**RĪGAS TEHNISKĀ UNIVERSITĀTE**

Liepājas akadēmija

Jānis Birznieks

Mehatronika 3.kurss

191LMC006

# Autonomais saules pulkstenis

Students

Darba vadītājs

Liepāja 2025

**Saturs**

# Ievads

Saules pulksteņi ir vieni no senākajiem laika noteikšanas instrumentiem, kas darbojas, izmantojot Saules pozīciju un ēnas krišanas virzienu, lai norādītu aptuvenu diennakts laiku. Lai tie pareizi darbotos, saules pulkstenim jābūt orientētam pret īstajiem ziemeļiem, jo tieši no šīs puses tiek veikti ģeogrāfiskie novērojumi. Turklāt gnomona — ēnas metēja — leņķim jāatbilst lietotāja ģeogrāfiskajai platuma grādu vērtībai, lai ēna precīzi rādītu laiku.

Mūsdienās, kad pulksteņi un viedierīces ir plaši pieejamas un ļoti precīzas, saules pulksteņi vairāk tiek izmantoti kā dekoratīvi vai izglītojoši objekti. Tomēr šo klasisko laika noteikšanas principu var apvienot ar mūsdienu tehnoloģijām, radot automatizētu saules pulksteni, kas spēj patstāvīgi noteikt savu orientāciju un pielāgoties lietotāja atrašanās vietai.

Šī projekta mērķis ir izstrādāt ierīci, kura, izmantojot mikrodatoru un sensorus, automātiski noregulē savu pozīciju un gnomona leņķi, nodrošinot precīzu un ērtu laika nolasīšanu no saules pulksteņa neatkarīgi no tā, kur ierīce tiek novietota.

Projekts apvieno vairākas tehnoloģiskās jomas — mehānisko konstrukciju, sensoru datu apstrādi, mikrokontrolieru un mikrodatoru komunikāciju, kā arī algoritmu izstrādi. Lai gan šīs ierīces praktiskā nozīme mūsdienās ir ierobežota, projekts sniedz vērtīgu pieredzi inženiertehniskajā risinājumā, apgūstot sensoru integrāciju un automatizētu kontroli.

*Attēls 1, Autonomā saules pulksteņa elektromehāniskais izpildījums*

# Tēmas pamatojums

Šī projekta izvēle balstās uz vēlmi apvienot klasisku, analogā laikmeta tehnoloģiju – saules pulksteni – ar mūsdienu digitālajām iespējām, demonstrējot, kā tradicionālus mehānismus iespējams modernizēt ar mikroelektroniku, automatizāciju un sensoriem. Lai arī izveidotā ierīce būtībā atkārto funkciju, kuru mūsdienās veic jebkurš viedtālrunis vai pulkstenis, šī "tehnoloģiski pārsarežģītā" pieeja ļauj padziļināti izprast:

- analogā un digitālā signālu apstrādi;
- dažādu sensoru un aktuatoru pielietošanu;
- ģeogrāfisko koordinātu izmantošanu aprēķinos;
- mikrokontrolleru un mikrodatoru sadarbību;
- automatizētu mehānismu vadības algoritmu izstrādi.

