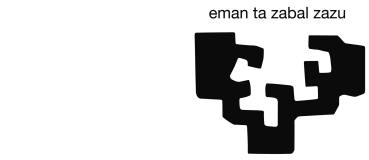


Informatika Ingeniaritzako Gradua

Konputazioa

Gradu Amaierako Proiektua

Teknibot Android App

Egilea

Ander Iriondo Azpiri

2017

Laburpena

Teknibot, IK4-TEKNIKER enpresak garatutako zerbitzu robot autonomoa da. Robot honekin elkarrekintza robotean bertan dagoen ukipen-pantaila baten bitartez egin daitekeen arren, gaur egun ezinezkoa da hau urrutitik kontrolatzea. Konputazioa espezialitateko Gradu Amaierako Proiektu honetan aplikazio mugikor bat garatu da, *Teknibot* robotarekin elkarrekintza hedatzeko helburu duena. Aplikazio mugikorra *Android* plataformara dago zuzendua eta ROSekin (Robot Operating System) komunikazioa landu da. Gainera erabiltzailearentzat erabilerraza gertatuko den interfaze bat garatu da. Proiektuaren garapenean ROS, C++, Java, XML, YAML, HTML, JSON eta C# teknologiak erabili dira, Unity motor grafikoaz gain.

Aplikazioak erabiltzailearentzat erabilgarriak diren funtzionalitate hauek ditu:

- Robotaren kokapena bistaratu, hau dagoen ingurune itxiaren maparen gainean.
- Robotak duen *Kinect* kamerak publikatzen dituen zenbait irudi bistaratu: RGB irudia, zuri-beltz irudia eta sakoneren irudia.
- Bai eskuz eta bai ahotsez aurretik definitutako zenbait kokapenetara gida lanak egiteko aukera eman .

Gaien aurkibidea

Irudien aurkibidea

Taulen aurkibidea

Algoritmoen zerrenda

1. KAPITULUA

Sarrera

Edukia

1.1 IK4-TEKNIKER

IK4 zenbait zentro teknologikoren arteko aliantza bat da eta honako enpresek osatzen dute: AZTERLAN, CEIT, CIDETEK, GAIKER, IDEKO, IKERLAN, LORTEK, VICOMTECH eta TEKNIKER. IK4-TEKNIKER Eibarren dago kokatuta eta proiektu hau bertan garatuko da, horrela, enpresa honetako teknologiak erabili ahal izango dira proiektua aurrera eramateko.

1.2 Android

Android mugikorrentzako sistema eragile bat da eta Linux nukleoan dago oinarrituta [1]. Googlek banatutako SDK deritzen garapenerako tresnak erabiliz edonork garatu ditzake plataforma honetarako aplikazioak. Google enpresak ia urtero ateratzen du SDKren bertsio berri bat eta proiektu honetan garatuko den aplikazio honi dagokionez 25.0 bertsioarekin konpilatuko da (Android 7) baina 19.0 bertsioraino egongo da erabilgarri (Android 4.4). Androiden web orri ofizialetik ateratako 1.1 irudian ikus daitekeen bezala, gailuen %80ak baino gehiagok dagoeneko Android 4.4 edo berriagoa dute eta horregatik erabaki da gutxieneko bertsiotzat hau jartzea.

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	1.7%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	1.6%
4.1.x	Jelly Bean	16	6.0%
4.2.x		17	8.3%
4.3		18	2.4%
4.4	KitKat	19	29.2%
5.0	Lollipop	21	14.1%
5.1		22	21.4%
6.0	Marshmallow	23	15.2%

1.1 Irudia: Android bertsio ezberdinen erabilpen tasa.

1.3 Teknibot

Jakina den bezala zerbitzu robotak gero eta garrantzi handiagoa hartzen ari dira gure bizitzetan. Hauek edozein arlotan aurki ditzakegu, adibidez nekazaritzan, abeltzaintzan, medikuntzan, edozein motako fabrikatan, etxeetan etab. Zerbitzu robotak gizakiari laguntzeko sortu dira eta gehienetan lan ez-atseginak edo errepikakorrak egiteko erabili ohi dira, etxeko eginbeharrak adibidez. Orokorrean autonomoak dira eta integratuta izaten dute kontrol sistema baten bidez kudeatzen dira.

Teknibot robota *IK4-TEKNIKER* enpresak sortutako zerbitzu robot autonomo, mugikor eta funtzio aniztuna da, leku itxietan gizakiarentzat lagungarri izateko diseinatua izan dena. Honen lehenengo bertsioak *KTbot* izena zuen eta hau garatzeko proiektuaren helburua leku itxietan laguntza eskaintzeko funtzionalitate "adimendunak" izango zituen zerbitzu-robot, mugikor eta autonomoaren prototipo ez komertzial bat garatzea zen.

Robot hau pertsonei laguntzeko pentsatua izan zen, gida lanak eginez hain zuzen ere. KTbot, Donostiako *Eureka* museoan egon zen duela urte batzuk eta bertsio berri honen helburua, honen gaitasunak hobetu eta berriz ere museoan martxan jartzea da.

1.3.1 Zerbitzu robotak

*IFR*ren (International Federation of Robotics) arabera [3]: *"Zerbitzu robot bat era autonomo edo semi-autonomoan nabigatzeko gai den eta gizakiaren ongizaterako baliagarriak diren zerbitzuak eskaintzen dituen robota da"*.

Ezaugarri nagusienetako bat gizakiarekiko duten gertutasuna da. Zerbitzu-roboten aplikazioak ondorengo arlo ezberdinetan multzokatzen dira IFRren arabera: Landako robotak, garbiketa profesionala, mantenu-sistemak, eraikuntza eta eraistea, sistema logistikoak, medikuntzarekin lotutako robotika, defentsa eta segurtasuna, ur azpiko sistemak...

Aurretik aipatutako arlo gehienetan autonomia maila txikia dute eta, gainera, egin beharreko lan asko errepikakorrak, aspergarriak eta askotan arriskutsuak dira. Gainera populazioaren etengabeko zaharkitzeak arlo hauetan lan egiteko roboten beharra handitzen du.

1.3.2 Teknibot robotaren funtzionalitateak

Teknibot robotak, 1.2 irudian ikus daitekeenak, eskaintzen dituen funtzionalitate guztiak gida lanak egitean oinarritzen dira eta hauek dira:

- **Gida:** Aurrez definitutako helburuz osatutako zerrenda batetik aukeratutako helburura gidatuko du erabiltzailea.
- **Laguntzaile:** Robotak egun bakoitzerako programatuta dauden ekitaldien zerrenda izango du eta hasi baino denbora tarte bat lehenago honi buruzko informazioa ahots

bidez esaten hasiko da. Ekitaldia hasi baino zertxobait lehenago bertara gidatuko ditu inguratutako pertsonak.

- **Logistika:** Aurrez definitutako ibilbide bat egingo du robotak, museoa osatzen duten eremu ezberdinetatik pasatuz.

1.2 Irudia: Teknibot robota.

1.4 ROS

Robot Operating System, *ROS* bezala ezagutua, sistema eragile batek eskaintzen duen funtzionalitatea duen framework bat da, robotentzat softwarea garatzea ahalbidetzen duena [11].

*ROS*en idatzitako programak *nodo* izenarekin ezagutzen dira. Guztiz banatuak dira eta, ondorioz, nukleo ezberdinetan prozesatzen dira inongo arazorik gabe. *Nodoek* euren artean komunikatu ahal izateko eta baita datuak transferitzeko *topic* izeneko datu egiturak erabiltzen dituzte. Proiektu hau adibidetzat hartuta, eskuartean daukagun robotak *topic* bat izango du bere posea publikatzeko; Beraz, datu hori behar duen edozein beste nodok

robotaren posizioa atzitu ahal izango du *topic* hori erabilita. *Topic* ezberdinetan idazten eta irakurtzen ari diren nodoak kontrolatzeko nodo nagusi bat existitzen da zeinak nodo guztien zerrenda bat izango duen, komunikazioa errazteko. Beraz, nodo batek *topic* batean informazioa publikatu nahi badu nodo nagusiari adierazi beharko dio zeinetan egin behar duen. Badaude aurretik definitutako zenbait datu egitura mezu gisa erabili ahal izateko, baina beharren arabera berriak definitu daitezke. Beste nodo batek *topic* bat irakurri nahi badu, nodo nagusiari adieraziko dio zeinetatik irakurri nahi duen. Horrela, nodo nagusiak *topic* horretan publikatzen duen nodoari adieraziko dio beste nodo batek publikatzen ari den informazioa behar duela eta, ondorioz, *peer-to-peer* motako konexio bat sortuko da bi nodoen artean.

Proiektu honi dagokionez, zenbait ROS nodo erabiliko ditugu robotak publikatzen duen informaziora suskribatzeko eta, ondoren, robotetik gailu mugikorrera bidali ahal izateko *TCP/IP* bidez. ROS, bai *Python*, *C++* eta *Java* programazio lengoaiekin erabili daiteke. Kasu honetan C++ erabiltzea erabaki da. Ondorioz, zenbait exekutagarri sortuko dira, gero banaka edo denak batera exekutatzeko aukerarekin.

1.5 Gaur egungo egoera

Gaur egun, populazioaren ehuneko handi batek du eskura telefono mugikor bat. Hauek eskaintzen dituzten baliabideak ikusita gero eta joera handiagoa dago plataforma haueta-rako aplikazioak inplementatzeko, erabiltzailearekiko irisgarritasun handia lortzen baita. Era guztietako aplikazioak existitzen diren arren, oraingoan robotekin elkarrekintza hobetzea helburu dutenetan zentratuko gara. Adibidetzat *iRobot* robota jarriko dugu [8]. Hau, xurgagailu lanak egiten dituen zerbitzu robot bat da eta aplikazio mugikor bat eskaintzen du urrunetik kontrolatzeko aukera emateko.

Rosjava, ROSen *java* bertsioa da eta Android sistema eragilerako egokitua dago [5]. Badaude zenbait aplikazio teknologia hau erabiliz inplementatu direnak, baina, Android bertsio zaharretarako inplementatuta daude eta aurkitu diren aplikazioetatik bat bera ere ez da komertziala. ROSen gain lan egiten duten robotekin komunikatzeko egokia dirudien arren, gaur egun zailtasunak daude Androiden bertsio berriekin erabili ahal izateko.

1.6 Proiektuaren helburuak

Teknibot robota zerbitzu robota izanik eta toki publiko batean pertsonei laguntzeko diseinatua izan denez, honek pertsonekin izango duen elkarrekintza oso handia izango da. Gaur egun, elkarrekintza integratuta duen ukipen-pantailaren bidez gauzatzen da eta honek dezente mugatzen du erabiltzailea, soilik pertsona batek izango duelako robotarekin elkarreragiteko aukera eta robotaren kokapenarekiko dependentzia sortzen delako. Hau hobetzeko asmoz eta telefono mugikorrek gure bizitzan duten garrantzia ikusita, oso baliabide egokia da elkarrekintza hedatzeko, aukera ematen baitu populazioaren zati handi batengana iristeko. Ondorioz, Android aplikazio mugikor bat sortzeko ideia sortu da, erabiltzaile soilentzat, aditu izateko beharrik gabe, robotarekin interakzio handia lortzeko aukera ematen baitu.

Beraz, proiektuaren helburu nagusia Android plataformarako aplikazio mugikor bat sortzea da aurretik aipatutako Teknibot robotarekin elkarrekintza hedatzeko asmoz. Bertan, robotaren inguruko zenbait informazio bistaratuko da erabiltzailearentzat erabilgarriak direnak, horien artean robotaren kokapena mapa baten gainean eta robotak duen *kinect* kamerak publikatzen dituen zenbait irudi. Gainera robotarekin elkarreragiteko aukera emango da, aurretik definituta izango dituen zenbait kokapenetara gida lanak egin ditzan robota kontrolatzeko aukera emanez. Hau bai eskuz, erabiltzaile interfazearen bidez, eta bai ahots agindu bidez, gauzatuko da. Aplikazioa proiektu handiago batean erabili ahal izateko diseinatuko da, alde batetik, lehen aipatu bezala, robotetik informazioa bidaltzeko modulua ROSeko nodoen bidez garatuko da zeinak guztiz modularrak diren. App mugikorraren atalari dagokionez, kontuan hartuko da aurrerago funtzionalitate berriak sartzeko aukera.

1.7 Hasierako erabakiak

Proiektuaren hasieran zenbait erabaki hartu behar izan dira, proiektuan garrantzi handia izan dutenak. Lehenik eta behin, aplikazioa garatzeko *rosjava* erabili edo ez erabaki behar izan da. Nahiz eta *rosjava* ROSeko *Kinetic* bertsioraino eskuragarri egon, ez du euskarririk. Zenbait proba egin ondoren eta Android bertsio berriekin erabili ahal izateko dituen zailtasunak ikusita, hau ez erabiltzea erabaki da. Beraz, Androideko aplikazioa ROSekiko independentea izango da.

Bestalde, Androideko aplikazioak izan beharreko funtzionalitateak erabaki ondoren, datu horiek bidaltzeko zerbitzariaren egitura erabaki behar izan da. Horrela, datu mota bakoitzerako zerbitzari bat sortzea erabaki da: Lehenengoa, robotaren poseak bidaltzeko; Bigarrena, komando bidezko eskaerak kudeatzeko; Eta hirugarrena, kamerako irudiak bidaltzeko.

2. KAPITULUA

Proiektuaren Helburuen Dokumentua

Edukia

2.1 Proiektuaren irismena

Atal honetan proiektuaren irismena deskribatzen da. Horretarako eta lehenik eta behin betekizunak *LDE* diagrama batean azalduko dira non atal bakoitzean egin beharreko lana deskonposatuko den. Gainera, proiektuaren emangarriak eta hauen mugarriak ere zerrendatuko dira.

Proiektu honen helburua *Teknibot* robotarekin elkarrekintza hedatzea da, horretarako Android plataformarako aplikazio mugikor bat sortuz, bezero-zerbitzari eredua jarraitzen duen aplikazio bat hain zuzen ere.
Bezero programa mugikorrerako aplikazioa izango da eta honen eginkizuna erabiltzaileren eta robotaren arteko lotura ezartzea izango da. Erabiltzaileak robota gida moduan erabili ahal izango du aginduak emateko aukera izanik, bai interfazearen bidez eta bai ahots bidez. Bestalde, robotaren kokapena ikusi ahal izango du robota kokatuta dagoen inguruneko maparen gainean. Azkenik, robotak duen kameraren irudi ezberdinak ikusteko aukera izango du, hala nola, RGB irudia, zuri-beltz irudia eta sakoneren irudia.
Zerbitzari programa robotean exekutatuko da eta honek, mugikorretik bidalitako eskaerak kudeatzeaz gain, robotetik zenbait informazio bidaliko dio bezeroari, aurretik aipatutako robotaren kokapena eta irudiak, besteak-beste.

Hauek izango dira proiektuaren atal nagusiak:

- **ROS-Android komunikazioa**
  - Bezeroa eta zerbitzariaren arteko TCP/IP konexioa.
- **Zerbitzariaren garapena**
  - Komandoak jaso eta erantzuteko nodoa.
  - Lokalizazioa bidaltzeko nodoa.
  - Irudiak bidaltzeko nodoa.
- **Bezeroaren garapena**
  - Interfazearen garapena.
  - Negozio logikaren garapena.
- **Probak**

  - Aplikazioa Teknibot robotean probatu.

2.1.1 LDE diagrama

2.1 irudian ikus daitekeen bezala, proiektua honako ataza desberdinetan banatzen da:

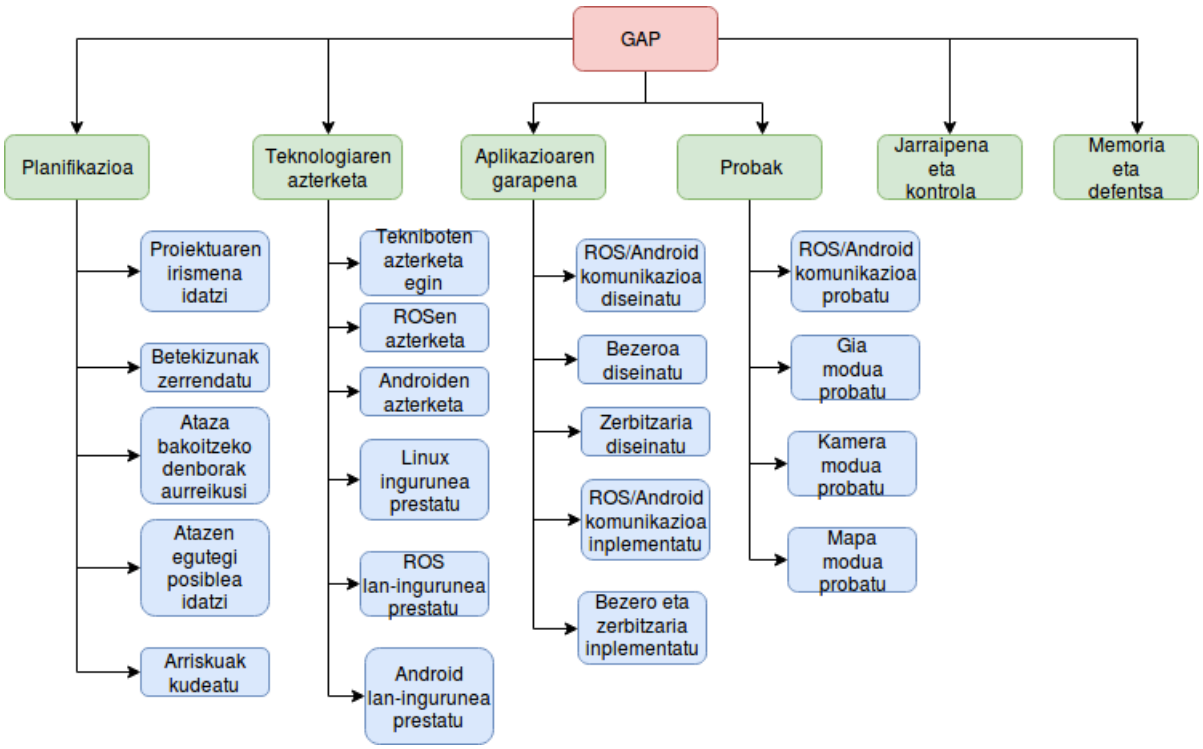
- **Planifikazioa:** Atal honetan, proiektuan zehar bete behar diren helburuak definitzen dira.
  - Irismena: Proiektua azaltzeaz gain, helburuak definitzen dira.
  - Betekizunak: Proiektuan zehar egin beharrekoak zerrendatzen dira.
  - Ataza bakoitzerako denborak aurreikusi: Ataza bakoitza aurrera eramateko behar izango den denbora aurreikusi.
  - Atazen egutegi posiblea idatzi: Ataza bakoitza denboran zehar noiz hasi daitekeen pentsatu eta egutegia egin.
  - Arriskuak kudeatu: Proiektuan zehar gerta daitezkeen arriskuak aurreikusi.
- **Teknologiaren azterketa:** Proiektua garatzen hasi aurretik erabili beharreko teknologiak erabaki eta aztertzen dira atal honetan.
  - Tekniboten azterketa: Teknibot robotak dituen funtzionalitateak aztertu, baita hauen egitura eta erabilera ere.
  - ROSen azterketa: Aplikazioaren zerbitzaria robotean exekutatuko denez, ROSen inplementatua egongo da eta beraz honen erabileraren inguruko azterketa egin beharko da.
  - Androiden azterketa: Bezeroa berriz, Android plataformarako diseinatuko da eta, beraz, honetarako teknologiak ere aztertu behar dira.
  - Linux ingurunea prestatu: Teknibot robotean ROS Linuxen gainean exekutatzen denez, Linux ingurunea prestatu behar da.
  - ROS ingurunea prestatu: Robotean exekutatuko den zerbitzaria, esan bezala ROSen inplementatua egongo da eta nahiz eta hau C++ lengoaian idazten den, *catkin* lan-ingurunea prestatu beharko dugu.
  - Android ingurunea prestatu: *Gradle* izena duen eta aplikazioak eraikitzeko balio duen sistema erabiliko da. Ondorioz, sistema honek eskatzen duen egitura izango duen lan-ingurunea sortu beharko da.

