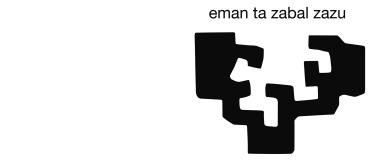


Informatika Ingeniaritzako Gradua

Konputazioa

Gradu Amaierako Proiektua

Lego MINDSTORMS EV3 eta Linux-en arteko konektibitatea

Egilea

Aratz Puerto Gonzalez

2015

Laburpena

Jarraian aurkezten den dokumentuan *LEGO Mindstorms EV3* eta *Linux*en arteko konektibitatea ahalbidetzen duen *driver*raren garapenaren nondik norakoak azaltzen dira. Zehatzago hitz eginez, *Linux* ingurune batean *Robot Operating System* (*ROS*) erabiliz, *EV3*aren sarrera portuen informazioa eskuratzeaz gain irteera portuak kontrolatzea ere lortu da. Sentsoreen informazioa eskuratu eta motorrak kontrolatzeaz gain odometriaren informazioa ere eskuragarri jartzen da.

Gaien aurkibidea

Irudien aurkibidea

Taulen aurkibidea

1. KAPITULUA

LEGO Mindstorms eta robotika

Atal honetan proiektua aurkeztuko da. Lehenik eta behin proiektuaren nondik norakoak azaltzen dira eta ondoren *LEGO Mindstorms EV3*. Amaitzeko *ev3_lang* liburutegi eta *Robot Operating System*en sarrera bat egiten da.

Sarreraren egitura
1. Proiektuaren helburua 2. *LEGO Mindstorms EV3* 3. *ev3_lang* liburutegia 4. *Robot Operating System*

1.1 Proiektuaren helburua

Nahiz eta *LEGO Mindstorms EV3* izan den *Linux* sistema eragile baten gainean funtzionatzen duen *LEGO*ren lehen robota, gaur egun ez dago mota honetako robota eta *Linux*en arteko konektibitaterik. Hau da, erabiltzaileak ezin du bere ordenagailutik zuzenean mota honetako robotik programatu. Ez behintzat *Linux* sistema eragile batetatik. Programatu nahi izan ezkero robotaren bloke programagarrian bertan egin beharko luke. Edo *Windows* edo *MAC OS* sistema eragileak erabili.

Proiektu honen helburua *LEGO Mindstorms EV3* eta *Linux*en arteko konektibitatea ahalbidetuko duen *driver* bat sortzea da. Honela, familia honetako robotak erabiltzailearen ordenagailutik *Robot Operating System* erabiliz programatzea ahalbidetuz. Beti ere *Linux* ingurune batetatik. Izan ere *Windows* eta *MAC OS* sistema eragile eta *LEGO Mindstorms EV3*en arteko konektibitatea hasieratik eskeintzen bait zen.

1.2 *LEGO Mindstorms EV3*

Ukaezina da *LEGO*k roboten eraikuntza, programazio eta kustomizazioan aurrekuntza handiak egin dituela. Are gehiago, *LEGO*ren robot familia desberdinei esker mota desberdinetako erabiltzaileek erabil ditzakete, hasi berrietatik erabiltzaile aurreratuenetaraino.

Interesatua dagoen edozein pertsonak modu errazean robot bat eraiki eta programatzen ikas dezake informatika edo elektronikari buruz aurrez ezer jakin gabe. *LEGO*k bere web orrian hainbat gida ditu eskuragarri jakin beharreko oinarrizko kontzeptuak azalduz baita robotari hainbat forma emateko pausuz pausu jarrai daitezkeen beste gida batzuk ere. Hemendik abiatuta robot hauekin egin nahi den lanaren limitea bakoitzak ezartzen du.

LEGO Mindstorms EV3, *LEGOren Mindstorms* familiako hirugarren generazioko robota da. 2013. urtean merkaturatu zen eta *Linux* sistema eragile baten gainean funtzionatzen duen *LEGO*ren lehen robota da. Honen egitura aurrerago aztertuko da *Tresna eta teknologiak* atalean.

*EV3*aren aurrekaria *NXT* deiturikoa da. *LEGO Mindstorms*en bigarren generazio honen eraikuntza eta programazioa *EV3*aren antzekoa da. *EV3*arekin bateragarriak diren sentsore eta motorretaz baliatuz robota guztiz kontrolatzea posible da eta kasu askotan *NXT*rako programatutako kodeak ere *EV3*n exekuta daitezke.

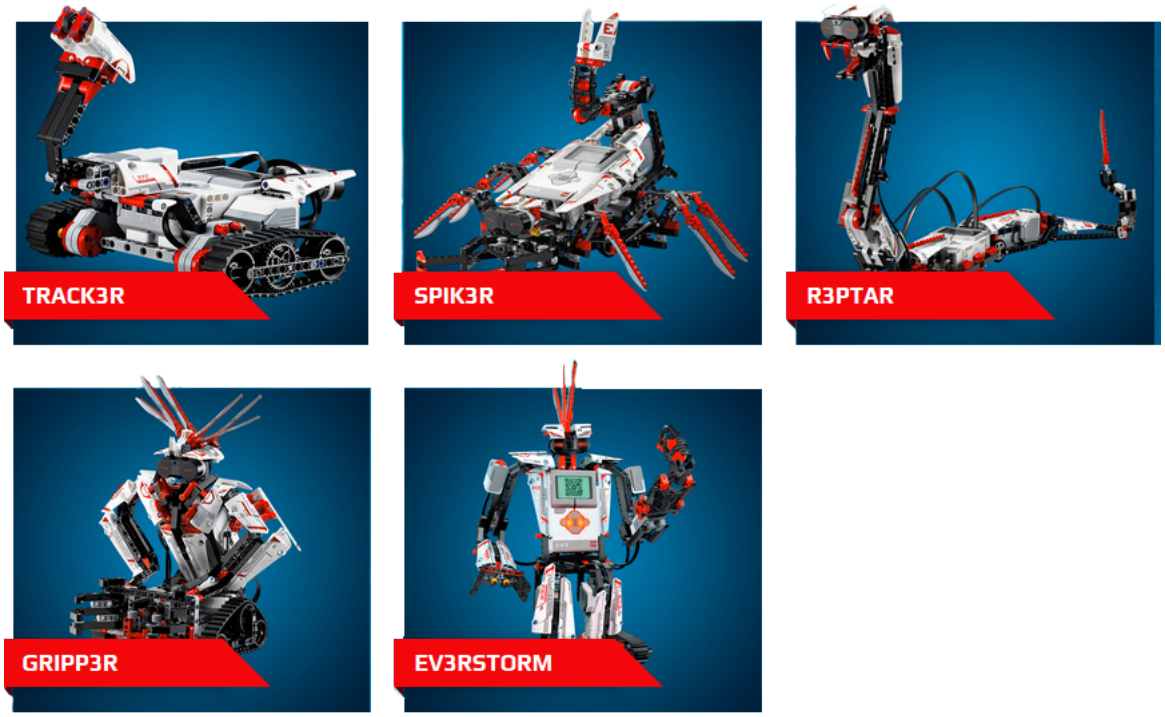
1.2.1 Roboten eraikuntza

Roboten eraikuntza *LEGO*k hain ezagun egin dituen plastikozko blokeak elkartuz egiten da. Hauetaz gain, *LEGO*k beste pieza berezi batzuk ere sortu ditu roboten funzionamendu eta mugimendua ahalbidetzeko. Eraikuntzaren muina 1.1 irudian ikusten den bloke programagarria da, honek kontrolatzen baititu robotak funtzionatzeko erabiliko dituen motor eta sentsoreak.

1.1 Irudia: *EV3* bloke programagarria

*LEGO*k, bere web orrian, bost robot prototipo ditu *EV3*arentzako. Robot hauek *LEGOren Heroi* izenez aurkezten dira eta *LEGO*ren roboten munduarekin lehen kontaktu erraz bat eskeintzea da hauen helburua. 1.2 irudian *LEGO*ren bost *Heroi* edo prototipoak ikus daitezke. Bakoitzak izen eta forma desberdin bat du eta bere portaera ere desberdina izango da *LEGO*ren tutorialak jarraitu ezkero. *LEGO*ren web orrian *LEGOren Heroi*en muntaia pausoz pauso azaltzen dituzten gidak aurki daitezke.

Esan bezala, sentsoreak eta motorrak erabiltzen dira robotaren eraikuntza eta kontrolerako. Jarraian *EV3*k erabiltzen dituen sentsore eta motorrak eta gaur egun hauek programatzeko dauden aukerak aurkezten dira.



1.2 Irudia: *LEGO*ren *Heroi*ak

1.2.2 Sentsoreak

*LEGO Mindstorms EV3*k robotari bost sentsore mota gehitzeko aukera ematen du. Sentsore hauen neurketez baliatuta robotaren portaera kontrola daiteke. Hala nola, koloreen detekzioa, biraketa angeluak, objektuen gertutasuna eta talkak detektatzeko erabil daitezke. Sentsore hauen ezaugarriak hurrengoak dira.:

Kolore sentsorea

EV3 kolore sentsoreak zazpi kolore desberdin detekta ditzake, baita kolore falta ere. Sentsoreak detekzio mota desberdinak ditu. Honela, argia eta honen intentsitatea detektatzeko aukera ematen du. Mota honetako sentsoreak askotan objektuen kolore bidezko identifikazioa edo robotak lerro bat jarraitzeko erabiltzen dira.

*EV3*ren kolore sentsoreak *NXT*ren kolore sentsoreekin konparatuz hobekuntzak jasan ditu. *NXT*ren kolore sentsorea sei kolore detektatzeko gai den bitartean *EV3*ren kolore sentsoreak zazpi kolore desberdin eta kolore falta ere detekta ditzake.

1.3 Irudia: EV3 kolore sentsorea

Biraketa sentsorea

Biraketa sentsorea robotaren egonkortasuna lortzeko erabiltzen da. Robotaren biraketa eta orientazioa ardatz desberdinetan neur dezake, honela erabiltzaileari angeluak kalkulatu ahal izatea ahalbidetuz. *EV3* biraketa sentsoreari esker oso mugimendu zehatzak egiteko gai diren robotak diseina daitezke. 440 gradu/segundoko neurketak egiteko gai da ± 3 graduko zehaztasunez.

NXT biraketa sentsorea ardatz bakarrean neurtzeko gai da. Ondorioz, *NTX*ren biraketa neurketen kalkuluaren zehaztasuna *EV3*rena baina murritzagoa da.

1.4 Irudia: *EV3* biraketa sentsorea

Infragorri sentsore eta baliza

1.6 irudiko balizak seinale infragorri bat igortzen du eta 1.5 irudiko sentsore infragorria seinale hau detektatzeko diseinatua dago. Gainera, baliza urruneko kontrol bezala ere erabil daiteke. Bi osagai hauen arteko elkar lanaz oztopoak detektatzeko gai diren robotak sor daitezke.

Sentsorea 70 zentrimetrotarainoko gertutasuna neurtzeko gai da eta balizarekin bi metro arteko distantzian komunika daiteke. Gainera iturri desberdinetatik hel daitzeken lau seinale desberdin aldi berean detektatu eta prozesatzeko aukera ematen du.

Bere aurrekariaren sentsore infragorriekin konparatuz, *EV3*ren sentsore infragorriak zehatzagoak izateaz gain ez dute beste hobekuntza aipagarririk.

1.5 Irudia: *EV3* sentsore infragorria

1.6 Irudia: *EV3* baliza infragorria

Ultrasoinu Sentsorea

Ultrasoinu sentsoreak uhinen bidez funtzionatzen du. Mota honetako sentsoreak robota eta bere ingurunearen arteko distantziak neurtzeko erabiltzen dira. Sentsoreak soinu uhinak bidaltzen ditu eta hauen oihartzuna jaso. Honela 250 zentimetrorainoko distantziak neurtzeko gai da ± 1 zentimetroko zehaztasunez.

*NXT*ren ultrasoinu sentsorearen neurketa distantzia maximoa *EV3*rena baina txikiagoa izateaz gain (5 zentimetro txikiagoa) ez ditu hain neurketa zehatzak egiten (± 3 zentimetroko zehaztasuna). Gainera *EV3*ren ultrasoinu sentsorea ezaugarri berri batekin dator, sonar modua.

1.7 Irudia: *EV3* ultrasoinu sentsorea

Talka sentsorea

EV3 talka sentsorea sentsore sinple eta zehatza da. Sentsore hau tresna analogiko sinple bat da, botoi bat du eta sakatua/askatua ekintzak detektatzeko gai da. Mota honetako sentsoreak normalean robota abiarazi/gelditzeko erabili ohi dira. Beste erabilera bat oztopo ekidite eta labirintoen irteera bilatzea izaten da. Sentsore honek ez du hobekuntzarik jasan aurreko generaziotik.

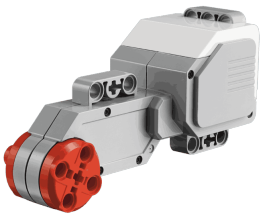
1.8 Irudia: EV3 talka sentsorea

1.2.3 Motorrak

*LEGO Mindstorms EV3*ak ahalmen eta erantzun denbora desberdinetako bi motatako erregulazio motorrak erabiltzen ditu: Handiak eta ertainak. Motor hauen bidez robotaren mugimendua kontrolatzen da. Bakoitzak bere ezaugarriak izanik hauen arteko diferentzia nagusia motor handiek ertainek baina indar handiagoa dutela da. Aldiz, motor ertainen erantzun denbora laburragoa da. Hauen ezaugarriak hurrengoak dira:

Motor handia

EV3 motor handia ahalmen handiko motorra da. Takometroa erabiliz motorraren posizio eta abiaduraren kontrol zehatza ahalbidetzen du gradu bateko zehaztasun mailaz. Barnean dakarren biraketa sentsorea erabiliz, motor inteligente hau mota bereko beste motorrekin lerrokatu eta sinkronizatzea posible da. Honela bi motorrei abiadura bera ezarriz robota lerro zuzenean mugitzea lor daiteke. Honetaz gain, motorraren odometria informazioaren irakurketa ere ahalbidetzen du. Motorraren odometriaren irakurketa egiterakoan motorraren uneko posizioa lortzen da. Hemendik, emandako bira kopurua eta biraketa angelua bezalako informazioa lor daiteke. Kontuan izan behar da biraketa bakoitzean motorrak 360 *tick* egiten dituela. Hortaz, adibidez, motorrak uneraino eman duen bira kopurua jakin nahi bada balio horrekin zatituz lor daiteke.



1.9 Irudia: *EV3* motor handia

Motor ertaina

Motor ertaina erantzun denbora azkarragoak edo tamaina txikiagoak behar diren kasuetarako erabilgarria da. Motor handien kasuan bezala takometroa erabiliz gradu bateko zehaztasuneko kontrol zehatza lortzen da, eta biraketa sentsore bat dakar barnean.

1.10 Irudia: *EV3* motor ertaina

1.2.4 Programazioa

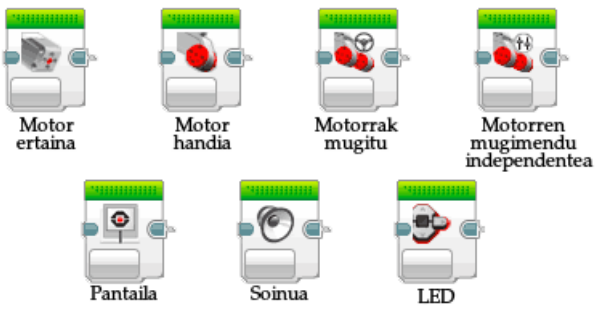
Behin erabiltzaileak robotari nahi duen forma emanda robotaren programazio fasea has daiteke. Arestian aipatu den bezala, robot hauek jakintza maila desberdinetako erabiltzaileei bideratua datoz eta hau, programazioaren ezagutza desberdineko erabiltzaileei ere aplikagarria da.

Hasiberriak

LEGOk bere web orrian *software* berezi bat du eskuragarri aurretik roboten programazioaren inguruko ezagutzarik ez duten erabiltzaileek robot hauek modu erraz eta intuitiboan programatu ahal izateko. *Mindstorms EV3* programazio *software*aren esentzia robotaren eraikuntzaren bera da. Hau da, robotaren forma blokeak elkartuz egiten den bezala programazioa ere modu honetan egin ahal izatea. Programa honek tutorial desberdinak ditu *LEGO*ren *Heroi*ak (1.2 irudikoak) ikono edo bloke bidezko programazioa erabilita programatzen ikasteko. Esan bezala, modu honetan robota programatzeko ez da aurretik programatzen jakin behar. Azken finean modu honetara programatuz ez da programazio lerro bakar bat ere idatzi behar.

Programazio mota honetan bost motatako blokeak erabiltzen dira, bakoitza kolore batez identifikatzen delarik:

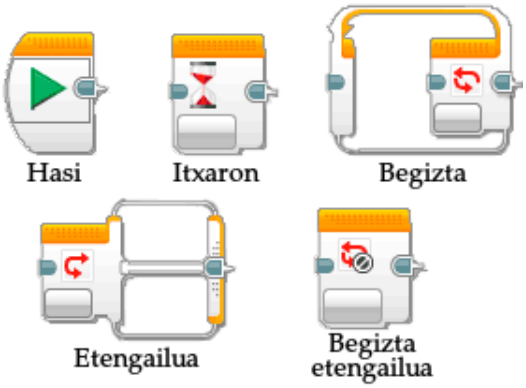
Ekintza blokeak:
Ekintza blokeek programaren ekintzak kontrolatzen dituzte. 1.11 irudian ikus daitekeenez, besteak beste, motorren biraketa, pantailaren irudiak, soinuak eta bloke programagarriaren LEDak ere programa daitezke bloke hauei esker.



1.11 Irudia: Ekintza blokeak (Orlegiak)

Fluxu blokeak:

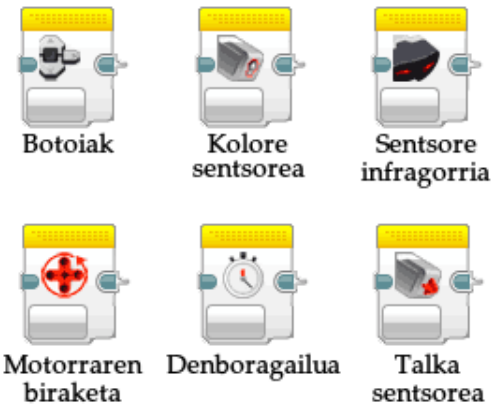
Fluxu blokeek programaren fluxua kontrolatzen dute. Sortutako programa guztiak hasieratzeko bloketik hasiko dira eta honi beste bloke batzuk kateatuz programaren fluxu osoa sortzen da. 1.12 irudian fluxu blokeekin programa daitezkeen ekintzak azaltzen dira.



1.12 Irudia: Fluxu blokeak (Laranjak)

Sentsore blokeak:

1.13 irudiko sentsore blokeei esker programak robotaren sentsoreen neurketak eskuratu ahal izango ditu.



1.13 Irudia: Sentsore blokeak (Horiak)

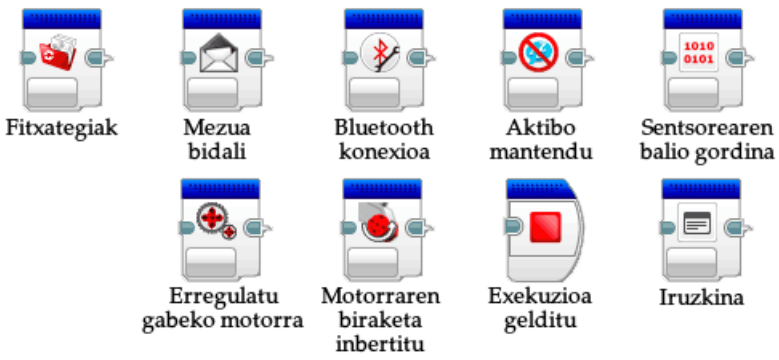
Datu eragiketa blokeak:

Datu eragiketa blokeak aldagaien kontrolerako erabiltzen dira. Besteak beste hauek irakurri eta idatzi eta balioen konparaketa egitea ahalbidetzen dute bloke hauek. 1.14 irudian Datu eragiketa blokeak ikus daitezke.

1.14 Irudia: Datu eragiketa blokeak (Gorriak)

Bloke aurreratuak:

Izenak dioen bezala, bloke aurreratuak ekintza aurreratuetarako erabiltzen dira. 1.15 irudian ikus daitekeenez, bloke aurreratuei esker fitxategien kudeaketa eta Bluetooth konexioa ezartzea bezalako ekintzak programa daitezke robotean.



1.15 Irudia: Bloke aurreratuak (Urdinak)

Bloke bidezko programazio honi esker aurretik programatzen ez dakien edozein erabiltzailek *LEGO Mindstorms EV3*a modu intuitibo eta errazean programatu ahal izatea ahalbidetzen da *LEGO*ren bloke kateamenduaren esentzia jarraituz. Erabiltzaile aurreratuagoek, berriz, beste aukera batzuk ere badituzte mota honetako robotak programatzeko.

Erabiltzaile aurreratuak

Programazio aurreratuago batek *LEGO Mindstorms EV3*aren oinarrizko *firmware*a atzitzea, eta kasu batzuetan ordezkatzea, eskatzen du honela sentsore eta motorren kontrol zuzena lortzeko. Hau ahalbidetzen duten *framework* ezagunenak hurrengoak dira:

1. **BrickOS**

   *BrickOs EV3*aren bloke programagarriaren *BIOS*a zuzenean atzitzea ahlbidetzen duen liburutegi eta tresna multzo bat da. Honi esker *BIOS*ean mikro sistema eragile bat instala daiteke blokearen baliabide guztiak eskuragarri jarriz. *BrickOs* instalatu ahal izateko jatorrizko *firmware*a instalatua dagoen memoria zatia berridatzi behar da, baina behin hau eginda *EV3*a *C*,*C++* eta *Makina lengoaian* programa daiteke Aukera hau *Windows*ek eta *Linux*en distribuzio gehienek onartzen dute.