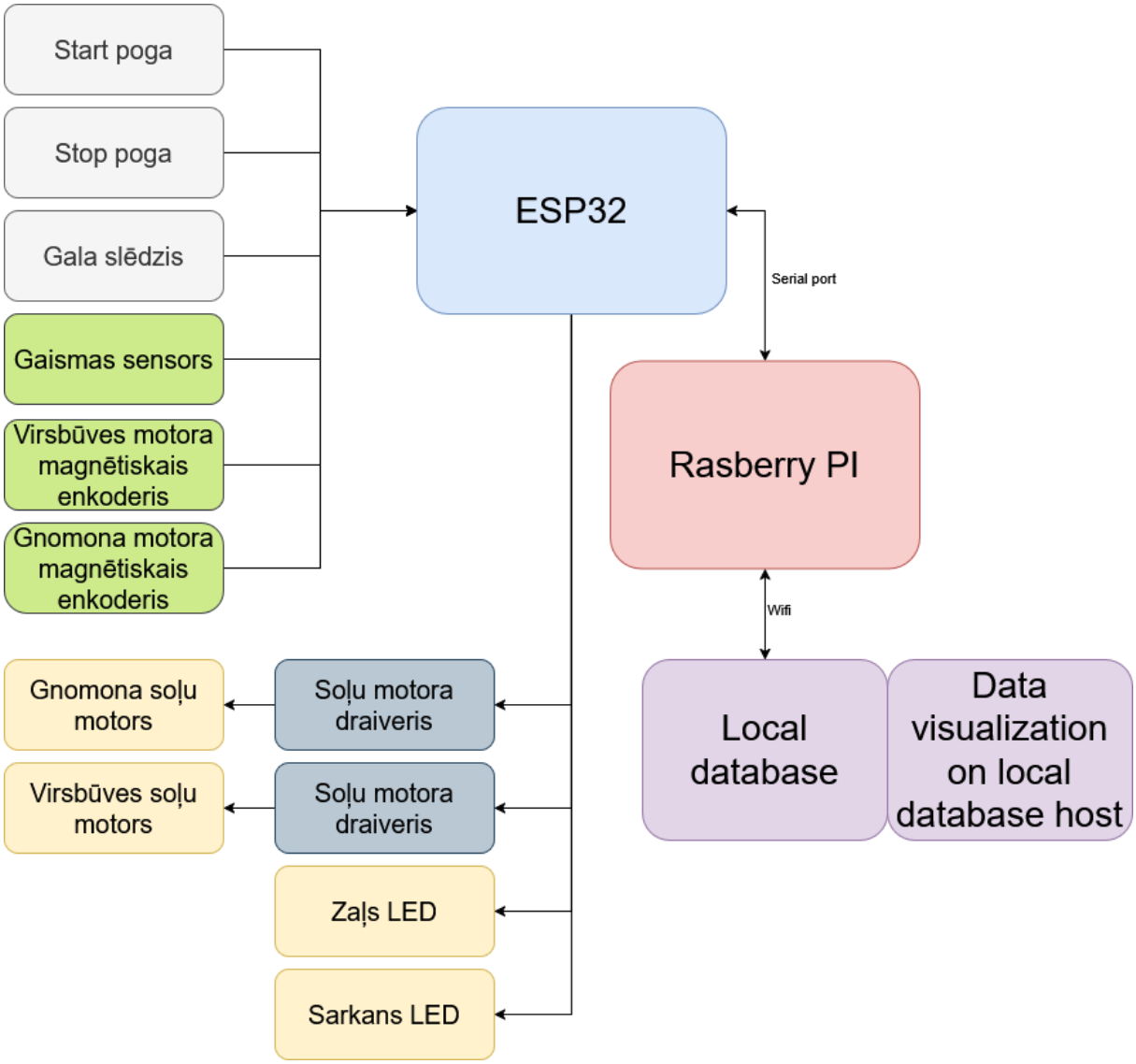
Tēmas izvēle arī parāda, kā ar šķietami vienkāršu un "nepraktisku" ideju iespējams radīt daudzslāņainu projektu, kurā nepieciešama nopietna tehniskā plānošana un īstenošana. Šāda pieeja ne tikai ļauj demonstrēt dažādu jomu prasmes, bet arī motivē radošu pieeju inženiertehnisku risinājumu meklējumos.

# Prasību specifikācija

**Funkcionālās prasības:**

- **Sistēmas ieejas (nolasāmie parametri):**
    - **Lietotāja atrašanās vieta** (platums un garums) – automātiski nolasīta (caur RasPi).
    - **Laiks** – iegūts no Raspberry Pi (interneta sinhronizācija).
    - **Gaismas sensori** – nosaka, no kurienes nāk spēcīgākais apgaismojums (analoga ievade).
    - **Enkoderi uz soļu motoriem** – precīzai pozīcijas kontrolei un kļūdu korekcijai (ja gnomons vai pulkstenis kustības laikā tiek bloķēts).
    - **Divas pogas** – ierīces ieslēgšana/izslēgšana, pauze vai attiestatīšana
- **Aktuatora darbības atkarība no ievades:**
    - **Soļu motori** rotē platformu, lai to orientētu pret sauli, un pielāgo gnomona leņķi atkarībā no platuma.
    - Sistēma automātiski aprēķina saules azimutu no laika un koordinātēm.
    - Pozicionēšana tiek veikta, balstoties uz aprēķināto saules pozīciju, ar atsauci no enkoderiem.
- **Datu apstrāde / pārsūtīšana / vizualizācija:**
    - **ESP32** saņem datus no **Raspberry Pi** (laiks, atrašanās vieta).
    - **LED indikatori**:
        - Zaļais – sistēma ir iestatījusies.
        - Dzeltenais - notiek sistēmas darbība,
        - Sarkanais – sistēma izslēgta.
    - **ESP32** pārsūta aziumta leņķi, gaismas sensoru vērtības, platuma leņķi, sistēmas stāvokli uz **Raspberry Pi,** kas to tālāk nosūta uz lokālo datubāzi.
    - **Datubāzes iekārta** veic datu vizualizāciju un apstrādi.
- **Lietotāja saskarne:**
    - 2 pogas.
    - 2 LED (zaļš un sarkans).

