VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

GRAFICKÉ UŽIVATELSKÉ ROZHRANÍ PRO OVLÁDÁNÍ ROBOTA

GRAPHICAL USER INTERFACE FOR ROBOT CONTROL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PAVEL PAKOSTA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JAROSLAV ROZMAN, Ph.D.**
SUPERVISOR

BRNO 2018

Abstrakt

Tato práce se zabývá tvorbou jednoduchého grafického rozhraní pro ovládání robota. Cílem práce je poskytnout uživatelům jednoduchý prostředek, který jim umožní spouštět ROS programy pro ovládání robota a zároveň je zbaví možnosti přímé manipulace se systémem robota. Pro možnost správy programů robota, je vytvořena pomocná aplikace, umožňující nahrávat a odstraňovat programy z robota. Obě aplikace jsou implementovány v jazyce C++ a prostředí Qt. Pro navázání spojení mezi robotem a pomocnou aplikací je použita knihovna libssh. Aplikace je cílena na zařízení Odroid xu4 s operačním systémem Ubutnu 16.04.

Abstract

This project undertakes the creation of a simple GUI for manipulation of a robot. The goal of this project is to provide the user with an elementary device which will later on allow the initiation of ROS programs for the manipulation of a robot, while at the same time not allowing the user to directly interface with the robots system. Would there occur the need to change or interact with the robots programs in any way, a created conducive application will allow the user to add or delete them. Both applications are implemented in the C++ programming language and a Qt interface. The library libssh can establish communication between the robot and the conducive application. The main application is designed for Ondroid xu4 device with OS Ubuntu 16.04.

Klíčová slova

C++, ROS, Qt, gui, libssh

Keywords

C++, ROS, Qt, gui, libssh

Citace

PAKOSTA, Pavel. *Grafické uživatelské rozhraní pro ovládání robota*. Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jaroslav Rozman, Ph.D.

Grafické uživatelské rozhraní pro ovládání robota

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jaroslava Rozmana, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Pavel Pakosta
16. května 2018

Poděkování

Děkuji panu Ing. Jaroslavovi Rozmanovi, Ph.D. za velmi užitečnou pomoc, kterou mi poskytl při vypracování mé bakalářské práce.

Obsah

Kapitola 1

Úvod

Uživatelská rozhraní, která v posledních letech zažívala stagnaci, začala zase být rozvíjeným oborem a to díky nástupu nových technologií, jako jsou dotykové obrazovky, snímače pohybu atd. Ne jinak je na tom robotika, která se v poslední době stává běžnou součástí života široké škály lidí.

Grafické uživatelské rozhraní, také označované jako GUI (angl. Graphical User Interface), je nejrozšířenější metoda pro komunikaci mezi uživatelem a aplikací. GUI je sestaveno z objektů, jako jsou tlačítka, rozbalovací menu, zatrhávací pole, a tyto objekty jsou vykreslovány na plochu a uspořádány do oken. Uživatel pomocí vstupních zařízení, v našem případě dotykové obrazovky, mění stav aplikace nebo jsou mu zpřístupněny informace získané interakcí systému s okolím. Vždy je potřeba brát v úvahu cílového uživatele - aplikace by měla být zjednodušením jeho práce, ne přítěží.

Robotika je rozsáhlý obor a pro samotného programátora by byl velice nesnadný úkol vytvořit celý systém od začátku do konce. Proto existují různé skupiny, které vytvářejí software tak, aby byl volně využitelný a snadno rozšiřitelný. Jednou z nich je ROS (Robot Operating System), jedná se o framework k vytváření programů pro roboty.

V rámci této práce budu vytvářet GUI pro školního robota, který má dotykovou obrazovku pro ovládání. Tento robot je využívaný studenty na škole, kteří pro něj mohou vytvářet programy. Vytvářené GUI by mělo uživateli umožnit snadný přístup k nahraným programům na robotovi a jejich spuštění. Také by měl přehledně zobrazovat data ze svých senzorů a zpřístupňovat ROS výstupy spuštěných programů.

Kapitola 2

Teorie

2.1 Robotický operační systém

Robotický operační systém (ROS) [7] je flexibilní framework pro vývoj robotického softwaru. Jedná se o kolekci nástrojů a knihoven, které si kladou za cíl zjednodušit vývoj komplexních a robustních systémů pro ovládání robotů. Systém je volně dostupný, podporovaný rozsáhlou komunitou, která vyvíjí nové nástroje, balíčky a knihovny. Jelikož se systém stále vyvíjí, jsou nabízeny stále nové distribuce. Školní distribuce je ROS Indigo, ale pro svoji práci jsem si zvolil novější verzi ROS Kinetic. Hlavním důvodem byly technické problémy, při instalaci verze Indigo, a to nekompatibilita tohoto softwaru s hardwarem, který mám k dispozici. Architektura ROSu je tvořena uzly, které spolu komunikují posíláním zpráv a voláním služeb.

Uzly [8] jsou procesy, které provádí výpočet. Jednotlivé uzly jsou navzájem závislé, tvoří hierarchickou strukturu, a mezi sebou komunikují pomocí streamovacích témat (topics), do kterých se přihlásí, nebo voláním služeb. Každý uzel se pak stará o určitou část výsledné aplikace. Například jeden se stará o motor kol, další plánuje cestu atd. Rozdělení programu do uzlů nese řadu výhod. Jednou z hlavních výhod je tolerance chyb. Jelikož jsou poruchy izolovány v jednotlivých uzlech, nedochází pak k pádu celého systému.

Zprávy jsou hlavním způsobem komunikace mezi uzly. Zprávy jsou posílány do určitých témat, z nichž je pak uzly odebírají. Komunikace probíhá pouze jednosměrně, tedy témata mají své přispěvatele a odběratele. Díky tomu se dají snadno přenášet data. Například jeden uzel zajišťuje snímání obrazu. Data, která nasnímá, posílá na své téma, odkud jiné uzly obraz odebírají a dále ho zpracovávají.

Služby – jedná se o další způsob komunikace mezi uzly. Komunikace probíhá stylem dotaz-odpověď. Uzel, který danou službu implementuje, na rozdíl od zpráv svoje data neposílá neustále na téma, ale odpovídá pouze na požádání.

Gazebo je aplikace pro simulaci různých robotických platforem. Umožňuje simulovat chování robota v komplexním venkovním, či vnitřním prostředí. Tak můžeme získat data ze senzorů v závislosti na prostředí, ve kterém se robot pohybuje. Tudíž můžeme pomocí tohoto nástroje bezpečně testovat různé metody a aplikace bez rizika poškození reálného robota.

Rviz [9] je vizualizační 3D prostředí, díky němuž můžeme zobrazit to, co robot vidí, myslí, nebo dělá. Vidíme pohled robota, ať už pochází z kamer nebo laserů. Lze si zvolit, která témata chceme odebírat a jejich data následně vizualizovat. Uživatel tak může zobrazovat obraz z kamery, point cloud z Kineticu, nebo polohové vlastnosti robota.

2.2 Typy senzorů

Každý robot má obvykle velké množství senzorů, které mohou být různých typů: mechanické, optické, či zvukové. Aby bylo možné tyto data nějak využít, je potřeba je vizualizovat. Problém je, že senzory poskytují data různého charakteru - 2D nebo 3D - proto se před vizualizací musí zpracovat.

Sonar

Sonar je zařízení, které využívá zvukových vln k navigaci, komunikaci nebo detekci objektů. Sonar může být pasivního typu, slouží pouze k poslouchání, nebo aktivní, který pomocí vysílání zvukových vln a poslouchaním jejich odrazů určuje vzdálenost překážky. Při znalosti rychlosti šíření zvuku v daném prostředí se určí vzdálenost překážky pomocí rozdílu doby, kdy byl signál vyslán, a doby, kdy byl přijat odražený zpět.

Laserové měření

Jedná se o velice přesné měření objektů a překážek pomocí laseru. Zařízení vysílá laserové paprsky (viditelné, či neviditelné) a pomocí senzorů zachytává odražené paprsky, čímž si vytváří množinu bodů se souřadnicemi. Laserové měření se obvykle požívá pro měření v rovině, ale pomocí naklánění senzorů lze přidat i třetí rozměr.

Encodér

Encodér je snímač, který převádí pohyb na elektrický signál, který může být použit pro určení polohy, změření rychlosti nebo určení směru otáčení. Pro vytvoření signálu se používají různé technologické metody: mechanické, magnetické, odporové a optické. Nejčastějším typem jsou optické, kdy světlo vyzařované z LED diody, prochází přes clonu a dopadá na fotodetektor. Clona je otáčivé mezikruží, které je pravidelně rozděleno na úseky světlopropustné a světlonepropustné. Otáčením se střídají jednotlivé úseky a tím přerušují tok světla ze zdroje. Encodér může být na robotovi použit pro zjišťování směru a rychlosti otáčení kol.

2.3 Qt

Qt (výslovnost „kjůt“)[6] je multiplatformní vývojový framework pro desktopy, vestavěné systémy a mobily. Je primárně určený pro jazyk C++, ale dá se použit i s ostatními jazyky, například: Python Java, C# atd. Mezi platformy které jsou podporovány Qt patří: Linux, OS X, Windows, BlackBerry a další. Qt samo o sobě není programovací jazyk, ale framework napsaný v C++, který díky svému preprocesoru MOC[1] rozšiřuje jazyk C++ o vlastnosti jako jsou signály a sloty. Před samotnou kompilací zdrojových kódů MOC projde zdrojové kódy, které byly napsány v C++ rozšířeném o Qt syntaxi a vygeneruje standardní zdrojový kód. Takže je možné přeložit samotné aplikace a knihovny pomocí standardních kompilátorů jako jsou Clang, GCC, MinGw, nebo MVSC.

