SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Inteligentné plánovanie pohybu robota

Tímový projekt

Študijný program: Robotika a kybernetika

Študijný odbor: 9.2.7 Kybernetika

Školiace pracovisko: Ústav robotiky a kybernetiky

Vedúci záverečnej práce/školiteľ: Ing. Miroslav Kohút

**Bc. Vladimír Vadkerti**

**Bc. František Balázsy**

**Bc. Lukáš Šníder**

**Bratislava 2020 Bc. Matej Krajči**



ABSTRAKT

Slovenská technická univerzita v Bratislave  
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný odbor: 9.2.7 Kybernetika

Študijný program: Robotika a kybernetika

Autori: Bc. Vladimír Vadkerti

Bc. František Balázsy

Bc. Lukáš Šníder

Bc. Matej Krajči

Tímový projekt: Inteligentné plánovanie pohybu robota

Vedúci tímového projektu: Ing. Miroslav Kohút

Mesiac, rok odovzdania: Máj, 2020

Kľúčové slová: Čierny robot, Robotický operačný systém, mapovanie, lokalizácia, navigácia, laserový skener, pohyb,

Cieľom tohoto tímového projektu bol návrh a zabezpečenie bezkolízneho navigovania mobilného robota v uzatvorenom lokálnom prostredí, pomocou robotického operačného systému (ROS). Práca sa zaoberá širokým spektrom tém, ako implementácia modelu robota, lokalizovanie v neznámom priestore, vytvorenie mapy, orientácia v priestore a bezkolízna navigácia na požadované miesto v zmapovanom prostredí. V práci stručne popisujeme funkčnosť algoritmov a porovnávame rôzne prístupy k lokálnej navigácii.

Obsah

[Obsah 4](#_Toc39527650)

[Zoznam použitých skratiek 6](#_Toc39527651)

[1 ROS 7](#_Toc39527652)

[1.1 Opis fungovania základných procesov ROS-u 7](#_Toc39527653)

[1.1.1 Uzly (Nodes) 7](#_Toc39527654)

[1.1.2 Správy (Messeges) 8](#_Toc39527655)

[1.1.3 Hlavný uzol (Master) 8](#_Toc39527656)

[1.2 Použitie balíčkov z ROS-u 8](#_Toc39527657)

[1.2.1 Teleop\_twist\_keyboard 8](#_Toc39527658)

[2 Lokalizácia a mapovanie 10](#_Toc39527659)

[2.1 Princím SLAM algortimu 10](#_Toc39527660)

[2.2 SLAM balíček Gmapping 11](#_Toc39527661)

[2.3 Laserový skener 11](#_Toc39527662)

[3 Pravdepodobnostný odhad polohy 14](#_Toc39527663)

[3.1 Lokalizácia v zmapovanom priestore 14](#_Toc39527664)

[3.1.1 Algoritmus Monte Carlo 14](#_Toc39527665)

[3.1.2 Balíček AMCL 14](#_Toc39527666)

[4 Navigácia 16](#_Toc39527667)

[4.1 Globálna navigácia 17](#_Toc39527668)

[4.1.1 Base global planner 17](#_Toc39527669)

[4.1.2 Carrot planner 18](#_Toc39527670)

[4.2 Lokálna navigácia 19](#_Toc39527671)

[4.2.1 Base local planner 19](#_Toc39527672)

[4.2.2 DWA local planner 20](#_Toc39527673)

[5 Implementácia v simulačnom prostredí 21](#_Toc39527674)

[5.1 Implementácia modulov mapovania 21](#_Toc39527675)

[5.1.1 Výsledky Gmapping mapovania 23](#_Toc39527676)

[5.2 Implementácia modulov lokalizácie 24](#_Toc39527677)

[5.2.1 Výsledky AMCL lokalizácie 26](#_Toc39527678)

[5.3 Implementácia modulov navigácie 26](#_Toc39527683)

[5.3.1 Výsledky Globálnej navigácie 28](#_Toc39527684)

[Výsledky Lokálnej DWA navigácie 28](#_Toc39527685)

[5.3.2 28](#_Toc39527686)

[5.4 Návrh napájania robota 29](#_Toc39527687)

[5.5 Implementácia HMI 29](#_Toc39527688)

[6 Záver 30](#_Toc39527689)

[6.1 Návod na spustenie 30](#_Toc39527690)

[6.2 Zhrnutie 31](#_Toc39527691)

[6.3 Harmonogram stretnutí a rozdelenie úloh 33](#_Toc39527692)

[Literatúra 34](#_Toc39527693)

Zoznam použitých skratiek

ROS – Robot operation system

DWA – Dynamic window approaches

AMCL – Adaptive Monte Carlos Localization

SLAM – Simulataneous Localization And Mapping

ID – Identifier

# ROS

Robotický operačný systém (ROS), je flexibilný pre tvorbu softvéru pre roboty. Ponúka súbor nástrojov, knižníc a konvencií, ktorých cieľom je zjednodušiť úlohu vytvárania komplexného a robustného správania robotov v širokej škále robotických platforiem.

Komunikácia v ROS je vypracovaná na úroveň výpočtového grafu. Tento graf slúži k peer-to-peer sieti procesov, ktoré spracúvajú údaje. Medzi základné parametre patria master, server, uzly (nodes), správy (messeges), služby (services), topic, ktoré rôzne poskytujú údaje pre výpočtový graf [1].

ROS je do istej miery podobný s robotickými frameworkami ako napríklad Player, Orca, Orocos a iné. Dôvodov pre výber ROS môže byť veľa. Je kompatibilný s prostredím Linux, obsahuje veľa balíkov pre širokú škálu použitia, využíva programovacie jazyky C++, Python a v limitovanom rozsahu aj Java.

Momentálne existuje dvanásť distribučných verzií ROS. My v tomto tímovom projekte budeme používať ROS Melodic. V prípade verzie ROS Melodic sme si zvolili verziu Ubuntu 18.04 LTS, ktorý je odporúčaný operačný systém.

Softvér ROS-u je spravovaný v balíčkoch, každý z balíčkov obsahuje modul s funkcionalitou, ktorá napomáha k riešeniu danej problematiky. Obsah balíčka pozostáva z uzlov, knižníc, konfiguračných súborov a iných častí, ktoré by korešpondovali s využívaním daného modulu [3].

## Opis fungovania základných procesov ROS-u

### Uzly (Nodes)

Základným prvkom ROS-u je práve uzol v preklade node. Uzol obsahuje vlastný PID. Je to proces ktorý môžeme spustiť na aktuálne bežiacich masteroch. Uzly nám ponúkajú možnosť transparentného rozdelenia veľkého problému na menšie subproblémy. Hlavným uzlom systému ROS je „rosout“, ktorý je nevyhnutným pre ovládanie ostatných uzlov.

Komunikácia medzi uzlami je, v ROS možná pomocou troch základných spôsobov, a to pomocou služieb, tém a akcií. Služby komunikujú prostredníctvom žiadostí a odpovedí, čo je veľmi využiteľné a slúži na občasnú komunikáciu. Témy sú spojenia s vlastným názvom a typom prenesenej správy. Témy dokážu prijímať a posielať správy, z ktoréhokoľvek uzlu. Komunikácia pomocou tém je založená na báze protokolov TCP/IP a UDP [5].

### Správy (Messeges)

Správy patria medzi najpoužívanejšie formy komunikácie medzi uzlami. Správy sú medzi uzlami publikované pomocou tém. Zloženie správy sa skladá z dátových typov, akými sú string, int, array, char, float, atď. Na začiatku správy býva hlavička, ktorej obsahom sú dodatočné informácie o správe, napríklad čas alebo ID.

### Hlavný uzol (Master)

ROS Master má zabezpečiť registrácie a pomenovania serverov pre ostatné uzly v ROS. Jeho hlavnou úlohou je správa povolení jednotlivým uzlom spájať sa s ostatnými. Ako náhle master nájde dané uzly, môžu medzi sebou komunikovať peer-to-peer.

## Použitie balíčkov z ROS-u

### Teleop\_twist\_keyboard

Medzi najjednoduchšie balíčky dostupné v ROS-e patrí balíček na ovládanie podvozku mobilného robota pomocou klávesnice. Tento balíček je možne nainštalovať a spustiť pomocou nasledujúcich príkazov v operačnom systéme Ubuntu [5].

Inštalácia balíčka:

sudo apt-get install ros-melodic-teleop-twist-keyboard

Spustenie balíčka:

rosrun teleop\_twist\_keyboard teleop\_twist\_keyboard.py

Ovládanie:

Reading from the keyboard and Publishing to Twist!

---------------------------

Moving around:

u i o

j k l

m , .

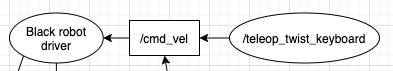
q/z : increase/decrease max speeds by 10%

w/x : increase/decrease only linear speed by 10%

e/c : increase/decrease only angular speed by 10%

anything else : stop

CTRL-C to quit



Obr. 1 Prepojenie uzlov pre riadenie podvozku prostredníctvom klávesnice

Na Obr. 1 je pomocou nástroja rqt\_graph dostupného v ROS-e zobrazené prepojenie uzlov. Vidíme, ako nám balíček teleop\_twist\_keyboard publikuje správy typu „[geometry\_msgs/Twist](http://docs.ros.org/api/geometry_msgs/html/msg/Twist.html)“ na uzol cmd\_vel, ďalej sa tieto správy publikujú už na samotný hardware robota (blackrobot\_driver). Tieto správy obsahujú najmä údaje o rýchlosti a smere otáčania jednotlivých kolies podvozku.

# Lokalizácia a mapovanie

Na správne navigovanie robota v priestore potrebujeme najprv vedieť, kde sa robot nachádza, čo je pred ním a kde už bol. To všetko nám zabezpečuje SLAM algoritmus a jeho balíčky.

SLAM (Simulataneous Localization And Mapping), čo v preklade znamená súbežná lokalizácia a mapovanie je algoritmus, ktorý v sebe zahŕňa všetky potrebné informácie pre navigovanie robota, a zároveň vytváranie mapy prostredníctvom údajov získaných zo skenera. SLAM balíčky sú voľne dostupné a používajú sa napríklad v inteligentných autách, bezpilotných vzdušných dopravných prostriedkoch, mobilných robotoch a iných zariadeniach, ktoré sa potrebujú autonómne orientovať v priestore [6].

## Princím SLAM algortimu

Cieľom algoritmu SLAM je použiť prostredie na spresnenie polohy robota. Keďže odometria nie je presná, nemôžme používať dáta priamo z nej, ale používame obraz prostredia, teda údaje zo senzora na aktualizovanie pozície určenej z odometrie. V nasledujúcich krokoch si jednoduchosti ukážeme ako funguje algoritmus SLAM.

