhadnout svou pozici a hovořit o vystavených technologiích, ale nebude schopen pohybu (viz 6.1.1).

- **Konfigurace a správa aplikace přes webové rozhraní**
  Pro konfiguraci a správu aplikace bude vytvořeno zabezpečené webové rozhraní. Pro účely hostování tohoto rozhraní bude vytvořen v hlavě robota webový server (v souladu s bodem (g) v sekci 3.1.5). Zaměstnanci skrze tuto aplikaci budou moci mj. přepnout robota do „předváděcího režimu“, ve kterém robot přestává reagovat na přítomnost člověka a do jeho úst a na jeho tablet lze vkládat libovolný text. Zároveň lze zobrazit mapu prodejny a sledovat v ní průběh robotovy lokalizace.

4.2 Návrh konverzace

Po spuštění aplikace bude robot aktivně vyhledávat kontakt se zákazníky. Pokud člověka spatří v dálce, pozdraví ho, nebo mu zamává, pokud se člověk přiblíží, nebo robota osloví, robot začne s člověkem konverzovat.

Konverzace s robotem se budou řídit scénáři, které je třeba nejprve vytvořit. Pro svou aplikaci jsem navrhl následující konverzační témata a scénáře:

- pozdrav; dotaz, s čím může robot pomoci
- rozhovor o náladě
- zákazník chce slevu na některou ze služeb operátora
- zákazník chce znát aktuální čas nebo datum
- rozhovor o robotově programátorovi/FIT ČVUT
- zákazník chce znát předpověď počasí (neimplementováno)
- zákazník se chce dozvědět, co robot umí
- zákazník chce, aby robot zatančil
- zákazník se chce s robotem vyfotit
- zákazník si chce zahrát hru (kvíz)
- zákazník chce slyšet vtip
- zákazník se chce dozvědět o konaných událostech
- zákazník se chce dozvědět o vystavených technologiích

Konverzaci robot začne pozdravem a dotazem, jak může zákazníkovi pomoci. Následně čeká na odpověď, ať už hlasovou, nebo pomocí volby rychlé odpovědi, která bude na základě konverzačního scénáře připravena na tabletu. V případě, že člověk odpovídá, konverzace se drží pravidel definovaných ve scénářích. V případě, že člověk neodpovídá, robot po chvíli sám nadhodí konverzační téma. Pokud ani poté člověk nereaguje, robot ukončí konverzaci – tedy nehlídá přítomnost člověka při konverzaci, pro skončení se spoléhá pouze na vypršení času od poslední odpovědi.

Konverzace o událostech
Pokud se zákazník chce dozvědět o konaných událostech, má několik možností. Jestliže se zeptá pouze obecně a neudá datum, nebo časový rámec, robot mu nabídne jednu z „TOP“ akcí vybraných operátorem. V případě, že se zákazník zeptá na akce v konkrétní měsíc, nebo ho zajímají třeba akce příští týden, robot v databázi operátora vyhledá akce pro žádaný časový interval. Pro zpracování data v „lidském formátu“ bude pravděpodobně potřeba využít knihovny.

Zákazníka o akcích informuje verbálně, vyjmenuje názvy akcí, jejich datum konání a zda jsou volně přístupné, nebo je nutné se registrovat. Pokud je vyhledána konkrétní akce, tabletová aplikace zobrazí QR kód s odkazem na událost ve webovém kalendáři operátora. Pokud došlo při vyhledávání k jakékoli chybě, robot ji zamlčí a na tabletu zobrazí pouze odkaz na kalendář všech akcí.

Kvíz
Jedinou „hrou“, kterou si v rámci mé aplikace bude s robotem zákazník moci zahrát, bude kvíz. Kvízy budou opět realizovány jako konverzační scénáře, avšak jejich generování proběhne automaticky z konfiguračních souborů. Kvízy budou rozlišeny svým tématem. Jedna hra kvízu na dané téma se bude vždy sestávat ze tří otázek. V rámci kvízu budou otázky dále rozděleny do kategorií, v jedné hře bude vždy maximálně jedna otázka z každé kategorie. Nebude probíhat žádné vyhodnocení kvízu, půjde jen o zábavný prvek konverzace.

Konverzace o vystavených technologiích
Operátor ve své prodejně představuje moderní technologie a snaží se s nimi seznámit zákazníky. Tuto práci převezme i robot Pepper. Bude odpovídat na otázky typu „pověz mi, co tu máte?“, „co tu vystavujete?“, zároveň, pokud bude konverzace stát, bude zákazníka zásobovat zajímavými fakty („Věděl jsi, že v našem hydroponickém stojanu vypěstujeme za měsíc hlávku salátu?“).

Konverzace o programátorovi
Jako zajímavost nabídne robot informaci o autorovi jeho aplikace. Přitom představí Laboratoř inteligentních vestavných systémů, roboty Ingrid, Vandu a Lauru a pozve zákazníka na den otevřených dveří FIT ČVUT.

Konverzace o čase a datu
Další pomocnou součástí konverzace bude možnost na dotázání informovat zákazníka o aktuálním čase a datu. Pro vyslovení data a času bude opět potřeba vytvořit sadu funkcí, které převedou strukturovaný počítačový formát data a času do formátu lidské řeči.

4.3 Metoda lokalizace

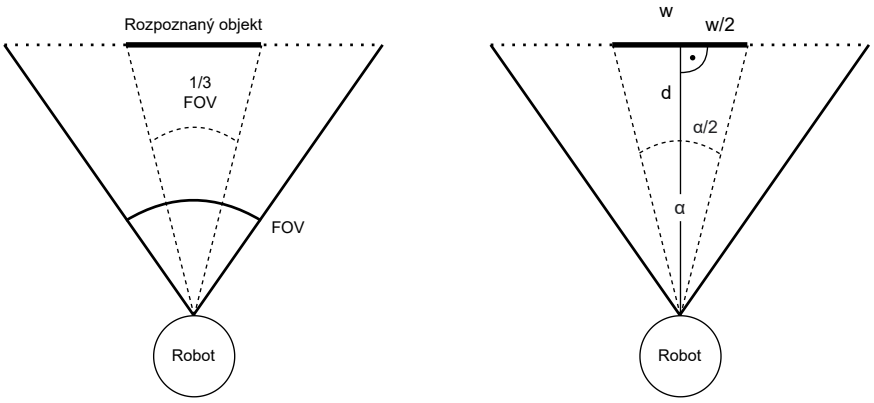
V této sekci vysvětluji svou metodu lokalizace robota Pepper v prostoru. Původní myšlenku využití rozpoznávání objektů pro účely lokalizace jsem získal díky semestrální práci „Rozpoznávání objektů v budově“ z předmětu MI-IVS kolegů Kasalického a Jiránkové.

Obrázek 4.1: Rozpoznávání objektů

Prvním krokem mé metody lokalizace je rozpoznání jednoho či více objektů v robotově okolí, to ilustruje obrázek 4.1. K tomu lze využít robotovu kameru a software pro identifikaci předem určených objektů, v mém případě bude rozpoznávání zajišťovat předučená neuronová síť.

Následně lze na základě informace o skutečné šířce rozpoznaného objektu a šířce v kamerových úhlech dopočítat přibližnou vzdálenost robota od objektu. Za předpokladu, že je objekt přímo před kamerou, jako na obrázku 4.2, můžeme vzdálenost vypočítat pomocí vzorce 4.1.

$$d = \frac{w}{2 * tg(\alpha/2)} \tag{4.1}$$



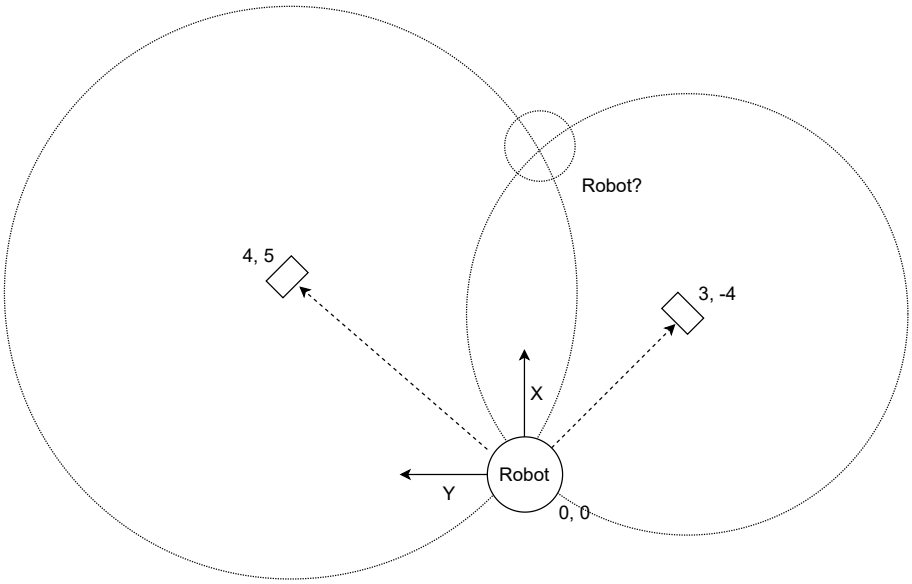
Obrázek 4.2: Ilustrace výpočtu vzdálenosti od objektu

Přesnost tohoto výpočtu se samozřejmě sníží při pohledu na rozpoznaný objekt z různých úhlů. Alternativou by mohlo být zjištění vzdálenosti pomocí 3D senzoru. Z mých dřívějších experimentů s tímto senzorem v rámci semestrální práce z předmětu BI-ZIVS však vím, že ani tato metoda by nebyla spolehlivá. Robustnější řešení by pro zjištění přesnějšího odhadu mohlo používat kombinaci těchto metod.

Ve chvíli, kdy známe přibližnou vzdálenost objektu od robotovy kamery, je potřeba dále vzít v úvahu, že robotova hlava není statická, ale může se otáčet a naklánět nahoru a dolů. Pro zjištění relativních souřadnic spatřeného objektu nám pomůže sada nástrojů knihovny ALMath a modulu ALMotion, který pro výpočet připraví transformační matici. Výsledkem výpočtů jsou vůči robotově podstavě relativní souřadnice X a Y.

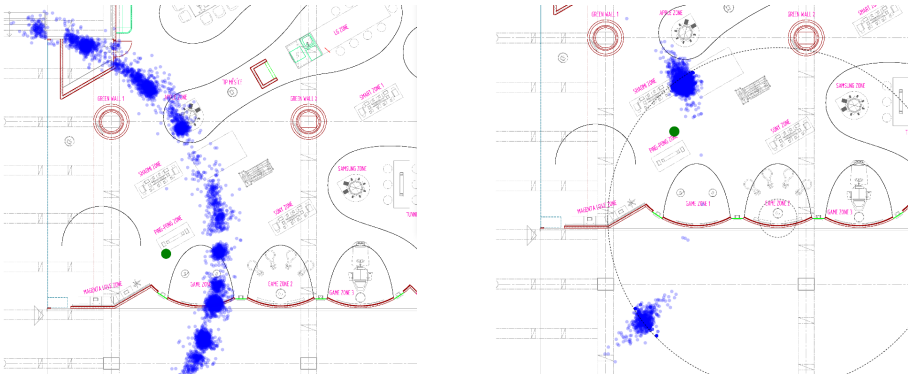
Na obrázku 4.3 je znázorněna detekce dvou známých objektů na relativních souřadnicích. Průsečíky kružnic určují, kde by se na základě takového pozorování robot mohl nacházet. Pokud detekujeme dva objekty, je zřejmé, že si nemůžeme být jisti, na které ze dvou pozic se nacházíme.

V tuto chvíli za nás však již data o relativních vzdálenostech objektů můžeme předat algoritmu particle filter localization. Přesně do tvaru kružnic



Obrázek 4.3: Zjištění polohy na pozorování objektů

a následně dvou bodů se částice na základě podobného pozorování přesunou. Na obrázku 4.4 z experimentu v prodejně lze vidět nejprve rozmístění částic okolo obvodu první kružnice, následně do dvou shluků okolo průsečíků po zpozorování druhého objektu.

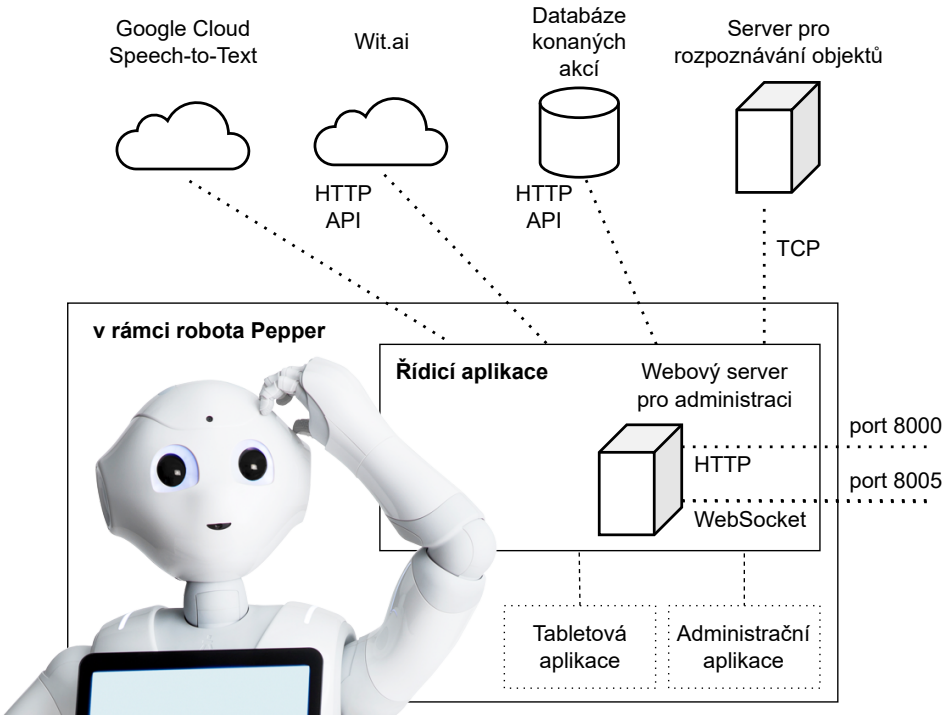


Obrázek 4.4: Průběh lokalizace

Algoritmus dále může vzít v úvahu odometrii robota a na základě odhadu úhlu natočení robota s částicemi pohybovat správným směrem. V úvahu při této simulaci musí brát i nepřesnost odometrických měření, to lze realizovat přidáním náhodné odchylky k pohybovému vektoru.

4.4 Architektura aplikace

Na základě analýzy požadavků výše lze sestavit blokové schéma aplikace (na obrázku 4.5). Jak vyplývá ze schématu, aplikace je spouštěna v rámci robotova systému a veškeré výpočetní úlohy, které systém dokáže bez přetížení zvládnout, jsou vykonávány lokálně. Zároveň je zobrazena veškerá komunikace s externími službami.

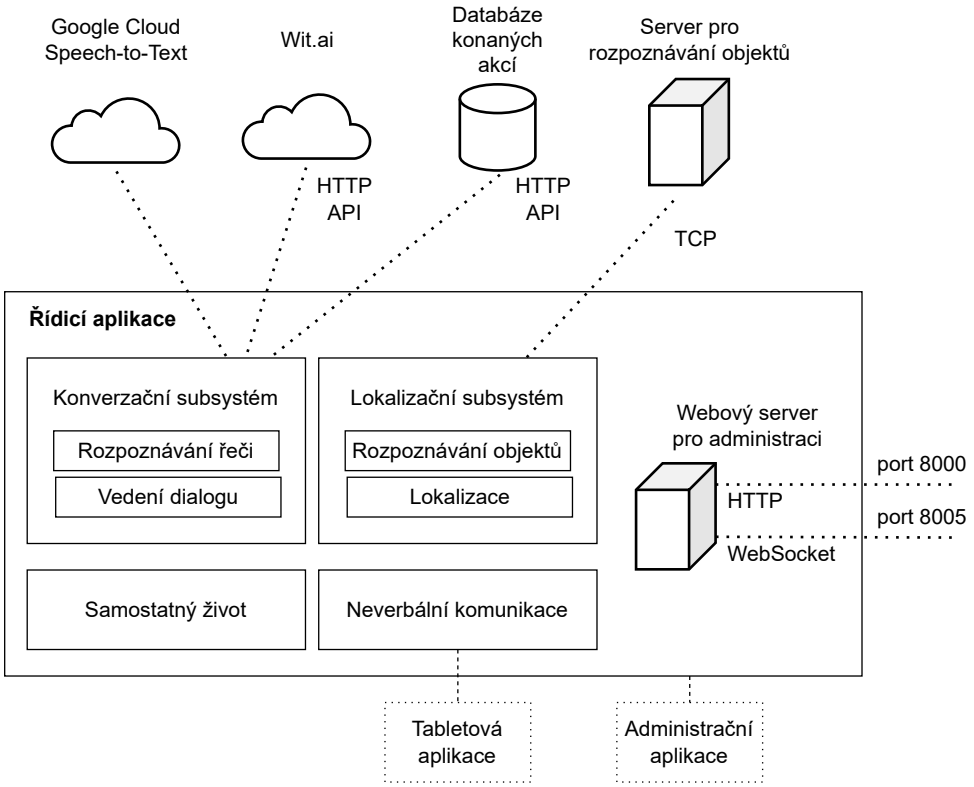


Obrázek 4.5: Blokové schéma aplikace [3]

Ve schématu jsou dále znázorněny dílčí části mé práce. Hlavní z nich je vyobrazena jako „řídicí aplikace“. Dalšími částmi jsou tabletová aplikace, administrační aplikace a server pro rozpoznávání objektů, které jsou rovněž blíže popsány v následujících podsekcích.

