eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

Informatika Ingeniaritzako Gradua

Konputagailuen Ingeniaritza

Gradu Amaierako Proiektua

Marisorgin robotaren simulazioa eta nabigazioa Gazebo-ROS ingurunean

---

Egilea

Oihane Parra Garmendia

Zuzendaria

Elena Lazkano Ortega

informatika fakultatea
facultad de informática

2015eko ekainaren 23a

Laburpena

Proiektu honetan RSAIT ikerketa taldearen Marisorgin robotaren simulazioa eta nabigazioa landu dira Gazebo-ROS ingurunean. Robota UPV/EHUko Donostiako Informatika Fakultateko hirugarren solairuan zehar mugituko da, korridore, bulego eta gainontzeko geletatik igaroz. Bestalde, Marisorgin RWI etxeko B21 robota da, lau gurpil ditu eta mugimendu-sistema sinkronoa dauka. Nabigaziorako, robota bere kokapenetik (jatorriko puntua) guk adierazitako puntura (helburuko puntua) mugitu beharko da modu autonomoan.

Gaien aurkibidea

Irudien aurkibidea

Taulen aurkibidea

1. KAPITULUA

Aurkezpena

Aurkibidea

1.1 Sarrera

Robot mugikor autonomoak ataza konplikatuak burutzeko gai diren sistema artifizialak dira. Benetan autonomoak izateko, beren ingurunean gizakiaren interbentziorik gabe nabigatu ahal izateko gai izan beharko dira. Honek, nabigatzeak, hain zuzen ere, robotak bere buruaren posizioa ebatzi, eta ondoren, helburuko posizio batera joateko bidea planifikatu eta jarraitzea eskatzen du.

Proiektu honetan Marisorgin robotaren simulazioa eta nabigazioa garatuko dira. Robota, bere posiziotik guk zehaztutako helburura mugituko da modu autonomoan, mapa eta laserraz baliatuz. Hau guztia Gazebo simulagailua eta ROS (*Robot Operating System*) *framework*-a erabiliz lortuko da. Batetik, Gazebo simulagailuan Informatika Fakultateko hirugarren solairua eta robota irudikatuko dira. Bigarrenik, ROS tresnaren laguntzaz robotaren nabigazioa ahalbidetzeko egin beharreko guztia burutuko da.

1.2 Proiektuaren deskribapen orokorra

Esan bezala, proiektu honetan Marisorgin robotaren simulazioa eta nabigazioa landuko dira Gazebo eta ROS erabiliz. Hortaz, proiektua bi fase nagusitan banatuko dela esan dezakegu, nahiz eta batzuetan bi faseak gainjartzea beharrezkoa izango den.

Simulazioari dagokionez, lehenik eta behin uneoro Marisorginen ingurunea izango den Informatika Fakultateko hirugarren solairua irudikatu beharko dugu. Horretarako, SDF (*Simulator Description Format*) formatuko fitxategi batean solairuko elementu nagusiak zehaztu beharko ditugu; hala nola, paretak, zutabeak eta ateak. Honez gain, Marisorgin robot errealaren informazioa ere bildu beharko dugu. Hau lortzeko ROS-en erabiltzen den URDF (*Universal Robotic Description Format*) formatuaz baliatuko gara, robotaren atal bakoitza eta hauen arteko loturak deskribatuz. Robotari aginduak eta informazioa pasa, nahiz beregandik datuak jasotzeko, plugin ezberdinak ere konfiguratu beharko dira.

Bigarrenik, nabigazioa posible izateko, hau da, robota A puntu batetik beste B puntu batera lekualdatzeko, nabigazio-sistemak mapa estatiko bat erabiliko du, eta honen arabera erabakiko du zein den jarraituko duen helbururainoko bidea. Mundu errealean, aldiz, ohikoa da ingurunea dinamikoa izatea: bertako elementuak ordezkatu, berriak gehitu edota pertsonak agertu bezain pronto desagertzea eguneroko gertaera arruntak izan daitezke. Hori dela eta, Marisorginek bere begiak balira bezala erabiltzen duen laserra da helbururako bidea egokitzea ahalbidetuko diona.

Memoria honek proiektuaren nondik norakoak azaltzen ditu, hainbat ataletan antolatuta. 2. atalean Proiektuaren Helburuen Dokumentua aurki daiteke, proiektuaren kudeaketa azaltzen duelarik. Jarraian, ROS *framework*-aren (3. atala) eta Gazebo simulagailuaren (4. atala) oinarrizko ideiak ikus ditzakegu. 5. atalean, aldiz, geure inguruneari buruzko zehaztasunak zerrendatu dira. Ondorengo hiru kapituluetan proiektuaren garapena azaltzen da: 6. atalean, Marisorgin robotaren simulazioa; 7.ean, berriz, nabigazioa Gazebo simulagailuan; eta 8.ean, Marisorgin robot errealaren nabigazioa Informatika Fakultatean bertan. Azkenik, behin proiektua garatzen amaituta, ateratako ondorioak eta etorkizunean egin daitezkeen hobekuntzak jasotzen dira 9. atalean.

Dokumentu honen eranskinetan, sortutako nabigazio sistemaren erabilpen gida, sortutako ROS paketeen egitura eta bileren aktak bildu dira.

2. KAPITULUA

Proiektuaren Helburuen Dokumentua

Aurkibidea

Proiektuaren Helburuen Dokumentuan garatu den sistemaren azalpena egingo da. Proiektuaren deskribapena eta helburuez gain, lanaren deskonposaketa egitura eta landutako atazak zerrendatuko dira. Gainera, proiektuan zehar egindako lana islatuko duen kronograma eta dedikazio estimazioa aurki daitezke. Komunikazio, kalitate eta arriskuen plana nahiz eskuraketen kudeaketa aztertu ondoren, lan metodologia eta lana egiteko eskainitako denbora erreala azalduz amaituko dugu kapitulua.

2.1 Irismena

Proiektuaren helburua Marisorgin robotaren simulazioa eta nabigazioa Gazebo-ROS ingurunean garatzean datza.

Marisorgin, UPV/EHUko Donostiako Informatika Fakultatean diharduen RSAIT ikerketa taldearen robot aitzindaria da. Robot hau aipatutako fakultateko hirugarren (azken) solairuan zehar mugituko da bere kokapenetik erabiltzaileak zehazten dion helbururaino.

Behin Gazebo-n robotaren nabigazioa simulatuta, Marisorgin errealarekin fakultatean bertan gertatzen dena aztertuko da.

Hau burutzeko, batetik ROS *framework*-a erabiliko da, Indigo bertsioa. Bertan, robotaren nabigazioa konfiguratuko da.

Bestalde, Gazebo simulagailuaren 2.2 bertsioaren bidez simulazioa garatuko da. Bertan, fakultateko hirugarren solairuko pareta eta zutabe guztiak irudikatuko dira 1:1 eskalan. Gainera, paretetan ateak egon beharko liratekeen lekuan “zuloak” agertuko dira robota arazorik gabe bulego eta gainontzeko guneetan sartu eta irteteko. Ate hauek guztiek 90 cm-ko zabalera izango dute. Jarriko ez diren ate bakarrak eskaileretara doazen hirurak (bi beherantz eta bat goiko terrazara) eta igogailukoa izango dira. Honez gain, ez da beharrezkoa izango korridore eta geletako oztopoak Gazebo-n jartzea, ondoren laserraren laguntzarekin ekidingo baitira.

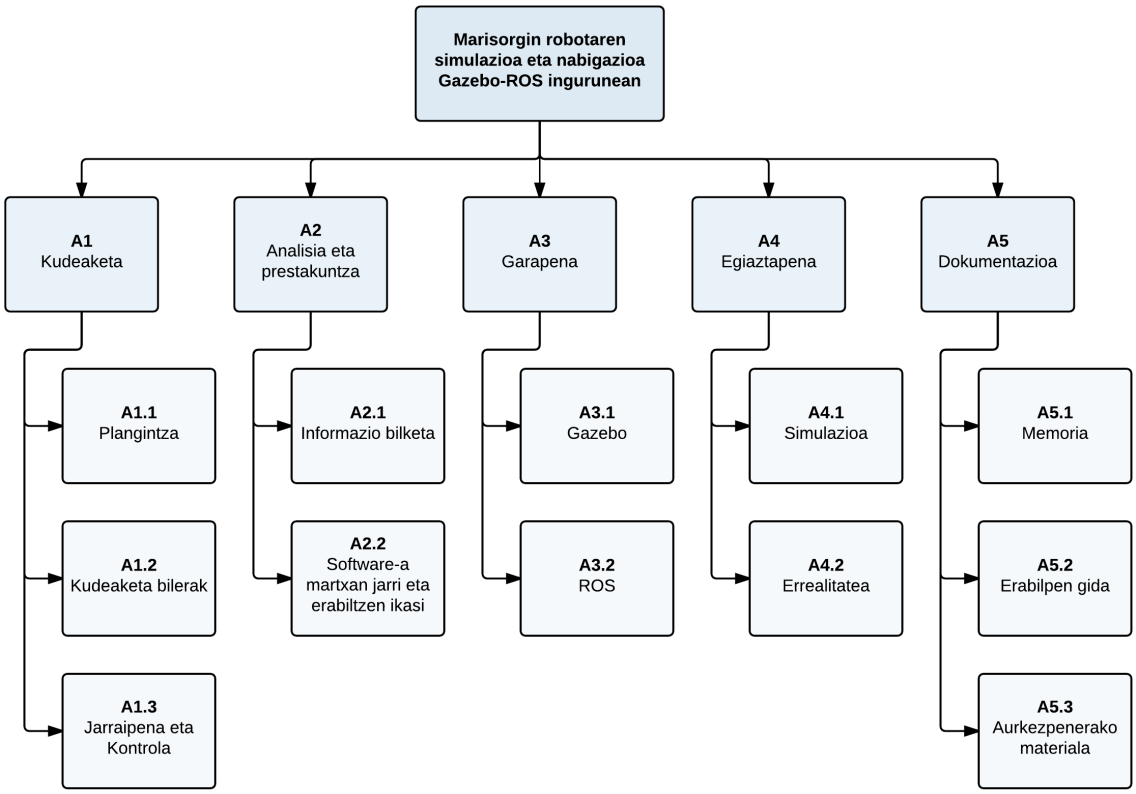
Robotaren URDF eredua RSAIT taldeak sortutakoa izango da, baina Gazebo-n erabiltzeko beharrezko aldaketak egingo zaizkio.

Inplementazioko zehaztasunak bat etorriko dira robot errealaren ezaugarriekin, hala nola, laser mota eta mugimendu abiadurak.

Aipatutako bi tresna hauek Ubuntu 14.04 sistema eragilea duen ordenagailu batean instalatu eta erabiliko dira.

2.2 Lanaren deskonposaketa egitura

2.1 irudiko diagraman proiektuan zehar landutako atal ezberdinak ikus daitezke lan-paketetan antolatuta.



2.1 Irudia: Lanaren deskonposaketa egitura (LDE diagrama)

2.3 Atazak

Jarraian, proiektua garatzeko beharrezkoak izan diren atazak zerrendatu dira, aurreko ataleko lanaren deskonposaketa egiturako lan-paketeak jarraituz.

A1. Kudeaketa

A1.1 Plangintza

A1.1.1 Irismena zehaztu

A1.1.2 LDE

A3.1.2 Marisorgin robotaren eredua (URDF) txertatu eta behar bezala moldatu

A3.1.3 Beharrezko plugin-ak konfiguratu

A3.2 ROS

A3.2.1 Robota mugitu komando lerro bidez (Gazebo-n simulatuz)

A3.2.2 Robota mugitu teklatu bidez (Gazebo-n simulatuz)

A3.2.3 Nabigaziorako aurrekariak prestatu

A3.2.4 Nabigazioa konfiguratu eta martxan jarri

A3.2.5 Helburuak bidaltzeko programa inplementatu

A4. Egiaztapena

A4.1 Simulazioa

A4.1.1 Oztoporik gabe, RViz tresnaren bidez helburuak zehaztuz

A4.1.2 Oztoporik gabe, sortutako programaren bidez helburuak zehaztuz

A4.1.3 Mapan agertzen ez diren oztopo estatikoekin

A4.1.4 Oztopo dinamikoekin

A4.1.5 Iritsi ezin daitekeen helburu bat

A4.1.6 Ikusten ez dituen oztopoak

A4.2 Errealitatea

A4.2.1 Oztoporik gabe

A4.2.2 Mapan agertzen ez diren oztopo estatikoekin

A4.2.3 Oztopo dinamikoekin

A4.2.4 Ikusten ez dituen oztopoekin

A4.2.5 Simulagailuan sortutako maparekin

A5. Dokumentazioa

A5.1 Memoria idatzi

A5.2 Erabilpen gida prestatu

A5.3 Defentsarako materiala prestatu

2.4 Kronograma

2.2 irudian proiekturako planifikatutako lana islatzen duen kronograma ikus daiteke. Bertan, lan-pakete eta atazak noiz egin diren adierazi da, asteka.

Aipatutako kronograman ikus daitekeen moduan, guztira 17 aste iraun ditu proiektuak. Lehenengo bi asteetan batez ere plangintza landu da, baina konstituzio bilera eta informazio bilketa ere egin dira.

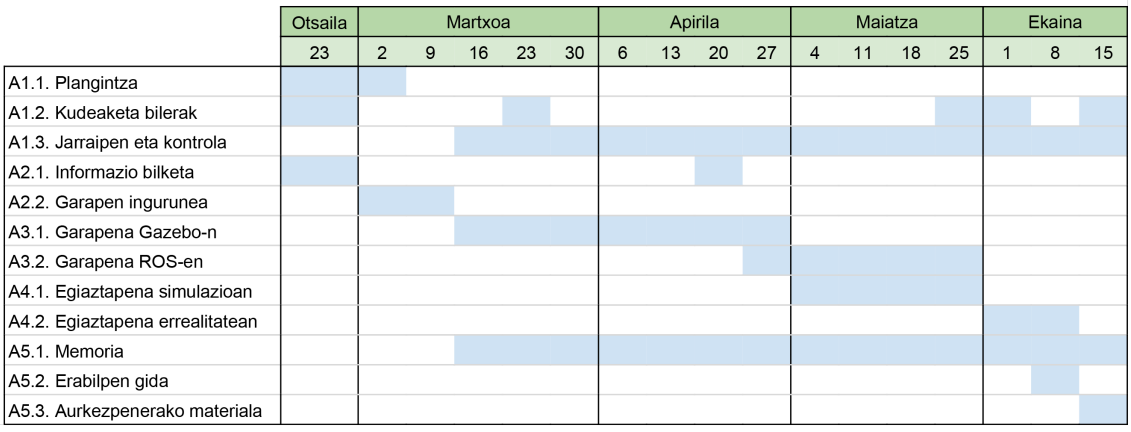
Martxoko lehenengo zatian garapen ingurunea ezagutzeari dedikatu zaio denbora. Martxoko bigarren zatian eta apirilean, berriz, Gazebo-ko garapena, memoria eta jarraipen eta kontrola landu dira, gehien iraun duten atazak, hain zuzen ere.

Maiatzean, aurreko atazekin jarraitzeaz gain, ROS-eko inplementazioak burutu dira, eta aldi berean simulagailuan egiaztapenak egin dira.

Ekaineko lehen bi asteei erreparatzen badiegu, Marisorgin errealaren nabigazioa egiaztatu da Informatika Fakultatean.

Ataza laburragoen artean, kudeaketa bilerak edo informazio bilketa aurki ditzakegu. Hauek denboran zehar sakabanatuagoak egon dira.

Zehaztasun gehiagorako ikusi 2.2 irudiko kronograma.



2.2 Irudia: Kronograma

2.5 Dedikazio estimazioa

2.1 taulan proiektua amaitzeko egindako lanaren dedikazio estimazioa ikus daiteke, lan-paketeka antolatuta.

Lan-paketea	Dedikazio estimazioa (ordutan)
A1. Kudeaketa	**16**
A1.1. Plangintza	6
A1.2. Kudeaketa bilerak	6
A1.3. Jarraipena eta Kontrola	4
A2. Analisia eta prestakuntza	**48**
A2.1. Informazio bilketa	3
A2.2. SWa martxan jarri eta erabiltzen ikasi	45
A3. Garapena	**160**
A3.1. Gazebo	65
A3.2. ROS	95
A4. Egiaztapena	**12**
A4.1. Simulazioa	9
A4.2. Errealitatea	3
A5. Dokumentazioa	**92**
A5.1. Memoria	80
A5.2. Erabilpen gida	2
A5.3. Defentsarako materiala	10
GUZTIRA	**328**

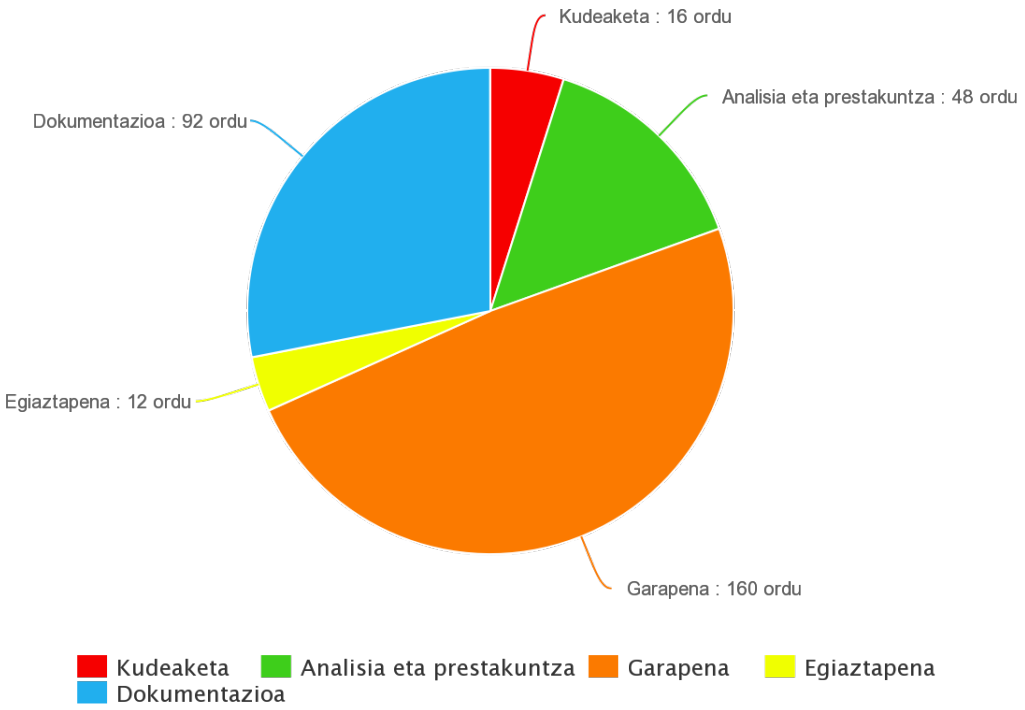
2.1 Taula: Dedikazioaren estimazioa

2.1 taula laburbilduz, sektore-diagrama bat ere sortu da (ikusi 2.3 irudia) lan-pakete bakoitzari dedikatzea aurreikusitako pisua argi ikusteko.

Bai 2.1 taulak, bai 2.3 irudiak erakusten diguten moduan, A3 lan-paketea, hots, garapenari dagokiona, izango da dedikazio gehien eskainiko zaiona. Hurrengoak, dokumentazioa eta analisia eta prestakuntza izango dira. Azkenik, kudeaketa eta egiaztapena lan-paketeak lan gutxien eskatuko dutenak izatea espero da.

2.6 Komunikazio plana

Proiektuaren garapenean zehar arazoren bat edota duda garrantzitsu bat balego, EHU-ko posta elektronikoa erabiliko da zuzendaria eta zuzendarikidearekin komunikatzeko.



2.3 Irudia: Dedikazio estimazioaren sektore-diagrama

Honez gain, aipatzekoa da zuzendariarekin eta zuzendarikidearekin egindako bilerak proiektuaren aurrerapenaren arabera egingo direla, guztiak Informatika Fakultateko robotikako laborategian. Hau da, konstituzio bilera proiektua hasi aurretik egiteaz gain, Gazebo-n fakultateko ingurunea sartutakoan izango da lehenengo bilera. Bigarrena, aldiz, nabigazioa simulatzea lortutakoan, eta hirugarrena, nabigazio akatsak zuzendu eta robot errealarekin egiaztatu behar denean. Azkenik, itxiera bilera egingo da proiektuari amaiera emateko.

2.7 Kalitate plana

Ondorengo lerroetan proiektuaren kalitate plana aurkeztuko da, proiektuaren kalitate betekizun eta dimentsioak definituz.

Produktuaren oinarrizko betekizunen artean honako hauek dauzkagu:

- Gazebo-n UPV/EHU-ko Informatika Fakultateko hirugarren solairua irudikatu 1:1 eskalan.

- Marisorgin robotaren simulazioa egin Gazebo 2.2 simulagailua erabiliz.
- Marisorgin-en nabigazioa prestatu ROS Indigo bertsioan.
- Marisorgin-en nabigazioa Gazebo-n simulatu.
- Marisorgin robot errealarekin prestatutako nabigazioaren funtzionamendua egiaztatu Informatika Fakultatean bertan.

Produktuaren kalitate dimentsioak, berriz, ondorengoak dira:

- Simulatutako fakultatearen itxurak (koloreak, etab.) errealitatearekin ahalik eta antza gehien izango du.
- Planotik lortutako neurriak erabili ordez, benetakoak neurtuz irudikatuko da fakultatea, ahal den neurrian.
- Hirugarren solairuan dauden oztopoen antzekoak Gazebo-n jarriko dira.
- Simulatutako Marisorgin robota benetakoarekin erraz identifikatuko da, kolore egokiak erabiliz.
- Marisorgini helburua bulego bidez zehazteko programa bat prestatuko da. Erabiltzaileak bulego zenbakia sartuz helburu horretara bidaliko du robota.

2.8 Arriskuen plana

Luzera handiko proiektuetan beharrezkoa da hau amaitu bitartean gerta litezkeen arazoak aurreikusi eta hauei aurre egiteko konponbideak planifikatzea.

Arrisku bakoitzak proiektuarengan izan dezakeen eragina ez da berdina izango. Batez ere eragin handiena dutenei ahalik eta konponbide onena prestatzen saiatuko gara, eta baita gertatzeko probabilitate handia dutenei ere.

Jarraian, proiektuan zehar gertatu daitezkeen arriskuen identifikazioa egin da. Hala ere, zerrenda honetan identifikatu gabeko arriskuak ere gerta daitezkeela kontutan izan behar dugu.

Esan bezala, hona hemen identifikatutako arriskuak eta konponbide posibleak.

1. **Garapen ingurunearen matxura.** Proiekturako darabilgun ordenagailua beste zeregin batzuetarako ere erabiltzen da eta posiblea litzateke hau hondatu eta egindako lana atzitu ahal ez izatea.

   - Arazo hau ekiditeko, lanean pauso garrantzitsu bat ematen den bakoitzean (fitxategi berri bat sortu edota aldaketa garrantzitsu bat egin) segurtasun kopia bat egingo da *Google Drive* plataformara proiektuko fitxategi guztiak igoz. Honela, uneoro eta edonondik eskuragarri izango dugu egindako lan guztia. Gainera, modu honetan, egindako aldaketa bat desegin nahi badugu, aurreko bertsio bat deskarga dezakegu bertatik. Bestalde, eramangarria konpondu bitartean, mahaigaineko ordenagailu bat egongo da eskuragarri, sistema eragile berdinarekin.

2. **Proiektuaren garatzailearen behin-behineko baja.** Proiektua garatuko duen pertsona bakanari ustekaberen bat suertatzen bazaio, atzerapenak sortuko dira. Ondorioz, birplangintza bat egin beharko da.

   - Honi aurre egiteko, atzeratutako atazei planifikatutakoak baino ordu gehiago dedikatu beharko zaizkie astean zehar, galdutako denbora berreskuratu arte.

3. **Denbora falta.** Aurreko puntuarekin erlazionatuta, baliteke garatzaileak ikasturtearen bigarren lauhilekoko beste irakasgaiei dedikatu beharreko lana dela eta arazoak izatea asteko dedikazio orduak betetzeko.

   - Plangintza egiterako orduan kontutan hartuko da zein garaitan izango duen lan gehiago eta lan banaketa ez da orekatua izango; batez ere maiatza erdialdetik aurrera dedikatuko da denbora gehiago, behin irakasgaiak bukatuta.

4. **Elkartzeko bateraezintasunak.** Zuzendariarekin bilera egiteko zailtasunak egon daitezke proiektu guztian zehar.

   - Biltzeko arazoak baleude, elkartu bitartean posta elektroniko bidez saiatuko da dudak ebazten eta beharrezkoa bada, beste ataza batzuekin lanean jarraituko da.

5. **Garapen tresnen erabileraren zailtasunak.** ROS eta Gazebo inguruneak erabiltzen ikasteko informazio gutxi egon liteke, edo uste baino denbora gehiago behar prozesu honek.

   - Honi aurre egiteko, denbora gehiago dedikatu beharko zaio dagokion atazari, eta ondorioz, plangintza birmoldatu. Honez gain, galdutako denbora errekuperatu beharko da astean ordu gehiago sartuz.

