TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Interaktivní ovládání humanoidního robota Nao pomocí SDK

Bakalářská práce

Studijní program: B2612 – Elektrotechnika a informatika
Studijní obor: 2612R011 – Elektronické informační a řídicí systémy

Autor práce: **Václav Jíše**
Vedoucí práce: Ing. Miroslav Holada, Ph.D.

Liberec 2017

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Interactive control of humanoid robot Nao using SDK

Bachelor thesis

Study programme: B2612 – Electrical Engineering and Informatics
Study branch: 2612R011 – Electronic Information and Control Systems

Author: **Václav Jíše**
Supervisor: Ing. Miroslav Holada, Ph.D.

Liberec 2017

Technická univerzita v Liberci
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2016/2017**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Václav Jíše**
Osobní číslo:	**M14000113**
Studijní program:	**B2612 Elektrotechnika a informatika**
Studijní obor:	**Elektronické informační a řídicí systémy**
Název tématu:	**Interaktivní ovládání humanoidního robota Nao pomocí SDK**
Zadávající katedra:	**Ústav informačních technologií a elektroniky**

Zásady pro vypracování:

1. Seznamte se s robotem NAO, zaměřte se na SDK od výrobce robotu.
2. Navrhněte software pro interaktivní ovládání robota NAO, který umožní realizaci základních funkcí robotu bez znalosti jeho programování.
3. Implementujte software na robota a prakticky ověřte.

Rozsah grafických prací: **Dle potřeby dokumentace**

Rozsah pracovní zprávy: **cca 30-40 stran**

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] **ALDEBARAN. Aldebaran documentation: NAO Documentation [online]. 2016 [cit. 2016-10-16]. Dostupné z: http://doc.aldebaran.com/2-1/home_nao.html**

[2] **NOVÁK, Petr. Mobilní roboty: pohony, senzory, řízení. 1. vyd. Praha: BEN - technická literatura, 2005. ISBN 80-7300-141-1.**

[3] **KISUNG, SEO. Using NAO: Introduction to interactive humanoid robots.**

Vedoucí bakalářské práce: **Ing. Miroslav Holada, Ph.D.**
Ústav informačních technologií a elektroniky

Konzultant bakalářské práce: **Bc. Šimon Škapik**

Datum zadání bakalářské práce: **12. září 2016**

Termín odevzdání bakalářské práce: **15. května 2017**

Technická univerzita v Liberci - 14 -

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

prof. Ing. Ondřej Novák, CSc.
vedoucí ústavu

V Liberci dne 12. září 2016

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 12. 5. 2017

Podpis: 

PODĚKOVÁNÍ

Rád bych zde poděkoval svému vedoucímu práce Ing. Miroslavu Holadovi, Ph.D. za jeho věcné poznámky a za vedení při vývoji aplikace.

ABSTRAKT

Předmětem této bakalářské práce je rozšíření možností ovládání a vytváření programů pro humanoidního robota NAO. Tohoto cíle je dosaženo pomocí vytvořené aplikace, která umožňuje intuitivně robota programovat, i pro méně zkušené uživatele. Uživatel je schopen vytvářet sekvence za sebou jdoucích příkazů, které by robot vykonával. Zaměření aplikace je hlavně na pohybové funkce robota. Aplikace je vytvořena užitím vývojových nástrojů SDK od výrobce robota. Z několika dostupných programovacích jazyků byl vybrán jazyk Python pro jeho rozsáhlou dokumentaci a velkou uživatelkou podporu. Vytvořené sekvence příkazů v aplikaci je možné následně ukládat a později znovu použít, popřípadě je rozšířit o nové příkazy.

Klíčová slova:

NAO, humanoidní robot, interaktivní ovládání, SDK, Python

ABSTRACT

The subject of this bachelor thesis is to expand the possibilities of controlling and creating new programs for the humanoid robot NAO. This new way of programming the robot should allow the user to program the robot intuitively, even for a less skilled user. The user is capable of creating sequences of consecutively commands, which the robot will do afterwards. The main focus of this application is on the motion functions of the robot. Application is created using development tools, also known as SDK, from the robot manufacturer. From several programming languages, Python was chosen for its large documentation. At the end, the user can save the sequences and use them later again. There is also possibility to extend them with new commands.

KEYWORDS:

NAO, humanoid robot, interactive control, SDK, Python

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM ZKRATEK

FPS – Frames Per Second, počet snímků za sekundu

IEEE - Institute of Electrical and Electronics Engineers - Institut pro elektrotechnické a elektronické inženýrství

RAM – Random Access Memory, operační pamět počítače

SDK – Software Development Kit, vývojové nástroje

TTS – Text To Speech, převod textu na řeč

USB – Universal Serial Bus, univerzální sériová sběrnice

1 Úvod

Humanoidní robot od francouzské firmy Aldebaran [viz Obrázek 1], jehož ovládáním se tato práce zaobírá, je primárně určený pro výukové účely a seznámení studentů a širší veřejnosti s robotikou. Robot je proto vybaven velkým množstvím senzorů, tlačítek a dalších ovládacích prvků. Toto však znamená, že používání robota bez předešlého studování je pro začátečníka relativně obtížné. Po navštívení několika veletrhů, kde robot slavil velký úspěch při lákání pozornosti a zájmu návštěvníků, se zjistilo, že se robot skvěle hodí pro prezentační účely, nebo pro komunikaci s lidmi. Proto vznikla myšlenka zjednodušit jeho programování. Proto je cílem této práce zjednodušení a zvýšení intuitivnosti ovládání a programování robota.

Na akcích jako jsou dny otevřených dveří, prezentace školy, nebo jenom pro nalákání studentů k oboru robotiky, je potřeba mít schopnost robota rychle naučit novým pohybům a příkazům. Na toto použití není potřeba využívat velké množství senzorů ani všech složitějších funkcí robota, ale je potřeba, aby se robot dal naučit v co nejkratším čase různým pohybovým sekvencím. Proto byla snaha vytvořit program pro ovládání robota, který by byl schopen ovládat člověk s téměř nulovou zkušeností s robotem. Způsob ovládání by měl být co nejvíce intuitivní a nejsnadnější, i za nevýhody ztráty některých funkcí robota. V programu by mělo být možno vytvářet sekvence pohybů a příkazů, které bude robot vykonávat.

V bakalářské práci je zvolen přístup programování robota pomocí SDK. Pro vývoj byl zvolen jazyk Python a v jazyce Python je tvořeno také grafické rozraní, a to pomocí knihoven Tkinter. Z několika dostupných jazyků byl vybrán Python pro jeho velkou podporu jak ostatních uživatelů, tak samotného výrobce a jeho rozsáhlou dokumentaci. Aplikace je spuštěna na PC a s robotem se spojí přes Wifi síť. Po spojení je pak možné robota ovládat a učit ho novým sekvencím pohybů. Další výhodou by měla být přenositelnost jednotlivých sekvencí, které se ukládají v podobě textového souboru, a také celého programu, jehož velikost je pouze několik MB. Program je dostupný ve formě instalátoru, ale také ve formě přenositelného programu bez nutnosti instalace, takže program může být spuštěn i například z USB disku.

2 ROBOT NAO

Humanoidní Robot NAO je vyrobený francouzskou firmou Aldebaran Robotics hlavně pro potřeby výuky studentů. Robot by měl seznámit studenta se základním programováním mobilních robotů, a také celkově s problematikou robotiky. Proto je robot vybaven velkým množstvím senzorů, tlačítek a způsobů interakce s člověkem. Na trh byl uveden v roce 2006, a od té doby prošel několika změnami. Díky prodejnosti přes 10 000 kusů, má relativně velkou uživatelskou základnu. Robot v průběhu let prošel několika verzemi. Pro vývoj byl k dispozici robot v nejnovější verzi, ale program vytvořený v této práci je kompatibilní se všemi ostatními verzemi. Firma má v nabídce aktuálně i jiné pokročilejší roboty jménem Pepper a Romeo.

OBRÁZEK 1 - ROBOT NAO

2.1 HARDWAROVÉ VYBAVENÍ

Robot je skoro 60 centimetrů vysoký a jeho váha je 5.6Kg. Tento robot je poháněn procesorem Intel Atom Z530, který je taktovaný na frekvenci 1.6 GHz. Tento procesor je doplněn o 1GB RAM paměti. Data je možné ukládat na Micro SDHC kartě, kterou je robot vybaven. Velikost karty je 8GB. Dále je v robotovi instalována Flash paměť o velikosti 2 GB. Tato paměť je využita pro operační systém. V případě robota NAO se jedná o systém, který využívá jádro Unix.

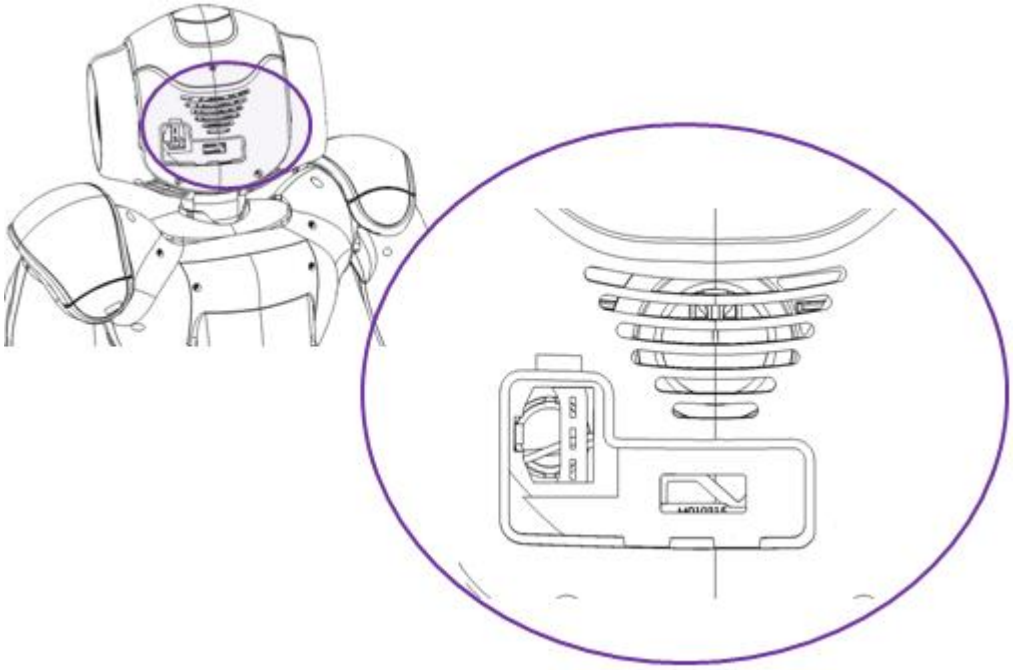
Jak je ze specifikací vidět, robot je vybaven dnes již celkem zastaralými komponenty. Proto je pro složitější programy, hlavně pokud se jedná o zpracování obrazu, nebo rozpoznávání objektů, nutné využít SDK. Při použití SDK se veškeré výpočty a operace s daty provádějí na PC, na kterém je program spuštěný. Tím se výrazně zvětší možnosti práce s obrazem.

2.1.1 MOŽNOSTI KOMUNIKACE

Robot nabízí několik způsobů komunikace s okolím. Hlavní komunikace probíhá bezdrátově přes WiFi připojení. Robot ve verzi V4 a nižší podporuje protokoly IEEE 802.11 b/g/n a verze V5 je schopna se připojit k síti s IEEE 802.11 a/b/g/n.

