EIROPAS SAVIENĪBA
Eiropas Reģionālās attīstības fonds

IEGULDĪJUMS TAVĀ NĀKOTNĒ

Elektronikas un telekomunikāciju fakultāte

Rīgas Tehniskās universitātes Elektronikas un telekomunikācijas fakultātes Radioelektronikas institūta projekts „Hibrīdās intelektuālās akustiski-optiskās sistēmas izstrāde nemedījamo un migrējošu putnu sugu nodarīto postījumu samazināšanai Latvijas akvakultūras nozarē" Nr. 17-00-F02201-000001

Atskaite par projekta uzdevumiem 2.1.-2.3.,3.1.

2019

Satura radītājs

1. U2.1 Korpusa un citu mehānisko komponenšu izstrāde/izvēle 10.-13.mēn.

1.1. Nepieciešamo korpusu un mehānisko komponenšu pārskats un tehniskās prasības

HAOS sistēmu veido sauszemes moduļi un peldošas platformas (plosti). Sauszemes moduļi ir aprīkoti ar lāzeri, skaļruņiem, vadības plati, sprieguma pārveidotājiem, bet nepieciešamības gadījumā arī ar akumulatoru un saules paneļiem. Uz plostiem atrodas vadības plate, sprieguma pārveidotāji, akumulators, saules panelis/-i, rupora skaļrunis/-i, zemūdens skaļrunis.

Prasības sauszemes moduļu korpusiem:

- Lāzera novietojums virs zemes līmeņa – vismaz 1,5 m;
- IP klase – ne mazāka par IP44;
- Jābūt stingri novietotiem lai izturētu stipras vēja brāzmas;
- Jābūt viegli transportējamiem un uzstādāmiem – bez ievērojamas vides transformācijas.

Prasības peldošam platformām:

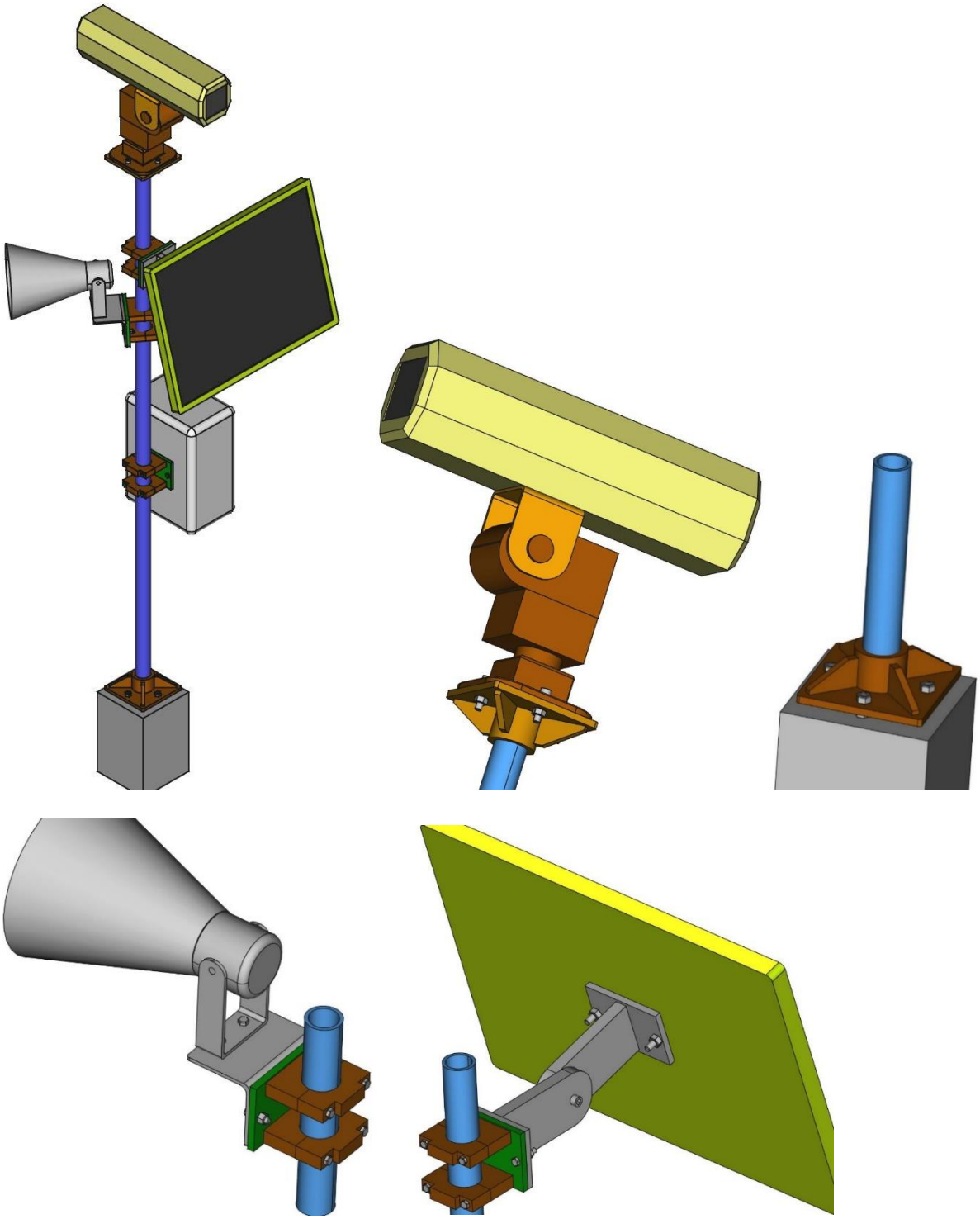
- Celtspēja – ne mazāka par 100 kg;
- Gabarīti – kvadrāta platforma ar malu ne lielāku par 1,2 m;
- Jābūt aprīkotām ar rokturiem ērtākai iekraušanai un uzstādīšanai.

Minētas prasības ir iespējams realizēt dažādos veidos. Zemāk ir parādīti daži varianti.

1.2. AP2.1.1 Korpusu un citu mehānisko komponenšu rasējumi

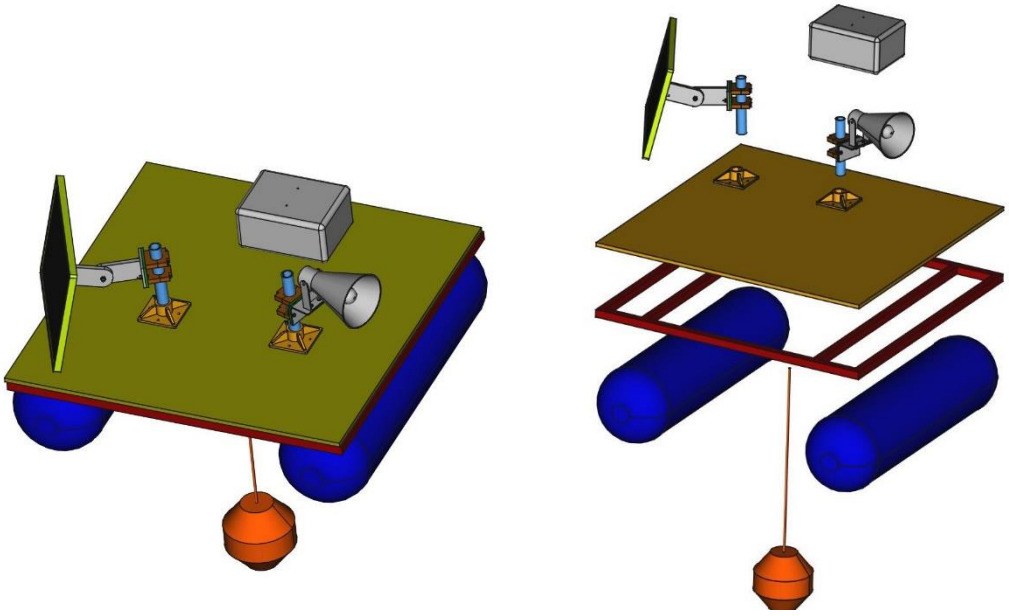
Sauszemes moduļa korpusa realizācija.

Kā pirmais variants sauszemes moduļa realizācijai tika apskatīts aparatūras izvietojums uz staba. Elektroniku ir iespējams novietot speciālajā kastē no paaugstinātas izturības plastmasas. Savukārt saules paneļa montāžai ir nepieciešams pasūtīt vai izgatavot speciālu balsteni. Pie šādas konstrukcijas plusiem var pieskaitīt mazu platību, relatīvi vienkāršu realizāciju gadījumā, ja tiek izmantoti rūpnieciski ražoti risinājumi. Galvenais mīnuss – nepieciešama stabila pamatne pie kuras var pieskrūvēt stabu, taču šāda veida uzstādīšana ir darbietilpīga un pret to arī izteicas dīķu īpašnieki.



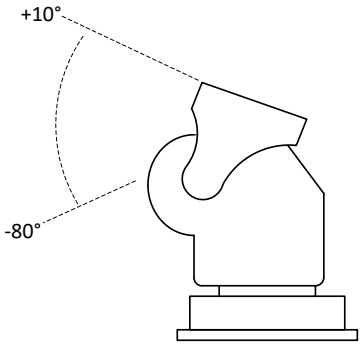
Peldošas platformas realizācija.

Pateicoties tam, ka dīķos ūdens ir diezgan mierīgs un augsti viļņi tur nav novērojami ir iespējams aparatūru izvietot uz plosta. Pastāv vairākas plostu realizācijas iespējas. Viena no tiem ir izmantot pontonus vai mucas pie kurām augšā var piestiprināt saplākšņa pamatni, savukārt pie kuras jau var stiprināt ūdensizturīgu kasti, skaļruņus un saules paneli. Plosta novietojuma uz dīķa fiksācijai var izmantot būvmateriālu veikalos nopērkamus betona blokus (noenkurošanai).



Visas konstrukcijas tehnisko dokumentāciju var redzēt pielikumā 1.

Lāzera orientēšanas sistēmas vertikālas kustības ierobežošana.



YP3040 ir spējīga pacelt lāzeri uz 10° virs horizonta - tas nozīme, ka var izveidoties situācija kad stars spīdes debesīs. Lai novērst šādu situāciju var realizēt programmatisku kompensāciju. Bet tas nedos 100% garantiju (jebkura programma var dot funkcionēšanas kļūdu). Maksimāli iespējamu garantiju var dot tikai elektromehāniskā ierobežošana ar slēdžu izmantošanu. Tādi lāzera piedziņā ir, bet ja horizontālā virzienā tos var regulēt, vertikālā tas nav iespējams. Rezultātā vienīga iespēja – papildināt YP3040 ar mehānisko mezglu, kas neļaus pacelties lāzera staram virs horizonta.

Tika realizētas divas konstrukcijas. Pirmā pilnīgi neatkarīga, bet manāmi palielina sistēmas kopēju augstumu un svaru.

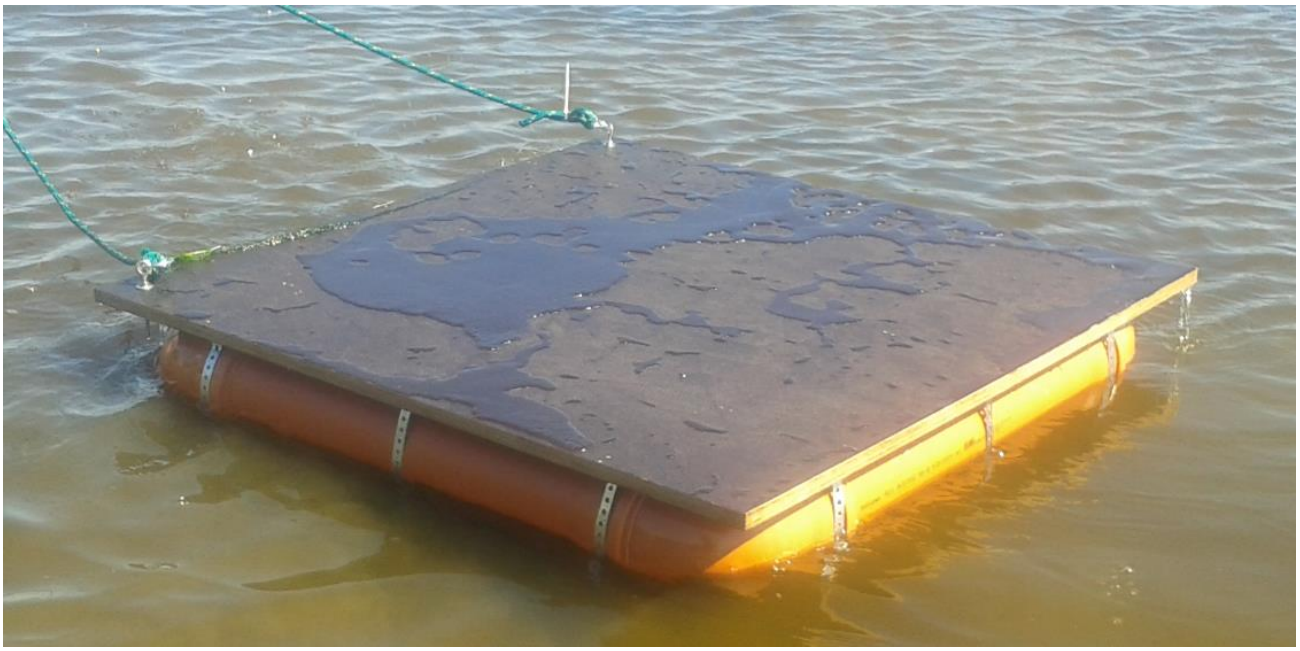
Otrā variantā tādu trūkumu nav, bet ir vajadzīga eksistējošas ierīces neliela piedzīšana.

1.3. AP2.1.2 Pirmā korpusa realizācija

Plosta realizācija.

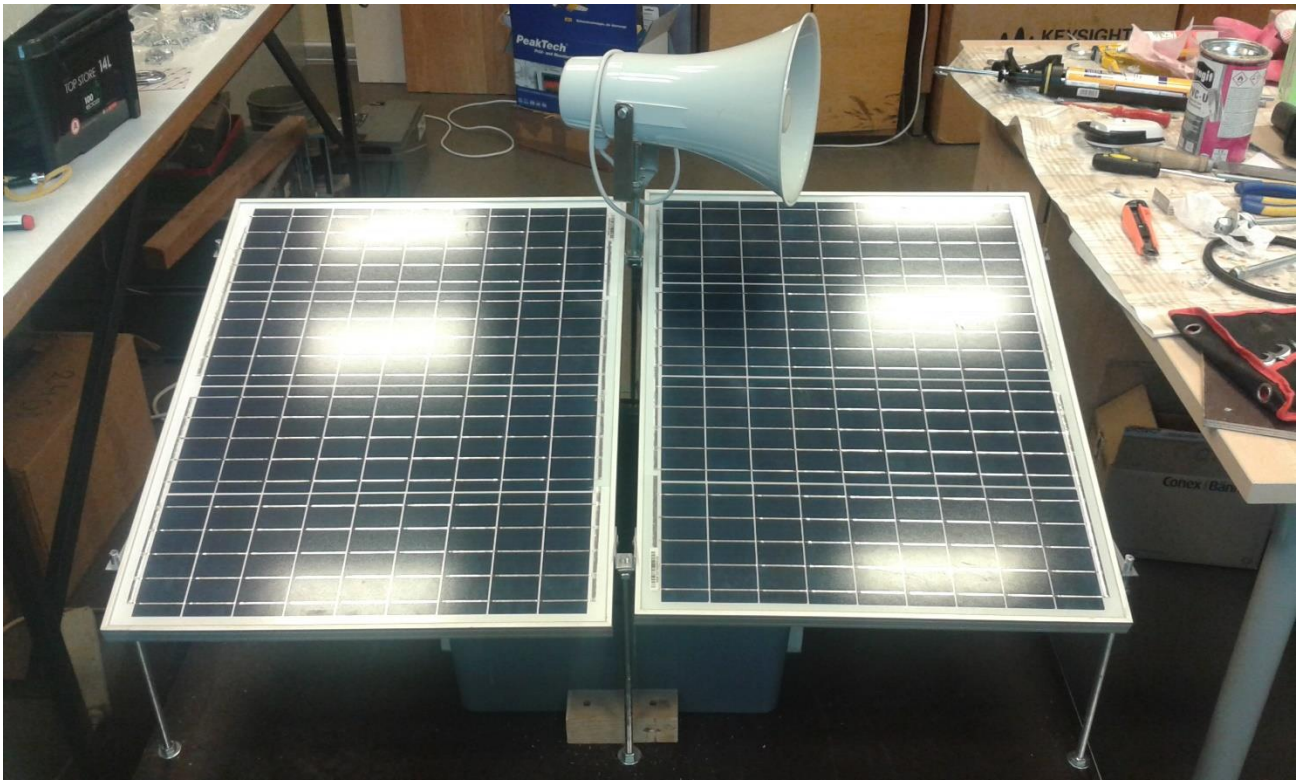
Rūpnieciski tiek izgatavoti samērā lieli plosti kas bieži vien ir domāti dažādu izklaides pasākumu organizācijai. Savukārt HAOS sistēmas realizācijai ir nepieciešams relatīvi neliels plosts ar diezgan zemu celtspēju. Tika izvēlēts risinājums no kanalizācijas caurulēm ar diametru 160 mm un atbilstošiem 90 grādu savienotājiem. Caurules savā starpā tika savienotas veidojot kvadrātu. Caurules tika pildītas ar poliuretāna putām, lai izslēgtu plosta nogrimšanas iespēju gadījumā, ja tiktu bojāts tā korpuss (piemēram, transportēšanas laikā, vai

ekspluatācijas gaitā). Pie caurulēm ar montāžas metāla lenti tika piestiprināta 20 mm bieza saplākšņa plate. Attēlā ir parādīts izveidotais plosts pirmā izmēģinājuma laikā:



Izveidotās konstrukcijas celtspēja izradījās nepietiekoša un tika izlemts kvadrāta iekšpuse zem pamatnes izvietot putuplasta plāksnes un piestiprināt tās ar polimērmateriāla lentes palīdzību. Šāda konstrukcija izturēja uz pamatnes izvietotas aparatūras svaru. Vissmagākais ir 90 Ah svina akumulators (26 kg), kas nodrošina aparatūras nepārtrauktu darbību.

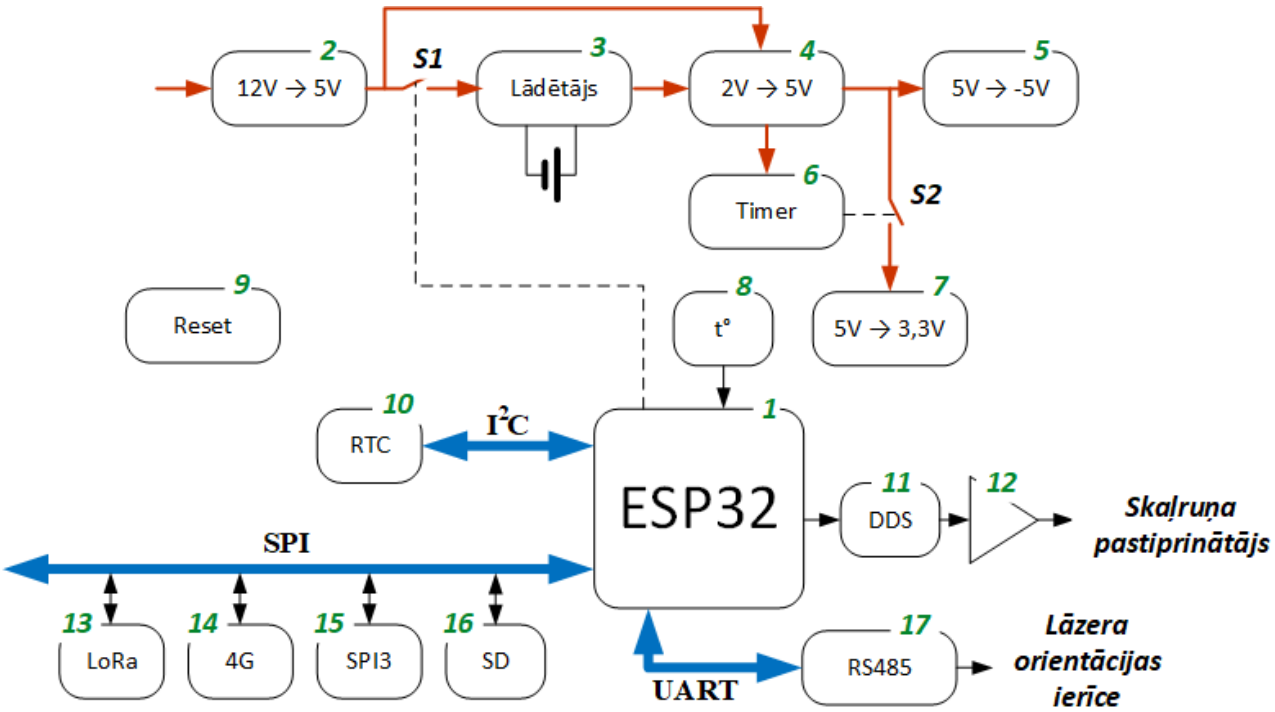
Tālāk pie saplākšņa plāksnes tika piestiprināta kaste akumulatoram un elektronikai, saules paneļi (2x50 W), rupora skaļrunis un zemūdens skaļrunis.


PeakTech

2. U2.2 Spiesto plašu izstrāde, izgatavošana un montāža 10.-13.mēn.

2.1. Sistēmas moduļu pārskats ar nepieciešamību veikt spiesto plašu izstrādi

Rezultējošas ierīces komplektācijas darbietilpības samazināšanai tika pieņemts lēmums izvietot paštaisīto elektroniku, pēc iespējamības, vienā blokā (tālāk – *kontroliera bloks*).



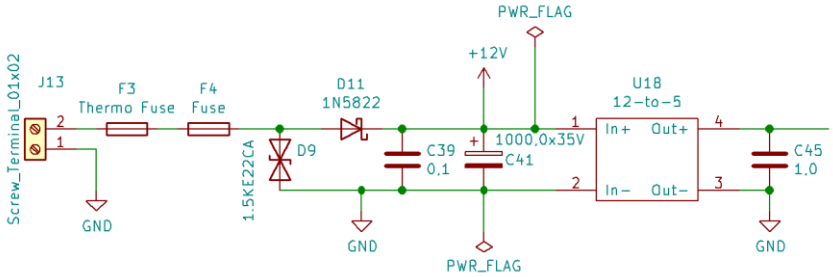
Zīm.2.1. Centrālā bloka struktūras shēma

Barošanas daļa sastāv no mezgliem **2**, **3**, **4**, **5** un **7**. Shēmā tika paredzēta iespēja izmantot rezerves akumulatoru ārējas barošanas pārtraukšanas gadījumā. Ņemot vērā, ka litija akumulatoram ir diezgan stingrs temperatūras režīms (it īpaši lādēšanās procesā), tika izmantots elektroniskais slēdzis **S1** un temperatūras devējs **8**. Akumulators nodrošina tikai 3,7V – tāpēc sprieguma palielināšanai tiek izmantots DC-DC pārveidotājs **4** (**MT3608** pamatā). Tā kā ir prognozēts, ka vienā zivkopības objektā būs izmantoti daži *kontroliera bloki*, darbu sinhronizācijai tiks izmantota LoRa tehnoloģija (mezgls 13). Distancēs vadībai tiks izmantots atsevišķais bloks, kas ar *kontroliera bloku* būs savienots ar mezglu 14 (4G/GSM). Ņemot vērā, ka akustisko signālu masīvi ir visai lieli, ir nepieciešams izmantot SD karti (mezgls 16). Tas arī ļauj viegli mainīt skaņas komplektu un

saglabāt ierīces darbības log-failu. Programmas loģiskas kļūdas gadījumā operatīvai restartēšanai eksistē mezgls 9.

Elementu kopējais skaits un SMD tehnoloģijas pielietošana pieprasa iespiesto plašu izmantošanu. *Kontroliera bloka* pilnā shēmā ir paradīta pielikumā 2.

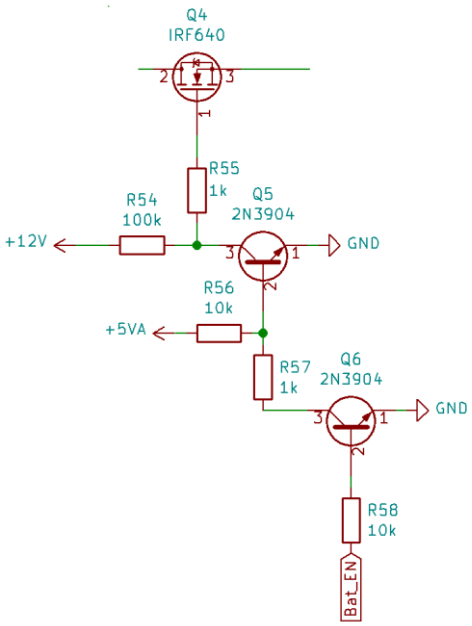
2.1.1. Modulis 2 (12V→5V)- apraksts



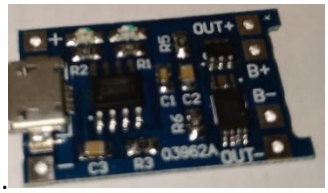
Ņemot vērā, ka viens no barošanas variantiem ir akumulators, tika izmantots sprieguma DC-DC pārveidotājs uz integrālās mikroshēmas **LM2596** pamata.

Drošinātājs **F3** izslēgs shēmu ja apkartēja temperatūra pārsniegs 60°C. Otrais drošinātājs **F4** kopā ar supresoru **D9** pasargās shēmu no sprieguma pārsniegšanas virs **22V**. Šotkija diodes neļaus pieslēgt akumulatoru nepareizā polaritātē.

2.1.2. Modulis 3 (Lādētājs) - apraksts

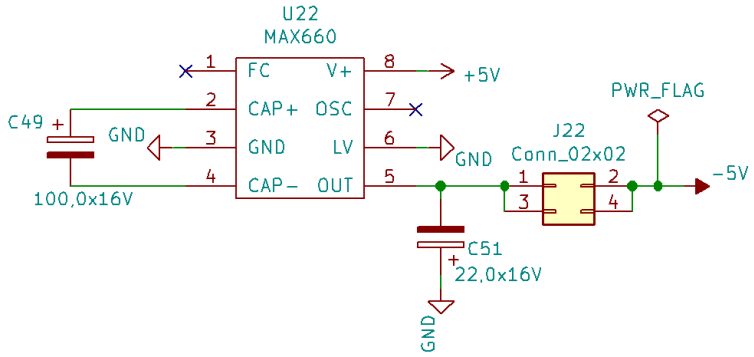


Tika izmantots gatavais lādēšanas modulis uz integrālas mikroshēmas TC4056 pamata. Lai būtu iespēja izmantot arī lētus akumulatorus tika izvēlēta plate ar speciālu iebūvētu aizsardzības shēmu.



Elektroniskais slēdzis tika uzbūvēts uz lauka tranzistora **IRF640** pamata. Tā kā šīs tranzistors droši strādā ar aizvara spriegumu virs **5V**, tika izmantots papildu tranzistors **Q5**. Bet šajā gadījumā shēma būs invertējoša (slēdzis būs atvērts ar vadības loģisko līmeni **0**). Dotas situācijas invertēšanai tika izmantots vēl viens tranzistors **Q6**.

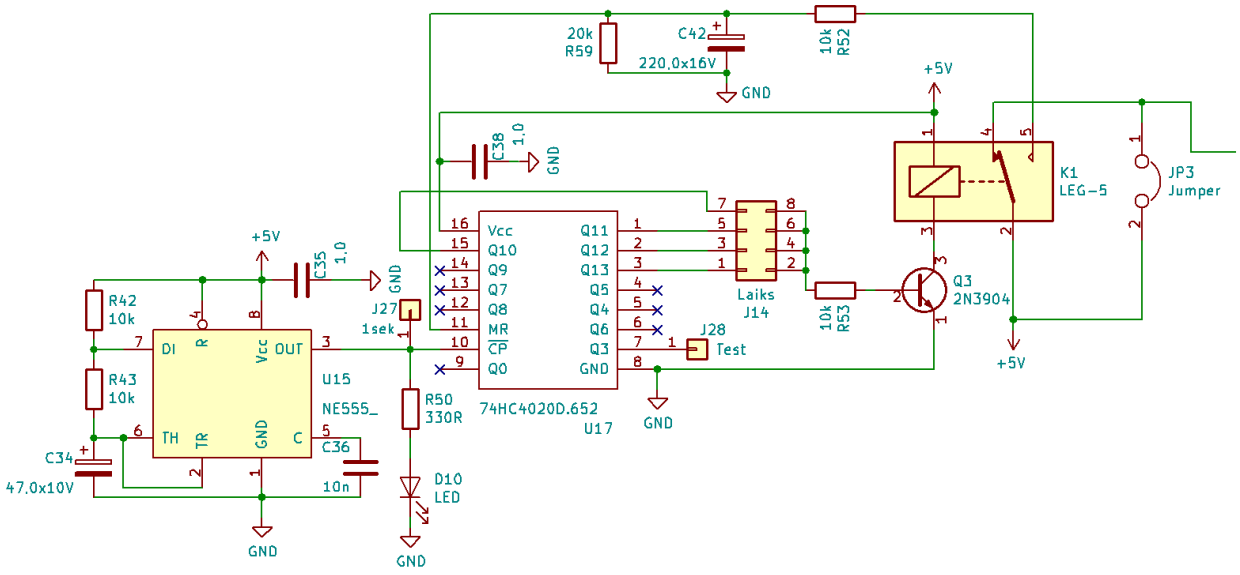
2.1.3. Modulis 5 (+5V→-5V) - apraksts



Akustiskā signāla izejošam pastiprinātajam ir nepieciešama bipolārā barošana. Tam tika izmantots integrālais invertors **MAX660** ar maksimālu slodzes strāvu **100mA**.

2.1.4. Modulis 6 (Timer) - apraksts

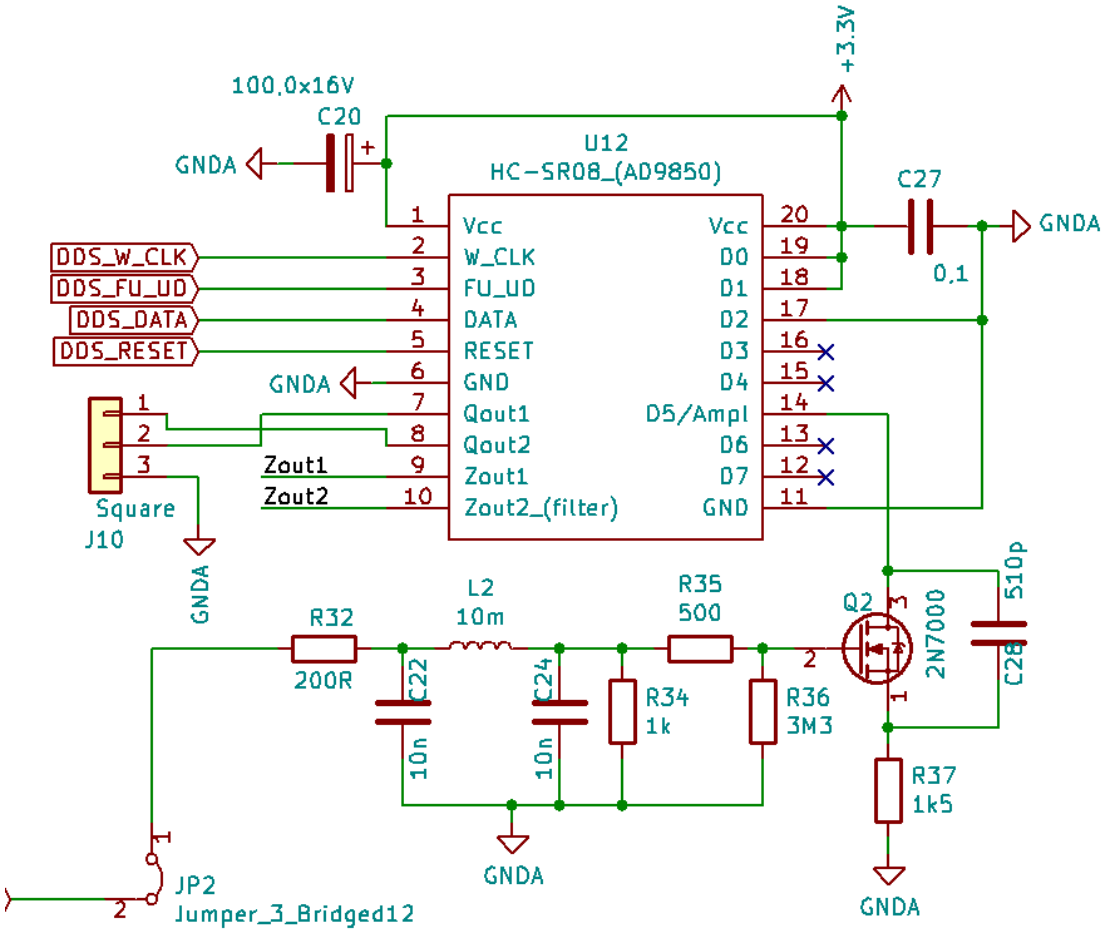
Tā kā ierīce neatrodas vieglas pieejamības attālumā, *kontroliera blokā* arī ir neatkarīgs taimers, kas regulāri restartē visu shēmu izslēdzot tā barošanu. Takts ģenerators (**NE555**) nodrošina impulsus ar frekvenci **1Hz**.



Integrālais skaitītais (**74HC4020**) pēc noteikta laika pārslēgs vienu no savam izejam aktīvā stāvoklī. Kas savukārt caur tranzistoru **Q3** iedarbinās releju **K1**. Relejs neaktīvā stāvoklī baro visa bloka shēmu (izvadi **2** un **4**), bet pēc pārslēgšanas strāva no **2** izvada plūdis uz **5**, nodrošinot skaitītāja restartēšanu. Elementi **R52**, **C42** un **R59** ir vajadzīgi, lai palielinātu restartēšanas intervālu. Slēdzis **J14** ļauj izvelēties laika intervālu no četriem variantiem (15, 30 minūtes un 1, 2 stundas).

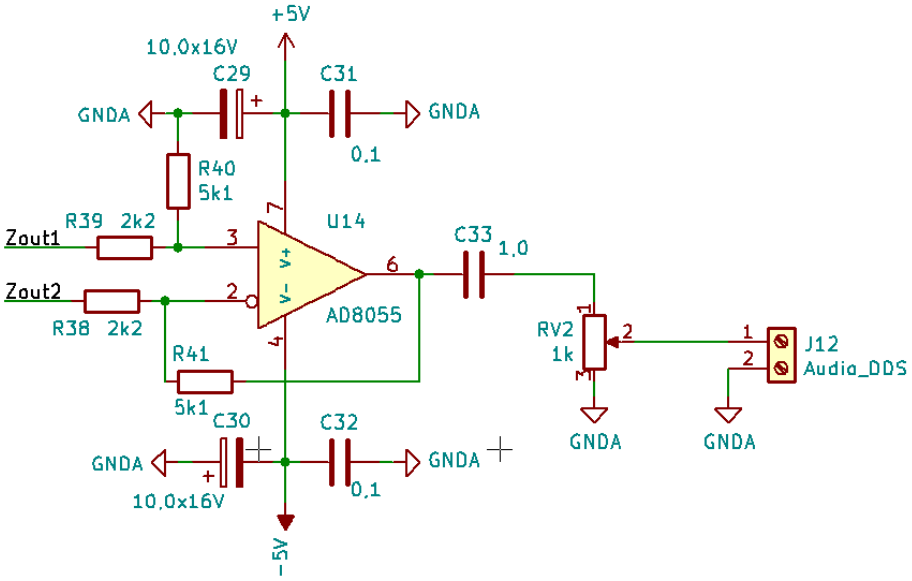
2.1.5. Modulis 11 (DDS) - apraksts

Akustisko signālu ģenerēšanai tiek izmantots modulis ar DDS integrālu mikroshēmu **AD9850**. Pēc noklusējuma šis modulis ļauj mainīt tikai izejas signāla frekvenci. Lai būtu iespēja mainīt arī amplitūdu, shēma tika papildināta ar tranzistoru **Q2**. Aizvara spriegumu ģenerē mikrokotrolieris **ESP32** izmantojot IPM (impulsu platuma modulācija). Līdz ar to, lai izfiltrētu no rezultējoša signāla parazītiskas pulsācijas, tika pielietots filtrs no elementiem **R32**, **C22**, **L2**, **C24** un **R34**.

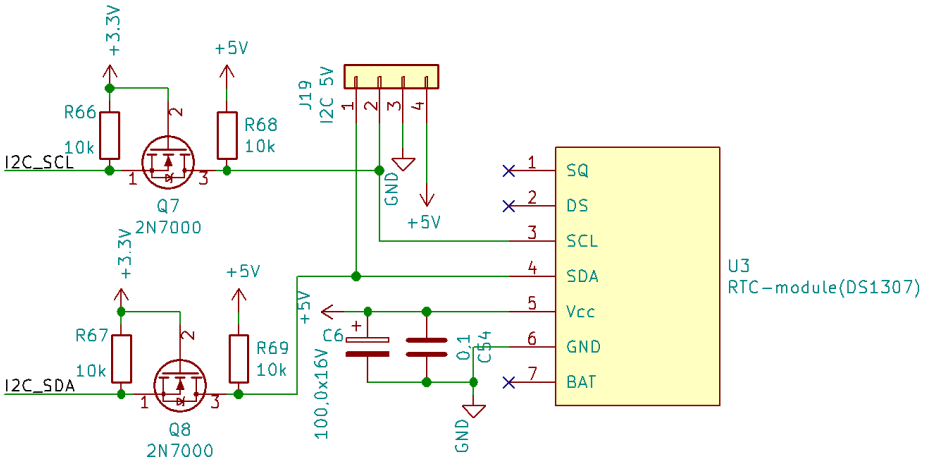


2.1.6. odulis 12 (izejoša signāla pastiprinātājs) - apraksts

Pretestības salāgošanai tika izmantota integrāla shēma AD8055. Operacionālais pastiprinātājs ir ieslēgts diferenciālā slēgumā, kas nodrošina palielinātu lineāritāti. Potenciometrs RV2 ir vajadzīgs izejas signāla vidējās vērtības regulēšanai.