2. **LejOS**
   *BrickOS*ek ez bezala *LejOS*ek ez du sistema eragilea jatorrizko *firmware*a ordezkatuz instalatzen. Aldiz, *Java Makina Birtual* bat instalatzen du, honela bloke programagarria *Java* programazio lengoaian programagarri eginik. Ondorioz, ez da inongo konpilatzaile edo ordezko sistema eragileren beharrik.

3. **Not Quite C**
   Not Quite C (*NQC*)k ere ez du jatorrizko *firmware*a ordezkatzen. Horren ordez, jatorrizko *firmware*aren sistema deiak emulatzen ditu blokea *NQC* programazio lengoaian programagarri eginez.

Framework hauetaz gain, *LEGO Mindstorms EV3*a *Linux* ingurune batean programatu ahal izateko aukerak murritzak badira ere, aurreko atalean aipatutako *BrickOS*, *LejOS* eta *NQC* ez dira aukera bakarrak. Emulatzaileak alde batera utzita tresna nabarmenenak Monodevelop eta ev3dev dira.

1. **Monodevelop** *Monodevelop* plataforma desberdinetarako IDE bat da. Honi esker *C#* eta beste *.NET* lengoaiatan *EV3*a programatu ahal izatea ahalbidetzen du.

2. **ev3dev** *ev3dev LEGO*ren *Linux kernel* egokitua da. Bere *Hardware driver* propioekin *EV3*a erabiltzaileak gogokoen duen programazio lengoaian programatu ahal izatea ahalbidetzen du. *EV3*aren osagai guztiak modu errezean kontrolatu ahal izatea helburu duen *Debian* distribuzio hau *micro-sd* txartel batean instala daiteke, honela, *LEGO*ren firmwarea berridatzi behar izan gabe oso modu erosoan erabil daiteke.

Edozein kasutan, aipatutako aukera hauek *LEGO Mindstorms EV3*rekin konektibitatea lortzeko oinarri bezala erabil badaitezke ere, bloke programagarriaren programazioa blokean bertan egin behar da, esan bezala gaur egun ez bait dago inongo konektibitaterik *Linux* makina eta *EV3*aren artean. Aukera hauek aztertu ondoren proiektu honentzako indartsu eta egokiena *ev3dev* dela erabaki da eta hau oinarritzat erabiliz eraman da aurrera hurrengo ataletan azaltzen den proiektua.

1.3 ev3_lang liburutegia

*ev3dev*ek eskeintzen dituen tresnen artean *ev3_lang* liburutegia dago. Liburutegi honek hainbat programazio lengoaietan programatu ahal izateko API desberdinak bateratzen ditu. Zehazki, *C++*, *LUA*, *Node.js* eta *Python* programazio lengoaietan *EV3*aren osagai guztiak era erosoan programatu ahal izatea ahalbidetzen duten APIak biltzen dituen liburutegi bat da hau.

Proiektu honen garapenean *C++* programazio lengoaiaren *API*a erabili da zerbitzari aldean sentsore eta motorren kudeaketa egin ahal izateko. *ev3_lang* liburutegiaren *C++* APIak *ev3_cpp* izena hartzen du.

1.4 Robot Operating System

Arestian aipatu den bezela proiektu honen helburua robota erabiltzailearen ordenagailutik *Robot Operating System* erabilita programatu ahal izatea da. *Robot Operating System* edo *ROS* robotentzako aplikazioak sortzeko *software* liburutegi eta tresna sorta bat da.

*ROS*ekin idatzitako programek *nodo* izena hartzen dute. *Nodo*ek, euren artean komunikatzeko (eta datuen tansferentzia eta kudeaketarako) *topic* izeneko datu egiturak erabiltzen dituzte. Honela, proiektu honetan, sentsoreek irakurritako datuak, motorrek hartuko dituzten abiadurak eta irakurritako motorren odometria *topic* desberdinetan utzi ezkero programatzen diren beste *nodo*ek informazio hau modu errezean eskuratu edo idatzi ahal izango dute. *Nodo* nagusiak *topic*etan irakurri edo idazten ari diren *nodo*en zerrenda bat gordetzen du ondoren *nodo*en arteko komunikazioa errazteko. *Nodo* batek *topic* batean idatzi nahi badu *advertise()* funtzioa erabili behar du. Hau egitean *nodo* nagusiari zein *topic*etan idatziko duen esaten dio. Ondoren, beste nodo batek *subscribe()* funtzioa (*topic*etatik irakurtzeko erabiltzen dena) erabiltzen badu *nodo* nagusiak *topic* horretan idazten ari diren *nodo*ei jakinaraziko die eta *peer-to-peer* motako konexio bat irekiko da bi nodoen artean. *advertise()* funtzioak *Publisher* motako objektu bat itzultzen du eta *subscribe()* funtzioak, berriz, *Subscriber* motako. Honegatik, hurrengo ataletan funtzio hauek erabiltzeari *subscriber* edo *publisher*rak sortzea esango zaio.

Bezero programa *ROS nodo* (exekutagarri) bat izango da. *ROS C* eta *C++* programazio lengoaietan erabil daitekenez, bezero programa programazio lengoaia hauetan programatua egongo den zerbitzariarekin ere bateragarria izango da. Bezero programaren exekuzioa errazteko *ROS*ek eskeintzen duen *roslaunch* abiarazlea erabiliko da. Honek, exekutatuko diren nodoak behar dituzten parametroekin (kasu honetan *EV3*aren *IP* helbidea) exekutatzen ditu dei bakarrean. Esan bezala sentsore eta motorren informazioa *ROS topic* desberdinetan utziko da.

2. KAPITULUA

Proiektuaren Helburuen Dokumentua

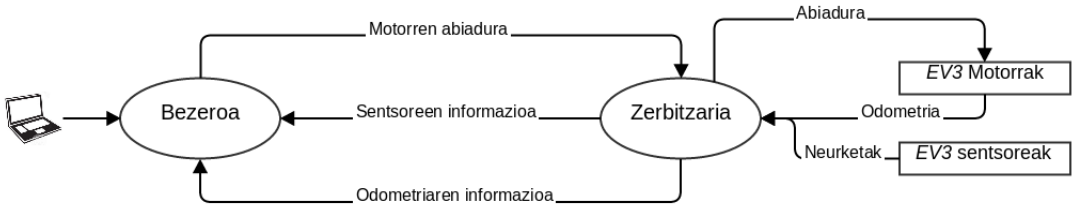
Kapitulu honetan proiektuaren planifikazioa azaltzen da. Hurrengo puntuak jorratuko dira jarraitzen duten ataletan:

Proiektuaren helburuen egitura
1. Proiektuaren irismena 2. Denboraren planifikazioa 3. Kalitatea 4. Komunikazio plana 5. Arriskuen kudeaketa plana

2.1 Proiektuaren irismena

Atal honetan proiektuaren irismena deskribatzen da. Hau da, proiektuaren betekizunak azalduko dira *LDE* diagrama batean egin beharreko lana deskonposatuz. Gainera, proiektuaren emangarriak zerrendatzeaz gain mugarriak ere identifikatzen dira.

Proiektuaren helburua *LEGO Mindstorms EV3* eta *Linux*en arteko konektibitatea ahalbidetzea da. Honetarako bezero-zerbitzari eredua jarraitzen duen *driver* bat diseinatu eta inplementatuko da. Bezero programak erabiltzaile eta *EV3*aren arteko tartekari lana egingo du. Honela, erabiltzaileak bere ordenagailutik *ROS* erabiliz zuzenean sentsoreen neurketak eta motorren odometria informazioa eskuratzeaz gain motorren kontrola ere izango du. Zerbitzari programak, *EV3*aren sarrera eta irteera portuen atzipen zuzena duenez sentsoreen neruketak irakurri eta etengabe bidaliko dio bezero programari. Gainera, bezero programak motorretan idatzi beharreko informazioa bidaltzen duenean, informazio hau eskuratu eta motorretan idazteaz ere arduratuko da. Esan bezala motorren odometria informazioa ere eskuratuko du etengabe bezero programari bidaliz.



2.1 Irudia: *Driver*raren eskema orokorra

Proiektuak nagusiki hurrengo atalak izango ditu:

1. **Datu transferentzia:**
   (a) *Driver*ra osatzen duten bezero eta zerbitzari programen arteko TCP/IP komunikazioa.
2. ***Driver*ra:** Hurrengo funtzionalitateak izango ditu
   (a) Sentsoreen datuen irakurketa.
   (b) Motorren abiaduraren idazketa.
   (c) Odometria informazioaren irakurketa.
3. **Oztopoak ekiditeko nodoa**
   (a) *Driver*raren funtzionamendua probatzeko *ROS nodo*a.

*Driver*ra osatzen duten ataza desberdinek *TCP* konexio propioa ezarriko dute zerbitzariarekin, beharrezkoak dituen datuen irakurketa edo idazketak egin ahal izateko.

Datu transferentzia hurrengoa izango da:

- **Sarrerako portuak** (sentsoreak):
  Zerbitzariak bezero programari hurrengo informazioa bidaliko dio *buffer* baten bidez, konektatua dagoen sentsore bakoitzeko:
  - Sentsorea konektatua dagoen portua.
  - Portu horretan irakurtzen ari den balioa.
- **Irteerako portuak** (motorrak):
  Bezero programak zerbitzariari hurrengo informazioa pasako dio *buffer* baten bidez konektatua dagoen motor bakoitzeko:
  - Idatzi nahi den portua.
  - Idatzi nahi den balioa.
- **Odometria:**
  Zerbitzariak bezero programari hurrengo informazioa pasako dio *buffer* baten bidez, konektatua dagoen motor bakoitzeko:
  - Motorra konektatua dagoen portua.
  - Motorraren uneko posizioa.

*Buffer*ra jaso ondoren, hauen informazioa *TCP* protokoloak zehazten duen moduan erauzi eta kasu bakoitzean beharrezkoa den tratamendua ematen zaie datuei.

Datuen tratamendua hurrengoa da:

- **Sarrerako portuak:** Zerbitzaritik jasotako portuaren arabera, *ROS topic* batean honen balioa publikatuko da. Portuaren arabera *topic* desberdin batean publikatzen da balioa.

- **Irteerako portuak:**
  Topic desberdinak irakurriz motorrek edukiko duten abiadura eskuratzen da. Irteerako portu bakoitzari *topic* desberdin bat dagokio:
  Topic hauetako balioa eskuratu ondoren, *buffer*rean dagokien portuarekin batera

sartu eta zerbitzariari bidaltzen zaio. Zerbitzariak orduan dagokion irteerako portuan konektatua dagoen motorrari bere abiadura emango dio.

- **Odometria:** Zerbitzaritik jasotako irteera portuaren arabera, *ROS topic* batean honen balioa publikatuko da. Portuaren arabera *topic* desberdin batean publikatzen da balioa.

2.1.1 Betekizunak

*Driver*raren garapena proiektuaren muina izan arren, beste hainbeste zeregin daude proiektuaren helburuen artean. Proiektuaren planifikazioa, teknologien azterketa, kudeaketa, proiektuaren memoria eta defentsa ere kontuan hartu beharreko puntuak dira. Proiektuaren nondik norakoak hobeto ulertzeko 2.1 eta 2.2 tauletan honen atazak biltzen dira.

1. **Proiektuaren planifikazioa**
   (a) Proiektuaren irismena
   (b) Ataza eta betekizunak zerrendatu
   (c) Aurreikusitako denbora kalkulatu
   (d) Atazen egutegi posiblea
   (e) Arriskuen kudeaketa
      i. Identifikazioa
      ii. Ekiditeko plana
   (f) Kalitate plana
   (g) Komunikazio plana
2. **Teknologiaren azterketa**
   (a) *LEGO Mindstorms EV3*ren azterketa
   (b) *ev3dev Linux* ingurunea prestatu
   (c) *ROS* lan ingurunea prestatu
   (d) *ev3dev-lang API*aren azterketa
   (e) *ROS*en azterketa.
      i. Nodoen funtzionamenduaren azterketa
      ii. *Topic*en funtzionamenduaren azterketa

2.1 Taula: Proiektuaren atazak.

3. ***Driver*raren garapena**
   (a) Bezero moduluaren diseinua
   (b) Zerbitzari moduluaren diseinua.
   (c) TCP komunikazioaren diseinua
   (d) Bezero eta zerbitzari moduluen inplementazioa
   (e) Oztopoak ekiditeko nodoaren diseinu eta inplementazioa
4. **Probak**[nosep]
   (a) TCP bidezko datu transferentzia probak
   (b) Sentsore probak
   (c) Motor probak
   (d) Odometria probak
   (e) Funtzionamendu probak
5. **Kudeaketa**[nosep]
   (a) Jarraipen eta kontrola
   (b) Bilerak
6. **Proiektuaren memoria garatu**
7. **Defentsa**

2.2 Taula: Proiektuaren atazak (jarraipena).

2.1.2 *LDE* Diagrama

2.2 irudian proiektuaren lan deskonposaketa irudikatzen da. Hauek dira honen elementu nagusien azalpenak:

- **GAP**: Gradu Amaierako Proiektua.
- **Proiektuaren planifikazioa**: hasieran proiektuan zehar bete behar diren helburuak definitzen dira:
  - **Proiektuaren irismena**: Proiektuaren azalpena. Zertan datzan, bere helburua eta abar.
  - **Atazak eta betekizunak zerrendatu**.
  - **Aurreikusitako denbora**: proiektuaren atal bakoitza egiteko beharrezkoa izango den denboraren estimazioa.
  - **Atazen egutegia**: proiektua kurtsoan zehar nola banatuko den. Ataza bakoitza noiz hasi eta bukatuko den zehaztu.
  - **Arriskuen kudeaketa**: proiektua garatzen den heinean, hau trabatu edo bertan behera utzi arazi dezaketen arrazoiak identifikatu eta kontrako neurriak hartu behar dira.
  - **Kalitate plana**: *Driver*ra, memoria eta defentsarako gardenkiek izan behar duten kalitate maila minimoa, egokia eta hoberena definitu.
  - **Komunikazio plana**: proiektuaren interesatuekin nola komunikatu definitu.
- **Teknologiaren azterketa**: *Driver*ra garatzen hasi aurretik erabiliko diren tresnak erabaki behar dira, uneko egoera aztertu eta erabiliko diren baliabideak ezagutzeko.
  - ***LEGO Mindstorms EV3*ren azterketa**: *EV3*aren egituraren azterketa, honek eskeintzen dituen aukerak ezagutu eta sarrera/irteerako portuen, sentsoreen eta motorren funtzionamendua ulertzeko.
  - ***ev3dev linux* ingurunea prestatu**: *LEGO Mindstorms EV3*an *ev3dev debian* distribuzioa eta ondoren erabiliko den softwarea eta dependentziak instalatu.
  - **Lan ingurunea prestatu**: *Driver*ra *C* eta *C++* lengoaiatan idatzia izango bada ere *ROS*era bideratua izango da. Hortaz *ROS*en *catkin* ingurunea prestatu behar da.

  - ***ev3dev-cpp* APIaren azterketa**: *Driver*raren zerbitzari aldean *ev3dev-lang* liburutegiaren *ev3_cpp API*a erabiltzen da sentsore, motor eta abar kontrolatzeko. API honen azterketa egin behar da bere erabilera ezagutzeko.
  - ***ROS*en azterketa**: *ROS*ek eskeintzen dituen baliabideen azterketa egin behar da hauek *driver*rari aplikatu ahal izateko.
    * **Nodoen funtzionamenduaren azterketa**: *ROS* nodoek nola funtzionatzen duten aztertu eta hauek erabili eta inplementatzen ikasi.
    * ***Topic*en funtzionamenduaren azterketa**: *ROS topic*ek nola funtzionatzen duten aztertu eta hauek erabili eta inplementatzen ikasi.

- ***Driver*raren garapena**: *Driver*raren garapena 5 ataletan bana daiteke

  1. **Bezero moduluaren diseinua**: Bezero modulua hainbat ataza aldi berean aurrera eramateko gai izan behar denez hiru hari desberdinetan banatzen da:
     - Sentsoreen informazioa irakurtzeko hariaren diseinua.
     - Motorren informazioa idazteko hariaren diseinua.
     - Odometria informazioa irakurtzeko hariaren diseinua.
  2. **Zerbitzari moduluaren diseinua**: Zerbitzari aldeko moduluaren diseinua.
  3. **TCP/IP Komunikazioaren diseinua**: Bezero eta Zerbitzari moduluen komunikaziorako TCP/IP komunikazioaren diseinua.
  4. **Inplementazioa**.
  5. **Oztopoak ekiditeko nodoaren diseinu eta inplementazioa**.

- **Probak**: *Driver*raren moduluen atal bakoitza inplementatzen den heinean hauen funtzionamendu zuzena ziurtatzeko hainbat proba egin behar dira. Honetaz gain, *driver*ra osatua dagoenean honek exekuzio garaian errorerik ematen ez duela ziurtatzeko ere probak egin behar dira.
  - **TCP bidezko datu transferentzia probak**: Bezero eta zerbitzari arteko komunikazioa ondo betetzen dela ziurtatu.
  - **Sentsore probak**: Sentsoreen informazioa ondo irakurtzen dela ziurtatu.
  - **Motor probak**: Motorretan informazioa ondo idazten dela ziurtatu.
  - **Odometria probak**: Odometria informazioa zuzen irakurtzen dela ziurtatu.

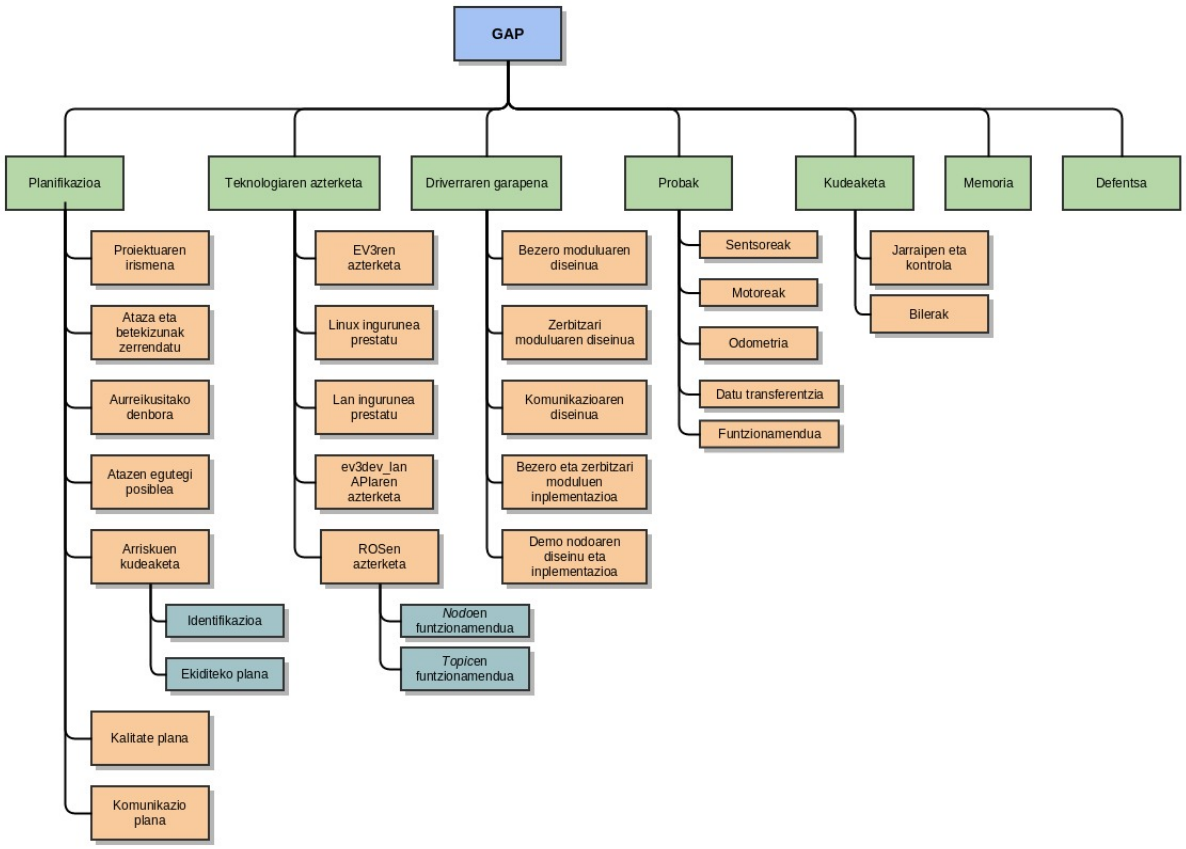
  - **Funtzionamendu probak**: *Driver*rak eta oztopoak ekiditeko nodoek exekuzio errore edota bestelako funtzionamendu errorerik ematen ez dutela ziurtatu.

- **Kudeaketa**: proiektuan zehar egindako lanaren kudeaketa.

  - **Jarraipen eta kontrola**: egindako ataza bakoitzaren kudeaketa, sortu diren arazoen kontrola eta behar izan den denbora.
  - **Bilerak**: Proiektuaren garapenak dirauen bitartean hainbat bilera egingo dira tutoreekin. Batzuk honen egoeraren berri emateko, beste batzuk berriz proiektuaren nondik norakoak zehazu edo dudak argitzeko.

- **Memoria**: Proiektuaren memoriaren garapena.

- **Defentsa**: Defentsarako gardenkiak eta aurkezpena prestatu.