Moduļu slēgums (komponentes)




*Attēls 2, Moduļu slēgums*

## Moduļu slēguma apraksts

Šajā sistēmā izmantota **modulāra pieeja**, kurā mikrokontrolleris **ESP32** pārvalda sensorus, pogas, motorus un indikatorus, bet **Raspberry Pi** mikrodators reģistrē lokāciju un laika datus, kā arī pārsūta informāciju uz datubāzi.

### ESP32 mezgls

ESP32 kalpo kā centrālais kontrolieris, kurš veic:

- **Signālu nolasīšanu no ieejām:**
  - **Start poga** – aktivizē sistēmas ciklu.

- **Stop poga** – iedarbina pauzes vai izslēgšanas režīmu.
- **Gala slēdzis** – kalpo kā gnomona references punkts.
- **Gaismas sensori (LDR)** – noteic Saules virzienu.
- **Magnetiskie enkoderi** – reģistrē bāzes un gnomona motora pozīciju, izmantojot analogās vērtības no AS5600 sensoriem.

- **Signālu nosūtīšanu uz izejām:**

  - **Soļu motora draiveri** – vadības signāli tiek sūtīti uz bāzes un gnomona motoriem (DRV8825), lai izpildītu kustības uz noteiktiem leņķiem.

  - **LED indikatori:**

    - **Zaļais LED** - sistēma ir pabeigusi darbību.
    - **Dzletenais LED -** sistēma ir darbībā vai pauzē.
    - **Sarkanais LED** - sistēma ir izslēgta.

