UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A
INFORMATIKY

EMULÁCIA KOŽE V SIMULÁTORE HUMANOIDNÉHO ROBOTA ICUB A AUTONÓMNA EXPLORÁCIA TELA

Diplomová práca

2016 **Bc. Martin Varga**

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A
INFORMATIKY

EMULÁCIA KOŽE V SIMULÁTORE HUMANOIDNÉHO ROBOTA ICUB A AUTONÓMNA EXPLORÁCIA TELA

Diplomová práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	2511 aplikovaná informatika
Školiace pracovisko:	Katedra Aplikovanej Informatiky
Školiteľ:	prof. Ing. Igor Farkaš, Dr.
Konzultant:	Mgr. Matěj Hoffmann, PhD.

Bratislava 2016 **Bc. Martin Varga**

98277565

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Martin Varga
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: aplikovaná informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Emulácia kože v simulátore humanoidného robota iCub a autonómna explorácia tela
Skin emulation in the iCub humanoid robot simulator and autonomous body exploration

Cieľ: Navrhnite a implementujte zvýšenie rozlíšenia pri emulácii kože v simulátore iCub. Navrhnite a otestujte spôsob, akým bude simulovaný robot skúmať svoje telo na základe dotykov svojho vlastného tela.

Literatúra:
Maiolino P., Maggiali M., Cannata G., Metta G. & Natale L. (2013). A flexible and robust large scale capacitive tactile system for robots, IEEE Sensors Journal, 13(10), str. 3910-3917.
Moulin-Frier C., Oudeyer P.-Y. (2013). Exploration strategies in developmental robotics: a unified probabilistic framework. IEEE Third Joint International Conference on Development and Learning and Epigenetic Robotics (ICDL).
Roncone A., Hoffmann M., Pattacini U. & Metta G. (2014), Automatic kinematic chain calibration using artificial skin: self-touch in the iCub humanoid robot. Proceedings of IEEE International Conference on Robotics and Automation, str. 2305-2312.

Anotácia: Obohatenie humanoidných robotov o dotykovú modalitu patrí medzi aktuálne ciele kognitívnej robotiky, ktorý je nutný v procese tvorby telovej schémy robota. Fyzický robot iCub bol vybavený dotykovými senzormi, z čoho vyplynula prirodzená potreba emulovať dotykovú modalitu v simulátore, ktorý je dostupný vedeckej komunite.

Vedúci: prof. Ing. Igor Farkaš, Dr.
Konzultant: Mgr. Matěj Hoffmann, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 02.11.2014

Dátum schválenia: 28.11.2014

prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Čestné vyhlásenie

Čestne prehlasujem, že som túto diplomovú prácu vypracoval samostatne s použitím uvedených zdrojov.

V Bratislave 2.5.2016 .....

Poďakovanie

Ďakujem svojmu vedúcemu práce prof. Ing. Igorovi Farkašovi, Dr. a konzultantovi Mgr. Matějovi Hoffmannovi, PhD. za odborné vedenie a pomoc pri tvorbe tejto diplomovej práce.

Abstrakt

Autor: Bc. Martin Varga
Názov diplomovej práce: Emulácia kože v simulátore humanoidného robota iCub a autonómna explorácia tela
Fakulta: Fakulta matematiky, fyziky a informatiky
Katedra: Katedra aplikovanej informatiky
Vedúci diplomovej práce: prof. Ing. Igor Farkaš, Dr.
Konzultant: Mgr. Matěj Hoffmann, PhD.
Rozsah práce: 55 strán
Bratislava, 2016

Rozvoj robotiky a jej prienik do každodenného života vyžaduje navrhnutie a implementáciu algoritmov zabezpečujúcich bezpečnosť ľudí pracujúcich s týmito robotmi. V tejto práci sa zameriavame na sebapoznanie vlastného tela robotom. Zapojili sme sa do vývoja simulátora iCub humanoidného robota s otvoreným kódom. Robot bol nedávno vybavený celotelovou kožou. Je množina približne 4000 taktilných senzorov. Zjemnili sme emuláciu kože v simulátore a tým zvýšili jej rozlíšenie. Implementovali sme autonómnu exploráciu tela, inšpirovanú predpokladanými mechanizmami vývoja kojencov, ktorá používa varianty náhodného motorického prehľadávania a prehľadávania cieľov.

Kľúčové slová: *iCub simulátor, autonómna explorácia, umelá koža*

Abstract

Author: Bc. Martin Varga
Title: Skin emulation in the iCub humanoid robot simulator and autonomous body exploration
Faculty: Faculty of mathematics, physics and informatics
Department: Department of applied informatics
Supervisor: prof. Ing. Igor Farkaš, Dr.
Consultant: Mgr. Matěj Hoffmann, PhD.
Size: 55 pages
Bratislava, 2016

Progress of robotics and its penetration to everyday life, demands to propose and implement algorithms ensuring safety of human working with these robots. In this work we focus on robots self-awareness of his body. We contributed to the development of the open-source simulator of the iCub humanoid robot. The robot has been recently equipped with a whole-body skin - an array of approximately 4000 tactile sensors. We refined the emulation if the skin in simulator and increased the resolution. We implemented body self-exploration, inspired by the putative mechanisms from infant development, using variants of random motor exploration and random goal exploration.

Keywords: *iCub simulator, autonomous exploration, artificial skin*

Obsah

Kapitola 1

Úvod

Humanoidný robot iCub bol ako prvý na svete vybavený dotykovými senzormi po celom povrchu tela. Táto umelá koža otvára nové možnosti rozvoja robotiky. Na účely testovania a pre širokú vedeckú komunitu bol vytvorený voľne dostupný iCub simulátor. V iCub simulátore bola zatiaľ umelá koža implementovaná len čiastočne.

Táto diplomová práca sa skladá z dvoch častí. Prvou je emulácia kože v iCub simulátore. Navrhli, implementovali a otestovali sme zvýšenie rozlíšenia kože na predlaktiach oproti pôvodnej verzii. Implementácia dostatočného rozlíšenia kože v iCub simulátore sprístupní možnosť experimentovať s dotykovými dátami, a je kľúčová pre druhú časť tejto práce.

Druhou časťou tejto práce je autonómna explorácia tela. S využitím emulácie kože na rukách a predlaktiach v iCub simulátore, a pomocou generovania dvojdotykov ukazováka pravej ruky o plochu ľavého predlaktia a dlane, navrhujeme, implementujeme a testujeme postupy na autonómne vytvorenie senzomotorického modelu. Pomocou senzomotorického modelu vieme navrhovať správne povely podľa dôsledku, ktorý chceme povelom dosiahnuť. Senzomotorický model je základom pre správne fungovanie autonómnych robotov.

Kapitola 2

Prehľad problematiky

Diplomová práca je rozdelená na dve na seba nadväzujúce časti. Prvou je emulácia kože v simulátore humanoidného robota iCubSIM. Druhou je autonómna explorácia tela simulovaného robota s použitím emulovanej kože. Táto kapitola sa venuje všeobecnej charakteristike a popisu robota a simulátora. Predstavuje riešenie kože na reálnom robotovi iCub. Bližšie špecifikácie, z ktorých sa v vychádza v ďalších kapitolách, sú uvedené v úvodoch daných kapitol.

2.1 Robot iCub

iCub je humanoidný robot vysoký približne jeden meter. Je navrhnutý tak, aby sa podobal ľudskému dieťaťu. Využíva sa pri výskumoch kognície a umelej inteligencie. Bol navrhnutý konzorciom európskych univerzít RobotCub a postavený v Italian Institute of Technology v Janove. K dnešnému dňu sa jeho exempláre využívajú vo viac ako dvadsiatich laboratóriách a univerzitách.[12]

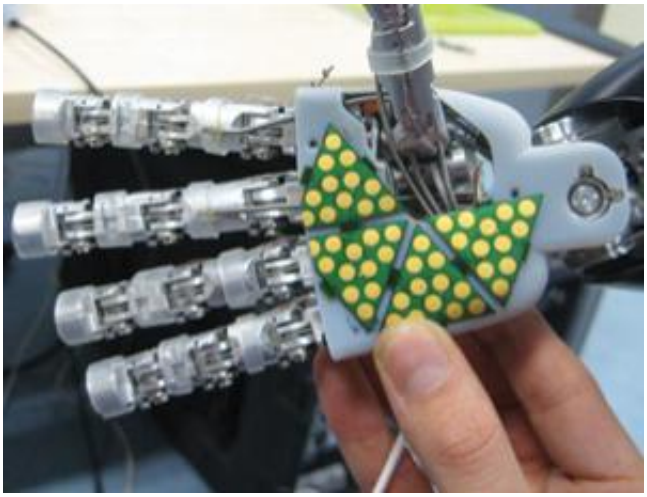
Robot má 53 motorov pohybujúcich jednotlivými kĺbmi a krkom a teda 53 stupňov voľnosti. Je vybavený senzormi sily a torznými senzormi. Oči reprezentujú dve pohyblivé kamery a v mieste uší má mikrofóny. Pomocou led diód na tvári dokáže robiť aj jednoduché výrazy tváre.[1]

Nemá vlastný zdroj energie a nebol navrhnutý na prekonávanie väčších vzdialeností, nakoľko je limitovaný kabelážou. Dokáže interagovať s okolím, uchopiť predmety, dokonca zvládne aj lukostreľbu.

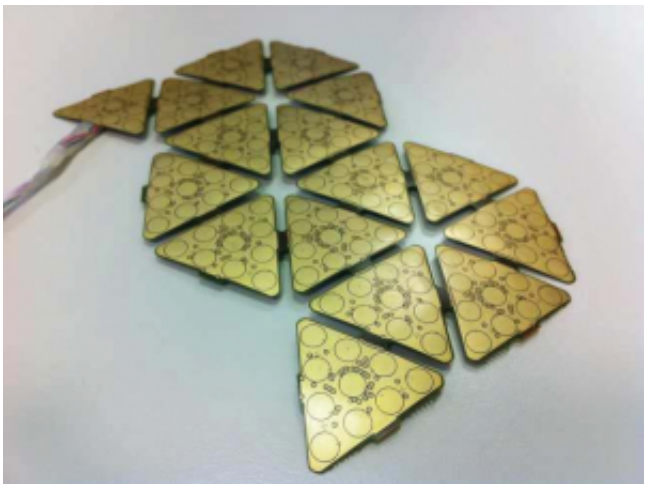
2.2 Koža robota iCub

Robot iCub bol postupne vybavený umelou kožou. Ide o skupiny tlakových senzorov, ktorými sú pokryté povrchové plochy jednotlivých častí robota. Konkrétne trup, horné a dolné končatiny, dlane a prsty na rukách. Táto umelá koža bola v priebehu vývoja robota zdokonalená a pre túto prácu sú zásadné dve jej verzie. Prvou verziou je verzia popísaná v [7] z roku 2012. Druhou verziou je verzia popísaná v [8] z roku 2013. Medzi týmito verziami sú síce zásadné technické rozdiely, avšak pri tejto práci, ktorá sa zaoberá hlavne rozmiestnením senzorov na tele robota, bolo možné a potrebné vychádzať z oboch verzií.

Základnou stavebnou časťou kože robota je taxel. Taxel je zjednodušene jeden tlakový senzor, jeden bod na koži. Taxely sú zoskupené na trojuholníkových platničkách. Trojuholníkový tvar a ohybný materiál platničiek umožňuje dobre pokryť zakrivený povrch robota. Na platničke sa nachádza v prípade staršej verzie kože dvanásť taxelov (obr. 2.1), v prípade novšej verzie kože desať taxelov (obr. 2.2).

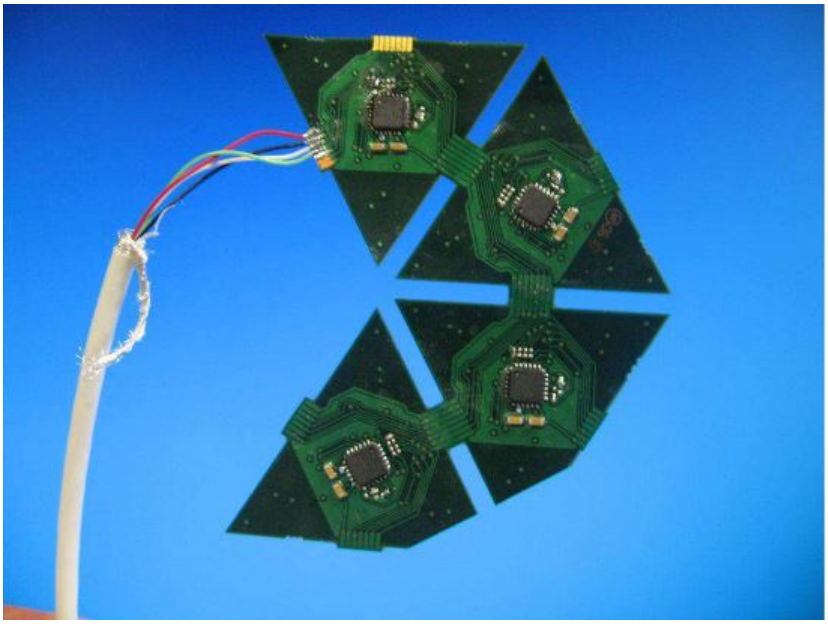


Obr. 2.1: Staršia verzia umelej kože. Zdroj: IIT Genova

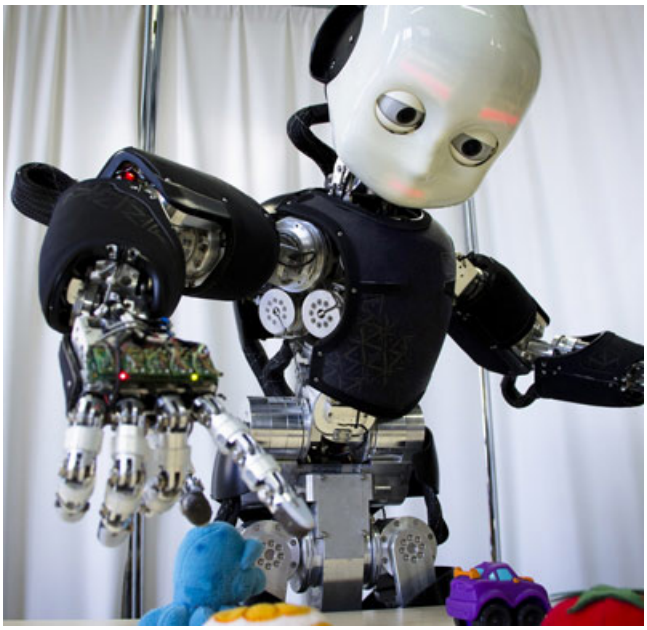


Obr. 2.2: Novšia verzia umelej kože. Zdroj: IIT Genova

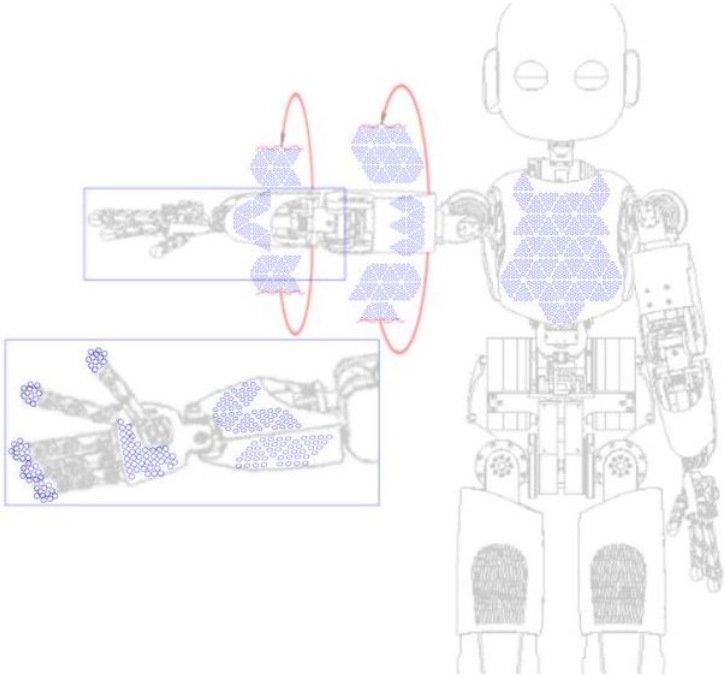
Jednotlivé platničky sa spájajú zbernicou do väčších celkov, až vytvoria celý povrch kože robota (obr. 2.3). Dôvodmi na vytvorenie takejto modulárnej kože bola nižšia cena výroby ako v prípade výroby celých kusov na mieru. Jej jednoduchšia prispôsobiteľnosť a možnosť aplikácie aj na skoro akéhokoľvek iného robota. Pokrytie robota už celistvými časťami umelej kože čiernej farby je znázornené na obrázku 2.4. Na obrázku 2.5 je znázornené pokrytie povrchu robota kožou s vyznačenými jednotlivými taxelmi.



