VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

INTERAKČNÍ METODY PRO POKROČILÉ OVLÁDÁNÍ ROBOTICKÉHO RAMENE

INTERACTION METHODS FOR ADVANCED CONTROL OF A ROBOTIC ARM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JURAJ VIDA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. ZDENĚK MATERNA**
SUPERVISOR

BRNO 2017

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Vida Juraj**
Obor: Informační technologie
Téma: **Interakční metody pro pokročilé ovládání robotického ramene**
Interaction Methods for Advanced Control of a Robotic Arm
Kategorie: Uživatelská rozhraní

Pokyny:

1. Seznamte se s metodami používanými k ovládání robotického ramene uživatelem.
2. Seznamte se s robotickým operačním systémem (ROS) a existujícími možnostmi ovládání ramen robota v rámci ARTable.
3. Proveďte analýzu a navrhněte interakční metody pro nastavení různých druhů omezení pohybu ramene uživatelem.
4. Realizujte navržené řešení.
5. Ověřte funkčnost a použitelnost řešení.
6. Porovnejte a zhodnoťte dosažené výsledky. Navrhněte vhodné způsoby využití vašeho řešení.

Literatura:

- Dle doporučení vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Materna Zdeněk, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Táto bakalárska práca sa zaoberá rozšírením funkcionality systému ARTable, ktorý je postavený na princípoch zjednodušeného programovania robotov. Tieto princípy majú za cieľ uľahčiť interakciu človeka s robotom. Pridáva novú inštrukciu, ktorá umožňuje naučiť robota ukladať objekty rovnakého typu, napríklad do krabice. Podporu novej inštrukcie tvoria dve časti (i) rozšírenie grafického užívateľského rozhrania a (ii) rozšírenie samotnej funkcionality robota. Rozšírenia sú implementované v jazyku Python s využitím framework-u Qt, robotieho operačného systému (ROS) a framework-u MoveIt!. Rozšírenie je plne funkčné, za predpokladu obmedzenia generovania úchopov len na y-ovej osi.

Abstract

The purpose of this thesis is to extend the functionality of ARTable system, which is based on principles of simplified robot programming. These principles aim to simplify the human-robot interaction. The new instruction is added, which enables to teach the robot to place objects of the same type into a grid, for example a box. This instruction consists of two parts: (i) the extension of GUI, and (ii) the extension of the robot functionality. These extensions are implemented in Python using Qt framework, The Robot Operating System (ROS) and MoveIt! framework. The extension of ARTable is fully functional assuming that grasps are generated only on the y axis sides.

Kľúčové slová

Zjednodušené programovanie robotov, ARTable, PR2, MoveIt!, GUI, Qt, manipulácia s objektom, ukladanie objektov, grid, interakcia človek-robot

Keywords

Simplified robot programming, ARTable, PR2, MoveIt!, GUI, Qt, object manipulation, placing objects, grid, human - robot interaction

Citácia

VIDA, Juraj. *Interakční metody pro pokročilé ovládání robotického ramene*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Zdeněk Materna

Interakční metody pro pokročilé ovládání robotického ramene

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením Ing. Zdeňka Materny. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Juraj Vida
17. mája 2017

Poďakovanie

Na tomto mieste by som rád poďakoval Ing. Zdeňkovi Maternovi za odborné vedenie, užitočné rady a pripomienky.

Obsah

Kapitola 1

Úvod

Táto práca sa zaoberá rozšírením výzkumného projektu skupiny Robo@Fit[1] - ARTable. Systém ARTable je miesto pre spoluprácu človeka s robotom. Projekt je postavený na princípoch zjednodušeného programovania - človek jednoduchým spôsobom naučí robota, čo a ako má robiť. Následne robot bude pomáhať zamestnancovi alebo bude vykonávať zadanú úlohu samostatne. Týmto by sa mala uľahčiť a najmä zefektívniť práca zamestnancov.

Pretože užívateľmi tohoto systému nebudú ľudia s informatickým vzdelaním, ale bežní zamestnanci, je potrebné, aby ovládanie a učenie robota bolo jednoduché a hlavne intuitívne. To je nutné zohľadňovať pri návrhu a tvorbe grafického užívateľského rozhrania, ale aj pri samotnom spôsobe učenia robota vykonávať požadovanú úlohu.

Systém v súčasnosti podporuje iba obmedzený počet inštrukcií, medzi nimi tieto dve: uchopenie daného typu z definovanej oblasti a položenie na zadané miesto.

Cieľom tejto bakalárskej práce je rozšíriť množinu inštrukcií podporovanú robotom a umožniť užívateľovi naučiť robota uložiť objekty rovnakého typu, napríklad do krabice.

Úvodna kapitola rozoberá zjednodušené programovanie robotov, popisuje výhody a nevýhody jednotlivých prístupov a uvádza príklady rozhraní, ktoré sa začínajú používať pre interakciu medzi človekom a robotom. Ďalšia kapitola popisuje existujúci systém ARTable. Štvrtá kapitola obsahuje teoretické informácie o tvorbe grafických užívateľských rozhraní pomocou framework-u Qt. Následuje analýza tých častí systému ARTable, ktoré sa budú rozširovať a samotný návrh rozšírení. Popis samotnej implementácie spolu so spôsobmi a výsledkami testovania je v kapitole 6.

[1] http://www.fit.vutbr.cz/research/groups/robo/.cs

Kapitola 2

Zjednodušené programovanie robotov

Programovanie industriálnych robotov dokáže byť veľmi zdĺhavý a únavný proces, kde často implementovanie aj len jednoduchej funkcie vyžaduje vedomosti z oblasti robotiky na expertnej úrovni. Navyše ľudia musia absolvovať týždne na školeniach, aby boli schopní týchto robotov ovládať, pričom sa firma stáva závislou na týchto pár ľudoch. [20]

Experti zvykli týchto robotov programovať na nízkej úrovni, čo výrazne znížilo ich flexibilitu na zmeny vo výrobe a spôsobilo dlhé a nákladné úpravy pri zavádzaní nových produktov do výroby. Spolu s vysokou cenou automatizácie sa vyplatilo využívanie industriálnych robotov zväčša len vo veľkovýrobe. Ďalším dôvodom, prečo boli využívané hlavne vo veľkovýrobe bolo, že v malých a stredne veľkých podnikoch sa často vyrábajú aj produkty na mieru pre konkrétnych zákazníkov. [16, 20]

V poslednom čase sa však objavili viaceré projekty podporované Európskou Úniou, ktoré sa snažia podporiť vývoj ľahko prenastaviteľných robotov, ktoré by boli dostatočne flexibilné aj pre tieto malé a stredne veľké podniky. To znamená vytvorenie jednoduchého a efektívneho spôsobu interakcie človeka s robotom, ktoré umožní jeho naprogramovanie a obmedzenie tradičného zložitého programovania vyžadujúceho hlboké vedomosti. [16]

Ideálnym riešením by bol všestranný robot, ktorého by bolo možné použiť na širokú škálu úloh v najrôznejších situáciach, avšak naprogramovanie takéhoto robota je nemožné. Preto sa objavujú prístupy umožňujúce programovať robotov až po ich nasadení do požadovanáho prostredia priamo koncovými užívateľmi. Existujúce riešenia možno rozdeliť do troch skupín. Tie, ktoré umožňujú [1, 16]:

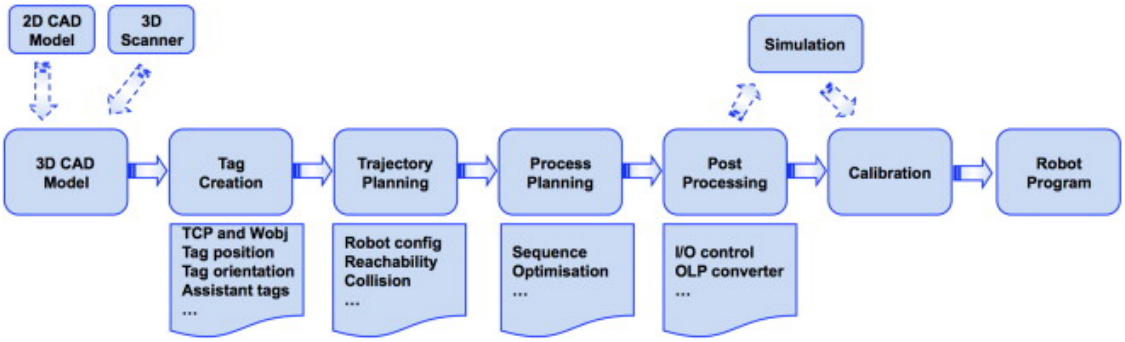
- Offline[1] programovanie robota, kde sa robot naprogramuje len raz
- Nepretržitú spoluprácu človeka a robota (online[2] programovanie)
- Kombináciu oboch predchádzajúcich možností

2.1 Offline programovanie robota

Je metóda programovania robota, pri ktorej proces tvorby programu nevyžaduje skutočného robota, ale je založený len na 3D modely kompletného pracoviska s robotom v simulátore.

[1] Programovanie v simulátore.
[2] Programovanie na skutočnom robotovi.

Offline programovanie neodstraňuje zdĺhavý proces programovania, ale len ho presúva z obsluhy robota na pracovisku na softwarového inžiniera v kancelárii. Tvorba programu touto metódou je veľmi komplexná a kľúčové kroky možno vidieť na obrázku 2.1. Po dokončení vývoja expertom, je tento program nahraný do robota. [19]



Obr. 2.1: Kľúčové kroky offline programovania [19]

Výhody [18, 19]:

- Znižuje dobu odstávky robota potrebnú na jeho programovanie, pretože programy sú vytvorené v simulátore, takže robota je nutné odstaviť len počas nahrávania a testovania nového programu, prípadne kalibrovania.
- V simulátore je možné otestovať mnoho rôznych prístupov k jednému problému, čo by reálny robot neumožňoval, pretože by jeho pridlhé odstavenie veľmi zbrzdilo produkciu.
- Vhodný na tvorbu komplexných systémov, efektívny a cenovo výhodný v podnikoch s vyššou produkciou.

Nevýhody [18, 8]:

- Virtuálne modely nie sú schopné napodobiť skutočný svet s úplnou presnosťou.
- Môže nastať situácia, že programátor musí riešiť problémy vytvorené samotným simulátorom namiesto toho, aby sa venoval vytváraniu programu.
- Programy, vygenerované týmto spôsobom, takmer vždy vyžadujú kalibráciu, než je možné ich použiť v reálnej produkcii. Počas kalibrácie sú opravené nepresnosti, ktoré vznikli kvôli odchýlkam medzi vymodelovaným prostredím a skutočným svetom. Z týchto dôvodov nie je možné už počas programovania zohladniť skutočné prostredie, v ktorom bude robot pracovať.

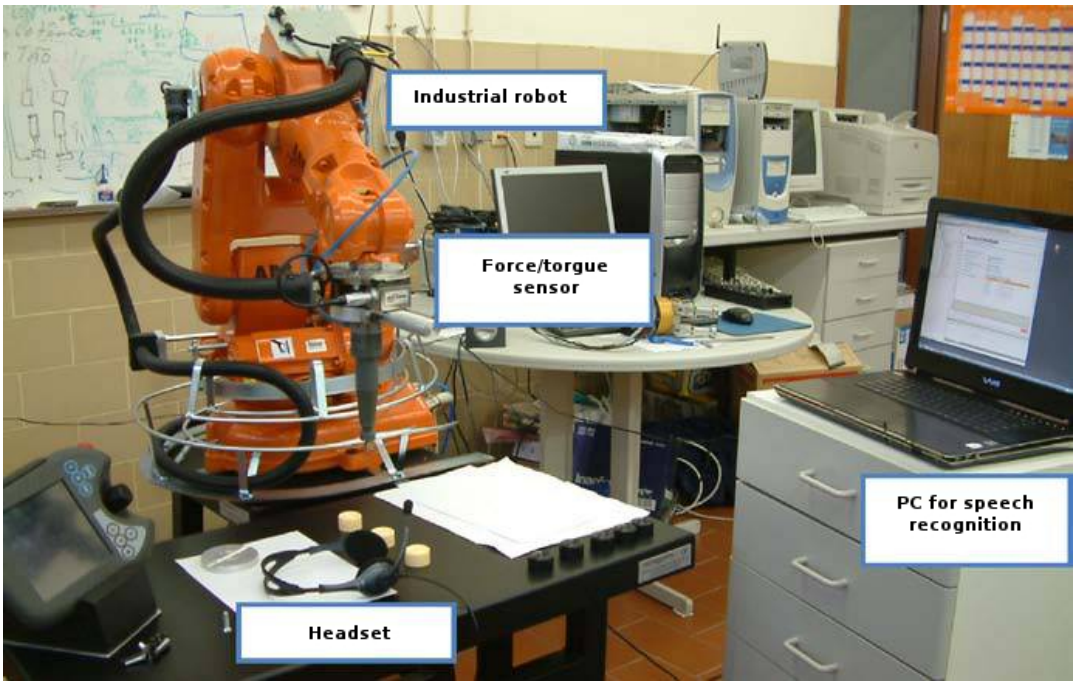
2.2 Online programovanie robota

Hlavnou myšlienkou daného prístupu je, že človek a robot spolupracujú v snahe splniť bežnú úlohu. Predpokladá sa, že zamestnanci, ktorí robota na pracovisku obsluhujú, ho zároveň „programujú“ podľa aktuálnych potrieb. Systém robota preto musí mať istú úroveň samostatnosti, aby dokázal spracovať informácie o okolí a o užívateľovi. Taktiež musí byť

programovateľný na mieste a počas danej úlohy pomocou prirodzeného a intuitívneho užívateľského rozhrania, cez ktoré aj neskúsený užívateľ zvládne robota naprogramovať. Tento vývoj však potrebuje vyššiu úroveň abstrakcie. [21]

Osvojenie si vysoko úrovňových techník programovania ľahko prekoná nedostatky klasických prístupov, pretože výrazne uľahčia užívateľom programovanie robotov. Cieľom je, aby človek dokázal robota naučiť riešenie úlohy jemu prirodzeným spôsobom. [21]

Dobrým príkladom je technika programovania ukážkou(po anglicky Programming-by-demonstration), ktorá umožňuje robota naučit, ako vykonať úlohu tak, že mu zamestnanec alebo iný robot vykonávanie tejto úlohy ukáže [13, 21]. Existuje viacero spôsobov, ako toho docieliť, napríklad snímaním gést, reči a atď [21]. Jeden takýto systém možno vidieť na obrázku 2.2.



Obr. 2.2: Príklad systému na programovanie ukážkou [21]

Výhody [18]:

- Intuitívnejší než simulačné programy, pretože riešenie úlohy je naprogramované skoro rovnako, ako by ju zamestnanec sám vykonal.
- Väčšinou od obsluhy robota nevyžaduje žiadne vedomosti o konceptoch programovania.
- Vhodný u úloh vyžadujúcich vysokú presnosť, kde by inak bolo potrebných veľa riadkov kódu na dosiahnutie rovnakého výsledku.

Nevýhody [18]:

- Používa skutočného robota, čiže neznižuje dobu odstávky robota až tak, ako offline metóda.
- Posunutie robota na presné súradnice nie je tak priamočiare, ako u iných metód.
- Nevhodný u „algoritmických“ úloh.

