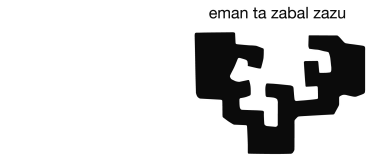


Informatika Ingeniaritzako Gradua

Konputagailuen Ingeniaritza

Gradu Amaierako Proiektua

Bateria baten egoera kontrolatzen duen txartel komertzial batentzako driverra eta interfaze grafikoa diseinatzea eta garatzea

---

Egilea

Jon De Miguel Lauzirika

2014

Laburpena

Eskuartean irakurgai duzu hainbat hilabetetan zehar garatutako proiektua deskribatzen duen memoria. Bateria baten egoera kontrolatzen duen txartel komertzial batentzako driverra eta interfaze grafikoa diseinatzea eta garatzean datza proiektua.

Azken finean, programatu beharrekoa serie portutik abiatuta txartel batekin komunikatzeko aplikazioa izan da, eta jasotako datuak interfaze grafiko baten bitartez bistaragarriak bihurtzea.

Esan daiteke proiektua bi fase nagusitan antolatu dela.

Lehenengoa, USB[1] portutik nahi dugun informazioa jaso eta bidaltzeko aukeran oinarritu da. Lehen fase honetan jasotako informazio hexadezimala prozesatzeaz gain, dagozkion balioak erakustea izango da helburua.

Bigarren fasea, aurretik jasotako balioak bistaratuko dituen interfaze grafikoari dagokiona da; eta interfaze horrek izango dituen botoiak, informazioa noiz jaso nahi den adierazi ahal izateko.

[1] Universial Serial Bus

Gaien aurkibidea

Irudien aurkibidea

Taulen aurkibidea

1. KAPITULUA

Sarrera

RSAIT (Robotika eta Sistema Autonomoen Ikerkuntza Taldea) taldeko laborategian *B21* motako robot handi bat dago, Marisorgin izena duena (ikus 1.1 irudia). Robotaren berrikuntza burutu zenean, bi aukera ikusi ziren: ordenagailu eramangarri bat jartzea edo mahaigaineko bat. Azken hau aukeratu zen, behar zuen potentzia kontuan hartuta.

Gailu eramangarriek bateriak behar izaten dituzte funtzionatzeko. Bateria horiek behar bezala kudeatzeko, hardware eta software bereziz hornituak daude. Hardwaretik pasatzen den korrontea neurtu egiten da, eta neurketa horri esker, softwarea gai izaten da zenbateko kontsumoa dagoen jakiteko, zenbat bateria gastatu den, etab.

Jakina den moduan, mahaiganekoaren arazoa elikadura izaten da, konektatuta egon behar dute. Izan ere, orokorrean, mahaigainekoak *AC*-ra konektatzen dira. Horregatik, ezinezkoa egiten da ordenagailu mota hau erabiltzea mugitzeko aukerak dituen robot baten kasurako.

Hori horrela, Marisorgin bateriekin elikatzea aukeratu zen, baina, baterien kontrola ezin zen nolanahi egin eta, arazoa konpontzeko, EVM[1] txartel bat erosi zen, hain zuzen ere, *BQ78412*-a (ikus 1.2 irudia).

Txartel horri esker, bat-bateko itzaltzeak ekiditeko aukera dago, hain zuzen ere, bateriaren kontsumoaren informazioa eskaintzeko gaitasuna duelako.

Plaka hau, USB bidez konektatzen da ordenagailura eta bateriaren inguruko informazioa

[1] EValuation Module

eskaintzen du. Plaka horren kontrolerako driverra, Windows sistema eragilearentzat sortua dago; baina robotean, Linux erabiltzen da, Ubuntu 12.04, hain zuzen ere.

Bateria kontrolatzeko, ez zen erosoa beste Windows eramangarri bat edo antzeko gailu batekin robotean ibiltzea, eta Linux sistema eragilerako bateria kontrolatzeko driver bat egitea pentsatu zen.

Horren guztiaren ondorioz sortu da proiektu hau.

Driver horrek, bateriaren kontsumoa, betetze maila ehunekotan emanda eta beste datu erabilgarri batzuk jasotzeaz gain, bateriaren kapazitatea eta beste datu garrantzitsu batzuk aldatzeko aukera ematen du.

Bestalde, lan guztia plaka horretan oinarriturik egin denez, komeni da Marisorgin robotaren barneko bateria eta plaka ikustea.

Bateria eta plaka behar bezala ikus daitezke 1.2 irudian.

1.1 Irudia: *Marisorgin robota.*

1.2 Irudia: *Marisorgin robotaren bateria eta horren kontrol-txartela.*

1.1 Txartelaren ezaugarriak

1.2 irudian ikusten den txartelaren ezaugarriak bi multzotan banatu dira.

Alde batetik, *buzzer* bat du txartelak. Buzzer horrek soinua egiteko balio du eta konfiguratu da bateria martxan jartzean txistu egin dezan. Bateriaren karga txikia denean horren berri eman dezan, eta abar.

Bestalde, txartela LED-ez hornitua dago. Kolore desberdinetako eskala bat dago oinarrian eta bateriaren karga adierazten dute kolore horiek. Karga beteta edo ia beteta dagoenean, LED-ak berderaino pizten dira; erdizka dagoenean, horiraino pizten dira; eta karga txikia geratzen denean edo ia kargarik gabe dagoenean, gorrian gelditzen dira.

2. KAPITULUA

Proiektuaren Helburuen Dokumentua

2.1 Aurkezpena

Proiektuaren helburu nagusia, Marisorgin robotean erabiltzeko Texas Instruments enpresak sorturiko [7] plakarentzat *driver* eta aplikazioa eraikitzea da, bateriaren egoera zein den une oro adierazteko.

Proiektu honen premia, Informatika Fakultateko RSAIT taldeko ikerleek Marisorgin robota berritu dutenean sortu da. Izan ere, robot hau oso zaharkitua geratu zen, eta berriz martxan jartzeko, pieza asko aldatu behar izan zaizkio. Besteak beste, bateria berria jarri zaio.

Bateria hori kontrolatzen duen txartelaren datuak jasotzeko aplikazioa Windows sistema eragilerako bakarrik dagoenez, ezin zen robotean erabili.

Texas Instruments enpresak sorturiko plaka hori Marisorgin robotean erabiltzen da, Linux sistema eragileaz baliatzen dena, Ubuntu 12.04 sistema eragilean hain zuzen ere.

Beraz, Linux sistema eragilean funtzionatzen duen driver baten beharra zegoen.

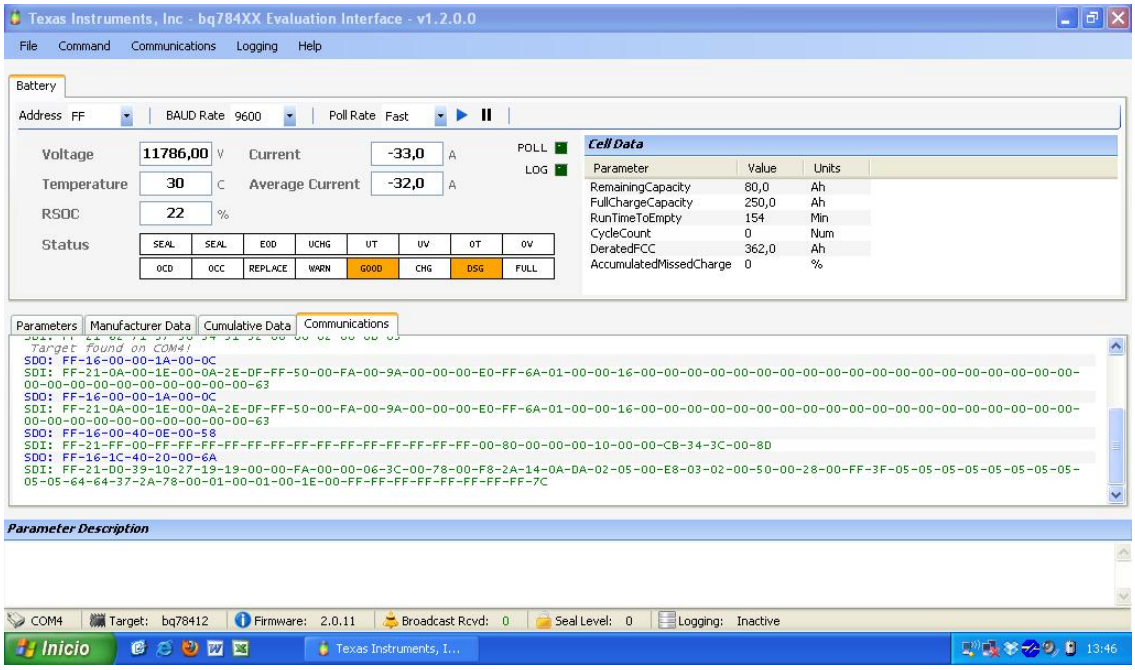
2.2 Proiektuaren helburuak

Proiektuaren helburu nagusia, Marisorgin robotaren erabiltzaileek bateria errazago kontrolatu ahal izatea da, hots, darabilten guztian zenbat bateria geratzen den jakitea une oro.

Ezin ahaztu, bateria plakarekin konektaturik izan gabe kargatuz gero, guztiz kargatu arren, plakara konektatzean honek %50-a kargatu balitz bezala hartzen du kontuan. Hori ekiditeko, bateria horren kapazitatea (RSOC[1] parametroa) %100-ean jarri behar da.

Hortaz gain, aplikazioa gauzatzeko orduan, *Python* lengoaia erabili da, Ubuntu sistema eragileak bere gain ekartzen baitu interpretea. Hasiera batean, *C* lengoaian egiteko asmoa zegoen arren, *Python* lengoaiarekin burutu da. Aldaketa honen arrazoiak aurrerago azalduko dira.

