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

UŽIVATELSKÉ ROZHRANÍ PRO ŘÍZENÍ POZEMNÍHO ROBOTA VE VENKOVNÍM PROSTŘEDÍ

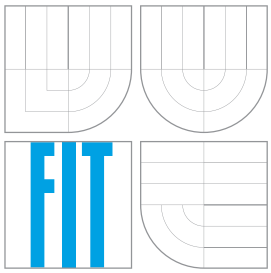
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PETER PAVLENKO

BRNO 2014

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

UŽIVATELSKÉ ROZHRANÍ PRO ŘÍZENÍ POZEMNÍHO ROBOTA VE VENKOVNÍM PROSTŘEDÍ

USER INTERFACE FOR GROUND VEHICLE CONTROL IN OUTDOOR ENVIRONMENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — PETER PAVLENKO
AUTHOR

VEDOUCÍ PRÁCE — Ing. BERAN VÍTĚZSLAV, Ph.D.
SUPERVISOR

BRNO 2014

Abstrakt

Tato bakalařská práce se zabývá návrhem aplikace s užívatelským rozhraním pro řízení pozemního robota ve venkovním prostředí. Teoretická část pojednáva o tvorbě aplikací pro android a přibližuje způsob řízení robota díky Robot Operating System. Hlavní částí práce je návrh a implementace aplikace schopné navigovat a řídit robota. V závěru je popsáno testování správnosti návrhu užívatelského rozhraní.

Abstract

This bachelor thesis includes design and technical aspects of GUI specific focused on remote control of ground robot in outside enviroment. The theoretical part deals with creating applications for the android platform and it takes a closer look at method of controlling the robot using the Robot Operating System. Main part of this thesis is the design and the implementation of application capable of navigation and control the robot. In conclusion you can find the testing done to verify validity of this graphic user interface.

Klíčová slova

aplikace, android, užívatelské rozhraní, robot, ROS, pioneer, navigace

Keywords

application, android, user interface, robot, ROS, pioneer, navigation

Citace

Peter Pavlenko: Uživatelské rozhraní pro řízení pozemního robota ve venkovním prostředí, bakalářská práce, Brno, FIT VUT v Brně, 2014

Uživatelské rozhraní pro řízení pozemního robota ve venkovním prostředí

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Vítězslava Berana Ph.D.

.....
Peter Pavlenko
20. května 2014

Poděkování

Rád bych poděkoval Ing. Vítězslavu Beranovi Ph.D. za pomoc, cenné rady a přátelský přístup při odborných konzultacích. Také chci poděkovat Ing. Zdeňku Maternovi za vstřícnost a za pomoc s robotem. Děkuji také Andrejovi Tichému za spolupráci při návrhu a implementaci rozhraní robota.

Obsah

Kapitola 1

Úvod

Dnešná doba priniesla obrovský nárast využívania mobilných technológií na všemožné úlohy. Už dávno sa mobilné zariadenia nevyužívajú iba na telefonovanie a SMSkovanie. Dnes je úplne bežné, že takéto zariadenia nahrádzajú kamery, fotoaparáty, organizéry, slovníky, televíziu a mnoho ďalších. Je to spôsobené najmä tým, že tieto zariadenia obsahujú mnoho rôzných komponentov umožňujúcich všemožné vstupy a výstupy zariadenia. Každým dňom tiež rastie ich výkon čo umožňuje vývojárom tvoriť stále zložitejšie a zaujímavejšie aplikácie bez toho aby si užívatelia všimli nejaký pomalší chod týchto aplikácií.

Preto som sa rozhodol sa vo svojej bakalárskej práci tiež venovať tvorbe aplikácie pre mobilné zariadenie a to konkrétne pre zariadenie so systémom android. Po konzultácií s vedúcim som si vybral konkrétne vytvoriť aplikáciu s užívateľským rozhraním pre ovládanie a navigáciu robota vo vonkajšom prostredí.

V druhej kapitole tejto práce sa budem venovať zozbieraniu teoretických poznatkov ohľadom tvorby aplikácií. Priblížim potrebné nástroje, vývojové prostredia a postupy, ktoré by sa pri tvorbe mali dodržiavať. Ďalej tiež poukážem na problémy vznikajúce pri tvorbe užívateľského rozhrania pre mobilné zariadenie a ich najbežnejšie riešenia. V závere kapitoly zasa priblížim robota, ktorého bude možné danou aplikáciou ovládať a tiež Robot Operating System využívaný pre ovládanie robotov.

Ďalšia, teda tretia kapitola je venovaná rozboru zadania práce, a návrhu jednotlivých komponentov riešenia. Medzi tieto komponenty patrí napríklad návrh užívateľského rozhrania, aplikácie, spôsobu komunikácie s robotom a uzla ovládajúceho pripojeného robota. V tejto kapitole je tiež spomenutý návrh jednoduchého simulátora daného robota.

Vo štvrej kapitole je popísana samotná realizácia, vrátane používaných nástrojov a taktiež testy zamerané na správnosť návrhu užívateľskkého rozhrania a samotné fungovanie aplikácie.

Kapitola 2

Teória

V tejto kapitole sa budem venovať hlavne zozbeiraniu a zhrnutiu teoretických znalostí nadobudnutých pri riešení tejto práce týkajúcich sa hlavne vývoja aplikácií pre platformu Android, tiež robota typu Pioneer a Robot Operating System (ROS) pre riadenie robota a komunikáciu. Kapitola je rozdelená na štyri podkapitoly zaoberajúce sa jednotlivými časťami riešenia.

V prvej časti sa budem venovať operačnému systému Android. Priblížim tento operačnéhý systém, spôsob vývoja aplikácii, Google Maps API a nakoniec spôsob publikácie aplikácie v obchode Google Play.

Druha časť je venovaná tvorbe užívateľských rozhraní pre Android a bežným problémom, ktoré pri návrhu nastávajú.

Tretia časť sa zaoberá robotom typu Pioneer a tiež niektorími senzormi, ktoré može daný robot používať.

V poslednej časťi predstavím Robot Operating System (ROS), spôsob práce s ním a objasním niektoré základné pojmy, ktoré budem používať v nasledujúcich kapitolách.

2.1 Android

Cieľom práce je návrh aplikácie pre zariadenia s operačným systémom android, preto v nasledujúcej časti priblížim samotný operačný systém a nástroj potrebné pre tvorbu aplikácií na ňom bežiacích.

Android je najobľúbenejším, najrozšírenejším a najrýchlejšie sa rozvíjajúcim operačným systémom v dnešných mobilných zariadeniach. K rozšírenosti určite prispelo, že jeho zdrojové kódy sú voľne dostupné a je navrhnutý tak, aby podporoval širokú škálu rôznych typov hardwaru.

Google tiež prináša kvalitný nástroj na tvorbu aplikácií s názvom Android Software Develpoer Kit, skrátene Android SDK, ktorý automaticky optimalizuje užívateľské rozhranie pre rôzne zariadenia s rôzným rozlíšením obrazovky. Spomenuté nástroje sú popísané v kinhe [13].

Android SDK

Android software development kit obsahuje roziahlú škálu nástrojov pre vývojárov aplikácii pre android. Medzi tieto nástroje patrí debugger, potrebné knižnice, emulátor, dokumentácia, jednoduché ukážky kódu a taktiež návody. Android SDK podporuje operačné systémy Linux(akákoľvek moderná distribúcia), Mac OS X (10.5.8 alebo novšie) a Windows (XP

alebo novšie). Oficiálne podporovaným integrovaným vývojovým prostredím(IDE) je Eclipse používajúce Android Development Tools Plugin, ale NetBeans je tiež podporovaný prostredníctvom zásuvného modulu. Podmienkou funkčnosti SDK je Java Development Kit(JDK).

Android Debug Bridge (ADB)

Android Debug Bridge (ADB) je jedným z nástrojov Android SDK určeným na odstaňovanie chýb aplikácie. Obsahuje klientskú aj serverovú časť, ktoré dokážu komunikovať s opačnou časťou v zariadeni. Najčastejšie sa k nemu pristupuje pomocou príkazoveho riadku, ale existujú aj grafické nadstavby.

Štruktúra príkazu: `adb [-d|-e|-s <serialNumber>] <command>`

Emulátor

Pre jednoduchšie ladenie aplikácii bez nutnosti pripojenia fyzického zariadenia je možné vytvoriť v počítači virtuálne mobilné zariadenie so zvolenými parametrami (veľkosť displaya, procesor, RAM,...) a verziou operačného systému (2.2, 2.3.3, 4.1,...).

Obrázek 2.1: Ukážka emulátora

Eclipse

Je to open source vývojové prostredie určené predovšetkým na programovanie v jazyku Java. Jeho najväčšiou výhodou je rozšíriteľnoť pomocou zásuvných modulov. Pre vývoj aplikácie je to konkrétne ADT zásuvný modul. Po nainštalovaní modulu umožňnuje eclipse

jednoducho vytvoriť "Projekt Android Aplikácie"a pomocou sprievodcu nastaviť prvotné parametre(vzhľad, štýl,...).

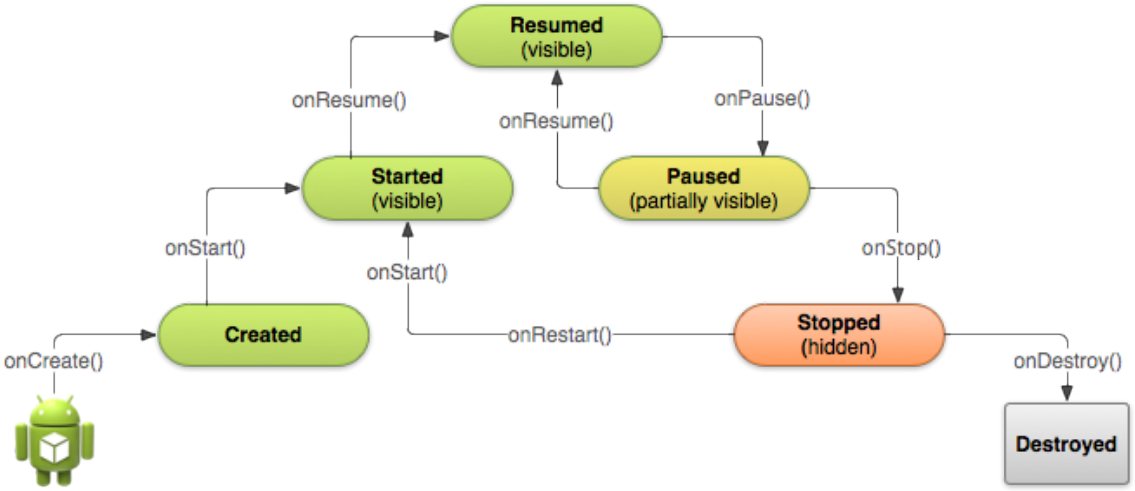
Android Studio

Android Studio je nové vývojové prostredie založené na myšlienkach IntelliJ. Je podobné Eclipsu s nainštalovaným ADT zásuvným modulom. Má integrované všetky nástroje pre vývoj aplikácie pre android, ktoré môže vývojár očakávať od IntelliJ. Poskytuje napríklad výstavby programu na báze Gradle, refactoring a rýchle opravy kódu na základe špecifikácií androidu a Lint nástroje pre zachytenie problémov s výkonom, použiteľnosťou a tiež kompatibility.