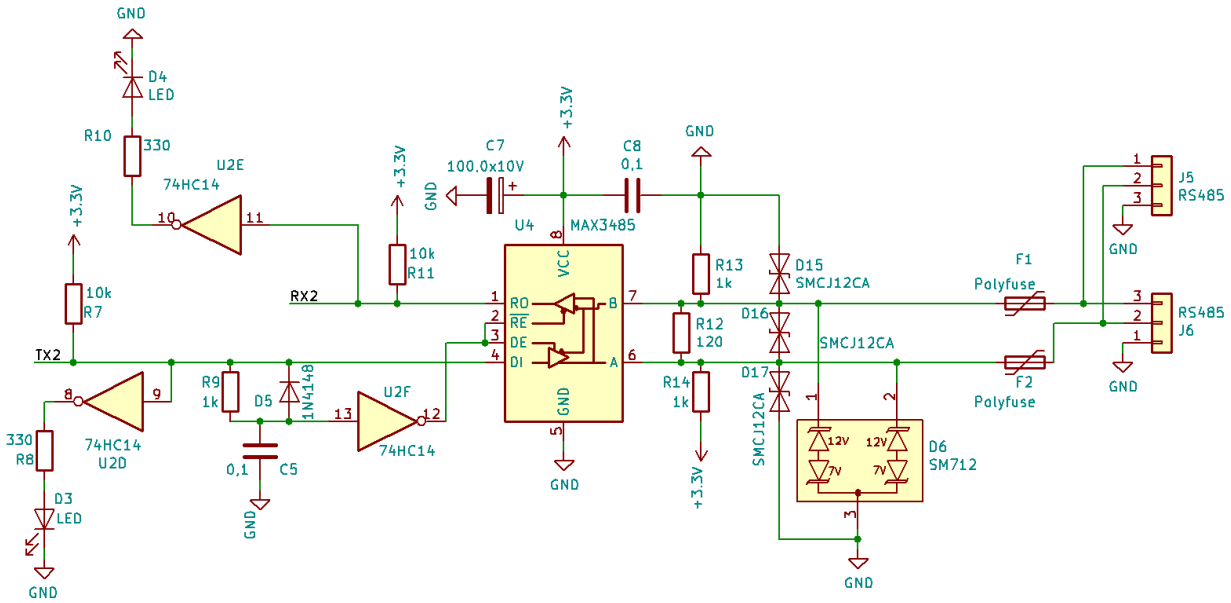
2.1.7. Modulis 10 (RTC) - apraksts



Mikrokontroliera programmā tiek aktīvi izmantoti laika parameti. Iekšēji tie tika nodrošināti ar iebūvētu taimeri. Bet ir jāzina arī konkrēta stundu diennaktī. Lai nodrošinātu tādas funkcijas, tika izmantots neatkarīgs reāla laika modulis (**RTC** – Real Time Clock) ar integrālo mikroshēmu **DS1307**. Ar ESP32 modulis savienots izmantojot **I²C** kopni. Ņemot vērā, ka dota RTC darba spriegums ir **5V**, shēmā tika izveidots loģisko līmeņu pāreidotājs: **R66**, **Q7**, **R68** (**R67**, **Q8**, **R69**).

2.1.8. Modulis 17 (RS485) - apraksts

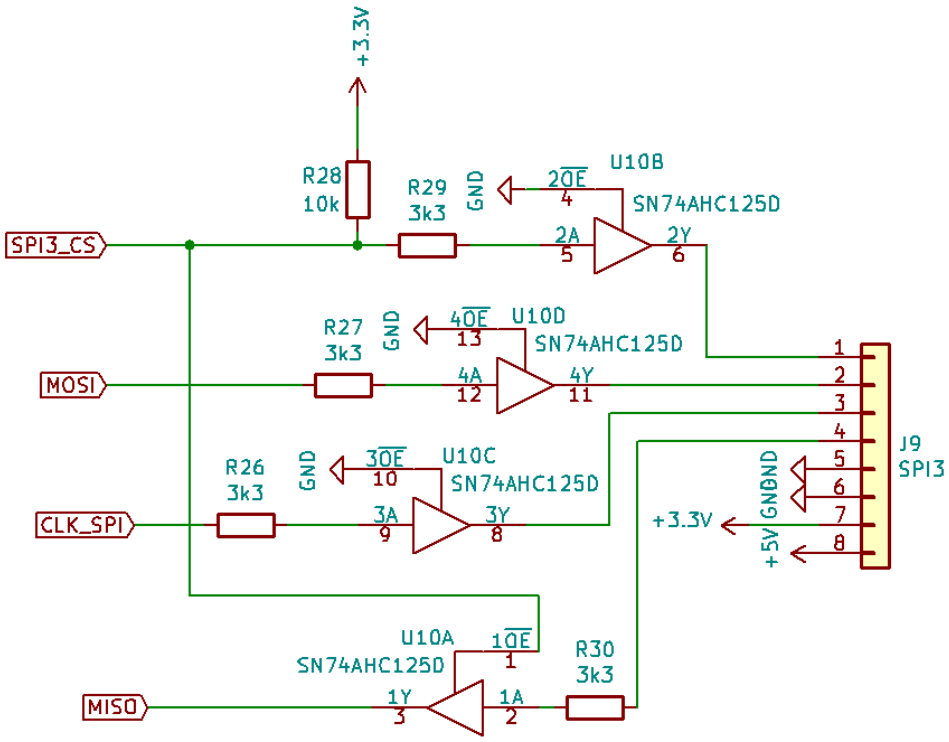
Lāzera orientēšanas sistēmas **YO3040** vadībai tiek izmantots standarts **RS485**. No standarta UART tas atšķiras ar sakaru kanāla konstrukciju. Programmatiskā konvertācija nav vajadzīga. Divu standartu salāgošanai tika



izmantota mikroshēma MAX3485. Shēmā ir elektriskā aizsardzība divos variants. Var izmantot trīs elementus **D15**, **D16** un **D17**, jeb vienu **D6**. Drošinātāji **F1** un **F2** ir nepieciešami abos gadījumos. Loģiskais elements ir vajadzīgs mikroshēmas **U4** režīmu pārslēgšanai. Elementi **R9**, **D5** un **C5** nodrošina laika aizkavējumu. Rezistori **R7**, **R11**, **R13**, **R12** un **R14** nodrošina attiecīgus spriegumu līmeņus.

2.1.9. Moduļi 13-16 - apraksts

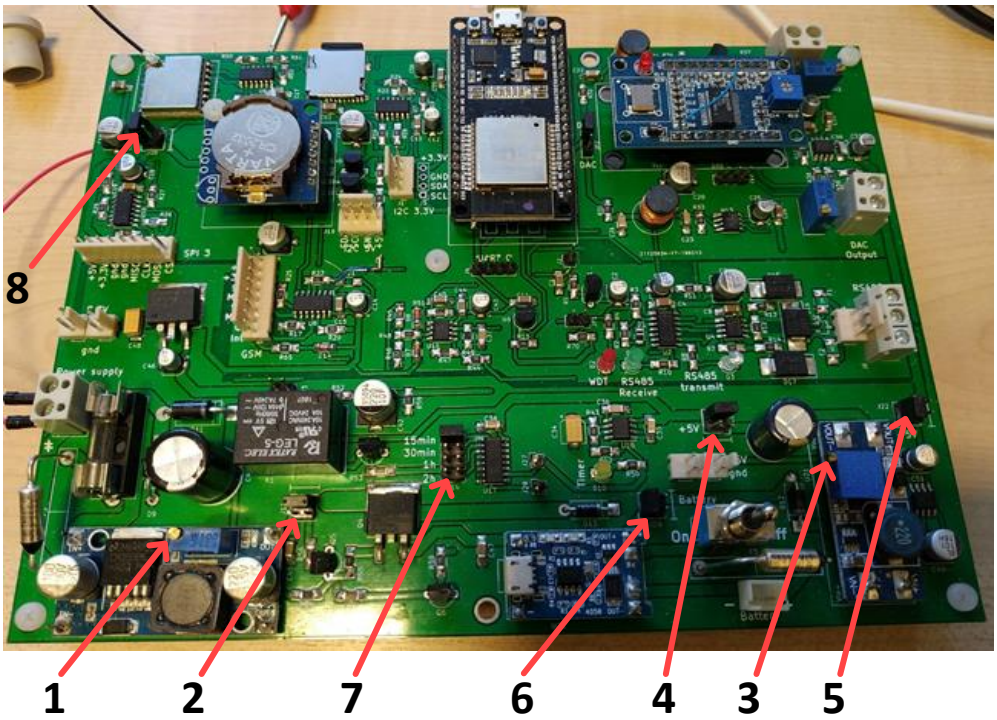
Shēmā plaši izmantota **SPI** kopne. Lai vairāki moduļi varētu droši strādāt vienā kopnē, tika pielietota buferizācija izmantojot integrālo mikroshēmu **74AHC125**. Darbības princips paradīts moduļa 15 (SPI3) piemērā. Līnijas **MOSI** un **CLK** vienmēr atrodas gatavības stāvoklī – tas netraucēs pārējiem moduļiem. Tomēr, kamēr mikrokontrolieris strādās ar citiem abonentiem, loģiskā elementa **U10A** izejai ir jābūt augstās impedances stāvoklī. Rezistors **R28** nav obligāts, ņemot vēra, ka mikrokontroliera programmas darbības sakumā visas **CS** līnijas uzstādīs loģiskā 1 (bet **R21** neļaus *kontroliera blokam* strādāt ESP32 īpašības dēļ).


+3.3V
R28
10k
R29
3k3
GND
2OE
4
U10B
SN74AHC125D
SPI3_CS
2A
5
2Y
6
R27
3k3
GND
4OE
13
U10D
SN74AHC125D
MOSI
4A
12
4Y
11
R26
3k3
GND
3OE
10
U10C
SN74AHC125D
CLK_SPI
3A
9
3Y
8
U10A
SN74AHC125D
1OE
1
R30
3k3
MISO
1Y
3
1A
2
+3.3V
GND
+5V
J9
SPI3
1
2
3
4
5
6
7
8

2.2. AP2.2.1 Izstrādātās spiestās plates

Projektēšanai tika izmantota programma KiCAD. Izgatavošanai tika izvelēta divslāņu plate, kā optimālais cena/iespējas variants. Praktiski visi elementi novietoti vienā slānī. Gatavi moduli izvietoti "otrā stāvā" un vieta zem tas arī izmantota elektronisko elementu izvietošanai. Slāņu fotošabloni un zīda slānis ir paradīti pielikuma 3.

2.3. AP2.2.2 Gatavās spiestās plates



Kontroliera bloka sākotnējas palaišanas procedūra:

1. Pēc ārējas barošanas pieslēgšanas, griežot rezistoru, dabūt DC-DC pārveidotāja izejā spriegumu 5V;
2. Pēc tam, izmantojot savienotājus, var pieslēgt nākamu shēmas daļu;
3. Griežot rezistoru dabūt DC-DC pārveidotāja izejā spriegumu 5V;
4. Pieslēgt pāreju shēmu pie 5V;
5. Pieslēgt pastiprinātāju pie -5V;
6. Savienotājus ir jēga izmantot tikai ja kopā ar plati tiks izmantots rezerves barošanas akumulators;
7. Ar savienotāju vajag izvelēties attiecīgo restartēšanas laiku; ja neviens kontaktu pāris nav savienots, tad šīs mezgls nestrādās un restartēšanas procedūra netiks izmantota;
8. Nav rekomendēta barošanas pieslēgšana kamēr pie LoRa moduļa nav pievienota antena.

2.4. Komunikāciju moduļa uzbūve un darbības principi

2.4.1. Vispārīgie principi

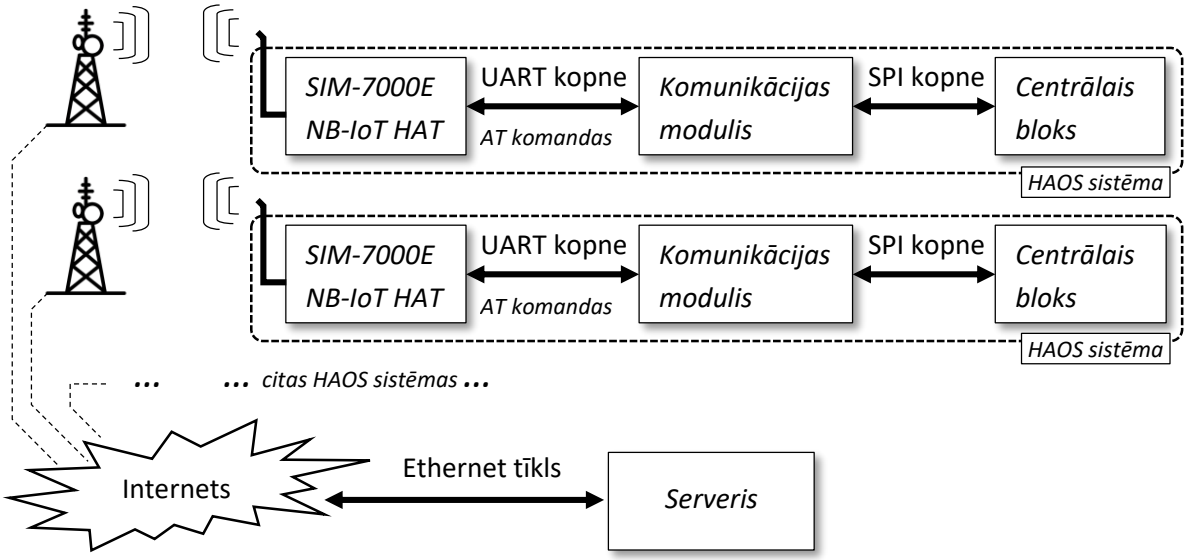
Lai attālināti monitorētu uzstādītas HAOS sistēmas ir izveidoti speciālais Komunikācijas modulis un datu savākšanas / vadības serveris (Serveris).

Komunikācijai par pamatu tika izvēlēts mobilo operatoru tīkls. Šādai izvēlei ir virkne priekšrocību:

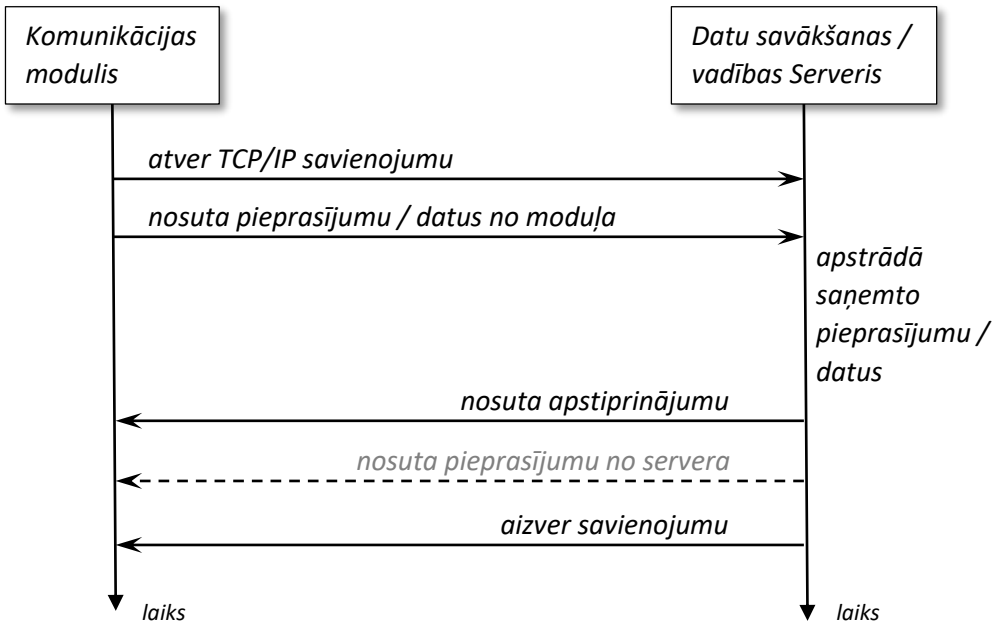
1. Šīs risinājums strādās visur kur ir mobilā tīkla pārklājums. Populārāko mobilo tīklu operatoriem (LMT / Tele2 / Bite) tā ir gandrīz visa Latvija teritorija.
2. Mazam sistēmu skaitam uzturēšanas maksa par komunikāciju būs neliela, ja izvēlas piemērotāku mobila interneta tarifu. Ja ir nepieciešams, 3G/4G tīklos varēs pārsūtīt arī lielus datu apjomus – skaņas / lāzera vadības programmas.
3. Ir pieejami dažādi gatavie risinājumi kas realizēs zemā līmeņa komunikāciju ar mobilo tīklu. Darbs ar šādiem moduļiem notiek pēc vienkārša protokola kas ir realizējams mikrokontrolierī.

Projektā tika izmantots SIM-7000E NB-IoT Hat modulis kas atbalsta GSM/GPRS, EDGE un pat 4G tīklus, bet no vadības kontroliera puses realizē dažādus protokolus, piemēram, TCP, HTTP, u.c. Modulis ļauj sūtīt/saņemt īsziņas un atbalsta GNSS pozicionēšanu.

Taču šīs tīkls pieprasa noteiktu sistēmas arhitektūru ar vairākiem Komunikācijas moduļiem un vienu centrālo Serveri.



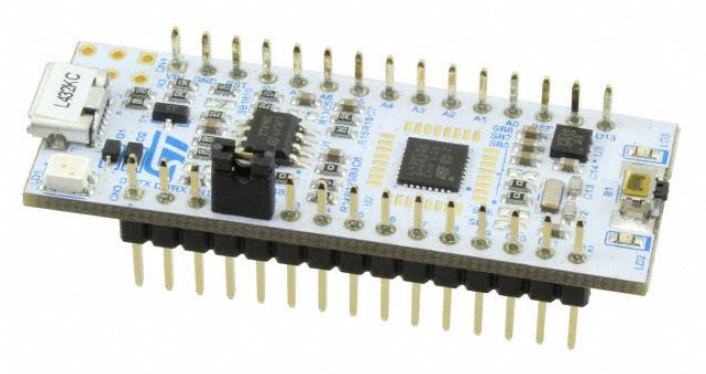
Arī komunikācijas protokolam tiek uzlikti sekojošie ierobežojumi – tā kā tikai Serverim ir iespējams nodrošināt pastāvīgu (fiksētu) reālo IP adresi, tad jebkura transakcija starp Komunikācijas moduli un Serveri norisinās no moduļa puses:



Serveris apstiprinājumu nosuta vienmēr, bet papildus pieprasījumu tikai tad, ja tāds ir jau sagatavots. Izmantojot tikai Interneta rīkus bez savienojuma ko atver Komunikācijas modulis nav iespējams nosūtīt

pieprasījumu no servera puses. Līdz ar to, Komunikācijas modulim regulāri jāslēdzas pie Servera ar tukšajiem pieprasījumiem arī tad ja nav ko sūtīt. Šos pieprasījumus sauc par Heartbeat.
Kā papildus komunikācijas kanāls var kalpot īsziņas, ar kurām var apmainīties jebkurā brīdī.

Par komunikācijas moduļa centrālo mezglu tika izmantots mikrokontroliera STM32L432KC izstrādes rīks NUCLEO-L432KC:



Mikrokontroliera atmiņas ierobežojuma dēļ (64 KiB) pieprasījumu maksimālais garums tika ierobežots līdz 4 KiB. Ja ir nepieciešams nosūtīt lielāku datu apjomu (piem. skaņas failu), datu paka jāsadala īsākās un jānosūta pa daļām.

3.1.2. Komunikācijas moduļa ziņojumi

3.1.2.1. Heartbeat (HB) ziņas

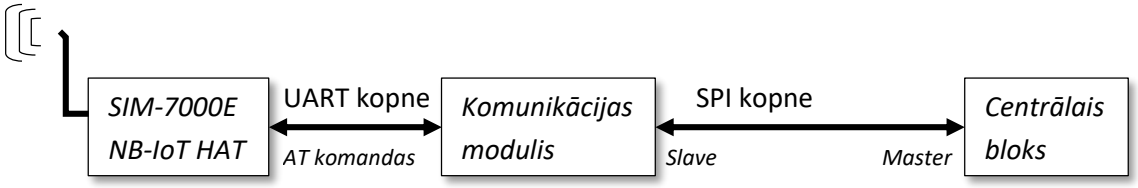
Galvenie uzdevumi ir:

1. regulāri paziņot Serverim par moduļa normālu darbību,
2. nodrošināt Serverim iespēju vismaz ar tādu regularitāti veikt pieprasījumus.

HB ziņu intervāls tika izvēlēts 1x reizi stundā, lai nenoslogotu tīklu un Serveri.

3.1.2.2. Diagnostikas (TD) ziņas

Galvenais uzdevums ir apkopot informāciju par Centrālā bloka darbības parametriem / stāvokli un regulāri ziņot Serverim.



Diagnostikas informāciju regulāri (reizē 15 minūtēs) apkopo Centrālais bloks un pa SPI kopni nodod Komunikācijas modulim, kas to noformē TD ziņas formā. Diagnostikas dati sevī iekļauj sekojošas vērtības:

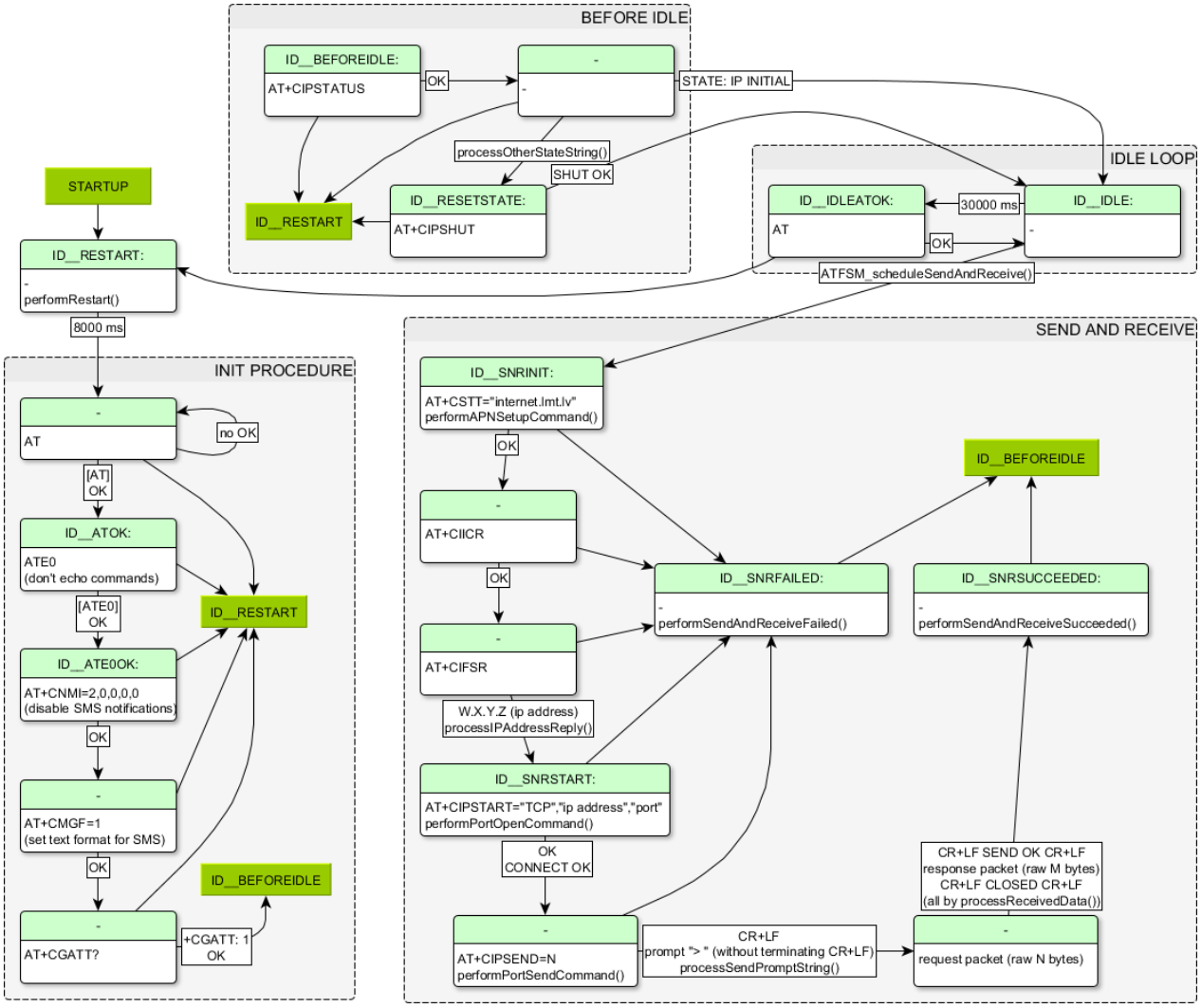
- Galvenais (ārējā avota) spriegums,
- Kontroliera rezerves akumulatora spriegums,
- Kontroliera rezerves akumulatora statuss (vai ir ieslēgta uzlādēšana),
- Temperatūra,
- Skaļruņa statuss,
- Lāzera statuss.

Serverī šie dati tiek saglabāti Log failā, lai vēlāk varētu tos analizēt. Ja stundas laikā no Centrālā bloka nenāk diagnostikas dati, Komunikācijas modulis noformulē TD ziņu ar saturu "IDLETIMEOUT".

3.1.3. Implementācija

Komunikācijas moduļa galvenais mezgls ir STM32L432KC mikrokontrolieris. Tā uzdevums ir klausīties divas kopnes – UART (no SIM-7000E moduļa) un SPI (no Centrālā bloka). Mikrokontrolierī ir implementēti sekojošie moduļi:

1. ***spi_processor*** (darbam ar SPI kopni – diagnostikas informācijas saņemšanai un TD ziņu noformēšanai Serverim),
2. ***atfsm*** (darbam ar SIM-7000E izmantojot moduļa AT komandas – realizē vienkāršu stāvokļa automātu viena pieprasījuma pilnai nodošanai Serverim un atbildes saņemšanai atpakaļ),
3. ***connection_manager*** (pieprasījumu rindas realizācijai, ziņu pareizai paketēšanai, Servera atbilžu apstrādei, HB ziņu regulārai ģenerēšanai).



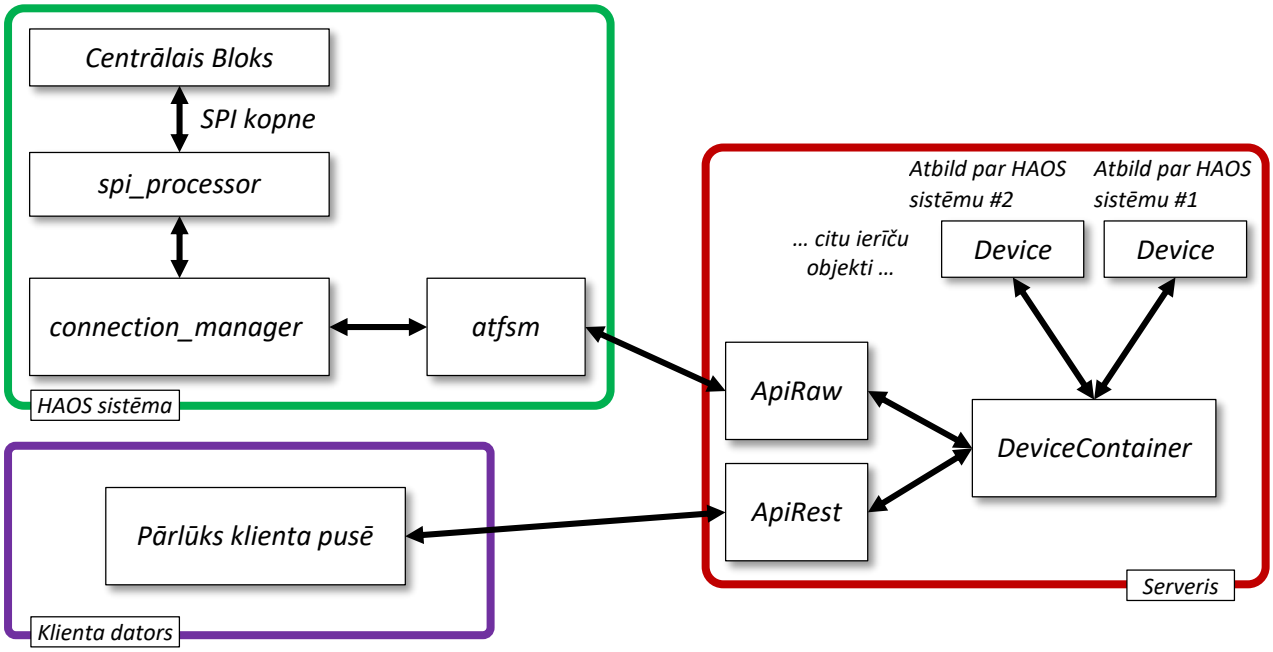
Serveris ir veidots uz Raspberry Pi pamata, operētājsistēma ir Raspbian Linux. Servera programmatūra uzrakstīta Python 3 valodā izmantojot ***web.py*** pakotni un Linux sockets. Palaižot servera programmatūru automātiski tiek palaisti divi pavedieni:

1. ***ApiRaw*** – klausās TCP/IP soketu un apstrādā pieprasījumus no Komunikācijas moduļiem. Šādiem pieprasījumiem ir vienkārša struktūra (nav sarežģītā formāta galvas, ir nelieli izmēri līdz 4 KiB) lai atvieglotu Komunikācijas moduļa mikrokontrolierim saziņu,
2. ***ApiRest*** – klausās citu TCP/IP soketu un izmantojot **web.py** pakotnes rīkus apstrādā REST-tipa pieprasījumus, kurus tipiski ģenerē mājaslapas.

Serverī palaists arī ***nginx*** REST pieprasījumu pāradresācijai – visi pieprasījumi kas attiecās pret Servera programmatūru tiek pāradresēti, bet tie kas neattiecās (piemēram, statiskā mājaslapas informācija) tiek apstrādāti ar pašu ***nginx***. Tādā veidā pie Servera var pieslēgties arī no parastā pārlūka.

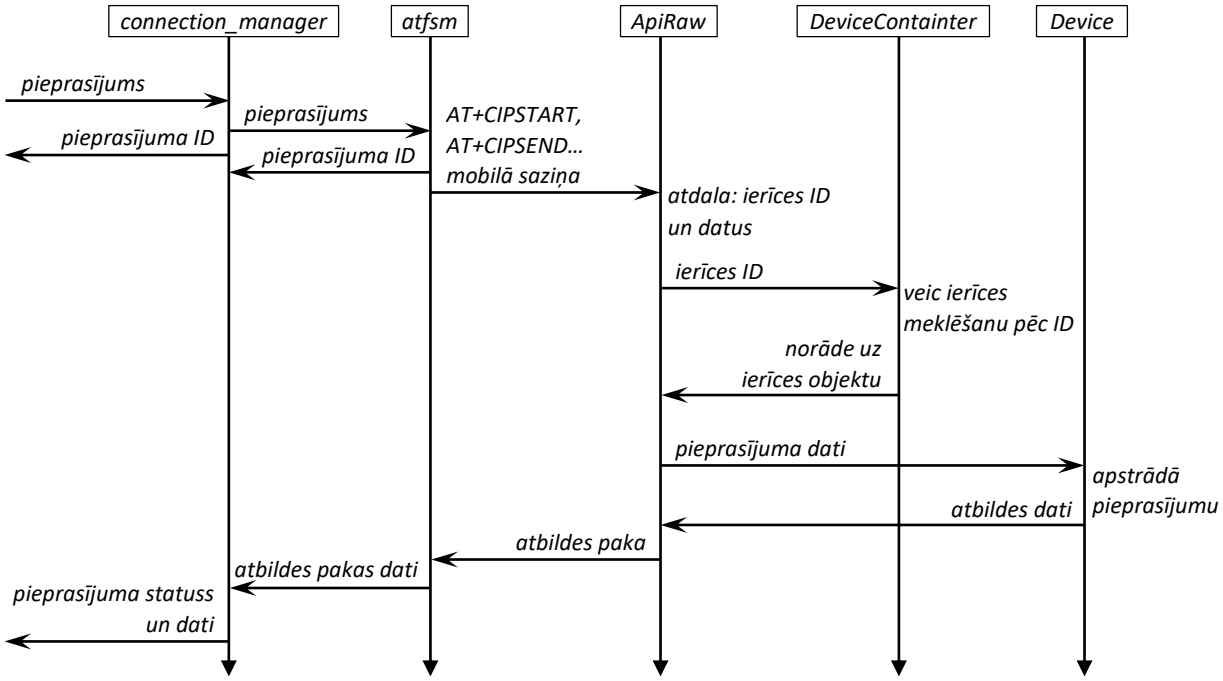
Pieprasījumi tiek izpildīti sekojoši: katrai reģistrētai HAOS sistēmai Servera pusē tiek izveidots objekts ***Device*** (ierīce). Modulis ***DeviceContainer*** glabā masīvu ar ierīcēm un pēc vajadzības tos atrod. Kad Serverim pienāk pieprasījums, ApiRaw / ApiRest moduļi identificē ierīci, tad atrod vajadzīgo objektu, un tad tam nodod pieprasījumu izpildei.

Kopumā komunikācijas sistēmas arhitektūru var atspoguļot sekojoši:



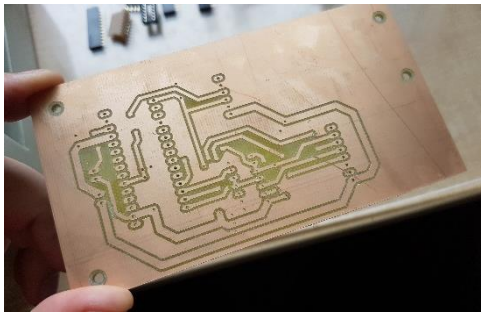
3.1.4. Pieprasījuma apstrādes piemērs

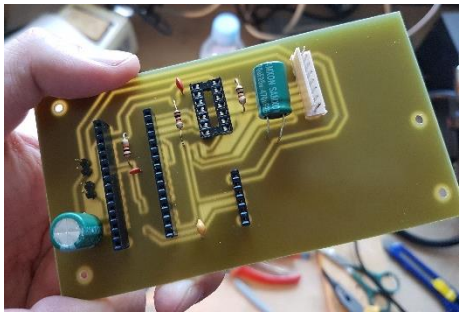
Diagramma demonstrē: kuri sistēmas moduļi, kādā secībā un kuru darbu veic transakcijas laikā.



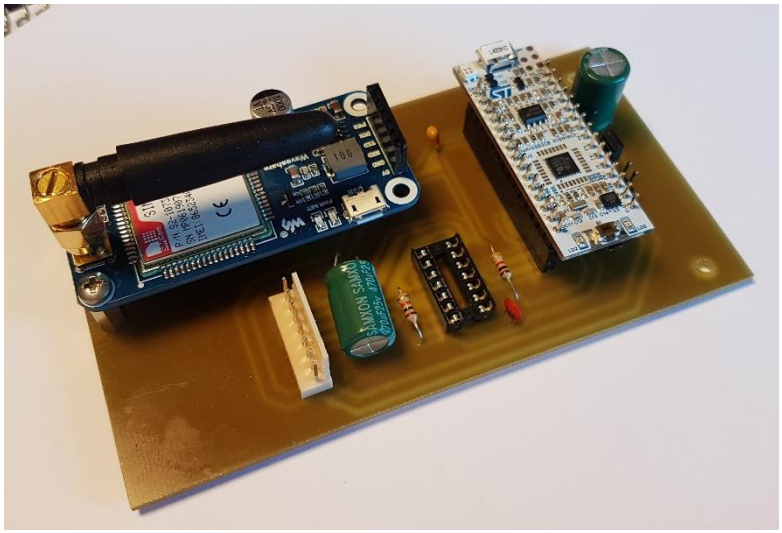
3.1.5. Prototipi

Komunikācijas modulim tika izveidota sekojoša iespiedplate:

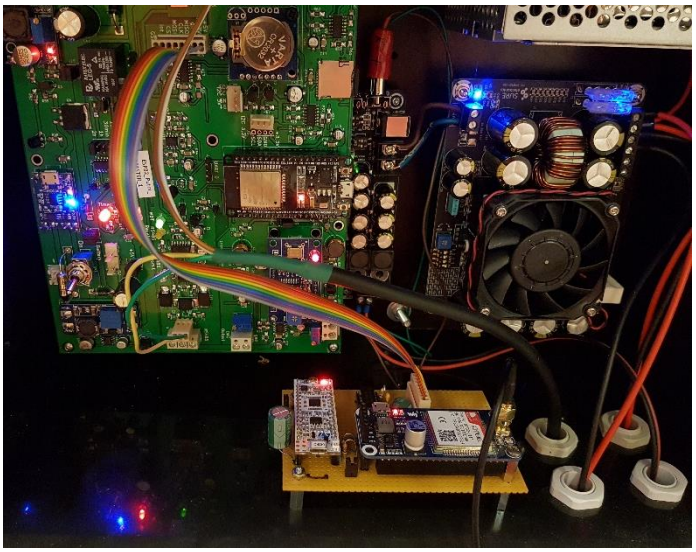




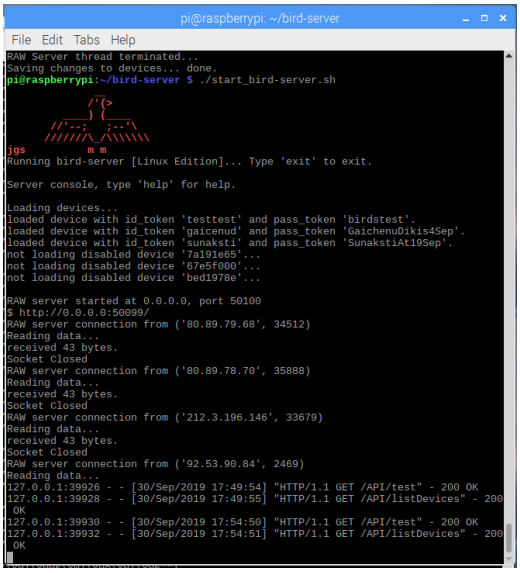
Un izveidots sekojošais prototips:



Komunikācijas modulis (maketplates versija) kopā ar Centrālo bloku:



Servera programmatūras logs:



Ierakstīto Log datu piemērs (objekts - Gaičēnu dīķis) JSON formātā:


```
…
{"date": "2019-09-20 07-41-47", "type": "HB"},
{"date": "2019-09-20 07-46-42", "type": "TD", "content": "b"\xfe\x07\x01w'\x01\x0f\x01\x06""},
{"date": "2019-09-20 08-01-52", "type": "TD", "content": "b'\xfe\x07\x01w&\x01\x0f\x01\x06'"},
{"date": "2019-09-20 08-16-52", "type": "TD", "content": "b"\xfe\x07\x01w'\x01\x0f\x01\x00""},
{"date": "2019-09-20 08-31-57", "type": "TD", "content": "b"\xfe\x07\x01w'\x01\x0f\x01\x08""},
{"date": "2019-09-20 08-43-53", "type": "HB"},
{"date": "2019-09-20 08-46-57", "type": "TD", "content": "b"\xfe\x07\x01v'\x01\x10\x01\x07""},
{"date": "2019-09-20 09-02-00", "type": "TD", "content": "b"\xfe\x07\x01w'\x01\x10\x01\x06""},
{"date": "2019-09-20 09-26-34", "type": "TD", "content": "b"\xfe\x07\x01w'\x01\x10\x01\x08""},
{"date": "2019-09-20 09-41-35", "type": "TD", "content": "b"\xfe\x07\x01w'\x01\x11\x01\x06""},
{"date": "2019-09-20 09-46-00", "type": "HB"},
…
```


Šīs ieraksts atbilst normālam darba režīmam (HB ziņas reizi stundā, TD ziņas reizi 15 minūtēs). TD ziņu saturs ir iekodēts pēc Python 3 *bytes* datu tipa pieraksta.

