UNIVERSITETI I PRISHTINËS "HASAN PRISHTINA"

Fakulteti i Inxhinierisë Mekanike

Prishtinë

UNIVERSITETI I PRISHTINËS "HASAN PRISHTINA"
FAKULTETI I INXHINIERISË MEKANIKE
PRISHTINË

Pranuar me: 04.07.2022			
Nj. org.	Numër	Shtojca	Vlera
08	1513	—	—

Këshillit të Fakultetit të Inxhinierisë Mekanike

P r i s h t i n ë

Në bazë të vendimit nr. 1162/2-2 të datës 13.09.2021, të Këshillit të Fakultetit të Inxhinierisë Mekanike në Prishtinë është formuar komisioni në përbërje:

1. Prof. Dr. Arbnor Pajaziti, *kryetar*
2. Prof. ass. Dr. Xhevahir Bajrami, *mentor*
3. Prof. Dr. Ahemt Shala, *anëtar*

Për vlerësimin e punimit Master me titull **"Lokalizimi dhe navigimi i robtit mobil: Shembull Turtlebot 2i"** të kandidates bachelor Agnesa Prebreza.

Pas kontrollimit të punimit të lartpërmendur Komisioni jep këtë:

R A P O R T

Punimi Master me titull titull **"Lokalizimi dhe navigimi i robtit mobil: Shembull Turtlebot 2i"** është hartuar në 6 (gjashtë) kapituj kryesor dhe është ilustruar me figura, grafe dhe tabela të nevojshme.

Në përmbledhje/abstrakt kandidatja ka mbuluar çdo hap me sqarime, diagrame, figura dhe rezultate praktike. Në përfundim ka arritur të përshkruajë se si mund të shkruhen programe të kostumizuara, në gjuhën Python, për detyra më specifike për Turtlebot.

Punimi është ndarë në gjashtë kapituj kryesorë:

Në kapitullin e parë janë prezantuar qëllimet e hulumtimit,e po ashtu në pjesën e parë të punimit është diskutuar lidhur me robotikën në përgjithësi dhe me stadin aktual të zhvillimit të robotikës, me fokus të veçantë në robotët mobil, ku janë sqaruar mjaft qartë: Roboti mobil Turtlebot 2i, baza mobile Kobuki, Kamera 3D SR300, Kamera 3D Orbbec Astra, Intel NUCBOXNUC6CAYH dhe Krahu Robotik Pincher MK3.

Në kapitullin e dytë kandidati fillimisht ka paraqitur informatat e përgjithshme në lidhje me ndërtimin e një modeli matematik që përshkruan robotët mobil. Nga ky modul mund të shohim ngjashmërinë mes trajtimit matematik të robotëve mobil dhe manipulatorëve robotik.

Pas kësaj është paraqitur shumë qartë: Modelimi i robotëve mobil – Shembull i Turtlebot 2i Robotët mobil me rrota, përshkrimi i pozitën së robotit mobil, modeli dhe kinematika e Turtlebot 2i , modelet e thjeshtuara kinematike për robotët mobil.

Kapitullin e përmbyll me disa diagrame të cilat ilustrojnë pjesën teorike.

Po ashtu janë përshkruar informatat e përgjithshme në lidhje me robotin mobil Turtlebot 2i. Në këtë kapitull janë saktësuar sensorët me të cilët është pajisur ky robot dhe po ashtu se si mund të përdoret.

Në kapitullin e tretë është diskutuar për sistemin operativ të robotit (ang. ROS). Ky është një sistem operativ në vete, që është i destinuar për të shërbyer si substrat mbi të cilin mund të nderohen platforma të ndryshme robotike mobile. Është diskutuar struktura e këtij sistemi operativ dhe po ashtu se si mund të përdoret ky sistem operativ në rastin e robotit Turtlebot.

Në vijim ka analizuar në detaje Sistemi Operativ i Robotit (ang. ROS - robot operating system), arkitektura dhe konceptet e ROS, niveli i sistemit të skedarëve (ang. ROS Filesystem level), niveli i llogaritjes (ang. Computation graph level), niveli i komunitetit (ang. Community level), installimi i ROS, ROS dhe Turtlebot.

Në kapitullin e katër është sqaruar shumë se navigimi është pjesa kryesore e këtij punimi. Navigimi si problematikë është e trajtuar mjaft detalisht përgjatë viteve në literaturë. Në rastin

e kandidates shohim se cilat janë qasjet bashkëkohore në zgjidhjen e problemit të lokalizimit. Në këtë pjesë shohim po ashtu se si është i implementuar SLAM algoritmi në rastin e robotit Turtlebot 2i, dhe pse ky algoritëm është mjaft i njohur në pjesën e robotikës mobile.

Në kapitullin e pestë shohim se si kandidatja ka bërë ndërlidhjen mes Matlab dhe Turtlebot, duke pasur kështu një kanal komunikimi mes këtyre dy komponentëve që vepron në kohë reale, ku është treguar një shembull lidhjeje mes Matlab dhe Turtlebot 2i, si dhe Lokalizimi me algoritmin Monte Carlo.

Në kapitullin e gjashtë në pjesën e fundit të punimi kandidatja ka hartuar për programin e robotëve mobil me Python. Python, duke qenë aktualisht një nga gjuhët programuese me zhvillimin më të hovshëm, ka gjetur një aplikim jashtëzakonisht të madh edhe në pjesën e robotëve mobil. Në fakt, pjesa dërrmuese e librarive bazë të ROS janë të shkruara pikërisht në Python. Për ta konkretizuar këtë hartim janë marrë tre shembuj demonstrues dhe për qëllime testuese, se si mund të shkruhet një program me Python për robotin Turtlebot. Këtu kemi të bërë me implementin e algoritmeve të kërkimit Dijkstra dhe A*.

Në fund ka dhënë rezultatet, konkluzionet dhe literaturën e shqyrtuar.

PËRFUNDIM

Në bazë të të dhënave të përshkruara më lartë, Komisioni për Vlerësimin e punimit Master konsideron se punimi është hartuar në nivel të duhur, i pasqyruar me figura, diagrame dhe tabela të nevojshme. Prandaj komisioni i propozon Këshillit të Fakultetit të Inxhinierisë Mekanike në Prishtinë, që punimin Master, me titull titull **"Lokalizimi dhe navigimi i robotit mobil:Shembull Turtlebot 2i"** , të kandidates *Bachelor* Agnesa Prebreza, ta aprovojë si punim për Master, dhe ta jepë në diskutim publik.

Prishtinë, 04 Korrik 2022

Komisioni:

1.Prof. Dr. Arbnor Pajaziti, *kryetar*_____

2.Prof. ass. Dr. Xhevahir Bajrami, mentor _____

3.Prof. Dr. Ahmet Shala, *anëtar*_____

UNIVERSITETI I PRISHTINËS "HASAN PRISHTINA"

FAKULTETI I INXHINIERISË MEKANIKE

DEPARTAMENTI: MEKATRONIKË-MASTER

PUNIM DIPLOME

TEMA: "Lokalizimi dhe navigimi i robotit mobil: Shembull Turtlebot 2i"

Mentori: Kandidatja:

Prof. Ass. Dr. Xhevahir Bajrami **Bsc. Agnesa Prebreza**

Prishtinë, 2022

PËRMBAJTJA

1. Hyrje 6
2. Roboti mobil Turtlebot 2i 12
   2.1 Baza mobile Kobuki 12
   2.2 Kamera 3D SR300 13
   2.3 Kamera 3D Orbbec Astra 14
   2.4 Intel NUCBOXNUC6CAYH 15
   2.5 Krahu Robotik Pincher MK3 16
3. Modelimi i robotëve mobil – Shembull i Turtlebot 2i 18
   3.1 Robotët mobil me rrota 18
   3.2 Përshkrimi i pozitën së robotit mobil 20
   3.3 Modeli dhe kinematika e Turtlebot 2i. 22
   3.4 Modelet e thjeshtuara kinematike për robotët mobil 25
4. Sistemi operativ i robotit (ang. ROS - robot operating system) 27
   4.1 Arkitektura dhe konceptet e ROS 28
      4.1.1 Niveli i sistemit të skedarëve (ang. ROS Filesystem level) 30
      4.1.2 Niveli i llogaritjes (ang. Computation graph level) 30
      4.1.3 Niveli i komunitetit (ang. Community level) 32
   4.2 Installimi i ROS 32
   4.3 ROS dhe Turtlebot 32
5. Navigimi i robotëve mobil – Algoritmi SLAM 35
   5.1 Hyrje në navigim 35
   5.2 Ndërtimi i hartës me Turtlebot 2i 36
   5.3 Lokalizimi përmes odometrisë 39
   5.4 Odometria në Turtlebot 2i. 40
   5.5 Algorithmi SLAM 42
      5.5.1 Mapimi laserik/vizual 45
      5.5.2 Të dhënat odometrike 45
      5.5.3 Pikat karakteristike (ang. landmarks) 46
   5.6 Implementimi i SLAM në ROS 46
6. Turtlebot dhe Matlab 51
   6.1 Shembull lidhjeje mes Matlab dhe Turtlebot 2i 51
   6.2 Lokalizimi me algoritmit Monte Carlo 54
7. Programimi i robotëve mobil me Python 59
   7.1 Shembull fillestare me Python për Turtlebot 2i 59
   7.2 Libraria Rospy dhe Turtlebot 2i 61
   7.3 Implementimi i Dijkstra me Python në Turtlebot 2i 63
   7.4 Implementimi i kërkimit A* me Python në Turtlebot 2i 71
8. Përfundim 82
9. Referencat 92

Lista e Figurave

Figura 1: Manipuluesit robotik për kapje dhe vendosjem [Wikipedia] 6
Figura 2: Manipuluesit robotik për ngjyrosje [Wikipedia] 7
Figura 3: Robotët mobil me përdorime ne agrikulturë [wiki] 8
Figura 4: Robotët mobil për bartje në fabrika prodhimi [Wikipedia] 9
Figura 5: Përafrimi i ecjes me dy këmbë si një poligon rrotullues [1] 10
Figura 6: Roboti Turtlebot 2i: robot mobil me manipulues robotik 10
Figura 7: Baza mobile Kobuki nga Yujin Robot 13
Figura 8: Kamera 3D Intel SR300 14
Figura 9: Pozicioni i kamerës SR300 14
Figura 10: Kamera Orbbec Astra 15
Figura 11: Pozicioni i Orbbec Astra 15
Figura 12: Intel NUC BOXNUC6CAYH 16
Figura 13: Manipuluesi robotik Pincher MK3 16
Figura 14: Kontrolleri Arbotix-M 17
Figura 15:Modeli i veturës i trajtuar me modelin e biçikletës [6] 19
Figura 16: Sistemet koordinative globale dhe lokale [8] 20
Figura 17: Shembull i pozitës/orientimit të një roboti mobil [8] 21
Figura 18: Roboti mobil ne raport me sistemit koordinativ global [2] 21
Figura 19: Turtlebot 2i dhe sistemet kordinative [11] 22
Figura 20: Rrotat e Turtlebot 2i, në pamjen e poshtme të bazës mobile Kobuki 23
Figura 21: Pozita aktuale dhe e deshiruar e robotit [11] 24
Figura 22: Roboti mobil me 4 rrota të motorizuara në formë të pavarur. Te ndryshohet figura 25
Figura 23: Disa nga robotët mobil që përdorin ROS 28
Figura 24:Block diagrami i komunikimeve me ROS [7] 29
Figura 25: Skema funksionale e nivelit Filesystem [7] 30
Figura 26: Skema funksionale e nivelit të llogaritjes [7] 31
Figura 27: Përgjigja që roboti jep për komandën roscore 33
Figura 28: Përgjigja pas komandës për teleoperim 34
Figura 29: Skema e përgjithshme e problemit të lokalizlimit të robotit [4] 35
Figura 30: Leximi nga infrastruktura për navigim e turtlebot 2i 37
Figura 31:Pamja në Rviz për mapinin e mjedisit [4] 38
Figura 32: Parimi bazik i odometrisë për lokalizim [1] 40
Figura 33: Trajektorja e Turtlebot [6] 42
Figura 34: SLAM Procesi [4] 43
Figura 35: Hapi fillestarë në ciklin e SLAM procesit. Identifikmi i landmarks. [2] 44
Figura 36: Pozita e robotit pas levizjes fillestare [2] 44
Figura 37: Pozita e menduar dhe ajo aktuale e robotit 45
Figura 38: Pozita reale e robotit bazuar në estimimet mbi odometri dhe pika karakteristike [2] 45
Figura 39: Modeli i Turtlebot 2i në Gazebo 47
Figura 40: Mjedisi i krijuar ne Gazeb për testim 47
Figura 41: Harta e gjeneruar në Rviz 48
Figura 42:Loklizimi i robotit në Rviz përmes SLAM 49
Figura 43: Trajektorja e levizjes autonome te robotit pasi të jetë përfunduar lokalizimi dhe hartëzimi me SLAM 50

Figura 44:Bota e krijuat nw Gazebo pwr testim 52
Figura 45: Interface grafik pas lidhjes sw suksesshme me Turtlebot 52
Figura 46: Gazebo si mjedis pwr simulim 55
Figura 47: Harta e thirrur per lokalizim 56
Figura 48: Grimcat lokalizuese të fituara pas perditësimit te parë [3] 57
Figura 49: Grimcat lokalizuese pas 8 përditësimeve[3] 57
Figura 50: Pozicioni i sakte i robotit bazuar ne leximet e laserit [3] 58
Figura 51: Karasim mes A* dhe Dijkstra [12] 64
Figura 52: Parimi i kwrkimit sipas A* [9] 81
Figura 53: Kwrkimi sipas A* [9] 81

Abstrakti

Përgjatë këtij punimi do të trajtojmë temën e robotëve mobil, më saktësisht, do të diskutojmë për robotin mobil Turtlebot 2i. Ky është një robot mobil edukativ, por për shkak të fleksibilitetit që ofron në programimin e tij, është një robot që në masë të madhe ka arritur të depërtoj edhe si produkt final komercial.

Në pjesën e parë të punimit do të diskutojmë lidhur me robotikën në përgjithësi dhe me stadin aktual të zhvillimi të robotikës, me fokus të veçantë në robotët mobil. Më pas, trajtimi do jetë më detalisht për robotët mobil. Gjatë pjesës së dytë do të ndërtojmë një model matematik që përshkruan robotët mobil. Nga ky modul mund të shohim ngjashmërinë mes trajtimit matematik të robotëve mobil dhe manipulatorëve robotik.

Në pjesën e tretë të punimit do të flasim më specifikisht për robotin mobil Turtlebot 2i. Në këtë kapitull do të saktësojmë sensorët me të cilët i pajisur ky robot dhe po ashtu se si mund të përdoret.

Më pas, në pjesën vijuese do të flasim për sistemin operativ të robotit (ang. ROS). Ky është një sistem operativ në vete, që është i destinuar për të shërbyer si substrat mbi të cilin mund të ndërtohen platforma të ndryshme mobile. Do të diskutojmë strukturën e këtij sistemi operativ dhe po ashtu se si mund të përdoret ky sistem operativ në rastin e robotit Turtlebot.

Navigimi do jetë pjesa kryesore e këtij punimi. Navigimi si problematikë është e trajtuar mjaft detalisht përgjatë viteve në literaturë. Në rastin tonë do të shohim cilat janë qasjet bashkëkohore në zgjidhjen e problemit të lokalizimit. Në këtë pjesë do të shohim po ashtu se si është i implementuar SLAM algoritmi në rastin e robotit Turtlebot 2i, dhe pse ky algoritëm është as i njohur në pjesën e robotëve mobil.

Në fund, për të konkretizuar diskutimin, do të shohim se si mund të shkruhen programe të kostumizuara, në gjuhen Python, për detyra më specifike për Turtlebot.

Abstract

Through this paper we will be discussing the topic of mobile robots, with emphasis on Turtlebot 2i mobile robot. Turtlebot 2i is an educative platform for mobile robotics, which, due to its flexibility in programming and use, has lately found also application as a final commercial product.

In the first part of this paper, we will discuss the current state of the art field of robotics, with more focus on mobile robots. Throughout this pas we will discuss in more detail of category of mobile robots.

In the second part, we will attempt to develop a mathematical model for describing mobile robots, and due to this discussion, we will be able to notice the similarities between modeling arm type robots and mobile robots.

In the third part of the paper, the Turtlebot 2i will be the main topic of discussion. We will discuss in details its infrastructure, construction and overall functionality. Sensors, actuators and other components of the turtlebot will be treated with focus on their importance to the robot capabilities.

Next, in the next section we will talk about the robot operating system (ROS). This is an operating system by itself, which is intended to serve as a substrate on which various robotic mobile platforms can be implemented. We will discuss the structure of this operating system and also how this operating system can be used in the case of the turtlebot robot.

Navigation will be the main part of this paper. Navigation as problematic has been treated in great detail over the years in the literature. In our case, we will see what are the contemporary approaches to solving the problem of localization. In this section we will also look at how the SLAM algorithm is implemented in the case of the Turtlebot 2i robot, and why this algorithm is so well known in the mobile robotics part.

Finally, to make the discussion more practical, we will look at how costume programs can be written, in Python, for more turtlebot-specific tasks.

1. Hyrje

Robotika ka arritur suksesin e saj më të madh deri më sot në botën e prodhimit industrial. Krahët robotik, ose manipuluesit, përbëjnë një industri prej 2 miliardë dollarësh. I lidhur në shpatullën e tij në një pozicion specifik në linjën e montimit, krahu i robotit mund të lëvizë me shpejtësi dhe saktësi të madhe për të kryer detyra të përsëritura si saldimi dhe lyerja. Në industrinë e elektronikës, manipuluesit vendosin komponentë të montuar në sipërfaqe me saktësi mbinjerëzore, duke bërë të mundur telefonin portativ dhe kompjuterin laptop. Patenta e parë për atë që ne tani do ta konsideronim robot u dorëzua në vitin 1954 nga George C. Devol dhe u lëshua në vitin 1961. Pajisja përbëhej nga një krah mekanik me një kapëse që ishte montuar në shina dhe sekuenca e lëvizjeve ishte e koduar si modele magnetike të ruajtura në një daulle rrotulluese. Kompania e parë e robotikës, Unimation, u themelua nga Devol dhe Joseph Engelberger në vitin 1956. Vizioni origjinal i Devol dhe Engelberger për automatizimin robotik është bërë realitet dhe janë ndërtuar shumë miliona robotë të tipit të krahut robotik (apo manipuluesit) dhe vihet në punë në detyra të tilla si saldimi, lyerja, ngarkimi dhe shkarkimi i makinerive, montimi elektronik, paketimi dhe paletizim. Përdorimi i robotëve ka çuar në rritjen e produktivitetit dhe përmirësimin e cilësisë së produktit. Në vend që të zërë vende pune, ajo ka ndihmuar në mbajtjen e qëndrueshme të industrive prodhuese në vendet me kosto të lartë të punës. Sot shumë produkte që blejmë janë montuar ose trajtuar nga një robot. Shembuj të manipuluesve robotik në teknologjinë e prodhimit janë treguar në figurën 1 dhe 2.

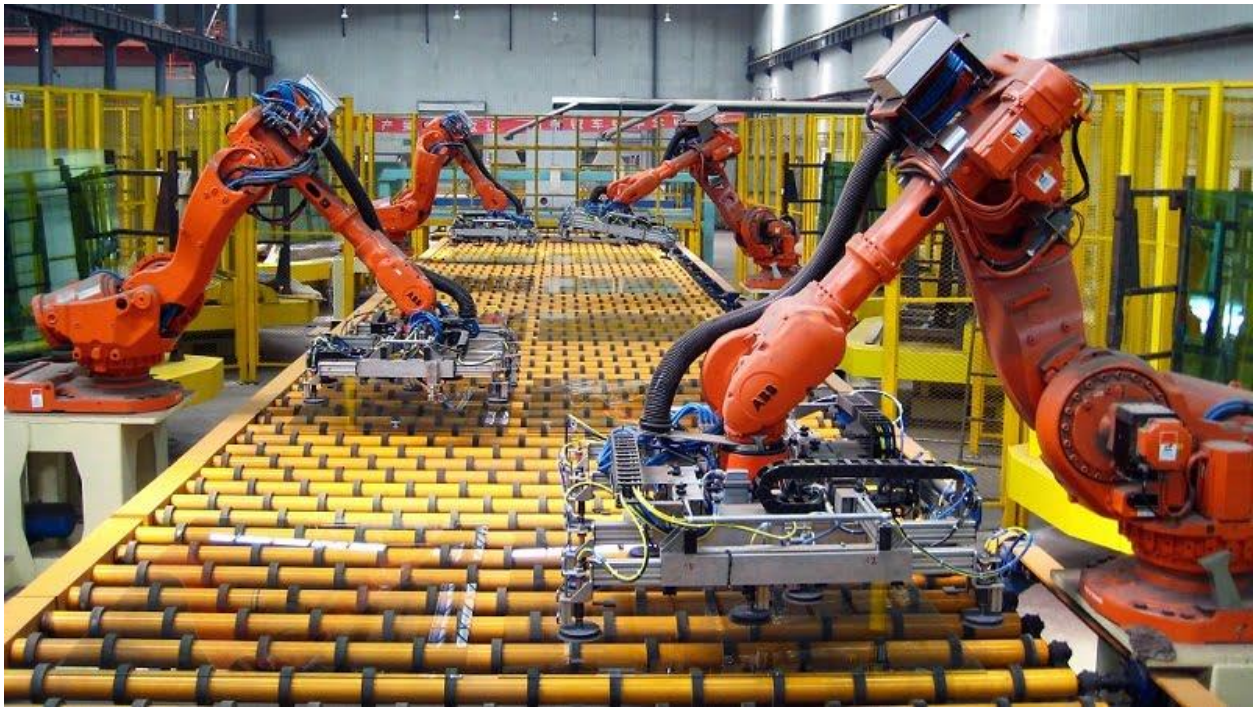

Figura 1: Manipuluesit robotik për kapje dhe vendosje [Wikipedia]

Figura 2: Manipuluesit robotik për ngjyrosje [Wikipedia]

Një robot prodhues është zakonisht një manipulues i tipit të krahut në një bazë fikse që kryen detyra të përsëritura brenda një qelize pune lokale. Pjesët i paraqiten robotit në një mënyrë të rregullt që maksimizon avantazhin e shpejtësisë dhe saktësisë së lartë të robotit. Robotët me shpejtësi të lartë janë të rrezikshëm dhe siguria arrihet duke përjashtuar njerëzit nga vendet e punës robotike. Ekzistojnë shumë lloje të ndryshme të krahëve robotik të disponueshëm në tregun e sotëm, secili i projektuar me aftësi dhe funksione thelbësore të rëndësishme që i bëjnë lloje të ndryshme specifike veçanërisht të përshtatshme për role të veçanta ose mjedise industriale.

Shumica e krahëve robotik kanë deri në gjashtë nyje që lidhin shtatë seksione, shumica ose të gjitha prej të cilave drejtohen nga forma të ndryshme të motorëve stepper dhe kontrollohen nga kompjuteri. Kjo mundëson pozicionimin tepër të saktë të pjesës 'dorë' ose efektore fundore të krahut, e cila në shumicën e përdorimeve industriale në përgjithësi do të jetë një lloj mjeti ose bashkëngjitjeje e specializuar, e krijuar për të kryer një veprim shumë specifik ose seri të përsëritshme artikulacionesh.

Në pjesën më të madhe, dallimi kryesor midis llojeve të ndryshme të krahëve robotik qëndron në mënyrën se si nyjat e tyre janë të dizajnuara për të artikuluar - dhe më pas gamën e lëvizjes dhe funksionet që ata janë në gjendje të kryejnë - si dhe llojin e kornizës që ata janë. mbështetur nga dhe gjurmën që ata kërkojnë për instalim dhe funksionim.

Krahët robotik mund të përdoren për të gjitha llojet e roleve të prodhimit industrial, përpunimit dhe prodhimit - çdo detyrë në të cilën kërkohen lëvizje jashtëzakonisht të sakta, të shpejta dhe të përsëritshme, në fakt. Krahët robotik të të gjitha llojeve përdoren sot në çdo shkallë prodhimi, nga montimi i hollësishëm i pllakave të qarkut deri te industritë e rënda me vëllim të madh si linjat e prodhimit të automobilave, si dhe në një gamë të madhe aplikacionesh 'zgjidh dhe vendos' (rripin transportues).

Megjithatë, me gjithë sukseset e tyre, këta robotë komercialë vuajnë nga një disavantazh themelor: mungesa e lëvizshmërisë. Një manipulues fiks ka një gamë të kufizuar lëvizjeje që varet nga

vendi ku është mbyllur. Në të kundërt, një robot i lëvizshëm do të jetë në gjendje të udhëtojë në të gjithë fabrikën e prodhimit, duke aplikuar në mënyrë fleksibël talentet e tij kudo që është më efektiv. Ky punim fokusohet në teknologjinë e lëvizshmërisë: si mundet një robot i lëvizshëm të lëvizë i pambikëqyrur nëpër mjediset e botës reale për të përmbushur detyrat e tij? Sfida e parë është vetë lëvizja. Si duhet të lëvizë një robot i lëvizshëm dhe çfarë ka të bëjë me një mekanizëm të veçantë lëvizjeje që e bën atë superior ndaj mekanizmave alternative të lëvizjes? Për shembull, robotët AGV (automjeti autonom i drejtuar) shpërndajnë në mënyrë autonome pjesë ndërmjet stacioneve të ndryshme të montimit duke ndjekur telat elektrikë të posaçëm udhëzues të instaluar në dysheme ose, ndryshe, duke përdorur laser në bord për t'u lokalizuar brenda një përdoruesi të specifikuar harta. Roboti i shërbimit Helpmate transporton ushqim dhe ilaçe nëpër spitale duke gjurmuar pozicionin e dritave të tavanit, të cilat i specifikohen manualisht robotit më parë. Disa kompani kanë zhvilluar robotë pastrimi autonome, kryesisht për ndërtesa të mëdha. Një robot i tillë pastrimi është në përdorim në Metronë e Parisit. Robotët e tjerë të specializuar të pastrimit përfitojnë nga modeli i rregullt gjeometrik i korridoreve në supermarkete për të lehtësuar detyrat e lokalizimit dhe navigimit. Shembuj të robotëve mobil janë treguar në figurat më poshtë.

Figura 3: Robotët mobil me përdorime ne agrikulturë [wiki]

Figura 4: Robotët mobil për bartje në fabrika prodhimi [Wikipedia]

Edhe pse robotët mobil kanë një grup të gjerë aplikacionesh dhe tregjesh siç u përmbledh më lart, ekziston një fakt që është i vërtetë për pothuajse çdo robot të suksesshëm celular: dizajni i tij përfshin integrimin e shumë trupave të ndryshëm të njohurive. Kjo e bën robotikën e lëvizshme një fushë aq ndërdisiplinore sa mund të jetë. Për të zgjidhur problemet e lëvizjes, roboti i lëvizshëm (roboti mobil) duhet të kuptojë mekanizmin dhe kinematikën, dinamikën dhe teorinë e kontrollit. Për të krijuar sisteme të forta perceptuese, roboti i lëvizshëm duhet të shfrytëzojë fushat e analizës së sinjalit dhe trupat e specializuar të njohurive si vizioni kompjuterik për të përdorur siç duhet një mori teknologjish sensorësh. Lokalizimi dhe navigimi kërkojnë njohuri të algoritmeve kompjuterike, teorisë së informacionit, inteligjencës artificiale dhe teorisë së probabilitetit.

Një robot i lëvizshëm ka nevojë për mekanizma lëvizjeje që i mundësojnë atij të lëvizë pa kufi në të gjithë mjedisin e tij. Por ka një larmi të madhe mënyrash të mundshme për të lëvizur, dhe kështu zgjedhja e qasjes së një roboti ndaj lëvizjes është një aspekt i rëndësishëm i dizajnit të robotit mobil. Në laborator, ka robotë hulumtues që mund të ecin, kërcejnë, vrapojnë, rrëshqasin, bëjnë patina, notojnë, fluturojnë dhe, natyrisht, rrokulliset. Shumica e këtyre mekanizmave të lëvizjes janë frymëzuar nga homologët e tyre biologjikë. Megjithatë, ekziston një përjashtim: rrota është një shpikje njerëzore që arrin efikasitet jashtëzakonisht të lartë në teren të rrafshet. Ky mekanizëm nuk është plotësisht i huaj për sistemet biologjike. Sistemi ynë i ecjes dykëmbëshe mund të përafrohet me një poligon të rrotulluar, me anë të barabarta në gjatësi me hapësirën e hapit, ashtu siç tregohet në figurën 5. Ndërsa madhësia e hapit zvogëlohet, shumëkëndëshi i afrohet një rrethi ose rrote. Por natyra nuk zhvilloi një nyje plotësisht rrotulluese, me fuqi aktive, e cila është teknologjia e nevojshme për lëvizjen me rrota.

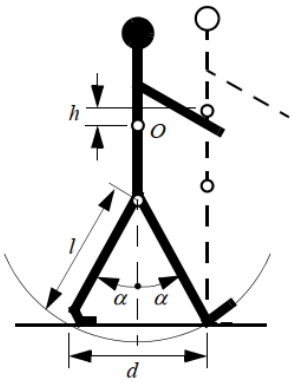

Figura 5: Përafrimi i ecjes me dy këmbë si një poligon rrotullues [1]

Por ndërsa sipërfaqja bëhet e butë, lëvizja me rrota grumbullon jo efikasitet për shkak të fërkimit të rrotullimit, ndërsa lëvizja me këmbë vuan shumë më pak sepse përbëhet vetëm nga kontaktet pika me tokën. Është e kuptueshme, pra, që natyra favorizon lëvizjen me këmbë, pasi sistemet e lëvizjes në natyrë duhet të funksionojnë në terren të ashpër dhe të pa strukturuar. Për shembull, në rastin e insekteve në një pyll, ndryshimi vertikal në lartësinë e tokës është shpesh një rend i madhësisë më i madh se lartësia totale e insektit. Në të njëjtën mënyrë, mjedisi njerëzor shpesh përbëhet nga sipërfaqe të projektuara, të lëmuara, si brenda ashtu edhe jashtë. Prandaj, është gjithashtu e kuptueshme që pothuajse të gjitha aplikacionet industriale të robotikës së lëvizshme përdorin një formë të lëvizjes me rrota.

Në rastin tonë, do të shqyrtojmë robotin mobil Turtlebot 2i, qëështë një robotik mobil, lëvizja e të cilit bëhet me rrota, por i cili është i pajisur me një manipulues robotik në strukturën e tij. Ky robot është treguar në figurën 6.

Figura 6: Roboti Turtlebot 2i: robot mobil me manipulues robotik

Turtlebot 2i, një robot modular i bazuar në ROS i mundësuar nga Intel Joule, mundëson zhvillimin e robotëve të avancuar të gjeneratës së ardhshme. Roboti Kërkimor Mobile TurtleBot 2i ofrohet nga Interbotix Labs dhe Fondacioni i Robotikës me Burim të Hapur (OSRF). TurtleBot 2i bazohet në versionet e mëparshme të platformës së njohur robotike të bazuar në Sistemin Operativ Robot (ROS) për të ofruar

aftësi të avancuara jashtë kutisë për prototipin e shpejtë të robotëve të gjeneratës së ardhshme. Qëllimi i platformës Turtlebot ka qenë gjithmonë të sjellë teknologji komplekse, të tilla si navigimi autonom dhe manipulimi robotik, në duart e novatorëve dhe zhvilluesve në një format më të arritshëm. Turtlebot 2i ofron krahun robotik Pincher MK3 5 DOF si një opsion standard. Kjo është hera e parë që një robot Turtlebot ka ofruar mbështetje të një krahu robotik. Ashtu si Turtlebot 3, Turtlebot 2i përdor modulin Intel Joule 570x. Së bashku me fuqinë e fuqishme kompjuterike, Joule 570x lejon TurtleBot 2i të integrohet lehtësisht me kamerat e dyfishta Intel RealSense 3D. TurtleBot 2i përdor një kamerë Intel ZR300 RealSense për navigim dhe hartografi, ndërsa kamera me rreze të shkurtër SR300 RealSense i dedikohet krahut të robotit MK3. Një përshkrim më të detajuar të këtij roboti mobil do ta bëjmë një kapitullin vijues.

