VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DIAGNOSTIKA MOBILNÍHO ROBOTU

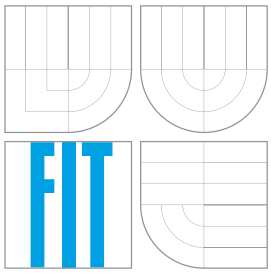
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VLADIMÍR BENČEK

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DIAGNOSTIKA MOBILNÍHO ROBOTU

DIAGNOSTICS OF A MOBILE ROBOT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VLADIMÍR BENČEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ZDENĚK MATERNA

BRNO 2013

Abstrakt

Včasná diagnostika a predpovede budúcich stavov robotických systémov môžu minimalizovať náklady na ich údržbu. Táto práca sa venuje diagnostike a monitoringu mobilného robota TB2. Zahŕňa zoznámenie sa s robotom, s jeho operačným systémom a popisuje návrh a implementáciu diagnostického balíčka, ktorý analyzuje jednotlivé časti robota a informácie o nich poskytuje užívateľovi.

Abstract

Early diagnostics and prediction of future conditions of robotic systems can minimize the costs for their maintenance. This thesis is focused on diagnostics and monitoring of monile robot TB2. It includes familiarisation with the robot, its operating system and describes the design and implementation of the diagnostic package, that analyses different robot parts and provides the information about them to the user.

Klíčová slova

Robot, robotický systém, diagnostika, monitoring, Robot monitor, TB2

Keywords

Robot, robotic system, diagnostics, monitoring, Robot monitor, TB2

Citace

Vladimír Benček: Diagnostika mobilního robotu, bakalářská práce, Brno, FIT VUT v Brně, 2013

Diagnostika mobilního robotu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Zdeňka Materny

.....
Vladimír Benček
15. května 2013

Poděkování

Chtěl bych se poděkovat vedoucímu, panu Ing. Zdeňku Maternovi, za jeho dohled a trpělivost při řešení úlohy.

Obsah

Kapitola 1

Úvod

Robotické systémy sa v súčasnej dobe používajú prakticky vo všetkých odvetviach ľudskej činnosti – či už je to v domácnosti, priemysle, medicíne alebo vo vojenskom sektore. Sú vystavované čoraz väčším nárokom a očakávajú sa presnejšie a spoľahlivejšie výsledky. So stúpajúcou zložitosťou však priamo úmerne rastie aj ich cena.

Včasná diagnostika, odhalenie začínajúcich porúch a predpovede budúcich stavov robotických ale aj iných systémov môžu znížiť, respektíve minimalizovať náklady na ich údržbu. Z tohoto dôvodu je dôležité mať kvalitný diagnostický systém.

Účelom našej práce je navrhnúť a implementovať diagnostický balíček pre mobilného robota, ktorý bude poskytovať základné stavové informácie o jeho jednotlivých častiach.

Na začiatku si v kapitole 2 vysvetlíme terminológiu týkajúcu sa diagnostiky a monitoringu robotických systémov, následne sa v kapitole 3 zoznámime s robotom TB2, využívaným na UPGM FIT VUT v Brne, a Robotickým Operačným Systémom, ktorý tento robot používa. Návrhu implementácie venujeme kapitolu 4, kde si priblížíme predstavu výslednej aplikácie, hlavne z pohľadu jej funkcionality. V kapitole 5 opíšeme vzhľad a fungovanie implementovanej aplikácie, vychádzajúcej z návrhu. Taktiež tu zhrnieme rozdiely medzi implementáciou a pôvodným návrhom. Ďalšie možnosti, rozšírenia a modifikácie našej aplikácie nájdeme v kapitole 6. Posledná kapitola je venovaná zhrnutiu a zhodnoteniu dosiahnutých výsledkov, ako aj niektorým návrhom pokračovania práce.

Kapitola 2

Diagnostika a monitoring robotických systémov

V tejto kapitole sa zoznámime s pojmami monitoring, diagnostika a vysvetlíme si základné informácie týkajúce sa získavania, analyzovania a zobrazovania informácií z robotických systémov. Ďalej sú tu v stručnosti popísané primárne znalosti, ktoré sú potrebné k začatiu práce na diagnostickom balíčku pre robota.

2.1 Robot a robotický systém

Robot je definovaný ako automaticky riadený, opätovne programovateľný, viacúčelový manipulátor pre činnosť v troch alebo viac osách, ktorý môže byť buď upevnený na mieste alebo mobilný pre použitie v priemyselných automatických aplikáciách [5].

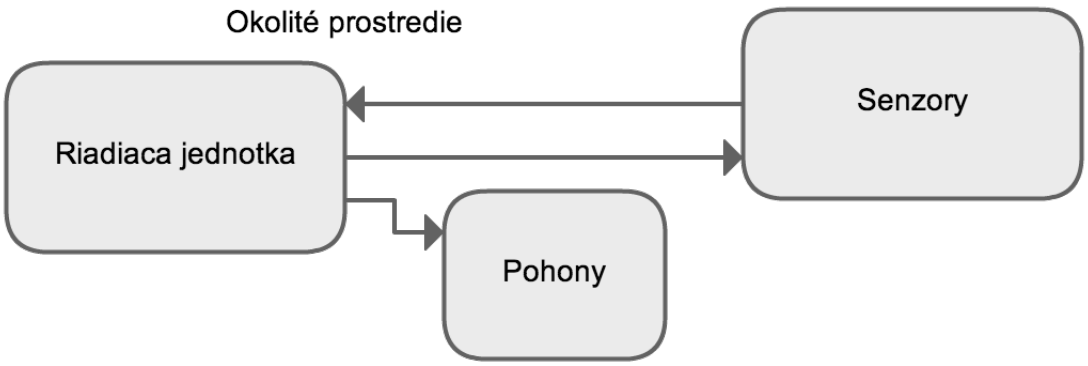
Čím je elektromechanický robot zložitejší, tým viac hovoríme už o robotickom systéme. Bežný robotický systém je komplexný mechanizmus, skladajúci sa z nasledovných častí:

- Nosná platforma – konštrukcia
- Pohony
  - Základňa
    - Pohyblivá
    - Pevná
  - Ramená
    - Pohyblivé
    - Pevné
- Riadiaca jednotka
  - Výpočtová jednotka
  - Riadiaci softvér
  - I/O modul
- Senzory
  - Polohové

  - Dotykové
  - Teplocitlivé
  - Svetlocitlivé
- Zdroj energie – autonómia
  - Batérie – plná autonómia
  - Prenos káblom – čiastočná autonómia
  - Iné
- Komunikačný systém
  - Rádiový
  - Zvukový
  - Svetelný
  - Iný

Mozgom každého moderného robotického systému je riadiaca jednotka. Riadiaca jednotka obsahuje ako bolo naznačené vyššie výpočtovú jednotku, riadiaci softvér a I/O modul. Jadrom je však riadiaci softvér. Všetky informácie, ktoré sú robotickému systému dostupné cez I/O modul sa spracovávajú a vyhodnocujú priamo týmto softvérom. Na základe vložených algoritmov riadiaci softvér určí nové parametre, ktoré opäť cez I/O modul posiela do okolia a riadi nimi svoje pohony. Tento cyklus sa v podstate opakuje až pokiaľ robot nedostane pokyn na ukončenie činnosti, alebo jeho činnosť priamo nezastaví na základe získaných informácii vlastný diagnostický systém.

Zjednodušená základná bloková schéma robota z pohľadu získania, prenosu a spracovania signálov je na obrázku 2.1.



Obrázok 2.1: Schéma robota z pohľadu získania, prenosu a spracovania signálov.

2.2 Monitoring

Pod pojmom monitoring rozumieme pravidelné pozorovanie a zaznamenávanie určitých aktivít, ktoré sa konajú v rámci programu alebo projektu, je to proces bežného získavania

informácií vo všetkých aspektoch projektu [2]. V našom prípade teda v priebehu času pravidelne získavame a pozorujeme informácie týkajúce sa rôznych častí zadaného mobilného robota.

Monitorovací system môže byť rozdelený do nasledovných funkčných častí [4]:

- Detekcia chyby – indikuje, že niečo nefunguje správne v monitorovanom systéme
- Izolácia chyby – klasifikácia čo nefunguje správne
- Identifikácia chyby – už určí hodnotu chyby

V konečnom dôsledku monitoring využíva systémy pre blokovanie neprípustných stavov. Tieto zabezpečujú ochranu systému pri poruche dôležitých častí robota, pri preťažení robota a pod. Ochrana systému proti prekročeniu prípustnných polôh členov sa zabezpečuje koncovými snímačmi.

Monitorovací softvér a následné algoritmy v prípade objavenia sa signálu havárie musia byť určené tak, aby sa riadením zrušil neželaný stav.

Okrem kontroly stavu mechanickej časti robota je potrebné blokovať činnosť systému predovšetkým pri poruche v riadiacej jednotke a vo výkonovej časti systému.

