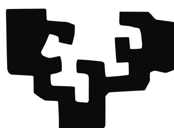
eman ta zabal zazu



Universidad del País Vasco | Euskal Herriko Unibertsitatea

Informatika Ingeniaritzako Gradua

Konputazioa

Gradu Amaierako Proiektua

Pertsonen garraiorako drone baten komunikazio softwarearen garapena

Egilea

Unai Cantero Acosta

informatika fakultatea

facultad de informática

2019

Laburpena

Dokumentu honen helburua Tecnalia enpresako “**fly free**” proiekturako egindako komunikazio software baten xehetasunak azaltzea da.

Fly free proiektuaren helburua pertsonen garraiorako drone edo aireontzi elektriko bat eraikitzea da. Normala den bezala, helburu hori lortzeko talde oso baten eta elementu asko integratzearen beharra dago. Dokumentu honetan drone horrek eramango duen komunikazio softwarearen garapenean zehar egindakoak zehaztuko dira. Software hori garatzeko hainbat hardware elementuren beharra izan da, era berean, hainbat teknika eta konfigurazio probatu dira komunikazio protokolo onenarekin eman den arte. Komunikazio protokolo horren abiadura optimizatzeko ere hainbat proba eta doikuntza egin behar izan dira.

Gaien aurkibidea

Laburpena i

Gaien aurkibidea iii

Irudien aurkibidea vii

Taulen aurkibidea ix

1 Sarrera 1

1.1 Helburuak 1

1.2 Fly free proiektua eta artearen egoera 2

1.3 Motibazioa 5

2 Proiektuaren Helburuen Dokumentua 7

2.1 Irismena 7

2.2 Plangintza 7

2.2.1 Atazen identifikazioa 7

2.2.2 Planifikatutako egutegia (Gantt diagrama) 9

2.2.3 Lan metodologia 10

2.3 Emangarriak 11

2.4 Arriskuen kudeaketa 11

2.5 Baliabideen kudeaketa 12

3 Oinarri teorikoak 13

3.1 Serieko komunikazioak 13

3.1.1 Sinkronizazio bit-ak komunikazio asinkronoan 14

3.2 UART portuak 15

3.3 Baud 16

3.4 Nabigazio angeluak 16

3.5 Thrust edo bultzada 18

4 Erabilitako tresnak 21

4.1 Hardware 21

4.1.1 Raspberry Pi 21

4.1.2 Pixhawk 23

4.1.3 USB-UART bihurgailua 25

4.2 Software 26

4.2.1 QGroundControl 26

4.2.2 MAVLink 28

4.2.3 PX4 autopilotoa 28

5 Garapena 41

5.1 Aurrekariak 41

5.1.1 Arkitektura zirkularra 42

5.1.2 Arkitektura banatua: BeagleBone 44

5.1.3 Arkitektura banatua: Raspberry Pi B+ 47

5.2 Pixhawk inplementazioa (esklaboak) 48

5.2.1 Fitxategiak eta aurrekariak 49

5.2.2 Serieko portuak 50

5.2.3 Mezuen jasotzea 53

5.2.4 Mezuen bidaltzea 55
5.2.5 Mezuen tratamendua 55
5.2.6 Sentsore eta eragingailuen kudeaketa 58
5.2.7 Inplementazio orokorra 60
5.3 Raspberry inplementazioa (bideratzailea) 68
5.3.1 Aurrekariak 68
5.3.2 Serieko portuak 73
5.3.3 Mezuen jasotzea 74
5.3.4 Mezuen bidaltzea 74
5.3.5 Inplementazio orokorra 75
5.4 Komunikazio probak 82
5.4.1 Monitorizazioa 82
5.4.2 Abiadura probak 83
5.4.3 Sendotasun probak 83

6 Ondorioak **85**
6.1 Priektuaren ondorioak 85
6.2 Etorkizuneko lana 85
6.3 Ondorio pertsonalak 86

Bibliografia **87**

A Funtzioen inplementazioak **89**
A.1 _setup_port() 89
A.2 _open_port() 90
A.3 open_serial() 91
A.4 start() 91

A.5 close_serial() 92
A.6 read_port_frame() 92
A.7 send_message_buf() 93
A.8 CalculateChecksum() 94
A.9 IsValid() 94
A.10 bytesToAngles() 95
A.11 convertProcessValue() 95
A.12 split4bytes() 96
A.13 Koaternoien funtzioak 96
A.14 resynchronize() 98
A.15 open_port() eta open_port_master() 98
A.16 config_port() eta config_port_master() 99
A.17 setFrameToMaster() 99
A.18 setFramesToSlaves() 102
A.19 Run() (Esklaboen programa nagusia) 103
A.20 main() (Bideratzailearen programa nagusia) 106

B Fitxategien edukiak **111**

B.1 codisava.h 111
B.2 codisava.cpp 111
B.3 serial_port.cpp 112
B.4 slave_controller.h 114

Irudien aurkibidea

1.1 Dubaien erabilitako **Volocopter** modeloa. 3
1.2 **Volocopter**-en dineinu ofiziala. 3
1.3 **Airbus** eta **Audiren** aireko taxia. 4
1.4 **TEA**-ren dineinuaren prototipoa eskala erdian. 4
1.5 **TEA**-ren dineinuaren prototipoa eskala erdian, beste angelu batetik ikusita. 5

2.1 LDE diagrama. 8
2.2 Gantt diagrama. 9

3.1 Komunikazio asinkronoaren adibide bat. 14
3.2 Bi **UART** porturen arteko konexioa. 16
3.3 Nabigazio angeluak hegazkin batean irudikatuta. 17
3.4 Nabigazio angeluak **ZYX** ardatz sisteman hegazkin batekin. 18

4.1 **Raspberry Pi 3 B+** modeloa. 22
4.2 **Rasberry Pi 3 B+** modeloaren pin-en eskema. 22
4.3 **Pixhawk**-en web orri ofizialean eskaintzen diren modelo batzuk. 23
4.4 **Pixwak2** hegaldi kontroladorea. 24
4.5 **Pixwak4** hegaldi kontroladorea. 25
4.6 **USB-UART** bihurgailua. 25

4.7 **USB-nanoUSB** kablea. 26
4.8 **QGroundControl**-en logoa. 27
4.9 **Pixhawk4** baten kalibrazioa **QGroundControl** erabiliz. 27
4.10 **MAVLink**-en logoa 28
4.11 **PX4**-ren logoa. 29
4.12 **PX4** kodearen konpilazioa. 30
4.13 **PX4** kodearen konpilazioa huts egitean. 31
4.14 **PX4** kontsola. 31
4.15 **PX4**-ren lehentsitako aplikazio zerrendaren zati bat. 32
4.16 Sortutako aplikazioa aplikazio zerrendan. 35
4.17 Pixhawk-en microSD txartelen edukia. 35

5.1 Arkitektura zirkularraren eskema grafikoa. 42
5.2 Arkitektura zirkularrean erabili den mezuaren egitura. 43
5.3 Beaglebone black. 44
5.4 Arkitektura banatua, **BeagleBone** bertsioa. 44
5.5 **BeagleBone**-aren pin-en eskema. 45
5.6 **BeagleBone**-aren **UART**-en kokapena. 45
5.7 Pixhawk esklaboei bideratzailetik iristen zaien mezua. 46
5.8 Arkitektura banatua, **RaspBerry** bertsioa. 47
5.9 Masterrak bideratzaileari bidaltzen dion mezua ID byte-ak kenduta. . . . 48
5.10 **microSD** memoria txartel bat. 69
5.11 **balenaEtcher** softwarearen logoa. 69
5.12 balenaEtcher programa. 69
5.13 balenaEtcher programa aukeraketak eginda. 70
5.14 balenaEtcher programa instalazioa egiten ari den bitartean. 70
5.15 Raspbian mahaigaina 71

Taulen aurkibidea

2.1 Ataza bakoitzari esleitutako ordu kopurua 10

1. KAPITULUA

Sarrera

Dokumentu hau 2019. urtean Unai Cantero Acostak UPV/EHU unibertsitateko Informatika Fakultatean egindako Gradu Amaierako Proiektuari dagokio. Proiektuko zuzendariak Basilio Sierra, Informatika Fakultateko irakaslea, eta Joseba Lasa **Tecnalia** enpresako **TEA** (*Tecnalia electric aircraft*) taldeko burua dira.

Proiektu hau askoz handiagoa den beste proiektu baten barruan dagoenez (**fly free**), honetaz mintzatzea beharrezkoa da proiektuaren motibazioa, helburuak eta artearen egoera ulertu ahal izateko.

Kapitulu honetan **fly free** proiektuaren gorabeherak eta garatu den komunikazio softwareak proiektu horretan duen zentzua azalduko dira.

1.1 Helburuak

Proiektuaren helburu nagusia **Tecnalia** enpresako **TEA** (*Tecnalia electric aircraft*) taldeak **fly free** izeneko proiektuan garatzen ari diren drone edo aireontzian erabiltzeko **komunikazio software** bat garatzea da. Horretarako, aireontzi horretan erabiltzen den hardware nahiz softwarea analizatu, landu eta ulertzea beharrezkoa izan da. Aipatutako hardwarea **Pixhawk** deitutako hegaldi kontroladoreak dira, zeinak **PX4** autopiloto soft-

warearekin egiten duten hegan.

Aurrekoaz gain, komunikazioak burutu ahal izateko, hardware artean komunikatzeko beharrezkoak diren tresnak eta teknikak ulertzea ezinbestekoa da, hala nola **UART serieko portuak** zer diren eta informazioa transmititzeko protokoloak nola funtzionatzen duten serieko portuen artean transmisio seguru bat egiteko.

Bukatzeko, ezinbestekoa ez izan arren, aeronautikako kontzeptu oso oinarrizko batzuk ere landu behar izan dira, **nabigazio angeluak** esate baterako (**roll**, **pitch** eta **yaw**). Hauen xehetasunak 3. kapituluan azalduko dira sakonago.

1.2 Fly free proiektua eta artearen egoera

Fly free proiektuaren helburua pertsonen garraiorako drone bat garatzea da. Proiektua **Tecnalia** enpresako **TEA** taldeak martxan jarri arren eta aireontziaren diseinua eta xehetasunak beraiek zehaztu arren, beste talde batzuen laguntza ere izan dute ontziaren fabrikazioa eta bestelako eginkizunak burutzeko.

Mundu osoan zehar existitzen dira jada pertsonen garraiorako droneak edo "**aireko taxiak**" egin nahi dituzten proiektu asko. Mota honetako proiektu famatuena eta oihartzun handiena sortzen ari dena Dubai hirian jaurtitzeko ia prest dagoen **AAT** (*autonomous aerial taxi*) zerbitzua da seguru aski. **Volocopter** enpresa alemaniarrak sortutako aireontzia erabiliko da proiektu hau martxan jartzeko[1]. 1.1 eta 1.2 irudietan ikusi daiteke enpresa honen aireaontziaren diseinua.

[1]https://mashable.com/2017/06/19/dubai-autonomous-taxi-service-scheduled/?europe=true#CD5AYdey75qG

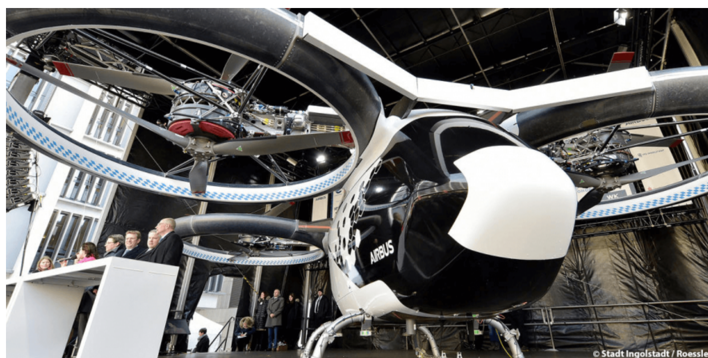
1.1 Irudia: Dubaien erabilitako **Volocopter** modeloa.

1.2 Irudia: **Volocopter**-en dineinu ofiziala.

Volocopter enpresaren esanetan beraien helburua munduko lehenengo **VTOL** (bertikalki aireratu eta lurreratzeko gaitasuna duen aireontzia) elektriko eta segurua eraikitzea da, modu honetan hiri modernoetan geroz eta handiagoa den mugikortasun arazoa murrizten lagunduz.

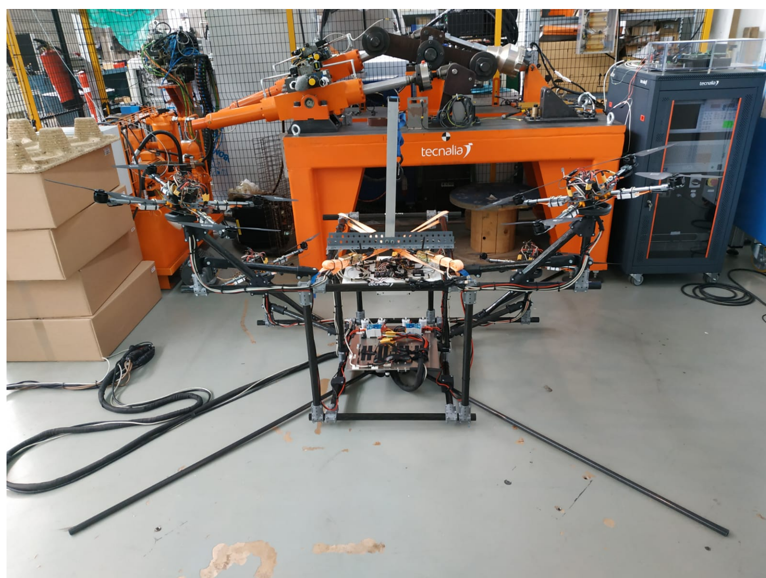
Mota berdineko proiektu garrantzitsu bat martxan duen beste enpresa bat **Airbus** enpresa frantsesa litzateke, zeinak bere aireko taxi bat diseinatu duen **Audiren** laguntzarekin. Bi enpresek martxan jarri duten proiektuari **Urban Air Mobilty** jarri diote izena eta 2019ko martxoaren 12an aurkeztu zuten beraien lehen prototipoa (1.3 irudia) [2].

[2]https://www.electrive.com/2019/03/12/airbus-electric-air-taxi-cityairbus-revealed-before-maiden-flight/



1.3 Irudia: Airbus eta **Audiren** aireko taxia.

TEA-koek diseinatutako aireontziak hainbat berezitasun ditu, aurretik azaldutako proiektuetatik desberdintzen dutenak. Aireontzi honekin efizientzia eta abiadura arteko oreka lortu nahi da, hau gauzatzeko garatu den diseinua lau drone independentez osatuta dago, zeinak helize moduan funtzionatzen duten. Honek eskaintzen duen abantaila nagusia “helize” hauek kabina bati fisikoki akoplatzean lortzen da, modu honetan bakoitza independenteki mugitu daiteke mugimendu askatasun handiagoa lortuz. Gainera helizeak mugitzen direnez, eserlekua ez da mugitzen, bidaiarien konforta hobetuz. 1.4 eta 1.5 irudietan ikusi daiteke TEA taldearen diseinuaren prototipo bat, eskala erdian.



1.4 Irudia: TEA-ren dineinuaren prototipoa eskala erdian.

1.5 Irudia: TEA-ren dineinuaren prototipoa eskala erdian, beste angelu batetik ikusita.

1.3 Motibazioa

Komunikazio software baten beharra dronearen helizeak drone independenteak direlako sortzen da. Helize bakoitzak bere hegaldi kontrolagailua du eta independenteki mugitzen da, besteekin sinkronizatuta, erdian dagoen kontrolagailu nagusi bati esker.

Azaldutako arkitektura honek nahitaez behar du komunikazio metodo bat kontrolagailu zentralak helizeei nahi diren aginduak bidali ahal izateko. **PX4** autopiloto softwareak **C**, **C++** eta **matlab** lengoaietan modifikazoak egitea ahalbidetzen duenez erabili diren **Pixhawk** kontrolagailuak lengaoaia hauetan programatu dira komunikazioak ahalbitzeko.

Kontrolagailu zentralak **matlab**-en programatutako kontrol software konplexu bat eramango du. Kontrol softwareak definitzen du beste helizeek egin behar dituzten mugimenduak drone osoa nahi den bezala kokatzeko. Agindu horiek transmititzeko eta helizeen posizioak jakiteko behar izan da hain zuzen proiektu honetan garatu den komunikazio softwarea.

2. KAPITULUA

Proiektuaren Helburuen Dokumentua

Kapitulu honetan, proiektuaren kudeaketarekin erlazionatutako atalak aztertuko dira, hala nola, proiektuaren irismena, plangintza, lan metodologia, emangarriak, arriskuen kudeaketa, baliabideen kudeaketa eta interesatuak.

2.1 Irismena

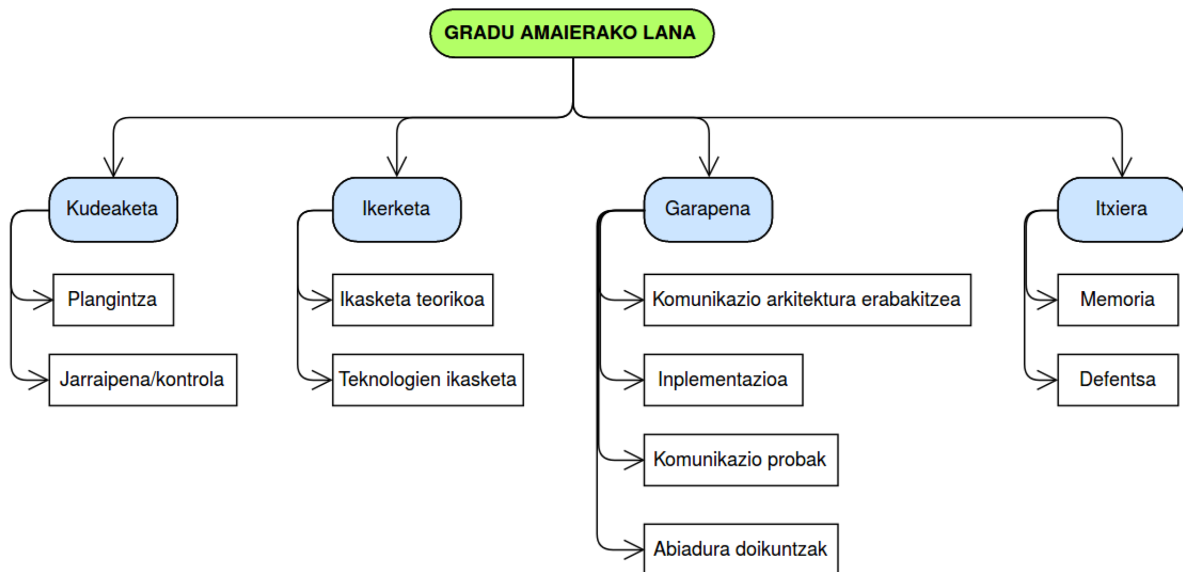
Proiektuaren irismena komunikazio software bat garatzea da Tecnaliaren **fly free** proiekturako. Zehazki garatzen ari den dronean erabiltzeko prest dagoen software bat garatzea da helburua.

2.2 Plangintza

Atal honetan, proiektuaren plangintza aztertuko da. Alde batetik, garatu beharreko ataza desberdinak azalduko dira eta ondoren, ataza bakoitzari eskaini beharreko denboraren aurreikuspena egingo da.

2.2.1 Atazen identifikazioa

Proiektua lau atal desberdinetan banatzea erabaki da. 2.1 irudian ataza hauek biltzen dituen **Lanaren Deskonposaketa Egitura**, **LDE** diagrama hain zuzen, azaltzen da.



2.1 Irudia: LDE diagrama.

Jarraian, ataza hauek azalduko dira banan banan:

- **Kudeaketa**: ataza honen barruan, **plangintza** eta **jarraipena eta kontrola** sartzen dira. **Plangintza** proiektuaren nondik norakoak erabakitzeko balio du, bete beharreko mugak zehaztuko dira garatu beharreko ataza desberdinetarako. **Jarraipen eta kontrola**ren bidez, aldiz, zehaztutako helburuak betetzen diren kontrolatuko da eta horrez gain, adibidez, egon daitezkeen arazoen jarraipena eramango da.

- **Ikerketa**: atal honetan proiektuaren garapenerako beharrezkoak diren oinarri teorikoak landuko dira. Bi ataza desberdinetan banatu da: **ikasketa teorikoa** eta **teknologien ikasketa**. Lehenengoan komunikazio informatiko eta hegazkingintzarekin lotutako hainbat kontzeptu teoriko hutsak landuko dira. Bigarrengoan proiektua burutzeko beharrezkoak diren hardware eta softwareari buruzko ezagutzak lortuko dira.

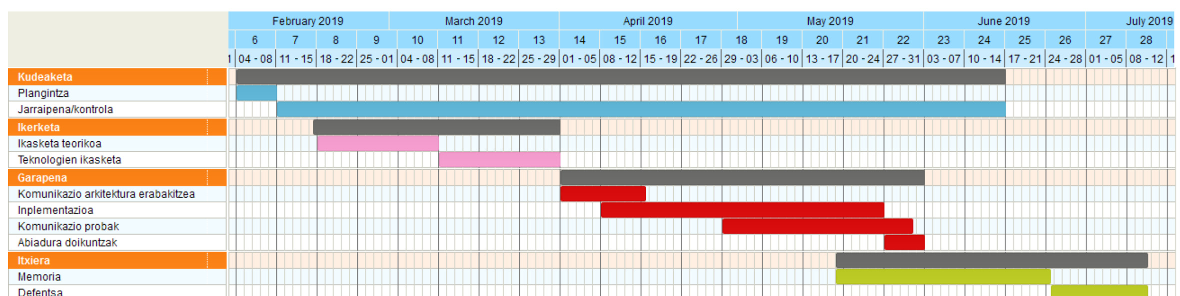
- **Garapena**: izenak adierazten duen bezala, ataza hau proiektuaren garapenari dagokio. Honetan lau atal desberdindu dira. Lehenengo erabiliko den **komunikazio arkitektura erabaki** behar da, hau da, komunikatuko diren tresna guztiak konektatzeko modua. Arkitektura erabakita, arkitektura honekiko softwarearen **inplementazioa** garatu behar da. Behin softwarea garatu dela, **komunikazio probak** egin

behar dira, hauei esker kodean dauden akatsak zuzenduz eta doikuntzak eginez. Azkenik **komunikazioen abiadura doitu** behar da, abiadura eraginkor bat lortu arte.

- **Itxiera**: azkeneko atal hau, proeiktuaren entrega eta aurkezpenerako bideratuta dago. Beraz, honen barruan idatzi beharreko **memoria** eta egin beharreko **defentsa**ren prestakuntza sartzen dira.

2.2.2 Planifikatutako egutegia (Gantt diagrama)

Proiektuak hartuko duen denboraren ideia egokiagoa izateko, 2.2 irudiko **Gantt diagrama** garatu da. Bertan, ataza bakoitzak egutegian hartuko duen denbora agertzen da. Ikusi daitekeenez, kudeaketa proiektu osoan zehar landuko da eta gainontzekoek, ordena kronologiko bat jarraituko dute. Lehenengo landuko dena ikerketa izango da, ondoren garapena eta azkenik, itxiera.



2.2 Irudia: Gantt diagrama.

Horrez gain, proiekturako 300 ordu daudela kontuan izanik, ordu hauek adierazitako atazei esleitu zaie 2.1 taulan agertzen den modura. Betiere esleipen hau proeiktua aurrera doan heinean aldatu daitekeela kontuan izanik, gertatu daitezkeen desbideraketak direla medio.

ATAZA	*ORDU KOPURUA*
Kudeaketa	**40 ordu**
Plangintza	20 ordu
Jarraipena/kontrola	20 ordu
Ikerketa	**60 ordu**
Ikasketa teorikoa	30 ordu
Teknologien ikasketa	30 ordu
Garapena	**125 ordu**
Komunikazio arkitektura erabakitzea	25 ordu
Inplementazioa	60 ordu
Komunikazio probak	25 ordu
Abiadura doikuntzak	15 ordu
Itxiera	**75 ordu**
Memoria	60 ordu
Defentsa	15 ordu
GUZTIRA	*300 ordu*

2.1 Taula: Ataza bakoitzari esleitutako ordu kopurua

2.2.3 Lan metodologia

Proiektuaren arrakasta ziurtatzeko lan metodologia bat zehaztu da. Proiektua bakarrik garatuko denez, hau da, ez denez talde baten menpe egongo, nahiko erraza izango da honen jarraipena eramatea. Hala ere, zuzendariekin bilerak maiz egingo dira, garatutakoari buruz eta jarraian egin beharreko atazei buruz hitz egiteko.

Horrez gain, egunero **Tecnalia** enpresan egingo da lana, enpresa horri dagokion lan bat izanda bertan hornitzen delako beharrezko material guztia. Gainera, modu honetan lanaren jarraipena eraman ahalko dute enpresako kideek.

Egun bakoitzean zer egiten den jakiteko, eguneroko bat eramango da Google Drive plataforman. **Overleaf**[1] -en idatziko da, LaTeX online zerbitzua eskaintzen duena. Tresna honek edozein ordenagailutik idaztea eta aldaketak modu argian ikustea ahalbidetuko du.

[1] https://www.overleaf.com/

2.3 Emangarriak

Emangarriei dagokienez, proiektuan hainbat sortuko dira eta 2.2.2 ataleko Gantt diagramaren arabera bukaera data desberdinak izango dituzte. Hauek dira sortuko diren emangarriak:

- **Kodea**: garatuko diren programa desberdinak biltzen dituen kode bilduma. Bukaera data maiatzaren 15ean ezarri da.
- **Memoria**: proiektuaren nondik norakoak, helburuak, garapena, emaitza eta ondorioak biltzen dituen txosten idatzia. Memoriaren entrega ekainak 23rako ezarrita dagoenez, ekainak 17rako bukatuta egotea espero da, denbora-tarte hori gertatu daitekeen edozein ezustetarako utziz.
- **Aurkezpena**: defentsaren egunean erabiliko den aurkezpena. Defentsa uztailak 01-12 egunetan izango denez, aurkezpena uztailaren 1erako bukatu nahi da.

2.4 Arriskuen kudeaketa

Proeiktuan zehar gertatu daitezkeen arriskuak aztertzea ezinbestekoa da hauek saihestu ahal izateko eta gertatzekotan nola jokatuko den erabakitzeko. Jarraian, gertatu daitezkeen arazoak eta hauen aurrean hartuko diren neurriak azalduko dira:

- **Informazio galera**: sistema informatikoan gertatu daitezkeen arazoak direla eta, informazio galerak egon daitezke. Hauek ekiditeko, garatutako kodea mahaigaineko ordenagailuan gordetzeaz gain, **GitHub** plataformara igotzea erabaki da. Gainera, memoria garatzerako orduan sortuko diren dokumentu guztiak **Overleaf**-en gordetzeaz gain, **Google Drive** zein ordenagailuan lokalean gordeko dira.
- **Arazo teknikoak**: erabiliko diren software eta hardware tresnetan gertatu daitezkeen arazoak dira. Hauek ekiditeko, tresna guztiak modu egokian erabili eta gordeko dira. Arazoren bat gertatuz gero, konponbidea bilatzen saiatuko da eta bestela, enpresako taldeko kideei laguntza eskatuko zaie.
- **Komunikazio falta**: proiektua modu egokian garatzeko, zuzendariekin komunikazioa ezinbestekoa da. Hortaz, bilerak maiz hitzartuko dira aurrerapen eta arazo guztiei buruz hitz egiteko.