Zemāk ir parādīts piemērs Log ierakstam kad izzuda ārējā avota spriegums (ar zaļo atzīmēts pēdējais ieraksts kad bija, ar sarkano – pirmais ieraksts kad vairs nebija):


```
…
{"date": "2019-09-20 14-24-00", "type": "TD", "content": "b"\xfe\x07\x01x'\x01\x1e\x01\x06""},
{"date": "2019-09-20 14-39-05", "type": "TD", "content": "b"\xfe\x07\x01w'\x01\x1e\x01\x06""},
{"date": "2019-09-20 14-54-10", "type": "TD", "content": "b"\xfe\x07\x01w'\x01\x1e\x01\x00""},
{"date": "2019-09-20 14-56-35", "type": "HB"},
{"date": "2019-09-20 15-09-10", "type": "TD", "content": "b"\xfe\x07\x01w'\x01\x1e\x01\x08""},
{"date": "2019-09-20 15-24-14", "type": "TD", "content": "b"\xfe\x07\x01\x00'\x01\x1a\x01\x08""},
{"date": "2019-09-20 15-39-16", "type": "TD", "content": "b"\xfe\x07\x01\x00'\x01\x16\x01\x06""},
{"date": "2019-09-20 15-54-35", "type": "TD", "content": "b"\xfe\x07\x01\x00'\x01\x15\x01\x06""},
{"date": "2019-09-20 15-58-42", "type": "HB"},
{"date": "2019-09-20 16-09-35", "type": "TD", "content": "b"\xfe\x07\x01\x00'\x01\x14\x01\x00""},
{"date": "2019-09-20 16-24-38", "type": "TD", "content": "b"\xfe\x07\x01\x00'\x01\x11\x01\x08""},
…
```

Ir piemērs kad atslēdzoties barošanas spriegumam uz pietiekami ilgu laiku Centrālais bloks pārstāja sūtīt diagnostikas informāciju. Te ar sarkano atzīmēti ieraksti kad nebija barošanas spriegums, ar zaļo atzīmēts brīdis kad ārējais barošanas spriegums atjaunojas un Centrālais bloks turpināja ar diagnostikas ziņām:

…
{"date": "2019-09-20 22-51-51", "type": "TD", "content": "b'\xfe\x07\x01\x00&\x01\x05\x01\x06'"},
{"date": "2019-09-20 23-06-50", "type": "TD", "content": "b'\xfe\x07\x01\x00&\x01\x06\x01\x06'"},
{"date": "2019-09-23 16-03-11", "type": "HB"},
{"date": "2019-09-23 17-02-53", "type": "TD", "content": "b'IDLETIMEOUT'"},
{"date": "2019-09-23 17-03-22", "type": "HB"},
{"date": "2019-09-23 18-02-57", "type": "TD", "content": "b'IDLETIMEOUT'"},
{"date": "2019-09-23 18-03-33", "type": "HB"},
{"date": "2019-09-23 19-02-57", "type": "TD", "content": "b'IDLETIMEOUT'"},
{"date": "2019-09-23 19-03-44", "type": "HB"},
{"date": "2019-09-23 20-03-00", "type": "TD", "content": "b'IDLETIMEOUT'"},
{"date": "2019-09-23 20-03-55", "type": "HB"},
{"date": "2019-09-23 21-01-40", "type": "TD", "content": "b"\xfe\x07\x01\x01'\x01\x11\x01\x08""},
…

3. U2.3 Iekārtas prototipa salikšana un laboratorijas testēšana (RTU) 13.-15.mēn.

3.1. Alternatīvās enerģijas komponentes un to integrēšana sistēmā

Saules paneļi

Ļoti mazam dīķu skaitam tuvumā atrodas pieslēguma vieta maiņstrāvas tīklam. Tāpēc, lai nodrošinātu moduļu autonomo darbību tika pieņemts lēmums izmantot saules paneļus. Iepirkuma procedūras rezultāta tika iegādāti 3 polikristāliskie saules paneļi: divi paneļi, kas nodrošina 50 W un viens 100 W.

Saules paneļu ražotājs: VICTRON ENERGY.

Ražotāja sniegtās saules paneļu specifikācijas pie apgaismojuma intensitātes 1000 W/m^2, 25 ^{o}C:

Kods	Gabarīti, mm	Svars, kg	Nominālā jauda, W	Spriegums pie maks. jaudas, V	Strāva pie maks. jaudas, A	Tukšgaitas spriegums, V	Īsslēguma strāva, A	Darba temperatūras, ^{o}C
SPP03050 1200	540×670×25	4,3	50	18	2.78	22.2	3.09	-40…+85
SPP03100 1200	1000×670×35	8,9	100	18	5,56	21.6	6.32	-40…+85

Pārbaudes laikā tika konstatēts, ka saules paneļi nedarbojas iekštelpās pat pie intensīva apgaismojuma. Tika veikti daži testi uz ielas, lai varētu pārliecināties par saules paneļu pareizo darbību:

Testu laikā tika noskaidrots, ka pie esošas apgaismojuma intensitātes no 50 W saules paneļa varēja dabūt aptuveni 23 W. Turklāt, kad sauli aizsedz tumšie mākoņi jauda krītas līdz nullei. Tāpēc, lai varētu nodrošināt akumulatora uzlādi un aparatūras barošanu tika izlemts savienot divus 50 W paneļus virknē un iegūt lielāku jaudu.

Dziļas izlādes svina akumulatori

Iepirkuma rezultātā tika iegādāti trīs 12 V dziļas izlādes svina akumulatori ar gēla elektrolītu.

Akumulatoru ražotājs: VICTRON ENERGY.

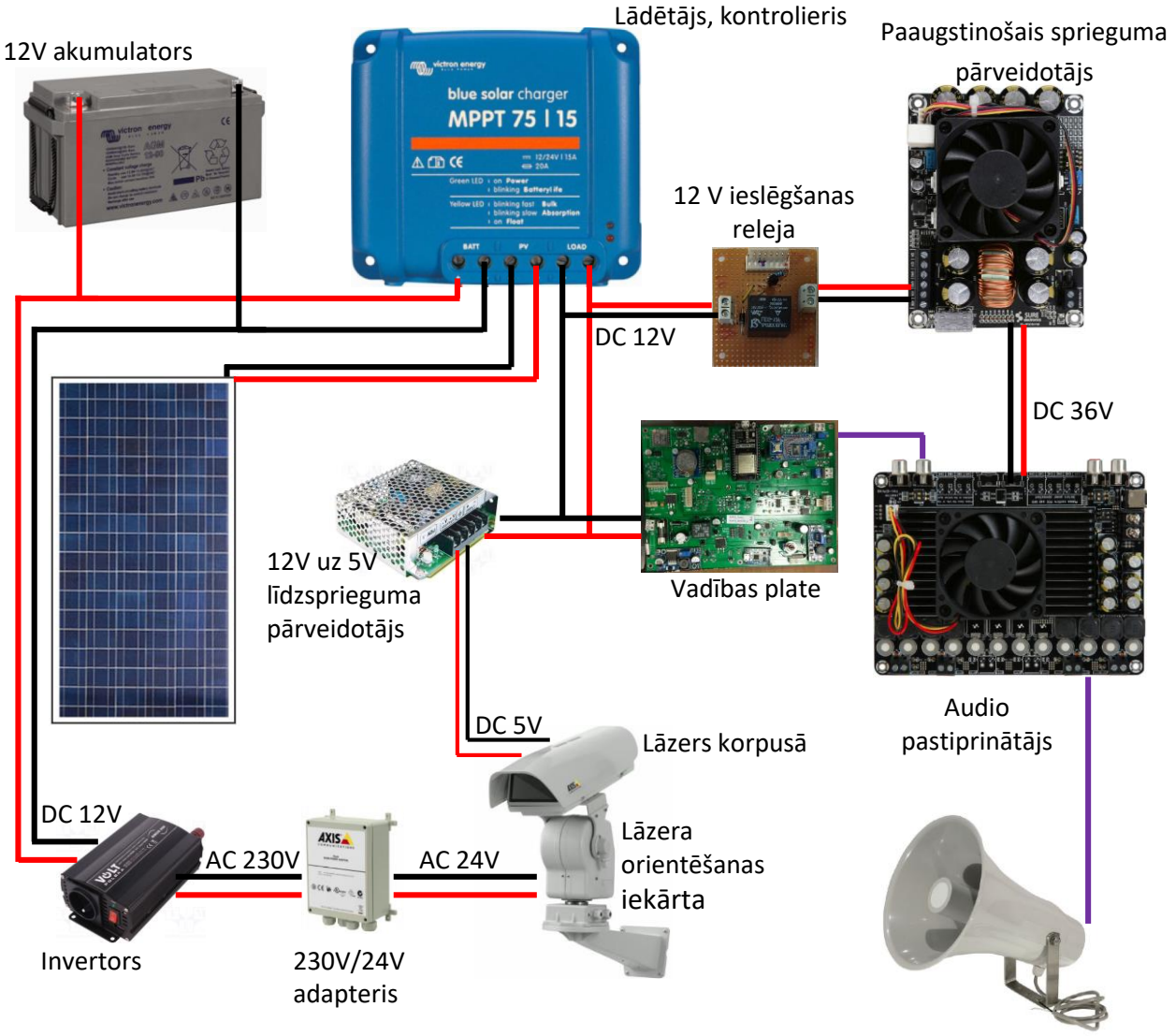
Ražotāja sniegtās akumulatora specifikācijas:

- Kods: BAT412800104
- Kapacitāte: 90 Ah
- Nominālais spriegums: 12 V
- Gabarīti: 350 x 167 x 183 mm
- Svars: 26 kg
- Darba mūžs atkarībā no izlādes pakāpes: 500 cikli pie 80% izlādes, 750 cikli pie 50% izlādes, 1800 cikli pie 30 % izlādes.

No specifikācijas var secināt, ka korpusa detaļām ir jābūt pietiekoši izturīgām, jo akumulatoram ir ievērojama masa. Akumulators kalpos ilgāk, ja saules paneļi spēs nodrošināt nepieciešamu jaudu, respektīvi, jo saulaināka vasara, jo lielāks akumulatora darba mūžs.

3.2. Sistēmas barošanas nodrošinājuma apraksts (Sergejs)

Autonoma sauszemes moduļa elektrobarošanas shēma



Sauszemes moduļa elektrobarošanu nodrošina 100 W saules panelis un 12 V 90 Ah dziļas izlādes gēla svina akumulators. Akumulatora uzlādi un slodzes komutāciju nodrošina MPPT 75/10 kontrolieris. Pa tiešo pie akumulatora ir pieslēgts invertors, kas nepieciešams lāzera orientēšanas iekārtas adaptera barošanai. Paša lāzera barošanai tiek izmantots pazemināša tipa impulsveida sprieguma pārveidotājs. 12 V spriegums tiek padots uz impulsveida paaugstinošo pārveidotaju caur komutācijas plati, kas ļauj ar vadības programmas palīdzību atslēgt enerģijas padevi akustiskai sistēmai, lai taupītu enerģiju. Ar paaugstinoša tipa pārveidotāju iegūst 36 V, kas nepieciešami audio signāla pastiprinātajam.

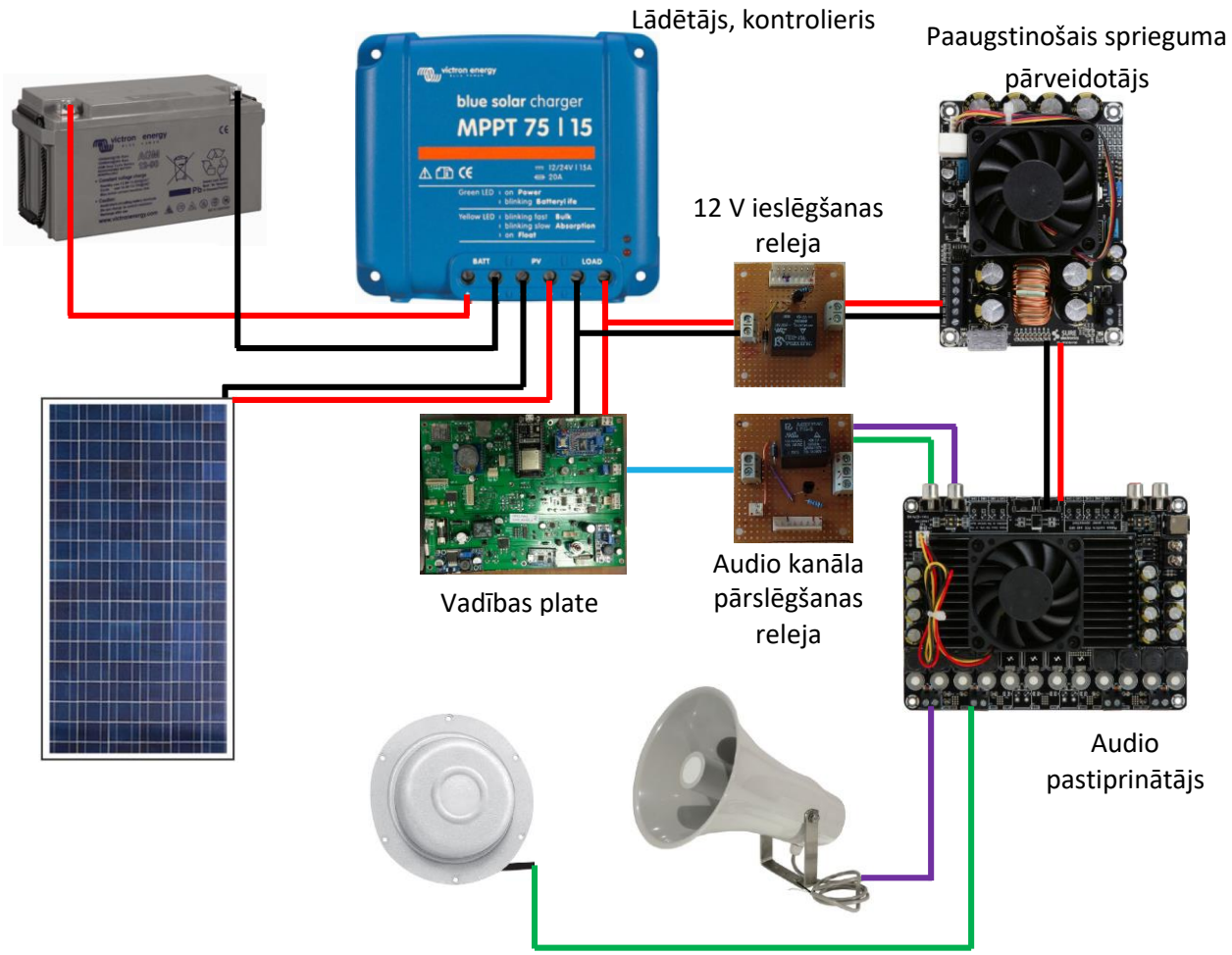
Komponēnšu izvietojums korpusā:

DP2 HAOS iekārtas darba prototipa izstrāde 10.-15.mēn.

Plosta elektrobarošanas shēma

Elektrobarošanas shēmas uzdevums ir nodrošināt ar elektroenerģiju sekojošus patērētājus: vadības plati, divus skaļruņus kas pieslēgti pie audio pastiprinātāja.

Elektrobarošanas sistēmas blokshēma:

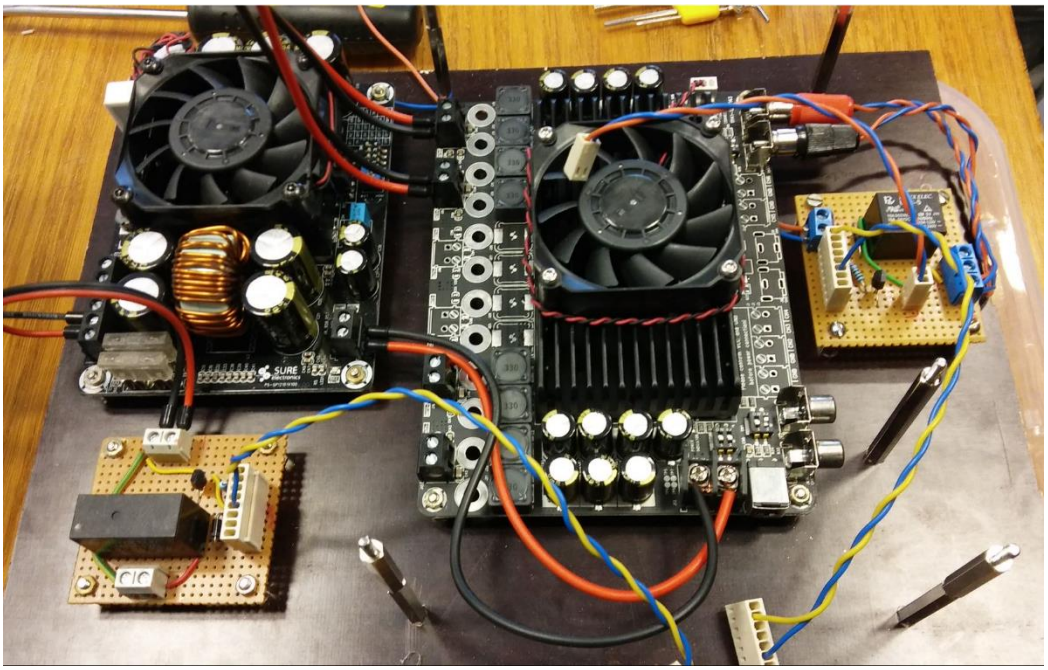


Elektrobarošanas shēmas apraksts:

Viens no svarīgākiem elektrobarošanas shēmas mezgliem ir MPPT 75/10 kontrolieris, kas nodrošina akumulatora uzlādi atbilstoši speciālam algoritmam, kas ļauj paildzināt akumulatora darba mūžu, kā arī komutē slodzi. Divi 50 W polikristāliskie saules paneļi ir saslēgti virknē un pieslēgti pie kontroliera PV ieejas.

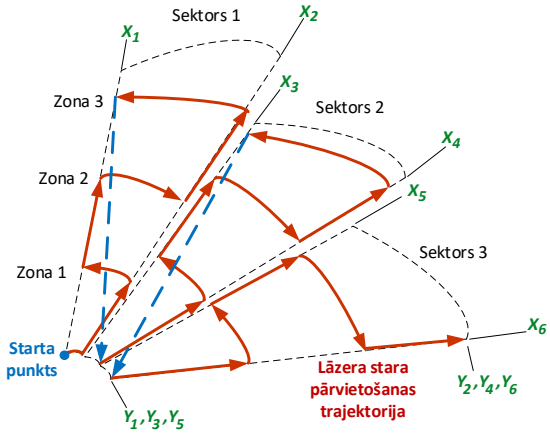
Rezultējošais spriegums tukšgaitā ir ap 33 V. No saules paneļiem iegūta elektroenerģija nodrošina akumulatora uzlādi un slodzes barošanu. Kontroliera izejā ir 12 V līdzspriegums, kas tiek padots uz vadības plati un barošanas komutācijas releja plati. 12 V komutācijas plate tika izveidota, lai varētu ieslēgt audio skaņas pastiprinātāju tikai signālu atskaņošanas laikā, jo mierā režīma tas, pie pieslēgtās barošanas, vienalga patērētu ap 700 mA lielu strāvu. Šāda veidā tiek panākts ilgāks autonomas darbības laiks. Aiz 12 V barošanas komutācijas releja plates atrodas paaugstinošā tipa sprieguma pārveidotājs, kas nodrošina audio pastiprinātājam nepieciešamos 36 V. Audio signāls no vadības plates nonāk uz "Audio kanāla pārslēgšanas plati", kas padot signālu uz atbilstošo kanālu pastiprinātāja platē, no kura tas aiziet uz sauszemes (taures formas) skaļruni vai uz zemūdens skaļruni. Tas tika izdarīts, jo zemūdens un sauszemes skaņas atšķiras un netiek atskaņotas vienlaicīgi.

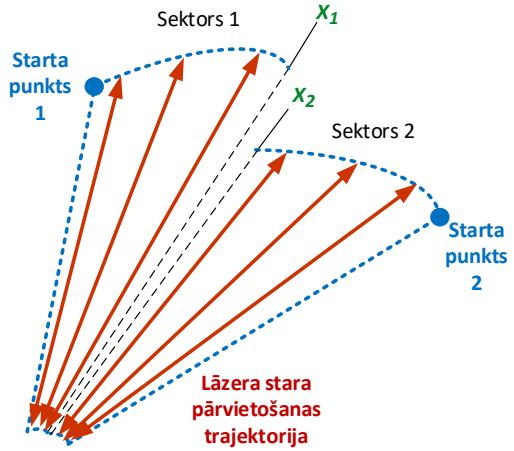
Paaugstinoša sprieguma pārveidotāja, audio pastiprinātāja un releju plašu izvietojums uz pamatnes:



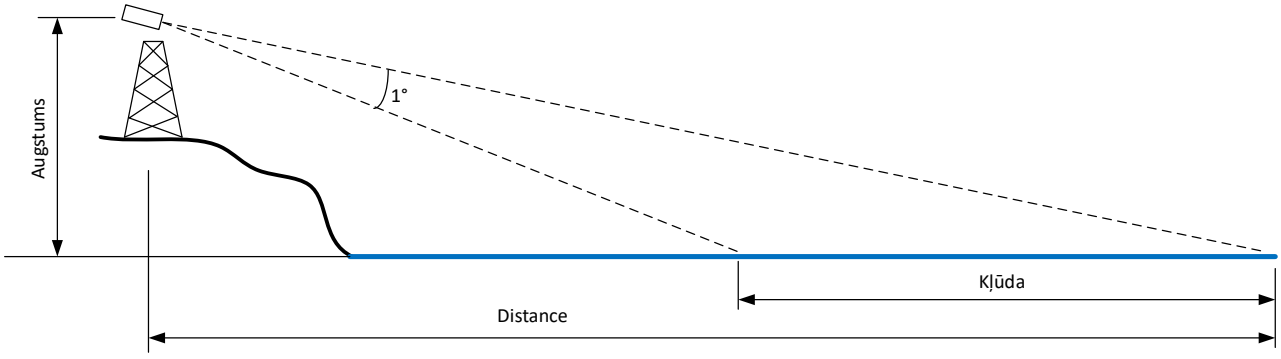
3.3. Virzošās sistēmas darbības algoritms optisko signālu robežu ievērošanai

Stara pārvietojumam tika izveidoti divi dažādi algoritmi. Pirmā variantā darbības laukums ir sadalīts trijos sektoros. Katrā sektorā var izvēlēties noteiktu zonu skaitu. Katrā zonā notiek viens lāzera pārvietojums horizontālā virzienā. Lai nodrošinātu sistēmas kalibrēšanu katru jauno ciklu programma sāk „stāvvietā" (stabilitāti nodrošina galu slēdži).





Testēšanas laikā izpaudās lāzera orientēšanas ierīces YR3040 trūkums – sistēmā nav iebūvētā leņķa mērīšanas devēja, līdz ar to tekošo pozīciju var novērtēt reizinot griešanas ātrumu un laiku. Mikrokontrolieris ir spējīgs regulēt laiku ar ļoti lielu precizitāti (1µsek), bet virzošās sistēmas elektronika var apstrādāt daudz lielāku intervālu, kas dot minimālu griešanas leņķu soli 2-3°. Lāzera tuvumā tas nodrošina pietekami labu precizitāti, bet, piemēram, attālumā 200 metri praktiskā kļūda jau ±50 metri (sk. tabulu). Tas nav kritiski horizontālā virzienā, bet ir ļoti jutīgi lāzera vertikālā pārvietošanā.



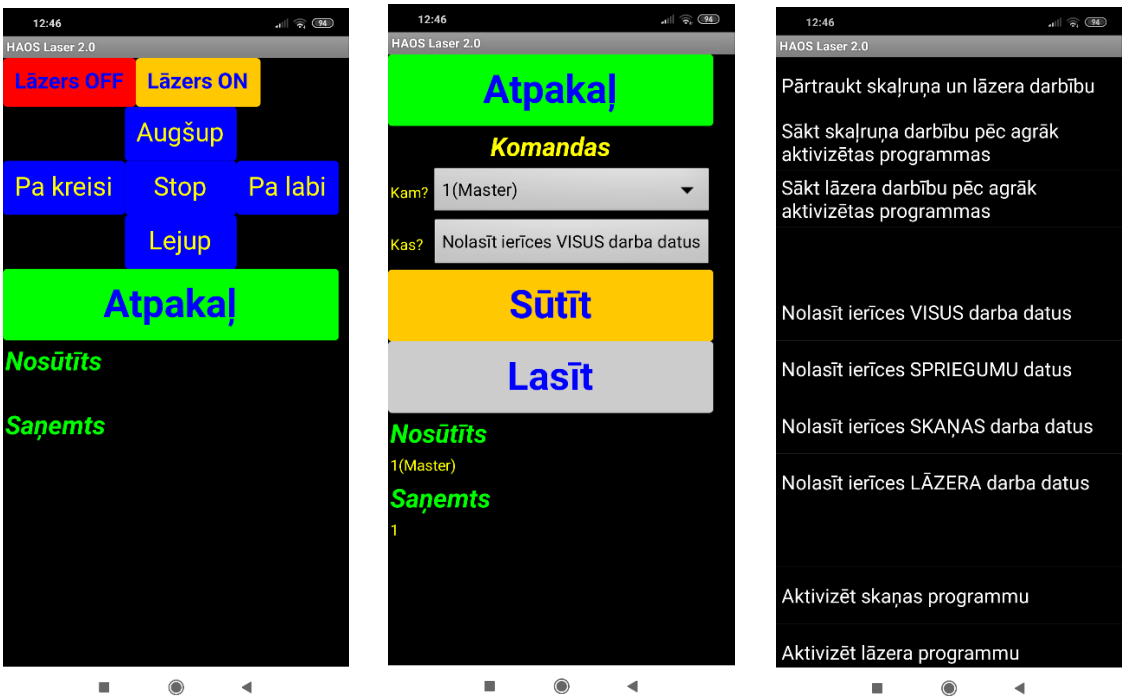
Augstums, m	Distance, m	Leņķis, grad
3	10	73,3
3	15	78,7
3	20	81,5

3	25	83,2
3	30	84,3
3	35	85,1
3	40	85,8
3	45	86,2
3	50	86,6
3	60	87,2
3	70	87,6
3	80	87,9
3	90	88,1
3	100	88,3
3	110	88,5
3	120	88,6
3	130	88,7
3	140	88,8
3	150	88,9
3	160	89,0
3	170	89,0
3	180	89,1
3	190	89,1
3	200	89,2
3	250	89,4
3	300	89,5

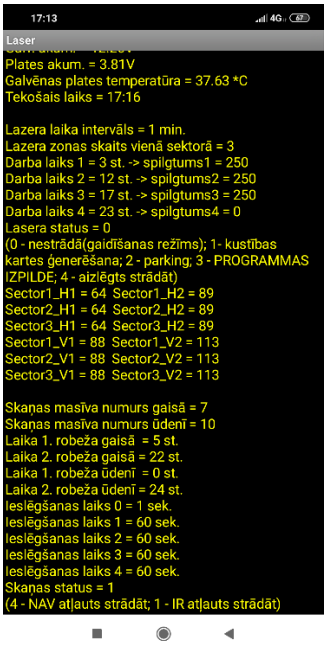
Tāpēc tika izveidots pilnveidots algoritms. Šeit ir tikai divi sektori un stara pārvietošana vertikālā virzienā notiek no vienas malas līdz otrai (līdz galu slēdžu ieslēgšanas momentam). Līdz ar to mēs kompensējam palielinātu kļūdu. Horizontāla virzienā pārvietošanas solis ir fiksēts - 5°.

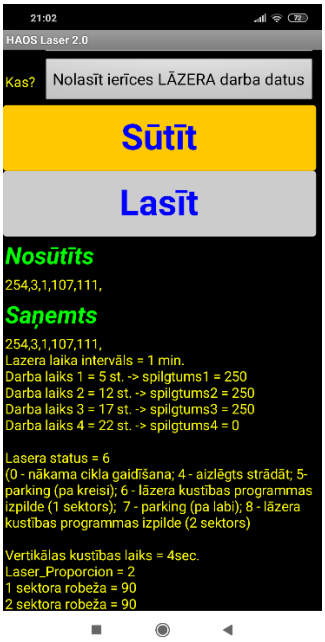
Sistēmas sākotnējai kalibrēšanai un testēšanai tika izveidota aplikācija viedtālrunim.

Aplikācijas galvenā , Bluetooth ierīces pieslēgšanās/izslēgšanas un Bluettoth ierīces izēvelēs lapas.



Lāzera vadības, komandas sūtīšanas un komandas izvēšanas lapas.





Kontroliera bloka atbildes piemēri.

Aplikācijas programmas pilnais teksts tika izvietots pielikumā 4.

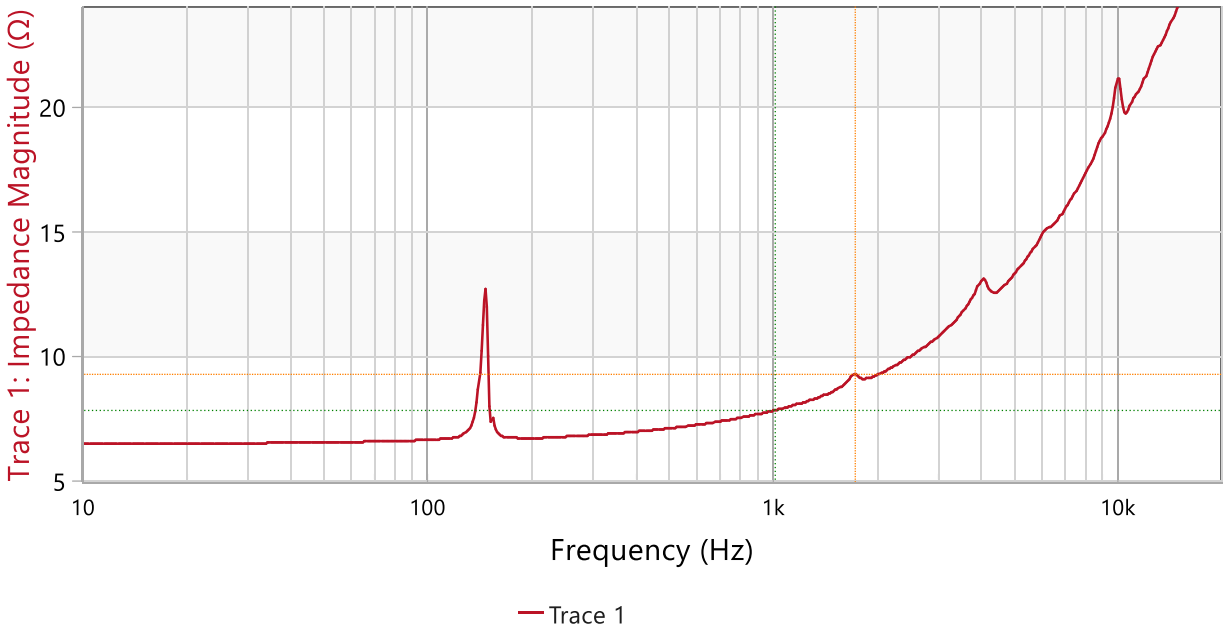
3.4. AP2.3.2 Laboratorijas eksperimentu atskaite

3.4.1. Zemūdens skaļruņu testēšanas dati

Zemūdens skaļruņu ražotāja sniegtas specifikācijas:

Modeļa nosaukums	UW30
Frekvenču diapazons	100 – 10 000 Hz
Ieejas jauda	30 W
Impedance	8 Ω
Darbības dziļums	līdz 3 m no ūdens virsmas
Rekomendējamais uzstādīšanas dziļums	1,2 m
Svars (sauszemē)	1,8 kg
Ģeometriskie izmēri, Diametrs Platums	 182,6 mm 66,3 mm

Skaļruņu impedance var ievērojami mainīties atkarībā no frekvences, tāpēc laboratorijā tika veikts zemūdens skaļruņa impedances mērījums, izmantojot Bode100 ķēžu analizatoru no Omicron Lab. Skaļrunis tika iemērcēts spainī, lai izslēgtu tā sabojāšanas un nodrošinātu vidi, kas ir tuva reālam pielietojumam.



No grafika var redzēt, ka pieaugot signāla frekvencei pieaug arī UW30 impedance, sasniedzot 20 Ω pie 10 kHz. Tas nozīme ka pie augstākam frekvencēm tas skanēs nedaudz klusāk.

Svarīgs parametrs ir akustiskais spiediens, ko rada dotais skaļrunis. Lai novērtētu šo parametru tika veikti mērījumi Daugavā iegremdējot zemūdens skaļruni UW30 1 m dziļumā. Akustiskais spiediens (tālāk tekstā SPL) tika mērīts, izmantojot Teledyne RESON hidrofonu TC4032.

Hidrofona TC4032 galvenie parametri:

Lineārais frekvenču diapazons	15 Hz līdz 40 kHz ±2dB
Uztvērēja jūtība	-170dB re 1V/ µPa
Horizontāla virziendarbība	Omnidirectional ±2dB pie 100kHz
Vertikāla virziendarbība	270° ±2dB pie 15kHz
Maksimālais darba dziļums	600 m
Darba temperatūras diapazons	no -2°C līdz +55°C
Maksimālais izejas spriegums	3,5 Vrms pie 12VDC
Barošanas spriegums	12 līdz 24 VDC

SPL tika mērīts 18.5 m attālumā no zemūdens skaļruņa.

Mērījumi dati:

Skaļruņa puse					Hidrofona puse				
f , Hz	|Z|, Ω	Pin_avg, W	Vin AC RMS, V	Icc, A	AV, mV	MAX, mV	SPL avg 18m [dB]	SPL avg 1m [dB]	SPL max 1m [dB]
147	12,7	30	19,6	4,4	1	1,7	139	158	163
300	6,8	31			0,018	0,021	104	123	124
500	7,1	31	14,7	2,6	0,01	0,01	99	118	118
1000	7,8	31	15,6	2,6	7	9	156	175	177
1500	8,6	31,7	16,5	2,65	13	16	161	180	182
2000	9,2	31,4	17	2,58	26	54	167	186	193
2500	10	30,9	17,6	2,57	7	13	156	175	180
3000	10,8	31	18,3	2,57	2	6	145	164	173
4000	13	30,5	19,9	2,59	4	16	151	170	182

Pie atšķirīgām frekvencēm tika uzturēts gandrīz konstanta ieejas signāla jauda, kā arī tika piefiksēta pastiprinātāja patērētā barošanas strāva. Eksperimentu gaitā tika noskaidrots, ka skaļrunis veido lielāku SPL frekvenču diapazonā no 1 kHz līdz 2,5 kHz. Vidējais jaudas patēriņš šajā diapazonā ir 31 W.

4. U3.1. Testešanas plāna izstrade

Nosacījumi: Izstrādāt HAOS prototipa darbības efektivitātes testēšanas plānu, ņemot vērā zivēdājputnu aktivitātes periodus, ūdenstilpņu pieejamību, dīķu īpašības (izmēra, formas, ūdensaugu veida un aizauguma pakāpes u.c.), dažādas zivju sugas un vecumus, zivju audzēšanas procesus un citus apstākļus.

Testēšanas plāns paredz:

Prototipa testu veikšanu ZI BIOR zivju audzētavu dīķos;

Prototipa testu veikšanu biedrības zivju audzētavu dīķos.

Dīķu lielumi (platība, ha):

ZI BIOR zivju audzētavu dīķi – 1ha; 1-2ha; 2-3ha;

Biedrības zivju audzētavu dīķi – 3-5ha; >20ha; 0,2-1ha (ziemošanas dīķi).

Zivju sugas un vecums (0+ vienvasaras; 1+ divvasaru; 2+ trīsvasaru un vecākas):

ZI BIOR zivju audzētavu dīķi – zandartu mazuļi (0+), vimbu mazuļi (0+);

Biedrības zivju audzētavu dīķi – karpu mazuļi (0+), karpas (1+), karpas (2+ un vecākas).

Zivju audzēšanas procesi:

ZI BIOR zivju audzētavu dīķi – zandartu un vimbu mazuļu audzēšana un nozvejas process;

Biedrības zivju audzētavu dīķi – karpu audzēšana un nozvejas process.

Testēšanas reģioni:

ZI BIOR zivju audzētavu dīķi – ZA Dole, ZA Tome (Ķeguma dīķi) Vidzemes reģions;

Biedrības zivju audzētavu dīķi – Vidzemes, Sēlijas un Kurzemes reģions.

HAOS prototipa darbības efektivitātes testēšana tiks veikta no jūlija līdz novembrim.

Mēnesis	**Vieta**	**Dīķa platība**	**Zivju suga**	**Vecums**	**Reģions**	**Audzēšanas process**
Jūlijs (4 dīķi)	ZA Dole, ZA Tome	1-3ha	Zandarti, vimbas	0+	Vidzeme	Audzēšana, nozveja

Augusts (4 dīķi)	ZA Dole, ZA Tome	1-3ha, 5ha	Zandarti, vimbas, karpas	0+, 1+	Vidzeme	Audzēšana, nozveja
Septembris (2 dīķi)	Biedrības ZA	5ha, >20ha	Karpas	1+, 2+	Vidzeme, Sēlija	Audzēšana, nozveja
Oktobris (2 dīķi)	Biedrības ZA	>20ha, <1ha	Karpas	2+	Sēlija, Kurzeme	Audzēšana, nozveja, ziemošana
Novembris (1 dīķis)	Biedrības ZA	<1ha	Karpas	2+	Kurzeme	Ziemošana

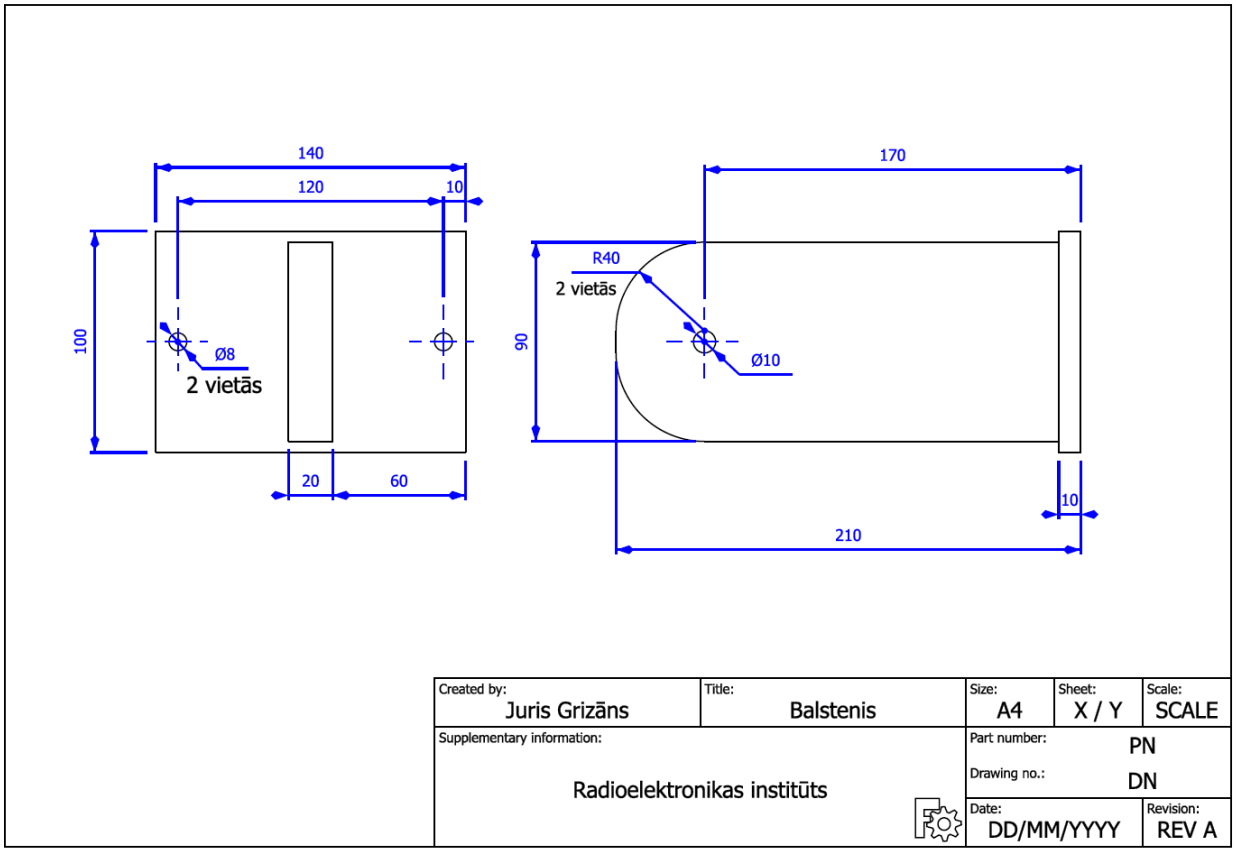
HAOS prototipa darbības efektivitātes testēšanas laikā tiks veikta novērojumu fiksēšana, rezultātu protokolēšana, īpašu uzmanību pievēršot laikapstākļiem, gaismas intensitātei, iekārtas darbībai, attālumiem, zivju uzvedībai, u.c. faktoriem.