2.3 Diagnostika a diagnostický systém

Diagnostika je výskum postupov na určovanie technického stavu zariadení, strojov, technológií a podobne [3]. Diagnostický systém je teda systém navrhnutý za výslovným účelom skúmania informácií a zisťovania možných problémov spojených s daným zariadením, respektíve projektom. Poznáme dve základné metódy testovania funkčnosti systémov:

- Black box – je to testovanie mechanizmu bez znalosti ako funguje, zameriava sa na hodnotu výstupu pri zadanej hodnote vstupu.
- White box – poznáme ako mechanizmus funguje a na základe toho vhodne zvolenými vstupmi testujeme situácie, ktoré môžu nastať.

V tomto projekte by sa testovanie dalo označiť ako black box testovanie. Na základe vstupného stavu robota sú pri monitoringu získané výstupné dáta z ovládačov rôznych častí mobilného robota, ktoré sú ďalej podľa požiadaviek užívateľa triedené a následne zobrazované. Pri zobrazovaní je kladený dôraz na zobrazenie funkčnosti respektíve nefunkčnosti daného komponentu, stavové informácie sú brané až ako vedľajšie.

2.4 Potrebné znalosti

Pre vytvorenie plnohodnotnej aplikácie poskytujúcej diagnostické informácie týkajúce sa mobilného robota bolo potrebné vedieť odpovedať na nasledujúce otázky:

- Čo analyzovať – Keďže robot sa skladá z množstva hardvérových komponentov, musíme poznať ako fungujú a z čoho sa skladajú. To nám pomôže lepšie sa zamerať pri analýze na časti, ktoré sú náchylnejšie na nefunkčnosť. Podrobný popis robota a jeho častí nájdeme v kapitole 3.2.1.
- Ako analyzovať – Keď už robota poznáme a vieme, ktoré jeho časti chceme diagnostikovať, treba stanoviť ako získané informácie triediť. Ktoré z informácií sú pre nás podstatné, ktoré menej a ako na základe nich určiť stav daného komponentu.

- Ako výsledok zobraziť – Po spracovaní získaných informácií je potrebné rozhodnúť akým spôsobom budeme výsledok interpretovať. Základným rozdelením je interpretácia cez štandardný výstup na príkazovom riadku a pomocou grafického užívateľského rozhrania, ktoré je bežnému užívateľovi bližšie.

Kapitola 3

Zoznámenie sa s platformou ROS a mobilným robotom

Táto kapitola poskytuje základné informácie týkajúce sa mobilného robota používaného na Ústave počítačovej grafiky a multimédií Fakulty informačných technológií Vysokého učenia technického v Brne (ďalej len UPGM FIT VUT). Zoznámime sa s jeho hardvérovým vybavením a priblížime si operačný systém, pod ktorým robot pracuje.

3.1 Robot Operating System

Robotický operačný systém (ďalej ROS) je voľne šíriteľný framework určený pre tvorbu a vývoj aplikácií určených pre robotické systémy. Poskytuje štandardné služby operačného systému, akými sú hardvérové abstrakcie, ovládanie zariadenia na nízkej úrovni, implementácia bežne používaných operácií, odovzdávanie správ medzi procesmi a správu balíkov. Taktiež poskytuje nástroje a knižnice pre získavanie, vytváranie, písanie a spúšťanie kódu na viacerých počítačoch.

Systém je dostupný pre rôzne operačné systémy, pričom primárne podporovaný systém je vždy závislý od verzie ROSu. Pre nás je dôležitý ROS verzie Electric, ktorý podporuje Ubuntu verzie 11.10 alebo 11.04, ostatné systémy sú označené len ako experimentálne podporované – nutná je kompilácia a inštalácia zo zdrojových kódov.

V súčasnosti je vďaka trom hlavným knižniciam možné vytvárať programy pre ROS v jazykoch Python, C++ alebo Lisp. Ako experimentálne je možné použiť ďalšie dve knižnice, ktoré umožňujú programovať v jazykoch Java a Lua.

3.1.1 Systém súborov ROSu

Z dôvodu väčšej flexibility a robustnosti je ROS zostavený z množstva menších modulov vykonávajúcich rozdielne úlohy, ktoré po spustení dokážu medzi sebou komunikovať a vytvoriť tak jeden komplexný celok. Tieto moduly sú organizované na disku do hierarchickej štruktúry pozostávajúcej z dvoch hlavných častí:

- Package (balíček) – základná organizačná jednotka architektúry ROSu. Je to adresár, obsahujúci jednotlivé uzly systému (bližšie v časti 3.1.2), knižnice, premenné, prípadne skripty a špeciálny konfiguračný súbor manifest.xml, popisujúci daný balíček a jeho závislosti.

- Stack (súbor balíčkov) – súbor príbuzných balíčkov, ktorého cieľom je zjednodušiť proces zdieľania kódu. Je to zároveň základný mechanizmus ROSu pri distribuovaní softvéru. Každý súbor balíčkov je v určitej verzii a má stanovené závislosti na iných súboroch a ich verziách, čo zaručuje lepšiu stabilitu pri vývoji. Tieto a ďalšie konfiguračné údaje nájdeme v súbore stack.xml.

3.1.2 Softvérová štruktúra ROSu

Po spustení jednotlivé moduly vytvoria klient-klient sieť (obrázok 3.1) vzájomne komunikujúcich procesov (synchrónne alebo asynchrónne), ktoré sú riadené pomocou registračnej služby (procesu) nazvanej „master“. Táto služba poskytuje vyhľadávací mechanizmus, umožňujúci procesom vzájomne sa nájsť za behu systému a zdielať medzi sebou dáta, prípadne dáta ukladať do akejsi centralizovanej databázy, tzv. „parameter servra“.

Vytvorená sieť sa skladá z niekoľkých základných častí:

- Nodes (uzly) – navzájom nezávislé procesy prevádzajúce vlastné operácie a výpočty. Pri spustení každý uzol automaticky poskytne master procesu informácie o sebe, aby mohla byť zahájená komunikácia s ostatnými uzlami.

- Messages (správy) – dátové štruktúry, pomocou ktorých uzly medzi sebou komunikujú. Každá správa obsahuje informácie uložené do základných dátových typov, respektíve ich polí. Typ správy je uložený v súbore meno_balíčka/msg/môjTyp.msg.

- Topic (topik) – pomenovaná zbernica, ktorá umožňuje asynchrónnu komunikáciu medzi uzlami na princípe publikovania a odoberania správ. Jeden alebo viac uzlov publikuje svoje správy na daný topik, odkiaľ ich jeden alebo viac uzlov môže následne odoberať. Na jeden topik je možné publikovať (resp. odoberať z neho) správy len jedného typu.

- Services (služby) – systém sprostredkovávajúci synchrónnu komunikáciu medzi práve dvoma uzlami na princípe žiadosť – odpoveď. Štruktúra žiadosti a odpovede je popísaná v súbore meno_balíčka/srv/môjTyp.srv.

- Bags – mechanizmus, pomocou ktorého je možné ukladať a znovu prehrávať komunikáciu prebiehajúcu formou správ.

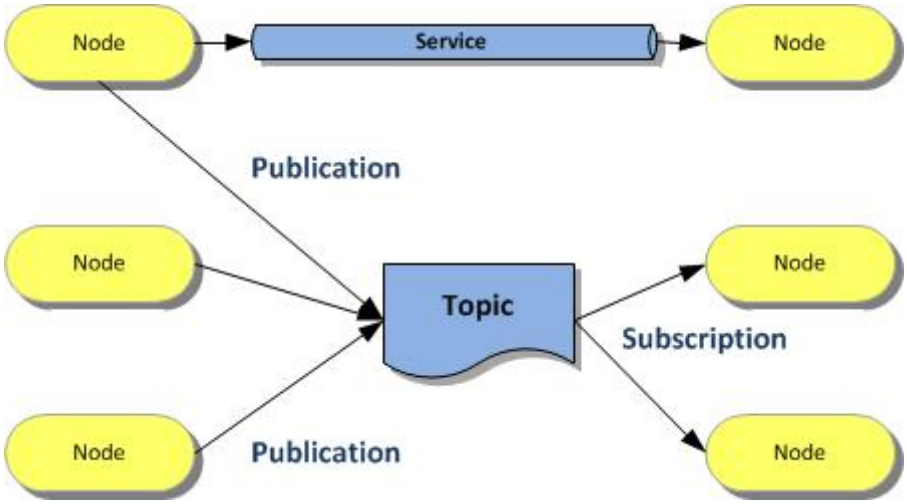
3.1.3 Užitočné balíčky a nástroje

Pri riešení tohoto projektu boli vo veľkej miere využívané nasledovné nástroje a balíčky:

- Gazebo – 3D simulátor určený pre vonkajšie prostredia, schopný simulovať väčší počet robotov, senzorov a objektov naraz. Dokáže generovať realistickú odozvu senzorov a verne zobraziť vzájomné pôsobenie pevných telies.

- Rviz – 3D vizualizátor, ktorý nezahŕňa výpočetný modul pre simuláciu, ponúka len možnosť zobrazenia rôznych typov dát publikovaných ostatnými uzlami v správnom formáte.

- Roslaunch – nástroj pre jednoduché spúšťanie rozličných uzlov ROSu. Ponúka možnosti lokálneho ako aj vzdialeného spustenia, štart viacerých uzlov naraz, spustenie



Obrázok 3.1: Typy komunikácie medzi uzlami (obrázok prevzatý z [1])

s parametrami ukladanými na parameter server, či opakované spustenie uzla v prípade nečakaného zastavenia. Konfiguračné informácie pre daný uzol sú uložené v tzv. launch súbore (koncovka .launch) vo formáte XML, pomocou ktorého sa následne daný uzol spúšťa.