2.3 Kombinácia online a offline programovania

V súčasnej dobe sa vývoj čoraz viac sústreďuje na kombináciu predchádzajúcich prístupov, kde cieľom je vybrať ich hlavné pozitíva a vytvoriť z nich nový spôsob programovania robotov na pracovisku, čím hranica medzi online a offline programovaním mizne.

Týmto vzniká modulárne online programovacie prostredie, tvorené hardwarom i softwarom, ktoré ponúka intuitívne učenie robota, ale tiež zachováva podporu pomocných algoritmov. Toto prostredie je väčšinou tvorené troma hlavnými komponentami [14]:

- Rozhranie medzi užívateľom a robotom – napríklad pomocou gést a rozšírenej reality.
- Geometrický model – obsahuje dáta o všetkých objektoch (napríklad prekážkach) na pracovisku robota, čo umožňuje rýchlu detekciu prípadných kolízií.
- Pomocné algoritmy – podporujú užívateľa pri práci, napríklad zabraňujú kolízii robota s okolím alebo samím sebou.

2.4 Pokrok vo vývoji rozhraní

Až u 90% industriálnych robotov zvyklo rozhranie na ich programovanie znamenať ručný ovládací panel (ang. teach pendant) a interakcia operátora s robotom bola vo forme tlačenia tlačítok na tomto ovládacom panely. Ďalšie spôsoby programovania robota nepotrebovali dokonca žiadne rozhranie, pretože v jednom prípade interakcia s robotom znamenala jeho fyzické presúvanie naprieč úlohou operátorom. V druhom prípade išlo o offline programovanie, čiže rozhranie nebolo potrebné, lebo vytvorený program v simulátore sa len nahral do robota. [2]

S vývojom nových prístupov programovania robotov nastáva pokrok aj vo vývoji rozhraní, ktoré sa používajú pri interakcii človeka s robotom. Príkladom môže byť:

- Užívateľské rozhranie s rozšírenou realitou (ang. augmented reality) - s príchodom rozšírenej reality sme schopní zobraziť plány úloh a pohyby robota človeku priamo na pracovisku, čím sa vyrazným spôsobom zlepšuje užívateľovo priestorové vnímanie a jeho porozumenie schopnostiam daného robota. Avšak hlavným pozitívom je, že užívateľ má možnosť zadávať inštrukcie robotovi prirodzenejším a intuitívnejším spôsobom. Takéto rozhranie môže byť napríklad premietané projektorom na stôl [11] alebo integrované do prenosného zariadenia [20].
- Rozhranie využívajúce na interakciu s užívateľom virtuálnu realitu - kde virtuálna realita sa využíva napríklad na trénovanie pohybu [12].
- Rozhrania využívajúce na interakciu prirodzený jazyk v písanej alebo v hovorenej forme, gestá, ukazovacie zariadenia (ang. pointing devices) a iné [20].

Kapitola 3

Systém ARTable

V tejto kapitole je bližšie popísaný už existujúci systém ARTable, pretože táto bakalárska práca ho rozširuje a teda je potrebné, aby si čitateľ dokázal vytvoriť lepšiu predstavu o celom systéme.

ARTable [15] je prototyp pracoviska s rozšírenou realitou pre spoluprácu zamestnanca s robotom. Zameriava sa na malé a stredne veľké podniky, kde by následne mali bežní zamestnanci s jeho pomocou dokázať naprogramovať robota a efektívne a bezpečne s ním spolupracovať na vykonávaní úloh. S robotom by malo byť možné neustále spolupracovať alebo by ho zamestnanec len naprogramoval a robot by následne pracoval samostatne.

Jeho cieľom je dosiahnúť vyššiu presnosť pre určité úlohy, oslobodiť skúsených zamestnancov od vykonávania monotónnych úloh a zvýšiť produktivitu. Preto systém ARTable zavádza nový prístup na uľahčenie programovania robota.

Tento prístup používa rozšírenú realitu na zobrazenie grafického užívateľského rozhrania, ktoré obsahuje aktuálny program, priebeh učenia robota, priebeh vykonávania programu, detegované objekty, inštrukcie pre užívateľa a podobne.

Aktuálne systém podporuje základné funkcie, ako vrátenie robotích ramien do pôvodnej polohy, uchopenie konkrétneho objektu alebo objektov daného typu zo zadanej oblasti a následné umiestnenie na zadané miesto. Samotná funkcionalita systému, z ktorej si užívateľ môže vyskladať požadovaný program, je programovaná expertmi. Užívateľ si potom len vyskladá program a ponastavuje požadované parametre, ako napríklad objekt na uchopenie a miesto na uloženie.

Rozhranie systému umožňuje užívateľovi vybrať program, nastaviť alebo upraviť jeho parametre a následne spolupracovať s robotom na vykonávaní úlohy.

3.1 Pracovisko s rozšírenou realitou

Projekt na ukážku blízkej budúcnosti používa bezpečného robota PR2, ktorý je umiestnený za stolom (obrázok 3.1), kde sa realizuje interakcia medzi zamestnancom a robotom.

Interakcia pozostáva z naprogramovania robota a následnej spolupráce pri vykonávaní naprogramovanej úlohy. Realizuje sa prostredníctvom dotykového stola, ktorý bude podrobnejšie popísaný neskôr.

Užívateľ je snímaný pomocou sensoru Kinect, ktorý je umiestnený na hlave robota. Na získanie informácií o užívateľovej polohe a smerovaní sa používa skeletal tracking[1].

[1] Xbox Kinect funkcia, ktorá umožňuje rozpoznávanie ľudí [17]

Obr. 3.1: Pracovisko so stolom a robotom[2]

Nad stolom je umiestnená kamera, ktorá detekuje objekty pomocou AR kódov, ktoré sú prilepené na ich vrchu.

3.2 ARTable hardware komponenty

V tejto časti je podrobnejsšie popísaný používaný robot PR2 a stôl s rozšírenou realitou.

3.2.1 Robot PR2

PR2 (Personal Robot 2) je platforma pre výskum a vývoj robotiky, ktorá umožňuje začať hneď inovovať. To znamená, že sa vývojár môže sústrediť na konkrétnu oblasť, zatiaľ čo využíva prácu špecialistov z iných oblastí v podobe samotnej PR2 platformy a viac ako tisíc knižníc. Takže už nie je nutné skladať hardware a písať software od úplných základov, ako tomu bolo v minulosti, čo predstavuje obrovský prínos pri vývoji nových vecí. Naviac PR2 je otvorenou platformou, čiže je možné upraviť si systém podľa potrieb. [10]

Robot PR2 sa trochu odlišuje od ostatných robotov, pretože pri jeho tvorbe bolo hlavným zámerom, aby bol použiteľný v domácnosti a nie v priemysle. Aj napriek tomu je však veľmi dobre uspôsobený na prácu v akomkoľvek prostredí a vďaka jeho vlastnostiam dokáže vykonať mnoho rôznych úloh. [3]

Robot má dve zhodné ramená, ktoré sa dokážu veľmi dobre prispôsobiť pre potreby požadovaného prostredia, pretože majú 7 stupňov voľnosti. Zvládne vykonať ďaleko viac úloh, než keby mal len jedno rameno, napríklad lebo zvládne operovať vo väčšej oblasti. „Ruky“ PR2 (ang. grippers) dokážu rovnobežné, ale aj valcovité úchopy a majú troj-osový akcelerometer. Jeho vrchnú časť tela môže pozdvihnúť, pokiaľ to niektorá z úloh vyžaduje. [3, 10]

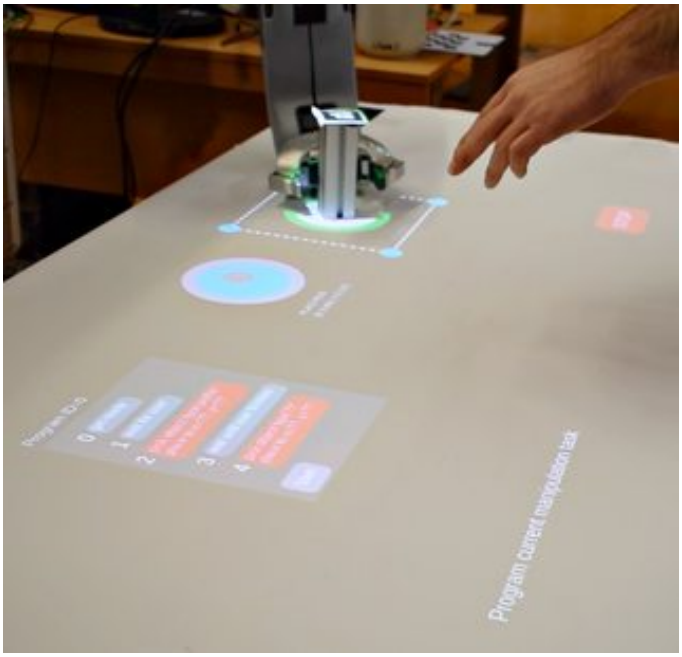
[2]Obrázok je prevzatý Git repozitáru https://github.com/robofit/artable

Robot má tiež kamery a lasery, pomocou ktorých vie rozoznávať objekty. V skutočnosti má umiestnený laser aj v „krku“, pričom tento laser pracuje neustále a informuje robota, ktorý tak neustále vyhodnocuje svoje okolie. [3]

U niektorých aplikácií, ako je napríklad otvorenie chladničky, musí robot vedieť, koľko sily musí vynaložiť, aby jej bolo dostatok na otvorenie, ale nie príliš, čo by mohlo spôsobiť odtrhnutie kľučky. Špičky jeho „prstov“ obsahujú tlakový sensor a vedia tak detegovať potrebnú silu, ktorú je treba vynaložiť pri manipulácií s daným objektom. [3, 10]

3.2.2 Dotykový stôl

Systém využíva dotykový stôl (obrázok 3.2), ktorý slúži ako vstupné zariadenie. Ďalej systém používa rozšírenú realitu, vo forme premietaného užívateľského rozhrania na stôl, ako spätnú väzbu. Pred používaním systému ARTable je nutné vykonať kalibráciu.



Obr. 3.2: Dotykový stôl s rozšírenou realitou[3]

3.3 ARTable software komponenty

Software systému ARTable je rozdelený do niekoľkých komponent.

Hlavnou časťou je art_brain, ktorý je „hlavou“ celého systému a riadi jeho priebeh. Taktiež umožňuje robota naučiť rôzne funkcie, ako napríklad uchopenie a uloženie obejktu.

Stále úložisko zabezpečuje art_db založená na MongoDB. Momentálne možno túto databázu naplniť pomocou skriptu, čím si užívateľ môže poskladať program z tzv. programových blokov. Zároveň si môže pridať rôzne typy objektov spolu s ich rozmermi.

O úvodné nakalibrovanie robota PR2 a zobrazenia užívateľského rozhrania na stôl sa stará art_calibration.

[3]Obrázok je prevzatý z Git repozitáru https://github.com/robofit/artable

Užívateľské rozhranie s rozšírenou realitou zaobstaráva art_projected_gui, ktoré generuje 2D scénu. Následne projektory tieto 2D snímky zobrazia na stôl. Toto je rozdelené do viacerých procesov, aby generovanie scény mohlo byť na inom stroji, než kam je pripojený projektor. Rozhranie ponúka aj možnosť zobrazenia debugovacieho okna, ktoré je možne kontrolovať myškou.

Ovládanie systému pomocou užívateľových gést zaisťuje art_table_pointing, ktoré na to využíva Xbox Kinect.

Detegovanie objektov na stole sa vykonáva pomocou AR kódov. O to sa stará art_arcode_detector spolu s art_simple_tracker.

Ovládanie ramien robota má na starosti art_pr2_grasping, ktoré realizuje uchopovanie a presúvanie obejktov na stole. Na presúvanie obejktov sa používa software MoveIt!.

Pre získanie presnejšej predstavy je možné si pozrieť blokové schéma v prílohe (obrázok A.1).

Kapitola 4

Tvorba grafického užívateľského rozhrania

Grafické užívateľské rozhranie umožňuje človeku komunikovať s počítačom pomocou grafických kontrolných prvkov, namiesto vkladania príkazov do príkazovej riadky, ako tomu bolo v minulosti. Tento grafický kontrolný prvok (ang. widget) môže byť okno, tlačítko, posuvník, zaškrtávacie menu a mnoho iných. Knižnice, ktoré obsahujú sadu týchto grafických kontrolných prvkov, umožňujú vytvárať užívateľské rozhranie tak, aby vývojár mohol znova použiť daný kód na podobné úlohy. Vďaka tomu užívateľ môže komunikovať cez rozhranie, ktoré mu je dobre známe a ktoré je konzistentné.

Jednou takouto knižnicou je práve Qt, ktoré bolo použité v tejto práci.

4.1 Qt framework

Qt je multiplatformový aplikačný framework pre vývoj aplikácií s grafickým užívateľským rozhraním. Je napísaný v jazyku C++, ktorý rozširuje o funkcie, ako napríklad signály a sloty (ang. signals and slots). [5]

4.1.1 Signály a sloty

Často pri programovaní grafického užívateľského rozhrania chceme, aby keď zmeníme jeden widget, aby ďalší widget bol o tejto zmene informovaný. Celkovo chceme, aby objekty akéhokoľvek druhu boli schopné medzi sebou komunikovať. Mnohé knižnice zaobstarávajú túto komunikáciu pomocou metódy spätného volania (ang. callback). Avšak táto metóda môže byť nedostatočne intuitívna a môže mať problémy pri zabezpečovaní správnosti typov argumentov. V Qt sa ako alternatíva ku callback metóde používajú signály a sloty. [7]

Signál je vyslaný v prípade, keď nastane špecifická udalosť. Widget-y Qt framework-u majú preddefinovaných mnoho signálov a kedykoľvek im možno pridať naše vlastné. Objekty vyšlú signál vždy, keď nastane zmena ich stavu, ktorá by mohla zaujímať iné objekty. Pričom toto je ich jediná činnosť, ktorú vykonávajú v rámci komunikácie. Nezaujíma ich a ani nevedia, či niekto prijíma ich vysielané signály. Ide o zapúzdrenie informácií, ktoré zaisťuje použiteľnosť objektu ako software komponenty.

Slot je funkcia, ktorá sa volá ako odpoveď na špecifický signál. Widget-y Qt framework-u majú preddefinovaných mnoho slotov, avšak je zvykom si pridať vlastné sloty na spracovanie signálov, ktoré nás zaujímajú. Sloty možno využiť na prijímanie signálov, ale zároveň sú normálnymi metódami. Tak, ako objekt nevie, či niekto prijíma jeho vysielaný signál, tak slot nevie, či je spojený s nejakým signálom. Vďaka čomu možno pomocou Qt vytvoriť naozaj nezávislé komponenty.

4.1.2 Graphics View Framework

Qt triedy pre tvorbu grafiky sú ideálnym riešením pre grafiky, v ktorých potrebujeme od pár do niekoľkých tisíc položiek (ang. items) a užívateľovi chceme umožniť klikať, presúvať, rotovať a vyberať tieto položky [4].

