UNIVERSITETI I PRISHTINËS "HASAN PRISHTINA"

Fakulteti i Inxhinierisë Mekanike

Prishtinë

Këshillit të Fakultetit të Inxhinierisë Mekanike

P r i s h t i n ë

Në bazë të vendimit nr. 398 të datës 08/03/2021, të Këshillit të Fakultetit të Inxhinierisë Mekanike në Prishtinë është formuar Komisioni në përbërje:

1. **Prof. Dr. Ramë Likaj, *kryetar***
2. **Prof. Dr. Arbnor Pajaziti, *mentor/anëtar***
3. **Prof. Asis. Dr. Xhevahir Bajrami, *anëtar***

Për vlerësimin e punimit Master me titull: **"LOKALIZIMI DHE KRIJIMI I HARTAVE TË LËVIZJEVE NË KOHË REALE (SLAM) TË ROBOTËVE MOBIL DUKE SHFRYTËZUAR INTELIGJENCËN ARTIFICIALE (AI)"** të kandidatit **Bachelor Albin Krrabaj.**

Pas kontrollimit të punimit të lartpërmendur Komisioni jep këtë:

R A P O R T

Punim Master me titull: **"LOKALIZIMI DHE KRIJIMI I HARTAVE TË LËVIZJEVE NË KOHË REALE (SLAM) TË ROBOTËVE MOBIL DUKE SHFRYTËZUAR INTELIGJENCËN ARTIFICIALE (AI)"** është punuar në 6 (gjashtë) kapituj si dhe është paraqitur me figura dhe ilustrime të nevojshme.

Në Përmbledhje kandidati ka mbuluar komplet procesin prej analizës, modelimit, simulimit dhe në fund aplikimit të procesit SLAM i cili ka për qëllim lokalizimin dhe krijimin e hartave në kohë reale, ku për realizimin e këtij procesi është përdorur roboti mobil Turtlebot 2i. Përgjatë lëvizjes së robotit janë krijuar harta në dimensionet 2D, ku saktësia e lëvizjes është në disa centimetra. Robotit mobil Turtlebot i është caktuar një IP statike. Pas krijimit të hartës për lëvizje atëherë është caktuar një pikë e dëshiruar ku më pas roboti lëvizë deri në atë pikë në mënyrë autonome, nëse përgjatë lëvizjes ndryshon harta p.sh., është vendosur ndonjë objekt i ri, atëherë roboti do të jetë në gjendje të detektojë pengesën e re dhe do të shmanget nga ajo deri sa të arrijë në pikën e caktuar.

Kapitulli i parë paraqet hyrjen ku është bërë shpjegimi i rëndësisë të procesit SLAM, përparësitë ndaj sistemit GPS, aplikimi i gjerë i robotëve mobil kohëve të fundit si dhe pse ROS ka filluar të bëhet një standard i ri në fushën e robotikës.

Kapitulli i dytë shpjegon në detaje procesin e SLAM, tri mënyrat kryesore të implementimit tij. Pastaj janë shpjeguar komponentët kryesore si navigimi, hartat, pozicioni i robotit, costmap, ndjeshmëria, planifikimi dhe realizimi i lëvizjes ku janë sqaruar tre filtra kryesorë që janë përdorur në këtë punim: EKF, AMCL dhe Gmapping.

Kapitulli i tretë shpjegon se çka është ROS, përparësitë, instalimi dhe konfigurimi si dhe është shpjeguar se si krijohet një aplikacion në ROS. Gjithashtu është shpjeguar ndërlidhja ndërmjet Turtlebot dhe ROS, si të ekzekutohet simulatori për Turtlebot, si të konfigurohet një IP statike për ROS dhe Turtlebot.

Kapitulli i katërt paraqet integrimin e procesit të SLAM në ROS, si krijohen hartat OGM dhe si bëhet ruajtja e tyre. Gjithashtu janë shfaqur fajlat konfigurues që përdoren për navigim, AMCL, DWA, Costmap, Gmapping etj.

Kapitulli i pestë paraqet modelimin dhe simulimin e SLAM në dy aplikacione Gazebo dhe Matlab. Në këtë kapituj janë përdorur disa shembuj të marrë nga literatura për të sqaruar më mirë këtë proces, si dhe konfigurimi i komunikimit ndërmjet këtyre dy aplikacioneve për robotin Turtlebot 2i.

Kapitulli i gjashtë paraqet pjesën e fundit të punimit ku është konfiguruar roboti fizik më pas procesi i SLAM është ekzekutuar në mjedis real, është bërë përshkrimi dhe analiza e rezultateve të fituara nga roboti Turtlebot 2i. Duke u bazuar në qëllimin e punimit më pas harta e krijuar është

ruajtur dhe duke përdorur aplikacionin RVIZ i cili është përdorur për vizualizim dhe kryerjen e disa funksioneve për navigim është arritur që të realizohet navigimi autonom i robotit Turtlebot 2i deri në arritjen e pikës së dëshiruar përkatësisht destinacionit, si dhe përparësitë dhe mangësitë që janë vërejtur përgjatë realizimit të këtij procesi.

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: "**LOKALIZIMI DHE KRIJIMI I HARTAVE TË LËVIZJEVE NË KOHË REALE (SLAM) TË ROBOTËVE MOBIL DUKE SHFRYTËZUAR INTELIGJENCËN ARTIFICIALE (AI)**", të kandidatit **Bachelor Albin Krrabaj**, ta aprovojë si punim për Master, dhe ta jepë në diskutim publik.

Prishtinë, Mars 2022

Komisioni:

1. Prof. Dr. Ramë Likaj, *kryetar*_____
2. Prof. Dr. Arbnor Pajaziti, *mentor/anëtar* _____
3. Prof. Asis. Dr. Xhevahir Bajrami, *anëtar*_____

UNIVERSITETI I PRISHTINËS "HASAN PRISHTINA"
FAKULTETI I INXHINIERISË MEKANIKE
Departamenti: MEKATRONIKË

PUNIM DIPLOME - MASTER

LOKALIZIMI DHE KRIJIMI I HARTAVE TË LËVIZJEVE NË KOHË REALE (SLAM) TË ROBOTËVE MOBIL DUKE SHFRYTËZUAR INTELIGJENCËN ARTIFICIALE (AI)

Mentori:
Prof.Dr. ARBNOR PAJAZITI

Kandidati:
Bsc.ALBIN KRRABAJ

Prishtinë,2022

PËRMBAJTJA

FALENDËRIM

LISTA E FIGURAVE

LISTA E SHKURTESAVE

ROS – Robot Operating System

GPS – Global Positioning System

AI – Artificial Intelligence

PC – Personal Computer

SLAM - Simultaneous Localization And Mapping

EKF - Extended Kalman Filter

IMU - Inertial Measurement Unit

LDS – Light Distance Sensor

LRF – Laser Range Finder

MCL - Monte Carlo Localization

AMCL - Adaptive Monte Carlo Localization

SIR - Sampling Importance Re-sampling

IoT - Internet of Things

IDE - Integrated Development Environment

TCP/IP – Transmission Control Protocol/Internet Protocol

DWA - Dynamic Window Approach

ABSTRAKT

Marrë parasysh rëndësinë e robotëve mobil në ditët e sotme si dhe aplikimin e tyre nëpër ambiente të mbyllura si robotët mobil për pastrim, informim, argëtim etj. kam vendosur të bëjë një hulumtim lidhur me mënyrën se si robotët mobil navigojnë nëpër këto terrene.

Sfida kryesore që shfaqet gjatë navigimit nëpër terrene të mbyllura është detektimi i pengesave, pra nevojitet që të behet matja e distancës në mënyrë shumë precize, kështu do të bëhet detektimi i këtyre pengesave duke shfrytëzuar sensor Lidar i cili skanon në mënyrë kontinuale si dhe në momentin që shfaqet ndonjë pengesë e re atëherë do të informojë robotin më pas robot në bazë të algoritmeve të ndryshëm kryen veprime të caktuara, zakonisht mënjanimi i tyre duke gjetur hapësire të re për lëvizje deri në arritjen e cakut.

Në këtë punim fillimisht do të bëhet modelimi dhe simulimi i lëvizjeve të robotit mobil Turtlebot duke shfrytëzuar sistemin operativ ROS (Robot Operating System), duke u bazuar se ROS është një platformë me burim të hapur (open-source) ekzistojnë shumë aplikacione që shfrytëzohen për modelim dhe simulim për robotë të ndryshëm por në këtë punim do të shfrytëzohen Gazebo, Rviz dhe Matlab. Pas modelimit dhe simulimit në këto aplikacione do të bëhet aplikimi i tyre në robotë fizik si dhe krahasimi i rezultateve të fituara.

1 HYRJE

Sikur njerëzit ashtu edhe robotët kanë nevojë për harta për të lëvizur. Sa do që janë zhvilluar sistemet e pozicionimit në hapësirë (GPS), në hapësira të brendshme robotët nuk mund të mbështeten në këtë sistem, kjo për shkak të saktësisë së matjes së këtyre sistemeve kur dihet që në shumicën e rasteve për lëvizjen në hapësira të mbyllura kërkohet saktësi prej disa cm. Kështu është shfaqur nevoja për zhvillimin e algoritmeve të ndryshme në mënyrë që të krijohen harta të lëvizjes në mënyrë kontiunale të këtyre robotëve, veçanërisht robotëve mobilë dhe atyre humaniodë. Kjo metodë ua mundëson robotëve të dinë pozicionin e tyre duke perceptuar mjedisin përmes sensorëve të ndryshëm të brendshëm dhe të jashtëm. Në parim kjo mund të duket mjaft e lehtë, por në fakt është një proces me shumë faza që përfshin krijimin e shumë algoritmeve për bashkëveprimin e të dhënave që të arrihet lëvizja në mjedis. Ndërtimi dhe testimi i aplikacioneve robotike duke simuluar mjediset reale janë detyra të vështira dhe të komplikuara [26]. Zhvilluesit duhet të marrin parasysh skenarë të pafundmë të situatave të botës reale, përpos asaj një sfidë mjaft e madhe për ta është qasja në një robot fizik, kjo ndoshta është edhe arsye së përse ka pak zhvillues të robotikës, ku përpos zhvillimit të aplikacioneve nevojitet edhe njohuri në fushën e elektronikës, mekanikës, automatikës, sensorikës dhe IA-së. Ka shumë raste kur zhvilluesit e aplikacioneve që kalojnë në programimin në robotikë fillimisht besuan që kishin bërë gabim në kodim, por përgjatë hulumtimit kanë vërejtur së ishin pjesët harduerike të cilat nuk punonin si duhet ose nuk punonin fare. Në këtë rast u shfaq nevoja për krijimin e një sistemi operativ në fushën e robotikës (Robot Operating System – ROS) falë këtij sistemi tani më zhvilluesit mund të programojnë lirshëm një robot në mjedis virtual të shkëputur komplet nga mjedisi real dhe sfidat që shfaqen në problemet harduerike. ROS po bëhet një standard i ri në programimin e robotikës. Fillimisht hyri në përdorim nëpër universitete për qëllime studimore, por u përhap shumë shpejt në botën e biznesit. Më herët secili robot kishte aplikacionin e vet të prodhuesit, që do të thotë që nëse ndryshoni modelin e robotit ose prodhuesin duhet të ndërroni edhe aplikacionin. Situata ishte e ngjashme me atë të kompjuterëve në vitet '80, kur çdo kompjuter kishte sistemin e vet operativ dhe ju duhej të krijonin të njëjtin program për çdo lloj kompjuteri. ROS sistem operativ me burim të hapur (angl. open source) për robotë siç është Linux për PC, ose iOS dhe Android për telefona.

Edhe pse zhvillimi i robotëve mobilë nuk ka arritur që të jetë komplet autonom shumë robotë mobil mund të kryejnë punë të ndryshëm me shumë pak interaksion me njeriun, shembull në spitale mjekët dhe motrat medicionale kryejnë shumë lëvizje për sigurimin e mjeteve, bartjen e pacienteve, bartjen e ushqimit, dezinfektimin e objektit etj. robotët mobilë kanë filluar që shumë prej këtyre proceseve t'i zëvendësojnë me përpikëri. Spitalet janë vetëm një prej shumë hapësirave të brendshme ku këta robotë po kryejnë punë, kështu ne fushën e hotelerisë janë krijuar gjithashtu shumë robotë mobilë ku kryejnë funksione si mikpritja e mysafirëve, pastrimi, informimi etj. Gjithashtu aplikim mjaft të lartë kanë filluar të kenë edhe robotë mobil për pastrim me vakum nëpër shtëpia private. Një robot mobil i quajtur Kiva ka krijuar një funksion jetik për gjigandin amerikan Amazon, ky robot bën bartjen e produkteve të ndryshme nëpër depo, ku më herët nevojiten orë për bartjen këtyre produkteve tash është çështje minutash, por në këtë rast nevojitet që të modifikohet hapësira për këta robotë, kështu ende mbetet sfiduese navgimi nëpër ambiente të panjohura ose të njohura po ku kemi objekte dinamike. Me rritjen e aplikimit të tyre shfaqet nevoja për avancimin e navigimit të këtyre robotëve edhe më tutje ku rol kyç luan saktësia e lëvizjes nëpër këto ambiente, kjo pasi që duhet procesuar sasi të mëdha të informacioneve të marra nga ambienti që rrethohen këta robotë informata të cilat sigurohen nga sensorë të ndryshëm si dhe parashikimi i situatave të ndryshme në mënyrë që roboti të reagojë në mënyrë të sigurte.

Fig 1. 1 Amazon Kiva

2 SLAM

SLAM është akronim në gjuhën angleze (SLAM - Simultaneous Localization And Mapping), i cili ka për qëllim lokalizimin dhe krijimin e hartave të lëvizjeve në kohë reale zakonisht të robotëve mobil duke shfrytëzuar inteligjencën artificiale (AI – Artificial Intelligence). SLAM mund të implementohet në shumë mënyra kështu shpesh herë konsiderohet si një koncept ose fushë studimi sesa vetëm një algoritëm, kjo pasi që ka edhe shumë algoritme të tjera që përdoren për realizimin e tij ku SLAM duhet të kujdeset në perceptimin e mjedisit që e rrethon, ndërlidhja e të dhënave, planifikimin e lëvizjes së ardhshme, arritja e lëvizjes së dëshiruar dhe krejt në fund përditësim i hartës së mjedisit përkatësisht lëvizjes. Lëvizjet mund të realizohen në 2D dhe 3D mjedise. Tek robotët mobil gjatë implementimit të SLAM është shumë me rendësi performanca odometrike e cila realizon lëvizjen e robotit prej pozitës aktuale deri në pozicionin e ardhshëm, performancë e cila arrihet përmes sensorëve të lëvizjes ku sa më i avancuar është sensori dhe aktuatori aq më të mira janë rezultatet pasi që roboti nuk mund të këtë margjinë të gabimit më të madhe se 2cm për distancë të lëvizjes prej 1m dhe gabim në shkallë të rrotullimit më shumë 2^{o} për rrotullime deri në 45^{o}.

SLAM është një proces i cili kalon nëpër disa hapa, ku qëllimi kryesor është që të detektojë mjedisin që e rrethon robotin dhe pozicioni ku ndodhet. Marrë parasysh mundësinë e gabimit të matjeve odometrike atëherë shfaqet nevoja për shfrytëzimin e sensorëve laserik për evitimin e gabimeve duke ripozicionuar robotin. Kjo mund të arrihet duke nxjerrë veçori nga mjedisi dhe duke e moniturar lëvizjen e robotit. Pjesa kryesore e SLAM është EKF (Extended Kalman Filter) e cila është përgjegjëse për përditësimin e informatave të informatave mbi lokacionin e robotit duke u bazuar në matje odometrike dhe laserike. EKF përbëhet nga dy hapa llogaritës, domethënë korrigjimi dhe parashikimi. Në fig.2.0 është paraqitur një përshkrim e procesit të SLAM [3].

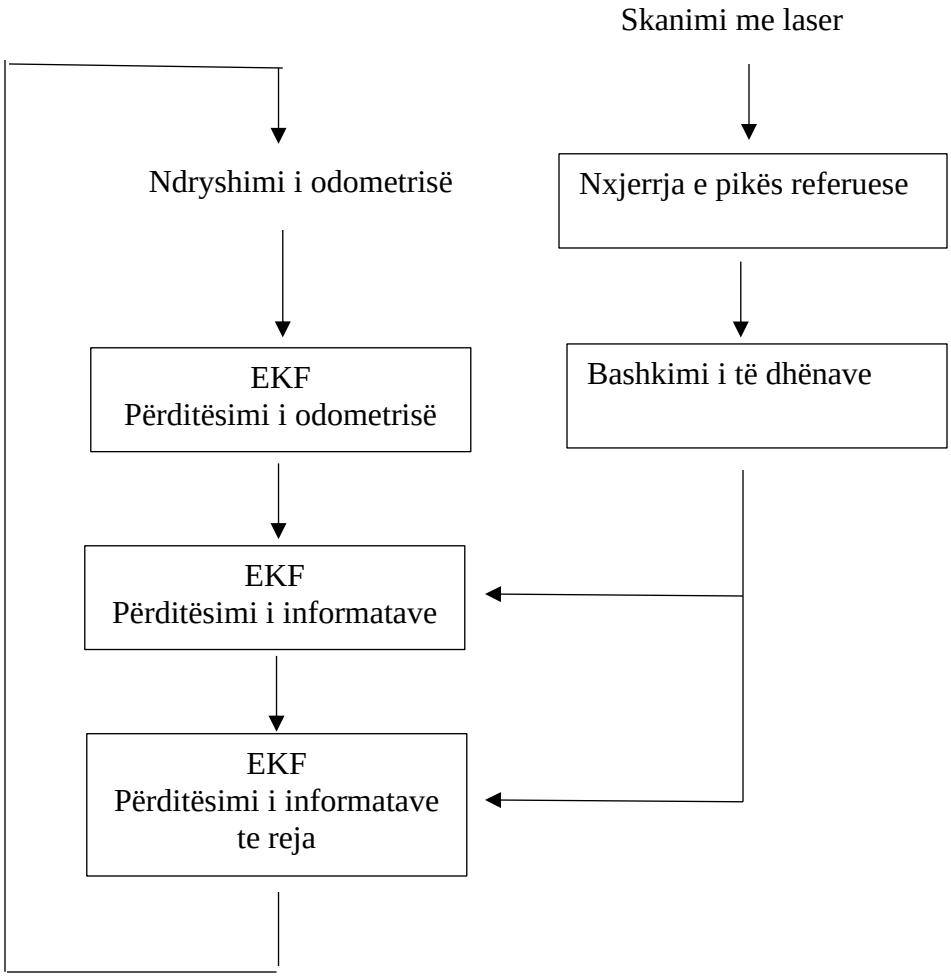

Fig 2. 1 Përshkrimi i procesit të SLAM

2.1 Tri mënyrat kryesore të implementimit të SLAM

Në parim SLAM është i varur nga mjedisi që e rrethon ku bëhet skenimi i tij dhe matja e distancës të objekteve fizike që e rrethon, dhe varësisht nga mënyra e detektimit të mjedisit dhe sensorëve që përdoren SLAM ndahet kryesisht në tre implementime kryesore [3].

Mënyra e parë është duke përdorur sensorët laserik të cilët janë mjaft preciz, efecientë dhe nuk kërkojnë fuqi kompjuterike të lartë për procesim. Zakonisht shfaqen probleme në detektimin e objekteve me sipërfaqe specifike p.sh xhamit pasi që shfaq të dhëna të lexuara keq, gjithashtu

nëse roboti mobil ku implementohet do të përdoret në mjedis nënujor atëherë saktësia e matjes reduktohet për shkak të ujit.

Mënyra e dytë është duke përdor sensorët ultrasonik(sonar), të cilët janë mjaft të lirë krahasuar me sensorët laserik por që saktësia e matjes është më e vogël por që në mjedise nënujorë mbetet njëri ndër sensorët më të mirë për implementim.

Mënyra e tretë është duke përdorur video-procesim ku kohëve të fundit në robotikë ka gjetur një aplikim mjaft të madh për shkak të mundësive që ofrojnë dhe kostove ekonomike. Kështu p.sh sensorët laserik dhe ultrasonik (sonar) edhe pse mundësojnë këtë implementim mënyra e shfaqjes së rezultateve të lexuara nuk është më e mirë se sa me video-procesim kjo pasi që na mundëson pamjen sikurse njeriu që e sheh, pa marrë parasysh ambientit në të cilin ndodhet video-procesimi është i varur shumë nga ndriçimi në mjedis p.sh nëse jemi duke skanuar në një dhomë e cila nuk ka ndriçim atëherë është e pamundur arritja e rezultateve për dallim nga sensorët e mëhershëm.

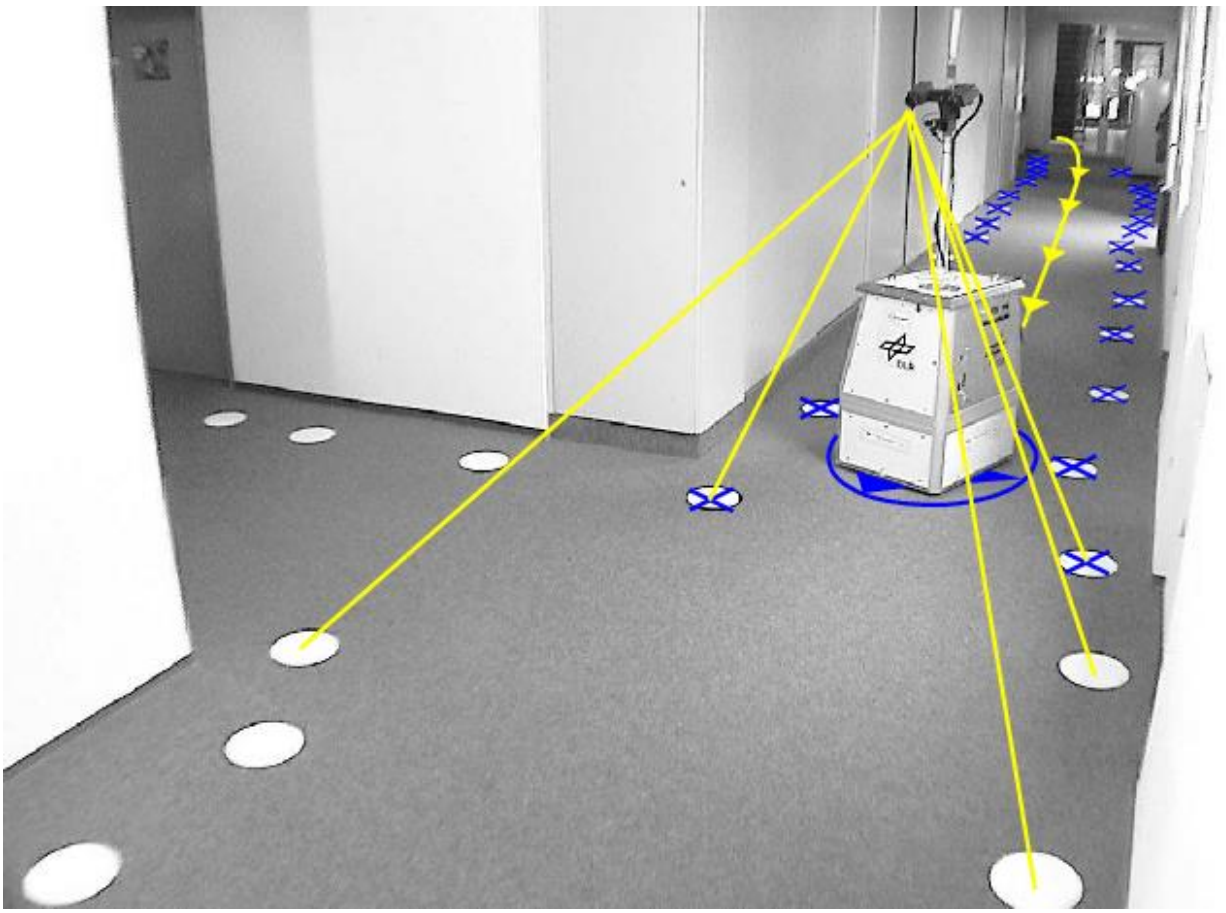

Fig 2. 2 Pamja e një roboti mobil gjatë realizimit të SLAM

2.2 Navigimi dhe komponentët tjera të SLAM

Nëse vendosim një destinacion në pajisjen e navigimit, navigimi ju lejon të kontrolloni distancën dhe kohën e udhëtimit nga vendndodhja aktuale deri në destinacion dhe mund të vendosën preferenca specifike dhe informacione të tilla si vendet për të ndaluar dhe rrugët e preferuara përgjatë rrugës. Sistemi i navigimit ka një histori relativisht të shkurtër. Në vitin 1981, prodhuesi japonez i makinave Honda së pari propozoi një sistem analog të bazuar në një xhiroskop me tre boshte dhe një hartë filmi të quajtur "Elektro Gryrocator1". Më pas Etak Navigator2, i cili funksionoi si një sistem navigimi elektronik me busull elektronike dhe sensorë të bashkangjitur në rrota që u prezantua nga panairin e automobilave në SHBA nga kompania e furnizimit Etak. Sidoqoftë montimi i sensorit dhe busullës elektronike në makinë ishte një sfidë e madhe për çmimet e automobilave dhe kishte probleme me besueshmërinë e sistemit të navigimit. Që nga vitet 1970 Shtetet e Bashkuara zhvilluan sisteme të pozicionimit satelitor për ushtrinë dhe në vitet 2000 24 satelitë GPS3 u bënë të disponueshëm për qëllime të përgjithshme dhe sistemet e navigimit të bazuara në GPS filluan që të përdorin këta satelitë.