- Roscore – Kolekcia uzlov a programov, ktoré sú predpokladom systému založenom na ROS. Roscore musí byť spustené, aby mohli uzly medzi sebou komunikovať. Roscore automaticky spúšťa:
  - ROS Master
  - ROS Parameter Server
  - Rosout logging node
- Robot Monitor – jednoduchý GUI nástroj, pre zobrazenie diagnostiky robota. Odoberá `/diagnostic_agg` topik a hierarchicky zobrazuje informácie v podobe stromu.
- Rosbag – nástroj na ukladanie komunikácie na topikoch a opätovné prehrávanie zaznamenaných správ.
- Rosdoc – nástroj na automatické generovanie dokumentácie zo zdrojových súborov.

3.1.4 Ďalšie možnosti

Zaujímavou možnosťou je *Rosjava*. Je to čistá implementácia jadra ROSu v jazyku Java, vytvorená firmou Google a Willow Garage tak, aby bola plne kompatibilná s mobilným operačným systémom Android. Vznikla z dôvodu súčasného rozšírenia Androidu medzi užívateľmi a jeho širokej škále možností sledovania, kontrolovania a ovládania robota (audio, video, akcelerometer, gyroskop, GPS, wifi, dotykový displej, atď.).

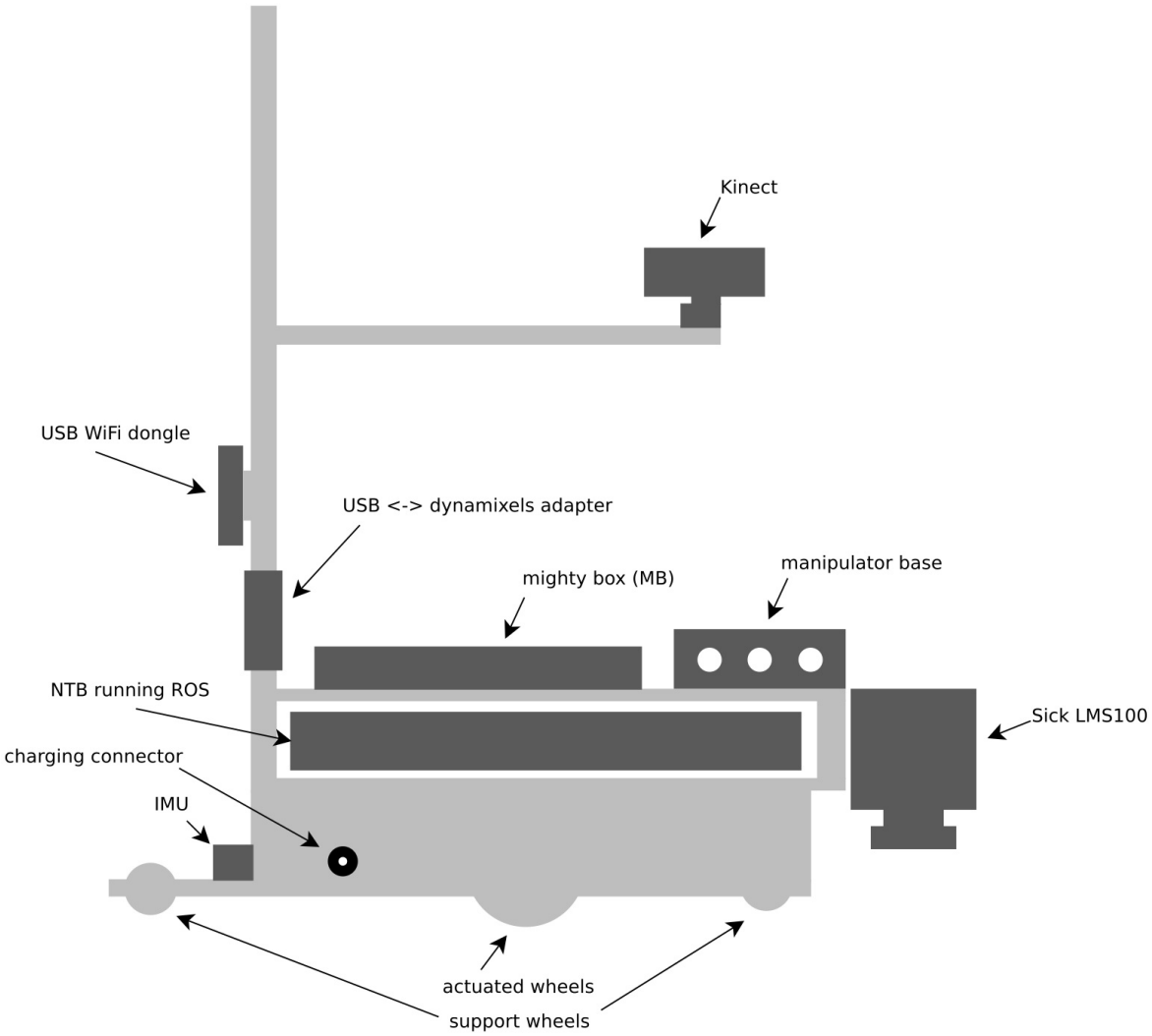
3.2 Mobilný robot

Pre účely tohoto projektu bol zvolený mobilný robot využívaný na UPGM FIT VUT v Brne, pod označením TB2. Je to mobilná robotická platforma určená do vnútorných priestorov,

bežiaca na ROSe aktuálne vo verzii Electric. Ponúka radu bežných senzorov, robotické rameno a dlhú výdrž batérií.

3.2.1 Hardvérové vybavenie TB2

TB2 vznikol na základe štandardnej robotickej platformy nazvanej Turtlebot, disponuje ale väčším množstvom senzorov, manipulátorom a vyššou životnosťou batérií. Základné časti TB2 (viď. obrázok 3.2):



Obrázok 3.2: Časti robotickej platformy TB2

- Základňa – TB2 je postavený na základe upraveného robotického vysávača iRobot Roomba 530 (všetky jeho čistiace časti boli odstránené). Komunikáciou prebiehajúcou cez SCI (Serial Command Interface) rozhranie je možné riadiť elektromotory a zisťovať stav všetkých senzorov.

- Napájanie robota – základňa je vybavená vlastnou Ni-Mh batériou (2x 3 Ah), starajúcou sa o pohyb robota. Ostatné senzory a pohybové jednotky (manipulátor) sú napájané z Li-On batérií umiestnených na zadnej časti robota, pričom väčšia (8,8 Ah) je obvykle použitá na napájanie robotickej ruky a menšia (6,6 Ah) na napájanie senzorov. V prípade potreby je možné tieto batérie zameniť.

- Doska plošných spojov (nazývaná tzv. mighty box) – pripojená do riadiaceho notebooku, slúžiaca ako konvertor z USB na sériový port pre základňu, dve IMU (inertial measurements unit) a vnútorný mikroprocesor. Taktiež obsahuje obvody pre poskytovanie +12 V napájania senzorom (Kinect, laserový skener) z Li-On batérií a dva USB porty, ktoré môžu slúžiť na napájanie zariadení ako externé harddisky a podobne.

  Vnútorný mikroprocesor monitoruje stav všetkých batérií a posiela ho v ASCII forme cez sériový port. Vidieť ho je možné po otvorení /dev/ttyUSB3 pri prenosovej rýchlosti 19200 baudov.

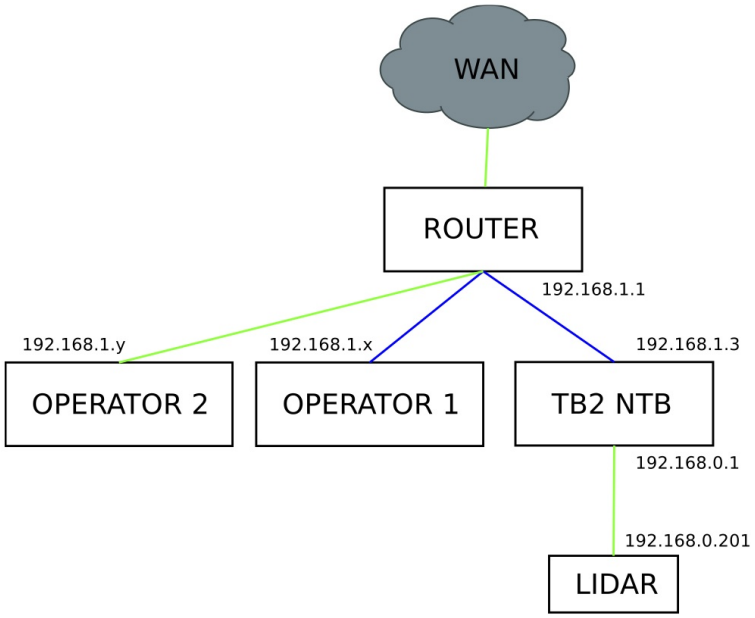
- Senzory – na vrchu robota sa nachádza Microsoft Kinect, ktorý je schopný produkovať hĺbkové RGBD obrázky s rozlíšením 640x480 pixelov pri 30 Hz. Pripojený je k notebooku pomocou USB 2.0 a napájaný z mighty boxu.