2. Roboti mobil Turtlebot 2i

Turtlebot 2i, një robot modular i bazuar në ROS i mundësuar nga Intel Joule. Platforma mbi në cilën është ndërtuar ky robot është një platformë e hapur (ang. open source). Për të mundësuar kompatibilitet mes strukturave të ndryshme robotike, ky robot mobil operon në sistemin operativ ROS. Për shkak të strukturës që ka, ky robot është i dedikuar për përdorim në ambiente të brendshme. Turtlebot 2i mund të zhvillojë një shpejtësi maksimale prej 2.5km/h dhe mund të operojë me një ngarkesë maksimalisht deri në 5kg.

Turtlebot2i ofron demo të shumta dhe paketa softuerësh që mundësojnë një zbatim të lehtë të aplikacioneve të krijuara vetë që varen nga veçoritë që janë renditur më poshtë:

- Navigim autonom
- Identifikim i hartave zonale
- Manipulim me objekte
- Planifikim i trajektores/rrugës
- Evitim i pengesave

Sensorët me të cilët ky robot është i pajisur janë:

- Kamera 3D RealSense SR300
- Kamera 3D Orbbec Astra
- Akselerometri/Xhiroskopi/Kompasi
- Sensorët për detektimin e teheve

Disa nga karakteristikat hardware-ike të këtij roboti janë:

- Baza mobile Kobuki
- Inter Joule 570X
- 4GB RAM
- Ubuntu 16.04 / ROS Kinetic
- 802.11AC WiFi / Bluetooth 4.0
- Pllakat e ndërrueshme të drurit
- Kapësja (efektori i fundëm) Pincher MK3
- LiPo bateria 4500mAh
- Robokontrolleri Arbotix-M

Pra ky robot, është i pajisur me një infrastrukturë mjaft të kompletuar për t'i përmbushur qëllimet që i parashtrohen një roboti mobil. Kjo përfshinë lëvizshmërinë, aftësinë për të krijuar harta tereni, aftësinë për të detektuar pengesat, aftësinë për lidhjen në rrjetë lokale (me kabllo apo pa tela) etj. Pikërisht për shkak të kësaj infrastukture, ky robot paraqet një platformë mjaft të përshtatshme për të implemtuar SLAM algoritmet për lokalizim dhe navigim autonom.

2.1 Baza mobile Kobuki

Kobuki është një bazë e lëvizshme e ofruar nga Yujin Robot. Ka sensorë, motorë dhe burime energjie, por në vetvete nuk mund të bëjë asgjë. Për të qenë funksional, ju kërkon të ndërtoni platformën tuaj në

majë të guaskës së saj. Nga ana e harduerit, kjo zakonisht përfshin shtimin e një netbook ose një bordi të integruar për të qenë bërthama llogaritëse për sistemin tuaj dhe zakonisht disa sensorë shtesë për ta bërë atë vërtet funksional. Nga ana e softuerit, kjo përfshin ose ndërtimin e të gjithë softuerit tuaj ose integrimin e softuerit nga grupet e tjera me burimet tuaja të zhvillimit. Kjo mund të marrë kohë, përpjekje dhe para. Nëse dëshironi të filloni shpejt, ose një platformë të standardizuar për grupin tuaj, mund të përdorni një platformë ekzistuese referimi ose t'i kërkoni shpërndarësit tuaj më të afërt të personalizojë një platformë të paketuar për grupin tuaj. Kjo bazë mobile është treguar në figurën 7.

Figura 7: Baza mobile Kobuki nga Yujin Robot

Kobuki është një bazë mobile për qëllime kërkimi dhe zhvillimi me burim të hapur. Mund të përdoret për të ndërtuar shpejt dhe me çmim të ulët platforma kërkimore dhe gjithashtu ofron akses në softuer të tjerë me burim të hapur, duke përfshirë ROS. Rezultati është një produkt që mund të përdorë shumë lehtë teknologjitë ekzistuese të sensorit 3D dhe algoritmet e navigimit për të zhvilluar aplikacione për kërkime. Në fakt, i tërë roboti TurtleBot 2i është i ndërtuar mbi bazën mobile Kobuki. Për konfigurimin, instalimin dhe punën me këtë bazë robotike do të flasim më në detaje në kapitullin dedikuar sistemit operativ ROS.

Kobuki është baza mobile e Turtlebot 2. Është një robot i lëvizshëm me rrota me dy rrota diferenciale dhe dy rrota kastor, gjithashtu përmban sensorë afërsisë, enkoder në rrota dhe një xhirometër për çdo aks. Baza e Kobuki gjithashtu mund të furnizojë me energji sensorë të jashtëm (Kinect, sensorë ultrasonikë, sensorë infra të kuqe, skanerë laser, kamera të tjera ...) ose aktuatorë (motorë, servomotorë).

2.2 Kamera 3D SR300

SR300 është një produkt nën-montues që zbaton një sistem imazhi 3D të lehtë të koduar me rreze të shkurtër. Madhësia e vogël e nën-montimit SR300 u siguron integruesve të sistemit fleksibilitet për t'u dizajnuar në një gamë të gjerë produktesh. Gama e gjerë e konfigurimeve të modalitetit 3D dhe aftësive të sinkronizimit të SR300 mundësojnë që produkti të jetë një zgjidhje optimale për aplikacionet e imazhit 3D. Kjo kamerëështë treguar në figurën 8.

Figura 8: Kamera 3D Intel SR300

Pikërisht për shkak se është kamerë me rreze veprim të shkurtër, kjo kamerë është e dedikuar për hapësirën e punës së manipulatorit robotik, me të cilin është i pajisur Turtlebot 2i. Pra duke përdorur këtë kamerë, mund të bëhet vizualizim i hapësirën së punës së krahut robotik, por edhe të mund të gjenerohet një stereo imazh qëështë i nevojshëm për gjenerim të trajektoreve të lëvizjes së krahut robotik. Pozicioni i kësaj kamere në robot është treguar në figurën 10. Nga figura shihet se mënyra e vendosjes së kësaj kamere është e tillë që të ketë një pamje të plotë të krahut robotik, por edhe të objekteve me të cilat manipulon krahu robotik, ashtu që algoritmet për lëvizjes e krahut të jetë të azhuruara me hapësirën e punës së krahut dhe të objekteve që ndodhen në këtë hapësirë pune.

Figura 9: Pozicioni i kamerës SR300

2.3 Kamera 3D Orbbec Astra

Kamera 3D OrbbecAstra implementon një stereo shikimin të thellësisë, një njësi matëse inercioni me 6 shkallë lirie, një sensor optik të tipit fisheye të gjitha këto në një interface të vetëm. Pra është një sistem i tërë i dedikuar për përpunim të imazheve dhe për ndijim të orientimit dhe pozicionit. Kjo bën që disa nga aplikimet e kësaj kamere të jenë njohja e objekteve, lokalizimi, përcjellja e objekteve, njohja e gjetësve etj. Kjo kamerë është treguar në figurën 10.

Figura 10: Kamera Orbbec Astra

Turtlebot përdor këtë kamerë për të bërë skanim të terenit në të cilin ndodhet. Kjo ngase kjo kamerë është me rreze-veprim të gjatë, andaj si e tillë është komponentë kyçe në implementimin e SLAM algoritmit. Pozicioni i kësaj kamere në Turtlebot është treguar në figurën 11.

Figura 11: Pozicioni i Orbbec Astra

Pra edhe këtu mund të shohim se pozita e kësaj kamere është e tillë që të ketë një pamje të pastër të terrenit në të cilin do të lëvizë roboti. Kjo i mundëson sistemit robotik që të jetë i përditësuar më pengesat që ndodhet gjatë rrugës, në lokalizim dhe navigim të robotit, por edhe në implementimin të SLAM algoritmit.

2.4 Intel NUCBOXNUC6CAYH

Turtlebot 2i është po ashtu i pajisur me një kompjuter kompakt të llojit NUC6CAYH. Në fakt, përkundër asaj që roboti është i ndërtuar mbi bazën Kobuki, i tërë sistemi operues ROS i Turtlebot 2i operon mbi kompjuterin Intel NUC BOXNUC6CAYH. Termi NUC i referohet “Next Unit of Computing”, që janë një seri shumë kompakte e një sistemi të tërë kompjuterik, dhe për shkak që janë kompakt janë shumë të përshtatshëm për qëllime të tilla sikurse në rastin e Turtlebot 2i. Ky model specifik në rastin e Turtlebot 2i është treguar në figurën 12.

Figura 12: Intel NUC BOXNUC6CAYH

Të gjitha programet, algoritmet dhe qëllime të tjera të punës për Turtlebot 2i shkruhen pikërisht në këtë kompjuter kompakt. Është këtu ku vendoset sistemi operativ ROS, me të cilin dirigjohet puna e robotit mobil.

2.5 Krahu Robotik Pincher MK3

Krahu robotik i vendosur në Turtlebot 2i është i llojit Pincher MK3, që ofrohen nga kompania Phantom X. Kjo është një shtesë në krahasim me modelet e mëhershme të Turtlebot, dhe si i tillë ofron mundësin që përveç mobilitetit të robotit, të kemi edhe aftësinë për manipulim me objekte. Në natyrën e saj më bazike, efektori i fundëm i këtij manipuluesi robotik është një kapëse. Andaj mundëson kapjen dhe vendosjen e objekteve. Turtlebot 2i ofron krahun robotik Pincher MK3 me 4 shkallë lirike si një opsion standard i mbështetur plotësisht që lejon robotin të ndërveprojë me objekte të vogla në botën reale, duke e transformuar në mënyrë efektive Turtlebot 2i në një manipulues celular jashtëzakonisht të aftë. Ky manipulues robotik dhe pozicioni i tij në Turtlebot është treguar në figurën 13.

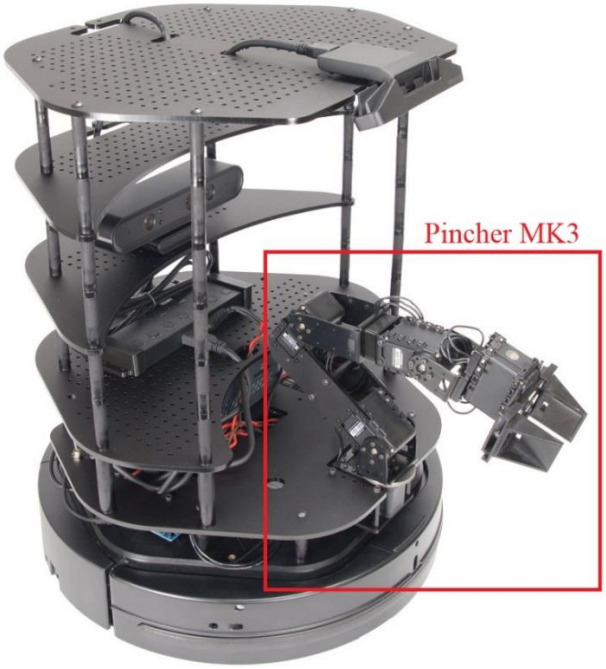

Figura 13: Manipuluesi robotik Pincher MK3

Secila nga shkallët e lirisë së këtij manipuluesi mundësohet një një servomotor. Lëvizja precize e manipuluesit mundëson nga lëvizjet e koordinuara të secilës nyje. Andaj, është e nevojshme edhe prania e një sistemi që mundëson ngajës se këtyre servomotorëve që janë pjesë e manipuluesit robotik. Kjo mundësohet nga robokontrolleri (kontroller i dedikuar për robotikë) Arbotix-M. Ky kontroller është treguar në figurën më poshtë.

Figura 14: Kontrolleri Arbotix-M

ArbotiX-M Robocontroller është një sistem i avancuar i telekomandimit për robotë të përmasave të vogla dhe të mesme si p.sh. kompleti robotik Hexapod PhantomX AX – dhe është kompatibil me bordet Arduino! Ai përfshin një mikrokontrollues të fuqishëm AVR, një modul radio wireless Xbee, një drejtues motori të dyfishtë dhe një numër titujsh servo me 3 pin për I/O. Robokontrolluesi ArbotiX-M është projektuar posaçërisht për kontroll duke përdorur servomotorët Dynamixel. Robokontrolluesi ArbotiX-M nuk është thjesht një zëvendësim për kontrolluesin CM-5, ai është më shumë një mjet i nivelit të lartë i dedikuar për ndërtimin e robotëve më të sofistikuar.

3. Modelimi i robotëve mobil – Shembull i Turtlebot 2i

Kinematika është studimi më themelor se si sillen sistemet mekanike. Në robotët mobil, ne duhet të kuptojmë sjelljen mekanike të robotit si për të dizajnuar robotë të përshtatshëm mobil për detyrat dhe për të kuptuar se si të krijojmë softuer kontrolli për një shembull të harduerit robotik të lëvizshëm. Komuniteti i robotikës mobile shtron shumë nga të njëjtat pyetje kinematike si komuniteti i manipulatorëve robotik. Hapësira e punës e një roboti manipulues është thelbësore sepse përcakton brezin e pozicioneve të mundshme që mund të arrihen nga efektori i tij fundor në lidhje me fiksimin e tij me mjedisin. Hapësira e punës e një roboti mobil është po aq e rëndësishme sepse përcakton gamën e pozicioneve të mundshme që roboti celular mund të arrijë në mjedisin e tij. Kontrollueshmëria e krahut të robotit përcakton mënyrën në të cilën përfshirja aktive e motorëve mund të përdoret për të lëvizur nga një pozicion në tjetrin në hapësirën e punës. Në mënyrë të ngjashme, kontrollueshmëria e një roboti mobil përcakton shtigjet dhe trajektoret e mundshme në hapësirën e tij të punës. Dinamika e robotit vendos kufizime shtesë në hapësirën e punës dhe trajektoren për shkak të konsideratave të masës dhe forcës. Roboti mobil është gjithashtu i kufizuar nga dinamika; për shembull, një qendër e lartë graviteti kufizon rrezen praktike të kthesës së një roboti të shpejtë, si makinë për shkak të rrezikut të rrotullimit.

Por ndryshimi kryesor midis një roboti të lëvizshëm dhe një krahu manipulues paraqet gjithashtu një sfidë të rëndësishme për vlerësimin e pozicionit. Një manipulues ka një skaj të fiksuar në mjedis. Matja e pozicionit të një efekti fundor të krahut është thjesht një çështje e të kuptuarit të kinematikës së robotit dhe matjes së pozicionit të të gjitha nyjave të ndërmjetme. Kështu, pozicioni i manipuluesit është gjithmonë i llogaritshëm duke parë të dhënat aktuale të sensorit. Por një robot i lëvizshëm është një automat i pavarur që mund të lëvizë plotësisht në lidhje me mjedisin e tij. Nuk ka asnjë mënyrë të drejtpërdrejtë për të matur pozicionin e një roboti mobil në çast. Në vend të kësaj, njeriu duhet të integrojë lëvizjen e robotit me kalimin e kohës. Shtojini kësaj pasaktësitë e vlerësimit të lëvizjes për shkak të rrëshqitjes dhe është e qartë se matja me saktësi e pozicionit të një roboti mobil është një detyrë jashtëzakonisht sfiduese.

Procesi i të kuptuarit të lëvizjeve të një roboti fillon me procesin e përshkrimit të kontributit që çdo rrotë ofron për lëvizjen. Çdo rrotë ka një rol për të mundësuar lëvizjen e të gjithë robotit. Në të njëjtën mënyrë, çdo rrotë imponon gjithashtu kufizime në lëvizjen e robotit; për shembull, refuzimi i rrëshqitjes anash. Në seksionin vijues, ne prezantojmë shënimin që lejon shprehjen e lëvizjes së robotit në një kornizë referimi globale, si dhe kornizën referuese lokale të robotit. Më pas, duke përdorur këtë shënim, ne demonstrojmë ndërtimin e modeleve të thjeshta kinematike të lëvizjes përpara, duke përshkruar se si roboti në tërësi lëviz si funksion i gjeometrisë së tij dhe sjelljes individuale të rrotave. Më pas, ne përshkruajmë zyrtarisht kufizimet kinematike të rrotave individuale dhe më pas kombinojmë këto kufizime kinematike për të shprehur të gjithë kufizimet kinematike të robotit. Me këto mjete, mund të vlerësohen shtigjet dhe trajektoret që përcaktojnë manovrimin e robotit.

3.1 Robotët mobil me rrota

Makinat me rrota janë një klasë shumë efektive e automjeteve. Në këtë seksion ne do të krijojmë një model për një automjet të ngjashëm me veturën dhe do të zhvillojmë kontrollues që mund ta drejtojnë makinën në një pikë, përgjatë një vijë, ndiqni një rrugë arbitrare dhe së fundi, drejtoni një pozë specifike. Një model i përdorur zakonisht për një automjet të ngjashëm me makinën me katër rrota është modeli i

biçikletës i paraqitur në figurën 15. Biçikleta ka një rrotë të pasme të fiksuar në trup dhe rrafshi i rrotës së përparme rrotullohet rreth boshtit vertikal për të drejtuar automjetin.

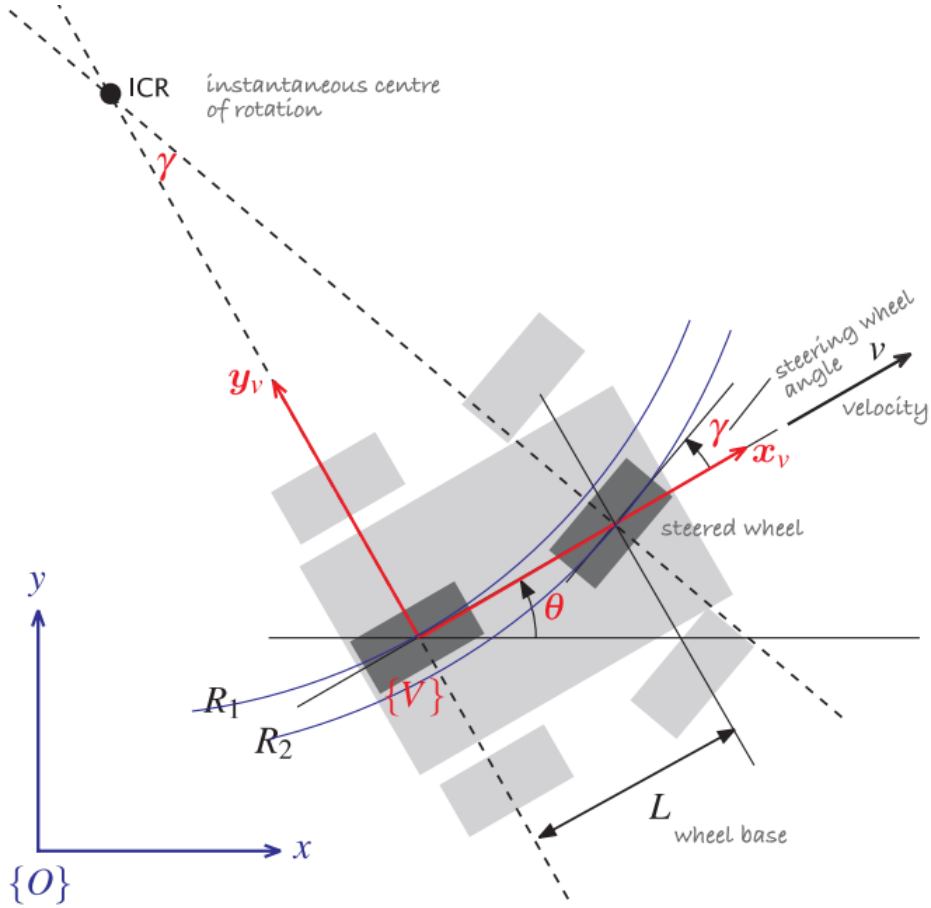

Figura 15: Modeli i veturës i trajtuar me modelin e biçikletës [6]

Pozita e veturës, nga figura më lartë, është përshkruar përmes sistemit koordinativ $\{V\}$, ku qendra e këtij sistemit është në boshtin e pasmë të veturës kurse vlerat pozitive të boshtit x janë në lëvizjen para të veturës. Konfiguracioni i veturës përshkruhet përmes koordinatave të përgjithësuara $q = (x, t, \theta)$. Shpejtësia e veturës përshkruhet me v dhe kjo bëhet vetëm në drejti të boshtit x, kjo ngase shpejtësia përgjatë boshtit y është zero sepse roboti nuk mund të lëvize lirshëm përgjatë këtij boshti. Nëse pozicioni i timonit është i fiksuar, atëherë trajektorja e lëvizjes së veturës do jetë një trajektore harkore dhe shpejtësia, nga figura më lartë, do jetë:

$$\dot{x} = v \cos \theta$$

$$\dot{y} = v \sin \theta$$

$$\dot{\theta} = \frac{v}{L} \tan \Upsilon$$

Ky model referohet si një model kinematikë pasi përshkruan shpejtësitë e automjetit, por jo forcat ose çift rrotullues që shkaktojnë shpejtësinë. Shpejtësia e ndryshimit të drejtimit Ë referohet si shpejtësia e kthesës, shpejtësia e drejtimit ose shpejtësia e devijimit dhe mund të matet me një xhiroskop. Mund të konkludohet gjithashtu nga shpejtësia këndore e rrotave në anën e majtë dhe të djathtë të automjetit, të cilat ndjekin harqe me rreze të ndryshme dhe për këtë arsye rrotullohen me shpejtësi të ndryshme.

3.2 Përshkrimi i pozitën së robotit mobil

Gjatë gjithë kësaj analize ne modelojmë robotin si një trup i ngurtë mbi rrota, që vepron në një plan horizontal. Dimensionet e përgjithshme të kësaj shasie roboti në rrafsh është tre, dy për pozicionin në rrafsh dhe një për orientimin përgjatë boshtit vertikal, i cili është ortogonal me rrafshin. Sigurisht, ka shkallë shtesë lirie dhe fleksibiliteti për shkak të boshteve të rrotave, nyjave të drejtimit të rrotave dhe nyjave rrotave kastore. Megjithatë, me shasinë e robotit ne i referohemi vetëm trupit të ngurtë të robotit, duke injoruar nyjat dhe shkallët e lirisë së brendshme të robotit dhe rrotave të tij. Për të specifikuar pozicionin e robotit në rrafsh, ne vendosim një marrëdhënie midis sistemit koordinativ referent global të rrafshit dhe sistemit koordinativ lokal të robotit, si në figurën 16.

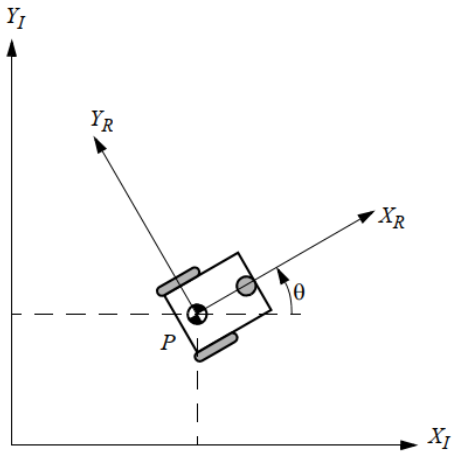

Figura 16: Sistemet koordinative globale dhe lokale [8]

Boshtet X_I dhe Y_I janë boshtet e sistemit kordinativ global. Për të përshkruar pozitën e robotit në rast me këtë sistem kordinativ, po marrim një pikë P në trup të robotit, e cila do jetë pika materiale që do reprezenton robotin. Boshtet X_R dhe Y_R në këtë rast janë boshtet e sistemit kordinativ lokal të robotit, dhe për këtë sistem pika P do jetë qendra e tij. Pozita e pikës P, në raport me sistemin koordinativ global do të përshkruhet përmes koordinatave x, y dhe këndit θ. Ky kënd paraqet shfazimin mes të dy sistemeve koordinatave. Andaj, tani pozita e robotit në raport me sistemin koordinativ global mund të përshkruhet përmes vektorit:

$$\xi = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

Për të pasur përshkrim të plotë të lëvizjes së robotit, tani është e nevojshme që të krijohet një ndërlidhje mes përshkrimit që roboti ka ndaj sistemit kordit lokal të tij, në raport më sistemin koordinativ global. Vendosja e kësaj ndërlidhje arrihet përmes matricave rrotulluese. Meqë lëvizja e robotit do jetë e tillë që të gjitha lëvizjes harkore kryhen ndaj boshtit z, atëherë forma e matricës rrotulluese do jetë:

$$R = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Po marrim si shembull rastin kur pozita e robotit mobil, ndaj sistemit referent global është si ajo e treguar në figurën 17.

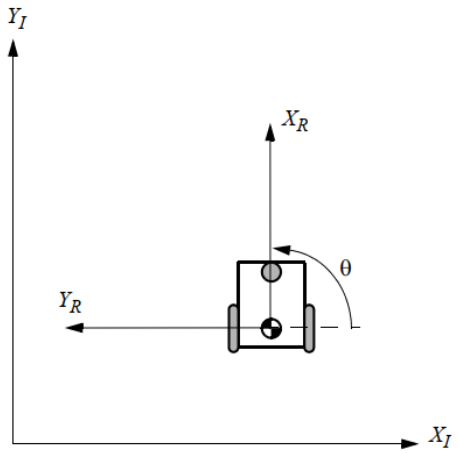

Figura 17: Shembull i pozitës/orientimit të një roboti mobil [8]

Për këtë rast matrica rrotulluese do jetë:

$$R = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Kjo vërehet qartë ngase këndi mes sistemit koordinativ lokal dhe atij global është$90°$. Poashtu, nëse e kemi të njohur vektorin e shpejtësisë$(\dot{x}, \dot{y}, \dot{\theta})$ në raport me sistemit global, atëherë duke shfrytëzuar këto relacione mund të gjejmë edhe shpejtësinë në raport me sistemin koordinativ lokal. Kjo bëhet përmes shprehjes:

$$\dot{\xi} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{y} \\ -\dot{x} \\ \dot{\theta} \end{bmatrix}$$

Në rastet më të thjeshta, transformimi i përshkruar nga ekuacionet paraprake është i mjaftueshëm për të gjeneruar një formulë që kap kinematikën direkte të robotit të lëvizshëm: si lëviz roboti, duke pasur parasysh gjeometrinë e tij dhe shpejtësinë e rrotave të tij? Më formalisht, për të përgjithësuar problem, po marrim në trajtim shembullin nga figura 18.

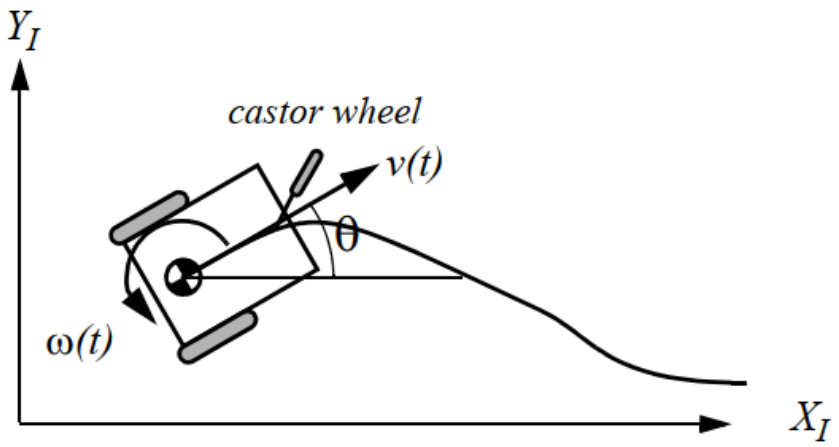

Figura 18: Roboti mobil nëraport me sistemit koordinativ global [2]

Po trajtojmë situatën kur pozita e robotit është e tillë që këndi θ është zero dhe lëvizja e robotit para në esencë do jetë kah vlerat pozitive të boshtit global X_I. Nëse njëra rrotë lëvizë përderisa tjera është

joaktive, atëherë pika materiale që përshkruan robotin do lëvizë me gjysmën e shpejtësisë totale. Shpejtësia do jetë:

$$\dot{x}_{r1} = \frac{1}{2} r\dot{\varphi}_1$$

Nëse të dy rrotat rrotullohet, por në drejtime të kundërta, atëherë shpejtësia efektive e robotit do jetë zero, që është e pritshme. Natyrisht fjalën e kemi për kontributin që lëvizja ka përgjatë boshtit X_I. Kjo ngase në rastin kur njëra rrotë lëvizë dhe tjetra është joaktive, apo ku rrotat rrotullohen në drejtime të kundër, do të kemi situatën që roboti do të rrotullohet. Në rastin e përgjithshëm shpejtësitë rrotulluese ω të secilës nga rrotat do jenë:

$$\omega_1 = \frac{r\dot{\varphi}_1}{2l}$$

$$\omega_2 = -\frac{r\dot{\varphi}_2}{2l}$$

Duke kombinuar këto ekuacione mund ta fitojmë vektorin e shpejtësisë së robotit, i cili është:

$$\dot{\xi} = \begin{bmatrix} \frac{r\dot{\varphi}_1}{2} + \frac{r\dot{\varphi}_2}{2} \\ 0 \\ \frac{r\dot{\varphi}_1}{2l} - \frac{r\dot{\varphi}_2}{2l} \end{bmatrix}$$

Në ekuacionet e mësipërme l është distanca e rrotave nga origjina e sistemit koordinativ global kurse r është diametri i rrotave.

3.3 Modeli dhe kinematika e Turtlebot 2i.

Duke mbi ndërtuar mbi diskutimin paraprak, dhe natyrisht duke pasur parasysh rregullat e përgjithshme që i parashtrohen një modeli matematik, tani do të shohim se si mund të ndërtohet një model i thjeshtë kinematikë për rastin e robotit mobil Turtlebot 2i. Ky robot, në model të thjeshtuar është treguar në figurën 19.

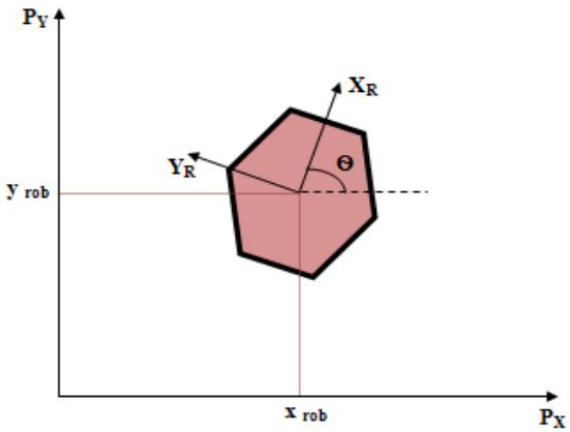

Figura 19: Turtlebot 2i dhe sistemet kordinative [11]

Ashtu si më parë, edhe nga figura më lartë shohim se kemi një sistem koordinatave global(X, Y) të fiksuar, dhe një sistem koordinativ lokal(X_R, Y_R) të robotit. Pozita e robotit në raport me sistemit koordinativ global përshkruhet me vektorin (x, y, θ), kurse ndërlidhja mes sistemeve koordinatave, ashtu si diskutuar më herët, do jetë përmes matricave rrotulluese:

$$R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rrotat motorizohen në mënyrë të pavarur. Kur të dy rrotat rrotullohen me të njëjtën shpejtësi në veprimin e drejtpërdrejtë, roboti lëviz përpara, përndryshe ai lëviz prapa. Kthimi djathtas bëhet duke aktivizuar rrotën e majtë me një shpejtësi më të madhe se ajo e rrotës së djathtë dhe anasjelltas për t'u kthyer majtas. Ai gjithashtu mund të rrotullohet në vend duke lëvizur një rrotë përpara dhe rrotën e dytë në drejtim të kundërt me të njëjtën shpejtësi. Rrota e tretë është një rrotë e lirë që ruan stabilitetin e robotit. Në rastin e Turtlebot 2i, fjala është për rrotat që janë pjesë e bazës mobile Kobuki. Këto janë treguar në figurën 20. Pra kjo bazë ka dy rrota të pavarura të motorizuara dhe një rrotë të lirë.