2.2 Irudia: Proiektuaren lan deskonposaketa irudikatzen duen LDE diagrama

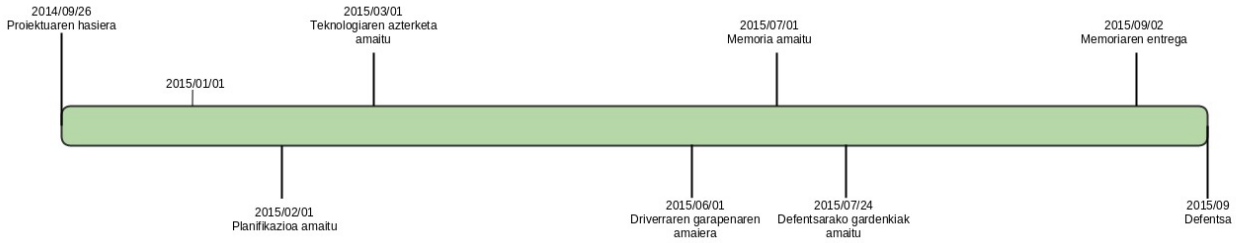
2.1.3 Emangarriak

Proiektuan zehar bi emangarri mota sortuko dira:

- *Driver*rarekin erlazionatuak:
  - *Driver*raren exekutagarria bai bezero bai zerbitzari aldean.
- Proiektuarekin orokorrean erlazionatuak:
  - Proiektua garatzeko plana.
  - Proiektuaren memoria.
  - Proiektuaren defentsarako gardenkiak.

2.1.4 Mugarriak

2.3 irudian proiektuaren mugarriak irudikatzen dira. Estimazio hauek hasieran egindakoak dira eta ez dute islatzen bukaeran betetakoa.



2.3 Irudia: Proiektuaren mugarriak

2.2 Denboraren planifikazioa

Atal honetan proiektuan erabiliko den denboraren plangintza azaltzen da.

2.2.1 Atazak

2.2 irudiko LDE diagraman proiektuaren atazak irudikatzen dira. Nahiz eta bertan proiektuaren atal guztiak azaldu, ez da hauen azpiatal guztien denbora estimatu.

2.2.2 Atazen estimazioa

Atal honetan proiektuaren atal bakoitza burutzeko estimatzen den denbora adierazten da:

Proiektuaren planifikazioa: 30 ordu.

Teknologiaren azterketa:

- *LEGO Mindstorms EV3*ren azterketa: 7 ordu
- *ev3dev linux* ingurunea prestatu: 2 ordu
- *ROS* lan ingurunea prestatu: 1 ordu
- *ev3dev-cpp API*aren azterketa: 2 ordu
- *ROS*en azterketa: 10 ordu

Hortaz, 22 ordu estimatzen dira ondorengo atazetan erabiliko den teknologia aztertzen.

***Driver*raren garapena**:

- Bezero moduluaren diseinua: 3 ordu
- Zerbitzari moduluaren diseinua: 3 ordu
- Komunikazioaren diseinua: 5 ordu
- Bezero eta zerbitzari moduluen inplementazioa: 140 ordu
  - Bezero moduluaren inplementazioa: 80 ordu
  - Zerbitzari moduluaren inplementazioa: 60 ordu
- Oztopoak ekiditeko nodoaren diseinu eta inplementazioa: 1.5 ordu
  - Oztopoak ekiditeko nodoaren diseinua: 1 ordu
  - Oztopoak ekiditeko nodoaren inplementazioa: 30 minutu

Ondorioz, *driver*raren garapena proiektuaren muina izanik 152.5 ordu estimatzen dira honen garapenerako.

Probak:

- TCP bidezko datu transferentzia probak: 2 ordu.
- Sentsore probak: 2 ordu.
- Motor probak: 2 ordu.
- Odometria probak: 2 ordu.
- Funtzionamendu probak: 10 ordu.

Hortaz, *driver*raren funtzionamendu egokia ziurtatzeko probetan 18 ordu estimatzen dira guztira.

Kudeaketa:

- Jarraipena eta kontrola: 15 ordu.
- Bilerak: 10 ordu.

Kudeaketa faserako 25 ordu estimatzen dira.

Memoria: 50 ordu.

Defentsa: 20 ordu.

Proiektu osoarentzako 317.5 ordu beharko direla estimatzen da. 2.3 taulan estimazioen laburpena ikus daiteke:

Ataza	Estimatutako orduak
Planifikazioa	30
Teknologiaren azterketa	22
Driverraren Garapena	152.5
Kudeaketa	25
Memoria	50
Defentsa	20
Guztira	317.5

2.3 Taula: Proiektuaren denbora estimazio orokorra

2.2.3 Atazen planifikazioa

2.4, 2.5 eta 2.6 irudiek atazen antolaketa Gantt diagrama baten bidez irudikatzen du. Ataza bakoitzaren datak erabakitzeko eragin dezaketen hurrengo egoerak kontuan hartu dira:

- **Unibertsitatea**: Proiektua garatzen den aldi berean klaseak, entregak eta azterketak egongo direnez denbora gutxi edukiko diren garaiak egongo dira. Ezin da aurreikusi kurtsoan zehar izango den lan karga baina azterketa egutegia erabil daiteke hauek noiz izango diren jakiteko.

- **Lana eta praktikak**: Kurtsoan zehar praktiketan eta ondoren lanean arituko naiz. Ondorioz honek denbora asko kenduko didala aurreikusi behar da.

- **Pertsonalak**: Ingurune akademikotik kanpoko jarduera batzuek ere denbora eskatuko didatela kontuan izan behar da.

Proiektua esleitua den unean hasiko da, baina lan handiena bigarren lauhilabetean garatuko da, hau baita libreago izango dudan lauhilabetea. Espero da proiektua denborarekin amaitzea eta Uztaila hasieran amaitzea. Praktika/lana eta fakultateko kargarekin hau lortzea zaila izango da, baina estimazio honekin denbora marjina edukiko da garaiz amaitzeko.

2.3 Kalitatearen kudeaketa

Gaur egun garatzen diren produktuak gero eta kalitate hobeagokoak izan behar dira. Erabiltzaileak ez dira edozerekin konformatzen, horregatik, helburu garrantzitsuenetakoa kalitatezko produktu bat sortzea izan behar da. Proiektua kalitatezkoa izateko lehenik eta behin maila minimoa eta egokia ezarri behar dira. Ez hori bakarrik, hauek betetzeko plan bat ere garatu behar da.

2.3.1 Kalitate maila planifikatu

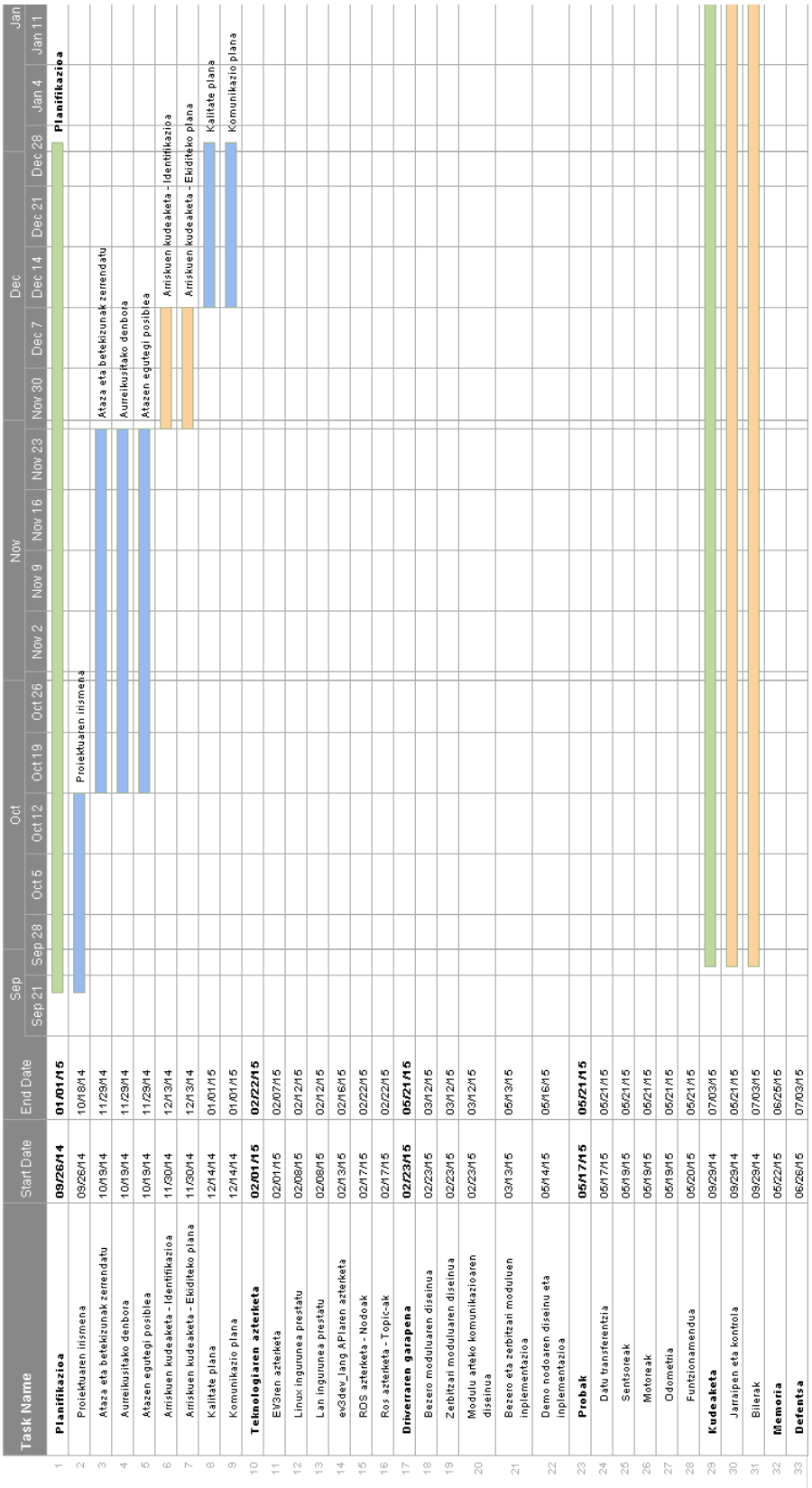
Proiektuan kalitate kontrola behar duten atalak hurrengoak dira:

- **Komunikazioa**: Proiektuaren interesatuen arteko komunikazioan maila egoki bat egon behar da arazorik egon ez dadin.

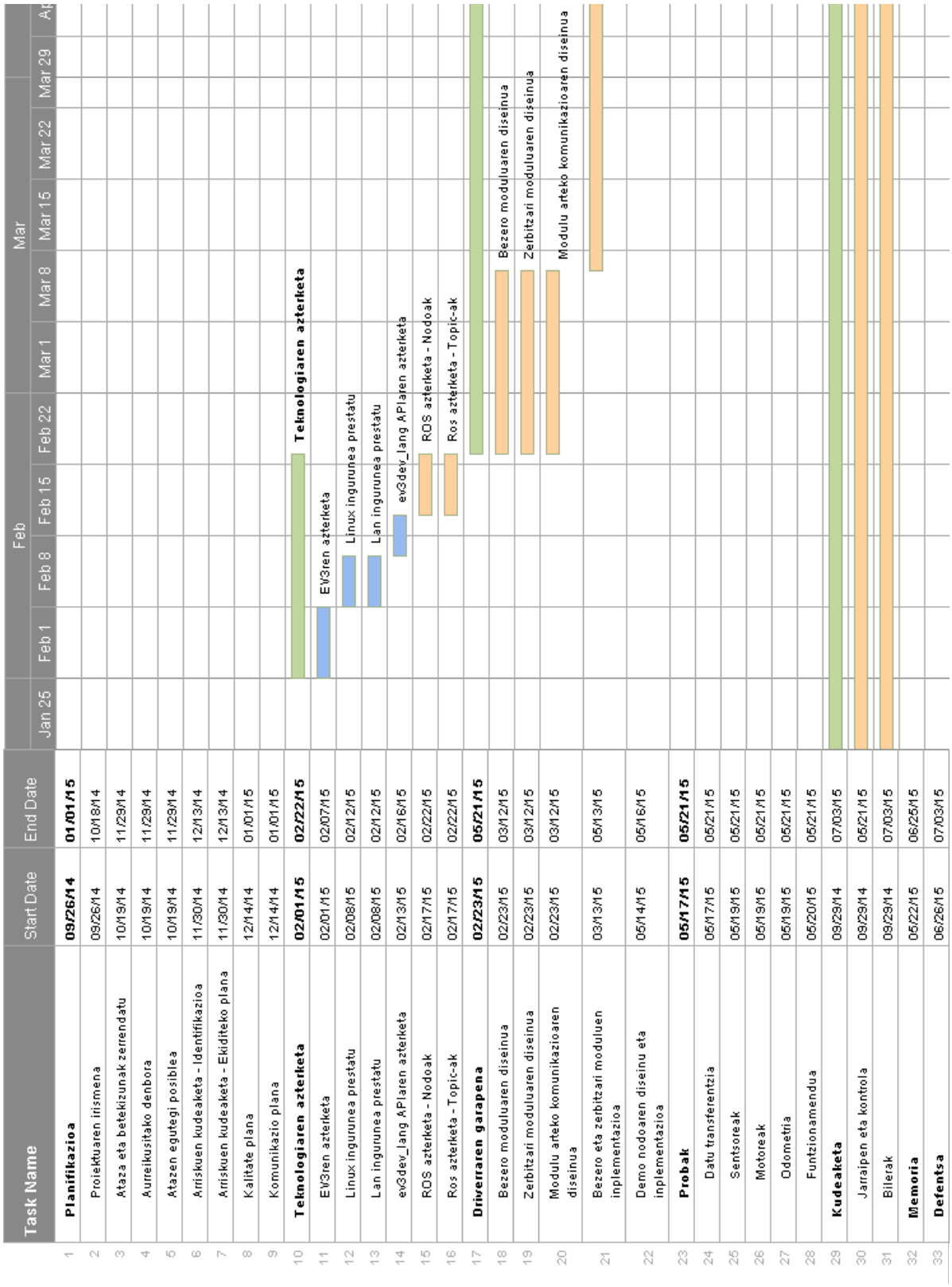
- **Produktua**: *Driver*ra erabilgarria izateko minimo batzuk bete behar dira

- **Memoria**.

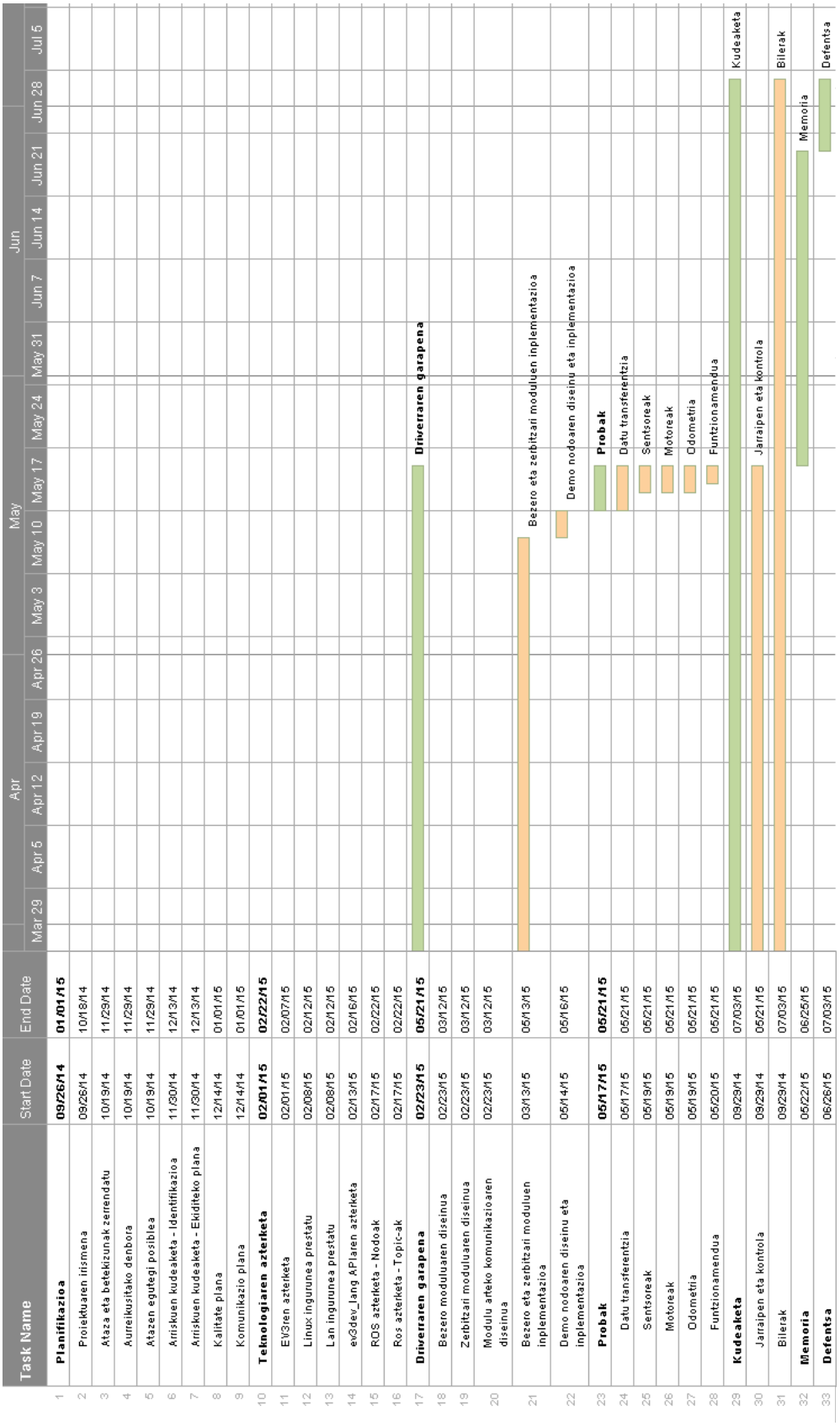
- **Defentsarako gardenkiak**.



2.4 Irudia: Proiektuaren atazen planifikazioa (2014ko Iraila - 2015eko Urtarrila)



2.5 Irudia: Proiektuaren atazen planifikazioa (2015eko Otsaila - 2015eko Apirila)



2.6 Irudia: Proiektuaren atazen planifikazioa (2015eko Apirila - 2015eko Uztaila)

Kalitate maila minimoa

Atal honetan minimo onargarriak zehazten dira.

- **Komunikazioa**: Tutorea informatua mantendu. Proiektuan ataza garrantzitsuren bat aurrera eramaten den bakoitzean tutoreari idatzi edo berarekin bildu. Ez da bilera kopuru minimo bat ezarriko.
- **Produktua**: Irismena atalean adierazitakoa bete. Hau da, sentsore eta motorrak erabili ahal izateaz gain odometria informazioa eskuratu ahal izatea.

  Honetaz gain inplementatutako moduluen kodeak irakurgarri eta ulergarria izan behar du.
- **Memoria**: Kalitatezko memoria bat idazteko EHUk eskatzen duen formatua jarraitu behar da. Gainera, hasi aurretik edukiaren aurkibide bat sortu eta tutoreari erakutsiko zaio. Tutoreak aurkibide hau ontzat eman arte ez da hasiko atal hau.
- **Defentsarako gardenkiak**: Memoriaren antzera, hasi aurretik tutorearen oniritzia eskatuko da. Gardenki kopurua ez da finkoa baina kontuan izan behar da aurkezpenerako denbora mugatua dela. Azaldu nahi den ideia bakoitzeko gardenki bat edo bi sortuko dira.

Kalitate maila egokia

Atal honetan kalitate maila egokia zein den definitzen da.

- **Komunikazioa**: Tutorea informatua mantentzeaz gain lortzen den edozein aurrerapen tutoreari erakutsiko zaio, honela tutoreak proiektuak aurrera egiten duela jakiteaz gain bere aholku eta oniritzia eman ditzan. Honek, gainera, dudak argitzeko aukera ere ematen du.
- **Produktua**: *Driver*raren maila egokia izan dadin betebeharrak betetzeaz gain makinen baliabideen erabilera kontuan izan behar da. *Driver*raren bezero programa lau prozesu paralelotan exekutatzeko gai izan beharko da. *TCP/IP* konexioa konkurrentea izan beharko da zerbitzariak prozesu bakarra erabiliz bezeroaren eskaera guztiei aurre egiteko. Azkenik, kodea txukuna izateaz gain dokumentatua egon beharko da funtzio bakoitzak egiten duena argi azalduz.

- **Defentsarako gardenkiak**: Gardenkiek ez dute eduki gehiegi izan behar. Gardenki bakoitzak testu gutxi eta asko jota 3 ideia nagusi azalduko ditu.

2.3.2 Kalitatearen kontrola

Kalitate mailak definitzea ez da nahikoa. Hauek betetzen direla ere ziurtatu behar da. Honetarako neurri batzuk hartuko dira:

- **Produktuan**: *Driver*raren garapenak dirauen artean proiektuarekin zerikusirik ez duen jendearen iritzia eskatuko da kalitate maila optimizatzeko helburuarekin.
- **Memorian**: Tutorearen iritzia eskatuko da. Ulergarria dela bermatzeko amaitutzat eman aurretik dokumentu osoa berrirakurri eta zuzenduko da.
- **Defentsarako gardenkietan**: Memoriaren kontrol bera erabiliko da.

2.4 Komunikazio plana

Komunikazioa proiektu kudeaketaren oso atal garrantzitsua da. Interesatuak nor diren definitu ondoren proiektuan duten interes maila zehaztu behar da komunikazio plan bat garatzeko.

2.4.1 Interesatuen identifikazioa

Proiektu honek behar bati aurre egiten dio eta ondorioz proiektuan interesatua dagoen jendearen interes maila orokorrean oso altua da. Hala ere *LEGO Mindstorms EV3 Linux* bidez programatzean interesa duen software askeko komunitatea *Interesatuen identifikazio*tik at utzi da.

Proiektuaren egilea

- **Interes maila**: Oso altua.
- **Kontaktua**: *apuerto001@ikasle.ehu.es*
- **Deskribapena**: Proiektuaren egilea. Proiektua gainditu behar duen pertsona izanik txukun eta ondo egitea interesatuen dagoen pertsona.