  Predná strana TB2 je vybavená laserovým skenerom Sick LMS100, so zorným poľom 270 ° rozdeleným na úseky veľkosti 0,5 °, dosahom 20 metrov a frekvenciou skenovania 50 Hz. Napájaný je z mighty boxu a k notebooku pripojený cez ethernet.

  Súčasťou robota sú dve inerciálne meracie jednotky (inertial measurement unit – IMU) CH Robotics UM6. Sú kombináciou gyroskopu, akcelerometra a magnetického senzoru a poskytujú orientáciu pri frekvencii až do 500 Hz.

- Manipulátor – robotická ruka CrustCrawler AX-18A Smart Robotic Arm, schopná manipulovať so záťažou do 0,907 kg, pri rotácii až do 300 °, umožňujúca krokovanie rýchlosti a točivého momentu v 1024 krokoch. Poskytuje informácie o polohe, rýchlosti, zaťažení, napätí a teplote pri maximálnej komunikačnej rýchlosti 1 Mbps.

- Wifi router – sieť TB2 pozostáva z wifi routra (aktuálne TP-LINK TL-WR1043ND – nie priamo súčasť TB2), robota a operátora (obrázok 3.3). Robot je vždy pripojený k routru na adrese 192.168.1.3, pri fungujúcom DNS stačí len jeho meno (turtlebot). Operátor sa môže pripojiť k routru káblom alebo wifi (max. 300 Mbps) a následne ovládať robota vzdialene.



Obrázok 3.3: Sieť TB2 robota, modrá znamená bezdrôtové spojenie, zelená linkové spojenie.

Kapitola 4

Návrh aplikácie

Keď sme sa oboznámili s hardvérovým a softvérovým vybavením mobilného robota, pre ktorého máme v tomto projekte vytvoriť diagnostický balíček, môžeme sa zamyslieť nad návrhom našej aplikácie. V tejto kapitole sa budeme zaoberať detailným popisom tohoto návrhu.

4.1 Požiadavky na aplikáciu

Základnou požiadavkou na aplikáciu je schopnosť komunikovať s jednotlivými časťami robota a vedieť zobraziť ich stavové informácie. Bude ju teda tvoriť uzol, zachytávajúci požadované informácie dostupné prostredníctvom topikov, prípadne z iných zdrojov, ak topiky tieto informácie neposkytujú. Následne informácie uloží a rozhodne, ktoré z nich bude ďalej zobrazovať a akým spôsobom. Práve z dôvodu množstva nastavení, by aplikácia mala užívateľovi umožňovať aj jednoduchú konfiguráciu.

4.1.1 Jazyk aplikácie

Pre potreby našej aplikácie nie je nutné sa zaoberať jazykmi podporovanými len experimentálne ako sú Java či Lua. Ako dostatočné bude použitie jazyka C++ (najrozšírenejší v ROSe, jeho knižnica je navrhnutá na vyšší výkon aplikácie) alebo jazyka Python (knižnica orientovaná na rýchlosť tvorby aplikácií pred výkonom). Ktorýkoľvek z nich poskytne prostriedky potrebné pre implementáciu a je možné ho teda použiť.

4.1.2 Konfigurácia aplikácie

Keďže nami navrhovaný balíček by mal byť nastaviteľný podľa potrieb každého užívateľa zvlášť, bolo by vhodné aby sa dal ľahko konfigurovať. Pre tento účel je vhodné použiť nástroj roslaunch (popísaný v časti 3.1.3), ktorý nám ROS ponúka. Možnosťou je aj pri spustení programu zadať parametre do príkazového riadku, no voči roslaunch je toto riešenie slabo prehľadné a málo efektívne.

Základnou nastaviteľnou vlastnosťou musí byť možnosť určiť, ktoré časti robota chceme monitorovať a diagnostikovať. Medzi ostatné upraviteľné možnosti by mala ďalej patriť konfigurácia zobrazenia získaných informácií tak, aby vyhovovali požiadavkám užívateľa.

4.1.3 Zachytávanie informácií z topikov

Ako bolo spomenuté v časti 3.1.2, informácie o jednotlivých častiach robota sú publikované formou správ na topiky. V našej aplikácii teda potrebujeme mať časť, ktorá sa prihlási k odberu týchto topikov a bude sa starať o zachytávanie správ z nich a prvotné ukladanie týchto správ.

4.1.4 Zbieranie informácií mimo topikov

Keďže nie všetky informácie sú dostupné prostredníctvom topikov (napr. intenzita wifi signálu, informácie o aktuálnom napätí v batériách nie je možné získať touto cestou) je potrebné vytvoriť samostatnú časť aj pre získavanie a ukladanie týchto informácií.

4.1.5 Triedenie informácií

Po zozbieraní a uložení všetkých dostupných informácií, týkajúcich sa zvolených častí robota, bude musieť aplikácia tieto informácie vedieť ďalej spracovávať. Základné triedenie informácií bude zahŕňať rozdelenie na informácie ohľadom funkčnosti komponentu (funguje/nefunguje) a informácie ohľadom jeho stavu (aktuálna teplota/rýchlosť/poloha...). Pokročilé triedenie sa môže následne starať o oddelenie pre užívateľa podstatných stavových informácií od tých menej podstatných.

4.1.6 Ukladanie informácií

Zozbierané a pretriedené dáta je následne nutné uložiť. Aby sme si uľahčili prácu pri ich zobrazovaní (4.1.7), by vhodným spôsobom uloženia mohla byť dátová štruktúra obsahujúca všetky informácie na jednom mieste. Pri ukladaní však treba vždy dbať na kontrolu duplicitných informácií. V prípade, že sa informácia k danému komponentu už v štruktúre nachádza, je nutné túto informáciu obnoviť najnovšími získanými dátami a nie ukladať ďalší krát.

4.1.7 Zobrazovanie informácií

Po uložení spracovaných dát do výslednej dátovej štruktúry je potrebné vytvoriť časť aplikácie, ktorá sa postará o zobrazenie týchto dát. Za vhodné spôsoby zobrazenia budeme považovať:

- Štandardný výstup príkazového riadku – najjednoduchšie na implementáciu, vhodné pri testovaní a debugovaní, menej vhodné pre koncového užívateľa
- Vlastné užívateľské rozhranie – vhodné pre koncového užívateľa, optimalizácia pre zobrazenie dát práve z TB2, implementačne náročné
- Aplikácie tretích strán – kompromis medzi štandardným výstupom a tvorbou vlastného rozhrania, implementačne nenáročné, vhodné pre koncového užívateľa

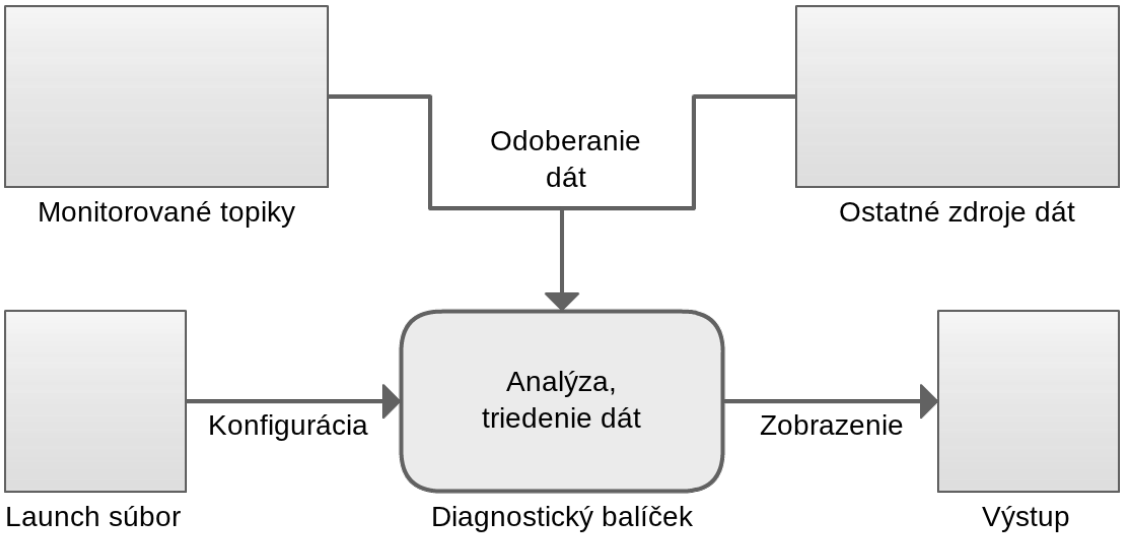
Možným riešením je aj kombinácia viacerých uvedených spôsobov zobrazenia. Pri implementácii však budeme potrebovať pre každé z nich zvlášť pridať časť starajúcu sa práve o jeden konkrétny druh zobrazenia.