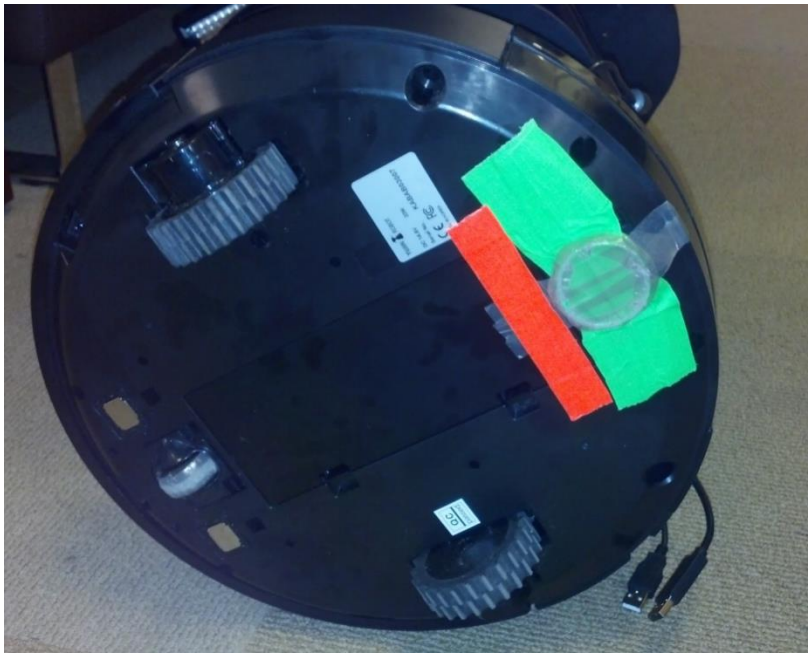

Figura 20: Rrotat e Turtlebot 2i, në pamjen e poshtme të bazës mobile Kobuki

Rrotat Turtlebot 2 janë jo-holonomike, andaj do përshkruhen me kushtëzimet në bazë të ekuacionit:

$$\dot{x}\sin\theta - \dot{y}\cos\theta = 0$$

Kurse relacioni mesh shpejtësisë së robotit (v, w) dhe shpejtësisë së rrotave (w_1, w_2) do jetë ashtu si e trajtuar në pjesën e mëparshme. Pra ky relacionet përshkruhet me:

$$v = v_x = v_1 + v_2 = r\frac{w_1 + w_2}{2}$$

$$w = w_1 + w_2 = r\frac{w_1 - w_2}{d}$$

ku me r është shënuar rrezja e rrotave kurse me d është shënuar distanca mes rrotave.

Po supozojmë që roboti ndodhet në një pozicion arbitrar P që përshkruhet me koordinatat (x_c, y_c, θ). Dëshirojmë ta lëvizim nga ky pozicion në një pozicion tjetër R, me koordinatat(x_d, y_d, β). Kjo situatë është treguar në figurën 21.

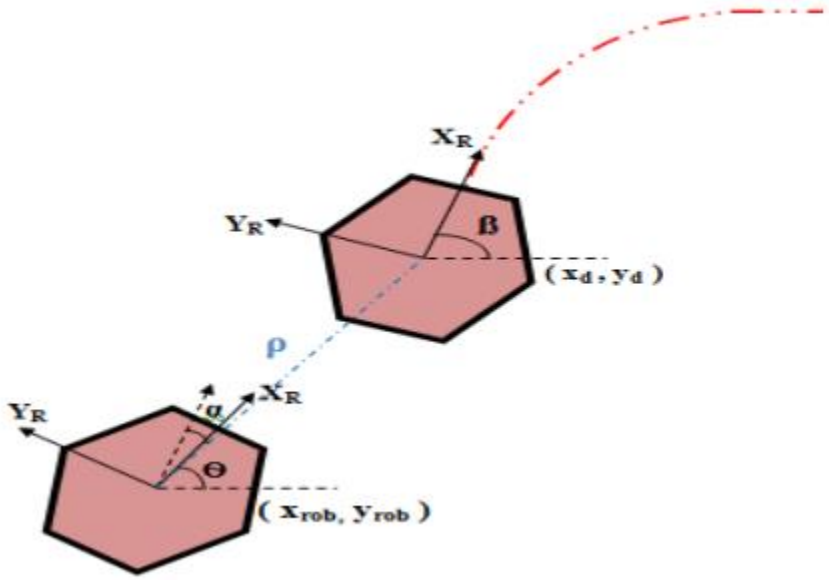

Figura 21: Pozita aktuale dhe e deshiruar e robotit [11]

Përshkrimi i robotit, në raport me sistemit global referent në koordinata karteziane do jetë:

$$\begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = w \end{cases}$$

Nga figura më lartë, me ρështë shënuar distanca mes pozitës aktuale dhe pozitës së dëshiruar të robotit. Duke pasur parasysh këtë, pozita e robotit po ashtu mund të përshkruhet me koordinata polare, duke pasur parasysh ρ si informatë shtesë. Përshkrimi me anë të koordinatave polare do jetë:

$$\begin{cases} \dot{\rho} = -v\cos(\beta - \theta) = -v\cos\alpha \\ \dot{\beta} = v\dfrac{\sin\alpha}{\rho} \\ \dot{\rho} = w \end{cases}$$

Ku α është këndi i treguar në figurën 21.

Po ashtu nga figura 21 shohim se ρ ka vlerën:

$$\rho = \sqrt{(x_d - x_c)^2 + (y_d - y_x)^2}$$

Andaj tani, vlerat e komponentëve të shpejtësive këndore gjatë lëvizjes së robotit do jenë:

$$\begin{cases} \dot{\rho} = -v\cos\alpha \\ \dot{\alpha} = -w + v\dfrac{\sin\alpha}{\rho} \\ \dot{\beta} = v\dfrac{\sin\alpha}{\rho} \end{cases}$$

Kjo formë e ndërtimit të një modeli kinematik, është një formë standarde e modelimit që bëhet për robotët mobil. Laramania mund të paraqitet ri rezultat i strukturave të ndryshme që mund të kenë

robotët. Nëse roboti mobil ka 4 rrota të motorizuara në mënyrë të pavarur, atëherë qasja e njëjtë do përdorej edhe në atë rast, mirëpo tani do duhet pasur parasysh trajtimin e ndikimi që secila rrotë ka ne zhvendosjen e robotit mobil. Një situatë të tillë do të trajtojmë në vijim.

3.4 Modelet e thjeshtuara kinematike për robotët mobil

Qëllimi i kinematikës direkte në robotikën e lëvizshme është të përcaktojë pozicionin dhe orientimin e robotit bazuar në matjet e rrotullimit të rrotave. Për ta arritur këtë, duhet krijuar modeli kinematikë i robotit. Si shembull do trajtojmë një robot mobil i njohur si ROSbot. ROSbot është robot i lëvizshëm me katër rrota me motor të veçantë për secilën rrotë, por për të thjeshtuar llogaritjen kinematike do ta trajtojmë atë si me dy rrota. Dy rrota virtuale (të shënuara si ËL dhe ËR në skemë) do të kenë bosht që kalon përmes qendrës gjeometrike të robotit. Në këtë mënyrë mund të përdorim modelin kinematikë më të thjeshtë të robotit me rrota diferenciale. Emri "diferencial" vjen nga fakti se roboti mund të ndryshojë drejtimin e tij duke ndryshuar shkallën relative të rrotullimit të rrotave të tij dhe nuk kërkon lëvizje shtesë të drejtimit. Skema e robotëve është paraqitur më poshtë.

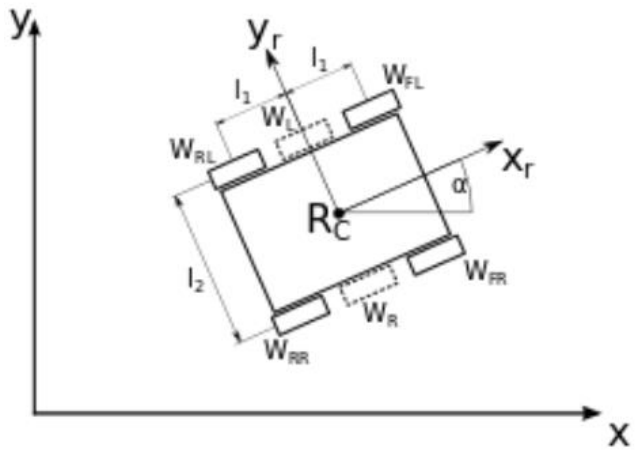

Figura 22: Roboti mobil me 4 rrota të motorizuara në formë të pavarur.

Parametrat e shënuar në figurën 22 janë:

$R_c - qendra\ gjeometrike\ e\ robtit$

$(x_c, y_x) - koordinata\ e\ qendres\ sheometrike\ t\ddot{e}\ robotit$

$(x_r, y_r)\ koordinatat\ ne\ raport\ me\ sistemit\ lokal$

$\alpha - pozita\ k\ddot{e}ndore\ e\ robotit$

$W_{FL} - rrota\ majtas\ para$

$W_{RL} - rrota\ majtas\ prapa$

$W_{FR} - rrota\ djathtas\ para$

$W_{RR} - rrota\ djathas\ prapa$

$W_L - rrota\ virtuale\ e\ majt\ddot{e}$

$W_R - rrota\ virtuale\ e\ djaht\ddot{e}$

$l_1 - distanca\ nga\ qendra\ tek\ rrotat\ para/prapa$

$l_2 - distanca\ nga\ qendra\ tek\ rrotat\ \frac{majtas}{djathtas}$

Roboti mobil ka kufizime. Mund të lëvizë vetëm në rrafshin x-y dhe ka 3 DOF (gradë lirie). Sidoqoftë, jo të gjitha DOF-të janë të kontrollueshme që do të thotë se roboti nuk mund të lëvizë në çdo drejtim të akseve të tij lokale (p.sh. nuk mund të lëvizë anash). Një sistem i tillë lëvizës quhet jo-holonomik. Kur sasia e DOF-ve të kontrollueshme është e barabartë me DOF-të totale, atëherë një robot mund të quhet holonomik. Për të arritur që disa robotë të lëvizshëm janë ndërtuar duke përdorur rrota Omni ose Mecanum dhe falë lëvizjes vektoriale ata mund të ndryshojnë pozicionin pa ndryshuar drejtimin (orientimin). Nëse robotin nga figura 22 e thjeshtojmë, duke marrë që lëvizja e tij bëhet vetëm nga dy rrota virtuale dhe jo nga katër, shpejtësia këndore ω dhe pozicioni këndor Φ e secilës rrotë virtuale do të jetë mesatarja e homologëve të saj realë. Pra do të kemi:

$$\phi_{W_L} = \frac{\phi_{W_{FL}} + \phi_{W_{RL}}}{2} \phi_{W_R} = \frac{\phi_{W_{FR}} + \phi_{W_{RR}}}{2}$$

$$\omega_{W_R} = \frac{\omega_{W_{FL}} + \omega_{W_{RL}}}{2} \omega_{W_R} = \frac{\omega_{W_{FR}} + \omega_{W_{RR}}}{2}$$

Kurse shpejtësia lineare e secilës rrotë do jetë:

$$v_R = \omega_{W_R} \times r_{v_L} = \omega_{W_L} \times r$$

Ku me r është shënuar rrezja e rrotës.

Ngjashëm, pozita këndore e robotit mund të caktohet duke shfrytëzuar barazitë më lartë. Do kemi:

$$\alpha = \left(\phi_{W_R} - \phi_{W_L}\right) \frac{r}{l_2} \dot{\alpha} = \frac{d\alpha}{dt}$$

Komponentët x dhe y të shpejtësisë do jenë:

$$\dot{x_c} = \left(v_L + \dot{\alpha} \frac{l_2}{2}\right) \cos\alpha\ \dot{y_c} = \left(v_L + \dot{\alpha} \frac{l_2}{2}\right) \sin\alpha$$

4. Sistemi operativ i robotit (ang. ROS - robot operating system)

Sistemi Operativ i Robotit (ROS ose ros) është një paketë programesh ndërmjetëse robotike me burim të hapur. Edhe pse ROS nuk është një sistem operativ, por një koleksion kornizash punuese softuerësh për zhvillimin e softuerit robotik, ai ofron shërbime të krijuara për një grup heterogjen kompjuterik si abstraksioni i harduerit, kontrolli i pajisjes në nivel të ulët, zbatimi i funksioneve të përdorura zakonisht, kalimi i mesazheve midis proceseve, dhe menaxhimin e paketave. Kompletet e ekzekutimit të proceseve të bazuara në ROS përfaqësohen në një arkitekturë grafike ku përpunimi kryhet në nyje që mund të marrin, postojnë dhe të shumëfishojnë të dhënat e sensorit, kontrollin, gjendjen, planifikimin, aktivizuesin dhe mesazhe të tjera. Pavarësisht rëndësisë së reaktivitetit dhe vonesës së ulët në kontrollin e robotëve, vetë ROS nuk është një OS në kohë reale (RTOS). Megjithatë, është e mundur të integrohen ROS me kodin në kohë reale. Mungesa e mbështetjes për sistemet në kohë reale është adresuar në krijimin e ROS 2, një rishikim i madh i API-së ROS i cili do të përfitojë nga bibliotekat dhe teknologjitë moderne për funksionalitetin bazë të ROS dhe do të shtojë mbështetje për kodin në kohë reale dhe pajisjen e integruar.

Sistemi Operativ i Robotit (ROS) është një kornizë që, në ditët e sotme, është gjerësisht e pranuar dhe e përdorur në komunitetin e robotikës. Qëllimi i tij kryesor është t'i bëjë komponentët e shumtë të një sistemi robotik të lehtë për t'u zhvilluar dhe ndarë, në mënyrë që ata të mund të punojnë në robotë të tjerë me ndryshime minimale. Kjo në thelb lejon ripërdorimin e kodit dhe përmirëson cilësinë e kodit duke e testuar atë nga një numër i madh përdoruesish dhe platformash. ROS u zhvillua fillimisht në 2007 nga Laboratori i Inteligjencës Artificiale Stanford (SAIL) në mbështetje të projektit Stanford AI Robot. Që nga viti 2008, Willow Garage vazhdoi zhvillimin dhe së fundmi Fondacioni i Robotikës me Burim të Hapur (OSRF) filloi të mbikëqyrë mirëmbajtjen e ROS dhe projekteve partnere, si Gazebo, duke përfshirë zhvillimin e veçorive të reja. ROS nganjëherë quhet një sistem operativ meta sepse kryen shumë funksione të një sistemi operativ, por kërkon një sistem operativ të një kompjuteri si Linux. Një nga qëllimet kryesore të tij është të sigurojë komunikim ndërmjet përdoruesit, sistemit operativ të kompjuterit dhe pajisjeve të jashtme të kompjuterit. Kjo pajisje mund të përfshijë sensorë, kamera, si dhe robotë. Si me çdo sistem operativ, përfitimi i ROS është abstraksioni i harduerit dhe aftësia e tij për të kontrolluar një robot pa pasur nevojë që përdoruesi të dijë të gjitha detajet e robotit. Për shembull, për të lëvizur krahët e një roboti, lëshohet një komandë ROS, ose skriptet në Python ose C++ të shkruara nga projektuesit e robotëve bëjnë që roboti të përgjigjet siç urdhërohet. Skriptet, nga ana tjetër, mund të thërrasin programe të ndryshme kontrolli që shkaktojnë lëvizjen aktuale të krahëve të robotit. Është gjithashtu e mundur të dizajnoni dhe simuloni robotin tuaj duke përdorur ROS.

ROS është një sistem softuerësh robotik me burim të hapur që mund të përdoret pa tarifa licencimi nga universitetet, agjencitë qeveritare dhe kompanitë tregtare. Përparësitë e softuerit me burim të hapur është se kodi burimor për sistemin është i disponueshëm dhe mund të modifikohet sipas nevojave të një përdoruesi. Më e rëndësishmja për disa përdorues, softueri mund të përdoret në një produkt komercial për sa kohë që citohen licencat e duhura. Softueri mund të përmirësohet dhe modulet mund të shtohen nga përdoruesit dhe kompanitë. ROS përdoret nga mijëra përdorues në mbarë botën dhe njohuritë mund të ndahen midis përdoruesve. Përdoruesit variojnë nga hobiistët tek zhvilluesit profesionistë të robotëve komercialë. Përveç grupit të madh të studiuesve ROS, ekziston një grup ROS-Industrial i dedikuar për aplikimin e softuerit ROS për robotët për prodhim. Disa nga robotët që përdorin ROS për të operuar janë Turtlebot, Baxter, Crazyflie, Care-o-bot etj. Disa nga këta robot janë treguar në figurën më poshtë.

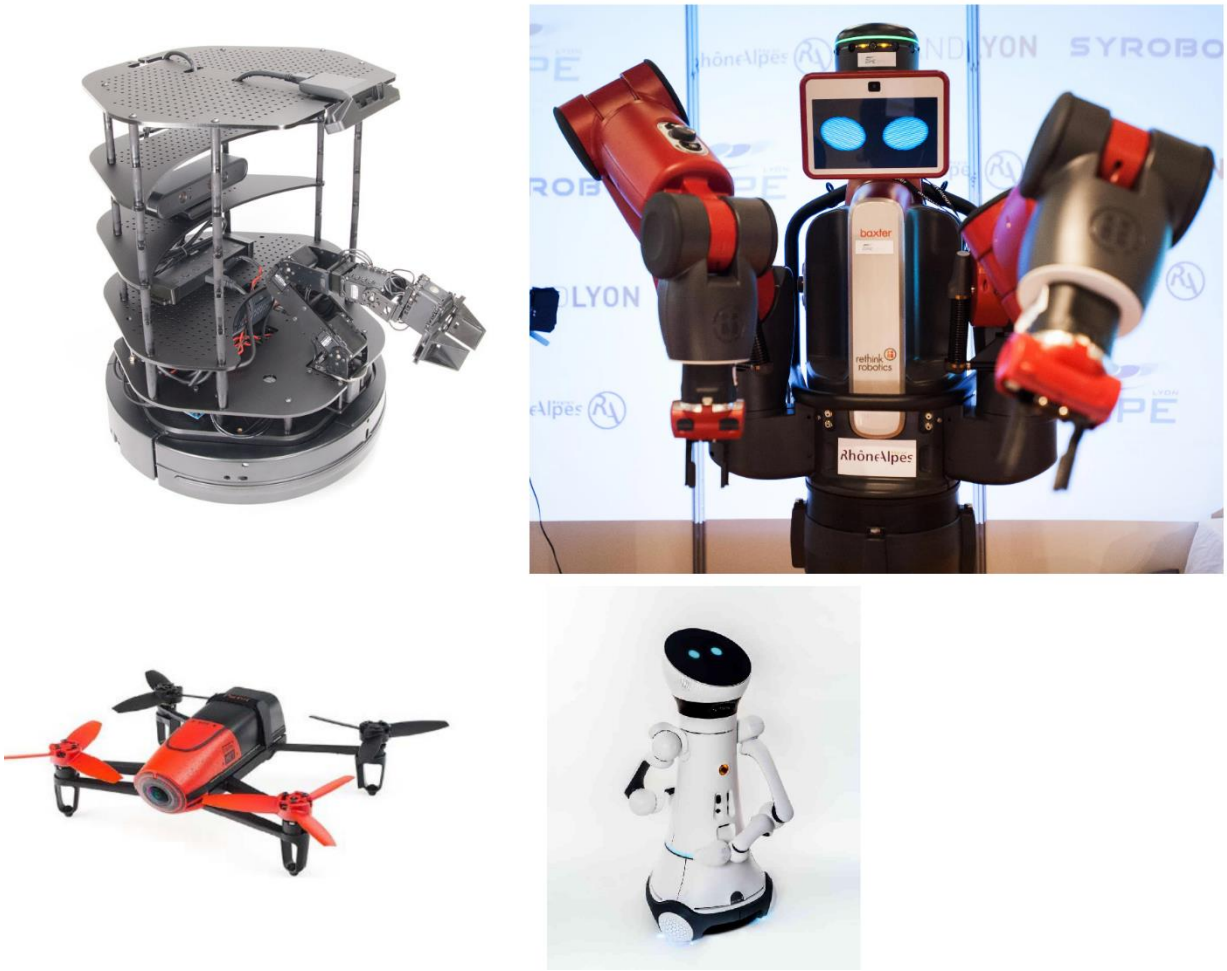

Figura 23: Disa nga robotët mobil që përdorin ROS

4.1 Arkitektura dhe konceptet e ROS

Një robot mund të ketë shumë sensorë dhe aktuatorë, si dhe një njësi llogaritëse. Si mund të kontrollojmë shumë aktuatorë dhe të përpunojmë kaq shumë të dhëna sensori? A mund ta bëjmë atë në një program të vetëm? Po, por kjo nuk është një mënyrë e mirë për ta bërë atë. Mënyra më e mirë është, ne mund të shkruajmë programe të pavarura për të trajtuar të dhënat e sensorëve dhe aktuatorëve, dhe shpesh mund të na duhet të shkëmbejmë të dhëna midis këtyre programeve. Kjo është situata ku ne përdorim ROS. Pra, a mund të programojmë një robot pa ROS? Po, por kompleksiteti i softuerit rritet sipas numrit të aktivizuesve dhe sensorëve. Figura 24 tregon dy programe të shënuara si nyja 1 dhe nyja 2. Kur ndonjë nga programet nis, një nyje komunikon me një program ROS të quajtur master ROS. Nyja dërgon të gjithë informacionin e saj te masteri ROS, duke përfshirë llojin e të dhënave që dërgon ose merr. Nyjet që dërgojnë një të dhënë quhen nyje transmetuese, dhe nyjet që marrin të dhëna quhen nyje abonentë. ROS Master ka të gjitha informacionet e transmetuesit dhe abonentëve që funksionojnë në kompjuterë. Nëse nyja 1 dërgon të dhëna të veçanta të quajtura "A" dhe të njëjtat të dhëna kërkohen nga nyja 2, atëherë masteri ROS dërgon informacionin tek nyjet në mënyrë që ato të komunikojnë me njëri-tjetrin.

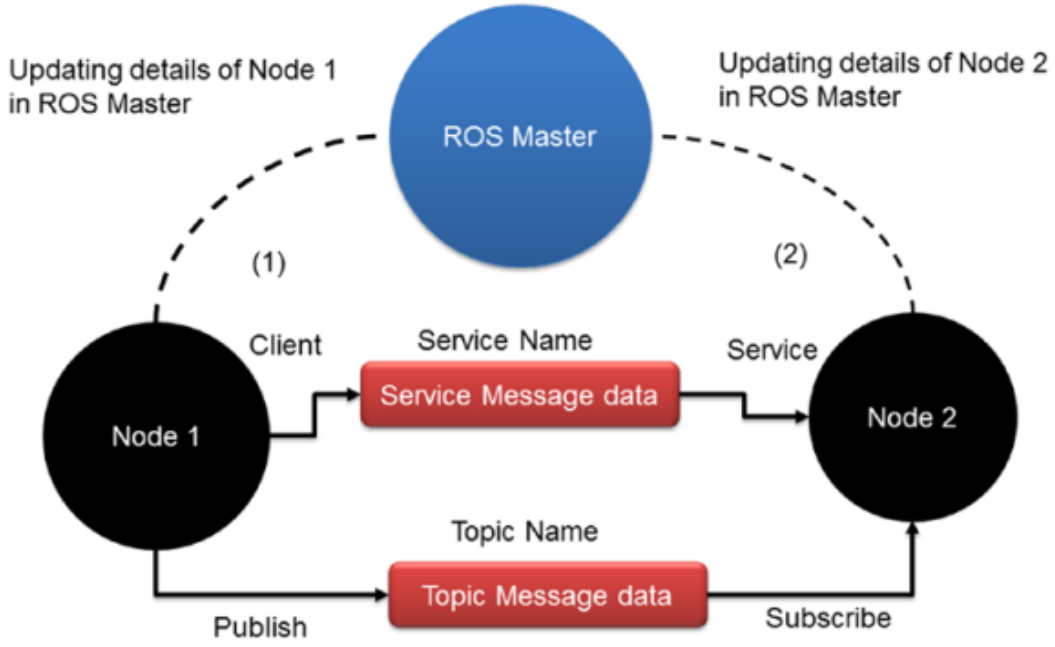

Figura 24:Block diagrami i komunikimeve me ROS [7]

Nyjat ROS mund t'i dërgojnë lloje të ndryshme të dhënash njëra-tjetrës, e cila përfshin lloje primitive të të dhënave si numër i plotë, float, string, etj. Llojet e ndryshme të të dhënave që dërgohen quhen mesazhe ROS. Me mesazhet ROS, ne mund të dërgojmë të dhëna me një lloj të vetëm të dhënash ose të dhëna të shumta me lloje të ndryshme të dhënash. Këto mesazhe dërgohen përmes një busi ose shtegu mesazhesh të quajtur tema ROS (ang. ROS topic). Çdo temë ka një emër; për shembull, një temë e quajtur "chatter" dërgon një mesazh vargu.

Arkitektura ROS është projektuar dhe ndarë në tre seksione ose nivele konceptesh:

- Niveli i sistemit të skedarëve (ang. The Filesystem level)
- Niveli i grafikut të llogaritjes (ang. The Computation Graph level)
- Niveli i komunitetit (ang. The Community level)

Niveli i parë është niveli i sistemit të skedarëve. Në këtë nivel, një grup konceptesh përdoren për të shpjeguar se si është formuar nga brenda ROS, strukturën e dosjeve dhe numrin minimal të skedarëve që i nevojiten për të punuar.

Niveli i dytë është niveli i Grafikut të Llogaritjes, ku ndodh komunikimi ndërmjet proceseve dhe sistemeve. Në këtë seksion, ne do të shohim të gjitha konceptet dhe mekanizmat që ka ROS për të vendosur sisteme, për të trajtuar të gjitha proceset dhe për të komunikuar me më shumë se një kompjuter të vetëm, etj.

Niveli i tretë është niveli i Komunitetit, i cili përfshin një sërë mjetesh dhe konceptesh për të ndarë njohuritë, algoritmet dhe kodin midis zhvilluesve. Ky nivel ka një rëndësi të madhe; si me shumicën e projekteve të softuerit me burim të hapur, të kesh një komunitet të fortë jo vetëm që përmirëson aftësinë e të sapoardhurve për të kuptuar ndërlikimet e softuerit, si dhe për të zgjidhur çështjet më të zakonshme, por është gjithashtu forca kryesore që nxit rritjen e tij.

4.1.1 Niveli i sistemit të skedarëve (ang. ROS Filesystem level)

Sistemi i skedarëve ROS përfshin paketat, paketat meta, manifestet e paketave, depot, llojet e mesazheve dhe llojet e shërbimeve. Paketat ROS janë njësitë individuale, ose njësitë atomike, të softuerit ROS. Të gjithë kodet burimore, skedarët e të dhënave, skedarët e ndërtimit, varësitë dhe skedarët e tjerë janë të organizuar në paketa. Një paketë meta ROS grupon një grup paketash të ngjashme për një aplikacion specifik. Një paketë meta ROS nuk ka asnjë skedar burimi ose skedar të dhënash. Ai ka varësitë e paketave të ngjashme. Një meta paketë ROS organizon një sërë paketash. Një manifest i paketës është një skedar XML i vendosur brenda një pakete ROS. Ai ka të gjitha informacionet kryesore të një pakete ROS, duke përfshirë emrin e paketës, përshkrimin, autorin, varësitë, e kështu me radhë.

Qëllimi kryesor i sistemit të skedarëve ROS është të centralizojë procesin e ndërtimit të një projekti, ndërsa në të njëjtën kohë të sigurojë fleksibilitet dhe mjete të mjaftueshme për të decentralizuar varësitë e tij. Skema funksionale e këtij niveli është treguar në figurën 25.

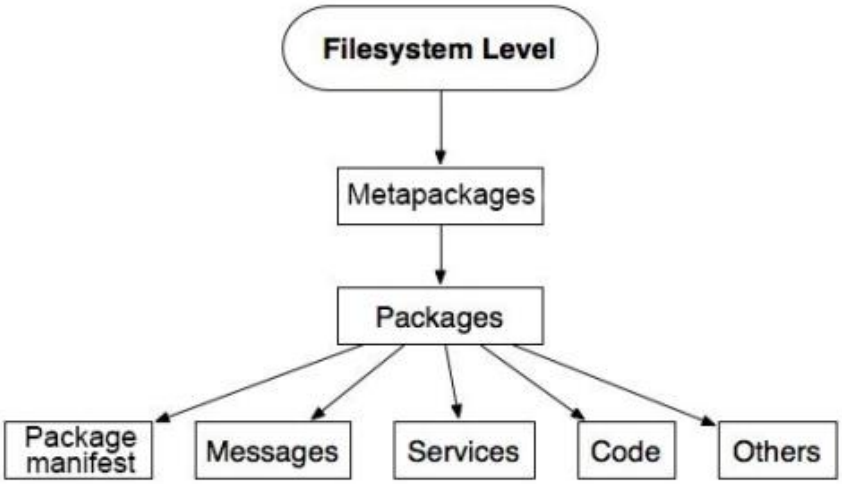

Figura 25: Skema funksionale e nivelit Filesystem [7]

4.1.2 Niveli i llogaritjes (ang. Computation graph level)

ROS krijon një rrjet ku të gjitha proceset janë të lidhura. Çdo nyje në sistem mund të hyjë në këtë rrjet, të ndërveprojë me nyje të tjera, të shohë informacionin që ata po dërgojnë dhe të transmetojë të dhëna në rrjet. Ky funksionaliteti i takon nivelit të llogaritjes, dhe skema funksionale e këtij niveli është treguar në figurën 26.

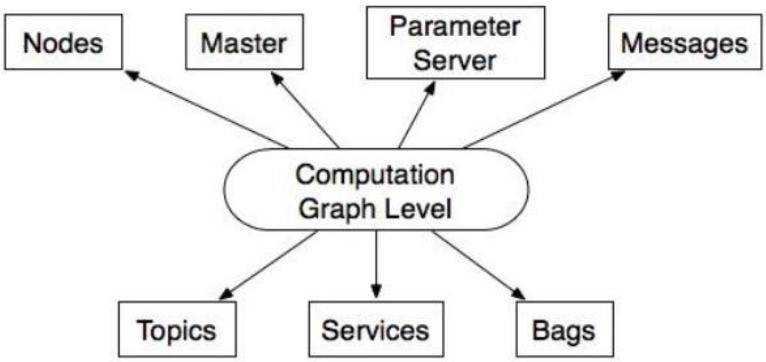

Figura 26: Skema funksionale e nivelit të llogaritjes [7]

Konceptet bazë në këtë nivel janë nyjat, masteri, serveri i parametrave, mesazhet, shërbimet, temat (ang. topics) dhe çanta (ang. bags), të cilat të gjitha i ofrojnë të dhëna grafikut në mënyra të ndryshme dhe shpjegohen në vijim.

Nyjat: Nyjat janë procese ku bëhet llogaritja. Nëse dëshironi të keni një proces që mund të ndërveprojë me nyje të tjera, duhet të krijoni një nyje me këtë proces për ta lidhur atë me rrjetin ROS. Zakonisht, një sistem do të ketë shumë nyje për të kontrolluar funksione të ndryshme. Do të shihni se është më mirë të keni shumë nyje që ofrojnë vetëm një funksionalitet të vetëm, sesa të keni një nyje të madhe që bën gjithçka në sistem.

Master: Master siguron regjistrimin e emrave dhe shërbimin e kërkimit për pjesën tjetër të nyjave. Ai gjithashtu vendos lidhje midis nyjave. Nëse nuk e keni në sistemin tuaj, nuk mund të komunikoni me nyje, shërbime, mesazhe dhe të tjera. Në një sistem të shpërndarë, ju do të keni masterin në një kompjuter dhe mund të ekzekutoni nyjat në këtë ose kompjuterë të tjerë.