Architektúra pre tvorbu grafiky pozostáva zo scény, ktorá je reprezentovaná triedou *QGraphicsScene*, a položiek na scéne, ktoré sú reprezentované podtriedami *QGraphicsItem*. Scéna (spolu s položkami) je zobrazená užívateľovi v pohľade (ang. view), ktorý je reprezentovaný triedou *QGraphicsView*. Tá istá scéna môže byť zobrazená vo viacerých pohľadoch, napríklad na zobrazenie rôzných častí rozsiahlej scény. Na rozhodnutie, či je nutné položku vykresliť, táto architektúra využíva hraničiaci pravouholník (ang. bounding rectangle). Vďaka čomu QGraphicsView dokáže veľmi rýchlo zobraziť ľubovoľne veľkú scénu, pokiaľ má byť viditeľných v danom čase len zlomok položiek. Tieto triedy pre tvorbu grafiky sú jednoduché na použitie a ponúkajú veľké množstvo funkcionality. [4]

Framework tiež ponúka architektúru na šírenie udalostí, ktorá zaisťuje precízne interakčné schopnosti položkám na scéne [6].

QGraphicsScene [6] uchováva kolekciu grafických položiek. Scéna nás môže informovať, ktoré položky sú v kolízii, ktoré sú vybrané a ktoré sú na konkrétnom mieste/oblasti. Grafické položky na scéne sú buď potomkom scény alebo inej položky, pričom akékoľvek transformácie u rodiča, sa aplikujú aj na jeho potomkov. QGraphicsScene poskytuje Graphics View scénu a má na zodpovednosti:

- Poskytovanie rýchleho rozhrania pre menežovanie veľkého množstva položiek.
- Propagáciu udalostí každej položke.
- Menežovanie stavu položky, ako napríklad spracovanie výberu.

QGraphicsView [6] je widget, ktorý prezentuje scénu, poskytuje posuvníky (ak je to potrebné) a je schopný aplikovať transformácie, ktoré ovplyvňujú, ako je scéna vyrenderovaná (ang. rendered). Toto je užitočné pre podporu približovania (ang. zooming) a rotácie pri prezeraní scény. Graphics View používa spôsob rozdelenia priestoru pomocou binárneho stromu (ang. Binary Space Partitioning tree), čo umožňuje rýchle objavovanie položiek. Výsledkom je zobrazenie aj rozsiahlej scény, dokonca aj s miliónmi položiek, v reálnom čase.

QGraphicsItem [6] je základná trieda pre grafické položky na scéne. Graphics View poskytuje viacero štandardných položiek pre typické tvary, ako pravouholníky, elipsy, polygóny, avšak najmocnejšie vlastnosti QGraphicsItem sú k dispozícii, keď vytvárame vlastnú položku. Mimo iné, QGraphicsItem podporuje následujúce:

- Stlačenie myši, posúvanie, upustenie a dvojité kliknutie myši. Ďalej vie rozpoznať, ked naň ukazuje myš a tiež pohyb koliečka na myške.

- Stlačenie kláves klávesnice
- Ťahanie a pustenie (ang. Drag and drop)
- Zoskupovanie
- Detekciu kolízií

Kapitola 5

Analýza a návrh riešenia

Pred tvorbou nového alebo rozširovaním existujúceho software je dôležitou fázou analýza. Preto pred návrhom konkrétneho riešenia je potrebné sa zamyslieť, aké požiadavky by daný software mal spĺňať. Kedže úlohou tejto práce je pridať funkcionalitu už vytvorenému systému, je nutné porozumieť funkcii tých častí, ktorých sa rozširovanie týka, pretože dané požiadavky a návrh budú nimi silným spôsobom ovplyvnené.

Táto kapitola sa teda zaoberá analýzou častí ARTable, požiadavkami na pridávanú funkcionalitu a návrhom riešenia.

5.1 Analýza už existujúceho systému

Pozornosť sa kladie na podrobný popis už existujúcich častí systému ARTable, ktoré táto bakalárska práca rozširuje, presnejšie časti art_projected_gui a art_brain.

V prípade zanedbania analýzy už vytvoreného GUI[1] by mohla nastať situácia, že po jeho rozšírení by výsledné GUI nepôsobilo ako jeden celok, čo by mohlo užívateľa zbytočne miasť.

5.1.1 GUI systému ARTable - art_projected_gui

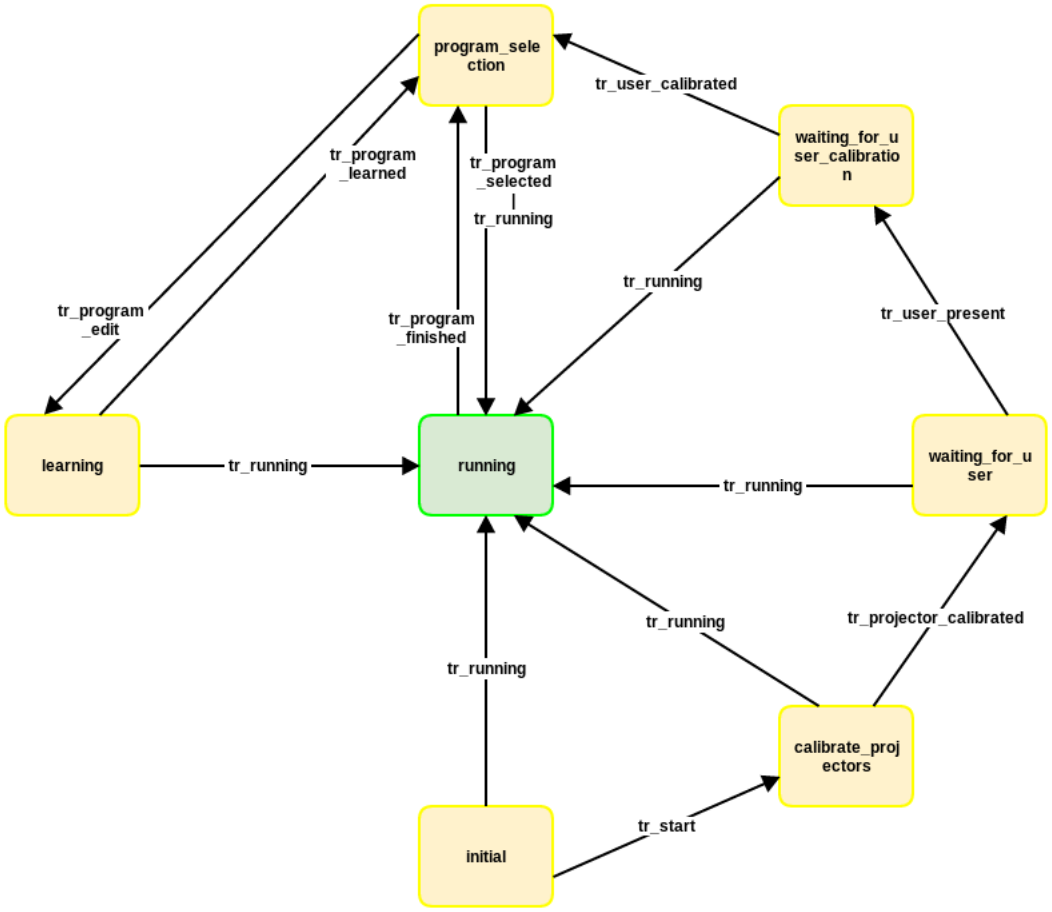
Grafické užívateľské rozhranie je vytvorené pomocou knižnice Qt.

Trieda UICore má na starosti vytvorenie scény(QGraphicsScene), pohľadu(QGraphicsView) a tcp serveru pre pripojenie projektorov(QTcpServer). Zároveň pripája signál na posielanie scény klientovi. Ďalšie funkcie:

- Pridáva objekty (inštancie tried ObjectItem, PlaceItem, PolygonItem) do scény.
- Odstraňuje objekty zo scény.
- Zobrazuje notifikácie užívateľovi.
- Zobrazuje debugovacie okno, pokiaľ je požadované.
- Zvýrazňuje zakliknuté objekty.
- Generuje 2D snímky pre projektory.

[1] Grafické užívateľské rozhranie.

Trieda UICoreRos je postavená na triede UICore, ktorej pridáva veci súvisiace s robotím operačným systémom (ROS) a implementuje aplikačnú logiku. Trieda využíva konečný automat (ang. Finite State Machine FSM), ktorého stavy a prechody sú zobrazené na obrázku 5.1. Ako prvé sa nakalibruje projektor/projektory. Následne sa čaká na prítomnosť užívateľa, aby mohla prebehnúť jeho kalibrácia. Po jeho kalibrácii ďalši prechod už zavisí od užívateľa, či chce rovno pustiť nejakú z už naučených činností alebo chce robota najprv naučiť novú činnosť, prípadne upraviť už existujúcu.



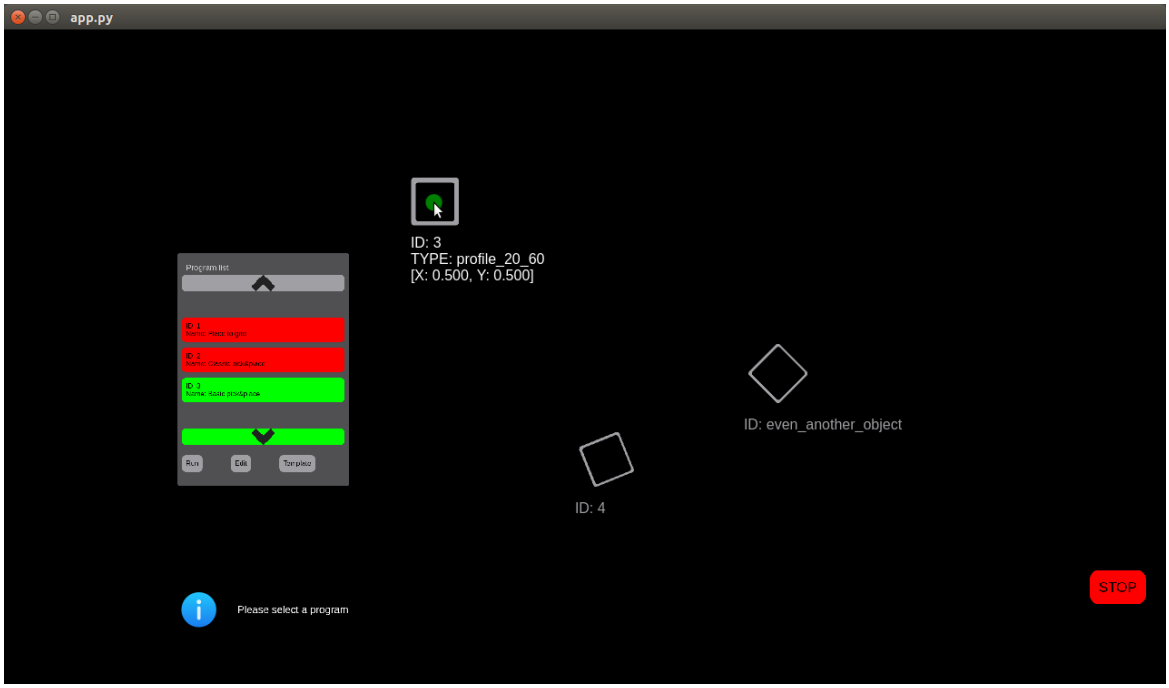
Obr. 5.1: Konečný automat triedy UICoreRos

Vytvorené GUI na obrázku 5.2 je kombináciou týchto dvoch tried, tried jednotlivých vykresľovaných objektov a pomocných tried. Ako už bolo spomenuté vyššie, scéna a vykreslené objekty sú vytvorené triedou UICore. Menu, kurzor a interakciu s užívateľom má na starosti trieda UICoreRos.

Rozhranie obsahuje rôzne elementy na zobrazenie stavu robota a stavu vykonávanej úlohy, ako napríklad aktuálne načítaný program. Program je užívateľovi zobrazený jak počas procesu učenia robota, tak počas vykonávania zadanej úlohy.

Oblasť okolo obvodu detegovaného objektu je ohraničená týmto rozhraním, ktoré tiež zobrazuje jeho názov.

Na interakciu s rozhraním užívateľ teda používa gestá. Na to, aby mal užívateľ predstavu kam presne ukazuje, sa mu zobrazuje kurzor, ktorý má podobu malého zeleného kruhu.



Obr. 5.2: Grafické užívateľské rozhranie systému ARTable

Na označovanie objektov sa využíva kolízia práve tohoto kurzoru s daným objektom a to tak, že najprv je objekt „predoznačený“ (zobrazia sa podrobnejšie údaje o objekte, ako napríklad súradnice jeho polohy) a ak táto kolízia pretrváva istý čas, tak sa objekt nastaví za označený. [15] Tento proces možno vidieť na obrázku 5.3, kde vľavo hore je objekt čisto detegovaný, v strede možno vidieť označený a vpravo je objekt „predoznačený“.

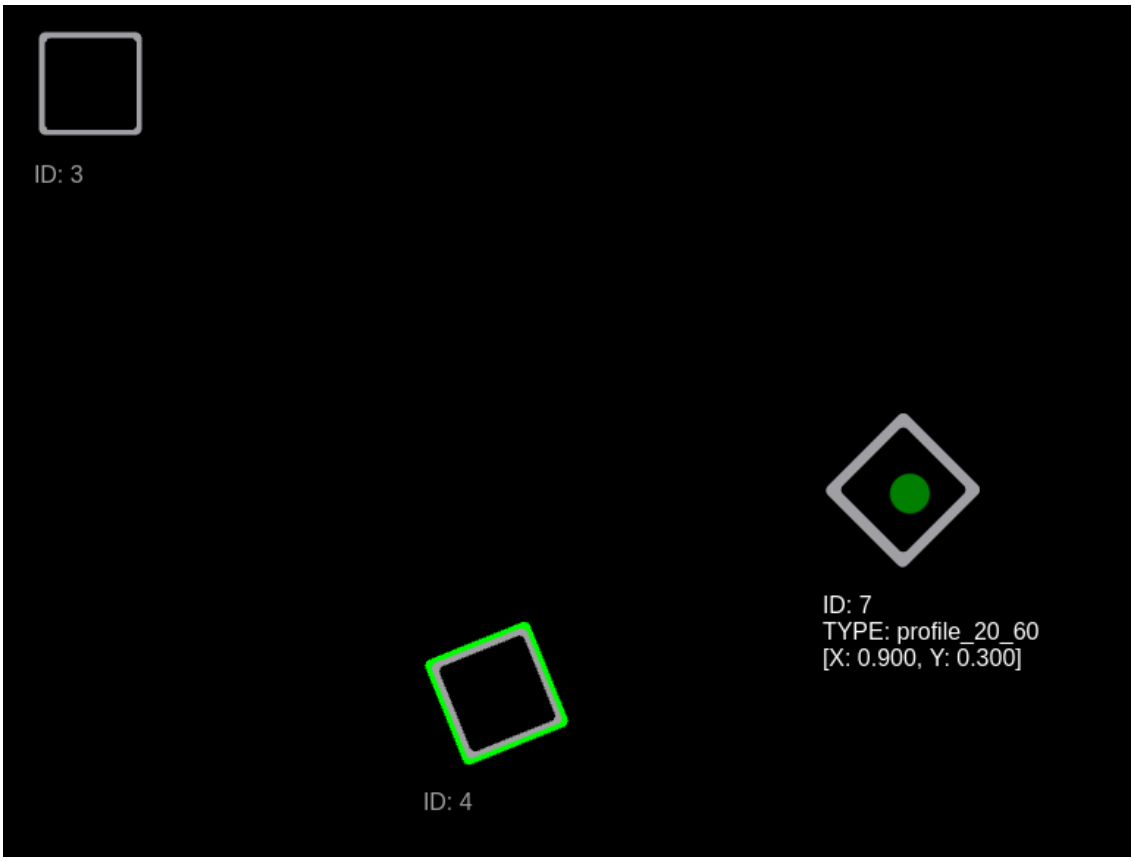
Po úvodných kalibráciách je zobrazené na scénu menu obsahujúce zoznam programov (obrázok 5.4), ktoré sú načítané z databázy. Programy zobrazené zeleno sú pripravené na spustenie, ale užívateľ ich môže v prípade potreby poupraviť. Pri programoch zobrazených červeno je nutné pred spustením robota doučiť niektorú z inštrukcií.