2.2.1 Navigimi i robotëve mobil

Le të kthehemi te robotët tani. Baza dhe pika kryesore e një roboti mobil është pa dyshim navigacioni. Navigimi në robotikë është i pandashëm dhe thelbësor. Kjo është lëvizja e robotit drejt një destinacioni të përcaktuar, i cili nuk është aq i lehtë sa duket. Por është e rëndësishme të dihet ku ndodhet vetë roboti dhe të ketë një hartë të mjedisit të caktuar. Është gjithashtu e rëndësishme për të gjetur itinerarin e optimizuar midis opsioneve të ndryshme të rrugëtimit dhe për të shmangur pengesat e ndryshme fizike.

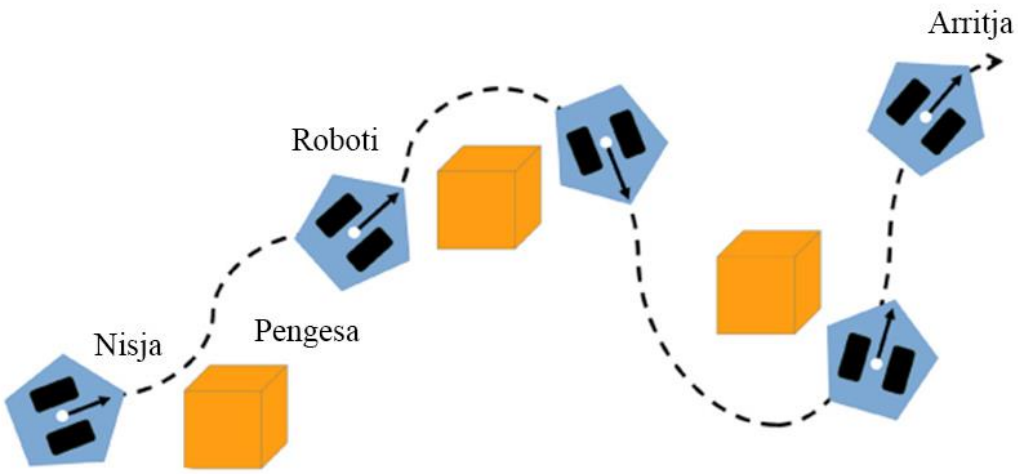

Fig 2. 3 Navigimi i robotit mobil përgjatë pengesave

Pra gjatë navigimit po e shohim se edhe pse ekzistojnë algoritme të ndryshme për navigim na nevojiten katër veçori apo funksione [11]:

1. Harta
2. Pozicioni i robotit
3. Ndjeshmëria (detektimin i ambientit)
4. Llogaritja e rrugës dhe lëvizja nëpër të

2.2.2 Hartat

Karakteristika e parë thelbësore për navigimin është harta. Sistemi i navigimit është i pajisur me një hartë shumë të saktë që nga momenti i blerjes dhe harta e modifikuar mund të shkarkohet periodikisht në mënyrë që të arrihet në destinacion sa më saktë. Por jo çdo hartë është e disponueshme p.sh dhoma ku do të vendoset roboti? Në këtë rast për navigim, roboti ka nevojë për një hartë, kështu që ne duhet të krijojmë një hartë dhe t'ia japim robotit ose roboti duhet të jetë në gjendje krijoni një hartë vetë. SLAM4 është zhvilluar për të lejuar robotin të krijojë një hartë me ose pa ndihmën e njeriut.

2.2.3 Pozicioni i robotit

Roboti duhet të jetë në gjendje të masë dhe vlerësojë pozicionin e tij (pozicioni + orientimi). Në rastin e një makine, GPS përdoret për të vlerësuar pozicionin e tij. Megjithatë GPS nuk mund të përdoret brenda edhe nëse mund të përdoret një GPS për shkak të gabimeve të mëdha nuk mund të përdoret për robotët. Në ditët e sotme përdoret një sistem me precizitet të lartë si DGPS5 por ky është gjithashtu i padobishëm në ambiente të mbyllura. Për të kapërcyer këtë problem përdoren metoda të ndryshme. Aktualisht, metoda më e përdorura në ambiente të brendshme është vlerësimi i pozicionit përmes llogaritjes së vdekur (dead reckoning) e cila është përdorur për një kohë të gjatë dhe përbëhet nga sensorë me kosto të ulët dhe arrin një rezultat të saktë të vlerësimit të pozicionit. Distanca e lëvizjes së robotit matet me rrotullimin e rrotës megjithatë ka një gabim në llogaritje me distancën e rrotullimit të rrotave dhe distancën aktuale të udhëtimit. Në këtë rast shfaqet nevoja për përdorimin e një sensori të matjes së inercisë (IMU) për të reduktuar gabimin duke kompensuar gabimin e pozicionit dhe orientimi ndërmjet vlerës së llogaritur dhe vlerës aktuale.

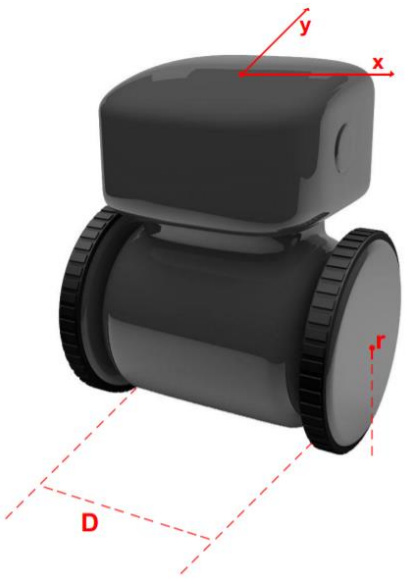

Fig 2. 4 Variablat e kërkuara për llogaritjen e vdekur (dead reckoning)

Për të bërë llogaritjen e vdekur apo dead reckoning kur kemi të bëjmë me robotët mobilë si në fig.2.3 atëherë kemi distanca D ndërmjet rrotave dhe r rrezja e rrotës. Supozojmë se roboti ka lëvizur në një distancë shumë të shkurtër përgjatë kohës T_e, shpejtësia e rrotullimit të motorit në rrota (v_l,v_r) ku v_l shpejtësia e rrotullimit të motorit në rrotën e majtë dhe v_r shpejtësia e rrotullimit

të motorit në rrotën e djathtë si dhe duke krahasuar edhe diferencën në shpejtësinë e rrotullimit të motorëve prej vlerës aktuale $E_l c$, $E_r c$ me atë të kaluar $E_l p$, $E_r p$ llogaritet duke u bazuar në ekuacionin 2.1 dhe 2.2

$$v_l = \frac{(E_l c - E_l p)}{T_e} \times \frac{\pi}{180}\ rad/_{sec} \dots \quad (2.1)$$

$$v_r = \frac{(E_r c - E_r p)}{T_e} \times \frac{\pi}{180}\ rad/_{sec} \dots \quad (2.2)$$

Ekuacionet 2.3 dhe 2.4 shfrytëzohen për te llogaritur shpejtësinë për rrotën e majtë dhe të djathtë (V_l, V_r). Pastaj prej tyre mund të llogarisim shpejtësinë lineare (v_t) dhe shpejtësinë këndore (ω_t) siç tregojnë ekuacionet 2.5 dhe 2.6

$$V_l = v_l \times r\ ^{meter}/_{sec} \dots \quad (2.3)$$

$$V_r = v_r \times r\ ^{meter}/_{sec} \dots \quad (2.4)$$

$$v_t = \frac{(V_r + V_l)}{2} meter/_{sec} \dots \quad (2.5)$$

$$\omega_t = \frac{(V_r + V_l)}{D} rad/_{sec} \dots \quad (2.6)$$

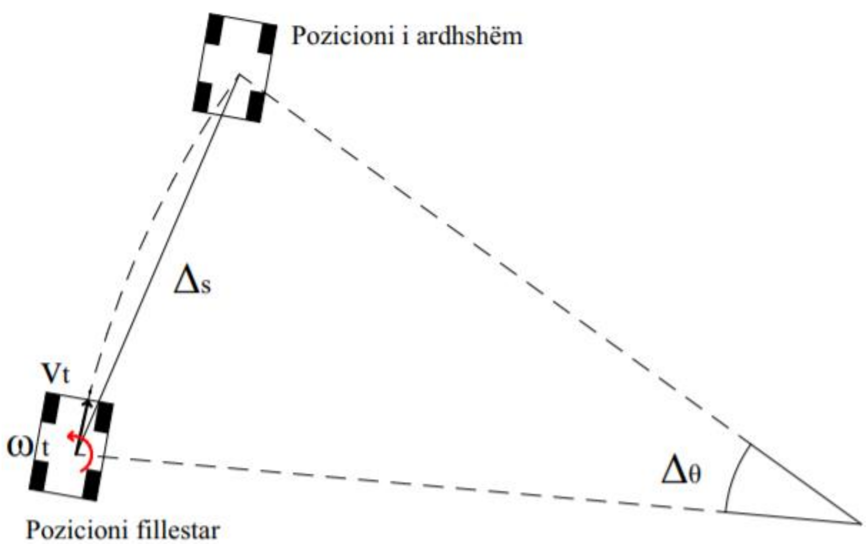

Fig 2. 5 Llogaritja e këndit të vdekur (Dead Reckoning)

Në fund duke përdorur vlerat e llogaritura më lartë, tani mund të llogarisim pozicionin ($x_{(t+1)}$,$y_{(t+1)}$) dhe orientimin ($\theta_{(t+1)}$) duke shfrytëzuar ekuacionet prej 2.7 deri 2.10.

$$\Delta s = v_t T_e; \quad \Delta\theta = \omega_t T_e \ldots\ldots\ldots (2.7)$$

$$x_{(t+1)} = x_t + \Delta s \cos\left(\theta_t + \frac{\Delta\theta}{2}\right) \ldots\ldots\ldots (2.8)$$

$$y_{(t+1)} = y_t + \Delta s si\, n\left(\theta_t + \frac{\Delta\theta}{2}\right) \ldots\ldots\ldots (2.9)$$

$$\theta_{(t+1)} = \theta_t + \Delta\theta \ldots\ldots\ldots (2.10)$$

2.2.3.1 Costmap

Më lartë treguam se si llogaritet odometria e robotit ku parimisht kemi një vlerë të matur të rrotullimit të rrotës e që në robotikë na ofrohet nga enkoderët e motorëve si dhe vlera tjetër e ofruar nga sensori IMU. Por na nevojitet që te kemi edhe një sensor që mat distancës ndërmjet robotit dhe pengesave, kështu gjatë navigimit të SLAM nevojitet të llogaritën katër faktorë [11]:

1. Harta statike
2. Zona e lirë për lëvizje,
3. Zona e mbyllur për lëvizje,
4. Zona e panjohur

Duke u bazuar në këta katër faktorë costmap bën llogaritjen e zonave ku ka pengesa, zonave ku mund të përplaset roboti si dhe lëvizjen e robotit. Ky shprehet si një vlerë prej 0-255 ku:

- 000 – paraqet zonën ku roboti mund të lëviz lirshëm
- 001~127 – paraqet zonën më probabilitet të vogël të përplasjes
- 128~252 – paraqet zonën më probabilitet të lartë të përplasjes
- 253~254 – zona e përplasjes
- 255 – paraqet zonën e ndaluar për lëvizje

2.2.4 Ndjeshmëria

Për të kuptuar nëse ka pengesa si mure dhe objekte atëherë kërkohen sensorë. Përdoren lloje të ndryshme sensorësh si sensorë të distancës dhe sensorë të shikimit. Sensori distancës përdor sensorë me bazë laseri si (LDS, LRF, LiDAR), sensorë ultrasonik dhe infra të kuqe. Sensori i shikimit ose vizionit përfshin kamera stereo, monokamera, omnidirectional kamerat, dhe së fundmi RealSense, Kinect, Xtion, të cilat përdoren gjerësisht si kamera të avancuara këto përdoret për të identifikuar pengesat. Por secili nga këta sensorë kanë përparësitë dhe mangësitë e veta. Më poshtë do të analizojmë dallimet kryesore ndërmjet kamerave dhe LiDAR sensorëve.

2.2.4.1 LiDAR Sensorët

LiDAR nënkuptojnë pajisje për matjen e distancës e bazuar në principin e punës së radarit por që përdor edhe dritën laserike për dallim nga radari tradicional i cili përdore valët e radios. Ky lëshon me miliona pulse elektromagnetike të dritës për të zbuluar objektet afër në atë mënyrë që të llogaritë fuqinë e kthyer në marrës për fuqinë laserike të transmetuar. Ky sistem duhet të jetë mjaft mirë i optimizuar dhe i saktë në mënyrë që koha e reaksionit të jetë më e mirë sesa e njeriut. Transmetimi mjaft i shpejtë i të dhënave të LiDAR-ëve mundëson që robotët mobil të kenë mjaftë kohë të reagojnë ndaj ndryshimeve të shfaqura në mjedis.

Përparësitë:

- Precizitet i lartë
- Laserët nuk i pengojnë përgjatë punës hijet, rrezet diellit ose drita të tjera
- Nevojitet fuqi kompjuterike mjaftë e ulët për procesim

Mangësitë:

- Interferenca nga LiDAR-ët tjerë
- Ndryshimet klimatike vështirë të dallueshme (shi,borë,mjegull)
- Identifikimi i teksteve, ngjyrave, formave
- Kosto e lartë

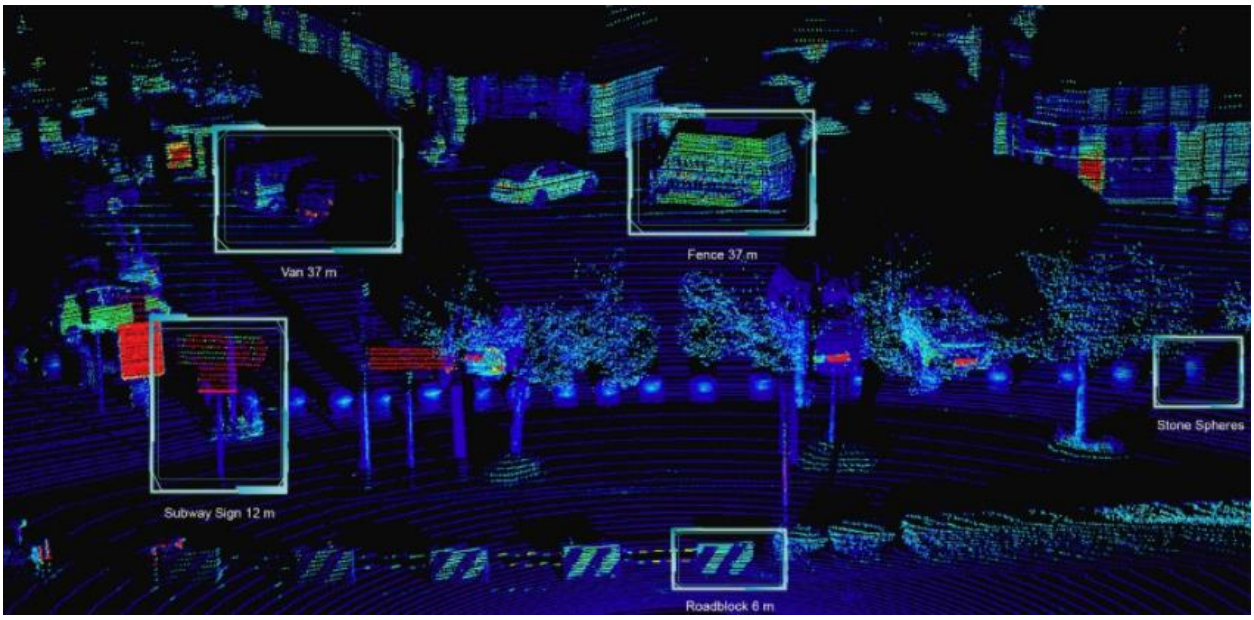

Fig 2. 6 Skanimi me LiDAR

2.2.4.2 Kamerat

Kamerat e avancuara sidomos ato 3D kanë gjetur një aplikim mjaft të lartë në fushën e robotikës në përgjithësi por gjithashtu në robotët mobil, kryesisht tek robotët mobilë përdoren për hartimin 3D dhe navigimin 3D. Ekzistojnë shumë arsye pse kamerat për dallim prej LiDAR-ëve kanë gjetur aplikim më të lartë më poshtë do të veçojmë disa përparësi edhe mangësi të tyre.

Përparësitë:

- Kamerat mund të perceptojnë mjedisin njëjtë si njeriu
- Mund të dallojnë kushtet atmosferike si bora, shiu, mjegulla
- Integrim i lehtë
- Kosto e ulët e prodhimit dhe mirëmbajtjes

Mangësitë:

- Nëse nuk ka ndriçim të mjaftueshëm ose shumë të lartë vështirë të dallojë objektet
- Matja e distancës nuk është shumë e saktë
- Fuqi e lartë kompjuterike për procesim

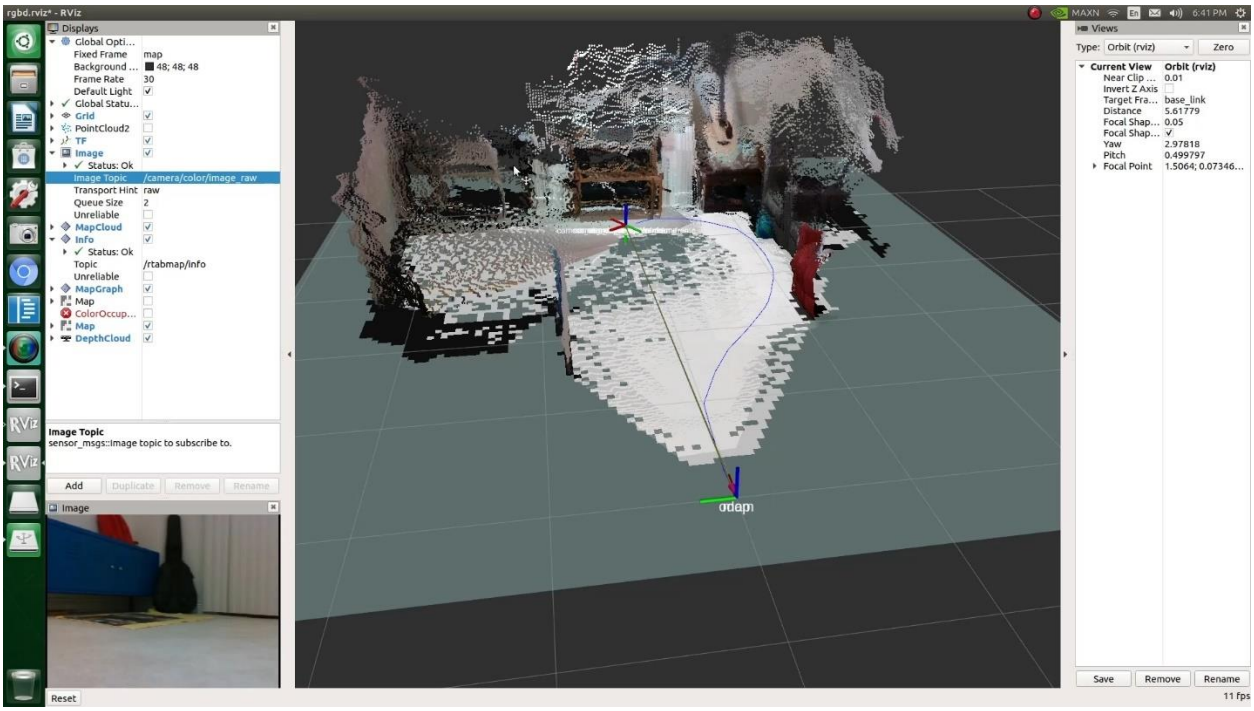

Fig 2. 7 Skanimi me kamera 3D

2.2.4.3 Sensor Fusion

Më lartë sqaruam disa përparësitë dhe mangësitë ndërmjet LiDAR-ëve dhe kamerave por në kohët e fundit është duke u përdorur kombinimi i këtyre dy teknologjive. Por bashkimi i këtyre ka në vetë një sfidë mjaft të madhe harmonizim e të dhënave ku secili input që vjen pa marrë parasysh nga LiDAR-ët apo kamerat duhet programuar që interpretimi i të dhënave të jetë më saktësi milisekonda. Zakonisht shumë sisteme të avancuara tek automjetet janë dukë përdorur këto dy teknologji më qëllim të vozitjes autonome e kohëve të fundit edhe tek robotët mobilë dhe ata manipulatorë.

2.2.5 Planifikimi dhe realizimi i lëvizjes së ardhshme

Karakteristika e fundit thelbësore për navigimin është llogaritja dhe udhëtimi i rrugës optimale për në destinacion. Ky quhet kërkimi dhe planifikimi i rrugës dhe ka shumë algoritme që funksionojnë si p.sh. Gmapping, Extended Kalman Filter (EKF), algoritmi A*, fusha e mundshme, filtri i grimcave etj. Më poshtë do të shpjegojmë më gjerësisht tre filtrat kryesor që përdoren për SLAM.

2.2.5.1 Extended Kalman Filter (EKF)

EKF janë filtra mjaft të zakonshëm në botën reale prej veturave autonome deri tek dronet [25]. Zakonisht përdoren kur sensorët e ndryshëm nuk përçojnë sinjale të pastërta gjë e cila zvogëlon saktësinë e matjes për këtë gjatë aplikimit të EKF korigjojmë matjet e sensorit dhe kalkulojmë më mirë pozicionin e robotit përgjatë kohës që lëviz roboti. Pra EKF është një algoritëm që shfrytëzon njohuritë tona për fizikën e lëvizjes së sistemit d.m.th modelin e hapësirës së gjendjes për të bërë rregullime të vogla në matjet aktuale të sensorit për të zvogëluar sasinë e zhurmës dhe si rezultat arrijmë një vlerësim më të mirë të gjendjes të një roboti.