4.4.1 Řídicí aplikace

Na následujícím schématu jsou vyobrazeny jednotlivé součásti řídicí aplikace. Podle úloh, které v aplikaci plní, jsou rozděleny do „subsystémů“. Rovněž je blíže upřesněno, které části komunikují s externími službami. Z důvodů, které vysvětluji v kapitole Implementace, je řídicí aplikace ve skutečnosti složena z mnou vytvořených modulů frameworku NAOqi. Veškerá komunikace mezi jednotlivými moduly, ale i mezi tabletovou a webovou aplikací je tedy v režii robotova frameworku.



Obrázek 4.6: Blokové schéma řídicí aplikace

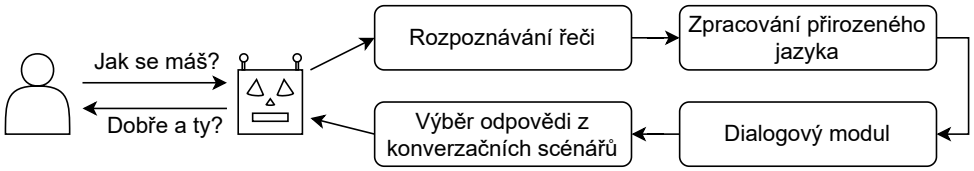
Samostatný život
Z důvodů, definovaných na začátku kapitoly Implementace, nemohu využít robotův modul pro řízení „autonomního života“ ALAutonomousLife. Ten funguje jako stavový automat, který je spuštěn po zapnutí robota a na základě předdefinovaných stimulů aktivuje aplikace („behaviours“ viz 2.2). Pro účely své aplikace jsem tento modul deaktivoval a navrhl vlastní řešení. Má aplikace je rovněž spuštěna hned po zapnutí robota a svůj životní cyklus si řídí sama pomocí stavového automatu, jenž má pouze dva stavy – robot osamocen a robot v aktivním kontaktu s člověkem. Ve chvíli, kdy robot přejde do aktivního kontaktu s člověkem se aktivuje konverzační subsystém. Stane se tak při detekci člověka v dostatečně malé vzdálenosti, při zjištění doteku na tablet nebo při rozpoznání libovolné z nakonfigurovaných frází (především pozdravů).

Konverzační subsystém
Konverzační subsystém je rozsáhlou součástí aplikace a z hlediska struktury ho lze dále rozdělit na několik dílčích částí. Zároveň plní velké množství stanovených cílů. Primární programovou součástí je vedení dialogu, to je aktivováno po navázání kontaktu s člověkem a předává pokyn k zahájení rozpoznávání řeči.

Rozpoznávání řeči bude po pokynu k naslouchání čekat na překročení nastavené hladiny okolního hluku. Posléze začne streamovat audio do cloudové služby Google Speech-to-Text a v průběhu zpracování bude předávat subsystému neverbální komunikace mezivýsledky prostřednictvím událostí. Po dokončení transkripce totožným způsobem zveřejní konečný výsledek.

V tuto chvíli režii převezme opět součást vedení dialogu. Ta provede s pomocí cloudové služby Wit.ai zpracování přirozeného jazyka. Výsledná odpověď bude formulována přizpůsobeným modulem robotova frameworku ALDialog. Vstupem do dialogového systému, který konverzaci vede podle předem připravených konverzačních scénářů, bude nyní mimo transkripci především detekovaný záměr a hodnoty entit. Modul ALDialog zároveň zajišťuje syntézu řeči a publikuje události o průběhu vyslovování textu, ty bude zpracovávat subsystém neverbální komunikace.

Robot tím pádem nebude schopen otevřeného dialogu, přestože je použita umělá inteligence ve formě zpracování přirozeného jazyka. Bude umět odpovídat jen na naučené fráze a jejich obměny. Záměry a entity budou naučeny pomocí vkládání trénovacích frází do služby Wit.ai. Hlavním důvodem pro toto rozhodnutí je veřejné prostředí prodejny, které pro experimenty s end-to-end konverzačními chatboty není vhodné.



Obrázek 4.7: Schéma konverzačního subsystému

Neverbální komunikace

Hlavním úkolem subsystému pro neverbální komunikaci je zpracování událostí od ostatních součástí aplikace a předání informací a příkazů tabletu a LED diodám v různých částech robotova těla. Pro tuto komunikaci je opět využito služeb frameworku. Neverbální komunikace je rovněž stěžejní pro splnění stanovených požadavků.

Webový server pro administraci

Ke zpřístupnění administrační webové aplikace je potřeba nasadit ji na server přístupný v lokální síti, aby na něj mohli přistupovat zaměstnanci pobočky. Pro jednoduchost jsem proto zakomponoval jednoduchý webový server přímo do řídicí aplikace. Spolu s webovým serverem je nasazen i WebSocket server pro snadné předávání aktuálních informací v reálném čase bez nutnosti aktualizování stránky.

Lokalizační subsystém
Lokalizační subsystém má, jak název napovídá, za úkol zjišťovat současnou polohu robota Pepper v prostoru prodejny. Lokalizace robota probíhá na pozadí aplikace nepřetržitě. Vzhledem ke mnou zvolenému způsobu lokalizace pomocí identifikace a odhadu vzdálenosti k orientačním bodům v prostoru, je zapotřebí implementovat modul rozpoznávání objektů. Ten má na starost především odesílání obrazových dat na vzdálený server a následně zpracování příchozích informací o detekcích. Díky tomuto modulu lze automaticky průběžně nalézat orientační body v prostoru a mít informace o jejich poloze vůči robotovi.

Informace o detekovaných orientačních bodech následně převezme modul pro lokalizaci, který pomocí algoritmu particle filter localization dokáže odhadnout polohu robota v prodejně. Pro lokalizaci je nutné průběžně získávat data o okolních orientačních bodech a v případě, že se robot v průběhu zjišťování polohy pohybuje, jsou rovněž zapotřebí data odometrie.

4.4.2 Server pro rozpoznávání objektů

Server pro rozpoznávání objektů komunikuje prostřednictvím sítě s modulem uvnitř robota, ten mu průběžně zasílá obrazová data a server odpovídá informacemi o detekovaných objektech. Na pozadí je pro rozpoznávání využita předučená neuronová síť.

Server pro rozpoznávání objektů je jedinou (mnou vytvořenou) součástí aplikačního celku, která je nasazena mimo robota. Důvody jsou opět objasněny v následující kapitole, stručně lze však poznamenat, že robotův hardware není k detekci pomocí umělé inteligence dostatečně výkonný a jeho systém by byl touto úlohou naprosto zahlcen.

4.4.3 Tabletová aplikace

Tabletová webová aplikace bude plnit stěžejní úlohu při neverbální komunikaci s uživatelem. Jak jsem již zmínil při analýze požadavků, bude zobrazovat robotem vyslovovaný text a transkripci hlasového vstupu (viz obrázek 4.8). Toho docílí díky komunikaci se subsystémem neverbální komunikace, který tyto informace zprostředkovává ze systému pro rozpoznávání řeči a vedení dialogu. Pro zpřístupnění bude aplikace doplněna o intuitivní animace. Funkcionalita rychlých odpovědí bude realizována přímou komunikací s konverzačním subsystémem.

Obrázek 4.8: Návrh rozhraní tabletové aplikace

4.4.4 Administrační aplikace

Administrační webová aplikace bude plnit úlohu předepsanou požadavkem na správu a konfiguraci aplikace. Jak bylo naznačeno, bude se jednat o jménem a heslem zabezpečené webové rozhraní, do kterého budou mít z lokální sítě přístup pověření zaměstnanci pobočky. Pro ovládání robotových funkcí bude aplikace opět komunikovat se součástmi řídicí aplikace a to jak díky funkcím frameworku, tak prostřednictvím WebSocket serveru, který jí bude bez nutností obnovování stránky předávat aktuální informace o průběhu lokalizace. Převedení robota do předváděcího režimu bude realizováno pozastavením subsystému samostatného života a zaměstnanci tak budou moci robota ovládat bez toho, aniž by se snažil navazovat kontakt s lidmi.

Kapitola 5

Implementace

V této kapitole se budu věnovat implementaci součástí své aplikace tak, jak byly popsány v kapitole Návrh, ale také implementaci několika pomocných skriptů. Nejprve však začnu výběrem způsobu implementace a následně ještě objasním, jak obejdu omezení instalace software do robotova systému, které jsem popisoval v sekci 2.2 kapitoly Robot Pepper.

5.1 Způsob implementace

Před zahájením implementace je nutné zvolit, jakým nástrojem nebo v jakém jazyce aplikaci vytvořit. V úvahu připadá programování v nástroji Choreographe, případně využití jazyka C++ nebo Python 2.7. Pro aplikace takového rozsahu, který je ke splnění cílů zapotřebí, je použití nástroje Choreographe nevhodné, především z důvodu nepřehlednosti větších schémat a obtížné implementaci složitějších funkcionalit [8]. Svou aplikaci jsem se rozhodl programovat v jazyce Python 2.7 z důvodu pohodlnější a rychlejší implementace oproti jazyku C++ a dostupnosti velkého množství potřebných knihoven.

Řídicí aplikaci jsem navrhl tak, aby vhodně využívala a doplňovala robotův modulární framework NAOqi. Jak jsem zmínil v 2.2, do frameworku má vývojář možnost přidat vlastní moduly a dosáhnout tak několika výhod oproti odlišné implementaci. Mimo jiné lze snadněji odebírat události ostatních modulů a využít stávajícího systému logování. Některé funkce navíc mimo moduly frameworku nejsou vůbec k dispozici, konkrétně přístup k datovým bufferům z mikrofonů – bez toho se ve své aplikaci neobejdu.

Abych odlišil své moduly od původních modulů frameworku, budu místo prefixu AL- používat prefix PM- (Pepper Module). Veškerou komunikaci s frameworkem NAOqi zajišťuje pythonová knihovna qi, jejímž prostřednictvím budou moduly registrovány.

Výhodou zvoleného způsobu implementace je i usnadnění tvorby tabletové a administrační aplikace. Komunikaci mezi nimi a řídící aplikací totiž umožní modul ALTabletService a javascriptová knihovna qi. Díky tomuto modulu lze

vzdáleně spouštět javascriptové příkazy na tabletu a prostřednictvím knihovny qi lze zase přímo z webové stránky přistupovat ke všem registrovaným modulům v rámci frameworku.

Pravidlo propojení s frameworkem NAOqi poruší server pro rozpoznávání objektů. Přestože je v rámci frameworku možné vytvořit vzdálené moduly (umístěné na jiném stroji v síti), ke kterým lze přistupovat, jako by byly lokální, znamenalo by to použití Python 2.7. Vzhledem k tomu, že k rozpoznávání objektů použiji knihovnu Tensorflow, která již několik let Python 2.x nepodporuje, bylo by zbytečně náročné toto propojení nuceně vytvářet. Místo toho implementuji jednoduché síťové propojení protokolem TCP, skrze které si budou robot a server vyměňovat data.

Před tím, než mohu začít vyvíjet svou aplikaci, musím překonat první zásadní překážku. Python 2 sice dosáhl svého EOL a od 1. ledna 2020 již oficiálně není podporován Python Software Foundation, přesto je možné stahovat a používat balíčky, které tuto zastaralou verzi podporují a jazyk tak dále s určitými riziky pro vývoj použít. Podpora Python 3 je ze strany knihovny qi, která zprostředkovává komunikaci s frameworkem NAOqi, bohužel zatím čistě experimentální a v současnou chvíli není možné ji stabilně využít.

Výraznějším problémem, než nutnost shánět starší verze balíčků (navíc pro specifickou architekturu i686), je však verze Python 2.7.6, která je na robotovi k dispozici. Vzhledem k jejímu extrémnímu stáří – vydána byla v roce 2013 – má tato verze problém se zabezpečeným připojením SSL. Absence podpory SNI v praxi znamená, že není možné instalovat balíčky pomocí příkazu pip, protože ten sám potřebuje zabezpečené spojení navázat. Balíčky lze sice instalovat i manuálně, ale pokud pro svou funkcionalitu vyžadují navazování zabezpečených připojení, opět se stávají nepoužitelnými. Řešením problému by byla instalace verze Python 2.7.18, poslední verze Python 2 vydané v dubnu 2020. Ta však není možná z důvodu nepřístupnosti uživatelského účtu root.

5.1.1 Gentoo Prefix

Vytrhnout pomyslný trn z paty vývojáři pomůže tzv. Gentoo Prefix. Jedná se o linuxovou nástavbu, kterou lze nasadit na širokou škálu systémů i bez administrátorských oprávnění. Následně lze instalovat libovolné balíčky pomocí nástroje emerge a spouštět libovolné, v „prefixu“ instalované, spustitelné soubory. Zdánlivě nemožného je dosaženo díky vytvoření nového uživatelského souborového systému uvnitř toho stávajícího. [38]

Zásluhu za vytvoření Gentoo Prefix pro robota Pepper má Sam Pfeiffer. Jeho repozitář[3] dokonce převzala sama společnost SoftBank Robotics a zveřejnila na svém Github účtu. Součástí předpřipraveného balíčku je velké množství softwarových balíčků, několik verzí jazyku Python včetně 2.7.17 a ROS Kinetic/Melodic. Pro implementaci dále uvažuji použití Gentoo Prefix.

[3] `https://github.com/awesomebytes/pepper_os`

5.1.2 Rozhraní modulů

Framework NAOqi nevyžaduje od vývojářů dodržení žádného specifického rozhraní pro moduly, které registrují. Pro své moduly jsem navrhl jednoduché rozhraní a sadu konvencí, kterých se v průběhu programování držím. Rozhraní vynucuje pouze tři pravidla:

- modul obsahuje konstruktor s jediným parametrem *app*
- modul obsahuje metodu *start* bez parametrů a bez návratové hodnoty
- modul obsahuje metodu *stop* bez parametrů a bez návratové hodnoty

Prostřednictvím parametru *app* v konstruktoru je modulu předána instance qi aplikace, tu moduly využijí především pro získání referencí na jiné moduly skrze funkci *app.session.service*. V konstruktoru zároveň moduly zpravidla inicializují logger pomocí konstruktoru *qi.Logger*.

Metoda *start* slouží ke spuštění modulu. V této metodě začínají moduly především odebírat události a načítat data z konfiguračních souborů. Pokud modul pro svou funkci vyžaduje smyčku, může si ji spustit v novém vlákně. K ukončení práce modulu slouží metoda *stop*. Zde modul musí ukončit veškerou práci, zrušit odběr událostí a „uklidit“ po sobě.

5.1.3 Systém událostí

V implementaci často zmiňuji, že si jednotlivé moduly předávají informace díky událostem. Framework NAOqi má propracovaný systém událostí, který hojně používá. Jeho funkci zajišťuje modul ALMemory. K odběru událostí se lze přihlásit pomocí metody *subscribeToEvent*, parametry jsou název události, název odebírajícího modulu a název metody, která se má zavolat. Tato callback metoda musí mít tři parametry – *key*, *value* a *message*, přičemž prakticky je využit jen *value* tj. hodnota předaná událostí. Pro tuto hodnotu navíc platí omezení – lze předávat jen základní datové typy a pole, slovníky a objekty předávat nelze. Moduly mohou rovněž deklarovat vlastní události pomocí *declareEvent* a následně je vyhlašovat pomocí *raiseEvent*, kde je jako první parametr předána hodnota.

5.2 Struktura řídicí aplikace

Řídicí aplikace je, jak jsem již objasnil, soubor modulů frameworku NAOqi. Ke svému fungování však potřebuje také konfigurační a datové soubory, spouštěcí a pomocné skripty. Pro lepší představu v této sekci představím strukturu aplikace a její neprogramové součásti. V adresářovém stromu na obrázku 5.1 je naznačena struktura části aplikace nasazené v robotově systému.

```
run_operator_application.sh ..... skript pro snadné spuštění aplikace
operator_application
    config ..... adresář s konfiguračními soubory
        quizes ..... adresář s definicemi kvízů
        topics ..... adresář s konverzačními soubory
        config.py ..... hlavní konfigurační soubor
    html ..... adresář tabletové aplikace
    html_admin ..... adresář administrační aplikace
    scripts ..... adresář řídicí aplikace
        main.py ..... spouštěcí skript řídicí aplikace
    _announce_error.py ..... skript pro oznámení chyby robotem
    _fix_robot_time.py ..... skript pro nastavení systémového času
```

Obrázek 5.1: Adresářová struktura aplikace v robotově systému

5.2.1 Konfigurační soubory

Konfigurační soubory pro řídicí aplikaci jsou umístěny v adresáři *config*. Mimo hlavní konfigurační soubor *config.py*, který obsahuje veškeré konstanty (cesty, adresy a porty, fráze…) pro běh aplikace, se zde nachází soubory s definicemi kvízů a dialogu nebo například soubor *pronunciation.yaml* s „překlady“ – výslovnostmi pro robotův systém syntézy řeči.

5.2.2 Snadné spuštění aplikace

Před spuštěním řídicí aplikace je nutné podniknout několik kroků. Pro zavedení jsem připravil bash skript *run_operator_application.sh*, který tyto kroky provede automaticky a navíc poskytuje několik dalších užitečných funkcí.

Skript lze spustit se dvěma přepínači. Přepínač *-c* přesměruje veškerý výstup skriptu, potažmo řídicí aplikace, do konzole. Pokud se přepínač nepoužije, výstup aplikace je přesměrován do souboru s aktuálním časem a datem v názvu. Tento soubor s příponou *.log* je umístěn do adresáře *operator_application*. Druhým přepínačem je *-b*, který spustí aplikaci na pozadí.