Vývoj aplikácie

Narozdiel od bežného paradigmatu, kde sú aplikácie spúšťané s metódou main(), spúšťa Androidový systém inštanciu Aktivity vyvolávajúcu metódy korešpondujúce s hlavnými udalosťami v jej životnom cykle.

Životný cyklus aktivity sa prirovnáva k pyramíde jednotlivých stavov. V každom stave existuje metóda, ktorá ho mení na stav umiestnený bližšie k vrcholu. Vrchol pyramídy predstavuje stav, kedy aktivita beží na popredí a užívateľ s ňou môže pracovať.



Obrázek 2.2: Životný cyklus aktivity. Prevzaté z [2]

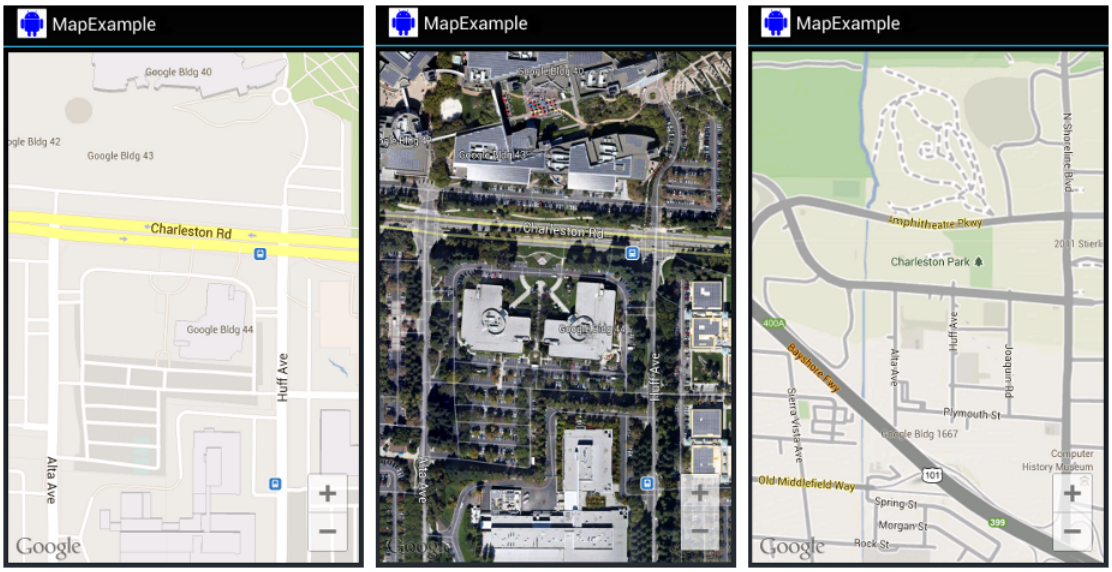
Implementácia týchto metód nieje vo všetkých prípadoch nutná, ale výrazne napomáha správnemu chodu vo všetkých štandardných situáciach, ktoré môžu nastať za behu aplikácie napríklad pri prijatí hovoru počas behu aplikácie.

Hlavné stavy aktivity:

- Resumed - Aktivita beží na popredí komunikuje s užívateľom.
- Paused - Aktivita je z časti zakrytá inou aktivitou na popredí a nedostáva žiadne vstupy.
- Stopped - Aktivita je spustená na pozadí, ale nemôže spúšťať kód.

Google Maps API

Je to rozhranie umožňujúce programátorom využívať vo svojich aplikáciách mapy poskytované Googlom a rôzne interakcie s nimi. Momentálne je dostupné vo verziách pre Android, webové aplikácie a IOS. Umožňuje používať rôzne verzie máp, pridávanie objektov, prácu s GPS súradnicami a ďalšie.



Obrázek 2.3: Rôzne typy máp. Prevzaté z [1]

Pre prácu s Google Maps API je nutné stiahnúť a nakonfigurovať Google Play Services SDK pomocou Android SDK. Každý projekt využívajúci toto rozhranie by mal mať Android certifikát a Google Maps API kľúč, ktorý je jedinečný pre každého vývojára, alebo aplikácu. Kľúč je zadarmo a dá sa získať po registrácii projektu na `https://cloud.google.com/console/project`. Po registrácii projektu je taktiež možné sledovať štatistiky využívania danej API v projekte. Podrobný popis registrácie a využitia API je na [1].

Kľúč je v podstate skrátenou formou digitálneho certifikátu aplikácie. Je to textový reťazec vytvorený SHA-1 hashovacým algoritmom z certifikátu.

Ukážka kľúča: `AIzaSyBdVl-cTICSwYKrZ95SuvNw7dbMuDt1KG0`

Publikovanie aplikácie

Google umožňuje publikovať aplikácie vytvorené pre platformu android na nato určenom internetovom obchode s názvom Google Play.

Príprava na publikáciu pozostáva z:

- Konfigurácie aplikácie na vydanie.
- Vytvorenie verzie vhodnej na vydanie.
- Testovanie tejto verzie.
- Príprava potrebných zdrojov.
- Príprava vzdialených serverov a služieb, na ktorých aplikácia závisí.

2.2 Užívateľské rozhranie (UI) pre Android

Úlohou je navrhnúť správne užívateľské rozhranie pre zariadenia s androidom. Android je podporovaný rozsiahlou škálou zariadení s rôznými vlastnosťami, preto musí aj návrh užívateľského rozhrania zohľadniť všetky druhy zariadení, na ktorých sa bude daná aplikácia spúšťať. Zariadenia sa môžu rozlišovať vo veľkosti obrazovky, hustote pixelov, pomere strán obrazovky a orientácii. Android však poskytuje API, ktoré umožní kontrolovať všetky tieto rozličné podmienky. Správne navrhnutá aplikácia poskytuje užívateľovi rovnaké možnosti na rozličných zariadeniach, no rozloženie jednotlivých prvkov užívateľského rozhrania nemusí byť rovnaké. Doporučenia pre správny návrh UI ako aj nasledujúce vlastnosti sú popísané v [3].

Dôležité vlastnosti obrazovky

Keďže navrhujem UI tak sa zameriam len na vlastnosti obrazovky a ostatné rozdiely medzi zariadeniami ponechám nespomenuté.

- **Veľkosť obrazovky**

  Je to fyzický rozmer uhlopriečky obrazovky meraný najčastejšie v palcoch. Android združil podobné veľkosti obrazovky do štyroch kategórií a to: small, normal, large a extra large.

- **Hustota pixelov(Screen density)**

  Hovorí o tom koľko je na danej obrazovke pixelov na jednotku dlžky. Najčastejšie sa používa počet bodov na palec(DPI). Zariadenia s vyšším DPI dokážu zobraziť na rovnako veľkej obrazovke viac informácií ako zariadenia s nižším DPI. Android podobne ako u veľkosti obrazovky delí zariadenia na štyri kateórie: low, medium, high a extra high.

- **Orientácia**

  Orientácia nieje priamo vlastnosť zariadenia, ale je rovnako dôležitá pri návrhu UI ako ostatné vlastnosti. Je to natočenie zariadenia voči užívateľovi a určuje šírku a výšku obrazovky.

- **Rozlíšenie**

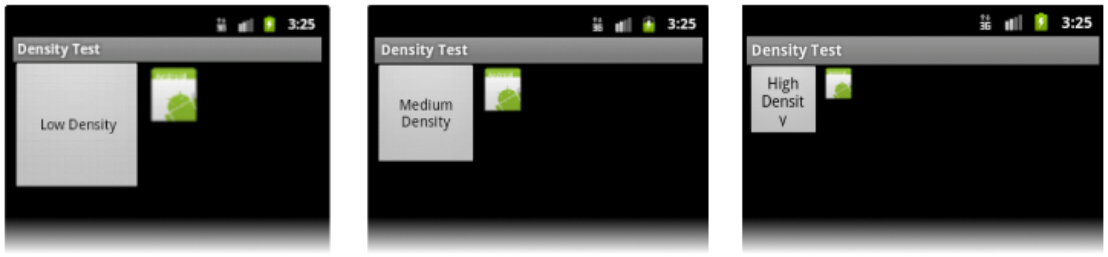
  Vyjadruje celkový počet pixelov na obrazovke a zapisuje sa ako šírka krát výška.

Nezávislosť na hustote obrazovky

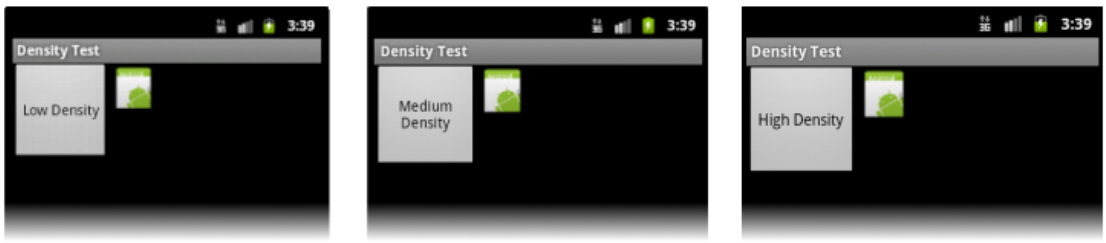
Aplikácia je nezávislá na hustote pokiaľ zachováva z pohľadu užívateľa, prvky užívateľského rozhrania nezmenené na obrazovkách s rozličnou hustotou.