Na doučenie robota je nutné, aby užívateľ rozklikol daný program (obrázok 5.5), kde komponenty zobrazené červeno vyžadujú doplnenie potrebných informácií. Doúčanie robota možno vidieť na obrázku 5.6, kde užívateľ učí robota, ktorý objekt má presunúť.

Užívateľ tiež môže robota naučiť nový program a to vyskladaním z už existujúcich komponent a následným vložením do databázy pred spustením samotnej aplikácie. Funkcionalita týchto komponent bola už predtým naprogramovaná expertami.

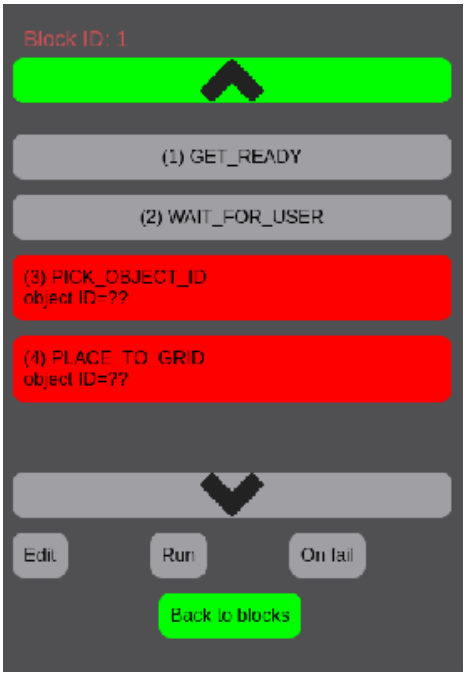
Rozhranie taktiež informuje užívateľa, čo práve prebieha alebo čo od neho potrebuje spraviť. Príklady takýchto notifikácií možno vidieť na obrázkoch 5.7 a 5.8.

Všetky objekty v scéne sú odvodené od triedy Item, ktorá je odvodená od triedy QGraphicsItem.

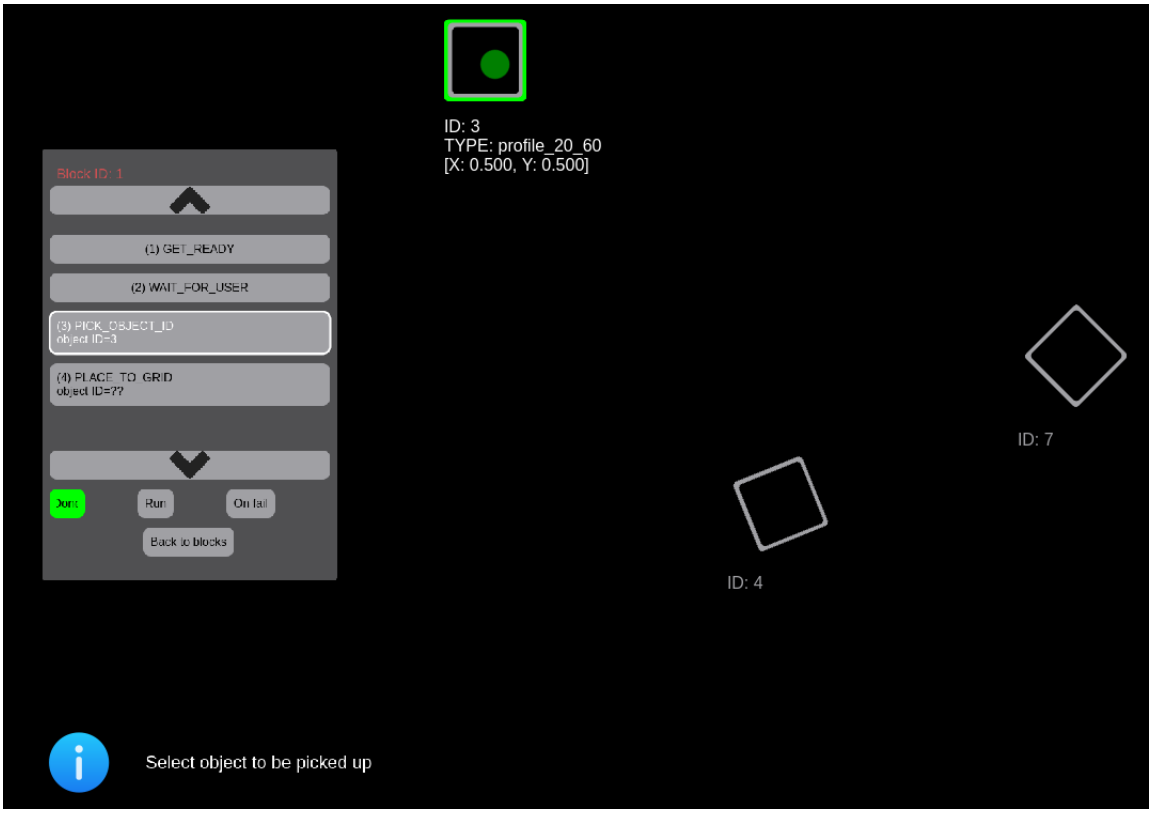


Obr. 5.3: Objekty na scéne

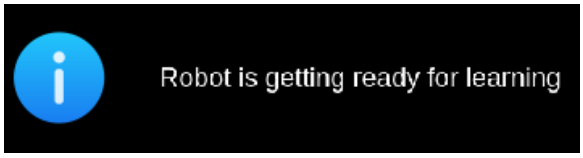
Obr. 5.4: Zoznam programov



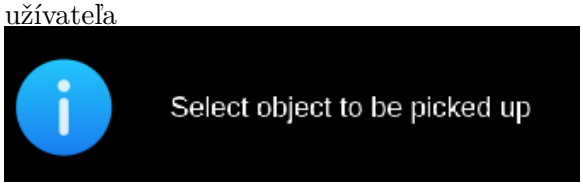
Obr. 5.5: Konkrétny program



Obr. 5.6: Proces učenia



Obr. 5.7: Ukážka jedného z upozornení pre užívateľa



Obr. 5.8: Ukážka jedného z upozornení pre užívateľa

5.1.2 „Mozog“ systému ARTable - art_brain

To, čo užívateľ robota v GUI naučí, je uložené do ROS správy typu Program, ktorej pre túto prácu dôležitou časťou je definícia typu ProgramItem:

```
uint16 id # id danej položky
uint16 on_success # id následujúcej položky
uint16 on_failure # 0 zmanená, že má skočiť na koniec

uint16 type

# Typy inštrukcií
uint16 GET_READY=0
uint16 WAIT_FOR_USER=50
uint16 WAIT_UNTIL_USER_FINISHES=51
uint16 PICK_FROM_POLYGON=100
uint16 PICK_OBJECT_ID=102
uint16 PLACE_TO_POSE=200

# Parametre inštrukcie
string[] object # id alebo typ
geometry_msgs/PoseStamped[] pose
geometry_msgs/PolygonStamped[] polygon
uint16[] ref_id # referencia na inú inštrukciu
```

Naučený program potom vykonáva art_brain. Ako môžeme vidieť v správe ProgramItem, art_brain momentálne podporuje 6 typov inštrukcií. Prvé tri inštrukcie sú jednoduchšie. Ďalšie tri inštrukcie sú už o čosi zložitejšie, kde prvé dve umožňujú uchopenie a posledná položenie objektu/objektov.

Keď art_brain narazí na položku programu typu:

- GET_READY - tak vráti obe ramená do východzej polohy.
- WAIT_FOR_USER - tak čaká, dokým je užívateľ pripravený na prácu.
- WAIT_UNTIL_USER_FINISHES - tak čaká, dokým užívateľ dokončí prácu.
- PICK_OBJECT_ID - tak sa bude uchopovať konkrétny objekt, ktorý bol vybraný užívateľom. Ako prvé sa skontroluje či objekt zadaný v správe ProgramItem je detekovaný na stole. Pokiaľ áno, tak sa rozhodne, ktoré z ramien sa použije k uchopeniu daného objektu. Následne sa nastavia potrebné parametre a cez actionlib[2] sa volá art_pr2_grasping, ktorá zabezpečí samotné uchopenie, čím táto položka programu končí.
- PICK_FROM_POLYGON - tak sa budú uchopovať všetky objekty zo špecifikovanej oblasti. Ako prvé sa skontroluje, či sú uložené body špecifikovanej oblasti v správe ProgramItem. Pokiaľ nie, tak sa bude brať prvý objekt zadaného typu z objektov detegovaných na stole. Pokiaľ áno, tak sa bude brať prvý objekt z detegovaných objektov, ktorý sa nachádza v danej oblasti. Následne sa postupuje rovnako, ako pri

[2]Balík actionlib poskytuje nástroje na tvorbu serverov realizujúcich dlhotrvajúce ciele, ktoré možno predvídať a klientov na komunikáciu s týmito servermi [9].

PICK_OBJECT_ID. Určí sa rameno a zavolá sa art_pr2_grasping. Avšak pri tejto inštrukcii sa počíta s tým, že sa program bude vykonávať v slučke - vezme sa jeden z týchto objektov a presunie sa, následne sa vezme ďalší a presunie sa. Týmto spôsobom sa postupne premiestnia všetky požadované objekty.

- PLACE_TO_POSE - tak sa bude pokladať uchopený objekt na zadané miesto. Ako prvé sa skontroluje či správa ProgramItem obsahuje súradnice miesta, kam má byť objekt položený. Následne sa skontroluje či je zadaná referencia na predchádzajúcu inštrukciu, ktorá mala zabezpečiť uchopenie objektu. Pomocou tejto referencie sa zistí, ktoré rameno sa použilo pri uchopení objektu a teda ktoré rameno momentálne drží objekt. Na záver sa zase nastavia potrebné parametre a zavolá sa art_pr2_grasping, ktorá položí objekt na požadované miesto.

5.2 Návrh riešenia

Systém ARTable je postavený na myšlienke, že človek dokáže robota naučiť vykonávať rôzne druhy úloh. Množinu týchto úloh určujú podporované inštrukcie, ktoré boli dopredu naimplementované vývojármi s informatickým vzdelaním. ARTable v čase písania tejto práce podporoval 6 inštrukcií, z čoho iba 3 umožňovali užívateľovi robota niečo naučiť, takže bola použiteľnosť systému ešte značne obmedzená. Účelom tejto bakalárskej práce je teda pridať novú inštrukciu do tejto množiny a vytvoriť tým ďalšiu činnosť poskytovanú robotom.

Pridaná inštrukcia by mala zabezpečiť ukladanie objektov do gridu[3]. Avšak nestačí iba naimplementovať požadovanú funkcionalitu do *art_brain*, pretože užívateľ musí najprv robota naučiť informácie potrebné na vykonávanie programu. Z tohoto dôvodu je nutné rozšíriť aj existujúce rozhranie *art_projected_gui*, ktoré slúži na interakciu (teda aj učenie) medzi užívateľom a robotom. Týmto vznikajú dva problémy, na ktoré je nutné navrhnúť riešenie:

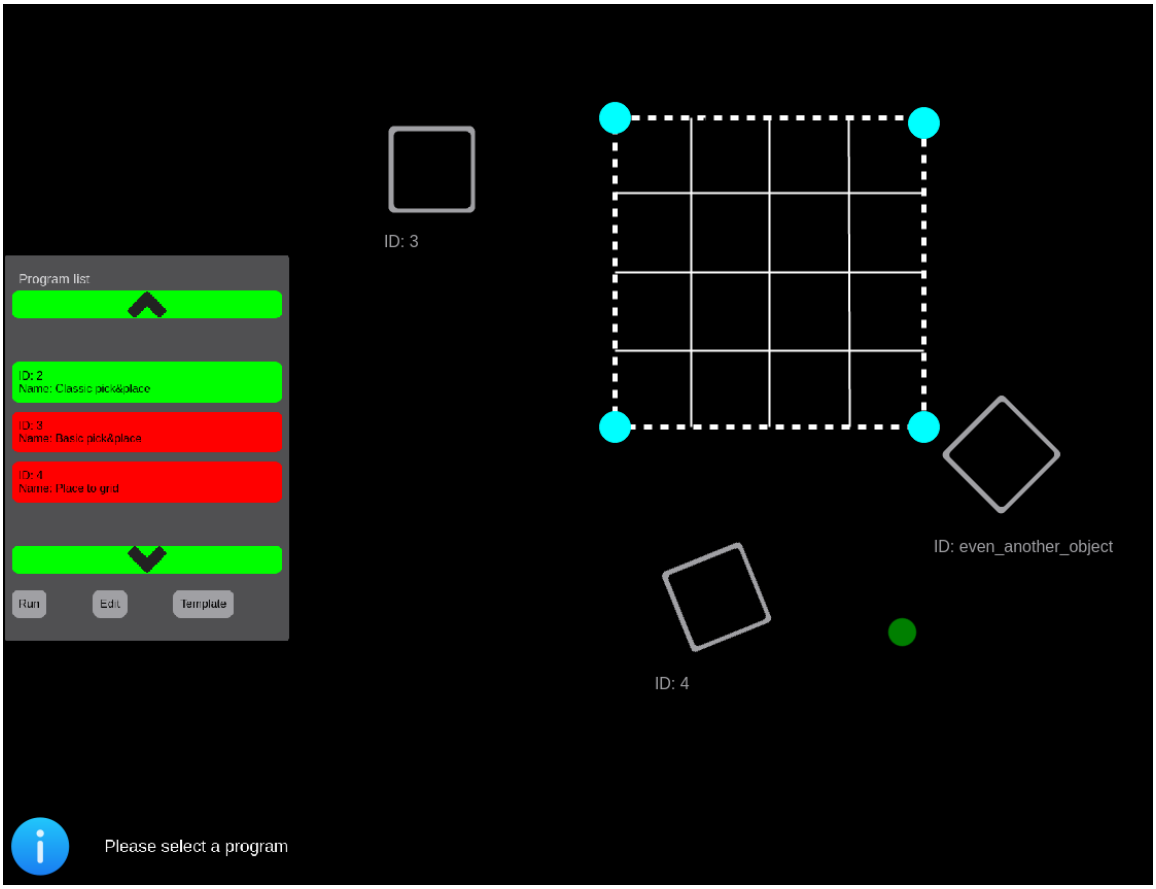
- Ako pridať do rozhrania možnosť naučiť robota uložiť objekty do gridu?
- Ako implementovať samotnú funkcionalitu robota ukladať objekty do gridu?

5.2.1 Návrh rozšírenia GUI

Prvým krokom je pridať do GUI vykresľovanie nového prvku - gridu, aby mohol užívateľ naučiť robota, kam sa budú objekty ukladať. Avšak je treba si uvedomiť, aké požiadavky treba klásť na tento prvok. Okrem klasických požiadaviek, akými sú jednoduchosť a intuitívnosť, je nutné brať v úvahu vzhľad už existujúceho rozhrania, aby systém pôsobil na užívateľa ako jeden celok. Preto návrh vzhľadu gridu je inšpirovaný výzorom už existujúcich prvkov a možno ho vidieť na obrázku 5.9.