4.2 Výsledný návrh

V časti 4.1 sme sa zoznámili s detailnými požiadavkami týkajúcimi sa vytváranej aplikácie. Z nich vyplýva, že výsledkom našej práce by teda mal byť uzol pre ROS, napísaný v jazyku Python alebo C++ starajúci sa o diagnostiku robota TB2. Čerpať dáta bude z topikov, ktoré obsahujú informácie o jednotlivých častiach robota, prípadne z iných zdrojov, pre informácie nezistiteľné z topikov (sila wifi signálu, napätie v batériách). Tieto informácie bude deliť na podstatné (uložené, zobrazované) a nepodstatné (zahadzované). V konečnom kroku uložené informácie vhodným spôsobom zobrazí – štandardný výstup, aplikácie tretích strán.

Konfiguráciu uzla bude zaisťovať nástroj roslaunch, kde užívateľ pomocou launch súboru stanoví, ktoré komponenty budú sledované, prípadne určí úpravu zobrazovaných dát.

Pre lepšiu predstavu je výsledný návrh aplikácie zobrazený na obrázku 4.1.



Obrázok 4.1: Diagram znázorňujúci výsledný návrh aplikácie.

4.3 Ďalšie možnosti

Možným vylepšením vytváranej aplikácie by bolo rozšírenie zoznamu konfigurovateľných parametrov. Pomocou launch súboru by bolo možné napríklad zvoliť požadovaný typ zobrazenia uložených diagnostických dát či upraviť zoznam sledovaných topikov. Tieto možnosti nie sú kľúčové pre našu aplikáciu, no prispeli by k univerzálnosti a väčšiemu komfortu pre užívateľa i keď na úkor jednoduchosti.

Kapitola 5

Implementácia

Na základe podrobného návrhu popísaného v kapitole 4 sme vytvorili aplikáciu, ktorá bude v čo najväčšej miere tento návrh spĺňať. Opis jej jednotlivých častí, spôsob implementácie a odlišnosti od návrhu si popíšeme v nasledujúcej kapitole.

5.1 Architektúra

Základom pre našu aplikáciu bolo vytvorenie balíčka, ktorý obsahuje implementované všetky časti potrebné v rámci tohoto projektu. Tento balíček je nazvaný `btb_diag` a jeho primárne umiestnenie je v súbore balíčkov `tb2_main_packages`, kde sa nachádzajú aj ostatné balíčky týkajúce sa TB2.

Za jazyk aplikácie sme zvolili jazyk C++. Bolo to prevažne z dôvodu lepšieho výkonu výslednej aplikácie, ale taktiež kvôli pokročilejším skúsenostiam s týmto jazykom voči jazyku Python.

Súčasťou balíčka `btb_diag` sú nasledujúce súbory:

- `mainfest.xml` – obsahuje základnú špecifikáciu balíčka ako je meno autora, druh licencie či stránka aplikácie. Dôležitou časťou je stanovenie balíčkov, na ktorých je naša aplikácia závislá. Sú nimi knižnica roscpp (aplikácia je písaná v jazyku C++) a balíčky týkajúce sa komunikácie, umožňujúce komunikovať pomocou správ rôzneho formátu (diagnostic_msgs, visualization_msgs).

  Ak sa súbor manifest.xml nachádza v adresári, všetky adresáre a súbory nachádzajúce sa vnútri tohoto adresáru sú súčasť jedného balíčka.

- `CMakeLists.txt` – vstupný súbor pre systém CMake, ktorý sa stará o zostavenie balíčkov. Popisuje ako kód zostaviť a kam ho nainštalovať.

- `Makefile` – načíta cmake definície, ktoré zaistia multiplatformový preklad aplikácie.

- `mainpage.dox` – šablóna pre automatické generovanie dokumentácie zo zdrojových kódov pomocou nástroja Doxygen. Obsahuje základný popis aplikácie. Vygenerovaná dokumentácia bude uložená do adresára btb_diag/doc.

- `btb_diag.launch` – konfiguračný súbor aplikácie, starajúci sa o jej spustenie a predovšetkým o predanie parametrov od užívateľa. Pre jeho detailný popis, pozri časť 5.2.

- `btb_diag.cpp` – hlavný súbor vytvorenej aplikácie. Nachádzajú sa v ňom funkcie starajúce sa o všetky aspekty fungovania programu. Detailný popis týchto funkcií nájdeme v časti 5.3.

- `btb_diag.h` – hlavičkový súbor k hlavnému súboru aplikácie. Obsahuje prototypy funkcií, použité knižnice a definície globálnych premenných.

Súbory, ktoré sú dôležité pre tento projekt, si v ďalšej časti kapitoly popíšeme detailnejšie.

5.2 Konfiguračný súbor

Konfiguračným súborom našej aplikácie rozumieme súbor `btb_diag.launch`. Jeho úlohou je aplikáciu spustiť a uzol `diagnostics_collector` v rámci nej. Ako parameter výstupu pre uzol bol zadaný parameter „screen“, na základe čoho budeme môcť štandardný výstup a štandardný chybový výstup zobraziť na obrazovku (miesto predvoleného súboru so záznamami). Keďže plánujeme výsledné diagnostické informácie zobrazovať aj týmto spôsobom, budeme túto možnosť potrebovať.

Užívateľ ďalej v tomto súbore špecifikuje hodnoty niekoľkých premenných potrebných k diagnostike robota. Do premennej „items_arr“ zadá názvy komponentov, ktoré bude program monitorovať a analyzovať, oddelené medzerou. Následne musí do premennej „names_arr“ zadať v tom istom poradí názvy popisujúce jednotlivé tieto komponenty, pod ktorými budú vo výsledku zobrazené.

Ďalšiou možnosťou konfigurácie je určenie hraničných hodnôt intenzity wifi signálu. K dispozícii je päť premenných „excelent“, „good“, „fair“, „poor“ a „nosignal“, ktoré stanovujú rozmedzia v ktorých bude signál označený danou úrovňou kvality.

Príklad konfiguračného súboru je uvedený v citácii zdrojového kódu 5.1.

```
<?xml version="1.0"?>
<launch>
    <node pkg="btb_diag" name="diagnostics_collector"
        type="btb_diag" output="screen"/>
<!-- sets break points between levels of wifi signal strength -->
    <param name="excelent" value="-45" type="int" />
    <param name="good" value="-65" type="int" />
    <param name="fair" value="-85" type="int" />
    <param name="poor" value="-85" type="int" />
    <param name="nosignal" value="-256" type="int" />
<!-- list of items to look for and show, separated with space -->
    <param name="items_arr" value="Motor Joint Controller"
        type="string" />
<!-- list of names to show items like, separated with space,
   have to correspond with items_arr -->
    <param name="names_arr" value="Motors Joints Controllers"
        type="string" />
</launch>
```

Zdrojový kód 5.1: Príklad konfiguračného launch súboru pre `btb_diag`.

5.3 Hlavná časť aplikácie

Hlavná časť aplikácie je tvorená súbormi `btb_diag.cpp` a `btb_diag.h`, pričom samotné jadro projektu je implementované v `btb_diag.cpp`. Jednotlivé požiadavky na aplikáciu spomínané v časti návrhu aplikácie (4.1) sme sa snažili v tomto súbore implementovať pomocou samostatných funkcií. V nasledujúcej časti si podrobne popíšeme spôsob implementácie navrhnutých častí a vytvorených funkcií.

5.3.1 Zachytávanie informácií z topikov

Pre čerpanie informácií dostupných formou správ na topikoch potrebujeme poznať názov topiku a typ správ na ňom publikovaných. Ďalej potrebujeme vytvoriť uzol, pomocou ktorého bude prebiehať komunikácia nášho programu s ostatnými časťami systému (bude využitý aj pri publikovaní dát).

Na základe časti **??** sme sa rozhodli, že najdôležitejším topikom, ktorý poskytuje dáta je pre nás `/diagnostic` topik, poskytujúci správy typu `diagnostic_msgs/DiagnosticArray`. Budeme sa teda venovať prevažne jemu.

Pre komunikáciu sme vytvorili uzol nazvaný `diagnostic_collector`. Pod týmto názvom bude vyhladatelný a zobrazitelný aj inými časťami a nástrojmi ROSu. Pomocou metódy `subscribe`, dostupnej prostredníctvom prístupového bodu nášho uzlu, sa prihlásime k topiku a určíme ktorá funkcia sa bude volať vždy pri publikovaní novej správy na tento topik. V našom prípade to bude funkcia `diag_callback`, v rámci ktorej prebehne aj prvotné uloženie získaných informácií do pripravenej štruktúry `diag_data`.

5.3.2 Spracovanie a triedenie informácií

O spracovanie a triedenie dát z topikov, uložených v štruktúre `diag_data`, sa stará funkcia `diag_callback`.

Na to aby sme tieto dáta mohli triediť, musíme poznať ich spôsob uloženia vrámci štruktúry. Jej hlavnými časťami sú pole obsahujúce prevažne dáta týkajúce sa jednotlivých časti robota, kde každá táto časť je tvorená názvom, správou, id a ďalším polom detailných informácií k danej časti.

Funkcia `diag_callback` vo `for` cykle postupne prechádza všetky elementy hlavného pola a vyhľadáva v názvoch časti robota zadané užívateľom pomocou konfiguračného launch súboru (časť 5.2). Pre každú nájdenú zhodu je možnosť pretriediť pole obsahujúce detailné informácie. V našom prípade však poskytujeme naspäť užívateľovi všetky dostupné informácie obsiahnuté v tomto poli.