* **Inicializácia robota**: Robotovi sa v neznámom prostredí určí východzia teda nulová poloha.
* **Meranie:** Prostredníctvom laserového skenera sa naskenuje priestor a získajú sa dáta pre robota.
* **Aktualizácia:** Robotické zariadenie pozoruje objekty v priestore, ktoré zároveň zmapuje.
* **Predikcia:** Odhad novej polohy robota, pričom pri pohybe stúpa neistota polohy objektov.
* **Meranie:** Robot pozoruje novo namerané objekty
* **Aktualizácia:** Neistota polohy zmapovaného priestoru vyplýva z kombinácie neistoty merania a neistoty polohy robota.

Tieto kroky sa opakujú, až pokiaľ sa robot nedostaneme na miesto z ktorého vychádzal, teda z nulovej polohy. Tu sa slučka uzavrie a odhad polohy robota bude korelovať s odhadom polôh naskenovaných objektov, tým pádom sa neistota polôh objektov začne zmenšovať.

## SLAM balíček Gmapping

SLAM disponuje balíčkom Gmapping, ktorý je založený na snímaní z laserového skenera. Pomocou uzlu slam\_gmapping môžeme vytvárať 2D mapy prostredia, ako napríklad pôdorys budovy. Na využívanie tohto balíčka potrebujeme získavať údaje z laserového skenera a vzájomnú transformáciu robota. Balíček má niekoľko vstupných nastaviteľných parametrov, ktoré si užívateľ zadáva pred spustením algoritmu [5].

Výhodou balíčka je ľahká dostupnosť k dokumentácii, jednoduchá konfigurácia a aplikovanie balíčka pre rôzne typy robotov. Nevyhnutnou potrebou pre funkčnosť algoritmu je odometria, ktorou ale niektoré typy robotov nedisponujú.

**Prijímané uzly balíka:**

1. tf(tf/tfMessages) – Vzájomná transformácia polohy laserového skenera, základne robota a odometrie.
2. scan(sensor\_msgs/LaserScan) – Namerané údaje z laserového skenera, nevyhnutne potrebné na vytvorenie mapy.

**Publikované uzly balíka :**

1. map\_metadata(nav\_msgs/MapMetaData) – Periodicky aktualizovaná výstupná mapa.
2. map(nav\_msgs/OccupancyGrid) - Periodicky aktualizovaná výstupná mapa.
3. ~entropy(std\_msgs/Float64) – Odhad polohy robota. Väčšia hodnota znamená väčšiu neistotu.

## Laserový skener

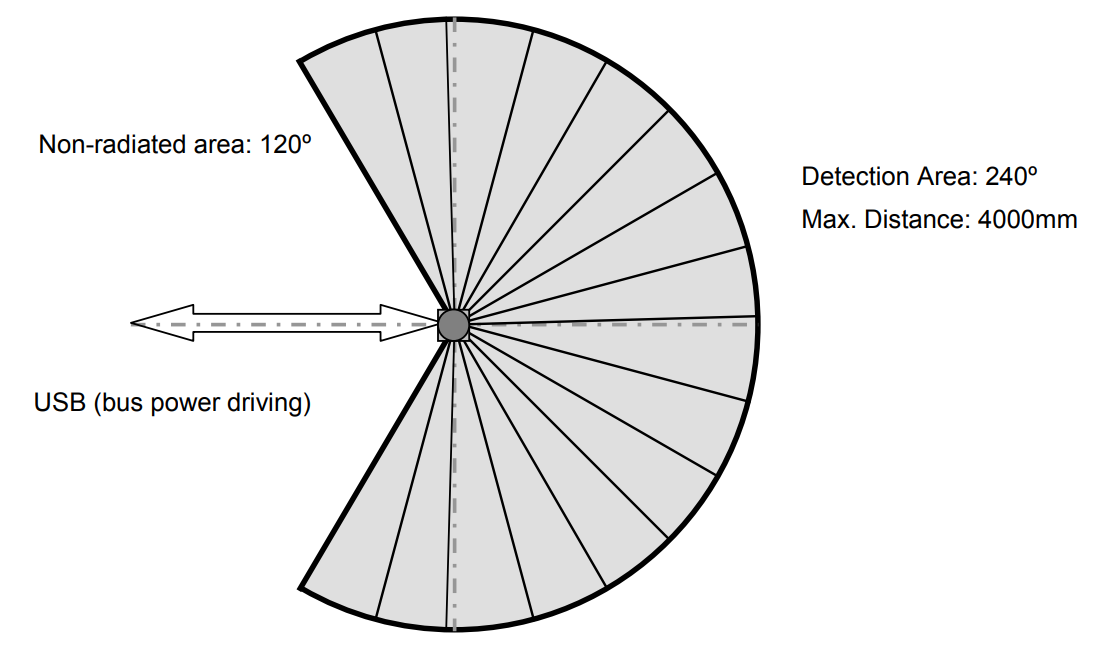
Orientácia robota v priestore je zabezpečená pomocou laserového skenera Hokuyo URG04LX UG01, ktorý je umiestnený v prednej časti podvozku robota. Laserový skener dokáže v priebehu jednej sekundy dostať tisíce dát zo svojho zorného poľa, s nízkou chybou merania. Skener je charakterizovaný nízkou spotrebou, malými rozmermi a hmotnosťou za nízke náklady [6].



Obr. 2 Laserový skener Hokuyo [6]

Hokuyo skener pracuje na princípe infračervenej laserovej diódy s vlnovou dĺžkou λ=785nm, a dokáže zdetegovať predmety v rozsahu 20 mm až 5600 mm. Výrobcom odporúčaná vzdialenosť merania je medzi 60 mm až 4095 mm s presnosťou nameraného údaju na 97%. Nevýhoda laserového skenera spočíva v tom, že nedokáže detegovať aj priehľadné materiály, od ktorých sa lúč neodráža do fotodiódy skenera.

Zorné pole senzora je 240° s inkrementačným uhlom 0,36°, čo predstavuje 683 meraní v jednom cykle. Senzor dokáže merať maximálnou rýchlosťou 10hz. Udávaná presnosť merania skenera výrobcom sú ± 3% z reálne nameranej vzdialenosti. Tento údaj je vyhodnotení z merania na bielom papieri. Princíp merania funguje na báze amplitúdovej modulácie, respektíve na princípe fázového posunu odrazeného a vyslaného lúča [6].



Obr. 3 Pracovný priestor hokuyo skenera [6]

Určenie vlnovej dĺžky modulovaného signálu λ určíme podľa vzorca**:**

(2.1)

kde parameter „f“ predstavuje modulovú frekvenciu a parameter „c“ je rýchlosť svetla. Celkovú vzdialenosť „D´“ ktorú vyslaný lúč prejde definujeme:

(2.2)

kde „L“ predstavuje vzdialenosť snímača fázového posunu od rotujúceho zrkadla umiestnené v senzore. Parameter „D“ predstavuje vzdialenosť snímaného objektu od senzora a parameter „O“ je odmeraný fázový posun medzi odrazeným a vyslaným lúčom.

# Pravdepodobnostný odhad polohy

Pre správnu orientáciu mobilného robota, vo vopred vytvorenej mape prostredníctvom SLAM algoritmov potrebujeme vedieť, kde sa robot v zmapovanom prostredí nachádza. Na to nám slúžia pravdepodobnostné a štatistické odhady stanovenia polohy.

Základné algoritmy určenia odhadu polohy sú napr. Kalmanov filter, Particle filter, kĺzavý priemer, Markova lokalizácia a nami zvolená lokalizácia AMCL (Adaptive Monte Carlos Localization). Proces odhadovania polohy delíme na dva základné kroky – predikcia (konaj) a korekcia (vnímaj).

Predikcia je odhad aktuálnej polohy mobilného robota, vzhľadom na aktuálne meranie z lokalizačného systému (model robota), na predchádzajúcu polohu (stav) robota. Pri predikcii sa zohľadňuje rozptyl odhadu polohy mobilného robota.

Korekcia je spresnený odhad polohy robota, vzhľadom na aktuálne meranie z iných lokalizačných snímačov (napr. model snímača) a odhad aktuálnej polohy mobilného robota z predikcie. Korekcia spresňuje odhad polohy mobilného robota [4].

## Lokalizácia v zmapovanom priestore

### Algoritmus Monte Carlo

Využíva na lokalizáciu Particle filter, takže pravdepodobnostný stav robota, pričom každý zo stavov je reprezentovaný hypotézou. Odhaduje polohu robota na základe pohybu robota a snímania prostredia. Algoritmus je využiteľný len v prostredí, ktoré je statické a nemení sa s časom.

Pri inicializácii nemá robot žiadnu znalosť, kde sa nachádza, preto predpokladá, že sa môže rovnako nachádzať kdekoľvek. Tento princíp sa nazýva aj rovnomerná náhodná distribúcia pravdepodobných stavov robota v priestore. Algoritmus na svoju činnosť potrebuje poznať priestor, v ktorom sa bude orientovať, takže mapu získanú napríklad zo SLAM algoritmov. Pravdepodobné polohy robota sú prevzorkované na základe rekurzívneho Bayesovho odhadu. Následne vyraďuje polohy nekonzistentné s pozorovaním a generuje nové polohy, ktoré sú bližšie ku konzistentným. Na konci by mali všetky pravdepodobné polohy skonvergovať k reálnej polohe robota [4].

### Balíček AMCL

Balíček AMCL (Adaptive Monte Carlos Localization) patrí medzi najčastejšie používané balíčky pre určenie pravdepodobnostného lokalizačného systému, pre roboty pohybujúce sa v 2D mapách. Realizuje adaptívny Monte Carlo lokalizačný prístup, ktorý využíva PF (Particle filter) na sledovanie pozície robota proti známej mape [5].

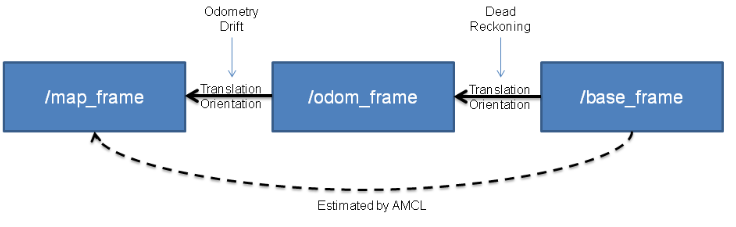
Na využívanie tohto balíčka potrebujeme získavať údaje z laserového skenera, vzájomnú transformáciu robota a hlavne mapu priestoru, v ktorom sa bude robot lokalizovať. Balíček má niekoľko vstupných nastaviteľných parametrov, ktoré si užívateľ zadáva pred spustením algoritmu.