- **Plangintzarekin bateraezintasuna**: garatutako plangintza jarraitzea ezinezkoa denean gertatzen diren atzerapen arazoak dira hauek. Plangintzan ataza bakoitzari zehaztutako denbora eskasa bada, ataza garrantzitsuenei lehentasuna emango zaie. Hala ere, desbiderapenak saihesteko, planifikatutakoa betetzen saiatuko da.

- **Osasun arazoak**: egon daitezkeen gaixotasunek sortutako arazoak dira. Ekidin ezin diren arazoak direnez, planifikatutakoa modu egokian bete behar da, gaixotasunen bat egonez gero, ahalik eta denbora gutxien galtzeko.

2.5 Baliabideen kudeaketa

Proiektuaren garapenean hainbat baliabide desberdin erabiliko dira. Memoria idazterako orduan LaTeX erabiliko da, **Overleaf** tresnarekin batera. Gainera, egindakoaren egunerokoa idazteko eta memoriaren bertsioak zein proiektuaren emaitzen taulak gordetzeko **Google Drive** erabiliko da. Aldiz, kodea **GitHub** plataforman gordeko da.

Beharrezko material guztia **Tecnalia** enpresak hornitzen du, lana egiteko ordenagailua barne.

3. KAPITULUA

Oinarri teorikoak

Proiektuan egin diren gauza batzuk ulertzeko oinarri teoriko batzuk behar izan dira, kapitulu honetan proiektuan egin dena ulergarriagoa izateko balioko duten kontzeptu teoriko batzuk azalduko dira.

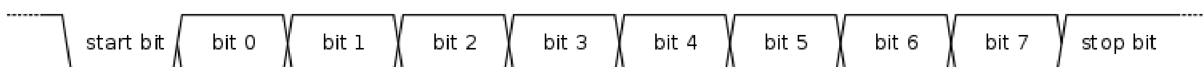
3.1 Serieko komunikazioak

Serieko komunikazioa edo komunikazio sekuentziala datuak bit-ak banaka bidalita komunikatzeko prozesuari deitzen zaio, komunikazio kanal edo bus baten bidez, komunikazio paraleloan ez bezala, non bit guztiak aldi berean bidaltzen diren [1], [2].

Serieko komunikazioaren abantaila nagusia komunikazio linea bakarraren beharra da, komunikazio paraleloan ez bezala, non bidali nahi den informazioa osatzen duten bit kopurua bezainbeste linea behar diren. Tamalez, honek bestelako arazo batzuk ere ekar ditzazke bit asko transmititu behar direnean, hala nola interferentzia edo desinkronizazioak. Horrez gain transmisio abiadura handietan errendimendu altuagoa du komunikazio paraleloak. Ahulezia hauek software bidez konpentsatu daitezke.

Serieko komunikazioetan komunikazio **sinkrono** eta **asinkronoa** desberdindu daitezke, proiektu honetan serieko komunikazio asinkronoa erabili da. Serieko komunikazio mota hau datuak sinkronizatzeko moduan desberdintzen da sinkronoarengandik. Mota sinkro-

noan komunikazioak “**clock**” seinale komun baten bidez sinkronizatzen dira, asinkronoan ordea “**start**” eta “**stop**” seinaleen bidez kontrolatzen da sinkronizazioa. Modu honetan **start** seinaleak hartzailea datuak jasotzeko prestatzen du eta **stop** seinaleak bere egoera berrabiarazten du beste sekuentzia bat hasi ahal izateko. 3.1 irudian ikusi daiteke komunikazio mota honen adibide grafiko bat.



3.1 Irudia: Komunikazio asinkronoaren adibide bat.

Komunikazio asinkronoa ahalbidetzen duten makina gailu daude, zeinak aukera ezberdin asko eskaintzen dituzte, kasu honetan **UART** (*universal asynchronous receiver-transmitter*) portuak erabili dira komunikazioak gauzatzeko.

3.1.1 Sinkronizazio bit-ak komunikazio asinkronoan

Aurretik azaldu diren **start** eta **stop** bit hauek mezuaren osotasuna ziurtatzeko balio dute, adibidez hartzaileak start bit bat jasotzen badu eta end bita jaso baino lehen, beste start bit bat jasotzen badu, jasotzaileak badaki mezuaren zati bat galdu dela, ez baitu mezuaren bukaera adierazten duen bita jaso.

Desinkronizazio bat detektatzeko ere balio dute, adibidez hartzaileak irakurketa bakoitzean irakurtzen duen lehen bita ez bada start, badaki jaso beharreko mezua erditik zatituta jasotzen ari dela eta hortaz, komunikazioak resinkronizatzen saiatzea komeni dela.

Batzuetan, komunikazioen sinkronizazioa oraindik gehiago ziurtatzeko hirugarren sinkronizazio bit bat erabiltzen da, **checksum** izenekoa [3]. Checksum bita oro har stop bitaren aurretik jartzen da eta bere helburua datu sekuentzia batean aldaketak detektatzea da, igorleak bidalitako mezuaren eta hartzaileak jasotakoaren arteko desberdintasunik ez dagoela egiaztatuz. Desberdintasunak daudela detektatzen bada, start eta stop bitak jaso diren arren, jasotako datuen artean zerbait galdu dela detektatu dezake jasotzaileak eta

ondorioz desnkronizazio bat edo erroreren bat egon dela ohartu.

Checksum bat kalkulatzeko modu ohikoena mezu baten datu guztien arteko **XOR** eragiketa bat egitean datza, modu honetan igorleak eragiketa horren emaitza ezartzen du checksum bitean. Gero hartzaileak mezua jasotzean, jasotako datuen arteko XOR eragiketa bat egingo du, bere checksuma kalkulatuz eta azkenik jasotako checksumaren eta kalkulatu berriaren arteko konparazio bat egingo du. Berdinak badira, mezua zuzen bidali dela esan nahi du eta bestela nolabaiteko errore bat jazo dela.

3.2 UART portuak

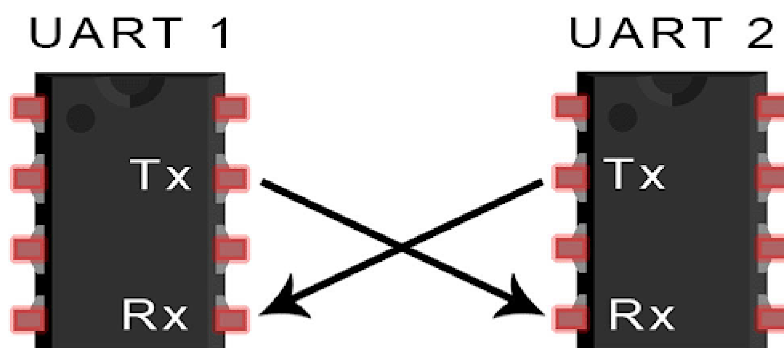
UART (*asynchronous receiver-transmitter*) bat komunikazio serial asinkronorako erabiltzen den ordenagailu hardware bat da, zeinean datuen formatua eta eta transmisio abiadura konfiguragarriak diren[4].

UART batek **transmisore** eta **hargailu** bat erabiltzen du datu transmisioak burutzeko. UART hardwarearen eragiketa guztiak "**clock**" seinale baten bidez kontrolatzen dira, hargailuak clock pultsu bakoitzean egiaztatzen du jasotako datuen egoera, **start** bita bilatzen. Start bita aurkitzen badu linea horren egoera lagintzen du truke erregistroan (datu trukaketa burutzeko erabiltzen den erregistroa) eta denbora labur bat itxaron ostean erregistroko datuak eskuragarri egiten dira. Datuen hartzea **stop** bita aurkitu arte irauten du, aurkitu ostean, prozesua berrabiarazten da.

Transmisoreak bestalde, ez du behar clock seinalerik eta ez dago inongo egoeraren mende, honek transmisio eragiketa hartze eragiketa baino askoz ere sinpleagoan bihurtzen du. Bidaltze sistemak truke erregistroan karaktere bat sartu bezain laster UART-ak start bit bat sortzen du, behar diren bit kopurua mugitzen ditu lineara eta azkenik stop bita bidaltzen du. Eragiketa batzuek hartze eta transmititzea aldi berean behar dutenez UART-ak bi truke erregistro erabiltzen dituzte, bata transmitutako karaktereentzat eta bestea jasotakoentzat.

Bi UART porturen arteko transmisio bat burutzeko beraz, UART baten **transmisorea** bestearen **hargailura** konektatu behar da. Oro har, UART portuen transmisoreari eta hargailuari **Tx** eta **Rx** deitzen zaie hurrenez hurren. 3.2 irudian ikusi daiteke bi UART porturen

arteko konexioaren adibide grafiko bat.



3.2 Irudia: Bi **UART** porturen arteko konexioa.

3.3 Baud

Baud bat telekomunikazioetan erabiltzen den neurri-unitate bat da. Unitate honek komunikazio bide digital batean segundoko bidaltzen diren ”sinbolo” kopurua adierazten du. Sinbolo bakoitzak bit bat baino gehiago izan ditzake, modulazio eskemaren arabera [5].

Adierazi behar da **baud abiadura** (*baud rate*) ez dela **bit tasarekin** (*bit rate*) nahastu behar, esan bezala sinbolo bakoitzak bit bat baino gehiago garraiatu ditzakelako. Soilik sinbolo bakoitzak bit bat garraiatzen duenean datotz bat bit tasa eta baud abiadura.

Baud abiadura adibidez **UART** portuen arteko komunikazioetan erabiltzen da abiadura neurri bezala.

3.4 Nabigazio angeluak

Nabigazio angeluak Eulerren angelu mota bat dira. Hauek objektu batek duen orientazioa deskribatzeko balio dute hiru dimentsioko espazio batean, hau da, koordenatu sistema mugikor bat (objektuarena) koordenatu sistema finko bati dagokiolarik dagoenean erabili daitezke sistema mugikor horren posizioa emateko [6].

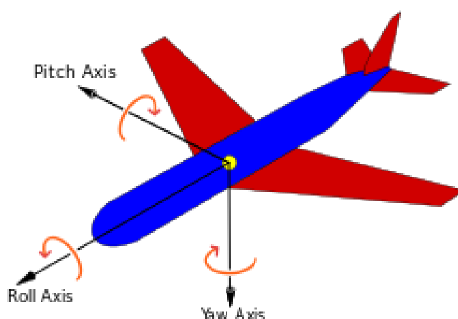
Hiru angelu daude, **yaw**, **pitch** eta **roll**. **Yaw** angeluak objektuaren **norabidea** deskribatzen du, **pitch** angeluak **goratzea** eta **roll** angeluak **kopadura**.

Hiru angelu hauek ondoz ondoko hiru maniobraren baliokide dira. Hiru ardatzeko sistema bat emanda, non ardatzak aurretik deskribatutako angeluen izen berdina duten (**yaw** ardatza edo **Z**, **pitch** ardatza edo **Y** eta **roll** ardatza edo **X**) eta hegazkin edo aireontzi bat hartuta, hiru errotazio nagusi existitzen dira, normalean egiten diren ardatzaren izen berdina ematen zaie, eta aireontziak duen ardatz sistema atzitzea ahalbidetzen dute. Ordena horretan emanda etorri eta egin behar dira (**yaw**, **pitch** eta **roll**), emaitza finala hauek egiten diren ordenaren araberakoa da eta:

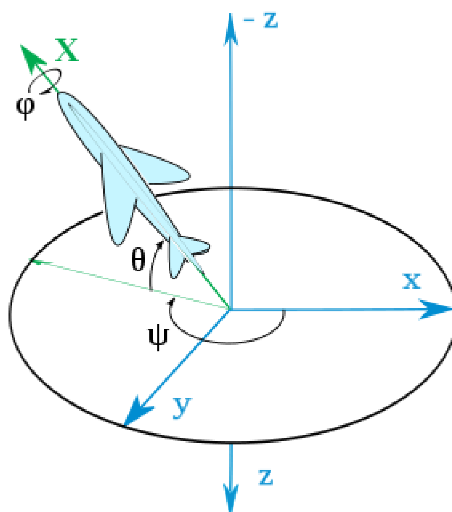
- **Yaw**: Hegazkinari perpendikularra den ardatz bertikalaren gaineko errotazioa.
- **Pitch**: Hegazkinaren muturraren makurdura edo hegazkinaren hegoek sortzen duten ardatzaren gaineko errotazioa.
- **Roll**: Hegazkinaren mutur eta isatsak sortzen duten ardatzaren gaineko errotazioa.

Hiru errotazio hauek intrintsekoak dira, hau da, sistema mugikorrari erlatiboak dira, hau oso erabilgarria da adibidez hegazkin bateko piloto batek maniobra bat deskribatu nahi duenean. 3.3 irudian ikusi daitezke nabigazio angeluak hegazkin batean irudikatuta eta 3.4 irudian hegazkina **XYZ** ardatzean kokatuta.

Nabigazio angeluak, **yaw**, **pitch** eta **roll**, oro har ψ, θ eta ϕ letrekin adierazten dira hurrenez hurren.



3.3 Irudia: Nabigazio angeluak hegazkin batean irudikatuta.



3.4 Irudia: Nabigazio angeluak **ZYX** ardatz sisteman hegazkin batekin.

Hegazkinaz gain, nabigazio angeluak dronetan ere aplikatu daitezke, proiektu honetarako; dronen gainean erabili behar izan dira.

3.5 Thrust edo bultzada

Nabigazio angeluez gain, drone baten posizioa jakiteko erabiltzen den beste parametro garrantzitsu bart **thrust** edo **bultzada** da [7].

Thrust parametroa teknikoki newtonen hirugarren legeak deskribatzen duen erreakzio indar bat da. Sistema batek masa bat azeleratzen duenean norabide jakin batean (**akzioa**) azeleratutako masa horrek indar berdina eragingo du baina kontrako norabide batean (**erreakzioa**).

Adibidez, hegazkin batek, aurreranzko thrust bat sortzen du, abiadura handiarekin biratzen duen helize batek airea bultzatzen duenean edo erreaktoreko gasak jaurtitzen dituenean hegazkinaren atzealdera. Aurreranzko norabidean sortzen den thrust-a bultzatzen den arearen masari proportzionala da, aire fluxuaren batez besteko abiadurarekin biderkatuta.

Dronetan thrust-a dronearen helizeak airea beheranzko norabidean azeleratzean sortzen da, aire honek goranzko bultzada bat sortuz. Drone bat sortzen ari den thrust-a zenbatekoa den jakinda, honen altuera zehaztu daiteke, nabigazio angeluekin batera bere posizioa determinatzea ahalbidetuz. Gainera drone baten helize bakoitzak sortzen duten bultzadak aldatuz lortzen da dronea hiru dimentsioko espazio batean mugiaraztea.

4. KAPITULUA

Erabilitako tresnak

Proiektua burutu ahal izateko hainbat tresnaren beharra izan dira, bai software eta bai hardware aldetik. Kapitulu honetan erabili diren tresna guztiak errepasatuko dira.

4.1 Hardware

Dronean erabiltzen diren hegaldi kontroladoreak komunikatu ahal izateko makina hardware osagai behar izan dira.

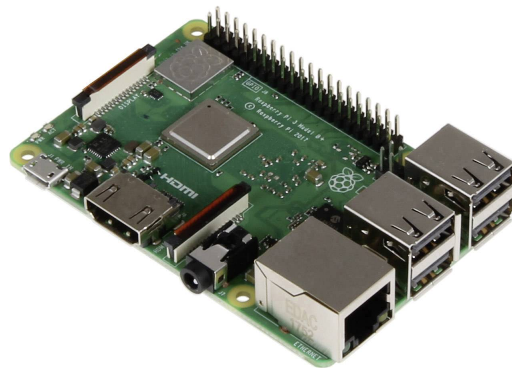
Guztira, **Raspberri Pi** bat, lau **Pixhawk4**, **Pixhawk2** bat, eta lau **USB-UART** bihurtzaile behar izan dira, dena konektatzeko kableekin batera, jarraian azalduko dira guztien xehetasunak.

4.1.1 Raspberry Pi

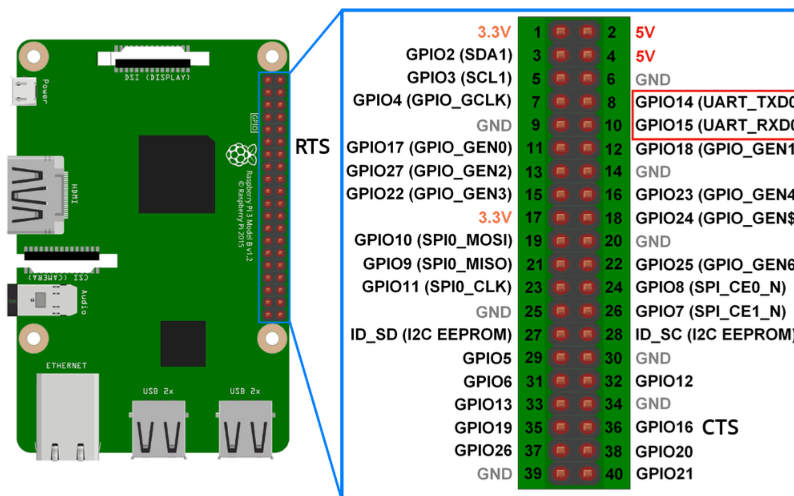
Raspberry Pi bat erabili da komunikaziak bideratzeko. Hardware honen funtzio nagusia kontroladore zentraletik iristen diren aginduak drone periferikoetara helaraztea da, aldi berean drone periferikoen erantzun bat kontroladore zentralera bidaltzen duen bitartean. Modu honetan kontroladore zentralak momentu oro daki nola dauden kokatututa helize guztiak.

Erabili den Raspberry Pi modeloa **Raspberry Pi B+** izan da (4.1 irudia). Honen arrazoia dituen **USB** portuak dira, izan ere, drone periferikoak eta kontroladore zentrala dena batera konektatu ahal izateko bost **UART** portu behar dira guztira. Raspberry Pi 3 B+ modeloak lau USB portu (UART bezala funtzionatu dezaketenak) eta UART portu bat du, honek bete nahi den eginkizuna lortzeko hardware ezin hobe batean bihurtzen du Raspberry-a.

4.2 irudian ikusten da hardware honen **pin**-en eskema, bertan garbi geratzen da non dauden erabili diren **UART** eta **USB** portuak.



4.1 Irudia: Raspberry Pi 3 B+ modeloa.



4.2 Irudia: Rasberry Pi 3 B+ modeloaren pin-en eskema.

4.1.2 Pixhawk

Pixhawk dronean erabili diren hegaldi kontroladoreen izena da. Pixhawk "open-hardware" proiektu independente bat da, zeinaren helburu nagusia koste baxuko eta kalitate handiko autopiloto hardwareak garatzea den, komunitate akademiko, garatzaile profesionalentzako edota hobby moduan erabili nahi dituztenentzako [8]. Pixhawk kontroladoreak hainbat software onartzen dituzte, proiektu honetan **PX4** softwarea erabili da, hurrengo puntuan azalduko dira honen xehetasunak.

Pixhawk izenpean hainbat hardware desberdin garatu dira, Pixhawk modelo bakoitzak bere berezitasunak ditu eta egoera edo zeregin desberdinetan erabili ahal izateko daude eginak. 4.3 irudian ikusi daitezke Pixhawk modeloen adibide batzuk.

4.3 Irudia: Pixhawk-en web orri ofizialean eskaintzen diren modelo batzuk.

Fly free proiektuan **Pixhawk2** eta **Pixhawk4** modeloak erabili dira, lehena kontrolagailu

zentral bezala funtzionatzeko (**Raspberry Pi** batekin batera) eta bigarrena drone periferikoetan.

Pixhawk2

Pixhawk2 modeloa (4.4) kontrolagailu zentral bezala erabili da, honek esan nahi du hardware hau izango dela aginduak drone periferiko edo helizeei bidaliko dizkiena eta ondorioz aireontzia kontrolatuko duena.

Honen arrazoi nagusia, bigarren kapituluan aipatu den kontrol softwarea da. Pixhawk2 honek kontrol software konplexu bat darama drone osoa kontrolatu ahal izateko, baina programa hori **matlab** programazio lengoaiarekin dago egina, eta Pixhawk2 modeloa da oraingoz matlab-ekin bateragarria den pixhawk-en modelo bakarra.

4.4 Irudia: Pixwak2 hegaldi kontroladorea.

Pixhawk2-a komunikatu ahal izateko, honek duen **TELEM2** izeneko **UART** portua erabili da **Raspberry Pi**-aren UART portura konektatuta.

Pixhawk4

Pixhawk4 modeloa (4.5) da gaur egun arte atera den Pixhawk-en modelo aurreratuena, hau izan da beraz helize funtzioa beteko duten drone periferikoetan erabiltzea erabaki den hardwarea, honek eskaintzen dituen prestazioengatik.

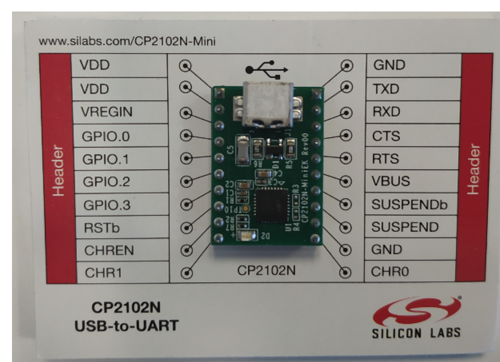
4.5 Irudia: Pixwak4 hegaldi kontroladorea.

Hauek, **Pixhawk2**-arekin gertatzen den bezala, **TELEM2** izeneko **UART** portuaren bidez konektatu dira **Raspberry Pi**-ari baina oraingoan azken honen USB portuetara, izan ere USB portuak UART bezala funtzionatu dezakete **USB-UART** bihurtzaile batekin konbinatzen badira.

4.1.3 USB-UART bihurgailua

Pixhawk4-ak **Raspberry**-aren **USB** portuetara konektatu ahal izateko, **USB-UART** bihurgailuak behar izan dira. Bihurgailu hauek ohiko UART portu bat USB portu batekin komunikatzea ahalbidetzen dute, izan ere, USB portu bat serieko portu bat denez, UART funtzioa egin dezake bihurgailu hauek erabiliz gero.

Mota honetako bihurgailu asko daude, aukeratutako bihurgailua **Silicon labs** enpresarena izan da, **CP2102N** modeloa zehazki.



4.6 Irudia: USB-UART bihurgailua.

4.6 irudian ikus daitezkeen pin guztietatik soilik **TXD** (transmsorea), **RXD** (hartzailea) eta **GND** (lurra) erabili dira, hauek baitira **UART** portuaren funtzioak betetzeko behar diren pinak.

Bihurgailua erabiltzeko **USB-nanoUSB** kable batekin (4.7 irudia) konektatu behar da, kasu honetan **Raspberry**-ra. Guztira lau biurgailu eta lau kable behar izan dira, bat helize bezala funtzionatuko duen drone bakoitzarentzat.

4.7 Irudia: USB-nanoUSB kablea.

4.2 Software

Hardwarearekin gertatzen zen ez bezala, software osagai gutxiago behar izan dira, baina pisu handikoak, izan ere, ondoren azalduko **PX4** softwarea izan da proiektua bideratu ahal izateko erabili den tresna garrantzitsuena.

Aipatutako PX4-az gain beste software batzuk ere erabili dira, garrantzia txikiagoa izan duten arren. Ondorengo ataletan azalduko dira hauen xehetasunak.

4.2.1 QGroundControl

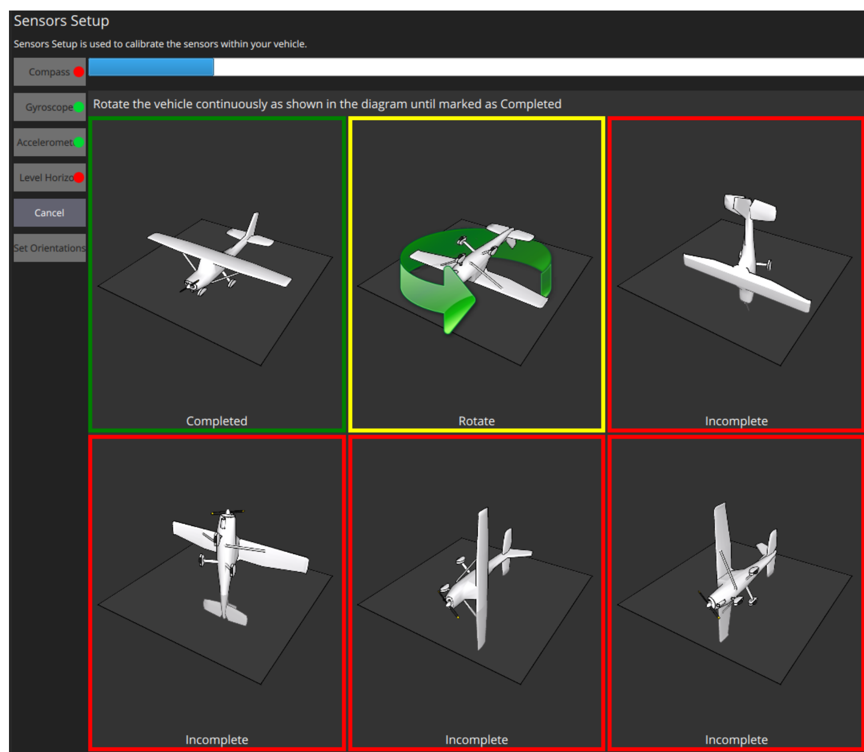
QGroundControl (4.8 irudia) hegaldiaren kontrol total bat eskaintzen du ordenagailu batetik. Bere helburu nagusia erabiltzaile profesionalak eta garatzaileak modu erraz batean erabiltzeko software bat izatea da [9]. **PX4** bezalaxe kode guztia irekia da eta edozeinek

egin ditzake modifikazioak eta softwarearen eboluzioan lagundu.

Proiektu honetan QGroundControl **Pixhawk** kontrolatzaileen kalibrazioak egiteko soilik erabili dira. Pixhawk-en sentsoreak kalibratu gabe datoz hasieran, hegan egiteko beharrezkoa da sentsore guztiak kalibratuta egotea, egon ezean ezin da dronea martxan jarri. QGroundControl erabili daiteke kalibrazio hau egin ahal izateko.

4.8 Irudia: QGroundControl-en logoa.

Sentsoreen kalibrazioa Pixhawk-a ordenagilura **USB** bidez konektatuz egin daiteke, behin konektatuta, aplikazioa irekitzean, honek detektatu egingo du eta sentsoreak kalibratuta ez badaude abixu bat emango du, ondoren kalibrazioa egiten hasi daiteke. 4.9 irudian ikusi daiteke kalibrazio honen adibide bat.



4.9 Irudia: Pixhawk4 baten kalibrazioa **QGroundControl** erabiliz.

4.2.2 MAVLink

MAVLink (*air vehicle communication protocol*, 4.10 irudia) droneekin komunikatzeko mezularitza protokolo bat da [10].

Mezuak **XML** formatuarekin definitzen dira. XML fitxategi bakoitzak MAVLink sistema jakin batek jasaten duen mezu sorta bat definitzen du, sorta hauei ”*dialekto*” deitzen zaie. Estazio eta autopiloto gehienak erabiltzen duten erreferentziazko mezu sorta *common.xml* fitxategian definitzen da, beste mezu sorta edo dialekto guztiak honen gainean eraikitzen dira.