Si rast studimi marrim një robot mobil si në *fig 2.7*. Për të përshkruar më mirë këtë robotë na nevojitet që të kalkulojmë hapësirat e gjendjes që lëvizjen e robotit të përkthejmë në ekuacione matematikore prej një hapi deri në hapin e ardhshëm.

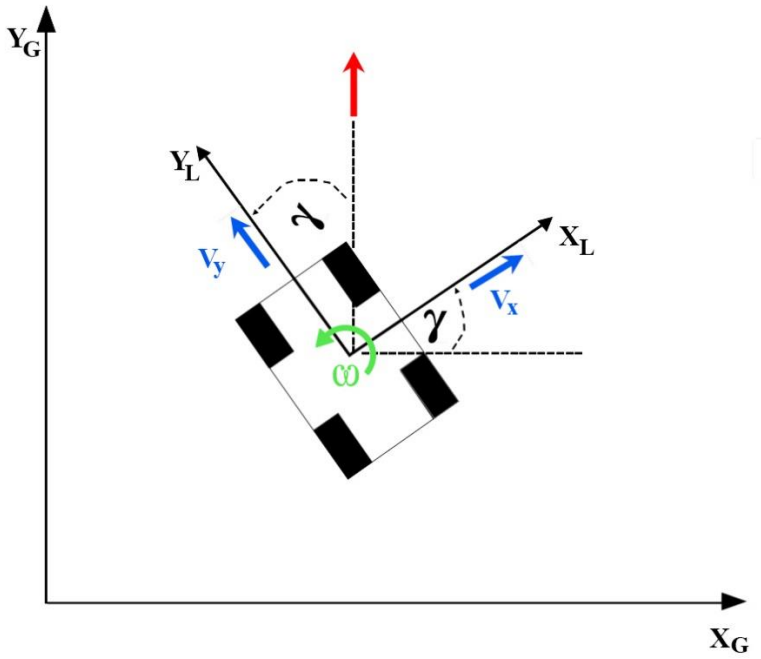

Fig 2. 8 Roboti mobil përgjatë boshtit koordinativ

Ku nga *fig 2.7* kemi:

γ – përshkruan rrotullimin përgjatë boshtit z

ω – shpejtësia këndore përgjatë boshtit z

v – shpejtësia e robotit

v_x – shpejtësia lineare përgjatë aksit x $\boldsymbol{v_x = v\cos(\gamma)}$

v_y – shpejtësia lineare përgjatë aksit y $\boldsymbol{v_y = v\sin(\gamma)}$

Le të supozojmë se roboti është në gjendje qetësie vektori për gjendjen e mëparshme të jetë për kohën *t-1* dhe se lëvizja e orientuar e robotit është përgjatë boshtit *x* atëherë kemi:

$$\hat{x}_{t-1|t-1} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \gamma_{t-1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.11)$$

Në këtë rast nëse roboti lëviz për një kohë prej d_t atëherë distanca është e barabartë më shpejtësinë e kaluar përgjatë kësaj kohe kështu kemi:

$$\begin{bmatrix} x_t \\ y_t \\ \gamma_t \end{bmatrix} = \begin{bmatrix} x_{t-1} + \vartheta_{t-1} cos\gamma_{t-1} * \ d_t \\ y_{t-1} + \vartheta_{t-1} sin\gamma_{t-1} * \ d_t \\ \gamma_{t-1} + \omega_{t-1} * \ d_t \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.12)$$

Përmes ekuacionit 2.12 arritëm të përshkruajmë lëvizjen e robotit përmes ekuacioneve matematikore por ky ekuacion është jo linear dhe gjatë kalkulimeve për EKF nevojitet që të bëjmë këtë ekuacion linear kështu ekuacioni linear për robotin mobil do të jetë:

$$x_t = A_{t-1} x_{t-1} + B_{t-1} u_{t-1} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.13)$$

ku:

- x_t – vektori i gjendjes aktuale $[x_t, y_t, \gamma_t]$
- x_{t-1} – gjendja paraprake e robotit për $[x_{t-1}, y_{t-1}, \gamma_{t-1}]$
- u_{t-1} – paraqet vektorin e vlerave hyrëse(inputeve) për kohën *t-1* $[\vartheta_{t-1}, \omega_{t-1}]$

Në këtë rast forma e plotë e ekuacionit për hapësirat e gjendjes është:

$$\begin{bmatrix} x_t \\ y_t \\ \gamma_t \end{bmatrix} = A_{t-1} \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \gamma_{t-1} \end{bmatrix} + B_{t-1} \begin{bmatrix} \vartheta_{t-1} \\ \omega_{t-1} \end{bmatrix} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.14)$$

Matrica A paraqet gjendjen e sistemit përkatësisht se si gjendja e sistemit ose pozicionet [x,y, γ] ndryshojnë nga koha paraprake k-1 në k kur nuk kemi asnjë vlerë hyrëse në këtë rast shpejtësi ose nxitim këndor të robotit. Pra numri i rreshtave dhe kolonave është i barabartë me numrin e hapësirave të gjendjeve 3x3. Kështu matrica A pas transformimeve Jakobiane dhe derivimit të matricës do të jetë e barabartë me:

$$A_{t-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.15)$$

Matrica B në rastin tonë është një matrice 3x2 e cila ka numrin e rreshtave të barabartë më numrin e gjendjeve dhe numrin e kolonave të barabartë me numrin e inputeve. Kjo përshkruan se si ndryshon gjendja e sistemit [x,y, γ] prej kohës t-1 në k gjatë vlerave hyrëse në këtë rast shpejtësisë v dhe shpejtësisë këndore ω. Pas transformimit Jakobian kemi:

$$B_{t-1} = \begin{bmatrix} cos\gamma_{t-1} * \ d_t & 0 \\ sin\gamma_{t-1} * \ d_t & 0 \\ 0 & d_t \end{bmatrix} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.16)$$

Gjatë kalkulimeve të EKF shpesh ndodh që robotët mobil nuk reagojnë ashtu siç është kalkuluar në këtë rast nevojitet të shtohet një zhurmë(sensorët) n_{t-1} për të llogaritur gabimin kjo për shkak që sinjalet hyrëse janë të papastër. Kështu ekuacioni final për hapësirat e gjendjes së robotit është:

$$\begin{bmatrix} x_t \\ y_t \\ \gamma_t \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \gamma_{t-1} \end{bmatrix} + \begin{bmatrix} cos\gamma_{t-1} * \ d_t & 0 \\ sin\gamma_{t-1} * \ d_t & 0 \\ 0 & d_t \end{bmatrix} \begin{bmatrix} \vartheta_{t-1} \\ \omega_{t-1} \end{bmatrix} + \begin{bmatrix} n_{t-1} \\ n_{t-1} \\ n_{t-1} \end{bmatrix} \ldots\ldots\ldots\ldots\ldots (2.17)$$

ku:

v_{t-1} – shpejtësia paraprake në kohën t-1

ω_{t-1} – shpejtësia këndore rreth boshtit z në kohën t-1

Më lartë sqaruam se si të llogarisim hapësirat e gjendjes së robotit për kohën k por e gjitha është e bazuar në parashikim. Ky parashikim nuk mund të jetë 100% i saktë kështu na duhet të parashikojmë kovariancën (kovarianca është një masë që tregon shkallën në të cilën ndryshojnë së bashku dy variabla të rastit) e matricës $P_{t|t-1}$ përgjatë kohës k, kjo matricë është katrore dhe në rastin tonë është 3x3 dhe në diagonale ka variancat kurse jashtë diagonales kovariancat pra kryesisht përdoret për permisimin e vlerës të saktësisë të gjendjes. P.sh për rastin tonë supozojmë

$$P_{t-1|t-1} = \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{bmatrix} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.18)$$

Le të marrim një matricë njësi F_t dhe e transponojmë atë F_t^T .

$$F_t = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.19)$$

$$F_t^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.20)$$

Gjithashtu na nevojitet llogaritja e kovariancës edhe për zhurmën përkatësisht matjen e sensorëve të robotit e njohur Q_t. Pasi roboti ka tre gjendje atëherë edhe kjo matricë është 3x3.

$$Q_t = \begin{bmatrix} kov(x,x) & kov(x,y) & kov(x,\gamma) \\ kov(y,x) & kov(y,y) & kov(y,\gamma) \\ kov(\gamma,x) & kov(\gamma,y) & kov(\gamma,\gamma) \end{bmatrix} \ldots\ldots\ldots\ldots\ldots\ldots (2.21)$$

Por kovarianca ndërmjet dy variablave është e barabartë më variancën kështu kemi:

$$Q_t = \begin{bmatrix} var(x,x) & kov(x,y) & kov(x,\gamma) \\ kov(y,x) & var(y,y) & kov(y,\gamma) \\ kov(\gamma,x) & kov(\gamma,y) & var(\gamma,\gamma) \end{bmatrix} \ldots\ldots\ldots\ldots\ldots\ldots (2.22)$$

Kur vlera e matricës Q_t është e lartë nënkupton që më e saktë është vlera e matur nga sensori sesa vlera e parashikuar përgjatë modelit. Ekuacioni do të jetë:

$$P_{t|t-1} = F_t P_{t-1|t-1} F_t^T + Q_t \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.23)$$

$$P_{t|t-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \ldots\ldots\ldots\ldots (2.24)$$

Pasi kemi llogaritur kovariancën nevojitet gjithashtu që të llogarisim edhe diferencën ndërmjet vlerave reale të zhurmës (vlerave të sensorit) dhe atyre të parashikuara. z_t është vektori që paraqet vlerat reale të lexuara nga sensori përgjatë kohës t për shembull një sensor që mat tre gjendjet e hapësirës kështu kemi:

$$z_t = \begin{bmatrix} x_t \\ y_t \\ \gamma_t \end{bmatrix} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.25)$$

$h(x\hat{}_{(\mathrm{t}|t-1)}$ është modeli i parashikuar paraqet matjet e parashikuara përgjatë kohës t duke pasur parasysh vlerat e parashikuara të gjendjes përgjatë kohës t ekuacioni 2.17. Pra përderisa një robot lëviz nëpër hapësirë ne marrim të dhëna prej sensorëve për të krijuar një hapësire të gjendjes. Në këtë rast ne po e bëjmë të kundërtën ne po e përdorim gjendjen aktuale ose të parashikuar për të llogaritur matjet e sensorëve i njohur ndryshe si model i vëzhgimit i cili është një ekuacion matematikor.

$$h(\hat{x}_{\mathrm{t|t-1}}) = H_t \hat{x}_{\mathrm{t|t-1}} + \omega_t \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.26)$$

ku:

H_t – matrica që konverton vlerat e parashikuara të gjendjes përgjatë kohës t në vlerat e parashikuara të sensorit përgjatë kohës t

ω_t – vektor me numër të njëjtë të elementeve më atë të sensorit. Zakonisht për sensorët real mund të gjenden vlerat në specifkacion teknik sa i përket zhurmës.

Në këtë rast në robotin mobil supozojmë se kemi vendosur një sensor që matë vlerat e gjendjes [x,y, γ]. Kështu matrica H_t marrim se është identike me matricën A.

$$H_t = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.27)$$

Kurse në rastin tonë supozojmë vlerat e zhurmës të sensorit si më poshtë:

$$\omega_t = \begin{bmatrix} 0.07 \\ 0.07 \\ 0.04 \end{bmatrix} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.28)$$

Përmes ekuacionit 2.29 mund të llogarisim $\tilde{y}_t$ që paraqet diferencën ndërmjet vlerave të matura nga sensorët real dhe ata të parashikuar:

$$\tilde{y}_t = z_t - h(\hat{x}_{\mathrm{t|t-1}}) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.29)$$

Më lartë shohim se si llogaritet diferenca ndërmjet vlerës së parashikuar dhe asaj që kemi matur, tani nevojitet të kalkulojmë ndryshimet e kovariancës së mbetur të matjes e njohur edhe si kovarianca e parashikimit të matjes S_t ku:

$$S_t = H_t P_{t|t-1} H_t^T + R_t \ldots \ldots \ldots \ldots \ldots \ldots \ldots (2.30)$$

ku:

H_t- përdoret për të konvertuar hapësirsat e gjendjes së parashikuar përgjatë kohës t në matjet e parashikuara të sensorëve në kohën t

R_t – paraqet matricën e kovariancës të zhurmës të sensorit, matricë e cila duhet të jetë e barabartë me numrin e rreshtave dhe kolonave me atë të sensorit. Më poshtë kemi paraqitur një shembull ku në diagonale kemi vendosur vlerën 1 për shkak që në sensor përdorim vlera të parashikuara, por në rastet kur jemi të sigurte lidhur me matjen e sensorit vlerat e matricës përgjatë diagonales bien në 0.

$$R_t = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \ldots \ldots \ldots \ldots \ldots \ldots \ldots (2.31)$$

Marrim vlerën e kovariancës së parashikuar $P_{t|t-1}$, matricën e vlerave të matura H_t si dhe diferencës e kovariancës së mbetur S_t atëherë mund të llogarisim vlerën optimale të filtrit Kalman K_t:

$$K_t = P_{t|t-1} H_t^T S_t^{-1} \ldots \ldots \ldots \ldots \ldots \ldots \ldots (2.32)$$

K_t paraqet vlerën sesa hapësirat e gjendjes dhe kovarianca duhet korrektuar si pasojë e vlerave të matura nga sensori aktual z_t.

Tani që kemi vlerat reale të matura mund të kalkulojmë hapësirat e gjendjes përgjatë kohës t e cila paraqet pozitën aktuale të robotit duke shfrytëzuar vlerat e parashikuara të gjendjes $\hat{x}_{t|t-1}$, vlerën optimale të filtrit Kalman K_t dhe diferencën ndërmjet vlerave të matura $\tilde{y}_t$.

$$\hat{x}_{t|t} = \hat{x}_{t|t-1} + K_t \tilde{y}_t \ldots \ldots \ldots \ldots \ldots \ldots \ldots (2.33)$$

Gjithashtu mund të shohim kovariancën reale ose të korrigjuar $P_{t|t}$, pas matjeve të reja të sensorit.

$$P_{t|t} = (I - K_t H_t) P_{t|t-1} \ldots \ldots \ldots \ldots \ldots \ldots \ldots (2.34)$$

ku:

I – paraqet matricë njësi

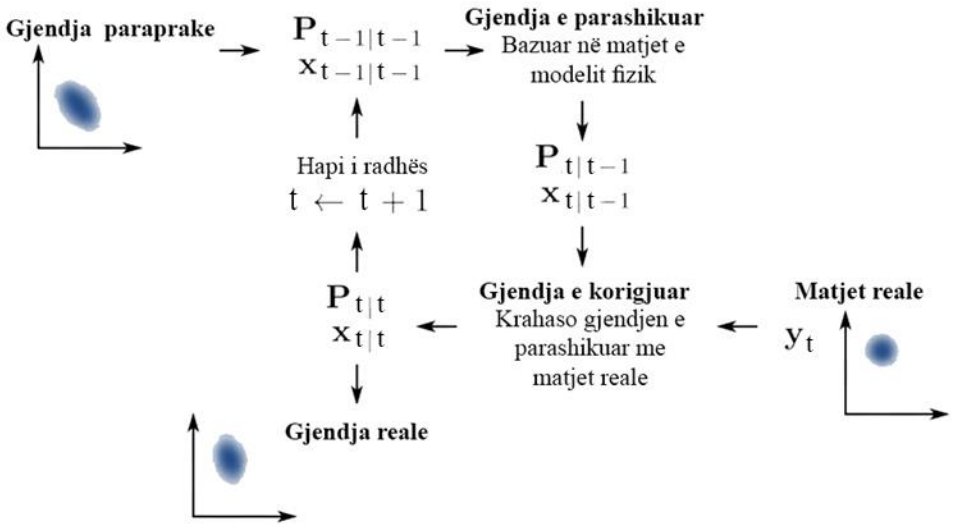

Fig 2. 9 Përshkrimi i EKF përmes diagramit

2.2.5.2 Filtri i Grimcave (Particle Filter)

Filtri i Grimcave (Particle Filter) përmban algoritmin më të popullarizuar sa i përket identifikimit të objekteve, këtë emërtim e morri për shkak që vlera e probabilitetit të gjeneruar në sistem paraqiten si grimca. Shembulli më i përdorur që e shfrytëzon këtë filtër është Monte Carlo [11]. EKF garanton saktësi vetëm për sisteme lineare siç u shpjegua më lartë si dhe në sisteme ku aplikohet zhurma Gaussian-e. Por shumica e problemeve në mjedis reale janë sisteme jo-lineare. Pasi robotët dhe sensorët janë gjithashtu jo-linear, filtri i grimcave zakonisht përdoret për llogaritje të pozicionit. Përderisa EKF është një metodë analitike që supozon sistemin si linear dhe kërkon parametra me lëvizje lineare, filtri i grimcave zakonisht përdoret teknikën e bazuar në simulim provo-gabim(try and error). Kështu tek SLAM ky vlerëson pozicionin e objektit duke supozuar se gabimi është përfshirë në hyrje në këtë rast vlerat e odometrisë varen nga sensori i distancës. Gjatë aplikimit të filtrit të grimcave pozicioni i panjohur përshkruhet si një grup i grimcave të quajtura mostra, varësisht prej lëvizjes së robotit lëvizin edhe grimcat dhe në të njëjtën kohë llogarisim edhe peshën e secilës grimcë, kështu duke i krahasuar më vlerën aktuale të matjes grimcat tjera shkojmë duke i zvogëluar derisa të arrijmë pozicionin aktual. P.sh tek robotët mobil secila grimcë përshkruhet me vlerat (x,y,i) ku përmes vlerave të x dhe y llogaritet pozicioni kurse i paraqet peshën e grimcës. Ky filtër zakonisht kalon nepër 5 hapa:

1. Inicializimi
2. Parashikimi
3. Përditësimi
4. Llogaritja e pozicionit
5. Ri-mostrimi

Më përjashtim të hapit të 1. Hapat prej 2-5 kryhen në mënyrë të përsëritur për të llogaritur pozicionin e robotit. Kjo metodë përdoret duke përditësuar shpërndarjen e grimcave në sistem që shfaq probabilitetin e lëvizjes së robotit në planin koordinativ x,y gjithmonë duke u bazuar në vlerën e matur të sensorit. Le të fokusohemi tek algoritmet MCL (MCL – Monte Carlo Localization) i cili ka gjetur një përdorim mjaft të lartë sa i përket llogaritjes së pozicionit dhe AMCL (Adaptive Monte Carlo Localization) i cili njihet si një version i përmisuar i MCL për shkak që arrin performancë më të lartë të procesimit në kohë reale, gjithashtu redukton kohën e ekzekutimit të algoritmit.

Objektivi kryesor i MCL është që të përcaktojë pozicionin e robotit në një mjedis, pra nevojitet që të dijmë x,y dhe θ të robotit në hartë. Pozicionin dhe orientimin (x,y,θ) përgjatë kohës t e shënojmë me x_t, informacionet e marra nga sensori i distancës përgjatë kohës t prej kohës fillestare e shenojmë më $z_{0...t}=\{z_0, z_1, ..., z_t\}$, si dhe informatat e lëvizjes të marra nga enkoderët e motorëve përgjatë kohës t prej kohës fillestare shënojmë me $u_{0...t}=\{u_0, u_1, ..., u_t\}$, atëherë me formulën e përditësimit Bayesian për llogaritjen e probabilitetit kemi:

$$bel(x_t) = \rho\left(x_t \middle| z_{0...t,} u_{0...t}\right) \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots (2.35)$$

Parashikimi i lëvizjes së ardhshme të cilin e shënojmë me $bel'(x_t)$ për kohën e ardhshme të robotit llogaritet përmes funksionit të lëvizjes $\rho(x_t|x_{(t-1)}, u_{(t-1)}$ edhe probabilitetit $bel(x_{t-1})$ të pozitës së mëparshme.

$$bel'(x_t) = \int \rho(x_t|x_{(t-1)}, u_{(t-1)}) bel(x_{t-1}) dx_{t-1} \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots (2.36)$$

Tani mund të llogarisim hapin e përditësimit. Modeli i sensorit $\rho(z_t|x_t)$, probabiliteti $bel(x_t)$ dhe konstanta për normalizim (η_t) përdoren për të llogaritur probabilitetin më të saktë përgjatë kohës gjithmonë duke u bazuar në informatat e marra nga sensori.

$$bel(x_t) = \eta_t \, \rho(z_t|x_t) bel'(x_t) \quad (2.37)$$

Pas llogaritjes së pozicionit, si dhe duke përdorur probabilitetin e pozicionit aktual $bel(x_t)$ mund të gjenerojmë grimca ose mostra N përreth robotit. E para është procesi i krijimit të mostrave të cilat kalojnë nëpër SIR (Sampling Importance Re-sampling) algoritmin ose procesin. P.sh le të shpërndajmë një grup mostrash (kampionesh) x_t duke u bazuar në lëvizjen e modelit përgjatë kohës t-1 $\rho(x_t|x_{(t-1)}, u_{(t-1)})$ dhe probabilitetin $bel(x_{t-1})$. Mostra "i" $x_t^{\prime(i)}$ përgjatë mostrave x_t, informatave të distancës z_t, dhe konstantes së normalizimit η përdorën për të llogaritur peshën e mostrës (kampionit) $\omega_t^{\prime(i)}$.

$$\omega_t^{(i)} = \eta \, \rho(z_t|x_t^{\prime(i)} \quad (2.38)$$

Tani që dijmë edhe peshën e mostrës mund të bëjmë ri mostrimin e tyre:

$$X_t = \left\{ x_t^{(j)} \middle| \, j = 1 \ldots N \right\} \sim \{x_t^{\prime(i)}, \omega_t^{(i)}\} \quad (2.39)$$

Kështu duke ripërsëritur SIR procesin përderisa grimcat janë në lëvizje, saktësia e llogaritjes të pozicionit të robotit rritet ndjeshëm.

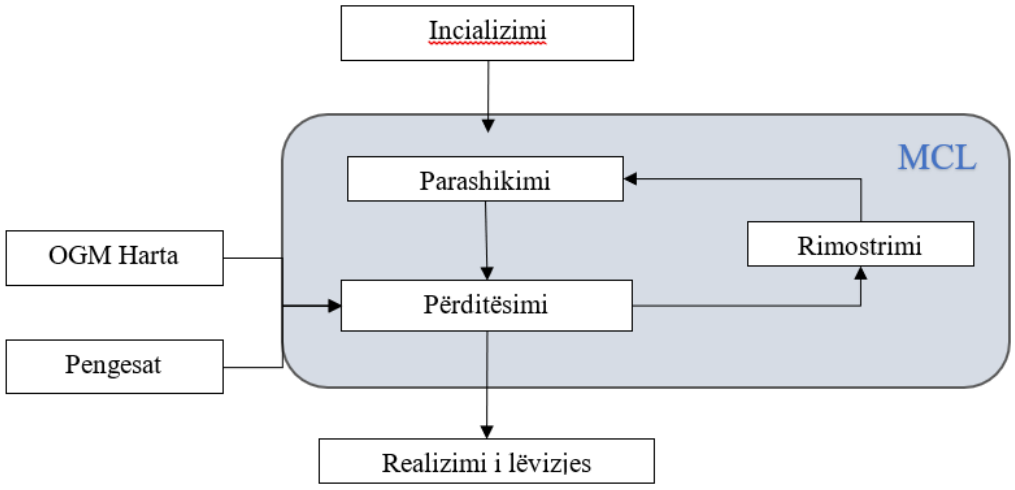

Fig 2. 10 Diagrami për MCL

2.2.5.3 GMapping

GMapping është një filtër i grimcave mjaft efikas i njohur si Rao-Blackwellized [6] i cili krijon harta në formë të rrjetës nga informatat e marra nga sensorët me laser. Këta filtra janë treguar mjaft efektiv sa i përket SLAM kjo pasi secila grimcë përmban një hartë individuale të mjedisit dhe ky filtër bën reduktimin për të mësuar hartën si dhe mundëson të kryhen operacione në mënyrë selektive të ri kalkulimit të mostrave. Ideja kryesore e këtij filtri është të kalkulojë lidhjen $\rho(x_{1:t}, m\,|z_{1:t,}u_{1:t-1})$ për hartën m dhe trajektoren $x_{1:t} = x_1, \ldots\ldots, x_t$ të robotit. Ky kalkulim arrihet duke përfshirë vlerat e marra nga sensori $z_{1:t} = z_1, \ldots\ldots z_t$ dhe matjet odometrike $u_{1:t-1} = u_1, \ldots\ldots u_{t-1}$ të robotit mobil ku kemi:

$$\rho(x_{1:t}, m\,|z_{1:t,}u_{1:t-1}) = \rho(m|x_{1:t}, z_{1:t}) \times \rho(x_{1:t}|z_{1:t}, u_{1:t-1}) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.40)$$

Ky faktorizim na mundëson që fillimisht të shqyrtojmë trajektoren e robotit më pas të kalkulojmë hartën për trajektoren e dhënë. Kështu për të vlerësuar trajektoren potenciale të kaluar aplikojmë një filtër grimcash (mostra). Çdo mostër përfaqëson një trajektore të mundshme të robotit, gjithashtu edhe hartat shoqërohen me nga një mostër. Hartat krijohen pas marrjes së vlerave të sensorit si dhe trajektores së gjeneruar në secilën grimcë. Edhe këtu si tek MCL përdoret algoritmi SIR. Ky proces ndahet në katër hapa kryesor.