2.9 Eskuraketen kudeaketa

Bi izango dira proiektu honetako eskuraketa garrantzitsuenak.

Lehenik eta behin, proiektuaren garapenarekin hasi aurretik beharrezkoa izango da Informatika Fakultateko hirugarren solairuko planoa eskuratzea; honen bidez, Gazebo-n gure ingurunea irudikatu dezakegu. Eskuraketa hau fakultateko Txelo Ruiz irakaslearen eskutik lortuko da, bera arduratuko baita idazkaritzako administratzaileari eskatzeaz.

Bestalde, Marisorgin robotaren URDF eredua izango da bigarren eskuraketa. Hau proiektuaren zuzendari den Elena Lazkanoren bidez eskuratuko da.

Gainontzeko eskuraketa guztiak (paketeak edo plugin-ak, batez ere) proiektuaren garapenean zehar egingo dira eta Internet erabiliz lortuko dira.

2.10 Lan metodologia

Proiektu hau aurrera eramateko jarraituko den metodologia sinplea da. Garatzaileak bere kasa prestatuko du nabigazioa Gazebo-ROS ingurunean, eta argitu ezin duen dudaren bat badu, proiektuaren zuzendariari galdetu dio, behar izanez gero, bilera bat eginez.

Bilera hauek, aurretik aipatu dugun moduan, proiektuaren aurrerapenaren arabera egingo dira.

Proiektuaren garapena fase ezberdinetan banatu da. Lehenengoan, Proiektuaren Helburuen Dokumentua prestatuko da eta egin beharrekoa zer den eta nola egin behar den ondo finkatu. Bigarren fasean, ROS eta Gazebo inguruneen erabileraren ezagutzak barneratuko dira. Jarraian, Gazebo eta ROS-eko inplementazioak burutuko dira. Eta amaitzeko, funtzionamendu probak egingo dira, bai simulazioan, bai errealitatean.

Nahiz eta memorian atal ezberdinetan azalduko den hauetariko bakoitza, fase asko aldi berean egingo dira; batez ere, nabigazioaren konfigurazioa eta probak.

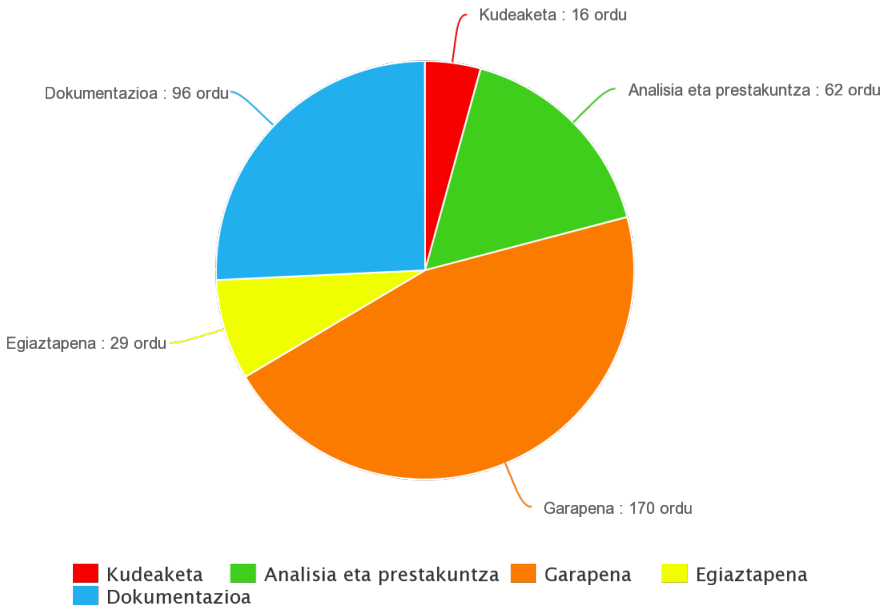
2.11 Denbora erreala

2.5 atalean proiektuari dedikatzea espero zen denbora ikusi dugu. Orain, behin proiektua amaituta egindako lan guztia islatuko dugu 2.2 taulan.

Lan-paketea	Dedikazio estimazioa (ordutan)	Benetako dedikazioa (ordutan)
A1. Kudeaketa	**16**	**16**
A1.1. Plangintza	6	7
A1.2. Kudeaketa bilerak	6	5
A1.3. Jarraipena eta Kontrola	4	4
A2. Analisia eta prestakuntza	**48**	**62**
A2.1. Informazio bilketa	3	2
A2.2. SWa martxan jarri eta erabiltzen ikasi	45	60
A3. Garapena	**160**	**170**
A3.1. Gazebo	65	65
A3.2. ROS	95	105
A4. Egiaztapena	**12**	**29**
A4.1. Simulazioa	9	7
A4.2. Errealitatea	3	22
A5. Dokumentazioa	**92**	**96**
A5.1. Memoria	80	83
A5.2. Erabilpen gida	2	3
A5.3. Defentsarako materiala	10	10
GUZTIRA	**328**	**373**

2.2 Taula: Benetako dedikazioa

2.2 taulako benetako dedikazioa laburbilduz, sektore-diagrama bat ere sortu da (ikusi 2.4 irudia) lan-pakete bakoitzari benetan dedikatu zaion pisua argi ikusteko.



2.4 Irudia: Benetako dedikazioaren sektore-diagrama

Lan-pakete bakoitzari dedikatu zaion denboraren ehunekoa ez da asko aldatu estimaziotik benetako dedikaziora.

Ordu kopuru totala kontutan hartzen badugu, berriz, 45 ordu gehiago behar izan direla ikusi dugu 2.2 taulan. Desbideraketa honen eragile nagusiak A2.2, A3.2 eta A4.2 atazak izan dira; batetik, ROS *framework*-aren erabilpena ulertzea uste baino konplexuagoa izan delako, eta bestetik, Marisorgin errealean nabigazioaren funtzionamendua egiaztatzean arazoak izan direlako. Gainera, hasieran espero ez ziren alderdi batzuk ere eragina izan dute: planoa neurririk gabe zegoen, eta Marisorginek proiektu honetatik at sortutako aldaketa batzuk jasan zituen. Hala ere, guztiak ondo funtzionatzea lortu denez, baita errealitateko robotean ere, desbideraketa onargarria kontsidera dezakegu.

3. KAPITULUA

ROS

Aurkibidea

3.1 Sarrera

Azken urteotan, robotikako komunitateak garapen handia izan du. Robot hardware fidagarri eta merkea inoiz baino eskuragarriagoa da, lurzoruan dabiltzan robotetatik hasi eta *quadrotor* helikoptero edo humanoideetaraino. Gainera, robotak autonomia gehiagorekin funtzionatzen laguntzeko algoritmoak ere garatu dira. Nahiz eta hobekuntza asko egon diren, oraindik robotekin lotutako erronka asko dauzkate robot garatzaileek. Zailtasun hauetariko batzuk arintzeko helburua duen software plataforma bat, *Robot Operating System* edo ROS izenekoa da.

ROS robotentzako softwarea garatzeko *framework* malgu bat da. Sistema eragile batengandik espero ditzakegun zerbitzuak eskaintzen ditu; besteak beste, hardware abstrakzioa, behe-mailako gailuen kontrola, prozesuen arteko mezu-trukea eta pakete kudeaketa. Honez gain, konputagailu anitzen arteko kodea lortu, eraiki, idatzi eta exekutatzeko tresnak eta liburutegiak eskuragarri jartzen ditu.

Grafo arkitekturan oinarrituta dago, eta grafoko nodo bakoitza zeregin jakin bat egiteko prestatuta egongo da. Normalean, ROS erabiliz sortutako sistema bat hainbat prozesuz osatuta dago, eta prozesu bakoitza makina desberdin batean koka daiteke, puntutik punturako (*point-to-point*) topologia bidez konektatuz.

Honez gain, lengoaia anitzekoa da; hau da, ROS lengoaiekiko independentea da eta C++, Python, Octave, Java eta beste hainbat lengoaiekin lan egin dezake.

2006an garatzen hasi ziren *framework* hau UNIX sistemarentzat orientatuta dago, nahiz eta beste sistema eragile batzuetarako egokitzen ari diren.

Marisorgin ardatz duen proiektu hau garatzea ahalbidetu duen tresna garrantzitsuena izan da ROS eta bere Indigo bertsioa. Tresna hau inoiz erabili ez bada, baliteke ulergaitza gertatzea funtzionamendua edota elementu nagusien erabilera. Honi aurre egiteko, ordea, tutorial eskaintza zabala dago ROS *framework*-aren web gune ofizialean [1].

Ondorengo atalean, proiektuaren garapena ulertzeko gutxienez barneratu beharreko ROS-en ezaugarriak azalduko dira.

3.2 Kontzeptu orokorrak

ROS-en kontzeptu nagusiak bi mailatan bana ditzakegu. Lehenengo mailako kontzeptuak fitxategi-sistemarekin lotuta daude, eta bertan paketeak edo mezu motak aurki di-

tzakegu, adibidez. Bigarrenik, beste maila konputazio grafoaren maila da; eta kontzeptu hauek, elkarren artean komunikatzen diren ROS prozesuek osatutako *peer-to-peer* sarearekin zerikusia dute.

3.2.1 Fitxategi-sistema

Fitxategi-sistema mailako kontzeptuak ROS baliabideekin daude erlazionatuta. Hona hemen proiektu honetan garrantzitsuenak izan direnak.

Paketeak

Paketeak ROS-en *software*-a antolatzeko unitate nagusiak dira. Pakete batek elementu desberdinak izan ditzake: exekuzio garaiko prozesuak (nodoak), liburutegiak, datu multzoak, konfigurazio fitxategiak, edo ondo antolatutako beste edozer. Pakete hauen bidez kodea berrerabiltzea posible izango da, eta elkarren artean pilatan (*stack*) antolatzen dira.

Pakete manifestuak

Manifestuek paketeei buruzko metadatuak gordetzen dituzte, pakete barruan dagoen *package.xml* izeneko fitxategi batean. Bertan, besteak beste, paketearen izena, bertsioa, deskripzioa eta lizentziari buruzko informazioa aurki ditzakegu.

Launch fitxategiak

Launch fitxategien bidez (.*launch* luzapena dutenak) nodo bat baino gehiago aldi berean exekuta ditzakegu, edo baita erabili nahi ditugun parametroak zehaztu ere. Beraz, lana asko errazten dute.

3.2.2 Konputazio-grafoa

Konputazio-grafoa elkarrekin datuak prozesatzen dituzten prozesuek sortutako *P2P* edo *peer-to-peer* sarea da. Jarraian azalduko diren konputazio-grafoko kontzeptuek modu desberdinetan datuak pasatzen dizkiote grafoari.

Nodoak

Nodoak konputazioa burutzen duten prozesuak dira. Normalean, robota kontrolatzen duen sistema hainbat nodoz osatuta dago.

Adibide modura, Marisorginen gisako nabigazio-sistema batean, nodo batek laserra kontrola dezake, beste batek mugimendu-sistemaren kontrola duen bitartean. Hurrengo nodo batek kokapenaren jarraipena egiten du, eta laugarren bat helbururainoko bidearen plana burutzeaz arduratuko da. Bosgarren nodo bat ere izango dugu sisteman gertatzen dena grafikoki ikustea ahalbidetzeko. Eta horrela beste nodo askorekin.

ROS gainbegiralea (*master*)

ROS gainbegiraleari esker nodoek elkar aurkitu dezakete, mezuak trukatu, eta zerbitzuei deitu. Izen-zerbitzari eta erroldatze-zerbitzuak ematen dizkie martxan dauden nodoei.

Parametro zerbitzaria

Parametro zerbitzariak datuak kokapen zentral batean gordetzea ahalbidetzen du; hau da, hiztegi partekatu moduko bat da, non nodoek exekuzio-denboran parametroak gordetzeko edo eskuratzeko erabiltzen duten.

Mezuak

Nodoek beraien artean komunikatzeko mezuak erabiltzen dituzte. Mezu bat mota ezberdinetako eremuak biltzen dituen data egitura bat da. Oinarrizko motak (osokoak, errealak, boolearrak, etab.) onartzen dira, eta baita hauez osatutako zerrendak ere. Egiturak eta zerrendak nahi bezala konbinatu daitezke desiratutako mezua bidaltzeko.

Topic-ak

Topic-a mezu baten edukia identifikatzeko erabiltzen den izen bat da, postontzia, alegia.

Mezuak argitaratu/harpidetu (*publish/subscribe*) ereduaren bidez bidaltzen dira. Hau da, nodo batek mezu bat bidali nahi badu, *topic* batean argitaratuko du. Bestalde, informazio jakin batean interesatuta dagoen nodo bat dagokion *topic*-era harpidetuko da.

Topic batean hainbat argitaratzaile egon daitezke aldi berean, eta baita harpidedun asko ere. Hori bai, argitaratutako mezu mota bat etorri beharko da *topic*-aren motarekin; ondorioz, harpidedunak ziurtatuta du mota jakin bateko mezuak bakarrik jasoko dituela *topic* horretan. Gainera, nodo batek *topic* batean baino gehiagotan argitaratu dezake, edota harpidetu.

Orokorrean, argitaratzaileek ez dute harpidedunen berri, ez eta alderantziz ere. Ideia informazioaren ekoizpena kontsumotik banatzea da.

Topic bidezko komunikazioa asinkronoa da.

Zerbitzuak

Argitaratu/harpidetu sisteman ez bezala, sistema banatuetan askotan beharrezkoak diren eskaera/erantzun elkarrekintzak zerbitzuen bidez egiten dira, ez bailitzateke egokia norabide bakarreko eta hainbat igorle eta hartzaile izan ditzakeen sistema erabiltzea. Eskaera/erantzunetan mezu bat eskaera burutzeko eta beste mezu bat erantzuna jasotzeko erabiltzen dira. Sistema honek zerbitzari/bezero ereduaren moduan funtzionatzen du: nodo batek zerbitzu bat eskainiko du eta bezero nodoak honi eskaera bat egin ondoren, erantzunari itxarongo dio.

Topic-etan ez bezala, zerbitzu bidezko komunikazioa sinkronoa da.

Bag fitxategiak

Bag fitxategiek ROS mezuen informazioa gorde eta berrerabiltzea ahalbidetzen dute. Adibidez, sentsoreetatik jasotako datuak bildu daitezke, eta ondoren, sistema fisiko errealaren beharrik gabe, irakurketa horiek prozesatu modu desberdinetan.

3.3 Beste kontzeptu batzuk

ROS plataformaren nukleoa arkitekturarekiko independentea izaten saiatzen da. Komunikatzeko modu ezberdinak eskaintzen ditu (*topic*-ak, zerbitzuak, parametroen zerbi-

tzaria...), baina ez du zehazten nola erabili edo nola izendatu behar diren. Honen eraginez, ROS arkitektura ezberdinekin erabili daiteke, baina ROS-en gainean sistema handiagoak eraikitzeko maila altuagoko kontzeptuak beharrezkoak dira. Jarraian, horietariko kontzeptu batzuk azalduko ditugu, proiektu honen garapena ulertzeko beharrezkoak direnak, hain zuzen ere.

Erreferentzia-sistemak/Transformazioak

Robot batekin zeregin desberdinak egiten gabiltzanean ezinbestekoa da robotak bera non dagoen jakitea, eta baita non dauden munduko gainontzeko elementuak berekiko ere. Adibide sinple bat ondorengoa da: robot mugikor batek baloi gorri bat bilatu eta bere eskuko heldulekuarekin ukitu behar du. Ekintza sinple hau burutzeko baloia eta heldulekuaren arteko erlazioa jakin beharra dago.

tf liburutegiak koordenatu-sistemak kontrolatu eta sistema osoan zehar datuak transformatzen ditu, transformatu-zuhaitza edo *transform tree* delakoa sortuz. Bertan erreferentzia-sistema batetik bestera pasatzeko aplikatu beharreko translazio eta errotazio informazioa gordetzen da. ROS pakete askok beren funtzionamendurako transformatu-zuhaitz hau argitaratuta egotea behartzen dute; adibidez, nabigazio atalean ikusiko dugun bezala, *navigation* izeneko pilak laserraren eta robotaren oinarriaren arteko transformazioa behar du.

Erreferentzia-sistema garrantzitsu asko dituen robot batekin lanean gaudenean, lan dezente da guztiak *tf*-n argitaratzeko konfiguratzea. Lan hau arintzeko, *robot_state_publisher* paketea dago. Bere laguntzaz, robotaren egoera *tf* liburutegira hedatuko da.

Robot eredua (URDF)

Unified Robot Description Format (URDF) ereduak roboten deskribapen fisikoa gordetzen duten fitxategiak dira. Fitxategi hauek XML formatua jarraitzen dute, beraz, etiketa ezberdinak erabiltzen dira. ROS-ek URDF eredu honen bidez robotak bizitza errealean daukan itxura eta zenbait ezaugarri fisiko dakizki.

Robot ereduaren elementu nagusiak *link*-ak eta *joint*-ak dira. *Link*-ek, robotaren zati zurrunen deskribapena egiten dute, eta *joint*-ak, berriz, *link*-en arteko loturen adierazle dira.

Xacro, XML makro lengoaia bat da, eta URDF formatukoak baino fitxategi laburragoak eta irakurtzeko errazagoak sortzea ahalbidetzen du makroak erabiliz. Modu honetan errazagoa da roboten deskribapen fitxategiak mantentzea, eta fitxategi barneko bikoizketak ekiditea. Gainera, *xacro* lengoaia erabiliz fitxategi ezberdinen artean bana daiteke deskribapena.

3.4 RViz

RViz roboten bistaratzerako erabiltzen den tresna ahaltsua da. Berez, ROS pakete bat da eta eskaintzen duen interfaze grafikoaren bidez sentsoreen datuak, robot ereduak, ingurunearen mapak, etab. ikusi ditzakegu hiru dimentsiotan. Hori dela eta, oso erabilgarria da gure robotaren kontrolagailuak sortu eta arazteko.

Proiektu honetan, RViz nabigazioko helburuak bidaltzeko erabiliko dugu, eta izandako arazoen zergatiak bilatzen ere lagunduko digu.

4. KAPITULUA

Gazebo

Aurkibidea

4.1 Sarrera

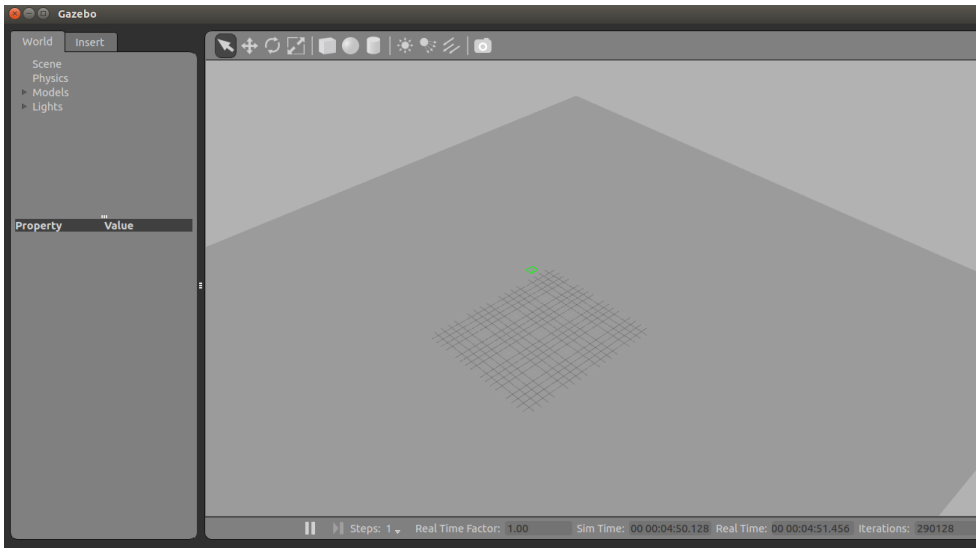
Roboten simulazioa oinarrizko tresna da robotekin lan egiten duen edonorentzat. Ondo diseinatutako simulagailu batek posible egiten du denbora gutxian algoritmoak probatu, robotak diseinatu, eta erregresio probak burutzea jokaleku errealistak erabiliz.

Gazebo hiru dimentsiotako simulagailu bat da, zinematikoa, dinamikoa eta robotanitzekoa. Robot artikulatuen simulazio zehatza eta eraginkorra egiteko aukera eskaintzen du, hiru dimentsiotako ingurune errealista konplexuetan, barnealdean nahiz kanpoaldean. Software librea da eta ROS-ekin bateragarria. Beraz, ROS-etik exekuta daiteke eta ROS erabili Gazebo-ko simulazioko robotak kontrolatzeko; hau da, robot horiekin komunikatu, datuak bidali eta jasoz.

Simulagailu honetan robotek ingurunearekin elkarreragin dezakete (objektuak hartu, bultza egin, lurrean biratu...), eta baita alderantziz ere, grabitatearen ondorioak pairatu edo objektuen kontra talka egin baitezakete.

Robot baten funtzionamendua simulatzeko, normalean honen ingurunea beharko dugu. Hori dela eta, Gazebo-n nahi dugun jokalekua (mundua) sor dezakegu, grabitate balioa edo beste parametro batzuk zehaztu ditzakegularik.

Proiektu honetan, Gazebo-ren 2.2 bertsioa erabili da (ikus 4.1 irudia) Marisorginen simulazioa burutzeko, ROS Indigo-k erabiltzen duen bertsio lehenetsia hori baita, eta hala gomendatuta dago.



4.1 Irudia: Mundu hutsa Gazebo simulagailuan

4.2 Eredu fitxategiak (*model*)

Gazebo-ko ereduek, ingelesez *model* deituak, elementu fisiko bat definitzen dute, propietate dinamiko, kinematiko eta ikuspenekoak erabiliz. Eredu batek edozer adierazi edo irudikatu dezake, bai forma sinple bat, bai robot konplexu bat; lurzorua ere eredu bat da. Honez gain, edozein ereduk plugin bat edo gehiago izan ditzake, eta hauek bere portaeran eragina izango dute.

Simulagailu honek datu base bat dauka ereduak gorde eta erabilgarri mantentzeko. Bertara guk sortutako eta erabilitako ereduak igo ditzakegu, eta baita deskargatu ere. Hortaz, Gazebo-n txertatutako ereduak guk sortuak edo ereduen datu basetik deskargatuak izan daitezke, eta erabiltzaile interfaze grafikoaren bidez nahiz kodea erabiliz gehitu ditzakegu bertara.

Eredu bat sortzeko SDF (*Simulator Description Format*) formatua erabiltzen da. SDF, roboten simulagailuentzat, bistaratzearentzat eta kontrolarentzat objektuak eta inguruneak deskribatzen dituen XML formatua da. Robotak, elementu estatiko eta dinamikoak, argiztapena, lurzorua eta baita fisika ere deskribatzeko gai da.

Deskribatzen ari garena eredu bat dela adierazteko <*model*> etiketa erabiltzen da, eta ereduari buruzko guztia etiketaren barruan joango da. Eredu bakoitzeko, eta ondorioz, fitxategi bakoitzeko, <*model*> etiketa bakarra erabili daiteke. Eredu bat deskribatzerakoan *link* eta *joint* elementuak zerrendatuko ditugu; hauek 3.3. zatiko URDF atalean azaldu dugun esanahi bera dute: *Link*-ek, robotaren zati zurrunen deskribapena egiten dute, eta *joint*-ak, berriz, *link*-en arteko loturen adierazle dira.

Gainera, eredua estatikoa izango den zehaztu beharko dugu (<*static*> etiketa erabiliz), edo balio lehenetsia erabili, eta baita bere gurasoarekiko izango duen kokapena ere (<*pose*>).

Eredua deskribatzen duen SDF fitxategiaz gain, direktorio berdinean konfigurazio fitxategi bat ere sortu behar da, non ereduari buruzko informazio orokorra gordetzen den; hala nola, ereduaren izena, bertsioa, deskribapena edo egilearen izena.

Jarraian, eredu sinple baten ($1m^3$-ko kaxa) konfigurazioa ikus daiteke:

```
1 <?xml version='1.0'?>
2 <sdf version="1.4">
3   <model name="kaxa">
4     <pose>0 0 0.5 0 0 0</pose>
5     <static>true</static>
```

```
    <link name="link">
      <collision name="collision">
        <geometry>
          <box>
            <size>1 1 1</size>
          </box>
        </geometry>
      </collision>
      <visual name="visual">
        <geometry>
          <box>
            <size>1 1 1</size>
          </box>
        </geometry>
      </visual>
    </link>
  </model>
</sdf>
```

URDF vs SDF

URDF eredua ROS *framework*-ean erabilgarria eta formatu estandarra den arren, bereizgarri asko falta zaizkio eta ez da egokitu robotikaren gaur egungo beharretara. URDF formatuak isolatutako robot bakan baten ezaugarri kinematiko eta dinamikoak bakarrik zehaztu ditzake; besteak beste, ezin du robotaren mundu barruko kokapena deskribatu, ezta honen marruskadura propietateak ere. Gainera, robotak ez diren elementuak ezin ditu zehaztu; adibidez, argiak, sakonera mapak edo 3D mapak, etab.