Podstatnou úlohou je zo správy každého elementu hlavného pola zistiť stav komponentu. Pri ukladaní (viď. 5.3.4) ho následne budeme prevádzať na level (0 – „ok“, 1 – „warn“, 2 – „error“, 3 – „stale“).

O spracovanie informácií nepochádzajúcich z topikov (intenzita wifi signálu, napätie v batériách) sa starajú ich samostatné funkcie popísané v časti 5.3.3.

Nasledujúcim krokom je nájdené informácie vhodne uložiť, aby boli jednoducho zobraziteľné.

5.3.3 Zbieranie informácií mimo topikov

Ako bolo spomínané v časti 4.1.4, informáciami nedostupnými prostredníctvom topikov sú v našom prípade intenzita wifi signálu a aktuálne napätie v batériách. Budeme preto musieť

pre ich získanie, spracovanie a uloženie vytvoriť samostatné časti, venujúce sa len tomuto problému.

Intenzita wifi signálu

O určenie kvality signálu sa stará funkcia `wifi_info`. Keďže jadrom robota je notebook pripojený k routru prostredníctvom wifi siete (obrázok 3.3), jedná sa prakticky o zistenie intenzity signálu prijímaného týmto notebookom. Operačným systémom notebooku je Linux (Ubuntu), intenzita signálu je teda dostupná v pseudosúbore `/proc/net/wireless`. Tento pseudosúbor však obsahuje aj ďalšie informácie (rozhranie, status, šum... – viď. obrázok 5.2, možnosť pridať ako rozšírenie aplikácie), kvôli čomu je nutné ho parsovať.

Po získaní samotnej hodnoty signálu, rozhodneme na základe parametrov zadaných prostredníctvom launch súboru (časť 5.2), do ktorej úrovne kvality túto hodnotu zaradíme. K dispozícii máme vytvorených päť úrovní: „EXCELENT“, „GOOD“, „FAIR“, „POOR“ a „NO SIGNAL!“. Pri úrovni signálu označenej „NO SIGNAL!“ je stav wifi signálu vyhlásený ako chybný – „ERROR“, inak je v poriadku – „OK“.

Ak sme priradili hodnote úroveň, musíme následne informácie uložiť a pripraviť k zobrazeniu, viac v časti 5.3.4.

```
Inter-| sta-|   Quality        |   Discarded packets               | Missed | WE
 face | tus | link level noise |  nwid  crypt   frag  retry   misc | beacon | 22
  eth1: 0000    5.  -256.  -57.        0      0      0     56      0        0
```

Obrázok 5.2: Obsah pseudosúboru `/proc/net/wireless`.

Napätie v batériách

Zisteniu napätia v batériách robota (sú tri, viď. 3.2) sa venuje funkcia `battery_info`. Ako bolo spomenuté v časti 3.2.1, vnútorný mikroprocesor dosky plošných spojov monitoruje stav všetkých batérií a posiela ho v ASCII forme, zobrazenej na obrázku 5.3, cez sériový port. Nám ho teda stačí zachytiť a spracovať.

Zachytiť ho je možné z kontrolného terminálu `/dev/ttyUSB3`, pri prístupe prenosovou rýchlosťou 19200 baudov. V prípade nedostupnosti terminálu je stav napätia v batériách automaticky vyhlásený ako chybný – „ERROR“, inak je v poriadku – „OK“.

Po získaní „surového“ stavu batérií je nutné ho z dôvodu lepšej narábatelnosti rozparsovať na hodnoty jednotlivých batérií, ktoré budú priradené premenným.

V poslednom kroku informácie opäť uložíme a tým pripravíme k zobrazeniu – časť 5.3.4.

```
#BS: 16,5V, BA: 16,6V, BR: : 13,5V
```

Obrázok 5.3: Obsah pseudosúboru `/proc/net/wireless`.

5.3.4 Ukladanie informácií

Keďže sme sa rozhodli zvoliť ako jeden zo spôsobov zobrazenia aplikáciu Robot Monitor (3.1.3), prispôsobíme ukladanie informácií aby jej vyhovovali v maximálnej možnej miere. Komunikácia s touto aplikáciou prebieha cez `/diagnostics_agg` topik pomocou správ typu

`diagnostic_msgs/DiagnosticArray`. Vytvoríme si teda štruktúru tohoto typu a dáta budeme ukladať priamo do nej. Bude globálna, aby k nej mali všetky funkcie jednoduchý prístup, nazveme ju `diag_array`.

Aby však informácie boli v aplikácii Robot Monitor správne zobrazené, musíme do štruktúry ukladať záznamy dvoch typov. Jeden záznam bude tvoriť názov komponentu, stav, level a pole s jeho detailami. Druhý bude združovať komponenty rovnakého druhu – meno bude pre ne spoločné (užívateľ ho určil cez launch súbor, viď. 5.2), level a stav budú odpovedať najhoršiemu levelu a stavu spomedzi komponentov tohoto druhu a pole budú tvoriť záznamy typu názov konkrétneho komponentu a jeho stav. Príklad tejto štruktúry je v citácii kódu 5.4.

Pozor treba dávať na duplicitu informácií. Ak sa informácie ohľadom daného komponentu ešte v štruktúre nenachádzajú, uložíme ich. Ak sa však nachádzajú, musíme ich vyhladať a obnoviť najnovšími získanými.

```
status:
-   level: 1
    name: Joints
    message: WARN
    values:
    -   key: Joint Controller (kinect_joint)
        value: OK
    -   key: Joint Controller (right_finger_joint)
        value: WARN
-   level: 0
    name: Joint Controller (kinect_joint)
    message: OK
    hardware_id: Robotis Dynamixel [11] on port smart_arm
    values:
    -   key: Velocity
        value: 0.0
    -   key: Temperature
        value: 35
```

Zdrojový kód 5.4: Úryvok štruktúry, ktorá ukladá dáta určené na zobrazenie.

5.3.5 Zobrazovanie informácií

Aby zobrazované dáta boli vždy presné, musia byť aktualizované najčastejšie ako je to možné. Znamená to teda, že vždy keď prídu nové dáta treba prepísať tie pôvodné. Keďže funkcia `diag_callback` je volaná vždy pri príchode nových dát na topik, je vhodné, aby táto funkcia pri zavolaní prepísala aj zobrazované dáta. Je však viacero možností zobrazenia. V našom projekte sme sa rozhodli dáta zobraziť troma spôsobmi:

Na štandardný výstup príkazového riadku

Pre účel zobrazenia informácií na štandardný výstup príkazového riadku sme vytvorili funkciu `print_data`. Bude však z dôvodu prehľadnosti zobrazovať len stav komponentov, bez detailných informácií. Funkcia vo `for` cykle prejde všetky záznamy štruktúry `diag_array`.

Keď narazí na záznam týkajúci sa kolekcie komponentov jedného druhu (vo svojom poli má informácie o jednotlivých komponentoch, nie detaily), vypíše názov kolekcie a následne všetky jej komponenty. V prípade nefunkčných komponentov vypíše informácie aj na štandardný chybový výstup. Príklad takéhoto výstupu je na obrázku 5.5.

```
[ INFO] [1368553622.826952233, 126.001000000]: Joints: OK
[ INFO] [1368553622.827174859, 126.001000000]: Joint (elbow_flex_joint): OK
[ INFO] [1368553622.827322996, 126.001000000]: Joint (kinect_joint): OK
[ INFO] [1368553622.827574271, 126.001000000]: Joint (left_finger_joint): OK
[ INFO] [1368553622.827827642, 126.001000000]: Joint (left_wheel_joint): OK
[ INFO] [1368553622.828059910, 126.001000000]: Joint (right_finger_joint): OK
[ INFO] [1368553622.828306433, 126.001000000]: Joint (right_wheel_joint): OK
[ INFO] [1368553622.831080728, 126.001000000]: Joint (shoulder_pan_joint): OK
[ INFO] [1368553622.831367849, 126.001000000]: Joint (shoulder_pitch_joint): OK
[ INFO] [1368553622.831638270, 126.001000000]: Joint (wrist_roll_joint): OK
[ INFO] [1368553622.833198328, 126.001000000]: ---
[ERROR] [1368553622.849271181, 126.002000000]: Batteries: ERROR
[ INFO] [1368553622.849763109, 126.002000000]: Batteries: ERROR
[ERROR] [1368553622.850306815, 126.002000000]: Sensors Battery: ERROR
[ INFO] [1368553622.850792315, 126.002000000]: Sensors Battery: ERROR
[ERROR] [1368553622.851278304, 126.002000000]: Arm Battery: ERROR
[ INFO] [1368553622.852859393, 126.002000000]: Arm Battery: ERROR
[ERROR] [1368553622.853373821, 126.002000000]: Robot's Internal Battery: ERROR
[ INFO] [1368553622.853889648, 126.002000000]: Robot's Internal Battery: ERROR
[ INFO] [1368553622.854430141, 126.002000000]: ---
[ INFO] [1368553622.854944150, 126.002000000]: WIFI: OK
[ INFO] [1368553622.855450054, 126.002000000]: Strength: -44
[ INFO] [1368553622.855974196, 126.002000000]: Status: EXCELLENT
[ INFO] [1368553622.856471435, 126.002000000]: ---
```