Windows-erako eginik dagoen aplikazioak ondorengo itxura du:



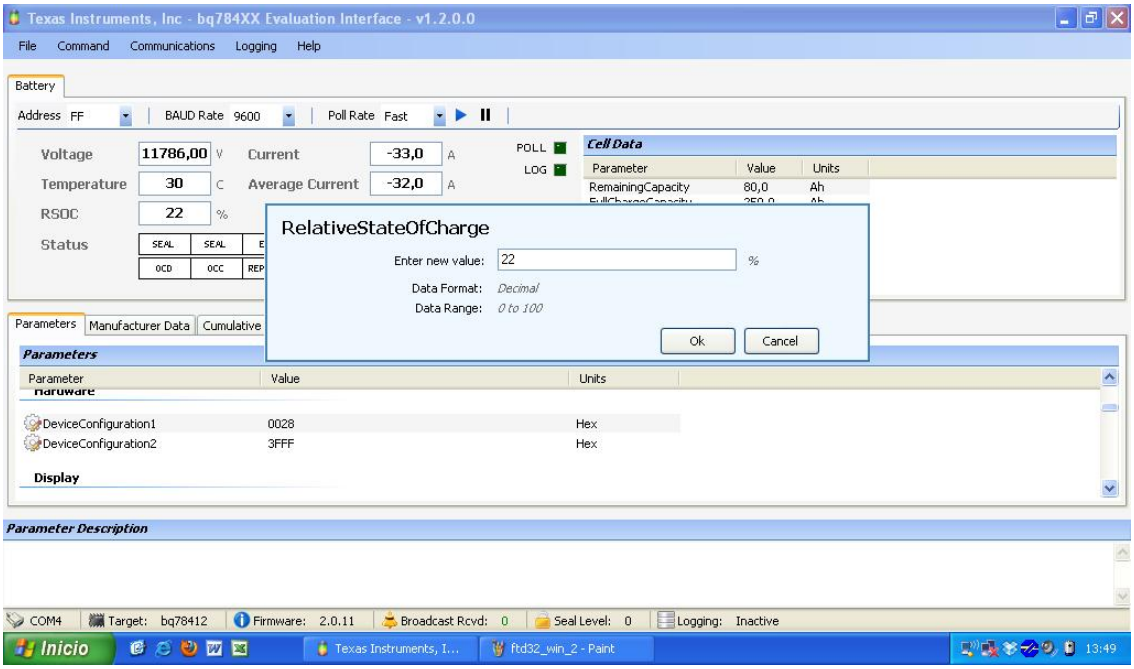
2.1 Irudia: *Windows-eko aplikazioaren leiho nagusia.*

Hemendik, datu garrantzitsuenak *Voltage, Temperature, Current, Average Current* eta *RSOC* direla esan behar da.

Beraz, gure aplikazioaren lehen helburua, balio hauek erakustea izango litzateke.

[1] Relative State of Charge

RSOC-aren kasuan, Windows-eko aplikazioak 2.2 leihoa erakusten du abiarazi nahi denean.



2.2 Irudia: *Windows-eko aplikazioan RSOC-a hasieratzeko leihoa.*

Bertan, ikusten denez, balioa aukeratzea dago. Urrats hori ekiditeko, eta %100-ean jartzeko, sortuko den aplikazioan botoi bat jarriko da zuzenean mezu hori bidaltzeko, eta galdetzen ez ibiltzeko Windows-eko aplikazioan bezala zein baliorekin abiarazi nahi den.

2.3 Azpiatazak

Proiektu hau zati desberdinetan banatu da: informazio bilketa, kudeaketa, garapena, dokumentazioa eta denbora estimazioa. Horietako bakoitzaren barnean, bestalde, azpiataza gehiago daude.

2.3.1 Informazio bilketa

Ataza honetan proiektua garatzeko beharrezkoa izan den informazio bilketa azaltzen da. Lehenik, proiektuaren zuzendariak esango du zer egin eta nondik nora jo helburuak lortzeko.

Jarraian, eskatutakoa betetzeko informazioa jasoko da: driverren programaziorako gidaliburuak, adibideak, eta azkenik, baina ez garrantzi gutxiagokoa delako, *BQ78412* plakaren gidaliburua ([7]).

Gidaliburu honi esker lortuko dugu jakitea zein mezu mota trukatu beharko zaion bateriaren datuak lortzeko, eta mezu bakoitza nola kodetu beharko den.

2.3.2 Kudeaketa

Ataza honetan proiektua garatzeko beharrezkoa izango den planifikazioa aztertuko da.

Bertan proiektuaren helburuak finkatzeaz gain, lan paketeetan banatu eta denboren arabera estimatuko dira. Proiektuaren bideragarritasun eta arrisku plana ere bertan garatuko dira.

2.3.3 Garapena

Hau, xehe-xehe zehaztu beharra dago, izan ere, proiektua hasiera batean C programazio lengoaian egiteko ustea zegoen, ohikoena baita horrelako kasuetan.

Baina Linux sistema eragilea erabiliz, ordenagailuak plakaren driverra ez zuen behar bezala hartzen, beraz, C-rekin behe mailan programatzen hasi beharra zegoen. Horretarako, Glade aplikazioa erabili zen interfazea diseinatzeko, eta gero kodea, VIM bezalako kode editore baten bidez programatu zen.

Sortutako arazoak ekiditeko, beste lengoaia bat ikasteko gogoak, eta gomendioak tartean, *Python* erabiltzea erabaki zen azken bertsioa sortzeko.

Hemen, argi utzi beharra dago, *Python*-ekin hastean, dena hasieratik berrikusi zela, interfazea bai kodea programatuz.

Aldaketa horri esker, denbora asko aurreztu zen, eta ondorioz, hobekuntza desberdinak ezartzeko aukera izan zen aplikazioan zehar, besteak beste, AppIndicator-a eta aukera aurreratuen leihoa.

2.3.4 Dokumentazioa

Ataza honetan, garapenarekin batera, baina batez ere garapena bukatu ostean, entregatu beharreko memoria osatzen joango da. Dokumentu honetan, proiektuaren helburuak zeintzuk diren, garapena nola joan den eta erabiltzaile gida bat agertuko dira.

Bestalde, proiektuaren defentsarako egin beharreko aurkezpena ere bertan sar daiteke.

2.3.5 Denbora estimazioa

Ondorengo taulan azalduko da nola banatu diren atazak eta gutxi gorabehera zenbateko iraupena izan duen ataza bakoitzak (ordutan adierazita) (2.1 taula).

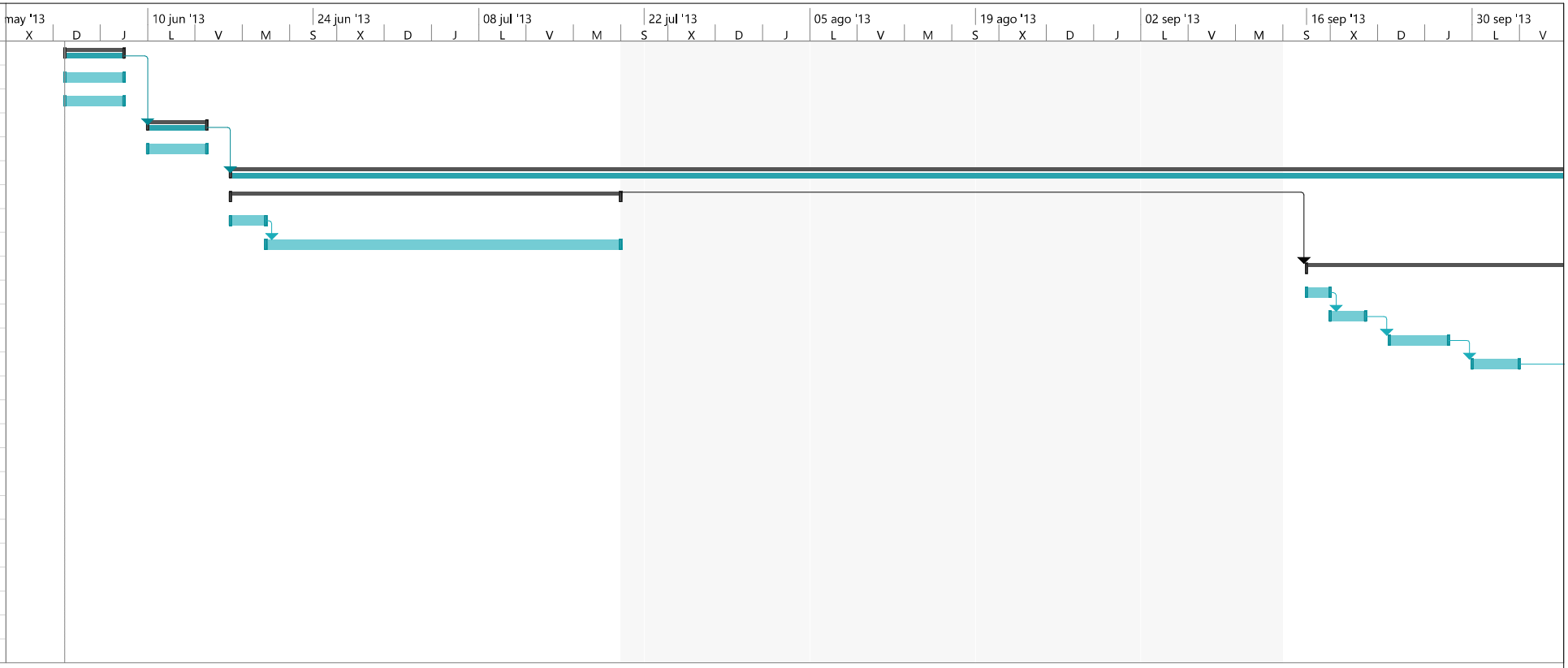
Ataza	Deskribapena	Estimazioa
1. Proiektua ezagutu		**20**
1.1 Informazioa bildu	Zer egin eta nola	10
1.2 Driver-ak nola programatu dokumentatu	Gidaliburuak bilatu eta irakurri	10
2. Plangintza		**10**
2.1 PHD egin	Helburuak definitu	10
3. Garapena		**129**
3.1 C zatia		*50*
3.1.1 Interfaze grafikoa	Glade programarekin	6
3.1.2 Kodea	Programatu	44
3.2 Python zatia		*79*
3.2.1 Klaseen egitura	Diagramak	4
3.2.2 Interfazea programatu	Programatu	6
3.2.3 Driver objektua	Programatu	25
3.2.4 Argumentuen kontrola	Informatu eta programatu	8
3.2.5 Hobekuntzak		*36*
3.2.5.1 Aukera aurreratuen leihoa	Programatu	10
3.2.5.2 Interfazearen aldaketak	Programatu	10
3.2.5.3 AppIndicator		*16*
3.2.5.3.1 Menua	Informatu eta programatu	8
3.2.5.3.2 Bateriaren portzentaia eta ikonoa	Informatu eta programatu	8
4. Dokumentazioa		**92**
4.1 Memoria bukatu	Memoria dokumentua idatzi	92
5. Aurkezpena	Aurkezpena egin defentsarako	**15**
Guztira		***266***

2.1 Taula: Atazak eta denbora estimazioa

GANTT diagrama

Proiektuaren atazak eta ataza bakoitza nondik nora joan den hobeto ikusteko, ondoren *Gantt*-en diagrama bat erabiliko da.