- **Aplikazioaren garapena:** Hemen, aplikazioaren garapenean eman beharreko pausuak definitzen dira.
  - ROS/Android komunikazioa diseinatu: Bezeroa eta zerbitzariaren arteko komunikazioa nolakoa izango den aztertu eta zein protokolo erabiliko den erabaki.
  - Bezeroa diseinatu: Androiden exekutatuko den aplikazio mugikorra nolakoa izango den aztertu, bai izango dituen funtzionalitateak eta bai interfazearen itxura ere.
  - Zerbitzaria diseinatu: Robotean exekutatuko den zerbitzariak bezeroari zein informazio bidaliko dion eta komando bidezko eskaerak nola kudeatuko dituen aztertu.
  - ROS/Android komunikazioa inplementatu.
  - Bezeroa inplementatu.
  - Zerbitzaria inplementatu.
- **Probak:** Behin aplikazioa garatuta honen funtzionamendua egokia dela ziurtatzeko egingo diren probak.
- **Jarraipena eta kontrola:** Proiektua garatu ahala, hasieran planifikatutakoarekiko geratu diren aldaketen kudeaketa.
- **Memoria eta defentsa:** Aplikazioaren garapena amaituta dagoenean, honi buruzko memoria garatu beharko da, azkenik honen defentsa egiteko.

2.1.2 Emangarriak

Proiektu honetan bi emangarri mota sortuko dira:

- **Aplikazioarekin lotutakoak:**
  - Aplikazioa osatzen duten bezero eta zerbitzariaren kodea, baita exekutagarriak ere.
- **Proiektuarekin lotutakoak:**
  - Proiektuaren memoria.
  - Proiektuaren defentsarako gardenkiak.



2.1 Irudia: LDE diagrama.

2.1.3 Mugarriak

Proiektu honetarako ezarritako mugarriak 2.2 irudian ikus daitezke.

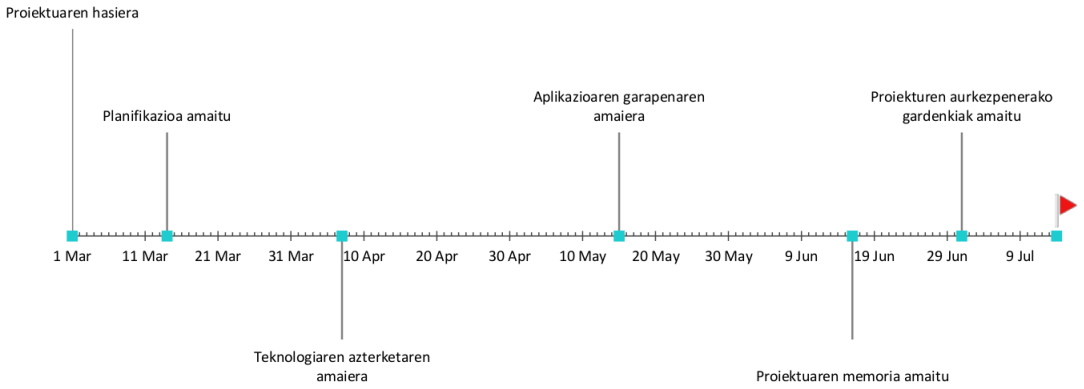
2.2 Denboraren planifikazioa

Estimazioak 2.1 irudian ikus daitekeen LDE diagramaren lehen mailako atazetan soilik egingo dira.

2.2.1 Atazen estimazioa

- **Proiektuaren planifikazioa egin:** 10 ordu.
- **Teknologiaren azterketa:** 30 ordu.
- **Aplikazioaren garapena:** 200 ordu.

- **Jarraipena eta kontrola:** 20 ordu.
- **Memoria eta defentsarako gardenkiak egin:** 80 ordu.



2.2 Irudia: Mugarrien diagrama.

Atazak	**Orduak**
Proiektuaren planifikazioa egin	10
Teknologiaren azterketa egin	30
Aplikazioa garatu	200
Jarraipena eta kontrola egin	20
Memoria eta defentsarako gardenkiak egin	80
GUZTIRA	340

2.1 Taula: Ataza bakoitzeko denbora estimazioak.

2.3 Kalitatearen kudeaketa

Atal honetan proiektuak jarraitu beharreko kalitate mailak zehaztuko dira.

2.3.1 Kalitatearen planifikazioa

Orokorrean proiektuak izan beharko duen kalitate maila zehazteko, bi maila hauek bereizi dira: Kalitate maila minimoa eta onargarria.

Kalitate maila minimoa

- **Produktua:** Produktuak irismena atalean aipatutako funtzionalitateak izan beharko ditu. Android aplikazio mugikorretik aginduak bidali ahal izango zaizkio robotari gida lanak egiteko, bai interfaze bidez, bai ahotsez. Bestalde, honek duen kamerak publikatzen dituen irudiak ikusi ahal izango dira. Azkenik, robotaren posizioa ikusi ahal izango da robota kokatuta dagoen ingurunearen maparen gainean. Kalitate maila minimoa betetzeko guzti honek aplikazio bakar batean funtzionatu beharko du.

  Gainera, bai bezero eta bai zerbitzariaren kodeak irakurgarria eta ulergarria izan behar du.

- **Memoria:** Memoria idazteko Informatika Fakultateak proposatzen duen formatuari jarraitu beharko zaio, horretarako Informatika Fakultatearen web orrian [4] dagoen txantiloia erabiliz.

- **Gardenkiak:** Aurkezpenerako denbora mugatua izanik, kontuan hartu behar izango da gardenki kopurua finkatzeko.

Kalitate maila onargarria

- **Produktua:** Gida moduan erabiltzaile bakar batek izan beharko du soilik robotaren kontrola, segurtasun arazorik ez gertatzeko. Ondorioz, aplikazioa esklusibotasun hori kudeatzeko gai izan beharko da. Beste bi funtzionalitateak, berriz, ez dira esklusiboak izango eta aplikazio ezberdinetatik atzigarri egon beharko dira. Horretarako zerbitzaria konkurrentea izan beharko da.

  Kodeari dagokionez, dokumentatuta egon beharko da, funtzio bakoitzak egiten duena adieraziz.

- **Memoria:** Memoriaren kalitateari dagokionez ez dago aldaketarik.

- **Gardenkiak:** Gardenkietan azalduko den testu kopurua lerro gutxitara mugatuko da, ideiak soilik idatziz.

2.3.2 Kalitatearen kontrola

- **Produktua:** Produktuaren kalitatea ebaluatzeko, *IK4-TEKNIKER* enpresako zenbait erabiltzailerekin esperimentu bat egingo da. Hauek aplikazio mugikorra instalatu eta probatuko dute, ondoren iritzia jasotzeko.
- **Memoria:** Memoriaren kalitatea proiektuaren zuzendariak ebaluatuko du entregatu aurretik.
- **Gardenkiak:** Hauek ere proiektuaren zuzendariak ebaluatuko ditu.

2.4 Interesatuak

Proiektu honengan interesatuak honako hauek dira:

- Proiektuaren egilea
- Proiektuaren zuzendaria
- IK4-TEKNIKEReko tutorea
- IK4-TEKNIKER

2.5 Arriskuen kudeaketa

Atal honetan proiektuan zehar gerta daitezkeen eta proiektuaren garapena atzera dezaketen arriskuak aurreikusiko dira.

2.5.1 Informazio galera

Azalpena

Proiektua osatzen duten fitxategien galera, hala nola, aplikazioaren kodearen fitxategiren bat edo dokumentu bat adibidez.

Prebentzioa

Hau ekiditeko bertsio kontrol sistema bat erabiliko da proiektuan zehar, *Git* [2] hain zuzen ere. Horrela proiektuaren garapena aurrera joan ahala *gitlab* plataforman joango dira aurrerapenak gordetzen.

2.5.2 Robotaren eskuragarritasuna

Azalpena

Aplikazioa *Teknibot* robotean erabiltzeko zuzenduta dagoen arren, enpresako lankide batzuk robot berarekin aritzen dira lanean eta, ondorioz, robota ez da beti eskuragarri egongo.

Prebentzioa

Ahal den neurrian proba guztiak konputagailu lokalean egingo dira, horrela robota eskuragarri egoteko zain egotea ekidinez denbora asko aurreztuko da. Gida modua, adibidez, robotean soilik proba daitekeenez, robotaren menpekotasuna erabatekoa izango da.

2.5.3 Aurreikuspenak ez betetzea

Azalpena

Erabili beharreko teknologia gehienak berriak direnez, hasiera bateko planifikazioak asko desbideratu daitezke.

Prebentzioa

Atazak ahalik eta modu eraginkorrenean burutzen joan denborarik galdu gabe.

3. KAPITULUA

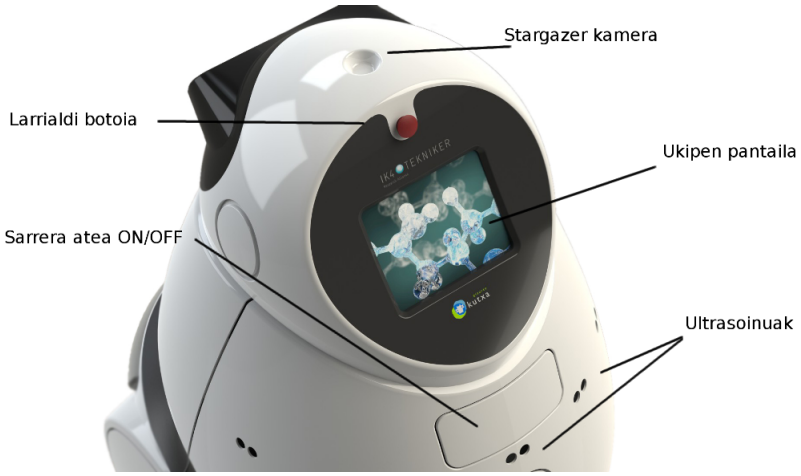
Tresnak eta teknologiak

Edukia

3.1 Teknibot robota

3.1 eta 3.2 irudietan Teknibot robota ikus dezakegu, kanpotik ikusgarriak diren elementuak zer diren azalduta.

3.1 Irudia: Teknibot robotaren ikuspegi orokorra.

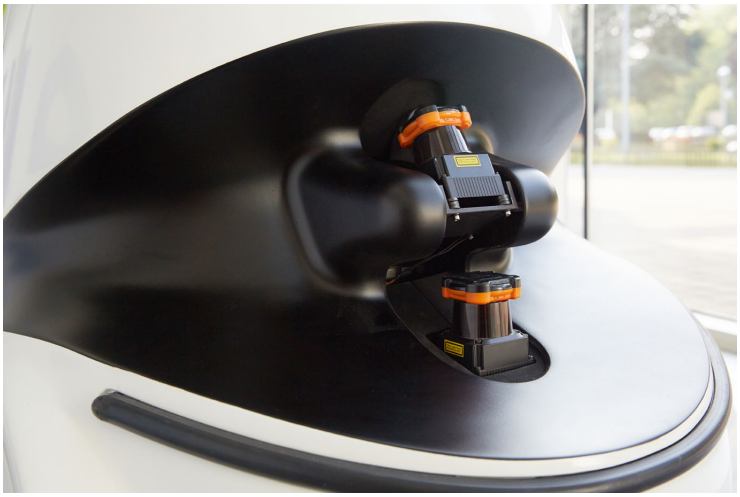


3.2 Irudia: Teknibot robotaren burua atzealdetik.

3.1.1 Sentsoreak

Teknibotek zenbait sentsore ezberdin erabiltzen ditu ingurunea antzemateko. Horietako bakoitzak *driver* bat izango du zeinak irakurketak *topic* batean publikatuko dituen. Horrela, irakurketa horiek eskuragarri egongo dira robotaren logika osatzen duten nodoek atzitu ditzaten. Erabiltzen dituen sentsoreak jarraian azaldutakoak dira:

- **Laserrak:** Bi laser sentsore erabiltzen dira. Bi laserrak *Hokuyo-UTM-30LX* motakoak dira eta 270º-ko atzipen eremua dute, 0.25º-ko erresoluzioarekin. Irakurketa bakoitza distantzia bat da eta 0.1 eta 30 metro bitarteko balioak emango dizkigu. Lehenengoa, estatikoa da eta horizontalki egiten ditu irakurketak. Bestea berriz, serbomotor bati lotuta dago eta honek bertikalki mugitzen du laserra. Horrela, *XY* planoko distantziaz gain, objektuen altuera ere antzeman daiteke. Laser sistemaren informazioa, oztopoak detektatzeko eta lokalizaziorako erabiltzen da. Sentsore hauek 3.3 irudian ikus daitezke.



3.3 Irudia: Laserrak, behekoa estatikoa eta goikoa serbomotorrari lotutakoa.

- **Infragorriak:** Sentsore hauek *Sharp GY2Y0A21YK (2Y0A21)* motakoak dira eta objektuek igortzen duten erradiazio elektromagnetiko infragorria jasotzeko gai dira. Hauek ere, oztopoak detektatzeko erabiltzen dira. 3.4 irudian ikus daitezke sentsore hauek. Teknibotek horrelako bi sentsore ditu eta aurre-azpialdean ditu kokatuta.

- **Kinect kamera:** Kamera honek zenbait sentsore ezberdin ditu: Infragorri kamera, RGB kamera eta mikrofono array bat. Gainera infragorri igorle bat ere badu, argi

3.4 Irudia: Infragorri sentsoreak.

gutxi dagoen egoeretan infragorri kamera ibili ahal izateko. Tekniboten *Kinect*a pertsonen detekziorako erabili izan da soilik, baina, gaur egun lokalizaziorako eta inguruneko hiru dimentsioko mapa sortzeko ere erabiltzen hasi dira. Kamera hau, 3.5 irudian ikus daiteke.

3.5 Irudia: Kinect kamera.

- **Ultrasoinuak:** Sentsore hauek *Devantech SRF08* motakoak dira eta hegaldi denboraren menpeko sentsoreak dira. Beraz, igorpen bat egin eta itzultzen den arteko denbora erabiltzen da objektuen gertutasuna kalkulatzeko. Hauek 3.6 irudian ikus daitezke. Teknibotek honelako zazpi sentsore ditu eta pantailaren azpian, atzealde

osoa inguratzen, kokatuta daude. Hauek erabiliz atzealdean dituen oztopoak antzemango ditu.

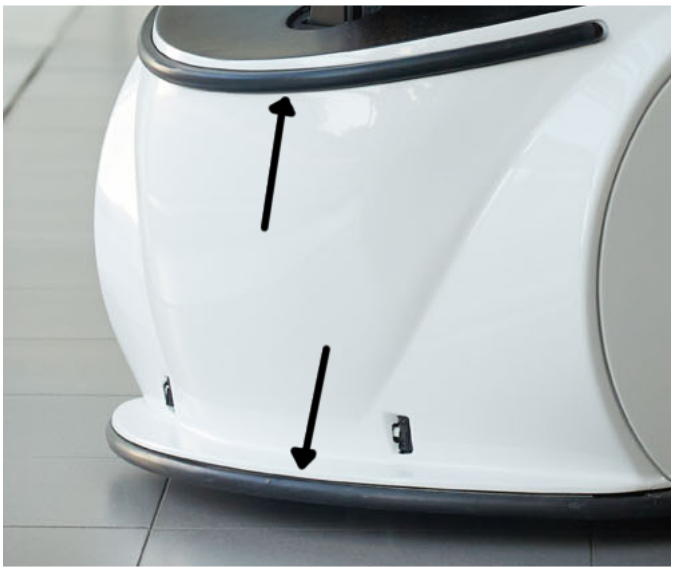
3.6 Irudia: Ultrasoinu sentsoreak.

- **Stargazer kamera:** *IDS USB 2 uEye UI-5240CP-NIR-GL* motako kamera infragorri bat da eta sabaian jarritako etiketa batzuk detektatzeko gai da. Kamera mota hau, roboten lokalizaziorako erabiltzen da eta hau 3.7 irudian ikus daiteke.

3.7 Irudia: Stargazer kamera.

- **Bumperrak:** Hauek talka sentsoreak dira eta izenak dioen bezala objekturen batekin talka egin duela antzemateko erabiltzen dira. Talkarik egon dela antzemanez

gero, robota gelditu egingo da, bai objektua/pertsona eta bai robota ez kaltetzeko. Teknibotek honelako bi ditu, bata oinarri osoa inguratzen, eta bestea aurrealdean laser sentsoreen azpian. Sentsore hau 3.8 irudian ikus daiteke.



3.8 Irudia: Bumperrak.

3.1.2 Nabigazio sistema

Robotaren nabigazioa, robotak bere koordenatu sisteman duen posizioa ondorioztatu eta helburu batera joateko bideak planifikatzeko gaitasunean oinarritzen da. Ingurune batean nabigatzeko, beharrezkoa da ingurune horren errepresentazio bat izatea eta horretarako normalean mapa bat erabiltzen da. Gainera, errepresentazio hori ulertzeko gaitasuna izan beharko du robotak. Beraz, nabigazioa hiru kontzeptu hauen konbinazioa da:

- Lokalizatzeko gaitasuna.
- Helburuetara joateko bideak planifikatzeko gaitasuna.
- Mapa sortu eta interpretatzeko gaitasuna.

Tekniboti dagokionez, bi dimentsioko leku itxietan nabigatzeko diseinatua dago. Nabigatzeko behar duen mapa, nahiz eta posible den automatikoki eratzen joatea nabigazioan

zehar, robotaren arduradunek eskuz sortzen dute aldez aurretik. Horretarako robota modu autonomoan jarri beharrean, *joystick* bat erabiliz gidatzen dute.

Nabigaziorako ROSeko *navigation* paketea erabiltzen da [9]. Honek, odometria, sentsoreen informazioa eta helburuen posea erabiliz, abiadura agindu batzuk sortzen ditu. Agindu horiek robotak mugitzeko behar duen oinarri higikorrera bidaltzen dira, Tekniboten kasuan *Segway RMP 200* motako *segway* bat.

Navigation paketea zenbait paketez osatuta dago, horien artean *amcl* [6]. *Amcl* lokalizazio sistema probabilistiko bat da, bi dimentsioko inguruneetan mugitzen diren robotetan erabiltzen dena. Pakete honek *adaptive Monte Carlo localization* algoritmoa inplementatzen du [14]; Algoritmo honek, partikula filtro bat erabiltzen du robotaren lokalizazioa mapa jakin baten gainean irudikatzeko. Partikula bakoitza robotaren kokaleku posible bat izango da. Hasiera batean partikulak mapa osoan zehar egongo dira sakabanatuta, baina robota mugitzen den heinean, sentsoreen informazioaz baliatuta, partikula multzoa txikituz joango da, gero eta zehatzagoa izanik. Horrela, poseen estimazioak hobetuz joango dira eta robotaren kokapena gero eta zehatzago adierazi ahal izango da maparen gainean. Pakete hau garrantzitsua izango da proiektuan zehar, Android aplikazioan Tekniboten lokalizazioa irudikatzeko beharrezko informazioa bertatik lortuko baita.