Obr. 2.3: Spojenie platničiek umelej kože. Zdroj: IIT Genova



Obr. 2.4: Umelá koža na robotovi (čierne panely). Zdroj: IIT Genova



Obr. 2.5: Umelá koža na robotovi s vyznačenými taxelmi. Zdroj: IIT Genova

2.3 Simulátor iCubSIM

Nakoľko je robot iCub, aj keď je vyrábaný sériovo, veľmi drahý, bol vyvinutý iCub simulátor. iCub simulátor je voľne šírený softvér s otvoreným zdrojovým kódom. Jeho pôvodnými autormi sú Vadim Tikhanoff a Paul Fitzpatrick. iCub simulátor sa snaží používateľovi čo najvernejšie napodobniť správanie reálneho iCub robota. [3] iCub simulátor používajú ľudia bez prístupu k robotovi na svoj výskum, alebo aj výskumníci s prístupom k robotovi na predbežné overenie a kontrolu.

2.3.1 YARP

YARP (Yet Another Robot Platform) je knižnica a nástroj na multiprocesovú komunikáciu, v tomto prípade používaný na kontrolu iCub robota aj iCub simulátora. Primárnym programovacím jazykom pre YARP je C++, ale pomocou SWIG bindings podporuje väčšinu dnes používaných programovacích jazykov.

Príkazy a odpovede pre robota alebo pre simulátor sú v rovnakom tvare a na rovnakých portoch. Programátor tak môže použiť ten istý kód pre ovládanie robota aj simulátora. V prípade simulátora tak nastáva ilúzia, že programátor komunikuje so skutočným robotom. Dôležité je poznamenať, že z programátorského a technického hľadiska nie je možné naprogramovať simulátor tak, aby na rovnaký príkaz odpovedali robot aj simulátor úplne rovnako.

Komunikácia so simulátorom prostredníctvom YARP prebieha nasledovne.

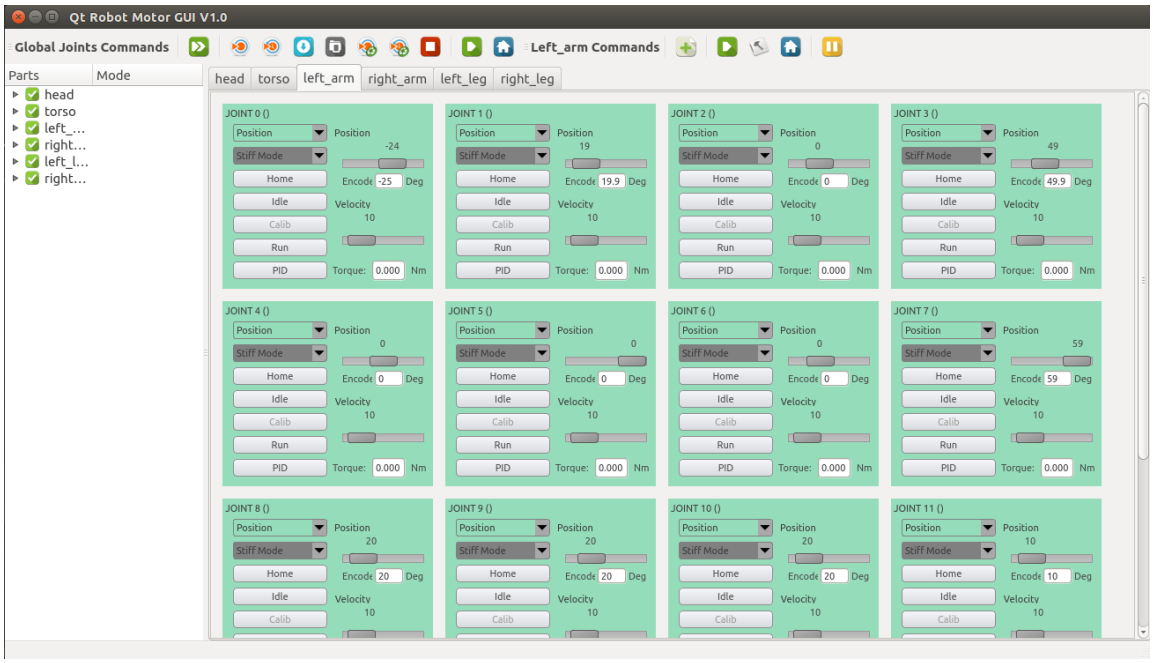
- Spustí sa program YARP.
- Spustí sa simulátor.
- Simulátor si v YARP pomocou príkazu */write* zaregistruje porty, na ktoré vysiela svoj aktuálny stav.
- Simulátor si v YARP pomocou príkazu */read* zaregistruje porty, na ktorých čaká príkazy.
- Programátor si v YARP pomocou príkazu */write* zaregistruje porty, na ktoré bude vysielať príkazy.

- Programátor si v YARP pomocou príkazu */read* zaregistruje porty, na ktorých čaká odpovede na príkazy, čiže aktuálny stav robota.
- Programátor v YARP pomocou príkazu */connect* spojí jednotlivé dvojice read a write portov.
- Teraz môže prebiehať obojsmerná komunikácia.

YARP je dostupný pre operačné systémy Linux a Windows. V rámci tejto diplomovej práce sme pracovali v operačnom systéme Linux.

2.3.2 Yarpmotorgui

Yarpmotorgui je program určený na ovládanie kĺbov robota či simulátora prostredníctvom YARP. Ponúka grafické používateľské rozhranie, kde je pre každý stupeň voľnosti každého kĺbu takzvaný slider, pomocou ktorého môžeme daný kĺb nastaviť na presný uhol (obr. 2.6). Tento program sme využívali hlavne pri testovaní emulácie kože.



Obr. 2.6: Program Yarpmotorgui.

2.3.3 SkinManager

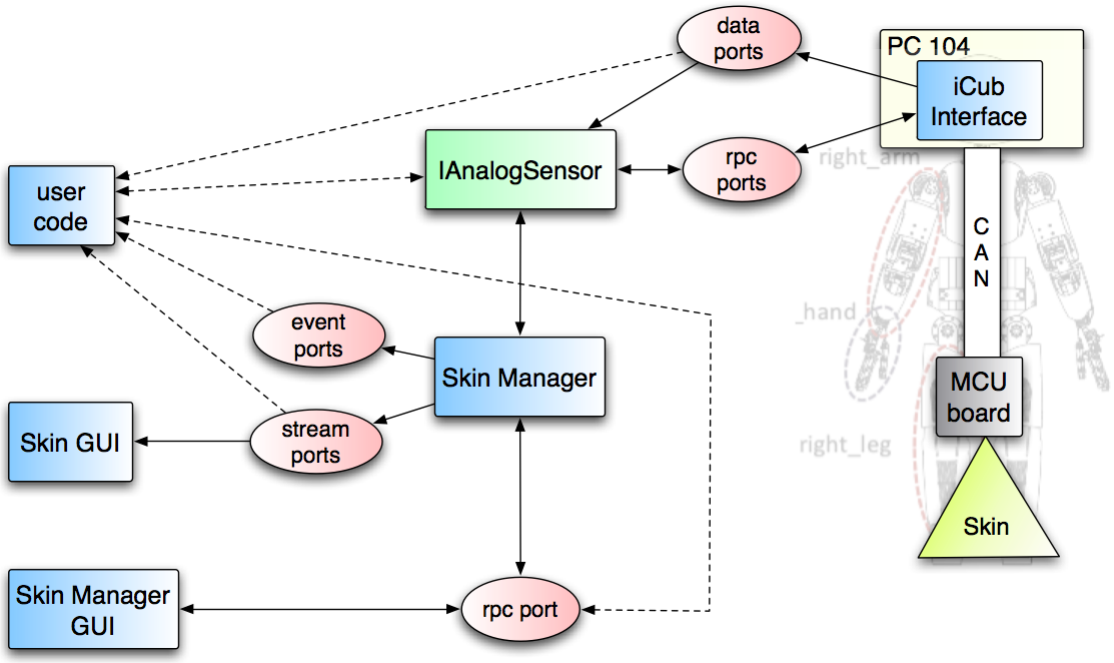
SkinManager je modul slúžiaci na reprezentáciu dotykových dát. Tento modul sa využíva u reálneho robota, v simulátore je tento modul zastúpený portom s rovnakým menom. S týmto portom pracujeme v kapitole 3. aj v kapitole 4. V kapitole 3. určujeme v *taxelList* tohto portu aktivované taxely. V kapitole 4. z neho odvodeného portu *skin_events* čítame dáta z *taxelList*. SkinManager okrem aktivovaných taxelov určí časť tela, na ktorej prebieha kontakt. Ďalej určí silu, normálu, tlak, moment a stred dotyku. S týmito dátami však nepracujeme.

2.3.4 iCubSkinGui

iCubSkinGui je program poskytujúci grafické používateľské rozhranie, ktoré vizualizuje výstup dotykových senzorov na 3D modeli robota a 2D mapách symbolizujúcich jednotlivé časti umelej kože. Väčšina vizualizácií v tejto práci bola vytvorená s použitím tohto programu. Funkcionalitu programu najlepšie ilustruje obrázok 3.1.

2.3.5 Tok dotykových dát

Práca s iCub simulátorom prebieha pomocou platformy pre multiprocesovú komunikáciu YARP. Obrázok 2.7 ilustruje generovanie a spracovanie dotykových dát v rámci iCub simulátora, alebo robota, a ich následné odoslanie prostredníctvom YARP portov, kde môžu byť spracované programátorom alebo vizualizačným programom iCubSkinGui.



Obr. 2.7: Tok dotykových dát. Zdroj: [13]

2.4 Použité programovacie jazyky a nástroje

2.4.1 MATLAB

MATLAB je programovacie prostredie (s vlastným programovacím jazykom) špecializujúce sa na vedeckotechnické numerické výpočty, modelovanie, návrhy algoritmov a počítačových simulácií, analýzu a prezentáciu údajov, merania a spracovania signálov, návrhy riadiacich a komunikačných systémov. [14]

2.4.2 C++

C++ je viacparadigmový programovací jazyk vyššej úrovne na všeobecné použitie, ktorý umožňuje pracovať aj s prostriedkami nízkej úrovne. Má statickú typovú kontrolu, podporuje procedurálne programovanie, dátovú abstrakciu, objektovo orientované programovanie, ale aj generické programovanie.

2.4.3 Python

Python je interpretovaný, interaktívny programovací jazyk, ktorý vytvoril Guido van Rossum, pôvodne ako skriptovací jazyk pre Amoeba OS schopný robiť systémové volania. Python je často porovnávaný s jazykmi Tcl, Perl, Scheme, Java a Ruby. Python je vyvíjaný ako open source projekt. [15]

2.4.4 SWIG

SWIG je nástroj na vývoj softvéru, ktorý prepája programy napísané v C a C++ s inými programovacími jazykmi vyššej úrovne. SWIG je možné využiť pri rôznych aj skriptovacích jazykoch. Napríklad Javascript, Perl, PHP, Python, Tcl a Ruby. [16]

Kapitola 3

Emulácia kože

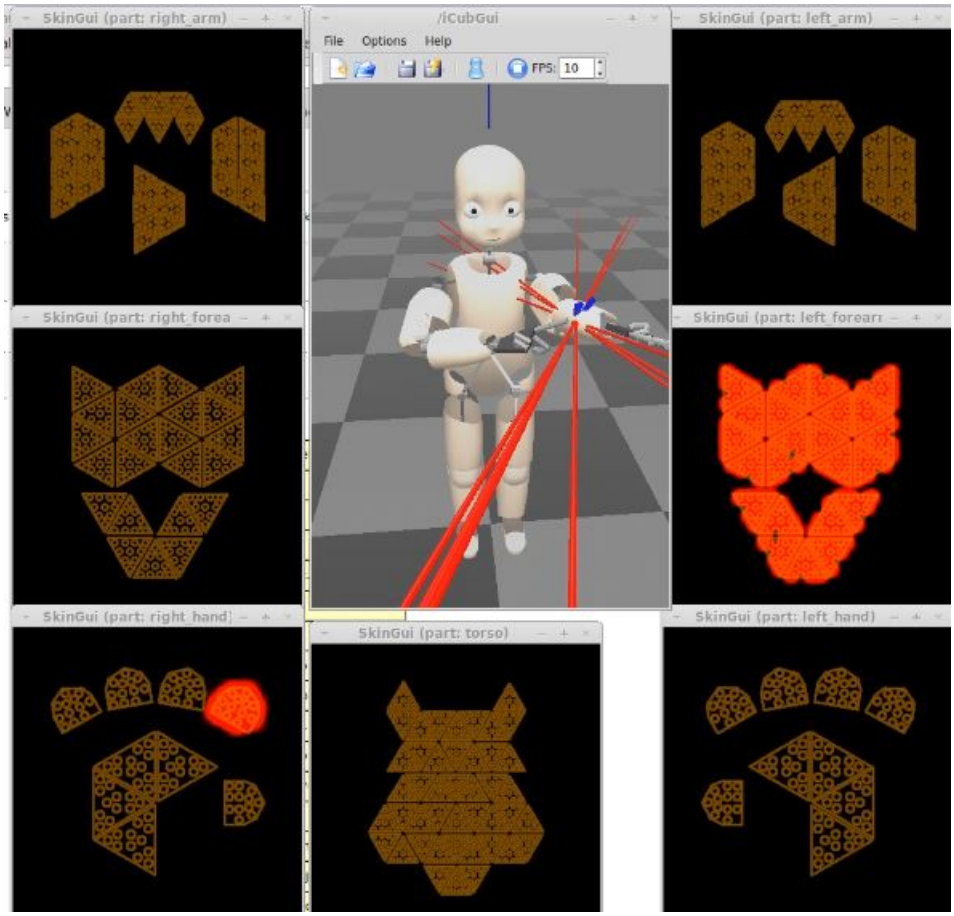
Prvou časťou práce je emulácia kože v iCub simulátore. Emulácia je napodobenie systému spracovania údajov, alebo jeho časti iným systémom, alebo technickým prostriedkom. Emulácia kože už bola čiastočne implementovaná v predošlých verziách simulátora, avšak nie dostatočne na to, aby sme mohli pomocou nej realizovať druhú časť práce.

Robot iCub má umelou kožou čiastočne pokrytý trup, ramená, lakte, dlane a prsty. Tieto sú emulované aj v simulátore. Dotyk prebieha tak, že po kontakte s kožou dostaneme presné 3D súradnice daného dotyku. Avšak toto nie je pri reálnom robotovi možné. Reálny robot má po tele rozmiestnené dotykové platničky a deteguje dotyk na danej platničke. Preto chceme aj v simulátore detegovať dotyk o konkrétnu platničku.

3.1 Emulácia kože v staršej verzii simulátora

V staršej verzii simulátora sa nedali rozlíšiť jednotlivé platničky. Táto možnosť bola zatiaľ implementovaná iba pri dlani a prstoch. Pri dlani a prstoch bol problém redukovaný na 2D, nakoľko sa žiadne platničky neprekrývajú v smere osi z. Taktiež je na dlani menej platničiek, a sú od seba lepšie oddelené.

Na ostatných častiach tela sa po kontakte aktivovali všetky platničky danej časti tela, čiže sa inak ako zo súradníc nedalo určiť, kde dotyk nastal. Na obrázku 3.1 je táto situácia znázornená a je vidno, že napravo je vyznačené úplne celé predlaktie, zatiaľ čo z ruky je vyznačený iba jeden prst.



Obr. 3.1: Staršia verzia emulácie kože. Autor: Matěj Hoffmann

3.2 Špecifikácia cieľa

Cieľom bolo dosiahnuť vyššiu presnosť pri emulovaní kože tak, aby bolo možné dostať čo najpresnejšie dáta a tiež čo možno najväčšiu množinu dát pri autonómnej explorácii tela. V budúcnosti bude možné túto emuláciu používať pri práci so simulátorom.

Motorické schopnosti robota iCub nie sú totožné s ľudskými. Telo robota je postavené z pevných častí, a preto prstami nedočiahne všade tam kde človek. V oblasti brucha a ramien je len málo bodov, kde je možný dotyk jeho vlastným prstom. Problém sme preto zredukovali len na ruky a predlaktia. Ako som už spomenul, emulácia kože na rukách už bola implementovaná. Naším cieľom bolo emulovať kožu na predlaktiach.