Framework Qt je založen na objektově orientovaném programování. Všechny elementy jsou definovány jako objekty, každý element má definované metody a díky dědičnosti lze vytvářet své objekty. Ty se potom dají rozšiřovat o další metody, které jim původně nebyly vývojáři přiděleny. Základní třídou, z které většina ostatních dědí, je QObject. Tato třída poskytuje důležité vlastnosti, mezi ně patří rodičovský systém, signály a sloty.

Rodičovský systém je strom hierarchie - každý objekt může mít svého potomka a pokud je rodičovský objekt zničen, budou zničeny i všichni potomci. Potomci objektu se dají vyhledat pomocí metody findChild. Každý potomek se automatiky objeví uvnitř svého rodičovského objektu. Díky této vlastnosti se dají objekty jednoduše vkládat do kontejnerů. Pozice vkládaných objektů, se dá ovlivnit pomocí metody SetGeometry, kdy se nastaví souřadnice a velikost objektu - souřadnice jsou brány vzhledem k rodičovskému objektu.

Signály a sloty se používají pro komunikaci mezi objekty. Pokud chceme informovat objekt o změně jiného objektu (například uživatel klikne na tlačítko „Close“), chceme předat informaci hlavnímu oknu (to zavolá funkci close()), tady přichází na řadu signály. Pokud dojde k požadované události v jednom z objektů, tak je vyvolán signál, který je připojený ke slotu jiného objektu. Slot je funkce, která se má vykonat, pokud dojde k nějaké události. Jelikož je slot zároveň normální funkcí, může být zavolán, jako každá běžná funkce. Je možné připojit jeden signál k více slotům, stejně tak se dá připojit libovolný počet signálů k jednomu slotu.

2.4 LIBSSH

Libssh [10] je knihovna napsaná v jazyku C, umožňující používat v programu SSH protokol. Knihovna libssh umožňuje vzdálené spuštění programu, přenos souborů, nebo bezpečný tunel pro vzdálené programy. SSH protokol je šifrovaný a zajišťuje integritu přenášených dat. Libssh se dá použit jak pro psaní klientských aplikací připojujících se k serveru, tak i samotných serverů. Knihovna spadá pod LPGL licenci a nemá nic společného s knihovnou libssh2.

Typické SSH spojení provádí několik kroků v průběhu připojování. Před připojením na server je potřeba zvolit šifrovací algoritmus, jméno uživatele a případně algoritmus pro kompresi. Při ustanovování spojení dochází k ověření pomocí handshake - pozdrav, při kterém dojde k ověření serveru. Poté se klient musí prokázat pomocí hesla, nebo veřejného klíče. Po ověření uživatele je potřeba otevřít kanál, pomocí kterého se bude komunikovat. Každá kanál má svůj standardní stream “stdout“ a stream pro chyby “stderr“. Počet kanálu otevřených na jednom SSH spojení není teoreticky nijak omezen. Pomocí kanálu se dají posílat příkazy na server, otevřit shell nebo přenášet soubory pomocí podsystému SFTP.

Secure File Transfer Protocol (SFTP) je protokol umožňující bezpečně přenášet data mezi lokálním a vzdáleným počítačem. Je podobný protokolu File Transfer Protocol (FTP), ale je mezi nimi několik rozdílů. FTP má textový základ a SFTP využívá paketů. Jinými slovy příkaz pro smazání souboru by byl po FTP poslán jako “DELE file.txt“ a na SFTP jako binární 0xBC a “file.txt“. Tím pádem se pomocí SFTP přenáší menší množství dat, tudíž je rychlejší a méně zatěžuje komunikační linku. Tento protokol nabízí velké množství operací nad vzdálenými soubory. Umožňuje posílání, přijímaní a upravování souborů, mazání a vytváření složek, získávání informací o vlastníkovi a skupině souborů. Při používání SFTP podsystému, programátor nepracuje přímo s SSH kanálem, ale otevírá SFTP spojení, které zapouzdřuje některé operace.

2.5 OTG

USB On-The-Go [1], dále už jen OTG, je specifikace, která umožňuje USB zařízením, jako jsou tablety, nebo chytré telefony, zastávat roli master v usb komunikaci. Díky tomu, že se

[1] Meta-Object Compiler

mohou chovat jako master zařízení, je k nim možné připojit flash disky, digitální kamery a klávesnice. OTG umožňuje těmto zařízením přepínání mezi stavy master a slave. Například telefon se může chovat jako master a číst z média data, poté se přepnout do role slave a být připojen k počítači jako externí zařízení připravené ke čtení. OTG definuje dvě role pro zařízení: OTG-A a OTG-B. OTG-A je zařízení, které poskytuje napětí do spojení, tedy je master. OTG-B je zařízení, které přijímá napětí, je tedy v roli slave. Počáteční role zařízení je určena podle typu konektoru, který je do zařízení připojen. V průběhu spojení je možné role zařízení prohodit pomoci Host Negotiation Protocol (HNP).

2.6 Zásady tvorby uživatelských rozhraní

K tomu, aby bylo možné vytvořit kvalitní a jednoduše použitelné uživatelské rozhraní, je potřeba držet se už při návrhu jistých zásad. [11] Dodržení těchto zásad nás přiblíží k tomu, aby rozhraní bylo přehledné a uživatelsky přívětivé. Důležité je, aby ovládací prvky byly dostatečně veliké a nedocházelo tak k překlikům. Zároveň je potřeba prvky nemít moc velké, aby nedocházelo k zbytečnému zabírání místa, které může být využito jinak.

Při návrhu je možné sledovat Schneidermanova kritéria pro UI:

- Viditelnost klíčových prvků
- Zřetězení všech akcí a adekvátní odpovědi od systému
- Vratnost všech uživatelských akcí tak, aby systém nebránil v experimentování
- Syntaktická korektnost taková, aby každá uživatelova akce byla legální operací
- Nahrazení komplexních příkazů jednoduchými manipulacemi s viditelnými objekty

Dalšími důležitými kritérii je **pochopitelnost**, **flexibilita** a **robustnost**.

Pochopitelnost rozhraní je parametrem vyjadřujícím schopnost člověka, který poprvé v životě vidí dané rozhraní, vyrovnat se s jeho ovládáním. Pochopitelnosti se docílí především tím, že bude rozhraní předvídatelné. Tedy pokud uživatel vidí nějaký ovládací prvek, je mu jasné, co se po jeho aktivaci stane. Pokud se systém bude chovat jiným způsobem, než očekává, nebude se systémem spokojen. Pro další zvýšení pochopitelnosti je potřebné, aby byl systém konzistentní. Tedy, aby stejné prvky rozhraní měly vždy stejný význam nehledě na to, v které části systému se vyskytují. Při návrhu by se měly brát v potaz zkušenosti, které mohl uživatel nabrat používáním jiných nástrojů, a tak mu usnadnit přechod na nový systém.

Flexibilita rozhraní udává, na kolik se dokáže rozhraní přizpůsobit uživateli. Aby bylo rozhraní flexibilní, mělo by uživateli umožnit upravovat rozložení ikon, či seskupení panelů nebo změnu vzhledu. Mělo by také uživateli při práci napovídat radou, či zamezením provedení operace, pokud by jeho činností mohlo dojít k narušení chodu aplikace.

Robustnost rozhraní zahrnuje jeho transparentnost a rychlost odezvy rozhraní. Ta by měla být co nejvyšší, aby se uživateli zdála téměř okamžitá. Pokud nejsme schopni takové rychlosti docílit, je nutné uživateli dát vědět, že zvolená akce probíhá. Uživatel by měl ve všech případech vědět, co se děje uvnitř systému tak, aby nenabyl dojmu, že se neděje nic, nebo nebyl překvapen vykonáním nějaké akce. Též by robustní rozhraní mělo umožňovat zotavení z chyb, aby nedošlo k přerušení chodu celého systému.

K navrhnutí úspěšného rozhraní je podstatné určit cílovou skupinu. Zaměřením se na tuto skupinu můžeme detailněji zkoumat požadavky na systém. Pokud má být rozhraní

přístupné více rozdílným skupinám, je potřeba zvolit kompromis, který bude vyhovovat zkušeným uživatelům i laikům. Dosáhnutí takového kompromisu je velice obtížné a vyžaduje mnohem více času a peněz, než vytvoření více specifického systému, a tento kompromis je též náročný na testování.

Kapitola 3

Návrh

3.1 Existující programy

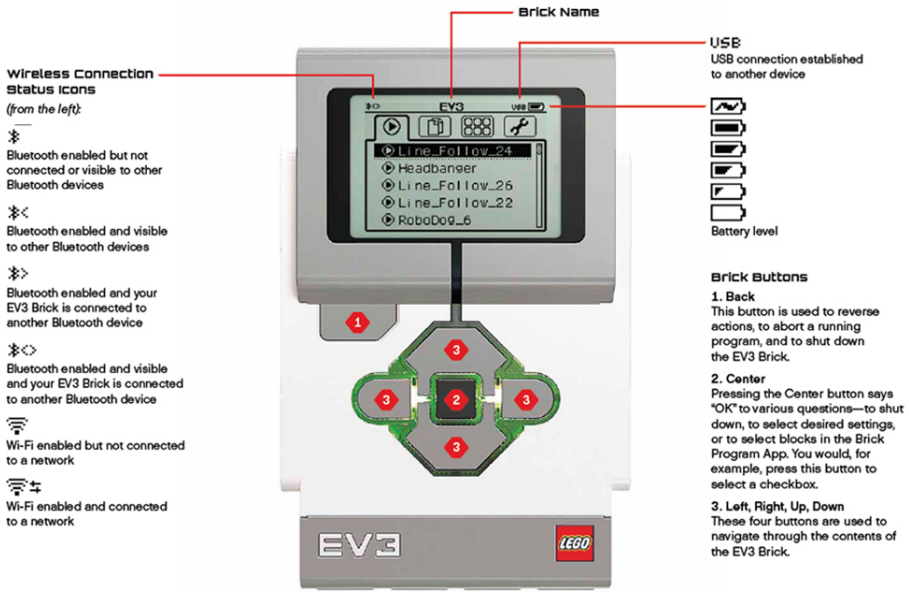
Před návrhem valstní aplikace, jsem provedl analýzu již existujících řešení. Na internetu se dají dohledat různá jednoduchá guička, vytvořená například v MATLABu, nebo právě v Qt. Zaměřil jsem se na zařízení vyráběná firmou Lego a to sérii Mindstorms. Jedná se o řadu programovatelných robotických stavebnic, umožnující si sestavit robota a napsat k němu program, který bude robot vykonávat. Lego Mindstorms je dobrý příklad, díky jeho rozšířenosti a hlavně jednoduchosti a jasnosti ovládání, takže je vhodný i pro nezkušené uživatele.