1. Marrja e mostrave – Mostrat e ardhshme ($x_t^{(i)}$) merren nga mostrat paraprake ($x_{t-1}^{(i)}$) duke u bazuar në shpërndarjen proporcionale π.
2. Rëndësia e peshës – Secila mostër ka një peshë $\omega_t^{(i)}$ këto janë të nevojshme për shkak se shpërndarja proporcionale π nuk është e barabartë më shpërndarjen e gjendjeve pasardhëse. Kjo mund të kalkulohet:

   $$\omega_t^{(i)} = \frac{\rho\,(x_{1:t}^{(i)}|z_{1:t}, u_{1:t-1})}{\pi(x_{1:t}^{(i)}|z_{1:t}, u_{1:t-1})} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.41)$$

3. Ri-mostrimi – Mostrat largohen në mënyrë proporcional varësisht peshës së tyre. Në këtë hap merren vetëm një numër i kufizuar i mostrave të cilat më pas shpërndahen përsëri në mënyrë proporcionale dhe me peshë të njëjtë.

4. Vlerësimi (kalkulimi) i hartës – për secilën mostër harta $\rho\left(m^{(i)}\middle|x_{1:t}^{(i)}, z_{1:t}\right)$ kalkulohet duke u bazuar në trajektoren $x_{1:t}^{(i)}$ të asaj mostre dhe vlerave hyrëse $z_{1:t}$.

Pasi që gjatësia e trajektores rritet përgjatë kohës atëherë ky algoritëm do të ishte jo efikas në këtë rast kalkulojmë vetëm dy gjendjet paraprake të shpërndarjes π në këtë rast:

$$\pi(x_{1:t}|z_{1:t}, u_{1:t-1}) = \pi(x_t|x_{1:t-1}, z_{1:t}, u_{1:t-1}) \times \pi(x_{1:t-1}|z_{1:t-1}, u_{1:t-2}) \dots\dots\dots\dots\dots\dots\dots (2.42)$$

Kështu duke u bazuar në ekuacion 2.41 dhe 2.42 pesha e mostrave është:

$$\omega_t^{(i)} = \frac{\rho(x_{1:t}^{(i)}|z_{1:t}, u_{1:t-1})}{\pi(x_{1:t}^{(i)}|z_{1:t}, u_{1:t-1})} = \frac{\eta_p(z_t|x_{1:t}^{(i)}, z_{1:t-1})\rho(x_t^{(i)}|x_{t-1}^{(i)}, u_{t-1})}{\pi(x_t^{(i)}|x_{1:t-1}^{(i)}, z_{1:t}, u_{1:t-1}} \frac{\rho\left(x_{1:t-1}^{(i)}\middle|z_{1:t-1}, u_{1:t-2}\right)}{\pi\left(x_{1:t-1}^{(i)}\middle|z_{1:t-1}, u_{1:t-1}\right)} \dots\dots (2.43)$$

Pasi $\pi\left(x_{1:t-1}^{(i)}\middle|z_{1:t-1}, u_{1:t-1}\right) = \omega_{t-1}^{(i)}$ dhe konstanta e normalizimit $\eta_p = \frac{1}{\rho(z_t|z_{1:t-1}, u_{1:t-1})}$ atëherë kemi:

$$\omega_t^{(i)} = \frac{\rho(z_t|m_{t-1}^{(i)}, x_t^{(i)})\rho(x_t^{(i)}|x_{t-1}^{(i)}, u_{t-1})}{\pi(x_t|x_{t-1}^{(i)}, z_{1:t}, u_{1:t-1})} \times \omega_{t-1}^{(i)} \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots (2.44)$$

Ekuacioni 2.44 paraqet modelin në të cilin më së shumti mbështeten për kalkulimin e peshës së mostrave.

Pra planifikimi i rrugës është një problem mjaft i vështirë dhe kompleks më lartë sqaruam metodën analitike të disa filtrave që përdoren për llogaritjen e pozicionit dhe realizimit të lëvizjes së ardhshme por këto të gjitha janë të supozuara në kushte ideale dhe nevojitet integrimi edhe i shumë faktorëve tjerë të mjedisit që këto lëvizje të realizohen.

Kështu në kapitullin 4 të këtij punimi do të shpjegojmë procesin se si janë integruar këto filtra përkatësisht GMapping në ROS duke mundësuar simulimin dhe gjithashtu aplikimin në robotë real të SLAM.

3 ROS

3.1 Pse ROS?

Fusha e robotikës kohëve të fundit ka filluar të këtë një rritje mjaft të shpejt. Në këto ditë shohim shumë aplikime të IoT (Internet of Things), shumë projekte open-source duke filluar nga robotët mobil, dronët deri tek robotët humaniod. Por pa marrë parasysh këtë zhvillim për shumë zhvillues mbetet një sfidë mjaft e madhe arritja e autonomisë së plotë të funksionimit të këtyre robotëve. Në këtë kapituj do të flitet për platformën softuerike të njohur si sistemi operativ për robotikë, ROS (Robot Operating System). Përshkrimi zyrtar për ROS është se ai është një sistem open-source meta operativ për robotin tuaj. Ky siguron shërbime që sistemet operative mundësojnë si menaxhimi i resurseve harduerike, bartja e informatave ndërmjet proceseve të ndryshme, kontrollimi i pajisjeve veç e veç. Gjithashtu vegla për programim, librari të gatshme për aplikim gjatë kodimit etj. Tre funksionet kryesore që veçojnë ROS janë:

1. Fuqia procesuese e shpërndarë,
2. Ripërdorimi i softuerit,
3. Testimi i shpejtë.

Fuqia procesuese e shpërndarë - Sistemet robotike të avancuara mbështeten në softuer që përfshin procese të ndryshme dhe funksionon në disa kompjuterë të ndryshëm. Për shembull:

- Shumë robotë të avancuar kanë kompjuterë të shumtë, ku secili kontrollon një grup të sensorëve dhe aktuatorëve që të realizojë një punë të caktuar.
- Edhe në rastet kur kemi një kompjuterë të vetëm, sugjerohet që programi i robotit të ndahet në disa pjesë që kryejnë funksione të pavarura që bashkëpunojnë për arritjen e qëllimit final.
- Kur këta robotë përpiqen që të bashkëpunojnë për një punë të caktuar, nevojitet një standard i komunikimit mes tyre për koordinimin e funksioneve.
- Mënyrat e kontrollimit të robotëve nga njeriu p.sh shpesh bëhet kontrollimi nga një kompjuter personal, laptop ose telefon të mençur.

Pra qëllimi kryesor në të gjitha këto raste mundësimi i nevojës për komunikim ndërmjet proceseve të shumta që janë ose jo në të njëjtin kompjuter.

Ripërdorimi i softuerit – Qëllimi kryesor i open-source është që shumë funksione të caktuara i marrim edhe përdorim të gatshme më pas t'i adoptojmë në sistemin robotikë që jemi duke krijuar. Kështu shumë algoritme si navigimi, planifikimi i lëvizjeve, krijimi i hartave, kontrollimi i lëvizjeve etj. zakonisht merren në formë librarie të gatshme dhe adaptohen për arritjen e qëllimeve të caktuara të robotit. Këtu ROS na ndihmon shumë në dy mënyra

- Libraritë standarde të ROS janë stabile, të pa korruptuara
- Komunikimi i informatave përmes ROS ka filluar të bëhet si një standard i ri i operimit në robotikë, ku prej kontrollimit të harduerit deri tek implementimi i algoritmeve gjendet paketa të gatshme ne ueb sajtin e ROS.

Në këtë rast shumë zhvillues që përdorin ROS fokusohen më shumë në krijimin e inovacioneve të reja dhe më pak kohë të "rizbulimit të rrotës".

Testimi i shpejtë – Shpesh zhvillimet në fushën e robotikës janë më sfiduese se zhvillimet tjera, kjo për shkak që koha e testimit dhe mundësitë e gabimeve janë të mëdha. Robotët fizikë jo gjithmonë janë afër për testim, edhe në rast kur janë testimi është shumë i ngadaltë. Kështu ROS mundëson që:

- Sistemet e dizajnuara në ROS mundësojnë që testimi të bëhet në mënyrë të ndarë nga hardueri, pra mundëson simulimin.
- Gjithashtu na mundëson ruajtjen e të dhënave të lexuara nga sensorët dhe aktuatorët në këtë rast është shumë më e lehtë simulimi i skenarëve të ndryshme përgjatë aplikimit se si mund të reagojë roboti.

Duhet të kemi parasysh që ROS nuk është një gjuhë programuese, zhvillimet e ROS zakonisht janë të programuara në gjuhen C++ por ka edhe librari të cilat janë kompatibile me Python, Java, gjithashtu ROS nuk është vetëm një paketë e librarive, por mund të përdoret si një server për multiprocesim, simulime grafike etj.

ROS nuk është një IDE (Integrated Development Environment), por është kompatibil më shumë prej tyre, por përgjatë punës më ROS duket mjaft e arsyeshme që programimi dhe ndryshimet në shumicën e rasteve të aplikohen nga tekst editorët që ofrohen nga ROS, dhe linjat komanduese (command line).

3.2 Instalimi dhe konfigurimi i ROS

Para se të startojmë programimin në ROS fillimisht do të përshkruajmë mënyrën e instalimit të tij dhe të realizojmë disa procese bazike me qëllim të kuptimit se si ROS funksionon. ROS është i gjithi i bazuar në sisteme operative Unix të cilat mund të jenë debian dhe rpm, në këtë rast është shfrytëzuar sistemi operativ Ubuntu ku do të instalojmë paketat që përmbajnë këtë aplikacion. Më poshtë është një udhëzim i shkurtër se di duhet instaluar ROS [24].

1.Shtojmë repository në sistemin operativ ku konfigurohet burimi nga web faqja zyrtare e ROS.

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" >/etc/apt/sources.list.d/ros-latest.list'
```

2. Importojmë SSL keys në server.

```
curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
```

3.Sistemi operativ fillimisht duhet të jetë i përditësuar

```
sudo apt update
```

4.Pastaj zgjedhim versionin e ROS që dëshirojmë të instalojmë ROS Indigo ose ROS Indigo

```
sudo apt install ros-indigo-desktop-full
```

5.Eksportojmë variablat globale që nevojiten përgjatë punës me ROS

```
echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc

source ~/.bashrc
```

6. Gjithashtu instalojmë edhe vegla të tjera opsionale që na nevojiten përgjatë punimit

```
sudo apt-get install python3-rosdep python3-rosinstall python3-rosinstall-generator python3-wstool build-essential
```

7. Inicializojmë ROS për herë të parë (ky hap nuk nevojitet për hera tjera)

```
sudo apt install python3-rosdep

sudo rosdep init

rosdep update
```

Këto janë hapat që nevojiten për instalimin e ROS-it në Ubuntu server pothuajse të ngjashme janë komandat edhe në sisteme operative të tjera Unix, ku dallon vetëm mënyra e ekzekutimit të komandës për shembull në Ubuntu **`apt install ros-indigo-desktop-full`** në Linux do të ishte **`yum install ros-indigo-full.`**

3.3 Startimi i ROS dhe programimi i një aplikacioni

Pasi kemi instaluar ROS hapim një terminal të ri dhe sigurohemi që të gjitha variablat janë të eksportuara në mënyrë që të arrijmë të lexojmë paketat e ROS. Në këtë rast ekzekutojmë komandën më poshtë ku shfaqen inc formata lidhur me versionin e ROS, IP e master node, folderat ku shkruan dhe lexon ROS.

```
akrrabaj@akrrabaj:~$ printenv | grep ROS
ROS_ROOT=/opt/ros/indigo/share/ros
ROS_PACKAGE_PATH=/opt/ros/indigo/share:/opt/ros/indigo/stacks
ROS_MASTER_URI=http://192.168.0.112:11311
ROSLISP_PACKAGE_DIRECTORIES=
ROS_DISTRO=indigo
ROS_IP=192.168.0.112
ROS_ETC_DIR=/opt/ros/indigo/etc/ros
```

Nga këtu shohim se mund të startojmë ROS edhe atë e startojmë duke ekzekutuar `roscore`.

```
akrrabaj@akrrabaj:~$ roscore
... logging to /home/akrrabaj/.ros/log/863f481e-2b97-11ec-ac8d-0800273f4a21/roslaunch-akrrabaj-3493.log
Checking log directory for disk usage. This may take aëhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://192.168.0.112:39898/
ros_comm version 1.11.21
SUMMARY
========
PARAMETERS
```

```
 * /rosdistro: indigo
 * /rosversion: 1.11.21

NODES

auto-starting new master
process[master]: started with pid [3505]
ROS_MASTER_URI=http://192.168.0.112:11311/

setting /run_id to 863f481e-2b97-11ec-ac8d-0800273f4a21
process[rosout-1]: started with pid [3518]
started core service [/rosout]
```

Pasi kemi startuar ROS na duhet të definojmë mjedisin(environment) për zhvillimin e aplikacioneve pasi që secili projekt në ROS ka mjedisin e vet të zhvillimeve. ROS për këtë qëllim shfrytëzon Catkin i cili është sistemi zyrtar për zhvillime në ROS, është shumë e ngjashme me CMake por që kjo mbështet gjetjen e paketave, infrastrukturës si dhe ndërtimin e projekteve të vogla për realizimin e projektit. Më poshtë është paraqitur krijimi i mjedisit për zhvillime në ROS. Krijojmë një folder në /home/akrrabaj/ me emrin **`catkin_ws`** si dhe një nënfolder **`src`**.

```
cd /home/akrrabaj
mkdir catkin_ws
cd catkin_ws/
mkdir src
akrrabaj@akrrabaj:~/catkin_ws$ pwd
/home/akrrabaj/catkin_ws
```

Pastaj që mjedisi duhet të jetë i qasshëm nga ROS na nevojitet që sa herë të hapim një sesion të ri variablat e mjedisit catkin të jenë aktive. Në këtë rast shtojmë folderin ku gjendet burimi i kodit në bash profile të userit

```
akrrabaj@akrrabaj:~/catkin_ws$ echo "source /home/akrrabaj/catkin_ws/devel/setup.bash" >>  /home/akrrabaj/.bashrc
```

Ndërtojmë mjedisin dhe inicializojmë Catkin

```
akrrabaj@akrrabaj:~/catkin_ws$ catkin build
akrrabaj@akrrabaj:~/catkin_ws$ catkin init
Catkin ëorkspace `/home/akrrabaj/catkin_ws` is initialized. No action taken.
--------------------------------------------------------------
Profile:                     default
Extending:             [env] /opt/ros/indigo
Ëorkspace:                   /home/akrrabaj/catkin_ws
--------------------------------------------------------------
Build Space:        [exists] /home/akrrabaj/catkin_ws/build
Devel Space:        [exists] /home/akrrabaj/catkin_ws/devel
Install Space:      [unused] /home/akrrabaj/catkin_ws/install
Log Space:          [exists] /home/akrrabaj/catkin_ws/logs
Source Space:       [exists] /home/akrrabaj/catkin_ws/src
DESTDIR:            [unused] None
--------------------------------------------------------------
Devel Space Layout:          linked
Install Space Layout:        None
--------------------------------------------------------------
Additional CMake Args:       None
Additional Make Args:        None
Additional catkin Make Args: None
Internal Make Job Server:    True
Cache Job Environments:      False
--------------------------------------------------------------
Ëhitelisted Packages:        None
Blacklisted Packages:        None
--------------------------------------------------------------
Ëorkspace configuration appears valid.
```

Pasi kemi incializuar me sukses Catkin mjedisin, ne vazhdim do te krijohet një test aplikacion ne te cilin fillimisht do te krijohet nje node e emërtuar **`fim_node`** që shfaq një mesazh **`FIM Mekatronike, Test API!`**. Krijojmë një pakete(package).

```
catkin create pkg fim_firstapi --catkin-deps roscpp
```

Krijojmë programin në `./src/fim_firstapi/src/fim_node.cpp`

```
/**
** Krijimi i nje node FIM
**/
#include <ros/ros.h>

int main(int argc, char* argv[])
{
  //Deklarojme variablen per startimin e node-it
  ros::init(argc, argv, "fim_node");

  //Importojme klasen per menaxhmin e node
  ros::NodeHandle nh;

  //Mesazhi qe duhet shfaqur pas startimit te Node
  ROS_INFO("FIM Mekatronike, Test API!");

  //Mbatja e node-it aktiv edhe pas mbylljes se terminalit.
  ros::spin();
}
```

Pastaj editojmë /home/akrrabaj/catkin_ws/src/fim_firstapi/CMakeLists.txt më qëllim që kodin më lartë të jetë i ekzekutueshëm.

```
akrrabaj@akrrabaj:~/catkin_ws/src/fim_firstapi$ cat /home/akrrabaj/catkin_ws/src/fim_firstapi/CMakeLists.txt
cmake_minimum_required(VERSION 2.8.3)
project(fim_firstapi)

## Compile as C++11, supported in ROS Kinetic and neëer
add_compile_options(-std=c++11)
find_package(catkin REQUIRED COMPONENTS
```

```
  roscpp
)


catkin_package(
#  INCLUDE_DIRS include
#  LIBRARIES fim_firstapi
#  CATKIN_DEPENDS roscpp
#  DEPENDS system_lib
)


###########
## Build ##
###########


## Specify additional locations of header files
## Your package locations should be listed before other locations
include_directories(
# include
  ${catkin_INCLUDE_DIRS}
)


## Declare a C++ executable
## Ëith catkin_make all packages are built ëithin a single CMake context
## The recommended prefix ensures that target names across packages don't coll
ide
add_executable(fim_firstapi src/fim_node.cpp)


## Specify libraries to link a library or executable target against
target_link_libraries(fim_firstapi
   ${catkin_LIBRARIES}
 )
```

Kompajlojmë programin.

```
akrrabaj@akrrabaj:~/catkin_ws/src/fim_firstapi$ cd /home/akrrabaj/catkin_ws/
```

```
akrrabaj@akrrabaj:~/catkin_ws$ catkin build
--------------------------------------------------------------
Profile:                     default
Extending:          [cached] /opt/ros/indigo
Ëorkspace:                   /home/akrrabaj/catkin_ws
--------------------------------------------------------------
Build Space:        [exists] /home/akrrabaj/catkin_ws/build
Devel Space:        [exists] /home/akrrabaj/catkin_ws/devel
Install Space:      [unused] /home/akrrabaj/catkin_ws/install
Log Space:          [exists] /home/akrrabaj/catkin_ws/logs
Source Space:       [exists] /home/akrrabaj/catkin_ws/src
DESTDIR:            [unused] None
--------------------------------------------------------------
Devel Space Layout:          linked
Install Space Layout:        None
--------------------------------------------------------------
Additional CMake Args:       None
Additional Make Args:        None
Additional catkin Make Args: None
Internal Make Job Server:    True
Cache Job Environments:      False
--------------------------------------------------------------
Ëhitelisted Packages:        None
Blacklisted Packages:        None
--------------------------------------------------------------
Ëorkspace configuration appears valid.
--------------------------------------------------------------
[build] Found '1' packages in 0.0 seconds.
[build] Package table is up to date.
Starting  >>> fim_firstapi
Finished  <<< fim_firstapi                [ 0.1 seconds ]
[build] Summary: All 1 packages succeeded!
[build]   Ignored:   None.
```

```
[build]   Ëarnings:  None.
[build]   Abandoned: None.
[build]   Failed:    None.
[build] Runtime: 0.1 seconds total.
```

Në fund startojmë nodin e krijuar dhe kontrollojmë me rostopic nëse është aktiv.