Tutorea

- **Interes maila**: Oso altua.
- **Kontaktua**: *e.lazkano@ehu.es*
- **Deskribapena**: Proiektuaren zuzendaria. Gaian esperientzia handia izateaz gain lanean *LEGO Mindstorms EV3* robota *Linux* igurune batean programatu ahal izatean interes handia du.

2.4.2 Informazioaren banaketa

Interesatuen arteko informazio trukearentzako komunikazio bide nagusia mezu elektronikoak izango dira. Honetaz gain, garrantzia handiko gaietarako tutorearekin zuzenean hitz egingo da.

2.5 Arriskuen kudeaketa plana

Atal honetan proiektuaren garapena atzera ditzaketen egoerak idenfikatu eta prebentzio planak sortuko dira.

Arazoak edozein aldetatik hel daitezke, gainera, askotan ekidinezinak dira. Horregatik ahalik eta egoera kaltegarri posible gehienak aurreikusi behar dira.

2.5.1 Konputagailua hondatzea

Azalpena: Ordenagailuak arrazoi askorengatik honda daitezke osoki edo partzialki erabilezin geratuz. Are gehiago eramangarriak.

Gertatzeko probabilitatea: Txikia.

Eragina: Oso handia. Ordenagailuan gordeta dagoen informazioaren zati bat edo bere osotasuna gal daiteke. Kasurik okerrenean ordenagailua erabilezin gera daiteke informazio guztia galduz eta lantoki berria bilatu behar izanez.

Jatorri posiblea: Ordenagailua hainbeste erabiltzeak honen osagairen bat hondatzea ekar dezake. Eramangarriak kolperen bat jasotzea ere gerta liteke emaitza antzekoa izanik.

Prebentzio plana: Ordenagailua arduraz erabili eta mugituko da. Eramangarrien kasuan, hau garraiatzeko orduan honen segurtasunerako zorro batean eramango da. Informazioa galdu ezkero *Informazio galera* ataleko prebentzio plana jarraituko da. Ordenagailua erabilezin geratzen bada fakultateko Gradu Amaierako Proiektuetarako gelako ordenagailu bat erabiliko da proiektua aurrera eramateko.

2.5.2 *LEGO Mindstorms EV3*a hondatzea

Azalpena: *LEGO Mindstorms EV3*a hainbat arrazoirengatik honda daiteke erabilezin geratuz.
Gertatzeko Probabilitatea: Txikia.
Eragina: Oso handia. *LEGO Mindstorms EV3*a hondatu ezkero *driver*raren garapena guztiz geldituko litzateke.
Jatorri posiblea: Robota garraiatzeko orduan honek kolperen bat jaso dezake edota euriarekin busti.
Prebentzio plana: *EV3*a hainbestetan garraiatu behar ez izateko fakultateko armairu edo takila bat erabiliko da hau gordetzeko. Garraiatzea ezinbestekoa den kasuetan, robota bere kutxan gordeta eta ondo tapatua eramango da. Arrazoiren batengatik robota hondatuko balitz fakultatean beste bat eskatuko da *driver*raren garapenarekin aurrera egin ahal izateko. Informazioa galdu ezkero *Informazio galera* ataleko prebentzio plana jarraituko da.

2.5.3 Informazio galera

Azalpena: Proiektuaren fitxategi edo dokumenturen bat galtzen den kasua.
Gertatzeko probabilitatea: Altua

Eragina: Handia. Galdutako fitxategia garrantzitsua bada proiektuaren garapen osoan eragin dezake.
Jatorri posiblea: Ordenagailua, eta zehazki disko gogorra, kaltetzen bada fitxategiak galdu daitezke. Gerta liteke nahi gabe fitxategiren bat ezabatzea ere.
Prebentzio plana: Segurtasun kopia eta bertsio kontrolaren erabilera. Segurtasun kopiak periodikoki egingo dira. Batez ere ataza bakoitzaren amaieran eta zerbait garrantzitsua amaitzeko orduan. Bi motakoak izango dira: Aplikazioaren kodea eta proiektuaren dokumentazioa. Aplikazioaren kodearen segurtasun kopiak bi motakoak izango dira:
Bertsio kontrola eta fitxategi bikoizketa. Informazio hau beste ordenagailu batean, USB giltza batean eta Google Driven biltegiratuko da.

2.5.4 Aurreikuspenak ez betetzea

Azalpena: Lana konplikatu eta estimatutako datan ez amaitzearen kasua.

Gertatzeko probabilitatea. Altua. Erabiliko den teknologia ez da ondo ezagutzen eta honek ezusteko zailtasunak ekar ditzake, ondorioz ordu gehiago behar izatea posible litzatekelarik. Honetaz gain praktikak eta fakultatearen exijentzia mailak ere poiektuaren garapena atzera dezake noizbehinka.

Eragina: Txikia. Proiektuaren jarraipen on bat eginez ez litzateke arazo handiegia izan behar ordu gehiago sartzea.

Prebentzio plana: Proiektua ahalik eta goizen hasi eta mugarrietarako marjinak utzita atzeratzea eragingo lukeen arazoren bat gertatu ezkero denbora gehiago edukiko litzateke proiektuaren garapenerako.

3. KAPITULUA

Tresnak eta teknologiak

Atal honetan proiektua garatzeko erabilitako tresna eta teknologia azaltzen dira.

Tresna eta teknologien egitura

1. *LEGO Mindstorms EV3*ren egitura
2. *ev3dev Debian* distribuzioa
3. *ev3_lang* liburutegia
4. *Robot Operating System*
5. Softwarea

3.1 *LEGO Mindstorms EV3*ren egitura

*LEGO Mindstorms EV3*ren bloke programagarriak (1.1 irudikoa) sentsore eta motorrak konektatu ahal izateko hainbat sarrera eta irteerako portu izateaz gain, blokearen egoera adierazten duten LEDak eta soinuarentzako bozgorailua ere baditu. Gainera USB eta SD txartel portu bat ere baditu. Konputagailuarekin puntuz puntuko konexioa egiteko mini-USB portu bat du.

3.1.1 Sarrera portuak

*EV3*aren bloke programagarriak lau sarrera portu ditu 3.1 irudian ikusten den bezala. Portu hauek blokearen azpi aldean kokatuak daude eta 1,2,3 eta 4 zenbakiekin identifikatuta. Sarrerako portuak *EV3* sentsoreak konektatzeko erabiltzen dira.

3.1 Irudia: *EV3*ren sarrera portuak

3.1.2 Irteera portuak

Irteera portuak sarrera portuen aurkako aldean kokatzen dira eta 3.2 irudian ikus daiteken bezala A, B, C eta D letrekin identifikatzen dira. Portu hauek *EV3* motorrak konektatzeko erabiltzen dira.

3.1.3 PC portua

Mini-USB PC portua, D irteera portuaren ondoan kokatua 3.2 irudian ikus daiteeken bezala. *EV3*aren bloke programagarria eta ordenagailuaren arteko puntuz puntuko konexioa egiteko erabiltzen da eta PC hitzaz identifikatzen da.

3.2 Irudia: *EV3*ren irteera eta PC portuak

3.1.4 USB portua

EV3 bloke programagarriaren ezker aldean, 3.3 irudian irudikatzen den bezala, *USB* portu bat aurki daiteke *USB* hitzaz identifikatuta. Hau *Wifi* egokitzaile (*dongle*) bat konektatuz ordenagailu eta blokearen arteko haririk gabeko konexioa ezartzeko erabili ahal izateaz gain, *EV3* bloke programagarri desberdinak (gehienez lau) elkarren artean konektatzeko ere erabil daiteke.

3.1.5 SD portua

SD portua USB portuaren ondoan kokatzen da eta 3.3 irudian azaltzen den bezala *SD* hitzaz identifikatzen da. Portu hau 32 GBrainoko *micro-SD* txartel bat txertatzeko erabiltzen da, honela bloke programagarriaren memoria eskuragarria handituz. Proiektu honen kasuan, *micro-SD* txartelean erabiliko den sistema eragilea biltegiratzen da.

3.3 Irudia: *EV3*ren *USB* eta *SD* portuak

3.1.6 Bozgorailua

Bloke programagarriaren eskuin aldean bozgorailu bat dago 3.4 irudian ikusten den bezala. Honi esker sortutako robotak soinu efektuak erabil ditzake.

3.4 Irudia: *EV3*ren bozgorailua

3.1.7 LED

Blokearen goi aldean, 3.5 irudian ikus daitekeen bezala, pantailaren azpian zehazki eta botoien inguruan blokearen edota exekuzioaren egoeraren berri ematen duten LEDak aurkitzen dira. Irudian hiru kolore ohikoenak (Ggorria , laranja eta orlegia) ikusten badira ere nahi den kolorea ezartzeko programa daiteke. Gainera argiak keinuka piztu eta itzaltzeko aukera eta honen maiztasuna ere programa daitezke.

Argi gorria: Arazoren bat dago. **Argi laranja:** Robota prozesatzen ari da. **Argi orlegia:** Dena ondo doa

3.5 Irudia: *EV3*ren LEDak

3.2 *ev3dev Debian* distribuzioa

*LEGO Mindstorms EV3*arekin datorren *softwarea* ona bada ere, batzuetan ez da erabiltzailearen beharretara heltzen. Hau da proiektu honen kasua hain zuzen ere, nahiz eta bloke programagarriak *Linux* sistema eragile bat dakarren instalatua, ez du bat ere erraztasunik ematen *EV3*aren osagaien *Linux* bidezko programaziorako.

ev3dev debian distribuzio bat izateaz gain *EV3*arentzako *hardware driver* propioak ditu *EV3*aren osagaiak edozein programazio lengoaia erabilita modu errazean programatu ahal izateko. Fitxategi bat irakurri edo idaztea nahikoa da programatu nahi den *EV3*aren funtzionalitatea guztiz kontrolatzeko. Adibidez, robotak 10 segunduz *KHz* 1eko tonoa jo dezan, *tone* fitxategian `1000 10000` (1000 *Hz* eta 10000 milisegundo) balioak idatzi behar dira.

```
$ echo 1000 10000 > tone
```

*ev3dev*ek ez du jatorrizko *firmwarea* berridazten. *SD* portuaz baliatuta *micro-SD* txartel batean instalatzen den sistema eragilea da eta ondorioz jatorrizko *firmware*ra itzuli nahi izango balitz nahikoa litzateke *micro-SD* txartel hau txertatu gabe blokea berrabiaraztea jatorrizko funtzionamendua berreskuratzeko.

3.2.1 Instalazioa

Esan bezala *ev3dev Debian* distribuzioa *micro-SD* txartel batean instalatzen da. *ev3dev*en bertsio berriena https://github.com/ev3dev/ev3dev/releases helbidean aurkitu ahal izango da beti eta honen instalazioa errazteko *ev3dev*en web orrian pausoz pausoko azalpenak aurki daitezke hurrengo helbidean:

http://www.ev3dev.org/docs/tutorials/writing-sd-card-image-linux-command-line/

Behin *ev3dev*en instalazioa amaituta *EV3* blokea piztea, *micro-SD* txartela txertatua duelarik, nahikoa da sistema eragile hau zuzenean exekuta dadin. Gainera, modu honetan instalatua egoteak beste abantaila bat ere badu. *ev3dev*en instalazioa bloke programagarriaren menpekoa ez denez edozein *LEGO Mindstorms EV3*tan *micro-SD*a txertatzea nahikoa da honek *ev3dev*ekin funtzionatzeko

Pausu hauek jarraitu ondoren, besterik ezeko erabiltzailea *root* eta pasahitza *r00tme* (zeroak dira, ez o letra) dira.

3.2.2 USB bidezko konexioa

*ev3dev*en web orrian, *http://www.ev3dev.org/docs/tutorials/connecting-to-the-internet-via-usb/* helbidean pausoz pauso eta oso argi azaldua dagoen gida bat aurki daiteke hasierako USB konexioa konfiguratzeko.

Behin instalazioa amaituta eta hasierako *USB* bidezko konexio hau konfiguratuta *SSH* bidez erabiltzailearen ordenagailutik bloke programagarriaren barne memoria atzi daiteke.

3.2.3 Wifi konexioa

Wifi egokitzaile bat eduki ezkero haririk gabeko konexio bat ezar daiteke konputagailu eta *EV3*aren artean. *ev3dev*ek konexio hau automatikoki blokearen pantaila eta botoiak erabilita ezartzea ahalbidetzen du. Honetarako menuaren *Wireless and Networks* atalean *Wifi* atala atzitu behar da. *Powered* aukeratua dagoela ziurtatuz *Start Scan* aukeratuz atzigarri dauden haririk gabeko konexioen lista bat agertuko da *EV3*aren pantailan. Erabili nahi

den konexioaren *SSID*a bilatu eta aukeratu ondoren menu berri bat azalduko da.

Menu honetan bi aukera daude, *Connect* eta *Network Connection*. Robota *SSID* honetara behin bakarrik konektatu behar bada *Connect* aukera nahikoa da. Hau aukeratuz *SSID*aren pasahitza eskatuko du eta hau zuzen sartu ezkero sarera konektatua egongo da robota.

Network Connection aukerak, berriz, lau fitxa dauzka:

- **Conn.**: Fitxa honetan automatikoki konektatzeko aukera bat dago *Connect automatically*. Hau aukeratu ezkero pasahitza gogoratuko du eta SSID hau atzigarri dagoenean automatikoki haririk gabeko konexioa ezarriko du. Honetaz gain, *Wifi* atalean bezala konektatzeko *Connect* aukera ere badu.

- **IPV4**: Fitxa honetan IPV4aren ezarpen automatikoak kargatzeko aukera dago. Komenigarria da lehenengo aldiz konektatzeko orduan *Change* aukera aukeratu eta *Load Linux defaults* aukeratzea.
- **DNS**: Fitxa honetan, nahi izan ezkero *DNS*ak gehitzeko aukera dago.
- **Enet.**: Fitxa honetan sare txartelaren informazioa aurki daiteke.

Behin *IPV4* fitxako ezarpenak kargatuta *Conn.* fitxako *Connect* aukera sakatzea nahikoa izango da, aurretik azaldutako pausoak jarraituz, haririk gabeko konexio bat ezartzeko.

3.2.4 Dependentziak

Proiektu hau aurrera eraman ahal izateko oinarrizko *ev3dev* instalazioa ez da nahikoa. Honetaz gain beste pakete batzuk instalatu behar izan dira:

- **build_essential**: Pakete honek, besteak beste, *C* eta *C++* konpilatzaileak erabili ahal izateko beharrezkoak diren fitxategi eta liburutegiak biltzen ditu.
- **libboost-dev**: *C++*en iturri fitxategiak biltzen dituen paketea.

Behin dependentzia hauek instalatuta *linux* ingurunea prest dago programatzen hasteko.

3.2.5 *LEGO Sensor* klasea

*ev3dev*ek sentsoreen programaziorako *LEGO Sensor* klasea eskuragarri jartzen du. Arestian esan bezala sentsoreen kudeaketa fitxategi baten irakurketa edo idazketa soilaz egin diateke.

Sentsorearen identifikazioa

Sentsore bat konektatzen den bakoitzean */sys/class/lego-sensor/* karpetan karpeta berri bat sortuko da *sensor#* izenarekin non # osoko zenbaki bat den. Zenbaki hau sentsore bat konektatzen den bakoitzean handitzen doa, banaka eta 0tik hasita. Sentsore bat erauzten denean, berriz, honen karpeta ezabatu egiten da karpeta horretatik. Hala ere sentsore berri bat konektatzean esandako zenbakiak ez du hartuko libre dagoen lehenengo osoko zenbakia. Aldiz, konexio bakoitzean unitate batean inkrementatzen joango da. Demagun sentsore bat konektatzen dugula 1 zenbakia duen sarrerako portuan, */sys/class/lego-sensor/sensor0* karpeta sortuko da. Ondoren 2 zenbakidun sarrerako portuan beste sentsore bat konektatuz */sys/class/lego-sensor/sensor1* karpeta sortzen da. Orain 1 zenbakia duen sarrerako portuan konektatua dagoen sentsorea deskonektatu eta portu berean berriz konektatu ezkero ez litzateke */sys/class/lego-sensor/sensor0* karpeta sortuko, */sys/class/lego-sensor/sensor2* karpeta baizik.

A Eranskinean karpeta honen barruan sortzen diren eta sentsorearen programaziorako erabiltzen diren fitxategiak ikus daitezke.

3.2.6 *Tacho motor* klasea

Motorren programazioa sentsoreenaren antzekoa da.

Motorren identifikazioa

Motorren identifikazioa sentsoreen identifikazioen berdina da baina */sys/class/tacho-motor/* karpetaren barnean

A eranskinean motorraren programaziorako erabiltzen diren fitxategiak aurki daitezke.

3.3 *ev3dev-lang* liburutegia

Aurreko atalean aipatzen diren fitxategiak sentsore eta motorrak programatzeko erabil badaitezke ere, maila horretan programatzea batzuetan zail egiten da. *ev3-dev*ek *ev3dev-lang* liburutegiaren bidez hurrengo programazio lengoaien *API*ak eskeintzen ditu:

- **C++**
- **Lua**
- **Node.js**
- **Pthon**

Liburutegi hauei esker sentsore eta motorren programazioa asko errazten da. Proiektu honetan *C++* liburutegia erabili da *driver*raren kodea idazteko orduan. *ev3dev-lang* liburutegiaren *C++*en APIak *ev3dev-cpp* izena hartzen du.

3.4 *Robot Operating System*

LEGO Mindstorms EV3 eta *Linux*en arteko konektibitatea lortzerakoan, erabiltzaileak bere ordenagailua erabiliz *EV3*a *Robot Operating System* bidez programatzea da helburuetako bat. *Robot Operating System* (*ROS*) robotentzako aplikazioak sortzeko *software* liburutegi eta tresna sorta bat da.

Proiektuaren muina den *driver*raren garapenean bi *ROS* kontzeptu erabiltzen dira: *Nodo*ak eta *Topic*-ak.

- ***Nodoa***: Agindu multzo bat biltzen duen exekutagarria.

- ***Topic***: Nodoen arteko datu transferentziarako erabiltzen diren biltegiratze unitateak. Nodo batek *topic* bat irakurri edo idatz dezake unean uneko beharren arabera.

*ROS nodo*ek *topic*etan irakurri edo idatz dezakete hurrengo funtzioak erabilita:

- ***advertise()***:

  Funtzio honek bi parametro hartzen ditu. Lehenengoa idatzi nahi den *topic*aren izena da. Bigarrena idatziko diren datuen ileraren tamaina. *Nodo* batek funtzio hau erabiltzen duenean *Publisher* motako objektu bat sortzen da. Objektu honi esker aukeratutako *topic*ean idatzi ahal izatea ahalbidetzen du *ROS*ek. *Publisher* honen instantzia guztiak ezabatuak direnean *nodoa*k ezingo du *topic* horretan berriro idatzi funtzio honi dei berri bat egin arte.

- ***subscribe()***:
  advertise funtzioak bezala, funtzio honek bi parametro hartzen ditu. Lehenengoa irakurri nahi den *topic*aren izena da. Bigarrena irakurriko diren datuen ileraren tamaina. *Nodo* batek funtzio hau erabiltzen duenean *Advertiser* motako objektu bat sortzen da. Objektu honi esker aukeratutako *topic*etik irakurri ahal izatea ahalbidetzen du *ROS*ek. *Advertiser* honen instantzia guztiak ezabatuak direnean *nodoa*k ezingo du *topic* horretatik berriro irakurri funtzio honi dei berri bat egin arte.

Funtzio hauei esker *nodo*en arteko *topic* bidezko komunikazioa ahalbidetzen da. *Nodo* nagusiak *Publisher* objekturen bat sortu duten *nodo*en zerrenda bat gordetzen du. *Nodo*ren batek *Advertiser* objektu bat sortzen duenean *topic* berarekin erlazionatuak dauden *nodo*ei, hots, *topic* horretan idazten ari direnei, jakinarazten die eta *peer-to-peer* motako konexio bat ezartzen da bi *nodo*en artean, honela hauen arteko elkarrekintza sortuz.

Honetaz gain, *ROS*ek hainbat tresna erabilgarri eskeintzen ditu. Proiekatu honetan erabilitako tresna aipagarriak hurrengoak dira:

- ***rqt_graph***: Proiektu honetan garrantzia handia izan duen *rqt_graph* tresna hau, exekuzio garaian *nodo* eta *topic*en arteko elkarrekintza irudikatzeko erabiltzen da.

- ***roslaunch***: *roslaunch ROS nodo* bat edo gehiago dei bakarrean exekutatzea ahalbidetzen duen tresna bat da. Gainera, exekuzio hau, lokala edo *SSH* bidezkoa izan daiteke. *roslaunch*ek *XML* konfigurazio fitxategi bat edo gehiago erabil ditzake (*.launch* luzapenarekin) zein makinatan exekutatu behar diren ezartzeaz gain exekutatu behar diren *nodoak* eta hauen parametroak ezartzeko.

3.5 Softwarea

*Driver*raren garapenean ez da *IDE*rik erabili, izan ere *driver*raren zerbitzari aldeko programa blokean bertan programatu baita eta bertan komando lerrotik exekuta daitezkeen programak soilik erabil baitaitezke. Ondorioz *Vim* testu editorea erabili da.

Dokumentazioa idazteko, berriz *LaTex* erabili da *TexMaker*en bitartez.

4. KAPITULUA

Proiektuaren garapena

Kapitulu honetan proiektuaren nondik norakoak azaltzen dira. *Driver*raren diseinu eta inplementazioak azaltzen dira. Jarraian, *Oztopoak ekiditeko nodo*aren diseinu eta inplementazioak. Azkenik, exekuzioari buruzko azalpenak eta egindako probak adierazten dira.