Serveri i parametrave: Serveri i parametrave na jep mundësinë e përdorimit të çelësave për të ruajtur të dhënat në një vendndodhje qendrore. Me këtë parametër, është e mundur të konfiguroni nyjat gjatë ekzekutimit ose të ndryshoni parametrat e punës së një nyje.

Mesazhet: Nyjat komunikojnë me njëra-tjetrën përmes mesazheve. Një mesazh përmban të dhëna që u japin informacion nyjave të tjera. ROS ka shumë lloje mesazhesh, dhe ju gjithashtu mund të zhvilloni llojin tuaj të mesazhit duke përdorur llojet standarde të mesazheve.

Temat (ang. topics): Çdo mesazh duhet të ketë një emër që do të drejtohet nga rrjeti ROS. Kur një nyje po dërgon të dhëna, themi se nyja po publikon një temë. Nyjat mund të marrin tema nga nyjet e tjera thjesht duke u abonuar në temë. Një nyje mund të abonohet në një temë edhe nëse nuk ka asnjë nyje tjetër që publikon këtë temë specifike. Kjo na lejon të shkëputim prodhimin nga konsumi. Është e rëndësishme që emrat e temave të jenë unikë për të shmangur problemet dhe konfuzionin midis temave me të njëjtin emër.

Shërbimet: Kur publikoni tema, po dërgoni të dhëna në një mënyrë shumë-për-shumë, por kur keni nevojë për një kërkesë ose një përgjigje nga një nyje, nuk mund ta bëni këtë me tema. Shërbimet na japin mundësinë e ndërveprimit me nyjat. Gjithashtu, shërbimet duhet të kenë një emër unik. Kur një nyje ka një shërbim, të gjitha nyjat mund të komunikojnë me të, falë bibliotekave të klientëve ROS.

Çantat: Çantat janë një format për të ruajtur dhe riprodhuar të dhënat e mesazheve ROS. Çanta janë një mekanizëm i rëndësishëm për ruajtjen e të dhënave, të tilla si të dhënat e sensorëve, që mund të jenë të vështira për t'u mbledhur, por janë të nevojshme për të zhvilluar dhe testuar algoritme. Do të përdorni shumë çanta ndërsa punoni me robotë kompleksë.

4.1.3 Niveli i komunitetit (ang. Community level)

Konceptet e nivelit të komunitetit ROS janë burimet ROS që u mundësojnë komuniteteve të veçanta të shkëmbejnë softuer dhe njohuri. Këto burime përfshijnë sa vijon:

- Shpërndarjet: Shpërndarjet ROS janë koleksione të metapaketave të versionuara që mund t'i instaloni. Shpërndarjet ROS luajnë një rol të ngjashëm me shpërndarjet Linux. Ata e bëjnë më të lehtë instalimin e një koleksioni softuerësh dhe gjithashtu mbajnë versione të qëndrueshme në një grup softuerësh.
- Depot: ROS mbështetet në një rrjet të federuar të depove të kodit, ku institucione të ndryshme mund të zhvillojnë dhe lëshojnë komponentët e tyre të softuerit robotik.
- ROS Wiki: ROS Wiki është forumi kryesor për dokumentimin e informacionit rreth ROS. Çdokush mund të regjistrohet për një llogari, të kontribuojë me dokumentacionin e vet, të sigurojë korrigjime ose përditësime, të shkruajë tutoriale dhe më shumë.
- Sistemi i biletave të gabimeve: Nëse gjeni një problem ose dëshironi të propozoni një veçori të re, ROS ka këtë burim për ta bërë atë.

4.2 Instalimi i ROS

Në vijim do të shohim se si mund ta instalojmë ROS. Me saktësisht, do të shohim se si mund ta instalojmë ROS Noetic në Ubutnu 20.02.02 LTS. Të gjitha detajet për instalimin e ROS janë të trajtuara në linkun: https://varhowto.com/install-ros-noetic-ubuntu-20-04/

4.3 ROS dhe Turtlebot

Pasi të jetë instaluar ROS, mund t'i ekzekutojmë disa komanda themelore, me anë të të cilave mund të testohet nëse instalimi është bërë si duhet dhe poashtu për të testuar funksionalitetin e inftrastrukturës së robotit. Për ta testuar nëse instalimi është bërë si duhet, ekzekutojmë instruksionin në vijim:

```
roscore
```

Këtë instruksion e ekzekutojmë në terminalin e komandave të sistemit operativ. Nëse instalimi është bërë si duhet, atëherë përgjigja që fitojmë do jetë si ajo e treguar më poshtë.

```
turtlebot@turtlebot-X200CA:~$ roscore
... logging to /home/turtlebot/.ros/log/6ef6185c-9127-11e4-83da-0c84dc11754b/ros
launch-turtlebot-X200CA-9168.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://192.168.0.8:45853/
ros_comm version 1.11.9


SUMMARY
========

PARAMETERS
 * /rosdistro: indigo
 * /rosversion: 1.11.9

NODES

auto-starting new master
process[master]: started with pid [9180]
ROS_MASTER_URI=http://192.168.0.8:11311/

setting /run_id to 6ef6185c-9127-11e4-83da-0c84dc11754b
process[rosout-1]: started with pid [9193]
started core service [/rosout]
```

Figura 27: Përgjigja që roboti jep për komandën roscore

Për ta testuar infrastrukturën e robotit, një nga testet më të shpeshta që përdoret është teleoperimi. Kjo nënkupton mundësinë e komandimit manual të lëvizjeve të robotit me anë të tastierës. Parakusht për të testuar këtë funksionalitet është që baza Kobuki të jetë aktive. Për ta bërë këtë, fillimisht duhet ekzekutuar instruksioni më poshtë:

```
roslaunch turtlebot_bringup minimal.launch
```

Ky instruksion në vetë bën ekzekutimin e një file-i më anë të të cilit aktivizohen driverat bazik për lëvizje që janë pjesë e bazës Kobuki. Në fakt, ky instruksion është pjesë në një librarie më të madhe që ka emrin Turtlebot Bringup, e cila library ka në vete implementimin e komunikimin të të gjithë infrastukturës që janë pjesë e Turtlebot 2i, ashtu që komponentët e këtij roboti I bën kompatibil për ekzekutim në ROS.

Pasi ky instruksion të jetë ekzekutuar, nëse ekzekutimi është bërë në mënyrë të suksesshme, nga roboti duhet dëgjuar një konfirmin audio, që tregon se roboti është gati. Për të mundësuar teleoperimin, duhet ekzekutuar instruksioni më poshtë:

```
roslaunch turtlebot_teleop keyboard_teleop.launch
```

Nëse ekzekutimi ka qenë i suksesshëm, atëherë fitojmë përgjigjen si në figurën më poshtë.

```
Control Your Turtlebot!
---------------------------
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%
space key, k : force stop
anything else : stop smoothly

CTRL-C to quit

currently:      speed 0.2       turn 1
```

Figura 28: Përgjigja pas komandës për teleoperim

Më anë të kësaj komande mund të bëjmë lëvizjen manuale të roboti, kontrollin e shpejtësisë lineare dhe këndore të lëvizjes etj.

Në mënyrë të ngjashme mund të ekzekutohet edhe komanda tjera që janë pjesë e librarisë Turtlebot Bringup, me qëllim të testimit të funksionalitetit të robotit. Në vijim, do të shohim se si mund të përdorim këto asete infrastrukturore për të bërë navigim të robotëve mobil.

5. Navigimi i robotëve mobil – Algoritmi SLAM

Navigimi i robotëve është problemi i drejtimit të një roboti drejt një qëllimi. Qasja njerëzore ndaj navigimit është krijimi i hartave dhe ngritja e tabelave, dhe në shikim të parë duket qartë se robotët duhet të veprojnë në të njëjtën mënyrë. Sidoqoftë, shumë detyra robotike mund të arrihen pa asnjë hartë fare, duke përdorur një qasje të njohur si navigacion reaktiv. Për shembull, drejtimi drejt një drite, duke ndjekur një vijë të bardhë në tokë, duke lëvizur nëpër një labirint duke ndjekur një mur, ose duke pastruar një dhomë duke ndjekur një rrugë të rastësishme. Roboti po reagon drejtpërdrejt në mjedisin e tij: intensiteti i dritës, pozicioni relativ i vijës së bardhë ose kontakti me një mur. Roboti Elsie i Grey Walter demonstroi sjellje "të ngjashme me jetën" - ajo reagoi ndaj mjedisit të saj dhe mund të kërkonte një burim drite. Sot më shumë se 5 milionë fshesa me korrent Roomba pastrojnë dyshemetë pa përdorur asnjë hartë të dhomave ku punojnë. Robotët punojnë duke bërë lëvizje të rastësishme dhe duke ndjerë vetëm se kanë kontaktuar me një pengesë.

Navigimi më i njohur i bazuar në harta të stilit njerëzor përdoret nga robotë më të sofistikuar. Kjo qasje mbështet detyra më komplekse, por në edhe në vetvete është më komplekse. Ai imponon një sërë kërkesash, ndër të cilat më e rëndësishmja është harta e mjedisit. Gjithashtu kërkon që pozicioni i robotit të dihet gjithmonë. Në vijim do të diskutojmë se si robotët mund të përcaktojnë pozicionin e tyre dhe të krijojnë harta. Pjesa tjetër e diskuton qasjet reaktive dhe të bazuara në harta për navigimin e robotëve me fokus në robotët me rrota që veprojnë në një mjedis planar.

5.1 Hyrje në navigim

Navigimi është një nga kërkesat më sfiduese që kërkohen nga një robot celular. Suksesi në navigim kërkon sukses në katër blloqet ndërtuese të navigimit: perceptimin (roboti duhet të interpretojë sensorët e tij për të nxjerrë të dhëna kuptimplota); lokalizimi (roboti duhet të përcaktojë pozicionin e tij në mjedis); njohja (Ang. congition) (roboti duhet të vendosë se si të veprojë për të arritur qëllimet e tij); dhe kontrollin e lëvizjes (roboti duhet të modulojë daljet e motorit të tij për të arritur trajektoren e dëshiruar).Nga këto katër shtylla, lokalizimi është problemi i parë serioz trajtohet. Lokalizimi ka të bëjë me aftësinë e robotit për të përcaktuar pozitën e tij në raport me mjedisin në të cilin ndodhet. Kjo informatë është kritike për përcaktimin e kursit të veprimit për robotët mobil. Skematikisht, kjo është treguar në figurën në vijim.

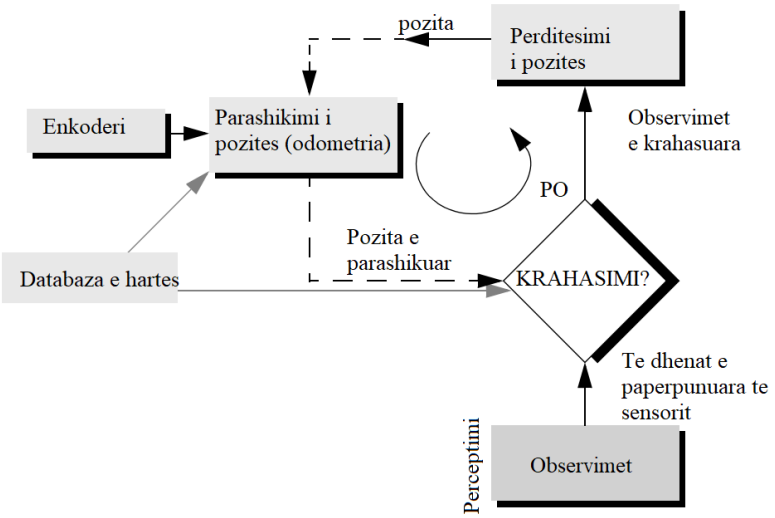

Figura 29: Skema e përgjithshme e problemit të lokalizimit të robotit [4]

Nëse mund të bashkëngjitet një sensor të saktë GPS (sistemi i pozicionimit global) në një robot të lëvizshëm, shumica e problemit të lokalizimit do të shmangej. GPS do ta informonte robotin për pozicionin e tij të saktë, brenda dhe jashtë, në mënyrë që të përgjigjet në pyetjen, "Ku jam unë?" do të ishte gjithmonë në dispozicion. Fatkeqësisht, një sensor i tillë aktualisht nuk është praktik. Rrjeti ekzistues GPS ofron saktësi deri në disa metra, gjë që është e papranueshme për lokalizimin e robotëve të lëvizshëm në shkallë njerëzore, si dhe robotëve celularë në miniaturë, si robotët e tavolinës dhe nanorobotët që navigojnë në trupin e së ardhmes. Për më tepër, teknologjitë GPS nuk mund të funksionojnë në ambiente të mbyllura ose në zona të penguara dhe kështu janë të kufizuara në hapësirën e tyre të punës.Por, duke parë përtej kufizimeve të GPS, lokalizimi nënkupton më shumë sesa njohjen e pozicionit absolut të dikujt në kornizën e referencës së Tokës. Konsideroni një robot që ndërvepron me njerëzit. Ky robot mund të ketë nevojë të identifikojë pozicionin e tij absolut, por pozicioni i tij relativ në lidhje me njerëzit e synuar është po aq i rëndësishëm. Detyra e tij e lokalizimit mund të përfshijë identifikimin e njerëzve duke përdorur grupin e tij të sensorëve, më pas llogaritjen e pozicionit të tij relativ me njerëzit. Për më tepër, gjatë hapit të njohjes një robot do të zgjedhë një strategji për arritjen e qëllimeve të tij. Nëse synon të arrijë një vend të caktuar, atëherë lokalizimi mund të mos jetë i mjaftueshëm. Robotit mund t'i duhet të përvetësojë ose të ndërtojë një model mjedisor, një hartë, që e ndihmon atë në planifikimin e një rruge drejt qëllimit. Edhe një herë, lokalizimi do të thotë më shumë sesa thjesht përcaktimi i një poze absolute në hapësirë; do të thotë të ndërtosh një hartë, pastaj të identifikosh pozicionin e robotit në lidhje me atë hartë.

5.2 Ndërtimi i hartës me Turtlebot 2i

Fillimisht duhet theksuar se një nga përparësitë e ROS është se mund të punojë edhe në mjedise të panjohura. Mjedis i panjohur nënkupton një hapësirë në të cilën roboti nuk ka qenë më parë, apo për të cilën nuk ka informata startuese. Kjo ngase robotët të bazuar në ROS zakonisht janë të pajisur me një mori sensorësh apo kamerash, përmes të cilëve mund të gjenerohet një hartë orientuese, dhe me pas problemi i lokalizimi ka të bëjë me përcaktimin e pozitës së robotit në rapor me vetë këtë hartë.

Instruksioni i parë që duhet ekzekutuar është ai që funksionalizom infrastrukturës e robotit:

```
roslaunch turtlebot_bringup minimal.launch
```

Kjo aktivizon driverët që komunikojnë me pjesët fizike të robotit. Më pas ekzekutojmë instruksionin:

```
roslaunch turtlebot_navigation gmapping_demo.launch
```

Ky instruksion aktivizon driverat e dedikuar për lokalizim. Në rastin e turtlebot 2i, varësisht nga varianti, kjo nënkupton aktivizimin e kamerave me thellësi (ang. Depth cameras) dhe lidarit.

Më pas ekzekutojmë:

```
roslaunch turtlebot_rviz_launchers view_navigation.launch
```

Instruksioni i fundit na mundëson të kemi një reprezentim grafik, lehtë të lexueshëm, të informatave që roboti është duke lexuar nga sensorët që përdorë për navigim. Kjo nënkupton leximin e hapësirës në të cilën ndodhet dhe po ashtu të pengesave. Në rastin tonë, leximi që vie nga turtlebot për mjedisin në të cilin ndodhet është treguar në figurën 30.

Për të gjeneruar një hartë dhe për ta ruajtur atë hartë, është e udhës që robotin ta lëvizim përgjatë tërë hapësirën të cilën dëshirojmë ta mapojmë. Ashtu si më herët, këtë mund ta bëjmë duke aktivizuar mekanizmin për teleoperim të robotit. Kjo bëhet me instruksionin më poshtë.

```
roslaunch turtlebot_teleop keyboard_teleop.launch
```

Ky instruksion, ashtu si me parë, do në ta e ofrojë një mënyre manuale për ta lëvizur robotin përgjatë hapësirës ku ndodhet. Duke theksuar se si pjesë integrale e kësaj lëvizje përfshihet edhe detektimi dhe evitimi i pengesave.

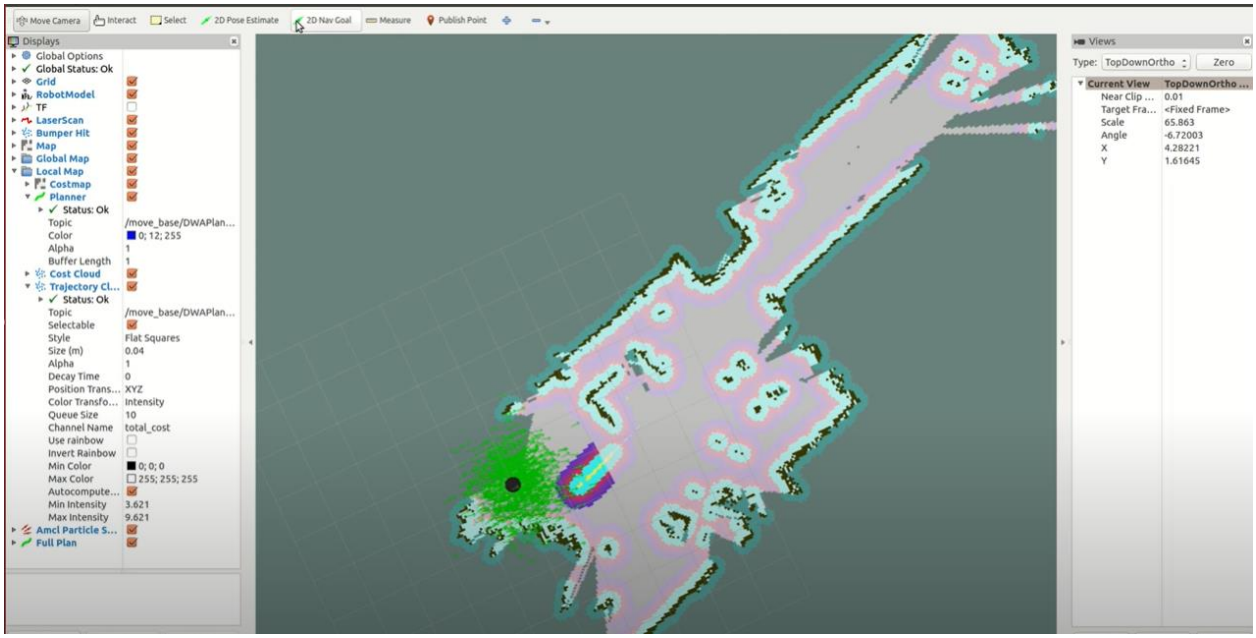

Figura 30: Leximi nga infrastruktura për navigim e turtlebot 2i

Pasi ta kemi lëvizur robotin përgjatë hapësirës për të cilën dëshirojmë të krijojmë hartën, më pas duhet ruajtur kjo hartë në kompjuterin e robotit. Kjo bëhet me anë të instruksionit të mëposhtëm.

```
rosrun map_server map_saver -f /tmp/harta_ime
```

Pra tani, me instruksionin më lartë, të gjitha incizimet bëra janë ruajtur në një file me emrin harta_ime, të cilin file më pas mund ta përdorim për të gjeneruar trajektor të lëvizjes së robotit.

Kjo qasje mund të përdoret, dhe poashtu është e rekomandueshme të përdoret, për të gjeneruar harta për të gjitha llojet e mjediseve në të cilat roboti paraprakisht nuk ka qene. Procedura e përgjithshme e krijimit të një harte për një mjedis të çfarëdoshëm është ilustruar në vijim.

Fillimisht po krijomë një folder, ku do të ruhen hartat tona të krijuara për mjedisin në të cilin ndodhet roboti. Kjo bëhet me instruksionin në vijim.

```
mkdir ~/turtlebot_custom_maps
```

Më pas starojmë Gazebon:

```
roslaunch turtlebot_gazebo turtlebot_world.launch
```

Gazebo është një imitues 3D, ndërsa ROS shërben si ndërfaqe për robotin. Kombinimi i të dyjave rezulton në një simulator të fuqishëm robotik. Me Gazebo mund të krijoni një skenar 3D në kompjuterin tuaj me robotë, pengesa dhe shumë objekte të tjera. Gazebo përdor gjithashtu një motor fizik për ndriçimin, gravitetin, inercinë, etj. Ju mund të vlerësoni dhe testoni robotin tuaj në skenarë të vështirë ose të rrezikshëm pa asnjë dëm për robotin tuaj. Shumicën e kohës është më e shpejtë të ekzekutosh një simulator në vend që të nisësh të gjithë skenarin në robotin tënd të vërtetë. Fillimisht Gazebo ishte krijuar për të vlerësuar algoritmet për robotët. Për shumë aplikacione është thelbësore të testoni aplikacionin tuaj robot, si trajtimi i gabimeve, jetëgjatësia e baterisë, lokalizimi, navigimi dhe kapja. Meqenëse kishte nevojë për një simulator me shumë robotë, Gazebo u zhvillua dhe u përmirësua.

Më pas, thërrasim skriptin që mundëson interpretimin grafik të informatave të lexuara nga sensoret e robotit. Kjo bëhet me instruksionet e mëposhtëm:

```
roslaunch turtlebot_gazebo gmapping_demo.launch
```

```
roslaunch turtlebot_rviz_launchers view_navigation.launch
```

Për ta mapuar tërë hapësirën përreth, është e nevojshme që robotin të lëvizim. Më së lehti kjo është të bëhet përmes teleoperimit. Këtë e bëjmë me instruksionin:

```
roslaunch turtlebot_teleop keyboard_teleop.launch
```

Ajo çfarë roboti sheh përmes sensorëve, tani do jetë e vizualizuar në Rviz dhe duket sikurse në figurën më poshtë.

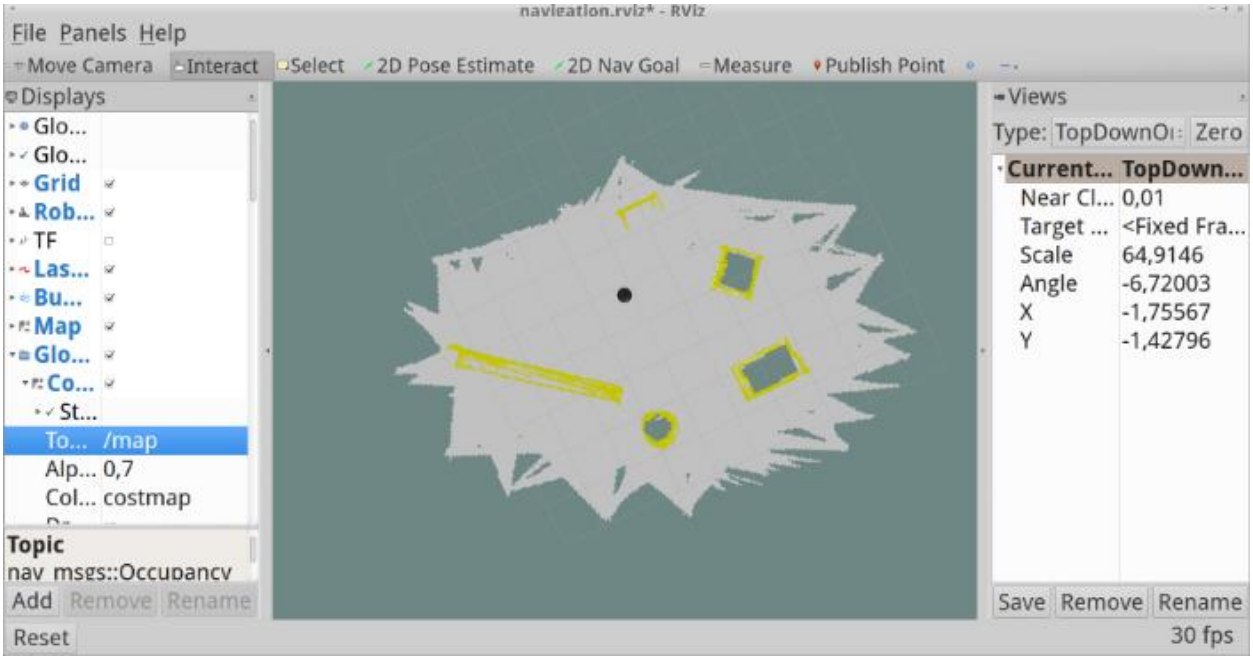

Figura 31:Pamja në Rviz për mapimin e mjedisit [4]

5.3 Lokalizimi përmes odometrisë

Fjala odometri është e përbërë nga fjalët greke odos (që do të thotë "rrugë") dhe metron (që do të thotë "masë"). Odometria është përdorimi i të dhënave nga sensorët e lëvizjes për të vlerësuar ndryshimin e pozicionit me kalimin e kohës. Përdoret në robotikë nga disa robotë me këmbë ose me rrota për të vlerësuar pozicionin e tyre në lidhje me një vend fillestar. Kjo metodë është e ndjeshme ndaj gabimeve për shkak të integrimit të matjeve të shpejtësisë me kalimin e kohës për të dhënë vlerësime të pozicionit. Mbledhja e shpejtë dhe e saktë e të dhënave, kalibrimi dhe përpunimi i instrumenteve kërkohen në shumicën e rasteve që odometria të përdoret në mënyrë efektive.

Po supozojmë se një robot i thjeshtë ka dy rrota të cilat mund të lëvizin përpara ose mbrapa dhe se ato janë të pozicionuara paralelisht me njëra-tjetrën dhe në distancë të barabartë nga qendra e robotit. Më tej, supozoni se çdo motor ka një enkoder inkremental rrotullues, dhe kështu mund të përcaktohet nëse secila rrotë ka udhëtuar një "njësi" përpara ose mbrapa përgjatë dyshemesë. Kjo njësi është raporti i perimetrit të rrotës me rezolucionin e enkoderit. Nëse rrota e majtë do të lëvizte përpara një njësi ndërsa rrota e djathtë qëndron e palëvizshme, atëherë rrota e djathtë vepron si një strumbullar dhe rrota e majtë gjurmon një hark rrethor në drejtim të akrepave të orës. Meqenëse njësia e distancës së dikujt është zakonisht mjaft e vogël, mund të përafrohet duke supozuar se ky hark është një vijë. Kështu, pozicioni origjinal i rrotës së majtë, pozicioni përfundimtar i rrotës së majtë dhe pozicioni i rrotës së djathtë formojnë një trekëndësh, të cilin mund ta quajmë A. Gjithashtu, pozicioni origjinal i qendrës, pozicioni përfundimtar i qendrës dhe pozicioni i rrotës së djathtë formojnë një trekëndësh të cilin mund ta quajmë B. Meqenëse qendra e robotit është e barabartë me secilën rrotë, dhe pasi ato ndajnë këndin të formuar në timonin e djathtë, trekëndëshat A dhe B janë trekëndësha të ngjashëm. Në këtë situatë, madhësia e ndryshimit të pozicionit të qendrës së robotit është gjysma e një njësie. Këndi i këtij ndryshimi mund të përcaktohet duke përdorur ligjin e sinusit. Parimisht, kjo situatë është ilustruar në figurën më poshtë.

Nëse shpejtësia e rrotave janë v_1 dhe v_2 respektivisht dhe distanca mes tyre është $2d$, atëherë do vlejë:

$$\omega(R + d) = v_1$$

$$\omega(R - d) = v_2$$

Duke rregulluar ekuacionet e mësipërme do fitojmë:

$$\omega = \frac{v_1 - v_2}{2d}$$

$$R = d\frac{v_2 + v_1}{v_1 - v_2}$$

Shpejtësia momentale e pikës së mesme (mes rrotave të robotit, apo qendrës së robotit) do jetë:

$$V = \omega R$$

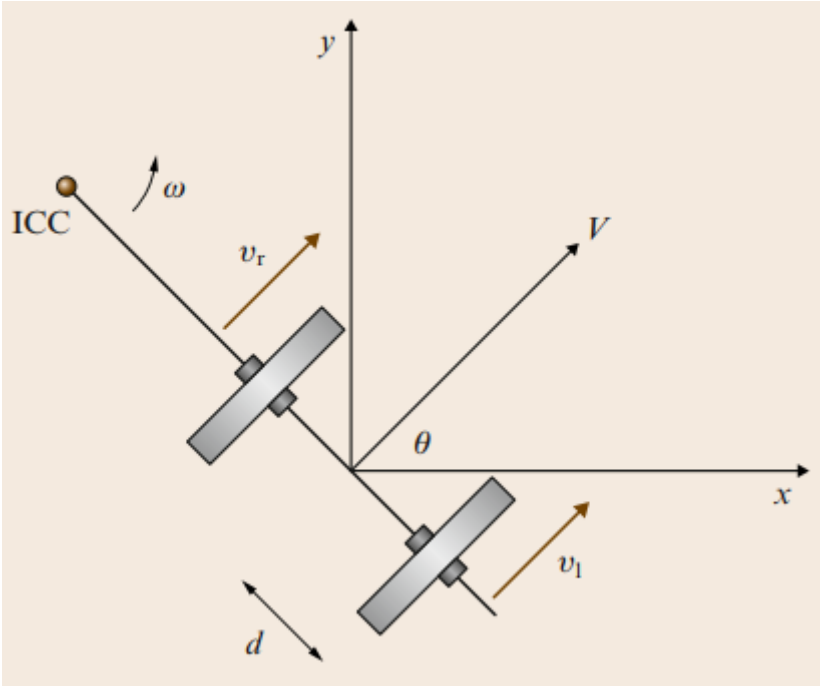

Figura 32: Parimi bazik i odometrisë për lokalizim [1]

Nga ekuacionet e mësipërme, me pak veprime trigonometrike dhe po ashtu duke përdorur teoremën e sinusit, mund të nxjerrim informatës mbi pozitën e robotit dhe po ashtu orientimin e tij. Kjo përshkruhet me ekuacionet vijuese:

$$x(t) = \int V(t)\cos\big(\theta(t)\big)\,dt$$

$$y(t) = \int V(t)\sin\big(\theta(t)\big)dt$$

$$\theta(t) = \int \omega(t)dt$$

Duke pasur parasysh një model të tillë dhe njohuri të plotë të inputeve të kontrollit, në parim, ne duhet të jemi në gjendje të vlerësojmë pozicionin e një roboti në çdo kohë. Në një botë të përsosur kjo do të ishte gjithçka që është e nevojshme për të vlerësuar me saktësi pozën e robotit në çdo kohë në të ardhmen. Fatkeqësisht gabime në modelim (vlerësime të pasakta të madhësisë së rrotave, madhësisë së automjetit), pasiguri në lidhje me inputet e kontrollit, joidealitetet e kontrolluesit të motorit (gabimet midis rrotullimit të komanduar të rrotës dhe rrotullimit të vërtetë), gabime në modelimin fizik të robotit (ngjeshja e rrotave, ngjeshja e tokës, rrëshqitja e rrotave, gjerësia jozero e gomave), etj., sjellin një gabim midis vlerësimit odometrik të lëvizjes së automjetit dhe lëvizjes së tij të vërtetë. Problemi i korrigjimit të këtij gabimi është problemi i mirëmbajtjes së pozicionit për automjetin dhe kërkon integrimin e vlerësimit të llogaritjes së vdekur me vlerësimet e marra nga sistemet e tjera të sensorëve

5.4 Odometria në Turtlebot 2i.

Nga diskutimi i mësipërm mund të inferojmë që parakusht për një odometri të suksesshme është pajisja me sistemin sensorik përkatës për lexim të zhvendosjeve, atyre lineare dhe këndore. Turtlebot 2i këtë e bën përmes enkoderëve inkremental që janë të vendosur në motorët që bëjnë rrotullimin e rrotave dhe po ashtu përmes senorëve inercial (xhiroskopit dhe akselerometrit).