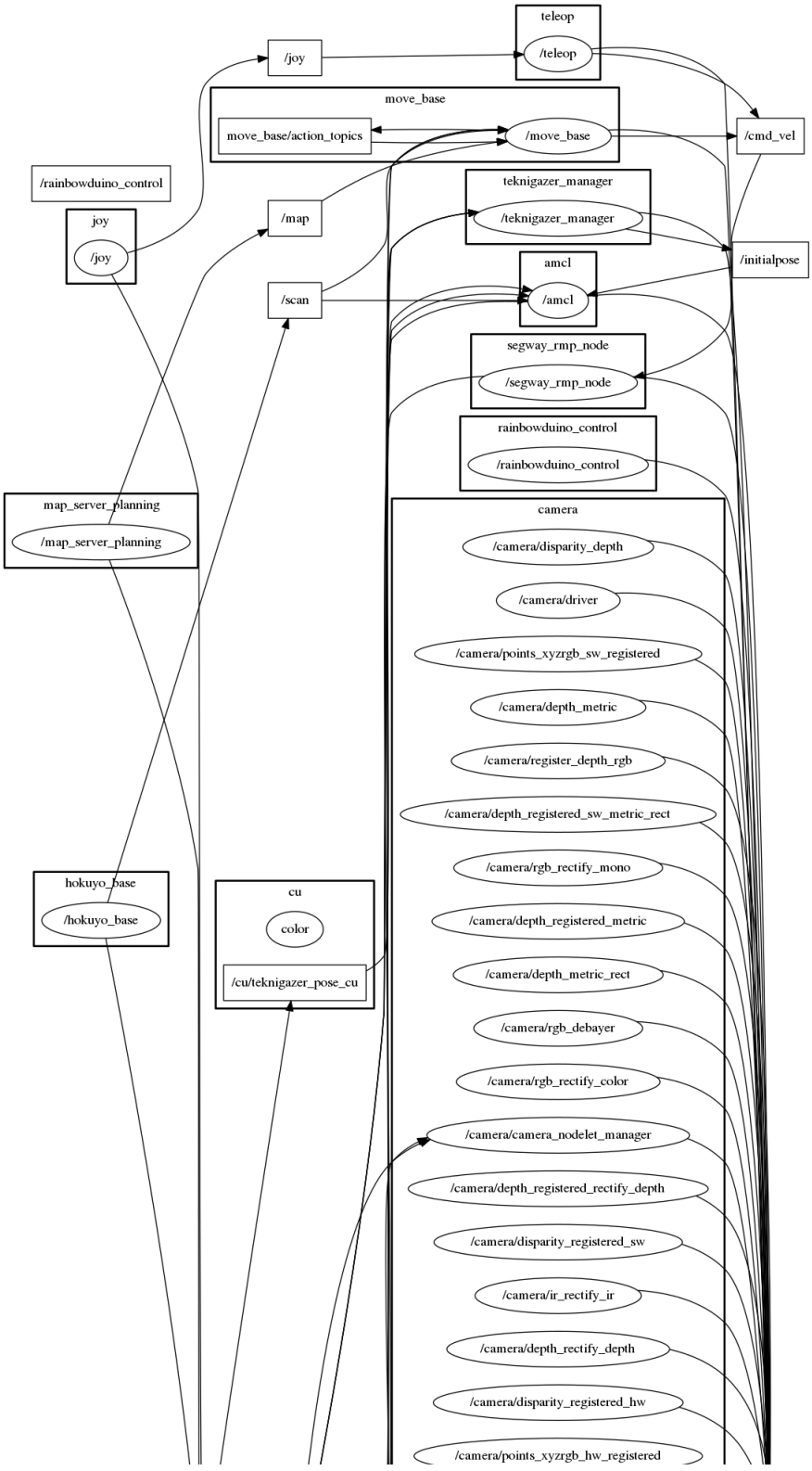
3.1.3 Teknibot robotaren topic-ak

Teknibot robotak erabiltzen dituen *topic*-ak 3.9 irudian ikus daitezke.

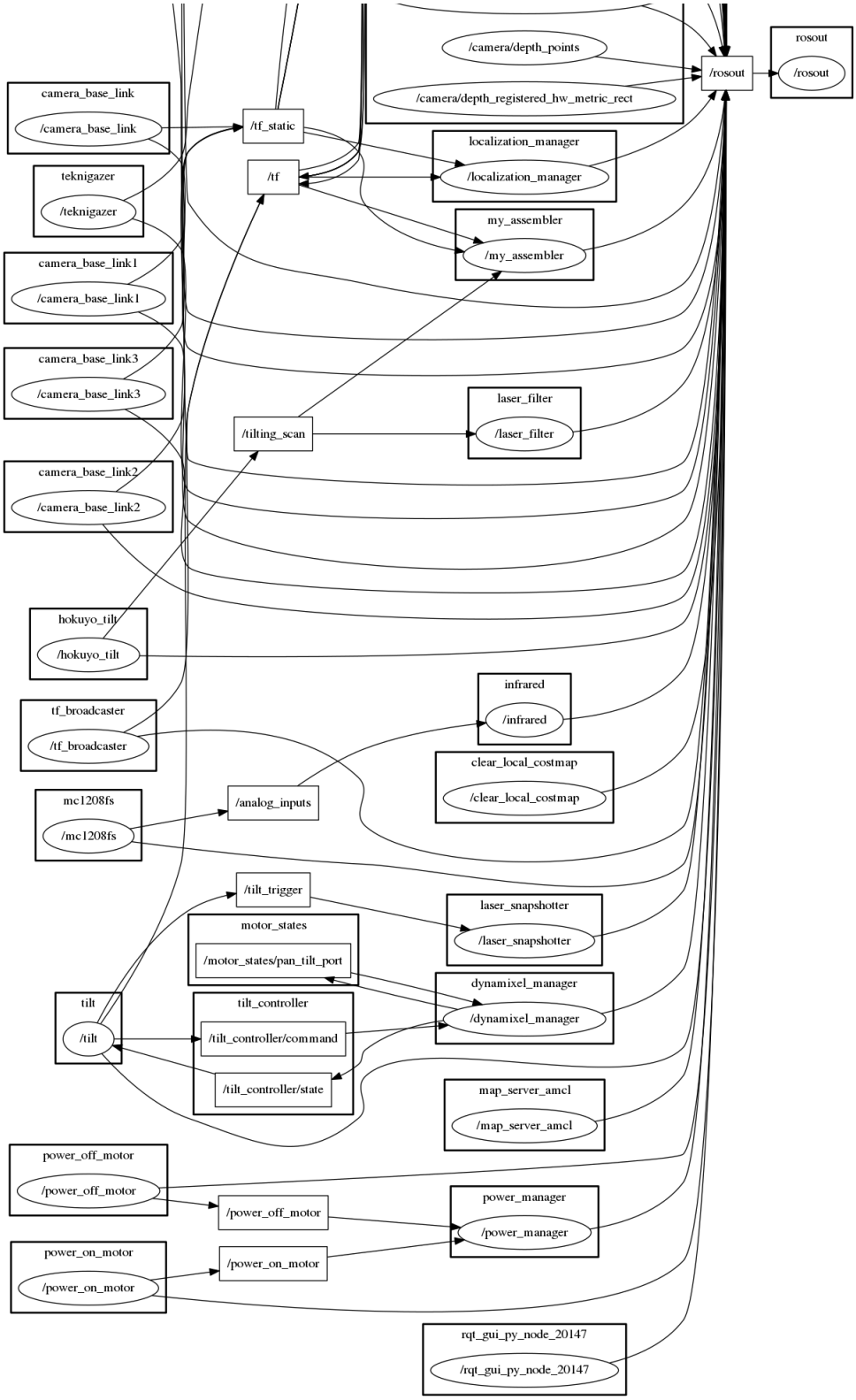
3.2 Android

3.2.1 Negozio logika

Lehen esan bezala aplikazio mugikorra Android sistema eragilerako diseinatu eta inplementatu da. Plataforma honetarako aplikazioak garatzeko jatorrizko lengoaia *java* da baina badaude zenbait framework beste lengoaia batzuetan programatzea ahalbidetzen dutenak, hala nola, *html, css, javascript* .... Proiektu honetan jatorrizko programazio lengoaia erabili da.



3.9 Irudia: Teknibotek erabiltzen dituen *topic*-ak.



3.9 Irudia: Teknibotek erabiltzen dituen *topic*-ak.

3.2.2 Interfazea

Interfazea garatzerako orduan, estatikoki edo dinamikoki eratzeko aukera dago. Dinamikoki sortzeak, interfazea eratzen duten elementuak jatorri kodetik sortu eta kokatu daitezkeela esan nahi du. Estatikoki sortzeak, berriz, *xml* fitxategiak definitzea esan nahi du non ikuspegi bakoitzeko fitxategi bat definitzen den, interfazeko elementuak aldez aurretik kokatuz. Aplikazio honetan gehienbat estatikoki definitu da interfazea nahiz eta honekin elkarrekintza jatorri kodetik kudeatzen den.

3.2.3 Android-Studio

Garapenerako *Android Studio* IDEa erabili da, Ubuntu 16.04ren gainean instalatuta. Hau Androiderako aplikazioak garatzeko tresna ofiziala da eta era askotariko tresnak eskaintzen ditu. Hona hemen horietatik gehien erabili direnak:

- **SDK manager:** Tresna honek aplikazioa konpilatzeko beharrezkoak diren tresna multzoak kudeatzeko aukera ematen du. Era errazean, erabili nahi den Android bertsioa aukeratu daiteke eta bertatik deskargatu.

- **AVD manager:** Jarraian sakonago azalduko den bezala, tresna honek makina birtualak sortzeko eta kudeatzeko aukera ematen du era errazean.

- **Monitoreak:** Tresna honek gure aplikazioak erabiltzen dituen baliabideak kontrolatzeko balio digu. Adibidez, aplikazio mugikorrak momentu bakoitzean erabiltzen duen PUZ portzentaiaren, RAM memoriaren erabileraren eta internet konexioaren abiaduraren grafikoak erakusten ditu.

- **Logcat:** Hau erabilita, garatzen ari garen aplikazioaren exekuzioan publikatzen diren log mezuak ikusi ahal izango dira, akatsak aurkitzeko lagungarria izanik. Bestalde, irteera estandarreko mezuak ere ikusi ahal izango dira bertatik.

- **Android Device Monitor:** Erabiltzen ari garen makina birtualaren "tripak" ikuskatzeko balioko digu tresna honek. Nahiz eta zenbait aukera ezberdin eskaintzen dituen, jarraian azalduko direnak soilik erabili dira proiektu honen garapenean.

  – File explorer: Tresna honekin makina birtualaren fitxategi sistemara sartu ahal izango gara, fitxategiak kudeatzeko. Fitxategiak sortu eta ezabatzeko aukera

emateaz gain, konputagailu lokaletik fitxategiak igotzeko aukera ere ematen du.

  - Hierarchy view: Tresna honek gure aplikazioaren interfazea osatzen duten ikuspegien hierarkia erakutsiko digu.

- **Interfazeak diseinatzeko tresna:** Interfaze estatikoak definitzea, hau da, *xml* fitxategietan definitzen direnak, eskuz idaztea nahiko pisutsua gerta daiteke. Horregatik tresna honek interfazea grafikoki diseinatzeko aukera ematen digu. Interfazeko elementuak kutxa batetik hautatu eta pantailan era bisualean kokatzea ahalbidetzen digu denbora asko aurreztuz. Gainera interfazeko elementu bakoitzaren propietateak erraz aldatzeko aukera ematen digu.

3.2.4 AVD

Android Virtual Device izenez ezagutzen dira aplikazioak konputagailu lokalean probatu ahal izateko emuladoreak. Hauek oso erabilgarriak gertatzen dira uneoro aplikazioa mugikorrera kopiatu eta instalatzen ibili behar ez izateko. Hauek Android Studiorekin edo independenteki erabili daitezke, komando lerrotik adibidez. Android Studiok eskaintzen duen tresna erabiliz gero, aukera izango dugu makina birtual berriak sortzeko, nahi ditugun baliabideak ezarriz, adibidez: RAM memoriaren edukiera, PUZaren abiadura, sistema eragilearen bertsioa, etab.

3.3 ROS Kinetic

Eskuarteko proiektuak bi atal izango ditu baina atal honetan "robotaren aldean" zentratuko gara. Robotaren aldea, aplikaziotik komando bidezko eskaerak kudeatzeaz gain, robotetik informazioa jaso eta aplikazio mugikorrera bidaltzeaz arduratzen den atalari deituko diogu.

Atal hau *Robot Operating System*en (ROSen) programatuko da, *Kinetic* bertsioa erabiliz hain zuzen ere. Robotak instalatua duen sistema eragilea Ubuntu 16.04 denez, beharrezkoa izango da ROSen bertsio hau erabiltzea. Hauek dira proiektuaren atal honetan erabili diren kontzeptu nagusiak:

- **Catkin ingurunea:** Nodoak *catkin* lan-ingurune batean garatuko dira.

- **Nodoa:** ROSeko aginduak biltzen dituen exekutagarri bat da. Hauek, jarraian azalduko den *topic*-etan publikatu edo suskribatu daitezke, euren artean komunikatzeko. Horrez gain *zerbitzuak* ere erabili ahal izango dituzte; Hauek, geroago azalduko diren bezala, beste era bateko komunikazioa eskaintzen dute. Gainera nodoak euren artean guztiz banatuak dira, exekuzio paraleloa ahalbidetuz.

- **Topic:** Nodoen arteko datu transferentzian erabiltzen diren biltegiratze-unitateak dira. Nodoek *topic*-etan idatzi edo hauetatik irakurri ahal izango dute eta nodo batek ez du jakingo publikatzen ari den *topic*-a norbait irakurtzen ari den edo alderantziz, ea irakurtzen ari den *topic*-ean zein nodo ari den idazten. Beraz, komunikazio asinkronoa ahalbidetzen duela esan daiteke, bi nodoren arteko idazketa/irakurketek ez dutelako zertan aldi berean izan behar.

- **Zerbitzua:** Publikatu/suskribatu bidez komunikatzeko era oso malgua da eta nodo anitzen arteko komunikazioa gertatu daiteke. Baina batzuetan bi nodoren arteko eskaera/erantzun itxurako komunikazioa behar izaten da, sistema banatuetan ohikoa den bezala. Komunikazio mota hau sinkronoa dela esaten da, eskaerak eta erantzunak denbora errealean egiten direlako.

- **Mezuak:** Nodoen arteko datu transferentziarako erabiltzen dira, bai *topic*-etan eta bai zerbitzuetan. *Topic*-ei dagokienez, *topic* bakoitzak mezu mota bat izaten du lotuta eta, beraz, bertan publikatzen duen nodoak mota horretako mezu bat sortu beharko du, dagokion informazioarekin bete eta, azkenik, publikatu. Suskribatzeko, berriz, jarraian azalduko den metodo bat erabili beharko da non mezu motaz gain, *callback* funtzio bat definitu beharko den jasotzen den mezu bakoitza prozesatzeko. Adibidetzat, 3.10 irudian *Pose.msg* mezu-fitxategiaren definizioa ikus daiteke. Zerbitzuei dagokienez, berriz, datu transferentzia bi mezuren bidez egiten da, bat eskaerarako eta beste bat erantzunerako. Hauek definitzeko *.srv* fitxategiak erabiltzen dira eta eskaerarako *request* eta erantzunerako *response* atalak izan ohi dituzte. 3.11 irudian *SetBool.srv* zerbitzu-fitxategiaren definizioa ikus daiteke.

- **Bag fitxategiak:** *Bag* fitxategiak ROSeko fitxategi mota bat dira eta mezuen datuak gordetzeko balio dute. Fitxategi hau sortzerako orduan grabatu nahi diren *topic*-ak aukeratu daitezke eta zenbait helburu ezberdinetarako erabiltzen dira, besteak beste, simulazioak egiteko, gordetzeko, analizatzeko etab. Gero fitxategi hauek erre-

```
Point position
Quaternion orientation
```

3.10 Irudia: .msg fitxategi baten definizioa.

```
bool data #request
---
bool success #response
string message
```

3.11 Irudia: .srv fitxategi baten definizioa.

produzitu daitezke, robotak publikatzen dituen *topic*-en simulazio bat lortuz. Bai fitxategiak sortzeko eta bai erreproduzitzeko *rosbag* komandoa erabiltzen da.

- **Launch fitxategiak:** Fitxategi hauek nodoen exekuzioa errazteko erabiltzen dira. Bertan, zenbait aukera ezberdin sartu daitezkeen arren, proiektu honetan exekutatu nahi diren nodoak eta parametro zerbitzariari pasa beharreko kanpo-parametroen fitxategia adieraziko dira. 3.12 irudian *.launch* formatuko fitxategi baten adibidea ikus daiteke.

```
<launch>
    <rosparam command="load" file="parameters.yaml"/>
    <param name="/use_sime_time" value="true"/>
    <node pkg="amcl_server" type="amcl_server_node" name="amcl_server"/>
    <node pkg="server" type="server_node" name="server"/>
    <node pkg="web_video_server" type="web_video_server" name="web_video_server"/>
    <node pkg="find_robot_server_udp" type="find_robot_server_udp_node" name="
     find_robot_server_udp"/>
</launch>

```

3.12 Irudia: Launch fitxategi baten adibidea.

- **Parametro zerbitzaria:** Hiztegi baten antzekoa da eta nodoek exekuzioan zehar hau atzitu dezakete parametroak elkarbanatzeko. Normalean konfigurazio parametroak elkarbanatzeko erabiltzen da eta parametro hauek *yaml* formatuko fitxategi batean definitu daitezke.

ROSeko nodoak *catkin* inguruneetan garatzen dira. *Catkin* ingurune bat, paketeak garatu eta instalatzeko sortutako karpeta bat da. Gure kasuan pakete bakar bat sortuko da zeinak nodo guztiak izango dituen barnean. *Cmake* tresnan oinarritzen da eta proiektu handien konpilazioa, instalazioa eta exekuzioa errazten ditu.

ROS nodoetan *topic*-etan irakurri edo idazteko honako metodo hauek erabiltzen dira:

- **advertise():** Nodo batetik *topic* batean idatzi nahi dugunean metodo hau erabili beharko dugu. Lehenik eta behin nodoak *Publisher* motako objektu bat sortu beharko du. Bestalde *NodeHandle* motako objektu bat ere sortu beharko du, honek baitu *advertise* metodoa inplementatzeko interfazea. Beraz, aipatutako *Publisher* motako objektuari *NodeHandle* objektuak duen *advertise* metodoa esleitu beharko zaio. Honek parametro bezala publikatu nahi den *topic*-aren izena eta idatziko dituen datuen ilararen tamaina jasotzen ditu. Behin *Publisher* motako objektua prest dagoela, *publish* metodoa erabili beharko da mezuak publikatzeko, parametro bezala mezua bera jasotzen duelarik. 3.13 irudian ikus daitek *Publisher* baten kodearen adibidea.