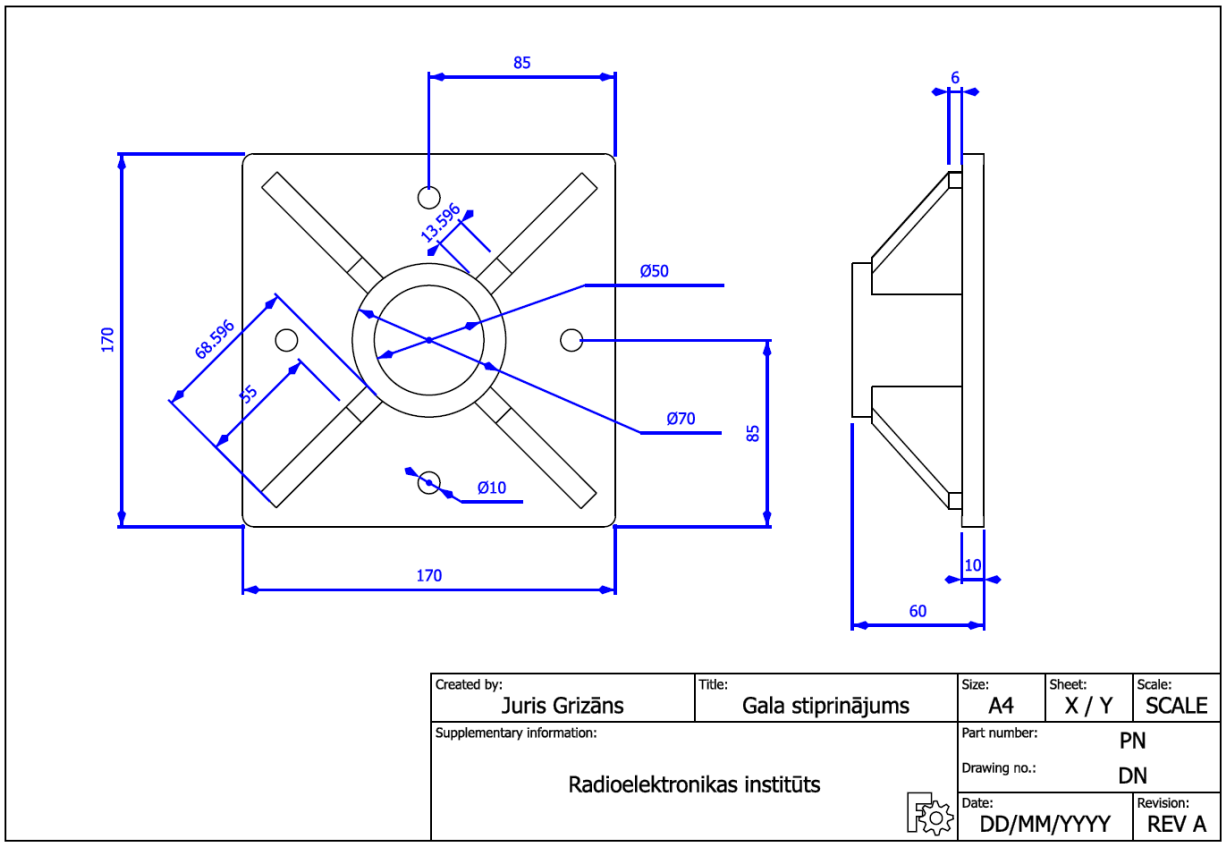
Testēšanai notiekot konkrētā saimniecībā, tiks intervēti arī tās atbildīgie zivkopji vai zivju audzētavas īpašnieki, lai noskaidrotu viņu viedokli un papildinātu iegūtos rezultātus.

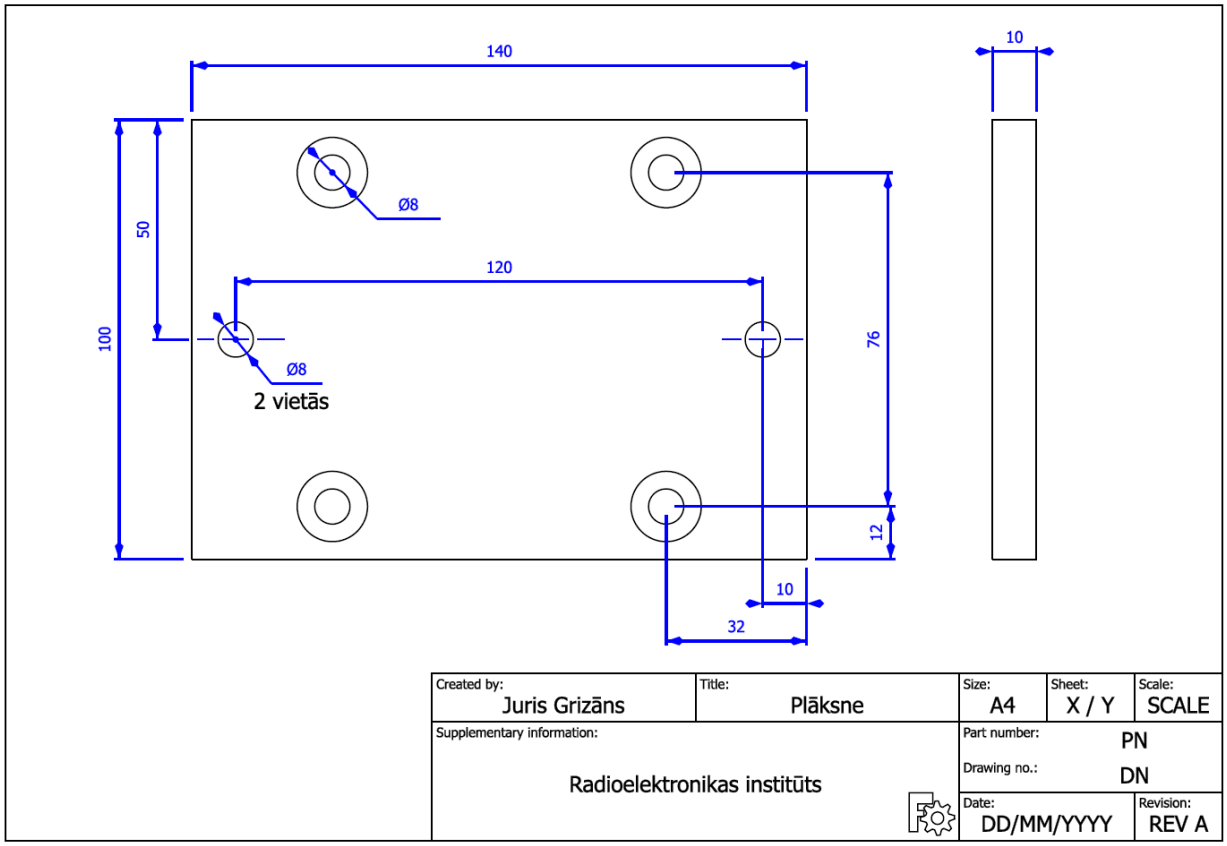
Galvenā uzmanība tiks pievērsta iekārtas prototipu efektivitātei attiecībā uz zivēdājputniem (Jūras krauklis, Pelēkais un Baltais gārnis, Ķīris, Laucis, Krīklis, Gaura u.c.), kā arī kāda būs ietekme uz citiem putniem (Meža pīles, gulbji, stārķi u.c.).

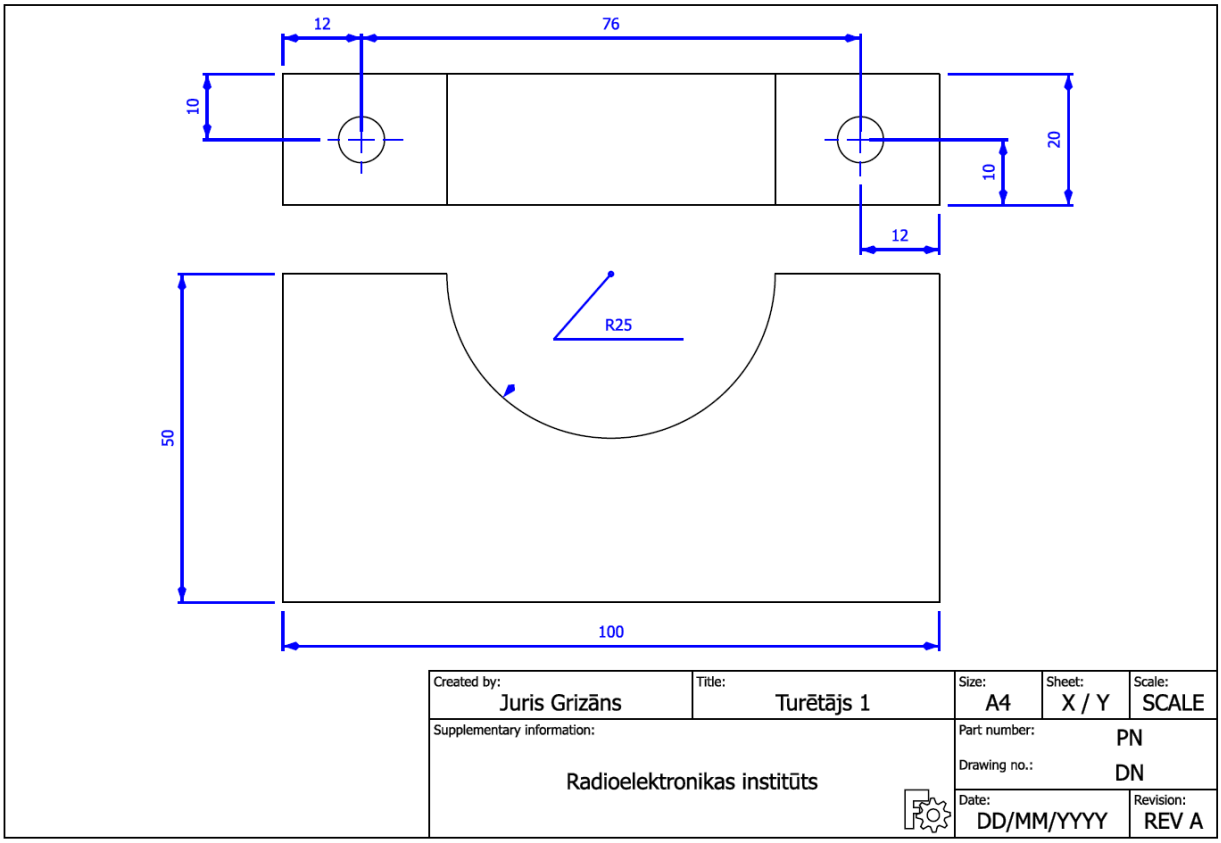
Par novērojumu būtiskākajiem rezultātiem regulāri tiks informēti RTU pētnieki, lai iespēju robežās procesa gaitā varētu HAOS prototipu uzlabot un optimizēt, lai padarītu to efektīvāku.

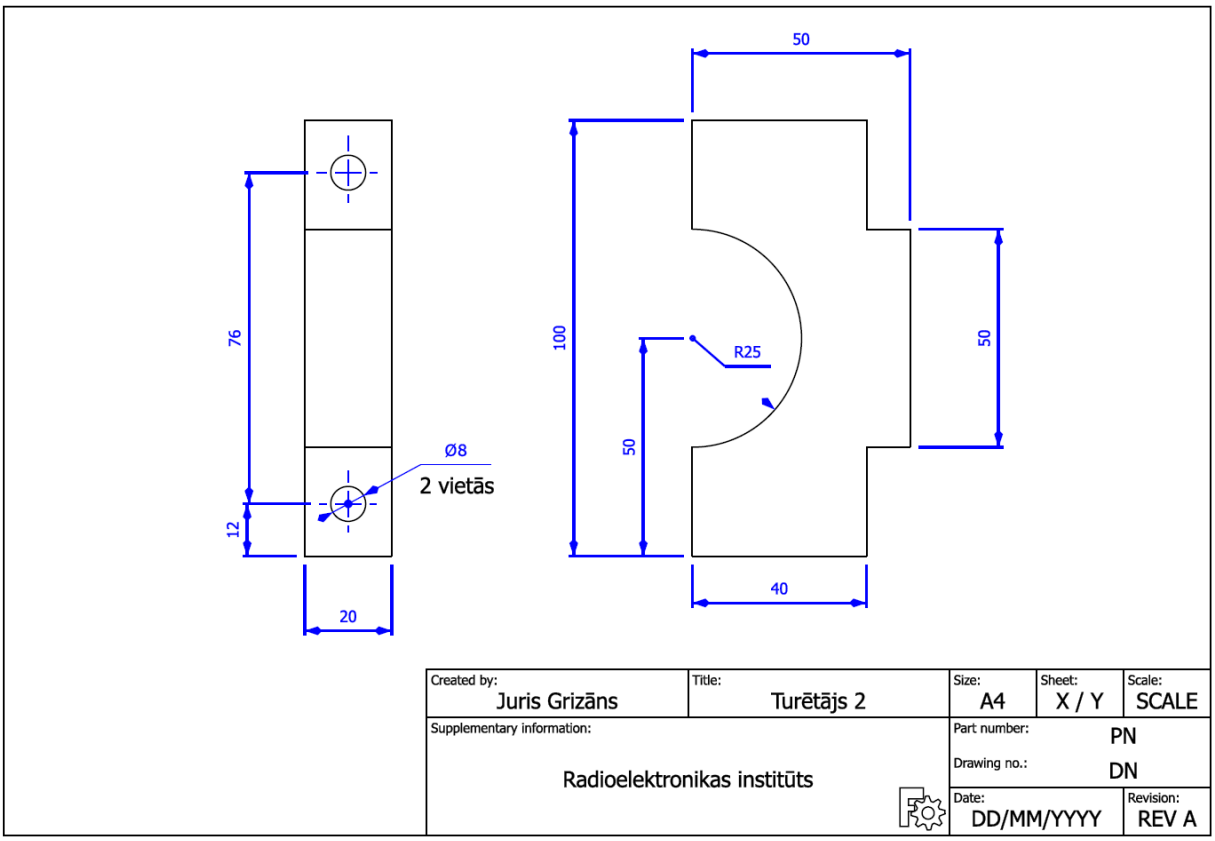
Pielikums 1. Sauszemes un virsūdens ierīces konstrukcija



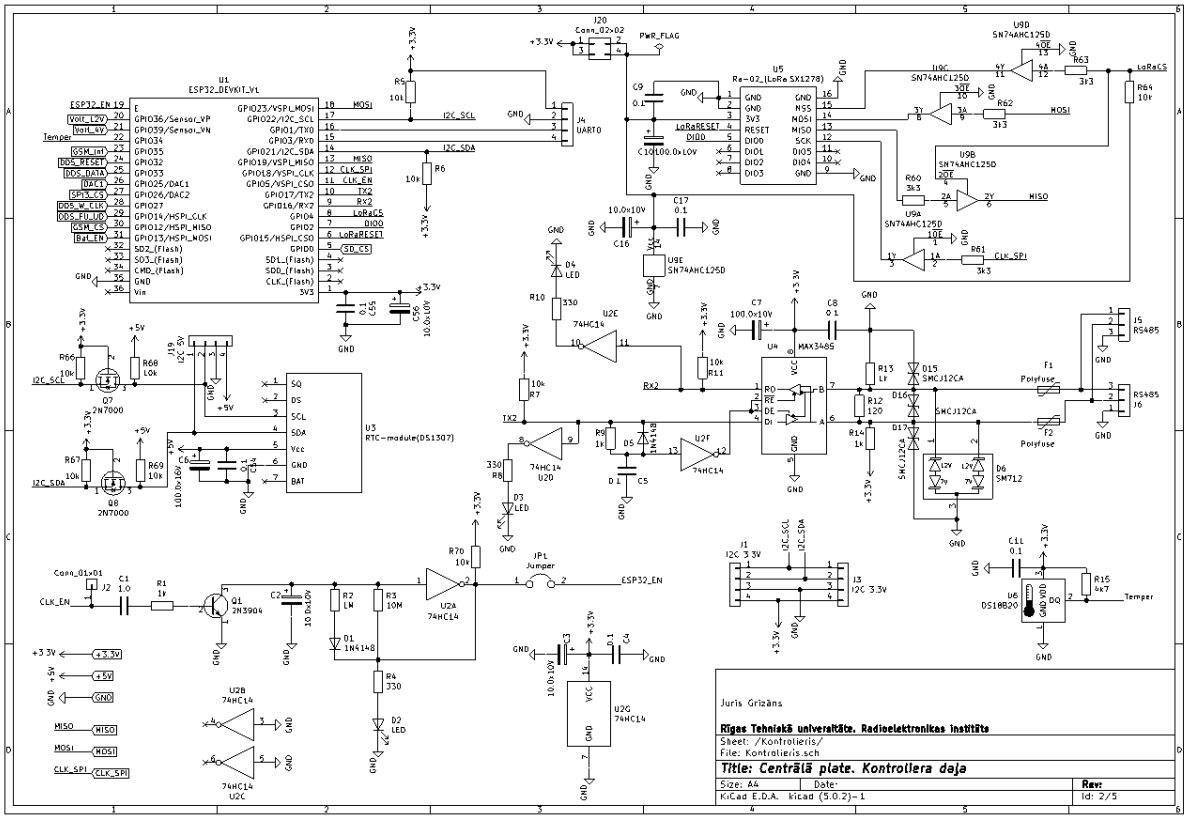

85
6
13.596
Ø50
170
68.596
55
Ø70
85
Ø10
170
10
60
Created by: Juris Grizāns
Title: Gala stiprinājums
Size: A4
Sheet: X / Y
Scale: SCALE
Supplementary information:
Radioelektronikas institūts
Part number: PN
Drawing no.: DN
Date: DD/MM/YYYY
Revision: REV A

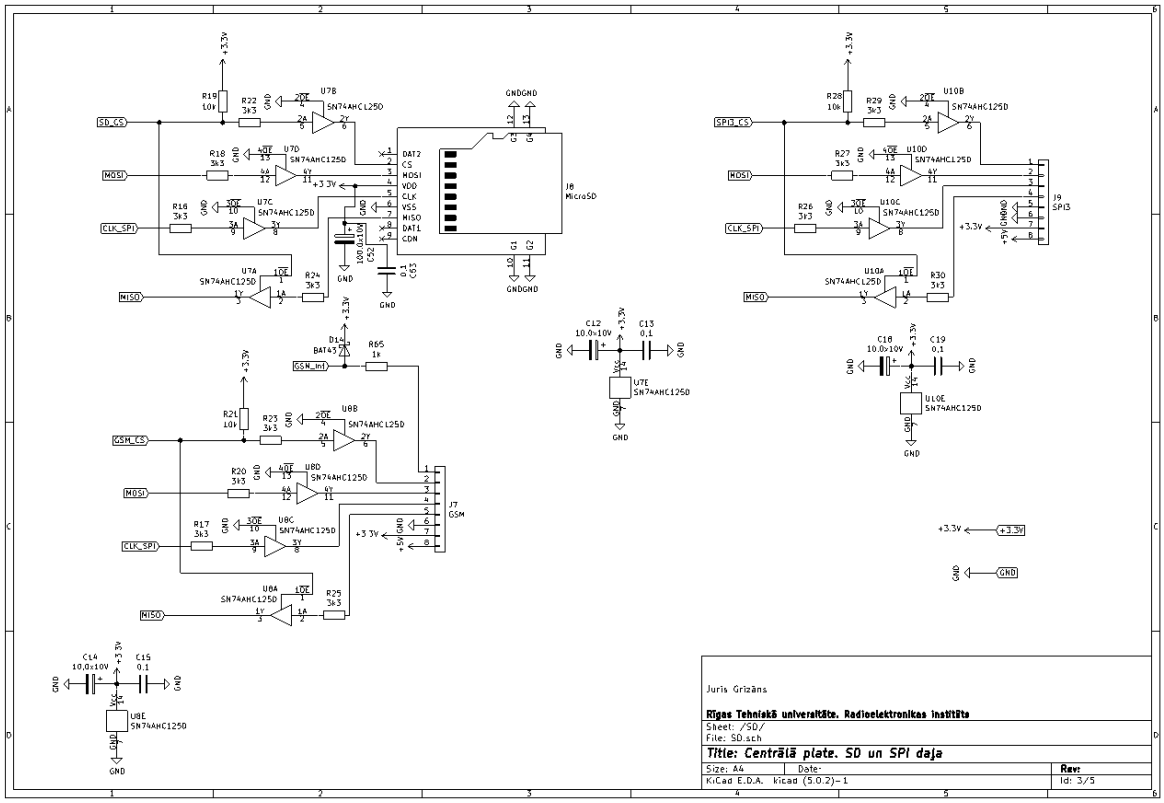

140
10
Ø8
50
120
100
76
Ø8
2 vietās
12
10
32
Created by: Juris Grizāns
Title: Plāksne
Size: A4
Sheet: X / Y
Scale: SCALE
Supplementary information:
Radioelektronikas institūts
Part number: PN
Drawing no.: DN
Date: DD/MM/YYYY
Revision: REV A

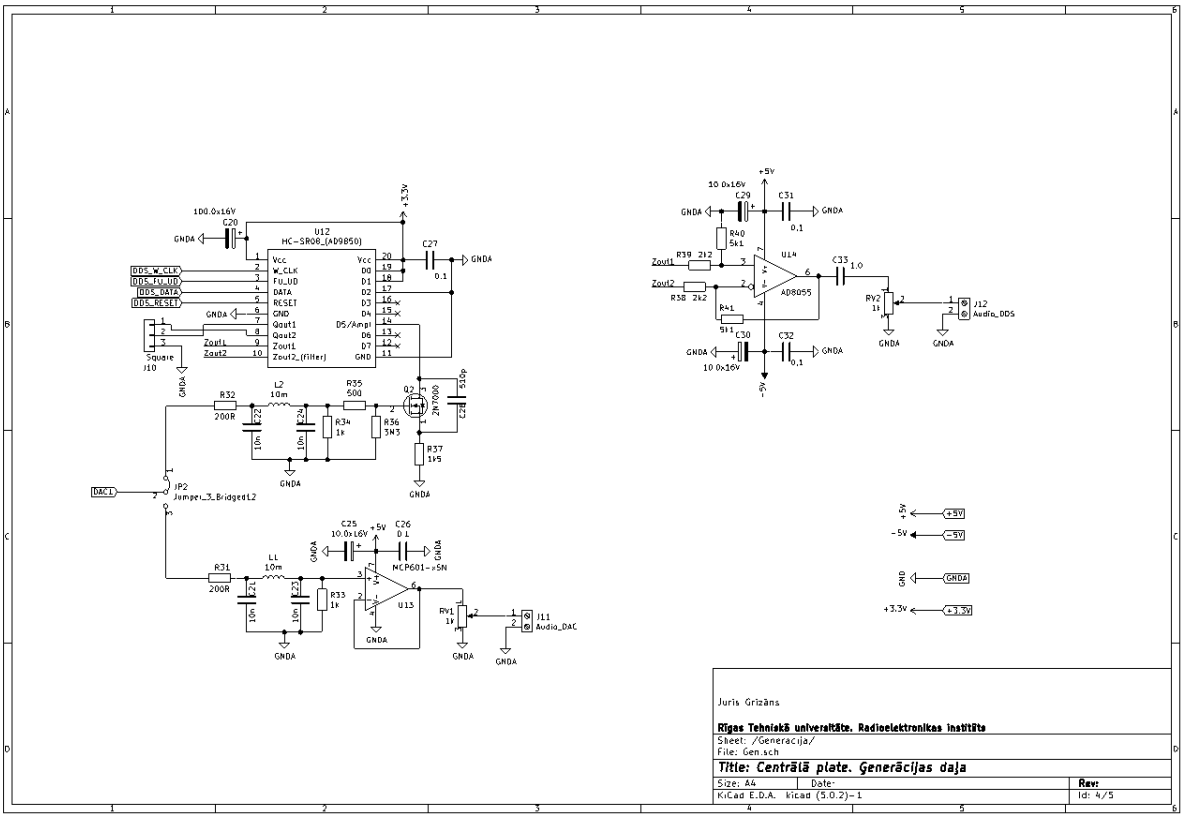

12
76
10
20
10
12
R25
50
100
Created by:
Juris Grizāns
Title:
Turētājs 1
Size:
A4
Sheet:
X / Y
Scale:
SCALE
Supplementary information:
Radioelektronikas institūts
Part number:
PN
Drawing no.:
DN
Date:
DD/MM/YYYY
Revision:
REV A

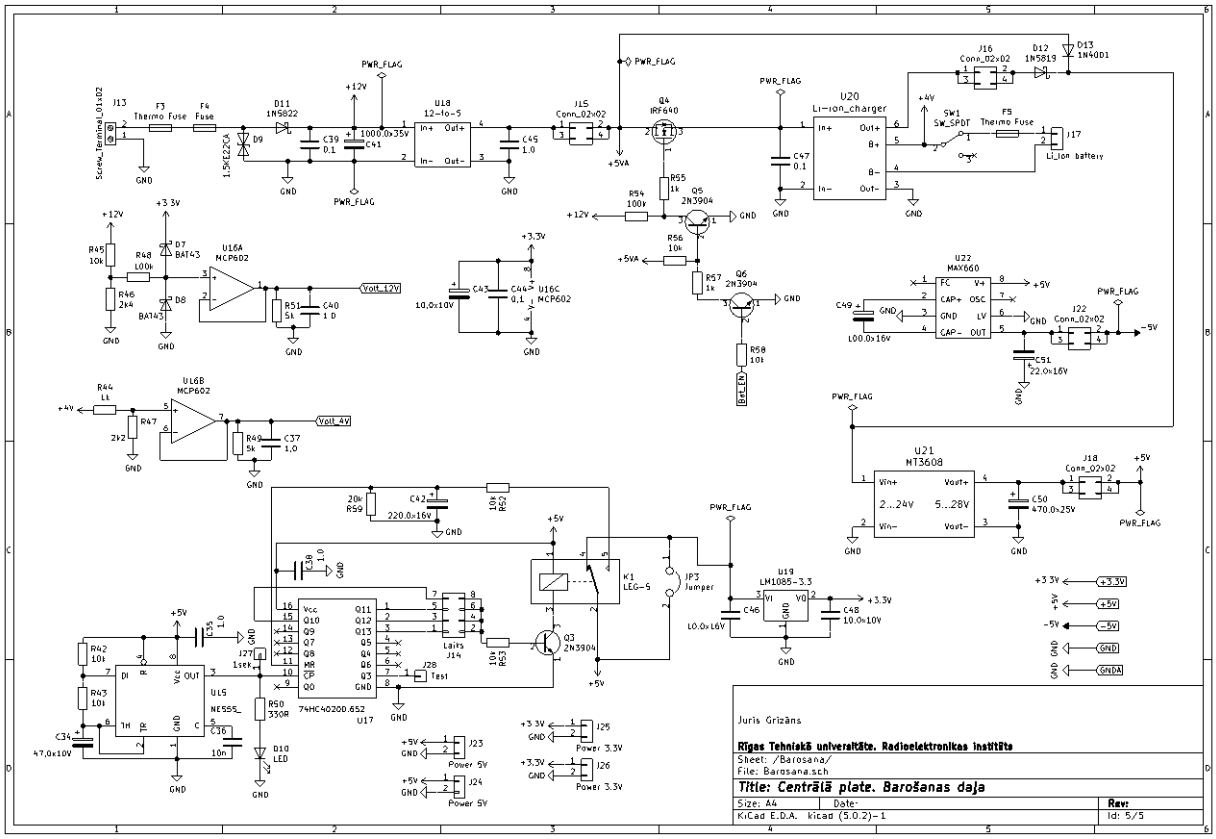

50
76
100
R25
50
50
Ø8
2 vietās
12
20
40
Created by: Juris Grizāns
Title: Turētājs 2
Size: A4
Sheet: X / Y
Scale: SCALE
Supplementary information:
Radioelektronikas institūts
Part number: PN
Drawing no.: DN
Date: DD/MM/YYYY
Revision: REV A

Pielikums 2. Centrālā bloka principiālās shēmas

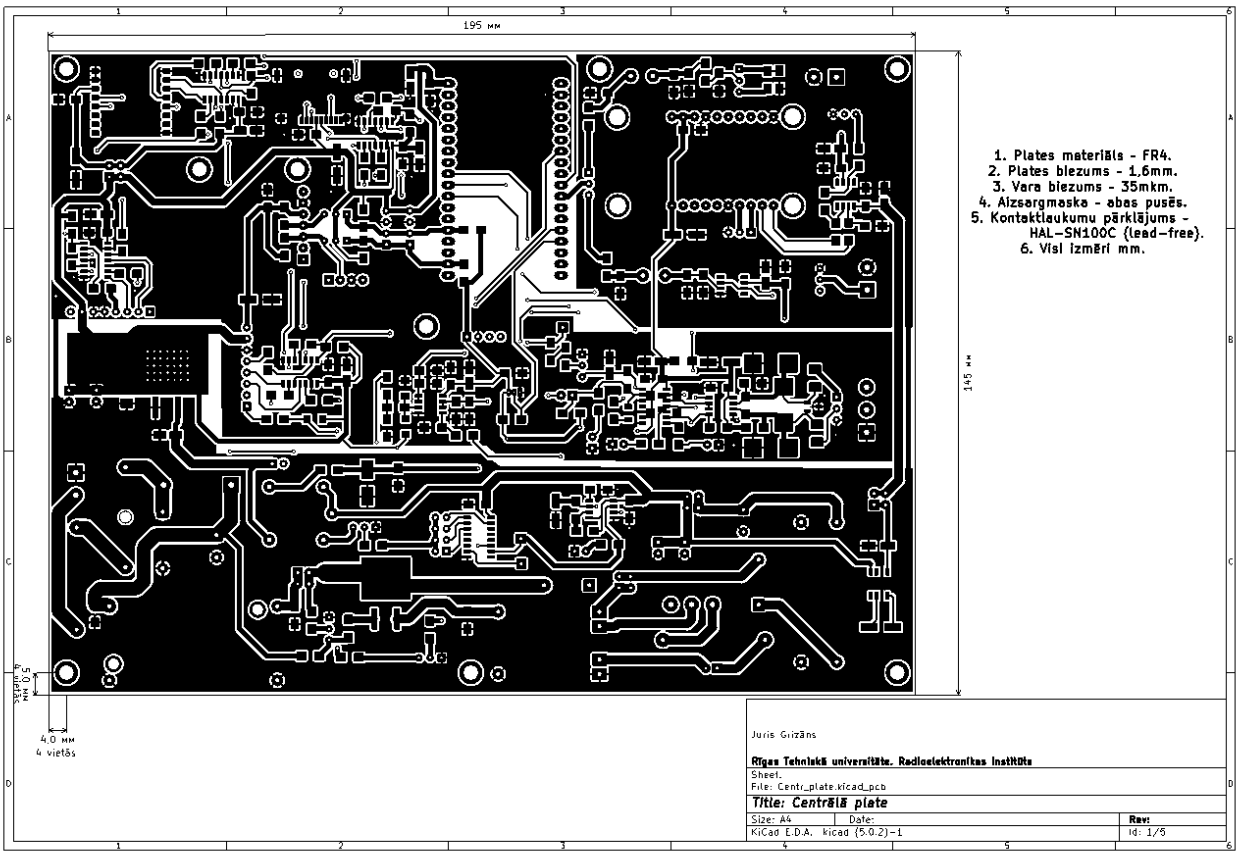


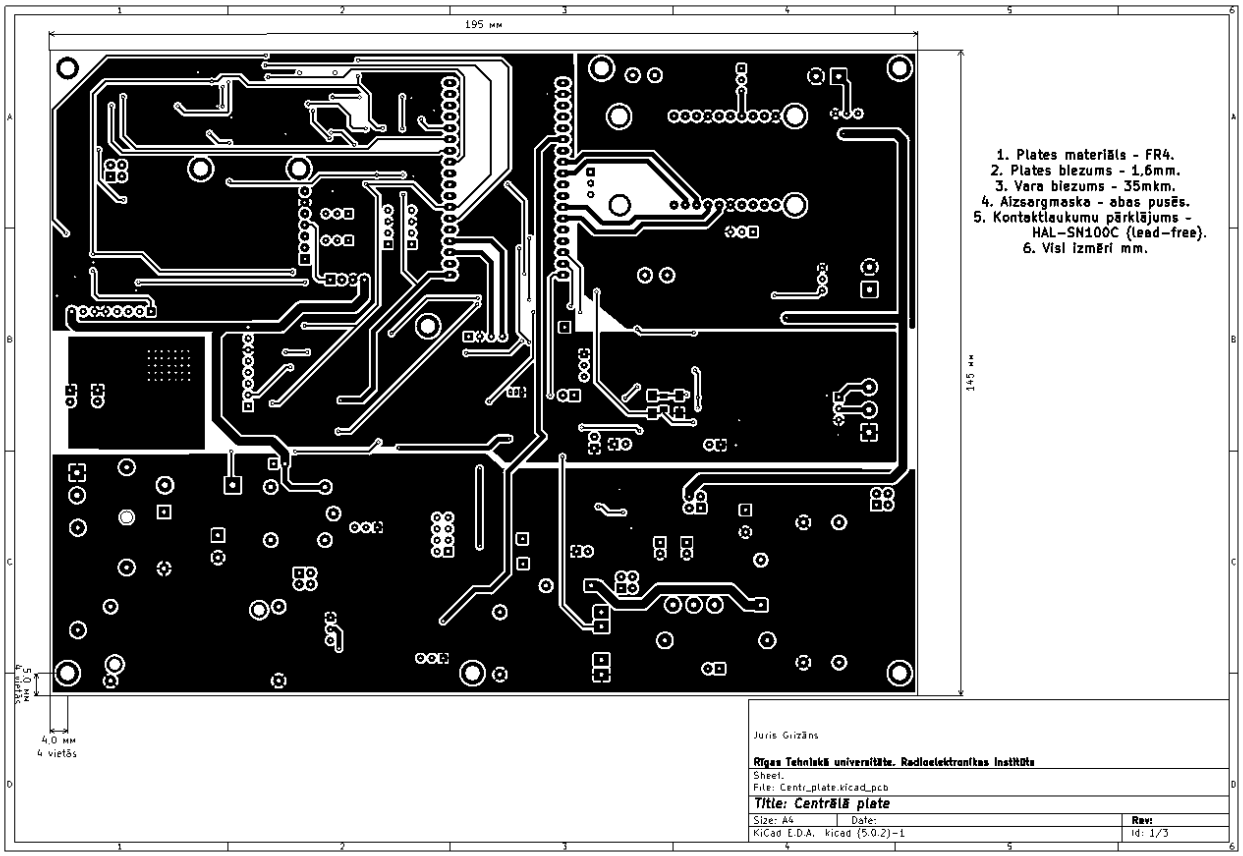

SD_CS
MOSI
CLK_SPI
MISO
GSM_CS
SPI3_CS
U7B
U7D
U7C
U7A
U8B
U8D
U8C
U8A
U10B
U10D
U10C
U10A
U7E
U8E
U10E
J8
MicroSD
J7
GSM
J9
SPI3
D14
BAT43
GSM_Int
Juris Grīzāns
Rīgas Tehniskā universitāte. Radioelektronikas institūts
Sheet: /SD/
File: SD.sch
Title: Centrālā plate. SD un SPI daja
Size: A4
Date:
Rev:
KiCad E.D.A. kicad (5.0.2)-1
Id: 3/5

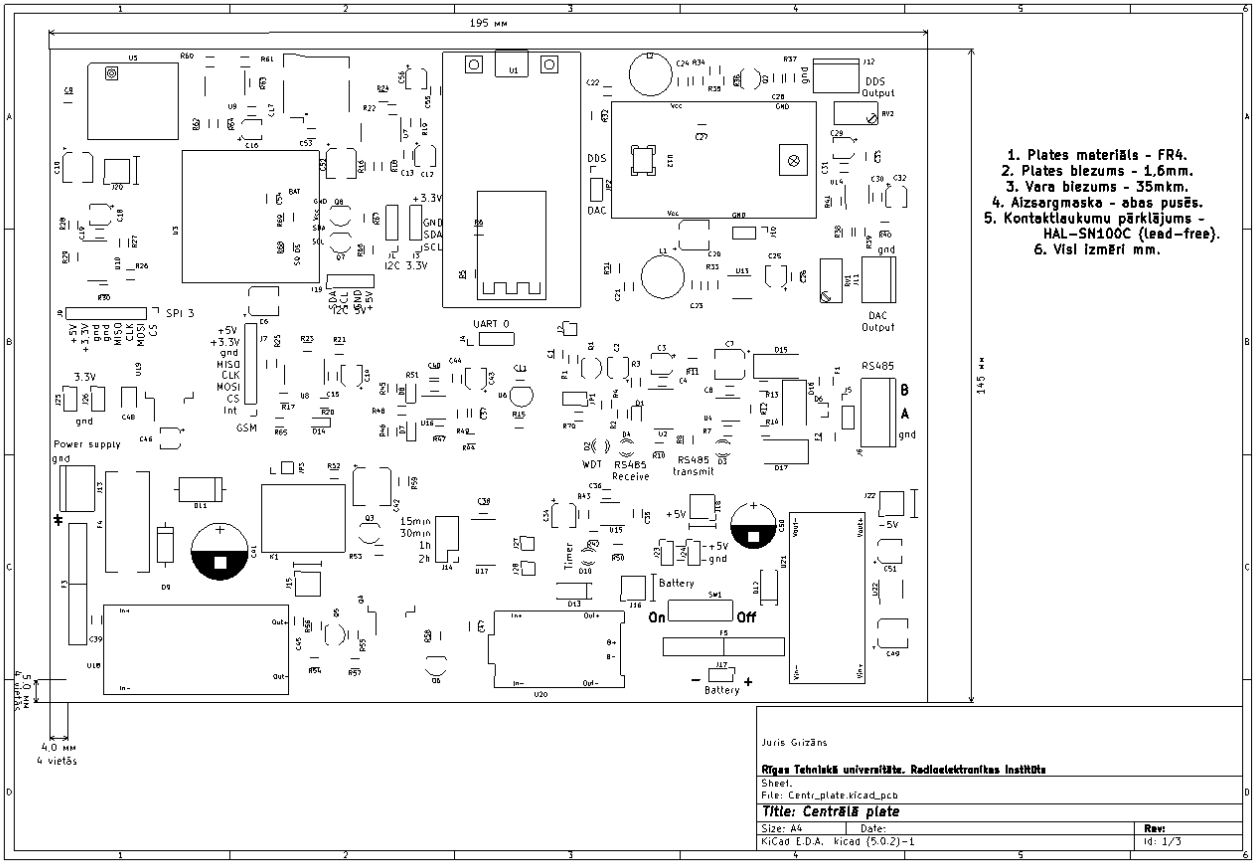

U12
HC-SR08_(AD9850)
C20
100.0x16V
C27
0.1
DDS_W_CLK
DDS_FU_UD
DDS_DATA
DDS_RESET
Square
J10
Zout1
Zout2
R32
200R
L2
10m
R35
500
R34
1k
R36
3N3
R37
1k5
C24
510p
JP2
Jumper_3_Bridged12
DACL
C25
10.0x16V
C26
0.1
R31
200R
L1
10m
R33
1k
U13
MCP601-xSN
RV1
1k
J11
Audio_DAC
C29
10.0x16V
C31
0.1
R40
5k1
R39 2k2
R38 2k2
U14
AD8055
C33
1.0
R41
911
C30
10.0x16V
C32
0.1
RV2
1k
J12
Audio_DDS
Juris Grīzāns
Rīgas Tehniskā universitāte. Radioelektronikas institūts
Sheet: /Generacija/
File: Gen.sch
Title: Centrālā plate. Ģenerācijas daļa
Size: A4
Date:
Rev:
KiCad E.D.A. kicad (5.0.2)-1
Id: 4/5