Druhým krokom je zachovať spôsob učenia, kde je dôležité, aby interakcia s gridom prebiehala rovnakým spôsobom, ako už u existujúcich prvkov. Napríklad, aby sa rovnakým spôsobom upravovala ich veľkosť. Taktiež správanie prvkov by malo byť rovnaké za stanovených podmienok. Napríklad, aby sa počas vykonávania programu prvky na scéne vykreslili fixné a počas učenia upraviteľné.

[3]Grid je štvoruholníková oblasť predstavujúca napríklad obvod krabice.



Obr. 5.9: Návrh, ako by mohol vyzerať grid po rozšíreni GUI

Na záver, by mal grid vhodným spôsobom vykresliť koľko objektov by sa do neho pri danej veľkosti zmestilo a aké by bolo ich rozmiestnenie, aby mal užívateľ predstavu, koľko objektov sa zmestí do ako veľkého gridu (napríklad krabice).

5.2.2 Návrh rozšírenia funkcionality robota

Systém podporuje inštrukciu na položenie objektu na konkrétne miesto, ktorá sa môže použiť ako odrazový mostík pri návrhu pokladania objektov do gridu. Avšak táto inštrukcia využíva úchop objektov zo strany. Keby sa tento úchop použil pri ukladaní objektov do gridu, tak by spôsobil viacero problémov.

V prípade, že by sa táto inštrukcia použila v praxi, kde by na miesto gridu bola umiestnená krabica, tak najzávažnejším problémom by bola kolízia ramena robota so stenami tejto krabice. Avšak aj keby sa použil len samotný grid, na uloženie objektov do tejto oblasti na stole a neboli by žiadne hrany na jeho obvode, tak by museli byť medzery medzi objektami veľmi veľké, aby pri ich ukladaní ramenu robota nezavadzali už predtým uložené objekty. Inštrukcia by sa stala v podstate nepoužiteľnou v praxi.

Preto pri návrhu riešenia bolo treba nájsť najlepší možný spôsob, ako tieto problémy odstrániť. Pred návrhom konkrétnych prístupov, je nevyhnutné si uvedomiť, akým spôsobom sa určuje pozícia pre úchop objektu. Pôvodné riešenie funguje tak, že sa vygenerujú

všetky možne pozície úchopu objektu (zo všetkých strán a z vrchu) a následne sa vyberie jeden úchop, ktorého realizácia by mala byť najideálnejšia.

2 možné prístupy:

- Úchop objektu z vrchu - toto riešenie by znamenalo, že by sa generovanie úchopov muselo obmedziť len na úchopy z vrchu.
- Úchop zo strany a rotácia objektu až vo vzduchu - takže vo výsledku by robot držal objekt z vrchu a pokladal by sa do gridu na ležato. Tento prístup by však znamenal obmedzenie generovania úchopov, len na úchopy z jednej strany. Pretože po uchopení objektu, je nutné objekt pootočiť o 90 stupňov, kde je potrebné vedieť, z ktorej strany sa objekt uchopil, aby bolo možné určiť os, okolo ktorej má robot objekt pootočiť.

Oba prístupy by zaistili, že by objekty mohli byť do gridu naukladané natesno, lebo by stačila len malá medzera, ktorú zaberá ruka robota pri držaní objektu.

Pre riešenie tejto práce bol pôvodne vybraný prvý prístup, pretože dávalo väčší zmysel, že užívateľ bude chcieť vložiť objekty do gridu na stojato, keď ich tak umiestnil na stôl. Avšak pri testovaní na robotovi sa ukázalo, že oblasť, v ktorej by robot dokázal manipulovať s objektami týmto spôsobom, je veľmi malá. Ďalším problémom bola spoľahlivosť, pretože boli dni, kedy robot nemal s úchopom z vrchu žiaden problém, ale boli aj dni, kedy zrazu nebol schopný vygenerovať jediný zrealizovateľný úchop z vrchu. Z týchto dôvodov, bol nakoniec implementovaný druhý spôsob.

Po vyriešení problému s úchopmi, už ostávalo iba navrhnúť postup ukladania objektov do krabice. Ale keďže ukladanie objektov bude v podstate využívať úchop z vrchu, tak nebol potrebný žiaden špeciálny postup a rozhodol som sa pre ukladanie po riadkoch.

Kapitola 6

Implementácia

V tejto kapitole sa nachádza stručný popis použitých nástrojov, ktoré boli kľúčovými v tejto práci. Ďalej popisuje implementáciu potrebného rozšírenia grafického užívateľského rozhrania premietaného na stôl. Následuje popis potrebných úprav v samotnom presúvaní objektov, aby bolo možné dosiahnúť požadovaný cieľ. Záver kapitoly obsahuje použité spôsoby testovania a ich výsledky.

6.1 Implementačné nástroje

6.1.1 ROS

ROS (Robot Operating System) [9] je flexibilný framework na tvorbu software pre robotov, ktorého cieľom je zjednodušiť vývojárom prácu, pretože tvorba skutočne robustného, viacúčelového robotieho software je veľmi náročná.

ROS poskytuje štandardné služby operačného systému ako abstrakcia hardware, kontrola nízkoúrovňových zariadení, implementácia bežne používanej funkcionality, predávanie správ medzi procesmi a správa balíkov. Taktiež poskytuje nástroje a knižnice pre získavanie, prekladanie, písanie a beh kódu naprieč viacerými počítačmi.

Jeho primárnym cieľom je podporiť znovu použitie kódu. Tento distribuovaný framework procesov (známy tiež ako Nodes) umožňuje, aby spustiteľné súbory boli navrhnuté individuálne a voľne spojované v čase behu. Tieto procesy môžu byť spojené do balíčkov (ang. Packages) a (ang. Stacks), ktoré následne môžu byť jednoducho zdieľané a distribuované.

ROS je open-source[1] software a je podporovaný a vylepšovaný širokou komunitou. Vďaka čomu sú k dispozícii tisíce ROS balíčkov od vývojárov rôzne zo sveta. Nová verzia vychádza zvyčajne raz za rok. V tejto práci sa využíva verzia Indigo.

6.1.2 MoveIt!

MoveIt! je framework na plánovanie pohybu, ktorý je postavený na robotiom operačnom systéme. Framework poskytuje funkcionalitu pre kinematiku, plánovanie pohybu/trajektórie, kontrolu kolízií, 3D percepciu, interakciu s robotom a mnoho iných. U mnohých funkcií pre manipuláciu v ROS, je hlavným zdrojom práve MoveIt!. Využíva niektoré zo základných nástrojov ROS, ako napríklad Rviz, čo je nástroj na 3D vizualizáciu. [22]

[1] S otvoreným zdrojovým kódom.

6.2 Triedy pre tvorbu gridu - SquareItem a SquarePointItem

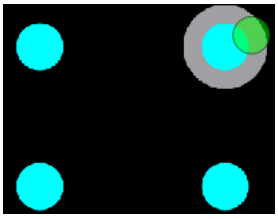
Triedy umožňujú užívateľovi si špecifikovať konkrétnu oblasť tvaru štvoruholník, do ktorej sa neskôr uložia vybrané objekty. Tieto triedy sú postavené na už existujúcej triede Item a to z dvoch dôvodov (i) zachovanie pôsobenia rozhrania ako jedného celku, a (ii) zbytočnosť vytvárať znova niečo, čo už bolo raz vytvorené.

Triedy implementujú následovnú funkcionalitu:

- Zadanie štvoruholníkovej oblasti.
- Vykreslenie maximálneho počtu objektov, ktoré sa do gridu zmestia.
- Rovnomerné rozloženie objektov v gride.
- Zmenu medzery medzi objektami.
- Rotáciu obejktov v gride.
- Presunutie celého gridu spolu s objektami.
- Kontrola kolízií pri učení robota.

6.2.1 Zadanie štvoruholníkovej oblasti

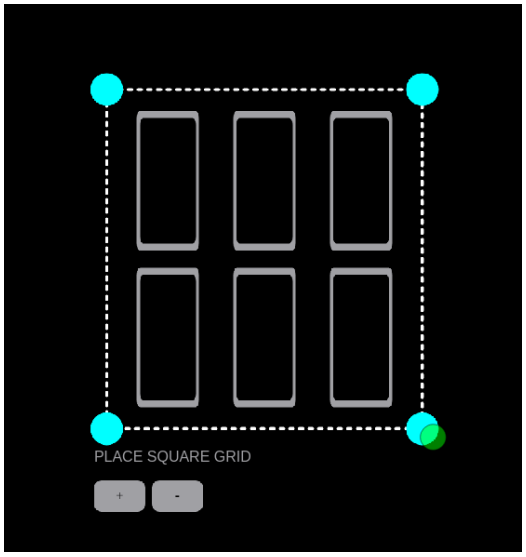
Ako prvé sa vytvoria 4 body, ktoré budú reprezentovať rohy gridu. Samotné body sú inštancie triedy SquarePointItem a možno ich vidieť na obrázku 6.1. Na tomto obrázku tiež vidno jednu z funkcionalít poskytnutých triedou Item, a to detegovanie, že má byť bod zvýraznený, lebo naň užívateľ ukazuje.



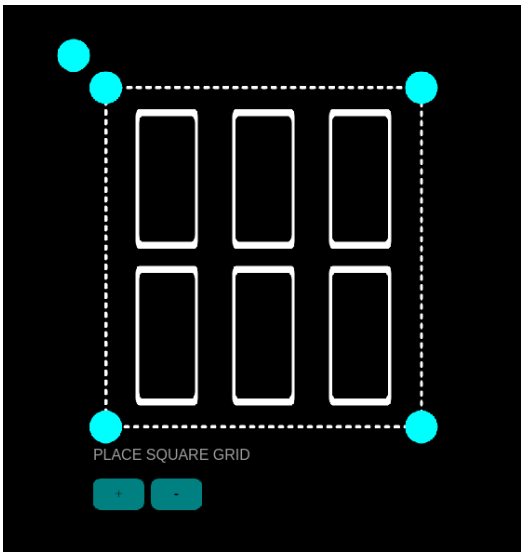
Obr. 6.1: Body predstavujúce rohy gridu

Informácie o pozíciách jednotlivých bodov sa ukladajú do ROS zprávy typu ProgramItem, presnejšie do parametra *geometry_msgs/PolygonStamped[]*, ktorého definícia je:

```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
geometry_msgs/Polygon polygon
  geometry_msgs/Point32[] points
    float32 x
    float32 y
    float32 z
```



Obr. 6.2: Grid, ktorý je fixný



Obr. 6.3: Grid, ktorý je možné upraviť

Pokiaľ užívateľ vytvára grid po prvý krát, tak obsah tejto zprávy je pochopiteľne prázdny, preto sa poloha týchto bodov nastaví na preddefinovanú hodnotu a to približne do stredu stola.

V prípade, že grid bol už niekedy vytvorený a užívateľ ho chce len upraviť, tak sa načítajú polohy bodov z danej zprávy.

Jakmile sú vytvorené jednotlivé body triedou SquareItem, je možné vykresliť štvoruholník, čo sa realizuje pomocou Qt funkcie *drawPolygon()*. Najprv sa nastaví hrúbka, farba a typ čiar, ktoré budú predstavovať obvod gridu a následne sa tejto funkcii ako argument predajú body. Pri vykresľovaní sa tiež zohľadňuje aktuálny stav programu, podľa ktorého sa grid vykreslí buď fixný (obrázok 6.2) alebo upraviteľný (obrázok 6.3). Vykreslený grid možno vidieť na obrázku 6.3, kde už mu bol pridaný aj popisok pomocou triedy DescItem.

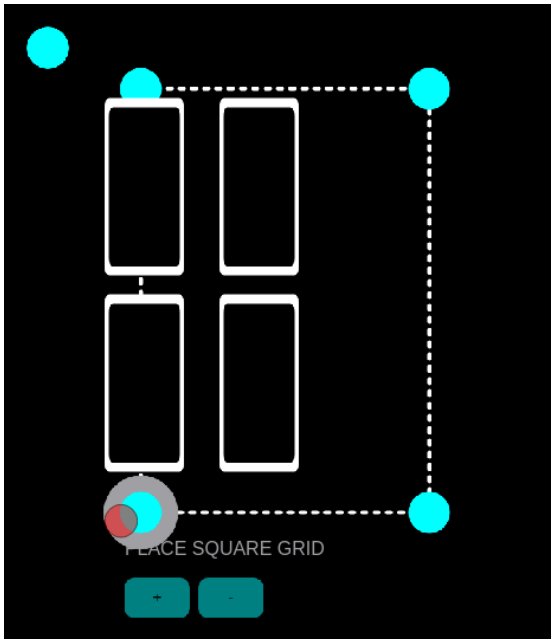
6.2.2 Vykreslenie maximálneho počtu objektov

Veľkosť gridu je možné meniť uchopením jedného z bodov a jeho následným posúvaním. Vtedy daný bod volá metódu *point_changed()* rodičovskej triedy SquareItem. Táto metóda má na starosti viacero vecí.

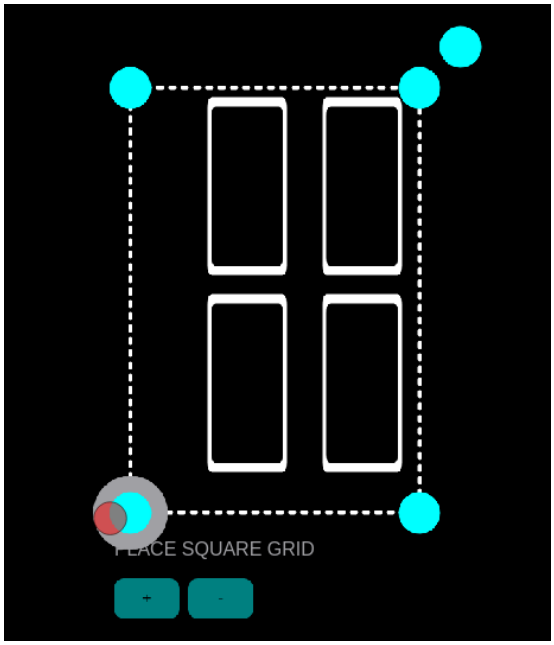
Jednou z nich je, že po zmene veľkosti prenastavuje polohu jednotlivých bodov gridu v odpovedajúcej zpráve. O tom, že upravovanie veľkosti bolo ukončené, ho zase informuje daný bod.

Ďalšou podstatnou funkciou je vykresľovanie objektov, ktoré sa realizuje pomocou do seba vnorenej dvojice *for* cyklov. Na vykreslenie objektov sa využíva trieda PlaceItem, ktorá bola rozšírená o pár parametrov, aby sa dosiahlo požadovanej funkcionality. Tu sa tiež zohladňuje, pomocou ktorého bodu sa mení veľkosť gridu. Objekty sú vždy vykresľované od bodu, ktorý je oproti tomu, s ktorým sa manipuluje. Dôvod je možné vidieť na obrázkoch 6.4 a 6.5.