Odometria përdoret nga TurtleBot për të vlerësuar pozicionin dhe orientimin e tij në lidhje me vendndodhjen e tij fillestare të dhënë në termat e një pozicioni x dhe y dhe një orientimi rreth boshtit z (lart) ndërsa TurtleBot lëviz. Të dhënat e odometrisë për të përcaktuar pozicionin dhe orientimin mund të bëhen shumë të pasakta ndërsa TurtleBot lëviz në një distancë të gjatë.Pasaktësia mund të jetë për shkak të gabimeve në parametrat e robotit, si p.sh. diametrat e gabuar të rrotave të përdorura në llogaritjen e distancës ose për shkak të sipërfaqeve të pabarabarta të drejtimit që bëjnë që koduesit e rrotave të gjenerojnë impulse të rreme.

Një formë e thjeshtë për të lexuar të dhënat odometrike të Turtlebotit është duke komunikuar me të përmes Matlabit. Vendosja e komunikimit mes Matlab dhe turtlebot bëhet me instruksionet në vijim.

```
ipaddress = '192.168.1.194'; % IP addressa e Turtlebotit
tbot = turtlebot(ipaddress,11311); % Krijimi i objektit turtlebot
tbot.Velocity.TopicName = '/cmd_vel'; % Leximi i te dhenave
```

Më pas ekzekutojmë instruksionin që kërkon të dhënat odometrike:

```
odom = getOdometry(tbot)
```

Përgjigja e turtlebotit do jetë si ajo në vijim.

```
odom = struct with fields:
       Position: [4.3772 1.4580 -0.0010]
    Orientation: [-2.2936 0.0032 7.8259e-06]
```

Shohim se përgjigja e robotit është saktësuar me anë të dy vektorëve tredimensional. Vektori i parë tregon pozitën (x,y,z) kurse vektori i dytë tregon orientimin e robotit. Të theksojmë se lokalizimi përmes odometrisë është informatë relative. Pra është informatë në raport më pikën ndaj të cilës ka filluar leximi i sensorëve. Në menyrë programore, përmes Matlabit mund ta bëjmë përcaktimit e pikës fillestare duke resetuar të dhënat ekzistuese odometrike. Kjo bëhet përmes instruksionin si në vijim.

```
resetOdometry(tbot)
```

Nëse dëshirojmë që të dhënat nga sensorët t'i tregojmë si trajektore e lëvizjes së robotit, këtë mund ta bëjmë më kodin në vijim.

```
odomList = zeros(20,2);
resetOdometry(tbot)

for i = 1:20
    odom = getOdometry(tbot);
    odomList(i,:) = [odom.Position(1) odom.Position(2)];

    if i < 10
        setVelocity(tbot,0.25,0.15)
    else
        setVelocity(tbot,0.25,-0.15)
    end
    pause(1);
end

plot(odomList(:,1),odomList(:,2))
```

Në kodin më lartë kemi disa instruksione të reja. Instruksioni

```
odom.Position(1)
```

lexon vetëm parametrin e parë nga vektori i pozitës, që është koordinata x në raport me pozitën fillestare. Kurse instruksioni

```
odom.Position(2)
```

lexon parametrin e dytë nga vektori i pozitës, që është koordinata y në raport me pozitën fillestare. Instruksioni

```
setVelocity(tbot,0.25,0.15)
```

Bën lëvizjen e robotit me shpejtësi $0.25\ m/s$ në rrotën e majtë, kurse me shpejtësi $0.15 m/s$ në rrotën e djathtë. Trajektorja nëpër të cilën roboti ka lëvizur, për këtë set e veprimeve, është treguar në figurën më poshtë.

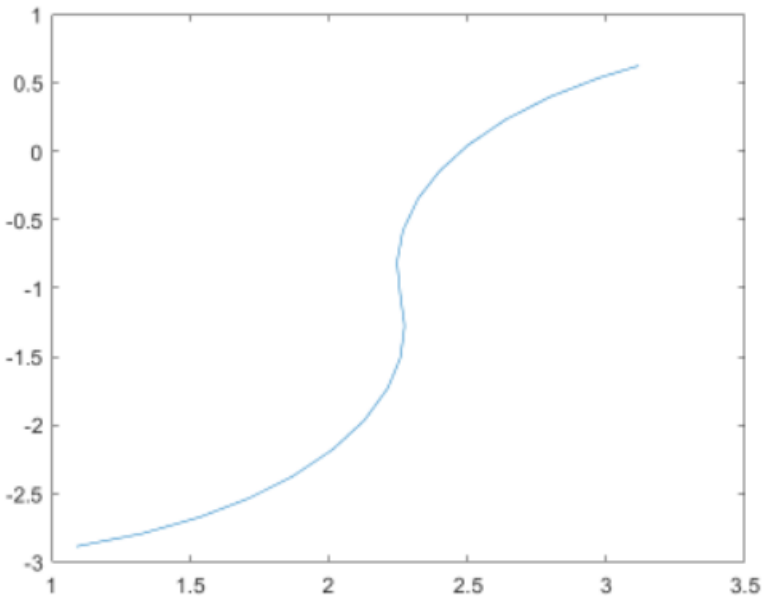

Figura 33: Trajektorja e Turtlebot [6]

5.5 Algoritmi SLAM

Lokalizimi dhe hartëzimi i njëkohshëm (SLAM) është problemi llogaritës i ndërtimit ose përditësimit të një harte të një mjedisi të panjohur duke mbajtur njëkohësisht gjurmët e vendndodhjes së një roboti brenda tij. Termi SLAM është siç thuhet një akronim për Lokalizimin dhe Hartëzimin e Njëkohshëm. Fillimisht u zhvillua nga Hugh Durrant-Whyte dhe John J. Leonard bazuar në punën e mëparshme nga Smith, Self dhe Cheeseman. Durrant-Whyte dhe Leonard fillimisht e quajtën atë SMAL por më vonë u ndryshua për të dhënë një ndikim më të mirë. SLAM ka të bëjë me problemin e ndërtimit të një harte të një mjedisi të panjohur nga një robot mobil, ndërsa në të njëjtën kohë navigon mjedisin duke përdorur hartën.

SLAM përbëhet nga pjesë të shumta; Nxjerrja e pikës referimi, asocimi i të dhënave, vlerësimi i gjendjes, përditësimi i gjendjes dhe përditësimi i pikës referimi. Ka shumë mënyra për të zgjidhur secilën nga pjesët e vogla. SLAM është i zbatueshëm për lëvizjen 2D dhe 3D.

Algoritmi SLAM përbëhet nga një numër hapash. Qëllimi i algoritmit është përdorimi i mjedisit për të përditësuar pozicionin e robotit. Meqenëse odometria e robotit (që i jep pozicionin e robotit) është shpesh e gabuar, ne nuk mund të mbështetemi drejtpërdrejt në odometrinë. Ne mund të përdorim skanime laser të mjedisit për të korrigjuar pozicionin e robotit. Kjo arrihet duke nxjerrë veçori nga mjedisi dhe duke ri-vëzhguar kur roboti lëviz përreth. Një EKF (Extended Kalman Filter) është zemra e procesit SLAM. Ai është përgjegjës për përditësimin se ku roboti mendon se bazohet në këto veçori. Këto veçori zakonisht quhen pika referimi dhe do të shpjegohen së bashku me EKF-në në dy kapitujt e ardhshëm. EKF mban gjurmët e një vlerësimi të pasigurisë në pozicionin e robotëve dhe gjithashtu pasigurinë në këto pika referimi që ka parë në mjedis. Skematikisht, kjo logjikë është treguar në figurën më poshtë.

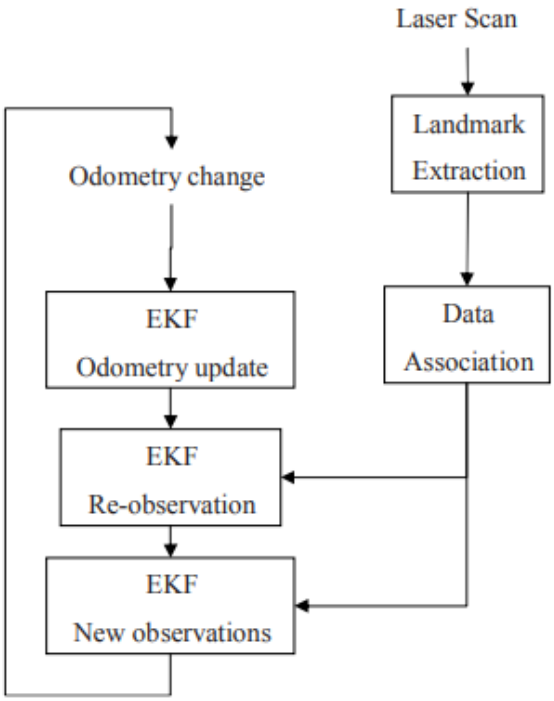

Figura 34: SLAM Procesi [4]

Kur odometria ndryshon për shkak se roboti lëviz, pasiguria në lidhje me pozicionin e ri të robotit përditësohet në EKF duke përdorur përditësimin e odometrisë. Pikat e referimit më pas nxirren nga mjedisi nga pozicioni i ri i robotëve. Roboti më pas përpiqet t'i asociojë këto pika referimi me vëzhgimet e pikave karakteristike (ang. Landmark) që ka parë më parë. Pikat referuese të ri-vëzhguara më pas përdoren për të përditësuar pozicionin e robotëve në EKF. Pikat e referimit që nuk janë parë më parë i shtohen EKF-së si vëzhgime të reja në mënyrë që ato të mund të ri-vëzhgohen më vonë. Duhet të theksohet se në çdo moment në këto hapa EKF do të ketë një vlerësim të pozicionit aktual të robotëve. Ky proces tregohet më lehtë duke vizualizuar ecurinë e tij. Po marrim shembull figurën 35. Roboti është markuar me trekëndësh. Në hapin fillestar roboti bën identifikim e pikave karakteristike (ang. landmarks) duke përdorur sistemin e sensorëve. Kjo mund të jetë një set i sensorëve ultrasonik, lidar 2D apo 3D, apo edhe kamerë me thellësi. Pasi të jenë identifikuar disa pika të tilla, roboti lëvizë (natyrisht duke evituar pengesat) në një mesatare lokale, dhe bën llogaritjen e pozitën së tij duke përdorur odometrinë. Kjo situatë është treguar në figurën 36.

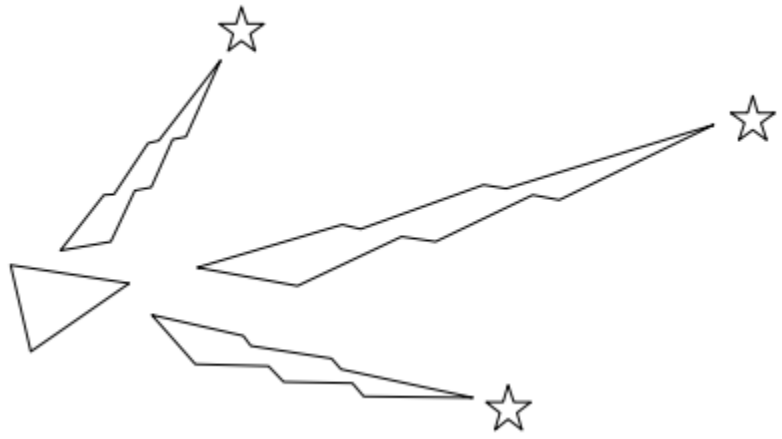

Figura 35: Hapi fillestarë në ciklin e SLAM procesit. Identifikmi i landmarks. [2]

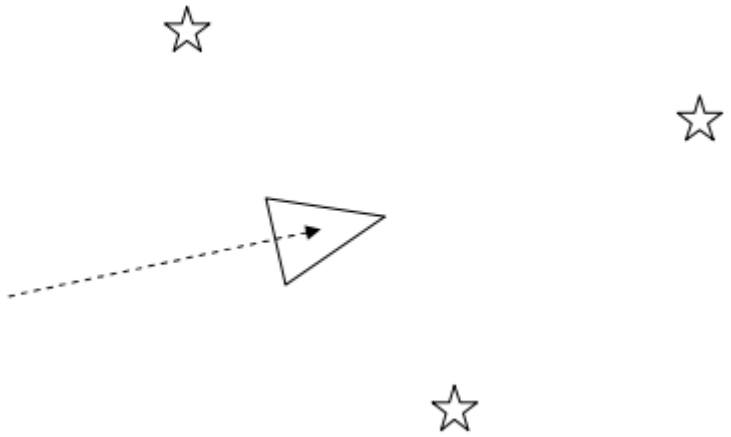

Figura 36: Pozita e robotit pas lëvizjes fillestare [2]

Pasi roboti të ketë lëvizur, pozitën aktuale në të cilën mendon se ndodhet (nga të dhënat odometrike), ai bën edhe një herë matjen e distancës ndaj pikave karakteristike. Nëse këto dy situata përputhen, atëherë roboti vazhdon lëvizjen. Nëse kemi mospërputhje mes pozitës nga odometria dhe mes matjes nga sensorët, atëherë roboti duhet të merr vendim. Natyrisht, roboti është më i prirur që më shumë të ju besoj të dhënave që i pranon nga leximet e sensorëve mbi pozitën relative ndaj pikave karakteristike sesa të dhënave odometrike. Një mospërputhje e tillë është ilustruar në figurën 37. Trekëndëshi më pika është pozita e robotit bazuar në matjes ndaj pikave referente, kurse ajo me vija të ndërprera është pozita sipas të dhënave odometrike. Pra asnjëra nga këto në fakt nuk janë informata mjaftueshëm të sakta mbi pozitën aktuale të robotit. Mënyra se si roboti nxjerr informatën për pozitën aktuale të tij, mbi të cilën informatë më pas edhe operon, bazohen në një estimim të pozitës kur për bazë merr edhe matjet ndaj pikave karakteristike por edhe të dhënat odometrike. Kjo gjeneron një estimim shumë të besueshëm mbi pozitën aktuale të robotit. Kjo është treguar në figurën 38.

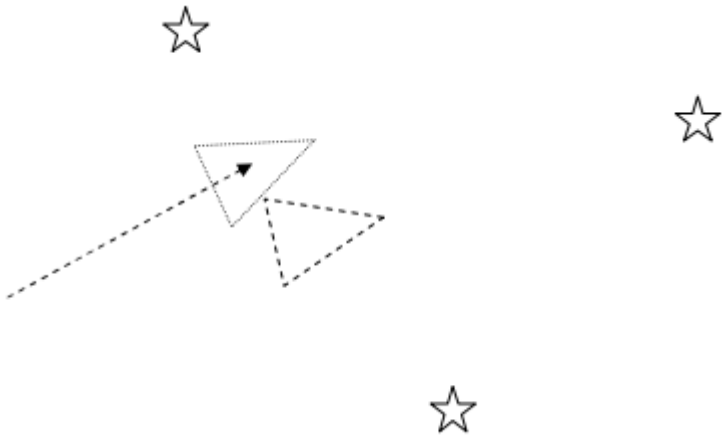

Figura 37: Pozita e menduar dhe ajo aktuale e robotit

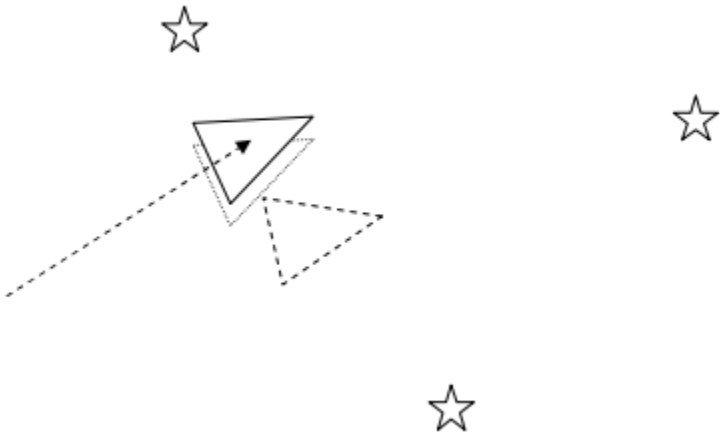

Figura 38: Pozita reale e robotit bazuar në estimimet mbi odometri dhe pika karakteristike [2]

Në fakt roboti është këtu. Sensorët nuk janë të përsosur, kështu që roboti nuk do ta dijë saktësisht se ku ndodhet. Megjithatë, ky vlerësim është më i mirë sesa të mbështetesh vetëm në odometrinë. Trekëndëshi me pika përfaqëson vendin ku mendon se është; trekëndëshi i ndërprerë ku odometria e tregonte se ishte; dhe trekëndëshi i fundit ku është në të vërtetë.

5.5.1 Mapimi laserik/vizual

Hapi i parë në procesin SLAM është marrja e të dhënave rreth rrethinës së robotit. Kjo mund të bëhet me një sërë mekanizmash që janë në dispozicion. Kjo mund të jetë një lidarë 2D apo 3D, nj137 set sensorësh ultrasonik apo një kamerë me thellësi. Në rastin tonë, turtlebot 2i është i pajisur me kamerën Intel RealSense 3D Camera SR300 të cilën e përdorë për të fituar informatën mbi distancat e objekteve që ndodhën përreth tij.

5.5.2 Të dhënat odometrike

Një aspekt i rëndësishëm i SLAM-it janë të dhënat e odometrisë. Qëllimi i të dhënave të odometrisë është të sigurojë një pozicion të përafërt të robotit të matur nga lëvizja e rrotave të robotit, për të shërbyer si hamendësimi fillestar se ku mund të jetë roboti në EKF. Marrja e të dhënave të odometrisë nga turtle është mjaft e lehtë duke përdorur serverin e integruar. Mund të dërgoni vetëm një varg teksti në serverin telnet në një port të caktuar dhe serveri do të kthejë përgjigjen. Pjesa e vështirë në lidhje me të dhënat e odometrisë dhe të dhënave laser është të përcaktosh kohën e duhur. Të dhënat e kamerës (apo laserit) në një kohë t do të jenë të vjetruara nëse të dhënat e odometrisë merren më vonë. Për t'u siguruar që ato janë të vlefshme në të njëjtën kohë, mund të ekstrapolohen të dhënat. Është më e lehtë për të ekstrapoluar të dhënat e odometrisë pasi kontrollet janë të njohura. Mund të jetë vërtet e vështirë të parashikosh se si do të jenë matjet e skanerit me laser. Nëse dikush ka kontroll se kur kthehen matjet, është më e lehtë të kërkosh njëkohësisht vlerat e skanerit laser dhe të dhënat e odometrisë.

5.5.3 Pikat karakteristike (ang. landmarks)

Pikat karakteristike (e referimit) janë veçori që lehtë mund të ri-vëzhgohen dhe dallohen nga mjedisi. Këto përdoren nga roboti për të gjetur se ku është (për të lokalizuar veten). Një mënyrë për të imagjinuar se si funksionon kjo për robotin është ta imagjinoni veten me sy të lidhur. Nëse lëvizni me sy të lidhur në një shtëpi, mund të shtriheni dhe të prekni objekte ose të përqafoni muret në mënyrë që të mos humbni. Gjërat karakteristike të tilla si ato që ndjehen duke prekur një kornizë dere mund t'ju ndihmojnë në krijimin e një vlerësimi se ku jeni. Kamerat me thellësi, sonarët dhe skanerët laser janë një ndjenjë prekjeje e robotëve.

Pikat e referimit duhet të ri-vëzhgohen duke i lejuar për shembull të shikohen (zbulohen) nga pozicione të ndryshme dhe si rrjedhim nga këndvështrime të ndryshme. Pikat e referimit duhet të jenë mjaft unike në mënyrë që ato të mund të identifikohen lehtësisht nga një hap në tjetrin pa i përzier ato. Me fjalë të tjera, nëse ri-vëzhgoni dy pika referimi në një moment të mëvonshëm kohor, duhet të jetë e lehtë të përcaktohet se cila nga pikat referuese është cila nga pikat referuese që kemi parë më parë. Nëse dy pika referimi janë shumë afër njëra-tjetrës, kjo mund të jetë e vështirë. Nëse vendosni që diçka të jetë një pikë referimi, ajo duhet të jetë e palëvizshme. Përdorimi i një personi si pikë referimi është një ide e keqe. Arsyeja për këtë kriter është mjaft e qartë. Nëse pikë referimi nuk është gjithmonë në të njëjtin vend, si mund ta dijë roboti duke pasur parasysh këtë pikë referimi se në cilin vend është.

Pikat kryesore në lidhje me pikat referente janë si më poshtë:

- Pikat e referimit duhet të jenë lehtësisht të ri-vëzhgueshme.
- Pikat e veçanta të referimit duhet të jenë të dallueshme nga njëra-tjetra.
- Pikat e referimit duhet të jenë të shumta në mjedis.
- Pikat e referimit duhet të jenë të palëvizshme.

5.6 Implementimi i SLAM në ROS

Për zbatimin autonom të navigimit mund të përdoret algoritmi SLAM. SLAM përdoret për të vlerësuar pozicionin e robotit duke e lëvizur atë përgjatë zonës së panjohur. Sa më shumë të jetë numri i përpjekjeve për të eksploruar zonën e panjohur, aq më shumë është cilësia e hartës së krijuar. Algoritmi SLAM

zbatohet duke përdorur GMapping Tool. GMapping Tool kërkon të dhëna odometrike (të dhënat e koduesit nga rrotat) dhe të dhëna laser (të dhëna kinect). Slam gmapping përdoret për të krijuar hartën e rrjetit të banimit 2-D, e cila është si një plan urbanistik i një ndërtese.

Zhvillimi i softuerit robotik bëhet në ROS (Robot Operating System). ROS ofron shërbime si kalimi i mesazheve ndërmjet proceseve dhe paketës, kontrolli i pajisjes në nivel të ulët, zbatimi i funksioneve të përbashkëta. Në ROS procesi tregohet në format grafik dhe përpunimi ndodh në nyje. Të gjitha simulimet komplekse 3D janë bërë në GAZEBO. Modelet, GZ Sever, GZ Client dhe GZ Web janë përbërësit kryesorë të tij. GAZEBO ka veçori si modele robotësh të integruar, simulim cloud, simulim dinamike, grafikë të avancuar 3D, mjete të linjës së komandës dhe transport TCP IP. Me ndihmën e GAZEBO ne mund të konceptojmë Inercinë, Forcat dhe Informacionin e Sensorit. Gazebo është një imitues dinamik 3D i përdorur për simulimin e modeleve komplekse robotike në mjedise të brendshme dhe të jashtme. Ai siguron aftësinë për të testuar modelin robotik në mjedisin e simuluar dhe për të inkorporuar të dhënat nga sensorët. Gazebo lejon testimin e performancës së modelit robotik në kushte ekstreme pa dëmtuar harduerin. Përdor një motor fizik për ndriçim, inerci, gravitet, etj. Një skedar XML i quajtur (URDF) Universal Robotic Description Format përdoret për të përshkruar elementë të ndryshëm të robotit. Figura 39 tregon modelin 3D të Turtlebotit të simuluar në gazebo. Modeli robotik është testuar në mjedisin e paraqitur në figurën 40.

Algoritmi SLAM zbatohet në aplikacione të ndryshme si makina vetëdrejtuese, monitorimi i shkëmbinjve nënujorë, eksplorimet e minierave dhe avionët pa pilot. Algoritmet SLAM zbatohen duke përdorur një ROS me burim të hapur.

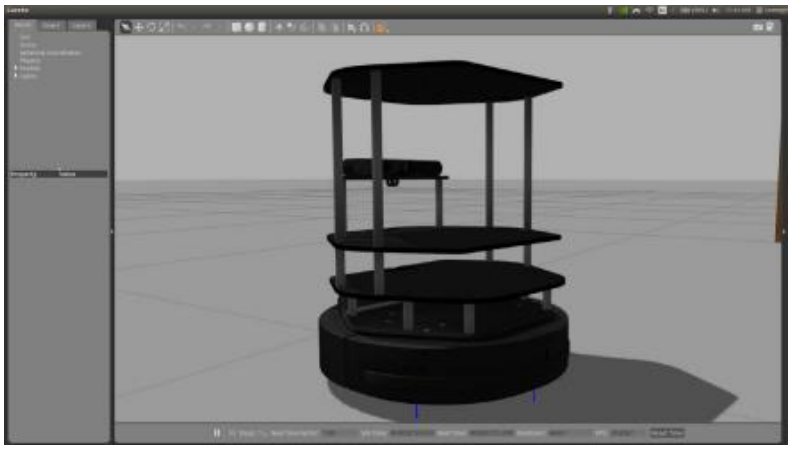

Figura 39: Modeli i Turtlebot 2i në Gazebo

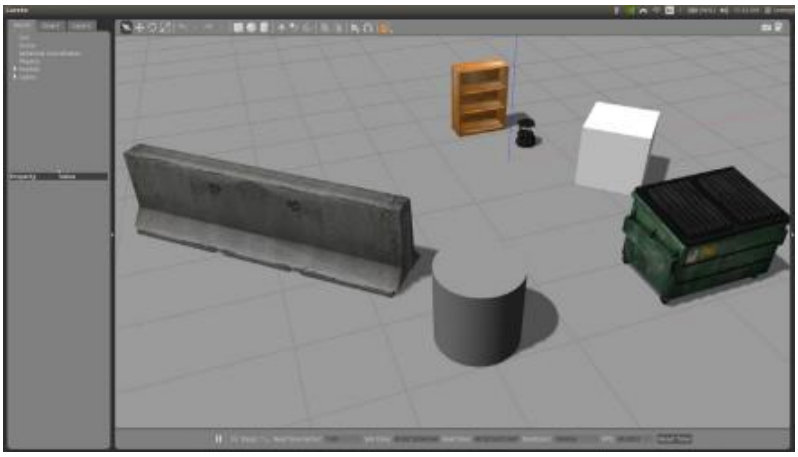

Figura 40: Mjedisi i krijuar në Gazebo për testim

Gjenerimi i hartës kryhet duke gjeneruar fillimisht mjedisin dhe duke importuar turtlebotin në Gazebo. Modeli i robotit përbëhet nga dy rrota dhe një sensor kinect është montuar brenda kornizës. Procesi i hartës në këtë punim zbatohet duke përdorur paketën gmapping. Për të përdorur gmapping modeli i robotit duhet të ofrojë të dhëna odometrike dhe duhet të jetë i pajisur me një zbulues rreze laser të montuar horizontalisht, fiks. Kinect kap imazhin e thellësisë së mjedisit duke përdorur sensorë infra të kuqe dhe vepron si një pajisje për gjetjen e rrezes. Duhen 16 bit për pixel të dhëna infra të kuqe me një rezolucion prej 640 x 480 si një format imazhi me ngjyra. Ky imazh i thellësisë konvertohet në një re me pika 3d duke përdorur paketën proc të imazhit të thellësisë në ROS. Më tej, këto të dhëna reje me pika 3D konvertohen në skanim laser 2D duke përdorur paketën pointcloud në paketën laser. Duke ofruar të gjithë parametrat e kërkuar për paketat në gmapping krijohet një hartë në rviz. Figura 41 tregon hartën e gjeneruar. Fillimisht roboti zhvendoset në mjedisin e simuluar duke përdorur paketën e çelësit teleop në ROS duke dërguar komanda të shpejtësisë nëpërmjet tastierës. Figura 42 tregon grafikun rqt të gmapping. Paketa gmapping ofron temën /map e cila mund të përdoret nga paketa e serverit të hartave për të ruajtur hartën e krijuar. Harta tani është gati për t'u përdorur për navigim autonom.