Id	EDT
1	**GAP1**
2	GAP1-A
3	GAP1-B
4	**GAP2**
5	GAP2-A
6	**GAP3**
7	**GAP3-A**
8	GAP3-A.1
9	GAP3-A.2
10	**GAP3-B**
11	GAP3-B.1
12	GAP3-B.2
13	GAP3-B.3
14	GAP3-B.4
15	**GAP3-B.5**
16	GAP3-B.5.1
17	GAP3-B.5.2
18	**GAP3-B.5.3**
19	GAP3-B.5.3.1
20	GAP3-B.5.3.2
21	**GAP4**
22	GAP4-A
23	GAP5
24	GAP6
25	GAP7
26	GAP8



Proyecto: GAP_Gantt
Fecha: jue 23/01/14

Tarea	Resumen del proyecto	Tarea manual	solo el comienzo	Fecha límite
División	Tarea inactiva	solo duración	solo fin	Progreso
Hito	Hito inactivo	Informe de resumen manual	Tareas externas	Progreso manual
Resumen	Resumen inactivo	Resumen manual	Hito externo	

Proyecto: GAP_Gantt
Fecha: jue 23/01/14

Tarea	Resumen del proyecto	Tarea manual	solo el comienzo	Fecha límite
División	Tarea inactiva	solo duración	solo fin	Progreso
Hito	Hito inactivo	Informe de resumen manual	Tareas externas	Progreso manual
Resumen	Resumen inactivo	Resumen manual	Hito externo	

Ohar moduan, esan beharra dago, atzealde grisa duen zatia, oporrak adierazteko intentzioarekin jarri zela, eta ez dela kontuan hartu orduak kalkulatzeko.

EDT-en esanahia

EDT	**Deskribapena**
GAP1	Proiektua ezagutu
GAP1-A	Informazioa bildu
GAP1-B	Driver-ak nola programatu informatu
GAP2	Plangintza
GAP2-A	PHD egin
GAP3	Garapena
GAP3-A	C zatia
GAP3-A.1	Interfaze grafikoa
GAP3-A.2	Kodea
GAP3-B	Python zatia
GAP3-B.1	Klaseen egitura
GAP3-B.2	Interfazea
GAP3-B.3	Driver objektua
GAP3-B.4	Argumentuen kontrola
GAP3-B.5	Hobekuntzak
GAP3-B.5.1	Aukera aurreratuak
GAP3-B.5.2	Interfazearen egokitzapena
GAP3-B.5.3	AppIndicator
GAP3-B.5.3.1	Menua
GAP3-B.5.3.2	Bateriaren ikonoa eta mezuak
GAP4	Dokumentazioa
GAP4-A	Memoria bukatu
GAP5	ADDI-ra igo
GAP6	Aurkezpena
GAP7	Aurkezpena entregatu
GAP8	Proiektuen defentsaren epeak

2.2 Taula: GANTT diagramako EDT-en esanahia.

2.4 Arriskuak

Proiektuan zehar arrisku asko egon litezke, ondo menperatzen ez den lengoaia batean programatu behar denean eta honek arazoak ekar litzazke.

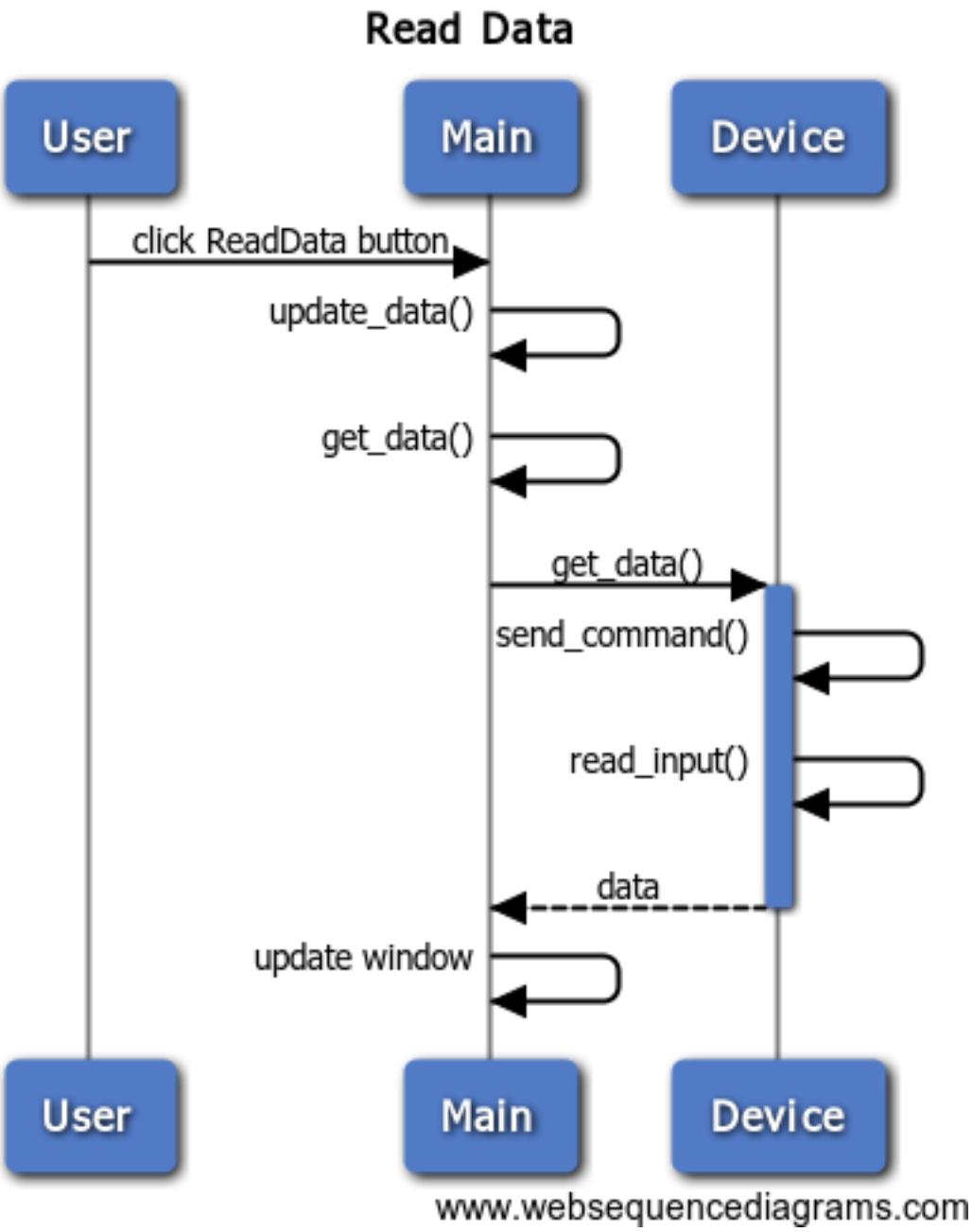
Bat-batean datuak ez galtzeko, eta berriz hutsetik ez hasi behar izateko, eramangarri batean kodearen kopiak egiteaz gain, bertsio kontrolerako *Bitbucket*-eko errepositorio pribatu bat erabiliko da, *git* komandoaren bitartez bertsioak igoz bertara.

Horrela, momentu oro eduki da eskura kodea, eta azken aldaketen aurretik erabilitako bertsioren bat behar izanez gero, bertan lor zitekeen.

Bestalde, memoriaren eta aurkezpenarekin sor zitezkeen arriskuak saihesteko, *Dropbox* eta eramangarri bat erabili dira, bertan eta disko gogorrean gordez beti egindako aldaketen kopiak.