3.1.1 Lego Mindstorms EV3

Jedná se o řadu programovatelných robotických stavebnic vyráběných firmou Lego [2] umožňující si sestavit robota a napsat k němu program, který bude robot vykonávat. Lego Mindstorms sestává z několika částí: velkého motoru, středního motoru, infračerveného senzoru, barevného senzoru, dotykového senzoru a hlavně z EV3 kostky (3.1). EV3 kostka je řídící částí robota, ke které se připojují všechny ostatní součástky. Je možné sériově propojit až 4 kostky a tím zvětšit počet možných připojených zařízení a jeho výkon. EV3 kostka se dá připojit k tabletu, počítači, či mobilu a z těchto zařízení se dá robot ovládat nebo na něj přenášet napsané programy. Programy se dají vytvářet v Legem dodávaných jazycích RCX a ROBOLAB. Dají se ale použít i jiné jazyky jako je C, C++, Java nebo Visual Basic. Kostka se dá k těmto zařízením připojit třemi způsoby: pomocí mini usb kabelu, bezdrátově přes wi-fi či přes bluetooth. Samotné rozhraní, které poskytuje kostka, se skládá z displeje a 6-ti ovládacích tlačítek. Uživatelské rozhraní zobrazuje v horní liště sílu signálu, jméno kostky a stav baterie. Zbytek je tvořen oknem se 4 záložkami. První záložka zobrazuje seznam nedávno spuštěných programů, což umožňuje jejich rychlé opětovné zapnutí. Druhá záložka slouží k vyhledávání programů uložených v adresářové struktuře robota. Další záložku tvoří programy, které jsou standardně uloženy v kostce. Jeden z nich je program umožňující vytváření kódu pro ovládání robota. Poslední záložkou je menu pro nastavení robota, kde se nastavuje způsob připojení, síla reproduktorů atd.

Implementace Lega Mindstorms byla použita jako inspirace při vytváření této práce. Je zde ale několik rozdílů. Je třeba si uvědomit, že jsou zde různé způsoby ovládání: lego používá tlačítka, zatímco náš robot bude ovládán přes dotykovou obrazovku, což sebou nese nutnost brát v úvahu dostatečnou velikost prvků, kvůli pohodlnému užívání. Dalším

[1] Převzato z



Obrázek 3.1: EV3 kostka
[1]

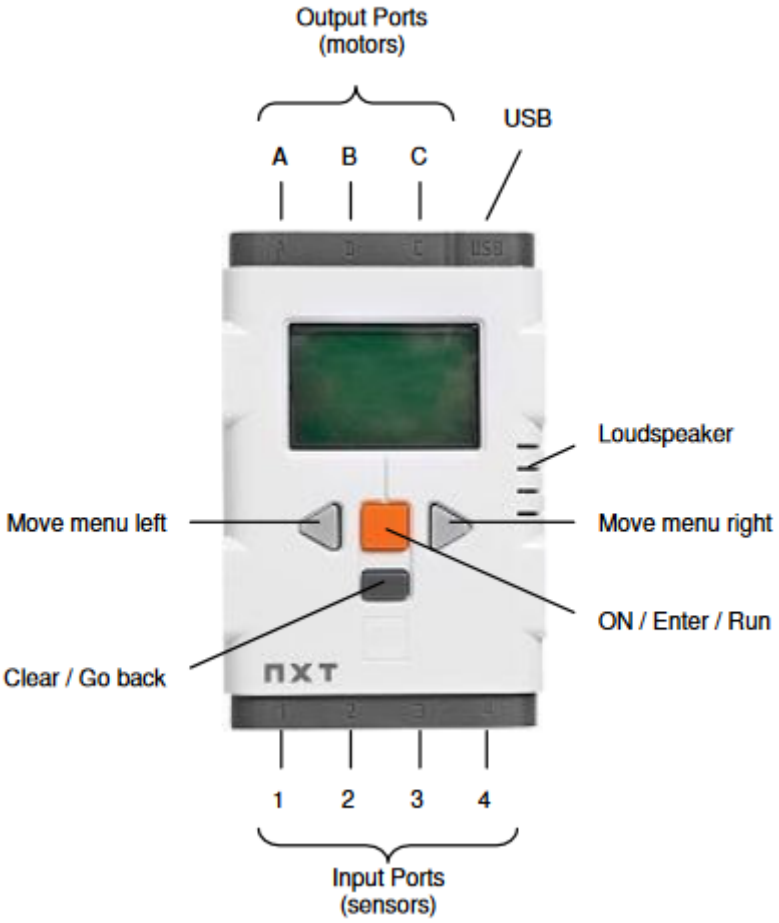
výrazným rozdílem je možnost procházení souborů pro vyhledávání programů. V mém případě jsem zvolil možnost, kdy rozhraní zobrazí programy, které budou uloženy v určitých adresářích robota, a tím pádem umístění souborů zůstane uživateli skryto. Tím se usnadní ovládání robota bez nutnosti prohledávat složky a zároveň se zvýší bezpečnost robota například tím, že se znemožní uživateli přístup ke všem uloženým programům.

3.1.2 Lego Mindstorms NXT

Jedná se o verzi, která předcházela modelu EV3, a podobně jako ve verzi EV3 je hlavní částí inteligentní kostka. Vzhledem se obě dvě verze liší jen minimálně - hlavní rozdíl je umístění tlačítka zpět pod tlačítko enter a absence šipek pro navigaci nahoru a dolů. NXT má menší počet výstupních portů, které jsou 3, a tudíž je možno připojit menší počet motorů. Co se týče senzorů a typů motorů, není mezi těmito verzemi žádný rozdíl ve funkčnosti, pouze v jejich vzhledu a velikosti. Další nevýhodou této starší verze je nemožnost spojení více kostek tak, aby se docílilo většího výkonu. Uživatelské rozhraní má stejnou horní lištu jako EV3, ale dál se už liší v možnostech záložek, které poskytuje:

- Moje soubory — zde se nalézají NXT programy, programy nahrané z počítače a zvukové soubory.
- NXT program – v této záložce se dají vytvářet programy přímo na kostce.
- Náhled – slouží k testování připojených senzorů nebo motorů. Uživatel si vybere, který port chce testovat a jaké je připojené zařízení, a jsou mu zobrazeny naměřené hodnoty.
- Bluetooth – pro nastavení bluetooth a vyhledání dostupných zařízení.
- Nastavení – zde si může uživatel nastavit uspání kostky, hlasitost a také má možnost mazat soubory nahranné na kostce.

- Vyzkoušej mě – slouží k testování jednotlivých částí programu. Dají se zde sledovat hodnoty senzorů a motorů v jednotlivých částech kódu.



Obrázek 3.2: NXT kostka

3.2 Návrh uživatelského rozhraní

Pro maximální využití robota více uživateli jsou potřeba dva programy: jeden, který se stará přímo o rozhraní mezi uživatelem a robotem (tedy pro jeho ovládání), a druhý pro správu programů v robotovi.

3.2.1 Odroid xu4

Odroid xu4 [3] je miniaturní počítač velikosti kreditní karty, v našem případě použit jako hlavní mozek robota. Je tedy i zařízením, na kterém poběží aplikace pro ovládání robota. Tento počítač je osazen 8 jádrovým ARM[3] procesorem Exynos 5422, Mali GPU[4] a Gigabitovou Ethernetovou kartou. To z něj dělá výkonné multifunkční zařízení vhodné jak pro domácí užití, tak pro vývoj aplikací. Jako operační systém byla použita upravená verze Ubuntu 16.04[2], skupinou hardkernel.

[2]https://wiki.odroid.com/odroid-xu4/os_images/linux/ubuntu/ubuntu [3]Advanced RISC Machines [4]Graphics Processing Unit

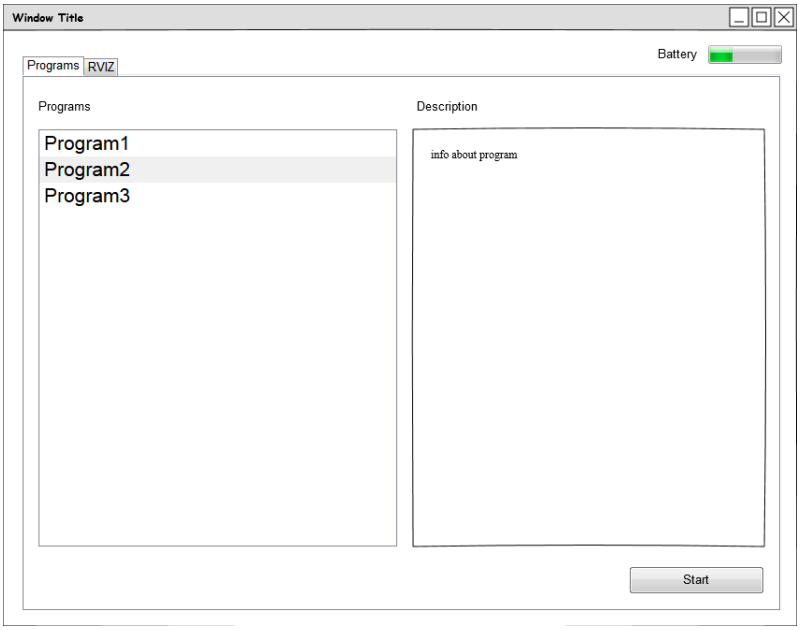
3.2.2 GUI pro ovládání robota

Při návrhu této části je potřeba brát v úvahu zobrazovací možnosti displeje umístěného na zadní části robota. Jelikož se jedná o dotykový displej, je nutné přizpůsobit velikost tlačítek pro snadné ovládání a zobrazovat jen potřebné informace tak, aby nedošlo k dezorientaci uživatele.