**Prijímané uzly balíka:**

1. tf (tf/tfMessages) – Vzájomná transformácia polohy laserového skenera, základne robota a odometrie.
2. scan (sensor\_msgs/LaserScan) – Namerané údaje z laserového skenera nevyhnutne potrebné na vytvorenie mapy.
3. initialpose ([geometry\_msgs/PoseWithCovarianceStamped](http://docs.ros.org/api/geometry_msgs/html/msg/PoseWithCovarianceStamped.html)) – Inicializačná pozícia robota v mape, s ktorou sa znovu inicializuje PF.
4. map ([nav\_msgs/OccupancyGrid](http://docs.ros.org/api/nav_msgs/html/msg/OccupancyGrid.html)) – vstupná mapa pre lokalizačný algoritmus

**Publikované uzly balíka :**

1. amcl\_pose ([geometry\_msgs/PoseWithCovarianceStamped](http://docs.ros.org/api/geometry_msgs/html/msg/PoseWithCovarianceStamped.html)) – Odhadovaná pozícia robota v mape.
2. particlecloud ([geometry\_msgs/PoseArray](http://docs.ros.org/api/geometry_msgs/html/msg/PoseArray.html)) – Odhad polohy vyhodnotený PF.
3. tf ([tf/tfMessage](http://docs.ros.org/api/tf/html/msg/tfMessage.html)) – Na mapu publikuje transformáciu z odometrie, ktorá môže byť premapovaná pomocou parametra odom\_frame\_id.

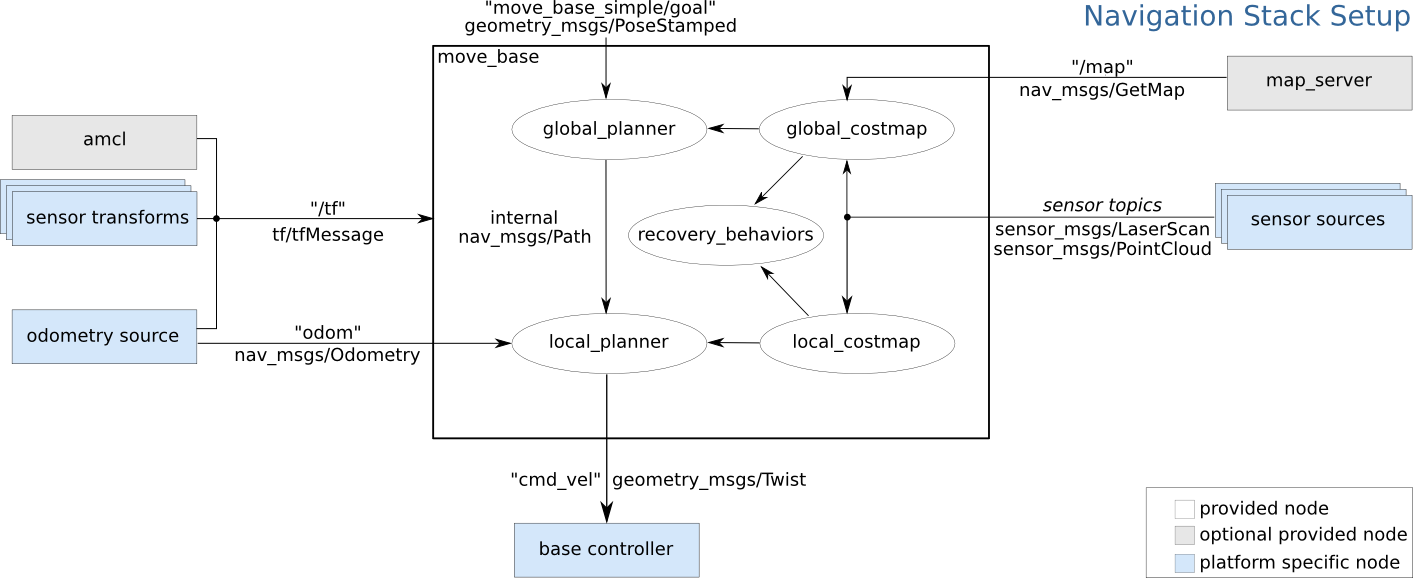


Obr. 4 Princíp AMCL Lokalizácie [5]

# Navigácia

Po zodpovedaní si na otázku, kde som?, to znamená vytvorením mapy preskenovaného prostredia a zorientovaním sa v ňom, pokračujeme na ďalšie dve otázky. Kam chcem ísť?, ako sa tam dostanem?. Na určenie požadovanej pozície robota, zo znamená otázky kam chcem ísť?, som si zvolil prístup prostredníctvom zadania súradníc (x, y) mapy na požadované miesto robota. Je viacero spôsobov ako odpovedať na túto otázku, napríklad posielanie aktuálnej súradnice užívateľa cez GPS mobilného telefónu alebo komunikácia a posielanie polohy medzi dvomi robotmi.

Po zodpovedaní si otázok kde som? a kam chcem ísť?, môžeme pokračovať na hlavnú a najdôležitejšiu otázku, ako sa tam dostanem? Na túto otázku nám slúži navigácia. Navigáciu delíme na dve základné skupiny, a to globálna v tomto prípade navigácia na báze topologickej mapy (global planner) a lokálnu navigáciu (local planner).



Obr. 5 Schéma prepojenia navigačného balíčku [5]

Na Obr. 5 môžeme vidieť schému reprezentujúcu prepojenie požadovaných balíčkov pre bezkolíznu navigáciu mobilného robota. Vidíme ako nám do balíka move\_base vstupujú už vyššie spomínané balíky amcl, map\_server, odom, sensor. Vstupom do balíka je aj move\_base\_simple/goal, - požadovaná pozícia mobilného robota. Výstupom balíka je správa geometry\_msgs/Twist, ktorá určuje rýchlosť a zároveň smer pohybu robota ku stanovenému cieľu.

## Globálna navigácia

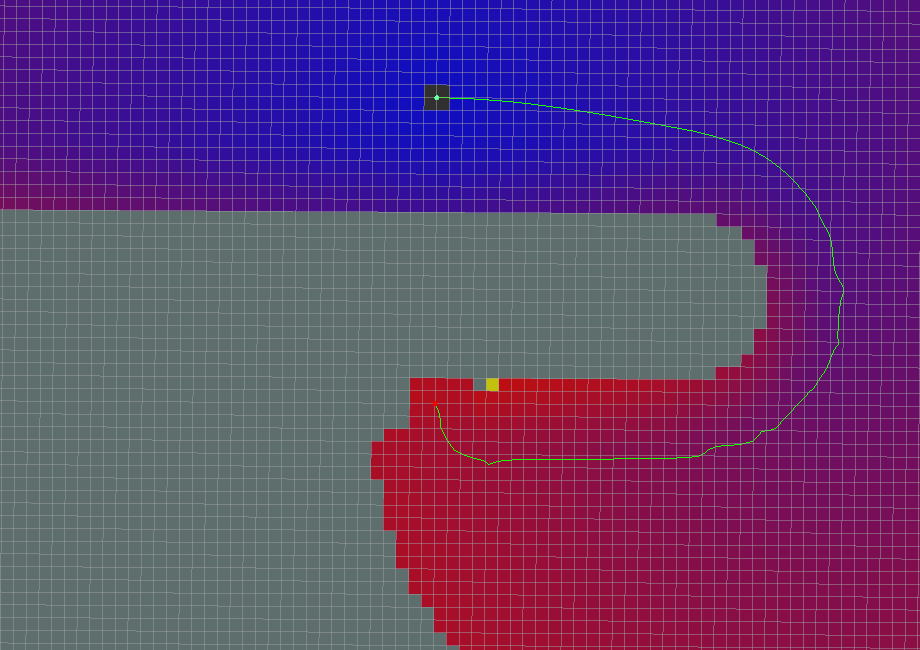
Nami zvolená globálna navigácia na báze topologických máp, taktiež nazývaná aj kvalitatívna navigácia. Vo všeobecnosti je cieľom globálnej navigácie nájsť najoptimálnejšiu dráhu medzi štartom (aktuálnou pozíciou robota) a cieľom (nami zvolenou pozíciou na mape), pričom musia spĺňať určité podmienky. Hlavnou úlohou metódy globálnej navigácie je minimalizovať alebo maximalizovať, tieto podmienky, ktorými sú napríklad spotreba elektrickej energie, dĺžka navrhnutej dráhy, čas pohybu k cieľu alebo vzájomná komunikácia.

Veľkou výhodou použitia topologyckých máp pri globálnej navigácii mobilných robotov, je nízka náročnosť na výpočtový výkon a pamäť. Ďalšou výhodou je jednoduché plánovanie dráhy robota v prostredí. Keďže topologická mapa je grafovou štruktúrou, pri plánovaní cesty v mape sa uplatňujú najmä algoritmy z teórie grafov, ako sú Dijkstra alebo A\* algoritmus.

### Base global planner

Balík poskytuje implementáciu rýchleho a interpolovaného globálneho plánovača navigácie. Dodržiava rozhranie nav\_core, ktoré bolo navrhnuté ako flexibilnejšia náhrada za „navfn“ globálnu navigáciu. Výhodou balíka je veľké množstvo parametrov nastaviteľných užívateľom a veľmi jednoduchá implementácia.

Po určení cieľu, je výstupom balíka globálny plán robota. Okrem určeného cieľa je vstupom do balíka aj vytvorená mapa prekonvertovaná na global\_costmap, čo je vlastne ohraničenie mapy. Mapa je ohraničená, kvôli tomu aby robot nevytváral globálny plán príliš blízko okrajov mapy.



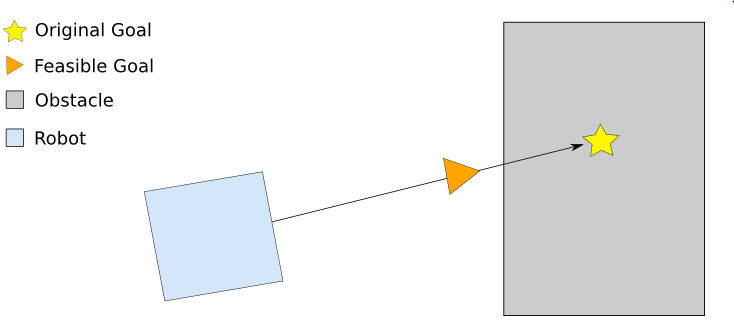
Obr. 6 – Vzorový globálny plán robota

Na Obr. 6 môžeme vidieť vyhodnotenie globálneho plánu. Sivá farba predstavuje nedostupný priestor, napríklad stenu ktorej sa musí robot vyhnúť, modrá farba meniaca sa do červenej predstavuje voľné miesto, kde sa robot môže bezkolízne pohybovať. Posledná zelená farba (plán cesty) určuje smer, akým sa má robot pohybovať ku cieľovému bodu (červenej bodke).