Dôležitým kritériom pri výbere riešenia bola výpočtová náročnosť použitého algoritmu. iCub simulátor sa skladá z množstva modulov a knižníc, ktoré značne zaťažujú systém. Okrem toho používatelia majú popri simulátore spustený ďalší softvér, napríklad na vizualizáciu dát a podobne. Dôležité je tiež to, že algoritmus musí odpovedať v reálnom čase, a pre viacero dotykov, aj na viacerých častiach tela. Treba si totižto uvedomiť, že dotyk môže spôsobiť všetkých desať prstov naraz. Taktiež môže dotyk spôsobiť kolízia dvoch iných častí tela, napríklad kolízia predlaktia s trupom. Dotyk môže spôsobiť aj kolízia s iným predmetom v prostredí simulátora. Možným scenárom je aj viacero typov dotykov naraz.

Úplná presnosť algoritmu nie je dôležitá a je prakticky nedosiahnuteľná. Model robota v simulátore totižto nezodpovedá do detailov reálnemu robotovi. Celá koža robota nie je pokrytá senzormi, avšak v simulátore chceme pokryť celú túto plochu. Ďalším problémom sú nepresné súradnice jednotlivých platničiek. Okrem toho pri výpočte súradníc kolízie je potrebný prepočet súradníc danej kolízie z globálnej súradnicovej sústavy robota do lokálnej súradnicovej sústavy danej časti tela. Tento prepočet nie je úplne presný, vzniká tu chyba.

3.3 Riešenie

Riešenie problému je rozdelené do troch častí: Návrh, Implementácia a Testovanie. Návrh pozostával z analýzy vstupných dát a rozdelenia priestoru podľa nich. Prebiehal v prostredí Matlab. Implementácia prebiehala v jazyku C++ v ktorom je naprogramovaný simulátor. Testovanie prebiehalo na niekoľkých neoficiálnych testovacích verziách simulátora. Išlo prevažne o vizuálnu kontrolu, nakoľko v tom čase ešte algoritmus na autonómnu exploráciu nebol naprogramovaný. Pri riešení sme prvé dve časti niekoľko krát opakoval, najmä kvôli výstupom testovania. Prvá verzia algoritmu, ktorá ale nepokrýva dokonale celý povrch kože predlaktí, je implementovaná v oficiálnej verzii simulátora od leta 2015.

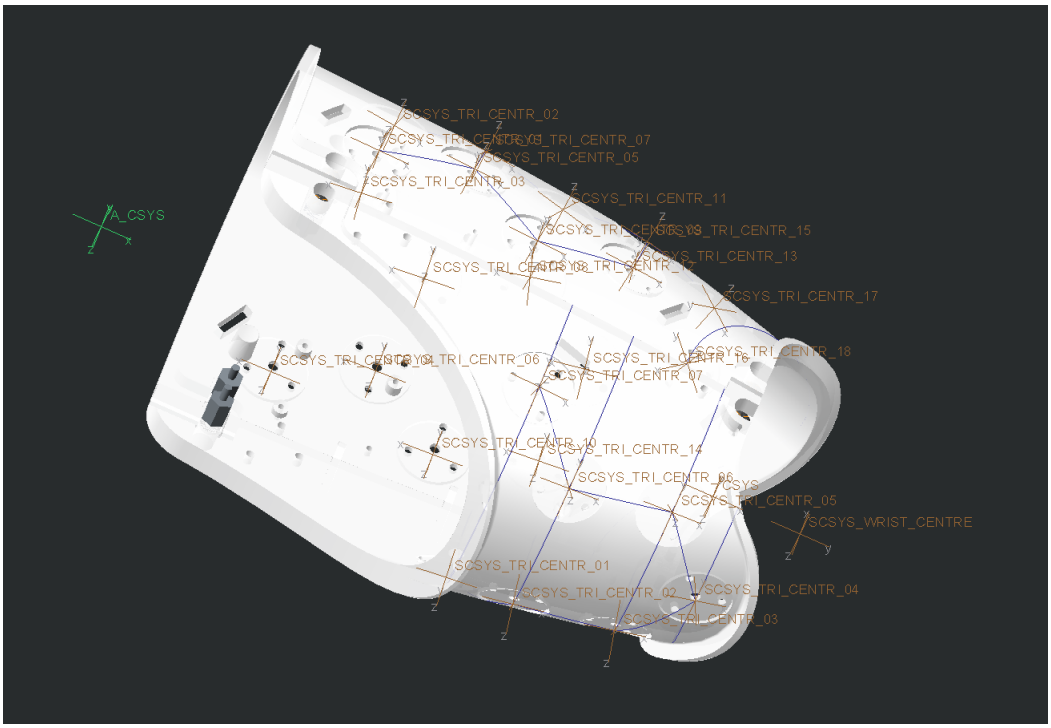
3.3.1 Návrh

3.3.1.1 Vstupné dáta

Vstupné dáta pozostávali z dvoch množín súradníc. Prvá množina sú súradnice získané z experimentu popísanom v [2]. Nakoľko dáta pochádzajú z roku 2011, kedy mal robot ešte staršiu verziu kože, obsahuje táto množina iný počet taxelov pre jednotlivé platničky. Výsledky tohoto experimentu nie sú dokonale presné a teda ani táto množina vstupných dát.

Množina pre predlaktie obsahuje 384 súradníc taxelov. Každá platnička je zastúpená 12-imi za sebou nasledujúcimi taxelmi. Spodná časť predlaktia začína na indexe 1 a končí na indexe 192, 16 platničiek. Horná časť predlaktia začína na indexe 205 a končí na indexe 360, 7 platničiek.

Druhá množina súradníc pozostáva zo súradníc dier na konštrukcii robota, do ktorých sa uchytávajú dotykové platničky pri jeho konštrukcii(obr. 3.2). Tieto súradnice približne zodpovedajú stredom platničiek. Táto množina súradníc presnejšie zodpovedá realite ako prvá a pri návrhu vychádzame hlavne z nej.



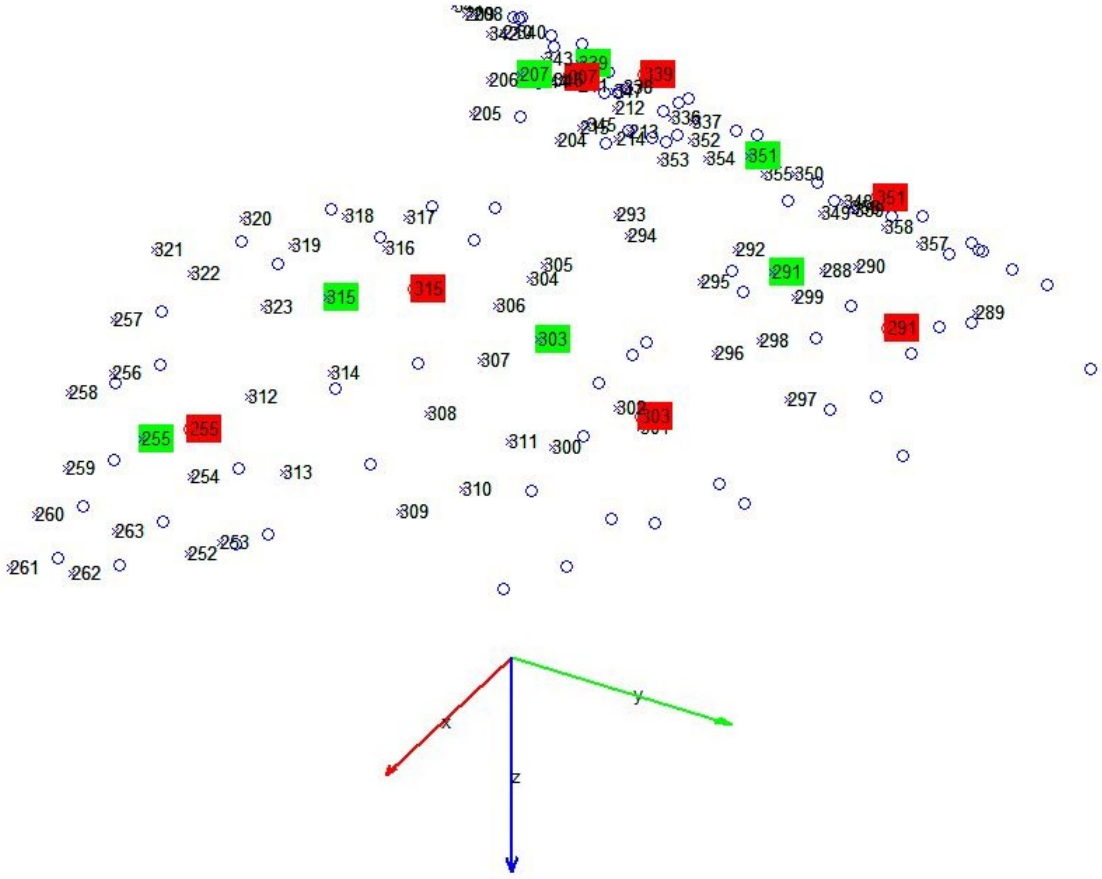
Obr. 3.2: Návrh konštrukcie povrchu predlaktia. Zdroj: IIT Genova

Množina obsahuje pre každú platničku jedny súradnice. Sú však indexované v inom poradí ako súradnice prvej množiny. Preto bolo potrebné súradnice vizualizovať a vizuálne navzájom množiny spárovať. Vizualizácia súradníc pre hornú časť predlaktia je zobrazená na obrázku 3.3.

3.3.1.2 Postup

V prvom kroku sme posunuli všetky súradnice prvej množiny o rozdiel medzi stredným taxelom a prislúchajúcou súradnicou z druhej množiny tak, aby sa súradnice druhej množiny stali stredmi jednotlivých platničiek. Toto bolo nutné spraviť pre to, aby sme videli aspoň približne kam jednotlivé platničky skutočne zasahujú, nakoľko ich trojuholníkový tvar a zakrivenie sa medzi jednotlivými verziami kože príliš nelíšia. Na obrázku 3.3 je vidno rozdiely medzi stredmi prvej množiny označenými zelenou farbou a stredmi druhej množiny označenými červenou farbou. Ďalej je znázornený posun súradníc prvej množiny označených čiernou farbou na nové pozície označené modrou farbou. Obrázok pokrýva iba hornú časť predlaktia. Obrázok pre spodnú časť predlaktia je možné vygenerovať pomocou MATLAB skriptu

[18] /matlab_left_forearm/skinTesselation3D_left_forearm.m nakoľko je príliš veľký, aby sa zmestil na papier a zároveň by sa dali odlíšiť jednotlivé body. Pod obrázkom nasleduje ukážka zdrojového kódu 3.1, ktorý zabezpečuje posun súradníc.



Obr. 3.3: Vizualizácia úpravy vstupných súradníc.

Ukážka 3.1: Posun súradníc v MATLAB.

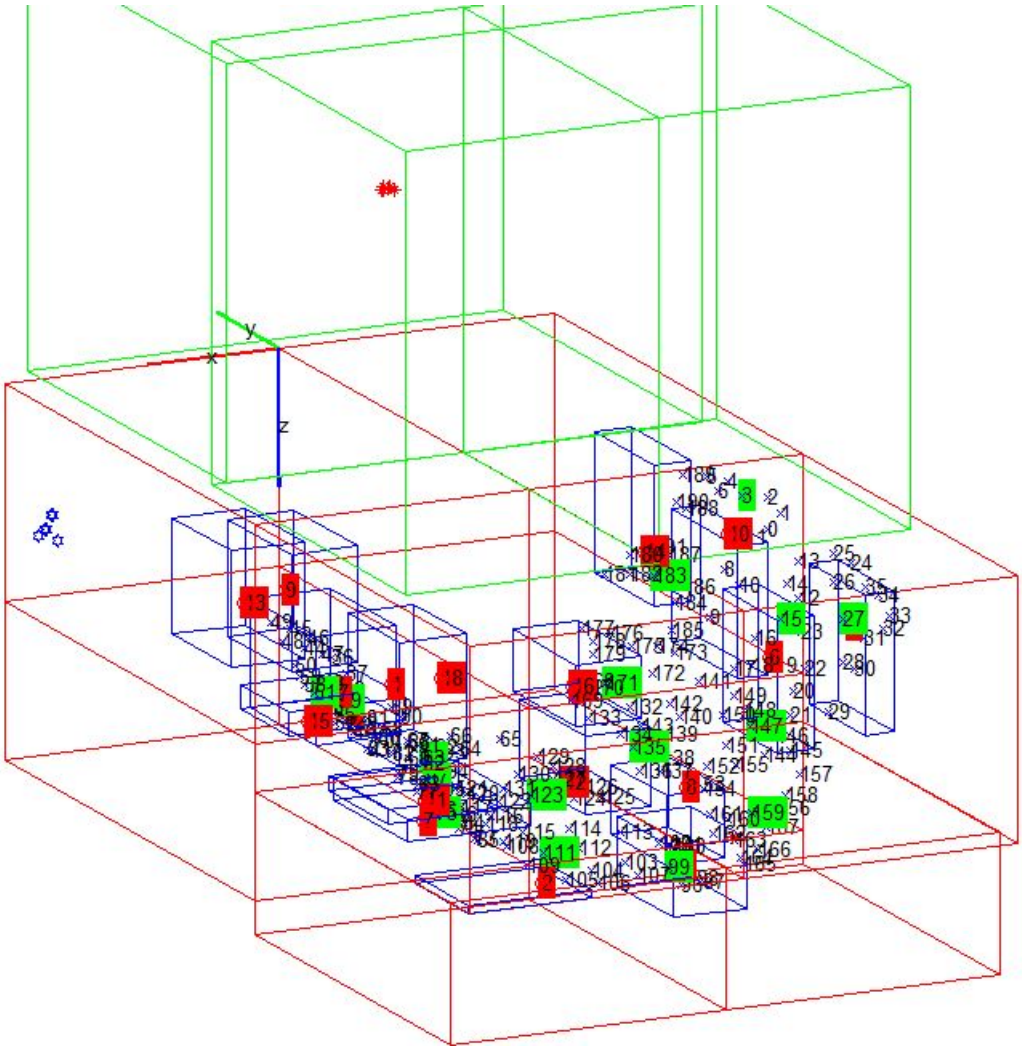
```
trans_207 = zeros(1,3);
 % planicka so stredom na indexe 207 (208 cislovanie od 1)

trans_207(1,:) = taxel_pos(208,:) - triangle_centers_CAD(1,:);
 % rozdiel medzi setmi

translated_coordinates(i, j) = taxel_pos(i,j) - trans_207(1,j);
 % odratanie rozdielu
```

V druhom kroku sme každú platničku tesne obalili do hranola, ktorého hrany sú rovnobežné s jednotlivými súradnicovými osami predlaktia. Na obrázku 3.4 sú tieto hranoly znázornené modrou farbou. Všimli sme si, že niektoré hranoly sa navzájom prekrývajú. Predlaktie môžeme rozdeliť na dve časti rovinou danou bodom (0, 0, 0) a vektorom (0, 0, 1) (zjednodušene povedané vodorovnou rovinou) tak, že nepretneme žiaden hranol. Tým sa nám predlaktie rozdelilo na hornú a dolnú časť. Takto vieme ďalej rozdeliť na hornej časti jednu trojicu a dve dvojice hranolov pomocou rovín, ktorých normálový vektor je rovnobežný s jednou zo súradnicových osí predlaktia. Podobne vieme určiť osem dvojíc na dolnej časti predlaktia.

V treťom kroku sme tieto dvojice a trojicu opäť obalili do hranolov, ktorých hrany sú rovnobežné s jednotlivými súradnicovými osami predlaktia tak, že celá dvojica resp. trojica sú obalené jedným hranolom. Medzi hranolmi nesmie ostať nepokrytý priestor. Každý hranol sme v danom smere zväčšili o polovicu rozdielu medzi stenami dvojice hranolov, medzi ktorými bol nepokrytý priestor. Na obrázku 3.4 sú tieto hranoly znázornené zelenou farbou pre hornú časť predlaktia a červenou farbou pre spodnú časť predlaktia.



Obr. 3.4: Vizualizácia rozdelenia priestoru pomocou hranolov.

Hranoly sú definované dvojicou bodov. Jedná sa o body s minimálnymi a maximálnymi hodnotami súradníc (x, y, z). Nakoľko sú hrany každého hranola rovnobežné s jednotlivými súradnicovými osami predlaktia, stačia nám tieto dva body na ich jednoznačné určenie.

3.3.2 Implementácia

Implementácia je riešená v jazyku C++. Dôvodom voľby jazyka C++ bol fakt, že išlo o doplnenie funkcionality už existujúcej funkcie `OdeSdlSimulation::mapPositionIntoTaxelList()` v rámci zdrojového kódu iCub simulátora v súbore

[17] /icub-main/src/simulators/iCubSimulation/odesdl/iCub_Sim.cpp .

Funkcia funguje tak, že má zadefinované jednotlivé taxely a musíme určiť tie, na ktorých nastal kontakt. Tieto taxely zapíšeme vo forme vektora (údajová štruktúra C++), ktorý nemá zložitejšiu štruktúru. Z teoretického hľadiska môžeme povedať, že ide o jednorozmerné pole. Tento vektor obsahuje indexy daných taxelov.