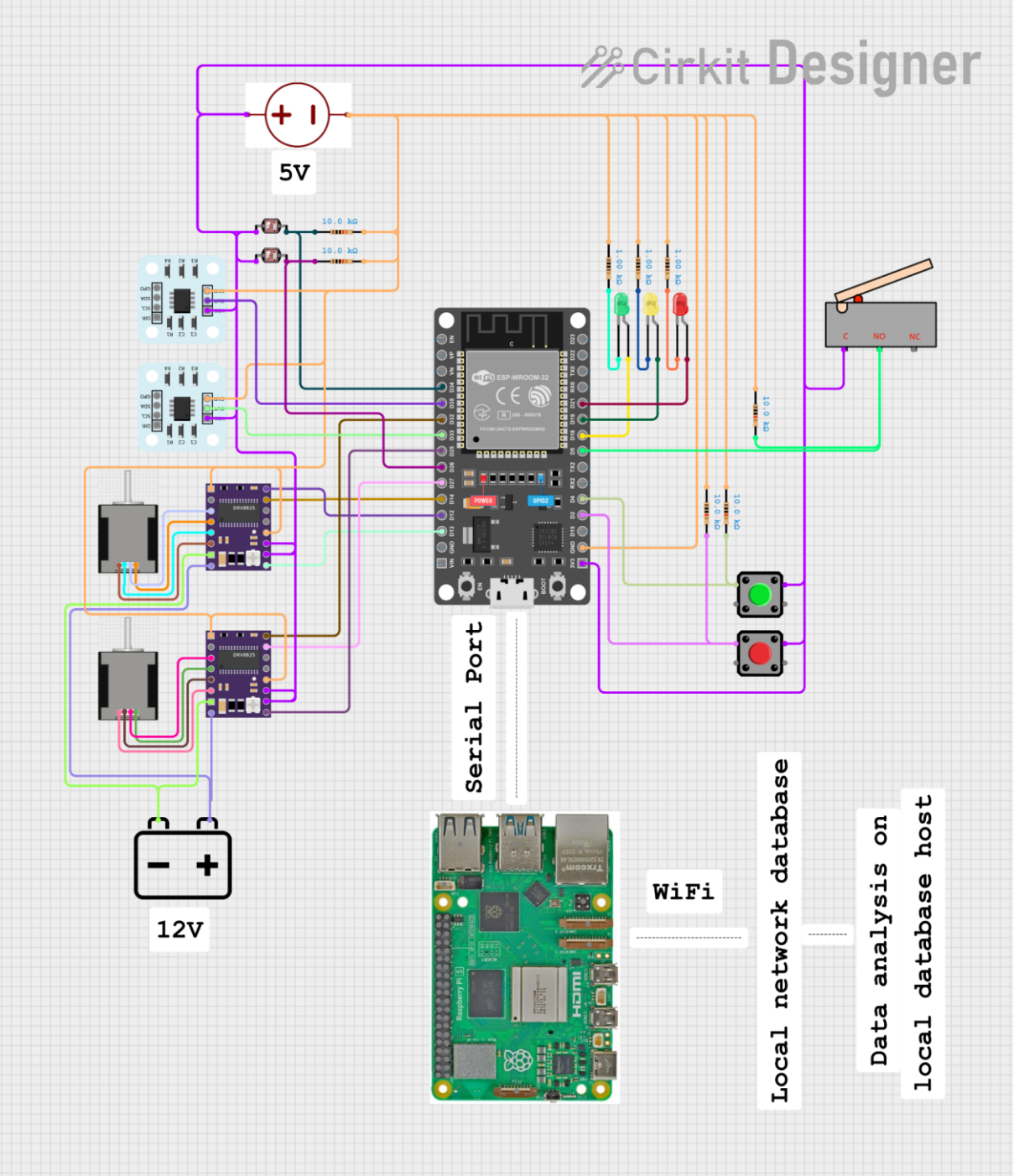
ESP32 arī veic aprēķinus, piemēram, pozicionēšanos pēc gaismas sensoriem un gnomona leņķa iestatīšanu pēc lietotāja koordinātām un laika.

**Raspberry Pi mezgls**

**Raspberry Pi** saņem seriālos datus no ESP32 caur serial port. Tas darbojas kā starpnieks starp ierīci un **vietējo MySQL datubāzi** – glabā datus par laiku, atrašanās vietu, Saules azimutu, sensoru mērījumiem, sistēmas stāvokli u.c. Datubāzes ierīcē notiek arī **datu vizualizācijas** – reāllaikā attēlo:

- Saules azimuta un gnomona leņķa polārās diagrammas,
- LDR vērtību vēsturi,
- sistēmas stāvokļa indikatoru.