Proiektuaren garapenaren egitura
1. *Driver*raren garapena 2. Oztopoak ekiditeko nodoaren garapena 3. Exekuzioa 4. Probak

Arestian aipatu den bezala, nahiz eta *LEGO Mindstorms EV3*k *Linux* sistema eragile baten gainean funtzionatzen duen, gaur egun ez dago familia honetako robot eta *Linux*en arteko konektibitaterik. Hots, erabiltzaileek ezin dute robotaren programazioa zuzenean euren ordenagailuetatik egin, *EV3*aren bloke programagarrian bertan baizik.

Proiektu honen helburua *LEGO Mindstorms EV3* eta *Linux*en arteko konektibitatea ahalbidetuko duen *driver* bat sortzea da. Honela, aukerako edozein *Linux* ingurunetan *ROS*erabiliz mota honetako robotak programatzea ahalbidetuko delarik.

4.1 *Driver*raren garapena

*EV3*aren portaera erabiltzailearen ordenagailutik zuzenean programatu ahal izatea ahalbidetuko duen *driver*rak konputagailu eta robotaren arteko datu transferentzia ahalbidetu behar du.

4.1.1 Diseinua

4.1 irudiko eskemak adierazten duen bezela, *driver*rak bi alde izango ditu. Bezero aldea eta zerbitzari aldea:

- **Zerbitzari programa**: *Driver*raren zati hau *EV3*an exekutatuko da. Honek sentsore eta motorrekin hartueman zuzena izango du, sentsoreen neurketak eta motorren odometria informazioa irakurri eta motorretan informazioa idaztearen ardura bere gain hartuko duelarik.
- **Bezero programa**: Zati hau erabiltzailearen ordenagailuan exekutatuko da. Erabiltzailearekin hartuemana izango da bere helburua, sentsoreetatik irakurri den informazioa eta motorren odometria eskuragarri jarri eta motorretan ezarri beharko den abiadura eskuratzearen lanaz arduratuz.

Honetaz gain bi programa hauen arteko komunikazioa ere existitu behar da. Datu transferentziarentzako TCP bidezko konexioa erabiliko da.

Zerbitzari programa

Zerbitzari programa *EV3*an etengabe exekutatzen ari den *daemon* bat da. Hiru ataza bete behar ditu:

- **Sentsoreen informazioa eskuratu**: Sentsoreren bat konektatzen denean honen informazioa etengabe irakurri eta bezero programari bidali behar dio.
- **Motorretan informazioa idatzi**: Bezero programatik motorren batean idatzi behar den informazioa jaso ezkero motorrean idatzi behar du, konektatua baldin badago. Kontuan izan behar da abiadura -100 eta 100 balioen artean bornatua dagoela eta 100 baino balio handietarako 100 balioa ematen zaiola automatikoki eta -100 baino balio txikiagoetarako -100 balioa.

- **Odometria informazioa bidali**: Motorren bat konektatua dagoenean odometria informazioa irakurri eta bezero programari bidali.

Programa hau robota hasieratzerakoan automatikoki exekutatu behar da.

Bezero programa

Bezero programa erabiltzaileak behar duenean exekutatuko duen *ROS nodo* bat da. Zerbitzari programak bezala, honek ere hiru ataza bete behar ditu:

- **Sentsoreen informazioa** eskuratu eta publikatu: Zerbitzaritik sentsoreren baten informazioa jasotzen bada hau eskuratu eta *topic* batean publikatu behar du. 4.1 taulan irudikatzen den bezela, sentsorea konektatua dagoen portuaren arabera *topic* desberdin batetan publikatuko da informazio hau:

Sarrera portua	*Topic* izena
1	*ev3_in1*
2	*ev3_in2*
3	*ev3_in3*
4	*ev3_in4*

4.1 Taula: Sentsoreen sarrera portu eta sentsoreen informazioa jasotzen duten *topic*en arteko erlazioa

  Topic hauek beste nodoetatik eskuragarri egongo direnez, erabiltzaileak bere programan irakurri nahi duen sarrera portuari dagokion *topic*a irakurtzea besterik ez du egin behar bertan konektatua dagoen sentsoreak neurtzen duen balioa jakiteko. Sentsore bat konektatua baldin badago zerbitzari programatik etengabe jasoko da informazioa eta, ondorioz, informazio hau *nodo*aren exekuzioak dirauen artean etengabe eguneratuko da.

- **Motorretan idatzi beharreko informazioa** eskuratu eta zerbitzari programari bidali: *Nodo*ak *topic* zehatz batzuetan informaziorik idatzi dela atzematen badu *topic* honetan idatzitako informazioa zerbitzari programari bidaliko dio honek motorren abiadura alda dezan. Sarrerako portuen kasuan bezela irteerako portuentzat ere lau *topic* erabiliko dira motorren informazioa jasotzeko eta 4.2 taulan ikus daitekeen bezela irteera portu bakoitzari *topic* zehatz bat dagokio.

Irteera portua	*Topic* izena
A	*ev3_outA*
B	*ev3_outB*
C	*ev3_outC*
D	*ev3_outD*

4.2 Taula: Sentsoreen irteera portu eta motorren abiadura jasotzen duten *topic*en arteko erlazioa

Sarrera portuekin bezela *topic* hauek beste nodoetatik atzigarri egongo dira. Hortaz, erabiltzaileak motor baten abiadura aldatu nahiko balu, motorra konektatua dagoen portuari dagokion *topic*ean motorrari eman nahi dion abiadura idaztea nahikoa izango da. Balio posibleak -100 eta 100 artekoak dira. Balio negatibo bat idatzi ezkero motorra erlojuaren orratzen aurka biratuko da emandako abiaduran, 0 idatzi ezkero motorra ez da mugituko eta balio positibo bat eman ezkero erlojuaren orratzen zentzuan.

- **Odometria informazioa** eskuratu eta publikatu: Motorren bat konektatua izan bada zerbitzari programak bezero programari etengabe honen odometria informazioa bidaliko dio. Zehazki motorraren uneko posizioa jasoko da bezero programan. Behin informazio hau jasota dagokion *topic*ean publikatuko da. Sarrera eta irteera portuekin gertatzen den bezela, motorra konektatua dagoen irteera portuaren arabera *topic* zehatz batean publikatuko da informazio hau. Erlazioak 4.3 taulan ikus daitezke.

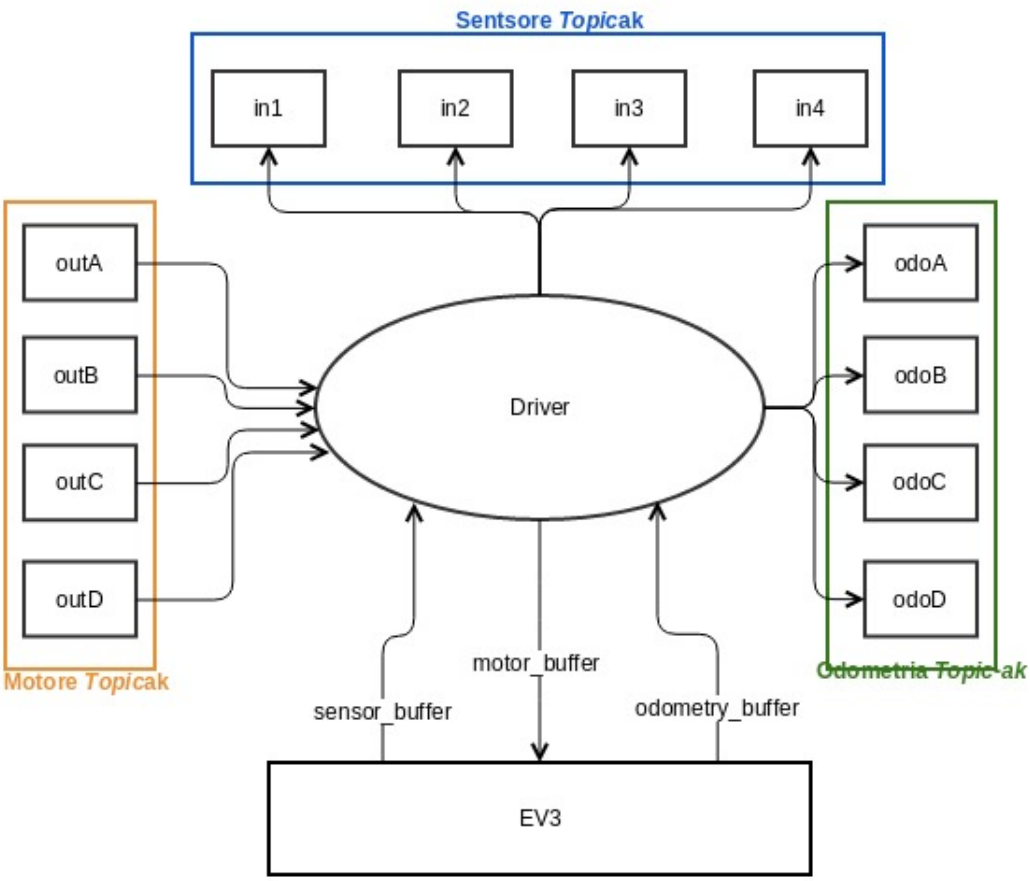
Irteera portua	*Topic* izena
A	*ev3_odoA*
B	*ev3_odoB*
C	*ev3_odoC*
D	*ev3_odoD*

4.3 Taula: Sentsoreen irteera portu eta odometriaren informazioa jasotzen duten *topic*en arteko erlazioa

Aurreko kasuetan bezela *topic* hauek beste nodoetatik atzigarriak izango dira. Ondorioz erabiltzaileak bere programan odometria informazioa eskuratu nahiko balu, motorra konektatua dagoen portua ezagutuz, dagokion *topic*a irakurri besterik ez du egin beharko.

Informazio honekin motorraren biraketa kopurua, abiadura eta angelua jakin daitezke.

4.1 irudiak *driver*arren funtzionamenduaren eskema irudikatzen du. Ikus daitekeenez *nodo* honen exekuzioak dirauen artean bost prozesu exekutatu beharko lirateke aldi berean. *Nodoa* bera, sentsoreen informazioa jasoko duen prozesua, motorren informazioa idatziko duen prozesua (eta honekin batera abiadura jasotzen duen *topic*a etengabe irakurtzen duena) eta odometria informazioa jasoko duen prozesua. Prozesu hauen paralelizazioa lortu eta memoria karga arintzeko helburuarekin prozesu umeak sortu beharrean hariak erabiliko dira, bai bezero eta baita zerbitzari aldean, sentsore, motor eta odometria prozesuetarako.



4.1 Irudia: *Driver*raren diseinuaren eskema

Datu transferentzia

Aurreko ataletan aipatu den bezela bezero eta zerbitzari programen arteko transferentzia etengabea existitzen da. Robotaren funtzionamendu zuzena bermatzeko informazioa bi aldeetara ondo bidali eta heltzen dela ziurtatu behar denez *TCP* konexio bidez bidaliko da informazioa.

Kodearen egitura

1. **Zerbitzaria**:

   Zerbitzaria *TCP* Zerbitzari konkurrentea izango da, bezero aldetik nahi adina konexio onar ditzan honen exekuzio bakar batekin. Ondorioz, zerbitzarian hurrengo pausoak jarraitu behar dira:

   - *Socket*a sortu
   - *Socket*ari helbide bat esleitu
   - *Socket*a entzute *socket* gisa ezarri
   - Begizta infinitua
     - Blokeatu konexio eskaera baten zain
     - Prozesu berria sortu
       - **Prozesu gurasoa:**
         - Elkarrizketa *socket*a itxi
       - **Prozesu umea:**
         - Entzute *socket*a itxi
         - Bezeroarekin elkarrizketa *socket*aren bitartez komunikatu
         - Elkarrizketa *socket*a itxi
         - amaitu

   Bezeroarekin elkarrizketa *socket* berriarekin komunikatzerakoan protokoloan zehaztutako eragiketak beteko dira. Hau da, protokoloan azaltzen den komandoetako bakoitza zerbitzarira heltzean, honek zehaztutako emaitzak iztuli ahal izateko egin beharreko eragiketak betetzea.

Komunikazio honen hasieran, zerbitzariak jasotako mezuaren amaieran \0 karakterea txertatzen du *string*en kudeaketak arazorik eman ez dezan.Ondoren, komando bakoitzarekiko berezkoa den betebeharra aurrera eramaten du.

Komandoetako bakoitzaren nondik norakoak hurrengoak dira:

(a) **INP**: Bezeroak sentsoreen informazioaren eskaera egin duela adierazten du. Hau konexioa ezartzerakoan egiten den lehenengo gauzetarikoa da. Zerbitzariak, komando hau jasotzean, bezeroari etengabe sentsoreen informazioa bidaliko dion haria exekutatzen du. Hari honek, *EV3*an konektatutako sentsore bakoitzerako, hau konektatua dagoen sarrera portua eta sentsore horren neurketa bidaltzen dizkio bezeroari.

(b) **OUT**: Erabiltzaileak motorren abiadura aldatu nahi duela adierazten du. Zerbitzariak komando hau jasotzean motorren abiadura aldatzeaz arduratuko den haria exekutatuko du. Bezeroak, eragin nahi duen motor bakoitzerako hau konektatua egon beharko den portua eta motorrari eman beharreko abiadura bidaltzen du. Erabiltzailearen ardura da adierazitako portuan motor bat dagoela (eta motor zuzena dela) ziurtatzea. Datu hauek jaso ondoren, zerbitzariak, esandako irteera portuan motor bat konektatua badago emandako abiadura ezarriko du -100 eta 100 balioen artean badago. Balio horien artean egon ezean balioa bornatu eta motorrari abiadura moduan ezarriko dio. Balioa -100 baina txikiagoa bada, -100 balioa emango zaio. 100 baino handiagoa bada, berriz, 100 balioa. Pasatako portuan ez badago motorrik ez da ezer egingo. Hau guztia egin ondoren bezeroari OK mezua bidaliko zaio.

(c) **ODO**: Bezeroak odometriaren informazioaren eskaera egin duela adierazten du. Hau konexioa ezartzerakoan egiten den lehenengo gauzetarikoa da. Zerbitzariak, komando hau jasotzean, bezeroari etengabe odometriaren informazioa bidaliko dion haria exekutatzen du. Hari honek, *EV3*an konektatutako motor bakoitzerako, hau konektatua dagoen irteera portua eta motor horren uneko posizioa bidaltzen dizkioo bezeroari.

2. **Bezeroa**: Bezeroak, zerbitzariarekin komunikatzeko, beharrezkoak dituen oinarrizko TCP konexioak ezarriko ditu. Konexio hauek hurrengo pausoak jarraituko dituzte:

   - *Socket*a sortu
   - *Socket*a zerbitzariko *socket* batekin konektatu
   - *Socket*aren bitartez komunikatu
   - *Socket*a itxi

   Esan bezela bezeroak hiru motatako konexioak ezarriko ditu zerbitzariarekin: Sentsoreen neurketak jasotzeko konexio bat, motorren odometria informazioa eskuratzeko konexio bat eta motorrei abiadura aldatzeko konexio bat. Lehenengo biek etengabeko transferentzia egiten dutenez ezinezkoa da hiru ataza hauek konexio bakarrean egitea, hortaz, lan hau paraleloan beteko da konexio bakoitzerako hari bat sortuz.

   Komandoetako bakoitzaren nondik norakoak hurrengoak dira:

   (a) **INP**: Bezeroak zerbitzariari sentsoreen neurketak jasotzeko transferentzia hasteko prest dagoela jakinarazten dio komando honen bidez. Komando hau parametrorik gabe pasako da. Eskaera egiten denetik konexioa ixten den arte, bezeroak zerbitzariak bidaltzen dizkion *buffer*etatik konektatuta dauden sentsoreen informazioa erauziko du. Informazio hau dagokion sarrera portuarentzako *topic*ean publikatu ondoren, zerbitzariari OK erantzuna bidaltzen zaio.

   (b) **OUT**: Bezeroak zerbitzariari motorren baten abiadura aldatu nahi duela jakinaratzen dio komando honen bidez. Zerbitzariak OK mezua bidaltzen dionean bezeroak, abiadura aldatu nahi zaion motor bakoitzeko beharrezko informazioa bidaltzen dio zerbitzariari eta. Ondoren zerbitzariak OK mezua jaso arte itxaroten du.

(c) **ODO**: Bezeroak zerbitzariari odometriaren informazioa jasotzeko transferentzia hasteko prest dagoela jakinarazten dio komando honen bidez. Komando hau parametrorik gabe pasako da. Eskaera egiten denetik konexioa ixten den arte, bezeroak zerbitzariak bidaltzen dizkion *buffer*etatik konektatuta dauden motorren odometria informazioa erauziko du. Informazio hau dagokion irteera portuarentzako *topic*ean publikatu ondoren, zerbitzariari OK erantzuna bidaltzen zaio.

Protokoloa

- **Sintaxia**:
  Komando, erantzun eta parametro guztiak ASCII kodeko karaketer kateak izango dira. Komando guztiek hiru karaktere izango dituzte eta erantzunak bi. Komandoen luzeera finkoa denez, hauen atzetik zuzenean parametroak joango dia, inongo banaketarik gabe. Parametro bat baino gehiago bidaltzen den kasuetan '@' karakterea erabiliko da parametroak banatzeko. Mezuen amaiera lerro jauzien bitartez adieraziko da.

- **Semantika**:
  Komandoei erreparatuz parametroren bat bidaltzen den kasuen formatua hurrengoa da:

  – **INP**: Sentsore bat konektatua duen sarrera portu bakoitzerako, sentsorea konektatua dagoen sarrera portua eta honek egindako neurketaren balioa, bakoitza '@' karaktereaz jarraituta.

  – **OUT**: Abiadura aldatu nahi zaion motor bakoitzerako, motorra konektatua dagoen irteera portua eta bertan idatzi beharreko balioa, biak '@' karaktereaz jarraituta.

  – **ODO**: Konektatuta dagoen motor bakoitzerako, motorra konektatua dagoen irteera portua eta honen uneko posizioa, bakoitza '@' karaktereaz jarraituta.

Komandoak:

Komandoa	Parametroa(k)	Semantika
INP	Sarrera portu eta neurketa balioaz osatutako bikoteen zerrenda edo ezer ez	Sentsoreen informazioa eskatu.
OUT	Irteera portu eta abiaduraren balioaz osatutako bikoteen zerrenda edo ezer ez	Motorren abiadura ezarri.
ODO	Irteera portu eta odometria balioaz osatutako bikoteen zerrenda edo ezer ez	Odometriaren informazioa eskatu.

4.4 Taula: TCP protokoloaren komandoak

Erantzunak:

Erantzuna	Parametroa(k)	Semantika
OK		Datu transferentzia eta hauen prozesaketaren amaiera

4.5 Taula: TCP protokoloaren erantzunak.

4.1.2 Inplementazioa

*Driver*raren inplementazioak bi atal izango ditu, bezero programa eta zerbitzari programaren inplementazioa.

Bezero programa

Diseinuan esan bezela bezero programa *ROS nodo* bat da, honela erabiltzaileak *ROS* erabilita *LEGO Mindstorms EV3*arekin elkarrekintza ahalbidetzeko. Bezero programak hurrengo pausoak jarraitzen ditu:

- **hasi**
  - *ev3pkg_node nodoa* abiarazi;
  - *IP* helbidea *nodo* parametroetatik eskuratu;
  - Sentsoreen informazioa eskuratzeko haria abiarazi;
  - Motorren abiadurak eskuratzeko haria abiarazi;
  - Motorren abiadurak zerbitzarira bidaltzeko haria abaiarazi;

- Odometria informazioa eskuratzeko haria abiarazi;

- **amaitu**

Bezero programan lau hari hasieratzen dira. Jarraian, hauen inplementazioak jarraitzen dituen pauso nagusiak azaltzen dira:

- **Sentsoreen informazioa eskuratzeko haria:**

  - **hasi**
    - *Socket*a sortu;
    - *Socket*ari helbidea esleitu;
    - Konektatu;
    - Sentsoreen informazioa publikatzeko *ROS publisher*ak sortu;
    - *Buffer*ean *INP* komandoa sartu;
    - *Buffer*ra zerbitzariari bidali;
    - **while** (1) **do**
      - Sentsoreen informazioa zerbitzaritik eskuratu;
      - Sentsoreen informazioa *buffer*retik erauzi;
      - Sentsoreen informazioa *topic*etan publikatu;
      - Zerbitzariari *OK* mezua bidali;
    - **end while**
    - *Socket*a itxi;
    - Haria itxi;
    - exit;
  - **amaitu**

- **Motorren abiadurak eskuratzeko haria:**

  - **hasi**
    - Motorren abiadura jasotzen duten *topic*etatik irakurtzeko *subscriber*rak sortu;
    - *ROS* begizta;
    - Haria itxi;
    - exit;
  - **amaitu**

Callback:
Motorretan abiadura idatzi nahi den edo ez jakiteko beharrezkoak diren *topic*etan etengabe irakurtzen duten callback funtzioak erabiltzen dira.