### Carrot planner

Je jednoduchý globálny plánovač, ktorý podobne ako global\_planner dodržiava rozhranie nav\_core. Plánovač berie cieľový bod od externého užívateľa. Kontroluje, či je cieľ určený užívateľom v prekážke, ak áno prechádza sa okolo prekážky a hľadá, čo najkratšiu vzdialenosť od nej. Týmto spôsobom umožňuje robotovi dostať sa, čo najbližšie k cieľovému bodu, ktorý je určený užívateľom.

Výhodou balíka je jeho jednoduchý algoritmus ktorý nevytvára plán cesty potrebný pre navigovanie robota. Bez tohto plánu sa robot nedostane na požadované miesto ale bude mať informáciu o dostupnosti zadaného miesta užívateľom.



Obr. 7 Carrot planner – hľadanie cesty k cieľu

Na Obr. 7 môžeme vidieť zmenu nami zvoleného cieľu ktorý je v prekážke za, cieľ vyhodnotený balíkom carrot\_plannet, ktorý je pre robota dostupný.

## Lokálna navigácia

Pri lokálnej navigácii na rozdiel od globálnej musíme brať do úvahy aj objekty ktoré sa pri prvotnom skenovaní priestoru nenachádzali vo vytvorenej mape, ale v reálnom prostredí sa okolo robota objavili ako napríklad osoby, presunuté objekty (stolička, stôl, krabica), a iné predmety, ktoré je potrebné v reálnom čase obchádzať.

Globálny plánovač nám vytvára len plán na základe mapy, v ktorej nie sú tieto objekty zahrnuté, ale len čistý priestor. Na to aby sme sa vedeli bezkolízne dostať na požadované miesto na mape potrebujeme lokálny plánovač. Pri navigovaní robota podľa globálneho plánu, nám lokálny plánovač upravuje tento plán podľa potreby. To znamená, že ak sa do plánu cesty dostane napríklad krabica, robot musí zareagovať, a okamžite upraviť prípadne vytvoriť nový plán k požadovanému cieľu.

Lokálna navigácia pracuje v ohraničenom priestore, takže robot vidí len v určenom rozsahu, ktorý je limitovaný senzorom robota a veľkosťou mapy. Tento rozsah si vie užívateľ vopred zvoliť, tak isto ako rozsah globálnej navigácie. Medzi najpoužívanejšie algoritmy lokálnej navigácie patria Trajectory Rollout a DWA (Dynamic Windows Approaches).

### Base local planner

Balík poskytuje implementácie z Trajectory Rollout algoritmov na lokálnu navigáciu robota v rovinnom priestore. Vzhľadom na globálny plán, ktorý má robot nasledovať balík vydáva príkazy rýchlosti na robota ktoré majú väčšiu prioritu ako príkazy global\_planner navigácie. Užívateľ môže pri behu vytvárania plánu a navigácie zadávať príkazy na robota. Tento balík podporuje všesmerové aj diferenciálne podvozky.

S cieľom zefektívnenia vyhodnocovania lokálneho plánu sa okolo robota vytvorí mriežková mapa (ohraničenie), podľa ktorej sa upravuje spomínaný globálny plán. Výhodou balíka je jeho jednoduchá implementácia, dokumentácia, a spoľahlivosť.

Balík disponuje množstvom nastaviteľných parametrov od rýchlosti pohybu robota k cieľu, až po nastavenie tolerancie pre zvolený cieľový bod. Lokálny plánovač potrebuje na svoju činnosť získavať dáta zo senzora, odometriu, globálny plán, a mapu prostredia.

**Prijímané uzly balíka:**

1. odom ([nav\_msgs/Odometry](http://docs.ros.org/api/nav_msgs/html/msg/Odometry.html)) – Informácie o odometrii, ktoré poskytujú lokálnemu plánovačovi aktuálnu rýchlosť robota.

**Publikované uzly balíka:**

1. global\_plan ([nav\_msgs/Path](http://docs.ros.org/api/nav_msgs/html/msg/Path.html)) – Časť z globálného plánu, ktorým sa lokálny plánovač snaží riadiť. Používa sa najmä na vizuálne účely.
2. local\_plan ([nav\_msgs/Path](http://docs.ros.org/api/nav_msgs/html/msg/Path.html)) – Lokálny plán robota, je to najlepšia hodnota z celkového vyhodnotenia najlepšej cesty ku cieľu. Používa sa najmä na vizuálne účely.
3. cost\_cloud ([sensor\_msgs/PointCloud2](http://docs.ros.org/api/sensor_msgs/html/msg/PointCloud2.html)) – Používa sa pri lokálnom plánovaní.

### DWA local planner

Dynamic windows approaches znamená v preklade metóda dynamických okien. Poskytuje na rozdiel od base local planner implementáciu algoritmov podľa DWA na lokálnu navigáciu robota v priestore, vzhľadom na globálny plán, ktorý ma robot nasledovať. Balík vydáva príkazy správy na podvozok robota ktoré majú väčšiu prioritu ako správy global\_planner navigácie.

DWA sa radí medzi kategóriu rýchlostných metód pre reaktívne navigovanie mobilných robotov. Pomocou tejto metódy môžeme nájsť najlepšie príkazy rýchlosti a smeru robota, k dosiahnutiu najlepšieho a najrýchlejšieho plánu a zároveň dosiahnutiu vopred určeného cieľa. Balík podobne ako base\_local\_planner disponuje množstvom nastaviteľných vstupných parametrov, ktoré vo veľkej miere môžu ovplyvniť výsledne navrhnutý plán robota.

**Prijímané uzly balíka:**

1. odom ([nav\_msgs/Odometry](http://docs.ros.org/api/nav_msgs/html/msg/Odometry.html)) – Informácie o odometrii, ktoré poskytujú lokálnemu plánovačovi aktuálnu rýchlosť robota.

**Publikované uzly balíka:**

1. global\_plan ([nav\_msgs/Path](http://docs.ros.org/api/nav_msgs/html/msg/Path.html)) – Časť z globálného plánu, ktorým sa lokálny plánovač snaží riadiť. Používa sa najmä na vizuálne účely.
2. local\_plan ([nav\_msgs/Path](http://docs.ros.org/api/nav_msgs/html/msg/Path.html)) – Lokálny plán robota, je to najlepšia hodnota z celkového vyhodnotenia najlepšej cesty ku cieľu. Používa sa najmä na vizuálne účely.

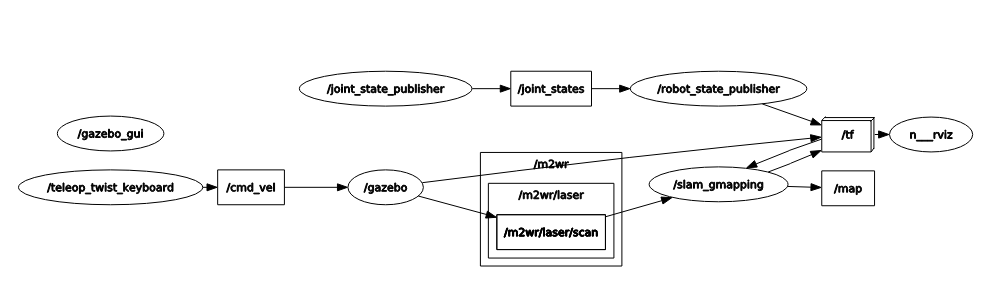
# Implementácia v simulačnom prostredí

 Pri implementácií modulov do riadiaceho systému robota sme si vyberali z dostupných vyššie spomínaných balíčkov ROS. Balíčky sme vyberali na základe overených recenzií užívateľmi.

## Implementácia modulov mapovania

Z dostupných SLAM modulov sme si zvolili slam\_gmapping, je to najčastejšie používaný balíček pre lokalizáciu a mapovanie prejdeného priestoru. Výhodou balíka je jeho jednoduchá implementácia, konfigurácia a dobre vypracovaná dokumentácia.

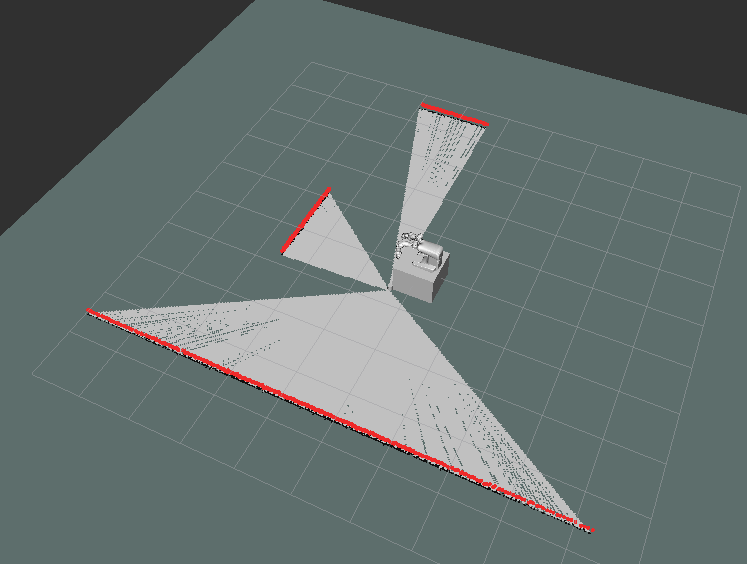
Zvolenú mapu sme si použitím balíka slam\_gmapping a balíka teleop\_twist\_keyboard zmapovali a vytvorili mapu obsadenosti priestoru. Priestor bol zmapovaný prejdením od začiatku bludiska, cez všetky slepé miesta až naspäť na začiatok, aby sa uzavrela slučka mapovania. Následne si ukážeme postupy mapovania a nastavovania zvolených balíčkov. Ako prvé je potrebné pomocou nástroju rqt\_graph dostupného v ROS-e zistiť prepojenie uzlov, ktoré by malo vyzerať podobne ako na Obr.8.