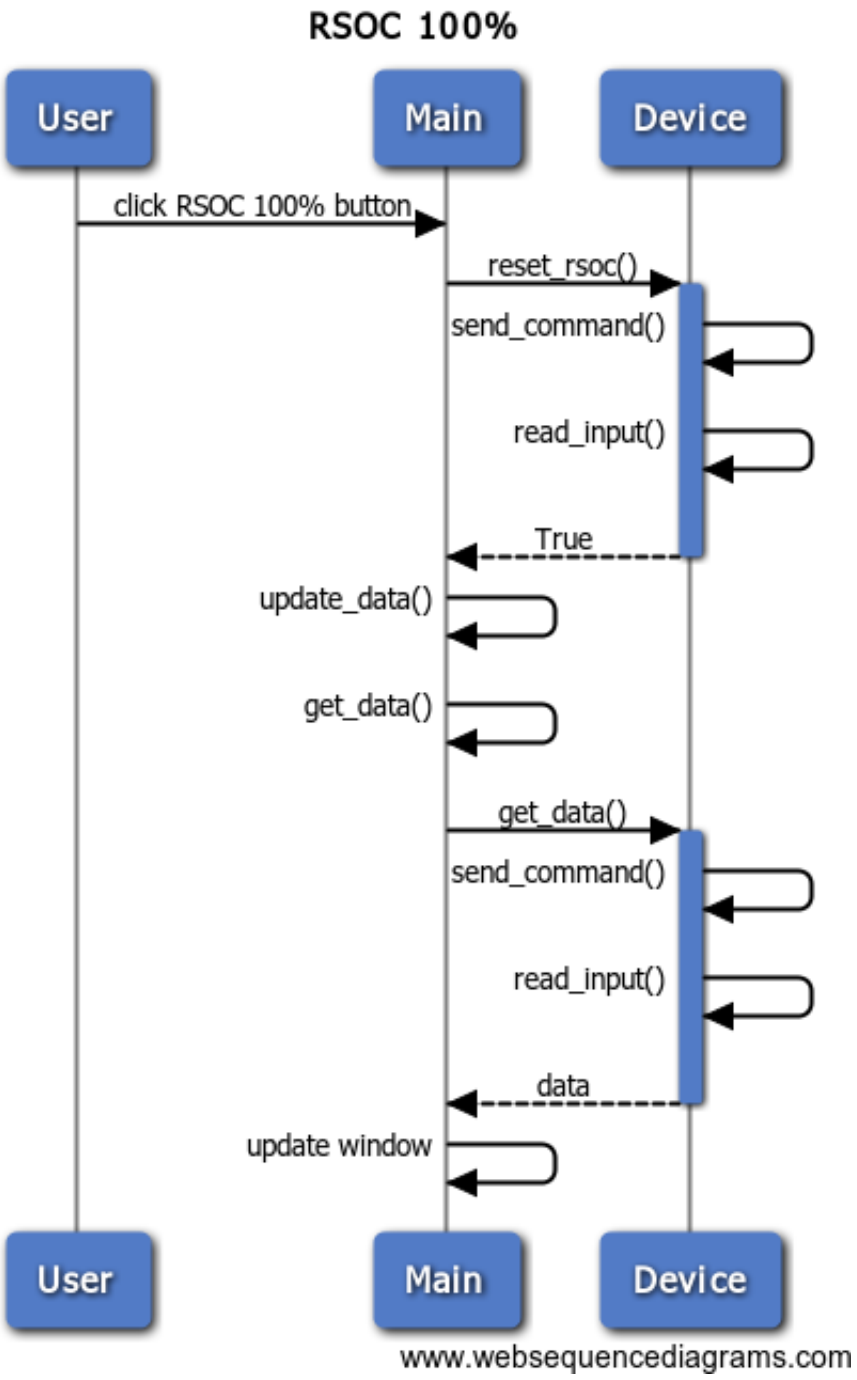
2.5 Diagramak

2.5.1 Read Data



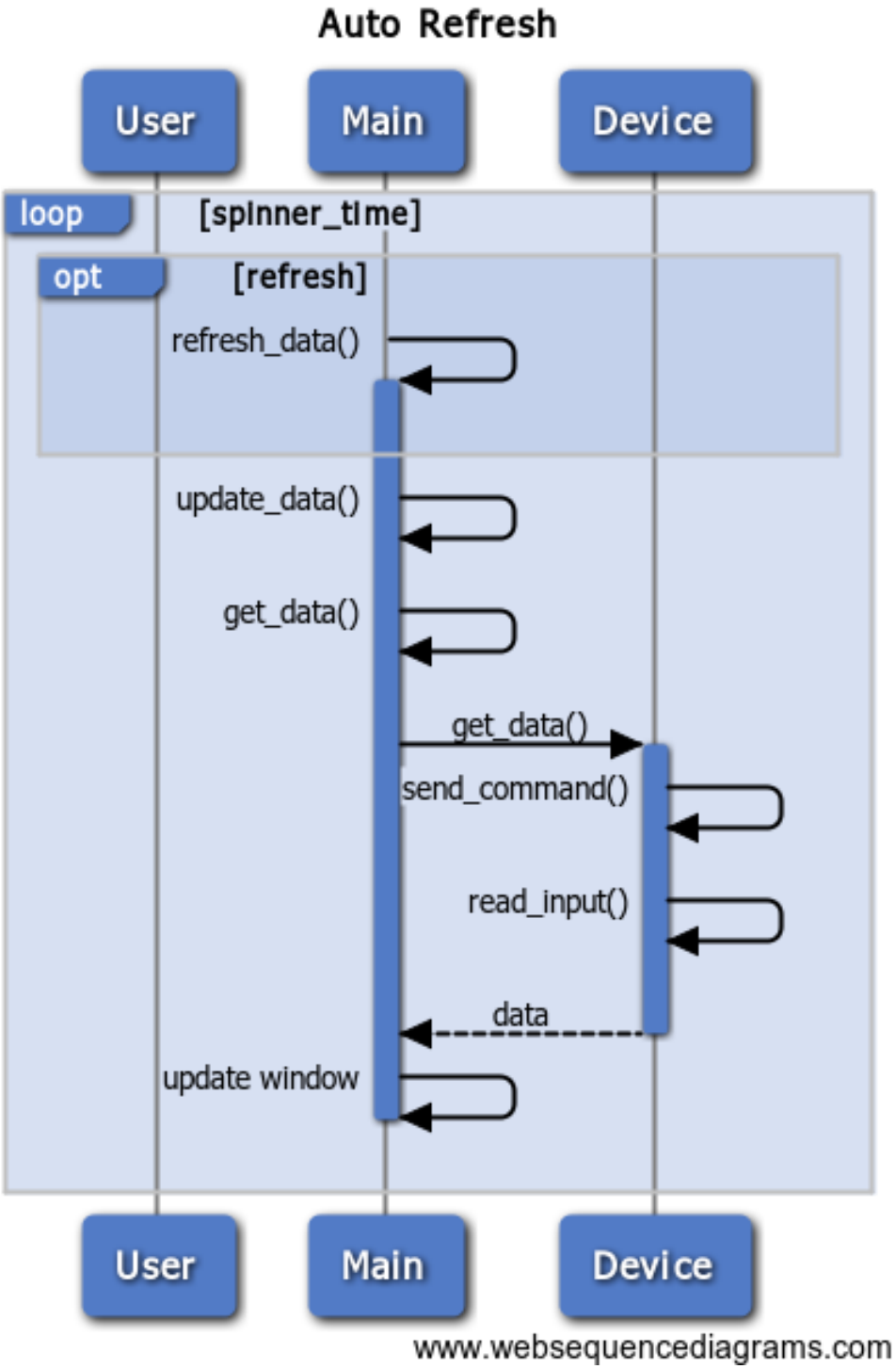
2.3 Irudia: *Read Data sekuentzi diagrama.*

2.5.2 RSOC 100%



2.4 Irudia: *RSOC 100% sekuentzi diagrama.*

2.5.3 Auto Refresh



2.5 Irudia: *Auto Refresh sekuentzi diagrama.*

3. KAPITULUA

Proiektuaren garapena

Atal hau bi zatitan banatu beharra dago. Alde batetik garapena aurrera eramateko erabili diren tresnak, bakoitzari buruz azalpen txiki batzuk erantsiaz.

Bestetik, ordea, proiektuaren garapena bera nola eraman den. Lehen bertsioak nolakoak izan ziren, eta proiektua bukatu arte egon diren aldaketa eta hobekuntza guztiak azalduz.

3.1 Erabilitako tresnak

Ondorengo atalean, proiektua aurrera eramateko erabilitako tresnak azalduko dira.

Esan beharra dago, tresna horien laguntzarik gabe, egin beharreko lana askoz ere astunagoa eta korapilotsuagoa izango zela.

3.1.1 Glade

Tresna hau GTK[1] bidezko interfaze grafikoak sortzeko aplikazioa da.

Honek, aukera ematen du "eskuz"egiteko aplikazioaren diseinua.

Bertan, leihoaren tamaina aukeratu, eta nahi izan diren etiketa, botoi...jar ditzazkegu grafikoki.

Sorturiko fitxategia gorde ostean, kodetik atzitu daiteke erraz asko erabiltzeko.

[1] GIMP Tool Kit

3.1.2 GCC

Honek, C lengoaiako aplikazioa konpilatzen laguntzen du, kode fitxategietatik exekutagarri bat lortuz.

3.1.3 SOCAT

Tresna honek, aukera ematen du plakarik gabe probak egiteko.

Bere laguntzari esker, ondorengo komandoa exekutatuz, bi serie portu birtual lortzen dira.

socat -d -d pty,raw,echo=0 pty,raw,echo=0

Guztia, makinaren terminal batean exekutatuko da. Exekutatzean, terminalean azalduko da zein bi portu birtual sortu dituen.

Hori burututa, egiteko geratuko den bakarra izango da aplikazioa esleituriko portu horietako batera konektatzea eta beste aldean, berriz, serie portua maneiatzeko terminal grafikoa, kasu honetan CuteCom.

Horrela, aplikaziotik agindu bat exekutatzean, beste aldean ikus daiteke jaso den mezua, ondo dagoen edo ez jakiteko; eta nahi izanez gero, mezu bat itzuli, ea aplikazioan lortu nahi dena gertatzen den ikusteko.

3.1.4 CuteCom

Serie portua maneiatzeko terminal grafikoa da, Windows sistema eragileak duen *Hyperterminal*-a bezala funtzionatzen duena, baina kasu honetan, Linux sistema eragilean.

3.1.5 Sublime Text

Testu editorea da, kodea idazteko laguntza askorekin. Ordainpekoa izan arren, probako bertsioa du eta hori erabili da.

3.1.6 VIM

Linux terminalerako testu editore oso ahaltsua da. Plugin batzuk jarri ostean, oso azkarra da berarekin programatzea, hasieran zaila badirudi ere.

Aplikazio hau sarritan erabili da programatzeko.

3.1.7 Texmaker

Dokumentazioa idazteko erabilitako aplikazioa izan da. LaTeX bitartez idatzi da, fakultateko txantiloia erabiliz.

Asko errazten du erabilera, eta *PDF*[2] formatura konpilatzeko aukera ematen du.

3.1.8 www.websequencediagrams.com

Web orri hau erabili da sekuentzia diagramak egiteko, izan ere, *Dia* edo *StarUML* baino erosoagoak da, eta bertan, azkarrago egin daitezke diagramak.

Diagramak, `www.websequencediagrams.com`-en sartu eta bertan dauden laguntzak erabiliz, oso erraz sortzen dira. Gero, bertatik atera daitezke diagramak irudi moduan.