Po zapnutí robota se automaticky zobrazí program pro jeho ovládání (3.3) se dvěma záložkami. Program je zobrazen přes celou plochu a pokud chce uživatel změnit velikost okna nebo program vypnout, zobrazí se okno, které požaduje zadání administrátorského hesla a to z důvodu, že běžný uživatel by neměl mít přístup jinam, než k ovládání robota.

První záložka zobrazuje nabídku programů v levém podokně. Jedná se o seznam programů, které jsou uloženy v robotu a jsou připraveny ke spuštění. Po kliknutí na program se v pravém podokně zobrazí informace, které jsou uloženy v xml souboru pod tagem „description“. Pokud nebude dostupný žádný popis programu, bude zobrazena hláška informující o absenci popisu. Při vybrání jednoho z programů a kliknutí na tlačítko „Start“ se spustí vybraný program. To se projeví otevřením nové záložky, ve které bude zobrazeno ROS, rozhraní vytvořené pro obsluhu spuštěného programu. Je možné mít spuštěný jenom jeden program, takže pokud se uživatel pokusí spustit další, zobrazí se mu okno informující o této skutečnosti s možností ukončit právě běžící program, nebo pokračovat v provádění běžícího programu.

V druhé záložce je ROS balíček rviz, který slouží k 3D zobrazení robota v prostoru a jeho okolí, které nasnímal svými senzory. Mimo jiné tato záložka obsahuje údaje o robotu jako jsou rychlost, vzdálenost od překážky a směr pohybu. Pokud dosáhne některý z těchto parametrů nebezpečných hodnot, budou tyto zobrazeny červeně.

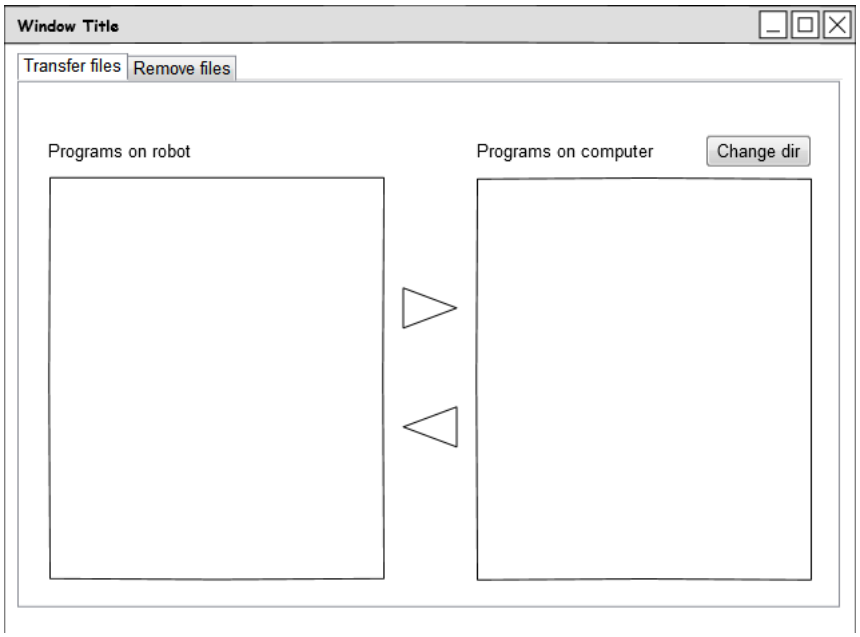


Obrázek 3.3: Hlavní okno programu

Záložka pro spuštěný program bude obsahovat jméno programu a jeho rozhraní, které bylo vytvořeno v ROSu. Pokud program nemá žádný grafický výstup, bude zde vypsán pouze název běžícího programu. Dalším prvkem této záložky bude tlačítko pro ukončení prováděného programu.

3.2.3 Správa programů

Jedná se o rozhraní, které umožní spravovat programy uložené v robotovi. Po připojení robota pomocí usb kabelu k počítači bude umožněno uživateli přidávat a odebírat programy. Je důležité, aby se toto rozhraní (3.4) dalo spustit z jakéhokoliv počítače nehledě na systém. Program bude rozdělen dvěma záložkami na část pro kopírování programů z/do robota a na odstranění programů z robota. Část pro kopírování programů bude tvořena dvěma poli. Levé bude reprezentovat programy uložené na robotovi a pravé programy na počítači. Jelikož programy na počítači mohou být uloženy v různých adresářích, tak zde bude tlačítko pro změnu adresáře. Mezi těmito poli budou dvě tlačítka ve tvaru trojúhelníků, ukazující jedno doleva a druhé doprava. Trojúhelník ukazující doleva kopíruje programy do robota, druhá šipka z robota. Po zkopírování programů na robota se provede jejich přeložení, výstup z překladu bude uložen ve složce ErrorLog, v souboru „catkin_make_date.txt“. Pokud se uživatel přepne do „remove files“, zobrazí se mu seznam programů na robotovi s možností zatrhnutí těch, které si přeje odstranit. Vybrané soubory se odstraní po zmáčknutí tlačítka „delete“.



Obrázek 3.4: Rozhraní pro přenos programů

Kapitola 4

Implementace

Tato kapitola obsahuje implementační detaily aplikací navržených v kapitole 3. Obě aplikace jsou implementované v jazyce C++, s použitím frameworku Qt, verze knihovny ROS je Kinetic. Aplikace nebyly vyvíjeny přímo na odroidu, ale na notebooku s OS Ubuntu 16.04 a to z důvodu jednodušší manipulace, než s odroidem. Názvy vytvořených aplikací jsou robot_gui a file_transfer. U obou aplikací došlo ke změně oproti návrhu a to kvůli zvýšení praktičnosti a v některých případech nereálnosti návrhu.

4.1 Robot_gui

Při spuštění aplikace se ověří, jestli běží instance roscore - jedná se o kolekci uzlů a programů, které jsou potřeba k běhu ROSu. Pokud roscore neběží, pak je program ukončen, pokud běží, je iniciován uzel, který má na starost poslouchat ros topics pro získávání dat ze senzorů. Program pokračuje dál - je vytvořena a inicializována instance třídy files, která má na starost zjistit programy, které je možné použít ke spuštění, a uchovat si potřebná data, aby mohly být později spuštěny.

Jak je vidět z obrázku (4.1), byl změněn počet záložek, byla přidána záložka Stats, do které bylo přesunuto zobrazení informací ze senzorů a stav baterky. K tomuto přesunu došlo, protože display, který je použitý k zobrazování, je menších rozměrů, a tak bude v záložce RVIZ potřeba veškeré místo k zobrazení mapy vytvářené RVIZem. Další velkou změnou je odstranění možnosti vypnutí programu. Běžný uživatel, který se přihlásí na účet odroid, má jedinou možnost interakce s robotem, a to pomocí grafického rozhraní. Toho bylo docíleno odebráním tlačítek na ukončení, minimalizování a změnění velikosti okna programu. Program je automaticky zapnut po přihlášení a to pomocí skriptu *startup.sh*, kdy tento skript zároveň pustí roscore. Jelikož uživatel nemá možnost vypnout program, tak pro vypnutí robota slouží tlačítko *turn off*.

4.1.1 Výběr programů

Jedná se o hlavní záložku, kde jsou zobrazeny všechny programy, které byly nalezeny v adresáři catkin_ws_src. Podmínkou pro zobrazení programu je, že složka obsahuje soubor package.xml, z kterého jsou načteny informace o jménu autora, název a popis programu. O zpracování těchto dat se stará třída files. Pro získání dat ze souboru jsem použil qt třídu QxmlStreamReader. Jedná se o rychlý parser pro čtení XML. Tento parser si bere jako vstup QIODevice – vstupní a výstupní zařízení, což jsou QFile, QBuffer, QTcpSocket (pro

naše použití se hodí třída QFile). Samotný parser funguje jako cyklus, který postupně čte jednotlivé elementy pomocí funkce readNext() až do konce souboru.

Pokud byl robot připojen k počítači a byl na něj pomocí programu file_transfer nahrán nový balíček, nebo byl odstraněn nějaký ze současných, je potřeba k zobrazení aktuálního seznamu balíčku provést znovu načtení souborů. Toho se docílí dvěma způsoby: vypnutím a zapnutím robota, nebo stisknutím tlačítka Refresh.

4.1.2 Spuštění programů

Po vybrání programu a stlačení tlačítka Start bude program spuštěn. Pro spuštění jsou dvě možnosti: pomocí příkazu „rosrun package_name package_name“ nebo pomocí launch souboru. První možnost je velice limitovaná - pouští jenom spustitelný soubor, který se jmenuje stejně jako balíček, i když v konkrétním balíčku může být více spustitelných souborů. Proto můj program preferuje startování pomocí příkazu „roslaunch package_name file.launch“, kde autor balíčku rozhodl, které uzly a v jakém pořadí spustit.