```
akrrabaj@akrrabaj:~$ rosrun fim_firstapi fim_firstapi
[ INFO] [1634075760.899299650]: FIM Mekatronike, Test API!
[ INFO] [1634075760.899299651]: FIM Mekatronike, Test API!
[ INFO] [1634075760.899299652]: FIM Mekatronike, Test API!
[ INFO] [1634075760.899299653]: FIM Mekatronike, Test API!
[ INFO] [1634075760.899299654]: FIM Mekatronike, Test API!
akrrabaj@akrrabaj:~$ rosnode list
/fim_node
/rosout
```

3.4 TURTLEBOT dhe ROS

Ekzistojnë afërsisht 180 robotë fizikë të cilët i suporton ROS ose janë kompatibil më të ku disa prej tyre janë robotë të personalizuar nga zhvilluesit apo kompanitë e ndryshme. Robotët më të njohur fizik në ROS janë TurtleBot dhe PR2. TurtleBot është pothuajse standard përgjatë aplikimeve dhe simulimeve në ROS, fjala Turtle rrjedh prej robotit Turtle i cili ishte zhvilluar në gjuhën programuese Logo1 në vitin 1967 ku secili version kishte një ikonë të re të Turtle, dhe për këtë edhe logo e ROS me nëntë pika rrjedh nga pjesa e prapme e Turtle(Breshkës). Janë tre versione kryesore të robotëve fizik TurtleBot. Turtlebot 1 i zhvilluar nga Open Robotics në vitin 2010 më pas ka filluar shitja në vitin 2011. Turtlebot2 e cila është zhvilluar nga Yujin Robot i bazuar në robotin iClebo Kobuki, në vitin 2017 gjithashtu është krijuarTutleBot3 një model më i avancuar i modeleve paraprake, më i vogël si dhe lehtë për t'u programuar për kërkime të ndryshme dhe krijimin e prototipave, si dhe TurtleBot 2i i krijuar nga Interbotix model të cilin do të përdorim në këtë punim.

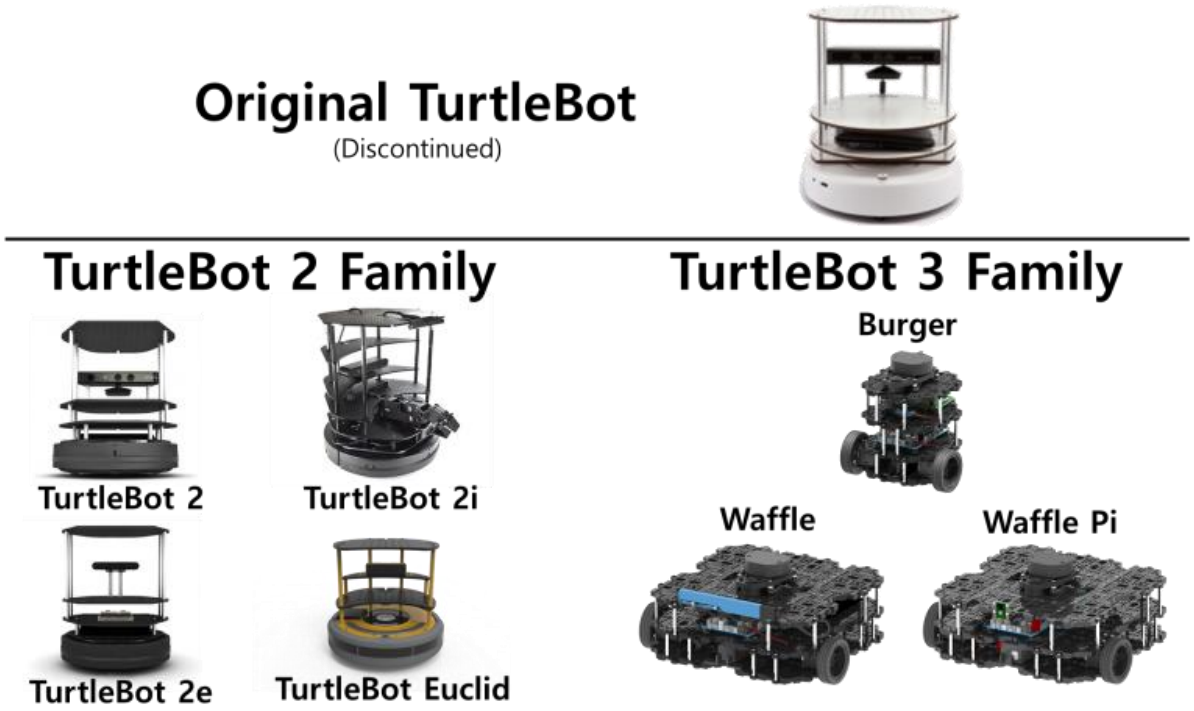

Fig 3. 1 Modelet e robotëve TurtleBot

3.4.1 TurtleBot 2i

TurtleBot 2i është një platformë robotike e gjitha e bazuar në ROS, kjo përpos që është ridizajnuar shasia e TurtleBot 2 është shtuar si funksion edhe krahu robotik Pincher MK3 me katër shkallë të lirisë më qëllim që roboti të ndërveprojë me objekte të vogla më botën reale, që e bën këtë model një robot manipulatorë mjaft të avancuar. Zakonisht ofrohen edhe disa softuer demo si:

- Navigimi Autonom,
- Identifikimi i hapësirës,
- Manipulimi dhe renditja e objekteve me krahë robotikë,
- Shmangia e pengesave,
- Shembuj me teleoperim etj.

Që të realizohen këta softuëer janë shfrytëzuar specifikiat harduerike:

CPU:

- INTEL NUC - BOXNUC6CAYH
- 8GB Ram
- 120GB SSD/HDD

- 802.11AC WiFi / Bluetooth 4.0
- Ubuntu 16.04 / ROS Kinetic

Senzorët:

- Intel RealSense 3D Camera SR300-Series
- Kamerë Orbbec Astra
- Senzorët për detektimin e pengesave dhe skajeve.

Ndërsa më poshtë janë shfaqur edhe disa karakterstika të tjera fizike.

- Shasia Kobuki,
- Krah Pincher MK3,
- Kontroller Arbotix-M,
- Shpejtësia maksimale e levizjës (nxitimit): 70 cm/s,
- Shpejtësia maksimale e rrotullimit: 180 deg/s,
- Ngarkesa 2 kg (pa krah), 1 kg (me krah),
- Nuk do të mund të largohet nga një një thellësi më të madhe se 5 cm,
- Kufiri i ngjitjes deri 12 mm ose më të ulëta,
- Koha e funksionimit, 4-6 orë (koha e funksionimit ndryshon në varësi të ngarkesës),
- Koha e pritshme e karikimit, 2-3 orë (koha e karikimit ndryshon në varësi të ngarkesës),
- Stacioni i karikimit automatik brenda një zone 2mx5m përballë stacionit të dokimit.

Fig 3. 2 Pamja në 3D Simulator e TurtleBot2i

3.4.2 Turtlesim

Në këtë pjesë do të instalojmë një Simulator të turtlebot të quajtur Turtlesim, ky Simulator ka disa module të gatshme që i përdorim për teleoperim gjithashtu edhe disa shembuj të tjerë. Për të instaluar këtë Simulator ekzekutojmë:

```
apt-get install ros-indigo-turtlesim
```

Hapim tre terminale në Ubuntu ku kemi instaluar ROS dhe Turtlesim, ekzekutojmë:

```
akrrabaj@akrrabaj:~$ roscore
akrrabaj@akrrabaj:~$ rosrun turtlesim turtlesim_node
akrrabaj@akrrabaj:~$ rosrun turtlesim turtle_teleop_key
```

Në terminalin ku kemi ekzekutuar `turtle_teleop_key` navigojmë përmes tastierës me →,←,↑,↓, dhe shohim simulimin e lëvizjeve.

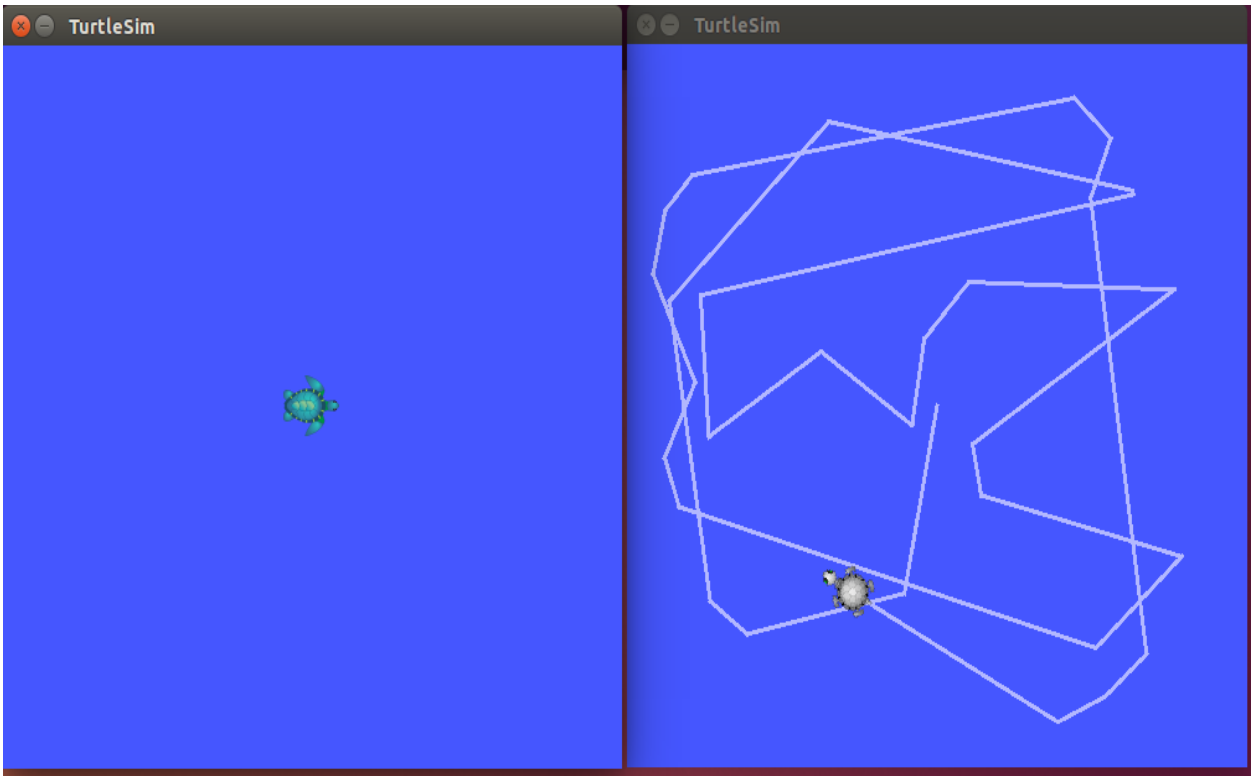

Fig 3. 3 Simulimi i lëvizjeve përmes tastierës në simulatorin Turtlesim

Në fig.3.3 shohim në anën e majtë se si duket Simulatori në momentin që startohet kurse në anën e djathtë është paraqitur simulimi i lëvizjeve të realizuara më teleoperim.

3.5 ROS Master node me IP statike

Deri tani kemi shpjeguar se si bëhet instalimi i ROS, programimi i një aplikacioni si dhe startimi i simulatorit Turtlesim, por të gjitha këto janë bërë në një Ubuntu Server në të cilin jemi të kyçur, gjë e cila pamundëson navigimin nga distanca. Në këtë rast do të konfigurojmë serverin në mënyrë që të mund të komunikojmë përmes SSH(port 22), ku do të vendosim një IP statike ku në momentin që startohet serveri do të ketë IP e caktuar. IP statike është në subnet privat dhe gjatë navgimit duhet që edhe pajisja e cila shfrytëzohen për navigim të jetë e kyçur në rrjet të njëjtë përndryshe nuk do të mund të komunikon me master node të ROS. Për të caktuar një IP statike konfigurimi mund të ndryshojë varësisht prej sistemit operativ, në këtë rast është përdorur Ubuntu 14.04 LTS dhe bëjmë këto ndryshime.

```
vi /etc/network/interfaces
# interfaces(5) file used by ifup(8) and ifdoën(8)
auto eth0
iface eth0 inet static
   address 192.168.0.112
   netmask 255.255.255.0
   netëork 192.168.0.0
   gateway 192.168.0.1
   dns-nameservers 8.8.8.8
```

Restartojmë servisin më qëllim që ndryshimet të kenë efekt.

```
service networking restart
```

Kontrollojmë IP, si dhe sigurohemi që SSH port është hapur.

```
root@akrrabaj:/home/akrrabaj# ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:3f:4a:21
          inet addr:192.168.0.112  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe3f:4a21/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:7135 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4371 errors:0 dropped:0 overruns:0 carrier:0
```

```
          collisions:0 txqueuelen:1000
          RX bytes:2972723 (2.9 MB)  TX bytes:432760 (432.7 KB)
root@akrrabaj:/home/akrrabaj# telnet 192.168.0.112 22
Trying 192.168.0.112...
Connected to 192.168.0.112.
Escape character is '^]'.
SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.13
^C
Connection closed by foreign host.
```

Pasi ky projekt është laboratorik dhe përdoret për qëllime eksperimentale atëherë mund të vazhdohet më IP private statike siç u definua më lartë. Por në rastet kur roboti mobil nevojitet që të kontrollohet navigimi nga distanca nga një subnet tjetër atëherë nevojitet që roboti mobil t'i sigurohet një IP publike nga providerët e ndryshëm, konfigurimi mbetet pothuajse i njëjtë vetëm ndryshohen IP,Subnet dhe Gateway. Kjo është njëra prej metodave pasi që mund të kemi edhe konfigurime të ndryshme p.sh VPN. Ekzistojnë aplikacione të ndryshme që përdoren për komunikim me TCP/IP për këtë punim zakonisht SSH si Putty, Secure CRT, Teraform etj.

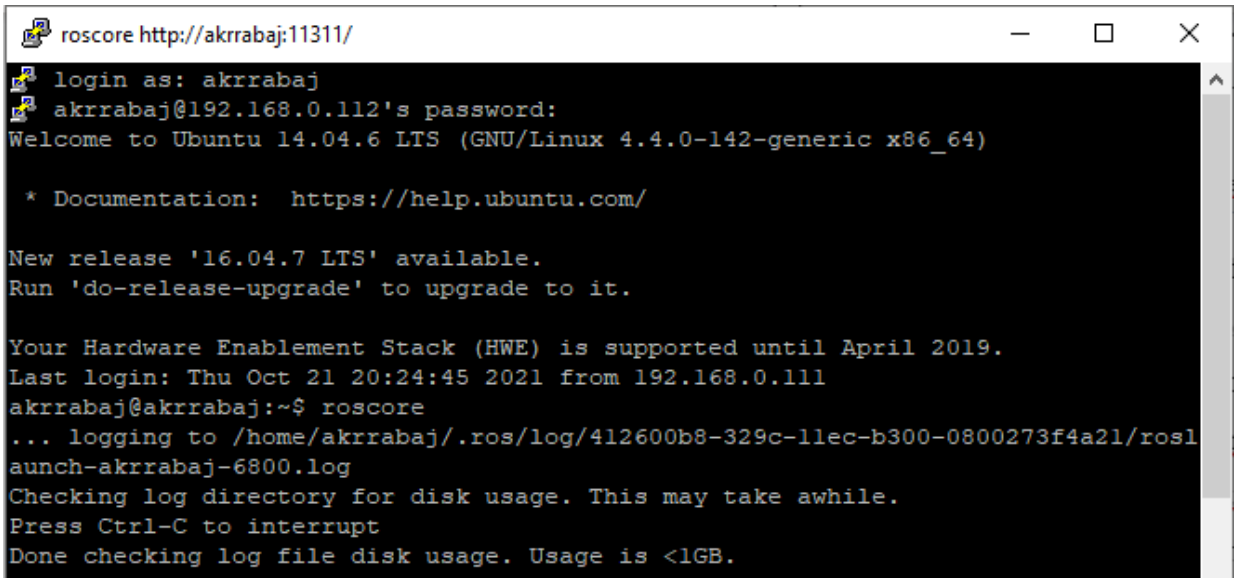

Fig 3. 4 Startimi i ROS përmes IP Statike

4 INTEGRIMI I SLAM NË ROS

Në kapitullin e dytë të këtij punimi sqaruam anën teorike të SLAM, tani do të sqarojmë se si SLAM është integruar në ROS. Për të realizuar SLAM nevojiten tri funksione kryesore:

1. Hartat
2. Pozicioni
3. Distanca

Në ROS secila nga këto është një funksion në vete të cilat ndërlidhen deri te arritja e realizimit të SLAM, këto funksione zakonisht janë të gatshme për aplikim në formë të paketave (packages) dhe se neve na duhet vetëm t'i aplikojmë dhe koordinojmë më qëllim të arritjes së rezultateve më efikase. Duke ju referuar qëllimit te punimit fillimisht do të sqarojmë se si krijohen hartat në ROS duke përdorur robotin mobil Turtlebot.

4.1 Hartat në ROS

Hartat janë shumë të rëndësishme në fushën e robotikës sidomos tek robotët mobilë. Sfida kryesore është që të arrijmë që të përkthejmë hartën në atë gjuhë që roboti e kupton p.sh roboti nuk mund të lexojë hartat nga letra. Definimi i hartave në robotikë vazhdon edhe sot duke u diskutuar dhe u zhvilluar kështu kohëve të fundit shohim shumë harta të cilat përfshijnë dimensione dy-dimensionale gjithashtu edhe ato tredimensionale. Sa i përket punimit tonë ne do të përdorim harta dy-dimensionale OGM (Occupancy Grid Map) që përdoren gjerësisht në ROS. Kjo i referohet një grupi të algoritmeve në fushën e probabilitetit të robotikës, ku gjenerohen harta varësisht prej zhurmës dhe matjeve të sensorit me supozimin se pozicioni i robotit është i njohur. Qëllimi kryesore i këtij algoritmi është të llogarisë probabilitetin e pasëm (Posterior Probability) për hartat duke marrë parasysh të dhënat e matura:

$$\rho(m|z_{1:t}, x_{1:t}) \quad (4.1)$$

ku:

m – paraqet hartën,

$z_{1:t}$ – paraqet matjet prej kohës 1 deri në t (vlerat hyrëse të matura nga sensori)

$x_{1:t}$ – paraqet pozicionin e robotit prej kohës 1 deri në t

Zakonisht në ROS hapësirat e OGM të shfaqura me ngjyra të lehta ose të bardha paraqesin zonën e lirë të lëvizjes kurse ato me ngjyra të errësuara paraqesin pengesat. Por në prapavijë këto ngjyra paraqesin vlerën e njohur si grayscale e cila mat vlera prej 0-255, vlerë e cila arrihet nga probabiliteti i pasëm dhe kjo vlerë konvertohet në integer [0-100], dhe kur vlera e këtij integer i afrohet vlera 0 tregon se hapësira përreth është e lirë dhe anasjelltas sa më shumë i afrohet vlerës 100 tregon se hapësira nuk është e lirë për lëvizje.

Tani do të realizojmë një manovrim të thjeshtë të SLAM më qëllim që të sqarojmë OGM hartat si dhe të ruajmë ato. Kështu në sistemin operativ hapim katër terminale dhe në secilin terminal ekzekutojmë komandat.

```
roslaunch turtlebot_gazebo turtlebot_world turtlebot_world.launch
roslaunch turtlebot_gazebo gmapping_demo.launch
roslaunch turtlebot_rviz_launchers view_nagivation.launch
roslaunch tutlebot_teleop keyboard_teleop.launch
Control Your Turtlebot!
---------------------------
Moving around:
   u    i    o
   j    k    l
   m    ,    .

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

Pas ekzekutimit të komandës së fundit shohim se si bëhet navigimi i turtlebot përmes tastierës, në këtë rast lëvizim robotin nëpër mjedis të caktuar në këtë rast janë disa korridore të shfaqura, mjedis ky i krijuar në Gazebo fig 4.1.

Fig 4. 1 Mjedisi i krijuar për manovrim të robotit në Gazebo

Pasi arrijmë një nivel të kënaqshmërisë së hartës OGM të krijuar nga Gmapping i cili shfaqet në aplikacionin RVIZ fig 4.2, atëherë shumë lehtë mund të ruajmë këtë duke ekzekutuar:

```
rosrun map_server map_saver -f /home/akrrabaj/fim_simulation
```

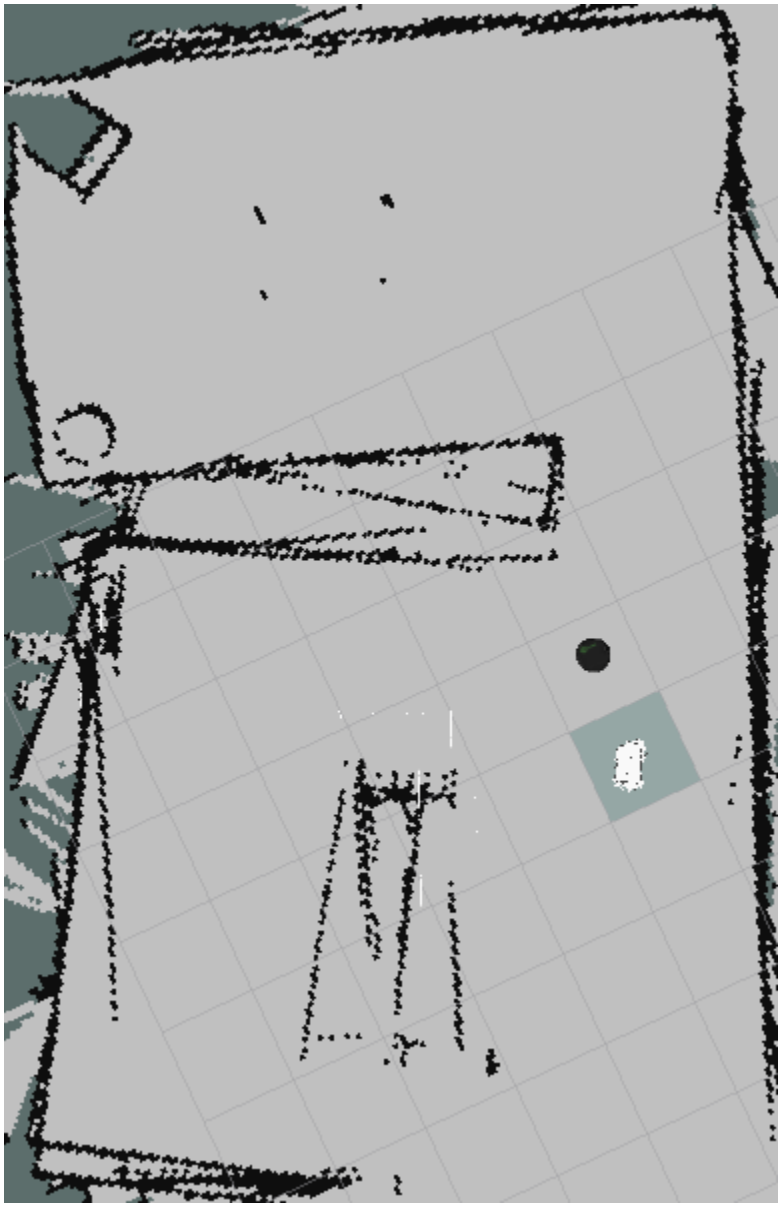

Fig 4. 2 OGM Harta e krijuar pas ekzekutimit të SLAM

Hartat krijohen duke u bazuar në odometrinë e robotit. Harta e krijuar ruhet në direktoriumin ku është duke u ekzekutuar komanda map_saver përderisa nuk e specifikojmë folderin sikur në rastin tonë, ku opsioni -f paraqet folderin dhe emrin e file-ave ku ruhen përkatësisht një *.pgm dhe *.yaml file. Këta file përmbajnë informacionet mbi hartën. File *.yaml përmban emrin e imazhit, rezolucionin i cili paraqet metër/pixel, pikën e origjinës (x,y,θ), përshkrimin e hapësirës së lirë të manovrimit.

```
akrrabaj@akrrabaj:~$ cat fim_simulation.yaml
image: /tmp/fim_simulation.pgm
resolution: 0.050000
origin: [-15.400000, -12.200000, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

Krijimi i këtyre hartave kërkon shumë kohë të manovrimit të robotit nëpër mjedis. Duke ju bazuar hapit të fundit të filtrave të sqaruara në kapitullin 2 atij të ri-mostrimit (resampling), atëherë shohim që sa më shumë të lëviz roboti nëpër mjedis aq më e saktë bëhet harta, dhe pas një kohë roboti mund të lëviz lirshëm nëpër mjedis.

4.2 Navigimi në ROS

Më lartë sqaruam se si krijohen hartat, por nevojiten edhe funksione tjera për realizimin e SLAM. Elementi i parë që na duhej për krijimin e hartës ishte distanca, kjo realizohet përmes sensorëve të ndryshëm si LDS, kamera etj. si dhe pozicioni. Më poshtë do të shpjegojmë më gjerësisht si realizohet navigimi në ROS deri te arritja e matjes së distancës dhe pozicionit. Në kapitullin 3 sqaruam se si secili funksion(topics) ka nga një algoritëm të veçantë për kryerjen e një funksioni të caktuar që zakonisht janë të gatshëm dhe varësisht prej qëllimit që roboti duhet të kryejë i aplikojmë ato. Tani le të analizojmë funksionet (topics) që aplikohen në navigim:

- Odometria e robotit (/odom; nav_msgs/Odometry) – paraqet informata lidhur me odometrinë e robotit si shpejtësia aktuale, këndi i rrotullimit etj.
- Transformimi i koordinatave TF (/tf;tf/fMessage) – paraqet pozicionin e robotit përgjatë boshtit koordinativ x,y,z.
- Sensori i distancës (/scan;sensor_msgs/LaserScan) – paraqet vlerën e matur nga sensori i distancës, si dhe përdoret për të llogaritur pozicionin aktual të robotit ose të planifikojmë pozicionin e ardhshëm duke përdorur filtrin AMCL.
- Harta (/map; nav_msgs/GetMap) – ky funksion paraqet harta OGM të përdorur gjatë navigimit.

- Koordinata e synuara (/move_base_simple/goal; geometry_msgs/PoseStamped) – paraqet funksioni ku përdoruesi mund të caktojë koordinatat x,y,θ.
- Shpejtësia (/cmd_vel) – përdoret në rastet kur për arritjen e koordinatave të synuara caktojmë edhe shpejtësinë e robotit.

4.2.1 Navigimi me Turtlebot

Për të sqaruar më mirë navigimin në ROS marrim robotin Turtlebot. Për të startuar robotin ekzekutojmë file-at për startim (launch files) dhe më pas roboti është i gatshëm për navigim.