- **subscribe():** Metodo hau *topic* batera suskribatu nahi dugunean erabiliko dugu. Kasu honetan ere hasiera batean *NodeHandle* motako objektua sortu beharko da, honek daukan *subscribe* metodoa inplementatzeko. Parametro bezala irakurri beharreko *topic*-aren izena, idatziko dituen datuen ilararen tamaina eta *callback* funtzioa jasotzen ditu. Funtzio honek parametro bezala *topic*-ak esleitua duen mota bereko mezu bat jasotzen du, ondoren mezuaren edukiarekin egin beharrekoa egiteko. Honen adibide bat 3.14 irudian ikus daiteke.

```
1  ros::NodeHandle n;
2  ros::Publisher chatter_pub = n.advertise<std_msgs::String>("topic_izena", 1000);
3  ros::Rate loop_rate(10);
4  int count = 0;
5  while (ros::ok())
6  {
7    std_msgs::String msg;
8    std::stringstream ss;
9    ss << "hello world " << count;
10   msg.data = ss.str();
11   ROS_INFO("%s", msg.data.c_str());
12   chatter_pub.publish(msg);
13   ros::spinOnce();
14   loop_rate.sleep();
15   ++count;
16 }
17 return 0;
18
```

3.13 Irudia: Publisher baten adibidea.

ROS zerbitzuetan, *topic*-en antzera, bi funtzio beharko ditugu bezero edo zerbitzari bezala jokatzeko.

- **serviceClient():** Metodo hau zerbitzu baten bezeroa sortzeko erabili beharko dugu. Hau ere *NodeHandle* interfazean dago definituta eta *ServiceClient* motako objektu

```
ros::NodeHandlePtr n=boost::make_shared<ros::NodeHandle>();
ros::Subscriber sub = n->subscribe("amcl_pose", 1000,poseCallback);
ros::Subscriber sub1 = n->subscribe("particlecloud", 1000, particleCloudCallback);
ros::spin();
```

(a) NodeHandle eta Subscriber objektuen sorrera.

```
void particleCloudCallback(const geometry_msgs::PoseArray::ConstPtr& msg){
\\msg ko datuak prozesatu
}
```

(b) *Callback* funtzioaren definizioa.

3.14 Irudia: Topic batera suskribatzeko adibidea.

bati esleitu beharko zaio. Parametro bezala zerbitzuren izena jasoko du. Bezeroa izango den objektua sortuta dagoenean, zerbitzu motari dagokion mezu bat sortu beharko da, behar izanez gero hauek duten *request* atalean beharrezko informazioa sartuz. Azkenik *ServiceClient* motako objektuak duen *call* funtzioa erabili behar da hau bidaltzeko. Metodo hau erantzunaren zain geratuko da eta, azkenik, bidalitako mezuaren *response* atalean eskatutako zerbitzuaren erantzuna jasoko dugu. Kode zati baten adibidea 3.15 irudian ikus daiteke.

- **advertiseService():** Zerbitzu baten zerbitzaria sortzeko metodo hau erabili beharko dugu. Hau ere *NodeHandle* interfazean dago definitua eta *ServiceServer* motako objektu bati esleitu beharko zaio. Parametro bezala zerbitzuaren izena eta *callback* funtzioaren izena jasoko ditu. Nodoen antzera, funtzio honek bezeroen eskaerak jasoko ditu, zerbitzuak esleitua duen motakoak. Honek jasotako mezuaren *response* atalean idatziko du bezeroari bidali beharreko erantzuna. 3.16 irudian ikus daiteke zerbitzu baten zerbitzariaren adibidea.

Exekuzioari dagokionez, nodoak banaka edo multzoka exekuta daitezke. Banaka exekutatu nahi izanez gero, lehenik eta behin komando lerrotik nodo nagusia hasieratu beharko da, honek kudeatzen baititu *topic* eta zerbitzuak. Horretarako *roscore* komandoa erabiltzen da. Bestalde, nodoak banaka exekutatzeko *rosrun* komandoa erabiltzen da, paketearen izena eta nodoaren izena pasata. Hau lan neketsua izan daiteke nodo kopurua handia denean eta horregatik *.launch* fitxategiak erabiltzen dira. Hauetan exekutatu nahi diren nodoak adierazten dira, baita pasa nahi diren parametroak ere.

```
ros::ServiceClient client;
client=nh.serviceClient<std_srvs::SetBool>("guia_mode_access");
std_srvs::SetBool srv;
srv.request.data=true;
if(client!=NULL){
   if(client.call(srv)){
       if(srv.response.success){
           return 0;
       }else{
           return 1;
       }
   }
}
return -1;

```

3.15 Irudia: Zerbitzu bateko bezeroaren adibidea.

```
ros::NodeHandle n;
ros::ServiceServer service = n.advertiseService("add_two_ints", add);
ROS_INFO("Ready to add two ints.");
ros::spin();

```

(a) NodeHandle eta ServiceServer objektuen sorrera.

```
bool add(beginner_tutorials::AddTwoInts::Request  &req,
         beginner_tutorials::AddTwoInts::Response &res)
{
  res.sum = req.a + req.b;
  return true;
}

```

(b) *Callback* funtzioaren definizioa.

3.16 Irudia: Zerbitzu bateko zerbitzariaren adibidea.

3.4 Ubuntu 16.04

Debian sistema eragilean oinarritutako *Linux* sistema eragilea da Ubuntu. Robotak instalatuta duen sistema eragilea da eta bertsio honi dagokion ROS bertsioa *Kinetic* da.

3.5 Windows 10 eta Unity

Robotaren lokalizazioa bistaratzeko Unity motor grafikoa erabili da [7]. Unity tresna ezaguna da mugikorretarako jokoak inplementatzeko, baliabide grafikoak ondo kudeatzen baititu azpitik *OpenGL* erabiliz. Gainera 2D edo 3D inguruneak sortzeko erraztasunak ematen ditu eta gure kasuan erabilgarria gertatuko zaigu robota mapan kokatzeko komenigarria baita koordenatu sistema bat sortzea. Gaur egun Unity-k beta bertsio bat dauka Linux sistema eragileetarako baina honek funtzionamendu arazoak ematen dituenez, Windows sistema eragilearen gainean erabiltzea erabaki da. Honen gaineko programazioa *C#*-en idatzitako script-en bidez egiten da, 5 kapituluan sakonago azalduko den bezala. Editoreari dagokionez, ingurunean kokatu beharreko irudien kudeaketa grafikoki egiten da. Programazioari dagokionez, *Visual Studio* erabiltzen da.

Atal honetan, *GameObject* izeneko kontzeptua oso garrantzitsua da. Unity-ko proiektu bat hasieratzerakoan bi edo hiru dimentsioko ingurunea eraiki nahi den erabaki ondoren, *GameObject* deritzenak dira ingurune horretan parte hartzen duten objektuak. Beraz, ingurune horretan kokatutako edozein objektu hasiera batean koordenatu sistema globalean egongo da baina, ondoren, posible izango da hori aldatzea. *GameObject*-ak bata bestearen ume egin daitezke, horrela umearen koordenatu sistema gurasoa izatera pasatuz. Gainera, objektu bakoitzaren jokabidea atxikituta duen script batean definitzen da, gure kasuan *MonoBehaviour* klasearen ume izanik. Unity-ko script guztiak honen herentziaz sortuak dira. Hauek dira erabilitako metodo batzuk:

- **Start():** Hau, *MonoBehaviour* klasetik herentzian hartutako metodo bat da. Metodo honetan objektua hasieratzen denean exekutatu beharreko kodea idazten da, normalean hasieraketak egiteko.
- **Update():** Metodo hau frame bakoitzeko exekutatzen da, hau da, ingurunea irudikatzen den bakoitzeko. Normalean objektuaren une bakoitzeko jokabidea aldatzeko erabiltzen da.

- **Funtzio arruntak:** Script hauetan funtzio arruntak defini daitezke, gure kasuan funtzio hauek erabiliko dira Androidetik komunikatzeko, posible baita bertatik hauei deitzea.

Unity erabiliz plataforma ezberdinetarako proiektuak sortu daitezkeen arren, proiektu honetan aplikazio mugikorrean integratu nahi denez, Android proiektu bezala esportatzea interesatuko zaigu. Beraz, mahaigaineko Unity editoreari Android SDK tresna multzoa non daukagun adierazi behar zaio, proiektuak sortu eta konpilatzeko behar baita. Behin Android Studiok ulertuko duen proiektua izanda, garapen guztia proiektu honen gainean egingo da.

3.6 LER

Gaztelaniazko testuen analisi semantikoa ahalbidetzen duen zerbitzua da, IK4-TEKNIKER enpresan garatua [13]. Zerbitzu honen erabilera Teknibot robotarekin lotuta dago, honi bidalitako aginduak "ulertzeko" pentsatua izan baitzen.

Aplikazio mugikorrak, beraz, gaztelaniazko ahots aginduak testu bihurtu beharko ditu gero zerbitzu hau erabili ahal izateko. Horrela, ahots bidezko aginduak interpretatzea lortuko dugu, robotarekin elkarrekintza hobetuz.

Zerbitzu honek *http* protokolo bidez egiten du lan eta komunikazioa *JSON* formatuan egingo da.

3.7 Sare lokala

Robota eta aplikazio mugikorraren arteko komunikazioa *Wifi* bitartez izango da eta biak sare lokal berdinera egon beharko dira konektatuta.

4. KAPITULUA

Eskakizunen analisia

Edukia

4.1 Eskakizun ez funtzionalak

Aplikazio mugikor hau honako helburu nagusiak kontuan izanda garatuko da:

- Klik gutxiren bitartez aplikazioak eskaintzen dituen funtzionalitate guztiak eskura edukitzea.
- Edozein erabiltzailerentzat erabilerraza izatea, trebezia handirik behar izan gabe.

Kontuan izanda eskuarteko robota toki publiko batean laguntza gisa erabiltzeko pentsatua izan dela, aintzat hartuko da erabiltzaileak adin tarte handikoak izan daitezkeela. Ondorioz, aplikazioak dituen funtzionalitateak eta hauek erabiltzeko erabiltzailearen interfazea horretan pentsatuz diseinatu eta inplementatuko dira.

4.1.1 Interfazearen eskakizunak

Interfazea ahalik eta intuitiboena egiteko, aplikazioak eskaintzen dituen funtzionalitate guztiak bistan jartzea erabaki da, uneoro bistan egongo den menu barra batekin. Gainera, funtzionalitate bakoitzak ikono bat izango du atxikituta, funtzionalitateek eskaini dezaketena adierazteko esanguratsuak.

4.2 Eskakizun funtzionalak

Aplikazio mugikorra robotarekin elkarrekintzan aritzeko sortuko denez, erabiltzailearentzat interesgarriak izan daitezkeen hiru funtzionalitate sortuko dira. Jarraian azalduko den modeloa zenbait pantaila edo ikuspegiz osatua dago zeinak erabiltzaile mota guztientzat komunak diren. Hauek *PX* izenez deituko ditugu hemendik aurrera, non *P* pantaila eta *X* pantaila-indizea diren.

4.2.1 Robota sare lokalean bilatzeko pantaila

Funtzionalitate honek duen ikuspegiari hemendik aurrera *P1* deituko diogu. Pantaila honen egitura guztiz arrunta izango da, botoi bat soilik izango baitu. Sakatuz gero, aplikazioa robota sare lokalean bilatzen hasiko da eta hori adierazteko karga ikuspegi bat bistaratuko du. Honen helburua erabiltzaileari aplikazioa lanean dagoela eta ez dela bertan behera geratu adieraztea izango da. Robota aurkitzen badu, 4.2.2 azpi-atalean azalduko den P2 pantailara pasako da. Aurkitu ezean, elkarrizketa-leiho bat irekiko da robotaren IP helbidea eskuz sartu ahal izateko. Erabiltzaile arrunt batek seguruenik haren IP helbidea jakingo ez duenez, leiho hori ixteko aukera izango du hasieran aipatutako botoia berriz ere bistaratuz eta bilaketa prozesua berrabiaraziz.

4.2.2 Pantaila nagusia

Pantaila hau *P2* etiketarekin erreferentziatuko da hemendik aurrera. Robotaren IP helbidea eskuratuta, aplikazioa zuzenean pantaila nagusira pasako da. Pantaila honek kudeatuko ditu robotaren funtzionalitateak. Behealdean nabigazio barra bat izango du, zeina funtzionalitatez aldatzeko erabiliko duen erabiltzaileak. Bestalde, P2 pantailan posible izango da aplikazioak eskaintzen dituen moduen artean batetik bestera aldatzea. Beraz, menuaz

gain pantaila honek ikuspegi nagusi bat izango du, beste hiru azpi-ikuspegi kudeatzeko. Azpi-ikuspegi hauek hemendik aurrera *AX* bezala aipatuko dira, non *A* azpi-ikuspegia eta *X* azpi-ikuspegiaren indizea den.

Gida modua

Erabilpen kasu honek duen ikuspegia, hemendik aurrera *A1*, elementu ezberdinez osatua egongo da. Erabilpen kasu hau da robotaren kontrola har dezakeen bakarra eta, beraz, oso garrantzitsua izango da eskaeren arteko koherentzia mantentzea. Izan ere, erabiltzaile bakar bat izango da robotaren kontrola izango duena eta baimenak ondo kudeatu beharko dira. Horretarako erabilpen kasu honek izango duen hasierako ikuspegian botoi soil bat agertuko da erabiltzaileak robotaren kontrola hartu nahi duela adierazteko. Botoia sakatu ondoren, inor ez bada robota kontrolatzen ari, hau kontrolatzeko baimena eskuratuko da eta beste ikuspegi batera pasako da erabiltzailea. Bigarren ikuspegi honetan bi aukera emango dira robotari aginduak bidali ahal izateko eta honek gida lanak egiteko, interfaze grafikoa erabiliz eta ahotsez.

Interfaze grafikoa erabiliz aginduak bidali ahal izateko, robotak aurretik definituta dituen zenbait helburu deskargatuko ditu aplikazioak eta zerrenda batean erakutsiko zaizkio erabiltzaileari. Bat aukeratuz gero, robota bertarantz joaten hasiko da eta aplikazioaren ikuspegia aldatu egingo da hirugarren batera pasatuz, robota bidean dela adierazteko eta erabiltzaileak helburu berririk ez aukeratzeko. Hirugarren ikuspegi honetan aukera egongo da helburua bertan behera uzteko eta robota lekuan bertan gelditzeko. Horretarako botoi sinple bat erabiliko da.

Ahots komandoak bidali ahal izateko, aurretik aipatu den zerrendaren ondoan botoi bat gehituko da mikrofono baten irudiarekin. Erabiltzaileak hori sakatuz gero, elkarrizketaleiho bat irekiko da eta ahotsa entzuten geratuko da. Aplikazioak, esandakoa ulertzen ez badu, errepikatzeko eskatuko dio erabiltzaileari. Ulertzen badu, berriz, beste elkarrizketaleiho bat irekiko da. Ulertutako agindua eta robotak duen ezagutzaren arteko konparazioa eginda, agindu posibleak erakutsiz. Horrela erabiltzaileari aukera berresteko eskatuko zaio gaizki-ulerturik ez egoteko. Ahotsa ulertzen badu baina bere ezagutzarekin bat datorren agindurik ez badu, mezu bat pantailaratuko da hori adieraziz.

Kamera

Ikuspegi honi *A2* deituko diogu. Erabilpen kasu honetan aplikaziotik robotera konektatzen den erabiltzaileak posible izango du robotak "ikusten" duena ikustea. Horrela, robotak duen *Kinect* kamerak publikatzen dituen irudi ezberdinak ikusteko aukera emango da hala nola RGB irudia, sakoneren irudia eta zuri-beltz irudia. Irudi horiek automatikoki mugikorraren pantailaren tamainara egokituta bistaratuko dira. Irudi mota batetik bestera aldatzeko modua ahalik eta gehien sinplifikatuko da, hiru botoi jarriz.

Mapa

Azken ikuspegi honi *A3* deituko diogu. Honen helburua robotak nabigatzeko erabiltzen dituen sentsoreen informazioa erabiltzaileari erakustaraztea izango da, hala nola robotaren lokalizazioa. Ikuspegi honen muina Unity-rekin sortutako azpi-ikuspegia izango da zeinak 2D ingurune bat adierazten duen, bere koordenatu-sistemarekin. Horrez gain tresna-barra bat ere izango du pantailaren goialdean. Gainera ingurune horrekin elkarrekintza erraza izateko diseinatu da, hatzak erabiliz posible izango da maparen gainean *zoom* egitea.

5. KAPITULUA

Proiektuaren garapena

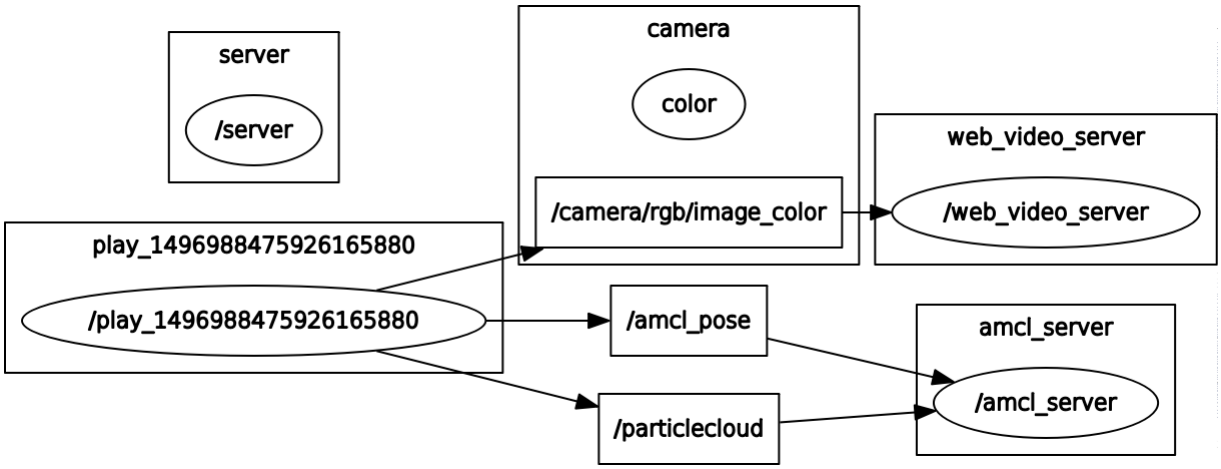
Edukia

5.1 Robotaren aldeko garapena: zerbitzaria

Atal honen helburu nagusia robotak publikatutako informazioa aplikazio mugikorrera bidaltzea izango da. Gainera, aplikaziotik egindako eskaerak kudeatu eta erantzun beharko dira. Eskakizunen analisian aipatu bezala, aplikazio mugikorrak hiru erabilpen kasu nagusi izango ditu eta, ondorioz, bakoitza kudeatzeko zerbitzari ezberdin bat sortu da:

- Maparen gainean irudikatzeko robotaren poseen estimazioak etengabe bidaltzeko zerbitzaria.
- Robotak duen kameraren irudiak irudi-korronte bidez bidaltzeko zerbitzaria .
- Komando bidezko eskaerak kudeatzeko zerbitzaria, batez ere gida moduan erabiltzen dena.

5.1 irudian zerbitzaria osatzen duten nodoen arteko *topic* bidezko komunikazioa ikus daiteke. Irudiko *amcl_server*, pose zerbitzaria da; *server*, komando zerbitzaria, eta *web_video_server*, berriz, irudien zerbitzaria.