Zachovanie nezávislosti na hustote je dôležité, hlavne preto lebo prvok užívateľského rozhrania (napríklad tlačidlo) sa zobrazí väčší na obrazovke s nižšou hustotou a menší na obrazovke s vyššou hustotou. Tieto zmeny veľkostí môžu spôsobiť problémy s rozvrhnutím aplikácie a následnou použiteľnosťou.

Systém Android pomáha pri vytváraní na hustote nezávislých aplikácií dvoma spôsobmi. Zaviedol pixel nezávislý na hustote (dp), ktorý sa prispôsobí práve používanej hustote obrazovky. Systém tiež upravuje veľkosti bitmapových zobrazovaných častí na základe hustoty obrazovky ak je to nevyhnutné. Tieto techniky je vidieť na obázkoch 2.2 a 2.2.



Obrázek 2.4: Príklad aplikácie závislej na hustote pixelov. Prevzaté z [3].



Obrázek 2.5: Príklad aplikácie nezávislej na hustote pixelov. Prevzaté z [3].

Zásady pre správnu tvorbu rozhrania na Adnroid

Aj keď android sa snaží prispôsobiť zobrazovanie užívateľského rozhrania na základe na hustote nezávislého pixelu je vhodné dodržiavať niektoré zásady, aby bolo zobrazovanie vždy správne.

- Explicitne deklarovať v manifeste, ktoré veľkosti obrazoviek aplikácia podporuje. To zaručí, že si aplikáciu môžu stiahnúť iba podporované zariadenia a u zariadení s väčšou obrazovkou to ovplivní vykreslenie aplikácie v režime obrazovkovej kompatibility.
- Vytvárať rôzne návrhy dispozície pre rôzne veľkosti obrazovky. Čo je v niektorých prípadoch vhodné použiť pre užívateľský prívetivejšie zobrazenie danej aplikácie.
- Mať vytvorené rôzne verzie bitmapových obrázkov pre rôzne hustoty obrazovky. Ak sa používa iba jedna verzia obrázku tak ju systém prispôsobuje aktuálnej hustote tým, že obrázok buď zväčší alebo zmenší a to môže spôsobiť neostré zobrazenie.

2.3 Toad

Toad je názov robota typu Pioneer, ktorý sa nachádza na Fakulte Infromačných Technológií a moja práca je určená práve pre neho. V tejto časti opíšem všeobecne robota typu Pioneer a potom senzory ktoré sú na Toadovi nainštalované a pri práci budem využívať ich dáta. Tieto senzory sú GPS modul, ktorého dáta sú upravované Kalmanovim filtrom, a Kinect, ktorý budem používať ako zdroj obrazových dát.

Robot typu Pioneer

Pioneer je označenie pre rodinu dvojkolesových aj štvorkolesových robotov zahrňujúcu Pioneer 1, Pioneer AT, Pioneer 2 -DX, -AT- CE atď. Najnovším je Pioneer 3 -DC/-AT.

Roboty tohto typu sú inteligentné mobilné zariadenia snímajúce a navigované hlavne vo vonkajšom prostredí a sú používane v množstve výskumných projektov zahrnujúcich aj niektoré projekty financované US Defense Advanced Research Projects Agency (DARPA). Informácie prevzaté z [7].

Robot Pioneer 3-AT je univerzálny štvormotorový robot s pohonom všetkych štyroch kolies vhodný do vonkajšieho prostredia. Pioneer 3-AT je výkonný, ľahko ovládateľný a flexibilný robot vhodný aj do náročnejšieho terénu. V základnej verzii obsahuje batériu, núdzový vypínač a mikrokontrolér s ARCOS firmware. Informácie z [6].

Obrázek 2.6: Obrázok robota typu Pioneer (Toad), ktorý sa nachádza na Fakulte Informačných Techológií. Prevzaté z [8].

GPS

GPS alebo Globalný pozičný systém je satelitný navigačný systém používaný na zisťovanie presnej polohy takmer všade na Zemi, alebo na zemskej orbite nezávisle na počasí 24 hodín denne. Používa minimálne 27 satelitov na zemskej orbite a je to pasívny družicový dlžkomerný systém.

GPS určuje polohu bodu na základe priesečníkov guľových plôch, ktorých polomer je daný meranými vzdialenosťami. Meria sa doba šírenia rádiového signálu medzi GPS satelitom a anténou GPS príjmača. Rýchloť šírenia signálu je rovná rýchlosti svetla. Družica v správe posiela tiež plán svojej dráhy, takže sa dá presne vypočítať jej aktuálna poloha [12].

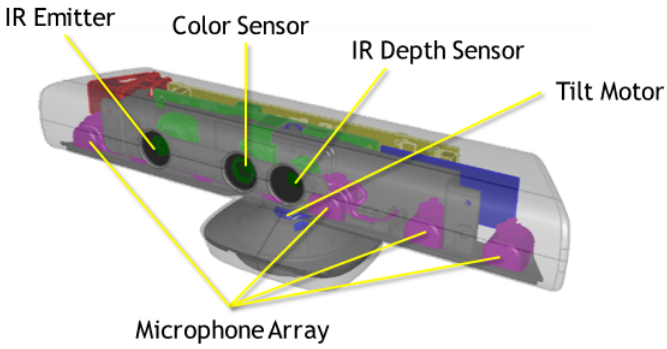
Kalmanov filter

Je to systém, ktorý vie odhadnúť premenné v širokom spektre procesov. Matematicky povedané Kalmanov filter odhaduje stavy lineárneho systému a minimalizuje variáciu chyby odhadu. U tohto robota ho používa GPS príjmač, aby potlačil odchylky určovania polohy a získaval tak presú polohu robota v danom čase [11].

Kinect

Kinect je zariadenie vyvýjané firmou Microsoft pre konzoly Xbox. Zariadenie obsahuje RGB CMOS kameru, hĺbkovy senzor a tiež multipoľový mikrofón. Tieto senzory umožňujú zachytávanie 3D scény a pohybu, rozpoznávanie tváre a hlasu. RGB kamera obsahuje senzor

s rozlíšením 640x480 a produkuje snímky rýchloťou 30fps. Infračervená kamera má rozlíšenie 1280x1024, ale produkuje výstup 640x480 rýchlosťou 30fps [9].



Obrázek 2.7: Zariadenie kinect. Obrázok prevzatý z [10].

2.4 Robot Operating System (ROS)

Pri práci s robotom budem potrebovať ovládať funkcionalitu robota. Na to slúžia rôzne frameworky, či operačné systémy. Ja som si vybral Robot Operating system hlavne z toho dôvodu, že na Toadovi bol už pripravený a vyskúšaný viacerími predchádzajúcimi projektami. Popíšem a vysvetlím princíp fungovania a hlavne základné pojmy používané ako bežne pri práci s ROS tak aj v tomto dokumente.

Robot Operating System (ROS) je voľne šíriteľný meta-operačný systém pre robota. Poskytuje služby, ktoré by programátor očakával od bežného operačného sytému, a to napríklad abstrakciu hardwaru, nízkoúrovňove ovládanie zariadení, implementáciu bežne používanej funkcionality, zasielanie správ medzi procesmi a tiež prácu s balíkmi. Taktiež poskytuje nástroje a knižnice pre získavanie, vytváranie, prekladanie a spúštanie kódu naprieč viacerími počítačmi. V niektorých ohľadoch je podobný "robot frameworkom"ako napríklad Player, YARP, Orocos, CARMEN a ďalším.

ROS "Graf"behu je peer-to-peer sieť procesov spojených používaním ROS komunikčnej infraštruktúry. ROS implementuje niekoľko rozličných druhov komunikácie pozostávajúcich z asynchrónneho RPC-štýlu pomocou služieb, asynchrónneho streamingu dát prostredníctvom topicu a ukladanie dát v Parametrovom servery [5].

Uzol

Uzol je proces vykonávajúci výpočet. ROS je navrhnutý tak aby bol čo najviac modulárny. Kontrolný systém robota pozostáva z množstva uzlov, napríklad jeden uzol pre ovládanie motorov, ďalší pre spracovávanie GPS súradníc a ďalší pre plánovanie trasy. Na vytváranie uzlov sa využívajú klientske ROS knižnice ako napríklad roscpp alebo rospy.

Správa

Uzly medzi sebou komunikujú zasielaním správ. Správa je jednoduchá datová štruktúra pozostavajúca zo zložiek určitého typu. Podporuje štandardné jednoduché typy ako integer, float, boolean atď. Tieto typi sú podporované tiež v poliach. Správy tiež môžu obsahovať

ľubovoľne vnorené štruktúry a polia podobne ako v jazyku C. Vytvorené neštandardné správy sú uložené v priečinku msg daného ROS balíčka.

Príklad neštandardnej správy:

```
int64 num
String name
float32MultiArray data
```

Topic

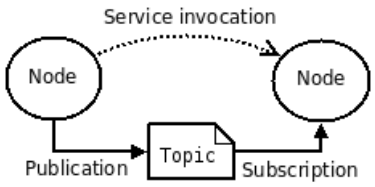
Správy su smerované cez transportný systém so semantikou na publikovanie aj odoberanie. Uzol zasiela správu publikovaním do topicu. Topic je meno, ktoré sa používa na identifikáciu obsahu správy. Uzol, ktorý má záujem o dané dáta bude odoberať zo zodpovedajúceho topicu.

Môže existovať viacero konkurentne publikujúcich ako aj odoberajúcich uzlov z jedného topicu a jeden uzol môže tiež publikovať či odoberať z viacerých topicov. Uzly vôbec netušia o existencii ostantých uzlov. Hlavnou myšlienkou je oddeliť produkovanie informácií od ich spotreby.

Služba

Model správ je veľmi flexibilný, ale stále je to hromadná komunikácia, ktorá nieje vhodná pre interakcie založené na požiadavke a odpovedi potrebné v niektorých distribuovaných systémoch. Systém požiadavky a odpovede umožnujú práve služby. Služby sú definované ako dvojica správ, jedna pre požiadavku a druhá pre odpoveď. Uzol poskytuje službu pod nejakým menom podobne ako u topicu. Klient požiada o službu zaslaním požiadavky a potom čaká na odpoveď. Služby sa správajú podobne ako vzdialené volanie procedúr. Vo vytvorenej službe sa oddeľuje správa požiadavky od správy odpovede pomocou troch mínus. Vytvorené neštandardné služby sú uložené v priešinku srv daného ROS balíčka. Príklad vytvorenej služby:

```
float64 x
float64 y
---
float64[] x
float64[] y
```



Obrázek 2.8: Znázornenie správy a služby. Obrázok prevzatý z [4].