3.2 Garapena

Atal hau berriz, bi zatitan banatuko da. Izan ere, hasiera batean proiektua modu batera egiten hasi arren, ez da horrela burutu azkenean.

Beraz, C lengoaiaz programatua zegoen atala eta Python lengoaiaz hasi eta egin diren hobekuntza guztiak azalduko dira ondoren.

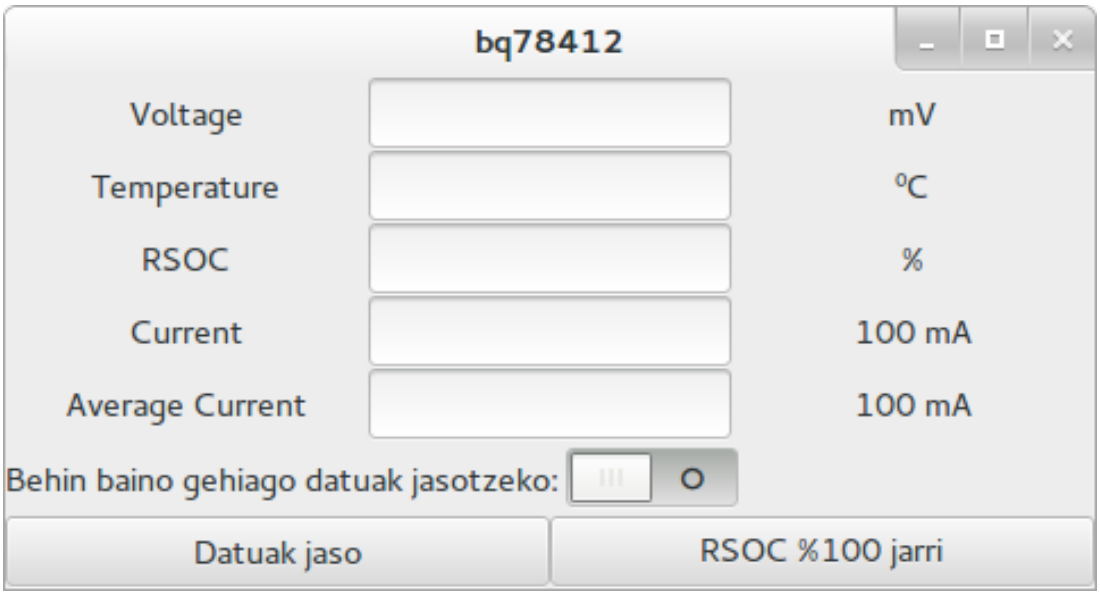
3.2.1 Lehen zatia, C lengoaia

Hasiera batean, ikusirik *driver*-ak programatzeko ohikoena C lengoaia zela, eta Texas Instrument enpresak plakari buruzko liburutegiak lengoaia horretan inplementatuak zituela, berarekin ekin zitzaion programatzeari.

C lengoaia aurretik erabilia izan arren, hain behe mailan erabiltzea, ez zen uste bezain erraza izan.

Hasiera batean, Glade aplikazioarekin ondorengo diseinua lortu zen aplikazioaren inplementaziorako (3.1 irudia).

[2]Portable Document Format



3.1 Irudia: *C-rako sorturiko leihoa.*

Argi ikus daitekeenez, oso sinplea da. Baina, izan ere, ez zen ezer gehiago behar ageri diren 5 datuez gain.

Programarekin aurrera jarraitu ahala, probak egitean ikusi zen punteroekin arazo asko sortzen zirela. Orduan, driver-ak programatzeko beste modu bat aurkitu beharra zegoen, zerbait errazagoa, eta [10] aurkitu zen.

Honi esker, ordenagailuak plaka ohiko aparatu bat bezala erazagutzea lortzen da (ez da bitez-bit mezuak bidaltzen ibili behar; konektatu, deskonektatu eta behar diren tramak bidali bakarrik), eta beraz, serie porturako ohiko liburutegiekin programatzeko aukera ematen du.

Hori egiteko erraztasun handiena **Python**-ek ematen zuenez, azkenean hortaz baliatuta landu zen aplikazioa hasieratik.

3.2.2 Bigarren zatia, Python lengoaia

Garapenaren zati luzeena da. Izan ere, komunikazioa behar bezala gauzatzea lortu ondoren, denbora asko eman zen plakara bidali beharreko mezuak behar bezala sortzen.

Lehenik, esan beharra dago, Glade-rekin sorturiko interfazea erabiltzeko aukera egon arren, kodearen bidez sortua dela, izan ere, errazagotzat jo baitzen behar bezala kontrolatzeko.

Horrek, hasiera batean ez zuen garrantzi handiegirik izan, bi mezu bakarrik bidali behar zirelako, biak aurretik finkatuak, eta jasotakoaren arabera, mezu horietatik beharrezko datuak atera.

Lehenbiziko bertsioa lortu ostean, eta aukera guztiak behar bezala zebiltzala ikusita, parametro bezala zegoen aukera bat, interfazean kokatzea izan zen hurrengo mugarria. Parametro horrek, datuak bakarka jasotzeko aukeratzen bada, datu bat eta hurrengoaren artean zenbat denbora pasa behar den adierazten du.

Hau interfazera eramatean, *Spinner* bat jarri zen erosotasunagatik (3.2 irudian *Auto Refresh* aukeraren lerroan ikus daitekeen moduan). Bertan, minututan adieraziko da zenbatero jaso nahi diren, minutu erdiroko tartearekin zehazteko aukerarekin.

C-ko interfazearen antzeko zerbait jarri zen Python-en, ondorengo interfazea sortuz (ikus 3.2 irudia).

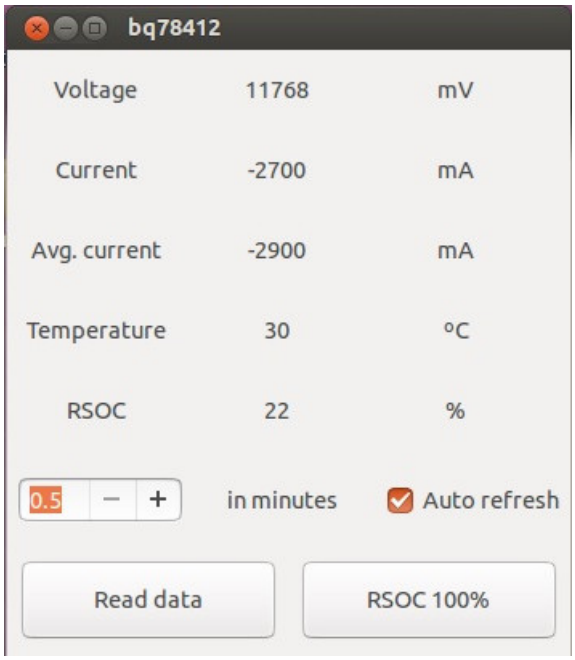
Irudian ageri den denak ondo funtzionatzen zuela konprobatu ostean, hobekuntzak etorri ziren.

Hobekuntzei dagokienean ere, hauek bi zatitan bana daitezkeela esan beharra dago, alde batetik, aukera aurreratuen leihoa, eta bestetik, ordenagailu eramangarri baten bateriaren kontrolean agertzen den bezala, erlojuaren alboan ikono bat, bateriaren edukia (ehunekotan emana) adieraziz, leihoa beti irekita ez eduki behar izateko.

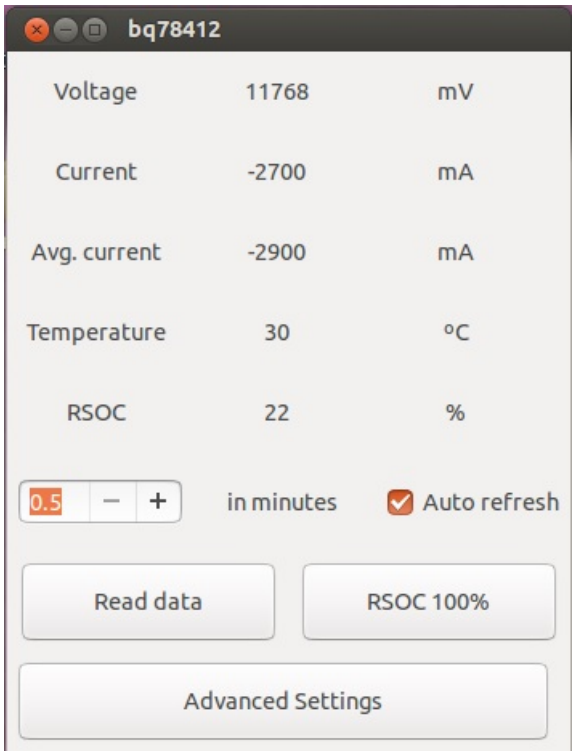
Hasteko, aukera aurreratuekin egin zen lan. Horretarako, aurretik ikusiriko interfazean (3.2 irudia), beste botoi bat gehitu zen, ondorengo irudian ikus daitekeen moduan.

Hortarako, 3.3 irudian agertzen den *Advanced Settings* botoia sakatuko da. Hori egin ostean, leiho berri bat irekiko da nagusiaren gainean.

Leiho horrek botoiak besterik ez ditu, ondorengo irudian ikus daitekeen moduan (ikus 3.4 irudia).