5.1 Irudia: Nodoen arteko *topic* bidezko elkarrekintza.

5.1.1 Robota sarean aurkitzeko UDP zerbitzaria

Diseinua

Robotaren bilaketa prozesua osatzen duen elkarrizketa bi mezuz osatua dagoenez, egokiena UDP protokoloa erabiltzea erabaki da antzeko bilaketa prozesuetan ohikoa delako. Jarraituko duen protokoloa 5.1 taulan azaltzen da.

Jasoko duen eskaera	HELLO
Bidaliko duen erantzuna	OK

5.1 Taula: Robota sarean aurkitzeko UDP zerbitzariak darabilen protokoloa.

Inplementazioa

1 algoritmoan zerbitzari honen sasikodea ikus daiteke:

Algoritmoa 1 UDP zerbitzariaren sasikodea.

```
1: Socket-a sortu.
2: Socket-ari helbidea esleitu.
3: while 1 do
4:     if bufferra==HELLO then
5:         bufferra=OK.
6:     end if
7:     Bufferra bidali.
8: end while
9: Socket-a itxi.
```

5.1.2 Poseen zerbitzaria

Diseinua

Zerbitzari hau ROSeko nodo bat izango da, honen helburua robotak publikatzen dituen poseen estimazioetara suskribatu eta aplikaziora bidaltzea da. Lehenik eta behin, datu horiek bidaltzeko erabili beharreko protokoloa aukeratu da. Aukeran TCP edo UDP izanda,

hasiera batean UDP erabiltzea pentsatu zen azkartasuna behar delako datu kopuru handiak era azkarrean bidaltzeko. Baina TCP erabiltzeak baditu bere abantailak, paketeen galeraren kudeaketa etab. Lehenengo proba TCP erabiliz egin ondoren eta abiadura aldetik arazorik ez dagoela ikusita, hau erabiltzea erabaki da. Hartu beharreko beste erabaki garrantzitsu bat zerbitzaria konkurrentea egin edo ez izan da. Jakinda uneoro datu horiek erabiltzaile batek baino gehiagok jasotzeak robotaren ikuspuntutik ez duela inongo segurtasun arazorik sortzen, konkurrentea izatea erabaki da.

Nodo honek robotaren kokapena lortzeko ROSeko *amcl* paketeak publikatzen dituen *amcl_pose* eta *particlecloud topic*-etara suskribatuko da.

- **amcl_pose:** *Topic* honetan robotaren une bakoitzean estimatutako posea publikatzen da kobariantza matrizearekin batera. Erabiltzen den mezu mota *geometry_msgs* paketeko *PoseWithCovarianceStamped* motakoa da. Mezu mota honek duen *PoseWithCovariance* motako *pose* izeneko aldagaia soilik beharko dugu. Pose honek puntu baten *X, Y* eta *Z* koordenatuak eta orientazioa adierazten duen koaternoi bat izango ditu barnean eta azken honetatik *Z* ardatzarekiko biraketa kalkulatu beharko dugu. Poseak maparen koordenatu sisteman egongo dira adierazita.
- **particlecloud:** *Topic* honetan estimatutako poseen partikula multzoa publikatzen da, multzo honetatik benetako posea izateko probabilitate gehien daukana izango da *amcl_pose topic*-ean publikatzen dena. *Topic* honek erabiltzen duen mezu mota *geometry_msgs* paketeko *PoseArray* motakoa da. Mezu honek duen *Pose[]* motako *poses* izeneko aldagaia erabiliko dugu, hau pose ezberdinez osatutako array bat izanik. Pose estimazio hauek aplikazioan pantailaratuz robotaren posizioa ez dela zehatza adieraziko dugu, estimatua baizik.

Azaldutako *topic* horietan mezu bat publikatzen den bakoitzeko, TCP zerbitzariak mezu bat eratuko du eta bezeroei bidaliko die.

Inplementazioa

Nodo honen sasikodea 2 algoritmoan ikus daiteke.

Beraz, bai *amcl_pose* eta bai *particlecloud*-en *callback* funtzioetan TCP zerbitzariak dituen *set_amcl_pose* eta *set_particlecloud* metodoak erabiliko dira bidali beharreko mezuak ezartzeko. TCP zerbitzariaren egitura, berriz, 3 algoritmoan ikus daiteke.

Algoritmoa 2 Poseen zerbitzariaren nodoaren sasikodea.

```
1: TCP zerbitzariaren objektu globala sortu portu bat emanda.
2: procedure NAGUSIA()
3:     ROS hasieratu.
4:     NodeHandle objektua sortu.
5:     amcl_pose topic-era suskribatu, callback funtzio bat parametro bezala adieraziz.
6:     particlecloud topic-era suskribatu, callback funtzio bat parametro bezala adieraziz.
7: end procedure
8: procedure AMCL_POSE_CALLBACK(PoseWithCovarianceStamped mezua)
9:     mezua.pose-tik x eta y puntuak lortu.
10:    mezua.quaternion erabiliz Z ardatzarekiko biraketa kalkulatu.
11:    Bidali beharreko mezua eratu.
12:    Tcp zerbitzariak duen metodoa erabili amcl_poseri dagokion mezua ezarri eta bidaltzeko.
13: end procedure
14: procedure PARTICLECLOUD_CALLBACK(PoseArray mezua)
15:    for pose in mezua.poses do
16:        pose tik x eta y puntuak lortu.
17:        quaternion erabiliz z ardatzarekiko biraketa kalkulatu.
18:        Bidali beharreko mezuari posea gehitu.
19:    end for
20:    TCP zerbitzariak duen metodoa erabili particlecloudi dagokion mezua ezarri eta bidaltzeko.
21: end procedure
```

Algoritmoa 3 Poseak bidaltzeko TCP zerbitzariaren sasikodea.

```
1:  procedure SET_AMCL_POSE(mezua)
2:      amcl_pose=mezua.
3:  end procedure
4:  procedure SET_PARTICLECLOUD(mezua)
5:      particlecloud=mezua.
6:  end procedure
7:  procedure KONEXIO_BERRIA(socket-a)
8:      lag=0.
9:      while 1 do
10:         if lag==0 then
11:             Bidali amcl_pose-ri dagokion aldagaia socket-a erabiliz.
12:             lag=1.
13:         else
14:             Bidali particlecloud-i dagokion aldagaia socket-a erabiliz.
15:             lag=0;
16:         end if
17:     end while
18: end procedure
19: procedure ONARPEN_BEGIZTA()
20:     while 1 do
21:         Konexioak onartu.
22:         konexio_berria(socket-a) metodoa hari berri batean exekutatu.
23:     end while
24: end procedure
25: procedure NAGUSIA()
26:     Socket-a sortu.
27:     Socket-ari helbidea esleitu.
28:     Socket-a entzute socket bezala jarri.
29:     onarpen_begizta().
30: end procedure
```

Topic	Komandoa
amcl_pose	POS
particlecloud	ARRAY

5.2 Taula: Pose zerbitzariaren protokoloa.

TCP zerbitzariaren bitartez *topic*-etatik jasotako datuak bezeroari bidaltzeko protokolo sinple bat diseinatu da, ikus bedi 5.1 taula. Protokolo horretako komandoak erabiliz, itxura honetako mezuak bidaliko zaizkio bezeroari:

- **amcl_pose:**

  POS x y z yaw

  non yaw z ardatzarekiko biraketa den.

- **particlecloud**

  ARRAY LUZ,x y z yaw,x y z yaw...

  non LUZ bektorearen luzera eta yaw *Z* ardatzarekiko biraketa den.

5.1.3 Irudien zerbitzaria

Diseinua

Hau ere ROSeko nodo bat izango da eta honek robotaren *Kinect* kamerak publikatzen dituen irudiak aplikazio mugikorrera bidaliko ditu. Hasiera batean *OpenCV* liburutegia erabiltzea pentsatu zen, ROSeko irudiak tratatzeko sarritan erabiltzen delako eta gainera Androiderako liburutegia ere eskuragarri dagoelako. Hasierako ideia, publikatutako irudi bakoitza prozesatu eta TCPz banaka banaka bidaltzea zen, azkenik aplikazio mugikorrean bistaratzeko. Lehenengo koska *OpenCV* Androiden erabiltzean topatu genuen, *OpenCV* ikusmen artifizialeko liburutegi bat izanik asko handitzen baitzuen aplikazioaren tamaina. Ondorioz, *web_video_server* izeneko ROSeko nodoa erabiltzea erabaki zen [10]. Honek irudien *topic*-ak irakurri eta *http* protokoloa erabiliz irudi-korronte bat sortzen baitu, gure proiekturako oso egokia izanik.

Nodo honek zenbait aukera eskaintzen ditu:

- **Irudiaren tamaina:** Parametro bezala jaso nahi den irudi tamaina zehaztu daiteke. Gure kasuan erabilgarria izango da irudia mugikor ezberdinen tamainetara egokitzeko.
- **Kalitatea:** Hau erabiliz irudiaren kalitatea aldatzeko gai izango gara. Egokia izan daiteke irudiaren kalitatea egokitzea robota konektatuta dagoen sarearen arabera.
- **Formatua:** Nodo honek sortzen duen irudi-korrontea honako formatu hauetan eska daiteke: *mjpeg* eta *vp8*. Gure kasuan *mjpeg* erabiliko da.

Inplementazioa

Aurretik aipatu bezala, Teknibot robotak *Kinect* kamera bat du eta hau erabiliko da Android aplikazioan bistaratuko diren irudiak eskuratzeko. Kamera arruntaz gain infragorri sentsorea ere erabiltzen du. Hau hegaldi denboraren menpeko sentsorea izanik, argi uhinak bidali eta iristen diren arteko denborak erabilita sakontasun mapa eraikitzen du [12]. Irudi horiek zenbait *topic*-etan publikatzen dira eta hauek dira aplikaziora bidaltzeko erabiliko direnak:

- **camera/rgb/image_color:** Kamera arruntaren RGB irudia.
- **camera/rgb/image_mono:** Kamera arruntaren zuri-beltz irudia.
- **camera/depth_registered/image:** Infragorriak sortutako sakonera irudia.

5.1.4 Komando bidezko eskaerak prozesatzeko zerbitzaria

Diseinua

Azken zerbitzari hau ere ROSeko nodo bat izango da eta Android aplikaziotik egindako komando bidezko eskaerak kudeatuko ditu. Honek ere TCP zerbitzari bat izango du eta bertatik bidali eta jasoko dira komandoak. Aurrekoen antzera, konkurrente egitea erabaki da erabiltzaile bat baino gehiago zerbitzatu ahal izateko aldi berean.

Honek jasotzen dituen eskaerek 5.3 taulan adierazitako protokoloari jarraitzen diote. Zerbitzariak erantzuteko beste protokolo bat erabiliko du, 5.4 eta 5.5 tauletan ikus daitekeena.

Eskari mota	**Komandoa**
Gida moduko helburuen lista eskatu	GIA_LIST
Maparen informazioa eskatu	GET_MAP_INFO
Maparen irudia eskatu	GET_MAP_IMAGE
Gida modura sartzeko baimen eskaera	CAN_ACCESS_GIA_MODE
Gida modutik ateratzeko baimen eskaera	CAN_EXIT_GIA_MODE
Gida moduko helburu batera joateko eskaera	GOTO
Gida moduko helburu batera joatea eten	CANCEL_GOTO
Robota dagoen zonaldea eskatu	GET_ZONE_ROBOT
Robotaren egoera eskatu	GUIDE_STATE
Robota sarean eskuragarri dagoela ikusi	HELLO

5.3 Taula: Komando bidezko zerbitzariaren eskaeren protokoloa.

Erantzun komandoak robotaren kontrola hartzeko baimena eskatzeko eta robotaren egoera kontsultatzeko komandoen erantzun gisa erabiliko dira soilik, beste eskaeretan, berriz, informazioa zuzenean bidaliko da.

Hona hemen azaldu diren eskari bakoitzeko zerbitzariak egin beharrekoa:

- **GIA_LIST:** Komando hau jasotzerakoan robotak Gida modurako aurredefinituak dituen helburuen zerrenda bidali beharko du aplikazioan bistaratzeko. Hau *xml* formatuko fitxategi bat izango da eta 5.2 irudian ikus daitekeen formatuari jarraitzen dio.

```
<locations>
  <location name="Recepción"  x="1.715" y="-2.45" yaw="3.14"/>
  <location name="Máquina de Café"  x="15.82" y="7.00" yaw="3.14"/>
  <location name="Servicios"  x="14.525" y="9.24" yaw="3.14"/>
  <location name="Auditorio"  x="15.015" y="11.48" yaw="1.57"/>
  <location name="Lurra 1-2"  x="16.415" y="-0.455" yaw="0.0"/>
  <location name="Ura 1-2"  x="17.08" y="8.19" yaw="0.0"/>
</locations>

```

5.2 Irudia: Helburu zerrenda duen fitxategiaren egitura.

- **GET_MAP_INFO:** Komando hau robota kokatuta dagoen ingurunearen maparen informazioa eskatzeko erabiliko da. Hori *yaml* formatuko fitxategi batean dago definituta. Bertan, 5.3 irudian ikus daitekeen bezala, honako datu hauek aurki ditzakegu:

- *image:* Aldagai honek mapa gordetzen duen irudi-fitxategiaren izena adierazten du.
- *resolution:* Maparen bereizmena adieraziko digu. Aurretik azaldu diren *amcl_pose* eta *particlecloud topic*-etatik eskuratutako informazioa metrotan baitago.
- *origin:* Maparen irudi-fitxategiaren jatorria non dagoen metrotan adierazita. Irudikatzerakoan puntu hori maparen (0,0)-tzat hartuko da.

Nahiz eta fitxategi horretan informazio gehiago egon, soilik azaldutako aldagaiak erabiliko dira. Fitxategi horretatik lortutako informazioa *CR* karakterearekin amaitutako buffer batean bidaliko da, atal bakoitza koma ikurraz banatuta.

```
image: Tknbt_taller_08_03_2017_gmapping_amcl.pgm
resolution: 0.050000
origin: [-95.615000, -105.810000, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196

```

5.3 Irudia: Maparen informazioa duen fitxategiaren egitura.

- **GET_MAP_IMAGE:** Komando hau maparen irudi-fitxategia eskatzeko jasoko da. Irudi hau *png* formatuan egongo da. Lehenik eta behin honen tamaina bidaliko da bezeroak jakin dezan zenbat *byte* irakurri behar dituen socket-etik. Azkenik, irudi-fitxategia bidaliko da *bytez-byte* buffer tamaina konkretu bat erabiliz.
- **CAN_ACCESS_GIA_MODE eta CAN_EXIT_GIA_MODE:** Komando hauekin bezeroak gida modura sartzeko eta bertatik irteteko baimena duen ala ez eskatuko du. Zerbitzariak erantzuna ROSeko zerbitzu batetik eskuratuko du: *guide_mode_access* zerbitzua. Honek erabiltzen duen mezu mota *std_srvs* paketeko *SetBool* motakoa da eta honako egitura dauka:
  - *request:* Atal honetan *data* izeneko aldagai boolear bat dauka. Eskaera egiterako orduan hau *true* balioarekin bidaliko dugu zerbitzua erabili nahi dugula eskatzeko eta, *false* balioarekin, berriz, zerbitzua utzi nahi dugula adierazteko.
  - *response:* Zerbitzuaren erantzuna izango den atal honetan, *success* izeneko aldagai boolearra erabiliko dugu erantzuna ebaluatzeko. Zerbitzuak *true* itzultzen badu, baimena daukagula esan nahiko du eskatu dugun ekintza egiteko. *False* itzuliz gero, berriz, baimenik ez daukagula.

Jasotako emaitzaren arabera 5.4 taulan adierazitako erantzuna bidaliko da.

Erantzun mota	**Komandoa**
Gida modura sartzeko baimena eman	OK_ACCESS_GIA_MODE
Gida modura sartzeko baimenik ez eman	NO_ACCESS_GIA_MODE
Gida modutik irteteko baimena eman	OK_EXIT_GIA_MODE
Gida modutik irteteko baimena ez eman	NO_EXIT_GIA_MODE

5.4 Taula: Robotaren kontrola eskuratzeko erantzunen protokoloa.

- **GOTO:** Bezeroak komando hau erabiliko du *GIA_LIST* komandoa bidali ondoren jasotako helburu-listako leku batera joateko adierazi bada edo baita ahots komando bidez agindu hori eman bada. Komando honekin batera helburu izen bat jasoko da. Robotari helburu horretara joan behar dela adierazteko *guide_destination* izeneko *topic*-a erabili beharko dugu. *Topic* honek erabiltzen duen mezu mota *std_msgs* paketeko *String* motakoa da. Ondorioz, jasotako helburuaren izena mota horretako mezu batean sartu eta publikatuko dugu.

- **CANCEL_GOTO:** Komando hau gida moduan helburu batera bidean doanean soilik jaso dezake zerbitzariak eta gida lana bertan behera uzteko erabiliko da. Jaso orduko, *cancel_current_guide_destination topic*-ean *std_msgs* paketeko *Empty* motako mezu bat publikatu beharko du. Mezu mota honetan ez da barneko aldagairik bete behar, soilik sortu eta bidali.

- **GET_ZONE_ROBOT:** Robota kokatuta dagoen inguruneak zenbait zona eduki ditzake, adibidez solairu edo gela ezberdinak. Hori kontsultatzeko erabiliko du bezeroak komando hau. *get_zone_robot* izeneko zerbitzua erabiliko da kontsulta hori gauzatzeko. Honek, *localization_manager* paketeko *GetZoneRobot* izeneko mezu mota erabiltzen du eta honako egitura du:

  – *request:* Atal honetan ez da ezer zehaztu behar.

  – *response:* Zerbitzuak, *string* motako *zone* aldagaian itzuliko du emaitza.

  Zona edukita, *CR* karakterearekin bukatutako buffer batean bidaliko da. Honen erabilera LER testu analizatzailera bideratuta dago. Izan ere, analizatzaile honek robotaren zona ezagutu behar du, analizatu beharreko eskaerei erantzun egokia eman ahal izateko.

- **GUIDE_STATE:** Komando hau, robota helburu batera gida lanak egiten dagoen bitartean jasoko du zerbitzariak. Hau, robotaren egoera kontsultatzeko erabiliko da, hala nola, helburura iritsi den edo errorerik gertatu den ala ez galdetzeko. Informazio hori *guide_state topic*-ean suskribatuz lortuko da eta honek darabilen mezu mota *std_msgs* paketeko *Byte* mota da. Honako balioak espero dira *topic* honetan:
  - **3:** Balio honek helburura iritsi dela adieraziko du. Hau jasotzen den momentuan dagokion boolear batean *true* balioa jarriko du.
  - **5:** Gida lanak egiten erroreren bat gertatu dela adieraziko du. Hau jasotzen den momentuan dagokion boolear batean *true* balioa jarriko du.
  - **Beste edozein balio:** Balio hauei ez zaie kasurik egingo.

  Beraz, komando hau jasotzen den momentuan, aipatutako boolear horietan egingo da kontsulta. Helburu berri bat definitzen den bakoitzean hauek *false* balioarekin hasieratzen dira. Eskaerei erantzuteko 5.5 taulan adierazitako protokoloari jarraitzen zaio.

Erantzun mota	**Komandoa**
Gida moduan arazorik ez	GUIDE_OK
Gida moduan errorea	GUIDE_ERROR
Gida moduan adierazitako helburura iritsi	GUIDE_GOAL_REACHED

5.5 Taula: Robotaren egoerari buruzko erantzunen protokoloa.

- **HELLO:** Komando hau zerbitzariarekin konexioa existitzen den jakiteko bidaliko du bezeroak. Komando zerbitzariak hau jasoz gero, komando berarekin erantzungo du.