Nejprve je provedena kontrola přítomnosti Google Prefix. Pokud není nalezen skript */home/nao/gentoo/executeonprefix*, skript ohlásí chybu a skončí. Následuje nastavení softlinku pro tabletovou aplikaci, ta musí být umístěna

v adresáři */home/nao/.local/share/PackageManager/apps/*. Následně jsou exportovány proměnné prostředí pro Gentoo Prefix a je spuštěn skript pro nastavení správného času. Exportování proměnných probíhá díky mnou upravenému skriptu *executeonprefix* skrze nastavení proměnných s prefixem PREFIX_. Při volání skriptu jsou vyhledány všechny proměnné prostředí s tímto prefixem a bez něj jsou exportovány v prostředí Gentoo Prefix. Na závěr je pomocí již zmíněného příkazu *executeonprefix* spuštěna aplikace (soubor *main.py* v adresáři *scripts*).

5.2.3 Nastavení správného času

V důsledku vybití CMOS baterie si někteří roboti nepamatují po restartu správný čas. To je zásadní překážkou pro veškerou zabezpečenou síťovou komunikaci. Skript *_fix_robot_time.py* nejprve získá aktuální čas z NTP serveru a následně pomocí příkazu *timedatectl* tento čas systémově nastaví.

5.2.4 Oznámení chyby za běhu aplikace

Aby robot mohl dát najevo, že se při spuštění nebo za běhu aplikace vyskytla závažná chyba, je možné využít modul frameworku ALNotificationManager. Ten má na starost indikovat jemu předaná varování nebo chyby rozblikáním LED diod na robotových ramenou. Zároveň modul zajišťuje oznámení obsahu těchto upozornění po stisku tlačítka na robotově hrudi. Pro usnadnění jsem vytvořil skript *_announce_error.py*, který tento modul využívá a jako první argument přijímá chybovou zprávu, kterou má robot oznamovat. Pokud spouštěcí skript zaznamená fatální chybu (především při inicializaci, ale i za běhu), robot ji oznámí prostřednictvím právě tohoto skriptu.

5.2.5 Hlavní spouštěcí skript řídicí aplikace

Hlavní spouštěcí pythonový skript aplikace se nazývá *main.py* a je umístěn ve složce *scripts*. Pokud uživatel není v prostředí Gentoo Prefix a nejsou nastavené proměnné prostředí, aplikaci se tímto skriptem nepodaří spustit – k tomu slouží bash skript *run_operator_application.sh*. Tento skript má za úkol:

- spustit „qi“ aplikaci
- nastavit logování
- zaregistrovat a spustit PM moduly
- odstranit předchozí chybová oznámení
- zachytávat chyby
- při ukončení korektně zastavit moduly a zrušit jejich registraci

Skript nejprve prostřednictvím knihovny qi vytvoří a spustí qi aplikaci. Z takto spuštěné aplikace lze následně získat klíčový objekt *session*. Skrze

metodu *service* tohoto objektu lze získat reference na registrované moduly frameworku. V následující ukázce kódu je znázorněno vytvoření aplikace a získání reference na modul ALMemory.

```
import qi
...
app = qi.Application()
app.start()
al_session = app.session
al_memory = al_session.service("ALMemory")
```

Následuje odstranění předešlých chybových notifikací, registrace a spuštění modulů ve stanoveném pořadí podle jejich závislostí. Jsou vytvořeny instance modulů, ty jsou dále registrovány pomocí metody *al_session.registerService* a spuštěny jejich instanční metodou *start*. Poté už nic nebrání zavolání metody *app.run*. Jde o blokující volání, které končí až při výskytu nezachycené chyby v aplikaci nebo zavolání *app.stop*. Z toho je patrné, že aplikace nemá žádnou hlavní smyčku, ale je řízena událostmi. Zachytávání chyb je vyřešeno pomocí bloku try-except. Pokud dojde k zachycení chyby, jsou zrušeny registrace modulů a program je ukončen.

5.3 Pomocné součásti řídicí aplikace

5.3.1 PronunciationHelper

Třída PronunciationHelper slouží k převodu („překladu“) textu do fonetické formy a zpět. Použití nalézá v modulech PMChat, PMSpeech a PMNonVerbal. Potřeba převodu do fonetické formy vyvstává kvůli syntéze řeči, zpětný převod je potřeba pro zobrazení textu v tabletové aplikaci. Zpětnému převodu se nedá vyhnout, protože text na tabletu se zobrazuje na základě událostí modulu syntézy řeči a k původnímu nefonetickému textu chybí přístup.

Do nové instance třídy je nejprve nutné načíst data pomocí metody *_load_pronunciation*. Ta je získá z konfiguračního souboru na cestě PRONUNCIATION_FILE_PATH (v tuto chvíli jde o soubor *pronunciation.yaml*.) Pro dva směry převodu slouží metody *translate* a *revert_translation*. Jediným parametrem funkcí je text určený k převodu. Jednotlivé fráze v textu jsou postupně překládány a to v pořadí, v jakém jsou uvedeny v konfiguračním souboru. Pro načtení překladů ze souboru ve formátu YAML slouží pomocná funkce *ordered_load*.

5.3.2 PMSpeech

Miniaturní modul PMSpeech byl vytvořen pro účely tabletové aplikace. Jeho jediná metoda *say*, na pozadí využívající třídu PronunciationHelper a ALAnimatedSpeech, slouží k překladu textu do fonetické podoby a jeho následnému vyslovení.

5.3.3 PMHumanPerception

Pomocný modul PMHumanPerception slouží k zpracování událostí modulu ALEngagementZones. Ten má za úkol sledovat osoby, které se nacházejí v robotově okolí. Podle vzdálenosti osob od robota je navíc přiřadí do jedné z tří zón.

Bohužel, při práci s rozhraním tohoto modulu jsem narazil na problém, kdy nespolehlivě vystavuje události *PersonApproached* a *PersonMovedAway*. Proto jsem vytvořil modul vlastní, který sleduje událost *EngagementZones/PeopleInZonesUpdated* a následně vyhodnocuje ALMemory proměnné *EngagementZones/PeopleInZone1* až *PeopleInZone3*. Na základě těch nejen korektně vyvolá události *PMHumanPerception/HumanMovedAway* a *HumanApproached*, ale zároveň udržuje i informaci o stavu přítomnosti člověka. Stavem může být jedna z hodnot *HumanAbsent*, *HumanInterested*, *HumanApproached* nebo *HumanEngaged*. Při změně tohoto stavu jsou informováni odběratelé události *PMHumanPerception/HumanPerceptionUpdated*.

5.4 Samostatný život

Jednou z mála nevýhod, kterou přináší využití nástavby Gentoo Prefix, je nemožnost využití modulu ALAutonomousLife. Skrze něj je možné spouštět aplikace nahrané do robotova systému dle nastavených triggerů. Přestože lze libovolnou pythonovou aplikaci zabalit do formy takto spustitelné aplikace, robot ji nedokáže spustit ve verzi Python 2.7.17 (dodané Gentoo Prefix).

5.4.1 PMLife

Pro svou aplikaci jsem navrhl modul PMLife. Stejně jako ALAutonomousLife jde o stavový automat řídící robotův samostatný život. Podobně jako zmíněný modul přechází PMLife mezi stavy, které se liší v přítomnosti člověka.

Stavy jsou reprezentovány vlastními třídami, které již jako moduly registrovány nejsou, dodržují však podobné rozhraní. Jedinou změnou je, že metoda start má parametr *passed_data*, což je slovník s daty, která může modul pro spuštění vzít v úvahu. Rodičem tříd reprezentující stavy je třída *LifeState*, která zároveň definuje metodu pro přechod mezi stavy *change_life_state*.

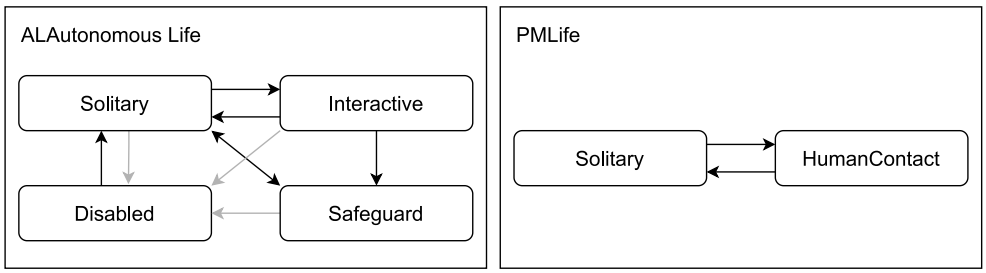
Po spuštění modulu PMLife jsou aktivovány moduly ALBackgroundMovement a ALBasicAwareness, které zajišťují, že se robot samovolně pohybuje a rozhlíží – vypadá živě. Zároveň v tu chvíli přejde robot do stavu „Solitary“ V tomto stavu robot není v kontaktu s člověkem, ale snaží se ho aktivně vyhledávat. Přechod do stavu „HumanContact“, neboli stavu, kdy je robot v přímém kontaktu s člověkem, může nastat libovolným ze tří způsobů: při detekci člověka v dostatečně malé vzdálenosti (na základě události modulu PMHumanPerception), při zjištění doteku na tablet (událost ALTabletService) nebo při rozpoznání jedné z nakonfigurovaných frází[4] (událost ALSpeechToText). Fráze pro oslovení robota jsou definovány konstantou GET_ROBOT_ATTENTION_VOCABULARY.

Stav „HumanContact“ na rozdíl od stavu „Interactive“ modulu ALAutonomousLife nespouští uložené aplikace, ale pouze aktivuje hlavní modul konverzačního subsystému PMChat metodou *start_session*, či *start_session_with_utterance* pokud byl robot osloven jednou z nakonfigurovaných frází. Jedinou další úlohou toho stavu je kontrola, zda nebyl kontakt s člověkem přerušen. Po uplynutí přednastavené doby po poslední interakci je znovu spuštěn stav „Solitary“. Toho je docíleno watchdog časovačem, který každých 2,5 sekund kontroluje návratovou hodnotu *heartbeat* modulu PMChat (viz 5.5.4 „Ukončení konverzace“).

Pro účely vývoje je připraven stav „Virtual“. Ten je určen pro virtuálního robota a stává se jediným stavem automatu, pokud je pravdivá konfigurační

[4]Pro rozpoznávání řeči ve stavu Solitary je použit robotův vlastní systém rozpoznávání řeči dodaný modulem ALSpeechToText a to z důvodu, že využití cloudové služby by bylo finančně neúnosné – účtovány jsou požadavky, ve kterých není lidská řeč rozpoznána.

konstanta IS_VIRTUAL_ROBOT. V takovém případě robot hned po spuštění modulu samostatného života aktivuje konverzační modul.



Obrázek 5.2: Diagramy stavových automatů ALAutonomousLife a PMLife

5.5 Konverzační subsystém

Konverzační subsystém je rozsáhlou součástí aplikace a jeho rozsáhlou funkcionalitu jsem rozdělil do několika modulů. V jejich popisu mohu postupovat chronologicky – od rozpoznání řeči přes zpracování přirozeného jazyka až po formulaci odpovědi. Nejprve však představím základní stavební kámen a princip tohoto subsystému.

5.5.1 Princip fungování

Základem hlavního modulu konverzačního subsystému PMChat je modul robotova frameworku ALDialog. Dříve než přejdu k implementačním detailům, musím upřesnit, jakým způsobem systém funguje a jak ho pro svoje potřeby mohu přizpůsobit.

Definice dialogů
Jak jsem uvedl v návrhu, modul ALDialog ke svému fungování potřebuje připravené scénáře konverzací. Pro definici těchto scénářů (dialogů) používá vlastní konverzační engine pojmenovaný QiChat[5]. Tento propracovaný engine je sám o sobě poměrně robustním řešením a proto jsem se rozhodl ho ve své aplikaci nenahrazovat.

Dialogové soubory pro QiChat mají vlastní syntaxi. Výběr cesty, kterou konverzace povede, určují pravidla. Na základě shody s hlasovým vstupem vybírá dialogový systém pravidlo, které může aktivovat. Pro lepší pochopení uvádím příklad jednoduchého konverzačního scénáře.

```
topic: ~smalltalk ()
language: czc

u:(ahoj) Čau!
u:(jak se máš) Mám se skvěle! Co ty?
  u1:(dobře) Jupí!
  u1:(špatně) To mě mrzí...
```

Pokud uživatel robota pozdraví – řekne „ahoj“, robot odpoví „Čau!“. Na otázku „jak se máš?“ robot odpoví „Mám se skvěle! Co ty?“ a konverzace se dále větví. Už z tohoto příkladu je patrné, že takový dialogový soubor nebude lehké vytvořit – co když uživatel odpoví „skvěle“? Nebo „nic moc“? Robot mu jednoduše nebude rozumět a nedostane se mu odpovědi.

Zpracování přirozeného jazka
Tento problém pomůže vyřešit můj přístup – začlenění zpracování přirozeného jazka, přesněji jeho části – detekce záměru a entit. Místo toho, aby mě zajímala

[5] http://doc.aldebaran.com/2-5/naoqi/interaction/dialog/aldialog.html

konkrétní slova, která uživatel vyslovil, stačí mi zjistit jeho záměr. Uvedu příklad, jak bych mohl dřívější konverzační scénář vylepšit.

```
topic: ~smalltalk_nlp ()
language: czc

u:($intent=="greet") [Čau Ahoj "Dobrý den"]
u:($intent=="ask_about_mood") Mám se skvěle! Co ty?
  u1:($intent=="tell_mood_good") Jupí!
  u1:($intent=="tell_mood_bad") To mě mrzí...
```

Syntaxe dialogového systému umožňuje pravidla aktivovat i na základě pravdivostní hodnoty. V příkladu přestavuje `$intent` proměnnou (modulu ALMemory), která nese název detekovaného záměru. Pokud se tedy shoduje detekovaný záměr, pravidlo bude vybráno. Pod jediným záměrem se může skrývat nespočet způsobů, kterými ho lze vyjádřit – je však nutné natrénovat umělou inteligenci, aby záměry rozpoznávala.

Zároveň je pro přirozenější konverzaci možné části odpovědí obměňovat. Pokud robota pozdravíme, tedy je zjištěn záměr *greet*, robot odpoví jedním z pozdravů „Čau“, „Ahoj“ nebo „Dobrý den“. Tímto způsobem lze zároveň definovat více možností i pro vstupy pravidel a vytvořit tak více alternativ, které mohou pravidla aktivovat. Je to však manuálně a časově náročná cesta k nedokonalému výsledku.

Uvedu ještě jeden příklad velmi základního přidání detekovaných entit:

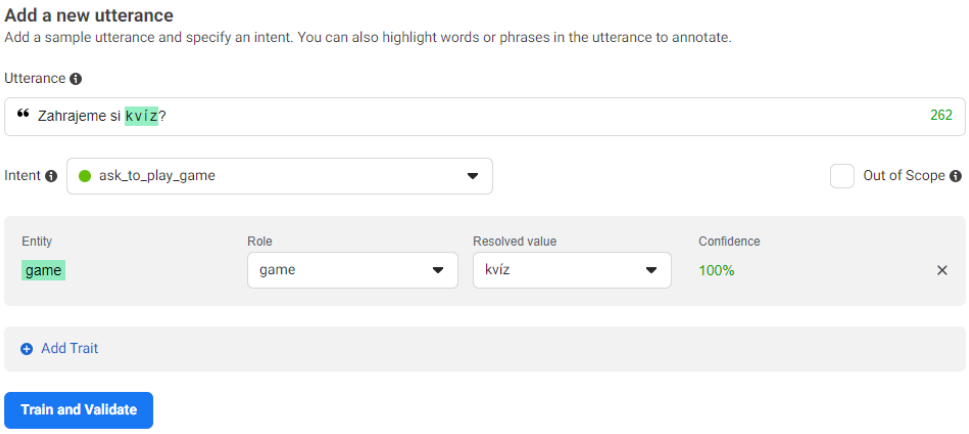
```
topic: ~smalltalk_nlp_with_entities ()
language: czc

u:($intent=="introduce_oneself")
    Těší mě, budu si pamatovat,
    že se jmenuješ $e_human_name !
```

Zde je pravidlo aktivováno pokud detekovaný záměr odpovídá *introduce_oneself*, tedy pokud se robotovi člověk představil. Jednou z entit definovaných pro tento záměr může být *human_name* (jméno člověka). Tu lze následně v konverzaci použít, jak je v ukázce vidět. Samozřejmě je i v tomto případě potřeba k rozpoznávání entit umělou inteligenci připravit.

Wit.ai

Zde přichází na řadu služba Wit.ai. Prostřednictvím jejího webového rozhraní lze snadno vytvářet trénovací fráze a učení dále probíhá automaticky bez zásahu uživatele. Na obrázku 5.3 je ukázka natrénování nové věty. Trénuji novou větu (utterance) „Zahrajeme si kvíz?“. Vybraný záměr je *ask_to_play_game*. Jako entitu *game* jsem v otázce vybral slovo „kvíz“. Na základě dříve

dodaných trénovacích vět dává rozhraní najevo, že toto slovo jako entitu *game* již identifikuje, dokonce se 100% pravděpodobností. Stiskem tlačítka Train and Validate je trénovací fráze zapamatována.



Obrázek 5.3: Rozhraní Wit.ai

Tímto způsobem jsem pro operátorovu aplikaci natrénoval přes 200 testovacích frází. Souhrn záměrů a entit, které tyto testovací fráze obsahují, uvádím v tabulce 5.1. Tyto záměry a entity dále používám především v hlavním dialogovém souboru *introduction.top*.

V následující sekci popíši hlavní dialogový soubor aplikace.