Na obrázku 6.4, kde sú objekty vždy vykresľované od ľavého horného rohu vzniká problém, že keď sa zmenšuje grid (iným než pravým dolným rohom) a už sa tam niektoré objekty nezmestia, tak sa vymažú objekty z opačnej strany, než je žiadané. Hoci je tento problém skôr estetický, vďaka rovnomernému rozloženiu objektov v krabici, ktoré by prebehlo hneď,



Obr. 6.4: Proces zmenšovania gridu s estetickým problémom



Obr. 6.5: Proces zmenšovania gridu s odstráneným estetickým problémom

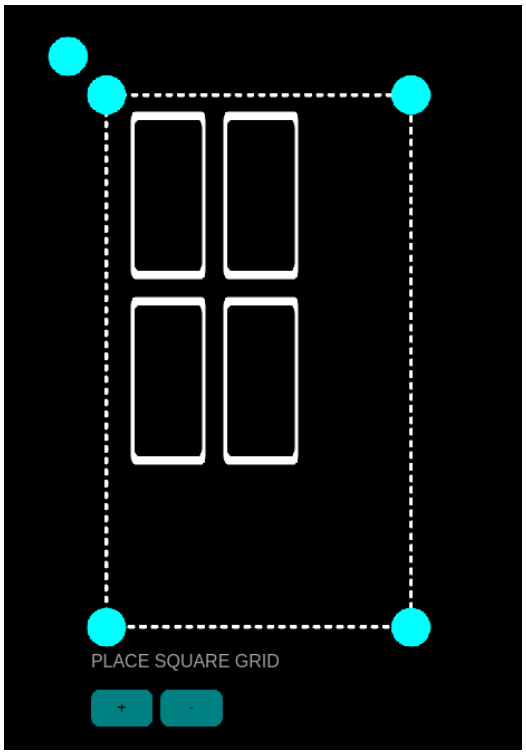
ako by užívateľ ukončil zmenšovanie a dalo by objekty na správne miesta, som sa rozhodol ho vyriešiť. Na obrázku 6.5 možno vidieť rovnaký postup pri zmenšovaní objektu, ale tu už spomínaný problem nenastáva.

Po vykreslení objektov je nutné ich polohy dostať do triedy *art_brain*, ktorá ich neskôr použije na presúvanie objektov do tohoto gridu. Tu sa tiež využíva ROS správa ProgramItem, ako tomu bolo aj pri bodoch gridu, avšak v tomto prípade je to parameter *geometry_msgs/PoseStamped[]*, ktorého definícia je:

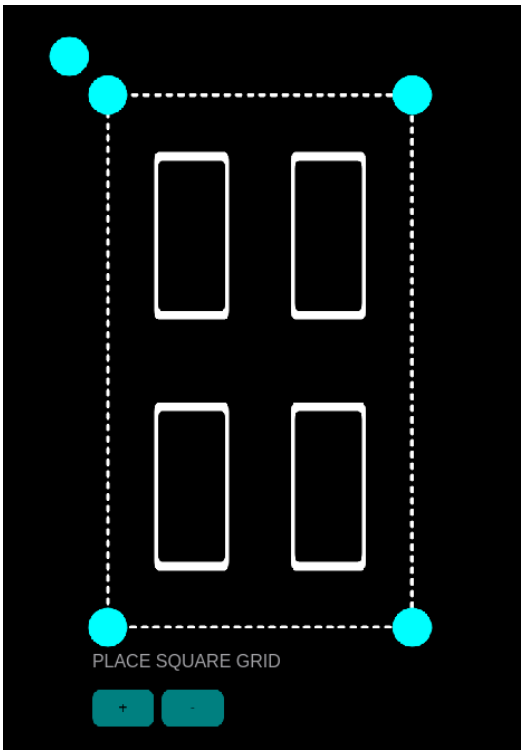
```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
geometry_msgs/Pose pose
  geometry_msgs/Point position
    float64 x
    float64 y
    float64 z
  geometry_msgs/Quaternion orientation
    float64 x
    float64 y
    float64 z
    float64 w
```

6.2.3 Rovnomerné rozloženie objektov v gride

Akonáhle užívateľ skončí s úpravami veľkosti gridu, prebehne automatické pozmenenie polôh objektov v gride tak, aby boli medzi objektami približne rovnaké medzery. Existujú dva dôvody pre dané riešenie (i) pokiaľ by sa ukladali objekty do krabice, tak je určite vhodnejšie, keď je váha objektov rozložená, a (ii) esteticky lepšie pôsobí na užívateľa. Na obrázku 6.6 možno vidieť, ako by to vyzeralo bez tejto funkcionality. Na obrázku 6.7 možno vidieť s danou funkcionalitou.



Obr. 6.6: Objekty pred rovnomerným rozmiestnením



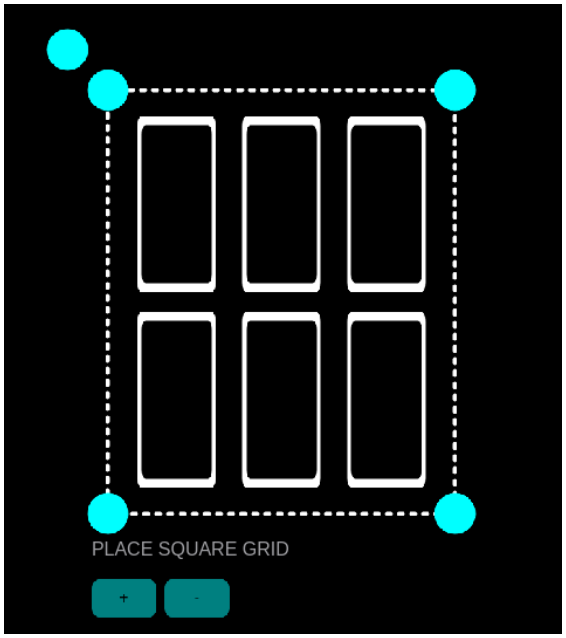
Obr. 6.7: Objekty po rovnomernom rozmiestnení

6.2.4 Zmena medzery, rotácia a presun celého gridu

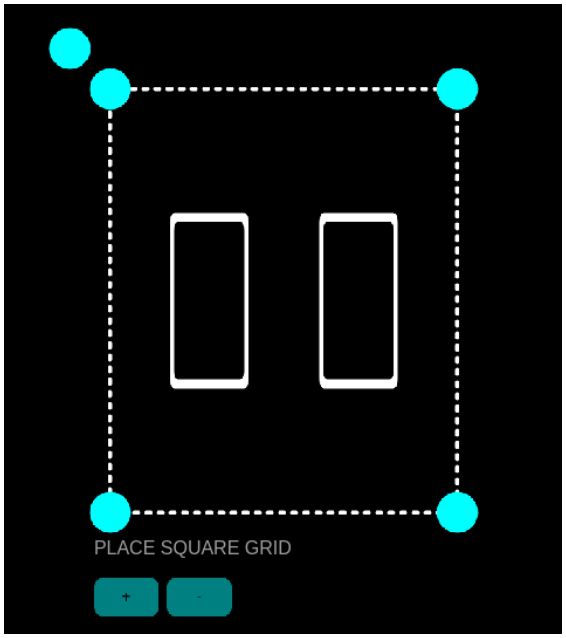
Užívateľ si tiež môže podľa potreby meniť medzery medzi objektami a to pomocou dvoch tlačítok, ktoré sú umiestnené pod gridom a sú vytvorené pomocou triedy ButtonItem. Jakmile je detegované stlačenie jedného z tlačítok, tak sa zvýši alebo zníži hodnota premennej reprezentujúcej medzeru a zavolá sa metóda *point_changed()*, ktorá znova prepočíta, koľko sa zmestí maximálne objektov do gridu pri novej medzere a následne ich prekreslí. Tento proces môžeme vidieť na obrázkoch 6.8 a 6.9.

Medzera má preddefinovanú minimálnu hodnotu, pretože treba zohľadniť ešte miesto, ktoré zaberá ruka robota pri držaní tohto objektu. Trieda teda zabezpečuje, aby užívateľ nebol schopný znížiť medzeru pod túto hodnotu.

Pokiaľ je vykreslený grid fixný, tak tlačítka nie sú aktívne a sú vykreslené šedo, čo môžeme vidieť na obrázku 6.2.



Obr. 6.8: Objekty pred zvýšením medzery



Obr. 6.9: Objekty po zvýšení medzery

Užívateľ má tiež možnosť uložiť objekty do gridu pootočené. Táto funkcia je už implementovaná triedou PlaceItem, avšak bolo nutné ju trochu upraviť pre potreby tejto práce, pretože pôvodná verzia vytvárala hneď tri komplikácie v gride:

1. Text pod objektami - trieda PlaceItem pod objekt zobrazuje text s typom objektu (obrázok 6.10), čo je pre potreby gridu zbytočné. Na vyriešenie tejto komplikácie stačilo pridať jeden parameter, avšak stále ostávali ešte dve.

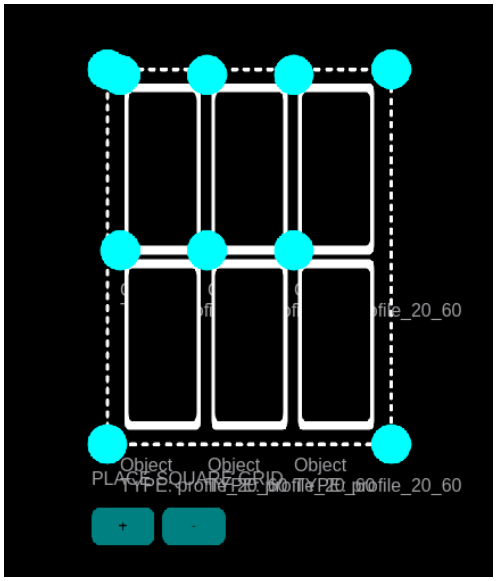
2. Rotačný bod u každého objektu - na obrázku 6.11 možno vidieť, ako by vyzeral grid, pokiaľ by sa neupravila existujúca implementácia rotácie pre jeho požiadavky. Grid by sa stal veľmi neprehľadný, pokiaľ by neboli nastavené dostatočné medzery. Naviac je pravdepodobnejšie, že užívateľ bude do gridu ukladať všetky objekty rovnako natočené. Z tohoto dôvodu je ďaleko praktickejšie, aby bol vykreslený rotačný bod iba u jedného z objektov a bolo ním možné natočiť všetky objekty súčasne. Po úprave triedy PlaceItem sa teda vykresľuje rotačný bod už len u jedného z objektov a je ním možné natočiť všetky objekty v gride súčasne.

3. Rotačný bod v rohu objektu - pokiaľ by rotačný bod ostal v ľavom rohu objektu a medzera by nebola dostatočne veľká, bolo by takmer nemožné rozlíšiť rotačný bod od rohu gridu, ako je tomu na obrázku 6.12. Riešením je presunutie rotačného bodu až za roh gridu (napríklad obrázok 6.8), pričom záleží, s ktorým rohom gridu užívateľ naposledy pracoval. Rotačný bod sa vždy vykresľuje objektu pri rohu, ktorý je oproti rohu, s ktorým užívateľ manipuluje/manipuloval naposledy, pretože tento jediný roh sa pri zmenšovaní/zväčšovaní nehýbe a teda aj najlepšie pôsobí na užívateľa.

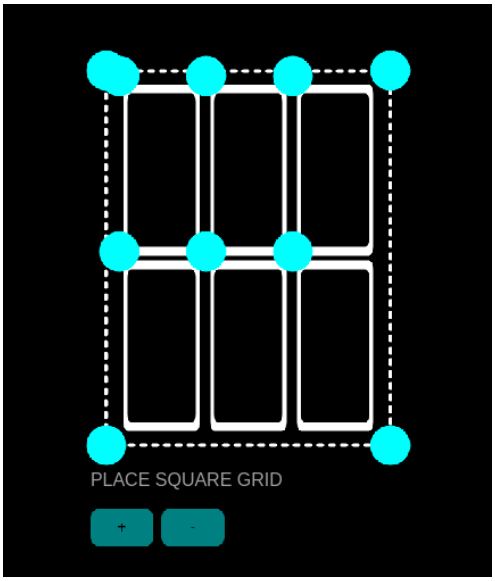
Grid po odstránení týchto komplikácií možno vidieť napríklad na obrázku 6.8.

Trieda taktiež ponúka užívateľovi presunúť celý grid spolu aj s objektami v ňom. Ak teda bude užívateľ potrebovať presunúť grid na iné miesto na stole, tak nebude nútený ho

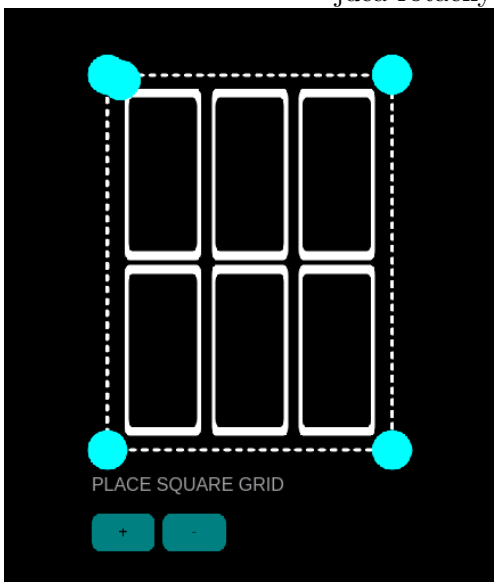
presúvať pracne a neprakticky pomocou rohov, ale ho len zaklikne a presunie, kam bude chcieť.



Obr. 6.10: Trieda PlaceItem zobrazujúca text pod objektami



Obr. 6.11: Trieda PlaceItem zobrazujúca rotačný bod každému objektu

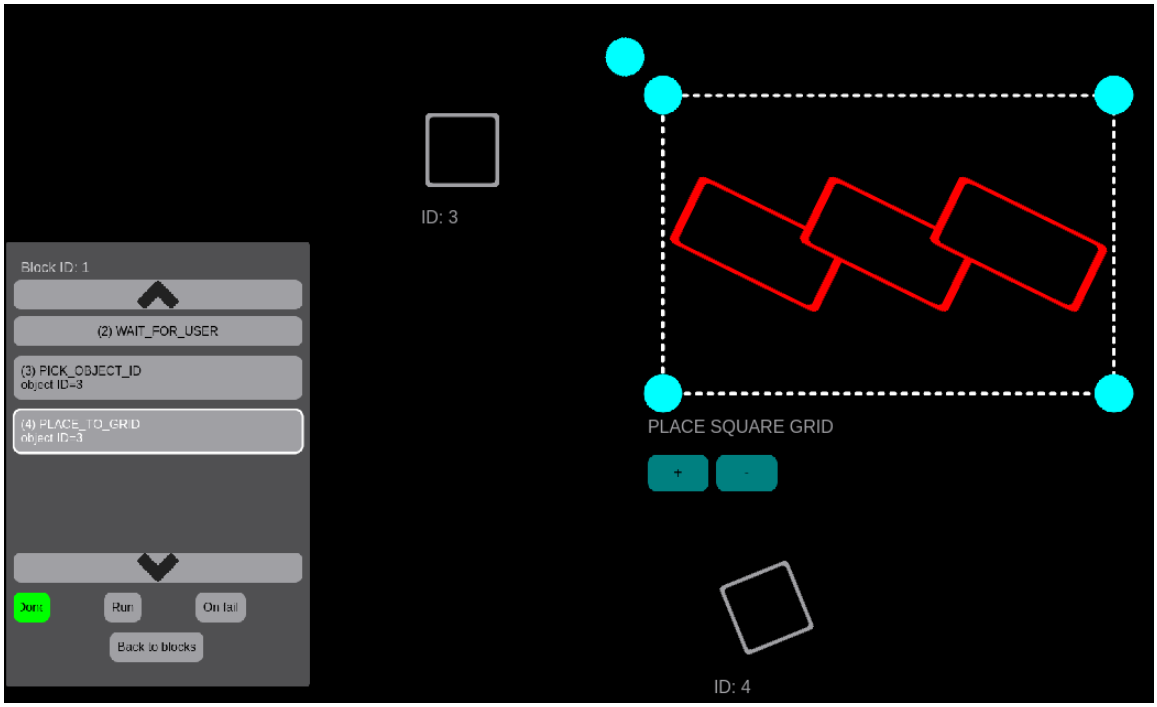


Obr. 6.12: Splývajúci rotačný bod s rohom gridu.