Master

ROS Master zodpovedá za registráciu mien a dohliada na zvyšok výpočtového grafu. Bez Mastera by uzly neboli schopné nájsť jeden druhého, vymienať si správy, ani prevádzkovať služby.

Bag

Bag je formát pre ukladanie a opetovne prehrávanie správ. Je to veľmi dôležitý mechanizmus pre ukladanie dát, ako sú napríklad dáta zo senzorov, ktoré sú ťažko zozbierateľné, ale sú dôležité pre testovanie a vývoj.

RosJava

RosJava je experimentálna klinetská knižnica zabezpečujúca mnohé nástroje a ovládače potrebné pre vytváranie uzlov, správ, služieb atď. podobne ako u roscpp, roslisp alebo rospy, ktoré sú hlavnými knižnicami ROSu.

Kapitola 3

Návrh

V tejto kapitole sa pokúsim analyzovať zadanie, navrhnúť aplikáciu so všetkými komponentami potrebnými na to aby vyhovovala zadaniu a taktiež navrhnúť správanie a postup pri realizácii danej aplikácie. Kapitolu som rozdelil na štyri časti.

V prvej časti sa venujem analýze zadania, predstave konečnej aplikácie a tiež dekompozícii zadania na jednotlivé problémy, ktoré je treba vyriešiť aby konečná aplikácia správne fungovala a spĺňala zadané požiadavky.

V druhej časti je už samotný návrh najprv celkového riešenia s vymenovaním potrebných celkov, a potom návrh realizácie aplikácie, užívateľského rozhrania a komunikačných protokolov.

Tretia časť sa zaoberá návrhom programu bežiaceho na samotnom robotovi so správami a službami, ktoré prevádzkuje.

Posledná časť je venovaná popisu návrhu jednouchého simulátora, ktorý by bol schopný nahradiť robota.

3.1 Analýza zadania

V tejto časti sa zameriam na analýzu zadania a problémy vznikajúce pri riešení tohoto zadania. Najprv načrtnem svoju prvotnú predstavu konečnej aplikácie s jej rozhraním so všetkými jej možnosťami. Budem sa snažiť definovať problémy, ktoré musia byť vyriešené aby aplikácia splňovala podmienky stanovené v zadaní. Nájdene problémy rozložím na podproblémy a ku každému podproblému sa pokusím nájsť najlepšie a najefektívnejšie riešenie.

Predstava hotovej aplikácie

Hotová aplikácia bažiaca na zariadení s Androidom po spustení umožní užívateľovi zadať URI robota poprípade nasnímať QR kód s danou URI. Po pripojení k robotovi mu dá na výber zobraziť režim mapy a plánovania trasy alebo ovládať robota na diaľku prostredníctvom šípok a prenášaného obrazu z kamery robota. V plánovacom režime musí zobraziť polohu užívateľa a polohu pripojeného robota. Užívateľ tiež môže zadať cieľ kam sa má robot presunúť a zvoliť naplánovanie trasy. Po naplánovaní sa trasa zobrazí a užívateľ môže prikázať robotovi aby sa po danej trase vydal. Užívateľ by mal mať stále možnosť zastaviť robota, alebo prepnúť na zobrazovanie stavu a manuálne riadenie.

Problémy vyplývajúce z požiadavkov na aplikáciu

Daná aplikácia musí bežať na rôzných zariadeniach s rôznymi rozlíšeniami a veľkosťami obrazovky. Z tohto vyplýva problém zaistiť aby zobrazenie bolo na rôznych zariadeniach rovnaké, alebo aby aspoň poskytovalo užívateľovi rovnaké možnosti. Ďalším problémom je navrhnúť intuitívne a príjemné užívateľské rozhranie, s ktorým by nenrobilo problémy pracovať ani laikovi. Aplikácia má ovládať robota prostredníctvom bezdrôtovej siete, čiže vzniká problém ako identifikovať robota v sieti, pripojiť sa k nemu a potom s ním komunikovať. Zadané príkazy sa musia vykonať u rôznych robotov rovnako takže potrebujeme vytvoriť rozhranie nezávislé od robota poskytujúce zber dát z robota a jeho riadenie. Kvôli tomu, že nebude možné mať stály prístup k robotovi bude taktiež potrebné vytvoriť jednoduchý simulátor daného robota, čo urýchly vývoj a uľahčí prácu i testovanie jednotlivých komponentov riešenia.

3.2 Návrh realizácie aplikácie

Táto časť sa zaoberá návrhom celkovej štruktúry riešenia a postuným rozborom a návrmi riešenia už spomýnaných problémov z analýzy zadania.

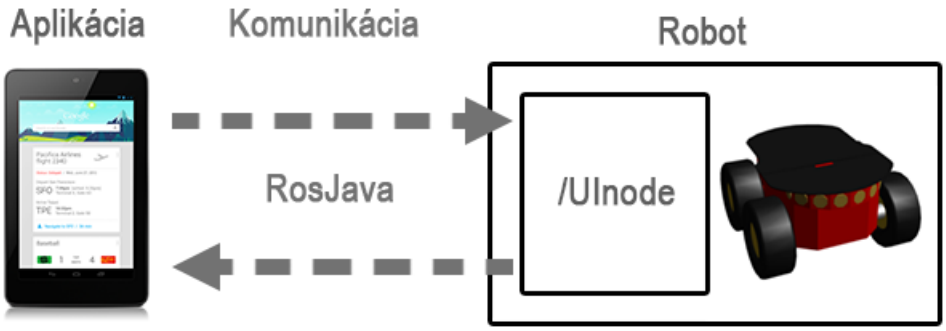
Celková štruktúra

Celkovú štruktúru môjho riešenia som rozdelil do troch najdôležitejších častí.

Prvá časť je aplikácia, ktorá bude reagovať na užívateľove pokyny a zobrazovať mu aktuálne informácie o stave robota. Je dôležité aby aplikácia mala intuitívne užívateľské rozhranie a aby sa s ňou pohodlne pracovalo. Aplikácia taktiež musí získať a zobraziť mapu, a polohu zariadenia.

Druhým dôležitým prvokm je komunikácia medzi robotom a aplikáciou. Komunikácia musí byť bezdrôtová, ale zase nieje podmienkou aby fungovala mimo lokálnu sieť. Ďalšou podmienkou je možnosť prenosu videa a rôznych typov dát.

Poslednou časťou riešenia je navrhnúť rozhranie, ktoré bude príjmať príkazy a dáta od aplikácie a na základe nich riadiť robota. Musí byť schopné fungovať na rôznych podobných robotoch. Rozhranie tiež bude zbierať a pripravovať dáta z robota tak aby nebolo potrebné upravovať aplikáciu.

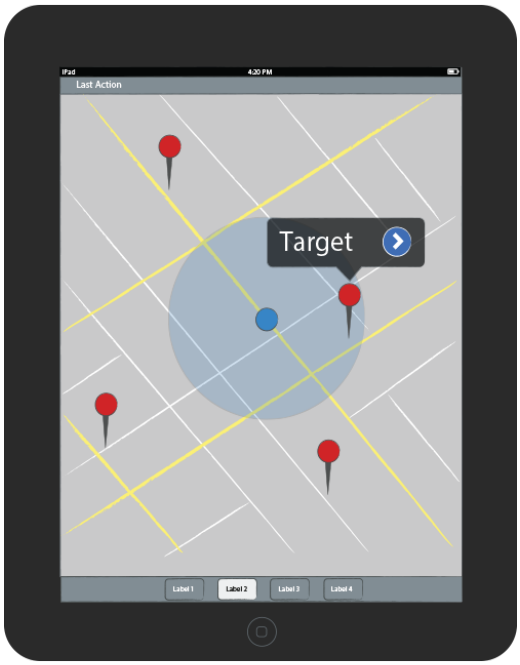


Obrázek 3.1: Štruktúra návrhu realizácie. Aplikácia komunikuje prostredníctvom rozhrania RosJava s uzlom v robotovi a ten riadi robota.

Užívateľské rozhranie

Keďže sa snažím narhnúť čo najjednoduchšie užívateľské rozhranie, rozhodol som sa aplikáciu rozdeliť na štyri hlavné aktivity.

Najdôležitejšou aktivitou je aktivita plánovania trasy. Hlavnou časťou aktivity pre plánovanie trasy je mapa. Na mape sa zobratuje poloha užívateľa a tiež poloha robota. Potom sú tam ovládacie prvky mapy ako zoom in a zoom out, kompas zobrazujúci natočenie mapy a tlačidlo poskytujúce možnosť vycentrovať mapu na užívateľa. Mapa tiež reaguje na dotyk a na danom mieste vytvorí značku pre cieľ trasy. Dlhším dotykom sa dá cieľ trasy posunúť na iné miesto. Nad mapou je umiestnená lišta so stavovým riadkom, ktorý informuje o poslednej aktivite užívateľa, či robota. Pod mapou sa nachádzajú tlačidla s možnosťami pre plánovanie, pohyb po trase, zmazanie trasy a zastavenie robota. Návrh rozloženia prvkov (wireframe) je zobrazený na obrázku 3.2.

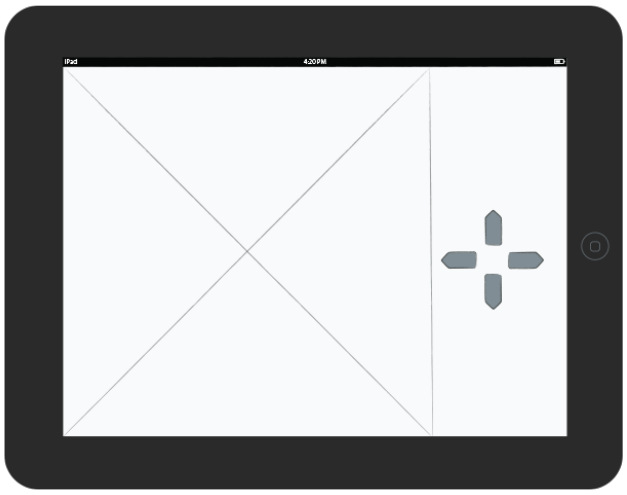


Obrázek 3.2: Náčrt obrazovky s plánovaním trasy.