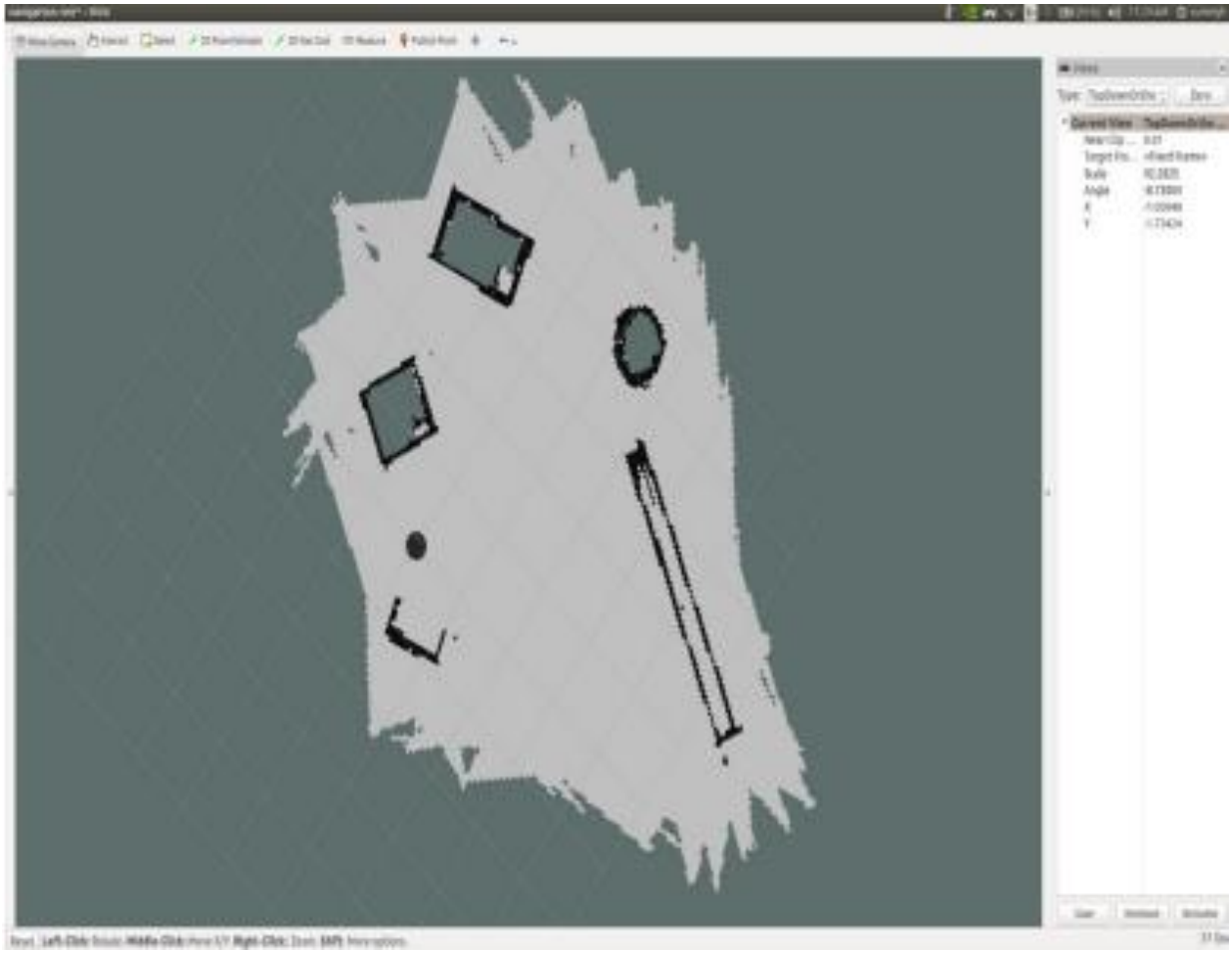

Figura 41: Harta e gjeneruar në Rviz

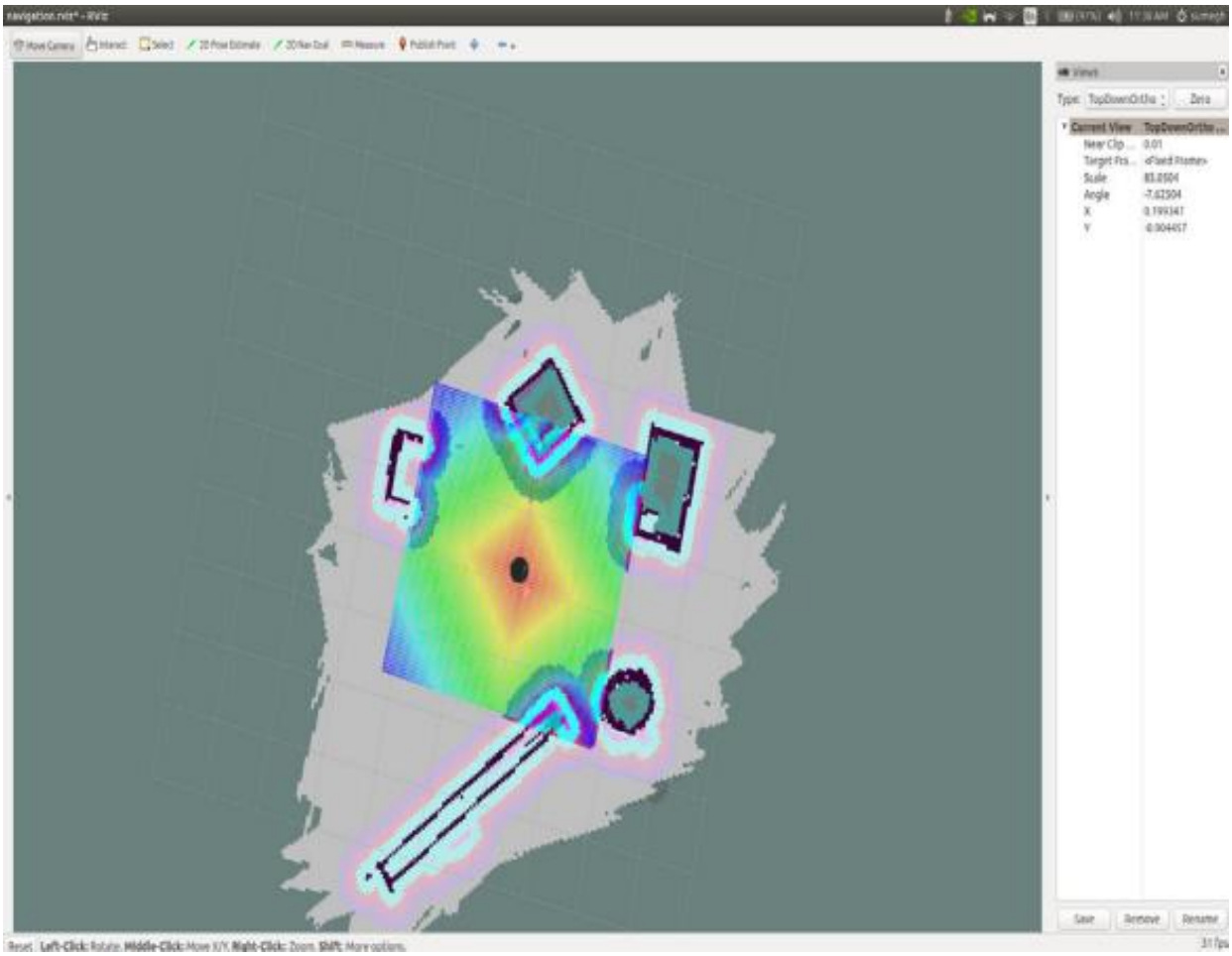

Figura 42: Lokalizimi i robotit në Rviz përmes SLAM

Navigimi autonom mund të kryhet pasi të përfundojë hartëzimi dhe lokalizimi. Paketat e stivës së navigimit ROS përdoren për të zbatuar AMCL. Nyja për të kryer lokalizimin në një hartë statike sigurohet nga paketa ROS AMCL. Kjo nyje pajtohet me të dhënat TF, 2D Të dhënat e skanimit me laser të ofruara nga roboti dhe harta statike e krijuar më parë. Poza e robotit dhe pozicioni i tij i vlerësuar në lidhje me hartën publikohet nga nyja AMCL. Informacioni rreth pengesave në botën e simuluar ruhet në dy harta kostosh. Harta globale e kostos përdoret për planifikimin e rrugës globale, ajo bën një plan afatgjatë të rrugës mbi hartë. Përdoret harta e kostos lokale për shmangien e pengesave dhe planifikimin lokal të rrugës. Rviz vizualizon hartën e kostos globale dhe lokale. Roboti tani mund të zhvendoset në mënyrë autonome. Qëllimi i navigimit 2D në Rviz përdoret për t'i caktuar robotit në hartë një objektiv destinacioni së bashku me orientimin e tij. Rruga drejt destinacionit të dëshiruar planifikohet nga roboti dhe komandat e shpejtësisë i jepen kontrollues robot. Figura 43 tregon shtegun e ndjekur nga roboti për të lundruar nga pozicioni i tij fillestar në qëllimin e dëshiruar të destinacionit të caktuar.

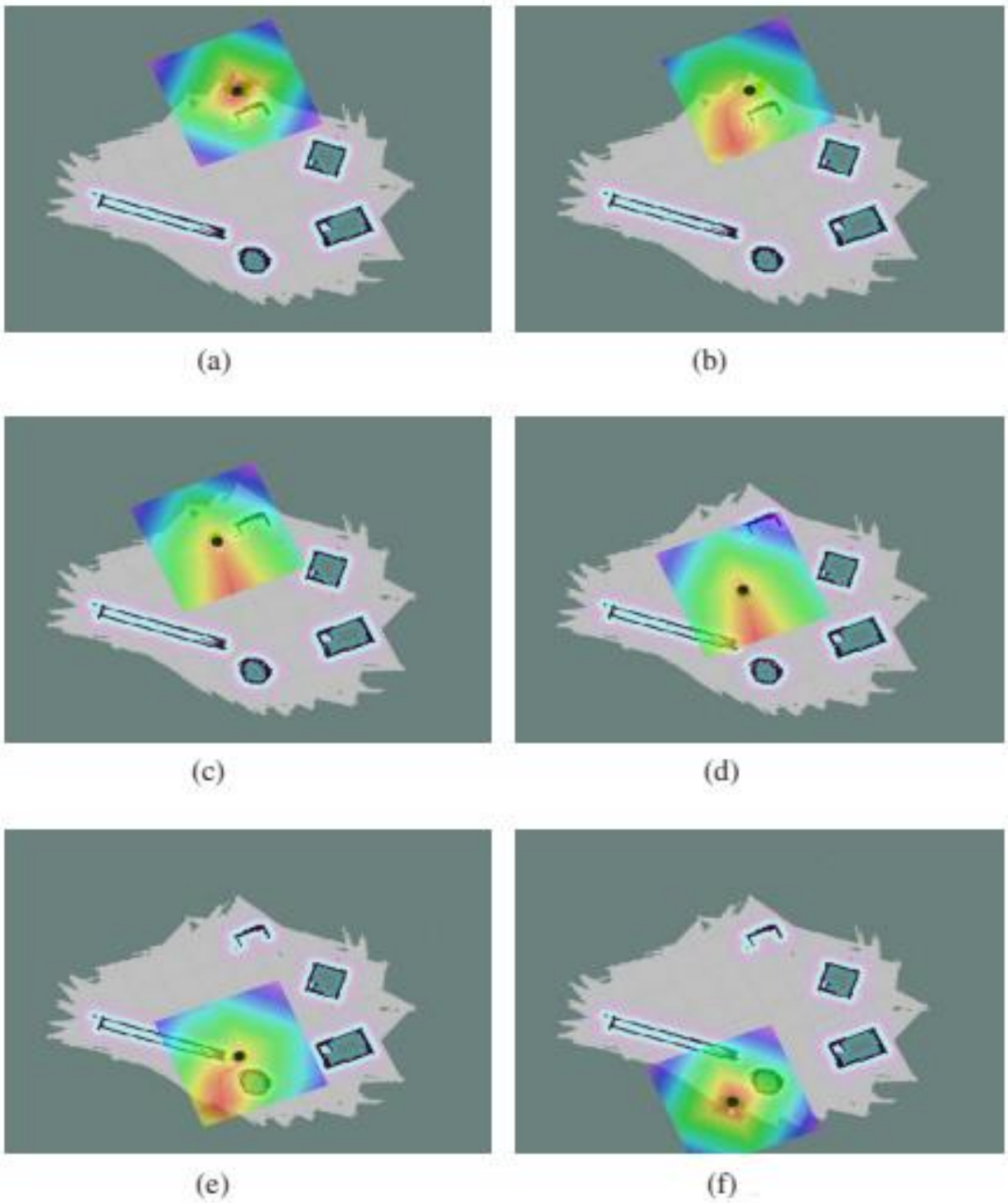

Figura 43: Trajektorja e lëvizjes autonome te robotit pasi të jetë përfunduar lokalizimi dhe hartëzimi me SLAM

6. Turtlebot dhe Matlab

MATLAB® është një platformë programimi e krijuar posaçërisht për inxhinierët dhe shkencëtarët për të analizuar dhe dizajnuar sisteme dhe produkte që transformojnë botën tonë. Zemra e MATLAB është gjuha MATLAB, një gjuhë e bazuar në matricë që lejon shprehjen më të natyrshme të matematikës llogaritëse. Miliona inxhinierë dhe shkencëtarë në mbarë botën përdorin MATLAB për një sërë aplikacionesh, në industri dhe akademi, duke përfshirë mësimin e thellë dhe mësimin e makinerive, përpunimin e sinjalit dhe komunikimin, përpunimin e imazheve dhe videove, sistemet e kontrollit, testimin dhe matjen, financat llogaritëse dhe biologjinë llogaritëse.

Në vijim do të shohim se si mund të bëjmë ndërlidhjen mes Matlab dhe Turtlebot, duke pasur kështu një kanal komunikimi mes këtyre dy komponentëve që vepron në kohë reale.

6.1 Shembull lidhjeje mes Matlab dhe Turtlebot 2i

Ky shembull tregon se si të krijoni një nyje ROS të bazuar në MATLAB® duke përdorur Paketën ROS I/O për të operuar një robot të bazuar në ROS që është i lidhur nga jashtë. Për qëllime demonstrimi, ne do të përdorim TurtleBot Simulator (linku http://wiki.ros.org/turtlebot_simulator?distro=hydro për më shumë informacion) si robotin tonë dhe një ndërfaqe përdoruesi të bazuar në MATLAB si mjet për të kontrolluar robotin. Prandaj, Demoja e Kontrolluesit TurtleBot përbëhet nga dy komponentët kryesorë të mëposhtëm:

Një robot simulator që funksionon në Gazebo në një platformë Ubuntu®.

Një nyje kontrolluesi robotësh që funksionon në MATLAB në kompjuterin tuaj pritës.

Në këtë shembull, ndërfaqja e mëposhtme e përdoruesit është projektuar me MATLAB GUIDE për të nisur një nyje ROS në MATLAB me një seri botuesish dhe pajtimtarësh që komunikojnë me TurtleBot Simulator.

Së pari duhet të lëshojmë ROS dhe Simulatorin TurtleBot nga makina virtuale Ubuntu përpara se të nisim nyjën TurtleBot Controller nga MATLAB. Duke supozuar se nuk e keni mbyllur makinën virtuale, le të kthehemi në desktopin e makinës virtuale. Klikoni shkurtoren Gazebo Playground në desktopin e makinës virtuale për të nisur ROS dhe Gazebo. TurtleBot Simulator duhet të fillojë automatikisht me një "shesh lojërash" si "botën" e tij. Nëse shihni një "botë" të zbrazët, minimizoni Gazebo dhe më pas kthejeni për ta parë. Kjo duhet të detyrojë një rifreskim të Gazebo. Pasi të ngarkohet TurtleBot Simulator, duhet të paraqitet skena e mëposhtme në Gazebo.

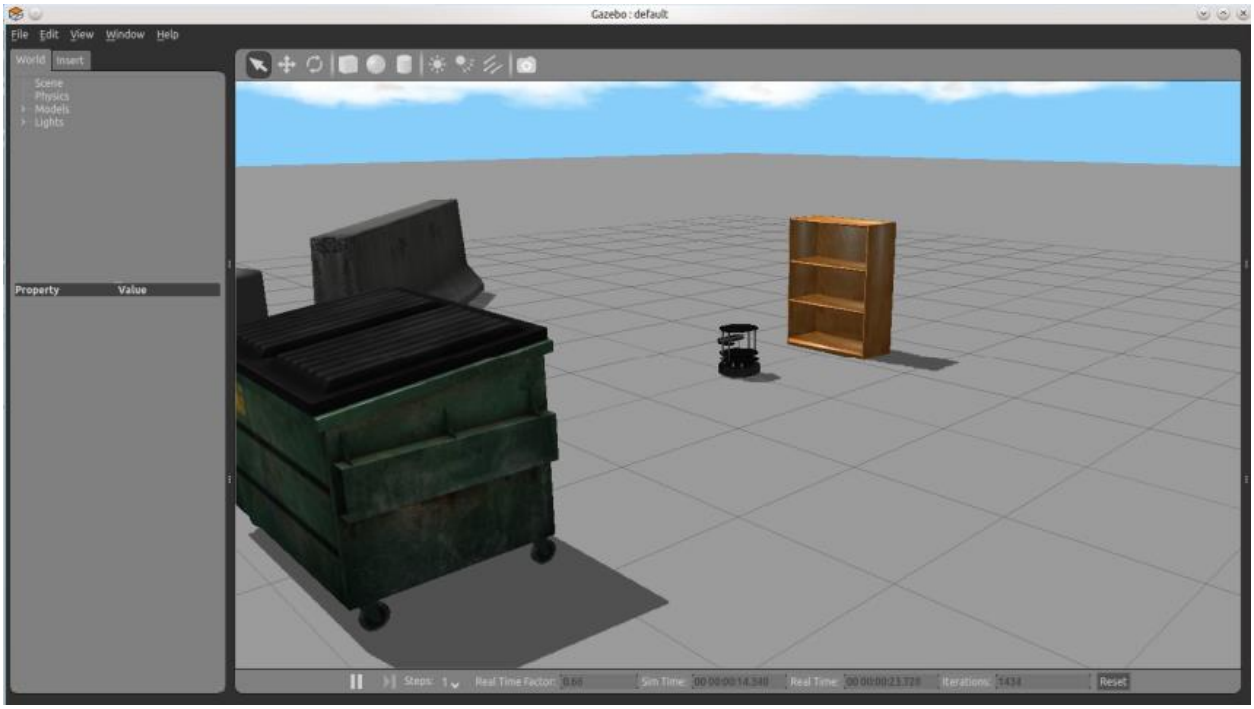

Figura 44: Bota e krijuar në Gazebo për testim

Tani jemi gati të nisim nyjën TurtleBot Controller. Kthehuni në desktopin MATLAB. Duke supozuar se dosja TurtlebotDemo është tashmë në rrugë, futni turtlebot_startup për të hapur ndërfaqen e përdoruesit për Kontrolluesin TurtleBot. Në figurën e mëposhtme është treguar interface (ndërfaqe) që i paraqitet përdoruesit nëse nyja TurtleBot Controller lidhet me sukses me masterin ROS që po funksionon në makinën virtuale Ubuntu.

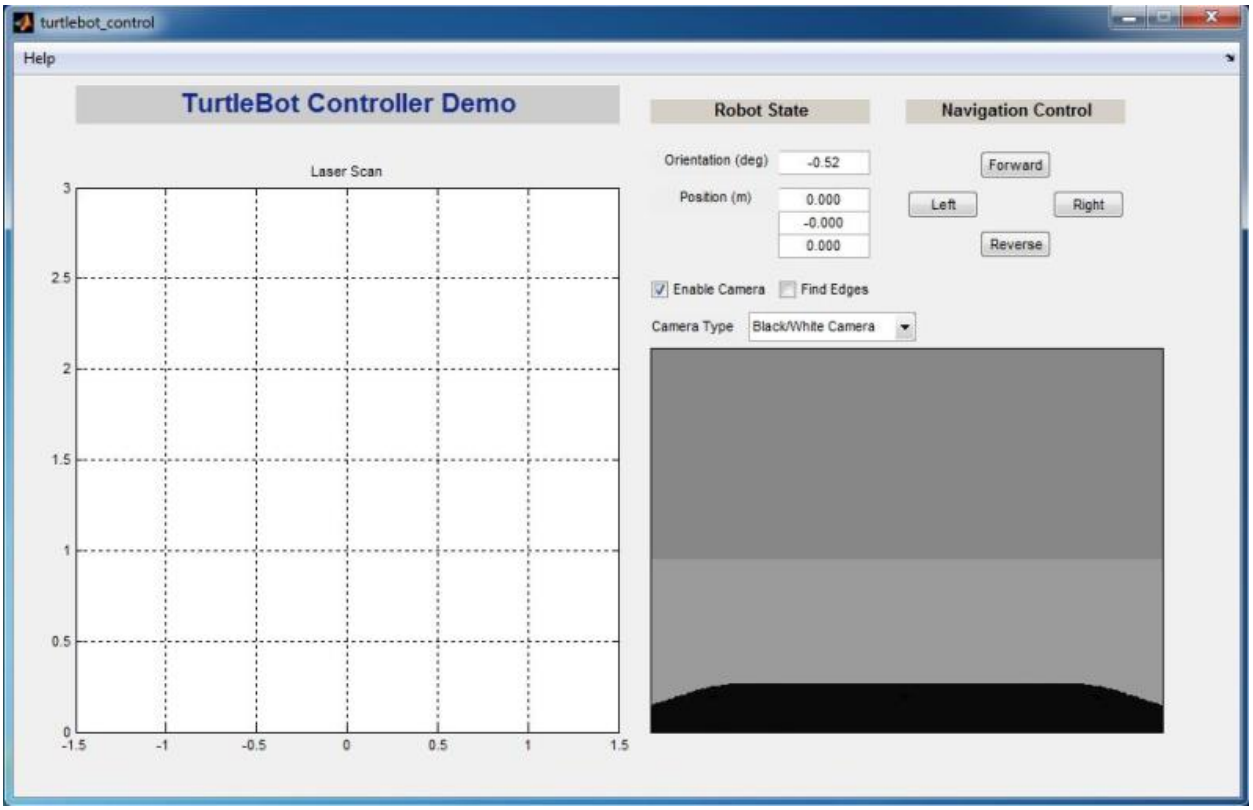

Figura 45: Interface grafik pas lidhjes swë suksesshme me Turtlebot

Interface e përdoruesit TurtleBot Controller përbëhet nga komponentët e mëposhtëm:

- Kontrolli i navigimit - Përdor një botues për të dërguar komanda të shpejtësisë te TurtleBot Simulator për të lëvizur robotin. Botuesi dërgon mesazhe geometry_msgs/Twist te një temë e quajtur /mobile_base/commands/velocity.
- Gjendja e robotit - Përdor dy abonentë për të shfaqur të dhënat e odometrisë dhe IMU të robotit. Abonentët marrin mesazhe nav_msgs/Odometry dhe sensor_msgs/Imu nëpërmjet temave të emërtuara/odom dhe /mobile_base/sensors/imu_data, respektivisht.
- Imazhi i kamerës - Përdor një nga dy abonentët për të shfaqur imazhin e kapur nga kamera në robot. Të dy abonentët marrin mesazhe sensor_msgs/Image nëpërmjet një teme të quajtur/camera/rgb/image_raw. Një menu kërcyese mbi ekran përdoret për të ndërruar mes këtyre dy abonentëve.
- Skanimi me laser – Përdor një pajtimtar për të projektuar të dhënat e diapazonit të kapur nga skaneri laser në robot. Abonenti merr mesazhe sensor_msgs/Laserscan nëpërmjet një teme të quajtur /scan.

Për të lëvizur robotin mund t'i përdorim pushbutonat që janë në *navigation control*. Ju duhet të shihni ndryshimin e gjendjes së robotit ndërsa lëvizni robotin në "sheshin e lojërave". Po kështu, ekranet e imazhit të kamerës dhe skanimit me laser duhet të përshtaten në përputhje me rrethanat ndërsa kamera dhe skaneri laser i robotit kapin ndryshime. Kur hapet ndërfaqja e përdoruesit, ajo krijon nyjën TurtleBot Controller të emërtuar

TB2_CONTROL. Udhëzimet që krijojnë nyjen dhe të gjithë botuesit/abonentët e përmendur më sipër regjistrohen në konstruktorin e TurtlebotCommunicator.

Udhëzimet që kryejnë detyrat për çdo botues/pajtimtar janë të gjitha të regjistruara në TurtlebotCommunicator.m.

- Botuesi i Komandës Velocity përdor metodat setAngularVelocity dhe setLinearVelocity të TurtlebotCommunicator për të publikuar të dhënat e shpejtësisë.
- Abonenti IMU Data përdor metodën imuDataCallback të TurtlebotCommunicator për të analizuar dhe shfaqur të dhënat e marra të orientimit.
- Abonenti Odometry Data përdor metodën odometriCallback të TurtlebotCommunicator për të analizuar dhe shfaqur të dhënat e pozicionit të marrë.
- Abonenti i Kamera Zi/Bardhë përdor metodën bwImgCallback të TurtlebotCommunicator për të analizuar të dhënat e imazhit për shfaqjen bardh e zi.
- Abonenti i Kamera me ngjyra përdor metodën colorImgCallback të TurtlebotCommunicator për të analizuar të dhënat e imazhit për shfaqjen me ngjyra.
- Abonenti Laser Scan përdor metodën laserCallback të TurtlebotCommunicator për të analizuar dhe shfaqur të dhënat e marra të skanimit me laser.

Komandat për çdo objekt UI në ndërfaqen e përdoruesit regjistrohen në turtlebot_control.m.

Për të dalë nga shembulli, fillimisht duhet të mbyllni nyjën TurtleBot Controller përpara se të mbyllni TurtleBot Simulator. Për të mbyllur nyjen TurtleBot Controller, thjesht mbyllni ndërfaqen e përdoruesit të TurtleBot Controller. Ju duhet të shihni daljen e mëposhtme të MATLAB pas një mbylljeje të suksesshme:

Mbyllja e nyjës /TB2_CONTROL në http://222.222.222.222:55555/.

Për të fikur TurtleBot Simulator:

- Kthehuni në desktopin e makinës virtuale.
- Shkoni në terminalin nga është nisur Gazebo. Ky është terminali që u hap
- automatikisht kur klikoni më herët shkurtoren Gazebo Playground.
- Në terminal, futni Ctrl-C. Kjo duhet të mbyllë si Gazebo ashtu edhe terminalin.
- Pas daljes së Gazebo, mund ta mbyllni makinën virtuale.

6.2 Lokalizimi me algoritmit Monte Carlo

Monte Carlo Localization (MCL) është një algoritëm për të lokalizuar një robot duke përdorur një particle filter. Algoritmi kërkon një hartë të njohur dhe detyra është të vlerësojë pozicionin (pozicionin dhe orientimin) e robotit brenda hartës bazuar në lëvizjen dhe ndjeshmërinë e robotit. Algoritmi fillon me një besim fillestar të shpërndarjes së probabilitetit të pozës së robotit, e cila përfaqësohet nga grimcat e shpërndara sipas një besimi të tillë. Këto grimca përhapen duke ndjekur modelin e lëvizjes së robotit sa herë që ndryshon poza e robotit. Me marrjen e leximeve të reja të sensorit, secila grimcë do të vlerësojë saktësinë e saj duke kontrolluar sa gjasa do të marrë lexime të tilla sensori në pozicionin e saj aktual. Më pas, algoritmi do të rishpërndajë (ri-shembullojë) grimcat tek grimcat e paragjykimit që janë më të sakta. Vazhdoni t'i përsërisni këto hapa lëvizës, ndijues dhe rimbushur, dhe të gjitha grimcat duhet të konvergjojnë në një grup të vetëm pranë pozës së vërtetë të robotit nëse lokalizimi është i suksesshëm.

Së pari, krijoni një TurtleBot të simuluar brenda një mjedisi zyre në një makinë virtuale duke ndjekur hapat në Fillimi me Gazebo dhe TurtleBot Simuluar (ROS Toolbox) për të nisur Gazebo Office World nga desktopi, siç tregohet më poshtë.

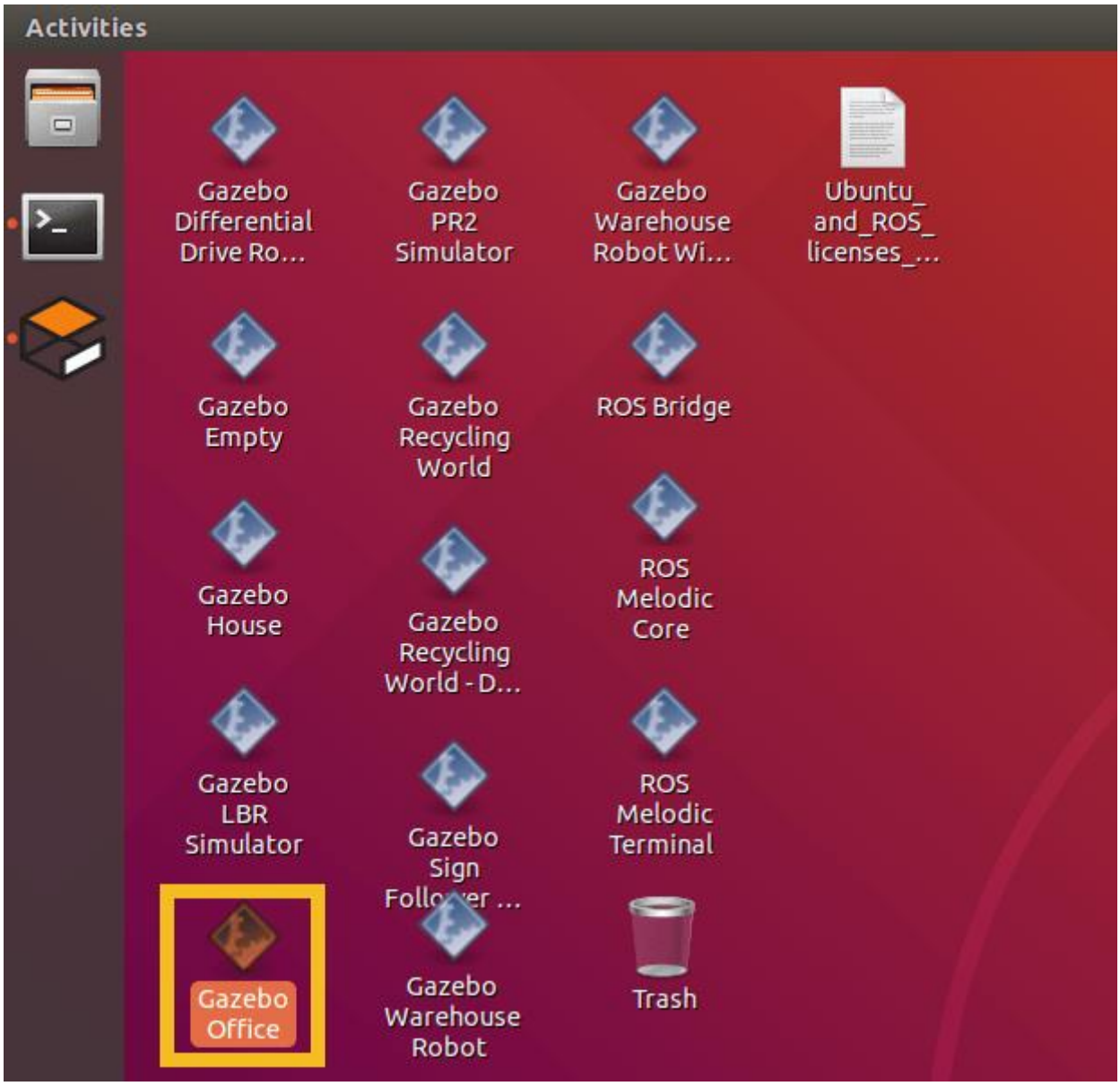

Figura 46: Gazebo si mjedis për simulim

Më pas, konektohemi me robotin duke përdorur instruksionet më poshtë:

```
ipaddress = '192.168.2.150';
rosinit(ipaddress,11311);
```

Me pas inicializojmë hartën për te naviguar neper te me instruksionin:

```
Initializing global node /matlab_global_node_73841 with NodeURI
http://192.168.2.1:53461/
```

Harta e inicializuar do të duket si më poshtë.

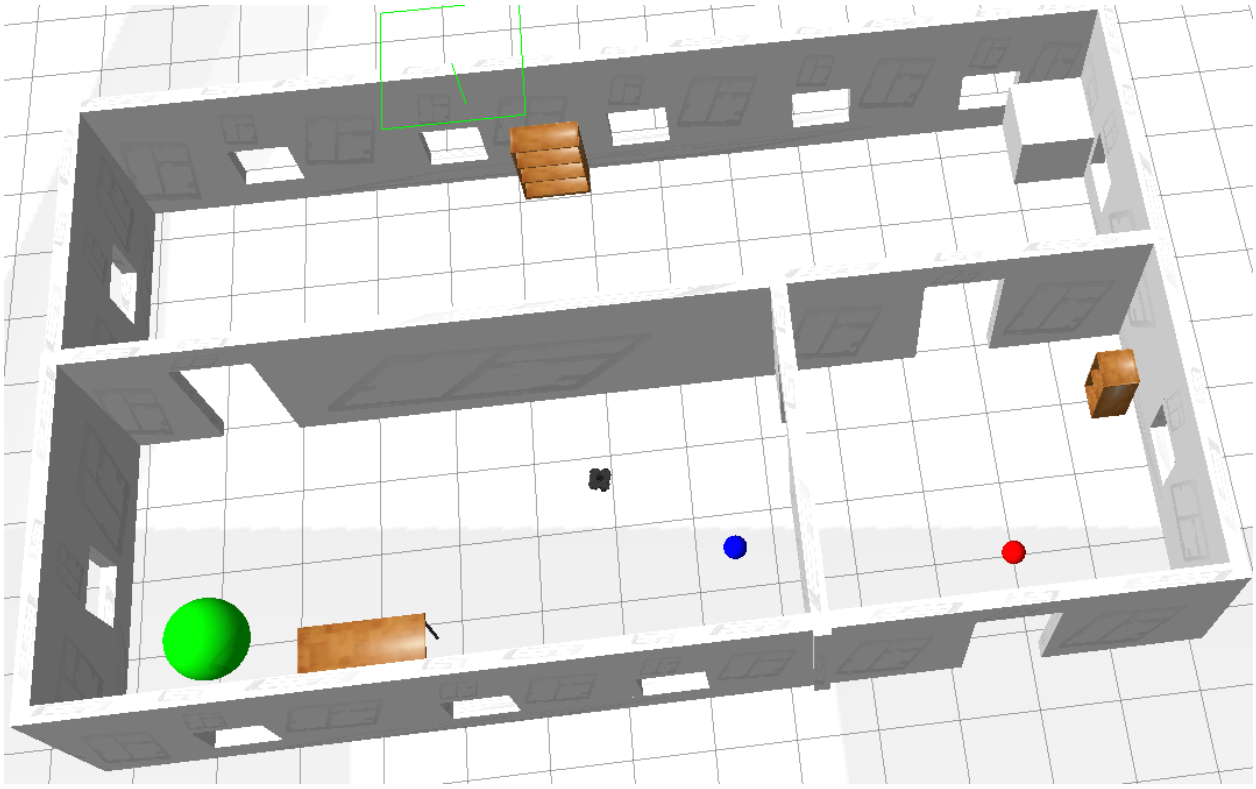

Figura 47: Harta e thirrur për lokalizim

```
Tani aktivizojmë laserin (sensorin laserik LiDAR) dhe caktojmë parametrat e
filtrit me instruksionet me poshtë:

odometryModel = odometryMotionModel;
odometryModel.Noise = [0.2 0.2 0.2 0.2];
```

Sensori në TurtleBot është një gjetës i simuluar i diapazonit i konvertuar nga leximet e Kinect. Metoda e fushës së gjasave përdoret për të llogaritur probabilitetin e perceptimit të një grupi matjesh duke krahasuar pikat fundore të matjeve të gjetësit të rrezes me hartën e zënies. Nëse pikat e fundit përputhen me pikat e zëna në hartën e zënies, probabiliteti për të perceptuar matje të tilla është i lartë. Modeli i sensorit duhet të akordohet që të përputhet me vetinë aktuale të sensorit për të arritur rezultate më të mira të testimit. Vetia SensorLimits përcakton diapazonin minimal dhe maksimal të leximeve të sensorit. Vetia Harta përcakton hartën e zënies së përdorur për llogaritjen e fushës së gjasave.