4.10 Irudia: MAVLink-en logoa

MAVLink erabiltzen duen kontrol estazioetako bat **QGroundControl** litzateke, zeinak bere komunikazioak gauzatzeko erabiltzen duen. **PX4** autopilotoak ere MAVLink erabiltzen du komunikazio batzuk gauzatzeko, adibidez, kontsola erabiltzeko **Pixhawk** batean MAVLink erabiliz inplementatutako **Python** programa bat erabiltzen da.

4.2.3 PX4 autopilotoa

PX4 autopilotoa (4.11 irudia) kode irekiko hegaldi kontrol software proiektu bat da, drone eta bestelako ibilgailu hegalarietan erabiltzeko. Proiektuak drone garatzaileentzat tresna ugari eskaintzen ditu, drone aplikazioentzat soluzio egokiak sortzeko eta teknologiak partekatu ahal izateko [11].

PX4 drone hardware eta sorftware euskarriak sortu eta partekatzeko estandar bat hornitzen du, horrela, bai hardwarea eta bai softwarea modu eskalatu batean garatu eta mantentzeko ekosistema bat sortuz.

Hasieran PX4 aurretik aipatu den **Pixhawk** proiektutik jaio zen, Zurich-eko **ETH** unibertsitatean (munduko teknologia unibertsitate garrantzitsuenetako bat) ikusmen artifizial bidezko hegaldi kontrolean oinarritutako plataforma bat sortu zenean Pixhawk proiekturako. Gaur egun 300 kolaboratzaile baino gehiago ditu mundu mailan eta munduko konpainia berriztatzaileenek erabiltzen dute aplikazio industriala duten drone aplikazio sorta zabalak garatzeko. Pixhawk eta PX4 proiektuen inguruko kode irekiko komunitatea, droneekin zerikusia duten eta industria askoren babesa duen garapen komunitate handiena da.

PX4 **Dronecode** organizazioaren parte da, zeina **Linux fundazioa**rengatik kudeatuta dagoen eta ibilgailu hegalarietan kode irekiko softwarearen erabilera sustatzea duen helburu. Droncode **QGroundControl**, **MAVLink** eta **SDK** softwareak ere kudeatzen ditu, denak ibilgailu hegalarietan erabiltzeko tresnak.

4.11 Irudia: PX4-ren logoa.

PX4 Pixhawk-en hegaldi kontroladore guztietan erabili daiteke. Erabiltzeko, **USB** bidez konektatu behar da ordenagailura eta kodea bertan konpilatu. Hau eginda, jada prest dago drone batean jartzeko eta hegan egiteko, sentsore guztiak kalibratu ondoren (azkeneko hau **QGroundControl** programarekin egin behar da). PX4 izango da dronearen funtzio guztiak kontrolatuko dituena.

Software honek noski, aplikazioak garatzeko aukera ematen du. Aplikazio berriak **C++** lengoaian garatu behar dira, hau baita PX4 osoa programatuta dagoen lengoaia. **Matlab** lengoaiak ere eskaintzen du PX4 programatzeko aukera, betiere kode hori C++ lengoaiara itzultzen bada.

PX4-ren erabilera

PX4 zuzenean deskargatu daiteke **Git** erabiliz, hurrengoa da erabili beharreko komandoa:

```
git clone https://github.com/PX4/Firmware
```

Kode hau terminal batean erabilita, PX4-ren ”**Firmware**” karpeta lortuko dugu, bertan dago PX4 softwarearen iturburu kode guztia.

Softwarea **Pixhawk** batean erabiltzeko, Pixhawk-a USB bitartez konektatu beharko dugu ordenagailura, behin konektatuta Firmware karpetan ireki beharko dugu terminal bat PX4-ren iturburu kodea konpilatzeko eta Pixhawk-ean instalatzeko. Erabili beharreko komandoa desberdina da Pixhawk modeloaren arabera; adibidez **Pixhawk4** batean konpilatzeko hurrengoa erabili behar da:

```
make px4_fmu-v5_default upload
```

”*upload*” komandoa izango da softwarea Pixhawk-ean instalatuko duena, azkeneko zati hau gabe erabiltzen badugu soilik iturburu kodea konpilatuko da.

Komandoa exekutatu eta gero, kodea konpilatu eta ordenagailua arakatuko du konektatutako Pixhawk baten bila, aurkitzen badu, bertan instalatuko du konpilatutako kodea (4.12 irudia).

```
tecnalia109435@109435:~/src/Firmware$ make px4_fmu-v5 upload
ninja: Entering directory `/home/tecnalia109435/src/Firmware/build/px4_fmu-v5_default'
[3/3] uploading px4
Loaded firmware for board id: 50,0 size: 1653332 bytes (80.09%), waiting for the bootloader...

Attempting reboot on /dev/serial/by-id/usb-3D_Robotics_PX4_FMU_v5.x_0-if00 with baudrate=57600...
If the board does not respond, unplug and re-plug the USB connector.

Found board id: 50,0 bootloader version: 5 on /dev/serial/by-id/usb-3D_Robotics_PX4_BL_FMU_v5.x_0-if00
sn: 003400203038510b35323635
chip: 10016451
family: STM32F7[6|7]x
revision: Z
flash: 2064384 bytes
Windowed mode: False

Erase  : [====================] 100.0%
Program: [====================] 100.0%
Verify : [====================] 100.0%
Rebooting. Elapsed Time 24.597
```

4.12 Irudia: PX4 kodearen konpilazioa.

Konexiorik ez badu aurkitzen, lehenengo zatian itxaroten geratuko da gailu bat konektatu/detektatu arte edo erabiltzaileak konpilazioa eskuz eragotzi arte (4.13 irudia).

```
tecnalia109435@109435:~/src/Firmware$ make px4_fmu-v5 upload
ninja: Entering directory `/home/tecnalia109435/src/Firmware/build/px4_fmu-v5_default'
[4/4] uploading px4
Loaded firmware for board id: 50,0 size: 1653332 bytes (80.09%), waiting for the bootloader...


^C
 Upload aborted by user.
ninja: build stopped: interrupted by user.
Makefile:182: fallo en las instrucciones para el objetivo 'px4_fmu-v5'
make: *** [px4_fmu-v5] Interrupción
```

4.13 Irudia: PX4 kodearen konpilazioa huts egitean.

Behin kodea konpilatuta, kontrolagailua prest dago drone batean sartzeko eta hegan egiteko.

PX4-k kontsola bat ere eskaintzen du aplikazioak probatzeko praktikan jarri baino lehen (4.14 irudia). Kontsola erabili ahal izateko Pixhawk bat konektatu behar da ordenagailura, behin konektatuta Firmware karpetako "*Tools*" karpetan ireki beharko dugu terminal bat eta bertan exekutatu hurrengo komandoa:

```
python mavlink_shell.py
```

Komandoa irakurriz ikusi daitekeenez, kontsola hau *Python* erabiliz egindako programa bat da. Programa hau MAVlink-ekin konektatzen da, honi esker komunikazio bat sortzen da kontrolagailuarekin, komandoak bidali eta erantzunak jasotzeko aukera emanez.

```
tecnalia109435@109435:~/src/Firmware/Tools$ python mavlink_shell.py
Using port /dev/serial/by-id/usb-3D_Robotics_PX4_FMU_v5.x_0-if00
Connecting to MAVLINK...

nsh>
```

4.14 Irudia: PX4 kontsola.

Lehenetsita datozten aplikazioak ikusteko "*help*" komandoa erabili daiteke, hau exekutatuta eskura dauden komando guztien zerrenda bat agertuko zaigu pantailan (4.15 irudia).

Aplikazio hauek guztiak komando bezala erabili daitezke, edozein **Linux** sistema bateko exekutagarriek funtzionatzen duten antzera.

```
nsh> help
help usage:  help [-v] [<cmd>]

  [           dirname     false       mkfatfs     pwd         true
  ?           date        free        mkfifo      rm          uname
  basename    dd          help        mkrd        rmdir       umount
  break       df          hexdump     mh          set         unset
  cat         echo        kill        mount       sh          usleep
  cd          printf      ls          mv          sleep       xd
  cp          exec        mb          mw          test
  cmp         exit        mkdir       ps          time

Builtin Apps:
  adc
  adis16448
  attitude_estimator_q
  batt_smbus
  blinkm
  bl_update
  bma180
  bmi055
  bmi160
  bmm150
  bmp280
  bottle_drop
  bst
  camera_feedback
  camera_trigger
  cdev_test
  cm8jl65
  commander
  commander_tests
  config
  controllib_test
```

4.15 Irudia: PX4-ren lehentsitako aplikazio zerrendaren zati bat.

PX4 barnean aplikazio edo programa berriak egiteko hurrengo pausoak jarraitu behar dira:

1. Karpeta berri bat sortu *Firmware/src/examples/* helbidean, *adibidea* izenarekin esate baterako.

2. Karpeta horren barruan sortu behar diren **C** edo **C++** fitxategi guztiak, hala nola *adibidea1.c* edo *adibidea2.cpp*.

3. *CMakeLists.txt* izeneko fitxategi bat sortu karpeta barruan. Fitxategi horren barruan hurrengoa idatzi beharko da:

```
################################################################
#
#   Copyright (c) 2015 PX4 Development Team. All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions
# are met:
#
# 1. Redistributions of source code must retain the above copyright
#    notice, this list of conditions and the following disclaimer.
# 2. Redistributions in binary form must reproduce the above copyright
#    notice, this list of conditions and the following disclaimer in
#    the documentation and/or other materials provided with the
#    distribution.
# 3. Neither the name PX4 nor the names of its contributors may be
#    used to endorse or promote products derived from this software
#    without specific prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
# "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
# LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
# FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
# COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
# INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
# BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
# OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
# AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
# ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
# POSSIBILITY OF SUCH DAMAGE.
#
##################################################################
px4_add_module(
    MODULE examples__adibidea
    MAIN main
    STACK_MAIN 2000
    SRCS
        adibidea1.c
        adibidea2.cpp
    DEPENDS
        platforms__common
    )
```

”*adibidea*” jartzen duen toki guztietan, gure aplikazioak izango duen izena jarri beharko dugu eta *CMakeLists.txt* fitxatekiko *SRCS* atalean aplikazio horrek izango dituen C eta C++ fitxategi guztiak adierazi behar dira.

Pauso horiek emandakoan gure aplikazioa prest egongo litzateke. Baina konpilatzera-

koan **PX4**-ren barne egoteko, *default.cmake* izeneko fitxategi baten edukiera aldatu beharko dugu. Fitxategi hori desberdina izango da erabiltzen dugun hardwarearen arabera, *Firmware/boards* helbidean daude PX4-rekin bateragarri diren hardware guztietarako *default.cmake* fitxategiak.

Kasu honetan **Pixhawk4** baterako ari garenez garatzen, *Firmware/boards/px4/fmu-v5* helbidera jo beharko dugu, bertan, *default.cmake* fitxategiaren edukia aldatuko dugu, gure aplikazioa gehituz *EXAMPLES* deitutako atalean

```
EXAMPLES
        bottle_drop
        fixedwing_control
        example_fixedwing_control
        hello
        hwtest
        #matlab_csv_serial
        #publisher
        px4_mavlink_debug
        px4_simple_app
        rover_steering_control
        segway
        #subscriber
        uuv_example_app
        adibidea # Hemen jarri aplikazioaren izena
```

Behin hau eginda, konpilatzeko, aurretik azaldutako prozesu berdina jarraitu behar da. Konpilazio komandoa erabili ondoren, **PX4**-ren iturburu kodea berriz ere konpilatzen saiatuko da, sortu den aplikazio berria barne. Errorerik ez badago aplikazio berria jada sortuta eta erabiltzeko prest egongo da. Erroreak egonez gero, konpilazioa eragotziko da eta aurkitu diren erroreak adieraziko zaizkigu konpondu ahal izateko.

Aplikazio berria ondo konpilatu den ikusteko ”*help*” komandoa erabili dezakegu eskura dauden aplikazioak ikusteko. Gure aplikazio berria ondo konpilatu bada, zerrenda horretan agertuko da, 4.16 ikusi daitekeen bezala.

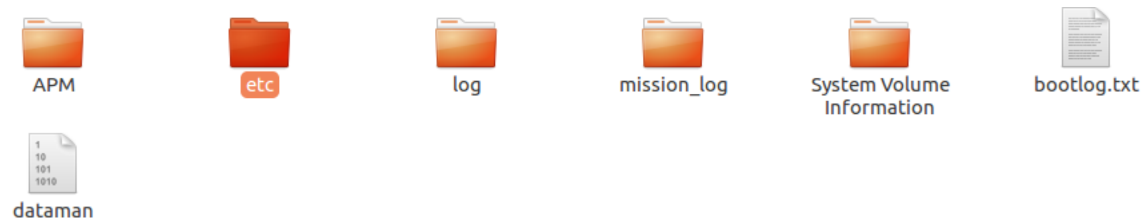
```
Builtin Apps:
  adc
  adibidea
  adis16448
  attitude_estimator_q
  batt_smbus
  blinkm
  bl_update
  bma180
  bmi055
  bmi160
  bmm150
  bmp280
  bottle_drop
  bst
```

4.16 Irudia: Sortutako aplikazioa aplikazio zerrendan.

Orain arte azaldutako hau terminal batez baliatuz exekutatzeko balio du, baina lortu nahi den helbururako, aplikazioaren exekuzioa hardwarea piztu bezain laster hastea behar da.

Hau lortzeko, Pixhawk-ak duen **microSD** memoria txartelaren edukia aldatu behar da. Txartelaren edukia atzitzeko **USB** bat erabili daiteke ordenagailu batera konektatuta. Behin ordenagailura konektatu dela, honen edukia ikusi eta aldatu daiteke.

Txartelean, besteak beste, pizteetan gertatutako erroreak, hegaldien informazioa eta abar gordetzen da fitxategi desberdinetan. Fitxategi hauen artean *etc* izeneko karpeta bat dago, honen edukia da aldatu behar dena nahi diren aplikazioak piztean abiarazteko. 4.17 irudian ikusu daiteke txartelaren edukiaren adibide bat.



4.17 Irudia: Pixhawk-en microSD txartelen edukia.

etc karpetaren barruan *extras.txt* izeneko testu fitxategi bat dago, zeina hasiera batean hutsik dagoen. Testu fitxategi honetan ezarri behar dira piztean abiarazi nahi diren aginduak *&* [1] sinbolo batez jarraituta. Demagun aplikazioaren izena "*adibidea*" dela, orduan

[1] Sinbolo honek agindua paraleloan exekutatuko dela esan nahi du, beste prozesuak oztopatu gabe.

honako hau idatzi beharko litzateke *extras.txt* fitxategian aplikazioa abiarazteko:

```
adibidea &
```

uORB mezularitza

uORB sistema, PX4 barnean dagoen argitaratze/harpideztze **API** bat da, barne-prozesuen arteko komunikazioetan erabiltzen dena. API honen bitartez lortzen dira Pixhawk-aren sentsoreen irakurketak eta egiten dira aldaketak dronearen kokapenean eta bestelako parametroetan. Automatikoki hasieratzen da sistema piztutakoan, PX4-ren aplikazio asko bere baitan funtzionatzen baitute [12].

uORB-en erabilera **uORB topikoen** bidez egiten da. Topiko hauek definitzen dute zein diren sisteman egin daitezkeen aldaketak zein irakurketak. Topiko erabilgarri asko daude, PX4-ren web-orrialde ofizialean[2] eskaintzen da grafo bat[3] topiko eskuragarri guztiekin, erabili nahi dena erraz aurkitu ahal izateko. Topiko guztien inplementazio fitxategiak *Firmware/msg* helbidean aurkitu daitezke iturburu kodearen karpetan. uORB topiko bat erabiltzeko bi modu daude, bat topiko horrek dituen balioak irakurtzeko eta bestea topiko horren balioak aldatzeko.

Topiko baten datuak **irakurtzeko**, topikoan harpidetu behar da, horetarako *orb_subscribe()* funtzioa erabiliz. Funtzio honek **int** bat izultzen du, harpidetza egin den topikoari erreferentzia egiten diona, hau da, fitxategi deskriptore (file deskriptor) bat lortuko dugu. Adibidez *vehicle_attitude*[4] izeneko topikoan harpidetu nahi badugu, honakoa da erabili beharreko kodea:

```
1 int sensor_sub_fd = orb_subscribe(ORB_ID(vehicle_attitude));
```

Hau egin ondoren, *sensor_sub_fd* aldagaian izango dugu topikoaren erreferentzia edo fitxategi deskriptorea eta aldagai hau erabili beharko da etorkizunean egiten diren eragiketetan. Irakurketa egin ahal izateko *px4_pollfd_struct_t* izeneko **struct** bat erabili behar da, struct honetan aurretik lortu dugun erreferentzia ezarri beharko dugu bere eremuetako batean:

[2] https://px4.io/
[3] https://dev.px4.io/en/middleware/uorb_graph.html
[4] Topiko honek dronearen posizioa adierazten du.

```
px4_pollfd_struct_t sensor_fd;
sensor_fd.fd = sensor_sub_fd; //Ezarri fitxategi deskriptorea
sensor_fd.events = POLLIN;
```

Orain gure topikoaren fitxategi deskriptorea *sensor_fd* aldagaian dugu gordeta.

Irakurketa egiteko, lehenik eta behin *px4_poll()* funtzioa erabili behar da, honen bidez jakingo dugu ea daturik eskura dagoen irakurtzeko. Funtzio honek *px4_pollfd_struct_t* motako struct bat, struct horretan dauden deskriptoreetatik zenbat irakurri nahi diren eta itxarote denbora (milisegundotan) eskatzen ditu parametro bezala. Honakoa da bere erabilera aurretik definitu dugun *sensor_fd* aldagaia erabiliz:

```
//Itxaron deskriptore bat gaurkotu arte 100 milisegundoz
int poll_ret = px4_poll(&sensor_fd, 1, 100);
```

Horrela erabilita funtzioak 100 milisegundoz itxarongo du *sensor_fd* aldagaiak erreferentziatzen duen fitxategi deskriptorean eguneratze bat egon arte. Eguneratzeak egon ezean -1 balioa itzuliko du eta eguneratzeak egotekotan (hau da, irakurtzeko datuak eskura badaude), 1 balioa itzuliko du. Hau kontuan izanda, sentsoreen irakurketa *px4_poll()* funtzioaren itzultze balioa 1 denean soilik egin ahalko dugu.

Azkenik, irakurketa gauzatzeko erabiltzen ari garen topikoari dagokion struct bat erabili beharko dugu. Topiko guztiek dute struct bat beraien datuak gorde ahal izateko. Kasu honetan, *vehicle_attitude* topikoa erabiltzen ari garenez, *vehicle_attitude_s* struct-a erabili beharko dugu. Struct horretan datuak gordetzeko *orb_copy()* funtzioa erabili behar da, honek zein topiko erabiliko dugun, topikoari erreferentzia egiten dion fitxategi deskriptorea (*sensor_sub_fd* kasu honetan) eta topikoari dagokion struct-a eskatzen ditu parametro bezala.

Hau guztia jakinda, errore kudeaketa txiki bat egin ondoren gauzatuko dugu sentsoreen irakurketa. Lehenik *px4_poll()* erabiliko dugu datuak eskura dauden ala ez jakiteko eta datuak eskura badaude *orb_copy()* erabiliko dugu sentsoreen datuak lortzeko:

```
//Ez bada irakurketarik egin errore kontadore bat eraman
int error_conuter;
if (poll_ret < 0)
{
                //Hau gertatzen bada, emergentzi bat dagoela esan nahi du
                 if (error_counter < 10 || error_counter % 50 == 0)
                 {
                         PX4_ERR("ERROR return value from poll(): %d", poll_ret);
                 }
                 //Errore kontagailuari bat gehitu
                 error_counter++;
}
else if (poll_ret>0 && sensor_fd.revents & POLLIN)
{
  //Datuak eskura daude, beraz sentsoreen irakurketa egin dezakegu
   struct vehicle_attitude_s raw_sensor;
   orb_copy(ORB_ID(vehicle_attitude), sensor_sub_fd, &raw_sensor);

}
```

Datuak eskura daudela aurkitzen bada, *raw_sensor* izeneko aldagaian izango ditugu sentsoreen irakurketen balioak. Aldagai hau *vehicle_attitude* motakoa denez, struct honen inplementazioa begiratu dezakegu zein eremu dituen ikusi eta lortutako informazioa erabili ahal izateko. Informazio hau PX4-ren web orri ofizialean aurkitu dezakegu lehen aipatutako grafoan:

```
# This is similar to the mavlink message ATTITUDE_QUATERNION, but for onboard use

uint64 timestamp        # time since system start (microseconds)

float32 rollspeed       # Bias corrected angular velocity about X body axis in rad/s
float32 pitchspeed      # Bias corrected angular velocity about Y body axis in rad/s
float32 yawspeed        # Bias corrected angular velocity about Z body axis in rad/s

float32[4] q            # Quaternion rotation from NED earth frame to XYZ body frame
float32[4] delta_q_reset    # Amount by which quaternion has changed during last reset
uint8 quat_reset_counter    # Quaternion reset counter

# TOPICS vehicle_attitude vehicle_attitude_groundtruth vehicle_vision_attitude
```

Dena garbiago egon dadin, horrela geratuko litzateke kodea, azaldutako zati guztiak batu ondoren:

```
//Topikoan harpidetu
int sensor_sub_fd = orb_subscribe(ORB_ID(vehicle_attitude));

px4_pollfd_struct_t sensor_fd;
sensor_fd.fd = sensor_sub_fd; //Ezarri fitxategi deskriptorea
sensor_fd.events = POLLIN;

//Errore kontagailua definitu
int error_conuter;

//Itxaron deskriptore bat gaurkotu arte 100 milisegundoz
int poll_ret = px4_poll(&sensor_fd, 1, 100);


if (poll_ret < 0)
{
                //Hau gertatzen bada, larrialdi bat dagoela esan nahi du
                 if (error_counter < 10 || error_counter % 50 == 0)
                 {
                         PX4_ERR("ERROR return value from poll(): %d", poll_ret);
                 }
                 //Errore kontagailuari bat gehitu
                 error_counter++;
}
else if (poll_ret>0 && sensor_fd.revents & POLLIN)
{
  //Datuak eskura daude, beraz sentsoreen irakurketa egin dezakegu
   struct vehicle_attitude_s raw_sensor;
   orb_copy(ORB_ID(vehicle_attitude), sensor_sub_fd, &raw_sensor);

}
```

Datuen **idazketa** egitea irakurketak egitea baino errazagoa da. Idazketak egiteko **publikatu** egin behar dugu harpidetu ordez, hau egiteko *orb_advertise()* eta *orb_publish()* funtzioak erabiltzen dira.

Lehenik eta behin aldatu nahi dugun topikoa iragarri behar da *orb_advertise()* funtzioaren bidez, honek topikoaren izena eta topiko horri dagokion struct-a hartzen ditu parametrotzat eta *orb_advert_t* motako struct bat itzultzen du. Adibidez *manual_control_setpoint*[5] topikoa hartuko dugu, topiko honi dagokion struct-a *manual_control_setpoint_s* da, hau erabiliko dugu *orb_advertise()* funtzioan:

```
struct manual_control_setpoint_s act;
memset(&act, 0, sizeof(act)); //memoria alokatu arazoak sahiesteko
orb_advert_t act_pub = orb_advertise(ORB_ID(manual_control_setpoint), &act);
```

[5] Topiko honek dronearen **roll**, **pitch**, **yaw** eta **thrust** balioak aldatzea ahalbidetzen du.

Orain egin nahi diren aldaketak *act* aldagaian egin behar dira. Esan bezala, *manual_control_setpoint_s* motakoa denez, *x*, *y*, *z* eta *r* eremuak aldatuko ditugu 0,5 balioa ezarriz adibide modura:

```
act.x = 0.5;  // roll
act.y = 0.5;  // pitch
act.z = 0.5;  // thrust
act.r = 0.5;  // yaw
```

Behin aldaketak egin direla *orb_publish()* funtzioa erabiliko dugu aldaketak argitaratzeko. Funtzio honek topikoaren izena, topikoa iragarritakoan lortu den struct-a (*orb_advertise()* funtzioarekin lortu dena) eta topikoari dagokion struct-a behar ditu parametro bezala:

```
orb_publish(ORB_ID(manual_control_setpoint), act_pub, &act);
```

Laburbilduz, horrela gelditzen da azaldu den kode guztia bateratuta:

```
//Definitu topikoari dagokion aldagaia eta iragarri
struct manual_control_setpoint_s act;
memset(&act, 0, sizeof(act)); //memoria alokatu arazoak sahiesteko
orb_advert_t act_pub = orb_advertise(ORB_ID(manual_control_setpoint), &act);

//Aldaketak egin
act.x = 0.5;  // roll
act.y = 0.5;  // pitch
act.z = 0.5;  // thrust
act.r = 0.5;  // yaw

//Aldaketak argitaratu
orb_publish(ORB_ID(manual_control_setpoint), act_pub, &act);
```

5. KAPITULUA

Garapena

Kapitulu honetan komunikazio softwarearen garapenenan zehar egindako guztia sakonki analizatuko da, garapen prozesua ahal den xehetasun handienarekin azalduz.

Kapituluan zehar komunikazioa ahalbidetzeko probatu diren arkitekturak azaldu, softwarearen kodea nola egin den argitu eta azkenik softwareak lortu nahi zen efizientziarekin funtzionatzeko egindako probak eta edukitako arazoak zehaztuko dira.

Argitu beharra dago kapitulu honetan zehar helize funtzioa beteko duten **Pixhawk4**-ei Pixhawk **esklaboak** edo **slaves** deituko zaiela eta kontrolagailu zentralaren funtzioa egingo duen **Pixhawk2**-ari Pixhawk **nagusia** edo **master**. Horrez gain argitu behar da erakusten diren kode zati gehienak sinplifikatuta daudela hauen ulermena errazteko asmoarekin, gainera funtzio eta fitxategi osoen inplementazioak A eta B eranskinetan ezarri dira txukunago geratu dadin.

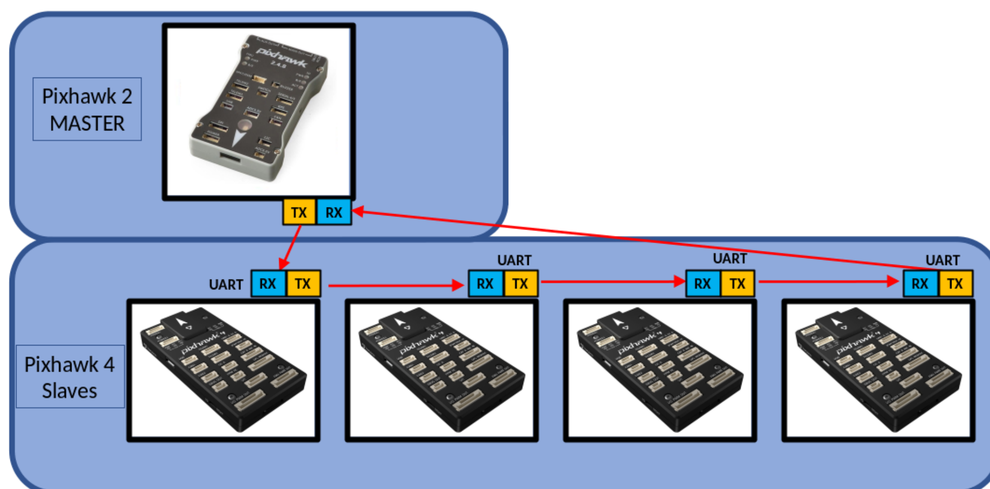
5.1 Aurrekariak

Garapena hasi aurretik arkitektura bat pentsatu behar zen. Arkitektura honen helburua, lau drone periferikoak kontroladore zentralarekin komunikatzea da eta beraz, helburu hau lortzeko hainbat konfigurazio proposatu ziren. Konfigurazio finalarekin eman arte proposamen hauetako batzuk probatu eta baztertu ziren, atal honetan zehar arkitektura horiek

azaldu eta analizatuko dira (xehetasun handietan sartu gabe), hauek baztertzearen arrazoiak argituz.