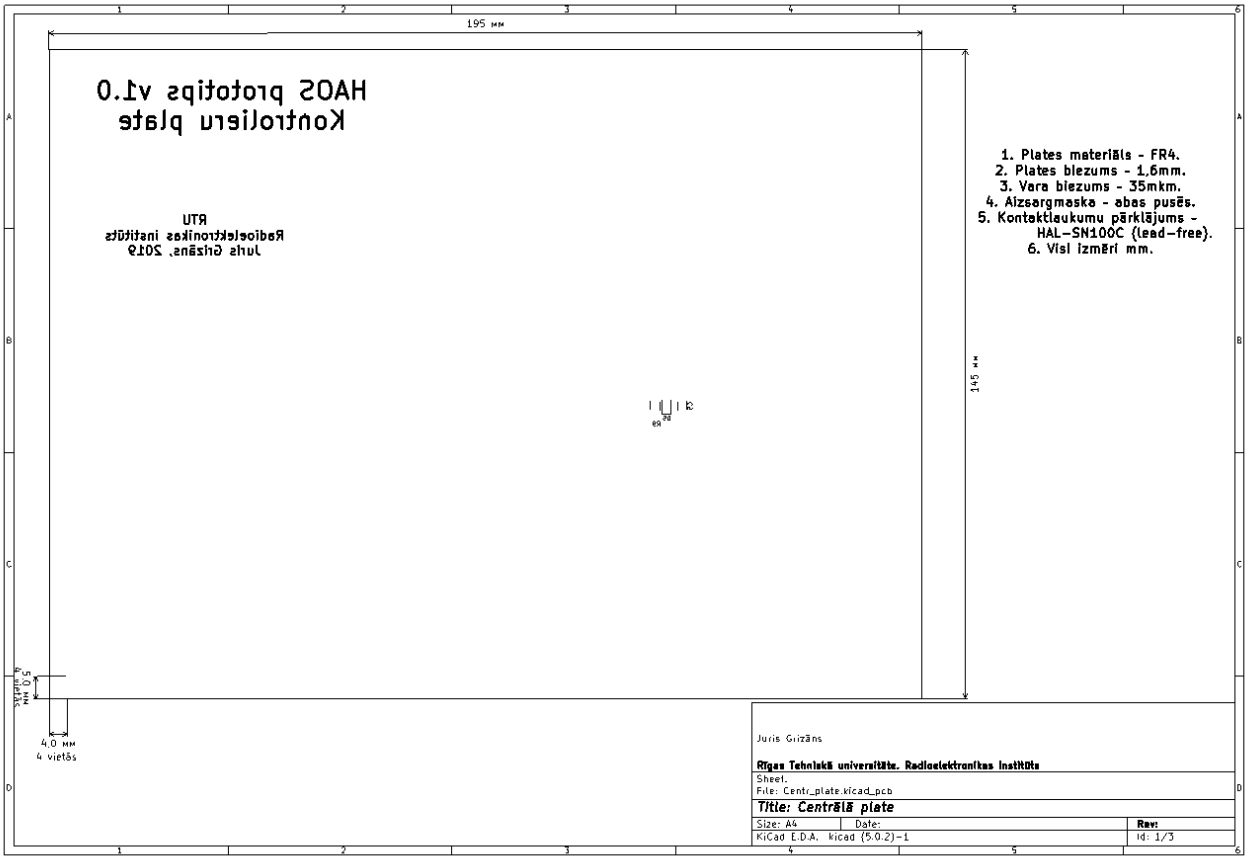

Juris Grizāns
Rīgas Tehniskā universitāte. Radioelektronikas institūte
Sheet: /Barosana/
File: Barosana.sch
Title: Centrālā plate. Barošanas daja
Size: A4
Date:
Rev:
KiCad E.D.A. kicad (5.0.2)-1
Id: 5/5

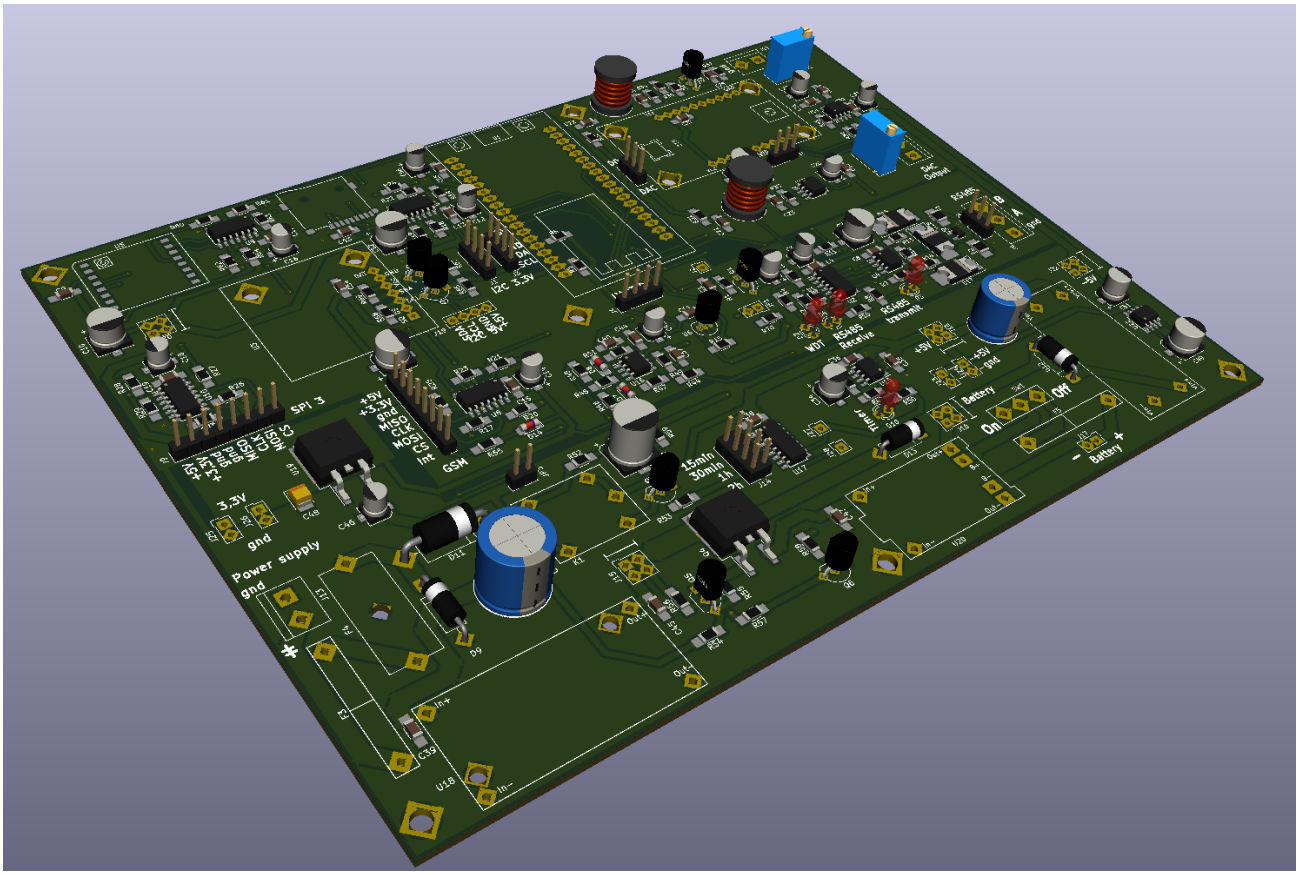
Pielikums 3. Centrālā bloka iespiestā plate



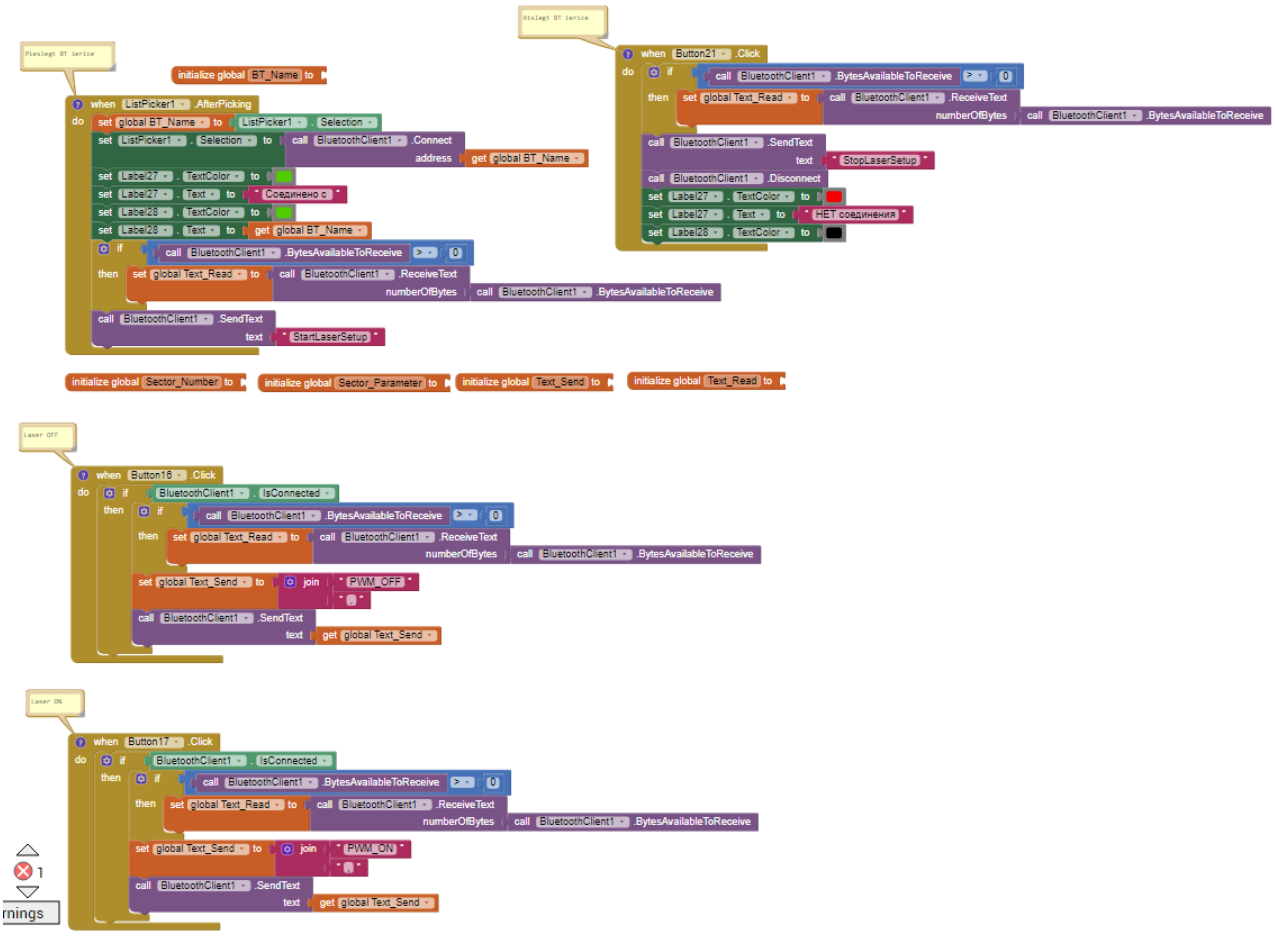

195 мм
145 мм
1. Plates materiāls - FR4.
2. Plates biezums - 1,6mm.
3. Vara biezums - 35mkm.
4. Aizsargmaska - abas pusēs.
5. Kontaktlaukumu pārklājums - HAL-SN100C (lead-free).
6. Visi izmēri mm.
4,0 мм
4 vietās
Juris Grizāns
Rīgas Tehniskā universitāte. Radioelektronikas institūts
Sheet:
File: Centr_plate.kicad_pcb
Title: Centrālā plate
Size: A4
Date:
Rev:
KiCad E.D.A. kicad (5.0.2)-1
Id: 1/3

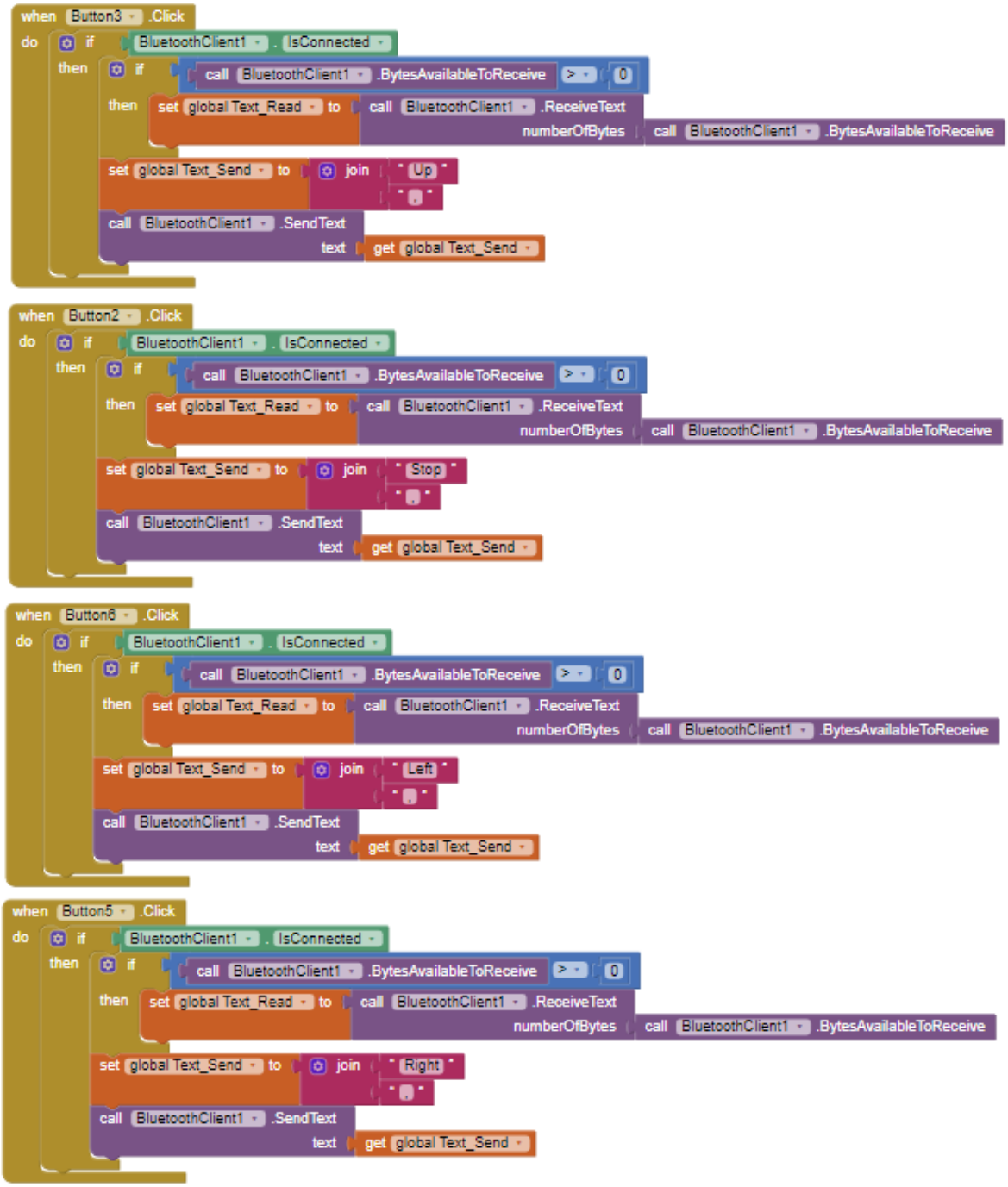

195 mm
145 mm
1. Plates materiāls - FR4.
2. Plates biezums - 1,6mm.
3. Vara biezums - 35mkm.
4. Aizsargmaska - abas pusēs.
5. Kontaktlaukumu pārklājums - HAL-SN100C (lead-free).
6. Visi izmēri mm.
DDS Output
DAC Output
UART 0
SPI 3
GSM
RS485
WDT
RS485 Receive
RS485 transmit
Power supply
Timer
Battery
On
Off
4,0 mm
4 vietās
Juris Grizāns
Rīgas Tehniskā universitāte. Radioelektronikas institūts
Sheet:
File: Centr_plate.kicad_pcb
Title: Centrālā plate
Size: A4
Date:
Rev:
KiCad E.D.A. kicad (5.0.2)-1
Id: 1/3

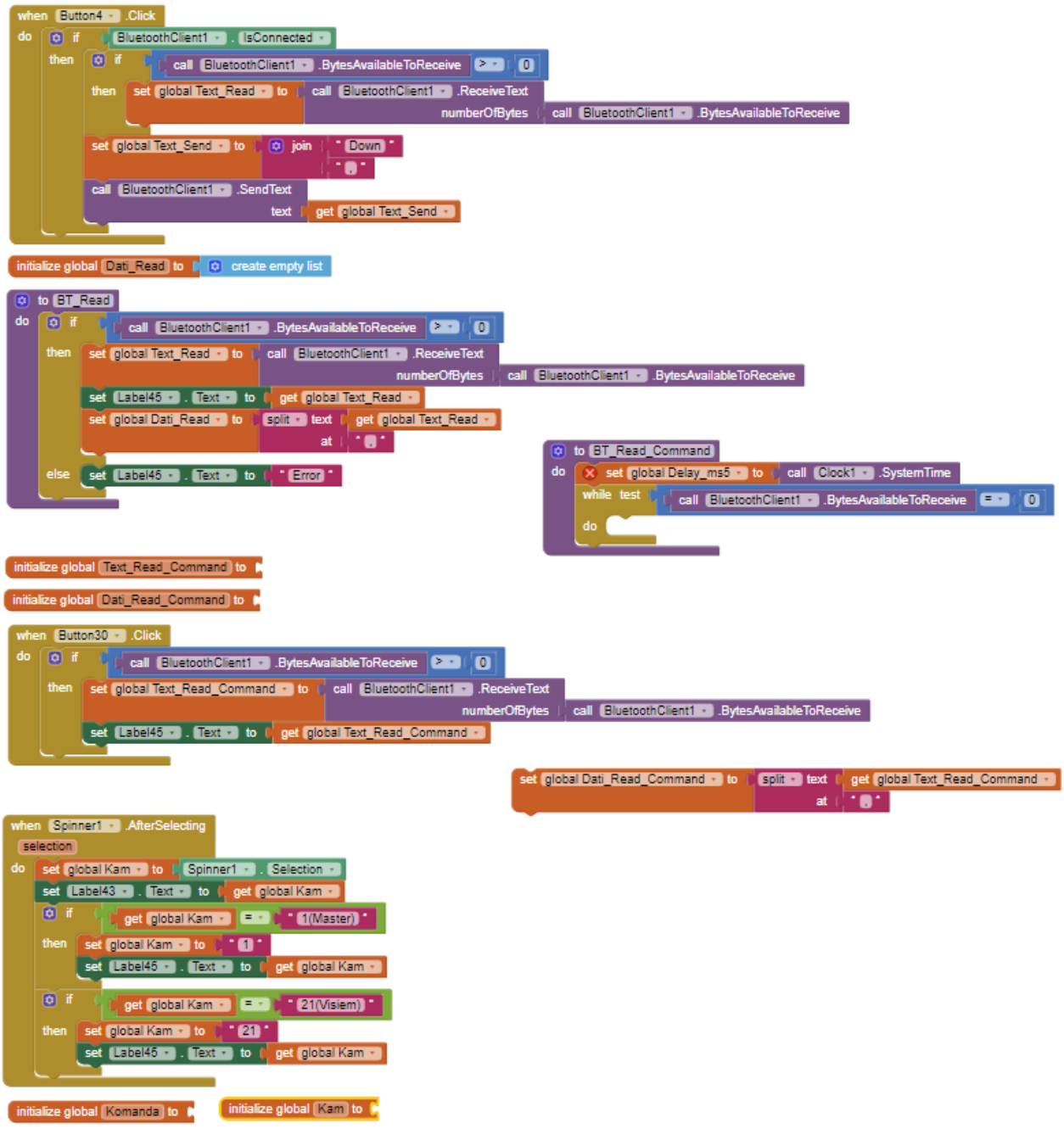

195 мм
HAOS prototips v1.0
Kontrolieru plate
RTU
Radioelektronikas institūts
Jurs Grizāns, 2019
1. Plates materiāls - FR4.
2. Plates biezums - 1,6mm.
3. Vara biezums - 35mkm.
4. Aizsargmaska - abas pusēs.
5. Kontaktlaukumu pārklājums - HAL-SN100C (lead-free).
6. Visi izmēri mm.
145 мм
4,0 мм
4 vietās
Juris Grizāns
Rīgas Tehniskā universitāte. Radioelektronikas institūts
Sheet:
File: Centr_plate.kicad_pcb
Title: Centrālā plate
Size: A4
Date:
Rev:
KiCad E.D.A. kicad (5.0.2)-1
Id: 1/3

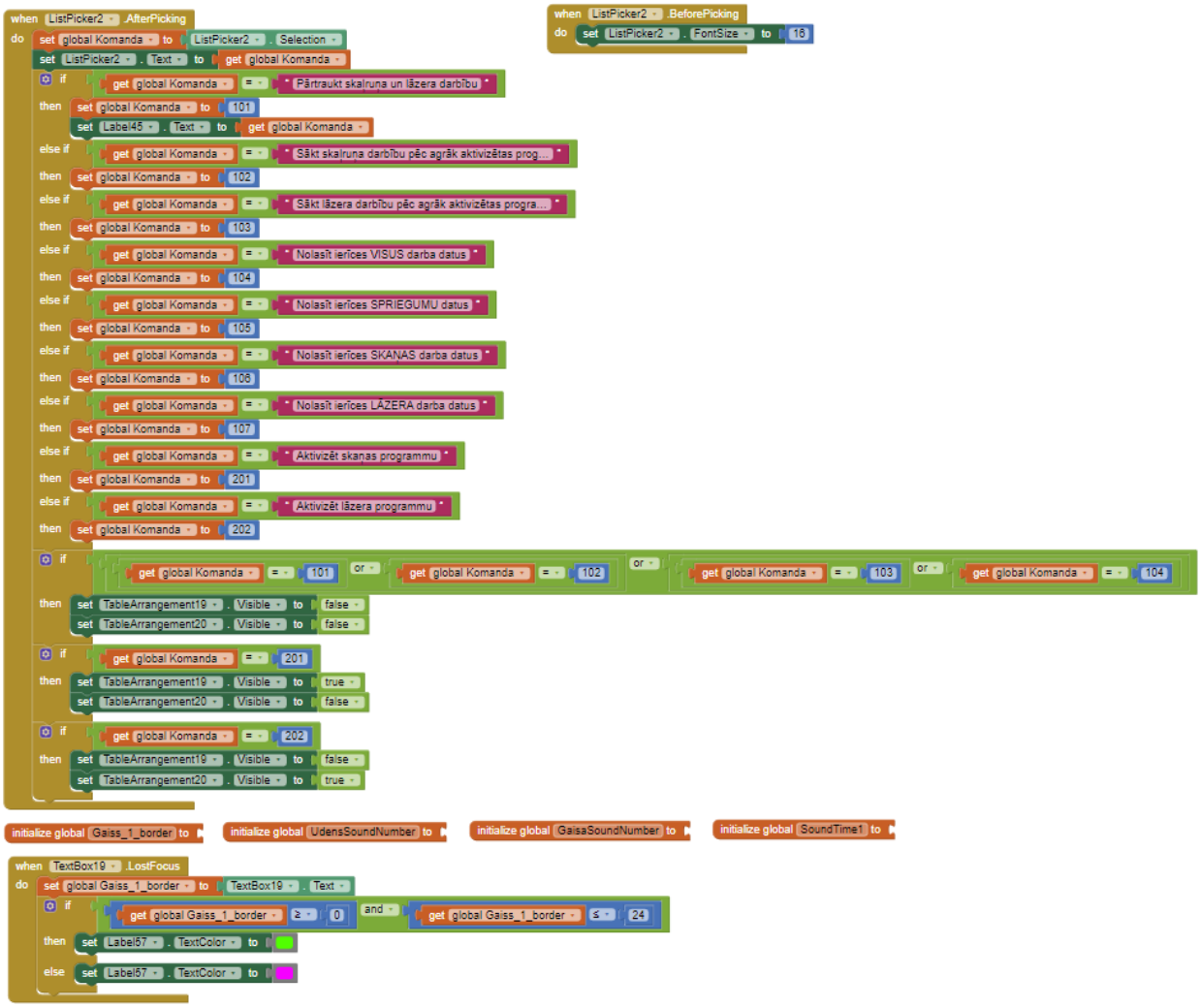

SPI 3
GSM
Power supply
gnd
3.3v
Battery
On
Off
Timer
WDT
RS485 Receive
RS485 transmit
I2C 3.3v
DAC
DAC Output
RS485

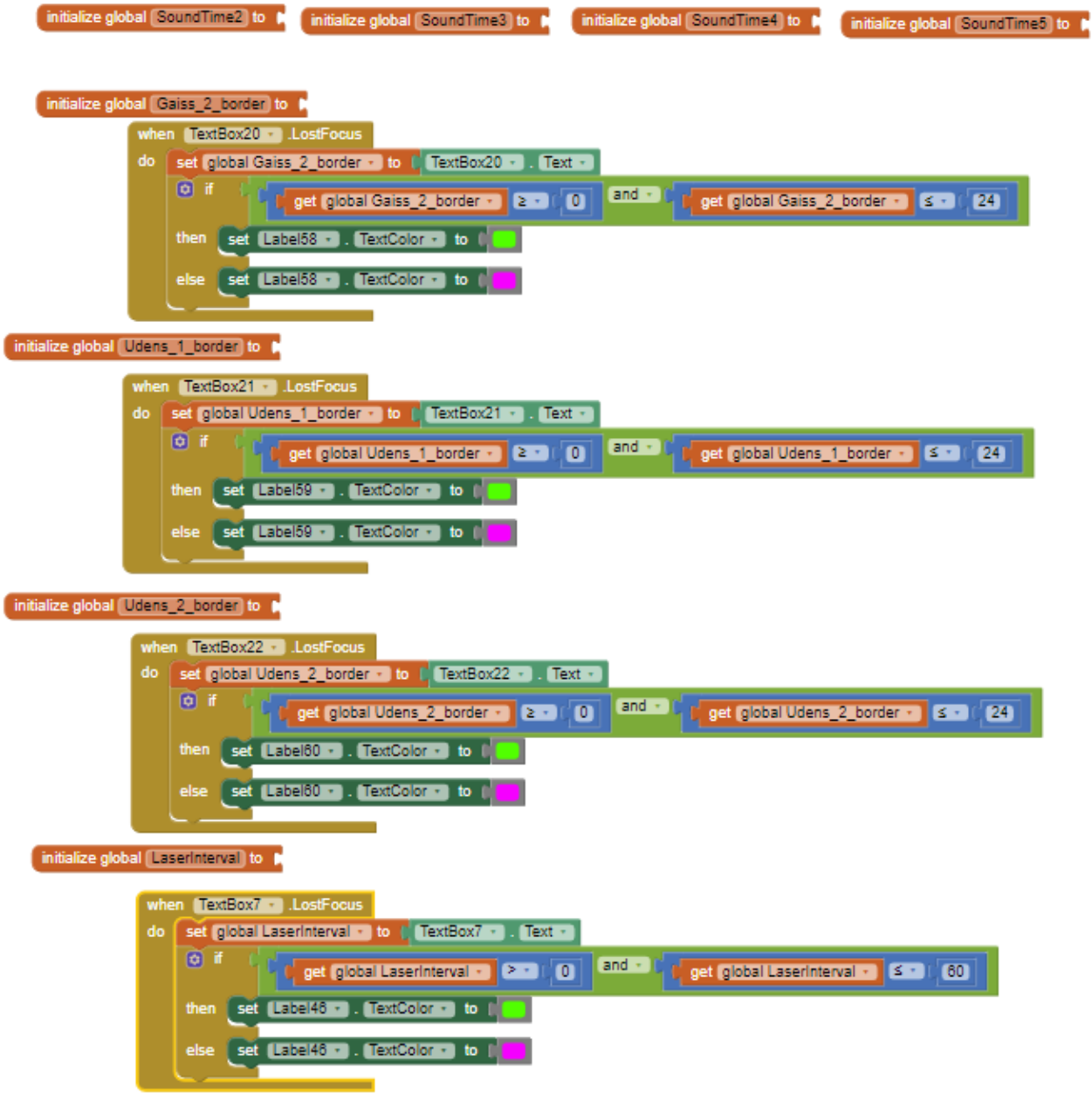
Pielikums 4. Viedtālruņa aplikācijas programma


initialize global BT_Name to
when ListPicker1 .AfterPicking
set global BT_Name to ListPicker1 . Selection
set ListPicker1 . Selection to call BluetoothClient1 .Connect address get global BT_Name
set Label27 . TextColor to
set Label27 . Text to " Соединено с "
set Label28 . TextColor to
set Label28 . Text to get global BT_Name
if call BluetoothClient1 .BytesAvailableToReceive > 0
then set global Text_Read to call BluetoothClient1 .ReceiveText numberOfBytes call BluetoothClient1 .BytesAvailableToReceive
call BluetoothClient1 .SendText text " StartLaserSetup "
when Button21 .Click
do if call BluetoothClient1 .BytesAvailableToReceive > 0
then set global Text_Read to call BluetoothClient1 .ReceiveText numberOfBytes call BluetoothClient1 .BytesAvailableToReceive
call BluetoothClient1 .SendText text " StopLaserSetup "
call BluetoothClient1 .Disconnect
set Label27 . TextColor to
set Label27 . Text to " НЕТ соединения "
set Label28 . TextColor to
initialize global Sector_Number to
initialize global Sector_Parameter to
initialize global Text_Send to
initialize global Text_Read to
when Button18 .Click
do if BluetoothClient1 . IsConnected
then if call BluetoothClient1 .BytesAvailableToReceive > 0
then set global Text_Read to call BluetoothClient1 .ReceiveText numberOfBytes call BluetoothClient1 .BytesAvailableToReceive
set global Text_Send to join " PWM_OFF "
call BluetoothClient1 .SendText text get global Text_Send
when Button17 .Click
do if BluetoothClient1 . IsConnected
then if call BluetoothClient1 .BytesAvailableToReceive > 0
then set global Text_Read to call BluetoothClient1 .ReceiveText numberOfBytes call BluetoothClient1 .BytesAvailableToReceive
set global Text_Send to join " PWM_ON "
call BluetoothClient1 .SendText text get global Text_Send
rmings

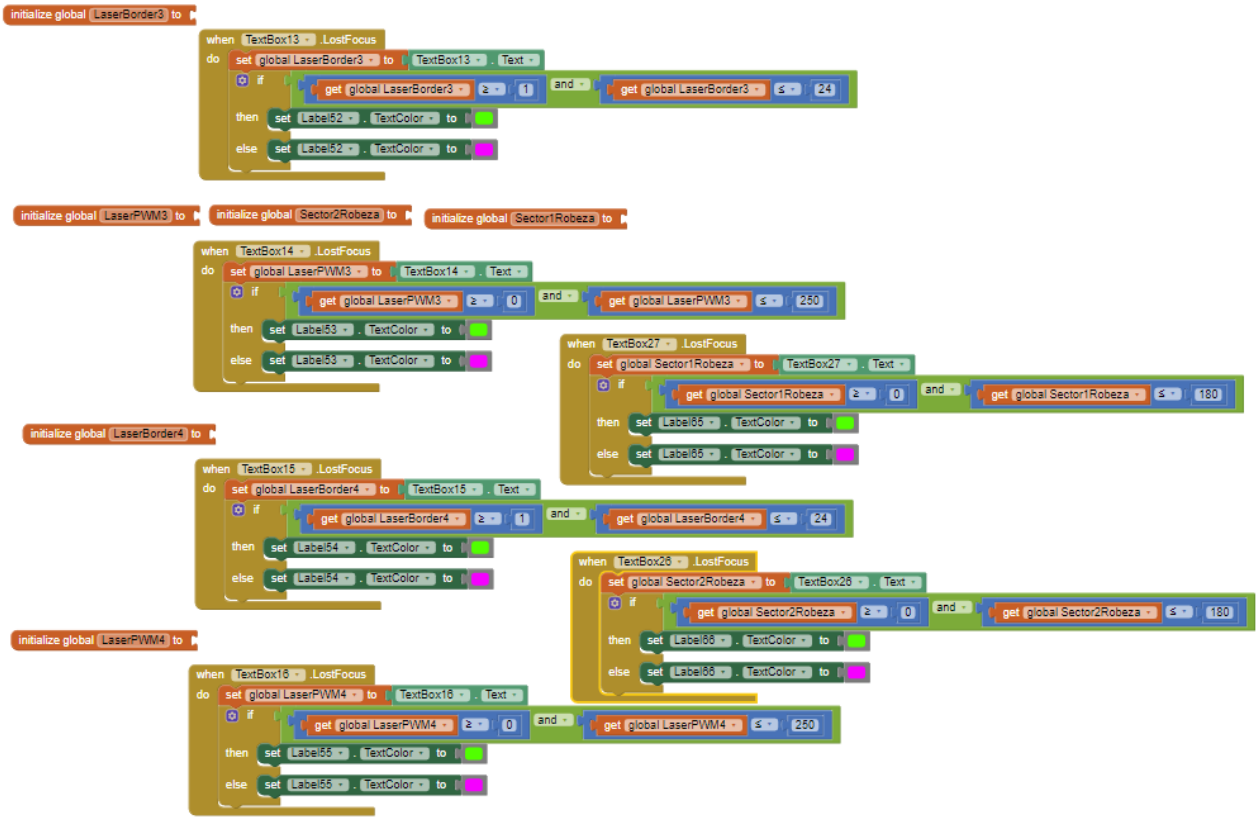

when Button3 .Click
do if BluetoothClient1 . IsConnected
then if call BluetoothClient1 .BytesAvailableToReceive > 0
then set global Text_Read to call BluetoothClient1 .ReceiveText
numberOfBytes call BluetoothClient1 .BytesAvailableToReceive
set global Text_Send to join "Up" ","
call BluetoothClient1 .SendText
text get global Text_Send
when Button2 .Click
do if BluetoothClient1 . IsConnected
then if call BluetoothClient1 .BytesAvailableToReceive > 0
then set global Text_Read to call BluetoothClient1 .ReceiveText
numberOfBytes call BluetoothClient1 .BytesAvailableToReceive
set global Text_Send to join "Stop" ","
call BluetoothClient1 .SendText
text get global Text_Send
when Button6 .Click
do if BluetoothClient1 . IsConnected
then if call BluetoothClient1 .BytesAvailableToReceive > 0
then set global Text_Read to call BluetoothClient1 .ReceiveText
numberOfBytes call BluetoothClient1 .BytesAvailableToReceive
set global Text_Send to join "Left" ","
call BluetoothClient1 .SendText
text get global Text_Send
when Button5 .Click
do if BluetoothClient1 . IsConnected
then if call BluetoothClient1 .BytesAvailableToReceive > 0
then set global Text_Read to call BluetoothClient1 .ReceiveText
numberOfBytes call BluetoothClient1 .BytesAvailableToReceive
set global Text_Send to join "Right" ","
call BluetoothClient1 .SendText
text get global Text_Send