Druhou najdôležitejšou aktivitou je aktivita zobrazujúca stav robota a umožňujúca ovládanie na diaľku. Hlavnou časťou je fragment zobrazujúci výstup z kamery pripojeného robota v reálnom čase. Vedľa toho sa nachádza popis stavu batérie a šípky pre ovládanie robota. Wireframe tejto aktivity je zobrazený na obrázku 3.2.

Prvá aktivita, ktorá sa užívateľovi zobrazí má na starosti pripojenie sa k robotovi. Obrazovka obsahuje miesto pre užívateľov vstup s MASTER URI robota ku ktorému sa plánuje pripojiť a tlačidlo pre pripojenie. Tiež sa tam nachádza aj tlačidlo pre prečítanie MASTER URI pomocou externej aplikácie na čítanie QR kódu vďaka čomu užívateľ nemusí prepísať MASTER URI ručne. Posledným tlačidlom je tlačidlo Cancel pre vypnutie aplikácie. Návrh rozloženia prvkov (wireframe) v tejto aktivite popisuje obrázok 3.2.

Po pripojení k robotovi sa zobrazí aktivita s možnosťami aplikácie, a to sú zobrazenie mapy a plánovania trasy alebo zobrazenie stavu robota.



Obrázek 3.3: Náčrt obrazovky so stavom robota a riadením.

Obrázek 3.4: Náčrt obrazovky s pripojením.

Mapa

Keďže je hlavnou časťou jednej z aktivít mapa rozhodol som sa využiť API pre prácu s mapou a najvhodnejšou voľbou pre mňa je Google Maps API, pretože implementuje mnoho funkcií ako napríklad približovanie a oddiaľovanie mapy, pridávanie značiek, kreslenie čiar.

Po spustení aktivity sa vykreslí cela oddialená mapa sveta. API získa GPS súradnice zariadenia a vykreslí ho na mapu. Aplikácia získava GPS súradnice polohy robota a tie v pravidelných intervaloch vykresľuje ako značku na mape. Mapa taktiež vyčkáva na stlačenie na nejaké miesto. Po stlačení si uloží polohu značky a vykreslí ju. Podobne ako na stlačenie na miesto čaká aj na dlhé stlačenie na značku a vtedy umožní jej posúvanie až pokiaľ stlačenie neskončí.

Po odoslaní cieľa na naplánovanie sa zmení farba značky cieľa a po obdržaní trasy v tvare zoznamu GPS súradníc bodov ktorými robot musí prejsť sa na mapu vykreslí celá trasa pospájaním jednotlivých bodov čiarami.

RosJava

Na komunikáciu medzi aplikáciou na androide a robotom som sa rozhodol použiť najprv klasickú socketovú komunikáciu pomocou BSD socketu. Tento spôsob však priesol pár problémov. Jedným z problémov je, že aplikácia je vytvorená v Jave a robot je ovládaný ROSom napísaným v C++ alebo Pythone. To môže spôsobiť napríklad nekompatibilitu typov. Ďalšou nevýhodou je nutnosť vytvorenia vlastného komunikačného protokolu a následovný preklad na oboch stranách na posielané príkazy, alebo dáta. Najväčšou nevýhodou však je obtiažny prenos a konverzia videa z kamery robota v reálnom čase.

Namiesto programovania celkovej komunikácie prostredníctvom BSD socketu som sa rozhodol použiť v aplikácii knižnicu RosJava. Táto knižnica využíva socketovú komunikáciu, ale odpadá tam nutnosť vytvárať vlastný protokol pretože dokáže komunikovať priamo s ROS uzlom v robotovi.

Spojenie robota s aplikáciou

Po zadaní MASTER URI robota a stlačení tlačidla propojiť sa aplikácia pripojí k robotovi na danú IP adresu a port ako ROS uzol. Akonáhle je aplikácia pripojená vytvorí inštanciu triedy pre uloženie konfigurácie a vykonávanie uzla. Daná trieda musí byť implementovaná ako jedináčik aby umožnila prístup k tomu istému uzlu aj ostatným aktivitám aplikácie. Od vytvorenia uzla už môže aplikácia komunikovať so všetkými uzlami vytvorenými v master ros jadre čiže u robota.

Príjmane správy

Keďže potrebujem z robota zísavať stav batérie, obraz z kamery, jeho GPS súradnice a naplánovanú trasu potrebujem pre každú zložku vytvoriť správu alebo službu starajúcu sa o tieto informácie.

Pre stav batérie som zvolil službu bežiacu v uzle na robotovi. Takže aplikácia len využije túto službu a opýta sa na stav batérie keď ho bude potrebovať. Uzol stav zistí a ako odpoveď pošle hodnotu vyjadrujúcu zostávajúcy stav batérie v percentách.

Uzol publikuje obraz z kamery do "topicu"ako správu, z tohoto topicu aplikácia získava obraz prostredníctvom RosJavy pomocou odoberateľa. Tento prístup zaručuje prenos snímkov z kamery čo najrýchlejšie na výstup aplikácie, takže užívateľ môže vidieť aktuálne snímaný obraz.

GPS súradnice robota sú taktiež získavane pomocou služby bežiacej v uzle robota. Aplikácia v pravidelných intervaloch požiada robota o zaslanie GPS súradníc a uzol ako odpoveď zašle správu v ktorej je zemepisná dĺžka a šírka ako uložená ako čísla s pohyblivou desatinnou čiarkou s dvojitou presnosťou.

Poslednou získavanou informáciou je naplánovaná trasa. Aplikácia požiada o naplánovanie trasy sprostredkovateľský uzol v robotovi s tým, že mu musí zaslať cieľ trasy. Z toho vyplíva, že daný prenos je taktiež realizovaný pomocou služby. Ako odpoveď na cieľ aplikácia obdrží trasu ako zoznam takzvaných "waypointov", z nich každý označuje zemepisnú šírku a výšku bodu, ktorým robot plánuje prejsť.

Odosielané správy

Aplikácia má robota riadiť takže je samozrejmé, že mu musí tiež zasielať nejaké riadiace správy. Ako už bolo spomenuté v príjmaných správach ak chce naplánovať trasu musí mu zaslať cieľ. Ďalšími dôležitími príkazmi sú zmazanie alebo zmena trasy, zastavenie a tiež riadiace správy pre manuálne ovládanie robota. Keďže tieto ďalšie správy sú len riadiace príkazy bez akýchkoľvek ďalších dát až na rýchlosť a uhol rozhodol som sa ich zasielať všetky na ten istý topic ako správy obsahujúce príkazove slovo uložené v reťazci znakov. Rýchlsť pohybu a otáčania v sebe musí niesť hodnotu, ale zvolil som použitie tohoto spôsobu pretože vytvárať nový topic s podobným zameraním len iným typom správy je podľa mňa neefektívne. Jednotlivé príkazy a ich funkcia:

- delete - zmaž uloženú trasu
- stop - zastav
- go - vydaj sa po naplánovanej trase
- up - choď dopredu
- down - choď dozadu
- left - choď vľavo
- right - choď vpravo
- speed X - zmeň rýchlosť na X
- angle X - zmeň rýchlosť otáčania na X

3.3 Návrh realizácie hlavného uzla robota

Hlavný uzol je uzol vytvorený ako python skript bežiaci na robotovi, aby uľahčil a štandardizoval komunikáciu s robotom. Nazval som ho hlavný uzol pretože je to jediný uzol, s ktorým bude aplikácia komunikovať. Ak by aplikácia komunikovala priamo s robotom tak by sa mohlo stať, že pri pripojení na iného robota by potrebný topic mal iné pomenovanie a to by spôsobilo nefunkčnosť aplikácie. Takáto aplikácia by sa musela pre každého robota upraviť a prekompilovať. Takýto návrh je podľa mňa neprijateľný pretože ako vyplýva zo zadania obsluhu by mal zvládnuť aj bežný užívateľ bez znalostí programovania android aplikácií.

Hlavný uzol by mal zvládať zbierať a pripraviť informácie o stave batérie, gps súradnice polohy robota, obraz z kamery robota, obsluhovať prípadne zrušiť naplánovanie trasy.

Uzol by mal taktiež vedieť vykonávať príkazy prichádzajúce z aplikácie v podobe textových reťazcov.

Spolupráca pri návrhu API uzla

Na odporúčanie vedúceho mojej práce som pri návrhu spolupracoval s kolegom Andrejom Tichým. Jeho témou je Webový portál pro správu pozemních robotů. Rozhodli sme sa pre spoluprácu z dôvodu, že naše zadania sú v mnohých smeroch podobné. Najdôležitejším faktorom bolo, že naše práce sú zamerané na rovnakého robota. Spolupracovali sme hlavne pri návrhu uzla, jeho topicov a služieb. Taktiež sme spoločne navrhli niektoré funkcie na spracovávanie dát získaných z robota ako napríklad stav batérie.

Informácie o stave batérie

Stav batérie nieje možno najhodnotnejšou informáciou o stave robota, ale v praktickom požívaní aplikácie je určite dobré vedieť približne koľko času ešte robotovi ostáva kým sa vybije aby ho vedel užívateľ pred vybitím napríklad dopraviť bližšie k sebe na opetovné nabitie.

Robot je funkčný pokiaľ sa napätie pohybuje medzi 14V, čo je maximum a 11V, pri kotrých ešte dokáže fungovať. Uzol pri požiadaní o službu zistí z topicu s informáciami o batérii akuálny stav a ten potom prevedie na percentálny stav. Keďže krivka vybýjania batérie nieje lineárna prevádza zistené napatie podľa dopredu vypočítaných a určených hodnôt, aby čo najviac zodpovedali reálnemu stavu. Percentuálny stav potom zašle aplikácii ako odpoveď na požiadavok služby.

GPS súradnice polohy robota

Zisťovanie polohy robota je jednou z kľúčových častí môjho projektu, pretože sa na ňom zakladá celkové plánovanie trasy a zobrazovanie na mape. Uzol odoberá stále aktuálnu pozíciu z topicu robota, v ktorom sa publikujú GPS súradnice robota upravené Kalmanovým filtrom aby sa minimalizovala prípadna odchylka. Po prijatí požiadavky v službe uzol pripraví odpoveď a odošle ju aplikácii.

Obraz z kamery robota

Obraz z kamery robota je asi najviac vypovedajúca informácia o stave robota pre užívateľa. Vďaka obrazu môže užívateľ zistiť napríklad či sa robot niekde zasekol alebo prečo nemôže pokračovať v danej trase. Vďaka obrazu tiež môže užívateľ robota manuálne riadiť po trase bez toho aby ho musel priamo vidieť.