5.5.2 Hlavní dialogový soubor

V hlavním dialogovém souboru *introduction.top* je definována většina konverzačních scénářů mé aplikace. Nutno podotknout, že schopnosti konverzace lze a je žádoucí v průběhu nasazení aplikace vhodné dále rozšiřovat a současný dialog není zdaleka tak bohatý, jak by mohl být. V této sekci uvedu několik zajímavých součástí tohoto souboru.

Informování o událostech
K informování o pořádaných událostech je zapotřebí několik programových součástí, především pak třída EventManager. Ta zajišťuje načítání dat budoucích akcí z databáze operátora prostřednictvím HTTP API. Po úvodním načtení akcí metodou *fetch_events* lze vyhledávat akce v určený den, týden, nebo měsíc (metody *get_events_on_date*, *get_events_for_week* a *get_events_for_month*). Zároveň poskytuje metodu *get_top_events*, ta vrací náhodnou „TOP“ událost, tedy jednu z těch, na které by měl robot zvát především.

Přístup k těmto funkcím zprostředkovává modul PMChatFunctions. Pro získání náhodné „TOP“ události lze využít metodu *get_random_top_event*. Sofistikovanější metoda *get_events_for_date*, která jako jediný parametr přijímá

záměr	příklad	entity
ask_about_abilities	Umíš tančit?	type_of_ability
ask_about_developer	Kdo tě naprogramoval?	
ask_about_events	Co se tu bude dít v květnu?	date, type_of_event
ask_about_mood	Jak se vede?	
ask_about_weather	Jak bude ve čtvrtek?	date, weather
ask_for_date	Kolikáteho bude zítra?	date, joke
ask_for_discount	Chci slevu na tartif!	type_of_discount
ask_for_joke	Řekni mi vtip.	
ask_for_time	Kolik je?	
ask_to_dance	Zatanči	
ask_to_play_game	Zahrajem si člobrdo?	game
ask_about_surroundings	Co vidíš kolem sebe?	
ask_to_take_photo	Vyfotíme se?	
say_goodbye	Měj se!	
tell_agree	Jo, chci	
tell_disagree	Ne, to mě nezajímá	
tell_mood_bad	Hrozně	
tell_mood_good	Úžasně	

Tabulka 5.1: Souhrn záměrů a entit

datum v „lidském formátu“, vrací tři události v očekávaném časovém rozsahu (např. za týden, v květnu). Získání data a časového intervalu (den, týden, měsíc) používá tato funkce mnou do češtiny lokalizovanou knihovnu *parsedatetime* (viz 5.5.6). Informace o událostech předává s pomocí další metody *_save_events_to_variables* skrze proměnné ALMemory s prefixem *event_*.

Sdělování data a času

Sdělování data a času opět jako prostředníka využívá modul PMChatFunctions, konkrétně jeho metody *get_humanized_time* a *get_humanized_date*. Ty vrací řetězec s lidsky formátovaným časem respektive datem. Implementovány jsou v souborech *time_humanizer.py* a *date_humanizer.py*.

Ukázka volání funkcí z konverzačního souboru:

```
### ASK FOR TIME ###
u:($e_intent == "ask_for_time")
  ^call(PMChatFunctions.get_humanized_time())
    c1:(_*) $1 .  ~show_random_ability_hints
### ASK FOR DATE ###
u:($e_intent == "ask_for_date" ^exist(e_date))
  ^call(PMChatFunctions.get_humanized_date($e_date))
    c1:(_*) $1 .  ~show_random_ability_hints
```

```
u:($e_intent == "ask_for_date")
  ^call(PMChatFunctions.get_humanized_date_today())
    c1:(_*) $1 .  ~show_random_ability_hints
```

Pravidlo je aktivováno, pokud je detekován správný záměr. Následně je zavolána příslušná funkce a její výsledek zachytí pravidlo `c1:(_*)`. Zachycený výstup funkce je robotem vysloven a na tablet jsou zobrazeny náhodné rychlé odpovědi (`~show_random_ability_hints`).

Konverzační zóny
Na základě lokalizace v prostoru může robot ve své konverzaci s člověkem vynechat vystavené technologie, které jsou od něj příliš vzdálené. Pomocí metody *_update_active_areas* jsou tyto „konverzační zóny“ aktualizovány při každém obdržení události *PMLocalization/PositionChanged* modulem PMChatFunctions. Definovány jsou v souboru *conversation_areas.yaml* (cestu k němu určuje CONV_AREA_CONFIG_FILE_PATH) pomocí souřadnice a poloměru, který určuje velikost zóny.

Informace, jestli robot právě stojí uvnitř zóny, se dialogovému systému předá aktivací nebo deaktivací tagu[6] s prefixem *area_*. Pravidla s deaktivovanými tagy robot nikdy neaktivuje, tedy nevysloví. Ukázka pravidel s tagem *%area_3D_tiskarny*. Pravidlo *proposal* slouží k vyslovení „pobídky“, tedy není aktivováno na základě hlasového vstupu, ale nezávisle na něm.

```
proposal: %area_3D_tiskarny
    Všiml sis našich 3D tiskáren? Máme jich tu spoustu,
    a~dokonce pořádáme i~workshopy
    na kterých se s~nimi můžeš naučit pracovat!

u:($e_intent == "ask_about_surroundings" %area_3D_tiskarny)
    Stojíme u~3D tiskáren.
    Věděl jsi, že tiskneme z~bioplastu PLA?
    Vyrábí se z~kukuřice!
```

Dále již k jednotlivým modulům konverzačního subsystému.

5.5.3 PMListener

Modul PMListener má na starost rozpoznávání řeči. Jeho veřejné rozhraní zahrnuje jen pět metod: *start_listening*, *start_listening_after*, *stop_listening*, *calibrate* a *get_detection_threshold*. První tři se týkají (zpožděného) spouštění a ukončování naslouchání, poslední dvě kalibrace.

[6] `http://doc.aldebaran.com/2-5/naoqi/interaction/dialog/dialog-syntax_full.html#tag`

Implementace modulu se podobá návrhovému vzoru Strategy, dvěma strategiemi jsou právě naslouchání a kalibrace (případně žádná z nich). Základem funkce modulu je metoda *processRemote*, skrze kterou framework periodicky dodává data z bufferů robotových čtyř mikrofonů. Po spuštění naslouchání je analyzována hladina hluku jednoho z nich a pokud je překročen přednastavený práh, započne fáze nahrávání.

Na začátku fáze nahrávání je vytvořen buffer pro zvuková data, který je inicializován daty z posledních několika vteřin naslouchání. To se děje díky tzv. prebufferu. Ten je implementován jako fronta s omezenou kapacitou, která zároveň určuje jeho časovou délku. Tu lze konfigurovat díky konstantě AUDIO_PREBUFFER_SIZE. Následně je zahájeno streamování do služby Google Cloud Speech-to-Text, díky knihovně google.cloud.speech.

Délku nahrávání dále ovlivňuje několik konstant. Pokud od posledního překročení prahu hlasitosti uběhne delší doba, než IDLE_RELEASE_TIME a zároveň MIN_RECORDING_DURATION, nahrávání je ukončeno. Přitom nesmí být překročena doba MAX_RECORDING_DURATION.

Streamování do cloudové služby probíhá v samostatném vlákně. V jeho průběhu přichází mezivýsledky zpracování, ty jsou událostí *PMListener/SpeechPreRecognized* předávány modulům, které tuto žádost odebírají. Poté co dojde k ukončení nahrávání, streamování a obdržení konečného výsledku je tento opět předán událostí, tentokrát *PMListener/SpeechRecognized*. Proces lze kdykoliv ukončit zavoláním *stop_listening*. V takovém případě je po řádném ukončení vyvolána událost *PMListener/StoppedListening*, případně, pokud bylo již zahájeno nahrávání, i událost *PMListener/StoppedRecording*.

Kalibraci hladiny, po jejímž překročení je spuštěno nahrávání, lze spustit metodou *calibrate*. Ta opět využije dat jednoho z mikrofonů, aby stanovila základní hladinu hluku. Vyhodnocování probíhá pomocí výpočtu efektivní hodnoty. Výsledný práh je získán vynásobením koeficientem CALIBRATION_COEFFICIENT. Délku lze ovlivnit konstantou CALIBRATION_DURATION. Aby nebylo zapotřebí při každém spuštění systém kalibrovat, lze výchozí hodnotu prahu uložit do konstanty DEFAULT_DETECTION_THRESHOLD.

5.5.4 PMChat

Modulem, který vede konverzaci, tedy má v režii spouštění naslouchání, zpracování transkripce a formulaci odpovědi je PMChat. Jak jsem již zmínil, jeho základem je modul ALDialog.

Po spuštění načte modul obsah dialogových souborů pomocí metody *_load_topics* ze složky určené konstantou CHAT_TOPICS_FOLDER_PATH a modulu ALDialog ho předá pomocí jeho metody *loadTopicContent*. Zároveň začne odebírat události *PMListener/SpeechRecognized* a *PMChat/QuickReply*.

Rovněž je nestandardním způsobem inicializován (není volána metoda *subscribe*) modul ALDialog a zavolána jeho metoda *stopPush*. Pro vysvětlení: jelikož jsou funkce modulu ALDialog vázány na AL moduly pro rozpoznávání

a syntézu řeči, nelze ho využít přímo – po jeho spuštění by ihned začal aktivně konverzovat. Naštěstí lze dialogový systém od těchto modulů oddělit pomocí metody *force_input*, skrze kterou lze „vnutit“ vlastní vstup a provozovat ho pasivně.

Dále již nezbývá než vyčkat na zavolání metody *start_session* nebo *start_session_with_utterance* při přechodu do životního stavu HumanContact. Pokud je zavolána metoda *start_session*, je započato naslouchání pomocí metody *start_listening* modulu PMListener. V případě předání pozdravu, kterým člověk robota pozdravil, je tento vstup rovnou předán metodě *_force_input* (pozor, metoda modulu PMChat). V opačném případě se čeká na událost *PMListener/SpeechRecognized*, skrze kterou se na vstup dialogového systému (opět zmíněnou metodu *_force_input*) dostane transkripce hlasového vstupu.

Metoda *_force_input* má na starost předání vstupu službě pro zpracování přirozeného jazyka, nastavení příslušných ALMemory proměnných na základě výsledku a předání vstupu metodou *force_input* modulu ALDialog.

Pokud není nastavena ALMemory proměnná *skip_intent_detection*, zpracování přirozeného jazyka provede metoda *_fetch_intent_and_entities*. Ta na pozadí použije pomocný skript *utterance_functions.py*. Jeho úlohou je připojit se k API služby Wit.ai a zpět předat strukturovaný výsledek. Následně je získaný záměr spolu s entitami uložen do proměnných ALMemory pomocí metody *_generate_variables*. Tyto proměnné následně používají dialogové soubory.

Nyní již nic nebrání předání vstupu modulu ALDialog. Ten prohledá pravidla ve všech načtených dialogových souborech a příslušně člověku odpoví. Poté, co robot dokončí svou odpověď, přijde na řadu metoda *_on_output_done*. Ta zkontroluje hodnotu proměnné *switch_to_topic* a případně změní aktivní konverzační soubor. Jinak, pokud mezitím nebyl dialog ukončen metodou *stop_session*, pouze znovu spustí naslouchání modulem PMListener.

Kvízy

Kvízy jsou rovněž implementovány pomocí dialogových souborů. Aby jejich obsah mohl tvořit kdokoliv bez znalosti syntaxe QiChat, jejich vytváření je však automatizováno – generují se z konfiguračních souborů ve formátu YAML.

Ve chvíli, kdy metoda *_load_topics* narazí na šablonu *quiz_template.top*, deleguje její obsah metodě *_load_quizzes*. Ta s pomocí třídy QuizHelper doplní pro všechny kvízy s konfiguračními soubory ve složce QUIZ_CONFIG_FOLDER_PATH (*config/quizes*) do šablony jejich otázky a odpovědi. Jelikož jsou kvízy vygenerované jako samostatné dialogové soubory, lze je aktivovat nastavením názvu kvízu do již zmíněné ALMemory proměnné *switch_to_topic*.

Ukončení konverzace

Jak uvádím v sekci Samostatný život, pokud je robot ve stavu HumanContact, je periodicky volána metoda *heartbeat* modulu PMChat. Ta kontroluje, zda

od poslední robotovy odpovědi nenásledované hlasovým vstupem od uživatele uběhl čas definovaný konstantou DIALOG_TIMEOUT_SECONDS. V případě, že uběhl, vrátí *False*, jinak *True*. Pro úplnost – timestamp poslední výměny si PMChat uchovává v proměnné *_last_answer_timestamp*.

Další funkcí této metody je však kontrola ještě jednoho časového intervalu, tím je PROPOSAL_TIMEOUT_SECONDS. Po uplynutí této doby, která musí být kratší než celkový timeout, je zavolána metoda modulu ALDialog *force_output*. Ta donutí robota vyslovit jedno z jeho *proposal* pravidel definovaných v dialogovém souboru tj. pobídek k další konverzaci. Příkladem může být třeba zajímavost o některé z vystavených technologií: „Věděl jsi, že tu máme i hydroponický stojan a pěstujeme si v něm vlastní bylinky?“

5.5.5 PMChatFunctions

Pomocný modul PMChatFunctions zprostředkovává některé funkce pro volání z dialogových souborů. Oddělení od modulu PMChat je čistě z důvodu větší přehlednosti. Modul využívá především hlavní dialogový soubor pro své funkce týkající se sdělování všech informací, které je potřeba získávat dynamicky – datum, čas, ale i informace o událostech a poloze robota v „konverzačních zónách“.

5.5.6 Zpracování data v lidském formátu

Modul PMChatFunctions pro své fungování spoléhá na zpracování časových údajů z přepisu hlasového vstupu. Je potřeba, aby knihovna z data dokázala identifikovat interval – jak jediný den (prvního srpna), tak rozmezí týdnů nebo měsíců (příští týden, v květnu, za tři měsíce).

Pro tyto účely jsem nejprve testoval knihovnu *dateparser*. Ta v nejnovější verzi již Python 2.x nepodporuje, dostupné jsou však starší verze. Knihovna nabízí částečnou podporu češtiny, zpracování vstupu však trvá z neznámého důvodu přes dvě sekundy. Knihovna má problém vyhodnotit výrazy jako „pozítří“ a „předevčírem“ a neporadí si ani s některými dalšími vstupy.

Pro svou aplikaci jsem nakonec vybral knihovnu *parsedatetime*. Ta podporu češtiny sice postrádá, nicméně výrazy vyhodnocuje rychle a spolehlivě. Vytvořil jsem proto konfigurační soubor s konstantami pro český jazyk a knihovnu jsem naučil česká data v lidském formátu zpracovávat. Tato knihovna navíc dokáže vyhodnotit i časové intervaly – výsledkem zpracování je počáteční datum a informace, zda jde o interval jeden den, týden, měsíc nebo rok.

5.6 Neverbální komunikace

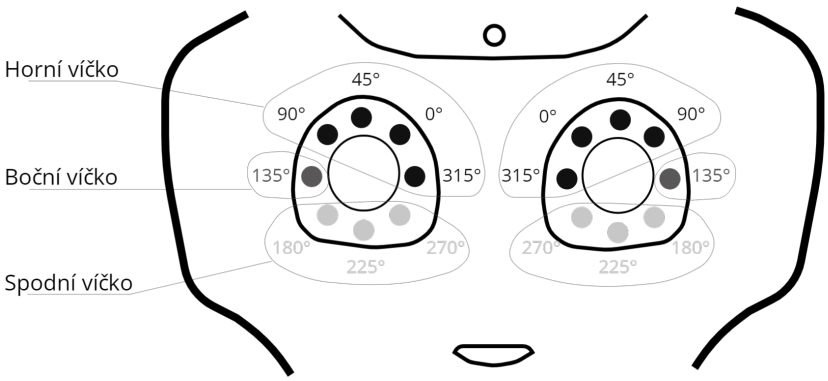
Hlavním prostředkem robotovy neverbální komunikace je jeho tablet, dalším prostředkem jsou LED diody umístěné na ramenou a v očích. Subsystém neverbální komunikace se proto skládá ze dvou modulů: PMNonVerbal a PMLeds a tabletové aplikaci se věnuje samostatná podsekce 5.6.3.

5.6.1 PMLeds

Úkolem modulu PMLeds je napravit nedostatky modulu ALLeds a poskytnout lepší rozhraní pro ovládání robotových LED diod. Modul PMLeds pro své fungování částečně používá ALLeds, ale především využívá přímého přístupu k hardwaru skrze modul DCM. Kvůli deaktivování ALAutonomousLife není možné spustit modul, díky kterému robot mrká, součástí PMLeds je tedy i nekonečná mrkací smyčka. Další rozhraní, které modul nabízí je zbarvení bočních a spodních víček a LED diod na ramenou.

Mrkání je implementováno pomocí samostatného vlákna vytvořeného pomocí funkce *qi.async*. Spuštěno je pomocí metody *start_blinking*, ukončeno pomocí *stop_blinking*. V tomto vláknu je s proměnlivou periodou volána privátní metoda *_blink_once*, která komunikuje skrze modul DCM přímo s hardware robota a nastavuje animaci mrkání.

Pro snadnější práci s osmi LED diodami, které lemují robotovy oči, jsem je rozdělil do tří skupin. Ty jsem definoval jako aliasy, na které se mohu odkazovat při práci s modulem DCM a nemusím tak diody ovládat samostatně. Rozdělení na horní, spodní a boční části víček znázorňuji na obrázku 5.4. Tyto aliasy při volání metody *start* inicializuje metoda *_init_dcm_aliases*.