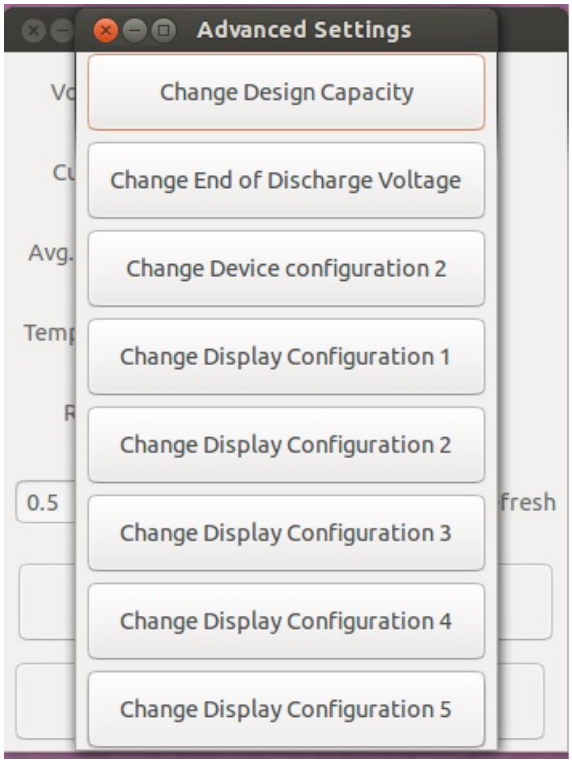
3.2 Irudia: *Python-eko hasierako leihoa.*



3.3 Irudia: *Python-eko leiho nagusia.*

Advanced Settings sakatutakoan, beste menu bat lortzen da. Bertan, plakaren parametro desberdinak konfiguratzeko aukerak agertuko dira.

Parametro hauen artean, LED bakoitza zenbat denboran zehar egongo den piztuta adieraz daiteke, noiz egin behar duen txistu *buzzer*-ak...



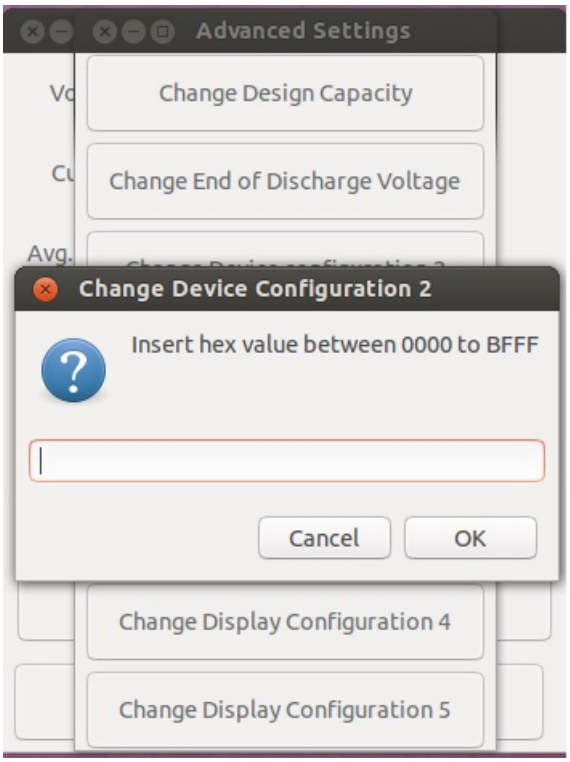
3.4 Irudia: *Python-eko aukera aurreratuen menua.*

Horietako edozein balio aldatzeko, *click* egitean, 3.5 irudiko leihoa edo antzekoa aterako da.

Adibide bezala hartu den irudian, ikus daiteke adierazita dagoela zein balio tartean egon behar duten balioek. Aukera bakoitzak desberdinak ditu, eta balio horietatik kanpo dagoen bat sartuz gero, informazio mezu bat aterako da horren berri ematen.

Puntu honetan, aplikazioa ez balitz erabili izan robot-ean proiektua bukatu aurretik, arazo larri bat ezkutuan uzteko arriskua sortu zatekeen.

Arazoa, balioa sartzeko garaian agertzen zen leihoan, *Cancel* botoia sakatzen bazen, *OK* botoiak egiten zuen bera egingo zuen, eta balio hutsa hartuko zuen sartutako balio bezala, hortaz, aplikazioak balioa ezarritako tarteetatik at zegoela ulertuko zuen, errore mezua azalduz.



3.5 Irudia: *Python-eko aukera aurreratuan datuak sartzea.*

Hori konpondu ostean, *AppIndicator* delakoa sortu zen. Ideia ona suertatu zen, oso lagungarria zelako leihoa irekita eduki gabe jakitea zenbat bateria geratzen zitzaion robotari.

Hori egiteko, Internet-eko adibideak probatu ziren, kodeak nola funtzionatzen zuen jakiteko. Esan beharra dago, adibide hauetatik, gehienak ez zebiltzala behar bezala.

AppIndicator

Zati hau esan liteke dela berriena, izan ere, lehen aldia da horrelako tresna bat sortzen dena.

Hemen, *Linux* sistema eragilerako sortu zen bateriaren karga adierazten zuen ikonoa, alboan bateriaren edukiera (ehunekotan) adieraziz, eta gainean sakatuz gero, menu bat ateratzen da, bertan ageri direlarik aplikazioan dauden aukerak (argiago, ikus 3.6 irudia).

Hori sortzeko, [1] eta [2] helbideetako gidetan oinarritu zen lana.

Bertan argi adierazten da menu bat nola egin. Jarritako adierazpideak jarraituz, bertsioekin borrokatu eta lana burutu ostean, lortu zen menu bat agertzea; baina arazo bat zegoen, ez zen sortu behar zuen tokian.

Izan ere, ordenagailuan, adibideetan ageri zen bezala, goiko menuan ez zuen ezer sortzen.

Hasiera batean, pentsatu zen dena gaizki zebilela, baina bapatean ikusi zen azpiko menuan (defektuz ezkutuan dagoenean) sortzen zuela.

Geroago ikusi zen, ordenagailuaren gauza zela, izan ere, erabilitako ordenagailuak *Gnome* darabil *Unity*-ren ordez, biak Ubuntu sistema eragilea izan arren. Honetaz ohartzeko, robotean probak egin behar izan ziren.

Menu honen sorkuntza, erosotasunaren aldetik sorturiko hobekuntza izan zen. Honi esker, ez zen zertan leiho nagusia ireki behar bateriaren edukia ezagutzeko, beraz, beste edozertan jardun arren, erabiltzaileak kontrola zezakeen zein edukiera zuen bateriak.

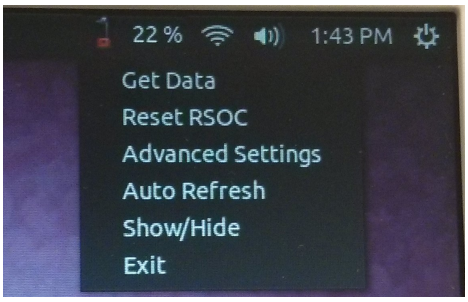
Hasiera batean, Ubuntuk ekartzen duen ikonoetariko bat hartu eta ordu batzuk pasa beharra izan ziren menua bakarrik ateratzeko nahi ziren aukerekin eta honek, nahi nuen bezala funtzionatzeko. Hori lortu ostean, ikonoen ordua iritsi zen.

Beraz, nabaria zen indikadore honek hiru egoera desberdin zituela, eta horiek ez zirela nahikoa bateriaren egoerak adierazteko. Baina beste gailu batzuek bateria adierazteko ikono kopuru gehiago erabiltzen bazituzten, horrek esan nahi zuen aukera egon behar zela.

Probak egin ostean, ikusi zen hori egiteko aukera, hau da, egoera berean ikono desberdinak erabiltzea zela.

Horrela, behar zen ikonoa erakusteko, egoera aldatu gabe, ikonoa aldatu besterik ez zen egin behar.

Ideia xume horren ondorioa, ondorengo irudian ikusten den emaitza:



3.6 Irudia: *AppIndicator*.

Irudian, argi ikusten da zein bateria edukiera (ehunekotan adierazia) duen, eta baita ere, kasu honetan, gorri dagoela ikonoa, bateria gutxi geratzen delako.

Gero, menuan, leiho nagusian dauden botoiez gain, *Show/Hide* aukera dago.

Botoi horri esker, gure aplikazioaren leihoa erakutsi edo ezkutatzeko aukera dago.

4. KAPITULUA

Ondorioak

Proiektuak hainbat ikasgai barneratzeko balio izan du, eta baita, beste zenbait gauza berri ikasteko ere.

Ikasitakoen artean azpimarratzekoak dira: Python-en programatzea eta driver bat nola programatu.

Esperientzia honetan zehar, Marisorgin robotarentzako driver bat programatzeari esker beste driver-en bat programatzeko erraztasuna eskuratu da zalantzarik gabe.

Baita ere, Java ez den beste lengoaia batean interfaze grafikoak egitea lan ildo berria izan da, eta hori era egokian nola burutu asmatzeko, denbora luzea behar izan da.

Eskertzekoa da probak egitean Socat aplikazioa erabiltzea, lana asko erraztu duelako. Horri esker, etxean bertan egin izan ahal dira proba guztiak, eta ez da BQ78412 plakaren beharrik izan ia ezertarako, proba gehienak tresna horren laguntzaz eta portu birtualak erabilita egin baitira.

Azkenik, hobekuntza modura jarri den bateriaren kontrolerako "indikadorea"izan da proiektuaren urrats interesgarrienetako bat, izan ere, jakinmina eragiten du eta.

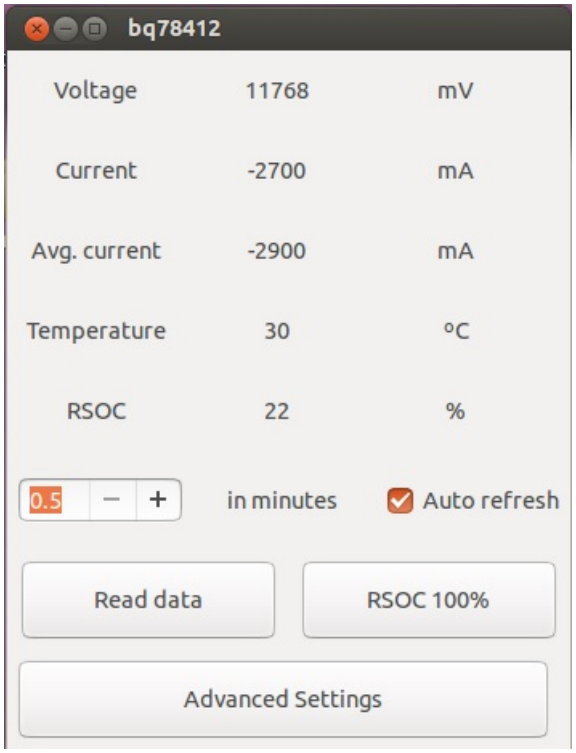
Bukatu aurretik ezin ahaztuko da proiektua oso probetxuzkoa izan dela, benetako aplikazio informatiko bat sortzeko aukera eman baitu. Aurrerantzean, Marisorgin robotarekin lan egiteko erabilgarria da programatutako driverra, eta Texas Instrument enpresako BQ78412 plaka Linux sistema eragilearekin erabili nahi duten guztientzat ere bai.