Inplementazioa

Diseinuan azaldu den bezala, zenbait fitxategi izango ditu zerbitzari honek eskuartean. Hauen path-ak kanpotik konfiguragarriak egiteko asmoz, ROSeko parametro zerbitzaria erabili da. Horretarako *yaml* formatua duen fitxategi bat sortu da zeinak parametro hauek dituen:

- **map_image_path:** Maparen irudi-fitxategiaren kokalekua.
- **map_yaml_path:** Maparen datuak dituen *yaml* fitxategiaren kokalekua.

- **goals_file_path:** Gida moduko helburuak *xml* formatuan zerrendatuta dituen fitxategiaren kokalekua.

Ondorioz, parametro horiek kudeatzeko klase bat sortu da, parametroak objektu soil batean eta ez banan-banan pasatu ahal izateko. Nodo honen sasikodea 4 algoritmoan ikusi daiteke.

Algoritmoa 4 Komando eskaerak kudeatzeko nodoaren sasikodea.

```
1:  TCP zerbitzariaren objetu globala sortu portu zenbakia pasata.
2:  function GETZONEROBOT()          ▷ Funtzio hau GET_ZONE_ROBOT komandoa jasotzean exekutatuko da.
3:      GetZoneRobot motako zerbitzu-mezua sortu eta zerbitzuari bidali.
4:      Zerbitzuaren erantzuna jaso.
5:      return zerbitzutik jasotako zona.
6:  end function
7:  function CANENTERGIAMODE()       ▷ Funtzio hau CAN_ENTER_GIA_MODE komandoa jasotzean exekutatuko da.
8:      SetBool motako mezua sortu.
9:      request atalean true jarri.
10:     Zerbitzuari eskaera bidali.
11:     Zerbitzuari eskaera bidali.
12:     Zerbitzutik erantzuna jaso.
13:     Erantzuna itzuli (true/false).
14: end function
15: procedure GUIDESTATECALLBACK(Byte mezua)
16:     if mezua==3 then
17:         Bezeroari bidali helburura iritsi dela.
18:     else if mezua==5 then
19:         Bezeroari bidali errore bat gertatu dela.
20:     end if
21: end procedure
22: procedure NAGUSIA()
23:     ROS hasieratu.
24:     NodeHandle objektua sortu.
25:     Kanpo parametroak jaso (fitxategi path-ak).
26:     guia_mode_access zerbitzuaren bezeroa sortu.
27:     guide_state topic-era suskribatu, guideStateCallback funtzioa pasata.
28:     get_zone_robot zerbitzuaren bezeroa sortu.
29: end procedure
```

TCP zerbitzariari dagokionez, poseen zerbitzariaren antzeko itxura dauka, baina konexio bakoitzak duen metodoan etengabe aldagaien balioak bidaltzen egon beharrean, eskaeren

zain geratzen da. Ondoren, jasotako komando bakoitzeko *process_command* metodoari deitzen dio hau erauzi eta dagokion prozesaketa egiteko. Metodo horien sasikodea 5 algoritmoan ikus daiteke.

Algoritmoa 5 Komando eskaerak kudeatzeko TCP zerbitzariaren sasikodea.

```
1:  procedure KONEXIO_BERRIA()
2:      while 1 do
3:          Zain egon socket-etik lerro bat irakurri arte.
4:          process_command(irakurritako_lerroa).
5:      end while
6:      Socket-a itxi.
7:  end procedure
8:  procedure PROCESS_COMMAND(command)
9:      Irakurritako lerrotik komandoa eta datuak banatu.
10:     if komandoa==GIA_LIST then
11:         Komandoa erantzuteko metodoa deitu.
12:     end if
13:     ...
14: end procedure
```

5.2 Android aplikazioaren garapena: Bezeroa

Atal honen helburu nagusia robotetik jasotako informazioa bistaratu eta robotarekin elkarrekintza lortzea izango da, komando bidezko eskaerak eginez. Hau, eskakizun analisian aipatutako ideiak jarraituz egin beharko da.

5.2.1 TCP eta UDP bezeroak

Robotaren aldeko garapenean azaldu den bezala, robotaren posea eta komando bidezko elkarrekintza TCP bidez egingo dira modu independentean. Beraz, Android aplikazioak bezero bat izan beharko du zerbitzari bakoitzarekin komunikatzeko. Gainera robota sarean automatikoki aurkitzeko UDP zerbitzuarekin komunikatu beharko den bezeroa izan beharko du. Bakoitza *thread* edo hari ezberdin batean exekutatuko da exekuzio konkurrentea baimenduz.

- **Posea jasotzeko bezeroa:** Bezero hau etengabe egongo da socket-ean entzuten. Jasotzen duen mezu bakoitzeko, zein motatakoa den erabakiko du eta datuak behar

bezala prozesatuko ditu. Hau edozein momentutan gelditu ahal izateko begizta nagusian boolear bat erabiltzen da. Honen egitura 6 algoritmoan azaldutakoa da.

Algoritmoa 6 Poseak jasotzeko bezeroaren sasikodea.

```
1:  procedure NAGUSIA()
2:      Socket-a sortu IP helbidea eta portu zenbakia emanda.
3:      Socket-etik input eta output stream-ak lortu.
4:      while gelditzeBool==true do
5:          Zain egon socket-etik lerro bat irakurri arte.
6:          process_command(irakurritako_lerroa).
7:      end while
8:      Socket-a itxi.
9:  end procedure
10: procedure PROCESS_COMMAND(command)
11:     Irakurritako lerrotik komandoa eta datuak banatu.
12:     if komandoa==POS then
13:         Poseak kudeatu.
14:     end if
15:     ...
16: end procedure
```

- **Komandoak bidali/jasotzeko bezeroa:** Honen egitura aurrekoaren guztiz ezberdina da eta honako ideiari jarraitzen dio: Komando bat bidaltzen den bakoitzeko, jarraian socket-ean entzuten geratuko da erantzun osoa jaso arte. Horrela erantzunak era egokian tratatzen direla ziurtatuko dugu. Hau egiteko Androidek eskaintzen duen *AsyncTask* klasea erabiliko dugu; Klase honek bigarren planoko eragiketak egitea baimenduko digu *doInBackGround* metodoa erabiliz eta, gainera, klase hau heredatzen duten klase guztietako metodoak seriean ala paraleloan exekutatzeko aukera emango digu. Gure kasuan interesgarria izango da seriean exekutatzea socket beretik idatzi eta irakurtzeko ordena mantendu ahal izateko. *AsyncTask* klasearen adibide bat 5.4 irudian ikus daiteke.

  Beraz, bezero hau ere *thread* bat izango da eta soilik begizta infinitu bat izango da, konexioa martxan egotea ahalbidetzen duena. Bezeroa objektu bat izanik, parametro bezala pasako zaie aurretik azaldu diren komandoak bidaltzeko erabiliko diren metodoei. Honen sasikodea 7 algoritmoan ikus daiteke.

- **Robotaren bilaketa prozesuko UDP bezeroa:** Azken bezero honek duen egitura guztiz arrunta da, bere sasikodea 8 algoritmoan adierazitakoa izanik.

```
1  public static class sendHello extends AsyncTask<clientParam,Void,Boolean>{
2
3          public Boolean doInBackground(clientParam ... params){
4              ClientGia clg=params[0].clg;
5              BufferedReader reader=clg.getReader();
6              if(clg.isRunning()){
7                  if(reader!=null){
8                      if(!clg.sendMessage(command_list.get(COMMAND_HELLO)+" "))return false;
9                      try {
10                         String response = reader.readLine();
11                         if(response==null){
12                             clg.stopClient();
13                             return false;
14                         }
15                         if(response.compareTo(command_list.get(COMMAND_HELLO))==0)return true;
16                         else return false;
17                     } catch(java.io.IOException e){
18
19                     }
20                 }
21
22
23             }
24             return false;
25         }
26     }
27
```

5.4 Irudia: AsyncTask klasearen erabileraren adibidea.

Algoritmoa 7 TCP komandoak jasotzeko bezeroa.

1: **procedure** NAGUSIA()
2: Socket-a sortu IP helbidea eta portu zenbakia emanda.
3: Socket-etik input eta output stream-ak lortu.
4: **while** gelditzeBool==true **do**
5: **end while**
6: Socket-a itxi.
7: **end procedure**

Algoritmoa 8 UDP bezeroaren sasikodea.

```
1:  procedure NAGUSIA()
2:      Socket-a sortu parametrotzat portu zenbakia emanda.
3:      SARELOKALEKOIGORPENHELBIDEALORTU()
4:      Datagrama pakete bat sortu, HELLO mezua, igorpen helbidea eta portua pasata.
5:      i=0.
6:      while i < SAIAKERA_KOP do
7:          Socket-etik datagrama paketea bidali.
8:          Erantzuna jaso, denbora bat pasata ez bada iritsi, jarraitu.
9:          if erantzuna==OK then
10:             P2-ra pasa, erantzunetik lortutako IPa pasata.
11:         end if
12:         i++.
13:     end while
14:     Socket-a itxi.
15: end procedure
16: procedure SARELOKALEKOIGORPENHELBIDEALORTU()
17:     DHCP zerbitzutik IPa eta maskara eskuratu.
18:     IP helbidea eta maskararen arteko bit mailako eragiketekin igorpen helbidea lortu.
19: end procedure
```

5.2.2 P1: Robota sare lokalean bilatzeko pantaila

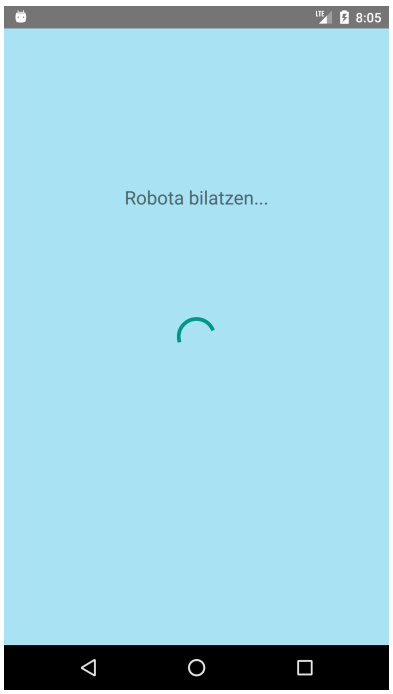
Diseinua

Esan bezala, aplikazioa hasi bezain laster pantaila hau azalduko zaigu sare lokalean robota bilatzen ari dela pantailaratuz. Bertan karga ikur bat azalduko da 5.5 irudian ikus daitekeen bezala harik eta hau aurkitu edo bilatzeko ahalegin kopuru maximoa gainditu arte.

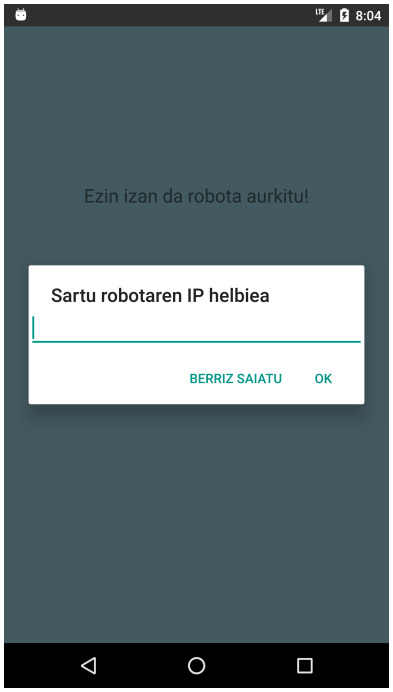
Robota ez badu aurkitzen, 5.6 irudian ikus daitekeen elkarrizketa-leihoa irekiko da bertan honen IP helbidea sartzeko aukera emanez. Erabiltzaile arruntak ziurrenik ez du helbidea jakingo eta horregatik bilaketa prozesua berriz hasteko aukera ematen da.

Inplementazioa

P1 pantailaren funtsa robota sare lokalean bilatzea izanik, *DHCP* bezeroari Android gailuaren IP helbidea eta sarearen maskara eskatzen zaizkio. Bi datu horiek erabiliz eta bit mailako zenbait eragiketa egin ondoren sare lokaleko igorpen helbidea lortzea izango da helburua. Horrela sare lokalean UDP igorpen bat egingo da portu zehatz batera, robotean



5.5 Irudia: P1 pantaila, robotaren bila dagoen kasua.



5.6 Irudia: P1 pantaila, robota aurkitu ez duen kasua.

bilaketa prozesu hau aurrera eramateko duen UDP zerbitzariak esleitua duen portura hain zuzen ere. Ondoren, erantzunaren zain geratuko da ezarrita izango duen hutsarte bat pasa arte. Prozesu hau hiru aldiz errepikatuko da robota aurkitzen ez den bitartean, aurkituz gero pantaila nagusira pasako da eta, bestela, berriz ere 5.6 irudiko elkarrizketa-leihoa azalduko da.

Mezu trukea 5.1 taulan azaldutakoa da.

5.2.3 P2: Pantaila nagusia

Diseinua

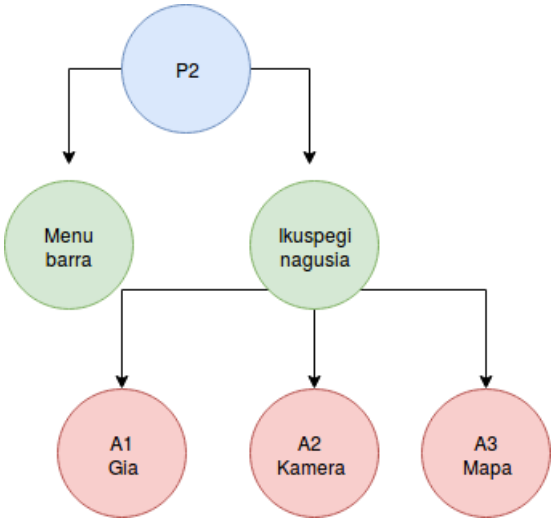
P2 pantaila izango da *activity* nagusia deituko duguna. Hau aplikazioaren hiru erabilpen kasuak bistaratzeaz arduratuko da eta honako elementuez osatua dago:

- **Menu-barra:** Menu hau pantailaren azpialdean kokatuko da eta dituen botoien bidez funtzionalitatez aldatzeko aukera emango dio erabiltzaileari.
- **Ikuspegi nagusia:** Pantailaren gainontzekoa ikuspegi honek beteko du. Honen ezaugarri nagusia, zenbait azpi-ikuspegi kudeatzeko gaitasuna da. Horrela, nahiz menu barrako botoiak erabiliz, nahiz hatzen mugimendua erabiliz azpi-ikuspegien artean nabigatzea posible izango da.

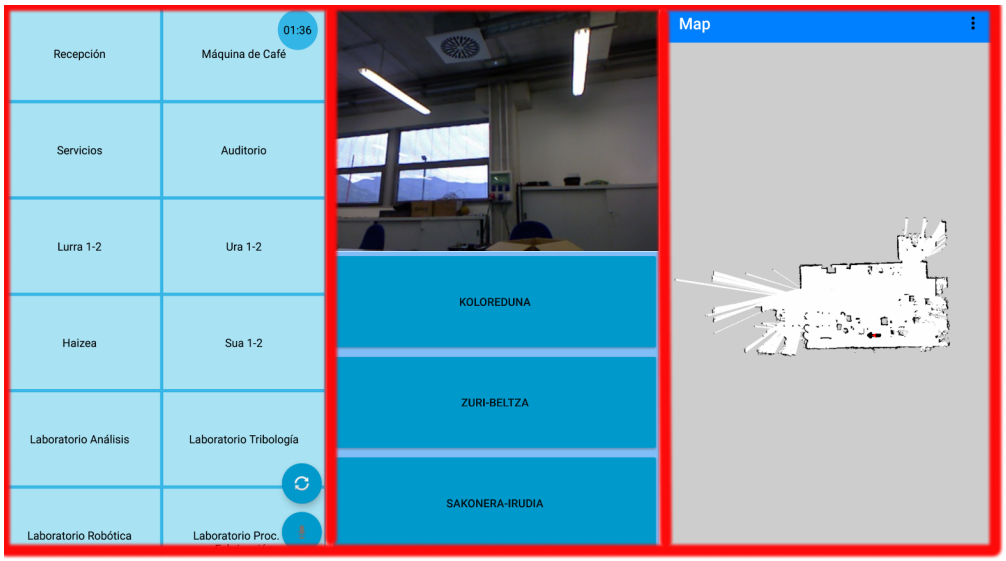
P2-ren egitura 5.7 irudian azaltzen da. Esan bezala, *ikuspegi-nagusia* erabilpen kasu bakoitzeko azpi-ikuspegi batez eratua dago. Kasu honetan *gida*, *kamera* eta *mapa* moduek osatuko dute. Era berean, horietako edozeinek exekuzioa aurrera doan heinean bere interfazea aldatzea posible izango du, baina ikuspegi nagusian izango duen kokapena beti berdina izango da. Ikus bedi 5.8 irudia egitura hobeto ulertzeko. Aipatutako irudian gorriz inguratuko azpi-ikuspegi bat soilik bistaratuko da aldi berean eta horien artean nabigatzea posible izango da lehen aipatu bezala.

Inplementazioa

P2 pantailaren eginkizun nagusia hiru erabilpen kasuak kudeatzea den arren, beste zenbait ezinbesteko ataza ere kudeatzen ditu. Alde batetik, *P1* pantailan eskuratutako IP helbidea erabiliz TCP bezeroak hasieratzen ditu. Poseak jasotzeko bezeroa eta komando bidezko



5.7 Irudia: P2 pantailaren egituraren eskema.



5.8 Irudia: P2-ko ikuspegi nagusiaren egitura; gorriz hau osatzen duten elementuak.

elkarrekintza lortzeko bezeroak martxan jarriko ditu portu ezberdinak erabiliz hain zuzen ere. Bestalde, zenbait metodo eskaintzen ditu azpi-ikuspegiak zerbitzariekin komunikatu ahal izateko.

5.2.4 A1: Gida ikuspegia

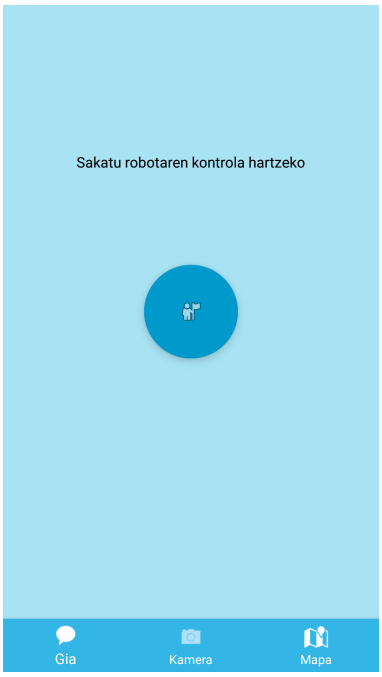
Lehenengo azpi-ikuspegi honek robotari gida lanak egiteko aginduak bidali ahal izatea ahalbidetuko dio erabiltzaileari. Kontuan hartu beharko da erabiltzaile bakar batek izango duela robotaren kontrola momentu bakoitzean, bestela segurtasun arazo handiak gerta litezkeelako. Esklusibotasun hori lortzeko, baimena eskatu beharko zaio zerbitzariari botoi bat sakatuz. Baimena eskuratuz gero kontrol-modura sartuko da aplikazioa eta honek kontrola galdu arte zerbitzariak ez dio inori baimenik emango robota kontrolatzeko. Esklusibotasun hori denbora batez dago mugatuta eta denbora amaitzean zerbitzariari adieraziko zaio libre geratzen dela. Aplikazioa ixten denean ere robotari bidaliko zaio libre geratu dela adierazteko komandoa.