Obr. 8 Prepojenie uzlov pri vytváraní mapy prostredia

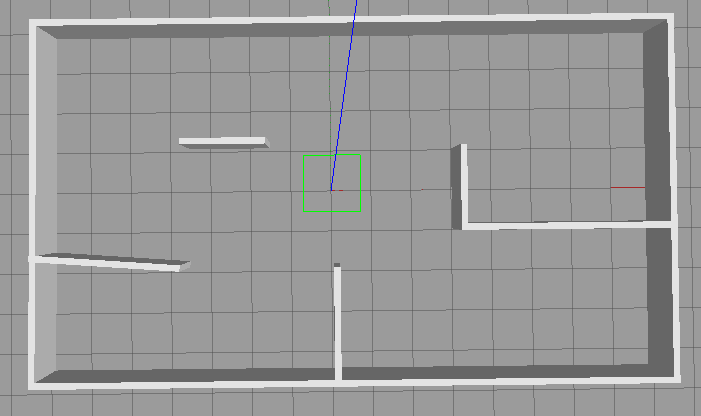
Vidíme ako nám balík teleop\_twist\_keyboard komunikuje s podvozokom robota. Prostredníctvom tohto balíka sme robotovi udávali smer a rýchlosť pri mapovaní priestoru. Súčasne s balíkom ovládania robota beží aj balík pre mapovanie priestoru slam\_gmapping do ktorého vstupujú údaje z hokuyo laserového skenera umiestneného v prednej časti mobilného robota a údaje zo vzájomnej transformácie polohy.

Výstupom z balíka je požadovaná 2D mapa obsadenosti, táto mapa má vopred preddefinované rozlíšenie. Toto rozlíšenie si je dobré vopred nastaviť podľa veľkosti skenovaného priestoru, aby v neskoršom navigovaní nedochádzalo k zbytočnému prepočítavaniu priestoru okolo mapy, ktoré by malo za následok neskoré reagovanie robota pri rôznych situáciách.

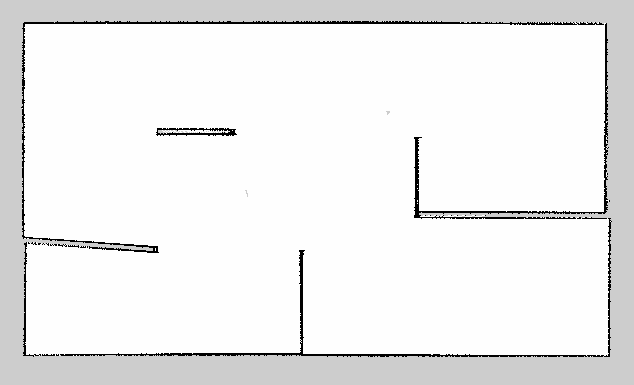


Obr. 9 Prvé skenovanie priestoru bludiska

Na Obr. 9 vidíme prvé skenovanie priestoru z východiskovej polohy robota. Do tejto polohy sa nám robot pri poslednom skenovaní opäť vracia, aby sa nám uzatvorila slučka mapovania. Pri mapovaní je ideálne priestor niekoľkokrát zoskenovať, aby sa vytvoril čo najpresnejší a najkvalitnejší pôdorys mapy.



Obr. 10 Simulačná mapa



Obr. 11 Výsledná mapa bludiska v Rviz

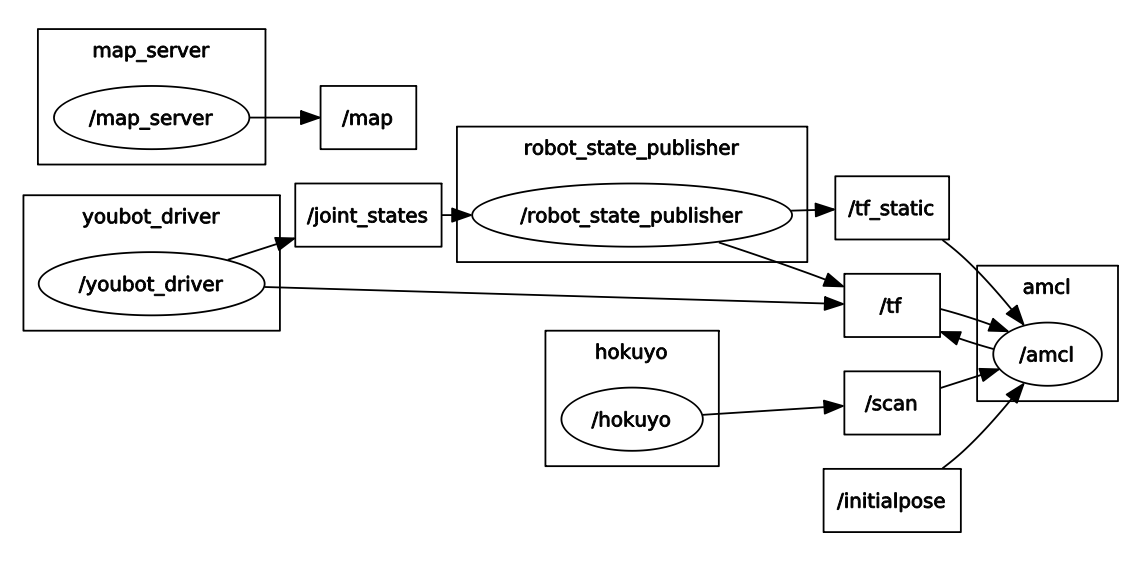
### Výsledky Gmapping mapovania

Z grafického porovnania pôdorysu simulačnej mapy na Obr. 10 a mapy generovanej pomocou balíka Gmapping na Obr. 11 usudzujeme, že mapovací algoritmus pracuje podľa požiadaviek. V reálnom prostredí sa predpokladá viac zdrojov chýb a odporúčame balíku adekvátne nastaviť jeho konfiguračný súbor.

## Implementácia modulov lokalizácie

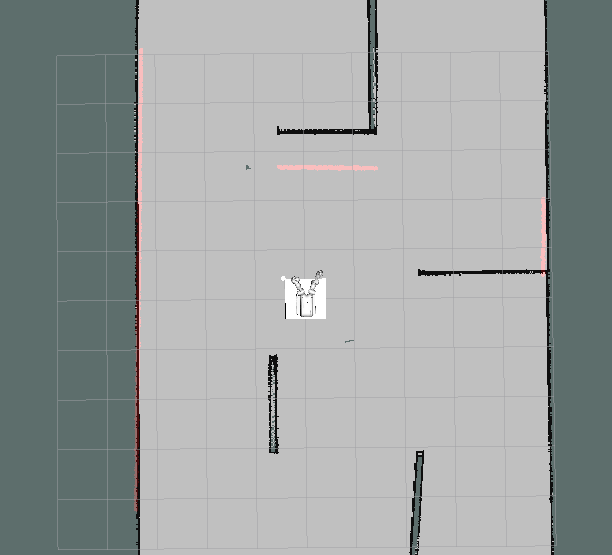
Z dostupných lokalizačných balíčkov sme si vybrali balík amcl, hlavne kvôli jeho jednoduchej implementácii a dostupnej dokumentácie. Balík na svoju správnu funkcionalitu potrebuje prijímať údaje z laserového skenera, mapu prostredia, inicializačnú pozíciu a vzájomnú transformáciu polohy. Inicializačná pozícia robota sa vytvorila pri skenovaní mapy prostredia balíkom gmapping.

Pred samotným testovaním balíka je potrebné v aplikácii rqt\_graph skontrolovať vyššie spomínané požiadavky na správne fungovanie balíka, ktoré by malo vyzerať podobne ako na Obr. 12. Na obrázku môžeme vidieť, že nám mapa beží na lokálnom serveri, z ktorého si ju balík amcl preberá.



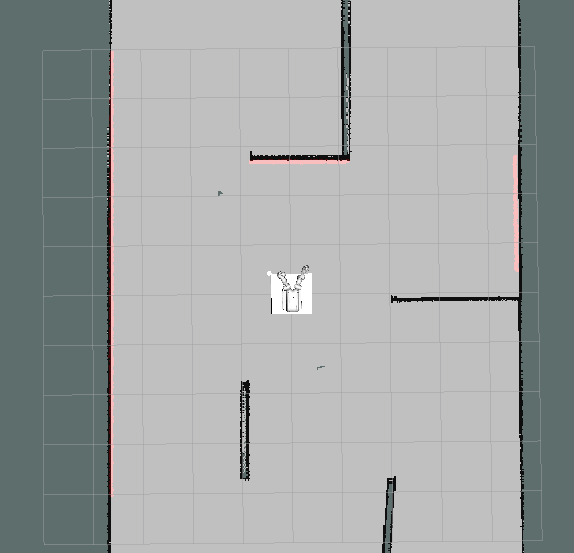
Obr. 12 Prepojenie uzlov pri lokalizácii sa v mape

Pri spúšťaní lokalizačného balíka je dobré robota umiestniť na vopred známu inicializačnú pozíciu, v našom prípade viď Obr. 10. V prípade veľmi vzdialeného umiestnenia od tejto pozície by balíku amcl trvalo príliš dlho zorientovať sa v zmapovanom priestore, v horšom prípade by sa mu to ani nemuselo podariť. Balík má možnosť konfigurácie viacerých parametrov, akými sú napríklad laser\_min\_range, laser\_max\_range, initial\_pose\_x, initial\_pose\_y a pod. Prvú lokalizáciu v bludisku môžeme vidieť na Obr. 13.



Obr. 13 Prvá lokalizácia zmapovaného priestoru

Na Obr. 13 si môžeme všimnúť, že aj napriek dodržaniu inicializačnej pozície vznikla nepresnosť prvej lokalizácie priestoru balíkom amcl. Túto nepresnosť vieme postupným pohybovaním robota po mape zmenšiť až na zanedbateľnú. Po dostatočnom počte prejdení mapy, získavame finálnu lokalizáciu mapy ako môžeme vidieť na Obr. 14.



Obr. 14 Finálna lokalizácia zmapovaného priestoru

### Výsledky AMCL lokalizácie

Robot sa pomocou balíku AMCL dokázal lokalizovať dostatočne rýchlo. Výhodné je inicializovať robota v bode, v ktorom bolo zahájené mapovanie pomocou Gmapping balíka.