Honi aurre egiteko, Gazebo-n erabiltzeko SDF formatua sortu zen; mundu mailatik robot mailaraino nahi dugun guztia deskribatzea posible egiten duen formatua. Hedakorra da eta bertan elementuak gehitu eta aldatzea erraza da.

URDF fitxategi bat Gazebo-n erabili nahi badugu, simulaziorako etiketa batzuk gehitu behar zaizkio Gazebo-n arazoak izan nahi ez baditugu. Adibidez, *link* elementu bakoitzak <*inertial*> etiketa izan behar du nahitanahiez. Aldaketa hauei esker ez daukagu beste SDF fitxategi bat sortu beharrik deskripzio fitxategiak bikoiztuz; Gazebo arduratuko da URDF fitxategia SDF bihurtzeaz.

Hona hemen bi gurpil eta oinarria dituen robot bakun baten URDF eredua:

```
<robot name="my_robot">
  <!-- Base link -->
  <link name="base_link">
    <inertial>
```

```
    <mass value="1"/>
    <inertia ixx="100" ixy="0" ixz="0" iyy="100" iyz="0" izz="100"/>
    <origin/>
  </inertial>
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
       <box size="1 0.5 0.25"/>
    </geometry>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
       <box size="1 0.5 0.25"/>
    </geometry>
  </collision>
</link>
<!-- Left Wheel -->
<link name="l_wheel">
  <inertial>
    <mass value="1"/>
    <inertia ixx="100" ixy="0" ixz="0" iyy="100" iyz="0" izz="100"/>
    <origin/>
  </inertial>
  <visual>
    <origin xyz="0 0 0" rpy="1.570795 0 0" />
    <geometry>
       <cylinder length="0.1" radius="0.2" />
    </geometry>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="1.570795 0 0" />
    <geometry>
       <cylinder length="0.1" radius="0.2" />
    </geometry>
  </collision>
</link>
<joint name="joint_l_wheel" type="continuous">
  <parent link="base_link"/>
  <child link="l_wheel"/>
  <origin xyz="0 0.30 0" rpy="0 0 0" />
  <axis xyz="0 1 0" rpy="0 0 0" />
</joint>
<!-- Right Wheel -->
<link name="r_wheel">
  <inertial>
    <mass value="1"/>
    <inertia ixx="100" ixy="0" ixz="0" iyy="100" iyz="0" izz="100"/>
    <origin/>
  </inertial>
  <visual>
    <origin xyz="0 0 0" rpy="1.570795 0 0" />
    <geometry>
```

```
58          <cylinder length="0.1" radius="0.2" />
59        </geometry>
60      </visual>
61      <collision>
62        <origin xyz="0 0 0" rpy="1.570795 0 0" />
63        <geometry>
64          <cylinder length="0.1" radius="0.2" />
65        </geometry>
66      </collision>
67    </link>
68    <joint name="joint_r_wheel" type="continuous">
69      <parent link="base_link"/>
70      <child link="r_wheel"/>
71      <origin xyz="0 -0.30 0" rpy="0 0 0" />
72      <axis xyz="0 1 0" rpy="0 0 0" />
73    </joint>
74  </robot>
```

4.3 Mundu fitxategiak (*world*)

Ereduen atalean azaldu dugun bezala, ingurunea zehazten duten fitxategiek ere SDF formatua jarraitzen dute.

Ingurunea (*world*), robot eta objektu multzoa (adibidez eraikinak, mahaiak, eta argiak) nahiz parametro globalak deskribatzen dituen termino bat da.

Ingurunea zehaztuko dugun SDF formatuko fitxategian, nahi ditugun ereduak gehi ditzakegu: robotak, eta hauek mugituko diren ingurunea, adibidez.

Ondorengo lerroetan hiru ereduz osatutako ingurunearen konfigurazioa ikus daiteke:

```
1  <?xml version="1.0" ?>
2  <sdf version="1.4">
3    <world name="my_world">
4      <include>
5        <uri>model://ground_plane</uri>
6      </include>
7      <include>
8        <uri>model://sun</uri>
9      </include>
10     <include>
11       <uri>model://kaxa</uri>
12     </include>
13   </world>
14 </sdf>
```

4.4 Plugin-ak

Plugin-ak Gazebo-rekin komunikatzeko mekanismo sinple eta erosoak dira. Komando lerro bidez kargatu daitezke, edo mundu/eredu fitxategi batean zehaztuz. Plugin-ei esker garatzaileak ia Gazebo-ren aspektu guztiak kontrolatzeko gai dira.

Funtzio desberdinak izan ditzakete; adibidez: ereduak mugitu, gertaerei erantzun, aurrebaldintzen arabera eredu berriak txertatu...

Plugin hauek bost mota ezberdinetan antolatzen dira: mundua (*world*), eredua (*model*), sentsorea (*sensor*), sistema (*system*) eta bistaratzea (*visual*). Horietako bakoitza Gazebo-ren osagai batek kontrolatzen du.

5. KAPITULUA

Gure ingurunea

Aurkibidea

5.1 Marisorgin

5.1.1 Sarrera

Marisorgin RWI etxeko B21 robota da, Euskal Herriko Unibertsitateko Informatika Fakultatean ikerketa lanak egiten dituen Robotika eta Sistema Autonomoen Ikerketa Taldeak (RSAIT) eskuratu zuen lehenengo robota (1996an), hain zuzen ere. Robot hau ROS sistemaren bidez kontrolatzen du taldeak.

Jatorrizko robotak aldaketa sakonak izan ditu egungo Marisorgin izan arte. Adibidez, proiektu honetan garrantzitsua izango den laser bat kokatu zaio gorputzaren gainean, eta baita Kinect kamera bat ere. Bestelako sentsoreak ere baditu, hala nola, sonarrak, infragorriak eta talka-sentsoreak, baina proiektu honetan eraginik izango ez dutenez, ez ditugu azalduko.

5.1.2 Egitura

Itxurari erreparatuz, nabari da ez dela robot berria. Ia guztia gorri kolorekoa da, beraz, ondo ikusteko modukoa. Metro bat baino altuagoa da, eta robot zirkular honek 27 cm-ko erradioa dauka oinarrian eta 25 cm-koa gorputzean.

Marisorgin robota zati ezberdinez osatuta dago. Behetik gorantz begiratzen badugu, lehenik eta behin lau gurpil topatu beharko genituzke, baina arretaz erreparatzen ez badugu, ez ditugu ikusiko, ia osorik oinarriak estaltzen baititu. Oinarria gorputzarekin lotuta dago "*gerria*ren" bitartez. Gorputzak 360 graduko biraketa eman dezake gurpilekin batera. Oinarria, berriz, ez da inoiz mugituko.

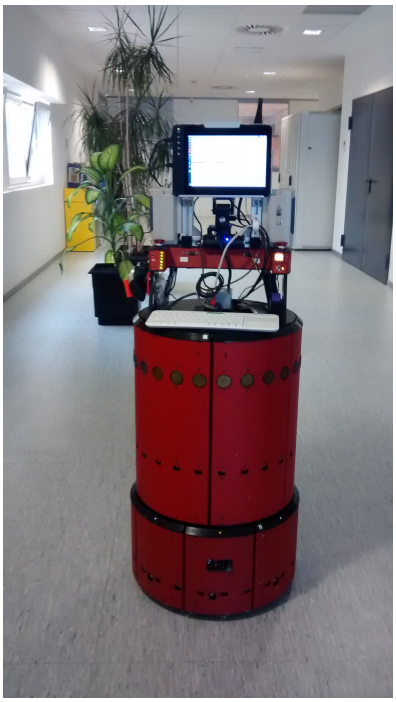
Gorputzaren gaineko plataforman laserra kokatzen da, gutxi gorabehera 1,25 metroko altueran, eta altuera horretan dauden oztopoen berri emango digu.

5.1a eta 5.1b irudietan Marisorgin robota ikus daiteke.

5.1.3 Mugimendu-sistema

Marisorginek mugimendu-sistema sinkronoa dauka. Aldi berean mugitzen diren lau gurpil ditu (ikus 5.2a irudia), eta errotazioa eta translazioa independenteak dira: translaziorako bi motor ditu, eta beste bat errotaziorako. Robot hau ez da holonomoa, eta on-

(a) Marisorgin robotaren aurreko aldea



(b) Marisorgin robotaren atzeko aldea

5.1 Irudia: Marisorgin robota

dorioz, X ardatzean bakarrik mugituko da, esan bezala, mugimendu-sistema sinkronoa baita.

Gurpilen mugimenduaz gain, aipatu dugun moduan, Marisorginek "*gerria*" ere mugitzen badaki: 360 graduko bira eman dezake oinarria mugitu barik. Hau da, gorputzak gurpilekin batera biratzen du, oinarriak beti orientazio bera mantentzen duen bitartean.

Bestalde, odometria oso garrantzitsua izango da nabigaziorako; hau da, mugimendu-sentsoreen datuen erabilera denboran zehar gertatutako posizio aldaketa estimatzeko, hasierako kokapena kontutan hartuta.

5.1.4 Laserra

Gorputz gainean kokatutako laserra Hokuyo etxeko UTM-30LX laserra da (ikus 5.2b irudia). Irakurketa bakoitzean distantziaren informazioa ematen digu. 10 cm-tik 30 metrorainoko atzipena dauka bermatuta, eta baldintza onenetan 60 metroraino irits daiteke. Angeluari dagokionez, 270 graduko irismena dauka. Sentsore honek laser diodo semieroale batekin funtzionatzen du.

0.1 eta 10 metro bitartean ±30mm-ko zehaztasuna du, eta 10 eta 30 metro artean, berriz, ±50mm-koa.

Erantzun azkarra duenez (25msec/scan) egokia da mugimendu azkarreko robotetan erabiltzeko.

Honez gain, USB kable bidez konektatzen da eta 12V-eko elikadura du.

(a) Marisorginen lau gurpiletako bat

(b) Hokuyo UTM-30LX laserra

5.2 Irudia: Robotaren osagaiak

5.2 Informatika Fakultateko hirugarren solairua

Euskal Herriko Unibertsitateak Donostian daukan Informatika Fakultatea hiru solairuz osatuta dago 0.a (*hall*-a) eta solairuartea kontutan hartu gabe.

Gure proiektuak eraikin honen azken solairua erabiliko du ingurune gisa (ikus 5.3 irudia). Hirugarren solairu honek 26 bulego, ikerketa taldeek erabiltzen dituzten 21 gela, sei komun, bi korridore zabal, sei korridore estu, hiru eskailera (bi beheko solairura jaisteko eta bat goiko terrazara igotzeko) eta igogailu bat ditu. Marisorgin hauetan guztietan zehar mugituko da, eskaileretan eta igogailuan izan ezik.

Mugimendua posible izateko, noski, gela bakoitzean 90 cm-ko zabalera duen ate bat egongo da, eta baita korridoreetan ere. Komuneko ateak, berriz, 10 cm estuagoak dira.

305 gelan Marisorginen abiapuntua izango den RSAIT Ikerketa Taldearen laborategia dago (ikus 5.4 irudia).

(a) Korridore estua

(b) Korridore zabala

5.3 Irudia: Informatika Fakultateko hirugarren solairua

5.4 Irudia: Informatika Fakultateko hirugarren solairuko planoa eta Marisorginen abiapuntua

6. KAPITULUA

Marisorgin robotaren simulazioa Gazebo-n

Aurkibidea

6.1 Simulazioa

4. atalean azaldu dugun moduan, roboten simulazioa oinarrizko tresna da robotekin lan egiten duen edonorentzat. Ondo diseinatutako simulagailu batek posible egiten du denbora gutxian algoritmoak probatu, robotak diseinatu, eta erregresio probak burutzea jokaleku errealistak erabiliz.

Honez gain, simulazioak abantaila asko ditu: robot erreala martxan jartzeko denbora eta lana saihestea (behin eta berriz egin behar denean gehienbat), mantentze kostuak ekiditea, robota errealitatean ibiliko den ingurunea simulatzea (leku horretara bertaratu behar izan gabe, eta bertan gerta litezkeen arriskuak ekidinez), etab. Gainera, robot erreala eskuragarri ez badago ere, simulagailuaren bidez robotarekin lanean has gaitezke arazorik gabe.

Aurretik aipatu dugun bezala, proiektu honetan erabilitako simulagailua Gazebo da.

6.2 Ingurunea

Gazebo-ren oinarrizko ideiak azaldu ditugunean, ingurunea SDF formatua jarraitzen duten ereduez osatuta dagoela ikusi dugu. Gure eredu nagusia Informatika Fakultateko hirugarren solairua izango da. Eredu hau 1:1 eskalan sortuko dugu ahalik eta simulazio errealena lortzeko asmoz. Eredu fitxategian paretak, zutabeak eta lurzorua definituko ditugu.

Gazebo-n ingurune hau sortzeko lehenengo pausoa ereduaren direktorioa eta beharrezko bi fitxategiak sortzea izango da; batetik, konfigurazio fitxategia, eta bestetik, eredua definitzen duen SDF fitxategia.

Konfigurazio fitxategian ereduaren izena, bertsioa, erabilitako SDF bertsioa, egileari buruzko datuak eta ereduaren deskribapena jarriko ditugu, jarraian ikus daitekeen moduan.

```
1 <?xml version="1.0" ?>
2 <model>
3    <name>Hiru_solairua</name>
4    <version>1.0</version>
5    <sdf version="1.4">model.sdf</sdf>
6    <author>
7       <name>Oihane Parra</name>
```

```
8       <email>oparra002@ikasle.ehu.es</email>
9     </author>
10    <description>Donostiako Informatika Fakultateko hirugarren
          solairua.</description>
11 </model>
```

Ondoren, *Hiru_solairua* izeneko eredua deskribatzen hasiko gara. Fitxategiak eskema hau jarraitu beharko du:

```
1 <?xml version='1.0'?>
2 <sdf version='1.4'>
3   <model name='Hiru_solairua'>
4     <pose>0 0 0 0 -0 0</pose>
5     <!-- Hemen elementuak gehituko ditugu -->
6     <static>1</static>
7   </model>
8 </sdf>
```

<*static*> etiketari 1 balioa eman diogu eredu hau ez delako dinamikoa.

Lehenik eta behin, fakultateko kanpoko paretak zehaztuko ditugu, eta ondoren barrukoak (gelak eta korridoreak sortzen dituztenak).

Elementu bakoitza izendatzeko jarraitu den irizpidea ondorengoa da: kanpoko paretak *Wall_O<Zenbakia>* (*Outside*) izango dira, eta barrukoak bitan banatuta daude, aurre aldekoek *Wall_IF<Zenbakia>* (*Inside*, *Front*) izena dute, eta atzeko aldekoek *Wall_IR<Zenbakia>* (*Inside*, *Rear*). Honen arrazoi nagusia, pareta kopuru oso handia izateagatik nahasketak ekiditea eta lana erraztea izan da.

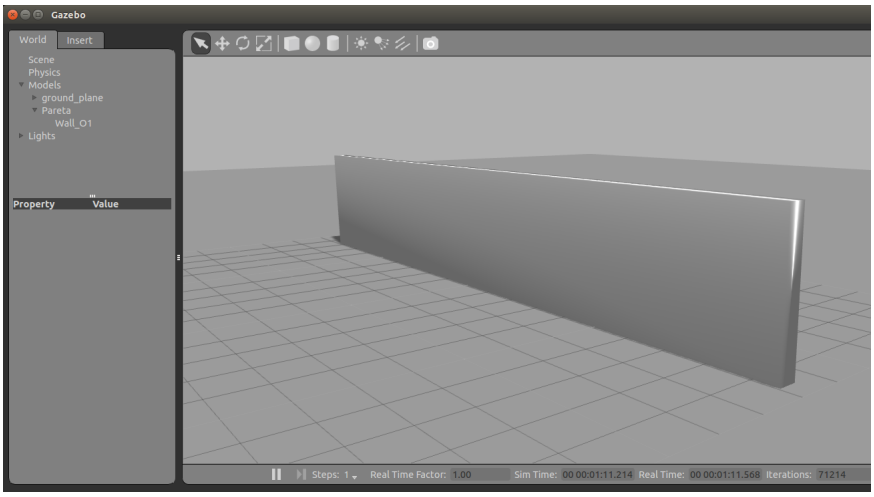
Pareta bakoitzeko jarraian dauden lerro hauen antzekoak gehituko ditugu, beti ere <*link*> etiketa erabiliz:

```
1 <link name='Wall_O1'>
2     <collision name='Wall_O1_Collision'>
3       <geometry>
4         <box>
5           <size>17.9 0.4 3</size>
6         </box>
7       </geometry>
8       <pose>0 0 1.5 0 -0 0</pose>
9     </collision>
10    <visual name='Wall_O1_Visual'>
11      <pose>0 0 1.5 0 -0 0</pose>
12      <geometry>
13        <box>
14          <size>17.9 0.4 3</size>
15        </box>
```

```
16        </geometry>
17      </visual>
18      <pose>8.95 2.2 0 0 -0 -3.14159</pose>
19  </link>
```

Kode zati horretan *Wall_O1* paretaren deskribapena dago, <*collision*>, <*visual*> eta <*pose*> etiketen bidez. Azken finean, *box* edo kutxa motako elementu bat gehitzen ari gara; 17,9 metroko luzera, 40 cm-ko zabalera eta 3 metroko altuera dituen kutxa, hain zuzen ere. <*pose*> etiketen bidez paretaren zentroaren kokapena eta orientazioa definituko ditugu.

6.1 irudian kode horren emaitza ikus daiteke simulagailuan.



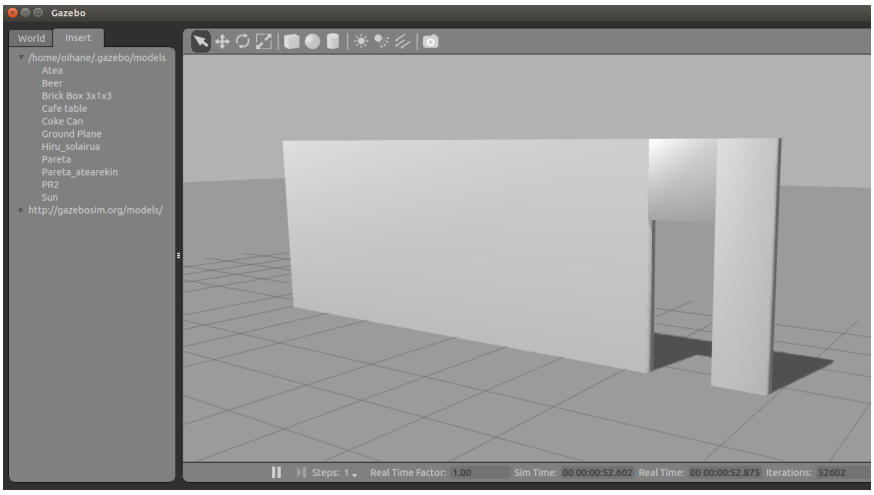
6.1 Irudia: *Wall_O*1 pareta Gazebo-n

Ateak simulatzeko, paretetan zuloak irudikatuko ditugu. Hau lortzeko, pareta zati desberdinetan banatzen da. Demagun pareta batean ate bakarra dagoela. Orduan, horma hori hiru zatitan banatuko dugu: atearen ezker aldeko pareta zatia, atearen goikoa eta atearen eskuinekoa. Kasu honetan, aurreko kode zatia hirukoiztu egingo da, hiru kolisio eremu eta beste hiru bistaratze eremu definituz. Bigarren pareta zatia, esan bezala, ate gainekoa da. Hau lortzeko, paretaren zentroa 2,5 metroko altueran dago, 0,9 metroko zabalera du eta paretaren altuera metro batekoa da. Guzti hori hurrengo lerroetan ikus daiteke, eta honen emaitza, aldiz, 6.2 irudian.

```
1    <link name='Wall_IF3'>
2      <collision name='Wall_IF3_Collision_1'>
3        <geometry>
4          <box>
```

```
          <size>7.3 0.2 3</size>
        </box>
      </geometry>
      <pose>0.1 3.65 1.5 0 -0 1.5708</pose>
    </collision>
    <visual name='Wall_IF3_Visual_1'>
      <pose>0.1 3.65 1.5 0 -0 1.5708</pose>
      <geometry>
        <box>
          <size>7.3 0.2 3</size>
        </box>
      </geometry>
    </visual>
    <collision name='Wall_IF3_Collision_2'>
      <geometry>
        <box>
          <size>0.9 0.2 1</size>
        </box>
      </geometry>
      <pose>0.1 7.75 2.5 0 -0 1.5708</pose>
    </collision>
    <visual name='Wall_IF3_Visual_2'>
      <pose>0.1 7.75 2.5 0 -0 1.5708</pose>
      <geometry>
        <box>
          <size>0.9 0.2 1</size>
        </box>
      </geometry>
    </visual>
    <collision name='Wall_IF3_Collision_3'>
      <geometry>
        <box>
          <size>0.7 0.2 3</size>
        </box>
      </geometry>
      <pose>0.1 8.55 1.5 0 -0 1.5708</pose>
    </collision>
    <visual name='Wall_IF3_Visual_3'>
      <pose>0.1 8.55 1.5 0 -0 1.5708</pose>
      <geometry>
        <box>
          <size>0.7 0.2 3</size>
        </box>
      </geometry>
    </visual>
    <pose>5.15 2 0 0 -0 0</pose>
  </link>
```

Paretez gain, zutabeak eta lurra ere gehitu ditugu ereduan sistema berdina erabiliz. Zutabeen kasuan, 3 metroko altuera, 80 zentimetroko luzera eta 30 zentimetroko sakonera dute. Horietako zutabe baten konfigurazioa jarraian ikus daiteke:



6.2 Irudia: *Wall_IF*3 pareta eta bertako atea Gazebo-n

```
<link name='Column_1_1'>
    <collision name='Column_1_1_Collision'>
      <geometry>
        <box>
          <size>0.8 0.3 3</size>
        </box>
      </geometry>
      <pose>0 0 1.5 0 -0 0</pose>
    </collision>
    <visual name='Column_1_1_Visual'>
      <pose>0 0 1.5 0 -0 0</pose>
      <geometry>
        <box>
          <size>0.8 0.3 3</size>
        </box>
      </geometry>
    </visual>
    <pose>0.25 2.55 0 0 -0 1.5708</pose>
   </link>
```

Lurra adierazteko, berriz, 0,1 metroko sakonerako bost zatitan banatu da. Hona hemen horietako zati bat:

```
<link name='Floor_1'>
    <collision name='Floor_1_Collision'>
      <geometry>
        <box>
          <size>27.9 2 0.1</size>
        </box>
      </geometry>
      <pose>0 0 0 0 -0 0</pose>
```

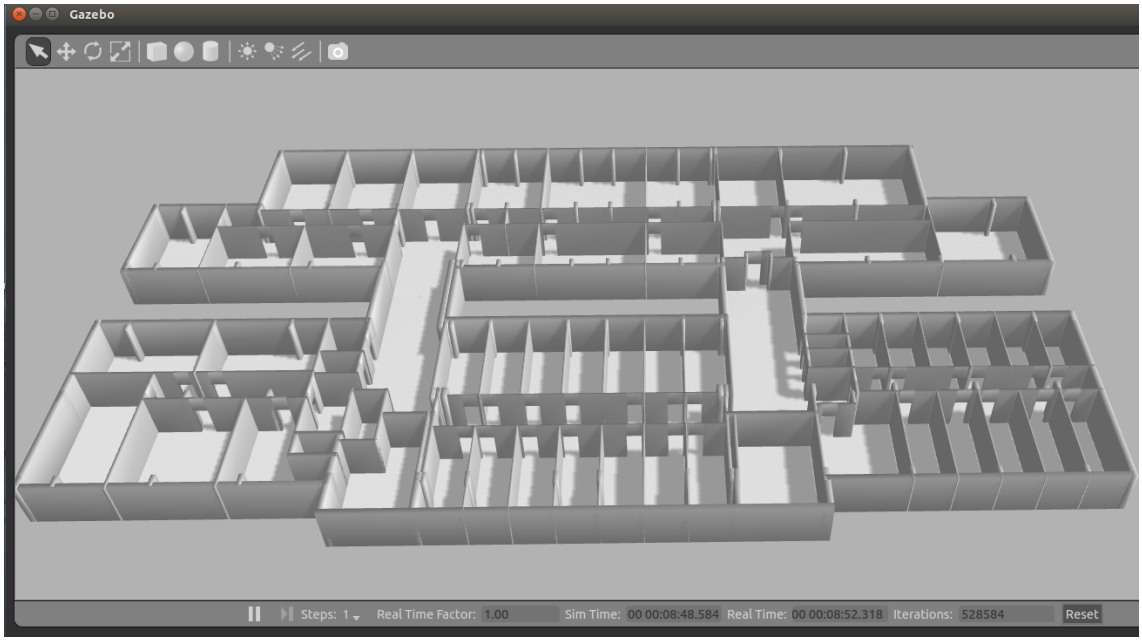
```
9       </collision>
10      <visual name='Floor_1_Visual'>
11        <pose>0 0 0 0 -0 0</pose>
12        <geometry>
13          <box>
14            <size>27.9 2 0.1</size>
15          </box>
16        </geometry>
17      </visual>
18      <pose>31.85 1.4 0 0 -0 0</pose>
19    </link>
```

Paretak, zutabeak eta lurra gehitu ondoren, amaitu dugu gure Informatika Fakultateko hirugarren solairuaren eredua definitzen.