Spuštění programu je pomocí třídy QProcess, která umožňuje pouštět externí programy a následně s nimi komunikovat. Pomocí ní se spustí externí bash, kterému se předají předtím zmiňované příkazy. Aby byla dodržena přehlednost rozhraní, tak pokud má spuštěná aplikace nějaké grafické rozhraní, pak bude umístěno do záložky programs. Ta obsahuje QX11EmbedContainer, což je grafický kontejner pro zobrazení externích oken. Aby se docílilo zanoření nově otevřeného okna do kontejnerů, je potřeba aplikaci předat ID kontejnerů nebo získat ID nově otevřeného okna a nechat kontejner okno pohltit.

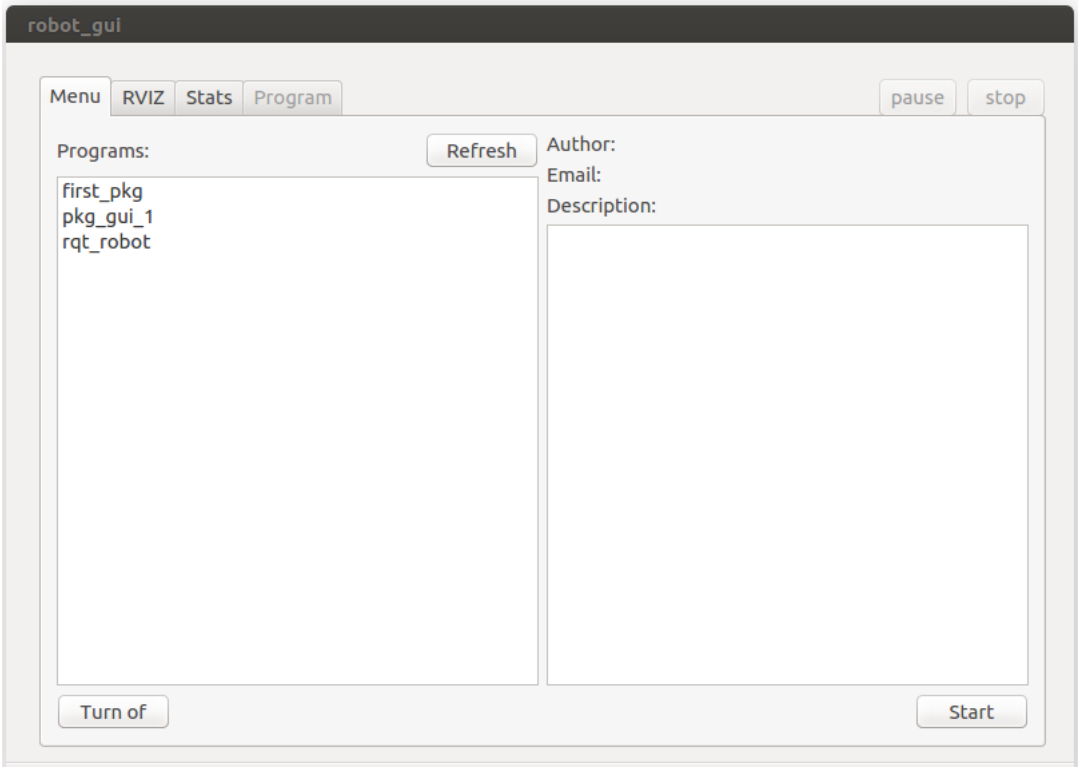
Z těchto dvou možností jsem si vybral tu druhou a to ze dvou důvodů. Současné aplikace s tímto zanořením nepočítají, tudíž jim není možné předat ID kontejneru. Druhým důvodem je, že před spuštěním programu není jasné, kolik oken bude otevřeno, tudíž je není možné všechny zanořit. Ovšem druhý způsob má také své nevýhody - musíme zjistit ID otevřeného okna, které získáme pomocí linuxového příkazu *wmctrl*, ten zobrazí seznam všech oken, která jsou k zobrazení. Následně vyfiltrujeme naše okno pomocí ID procesu. Jenže spuštění pomocí roslaunch může spustit několik programů a nad nimi nemáme žádnou kontrolu. Takže si pomocí linuxového příkazu *ps* vyfiltrujeme ID všech potomků příkazu roslaunch a u nich zkontrolujeme, zda nemají nějaké okno.

Pozastavení a zastavení spuštěných programů je docíleno pomocí vyslání signálu SIGSTOP a SIGTERM. Po pozastavení je možné program opět spustit, pomocí signálu SIGCONT, ale v tomto případě není zaručeno chování znovu spuštěné aplikace. Pokud s tím pozastavená aplikace nepočítá, může dojít k nečekaným stavům a je možné, že bude muset být ukončena.

4.1.3 Rviz

Pro připojení rvizu do mého programu jsem potřeboval tyto tři třídy: rviz/visualization_manager, *rviz/render_panel, rviz/display*. Instance první třídy je hlavním správcem, udržuje si všechny displeje, nástroje a ovladač zobrazení. Stará se o to, aby v pravidelných intervalech byla volána funkce *update()* a tím každý display zobrazoval aktuální data. Druhá třída má na starost zobrazovat prostředí rendrované pomocí OGRE(Object-Oriented Graphics Rendering Engine). Taky se stará o předávání kliknutí myší do zobrazované scény. Poslední třída *rviz/display* se stará o jednotlivé displeje, které jsou přidávány do scény, přihlašuje je k odběru topicu, na který jsou posílány data ze senzorů a které se mají zobrazit.

Pro zobrazení průběhu mapovaní je potřeba vytvořit několik displejů tak, aby byla scéna přehledná. Základním displejem je 2D mřížka souběžná s rovinou, která slouží k



Obrázek 4.1: Ukázka GUI po spuštění

lepší orientaci ve scéně. Dalším zobrazovaným prvkem je model robota. Pokud není žádný definovaný, bude zobrazen černý blok. O zobrazení průběhu skenování okolí se stará displej *LaserScan*. V celkové scéně budou zobrazeny body, které reflektují objekt, od kterého se laser odrazil. Tyto data si displej bere z topicu */scan*. Laser není jediným senzorem robota, kterým se získává informace o okolí. Robot je vybaven sonary, jejichž aktivita se dá též zobrazit a to pomocí displeje typu *Range*. Vysílání zvukových vln je v RVIZu zobrazeno jako poměrně velký kužel. Rozhodl jsem se zobrazovat pouze jeden sonar i přes to, žerobot jich má 6. Kód pro ostatní sonary je připravený, jen je potřeba nastavit jejich zobrazení. K zobrazení mapy je potřeba sada displejů. Mapa je vytvářena tím, jak se robot pohybuje a získává informace o svém okolí pomocí senzorů. Při mapování je vytvářena globální mapa a lokální mapa nebo-li to, co je v blízkosti robota. Topic pro globální mapu je */move_base/global_costmap/costmap*, pro lokální to je */move_base/local_costmap/astma*. Nedílnou součástí mapování je plánování cesty. Pro zobrazení naplánované cesty jsou potřeba dva displeje typu Path. Jeden displej zobrazuje celkový plán cesty a druhý zobrazuje krátký plán cesty, která bude následně provedena. Cesty odebírají tyto topics */move_base/global_plan/plan*, */move_base/local_plan/plan*. Poslední věcí k zobrazení je pozice robota. Vždy je potřeba zobrazit dvě pozice - tu, do které se má dostat, a odhad aktuální pozice. Aktuální pozice robota není přesná, nýbrž se jedná o pole možných bodů, kde by se robot mohl nacházet. Tudíž je na zobrazení aktuální pozice a cílové pozice potřeba dvou různých typů displejů: *PoseArray* a *Pose*. Uvedené displeje poslouchají tyto topics: *move_base/current_pozizition a move_base/current_goal/pose*.