```
roslaunch turtlebot_gazebo turtlebot_world.launch
roslaunch turtlebot_navigation gmapping_demo.launch
```

Pasi kemi startuar simulatorin e Turtlebot, atëhere nëse listojmë funksionet (topics) kemi:

```
akrrabaj@akrrabaj:~$ rostopic list
/camera/depth/camera_info
/camera/depth/image_raw
/camera/depth/points
/camera/depth_registered/sw_registered/camera_info
/camera/depth_registered/sw_registered/image_rect_raw
/camera/depth_registered/sw_registered/image_rect_raw/compressed
/camera/depth_registered/sw_registered/image_rect_raw/compressed/parameter_descriptions
/camera/depth_registered/sw_registered/image_rect_raw/compressed/parameter_updates
/camera/depth_registered/sw_registered/image_rect_raw/compressedDepth/parameter_descriptions
/camera/depth_registered/sw_registered/image_rect_raw/compressedDepth/parameter_updates
/camera/depth_registered/sw_registered/image_rect_raw/theora
/camera/depth_registered/sw_registered/image_rect_raw/theora/parameter_descriptions
/camera/depth_registered/sw_registered/image_rect_raw/theora/parameter_updates
/camera/ir_rectify_ir/parameter_descriptions
/camera/parameter_descriptions
/camera/parameter_updates
```

```
/camera/rgb/camera_info
/camera/rgb/image_raw
/camera/rgb/image_raw/compressed
/camera/rgb/image_raw/compressed/parameter_descriptions
/camera/rgb/image_raw/compressed/parameter_updates
/camera/rgb/image_raw/compressedDepth
/camera/rgb/image_raw/compressedDepth/parameter_descriptions
/camera/rgb/image_raw/compressedDepth/parameter_updates
/camera/rgb/image_raw/theora
/camera/rgb/image_raw/theora/parameter_descriptions
/camera/rgb/image_raw/theora/parameter_updates
/cmd_vel_mux/active
/cmd_vel_mux/input/navi
/cmd_vel_mux/input/safety_controller
/cmd_vel_mux/input/switch
/cmd_vel_mux/input/teleop
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/set_link_state
/gazebo/set_model_state
/joint_states
/mobile_base/commands/motor_power
/mobile_base/commands/reset_odometry
/mobile_base/commands/velocity
/mobile_base/events/bumper
/mobile_base/events/cliff
/mobile_base/sensors/bumper_pointcloud
/mobile_base/sensors/core
/mobile_base/sensors/imu_data
/mobile_base_nodelet_manager/bond
/odom
```

```
/rosout
/rosout_agg
/tf
/tf_static
```

Në këtë rast shohim se janë krijuar edhe funksionet /odom, /tf, /cmd_vel, por nëse komandën e fundit e ekzekutojmë përsëri por specifikojmë hartën e krijohet një funksion i ri /map.

```
roslaunch turtlebot_navigation amcl_demo.launch map_file :=$HOME/fim_simulation.yaml
```

Paketat që mundësojnë navgimin me Turtlebot përmbajnë disa file startues (launch files), të cilat përmbajnë konfigurime të ndryshme si file-at për hartën, konfigurimet për simulatorin RVIZ dhe që zakonisht janë në formatin xml ose yaml të cilat mundësojnë përkthimin e një strukture të dhënash në një format që të dhënat mund të ruhen ose të transmetohen. Në këtë rast do të shyqyrtojmë file-at startues për navigim të Turtlebot në ROS.

4.2.1.1 Navigimi

```
akrrabaj@akrraba:~$ cat /opt/ros/indigo/share/turtlebot_navigation/package.xml
<package>
  <name>turtlebot_navigation</name>
  <version>2.3.7</version>
  <description>turtlebot_navigation</description>
  <maintainer email="turtlebot@osrfoundation.org">OSRF</maintainer>
  <license>BSD</license>
  <url type="website">http://ros.org/wiki/turtlebot_navigation</url>
  <url type="repository">https://github.com/turtlebot/turtlebot_apps</url>
  <url type="bugtracker">https://github.com/turtlebot/turtlebot_apps/issues</url>
  <author>Tully Foote</author>
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>tf</build_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>sensor_msgs</build_depend>
  <run_depend>tf</run_depend>
```

```
  <run_depend>roscpp</run_depend>
  <run_depend>sensor_msgs</run_depend>
  <run_depend>move_base</run_depend>
  <run_depend>map_server</run_depend>
  <run_depend>amcl</run_depend>
  <run_depend>gmapping</run_depend>
  <run_depend>turtlebot_bringup</run_depend>
  <run_depend>dwa_local_planner</run_depend>
  <export>
  </export>
</package>
```

4.2.1.2 AMCL

```
akrrabaj@akrrabaj:~$ cat /opt/ros/indigo/share/turtlebot_navigation/launch/includes/amcl/amcl.launch.xml
<launch>
  <arg name="use_map_topic"   default="false"/>
  <arg name="scan_topic"      default="scan"/>
  <arg name="initial_pose_x"  default="0.0"/>
  <arg name="initial_pose_y"  default="0.0"/>
  <arg name="initial_pose_a"  default="0.0"/>
  <arg name="odom_frame_id"   default="odom"/>
  <arg name="base_frame_id"   default="base_footprint"/>
  <arg name="global_frame_id" default="map"/>

  <node pkg="amcl" type="amcl" name="amcl">
    <param name="use_map_topic"             value="$(arg use_map_topic)"/>
    <!-- Publish scans from best pose at a max of 10 Hz -->
    <param name="odom_model_type"           value="diff"/>
    <param name="odom_alpha5"               value="0.1"/>
    <param name="gui_publish_rate"          value="10.0"/>
    <param name="laser_max_beams"             value="60"/>
    <param name="laser_max_range"           value="12.0"/>
    <param name="min_particles"             value="500"/>
```

```
    <param name="max_particles"             value="2000"/>
    <param name="kld_err"                   value="0.05"/>
    <param name="kld_z"                     value="0.99"/>
    <param name="odom_alpha1"               value="0.2"/>
    <param name="odom_alpha2"               value="0.2"/>
    <!-- translation std dev, m -->
    <param name="odom_alpha3"               value="0.2"/>
    <param name="odom_alpha4"               value="0.2"/>
    <param name="laser_z_hit"               value="0.5"/>
    <param name="laser_z_short"             value="0.05"/>
    <param name="laser_z_max"               value="0.05"/>
    <param name="laser_z_rand"              value="0.5"/>
    <param name="laser_sigma_hit"           value="0.2"/>
    <param name="laser_lambda_short"        value="0.1"/>
    <param name="laser_model_type"          value="likelihood_field"/>
    <!-- <param name="laser_model_type" value="beam"/> -->
    <param name="laser_likelihood_max_dist" value="2.0"/>
    <param name="update_min_d"              value="0.25"/>
    <param name="update_min_a"              value="0.2"/>
    <param name="odom_frame_id"             value="$(arg odom_frame_id)"/>
    <param name="base_frame_id"             value="$(arg base_frame_id)"/>
    <param name="global_frame_id"           value="$(arg global_frame_id)"/>
    <param name="resample_interval"         value="1"/>
    <!-- Increase tolerance because the computer can get quite busy -->
    <param name="transform_tolerance"       value="1.0"/>
    <param name="recovery_alpha_sloë"       value="0.0"/>
    <param name="recovery_alpha_fast"       value="0.0"/>
    <param name="initial_pose_x"            value="$(arg initial_pose_x)"/>
    <param name="initial_pose_y"            value="$(arg initial_pose_y)"/>
    <param name="initial_pose_a"            value="$(arg initial_pose_a)"/>
    <remap from="scan"                      to="$(arg scan_topic)"/>
  </node>
</launch>
```

4.2.1.3 Planifikimi i lëvizjes

```
akrrabaj@akrrabaj:~$ cat /opt/ros/indigo/share/turtlebot_navigation/launch/includes/move_base.launch.xml

<!--

    ROS navigation stack with velocity smoother and safety (reactive) controller

-->

<launch>

  <include file="$(find turtlebot_navigation)/launch/includes/velocity_smoother.launch.xml"/>

  <include file="$(find turtlebot_navigation)/launch/includes/safety_controller.launch.xml"/>


  <arg name="odom_frame_id"   default="odom"/>

  <arg name="base_frame_id"   default="base_footprint"/>

  <arg name="global_frame_id" default="map"/>

  <arg name="odom_topic" default="odom" />

  <arg name="laser_topic" default="scan" />

  <arg name="custom_param_file" default="$(find turtlebot_navigation)/param/dummy.yaml"/>


  <node pkg="move_base" type="move_base" respaën="false" name="move_base" output="screen">

    <rosparam file="$(find turtlebot_navigation)/param/costmap_common_params.yaml" command="load" ns="global_costmap" />

    <rosparam file="$(find turtlebot_navigation)/param/costmap_common_params.yaml" command="load" ns="local_costmap" />

    <rosparam file="$(find turtlebot_navigation)/param/local_costmap_params.yaml" command="load" />

    <rosparam file="$(find turtlebot_navigation)/param/global_costmap_params.yaml" command="load" />

    <rosparam file="$(find turtlebot_navigation)/param/dëa_local_planner_params.yaml" command="load" />

    <rosparam  file="$(find  turtlebot_navigation)/param/move_base_params.yaml" command="load" />

    <rosparam file="$(find turtlebot_navigation)/param/global_planner_params.yaml" command="load" />

    <rosparam file="$(find turtlebot_navigation)/param/navfn_global_planner_params.yaml" command="load" />
```

```
    <!-- external params file that could be loaded into the move_base namespac
e -->
    <rosparam file="$(arg custom_param_file)" command="load" />

    <!-- reset frame_id parameters using user input data -->
    <param name="global_costmap/global_frame" value="$(arg global_frame_id)"/>
    <param name="global_costmap/robot_base_frame" value="$(arg base_frame_id)"
/>
    <param name="local_costmap/global_frame" value="$(arg odom_frame_id)"/>
    <param name="local_costmap/robot_base_frame" value="$(arg base_frame_id)"/
    <param name="DËAPlannerROS/global_frame_id" value="$(arg odom_frame_id)"/>

    <remap from="cmd_vel" to="navigation_velocity_smoother/raw_cmd_vel"/>
    <remap from="odom" to="$(arg odom_topic)"/>
    <remap from="scan" to="$(arg laser_topic)"/>
  </node>



</launch>
```

4.2.1.4 Costmap

Siç u shpjegua më lartë, navigimi përdor OGM hartat për lëvizje. Duke u bazuar në këto harta secili piksel llogaritet si një pengesë, hapësire e bllokuar ose hapësire e lirë varësisht prej vlerave të lexuara nga sensori, kjo llogaritje njihet si costmap.

```
akrrabaj@akrrabaj:~$ cat /opt/ros/indigo/share/turtlebot_navigation/param/cost
map_common_params.yaml
max_obstacle_height: 0.60  # assume something like an arm is mounted on top of
the robot


# Obstacle Cost Shaping (http://wiki.ros.org/costmap_2d/hydro/inflation)
robot_radius: 0.20  # distance a circular robot should be clear of the obstacl
e (kobuki: 0.18)
# footprint: [[x0, y0], [x1, y1], ... [xn, yn]]  # if the robot is not circula
r
```

```
map_type: voxel


obstacle_layer:
  enabled:              true
  max_obstacle_height:  0.6
  origin_z:             0.0
  z_resolution:         0.2
  z_voxels:             2
  unknoën_threshold:    15
  mark_threshold:       0
  combination_method:   1
  track_unknoën_space:  true    #true needed for disabling global path plannin
g through unknoën space
  obstacle_range: 2.5
  raytrace_range: 3.0
  origin_z: 0.0
  z_resolution: 0.2
  z_voxels: 2
  publish_voxel_map: false
  observation_sources:  scan bump
  scan:
    data_type: LaserScan
    topic: scan
    marking: true
    clearing: true
    min_obstacle_height: 0.25
    max_obstacle_height: 0.35
  bump:
    data_type: PointCloud2
    topic: mobile_base/sensors/bumper_pointcloud
    marking: true
    clearing: false
    min_obstacle_height: 0.0
```

```
    max_obstacle_height: 0.15

  # for debugging only, let's you see the entire voxel grid


#cost_scaling_factor and inflation_radius were now moved to the inflation_laye
r ns

inflation_layer:

  enabled:              true

  cost_scaling_factor:  5.0  # exponential rate at which the obstacle cost dro
ps off (default: 10)

  inflation_radius:     0.5  # max. distance from an obstacle at which costs a
re incurred for planning paths.

static_layer:

  enabled:              true
```

4.2.1.5 DWA

DWA paraqet një strategji të shmangies së përplasjeve në kohë reale tek robotët mobil. Kjo paketë paraqet konfigurimet lidhur me shpejtësinë e robotit. Algoritmi i kësaj ka për qëllim që të caktojë shpejtësinë e duhur më qëllim që roboti të arrijë sa më shpejtë pikën e caktuar.

```
akrrabaj@akrrabaj:~$ cat /opt/ros/indigo/share/turtlebot_navigation/param/dëa_
local_planner_params.yaml

DWAPlannerROS:

# Robot Configuration Parameters - Kobuki

  max_vel_x: 0.5  # 0.55

  min_vel_x: 0.0


  max_vel_y: 0.0  # diff drive robot

  min_vel_y: 0.0  # diff drive robot


  max_trans_vel: 0.5 # choose slightly less than the base's capability

  min_trans_vel: 0.1  # this is the min trans velocity ëhen there is negligibl
e rotational velocity

  trans_stopped_vel: 0.1


  # Warning!
```

```
  #   do not set min_trans_vel to 0.0 otherwise dwa will always think translat
ional velocities
  #   are non-negligible and small in place rotational velocities ëill be crea
ted.


  max_rot_vel: 5.0  # choose slightly less than the base's capability
  min_rot_vel: 0.4  # this is the min angular velocity when there is negligibl
e translational velocity
  rot_stopped_vel: 0.4


  acc_lim_x: 1.0 # maximum is theoretically 2.0, but ëe
  acc_lim_theta: 2.0
  acc_lim_y: 0.0      # diff drive robot


# Goal Tolerance Parameters
  yaw_goal_tolerance: 0.3  # 0.05
  xy_goal_tolerance: 0.15  # 0.10
  # latch_xy_goal_tolerance: false


# Forward Simulation Parameters
  sim_time: 1.0       # 1.7
  vx_samples: 6       # 3
  vy_samples: 1       # diff drive robot, there is only one sample
  vtheta_samples: 20  # 20
# Trajectory Scoring Parameters
  path_distance_bias: 64.0      # 32.0   - weighting for how much it should st
ick to the global path plan
  goal_distance_bias: 24.0      # 24.0   - weighting for how much it should at
tempt to reach its goal
  occdist_scale: 0.5            # 0.01   - weighting for how much the controll
er should avoid obstacles
  forward_point_distance: 0.325 # 0.325  - how far along to place an additiona
l scoring point
  stop_time_buffer: 0.2         # 0.2    - amount of time a robot must stop in
before colliding for a valid traj.
  scaling_speed: 0.25           # 0.25   - absolute velocity at which to start
scaling the robot's footprint
```

```
  max_scaling_factor: 0.2       # 0.2    - how much to scale the robot's footp
rint when at speed.

# Oscillation Prevention Parameters

  oscillation_reset_dist: 0.05  # 0.05   - how far to travel before resetting
oscillation flags

# Debugging

  publish_traj_pc : true

  publish_cost_grid_pc: true

  global_frame_id: odom

# Differential-drive robot configuration - necessary?

#  holonomic_robot: false
```

4.3 SLAM në ROS

Për të krijuar harta në SLAM nevojiten po ashtu disa paketa në rastin tonë do të përdorim *turtlebot_bringup minimal.launch* dhe *turtlebot_navigation gmapping_demo.launch*, si dhe për vizualizim të rezultateve *turtlebot_rviz_launchers view_navigation.launch*. Kur startojmë këto dhe i listojmë subjektet (topics) si dhe duke u bazuar në diagramin për realizimin e SLAM *fig 4.3* atëherë shohim së i kemi të gjitha komponentët për të realizuar SLAM në ROS.

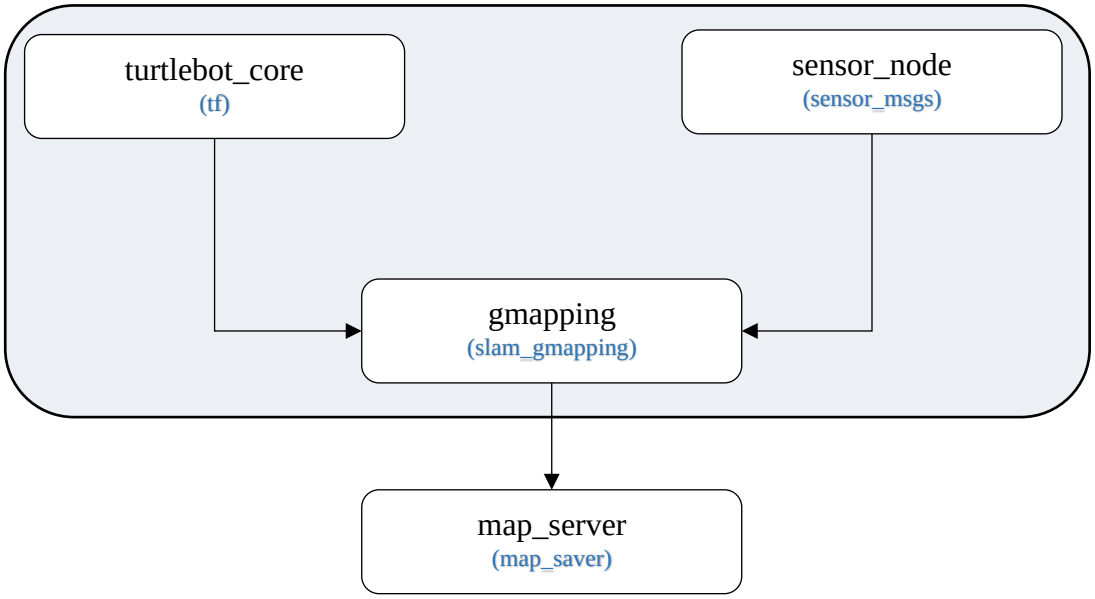

Fig 4. 3 Diagrami i SLAM

- /scan – paraqet vlerat e marra nga sensori
- /turtlebot_teleop përmban komandat për lëvizjen e turtlebot
- /turtlebot – paraqet robotin që është i gatshëm të pranojë dhe dërgoj informata
- /slam_gmapping – krijon hartën duke u bazuar nga të dhënat e matura nga sensori
- /map – mundëson ruajtjen e OGM hartave.

Më poshtë janë paraqitur të gjitha subjektet (topics) të krijuara pas startimit të turtlebot për të gjeneruar harta përmes SLAM, proces i cili do të simulohet në kapitullin 5.

```
akrrabaj@akrrabaj:~$ rostopic list
/capability_server/bonds
/capability_server/events
/cmd_vel_mux/active
/cmd_vel_mux/input/navi
/cmd_vel_mux/input/safety_controller
/cmd_vel_mux/input/sëitch
/cmd_vel_mux/input/teleop
/cmd_vel_mux/parameter_descriptions
/cmd_vel_mux/parameter_updates
/depthimage_to_laserscan/parameter_descriptions
/depthimage_to_laserscan/parameter_updates
/diagnostics
/diagnostics_agg
/diagnostics_toplevel_state
/gateway/force_update
/gateway/gateway_info
/info
/interactions/interactive_clients
/interactions/pairing
/joint_states
/kobuki_safety_controller/disable
/kobuki_safety_controller/enable
/kobuki_safety_controller/reset
/laptop_charge
```

```
/map
/map_metadata
/mobile_base/commands/controller_info
/mobile_base/commands/digital_output
/mobile_base/commands/external_poëer
/mobile_base/commands/led1
/mobile_base/commands/led2
/mobile_base/commands/motor_power
/mobile_base/commands/reset_odometry
/mobile_base/commands/sound
/mobile_base/commands/velocity
/mobile_base/controller_info
/mobile_base/debug/raw_control_command
/mobile_base/debug/raw_data_stream
/mobile_base/events/bumper
/mobile_base/events/button
/mobile_base/events/cliff
/mobile_base/events/digital_input
/mobile_base/events/power_system
/mobile_base/events/robot_state
/mobile_base/events/wheel_drop
/mobile_base/sensors/bumper_pointcloud
/mobile_base/sensors/core
/mobile_base/sensors/dock_ir
/mobile_base/sensors/imu_data
/mobile_base/version_info
/mobile_base_nodelet_manager/bond
/move_base/current_goal
/move_base/goal
/move_base_simple/goal
/navigation_velocity_smoother/parameter_descriptions
/navigation_velocity_smoother/parameter_updates
/navigation_velocity_smoother/raw_cmd_vel
```

```
/odom
/rosout
/rosout_agg
/scan
/slam_gmapping/entropy
/tf
/tf_static
/turtlebot/incompatible_rapp_list
/turtlebot/rapp_list
/turtlebot/status
/zeroconf/lost_connections
/zeroconf/new_connections
```

4.2.1.3 Gmapping

Kurse më poshtë është paraqitur konfigurimi për realizimin e gmapping në ROS

```
akrrabaj@akrrabaj:~$ cat /opt/ros/indigo/share/turtlebot_navigation/launch/inc
ludes/gmapping/gmapping.launch.xml
<launch>
  <arg name="scan_topic"  default="scan" />
  <arg name="base_frame"  default="base_footprint"/>
  <arg name="odom_frame"  default="odom"/>
  <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping" output="scree
n">
    <param name="base_frame" value="$(arg base_frame)"/>
    <param name="odom_frame" value="$(arg odom_frame)"/>
    <param name="map_update_interval" value="5.0"/>
    <param name="maxUrange" value="6.0"/>
    <param name="maxRange" value="8.0"/>
    <param name="sigma" value="0.05"/>
    <param name="kernelSize" value="1"/>
    <param name="lstep" value="0.05"/>
    <param name="astep" value="0.05"/>
    <param name="iterations" value="5"/>
```

```
    <param name="lsigma" value="0.075"/>
    <param name="ogain" value="3.0"/>
    <param name="lskip" value="0"/>
    <param name="minimumScore" value="200"/>
    <param name="srr" value="0.01"/>
    <param name="srt" value="0.02"/>
    <param name="str" value="0.01"/>
    <param name="stt" value="0.02"/>
    <param name="linearUpdate" value="0.5"/>
    <param name="angularUpdate" value="0.436"/>
    <param name="temporalUpdate" value="-1.0"/>
    <param name="resampleThreshold" value="0.5"/>
    <param name="particles" value="80"/>
  <!--
    <param name="xmin" value="-50.0"/>
    <param name="ymin" value="-50.0"/>
    <param name="xmax" value="50.0"/>
    <param name="ymax" value="50.0"/>
  make the starting size small for the benefit of the Android memory  -->
    <param name="xmin" value="-1.0"/>
    <param name="ymin" value="-1.0"/>
    <param name="xmax" value="1.0"/>
    <param name="ymax" value="1.0"/>
    <param name="delta" value="0.05"/>
    <param name="llsamplerange" value="0.01"/>
    <param name="llsamplestep" value="0.01"/>
    <param name="lasamplerange" value="0.005"/>
    <param name="lasamplestep" value="0.005"/>
    <remap from="scan" to="$(arg scan_topic)"/>
  </node>
</launch>
```

5 MODELIMI DHE SIMULIMI I SLAM (GAZEBO DHE MATLAB)

Në kapitujt më lartë sqaruam çka është SLAM, si realizohet në mënyrë analitike, si është bërë integrimi në ROS. Në ketë kapitull do të përdorim dy aplikacione për të bërë modelimin dhe simulimin e këtij procesi. Aplikacionet janë Gazebo dhe Matlab.

5.1 Konfigurimi i komunikimit ndërmjet Gazebo dhe Matlab

Komunikimi ndërmjet këtyre dy aplikacioneve zakonisht bëhet përmes protokolleve TCP/IP [30], në këtë rast për të evituar problemet me subnetime të rrjetave atëherë i vendosim edhe Turtlebot dhe PC ku ekzekutojmë Matlab në të njëjtën subnet. Në Ubuntu ku është ROS ekzekutojmë:

```
akrrabaj@akrrabaj:~$ ifconfig
enp0s17: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.15  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::f3ea:a452:40c3:d764  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:d7:71:d4  txqueuelen 1000  (Ethernet)
        RX packets 824  bytes 945131 (945.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 457  bytes 58220 (58.2 KB)
```

Gjithashtu janë dy variabla kryesorë që na mundësojnë komunikimin me ROS:

1. ROS_MASTER_URI – paraqet linkun dhe portin ku startohet ROS
2. ROSIP – paraqet IP e sistemit operativ ku është i hostuar ROS

Në këtë rast këto variabla duhet të jenë të eksportuara sa herë që krijojmë një sesion dhe më qëllim të evitimit punës manuale i vendosim në ~/home/akrrabaj/.bashrc

```
echo export ROS_MASTER_URI=http:// 192.168.1.18:11311 >> ~/.bashrc
echo export ROS_IP=192.168.1.18>> ~/.bashrc
```

Pasi kemi startuar turtlebot në ROS, tani mund të provojmë komunikimin ndërmjet MATLAB dhe ROS, në Matlab deklarojmë dy variablat kryesore që u përmenden më herët:

```
>> ipaddress = "http://192.168.1.15:11311";
>> rosinit(ipaddress)
Initializing global node /matlab_global_node_77549 with NodeURI http://192.168
.1.8:52262/
```

Shohim se incializimi është kryer me sukses, pas këtij hapi do të bëjmë një skenim prej 60 sekondash me Lidar sensorin duke thirrur subjektin "/scan". Në matlab ekzekutojmë:

```
>>lidarSub = rossubscriber("/scan","DataFormat","struct");
>>tic
while toc < 300
  scanMsg = receive(lidarSub);
  rosPlot(scanMsg)
end
```

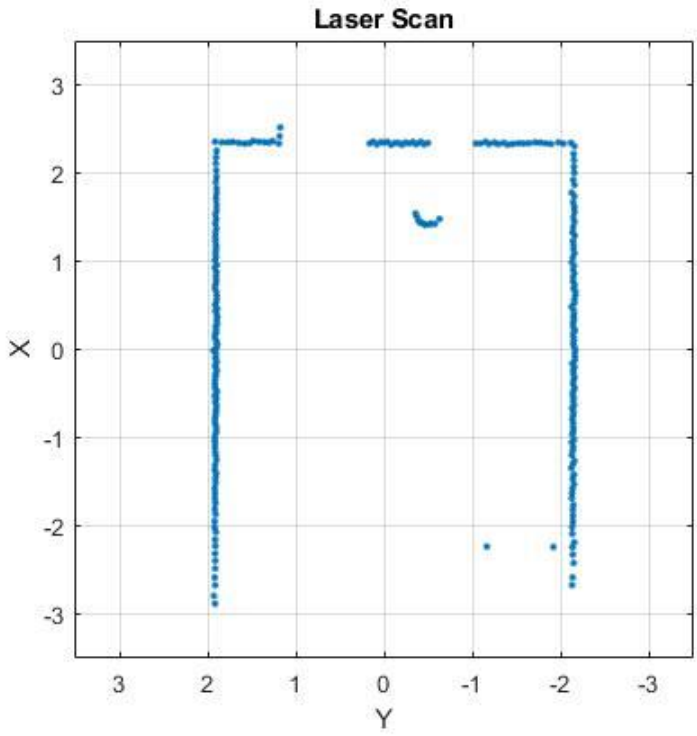

Fig 5. 1 Skenimi me Lidar përmes Matlab

5.2 Simulimi i SLAM në ROS

Për të simuluar SLAM në ROS ne do të shfrytëzojmë paketat të gatshme të ofruara nga ROS. Kemi tri paketa kryesore për startimin e robotit *turtlebot_world.launch*, për algoritëm të SLAM *gmapping_demo.launch*, për paraqitje vizuale *view_navigation.launch* si dhe për teleoperim *keyboard_teleop.launch*. Në ROS hapim katër terminale dhe ekzekutojmë [24]:

```
roslaunch turtlebot_gazebo turtlebot_world turtlebot_world.launch
roslaunch turtlebot_gazebo gmapping_demo.launch
roslaunch turtlebot_rviz_launchers view_nagivation.launch
roslaunch tutlebot_teleop keyboard_teleop.launch
```

Pas navigimit nëpër gjithë hartën i cili ka zgjatur më shumë se 2 orë, ruajmë hartën si dhe hapim AMCL për navigimin nëpër hartë:

```
rosrun map_server map_saver -f /home/akrrabaj/fim_simulation
roslaunch turtlebot_navigation amcl_demo.launch map_file:=/home/akrrabaj/fim_s
imulation.yaml
```

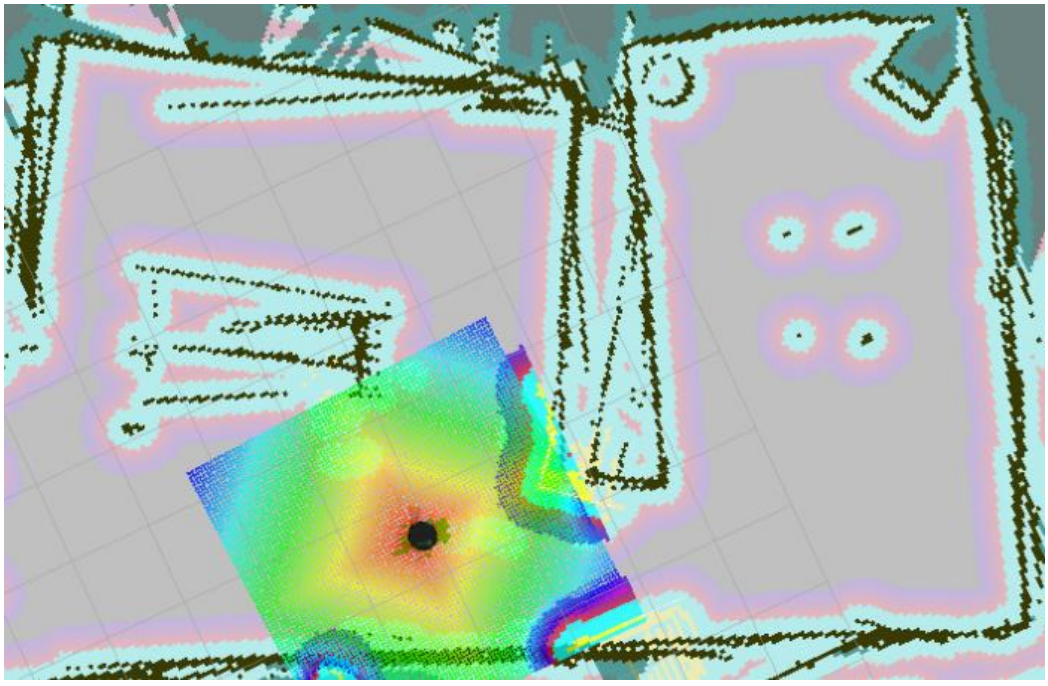

Fig 5. 2 Simulimi i SLAM dhe krijimi i hartës në ROS

5.3 Simulimi i SLAM në MATLAB

Në kaptullin 4.1 shpjeguam se çka janë OGM harta si krijohen përgjatë navigimit dhe si t'i ruajmë ato. Kurse sa i përket lëvizjeve sqaruam disa filtra e pamë se një rendësi të veçantë kishte filtri MCL përkatësisht AMCL. Më poshtë do të bëjmë simulimin e SLAM [35] ku do të shfrytëzojmë një hartë të gatshme nga libraria e ROS "*offlineSlamData.mat"* e cila përmban vlerat e lexuara nga LiDAR sensori i vendosur në Turtlebot, gjithashtu krijojmë një objekt *lidarSLAM* i cili përfshin informata si vlerën maksimale që lexon sensori *maxLidarRange* si dhe *mapResolution* e cila paraqet hartën e rrjetëzuar ku secili katror 20 qelia janë të barabartë më një metër si dhe krijojmë një unazë loop. Në Matlab ekzekutojmë [30]:

```
>>load('offlineSlamData.mat');
>> maxLidarRange = 8;
>>mapResolution = 20;
>>slamAlg = lidarSLAM(mapResolution, maxLidarRange);
>>slamAlg.LoopClosureThreshold = 210;
>>slamAlg.LoopClosureSearchRadius = 8;
```

Tani le të startojmë skanimin nëpër hartë dhe planifikimin e trajektores së robotit.

```
>> for i=1:50
    [isScanAccepted, loopClosureInfo, optimizationInfo] = addScan(slamAlg, sca
ns{i});
    if isScanAccepted
        fprintf('Added scan %d \n', i);
   end
end
```

Pasi janë kryer skenimet atëherë ato mund t'i vizualizojmë:

```
>> figure;
show(slamAlg);
title({'Harta per mjedisin e ROS','Grafiku i pozicionit per 50 skenime'});
```

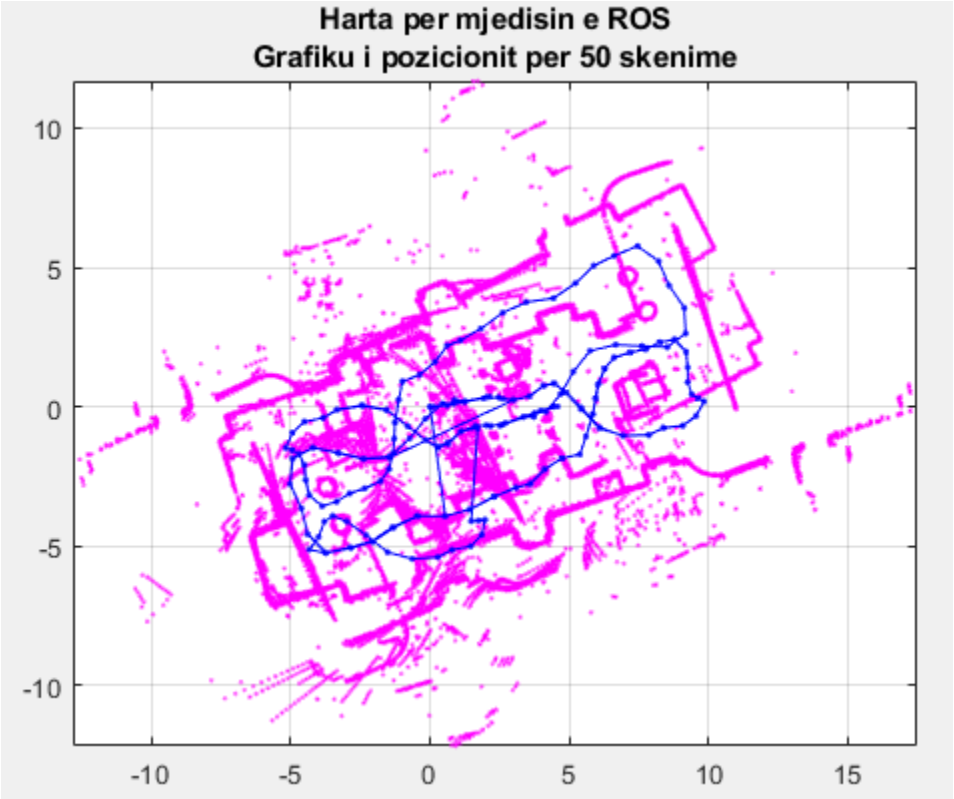

Fig 5. 3 Simulimi i SLAM dhe krijimi i hartës në Matlab

6 ANALIZA DHE KRAHASIMI I REZULTATEVE TË FITUARA ME ROBOTIN REAL

Më lartë sqaruam pothuajse të gjitha elementet e SLAM, krijuam simulimin e hartave në një mjedis virtual etj. në këtë kapitull do të realizojmë SLAM duke përdorur robotin fizik Turtlebot 2i. Në fillim do të konfigurojmë mjedisin, pastaj krijimin e hartës duke përdorur SLAM dhe në fund navigimit autonom nëpër mjedis.

6.1 Konfigurimi i mjedisit ndërmjet ROS dhe Turtlebot 2i

Pasi që na nevojitet krijimi i hartës në kohë reale si dhe ekzekutimi i SLAM dhe shfaqja e rezultateve po ashtu në kohë reale, paraqitet nevoja e konfigurimit të komunikimit ndërmjet ROS i cili është i instaluar në kompjuter personal dhe i referohemi si klient dhe ROS që është i instaluar në Turtlebot2i. Roboti fizik do të jetë master node kurse klient do të jetë ROS i cili është i instaluar në PC. Duke ju referuar kapitullit 3.5 vendosim IP statike në të dy sistemet operative ku:

1. 192.168.0.194 – Turtlebot2i
2. 192.168.0.195 – ROS klienti.

Në Turtlebot2i vendosim variablat globale:

```
echo export ROS_MASTER_URI=http:// 192.168.0.194:11311 >> ~/.bashrc
echo export ROS_HOSTNAME=192.168.0.194>> ~/.bashrc
source ~/.bashrc
```

Po ashtu në klient i eksportojmë këto variabla por MASTER_URI do të jetë roboti fizik.

```
echo export ROS_MASTER_URI=http:// 192.168.0.194:11311 >> ~/.bashrc
echo export ROS_HOSTNAME=192.168.0.195>> ~/.bashrc
source ~/.bashrc
```

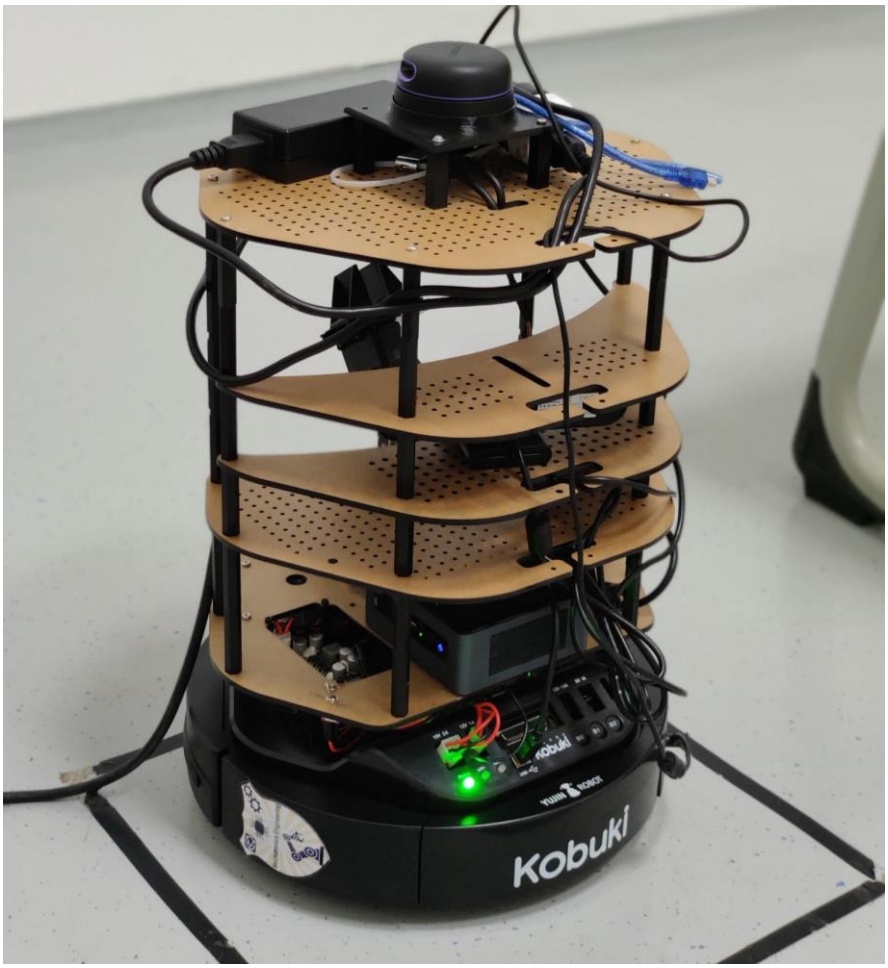

Fig 6. 1. Roboti Fizik Turtlebot 2i

6.2 SLAM në mjedis real

Roboti fizik është vendosur në një laborator të Mekatronikës ku do të ekzekutojmë SLAM si dhe më pas navigojmë me robot nëpër laborator derisa të arrijmë rezultate të kënaqshme të hartës së krijuar. Në robotin fizik Turtlebot2i hapim dy terminale startojmë robotin dhe algoritmin gmapping:

```
roslaunch turtlebot_bringup minimal.launch
roslaunch turtlebot_navigation gmapping_demo.launch
```

Në klient startojmë RVIZ për vizualizim të rezultateve si dhe launch file-in për teleoperim.

```
roslaunch turtlebot_rviz_launchers view_navigation.launch
roslaunch turtlebot_teleop keyboard_teleop.launch
```

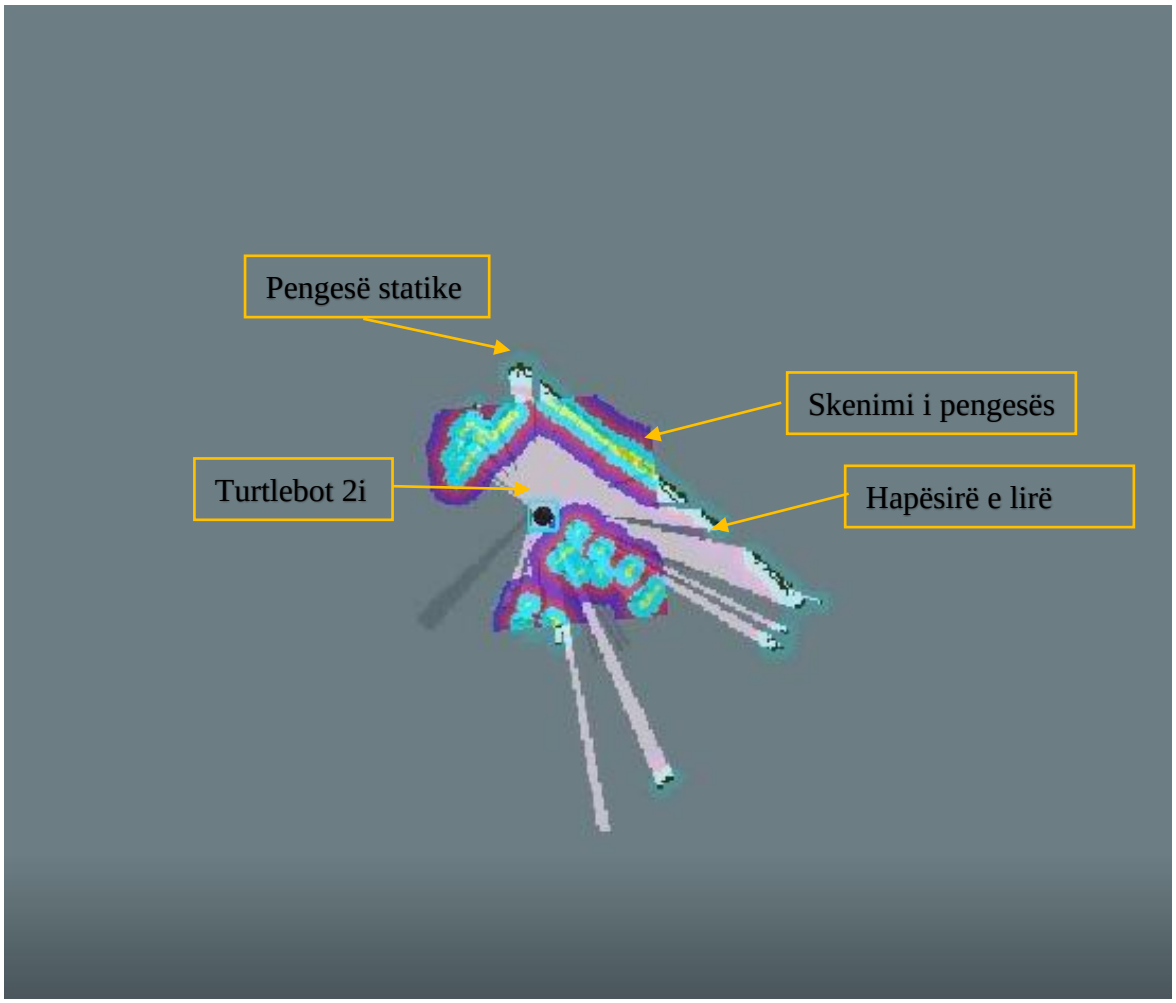

Fig 6. 2Rezultatet e para të SLAM pas navgimit për 30 sekondave

Për të arritur rezultate më të mira në këtë projekt është naviguar nëpër të gjithë laboratorin si dhe në pjesë të hollit hyrës të laboratorit. Gjatë skenimit shpesh është dashur që roboti të ndalet në një pikë dhe të bëhet skenimi në 360º i mjedisit më qëllim të krijimit sa më të saktë të pozicionit të robotit në raport me pengesat statike.

Skenimi ka zgjatur rreth 30min por sa më e gjatë të jetë koha e navigimit të krijimit të hartës aq më e saktë do të ishte harta e krijuar. Kështu pasi kemi arritur rezultatet e duhura ruajmë hartën ku në Turtlebot ekzekutojmë:

```
rosrun map_server map_saver -f /home/akrrabaj/fim_simulation2
```

Më poshtë është paraqitur përmbajtja e fim_simulation2.yaml

```
akrrabaj@akrrabaj:~$ cat fim_simulation2.yaml
image: /home/akrrabaj/fim_simulation2.pgm
resolution: 0.050000
origin: [-23.400000, -12.200000, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

Kurse në fig 6.3 harta finale e krijuar përmes SLAM.