- **hasi**
  - *Topic*a irakurri;
  - Motorraren abiadura gorde;
- **amaitu**

- **Irteera portuen kudeaketa haria:**

- **hasi**
  - *Socket*a sortu;
  - *Socket*ari helbidea esleitu;
  - Konektatu;
  - *Buffer*ean *OUT* komandoa sartu;
  - *Buffer*ra zerbitzariari bidali;
  - Itxaron OK erantzuna;
  - **while** (1) **do**
    - *Buffer*rean *OUT* komandoa sartu;
    - Idatzi nahi diren motorrak konektatuta dauden portu eta ezarri nahi zaien abiadura *buffer*rean sartu;
    - *Buffer*ra zerbitzarira bidali;
    - Itxaron OK erantzuna;
  - **end while**
  - *Socket*a itxi;
  - Haria itxi;
  - exit;
- **amaitu**

- **Odometria informazioa eskuratzeko haria:**

- **hasi**
  - *Socket*a sortu;
  - *Socket*ari helbidea esleitu;
  - Konektatu;
  - Odometria informazioa publikatzeko *ROS publisher*ak sortu;

- *Bufferean ODO* komandoa sartu;
- *Bufferra* zerbitzariari bidali;
- **while** (1) **do**
  - Odometria informazioa zerbitzaritik eskuratu;
  - Odometria informazioa *buffer*retik erauzi;
  - Odometria informazioa *topic*etan publikatu;
  - Zerbitzariari *OK* mezua bidali;
- **end while**
- *Socket*a itxi;
- Haria itxi;
- exit;

– amaitu

Zerbitzari programa

Zerbitzari programaren inplementazioan hurrengo urratsak jarraitu dira:

- **hasi**
  - *Socket*a sortu;
  - *Socket*ari helbidea esleitu;
  - *Socket*a entzute *socket* gisa ezarri
  - **While**(1)**do**
    - Elkarrizketa *socket*eta konektatu;
    - **fork**:
      - **Prozesu umea**:
        - *Socket*a itxi;
        - Irteera portuetan konektatuta dauden motoreen lista berrezarri;
        - **While**(1)**do**
          - Bezerotik *buffer*ra jaso;
          - *Buffer*retik komandoa erauzi;
          - **if** (Komandoa=*INP*) **then**
            - Sentsoreen informazioa eskuratzeko haria abiarazi;
          - **else if** (Komandoa=*OUT*) **then**
            - Bezeroari *OK* mezua bidali;

- Motorretan informazioa idazteko haria abiarazi;
- **else if** (Komandoa=*ODO*) **then**
- Odometria informazioa eskuratzeko haria abiarazi;
- **end if**
- Elkarrizketa *socketa* itxi;
- exit;

- **Guraso prozesua:**
- Elkarrizketa *socketa* itxi;

- **end fork:**
- exit;

- **end while**
- exit;

- **amaitu**

Bezero programan egin den bezela, hemen ere ataza bakoitza hari batean exekutatzen da memoria karga murrizteko. Jarraian erabiltzen diren harien inplementazioek jarraitzen dituzten pausoak azaltzen dira:

- **Sentsoreen informazioa eskuratzeko haria:**

- **hasi**
- **while**(1)**do**
- Konektatutako sentsoreen lista lortu;
- **for** i=0 **to** 4 **do**
- **if** (sentsorea[i] konektatua) **then**
- Sentsorearen informazioa eskuratu;
- Sentsorearen informazioa *bufferrean* gorde;
- **end if**
- **end for**
- *Bufferra* bezeroari bidali;
- Bezerotik *OK* mezua jaso arte itxaron;
- **end while**
- **amaitu**

- **Motorretan informazioa idazteko haria:**

```
- hasi

  - Konektatutako motor lista lortu;
  - Idatzi nahi den motor baoitzerako egin:

     - if (motorra konektatua) then

        - Abiadura berria ezarri;
        - run_forever();

     - end if

- amaitu
```

- **Odometria informazioa eskuratzeko haria:**

```
- hasi

  - while(1)do

     - Konektatutako motorren lista lortu;
     - for i=0 to 4 do

        - if (motorra[i] konektatua) then

           - Motorraren posizioa eskuratu;
           - Motorraren posizioa bufferrean gorde;

        - end if

     - end for
     - Bufferra bezeroari bidali;
     - Bezerotik OK mezua jaso arte itxaron;

  - end while

- amaitu
```

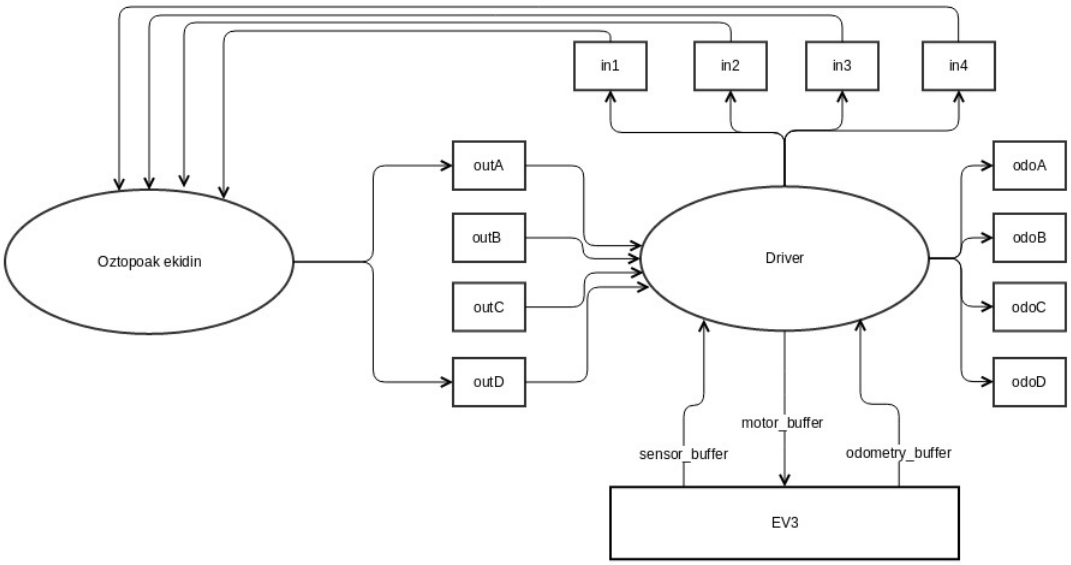
4.2 Oztopoak ekiditeko nodoaren garapena

4.2.1 Diseinua

Oztopoak ekiditeko nodoak helburu bikoitza du. Alde batetik *driver*raren funtzionamendua probatu eta erakustea eta bestetik erabiltzaile berriei *driver*raren funtzionamendurako eredu bat ematea. Sentsore desberdinen arabera motorrak mugi daitezen, prototipoa oztopoak ekiditeko *ROS nodo* bat izatea erabaki da.

Nodo honek sentsoreren bat konektatua dituzten portuak erazagutuko ditu. Gauza bera egingo du motorrekin. Erabiltzailearen ardura izango da portu hauek modu zuzenean erazagutzea. Behin hau eginda, konektatuta dauden sentsoreen informazioa lortzeko dagokion *topic*a irakurriko du eta honen balioaren arabera motorren mugimenduaren kontrolerako *topic*ean nahi den informazioa idatziko du.

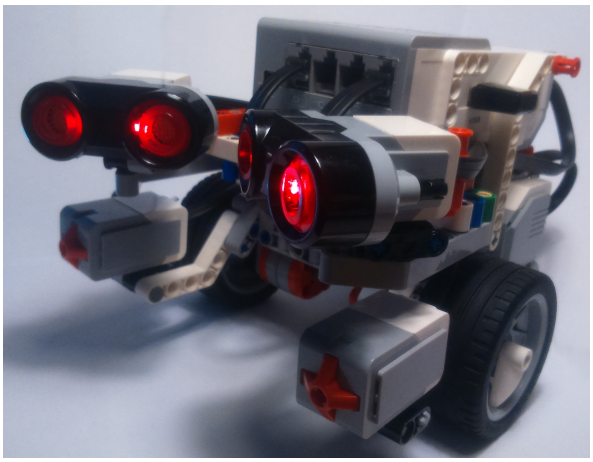
4.2 irudian oztopoak ekiditeko nodoaren funtzionamenduaren eskema ikus daiteke:



4.2 Irudia: Oztopoak ekiditeko nodoaren diseinuaren eskema

4.2.2 Robotaren eraikuntza

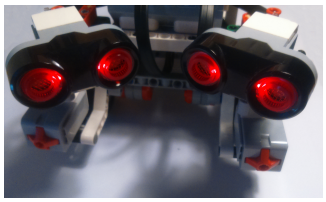
Robotari, oztopoak ekiditeko helburuarekin lau sentsore eta bi motor jarri zaizkio. 4.3 irudian eman zaion itxura ikus daiteke. Hurrengo ataletan hauen kokapena azaltzeaz gain *oztopoak ekiditeko nodo*arekin lotura egingo da.



4.3 Irudia: Robotaren itxura

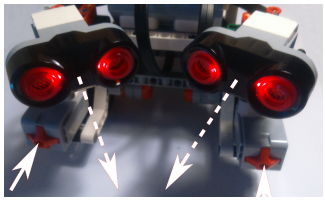
Sentsoreak

Kasu honetan bi talka sentsore eta bi ultrasoinu sentsore konektatu dira 4.4 irudian ikusten den moduan.



4.4 Irudia: Robotaren sentsoreen kokapena

Talka sentsoreen artetik objektu estu edo txikiren bat sar daitekenez ultrasoinu sentsoreak biratuak jarri dira hau ekiditeko. Geziek adierazten duten espazioan objekturik sartuko balitz robotak atzera egingo du ultrasoinuen kokapen honi esker.



4.5 Irudia: Robotaren sentsoreen kokapena (azalpena)

Sentsore eta sarrerako portuen erlazioa 4.6 taulan ikus daitekena da. Talka sentsoreak 1 eta 4 sarrera portuetan eta ultrasoinu sentsoreak 2 eta 3 portuetan kokatu dira.

Sentsorea	Sarrera portua
Ezker talka sentsorea	*in1*
Ezker ultrasoinu sentsorea	*in2*
Eskuin ultrasoinu sentsorea	*in3*
Eskuin talka sentsorea	*in4*

4.6 Taula: Oztopoak ekiditeko nodoaren sentsoreen kokapena

Ondorioz, sentsorearen balioa jakiteko irakurri beharko diren *topic*ak hurrengo taulan azaltzen direnak dira:

Sentsorea	*Topic* izena
Ezker talka sentsorea	*ev3_in1*
Ezker ultrasoinu sentsorea	*ev3_in2*
Eskuin ultrasoinu sentsorea	*ev3_in3*
Eskuin talka sentsorea	*ev3_in4*

4.7 Taula: Oztopoak ekiditeko nodoaren sentsore eta *topic*en erlazioa

Motorrak

Motorrei dagokionez, bi motor handi erabiliko dira robota mugiarazteko. Hauek hurrengo taulan ikusten den bezala konektatuko dira:

Motorra	Irteera portua
Ezker motorra	*outA*
Eskuin motorra	*outB*

4.8 Taula: Oztopoak ekiditeko nodoaren motorren kokapena

Hortaz, motorren abiadurak hurrengo taulan ikusten diren *topic*etan idatzi beharko dira:

Motorra	Irteera portua
Ezker motorra	*ev3_outA*
Eskuin motorra	*ev3_ooutB*

4.9 Taula: Oztopoak ekiteko nodoaren motorren eta *topic*en arteko erlazioa

Gainera, motor hauen odometriaren informazioa jakin nahi izan ezkero hurrengo *topic*en irakurketa egin beharko da:

Motorra	Irteera portua
Ezker motorra	*ev3_odoA*
Eskuin motorra	*ev3_odoB*

4.10 Taula: Oztopoak ekiditeko nodoaren motor eta odometria *topic*en arteko erlazioa

4.2.3 Inplementazioa

Oztopoak ekiditeko nodoaren inplementazioak hurrengo pausoak jarraitzen ditu:

- **hasi**
  - *ev3pkg_avoid_obstacles nodoa* abiarazi;
  - Motorren abiadura *topic*etan idatziko duten *publisher*rak sortu;
  - **while** (1) **do**
    - **if** (Ezker ultrasoinu irakurketa < 10) edo (Eskuin ultrasoinu irakurketa < 10) **then**
      - Atzera ezkerrerantz mugitu;
    - **if** Talka sentsorerik ez dago sakatua **then**
      - Aurreraka mugitu;
    - **else if** Eskuin talka sentsorea sakatua **then**
      - Atzeraka eskuinerantz mugitu;
    - **else**
      - Atzeraka ezkerrerantz mugitu;
    - **end if**
  - **end while**
- **amaitu**

Hau prototipo bat bada ere, orain *EV3*a oso modu sinplean programa daitekeela ikus daiteke aurreko adibidean.

4.3 Exekuzioa

4.3.1 Zerbitzari programa

Zerbitzari programa *daemon* bat izango da eta robota pizten den unetik aurrera exekutatzen arituko da. Script bat sortu da hau behin exekutatuz beharrezko konfigurazioa egiten duena. *ev3_install* scripta exekutatu eta robota berrabiarazi ondoren *driver*ra automatikoki abiaraziko da robota pizten den bakoitzean.

4.3.2 Bezero programa

Bezero programaren exekuzioarako *ev3pkg_driver* paketeak *roslaunch*en laguntzaz *driver*raren nodoa modu zuzenean exekutatzeko *.launch* fitxategi bat dakar (*ev3pkg/launch/ev3pkg.launch*). Hurrengo kode zatian *ROS abiarazlea* ikus daiteke:

- `<launch>`
  - `<node pkg='ev3pkg' name='ev3_driver' type='ev3pkg_node' output='screen'>`
    - `<param name='ip' value='10.42.0.3' type='str'/>`
  - `</node>`
- `</launch>`

Ikus daitekeen bezala *launch* fitxategi honek *EV3*aren *IP* helbidea parametro bezala pasako dio *driverrari*. Erabiltzailearen ardura da nodoari helbide zuzena pasatzen zaiola ziurtatzea. *.launch* fitxategi honi esker, driverra exekutatzeko, driverra instalatua dagoelarik, hurrengo lerroa exekutatzea nahikoa da:

```
roslaunch ev3pkg ev3pkg.launch
```

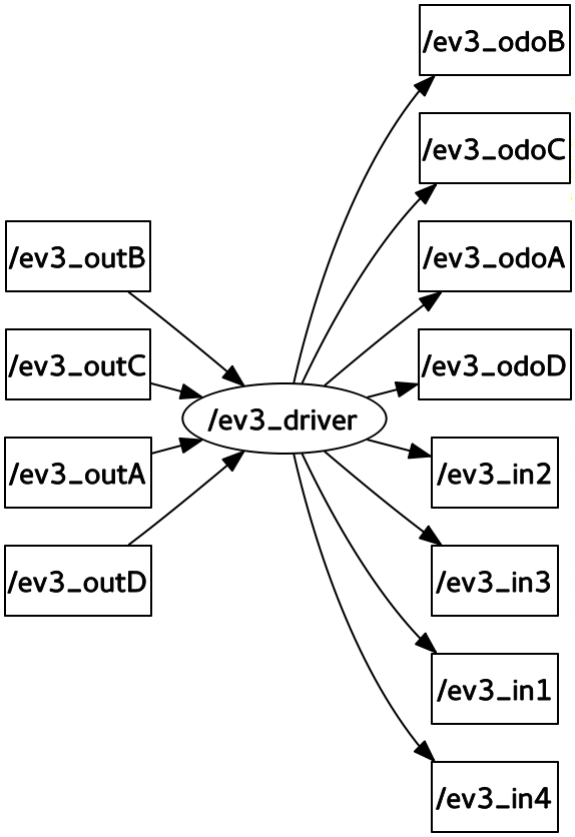
Oztopoak ekiditeko nodoa driverraren pakete berean datorrenez, hau exekutatzea ere oso erreza da *ROS* dei bakar batekin:

```
rosrun ev3pkg ev3pkg_avoid_obstacles
```

4.3.3 *rqt_graph*

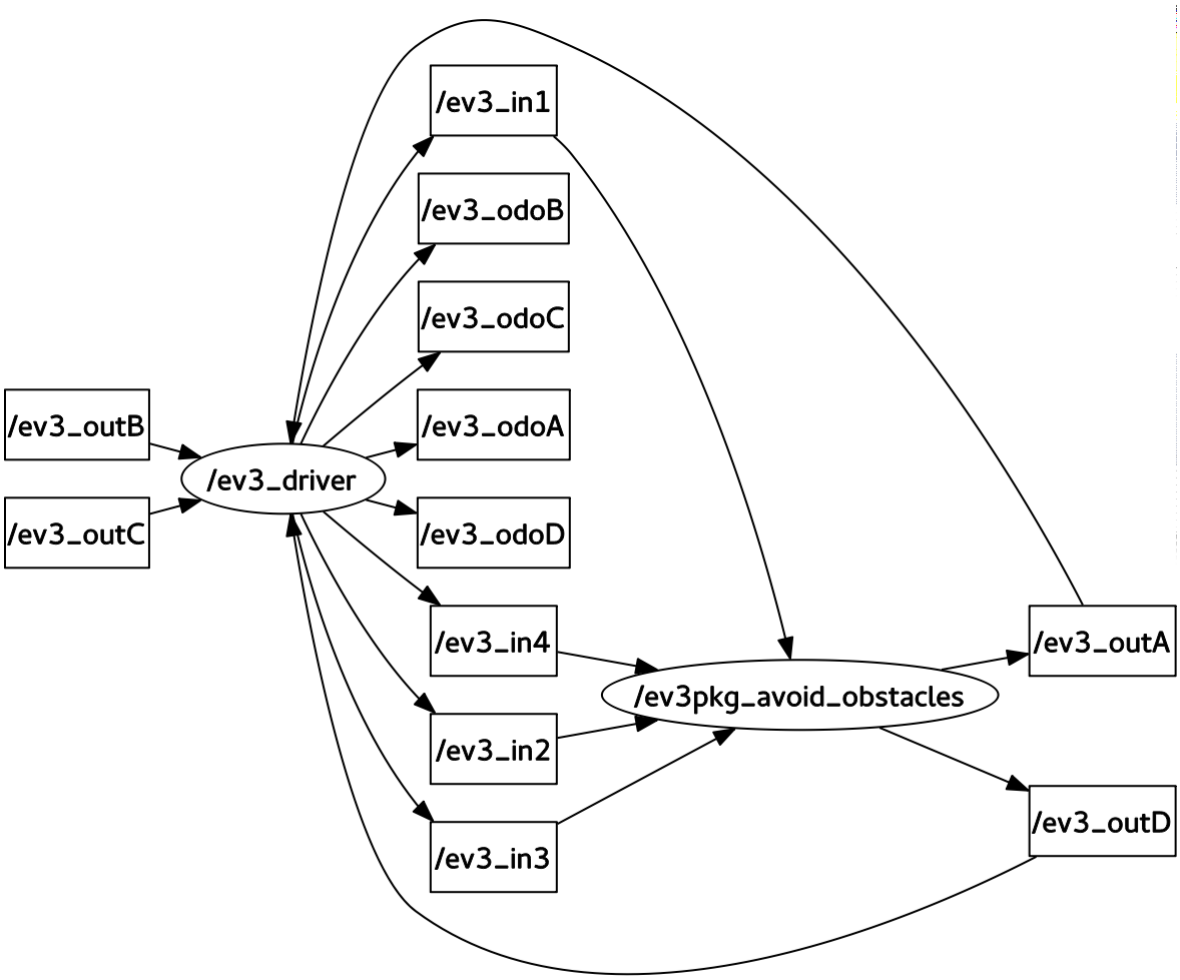
Arestian esan bezala *rqt_graph ROS*en oso tresna erabilgarri bat da. Honi esker, besteak beste, *nodo* eta *topic*en arteko elkarrekintza grafo baten bidez irudikatua ikus daiteke.

Driverraren exekuzioaren ondorioz sortutako grafoa 4.6 irudikoa da:



4.6 Irudia: *Driver* nodoaren exekuzioaren grafoa

Ikus daitekeen bezala, *ev3_driver* nodoak, *ev3_in1*, *ev3_in2*, *ev3_in3*, *ev3_in4*, *ev3_odoA*, *ev3_odoB*, *ev3_odoC* eta *ev3_odoD topic*etan idazten du eta *ev3_outA*, *ev3_outB*, *ev3_outC* eta *ev3_outD topic*etatik irakurri. Ikustagun grafo honek zein aldaketa jasaten dituen Oztopoak ekiditeko nodoa exekutatzean:



4.7 Irudia: *Driver* eta Oztopoak ekidite nodoen exekuzioaren grafoa

4.7 irudian *driver*rraren exekuziotik sortutako grafoak gehikuntza batzuk izan dituela ikusten da. Oztopoak ekiditeko nodoak (*ev3pkg_avoid_obstacles*), *ev3_outA* eta *ev3_outD* nodoetan idazten du eta *ev3_in1*, *ev3_in2*, *ev3_in3*, *ev3_in4 topic*etatik irakurri.

Egoera honek proiektuan garatu nahi zen driverraren portaera islatzen du, *driver*rak sentsoreen informazio eta motorren odometria informazioa publikatzen dituen bitartean erabiltzaileak motorren abiadura aldatu nahi duen irakurtzen du. Oztopoak ekidin nodoak, aldiz, sentsoreen neurketen irakurketak egiten ditu eta irakurritakoaren arabera motorren abiadura aldatzen du.

Grafo honi esker driverra eta Oztopoak ekiditeko nodoek bere helburua betetzen dutela modu argi eta errazean ikusten da.

4.4 Probak

*Driver*raren funtzionamendu zuzena bermatzeko hurrengo probak egin dira:

1. **Sentsoreak**:
    - Sentsorerik ez konektatu.
    - Sentsore kopuru desberdinak konektatu.
    - Sentsoreak konektatu eta deskonektatu.
    - Sentsoreak portu desberdinetan trukatu.
    - Sentsore bakoitzerako balio desberdinak lortzera behartu.
2. **Motorrak**:
    - Motorrik ez konektatu.
    - Motor-kopuru desberdinak konektatu.
    - Motorrei abiadura desberdinak ezarri.
    - Motorrei borneetatik kanpoko balioak ezarri abiadura ezartzeko orduan.
    - Motorrak konektatuta dauden portuak trukatu.
    - Motorrak konektatu eta deskonektatu.

3. **Odometria**:

    - Motorrik ez konektatu.
    - Motor-kopuru desberdinak konektatu.
    - Motorrak konektatuta dauden portuak trukatu.
    - Motorrak konektatu eta deskonektatu.
    - Motorrak eskuz manipulatu odometria balioak zuzenak direla ziurtatzeko.

4. **Datu transferentzia**:

    - *Buffer*retan balio desberdinak bidali.
    - *Buffer*retan balio zuzenak bidaltzen direla ziurtatzeko hauen balioa inprimatu.