# Principiālā shēma

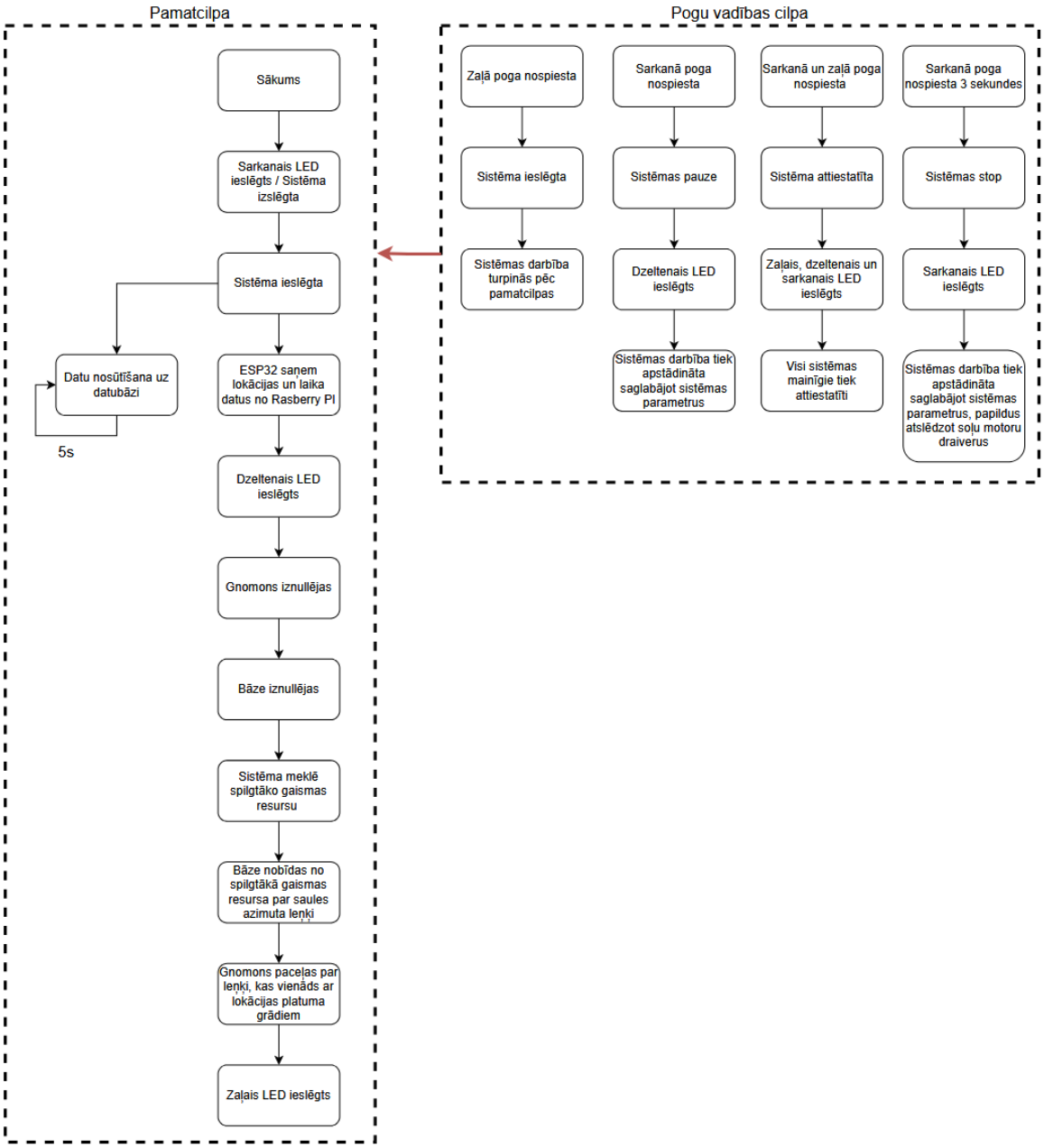



*Attēls 3, Principiālā shēma*

| Komponente | Pins | Savienots ar |
|---|---|---|
| NEMA 17 | GREEN (A1) | DRV8825 A1 |
| | BLUE (A2) | DRV8825 A2 |
| | BLACK (B1) | DRV8825 B1 |
| | RED (B2) | DRV8825 B2 |
| NEMA 14 | RED (B2) | DRV8825 B2 |
| | BLACK (B1) | DRV8825 B1 |
| | GREEN (A1) | DRV8825 A1 |
| | BLUE (A2) | DRV8825 A2 |
| 12V barošana | | DRV8825 VMOT, VMOT, GND |
| ESP32 | D14 | DRV8825 STEP |
| | D12 | DRV8825 DIR |
| | D13 | DRV8825 EN |
| | D27 | DRV8825 STEP |
| | D32 | DRV8825 DIR |
| | D25 | DRV8825 EN |
| Photocell (LDR1) | Pin 1 | Resistor, ESP32 D34 |
| Photocell (LDR2) | Pin 1 | ESP32 D26 |
| Galaslēdzis | NO | ESP32 D5 |
| Pushbutton | Pin 1 | ESP32 D2 |
| Pushbutton | Pin 1 | ESP32 D4 |
| LED (green) | Anode | ESP32 D18 |
| LED (yellow) | Anode | ESP32 D19 |
| LED (red) | Anode | ESP32 D21 |

*Tabula 1, Slēguma tabula*

# Algoritms




*Attēls 4, Programmas algoritma blokshēma*

**Galvenā cikla darbība**

Sistēmai pirmoreiz uzsākot darbu, tiek ieslēgts **sarkans indikators**, norādot, ka sistēma ir izslēgta vai gaidstāvē. Kad lietotājs to ieslēdz, iedegas **dzeltens LED**, un sākas sistēmas darbības process . Vispirms ESP32 saņem **reālā laika informāciju** un **atrašanās vietu** no Raspberry Pi (kas to attiecīgi ir saņēmis no interneta), un no šiem datiem aprēķina Saules azimuta leņķi.Tālāk gan bāze, gan gnomons tiek atgriezti sākuma pozīcijā.