Obrázek 5.4: Rozdělení LED diod ve víčkách do tří skupin

Pro další ovládání barvy víček zpřístupňuji metody *set_side_eyelid*, *set_lower_eyelid* a *animate_side_eyelid*. Jako tři parametry *r*, *g* a *b* přijímají kód barvy k okamžitém nastavení nebo animaci prolnutí.

Posledními veřejnými funkcemi jsou *restore_set_values* a *restore_chest_leds*. Ty resetují barvy LED diod v očích a na ramenou na jejich poslední nastavené barvy (barva ramenou je konstantně magentová). Hodí se především, když jiná součást systému (notifikace, animace, modul rozpoznávání řeči ALSpeechToText) ovlivní barvu ovládaných LED diod.

5.6.2 PMNonVerbal

Modul PMNonVerbal zastřešuje neverbální komunikaci jako celek. Odebírá události ostatních modulů a komunikuje s modulem PMLeds a s tabletem, potažmo jeho aplikací, prostřednictvím modulu ALTabletService. Dále má na starosti rozdělování textu, který robot vyslovuje, na tak malé části, aby je bylo možné zobrazit na tabletu písmem o čitelné velikosti. Skrze tento modul rovněž PMChat zobrazuje na tabletu návrhy „rychlé odpovědi“ a QR kódy.

Přehled událostí, které modul odebírá a akce, které na základě nich provádí znázorňuje tabulka 5.2.

událost	akce
PMListener/SpeechRecognized	zobrazí transkripci v tab. apl.
PMListener/SpeechPreRecognized	zobrazí dílčí transkripci v tab. apl.
PMListener/StartedListening	spustí animaci naslouchání
PMListener/StoppedListening	skryje zrovna zobrazenou animaci
PMListener/StartedRecording	spustí animaci nahrávání
PMListener/StoppedRecording	spustí animaci zpracovávání
ALTextToSpeech/CurrentSentence	zobrazí začátek vysl. textu
ALTextToSpeech/CurrentWord	zobrazí další část vysl. textu
PMChat/QuickReply	zobrazí text zvolené rychlé odp.
PMChat/SessionStopped	obnoví vzhled tabletové aplikace
notificationRemoved	resetuje barvu LED na ramenou

Tabulka 5.2: Události, na které reaguje modul PMNonVerbal

Zobrazování textu vyslovovaného robotem na tabletu
Robotem vyslovovaný text získává modul prostřednictvím události *ALTextToSpeech/CurrentSentence*. Název události může vzbuzovat dojem, že obsahem bude jediná věta, ve skutečnosti jde však o jakkoli dlouhé souvětí, které bylo dialogovým modulem předáno modulu syntézy řeči. Pro zobrazení na tabletu je tedy potřeba tento text rozdělit, aby se zobrazoval po kratších částech, které bude možné zobrazit písmem o čitelné velikosti.

Rozdělování a zobrazování částí zajišťují tři metody. Celý vyslovovaný text přebírá zmíněnou událostí metoda *show_robot_prompt*. Ta pomocí regulárních výrazů opraví typografické faux-pas, které způsobí dialogový systém. Jde především o odstranění nadbytečných mezer okolo interpunkčních znaků

(.,?!:). Dále volá metodu *_split_into_sentences*, která rozdělí text podle interpunkce a minimální délky na části. Následně jsou tyto části uloženy do fronty *_next_sentences* a jejich obsah je ještě zpětně přeložen díky pomocné třídě PronunciationHelper (viz 5.3.1). To znamená, že jsou slova z jejich fonetického tvaru převedena zpět do původního tvaru.

Samotné zobrazení zajišťuje až metoda *advance_robot_prompt*. Tu vyvolá událost *ALTextToSpeech/CurrentWord*. Přestože by se opět mohlo zdát, že je vyhráno a robot tuto událost spustí při vyslovování každého slova, realita je jiná. Událost nerespektuje kadenci robotovy mluvy, a její rychlost nesprávně kolísá. Navíc je zpravidla událost vyvolána alespoň dvě sekundy před skutečným vyslovením slova. S tím má funkce počítá a sleduje, zda je právě vyslovované slovo začátkem následující části věty ve frontě *_next_sentences*. Pokud je tomu tak, se zpožděním právě dvě sekundy je zavolána (díky funkci *executeJS* modulu ALTabletService) javascriptová funkce tabletové aplikace, která zobrazí vyslovovaný text.

Zobrazování animací a transkripce hlasového vstupu
Animace jsou v tabletové aplikaci zobrazovány voláním příslušných javascriptových funkcí. Ty jsou v reakci na události modulu PMListener volány pomocí funkce *executeJS* modulu ALTabletService. Celé rozhraní javascriptové části tabletové aplikace je popsáno v následující podsekci 5.6.3.

Rychlé odpovědi, konec dialogu a notifikace
Modul reaguje i na zbylé události modulu PMChat. Událostí *PMChat/QuickReply* předává dialogový modul středníkem oddělené rychlé odpovědi k zobrazení na tabletu. Další událostí *PMChat/SessionStopped* pak dává najevo, že skončil dialog s uživatelem, v reakci na to jsou opět pomocí volání javascriptových funkcí skryty dříve zobrazené texty a animace.

Poslední úlohou modulu je odebírání události *notificationRemoved* modulu *ALNotificationManager*. Ta je vyvolána při odstranění notifikace (která se projevuje blikáním LED na ramenou). Pokud je tedy odstraněna poslední notifikace, je barva diod resetována do výchozího nastavení.

Zbylé veřejné metody
Metoda *show_qr_code_link* zobrazuje na tabletu QR kód, metody *show_intro_prompt* a *show_out_of_order_prompt* zobrazují uvítací text, resp. zprávu, že je robot mimo provoz. Veškerou funkcionalitu zajišťuje sama tabletová aplikace.

5.6.3 Tabletová aplikace

Tabletová aplikace je webová stránka využívající technologií HTML, CSS a skriptovacího jazyka Javascript. Spojení s frameworkem je zajištěno pomocí javascriptové knihovny qi a pro další účely jsou zahrnuty knihovny FastClick[7], qrcode[8] a jQuery. Javascriptová součást aplikace musí být kompatibilní s extrémně zastaralým prohlížečem robotova tabletu, proto je psána velmi jednoduše a bez vymožeností moderního Javascriptu (v souladu s bodem (h) v sekci 3.1.5).

Nejprve k vzhledu a struktuře stránky. Jak jsem uvedl v návrhu, stránka je vizuálně rozdělena do dvou částí, primární vrchní části zobrazující vyslovovaný text a spodní části s šedou lištou, která zobrazuje transkripci textu a animace. Vzhled pomáhá formovat CSS knihovna Bootstrap.

Jediným souborem celé aplikace je *index.html*. Jeho pozadí a grafické rozvržení prvků je statické a veškerá úprava obsahu probíhá pomocí volání javascriptových funkcí uvedených ve zbytku této sekce.

Animace
Animace (implementované v kaskádových stylech) jsem na základě MIT licence převzal z webu codepen.io[9] a následně je upravil pro své potřeby. Animace napodobují ty od hlasového asistenta společnosti Google. V následující tabulce uvádím přehled animací a funkcí, které lze pro jejich zobrazení zavolat. Skyrtí animací zajišťuje funkce *hide_animation*.

animace	funkce
naslouchání	show_listening_animation
nahrávání	show_recording_animation
zpracování	show_processing_animation
zmizení	show_disappear_animation

Tabulka 5.3: Animace a funkce pro jejich spuštění

Automatická velikost textu
Pro zvolení vhodné velikosti textu tak, aby se vešel na šířku a výšku do vyhrazených částí stránky, jsem vytvořil vlastní jednoduchý algoritmus. Reprezentován je funkcí *find_suitable_size* a pro výpočet skutečné pixelové velikosti používá funkci *measureText* HTML prvku canvas[10]. Algoritmus používají funkce *show_human_utterance* a *show_robot_prompt*, které zobrazují přepis hlasového vstupu, respektive vyslovovaný text. Parametry jsou *sentence*, tj. text, který se má zobrazit, minimální a maximální velikost písma, *step* – krok, po

[7] `https://github.com/ftlabs/fastclick`
[8] `http://davidshimjs.github.io/qrcodejs/`
[9] `https://codepen.io/simoberny/pen/ZbYxvq`, tímto děkuji autorovi Simonu Bernabè
[10] `https://www.w3schools.com/tags/canvas_measuretext.asp`

kterém se mají jednotlivé velikosti zkoušet a výška a šířka elementu, který bude text zobrazovat.

```
function find_suitable_size(
  sentence, min, max, step, parent_width, parent_height
) {
    size = max;
    while(size >= min) {
        last_height = get_text_height(
            sentence, size, parent_width
        );
        if(last_height < parent_height * 0.9) {
            return size;
        }
        size -= step;
    }
    return size;
}
```

QR kódy
Pro usnadnění komunikace s uživatelem předávám odkazy na webové stránky skrze QR kódy. Ty zobrazuji pomocí knihovny qrcodejs. Popisek a url adresu stránky jsou tabletové aplikaci předány pomocí funkce *show_qr_code_link*. Využití v implementované aplikaci nalezly QR kódy především pro předávání odkazů na konkrétní události, nebo na webovou stránku s jejich kalendářem (viz obr 5.5).

Rychlé odpovědi
Jediným důvodem zahrnutí knihovny qi do tabletové aplikace je implementace rychlých odpovědí. Ty jsou zobrazeny po zavolání funkce *show_answer_suggestions*. Vstupem jsou středníkem oddělené hodnoty, výsledkem jsou vygenerovaná tlačítka zobrazená na vrchu lišty s transkripcí. Právě po stisku některého z těchto tlačítek je vyvolána událost PMChat/QuickReply, na kterou reaguje modul PMChat. Představu o vzhledu aplikace doplní obrázek 5.6.

Další funkce
K popisu zbývají dvě důležité funkce. Pro zobrazení úvodního pozdravu a instrukce k oslovení robota nebo doteku na tablet slouží funkce *show_intro_prompt*. Naproti tomu funkce *show_out_of_order_prompt* zobrazuje velkými písmeny na černém pozadí nápis „MIMO PROVOZ“. Obě funkce souvisí s modulem samostatného života, volány jsou při jeho aktivaci a deaktivaci.

FastClick
Vzhledem k zastarání webového prohlížeče vyvstává další problém, kterému se

Obrázek 5.5: Ukázka zobrazení QR kódu v tabletové aplikaci

Obrázek 5.6: Ukázka tabletové aplikace

nevyhnula ani stávající aplikace operátora. Prohlížeče v mobilních zařízeních před rokem 2015 vkládaly mezi stisk tlačítka a event *onclick* 300ms prodlevu. Důvodem vyčkávání bylo ověření, zda se uživatel nesnaží o dvojklik. Tuto prodlevu ve své aplikaci eliminuji právě díky knihovně FastClick.

5.7 Lokalizační subsystém

Lokalizační subsystém řídicí aplikace se sestává z modulů PMObjectDetection a PMLocalization. Před jejich popisem opět upřesním, jak přesně lokalizace funguje a jak probíhal vývoj této části aplikace.

5.7.1 Implementační detaily a postup

Jak jsem uvedl v kapitole Návrh, můj způsob lokalizace spoléhá na rozpoznávání objektů ze snímků kamer, odhadu jejich relativní polohy vůči robotovi a předání těchto dat algoritmu particle filter localization.

Rozpoznávání objektů
Rozpoznávání objektů jsem se rozhodl po vzoru kolegů realizovat pomocí knihovny TensorFlow a její Object Detection API. Původní myšlenka nasazení na robotovi bohužel musela padnout vzhledem k extrémnímu zatížení procesoru při rozpoznávání objektů. Doba rozpoznávání v jediném snímku totiž při experimentu průměrně přesahovala tři sekundy oproti několika desítkám milisekund na středně výkonném PC. Proto je tato úloha přesunuta na externí server.

Sběr dat pro učení
Nejprve jsem identifikoval objekty, které by robot mohl rozpoznávat. Bylo zapotřebí se soustředit na objekty, které nelze přemísťovat, nebo se v nejbližší době neočekává jejich odstranění. Jedná se tedy především o světelné nápisy a vestavěné spotřebiče. Kompletní seznam 22 objektů uvádím v tabulce 5.4 a zároveň je zobrazuji v mapce prodejny 5.7.

Následně bylo potřeba získat velký vzorek obrazových dat, který by objekty zachycoval v různých úhlech a vzdálenostech. Prvním pokusem bylo použití mobilního telefonu k natočení videa, ze kterého jsem následně vybral snímky pro učení. Po označení přibližně 500 snímků programem labelimg jsem s pomocí návodu[11] natrénoval model (pomocí tzv. *fine-tuning* jsem přetrénoval model *ssd_mobilenet_v2_coco*) a zmrazil ho pro použití při rozpoznávání. Bohužel, setkal jsem se s velkým neúspěchem – robot poznal jen velmi málo objektů a to navíc jen z velmi mála pozorovacích úhlů a vzdáleností.

Pro další vzorkování (zachycení videa) jsem použil samotného robota Pepper. Vytvořil jsem si skript pro ovládání robotova skrze ovladač ke konzoli Xbox a prodejnu s robotem projel, abych zachytil objekty z robotova úhlu pohledu. Po označení objektů na více než 2200 snímcích a natrénování modelu jsem dosáhl většího úspěchu. Přestože robot nedokáže rozpoznat všechny natrénované objekty, dokáže spolehlivě identifikovat světelné nápisy, přestože je občas zaměňuje. Pro lepší výsledky jsem však musel upravit nastavení ka-

[11] `https://tensorflow-object-detection-api-tutorial.readthedocs.io/`

	identifikátor	popis
1	trouba	spotřebič
2	kuch_linka	kuchyňská linka
3	pracka	spotřebič
4	love_mag_napis	nápis
5	cow_kitch_napis	světelný nápis
6	rossmann_napis	světelný nápis
7	cow_cafe_striska	stříška
8	mag_home_striska	stříška
9	mag_home_napis	stříška
10	mag_garden	samolepka na podlaze
11	win_the_golden_shoe	nápis
12	podcast_napis	světelný nápis
13	gaming_napis	světelný nápis
14	hydroponie	hydroponický stojan
15	zelena_praha	nápis
16	kamery_love	kamery
17	tuning_stage_napis	světelný nápis
18	hasici_pristroj	hasicí přístroj
19	beat_the_lap_record	nápis
20	cow_cafe_napis	světelný nápis
21	lednice	spotřebič
22	digestor	spotřebič

Tabulka 5.4: Seznam rozpoznávaných objektů

mery tak, aby se světelné nápisy nejevily rozmazaně. Tím však klesla celková světelnost snímku a proto není možné rozpoznat některé neosvětlené nápisy.

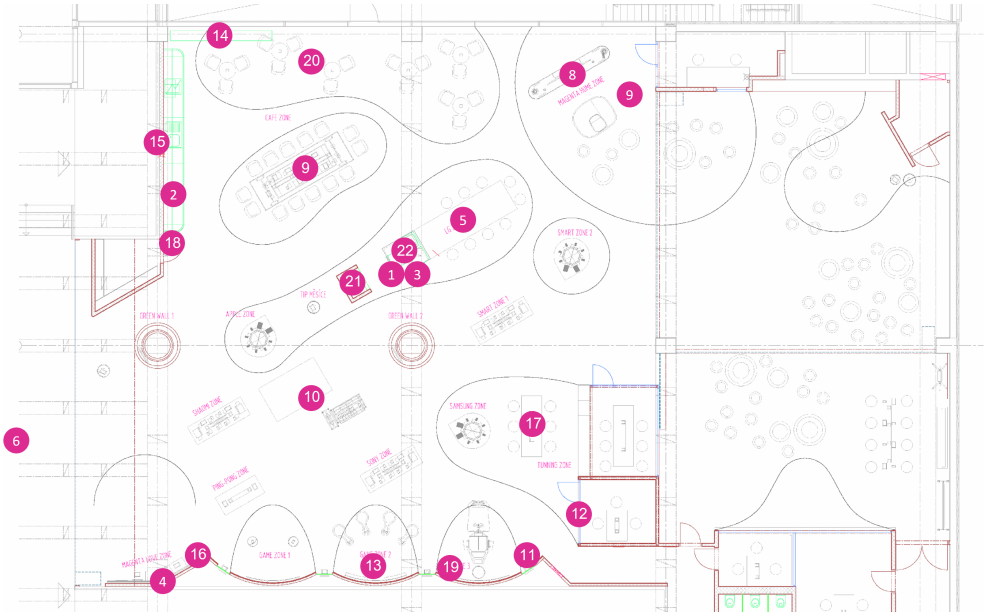
Zároveň, vzhledem k tomu že do prodejny vniká světlo zvenčí, kvalitu rozpoznávání ovlivňuje i denní doba. Jelikož jsem video s robotem natáčel během večera, objekty blízko prosklené výlohy rozpoznává přes den stěží. Bohužel jsem s tímto nepočítal a nepřidal do datasetu snímky ze dne, domnívám se však, že by se díky takové úpravě kvalita rozpoznávání opět zvýšila.

Konečný seznam objektů, které robot dokáže rozeznat: *veškeré světelné nápisy, pračka, Magenta Garden, hydroponický stojan.*

Odhad vzdálenosti

Odhad vzdálenosti od objektů je založen na jejich stanovených souřadnicích a reálné šířce. Tato data jsou uložena v konfiguračním souboru. Nejedná se však o úplně přesnou metodu – objekt mění svou zdánlivou šířku dle úhlu, pod kterým ho pozorujeme. Jako další alternativa pro stanovení vzdálenosti by mohla posloužit data z 3D senzoru, ve své implementaci jsem se touto cestou z časových důvodů již nevydal.

Dál již k jednotlivým modulům subsystému.