5.1.1 Arkitektura zirkularra

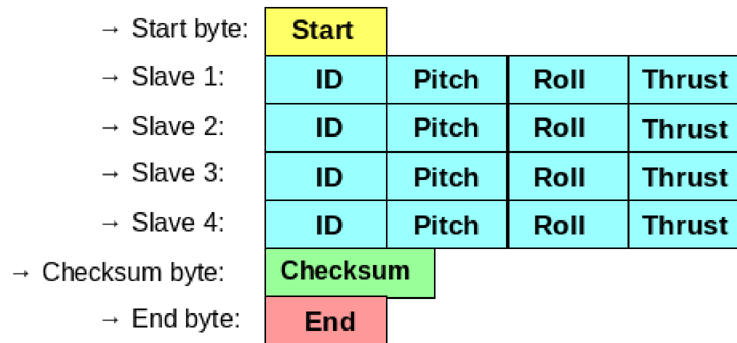
Probatu zen lehenengo arkitektura, arkitektura **arkitektura zirkularra** izan zen. Izen hau, hardware elementuak konektatzeko eragatik ematen zaio. Arkitektura honetan lau Pixhawk esklaboak eta Pixhawk nagusia bata bestearen **UART** portura (**TELEM2**) konektatzen dira zirkulu moduko bat osatuz, 5.1 irudian ikusi daitekeen moduan.



5.1 Irudia: Arkitektura zirkularraren eskema grafikoa.

Arkitektura hau erabiliz komunikatzeko, Pixhawk nagusiak informazio pakete bat bidaliko du, mezu honek egitura jakin bat izango du esklabo guztiak ulertu ahal izateko. Honakoa da mezuaren egitura (5.2 irudian ikusi daiteke adibide bat):

- **Identifikadore** bat, bat esklabo bakoitzeko, bakoitzak bere agindua jaso dezan.
- Esklabo guztientzako aginduak, **pitch roll** eta **thrust** parametroekin.
- **Start** eta **end** byte bat, mezuaren hasiera eta bukaera adieraziz, honen osotasuna ziurtatzeko.
- **Checksum** byte bat, hau ere mezuaren osotasuna ziurtatzeko.



5.2 Irudia: Arkitektura zirkularrean erabili den mezuaren egitura.

Mezua hasieran Pixhawk nagusiak bidaltzen du lehenengo esklabora, honek mezua ondo iritsi dela egiaztatzen du **start**, **end** eta **checksum** byteak egiaztatuz. Ondoren, bere aginduak bilatzen ditu **ID** byteaz baliatuz, agindu horiek jaso, helizearen posizioa aldatu agindutako posizioa ezarriz eta hau egindakoan jasotako mezua eguneratu helizearen posizio berriarekin. Azkenik mezu eguneratuaren checksum bytea kalkulatzen du eta hurrengo esklaboari pasatzen dio mezua, gauza berdina egin dezan. Prozesu hau, esklabo guztietan errepikatzen da Pixhawk nagusira bueltatzen den arte, nagusiak jasotzen duenean, helizeen posizio berriak egiaztatzen ditu eta hurrengo mezua bidaltzen du prozesua berrabiaraziz.

Arkitektura hau inplementatzen saiatzearen arrazoi nagusia, hardware nahiz software aldetik eskaintzen zuen sinpletasuna zen. Alde batetik hardware elementu gutxiren beharra zegoen, soilik ezinbestekoak ziren bost Pixhawk kontrolagailuak. Beste aldetik, softwarearen inplementazioa soilik Pixhawk-etan egin behar zen eta guztiek programa berdina erabiliko zuten (desberdintasun txikiekin), Pixhawk nagusiak izan ezik, honek aurretik azaldu bezala, kontrol softwarea eramango du.

Nahiz eta erraztasun batzuk dituen arkitektura honek arazo asko ditu. Arazorik larriena, Pixhawk edo kable bakar batek kale egitean sortzen da, izan ere komunikazio guztiak eteten dira batek kale egiten badu. Honek sendotasuna kentzen dio arkitekturari.

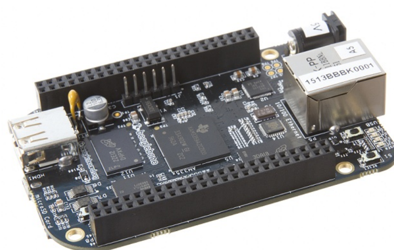
Nahiz eta arkitekturak sendotasun falta handia izan, inplementazioa egiten saiatu zen. Inplementazioan zehar eduki ziren arazoak izan ziren arkitektura baztertzearen arrazoi nagusiena. Arazo hauen artean, Pixhawk-en artean sortzen ziren blokeoak eta abiadura falta

zegoen, izan ere abiadura handietan komunikazioak blokeatu egiten ziren eta abiadura txikietan nahiz eta komunikazioak ez blokeatu arintasun falta handia nabaritzen zen.

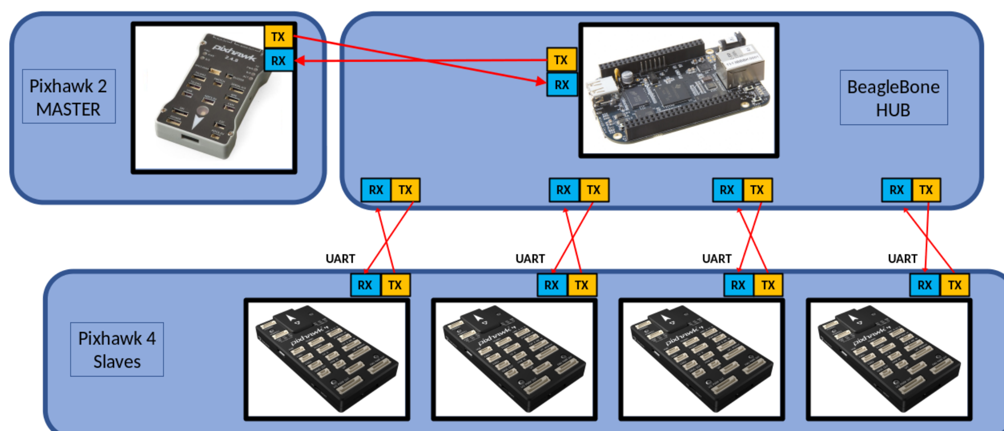
5.1.2 Arkitektura banatua: BeagleBone

Inplementatzen saiatu zen bigarren arkitektura izan zen **arkitektura banatua**, honek arkitektura zirkularrarekiko zuen desberdintasun eta abantaila nagusiena eskaintzen zuen sendotasun handia zen. Arkitektura banatua izaki, nahiz eta Pixhawk batek kale egin, sistemak funtzionatzen jarraitu dezake aurrekoan ez bezala.

Arkitektura zirkularrarekiko duen desabantailetako bat ordea, hardware gehiagoren beharra litzateke, hasieran **BeagleBone black** (5.3) bat erabiltzea erabaki zen bideratzaile bezala funtzionatzeko. BeagleBone hardware honek, bost **UART** portu erabilgarri eskaintzen zituen, beraz portu bat erabili zitekeen **Pixhawk** bakoitzarentzat.

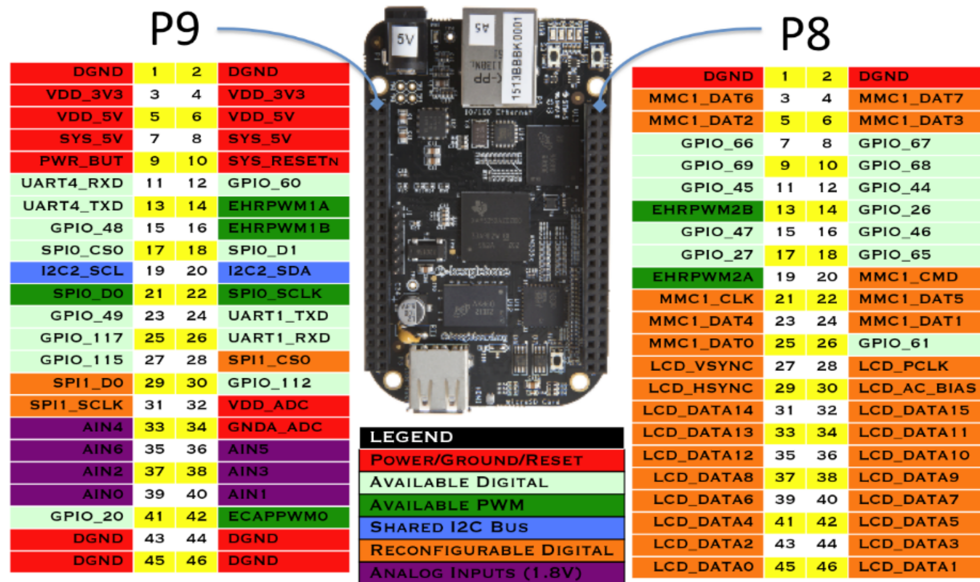


5.3 Irudia: Beaglebone black.



5.4 Irudia: Arkitektura banatua, **BeagleBone** bertsioa.

BeagleBone black-aren **UART** portu guztiak bere pin-ak konfiguratuz erabili behar ziren, arrazoi honengatik pre-programazio bat egin behar zen hardwarea era egokian erabili ahal izateko.



5.5 Irudia: BeagleBone-aren pin-en eskema.

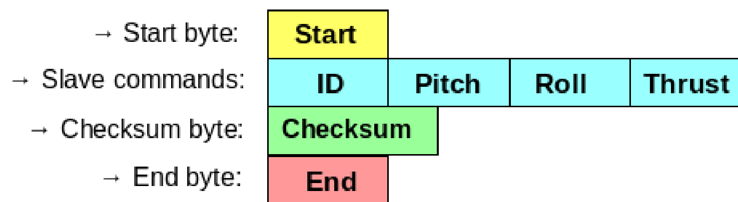
	RX	TX	CTS	RTS	Device	Remark
UART0	J1_4	J1_5			/dev/ttyO0	BeagleBone Black only
UART1	P9_26	P9_24	P9_20	P9_19	/dev/ttyO1	
UART2	P9_22	P9_21	P8_37	P8_38	/dev/ttyO2	
UART3		P9_42	P8_36	P8_34	/dev/ttyO3	TX only
UART4	P9_11	P9_13	P8_35	P8_33	/dev/ttyO4	
UART5	P8_38	P8_37	P8_31	P8_32	/dev/ttyO5	

5.6 Irudia: BeagleBone-aren **UART**-en kokapena.

Arkitektura hau erabiliz komunikatzeko modua zirkularrean baina askoz errazagoa litzateke. Kasu honetan, mezu handia, arkitektura zirkularrean erabilitakoa, (5.2 irudia) Pixhawk nagusitik bideratzaileari iristen zaio, (kasu honetan BeagleBone-ari) eta honek mezua lau

zatitan ebakitzen du esklabo bakoitzari soilik bere aginduak iristeko. Mezuaren zati bakoitzak **start**, **end** eta **checksum** byteak ere eramango ditu, osotasuna ziurtatzeko. 5.7 irudian ikusten da esklaboei iristen zaien mezuaren formatua.

Gainera, pixhawk guztiek gauza bera egiten dute, hau da, guztiek programa bera eraman dezakete desberdintasunik gabe, xehetasun honek asko errazten du programazioa. Aipatu behar da **ID** bytea orain ez dela beharrezkoa, bideratzeileak erabakitzen baitu zeini bidali zein mezu. Hala ere ID bytea mantetzea erabaki zen, etorkizunean beharrezkoa balitz informazio gehigarria bidaltzeko.



5.7 Irudia: Pixhawk esklaboei bideratzailetik iristen zaien mezua.

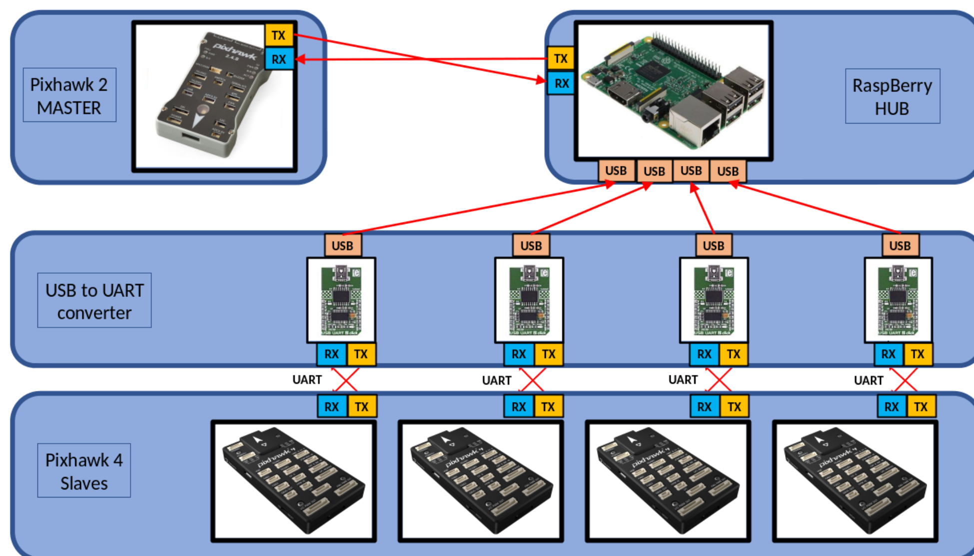
Esklaboak mezua jasotakoan, mezua osorik dagoela egiaztatu, aginduak gauzatu eta mezu berdina sortzen dute helizeen posizioekin. Mezu berri hori bideratzaileari iristen zaio lau esklaboen aldetik eta bideratzaileak berriz ere sortzen du mezu handi handi bat esklaboen helizeen posizioekin, nagusiari erantzuna bidaltzeko, prozesua berrabiaraziz.

5.6 irudian ikus daitekeen moduan sei **UART** portu daude BeagleBone-ean. Sei horietatik, batek soilik **TX** transmisorea du erabilgarri, beraz guztira bost dira guztiz funtzionalak diren portuak. Arazoak **UART0** portua konfiguratzerakoan etorri ziren, portu hau kontsola batean **debug** funtzioak betetzeko baitago erreserbatua. Nahiz eta era egokian konfiguratzea lortu zen, portu honek hainbat arazo ematen zituen komunikazioak gauzatzerako orduan, adibidez batzuetan irakurketa zikinak egiten zituen. Honez gain beste UART-ek ere arazoak ematen zituzten konfiguratzerakoan eta erabileran, gainera BeagleBone bat proba gutxi batzuk egin ondoren matxuratu egin zen eta horrek sendotasun faltaren sentsazioa ematen zuen. Arrazoi hauengatik guztiengatik arkitektura mantentzea erabaki zen baina bideratzailea aldatuz.

5.1.3 Arkitektura banatua: Raspberry Pi B+

BeagleBone hardwareak gabezia batzuk zituela ikusita, arkitektura eta protokoloa mantenduz hardwarea aldatzea erabaki zen. Aukeratutako hardwarea **Rasberry Pi B+** modeloa izan zen, honen arrazoia hardwareak eskaintzen dituen lau **USB** portuak eta **UART** portua izan ziren. Elementu hauei esker da posible komunikazioak gauzatzeko beharrezkoak diren bost Pixhawk-ak konektatzea.

Raspberry-ak BeagleBone-arekiko duen desabantaila nagusiena hain zuzen, aurretik aipatutako lau USB portuak dira. Lau portu hauek Pixhawk-en UART portuetara konektatu ahal izateko **USB-UART** bihurgailuak dira beharrezkoak, ondorioz lau hardware elementu osagarri erabili behar izan dira. 5.8 irudian ikusi daiteke nola geratzen den arkitektura Raspberry-arekin eta bihurgailuekin. Elementu osagarri hauen erabilera sendotasuna kentzen dio arkitekturari, zenbat eta elementu gehiago erabili, elementuren batek huts egiteko aukera gehiago dagoelako. Hala eta guztiz ere, Raspberry-ak komunikazioetan eskaintzen duen abiadurarekin eta portuak konfiguratzeko erraztasunarekin orekatzen du desabantaila hori.

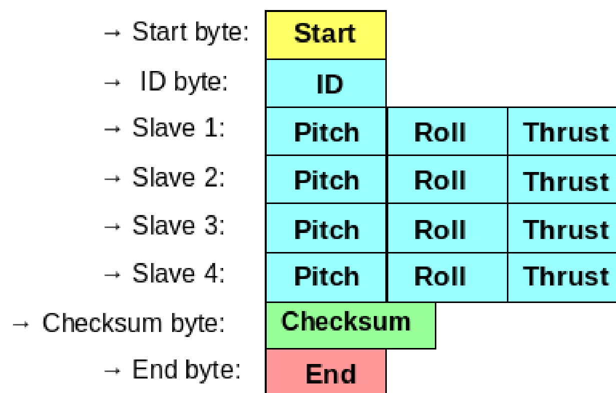


5.8 Irudia: Arkitektura banatua, **RaspBerry** bertsioa.

Erabili den **Raspberry** modeloaren eta **USB-UART** bihurtzailearen xehetasunak 4. kapi-

tuluan errepasatu dira sakonago, 4.1.1 eta 4.1.3 sekzioetan hurrenez hurren.

Komunikazio protokoloa ez da aldatzen BeagleBone-a erabiltzen zuen bertsiotik (ikusi 5.1.2 sekzioa xehetasun gehiagorako). Hala ere, masterrak bidaltzen zuen mezua zerbait aldatzea erabaki zen (5.2 irudia) inplementazioan zehar, honen arrazoia soberan zeuden **ID** byteak izan ziren. Aurreko sekzioan azaldu da esklaboei bidaltzen zaien ID bytea mantentzea erabaki zela informazio gehigarria bidaltzeko asmoarekin, baina nahikoa da masterrak ID byte bakar bat bidaltzea, gero byte berdina bidaliko die bideratzaileak esklabo guztiei. 5.9 irudian ikusten da mezu honen formatua grafikoki.



5.9 Irudia: Masterrak bideratzaileari bidaltzen dion mezua ID byte-ak kenduta.

Arkitekturaren bertsio honek ez zuen arazorik eman portuen konfigurazioan eta komunikazioetan abiadura egoki bat lortu zen komunikazio sinple batzuk probatu eta gero, hortaz **arkitektura banatuaren** bertsio hau erabiltzea erabaki da proiektuan.

5.2 Pixhawk inplementazioa (esklaboak)

Behin arkitektura eta konfigurazio egokiak aukeratuta, programazioarekin hasi behar da. Azaldutako komunikazio protokoloa inplementatzeko, bi alderdi desberdindu behar dira, alde batetik programazioa Pixhawk esklaboetan egin behar da, denek programa berdina eramango dute. Beste aldetik, bideratzailearen inplementazioa egin behar da (RaspBerry).

Pixhawk nagusiaren inplementazioa ere egin beharko litzateke, baina aurretik esan bezala, honek kontrol software konplexu bat darama eta ez da proiektu honen parte.

Inplementazioaz mintzatu ahal izateko, aurretik inplementazioaren xehetasun batzuk argitzea komeni da, bukaeran inplementazio osoa hobeto uler dadin. Sekzio honetan zehar xehetasun hauek azalduko dira, amaieran inplementazio osoa azaltzearekin batera.

5.2.1 Fitxategiak eta aurrekariak

Aurretik hainbatetan esan bezala, Pixhawk-ak programatzeko **PX4** autopilotoan aplikazio berri bat sortu behar da (4.2.3 sekzioan azaltzen da PX4-ren erabilera).

Sortutako aplikazioa **codisava** (***co****ntrol* ***dis****tribuido* ***ava****nzado*) deitzen da (**Fly Free** proiektuari eman zaion beste izen bat). Aplikazioaren inplementazioak hainbat zati ditu:

- Serieko portuen konfigurazioa.
- Mezuen jasotze, tratamendu eta bidaltzea.
- Mezuen bitartez jasotako aginduen tratamendua eta agindu horien argitalpena eragingailuetan.
- Sentsoreen irakurketa eta irakurketa horren interpretazio eta tratamendua.

Honakoak dira aplikazioak dituen fitxategiak:

- *serial_port.cpp* eta *serial_port.h*: serieko portuekin zerikusia duten funtzioen definizio eta inplementazioak (irekiera, irakurketa eta bidaltze funtzioak).
- *slave_controller.h* eta *slave_controller.cpp*: esklaboek egingo dituzten funtzio guztien definizio eta inplementazioak (irakurketak, bidalketak, mezuen tratamenduak...).
- *codisava.h* eta *codisava.cpp*: hasieraketak (portuaren izena, abiadura...) eta **main** programaren definizio eta inplementazioak.

5.2.2 Serieko portuak

Serieko portuen erabilera ohiko fitxategien antzera egiten da, agindu berdinak erabiliz. Serieko portu bat irekitzeko *open()* funtzioa eraili behar da, agindu honen bidez fitxategi baten eta fitxategi deskriptore (file descriptor) baten arteko lotura bat sortzen da. deskriptore hori sarrera/irteerako beste funtzio batzuk erabiliko dute ireki den fitxategi horri erreferentzia egiteko.

Funtzioa erabiltzeko, ireki nahi den fitxategiaren helbidea jakin behar da. Kasu honetan ez da fitxategi bat erabiliko, baina erabilera berdina izango da.

Pixhawk-etan, serieko portuek izen jakin batzuk dituzte, aipatu bezala proiektuan **TELEM2** izeneko serieko portua erabili da, honakoa da bere helbidea: */dev/ttyS2*.

Portuaren helbideaz gain, irekiera modua ere adierazi behar da aukera sorta baten artean. Modu honek irekiko den fitxategiaren portaera adieraziko du etorkizunean egingo diren eragiketetan. Huek dira irekiera modu garrantzitsuenak:

- O_RDONLY: Irakurketarako bakarrik irekiko da.
- O_WRONLY: Idazketatarako bakarrik irekiko da.
- O_RDWR: Idazketa eta irakurketarako irekiko da.
- O_NOCTTY: Karaktere bereziak alde batera utziko ditu irakurketak egiterakoan. Aukera hau bereziki erabilgarria da serieko portuak erabiltzerakoan.
- O_NONBLOCK: Hau ez bada ezartzen, irakurketa bat egiterakoan programaren exekuzioa blokeatu egingo da irakurtzeko datuak eskura egon arte, hau ezartzerakoan exekuzioa ez da blokeatuko eta ez da ezer irakurriko daturik ez badago eskura.

Beraz, hau guztia jakinda **Pixhawk**-aren portua ireki ahal izateko irakurketak nahiz idazketak egiteko, irakurketetan blokeatu gabe eta karaktere bereziak baztertuz, hurrengoa litzateke erabili beharreko kodea:

```
1 int fd = open("/dev/ttyS2", O_RDWR | O_NONBLOCK | O_NOCTTY);
```

Kode hau kontuan izanda, *fd* aldagaia ezango da etorkizunean portuarekin eragiketak egiteko erabili behar dena.

Eragiketak egiten hasi aurretik portua konfiguratzea komeni da, horretarako, *termios* izeneko **struct** bat erabili behar da. Struct hau soilik serieko portu bat ireki bada erabili daiteke, hurrengoa da bere erabilera:

```
struct termios config; //definitu termios aldagaia
tcgetattr(fd, &config); //lortu portuaren uneko konfigurazioa
```

Orain *config* aldagaian egin beharko dira aldaketak aurretik lortu da *fd* deskriptorean aplikatu ahal izateko. Dauden aukera guztietatik oraingoz soilik **baud** abiadura aldatzean jarriko dugu arreta, hau baita konfigurazioan parametro garrantzitsuena. Abiadura ezartzeko bi funtzio erabili beharko ditugu *cfsetospeed()* eta *cfsetispeed()*. Lehenengoak irteerako abiadura ezartzeko balio du, bigarrengoak aldiz sarrerako abiadura ezartzeko, abiadurak **baud**-etan adierazten dira. Hurrengo konstanteen bidez adierazi behar dira:

```
B0
B50
B75
B110
B134
B150
B200
B300
B600
B1200
B1800
B2400
B4800
B9600
B19200
B38400
B57600
B115200
B230400
```

Konstante hauek soilik erabili daitezke portuaren abiadurak definitzeko, kasu honetan **B115200** abiadura erabiltzea erabaki da, hau baita abiadura estandarrena serieko portuetan. Honela geratuko da konfigurazioaren kodea:

```
cfsetispeed(&config, B115200);
cfsetospeed(&config, B115200);
```

Behin konfigurazioa egin dela, orain konfigurazio berria *fd* deskriptoreari itzuli behar zaio, horretarako beste funtzio bat erabili behar da; *tcsetattr()*. Funtzio honek ere aukera batzuk eskaintzen ditu aldaketak nola gertatuko diren ezartzeko, honakoak dira aukera horiek:

- TCSANOW: Aldaketak segituan gertatuko dira.
- TCSADRAIN: Aldaketak deskriptorean irteerako bezala idatzitako datu guztiak transmitititutakoan gertatuko dira, aukera hau erabilgarria da da batez ere irteerari eragiten dieten parametroak aldatzean.
- TCSAFLUSH:Aldaketak deskriptorean irteerako bezala idatzitako datu guztiak transmititutakoan gertatuko dira eta gainera sarrera bezala jasotako datu guztiak baztertuko dira (jasotzen badira) aldaketak gauzatu baino lehen.

Aldaketak segituan gertatzea nahi denez, honela geratuko zen erabili beharreko kodea:

```
tcsetattr(fd, TCSANOW, &config);
```

Puntu honetan portuaren konfigurazioa jada eginda egongo litzateke, hortaz, laburpen modura horrela geratu zen kode osoa:

```
//Ireki erabili nahi den portua
int fd = open("/dev/ttyS2", O_RDWR | O_NONBLOCK | O_NOCTTY);

struct termios config; //Definitu termios aldagaia
tcgetattr(fd, &config); //Lortu portuaren uneko konfigurazioa

//Ezarri sarrerra eta irteera abidurak
cfsetispeed(&config, B115200);
cfsetospeed(&config, B115200);

//Aldaketak irekitako portuan gauzatu
tcsetattr(fd, TCSANOW, &config);
```

Normala den bezala, azalpena emateko erabili den kode hau proiektuan erabili denaren sinplifikazio bat da. Proiekturako erabili den kodean funtzio ugari egin dira errore kudeaketa egiten dutenak eta azaldu diren aukerak eta parametroak era erraz batean probatzea ahalbidetzen dutenak.