Kad tas paveikts, sistēma izmanto LDR (gaismas sensorus), lai atrastu spilgtāko gaismas avotu. Pēc tam, pamatojoties uz šo informāciju:

- **Bāze tiek novirzīta** par saules azimuta leņķi no spilgtākā gaismas avota.
- **Gnomons paceļas** atbilstoši lokācijas platuma grādiem.

Sistēma aktivizē **zaļo LED**, kas norāda uz normālu darbību.Visu darbības laiku,ik pēc **5 sekundēm** sistēma nosūta datus uz MySQL datubāzi, tai skaitā:

- laiku,
- aprēķināto Saules azimutu,
- LDR sensoru rādījumus,
- lokācijas platuma leņķi,
- sistēmas stāvokli.

**Pogu vadības apakšcikls**

Šis cikls darbojas **neatkarīgi no galvenā cikla** un ļauj lietotājam jebkurā brīdī mainīt sistēmas stāvokli:

- **Sarkanā poga** aktivizē **pauzes režīmu**, apstādinot motorus. Iedegas dzeltens LED.
- **Zaļā poga** atsāk darbību pēc pauzes.
- **Sarkanā + zaļā poga** kopā izraisa **atiestatīšanu** – visi mainīgie tiek atgriezti sākotnējā stāvoklī.
- **Sarkanā poga, turot to nospiestu 3 sekundes**, izslēdz sistēmu, deaktivizē motoru draiverus – aktivizējas sarkanais LED.

# Kods (apraksts)

**P**rogrammatūras sistēma ir paredzēta, lai fizikāli izkārtotu saules pulksteni atbilstoši saules pašreizējai pozīcijai, izmantojot reāllaika atrašanās vietas un laika datus. Sistēma sastāv no trim savstarpēji savienotām ierīcēm, kurās katrā darbojas specifiska programma:

- **ESP32** — *Autonomous Sundial (main.cpp)*: Vada aktuatorus, nolasa sensorus un izpilda visu saules pulksteņa mehānisko kustību.
- **Raspberry Pi** — *ESPSerialSender.py* : Iegūst ģeolokāciju un laiku no interneta, sazinās ar ESP32 un pārsūta sistēmas datus uz datubāzi.
- **Datu bāzes ierīce** — *Sundial_Plots.py* : Iegūst datus no MySQL datubāzes un vizualizē sistēmas statusu, sensoru rādījumus un saules orientāciju reāllaikā.

### ESP32 — *Autonomous Sundial* (`main.cpp`)

#### Programmas mērķis

Šī programma darbojas uz **ESP32** un veido **sistēmas galveno vadības loģiku**. Tās uzdevumi ir:

- Vadīt **bāzes un gnomona motorus**, izmantojot soļu motoru draiverus un analogo sensoru atgriezenisko saiti.
- **Saņemt laiku un atrašanās vietas datus** no Raspberry Pi.
- **Aprēķināt saules pozīciju** ar saules pozicionēšanas bibliotēku.
- Izmantot **gaismas sensorus (LDR)**, lai nomērītu saules atrašanās pozīciju.
- Apstrādāt **lietotāja ievadi** caur pogām un norādīt **sistēmas stāvokli** ar LED indikatoriem.
- Nosūtīt sistēmas statusu un datus atpakaļ uz Raspberry Pi caur seriālo savienojumu.

Programma organizēta kā **state machine**, kas pārvalda sistēmas darbību (OFF, ON, PAUSED, RESET), un **sub-state machine**, kas veic saules pulksteņa uzstādīšanas un pozicionēšanas procedūru.

#### Galvenās apakšprogrammas un funkcijas

Zemāk aprakstīti katras funkcijas uzdevumi:

#### 1. `void resetMotorStates()`

Atiestata visu motoru stāvokļu mainīgos. Izsaucas pēc sistēmas restartēšanas, lai motori sāktu no nulles pozīcijas.