Obrázok 5.5: Príklad zobrazenia výstupu aplikácie na štandardný výstup príkazového riadku

Pomocou aplikácie Robot Monitor

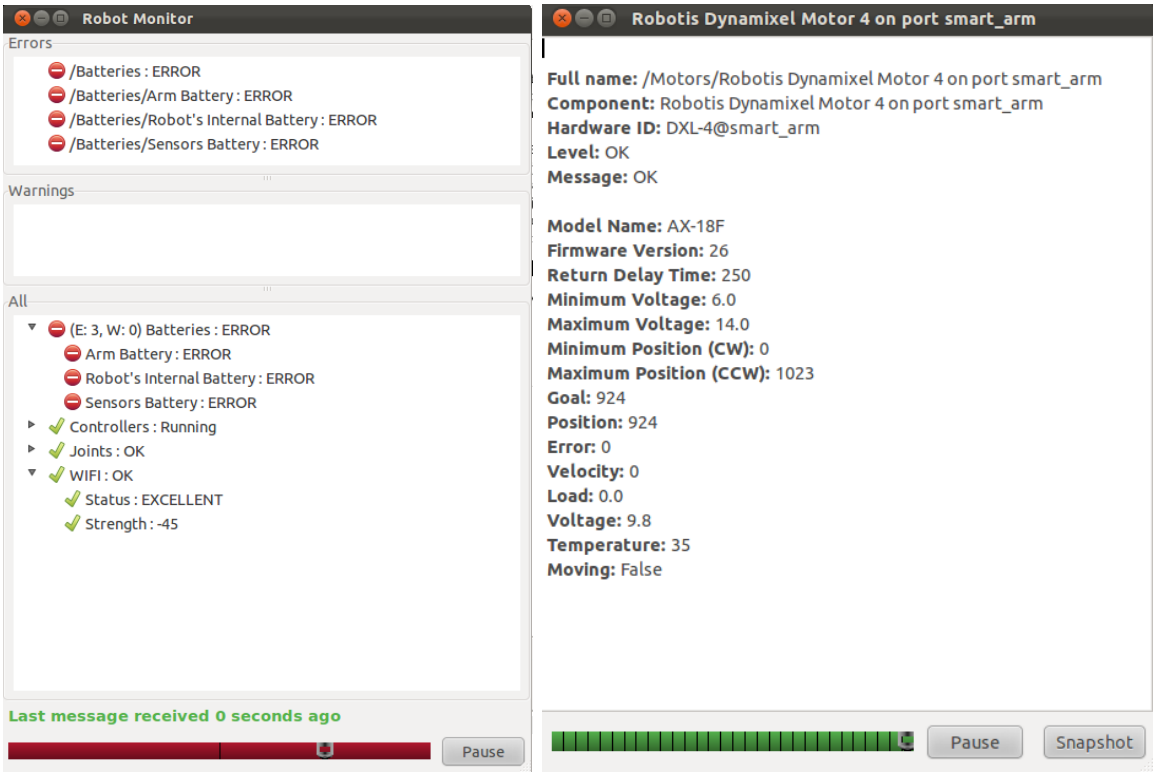
Ak chceme informácie zobraziť v aplikácii Robot Monitor (3.1.3), musíme ich publikovať v správnom formáte na `/diagnostics_agg` topik, z ktorého ich táto aplikácia čerpá. Keďže ich v správnom formáte máme (`diagnostic_msgs/DiagnosticArray`), stačí sa postarať o publikovanie. Pomocou metódy `advertise`, dostupnej prostredníctvom prístupového bodu nášho uzlu `diagnostic_collector`, oznámime procesu master, že budeme na topik `/diagnostics_agg` publikovať dáta. Táto metóda vráti objekt, ktorý pomocou metódy `publish` dáta vo funkcii `diag_callback` odošle.

Príklad zobrazenia informácií pomocou aplikácie Robot Monitor, vrátane detailného popisu jedného komponentu, nájdeme na obrázku 5.6.

Pomocou vizualizátoru Rviz

Rviz dokáže naše informácie zobraziť pomocou markerov. Keďže zobrazenie všetkých dát by bolo týmto spôsobom neprehľadné, rozhodli sme sa zobrazovať vždy len jednu informáciu – chybovú. Informácie o nefunkčnosti niektorej časti sú pre nás dôležitejšie a preto výraznejšie zobrazené. Po odstránení závady marker zmizne, prípadne ho nahradí iný, ak je ešte niektorá časť robota označená ako chybná.

Pri zobrazovaní informácií pomocou nástroja Rviz 3.1.3 je situácia veľmi podobná, ako pri aplikácii Robot Monitor. Rviz takisto zobrazuje informácie publikované na topikoch. Náš marker musí byť typu `visualization_msgs/Marker` a publikovaný bude na topik `/visualization_marker`. Musíme však nastaviť niektoré ďalšie parametre ako je pozícia, veľkosť, farba či dĺžka zobrazenia. Všetky z nich okrem dĺžky zobrazenia sú relatívne –



Obrázok 5.6: Príklad zobrazenia výstupu aplikácie prostredníctvom Robot Monitoru spolu s detailom komponentu

môžu byť upraviteľné podľa predstáv užívateľa (pomocou launch súboru – možné rozšírenie). Dĺžka zobrazenia je stanovená na celú dobu behu master procesu – marker sa zmení len pri zmene stavu komponentu.

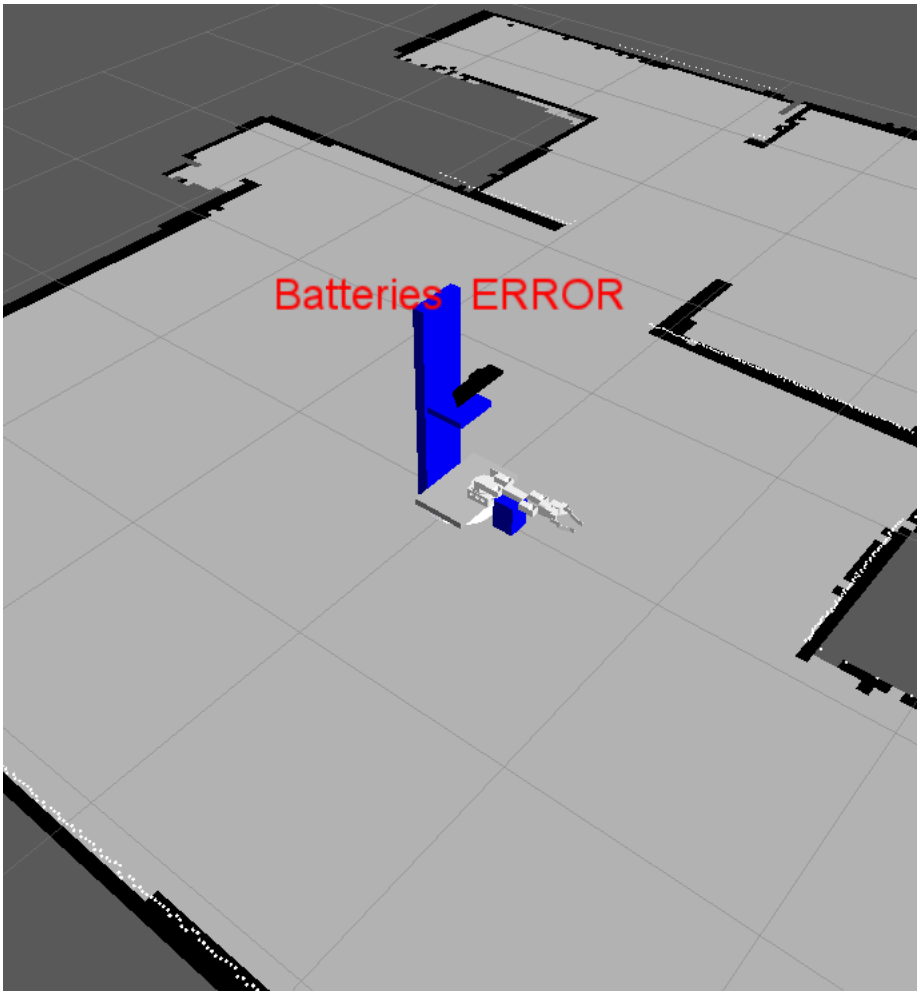
Príklad zobrazenia chybovej informácie v nástroji Rviz, spolu s modelom robota v cvičnom prostredí máme možnosť vidieť na obrázku 5.7.

5.4 Výsledná aplikácia

Po spojení všetkých modulov a funkcií spomínaných v časti 5.3 sme vytvorili jadro požadovanej aplikácie. Aplikácia po zistení konfigurácie z launch súboru `btb_diag.launch`, prostredníctvom uzla `diagnostic_collector` čerpá dáta z `/diagnostic` topiku, pseudosúboru `/proc/net/wireless` a kontrolného terminálu `/dev/ttyUSB3`. Spracuje ich a následne pošle v upravených formách na štandardný výstup príkazového riadku a znovu za pomoci uzla `diagnostic_collector` na topiky `/visualization_marker` a `/diagnostic_agg`, odkiaľ ich preberú a zobrazia nástroje Rviz respektíve Robot Monitor.