Diseinua

Azpi-ikuspegi honek dituen erabilpen moduak honakoak dira:

- **Kontrol gabeko modua:** Modu honetan erabiltzaileak ez du robotaren kontrola izango baina hau eskatzeko aukera dauka.
- **Kontrol-modua:** Robotaren kontrola hartzeko baimena eskuratzen denean sartuko da aplikazioa modu honetan. Bertan, robotari aginduak bidaltzeko aukera dago, bai interfaze grafikoaren bitartez, eta bai ahotsez.
- **Bidaia-modua:** Helburu batera joateko agindua bidali ondoren sartzen da aplikazioa modu honetan. Bertan, nabigazioa bertan behera laga ahal izango da botoi bat erabiliz.

Hasierako modu honi kontrol gabeko modua deituko zaio. Robotaren kontrola hartu nahi dela adierazteko 5.9 irudian ikus daitekeen ikuspegiak duen botoia sakatu beharko da. Robota kontrolatzeko baimena eskuratzen bada, aplikazioa kontrol-modura pasako da. Kontrol-moduko interfazeak 5.10 irudiko osagaiak ditu.



5.9 Irudia: Robotaren kontrola eskatzeko ikuspegia.

- **Helburu-zerrenda:** Robotak aurredefinituak dituen helburuen-zerrenda erakutsiko da. Erabiltzaileak hauetako edozein hautatu ahal izango du, bakoitzaren gainean klik eginez.
- **Ahots botoia:** Botoi hau sakatuta aginduak ahots bidez emateko aukera dago.
- **Birkarga botoia:** Helburu-zerrenda eskuratzen arazorik egon bada edo helburu berriren bat gehitu bada, hau birkargatzeko erabili ahal izango da.
- **Denbora markagailua:** Erabiltzaile bakoitzak robotaren esklusibotasuna denbora mugatu batez soilik izango du eta markagailu honetan denbora hori amaitzeko geratzen den denbora adierazten da.

Modu honetan, zenbait kasu bereiziko ditugu:

- **Helburu-zerrendako elementu bat hautatu:** Hau eginez gero, robotari helburu horretara joateko agindua bidaliko zaio eta interfazea bidaia-modura pasako da. Bertan helburuaren izena pantailaratuko da eta gida lana bertan behera uzteko aukera ematen duen botoi bat agertuko da, 5.11 irudian ikus daitekeen bezala. Helburura iristen denean berriz ere kontrol-modura pasako da. Bitartean erroreren bat gertatzen bada ere, pantailaratu eta kontrol-modura itzuliko da.

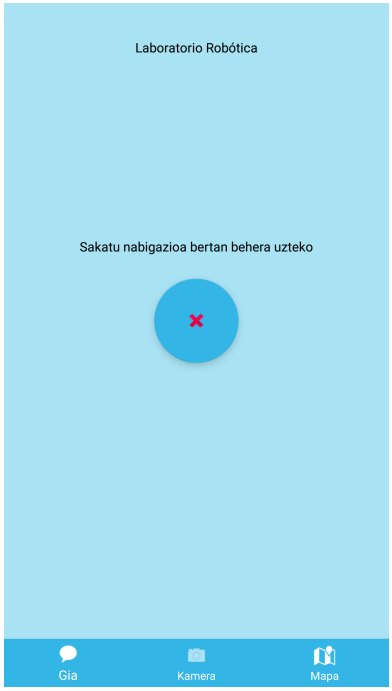
5.10 Irudia: Kontrol moduaren ikuspegia.

- **Ahots bidez agindu bat esan:** Ahots botoia sakatuz gero, elkarrizketa-leiho bat irekiko da 5.12 irudian ikus daitekeenez. Bertan, *gaztelaniazko* galderak soilik ulertuko dira, *LER* tresna hizkuntza horretarako soilik baitago prestatua. Erabiltzaileak botatako agindua ontzat ematen bada beste elkarrizketa-leiho bat irekiko da ulertu dituen aukera ezberdinen artean konfirmazioa eskatzeko. Bestela, agindua ulertu ez duela adieraziko du. Ikus 5.13 irudia.

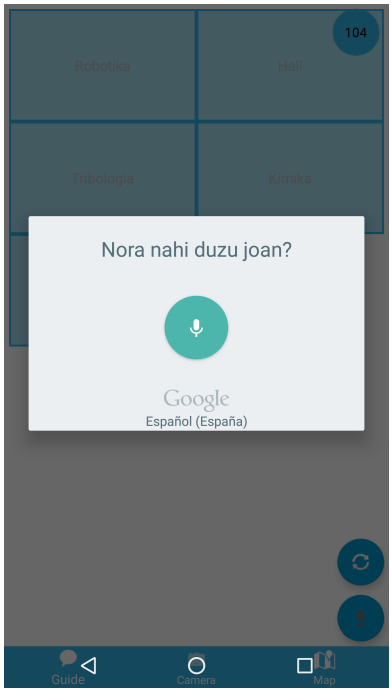
- **Kontrol denbora amaitu:** Kasu honetan kontrol gabeko modura pasako da aplikazioa eta gida lanak martxan badaude, bertan behera lagako ditu, robotari dagokion komandoa bidaliz. Horrela, robotaren kontrolaren esklusibotasuna amaituko da eta berriz kontrola hartu nahi izanez gero, berriz ere baimena eskatu beharko zaio zerbitzariari.

Inplementazioa

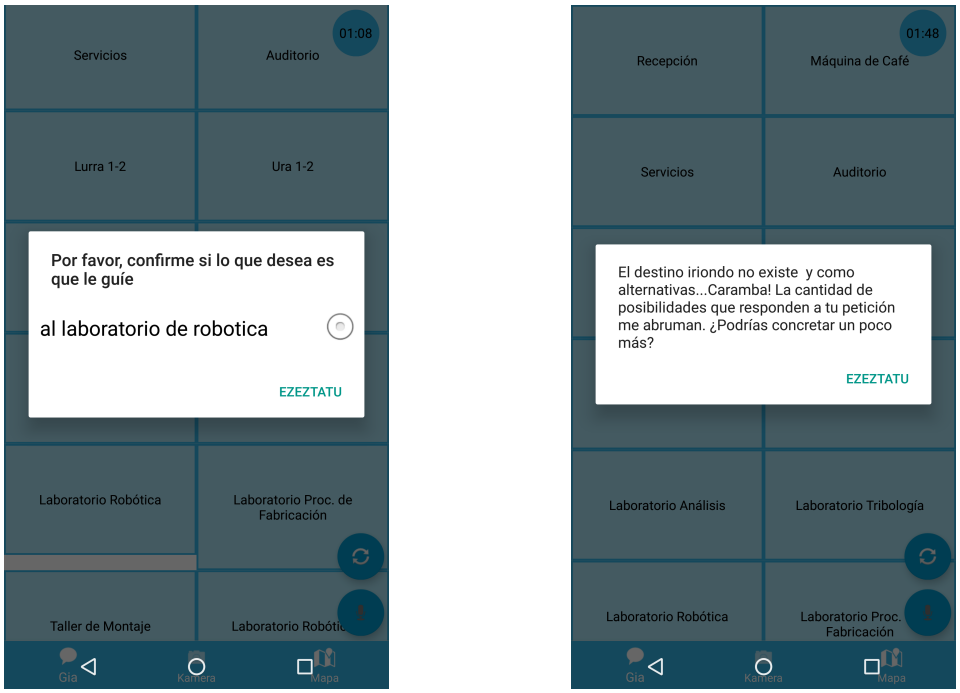
Erabilpen kasu honen inplementazioaren zati garrantzitsuena robotari komando bidezko eskaerak egitean eta erantzunak kudeatzean oinarritzen da, izan ere, hau ezinbestekoa izango da kontrolaren esklusibotasuna mantendu ahal izateko. Hau 5.3, 5.4 eta 5.5 tau-



5.11 Irudia: Bidaia-moduaren ikuspegia.



5.12 Irudia: Ahots bidez robota agintzeko ikuspegia.



(a) Ulertutako aukeren baieztapen zerrenda. **(b)** Ulertu gabeko ahots komandoa.

5.13 Irudia: Ahots komandoen erantzunak.

letan azaldutako protokoloari jarraituz egingo da. Bestalde, ahots bidezko aginduak interpretatzeko, ahotsa testu bihurtu ondoren, LER zerbitzua erabiliko da testuaren analisi semantikoa egiteko. Atal bakoitza bere aldetik azalduko da:

- **Esklusibotasunaren kontrola:** Esklusibotasunaren kontrola, bereziki robotaren aldeko zerbitzarietan kudeatzen da, aplikazioan soilik komando bidezko kontsultak egingo direlarik. Gida moduko kontrol-modura sartzeko lehendabizi baimena eskatuko zaio zerbitzariari horretarako komando bat bidaliz. Beraz, soilik erantzuna baiezkoa izan bada sartu ahal izango gara modu honetara. Baimena eskuratuz gero, atzeranzko kronometro bat jarriko da martxan; Hau bigarren planoan exekutatuko da eta erabilgarria izango da denbora amaitzean *callback* funtzio bat exekutatzeko aukera ematen baitu. Denbora amaitzean zerbitzariari zerbitzua uzteko baimen eskaera bidaliko zaio, eta baiezkoa bada, berriz ere kontrol gabeko modura itzuliko da. Gida lanik egiten badago bertan behera uzteko komandoa bidaliko da.

- **Interfaze bidezko aginduak:** Bai helburu batera gida lanak egiten joateko, bai gida lanak bertan behera uzteko, komando bidezko eskaera arruntak egiten dira. Hauetan ez da zerbitzariaren konfirmazioa espero. Teknibot bidaia-moduan dagoenean, aplikazioak segundoero komando bidezko eskaera bat egingo du robotaren

egoera zein den jakiteko. Eskaera hauek erantzuna jasoko dute eta horrela robota helburura iritsi dela edo erroreren bat gertatu dela jakin ahal izango da.

- **Ahots aginduak:** Ahots aginduak testu bihurtzeko Googlek eskaintzen duen *Speech to text* zerbitzua erabiltzen da. Behin ahots agindua jasota, ez badu ezer ulertu agindua errepikatzeko eskatzen du. Bestela, erantzuna jaso eta aipatutako LER zerbitzura bidaltzen du testua, honen analisi semantikoa gauzatzeko. Testuaz gain, robotaren zona ere bidali behar zaio zerbitzu honi, izan ere, beharrezkoa izango baita zerbitzuak eman beharreko erantzuna erabakitzeko. Egindako eskaera zona bereko helburu batera joatea bada posible izango du erabiltzailea bertara gidatzea, baina, bestela helbururaino ezingo duela gidatu adierazi beharko dio, handik gertuen dagoen zona bereko posiziora baizik. Zerbitzuari, JSON formatu konkretu bati jarraituz, *http* protokoloko POST metodoa erabiliz eskaera bidaltzen zaio. Erantzuna ere formatu berean jasotzen da. Zerbitzu hau Tekniboten gida moduari lotua dagoenez, soilik aurredefinituak dauden tokietara joateko aginduak ematen ditu ontzat. Emaitzatzat, ulertutakoaren arabera zenbait tokiren izenak eta hauen koordenatuak bueltatzen ditu, probabilitatearen arabera ordenatuak. Azkenik, informazio hori interpretatu eta beharrezko informazioa pantailaratzen da diseinuan azaldu bezala.

5.2.5 A2: Kamera ikuspegia

Erabilpen kasu honetan robotak duen kameraren irudiak bistaratzen dira, erabiltzaileari irudi mota aukeratzeko aukera emanez.

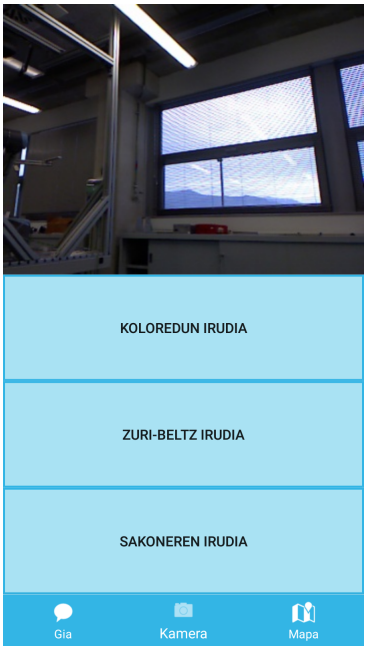
Diseinua

Azpi-ikuspegi hau diseinatzerako orduan, irudi zerbitzariak eskaintzen dituen irudien tamainaren proportzioa mantentzen da. Ondorioz, bi diseinu egin dira: Bat, 5.14a irudian ikus daitekeen bezala, mugikorra tente edo *portrait* moduan dagoenerako. Bestea, berriz, 5.14b irudian ikus daitekeen bezala, etzanda edo *landscape* moduan dagoenerako. Horrela pantailaren tamainara egokitutako irudi bat pantailaratzea lortzen da.

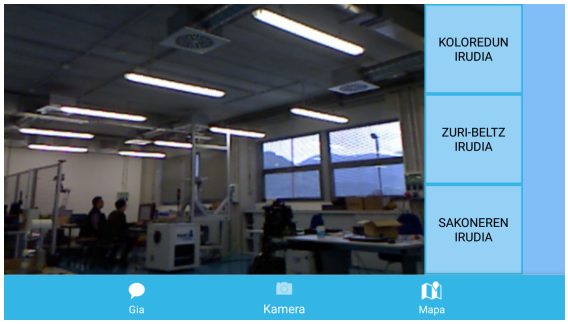
Interfazea osatzen duten elementuen egitura honakoa da:

- **Web ikuspegia:** Robotaren aldeko irudi zerbitzariak irudi korrontea bidaltzeko *http* protokoloa erabiltzen duenez, web orrialde sinple bat sortuko dugu *HTML5* erabiliz ikuspegi honetan irudia bistaratzeko.

- **Botoien ikuspegia:** Ikuspegi honetan irudiz aldatzea ahalbidetzen duten botoiak kokatuko dira.



(a) Portrait moduko diseinua.



(b) Landscape moduko diseinua.

5.14 Irudia: Kamera moduaren diseinuak.

Inplementazioa

Web ikuspegian irudia bistaratu ahal izateko HTML5-eko *img* elementua erabiltzen da *mjpeg* formatuko irudi korronteak bistaratzeko egokia delako. 5.1.3 atalean azaldu den bezala, irudiak tamaina ezberdinekin eskatu daitezke eta, horregatik, aukera hau erabiltzen da pantailaren tamainaren araberako irudiak eskatzeko. Esandako *img* atributuari, irudi iturria izango den zerbitzariaren URL helbidea pasatzen zaio eta honako itxura du:

$$http://SERVER\_IP:PORT/stream?topic=TOPIC\_NAME\&width=imageWidth\&height=imageHeight$$

Irudien zerbitzariari eskatu beharreko irudien tamaina kalkulatzeko honako parametro hauek hartzen dira kontuan:

- Zabalera eta altueraren arteko ratioa, beti berdina izango dena.

- Mugikorra dagoen modua: *portrait* edo *landscape*.
- Pantailaren tamaina.

Beraz, irudiaren tamaina ikuspegia hasieratzerakoan kalkulatuko da. Botoiei dagokienez, bakoitzak *topic* izen bat izango du atxikituta eta, beraz, sakatzen den momentuan dagokion *topic* izenarekin URL bat sortu eta honekin web ikuspegiari ezartzeko HTML5 orri bat sortuko du lehen azaldu bezala. Azkenik, web ikuspegiaren edukia aldatuko du sortutako berriarekin.

5.2.6 A3: Mapa ikuspegia

Izenean antzeman daitekeenez, azken erabilpen kasu honek robotaren lokalizazioa pantailaratzea du helburu.

Diseinua

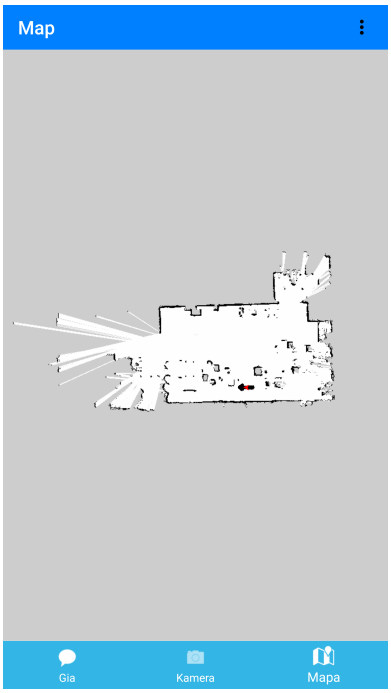
Azpi-ikuspegi honen egitura (5.15 irudian ikus daitekeen bezala) honako elementuez osatua dago:

- **Tresna-barra:** Pantailaren goialdean dago kokatuta eta Unity ikuspegiarekin elkarrekintza tresna-barra honetako aukeren bitartez egiteko pentsatua izan da. Etorkizunean Unity-n funtzionalitate gehiago inplementatuko balira, hauek kudeatzeko egokia izateko diseinatu da.
- **Unity ikuspegia:** Erabilpen kasu honen muina da eta bertan robotaren lokalizazioa bistaratzen da maparen gainean. Hatzak erabiliz, erabiltzaileak posible izango du maparen gainean *zoom* egitea, hau mugitzea eta biratzea.

Inplementazioa

Unity motor grafikoak adierazitako ingurunean honako objektu hauek txertatu dira:

- **Kamera:** Sortutako bi dimentsioko ingurunean txertatutako objektuak bistaratzeko beharrezkoa da. Honen mota *ortografikoa* da eta maparen gainean dago kokatua.



5.15 Irudia: Maparen ikuspegia.

- **Objektu nagusia:** Objektu honek ez dauka adierazpen grafikorik eta, beraz, ikusezina da. Kontuan hartuta Unity-n programatzeko *C#* script-ak objektuei atxikituta egon behar direla, hau erabiliko da hasieraketak egiteko.
- **Mapa:** Robotetik eskuratuko den maparen irudia. Irudiko pixel bakoitza Unity inguruneko unitate bat izango da.
- **Robota:** Objektu honek robotaren kokapena adierazten du eta honek duen irudia aurretik dago definituta. Hau maparen ume izango da eta, beraz, bere koordenatu sistema ingurunea izan ordez, mapa izango da.
- **Partikula multzoa:** Hau objektu multzo bat da eta hauekin robotaren estimatutako pose ezberdinak irudikatzen dira. Hauen irudia ere aurrez dago definituta eta robotaren antzera maparen ume izango dira.

Kamera eta objektu nagusia estatikoki daude definituak eta, beraz, Unity ikuspegia hasi bezain laster hauek ingurunean kokatuko dira. Kamera, ingurunearen jatorrian altuera zehatz batean kokatuko da eta objektu nagusiaren kokapena, berriz, ez da garrantzitsua izango, ikusezina baita. Gainerako objektuak dinamikoki sortu eta kokatuko dira exekuzioan zehar. Hauek irudikatu ahal izateko behar beharrezkoa da aplikazioak dagoeneko

maparen irudi-fitxategia eta datuak eskuratuta izatea. Horrela ez bada, ez da ezer irudikatuko pantailan.