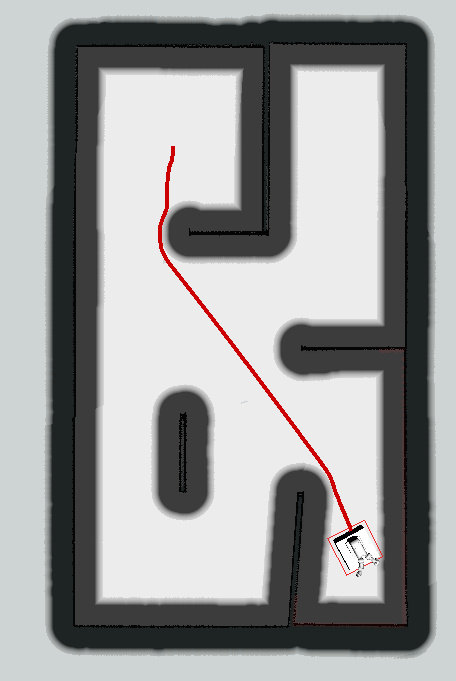
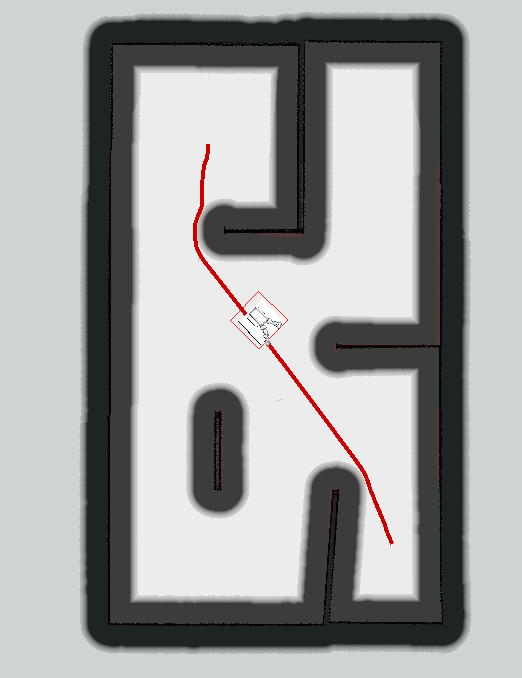
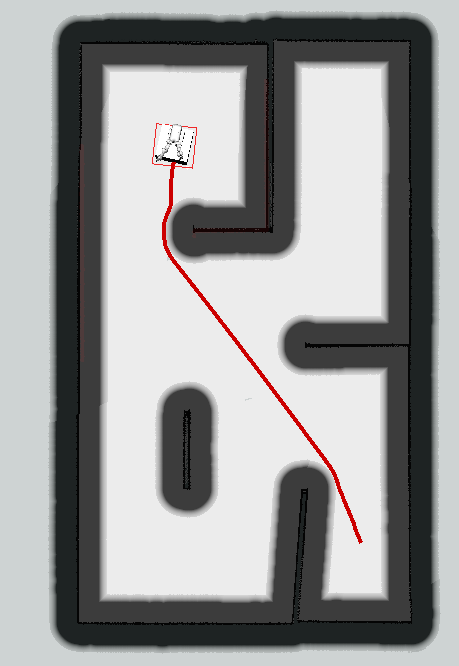
## Implementácia modulov navigácie

Pri výbere z dostupných navigačných modulov sme sa rozhodovali na základe spoľahlivosti a zložitosti implementácie jednotlivých modulov. Tieto moduly sme si vopred rešeršovali a sú rozpísané v sekcii 5 Navigácia. Na rozdiel od vyššie spomínaných a testovaných modulov mapovania a lokalizácie sa pri navigácií vyberajú dva moduly, a to zvlášť pre globálnu a lokálnu navigáciu. Testovanie navigačných modulov prebiehalo v umelo vytvorenom simulačnom bludisku. Neoddeliteľnou súčasťou navigácie je mapa prostredia vygenerovaná zvoleným balíkom Gmapping a AMCL lokalizácia v prostredí.

Pre globálnu navigáciu sme si zvolili navigačný balík base\_global\_planner, ktorý nám zabezpečí bezkolízne globálne navigovanie sa  priestoroch bludiska. Požadovaným vstupom balíka je mapa prostredia v ktorej sa má robot navigovať, odometria obdržaná balíkom amcl a samozrejme súradnice cieľovej pozície. Požadovaným výstupom sú správy na podvozok robota ktoré ho majú bezkolízne navigovať k cieľu. Je dôležité si uvedomiť, že globálna navigácia zabezpečuje len bezkolízne plánovanie navigácie a nezaoberá sa údajmi z laserového skenera. V prípade prekážky v bludisku ktorá by nebola zaznamenaná v mape by do nej robot vrazil.

Pri lokálnej navigácii sme si podobne ako pri globálnej vyberal zo spoľahlivých voľne dostupných navigačných balíkov. Zvolili sme si balík DWA\_local\_planner, ktorý sme prepojili s balíkom base\_global\_planner, tým pádom vznikol navigačný uzol nazývaný aj ako nav\_core, ktorého schému môžeme vidieť na Obr. 5. Medzi hlavné vstupujúce údaje do lokálneho navigačného balíka patria, plán globálnej navigácie, údaje z laserového skenera, mapa prostredia a odometria. Správy na podvozok robota, ktorý sa má bezkolízne dopraviť k zvolenému cieľu sú v prípade lokálnej navigácie, vždy predradené správam z globálnej navigácie. Toto zabezpečenie pomáha v prípadnom obmedzení pohybu robota po globálnom pláne prepísať aktuálny globálny plán na lokálny pomocou údajov z laserového skenera, a tým sa vyhnúť nežiaducej kolízii robota. Tieto obmedzenia môžu byť spôsobené napríklad osobami, predmetmi, či inými pohybmi pred robotom. Laserový skener je v našom prípade inštalovaný v prednej časti robota. To znamená, že ak by niekto narazil do robota zo zadnej časti alebo príkreho boku, robot by kolíziu nezaznamenal a teda nezmenil plán k cieľu.

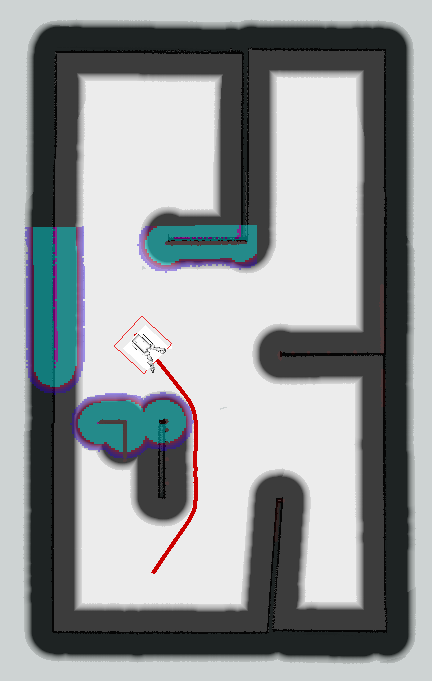
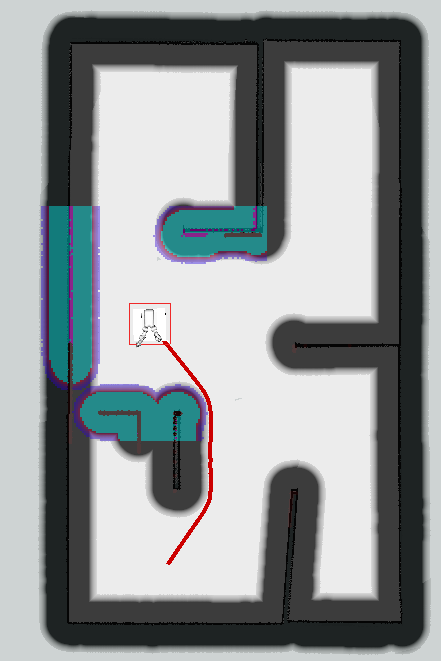
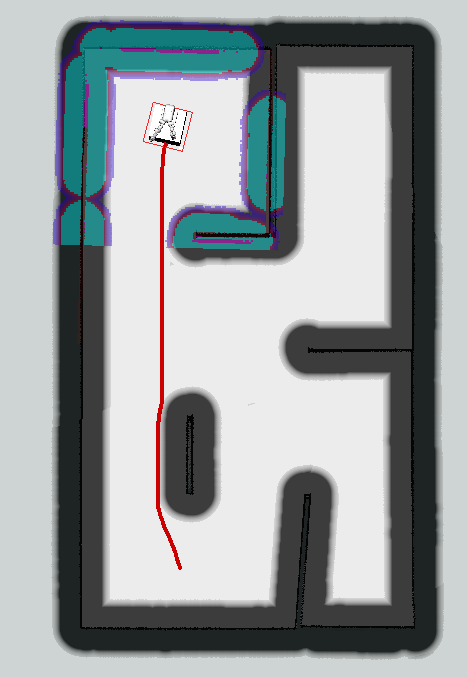
### Výsledky Globálnej navigácie



Obr. 15 Výsledky globálnej navigácie

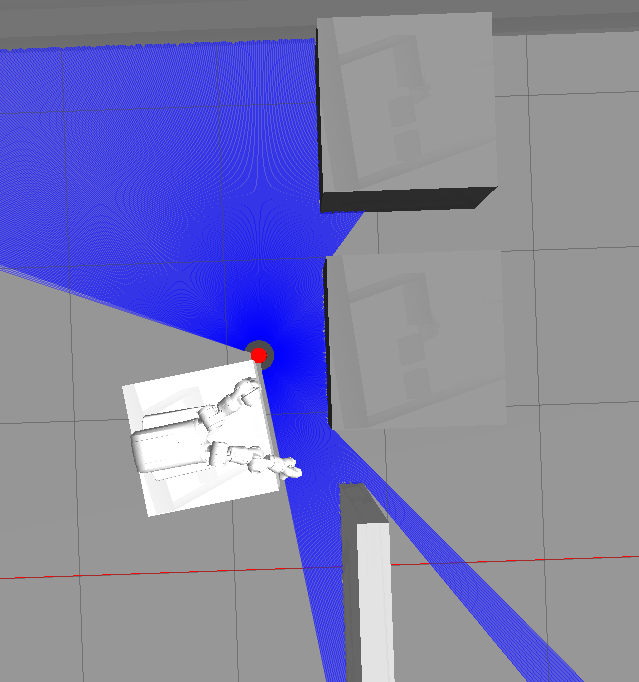
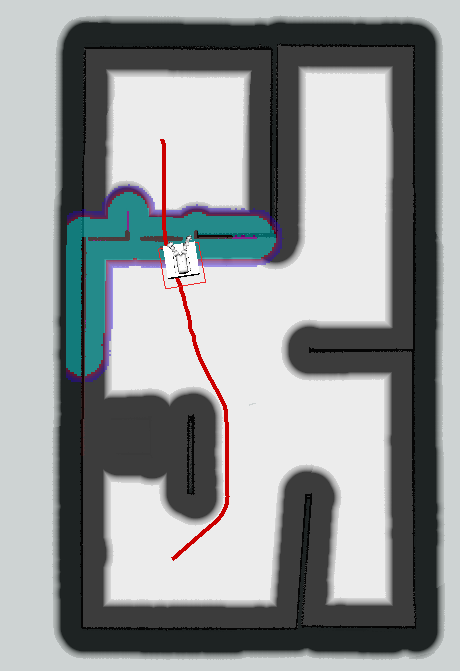
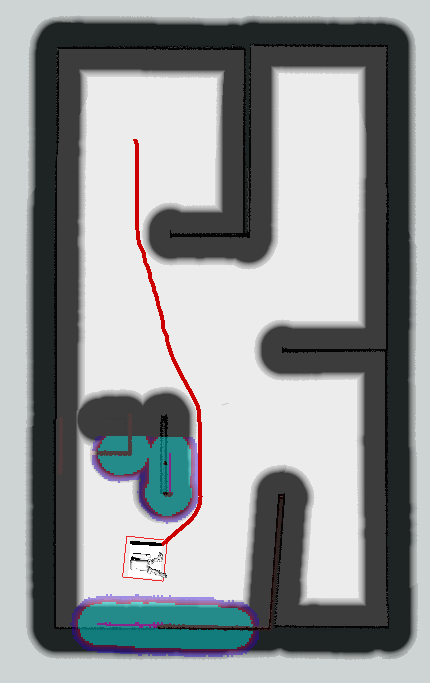
Na Obr. 15 môžme vidieť ako nám globálny plánovač vytvoril plán cesty k požadovanej pozícii. Plánovač je nastavený tak aby vytváral dráhu čo najbližšie k hrane costmapy. Robot kopíruje plán cesty podľa požiadaviek.