Obraz je získavaný uzlom odoberaním z topicu kamery kde sa zverejňuje komprimovaný obraz. Komprimovaný obraz som vybral preto lebo sa musí prenášať po bezdrôtovej sieti a prispieva to k svižnosti zobrazovaného videa, čo je určite pre užívateľa prívetivejšie.

Plánovanie trasy

Táto práca je zameraná hlavne na užívateľské rozhranie pre riadenie robota vo vonkajšom prostredí a presnosť plánovania trasy je dôležitou zložkou tohto projektu. Samotné plánova-

nie na základe open street mapy je však podľa mňa náročnosťou nad rozsahom tejto práce preto som sa rozhodol použiť program na plánovanie trasy, ktorý je už vytvorený.

Uzol po prijatí požiadavky o naplánovanietrasy, ktorá v sebe nesie súradnice cieľa trasy vytvorí požiadavku pozostávajúcu zo súradníc začiatočnej lokácie, súradníc cieľovej lokácie a číslo označujúce maximálny počet segmentov trasy. Uzol potom požiadavku odošle službe pre plánovanie trasy a ako odpoveď dostáva zoznam bodov cesty, ktorími sa chystá robot prejsť. Tento zoznam upraví na formu vhodnú pre kreslenie čiar v v aplikácii a prepošle.

Obsluha prichádzajúcich príkazov

Hlavný uzol musí zvládať obsluhovať jednoduché príkazy prichádzajúce od aplikácie a na ich základe upravovať stav a správanie robota. Vďaka tejto obsluhe píkazov umožní manuálne riadenie robota alebo napríklad zmazanie vytvorenej trasy.

Na riadenie robota slúži topic `cmd_vel` , z ktorého robot odoberá rýchlosť uvedenú v metroch za sekundu a tiež uhol natočenia zadaný v radiánoch. Keď uzol obdrží príkaz na pohyb nejakým smerom vytvorí správu na základe aktuálne nastavených rýchlostí a odošle ju robotovi. Správy delete, stop a go sa preposialjú robotovi. Správy s rýchloťami sú určené len pre tento uzol.

3.4 Simulácia

Pre možnosť testovania funkčnosti jednotlivých prvkov pri vývoji aplikácie a pre demonštračné účely je potrebné vytvoriť jednoduchý simulátor robota, ktorý by dokázal reagovať na všetky možné požiadavky od hlavného uzla. Musí dokázať simulovať polohu robota, jeho rýchlosť a natočenie, stav batérie, obrazový záznam a tiež naplánovanie trasy.

Simulovaná poloha robota a stav batérie

Pri spustení simulácie sa vygenerujú GPS súradnice robota v uložené v simulátore ako čísla s pohyblivou desatinnou čiarkou s dvojitou presnoťou. Po vygenerovaní simulátor vytvorí topic do ktorého tieto dáta publikuje v pravidelných intervaloch.

Keď sa robot simulovane vydá po naplánovanej trase spustí nové vlákno, aby sa zabránilo blokovaniu publikovania polohy a v novom vlákne túto polohu postupne mení podľa naplánovanej trasy.

Stav batérie je jednoduché vygenerované číslo medzi 11 a 14, ktoré sa uloží podobne ako gps súradnice a potom simulátor vygeneruje správu ktorú publikuje do topicu so stavom batérie.

Simulované viedo a pohyb

Na simulovanie ovládania robota a zhotovovania obrazového záznamu som sa rozhodol použiť externé programy, ktoré sa využívajú vo výukových príkladoch k ROS.

Pre simuláciu videa som zvolil program `usb_cam`, ktorý sníma obraz z webkamery počítača na ktorom simulácia beží a publikuje ho do topicu podobne ako kamera robota. Tento spôsob som vybral preto lebo je tam podľa môjho názru lepšie vidiet aktuálnosť obrazu ako u umelo vygenerovaného obrazu.

Na simuláciu pohybu som sa rozhodol použiť program `turtlesim`. Tento program reaguje na správy publikované v topicu `cmd_vel` rovnako ako robot a pohyb zobrazuje na obrazovke. Pre tento program som sa rozhodol, lebo zobrazovaný výstup presne znázorňuje

ako by sa správal robot a grafický výstup je v tomto prípade podľa mňa lepšie zrozumiteľný ako textový výstup, ktorý by produkoval priamo môj simulátor.

Simulovanie plánovania trasy

Simulátor po štarte spustí službu generovania trasy totožnú so službou reálneho plánovača u robota a čaká na požiadavky na naplánovanie. Keďže plánovanie trasy je zložitý problém, v simulátore som sa rozhodol generovať len pravouhlé trasy medzi zadanými začiatočnými a koncovými bodmi. Vytvorenú trasu potom pošle ako odpoveď totožnú s výstupom z reálneho plánovača.

Kapitola 4

Realizácia, experimenty a vyhodnotenie

4.1 Nástroje použité pri realizácii

Realizácia aplikácie

Navrhnutá aplikácia bola implementovaná v jazyku Java pomocou vývojového prostredia Android Studio. Ako základ boli použité zdorjové kódy získané z tutoriálov pre implementáciu RosJava do android aplikácie. Ďalšou dôležitou časťou bolo použitie Google Maps API v2 na zobrazovanie mapy, značiek polôh a tiež vykresľovanie trasy. Medzi základné knižnice použité v aplikácií patri `android-support-v4`, ktorá je hlavnou knižnicou podporujúcou vývoj pre android API úrovne 4 a viac, ďalšou dôležitou knižnicou bola `google-play-services`. Táto knižnica obsahuje najnovšie rozhrania pre mnohé služby poskytovane Googleom ako sú napríklad Google Maps, Locations API, Cloud Massaging atď. V tejto aplikácii bola knižnica využitá na prácu s Google Maps API. Poslednou z nepostrádateľných knižníc tejto aplikácie je knižnica umožňujúca prácu s `RosJava`.

Cieľová platforma pre aplikáciu je zariadenie s operačnýcm systémom Android verzie 4 a vyšie. Štvrtá verzia je nutná najmä kvôli použitiu nového Google Maps API v2.

Realizácia hlavného uzla

Hlavný uzol bol realizovaný v jazyku Python a to konkrétne verzie 2.7. Tento jazyk som zvolil preto lebo bol flexibilnejší ako C++, ktoré sa pri využití Robot Operating System tiež veľmi často používa, umožňnoval úpravy kódu bez nutnosti opetovného prekompilovania celého projektu. Najdôležitejšou použitou knižnicou je `rospy` umožnujúca prácu s ROS a komunikáciu s robotom. Vytvoril som vlastné typy správ a služieb vďaka nízkoúroňovému buildovaciemu systému catkin ktorý je súčasťou ROS. Pri práci s robotom bola použitá verzia Robot Operating System s označením Hydro.

Cieľovým zariadením pre spúšťanie hlavného uzla je robot typu Pioneer s operačným systémom Linux Ubuntu verzie 12.04 s nainštalovaným Robot Operating System verzie Hydro. Robot by mal tiež obsahovať GPS modul a kameru pre úplnú podporu všetkých funkcií aplikácie.

4.2 Samotná realizácia

Užívateľské rozhranie

Aplikácia pozostáva zo štyroch základných aktivít, ktorých užívateľské rozhranie je realizované podľa návrhu z kapitoly 3.2. Vo všetkých aktivitách je použitá relatívna dispozícia prvkov takže by sa aplikácia mala prispôsobiť všetkým rozlíšeniam obrazovky. Každý prvok je nezávislý na hustote obrazovky zariadenia. Ako hlavnú tému som použil štandardnú tému Android Holo Light. Tlačidlá som upravil zaoblením rohov a zmenou farby na Android Holo Blue používanú štandardne v aplikáciach s verziou systému Ice Cream Sandwich a vyššou.

Aby bola aplikácia úplne nezávislá na hustote pixelov, musel som vytvoriť všetky používané obrázky v štyroch veľkostiach. Takto som musel upraviť šípky na pohyb robota a tiež ikonu aplikácie.

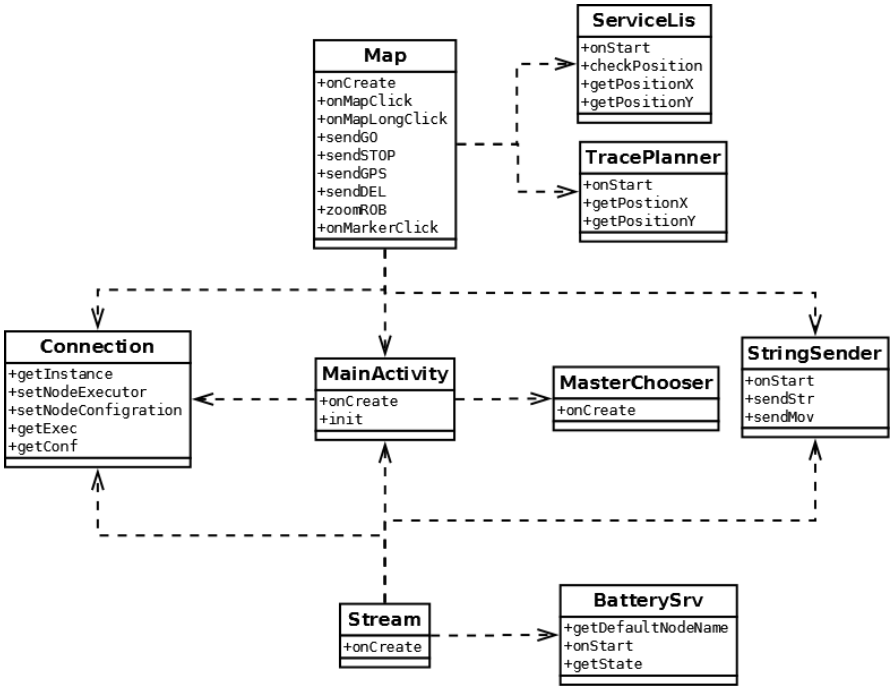
Obrázek 4.1: Na hustote nezávislá ikona aplikácie.