Dále je k dispozici ethernetový port. Ten slouží hlavně při prvním nastavení robota, pro aktualizaci robota a popřípadě k rychlejšímu přenosu souborů z robota a do robota. Maximální rychlost Ethernetu je 1000Mb/s. Tento port je přístupný po odklopení dvířek na zadní straně hlavy robota.

Periferie jako například USB disky, Kinect, nebo Asus 3D pro práci s 3D informacemi a mapováním okolí, lze připojit přes USB port také na zadní straně hlavě [viz Obrázek 2] pod odnímatelnou záslepkou. Přes USB lze také připojit vývojová deska Arduino pro rozšíření senzorických schopností robota, případně přidání dalších možností interakce s robotem.



OBRÁZEK 2 - UMÍSTĚNÍ PORTŮ NA HLAVĚ [3]

2.1.2 KAMERA

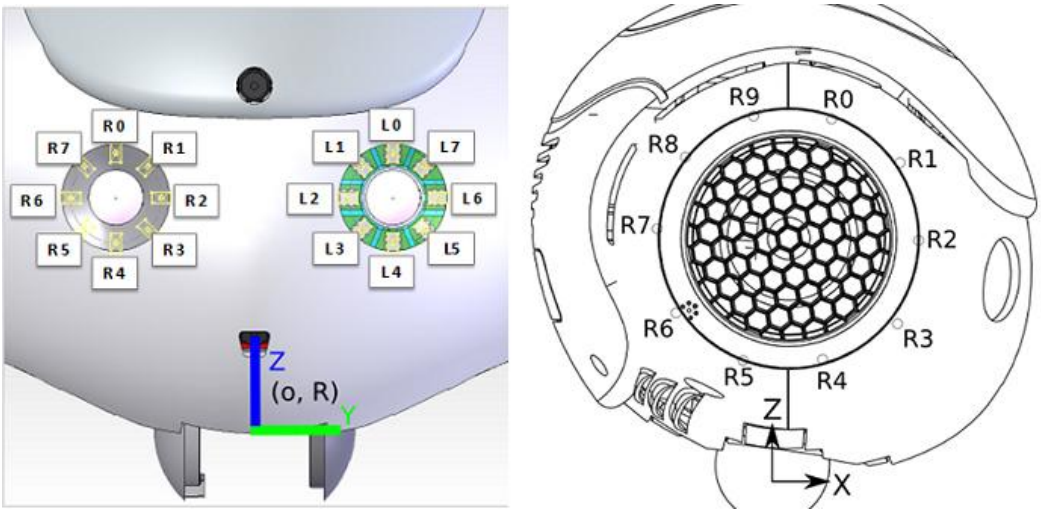
Robot NAO je vybaven dvěma kamerami pro snímání svého okolí. První kamera je umístěná na čele robota a slouží k pohledu před robota. Druhá je ve spodní části hlavy. Tato kamera je využívána hlavně k detekci překážek před robotem. Jedná se ale o kamery s relativně nízkým rozlišením 1.22 MP. Kamera je dostačující pro základní orientaci robota v prostoru, ale pro potřebu rozpoznání objektů je nutné předmět umístit do blízké vzdálenosti od hlavy robota, aby byl schopný předmět rozpoznat. Také při nízkém osvětlení se obraz velmi zhorší. Při zhoršujícím se osvětlením začne obraz obsahovat velké množství šumu. Maximální frekvence snímkování je 30 FPS. Dostupná je také hlava s kamerami umístěními vedle sebe místo očí. Robot pak je schopen stereovidění, které se využívá například na mapování prostoru a okolí robota.

2.1.3 ZABUDOVANÉ REPRODUKTORY

Pro potřeby přehrávání je možné využít dva hlasité reproduktory umístěné v hlavě robota. V kombinaci s dostupným TTS systémem je robot schopen komunikovat s okolím. Robot také umožňuje přehrávat hudbu, nebo jakoukoli nahrávku ve formátu wav, mp3 nebo ogg.

2.1.4 LED DIODY

Pro signalizaci stavů využívá robot mimo slovního hlášení také LED diody. Tyto diody jsou uživatelsky dostupné a proto je možné je využít také v programech. K dispozici je 12 diod na vrchní straně hlavy, 8 RGB diod v každém oku a 10 diod na každé straně hlavy. Jejich rozložení je vidět na obrázku [viz Obrázek 3].



OBRÁZEK 3 - UMÍSTĚNÍ DIOD V HLAVĚ ROBOTA [6]

2.1.5 ULTRAZVUKOVÝ SENZOR

Pro měření vzdálenosti překážky od robota slouží dva ultrazvukové senzory v hrudi robota. Senzor v poslední verzi robota V5 je schopný rozpoznat vzdálenosti od 0.20m do 2.55m s rozlišením 1 – 4cm v závislosti na vzdálenosti. Pod hranici 0.20m robot není schopen s jistotou určit, jak daleko před senzorem se objekt nachází před senzorem. Robot pouze vrátí informaci, že se před ním nachází objekt. Senzory se využívají pro detekci překážek a orientaci robota v prostoru.

2.1.6 INERCIÁLNÍ JEDNOTKA

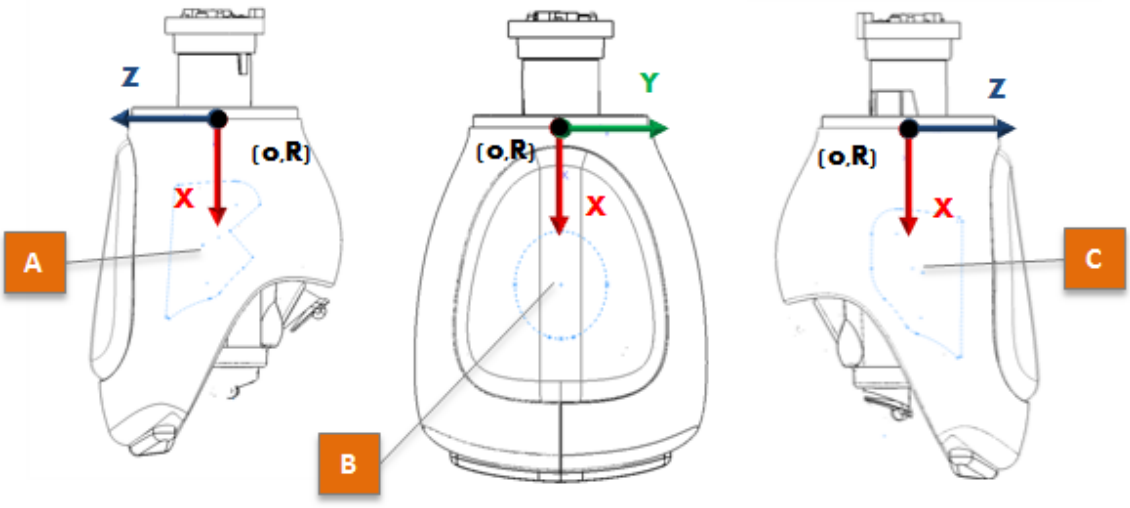
V robotovi se nachází inerciální jednotka, která v poslední verzi robota obsahuje 3osý gyroskop a 3osý akcelerometr. Tyto senzory se využívají primárně systémem pro detekci pádu robota, ale tato data jsou i uživatelsky přístupná. Tyto snímače jsou schopné detekovat změnu vybočení (yaw), klonění (pitch) a také klopení (roll). U robota NAO je standardně akcelerometr využíván pro stanovení polohy torza robota, a pokud je zaznamenán pohyb robota, úhel robota je dopočítán z gyroskopu. Gyroskop se využívá pro jeho lepší dynamické vlastnosti při pohybu.

2.1.7 ODPOROVÉ SENZORY SÍLY

Odporové senzory síly se nachází v chodidlech robota. Měří změnu odporu senzoru, která je úměrná síle působící na chodidla robota. Senzory jsou schopné změřit sílu od 0N do 25N. Tento senzor využívá manažer pádu zabudovaný v systému robota. Pokud se alespoň jedno chodidlo dotýká země, takže na něj působí síla, je tento manažer aktivní. Když robot ztratí rovnováhu a začne padat, robot zaujme bezpečnou polohu, kdy natáhne ruce před sebe a vypne přívod elektrického proudu do kloubů robota. Toto má za následek ochránění torza robota a díky povolení kloubů také zmenšení rizika zničení robota a jeho končetin.

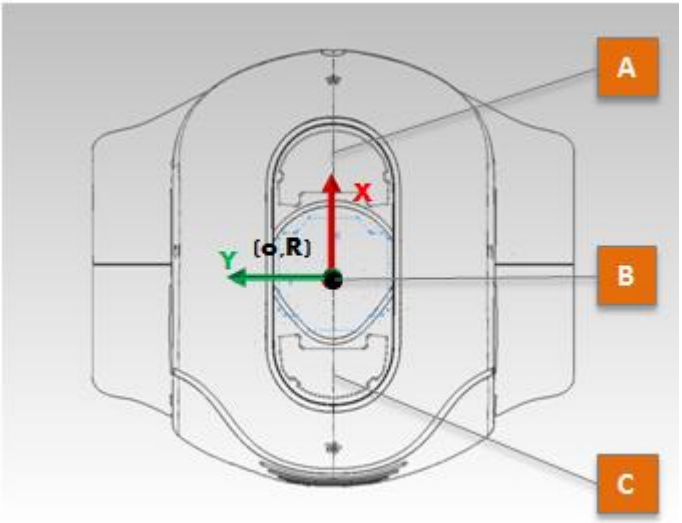
2.1.8 UŽIVATELSKY DOSTUPNÁ TLAČÍTKA

Na robotovi se také nachází několik uživatelsky přístupných tlačítek. Mezi ně patří tři dotyková tlačítka na vrchní straně hlavy robota, dále také tři další tlačítka na každé ruce robota. Na každém chodidle robota potom nalezneme další dvě mechanická tlačítka sloužící například pro detekci překážek. Poslední a nejdůležitější tlačítko se nachází na hrudi robota. Krátkým stiskem tlačítka robot ohlásí základní údaje o svém stavu, jako je IP adresa, nebo chybové hlášení, a dlouhým stiskem tlačítka robota zapneme či vypneme.