Označujeme naraz celú platničku. Každá platnička má 10 taxelov. Indexy týchto taxelov nasledujú vždy po sebe. Vytvorili sme preto pomocnú funkciu OdeSdlSimulation::pushTriangleToTaxelList(), ktorá po zadaní prvého taxelu platničky automaticky určí aj ostatné. Dôležité je povedať, že porty majú stále zadefinovaných pôvodných 12 taxelov zo staršej verzie kože. Musíme preto vždy vynechať siedmy a jedenásty taxel, nakoľko tieto už v novej koži nie sú (ukážka 3.2).

Ukážka 3.2: Určenie taxelov určitej platničky v C++.

```
1  void OdeSdlSimulation::pushTriangleToTaxelList
2  (const int startingTaxelID,std::vector<unsigned int>& list_of_taxels)
3  {
4      int i = startingTaxelID;
5      for (i=startingTaxelID;i<startingTaxelID+6;i++){
6          list_of_taxels.push_back(i);
7      }
8      //skipping 7th and 11th taxel - thermal pads
9      for (i = startingTaxelID + 7; i < startingTaxelID + 10; i++){
10         list_of_taxels.push_back(i);
11     }
12     list_of_taxels.push_back(startingTaxelID+11);
13 }
```

V hlavnej funkcii dostaneme stred dotyku. Je to jeden bod so súradnicami (x, y, z). Tieto súradnice sú v lokálnej súradnicovej sústave predlaktia. Porovnaním určíme, do ktorého hranola podľa návrhu daný bod patrí. Následne pre každý taxel daného hranola spustíme vyššie popísanú pomocnú funkciu, ktorá označí všetky jeho taxely. Pokiaľ bod nepatrí do žiadneho hranola, simulátor vypíše chybu s danou súradnicou. Napriek chybe simulácia ďalej pokračuje.

V ukážke kódu 3.3 je implementovaná kontrola pre jeden konkrétny hranol, do ktorého patria dva trojuholníky.

Ukážka 3.3: Určenie referenčného hranola a jeho taxelov v C++.

```
if((geo_center_link_FoR[0]>-0.0345) && (geo_center_link_FoR[0]<0)
&& (geo_center_link_FoR[1]<0.1087) && (geo_center_link_FoR[1]>0.0528)
&& (geo_center_link_FoR[2]<0.0569) && (geo_center_link_FoR[2]>0)){
       //triangle 204:215
      pushTriangleToTaxelList(204,list_of_taxels);
      //triangle 336:347
      pushTriangleToTaxelList(336,list_of_taxels);
}
```

3.3.3 Testovanie

Po implementácii bola emulácia kože na predlaktiach zaradená do vtedy aktuálnej verzie iCub simulátora. Avšak nastalo niekoľko problémov, ktoré neovplyvňujú chod simulátora, ale spôsobujú výskyt nesprávnych výsledkov. Bolo teda nutné vykonať testovanie a opraviť chyby, resp. nepresnosti v návrhu a implementácii.

Boli to chyby spôsobené nedostatočným pokrytím priestoru hranolmi. Nastala situácia, že bol zaevidovaný dotyk na povrchu predlaktia, ale nebol zadelený do žiadneho hranola, a preto sa neoznačili žiadne taxely. Bolo teda nutné skontrolovať rôzne konfigurácie modelu robota a zistiť, kedy dochádza k chybe.

Druhou časťou testovania bolo zistiť či je rozdelenie priestoru správne. Teda určiť, ak detegujeme a zaradíme dotyk do určitého hranola, či je naozaj zaradený do správneho hranola.

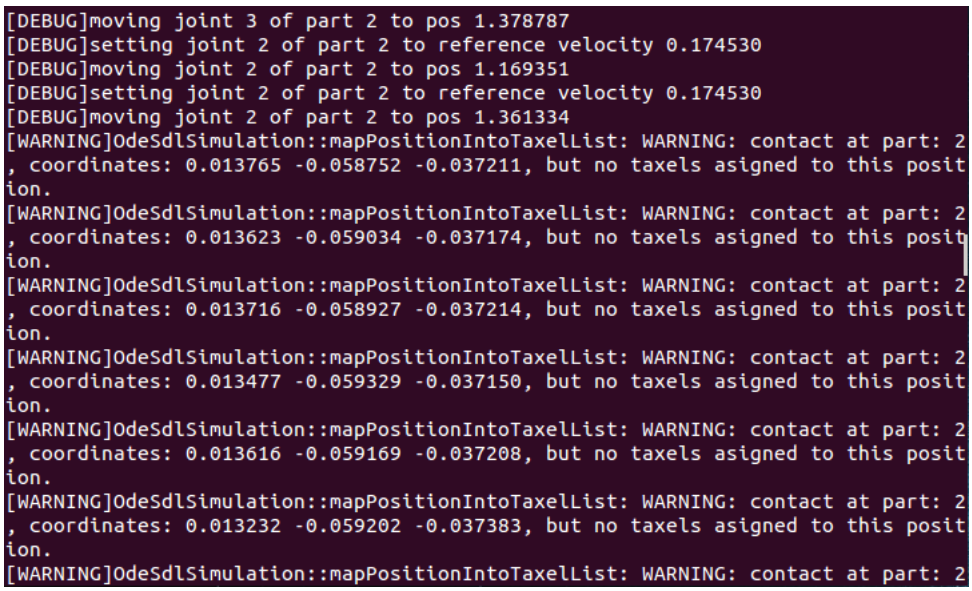
3.3.3.1 Postup

Obe časti testovania sme robili naraz a bez použitia špecializovaného testovacieho algoritmu alebo programu. A to z toho dôvodu, že autonómne prehľadávanie je až

druhou časťou práce a v dobe testovania ešte táto časť nebola implementovaná. Existujú aj algoritmy na generovanie dotykov vytvorené v rámci iných projektov s iCub simulátorom. Rozhodli sme sa ale, že bude lepšie robiť kontrolu ručne, hlavne kvôli jemnému nastaveniu uhlov jednotlivých kĺbov.

Testovanie prebiehalo najprv na oficiálnej verzii simulátora, neskôr na neoficiálnych upravených verziách. Keď sa vykoná zmena v zdrojovom kóde simulátora, je potrebné tento znovu skompilovať a nainštalovať. Manipulácia s robotom bola zabezpečená pomocou programu Yarpmotorgui.

Simulátor bol spúšťaný spolu s príkazom *-verbosity4*, ktorý umožňuje automatický výpis hlásení. Vďaka tomu bolo možné jednoducho z terminálu odčítať hodnoty súradníc, kde nastal nedetekovaný dotyk. Na obrázku 3.5 je tento výpis zobrazený. Program najprv vypíše povely, ktoré robot vykonal, a následne vypíše chybu. Najprv vypíše názov funkcie, potom číselne časť tela, kde nastal kontakt a nakoniec súradnice tohoto kontaktu.



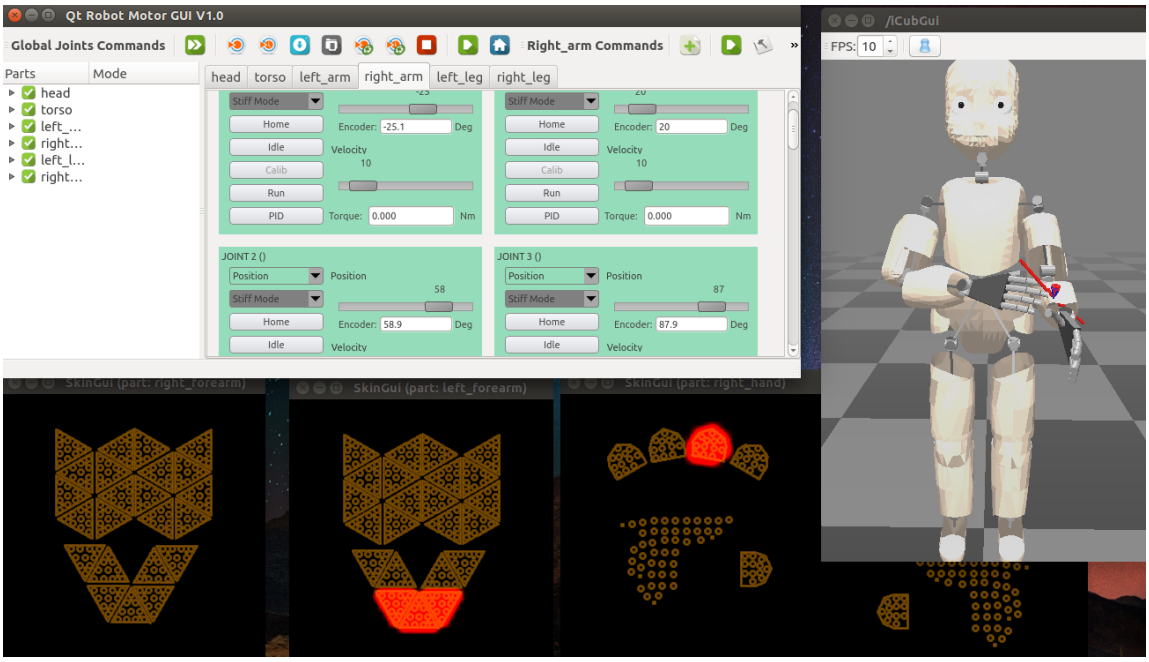
Obr. 3.5: Chybové hlásenie o nezaradenom dotyku.

Následne boli tieto súradnice zadané do návrhu v prostredí Matlab. Súradnice boli

zakreslené spolu s návrhom, kedy sa ukázalo, kde je návrh chybný, respektíve rozmery ktorých hranolov sú nedostatočné. Napriek tomu, že sa jednalo len o maličké rozdiely, každý dotknutý hranol bol v danom smere zväčšený o 0,02 (dĺžkovej jednotky simulátora). Môžeme si dovoliť každý hranol zväčšiť aj o pomerne veľkú hodnotu 0,02 z toho dôvodu, že najprv určíme konkrétnu časť tela, a až potom sa vykonáva rozdelenie do hranolov. Čiže ku chybnému určeniu a zásahu do inej časti tela nemôže dojsť. Zmeny boli následne vykonané vo funkcii určujúcej rozdelenie priestoru v jazyku C++ podľa nového návrhu.

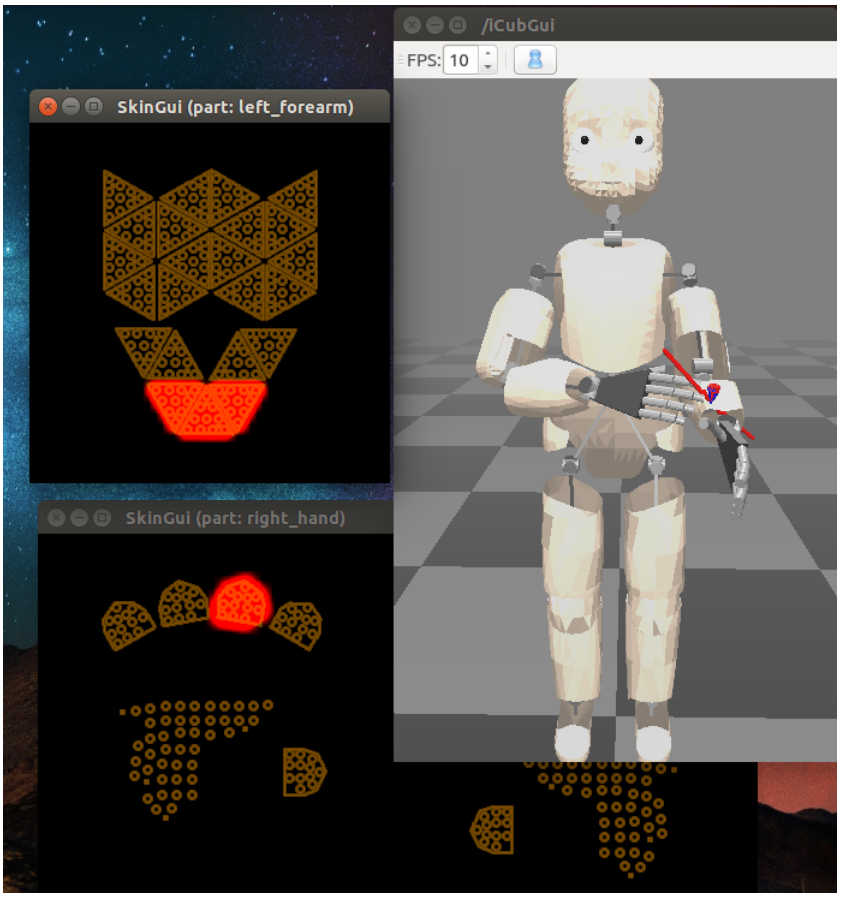
Paralelne s tým bola robená vizuálna kontrola správneho rozmiestnenia hranolov. Vizualizácia procesu bola zabezpečená programom Yarpmanager a iCubSkinGui. Tieto vizualizujú 3D model robota, ktorého hlavnou časťou sú plochy reprezentujúce kožu a vizualizuje každý dotyk, aj taký, ktorý nie je zaradený do žiadneho hranola, respektíve obdĺžnika v prípade ruky. Ďalej vizualizuje 2D modely všetkých dotykových platničiek a ich taxelov. Tieto ale už reagujú iba na dotyk zaradený do niektorého hranola, teda taký, ktorému sú priradené konkrétne taxely.

Na obrázku 3.6 sú znázornené jednotlivé okná programov používaných pri testovaní. Bielo-zelené okno je Yarpmotorgui. Ostatné okná patria programu iCubSkinGui, vpravo je model robota a dole sú čierne 2D modely dotykových platničiek.

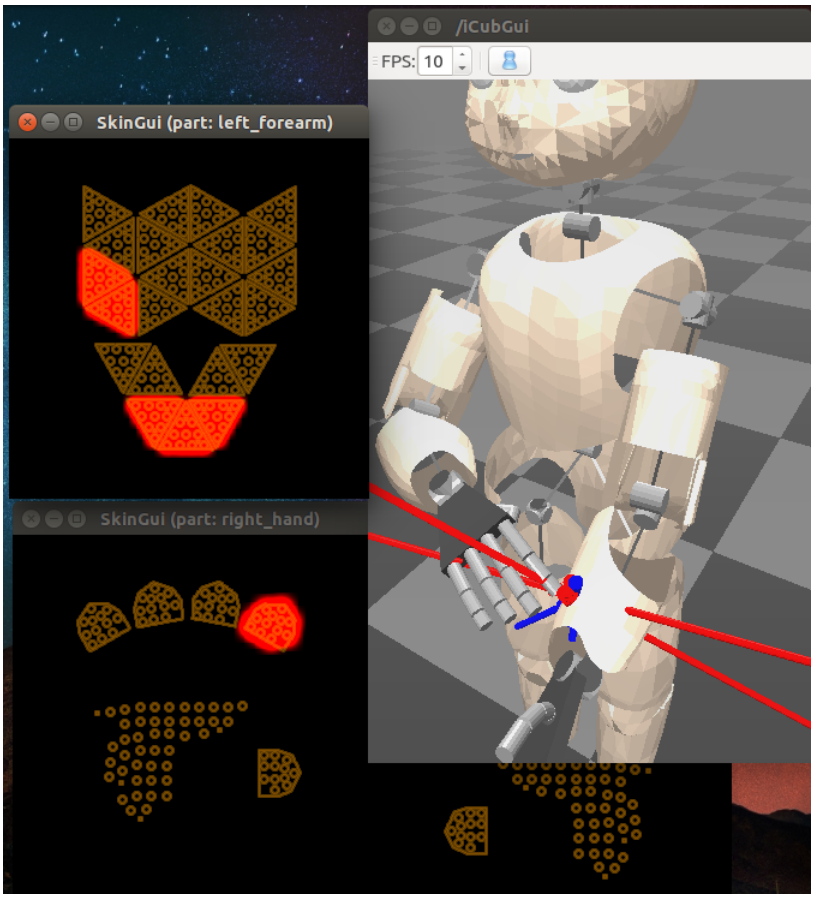


Obr. 3.6: Programy využívané pri testovaní emulácie kože.