Serieko portuak erabiltzeko *open_serial()*, *_open_port()*, *start()*, *_setup_port()* eta *close_serial()* sortu dira *serial_port.h* eta *serial_port.cpp* fitxategietan, hona hemen funtzio hauen xehetasunak, inplementazioak A eranskinean daude:

- *_setup_port()*: Portuaren konfigurazio guztia egiten du, nahi den abiadura ezartzeko aukera emane, horretarako *switch* bat erabiltzen da. Beste aukera batzuk ezartzeko aukera ere ematen du, adibidez irakurketan blokeatu nahi den ala ez, baina proba askoren ondoren aukera hauek beti berdinak izango direnez azkenean abiadura bakarrik da aldagarria, beste aukerak konstante diren bitartean. Funtzio hau pribatua denez ezin da kanpoko fitxategietan erabili. A.1 eranskinean dago funtzioaren inplementazioa.
- *_open_port()*: Exekuzioaren aurretik definitu den portua (*m_uart_name* aldagaia) irekitzen du errore posibleen kudeaketa eginez. Funtzio hau ere pribatua da. A.2 eranskinean dago funtzioaren inplementazioa.
- *open_serial()*: Aurreko bi funtzioak biltzen ditu, ezarritako serieko portua irekiz eta konfiguratuz eta hauek sortu dezaketen erroreen kudeaketa eginez. Funtzio hau ere pribatu bezala definitu da. A.3 eranskinean dago funtzioaren inplementazioa.
- *start()*: Parametro guztiak hasieratzen ditu eta *open_serial()* funtzioa erabiltzen du nahi den portua ireki eta konfiguratzeko. Hau izango da beste fitxategietan portuak irekitzeko erabili behar den funtzioa, bere baitan biltzen ditu aurretik azaldutakoak. A.4 eranskinean dago funtzioaren inplementazioa.
- *close_serial()*: Irekitako serieko portua ixten du, errore kudeaketa txiki bat eginez. A.5 eranskinean dago funtzioaren inplementazioa.

5.2.3 Mezuen jasotzea

Mezuak, lehenago azaldu bezala, 5.7 irudiko formatuarekin iritsiko dira portura. Mezuak tratatu ahal izateko, lehenik eta behin irekitako portua irakurri behar da, horretarako *read()*

funtzioa erabili daiteke. Funtzio honek fitxategi deskriptore bat, buffer bat (irakurketaren emaitza gordetzeko) eta irakurri behar den mezuaren tamaina behar ditu (byte-tan). Esklaboen mezuek 16 byteko tamaina dute **pitch**, **roll** eta **thrust** eremuek 4 byteko tamaina dutelako. Suposatuz aurretik irekitako fitxategi deskriptore bat dugula (*fd*), honakoa litzateke agindu honen erabilera:

```
char buf[16];
int bytesRead =0;
bytesRead = read(fd, buf, 16);
```

Aginduak *buf* aldagaian gordetzen ditu irakurritako datuak eta irakurritako byte kopurua itzultzen du. Irakurketa hauek egiteko *read_port_frame()* funtzioa sortu da *serial_port.cpp* fitxategian.

Hasieran *read()* agindua bere kabuz erabiltzeak arazo batzuk eman zituen portuan gordetako datu guztiak osorik irakurtzeko eta zatika irakurtzen zituen 16 byteak, denak batera irakurri ordez. Arazo honi irtenbidea emateko sortu zen hain zuzen *read_port_frame()* funtzioa. Funtzio honek 16 byteen irakurtzea behartzen du, hau lortzeko kontagailu bat ezartzen da guztira irakurri diren byte kopuruarekin eta nahi den kantitatea irakurri arte deitzen zaio *read()* aginduari, irakurketa bakoitzeko emaitza bufferrean gordez.

Gainera, errore kudeaketa egin ahal izateko, bufferrean gorde den informazioa pantailaratzeko begizta bat ere sortu da. Pantailaratzeak noiz egin nahi diren kontrolatzeko *showPrints* izeneko boolear bat erabili da, honek egiazko balioa duenean soilik pantailaratzen da mezuaren edukia, izan ere pantailaratzeak konputazio kostua gehitzen dio programari eta exekuzioa abiadura moteldu dezake. A.6 eranskinean dago funtzio honen inplementazio osoa.

Irakurtzen diren datuen egituraren formatua 16 bytez osatuta dagoenez, denak positiboak izatea nahi dugu, horregatik datu negatibo bat irakurtzen bada pantailaratu baino lehen bihurtze bat egin behar zaio 256 gehituz, balio zuzena erakutsi dezan.

Funtzioak parametro bezala eskatzen duen *buf* aldagaian gordeko du irakurketaren emaitza, hona hemen bere erabileraren adibide bat, irakurritakoa pantailan erakutsi nahi badugu:

```
//Lehenengo serieko portua ireki eta konfiguratu
#include "serial_port.h"
SerialPort m_inputPort;
m_inputPort.Start("/dev/ttyS2", 115200, false, false);

//Definitu erabiliko diren aldagaiak
char buf[16];
int len = 16;

//Irakurketa egin
m_inputPort.read_port_frame(buf, len, true);
```

5.2.4 Mezuen bidaltzea

Mezuak jasotzen diren bezala, bidali ere egin behar dira. Bidaltzea egiteko *write()* funtzioa erabili daiteke. Funtzio honek fitxategi deskriptore bat, idatziko den edukia eta edukiaren luzeera jasotzen ditu parametro bezala.

write() erabiltzeko *send_message_buf()* funtzioa sortu da *serial_port.cpp* fitxategian. Funtzio honek mezuaren edukia, *char array* batean, eta mezu horren luzeera jasotzen ditu parametro bezala *write()* funtzioari barrutik deitzeko. Bidalitako byte kopurua itzuliko du. A.6 eranskinean ikusi daiteke inplementazioa.

Erabiltzeko, serieko portu bat definitu, ireki, mezu bat ezarri eta mezu hori bidali behar da:

```
//Lehenengo serieko portua ireki eta konfiguratu
#include "serial_port.h"
SerialPort m_inputPort;
m_inputPort.Start("/dev/ttyS2", 115200, false, false);

//Mezua bidali
m_inputPort.read_port_frame("kaixo", 5);
```

5.2.5 Mezuen tratamendua

Mezuak jasotakoan hauen edukia tratatu behar da. Lehenik eta behin mezua osorik datorren ala ez egiaztatu behar da, horretarako, *IsValid()* eta *CalculateChecksum()* funtzioak inplementatu dira *slave_controller.cpp* fitxategian.

CalculateChecksum() funtzioak, bere izenak adierazten duen bezala, iristen den mezuaren **checksum** bytea kalkulatzen du. *IsValid()* barruan erabiltzen da. 3. kapituluan azaldu bezala, checksum byte-a kalkulatzeko mezuaren eduki guztiaren arteko **xor** eragiketa bat egiten da. Kontuan izanda mezuek 16 byte dituztela eta **STX** (start) zein **ETX** (stop) byteak ez direla kontuan hartzen eragiketa egiteko. Inplementazioa A.8 eranskinean ikusi daiteke.

Behin checksum bytea kalkulatzeko funtzio bat prest izanda, bakarrik STX eta ETX byteak ondo daudela egiaztatzea falta da *IsValid()* funtzioan. A.9 eranskinean dago inplementazioa.

Mezua zuzen iritsi dela egiaztatu ondoren, edukiaren tratamendua egin behar da. Edukian, **pitch**, **roll** eta **thrust** parametroak ditugu, bakoitzaren balioa 4 byte-tan zatituta. Hiru parametro hauen benetako balioa lortu ahal izateko bihurtze bat egin behar da balioa bateratzeko. pitch eta roll parametroen balioak Eulerren angeluak dira, thrust parametroarena balioa ordea, zenbaki erreal positibo bat.

Lortu nahi ditugun balioak $[0, 2^{31}]$ (lau byte) tarteko balioak dira, $[0, 2^8]$ (byte bat) tarteko balioa duten lau zati bateratuz. Bihurtze hau egiteko modua desberdina da pitch eta roll balioentzat edo thrust balioarentzat. **Pitch** eta **roll** $[-\pi, \pi]$ tarteko balioa izango dute, **thrust** $[0, \infty]$ tarteko balioa izango duen bitartean.

Desberdintasun hauek kontuan izanda, bihurtzea egiteko lehenengo 4 bytetan banatutako balioak bateratu behar dira hauen balioak aldagai bakar batean batuz. Hau lortzeko batuketa bat egin behar da, byte bakoitza berari dagokion bit posizioarekin biderkatuz; lehenengo byteari 2^0 posizioa dagokio, bigarrenari 2^8, hirugarrenari 2^{16} eta laugarrenari 2^{24}. Horrela geratzen da eragiketa matematikoki:

$$jasotakoBalioa = byte1 + byte2 * 2^8 + byte3 * 2^{16} + byte4 * 2^{24}$$

Behin balioa bateratuta dagoela bihurtzea egin dezakegu. **Pitch** eta **roll** balioentzako hurrengoa da bihurtzeko modua:

$$benetakoBalioa = \frac{jasotakoBalioa * 2\pi}{2^{31}} - \pi$$

Thrust balioarentzat:

$$benetakoBalioa = \frac{jasotakoBalioa}{2^{31}}$$

Bihurtzeak egiteko, *bytesToAngles()* funtzioa inplementatu da *slave_controller.cpp* fitxategian. Funtzio honek buffer bat hartzen du, non jasotako mezua egongo den eta 4 bytetan zatituta dauden balioak aldagai bakar batean biltzen ditu, nahi den angelua lortzeko bihurtzea eginez. Begiratu A.10 eranskina inplementazioa ikusteko.

Bere erabilera honakoa da, suposatuz aurretik irakurritako eta egiaztatutako mezu bat *buf* aldagaian gordeta dugula:

```
double pitch = 0;
double roll = 0;
double thrust = 0;

bytesToAngles(pitch, roll, thrust, buf);
```

Mezuak jasotakoan bihurtze bat egin behar bada, mezuak bidaltzerakoan ere gauza bera egin behar da sentsoreen irakurketekin. Kasu honetan bihurtzea kontrako moduan egin behar da, balio bakar bat 4 bytetan banatuz. Horretarako, lehenik eta behin irakurketatik lortutako balioa, zeinak $[-\pi, \pi]$ tarteko balioa izango duen, $[0, 2^{31}]$ tarteko balio batean bihurtu behar da hurrengo moduan:

$$bidaltzekoBalioa = (benetakoBalioa + \pi) * \frac{2^{31}}{2\pi}$$

Behin balio hau lortu dela, hau izango da 4 byte-tan banatuko dena. Banatzea egiteko, batzean bezalaxe, balioaren biten posizioak hartu behar dira kontuan baina oraingoan zatituz biderkatu ordez. Zatiketa bakoitzeko emaitzatik zati osoa hartuko da bytearen balioarentzako eta zatiketa horren hondarra erabiliko da hurrengo zatiketan, beharrezko 4 byteak lortu arte:

$$hondarra1 = bidaltzekoBalioa \mod 2^{24}; byte1 = \frac{bidaltzekoBalioa}{2^{24}}$$

$$hondarra2 = hondarra1 \mod 2^{16}; byte2 = \frac{hondarra1}{2^{16}}$$

$$hondarra3 = hondarra2 \mod 2^{8}; byte3 = \frac{hondarra2}{2^{8}}$$

$$byte4 = hondarra3$$

Azken bihurtze hau egiteko, bi funtzio egin dira, baita ere *slave_controller.cpp* fitxategian:

- *convertProcessValue()*: funtzio honek sentsoreetatik lortutako irakurketaren balioa $[0, 2^{31}]$ tarteko balio batean bihurtzeko erabiltzen da, **double** motako datu bat hartu eta **int** motako datu batean bihurtzen du. Inplementazioa A.11 eranskinean.
- *split4bytes()*: aurretik bihurtutako $[0, 2^{31}]$ tarteko balio bat 4 bytetan banatzen du, balioak **array** batean sartuz. **int** bat eta **char array** bat jasotzen ditu parametro bezala. Inplementazioa A.12 eranskinean.

Hortaz, suposatuz *pitch* eta *roll* izeneko bi aldagai ditugula, sentsoreetatik lortutako irakurketen balioekin, hurrengo moduan egiten da balio hauen bihurtzea:

```
//Definitu banatutako balioak gordetzeko array-ak
unsigned char pitch4bytes[4];
unsigned char roll4bytes[4];

//Lortu 0-tik 2^31-ra balioa doan balioa bi irakurketetatik
truePitch = convertProcessValue(pitch);
trueRoll = convertProcessValue(roll);

//Banatu balio horiek 4 bytetan
split4bytes(truePitch, pitch4bytes);
split4bytes(trueRoll, roll4bytes);
```

5.2.6 Sentsore eta eragingailuen kudeaketa

Aginduen argitalpena eragingailuetan

Mezuak jaso eta tratatu direnenean aginduen edukia argitaratu behar da dronearen eragingailuetan. Horretarako, **PX4** softwareak eskaintzen duen **uORB API**-a erabili da. API

honen erabilera sakonago azaldu da 4. kapituluan 4.2.3 sekzioan.

Nahi diren eragingailuak atzitzeko *manual_control_setpoint* topikoa erabili beharko da. Topiko hau erabiltzeko modua aurretik azaldu da, beraz, adibide horretan oinarrituta, lehenengo topikoa **iragarri** behar da. Hau lortzeko, lehenengo topikoari dagokion struct-a definitu behar da (aldagaiarentzako memoria alokatzea ere komani da):

```
struct manual_control_setpoint_s act;
memset(&act, 0, sizeof(act));
orb_advert_t act_pub = orb_advertise(ORB_ID(manual_control_setpoint), &act);
```

Orain eragingailuen parametroen balioak aldatzeko, sortu den *act* aldagaia aldatu behar da. Suposatu jada mezu bat jaso dela eta bere aginduen edukia tratatu dela (aurretik azaldutako moduan) *pitch*, *roll* eta *thrust* aldagaietan gordez.

Oso kontuan izan behar da topiko honek -1-etik 1-era doazen balioak ulertzen dituela, hortaz, jasotako agindu bat (zeina $[-\pi, \pi]$ tartean dagoen) topiko honetan erabiltzeko, π balioarekin zatitu behar da, proportzionala lortzeko:

```
act.x = roll / M_PI;    // roll
act.y = pitch / M_PI;   // pitch
act.z = thrust / M_PI;  // thrust
act.r = 0.0;            // yaw
```

Azkenik, argitalpena egiteko, *orb_publish()* funtzioa erabili:

```
orb_publish(ORB_ID(manual_control_setpoint), act_pub, &act);
```

Argitalpena egiteko kontuan izan behar da lehenengo eragingailuak armatu behar direla. Horretarako beste topiko bat erabili daiteke, *actuator_armed* izenekoa. Honen erabilera aurrekoaren antzekoa da, lehenengo iragarri, aldatu eta azkenik aldaketak argitaratu:

```
//Topikoa iragarri eta dagokion structa lortu
struct actuator_armed_s arm;
memset(&arm, 0, sizeof(arm));
orb_advert_t act_pub_armed = orb_advertise(ORB_ID(actuator_armed), &arm);


//Armatuta ez badago, armatu
if(!arm.armed)
```

```
9       arm.armed=true;
10
11
12  //Aldaketak argitaratu
13  orb_publish(ORB_ID(actuator_armed), act_pub_armed, &arm);
```

Sentsoreen irakurketa eta irakurketaren kudeaketa

Behar den irakurketa dronearen posizioarena da. Irakurketa hau egiteko *vehicle_attitude* topikoa behar da. Topiko honek aireontziaren posizioa **koaternoi** baten bidez ematen du. Koaternioiak zenbaki konplexuen hedadura bat dira. Koaternioien multzoak lau dimentsioetako bektore espazio bat osatzen du. Multzo hau $\mathbb{R}^4$ multzoarekin identifikatu daiteke, alegia, errealen gaineko 4 dimentsioetako bektore espazio bat osatzen dute. Koaternoien bidez orientazio eta errotazioak adierazi daitezke, hortaz aireontzi baten posizioa adierazteko ere balio du.

Kasu honetan koaternoiak ez dute balio topikoan zuzenean erabiltzeko, hortaz koaternoi horretatik atera behar dira **pitch** eta **roll** parametroen balioak. Hau lortzeko funtzio batzuk inplementatu dira *slave_controller.cpp* fitxategian. Inplementazioa egiteko informazioa iterneteko orrialde batean lortu da [1]. Funtzioen inplementazioak A.13 eranskinean daude.

5.2.7 Inplementazio orokorra

Inplementazioaren xehetasun guztiak azalduta, honen garapen orokorra azaldu daiteke. Hau azaltzeko, aplikazioa osatzen duten fitxategien eduki guztia analizatu eta argituko da.

codisava.cpp eta codisava.h

codisava.h fitxategian aplikazioan erabiltzen diren konstante guztien definizioak daude, modu honetan zerbait aldatu behar bada hemendik aldatu daiteke zuzenean. Bere edukia

[1] http://www.euclideanspace.com/maths/geometry/rotations/conversions/quaternionToEuler/index.htm

B.1 eranskinean ikusi daiteke.

Jarraian konstanteak zertarako diren azalduko da:

- CODISAVA_BAUD_RATE: Irekiko diren serieko portuen abiadura definitzen du, abiadura desberdinak probatu dira, baina azkenean abiadura estardarra erabiliko da; 115200.
- CODISAVA_INPUT_PORT eta CODISAVA_OUTPUT_PORT: Datuak jaso eta bidaltzeko irekiko den serieko portuaren helbidea adierazten dute, nahiz eta desberdinak izan daitezkeen azkenean portu berdina erabiltzea erabaki zen eta beraz, balio berdina dute biek.
- CODISAVA_SLAVE_SLEEP_USECS: Iterazioen abiadura moldatzeko sartu ziren *usleep()* sistema deien balioa. Exekuzioa geldiarazten da adierazitako mikrosegundo kopuru horretan.
- SLAVE_FRAME_SIZE: Esklaboen mezuen tamana bytetan definitzen du.
- CODISAVA_SENSOR_READING_FREQ: Sentsoreen irakurketa maiztasuna ezarriko du.
- CODISAVA_MASTER_SLEEP_USECS eta MASTER_FRAME_SIZE: Hasieran erabili ziren ”master faltsu” bat egiteko asmoarekin, bukaeran ez dira ezertarako erabili.

codisava.cpp fitxategian dago **main** funtzioa, fitxetagi honetatik jaurtitzen da aplikazio osoa. Hasieran 4 exekuzio modu definitu ziren (*Master*, *Send-data*, *Slave* eta *None*), exekuzio modu desberdin hauek probak egiteko balio zuten eta aplikazioaren portaera definitzen zuten exekuzioa hasieratzerakoan. Bukaeran soilik *Slave* exekuzio modua erabiltzen da, honek aplikazioa esklabo funtzioarekin hasieratzen du. Inplementazioa B.2 eranskinean ikusi.

Exekuzio modua definitzeko sortu da *ParseExecutionMode()* funtzioa, honek komando bidez sartutako argumentuak hartu eta egiaztatzen ditu. **main** programak erabakiko du, sartutako argumentuen, arabera zein den exekutatuko den programaren hurrengo zatia. Hau egiteko *SlaveController* eta *TestDataSender* motako bi aldagai definitu dira (klase

hauek dagokien fitxategietan definituta daude) eta ezarritako exekuzio moduaren arabera bata edo bestearen *Run()* funtzioa exekutatzen du. Aipatu behar da proiektuaren amaierarako soilik **Slave** exekuzio modua erabiltzen den arren, bertan utzi direla bestelako aukerak badaezpada ere.

serial_port.cpp eta serial_port.h

Izenak argitzen duen legez, bi fitxategi hauek serieko portuekin zerikusia duten funtzio guztiak gordetzen dituzte.

serial_port.h fitxategian klasearen eta honek dituen funtzio guztien definizioak daude, konstante batzuk ere definituta daude baina azkenean konstante hauek baztertu egin dira eta ez dira erabiltzen. Fitxategiaren edukia B.3 eranskinean.

serial_port.cpp fitxategian daude funtzioen inplementazioak. 5.2.2 sekzioan azaldu dira *open_serial()*, *_open_port()*, *start()*, *_setup_port()* eta *close_port()* funtzioenak eta 5.2.4 zein 5.2.3 sekzioetan *send_message_buf()* eta *read_port_frame()* funtzioenak.

Azaltzeko falta den bakarra *resynchronize()* funtzioa da, honek duen helburua, mezu bidalketan zehar gertatu daitezkeen desinkronizazioak konpontzea da. Helburua lortzeko, mezu bat jasotzen du parametro bezala eta hau arakatzen du start bita aurkitzen duen arte, honen ondoren dagoen mezuaren zatia gordetzeko. Honen inplementazioa A.14 eranskinean.

slave_controller.cpp eta slave_controller.h

Bi fitxategi hauetan aurkitzen da aplikazioaren zati garrantzitsuena, esan daiteke hemen dagoela inplementatuta ”benetako” **main** programa.

Beti bezala, *slave_controller.h* fitxategian zenbait definizio egongo dira, baina oraingo honetan funtzio gehienen inplementazio eta definizioak zuzenean *slave_controller.cpp* fitxategian egin dira, izan ere, soilik hemen erabiltzeko daude eginak. *slave_controller.h* fitxategiaren edukia B.4 eranskinean ikusi daiteke.

Run() funtzioan dago aplikazioaren muina. Esan bezala, *codisava.cpp* fitxategian dago honen inplementazioa. Fitxategi honetako beste funtzioen inplementazioen funtsak aurreko sekzioetan zehar azaldu dira, hortaz, oraingoan *Run()* funtzioarenak analizatuko dira, honen inplementazioa ulertu ahal izateko. Funtzio honen inplementazio osoa A.19 eranskinean ikusi daiteke.

Lehenik eta behin erabiliko diren serieko portuak ireki behar dira, bai sarrerarako, bai irteerarako. Horretarako *start()* funtzioa erabili da:

```
1 m_inputPort.start(CODISAVA_INPUT_PORT, CODISAVA_BAUD_RATE, m_useNonBlockingIO,
      m_outputConsoleMsgs);
2 m_outputPort.start(CODISAVA_OUTPUT_PORT, CODISAVA_BAUD_RATE, m_useNonBlockingIO,
      m_outputConsoleMsgs);
```

Erabiliko diren **uORB** topikoak ere definitu behar dira (ikusi 4.2.3 azalpen sakon baterako). Kasu honetan erabiliko diren topikoak *manual_control_setpoint* (eragingailuen kontrolerako) eta *actuator_armed*(eragingailuak armatzeko) dira idazketarako eta *vehicle_attitude* irakurketarako. Lehenengo biak iragarri egin behar dira gero erabili ahal izateko, azkenekoan ordea harpidetu egin behar da:

```
1  struct manual_control_setpoint_s act;
2  memset(&act, 0, sizeof(act));
3  orb_advert_t act_pub = orb_advertise(ORB_ID(manual_control_setpoint), &act);
4
5  struct actuator_armed_s arm;
6  memset(&arm, 0, sizeof(arm));
7  orb_advert_t act_pub_armed = orb_advertise(ORB_ID(actuator_armed), &arm);
8
9
10 int sensor_sub_fd = orb_subscribe(ORB_ID(vehicle_attitude));  orb_set_interval(sensor_sub_fd,
       CODISAVA_SENSOR_READING_FREQ);
11 px4_pollfd_struct_t sensor_fd;
12 sensor_fd.fd = sensor_sub_fd;
13 sensor_fd.events = POLLIN;
```

Hemen orain arte azaldu ez den funtzio bat erabiltzen da, *orb_set_interval()* izenekoa. Honen bitartez sentsoreen irakurketa maiztasuna ezartzen da.

Hasieraketekin bukatzeko, erabiliko diren aldagai guztiak definitu dira. Lehenengo, sentsoreen irakurketak gordeko dituzten aldagaiak:

```
//Variables to save sensor data
double rxPitch= 0, rxRoll = 0, rxThrust=0;
double lastPitch= 0, lastRoll = 0;
int truePitch= 0, trueRoll = 0;
```

Mezua jasotakoan gordetzeko bufferra eta sentsoreen irakurketaren edukia zatitutakoan gordetzeko bufferrak:

```
//Main fuffer
char buf[16];

//buffers to save pitch, roll and thrust in 4 bytes
unsigned char pitch4bytes[4];
unsigned char roll4bytes[4];
```

Jasotako mezuen sinkronizazioaren kontrola eramateko zenbait aldagai; jasotako eta bidalitako mezuen kontaketa egiteko, jasotako mezuetatik okerrak direnen kontua eramateko eta resinkoronizazioa egin behar bada, zenbat byte irakurru behar diren jakiteko:

```
//Variables to control the synchro of the communication
int correctFrames =0, framesSent=0, framesReceived=0, incorrectCount = 0; //Counters
int  syncBytes=0; //Number of bytes to read in case resynchronization is needed
```

Aipatu behar da aldagai hauek pantailan erakusten direla erroreak kudeatu ahal izateko, gerora, ez dute ezertarako balioko, pantailarik ez delako egongo errore hauek ikusi ahal izateko.

Aldagai hauek guztiak definitu ondoren begizta infinitu batean sartu behar da ondorengoa, funtzionamenduak konstantea izan behar duelako. Begizta honen barruan egingo da mezuen irakurketa, aginduen argitalpena, sentsoreen irakurketa eta erantzunaren bidalketa, prozesua konstanteki errepikatuz.

Begizta barruan egiten den lehenengo gauza mezua irakurtzen saiatzea da, kasu honetan exekuzioa blokeatuta geratuko da irakurtzeko daturen bat prest egon arte. Irakurketa egiteko, *read_port_frame()* funtzioa erabiltzen da:

```
1  //Begiztaren hasiera
2  while (true)
3  {
4      //Recieve the starting message
5      m_inputPort.read_port_frame(&buf[syncBytes], SLAVE_FRAME_SIZE-syncBytes, showPrints);
6      framesReceived++;
```

Ikusi daitekeenez, mezua jaso eta gero jasotako mezuen kontua eramaten duen kontagailuari bat gehitzen zaio. Horrez gain aipatu behar da, *read_port_frame()* funtzioari ez zaizkiola parametroak ohiko erara pasatzen, izan ere, byte kopuru desberdin bat irakurri beharko ditu aurreko mezua era nahasi batean iritsi baldin bada.

Mezua nahasita iristen bada, honek esan nahi du komunikazioa desinkronizatuta dagoela, hau da, mezuaren hasiera ez dagoela egon beharlo lukeen tokian eta aurreko mezu baten bukaera eta hurrengoaren hasieraren zati bat irakurri dela. Arazo hau konpontzeko, *resynchronize()* funtzioa dugu. Funtzio honek mezuaren hasiera bilatuko du eta hasiera hau kontuan izanda, zuzen irakurritako byten kopurua itzuliko du. Zuzen irakurritako byte kopuru hau *syncBytes* aldagaian gordetzen da eta aldagai honen arabera byte gehiago edo gutxiago irakurtzen dira. Modu honetan nahastutako mezu bat bi zatitan irakurtzea lortzen da eta azkenik komunikazioaren sinkronizazioa berreskuratzen da.

Azaldutakoa lortzeko, mezua baliozkoa den ala ez egiaztatu behar da, horretarako *isValid()* funtzioa dugu, honek mezuaren **start**, **end** eta **checksum** byteak egiaztatzen ditu mezua baliozkoa den ala ez jakiteko. Mezua baliozkoa bada *syncBytes* aldagaia 0-ra jarri behar da, (komunikazioak sinkronizatuta daudelako) eta aginduen argitalpena egin daiteke.

Horrez gain eragingailuen armatzea ere egin behar da eginda egon ezean. Armatzea kontrolatzeko, orain arte erabili ez den **ID** bytea erabili da. Erabilera sinplea da; bytea osatzen dute 8 bitetatik lehenengoa aktibatuta badago, hau da, 1 balioa badu, orduan armatzea egiten da, bestela, desarmatu egiten da.