6.2.5 Kontrola kolízií pri učení robota

Ako už bolo spomenuté v kapitole venovanej opisu systému ARTable, pokiaľ niektorá z komponent programu je červená, robota je nutné doučiť potrebné informácie. Avšak toto učenie nemusí vždy dopadnúť úspešne. Jeden takýto príklad možno vidieť na obrázku 6.13, kde

sa užívateľ snaží naučiť robota uložiť do gridu 6 objektov, pričom ignoruje, že mu systém dáva zjavne najavo, že by nastala v tomto prípade kolízia objektov.

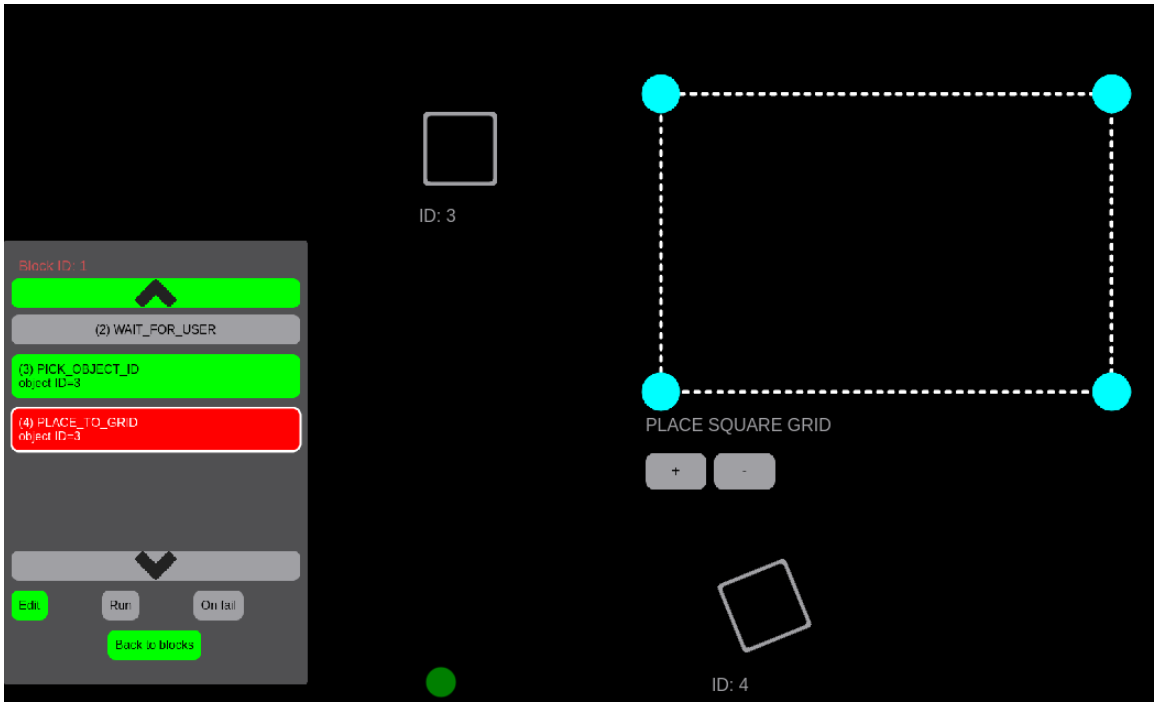


Obr. 6.13: Systém dáva najavo, že by v tomto prípade nastala kolízia objektov

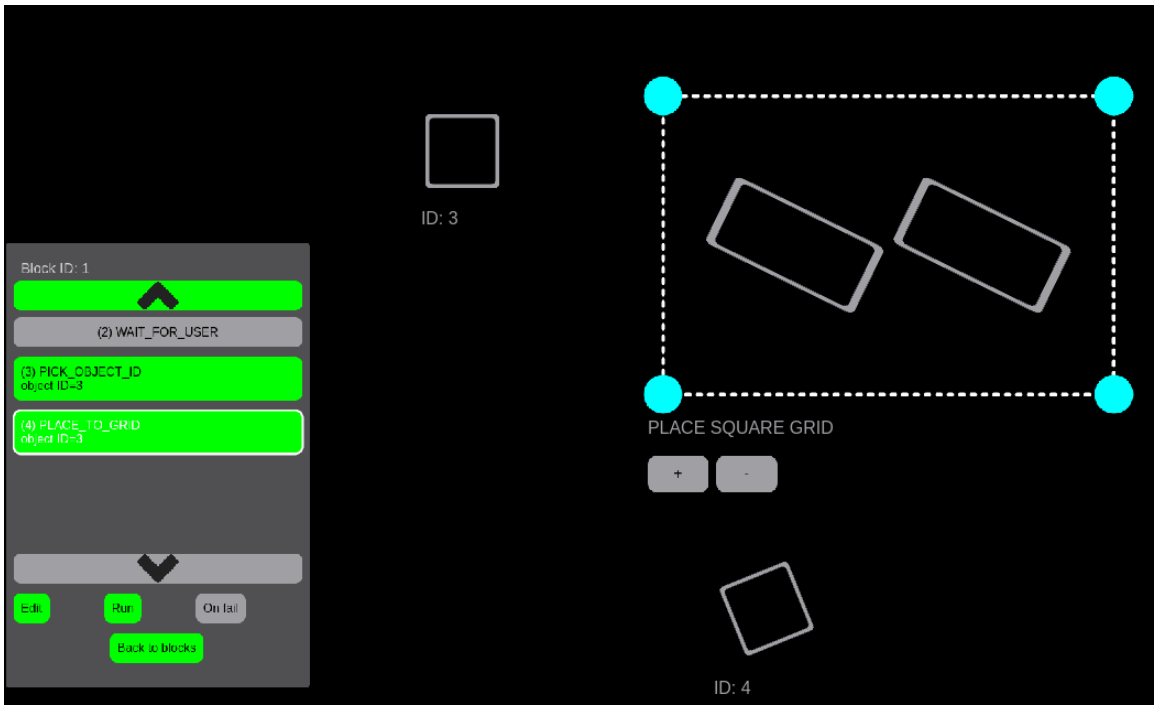
Tento problém je vyriešený veľmi jednoducho. Ak sú objekty v kolízii, tak sa ich polohy neuložia do ROS zprávy, ktorá sa po doučení kontroluje. Je nutné, aby obsahovala jak polohu gridu (pozície 4 bodov), tak polohu aspoň jedného objektu, aby sa komponenta považovala za kompletne naučenú a zozelenala.

Preto hoci sa zdá, že užívateľ robota doučil potrebné informácie, tak v skutočnosti sa neuložia a daná komponenta ostane červená (obrázok 6.14) a užívateľ je stále nútený pred spustením programu robota požadované informácie doučiť.

Na obrázku 6.15, už vidíme, že užívateľ zväčšil medzery medzi objektami a už nedochádza ku kolízii a teda učenie prebehlo úspešne.



Obr. 6.14: Učenie robota nebolo úspešné



Obr. 6.15: Učenie robota bolo úspešné

6.3 Rozšírenie art_brain

Užívateľ už je schopný v GUI naučiť robota uložiť objekty do gridu vďaka triedam na jeho tvorbu. Avšak je potrebné rozšíriť aj art_brain, aby bol schopný aj takýto program vykonať.

6.3.1 Podpora novej inštrukcie

Ako prvé bolo nutné pridať nový typ inštrukcie PLACE_TO_GRID do správy ProgramItem, aby následne mohla byť pridaná funkcionalita do art_brain, ktorá zabezpečí samotné ukladanie objektov do gridu.

Keď teda upravený art_brain narazí na položku programu typu PLACE_TO_GRID, tak v úvode sa bude postupovať veľmi podobne, ako tomu je u PLACE_TO_POSE. Na začiatku sa skontroluje či bola zadaná aspoň jedna poloha pre uloženie objektu a referencia na inštrukciu zaisťujúcu uchopenie, pomocou ktorej sa určí rameno, v ktorom je objekt. Rozdiel nastáva pri určovaní polohy pre uloženie.

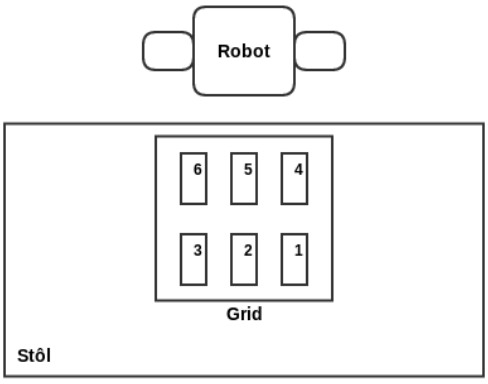
6.3.2 Určenie polohy pre uloženie

Správa ProgramItem obsahuje pri tomto type inštrukcie pole všetkých možných polôh v gride, pretože rozšírenie GUI je navrhnuté tak, že sa vždy vypočítajú a uložia do správy polohy pre maximálny počet objektov, ktorý by sa do gridu pri zadaných parametroch zmestil. Pri tejto inštrukcii sa tiež ráta s tým, že sa bude vykonávať v slučke.

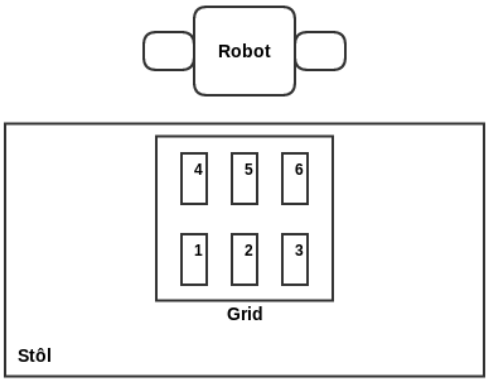
Pre každý objekt, ktorý sa má vložiť do gridu, sa vezme prvá poloha z pola, ktorá sa po jeho uložení z neho zároveň odstráni. To akým spôsobom sa budú ukladať objekty do gridu je teda určené tým, v akom poradí sa v GUI uložia do pola v ProgramItem správe.

6.3.3 Spôsob ukladania objektov do gridu

Ukladanie objektov do gridu je realizované po riadkoch, pričom sa začína od riadku, ktorý je najďalej od robota. Na obrázkoch 6.16 a 6.17 možno vidieť dva postupy, ktoré sa používajú. Prvý postup ukladá objekty sprava doľava a druhý ukladá zľava doprava. O tom, ktorý z postupov sa použije, sa rozhoduje na základe použitého rohu pri upravovaní veľkosti gridu.



Obr. 6.16: Postup ukladania objektov sprava doľava

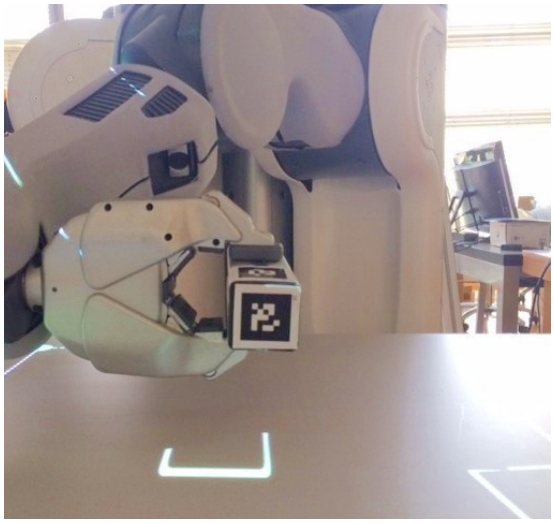


Obr. 6.17: Postup ukladania objektov zľava doprava

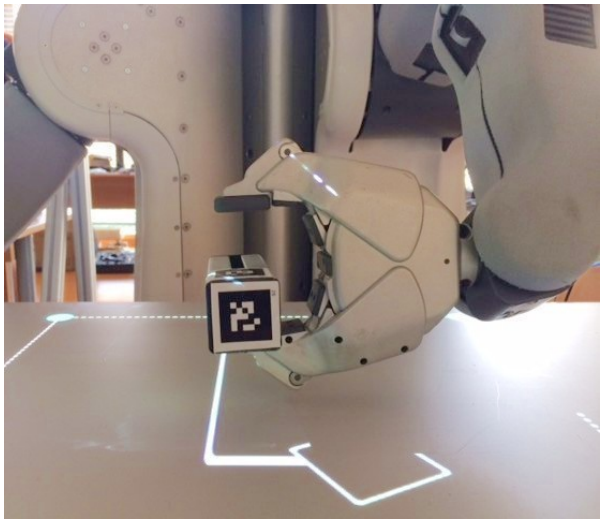
6.3.4 Úprava art_pr2_grapsing a moveit_simple_grasps

Existujúce riešenie by vytváralo problémy v prípade, že by sa vybral úchop zo strany, pretože úchopy zo strany potrebujú viac miesta. Naviac by obrovskou prekážkou boli steny krabice, ktorú grid reprezentuje. Úchopy z vrchu tiež nebolo možné použiť, pretože robot, okrem 2 týždňov, nebol schopný vygenerovať jediný takýto zrealizovateľný úchop.

Z týchto dôvodov sú pre ukladanie objektov do gridu použiteľné iba úchopy zo strany, ktoré pred položením objekt pootočia o 90 stupňov. Objekt sa teda bude pokladať do gridu, ako keby išlo o úchop z vrchu. Na pootočenie objektu je nutné vedieť, z ktorej strany sa uchopil. Túto informáciu moveit_simple_grasps neposkytuje, iba vyfiltruje najideálnejši zo všetkých vygenerovaných úchopov. Bolo nutné si vybrať úchopy len na x-ovej osi alebo úchopy len na y-ovej osi a preskočiť generovanie ostatných, aby bola istota, že sa uchopený objekt pootočí okolo správnej osi. Prípady otočenia objektu okolo nesprávnej osi možno vidieť na obrázku 6.18 a 6.19.



Obr. 6.18: Príklad otočenia objektu okolo nesprávnej osi



Obr. 6.19: Príklad otočenia objektu okolo správnej osi

Implementácia tejto práce využíva len úchopy na y-ovej osi, čo sa realizuje pridaním parametra *pick_only_y_axis* do moveit_simple_grasps.

Ako ďalšie bolo treba rozšíriť PickPlace.action o premennú *pick_only_y_axis* typu boolean, aby bolo možné v art_brain pri nastavovaní parametrov potrebných pri volaní art_pr2_grapsing špecifikovať, že chceme iba takéto úchopy.