OBRÁZEK 4 - UMÍSTĚNÍ TLAČÍTEK NA RUCE [4]

Tlačítko A je potom v programu nazváno jako Left, B jako Back a C jako Right



OBRÁZEK 5 - TALČÍTKA NA HLAVĚ ROBOTA [4]

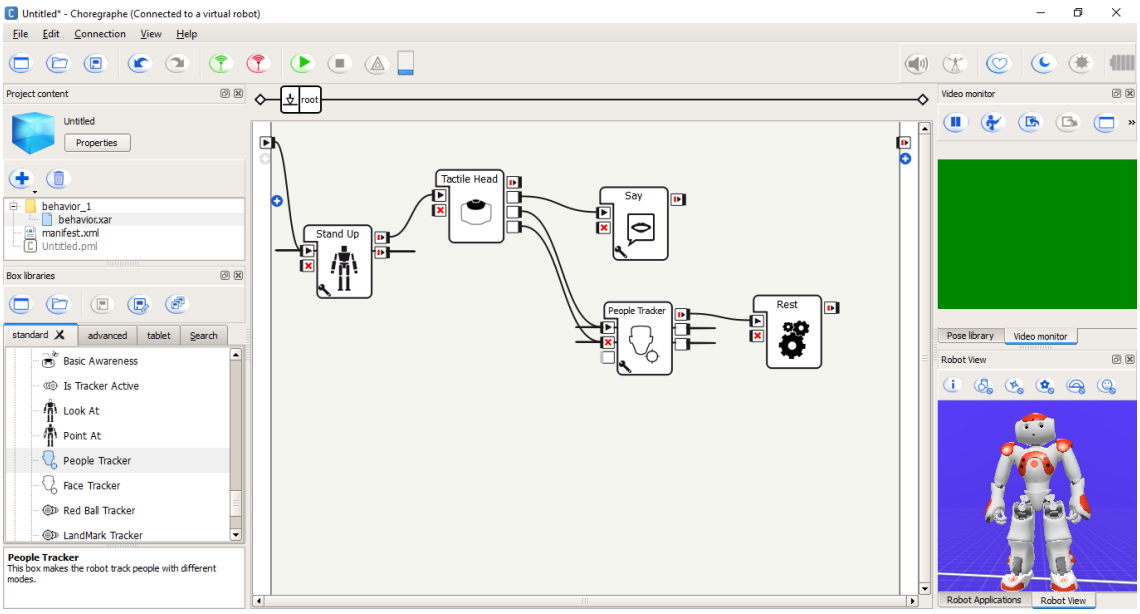
V programu jsou tato tlačítka označována jako Front, Middle a Rear.

2.2 PROGRAMOVÁNÍ ROBOTA

Protože robot slouží jako výuková pomůcka, výrobce nabízí několik možností, jak s robotem pracovat. Na stránkách výrobce je možné stáhnout veškeré potřebné nástroje pro používání robota a vývoj aplikací. Vývojář si také může vybrat z několika dostupných programovacích jazyků.

2.2.1 DOSTUPNÝ SOFTWARE

Nejjednodušší způsob programování a ovládání robota, ale také nejvíce omezený, je přístup programování pomocí výrobcem dodávaného programu Choregraphe [viz Obrázek 6]. Choregraphe umožňuje vytvářet programy a ovládat robota. Tento přístup se doporučuje pro úplné začátečníky bez zkušeností s ovládáním robota nebo s programováním [5]. Programování a ovládání robota probíhá využíváním předem definovaných bloků. Tyto bloky se poté spojují, a tím se vytváří program pro ovládání.



OBRÁZEK 6 - GRAFICKÉ ROZHRANÍ PROGRAMU CHOREGRAPHE

Výhodou programu Choregrape je, že nabízí několik dalších funkcí. Jak je vidět z obrázku grafického rozhraní aplikace [viz Obrázek 6], tak v pravé dolní části rozhraní nalezneme 3D model robota, který se v reálném čase pohybuje zároveň s reálným robotem. Dále je v programu Choregraphe možnost sledovat v reálném čase výstup z kamery na hlavě robota.

Součástí aplikace je také virtuální robot, který umožní vytváření a spouštění programů i bez přítomnosti reálného robota. Veškeré pohyby virtuálního robota se promítají na 3D model v aplikaci. Virtuální robot umožňuje pracovat s veškerými pohybovými funkcemi, ale například jakékoli obrazové funkce, ať se jedná o sledování, či rozpoznání objektů, standardně neumožňuje. Tyto funkce je možné využít po doinstalování programu Webots od firmy Cyberbotics, ve kterém je robot vymodelován. V tomto programu lze vytvářet 3D prostředí a do něho robota umístit. Poté se zpřístupní i živý přenos z kamery robota a s ním také rozpoznávací funkce robota.

OBRÁZEK 7 - SIMULACE ROBOTŮ V APLIKACI WEBOTS [10]

Program vytvořený pomocí Choregraphe se potom nahraje do robota a je vykonáván do doby, než je ručně zastaven, nebo robot vypnut.

Možnosti tohoto programu jsou však omezené pouze předpřipravenými bloky. Ve vývojovém prostředí je sice možnost vytvářet vlastní bloky, ale k tomu uživatel potřebuje znát programovací jazyk Python a naučit se strukturu, kterou jsou tyto bloky programů vytvářeny.

Další možnost je využít jeden z programovacích jazyků, pro které výrobce nabízí vývojové nástroje SDK. Stačí stáhnout výrobcem dodávané SDK na počítač uživatele, potom se pomocí několika příkazů uživatel spojí s robotem a může začít vytvářet program. Tímto přístupem získáme kompletní kontrolu nad chováním robota. Program může být poté přenesen do robota, ale pro složitější programy je výhodný přístup, kdy program běží na počítači uživatele, protože výkon, kterým disponuje robot, je velmi nízký. Na výběr má vývojář jazyky Python, C++ a Java.

2.2.2 PROGRAMOVÁNÍ V JAZYCE PYTHON

Pro vývoj byl zvolen jazyk Python kvůli předešlým zkušenostem, a také protože je doporučován jako jazyk při začátku používání SDK. Pro jazyky C++ a Python je dokumentace od výrobce nejrozsáhlejší. Pro tvorbu grafického rozhraní bylo použito knihoven Tkinter. Tyto knihovny jsou standardním prostředkem pro tvorbu grafického rozhraní pro tento jazyk. Rozhraní je tvořeno kořenovým oknem. Toto okno by mělo být vytvořeno v programu pouze jedno, a všechny ostatní prvky by měly být definovány až za definicí tohoto okna. Prvky, které můžeme využívat, jsou nazývány tzv. widgety [2]. Widgetem je označováno vše od tlačítek, přes pole pro vkládání textu, až po různá plátna pro kreslení a vkládání dalších prvků.

Prvek musíme nejprve umístit do nějakého okna, nebo nějakého rámu, a dále ho v tomto okně někam umístit. Pro umístění máme dvě možnosti. Buď pomocí funkce pack(), nebo grid(). Obě tyto funkce řeknou prvku, aby se zvětšil na potřebnou velikost podle obsahu a zobrazil se na obrazovce. Rozdíl mezi těmito funkcemi je ten, že metoda pack() vkládá prvky jeden pod druhý, a metoda grid() do jakési pomyslné mřížky. Tím pádem metodě grid() musíme do parametru zadat do jakého sloupce a řádku se má prvek vložit. Tyto dva přístupy však nelze kombinovat mezi sebou v rámci jednoho rámu nebo okna.

Na konec je pak nutné zavolat funkci mainloop() pro vytvořené hlavní okno. Touto funkcí vstoupí program do nekonečné smyčky, která bude reagovat na události programu a uživatele. Smyčka se také stará o vykreslování prvků a jejich obnovování. Příklad Hello World, s jedním tlačítkem na uzavření, by potom vypadal následovně.

```
from Tkinter import *

okno = Tk()
napis=Label(okno,text="Ahoj svete!")
#napis.pack()
napis.grid(row=0,column=0)
tlac=Button(okno,text="zavrit",command=okno.destroy)
#tlac.pack()
tlac.grid(row=0,column=1)
okno.mainloop()
```

Všechny prvky, co se mají zobrazit, a také celý program, co se cyklicky vykonává, musí ležet mezi definicí hlavního okna a zavoláním metody mainloop().

3 APLIKACE PRO OVLÁDÁNÍ ROBOTA

3.1 PODMĚT K VYTVOŘENÍ APLIKACE

Program byl vytvořen z časté potřeby naučit robota jednoduchým pohybům pro různé prezentace a dny otevřených dveří školy. Problémem bylo, že k vytváření i těchto jednoduchých programů je potřeba dlouhá doba učení se s robotem, hlavně pro člověka, který s ním nikdy nepracoval. Právě z tohoto důvodu a potřeby rychlého naučení robota základním a jednoduchým úkolům byl po konzultaci s vedoucím práce vytvořen tento program.

Největší důraz byl kladen na jednoduché a intuitivní ovládání robota i pro uživatele, který s robotikou a programováním nemá žádné zkušenosti. Díky tomuto programu je možné robota naučit sérii pohybů a příkazů, bez nutnosti znalosti jakéhokoli programovacího jazyku a větších předešlých zkušeností s robotikou, a to vše v rychlém časovém horizontu.

3.2 DOSTUPNÉ FUNKCE APLIKACE

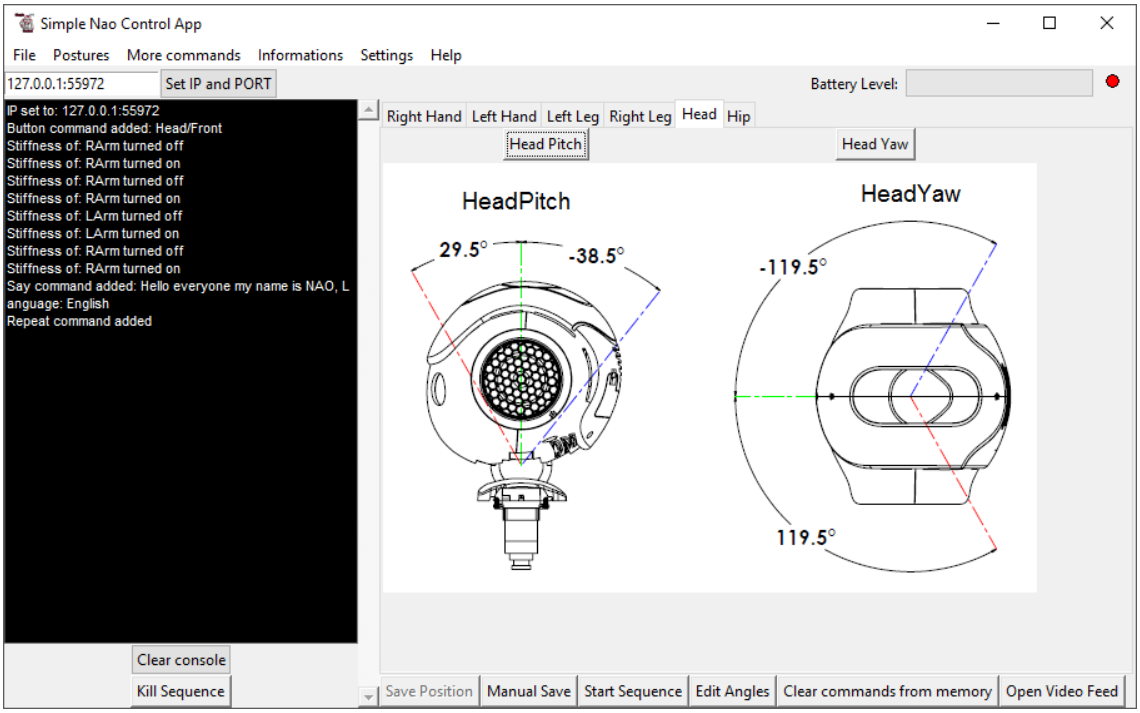
Program umožňuje naučit robota několika pokynům. Mezi hlavní patří ovládání a ukládání poloh kloubů, čekání na stisk definovaného tlačítka uživatelem pro kontrolu nad sekvencí, převod textu na řeč a další.

Po otevření programu se zobrazí základní obrazovka [viz Obrázek 9]. Na ní nalezneme veškeré potřebné prvky k začátku ovládání robota.