Obrázek 5.7: Rozpoznávané objekty v mapce

5.7.2 PMObjectDetection

K rozpoznávání objektů a výpočtu jejich vzdálenosti od robota slouží modul PMObjectDetection. Po svém spuštění začne odebírat data z horní kamery na robotově hlavě a v samostatném vlákně spustí svou hlavní smyčku. V této smyčce nejprve proběhne připojení k serveru pro rozpoznávání objektů a následně jsou v cyklu odesílány snímky z kamery. V návaznosti na jednotlivé snímky odesílá server odpovědi s daty o detekcích v jednoduchém formátu – jde o pole detekcí, jeho prvky jsou opět pole s hodnotami reprezentujícími identifikátor objektu a jeho čtyři souřadnice ve snímku. Dekódování těchto dat provádí metoda *_decode_detection_response*. Relativní souřadnice vůči robotovi zjišťuje na základě známé šířky objektů a transformační matice metoda *_get_relative_coordinates_for_object*. Funkce pro výpočet transformací na základě natočení a sklonu robotovy hlavy poskytuje modul ALMotion (metoda *getTransform*) a knihovna ALMath (funkce *Transform*).

Informace o detekovaných objektech mohou ostatní součásti systému získat veřejnou metodou *get_detected_objects*, případně *get_recent_objects*. První vrací objekty nalezené v posledním analyzovaném snímku, druhá připojuje k objektům údaj o čase jejich posledního spatření. O aktualizaci množiny detekovaných objektů informuje událost *PMObjectDetection/ObjectsDetected*.

Informace o objektech (jejich skutečnou šířku a souřadnice v mapě prodejny) načítá z konfiguračního souboru *config/detection_objects.yaml* (konstanta OBJECT_CONFIG_FILE_PATH) metoda *_load_object_data*. Stav spojení se serverem lze zjistit voláním metody *is_connected*. Pro síťovou komuni-

kaci slouží metody *_send_data* a *_receive_data*.

Pro úplnost dodám, že modul je částečně připraven i pro rozpoznávání orientačních bodů Naomarks a QR kódů. Robot takové rozpoznávání nabízí prostřednictvím modulů ALLandMarkDetection a ALBarcodeReader. Dále jsem se však implementací rozpoznávání nezabýval, jelikož by bylo nutné s vedením konzultovat umístění těchto grafických prvků do prostoru prodejny.

5.7.3 Server pro rozpoznávání objektů

Server pro rozpoznávání objektů je jednoduchý pythonový skript odvozený z části zmíněného návodu k TensorFlow Object Detection API (*Detect Objects Using Your Webcam*). Já jsem pouze doplnil implementaci TCP serveru a komunikaci skrze již zmíněné metody *_send_data* a *_receive_data*.

5.7.4 PMLocalization

Pro odhad pozice robota na základě informací o relativních souřadnicích detekovaných objektů slouží modul PMLocalization. Samotný algoritmus particle filter localization, který je srdcem tohoto modulu, implementuje třída *LandmarkParticleFilter*, modifikovaná z implementace Mikołaje Łysakowskiho[12].

Lokalizační modul po svém spuštění inicializuje filtr a začne odebírat událost *PMObjectDetection/ObjectsDetected*. Zároveň je spuštěna periodická úloha (*qi.PeriodicTask*), která v intervalu tří sekund kontroluje změny v robotově odometrii pomocí metody *_update_odometry*. Ta, pokud došlo k pohybu robota, volá metodu filtru *predict_delta*. Metoda s jistou mírou vložené náhody, která reprezentuje nepřesnost odometrie, všechny částice posune o naměřenou vzdálenost dle jejich úhlu natočení.

V případě, že jsou v obrazu detekovány objekty, reaguje metoda *_objects_detected*. Ta nejprve načte objekty do instanční proměnné *_latest_detected_objects* a následně volá metodu *_update_and_resample*. Ta na particle filter volá metodu *update*, která aktualizuje váhy jednotlivých částic podle přesnosti jejich odhadu vůči novému pozorování. V návaznosti na tuto aktualizaci je ještě vypočítána efektivní hodnota *eff*, tj. počet částic, které pravděpodobně dobře odhadují robotovu pozici. Pokud tento počet klesne pod experimentálně stanovenou hodnotu (čtvrtina počtu částic), je na filtr zavolána metoda *resample*. Ta provede „resampling“ částic, což je proces, který náhodně, přesto podle váhy, vybere částice, které budou zachovány.

Souřadnice robota jsou odhadnuty aritmetickým průměrem souřadnic jednotlivých částic. Vhodný indikátor přesnosti tohoto odhadu by mohl být rozptyl, já však přesnost lokalizace nevyhodnocuji. Přístup k takto odhadnuté poloze (souřadnicím) můžou ostatní moduly získat zavoláním metody *get_position*.

[12] `https://github.com/kolaszko/particle_filter`

5.8 Administrační aplikace

Administrační aplikace je jednoduchá webová stránka přístupná skrze robotovu adresu v místní síti na portu 8000. Využívá identickou javascriptovou knihovnu pro komunikaci s frameworkem NAOqi jako aplikace robotova tabletu. Veškerá funkcionalita je zajištěna voláním funkcí PM a AL modulů frameworku skrze tuto knihovnu. Rozhraní je responzivní a o jeho vzhled se opět stará framework Bootstrap. Tvoří ho dva HTML soubory, *index.html* a *map.html*, mezi nimiž lze přepínat v horním menu aplikace.

Výhodou mé administrace je možnost nastavení jiného uživatelského jména a hesla, než používá uživatelský účet nao. Údaje lze nastavit v konfiguračním souboru řídicí aplikace *config.py*. Zabezpečení je provedeno pomocí „Basic authentication“, tedy díky HTTP kódu 401 Authorization Required, stejně jako u robotovy vlastní webové administrace. Toto zabezpečení obstarává modul PMAdminServer.

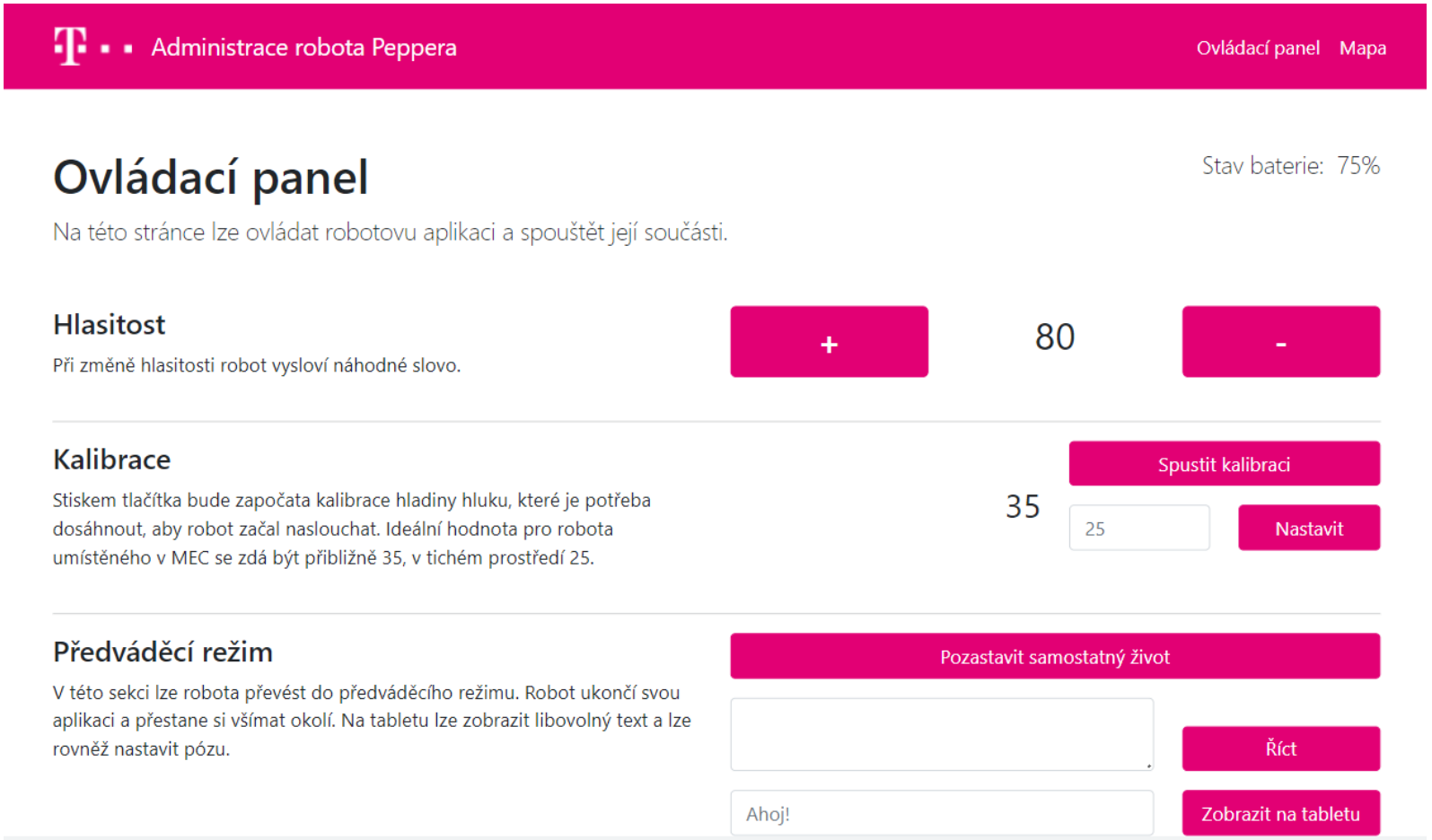
Ovládací panel
Ovládací panel, reprezentovaný souborem *index.html*, slouží k ovládání a konfiguraci parametrů aplikace za jejího běhu. Několik jeho funkcí se překrývá s těmi, které jsou již dostupné v robotově výchozí webové aplikaci. Jde mimo jiné o úpravu hlasitosti, vyslovení libovolného textu a možnost restartování nebo vypnutí robota. Zároveň je v horním pravém rohu rovněž zobrazena procentuální hodnota stavu nabití robotovy baterie.

Vyslovení textu pomocí syntézy řeči je však obohaceno o překlad psaného tvaru známých slov na fonetický tvar nutný pro jejich správné vyslovení. Pro tento jediný účel existuje modul PMSpeech. Jeho jediná metoda *say* toho dociluje díky již zmíněnému PronunciationHelper. Místo [té mobile] tak robot například umí vyslovit název operátora – [týmobajl].

Dále ovládací panel umožňuje kalibrovat nebo manuálně nastavit hladinu pro započetí nahrávání (viz modul PMListener, jehož metodu *calibrate* nebo *set_detection_threshold* volá).

Důležitou možností, kterou administrace poskytuje, je zastavení a opětovné spuštění modulu samostatného života. Lze tak učinit v sekci Předváděcí režim, na pozadí jsou volány funkce *start* a *stop* modulu PMLife. Mimo vyslovení libovolného textu, lze v této sekci i zobrazit libovolný text na tabletu a nebo pózu prostřednictvím modulu ALAnimationPlayer. V následující sekci lze podobným způsobem aktivovat nebo deaktivovat modul pro lokalizaci.

Při prvotním načtení stránky jsou všechny neznámé údaje nahrazeny animacemi načítání. Až po připojení QiSession jsou tyto údaje přepsány jejich skutečnými hodnotami. Případný neúspěch připojení (indikovaný zavoláním callbacku *disconnected*, nebo zachycením chyby nad připojovací funkcí) v dolní pravé části indikuje vyskakovací lišta. Naopak po úspěšném připojení jsou uvnitř callbacku *connected* navázány reference na veškeré moduly potřebné pro webovou administraci.



Obrázek 5.8: Ukázka administrační aplikace

Mapa
Zobrazení průběhu lokalizace zobrazuje soubor *map.html*. Zde je pomocí javascriptu navázáno po načtení stránky spojení s WebSocket serverem na portu 8005 a pokud je zapnut modul PMLocalization, v průběhu jeho práce jsou klientům zasílána data ve formátu JSON. Ta obsahují pozice jednotlivých particles a odhad pozice robota. Javascriptová funkce *process* zpracovává tyto příchozí zprávy a zakresluje je do zobrazené mapy, která je ve skutečnosti HTML prvkem canvas. Zakreslování probíhá pomocí funkcí prvku canvas[13], particles jsou zobrazeny jako modrá kolečka s jemnou průhledností a odhad pozice je zobrazen jako zelený bod. Umístění podkladového obrázku mapy pobočky a port WebSocket serveru jsou pevně stanoveny v kódu.

Díky spojení s frameworkem je v pravém horním rohu stránky zobrazen a periodicky aktualizován status spojení řídicí aplikace se serverem pro rozpoznávání obrázků. Zároveň, pokud je modul lokalizace vypnutý, lze manuálně nastavit pozici robota v prostoru pomocí dvou vstupních polí pro souřadnici X a Y. V následku tohoto manuálního nastavení jsou aktualizována konverzační témata dle oblastí, do kterých robot na základě své nové pozice spadá (viz 5.5.2).

5.8.1 PMAdminServer

Aby bylo možné webovou administraci zobrazit skrze robotovu IP adresu, vytvořil jsem modul PMAdminServer, který má za úkol při svém spuštění vytvořit na předdefinovaném portu webový server.

V metodě *start* vytváří pomocí funkce frameworku *qi.async* dvě vlákna, jejichž úkolem je inicializace webového resp. websocketového serveru a spuštění nekonečné obslužné smyčky. Vláknům je předán objekt *promise*, jehož metoda *isCancelRequested* je v průběhu obsluhy požadavků ověřována. Tímto způsobem jsou při volání metody *stop* ukončeny procesy serverů.

Webový server
Webový server je realizován pomocí třídy TCPServer dodané balíčkem SocketServer. Pro zpracování požadavků jsem vytvořil třídu HTTPRequestHandler dědící od SimpleHTTPRequestHandler, která implementuje metodu *do_AUTH* pro autentizaci. V následujícím úryvku kódu je demonstrován způsob, jakým je zabezpečení implementováno.

```
def do_AUTH(self):
    self.send_response(401)
    self.send_header(
        'WWW-Authenticate', 'Basic realm="Pepper Admin"'
    )
    self.send_header('Content-type', 'text/html')
```

[13] https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API

```
    self.end_headers()

def do_GET(self):
    credentials =
        HTML_ADMIN_USER + ":" + HTML_ADMIN_PASSWORD
    encoded_credentials = credentials.encode('base64')
    if self.headers.get('Authorization')
        == str('Basic ' + encoded_credentials).rstrip():
        SimpleHTTPRequestHandler.do_GET(self)
    else:
        self.do_AUTH()
        self.wfile.write('Not authenticated.')
```

WebSocket server

Websocketový server dodává knihovna simple_websocket_server. Má pomocná implementace třídy WebSocketServerWithClients zajišťuje, že si server drží reference na svoje klienty a díky metodě *send_message* jim umí hromadně rozeslat zprávu.

Hromadné rozesílání zpráv je využito pro odeslání dat z lokalizačního modulu. PMAdminServer odebírá událost *PMLocalization/PositionUpdated*, která informuje o změnu odhadu polohy robota v prodejně, a na jejím základě odesílá data všem aktuálně připojeným klientům. Pro správné převedení dat z formátu knihovny numpy do formátu JSON používám třídu NumpyEncoder[14]. Ukázka registrace k odebírání události a metody zpracovávající příchozí událost:

```
self._al_memory.subscribeToEvent(
  "PMLocalization/PositionUpdated",
  "PMAdminServer",
  "_position_updated"
)
...
def _position_updated(self, _key, _value, _message):
  data = {
    "particles":
      self._pm_localization.get_particles().astype(int),
    "position":
      self._pm_localization.get_position().astype(int)
  }
  self._ws_server.send_message(
    json.dumps(data, cls=NumpyEncoder)
  )
```

[14] https://pypi.org/project/numpyencoder/

Kapitola 6

Testování

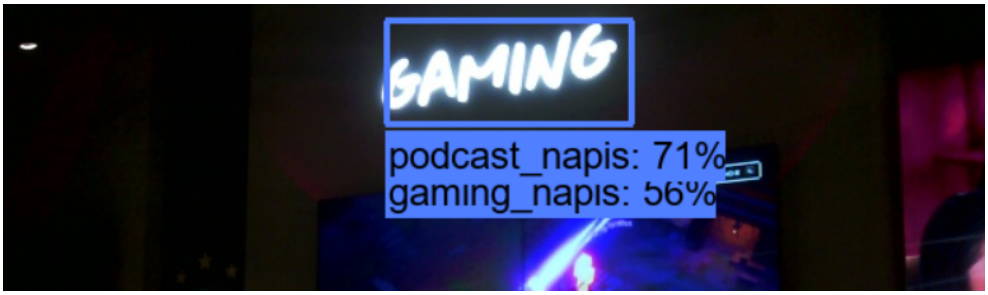
6.1 Lokalizace

Svou implementaci lokalizačního subsystému jsem testoval více způsoby. Robota jsem přitom pokaždé umístil do náhodně vybrané části prodejny a ve webové administraci jsem sledoval průběh jeho lokalizace. Při úvodních pokusech jsem s robotem nepohyboval.

Nejprve jsem testoval pomocí ruční manipulace robotovou hlavou, tedy otáčel jsem jí cíleně tak, aby spatřil a rozeznal naučené objekty. Výsledky byly uspokojivé, bohužel jsem však narážel na problémy se záměnou a dvojitou detekcí při rozpoznávání světelných nápisů (viz obrázky 6.1 a 6.2).