Vizuálna kontrola je samozrejme nepresná. Ide tu hlavne o hrubú kontrolu správneho rozmiestnenia hranolov. Prebiehala porovnávaním polohy dotyku na 3D modeli robota a 2D modeloch dotykových platničiek. Kontroloval sa aj plynulý prechod medzi susednými hranolmi, respektíve platničkami. Pri tom sme si všimli pomerne zaujímavé správanie algoritmu. Keď sa totižto dotýkajúci sa predmet dostane na rozhranie dvoch hranolov, platničky oboch hranolov indikujú dotyk. Zhodli sme sa na tom, že takéto správanie je v poriadku, nakoľko nejde o kolíziu bodu s plochou, ale plochy s plochou, a v takom prípade algoritmus v jednom kroku simulácie prebehne viackrát, a môže označiť zároveň platničky z rôznych hranolov. Na obrázku 3.7 je znázornený dotyk jedného prsta o platničky jedného hranola. Na obrázku 3.8 je znázornený dotyk jedného prsta o platničky na rozmedzí dvoch hranolov. Môže sa zdať, že na obrázku 3.8 nejde o susedné platničky. Treba si ale uvedomiť, že v tomto konkrétnom prípade je to prechod medzi hornou a dolnou časťou predlaktia a iCubSkinGui ich v 2D vizualizuje oddelene.



Obr. 3.7: Testovanie emulácie kože - štandardný prípad.



Obr. 3.8: Testovanie emulácie kože - dotyk na rozhraní dvoch hranolov.

Postup bol niekoľko krát opakovaný tak, aby sa našlo a opravilo čo najviac chýb a nepresností.

3.4 Výsledok

Výstupom tejto časti diplomovej práce je algoritmus na určenie konkrétnych taxelov, zasiahnutých pri dotyku predlaktia iCub simulátora. Algoritmus je súčasťou funkcie `OdeSdlSimulation::mapPositionIntoTaxelList()` v rámci zdrojového kódu iCub simulátora v súbore [17] `/icub-main/src/simulators/iCubSimulation/odesdl/iCub_Sim.cpp` . Pokiaľ nastane akýkoľvek dotyk na koži iCub simulátora, spúšťa sa táto funkcia. Tá dostane ako vstupný parameter súradnice dotyku.

Funkcia najprv rozhodne, o ktorú časť tela ide. Pokiaľ ide o jedno z predlaktí, spúšťa sa náš algoritmus pre dané predlaktie. Najprv sa určí pomyselný hranol, do ktorého súradnice patria. Nakoľko celý priestor predlaktia je rozdelený na navzájom sa neprekrývajúce hranoly, ktorých hrany sú rovnobežné so súradnicovými osami predlaktia, ku každému hranolu sú pridelené dve až tri dotykové platničky. Každá platnička obsahuje desať taxelov. Pokiaľ je zvolený konkrétny hranol, funkcia zaradí všetky jeho taxely do poľa, ktoré je výstupom funkcie. Všetky taxely, ktoré sú na výstupe z funkcie, sú považované za také, ktoré by na reálnom iCub robotovi indikovali dotyk. Informácie o týchto vybraných taxeloch sú výstupom simulácie a dajú sa ďalej spracovávať.

3.5 Spracovanie výstupu dotykových dát v jazyku Python

Dôležitým medzikrokom medzi emuláciou kože a jej využitím vo výskume je spracovanie tohoto výstupu na strane programátora. Nakoľko ďalšia časť tejto práce je implementovaná v programovacom jazyku Python, avšak nevyužíva žiadnu funkcionalitu unikátnu pre tento jazyk, a tým že využívame YARP SWIG bindings, je tento kód jednoducho preložiteľný do bežných jazykov, napríklad C++.

Navrhli sme triedu armReader [18] `/goal-babling-fix-start-randomer/armreader.py`, ktorá zabezpečí komunikáciu prostredníctvom YARP, spracovanie vstupu a výstup vo forme reťazca. Trieda armReader spracúva iba vstupy z častí tela, ktoré sme už emulovali, čiže dlane a predlaktia, spôsobené dotykom vlastného prsta. Pre spracovanie rôznorodejších dát je možné podmienku dotyku samého seba vlastným prstom takzvaný dvojdotyk odstrániť. Avšak pre potreby druhej časti tejto práce bola táto podmienka nutná.

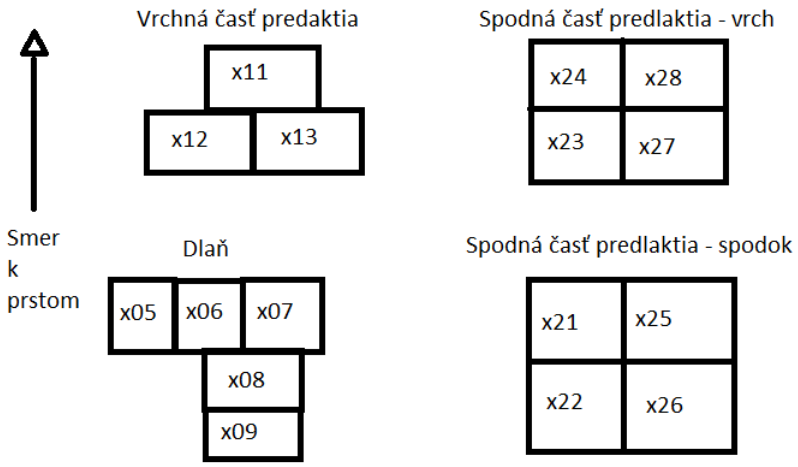
V konštruktore sa vytvorí port použitím `yarp.BufferedPortBottle()` a spojí sa s portom `/icubSim/skinManager/skin_events:o`. Vo funkcii getData si načíta jeden výstup z portu. Ten prevedie na reťazec, ktorý sa následne rozdelí na pole podreťazcov, pričom každý podreťazec obsahuje iba jednu dotykovú informáciu. Tým, že ide o dotyk plochy o plochu, sú mnohé podreťazce z nášho pohľadu zastúpené duplicitne, ale líšia sa

v parametroch, ktoré nesledujeme. Skontrolujeme, či aspoň jeden podreťazec obsahuje záznam o dotyku ukazovákom. Ak áno, pokračujeme v spracovaní. Odčítame sekvenciu dvoch znakov určujúcich časť tela, na ktorej prebieha dotyk podľa[13]. Keď poznáme časť tela, načítame id taxelov z TaxelList, ktorý je tiež súčasťou každého podreťazca. Pre každú oblasť jednej, dvoch, alebo troch patničiek podľa návrhu, sme určili unikátny identifikátor. Pre každý zaregistrovaný dotyk preložíme id taxelov z TaxelList na tento identifikátor a ten zaradíme do výstupu funkcie (ukážka 3.4).

Ukážka 3.4: Spracovanie dotykov v Python.
Originál: [18]/arms_both/goal-babling-randomer/armreader.py

```
class armReader: lass armReader:
def __init__(self):
    self.in_port = yarp.BufferedPortBottle()
    self.in_port.open("/example/input:i")
    yarp.Network.connect("/icubSim/skinManager/skin_events:o",
     "/example/input:i")
def getData(self):
    btl = self.in_port.read(True)
    my_data = btl.toString()
    mydata = my_data.split('((')
    cis = ""
    ok = False
  for dat in mydata:
    if (len(dat) > 10) and (" 6 4)" in dat) and " 11) " in dat:
         ok = True
  if ok:
     for dat in mydata:
       if len(dat) > 10:
         if " 6 1)" in dat: #lava ruka
           if " (48 " in dat: #prsty
             cis += "000 " ...
           if " 121 " in dat: # dlan
             cis += "005 " ...
         if " 4 2)" in dat: #lave predlaktie
           if " 299 300 " in dat: # vrchna cast predlaktia
             cis += "011 " ...
           if " 143 168 " in dat: #spodna cast predlaktia
             cis += "021 " ...
         #podobne prava koncatina
    return cis
```

Na obrázku 3.9 je znázornené priradenie identifikátorov k jednotlivým oblastiam. Obrázok zodpovedá ľavej končatine. Pre pravú končatinu platí obrázok osovo súmerný s týmto obrázkom. Ako jediný správny údaj o smere bolo možno použiť smer od lakťa k prstom, nakoľko súradnicové osi x a y sú zrotované o 180°okolo osi z v prípade pravej končatiny. Písmeno "x"sa nahrádza buď 0 pokiaľ ide o ľavú končatinu alebo 1 ak o pravú.



Obr. 3.9: Identifikátory oblastí na dlani a predlaktí.

3.6 Zhrnutie

Výsledkom tejto časti práce je vylepšenie emulácie kože predlaktí iCub simulátora, ktoré spĺňa všetky zadané ciele. Využitím nie príliš zložitého prístupu sme navrhli a vytvorili algoritmus, ktorý nespomaľuje chod simulátora. V prípade budúcej zmeny v návrhu kože robota je kýmkoľvek ľahko modifikovateľný. Z nášho postupu a algoritmu je možné vychádzať pri ďalších zatiaľ nespracovaných častiach tela. Nevýhodou algoritmu je to, že nedosahuje takú presnosť ako skutočné dotykové senzory na reálnom iCub robotovi. To je však ťažko dosiahnuteľné a nebolo to naším cieľom.

Kapitola 4

Autonómna explorácia tela

Druhou časťou tejto diplomovej práce je autonómna explorácia tela. Cieľom bolo otestovať emuláciu kože v iCub simulátore a možnosti iCub simulátora samostatne vybudovať model vlastného tela na základe analýzy dotykových a motorických dát s minimálnym zásahom programátora.

4.1 Východiská

4.1.1 Priestory

Uvažujeme dva priestory: motorický M a senzorický S.

4.1.1.1 Motorický priestor

Všeobecne je motorický priestor, alebo priestor motorických povelov, priestor obsahujúci všetky rôzne povely, ktoré je robot schopný vykonať. Zložitejšie roboty sú väčšinou tvorené ramenom, alebo sústavou ramien, pozostávajúcich zo sekcií, ktoré sú upevnené na otočných kĺboch. Príkazy sú im zadávané vo forme nastavení jednotlivých kĺbov v stupňoch. Jedným bodom motorického priestoru je jedno nastavenie všetkých kĺbov robota, prípadne všetkých kĺbov jedného ramena.

Motorický priestor robota iCub je tvorený vektorom uhlov, v ktorých sú nastavené jednotlivé kĺby (motory). Rozsahy týchto motorov sú rôzne veľké intervaly patriace do intervalu $\langle -180°, 180° \rangle$.V rámci tejto práce využívame sedem motorov na ľavej hornej končatine a sedem motorov na pravej hornej končatine. Výsledný motorický priestor je tvorený $(m_0, ..., m_1 3)$.

Robot iCub má na každej hornej končatine viac ako týchto, v práci použitých, sedem motorov. Sú to motory pohybujúce prstami. Pri tomto experimente nastavenie prstov nemeníme. Tieto hodnoty ostávajú konštantné a môžeme ich vo výslednom vektore zanedbať.

4.1.1.2 Senzorický priestor

Na rozdiel od motorického priestoru, ktorý je pri všetkých humanoidných robotov podobný, senzorický priestor je širší pojem a u rôznych robotov môže byť rôzny. Senzorickým priestorom sa rozumie priestor výsledkov motorických akcií, ktoré robot pomocou svojich alebo externých senzorov zaznamená.

Pre ilustráciu si predstavme robotické rameno s viacerými sekciami, pohybujúce sa po ploche R^2. Na konci tohoto ramena je senzor, ktorý vie zaznamenať svoju aktuálnu polohu v rámci plochy. V tomto prípade bude teda senzorickým priestorom plocha R^2. Rôzne roboty vybavené rôznymi senzormi majú rôzne a rôzne zložité senzorické priestory. Napríklad robot vybavený stereokamerou, alebo hĺbkovým senzorom, má senzorický priestor R^3. Podľa počtu pozorovaných bodov má takýto robot senzorický priestor obsahujúci počet priestorov R^3 podľa počtu bodov, ktoré sleduje.

V tejto práci sledujeme dotyk špičky ukazováka pravej ruky o povrch ľavej hornej končatiny. Povrch ľavej hornej končatiny sa skladá z povrchu dlane a povrchu predlaktia. Povrch dlane je tvorený piatimi obdĺžnikovými plochami a povrch predlaktia je tvorený jedenástimi plochami tvaru zodpovedajúcemu prieniku povrchu predlaktia a kvádra, ktorý plochu ohraničuje v priestore R^3 podľa kapitoly 3. Zaujíma nás, či je daná plocha stlačená, teda či na nej prebieha dotyk špičkou ukazováka. Senzorický priestor je vektor (vzorec 4.1).

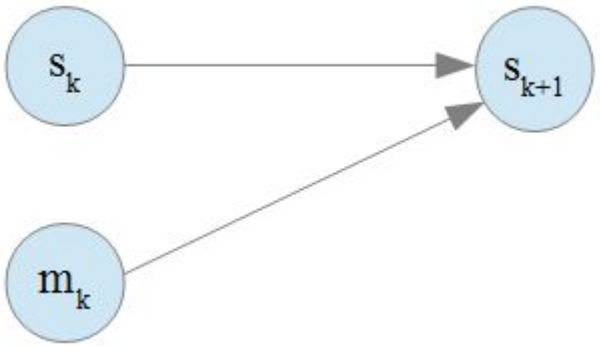
$$(s_0, s_1, ..., s_{15}), s_n = \begin{cases} 0, & \text{dotyk nenastal} \\ 1, & \text{dotyk nastal} \end{cases} \tag{4.1}$$

4.1.2 Senzomotorický model

Senzomotorické modely sa využívajú na predikciu v robotike aj v iných odboroch. Poznáme dva druhy senzomotorických modelov. Dopredný model a inverzný model [9].

4.1.2.1 Dopredný model

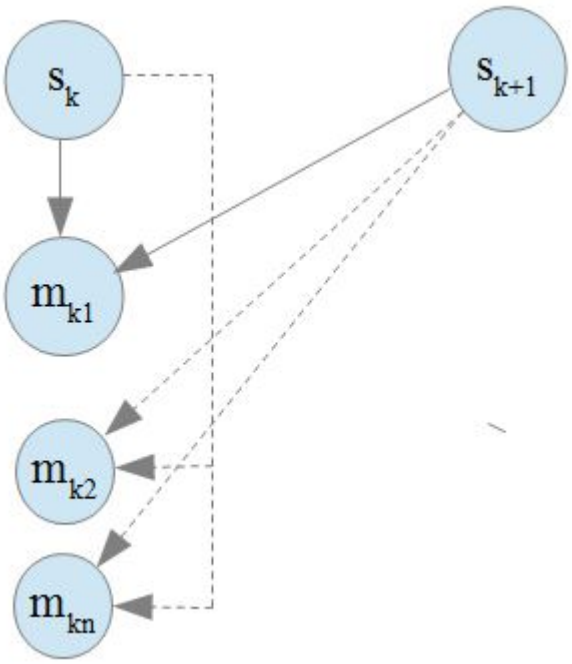
Poznáme aktuálny stav systému, a povel, ktorý sa v nasledujúcom kroku vykoná. Dopredný model z týchto dát určí nasledujúci stav systému. Je to jednoznačné zobrazenie $f(s_1 \in S, m \in M) = s_2 \in S$, pretože systém nemôže byť vo viacerých stavoch súčasne (obr 4.1).



Obr. 4.1: Dopredný model.

4.1.2.2 Inverzný model

Poznáme aktuálny stav systému, a stav systému, ktorý chceme v nasledujúcom kroku docieliť. Inverzný model z týchto dát určí povel, ktorý treba na dosiahnutie stavu vykonať. Je to o nejednoznačné zobrazenie $f^{-1}(s_1, s_2 \in S) = (m_1, ..., m_n) \in M$, pretože viacero rôznych povelov môže vyústiť do toho istého stavu systému (obr 4.2).



Obr. 4.2: Inverzný model.

4.1.3 Exploračné stratégie

Pre autonómnu exploráciu tela je dôležité určiť exploračnú stratégiu, čiže spôsob akým budeme voliť ďalší krok explorácie na základe už získaných priebežných výsledkov. Vychádzame zo štyroch stratégií popísaných vo [4].

4.1.3.1 Náhodná motorická explorácia (ACTUATOR-RANDOM)

V každom kroku si robot zvolí náhodný motorický povel z motorického priestoru $m \in M$. Tento povel vykoná. Pozoruje senzorickú informáciu ako výsledok motorickej akcie $s = f(m)$. Aktualizuje senzomotorický model o dvojicu (m, s). Nevýhodou tejto stratégie je to, že neanalyzuje výsledky predošlých krokov, čoho dôsledkom môže byť dlhšie prehľadávanie. Na druhej strane je táto stratégia výhodná pokiaľ nezáleží na čase prehľadávania, ale chceme získať čo najviac rôznych pozorovaní.