Zati honetan azaldutako kodea ulergarriagoa izateko asmoz, ez dira materiala zehazten duten etiketak jarri, baina proiektuan sortutako kodeari erreparatzen badiogu konturatuko gara fakultateko koloreak errespetatu direla: paretak eta zutabeak zuriak, eta lurra gris kolorekoa.

Korridore eta geletako oztopoak ez ditugu ereduan jarriko, ondoren dinamikoki gehitu eta laserraren laguntzarekin ekidingo baitira.

Solairu osoaren simulazio ingurunea 6.3 irudian ikus daiteke.



6.3 Irudia: Informatika Fakultateko hirugarren solairua Gazebo-n

6.3 Robotaren eredua

Robotaren eredua *mmari.urdf* izeneko URDF fitxategi batean definituta daukagu.

Marisorginen ereduak benetako robotaren antza handia du, 6.4 irudian ikus daitekeen bezala. Zati garrantzitsuenak oinarria, gorputza eta goiko plataforma dira; eta azken honen gainean laserra kokatzen da. Robot errealean bezala, gorputzak 360 graduko biraketa eman dezake. Bestalde, eredu honetan ez dugu gurpilik aurkituko, ez baitira beharrezkoak izan Marisorgin Gazebo-n mugitu ahal izateko.

RSAIT taldeak garatutako URDF fitxategi honi lerro batzuk gehitu zaizkio Gazebo-n simulazioa ondo burutzeko.

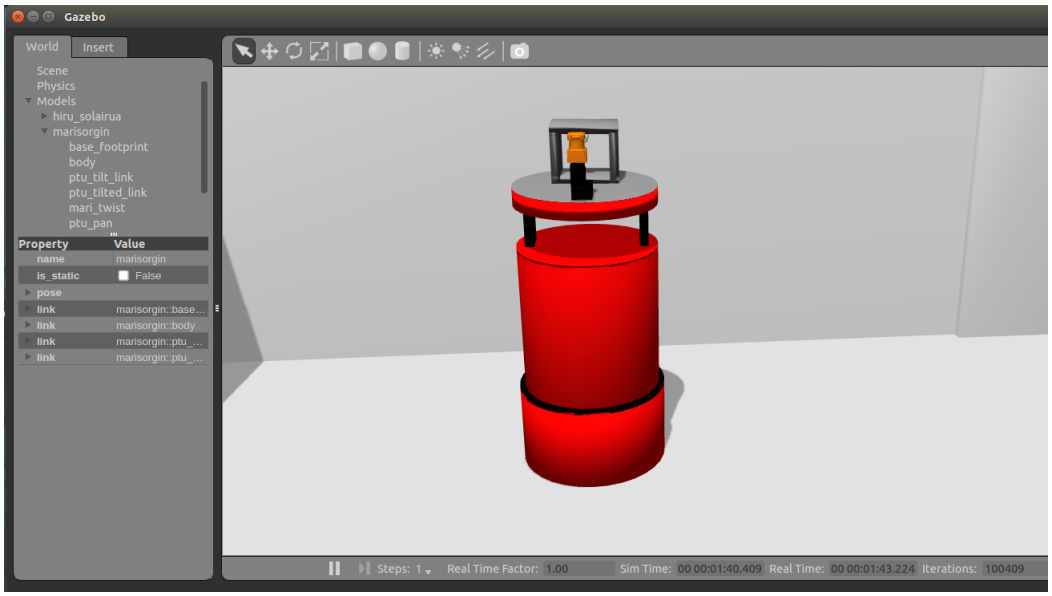
Batetik, *mari_twist* (oinarria eta gorputzaren arteko *joint*-a) mugitu ahal izateko <*transmission*> etiketa gehitu behar da:

```
<transmission name="tran1">
   <type>transmission_interface/SimpleTransmission</type>
   <joint name="mari_twist">
     <hardwareInterface>EffortJointInterface</hardwareInterface>
   </joint>
   <actuator name="motor1">
     <hardwareInterface>EffortJointInterface</hardwareInterface>
     <mechanicalReduction>1</mechanicalReduction>
   </actuator>
</transmission>
```

Bigarrenik, plugin-ak eta itxura (koloreak) hobetzeko zehaztasunak ere jarriko ditugu. Plugin-en konfigurazioa hurrengo atalean ikusiko dugu.

6.4 Plugin-ak

Marisorginen simulazioa egiteko bi plugin konfiguratu behar izan dira. Lehenengoak laserretik informazioa jasotzea ahalbidetuko du, eta bigarrenak, berriz, Marisorgin mugitzea. Bi plugin hauek *mmari.urdf* fitxategian zehaztu dira eta Marisorgin Gazebo-n martxan jartzen dugunean kargatuko dira.



6.4 Irudia: Marisorgin Gazebo-n

6.4.1 Laserra

Laserra sentsore errealaren ahalik eta antzekoena izaten saiatu gara; simulatutakoa ere Hokuyo nodo bat da eta Gazebo/ROS-en *gazebo_ros_head_hokuyo_controller* izeneko plugin-a erabili da. Ezaugarriak jatorrizkoarekin bat datoz:

- Eguneratze maiztasuna: 40 Hz
- Bereizmen angeluarra: 0.25º
- Atzipen irismena: minimoa 0.10 metro eta maximoa 30 metro.
- Laserraren irismen angeluarra 270 gradukoa den arren, Marisorgin robot errealean bakarrik 180 graduko irakurketak egiteko konfiguratuta dago, robotaren osagaiak oztopoekin ez nahasteko. Hori dela eta, simulazioan berdina egitea erabaki da ($-\pi$ eta π radian artean bilatuko ditu oztopoak).

Gainera, garrantzitsuena laserraren *topic* eta *frame*-a ondo zehaztea izango da; hau da, ondoren nabigazio sistemak datuak irakurtzeko erabiliko dituen *topic* eta *frame* berdinak izatea. Hauek errealitateko Marisorgin-ekin bat datoz: *topic*-a */scan* da eta *frame*-a, berriz, */laser*.

Bestalde, <*visualize*> etiketak Gazebo-n laserraren izpiak ikusteko aukera eskaintzen du, eta hau oso interesgarria izan daiteke konfigurazioa ondo dagoela egiaztatzeko eta zein objektu ikusten dituen jakiteko.

Jarraian, plugin honen konfigurazio lerroak ikus daitezke.

```
<!-- hokuyo -->
<gazebo reference="laser">
  <sensor type="ray" name="head_hokuyo_sensor">
    <pose>0 0 0 0 -0 0</pose>
    <visualize>true</visualize>
    <update_rate>40</update_rate>
    <ray>
      <scan>
        <horizontal>
          <samples>720</samples>
          <resolution>1</resolution>
          <min_angle>-1.570796</min_angle>
          <max_angle>1.570796</max_angle>
        </horizontal>
      </scan>
      <range>
        <min>0.10</min>
        <max>30.0</max>
        <resolution>0.01</resolution>
      </range>
      <noise>
        <type>gaussian</type>
             <mean>0.0</mean>
        <stddev>0.01</stddev>
      </noise>
    </ray>
<plugin name="gazebo_ros_head_hokuyo_controller" filename="
    libgazebo_ros_laser.so">
      <topicName>/scan</topicName>
      <frameName>/laser</frameName>
    </plugin>
  </sensor>
 </gazebo>
```

6.4.2 Oinarriaren mugimendua

Esan bezala, beste plugin bat ere beharrezkoa izan da: Marisorgin mugitu ahal izateko plugin-a, hain zuzen ere. Honetarako, robotaren oinarria mugituko duen plugin bat erabili da: Gazebo/ROS-en *Planar Move* plugin-a, non objektu bat plano horizontalean mugitzen duen *geometry_msgs/Twist* mezu mota erabiliz. Bere funtzionamendua objektuari zikloro

abiadura lineal eta angeluarrak bidaltzean datza. Baina plugin honek arazo bat dauka, robot holonomoentzat dago prestatuta, hau da, abiadura linealak bai X eta bai Y ardatzetan bidaltzen ditu, eta ondorioz, plugin honi aldaketa batzuk egin behar izan zaizkio robota bakarrik X ardatzean mugitzeko.

Egindako aldaketa garrantzitsuenak laburbilduz, jasotako Y ardatzeko abiadura zerora jartzea, eta odometria kalkulatzerakoan ere, Y abiadura kontutan ez hartzea izan dira. Modu honetan, nahiz eta Y ardatzean mugitzeko agindua pasa robotari, ez du kasurik egingo, benetako Marisorginen modura.

Plugin honen moldaketa egiteko, ROS pakete berri bat erabili da: *gazebo_plugins*.

mmari.urdf fitxategiko ondorengo lerroek plugin-aren konfigurazioa erakusten digute, eta oraingo honetan ere ikus dezakegu simulatutako robotaren *frame*-ak bat datozela Marisorgin errealak konfiguratuta dituenekin. *Topic*-etan, aldiz, aldaketa txiki bat dago, Gazebo-n */mari/cmd_vel* eta */mari/odom* ditugun bitartean, Marisorginen *driver*-an konfiguratutako *topic*-ak */mari_driver/cmd_vel* eta */mari_driver/odom* dira, hurrenez hurren.

```
<gazebo>
  <plugin name="object_controller" filename="libnon_holonomic_drive
      .so">
    <commandTopic>/mari/cmd_vel</commandTopic>
    <odometryTopic>/mari/odom</odometryTopic>
    <odometryFrame>/odom_combined</odometryFrame>
    <odometryRate>20.0</odometryRate>
    <robotBaseFrame>/base_footprint</robotBaseFrame>
  </plugin>
</gazebo>
```

6.4.3 *Joint*-en mugimendua

Eredu osoa mugitzeaz gain, *mari_twist joint*-a (“gerria”) ere mugitzea nahi badugu, URDF-an gehitutako <*transmission*> etiketaz gain, plugin berri bat behar dugu: *ros_control* plugin-a, Gazebo-n *joint*-ak mugitzea ahalbidetzen duena, hain zuzen ere.

Hona hemen plugin honen konfigurazioa:

```
<gazebo>
  <plugin name="gazebo_ros_control" filename="libgazebo_ros_control
      .so">
    <robotNamespace>/mari</robotNamespace>
```

```
    <robotSimType>gazebo_ros_control/DefaultRobotHWSim</robotSimType>
  </plugin>
</gazebo>
```

6.5 Martxan jarri

Behin *mari_description* paketean robotaren eredua Gazebo-n erabili ahal izateko konfiguratuta, eta Informatika Fakultateko hirugarren solairuaren eredua prestatuta, guztia elkartu eta martxan jarri beharko dugu. Horretarako, beste ROS pakete bat sortuko dugu: *mari_gazebo*.

6.5.1 Eredua

Lehenik eta behin, aurretik sortu dugun Informatika Fakultateko hirugarren solairuaren eredua *mari_gazebo* paketean kopiatuko dugu, *models* izeneko karpeta batean. Gazebo martxan jartzean eredu hau kargatu ahal izateko, Gazebo-ri eredu honen kokapena non dagoen esango diogu. Horretarako, *GAZEBO_MODEL_PATH* aldagaiari ereduaren bide absolutua emango diogu:

```
# export GAZEBO_MODEL_PATH=$HOME/catkin_ws/src/mari_gazebo/models
```

Hau hasieratze script batean gehi dezakegu erabiltzen dugun bakoitzean eskuz ez exekutatzeko.

6.5.2 Mundua

Pakete berri honetan, bigarrenik, *worlds* izeneko karpetan Marisorginen ingurunea definitu dugu bi eredu gehituz: eguzkia (*sun* izenekoa) eta guk garatutako fakultateko hirugarren solairuari dagokiona (*hiru_solairua*). Fitxategi honen izena *mari.world* da; eta bere edukia, honakoa:

```
<?xml version="1.0" ?>
<sdf version="1.4">
   <world name="default">
      <include>
         <uri>model://sun</uri>
```

```
      </include>
      <include>
         <uri>model://hiru_solairua</uri>
         <name>hiru_solairua</name>
         <pose>0 0 0 0 0 0</pose>
      </include>
      <scene>
         <grid>false</grid>
      </scene>
   </world>
</sdf>
```

6.5.3 *Launch* fitxategia

launch izeneko beste direktorio batean, aldiz, *mari.launch* fitxategia sortu dugu, Gazebo martxan jarriko duena, behar dugun guztia kargatuz (ingurunea, Marisorgin eta pluginak).

Errazena Gazebo/ROS-en *empty_world.launch* fitxategia erabiltzea izango da, bertan parametro asko zehaztuta baititugu jadanik. Aldatu beharreko parametro bakarra mundua da, aurreko atalean azaldu dugun *mari.world* mundua, hain zuzen ere. Parametroak zehazteaz gain, bi nodo ere sortuko ditu: *Gazebo server* eta *Gazebo client*.

Hurrengo pausoa robotaren eredua bilatzea da, URDF fitxategia non dagoen esanez. Gainera, nodo bat sortuko dugu Marisorginen eredua kargatzeko. Nodo hau martxan jartzerakoan robotaren hirugarren solairuko hasierako posizioa zehaztuko dugu. Gure kasuan, RSAIT Ikerketa Taldeko laborategia.

```
<launch>
  <param name="/use_sim_time" value="true" />
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find mari_gazebo)/worlds/mari.
       world"/>
  </include>

  <!-- Load the URDF into the ROS Parameter Server -->
  <param name="robot_description" textfile="$(find mari_description)/
     urdf/mmari.urdf"/>

  <!-- Run a python script to the send a service call to gazebo_ros
     to spawn a URDF robot -->
  <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model"
     respawn="false" output="screen"
    args="-urdf -model marisorgin -param robot_description -x 10 -y
       13"/>
```

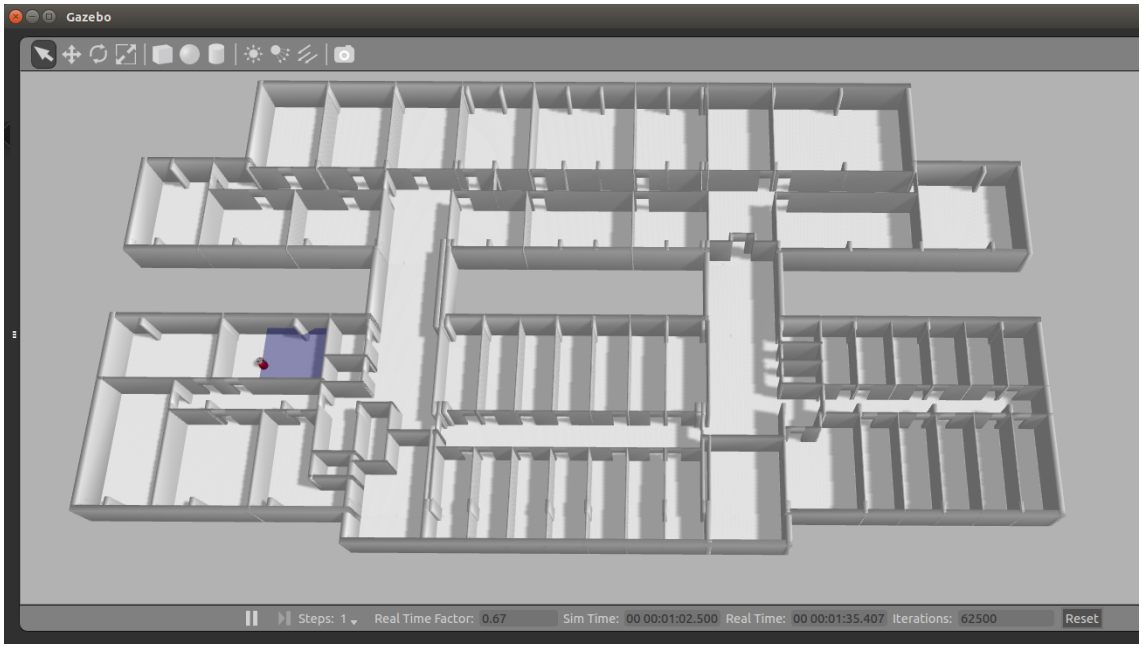
```
13 </launch>
```

6.5.4 Exekuzioa

Launch fitxategia exekutatu, eta ondorioz, konfiguratutako guztia Gazebo-n martxan jartzeko, *roslaunch* komandoa erabiliko dugu:

```
# roslaunch mari_gazebo mari.launch
```

Arazorik egon ez bada, Gazebo irekiko da eta hirugarren solairuko elementu guztiak nahiz Marisorgin ikusiko ditugu bertan, 6.5 irudian bezala. Urdinez ikusten dena laserraren izpiak dira.



6.5 Irudia: Simulazio osoa Gazebo-n

6.6 Komando lerro bidezko mugimendua

Robotaren oinarria mugitzen duen plugin-aren funtzionamendu zuzena egiaztatzeko, batetik, komando lerroa erabiliko dugu, eta bestetik, teklatu bidez mugiaraziko duen programa bat.

Komando lerro bidez, *geometry_msgs/Twist* motako mezu bat argitaratu beharko dugu */mari/cmd_vel topic*-ean. Hori bai, gero eta balio handiagoak, orduan eta azkarrago mugituko da Marisorgin.

Robota X ardatzean mugitzeko:

```
# rostopic pub -1 /mari/cmd_vel geometry_msgs/Twist - '[1.0, 0.0, 0.0]' '[0.0, 0.0, 0.0]'
```

Robotak bira dezan ondorengoa exekutatuko dugu:

```
# rostopic pub -1 /mari/cmd_vel geometry_msgs/Twist - '[0.0, 0.0, 0.0]' '[0.0, 0.0, 1.0]'
```

Robot ez-holonomoa denez, Y ardatzean ez mugitzeko konfiguratu dugu. Hala den ala ez egiaztatu beharko dugu:

```
# rostopic pub -1 /mari/cmd_vel geometry_msgs/Twist - '[0.0, 1.0, 0.0]' '[0.0, 0.0, 0.0]'
```

Dena den, nabigazio paketea konfiguratuko dugunean ez du abiadurarik bidaliko Y ardatzean, beraz, bestela ere ez litzateke arazorik egongo.

6.7 Teklatu bidezko mugimendua

Komando lerroa erabiliz mezuak argitaratzea baino erosoagoa teklatua erabiltzea da. Hori dela eta, *turtlebot_teleop* paketeaz baliatuko gara Marisorgin fakultateko korridore eta geletan zehar mugitzeko, eta guztiak normaltasunez funtzionatzen duela egiaztatzeko.

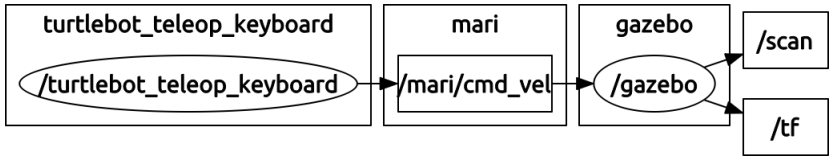
Honetarako, pakete horretako *keyboard_teleop.launch* fitxategian lerro bat gehituko da erabili beharreko *topic*-a zein den zehazteko:

```
<remap from="turtlebot_teleop_keyboard/cmd_vel" to="/mari/cmd_vel"/>
```

Pakete honetako teklatu bidezko mugimenduak hainbat aukera eskaintzen ditu: aurrera eta atzera mugitu (X ardatza), eta ezkerrerantz edo eskuinerantz biratu (Yaw ardatza).

6.6 irudian Marisorgin mugitzeko nodo desberdinen artean komunikazioa nola egin den ikus dezakegu. Nodoak biribilduta agertzen direnak dira (*turtlebot_teleop_keyboard* eta *gazebo*), eta */mari/cmd_vel*, berriz, abiadura mezuak argitaratzeko erabiltzen den *topic*-aren izena da. *turtlebot_teleop_keyboard* nodoak *geometry_msgs/Twist* motako

mezuak argitaratuko ditu */mari/cmd_vel topic*-ean, guk sakatutako teklen arabera aldatuko diren mezuak. Gazebo-ko plugin-a, aldiz, *topic* horretara harpidetu da eta ondorioz, mugimendua gauzatuko da.



6.6 Irudia: Konputazio grafoa (teklatu bidezko mugimendua)

7. KAPITULUA

Marisorgin robotaren nabigazioa Gazebo-n

Aurkibidea

7.1 Sarrera

Marisorginen fakultateko nabigazioa burutzeko ROS-en *Navigation Stack* izeneko nabigazio pila erabiliko dugu. Nabigazio sistema hau maila kontzeptual batean nahiko sinplea da: sentsoreek eta odometriak eskainitako informazioa jasotzen du, nabigazio helburu batekin batera, eta, jarraian, robotaren oinarri mugikorrari abiadura aginduak pasako dizkio. Sistema honen behe mailako arkitektura, aldiz, konplexua da, eta aldi berean lan egin behar duten elementu askoz osatuta dago.

Nabigazio pila robotaren ezaugarrien arabera konfiguratu behar da maila altuko portaera lortu nahi badugu. Bestalde, nahiz eta pila hau ahalik eta erabilera zabalena izateko diseinatuta dagoen, erabilpen hau mugatzen duten hardware betekizun batzuk daude:

1. Soilik mugimendu sistema diferentziala duten edo holonomoak diren robotentzat dago prestatuta. Robotaren oinarri mugikorra bidalitako abiadurekin kontrolatuko da, eta abiadura hauek X abiadura, Y abiadura eta biraketa abiadura izango dira.

2. Robotak bi dimentsiotako (*planar*) laser bat beharko du. Laser hau mapak eraikitzeko eta kokapenerako erabiltzen da.

3. Nabigazio pila robot karratuentzat prestatua izan zen; horregatik, errendimendu hoberena robot karratu edo zirkularrekin izango du.

Proiektu honetan ROS-en nabigazio pila osoarekin komunikatuko gaituen pakete bat erabiliko dugu: *move_base*. Aurrerago azalduko dugu pakete honen portaera eta konfigurazioa.

Guk, zehazki, *mari_2dnav* izeneko paketea sortuko dugu nabigazioarekin lotutako fitxategiak bertan gordetzeko.

7.2 Aurrekariak

Nabigazioko pilak bere lana ondo egiteko, robota modu zehatz batean konfiguratuta egon behar da; eta horregatik, hainbat aurrebaldintza bete behar dira. Jarraian, baldintza horiek zeintzuk diren eta nola ziurtatu ditzakegun azalduko dugu.

7.2.1 ROS

Lehenengo baldintza sinplea baina garrantzitsua da. Nabigazio pilak robotak ROS erabiltzea eskatzen du.

7.2.2 Transformazio konfigurazioa

Robotak bere koordenatu-sistemen arteko erlazioei buruzko informazioa argitaratu beharko du *tf* software liburutegia erabiliz; beste modu batera esanda, robotaren transformazio-zuhaitza argitaratu behar da.

Marisorginen kasuan, nabigazioan arazorik ez izateko, laserraren eta oinarriaren arteko transformatua beharrezkoa izango da, laserrak irakurritako balioak robotaren jatorritik zein distantziara kokatuta dauden jakiteko. Beste koordenatu-sistemen artekoa, aldiz, hautazkoa izango da nabigaziorako. Hala ere, errazena guztien arteko zuhaitza osatzea dela ikusiko dugu.

Hau lortzeko modu bat baino gehiago dago. Aukera bat eskuz egitea da, transformazio hauek argitaratuko dituen programa sinple bat garatuz. Kasu honetan transformazioa zein *frame* artean egin nahi dugun zehaztu beharko dugu (gure ereduan, *base_footprint* eta *laser*), eta bien arteko aldaketa zein den; batetik, kokapena, eta bestetik, orientazioa.

Gure kasuan orientazioa berdina da, baina kokapenean esan beharko diogu X ardatzean 0.25 metroko aldea, Y ardatzean -0.011 metro eta Z ardatzean 1.242 metroko aldea dagoela. Ikusten dugunez, batez ere altueran dago diferentzia, baina 25 cm horiek ere oso garrantzitsuak dira laserraren informazioa ondo interpretatzeko.

Behin konfigurazioa amaituta, konpilatu eta nabigazio paketea martxan jarri aurretik, *tf_broadcaster* izena eman diogun programa hau exekutatu behar da:

```
# rosrun mari_setup_tf tf_broadcaster
```

Jarraian, programa txiki hori nola prestatu daiteken ikus dezakegu:

```
1 #include <ros/ros.h>
2 #include <tf/transform_broadcaster.h>
3
4 int main(int argc, char** argv){
5   ros::init(argc, argv, "robot_tf_publisher");
6   ros::NodeHandle n;
7
8   ros::Rate r(100);
```

```
9
10  tf::TransformBroadcaster broadcaster;
11
12  while(n.ok()){
13    broadcaster.sendTransform(
14      tf::StampedTransform(
15        tf::Transform(tf::Quaternion(0, 0, 0, 1), tf::Vector3(0.25,
              -0.011, 1.242)),
16        ros::Time::now(),"base_footprint", "laser"));
17    r.sleep();
18  }
19 }
```

Beste aukera bat, *robot_state_publisher* eta *joint_state_publisher* paketeez baliatzea da. Hauek automatikoki koordenatu sistema guztien arteko transformazioak argitaratuko dituzte, eta ez dugu ezer gehiago konfiguratu beharrik izango. Errazena *launch* fitxategi baten bidez (*mari_state_publisher2.launch* deiturikoa) bi nodo martxan jartzea izango da, jarraian ikus daitekeen moduan.

```
1 <launch>
2  <param name="robot_description" textfile="$(find mari_description)/
       urdf/mmari.urdf"/>
3
4   <node pkg="robot_state_publisher" type="robot_state_publisher"
        name="mari_state_pub" >
5   </node>
6   <!-- send fake joint values -->
7    <node name="joint_state_publisher" pkg="joint_state_publisher"
        type="joint_state_publisher">
8          <param name="use_gui" value="FALSE"/>
9    </node>
10 </launch>
```

7.2.3 Sentsoreen informazioa

Nabigazio pilak sentsoreen informazioa erabiltzen du inguruneko oztopoak saihesteko. Horretarako, sentsoreek uneoro *sensor_msgs/LaserScan* edo *sensor_msgs/PointCloud* mezuak argitaratu beharko dituzte.