Obrázek 6.1: Záměna světelného nápisu

I přes tyto problémy jsem byl schopen dosáhnout uspokojivých výsledků. Z deseti pokusů se robot úspěšně lokalizoval (s odchylkou menší než 3 metry od jeho skutečné pozice) v sedmi případech. V případech, kdy se robot lokalizoval nesprávně, byla na vině především záměna světelných nápisů Magenta Home a Gaming.



Obrázek 6.2: Dvojitá detekce a záměna světelného nápisu

Následně jsem testoval lokalizaci pomocí robotova samostatného rozhlížení v rámci jeho reflexů, které zajišťuje modul ALBasicAwareness. Při testování se ukázalo, že by přišlo vhod implementovat algoritmus, který se zpočátku bude systematicky, ale přirozeně, rozhlížet, a poté, co objeví objekt, bude se tento snažit dostat do svého záběru celý a ujistit se tak o jeho skutečné vzdálenosti. Současné reflexy, které nutí robota rozhlížet se musí být vyvolány vnějšími vlivy, především zvuky, jejichž původ se snaží robot nalézt.

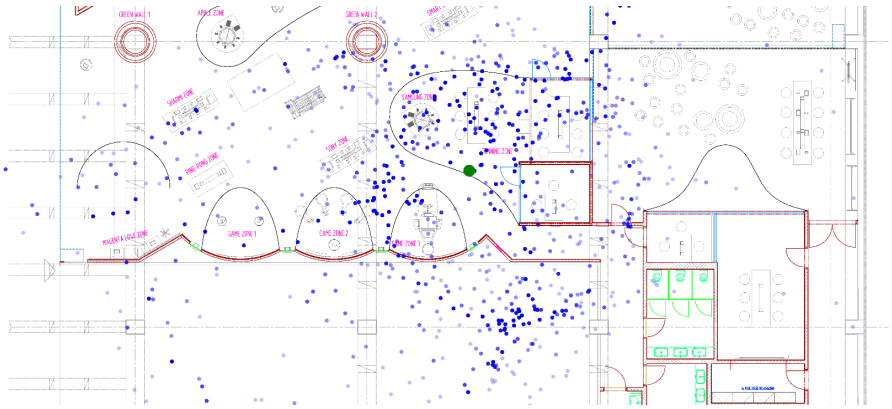
Přestože samostatné rozhlížení průběh lokalizace velmi zpomalilo, na její úspěšnost se zásadně neprojevilo. Z deseti pokusů se robot lokalizoval úspěšně v šesti případech a na vině při nesprávných výsledcích lokalizace byla opět záměna detekovaných objektů, které jsem při těchto experimentech nemohl cíleně zabránit.

typ experimentu	pokusů	úspěch	prům. odchylka při úspěchu
manuální rozhlížení	10	7	1.8 m
samostatné rozhlížení	10	6	2.4 m

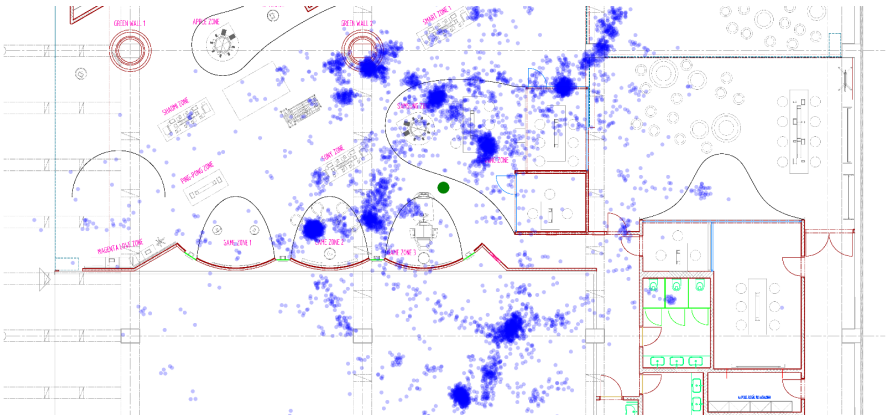
Tabulka 6.1: Výsledky experimentů

Na následujících obrázcích je patrný průběh jednoho z lokalizačních experimentů se samostatným rozhlížením včetně otáčení robotova těla. Grafické podklady jsou snímky obrazovky z webové administrační aplikace. Na prvním snímku 6.3 jsou slabě patrné (mimo jiné) dvě kružnice vzniklé odhadem vzdálenosti a relativní polohy nápisů Gaming a Podcast.V dalším snímku 6.4 jsou částice mírně rozptýleny, to se stalo v důsledku robotova pootočení, v tomto snímku jsou patrné i další kružnice odhadující vzdálenost od dalších detekovaných objektů. Na dalších dvou snímcích 6.5 je znázorněn konečný odhad polohy robota po dalších pozorováních. V tomto případě byla poloha určena s odchylkou menší než jeden metr.

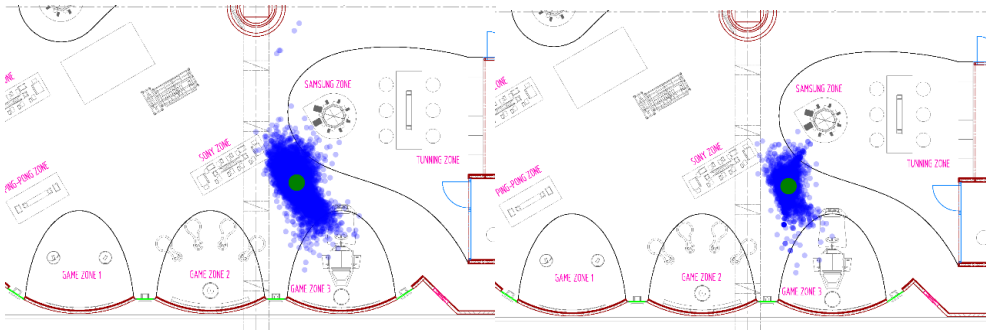
Při testování jsem dále odhalil jeden nedostatek. V důsledku absence vyloučených zón v mapě prodejny se robot lokalizuje i do míst, kde není možné, aby se robot nacházel. Problém ilustruje obrázek 4.4, kde se sice částice dle pozorování shlukují do dvou míst, to nižší je však mimo prodejnu. Řešením by mohlo být začernění mapy v nepřístupných oblastech a přemístění částic, které se na začerněných souřadnicích vyskytnou.



Obrázek 6.3: První krok, rozpoznání nápisů Gaming a Podcast



Obrázek 6.4: Další krok, rozptýlení částic jako důsledek pootočení



Obrázek 6.5: Konečný odhad polohy po dalších pozorováních

6.1.1 Robot se nedokáže pohybovat

Po otestování lokalizace jsem měl v plánu přesunout se k implementaci pohybu po prodejně. Narazil jsem však na zásadní problém – robot se nedokáže pohybovat.

Jelikož by bylo nepřiměřeně náročné implementovat vlastní algoritmus vyhýbání se překážkám, rozhodl jsem se pohyb robota ovládat pomocí modulu ALNavigation. Ten nabízí metodu *navigateTo*, jejímž vstupem jsou relativní souřadnice cíle pohybu. Při zběžném testování jsem však narazil na problém. S největší pravděpodobností v důsledku osvětlení prodejny ostrými bodovými reflektory v kombinaci s mírně lesklou podlahou robot vidí neexistující překážky. To se projevuje buď tím, že se vůbec nerozjede a volání vrátí *False*, případně se rozjede a v průběhu jízdy se prudce zastaví nebo se v otevřeném prostoru opatrně snaží projíždět kolem neexistujících překážek.

Robot toto chování vykazuje kvůli jeho bezpečnostním reflexům. Ty se dají úplně deaktivovat, což pro prostředí veřejně přístupného místa nepřipadá v úvahu. Navíc by i tak bylo nutné implementovat vlastní systém reflexů – minimálně z toho důvodu, aby robot neublížil nárazem do překážky sám sobě.

Nutno dodat, že robot toto chování vykazuje s deaktivovaným modulem ALBasicAwarenes, potažmo ALAutonomousLife, které by do pohybu mohly zasahovat a zároveň s nastavenými bezpečnostními vzdálenostmi od překážek na minimální hodnotu 0,1 m (metody *setOrthogonalSecurityDistance* a *setTangentialSecurityDistance* modulu ALMotion).

Abych ověřil, který senzor způsobuje falešné detekce překážek, pokusil jsem se postupně jednotlivé druhy senzorů zakrýt lepicí páskou (obrázek 6.6). V tu chvíli se však robot automaticky převedl do bezpečné pozice a ohlásil chybovou zprávu, že jeden z jeho kritických senzorů je nefunkční.

Obrázek 6.6: Zakryté laserové senzory, „shovel“ senzor a infračervený senzor

V důsledku tohoto jištění jsem se v návrhu a implementaci omezil, co se týče požadavku na komentovanou prohlídku, na lokalizaci a hovoření o vystavených technologiích. Pohyb po prodejně může být předmětem dalšího rozvoje aplikace, bohužel je však mimo rozsah mé bakalářské práce.

6.2 Průzkum mínění zákazníků

Pro zhodnocení aplikace jsem provedl krátký průzkum mínění mezi náhodně vybranými zákazníky operátora. Cílem průzkumu bylo zhodnotit především konverzační schopnosti robota, tabletovou aplikaci a srozumitelnost aplikace. Respondenti dostali úkol vést s robotem rozhovor na zadané téma a následně ohodnotit robotův výkon a doplnit své postřehy. Zadával jsem tyto úkoly:

- Zjistěte, o čem s robotem můžete hovořit.
- Pohovořte s robotem o své náladě.
- Zjistěte, jaké akce se budou konat.
- Zahrajte si s robotem hru.
- Řekněte si o slevu.

S žádným z úkolů nevyvstal zásadní problém a respondenti je byli schopni zdárně splnit. I přes tento úspěch jsem zaznamenal technické nedostatky, které popisuji v následující sekci.

Robustnost dialogového systému prověřily nepřesné transkripce hlasových vstupů. Když se jeden ze zákazníků ptal na pořádané akce, robot mu rozuměl „Prý se to nějaké akce“. Zpracování přirozeného jazyka však záměr správně vyhodnotilo jako dotaz na pořádané akce a robot tak dokázal vhodně odpovědět. Podobný omyl vznikl, když si další zákazník chtěl s robotem zahrát kvíz. Robot zákazníkovi nesprávně rozuměl „Chtěl bych si zahrát please“. V tomto případě robot zákazníkovi sdělil, že takovou hru nezná a rovnou mu nabídl, že si zahrají kvíz. Mírné nedorozumění pomohla vysvětlit transkripce hlasového vstupu zobrazená na tabletu.

Dalšími mylnými transkripcemi, které byly vyhodnoceny správně, byly „Zajímalo by mě, co se to bude dít za týden“, „Spusť cluse“ a „Chci si zahrát PC“.

Oproti předchozí aplikaci zákazníci vyzdvihovali vyšší informační hodnotu, přestože možná na úkor zábavnosti. Zákazníkům chyběla možnost dále hovořit o vystavených technologiích, to by v budoucnu mohlo pomoci vyřešit připojení dialogového systému do nějaké formy znalostní databáze. Zákazníci dále ocenili tabletovou aplikaci, většina z nich naprosto bez váhání zvládla využít připravené rychlé odpovědi.

6.2.1 Zjištěné nedostatky

Při testování aplikace s prvními respondenty jsem narazil na několik technických nedostatků. V rámci běžné konverzace se robot drží v mezích tří sekund od rozpoznání hlasu až po vyslovení odpovědi, což vyhovuje bodu (f) v sekci 3.1.5.

Problém nastává při prvním dotazu na službu Wit.ai po delší chvíli. Doba čekání na výsledek zpracování při těchto dotazech pohybuje okolo 6 vteřin, což je nežádoucí. Abych předešel čekání na výsledky zpracování při prvním

pozdravu, ale i jindy, kdy zpracování přirozeného jazyka nepotřebuji, vytvořil jsem ALMemory proměnnou *skip_intent_detection*, která, pokud je v ní uložena hodnota *True*, přeskočí krok zjišťování záměru a entit a dialogovému systému předá pouze transkripci hlasového vstupu.

Na rozdíl od autorů aplikace pro robota v roli recepčního jsem nezaznamenal, že by v okolí mikrofonů vznikal hluk od ventilátorů (bod (e) v sekci 3.1.5). Vyvstal však jiný problém specifický k mojí implementaci. Robot v rámci jeho reflexů občas velmi prudce pohne hlavou do maximální polohy a ozve se hlasitý náraz. Pokud se tak stane když robot naslouchá, spustí se streamování do cloudové služby přestože na robota nikdo nehovoří. Tento problém jsem částečně eliminoval nastavením vyššího prahu pro začátek nahrávání, ovšem úplně se mi ho vyřešit nepodařilo.

6.3 Navazující práce

Přestože jsem s výsledkem své práce spokojen, aplikace nabízí prostor pro vylepšení. Konverzační scénáře bude vhodné v průběhu testovacího nasazení rozšiřovat podle nejčastějších dotazů zákazníků shromážděných díky službě Wit.ai. Jak jsem naznačil, dialogový systém by rovněž bylo vhodné připojit do znalostní databáze, ve které by se shromažďovaly informace o vystavených technologiích, ale i další informace například o provozu prodejny.

Po ukončení testovacího režimu systému pro lokalizaci pomocí čipových karet, který je nyní v prostoru nasazen, by bylo možné prozkoumat možnosti jeho propojení s aplikací. Zároveň by bylo vhodné dále pracovat na vylepšení neuronové sítě pro rozpoznávání objektů tak, aby dosahovala lepších výsledků.

Zásadním problémem, kterému jsem ve své práci čelil, byla nepohyblivost robota v prostředí prodejny. Opět jsem nastínil možnost řešení tohoto problému implementací vlastních bezpečnostních reflexů, které by lépe pracovaly s údaji ze světlem zmatených senzorů. Nebude se jednat o snadný úkol, jelikož jde o bezpečností prvek aplikace nasazené ve veřejném prostoru s pohybem zákazníků.

Závěr

Cílem této práce bylo navrhnout a implementovat interaktivní aplikaci pro robota Pepper, která bude sloužit pro podporu péče o zákazníky na pobočce mobilního operátora. Nová aplikace se měla poučit z chyb a nedostatků jiných podobných aplikací, zahrnout a vhodně využít moderní technologie.

V práci jsem nejprve krátce představil robota Pepper a jeho vybavení. Následně jsem provedl analýzu podobných existujících aplikací, dostupných technologií pro rozpoznávání řeči, zpracování přirozeného jazyka, ale i lokalizaci v prostoru. Na základě analýzy jsem vytvořil návrh nové aplikace, jehož součástí byly konverzační scénáře, metoda lokalizace ale i architektura celé aplikace.

Před implementací jsem nalezl řešení, jak do robotova systému instalovat software bez cross-compilingu, především jsem potřeboval obejít zastaralou verzi Python 2.7.6. Na robota jsem úspěšně nasadil nástavbu Gentoo Prefix, díky které jsem problém překonal a mohl dále pracovat s novější verzí Python a používat potřebné knihovny.

Dále jsem v rámci navržené modulární řídicí aplikace integrované s frameworkem NAOqi implementoval konverzační subsystém. Nejprve jsem aplikaci propojil s cloudovou službou rozpoznávání řeči Google Cloud Speech-to-Text. Dále jsem v cloudové službě pro zpracování přirozeného jazyka Wit.ai natrénoval umělou inteligenci tak, aby zjišťovala záměry z dodaného přepisu hlasového vstupu a službu jsem následně propojil se svou aplikací. Pro stávající robotův dialogový modul jsem dle návrhu implementoval konverzační scénáře a přizpůsobil ho tak, aby jeho hlavním vstupem byly výsledky zpracování přirozeného jazyka.

Do konverzace jsem začlenil i kvízy, které lze dále jednoduše uživatelsky vytvářet pomocí konfiguračních souborů bez znalosti syntaxe konverzačních souborů. Zároveň jsem aplikaci propojil s operátorovou databází konaných událostí a naučil robota zvát na vyhledané akce. Robota jsem naučil zpracovat člověkem vyslovené časové údaje tím, že jsem lokalizoval a upravil knihovnu parsedatetime. Dále jsem robotovi vštípil schopnost správně česky vyslovit

datum a čas. Aby robot správně vyslovoval slova, která mu dělají problém, připravil jsem konfigurační soubor s fonetickými přepisy těchto slov, které následně ve své mluvě robot používá.

Pro neverbální komunikaci jsem vytvořil intuitivní tabletovou aplikaci inspirovanou hlasovými asistenty a pomocí frameworku NAOqi ji propojil s řídicí aplikací. Pro usnadnění aplikace zobrazuje přepis hlasového vstupu i robotovy odpovědi a nabízí rychlé odpovědi, pro zákazníky, kteří na robota nechtějí hovořit. Zároveň jsem přizpůsobil rozhraní pro ovládání robotových LED diod v očích a na ramenou a do dialogu začlenil gestikulaci.

V důsledku problémů s pohybem v prodejně sice robot nedokáže provést zákazníka prostorem, zato se ale orientuje ve vystavených technologiích, a umí o nich krátce pohovořit. Pro lokalizaci používá orientační body, které z obrazu kamer identifikuje pomocí umělé inteligence. Pro natrénování neuronové sítě jsem připravil rozsáhlý dataset s trénovacími daty. Pro rozpoznávání objektů jsem musel vzhledem k malému výkonu robotova hardwaru vytvořit program pro externí server, se kterým aplikace komunikuje po síti. Následné vyhodnocení pozice v mapě na základě detekovaných orientačních bodů a robotovy odometrie probíhá pomocí implementovaného algoritmu particle filter localization. Lokalizační systém jsem na závěr své práce s uspokojivými výsledky otestoval.