4.1.4 Zobrazení dat ze senzorů

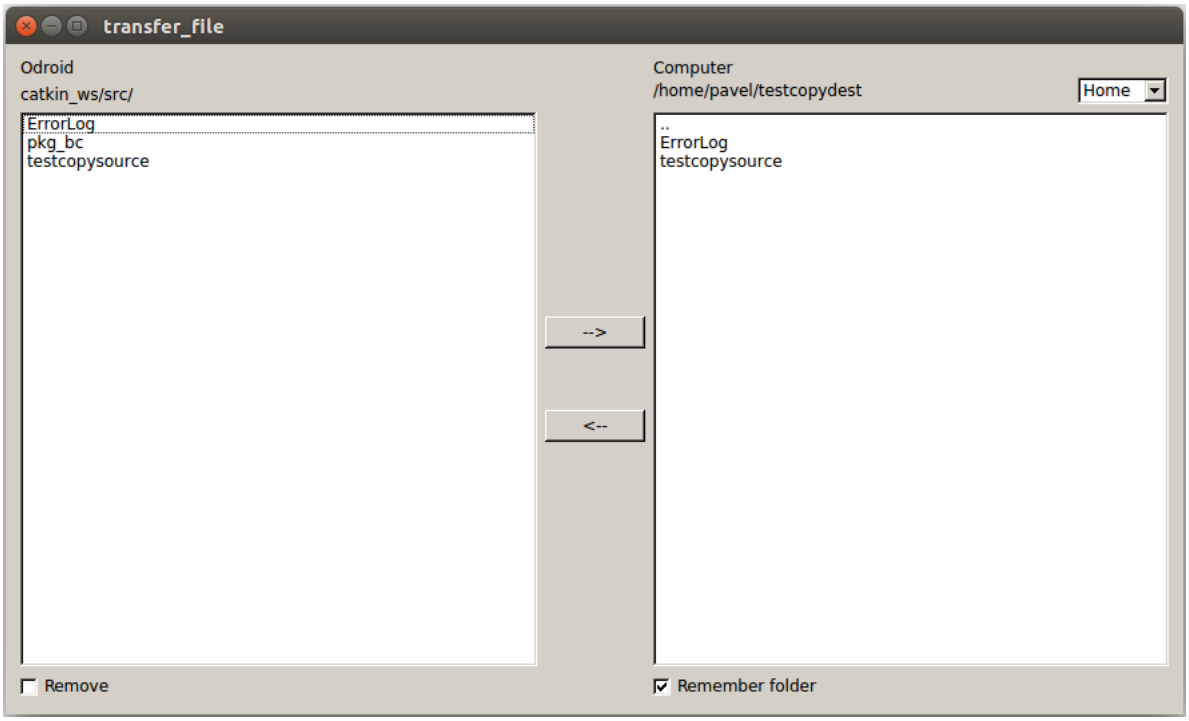
V této záložce jsou zobrazeny všechny informace ze senzorů, které se dají zobrazit jednoduchou číselnou hodnotou nebo případně jednoduchým grafickým zobrazením, jako je šipka. O přijímání dat se stará ros uzel, který poslouchá na jednotlivých tématech. Pokud přijdou nějaká data, tak jsou nejprve zpracována a potom pomocí qt signálů odeslána do hlavní části programu, odkud jsou již zobrazena. Pro příjímání dat ze sonarů a encodérů jsem si vytvořil vlastní zprávy – *msg_sonars* a *msg_encoders*. Obsahem zpráv je pole typu *sensor_msgs/Range*, respektive *geometry_msgs/Twist*. Vlastní zprávy bylo potřeba zakomponovat, protože základní ros zprávy nelze předávat jako pole, a tak by bylo pro příjímání dat ze sonarů a encodérů potřeba přihlásit odběr pro každý sonar, respektive encodér, zvlášť. Toto není ideální, protože by vznikala zbytečně velká komunikace, a mohlo by dojít k přílišnému zatěžování zařízení. Pro příjímání dat z akcelerometru a gyroskopu je použit typ zpráv *sensor_msgs/Imu*, kde lineární složka je použita pro akcelerometr a úhlová pro gyroskop. Pro zjišťování stavu baterky jsem zvažoval možnost kontrolování změny obsahu systémového souboru */sys/class/power_supply/BAT0/capacity*, ale protože není záruka, že název tohoto souboru bude vždy stejný, rozhodl jsem se využít ROS zpráv typu *sensor_msgs/BatteryState*, konkrétněji proměnnou *percentage*. Jednotlivá témata, ze kterých jsou brána data, jsou:*robot/sonars, robot/encoders, robot/imu a robot/battery*.

4.2 File_transfer

Aplikace je multiplatformní, tedy je plně funkční na Windows i Linux. Nezkoušel jsem implementovat řešení pro macOS z důvodu absence možností testování implementace. Při návrhu aplikace jsem plánoval propojení mezi odroidem a počítačem pomocí usb kabelů. V průběhu implementace jsem zjistil, že to pravděpodobně nebude možné, protože usb protokol je stavěný na komunikaci mezi zařízením, které se chová jako master, a zařízením v roli slave. Ovšem odroid i počítač jsou zařízení, chovající se jako master. Tento problém jsem chtěl vyřešit pomocí OTG, ale ukázalo se, že to z hardwarového hlediska není možné[5]. Proto pro propojení počítače a odroidu byl použit křížený kabel RJ45 a ke komunikaci byla využita knihovna libssh. Jedná se o multiplatformní knihovnu implementující SSH protokol. V průběhu testování jsem zjistil, že je nepraktické pro mazání souborů přepínat do jiné záložky, a tak oproti původnímu návrhu, byla tato funkce zahrnuta do hlavního okna, kdy se uživatel přepne do módu mazání pomocí zaškrtávacího políčka *Remove* (viz 4.2).

4.2.1 Spuštění programu

Program je potřeba pouštět s oprávněním správce, respektive jako root uživatel. Správcovo oprávnění je třeba při běhu programu, protože před navázáním spojení s odroidem je potřeba přenastavit IP adresu počítače. Adresa se mění pro ethernetový port na adresu 192.168.1.3 tak, aby byla ve stejném rozsahu jako odroid. Na systému typu Linux je potřeba zjistit název ethernetového portu. Název se určuje podle skutečného hardwaru v zařízení. Tento název se získá pomocí linuxového příkazu *lshw*. *Lshw* je nástroj pro zobrazování detailních informací o nastavení hardwaru. Při ukončování programu je ethernetový port nastaven zpět na dhcp. Po změnění IP adresy dojde k otevření spojení mezi zařízeními pomocí funkce *ssh_connect* z knihovny libssh. *Ssh_connect* zapouzdřuje protokol Secure Shell (ssh). Po úspěšném propojení je potřeba otevřít kanál pro přenos dat, čehož se docílí další knihovní funkcí *sftp_new*, která používá protokol SFTP-Secure File Transfer Protocol.



Obrázek 4.2: Ukázka programu File_transfer

4.2.2 Registrace připojení externích zařízení

Pro umožnění kopírování dat z/na externí zařízení, jako jsou flash disky a externí disky, je potřeba zachytit připojení nebo odpojení v reálném čase. Na každém operačním systému se připojení těchto zařízení projeví jinak. Na systému Windows[4] je potřeba přihlásit se k odběru stavu zařízení pomocí funkce *RegistrDeviceNotification* z Windows knihovny *Windows.h*. Funkci je potřeba předat identifikátor naší aplikace a typ zařízení, které bude monitorováno. Následujícím krokem je přetížení funkce *nativeEvent*, kde se vyfiltrují jenom zprávy o připojení a odpojení zařízení. Na Linuxu se sleduje změna stavu složky *media/user_name*, do které se registrují všechna připojená externí zařízení. Pro sledování složky je použita třída *QFileSystemWatcher*. Při inicializování je této třídě předána cesta k souboru nebo složce, která má být monitorována. V případě změny monitorované složky je vyvolán signál *direcotoryChanged*, kdy se obsluhou toho signálu monitoruje připojení a odpojení zařízení.

4.2.3 Odeslání programů

Pro přenos souborů byl využit sftp kanál, kdy k otevření kanálu dojde při spuštění programu (viz 4.2.1). Ovšem knihovna libssh neposkytuje žádnou funkci pro kopírování složek, a tedy každý soubor musí být zkopírován samostatně a musí být vytvořen adresář/podadresář. To znamená, že na straně počítače se musí přesouvaná složka postupně procházet - pokud se jedná o soubor, tak je nutné ho otevřít a zkopírovat, pokud o adresář, tak je na odroidu potřeba vytvořit stejnou složku, na počítači ji otevřít a projít její obsah. Vzhledem k povaze tohoto problému, jsem řešil kopírování rekurzivně, tedy pokud je právě kopírovaná položka složka, dojde k zanoření. Samotné kopírování probíhá následovně: soubor je otevřen do vstupního streamu, potom je přečtený blok dat, který se odešle pomocí funkce

sftp_write a takto se pokračuje až do konce souboru. Posílané bloky jsou o velikosti 16 kB. Po překopírování zvolených programů, je potřeba provést překlad programů pomocí příkazu *catkin_make* tak, aby bylo možné nově nahrané programy na robotovi spustit. Výsledek překladu kódu je uložen do složky ErrorLog a to do souboru catkin_make_errors_date.txt.

4.2.4 Stahovaní a mazaní programů

Při stahování i mazání programů se jedná o obdobný proces jako u odesílání programů, jen je trošku náročnější z hlediska mapování struktury kopírované složky. Knihovna libssh nemá funkci pro zjištění struktury složky, proto je při stahování potřeba otevřít složku pomocí funkce *sftp_opendir*, která vrací referenci na otevřenou složku. S touto referencí se volá funkce *sftp_readdir*, která vrací vždy jednu položku (soubor nebo adresář) z otevřené složky, nebo konec složky. Pokud je vracenou položkou soubor, dojde k odesílání dat na počítač, pokud složka, tak dojde k zanoření. Stejně funguje i mazání programů, jen s tím rozdílem, že soubor se nikam neodesílá, ale je odstraněn. Při kopírování programů na počítač s OS Linux, dochází při vytváření souborů a složek ke komplikaci. Protože byl program spuštěn s oprávněním root, tak jsou práva k těmto souborům příděleny uživateli root. Je tedy potřeba změnit vlastníka těchto dat z root na uživatele, který tento program ve skutečnosti pustil. To se provede pomocí linuxového příkazu *chown*, kdy za použití přepínače -R se rekurzivně změní vlastník složky i jejího veškerého obsahu.

Kapitola 5

Testování

V poslední kapitole této práce se budu věnovat testování vytvořených aplikací za účelem vyhodnocení kvality vytvořené práce. Aplikace byly testovány jednak mnou, tak uživateli. Mnou prováděné testy byly zaměřeny na ověření funkčnosti, tedy jestli aplikace robot_gui funguje na cílovém zařízení – odroid xu4. A zda aplikace file_transfer funguje jednak na Windows, tak i na Linuxových systémech. Uživatelské testování bylo zaměřeno na pochopení a schopnost ovládat aplikace.

Vzhledem k tomu, že jsem neměl k dispozici skutečného robota, který by mi poskytoval data ze senzorů, musel jsem vytvořit náhodná data, která by mohla být naměřena senzory tak, aby uživatelé měli co nejblíže k užívání skutečného robota. Data pro simulaci mapování není tak jednoduché vytvořit, proto jsem pro účely testování použil ros tutoriál Husky Frontier Exploration Demo[3]. V tomto tutoriálu se pouští gazebo simulace robota, rviz a program pro navigaci při prozkoumávání okolí. Pro zobrazení těchto dat jsem přepsal topics, na kterých poslouchá má aplikace tak, aby zobrazovala data, která by byla jinak zobrazována pomocí rvizu v tomto tutoriálu.