Naša implementácia sa od návrhu výraznejšie líši v dvoch častiach:

- Množstvo topikov – výsledná aplikácia čerpá dáta len z jedného topiku, pričom v návrhu bolo plánované zahrnúť ich viacero.
- Zobrazované detaily – miesto pôvodného návrhu zobraziť len určité detaily ku každej monitorovanej časti, sú zobrazované všetky detaily.



Obrázok 5.7: Príklad zobrazenia výstupu aplikácie vo vizualizátore Rviz

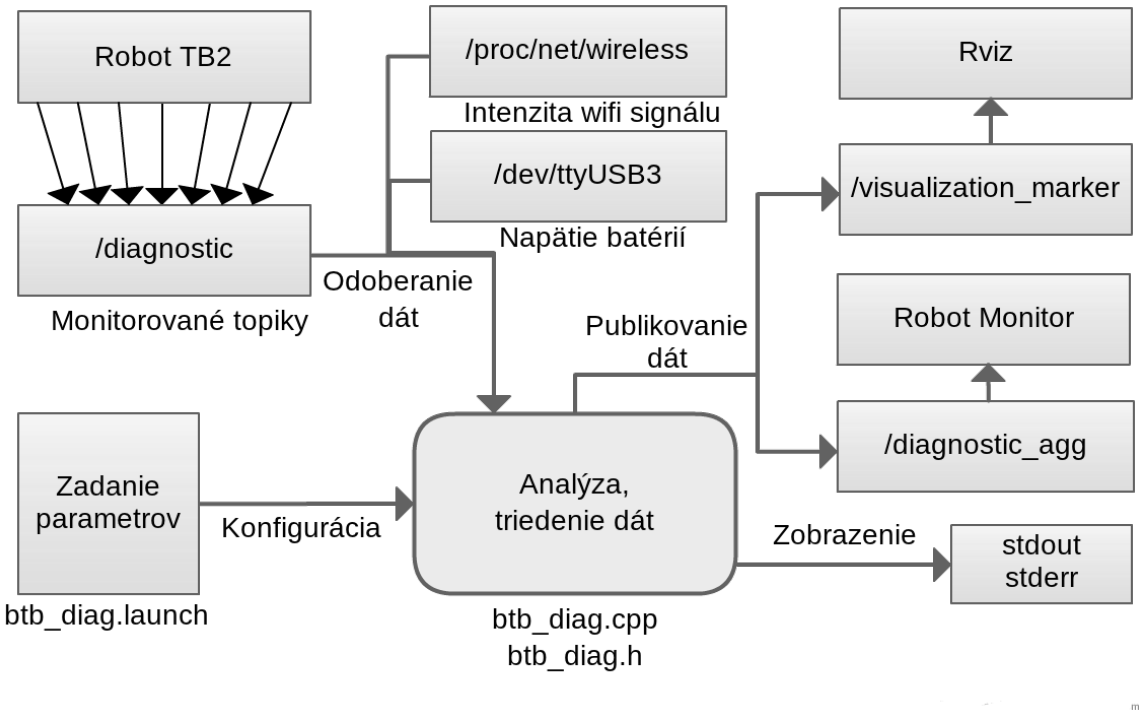
Graf znázorňujúci štruktúru výslednej implementovanej aplikácie je zobrazený na obrázku 5.8.

5.4.1 Testovanie

Testovanie počas implementácie aplikácie aj po jej dokončení prebiehalo dvoma spôsobmi:

- Pomocou simulátora – simulátor Gazebo (viď. 3.1.3), poskytoval základné možnosti pre testovanie aplikácie. Chýbala však podpora zobrazovania napätia v batériách.
- Pomocou reálneho robota – poskytoval možnosť plného testovania aplikácie. Ak nebol k dispozícii, využili sme záznamy vytvorené pomocou nástroja Rosbag (viď. 3.1.3).

Počas testovania boli sledované informácie zobrazené na štandardnom výstupe príkazového riadku, v simulátore Rviz aj prostredníctvom Robot Monitora. Simulované boli niektoré základné chyby komponentov a u niektorých vybraných bol skúmaný aj ich detailný stav.



Obrázok 5.8: Výsledná štruktúra implementovanej aplikácie.

Kapitola 6

Možné rozšírenia

Po implementácii a otestovaní našej aplikácie sme dospeli k niekoľkým možnostiam ako túto implementáciu vhodným spôsobom rozšíriť. Niektoré z nich by mohli určiť možný smer budúceho vývoja.

V nasledujúcej kapitole si v stručnosti tieto možnosti popíšeme.

6.1 Úplná konfigurácia aplikácie

Možnosť rozsiahlejšej konfigurácie pomocou launch súboru bola už niekoľkokrát spomínaná. V tomto prípade by sa však jednalo o kombináciu všetkých zmieňovaných možností. Pomocou konfiguračného launch súboru by teda užívateľ mal možnosť ovplyvniť: ktoré komponenty sledovať, ktoré z ich podrobných informácií sledovať, ako informácie zobraziť (ktorým nástrojom), upraviť zoznam sledovaných topikov, zvoliť zoznam výstupných topikov či špecifikovať niektoré druhy zobrazenia (napr. pomenovanie úrovní intenzity wifi signálu, stanovenie ich hraničných hodnôt, určenie parametrov markra pre Rviz – farba, veľkosť, poloha...).

6.2 Vlastné grafické užívateľské rozhranie

Keďže naša aplikácia pre zobrazovanie dát používa štandardný výstup príkazového riadku alebo aplikácie tretích strán, bolo by vhodné vytvoriť vlastné grafické užívateľské rozhranie. To však nemusí znamenať nutnosť zrušiť publikovanie dát na topiky pre pôvodné formy zobrazenia.

Pomocou tohoto GUI by mohla byť ďalej umožnená konfigurácia aplikácie aj za behu. Užívateľ by si mohol napríklad dynamicky voliť ktoré topiky sledovať, prípadne kam informácie odosielať.

6.3 Experimentovanie s Androidom

Na základe faktu, že pre mobilný operačný systém Android existuje prerobené celé jadro ROSu, máme k dispozícii hneď niekoľko možností:

- Aplikácia zobrazujúca dáta – vytvoriť aplikáciu podobného typu ako Robot Monitor, akurát pre systém Android. Dokázala by čerpať dáta z topiku, na ktorý by ich naša aplikácia publikovala, vedela by ich spracovať a zobraziť.

- Android aplikácia – prerobiť kompletne celú aplikáciu s rovnakou funkčnosťou pre systém Android.
- UDP aplikácia – upraviť našu aplikáciu pre odosielanie dát cez sieť, napríklad prostredníctvom UDP packetov. Následne vytvoriť aplikáciu na Android, ktorá by nemala s rosom nič spoločné. Vedela by prijímať posielané UDP packety, spracovať ich a zobraziť ich formou grafického užívateľského rozhrania.

Kapitola 7

Záver

Zámerom tejto práce bolo vytvoriť diagnostický balíček pre mobilného robota TB2. Práca spočívala v naštudovaní problematiky, zoznámení sa s časťami robota a jeho operačným systémom, ďalej v navrhnutí a implementácii aplikácie spolu s premyslením možných rozšírení.

Pri návrhu sme sa zamerali na splnenie všetkých bodov zadania a jednoduchosť aplikácie ako z pohľadu koncového užívateľa, tak pri implementácii.

V rámci implementácie bola vytvorená aplikácia v jazyku C++, ktorá na základe požiadaviek užívateľa získava informácie o stave zadaných častí robota TB2, ako z topikov (`/diagnostics`), tak z iných zdrojov (pseudosúbor `/proc/net/wireless`, kontrolný terminál `/dev/ttyUSB3`), spracováva ich a poskytuje naspäť užívateľovi. Pre zobrazenie používa nástroje Rviz, Robot Monitor a štandardný výstup príkazového riadku.

Medzi zaujímavé možnosti rozšírenia rozhodne patrí tvorba vlastného grafického užívateľského rozhrania, poskytujúceho možnosti konfigurácie za chodu programu a lepšiu kompatibilitu s vytvorenou aplikáciou ako nástroje tretích strán. Inou možnosťou by bolo preniesť časť zobrazovania dát na mobilný operačný systém Android.

Literatura

[1] http://www.generationrobots.com/ros-robot-operating-system,us,8,74.cfm.

[2] Bartle, P.: The Nature of Monitoring and Evaluation [online]. http://cec.vcn.bc.ca/cmp/modules/mon-wht.htm, 2011-09-30 [cit. 2013-05-15].

[3] Brukker, G.; Opatíková, J.: Veľký slovník cudzích slov [online]. http://www.voltaire.netkosice.sk/docs/vscs.pdf, 2006 [cit. 2013-05-15].

[4] Gertler, J. J.: *Fault detection and diagnosis in engineering systems*. Marcel Dekker Incorporated USA, 1998, iSBN 0-8247-9427-3.

[5] Peňázová, M.: Když se řekne robot [online]. http://www.automatizace.cz/article.php?a=2194, 2008 [cit. 2010-06-24].

Příloha A

Obsah CD

Na priloženom CD sa nachádzajú zdrojové kódy aplikácie zabalené v adresári `btb_diag`, programová dokumentácia v adresári `btb_diag/doc` a tento text uložený v pdf.