Na levé straně je umístěna konzole pro výpis stavu robota, nad ní je pole pro zadání IP adresy a portu robota. IP adresu je možné zjistit stiskem tlačítka na hrudi robota. V pravé části je nejdůležitější část okna, a to ovládání kloubů robota. Během vývoje aplikace bylo vyzkoušeno několik přístupů k ovládání tuhosti kloubů.

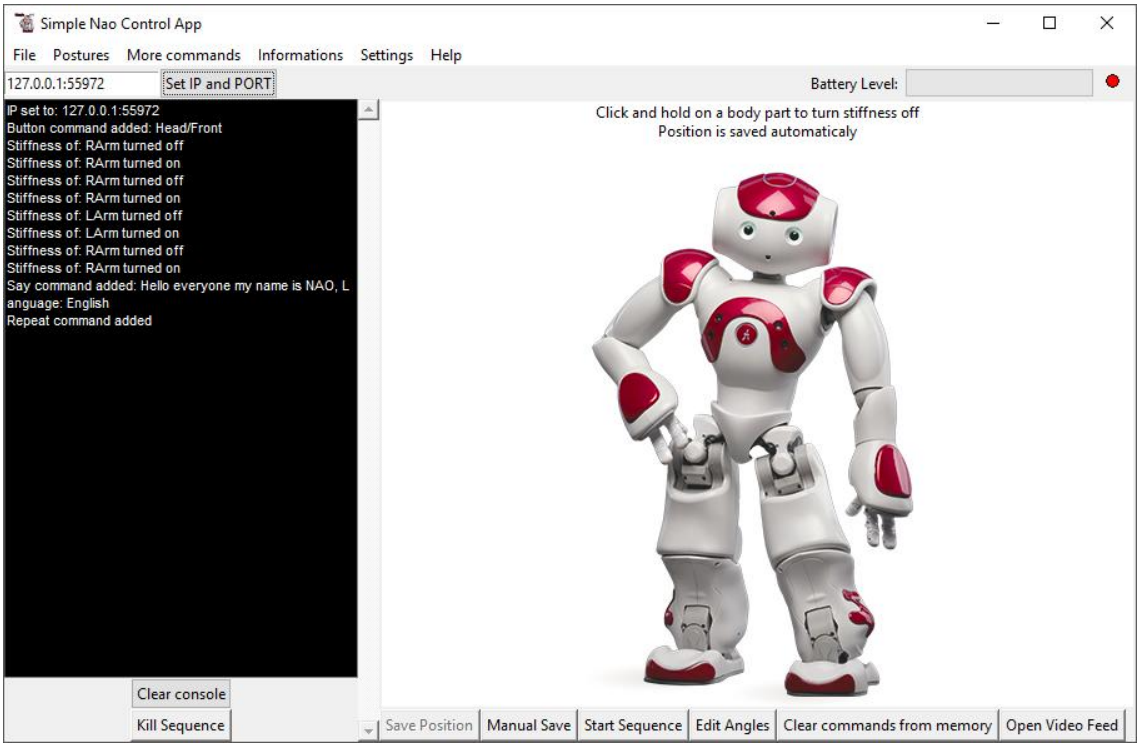
První verze byla založena na ovládání pomocí tlačítek příslušných kloubů [viz Obrázek 8]. Při kliknutí se kloub uvolnil a část těla robota se nastavila do požadované polohy. Poté se musela část těla stále držet v poloze a kliknout znovu na tlačítko, aby se

kloub zase zpevnil. Tento způsob byl později zavržen, protože programování bylo pomalé, a také náročné na ovládání, pokud s robotem pracoval pouze sám uživatel. Tento způsob ovládání je ale stále v programu dostupný, protože umožňuje i uvolnění kloubu pouze v jednom směru. Toto lze využít při potřebě přesně nastavit polohu části těla.



OBRÁZEK 8 - POKROČILÝ MÓD OVLÁDÁNÍ KLOUBŮ

Druhý způsob, který je implementovaný ve zhotoveném programu jako základní, je přizpůsobený pro rychlejší a intuitivnější ovládání. Na hlavní obrazovce [viz Obrázek 9] se nachází obrázek robota. Po dobu stisknutí tlačítka na příslušnou část těla robota se klouby uvolní a graficky se znázorní oblast uvolnění. Po puštění tlačítka myší je do kloubu opět přiveden proud, a proto drží polohu. V základním režimu je také nová poloha automaticky uložena.



OBRÁZEK 9 - PODOBA GRAFICKÉHO ROZHRANÍ

K dispozici je také možnost manuálního ukládání polohy. V případě kdy uživatel chce, aby robot vykonával několik pohybů najednou, jako například pohyboval oběma rukama, je nutné přejít do manuálního režimu ukládání. Po stisknutí tlačítka aplikace funguje stejně jako v předešlém případě, ale po nastavení kloubů do požadované polohy musí uživatel stisknout tlačítko Save Postition, které uloží pozici do paměti.

3.2.1 VÝVOJ APLIKACE

Aplikace byla vyvíjena v jazyce Python a to hlavě z důvodu jednoduché implementace a široké podpory tohoto programovacího jazyku jak výrobcem, tak uživateli. K vývoji stačí stáhnout dodávané vývojové nástroje tzv. SDK, které obsahuje všechny potřebné knihovny pro začátek. Tyto knihovny potom využijeme pro připojení, ovládání i zjišťování informací o robotovi.

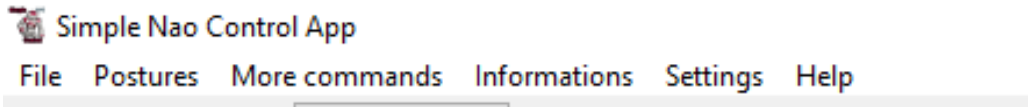
3.2.2 PŘIPOJENÍ K ROBOTU

Pro připojení je nutné vyplnit pole s IP adresou robota. Adresu je možné zjistit zmáčknutím tlačítka na hrudi robota. Port reálného robota je standardně 9559. Po stisknutí tlačítka Set IP and PORT, by se měla v konzoli na levé straně obrazovky zobrazit hláška: „IP set to: [IP adresa]:[Port]“. Toto znamená úspěšné spojení s robotem. Při ne-

úspěšném pokusu se zobrazí chybová hláška. Po připojení je možné začít práci s robotem.

3.2.3 OVLÁDÁNÍ KLOUBŮ ROBOTA

Ovládání kloubů je jedna z nejdůležitějších částí programu. Po zapnutí aplikace se zobrazí obrazovka [viz Obrázek 9], kde je hlavní část věnována právě ovládání kloubů. Zmáčknutím a držením tlačítka myši na části těla se povolí klouby robota, aby s nimi bylo možné pohybovat. Během několika kliknutí se robot dokáže naučit nové pohyby, které by se jinak musely programovat ručně buď pomocí vývojového prostředí Choregraphe, nebo pomocí jednoho z dostupných programovacích jazyků. Po uvolnění tlačítka myši je poloha automaticky uložena. Proto je tvoření sekvencí pohybu velmi rychlé a intuitivní.



OBRÁZEK 10 - MENU DOSTUPNÝCH FUNKCÍ V APLIKACI

3.2.4 MENU PŘEDEM DEFINOVANÝCH POLOH

V menu [viz Obrázek 10] Postures je na výběr několik předem definovaných poloh robota. Pro rychlejší programování byla vytvořena dvě podmenu. První s názvem Save postures robota uvede do dané polohy a následně polohu uloží. Naopak po zvolení polohy v podmenu Go to postures robot přejde do zvolené polohy, ale ta není uložena. Například při začátku programu, pokud uživatel chce robota uvést do jiné polohy, než je výchozí. Na výběr je z polohy Stand, Sit, Crouch a Rest. Robot se tedy postaví, sedne si, přikrčí, anebo se přikrčí a uvede se do úsporného režimu, ve kterém se šetří baterie, ale také se díky vypnutí kloubů se robot nezahřívá.

3.2.5 MENU DALŠÍCH PŘÍKAZŮ

V menu More commands jsou k dispozici některé pokročilejší funkce robota jako chůze, převod textu na řeč, čekání na stisk tlačítka a také sledování objektu, anebo čekání na rozpoznání určitého slova před vykonáváním zbytku programu.

3.2.6 PŘÍKAZ PRO CHŮZI

Po zvolení této funkce se otevře okno pro vkládání parametrů chůze. Zde se zadají parametry jako vzdálenost, kterou má robot ujít a to v ose X a Y. Dále se musí zadat úhel natočení chůze. Poté je na výběr možnost příkaz chůze uložit, nebo provést. Pokud uživatel stiskne tlačítko Walk, robot se vydá v zadaném směru. Pokud chce však chůzi uložit, musí stisknout tlačítko Save.

3.2.7 PŘEVOD TEXTU NA ŘEČ

Díky přítomnosti enginu pro převodu textu na řeč a přítomnosti dvou reproduktorů může robot předčítat uživatelem zadaný text. Na výběr je možnost zadat text ručně pomocí podmenu Say text. Zde je na výběr jazyk čtení a pole pro zadání textu. Výrobce dodává několik jazyků pro převod textu na řeč, ale v robotovi mohou být nainstalovány pouze dva. Seznam dostupných jazyků je z robota stažen, a podle toho je možnost výběru v aplikaci.

Další možnost je předčítat ze souboru. Po kliknutí na Read text from file se otevře okno s výběrem souboru. Nejprve uživatel zvolí, ve které složce na disku se soubory nachází. Dále vybere soubor a ten se zobrazí v pravé části okna. Aplikace také umožňuje, aby při předčítání textu robot vykonával další příkazy v sekvenci, nebo aby robot vyčkal, než dočte celý text, před vykonáváním zbytku příkazů.

3.2.8 ČEKÁNÍ NA STISK TLAČÍTKA

Tato funkce je důležitá pro interaktivní ovládání robota. Po otevření okna uživatel dostane možnost nejprve vybrat část těla, na které se tlačítko nachází. Po výběru části těůa se vyplní zbytek možností pro výběr senzorů.

Vykonávání sekvence příkazů bude čekat do doby, než tlačítko uživatel stiskne. Tímto způsobem můžeme vytvořit několik sekvencí za sebou, kde po vykonání jedné bude robot čekat na stisk tlačítka před vykonáním další sekvence.

3.2.9 OPAKOVÁNÍ SEKVENCE

V menu dalších příkazů nalezneme také funkci opakování. Tuto funkci můžeme využít, pokud chceme, aby se sekvence vykonávala do nekonečna bez nutnosti ručního spuštění. Sekvence pak bude probíhat do doby, než ji ručně vypneme, nebo nedojde k restartu robota. Případné odpojení od robota však sekvenci také přeruší.

3.2.10 ČEKÁNÍ NA DETEKOVANÝ OBLIČEJ

Po vložení tohoto příkazu, robot přestane vykonávat sekvenci do doby, než dojde k úspěšné detekci obličeje člověka. Společně pak s funkcí opakování můžeme například spustit sekvenci vždy, když se před robota někdo postaví.