5.1 Testování funkčnosti

K testování obou aplikací jsem si připravil testovací sadu, kdy jsem u jednotlivých testů stanovil cíl a testovaná data, a poté jsem vyhodnocoval, zda došlo k předpokládanému výsledku nebo došlo k nějaké chybě. Testoval jsem jak běžnou činnost, tak stavy, které by se neměly normálně vyskytovat a záměrně špatné hodnoty. Aplikaci robot_gui jsem testoval, na odroidu xu4, OS Ubuntu mate 16.04, s připojenou dotykovou obrazovkou (HDMI 5"800x480 Display Backapack). Výsledky jsem porovnával s chováním na notebooku (msi gp62 6qf leopard pro) s OS Ubuntu 16.04. File_transfer byl testován na stejném notebooku, ale na systémech Windows 10 a Ubuntu 16.04. Nebudu zde zmiňovat všechny testy, jen ty, které se mi zdály zajímavé, a ty, které mi pomohly odhalit nějaké chyby.

Ukázalo se, že nebude možné zobrazit rviz na odroidu a to z důvodu hardwarové limitace. Odroid je postaven na ARM procesoru a tudíž nepodporuje OpenGl, ale používá OpenGL ES. K běhu rvizu je potřeba OpenGl i OpenGL ES, takže není možné rviz spustit. Existují ovladače pro překlad OpenGl na OpenGL ES, ale ani to nebude fungovat, protože rviz používá oboje a docházelo by k chybám. Proto není možné zobrazit rviz na odroidu, řešením by mohlo být vyžití jiného zařízení, jako je třeba Raspberry 3, ale ani u něho není zaručeno,

[3] http://wiki.ros.org.husky_navigation/Tutorials/Husky%20Fronier%20Exploration%20Demo

že bude mít dostatečný výkon pro potřebná zobrazení. Dalším řešením by bylo nezobrazovat rviz na robotovi, ale naměřené hodnoty odesílat a zobrazovat na počítači.

Testování otvírání nových oken: cílem testu bylo zjištění chování aplikace ve chvíli, kdy nově otevřené okno je výrazně větší, než kontejner, ve kterém má být zobrazeno. Jak jsem očekával, bylo zobrazeno to, co se vešlo do kontejneru, a zbytek zobrazovaného okna byl za hranicemi zobrazení a nebyl k němu žádný přístup. To platilo třeba pro zobrazení uzlu turtlesim. Turtlesim je uzel používaný pro výuku komunikace v ROSu - zobrazí želvičku, která představuje robota a může se pohybovat na základě přijatých zpráv. Při zobrazení tohoto uzlu v kontejneru došlo k přetečení plochy, po které se želvička může pohybovat a tak se mohla dostat mimo zobrazovanou plochu a tudíž uživatel přišel o možnost sledování jejího pohybu. Ovšem u aplikací napsaných v Qt došlo ke zmenšení zobrazovaného obsahu tak, aby se vše vešlo do kontejneru. Proto je potřeba, aby vytvářené programy počítaly s velikostí kontejneru, v kterém budou zobrazovány, a to, 300x300 px. Pokud přesáhnou tuto velikost, není definováno, jak se zobrazovaný program zachová.

Při situaci, kdy byl puštěn některý z programů, který má obsluhovat robota a došlo k vypnutí/pádu aplikace (tedy robot_gui), by mělo dojít k ukončení spuštěného programu, protože dojde ke zničení Qt třídy QProcces, která se stará o spuštěný program. To se do jisté míry potvrdilo testem, kdy jsem pomocí příkazu kill vypnul aplikaci robot_gui. Zrovna běžící program byl ukončen, to ale platí jen u programů, které byly spuštěny pomocí rosrun, tedy má nad nimi kontrolu třída QProcess. Pokud byl program puštěn pomocí roslaunch, tak třída QProcess nemá žádnou přímou kontrolu nad spuštěnými programy, jelikož se jedná o potomky příkazu roslaunch. Po pádu aplikace spuštěný program pokračoval v normálním chodu, pokud měl nějaké gui, které bylo zobrazováno uvnitř kontejnérů, došlo k znovuotevření tohoto gui na ploše robota. Myslím, že druhá varianta, kdy program pokračuje i po pádu mé aplikace, je lepší, protože dojde k dokončení činnosti robota a tím pádem nedojde k nečekaným stavům robota.

Testování nastavení spojení mezi odroidem a notebookem (systém Ubuntu 16.04), kdy je zároveň zapnutá wi-fi: Očekával jsem, že aplikace (file_transfer) by měla nastavit IP adresu na statickou a navázat spojení. Při spuštění docházelo k nastavení správné adresy, ale v průběhu spojení občas docházelo k výpadku spojení, neboť počítač se nemohl rozhodnout, který druh spojení používat. Dospěl jsem proto k závěru, že je lepší před spuštěním aplikace, vypnout wi-fi a tím zabránit případným výpadkům spojení a pádu aplikace.

Kopírování programů z notebooku (systém Windows 10) na Odroid, kdy jsem testoval dvě situace: program na Odroidu už byl, a šlo jen o přepsání starých souborů novými. Program na Odroidu nebyl, a bylo potřeba vytvořit nové soubory. Pokud program na Odroidu byl, proběhlo přepsání starých souborů bez problému, ale při kopírování nových programů došlo k chybě u vytváření souborů. Zdá se, že byl problém s právy k vytváření souborů. Tento problém jsem obešel použitím příkazu touch, kdy jsem pomocí něho vytvořil prázdný soubor a ten se potom dal normálně otevřít.

5.2 Uživatelské testování

Pro dosažení co nejrozmanitější skupiny testujících, jsem se rozhodl testovat jak na zástupcích skupiny zběhlé ve využívání počítačů i vyvíjení aplikací, tak na zástupcích skupiny laičtější. Zástupce první skupiny jsem našel jednak mezi svými spolužáky z VUT FIT, tak mezi kamarády studujícími na FI MU. Za zástupce druhé skupiny jsem použil svoji rodinu. Uživatelům bylo nejdříve vysvětleno, co bylo cílem práce a k čemu by měly sloužit vytvo-

řené aplikace. Po té měli příležitost si aplikaci vyzkoušet, sami si ji projít a po nějakém čase jsem jim zadal několik úkolů, které mají pomocí aplikací provést.

Vyhodnocování probíhalo jednak pomocí přímého pozorování, kdy jsem sledoval, jak uživatelé zacházejí s aplikací, s jakou přesností jsou schopni ji ovládat, a jak dlouho jim provedení úkolu zabere. Druhou částí byl dotazník, který uživatelé dostali na konci testování. Dotazník využíval převážně metody zvané Likertova škála[4]. U této metody respondent odpovídá na sadu výroků pomocí čísla na škále 1 – 10, přičemž je vždy uveden význam krajních hodnot.

5.2.1 Robot_GUI

Při testování dostali uživatelé vždy dva úkoly. Prvním úkolem bylo spustit program, který byl vytvořen pro testování a jehož funkcí bylo simulovat vysílání dat ze senzorů a otevřít dvě okna. První okno obsahovalo uzel turtlesim, želvička za sebou při pohybu zanechává bílou čáru. V druhém okně bylo ovládání pro uzel turtlesim, které sestává ze čtyř tlačítek - dvě pro natočení želvičky o 90 ° doleva nebo doprava a dvě pro pohyb želvičky dopředu a dozadu. Druhým úkolem bylo spustit program, který nahráli při testování programu file_transfer. Úkony, které měli uživatelé vykonat v rámci prvního úkolu:

- Nakreslit se želvičkou čtverec.
- Pozastavit spuštěný program v průběhu pohybu želvičky a opět program spustit.
- Vysvětlit zobrazovaná data, porozumění hodnotám ze senzorů a zobrazení rvizu.
- Vypnout a zapnout testovací program.

Se všemi úkoly si testeři poradili a to i přes nepříjemnost, že ovládání a simulace želvičky byly zobrazeny v jiných záložkách. Při jejich obsluze jsem objevil menší chybu - pokud uživatel pozastavil spuštěný program a potom jej ukončil, bylo tlačítko pro pozastavení/-puštění stále přepnuto do stavu puštění. Pokud byl puštěn nový program, uživateli byla nabízena pouze možnost puštění, a neměl tedy možnost puštěný program pozastavit.

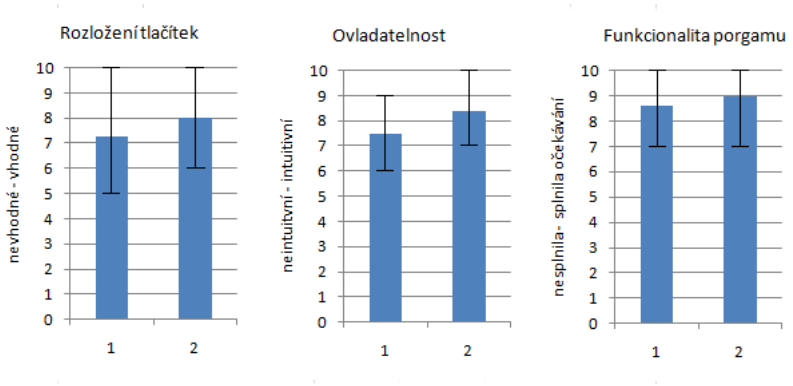
Z vyplněných dotazníků jsem zjistil, že testeři byli spokojeni s rozložením tlačítek i jejich velikostí a vesměs i s jejich významem. Nejvíce byli zmatení z tlačítka turn off, neboť předpokládali, že po jeho použití dojde k vypnutí aplikace robot_gui, ne celého stroje. Proto jsem přidal okno pro potvrzení, zda uživatel doopravdy plánuje vypnout robota. Původně jsem toto potvrzení v aplikaci nechtěl, protože bude zkušené uživatele pouze zdržovat. Hodnocení aplikace se dá vyčíst z grafů, první skupina grafů zobrazuje spokojenost uživatelů s: rozložením tlačítek, ovladatelností a funkcionalitou. Trošku nižší hodnocení u aplikace robot_gui si vysvětluji tím, že byla ovládána pomocí dotykového displeje. Druhá sada grafů ukazuje, jak byli uživatelé spokojení se zobrazením dat a který prvek jim přišel nejvíce matoucí.