5. **Exekuzio probak**:

    - Luzapen desberdinetan robota martxan mantendu.
    - Oztopoz betetako ingurune batean probak egin.

5. KAPITULUA

Jarraipena eta kontrola

Kapitulu honetan proiektuaren jarraipen txostena azaltzen da. Hasieran planifikatu zen denbora eta erabilitakoa alderatuko da, beharrezkoa denean azalpenak emanik. Ondoren, kalitatearen, komunikazioaren eta arriskuen kudeaketa azalduko da.

5.1 Burututako Lana

Proiektuaren hasieran 317.5 ordu beharko zirela estimatu zen, bukaeran 280 izan dira. 5.1 taulan ikus daiteke ataza bakoitzean zenbat ordu estimatu zen, zenbat behar izan diren eta zenbateko desbideratzea gertatu den.

Desbideratze nabarmenenak aplikazioa garatzerakoan eta teknologiaren azterketan gertatu dira.

Ataza	Estimatutako orduak	Erabilitako orduak	Desbideratzea (%)
Planifikazioa	30	20	-33.33
Teknologiaren azterketa	22	25	13.65
Driverraren garapena	152.5	140	-8.19
Kudeaketa	25	25	0
Memoria	50	50	0
Defentsa	20	20	0
Guztira	317.5	280	-11.81

5.1 Taula: Proiektuan erabilitako orduen desbideratzeak

5.1.1 Desbideratze nagusien arrazoiak

Desbideratze handienak *driver*raren garapen prozesuan gertatu dira:

Teknologiaren azterketaren desbideratzearen arrazoiak

Teknologiaren azterketa fasean oso goiz azaldu ziren arazoak. *Robot Operating Systemen Catkin* ingurunea instalatzeko orduan errore asko azaldu ziren eta hau instalatzeko tutorearen laguntzarekin ere bere denbora eskatu zuen.

*Driver*raren garapenaren desbideratzearen arrazoiak

*Driver*raren hasierako diseinuak ez zuen odometria hari bat erabiltzen. Honen ordez *geometry_msgs/Twist* motako *topic* batean motorrak izango duen abiadura lineal eta angularra idazten zituen eta hauetan oinarrituta motorrean idatzi beharreko abiadura 0 eta 100 arteko zenbaki batean idatzi behar zen. Abiaduraren kalkulu honetan gehiegizko denbora pasa ondoren *driver*raren bezero nodoaren egitura aldatzea erabaki zen. Motorretan idazten duen hariak orain 0 eta 100 arteko zenbaki bat hartzen du eta hari berri bat inplementatu da odometriaren informazioa jaso eta *topic*etan publikatzen duena. Egitura aldaketa honek atzerapen bat ekarri zuen, izan ere, motorren abiadurarekin zerikusia zuten atal guztietan aldaketak egin behar izan baitira.

Bestalde, kodeak hainbat egokitzapen jasan ditu. Besteak beste, hasieran motorrak portu zehatz batean konektatu behar ziren eta amaieran *driver*ra generiko egin da erabiltzaileak motorrak irteerako zein portutan konektatzea erabakitzen duelarik. Honek, hala ere, aldaketa sakonak dakartza bai nodo mailan bai eta TCP komunikazio mailan ere.

Azkenik, *driver*raren garapenaren amaieran informaziorik ematen ez zuen errore bat azaldu zen, *ROS*en *segmentation fault* bat. Esan bezala, errore honen informazioan *Segmentation fault* motakoa zela besterik ez zuen esaten eta *ROS*en arazten jakin gabe kodean trazak idaztea bezalako praktikak erabili behar izan ziren errorea aurkitu ahal izateko. Errore hau, gainera, ez zen bat ere zehatza, batzuetan robota martxan jarri eta segundo batzuetara gertatzen zen, bestetan, berriz, ordubetera. Ez zuen eredu zehatz bat jarraitzen. Ondorioz probak ere luzatu ziren. Izan ere, exekuzio garaian errore bat suertatu ezkero jakin zitekeen errorea ez zela desagertu, baina ordubeteko exekuzio proba baten ostean errorerik ez gertatzeak ez zuen zertan esan nahi errorea konpondua zegoela. Ondorioz proba gehiago eta luzeagoak egin behar izan dira errore hau konpondu dela ziurtatzeko.

5.1.2 Atazak garatzeko datak

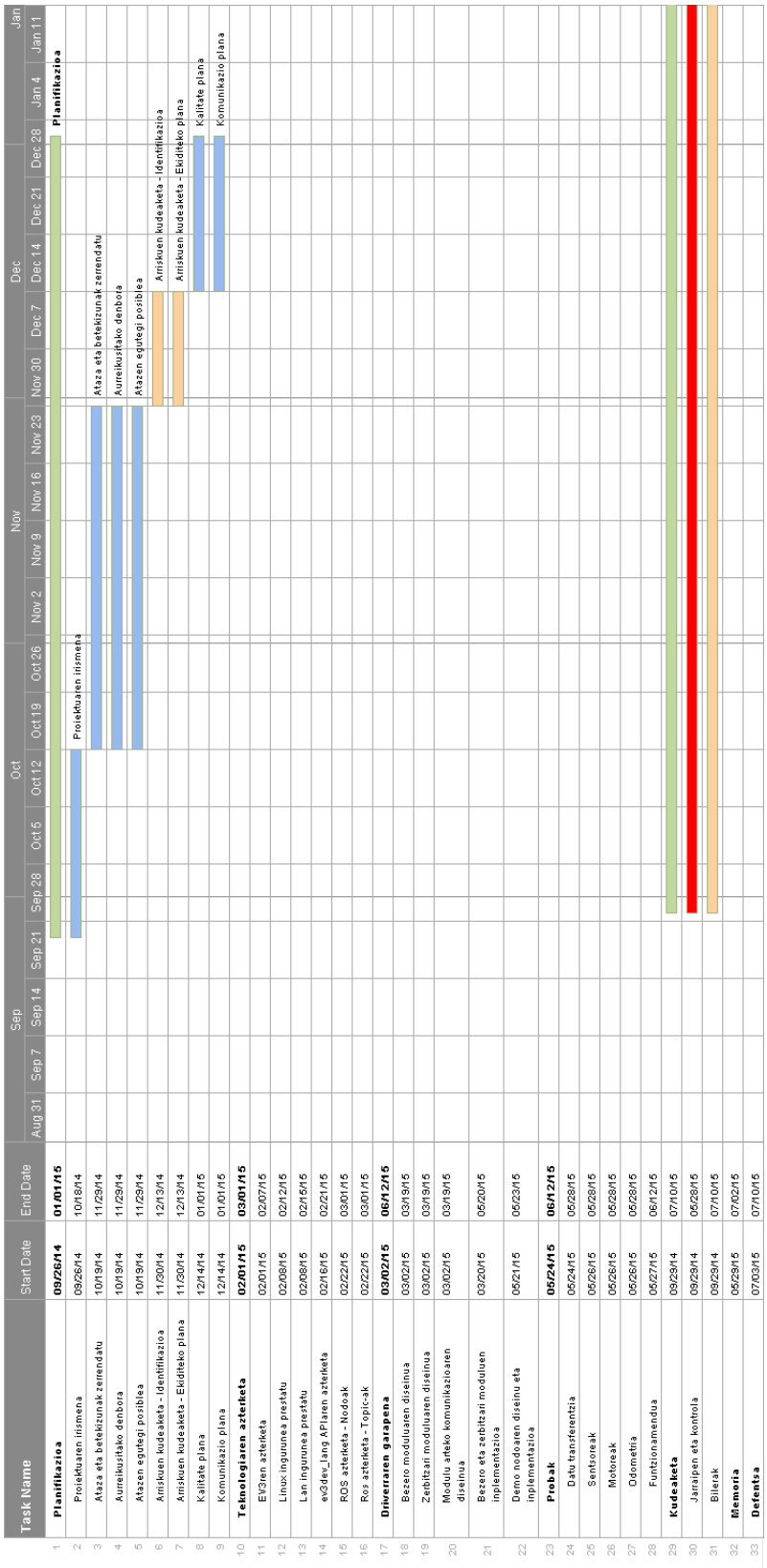
Proiektuaren garapenean zehar denbora desbideratzeak egon arren, ezarritako data limiteak betetzea lortu da. Are gehieago, 5.1 taulan ikus daitekeen bezala, erabilitako ordu kopurua ez da, askotan, estimatutakoa baina handiagoa izan. Hala ere, atazak amaitzeko estimatutako epea askotan luzatu behar izan da.

Hasieran proiektuaren garapenaren zati handiena bigarren lauhilabetean egin beharko nuela kontuan izan nuen, lehenengo lauhilabetean fakultateko lan karga eta praktikak zirela eta . Praktikak berez 2014eko Abenduaren amaieran amaitzen zirenez eta Urtarrilean azterketak izanda 2015eko Martxotik aurrera denbora gehiago izanda puntu honetan sartuko lirateke ordu gehienak. Praktikak amaitu ondoren, hauek bi hilabete gehiago luzatu eta ondoren kontratua sinatzeko aukera eman zidaten. Honetaz gain, fakultatean bigarren lauhilabetean izango zen lan karga gaizki aurreikusi zen, uste zena baino lan karga handiagoa izan zelarik. Ondorioz bigarren lauhilabeteko lanaren birplanifikazioa egin behar izan zen. 5.1, 5.2 eta 5.3 irudietako diagrametan kolore ilunago batez (gorriz) ikus daitezke aldaketak jaso dituzten atazak.

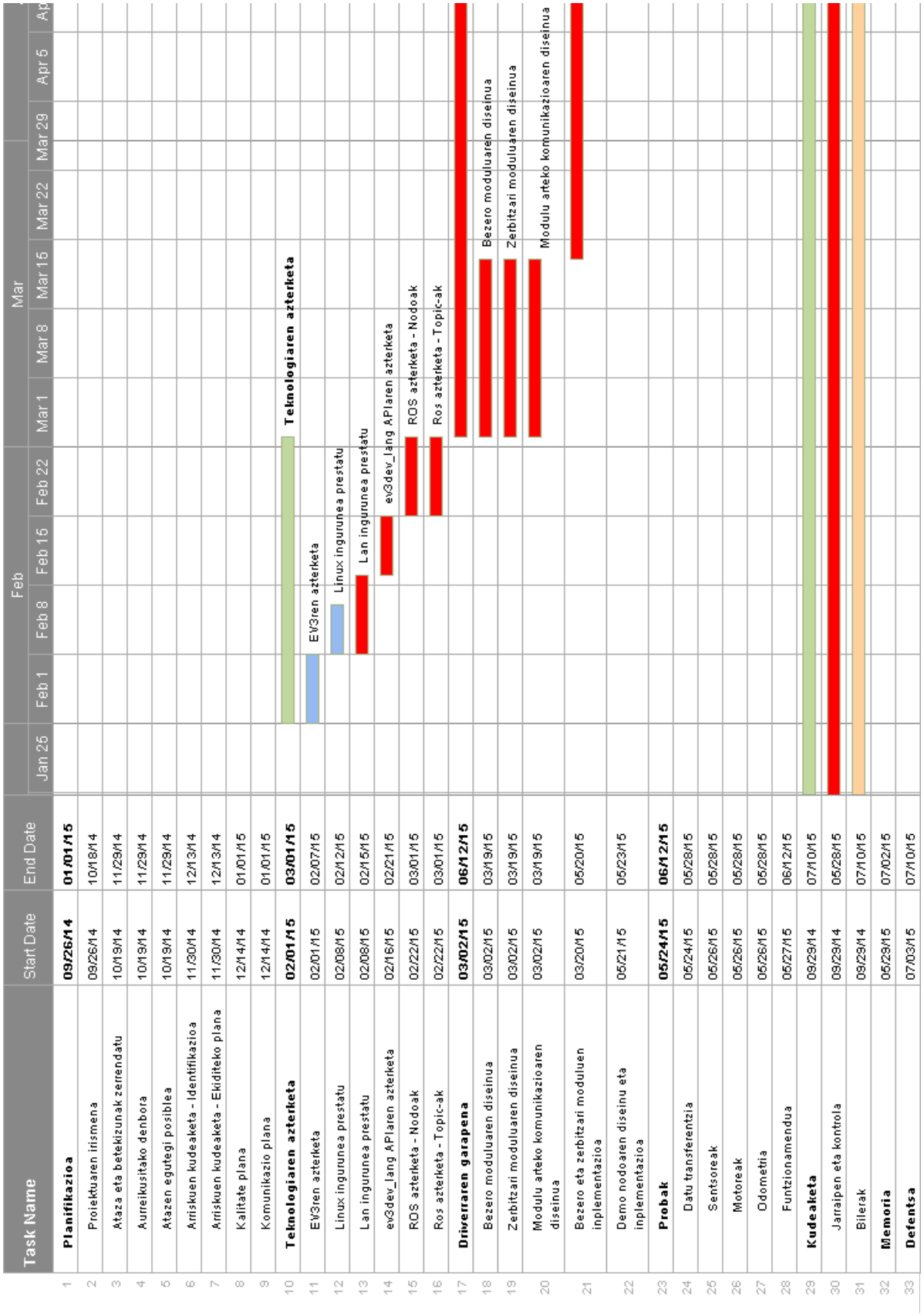
Honek, noski, bere atzetik datozen atazak atzeratu zituen, baina, esan bezala honek ez du esan nahi proiektuari ordu gehiago eman behar izan diodanik.

5.2 Komunikazioa

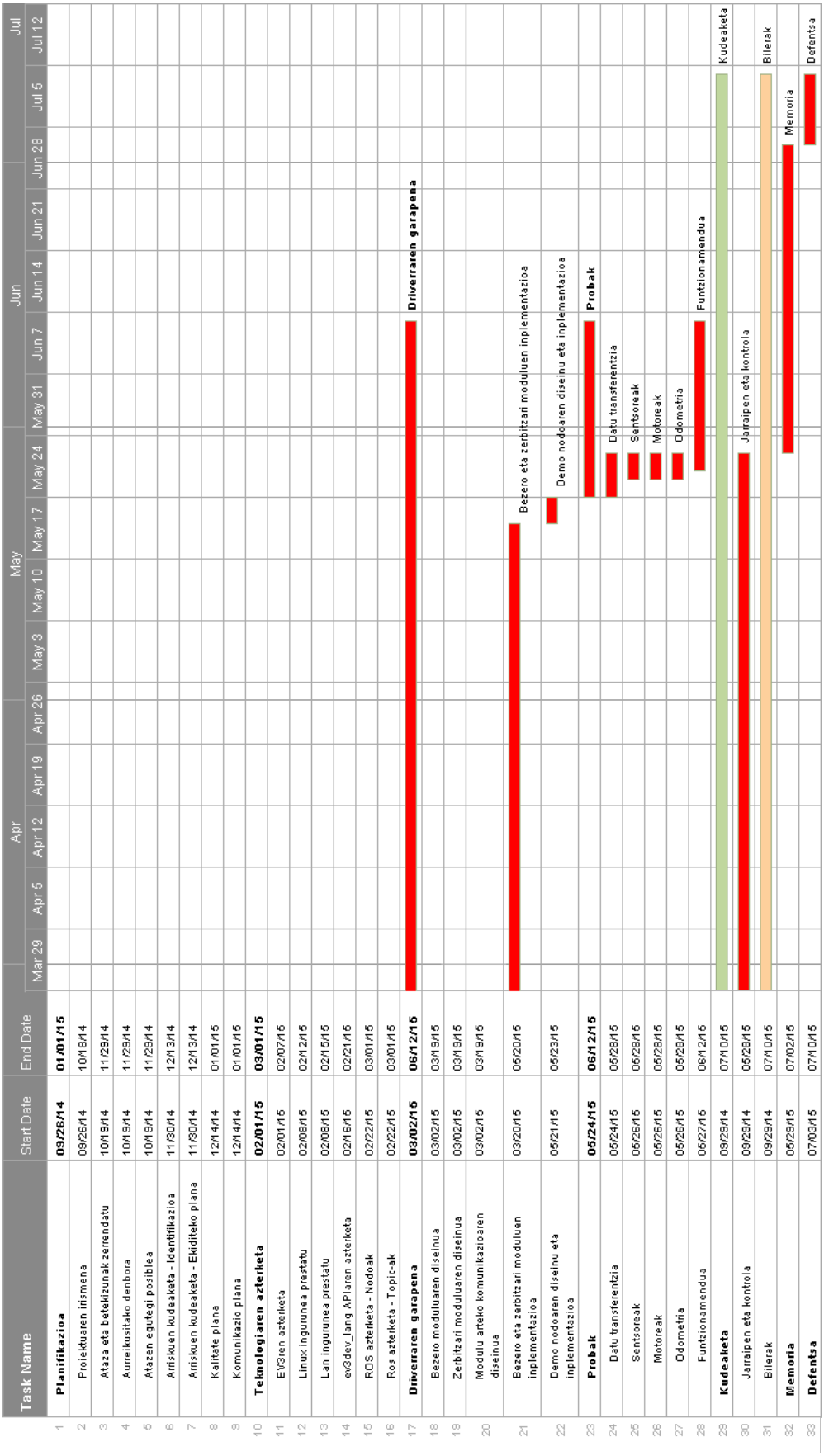
Proiektuaren hasieran komunikazio bide erabiliena mezu elektronikoa izatea planifikatu zen, baina proiektuan interesatuak egilea eta tutorea izanik gehiagotan tutorearekin zuzenean hitz egitea erabaki da, honela, dudak galdetzeaz gain tutorea proiektuaren garapena zein puntutan zegoen informatu ahal izateko.



5.1 Irudia: Burututako lanaren Gantt diagrama (2014ko Iraila - 2015eko Urtarrila)



5.2 Irudia: Burututako lanaren Gantt diagrama (2015ko Otsaila - 2015eko Apirila)



5.3 Irudia: Burututako lanaren Gantt diagrama (2015ko Apirila - 2015eko Uztaila)

5.3 Kalitatea

Proiektuaren hasieran kalitate kontrolak izan behar dituen lau atal identifikatu ziren.

5.3.1 Komunikazioa

Aurreko atalean azaldu den bezala, interesatuekiko komunikazioen minimoak bete direnez kalitatezko komunikazioa izan dela esan daiteke. Tutorea denbora guztian informatua mantendu da eta zerbait behar izan den bakoitzean posta elektronikoz komunikazioa mantendu da.

5.3.2 Produktua

Proiektuaren irismenean zehaztutako minimoak betetzeaz gain memoriaren erabilera ere kontuan izan da. Bezero programak exekutatu behar duen lan bakoitzerako prozesu berri bat erabili beharrean hariak erabili dira hauen exekuziorako. Gainera, zerbitzari aldeko programak TCP konexio konkurrenteak onartzen ditu, honela exekuzio bakarra behar delarik datu transferentzia ahalbidetzeko.

Gainera, *ROS* hobeto ulertzen joan ahala nodoek hobekuntzak jasan dituzte. Honela nodoak generikoak izatea lortu da erabiltzaileari askatasun maila handiagoa emanik. Gainera, *driver*raren nodoaren abiarazlean (*.launch* fitxategian) robotaren *IP* helbidea aldatzea nahikoa izatea ere lortu da kodearen birkonpilaketa beharrezkoa izan ez dadin *IP* desberdinetarako.

Ondorioz, garatutako *driver*raren funtzionalitateek kalitate maila altua dutela esan daiteke. Kalitate maila honen ondorioz kode irekiko komunitatearentzako errepositoriora igotzea erabaki da *LEGO Mindstorms EV3*a *linux* ingurunean programatzean interesatua dagoen edozeinek erabili eta *driver*raren garapena aurrera eraman ahal izateko nahi izan ezkero.

5.3.3 Memoria

Hasieran memoriaren kalitate onarentzako definitutako puntuak betetzen dira. Memoria ondo idatzia, EHUk dokumentuentzako eskatutako arauak betetzen ditu eta ulergarria dela esan daiteke eta tutorearen oniritzia jaso duenez hau kalitatezko memoria dela esan daiteke.

5.3.4 Defentsarako gardenkiak

Memoriaren kasuan bezala hasierako puntu guztiak bete dira. Ondorioz gardenkien kalitatea ere bermatua geratzen da

5.4 Arriskuak

Proiektuaren hasieran kaltegarriak izan zitezkeen hainbat egoera identifikatu eta hauei aurre egiteko plan batzuk zehaztu ziren. Arazo batzuk sortu badira ere, amaieran dena ondo amaitu da. Batzuetan gerta zitekeenaren aurreikuspena bete ez delako eta besteetan gertatutakoan plana jarraituz konponbidea eman zaiolako.

Konputagailua hondatzea

Definitutako hartu beharreko neurriei esker ordenagailuari ez zaio ezer gertatu.

Robota hondatzea

Robota bera hondatu ez bada ere, honen osagaietako bat hondatu zen. Hasieran bateria blokea hondatu zela pentsatu zen. Tutoreekin hitz egin ondoren bateria bloke berria eman zidaten baina hau agortzean hau ere kargatzen ez zela ikusita kargadorea hondatu zela ikusi zen. Tutoreekin hitz egitea nahikoa izan da edozein kasutan arazo hauek konpontzeko eta ez dute proiektuaren garapenean eraginik izan.

Informazio galera

Nahigabe fitxategiren bat ezabatua izan da eta besteren batean fitxategiaren atalen bat guztiz ezabatua ere. Segurtasun kopia eta bertsio kontrolari esker ez da ondorio txarrik izan.

Aurreikuspenak ez betetzea

Praktikak eta aurkitutako arazoak medio proiekuaren atal batzuk atzeratu badira ere, atal bakoitzean utzitako marjinari esker mugarri guztiak betetzea lortu da.

6. KAPITULUA

Ondorioak eta etorkizunerako lana

Kapitulu honetan proiektuaren ondorioak azaltzen dira, aplikazioaren garapenaren laburpena, proiektuan zehar ikasitakoa, merkaturatze aukerak eta etorkizunerako lana azalduko dira bertan.