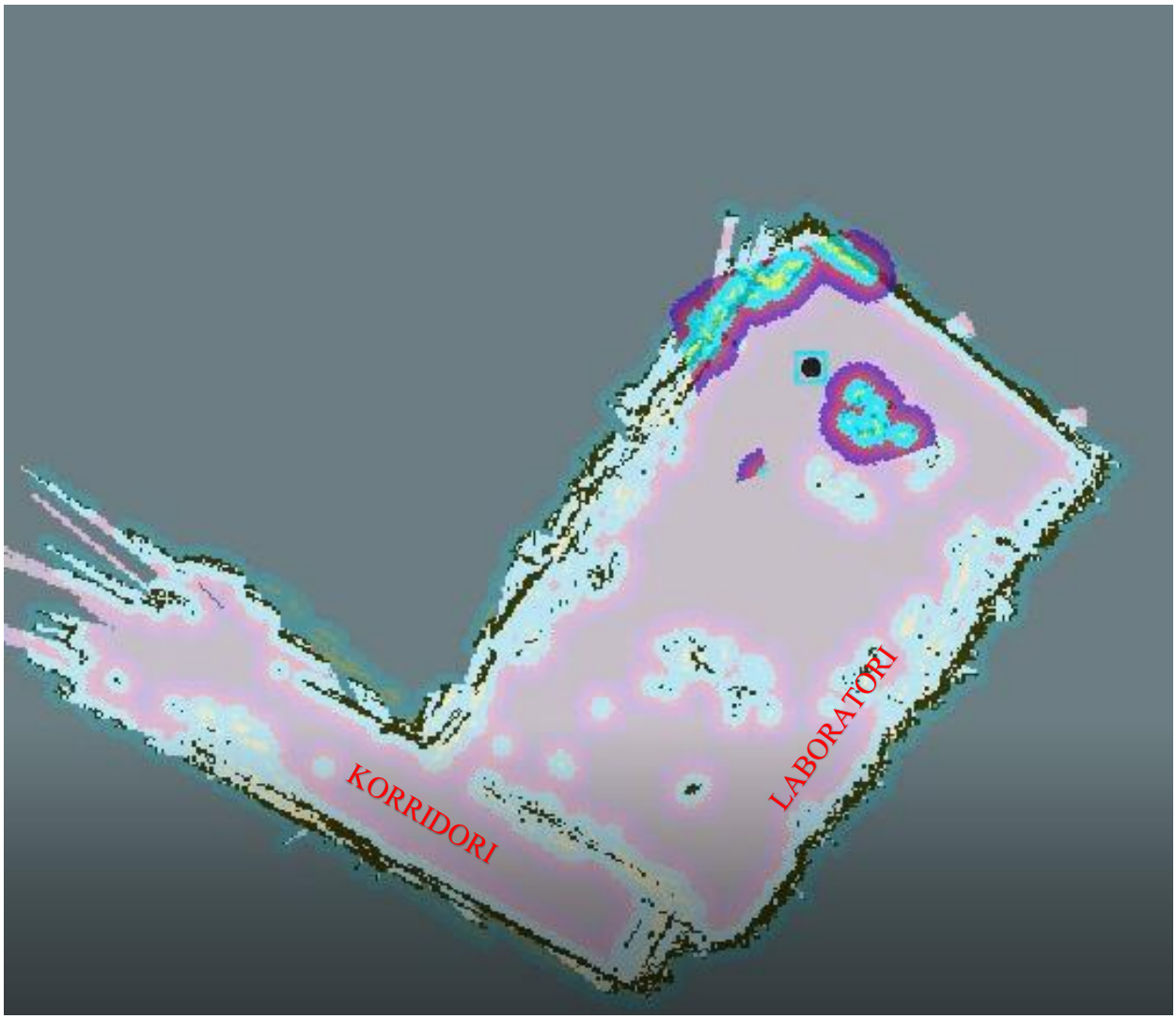

Fig 6. 3Harta e krijuar pas përfundimit të SLAM

6.3 Navigimi autonom i robotit fizik Turtlebot 2i

Hartën finale të krijuar përmes SLAM mund t'a shfrytëzojmë për qëllime të ndryshme. Duke u bazuar në qëllimin e punimit përmes hartës së krijuar do të caktojmë një pikë të dëshiruar në të cilën roboti duhet të navigojë deri në atë pikë në mënyrë autonome. Kështu në Turtlebot startojmë robotin dhe filtrin AMCL ku ekzekutojmë:

```
roslaunch turtlebot_bringup minimal.launch

roslaunch turtlebot_navigation amcl_demo.launch map_file:=/home/akrrabaj/fim_simulation2.yaml
```

Kurse në klient startojmë RVIZ për vizualizim:

```
roslaunch turtlebot_rviz_launchers view_navigation.launch --screen
```

Në figurën 6.4 është paraqitur filtri AMCL, ku secila shigjetë paraqet një mostër (e sqaruar në kapitullin 2.2.5.2) përkatësisht një probabilitet të lëvizjes.

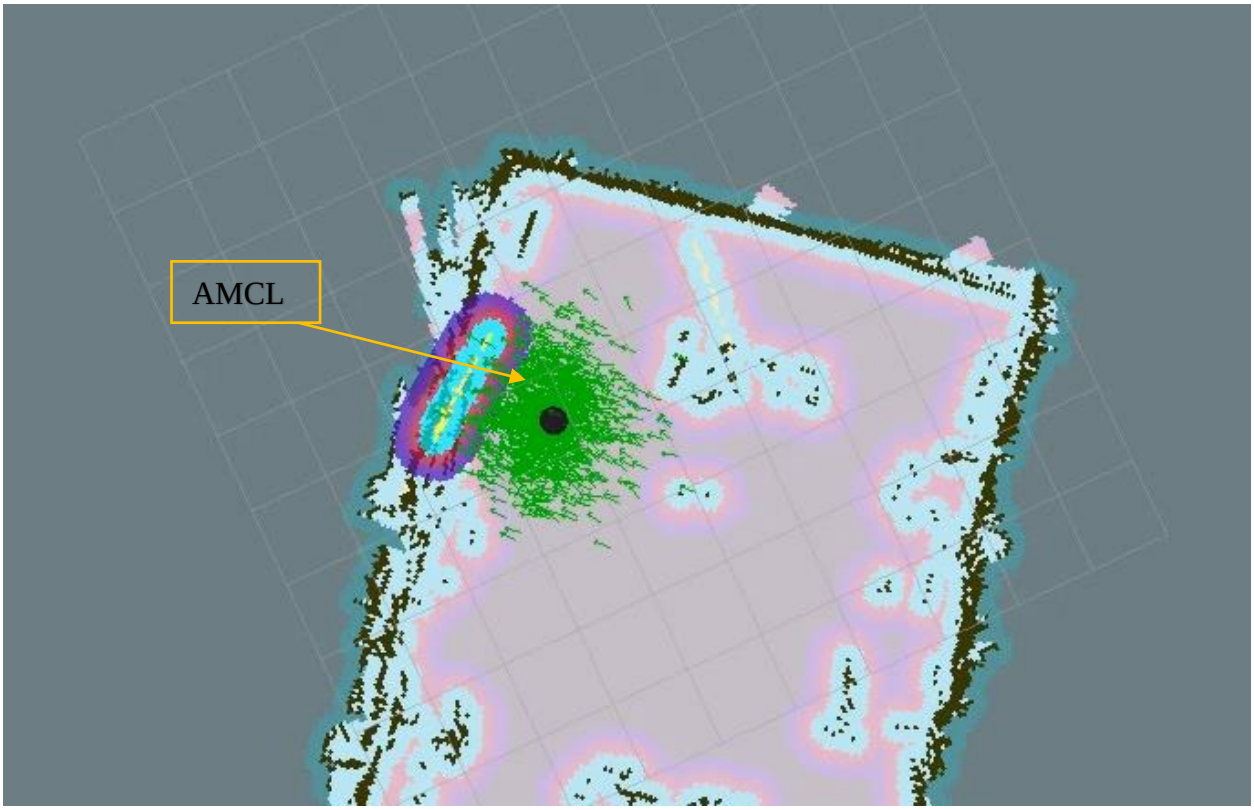

Fig 6. 4Paraqitja vizuale e AMCL

Pasi kemi ekzekutuar komandat më lartë roboti Turtlebot nuk është në gjendje të detektojë pozicionin në hartë atëherë duke shfrytëzuar aplikacionin RVIZ caktojmë pozicionimin e robotit në hartë si dhe drejtimin e robotit, RVIZ>2D Post Estimate> Selektojmë pikën ku gjendet roboti. Pas këtij hapi shumë lehtë mund të caktojmë pikën ose destinacionin e dëshiruar, ku në shembulli tonë është caktuar holli (korridori) i laboratorit ku selektojmë në RVIZ>2D Nav Goal> Selektojmë pikën ku dëshirojmë të navigojë roboti. Në fig 6.5 janë përshkruar disa elemente të RVIZ si dhe caktimi i pikës së dëshiruar.

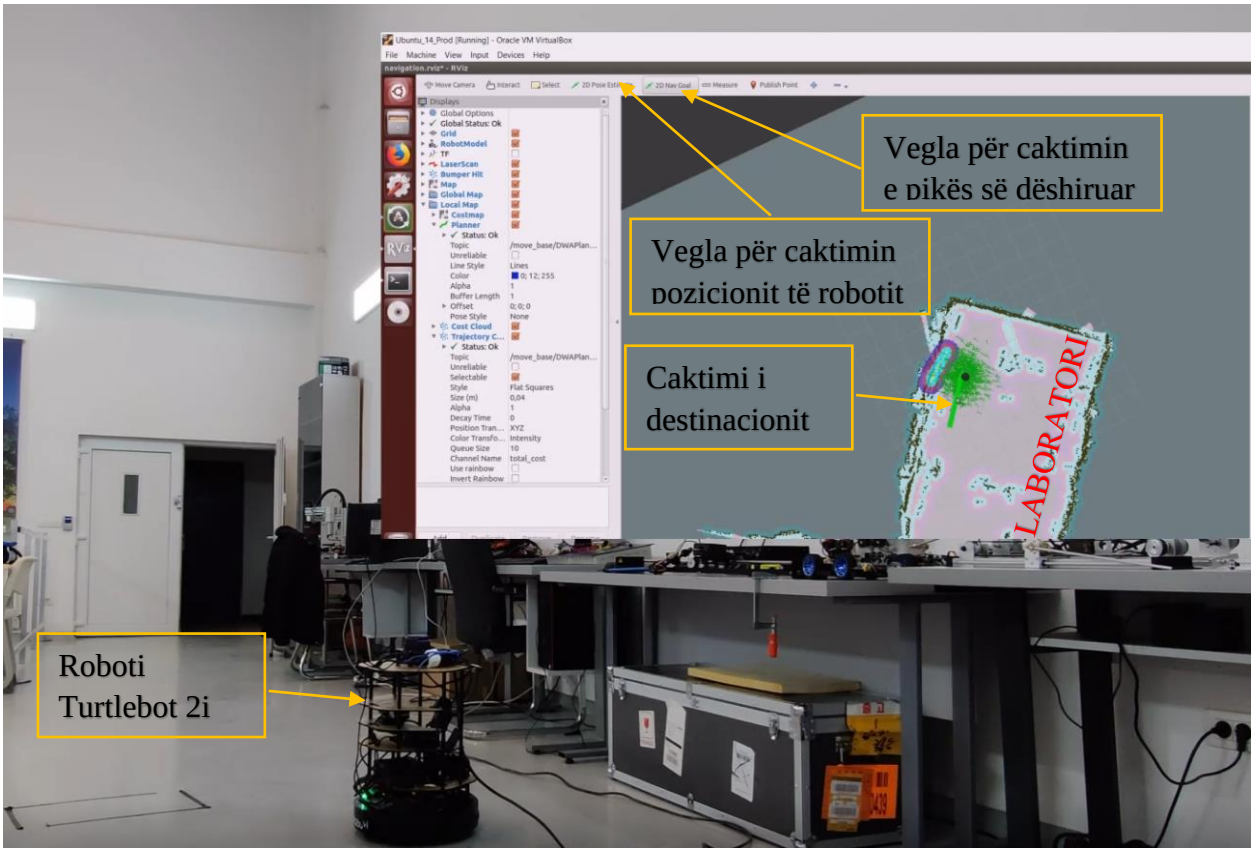

Fig 6. 5 Caktimi i pikës (destinacionit) të dëshiruar për navigim autonom

Pasi kemi caktuar pikën e dëshiruar roboti fizik fillon të llogarisë pozicionin aktual pas asaj krijon disa vlera të mostrave AMCL përreth tij dhe përmes tyre kalkulon se cilat janë mundësitë e lira për lëvizje dhe ky kalkulim vazhdon deri në arritjen e destinacionit final të paraqitu në fig 6.6.

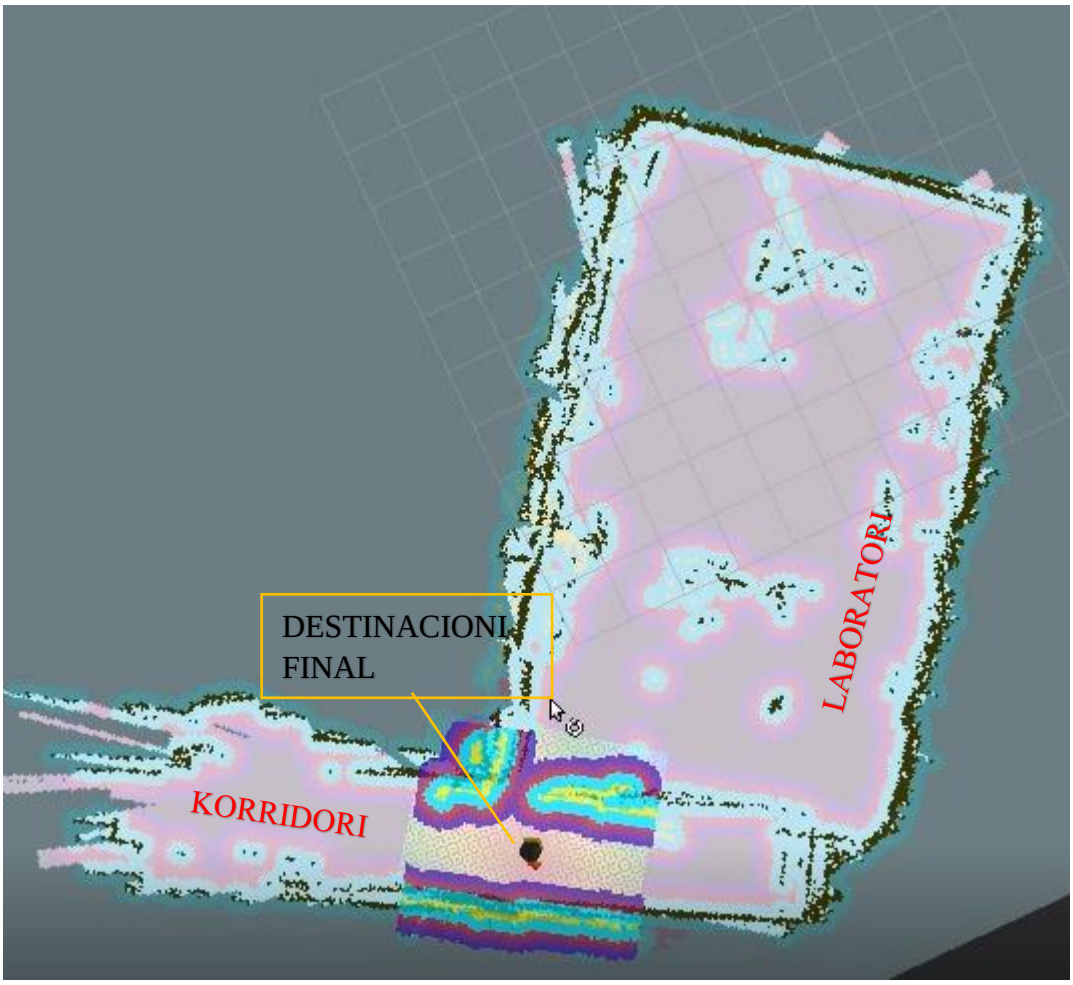

Fig 6. 6 Navigimi Autonom (Arritja në pikën e dëshiruar)

Pas realizimit të SLAM në mjedis real nëse referohemi në fig 6.6 shohim se ky proces është mjaftueshëm i saktë por ka përparësitë dhe mangësitë e veta të listuara më poshtë:

Tabela 6.1 Përparësitë dhe mangësitë e SLAM

	PËRPARËSITË	**MANGËSITË**
SLAM	OGM Harta të sakta	Pamundësia e përditësimit të hartave
	Simulimi në kohë reale	Kërkohet fuqi e lartë kompjuterike
	Manovrim i lehtë përmes aplikacionit	Navigimi në mënyrë manuale
	Saktësia e matjes së distancës në hapësira të brendshme	Pozicionimi i robotit në mënyrë manuale
	Detektimi i pengesave dinamike	Ndjeshmëria ndaj dritës (tek kamerat)
	Hartat e krijuara mund të ruhen	Pamundësia e detektimit të kushteve klimatike (tek lidarët)

7 PËRFUNDIM

Si u shpjegua edhe në kapitullin e parë të këtij punimi SLAM paraqet njërën nga metodat më të avancuara për navgimin nëpër mjedise të mbyllura ku saktësia duhet të jetë disa cm. Por që kjo të realizohet nevojiten të merren shumë informacione për ambientin që rrethohet roboti si dhe këto informata të kalojnë nëpër shumë algoritme të ndryshme deri në arritjen e qëllimit. Në këtë punim fillimisht është bërë përshkrimi analitik i SLAM ku sqaruam se parimisht SLAM bazohet në filtrin EKF por që ky nuk është i mjaftueshëm kur harta fillon të rritet në këtë rast është shfaqur nevoja e përdorimit edhe shumë filtrave tjera si AMCL, Gmapping, A* etj. dhe varësisht prej qëllimit të robotit bëhet edhe përdorimi i tyre. Në rastin tonë kemi përdorur filtrin Gmapping për krijimin e hartave dhe AMCL për navigim autonom. Por për të gjithë këta filtra nevojitet fuqi kompjuterike për përpunim të të dhënave kështu kemi parë rëndësinë dhe efikasitetin e ROS në robotikë i cili tashmë ka filluar të bëhet edhe një standard. Gjithashtu është shpjeguar se sa e rëndësishme është që procesin e modeluar të kemi mundësinë të simulojmë dhe të arrijmë rezultate para se të kalojmë në aplikim në mjedisin real. Roboti Turtlebot 2i është njëri ndër modelet më të përdorura të familjes Turtlebot si dhe në ROS, për realizimin e qëllimit të punimit përmes këtij roboti arritëm të konfigurojmë, simulojmë dhe aplikojmë SLAM procesin në mënyrë të suksesshme, ku më pas arritëm që të realizojmë lëvizje autonome duke u bazuar në hartën e krijuar nga SLAM dhe filtrit AMCL.

Por edhe pse SLAM ka filluar të këtë aplikim të lartë në fushën e robotikës ende ka shumë pengesa që SLAM të arrijë efikasitetin e kërkuar kjo për shkak që të disa mangësive të shpjeguara në kapitullin 6 gjithashtu gjatë skenimit të SLAM secila lëvizje ka një margjinë të gabimit kështu me kalimin e kohës këto gabime grumbullohen që dërgon në devijime të përgjithshme të procesit. Gjithashtu pamundësia e përditësimit të hartave të krijuara si dhe kur kemi realizim të SLAM me video-procesim atëherë shfaqet nevoja edhe për fuqi të lartë procesuese. Kështu rekomandohet që gjatë dizajnimit dhe planifikimit për të aplikuar SLAM të shikohet mundësia e bashkimit të sensorëve(LiDAR,Kamera etj.) dhe varësisht prej qëllimit të bëhet aplikimi pasi si për zvogëlimin e margjinës së gabimit si për rritjen e efikasitetit nevojitën më shumë sensorë e të cilët po kanë kosto të lartë ekonomike.

8 LITERATURA

[1] *Pajaziti A.*: SLAM – Map Building and Navigation via ROS, 2014.

[2] *Jason M. O'Kane*: A Gentle Introduction to ROS, 2018.

[3] *Riisgaard, S., Blas, M.*: SLAM for Dummies, 2005

[4] *Keatmanee Ch., Baber J., Bakhtyar M.*: Simple Example of Applying EKF, 2014

[5] *Wang F., Zhang J., Lin B.*: Two stage particle filter for nonlinear Bayesian Esitmation, 2018

[6] *Grisetti G., Stachniss C., Burgrad W*: Improved Techniques for Grid Mapping

with Rao-Blackwellized Particle Filters, 2018.

[7] *Teame W., Zhongmin W., Yanan Y*: Optimization of SLAM Gmapping based on Simulation, 2020

[8] *Kshitij, T., Chong, N.*: Multi-robot Exploration for Environmental Monitoring, 2019.

[9] *Koubaa, A*: Robot Operating System (ROS), 2017.

[10] *Quigley, M., Gerkey, B., Smart, W.*: A Practical Introduction to the Robot Operating System, 2015.

[11] *Pyo, Y., Cho, H., Jung, R., Lim T.,*: ROS Robot Programming, 2017

[12] *Cousins, S.,*: "Welcome to ROS Topics", Robotics & Automation Magazine,2010

[13] *Cousins, S.,*: Is ROS Good for Robotics? IEEE Robotics & Automation Magazine,2012

[14] *Mur-Artal, R., Montiel, J., Tardos, J.*: ORB-SLAM: A Versatile and Accurate Monocular SLAM System, 2015.

[15] *Murali, A.,*: PyRobot: An Open-source Robotics Framework for Research and Benchmarking, 2019

[16] *Jatavallabhula, K.,*: Automagically differentiable SLAM, 2019

[17] *Joseph, L.:* ROS Robotics Projects, 2017

[18] *Cook, G., Zhang, F.*: Mobile Robots: Navigation, Control and Sensing, Surface and AUVs, 2020.

[19] *Alan, A., Pritsker B.*: Introduction to Simulation and SLAM II,2015.

[20] *Mur-Artal,* R.,: ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras, 2016

[21] *Bailey, T., Durrant-Whyte H.*: Simultaneous Localization and Mapping (SLAM) Part II, 2006.

[22] *Theodoridis, T., Hu, H., McDonald-Maier, K., Gu, D.*: Kinect Enabled Monte Carlo Localisation for a Robotic Wheelchair, 2013.

[23] *Missura, M., Bennewitz, M.*: Predictive Collision Avoidance for the Dynamic Window Approach, 2019.

[24] https://www.ros.org/

[25] https://automaticaddison.com/extended-kalman-filter-ekf-with-python-code-example/

[26] https://blogs.nvidia.com/blog/2019/07/25/what-is-simultaneous-localization-and-mapping-nvidia-jetson-isaac-sdk

[27] https://www.mathworks.com/help/ros/ug/explore-basic-behavior-of-the-turtlebot.html

[28] https://www.turtlebot.com

[29] http://gazebosim.org/tutorials?tut=ros_overview

[30] https://www.mathworks.com/products/ros.html

[31] http://wiki.ros.org/slam_toolbox

[32] http://wiki.ros.org/gmapping

[33] http://wiki.ros.org/dwa_local_planner

[34] http://wiki.ros.org/amcl

[35] https://www.mathworks.com/help/nav/ug/implement-simultaneous-localization-and-mapping-with-lidar-scans.html

UNIVERSITETI I PRISHTINËS “HASAN PRISHTINA”
FAKULTETI I INXHINIERISË MEKANIKE
Rruga Agim Ramadani, Ndërtesa e Fakulteteve Teknike, 10 000 Prishtinë, Republika e Kosovës
Tel: +383 38 552 126 ext. 101 * E-mail: fim@uni-pr.edu * www.fim.uni-pr.edu

DEKLARATË E STUDENTIT PËR PUNË AUTENTIKE

Me anë të kësaj deklarate, unë Albin Krrabaj, me përgjegjësi deklaroj se ky punim nuk është prezantuar për vlerësim apo botuar më parë, pjesërisht apo në tërësi, pranë këtij apo ndonjë institucioni tjetër. Më tej deklaroj që:

a. punimi i paraqitur këtu është origjinal dhe është punuar në tërësi nga unë;
b. punimi nuk është marrë nga studentë të tjerë apo punime të tjera në Universitetin e Prishtinës “Hasan Prishtina” ose nga ndonjë universitet tjetër;
c. punimi nuk është kopje e ndonjë punimi të marrë në internet apo bibliotekë;
d. punimi nuk përmban modifikim të dhënash, duke i paraqitur ato si kontribut origjinal;
e. punimi respekton të gjitha kërkesat për të drejtat e autorit, duke saktësuar dhe cituar të gjitha kontributet nga burime te tjera.

Punimi i diplomës në fjalë vlen për nivelin Master të studimeve dhe mban titullin:

“LOKALIZIMI DHE KRIJIMI I HARTAVE TË LËVIZJEVE NË KOHË REALE (SLAM) TË ROBOTËVE MOBIL DUKE SHFRYTËZUAR INTELIGJENCËN ARTIFICIALE (AI)”

Dëshmoj se jam vënë në dijeni që vërtetimi ndryshe i atyre që u thanë më sipër do të rezultojë në tërheqjen e titullit të fituar bazuar në këtë punim.

Prishtinë, më__/__/__ Studenti/ja-nënshkrimi

_____