5.2.2 File_transfer

U testování tohoto programu dostali uživatelé tyto úkoly:

- Zkopírovat jednu složku s programem.
- Zkopírovat větší počet složek.

[4] https://en.wikipedia.org/wiki/Likert_scale



Obrázek 5.1: Vyhodnocení testů: 1-Robot_gui, 2-File_transfer

- Připojit flash disk, najít na něm požadovanou složku a zkopírovat ji.
- Odstranit programy z odroidu.
- Zkontrolovat stav překladu programu.

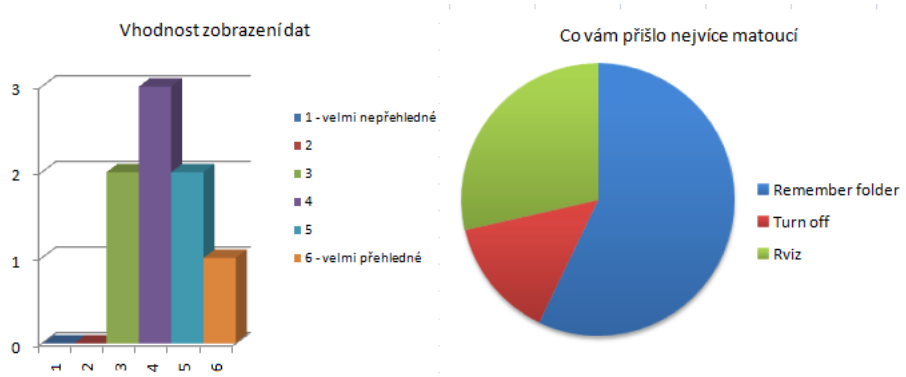
Kopírování programů nedělalo uživatelům žádný problém. V případě pohybu mezi adresáři měli nezkušení uživatelé problémy s pochopením významu položky „..“, která slouží k posunutí se v adresářích o krok zpět. Ale jelikož je moje aplikace cílena na studenty informatiky, nezdá se mi to jako zásadní problém. Většina uživatelů ale měla problém s pochopením významu zatrhávacího pole s popisem „remember folder“. Vymýšleli varianty jako: zapamatuje si složku, ve které byla tato možnost zaškrtnuta, a při odškrtnutí do ní uživatele vrátí, dalším výkladem bylo: pokud bude tato možnost zatrhnuta a přejde se do nějakého adresáře, kde se vyberou složky ke zkopírování, dojde ke zkopírování i zapamatované složky. Nenapadá mě žádný lepší popis pro toto zatrhávací pole, který by nebyl příliš dlouhý, jedinou možností mě napadá přidat nějakou nápovědu ve formě tooltipu, ale nevím, zda by to nebylo ještě víc nepřehledné. Celkové hodnocení aplikace file_transfer uživateli je zobrazeno v grafech.

Při uživatelských testech, na Ubuntu 16.04 došlo k velice zajímavé chybě. Po připojení flash disku k počítači, došlo ke změněně nastavení adresy na ethernetovém portu, ze statické adresy nastavené zapnutím programu, na adresu získanou pomocí dhcp. Tím pádem, bylo přerušeno spojení, mezi počítačem a Odroidem. Tuto chybu se mi už nepovedlo nikdy vyvolat, takže si nejsem jistý, z jakého důvodu nastala, ani zda je to nějaká chyba v mém programu, nebo došlo k nějakému nečekanému stavu v systému.

5.3 Náměty k další práci

Hlavním cílem další práce by mělo být vyřešení problému s RVIZem, na Odroidu se nepodaří rozjet. Ale jsou tu další možnosti již zmíněné Raspberry 3, nebo vzdát se zobrazování přímo na robotovi a celé gui, nebo jen RVIZ přenést na počítač. To by znamenalo připojit robota přes wi-fi k počítači a z toho ho kompletně ovládat, nebo jen zobrazovat RVIZ.

Při testování se ukázalo, že obě aplikace splňují požadavky uživatelů na obsluhu a přehlednost. Ovšem jednalo se o uživatele, kteří s robotem nepracují, a tudíž jejich spokojenost nemusí znamenat, že není co vylepšovat. Určitě se najde spousta funkcí, které by uživatelé pracující s robotem chtěli přidat, nebo pozměnit pro lepší efektivnost. První věcí, která mě



Obrázek 5.2: Graf zpětné vazby uživatelů

napadá, je možnost vypínat a zapínat zobrazované věci v RVIZu. Toto by umožnilo, aby si uživatel mohl vždy zobrazit jen to, co potřebuje, a nebyl zahlcen daty z ostatních senzorů.

V aplikaci pro přenos dat by se dala přidat možnost nastavení IP adresy, jméno uživatele a heslo uživatele, na kterého se mají přenášet programy pro ovládání robota. Aplikace v současném stavu počítá s tím, že IP adresa je 192.168.1.2 a uživatel je odroid, s heslem odroid. Přidáním možnosti nastavit tyto údaje, by se stal program univerzálnějším. Další věcí, která by se dala přidat, je zobrazení průběhu kopírování, ale nevím, zda by to mělo využití, neboť většina dat, která by se měla přenášet, je natolik malá, že dochází k téměř okamžitému přenosu.

Kapitola 6

Závěr

Cílem této práce bylo za použití frameworku ROS a Qt navrhnout a implementovat aplikaci, která dokáže uživateli zprostředkovat možnost spouštění programů pro ovládání školního robota a zobrazovat data z jeho senzorů. Také bylo úkolem, vytvořit podpůrnou aplikaci, která umožní nahrávat programy na robota.

Z důvodu nedostatečné podpory hardwaru, nelze na Odroidu použít RVIZ, ale pokud by byla moje aplikace spuštěna na jiném zařízení, tak je RVIZ připraven k použití. Odroid xu4 by se dal nahradit za Raspberry Pi 3, který by měl být dostatečný pro běh RVIZu.

Implementace obou programů se mi jinak zdařila, a programy splňují zadané požadavky, i když jsem musel upustit od původního plánu spojení Odroidu s počítačem pomocí usb kabelu. Ale nutno podotknout, že náhradní řešení - propojení pomocí rj 45 - je plně funkční. Výhodu mé aplikace vidím též v tom, že úplně odstiňuje uživatele od možnosti ovládání robota jiným způsobem, než přes spuštění nahraných programů, takže se uživatel nemusí zabývat ničím dalším. Stejně tak z uživatelských testů vyplývá, že aplikace byla navržena úspěšně, neboť uživatelé jsou schopni bez větších obtíží aplikaci ovládat.

Pokud by měla být aplikace využívána stejně jako Mindstorms vyráběné firmou Lego, bylo by potřeba provést testování na uživatelích mladšího věku, také by se musel upravit vzhled aplikace, v současném stavu mají aplikace spíše strohý vzhled, který by nezaujal mladší uživatele. Na druhou stranu jsou aplikace cíleny pro studenty a jejich interakci s robotem, kdy je nejdůležitějším požadavkem funkčnost a jasnost ovládání, a strohost vzhledu pak není potřeba řešit.

Literatura

[1] Bill Stanley, C. B. a. d.: *On-The-Go Supplement to the USB 2.0 Specification.* [Online; navštíveno 07.05.2018].
URL http://www.usb.org/developers/onthego/otg1_0.pdf

[2] Group, L.: *Mindstorms Lego.com.* [Online; navštíveno 06.05.2018].
URL https://www.lego.com/cs-cz/mindstorms/downloads

[3] Hardkernel, L.: *ODROID-XU4 Manual. Home Page | ODROID Magazine* . [Online; navštíveno 07.05.2018].
URL https://magazine.odroid.com/odroid-xu4

[4] Microsoft: *Learn to Develop with Microsoft Developer Network | MSDN.* [Online; navštíveno 07.05.2018].
URL https://msdn.microsoft.com/en-us/library/aa363431.aspx

[5] odroid: *XU4 OTG slave, forum.* [Online; navštíveno 07.05.2018].
URL https://forum.odroid.com/viewtopic.php?f=97&t=30781&p=221829#p221829

[6] *About Qt - Qt Wiki.* [Online; navštíveno 07.05.2018].
URL https://wiki.qt.io/About_Qt

[7] *ROS Documentation.* [Online; navštíveno 06.05.2018].
URL http://wiki.ros.org/

[8] *ROS Nodes.* [Online; navštíveno 06.05.2018].
URL http://wiki.ros.org/Nodes

[9] *ROS Rviz Package Summary.* [Online; navštíveno 06.05.2018].
URL http://wiki.ros.org/rviz

[10] libssh Team: *libssh: The Tutorial.* [Online; navštíveno 07.05.2018].
URL http://api.libssh.org/master/libssh_tutorial.html

[11] Zemčík, P.: *Tvorba uživatelských rozhraní.* [Online; navštíveno 06.05.2018].
URL https://wis.fit.vutbr.cz/FIT/st/course-files-st.php?file=%2Fcourse%2FITU-IT%2Ftexts%2FlTU-Podpora.pdf&cid=11486

Příloha A

Obsah CD

Přiložené CD obsahuje:

- Zdrojové kódy k oběma aplikacím
- Manuál pro použití aplikací
- Skript pro automatické zapnutí aplikace po přihlášení uživatele
- Adresář, který obsahuje zdrojové kódy technické zprávy
- Technickou zprávu ve formátu PDF