Gure ereduan Hokuyo laser sentsore bakarra daukagu eta Gazebo-n konfiguratutako plugin-aren bidez *sensor_msgs/LaserScan* motako mezuak argitaratzen ditu */scan topic*-ean.

7.2.4 Odometria informazioa

Odometria informazioa *tf* bidez argitaratzea beharrezkoa da nabigazioa posible izateko. Hau *nav_msgs/Odometry* mezu motaren bidez egiten da.

Odometria informazioaren argitaratzea ere jadanik konfiguratuta daukagu. Zehazki, robotaren URDF ereduan definitutako Gazebo-ko *non_holonomic_drive* plugin-ak dauka zeregin hau, eta */mari/odom topic*-ean argitaratzen ditu mezuak.

7.2.5 Oinarriaren kontrolatzailea

Nabigazio pilak *geometry_msgs/Twist* motako abiadura komandoak argitaratzen ditu robota mugi dadin. Hori dela eta, nodo bat mezu horiek argitaratzen dituen *topic*-era harpidetu beharko da, eta X eta Y abiadura linealak eta Z abiadura angeluarrak jaso. Ondoren, robotaren oinarria mugitzeko motor komandoak sortuko ditu.

Odometria kalkulatzeaz gain, Gazebo-ko *non_holonomic_drive* plugin-ak oinarriaren kontrolatzaile gisa ere jokatzen du simulagailuan. Robot errealean, berriz, mugimendu-sistemaren *driver*-a arduratuko da hortaz.

Mezu horiek argitaratzen diren *topic*-a */mari/cmd_vel* da simulagailuan, eta */mari_driver/cmd_vel* Marisorgin errealean.

7.2.6 Mapa

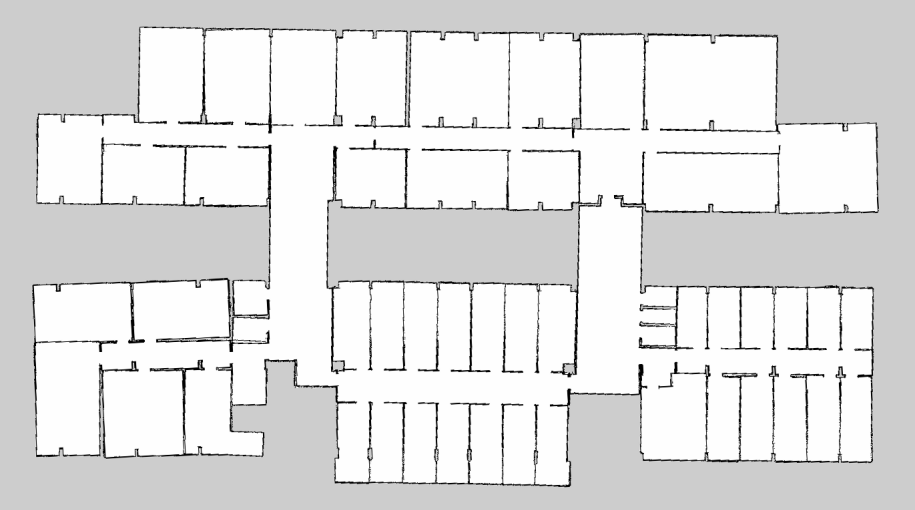
Nabigazio pilak ez du mapa bat behar funtzionatzeko. Hala ere, nabigazioa optimizatzeko asmoz, guk nabigazio sistemak erabili dezakeen Informatika Fakultateko hirugarren solairuaren mapa sortuko dugu. Mapa hau eta Gazebo-n sortutako eredua oso gauza ezberdinak dira; hobeto ulertzeko, Gazebo simulagailukoa mundu errealeko fakultatetzat har genezake.

Mapa hau eraikitzea sinplea da *gmapping* eta *map_server* paketeak erabiliz. Lehenik eta behin, Gazebo martxan jarriko dugu, ohiko moduan Marisorgin eta ingurunea kargatuz.

Jarraian, *tf* zuhaitza publikatzeko *mari_state_publisher2.launch* exekutatu eta RViz irekiko dugu.

gmapping paketeko *slam_gmapping* martxan jartzean (`rosrun gmapping slam_gmapping`) RViz-en mapa sortzen hasi dela ikus dezakegu.

Marisorgin solairu guztian zehar ordenagailuko teklatu bidez mugitzen badugu, mapa osoa osatuko da, eta amaitzen dugunean "`rosrun map_server map_saver`" exekutatzea besterik ez dugu: beharko dugun mapa *map.pgm* fitxategian gordeko da (ikus 7.1 irudia).



7.1 Irudia: Nabigaziorako ingurune simulatuaren mapa

7.2.7 Kokapena

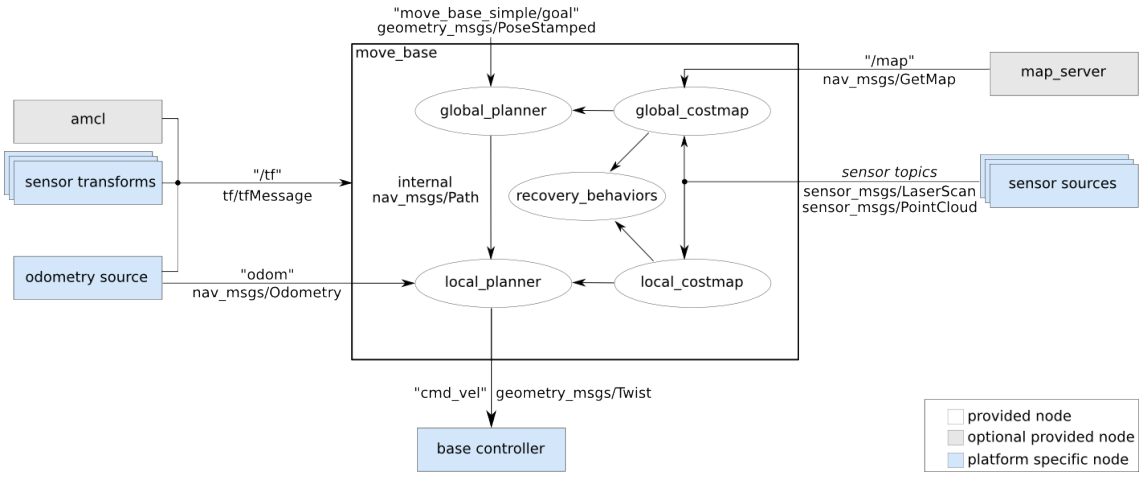
Azkenik, mapa eta odometriaren arteko transformazioa behar dugu zuhaitz osoa lotuta izateko. Hau lortzeko, *amcl* (*Augmented Monte Carlo Localization*) paketeaz baliatuko gara.

amcl bi dimentsiotan mugitzen den robot batentzako kokapen sistema probabilistikoa da. Laserraren bidez sortutako mapa bat, laserrak jasotako informazioa eta transformazio mezuak jasotzen ditu, eta ondoren, kokapen estimazioak sortzen ditu; baita mapa eta odometria arteko transformazio mezuak argitaratu ere. Robotaren hasierako posizioa eta orientazioa guk zehaztu ditzakegu, beste parametro batzuen konfigurazioarekin batera.

RViz tresna erabiliz robotaren kokapenaren banaketa probabilistikoaren egoera bistaratu daiteke.

7.3 *move_base* paketea

move_base nodoak robot baten nabigazio pila konfiguratu, martxan jarri eta harekin komunikatzeko ROS interfazea eskaintzen du. Munduko helburu bat emanda oinarri mugikorra bertara iritsi dadin saiatuko da. Goi-mailako ikuspegi batean, *move_base* nodoak planifikatzaile globala (*global planner*) eta planifikatzaile lokala (*local planner*) lotzen ditu bere eginkizuna burutzeko. Gainera, horretarako, bi kostu-mapa (*costmap*) ere baditu, bat planifikatzaile globalarentzat eta bestea lokalarentzat. Hauen arteko loturak eta beste elementu batzuekin dituztenak 7.2 irudian ikus daitezke. Urdinez dauden taulak gure robotaren arabera aldatuko dira, grisak hautazkoak dira (guk erabiliko ditugu), eta zuri kolorekoak, berriz, beharrezkoak.



7.2 Irudia: Nabigazio Pilaren antolaketa

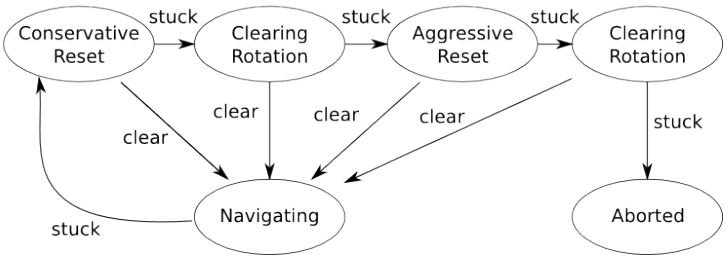
7.3.1 Berreskuratze-portaerak

move_base nodoari helburu bat bidaltzen diogunean, guztia ondo konfiguratuta badaukagu robota posizio horretaraino mugitzen saiatuko da. Bi aukera egon beharko lirateke (konfigurazio arazorik ez badago): robota helburura iritsiko da, edo hori ezinezkoa delako erabiltzaileari porrota adierazten duen mezu bat bidaliko dio. Helburura bidean robota lekuren batean trabatuta geratzen bada, berreskuratze-portaerak martxan jarriko dira berriro ere helburu-puntura iristen saia dadin.

Berreskuratze-portaera lehenetsiak lau dira (ikus 7.3 irudia). Robota trabatuta badago, lehenik eta behin, erabiltzaileak ezarritako eremu baten kanpoko oztopoak mapatik ezabatuko ditu. Jarraian, oraindik ere jarraitu ezin badu, robota dagoen lekuan biratzen hasiko

da oztoporik gabeko bide bat aurkitu asmoz. Honek ere ez badu konponbidea eman, hasieratze bortitzago bat egingo du. Azkenik, berriro ere porrot egin badu, dagoen lekuan beste bira bat emango du. Lau berreskuratze-portaera hauek huts egin badute, robotak amore emango du eta erabiltzaileari helburura iritsi ezin dela esango dio.

Azaldutako portaera hauek lehenetsiak dira, baina aldatu daitezke, eta baita desgaitu ere *recovery_behavior_enabled* parametroaren bidez.



7.3 Irudia: Berreskuratze-portaera lehenetsiak

7.3.2 Kostu-mapak

Kostu-mapak *costmap_2d* paketean daude inplementatuta. Pakete hau bi dimentsiotako kostu-maparen inplementazioa da, sentsoreetatik informazioa jasotzen du, bi dimentsiotako okupazio mapa eraiki eta kostuak puzten ditu okupazio informazioa eta erabiltzaileak zehaztutako puzte erradioaren arabera.

costmap_2d paketeak robotak non nabigatu beharko lukeen gordetzen duen egitura konfiguragarri bat eskaintzen du. Sentsoreetatik jasotako datuez nahiz mapa estatikoko informazioaz baliatzen da oztopoei buruzko informazioa gorde eta eguneratzeko.

7.3.3 Parametroen konfigurazioa

Jarraian, gure proiektuan nabigazioa martxan jarri aurretik egindako parametroen konfigurazioa aztertuko dugu.

Aurretik aipatu dugun moduan, nabigazio pilak bi kostu-mapa erabiltzen ditu munduko oztopoei buruzko informazioa gordetzeko: bat plan globala egiteko (ingurune guztian zehar epe-luzeko planak sortuz); eta, bestea, plan lokala egin eta oztopoak ekiditeko. Konfigurazio aukera batzuk bi kostu-mapek jarraitzea nahi dugu, eta beste batzuk, berriz, mapa bakoitzari esleituko zaizkio.

Bestalde, oinarri mugikorrarentzako abiadura aginduak sortuko dituen planifikatzaile lokalaren parametroak ditugu. Planifikatzaile honek planifikatzaile globalak sortutako plana erabiltzen du eta hau ahalik eta hobekien jarraitzen saiatzen da, robotaren kinematika eta dinamika nahiz oztopoen informazioa kontutan hartuta.

Nabigazioarekin zerikusia duen guztia gordetzeko *mari_2dnav* paketea sortu dugu, bai konfigurazio fitxategiak, bai launch fitxategiak bertan biltzeko.

Pakete horretan, beraz, bi direktorio aurki ditzakegu. Lehenengoa *config* izenekoa da eta ondorengo lerroetan azalduko ditugun lau konfigurazio fitxategiak topatu ditzakegu bertan. Bigarren karpeta, berriz, *launch* izenekoa da eta nabigazioa martxan jarriko duen *launch* fitxategia izango du barnean.

Bi kostu-mapen parametro komunak

Jarraian ikus daitezkeen sei parametroak bai kostu-mapa globalak, bai kostu-mapa lokalak erabiliko dituzte, eta *costmap_common_params.yaml* fitxategian aurki ditzakegu.

```
1 obstacle_range: 2.5
2 raytrace_range: 3.0
3
4 robot_radius: 0.27
5 inflation_radius: 0.55
6
7 observation_sources: laser_scan_sensor
8 laser_scan_sensor: {sensor_frame: laser, data_type: LaserScan, topic
    : /scan, marking: true, clearing: true}
```

Lehenengo bi parametroek oztopoak mapetan noiz ezarri eta kendu zehazten dute. *obstacle_range* parametroak sentsore irakurketek oztopoak hauteman eta kostu-mapan sartzeko distantzia maximoa definitzen du. Hemen, 2.5 metrotan ezarri dugu muga hori, eta ondorioz, robotak bere mapa robotetik 2.5 metrorainoko oztopoen informazioarekin bakarrik eguneratuko du. *raytrace_range*, berriz, oztopoa mapatik noiz kendu ezartzeko erabiltzen da; zehazki, mapatik kentzeko robotetik zein distantziara egon behar den definitzen du. Gure kasuan, robota 3 metrorainoko ingurua garbitzen saiatuko da laser irakurketen laguntzaz.

Jarraian, robotaren muga fisikoak ezarri dira. Marisorgin robot zirkularra denez, *robot_radius* parametroaren bidez bere erradioa zehaztu da (27 cm). Zirkularra ez balitz, robota mugatzen duen laukizuzena zein den esan beharko genuke *footprint* parametroaren bidez. *inflation_radius* parametroak, aldiz, oztopoen kostu balioak puzteko erradioa

definitzen du, metrotan. Hau da, robota oztopoetatik erradio hori baino urrutiago mantentzen saiatuko da uneoro.

Hurrengo bi parametroak laserrarekin zuzen-zuzenean lotuta daude. *observation_sources* kostu-mapari informazioa pasako dioten sentsoreez osatutako zerrenda bat da. Gure kasuan laser bat bakarrik daukagu eta azken lerroan laser honen ezaugarriak zehazten dira: sentsoreari dagokion *frame*-a, erabilitako datu mota, argitaratzen duen *topic*-aren izena, eta azkenik, sentsore honen bidez oztopoak kostu-mapetatik gehitu nahiz kendu egingo direla definitu dugu.

Kostu-mapa globala

Jarraian azalduko ditugun parametroak *global_costmap_params.yaml* fitxategian gorde dira, eta kostu-mapa globalaren funtzionamenduari eragiten diote.

```
1 global_costmap:
2   global_frame: /map
3   robot_base_frame: /base_footprint
4   update_frequency: 5.0
5   static_map: true
```

Lehenengo parametroen bidez ikus dezakegu kostu-mapa globalak */map frame*-a erabiltzen duela *frame* global moduan. Gainera robotaren oinarriaren *frame*-a zein den ere zehaztu da.

Kostu-mapa globala eguneratzeko maiztasuna *update_frequency* parametroaren bidez definitu dugu, zehazki 5 Hz, hau da, segundoro bost aldiz eguneratuko da.

Azkenik, mapa estatiko bat erabiliko dela zehaztu dugu. Mapa hau, aurretik aipatu dugun moduan, *map_server* paketearen laguntzaz lortuko dugu.

Kostu-mapa lokala

Ondorengo lerroak *local_costmap_params.yaml* fitxategian aurki ditzakegu eta kostu-mapa lokalaren funtzionamendua zehazten dute.

```
1 local_costmap:
2   global_frame: odom
3   robot_base_frame: /base_footprint
4   update_frequency: 5.0
```

```
  publish_frequency: 10.0
  static_map: false
  rolling_window: true
  width: 6.0
  height: 6.0
  resolution: 0.05
```

global_frame, *robot_base_frame*, *update_frequency* eta *static_map* parametroak aurreko zatian ikusi dugun konfigurazio globalean deskribatutakoarekin bat datoz.

publish_frequency parametroaren bidez kostu-mapak bistaratze informazioa zein maiztasunekin (Hz-etan) argitaratuko duen definitu dugu.

rolling_window parametroari *true* balioa emanda, kostu-mapa uneoro robotean zentratuta egotea lortuko dugu, nahiz eta robotaren kokapena ingurunean aldatu.

width, *height* eta *resolution* parametroek kostu-maparen zabalera (metrotan), altuera (metrotan) eta bereizmena (metro/gelaxketan) adierazten dute, hurrenez hurren. Orokorrean bereizmen hau bat etorri ohi da mapa estatikoaren bereizmenarekin, nahiz eta ez izan beharrezkoa.

Planifikatzaile lokala

base_local_planner paketea nabigaziorako abiadura aginduak sortzeaz (maila altuko plan batetik abiatuz) eta robotaren oinarri mugikorrari bidaltzeaz arduratzen da. Bere lana guk nahi dugun moduan burutu dezan, gure robotean oinarritzen diren beste konfigurazio aukera batzuk zehaztu behar ditugu. Horretarako, *base_local_planner_params.yaml* fitxategia sortu dugu eduki honekin:

```
controller_frequency: 5.0
planner_frequency: 1.0

TrajectoryPlannerROS:
  max_vel_x: 0.3
  min_vel_x: 0.1
  max_vel_theta: 0.6
  min_vel_theta: -0.6

  pdist_scale: 1.0

  yaw_goal_tolerance: 0.2
  xy_goal_tolerance: 0.2
  latch_xy_goal_tolerance: true

  holonomic_robot: false
```

controller_frequency parametroak kontrol begizta zein maiztasunekin exekutatuko den, eta ondorioz, abiadura aginduak robotaren oinarriari zein maiztasunekin bidaliko dizkion zehazten du.

Bigarren parametroa *planner_frequency* da eta planifikatzaile globalaren begiztaren exekuzio maiztasuna adierazten du.

Segidan, robotaren abiadura linealen eta angeluarren mugak definitu ditugu. X ardatzeko abiadura maximoa nahiko txikia ezarri diogu, fakultatean jendea ibili ohi delako eta ezustekoak ekiditeko asmoz. Azelerazio mugak, aldiz, lehenetsiak utzi ditugu, simulazioan portaera hobea duela ikusi baita.

pdist_scale parametroa kontrolagailuak jasotako bidea zein gertutasunez jarraitu beharko duen haztatzeko erabiltzen da. Guk balio lehenetsia (0.6) apur bat handitu dugu.

yaw_goal_tolerance, *xy_goal_tolerance* eta *latch_wx_goal_tolerance* parametroek robota helmugara zein zehaztasunez iritsi behar den definitzen dute, baimendutako tolerantzia zehaztuz.

Azkenik, *holonomic_robot* parametroa robota holonomoa den edo ez zehazteko da; gure kasuan, noski, *false* balioa eman diogu, ez baitugu Y ardatzeko abiadurarik jaso nahi.

7.3.4 Martxan jarri

Behin *move_base* paketeko elementuak eta aurrebaldintzak konfiguratuta, *launch* fitxategi bat sortuko dugu nabigazioa martxan jartzeko.

Fitxategi honen bidez hainbat nodo abiaraziko ditugu. Lehenik eta behin, aurretik aipatu dugun *mari_state_publisher2.launch* fitxategia exekutatuko dugu, bertako nodoak martxan jarri eta robotaren koordenatu-sistemen arteko aldaketak argitaratuta izan ditzagun.

Fitxategi honetan martxan jarritako lehenengo nodoa *map_server* izango da eta aurretik sortuta daukagun maparen informazioa emango digu, parametro gisa maparen fitxategia pasatuz.

Bigarren nodoa *amcl* da eta robotaren mapako kokapen probabilistikoa emango digu. Nodo hau martxan jartzeko bost parametro nagusi zehaztuko dizkiogu. Batetik, odometriaren erreferentzia-sistema zein den definitiko dugu *odom_frame_id* parametroan. Biga-

rrenik, *base_frame_id*, hau da, robotaren oinarriaren erreferentzia-sistema (gure kasuan *base_footprint*). Eta azkenik, robotaren hasierako kokapena eta orientazioa zeintzuk diren esango dioten hiru parametroak (*initial_pose_x*, *initial_pose_y* eta *initial_pose_a*). Gainera, *odom_alphaX* parametroek odometria estimazioetan esperotako zarata adierazten dute.

Orain guztia prest dago *move_base* nodoa abiarazteko. Lehenengo, gure ereduan erabilitako odometria eta abiadura mezuak argitaratuko diren *topic*-ak zeintzuk diren esan beharko diogu. Honez gain, nabigazioko parametroak gorde ditugun lau fitxategiak kargatu beharko ditugu.

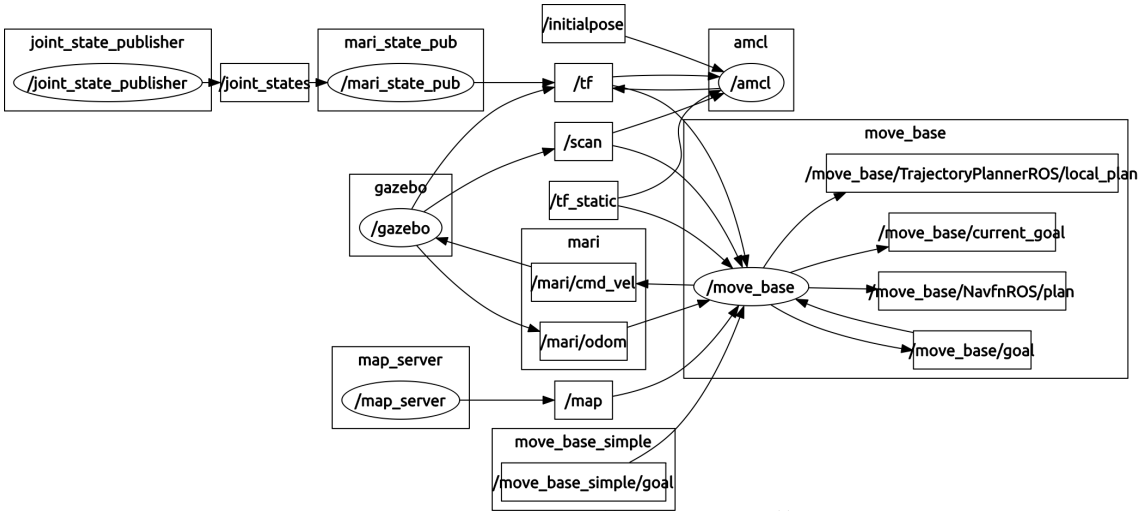
Bide batez, RViz programa ere martxan jarriko dugu.