3.2.11 ČEKÁNÍ NA HLASOVÝ POVEL PŘED POKRAČOVÁNÍM

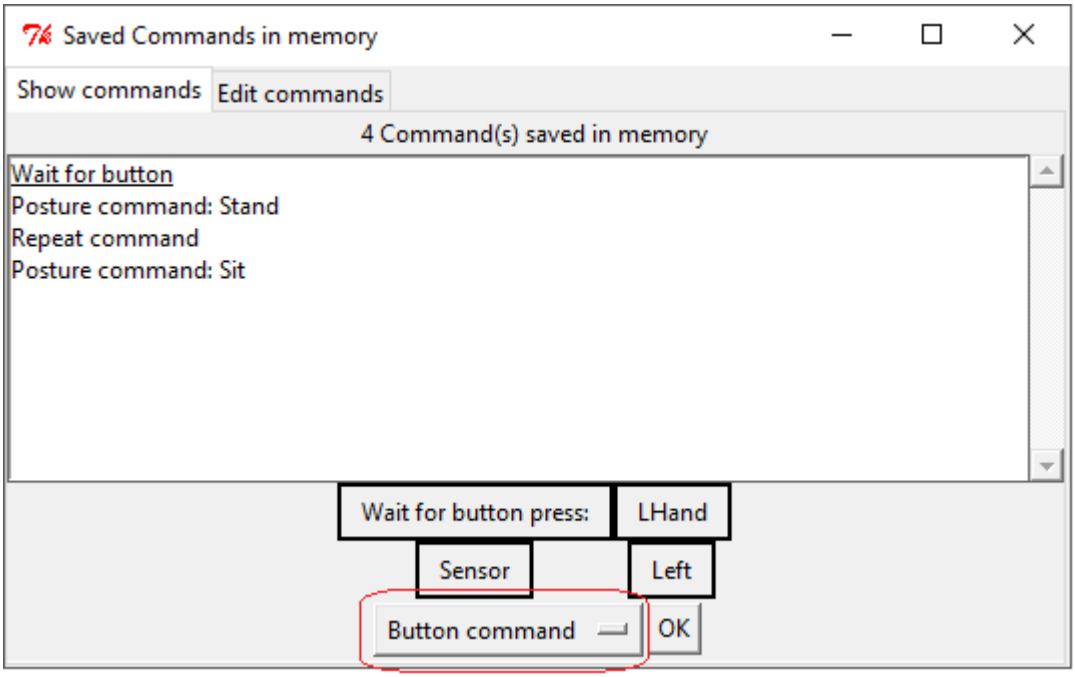
Pokud uživatel vloží do sekvence tento příkaz, robot zastaví vykonávání sekvence v daném místě, dokud uživatel nevysloví zadané slovo. Robot je však schopný rozpoznat pouze slova a ne celé věty. Rozpoznání však funguje i z mluvené řeči. Robot dokáže ve větě slova oddělit a porovná jednotlivá slova s přednastaveným. Pokud nastane shoda, robot bude pokračovat ve vykonávání sekvence. Při neúspěchu bude čekat na další větu.

3.2.12 POKROČILE FUNKCE

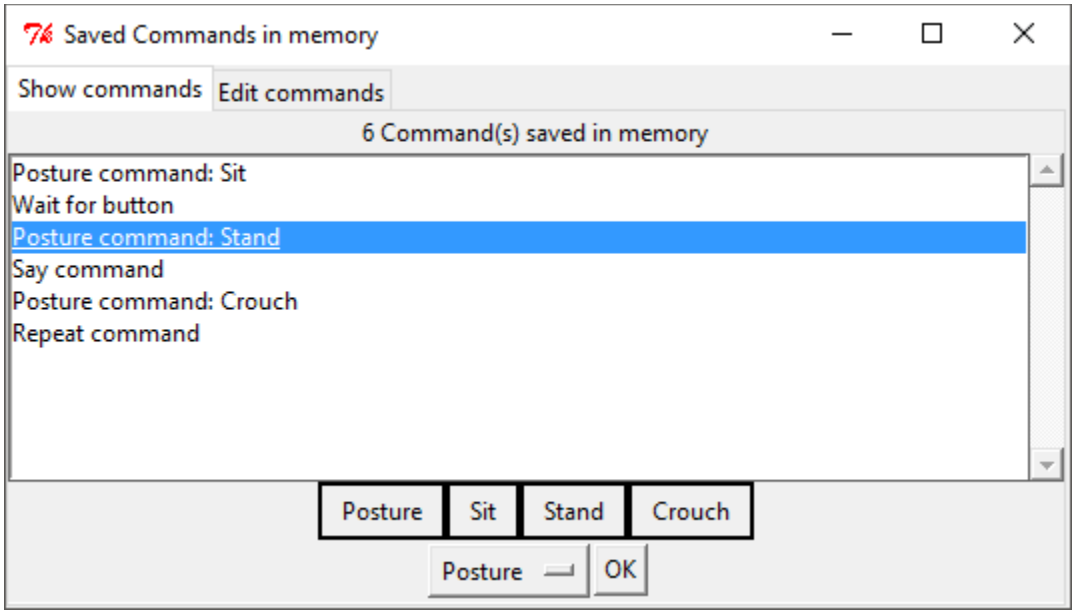
Z pokročilých funkcí bylo vybráno sledování obličeje, protože je to dobrá ukázka schopností robota na prezentacích. Je na výběr, jestli má robot sledovat obličej pouze po určitou dobu nebo do stisku tlačítka na robotu. Také je možné zadat, jak dlouho bude robot obličej sledovat. Pokud necháme v poli hodnotu 0, znamená to, že robot bude sledovat obličej po neomezenou dobu. Sledování se ukončí stisknutím vybraného tlačítka. Na výběr je také možnost, že robot sleduje člověka a přitom se pohybuje celým tělem, anebo pouze hlavou.

3.2.13 MENU ZOBRAZENÍ A UPRAVENÍ SEKVENCE

V nabídce Information se nachází možnost úpravy a zobrazení právě vytvořené sekvence pohybů. Uživatel zde nalezne výpis všech vložených příkazů. Po výběru příkazu z nabídky [viz Obrázek 11] dostane uživatel další informace o daném příkazu. Pokud je příkaz v sekvenci několikrát, zobrazí se všechny výskyty [viz Obrázek 12]. V další záložce je možné jednotlivé příkazy mazat, přesouvat, nebo vymazat celou paměť příkazů.



OBRÁZEK 11 - ÚPRAVA A VÝPIS PŘÍKAZŮ V PAMĚTI



OBRÁZEK 12 - UKÁZKA NĚKOLIKA PŘÍKAZŮ STEJNÉHO TYPU

3.2.14 INFORMACE O ROBOTOVI

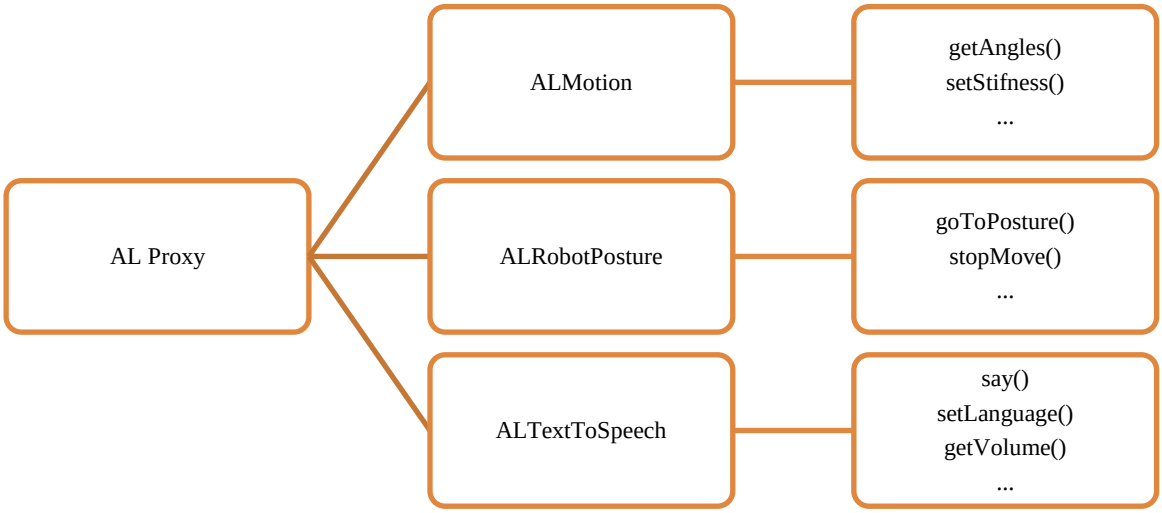
V záložce informace jsou vypsány hlavně teploty jednotlivých motorů robota. Při delší práci s robotem, kdy se s ním provádí pohybové akce, se stává, že se některé z motorů, hlavně motory kolenních a kyčelních kloubů, začnou přehřívat. Robot tuto skutečnost ohlásí krátkým pípnutím, uživatel se nedozví, o které klouby se jedná. Proto je zde k dispozici kompletní výpis teplot.

3.3 DŮLEŽITÉ ČÁSTI KÓDU APLIKACE

Celá aplikace je realizována v jazyce Python. Veškeré ovládání robota probíhá přes knihovny SDK od výrobce robota. Program je vyvíjen objektově, to znamená, že se skládá z několika tříd. Každá z nich má na starost pouze určitou funkci v programu a obsahuje metody, které tyto funkce umožňují provádět. Tento přístup má výhodu v tom, že třídy lze opakovaně použít a také pokud se v nějaké funkci vyskytne chyba, stačí ji opravit pouze na jednom místě.

3.3.1 POUŽÍVÁNÍ VÝVOJOVÝCH NÁSTROJŮ V JAZYCE PYTHON

Veškeré dostupné funkce jsou obsažené v knihovně ALProxy, která je součástí SDK od výrobce. Po importování knihovny do programu je pak možné využívat funkce knihovny. K dispozici je pak několik modulů, v kterých jsou dostupné funkce pro ovládání robota. Struktura pak vypadá jako na obrázku [viz Obrázek 13].



OBRÁZEK 13 - HIEARCHIE STRUKTURY KNIHOVEN

Příklad pro využití modulu ALMotion a jeho funkce setStiffnesses() v jazyce Python vypadá následovně:

```
      from naoqi import ALProxy
      #IP je IP adresa robota a PORT jeho port. IP adresu lze zjistit
při stisku tlačítka na hrudi robota a port je standartně 9559.

      motionProxy = ALProxy("ALMotion",IP,PORT)
```

V tuto chvíli jsou k dispozici, přes tečkovou notaci, všechny funkce modulu ALMotion. Například pro nastavení kloubů do definované polohy lze potom použít

```
      names =
      ["RShoulderPitch","LShoulderPitch","RShoulderRoll","LShoulderRol
l"]
      angles = [9.7*almath.TO_RAD,9.7*almath.TO_RAD,-
2.1*almath.TO_RAD,2.1*almath.TO_RAD]
      fractionMaxSpeed = 0.1
      motionProxy.setAngles(names,angles,fractionMaxSpeed)
```

Nejprve musíme definovat pole motorů, které chceme ovládat. Dále musíme definovat úhly, na které chceme jednotlivé motory nastavit, a také rychlost pohybu. Pak zavoláme funkci setAngles() s těmito parametry a klouby robota se nastaví do požadované polohy. V systému je zabudovaná detekce kolize s jinou částí robota. To znamená, že systém má informace o poloze částí těla v reálném čase a po spuštění příkazu pro nastavení polohu se vybere cesta, při které nedojde ke kolizi. Zároveň systém také hlídá koncové body kloubů, aby nedošlo k poškození robota.