4.1.3.2 Náhodná explorácia cieľov (SAGG-RANDOM)

V každom kroku si robot zvolí náhodný cieľ zo senzorického priestoru $g \in S$. Pomocou inverzného modelu naučeného v predošlých krokoch zvolí motorický

povel $m \in M$. Tento povel vykoná. Pozoruje senzorickú informáciu ako výsledok motorickej akcie $s = f(m)$. Aktualizuje senzomotorický model o dvojicu (m, s). Oproti ACTUATOR-RANDOM, tu už vieme zmenšiť priestor prehľadávania a cielene prehľadávať oblasti, ktoré neboli prehľadané, čo niekoľkonásobne (podľa charakteristiky zvolených priestorov) zvyšuje rýchlosť prehľadávania. Nevýhodou tejto stratégie je to, že pri aplikácii na zložitejšie roboty ako je iCub s veľkým motorickým priestorom a naopak malým senzorickým priestorom, všetky nasledujúce výsledky explorácie závisia od prvej zaznamenanej senzomotorickej dvojice, a všetky zaznamenané motorické akcie budú podobné.

4.1.3.3 Aktívna motorická explorácia (ACTUATOR-RIAC)

V každom kroku si robot zvolí motorický povel z motorického priestoru $m \in M$ tak, aby na základe predošlých pozorovaní maximalizoval mieru záujmu v motorickom priestore M. Miera záujmu závisí od toho, nakoľko je okolie danej oblasti preskúmané. Tento povel vykoná. Pozoruje senzorickú informáciu ako výsledok motorickej akcie $s = f(m)$. Aktualizuje senzomotorický model o dvojicu (m, s). Aktualizuje mieru záujmu v danej oblasti. Miera záujmu je číselná informácia určujúca, na koľko v danej oblasti má význam prehľadávať. Málo zmapované oblasti majú vysokú mieru záujmu, dobre zmapované nízku. Oproti ACTUATOR-RANDOM je čas behu prehľadávania nižší, nakoľko sa nemôže stať, že by sme duplicitne prehľadávali dobre zmapované oblasti v priestore M. Efektivita voči SAGG-RANDOM závisí od charakteru priestoru S.

4.1.3.4 Aktívna explorácia cieľov (SAGG-RIAC)

V každom kroku si robot zvolí cieľ zo senzorického priestoru $g \in S$ tak, aby na základe predošlých pozorovaní maximalizoval mieru záujmu v senzorickom priestore S. Pomocou inverzného modelu, naučeného v predošlých krokoch, zvolí motorický povel $m \in M$. Tento povel vykoná. Pozoruje senzorickú informáciu ako výsledok motorickej akcie $s = f(m)$. Aktualizuje senzomotorický model o dvojicu (m, s). Aktualizuje mieru záujmu v danej oblasti.[6] V prípade veľkých a komplikovaných priestorov M a S je táto stratégia najefektívnejšia. Avšak pri aplikácii na jednoduchšie priestory sa efektivitou nelíši od SAGG-RANDOM.

4.2 Špecifikácia cieľa

Navrhnúť, implementovať a otestovať exploračné stratégie vychádzajúce zo stratégií popísaných v [4]. Vychádzajúc z výskumov o vývoji novorodencov popísaných v [10]. Na príkladoch dvojitého dotyku s jednou hornou končatinou vo fixnej polohe a dvojitého dotyku s oboma končatinami aktívnymi, sledujeme dotyk ukazováka pravej ruky o plochu na ľavej hornej končatine (predlaktie a dlaň). Zadávanie príkazov je povolené vo forme nastavení jednotlivých kĺbov končatín v stupňoch. Zo senzorov je dovolené použiť iba umelú kožu. A to z dôvodu, že zrak novorodencov nie je dostatočne vyvinutý. Preto nepoužijeme stereo kameru.

4.3 Riešenie

Riešenie problému je rozdelené do troch častí: Návrh, Implementácia a Testovanie. Návrh pozostával z analýzy využiteľných exploračných stratégií a knižníc, využiteľných senzorov a ovládačov. Implementácia prebiehala v jazyku Python. Testovanie pozostávalo z odskúšania jednotlivých stratégií pri rôznych nastaveniach.

4.3.1 Návrh

Sledujeme dotyk ukazováka pravej ruky na ploche ľavej hornej končatiny (predlaktie a dlaň), podobne ako v [5]. Ale zásadným rozdielom je to, že algoritmus pracuje autonómne, čiže bez zásahu človeka. Uskutočníme dva experimenty. V prvom je ľavá horná končatina statická. V druhom je ľavá horná končatina pohyblivá.

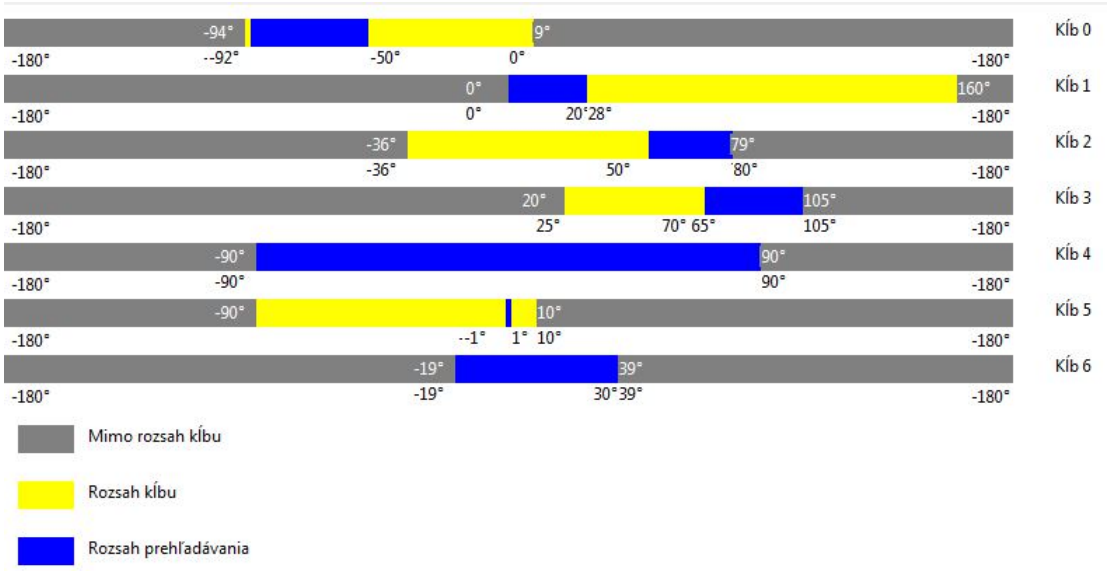
4.3.1.1 Motorický priestor

4.3.1.1.1 Experiment 1

Ľavú hornú končatinu fixujeme v presne určenej polohe. Hýbeme iba pravou hornou končatinou. Na pravej hornej končatine využívame prvých sedem motorov, čiže ramenný, lakťový a zápästný kĺb. Ostatné kĺby sú vo fixnej základnej polohe. Motorický priestor teda pozostáva z vektorov $m = (j_0, j_1, j_2, j_3, j_4, j_5, j_6) | m \in M$.

Nakoľko je rozsah hornej končatiny veľký, limitujeme rozsahy jednotlivých kĺbov tak, že pre každé nastavenie daného kĺbu patriaceho do limitu existuje aspoň jedno nastavenie ostatných kĺbov, ktoré produkuje dotyk na ľavej hornej končatine. Limity sme stanovili

na základe pozorovania. Týmto sme vylúčili množstvo konfigurácií, ktoré nemá zmysel prehľadávať a zrýchlili sme samotné prehľadávanie. Na obrázku 4.3 je znázornený pôvodný rozsah jednotlivých kĺbov a nami stanovený limit.

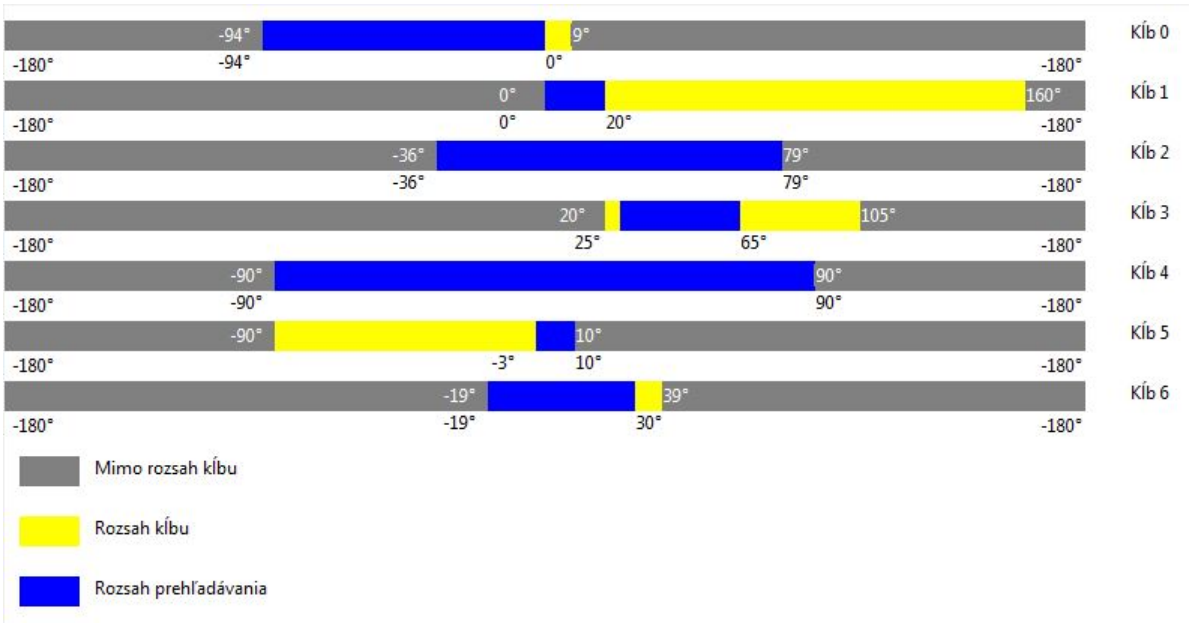


Obr. 4.3: Obmedzenie rozsahu kĺbov pri prehľadávaní jednou končatinou.

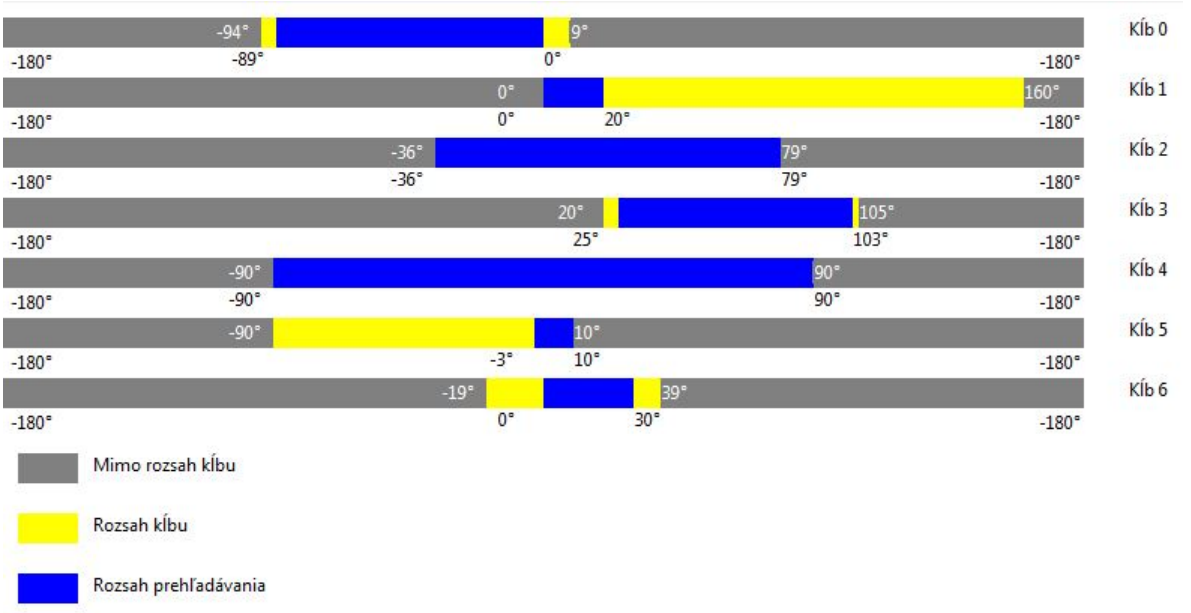
4.3.1.1.2 Experiment 2

Obe horné končatiny sú pohyblivé. Najprv pohneme ľavou, následne pravou. Na oboch horných končatinách využívame prvých sedem kĺbov. Ostatné kĺby sú vo fixnej základnej polohe. Motorický priestor teda pozostáva z vektorov dĺžky štrnásť $m = (j_0, j_1, j_2, j_3, j_4, j_5, j_6, k_0, k_1, k_2, k_3, k_4, k_5, k_6), m \in M$.

Nakoľko je rozsah hornej končatiny veľký, limitujeme rozsahy jednotlivých kĺbov tak, že sa snažíme aby výsledná poloha horných končatín bola pred telom, kde môže nastať dotyk a nie vedľa tela, kde dotyk nastať nemôže. Týmto sme vylúčili množstvo konfigurácii, ktoré nemá zmysel prehľadávať a zrýchlili sme samotné prehľadávanie. Na obrázku 4.4 je znázornený rozsah a nami stanovený limit pre ľavú končatinu a na obrázku 4.5 pre pravú končatinu.



Obr. 4.4: Obmedzenie rozsahu kĺbov pri prehľadávaní oboma končatinami - ľavá končatina.



Obr. 4.5: Obmedzenie rozsahu kĺbov pri prehľadávaní oboma končatinami - pravá končatina.

4.3.1.2 Senzorický priestor

V oboch experimentoch je totožný. Skladá sa z vektorov dimenzie šestnásť. Jeden člen vektora reprezentuje jednu oblasť na ľavom predlaktí alebo dlani. Členy nadobúdajú hodnoty 0, ak dotyk na danom mieste nenastal, alebo 1, ak dotyk nastal. $s = (c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11}, c_{12}, c_{13}, c_{14}, c_{15}), s \in S$.

4.3.1.3 Exploračné stratégie

Využijeme exploračné stratégie ACTUATOR-RANDOM a SAGG-RANDOM.
ACTUATOR-RANDOM bude základná stratégia na určenie prvého dotyku. Náhodne generujeme motorické povely, kým nenastane dotyk. Následne, keď poznáme prvú senzomotorickú dvojicu (s, m), môžeme buď pokračovať v ACTUATOR-RANDOM, alebo prejsť na SAGG-RANDOM.

V prípade SAGG-RANDOM potrebujeme poznať jednu senzomotorickú dvojicu. Zvolíme niektorú zo zatiaľ neobjavených oblastí, ktorej susedná oblasť je už objavená a zistíme priemerný motorický povel potrebný na dosiahnutie niektorej z objavených susedných oblastí hľadanej oblasti. Na základe tohoto povelu náhodne vygenerujeme podobný povel a zaznamenáme výsledok. Postup opakujeme dovtedy, kým nepoznáme všetky dvojice.

ACTUATOR-RIAC a SAGG-RIAC nevyužijeme. Dôvodom je problém s určením miery záujmu. Tieto stratégie sú primárne navrhnuté pre spojité priestory reálnych čísel. V takýchto priestoroch v každom kroku vyprodukujeme senzomotorickú dvojicu. V našom prípade máme síce spojitý motorický priestor reálnych čísel, ale senzorický priestor je priestor šestnástich diskrétnych hodnôt. V prípade robota iCub vyprodukujeme dotyk a k nemu prislúchajúcu senzomotorickú dvojicu iba výnimočne. Väčšina povelov dotyk nevyprodukuje.

4.3.2 Implementácia

Implementácia prebiehala v jazyku Python, a to z dvoch dôvodov. Prvým bol pôvodný zámer použiť knižnicu Explauto [11]. Je to knižnica implementujúca autonómnu exploráciu podľa [4]. Avšak nakoniec sme zistili, že by bolo potrebné prispôsobiť náš kód knižnici natoľko, až sa jej využitie stalo zbytočným a algoritmy, ktoré sme plánovali

použiť z tejto knižnice, sme implementovali sami, priamo podľa potrieb a možností iCub simulátora.

Druhým dôvodom je výhoda jazyka Python oproti jazyku C++, ktorý sme zvažovali ako alternatívu to, že ide o interpretovaný jazyk, čiže po úprave kódu nie je potrebná kompilácia. Ďalej je to o beztypový jazyk, čiže úprava zdrojového kódu je jednoduchšia, a samotný kód je kratší.

Nasledujúce ukážky pozostávajúce z kombinácie pseudokódu a jazyka Python vysvetľujú fungovanie použitých algoritmov. Všetky zdrojové kódy sú súčasťou prílohy a sú zverejnené na [18].

Najskôr zadefinujeme senzomotorickú dvojicu. Tá pozostáva z rôzne dlhého poľa ID oblastí na koži. Rôzne dlhého preto, že dotyk môže nastať naraz vo viacerých oblastiach. Ďalej zaznamenávame nastavenie kĺbov ako kópiu objektu, ktorý určuje ich aktuálne nastavenie. Je to 16-miestne pole. Zadefinujeme senzomotorický model ako pole. Toto pole bude obsahovať senzomotorické dvojice.