Hona hemen *launch* fitxategia (*move_base_gazebo.launch*):

```
<launch>
  <master auto="start"/>

  <!-- Run transforms -->
  <include file="$(find mari_description)/launch/
      mari_state_publisher2.launch"/>

  <!-- Run the map server -->
  <node name="map_server" pkg="map_server" type="map_server" args="$(
      find map)/simulazioa/map.yaml"/>

  <!--- Run AMCL -->
  <node name="amcl" pkg="amcl" type="amcl" output="screen">
    <param name="odom_frame_id" value="/odom"/>
    <param name="base_frame_id" value="/base_footprint"/>
    <param name="initial_pose_x" value="10"/>
    <param name="initial_pose_y" value="13.0"/>
    <param name="initial_pose_a" value="0"/>

    <param name="odom_alpha1" value="0.1"/>
    <param name="odom_alpha2" value="0.1"/>
    <param name="odom_alpha3" value="0.1"/>
    <param name="odom_alpha4" value="0.1"/>
  </node>

  <node pkg="move_base" type="move_base" respawn="false" name="
      move_base" output="screen">
    <remap from="odom" to="/mari/odom" />
    <remap from="cmd_vel" to="/mari/cmd_vel" />
    <rosparam file="$(find mari_2dnav)/config/costmap_common_params.
        yaml" command="load" ns="global_costmap"/>
    <rosparam file="$(find mari_2dnav)/config/costmap_common_params.
        yaml" command="load" ns="local_costmap"/>
    <rosparam file="$(find mari_2dnav)/config/local_costmap_params.
```

```
      yaml" command="load" />
30    <rosparam file="$(find mari_2dnav)/config/global_costmap_params.
         yaml" command="load" />
31    <rosparam file="$(find mari_2dnav)/config/
         base_local_planner_params.yaml" command="load"/>
32   </node>
33
34   <!-- Run rviz -->
35   <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
        mari_description)/config/mari.rviz"/>
36
37 </launch>
```

7.4 irudian, nabigazioa martxan jarrita lortuko dugun konputazio-grafoa ikus dezakegu. Bertan, beraien artean komunikatzen ari diren nodo guztiak agertzen dira, eta baita elkar trukatutako *topic*-en izenak ere, argitaratzailea eta harpideduna zeintzuk diren adieraziz. Nodoak biribilduta agertzen direnak dira, eta lauki txikiak, berriz, *topic*-ak dira.



7.4 Irudia: Konputazio-grafoa (nabigazioa martxan jarrita)

7.4 Helburuak bidaltzeko programa

7.4.1 Sarrera

Proiektu honen muina Marisorgin robotari helburu bat bidali eta bera puntu horretara mugitzea da. Orain arte helbururainoko nabigazioarekin lotutako azalpenak ikusi ditugu; helburuaren esleipena, ordea, ez dugu aipatu.

Bi modu ezberdin erabili dira oztopoak zehazteko. Batetik, eta bietatik sinpleena, RViz tresnak eskaintzen duen "*2D Nav Goal*" aukera erabiltzea da, botoi horri sakatu eta mapan helburua eta orientazioa zehaztu besterik ez dugu egin behar. Proben atalean ikusiko dugu zehatzago honen funtzionamendua.

Bigarren modua *actionlib* paketearen bidez erabiltzaileak robota zein bulego edo laborategia joatea nahi duen zehaztean datza. Horretarako, bulego/laborategi zenbakia idatziko du komando lerro bidezko interfazea erabiliz. Atal honetan aukera honen ezaugarriak eta berau sortzeko urratsak deskribatuko ditugu.

Helburuak bidaltzeko programa honek abantailak eta desabantailak ditu.

Abantailen artean erabiltzeko erraztasuna eta erosotasuna daude, ez baita inongo tresna erabiltzeko ezagutzarik behar. Fakultateak bertako bulego eta ikerkuntza laborategi guztien telefono zenbakia eta gela zenbakia biltzen dituen zerrenda bat du atzigarri; eta gainera, ohikoa da irakasleek beren bulego zenbakia ematea interesdunei. Gainera, aurrerago ikusiko dugun moduan, programa honek bulegoen zerrenda ikusteko aukera ere eskaintzen du.

Desabantailak kontutan hartzen baditugu, berriz, zehaztasun falta aurki dezakegu; hau da, ezin dugu fakultateko hirugarren solairuko edozein txokora bidali Marisorgin, gela bakoitzeko aurrezarritako puntu bakarrera baizik. Korridoreetan ere ezingo diogu jomuga ezarri. Gainera, garapen prozesuan ez da erabilgarria, arazoak daudenean ez baitigu lagunduko hauen arrazoiak bilatzen. Kasu hauetan RViz erabiltzea egokiagoa da.

Seguruenik, erabiltzaileak bere jakintasun mailaren eta nahien arabera aukeratuko du RViz eta guk sortutako programaren artean.

7.4.2 Programaren ezaugarriak eta funtzionamendua

Helburuak bidaltzeko programa hau sortzeko ROS pakete berri bat garatu da, *simple_navigation_goals* izenekoa. Pakete honek nodo bakarra du barnean, izen bereko exekutagarria, hain zuzen ere. Nodo hau lortzeko *simple_navigation_goals.cpp* fitxategia sortu da C++ lengoaiaz programaren kodea idazteko. Aipatu dugun moduan, *actionlib* paketea erabiltzen da programa honen helburua lortzeko.

Fitxategi honez gain, *bulegoak.txt* testu fitxategia ere aurki dezakegu pakete honetan. Azken honek bulegoen zerrenda eta hauen kokapena gordetzen du. Zehazki lau zutabe ditu. Lehenengoan bulego/laborategi zenbakia ikus dezagu; bigarrenean eta hirugarrenean

helburuko X koordenatua eta Y koordenatua, hurrenez hurren; eta laugarren zutabeak, berriz, bulegoa edo laborategia zein irakasle edo ikerkuntza talderi dagokion gordetzen du.

Robotaren helburuko orientazioa beti berdina izango da: π radian.

Erabiltzailearen ikuspuntutik erabilera sinplea du programa honek. Exekutatzeko rosrun komandoa erabiliko dugu, segidan paketearen izena eta programaren izena idatziz. Honela:

```
# rosrun simple_navigation_goals simple_navigation_goals
```

Jarraian, arazorik egon ez bada, helburuko bulegoa eskatuko dio erabiltzaileari. Nahi izanez gero "*z*" tekla sakatuta bulegoen zerrenda inprimatuko da pantailan, eta hor ikus daiteke zein den nahi dugun irakasle/ikerlariaren gela zenbakia.

Zenbakia sakatutakoan, bulego hori existitzen bada Marisorgin bertara mugitzen hasi beharko litzateke. Bestela, mezu batek adieraziko digu bulego okerra idatzi dugula. Marisorginek ez badu arazorik izan helburura iristeko, momentu horretan mezu bat azalduko da pantailan robota adierazitako bulegoan dagoela jakinarazteko. Ezin izan bada iritsi ere (atea itxita bazegoen, adibidez), hala adieraziko digu beste mezu batek.

Programaren barrenak aztertzen baditugu, ikusiko dugu lau ataletan banatu dezakegula:

1. *move_base* nodoarekin kontaktuan jarri.
2. Aldagaiak hasieratu eta bulegoen fitxategia ireki.
3. Erabiltzailearekin elkarreragin.
4. Eskatutako helburua existitzen bada (bulego zenbakia zuzena bada, alegia), *move_base* nodoari pasa eta honek bere lana egin dezan itxaron.

Hurrengo atalean, hauetako atal bakoitza nola garatu den ikusiko dugu.

7.4.3 Programaren garapena

ROS-en oinarritutako edozein sistema handitan ohikoa da nodo bati eskaera bat bidali nahi izatea, honek betebehar bat burutu dezan, eta baita eskaeraren erantzuna jasotzea ere. Hau ROS zerbitzuen bidez lortu daiteke.

Batzuetan, ordea, zerbitzuak denbora asko behar badu exekutatzeko, erabiltzaileari eskaera deuseztatzeko gaitasuna izatea gustatuko litzaioke, edota prozesuaren egoerari buruzko informazioa jasotzea periodikoki. Hau *actionlib* paketea erabiliz lor dezakegu. Pakete honek, gure helburua beteko duen zerbitzaria, eta eskaerak egingo dituen bezeroa sortzeko tresnak eskaintzen ditu. Beraz, oso baliagarria izan zaigu helburuak bidaltzeko programa inplementatzeko orduan.

move_base nodoak zerbitzari hori eskaintzen du (*SimpleActionServer*-aren inplementazio bat), *geometry_msgs/PoseStamped* motako mezuen bidez helburuak jasoz. Zerbitzari honekin komunikatzeko bezeroa (*SimpleActionClient*) sortzea falta zaigu, eta hau da atal honetan egingo duguna.

Programa sortzeko, lehenik eta behin *simple_navigation_goals* paketea sortu dugu, eta jarraian, bulegoen informazioa biltzen duen fitxategia (*bulegoak.txt*) osatu.

Programa, aipatu dugun bezala, C++ lengoaia erabiliz idatzi da, eta nodoaren iturburukodea *simple_navigation_goals.cpp* fitxategian topa daiteke.

Kodea aurreko atalean zerrendatutako lau zatitan bana dezakegu.

Lehenengo zatian, *move_base* nodoaren zerbitzariarekin kontaktuan jarriko gara.

Jarraian, erabiliko diren aldagaiak definitu eta hasieratuko ditugu. Gainera, bulegoen zerrenda biltzen duen fitxategia irekiko dugu datuak irakurtzeko prest egoteko. Fitxategi hau erabilgarri ez badago, errore mezu bat inprimatu eta programa amaituko da.

Hirugarren zatian, erabiltzaileari bulego zenbaki bat idazteko eskatuko zaio. Beste aukera bat "*z*" tekla sakatzea izango da, bulegoen zerrenda pantailan inprimatzeko. Horregatik, irakurritako balioa "*z*" den begiratu beharko dugu lehenik eta behin. Hala bada, *bulegoak.txt* fitxategitik balioak irakurri eta bulego zenbakia eta irakasle/ikertzaile izenak erakutsiko zaizkio erabiltzaileari. Segidan, berriro ere bulego zenbakia idazteko eskatuko zaio.

Bai hasiera batean bulego zenbakia idatzita, bai zerrenda inprimatu ondoren, bulego hori Informatika Fakultateko hirugarren solairuan dagoen edo ez begiratuko da fitxategian. Existitzen bada, helburuko koordenatuak gorde eta *move_base* nodoari pasako zaio helburua, honek bere lana egin dezan. Robota bidalitako helburura iritsi bada, erabiltzaileari jakinaraziko zaio. Bestela, arazoren bat izan badu eta ezin izan bada iritsi ere (adibidez, bulegoko atea itxita baldin badago), mezu bat inprimatuko da pantailan.

Behin paketea eta kodearen fitxategia sortuta, konpilatu eta probatu egingo dugu. Horretarako, lehenengo egin beharrekoa *simple_navigation_goals.cpp* fitxategia paketeko

CmakeLists.txt fitxategian gehitzea izango da, exekutagarria sortu dadin. Hau egin ondoren, exekutagarria sortuko dugu *catkin_make* komandoa erabiliz. Orain, exekutatzeko prest dago. Nabigazio pila martxan badago eta exekutagarria ere abiarazten badugu (`rosrun simple_navigation_goals simple_navigation_goals`), bulego zenbaki bat idaztean ikusiko dugu Marisorgin helburu horretara bidean mugitzen hasiko dela.

7.5 Probak

Atal honetan, aurretik prestatu dugun nabigazioaren funtzionamendua ikusiko dugu Gazebo simulagailuan; eta RViz ikuskatzaileak eskaintzen dituen tresnen laguntzaz, gertatzen ari dena zehaztasun gehiagorekin aztertuko dugu.

7.5.1 Oztoporik gabe, RViz bidez helburuak zehaztuz

Proba hau egiteko bi *launch* fitxategi nagusiak abiaraziko ditugu: *mari_gazebo* paketeko *mari.launch*, eta *mari_2dnav* paketeko *move_base_gazebo.launch*. Honela guztia prest dugu helburu bat bidaltzeko, RViz ere irekita egongo baita.

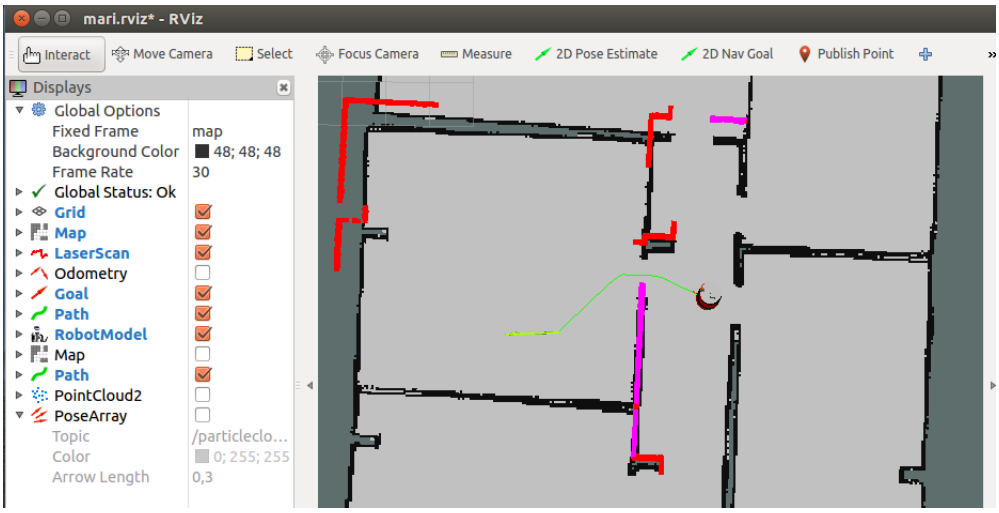
Jarraian, RViz-eko "*2D Nav Goal*" botoia sakatu eta Marisorginen helburua mapan klikatu dezakegu. Momentu horretan, automatikoki helbururako plana sortu eta Marisorgin bertara mugitzen hasiko da.

Nabigazioari buruzko zehaztasunak ikusi nahi baditugu RViz erabiltzea dugu hoberena. Adibidez zein bide jarraituko duen ikusiko dugu bertan (ikus 7.5 irudia).

7.5.2 Oztoporik gabe, sortutako programaren bidez helburuak zehaztuz

Bigarren proba honetan helburuak bidaltzeko programa erabiliko dugu. Aurretik exekutagarria sortu dugula kontutan hartuta, erraza da hau abiaraztea.

Lehenengo, beti egingo dugun moduan, Gazebo eta nabigazioa martxan jarriko ditugu bi *launch* fitxategiak erabiliz. Ondoren `rosrun simple_navigation_goal simple_navigation_goals` exekutatuz komando lerroan, programak helburuko bulegoa



7.5 Irudia: Nabigazioa RViz-en

eskatuko digu (ikus 7.6 irudia). Guk nahi dugun bulegoaren zenbakia idatzi, eta aurreko kasuan bezala automatikoki helbururako plana sortu eta Marisorgin bertara mugitzen hasiko da.

Oraingo honetan ere RViz oso erabilgarria izango zaigu nabigazioari buruzko zehaztasunak ikusteko.

```
oihane@ubuntu-laptop: ~ 162x41
oihane@ubuntu-laptop:~$ rosrun simple_navigation_goals simple_navigation_goals
[ INFO] [1434394901.426088329, 801.948000000]: Waiting for the move_base action server to come up

----- HELBURU BAT BIDALI MARISORGINI -----

Teklatua erabiliz aukeratu bulego edo laborategi bat (idatzi bulego zenbakia).
Bulego eta laborategiak zerrendatzea nahi baduzu, sakatu z
Idatzi bulego zenbakia: 305

Helburuko posizioa:(9.000000,12.600000), 305 bulego/laborategia (RSAIT).

[ INFO] [1434394903.389209851, 803.306000000]: Sending goal
[ INFO] [1434394952.677018944, 831.908000000]: Destination reached!

Marisorgin iritsi da 305 bulego/laborategira (RSAIT).

oihane@ubuntu-laptop:~$
```

7.6 Irudia: Nabigazioa helburua zehazteko programa erabiliz

7.5.3 Mapan agertzen ez diren oztopo estatikoekin

Nabigazioa ondo dabilela ikusi dugu ingurunean aldaketarik egon ez bada. Oraingo honetan, bizitza errealean ohikoa den moduan oztopoak sartuko ditugu ingurunean, Ga-

zebo simulagailua erabiliz. Oztopo hauek edozein unetan gehi ditzakegu, bai nabigazioa hasi aurretik, bai ondoren. Proba honetan gehituko ditugun oztopoak laserraren kokapena baino altuagoak dira, eta ondorioz, guztiak atzemango ditu Marisorginek. Bestalde, oztopo hauek estatikoak izango dira: ez dira mugituko nabigazioa martxan dagoen bitartean.

Korridoreko ate bat itxita

Planifikatzaile globalak helbururainoko kostu baxueneko bidea aukeratuko du beti, eta ondorioz, bestelako arazorik ez badago behintzat, bide motzena prestatuko du.

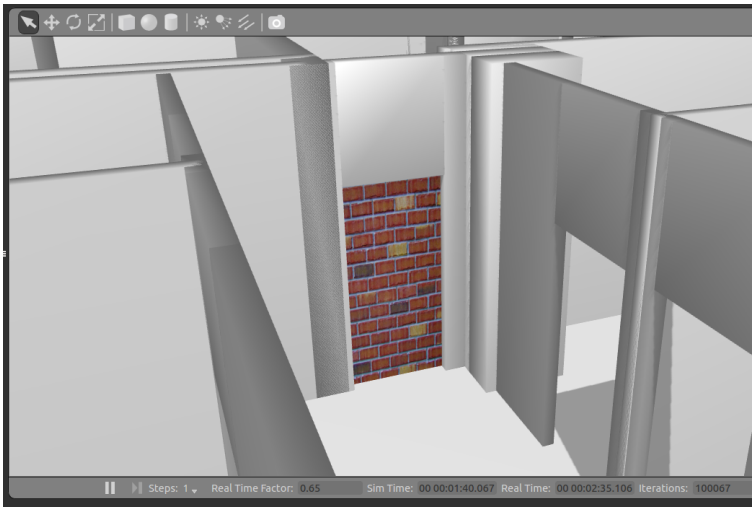
Proba honetan, hasiera batean prestatutako plana jarraituz gero zeharkatu beharko lukeen ate bat itxiko dugu. Ondorioz, plana aldatu eta bide luzeago bat hartu beharko du Marisorginek.

Gazebo martxan jarri (*mari.launch* bidez) eta korridoreko ate bat itxiko dugu, baina helbururainoko bidea baimentzen duen ate bat; hau da, beste nonbaitetik joateko aukera izango du robotak (ikus 7.7 irudia).

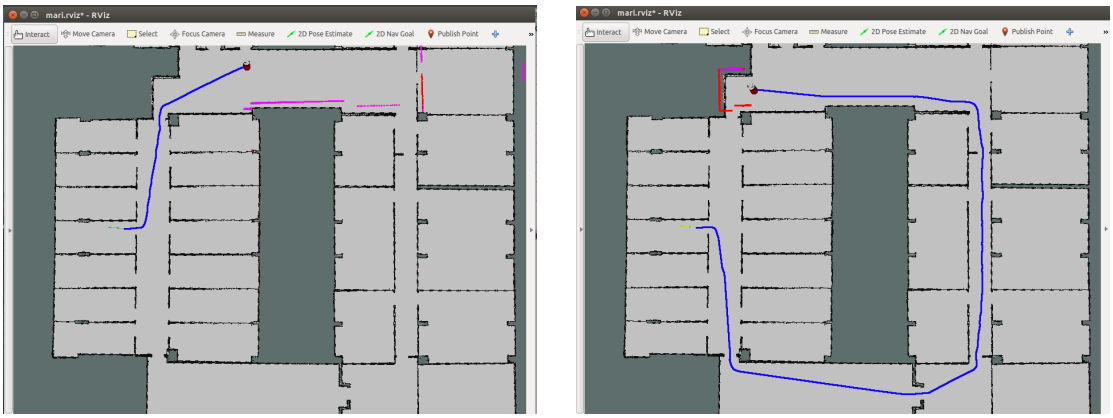
Robota oztopotik (atetik) 2,5 metrora dagoenean, *obstacle_range* parametroarekin adierazi diogunez, oztopoa kostu-mapan sartuko du, eta une horretan, konturatuko da ezin duela hasierako plana jarraitu eta berria sortuko du.

Modu honetan, plan berria jarraitzeko ez du arazorik izango eta guk adierazitako helburura iritsiko da.

7.8a irudian hasierako plana ikusi dezakegu, eta 7.8b irudian, aldiz, plan berria.



7.7 Irudia: Atea itxita gazebon



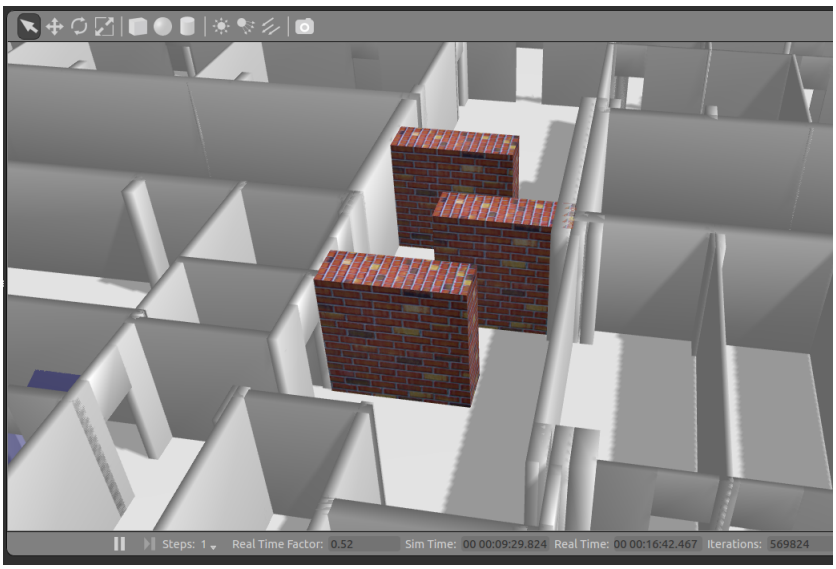
(a) Hasierako plana **(b)** Plan berria

7.8 Irudia: Helbururako plana (oztopo estatiko batekin)

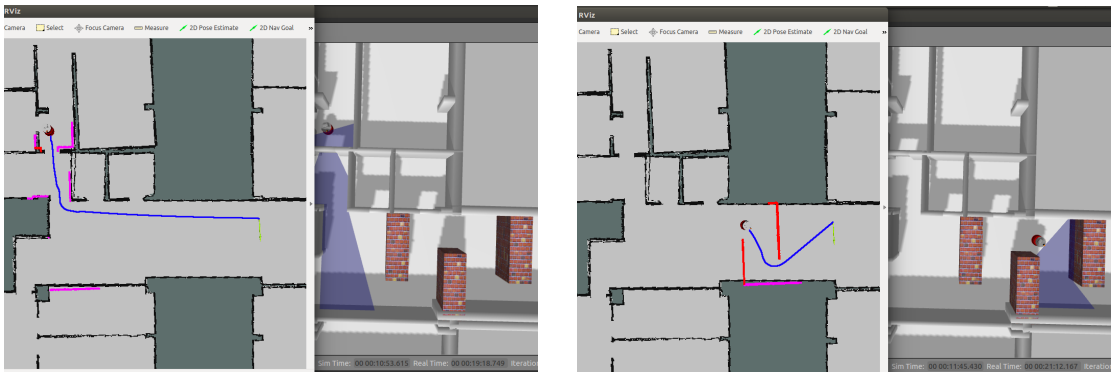
Oztopo multzoa (labirintoa)

Oztopo estatikodun bigarren proba honetan oztopo bat baino gehiago jarriko dizkiogu bidean. Labirinto moduko bat sortuko dugu eta ikusiko dugu arazorik gabe zeharkatuko duela korridorea (ikus 7.9 irudia).

Kasu honetan, plana pixkanaka eguneratzen joango da, oztopoak ikusten dituen heinean (7.10a eta 7.10b irudiak).



7.9 Irudia: Oztopo multzoa Gazebo-n



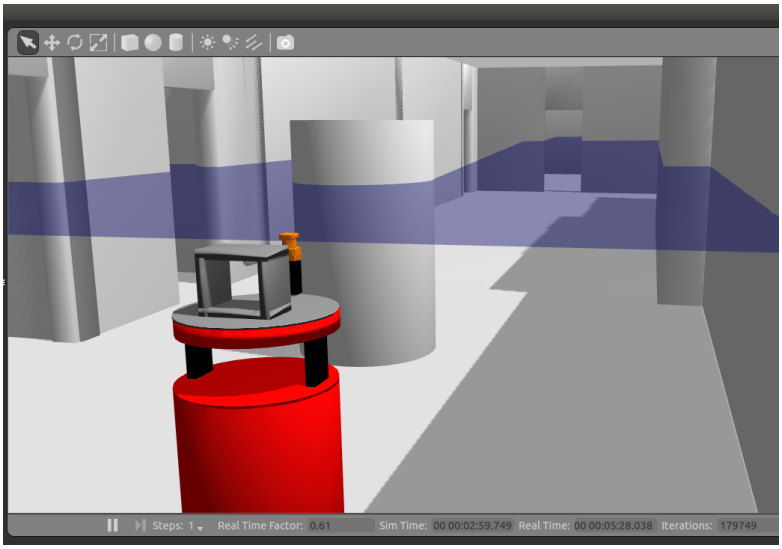
(a) Hasierako plana **(b)** Plan berria

7.10 Irudia: Helbururako plana (oztopo estatiko multzoarekin)

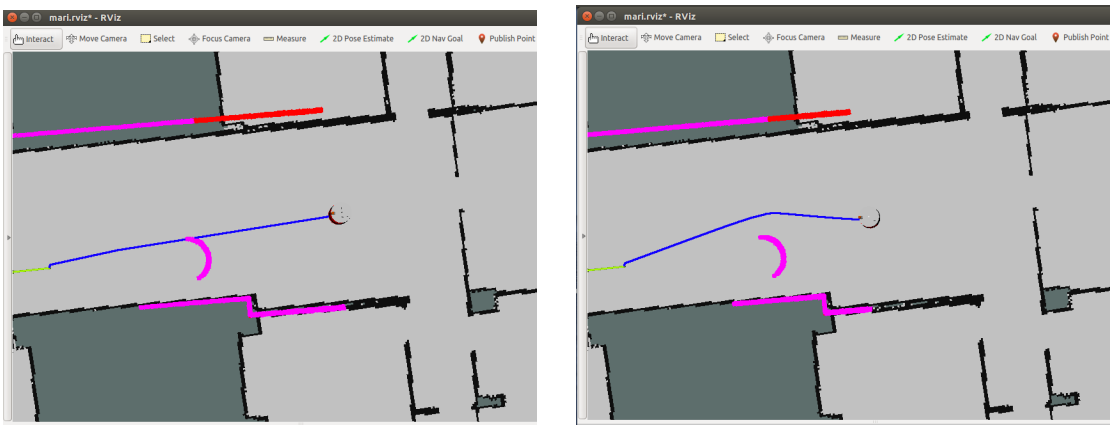
7.5.4 Oztopo dinamikoekin

Informatika Fakultatean gerta daitekeen arrisku handiena robota pertsona batekin gurutzatu eta berarekin talka egitea da. Horregatik, Gazebo-n egoera hau simulatzen saiatuko gara. Helburu hau lortzeko oztopo bat lekuz aldatuko dugu periodikoki, pertsona baten mugimendua simulatuz. Oraingo honetan ere nabigazio pilak plana eguneratuko du oztopoarekin topatzen denean; kasu honetan, plana behin eta berriz eguneratuko du plan berriaren bidean oztopo bat jartzen bada.