Implementoval jsem i webovou administrační aplikaci, která je přístupná zaměstnancům operátora a slouží k ovládání a konfiguraci aplikace, zároveň zobrazuje i průběh robotovy lokalizace v mapě prodejny.

Výsledná aplikace uspěla v průzkumu mínění a je připravena převzít roli té stávající. Zároveň má aplikace potenciál být snadno rozšířena a upravena pro podobné scénáře nasazení robota Pepper.

Literatura

[1] Pepper the Robot. In: *Wikimedia Commons* [online], Wikimedia Commons. 2016, [cit. 2022-04-06]. Dostupné z: https://commons.wikimedia.org/wiki/File:Pepper_the_Robot.jpg

[2] Jak to u nás vypadá. In: *Magenta Experience Center* [online], Experience centrum od T-Mobile Czech Republic a. s. 2021, [cit. 2022-04-15]. Dostupné z: https://magentaexperience.cz/img/content/gallery/00-uvod/mex_intro_image_6.jpg

[3] A robot designed to interact with humans. In: *Pepper the humanoind and programmable robot* [online], SoftBank Robotics. [cit. 2022-04-15]. Dostupné z: https://www.softbankrobotics.com/emea/en/pepper

[4] PANDEY, Amit Kumar, GELIN, Rodolphe: A Mass-Produced Sociable Humanoid Robot: Pepper: The First Machine of Its Kind. *IEEE Robotics Automation Magazine*, [online], ročník 25, č. 3, 2018: s. 40–48, doi:10.1109/MRA.2018.2833157, [cit. 2022-04-13]. Dostupné z: https://ieeexplore.ieee.org/document/8409927

[5] Pepper - Documentation [online]. SoftBank Robotics, [cit. 2022-04-15]. Dostupné z: http://doc.aldebaran.com/2-5/home_pepper.html

[6] NAOqi - Developer guide [online]. SoftBank Robotics, [cit. 2022-04-15]. Dostupné z: http://doc.aldebaran.com/2-5/index_dev_guide.html

[7] POT, E., MONCEAUX, Jerome et al.: Choregraphe: a graphical tool for humanoid robot programming. In: *RO-MAN 2009 - The 18th IEEE International Symposium on Robot and Human Interactive Communication* 2009, s. 47–48, doi:10.1109/ROMAN.2009.5326209, [cit. 2022-04-15]. Dostupné z: https://ieeexplore.ieee.org/document/5326209

[8] GARDECKI, Arkadiusz and PODPORA, Michal: Experience from the operation of the Pepper humanoid robots. In: *2017 Progress*

in Applied Electrical Engineering (PAEE) [online], 2017, str. 3, doi: 10.1109/PAEE.2017.8008994, [cit. 2022-04-18]. Dostupné z: https://ieeexplore.ieee.org/document/8008994

[9] ROBOT PEPPER T-DEE JE OD 6. PROSINCE SOUČÁSTÍ T-MOBILE TÝMU [online]. In: *TISKOVÉ CENTRUM*, T-Mobile Czech Republic a.s., 2014, [cit. 2022-04-16]. Dostupné z: http://www.t-press.cz/cs/novinky/robot-pepper-t-dee-je-od-6-prosince-soucasti-t-mobile-tymu.html

[10] VACOVSKÝ, Marek: Robot T-Dee se naučil česky a za úkol má bavit zákazníky T-Mobile [online]. In: *mobilenet.cz*, 24net s.r.o., 2019, [cit. 2022-04-16]. Dostupné z: https://mobilenet.cz/clanky/robot-t-dee-se-naucil-cesky-a-za-ukol-ma-bavit-zakazniky-t-mobile-36992

[11] PAPADOPOULOS, Ioannis: Vydejte se s námi do unikátního T-Mobile Magenta Experience Center [online]. In: *mobilenet.cz*, 24net s.r.o., 2021, [cit. 2022-04-16]. Dostupné z: https://mobilenet.cz/clanky/vydejte-se-s-nami-do-unikatniho-t-mobile-magenta-experience-center-videoreportaz-z-prodejny-budoucnosti-43144

[12] TOLIMAT, Rani: T-Dee Pepper aneb robot, který umí česky [online]. In: *SVĚTCHYTŘE.CZ*, SocialBooster s.r.o., 2019, [cit. 2022-04-16]. Dostupné z: https://www.svetchytre.cz/a/pr45k/t-dee-pepper-aneb-robot-ktery-umi-cesky

[13] GARDECKI, Arkadiusz, PODPORA, Michal et al.: The Pepper Humanoid Robot in Front Desk Application. In: *2018 Progress in Applied Electrical Engineering (PAEE)* [online], 2018, s. 1–7, doi: 10.1109/PAEE.2018.8441069, [cit. 2022-04-16]. Dostupné z: https://ieeexplore.ieee.org/document/8441069

[14] STAHOVSKIS, Vladislav: Developing an application for Pepper – problems and solutions [online]. In: *diatomenterprises.com*, Diatom Enterprises, [cit. 2022-04-16]. Dostupné z: https://diatomenterprises.com/developing-an-application-for-pepper-problems-and-solutions/

[15] ALLEGRA, Dario, ALESSANDRO, Francesco et al.: Experiences in Using the Pepper Robotic Platform for Museum Assistance Applications. In: *2018 25th IEEE International Conference on Image Processing (ICIP)* [online], 2018, s. 1033–1037, doi:10.1109/ICIP.2018.8451777, [cit. 2022-04-16]. Dostupné z: https://ieeexplore.ieee.org/document/8451777

[16] ABDELHAMID, Abdelaziz A., ABDULLA, Waleed H. et al.: Robo-ASR: A Dynamic Speech Recognition System for Service Robots. In:

Social Robotics [online], Springer Berlin Heidelberg, 2012, ISBN 978-3-642-34103-8, str. 485, doi:10.1007/978-3-642-34103-8_49, [cit. 2022-04-18]. Dostupné z: https://link.springer.com/chapter/10.1007/978-3-642-34103-8_49

[17] DEUERLEIN, Christian, LANGER, Moritz et al.: Human-robot-interaction using cloud-based speech recognition systems. *Procedia CIRP*, [online], ročník 97, 2021: s. 130–135, ISSN 2212-8271, doi: 10.1016/j.procir.2020.05.214, [cit. 2022-04-18]. Dostupné z: https://www.sciencedirect.com/science/article/pii/S2212827120314359

[18] DOOSTDAR, Masrur, SCHIIFER, Stefan et al.: A Robust Speech Recognition System for Service-Robotics Applications. In: *RoboCup 2008: Robot Soccer World Cup XII* [online], Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, ISBN 978-3-642-02921-9, s. 1–12, [cit. 2022-04-18]. Dostupné z: https://link.springer.com/chapter/10.1007/978-3-642-02921-9_1

[19] Speech to text [online]. Microsoft, [cit. 2022-04-16]. Dostupné z: https://azure.microsoft.com/en-us/services/cognitive-services/speech-to-text

[20] Watson Speech to Text [online]. IBM, [cit. 2022-04-16]. Dostupné z: https://www.ibm.com/cloud/watson-speech-to-text

[21] Speech-to-Text [online]. Google, [cit. 2022-04-16]. Dostupné z: https://cloud.google.com/speech-to-text

[22] JI, Yangfeng, HAKKANI-TÜR, Dilek et al.: A variational Bayesian model for user intent detection. In: *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* [online], 2014, s. 4072–4076, doi:10.1109/ICASSP.2014.6854367, [cit. 2022-04-18]. Dostupné z: https://ieeexplore.ieee.org/document/6854367

[23] ZHANG, Hanxiao, SONG, Wei et al.: Query Classification Using Convolutional Neural Networks. In: *2017 10th International Symposium on Computational Intelligence and Design (ISCID)*, ročník 2 [online], 2017, s. 441–444, doi:10.1109/ISCID.2017.212, [cit. 2022-04-18]. Dostupné z: https://ieeexplore.ieee.org/document/8283309

[24] LIU, Jiao, LI, Yanling, LIN, Min: Review of Intent Detection Methods in the Human-Machine Dialogue System. *Journal of Physics: Conference Series*, [online], ročník 1267, č. 1, 2019: str. 3, doi:10.1088/1742-6596/1267/1/012059, [cit. 2022-04-18]. Dostupné z: https://doi.org/10.1088/1742-6596/1267/1/012059

[25] Dialogflow [online]. Google, [cit. 2022-04-17]. Dostupné z: https://cloud.google.com/dialogflow

[26] Language Understanding (LUIS) [online]. Microsoft, [cit. 2022-04-17]. Dostupné z: https://www.luis.ai/

[27] Build Natural Language Experiences [online]. Wit.ai, Inc, [cit. 2022-04-17]. Dostupné z: https://wit.ai/

[28] Introduction to Rasa Open Source [online]. Rasa Technologies Inc, [cit. 2022-04-17]. Dostupné z: https://rasa.com/docs/rasa/

[29] PERERA, Vittorio, PEREIRA, Tiago et al. Setting Up Pepper For Autonomous Navigation And Personalized Interaction With Users. [online], 2017, doi:10.48550/ARXIV.1704.04797, [cit. 2022-04-18]. Dostupné z: https://arxiv.org/abs/1704.04797

[30] GÓMEZ, Cristopher, MATTAMALA, Matías et al.: Visual SLAM-Based Localization and Navigation for Service Robots: The Pepper Case. In: *RoboCup 2018: Robot World Cup XXII* [online], Cham: Springer International Publishing, 2019, ISBN 978-3-030-27544-0, s. 32–44, [cit. 2022-04-18]. Dostupné z: https://link.springer.com/chapter/10.1007/978-3-030-27544-0_3

[31] GROOT, Robert: *Autonomous Exploration and Navigation with the Pepper robot*. Diplomová práce, Utrecht University, 2018, [cit. 2022-04-18]. Dostupné z: https://studenttheses.uu.nl/handle/20.500.12932/40276

[32] SILVA, João R. and SIMÃO, Miguel et al.: Navigation and obstacle avoidance: a case study using Pepper robot. In: *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*, ročník 1 [online], 2019, s. 5263–5268, doi:10.1109/IECON.2019.8927009, [cit. 2022-04-18]. Dostupné z: https://ieeexplore.ieee.org/abstract/document/8927009

[33] QUIGLEY, Morgan, GERKEY, Brian et al.: ROS: an open-source Robot Operating System. In: *ICRA 2009* [online], 2009, [cit. 2022-04-20]. Dostupné z: http://robotics.stanford.edu/~ang/papers/icraoss09-ROS.pdf

[34] TAKETOMI, Takafumi, UCHIYAMA, Hideaki, IKEDA, Sei: Visual SLAM algorithms: a survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications*, [online], ročník 9, 2017, doi: 10.1186/s41074-017-0027-2, [cit. 2022-04-21]. Dostupné z: https://ipsjcva.springeropen.com/articles/10.1186/s41074-017-0027-2

[35] AZZAM, Rana, TAHA, Tarek et al.: Feature-based visual simultaneous localization and mapping: a survey. *SN Applied Sciences*, [online], ročník 2, 2020, doi:10.1007/s42452-020-2001-3, [cit. 2022-04-21]. Dostupné z: https://link.springer.com/article/10.1007/s42452-020-2001-3

[36] THRUN, Sebastian: Particle Filters in Robotics. In: *Uncertainty in AI (UAI)*, ročník 2 [online], Citeseer, 2002, s. 511–518, [cit. 2022-04-21]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.20.567&rep=rep1&type=pdf

[37] STACHNISS, Cyrill, BURGARD, Wolfram: Particle Filters for Robot Navigation. *Foundations and Trends® in Robotics*, [online], ročník 3, 2014: s. 211–282, doi:10.1561/2300000013, [cit. 2022-04-21]. Dostupné z: https://www.nowpublishers.com/article/Details/ROB-013

[38] XU, Benda, AMADIO, Guilherme et al.: Gentoo Prefix as a Physics Software Manager. *EPJ Web Conf.*, [online], ročník 245, 2020: str. 05036, doi:10.1051/epjconf/202024505036, [cit. 2022-04-20]. Dostupné z: https://doi.org/10.1051/epjconf/202024505036

Příloha **A**

Uživatelská příručka

Instalace (zavedení) řídicí aplikace

Postup

1. přenos archivu s Gentoo Prefix *pepper_gentoo_prefix.tar.gz* do adresáře */home/nao* v robotově souborovém systému
2. rozbalení archivu (`tar -xvf pepper_gentoo_prefix.tar.gz`)
3. přenos zdrojových souborů aplikace, tzn. *run_operator_application.sh* a adresáře *operator_application*, rovněž do */home/nao*
4. **pokud se má aplikace spouštět po zapnutí robota**, je možné navíc nahradit soubor *autoload.ini* na totožné cestě jako v tomto repozitáři, tj. */home/nao/naoqi/preferences/autoload.ini*

Po dokončení tohoto postupu je instalace aplikace dokončena, nyní lze přejít k jejímu spuštění.

Doplnění k Gentoo Prefix

Pro aplikaci je stěžejní „instalace“ Gentoo Prefix[15], utility, která umožňuje bez oprávnění uživatele root na robotovi instalovat a spouštět jakékoli balíčky a programy. V připraveném archivu *pepper_gentoo_prefix.tar.gz* je upravená verze Gentoo Prefix od Softbank robotics[16], respektive Sama Pfeiffera[17], která mimo jiné obsahuje ROS, Python 2.7.18(!) a 3.6 a knihovny nezbytné pro aplikaci.
Konkrétně se jedná o knihovny:

[15] `https://wiki.gentoo.org/wiki/Project:Prefix`
[16] `https://github.com/softbankrobotics-research/sbre_robot_ros_gentoo_prefix`
[17] `https://github.com/awesomebytes/gentoo_prefix_ci`

```
google-cloud-speech==1.3.2, pexpect, ntplib, pytz,
simple_websocket_server, numpy, opencv-python, scipy, pandas
wave, parsedatetime (mnou lokalizovaná verze)
```

Spuštění řídicí aplikace

Pro spuštění řídicí aplikace (po její instalaci) je třeba spustit skript *run_operator_application.sh*.
Je možné použít následující přepínače:

```
./run_operator_application.sh
-c    přesměruj výstup do konzole
      (výchozí je výstup do logovacího souboru)
-b    spusť aplikaci na pozadí
```

Doplnění k roli skriptu při spouštění aplikace

Úlohou skriptu *run_operator_application.sh* je příprava prostředí pro běh aplikace a její následné spuštění v rámci Gentoo Prefix.
Konkrétně vykoná následující kroky:

- vytvoří logovací soubor s aktuálním datem v názvu (pokud není spuštěn s přepínačem -c)
- vytvoří/ověří existenci softlinku pro zdrojové kódy tabletové aplikace do */home/nao/.local/share/PackageManager/apps/operator_application*
- zkontroluje přítomnost Gentoo Prefix v adresáři */home/nao*
- inicializuje proměnné prostředí v Gentoo Prefix, konkrétně PYTHONPATH a GOOGLE_APPLICATION_CREDENTIALS
- pomocí skriptu *_fix_robot_time.py* nastaví robotovi aktuální datum a čas (nutné pro roboty, kteří si ho po restartu nepamatují)
- následně již spustí aplikaci (*main.py*) v prefixovaném prostředí pomocí skriptu *gentoo/executeonprefix* ∗

Pokud se jakýkoliv z uvedených kroků nepodaří provést, je spuštění zrušeno a chyba robotem verbálně oznámena pomocí skriptu *_announce_error.py*.

∗ Vzhledem k tomu, že aplikace vyžaduje Gentoo Prefix (potřebuje vyšší verzi Pythonu 2.7.17 a balíčky, které nefungují, nebo je nelze nainstalovat do verze 2.7.6), je třeba ji v tomto prefixovaném prostření (*/tmp/gentoo*) spustit pomocí skriptu *executeonprefix*.

Spuštění serveru pro rozpoznávání objektů

Server pro rozpoznávání objektů je nutné spustit na stroji v robotově místní síti. Předpokladem je nainstalovaný Python 3.x a knihovna Tensorflow ve verzi 1.14. Spuštění probíhá pomocí skriptu *server.py*. IP adresu serveru je nutné nastavit jak do tohoto skriptu, tak do konfigurační proměnné OBJECT_DETECTION_SERVER_IP_ADDRESS v souboru *config.py* řídicí aplikace.

Příloha **B**

Seznam použitých zkratek

API Application Programming Interface

CSS Cascading Style Sheets

EOL End Of Life

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

JSON JavaScript Object Notation

LED Light-Emitting Diode

REST Representational state transfer

RMS Rooot Mean Square

ROS Robot Operating System

SDK Software Development Kit

SLAM Simultaneous Localization and Mapping

SNI Server Name Indication

SSL Secure Sockets Layer

TCP Transmission Control Protocol

UWB Ultra-Wideband

vSLAM Visual Simultaneous Localization and Mapping

XML Extensible markup language

YAML YAML Ain't Markup Language

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `thesis.pdf` ..... text práce ve formátu PDF
- `robot` ..... adresář s náležitostmi pro řídicí aplikaci
  - `run_operator_application.sh` ..... spouštěcí skript řídicí aplikace
  - `pepper_gentoo_prefix.tar.gz` ..... archiv s Gentoo Prefix
  - `libraries.txt` .. výpis knihoven pro Python 2.7 uvnitř Gentoo Prefix
  - `operator_application` ..... zdrojový kód řídicí aplikace
  - `naoqi` ..... konfigurační soubor pro spuštění při startu
- `server` ..... zdrojový kód pro server pro rozpoznávání objektů
  - `server.py` ..... spouštěcí skript serveru