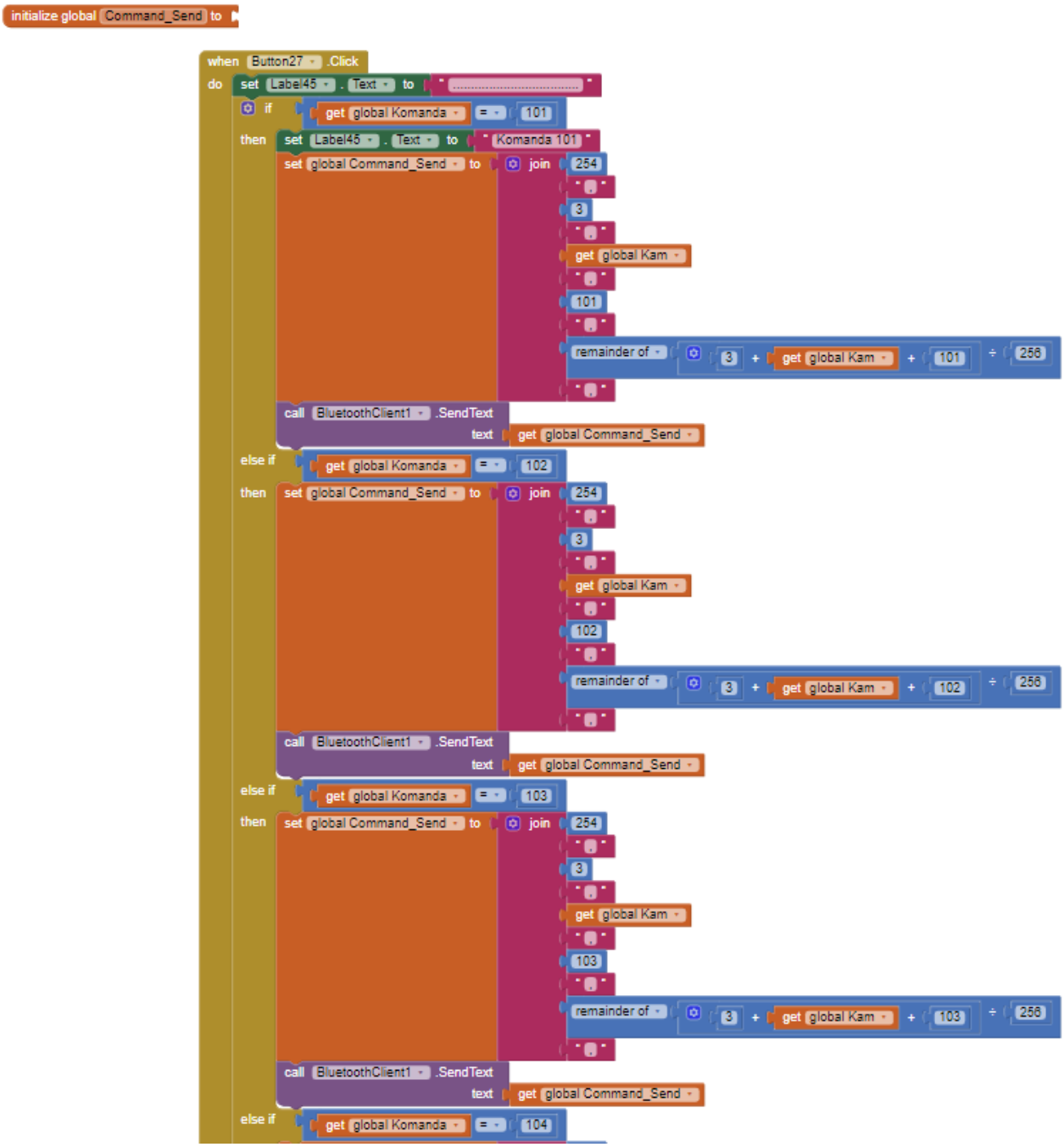

when Button4 .Click
do if BluetoothClient1 . IsConnected
then if call BluetoothClient1 .BytesAvailableToReceive > 0
then set global Text_Read to call BluetoothClient1 .ReceiveText
numberOfBytes call BluetoothClient1 .BytesAvailableToReceive
set global Text_Send to join " Down "
" "
call BluetoothClient1 .SendText
text get global Text_Send
initialize global Dati_Read to create empty list
to BT_Read
do if call BluetoothClient1 .BytesAvailableToReceive > 0
then set global Text_Read to call BluetoothClient1 .ReceiveText
numberOfBytes call BluetoothClient1 .BytesAvailableToReceive
set Label45 . Text to get global Text_Read
set global Dati_Read to split text get global Text_Read
at " , "
else set Label45 . Text to " Error "
to BT_Read_Command
do set global Delay_ms5 to call Clock1 .SystemTime
while test call BluetoothClient1 .BytesAvailableToReceive = 0
do
initialize global Text_Read_Command to
initialize global Dati_Read_Command to
when Button30 .Click
do if call BluetoothClient1 .BytesAvailableToReceive > 0
then set global Text_Read_Command to call BluetoothClient1 .ReceiveText
numberOfBytes call BluetoothClient1 .BytesAvailableToReceive
set Label45 . Text to get global Text_Read_Command
set global Dati_Read_Command to split text get global Text_Read_Command
at " , "
when Spinner1 .AfterSelecting
selection
do set global Kam to Spinner1 . Selection
set Label43 . Text to get global Kam
if get global Kam = " 1(Master) "
then set global Kam to " 1 "
set Label45 . Text to get global Kam
if get global Kam = " 21(Visiem) "
then set global Kam to " 21 "
set Label45 . Text to get global Kam
initialize global Komanda to
initialize global Kam to

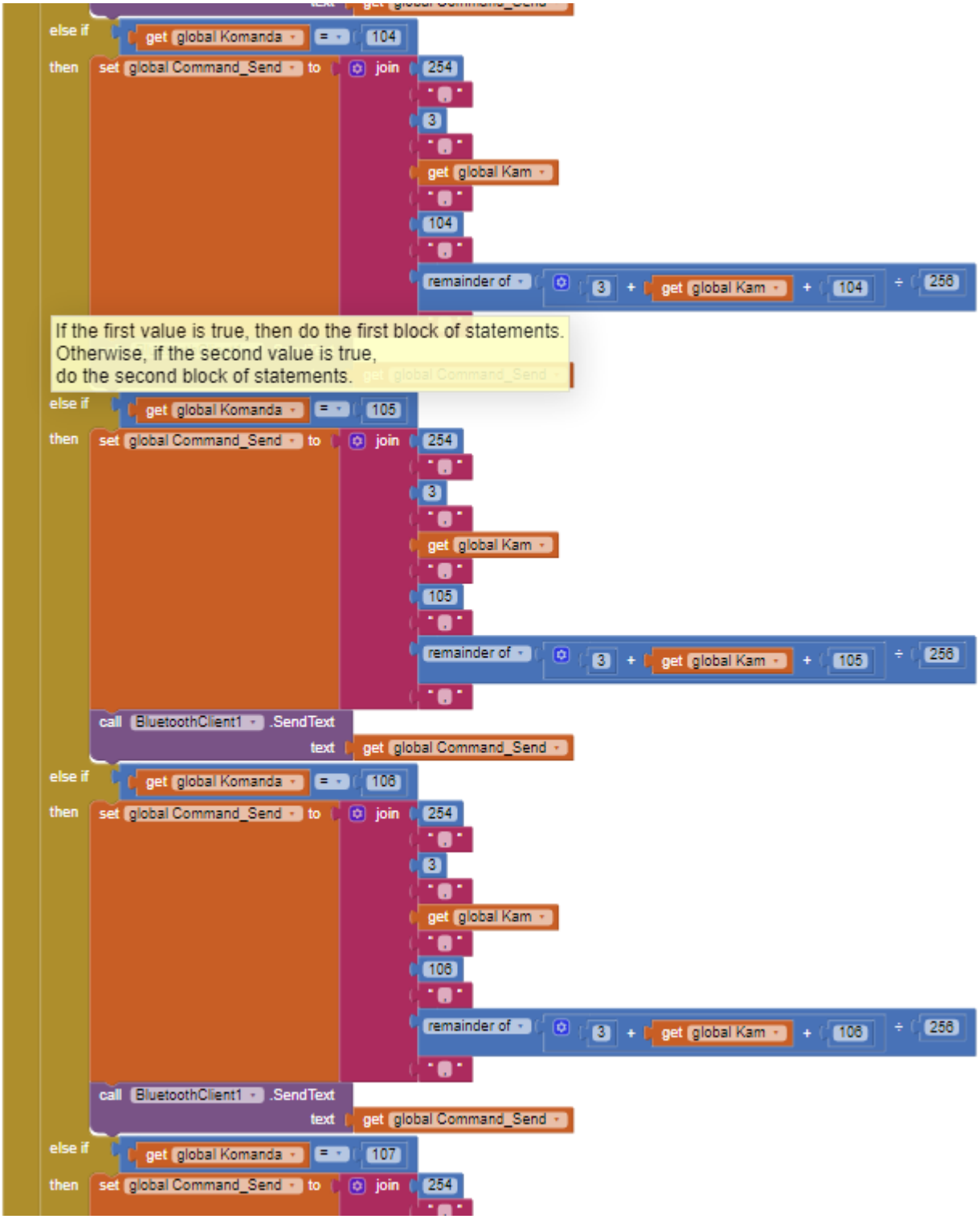

when ListPicker2 .AfterPicking
do set global Komanda to ListPicker2 . Selection
set ListPicker2 . Text to get global Komanda
if get global Komanda = " Pārtraukt skalruņa un lāzera darbību "
then set global Komanda to 101
set Label45 . Text to get global Komanda
else if get global Komanda = " Sākt skalruņa darbību pēc agrāk aktivizētas prog... "
then set global Komanda to 102
else if get global Komanda = " Sākt lāzera darbību pēc agrāk aktivizētas progra... "
then set global Komanda to 103
else if get global Komanda = " Nolasīt ierīces VISUS darba datus "
then set global Komanda to 104
else if get global Komanda = " Nolasīt ierīces SPRIEGUMU datus "
then set global Komanda to 105
else if get global Komanda = " Nolasīt ierīces SKAŅAS darba datus "
then set global Komanda to 106
else if get global Komanda = " Nolasīt ierīces LĀZERA darba datus "
then set global Komanda to 107
else if get global Komanda = " Aktivizēt skaņas programmu "
then set global Komanda to 201
else if get global Komanda = " Aktivizēt lāzera programmu "
then set global Komanda to 202
if get global Komanda = 101 or get global Komanda = 102 or get global Komanda = 103 or get global Komanda = 104
then set TableArrangement19 . Visible to false
set TableArrangement20 . Visible to false
if get global Komanda = 201
then set TableArrangement19 . Visible to true
set TableArrangement20 . Visible to false
if get global Komanda = 202
then set TableArrangement19 . Visible to false
set TableArrangement20 . Visible to true
when ListPicker2 .BeforePicking
do set ListPicker2 . FontSize to 16
initialize global Gaiss_1_border to
initialize global UdensSoundNumber to
initialize global GaisaSoundNumber to
initialize global SoundTime1 to
when TextBox19 .LostFocus
do set global Gaiss_1_border to TextBox19 . Text
if get global Gaiss_1_border ≥ 0 and get global Gaiss_1_border ≤ 24
then set Label67 . TextColor to
else set Label67 . TextColor to

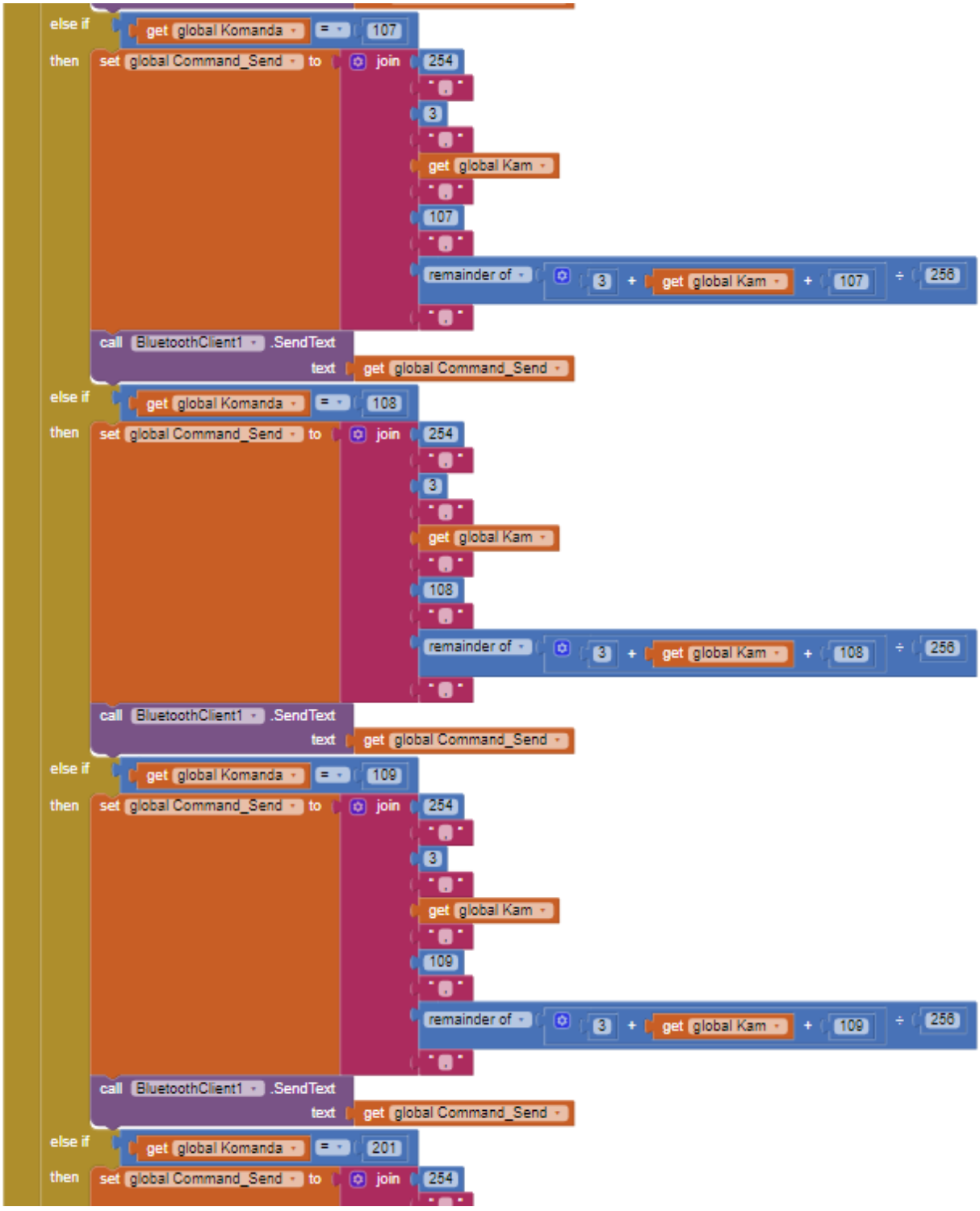

initialize global SoundTime2 to
initialize global SoundTime3 to
initialize global SoundTime4 to
initialize global SoundTime5 to
initialize global Gaiss_2_border to
when TextBox20 .LostFocus
do set global Gaiss_2_border to TextBox20 . Text
if get global Gaiss_2_border ≥ 0 and get global Gaiss_2_border ≤ 24
then set Label58 . TextColor to
else set Label58 . TextColor to
initialize global Udens_1_border to
when TextBox21 .LostFocus
do set global Udens_1_border to TextBox21 . Text
if get global Udens_1_border ≥ 0 and get global Udens_1_border ≤ 24
then set Label59 . TextColor to
else set Label59 . TextColor to
initialize global Udens_2_border to
when TextBox22 .LostFocus
do set global Udens_2_border to TextBox22 . Text
if get global Udens_2_border ≥ 0 and get global Udens_2_border ≤ 24
then set Label60 . TextColor to
else set Label60 . TextColor to
initialize global LaserInterval to
when TextBox7 .LostFocus
do set global LaserInterval to TextBox7 . Text
if get global LaserInterval > 0 and get global LaserInterval ≤ 60
then set Label46 . TextColor to
else set Label46 . TextColor to

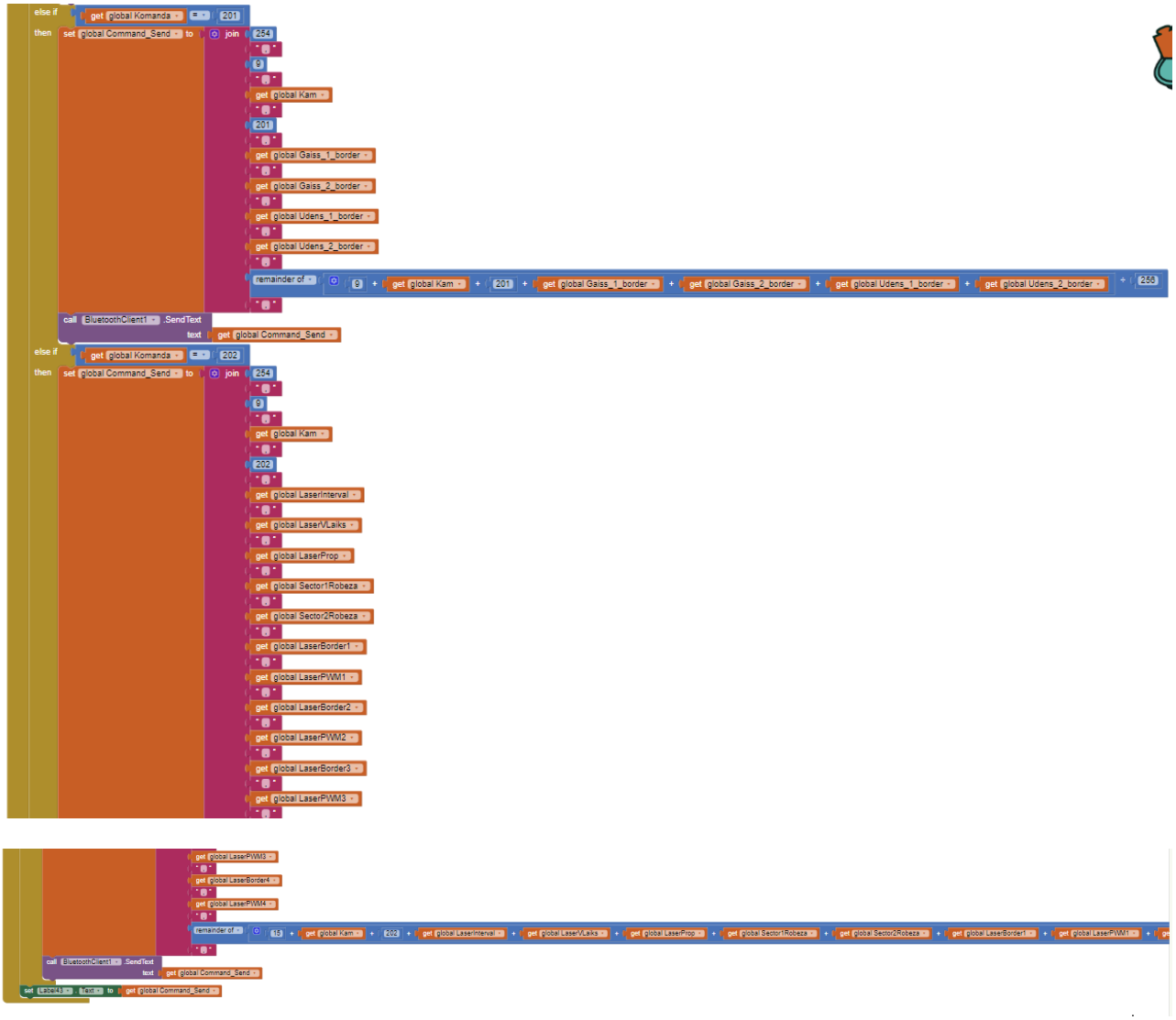

initialize global LaserVLaiks to
when TextBox8 .LostFocus
do set global LaserVLaiks to TextBox8 . Text
if get global LaserVLaiks ≥ 1 and get global LaserInterval ≤ 40
then set Label47 . TextColor to
else set Label47 . TextColor to
initialize global LaserProp to
when TextBox23 .LostFocus
do set global LaserProp to TextBox23 . Text
if get global LaserProp ≥ 1 and get global LaserProp ≤ 20
then set Label62 . TextColor to
else set Label62 . TextColor to
initialize global LaserBorder1 to
when TextBox9 .LostFocus
do set global LaserBorder1 to TextBox9 . Text
if get global LaserBorder1 ≥ 1 and get global LaserBorder1 ≤ 24
then set Label48 . TextColor to
else set Label48 . TextColor to
initialize global LaserPWM1 to
when TextBox10 .LostFocus
do set global LaserPWM1 to TextBox10 . Text
if get global LaserPWM1 ≥ 0 and get global LaserPWM1 ≤ 250
then set Label49 . TextColor to
else set Label49 . TextColor to
initialize global LaserBorder2 to
when TextBox11 .LostFocus
do set global LaserBorder2 to TextBox11 . Text
if get global LaserBorder2 ≥ 1 and get global LaserBorder2 ≤ 24
then set Label50 . TextColor to
else set Label50 . TextColor to
initialize global LaserPWM2 to
when TextBox12 .LostFocus
do set global LaserPWM2 to TextBox12 . Text
if get global LaserPWM2 ≥ 0 and get global LaserPWM2 ≤ 250
then set Label51 . TextColor to
else set Label51 . TextColor to


initialize global LaserBorder3 to
when TextBox13 .LostFocus
do set global LaserBorder3 to TextBox13 . Text
if get global LaserBorder3 ≥ 1 and get global LaserBorder3 ≤ 24
then set Label52 . TextColor to
else set Label52 . TextColor to
initialize global LaserPWM3 to
initialize global Sector2Robeza to
initialize global Sector1Robeza to
when TextBox14 .LostFocus
do set global LaserPWM3 to TextBox14 . Text
if get global LaserPWM3 ≥ 0 and get global LaserPWM3 ≤ 250
then set Label53 . TextColor to
else set Label53 . TextColor to
when TextBox27 .LostFocus
do set global Sector1Robeza to TextBox27 . Text
if get global Sector1Robeza ≥ 0 and get global Sector1Robeza ≤ 180
then set Label65 . TextColor to
else set Label65 . TextColor to
initialize global LaserBorder4 to
when TextBox15 .LostFocus
do set global LaserBorder4 to TextBox15 . Text
if get global LaserBorder4 ≥ 1 and get global LaserBorder4 ≤ 24
then set Label54 . TextColor to
else set Label54 . TextColor to
when TextBox26 .LostFocus
do set global Sector2Robeza to TextBox26 . Text
if get global Sector2Robeza ≥ 0 and get global Sector2Robeza ≤ 180
then set Label66 . TextColor to
else set Label66 . TextColor to
initialize global LaserPWM4 to
when TextBox16 .LostFocus
do set global LaserPWM4 to TextBox16 . Text
if get global LaserPWM4 ≥ 0 and get global LaserPWM4 ≤ 250
then set Label55 . TextColor to
else set Label55 . TextColor to


initialize global Command_Send to
when Button27 .Click
do set Label45 . Text to
if get global Komanda = 101
then set Label45 . Text to Komanda 101
set global Command_Send to join 254 3 get global Kam 101 remainder of 3 + get global Kam + 101 ÷ 256
call BluetoothClient1 .SendText
text get global Command_Send
else if get global Komanda = 102
then set global Command_Send to join 254 3 get global Kam 102 remainder of 3 + get global Kam + 102 ÷ 256
call BluetoothClient1 .SendText
text get global Command_Send
else if get global Komanda = 103
then set global Command_Send to join 254 3 get global Kam 103 remainder of 3 + get global Kam + 103 ÷ 256
call BluetoothClient1 .SendText
text get global Command_Send
else if get global Komanda = 104


else if
get global Komanda = 104
then
set global Command_Send to
join
254
3
get global Kam
104
remainder of
3 + get global Kam + 104 ÷ 256
If the first value is true, then do the first block of statements.
Otherwise, if the second value is true,
do the second block of statements.
else if
get global Komanda = 105
then
set global Command_Send to
join
254
3
get global Kam
105
remainder of
3 + get global Kam + 105 ÷ 256
call BluetoothClient1 .SendText
text
get global Command_Send
else if
get global Komanda = 106
then
set global Command_Send to
join
254
3
get global Kam
106
remainder of
3 + get global Kam + 106 ÷ 256
call BluetoothClient1 .SendText
text
get global Command_Send
else if
get global Komanda = 107
then
set global Command_Send to
join
254


else if
get global Komanda = 107
then
set global Command_Send to join 254 3 get global Kam 107
remainder of 3 + get global Kam + 107 ÷ 256
call BluetoothClient1 .SendText
text get global Command_Send
else if
get global Komanda = 108
then
set global Command_Send to join 254 3 get global Kam 108
remainder of 3 + get global Kam + 108 ÷ 256
call BluetoothClient1 .SendText
text get global Command_Send
else if
get global Komanda = 109
then
set global Command_Send to join 254 3 get global Kam 109
remainder of 3 + get global Kam + 109 ÷ 256
call BluetoothClient1 .SendText
text get global Command_Send
else if
get global Komanda = 201
then
set global Command_Send to join 254



Pielikums 5. Kontroliera bloka programma

projekts „Hibrīdās intelektuālās akustiski-optiskās sistēmas izstrāde nemedījamo un migrējošu putnu sugu nodarīto postījumu samazināšanai Latvijas akvakultūras nozarē”
Nr. 17-00-F02201-000001