Nadviaže sa spojenie so simulátorom prostredníctvom YARP. Následne definujeme prvú pozíciu, do ktorej robota nastavíme, a počiatočnú pozíciu, do ktorej sa bude po vykonaní povelu a odčítaní výsledku vracať. Nasleduje implementácia samotnej exploračnej stratégie. Tie sú popísané každá zvlášť. Algoritmus končí vypísaním výsledkov na štandardný výstup a do súboru (ukážka 4.1).

Ukážka 4.1: Senzomotorická dvojica v Python.

```
class senso_motor_pair:
  def __init__(self, ID, j):
    self.skin_cube_ID = str(ID.split())
    self.joints = copy.deepcopy(j)
    return

  def write(self):
    vypise obsah objektu do konzoly
    return

  def writefile(self):
    pripravi obsah objektu na zapis do suboru
    return vysledok

model = []

#Nadviazeme spojenie s YARP
#Estabilish YARP connection

tmp=yarp.Vector(jnts)
tmp.set(0, val)
... # 0-7
hom=yarp.Vector(jnts)
hom.set(0, val)
... # 0-7

#Exploracna strategia
#Exploration strategy

model[i].write()
```

4.3.2.1 Experiment 1

4.3.2.1.1 ACTUATOR-RANDOM

Náhodnú motorickú exploráciu s ľavou hornou končatinou vo fixnej polohe implementujeme nasledovne. Definujeme nastavenie kĺbov aktívnej pravej hornej končatiny a pole známych, už objavených oblastí. Na začiatku sú všetky oblasti neobjavené. V cykle najprv nastavíme všetkých sedem kĺbov na náhodnú hodnotu

v rozmedzí určenom podľa návrhu. Vykonáme povel a po troch sekundách odčítame výsledok. Pokiaľ výsledok obsahuje senzorickú informáciu, tak do senzomotorického modelu pridáme novú senzomotorickú dvojicu. Následne prevedieme kódy zasiahnutých oblastí na index a zaznačíme objavené oblasti do poľa známych oblastí. Pokiaľ má pole známych oblastí dostatok záznamov, čiže sme našli všetky dosiahnuteľné oblasti v danej konfigurácii, explorácia končí. Ak nie, nastavíme počiatočnú polohu a explorácia pokračuje (ukážka 4.2).

Ukážka 4.2: ACTUATOR-RANDOM v Python.
Originál: [18]/arm_single/motor-babling/motor-babling2.py

```
joint_setting= [-30, 15, 40, 70, 20, 0, 0]
zname = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] #objavene oblasti
semafor = True
x = 0
while semafor:
  x += 1
  for i in range (0, 7):
    r = random.uniform(min, max)
    tmp.set(i, r)
    joint_setting[i] = r

  iPos.positionMove(tmp.data())#zadanie povelu

  time.sleep(3) #cakame prebieha vykonavanie povelu
  dat = citac.getData() #prijem senzorickej informacie
  if dat != "":
    model.append(senso_motor_pair(dat, joint_setting))
    d = dat.split()
    for item in d:
      if item == "011":
        zname[0] = 1
      elif item == "012":
        zname[1] = 1
      ... #range(0,16)

    #kontrola zastavenia prehladavania
    suma = 0
    for j in range(0, 16):
      suma += zname[j]
    if suma > 11:
      semafor = False

  #domovka pozicia
  iPos.positionMove(hom.data())
  time.sleep(2)
```

4.3.2.1.2 SAGG-RANDOM

Vychádzame z algoritmu ACTUATOR-RANDOM. Najprv pomocou tohto algoritmu nájdeme prvú náhodnú senzomotorickú dvojicu. Následne pokračujeme v cielenom prehľadávaní.

Najprv určíme oblasť, ktorú chceme hľadať. Musí to byť taká, ktorá má už objavenú aspoň jednu susednú oblasť. Vychádzame z matice susednosti, ktorá je navrhnutá tak, že susedné oblasti sú tie, medzi ktorými nie je žiadna iná oblasť, pokiaľ sa pohybujeme po povrchu končatiny do ľubovolného smeru. Jedna oblasť môže mať v našom prípade aj sedem susedných oblastí. Oblasť určujeme náhodne.

Zistíme priemernú motorickú zložku senzomotorických dvojíc pre vybrané susedné oblasti. Z oblastí náhodne vyberieme jednu, ktorej priemernú motorickú zložku použijeme ako základ pre nový povel. Nový povel vyprodukujeme úpravou starého a to v každom kĺbe o náhodnú odchýlku v určitom malom rozmedzí.

Rovnako ako pri ACTUATOR-RANDOM vykonáme povel, odčítame výsledok, a ak môžeme, pridáme senzomotorickú dvojicu do modelu. Postup opakujeme kým nenájdeme všetky dosiahnuteľné oblasti (ukážka 4.3).

Ukážka 4.3: SAGG-RANDOM v Python.
Originál: [18]/arm_single/goal-babling2/goal-babling.py

```
#ACUTATOR-RANDOM
#pokial nenajdeme prvy senzomotoricky par


while True:
  #vyberieme neznamu oblast s aspon 1 znamim kamaratom
  vybrana = False
  i = -1
  kamaratiindexy = [] #indexy znamich susednych oblasti
  while vybrana == False:
    i = int(random.uniform(0,16))
    kamaratiindexy = distmat.getkamaratov(i, zname)
    if len(kamaratiindexy) > 0 and zname[i] == 0:
      vybrana = True

  #pre vybranu oblast zistime priemernu motoricku zlozku susednych oblasti
  kamarati = []
  for item in kamaratiindexy:
    kamarati.append(getaveragemotor(item))

  #vyberieme nahodnu znamu susednu oblast
  a = int(random.uniform(0, len(kamarati)))
  item = kamarati[a]

  #nahodne mierne upravime motoricky povel vybranej oblasti a vykoname
  for a in range(0,7):
    r = random.uniform(float(item[a]) + err, float(item[a]) - err)
    tmp.set(a, r)
    joint_setting[a] = r

  #vykoname povel
  iPos.positionMove(tmp.data())
  #analizujeme vysledok
  #opakujeme
```

4.3.2.2 Experiment 2

4.3.2.2.1 ACTUATOR-RANDOM

Postupovali sme rovnako, ako v prípade experimentu 1. Rozdiel je v tom, že pohybujeme oboma končatinami. Preto má motorická zložka senzomotorickej dvojice a aj povel dvojnásobnú dĺžku. Najprv nastavíme ľavú končatinu do náhodnej polohy, následne vykonáme náhodný povel pravou končatinou. Nakoľko je pohyb regulovaný minimálne, pridali sme kontrolu správnosti vykonania daného povelu. Po vykonaní povelu odčítame z YARP aktuálne nastavenie pravej končatiny, nakoľko sa povel nemusel pri kontakte s inou časťou tela vykonať správne. Spracujeme a zaznamenáme výsledok a pokračujeme v prehľadávaní, pokým nenájdeme všetky oblasti (ukážka 4.4).

Ukážka 4.4: ACTUATOR-RANDOM pre obe končatiny v Python.
Originál: [18]/arms_both/motor-babling/motor-babling.py

```
#ACUTATOR-RANDOM podobne ako pri experimente 1
joint_setting= [-30, 15, 40, 70, 20, 0, 0, -30, 15, 40, 70, 20, 0, 0]
semafor = True
x = 0
while semafor:
  # prava
  for x in range(0,7):
    r = random.uniform(min, max)
    joint_setting[x] = r
    tmp.set(x, r)
  #lava
  for x in range(0,7):
    r = random.uniform(min, max)
    joint_setting[x+7] = r
    tmp2.set(x, r)
  iPos2.positionMove(tmp2.data())#lava
  time.sleep(1)
  iPos.positionMove(tmp.data())#prava
  time.sleep(3)
  dat = citac.getData()
  #kontrola spravneho vykonania
  nastavenie = rightcitac.getData(); #aktualne nastavenie iCubSim
      for a in range(0,7):
        joint_setting[a] = float(nastavenie[a])
  #spracovanie vysledku
```

4.3.2.2.2 SAGG-RANDOM

Kombinujeme postup pre SAGG-RANDOM z prvého experimentu a ACTUATOR-RANDOM z druhého experimentu.

Po nájdení prvej oblasti začíname cielene hľadať. Postupujeme rovnako ako pri SAGG-RANDOM v experimente 1 s tým rozdielom, že motorická zložka je dvojnásobne dlhá. Pri generovaní náhodného podobného povelu mierne náhodne upravíme aj nastavenie ľavej končatiny. Skontrolujeme správnosť vykonania povelu. Spracujeme a zaznamenáme výsledok a pokračujeme v prehľadávaní pokým nenájdeme všetky oblasti(ukážka 4.5).

Ukážka 4.5: SAGG-RANDOM pre obe končatiny v Python.
Originál: [18]/arms_both/goal-babling-randomer/goal-babling.py

```
#ACTUATOR-RANDOM po prvy (s,m) par
#Vyber ciela SAGG-RANDOM

#urcime povel pre obe ruky
#prava
u = 4
for x in range(0,7):
  r = random.uniform(float(item[x]) + u, float(item[x]) - u)
  tmp.set(x, r)
  joint_setting[x] = r
#lava
v = 2;
for x in range(0,7):
  r = random.uniform(float(item[x+7]) + v, float(item[x+7]) - v)
  joint_setting[x+7] = r
  tmp2.set(x, r)

#vykonanie
#kontrola spravneho vykonania
#spracovanie vysledku
#ako pri ACTUATOR-RANDOM
```

4.3.3 Testovanie

Testovanie pozostávalo z vyskúšania jednotlivých stratégii v iCub simulátore. Podľa získaných výsledkov a pozorovaného správania modelu robota v simulátore

sme pristupovali k úprave parametrov a ďalšiemu testovaniu. Všetky časy behov prehľadávania sú nezávislé na výkone počítača, a to z toho dôvodu, že vyhodnotenie výsledkov nie je časovo náročné, ale čakáme na presun robota do novej pozície a jeho ustálenie. To predstavuje v jednom kroku prehľadávania tri sekundy pri presune na určenú pozíciu a dve sekundy na presun na počiatočnú pozíciu. Pri presune na určenú pozíciu je čas o sekundu dlhší preto, lebo môže dojsť ku kolízii končatín a model robota zapruží. Preto čakáme dlhšie, kým sa ustáli.

4.3.3.1 Experiment 1

4.3.3.1.1 ACTUATOR-RANDOM

Pri tomto experimente bolo potrebné správne určiť polohu pasívnej končatiny tak, aby bola v polohe pri ktorej môže nastať dotyk na čo najväčšom počte oblastí. Zvolili sme polohu pred telom s miernym ohnutím v lakti. Počas pokusu sme pozorovali, že častejšie nastane dotyk na predlaktí. Je to spôsobené tým, že predlaktie je v strede prehľadávanej oblasti a použitím náhodného výberu povelov často vyberieme povel do stredu oblasti. Dlaň je menšia a na okraji dosahu aktívnej končatiny. Existuje preto menej povelov, ktoré by vyústili do dotyku dlane. Na dosiahnutie všetkých dosiahnuteľných oblastí pri tomto nastavení a stratégii je potrebných približne 700 pokusov.

4.3.3.1.2 SAGG-RANDOM

Tu sme sa stretli s problémom správneho nastavenia rozsahu, v ktorom budeme generovať náhodný povel podobný povelu z ktorého vychádzame. Pokiaľ zvolíme malý rozsah, budeme stále generovať dotyky v oblasti kde nastal prvý dotyk a ďalej sa prakticky nedostaneme. Vyplýva to zo štruktúry povrchu, ktorý prehľadávame. Najbližších susedov nájdeme pomocou podobného povelu, ale suseda suseda opäť hľadáme pomocou povelu, ktorý vychádza z pôvodného. Avšak táto oblasť je väčšinou na opačnej strane danej časti tela a potrebujeme vygenerovať natoľko odlišný povel, že sa ho už model sám nedokáže naučiť.

Naopak, ak zvolíme príliš veľký rozsah, karteziánsky priestor v ktorom prehľadávame sa zväčší. Dostaneme sa do momentu, že algoritmus nemôžeme označiť ako prehľadávanie SAGG-RANDOM ale ako ACTUATOR-RANDOM. Je to preto, že v danom priestore sa nachádza už celý povrch končatiny, ktorú prehľadávame.

Na úspešnosť prehľadávania výrazne vplýva aj počiatočná poloha končatiny, do ktorej sa končatina vracia po vykonaní povelu. Aby sme predišli zaseknutiu a ovplyvňovaniu výsledkov predošlým krokom prehľadávania, končatinu musíme vždy vrátiť na počiatočnú pozíciu. Počiatočnú polohu je najlepšie určiť tak, aby nastávalo čo najmenej kolízii ešte pri prechode na určenú pozíciu, teda aby bol ukazovák aktívnej končatiny približne vo výške pasívnej končatiny.

Pri najlepších nastaveniach aké sa nám podarilo dosiahnuť, sme dvanásť dosiahnuteľných oblastí z pôvodných šestnástich dosiahli pri prvom pokuse v 125-om kroku od začiatku SAGG-RANDOM, a v druhom pokuse v 195-om kroku.

4.3.3.2 Experiment 2

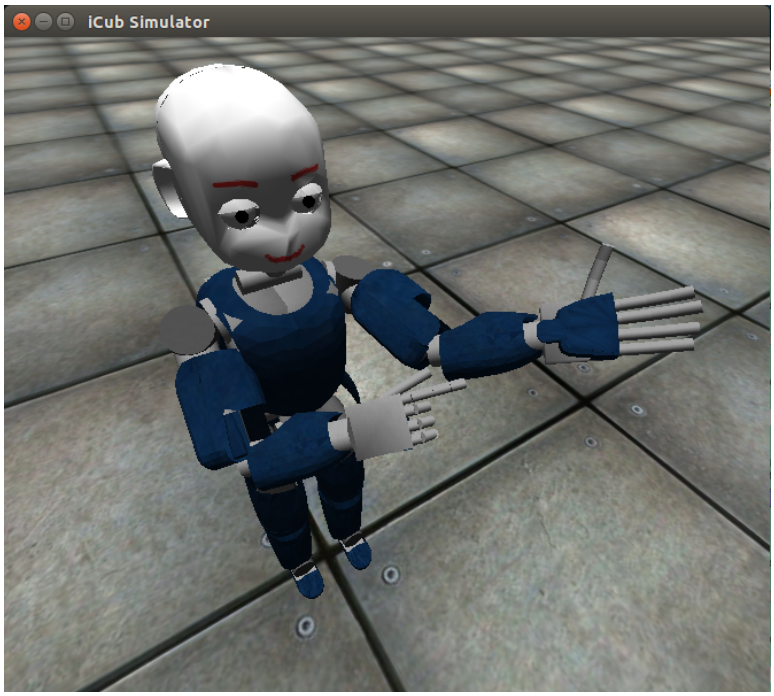
4.3.3.2.1 ACTUATOR-RANDOM

Z dôvodu časovej náročnosti tohoto algoritmu boli vykonané iba dva pokusy. Prvý pokus skončil neúspešne zaseknutím robota v neštandardnej polohe rukou zakliesnenou o panvu. Ide o chybu implementácie kinematiky samotného simulátora. Takéto chyby môžu nastať zriedkavo, ale je potrebné ich brať do úvahy. Pri testoch na reálnom robotovi, s prihliadnutím na cenu jeho hardvéru, navrhujeme použiť pri prehľadávaní automatické kontroly, aby k takýmto situáciám nemohlo dojsť, nakoľko môžu vyústiť do poruchy či zničenia niektorých komponentov.