Mezua baliozkoa ez bada, hurrengo mezu bat irakurriko da eta mezu bat baino gehiago iristen badira nahasita, komunikazioa desinkronizatuta dagoela esan nahi du. Kasu honetan *resynchronize()* funtzioa erabili behar da mezuaren erdi bat irakurtzeko eta hurrengo

irakurketara arte itxaron mezuaren bukaera lortzeko eta aginduak argitaratzeko:

```
//If the message id valid, publish the data in actuators
if(IsValid(buf)){
        syncBytes=0;
        //publish received command for the actuator
        correctFrames++;
        PX4_INFO("CORRECT MESSAGE");

        //Take buffer values and convert them to angles
        bytesToAngles(rxPitch, rxRoll, rxThrust, buf);
        act.x = rxRoll/M_PI;    // roll
        act.y = rxPitch/M_PI;   // pitch
        act.z = rxThrust;  // thrust
        act.r = 0.0;              // yaw

        //Arm or disarm depending on ID byte
        if((buf[1] & (1<<1)) == 1)
            arm.armed=true;
        else
            arm.armed=false;

        orb_publish(ORB_ID(actuator_armed), act_pub_armed, &arm);
        orb_publish(ORB_ID(manual_control_setpoint), act_pub, &act);
}else{
        //If lot of frames incorrect, then try to resynchronize
        incorrectCount++;
        if(incorrectCount > 1){
            syncBytes = m_inputPort.resynchronize(buf);
            incorrectCount=0;
        }
}
```

Mezuaren irakurketa eta aginduen argitalpena eginda, bigarren zatia sentsoreen irakurketa eta erantzunaren bidalketa litzateke. Hau lortzeko aurretik harpidetutako *vehicle_attitude* topikoa erabili da. Topiko honek dronearen posizioa ematen du koaternioietan, hortaz balio horretatik erauzi behar dira **pitch** eta **roll** balioak. Hau egiteko *pitchFromQuaternion()* eta *rollFromQuaternion()* funtzioak ditugu. Gero balio horiek 4 bytetan zatitu behar dira erantzuna bidali ahal izateko, hau lortzeko *convertProcessValue()* funtzioa dugu $[0, 2^{31}]$ tarteko balioa lortzeko eta *split4bytes()* balio hori lau zatitan banatzeko:

```
int poll_ret = px4_poll(&sensor_fd, 1, 100); //we wait up to 0.1 secs
if(poll_ret < 0)
{
    // this is seriously bad - should be an emergency
    if (error_counter < 10 || error_counter % 50 == 0)
    {
        PX4_ERR("ERROR return value from poll(): %d", poll_ret);
    }
    error_counter++;
}else if (poll_ret>0 && sensor_fd.revents & POLLIN){
    //there is sensor data available: get it and dispatch it to the previous slave or master
    struct vehicle_attitude_s raw_sensor;
    orb_copy(ORB_ID(vehicle_attitude), sensor_sub_fd, &raw_sensor);

    //Get angle value
    lastPitch = pitchFromQuaternion(raw_sensor.q);
    lastRoll = rollFromQuaternion(raw_sensor.q);

    //Convert value
    truePitch = convertProcessValue(lastPitch);
    trueRoll = convertProcessValue(lastRoll);

    //Separate in 4 bytes
    split4bytes(truePitch, pitch4bytes);
    split4bytes(trueRoll, roll4bytes);

}
```

Hau eginda, jada mezua prestatu daiteke erantzuna bidaltzeko *send_message_buf()* funtzioaren bitartez. Mezua bidalitakoan *framesSent* kontagailua eguneratu behar da bidalitako mezuen kontua eramateko. Begizta bukatzeko *usleep()* funtzioa erabiltzen da 10 mikrosegundoko geldialdi txiki bat egiteko (CODISAVA_SLAVE_SLEEP_USECS aldagaiaren balioa), komunikazioak ez asetzeko helburuarekin:

```
    buf[0] = 2;
    buf[1] = 0;
    buf[2] = pitch4bytes[0];
    buf[3] = pitch4bytes[1];
    buf[4] = pitch4bytes[2];
    buf[5] = pitch4bytes[3];
    buf[6] = roll4bytes[0];
    buf[7] = roll4bytes[1];
    buf[8] = roll4bytes[2];
    buf[9] = roll4bytes[3];
    buf[10] = 0;
    buf[11] = 0;
    buf[12] = 0;
    buf[13] = 0;
```

```
15        buf[14] = CalculateChecksumB(buf);
16        buf[15] = 3;
17
18        //Send the frame to the master
19        bytesWritten = m_outputPort.send_message_buf(buf, SLAVE_FRAME_SIZE);
20        framesSent++;
21        usleep(CODISAVA_SLAVE_SLEEP_USECS);
22    }
23    //Begiztaren bukaera
```

5.3 Raspberry inplementazioa (bideratzailea)

Pixhawk-en aplikazioarekin batera bideratzailearen inplementazioa ere egin behar da. Txostenean zehar azaldu bezala bideratzeilea Raspberry bat izango da, zehazki **Raspberry Pi 3 B+ modeloa**.

Pixhawk-en aplikazioarekin gertatzen den ez bezala, kasu honetan fitxategi bakar batean egin da inplementazio osoa, garapena errazteko asmoarekin. Sekzio honetan zehar, inplementazioa gauzatzeko aurrekariak eta honen xehetasunak azalduko dira.

5.3.1 Aurrekariak

Inplementazioa gauzatzeko, lehenik eta behin Raspberry-a prestatu behar da. Hardware hau erabiltzeko honen web orri ofizialetik[2] lortu behar da sistema eragilea, **Raspbian Stretch** izenekoa, zeina **Linux**-en oinarrituta dagoen. Kasu honetan erabili den sistema *Raspbian Stretch with desktop and recommended software* izena du. Sitema eragile honek interfaze grafiko bat du lana errazteko eta programa anitz ditu instalatuta hasieratik.

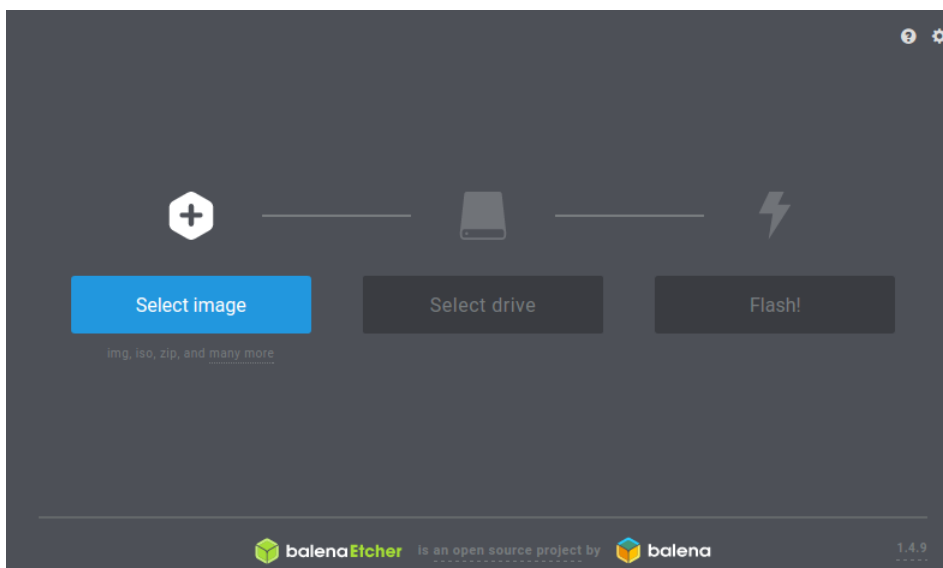
Sistema eragilea Raspberry-an erabili ahal izateko **microSD** memoria txartel batean instalatu behar da (5.10 irudia). Instalazio hau egin ahal izateko programa berezi bat erabili behar da, **balenaEtcher** izenekoa (tankera bereko beste software batzuk ere badaude). Software hau web-orri ofizialean lortu daiteke (5.11 irudia)[3].

[2] https://www.raspberrypi.org/downloads/raspbian/
[3] https://www.balena.io/etcher/

5.10 Irudia: microSD memoria txartel bat.

5.11 Irudia: balenaEtcher softwarearen logoa.

Instalazioa egiteko **Rasbpbian** sistema eragilea eta **balenaEtcher** programak deskargatu behar dira. Gero, balenaEtcher programa ireki behar da, orduan leiho bat irekiko da eta bertan *.zip* formatuan deskargatu den **Raspbian** sistema eragilea aukeratu behar da (5.12 irudia).

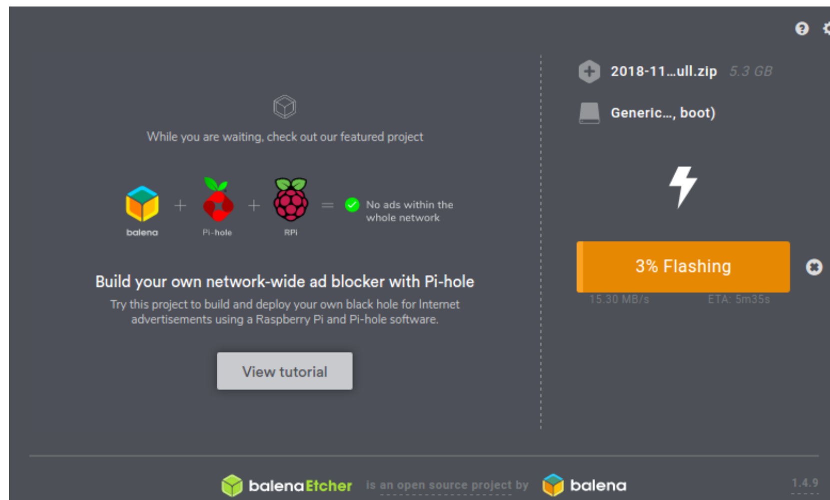


5.12 Irudia: balenaEtcher programa.

Instalatuko den softwarearen aukeraketa egin ondoren, instalatzeko hardware bat aukeratu behar da. Kasu honetan microSD batean instalatu nahi denez **USB** bat erabili behar

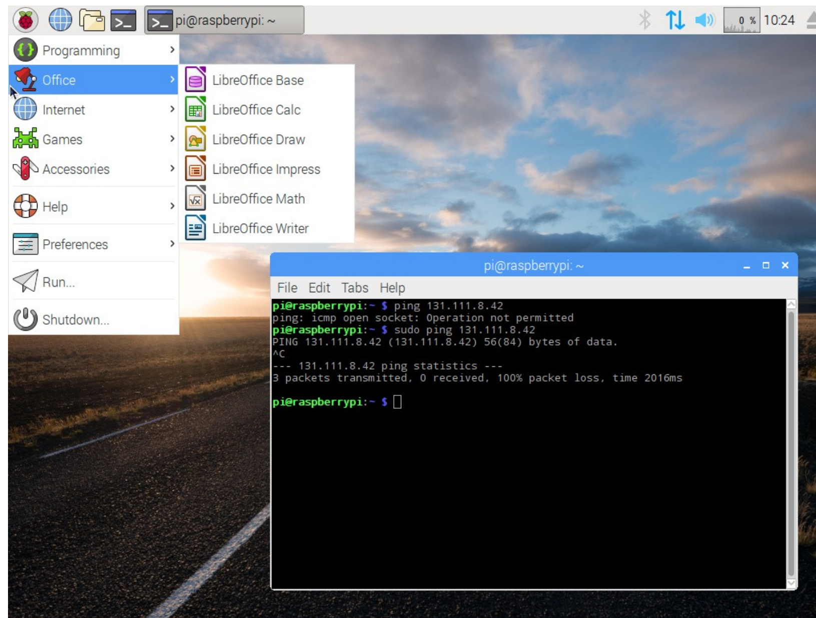
da ordenagailura konektatzeko. Behin aukeraketak egin direla, ”*flash!*” botoia sakatu eta instalazioa hasiko da (5.13 eta 5.14 irudiak).

5.13 Irudia: balenaEtcher programa aukeraketak eginda.



5.14 Irudia: balenaEtcher programa instalazioa egiten ari den bitartean.

Instalazioa bukatutakoan, memoria txartela Rasberry-an sartu eta **HDMI** kable baten bitartez konektatu daiteke monitore batera, modu honetan ordenagilu bat bezala erabili ahal izateko, honek inplementazioa asko errazten du. 5.15 irudian ikusten da raspbian mahigainaren adibidea.



5.15 Irudia: Raspbian mahaigaina

Inplementazioa egiteko programazio lengoaia askoren artean aukeratu daiteke **Raspberry** plataformak jasaten dituenen artean (C, C++, Java, Python...), inplementazioaren lengoaiak ez du zertan izan behar Pixhawk-etan erabili denaren berdina. Hala ere, aukeratu den lengoaia **C++** izan da, Pixhawk-ak programatzen lortutako esperientzia aprobetxatzeko.

Aipatu bezala, fitxategi bakar bat erabili da inplementazioan zehar, honi *pixhawk_communication.cpp* izena eman zaio. Honen konpilazio eta erabilera, edozein linux sistema batean egiten den bezala egiten da; terminal bat ireki, fitxategia *gcc* komandoa erabiliz konpilatu eta sortzen den exekutagarria exekutatu. Hala ere, lortu nahi den helbururako, exekuzio metodo honek soilik ez du balio, izan ere exekuzioa hardware guztia piztu bezain laster hastea nahi da.

Helburu hau lortzeko makina metodo daude Raspberry sistema batean, kasu honetan Pixhawk-etan egiten denaren antzeko zerbait egin da (4.2.3 sekzioa), fitxatgi bateko edukia aldatuz abiaraztean nahi diren programak exekutatu daitezen. Fitxategiaren izena *rc.local* da, eta *etc* izeneko karpetaren barruan dago, honakoa da helbidea */etc/rc.local*. Fitxategia aldatzeko terminal bat ireki eta *nano* sistema deia erabili daiteke, *sudo* ere erabili beharko dugu aldaketak gorde ahal izateko. Fitxategia irekitakoan *sudo nano /etc/rc.local* agindua

erabiliz, honakoa izango da bere edukia:

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
fi


exit 0
```

Nahi den programa exekutatzeko, ”*exit 0*” komandoa baino lehen ezarri behar da agindua *&* sinbolo batekin jarraituta. Hau egiteko exekutagarrairen helbide osoa idatzi behar da, demagun mahaigainan dagoela eta exekutagarriaren izena *ph_communication* dela:

```
sudo /home/pi/Desktop/./ph_communication &
exit 0
```

Aldaketa hauek gordetakoan programa Raspberry-a piztu bezain laster exekutatuko da, honen arazoa gertatzen dena ezin dugula ikusi da, ez baitago pantaila edo terminalik programak izan ditzakeen pantailaratzeak ikusteko. Hala ere arazo honi irtenbide bat emateko modu bat dago, bertatik ateratzen diren pantailaratze guztiak testu fitxategi batean gordetzen:

```
sudo /home/pi/Desktop/./ph_communication > /home/pi/Desktop/log.txt &
exit 0
```

Agindu honi esker *log.txt* fitxategian gordeko da programaren exekuzioan zehar gertatzen den guztia eta hau begiratu ahal izango da pantaila bat eskuragarri dagoenean arazoren bat badago.

5.3.2 Serieko portuak

Serieko portuen erabilera Pixhawk aldeko inplementazioko 5.2.2 sekzioan azaldu da sakonki.

Raspberry-rako, serieko portuen konfigurazio eta irekitzearekin zerikusia duten lau funtzio egin dira; *open_port()*, *config_port()*, *open_port_master()* eta *config_port_master()*. Izenak adierazten duten legez, funtzio hauetako bi, **UART**-era bihurtutako USB portuekin erabiltzeko dira, hauetara **Pixhawk4**-ak edo esklaboak konektatzen dira. Beste biak masterrarekin erabiltzeko dira (**Pixhawk2**), zeina Raspberry-aren UART portura konektatzen den zuzenean.

Portuak irekitzeko diren funtzioen arteko desberdintasuna irekiera moduan dago; alde batetik ez da nahi exekuzioa blokeatzea esklaboen erantzuna jasotzeko zain, batek kale eginez gero, gutxienez besteak funtzionatzen jarraitzea nahi delako. Beste aldetik exekuzioa blokeatu nahi da masterraren erantzunaren zain dagoenean, modu honetan masterrak jartzen du zein den exekuzioaren erritmoa, gainera masterrak kale egiten badu sistema guztia erortzen da nahiz eta itxaroten ez geratu. Bien inplementazioa A.15 eranskinean dago.

Bi funtzioek ireki nahi den portuaren helbidea jasotzen dute eta errore kudeaketa txiki bat egiten dute. Portua irekitzea lortzen bada honen deskriptorea itzultzen dute.

Konfigurazio funtzioetan aurretik azaldu ez den funtzio bat erabiltzen da, *tcflush()* izenekoa, honek ezarritako deskriptorearen bufferra husten du. **TCIOFLUSH** aukerarekin sarrera nahiz irteera bufferrak husten ditu, horrela exekuzioa hastean aurreko komunikazio baten hondakinak egotea saihesten da[4].

tcflush() funtzio hau da arazoak sortzen dituena eta biak desberdinak izatearen ondorioa, izan ere, masterraren funtzioaren kasuan, hau da, Raspberry-aren ohiko UART portua erabiltzen duena, *sleep()* bat sartu behar da hustea gauzatu aurretik, bestela hustea ez da era egokian egiten. Inplementazioak A.16 eranskinean begiratu.

[4] Pixhawk-etan ez da beharrezkoa, pizte bakoitzean automatikoki husten baitira bufferrak.

5.3.3 Mezuen jasotzea

Mezuak jasotzeko metodoa Pixhawk-etan egiten den berdina da, erabiltzen diren funtzioak barne. Desberdintasun bakarra mezuen tamaina da, oraingoan masterrak bidaltzen dituen mezuak **start**, **stop**, **ID**, **esklabo guztien aginduak** eta **end** byteak ditu (ikusi 5.9 irudia). Honek guztira 52 byteko mezu bat osatzen du.

5.2.3 sekzioan azaldu den *read_port_frame()* funtzioak arazorik gabe irakurri ditzake behar diren 52 byteak, aipatutako sekzio horretan azaldu dira sakonki mezuak jasotzeko prozedurak.

Mezuen zuzentasuna egiaztatzeko funtzioak ere berdinak dira, *isValid()* eta *calculateChecksum()* funtzioekin. Oraingoan ere desberdintasun bakarra, mezuaren tamaina litzateke, baina arazorik gabe funtzionatzen dute tamaina berriarekin.

Esklaboen mezu guztiak jasotakoan, mezu horiek masterrari bidaltzeko mezu batean bateratu behar dira, hauen erantzuna jaso dezan. Bateraketa egiteko 52 byteko mezu bat osatu behar da, esklabo guztien aginduak eta **start**, **ID** (denek berdina eramango dute), **checksum** eta **end** byteak biltzen dituena, hasieran jasotzen den mezuaren formatu berdina. Ataza hau burutzeko *setFrameToMaster()* funtzioa egin da, honek bost buffer jasotzen ditu parametro bezala, hauetako lau esklaboen mezuak dira, bestea masterrari bidaltzeko mezua eramango duen bufferra. A.17 eranskinean ikusi daiteke funtzioaren inplementazioa.

5.3.4 Mezuen bidaltzea

Mezuen bidaltzea egiteko funtzio nagusiena *write_port()* da, Pxhawk inplementazioko 5.2.4 sekzioan azaldu den berdina. Oraingoan 4 mezu desberdin bidali beharko dira, bat esklabo bakoitzarentzako, hau lortzeko masterretik jasotzen den mezua zatitu behar da.

Zatiketa hori egiteko funtzio bat sortu da, void *setFramesToSlaves()* izenekoa, honek bost buffer jasotzen ditu parametro bezala, lau buffer esklaboentzat eta masterraren mezua duen bufferra. Funtzioak masterraren mezua duen buferra beste lau buffer txikiagoetan banatzen du, bakoitzari bere aginduak ezarriz. Honekin batera mezu guztiak **start**, **ID** eta

end byteekin eta bakoiztaren **checksum** bytea kalkulatuz osatu behar dira. A.18 eranskinean ikusi daiteke funtzioaren inplementazioa.

5.3.5 Inplementazio orokorra

Sekzio honetan main programaren inplementazioa pausoz pauso azalduko da, inplementazio osoa A.20 eranskinean ikusi daiteke.

Programaren inplementazioa argitzeko lehenengo aurretik definitutako konstante orokor batzuk azaldu behar dira, hauek dira aldagaiak:

```
#define STX 2
#define ETX 3
#define MASTER_FRAME_SIZE 52
#define SLAVE_FRAME_SIZE 16
#define SLAVE_TIMEOUT_COUNT 10
#define SLAVE_TIMEOUT_TIME 200
```

- STX eta EXT: Aurretik hainbatetan azaldutako **start** eta **end** byte-en balioak
- MASTER_FRAME_SIZE: Masterretik jasoko diren eta masterrera bidaliko diren mezuen tamaina adierazten du, 52 byteko tamaina dute.
- SLAVE_FRAME_SIZE: Esklaboetara bidaltzen diren mezuen tamaina adierazten du, 16 byteko tamaina dute.
- SLAVE_TIMEOUT_COUNT eta SLAVE_TIMEOUT_TIME: Bien artean esklabo bakoitzaren erantzuna itxaroten geratuko den denbora adierazten dute, jarraian azalduko da sakonago aldagai hauen eginkizuna.

Main programaren hasieran aldagai batzuk ere definitu dira. Hasteko, irakurritako eta bidalitako byte kopuruak gordetzeko aldagaiak (bidalketa edo irakurketa bat dagoen bakoitzean) eta bidalitako zein jasotako mezuen kopurua eramateko kontagailuak:

```
//Input-output counter variables
int bytesWritten, bytesRead;
int framesSent1 =0, framesSent2 =0, framesSent3 =0, framesSent4 =0;
int framesReceived1 =0, framesReceived2 =0, framesReceived3 =0, framesReceived4 =0;
```

Komunikazioen sinkronizazioa berreskuratzeko balioko duen aldagai bat:

```
//Variable for resynchronization
syncBytes = 0;
```

Portuak ireki eta lortutako fitxategi deskriptoreak gordetzeko aldagaiak, portu horien konfigurazioa ere egin behar da:

```
//Open the ports
int fdSlave1 = open_port("/dev/ttyUSB0");
int fdSlave2 = open_port("/dev/ttyUSB1");
int fdSlave3 = open_port("/dev/ttyUSB2");
int fdSlave4 = open_port("/dev/ttyUSB3");
int fdMaster = open_port_master("/dev/ttyS0");

//Port configuration
config_port(fdSlave1);
config_port(fdSlave2);
config_port(fdSlave3);
config_port(fdSlave4);
config_port_master(fdMaster);
```

Esklabo eta masterren mezuak godetzeko bufferrak eta hauen hasieratzeak, hauek ez hasieratzea ekar ditzakeen arazoak ekiditeko:

```
//Create the buffers
char bufMaster[MASTER_FRAME_SIZE];
char bufSlave1[SLAVE_FRAME_SIZE];
char bufSlave2[SLAVE_FRAME_SIZE];
char bufSlave3[SLAVE_FRAME_SIZE];
char bufSlave4[SLAVE_FRAME_SIZE];

//Init Buffers
for(int i=0; i< MASTER_FRAME_SIZE; i++)  bufMaster[i] = 0;
for(int i=0; i< SLAVE_FRAME_SIZE; i++){
    bufSlave1[i] = 0;
    bufSlave2[i] = 0;
    bufSlave3[i] = 0;
    bufSlave4[i] = 0;
}
```

Esklaboen irakurketa denboraren itxaroteak (timeout) kontrolatzeko eta **ID** byte-a aldatzeko aldagaiak:

```
//Read counter for timeout and ID to warn the master
int timeoutCount;
int timeoutID;
```

timeoutID aldagaia masterraren mezutik iristen den ID byte-aren ”kopia” bat gordetzeko erabili da. Exekuzioan zehar aldagai hau aldatzen joaten da, ID bytea erantzun ez duten esklaboak zintzuk izan diren jakinarazteko erabili baita. Sekzio honetan zehar zehaztuko da aldagai honen funtzionamendua.

Definizio guztiak egin eta gero portu guztiak era egokian ireki direla egiaztatu behar da, baten bat ez bada ondo ireki exekuzioa eten behar da, denak era egokian ireki badira ordea, begizta infinitu batean sartuko da exekuzioa komunikazioak abiarazteko.

```
if (fdSlave1 == -1 || fdSlave2 == -1 || fdSlave3 == -1 || fdSlave4 == -1 || fdMaster == -1)
{
    close(fdSlave1);
    close(fdSlave2);
    close(fdSlave3);
    close(fdSlave4);
    close(fdMaster);
    return(-1);
}else{

    //Exekuzioaren begizta hasi

}
```

Komunikazioen begizta hasteko, masterrak bidali duen mezua irakurri behar da, hau irakurritakoan mezuaren zuzentasuna egiaztatu behar da*isValid()* funtzioarekin. Honen funtzionamendua Pixhawk inplementazioan erabili denaren berdina da, 5.2.5 sekzioan azaldu da xehetasun gehiagorekin.

Mezua baliozkoa bada, *setFramesToSlaves()* funtzioa erabiliko da esklaboei bidaliko zazkien mezuak prestatzeko bakoitzaren aginduekin. Baliozkoa ez bada mezua nahasita datorrela ulertuko da (ondorioz komunikazioak desinkronizatu direla) eta *resynchronize()* funtzioa erabiliko da mezuaren zati bat irakurtzeko. Hurrengo iterazioan mezuaren bigarren zatia irakurriko da eta komunikazioak resinkronizatuko dira. Resinkronizazio prozesu hau era berean burutu da esklaboen inplementazioan 5.2.7 sekzioan azaldu den bezala.