Eranskinak

A. ERANSKINA

Gidaliburua

Jarraian, aplikazioaren gidaliburu txiki bat idatziko da, aplikazioak dituen aukerak eskura edukitzeko.



A.1 Irudia: *Lehen pantaila.*

Aplikazioaren lehen pantailan argi ikus daitekeen moduan, 4 aukera desberdin daitezke.

Banan-banan azalduko dira ondorengo puntuetan.

A.1 Read Data

Botoi honek plakari mezu bat bidaltzen dio, beste honek erantzun gisa bateriaren datuak itzultzeko itxaropenarekin.

Datuak jasotzean, aplikazioak bakoitza dagokion tokian erakutsiko du, jasotako mezutik behar dituen datuak aterata.

A.2 RSOC 100%

Bateria plakara konektatu gabe dagoela kargatzen bada, plakak bateriaren kapazitatea %50-ean dagoela aurreikusten du. Horretarako, botoi hau sakatuko da, bateria kargatu berria jarri ostean, plakari adierazteko bateriaren kapazitatea guztiz betea dagoela.

Hori gertatzen denean, plakari mezu bat bidaltzen zaio adierazteko, eta honek erantzun zuzena jaso ostean, *Read Data* botoia sakatu ostean gertatzen den bera egingo du, datuak irakurri, erabiltzaileari argi erakutsiz bateriaren kapazitatea behar bezala dagoela.

A.3 Auto refresh

Aukera honek *Read Data*-k egiten duen berbera egiten duela esan daiteke.

Hau da, aktibaturik dagoen bitartean, ezkerretan duen balioaren arabera[1] (A.2 irudia), datuak jasotzen jarriko litzateke, adierazitako denbora pasatzen den aldiro. Ziurrenik aukera erabiliena izango da, izan ere, beste aplikazioren bat martxan dagoen bitartean, botoia sakatzen ibiltzea ez baita oso erosoa.

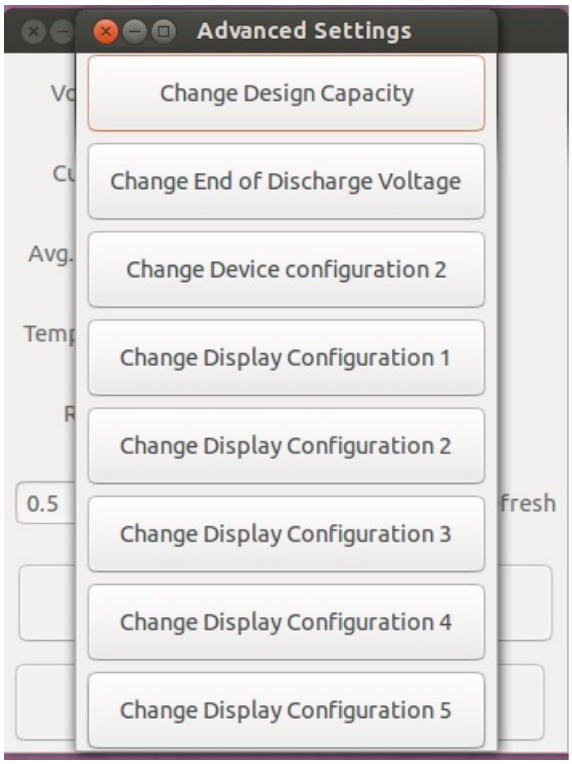
A.2 Irudia: *Auto Refresh*.

Beraz, hau martxan jarri eta berak bakarrik egingo luke lana.

[1] Balioa minututan adierazia

A.4 Advanced Settings

Hemen *click* egitean, beste leiho bat agertuko da dugun leihoaren gainean, beste 8 aukera erakutsiz. Ondorengo leihoa hain zuzen ere (A.3 irudia).



A.3 Irudia: *Aukera aurreratuak.*

Aukera hauek ez dira ohikoak, plakak dituen *LED*-en frekuentzia konfiguratu, plakak egiten dituen soinuak konfiguratu... bezalako gauzak egiteko balio baitu.

Hauek, behin konfiguraturik, plakak bere barnean gordeko ditu, beraz, ez da derrigorrezkoa ordenagailua pizten den bakoitzean aldatzen ibiltzea.

Esan beharra dago, hemen bakoitzaren funtzionamendua modu ahal den argienean eta laburrenean azaltzen saiatu arren, informazio gehiago lortzeko erarik onena plakaren eskuliburua irakurtzea izango litzatekeela. [7].

A.4.1 Change Design Capacity

Bateriaren edukieraren kapazitatea adierazteko da. Normalean, balioa ez da aldatu beharrik izango. Aldaketa, beste mota bateko bateria bat sartzean gertatuko da.

Balio hau, hamartarrez idatzia egongo da, eta **1** eta **3270** balioen arteko balio bat izan behar du. Balio hauek *100mAhr* unitatean adieraziko dira.

Beraz, adibide moduan **3** balioa sartzean, honek esan nahi du, *300 mAh* direla, beraz, *0,3 Ah* balioa esleitu zaiola.

A.4.2 Change End of Discharge Voltage

Balio honek adieraziko digu zein tentsioren azpitik detektatu behar den bateriaren deskargaren amaiera.

Balio hau, hamartarrez idatzia egongo da, eta **4000 mV** eta **65535 mV** balioen arteko balio bat izan behar du.

A.4.3 Change Device Configuration 2

Honekin plakaren *buzzer*-ak noiz soinua aterako duen konfigura daiteke. Txistu antzeko bakoitza segundu batekoa izango da, eta txistu hauen arteko denbora ere, segundu batekoa da.

Balio hau, hexadezimalez idatzia egongo da, eta **0000** eta **BFFF** balioen arteko balio bat izan behar du.

BITS	CONDITION	DESCRIPTION	NUMBER OF BEEPS
[1:0]	Empty	RTTE = 0 minutes, during discharge	0 to 3
[3:2]	LED0 turns off	RTTE = time in LED0, during discharge	
[5:4]	LED1 turns off	RTTE = time in LED0 + time in LED1, during discharge	
[7:6]	LED2 turns off	RTTE = time in LED0 + time in LED1 + time in LED2, during discharge	
[10:8]	Overvoltage	Battery voltage > OvThresh	0 to 7
[13:11]	Undervoltage	Battery voltage < UvThresh	

A.1 Taula: Device Configuration 2

A.4.4 Change Display Configuration[1:5]

Atal honetan azken 5 botoien erabilera azalduko da. Izan ere, denek berdin funtzionatzen dute, bakarrik, bakoitzak LED desberdinarentzako balioak aldatzen dituela.

Balio hauek, hexadezimalez idatziak egongo dira, eta **0000** eta **FFFF** balioen arteko balio bat izan beharko dute.

Parameter	Bits[15:8] Allocation	Bits[7:0] Allocation
DisplayConfiguration1	Time in LED1	Time in LED0
DisplayConfiguration2	Time in LED3	Time in LED2
DisplayConfiguration3	Time in LED5	Time in LED4
DisplayConfiguration4	Time in LED7	Time in LED6
DisplayConfiguration5	Time in LED9	Time in LED8

A.2 Taula: Display Configuration

A.2 taulan ikus daiteke, zein bit aldatu behar diren eta zein helbidetara bidali LED diodoen konfigurazio zuzena egiteko.

Plakan 10 LED diodo daude, bateriaren egoera adierazteko. Bitetan adierazi beharko da zenbat minutu komeni zaigun LED bakoitza pizturik edukitzea.

LED hauek A.4 irudian ageri diren LED-ak dira.

Hemen idatzitako guztia hobeto ulertzeko, eta informazio gehiago eskuratzeko, [7] dokumentuan begiratzea gomendatzen da.

A.4 Irudia: *BQ78412 plaka.*

A.5 Parametroak

Azken atal honetan, kontsolatik (sistemaren terminala) aplikazioa exekutatzean, parametro desberdinak esleitu dakizkioke aplikazioari.

Honako hauek dira dauden aukerak:

`-h, --help:` Laguntza mezua erakusten du. Hemen azaltzen dira zertarako balio duen parametro bakoitzak.

`-p, --port:` Plakari esleituriko portua zein den adierazteko balio du.

Ezer adierazi ezean, */dev/ttyUSB0* portura konektatzen saiatuko da.

`-t, --timeout:` *Timeout*[2]-a adierazteko balio du. Ohikoa den parametroa serie portura konektatzen den gailuetan.

Ezer adierazi ezean, segundu bateko *Timeout*-a erabiltzen du.

Balioak 1 eta 10 segundu tartean egon beharko du.

`-v, --verbose:` Era honetan adierazten da bidalitako mezuak, jasotakoak eta abar kontsolatik erakusteko aukera. Normalean, *log*-etan erabiltzen den sistema da.

Zein unetan zer mezu jaso edo bidali duen adierazten du.

Parametro hau adieraztearekin nahiko da martxan jartzeko (ez du baliorik jasotzen).

Honako adibidean honetan, */dev/ttyUSB1*-era nola konektatuko ginatekeen, 3 segunduko *Timeout*-arekin eta *Verbose* modua gaiturik erakusten da:

./main.py -p /dev/ttyUSB1 -t 3 -v

edo

./main.py --port /dev/ttyUSB1 --timeout 3 --verbose

Aukeratu egokiena edo erosoena.

[2]Denbora-muga: zenbat denbora geratu plakaren erantzunaren zain.

B. ERANSKINA

Device klasearen erabilera

Serie portua erabiltzeko sorturiko klasea da. Hemen adieraziko da nola eta zertarako erabili den funtzio bakoitza.

Ondorengo funtzioak erabiltzeko, istantzia bat egitea besterik ez da beharrezkoa.