Na záver robot objekty pred položením otočí o 90 stupňov okolo x-ovej osy, čo sa robí upravením orientácie daného objektu v art_brain, kde sa tiež povolí pootočenie objektu o 180 stupňov okolo z-ovej osy, aby bolo viac zrealizovateľných úchopov. Ukážka implementácie otáčania v art_brain:

```
# povolenie pootočenia o 180 stupňov okolo z-ovej osy
goal.z_axis_angle_increment = 3.14

# otočenie objektu o 90 stupňov okolo x-ovej osy
goal.pose.pose.orientation.x = math.sqrt(0.5)
goal.pose.pose.orientation.w = math.sqrt(0.5)
```

6.4 Testovanie

Testovanie riešenia tejto bakalárskej práce prebiehalo v dvoch fázach, a to testovaním funkčnosti:

- rozšírenia GUI
- upraveného generovania úchopov a pokladania objektov do gridu

6.4.1 Testovanie GUI

Na testovanie rozšírenia GUI sa využívala vlastnosť systému ARTable a to zobrazenie debuggovacie okna. Týmto spôsobom bolo možné priebežne kontrolovať správnosť funkčnosti vytvoreného gridu a odhalovať jeho nedostatky kedykoľvek bez potreby fyzickej interakcie s robotom. Približne každé dva týždne bola podoba GUI prezentovaná vedúcemu tejto práce a na základe jeho spätnej väzby sa upravovala, prípadne rozširovala o ďalšiu funkcionalitu.

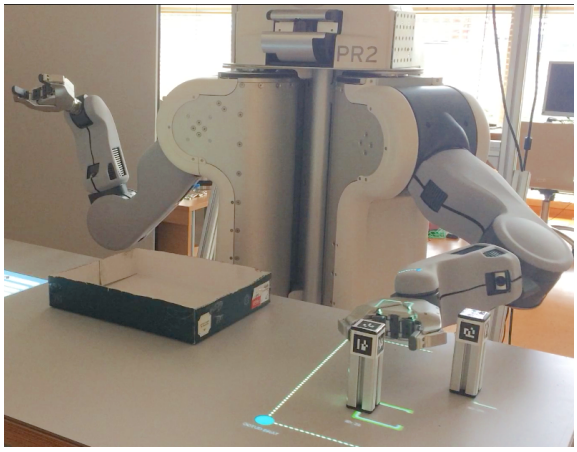
Akonáhle bola dosiahnutá finálna podoba daného rozšírenia, tak správnosť GUI bola samozrejme otestovaná aj priamo na pracovisku ARTable v laboratóriu, pre prípadné odhalenie nedostatkov, ktoré by sa v debuggovaciom okne neobjavili.

6.4.2 Testovanie úchopu a pokladania

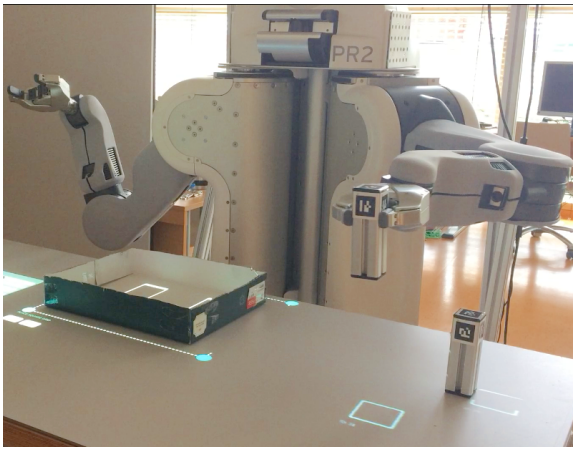
Po úprave generovania úchopov bolo nutné otestovať funkčnosť na robotovi, čo prebiehalo opakovaným presúvaním objektov z najrôznejších pozícií na stole a ich uložením na najrôznejšie miesta. Taktiež prebiehalo testovanie minimálnej potrebnej medzery medzi objektami a hranami gridu.

Na záver sa funkčnosť inštrukcie vyskúšala formou jednoduchého testu, ktorý simuloval použitie v praxi, kde cieľom bolo uložiť 2 objekty do krabice. Tieto objekty boli umiestnené do polygonu, z ktorého ich následne robot po jednom uchopil, pootočil a položil do krabice. Test vo väčšine prípadov dopadol úspešne, občas sa však stalo, že výber najideálnejšieho úchopu zlyhal a užívateľ musel určiť robotovi, aby sa pokúsil o rovnaký úkon ešte raz.

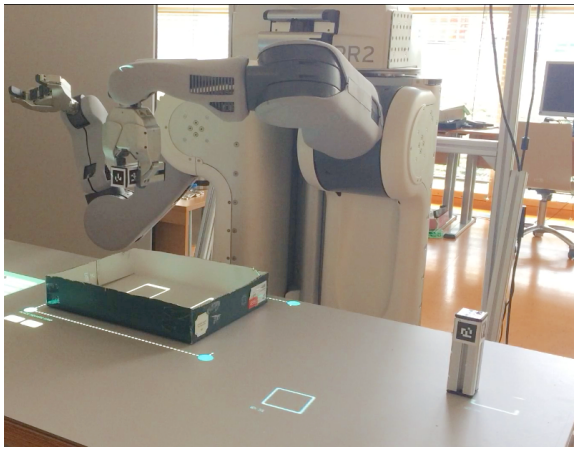
Priebeh testu možno vidieť na obrázkoch 6.20 až 6.27.



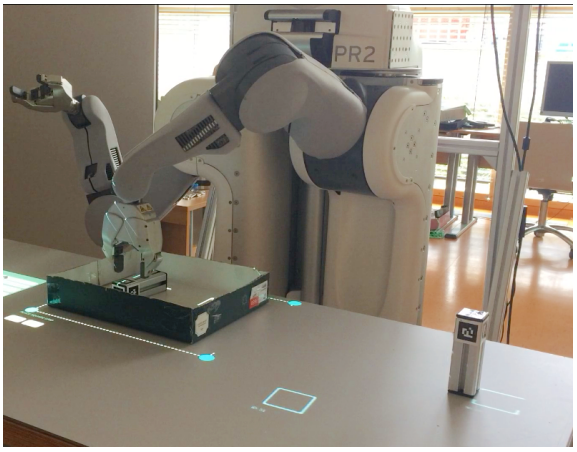
Obr. 6.20: Robot ide uchopiť prvý objekt



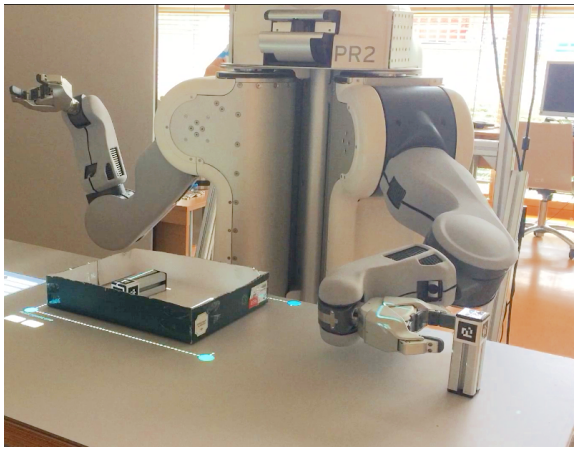
Obr. 6.21: Robot presúva prvý objekt



Obr. 6.22: Robot pokladá prvý objekt do krabice



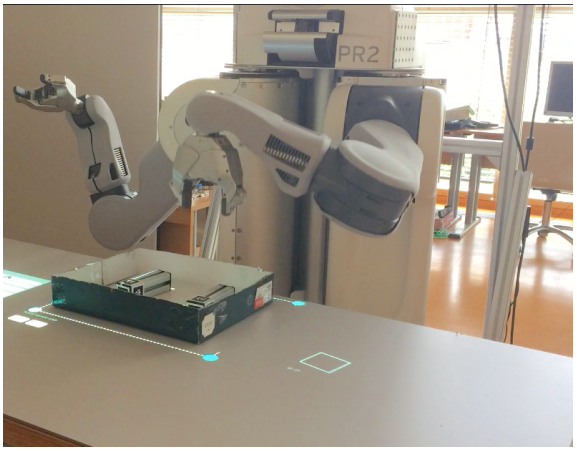
Obr. 6.23: Robot uložil prvý objekt do krabice a vracia sa do pôvodnej polohy



Obr. 6.24: Robot ide uchopiť druhý objekt

Obr. 6.25: Robot presúva druhý objekt

Obr. 6.26: Robot pokladá druhý objekt do krabice



Obr. 6.27: Robot uložil druhý objekt do krabice a vracia sa do pôvodnej polohy

Kapitola 7

Závěr

Cieľom tejto bakalárskej práce bolo rozšíriť množinu robotom podporovaných inštrukcií a umožniť užívateľovi naučiť robota uložiť objekty rovnakého typu, napríklad do krabice. Implementácii predchádzalo podrobné štúdium systému ARTable, framework-u Qt a robotieho operačného systému (ROS). Výsledné riešenie sa podarilo úspešne integrovať do už existujúceho systému a je plne funkčné s obmedzením len na úchopy na y-ovej osi a umožňuje užívateľovi naučiť robota nový typ úlohy. Učenie robota prebieha prostredníctvom rozšírenia grafického užívateľského rozhrania.

Využiteľnosť pridanej inštrukcie v praxi bola vyskúšaná formou testu, ktorý spočíval v uložení dvoch objektov do krabice. Test vo väčšine prípadov dopadol úspešne, občas však zlyhal výber najideálnejšieho úchopu, čo bolo spôsobené práve obmedzením generovania úchopov len na y-ovú os.

Implementované riešenie ukladania objektov, založené na generovaní úchopov len na y-ovej osi a ich následnom otočení o 90 stupňov okolo osy x, zaisťuje postačujúce výsledky. Avšak takto vygenerované úchopy, ktoré sú vybrané ako najideálnejšie, môžu znamenať zbytočne komplikovanú trajektóriu pre rameno robota. V skutočnosti by možno existovalo jednoduchšie a rýchlejšie riešenie, pokiaľ by sa v danom prípade brali do úvahy aj úchopy na x-ovej osi. Ďalší vývoj by mohol spočívať v rozlišovaní týchto úchopov, aby robot mohol pracovať so všetkými úchopmi a zároveň dokázal otočiť objekt okolo spravnej osi. Riešením by možno mohlo byť využitie Kinectu na detegovanie, z ktorej strany „ruka“ objekt uchopuje.

Literatúra

[1] Alexandrova, S.; Tatlock, Z.; Cakmak, M.: RoboFlow: A flow-based visual programming language for mobile manipulation tasks. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, Máj 2015, ISSN 1050-4729, s. 5537–5544.

[2] Association, B. A. . R.: Robot programming methods. 2017, [Online; navštívené 24.03.2017].
URL http://www.bara.org.uk/robots/robot-programming-methods.html

[3] Bernier, C.: Collaborative Robot Series : PR2 from Willow Garage. Jún 2013, [Online; navštívené 01.05.2017].
URL http://blog.robotiq.com/bid/65419/Collaborative-Robot-Series-PR2-from-Willow-Garage

[4] Blanchette, J.; Summerfield, M.: *C++ GUI programming with Qt 4*. Upper Saddle River : Prentice-Hall, druhé vydání, 2008, ISBN 0-13-235416-0.

[5] Company, T. Q.: About Qt. 2017, [Online; navštívené 20.04.2017].
URL http://wiki.qt.io/About_Qt

[6] Company, T. Q.: Graphics View Framework. 2017, [Online; navštívené 20.04.2017].
URL http://doc.qt.io/qt-4.8/graphicsview.html

[7] Company, T. Q.: Signals & Slots. 2017, [Online; navštívené 20.04.2017].
URL http://doc.qt.io/qt-5/signalsandslots.html

[8] Dietz, T.; Schneider, U.; Barho, M.; aj.: Programming System for Efficient Use of Industrial Robots for Deburring in SME Environments. In *ROBOTIK 2012; 7th German Conference on Robotics*, Máj 2012, s. 1–6.

[9] Foundation, O. S. R.: ROS. 2017, [Online; navštívené 15.04.2017].
URL http://wiki.ros.org/ROS/Introduction

[10] Garage, W.: Personal Robot 2. 2015, [Online; navštívené 24.04.2017].
URL http://www.willowgarage.com/pages/pr2/overview

[11] Gaschler, A.; Springer, M.; Rickert, M.; aj.: Intuitive robot tasks with augmented reality and virtual obstacles. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, Máj 2014, s. 6026–6031.

[12] Guerin, K. R.; Riedel, S. D.; Bohren, J.; aj.: Adjutant: A framework for flexible human-machine collaborative systems. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, September 2014, s. 1392–1399.

[13] Gutierrez, J. H.; Perez, A. N.: Combination of two programming by demonstration techniques to improve the quality of a grasp. In *2016 IEEE Colombian Conference on Robotics and Automation (CCRA)*, September 2016, s. 1–6.

[14] Hein, B.; Hensel, M.; Worn, H.: Intuitive and model-based on-line programming of industrial robots: A modular on-line programming environment. In *2008 IEEE International Conference on Robotics and Automation*, Máj 2008, s. 3952–3957.

[15] Materna, Z.; Kapinus, M.; Beran, V.; aj.: Using Persona, Scenario, and Use Case to Develop a Human-Robot Augmented Reality Collaborative Workspace. In *HRI 2017*, Association for Computing Machinery, 2017, ISBN 978-1-4503-4885-0, s. 1–2.
URL http://www.fit.vutbr.cz/research/view_pub.php.en?id=11301

[16] Materna, Z.; Kapinus, M.; Španěl, M.; aj.: Simplified Industrial Robot Programming: Effects of Errors on Multimodal Interaction in WoZ experiment. In *Robot and Human Interactive Communication (RO-MAN)*, Institute of Electrical and Electronics Engineers, 2016, ISBN 978-1-5090-3929-6, s. 1–6.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=11106

[17] Microsoft: Skeletal Tracking. 2017, [Online; navštívené 27.04.2017].
URL https://msdn.microsoft.com/en-us/library/hh973074.aspx

[18] Owen-Hill, A.: What Are the Different Programming Methods for Robots? Marec 2016, [Online; navštívené 29.04.2017].
URL http://blog.robotiq.com/what-are-the-different-programming-methods-for-robots

[19] Pan, Z.; Polden, J.; Larkin, N.; aj.: Recent progress on programming methods for industrial robots. *Robotics and Computer-Integrated Manufacturing*, ročník 28, č. 2, 2012: s. 87 – 94, ISSN 0736-5845.

[20] Perzylo, A.; Somani, N.; Profanter, S.; aj.: Multimodal binding of parameters for task-based robot programming based on semantic descriptions of modalities and parameter types. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Workshop on Multimodal Semantics for Robotic Systems*, Hamburg, Germany, 2015.

[21] Pires, J. N.; Veiga, G.; Araújo, R.: Programming-by-demonstration in the coworker scenario for SMEs. *Industrial Robot: An International Journal*, ročník 36, č. 1, 2009: s. 73–83.

[22] Sucan, I. A.; Chitta, S.: MoveIt! 2017, [Online; navštívené 20.04.2017].
URL http://moveit.ros.org/

Prílohy

Príloha A

Blokové schéma systému ARTable

Obr. A.1: Blokové schéma software komponent systému ARTable