```
while(1){  //Begiztaren hasiera

if(isValid(bufMaster, MASTER_FRAME_SIZE)){
    syncBytes=0;
    setFramesToSlaves(bufMaster, bufSlave1, bufSlave2, bufSlave3, bufSlave4);
}else{
    incorrectCount++;
    if(incorrectCount >= 1){
    printf("Desynchronization occurred\n");
    syncBytes = resynchronize(bufMaster, MASTER_FRAME_SIZE);
    incorrectCount=0;
    }
}
```

Masterraren mezua jasotakoan, esklabo bakoitzari berari dagokion mezua bidali behar zaio. Aipatu behar da bidalketa hau masterraren mezua baliozkoa ez izan arren ere egiten dela, egin ezean esklaboen exekuzioak blokeatuta geratuko lirateke mezu berri bat jasotzen duten arte eta ez lukete inongo erantzunik bidaliko, hau konpontzeko mezuak beti bidaltzen zaizkie. Komunikazioak desinkronizatuta badaude normalean azkenekoz jasotako mezu zuzenak zituen aginduak bidaltzen dira denbora guztian (resinkronizatzea lortu arte), modu honetan droneak posizioa mantenduko luke komunikazioak egonkortzea lortzen den arte eta masterrak esklaboen posizioak jasotzen jarraitzen du.

Hortaz, masterretik jasotako mezuaren aginduak esklaboei bidaltzen zaizkie eta bat gehitzen zaie bidalitako mezuen kontagailuei:

```
//Send the frame to the slaves
bytesWritten = write_port(fdSlave1, bufSlave1, SLAVE_FRAME_SIZE);//slave1
framesSent1++;
bytesWritten = write_port(fdSlave2, bufSlave2, SLAVE_FRAME_SIZE);//slave2
framesSent2++;
bytesWritten = write_port(fdSlave3, bufSlave3, SLAVE_FRAME_SIZE);//slave3
framesSent3++;
bytesWritten = write_port(fdSlave4, bufSlave4, SLAVE_FRAME_SIZE);//slave4
framesSent4++;
```

Mezuak esklaboei bidali bezain laster hauen erantzunak jasotzea espero da. Erantzunak jasotzeko, aurretik aipatu bezala, itxarote denbora edo **timeout** bat ezarri da [5]. *read_port_frame()* funtzioaren barruan erabiltzen den *read()* sistema deiak ez du eskaintzen hau egiteko inongo aukerarik eta beraz, timeout ”faltsu” bat inplementatu da. Hau egiteko *usleep()* sistema

[5] Gogorarazi behar da esklaboen fitxategi deskriptoreak irakurtzerakoan exekuzioa ez dela blokeatzen.

deia, kontagailu bat eta begizta bat konbinatu dira, hau da, iterazio (**while**) bat egiten da *read_port_frame()* funtzioari deituz kontagailu batek definitzen duen buelta kopuru jakin bat eta buelta bakoitzean *usleep()* bat egiten da denbora labur batez exekuzioa geldiaraziz. Sistema honekin itxarote denbora bat kalkulatu daiteke eta baita denbora horren barruan egiten diren irakurketa saiakera kopurua ere.

Begizta honetan erabiltzen dira aurretik definitutako *timeoutCount* aldagaia eta SLAVE_TIMEOUT_COUNT konstantea. *timeoutCount* irakurketa guztiak egin aurretik berrabiarazi behar da 0-ra ezarriz, SLAVE_TIMEOUT_COUNT konstanteak definituko du ematen diren buelta kopurua eta SLAVE_TIMEOUT_TIME konstanteak begiztako buelta bakoitzean itxaroten den denbora. Hortaz begiztak SLAVE_TIMEOUT_COUNT aldagaian definitutako buelta kopurua edo erantzun bat jaso arte iraungo du.

Aldagaientzako hainbat konfigurazio probatu ondoren, egokiena 200 mikrosegundoko 10 buelta ematea izan da, beste era batera esanda, gehienez 10 irakurketa egitea 200 mikrosegundoko itxarotearekin. Honek esan nahi du gehienez 2 milisegundoko itxaroteak egongo direla esklaboren batek erantzuten ez badu.

Hortaz, *timeoutCount* aldagaia berrabiarazi eta gero, irakurketa saiakera bat egiten da. Lehenengo irakurketa saiakera hau ondo ateratzen bada, jasotako mezu kopuruari bat gehitzen zaio, ez bada ezer irakurri begiztan sartzen da eta ezarritako irakurketa kopurua egiten da iterazio bakoitzean ezarritako denbora itxaroten. Begiztan zehar irakurtzea lortzen bada, jasotako mezu kopuruari bat gehituko zaio. Irakurketak egitea lortu ezean, timeout bat egon dela kontsideratuko da eta timeoutID aldagaia aldatuko da masterrari arazoaren berri emateko.

Gogorarazi behar da *timeutID* aldagaia masterretik jastako mezuak daraman **ID** byte-aren berdina dela. Masterrari jakinarazteko, aldagaiaren lau bit-etik azkenekora arteko bit-ak aldatuko dira, hauetako bat aktibatuta iristen bazaio masterrari esan nahi du esklaboetako batek ez duela erantzunik eman.

Jarraian dago azaldutakoaren inplementazioa, prozesua lau aldiz errepikatu behar da, bat esklabo bakoitzarentzat eta prozesu bakoitzean esklabo bakoitzaren aldagaiak erabiliz. Mezu batzuk ere pantailaratzen dira errore kudeaketarekin laguntzeko:

```
1                            ////Slave1////
2   timeoutCount = 0;
3   bytesRead = read_port_frame(fdSlave1, bufSlave1, SLAVE_FRAME_SIZE); //Try to read
4   if(bytesRead < 0){
5       printf("Could not read port, triying to read...\n");
6       while(bytesRead < 0 && timeoutCount < SLAVE_TIMEOUT_COUNT){ //Try to read X times
7           bytesRead = read_port_frame(fdSlave1, bufSlave1, SLAVE_FRAME_SIZE);
8           timeoutCount++;
9           usleep(SLAVE_TIMEOUT_TIME);
10          }
11      if(bytesRead < 0){
12      printf("Timeout occurred in Slave 1\n");
13          timeoutID |= (1u << 4); //Change ID byte to inform the master
14      }
15      else
16          framesReceived1++;
17  }else
18      framesReceived1++;
19
20
21                           ////Slave2////
22  timeoutCount = 0;
23  bytesRead = read_port_frame(fdSlave2, bufSlave2, SLAVE_FRAME_SIZE); //Try to read
24  if(bytesRead < 0){
25      printf("Could not read port, triying to read...\n");
26      while(bytesRead < 0 && timeoutCount < SLAVE_TIMEOUT_COUNT){ //Try to read X times
27          bytesRead = read_port_frame(fdSlave2, bufSlave2, SLAVE_FRAME_SIZE);
28          timeoutCount++;
29          usleep(SLAVE_TIMEOUT_TIME);
30          }
31      if(bytesRead < 0){
32      printf("Timeout occurred in Slave 2\n");
33          timeoutID |= (1u << 5); //Change ID byte to inform the master
34      }
35      else
36          framesReceived2++;
37  }else
38      framesReceived2++;
39
40
41                          ////Slave3////
42  timeoutCount = 0;
43  bytesRead = read_port_frame(fdSlave3, bufSlave3, SLAVE_FRAME_SIZE); //Try to read
44  if(bytesRead < 0){
45      printf("Could not read port, triying to read...\n");
46      while(bytesRead < 0 && timeoutCount < SLAVE_TIMEOUT_COUNT){ //Try to read X times
47          bytesRead = read_port_frame(fdSlave3, bufSlave3, SLAVE_FRAME_SIZE);
48          timeoutCount++;
49          usleep(SLAVE_TIMEOUT_TIME);
50          }
51      if(bytesRead < 0){
```

```
52      printf("Timeout occurred in Slave 3\n");
53          timeoutID |= (1u << 6); //Change ID byte to inform the master
54      }
55      else
56          framesReceived3++;
57  }else
58      framesReceived3++;
59
60
61                          ////Slave4////
62  timeoutCount = 0;
63  bytesRead = read_port_frame(fdSlave4, bufSlave4, SLAVE_FRAME_SIZE); //Try to read
64  if(bytesRead < 0){
65      printf("Could not read port, triying to read...\n");
66      while(bytesRead < 0 && timeoutCount < SLAVE_TIMEOUT_COUNT){ //Try to read X times
67          bytesRead = read_port_frame(fdSlave4, bufSlave4, SLAVE_FRAME_SIZE);
68          timeoutCount++;
69          usleep(SLAVE_TIMEOUT_TIME);
70          }
71      if(bytesRead < 0){
72      printf("Timeout occurred in Slave 4\n");
73          timeoutID |= (1u << 7); //Change ID byte to inform the master
74      }
75      else
76          framesReceived4++;
77  }else
78      framesReceived4++;
```

Behin erantzun guztiak jaso direla, masterraren mezua osatzen da *setFrameToMaster()* funtzioarekin, mezua bidali eta jasotako zein bidalitako mezuen kopuruak pantailaratzen dira. Programaren pantailaratzerik gabeko bertsio bat ere egin da, honek duen konputazio kosturik gabeko bertsio bat edukitzeko asmoarekin. Bukaeran 20 milisegundoko *usleep()* bat egiten da, esklaboen inplementazioekin gertatzen den bezala, komunikazioak ez asetzeko asmoarekin. Hona inplementazioa:

```
1   //Re-construct the frame to send the feedback received from the slaves to the master
2   setFrameToMaster(bufMaster, bufSlave1, bufSlave2, bufSlave3, bufSlave4, timeoutID);
3   bytesWritten = write_port(fdMaster, bufMaster, MASTER_FRAME_SIZE);
4
5   //Print the results
6   printf("Frames SENT: slave1-> %d, slave2-> %d, slave3-> %d, slave4-> %d\n", framesSent1,
        framesSent2, framesSent3, framesSent4);
7   printf("Frames RECEIVED: slave1-> %d, slave2-> %d, slave3-> %d, slave4-> %d\n", framesReceived1,
        framesReceived2, framesReceived3, framesReceived4);
8
9   //Sleep
10  usleep(20000);
11
```

```
12
13 }//Begiztaren bukaera
```

Azkenik, inplementazioa bukatzeko, begiztaren ondoren irekitako fitxategi deskriptoreak ixten dira:

```
1 close(fdSlave1);
2 close(fdSlave2);
3 close(fdSlave3);
4 close(fdSlave4);
5 close(fdMaster);
6 return 0;
```

5.4 Komunikazio probak

Garapenenan zehar komunikazio proba anitz egin dira, hauen eraginkortasuna frogatzeko asmoarekin. Proba hauek egiteko garapenean zehar azaldu diren mezuen pantailaratzeak erabili dira, bai bideratzailean bai esklaboetan. Honez gain komunikazioen sendotasuna egiaztatzeko eta hauen abiadura optimizatzeko probak ere egin dira.

Sekzio honetan zehar proba hauen xehetasunak eta emaitzak azalduko dira.

5.4.1 Monitorizazioa

Proben emaitzak monitorizatu ahal izateko hainbat pantaila erabili behar izan dira, guztira 5 monitore beharko lirateke arkitektura osatzen duten elementu guztiak monitorizatzeko. Hainbeste monitore eskuragarri ez zeudenez guztira hiru pantaila erabili dira, bi Pixhawk eta Raspberry-a monitorizatu ahal izateko.

Pixhawk-etan monitorizatzen dena jasotako mezuen edukia, jasotako mezu zuzenen kopurua, jasotako mezu kopurua guztira eta bidalitako mezu kopurua dira. Garapenean zehar bere sentsoreen balioak ere erakutsi dira errore kudeaketa egin ahal izateko.

Raspberry-an monitorizatzen dena masterretik jasotako mezuen edukia, esklaboen *timeout*-en abisuak, esklaboei bidaltzen zaizkien mezuen edukia eta bidaltzen diren nahiz

jaso diren mezuen kopurua dira.

Pantailaratzerik gabeko programen bertsioak ere erabili dira konputazio kostua murrizteko. Hirugarren bertsio bat ere egon da, hauetan aldagai boolear bat erabili da, monitorizazioa begizten iterazio kopuru jakin batean soilik egiteko (adibidez pantilaratze bat 1000 bueltako), modu honetan monitorizatzea eta exekuzioa ez moteltzea lortzen da era berean.

5.4.2 Abiadura probak

Komunikazioen hasierako probetan, hardware elementuak, batez ere Pixhawk-ak, blokeatu egiten ziren, abiadura handiegiarekin egiten zirelako irakurtze eta bidaltzeak. Arazo honi irtenbidea emateko, komunikazioen abiadura doitu behar da eta abiadura hau **optimizatu**.

Aipatutako doikuntza hauek egiteko sartu dira *usleep()* funtzioak begizten bukaeretan. Funtzio honek exekuzioa blokeatzen du ezarritako mikrosegundo kopuruan zehar. Honekin lortzen dena, bidaltze maiztasun bat ezartzea da, adibidez 100000 balioa ezartzen bada honek esan nahi du mezuak 100 milisegundoko maiztasunarekin bidaltzen direla.

Probak balio askorekin egin dira eta azkenean softeareak inongo arazorik gabe funtzionatzeko lortu den maiztasun maximoa 20 milisegundokoa izan da, hau da, 20000 balioa. Aipatu behar da masterrak ezartzen duela azken batean bidaltze abiadura, besteak blokeatuta geratzen direlako honek zerbait bidaltzen duen arte.

Honek esan nahi du komunikazioen abiadura **optimoa** minutoko 300 mezu direla gutxi gorabehera. Abiadura honekin nahikoa da komunikazioak eraginkorrak izateko.

5.4.3 Sendotasun probak

Komunikazioen sendotasuna probatzeko, softwarea egunetan zehar utzi da exekutatzen, guztira lau egunetan zehar funtzionatzea lortu da, desinkronizazio edo blokeorik gabe eta mezu guztiak zuzenak izanda. Honek esan nahi du bai programa eta bai hardwarea sendoak direla eta ordu askotan zehar funtzionatu dezaketela kale egin gabe.

6. KAPITULUA

Ondorioak

Kapitulu honetan, proiektua garatu ondoren ateratako ondorioak azalduko dira, bai lortutako emaitzen ondorioak, bai pentsatutako ondorio pertsonalak ere. Gainera, proiektuarekin zerikusia duen etorkizunerako lana ere aurkeztuko da.

6.1 Priektuaren ondorioak

Proiektuari dagokionez, lortu diren emaitzak esperotakoak izan dira. Komunikazio software sendo bat garatzea lortu da eta abiadura egoki batean. Hortaz, esan daiteke ezarritako helburuak lortu direla proiektuan zehar.

Garatutako softwarea mahai gainean monitoreekin eta monitore gabe probatu da, drone batekin ere probatu da, sentsoreen argitalpena ongi funtzionatzen duen ikusteko baina oraindik ez da hegaldirik egin komunikazio software hau erabiliz.

6.2 Etorkizuneko lana

Etorkizunean komunikazio software hau **TEA**-koek garatu duten dronean erabiltzeko egokitu daiteke. Honek seguru aski arazoak emango ditu hasiera batean eta beraz doikuntzak egin beharko dira modu egokian funtzionatzeko.

Horrez gain beste aplikazio batzuk ere izan ditzake, aplikazio horietarako egokitzen bada.

6.3 Ondorio pertsonalak

Niri dagokidanez, proiektu honen garapenenean zehar asko ikasi dut ezagutzen ez nituen hardware berriei eta droneei buruz, gainera berrikutntza handia suposatzen duen proiektu baten garapena barrutik ikusteko aukera izan dut, mota honetako lan talde baten funtzionamendua ikusiz.

Proiektuarekin zerikusia zuten gauza guztiak nire kabuz erabiltzen ikasteak eta garapenean zehar sortu diren arazo guztiak nire kabuz konpontzea ere ikasketa handia izan da, modu honetan karreran zehar lortutako ezagutzak eta trebetasunak probatu izan ahal ditut.

Orokorrean proiektu hau **Tecnalia** enpresan garatzea oso esperientzia aberasgarria izan dela ondorioztatu dezaket.

Bibliografia

[1] Dogan Ibrahim. *PIC Microcontroller Projects in C (Second Edition)*. 2014.

[2] W. Bolton. *Programmable Logic Controllers (Sixth Edition)*. 2015.

[3] Michael Barr. *Barr Group's CRC Code-C*. 1999.
`https://barrgroup.com/Embedded-Systems/How-To/Additive-Checksums`

[4] C. Gordon Bell, J. Craig Mudge, John E. McNamara. *Computer Engineering: A DEC View of Hardware Systems Design*. Digital Press, 2014.
`http://bitsavers.org/pdf/dec/_Books/Bell-ComputerEngineering.pdf`

[5] Banks, Michael A. *BITS, BAUD RATE, AND BPS Taking the Mystery Out of Modem Speeds*. Brady Books/Simon & Schuster, 1990.

[6] *The Motion Imagery Standards Board (MISB) Engineering Guideline*. Motion Imagery Standards Board (MISB), 2014.
`https://gwg.nga.mil/misb//docs/standards/ST0601.8.pdf`

[7] NASA: What is Thrust?,
`https://www.grc.nasa.gov/WWW/k-12/airplane/thrust1.html`

[8] Pixhawk: What is Pixhawk?,
`https://pixhawk.org`

[9] QGroundControl: Intuitive and Powerful Ground Control Station for the MAVLink protocol,
`http://qgroundcontrol.com/`

[10] MAVLink: MAVLink Developer Guide,
`https://mavlink.io/en/`

[11] PX4: What is PX4?,
https://px4.io/

[12] PX4: uORB Messaging,
https://dev.px4.io/en/middleware/uorb.html

A. ERANSKINA

Funtzioen inplementazioak

A.1 _setup_port()

```
1  bool SerialPort::_setup_port(int baud, int data_bits, int stop_bits, bool parity, bool
        hardware_control)
2  {
3      // Check file descriptor
4      if(!isatty(fd))
5      {
6          fprintf(stderr, "\nERROR: file descriptor %d is NOT a serial port\n", fd);
7          return false;
8      }
9
10     // Read file descritor configuration
11     struct termios config;
12
13     if(tcgetattr(fd, &config) < 0)
14     {
15             printf("Error: could not retrieve port configuration (descriptor %d)\n", fd);
16             return false;
17         }
18
19     // Apply baudrate
20     switch (baud)
21     {
22
23     case 115200:
24         if (cfsetispeed(&config, B115200) < 0 || cfsetospeed(&config, B115200) < 0)
25         {
26             fprintf(stderr, "\nERROR: Could not set desired baud rate of %d Baud\n", baud);
```

```
27            return false;
28        }
29        break;
30
31        /*HEMEN BESTE AUKERA GUZTIAK EGONO LIRATEKE, ERABILI DEN AUKERA BAKARRIK
32        JARRIKO DA TXUKUNAGO GERA DADIN*/
33
34    }
35    // Finally, apply the configuration
36    if(tcsetattr(fd, TCSANOW, &config) < 0)
37    {
38        fprintf(stderr, "\nERROR: could not set configuration of fd %d\n", fd);
39                perror("tcsetattr() error");
40
41        return false;
42    }
43        printf("Input speed = %d // Output speed = %d\n", (int) cfgetispeed(&config), (int)
     cfgetospeed(&config));
44        printf("Input flag = %d // Output flag = %d\n", (int) config.c_iflag, (int) config.
     c_oflag);
45        printf("Control mode = %d // Local mode = %d\n", (int) config.c_cflag, (int) config.
     c_lflag);
46
47
48    // Done!
49    return true;
50 }
```

A.2 _open_port()

```
1  int SerialPort::_open_port()
2  {
3      // Open serial port
4      // O_NOCTTY - Ignore special chars like CTRL-C
5      if (!m_useNonBlockingIO)
6          fd = open(m_uart_name, O_RDWR | O_NOCTTY );
7      else
8          fd = open(m_uart_name, O_RDWR | O_NOCTTY | O_NONBLOCK | O_NDELAY);
9
10     // Check for Errors
11     if (fd == -1)
12     {
13         fprintf(stderr, "\nERROR: could not open serial port\n");
14         return(-1);
15     }
16     if (!m_useNonBlockingIO)
17        fcntl(fd, F_SETFL, 0); //it was originally set as a blocking file descriptor
```

```
18
19      return fd;
20  }
```

A.3 open_serial()

```
1   void SerialPort::open_serial()
2   {
3       printf("OPEN PORT: %s\n", m_uart_name);
4
5       fd = _open_port();
6
7       if (fd == -1)
8       {
9           printf("failure, could not open port.\n");
10          return;
11      }
12
13      bool success = _setup_port(m_baudRate, 8, 1, false, false);
14
15      if (!success)
16      {
17          printf("Failure, could not configure port.\n");
18          return;
19      }
20      if (fd <= 0)
21      {
22          printf("Connection attempt to port %s with %d baud, 8N1 failed, exiting.\n", m_uart_name,
          m_baudRate);
23          return;
24      }
25
26      printf("Connected to %s with %d baud, 8 data bits, no parity, 1 stop bit (8N1)\n",
         m_uart_name, m_baudRate);
27
28      status = true;
29
30      printf("\n");
31  }
```

A.4 start()

```
void SerialPort::start( const char* portName, int baudRate, bool useNonBlockingIO, bool debug)
{
    m_uart_name= portName;
    m_baudRate = baudRate;

    m_numBytesRead= 0;
    m_magicNumberRead = false;
    m_useNonBlockingIO= useNonBlockingIO;
    m_debug= debug;

    open_serial();
}
```

A.5 close_serial()

```
void SerialPort::close_serial()
{
    printf("CLOSE PORT\n");

    int result = close(fd);

    if ( result )
    {
        fprintf(stderr,"WARNING: Error on port close (%i)\n", result );
    }

    status = false;

    printf("\n");
}
```

A.6 read_port_frame()

```
void SerialPort::read_port_frame(char *buf, int len, bool showPrints)
{
    int bytesReadTotal=0;
    int bytesRead=0;

    while(bytesReadTotal != len){
        bytesRead = read(fd, &buf[bytesReadTotal], (unsigned)len);
        bytesReadTotal += bytesRead;

```

```
10      }
11
12      int num =0;
13      if (bytesRead == -1)
14
15      {
16              if (showPrints) fprintf(stderr,"\nERROR: could not read port\n");
17
18      }else{
19
20          if (showPrints){
21          printf("\nMessage read correctly, bytes read in total: %d\n", bytesReadTotal);
22            for(int i=0; i<bytesReadTotal; i++){
23                  if((int8_t)buf[i] < 0)
24                      num = (int8_t)buf[i] + 256;
25                  else
26                      num = (int8_t)buf[i];
27                      printf("I(%d): %d\n",i,num);
28                    }
29              }
30          }
31  }
```

A.7 send_message_buf()

```
1
2   int SerialPort::send_message_buf(char* buf, int len)
3   {
4       // Write buffer to serial port, locks port while writing
5       const int bytesWritten = write(fd, (char*) buf, (unsigned)len);
6
7       return bytesWritten;
8   }
```

A.8 CalculateChecksum()

```
unsigned char CalculateChecksum(char* buf)
{
    unsigned char checksum = 0;

    for (int byte= 1; byte<=13; byte++)
    {
        checksum = checksum xor buf[byte];
    }

    return checksum;
}
```

A.9 IsValid()

```
bool SlaveController::IsValid(char* buf)
{
    //ETX eta STX egiaztatu
    if (buf[0] != STX || buf[15] != ETX)
        {
            PX4_INFO("STX and ETX are incorrect");
            return false;
        }
    //Checksum byte-a kalkulatu eta egiaztatu
    if (CalculateChecksum(buf) == buf[14])
        {
        PX4_INFO("CORRECT MESSAGE");
        return true;
        }
        PX4_INFO("INVALID CHECKSUM");
    return false;
}
```

A.10 bytesToAngles()

```
void bytesToAngles(double pitch, double roll, double thrust, char* buf)
{
        unsigned char pitch1 = buf[2];
        unsigned char pitch2 = buf[3];
        unsigned char pitch3 = buf[4];
        unsigned char pitch4 = buf[5];

        unsigned char roll1 = buf[6];
        unsigned char roll2 = buf[7];
        unsigned char roll3 = buf[8];
        unsigned char roll4 = buf[9];

        unsigned char thrust1 = buf[10];
        unsigned char thrust2 = buf[11];
        unsigned char thrust3 = buf[12];
        unsigned char thrust4 = buf[13];

        unsigned long txPitch =pitch4 + pitch3*256 + pitch2*65536 + pitch1*16777216;
        unsigned long txRoll = roll4 + roll3*256 + roll2*65536 + roll1*16777216;
        unsigned long txThrust = thrust4 + thrust3*256 + thrust2*65536 + thrust1*16777216;

            pitch = (txPitch*M_PI)/pow(2,30) - M_PI;
            roll = (txRoll*M_PI)/pow(2,30) - M_PI;
            thrust = (txThrust)/pow(2,31);
}
```

A.11 convertProcessValue()

```
int convertProcessValue(double eu_angle)
{
    double angle = ((eu_angle + M_PI) * pow(2,31))/(2*M_PI);
    return (int)angle;
}
```

A.12 split4bytes()

```
1  void split4bytes(int angle, unsigned char* angle4bytes)
2  {
3       int rest1, rest2, rest3;
4       int byte1, byte2, byte3, byte4;
5       //first byte
6       rest1=angle%16777216;
7       byte1=angle/16777216;
8
9       //second byte
10      rest2=rest1%65536;
11      byte2=rest1/65536;
12
13      //third byte
14      rest3=rest2%256;
15      byte3=rest2/256;
16
17      //fourth byte
18      byte4=rest3;
19
20      //Put the bytes in the array
21      angle4bytes[0] = (unsigned char)byte1;
22      angle4bytes[1] = (unsigned char)byte2;
23      angle4bytes[2] = (unsigned char)byte3;
24      angle4bytes[3] = (unsigned char)byte4;
25 }
```

A.13 Koaternoien funtzioak

```
1  //Pitch ateratzeko
2  double pitchFromQuaternion(float* quat)
3  {
4      // pitch (y-axis rotation)
5      double test = quat[0]*quat[1] + quat[2]*quat[3];
6      if (test > 0.499) // singularity at north pole
7          return 2 * atan2((double)quat[0], (double)quat[3]);
8
9      if (test < -0.499) // singularity at south pole
10         return -2 * atan2((double)quat[0], (double)quat[3]);
11
12     double sqy = quat[1]*quat[1];
13     double sqz = quat[2]*quat[2];
14     return atan2((double) (2 * quat[1]*quat[3] - 2 * quat[0]*quat[2]), (double) (1 - 2 * sqy - 2
        * sqz));
```

```
15  }
16
17
18  //Roll ateratzeko
19  double rollFromQuaternion(float* quat)
20  {
21      // roll (z-axis rotation)
22      double test = quat[0]*quat[1] + quat[2]*quat[3];
23      if (test > 0.499) // singularity at north pole
24          return M_PI * 0.5;
25
26      if (test < -0.499) // singularity at south pole
27          return -M_PI * 0.5;
28
29      return asin((double) (2 * test));
30  }
31
32  //Yaw ateratzeko
33  double yawFromQuaternion(float* quat)
34  {
35      // yaw (x-axis rotation)
36      double test = quat[0]*quat[1] + quat[2]*quat[3];
37      if (test > 0.499) // singularity at north pole
38          return 0.0;
39
40      if (test < -0.499) // singularity at south pole
41          return 0.0;
42
43      double sqx = quat[0]*quat[0];
44      double sqz = quat[2]*quat[2];
45
46      return atan2((double) (2 * quat[0]*quat[3] - 2 * quat[1]*quat[2]), (double)( 1 - 2 * sqx - 2
         * sqz));
47  }
```

A.14 resynchronize()

```
int SerialPort::resynchronize(char *buf)
{
       int bytesCorrect=0, i=0;
       bool stxFound=false;
        //Search STX in the incorrect buffer
        while(!stxFound){
                if (buf[i] == STX){
                      buf[0]=buf[i];
                        bytesCorrect++;
                        stxFound = true;
                }
                i++;
        }
        //Save the correct part of the buffer
        while(i<16){
                buf[bytesCorrect] = buf[i];
                bytesCorrect++;
                i++;
        }
  return bytesCorrect;
}
```

A.15 open_port() eta open_port_master()

```
int open_port(char *port)
{
    int fd;
    // Open serial port
    fd = open(port , O_RDWR | O_NOCTTY | O_NONBLOCK | O_NDELAY);
    // Check for Errors
    if (fd == -1)
    {
        printf("\nERROR: could not open '%s' serial port\n", port);
        return(-1);
    }else
        printf("\n'%s' port correctly opened\n", port);
    return fd;
}

int open_port_master(char *port)
{
    int fd;
    // Open serial port
```

```
20      fd = open(port , O_RDWR);
21      // Check for Errors
22      if (fd == -1)
23      {
24          printf("\nERROR: could not open '%s' serial port\n", port);
25          return(-1);
26      }else
27          printf("\n'%s' port correctly opened\n", port);
28      return fd;
29  }
```

A.16 config_port() eta config_port_master()