Pas përditësimit të parë të AMCL, grimcat gjenerohen duke marrë kampionimin e shpërndarjes Gaussian me mesatare të barabartë me amcl. Pozicioni fillestar dhe kovarianca e barabartë me amcl.Kovarianca fillestare. Kjo është treguar në figurën më poshtë

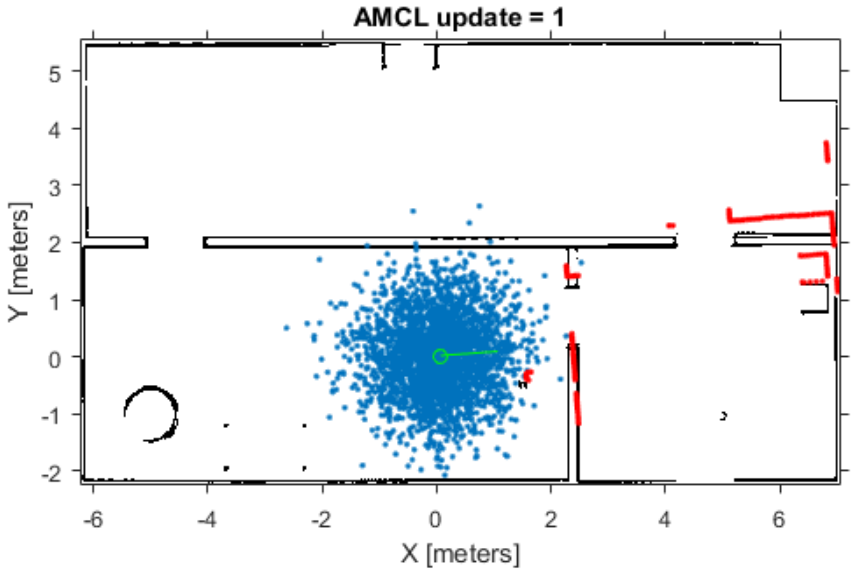

Figura 48: Grimcat lokalizuese të fituara pas përditësimit të parë [3]

Pas 8 përditësimeve, grimcat fillojnë të konvergjojnë në zona me probabilitet më të lartë. Kjo është treguar në figurën ne vijim.

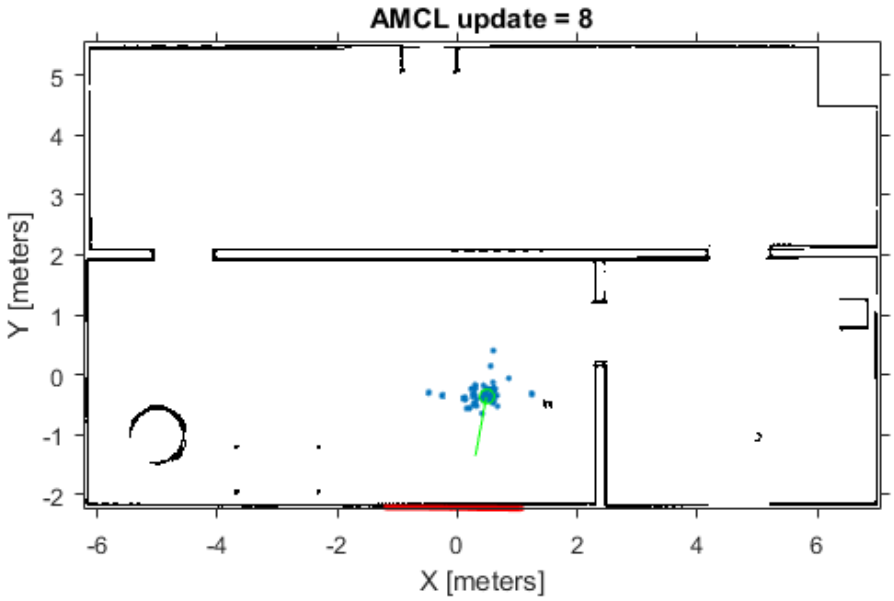

Figura 49: Grimcat lokalizuese pas 8 përditësimeve [3]

Pas 60 përditësimeve, të gjitha grimcat duhet të konvergojnë në pozicionin e duhur të robotit dhe skanimet me laser duhet të përputhen ngushtë me skicat e hartës. Kjo tregohet mw poshtw.

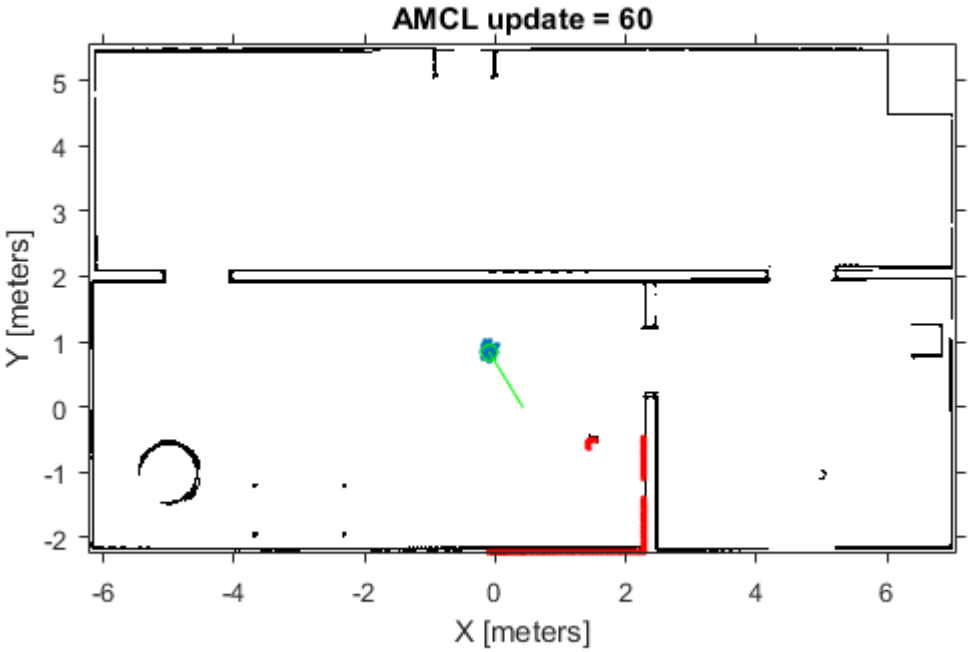

Figura 50: Pozicioni i saktë i robotit bazuar në leximet e laserit [3]

7. Programimi i robotëve mobil me Python

Python është bërë një nga gjuhët më të njohura të programimit në botë vitet e fundit. Përdoret në çdo gjë, nga machine learning deri te ndërtimi i faqeve të internetit dhe testimi i softuerit. Mund të përdoret si nga zhvilluesit ashtu edhe nga jo-zhvilluesit. Një nga gjuhët më të njohura, Python, përdoret gjerësisht nga zhvilluesit e teknologjisë në zhvillim, si dhe nga studiuesit e robotikës. Në robotikë, kjo gjuhë programimi është bërë një pjesë kyçe e sistemit operativ robotik (ROS) dhe përdoret për dizajnimin e sistemeve të integruara (ang. Embedded systems). Për shembull, sistemet e integruara dhe paketat shteruese të automatizimit të Raspberry Pi dhe Arduino janë krijuar duke përdorur këtë gjuhë.

7.1 Shembull fillestar me Python për Turtlebot 2i

Në vijim do të shohim një procedurë të thjeshtë për ekzekutim e një skripte fillestare për Turtlebot, të shkruar në Python.

Hapi i parë do jetë startimi i driverave minimal të Turlebot 2i. Këtë e bëjmë me instruksionin më poshtë.

```
roslaunch turtlebot_bringup minimal.launch
```

Më pas, skripten me përmbajtje standard po e marrim nga një folder në Git, që është pjesë e një dukementimi mjaft të detajuar mbi operimin e Turtlebot 2i. Këtë e bëjmë me kodin si më poshtë.

```
mkdir ~/helloworld
cd ~/helloworld/
git clone https://github.com/markwsilliman/turtlebot/
cd turtlebot
python goforward.py
```

Rrethi i parë krijon një folder me emrin helloworld. Rreshti i dytë kalon dhe hyn në këtë folder. Rreshti i tretë bën marrjen e file-it nga Git dhe e ruan në folderin tonë helloworld. Rreshti i katërt futet brenda folderitm turtlebot, dhe me rreshtin e fundit ekzekutojmë një skripte e shkruar në python, me emrin *goforward.py.* Në këtë rast, instruksionet për ta lëvizur robotin janë brenda këtij file. Përmbajta e këtij file-i është treguar në kodin në vijim.

```
importrospy
              fromgeometry_msgs.msgimportTwist
              classGoForward():
              def__init__(self):
              # initiliaze
              rospy.init_node('GoForward', anonymous=False)
                     # tell user how to stop TurtleBot
                     rospy.loginfo("To stop TurtleBot CTRL + C")
              # What function to call when you ctrl + c
              rospy.on_shutdown(self.shutdown)
                     # Create a publisher which can "talk" to TurtleBot and tell it to move
```

```
# Tip: You may need to change cmd_vel_mux/input/navi to /cmd_vel if you're
not using TurtleBot2
self.cmd_vel=rospy.Publisher('cmd_vel_mux/input/navi', Twist, queue_size=10)
       #TurtleBot will stop if we don't keep telling it to move.  How often
should we tell it to move? 10 HZ
r=rospy.Rate(10);
# Twist is a datatype for velocity
move_cmd=Twist()
       # let's go forward at 0.2 m/s
move_cmd.linear.x=0.2
       # let's turn at 0 radians/s
       move_cmd.angular.z=0
       # as long as you haven't ctrl + c keeping doing...
whilenotrospy.is_shutdown():
       # publish the velocity
self.cmd_vel.publish(move_cmd)
       # wait for 0.1 seconds (10 HZ) and publish again
r.sleep()
defshutdown(self):
# stop turtlebot
rospy.loginfo("Stop TurtleBot")
       # a default Twist has linear.x of 0 and angular.z of 0.  So it'll stop
TurtleBot
self.cmd_vel.publish(Twist())
       # sleep just makes sure TurtleBot receives the stop command prior to
shutting down the script
rospy.sleep(1)
if__name__=='__main__':
try:
GoForward()
except:
rospy.loginfo("GoForward node terminated.")
```

Nëse analizojmë kodin e mësipërm, do të shohim se struktura e tij është e formuar ashtu që të jetë e përshtatur me kërkesat e ROS. Kjo është shumë e rëndësishme që të ketë kompatibiletet mes ROS dhe një skripte të një pale tjetër. Kurse sa i përket levizjes së robotit, instruksioni që shkakton lëvizjes fizike të tij është ai në vijim:

```
self.cmd_vel=rospy.Publisher('cmd_vel_mux/input/navi', Twist, queue_size=10)
```

Pra nga kodi që kemi më lartë, tërë veprimi që robot kryhen është një lëvizje për 10 sekonda me shpejtësi prej 0.2 m/s përgjatë aksit x. Shpejtësia caktohet nga instruksioni më poshtë:

```
move_cmd.linear.x=0.2
```

Nëse kodin e mësipërm e ndryshojmë ashtu që vlerën linear.x e kalojmë nga 0.2 në 0, dhe poashtu vlerën angular.z e kalojmë nga 0 në 0.5, atëherë roboti do ketë vetëm komponentë të lëvizjes këndore dhe jo vijëdrejtë. Në esencë roboti do rrotullohet në trajektore rrethore.

Pjesë e folderit që kemi shkarkuar nga Git është edhe një Python file me emrin draw_a_square.py. Ky file është i dizajnuar ashtu që lëvizja e robotit të jetë në trajektore të mbyllura drejtkëndëshe. TurtleBot fillon të vizatojë katrorë në dysheme, por do të shihni që ai shpejt fillon të largohet nga rruga e tij fillestare. Këtu robotët dhe kompjuterët veprojnë shumë ndryshe. Nëse i kërkoni një kompjuteri të bëjë 1 + 1, do të merrni gjithmonë 2. Nëse i kërkoni një roboti të ecë përpara 1 metër, ai do të shkojë afërsisht një metër dhe jo plotësisht drejt. Njohja se ku ndodhet roboti (lokalizimi) është një nga sfidat klasike robotike, veçanërisht në mjedise të brendshme ku GPS nuk është mjaftueshëm i besueshëm ose i saktë. Ju mund të mendoni se për shkak se mund t'i thoni robotit të ecë përpara me 0.2 m/s, do të ishte e lehtë të programoni një robot në mënyrë që të shkojë përpara një metër duke publikuar një linear.x = 0.2 m/s për 5 sekonda. Nëse roboti do të ishte një kompjuter, kjo është saktësisht e drejtë, por për robotët kjo është shumë e pasaktë për shkak të rrëshqitjes, kalibrimit të papërsosur dhe faktorëve të tjerë. Nëse shkruani një skenar si ky për të vozitur në një shesh dhe për të vrapuar për 10 cikle, do të përfundoni diku krejtësisht ndryshe nga ku keni filluar!

7.2 Libraria Rospy dhe Turtlebot 2i

Rospy është një bibliotekë e pastër e klientit Python për ROS. API-ja e klientit rospy u mundëson programuesve të Python të ndërlidhen shpejt me Temat, Shërbimet dhe Parametrat e ROS. Dizajni i rospy favorizon shpejtësinë e zbatimit (d.m.th. kohën e zhvilluesit) mbi performancën e kohës së ekzekutimit, në mënyrë që algoritmet të mund të prototipohen shpejt dhe të testohen brenda ROS. Ai është gjithashtu ideal për kodin e rrugës jo-kritike, siç është kodi i konfigurimit dhe inicializimit. Shumë nga mjetet ROS janë të shkruara në ngjyrë rozë për të përfituar nga aftësitë e introspeksionit të tipit. Shumë nga mjetet ROS, të tilla si rostopic dhe rosservice, janë ndërtuar mbi rospy.Një përmbledhje e përgjithshme e rospy është e disponueshme këtu: http://wiki.ros.org/rospy/Overview. Në praktikë, ne përdorim rospy kryesisht për: (a) inicializimin dhe mbylljen e Turtlebot, (b) printimin e mesazheve në tastierë dhe (c) krijimin e botuesve dhe abonentëve.

Inicializimi

Kur shkruhet një program duke përdorur arkitekturën ROS, duhet të inicializoni një nyje -- një nyje për proces. Nyjet formojnë kolektivisht një grafik i cili lehtëson komunikimin e ndërnyjave nëpërmjet rrymave, shërbimeve dhe parametrave ROS. Për një diskutim të plotë të nyjave, lexoni: http://wiki.ros.org/Nodes (Lidhje me një sajt të jashtëm.). Tani për tani, duhet vetëm të inicializohet një nyje të emërtuar duke përdorur formatin si më poshtë:

```
rospy.init_node("nyja1")
```

Mbyllja

Ctrl + C është hyrja standarde e tastierës për mbyllje kur ekzekutohet një skrip Rospy. Në mbyllje, skripti aktual ekzekutohet dhe gjendja e robotit bëhet statike. Prandaj, është e nevojshme të zvogëlohen shpejtësitë e motorit përpara mbylljes. Kjo mund të arrihet si më poshtë:

```
rospy.on_shutdown(self.shutdown)
```

ku self.shutdown është një funksion në të cilin gjendja e robotit trajtohet menjëherë përpara daljes së skriptit. Një shembull i funksioneve të mbylljes është:

```
def shutdown(self):
    rospy.loginfo("Stop!")
```

```
    self.cmd_vel.publish(Twist())
    rospy.sleep(1)
```

Botuesit dhe Abonentët

Për të kuptuar rolin jetik të botuesve dhe abonentëve gjatë kodimit me rospy, këshillohet që së pari të kuptoni temat e ROS. Një përmbledhje e plotë mund të gjendet këtu: http://wiki.ros.org/Topics (Lidhje me një sajt të jashtëm.). Në thelb, një temë është një radhë e emërtuar dhe e aksesueshme mbi të cilën nyjat mund të shkëmbejnë mesazhe. Një temë është shtypur fort në një mesazh; vetëm informacioni i një lloji të caktuar mesazhi mund të publikohet në një temë.

Për të publikuar informacion në një temë, fillimisht duhet të inicializoni botuesin. Kjo mund të arrihet në mënyrën e mëposhtme:

```
pub = rospy.Publisher('topic_name', std_msgs.msg.String, queue_size=10)
```

Pasi e kemi emërtuar botuesin e mësipërm si "pub", tani mund të publikojmë mesazhe për atë temë si më poshtë:

```
pub.publish("Hello World")
```

Si parazgjedhje, një botues është sinkron; kalimi i argumentit queue_size si parazgjedhje në vend të zbatimit asinkron të rekomanduar.

Abonimi është një koncept më intuitiv. Duke supozuar se ekziston një temë, mund të abonoheni në mënyrën e mëposhtme:

```
rospy.Subscriber('topic_name', std_msgs.msg.String, process_topic)
```

Ku process_topic është një funksion i cili përpunon çdo informacion të marrë nga emri_temës. Për shembull, nëse topic_name përbëhej nga ndonjë informacion në lidhje me gjendjen e sensorëve të përplasjes në bazën Kobuki, atëherë procesi_topic duhet të thirret kur një ngjarje, p.sh. bumperi shtypet ose lëshohet, ndodh.

Skriptat roslaunch

Në mënyrë që netbook-i i Turtlebot të komunikojë me sensorët e tij, ne duhet të lëshojmë disa Nyje ROS për të trajtuar ato detaje për ne. Në veçanti, thirrja:

```
roslaunch turtlebot_bringup minimal.launch
```

na jep akses në ROS Nodes cmd_vel_mux dhe bazën mobile, ndër të tjera. Nga nyjet këto, cmd_vel_mux/input/navi dërgon kontrollin e motorëve në bazën Kobuki; /mobile_base/commands/led1 (dhe led2) na mundëson kontrollimin e LED-ve në bazë;

Në mënyrë të ngjashme, thirrja:

```
roslaunch turtlebot_bringup 3dsensor.launch
```

jep akses në nyjen e thellësisë së kamerës. Për të fituar njëw listë të plotë të mundësive me roslaunch, mund ta përdorim instruksionin:

```
rostopic list
```

për të parë të gjitha nyjat në dispozicion për ju. Në veçanti, '/scan' përmban skanimin me laser të skenës. '/camera/rgb/image_raw' përmban imazhin e papërpunuar RGB. Dhe '/camera/depth/image' përmban imazhin e thellësisë së papërpunuar.

Sensorat goditjes (Bumper sensors)

Të gjitha ngjarjet që lidhen me sensorin Bump dërgohen përmes një botuesi kur ekzekutoni komandën e nisjes:

```
roslaunch turtlebot_bringup minimal.launch
```

Prandaj, për të reaguar ndaj ngjarjeve të përplasjes, ne abonohemi te botuesi Bump. Një shembull vijon:

```
rospy.Subscriber('mobile_base/events/bumper', BumperEvent, processBump)
```

Në të cilin mobile_base/events/bumper është emri i Nyjes, BumperEvent është lloji i informacionit të publikuar në Node dhe processBump është një funksion që ne ofrojmë që përpunon çdo ngjarje. Kodi duhet të përfshijë thirrjen e mëposhtme të importit:

```
kobuki_msgs.msg import BumperEvent
```

Në mënyrë efektive, ne po kalojmë informacionin nga sensori i bump-it në processBump si parametër; ne ekzekutojmë trajtimin tonë të ngjarjeve në funksionin processBump.

Komandimi i motorëve

Për të kontrolluar motorët në bazën Kobuki, ne krijuam një botues që i kalon mesazhet një Nyje ROS. Në veçanti, ne përdorim një linjë si më poshtë për të hyrë në nyjën cmd_vel_mux/input/navi:

```
self.cmd_vel = rospy.Publisher('cmd_vel_mux/input/navi', Twist, queue_size=10)
```

Ky botues dërgon një radhë deri në 10 mesazhe të tipit Twist në nyjën që kontrollon motorët Kobuki.

7.3 Implementimi i Dijkstra me Python në Turtlebot 2i

Algoritmi i Dijkstra është një algoritëm për gjetjen e shtigjeve më të shkurtra ndërmjet nyjeve në një grafik, i cili mund të përfaqësojë, për shembull, rrjetet rrugore. Ajo u konceptua nga shkencëtari kompjuterik Edsger W. Dijkstra në 1956 dhe u botua tre vjet më vonë. Algoritmi Dijkstra përdor etiketa që janë numra të plotë pozitivë ose numra realë, të cilët janë të renditur plotësisht. Mund të përgjithësohet përdorimi i çdo etiketimi që është pjesërisht i renditur, me kusht që etiketat e mëvonshme (një etiketë e mëvonshme prodhohet kur përshkohet një skaj) janë monotonike jo-zvogëluese. Ky përgjithësim quhet algoritmi gjenerik i rrugës më të shkurtër të Dijkstra. Supozoni se dëshironi të gjeni shtegun më të shkurtër midis dy kryqëzimeve në një hartë të qytetit: një pikënisje dhe një destinacion. Algoritmi i Dijkstra fillimisht shënon distancën (nga pika e fillimit) deri në çdo kryqëzim tjetër në hartë me pafundësi. Kjo nuk bëhet për të nënkuptuar se ka një distancë të pafundme, por për të theksuar se ato kryqëzime nuk janë vizituar ende. Disa variante të kësaj metode i lënë të paetiketuar distancat e kryqëzimeve. Tani zgjidhni kryqëzimin aktual në çdo përsëritje. Për përsëritjen e parë, kryqëzimi aktual do të jetë pika fillestare dhe distanca deri në të (etiketa e kryqëzimit) do të jetë zero. Për përsëritjet e mëvonshme (pas të parës), kryqëzimi aktual do të jetë kryqëzimi më i afërt i pavizituar me pikën e fillimit (kjo do të jetë e lehtë për t'u gjetur).

Duke u bazuar mbi diskutimin e mësipërm, kjo qasje mund të përdoret në robotikë për të gjetur rrugën më të shkurtër që robot duhet të ndjekë për të gjeneruar një trajektori nga një pikë startuese deri tek caku. Në krahasim me algoritmet tjera të kërkimit (ang. search algorithms), algoritmi Dijkstra tregohet të jetë shumë më pak efektiv. Mirëpo ky algoritëm duhet vlerësuar për bazën që ofron mbi të cilin më pas janë ndërtuar algoritme shumë më efikase. Një krahasim i procesit të kërkimit mes algorimit A* dhe Dijkstra është treguar në figurën më poshtë.

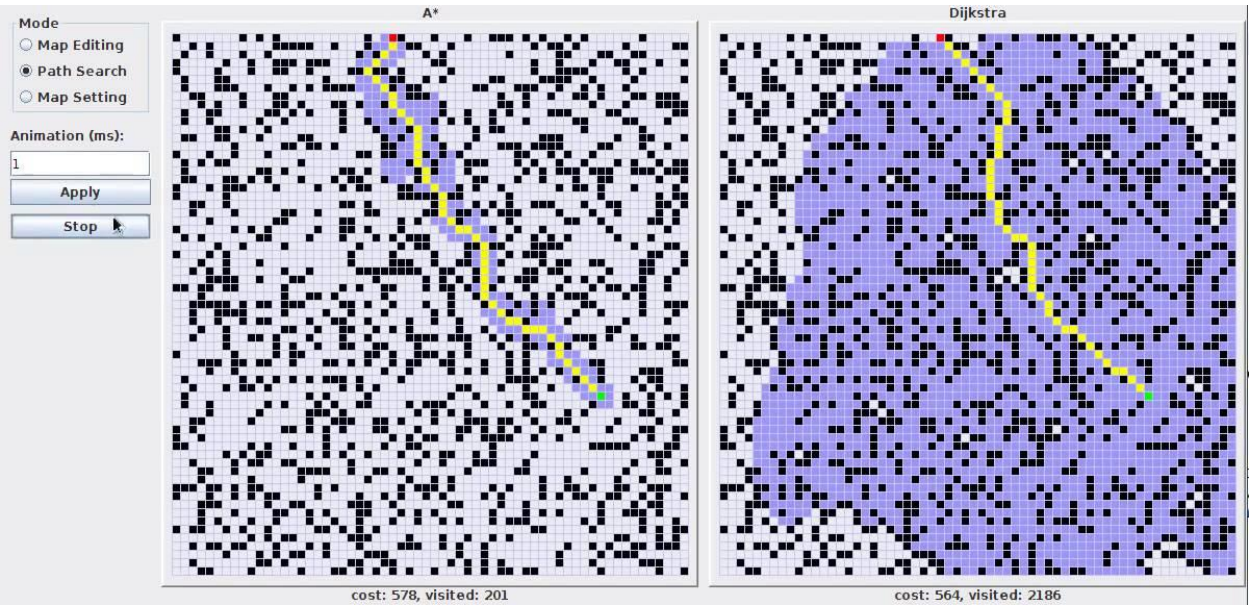

Figura 51: Krahasim mes A dhe Dijkstra [12]*

Një implementim i këtij algoritmi për të gjetur nga pikënisja deri në cak, duke përdorur turtlebotin si agjent gjatë procesit të kërkimit është treguar në vijim.

Fillimisht importohet libraritë dhe funksionet e nevojshme brenda këtyre librarive.

```
import maze
import roslib; roslib.load_manifest('project4')
import rospy
import tf
import transform2d
import numpy

from std_msgs.msg import String
from sensor_msgs.msg import LaserScan
from geometry_msgs.msg import Twist
from kobuki_msgs.msg import SensorState
```

Më pas krijojmë mjedisin (labirinthin) brenda të cilit do të vendoset turtleboti që të kërkojë të gjejë rrugën.

```
maze1 = [
    '+-+-+-+-+',
    '| |     |',
    '+ + +-+ +',
    '| | |   |',
```

```
        '+ + +-+-+',
        '|       |',
        '+-+-+ +-+',
        '|       |',
        '+ + +-+-+',
        '| |     |',
        '+ +-+-+ +',
        '| |     |',
        '+-+-+-+-+',
    ]
```

Krijohen variablat statike që janë të nevojshme përgjatë procesit.

```
CONTROL_PERIOD = rospy.Duration(0.01)
DEG = numpy.pi / 180.0
FT = 0.3048
ANGULAR_TOL = 0.03 # rad
ANGULAR_RAMP_TIME = 2.0 # s
ANGULAR_RAMP_SPEED = 0.8 # rad/s
ANGULAR_MIN_SPEED = 0.2 # rad/s
ANGULAR_GAIN = 3.0 # converts rad to rad/s
VELOCITY_TOL = 0.05
VELOCITY_RAMP_TIME = 2.0
VELOCITY_GAIN = 0.5
VELOCITY_RAMP_SPEED = 0.3
VELOCITY_MIN_SPEED = 0.1
VALID_COMMANDS = ['straight', 'turn_left', 'turn_right', 'nudge', 'forward']
VALID_DIRECTIONS = ['up','down','left','right']
```

Dhe më pas, implementimi është bërë me kodin si më poshtë:

```
class Controller:

    def __init__(self):

        rospy.init_node('Controller')

            self.tf_listener = tf.TransformListener()

        self.base_from_depth = None

        self.reset_state('initializing')

        self.angles = None
        self.ranges = None
        self.points = None

        self.cmd_vel_pub = rospy.Publisher('/mobile_base/commands/velocity',
                                           Twist)
```

```
    rospy.Subscriber('/scan', LaserScan, self.scan_callback)

    rospy.Subscriber('/maze_command', String, self.command_callback)

    rospy.Subscriber('/mobile_base/sensors/core',
                     SensorState, self.sensor_callback)

        self.odom_listener = tf.TransformListener()

    rospy.Timer(CONTROL_PERIOD, self.control_callback)

  def command_callback(self, msg):
    if self.state != 'idle':
      rospy.loginfo('not idle, so ignoring command {}'.format(
            msg.data))
      return

        m = maze.Maze()
        m.load_from_array(maze1)

        h = m.height()
        w = m.width()

        x0, y0, dir0, x1, y1 = maze.split_command(msg.data)
        if x0 > w-1 or x1 > w-1 or y0 > h-1 or y1 > h-1:
          rospy.logwarn('invalid coordinate: {}'.format(msg.data))
          return

        path = m.find_path(x0, y0, x1, y1)
        self.path_commands = maze.path_to_command(m, path, dir0)
        next_state = self.path_commands.pop(0)
        print "path_commands after first pop", self.path_commands
        self.reset_state(next_state)

  def reset_state(self, state, reset_time=True):

    self.state = state
    self.start_pose = None

    if reset_time:
      self.start_time = rospy.get_rostime()

    if state == 'initializing':
      rospy.loginfo('waiting for TF and laser scan...')
    elif state == 'idle':
      rospy.loginfo('waiting for commands...')

  def sensor_callback(self, msg):
```

```
    if ((msg.cliff or msg.bumper) and
        self.state not in ['idle', 'initializing']):
        rospy.loginfo('going to idle for safety stop')
        self.reset_state('idle')

  def setup_tf(self):

    try:

        ros_xform = self.tf_listener.lookupTransform(
            '/base_footprint', '/camera_depth_frame',
            rospy.Time(0))

    except tf.LookupException:

        return False

    self.base_from_depth = \
        transform2d.transform2d_from_ros_transform(ros_xform)

    return True

  def scan_callback(self, msg):

    if self.base_from_depth is None:
        if not self.setup_tf():
            return

    count = len(msg.ranges)

    self.angles = (msg.angle_min +
                   numpy.arange(count, dtype=float) * msg.angle_increment)

    self.ranges = numpy.array(msg.ranges)

    pts = self.ranges * numpy.vstack( ( numpy.cos(self.angles),
                                        numpy.sin(self.angles) ) )

    self.points = self.base_from_depth.transform_fwd(pts).T

  def lookup_angles(self, desired_angles, cutoff=3*DEG):

    if self.angles is None:
        return None

    ok_idx = numpy.nonzero(~numpy.isnan(self.ranges))[0]

    if not len(ok_idx):
```

```
        return None

    indices = []

    for angle in desired_angles:

        angle_err = numpy.abs(angle - self.angles[ok_idx])
        i = angle_err.argmin()

        if angle_err[i] > cutoff:
            return None

        indices.append(ok_idx[i])

    return indices

def points_at_angles(self, desired_angles, cutoff=3*DEG):

    indices = self.lookup_angles(desired_angles, cutoff)
    if indices is None:
        return None

    return self.points[indices]

def get_current_pose(self):

    try:
        ros_xform = self.tf_listener.lookupTransform(
            '/odom', '/base_footprint',
            rospy.Time(0))

    except tf.LookupException:
        return None

    xform2d = transform2d.transform2d_from_ros_transform(ros_xform)

    return xform2d

def envelope(self, error, tol, gain,
             min_speed, ramp_time, ramp_speed):

    time = (rospy.get_rostime() - self.start_time).to_sec()
    command = error * gain
    command = numpy.sign(command) * max(abs(command), min_speed)
    effective_time = min(time, ramp_time)
    max_speed = effective_time * ramp_speed/ramp_time
    command = numpy.clip(command, -max_speed, max_speed)
    done = abs(error) < tol
```

```
    return command, done

def angular_envelope(self, error):
    return self.envelope(error, ANGULAR_TOL, ANGULAR_GAIN,
                         ANGULAR_MIN_SPEED,
                         ANGULAR_RAMP_TIME, ANGULAR_RAMP_SPEED)

def linear_envelope(self, error):
        return self.envelope(error, VELOCITY_TOL,  VELOCITY_GAIN,
                             VELOCITY_MIN_SPEED,
                             VELOCITY_RAMP_TIME,  VELOCITY_RAMP_SPEED)

def go_straight(self, rel_pose, desired_distance):
        cmd_vel = Twist()
    distance_error = desired_distance - rel_pose.x
        command, done = self.linear_envelope(distance_error)

    cmd_vel.linear.x = command
        cmd_vel.angular.z = 0

        return cmd_vel, done


def control_callback(self, timer_event=None):
        print 'state: ',self.state
    cmd_vel = Twist()

    cur_pose = self.get_current_pose()

    if cur_pose is not None:
        if self.start_pose is None:
            self.start_pose = cur_pose.copy()
        rel_pose = self.start_pose.inverse() * cur_pose

    if self.state == 'initializing':

        if cur_pose is not None and self.angles is not None:
            self.reset_state('idle')

    elif self.state == 'straighten':

        points = self.points_at_angles([1*DEG, -1*DEG])

        if points is None or points[0][1] > 6*FT or points[1][1] > 6*FT:
                         next_state = self.path_commands.pop(0)
            self.reset_state(next_state)
```

```
else:
   delta = points[1]-points[0]

   angular_error = numpy.arctan2(delta[0], -delta[1])

   command, done = self.angular_envelope(angular_error)

   cmd_vel.angular.z = command

   if done:
                next_state = self.path_commands.pop(0)
      self.reset_state(next_state)

  elif self.state == 'turn_left':
     angle_error = (numpy.pi)/2 - rel_pose.theta
     command, done = self.angular_envelope(angle_error)
     cmd_vel.angular.z = command
     if done:
   self.reset_state ('straighten', reset_time = False)

  elif self.state == 'turn_right':
     angle_error = -(numpy.pi-0.1)/2 - rel_pose.theta
     command, done = self.angular_envelope(angle_error)
     cmd_vel.angular.z = command

     if done:
  self.reset_state ('straighten', reset_time = False)

  elif self.state == 'forward':

     cmd_vel, done = self.go_straight(rel_pose, 0.756)

     if done:
  self.reset_state('nudge', reset_time = False)

  elif self.state == 'nudge':
            point_0 = self.points_at_angles([0])

            if point_0 is None:

            rospy.logwarn('points was None in nudge state - '
             'backing up')
                     cmd_vel, done = self.go_straight(rel_pose, -0.2)

  else:
                     measured_dist = point_0[0][0]
                     rounded_dist = numpy.floor(measured_dist/(3*FT))*3*FT + 1.5*FT
                     difference_dist = measured_dist - rounded_dist
```

```
                        command, done = self.linear_envelope(difference_dist)
                        cmd_vel.linear.x = command

                        if done:
                                next_state = self.path_commands.pop(0)
                                self.reset_state(next_state)

        elif self.state != 'idle':

          rospy.logerr('invalid state {}'.format(self.state))
          rospy.signal_shutdown('invalid state')

        self.cmd_vel_pub.publish(cmd_vel)

    def run(self):
        rospy.spin()

if __name__ == '__main__':
    try:
        c = Controller()
        c.run()
    except rospy.ROSInterruptException:
        pass
```

7.4 Implementimi i kërkimit A* me Python në Turtlebot 2i

A* është një algoritëm i kalimit të grafikut dhe kërkimit të shtigjeve, i cili përdoret shpesh në shumë fusha të shkencës kompjuterike për shkak të plotësisë, optimalitetit dhe efikasitetit optimal. Një pengesë kryesore praktike është kompleksiteti i hapësirës, pasi ruan të gjitha nyjet e krijuara në memorie. Kështu, në sistemet praktike të rrugëtimit të udhëtimit, ai përgjithësisht është më i mirë se algoritmet që mund të përpunojnë paraprakisht grafikun për të arritur performancë më të mirë, si dhe qasje të kufizuara me memorie; megjithatë, A* është ende zgjidhja më e mirë në shumë raste. Në krahasim me algoritmin Dijkstra, ky algoritëm është mesatarisht 60 herë më i shpejtë.