Egoera hau 7.11 irudian ikus daiteke, eta 7.12a eta 7.12b irudietan, berriz, plana nola eguneratzen duen.



7.11 Irudia: Oztopo dinamikoa Gazebo-n



(a) Hasierako plana **(b)** Plan berria

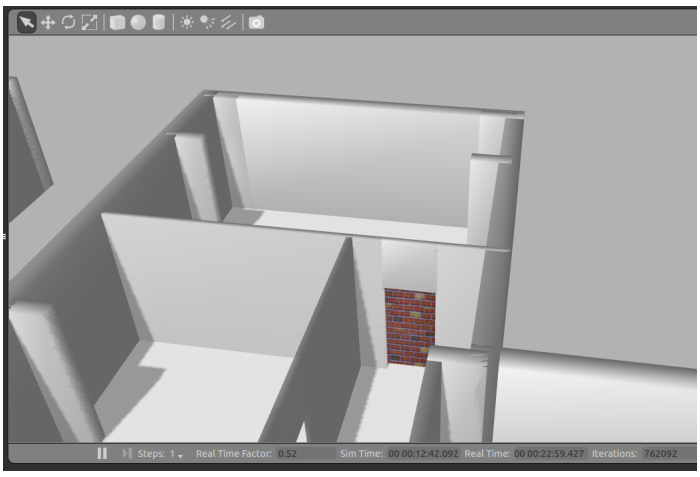
7.12 Irudia: Helbururako plana (oztopo dinamikoekin)

7.5.5 Iritsi ezin daitekeen helburu bat

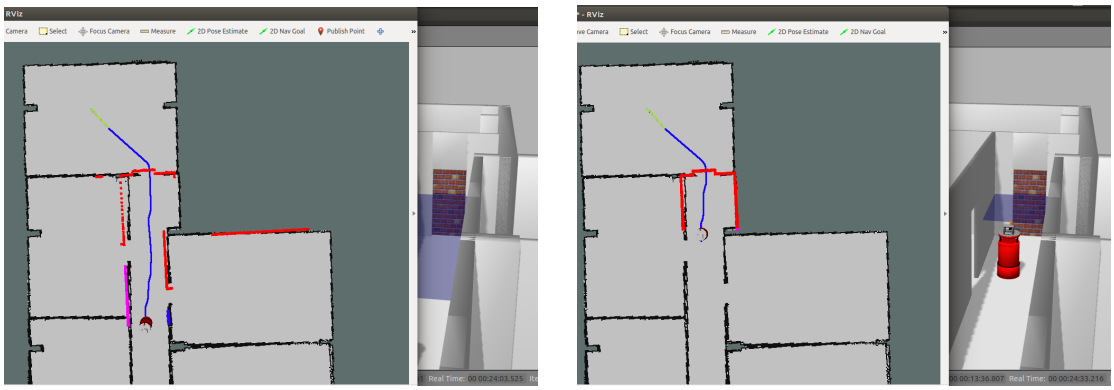
Orain arteko helburu guztiak atzigarriak izan dira, nahiz eta plana aldatu behar izan. Kasu honetan, ordea, helburuko bulegoaren atea itxiko diogu. Egoera honetan, noski, robota ezingo da gelara sartu (ikus 7.13 eta 7.14 irudiak).

Amore eman aurretik, berreskuratze-portaera guztiak jarraituko ditu eta amaieran errore mezu bat ikusiko dugu plan egokirik aurkitu ezin dela adieraziz:

```
[ERROR] [1434042708.354356123, 543.610000000]: Aborting because a valid
plan could not be found. Even after executing all recovery behaviors
```



7.13 Irudia: Helburuko bulegoko atea itxita Gazebo-n

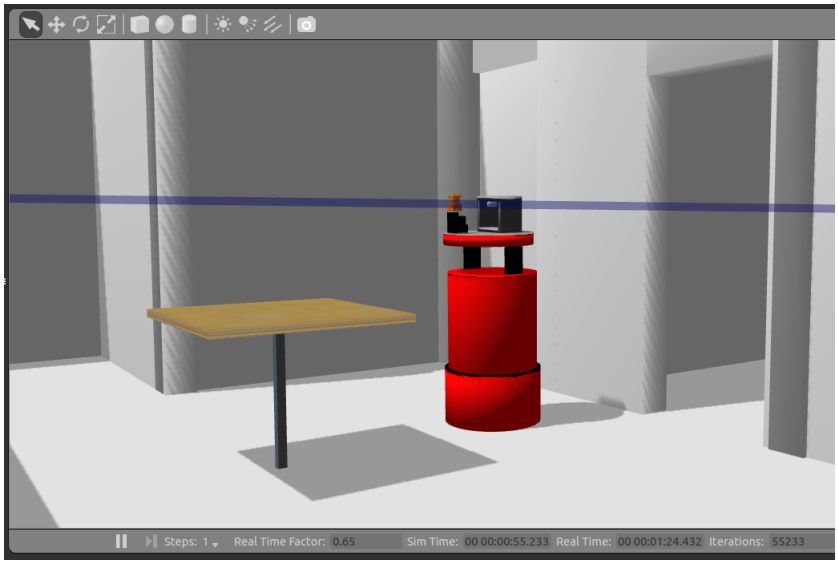


(a) Hasierako plana **(b)** Nabigazio amaiera

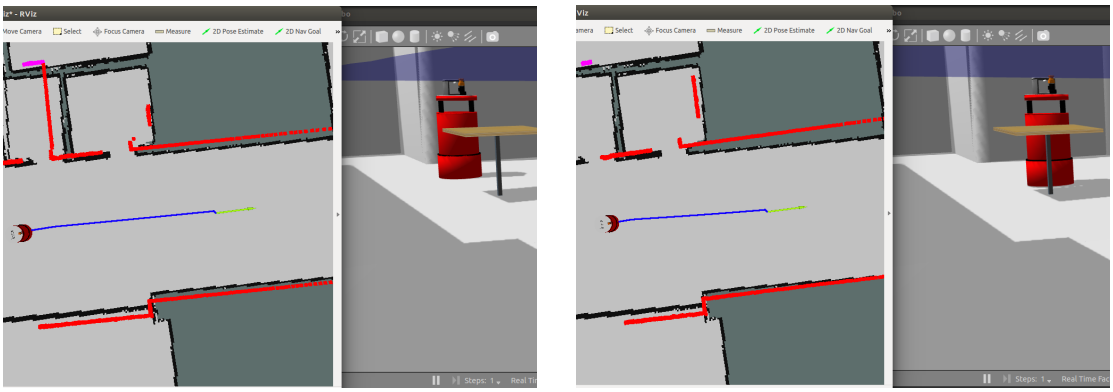
7.14 Irudia: Helbururako plana (iristea ekiditen dion oztopo batekin)

7.5.6 Ikusten ez dituen oztopoak

Marisorginen nabigazioaren arazo handienetariko bat laserra oso altu kokatuta dagoela da, 1,25 metro inguruko altueran. Hori dela eta, altuera hori baino baxuagoak diren oztopoak ez ditu ikusten eta egongo ez balira bezala jokatzen du. Hau da, bidean oztopo "baxu" bat badago, berarekin talka egingo du. 7.15, 7.16a, 7.16b irudi sekuentziak erakusten du egoera hau.



7.15 Irudia: Oztopo baxu bat Gazebo-n



(a) Hasierako plana **(b)** Oztopoarekin talka

7.16 Irudia: Helbururako plana (oztopo baxu batekin)

8. KAPITULUA

Marisorgin robot errealaren nabigazioa

Aurkibidea

8.1 Sarrera

Proiektu honen helburu nagusia Marisorgin robotaren nabigazioa Gazebo-ROS inguruanean prestatzea izan da. Simulazioan robotak nabigazio egokia egiten duela ikusi dugu, baina simulazioa eta errealitatea ez dira guztiz berdinak, eta ondorioz, emaitzak ere desberdinak izan ohi dira. Hori dela eta, interesgarria da simulatu duguna mundu errealean ikustea, azken finean, Marisorgin mundu errealean mugituko baita. Honetarako, Informatika Fakultateko hirugarren solairuan bertan Marisorgin mugiaraziko dugu.

Nabigatzen hasi aurretik aldaketa batzuk egin beharko ditugu gure sisteman: mapa berria sortu, robotaren *driver*-ak argitaratutako *topic*-etara moldatu eta nabigazio parametroak aldatu. Aldaketa hauek guztiak hurrengo puntuan azalduko ditugu.

Hauek aldatu behar izateaz gain, ordea, arazo nagusia beste bat izan da proiektuaren fase honetan: *driver*-ak odometriako informazioa unitate okerretan argitaratzen zuen. Honen inguruko azalpenak ere emango ditugu 8.2.3 atalean.

8.2 Aldaketak

8.2.1 Mapa berria sortu

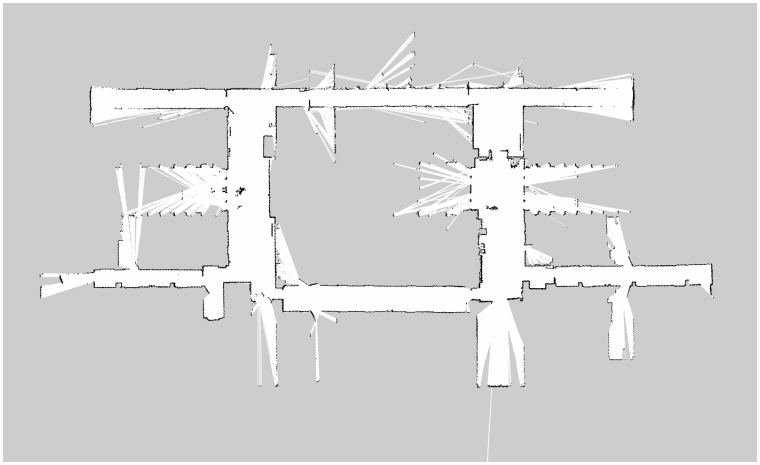
Hirugarren solairuan diharduten ikertzaileak ez gogaitzeko asmoz, nabigazioa korridoreetan bakarrik egingo da. Ondorioz, batetik, ez dugu erabiliko helburuak bidaltzeko sortu dugun programa (bulego eta laborategiak ez direlako erabiliko); eta, bestetik, mapa berria sortuko dugu mundu errealarekin ahalik eta hobekien bat etor dadin. Hala ere, aipatu behar da Gazebo-n sortutako mapa baliozkoa suertatu dela robot errealaren nabigaziorako.

Mapa hau sortzeko, simulazioan emandako pauso berdinak jarraitu behar dira. Marisorgin errealaren eta simulatutakoaren arteko desberdintasunen artean, orain Gazebo ordez dispositibo errealen, hots, mugimendu-sistemaren eta laserraren, *driver*-ak abiarazi behar ditugula dago batetik; eta bigarrenik, *joystick*-a erabiliko dugula teklatuaren ordez, robota solairuan zehar gidatzeko (mapa sortzerakoan bakarrik). Hortaz, bi horiek martxan jarri eta *gmapping* eta *map_server* paketeak erabiliko ditugu simulagailuan bezala. Ikusi 7.2.6 atala zehaztasun gehiago jakiteko.

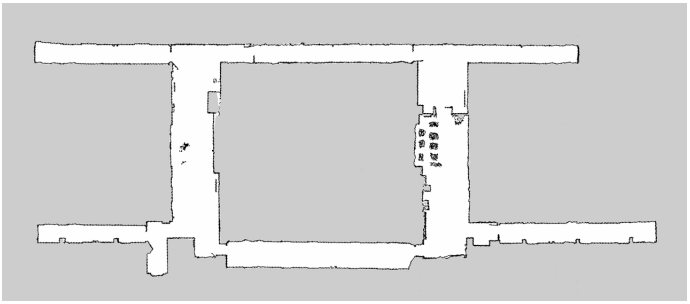
Behin mapa gordeta, apur bat editatu beharko dugu, Marisorgin bertara joatea nahi

ez dugun espazioa mugatuz; adibidez, eskailerak dauden guneak. Gainera, laborategi batzuek kristala daukate korridore aldean, eta laserra kokatuta dagoen altuera dela eta, espazio libre moduan ikusten ditu leku horiek. Ate irekiek edo korridoreetako leihoek ere espazio libre okerra irudikatuko dute. Irudien editore baten bidez beltzez oztopoak zehazten dira (mapan irudikatu ez direnak), zuriz espazio libreak (oker irudikatu direnak ezabatzea izango da egokiena) eta gris kolorez eremu ezezagunak. Jatorrizko mapa eta moldaketak egin ondoren lortutako maparen arteko aldeak 8.1 eta 8.2 irudietan ikus daitezke.

Horrez gain, beste arazo bat ere izan genezake fakultatean. Korridore batean robotak ikusten ez dituen bost aulki eta bi mahai daude; hau da, oztopo baxuak. Bertara joatea ekidin nahi badugu, mapan beltzez irudikatu dezakegu oztopo hauen kokapena.



8.1 Irudia: Nabigaziorako ingurune errealaren jatorrizko mapa



8.2 Irudia: Nabigaziorako ingurune errealaren mapa moldatua (erabiliko dena)

8.2.2 *Launch* fitxategi berria

Gazebo-n odometria eta abiadura mezuak argitaratu eta harpidetzeko erabilitako *topic*-ak */mari/odom* eta */mari/cmd_vel* izan dira, hurrenez hurren. Marisorginen *driver*-ak, berriz, */mari_driver/odom* eta */mari_driver/cmd_vel* erabiltzen ditu.

Baterаezintasun hau konpontzeko aldaketa txiki bat egitearekin nahikoa da: *move_base_gazebo.launch* fitxategiko bi *topic* horiek aldatu eta *move_base.launch* sortuko dugu, gainontzeko edukia berdin mantenduz.

Hona hemen aldatutako bi lerro horiek:

```
1 <remap from="odom" to="/mari_driver/odom" />
2 <remap from="cmd_vel" to="/mari_driver/cmd_vel" />
```

Bestalde, ingurune errealean erabiliko den mapa ere desberdina izango da. Arazo honi aurre egiteko *map_server* nodoa abiarazteko lerroa aldatuko dugu:

```
1 <node name="map_server" pkg="map_server" type="map_server" args="$(
    find map)/errealitatea/map.yaml"/>
```

Hala ere, esan Gazebo-n sortutako mapa erabiliz ere nabigazio egokia lortzen dela.

8.2.3 Odometria

Marisorgin errealaren nabigazioa martxan jarritakoan, parametroak nahiz eta behin eta berriz aldatu, bide lokalak globala jarraitzen ez zuela konturatu ginen, eta ondorioz, robotak ez zuen guk esperotako portaera.

Planifikatzaile lokalak hiru sarrera elementu erabiltzen ditu bere lana egiteko: planifikatzaile globalak sortutako bidea, kostu-mapa lokala (eta hau sortzeko erabiltzen den laserra), eta odometriako informazioa.

Plan globala ondo sortzen zela ikusten genuen RViz-en, beraz, hau ez zen arazoaren eragilea. Jarraian, laserraren detekzioak aztertu genituen: RViz-en ikusi genuen irakurketak bat zetozela maparekin, eta kostu-mapa lokalak itxura ona zeukan, baina badaezpada ere, irakurketa balio arrarorik zegoen begiratu genuen. Hau ere zuzen zebilela zirudien.

Azkenik, arazoa topatu genuen. Marisorginen *driver*-ak argitaratzen zituen */mari_driver/odom topic*-eko mezuek unitate desegokiak zituzten. Zehazki, abiadura linea-

la cm/s-tan eta angeluarra gradu/s-tan argitaratzen zituen eta nabigazioak m/s eta rad/s unitateak behar zituen, hurrenez hurren.

Hau zuzendu ostean, guztiz aldatu zen Marisorginen portaera eta hurrengo atalean azalduko ditugun nabigazio parametroak findu besterik ez genuen egin behar izan.

8.2.4 Nabigazio parametroak

Odometriaren arazoa konpondu ostean, simulagailurako prestatutako nabigazio sistemaren parametro gutxi batzuk aldatu dira nabigazioa ahalik eta txukunena izateko helburuarekin. Parametroen esanahi zehatzaz oroitzeko, begiratu 7.3.3 atala.

- Puzte erradioa apur bat handitu da oztopoetara gutxiago gerturatzeko.
  $inflation\_radius$: 0.60
- Kontrolagailuaren maiztasuna handitu egin da eta balio lehenetsia utzi da.
  $controller\_frequency$: 20.0
- Biraketa abiadurak moteldu dira.
  max_vel_theta: 0.3
  min_vel_theta: -0.3
  $min\_in\_place\_vel\_theta$: 0.3
- $pdist\_scale$ parametroaren balio lehenetsia utzi da: 0.6

8.3 Probak

Simulagailuan nabigazio probak egin ditugunean, zuzenean Marisorgin bere simulagailuko kokapenean ikusten dugu RViz-en, hau horrela konfiguratu baitugu *amcl* nodoa martxan jartzean. Oraingo honetan, aldiz, Marisorginen hasierako posizioa ez da beti berdina izango; hortaz, RViz irekitzerakoan "*2D Pose Estimate*" botoia sakatu, eta mapan robotaren kokapena eta orientazioa adieraziko ditugu.

Orain bai, prest gaude nahi dugun helburua bidaltzeko robotari. Jarraian ingurunearen kasu ezberdinak aztertuko ditugu, eta konturatuko gara Gazebo-n erakutsitako portaera berdina duela nabigazioak.

8.3.1 Oztoporik gabe

Lehenengo kasu hau sinpleena da. Hirugarren solairuan inor ez dabilen une batean, Marisorgin puntu batetik beste batera bidaliko dugu, mapa sortutakoan zeuden oztopoen berri bakarrik jakinda. Ondorioz, hasieran sortutako plana jarraituko du amaieraraino. Prozesu hau 8.3 irudian ikus daiteke.

(a) Hasierako posizioa **(b)** Nabigazio hasiera

(c) Bidearen jarraipena **(d)** Nabigazioaren amaiera

8.3 Irudia: Oztoporik gabeko nabigazioa

8.3.2 Mapan agertzen ez diren oztopo estatikoekin

Bigarren proba honetan, oztopo berriak sartuko ditugu ingurunean. Batetik, pertsonak erabiliko ditugu oztopo gisa, eta bestetik ateak itxiko ditugu.

Pertsonak geldi

Robota hasierako posizioan dagoela, laserrak ikusten ez dituen lekuetan bi pertsona jarriko ditugu geldi, ondo pentsatutako posizioetan. Ondoren, helburua zehaztuko diogu, hasiera batean sortutako plana pertsona horiek dauden lekutik pasako dela ziurtatuz. Konturatuko gara oztopoak 2,5 metrora dituenean plana aldatu eta hauek saihestuko dituela.

Atea itxita

Oraingo honetan Gazebo-n egin genuen proba berdina egingo dugu: hasiera batean prestatutako plana jarraituz gero zeharkatu beharko lukeen ate bat itxiko dugu. Ondorioz, plana aldatu eta bide luzeago bat hartu beharko du.

Robota oztopotik (atetik) 2,5 metrora dagoenean, oztopoa kostu-mapan sartuko du eta une horretan konturatuko da ezin duela hasierako plana jarraitu eta berria sortuko du. Modu honetan, plan berria jarraitzeko ez du arazorik izango eta guk adierazitako helburura iritsiko da.

8.3.3 Oztopo dinamikoekin

Proba hau inguruneko ohiko egoera bat da: pertsonak korridoreetan zehar mugitzea. Fakultatean gerta litekeen arrisku handiena robota pertsona batekin gurutzatu eta berarekin talka egitea da. Horregatik, portaera egokia duela aztertu beharko dugu. Robota mugimenduan dagoela bere aurretik gurutzatzen bagara, ikusiko dugu kostu-mapa lokala nola eguneratzen den oztopoa mugitzen den heinean. Normalean, oso azkar mugitzen bagara, ez du plana aldatu ere egingo, eta bestela, behin eta berriz aldatuko du, beti kostu txikieneko bidea aukeratzen saiatuko baita.

8.4 irudiak Marisorginek oztopo dinamiko batekin duen portaera erakusten du.

(a) Hasierako posizioa **(b)** Oztopo dinamikoa

(c) Oztopoa saihesten **(d)** Oztopoa gainditu ondoren

8.4 Irudia: Nabigazioa oztopo dinamiko batekin

8.3.4 Ikusten ez dituen oztopoekin

Azken proba honetan laserraren altuera baino baxuagoak diren bi oztopo jarri ditugu Marisorginen ibilbidean. Oztopo hauek pisu gutxiko bi kaxa izan dira, kalteturik ez egoteko helburuarekin.

Errealitatean robotak simulagailuan duen portaera berdina erakutsi du kasu honetan ere: oztopoekin talka egin eta hauek bultzatu egiten ditu (ikus 8.5 irudia).

(a) Hasierako posizioa **(b)** Nabigazio hasiera

(c) Oztopoetatik gertu **(d)** Talka

8.5 Irudia: Nabigazioa oztopo baxuekin

8.3.5 Simulagailuan sortutako maparekin

Simulagailuan sortutako mapa ez dator guztiz bat benetako neurriekin, baina oso antzekoa da.

Mapa hau erabiltzearen abantaila nagusia, Marisorgin hirugarren solairuko edozein bulegora bidal dezakegula da. Bestela, fakultatean mapa sortzerakoan bulegoetan banan-banan sartu beharko ginateke mapa osatu arte.

Gazebo-n sortutako mapa honekin robotak portaera berdina duela frogatu da, eta bai laborategi batera, bai bulego batera inongo arazorik gabe sartu da Marisorgin.

9. KAPITULUA

Ondorioak eta etorkizuneko lana

Aurkibidea

Azken kapitulu honetan proiektuaren garapenean zehar ondorioztatutako puntu ezberdinak zerrendatuko dira. Honez gain, proiektu honek izan ditzakeen hobekuntzak, edo etorkizunean egiteko interesgarriak izan daitezkeen lanak azalduko dira.

9.1 Ondorioak

Proiektuaren ikuspuntu orokor bat hartuta, hasieran ezarritako helburu guztiak bete direla esan dezakegu. Garatutako ROS paketeei esker, Marisorgin robotaren nabigazioa Gazebo-ROS ingurunean martxan jarri dugu. Gainera, hobekuntza batzuk ere burutu dira; adibidez, nabigazioak robot errealean izan duen funtzionamendu zuzena ikusi ahal izan dugu.

Jarraian, proiektuaren garapenean zehar ateratako hainbat ondorio aztertuko dira.

1. Proiektua simulagailuan eta errealitatean garatzearen ondorioz, argi geratu dira simulagailu bat erabiltzearen abantailak. Gure kasuan, batez ere, robot erreala martxan jartzeko denbora saihestu dugu egindako proba kopuru handia kontutan hartuta. Gainera, fakultatean bertan gerta litezkeen arriskuak ekidin dira simulagailuaren bidez. Lana ere errazagoa da ordenagailu parean, uneoro Marisorginen atzetik ibili ordez.

2. Simulazioa eta errealitatearen artean beti desberdintasunak egongo dira. Hala ere, ahalik eta berdinenak izaten saiatuko gara ondoren emaitza antzekoak lortzeko. Gazebo simulagailuak aukera asko eskaintzen ditu, baina lan asko eskatzen du mundu erreala zehaztasunez simulatzeak, batez ere eremu zabal batean lan egin behar badugu.

3. Simulagailuan emaitza zuzenak lortu ondoren, errealitatean berriro ere beharrezko proba guztiak burutuko dira, seguruenik portaera desberdinak lortuko baitira.