Behin horiek edukita, lehendabizi mapa adieraziko duen objektua sortuko da. Hau *GameObject* bat izango da eta *SpriteRenderer* motako atributu bat gehituko zaio, bisualki irudi bat izan dezan. Ondoren, irudia fitxategitik kargatuko du testura bezala eta objektuari esleituko dio. Mapa adierazten duen objektua ingurunean kokatzeko, zerbitzaritik eskuratzen dira mapak duen *yaml* formatuko fitxategian definitutako datuak, hala nola, erresoluzioa, jatorria etab. 5.1.4 atalean aipatu bezala, *origin* atributuak maparen jatorria non dagoen adieraziko digu eta, beraz, hori erabiliko dugu maparen jatorria 2D ingurunearen jatorrian kokatzeko.

Hurrengo pausua robota irudikatzea izango da, hau ere *GameObject* bat izango da eta honi ere *SpriteRenderer* atributua gehituko zaio. Honek izango duen irudia Unity ingurunea sortzerakoan zehaztuko da eta, beraz, exekuzioan liburutegi batetik inportatuko du irudia. Hau, mapa adierazten duen objektuaren ume bezala jartzean, honen koordenatu sistemaren jatorria maparen beheko ezker ertza izatera pasako da. Guri robotaren koordenatu sistemaren jatorria maparen jatorriaren berdina izatea interesatuko zaigu eta, beraz, bertan kokatuko dugu. Zerbitzaritik jasotako poseak irudikatzeko, soilik metrotan jasotako balioak pixeletara pasatzea faltako zaigu, esan bezala gure kasuan Unity inguruneko unitate bat pixel baten baliokidea delako. Hau egin ahal izateko maparen datuetako *resolution* atributua erabili beharko dugu. Azkenik, jasotako poseetatik metrotan adierazitako puntuak hartu eta *resolution* atributuarekin zatituz pixeletara bihurtuko ditugu. Orientazioa, zerbitzarian pasako da radianetatik graduetara eta, beraz, dagoeneko graduetan jasoko dugu.

Poseen partikula multzoa irudikatu ahal izateko robotarekin jarraitutako pausu berberak jarraitzen dira. Multzo honen kopurua robotaren exekuzioan zehar aldakorra denez, beti, 500 irudikatzea erabaki da, gutxiago izanez gero bata bestearen gainean irudikatzen dira. Izan ere, uneoro irudi kopuru handi bat kargatzeak kostu handia du eta horregatik kopuru finko bat kargatuta izatea erabaki da.

Androidetik datuak bidali ahal izateko, zenbait funtzio definitu dira, posible baita Unityren funtzioei deitzea. Dinamikoki mugitu beharreko objektuak robota eta partikula multzoko elementuak dira, horregatik objektu hauetako bakoitzak duen script-ean posea ezartzeko metodo bat izango dute.

5.2.7 Hizkuntzak

Hizkuntzei dagokienez, aplikazio mugikorra hiru hizkuntzatan egongo da erabilgarri: Euskaraz, gaztelaniaz eta ingelesez. Hau erabiltzaile bakoitzaren mugikorrak ezarrita duen hizkuntzaren arabera aldatuko da automatikoki. Mugikorrean ezarritako hizkuntza hiru horietako bat ere ez bada, defektuzko hizkuntza ingelesa izango da.

5.3 Probak

Hasieran aplikazioa robotean probatzea pentsatua zegoen arren, ezinezkoa izan da hau gauzatzea proiektutik kanpo geratzen diren zenbait arrazoirengatik. Ondorioz, probak konputagailu lokalean egin behar izan dira. Horretarako robotean ROSeko *bag* fitxategi bat grabatu da zeinak *topic*-etan publikatutako guztia gordetzen duen. Arrazoi beragatik, gida modua ezin izan da probatu, soilik benetako robotean ikus daitekeelako funtzionatzen duen edo ez. Baina esan behar da, gida modua inplementatzeko IK4-TEKNIKER enpresan zehaztutako aurrebaldintzen gain egin dela lan, funtzionalitate honen inplementazioaren zati batzuk proiektutik kanpo geratzen baitziren.

Beraz, probak bi eratan egin dira: Alde batetik, IK4-TEKNIKERen, 3 kapituluan aipatutako AVDa eta konputagailu lokala erabiliz egin dira. Bestalde, etxeko sare lokala erabiliz, aplikazioa mugikorrean probatu da, kasu honetan robotaren papera egingo duen zerbitzaria konputagailu eramangarri bat izanik.

- AVDan probak egiteko, enpresako konputagailua soilik behar izan da. Bertan, aipatutako *bag* fitxategia exekutatuz robotaren *topic*-en simulazio bat lortu da. Inplementatutako zerbitzariak, *topic* hauetara suskribatu eta konexio-eskaeren zain geratzen dira. Beraz, AVDan *localhost* interfazea erabiliz zerbitzarietara konektatzea lortu da. Behin konexioa lortuta, mapa eta kamera funtzionalitateak robotera konektatuta egongo balira bezala funtzionatzen dute, soilik *topic*-etatik jasotako informazioa pantailaratzen baitute. Gida moduan, berriz, helburuen zerrenda eskuratzen da baina aginduak ezin dira aurrera eraman, soilik bezerotik zerbitzarira bidaltzen dira. Robotarekin ezin izan dugunez probarik egin, zerbitzariak jasotzen dituen aginduek ez dute inongo eraginik izango. AVDa erabiliz, soilik zerbitzarira gailu bat konektatzen egin dira probak, esperotako emaitzak lortuz.

- Nahiz eta emuladorean lortutako emaitzak onak izan, benetako sare batean aplikazioaren jokabidea ezberdina izan daiteke, eta mugikor bat baino gehiagorekin probak egin ahal izateko, benetako sare lokal batean egin dira probak. Horrela, 3 mugikor konektatu dira aldi berean zerbitzarira eta era guztietako eskaerak eginez egin dira probak. Bai bezeroen eta bai zerbitzarien jokabidea orokorrean egokia izan da. Zerbitzari lanak egiten dituen konputagailu eramangarriaren sare txartela ez da gaitasun handikoa eta, ondorioz, kamerako irudiak batzuetan atzerapen pixka batekin zerbitzatzen ditu. Hau, konektatutako gailuen araberakoa izango da eta gaitasun handiagoko sare txartel batekin ondo ibili beharko litzake.

Testua interpretatzeko LER zerbitzuari dagokionez, IK4-TEKNIKER enpresan soilik dago atzigarri eta beraz probak enpresan egin behar izan dira. Hau probatu ahal izateko, ez da robotaren eskuragarritasuna behar, mugikorrak egiten baitu zerbitzuaren eta robotaren arteko bitartekari lana. Zenbait proba egin dira ahots eskaera ezberdinak eginez eta kasu guztietan esperotako emaitza lortu da.

Gida moduari dagokionez, robota atzigarri dagoenean egingo dira probak. Funtzionalitate guztiak martxan daudenean, IK4-TEKNIKER enpresako zenbait kiderekin azterketa bat egingo da, ondoren honi buruzko iritzia jasotzeko.

Android aplikazioaren funtzionamenduaren demo txiki bat hurrengo estekan ikus daiteke: [https://youtu.be/RWQXRAlMJWo]. Lehenengo proba, IK4-TEKNIKERen egin da, AVDa erabiliz. Bigarrena berriz, etxeko sare lokala eta zenbait gailu erabiliz.

6. KAPITULUA

Jarraipena eta kontrola

Edukia

6.1 Burututako lana

6.1 taulan ikus daitekeen bezala, errealitatean geratutakoa planifikatutakoarekin konparatuz nahiko parekatuta dagoela esan dezakegu nahiz eta desbiderapen batzuk gertatu diren.

6.1.1 Desbiderapenak

- **Teknologiaren azterketan:** Arazo nagusia ROS erabiltzeko lehenengo pausuetan gertatu zen. Teknologia guztiz ezezaguna izanik, honen funtzionamendua ulertzeko

Atazak	Estimatutako ordu kop.	Benetako ordu kop.
Proiektuaren planifikazioa egin	10	8
Teknologiaren azterketa egin	30	40
Aplikazioa garatu	200	215
Jarraipena eta kontrola egin	20	20
Memoria eta defentsarako gardenkiak egin	80	70
GUZTIRA	340	353

6.1 Taula: Planifikatutako eta errealitateko ataza bakoitzeko ordu kopurua.

espero baino denbora gehiago behar izan zen. Gainera, inplementazioan beharrezkoa zen *catkin* ingurunea prestatzen ere espero baino denbora gehiago behar izan zen.

- **Aplikazioren garapenean:** Aplikazioaren garapenean zenbait desbiderapen gertatu dira, garapenerako behar izan diren ordu kopurua dezente handitu dutenak.

  - Komandoak bidaltzeko bezeroa inplementatzerako orduan, hasiera batean inplementatu zen bezala, zenbait arazo sortzen ziren socket-etik behar ez zen momentuan irakurtzeagatik. Ondorioz, guztiz berrantolatu behar izan zen eta beste era batera inplementatu, socket-etik komando bat bidaltzen zen bakoitzean jarraian erantzunaren zain entzuten geratuz.

  - Mapa moduko interfazea inplementatzerakoan, hasiera batean maparen irudia fondo arrunt bezala jartzea pentsatu zen ondoren honen gainean robota irudikatu ahal izateko. Honek, RAM baliabide izugarria eskatzen zuen eta, gainera, koordenatu-sistemarik ez edukitzeak dezente zailtzen zuen robotaren kokapena. Ondorioz, baliabide grafikoak era eraginkorrean kudeatu ahal izateko eta koordenatu sistema bat izango zuen ingurune bat sortu ahal izateko Unity motor grafikoa erabiltzea pentsatu zen. Hau teknologia guztiz berria izanik, denbora dezente behar izan zen honen funtzionamendua ulertzeko.

- **Memoria idaztean:** Hau idazten espero baino denbora gutxiago behar izan da, planifikatzerako orduan gaizki egindako estimazio baten ondorioz izan daiteke.

6.2 Kalitatea

Kalitateari dagokionez, planifikatutako kalitate maila lortu da jarraian azalduko den bezala.

6.2.1 Produktua

Produktuari dagokionez, esan daiteke kalitate maila onargarria betetzen duela. Lehenengo baldintza erabiltzaile batek baino gehiagok aplikazioa batera erabili ahal izatea da eta nahiz eta gida modua oraindik probatu gabe egon, beste bietan egindako probetan emaitza egokiak lortu dira. Horrez gain, kodea dokumentatu egin da funtzio bakoitzak egiten duena azalduz.

6.2.2 Memoria

Memoriari dagokionez ere, kalitate maila onargarria betetzen du, izan ere Informatika Fakultatearen web orrian jarritako txantiloia erabili baita hau idazteko.

6.2.3 Defentsarako gardenkiak

Nahiz eta oraindik defentsarako gardenkiak ez dauden prest, kalitate onargarria betetzea izango da helburua.

6.3 Arriskuak

Hauek izan dira hasieran aurreikusi ziren eta proiektuan eragina izan duten arriskuak:

- Robotaren eskuragarritasunak eragin handia izan du proiektu honetan. Planifikazioan azaldu zen bezala, robotaren eskuragarritasunak proiektua mugatu duela esan daiteke. Izan ere, hasiera batean robotean egiteko pentsatuta zeuden probak ordenagailu lokalean egin behar izan dira. Gainera enpresako sare lokaleko zenbait portutako trafikoa moztuta zegoenez, ezin izan da mugikorra konputagailu lokalarekin konektatu eta, beraz, probak emuladorean egin behar izan dira.

- Lehen azaldu den bezala, aurreikusitako ataza bakoitzari eskaini beharreko ordu kopurua eta benetan behar izan diren ordu kopurua ez datoz bat baina esan beharra dago orokorrean desbiderapena ez dela oso handia izan. Zehazki 13 ordu gehiago (%3.8) behar izan dira.

7. KAPITULUA

Ondorioak eta etorkizuna

Edukia

7.1 Ondorioak

Proiektu honen ondorioz *Teknibot* robotarekin urruneko elkarrekintza hobetzeko lehenengo pausuak izan diren Android aplikazio mugikor bat garatu da. Nahiz eta hasiera bateko helburua hau robotean probatzea izan, proiektutik zenbait kanpo eragileren ondorioz ezin izan dira definitutako epean probak egin. Konputagailu lokaleko emuladorean eta benetako mugikorrekin egindako proben emaitzak ikusita esan daiteke proiektuaren helburua lortu dela.

Aplikazioaren funtzionalitateak erabaki ondoren, erabili beharreko teknologiak aukeratzerako orduan zenbait arazo sortu ziren ezagutza faltagatik. Ondorioz, zenbait erabaki garrantzitsu hartu behar izan dira, adibidez *rosjava* (ROSen Androideko bertsioa) erabili ala aplikazioa ROSekiko independentea egin.

Hori erabaki ondoren, eta aplikazioa inplementatzen hasi aurretik, diseinuak izan duen

garrantzia azpimarratzekoa da. Izan ere, egin beharrekoaren ideia batzuk argi izanda denbora dezente aurreztea lortu da.

Inplementazioan, hasiera batean espero ez ziren zenbait arazo sortu dira baita ere. Adibidez, mapa funtzionalitatearen interfazea inplementatzerako orduan, hasiera batean erabaki ziren teknologien eraginkortasun eskasa ikusita, beste batzuk bilatzeko beharra sortu da.

Orokorrean, proiektu honen garapenean zehar zenbait teknologia berri ikasi ditut guztiz interesgarriak gertatu zaizkidanak. Gainera, nahiz eta hasiera batean zailtasun handiko proiektua iruditu, aurrera egin ahala gero eta erosoago sentitu naiz emaitza onak lortzen lagunduz. Polita izango litzake funtzionalitate gehiago gehitzeko aukera izatea baina eskura izan den denborarekin, ahalik eta erabilgarrienak direnak inplementatu dira.

Jarraian, proiektuan zehar ikasitako eta garrantzitsuak diren bi aipamen egingo dira:

- **Teknologiaren azterketaren garrantzia:** Proiektu baten garapenean erabiliko diren teknologiek ondorio zuzena dute amaierako produktuan. Izan ere, gure kasuan aplikazioa ahalik eta eraginkorrena izan beharko baita, honen funtzionamendua egokia dela ziurtatzeko baliabide ezberdinak dituzten gailuetan.

  Hau esanda, proiektuaren hasieran egin beharreko teknologiaren azterketaren garrantzia argi geratzen da. Aukeraketa desegokiak, proiektua aurrera joan ahala gero eta pisu handiagoa izango du, gero eta kostu handiagoa izango du beste teknologia bat erabiltzera pasatzeak.

  Proiektu honetan, lehen aipatu bezala, mapa irudikatzeko teknologia aukeratzerako orduan zenbait zalantza sortu ziren. Izan ere, grafikoak kudeatzeko teknologiak erabiltzea lan gehiegi izan zitekeela pentsatu zen. Nahiz eta hasieran okerreko aukeraketa egin, garaiz erabaki zen grafikoak kudeatzeko tresna bat behar zela eta horregatik hartu zen Unity erabiltzeko erabakia.

- **Diseinuaren garrantzia:** Proiektuaren produktua inplementatzen hasi aurretik, gomendagarria da denbora bat hartzea honek izango duen egituran pentsatzeko. Horrela, inplementazioa errazagoa egingo zaigu normalean, hasieratik ideia garbi bat izango dugulako buruan. Ohikoa izaten da pausu hau gainetik kendu eta zuzenean inplementatzen hastea, ezer pentsatu gabe, baina horrela arrisku handia dago egindakoa aldatu behar izateko, egindako beste atalen batekin bat ez datorrelako.

7.2 Limitazioak

Sortutako aplikazioa ez da inolaz ere aplikazio komertzial bat, erakusle bat baizik. Beraz, honen funtzionamendua erabiltzaile gutxi batzuekin probatu da eta hau robotak duen sare txartelaren menpe egongo da. Izan ere, gero eta ahalmen handiagoa izan, orduan eta erabiltzaile gehiago zerbitzatzeko gai izango baita.

7.3 Etorkizuna eta hurrengo pausoak

Nahiz eta aplikazioa eta robotaren arteko probak ez diren proiektu honetan sartu, proba hauek aurrera eramango dira aipatutako arazoa konpondu bezain laster, IK4-TEKNIKER enpresan nire kontribuzioa amaitzeko hilabete geratzen baita. Izan ere, bai enpresaren eta bai nire interesa aplikazio erabilgarri bat izatea delako.

Etorkizunari begira eta aplikazio hau Teknibot eta erabiltzaileen arteko urrutiko elkarrekintzaren lehenengo pausua izanik, aukera on bat izan daiteke aplikazio sendoago eta osoago bat egitea hau oinarritzat hartuta. Hauek izan daitezke inplementatu ez diren eta interesgarriak diren zenbait funtzionalitate:

- Ikusita robotak lokalizatzeko sentsore ezberdinak erabiltzen dituela (*kinect* kamera, laserra ...) interesgarria izan daiteke dagoeneko sortu den bi dimentsioko mapan hauen irakurketak irudikatzea. Adibidez, laserretik lortutako puntu lainoak (*pointcloud*) erabiliz, ingurunean dauden oztopoak irudikatu daitezke.
- Aurreko ideia urrunago eramanda, bi dimentsioko ingurunea izan beharrean hirukoa izatea.
- Proiektu honetan sortu den aplikazioa sare lokalean soilik ibili ahal izateko diseinatu da. Beraz, interesgarria izan daiteke robotera internet zabaletik konektatu ahal izatea.
- Android aplikazioan erabiltzaile mota ezberdinak gehitu litezke, mota bakoitzari funtzionalitate ezberdinak eskainiz.

Bibliografia

[1] Android. URL https://www.android.com/. 2017-06-08.

[2] Git. URL https://git-scm.com/. 2017-06-07.

[3] International Federation of Robotics. URL https://ifr.org/. 2017-06-07.

[4] Informatika Fakultatea - Informatika Fakultatea - UPV/EHU. URL http://www.ehu.eus/eu/web/informatika-fakultatea. 2017-06-07.

[5] rosjava - ROS Wiki. URL http://wiki.ros.org/rosjava. 2017-06-15.

[6] amcl - ROS Wiki, 2017. URL http://wiki.ros.org/amcl. 2017-06-09.

[7] Unity - Game Engine, 2017. URL https://unity3d.com/. 2017-06-09.

[8] Download the iRobot HOME App for the Roomba Vacuuming Robot, 2017. URL http://www.irobot.com/For-the-Home/App/Download.aspx. 2017-06-09.

[9] navigation - ROS Wiki, 2017. URL http://wiki.ros.org/navigation. 2017-06-13.

[10] web_video_server - ROS Wiki, 2017. URL http://wiki.ros.org/web_video_server. 2017-06-09.

[11] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A.Ñg. ROS: an open-source Robot Operating System. URL https://www.willowgarage.com/sites/default/files/icraoss09-ROS.pdf.

[12] H. Sarbolandi, D. Lefloch, and A. Kolb. Kinect range sensing: Structured-light versus Time-of-Flight Kinect. *Computer Vision and Image Understanding*, 2015. ISSN 1090235X. doi: 10.1016/j.cviu.2015.05.006.

[13] L. Susperregi, A. Fernandez, I. Fernandez, S. Fernandez, and I. Maurtua. Talking with a robot: understanding human instructions to a guidance autonomous robot. URL https://pdfs.semanticscholar.org/3d17/7385eea1bcd39fd9b30e649cbdfd4c086e14.pdf.

[14] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. MIT Press, 2005. ISBN 9780262201629. URL https://en.wikipedia.org/wiki/Monte_Carlo_localization.