Aplikácia

Najdôležitejšou, hoci graficky najmenej zaujímavou je aktivita, v ktorej je možnosť zvoliť režim plánovania tras, alebo zistovania stavu robota. Táto aktivita sa spúšťa ako prvá po štarte robota a volá aktivitu pre prihlásenie k robotovi. Po prihlásení vytvára novú inštanciu triedy `Connection` pre uloženie údajov potrebných pre prácu s robotom, ktorá je implementovaná ako jedináčik (singleton). Nosná aktivita tejto aplikácie je nazvaná `Map` umožňuje prácu s mapou a posielá požiadavky na naplánovanie trasy prostredníctvom triedy `Trace_plan` a zisťuje polohu robota pomocou triedy `ServiceLis` pre zasielanie požiadavkov služby a príjmanie odpovede. Aktivitu pre zobrazovanie stavu robota som nazval `Stream`, pretože prenos videa je asi najviac vypovedajúcim prvkom o aktuálnom stave robota. Táto aktivita príjma komprimované video z robota pomocou triedy implementovanej v knižnici `RosJava` s názvom `RosImageView` a zasiela príkazy ako reťazce znakov pomocou triedy `StringSender`.

Uzol

Uzol pracuje ako sprostredkovateľ medzi aplikáciou a samotným robotom. Komunikuje s oboma prostredníctvom ROS správ a služieb. Získava dáta z robota a upravuje ich aby boli ľahko interpretovateľné v android zariadení. Taktiež na základe prijatých príkazov ovláda robota, poprípade volá potrebné služby.



Obrázek 4.2: Diagram najdôležiitejších tried aplikácie.

- **/uinode/command**

  Tento topic odoberá všetky jednoduché príkazy aplikácie ako sú napríklad pohyb, stop, go, delete, nastavenia rýchlostí a na ich základe vykonáva obslužné rutiny pre každý príkaz. Topic príjma štandardnú správu vo forme reťazca znakov. Príkaz na zmenu rýchlosti nesie tiež hodnotu rýchlosti, ktorú uzol uloží co svojích globálnych premenných. Pre pohyb vytvorí správu s vhodnými parametrami na základe rýchlostí uložených v globálnych premenných a odošle ju na topic `/cmd_vel`. Príkaz stop nastaví obe rýchlosti v uzle na nulu a pošle stop tiež uzlu, ktorý ma na starosti pohyb po trase. Príkazy delete a go sa len preposielaju ďalej robotovi, prípadne simulácii.

- **uinode/battery**

  Uzol poskytuje túto službu na zisťovanie stavu batérie. Aplikácia o ňu požiada len vtedy keď ju potrebuje aby sa zamedzilo zbytočnej réžii. Po prijatí požiadavky uzol odoberie stav batérie z topicu poskytovaného robotom/simuláciou a prevedie ho na percentuálny stav, ktorý odošle ako odpoveď. Požiadavkom tejto služby je reťazec znakov s typom batérie a odpoveďou je celočíselná hodnota s percentuálnym stavom.

- **uinode/location**

  Táto služba je určená pre prenos GPS súradníc polohy robota. Uzol odoberá stále aktuálnu polohu a ukladá si ju vo svojích premenných. Aplikácia posiela požiadavku len keď polohu potrebuje a uzol odpovie dvoma číslami s pohyblivou desatinnou čiarkou s dvojitou presnosťou označujúcimi zemepisnú šírku a výšku.

- **uinode/plan_trace**

  Vďaka tejto službe je možné plánovanie trasy. Služba príjme zadaný cieľ trasy určený dvoma desatinnými. Po prijatí uzol vytvorí požiadavok pre službu plánovania trasy

v robotovi pridaním štartovacej lokácie a maximálnym počtom častí trasy. Po prijatí odpovede prevedie smerové body trasy na polia desatinných čísel a odošle aplikácii. Táto služba mohla byť teoreticky vynechaná tým, že by aplikácia posielala požiadavky priamo službe s planovaním v robotovi, ale tento spôsob poskytuje možnosť zmeniť plánovač bez zásahu do aplikácie.

4.3 Testovanie

V tejto kapitole sa budem zaoberať návrhom testov, samotným testovaním a vyhodnocovaním výsledkov testov. Pretože mojou prioritou v tejto práci je návrh a vyhotovenie užívateľského rozhrania, musím vytvorenú aplikáciu testovať aj prostredníctvom viacerých užívateľov aby som zistil vhodnosť a intuitívnosť užívateľskho rozhrania.

Testovanie funkčnosti

Testovanie funkčnosti prebiehalo priebežne s realizáciou aplikácie ako na robotovi tak aj za pomoci simulácie. V konečnej aplikácii som otestoval funkčnosť správnosti návrhu prerušovanou prácou s aplikáciou, keď som ju viackrát minimalizoval a maximalizoval.Taktiež som za pomoci kolegov s podobnými zadaniami spravil záťažový test, kde sme na jednej sieti spustili 3 naše riešenia bez toho aby sme si všimli nejaké vzájomné rušenie. Z tohoto testu vyplíva, že moja aplikácia potrebuje na správny chod pomerne malú šírku pásma.

Testovanie užívateľského rozhrania

Pretože táto práca je zameraná na vytvorenie intuitívneho užívateľského rozhrania, ktoré dokáže ovládať aj človek bez väčších skúseností s používaním aplikácií tohoto druhu je testovanie návrhu rozhrania veľmi dôležité. Testy som navrhol tak, aby obsiahli všetky dôležité ovládacie prvky aplikácie. Testovanie pozostávalo zo štyroch úloh, ktoré musel užívateľ zvládnuť. Pre každú úlohu som zmeral čas, za aký užívateľ danú úlohu splnil a všímal som si situácie, v ktorých sa zdržal najviac.

Užívateľ	1.	2.	3.	4.
1.	2:10	0:28	0:22	0:40
2.	0:48	0:18	0:29	1:00
3.	1:30	0:30	0:26	0:56
4.	0:30	0:31	0:16	0:29
5.	1:10	0:22	0:24	0:42

1. Pripojte sa k robotovi a prikážte mu aby šiel na miesto vzdialené aspoň 100 metrov na severovýchod od robota.

Prvá úloha bola zameraná na zoznámenie sa s aplikáciou a na to ako sa užívatelia vysporiadajú s plánovaním trasy, keďže práca s cieľom trasy je založená hlavne na princípoch využívajúcich dotykovú obrazovku. Tieto princípy by mohli pôsobiť problémy užívateľom s menšími skúsenosťami s androidom.

Táto úloha zabrala priemerne najviac času a bola najproblematickejšou na riešenie. Pri jej riešení sa vyskytli dve hlavné problémy. Hlavným problémom bolo zadávanie a manipulácia s cieľom trasy u užívateľov, ktorí nepoužívajú často tablet alebo smartphone. Plánovanie a príkaz k pohybu už potom užívateľom nenepôsobil žiadne problémy.

Ďalšou dôležitou príčinou zdržania bolo priblíženie robota na mape na úroveň ulíc, aby mohli užívatelia zadať miesto v určenej vzdialenosti.

2. Zastavte pohyb robota a zmeňte jeho trasu aby sa vrátil na miesto kde sa nachádzate.

Druha úloha mala preveriť použiteľnosť ďalších ovládacích prvkov a ukazateľov.

Táto úloha už užívateľom išla rýchlejšie, keďže už vedeli ako plánovať trasu a mapa bola dostatočne priblížená. Zastavenie robota v pohybe a mazanie aktuálnej trasy a nájdenie svojej polohy užívateľom nespôsobovalo žiadne problémy.

3. Zistite stav batérie a zmonitorujte okolie robota pomocou kamery.

Táto úloha sa venovala duhej polovici aplikácie, ktorá je zameraná na zisťovanie stavu robota a teleoperáciu. Hlavním cieľom tejto úlohy bolo overiť správnosť rozmiestnenia prvkov a prirodzenosť ovládania.

U niektorých užívateľov v tejto úlohe došlo k zdržaniu spôsobenému hlavne názvom tejto aktivity v menu kde nebola spomenutá teleoperácia. Zistenie stavu batérie nepôsobilo žiadne problémy, taktiež sa testovaní užívatelia rýchlo zorientovali v princípoch ovládania robota.

4. Rýchlosťou 0.5 m/s a rýchlosťou otáčania 0.5 rad/s prejdite v simulácii štvorcovu dráhu.

Cieľom poslednej úlohy bolo zistiť vhodnosť vybraného spôsobu ovládania pomocou krížovo umiestnených šípok.

Splnenie tejto úlohy zabralo pomerne veľa času. Niektorích užívateľov zdržalo nastavovanie správnych rýchlostí no toto zdržanie nebolo veľké v porovnaní s časom stráveným samotným pohybom robota. Zdržanie nebolo však spôsobené chybou užívateľského rozhrania, ale skôr dĺžkou dráhy a snahou užívateľov o čo najlepší výsledok.

Zhodnotenie výsledkov a návrh vylepšení

Vďaka týmto štyrom jednoduchým testom som bol schopný odhaliť niektoré vážnejšie nedostatky môjho návrhu a odstrániť tieto nedostatky v maximálnej možnej miere. Aj keď meraný čas bol dôležitým faktorom najviac mi pomohlo sledovať jednotlivých užívateľov a ich postupov pri riešení zadanej úlohy.

Väčšina testovaných v prvej úlohe stláčala ako prvé tlačidlo `Plan` namiesto toho aby najprv zadali cieľ plánovania trasy. Preto som sa rozhodol pridať do tejto aktivity správu, ktorá by užívateľa naviedla na interakciu s mapou a pridanie cieľa. Keďže veľké zdržanie pôsobilo aj približovanie mapy a zameriavanie na robota, pridal som tlačidlo `Find robot`, po stlačení ktorého aplikácia nájde robota a zobrazí ho s mapou priblíženou na šestnástu úroveň priblíženia, čo je priblíženie na úroveň ulíc. Pridanie tohoto tlačidla by malo značne urýchliť prácu s aplikáciou.