3.3.2 PŘIPOJENÍ K ROBOTOVI

Pro lepší práci s knihovnami a moduly je dobré vytvořit tzv. broker. Ten pak zprostředkovává veškeré požadavky a komunikaci z internetu, nebo z PC po místní síti a umožní, že veškeré moduly jsou schopny nalézt jakoukoli funkci, kterou obsahují. Tím se výrazně zjednoduší práce. Po vytvoření brokeru pak kdekoli v programu můžeme jednotlivé moduly inicializovat a volat jejich funkce. Jednoduché navázání spojení vypadá následovně.

```
      from naoqi import ALBroker
      broker = ALBroker("broker","0.0.0.0", 0, IP,PORT)
```

První argument musí být název proměnné, následuje IP adresa, které broker bude naslouchat a další argument je port naslouchání. V případě zadání adresy 0.0.0.0 a portu 0 bude naslouchat jakékoli adrese na jakémkoli portu. Těmito parametry by šla omezit komunikace robota tak, aby komunikoval pouze s jedním počítačem. Poslední dva argumenty jsou pak IP adresa a port robota.

Po ukončení práce s robotem stačí zavolat příkaz:

```
        broker.shutdown()
```

A veškerá komunikace s robotem bude uzavřena. Také veškeré moduly, co uživatel v programu inicializoval, by se měly automaticky ukončit.

3.3.3 VYTVÁŘENÍ, ČTENÍ A ZPRACOVÁNÍ SEKVENCE

Již při začátku vývoje aplikace bylo nutné rozmyslet, jak se budou povely, které uživatel bude robotovi zadávat, kódovat v paměti. Při zpětném čtení sekvence se pak podle zakódování zjistí, zdali se jedná o příkaz pohybu, mluvení nebo jiné příkazy. Bylo rozhodnuto, že se každému příkazu přiřadí číselná hodnota, takže číslo pak při pozdější práci s příkazem umožní poznat, co má robot vykonávat za činnost a jak naložit se zbytkem obdržených dat.

Sekvence je celá uložená v paměti do proměnné typu seznam. Struktura typu seznam se na tento typ použití hodí mnohem více než například obyčejné pole, hlavně pro jednoduchost přidávání, odebírání a další manipulaci s prvky. Struktura typu pole musí mít na začátku definovanou velikost. Při naplnění pole by se muselo vytvořit pole nové, o nějaký počet prvků větší, a staré pole do něho nakopírovat. Struktura seznamu umožňuje přidávat a odebírat prvky podle potřeby. Tím se práce se sekvencí zjednodušuje a uživatel může tvořit sekvence téměř neomezené velikosti. Jednotlivé příkazy jsou pak uložené také jako seznamy a postupně vkládány do hlavního seznamu sekvence. Přidávání příkazů do sekvence je otázkou pouze jednoho příkazu, kdy se do globální proměnné `commandList` přidá další příkaz.

```
        def saveSay(self,text, language):
                    newEntry = ["3", text, language]
                    self.commandList.append(newEntry)
                    Console().writeToConsole("Say command added: "+ text
+ ", Language: " + language + "\n",gui.console)
```

Čtení a zpracování sekvence probíhá tak, že proběhne cyklus, který projde veškeré prvky v seznamu. Díky nepřítomnosti struktury typu switch nebo case je použito `if-elif-elif-elif…` struktury. Rozklíčování probíhá následovně.

Nejprve projdeme všechny záznamy v seznamu s příkazy uživatele. Při tvorbě sekvence je daná struktura příkazu tak, že na první pozici je vždy číslo, které vyjadřuje o jaký příkaz se jedná. Na dalších místech v seznamu jsou pak doplňující data k příkazu. V případě příkazu pro pohyb kloubu (na prvním místě nula) do dané polohy jsou na

dalších místech úhly natočení kloubů robota. Pokud je číslo příkazu například tři, znamená to, že se jedná o příkaz převodu textu na řeč. Tento příkaz má na dalších místech v seznamu data jako jazyk, ve kterém je text napsaný a také text, který má robot říci.

```
      #klic - 0=pozice kloubu, 1=predem definovane polohy, 2=chuze,
3=tts, 4=cekani na stisk,5=tts(soubor),8=sledovani obliceje…

      for command in commandList:
      if str(command[0]) =="0":
                  Console().writeToConsole(command[1] + " position
\n",gui.console)
                  self.goToPosition(command[2:len(command)],gui)
      elif str(command[0]) == "1":
                  try:
                        if str(command[1]) != "Rest":
                              Console().writeToConsole("going to
posture: " + command[1] + "\n",gui.console)

      Self.postureProxy.goToPosture(command[1],1.0)

      else:
                              self.motionProxy.rest()
                  except NameError:
                        Console().writeToConsole("It seems that there
is no connection to NAO \n")

      elif str(command[0]) == "2":
      …
```

Přiřazováním číselných kódů pro příkazy je možné rychle přidávat další funkce do programu. V programu stačí doplnit metodu, která bude daný příkaz zapisovat do seznamu a do struktury `if-elif-elif-elif…` doplnit, jak data z uložené položky zpracovat.

3.3.4 VYTVÁŘENÍ REAKCE NA UDÁLOSTI

Aby byla možná funkce, kdy program čeká na stisknutí tlačítka uživatelem, nebo například na detekci obličeje, je potřeba mít schopnost reagovat na události vytvořené robotem. Tyto funkce umožňuje například oddělit několik sekvencí od sebe, bez nutnosti nahrávání jiné do paměti.

Aby bylo možné reagovat například na stisk tlačítka, potřebujeme nejprve vytvořit modul definující, na kterou událost budeme reagovat. V tomto modulu se musí v paměti robota pomocí funkce subscribeToEvent definovat jaký modul, a která funkce se při výskytu události má zavolat. První argument je název události. Můžeme reagovat na stisk dotykových tlačítek, na úspěšnou detekci obličejů, ale například i na vlastní udá-

lost, kterou můžeme kdykoli v programu vytvořit. Další argumenty jsou potom název modulu a funkce tohoto modulu, která se má zavolat při vyvolání.

```
memory = None
class TouchChangedModule(ALModule):
gui = None
sensor = None
def __init__(self, name):
            ALModule.__init__(self, name)
            global memory
            memory = ALProxy("ALMemory")
            memory.subscribeToEvent("TouchChanged",
              "TouchChanged",
              "onTouched")


def onTouched(self, strVarName, value):
            memory.unsubscribeToEvent("TouchChanged",
            "TouchChanged")
            print "touched"
            touched_bodies = []
            if value != None:
                  for p in value:
                        if p[1]:
                              touched_bodies.append(p[0])
                  for b in touched_bodies:
                        print b
                        if b==self.sensor:
                              self.gui.continueSequence = True
                              return
            memory.subscribeToEvent("TouchChanged",
            "TouchChanged",
            "onTouched")
```

Z příkladu je vidět, že funkce onTouched má několik argumentů. Mezi nimi je nejdůležitější argument value. V něm jsou uloženy názvy všech stisknutých senzorů. Proto pokud není hodnota rovna None, projdou se všechny prvky v této proměnné a porovnají se, zda se hodnota rovná s požadovaným senzorem, který uživatel určil. V průběhu je však potřeba, abychom z paměti odstranili vazbu na tento modul, aby nedocházelo k několikanásobným detekcím. Pokud je rozpoznán stisk stejného senzoru, který uživatel zvolil, tak není potřeba modul znovu do paměti zavádět. V opačném případě se znovu zavolá metoda subscribeToEvent() a čeká se na další stisk tlačítka.

Podobným způsobem je realizována většina modulů, které reagují na událost v paměti robota. V aplikaci se podobný modul využívá také třeba pro reagování na detekci obličeje, ale také pro detekci rozpoznání slova.

3.3.5 ZOBRAZENÍ OBRAZU Z KAMERY ROBOTA

Aby bylo možné monitorovat robota na dálku, je v aplikaci přidaná funkce k zobrazení okna, ve kterém se nachází živý přenos z kamery robota. Robot je však schopen přenášet pouze obraz v rozlišení 320x240 pixelů. Vzorkovací frekvence je maximálně 15 snímků za sekundu. V programu je nastavena frekvence 10 FPS, protože obraz zde slouží pouze pro kontrolu. Při přenosu na větší vzdálenost se může stát, že nebude dosaženo ani 10 FPS, ale frekvence bude ještě nižší.

Implementace do grafického rozhraní Tkinter je pomocí widgetu typu Label, na který je vykreslen každých 50ms obrázek stažený z robota. Operační systém robota totiž neumožňuje vysílat souvislý tok dat jako video. Proto se musí periodicky obraz z robota stahovat. Zároveň Tkinter nedisponuje žádným widgetem pro zobrazení videa, proto byl zvolen přístup překreslování Labelu, ve kterém je vložen obrázek. V Pythonu je toho docíleno následujícím způsobem:

```
      import sys
      from naoqi import ALProxy
      from ttk import *
      import Tkinter as tk
      from Tkinter import *
      from PIL import Image, ImageTk
      import os
      import numpy as np
      class VideoShow():
            def __init__(self,window,master):
                  self.window = window
                  self.master = master
                  self.label = tk.Label(master = window)
                  self.label.pack()
                  window.title("Live video from NAO")
                  try:
                        self.camProxy = ALProxy("ALVideoDevice")
                  except Exception as e:
                      print e
                  resolution = 2    # VGA
                  colorSpace = 11   # RGB
                  self.videoClient = self.camProxy.subscribe("_client",
resolution, colorSpace, 5)
                  self.show_vid()
```

Nejprve se načtou všechny potřebné knihovny pro práci a stažení obrazu z robota. Dále se vytvoří třída, která bude vykreslovat obrázky. Definují se základní proměnné jako okno, ve kterém se vytvoří Label pro obrázek, a změní se název okna. Dále se zavolá funkce show_vid(), která se bude periodicky po 10ms opakovat, a tím se vytvoří efekt videa.

```
            def show_vid(self):
                  frame = self.getImage()
                  imgtk = ImageTk.PhotoImage(image=frame)
                  self.label.imgtk = imgtk
                  self.label.configure(image=imgtk)
                  self.master.update_idletasks()
                  self.window.after(50, self.show_vid)
```

V této funkci se jako první pokusí program získat obraz z robota funkcí getImage(). Dále se provedou potřebné úpravy obrazu, aby bylo možné ho zobrazit v okně. Nakonec se zavolá funkce after(), která je dostupná v knihovnách Tkinter, způsobující periodické zavolání funkce show_vid() každých 50ms.

```
            def getImage(self):
                  naoImage =
self.camProxy.getImageRemote(self.videoClient)
                  imageWidth = naoImage[0]
                  imageHeight = naoImage[1]
                  array = naoImage[6]
                  im = Image.frombytes("RGB", (imageWidth,
imageHeight), array)
                  return im
```

Funkce getImage() se pokusí získat aktuální obraz z robota. Při komunikaci přes Wifi síť se může stát, že se naskytne chyba v přenosu a obraz program neobdrží. Obraz z robota se přenáší jako pole ASCII znaků. Aby bylo možné obraz zobrazit v knihovnách Tkinter, je potřeba ho následně převést na RGB obrázek. K tomu je využita knihonva PIL (Python Imaging Library), která umožní zpracovávat obrázky v jazyku Python.