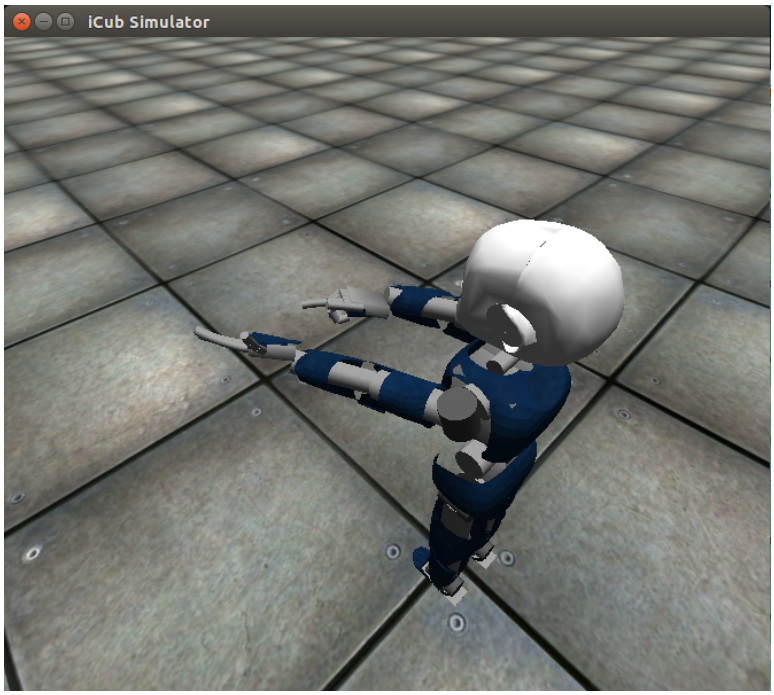
Druhý pokus skončil úspešne. Zasiahli sme všetky oblasti. Avšak časová náročnosť tohoto algoritmu je približne 90 minút. Po 500 pokusoch bolo objavených 13 zo 16 oblastí. Po 650 pokusoch bolo objavených 14 zo 16 oblastí. Po 900 pokusoch bolo objavených 15 zo 16 oblastí. Až po 1400 pokusoch boli objavené všetky oblasti.

4.3.3.2.2 SAGG-RANDOM

Prvým problémom s ktorým sme sa stretli, je nastavenie ľavej končatiny pri prvom dotyku. Môže sa stať, a často sa aj stáva, že dosiahneme dotyk v takom nastavení, že vzdialenejšie oblasti sú mimo dosahu pravej končatiny (obr 4.6, 4.7). Riešením je zvýšiť voľnosť ľavej končatiny. Avšak z toho plynú ďalšie nevýhody.



Obr. 4.6: Prvý dotyk v ťažšie prehľadávateľnom nastavení.



Obr. 4.7: Nastavenie pri ktorom nevieme dosiahnuť na dlaň.

Voľnosť ľavej končatiny je problematická podobne, ako to bolo pri Experimente 1 s pravou končatinou. Pokiaľ je voľnosť malá, nedosiahneme všetky oblasti, a v zásade ide o prehľadávanie jednou končatinou. Pokiaľ nastavíme veľkú voľnosť, dramaticky predlžujeme beh algoritmu a blížime sa k motorickému prehľadávaniu.

Ako dobré vylepšenie sa ukázalo ponechať ľavú končatinu skoro statickú, až na kĺb, ktorý ovláda rotáciu predlaktia. Takto otáčame predlaktie aj dlaň okolo osi y. Tým dosiahneme viac rôznych dotykov pri rovnakom nastavení pravej ruky. Pri tomto nastavení sa nám podarilo objaviť všetky oblasti v 271. kroku prehľadávania.

4.3.3.3 Všeobecné problémy pri explorácii

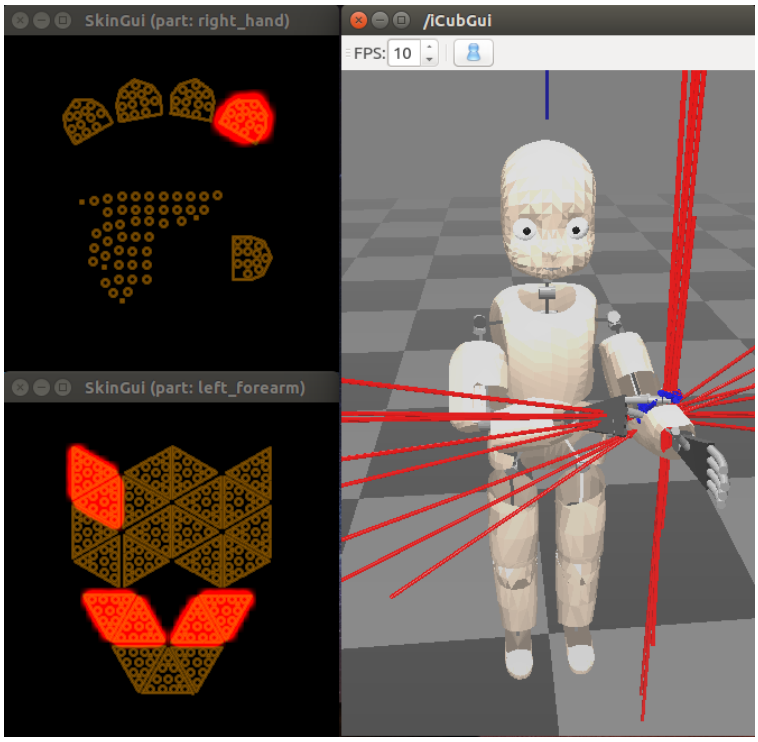
Pozorovali sme dve kĺbom oddelené časti tela, dlaň a predlaktie. Pri SAGG-RANDOM, či v experimente 1 alebo 2, je problematický prechod medzi týmito časťami tela. Medzi jednotlivými časťami tela je medzera, oblasti ohraničujúce dotykové platničky na seba nenadväzujú, zatiaľ čo v rámci jedenej časti tela tieto oblasti na seba nadväzujú. Väčšinou platí, že dlaň je od trupu viac vzdialená ako predlaktie. To má dosah na presnosť prehľadávania pomocou pravej ruky. Pokiaľ prehľadávame na predlaktí, jemné prehľadávanie je najlepšie robiť jemným nastavením lakťa. Pokiaľ prestavíme rameno, prst sa posunie príliš ďaleko. U dlane je to naopak. Lakeť je vystretý a viac pohybujeme ramenom. Riešením je prehľadávanie počas behu algoritmu rozdeliť na dve časti, a každú časť tela prehľadávať pomocou iných parametrov. Tým ale nenavrhneme všeobecne použiteľný robustný algoritmus. Snažili sme sa postupovať tak, aby sme našli hodnoty parametrov čo najviac vyhovujúce obom častiam tela.

Počas prehľadávania sa generujú dotyky aj inými časťami pravej končatiny ako ukazovákom. Tomuto správaniu sa nedá predísť. Môžeme sa iba pokúšať filtrovať výsledky. Avšak to je možné iba do určitej miery. Filtrovať výsledky môžeme tak, že budeme zisťovať či nenastal dotyk na pravej končatine aj inde ako na ukazováku. To však neodfiltruje všetky nechcené dotyky, pretože dotyk môže nastať aj časťou tela, ktorá nie je pokrytá kožou, napríklad článkami prstov. Iným riešením by bolo filtrovať dotyky, ktoré na ľavej ruke aktivujú dve oblasti, ktoré nesusedia. Otázne však pri použití filtrácie je, že či naozaj chceme takéto dotyky filtrovať. Získali sme totižto informáciu s ktorou vieme ďalej pracovať. Dá sa zaradiť do dopredného modelu ako špeciálny prípad. V rámci inverzného modelu by bola táto informácia tiež využiteľná, lebo iba

malou úpravou motorickej zložky, máme vysokú šancu získať dotyk iba ukazovákom.

Na obrázku 4.8 je znázornené nastavenie kedy sa ľavého predlaktia dotýkame zároveň pravým ukazovákom a hornou stranou ostatných prstov, ktorá nie je pokrytá kožou. Na obrázku 4.9 je znázornené spracovanie tejto situácie simulátorom vizualizované pomocou iCubSkinGui.

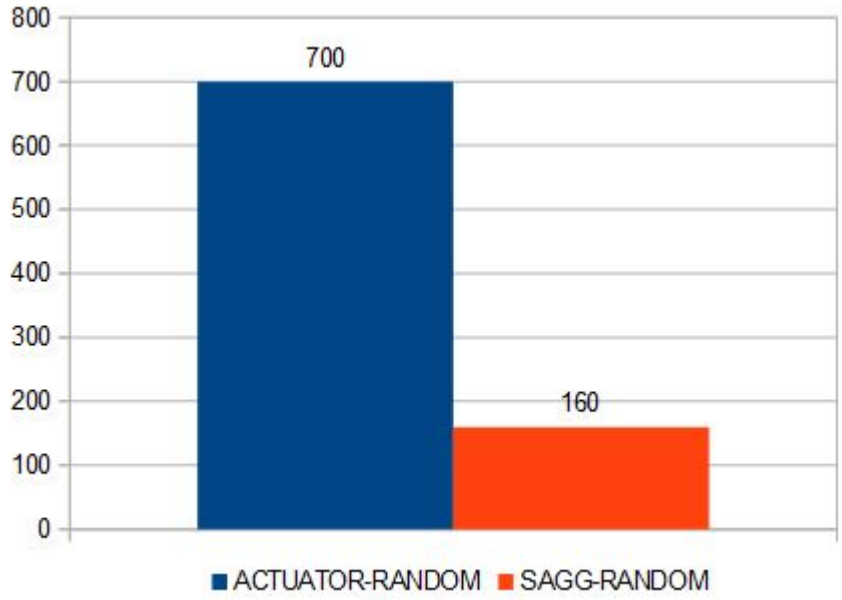
Obr. 4.8: Nastavenie simulátora do polohy produkujúcej ťažko spracovateľnú dotykovú informáciu.



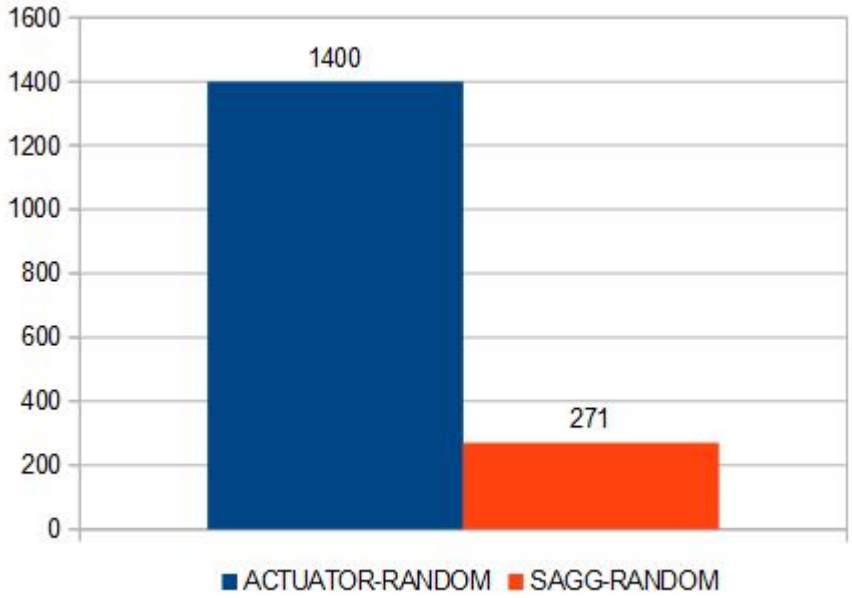
Obr. 4.9: Ťažko spracovateľná dotyková informácia.

4.4 Vyhodnotenie

Výsledky testovania ukázali, že stratégie zamerané na prehľadávanie cieľov dokážu, v porovnaní s motorickými stratégiami, výrazne urýchliť proces prehľadávania u iCub simulátora. Dôležitá je však správna voľba parametrov pri prehľadávaní. Pokiaľ zvolíme zlé parametre, prehľadávanie skončí neúspešne. Grafické porovnanie priemernej nameranej dĺžky behu prehľadávania ACTUATOR-RANDOM a priemernej nameranej dĺžky behu prehľadávania SAGG-RANDOM pre najlepšie nami zvolené nastavenie parametrov je znázornené na obrázku 4.10 pre experiment 1 a na obrázku 4.11 pre experiment 2.



Obr. 4.10: Porovnanie dĺžky behu prehľadávania jednou končatinou u jednotlivých stratégií.



Obr. 4.11: Porovnanie dĺžky behu prehľadávania oboma končatinami u jednotlivých stratégií.

Pri ďalšom výskume nadväzujúcom na túto prácu, ktorého cieľom by malo byť vytvoriť bohatý senzomotorický model, navrhujeme vychádzať zo stratégie ACTUATOR-RANDOM aj SAGG-RANDOM. Tieto stratégie dokážu generovať rôzne výsledky a tým obohatiť senzomotorický model. Dôležité je implementovať bezpečnostné algoritmy zabraňujúce neželaným kolíziám. Ďalej je potrebné rozšíriť prehľadávanie pre všetky prsty. Presnejšie výsledky a kontrola behu algoritmu by sa dali zabezpečiť využitím obrazového výstupu stereokamery. Presnejšiu klasifikáciu senzomotorických párov by mohol zabezpečiť senzomotorický model v kombinácii s neurónovou sieťou.

Kapitola 5

Záver

Výsledkom tejto práce je funkčná emulácia kože na predlaktiach v iCub simulátore. Vďaka tejto emulácii sme sprístupnili generovanie dotykových dát pre celú vedeckú komunitu a každého, kto má záujem o výskum v tejto oblasti. Nedosiahli sme rozlíšenie aké má fyzický robot iCub, ale nami implementovaná emulácia poskytuje dostatočné rozlíšenie pre širokú škálu experimentov.

V druhej časti práce sme navrhli, implementovali a otestovali algoritmy určené na autonómnu exploráciu tela s využitím umelej kože. Výsledky ukázali perspektívu algoritmov zameraných na voľbu cieľov, ale aj potrebu vychádzať z pomalších, motoricky orientovaných algoritmov. Predpokladáme, že autonómna explorácia tela bude v budúcnosti kľúčová pri sériovej výrobe humanoidných robotov a ich údržbe.

Literatúra

[1] Metta, G., Natale, L., Nori, F., Sandini, G., Vernon, D., Fadiga, L., von Hofsten, C., Rosander, K., Lopes, M., Santos-Victor, J., Bernardino, A. and L. Montesano (2010). The iCub humanoid robot: An open-systems platform for research in cognitive development. Neural Networks 23(8-9), 1125-1134.

[2] Del Prete, A., Denei, S., Natale, L., Mastrogiovanni, F., Nori, F. and Cannata, G. (2011). Skin Spatial Calibration Using Force/Torque Measurements. IEEE/RSJ International Conference on Intelligent Robots and Systems, San Francisco, CA, USA.

[3] Tikhanoff, V., Cangelosi, A., Fitzpatrick, P., Metta, G., Natale, L. and Nori, F. (2008). An open-source simulator for cognitive robotics research: the prototype of the iCub humanoid robot simulator. Proceedings of the 8th workshop on performance metrics for intelligent systems 57-61.

[4] Moulin-Frier, C. and Oudeyer, P. Y. (2013). Exploration strategies in developmental robotics: a unified probabilistic framework. Development and Learning and Epigenetic Robotics (ICDL), IEEE Third Joint International Conference on. IEEE.

[5] Roncone, A., Hoffmann, M., Pattacini, U. and Metta, G. (2014). Automatic kinematic chain calibration using artificial skin: self-touch in the iCub humanoid robot. 'Proc. IEEE Int. Conf. Robotics and Automation (ICRA)', pp. 2305–2312.

[6] Baranes, A. and Oudeyer, P. Y. (2013). Active learning of inverse models with intrinsically motivated goal exploration in robots. Robotics and Autonomous Systems 61.1: 49-73.

[7] Maiolino,P., Ascia, A., Natale, L., Cannata, G., and Metta, G. (2012). Large Scale Capacitive Skin for Robots. Smart actuation and sensing systems - Recent

advances and future challenges. In Tech Ed. Berselli, G.; Vertechy, R.; Vassura, G. ISBN:979-953-307-990-4.

[8] Maiolino, P., Maggiali, M., Cannata, G., Metta, G. and Natale, L. (2013). A flexible and robust large scale capacitive tactile system for robots. Sensors Journal, IEEE 13(10), 3910–3917.

[9] Wolpert, D., Ghahramani, Z. and Jordan, M. (1995). An internal model for sensorimotor integration. Science 269 (5232): 1880–1882.

[10] Rochat, P. (1998). Self-perception and action in infancy. Experimental brain research 123.1-2: 102-109.

[11] Moulin-Frier, C., Rouanet, P., and Oudeyer, P. Y. (2014). Explauto: an open-source Python library to study autonomous exploration in developmental robotics. Development and Learning and Epigenetic Robotics (ICDL-Epirob), Joint IEEE International Conferences on. IEEE, 2014.

[12] iCub [online] http://www.icub.org (2.5.2016).

[13] Tactile sensors (aka Skin) [online] http://wiki.icub.org/wiki/Tactile_sensors_(aka_Skin) (2.5.2016).

[14] MATLAB [online] http://www.mathworks.com/products/matlab/ (2.5.2016).

[15] Python [online] https://www.python.org (2.5.2016).

[16] SWIG [online] http://www.swig.org (2.5.2016).

[17] Robotology [online] https://github.com/robotology (2.5.2016).

[18] Projektový repozitár [online] https://github.com/Tytam/martin-iCubSIM_skin (2.5.2016).

Príloha

Priložené CD obsahuje zdrojové kódy implementácií jednotlivých častí práce. Je kópiou projektového repozitára [18].