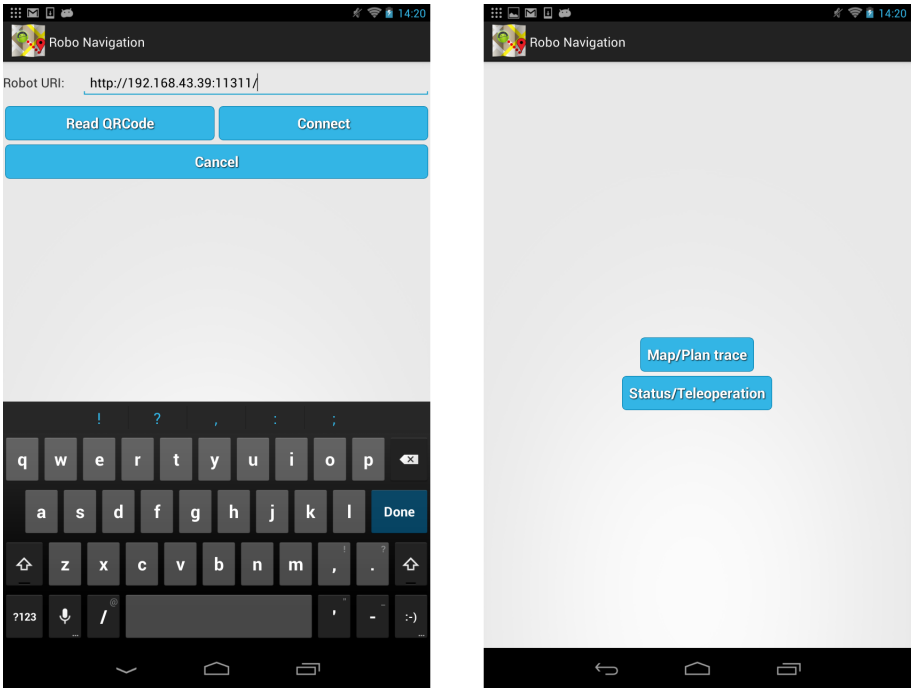
Druhá úloha potvrdila to, že akonáhle je užívateľ oboznámeny s princípmi ovládania mapy gestami na dotykovej obrazovke nepôsobí mu práca s touto časťou aplikácie žiadne problémy.

Tretiu úlohu splnili užívatelia pomerne rýchlo, čo vypovedá v prospech rozloženia ovládacích prvkov. Jediný problém spôsobovali nedostatočne vypovedajúce názvy v hlavnom menu aplikácie.

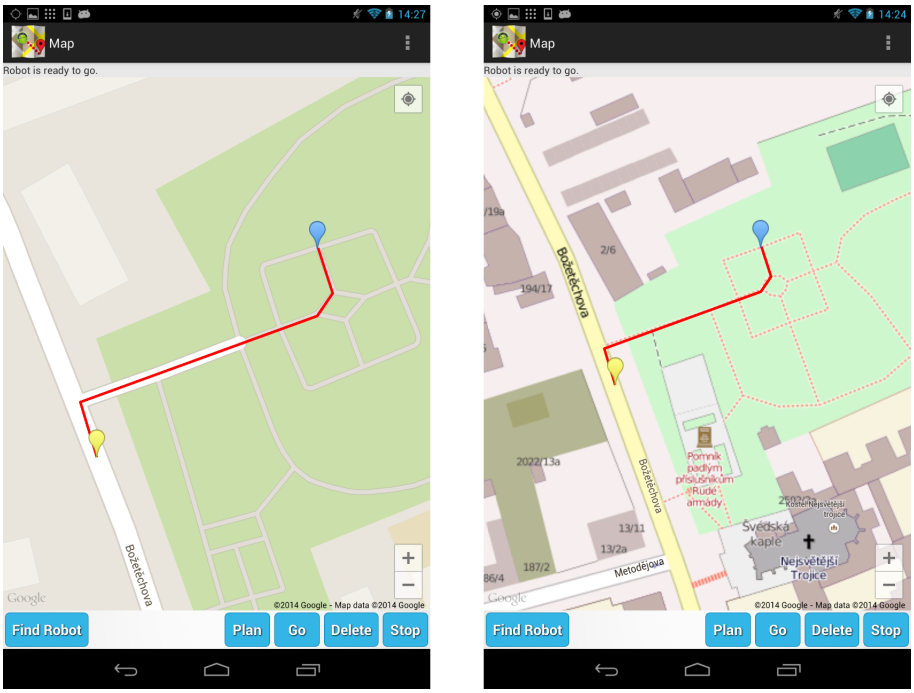
Jedinou pozorovanou chybou užívateľského rozhrania v poslednej úlohe bolo, že užívateľ nevedel presný rozah hodnôt na posuvníkoch pri nastavovaní rýchlostí. Tento ptoblém som odstránil jednoduchým pridaním maximálnej hodnoty pri oboch posuvníkoch.

4.4 Výsledná aplikácia

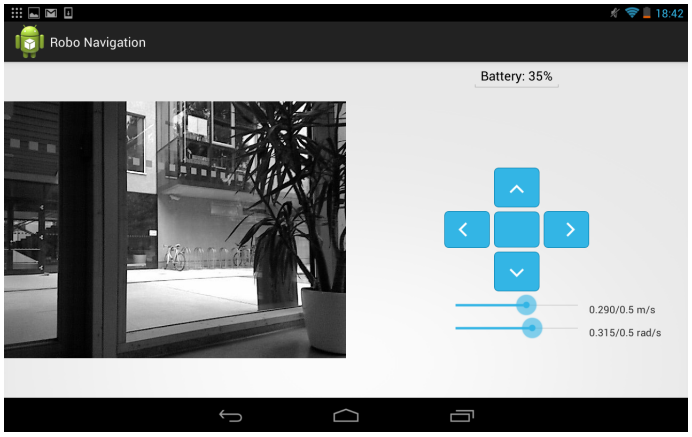
Po vyhodnotení testov a identifikácii nedostatkov som vytvoril konečnú podobu funkčnej aplikácie pripravenú na používanie.



Obrázek 4.3: Naľavo prihlasovacia obrazovka, napravo menu na výber možností.



Obrázek 4.4: Aktivita pre plánovanie trasy. Naľavo zobrazenie Google mapy, napravo zobrazenie Open Street mapy



Obrázek 4.5: Aktivita s informáciami a možnosťou ovládať robota.

Kapitola 5

Záver

Cieľom tejto bakalárskej práce bolo vytvoriť funkčnú aplikáciu s užívateľským rozhraním pre jednoduchú navigáciu a ovládanie robota vo vonkajšom prostredí. Aby som zadanie mohol splniť musel som sa oboznámiť s postupmi a nástrojmi používanými pri vývoji mobilných aplikácií a princípy tvorby užívateľského rozhrania pre tieto aplikácie. Taktiež som študoval doteraz pre mňa úplne neznámu, ale o to zaujímavejľiu platformu Robot Operating System (ROS) a jej implementáciu v programovacom jazyku java RosJava.

Vďaka získaným poznatkom som navrhol užívateľské rozhranie, ktoré by malo byť ľahko pochopiteľné a ovládateľné aj užívateľovi, ktorý sa s podobným typom aplikácie ešte nestretol. Navrhol som tiež aplikáciu riadenú týmto rozhraním, ktorá pomocou RosJava komunikuje s robotom, spracováva prijaté údaje a sprostredkuje ich užívateľovi napríklad prostredníctvom streamu videa alebo Google Maps API. Aby som umožnil podporu rôznych typov výstupov dát z robota bez nutnosti úpravy samotnej aplikácie rozhodol som sa tiež navrhnúť uzol bežiaci priamo na robotovi, ktorý sprostredkuje informácie a upravý ich do formy vhodnej pre aplikáciu. Uzol taktiež riadi robota na základe príkazov prijatých od aplikácie. Kvôli urýchleniu a uľahčeniu vývoja aplikácie som tiež navrhol jednoduchý simulátor napodobňujúci všetky dvôležité funkcie a výstupy robota.

Navrhnuté prvky som implementoval a otestoval. Aplikáciu som tiež podrobil užívateľským testom s piatimi dobrovoľníkmi. Na základe testov som zistil niektoré nedostatky užívateľského rozhrania, ktoré trocha spomaľovali prácu s aplikáciou u niektorých užívateľov. Tieto nedostatky som v ďalšej verzii aplikácie odstránil.

Práca na tomto zadaní mi priniesla cenné vedomosti o tvorbe aplikácií pre android ako aj o programovaní robotov. Obe obory ma zaujali a rád by som sa im venoval aj v budúcnosti.

Literatura

[1] Google Maps Android API v2 Documentation [online]. https://developers.google.com/maps/documentation/android/map?hl=sk, [cit. 2014-02-07].

[2] Managing the Activity Lifecycle [online]. http://developer.android.com/training/basics/activity-lifecycle/starting.html, [cit. 2014-02-07].

[3] Multiple Screen Support [online]. http://developer.android.com/guide/practices/screens_support.html, [cit. 2014-02-07].

[4] ROS/Concepts - ROSWiki. http://wiki.ros.org/ROS/Concepts, [cit. 2014-05-02].

[5] ROS/Introduction - ROSWiki. http://wiki.ros.org/ROS/Introduction, [cit. 2014-05-02].

[6] Pioneer 3-AT Datasheet. http://www.mobilerobots.com/Libraries/Downloads/Pioneer3AT-P3AT-RevA.sflb.ashx, [cit. 2014-05-15].

[7] Pioneer 3 Operations Manual, Version 3. http://www.ist.tugraz.at/_attach/Publish/Kmr06/pioneer-robot.pdf, [cit. 2014-05-15].

[8] RoboLab - Laboratory of Robo@FIT hardware and platforms. http://merlin.fit.vutbr.cz/wiki/index.php/RoboLab, [cit. 2014-05-15].

[9] Kinect - slides. http://www.lsi.upc.edu/~virtual/RVA/Course%20Slides/Kinect.pdf, [cit. 2014-05-16].

[10] Kinect for Windows Sensor Components and Specifications. http://msdn.microsoft.com/en-us/library/jj131033.aspx, [cit. 2014-05-16].

[11] Senzorika - Letecká fakulta. http://www.senzorika.leteckafakulta.sk/?q=node/269, [cit. 2014-05-16].

[12] Ahmad Raza, T. M., Shafqat Hameed: *Global Positioning System - Working and its Applications*. Springer Netherlands, 2008, iSBN 978-1-4020-8735-6.

[13] Lee, W.-M.: *Beginning Android 4 Application Development*. John Wiley & Sons, Inc., 2012, 978-1-118-19954-1.

Příloha A

Obsah CD

- `/app` - zdrojové súbory android aplikácie
- `/node` - ROS balík uzla
- `/apk` - skompilovaná aplikácia
- `/tz` - táto technická správa
- `/tz/latex` - zdrojové súbory technickej správy
- `/demo` - demonštračné video
- `/plagat` - plagát prezentujúci túto prácu