Dále se také ve třídě nachází metoda onExit(), které při ukončení okna zajistí přerušení komunikace s robotem a vymazání proxy služby z paměti robota

```
      def onExit(self):
                  self.camProxy.unsubscribe(self.videoClient)
```

3.3.6 UVOLŇOVÁNÍ KLOUBŮ ROBOTA PŘI KLIKNUTÍ NA ČÁST TĚLA

Tato aplikace byla vyvíjena hlavně pro funkci uvolňování kloubů robota. Umožňuje uživateli rychle naučit robota pohybům, které by jinak musel uživatel programovat ručně. Tato funkce však přináší při vývoji aplikace několik problémů.

První byl spojen s ovládáním robota. Pro intuitivnost byl zvolen přístup, kdy uživatel bude klikat na obrázek robota pro uvolnění kloubů. Aby bylo možné rozeznat, o jakou část robota se jedná, bylo nutné mít vždy vycentrovaný obrázek robota na obrazovce uživatele. V knihovnách Tkinter není možnost u prvku zvolit, že má být vycentrován. V programu je centrování vyřešeno tak, že se při každé změně velikosti okna přepočítají souřadnice, na kterých je obrázek robota vykreslen. Obrázek je vykreslen na prvek typu Canvas. Tento prvek je možné využít pro vykreslení čar, obrázků, ale i ostatních prvků jako jsou tlačítka, vkládací pole a další. Nejprve musíme v programu definovat, že chceme vědět o události při zvětšení okna.

```
        self.canvas.bind("<Configure>", self.resize)
```

Tímto bylo definováno, že při změně velikosti okna se zavolá metoda resize(). Definice metody v jazyce Python probíhá následovně:

```
def resize(self,event):
        self.canvas.coords(self.image,event.width/2-
self.imgNao.width()/2,50)
        box = self.canvas.bbox(self.image)
        imgWidth = box[2] - box[0]
        imgheight = box[3] - box[1]
        Rect = namedtuple('Type', 'x0, y0, x1, y1')
        x0,y0 = self.canvas.coords(self.image)
        img_rects = [Rect(x0+100, y0, x0+220, y0+100), #Head
        Rect(x0+16, y0+70, x0+80, y0+250), #LHand
        Rect(x0+imgWidth-90, y0+90, x0+imgWidth,y0+300), #RHand
        Rect(x0+40, y0+230, x0+130, y0+imgheight-10), #LLeg
        Rect(x0+imgWidth-150, y0+230, x0+imgWidth-40, y0+imgheight-10)]
#RLeg
        self.imagemapper = ImageMapper(self.imgNao, img_rects)

        self.canvas.coords(self.text1,(event.width/2),10)
                    self.canvas.coords(self.text2,(event.width/2),25)
```

Při změně velikosti se posune obrázek robota na souřadnici x vypočítanou jako polovina velikosti plátna, na kterém obrázek vykreslujeme. Od této hodnoty se odečte polovina velikosti obrázku. Souřadnice y je konstantní.

Po stisku a držení tlačítka myší na části robota, kterou chce uživatel povolit, se tato část zvýrazní. Zvýraznění a rozpoznání o jakou část se jedná je vyřešeno tak, že se obrázek rozdělí na několik obdélníků. Tyto obdélníky pak odpovídají jednotlivým částem těla. Při kliknutí na obrázek se vyvolá událost, ve které se zjistí, na které souřadnice uživatel kliknul, a kterému obdélníku, respektive části těla, tato oblast patří. Po zjištění, o kterou část těla se jedná, vypneme pro tuto část přívod elektrického proudu do kloubů, a tím kloub uvolníme. Ten zůstane uvolněn až do puštění tlačítka myši.

To znamená, že při každém překreslení obrázku, kvůli změně velikosti okna, se také musí přepočítat souřadnice jednotlivých částí těla robota. Proto musí být souřadnice obdélníku ohraničující části těla počítány jako relativní souřadnice vůči obrázku. Přepočet souřadnic je vidět ve zdrojovém kódu uvedeném výše. Nejprve se zjistí poloha nově vloženého obrázku označená jako x0, y0. Tyto souřadnice jsou pro pravý horní roh obrázku. Dále se vytvoří proměnná img_rects, která bude obsahovat všechny obdélníky označující části robota. Tyto obdélníky jsou definovány pomocí levého horního a pravého spodního rohu. Poté se zavolá funkce, která tyto definované obdélníky přeloží přes obrázek robota, a na závěr se posunou i nápisy pro nápovědu k ovládání aplikace.

Dále je také definovaná nejmenší velikost okna, kdy jsou všechny prvky aplikace viditelné. Zároveň je tato velikost zvolena tak, aby bylo možné ovládat všechny části robota.

3.3.7 PŘEDÁVÁNÍ INFORMACÍ MEZI VLÁKNY APLIKACE

Aplikace je vytvářená tak, aby uměla využít více procesorových jader. Bylo rozhodnuto použít více vláken hlavně kvůli periodické kontrole a vypisování informací o robotovi. Aby toto čtení informací z paměti robota zbytečně nezpomalovalo běh programu, vytvořilo se druhé vlákno aplikace. Jak je již řečeno dříve (viz Kapitola 2.2.2), celé grafické rozhraní běží v nekonečné smyčce. Při některých příkazech je nutné, aby program vyčkával, dokud nenastane nějaká událost. Aby nepřestalo grafické rozhraní reagovat, musí se toto čekání provádět v jiném vlákně.

Aby bylo možné si předávat informace z jednoho vlákna do druhého, musela se implementovat v Pythonu knihovna Fronty [1]. Pokud obě vlákna pracují a potřebujeme z každého číst nebo měnit stejnou proměnnou, mohl by nastat problém. Kdyby se stalo, že na jedno místo v paměti, tzn. do jedné proměnné, by se zapisovalo z více míst ve stejnou chvíli, pravděpodobně by data v proměnné byla nepoužitelná, a proto je nutné vlákna nějak synchronizovat. Toho lze dosáhnout několika způsoby. Může se vytvořit událost pro sledování a změnu proměnné, využít objektu typu Lock, který dokáže uzamknout proměnnou pro jedno vlákno, anebo třeba objekt typu Semaphore. Pokud je však potřeba předat pouze nějaké informace, postačí právě objekt typu Queue neboli Fronta. Třída Fronty obsahuje všechny potřebné mechanizmy pro zamknutí jednotlivých proměnných a vláken při zápisu do fronty.

Frontu je možné vytvořit jako FIFO, LIFO nebo Priority. Ve frontě FIFO (First In First Out) je prvek, který byl přidán jako první, také jako první vyjmut. Tuto frontu si lze představit jako klasický zásobník. Naopak ve frontě LIFO (Last In First Out) je prvek, který byl přidán jako poslední vyjmutý jako první. V posledním typu zadává programátor mimo dat také číselnou hodnotu označující prioritu dat. Celá fronta je potom udržována seřazená právě podle této hodnoty priority. V případě této aplikace byla využita klasická fronta v podobě FIFO. Jelikož se předává málo typů, a malé množství dat, bohatě postačí tento typ. Proměnnou, která označuje instanci této třídy, se musí předat všem vytvářeným vláknům, mezi kterými chce vývojář předávat informace.

Pro obsluhu fronty stačí pouze několik příkazů. V aplikaci vytvoření a používání třídy vypadá následovně.

```
      import Queue

      self.queue = Queue.Queue()
      self.gui = GUI(master, self.queue, self.endApplication)

      self.periodicCall(self.queue)
```

Nejprve importujeme knihovny Fronty. Následně se fronta vytvoří. Jak již bylo zmíněno, tato fronta bude typu FIFO. Pokud by měla být vytvořená fronta jiného typu, pak by kód vypadal následovně `self.queue = Queue.LifoQueue()` nebo `Queue.PriorityQueue()`. V dalším řádku se předají potřebné proměnné třídě pro tvorbu grafického rozhraní. Zde se musí předat právě proměnná označující frontu. Dále se zavolá metoda pro periodické zjišťování informací, v našem případě stavu baterie.

```
def periodicCall(self,queue):
      batteryData = self.gui.RobotCommands.batteryLevel()
      if batteryData != None:
            batteryData.insert(0,"batteryData")
            queue.put(batteryData)
            self.gui.processIncoming()
            if not self.running:
                  import sys
                  sys.exit(1)
      self.master.after(5000, self.periodicCall,queue)
```

V této metodě se nejprve z robota vyčtou data obsahující informace o baterii, dále pokud se podařila nějaká data načíst, přidá se na první místo informace, že se jedná o data z baterie, a nakonec se příkazem put data vloží do fronty. Poté se zavolá funkce processIncoming() která se stará právě o obsluhu fronty. Na konci metody periodicCall() ještě využijeme funkce knihovny Tkinter, kdy je možné periodicky volat metodu

po určité době, funkcí after(). Tím se zaručí, že se tato metoda bude vykonávat každých pět sekund.

```
    def processIncoming(self):
        while self.queue.qsize(  ):
            try:
                msg = self.queue.get()
                if msg[0] == "batteryData":
                    self.batteryBarUpdate(msg)
                elif msg[0] == "continueSequence":
                    self.continueSequence = msg[1]
            except Queue.Empty:
                pass
```

Toto je metoda pro obsluhu fronty. Nejprve se zkontroluje, jestli je ve frontě nějaký prvek. Funkce qsize() však nemůže zaručit to, že zrovna tuto frontu neblokuje jiné vlákno. Proto pokud vrátí funkce qsize() hodnotu > 0, tak obecně nemůže program spoléhat na to, že ve frontě nějaký prvek je. Mezitím ho totiž mohlo jiné vlákno příkazem get() vyjmout. Pro zamezení pádu programu je zde použito struktury try-except, která pádu zabrání.

Obsluha třídy se pak skládá pouze z vyjmutí dat příkazem get() a podle dříve vloženého klíče na první místo zprávy, se pozná, o jaký typ dat se jedná. Podle typu je pak zavolána příslušná metoda. Třída se díky while cyklu vždy celá vyprázdní.

3.3.8 ZMĚNA KÓDOVÁNÍ PŘI ČTENÍ TEXTU V ČESKÉM JAZYCE

Robot je schopný předčítat text v několika jazycích. Mezi tyto jazyky patří i český jazyk, ale pokud se zadává text, který obsahuje diakritiku, nastává problém při čtení. Pokud se uloží v Pythonu do proměnné text s diakritikou a následně se předá modulu pro převod textu na řeč, modul tento text převezme, nezahlásí žádnou chybu, ale robot text nepřečte. Z toho se usoudilo, že je problém v kódování textu. I po přidání následujících řádků na začátek programu, které by měly oznámit používání kódování UTF-8 v aplikaci, robot stále text s diakritikou nepředčítal.