### Výsledky Lokálnej DWA navigácie



Obr. 16 Výsledky lokálnej navigácie pri vložení prekážky

Na Obr. 16 môžeme vidieť ako nám balík DWA po vložení prekážky prepočítal plán cesty a tým pádom bezkolízne uriadil robota na požadovanú pozíciu. Veľkosť lokálnej costmapy(modrá farba) sme nastavili na 6x6 metrov.



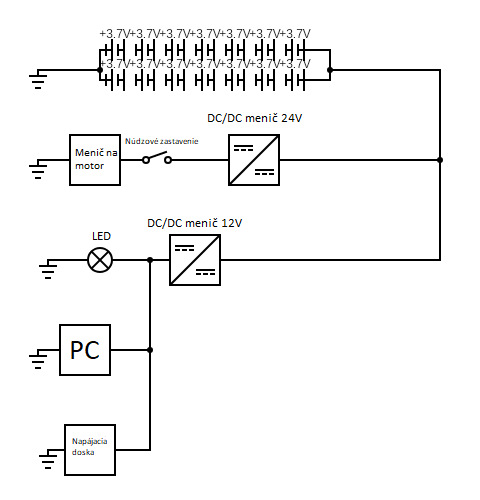
Obr. 17 Výsledky lokálnej navigácie pri pascovej situácii

Pri simulácii pascovej situácie sme poslali robota na nedosiahnuteľnú pozíciu. Na Obr. 17 v poslednom stĺpci môžme vidieť ako nám robot zastal pred neprekonateľnou prekážkou.

## Návrh napájania robota

|  |  |  |  |
| --- | --- | --- | --- |
| **Spotreba** | **Kludové zaťaženie** | **Špička** | **Napätie** |
| PC | 1,5A | 2,8A | 12V |
| Meniče | 3A | 6A | 24V |
| LED | 2A | 2A | 12V |
| Napájacia doska | 240mA | 240mA | 12V |

Pre napájanie robota sme vybrali batérie typu Li-on , keďže ponúkajú najväčšiu hustotu elektrickej energie na kilogram. Konkrétne sa jedná o typ 18650. S kapacitou 3,7V a 2600mAh. Tieto batérie môžeme ukladať do rôznych blokov, pre nášho robota som vybral sériovo paralelne zapojenie. V budúcnosti je možné toto zapojenie rozšíriť bez veľkých úprav.



Obr 18. Schéma zapojenia bateriek

Pre nášho robot boli vybraté 2 meniče zodpovedajúce spotrebe s rezervou pre budúce rozšírenie.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Meniče DC/DC** | **Vstupné napätie** | **Výkon** | **Výstupne napätie** | **Výstupný prúd** |
| SD-200B-24 | 19-36V DC | 200W | 24V | 0~8,4A |
| SD-100B-12 | 18-36V DC | 100W | 12V | 0~8,5A |

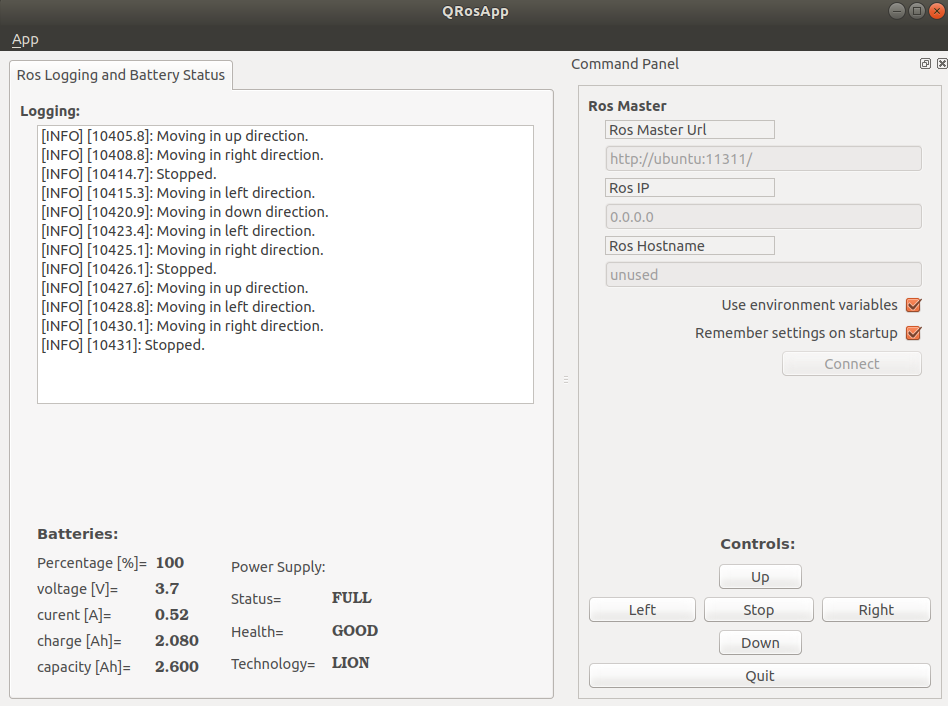
<http://www.meanwell.co/napajacie-zdroje/dc-dc-menice/sd-priemyselne/sd-200/show/sd-200b-24>

<http://www.meanwell.co/napajacie-zdroje/dc-dc-menice/sd-priemyselne/sd-100/show/sd-100b-12>

## Implementácia HMI

Na prácu s robotom sme vyvinuli aj používateľské rozhranie, v ktorom je možné ovládať robota v simulácií, alebo sledovať status bateriek.

Po spustení príkazom **rosrun tp\_gui tp\_gui** uvidíme obrazovku ako je uvedená na Obr. 17. Do textového poľa označeného ako *Ros Master Url* je potrebné zadať adresu, na ktorej beží master node a pod ním je potrebné vyplniť *Ros IP*. Tieto údaje je možné získať aj zo systémových premenných, pokiaľ odklikneme *Use environment variables*.



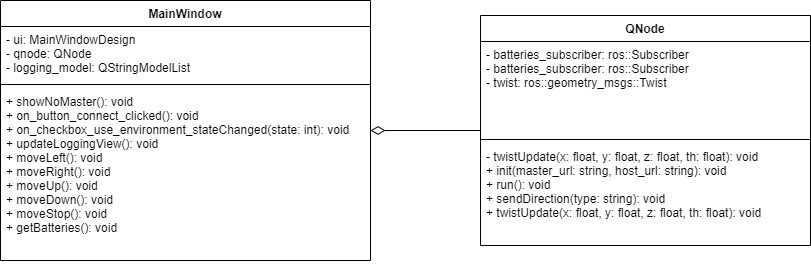
Obr. 19 Aplikácia pre ovládanie robota a sledovanie priebehu pohybu a stavu bateriek

V ľavej hornej časti sú zalogované spúšťané príkazy na riadenie robota, pod týmto oknom je zoznam údajov o baterkách, ktoré sme pre účely testovania získavali z balíka **sensor\_msgs**

V pravom panely je pod ovládaním na pripojenie k master node-u a ovládanie robota, s ktorým ho môžeme navigovať po mape. Môžeme ho ovládať v štyroch smeroch a jeho pohyb následne vidieť v *gazebo* simulácií.

Toto rozhranie sme implementovali v Qt Creator IDE ako QWidget aplikácia, ktorá komunikuje s ostatnými balíkmi štandardne prostredníctvom publisher/subscriber vzoru. Pri jeho tvorbe sme vychádzali z dostupného balíka **ros-melodic-qt-gui.**

V samotnej implementácii sme pracovali primárne s dvoma triedami, ktoré sú uvedené na Obr. 19. V triede **MainWindow** sa nachádza časť súvisiaca s prezentačnými prvkami, teda všetky rôzne viditeľné prvky v aplikácií a v triede **QNode** je implementovaná samotná logika komunikácie s ostatnými uzlami.

****

Obr. 20 softvérový Class diagram tp\_gui

Pri tejto implementácií sme využili koncept signálov a slotov, ktoré Qt využíva pre komunikáciu medzi dvoma objektmi, ide o jednoduché prepojenie medzi objektom, ktorý spustí nejakú udalosť, napr. kliknutie na tlačidlo, a objektom, ktorý túto udalosť má nejako spracovať, napr. odoslanie správy.

Spustenie aplikácie: **rosrun tp\_gui tp\_gui**

# Záver

## Návod na spustenie

Po nainštalovaní všetkých potrebných balíkov a naklonovaní priloženého repozitáru sa pripravené softvérové riešenie spúšťa nasledujúcou sekvenciou príkazov.