```
1   void config_port(int uart0_filestream)
2   {
3       struct termios options;
4           tcgetattr(uart0_filestream, &options);
5           cfsetispeed(&config, B115200);
6           cfsetospeed(&config, B115200);      //<Set baud rate
7           options.c_iflag = IGNPAR;
8           options.c_oflag = 0;
9           options.c_lflag = 0;
10          tcflush(uart0_filestream, TCIOFLUSH);
11          tcsetattr(uart0_filestream, TCSANOW, &options);
12  }
13
14
15  void config_port_master(int uart0_filestream)
16  {
17      struct termios options;
18          tcgetattr(uart0_filestream, &options);
19          cfsetispeed(&config, B115200);
20          cfsetospeed(&config, B115200);      //<Set baud rate
21          options.c_iflag = IGNPAR;
22          options.c_oflag = 0;
23          options.c_lflag = 0;
24          sleep(1);                         //Sleep needed to flush correctly
25          tcflush(uart0_filestream, TCIOFLUSH);
26          tcsetattr(uart0_filestream, TCSANOW, &options);
27  }
```

A.17 setFrameToMaster()

```
void setFrameToMaster(char* bufMaster, char* bufSlave1, char* bufSlave2, char* bufSlave3, char*
     bufSlave4, int timeoutID)
{
    //STX and ID
    bufMaster[0] = 2;
    bufMaster[1] = timeoutID;

    ////////////////////////////PAYLOAD///////////////////////////////
    /////Slave 1 payload/////
    //pitch
    bufMaster[2] = bufSlave1[2] ;
    bufMaster[3] = bufSlave1[3] ;
    bufMaster[4] = bufSlave1[4] ;
    bufMaster[5] = bufSlave1[5] ;
    //roll
    bufMaster[6] = bufSlave1[6] ;
    bufMaster[7] = bufSlave1[7] ;
    bufMaster[8] = bufSlave1[8] ;
    bufMaster[9] = bufSlave1[9] ;
    //thrust
    bufMaster[10] = bufSlave1[10] ;
    bufMaster[11] = bufSlave1[11] ;
    bufMaster[12] = bufSlave1[12] ;
    bufMaster[13] = bufSlave1[13] ;
    ////////////////////////
    ////////////////////////
    ////Slave 2 payload//////
    //pitch
    bufMaster[14] = bufSlave2[2] ;
    bufMaster[15] = bufSlave2[3] ;
    bufMaster[16] = bufSlave2[4] ;
    bufMaster[17] = bufSlave2[5] ;
    //roll
    bufMaster[18] = bufSlave2[6] ;
    bufMaster[19] = bufSlave2[7] ;
    bufMaster[20] = bufSlave2[8] ;
    bufMaster[21] = bufSlave2[9] ;
    //thrust
    bufMaster[22] = bufSlave2[10] ;
    bufMaster[23] = bufSlave2[11] ;
    bufMaster[24] = bufSlave2[12] ;
    bufMaster[25] = bufSlave2[13] ;
    ////////////////////////
    ////////////////////////
    ////Slave 3 payload//////
    //pitch
    bufMaster[26] = bufSlave3[2];
    bufMaster[27] = bufSlave3[3];
    bufMaster[28] = bufSlave3[4];
    bufMaster[29] = bufSlave3[5];
    //roll
```

```
51      bufMaster[30] = bufSlave3[6];
52      bufMaster[31] = bufSlave3[7];
53      bufMaster[32] = bufSlave3[8];
54      bufMaster[33] = bufSlave3[9];
55      //thrust
56      bufMaster[34] = bufSlave3[10];
57      bufMaster[35] = bufSlave3[11];
58      bufMaster[36] = bufSlave3[12];
59      bufMaster[37] = bufSlave3[13];
60      ////////////////////////
61      ////////////////////////
62      ////Slave 4 payload/////
63      //pitch
64      bufMaster[38] = bufSlave4[2];
65      bufMaster[39] = bufSlave4[3];
66      bufMaster[40] = bufSlave4[4];
67      bufMaster[41] = bufSlave4[5];
68      //roll
69      bufMaster[42] = bufSlave4[6];
70      bufMaster[43] = bufSlave4[7];
71      bufMaster[44] = bufSlave4[8];
72      bufMaster[45] = bufSlave4[9];
73      //thrust
74      bufMaster[46] = bufSlave4[10];
75      bufMaster[47] = bufSlave4[11];
76      bufMaster[48] = bufSlave4[12];
77      bufMaster[49] = bufSlave4[13];
78      /////////////////////////
79      ////////////////////////////////////////////////////////////////////
80
81      //ETX and Checksum
82      bufMaster[50] = CalculateChecksumB(bufMaster, MASTER_FRAME_SIZE-3);
83      bufMaster[51] = 3;
84  }
```

A.18 setFramesToSlaves()

```
void setFramesToSlaves(char* bufMaster, char* bufSlave1, char* bufSlave2, char* bufSlave3, char*
    bufSlave4)
{
    ////Slave 1////
    bufSlave1[0] = 2;
    bufSlave1[1] = bufMaster[1];
    bufSlave1[2] = bufMaster[2];
    bufSlave1[3] = bufMaster[3];
    bufSlave1[4] = bufMaster[4];
    bufSlave1[5] = bufMaster[5];
    bufSlave1[6] = bufMaster[6];
    bufSlave1[7] = bufMaster[7];
    bufSlave1[8] = bufMaster[8];
    bufSlave1[9] = bufMaster[9];
    bufSlave1[10] = bufMaster[10];
    bufSlave1[11] = bufMaster[11];
    bufSlave1[12] = bufMaster[12];
    bufSlave1[13] = bufMaster[13];
    bufSlave1[14] = CalculateChecksumB(bufSlave1, SLAVE_FRAME_SIZE-3);
    bufSlave1[15] = 3;

    ////Slave 2////
    bufSlave2[0] = 2;
    bufSlave2[1] = bufMaster[1];
    bufSlave2[2] = bufMaster[14];
    bufSlave2[3] = bufMaster[15];
    bufSlave2[4] = bufMaster[16];
    bufSlave2[5] = bufMaster[17];
    bufSlave2[6] = bufMaster[18];
    bufSlave2[7] = bufMaster[19];
    bufSlave2[8] = bufMaster[20];
    bufSlave2[9] = bufMaster[21];
    bufSlave2[10] = bufMaster[22];
    bufSlave2[11] = bufMaster[23];
    bufSlave2[12] = bufMaster[24];
    bufSlave2[13] = bufMaster[25];
    bufSlave2[14] = CalculateChecksumB(bufSlave2, SLAVE_FRAME_SIZE-3);
    bufSlave2[15] = 3;

    ////Slave 3////
    bufSlave3[0] = 2;
    bufSlave3[1] = bufMaster[1];
    bufSlave3[2] = bufMaster[26];
    bufSlave3[3] = bufMaster[27];
    bufSlave3[4] = bufMaster[28];
    bufSlave3[5] = bufMaster[29];
    bufSlave3[6] = bufMaster[30];
    bufSlave3[7] = bufMaster[31];
```

```
48      bufSlave3[8] = bufMaster[32];
49      bufSlave3[9] = bufMaster[33];
50      bufSlave3[10] = bufMaster[34];
51      bufSlave3[11] = bufMaster[35];
52      bufSlave3[12] = bufMaster[36];
53      bufSlave3[13] = bufMaster[37];
54      bufSlave3[14] = CalculateChecksumB(bufSlave3, SLAVE_FRAME_SIZE-3);
55      bufSlave3[15] = 3;
56
57      ////Slave4////
58      bufSlave4[0] = 2;
59      bufSlave4[1] = bufMaster[1];
60      bufSlave4[2] = bufMaster[38];
61      bufSlave4[3] = bufMaster[39];
62      bufSlave4[4] = bufMaster[40];
63      bufSlave4[5] = bufMaster[41];
64      bufSlave4[6] = bufMaster[42];
65      bufSlave4[7] = bufMaster[43];
66      bufSlave4[8] = bufMaster[44];
67      bufSlave4[9] = bufMaster[45];
68      bufSlave4[10] = bufMaster[46];
69      bufSlave4[11] = bufMaster[47];
70      bufSlave4[12] = bufMaster[48];
71      bufSlave4[13] = bufMaster[49];
72      bufSlave4[14] = CalculateChecksumB(bufSlave4, SLAVE_FRAME_SIZE-3);
73      bufSlave4[15] = 3;
74  }
```

A.19 Run() (Esklaboen programa nagusia)

```
1   int SlaveController::Run(int argc, char* argv[])
2   {
3       m_inputPort.start(CODISAVA_INPUT_PORT, CODISAVA_BAUD_RATE, m_useNonBlockingIO,
         m_outputConsoleMsgs);
4       m_outputPort.start(CODISAVA_OUTPUT_PORT, CODISAVA_BAUD_RATE, m_useNonBlockingIO,
         m_outputConsoleMsgs);
5
6
7       struct manual_control_setpoint_s act;
8       memset(&act, 0, sizeof(act));
9       orb_advert_t act_pub = orb_advertise(ORB_ID(manual_control_setpoint), &act);
10
11      struct actuator_armed_s arm;
12      memset(&arm, 0, sizeof(arm));
13      orb_advert_t act_pub_armed = orb_advertise(ORB_ID(actuator_armed), &arm);
14
15
```

```
16      int sensor_sub_fd = orb_subscribe(ORB_ID(vehicle_attitude));  orb_set_interval(sensor_sub_fd,
          CODISAVA_SENSOR_READING_FREQ);
17      px4_pollfd_struct_t sensor_fd;
18      sensor_fd.fd = sensor_sub_fd;
19      sensor_fd.events = POLLIN;
20
21      //Variables to save sensor data
22      double rxPitch= 0, rxRoll = 0, rxThrust=0;
23      double lastPitch= 0, lastRoll = 0;
24      int truePitch= 0, trueRoll = 0;
25
26      //Main fuffer
27      char buf[16];
28
29      //buffers to save pitch, roll and thrust in 4 bytes
30      unsigned char pitch4bytes[4];
31      unsigned char roll4bytes[4];
32
33
34      //Variables to control the synchro of the communication
35      int correctFrames =0, framesSent=0, framesReceived=0, incorrectCount = 0; //Counters
36      int  syncBytes=0; //Number of bytes to read in case resynchronization is needed
37
38      //Begiztaren hasiera
39      while (true)
40      {
41          //Recieve the starting message
42          m_inputPort.read_port_frame(&buf[syncBytes], SLAVE_FRAME_SIZE-syncBytes, showPrints);
43          framesReceived++;
44
45          //If the message id valid, publish the data in actuators
46          if(IsValid(buf)){
47              syncBytes=0;
48              //publish received command for the actuator
49              correctFrames++;
50               PX4_INFO("CORRECT MESSAGE");
51
52              //Take buffer values and convert them to angles
53              bytesToAngles(rxPitch, rxRoll, rxThrust, buf);
54              act.x = rxRoll/M_PI;    // roll
55              act.y = rxPitch/M_PI;   // pitch
56              act.z = rxThrust;  // thrust
57              act.r = 0.0;              // yaw
58
59              //Arm or disarm depending on the ID byte
60              if((buf[1] & (1<<1)) == 1)
61                  arm.armed=true;
62              else
63                  arm.armed=false;
64
65              orb_publish(ORB_ID(actuator_armed), act_pub_armed, &arm);
66              orb_publish(ORB_ID(manual_control_setpoint), act_pub, &act);
```

```
67          }else{
68              //If lot of frames incorrect, then try to resynchronize
69              incorrectCount++;
70              if(incorrectCount > 1){
71                  syncBytes = m_inputPort.resynchronize(buf);
72                  incorrectCount=0;
73              }
74           }
75
76          int poll_ret = px4_poll(&sensor_fd, 1, 100); //we wait up to 0.1 secs
77          if(poll_ret < 0)
78          {
79              // this is seriously bad - should be an emergency
80              if (error_counter < 10 || error_counter % 50 == 0)
81              {
82                  PX4_ERR("ERROR return value from poll(): %d", poll_ret);
83              }
84              error_counter++;
85          }else if (poll_ret>0 && sensor_fd.revents & POLLIN){
86              //there is sensor data available: get it and dispatch it to the previous slave or
        master
87              struct vehicle_attitude_s raw_sensor;
88              orb_copy(ORB_ID(vehicle_attitude), sensor_sub_fd, &raw_sensor);
89
90              //Get angle value
91              lastPitch = pitchFromQuaternion(raw_sensor.q);
92              lastRoll = rollFromQuaternion(raw_sensor.q);
93
94              //Convert value
95              truePitch = convertProcessValue(lastPitch);
96              trueRoll = convertProcessValue(lastRoll);
97
98              //Separate in 4 bytes
99              split4bytes(truePitch, pitch4bytes);
100             split4bytes(trueRoll, roll4bytes);
101
102           }
103
104         buf[0] = 2;
105         buf[1] = 0;
106         buf[2] = pitch4bytes[0];
107         buf[3] = pitch4bytes[1];
108         buf[4] = pitch4bytes[2];
109         buf[5] = pitch4bytes[3];
110         buf[6] = roll4bytes[0];
111         buf[7] = roll4bytes[1];
112         buf[8] = roll4bytes[2];
113         buf[9] = roll4bytes[3];
114         buf[10] = 0;
115         buf[11] = 0;
116         buf[12] = 0;
117         buf[13] = 0;
```

```
118          buf[14] = CalculateChecksumB(buf);
119          buf[15] = 3;
120
121          //Send the frame to the master
122          bytesWritten = m_outputPort.send_message_buf(buf, SLAVE_FRAME_SIZE);
123          framesSent++;
124          usleep(CODISAVA_SLAVE_SLEEP_USECS);
125
126      }
127
128  }
```

A.20 main() (Bideratzailearen programa nagusia)

```
1   int main(int argc, char *argv[])
2   {
3
4       //sleep(3);
5
6       //Input-output counter variables
7       int bytesWritten, bytesRead;
8       int framesSent1 =0, framesSent2 =0, framesReceived1 =0, framesReceived2 =0;
9       int framesSent3 =0, framesSent4 =0, framesReceived3 =0, framesReceived4 =0;
10
11      //variables for resynchro and frame validation
12    syncBytes = 0;
13
14
15      //Open the ports
16      int fdSlave1 = open_port("/dev/ttyUSB0");
17      int fdSlave2 = open_port("/dev/ttyUSB1");
18      int fdSlave3 = open_port("/dev/ttyUSB2");
19      int fdSlave4 = open_port("/dev/ttyUSB3");
20      int fdMaster = open_port_master("/dev/ttyS0");
21
22      //Port configuration
23      config_port(fdSlave1);
24      config_port(fdSlave2);
25      config_port(fdSlave3);
26      config_port(fdSlave4);
27      config_port_master(fdMaster);
28
29      //Create the buffers
30      char bufMaster[MASTER_FRAME_SIZE];
31      char bufSlave1[SLAVE_FRAME_SIZE];
32      char bufSlave2[SLAVE_FRAME_SIZE];
33      char bufSlave3[SLAVE_FRAME_SIZE];
```

```
34      char bufSlave4[SLAVE_FRAME_SIZE];
35
36      //Init Buffers
37      for(int i=0; i< MASTER_FRAME_SIZE; i++)  bufMaster[i] = 0;
38      for(int i=0; i< SLAVE_FRAME_SIZE; i++){
39              bufSlave1[i] = 0;
40              bufSlave2[i] = 0;
41              bufSlave3[i] = 0;
42              bufSlave4[i] = 0;
43      }
44
45      //Read counter for timeout and ID to warn the master
46      int timeoutCount;
47      int timeoutID;
48
49      //Variables to control monitorizing freq
50      int showPrintsCount;
51      int showPrints;
52
53      if (fdSlave1 == -1 || fdSlave2 == -1 || fdSlave3 == -1 || fdSlave4 == -1 || fdMaster == -1)
54
55      {
56          close(fdSlave1);
57          close(fdSlave2);
58          close(fdSlave3);
59          close(fdSlave4);
60          close(fdMaster);
61          return(-1);
62
63      }else{
64
65          //bool desync = false;
66          while(1)
67          {
68
69              showPrintsCount++;
70
71              if(showPrintsCount > 10000){
72                  showPrints = true;
73                  showPrintsCount = 0;
74              }else if(showPrints) showPrints=false;
75
76              //Set ID to 0 at first
77              timeoutID = 0;
78              //if (desync) printf("Desynchronization occurred\n");
79              //Receive the frame from the master and prepare it for the slaves
80              read_port_frame(fdMaster, &bufMaster[syncBytes], MASTER_FRAME_SIZE-syncBytes,
     showPrints); //master
81              timeoutID = bufMaster[1];
82              //Check if the message is valid
83              if(isValid(bufMaster, MASTER_FRAME_SIZE)){
84                  syncBytes=0;
```

```
85                  setFramesToSlaves(bufMaster, bufSlave1, bufSlave2, bufSlave3, bufSlave4);
86              }else{
87                  incorrectCount++;
88                  if(incorrectCount >= 1){
89                          printf("Desynchronization occurred\n");
90                          syncBytes = resynchronize(bufMaster, MASTER_FRAME_SIZE);
91                          incorrectCount=0;
92                  }
93
94              }
95
96                  //Send the frame to the slaves
97                  bytesWritten = write_port(fdSlave1, bufSlave1, SLAVE_FRAME_SIZE);//slave1
98                  framesSent1++;
99                  bytesWritten = write_port(fdSlave2, bufSlave2, SLAVE_FRAME_SIZE);//slave2
100                 framesSent2++;
101                 bytesWritten = write_port(fdSlave3, bufSlave3, SLAVE_FRAME_SIZE);//slave3
102                 framesSent3++;
103                 bytesWritten = write_port(fdSlave4, bufSlave4, SLAVE_FRAME_SIZE);//slave4
104                 framesSent4++;
105                 //Receive the feedback from the slaves
106
107                                         ////Slave1////
108                 timeoutCount = 0;
109                 bytesRead = read_port_frame(fdSlave1, bufSlave1, SLAVE_FRAME_SIZE, showPrints);
      //Try to read
110                 if(bytesRead < 0){
111                     printf("Could not read port, triying to read...\n");
112                     while(bytesRead < 0 && timeoutCount < SLAVE_TIMEOUT_COUNT){ //Try to read X
      times
113                         bytesRead = read_port_frame(fdSlave1, bufSlave1, SLAVE_FRAME_SIZE,
      showPrints);
114                         timeoutCount++;
115                         usleep(SLAVE_TIMEOUT_TIME);
116                         }
117                         if(bytesRead < 0){
118                             printf("Timeout occurred in Slave 1\n");
119                             timeoutID |= (1u << 4); //Change ID byte to inform the master
120                         }
121                         else
122                             framesReceived1++;
123                     }else
124                         framesReceived1++;
125
126                                         ////Slave2////
127                 timeoutCount = 0;
128                 bytesRead = read_port_frame(fdSlave2, bufSlave2, SLAVE_FRAME_SIZE, showPrints);
      //Try to read
129                 if(bytesRead < 0){
130                     printf("Could not read port, triying to read...\n");
131                     while(bytesRead < 0 && timeoutCount < SLAVE_TIMEOUT_COUNT){ //Try to read X
      times
```

```
132                         bytesRead = read_port_frame(fdSlave2, bufSlave2, SLAVE_FRAME_SIZE,
        showPrints);
133                         timeoutCount++;
134                         usleep(SLAVE_TIMEOUT_TIME);
135                         }
136                         if(bytesRead < 0){
137                           printf("Timeout occurred in Slave 2\n");
138                             timeoutID |= (1u << 5); //Change ID byte to inform the master
139                         }
140                         else
141                             framesReceived2++;
142                     }else
143                         framesReceived2++;
144 
145 
146                                         ////Slave3////
147                 timeoutCount = 0;
148                 bytesRead = read_port_frame(fdSlave3, bufSlave3, SLAVE_FRAME_SIZE, showPrints);
        //Try to read
149                 if(bytesRead < 0){
150                 printf("Could not read port, triying to read...\n");
151                     while(bytesRead < 0 && timeoutCount < SLAVE_TIMEOUT_COUNT){ //Try to read X
        times
152                         bytesRead = read_port_frame(fdSlave3, bufSlave3, SLAVE_FRAME_SIZE,
        showPrints);
153                         timeoutCount++;
154                         usleep(SLAVE_TIMEOUT_TIME);
155                         }
156                         if(bytesRead < 0){
157                           printf("Timeout occurred in Slave 3\n");
158                             timeoutID |= (1u << 6); //Change ID byte to inform the master
159                         }
160                         else
161                             framesReceived3++;
162                     }else
163                         framesReceived3++;
164 
165 
166                                       ////Slave4////
167                 timeoutCount = 0;
168                 bytesRead = read_port_frame(fdSlave4, bufSlave4, SLAVE_FRAME_SIZE, showPrints);
        //Try to read
169                 if(bytesRead < 0){
170                     printf("Could not read port, triying to read...\n");
171                     while(bytesRead < 0 && timeoutCount < SLAVE_TIMEOUT_COUNT){ //Try to read X
        times
172                         bytesRead = read_port_frame(fdSlave4, bufSlave4, SLAVE_FRAME_SIZE,
        showPrints);
173                         timeoutCount++;
174                         usleep(SLAVE_TIMEOUT_TIME);
175                         }
176                         if(bytesRead < 0){
```

```
177                         printf("Timeout occurred in Slave 4\n");
178                           timeoutID |= (1u << 7); //Change ID byte to inform the master
179                       }
180                       else
181                           framesReceived4++;
182                   }else
183                       framesReceived4++;
184 
185               //Re-construct the frame to send the feedback received from the slaves to the
      master
186               setFrameToMaster(bufMaster, bufSlave1, bufSlave2, bufSlave3, bufSlave4, timeoutID
      );
187               bytesWritten = write_port(fdMaster, bufMaster, MASTER_FRAME_SIZE);
188 
189               //Print the results of the communication
190               if(showPrints) printf("Frames SENT: slave1-> %d, slave2-> %d, slave3-> %d, slave4
      -> %d\n", framesSent1, framesSent2, framesSent3, framesSent4);
191               if(showPrints) printf("Frames RECEIVED: slave1-> %d, slave2-> %d, slave3-> %d,
      slave4-> %d\n", framesReceived1, framesReceived2, framesReceived3, framesReceived4);
192 
193               //Sleep 20 miliseconds
194               usleep(20000);
195 
196 
197         }
198     }
199 
200 
201     close(fdSlave1);
202     close(fdSlave2);
203     close(fdSlave3);
204     close(fdSlave4);
205     close(fdMaster);
206     return 0;
207 }
```

B. ERANSKINA

Fitxategien edukiak

B.1 codisava.h

```
 
#define CODISAVA_BAUD_RATE 115200

#define CODISAVA_INPUT_PORT "/dev/ttyS2"
#define CODISAVA_OUTPUT_PORT "/dev/ttyS2"

#define CODISAVA_SENSOR_READING_FREQ 10 //time in milliseconds between sensor updates

#define CODISAVA_MASTER_SLEEP_USECS 2000000 //time (micro-seconds) between messages sent by the
    fake master
#define CODISAVA_SLAVE_SLEEP_USECS 10 //micro-seconds to sleep after each iteration

#define STX 2
#define ETX 3

#define SLAVE_FRAME_SIZE 16
#define MASTER_FRAME_SIZE 52
```

B.2 codisava.cpp

```
#include <px4_log.h>
#include <string.h>
#include "serial_port.h"
```

```
4  #include "slave_controller.h"
5  #include "master_controller.h"
6  #include "test_data_sender.h"
7
8
9  extern "C" __EXPORT int codisava_main(int argc, char *argv[]);
10
11 enum ExecutionMode {Slave, Master, SendData, None};
12 //Slave: receive messages from input serial port and dispatch them to either the motor controller
       or the next slave
13 //SendData: send test data through TELEM2 (used as output)
14
15 ExecutionMode ParseExecutionMode(int argc, char* argv[])
16 {
17     for (int i= 0; i<argc; i++)
18     {
19         if (!strcmp(argv[i],"slave"))
20             return ExecutionMode::Slave;
21         if (!strcmp(argv[i],"send-data"))
22             return ExecutionMode::SendData;
23     }
24     return ExecutionMode::None; //execution mode undefined
25 }
26
27
28 int codisava_main(int argc, char *argv[])
29 {
30     SlaveController slaveController;
31     TestDataSender testDataSender;
32
33     ExecutionMode mode= ParseExecutionMode (argc, argv);
34
35     if (mode==ExecutionMode::Slave)
36         slaveController.Run(argc, argv);
37     else if (mode == ExecutionMode::SendData)
38         testDataSender.Run(argc, argv);
39
40     fprintf(stderr, "Error: Undefined execution mode. Usage:\n\n");
41     fprintf(stderr, "    -codisava slave id=[1,4] <no-feedback> <debug> <non-blocking>\n");
42     fprintf(stderr, "    -codisava send-data <no-feedback> <debug> <non-blocking>\n");
43
44     return OK;
45 }
```

B.3 serial_port.cpp

```
#include <cstdlib>
#include <stdio.h>   // Standard input/output definitions
#include <unistd.h>  // UNIX standard function definitions
#include <fcntl.h>   // File control definitions
#include <termios.h> // POSIX terminal control definitions
#include <pthread.h> // This uses POSIX Threads
#include <signal.h>

#include "codisava.h"

// ------------------------------------------------------------------------------
//   Defines
// ------------------------------------------------------------------------------

// The following two non-standard baudrates should have been defined by the system
// If not, just fallback to number
#ifndef B460800
#define B460800 460800
#endif

#ifndef B921600
#define B921600 921600
#endif


// Status flags
#define SERIAL_PORT_OPEN   1;
#define SERIAL_PORT_CLOSED 0;
#define SERIAL_PORT_ERROR -1;


// ----------------------------------------------------------------------------------
//   Serial Port Manager Class
// ----------------------------------------------------------------------------------
/*
 * Serial Port Class
 *
 * This object handles the opening and closing of the offboard computer's
 * serial port over which we'll communicate.  It also has methods to write
 * a byte stream buffer.  MAVlink is not used in this object yet, it's just
 * a serialization interface.  To help with read and write pthreading, it
 * gaurds any port operation with a pthread mutex.
 */
class SerialPort
{

    const char *m_uart_name;
    int  m_baudRate;
    int  status;

    unsigned int m_numBytesRead = 0;
```

```
52          bool m_magicNumberRead = false;
53          bool m_useNonBlockingIO = false;
54      bool m_debug= false;
55
56
57  public:
58
59      SerialPort();
60      ~SerialPort();
61
62
63      int send_message_buf(char* buf, int len);
64      void read_port_frame(char *buf, int len, bool showPrints);
65      int resynchronize(char *buf);
66      void open_serial();
67      void close_serial();
68
69      void start( const char* portName, int baudRate, bool useNonBlockingIO, bool debug );
70
71  private:
72      int  fd;
73
74      int  _open_port();
75      bool _setup_port(int baud, int data_bits, int stop_bits, bool parity, bool hardware_control);
76  };
```

B.4 slave_controller.h

```
1   #include "serial_port.h"
2
3   class SlaveController
4   {
5       SerialPort m_inputPort, m_outputPort;
6       int m_id= 0;
7       bool m_sendFeedback= true;
8       bool m_outputConsoleMsgs= false;
9       bool m_useNonBlockingIO = false;
10
11      void ParseArguments(int argc, char* argv[]);
12      bool IsValid(char* buf);
13  public:
14      SlaveController();
15      ~SlaveController();
16
17      int Run(int argc, char* argv[]);
18  };
```