4. Maila honetako proiektu bat garatzeko ROS-en erabileraren oinarrizko ezagutza beharrezkoa da. Proiektuarekin lanean hasi aurretik ezagutza hau lortu beharko dugu, alperrik denbora galdu nahi ez badugu.

5. Batzuetan, kontutan izan behar da balitekeela arazoa ez egotea guk uste dugun lekuan, edo guk egindako lanean; itsu-itsuan guk egindakoari begira egon ordez. Hau da hasiera batean gertatu zitzaiguna odometriaren arazoaz jabetu aurretik.

Bestalde, ondorio teknikoak alde batera utzita, aipatu laguntzarik gabe ia ezinezkoa litzatekeela errealitatean lortu den portaera egokia ikustea. Asko eskertzen da RSAIT taldeko kideek eskainitako laguntza hori.

9.2 Etorkizunerako lana

Gradu Amaierako Proiektua amaitu dugun arren, honi jarraipena eman diezaioketen lan dezente egin daitezke, batez ere, sistemaren portaera findu eta erabilpena zabaltzeko. Hona hemen proposatutako hobekuntza posible batzuk:

1. Euskal Herriko Unibertsitatearen Informatika Fakultate osoa sartu ingurunean. Honetarako, simulagailuan gainontzeko hiru solairuak (0, 1 eta 2) eta solairuartea irudikatuko genituzke. Robotak solairu batetik bestera mugitzeko igogailuak erabiliko lituzke. Igogailuak kontrolatzeko plugin bat jadanik badu Gazebo-k.

2. Geletarako sarbidea benetakoarekin bat etortzeko, itxi eta irekitzen diren ateak jarri ingurunean. Honela, nahi ditugunak bakarrik ireki edo itxi ditzakegu, horretarako prestatutako tresna bat erabiliz.

3. Ingurunea ahalik eta errealena izateko asmoz, oztopo guztiak jarri daitezke simulagailuan. Gainera, neurriak planotik neurtu ordez, fakultatean bertan hartu daitezke, zehaztasuna handitzeko.

4. Simulagailuan abiadura aginduak jasotzerakoan robot osoa mugitzen da. Errealitatean, aldiz, gurpilak gorputzarekin batera mugitzen dira, eta oinarriak orientazio berdina mantentzen du beti. Robotak simulagailuan errealitateko portaera berdina izan dezan konfigura genezake. Hala ere, Marisorginen oinarria fisikoki zirkularra denez, ez genuke aldaketa garrantzitsurik ikusiko. Bestalde, URDF ereduan ez dago gurpilik eta Gazebo-ko plugin-ak oinarria mugitzen du gurpilen ordez. Ahalik eta portaera errealistena lortzeko, gurpilak gehitu eta plugin berri bat erabili beharko genuke.

5. Robotaren URDF ereduan ez dira robotaren benetako ezaugarri fisikoak islatzen; hala nola, pisua edo inertzia. Ezaugarri hauek benetako datuekin ordezkatzen baditugu, askoz portaera errealistagoa izango genuke.

6. Laserra, askotan aipatu dugun moduan, oso altu dago kokatuta robotean, eta ondorioz, altuera hori baino baxuagoak diren oztopoak ez ditu atzematen. Beherago mugitzea izango litzateke konponbide onena, baina horretarako, robotaren egiturak aldaketa morfologiko sakonak jasan beharko lituzke, oraintxe apur bat jaisteko lekua besterik ez baitu. RSAIT taldea jadanik hasi da honen konponbideak aztertzen.

7. Marisorgin erreala gidatzeko erabiltzen den *joystick*-arentzako kokalekua oinarriaren eremutik kanpo dago, eta beraz, posible da honek oztopoekin talka egitea. Bere kokapena enbarazu egiten ez duen beste leku batean jarriz gero, ez genuke arazo hau izango.

8. Bidalitako helburua bertan behera uzteko agindua modu sinple batean emateko prestatu; hau da, ROS *framework*-a ulertzen ez duen erabiltzaile batek egiteko moduan.

9. GUI bat garatu, erabiltzailearentzat errazagoa izan dadin helburuak bidaltzea.

Eranskinak

A. ERANSKINA

Erabilpen gida

Aurkibidea

A.1 Sarrera

Eskuliburu honen bidez Marisorgin robotaren nabigazioa martxan jartzeko eman beharreko pausoak azalduko dira.

Marisorgin robotaren nabigazioa bi ingurugirotan egin da. Batetik, Gazebo simulagailua erabiliz; eta bestetik, RSAIT taldearen benetako Marisorgin robota UPV/EHUko Informatika Fakultateko hirugarren solairuan mugituz.

Ondorioz, nabigazioaren erabilpen gida hau bi zati nagusitan banatzen da: simulazioa eta errealitatea. Bietako bakoitzean martxan jarri aurretik egin beharrekoa azaldu da, eta ondoren, nabigazioa burutzeko jarraitu beharreko pauso bakoitza zerrendatu da.

Eskuliburu honetan ez da ROS, Gazebo eta RViz erabiltzeko oinarrizko azalpenik emango. Zalantzaren bat izanez gero, beren webgune ofizialetara jotzea gomendatzen da.

A.2 Simulazioa

A.2.1 Aurrekariak

Instalazioa

Nabigazioa martxan jarri aurretik, ROS eta Gazebo ordenagailuan instalatuta ditugula ziurtatu beharko dugu. Hala ez bada, proiektu hau abiarazteko gomendatutako bertsioa *ROS Indigo* da eta konfigurazioa *Desktop-Full* izenekoa. Horretarako, komando hau exekuta dezakegu:

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

Konfigurazio hori erabiliz ROS-ez gain, RViz, Gazebo eta nabigazioa ere instalatuko dira, beste ROS pakete batzuen artean. Bestela, beharrezko ROS paketeak banaka instala ditzakegu.

Sortutako paketeak kopiatu eta konpilatu

Proiektu honetan sei ROS pakete eta haien fitxategiak sortu edo editatu dira: *mari_description*, *mari_gazebo*, *gazebo_plugins*, *map*, *mari_2dnav* eta *sim-*

ple_navigation_goals. Beraz, pakete guzti hauek *catkin workspace*-ean kopiatu eta *catkin_make* erabiliz konpilatuko ditugu.

Bestalde, Informatika Fakultateko hirugarren solairuaren eredua kargatu ahal izateko, Gazebo-ri eredu honen kokapena non dagoen esango diogu. Horretarako, *GAZEBO_MODEL_PATH* aldagaiari ereduaren bide absolutua emango zaio:

```
# export GAZEBO_MODEL_PATH=$HOME/catkin_ws/src/mari_gazebo/models
```

Hau hasieratze script batean gehi dezakegu erabiltzen dugun bakoitzean eskuz ez exekutatzeko.

A.2.2 Nabigazioa helburuak RViz bidez bidaliz

RViz bidez Marisorgini helburuak bidali nahi badizkiogu, bi *launch* fitxategi exekutatzearekin nahikoa da. Lehenengoak, Gazebo martxan jarri eta ingurunea nahiz Marisorgin kargatuko ditu. Bigarrenak, aldiz, nabigaziorako beharrezko nodoak eta RViz jarriko ditu martxan (*map_server*, *amcl*, *move_base* eta *tf* zuhaitza publikatzen duen *mari_description* paketeko *mari_state_publisher2.launch* fitxategiari deitzen dio).

Hortaz, bi komando hauek exekutatuko ditugu:

```
# roslaunch mari_gazebo mari.launch
```

```
# roslaunch mari_2dnav move_base_gazebo.launch
```

Guztia ondo joan bada, bi leiho irekiko dira: Gazebo eta RViz. Gazebon ingurunea ikus dezakegu eta bertan Marisorgin egongo da. RViz-en mapa bat ikusiko dugu eta hau erabiliko dugu helburuko kokapena zehazteko.

Helburua zehazteko RViz-en pantailaren goiko aldean agertzen den "*2D Nav Goal*" botoia sakatuko dugu lehenik eta behin (ikusi A.1 irudia), eta ondoren, mapako puntu batean klikatu eta helburuko orientazioa zehaztuko dugu geziaren bidez. Behin hau eginda, robota mugitzen hasiko da. Nahi dugun momentuan beste helburu bat bidali diezaiokegu Marisorgini.

A.2.3 Nabigazioa helburuak sortutako programaren bidez bidaliz

Marisorgini sortutako programarekin helburuak bidali nahi badizkiogu, bi *launch* fitxategi exekutatzearekin nahikoa da. Lehenengoak, Gazebo martxan jarri eta ingurunea

A.1 Irudia: Rviz-eko "*2D Pose Estimate*"eta "*2D Nav Goal*" botoiak

nahiz Marisorgin kargatuko ditu. Bigarrenak, aldiz, nabigaziorako beharrezko nodoak eta RViz jarriko ditu martxan (*map_server*, *amcl*, *move_base* eta *tf* zuhaitza publikatzen duen *mari_description* paketeko *mari_state_publisher2.launch* fitxategiari deitzen dio). RViz martxan jartzea nahi ez badugu, *move_base_gazebo.launch* fitxategitik nodo hori abiarazten duen lerroa ezabatu edo komentatu behar da.

Hortaz, bi komando hauek exekutatuko ditugu:

```
# roslaunch mari_gazebo mari.launch
```

```
# roslaunch mari_2dnav move_base_gazebo.launch
```

Guztia ondo joan bada, Gazebo martxan izango dugu.

Jarraian, komando lerro bidez hirugarren agindu bat idatziko dugu:

```
# rosrun simple_navigation_goals simple_navigation_goals
```

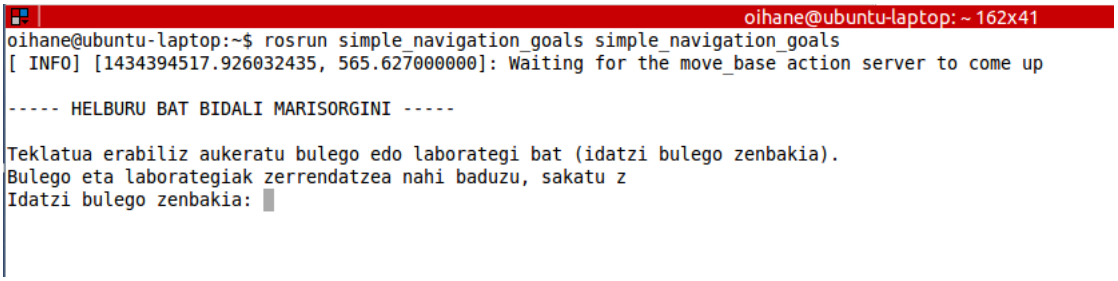
Pantailan A.2 irudian ikusten duguna azalduko zaigu. Idatzitakoa irakurtzen badugu, bi aukera dauzkagula ikusiko dugu:

1. Helburuko bulegoaren zenbakia idatzi.

2. "*z*" tekla sakatu eta bertan ikusi bulego posible guztiak.

Lehenengoa aukeratuz gero, bulego zenbakia idatzi dugun unean Marisorgin helbururantz mugitzen hasiko da. Bulegoen zerrenda pantailaratzeko eskatu badugu, aldiz, bulego zenbakia galdetuko digu berriro, eta hau sartutakoan helburua bidaliko zaio robotari.

Marisorgin helburura iristen denean, mezu batek hala adieraziko du. Helburuko puntura iritsi ezin badaiteke, berriz, errore mezu bat izango da gertatutakoaren berri emango duena.



A.2 Irudia: Helburuak bidaltzeko programaren interfazea

A.3 Errealitatea

A.3.1 Aurrekariak

Instalazioa

Nabigazioa martxan jarri aurretik, ROS robotean instalatuta dagoela ziurtatu beharko dugu. Hala ez bada, proiektu hau abiarazteko gomendatutako bertsioa *ROS Indigo* da eta konfigurazioa *Desktop-Full* izenekoa. Horretarako komando hau exekuta dezakegu:

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

Konfigurazio hori erabiliz ROS-ez gain, RViz, eta nabigazioa ere instalatuko dira, beste ROS pakete batzuen artean. Bestela, beharrezko ROS paketeak banaka instala ditzakegu.

Sortutako paketeak kopiatu eta konpilatu

Proiektu honetan sei ROS pakete eta haien fitxategiak sortu edo editatu dira: *mari_description*, *mari_gazebo*, *gazebo_plugins*, *map*, *mari_2dnav* eta *simple_navigation_goals*. Errealitatean, ordea, Gazebo-rekin zerikusia duten biak (*mari_gazebo* eta *gazebo_plugins*) ez ditugu behar. Beraz, gainontzeko lau paketeak *catkin workspace*-ean kopiatu eta *catkin_make* erabiliz konpilatuko ditugu.

A.3.2 Nabigazioa helburuak RViz bidez bidaliz

RViz bidez Marisorgini helburuak bidali nahi badizkiogu, bi *launch* fitxategi exekutatzearekin nahikoa da. Lehenengoak, robotaren *driver*-ak martxan jarriko ditu. Bigarrenak, aldiz, nabigaziorako beharrezko nodoak eta RViz jarriko ditu martxan (*map_server*, *amcl* eta *move_base*).

Hortaz, bi komando hauek exekutatuko ditugu:

```
    # roslaunch mari_bringup mari_platform.launch use_ptu:=1
use_laser:=1
```

```
    # roslaunch mari_2dnav move_base.launch
```

Guztia ondo joan bada, RViz-en pantaila irekiko da eta mapa bat ikusiko dugu bertan. Mapa hau helburuko kokapena zehazteko erabiliko dugu.

Lehenik eta behin, robota une horretan gutxi gorabehera non dagoen adierazi behar dugu, RViz-en pantailaren goiko aldean agertzen den "*2D Pose Estimate*" botoia erabiliz (ikusi A.1 irudia). Mapan klikatuz robotaren uneko kokapena eta orientazioa adierazteko aukera eskaintzen du tresna honek.

Helburua zehazteko, berriz, "*2D Nav Goal*" botoia sakatuko dugu lehendabizi (ikusi A.1 irudia), eta ondoren, mapako puntu batean klikatu eta helburuko orientazioa zehaztuko dugu geziaren bidez. Behin hau eginda, robota mugitzen hasiko da.

Nahi dugun momentuan beste helburu bat bidali diezaiokegu Marisorgini.

A.3.3 Nabigazioa helburuak sortutako programaren bidez bidaliz

Helburuak bidaltzeko programa fakultatean bertan martxan jartzeko beharrezkoa da Gazebo bidez sortutako mapa erabiltzea; edo bestela fakultatean mapa sortu beharko genuke eta *bulegoak.txt* fitxategiko koordenatuak aldatu. Gazebo-ko mapa erabiliz portaera egokia duela egiaztatu da, beraz, ez litzateke arazorik egon behar. Hala ere, erabiliko duen mapa lehenetsia fakultatean sortutakoa da, eta Gazebo-koaz baliatzea erabakitzen bada, *move_base.launch* fitxategian aldaketa txiki bat egin beharko da:

```
1 <node name="map_server" pkg="map_server" type="map_server" args="$(
    find map)/simulazioa/map.yaml"/>
```

Erabakitako mapa alde batera utzita, Marisorgini sortutako programa erabiliz helburuak bidali nahi badizkiogu, bi *launch* fitxategi exekutatzearekin nahikoa da. Lehenengoak, robotaren *driver*-ak martxan jarriko ditu. Bigarrenak, aldiz, nabigaziorako beharrezko nodoak eta RViz jarriko ditu martxan (*map_server*, *amcl* eta *move_base*). RViz martxan jartzea nahi ez badugu, *move_base_gazebo.launch* fitxategitik nodo hori abiarazten duen lerroa ezabatu edo komentatu behar da.

Hortaz, bi komando hauek exekutatuko ditugu:

```
# roslaunch mari_bringup mari_platform.launch use_ptu:=1
use_laser:=1
```

```
# roslaunch mari_2dnav move_base.launch
```

Guztia ondo joan bada, nabigazioa martxan izango dugu.

Jarraian, komando lerro bidez hirugarren agindu bat idatziko dugu:

```
# rosrun simple_navigation_goals simple_navigation_goals
```

Pantailan A.2 irudian ikusten duguna azalduko zaigu. Idatzitakoa irakurtzen badugu, bi aukera dauzkagula ikusiko dugu:

1. Helburuko bulegoaren zenbakia idatzi.
2. "*z*" tekla sakatu eta bertan ikusi bulego posible guztiak.

Lehenengoa aukeratuz gero, bulego zenbakia idatzi dugun unean Marisorgin helbururantz mugitzen hasiko da. Bulegoen zerrenda pantailaratzeko eskatu badugu, aldiz, bulego zenbakia galdetuko digu berriro, eta hau sartutakoan helburua bidaliko zaio robotari.

Marisorgin helburura iristen denean, mezu batek hala adieraziko du. Helburuko puntura iritsi ezin badaiteke, berriz, errore mezu bat izango da gertatutakoaren berri emango duena.

B. ERANSKINA

Sortutako ROS paketeen egitura

Eranskin honen bidez, proiektu guztian zehar sortutako (edo aldatutako) ROS paketeak eta hauetako fitxategi garrantzitsuenak zerrendatu dira.

```
../catkin_ws/src
├── mari_description
│   ├── urdf
│   │   └── mmari.urdf
│   ├── launch
│   │   └── mari_state_publisher2.launch
│   └── config
│       └── mari.rviz
├── mari_gazebo
│   ├── models
│   │   └── hiru_solairua
│   │       ├── model.config
│   │       └── model.sdf
│   ├── worlds
│   │   └── mari.world
│   └── launch
│       └── mari.launch
└── gazebo_plugins
```

```
    └── non_holonomic_drive.cpp
├── map
    ├── simulazioa
        ├── map.yaml
        └── map.pgm
    └── errealitatea
        ├── map.yaml
        └── map.pgm
├── mari_2dnav
    ├── config
        ├── base_local_planner_params.yaml
        ├── costmap_common_params.yaml
        ├── global_costmap_params.yaml
        └── local_costmap_params.yaml
    └── launch
        ├── move_base_gazebo.launch
        └── move_base.launch
└── simple_navigation_goals
    ├── bulegoak.txt
    └── src
        └── simple_navigation_goals.cpp
```

C. ERANSKINA

Bilera aktak

Aurkibidea

C.1 Konstituzio bilera

Data: 2015/02/23, 12:30

Lekua: Informatika Fakultateko robotikako laborategia

Bilerara hurbilduak:

- Oihane Parra
- Elena Lazkano
- Ekaitz Jauregi
- Txelo Ruiz
- Igor Rodriguez

Helburuak

- Proiektuan egin beharrekoa finkatu.

Bileran hitz egindakoa eta hartutako erabakiak

- Proiektua garatzeko erabiliko den ingurunea: *Ubuntu 14.04 SE*, *ROS Indigo* eta *Gazebo 2.2*. Honetarako, *ros-indigo-desktop-full* instalatu.
- Gazebo-n irudikatuko den eremua fakultateko hirugarren solairua izango da.
- Proiektua garatzeko fakultateko 1.3 laborategia erabiliko da aukeran.

Egin beharrekoak

Oihanek:

- *Ubuntu 14.04* sistema eragilea instalatu.
- Gazebo-ROS instalatu.
- Gazebo erabiltzen ikasi.
- Gazebo-n hirugarren solairua modelatu.

Txelok:

- Fakultateko planoak idazkariari eskatu.
- Proiektuaren garatzailea 1.3 laborategira sartzeko baimena eskatu.

C.2 Lehenengo bilera

Data: 2015/03/24, 11:30

Lekua: Informatika Fakultateko robotikako laborategia

Bilerara hurbilduak:

- Oihane Parra
- Elena Lazkano
- Ekaitz Jauregi
- Igor Rodriguez

Helburuak

- Egindakoa erakutsi.
- Eman beharreko hurrengo pausoak finkatu.

Aurretik egin beharrekoak

- Gazebo-n fakultateko hirugarren solairua modelatu.

Bileran hitz egindakoa eta hartutako erabakiak

- Gazebo-n garatutakoa erakutsi. Ate guztiak (eskaileretakoak izan ezik) gehitzea erabaki da, Marisorgin geletara sartu ahal izateko.
- RSAIT taldearen robotari dagokion URDF eredua proiektuaren garatzaileari pasa.
- Robotaren mugimendu-sistemaren funtzionamendua garatzaileari azaldu.
- Laserraren funtzionamendua garatzaileari azaldu.

Egin beharrekoak

- Ate guztiak Gazebo-n irudikatu (eskaileretakoak izan ezik)
- Robotaren URDF eredua aztertu
- Robota Gazebo-n sartu eta abiadura mezuak argitaratuz mugitu.
- Oztopoak ekiditeko sistemaren funtzionamendua ikertu
- Nabigazioa prestatu

C.3 Bigarren bilera

Data: 2015/05/25, 12:00

Lekua: Informatika Fakultateko robotikako laborategia

Bilerara hurbilduak:

- Oihane Parra
- Elena Lazkano

- Ekaitz Jauregi

Helburuak

- Egindakoa erakutsi.
- Zalantzak argitu.
- Eman beharreko hurrengo urratsak finkatu.

Aurretik egin beharrekoak

- Ateak Gazebo-n irudikatu.
- Nabigazioa prestatu.

Bileran hitz egindakoa eta hartutako erabakiak

- Gazebon egindako zuzenketak eta simulagailuko nabigazioa erakutsi. Helburuak bidaltzeko programa egokia dela erabaki da.
- Nabigazioak oztopoekin dituen akatsak zuzendu behar dira.
- Nabigazioko parametroak ez datoz bat robot ez-holonomoekin. Hau zuzendu behar da.
- Gazebo-ko ate guztiei 90 cm-ko zabalera jarri behar zaie.
- Robotak aldaketa batzuk izan dituenez (laserra lekuz aldatu dute), eredu berria pasa zaio proiektuaren garatzaileari.
- *Topic*-ak desberdinak dira Marisorgin errealean eta simulatutakoan. Ondoren errealitatean probatzeko arazoak ekiditeko, *topic* horiek aldatu.
- Memoria LaTeX erabiliz idatziko da.
- Hurrengo pausoa Marisorgin errealarekin nabigazioa probatu eta beharrezkoa zuzentzea da. Honetarako ekainak 3an gelditzea erabaki da.

Egin beharrekoak

- Gazebo-ko ateak konpondu (guztiak 90 cm-ko zabalerakoak).
- URDF eredu berria erabiltzeko beharrezko aldaketak egin.
- *Topic*-ak aldatu robot errealarenekin bat etortzeko.
- Oztopoak jarrita duen funtzionamendua ahalik eta hoberena izaten saiatu.
- Helburuak bidaltzeko programa amaitu.

C.4 Hirugarren bilera

Data: 2015/06/03, 10:00

Lekua: Informatika Fakultateko robotikako laborategia

Bilerara hurbilduak:

- Oihane Parra
- Elena Lazkano
- Txelo Ruiz
- Igor Rodriguez

Helburuak

- Zalantzak argitu.
- Robot errealean duen funtzionamendua aztertu.

Aurretik egin beharrekoak

- Simulagailuko nabigazioan hobekuntzak egin.
- Helburuak bidaltzeko programa amaitu.
- *Topic*-ak egokitu.

Bileran hitz egindakoa eta hartutako erabakiak

- Erabilitako URDF eredua ez zen zuzena. Berriarekin gertatzen dena aztertu behar da.
- Nabigazioko parametroak aztertu behar dira robot errealean duen portaera zuzentzeko.
- GAURen proiektuaren izenburua eguneratu behar da. Hau egiteko fakultateko idazkariari eskatuko zaio.

Egin beharrekoak

Oihane:

- Nabigazio simulatuak URDF berri zuzenarekin duen portaera aztertu.
- Marisorgin errealaren portaera zuzentzeko parametroak aztertu, eta behin eta berriz saiatu.

Elena:

- Fakultateko idazkariari GAURen proiektuaren izenburua eguneratzeko eskatu.

C.5 Itxiera bilera

Data: 2015/06/16

Lekua: Informatika Fakultateko robotikako laborategia

Bilerara hurbilduak:

- Oihane Parra
- Elena Lazkano

Helburuak

- Guztiaren funtzionamendua erakutsi (simulagailuan eta errealitatean).
- Egiteke falta dena zehaztu.

Aurretik egin beharrekoak

- Marisorgin errealean nabigazioak duen portaera zuzendu.
- Simulagailuko nabigazioa URDF berriarekin martxan jarri.

Bileran hitz egindakoa eta hartutako erabakiak

- Guztiak espero bezala funtzionatzen du. Hobekuntza posibleak memorian zerrendatuko dira.
- Memoria ahalik eta azkarren zuzendariari pasa behar zaio.

Egin beharrekoak

- Memoria amaitu eta defentsarako materiala prestatu.

Bibliografia

[1] Robot Operating System (ROS). http://ros.org. [2015eko maiatzean atzitua].

[2] Gazebo. http://gazebosim.org. [2015eko maiatzean atzitua].

[3] SDFormat. http://sdformat.org/. [2015eko maiatzean atzitua].

[4] Hokuyo. http://www.hokuyo-aut.jp/. [2015eko maiatzean atzitua].

[5] Robotika eta Sistema Autonomoen Ikerketa Taldea (RSAIT). http://www.sc.ehu.es/ccwrobot. [2015eko maiatzean atzitua].