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*
```

V programu bylo nutné kódování změnit ručně. Muselo se změnit standardní kódování celého programu. Toho bylo docíleno využitím knihovny sys, ve které nalezneme parametry a funkce svázané s překladačem. V této knihovně se nachází funkce pro změnu kódování. Při začátku programu se musí kódování změnit následujícími příkazy:

```
Import sys
reload(sys)
sys.setdefaultencoding('utf-8')
```

Potom už převod textu na řeč bude probíhat bez problémů, protože veškerý text vyskytující se v aplikaci, bude zakódován ve stejném formátu.

3.4 VYTVOŘENÍ SPUSTITELNÉHO SOUBORU

Díky velké popularitě programovacího jazyka Python, jsou k dispozici nástroje pro převod kódu do spustitelného souboru pro operační systém Windows. Pomocí těchto nástrojů je možné vytvářet .exe soubor a veškeré potřebné soubory a knihovny pro běh aplikace. Uživatel tak nemusí mít nainstalovanou žádnou z doplňujících knihoven, SDK ani překladač Pythonu. Stačí, aby obdržel spustitelný soubor spolu se všemi potřebnými soubory, a může začít pracovat s programem. Spustitelný soubor je vytvářen pro větší pohodlí při používání, právě proto, aby uživatel nemusel instalovat žádné doplňující programy.

3.4.1 MOŽNOSTI PŘEVODU

Pro převod se nabízí několik volně stažitelných nástrojů. Mezi ně patří například py2exe jak pro Python ve verzi 2.6, tak pro 3.3-3.5, pyinstaller také pro obě verze, nebo nástroj, který byl pro tuto práci použit, cx_Freeze.

3.4.2 PŘEVOD KÓDU NA SPUSTITELNÝ SOUBOR

Pro převod je nejprve nutné vytvořit soubor setup.py, ve kterém jsou informace o projektu, a doplňující pravidla k převodu kódu. Tento soubor by měl být umístěn ve stejné složce jako vytvořený program. K dispozici jsou předpřipravené setupy, ale pokud je potřebné zahrnout i další soubory jako obrázky či ikony, musí se vytvořit vlastní soubor.

Nejprve je potřeba nahrát knihovnu cx_Freeze a také knihovnu sys, která obsahuje systémové proměnné.

```
from cx_Freeze import setup, Executable
import sys
```

Dále musíme zadat cestu k potřebným souborům v programu. V případě této práce jsou potřeba všechny soubory ve složce img. Proto uvedeme celou složku. Cestu je třeba zadávat jako relativní vůči souboru setup.py. Dále se uvede jméno projektu.

```
      buildOptions = dict(include_files = ['img/']) #folder,relative
path.
      productName = "NAO Simple Control App"
```

Aby fungovala aplikace i pro vytvořené zástupce, musí být definováno, kde se nachází pracovní složka programu, hlavně kvůli obrázkům v aplikaci, ale také například kvůli ikoně aplikace. Proto se musí vytvořit tabulka, podle pravidel pro vytváření zástupců [9]

```
      shortcut_table = [
          ("DesktopShortcut",        # Shortcut
           "DesktopFolder",          # Directory_
           "NAO Control App",        # Name
           "TARGETDIR",              # Component_
           "[TARGETDIR]NAO Simple Control.exe",# Target
           None,                     # Arguments
           None,                     # Description
           None,                     # Hotkey
           None,                     # Icon
           None,                     # IconIndex
           None,                     # ShowCmd
           'TARGETDIR'               # WkDir
           )
          ]
      msi_data = {"Shortcut": shortcut_table}
      bdist_msi_options = {'data': msi_data}
      opt = dict(build_exe = buildOptions)
```

Pomocí nástrojů cx_Freeze lze vytvářet i instalační průvodce pro systém Microsoft Windows. Při plánování použití této funkce, je dobré definovat základní cestu, do které se program bude instalovat. Ve většině případů se jedná o složku Program Files.

```
      if 'bdist_msi' in sys.argv:
          sys.argv += ['--initial-target-dir', 'C:\Program Files\\' +
productName]
```

Také je potřeba definovat o jaký typ aplikace se jedná. Může to být aplikace s grafickým rozhraním jako v tomto případě, konzolová aplikace, anebo služba. Uvedeme zdrojový soubor s kódem napsaném v Pythonu a poslední parametr je název spustitelného souboru.

```
      exe = Executable(
            script="GUI_Classes.py",
            base="Win32GUI", # app type, None for console app
            targetName="NAO Simple Control.exe"
      icon='img/fav.ico'
           )
```

Na konec se zavolá funkce setup(), které se předají informace o programu jako název, verzi, autora, popis programu a také zvolené soubory a předešle definovanou proměnnou exe

```
      setup(
            name="NAO Simple Control.exe",
            version="1.0",
            author="Vaclav Jise",
            description="Copyright 2017, Vytvořeno pro TUL",
            options = {"bdist_msi":
bdist_msi_options,"build_exe":buildOptions},
            executables=[exe],
            )
```

Zavoláním příkazu `python setup.py build` se vytvoří exe soubor a ostatní doplňkové soubory. Jak bylo již zmíněno, pomocí nástrojů cx_Freeze lze vytvářet i instalační soubory. Pro vytvoření se zavolá místo příkazu build příkaz bdist_msi.

3.5 MOŽNÉ VYLEPŠENÍ PŘI DALŠÍM VYVÍJENÍ APLIKACE

Při pokračování vývoje aplikace by bylo pro vylepšení funkčnosti dobré přidat možnost, aby nemusel být počítač připojený celou dobu k robotovi. Jedno z možných řešení by bylo místo generování sekvence aplikací, generovat přímo kód v Pythonu, který by se pak do robota přenesl a následně také vykonal. Vygeneroval by se tak přímo zdrojový kód pro obsluhu robota. Díky možnosti nastavení automatického spouštění programu při startu robota by pak uživatel robota pouze zapnul a robot by sekvenci začal vykonávat.

Další přístup by byl přenést vytvořený textový soubor se sekvencí do robota, ve kterém by na pozadí běžel program pro dekódování sekvence tak, jak je vytvořený v této aplikaci. Program by na pozadí kontroloval, jestli se nezměnil textový soubor se sekvencí a po změně by soubor načetl a začal sekvenci provádět. V tomto případě by bylo nutné vyřešit, jak se daný soubor se sekvencí přenese.

U obou těchto příkladů by ještě musela být naprogramována možnost, jak celou sekvenci zastavit, a následně způsob jejího opětovného spuštění.

Druhým vylepšením by měla být možnost tvorby větvení vytvořené sekvence. Tato funkce by umožnila tvorbu složitějších programů, a tím by přispěla k zvýšení celkové využitelnosti této aplikace.

Větvení typu If – Than – Else při vytváření by mohlo vést k několika seznamům sekvencí, které by byly uloženy v paměti. Podle vyhodnocení podmínky by se rozhodlo, jaký seznam sekvencí se má vykonávat. Uživatel by pak musel mít možnost vytvářet několik seznamů sekvencí, které by byly volané z hlavního programu.

Mít možnost různé reakce, například na každé z tří tlačítek na hlavě robota, by vedlo k větší kontrole nad robotem. Zároveň by to vedlo k lepším možnostem interakce s člověkem, a také k možnosti mít v robotu nahranou pouze jednu sekvenci, která by díky větvení mohla obsahovat několik variant, jež by robot vykonával.

Aby nedošlo zbytečně ke zkomplikování tvoření sekvencí, program by měl obsahovat možnost, ve které by se větvení nevyskytovalo, a uživatel by tvořil pouze jednu sekvenci za sebou jdoucích příkazů.

4 ZÁVĚR

V této práci byla vytvořena aplikace pro snadnější ovládání robota a jeho programování. Pro vývoj bylo využito SDK od výrobce robota hlavě pro větší kontrolu nad jeho chováním a pohyby. SDK umožní jednoduchým způsobem využívat veškerou funkčnost robota v několika programovacích jazycích. V této práci byl pro vývoj zvolen jazyk Python a jeho grafická nástavba v podobě knihoven Tkinter. Právě využití SDK umožnilo tvorbu vlastní aplikace pro ovládání a programování robota.

Programování robota spočívá ve vytváření sekvencí pohybů a příkazů pro robota. Uživatel pomocí těchto příkazů robota může naučit novým pohybům, a tak pro něho nalézt další využití. Mezi funkce aplikace patří hlavně možnost ovládání kloubů robota tak, že uživatel vybere, kterou část těla robota chce ovládat, robot tuto část povolí, aby uživatel mohl klouby nastavit do požadované polohy. Tato poloha je pak automaticky uložena a přidána do sekvence pohybů. K dispozici je i možnost manuálního ukládání jednotlivých poloh. Další funkce jsou například převod textu na mluvenou řeč, chůze, přechod robota do předdefinovaných poloh, anebo čekání na stisk uživatelem definovaného tlačítka pro pokračování sekvence.

Aplikace se zjednodušeným ovládáním humanoidního robota NAO by měla zvýšit využitelnost robota tam, kde je potřeba rychle vytvářet jednoduché sekvence příkazů a pohybů. Aplikaci by měl být schopen využívat i člověk s malou zkušeností s robotem. Grafické rozhraní je zaměřené hlavně na jednoduchost a intuitivnost ovládání.

Aplikace již byla otestována a použita na veletrhu Ampér v Brně, kde robot sloužil k nalákání a upoutání pozornosti návštěvníků veletrhu. Robot byl v poloze sedu a při rozpoznání obličeje kamerou se zvedl, zamával, při tom pozdravil a řekl několik informací o škole. Poté si znovu sedl a čekal na další obličej. Robot byl velmi úspěšný, upoutal na sebe velkou pozornost a mnoho lidí projevilo velký zájem se dozvědět o něm více informací.

SEZNAM LITERATURY:

[1] 8.10. Queue — A synchronized queue class — Python 2.7.13 documentation. 302 Found [online]. Copyright © [cit. 08. 04. 2017]. Dostupné z: https://docs.python.org/2/library/queue.html#module-Queue

[2] An Introduction to Tkinter (Work in Progress). effbot.org [online]. Dostupné z: http://effbot.org/tkinterbook/

[3] Connectivity — Aldebaran 2.1.4.13 documentation. SoftBank Robotics Documentation [online]. Dostupné z: http://doc.aldebaran.com/2-1/family/robots/connectivity_nao.html

[4] Contact and tactile sensors — Aldebaran 2.1.4.13 documentation. SoftBank Robotics Documentation [online]. Dostupné z: http://doc.aldebaran.com/2-1/family/robots/contact-sensors_robot.html

[5] KISUNG, SEO. Using NAO: Introduction to interactive humanoid robots.

[6] LEDs — Aldebaran 2.1.4.13 documentation. SoftBank Robotics Documentation [online]. Dostupné z: http://doc.aldebaran.com/2-1/family/robots/leds_robot.html

[7] NAO Documentation — Aldebaran 2.1.4.13 documentation. SoftBank Robotics Documentation [online]. Dostupné z: http://doc.aldebaran.com/2-1/home_nao.html

[8] NOVÁK, Petr. Mobilní roboty: pohony, senzory, řízení. 1. vyd. Praha: BEN - technická literatura, 2005. ISBN 80-7300-141-1.

[9] Shortcut Table (Windows). Learn to Develop with Microsoft Developer Network | MSDN [online]. Copyright © Microsoft 2017 [cit. 15. 04. 2017]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/aa371847(v=vs.85).aspx

[10] Webots: screenshots. Webots: robot simulator [online]. Copyright © [cit. 09. 04. 2017]. Dostupné z: https://www.cyberbotics.com/screenshots

A Obsah přiloženého CD

Přiložené CD obsahuje:

- Text bakalářské práce
  - BP_Vaclav_Jise_2017.pdf
- Program pro ovládání robota
  - Instalátor programu
  - Přenositelná verze
  - Zdrojový kód programu (v jazyce Python)