A* është një algoritëm kërkimi i informuar, ose një kërkim i parë më i mirë, që do të thotë se është formuluar në terma të grafikëve të peshuar: duke u nisur nga një nyje specifike fillestare e një grafi, ai synon të gjejë një shteg për në nyjen e synuar të dhënë që ka më të vogël kosto (distanca më e vogël e udhëtuar, koha më e shkurtër, etj.). Ai e bën këtë duke mbajtur një pemë shtigjesh me origjinë nga nyja fillestare dhe duke i zgjeruar ato shtigje një skaj në një kohë derisa të plotësohet kriteri i përfundimit të tij.

Për shkak të këtyre vetive që ky algoritëm ka, është e natyrshme që të gjejë aplikim në robotikë, me theks të veçantë tek robotët mobil. Një implementim i këtij algoritmi, duke përdorur gjuhën programuese Python, për rastin e turtlebot është treguar në vijim.

```
import math
```

```
import rospy
from geometry_msgs.msg import Twist
from std_msgs.msg import String
import numpy as np
import time
from heapq import heappush, heappop
from matplotlib.patches import Rectangle
import matplotlib.pyplot as plt
plt.ion()

rospy.init_node('Astar', anonymous=True)
velPub = rospy.Publisher('cmd_vel', Twist, queue_size=100)
msg = Twist()

turtlebot3_model = rospy.get_param("model", "burger")

msg.linear.x = 0.0
msg.linear.y = 0.0
msg.linear.z = 0.0
msg.angular.x = 0.0
msg.angular.y = 0.0
msg.angular.z = 0.0

velPub.publish(msg)
thresholdDistance=0.1  #0.01
nd=int(1/thresholdDistance)  #100
sizex=10*nd  #1000
thresholdAngle=15   #10
na=int(360/thresholdAngle)  #36

def ros_inputs(Xi, Yi, Thetai, UL, UR):
    r = 0.033 #in metres
    L = 0.16 #in metres
    dt = 10
    UL = UL*2*math.pi/60
    UR = UR*2*math.pi/60
    thetan = 3.14 * Thetai / 180
    theta_dot = (r / L) * (UR - UL)
    change_theta = theta_dot + thetan
    x_dot = (r / 2) * (UL + UR) * math.cos(change_theta)
    y_dot = (r / 2) * (UL + UR) * math.sin(change_theta)
    vel_mag = math.sqrt(x_dot** 2 + y_dot** 2)
    F_theta = (180*change_theta/ 3.14)
    return vel_mag, theta_dot, F_theta

#Normalizimi i pozitës së pragjeve
def normalize(Node):
    x,y,t = Node[0],Node[1],Node[2]
```

```
    x = round(x/thresholdDistance)* thresholdDistance
    y = round(y/thresholdDistance)* thresholdDistance
    t = round(t/thresholdAngle) * thresholdAngle
    x=round(x,4)
    y=round(y,4)
    n=t//360
    t=t-(360*n)
    t=(t/thresholdAngle)
    return [x,y,int(t)]


# Llogaritja e distancës
def distance(startCoordinate,goalCoordinate):
    sx,sy = startCoordinate[0],startCoordinate[1]
    gx,gy = goalCoordinate[0],goalCoordinate[1]
    return math.sqrt((gx-sx)**2 + (gy-sy)**2)#instead of using scipy.spatial importing distance
def listToString(s):
        str1 = ""
        for ele in s:
            str1 += str(ele)
        return str1

#Kontrollimi i kufinjeve tw pengesave qw haspwsiten e robotit
def boundary_check(i,j):
    if (i<tot or j>=10-tot or j<tot or i>=10-tot):
        return 0
    else:

        return 1
#Harta e pengesave
def obs_map(x,y):
    circle1 = ((np.square(x-5))+ (np.square(y-5)) <=np.square(1+tot))
    circle2 = ((np.square(x-3))+ (np.square(y-2)) <=np.square(1+tot))
    circle3 = ((np.square(x-7))+ (np.square(y-2)) <=np.square(1+tot))
    circle4 = ((np.square(x-7))+ (np.square(y-8)) <=np.square(1+tot))
    square1=(x>=0) and (x<=1.75+tot) and (y>=4.25-tot) and (y <= 5.75+tot)
    square2=(x>=8.25-tot) and (x<=9.75+tot) and (y>=4.25-tot) and (y <= 5.75+tot)
    square3=(x>=2.25-tot) and (x<=3.75+tot) and (y>=7.25-tot) and (y <= 8.75+tot)
    if circle1 or circle2 or circle3 or circle4 or square1 or square2 or square3 :
        obj_val =0
    else:
        obj_val = 1

    return obj_val


def plot_curve(Xi,Yi,Thetai,UL,UR):
    t = 0
```

```
    r = 0.033
    L = 0.16
    dt = 1
    Xn=Xi
    Yn=Yi
    Thetan = math.radians(Thetai)
    while t<10:
        t = t + dt
        Xs = Xn
        Ys = Yn
        Xn += (0.5*r) * (UL + UR) * math.cos(Thetan) * dt
        Yn += (0.5*r) * (UL + UR) * math.sin(Thetan) * dt
        Thetan += (r / L) * (UR - UL) * dt
        plt.plot([Xs, Xn], [Ys, Yn], color="red")
    Thetan = math.degrees(Thetan)
    return [Xn, Yn, Thetan]

def move_bot(Xi,Yi,Thetai,UL,UR,s,n):
    t = 0
    r = 0.033
    L = 0.16
    dt = 1
    Xn=Xi
    Yn=Yi
    length=0
    Thetan = math.radians(Thetai*thresholdAngle)
    while t<10:
        t = t + dt
        Xs = Xn
        Ys = Yn
        Xn += (0.5*r) * (UL + UR) * math.cos(Thetan) * dt
        Yn += (0.5*r) * (UL + UR) * math.sin(Thetan) * dt
        length+=distance([Xs,Ys],[Xn,Yn])
        status=boundary_check(Xn,Yn)
        flag=obs_map(Xn,Yn)
        if (status!=1) or (flag != 1):
            return None
        Thetan += (r / L) * (UR - UL) * dt
        s.append((Xs,Ys))
        n.append((Xn,Yn))
    Thetan = math.degrees(Thetan)
    # print(Xn, Yn, Thetan,length)
    return [Xn, Yn, Thetan,length]

# Priority Queue Function
def  convergence(a,goal_node):
        if((np.square(a[0]-goal_node[0]))+ (np.square(a[1]-goal_node[1])) <=np.square(0.25)) :
                return 0
```

```
        else:
            return 1

###############  PWRCJELLA E INPUTEVE ################################
# fusheveprimi i robotit
print("#####################################################")
print("Per caktim manual shtypni 1, per tjera shtypni 2")
print("#####################################################")
inp=int(input())
m=False
if inp==1:
   m=True


########  para startimi #####################
if(not m):
      tot=0.18
      rpm1=2
      rpm2=3
      x_start = float(0.5)
      y_start = float(2)
      theta_start = int(0)
      x_goal=float(4.5)
      y_goal=float(2)
      theta_goal = int(15)

actions=[[rpm1,0],[0,rpm1],[rpm1,rpm1],[0,rpm2],[rpm2,0],[rpm2,rpm2],[rpm1,rpm2],[rpm2,rpm1]]
      start=normalize([x_start,y_start,theta_start])
      norm_current_node=start
      goal=normalize([x_goal,y_goal,theta_goal])
      goal_node=[goal[0],goal[1],goal[2]]
# print(move_bot(norm_current_node[0], norm_current_node[1], norm_current_node[2], actions[0][0],
actions[0][1]))

#################### user init ################################
#Shpejtesia e rrotave (RPM)
if(m):
      # fusheveprimi i robotit
      tot=float(input("Jepni fusheveprimin e robotit:"))

      rpm1=int(input("Jepni RPM per motorin 1:"))
      rpm2=int(input("Jepni RPM per motorin 2:"))

actions=[[rpm1,0],[0,rpm1],[rpm1,rpm1],[0,rpm2],[rpm2,0],[rpm2,rpm2],[rpm1,rpm2],[rpm2,rpm1]]

      #Getting the start nodes from the user
      x_start = float(input("Please enter start point x coordinate:"))
      x_start=5-x_start
```

```
    y_start = float(input("Please enter start point y coordinate:"))
    y_start=5-y_start
    theta_start = int(input("Please enter start orientation in degrees:"))
    start_obs = obs_map(x_start,y_start)
    start_boundary = boundary_check(x_start,y_start)

    while(start_obs!=1 or start_boundary!=1):
      print("Incorrect start point! Please enter a valid start point:")
      x_start = float(input("Please enter start point x coordinate:"))
      x_start=5-x_start
      y_start = float(input("Please enter start point y coordinate:"))
      y_start=5-y_start
      theta_start = int(input("Please enter start orientation in degrees:"))
      start_obs = obs_map(x_start,y_start)
      start_boundary = boundary_check(x_start,y_start)

    start=normalize([x_start,y_start,theta_start])
    # start_node=[int(start[0]),int(start[1]),start[2]]
    norm_current_node=start
    #Geting the goal nodes from the user
    x_goal=float(input("Please enter goal point x coordinate:"))
    x_goal=5-x_goal
    y_goal=float(input("Please enter goal point y coordinate:"))
    y_goal=5-y_goal
    theta_goal=0
    goal_obs=obs_map(x_goal,y_goal)
    goal_boundary=boundary_check(x_goal,y_goal)


    while(goal_obs!=1 or goal_boundary!=1):
      print("Incorrect goal point! Please enter a valid goal point:")
      x_goal=float(input("Please enter another goal point x coordinate:"))
      x_goal=5-x_goal
      y_goal=float(input("Please enter another goal point y coordinate:"))
      y_goal=5-y_goal
      theta_goal=0
      goal_obs=obs_map(x_goal,y_goal)
      goal_boundary=boundary_check(x_goal,y_goal)
    goal=normalize([x_goal,y_goal,theta_goal])
    goal_node=[goal[0],goal[1],goal[2]]
################### INTIALISING COST,VISITED,PARENT AND QUEUE ###############
#Initializing cost as infinity
cost_array=np.array(np.ones((sizex,sizex,na)) * np.inf)
#Initializing visited nodes as empty array
visited=np.array(np.zeros((sizex,sizex,na)))
#Heuristic distance-Eucledian

#Initializing total cost with cost and distance
```

```
# totalcost=np.array(np.ones((sizex,sizex,na) * np.inf))
totalcost=np.array(np.ones((sizex,sizex,na)) * np.inf)

parent={}
Q=[]
# append start point and initialize it's cost to zero
heappush(Q,(0,start))
cost_array[int(nd*start[0])][int(nd*start[1])][start[2]]=0
totalcost[int(nd*start[0])][int(nd*start[1])][start[2]]=0
explored=[]
###################### A-STAR#################################
start_time=time.time()
breakflag=0
s=[]
n=[]
ps=[]
pn=[]

while convergence(norm_current_node,goal_node):
   if breakflag==1:
      break
   norm_current_node=heappop(Q)[1]
   if convergence(norm_current_node,goal_node)==0:
      goalfound=[norm_current_node,action]
      print("found")
      print(norm_current_node)
      break
   for action in actions:
      curr_node=move_bot(norm_current_node[0],     norm_current_node[1],     norm_current_node[2],
action[0], action[1],s,n)
      if(curr_node==None):
         continue
      L=curr_node[3]
      curr_node=curr_node[0:3]
      norm_curr_node=normalize(curr_node)
      if convergence(norm_curr_node,goal_node)==0:
         print("found")
         parent[listToString(norm_curr_node)]=[norm_current_node,action]
         goalfound=[norm_curr_node,action]
         print(norm_curr_node)
         breakflag=1
         break
      status=boundary_check(norm_curr_node[0],norm_curr_node[1])
      flag=obs_map(norm_curr_node[0],norm_curr_node[1])
      if (status and flag == 1):
        if visited[int(nd*norm_curr_node[0]),int(nd*norm_curr_node[1]),norm_curr_node[2]]==0:
         visited[int(nd*norm_curr_node[0]),int(nd*norm_curr_node[1]),norm_curr_node[2]]=1
         explored.append([norm_current_node,action,norm_curr_node])
```

```
        parent[listToString(norm_curr_node)]=[norm_current_node,action]

cost_array[int(nd*norm_curr_node[0]),int(nd*norm_curr_node[1]),norm_curr_node[2]]=L+cost_array[i
nt(nd*norm_current_node[0]),int(nd*norm_current_node[1]),norm_current_node[2]]

totalcost[int(nd*norm_curr_node[0]),int(nd*norm_curr_node[1]),norm_curr_node[2]]=cost_array[int(n
d*norm_curr_node[0]),int(nd*norm_curr_node[1]),norm_curr_node[2]]   +   distance(norm_curr_node,
goal_node)
        heappush(Q,(
totalcost[int(nd*norm_curr_node[0]),int(nd*norm_curr_node[1]),norm_curr_node[2]]
,norm_curr_node ))
      else:
         if
totalcost[int(nd*norm_curr_node[0]),int(nd*norm_curr_node[1]),norm_curr_node[2]]>(totalcost[int(nd
*norm_current_node[0]),int(nd*norm_current_node[1]),norm_current_node[2]]+sum(action)):

totalcost[int(nd*norm_curr_node[0]),int(nd*norm_curr_node[1]),norm_curr_node[2]]=(totalcost[int(nd
*norm_current_node[0]),int(nd*norm_current_node[1]),norm_current_node[2]]+sum(action))
            explored.append([norm_current_node,action,norm_curr_node])
            parent[listToString(norm_curr_node)]=[norm_current_node,action]


################## BACKTRACK ##########################
path=[]
def path_find(goal,start):
   path.append(goal)
   GN=parent[listToString(goal[0])]
   path.append(GN)
   while (GN[0]!=start):
      GN=parent[listToString(GN[0])]
      path.append(GN)
   return path


path=path_find(goalfound,start)
path.reverse()
print("Time taken")
print(time.time()-start_time)

sx=x_start
sy=y_start
sz=theta_start
fig, ax = plt.subplots()
ax.set(xlim=(0, 10), ylim=(0, 10))

plt.plot(norm_curr_node[0], norm_curr_node[1], color='green', marker='o', linestyle='dashed',
linewidth=1,markersize=4)
plt.plot(start[0], start[1], color='yellow', marker='o', linestyle='dashed', linewidth=1,markersize=4)
```

```
c1 = plt.Circle((5, 5), 1, fill=None)
c2 = plt.Circle((3, 2), 1, fill=None)
c3 = plt.Circle((7, 2), 1, fill=None)
c4 = plt.Circle((7, 8), 1, fill=None)
currentAxis = plt.gca()
currentAxis.add_patch(Rectangle((0.25, 4.25), 1.5, 1.5, fill=None, alpha=1))
currentAxis.add_patch(Rectangle((8.25, 4.25), 1.5, 1.5, fill=None, alpha=1))
currentAxis.add_patch(Rectangle((2.25, 7.25), 1.5, 1.5, fill=None, alpha=1))
currentAxis.add_patch(Rectangle((0, 0), 10, 10, fill=None, alpha=1))

ax.add_artist(c1)
ax.add_artist(c2)
ax.add_artist(c3)
ax.add_artist(c4)
ax.set_aspect('equal')
plt.grid()
for action in path:
   x1= plot_curve(sx,sy,sz, action[1][0],action[1][1])
   sx=x1[0]
   sy=x1[1]
   sz=x1[2]
plt.show()
plt.pause(1)
plt.close()
ros=[]
for i in range (len(path)):
   ros.append(path[i][1])
print(ros)
c =0
r = rospy.Rate(10)
for action in ros:
      print("action")
      print(action)

      while not rospy.is_shutdown():
         if c== 101:
            msg.linear.x = 0
            msg.angular.z = 0
            velPub.publish(msg)
            break
         else:
            vel , th, F_theta = ros_inputs(4.5,3,0,action[0],action[1])
            msg.linear.x = vel*10
            msg.angular.z =  th*10
            velPub.publish(msg)
            c=c+1
            r.sleep()
```

c=0

Një shembull më i kondenzuar i implementimit të algoritmit të kërkimit A* si pseudokod, është treguar në vijim.

```
makeanopenlistcontainingonlythestartingnode// krijimi i nje liste e cila ka vetem
nyjen startuese

   makeanemptyclosedlist  // krijimi i nje liste te mbyllur

   while (thedestinationnodehas not beenreached): // ne qofte se qellimi nuk
eshte arritur

       considerthenodewiththelowestfscore in theopenlist  // konsiderohet nyja qe
ka vleren f me te vogel ne listen e hapur

       if (thisnodeisourdestinationnode) : // ne qofse kjo nyje eshte
destinacioni yne

           wearefinished// atehere ketu arrihet qellimi

       if not:

           putthecurrentnode in theclosedlist and lookatallofitsneighbors// nyja
e cila nuk ka qene qellimi atehere kalon ne listen e mbyllur

           for (eachneighborofthecurrentnode):

               if (neighborhaslowergvaluethancurrent and is in theclosedlist) :
// ne qofse fqinji ka vlere me te vogel se g dhe eshte ne listen e mbyllur

                   replacetheneighborwiththenew, lower, gvalue  // zevendesojme
fqinjin me nje vleren g me te vogel

                   currentnodeisnowtheneighbor'sparent

               elseif (currentgvalueislower and thisneighboris in theopenlist ) :
// ne qofte se vlera g eshte me e vogel dhe fqinji i tij eshte ne listen e hapur

                   replacetheneighborwiththenew, lower, gvalue//zevenedsojm
fqinjin me nje vlere g me te vogel

                   changetheneighbor'sparenttoourcurrentnode


               elseifthisneighboris not in bothlists:

                   addittotheopenlist and setitsg
```

Të rikujtojmë se A* është një fakt një algoritëm heuristik, që nënkupton se kërkimet në stadet më të thella të hartës bazohen jo vetëm në leximet aktuale që bëhen nga infrastruktura heuristike, por edhe nga leximet e suksesshme që janë bërë më herët gjatë iterimit nëpër hartë. Një shembull që tregon parimin e kërkimit të këtij algoritmi është treguar në figurën më poshtë.

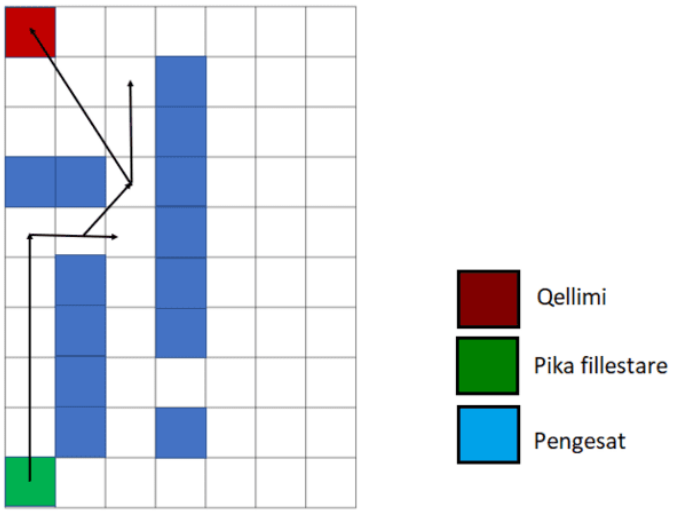

Figura 52: Parimi i kërkimit sipas A [9]*

Kurse kërkimi në një fushë më të gjerë, që mund t'i asociojë një situate reale, është treguar në figurën vijuese.

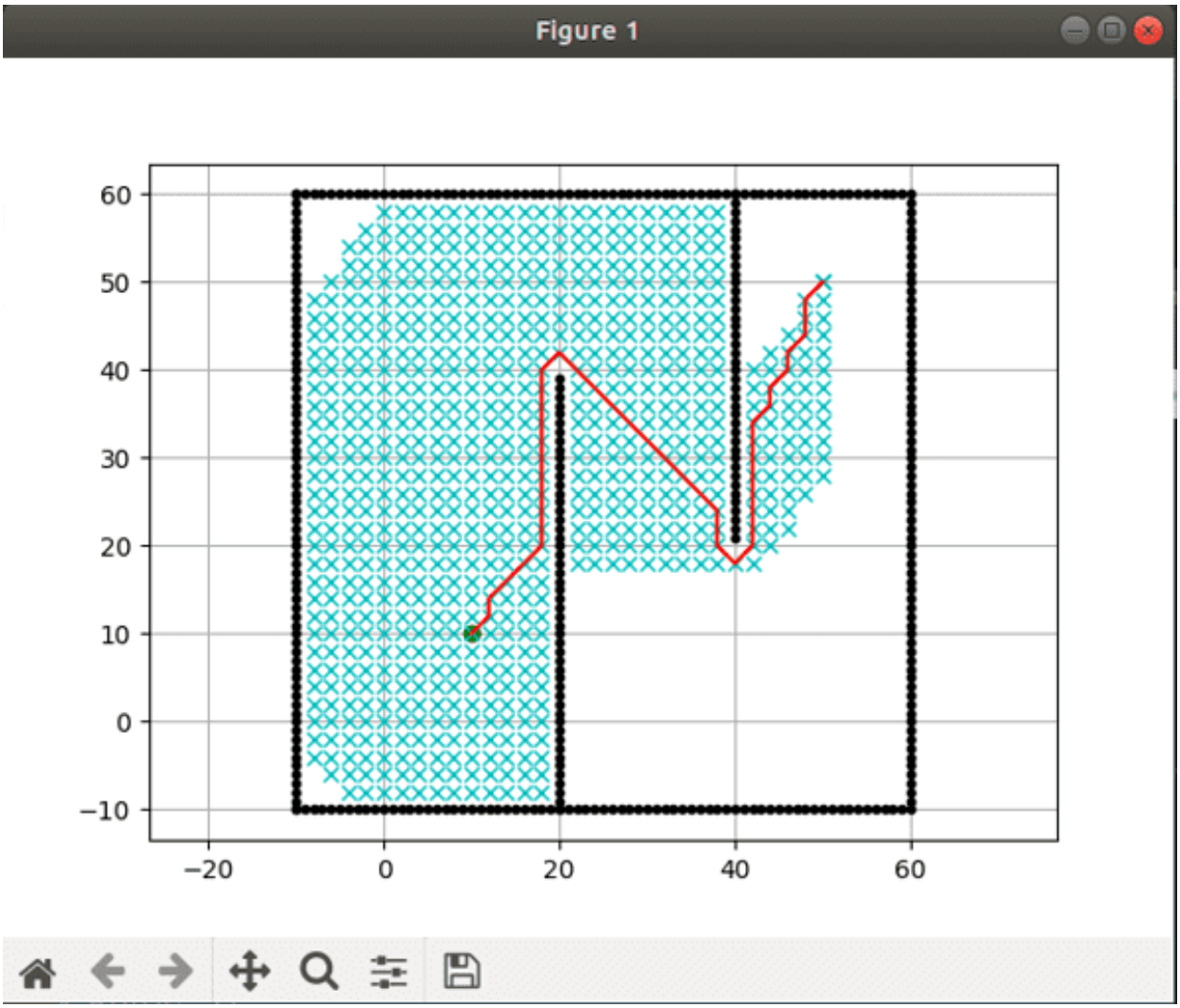

Figura 53: Kërkimi sipas A [9]*

8. Përfundim

Gjatë këtij punimi, kemi trajtuar në përgjithësi problematikën e robotëve mobil, por me fokus të veçantë tek roboti Turtle 2i. Është trajtuar modelimi i robotëve mobil dhe ndërtimi i një baze matematikore për përshkrimin e këtyre robotëve. Bazuar mbi këtë model matematik kemi derivuar një informatë shumë të rëndësishme për robotët mobil, që është përdorimi i odometrisë për lokalizim dhe navigim. Pra duke u bazuar në modelin matematik, kemi një nxjerrë një alternativë se si mund të gjejmë pozitën e robotit duke u bazuar vetëm në informatën e lëvizjes së tij. Nga ky diskutim më pas kemi parë se pasigurisë dhe jo-idealiteti që paraqiten në këto situata janë të tilla që odometrinë e bëjnë jo të mjaftueshme për navigim/lokalizim.

Për të pasur një besueshmëri më të madhe për lokalizim, robotët janë të pajisur me një infrastrukturë të tërë sensorët për të ndijuar mjedisin në të cilin ndodhet. Si më të rëndësishme mund t'i përmendim sonarët, laserët dhe kamerat me thellësi. Duke u bazuar në leximet nga këta sensorë, dhe poashtu duke përdorur edhe odometrinë, roboti është në gjendje të bëjë një vlerësim mjaft të besueshëm për të përcaktuar pozitën e tij në mjedisin ku ndodhet. Si rrjedhim i natyrshëm i kësaj problematike është edhe diskutimi mbi algoritmin SLAM. Ky algoritëm mundëson që në të njëjtën kohë të bëhet lokalizim i robotit, por edhe gjenerimi i një harte mbi të cilën tani roboti mund të navigojë. Në rastin e Turtlebot 2i, SLAM algoritmi është i implementuar në të, ashtu që navigimi autonom i këtij roboti veç se është i ndërtuar mbi këtë algoritëm.

Një temë tjetër shumë e rëndësishme për robotikën në përgjithësi, por sidomos për robotët mobil, është sistemi operativ i robotit (ang. ROS). Kemi diskutuar mbi rëndësinë e këtij sistemi, strukturën e tij të ndërtimit dhe veglat e rëndësishme përcjellëse të këtij sistemi. Natyrisht, të gjitha testimet, vizualizmet dhe mapimet janë bërë pikërisht në këtë sistem operativ. Gjatë trajtimit të kësaj problematike, një nga konkludimet e rëndësishme ishte që, mbi të gjitha, ROS ofron një kompatibilitet ndër robotik. Kjo nënkupton që një program i shkruar dhe i testuar në një robot, me shumë pak ndryshime, mund të bartet tek roboti tjetër.

Në pjesën e fundit të punimit kemi diskutuar për programimin e robotëve mobil me Python. Python, duke qenë aktualisht një nga gjuhët programuese me zhvillimin më të hovshëm, ka gjetur një aplikim jashtëzakonisht të madh edhe në pjesën e robotëve mobil. Në fakt, pjesa dërrmuese e librarive bazë të ROS janë të shkruara pikërisht në Python. Për ta konkretizuar këtë diskutim, kemi marrë tre shembuj demonstrues dhe për qëllime testuese, se si mund të shkruhet një program me Python për robotin Turtlebot. Kemi bërë implementin e algoritmeve të kërkimit Dijkstra dhe A*.

9.Referencat

[1]. Autonomous Mobile Tobots and Multi-Robot Systems ; E.Kagan N. Shvalb, I. Ben-Gal , Wiely Publishing 2020

[2]. Introduction to Autonomous Mobile Robots; R. Siegwart, I.R. Nourbakbhsh, D. Scaramuzza; MIT Press 2011

[3]. ROS Programming: Building Powerful Robots; A. Mahtani, L. Sanchez, E. Fernandez, A. Martinez, L. Joseph; Packt Publishin; 2018

[4]. ROS Robotics by Example; C. Farchild, T.L. Harman; Packt Publishing; 2016

[5]. Bazat e Robotikës; A. Pajaziti, Universiteti i Prishtinës, 2002

[6]. Robotics, vision and control; P. Corke; Springer Tracts in Advanced Robotics 2011

[7]. Robot Operating Systems: Volumes 1-6; Springer Publishing; 2021

[8]. Modelling and Control of Wheeled Robots; P.F. Muir; PhD Dissertaion; 1988

[9]. Navigimi i Robotëve Mobil Përmes Identifikatorëve Vizuel; A. Fandaj; Universiteti i Prishtinës - FIM 2022 (temë diplome)

[10]. Lokalizimi dhe krijimi i hartave të lëvizjeve në kohë reale (SLAM) të robotëve mobil duke shfrytëzuar inteligjencën artificale (AI); A. Krrabaj; Universiteti i Prishtinës – FIM 2022 (temë diplome).

[11]. Chaima Bensaci, Youcef Zennir, Denis Pomorski. Nonlinear Control of a differential wheeled mobile robot in real time-Turtlebot 2. International Conference on Advanced Technologies and Electrical Engineering (ICTAEE’18), 2018, Skikda, Algeria. pp.12 - 14. ffhal-02014895ff

[12]. Multi-Robot Path Planning using Disktra’s Algorithm with Multi-Layer Dictionaries; S.A.M. Agbenah, A.A. Jamal, S.A. Fadzli; IJARCCE Publishin 2021