B.1 __init__

Python-en eraikitzailea adierazteko modua.

Funtzioaren goiburukoa:

```
def __init__(self, address, timeout, bitrate)
```

Ikus daitekeenez, hiru parametro jasotzen ditu.

1. *Address* = Konektatuko garen portuaren helbidea adierazteko erabiltzen da.
2. *Timeout* = Zenbat segundu egon behar den mezu baten zain errorea gertatu dela jakin aurretik.
3. *Bitrate* = Zenbat *bps*[1]-tara funtzionatzen duen konexioak. Bidalitako mezuak erabiltzen duen abiadura, plakak jasotzen duenaren berdina izan behar du egoki funtzionatzeko.

[1] Bits per second

B.2 get_data

Funtzioaren goiburukoa:

```
def get_data(self)
```

Ez du parametrorik jasotzen. Funtzio honek plakari irakurri nahi ditugun datuak jasotzeko mezua bidaltzen dio eta datuen zain geratzen da.

Datuak badaude, *array* batean banatzen dira datuak, behar bezala adieraziz, eta *array*-a itzuli egiten da.

Bestela, **None** itzuliko du.

B.3 reset_rsoc

Funtzioaren goiburukoa:

```
def reset_rsoc(self)
```

Ez du parametrorik jasotzen. Funtzio honek %100-ean jartzen du RSOC-a. Horretarako, mezua bidaltzen du, eta jasotzen duen erantzuna *ACK*[2]-ren zain geratzen da.

B.4 send_command

Funtzioaren goiburukoa:

```
def send_command(self, command)
```

Parametro bakarra jasotzen du. *command* parametroan adierazi behar da plakara bidali nahi den mezua.

[2] Acknowledgment

B.5 read_input

Funtzioaren goiburukoa:

```
def read_input(self, size)
```

Parametro bakarra jasotzen du. *size* parametroan zenbat *byte* irakurri nahi diren adierazi behar da.

Funtzio honek lau errore mota eman ditzake, eta bakoitzak bere salbuespena altxako du:

1. ACK = Jasotako mezuan ACK ez bada jasotzen. *ACKError* altxako da.
2. CRC[3] = Jasotako mezuaren CRC-ak ez badu behar duen CRC-a eramaten. *CRCError* altxako da.
3. Size = Jasotako mezuaren luzera ez bada jaso nahi dugunaren berdina. *sizeError* altxako da.
4. timeout = Mezurik ez badu jaso sortzailean jarritako timeout-a pasa ostean. *timeoutError* altxako da.

B.6 parse_data

Funtzioaren goiburukoa:

```
def parse_data(self, raw_data)
```

Parametro bakarra jasotzen du. Bertan, *Read Data* egitean jasotzen den trama pasako dugu parametro gisa, eta honek, trama horretatik *array* bat itzuliko du.

Array honetan ondorengo datuak egongo dira:

1. current
2. avg_current
3. temperature
4. rsoc

[3]Cyclic Redundancy Check

B.7 CRC

Funtzioaren goiburukoa:

```
def crc(self, data)
```

Parametro bakarra jasotzen du. Bertan pasatako tramaren CRC-a kalkulatzen du, eta gure kasuan, tramaren azken balioarekin konparatzen du, izan ere, hau da plakak bidaltzen digun CRC-a.

Berdinak badira, *True* itzuliko du, bestela, *False*.

B.8 ACK

Funtzioaren goiburukoa:

```
def ack(self, data)
```

Parametro bakarra jasotzen du. Parametro hau jasotako trama izango da, eta ACK konprobatzen du. Gure kasuan, bigarren posizioan "*!*"ikurra doan.

Kasu honetan ere, boolear bat itzuliko du, *True* berdinak diren kasuan, bestela, *False*

C. ERANSKINA

Instalazioa

Atal honetan, aplikazioa behar bezala jartzeko pausuak azalduko dira.

Lehenik, *PySerial* nola instalatu adieraziko da, hau gabe, aplikazioak ez baitu funtzionatuko.

Ondoren, nola sortu ikono bat aplikazioa exekutatzeko; eta azkenik, nola jarri automatikoki exekutatzeko aplikazioaren aukera.

C.1 PySerial

Lehenik, https://pypi.python.org/pypi/pyserial helbidetik tar.gz fitxategia deskargatuko dugu.

Behin ordenagailuan dugula, barneko fitxategi guztiak aterako dira karpeta batera.

Karpeta horretara terminaletik jo behar da, eta bertan izanik, ondorengo bi komandoak exekutatuko ditugu:

- `python3 setup.py build`
- `python3 setup.py install`

C.2 Konfiguratu ordenagailua portu bera esleitzeko beti

Zati honetan, portu berdina esleitu araziko diogu ordenagailuari, izan ere, berrabiarazten den bakoitzean, nahi duen portua esleitzen du logikarik gabe, eta honek, arazoak ekar ditzake gero jakiteko ze porturekin lotu behar den aplikazioa konexioa zuzen egin dezan.

Horretarako, ondorengo lerroa duen fitxategi bat sortuko da:

```
KERNELS=="3-3.3", ATTRS{product}=="FT232R USB UART" NAME="ttyUSB1",
SYMLINK+="ftd32"
```

Kasu honetan, fitxategi hau *90-usb-serial.rules* moduan gorde da, fitxategi sistemako */etc/udev/rules.d/* karpetaren barnean.

Lerro honekin lortuko duguna ondorengoa da: *FT232R USB UART* etiketa duen USB bat konektatzean, hau beti */dev/ttyUSB1* portura konektatzea eta gainera, esteka sinboliko bat sortzea */dev/ftd32* portura, azken modu hau errazagoa delarik gogoratzeko.

C.3 USB portuari baimenak eman

Erabiltzaile guztiek idazteko eta irakurtzeko baimenak ez dituztenez USB portuetatik, baimenak eman beharra egon daiteke. Horretarako, ondorengo kodea duen fitxategi bat sortuko da:

```
#! /bin/sh
### BEGIN INIT INFO
# Provides:          ChangeUSBPermissions
# Required-Start:    $remote_fs $syslog $all
# Default-Start:     2 3 4 5
# Short-Description: change permissions of ttyUSB* ports
### END INIT INFO

/bin/chmod a+rw /dev/ttyUSB0
/bin/chmod a+rw /dev/ttyUSB1

exit 0
```

Nahi dugun izenarekin gordeko da fitxategia, esate baterako, *baimenak* izenarekin. Hau egitean, kontsolaren bitartez fitxategia dagoen karpetara joango gara eta ondorengo komandoak exekutatuko:

- `chmod +x baimenak`
- `sudo ./baimenak`

Azken komandoa super-erabiltzaile moduan exekutatu behar da, ezin bait zaizkio norbere buruari nahi diren baimenak eman.

C.4 Mahaigaineko ikonoa

Bi aukera daude hau egiteko:

1. Ondorengo kodea duen fitxategi bat sortu:

   ```
   #!/usr/bin/env xdg-open

   [Desktop Entry]
   Version=1.0
   Type=Application
   Terminal=false
   Icon[en_US]=/home/bee/softwarea/ftd32/jon/icons/gpm-battery-100.png
   Name[en_US]=ftd32
   Exec=python3 /home/bee/softwarea/ftd32/jon/main.py -p /dev/ttyUSB1
   Name=ftd32
   Icon=gnome-panel-launcher
   ```

   Hau, adibidez, *ftd32launcher.desktop* moduan gorde. Kontuan izan behar da *Icon[en_US]* parametroa eta *Exec* parametroa, aldatu beharra izango direla.

   Eta azkenik, */usr/share/applications* karpetan kopiatu fitxategia.

2. *gnome-panel* aplikazioa instalatu eta ondorengo komandoa exekutatu terminal batean:

   ```
   gnome-desktop-item-edit /usr/share/applications --create-new
   ```

   Hor dauden eremuak bete ostean, gorde aldaketak eta sortuko zaigu ikonoa.

C.5 Aplikazioa abiaraztea ordenagailua piztean

Honetarako, biderik errazena *startup applications* aplikazioa bilatu, exekutatu, eta berri bat sortu

```
python3 /home/bee/softwarea/ftd32/jon/main.py
```

exekutatuko duena.

Kontuan hartu, aplikazioa non gorde den eta horren arabera, helbidea aldatu beharko dela.

Bibliografia

[1] Application indicators. URL: http://developer.ubuntu.com/resources/technologies/application-indicators/.

[2] Create indicator applet for ubuntu unity (with python!), jun 2011. URL: http://conjurecode.com/create-indicator-applet-for-ubuntu-unity-with-python/.

[3] How can i create launchers on my desktop?, jun 2011. URL: http://askubuntu.com/questions/64222/how-can-i-create-launchers-on-my-desktop.

[4] Micah Carrick. Gtk+ and glade3 gui programming tutorial, dec 2007. URL: http://www.micahcarrick.com/gtk-glade-tutorial-part-1.html.

[5] FTDI Chip. D2xx programmer's guide, February 2012. URL: http://www.ftdichip.com/Support/Documents/ProgramGuides/D2XX_Programmer%27s_Guide%28FT_000071%29.pdf.

[6] Python Software Foundation. Logging facility for python. URL: http://docs.python.org/3.3/library/logging.html.

[7] Texas Instruments. bq78412 pb-acid battery state-of-charge indicator with run-time display, oct 2010. URL: http://www.ti.com/lit/ds/symlink/bq78412.pdf.

[8] JustChecking. Howto: Virtual serial ports on linux using socat, and more, June 2009. URL: https://justcheckingonall.wordpress.com/2009/06/09/howto-vsp-socat/.

[9] Chris Liechti. pyserial api. URL: http://pyserial.sourceforge.net/pyserial_api.html.

[10] Dinesh Rajpoot. How to enable usb to serial port(ft232r) in linux, June 2010. URL: http://dinesh-rajpoot.blogspot.com.es/2010/06/how-to-enable-usb-to-serial-portft232r.html.

[11] Sutekh. Create your own udev rules to control removable devices, apr 2006. URL: http://ubuntuforums.org/showthread.php?t=168221.