Sistēmas centrāla kontroliera programma.
1.9 versija.

```
  Juris Grizāns, 2019

  Jauns algoritms: lazera pārvietošanas notiek vertikalā virzienā ar horizontālu soli 5 grādi.
*/

#include <Arduino.h>

#include "BluetoothSerial.h"
#include "esp32-hal-ledc.h" // PWM
#include <EEPROM.h>

#include <SPI.h>
#include <SD.h>

#include <Wire.h>
#include <RTClib.h>

#include <LoRa.h>

RTC_DS1307 rtc;

const byte EEPROM_Size = 150;

//=========================================================================
String Module_Tipe = "MASTER"; //  MASTER; SLAVE_LASER; SLAVE_WATER
byte Module_Number = 1; // Master vienmēr ir PIRMAIS!
String BT_Name = "ESP32_Putni_MASTER_4"; // ESP32_Putni_SLAVE_WATER_2 ;// ESP32_Putni_MASTER_1;
//=========================================================================



BluetoothSerial ESP_BT; // Bluetooth objekts
String Text_Read_BT;
unsigned long Laser_BT_Time1 = 0; // Pārvietošanas sakuma laiks, ms
unsigned long Laser_BT_Time2 = 0; // Pārvietošanas sakuma laiks, ms
int Hi = 0; // Horizontāla koordināte lazera kalibrēšanai Bluetooth režīmā
int Vi = 0; // Vertikāla koordināte lazera kalibrēšanai Bluetooth režīmā
int Hmax = 300;
int Vmax = 90;
String ParameterBT = " ";
String BT_Commands [40];


const byte Volt_12V_PIN = 36;
const byte Volt_4V_PIN = 39;
const byte Temper_PIN = 34;
const byte GSM_Int_PIN = 35;
const byte DDS_Reset_PIN = 32;
const byte DDS_Data_PIN = 33;
const byte DAC1_PIN = 25;
const byte SPI3_CS_PIN = 26;
const byte DDS_W_Clk_PIN = 27;
const byte DDS_FU_UD_PIN = 14;
const byte GSM_CS_PIN = 12;
const byte Bat_EN_PIN = 13;
const byte MOSI_PIN = 23;
const byte I2C_SCL_PIN = 22;
const byte TX0_PIN = 1;
const byte RX0_PIN = 3;
const byte I2C_SDA_PIN = 21;
```

```
const byte MISO_PIN = 19;
const byte CLK_SPI_PIN = 18;
const byte CLK_EN_PIN = 5;
const byte TX2_PIN = 17;
const byte RX2_PIN = 16;
const byte LoRa_CS_PIN = 4;
const byte DIO0_PIN = 2;
const byte LoRa_RESET_PIN = 15;
const byte SD_CS_PIN = 0;

// ************************************ SKAŅA (sakums) ******************************************
byte Sound_Air_Number = 1; // Skaņas masīva numurs gaisā
byte Sound_Woter_Number = 2; // Skaņas masīva numurs ūdenī

byte Sound_Air_Numbers_Array [] = {13,0,0,0,7, 0,0,15,0,0, 9,0,0,0,11, 0,0,13,0,5, 0,0,0,0,0,0,0,0,0};
byte Sound_Woter_Numbers_Array [] = {2,4,6,8, 10,12,14,16,18, 20,22,4,2,6, 8,10,20,18,22, 0,0,0,0,0,0,0,0,0};
byte Sound_Number = 0;
byte Sound_Number_Max = 20;

//byte Sound_Loudspeaker; // 0 - gaisa skaļrunis; 1 - ūdens skaļrunis
const int Sound_Point_Number_Max = 20000; // maksimālais skaņās masīva punktu skaits
int Sound_Point_Number; // aktīva skaņās masīva punktu skaits
byte Sound_Package [Sound_Point_Number_Max + 5][3]; // 0-laiks; 1-frekvence; 2-amplitūda
const byte Ampl_Low = 125; //166; // amplitūdas atļautas vērtības diapazons
const byte Ampl_High = 165; //250;

byte Sound_Start_Time [5] = {0, 20, 60, 60, 60}; // Laika punkti, kas ierakstīti ar komandu no servera !!!!! НУЖНА ПРОВЕРКА - ЧИСЛА ДОЛЖНЫ ИДТИ В ПОРЯДКЕ ВОЗРАСТАНИЯ !!!!!!
//int Sound_Time_Number = 2; // skaļruņa darbības laiku punktu skaits
byte Sound_Start_Random [5]; // Laika punkti, kas nobīdīti nejaušā kārtībā
byte Sound_Start_Random_Status [5]; // 0 - laika punkts nav nostrādāts
unsigned long Sound_Time1; // sekundes intervāla sakuma laiks
unsigned long Sound_Time2; // sekundes intervāla tagadējais laiks
int Sound_Count; // skaņas punkta skaitītājs
//const int Sound_Interval = 1000; // nepārtrauktas skaņas intervāls
byte Sound_Time_Second; // Tekošais laiks sekundēs

byte Loudspeaker_Time;
byte Gaiss_1_Border = 10; // Laika robežas, kad var strādāt gaisa skaļrunis
byte Gaiss_2_Border = 20;
byte Udens_1_Border = 0; // Laika robežas, kad var strādāt ūdens skaļrunis
byte Udens_2_Border = 24;
// ************************************ SKAŅA (beigas) ******************************************


// ******************************************************
//         Karogu saraksts
// ------------------------------------------------------
byte Sound_Flag = 0; // 0 - nestrāda dotājā laikā; 1 - strāda
byte Sound_Status = 1; // 4 - NAV atļauts strādāt; 1 - IR atļauts strādāt
//byte Random_Flag = 1; // iepriešejā skaņu nejaušā secība vēl NAV izpildīts pilnībā

// ******************************************************


// ************************************ LĀSERS (sakums) ******************************************
const int Right = B00000010; // Komandas maska
const int Left  = B00000100; // Komandas maska
const int Up    = B00001000; // Komandas maska
```

```
const int Down  = B00010000; // Komandas maska
const int Stop  = B00000000; // Komandas maska

const int SpeedRL = 0x3F; // Griešanas ātrums, kods servodzīnejam
const int SpeedDU = 0x3F; // Liekšanas ātrums, kods servodzīnejam
const int Addr = 0x01; // Ierīces adrese

const float Laser_Speed_H = 8.2; // Pārvietošanas ātrums, grad/sek (Datasheet -> 7.5)
const float Laser_Speed_V = 5.2; // Pārvietošanas ātrums, grad/sek (Datasheet -> 6)

const long Laser_Parking_Time = 30000; // Laiks parkingam intervāls, ms
const long Laser_Vertical_Time = 20000; // Laiks darba intervāls, ms

int Laser_Angle_H = 0; // Tekošais lazera horizontālais leņķis

//int Command;

byte Laser_Move_Direction = 0; // 0 - pa labi; 1 - pa kreisi
int Laser_Zone = 4; // Cik zonas katrā sektorā
int Laser_Step_V; // Soļa lielums vertikālā virzienā
int Laser_Step_H; // Soļa lielums horizontālā virzienā
int Laser_Sector_Number = 1; // Tekoša sektora numurs
int Laser_Sector_Number_Max = 1; // Darba sektoru skaits
//int Laser_Step_Sum = 0;
unsigned long Laser_Time1 = 0; // Pārvietošanas sakuma laiks, ms

byte Laser_Status = 0; // 0 - nestrādā(gaidīšanas režīms); 1- kustības kartes ģenerēšana; 2 - parking; 3 - PROGRAMMAS IZPILDE; 4 - aizlēgts strādāt; 5
byte Laser_PWM = 10; // Lāzera spilgtums: 0 - nespīd; 250 - pilna jauda

byte Laser_Time_Border1 = 0;
byte Laser_Time_Border2 = 2;
byte Laser_Time_Border3 = 0;
byte Laser_Time_Border4 = 2;
byte Laser_PWM_Border1 = 1;
byte Laser_PWM_Border2 = 1;
byte Laser_PWM_Border3 = 1;
byte Laser_PWM_Border4 = 1;

byte Laser_Delay_Time = 1; // laika intervāls starp programmas izpildīšanas, min
unsigned long Laser_Time2 = 0; // Starpprogrammu intervāla sakuma laiks, ms

int Laser_Sector1_H1 = 1; // Testa cipari
int Laser_Sector1_H2 = 30;
int Laser_Sector1_V1 = 10;
int Laser_Sector1_V2 = 40;

int Laser_Sector2_H1 = 30;
int Laser_Sector2_H2 = 50;
int Laser_Sector2_V1 = 10;
int Laser_Sector2_V2 = 50;

int Laser_Sector3_H1 = 50;
int Laser_Sector3_H2 = 80;
int Laser_Sector3_V1 = 10;
int Laser_Sector3_V2 = 20;

byte Laser_Karte [50][2]; // Kustības karte [i][0]-Vi; [i][1]-Hi
long Laser_Work_Time = 0; // Laiks, kas ir nepieciešams kartajai pārvietošanai
byte Laser_Nr = 1; // Lāzera kartes punkta numurs
byte Laser_j=1;
```

```
// ************************************ LASER (beigas) ***************************************

float Temperatura;
byte Battery_Status=0; // 0-uzladēšana ir izslegta; 1-uzladēšana ir iezslegta
float Volt_12V;
float Volt_4V;

// ************************************ LoRa (sakums) ****************************************
byte LoRaSyncWord = 0x64;
int flag=0; //
unsigned long LoRa_Time = millis();
String LoRa_Message;
String LoRa_Commands [40];
// ************************************ LoRa (beigas) ****************************************

// ************************************ Log (sakums) *****************************************
byte Log_Flag=0;
byte Log_Time = 0;
String LogFileName = "/Logs/Voltage.log";
// ************************************ Log (beigas) *****************************************

#define pulseHigh(pin) {digitalWrite(pin, HIGH); digitalWrite(pin, LOW); }

void WDT_Reset(void)
{
  //digitalWrite(CLK_EN_PIN, HIGH);
  //delay(300);
  //digitalWrite(CLK_EN_PIN, LOW);
}

void tfr_byte(byte data)
{
 for (int i=0; i<8; i++, data>>=1) {
   digitalWrite(DDS_Data_PIN, data & 0x01);
   pulseHigh(DDS_W_Clk_PIN);   //after each bit sent, CLK is pulsed high
 }
}

void sendFrequency(double frequency)
{
 int32_t freq = frequency * 4294967295/125000000;  // note 125 MHz clock on 9850
 for (int b=0; b<4; b++, freq>>=8) {
   tfr_byte(freq & 0xFF);
 }
 tfr_byte(0x000);   // Final control byte, all 0 for 9850 chip
 pulseHigh(DDS_FU_UD_PIN);  // Done!  Should see output
}

void Sound_Package_Clear(void)
{
 Serial.println(" Sound_Package_Clear ");
 for(int i=0; i<=Sound_Point_Number_Max+4; i=i+1)
   {
     Sound_Package [i][0] = 0;
     Sound_Package [i][1] = 0;
     Sound_Package [i][2] = 0;

   }
}
```

```
void SD_Log(void)
{
  DateTime now = rtc.now();
  //byte Log_Flag=0;
  Log_Time = now.minute();
  if(Log_Time>1 && Log_Time<10 && Log_Flag==0)
   {
     Serial.println("================== SD Log write START =================");
     Serial.println("LogFileName = " + LogFileName);

     File myFile = SD.open(LogFileName, FILE_APPEND);
     if (myFile)
      {
        Serial.println (" SD write START ");
        byte Log_Year = now.year();
        byte Log_Month = now.month();
        byte Log_Day = now.day();
        byte Log_Hour = now.hour();
        byte Log_minute = now.minute();
        myFile.println(String(Log_Day)+"."+String(Log_Month)+"."+String(Log_Year)+"  "+String(Log_Hour)+":"+String(Log_minute)+"  12V -> "+String(Volt_12V)+" 4V -> "+String(Volt_4V));

      }
     Log_Flag=1;
     myFile.close();
   }
  if(Log_Time>10) Log_Flag=0;
}



void SD_Sound_Read(byte SN)
{
  unsigned long Time1 = 0;
  unsigned long Time2 = 0;

  Sound_Package_Clear();
  Serial.println("================== SD Read START =================");
  String SoundFileName = "/Sound/" + String(SN) + ".dds";
  Serial.println("SoundFileName = " + SoundFileName);


  Time1 = millis();
  File myFile = SD.open(SoundFileName, FILE_READ); //
  if (myFile)
    {
      Serial.println (" SD read START ");
        Sound_Package [0][0]=myFile.read();
        Sound_Package [0][1]=myFile.read();
        Sound_Package [0][2]=myFile.read();
        Sound_Point_Number=(((Sound_Package [0][2]<<8)&0xFFFF)+(Sound_Package [0][1]&0xFF));
        if(Sound_Point_Number > Sound_Point_Number_Max)
         {
           Serial.print("Sound Point Number is too big  -> ");
           Serial.println(Sound_Point_Number);
           Sound_Point_Number = Sound_Point_Number_Max;
         }
        Serial.print (" Sound_Point_Number = ");
        Serial.println (Sound_Point_Number);
```

```
          for(int i=1; i<=Sound_Point_Number; i=i+1)
            {
               Sound_Package [i][0]=myFile.read();
               Sound_Package [i][1]=myFile.read();
               Sound_Package [i][2]=myFile.read();
             }
          Sound_Package [Sound_Point_Number+1][0]=myFile.read();   // faila beigas zīmes (253????) nolasīšana
        //}
      Serial.println(String(Sound_Package [1][0])+"-"+String(Sound_Package [1][1])+""+String(Sound_Package [2][0]));
      Serial.println (" SD read STOP ");
      Serial.print(" Fail Size = ");
      Serial.println(myFile.size());

      myFile.close();
    } else
       {
          Serial.println("error opening sound file!!!");
       }
   Time2 = millis();
  Serial.print("Laiks = ");
  Serial.print(Time2 - Time1);
  Serial.println(" ms ");
  Serial.println("================== SD Read END =================");
}

void Sound_Play (void)
{
  Serial.println("===== Sound Play =======");

  DateTime now = rtc.now();
  Loudspeaker_Time = now.hour();
  Serial.println("Loudspeaker_Time="+String(Loudspeaker_Time)+" st.");
  Serial.println("Gaiss_1_Border="+String(Gaiss_1_Border)+" st.");
  Serial.println("Gaiss_2_Border="+String(Gaiss_2_Border)+" st.");


  WDT_Reset();

     Sound_Air_Number = Sound_Air_Numbers_Array[Sound_Number];
     Sound_Woter_Number = Sound_Woter_Numbers_Array[Sound_Number];
     Serial.println("Sound_Number="+String(Sound_Number)+"                    Sound_Air_Number="+String(Sound_Air_Number)+"
Sound_Woter_Number="+String(Sound_Woter_Number));

     if(Module_Tipe == "MASTER")
      {
        if (Loudspeaker_Time >= Gaiss_1_Border && Loudspeaker_Time < Gaiss_2_Border)
        {
         if(Sound_Air_Number != 0)
         {
         Serial.println("Sound for air (MASTER)");
         SD_Sound_Read(Sound_Air_Number);

         Serial.println("Sound_Point_Number="+String(Sound_Point_Number));

         for(int i=1; i<Sound_Point_Number-1; i++)
          {
            sendFrequency(Sound_Package [i][1]*20);
            dacWrite(DAC1_PIN,map(Sound_Package [i][2], 1, 252, Ampl_Low, Ampl_High));
```

```
        delay(Sound_Package [i][0]);
       }
     sendFrequency(0);
     dacWrite(DAC1_PIN,0);

     WDT_Reset();
     }
    }
    else Serial.println(" Sound time outside (air for MASTER)");

    if (Sound_Number < Sound_Number_Max) Sound_Number = Sound_Number + 1;
     else Sound_Number = 0;
  }

if(Module_Tipe == "SLAVE_LASER")
  {
    if (Loudspeaker_Time >= Gaiss_1_Border && Loudspeaker_Time < Gaiss_2_Border)
     {
      if(Sound_Air_Number != 0)
      {
        digitalWrite (GSM_CS_PIN, 1); // Amplifier is ON
      delay(2000);
      Serial.println("Sound for air (SLAVE)");
      SD_Sound_Read(Sound_Air_Number);

      for(int i=1; i<Sound_Point_Number-1; i++)
       {
        sendFrequency(Sound_Package [i][1]*20);
        dacWrite(DAC1_PIN,map(Sound_Package [i][2], 1, 252, Ampl_Low, Ampl_High));
        delay(Sound_Package [i][0]);
       }
      sendFrequency(0);
      dacWrite(DAC1_PIN,0);
      digitalWrite (GSM_CS_PIN, 0); // Amplifier is OFF

      WDT_Reset();
      }
     }
     else Serial.println(" Sound time outside (air for SLAVE_LASER)");

     if (Sound_Number < Sound_Number_Max) Sound_Number = Sound_Number + 1;
      else Sound_Number = 0;
  }

if(Module_Tipe == "SLAVE_WATER")
   {
     if (Loudspeaker_Time >= Gaiss_1_Border && Loudspeaker_Time < Gaiss_2_Border)
      {
       if(Sound_Air_Number != 0)
        {
         digitalWrite (GSM_CS_PIN, 1); // Amplifier is ON
         delay(2000);
         digitalWrite (SPI3_CS_PIN, 0); // 0 - gaisa skaļrunis; 1 - ūdens skaļrunis
         Serial.println("Sound for air");
         SD_Sound_Read(Sound_Air_Number);

         for(int i=1; i<Sound_Point_Number-1; i++)
          {
            sendFrequency(Sound_Package [i][1]*20);
            dacWrite(DAC1_PIN,map(Sound_Package [i][2], 1, 252, Ampl_Low, Ampl_High));
```

```
              delay(Sound_Package [i][0]);
             }
            WDT_Reset();
           }
         }
         else Serial.println(" Sound time outside (air for SLAVE_WATER)");

        if (Loudspeaker_Time >= Udens_1_Border && Loudspeaker_Time < Udens_2_Border)
         {
          if(Sound_Woter_Number != 0)
           {
            digitalWrite (SPI3_CS_PIN, 1); // 0 - gaisa skaļrunis; 1 - ūdens skaļrunis
            Serial.println("Sound for water");
            SD_Sound_Read(Sound_Woter_Number);
            digitalWrite (GSM_CS_PIN, 1); // 0 - gaisa skaļrunis; 1 - ūdens skaļrunis
            for(int i=1; i<Sound_Point_Number-1; i++)
             {
              sendFrequency(Sound_Package [i][1]*20);
              dacWrite(DAC1_PIN,map(Sound_Package [i][2], 1, 252, Ampl_Low, Ampl_High));
              delay(Sound_Package [i][0]);
             }

            WDT_Reset();
           }
         }
         else Serial.println(" Sound time outside (water for SLAVE_WATER)");
         sendFrequency(0);
         dacWrite(DAC1_PIN,0);
         digitalWrite (GSM_CS_PIN, 0); // Amplifier is OFF
         if (Sound_Number < Sound_Number_Max) Sound_Number = Sound_Number + 1;
         else Sound_Number = 0;

      }


}


void BT_Parsing(String var)
{
  int b_max=var.lastIndexOf(",");
  int a=0;
  int b=0;
  int i=0;
  Serial.println("******** BT Parsing**********");
  Serial.println(" Elementu skaits = "+String(b_max));
  Serial.println("b_max="+String(b_max));
   WDT_Reset();
  while (b<b_max)
  {
    b=var.indexOf(",", a);
    BT_Commands [i]=var.substring(a,b);
    Serial.println(String(i)+" - "+BT_Commands [i]);
    i++;
    a=b+1;
  }
}

void LoRa_Parsing(String var)
{
```

```
  int b_max=var.lastIndexOf(",");
  int a=0;
  int b=0;
  int i=0;
  Serial.println("******** LoRa Parsing**********");
  Serial.println(" Elementu skaits = "+String(b_max));
  Serial.println("b_max="+String(b_max));
   WDT_Reset();
  while (b<b_max)
  {
    b=var.indexOf(",", a);
    LoRa_Commands [i]=var.substring(a,b);
    Serial.println(String(i) + " - " + LoRa_Commands [i]);
    i++;
    a=b+1;
  }
}

void Laser_Power(String PWM)
{
  if (Module_Tipe == "MASTER" || Module_Tipe == "SLAVE_LASER")
    {
     if (PWM=="OFF")
      {
        Serial.print(" OFF ");
        ledcWrite(0, 0);
      }
     if (PWM=="FULL")
      {
       Serial.print(" FULL ");
       ledcWrite(0, 250);
      }
     if (PWM=="ON")
      {
        DateTime now = rtc.now();
        byte Laser_Time_Hour = now.hour();
     Serial.print(" ON ("+String(Laser_Time_Hour)+"hour  ");
     if(Laser_Time_Hour>=Laser_Time_Border1 && Laser_Time_Hour<Laser_Time_Border2) // rīts
       {
         ledcWrite(0, Laser_PWM_Border1);
         Serial.print("PWM="+String(Laser_PWM_Border1)+" ) ");
       }
     if(Laser_Time_Hour>=Laser_Time_Border2 && Laser_Time_Hour<Laser_Time_Border3) // diena
       {
         ledcWrite(0, Laser_PWM_Border2);
         Serial.print("PWM="+String(Laser_PWM_Border2)+" ) ");
       }
     if(Laser_Time_Hour>=Laser_Time_Border3 && Laser_Time_Hour<Laser_Time_Border4) // vakars
       {
         ledcWrite(0, Laser_PWM_Border3);
         Serial.print("PWM="+String(Laser_PWM_Border3)+" ) ");
       }
     if((Laser_Time_Hour>=Laser_Time_Border4 && Laser_Time_Hour<24) || (Laser_Time_Hour>=0 && Laser_Time_Hour<Laser_Time_Border1)) //
nakts
       {
         ledcWrite(0, Laser_PWM_Border4);
         Serial.print("PWM="+String(Laser_PWM_Border4)+" ) ");
       }

   }
```

```
    }

}

void SPI_Close_All(void)
{
  if (Module_Tipe=="MASTER")
    {
       digitalWrite(GSM_CS_PIN, HIGH);
       Serial.println(" Modulis ir MASTER --> GSM_CS_PIN = HIGH");
    }
    else
     {
       digitalWrite(GSM_CS_PIN, LOW);
       Serial.println(" Modulis nav MASTER --> GSM_CS_PIN = LOW");
     }

  digitalWrite(LoRa_CS_PIN, HIGH);
  digitalWrite(SD_CS_PIN, HIGH);
}

void SendData_PelcoD(byte command_2)
{
   if (command_2 == Up) Serial.print(" Up ");
   if (command_2 == Down) Serial.print(" Down ");
   if (command_2 == Stop) Serial.print(" Stop ");
   if (command_2 == Right) Serial.print(" Right ");
   if (command_2 == Left) Serial.print(" Left ");
   int command_1 = 0x00;
   int CheckSum = 0;
   CheckSum = Addr + command_1 + command_2 + SpeedRL + SpeedDU;
   CheckSum %= 256;
   Serial2.write(0xFF);
   Serial2.write(Addr);
   Serial2.write(command_1);
   Serial2.write(command_2);
   Serial2.write(SpeedRL);
   Serial2.write(SpeedDU);
   Serial2.write(CheckSum);

}

void SendData_PelcoD_Moving(byte command_5)
{
   byte command_3 = command_5;
   DateTime now = rtc.now();
   byte Moving_Time_Hour = now.hour();

   if(Moving_Time_Hour>=Laser_Time_Border1 && Moving_Time_Hour<Laser_Time_Border2) // rīts
     {
       if(Laser_PWM_Border1 != 0) SendData_PelcoD(command_3);
     }
   if(Moving_Time_Hour>=Laser_Time_Border2 && Moving_Time_Hour<Laser_Time_Border3) // diena
     {
       if(Laser_PWM_Border2 != 0) SendData_PelcoD(command_3);
     }
   if(Moving_Time_Hour>=Laser_Time_Border3 && Moving_Time_Hour<Laser_Time_Border4) // vakars
     {
       if(Laser_PWM_Border3 != 0) SendData_PelcoD(command_3);
     }
```

```
  if((Moving_Time_Hour>=Laser_Time_Border4 && Moving_Time_Hour<24) || (Moving_Time_Hour>=0 && Moving_Time_Hour<Laser_Time_Border1)) // nakts
    {
      if(Laser_PWM_Border4 != 0) SendData_PelcoD(command_3);
    }

}



void Laser_Parking_Left(void)
{
  Laser_Power("OFF"); // Lāsers nespīd
  Laser_Status = 5; // parking
  Laser_Time1 = millis();

  SendData_PelcoD(Left | Down);

  Serial.println(" Parking (LEFT) start time = "+String(Laser_Time1));
}

void Laser_Parking_Right(void)
{
  Laser_Power("OFF"); // Lāsers nespīd
  Laser_Status = 7; // parking
  Laser_Time1 = millis();

  SendData_PelcoD(Right | Down);

  Serial.println(" Parking start time = "+String(Laser_Time1));
}

void Laser_Parking_BT(void)
{
  unsigned long Laser_BT_Parking_Time1 = 0;
  unsigned long Laser_BT_Parking_Time2 = 0;
  Laser_Power("OFF"); // Lāsers nespīd
  Laser_BT_Parking_Time1 = millis();
  SendData_PelcoD(Left | Down);
  Serial.print("Parking BT.   ");
  Serial.print(" Start time = ");
  Serial.print(Laser_BT_Parking_Time1);
  Vi=0;
  Hi=0;


  Laser_BT_Parking_Time2 = millis();
  while ((Laser_BT_Parking_Time2 - Laser_BT_Parking_Time1) < Laser_Parking_Time)
    {
      Laser_BT_Parking_Time2 = millis();
      WDT_Reset();
    }
  Serial.print("  Stop time = ");
  Serial.println(Laser_BT_Parking_Time2);

  ESP_BT.print("Parking,OK,");
}

void EEPROM_Read(void)
{
```

```
// ************ Atmiņu datu nolasīšana ************************

EEPROM.begin(EEPROM_Size);


Laser_Status=EEPROM.read(0);
if(Laser_Status != 4) Laser_Status=0;

Laser_Delay_Time=EEPROM.read(1);
Laser_Zone=EEPROM.read(2);

Laser_Time_Border1=EEPROM.read(3);
Laser_Time_Border2=EEPROM.read(4);
Laser_Time_Border3=EEPROM.read(5);
Laser_Time_Border4=EEPROM.read(6);

Laser_PWM_Border1=EEPROM.read(7);
Laser_PWM_Border2=EEPROM.read(8);
Laser_PWM_Border3=EEPROM.read(9);
Laser_PWM_Border4=EEPROM.read(10);

Laser_Sector1_H1=EEPROM.read(11);
Laser_Sector1_H2=EEPROM.read(12);
Laser_Sector1_V1=EEPROM.read(13);
Laser_Sector1_V2=EEPROM.read(14);

Laser_Sector2_H1=EEPROM.read(15);
Laser_Sector2_H2=EEPROM.read(16);
Laser_Sector2_V1=EEPROM.read(17);
Laser_Sector2_V2=EEPROM.read(18);

Laser_Sector3_H1=EEPROM.read(19);
Laser_Sector3_H2=EEPROM.read(20);
Laser_Sector3_V1=EEPROM.read(21);
Laser_Sector3_V2=EEPROM.read(22);

Sound_Air_Number=EEPROM.read(30);
Sound_Woter_Number=EEPROM.read(31);

Gaiss_1_Border=EEPROM.read(32);
Gaiss_2_Border=EEPROM.read(33);
Udens_1_Border=EEPROM.read(34);
Udens_2_Border=EEPROM.read(35);

Sound_Start_Time [0]=EEPROM.read(36);
Sound_Start_Time [1]=EEPROM.read(37);
Sound_Start_Time [2]=EEPROM.read(38);
Sound_Start_Time [3]=EEPROM.read(39);
Sound_Start_Time [4]=EEPROM.read(40);
Sound_Status=EEPROM.read(41);


Serial.println("*************** EEPROM ********************");

Serial.println("Laser_Status="+String(Laser_Status));
Serial.println("Laser_Delay_Time="+String(Laser_Delay_Time));
Serial.println("Laser_Zone="+String(Laser_Zone));

Serial.println("Laser_Time_Border1="+String(Laser_Time_Border1));
Serial.println("Laser_Time_Border2="+String(Laser_Time_Border2));
```

```
 Serial.println("Laser_Time_Border3="+String(Laser_Time_Border3));
 Serial.println("Laser_Time_Border4="+String(Laser_Time_Border4));

 Serial.println("Laser_PWM_Border1="+String(Laser_PWM_Border1));
 Serial.println("Laser_PWM_Border2="+String(Laser_PWM_Border2));
 Serial.println("Laser_PWM_Border3="+String(Laser_PWM_Border3));
 Serial.println("Laser_PWM_Border4="+String(Laser_PWM_Border4));

 Serial.println("Laser_Sector1_H1="+String(Laser_Sector1_H1));
 Serial.println("Laser_Sector1_H2="+String(Laser_Sector1_H2));
 Serial.println("Laser_Sector1_V1="+String(Laser_Sector1_V1));
 Serial.println("Laser_Sector1_V2="+String(Laser_Sector1_V2));

 Serial.println("Laser_Sector2_H1="+String(Laser_Sector2_H1));
 Serial.println("Laser_Sector2_H2="+String(Laser_Sector2_H2));
 Serial.println("Laser_Sector2_V1="+String(Laser_Sector2_V1));
 Serial.println("Laser_Sector2_V2="+String(Laser_Sector2_V2));

 Serial.println("Laser_Sector3_H1="+String(Laser_Sector3_H1));
 Serial.println("Laser_Sector3_H2="+String(Laser_Sector3_H2));
 Serial.println("Laser_Sector3_V1="+String(Laser_Sector3_V1));
 Serial.println("Laser_Sector3_V2="+String(Laser_Sector3_V2));

 Serial.println("Sound_Air_Number="+String(Sound_Air_Number));
 Serial.println("Sound_Woter_Number="+String(Sound_Woter_Number));

 Serial.println("Gaiss_1_Border="+String(Gaiss_1_Border));
 Serial.println("Gaiss_2_Border="+String(Gaiss_2_Border));
 Serial.println("Udens_1_Border="+String(Udens_1_Border));
 Serial.println("Udens_2_Border="+String(Udens_2_Border));

 Serial.print("Sound_Start_Time -> "+String(Sound_Start_Time [0])+" "+String(Sound_Start_Time [1]));
 Serial.println(" "+String(Sound_Start_Time [2])+" "+String(Sound_Start_Time [3])+" "+String(Sound_Start_Time [4]));
 Serial.println("Sound_Status="+String(Sound_Status));

 if(Sound_Air_Numbers_Array [0]==0 || Sound_Air_Numbers_Array [0]==255)
  {
   Serial.println();
   Serial.println(" Number  |   Air   |  Woter");
   for (byte k = 0; k < 19; k++)
    {
      Sound_Air_Numbers_Array [k]=EEPROM.read(k+50);
      Sound_Woter_Numbers_Array [k]=EEPROM.read(k+80);
      Serial.println(String(k)+ "       |   "+String(Sound_Air_Numbers_Array[k])+"   |   "+String(Sound_Woter_Numbers_Array[k]));
    }
  }
  else Serial.println(" Sound_Air_Numbers_Array and Sound_Woter_Numbers_Array is EMPTY ");

 EEPROM.end();

 // ************************************************************
}

void LoRaSendMessage(String message)
{
 Serial.println("Send LoRa -> " + message);
 LoRa.beginPacket();                   // start packet
 LoRa.print(message);                  // add payload
 LoRa.endPacket();                     // finish packet and send it
 flag =0;
```

```
  delay(2000);
}

void LoRaReadMessage(int packetSize)
{
  if (packetSize > 0)
  {
    String incoming = "";
   WDT_Reset();
  while (LoRa.available())
    {
      incoming = LoRa.readString();
      flag=1;
    }

  Serial.println();
  Serial.println("*********************************************************************");
  Serial.println("Message: " + incoming);
  Serial.println("RSSI: " + String(LoRa.packetRssi()));
  Serial.println("Snr: " + String(LoRa.packetSnr()));
  Serial.println("*********************************************************************");
  Serial.println();
  LoRa_Message=incoming;
  delay(2000);
  }
}



void setup()
{
  Serial.begin(9600);
  Serial2.begin(2400);
  ESP_BT.begin(BT_Name);

  if (Module_Tipe == "MASTER" || Module_Tipe == "SLAVE_LASER")
   {
     ledcSetup(0, 1000, 8); // PWM: 0-kanāls; 1000-frekvence, Hz; 8-izsķirtspēja (8bit)
     ledcAttachPin(SPI3_CS_PIN, 0); // PWM: SPI3_CS_PIN - izvads; 0-kanāls
   }


  pinMode(SPI3_CS_PIN, OUTPUT);
  pinMode(GSM_CS_PIN, OUTPUT);
  pinMode(LoRa_CS_PIN, OUTPUT);
  pinMode(SD_CS_PIN, OUTPUT);

  pinMode(Bat_EN_PIN, OUTPUT);

  pinMode(DDS_FU_UD_PIN, OUTPUT);
  pinMode(DDS_W_Clk_PIN, OUTPUT);
  pinMode(DDS_Data_PIN, OUTPUT);
  pinMode(DDS_Reset_PIN, OUTPUT);

  pinMode(CLK_EN_PIN, OUTPUT);

  pulseHigh(DDS_Reset_PIN);
  pulseHigh(DDS_W_Clk_PIN);
  pulseHigh(DDS_FU_UD_PIN);
```

```
 WDT_Reset();

if (! rtc.begin())
{
  Serial.println("Couldn't find RTC");
  while (1);
}

SPI_Close_All();

 WDT_Reset();
if (!SD.begin(SD_CS_PIN))
{
  Serial.println("   **********  Card failed, or not present  !!!!!!!!!!!!  ");
  while (1);
}

EEPROM_Read();

SD_Sound_Read(Sound_Air_Number);

Serial.print(" Sound_Point_Number = ");
Serial.println(Sound_Point_Number);

Serial.println(" Nr  |  ms  |  Freq  |   Amp  |");

Serial.print(Sound_Package [0][0]);
Serial.print("  ");
Serial.print(Sound_Package [0][1]);
Serial.print("  ");
Serial.println(Sound_Package [0][2]);

for(int i=1; i<=Sound_Point_Number-1000; i=i+1000)
  {
    Serial.print(i);
    Serial.print(" ---> ");
    Serial.print(Sound_Package [i][0]);
    Serial.print(" ms,  ");
    Serial.print(Sound_Package [i][1]);
    Serial.print(" => ");
    Serial.print(Sound_Package [i][1]*20);
    Serial.print(" Hz, ");
    Serial.print(Sound_Package [i][2]);
    Serial.print(" => ");
    Serial.println(map(Sound_Package [i][2], 1, 252, Ampl_Low, Ampl_High));

  }
Serial.print(Sound_Point_Number+1);
    Serial.print(" ---> ");
    Serial.print(Sound_Package [Sound_Point_Number+1][0]);
    Serial.print("  ");
    Serial.print(Sound_Package [Sound_Point_Number+1][1]);
    Serial.print("  ");
    Serial.println(Sound_Package [Sound_Point_Number+1][2]);

 Serial.printf("Total space: %lluMB\n", SD.totalBytes() / (1024 * 1024));
 Serial.printf("Used space: %llukB\n", SD.usedBytes() / 1024);

WDT_Reset();
Serial.println();
```

```
 Serial.println();
 Serial.println(" ************  START  ******************");

 LoRa.setPins(LoRa_CS_PIN, LoRa_RESET_PIN, DIO0_PIN);
  WDT_Reset();
 while (!LoRa.begin(433E6))
   {
     Serial.println("LoRa init failed. Check your connections.");
     delay(500);
   }
 LoRa.setSpreadingFactor(12);
 LoRa.setSyncWord(LoRaSyncWord);
 Serial.println();
 Serial.println("LoRa Initializing OK!");


  WDT_Reset();

  Serial.println();
  Serial.println(" ******************** Programmas darbības SAKUMS ******************");
  Serial.println();
  DateTime now = rtc.now();
  Serial.println("LogFileName = " + LogFileName);

  File myFile = SD.open(LogFileName, FILE_APPEND);
  if (myFile)
       {
         byte Log_Year = now.year();
         byte Log_Month = now.month();
         byte Log_Day = now.day();
         byte Log_Hour = now.hour();
         byte Log_minute = now.minute();
         myFile.println(String(Log_Day)+"."+String(Log_Month)+"."+String(Log_Year)+"  "+String(Log_Hour)+":"+String(Log_minute)+" START ");

       }
  myFile.close();

}

void loop()
{
  // ************************************** LoRa *********************************************
  LoRa_Message="";

  LoRaReadMessage(LoRa.parsePacket());


  if(LoRa_Message!="")
    {
        WDT_Reset();
        LoRa_Parsing(LoRa_Message);
        if(LoRa_Commands[0]=="254")
            {
              Serial.println(" ******* LoRa  -- Komanda (254) ************");
              if(LoRa_Commands [2]==String(Module_Number)) // Komanda tieši šim modulim
                {
                  if(LoRa_Commands [3]=="101") // pārtraukt skaļruņa un lāzera darbību (no LoRa)
                   {
                     WDT_Reset();
                    Laser_Status=4;
```

```
                Sound_Status=4;
                EEPROM.begin(EEPROM_Size);
                EEPROM.write(0,Laser_Status);
                EEPROM.write(41,Sound_Status);
                EEPROM.end();
                Serial.println("==== pārtraukt skaļruņa un lāzera darbību (no LoRa) ====");
                LoRaSendMessage("pārtraukt skaļruņa un lāzera darbību");
              }
             if(LoRa_Commands [3]=="102") // sākt skaļruņa darbību pēc agrāk aktivizētas programmas (no LoRa)
              {
                 WDT_Reset();
                Sound_Status=1;
                EEPROM.begin(EEPROM_Size);
                EEPROM.write(41,Sound_Status);
                EEPROM.end();
                Serial.println("==== sākt skaļruņa darbību pēc agrāk aktivizētas programmas (no LoRa) ====");
                LoRaSendMessage("sākt skaļruņa darbību pēc agrāk aktivizētas programmas");
              }
             if(LoRa_Commands [3]=="103") // sākt lāzera darbību pēc agrāk aktivizētas programmas (no LoRa)
              {
                 WDT_Reset();
                Laser_Status=0;
                EEPROM.begin(EEPROM_Size);
                EEPROM.write(0,Laser_Status);
                EEPROM.end();
                Serial.println("==== sākt lāzera darbību pēc agrāk aktivizētas programmas (no LoRa) ====");
                LoRaSendMessage("sākt lāzera darbību pēc agrāk aktivizētas programmas");
              }
             if(LoRa_Commands [3]=="104") // nolasīt ierīces VISUS darba datus
              {
                    WDT_Reset();

                    Serial.println("Dati -> LoRa");
                    String LoRa_Atbilde;

                    DateTime now = rtc.now();
                    byte Laiks_for_Android = now.hour();

                    LoRa_Atbilde = "254,x,1,203,104,"+String(Module_Number)+",";

                    LoRa_Atbilde = LoRa_Atbilde + String(Volt_12V)+","+String(Volt_4V)+","+String(Temperatura)+","+String(Laiks_for_Android) + ",";

                    Laiks_for_Android = now.minute();
                    LoRa_Atbilde = LoRa_Atbilde + String(Laiks_for_Android)+","+String(Laser_Delay_Time)+","+String(Laser_Zone)+",";
                    LoRa_Atbilde = LoRa_Atbilde + String(Laser_Time_Border1)+","+String(Laser_PWM_Border1)+",";

                    LoRa_Atbilde = LoRa_Atbilde + String(Laser_Time_Border2)+","+String(Laser_PWM_Border2)+","+String(Laser_Time_Border3)+","+String(Laser_PWM_Border3)+",";
                    LoRa_Atbilde = LoRa_Atbilde + String(Laser_Time_Border4)+","+String(Laser_PWM_Border4)+","+String(Laser_Status)+",";
                    LoRa_Atbilde = LoRa_Atbilde + String(Sound_Air_Number)+","+String(Sound_Woter_Number)+",";
                    LoRa_Atbilde = LoRa_Atbilde + String(Gaiss_1_Border)+","+String(Gaiss_2_Border)+",";
                    LoRa_Atbilde = LoRa_Atbilde + String(Udens_1_Border)+","+String(Udens_2_Border)+",";
                    LoRa_Atbilde = LoRa_Atbilde + String(Sound_Start_Time [0])+","+String(Sound_Start_Time [1])+",";
                    LoRa_Atbilde = LoRa_Atbilde + String(Sound_Start_Time [2])+","+String(Sound_Start_Time [3])+",";
                    LoRa_Atbilde = LoRa_Atbilde + String(Sound_Start_Time [4])+","+String(Sound_Status)+",";

                    LoRaSendMessage(LoRa_Atbilde);

                    WDT_Reset();
```

```
 }
if(LoRa_Commands [3]=="105") // nolasīt ierīces SPRIEGUMU un temperatūras datus
 {
      WDT_Reset();

      Serial.println("Dati -> LoRa");
      String LoRa_Atbilde;


      LoRa_Atbilde = "254,x,1,203,105,"+String(Module_Number)+",";

      LoRa_Atbilde = LoRa_Atbilde + String(Volt_12V)+","+String(Volt_4V)+","+String(Temperatura)+",";

      LoRaSendMessage(LoRa_Atbilde);

      WDT_Reset();

 }

if(LoRa_Commands [3]=="106") // nolasīt ierīces SKAŅAS darba datus
 {
      WDT_Reset();

      Serial.println("Dati -> LoRa");
      String LoRa_Atbilde;

      LoRa_Atbilde = "254,x,1,203,106,"+String(Module_Number)+",";

      LoRa_Atbilde = LoRa_Atbilde + String(Gaiss_1_Border)+","+String(Gaiss_2_Border)+",";
      LoRa_Atbilde = LoRa_Atbilde + String(Udens_1_Border)+","+String(Udens_2_Border)+",";
      LoRa_Atbilde = LoRa_Atbilde + String(Sound_Start_Time [0])+","+String(Sound_Start_Time [1])+",";
      LoRa_Atbilde = LoRa_Atbilde + String(Sound_Start_Time [2])+","+String(Sound_Start_Time [3])+",";
      LoRa_Atbilde = LoRa_Atbilde + String(Sound_Start_Time [4])+","+String(Sound_Status)+",";

      LoRaSendMessage(LoRa_Atbilde);

      WDT_Reset();

 }

if(LoRa_Commands [3]=="107") // nolasīt ierīces LĀZERA darba datus
 {
      WDT_Reset();

      Serial.println("Dati -> LoRa");
      String LoRa_Atbilde;

      LoRa_Atbilde = "254,x,1,203,107,"+String(Module_Number)+",";

      LoRa_Atbilde = LoRa_Atbilde + ","+String(Laser_Delay_Time)+","+String(Laser_Zone)+",";
      LoRa_Atbilde = LoRa_Atbilde + String(Laser_Time_Border1)+","+String(Laser_PWM_Border1)+",";

      LoRa_Atbilde = LoRa_Atbilde + String(Laser_Time_Border2)+","+String(Laser_PWM_Border2)+","+String(Laser_Time_Border3)+","+String(Laser_PWM_Border3)+",";
      LoRa_Atbilde = LoRa_Atbilde + String(Laser_Time_Border4)+","+String(Laser_PWM_Border4)+","+String(Laser_Status)+",";

      LoRaSendMessage(LoRa_Atbilde);

      WDT_Reset();
```

```
 }

if(LoRa_Commands [3]=="108") // nolasīt skaņas failu secību gaisam
 {
      WDT_Reset();

      Serial.println("Dati -> LoRa");
      String LoRa_Atbilde;

      LoRa_Atbilde = "254,x,1,203,108,"+String(Module_Number)+",";

      LoRaSendMessage(LoRa_Atbilde);

      WDT_Reset();

 }

if(LoRa_Commands [3]=="109") // nolasīt skaņas failu secību ūdenim
 {
      WDT_Reset();

      Serial.println("Dati -> LoRa");
      String LoRa_Atbilde;

      LoRa_Atbilde = "254,x,1,203,109,"+String(Module_Number)+",";


      LoRaSendMessage(LoRa_Atbilde);

      WDT_Reset();

 }
if(LoRa_Commands [3]=="201") // aktivizēt skaņas programmu (no LoRa)
 {
   WDT_Reset();

   Sound_Air_Number = LoRa_Commands [4].toInt();
   Sound_Woter_Number = LoRa_Commands [5].toInt();
   Gaiss_1_Border = LoRa_Commands [6].toInt();
   Gaiss_2_Border = LoRa_Commands [7].toInt();
   Udens_1_Border = LoRa_Commands [8].toInt();
   Udens_2_Border = LoRa_Commands [9].toInt();
   Sound_Start_Time [0] = LoRa_Commands [10].toInt();
   Sound_Start_Time [1] = LoRa_Commands [11].toInt();
   Sound_Start_Time [2] = LoRa_Commands [12].toInt();
   Sound_Start_Time [3] = LoRa_Commands [13].toInt();
   Sound_Start_Time [4] = LoRa_Commands [14].toInt();

   Serial.println("==== aktivizēt skaņas programmu (no LoRa) ====");
   Serial.println("Sound_Air_Number = "+String(Sound_Air_Number));
   Serial.println("Sound_Woter_Number = "+String(Sound_Woter_Number));

   Serial.println("Gaiss_1_Border = "+String(Gaiss_1_Border));
   Serial.println("Gaiss_2_Border = "+String(Gaiss_2_Border));
   Serial.println("Udens_1_Border = "+String(Udens_1_Border));
   Serial.println("Udens_2_Border = "+String(Udens_2_Border));

   Serial.print("Sound_Start_Time -> "+String(Sound_Start_Time [0])+" "+String(Sound_Start_Time [1]));
   Serial.println(" "+String(Sound_Start_Time [2])+" "+String(Sound_Start_Time [3])+" "+String(Sound_Start_Time [4]));
   Serial.println();
```

```
    EEPROM.begin(EEPROM_Size);
    EEPROM.write(30,Sound_Air_Number);
    EEPROM.write(31,Sound_Woter_Number);

    EEPROM.write(32,Gaiss_1_Border);
    EEPROM.write(33,Gaiss_2_Border);
    EEPROM.write(34,Udens_1_Border);
    EEPROM.write(35,Udens_2_Border);

    EEPROM.write(36,Sound_Start_Time [0]);
    EEPROM.write(37,Sound_Start_Time [1]);
    EEPROM.write(38,Sound_Start_Time [2]);
    EEPROM.write(39,Sound_Start_Time [3]);
    EEPROM.write(40,Sound_Start_Time [4]);
    EEPROM.end();
    LoRaSendMessage("aktivizēt skaņas programmu");
  }
 if(LoRa_Commands [3]=="202") // aktivizēt lāzera programmu (no LoRa)
  {
      WDT_Reset();

      Laser_Delay_Time = LoRa_Commands [4].toInt();
      Laser_Zone = LoRa_Commands [5].toInt();
      Laser_Time_Border1 = LoRa_Commands [6].toInt();
      Laser_PWM_Border1 = LoRa_Commands [7].toInt();
      Laser_Time_Border2 = LoRa_Commands [8].toInt();
      Laser_PWM_Border2 = LoRa_Commands [9].toInt();
      Laser_Time_Border3 = LoRa_Commands [10].toInt();
      Laser_PWM_Border3 = LoRa_Commands [11].toInt();
      Laser_Time_Border4 = LoRa_Commands [12].toInt();
      Laser_PWM_Border4 = LoRa_Commands [13].toInt();
      Serial.println("==== aktivizēt lāzera programmu (no LoRa) ====");
      Serial.println("Laser_Delay_Time = "+String(Laser_Delay_Time));
      Serial.println("Laser_Zone = "+String(Laser_Zone));
      Serial.println("Laser_Time_Border1 = "+String(Laser_Time_Border1));
      Serial.println("Laser_PWM_Border1 = "+String(Laser_PWM_Border1));
      Serial.println("Laser_Time_Border2 = "+String(Laser_Time_Border2));
      Serial.println("Laser_PWM_Border2 = "+String(Laser_PWM_Border2));
      Serial.println("Laser_Time_Border3 = "+String(Laser_Time_Border3));
      Serial.println("Laser_PWM_Border3 = "+String(Laser_PWM_Border3));
      Serial.println("Laser_Time_Border4 = "+String(Laser_Time_Border4));
      Serial.println("Laser_PWM_Border4 = "+String(Laser_PWM_Border4));
      Serial.println();
      EEPROM.begin(EEPROM_Size);
      EEPROM.write(1,Laser_Delay_Time);
      EEPROM.write(2,Laser_Zone);
      EEPROM.write(3,Laser_Time_Border1);
      EEPROM.write(4,Laser_Time_Border2);
      EEPROM.write(5,Laser_Time_Border3);
      EEPROM.write(6,Laser_Time_Border4);
      EEPROM.write(7,Laser_PWM_Border1);
      EEPROM.write(8,Laser_PWM_Border2);
      EEPROM.write(9,Laser_PWM_Border3);
      EEPROM.write(10,Laser_PWM_Border4);
      EEPROM.end();