**Spawnem svet (nezabudni si nahrat bludisko)**

roslaunch gazebo\_ros empty\_world.launch

**Spawnem robota**

roslaunch m2wr\_description spawn.launch

**Rviz pre vizualizaciu a nastavenie pozadovanej polohy robota**

roslaunch m2wr\_description rviz.launch

**Mapovaci balicek**

roslaunch gmapping slam\_gmapping\_pr2.launch

**Ovladanie robota**

rosrun teleop\_twist\_keyboard teleop\_twist\_keyboard.py

**Prikaz na ulozenie mapy**

rosrun map\_server map\_saver -f test.yaml

**Prikaz na nahratie mapy**

rosrun map\_server map\_server test.yaml.yaml

**Prikaz na spustenie lokalizacie**

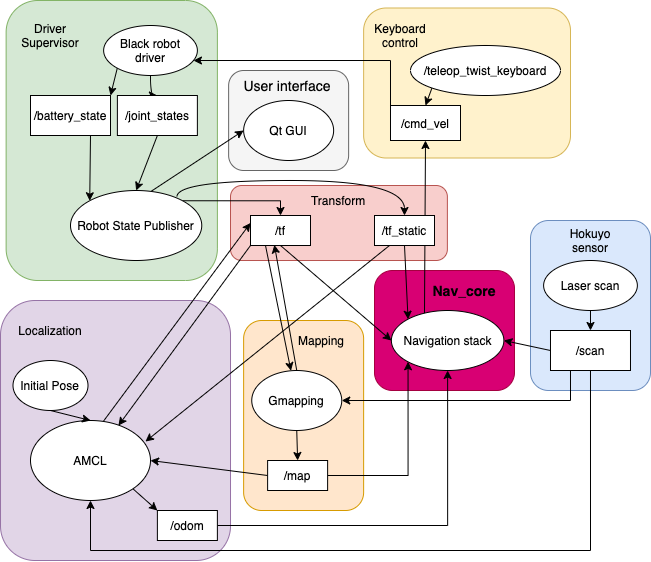
roslaunch amcl amcl\_diff.launch #alebo toto > rosrun amcl amcl scan:=m2wr/laser/scan

**Prikaz na spustenie navigacie**

roslaunch move\_base move\_base.launch

Následne si už len pomocou klávesnice prebehneme robotom po mape aby sa zlokalizoval a na vrchnej lište v Rviz si zvolíme pomocou tlačidla 2DNavGoal cieľovú pozíciu robota.

## Zhrnutie



Obr. 20 Kompletný diagram ROS stacku pre čierneho robota

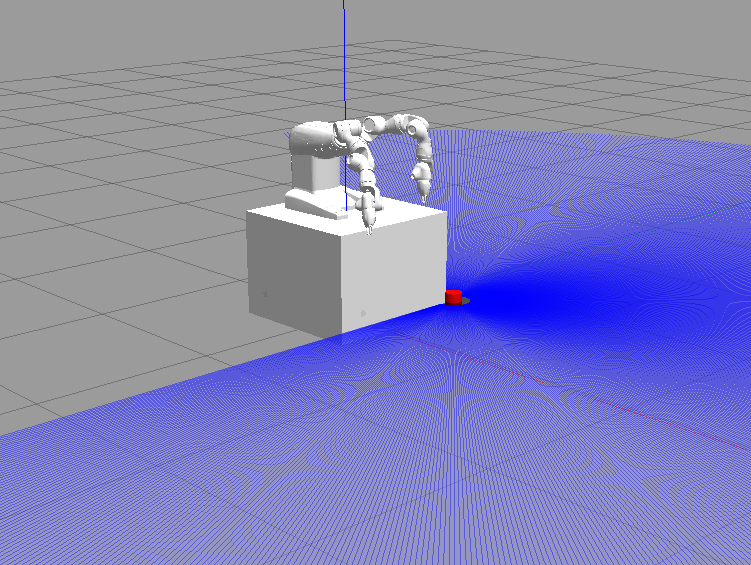
Cieľom projektu bolo pripraviť ROS softvérový stack pre bezkolíznu navigáciu čierneho robota.

Prvým krokom k vytvoreniu bezkolíznej navigácie je mapovanie prostredia. V našom prípade sme ho realizovali pomocou balíka Gmapping, Na základe laserového scanu a výstupu transformačnej funkcie nám vygeneroval mapu prostredia. Balík Gmapping sme si zvolili, pretože je v porovnaní s ostatnými balíkmi výrazne rýchlejší a presnejší.

Po úspešnom zmapovaní prostredia sa v ňom potrebujeme lokalizovať. Lokalizácia v zmapovanom prostredí prebieha prostredníctvom AMCL algoritmu inak známeho aj pod názvom Monte Carlo. Vstupom do AMCL je mapa prostredia, v ktorom sa chceme lokalizovať, údaje z laserového skenera, výstup transformačnej funkcie a inicializačná pozícia robota. Výstupom je odhadovaná poloha robota v prostredí, ktorá sa na postupným pohybovaním robota spresňuje.

Keďže už poznáme mapu prostredia a polohu robota, disponujeme všetkými potrebnými údajmi pre navigovanie robota. Globálna navigácia je realizovaná pomocou balíka base\_planner a lokálna navigácia pomocou dwa\_local\_planner. Oba balíky bolo pre správnu funkčnosť potrebné správne navoliť parametre konfiguračných súborov. Pri nasadzovaní v reálnych podmienkach predpokladáme, že bude potrebné dodatočné ladenie týchto parametrov. Pre lepšie uchopenie problematiky odporúčame preštudovať článok Makkalamuho Malhomäkki[8]. Výstupom navigačného stacku je signál /cmd\_vel, ktorý smeruje do driveru robota, kde sa pomocou neho ovládajú pohony podvozku. Výsledkom je úspešné kopírovanie naplánovanej trajektórie cesty. Ako je demonštrované na Obr. 16 a Obr. 17 robot úspešne obchádza rôzne prekážky a je pripravený na pascové situácie.

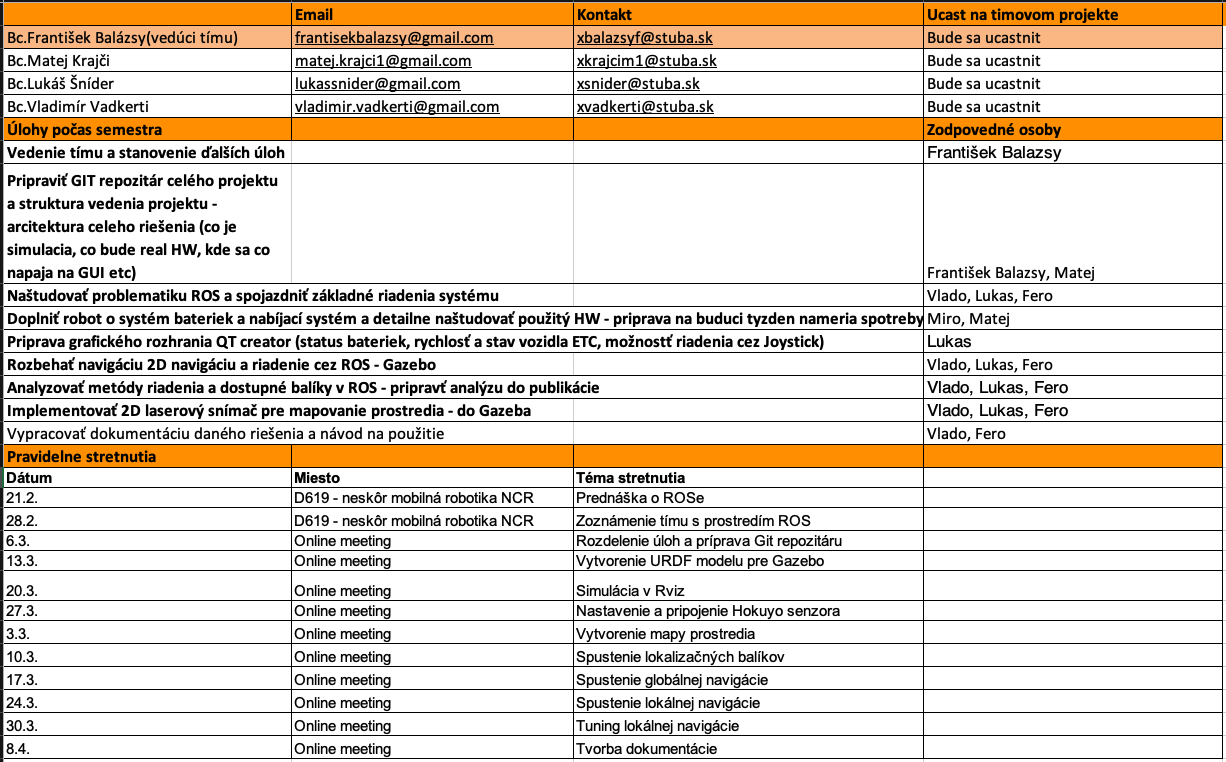
Pre čo najjednoduchšie nasadenie na reálneho robota sme pripravili diagram, ktorý ilustruje jednotlivé prepojenia potrebných balíkov. Diagram na Obr. 18 demonštruje aplikáciu na reálnom robotovi. V simulačnom prostredí sme si pre reprezentáciu reálnych signálov vytvorili URDF model, ktorý v prostredí Gazebo slúžil na generovanie /joint\_states a /scan topicov. V prípade reálnej aplikácie je potrebné tieto signály nahradiť skutočnými hodnotami so senzorov, pričom odporúčame čerpať z informácií dostupných v dokumentácii čierneho robota.



Obr. 21 URDF model čierneho robota

## Harmonogram stretnutí a rozdelenie úloh

Stretnutia prebiehali v učebni D619 a neskôr v rámci súčasnych podmienok formou online hovorov. Témy diskusií a priebeh projektu je uvedený na Obr. 19.



Obr. 22 Rozpis tém jednotlivých stretnutí a úloh členov tímu

Literatúra

1. M. Kohút a kol. 2015. Začíname s ROS-om. In Atp journal. ISSN 1335-2237, 2015, roč.17, č.1.
2. JasonM. O´Kance 2014. A Gentle Introduction to ROS.online.Columbia : JasonM. ISBN 978-14-92143-23-9
3. Autor neuvedený: Robot Operating System [Online]. Citované 20.12.2018.

Dostupné na < <https://en.wikipedia.org/wiki/Robot_Operating_System>>.

1. DUCHOŇ F.: Lokalizácia a navigácia mobilných robotov do vnútorného prostredia. Slovenská technická univerzita v Bratislave FEI.
2. Autor neuvedený: ROS.Org [Online]. Citované 20.12.2018.

Dostupné na <[http://wiki.ros.org/](http://wiki.ros.org/nav_core)>.

1. HOKUYO AUTOMATIC CO 2014 URG-04LX-UG01-Specifications [online]. [citované 22.12.2017]. Dostupné na <<http://hokuzo-aut.jp>>.
2. KAIYU ZHENG : 2016 ROS Navigation Tuning Guide. Dostupné na: <<https://arxiv.org/pdf/1706.09068.pdf?fbclid=IwAR2XeIcfrX6gyICUjd1HlC5kC2qTzxNj2H5kHOGLOBUZAjlVVI-ArDS7qH8>>