Ondorioak eta etorkizunerako lanaren egitura
1. Ondorioak 2. Ikasitako lezioak 3. Merkaturatze perspektibak 4. Etorkizunerako lana

6.1 Ondorioak

Proiektu honetan egindako lanaren ondorioz *Linux* eta *LEGO Mindstorms EV3*ren arteko konektibitatea lortu da. Hots, erabiltzaileak bere ordengailutik, *Linux* ingurune batean *ROS* erabiliz *LEGO Mindstorms EV3* robotak programa ditzakeenez, proiektuaren helburua bete da.

Proiektuaren garapenaren lehen urratsa egoera azterketa eta beharrezkoa izango zen teknologiari buruzko informazio bilaketa izan da. Fase hau orokorrean ikaskuntza fase bat izan da. Azken finean *driver* bat egin dudan lehen aldia da eta *Robot Operating System* ere aurretik ez nuen erabilia. Ikasketa lan handiena *ROS* erabiltzen ikastea izan da.

Behin erabiliko zen teknologia ezagutu eta oinarrizko ezagutza bat lortuta hurrengo pausua *driver*raren diseinu eta inplementazioa izan da. *Driver*rak jorratzen dituen ezaugarri bakoitzaren (harien) diseinua bakoitza bere aldetik egin behar izan da, hau konposatzen duten bi aldeen (bezero eta zerbitzari) diseinua eta euren arteko komunikazioa ere diseinatu behar izan delarik.

EV3 eta *Linux*en arteko konektibitatea aurretik existitzen ez zenez eta hau, esan bezala, sortu dudan lehenengo *driver*ra izanik nire garapen pertsonal eta akademikorako lagungarria izan zaidala esan daiteke.

Orokorrean proiektu honekin asko ikasi dudala esan behar dut. Alde batetik hasieran oso zaila iruditu zitzaidan erronka bati aurre egitea lortu dut, bidean kontzeptu eta teknologia berriak ikasiz. Amaierako emaitza, ona bada ere eta proiektuaren helburuak betetzen baditu ere, pertsonalki *driver*rean funtzionalitate batzuk barneratu gabe geratzeak aho zapore txarra utzi dit.

6.2 Ikasitako lezioak

Proiektuan zehar etorkizunerako baliagarri diren hurrengo lezioak ikasi ditut.

6.2.1 Proiektuen planifikazioaren garrantzia

Unibertsitateko *Proiektuen Kudeaketa* ikasgaian proiektuen planifikazioaren garrantzia irakasten da eta proiektu honen garapenean kontzeptu hauen erabilerak proiektua aurrera eramaten asko laguntzen duela argi ikusi dut. Proiektuen atazak ondo zehaztu eta hauentzako datak zehazteak benetan laguntzen du proiektuaren garapenaren kontrola egin ahal izateko. Honetaz gain, garapenaren zein puntutan zoazen argi ikusten ere laguntzen du.

Datak zehaztuta, askotan hauek betetzea ezinezkoa izan bazait ere mugarri guztiak betetzea lortu da.

6.2.2 Arriskuak identifikatzearen garrantzia

Arriskuak aurrez ondo identifikatu eta prebentzio planen garrantzia ere argi geratu zait proiektu honen garapenean. Honi esker arazoren bat sortu zaidan bakoitzean ez du proiektuaren garapenean eragin kritikorik izan. Arazorik gabe joan da aurrera eta ez dit denbora galdu arazi.

6.2.3 Diseinuaren garrantzia

Nahiz eta unibertsitatean honi berebiziko garrantzia ematen dioten, ukaezina da askotan pausu hau saltatu eta zuzenean programatzen hasten garela. Proiektu honetan zehar *driver*raren betebeharrak eta hau konposatzen duten atazen arteko erlazioak oso argi izatea funtsezkoa izan da. Proiektuak aurrera egin ahala, diseinuan aldaketaren bat edo kontzeptu/erlazioren bat argi izan ezean kodetzen hasi eta punturen batean atzera egin eta idatzitakoa ezabatu edo birmoldatzea ekidin ezina dela ikusi dut.

6.3 Merkaturatze perspektibak

Arestian esan bezala, *Linux* eta *EV3*ren arteko konektibitatea ez zen aurretik existitzen eta proiektuaren lehenengo fasean, informazio biltze fasean hain zuzen ere, foro eta gaiarekin erlazionatutako hainbat *blog*etan ikusi dut jendea honetan interesatua dagoela. *Driver* hau repositoriora igo daiteke software askeko komunitateak erabili eta etorkizunean osa dezan.

6.4 Etorkizunerako lana

Hurrengoak dira etorkizunean *driver*ra osatzeko garatu beharko liratekeen lanak:

1. Garatu gabe geratu diren funtzionalitateen inplementazioa:
    - **Sentsore**en neurketa modua aldatu ahal izatea. Momentuz, hau robotaren pantaila eta botoiak erabilita egin daiteke.
    - **Motor**ren mugitzeko modua aldatu ahal izatea. Posizio absolutu edo erlatibo batetaraino mugitu eta denboraren araberako mugimenduak egin (hau *ROS* bidez egin badaiteke ere motorren komandoetako bat da) ahal izatea.
    - **LED**en kontrola. Hauek piztu eta itzaltzeko, kinuka piztu eta itzaltzeko eta kinuen maiztasuna aukeratzeko ahalmena eman.
    - **Botoi**en kontrola. Botoiek sortutako programan eragina egitea. **Ad:** Erdiko botoiari ematean robota gelditzea.
    - **Bozgorailua**ren kontrola. Soinuak edota esaldiak bozgorailutik entzuteko aukera.
    - **Pantaila**. Pantailan agertzen dena programatu ahal izatea.
2. **Odometria informazioa**ren tratamendua: Une honetan odometria hariak motorraren uneko posizioa bakarrik itzultzen du. Honetatik hainbat datu atera daitezke, hala nola biraketa kopurua, abiadura eta angelua, eta hauen kalkulua ere egiteko geratu da.

Eranskinak

7. KAPITULUA

A Eraskina: *ev3_dev*en sentsore eta motorren kontrolerako fitxategiak

7.1 Sentsoreen fitxategiak

*LEGO Mindstorms EV3*an sentsore bat konektatzen den bakoitzeak karpeta bat sortzen da. Bertan sentsore honen kontrolerako erabiltzen diren hurrengo fitxategiak aurki daitezke:

- **bin_data**(irakurketa): Fitxategi hau irakurtzeak *value<N>* fitxategiaren balio gordinak itzuliko ditu. Balio hauek nola interpretatu ulertzeko *bin_data_format*, *num_values* eta sentsore bakoitzaren dokumentazioa erabili.
- **bin_data_format** (irakurketa): Uneko moduarentzako *bin_data* ren balioen formatua itzultzen du. Balio posibleak hurrengoak dira:
  - **u8**: 8 biteko zeinurik gabeko osokoa (byte).
  - **s8**: 8 biteko zeinudun osokoa (sbyte).
  - **u16**: 16 biteko zeinurik gabeko osokoa (ushort).
  - **s16**: 16 biteko zeinudun osokoa (short).
  - **s16_be**: 16 biteko zeinudun osokoa, big endian

  - **s32**: 32 biteko zeinudun osokoa (int)
  - **float**: 32 biteko koma higikorreko IEEE 754 erreala (float)

- **command** (idazketa): Fitxategi honetan idatzita komando bat bidaltzen zaio sentsoreari. Ikus *commands* agindu erabilgarriak ikusteko.

- **commands** (irakurketa soilik): Zuriunez banandutako zerrenda bat itzultzen du uneko sentsorearentzako agindu erabilgarriekin. Agindurik ez badu onartzen -EOPNOTSUPP itzultzen du.

- **direct** (irakurketa/idazketa): *Lego sensor* klasearekin erabil ezin daitezkeen ezaugarri aurreratuak erabili ahal izateko komunikazio zuzena ahalbidetzen du sentsorearekin. Sentsoreak hau onartzen ez badu -EOPNOTSUPP itzultzen du. Momentuz I2C sentsorearekin bakarrik funtzionatzen du. I2C sentsoreentzako *seek* agindua erabili irakurri edo idatzi nahi den erregistroa ezartzeko, ondoren nahi den byte kopurua irakurri edo idatzi.

- **decimals** (irakurketa): *value*<*N*> atributuen hamartarren kokapena itzultzen du.

- **driver_name** (irakurketa): Sentsorearen izena itzultzen du.

- **fw_version** (irakurketa): Eskuragarri egon ezkero *firmware* bertsioa itzultzen du. Momentuz NXT/I2C sentsoreek bakarrik dute eskuragarri.

- **mode** (irakurketa/idazketa): Irakurtzean sentsorearen uneko neurtzeko modua itzultzen du. *modes* aginduak itzultzen duen zerrendako aukeratako bat *mode* fitxategian idaztean sentsorearen neurketa modua aldatzen da.

- **modes** (irakurketa): Zuriunez banandutako zerrenda bat itzultzen du sentsoreak erabil ditzaken neurketa modu erabilgarriekin.

- **num_values** (irakurketa): Itzuliko den *value*<*N*> atributu kopurua itzultzen du. Balio bat itzul dezaketen aginduek itzuliko dituzten atributu kopurua itzultzen du.

- **poll_ms** (irakurketa/idazketa): Sentsorearen inkesta periodoa itzultzen du milisegundotan. Idazterakoan inkesta periodoa aldatzen du. 0 idazteak inkesta desgaitzen du. -EOPNOTSUPP itzultzen du inkestaren periodoaren aldaketa ez bada onartzen. Oraingoz NXT/I2C sentsoreek bakarrik alda dezakete inkesta periodoa.

- **port_name** (irakurketa): Sentsorea konektatua dagoen portuaren izena itzultzen du. Adib:*in1*.

- **units** (irakurketa): Neurtutako balioarentzako unitatea itzultzen du. String hutsa itzul dezake.

- **value**<**N**> (irakurketa): Sentsoreak neurtutako balio edo balioak itzultzen d(it)u. Ikus *num_values* zenbat balio itzuliko diren ikusteko. N >= *num_values* balioek errorea itzuliko dute. Balioak hamartar zehatzak dira, ikus *decimals* benetako balioa lortzeko zatiketa beharrezkoa den jakiteko.

7.2 Motorren fitxategiak

Sentsoreekin gertatzen den bezela, *LEGO Mindsorms EV3*an motor bat konektatzen den bakoitzean karpeta bat sortzen da. Bertan, motorraren kontrolerako erabiltzen diren hurrengo fitxategiak aurki daitezke:

- **command** (idazketa): Motorraren kontrolatzaileari agindu bat bidaltzen dio. Ikus *commands* eskuragarri dagoen agindu zerrenda ikusteko.

- **commands** (irakurketa): Zuriunez banandutako zerrenda bat itzultzen du erabilgarri dauden aginduekin. Balio posibleak *run-forever*,*run-to-abs-pos*, *run-to-rel-pos*, *run-timed*, *run-direct*, *stop* eta *reset* dira. Litekeena da aginduren bat erabilgarri ez izatea.
  run-forever aginduak motorrak beste agindu bat jaso arte martxan jarriko du.
  run-to-abs-pos position_sp fitxategian azaltzen den posiziora heldu arte martxan jarraitzen du eta ondoren *stop* aginduaz gelditzen du.
  run-to-rel-pos agindua uneko posizioarekiko posizio erlatibora heldu arte mugituko da motorra. Posizio berria uneko posizioa + *position_sp* izango da. Behin posizio berrira helduta motorra *stop_command* agindua erabilita gelditzuko da.
  run-timed aginduak *time_sp* fitxategiko balioak adierazten duen milisegundo kopuruaz motorra martxan mantenduko du eta ondoren *stop_command* aginduaz gelditzen du motorra.
  run-direct aginduak *duty_cycle_sp* fitxategiaren balioak adierazten duen abiaduran mugitzen du motorra. Beste aginduek ez bezala *duty_cycle_sp*ren balioa aldatzeak

zuzenean aldatuko du motorraren abiadura. *stop* aginduak martxan dagoen edozein *run* agindu gelditu ko du.
reset aginduak motorraren parametro guztiak berrezarriko ditu. Honek motorra gelditu ere egingo du.

- **count_per_rot** (irakurketa): Biraketa bakarrean dagoen *tacho count* edo *tick* kopurua itzultzen du. Tick hauek posizio eta abiadura atributuek erabiltzen dituzte eta balio hau biraketak edo graduak tick-etara itzultzeko erabil daiteke.

- **driver_name** (irakurketa): Motorraren *driver*raren izena itzultzen du.

- **duty_cycle** (irakurketa): Motorraren uneko abiadura itzultzen du. Balioak -100 eta 100 artekoak dira.

- **duty_cycle_sp** (irakurketa/idazketa): Idazketak motorraren abiadura aldatzen du. Irakurketa abiadura itzultzen du. Unitateak ehunekoak dira. Balioak -100 eta 100 artekoak dira. Balio hau *speed_regulation* desgaitua dagoenean erabiltzen da.

- **encoder_polarity** (irakurketa/idazketa): Biraketa kodetzailearen polaritatea aldatzen du. Gailuaren *driver*rak ezarri beharreko balioa da hau. Balio hau onartuak ez dauden gailuek bakarrik aldatu behar dute. Erabil daitezken balioak *normal* eta *inversed* dira.

- **polarity** (irakurketa/idazketa): Motorraren polaritatea zehazten du. *normal* polaritatearekin *duty_cycle* positibo batek erlojuaren orratzen norantzan biraraziko du motorra. *inversed* polaritatearekin berriz erlojuaren orratzen aurka. Balio posibleak *normal* eta *inversed* dira.

- **port_name** (irakurketa): Motorra konektatua dagoen portuaren izena itzultzen du.

- **position** (irakurketa/idazketa): Motorraren uneko posizioa itzultzen du. Motorrak erlojuaren orratzen norantzan biratzen duenean posizioa handitu egingo da. Erlojuaren orratzen aurka mugitu ezkero balioa txikitu egingo da.

- **hold_pid/Kd** (irakurketa/idazketa): posizioaren *PID*aren (*proportional-integral-derivative*) deribatuaren konstantea itzultzen du.

- **hold_pid/Ki** (irakurketa/idazketa): posizioaren PIDaren (*proportional-integral-derivative*) integrazioaren konstantea itzultzen du.

- **hold_pid/Kp** (irakurketa/idazketa): posizioaren PIDaren (*proportional-integral-derivative*) konstante proportzionala itzultzen du.

- **position_sp** (irakurketa/idazketa): Idazterakoan *run-to-abs-pos* eta *run-to-rel-pos* aginduentzako posizioa zehazten du. Irakurtzean balioa itzultzen du. Unitateak *tacho count* edo motor *tick*ak dira. *counts_per_rot* balioa erabil daiteke tick-ak biraketa edota graduetara bihurtzeko eta alderantziz.

- **speed** (irakurketa): Motorraren abiadura itzultzen du *tick*/segundotan. *counts_per_rot* balioa erabil daiteke *tick*-ak biraketa edota graduetara bihurtzeko eta alderantziz.

- **speed_sp** (irakurketa/idazketa): Idaztean *speed_regulation* gaitua dagoenean motorraren abiadura tick/segundotan ezartzen du. Irakurtzean balioa itzultzen du. *counts_per_rot* balioa erabil daiteke tick-ak rotazio edota graduetara bihurtzeko eta alderantziz.

- **ramp_up_sp** (irakurketa/idazketa): Idaztean *ramp up* atalasea zehazten du. Irakurtzean balioa itzultzen du. Unitateak milisegundoak dira. Balio positibo bat idazten denean motorraren abiadura %0 tik %100ra *ramp_up_sp* fitxategian dagoen balioak zehazten duen milisegundo kopuruan pasako da.

- **ramp_down_sp** (irakurketa/idazketa): Idaztean *ramp down* atalasea zehazten du. Irakurtzean balioa itzultzen du. Unitateak milisegundoak dira. Balio positibo bat idazten denean motorraren abiadura %100 tik %0ra *ramp_down_sp* fitxategian dagoen balioak zehazten duen milisegundo kopuruan pasako da.

- **speed_regulation** (irakurketa/idazketa): Abiadura erregulazioa gaitu edo desgaitzen du. Gaitua baldin badago motorraren kontrolatzaileak motorraren abiadura *speed_sp*k zehazten duen baliora berdinduko du. Desgaitua baldin badago *duty_cycle_sp*ren balioa erabiltzen da. Balio posibleak *on* eta *off* dira.

- **speed_pid/Kd** (irakurketa/idazketa): Abiadura erregulazioaren PIDaren (*proportional-integral-derivative*) deribatuaren konstantea itzultzen du.

- **speed_pid/Ki** (irakurketa/idazketa): Abiadura erregulazioaren PIDaren (*proportional-integral-derivative*) integrazioaren konstantea itzultzen du.

- **speed_pid/Kp** (irakurketa/idazketa): Abiadura erregulazioaren PIDaren (*proportional-integral-derivative*) konstante proportzionala itzultzen du.

- **state** (irakurketa): Zuriunez banandutako egoera adierazleen zerrenda bat itzultzen du. Egoera adierazle posibleak *running*, *ramping holding* eta *stalled* dira.

- **stop_command** (irakurketa/idazketa): Irakurketak uneko gelditze agindua itzultzen du. Idazketak gelditze agindua zehazten du. Gainera mugimendu agindu baten amaieran motorrarean portaera zehazten du. Ikus *stop_commands* balio posibleen zerrenda ezagutzeko.

- **stop_commands** (irakurketa): Zuriunez banandutako gelditze moduen zerrenda itzultzen du. Balio posibleak *coast*, *brake* eta *hold* dira. *coast* aukerak motorraren korrontea guztiz kenduko du motorretik. *brake* aginduak korrontea kendu ondoren motorrean karga elektriko pasibo bat ezarriko du. Karga honek biraketaren energia xurgatuko du eta *coast* aukerak baina azkarrago gelditzuko du motorra. *hold* aukerak ez du korronterik kenduko motorretik. Horren ordez motorra uneko posizioan mantentzen saiatuko da. Kanpoko indar batek motorra bultzatzen badu posizio horretara itzuli arte biratuko da motorra.

- **time_sp** (irakurketa/idazketa): Idazketak *run-timed* agindua erabiltzean motorrak zenbat milisegundoz biratu behar duen zehazten du. Irakurketak bere balioa itzultzen du. Unitateak milisegundoak dira.

8. KAPITULUA

B Eraskina: Erabiltzailearentzako eskuliburua

Atal honetan *driver*ra nola instalatu eta exekutatu azaltzen da.

8.1 Instalazioa

Driver honen instalazioak bi atal ditu, alde batetik robotean instalatu behar da eta bestetik erabiltzailearen ordenagailuan.

8.1.1 *Driver*raren instalazioa *EV3*an

Behin paketea eskuratuta, *EV3*aren fitxategiak bertara kopiatu behar dira. Hauek *ev3pkg_ev3* karpetan aurki daitezke. Demagun paketearen bi karpetak (*EV3*an instalatu beharrekoa eta *ROS*en instalatu beharrekoa) biltzen dituen karpetan aurkitzen garela, hurrengo lerroa nahikoa da fitxategiak *EV3*aren *SD* txartelean kopiatzeko:

```
scp -r ev3pkg_ev3/ <ev3_erabiltzaile_izena>@<ev3_ip_helbidea>:
```

Adibidez: `scp -r ev3pkg_ev3/ev3pkg aratz@10.42.0.3:`

Honek *EV3*aren erabiltzailearen *home* karpetan kopiatuko du *driver*raren edukia.

Behin fitxategiak *EV3*an edukita, *ev3pkg* karpetan aurki daitekeen *ev3_install* fitxategia exekutatuz driverra robota abiarazten den bakoitzean automatikoki exekutatzea lortzen da. Exekutagarri honek parametro bezela paketea aurkitzen den karpetaren helbide absolutua hartzen du:

Adib: `./ev3_install /aratz/ev3pkg`

Kontuan izan behar da instalazio amaieran robota berrabiarazi behar dela driverra exekuta dadin. Robota erabiltzen jarraitu nahi bada *reboot* komandoa exekutatu behar da. Bestela, besterik gabe itzal daiteke eta hurrengo aldiz piztean automatikoki exekutatuko da driverra. Posible da honetarako baimenak behar izatea. Ondorioz *sudo* komandoa lerroaren hasieran gehitu beharko da kasu batzuetan:

```
sudo ./ev3_install /aratz/ev3pkg
```

8.1.2 *Driver*raren instalazioa erabiltzailearen ordenagailuan

Driver hau *ROS nodo* bat da. Ondorioz, *ROS*en *catkin* ingurunetan beste edoizein pakete *source*tik instalatzen den bezela instalatzen da.

8.2 *Driver*raren exekuzioa

Behin *ev3pkg* paketea instalatuta driverra exekutatzeko honen *ROS* abiarazlea (*launcher*) exekutatzea nahikoa da. Honetarako hurrengo lerroa exekutatu behar da:

```
roslaunch ev3pkg ev3pkg.launch
```

Robotean ez da ezer exekutatu beharko driverra automatikoki exekutatzen bait da.

Bibliografia

[1] ev3dev-en webgune ofiziala. http://www.ev3dev.org/.

[2] Intorobotics robotikaren inguruko web orria. http://www.intorobotics.com/.

[3] Lego mindstormsen webgune ofiziala. http://www.lego.com/en-gb/mindstorms/.

[4] Lego mindstormsen webgune ofizialaren *education* atala. http://www.legoeducation.com/MINDSTORMS.

[5] Robot operating systemen webgune ofiziala. http://www.ros.org/.

[6] Robotsquare robotikaren inguruko web orria. http://robotsquare.com/.

[7] Wikibooks latex. https://en.wikibooks.org/wiki/LaTeX.

[8] Wikipediaren ev3aren atala. https://en.wikipedia.org/wiki/Lego_Mindstorms_EV3.