      LoRaSendMessage("aktivizēt lāzera programmu");
  }
if(LoRa_Commands [3]=="203") // Pārsūtīt datu serverim (no LoRa)
 {
   LoRaSendMessage(LoRa_Message);
```

```
            }
          }


        }
  }

// ************************************* LoRa (beigas) *********************************

// ************************************* Skaņa *****************************************
if(Sound_Status == 1) // 1 - IR atļauts strādāt
  {
    if(Sound_Flag == 0) // Skaņas laika tabula NAV uzģenerēta
      {
        DateTime now = rtc.now();
        Sound_Time_Second = now.second();
        if(Sound_Time_Second>0 && Sound_Time_Second<10)
         {
           Serial.println("******* Skaņas grafika ģenerēšana ******");
           for (int i = 0; i < 4; i++)
            {
              Sound_Start_Random [i] = 0;
            }

           for (int i = 0; i < 4; i++)
            {
              if(Sound_Start_Time[i]<Sound_Start_Time[i+1])
               {
                 Serial.println (String(Sound_Start_Time[i])+" - "+String(Sound_Start_Time[i+1]));
                 Sound_Start_Random [i] = random(Sound_Start_Time[i], Sound_Start_Time[i+1]);
                 Sound_Start_Random_Status[i] = 0;
                 Serial.println(Sound_Start_Random [i]);
               }
            }
           Sound_Flag = 1;
         }
      }
      else // Skaņas laika tabula JAU uzģenerēta -> var atskaņot
        {
          DateTime now = rtc.now();
          Sound_Time_Second = now.second();
          WDT_Reset();
          Serial.println();
          Serial.println(" Tekošais laiks, sek = "+String(Sound_Time_Second));
          delay (1000);

          //   tekošais laiks >= pirmais laiks garfikā       pirmais laiks garfikā vēl NAV nostrādāts     laiks NAV 0
          if(Sound_Time_Second >= Sound_Start_Random [0] && Sound_Start_Random_Status[0] == 0 && Sound_Start_Random [0] != 0)
            {
              Sound_Play();
              Sound_Start_Random_Status[0] = 1;
              Serial.println(" Sound Play 0 ");
              if(Sound_Start_Random [1] == 0) Sound_Flag = 0;
            }
          if(Sound_Time_Second >= Sound_Start_Random [1] && Sound_Start_Random_Status[1] == 0 && Sound_Start_Random [1] != 0)
            {
              Sound_Play();
              Sound_Start_Random_Status[1] = 1;
              Serial.println(" Sound Play 1 ");
              if(Sound_Start_Random [2] == 0) Sound_Flag = 0;
```

```
          }
        if(Sound_Time_Second >= Sound_Start_Random [2] && Sound_Start_Random_Status[2] == 0 && Sound_Start_Random [2] != 0)
          {
            Sound_Play();
            Sound_Start_Random_Status[2] = 1;
            Serial.println(" Sound Play 2 ");
            if(Sound_Start_Random [3] == 0) Sound_Flag = 0;
          }
        if(Sound_Time_Second >= Sound_Start_Random [3] && Sound_Start_Random_Status[3] == 0 && Sound_Start_Random [3] != 0)
          {
            Sound_Play();
            Sound_Start_Random_Status[3] = 1;
            Serial.println(" Sound Play 3 ");
            Sound_Flag = 0;
          }

       }

  }
// ************************************ SKAŅA (beigas) ***************************************


// ****************************** Temperatura ***************************************
Temperatura=analogRead(Temper_PIN)*0.0009;
Temperatura= -1481.96+sqrt(2196200+((1.8639-Temperatura)/0.00000388));

if(Temperatura > 40 && Battery_Status==1)
 {
  digitalWrite(Bat_EN_PIN, LOW);
  Battery_Status=0;
 }
if(Temperatura <= 35 && Battery_Status==0)
 {
  digitalWrite(Bat_EN_PIN, HIGH);
  Battery_Status=1;
 }
// ****************************** Temperatura (beigas) ***************************************

// ********************************** Baterijas spriegums ************************************************

Volt_12V=analogRead(Volt_12V_PIN)*0.00091*5.174;
Volt_4V=analogRead(Volt_4V_PIN)*0.00086*1.52;

SD_Log();

// ********************************** Baterijas spriegums (beigas) ***************************************

// ********************************** Bluettoth (sakums) ************************************************
if(ESP_BT.available())
 {
   Text_Read_BT = ESP_BT.readString();
   Serial.println();
   Serial.println(" ********** Strādā Bluettoth modulis **********");
   Serial.println(Text_Read_BT);
   while (Text_Read_BT!="StopLaserSetup")
   {
      WDT_Reset();
     if(ESP_BT.available())
      {
```

```
Text_Read_BT = ESP_BT.readString();
Serial.println(Text_Read_BT);

if(Text_Read_BT=="Parking,") Laser_Parking_BT();

if(Text_Read_BT=="Sector1,H1,")
 {
   ParameterBT = "Sector1,H1,";
   Serial.println("ParameterBT = Sector1,H1,");


   Laser_Sector1_H1 = Hi;
            Serial.println("Laser_Sector1_H1 = "+String(Laser_Sector1_H1));
            ESP_BT.print("Sector1,H1,"+String(Laser_Sector1_H1));
            EEPROM.begin(EEPROM_Size);
            EEPROM.write(11,Laser_Sector1_H1);
            EEPROM.end();
 }
if(Text_Read_BT=="Sector1,H2,")
 {
   ParameterBT = "Sector1,H2,";
   Serial.println("ParameterBT = Sector1,H2,");


   Laser_Sector1_H2 = Hi;
            Serial.println("Laser_Sector1_H2 = "+String(Laser_Sector1_H2));
            ESP_BT.print("Sector1,H2,"+String(Laser_Sector1_H2));
            EEPROM.begin(EEPROM_Size);
            EEPROM.write(12,Laser_Sector1_H2);
            EEPROM.end();
 }
if(Text_Read_BT=="Sector1,V1,")
 {
   ParameterBT = "Sector1,V1,";
   Serial.println("ParameterBT = Sector1,V1,");
   //ESP_BT.print(ParameterBT);

   Laser_Sector1_V1 = Vi;
            Serial.println("Laser_Sector1_V1 = "+String(Laser_Sector1_V1));
            ESP_BT.print("Sector1,V1,"+String(Laser_Sector1_V1));
            EEPROM.begin(EEPROM_Size);
            EEPROM.write(13,Laser_Sector1_V1);
            EEPROM.end();
 }
if(Text_Read_BT=="Sector1,V2,")
 {
   ParameterBT = "Sector1,V2,";
   Serial.println("ParameterBT = Sector1,V2,");
   //ESP_BT.print(ParameterBT);

   Laser_Sector1_V2 = Vi;
            Serial.println("Laser_Sector1_V2 = "+String(Laser_Sector1_V2));
            ESP_BT.print("Sector1,V2,"+String(Laser_Sector1_V2));
            EEPROM.begin(EEPROM_Size);
            EEPROM.write(14,Laser_Sector1_V2);
            EEPROM.end();
 }
```

```
if(Text_Read_BT=="Sector2,H1,")
 {
   ParameterBT = "Sector2,H1,";
   Serial.println("ParameterBT = Sector2,H1,");
   //ESP_BT.print(ParameterBT);

   Laser_Sector2_H1 = Hi;
            Serial.println("Laser_Sector2_H1 = "+String(Laser_Sector2_H1));
            ESP_BT.print("Sector2,H1,"+String(Laser_Sector2_H1));
            EEPROM.begin(EEPROM_Size);
            EEPROM.write(15,Laser_Sector2_H1);
            EEPROM.end();
 }
if(Text_Read_BT=="Sector2,H2,")
 {
   ParameterBT = "Sector2,H2,";
   Serial.println("ParameterBT = Sector2,H2,");
   //ESP_BT.print(ParameterBT);

   Laser_Sector2_H2 = Hi;
            Serial.println("Laser_Sector2_H2 = "+String(Laser_Sector2_H2));
            ESP_BT.print("Sector2,H2,"+String(Laser_Sector2_H2));
            EEPROM.begin(EEPROM_Size);
            EEPROM.write(16,Laser_Sector2_H2);
            EEPROM.end();
 }
if(Text_Read_BT=="Sector2,V1,")
 {
   ParameterBT = "Sector2,V1,";
   Serial.println("ParameterBT = Sector2,V1,");
   //ESP_BT.print(ParameterBT);

   Laser_Sector2_V1 = Vi;
            Serial.println("Laser_Sector2_V1 = "+String(Laser_Sector2_V1));
            ESP_BT.print("Sector2,V1,"+String(Laser_Sector2_V1));
            EEPROM.begin(EEPROM_Size);
            EEPROM.write(17,Laser_Sector2_V1);
            EEPROM.end();
 }
if(Text_Read_BT=="Sector2,V2,")
 {
   ParameterBT = "Sector2,V2,";
   Serial.println("ParameterBT = Sector2,V2,");
   //ESP_BT.print(ParameterBT);

   Laser_Sector2_V2 = Vi;
            Serial.println("Laser_Sector2_V2 = "+String(Laser_Sector2_V2));
            ESP_BT.print("Sector2,V2,"+String(Laser_Sector2_V2));
            EEPROM.begin(EEPROM_Size);
            EEPROM.write(18,Laser_Sector2_V2);
            EEPROM.end();
 }



if(Text_Read_BT=="Sector3,H1,")
 {
   ParameterBT = "Sector3,H1,";
   Serial.println("ParameterBT = Sector3,H1,");
```

```
    //ESP_BT.print(ParameterBT);

    Laser_Sector3_H1 = Hi;
            Serial.println("Laser_Sector3_H1 = "+String(Laser_Sector3_H1));
            ESP_BT.print("Sector3,H1,"+String(Laser_Sector3_H1));
            EEPROM.begin(EEPROM_Size);
            EEPROM.write(19,Laser_Sector3_H1);
            EEPROM.end();
  }
if(Text_Read_BT=="Sector3,H2,")
  {
    ParameterBT = "Sector3,H2,";
    Serial.println("ParameterBT = Sector3,H2,");
    //ESP_BT.print(ParameterBT);

    Laser_Sector3_H2 = Hi;
            Serial.println("Laser_Sector3_H2 = "+String(Laser_Sector3_H2));
            ESP_BT.print("Sector3,H2,"+String(Laser_Sector3_H2));
            EEPROM.begin(EEPROM_Size);
            EEPROM.write(20,Laser_Sector3_H2);
            EEPROM.end();
  }
if(Text_Read_BT=="Sector3,V1,")
  {
    ParameterBT = "Sector3,V1,";
    Serial.println("ParameterBT = Sector3,V1,");
    //ESP_BT.print(ParameterBT);

    Laser_Sector3_V1 = Vi;
            Serial.println("Laser_Sector3_V1 = "+String(Laser_Sector3_V1));
            ESP_BT.print("Sector3,V1,"+String(Laser_Sector3_V1));
            EEPROM.begin(EEPROM_Size);
            EEPROM.write(21,Laser_Sector3_V1);
            EEPROM.end();
  }
if(Text_Read_BT=="Sector3,V2,")
  {
    ParameterBT = "Sector3,V2,";
    Serial.println("ParameterBT = Sector3,V2,");
    //ESP_BT.print(ParameterBT);

    Laser_Sector3_V2 = Vi;
            Serial.println("Laser_Sector3_V2 = "+String(Laser_Sector3_V2));
            ESP_BT.print("Sector3,V2,"+String(Laser_Sector3_V2));
            EEPROM.begin(EEPROM_Size);
            EEPROM.write(22,Laser_Sector3_V2);
            EEPROM.end();
  }



if(Text_Read_BT=="PWM_ON,")
  {
    Laser_Power("FULL"); // Laser ON
    Serial.println("Laser ON");
  }
if(Text_Read_BT=="PWM_OFF,")
  {
    Laser_Power("OFF");
    Serial.println("Laser OFF");
```

```
 }

if(Text_Read_BT=="Up,")
  {
    Laser_BT_Time1=millis();
    SendData_PelcoD(Up);
    Serial.println("Up  "+String(Laser_BT_Time1));
    while (Text_Read_BT != "Stop,")
     {
       Text_Read_BT = ESP_BT.readString();
        WDT_Reset();
     }
    Laser_BT_Time2=millis();
    SendData_PelcoD(Stop);
    Serial.println("Stop "+String(Laser_BT_Time2));
    Serial.println("Pārvietošanas laiks="+String(Laser_BT_Time2-Laser_BT_Time1));
    Serial.print("Vertikālais leņķis: vecais-"+String(Vi));
    Vi = Vi + (Laser_BT_Time2-Laser_BT_Time1)/1000*Laser_Speed_V;
    Serial.println("  jaunais-"+String(Vi));

  }

if(Text_Read_BT=="Down,")
  {
    Laser_BT_Time1=millis();
    SendData_PelcoD(Down);
    Serial.println("Down  "+String(Laser_BT_Time1));
    while (Text_Read_BT != "Stop,")
     {
       Text_Read_BT = ESP_BT.readString();
        WDT_Reset();
     }
    Laser_BT_Time2=millis();
    SendData_PelcoD(Stop);
    Serial.println("Stop "+String(Laser_BT_Time2));
    Serial.println("Pārvietošanas laiks="+String(Laser_BT_Time2-Laser_BT_Time1));
    Serial.print("Vertikālais leņķis: vecais-"+String(Vi));
    Vi = Vi - (Laser_BT_Time2-Laser_BT_Time1)/1000*Laser_Speed_V;
    Serial.println("  jaunais-"+String(Vi));
  }


if(Text_Read_BT=="Right,")
  {
    Laser_BT_Time1=millis();
    SendData_PelcoD(Right);
    Serial.println("Right  "+String(Laser_BT_Time1));
    while (Text_Read_BT != "Stop,")
     {
       Text_Read_BT = ESP_BT.readString();
        WDT_Reset();
     }
    Laser_BT_Time2=millis();
    SendData_PelcoD(Stop);
    Serial.println("Stop "+String(Laser_BT_Time2));
    Serial.println("Pārvietošanas laiks="+String(Laser_BT_Time2-Laser_BT_Time1));
    Serial.print("Horizontālais leņķis: vecais-"+String(Hi));
    Hi = Hi + (Laser_BT_Time2-Laser_BT_Time1)/1000*Laser_Speed_H;
    Serial.println("  jaunais-"+String(Hi));
  }
```

```
if(Text_Read_BT=="Left,")
  {
    Laser_BT_Time1=millis();
    SendData_PelcoD(Left);
    Serial.println("Left  "+String(Laser_BT_Time1));
    while (Text_Read_BT != "Stop,")
     {
       Text_Read_BT = ESP_BT.readString();
        WDT_Reset();
     }
    Laser_BT_Time2=millis();
    SendData_PelcoD(Stop);
    Serial.println("Stop "+String(Laser_BT_Time2));
    Serial.println("Pārvietošanas laiks="+String(Laser_BT_Time2-Laser_BT_Time1));
    Serial.print("Horizontālais leņķis: vecais-"+String(Hi));
    Hi = Hi - (Laser_BT_Time2-Laser_BT_Time1)/1000*Laser_Speed_H;
    Serial.println("  jaunais-"+String(Hi));
  }


BT_Parsing(Text_Read_BT);
if(BT_Commands [0]=="254")
 {
   Serial.println(" ******* Komanda (254) ***********");
   ESP_BT.print(Text_Read_BT);

   if(BT_Commands [2]==String(Module_Number)) // Komanda tieši šīm modulim
     {
       if(BT_Commands [3]=="101") // pārtraukt skaļruņa un lāzera darbību
        {
          Laser_Status=4;
          Sound_Status=4;
          EEPROM.begin(EEPROM_Size);
          EEPROM.write(0,Laser_Status);
          EEPROM.write(41,Sound_Status);
          EEPROM.end();
          Serial.println("==== pārtraukt skaļruņa un lāzera darbību ====");
        }
       if(BT_Commands [3]=="102") // sākt skaļruņa darbību pēc agrāk aktivizētas programmas
        {
          Sound_Status=1;
          EEPROM.begin(EEPROM_Size);
          EEPROM.write(41,Sound_Status);
          EEPROM.end();
          Serial.println("==== sākt skaļruņa darbību pēc agrāk aktivizētas programmas ====");
        }
       if(BT_Commands [3]=="103") // sākt lāzera darbību pēc agrāk aktivizētas programmas
        {
          Laser_Status=0;
          EEPROM.begin(EEPROM_Size);
          EEPROM.write(0,Laser_Status);
          EEPROM.end();
          Serial.println("==== sākt lāzera darbību pēc agrāk aktivizētas programmas ====");
        }
       if(BT_Commands [3]=="104") // nolasīt ierīces VISUS darba datus
        {

            Serial.println("Dati -> BT");
            ESP_BT.print("\nGalv. akum. = "+String(Volt_12V)+"V \nPlates akum. = "+String(Volt_4V)+"V");
```

```
                    ESP_BT.print("\nGalvēnas plates temperatūra = "+String(Temperatura)+" *C");
                    DateTime now = rtc.now();
                    byte Laiks_for_Android = now.hour();
                    ESP_BT.print("\nTekošais laiks = "+String(Laiks_for_Android));
                    Laiks_for_Android = now.minute();
                    ESP_BT.print(":"+String(Laiks_for_Android));
                    ESP_BT.print("\n");

                    ESP_BT.print("\nLazera laika intervāls = "+String(Laser_Delay_Time)+" min. \nLazera zonas skaits vienā sektorā =
"+String(Laser_Zone));
                    ESP_BT.print("\nDarba laiks 1 = "+String(Laser_Time_Border1)+" st. -> spilgtums1 = "+String(Laser_PWM_Border1));
                    ESP_BT.print("\nDarba laiks 2 = "+String(Laser_Time_Border2)+" st. -> spilgtums2 = "+String(Laser_PWM_Border2));
                    ESP_BT.print("\nDarba laiks 3 = "+String(Laser_Time_Border3)+" st. -> spilgtums3 = "+String(Laser_PWM_Border3));
                    ESP_BT.print("\nDarba laiks 4 = "+String(Laser_Time_Border4)+" st. -> spilgtums4 = "+String(Laser_PWM_Border4));
                    ESP_BT.print("\nLasera status = "+String(Laser_Status));
                    ESP_BT.print("\n(0 - nestrādā(gaidīšanas režīms); 1- kustības kartes ģenerēšana; 2 - parking; 3 - PROGRAMMAS IZPILDE; 4 - aizlēgts
strādāt)");
                    ESP_BT.print("\nSector1_H1 = "+String(Laser_Sector1_H1)+"  Sector1_H2 = "+String(Laser_Sector1_H2));
                    ESP_BT.print("\nSector2_H1 = "+String(Laser_Sector2_H1)+"  Sector2_H2 = "+String(Laser_Sector2_H2));
                    ESP_BT.print("\nSector3_H1 = "+String(Laser_Sector3_H1)+"  Sector3_H2 = "+String(Laser_Sector3_H2));
                    ESP_BT.print("\nSector1_V1 = "+String(Laser_Sector1_V1)+"  Sector1_V2 = "+String(Laser_Sector1_V2));
                    ESP_BT.print("\nSector2_V1 = "+String(Laser_Sector2_V1)+"  Sector2_V2 = "+String(Laser_Sector2_V2));
                    ESP_BT.print("\nSector3_V1 = "+String(Laser_Sector3_V1)+"  Sector3_V2 = "+String(Laser_Sector3_V2));
                    ESP_BT.print("\n");

                    ESP_BT.print("\nSkaņas masīva numurs gaisā = "+String(Sound_Air_Number)+"\nSkaņas masīva numurs ūdenī =
"+String(Sound_Woter_Number));
                    ESP_BT.print("\nLaika 1. robeža gaisā  = "+String(Gaiss_1_Border)+" st. \nLaika 2. robeža gaisā = "+String(Gaiss_2_Border)+" st.");
                    ESP_BT.print("\nLaika 1. robeža ūdenī  = "+String(Udens_1_Border)+" st. \nLaika 2. robeža ūdenī = "+String(Udens_2_Border)+"
st.");
                    ESP_BT.print("\nIeslēgšanas laiks 0 = "+String(Sound_Start_Time [0])+" sek. \nIeslēgšanas laiks 1 = "+String(Sound_Start_Time
[1])+" sek.");
                    ESP_BT.print("\nIeslēgšanas laiks 2 = "+String(Sound_Start_Time [2])+" sek. \nIeslēgšanas laiks 3 = "+String(Sound_Start_Time
[3])+" sek.");
                    ESP_BT.print("\nIeslēgšanas laiks 4 = "+String(Sound_Start_Time [4])+" sek. \nSkaņas status = "+String(Sound_Status));
                    ESP_BT.print("\n(4 - NAV atļauts strādāt; 1 - IR atļauts strādāt)");
                }
                if(BT_Commands [3]=="105") // nolasīt ierīces SPRIEGUMU un temperatūras datus
                {
                    Serial.println("Dati -> BT");
                    ESP_BT.print("\nGalv. akum. = "+String(Volt_12V)+"V \nPlates akum. = "+String(Volt_4V)+"V");
                    ESP_BT.print("\nGalvēnas plates temperatūra = "+String(Temperatura)+" *C");
                    DateTime now = rtc.now();
                    byte Laiks_for_Android = now.hour();
                    ESP_BT.print("\nTekošais laiks = "+String(Laiks_for_Android));
                    Laiks_for_Android = now.minute();
                    ESP_BT.print(":"+String(Laiks_for_Android));
                    ESP_BT.print("\n");
                }
                if(BT_Commands [3]=="106") //  nolasīt ierīces SKAŅAS darba datus
                {

                    Serial.println("Dati -> BT");
                    ESP_BT.print("\nSkaņas masīva numurs gaisā = "+String(Sound_Air_Number)+"\nSkaņas masīva numurs ūdenī =
"+String(Sound_Woter_Number));
                    ESP_BT.print("\nLaika 1. robeža gaisā  = "+String(Gaiss_1_Border)+" st. \nLaika 2. robeža gaisā = "+String(Gaiss_2_Border)+" st.");
                    ESP_BT.print("\nLaika 1. robeža ūdenī  = "+String(Udens_1_Border)+" st. \nLaika 2. robeža ūdenī = "+String(Udens_2_Border)+"
st.");
                    ESP_BT.print("\nIeslēgšanas laiks 0 = "+String(Sound_Start_Time [0])+" sek. \nIeslēgšanas laiks 1 = "+String(Sound_Start_Time
[1])+" sek.");
```

```
                ESP_BT.print("\nIeslēgšanas laiks 2 = "+String(Sound_Start_Time [2])+" sek. \nIeslēgšanas laiks 3 = "+String(Sound_Start_Time [3])+" sek.");
                ESP_BT.print("\nIeslēgšanas laiks 4 = "+String(Sound_Start_Time [4])+" sek. \nSkaņas status = "+String(Sound_Status));
                ESP_BT.print("\n(4 - NAV atļauts strādāt; 1 - IR atļauts strādāt)");
              }
             if(BT_Commands [3]=="107") // nolasīt ierīces LĀZERA darba datus
              {

                Serial.println("Dati -> BT");
                ESP_BT.print("\nLazera laika intervāls = "+String(Laser_Delay_Time)+" min. \nLazera zonas skaits vienā sektorā = "+String(Laser_Zone));
                ESP_BT.print("\nDarba laiks 1 = "+String(Laser_Time_Border1)+" st. -> spilgtums1 = "+String(Laser_PWM_Border1));
                ESP_BT.print("\nDarba laiks 2 = "+String(Laser_Time_Border2)+" st. -> spilgtums2 = "+String(Laser_PWM_Border2));
                ESP_BT.print("\nDarba laiks 3 = "+String(Laser_Time_Border3)+" st. -> spilgtums3 = "+String(Laser_PWM_Border3));
                ESP_BT.print("\nDarba laiks 4 = "+String(Laser_Time_Border4)+" st. -> spilgtums4 = "+String(Laser_PWM_Border4));
                ESP_BT.print("\nLasera status = "+String(Laser_Status));
                ESP_BT.print("\n(0 - nestrādā(gaidīšanas režīms); 1- kustības kartes ģenerēšana; 2 - parking; 3 - PROGRAMMAS IZPILDE; 4 - aizlēgts strādāt)");
                ESP_BT.print("\n");
                ESP_BT.print("Sector1 H1 = "+String(Laser_Sector1_H1)+" Sector1 H2 = "+String(Laser_Sector1_H2)+"\n");
                ESP_BT.print("Sector2 H1 = "+String(Laser_Sector2_H1)+" Sector2 H2 = "+String(Laser_Sector2_H2)+"\n");
                ESP_BT.print("Sector3 H1 = "+String(Laser_Sector2_H1)+" Sector3 H2 = "+String(Laser_Sector2_H2)+"\n");

                ESP_BT.print("Sector1 V1 = "+String(Laser_Sector1_V1)+" Sector1 V2 = "+String(Laser_Sector1_V2)+"\n");
                ESP_BT.print("Sector2 V1 = "+String(Laser_Sector2_V1)+" Sector2 V2 = "+String(Laser_Sector2_V2)+"\n");
                ESP_BT.print("Sector3 V1 = "+String(Laser_Sector2_V1)+" Sector3 V2 = "+String(Laser_Sector2_V2)+"\n");
              }
             if(BT_Commands [3]=="108") // nolasīt skaņas failu secību gaisam
              {
                Serial.println("Dati -> BT");
                for (byte f = 0; f < 20; f++)
                  {
                    ESP_BT.print(String(Sound_Air_Numbers_Array [f])+", ");
                  }
              }
             if(BT_Commands [3]=="109") // nolasīt skaņas failu secību ūdenim
              {
                Serial.println("Dati -> BT");
                for (byte f = 0; f < 20; f++)
                  {
                    ESP_BT.print(String(Sound_Woter_Numbers_Array [f])+", ");
                  }
              }
             if(BT_Commands [3]=="201") // aktivizēt skaņas programmu
              {
                Sound_Air_Number = BT_Commands [4].toInt();
                Sound_Woter_Number = BT_Commands [5].toInt();
                Gaiss_1_Border = BT_Commands [6].toInt();
                Gaiss_2_Border = BT_Commands [7].toInt();
                Udens_1_Border = BT_Commands [8].toInt();
                Udens_2_Border = BT_Commands [9].toInt();
                Sound_Start_Time [0] = BT_Commands [10].toInt();
                Sound_Start_Time [1] = BT_Commands [11].toInt();
                Sound_Start_Time [2] = BT_Commands [12].toInt();
                Sound_Start_Time [3] = BT_Commands [13].toInt();
                Sound_Start_Time [4] = BT_Commands [14].toInt();

                Serial.println("==== aktivizēt skaņas programmu ====");
                Serial.println("Sound_Air_Number = "+String(Sound_Air_Number));
                Serial.println("Sound_Woter_Number = "+String(Sound_Woter_Number));
```

```
    Serial.println("Gaiss_1_Border = "+String(Gaiss_1_Border));
    Serial.println("Gaiss_2_Border = "+String(Gaiss_2_Border));
    Serial.println("Udens_1_Border = "+String(Udens_1_Border));
    Serial.println("Udens_2_Border = "+String(Udens_2_Border));

    Serial.print("Sound_Start_Time -> "+String(Sound_Start_Time [0])+" "+String(Sound_Start_Time [1]));
    Serial.println(" "+String(Sound_Start_Time [2])+" "+String(Sound_Start_Time [3])+" "+String(Sound_Start_Time [4]));
    Serial.println();
     EEPROM.begin(EEPROM_Size);
     EEPROM.write(30,Sound_Air_Number);
     EEPROM.write(31,Sound_Woter_Number);

     EEPROM.write(32,Gaiss_1_Border);
     EEPROM.write(33,Gaiss_2_Border);
     EEPROM.write(34,Udens_1_Border);
     EEPROM.write(35,Udens_2_Border);

     EEPROM.write(36,Sound_Start_Time [0]);
     EEPROM.write(37,Sound_Start_Time [1]);
     EEPROM.write(38,Sound_Start_Time [2]);
     EEPROM.write(39,Sound_Start_Time [3]);
     EEPROM.write(40,Sound_Start_Time [4]);
     EEPROM.end();
  }
if(BT_Commands [3]=="202") // aktivizēt lāzera programmu
  {
        Laser_Delay_Time = BT_Commands [4].toInt();
        Laser_Zone = BT_Commands [5].toInt();
        Laser_Time_Border1 = BT_Commands [6].toInt();
        Laser_PWM_Border1 = BT_Commands [7].toInt();
        Laser_Time_Border2 = BT_Commands [8].toInt();
        Laser_PWM_Border2 = BT_Commands [9].toInt();
        Laser_Time_Border3 = BT_Commands [10].toInt();
        Laser_PWM_Border3 = BT_Commands [11].toInt();
        Laser_Time_Border4 = BT_Commands [12].toInt();
        Laser_PWM_Border4 = BT_Commands [13].toInt();
        Serial.println("==== aktivizēt lāzera programmu ====");
        Serial.println("Laser_Delay_Time = "+String(Laser_Delay_Time));
        Serial.println("Laser_Zone = "+String(Laser_Zone));
        Serial.println("Laser_Time_Border1 = "+String(Laser_Time_Border1));
        Serial.println("Laser_PWM_Border1 = "+String(Laser_PWM_Border1));
        Serial.println("Laser_Time_Border2 = "+String(Laser_Time_Border2));
        Serial.println("Laser_PWM_Border2 = "+String(Laser_PWM_Border2));
        Serial.println("Laser_Time_Border3 = "+String(Laser_Time_Border3));
        Serial.println("Laser_PWM_Border3 = "+String(Laser_PWM_Border3));
        Serial.println("Laser_Time_Border4 = "+String(Laser_Time_Border4));
        Serial.println("Laser_PWM_Border4 = "+String(Laser_PWM_Border4));
        Serial.println();
        EEPROM.begin(EEPROM_Size);
        EEPROM.write(1,Laser_Delay_Time);
        EEPROM.write(2,Laser_Zone);
        EEPROM.write(3,Laser_Time_Border1);
        EEPROM.write(4,Laser_Time_Border2);
        EEPROM.write(5,Laser_Time_Border3);
        EEPROM.write(6,Laser_Time_Border4);
        EEPROM.write(7,Laser_PWM_Border1);
        EEPROM.write(8,Laser_PWM_Border2);
        EEPROM.write(9,Laser_PWM_Border3);
        EEPROM.write(10,Laser_PWM_Border4);
```

```
                    EEPROM.end();

                }

               if(BT_Commands [3]=="204") // ierakstīt skaņas failu secību
                {
                  Serial.println();
                  Serial.println(" ********** Bluettoth skaņas failu secības ierakstīšana **********");
                  Serial.println(" Number  |  Air  |  Woter");
                  for(byte q=0; q<19; q++)
                   {
                     Sound_Air_Numbers_Array [q]=BT_Commands [q+4].toInt();
                     Sound_Woter_Numbers_Array [q]=BT_Commands [q+24].toInt();

                     EEPROM.write(q+50,Sound_Air_Numbers_Array [q]);
                     EEPROM.write(q+80,Sound_Woter_Numbers_Array [q]);

                     Serial.println(String(q)+"  |  "+String(Sound_Air_Numbers_Array[q])+"  |  "+String(Sound_Woter_Numbers_Array[q]));
                   }



                }
              }
              else // Ja komanda citam modulim
                {
                   LoRaSendMessage(Text_Read_BT);
                   if (BT_Commands [3]=="104" || BT_Commands [3]=="105" || BT_Commands [3]=="106" || BT_Commands [3]=="107" || BT_Commands [3]=="108" || BT_Commands [3]=="109") // Ja citam modulim bija komanda "Nolasīt ierīces darba datus"
                    {
                      Serial.println("");
                      flag=0;
                      LoRa_Time = millis();
                      int LoRa_Count = 0;
                      while (flag==0 && millis()-LoRa_Time<30000) // Gaidam atbildi
                        {
                          LoRa_Message="";
                          LoRaReadMessage(LoRa.parsePacket());
                          delay(1000);
                          LoRa_Count++;
                          Serial.println("LoRa_Count="+String(LoRa_Count));
                        }
                      if (LoRa_Message!="")
                        {

                          Serial.println(" LoRa_Message  = " + LoRa_Message);
                          Serial.println("Dati -> BT");

                          LoRa_Parsing(LoRa_Message);
                          if(LoRa_Commands[4] == "104")
                           {
                             ESP_BT.print(" MODULIS "+String(LoRa_Commands[5]));
                             ESP_BT.print("\nGalv. akum. = "+String(LoRa_Commands[6])+"V \nPlates akum. = "+String(LoRa_Commands[7])+"V");
                             ESP_BT.print("\nGalvēnas plates temperatūra = "+String(LoRa_Commands[8])+" *C");

                             ESP_BT.print("\nTekošais laiks = "+String(LoRa_Commands[9]));
                             ESP_BT.print(":"+String(LoRa_Commands[10]));
                             ESP_BT.print("\n");
```

```
                    ESP_BT.print("\nLazera laika intervāls = "+String(LoRa_Commands[11])+" min. \nLazera zonas skaits vienā sektorā =
"+String(LoRa_Commands[12]));
                    ESP_BT.print("\nDarba laiks 1 = "+String(LoRa_Commands[13])+" st. -> spilgtums1 = "+String(LoRa_Commands[14]));
                    ESP_BT.print("\nDarba laiks 2 = "+String(LoRa_Commands[15])+" st. -> spilgtums2 = "+String(LoRa_Commands[16]));
                    ESP_BT.print("\nDarba laiks 3 = "+String(LoRa_Commands[17])+" st. -> spilgtums3 = "+String(LoRa_Commands[18]));
                    ESP_BT.print("\nDarba laiks 4 = "+String(LoRa_Commands[19])+" st. -> spilgtums4 = "+String(LoRa_Commands[20]));
                    ESP_BT.print("\nLasera status = "+String(LoRa_Commands[21]));
                    ESP_BT.print("\n");

                    ESP_BT.print("\nSkaņas masīva numurs gaisā = "+String(LoRa_Commands[22])+"\nSkaņas masīva numurs ūdenī =
"+String(LoRa_Commands[23]));
                    ESP_BT.print("\nLaika 1. robeža gaisā = "+String(LoRa_Commands[24])+" st. \nLaika 2. robeža gaisā =
"+String(LoRa_Commands[25])+" st.");
                    ESP_BT.print("\nLaika 1. robeža ūdenī = "+String(LoRa_Commands[26])+" st. \nLaika 2. robeža ūdenī =
"+String(LoRa_Commands[27])+" st.");
                    ESP_BT.print("\nIeslēgšanas laiks 0 = "+String(LoRa_Commands[28])+" sek. \nIeslēgšanas laiks 1 =
"+String(LoRa_Commands[29])+" sek.");
                    ESP_BT.print("\nIeslēgšanas laiks 2 = "+String(LoRa_Commands[30])+" sek. \nIeslēgšanas laiks 3 =
"+String(LoRa_Commands[31])+" sek.");
                    ESP_BT.print("\nIeslēgšanas laiks 4 = "+String(LoRa_Commands[32])+" sek. \nSkaņas status =
"+String(LoRa_Commands[33]));
                    ESP_BT.print("\n(4 - NAV atļauts strādāt; 1 - IR atļauts strādāt)");
                   }
                  if(LoRa_Commands[4] == "105")
                   {
                    ESP_BT.print(" MODULIS "+String(LoRa_Commands[5]));
                    ESP_BT.print("\nGalv. akum. = "+String(LoRa_Commands[6])+"V \nPlates akum. = "+String(LoRa_Commands[7])+"V");
                    ESP_BT.print("\nGalvēnas plates temperatūra = "+String(LoRa_Commands[8])+" *C");
                   }
                  if(LoRa_Commands[4] == "106")
                   {
                    ESP_BT.print(" MODULIS "+String(LoRa_Commands[5])+"\n");
                    ESP_BT.print("\nSkaņas masīva numurs gaisā = "+String(LoRa_Commands[22])+"\nSkaņas masīva numurs ūdenī =
"+String(LoRa_Commands[23]));
                    ESP_BT.print("\nLaika 1. robeža gaisā = "+String(LoRa_Commands[24])+" st. \nLaika 2. robeža gaisā =
"+String(LoRa_Commands[25])+" st.");
                    ESP_BT.print("\nLaika 1. robeža ūdenī = "+String(LoRa_Commands[26])+" st. \nLaika 2. robeža ūdenī =
"+String(LoRa_Commands[27])+" st.");
                    ESP_BT.print("\nIeslēgšanas laiks 0 = "+String(LoRa_Commands[28])+" sek. \nIeslēgšanas laiks 1 =
"+String(LoRa_Commands[29])+" sek.");
                    ESP_BT.print("\nIeslēgšanas laiks 2 = "+String(LoRa_Commands[30])+" sek. \nIeslēgšanas laiks 3 =
"+String(LoRa_Commands[31])+" sek.");
                    ESP_BT.print("\nIeslēgšanas laiks 4 = "+String(LoRa_Commands[32])+" sek. \nSkaņas status =
"+String(LoRa_Commands[33]));
                    ESP_BT.print("\n(4 - NAV atļauts strādāt; 1 - IR atļauts strādāt)");
                   }
                  if(LoRa_Commands[4] == "107")
                   {
                    ESP_BT.print(" MODULIS "+String(LoRa_Commands[5])+"\n");
                    ESP_BT.print("\nLazera laika intervāls = "+String(LoRa_Commands[11])+" min. \nLazera zonas skaits vienā sektorā =
"+String(LoRa_Commands[12]));
                    ESP_BT.print("\nDarba laiks 1 = "+String(LoRa_Commands[13])+" st. -> spilgtums1 = "+String(LoRa_Commands[14]));
                    ESP_BT.print("\nDarba laiks 2 = "+String(LoRa_Commands[15])+" st. -> spilgtums2 = "+String(LoRa_Commands[16]));
                    ESP_BT.print("\nDarba laiks 3 = "+String(LoRa_Commands[17])+" st. -> spilgtums3 = "+String(LoRa_Commands[18]));
                    ESP_BT.print("\nDarba laiks 4 = "+String(LoRa_Commands[19])+" st. -> spilgtums4 = "+String(LoRa_Commands[20]));
                    ESP_BT.print("\nLasera status = "+String(LoRa_Commands[21]));
                   }
                 }
                 else
                  {
```

```
                    Serial.println(" LoRa ERROR  1");
                    ESP_BT.print("LoRa ERROR,"); // Tā arī nesaņemam cita moduļa atbildi
                   }

                }
                else // Ja bija citas komandas
                 {
                   Serial.println("");
                  flag=0;
                  LoRa_Time = millis();
                  while (flag==0 && millis()-LoRa_Time<10000) // Gaidam atbildi
                    {
                      LoRaReadMessage(LoRa.parsePacket());
                      delay(1000);
                    }
                  if (millis()-LoRa_Time < 10000)
                    {
                      //ESP_BT.print(LoRa_Message); // Pārsūtam atbildi viedtālrunim
                      Serial.println(" LoRa_Message  = " + LoRa_Message);
                      Serial.println("Dati -> BT");
                      ESP_BT.print(LoRa_Message);
                    }
                 }


              }


          }

        }
    }
    Laser_Status=0;
    Serial.println();
    Serial.println(" ********** Izlegts Bluettoth modulis **********");

  }
 // ********** Bluettoth (beigas) **********


// ********** Lāzers (sakums) **********
 if(Module_Tipe=="MASTER" || Module_Tipe=="SLAVE_LASER")
  {
    if(Laser_Status!=4)
     {
       switch (Laser_Status)
        {
          case 0: // Nakama cikla gaidīšana
            if((millis()-Laser_Time2 >= Laser_Delay_Time*1000*60))
             {
               Laser_Status = 5;
               Serial.print("  ");
               Serial.print(millis());
               Serial.println("-"+String(Laser_Time2));
               Laser_Parking_Left();
             }
          break;
          case 5: // Parking (pa kreisi)
            if((millis()-Laser_Time1 >= Laser_Parking_Time))
              {
```

```
        Serial.print(" Laser Parking (LEFT) Start ="+String(Laser_Time1)+" - Laser Parking Stop =");
        Serial.print(millis());
        Serial.print(" -> ");
        Serial.println((millis() - Laser_Time1)/1000);

        Laser_Status = 6; // Tālāk būs lazera kustības programmas izpilde
        //Laser_Nr = 1; // Lāzera kartes punkta numurs
        Laser_Angle_H = 0;

        Serial.println(" ===== Lazera 1 sektora sakums ======");
        Serial.println("    (Tekošais leņķis = "+String(Laser_Angle_H)+")");
      }
      else  WDT_Reset();
break;
case 6: // lazera kustības programmas izpilde (1 sektors)

   Serial.println(" Horizontal angle = " + String(Laser_Angle_H) + "  Laser_Sector1_H2 = "+String(Laser_Sector1_H2));
   if (Laser_Angle_H < Laser_Sector1_H2-5)
    {
      Laser_Power("ON"); // Laser ON
      if(Laser_Move_Direction == 0)
       {
         SendData_PelcoD_Moving(Up);
         delay(Laser_Vertical_Time);
       }
      if(Laser_Move_Direction == 1)
       {
         SendData_PelcoD_Moving(Down);
         delay(Laser_Vertical_Time+(Laser_Vertical_Time/2));
       }

      SendData_PelcoD(Stop);
      Laser_Power("OFF"); // Laser OFF
      Laser_Angle_H = Laser_Angle_H + 5;
      Laser_Move_Direction = !Laser_Move_Direction;

      Serial.println(" New horizontal angle = " + String(Laser_Angle_H));
      Laser_Work_Time = (5/Laser_Speed_H)*1000;
      Serial.println("  Laser_Work_Time="+String(Laser_Work_Time)+" ms");
      SendData_PelcoD_Moving(Right);
      delay(Laser_Work_Time);
      SendData_PelcoD(Stop);
      //Serial.println("Laser_Status = "+String(Laser_Status));
    }
    else
     {
       Laser_Status = 7; // Parking pa labi
       Serial.println("Laser_Status = "+String(Laser_Status));
       Laser_Move_Direction = 0;
       Laser_Parking_Right();
     }
break;
case 7: // Parking (pa labi)
  //Serial.println("Laser_Status (Parking (pa labi)) = "+String(Laser_Status));
  if((millis()-Laser_Time1 >= Laser_Parking_Time))
    {
      Serial.print(" Laser Parking (RIGHT) Start ="+String(Laser_Time1)+" - Laser Parking Stop =");
      Serial.print(millis());
      Serial.print(" -> ");
      Serial.println((millis() - Laser_Time1)/1000);
```

```
            Laser_Status = 8; // Tālāk būs lazera kustības programmas izpilde
            //Laser_Nr = 1; // Lāzera kartes punkta numurs
            Laser_Angle_H = 0;

            Serial.println(" ===== Lazera 2 sektora sakums ======");
            Serial.println("    (Tekošais leņķis = "+String(Laser_Angle_H)+")");
           }
           else  WDT_Reset();
         break;
         case 8: // lazera kustības programmas izpilde (2 sektors)
           if (Laser_Zone > 3) // Сейчас параметр не нужен, просто использую для выбора выполнять/не выполнять второй сектор
            {

           Serial.println(" Horizontal angle = " + String(Laser_Angle_H) + "      Laser_Sector1_H2 = "+String(Laser_Sector1_H2)+"
Laser_Work_Time="+String(Laser_Work_Time)+" ms");
           if (Laser_Angle_H < Laser_Sector3_H1-5)
            {
              Laser_Power("ON"); // Laser ON
              if(Laser_Move_Direction == 0)
               {
                 Laser_Work_Time = (Laser_Sector3_V2/Laser_Speed_V)*1000;
                 Serial.println(" Laser_Work_Time="+String(Laser_Work_Time)+" ms");
                 SendData_PelcoD_Moving(Up);
                 delay(Laser_Work_Time);
               }
              if(Laser_Move_Direction == 1)
               {
                 Laser_Work_Time = (Laser_Sector3_V2/Laser_Speed_V)*1000;
                 Serial.println(" Laser_Work_Time="+String(Laser_Work_Time)+" ms");
                 SendData_PelcoD_Moving(Down);
                 delay(Laser_Work_Time+(Laser_Work_Time/2));
               }

              SendData_PelcoD(Stop);
              Laser_Power("OFF"); // Laser OFF
              Laser_Angle_H = Laser_Angle_H + 5;
              Laser_Move_Direction = !Laser_Move_Direction;

              Serial.println(" New horizontal angle = " + String(Laser_Angle_H));
              Laser_Work_Time = (5/Laser_Speed_H)*1000;
              Serial.println("  Laser_Work_Time="+String(Laser_Work_Time)+" ms");
              SendData_PelcoD_Moving(Left);
              delay(Laser_Work_Time);
              SendData_PelcoD(Stop);
              //Serial.println("Laser_Status = "+String(Laser_Status));
            }
            else
             {
               Laser_Status = 0; // Nakama cikla gaidīšana
               Serial.println("Laser_Status = "+String(Laser_Status));
               Laser_Move_Direction = 0;

             }
            }
            else
             {
               Serial.println("*****************************************************");
               Serial.println("      Lazera 2 sektora programma netiks izpildita    ");
               Serial.println("*****************************************************");
```

```
                Laser_Status = 0; // Nakama cikla gaidīšana
                Serial.println("Laser_Status = "+String(Laser_Status));
                Laser_Move_Direction = 0;
               }


          break;



        }
      }
    }
 // ********************************** Lāzers (beigas) ***************************************

}
```