#### 2. `bool trackLightSource(...)`

**LDR balstīta gaismas avota izsekošana**:

- Salīdzina kreisās un labās LDR sensoru vērtības.
- Groza bāzes motoru, līdz sensori ir līdzsvaroti **tolerances robežās**.
- Ņem vērā spilgtāko gaismas avotu
  Šis posms palīdz noteikt saules aptuveno pozīciju.

### 3. `bool rotateMotorByOffset(...)`

Groza motoru par **relatīvu leņķi** (piemēram, saules azimuta nobīdi):

- Nolasa pašreizējo pozīciju no AS5600 sensora.
- Aprēķina mērķa pozīciju kā: pašreizējā + nobīde.
- Kustība turpinās, līdz tiek sasniegta mērķpozīcija.

### 4. `bool rotateMotorToPosition(...)`

Groza motoru uz **konkrētu absolūtu pozīciju**:

- Līdzīga kā `rotateMotorByOffset`, bet ar fiksētu mērķi (piemēram, sākuma pozīciju).
- Lieto bāzes vai gnomona "nulēšanai".

### 5. `void checkButtons()`

Apstrādā **pogu ievades**, izmantojot `Bounce2` bibliotēku:

- Zaļā + sarkanā poga → sistēmas **RESET**
- Zaļā poga → sistēma **ON**
- Sarkanā poga → sistēma **PAUSED**
- Sarkanās pogas turēšana 3 sekundes → sistēma **OFF**

### 6. `void calculateSunAzimuth()`

Aprēķina **saules pašreizējo azimutu**, izmantojot laiku un atrašanās vietu:

- Izmanto saules pozīciju bibliotēku.
- Leņķis pārveidots motoru mērogā (0–4096).

### 7. `void parseSerialCommand(const String& input)`

Apstrādā ienākošās seriālās komandas, kas atjauno sistēmas iekšējos mainīgos laika un vietas datiem.:

- `TIME:<hh>,<mm>,<ss>,<dd>,<mm>,<yyyy>`
- `LOC:<platums>,<garums>`

### 8. `void receiveSerialData()`

Nolasa seriālo buferi un nodo ievadītos datus `parseSerialCommand` funkcijai. Nodrošina, ka ESP32 spēj saņemt Raspberry Pi sūtītos datus.

### 9. `void sendSystemData()`

Nosūta telemetrijas datus atpakaļ uz Raspberry Pi:

- Iekļauj: laika zīmogu, platumu, garumu, saules azimutu, LDR vērtības, bāzes leņķi, sistēmas stāvokli.

- Dati formatēti kā teksta rinda ar `DATA:` prefiksu.

**Sistēmas inicializācija un galvenais cikls**

### `void setup()`

Inicializē:

- Seriālo savienojumu
- Motora "enable" pinus
- Pogas un LED pinus
- Pogu atskaites laiku (`debounce`)
- Sākas `SYSTEM_OFF` stāvoklī.

### `void loop()`

Galvenā programma strukturēta ap **stāvokļu mašīnu**:

**Sistēmas stāvokļi**

1. **SYSTEM_OFF**
   - Motori atspējoti.
   - Iedegas sarkanais LED.
   - Sistēma gaida.
2. **SYSTEM_ON**
   Aktivizē **apakšstāvokļu** secību, lai sagatavotu un pozicionētu saules pulksteni:
   **Apakšstāvokļi:**
   - `STEP_ZERO_GNOMON`: Gnomons tiek rotēts līdz mehāniskajai nulles pozīcijai (izmantojot gala slēdzi).
   - `STEP_ZERO_BASE`: Bāze tiek noregulēta uz fiksētu sākuma pozīciju (izmantojot sensoru).
   - `SUN_TRACKING`: Bāze tiek virzīta uz spožāko gaismas avotu (izmantojot LDR sensorus).
   - `STEP_ROTATE_BASE`: Bāze tiek rotēta atbilstoši saules azimuta aprēķinam.
   - `STEP_ROTATE_GNOMON`: Gnomons tiek pacelts līdz vietējam **platuma leņķim**.
   - `STEP_DONE`: Visi soļi pabeigti. Zaļais LED iedegas. Sistēma ir noregulēta.
3. **SYSTEM_PAUSED**
   - Motori apstājas.
   - Dzeltens LED iedegas.
4. **SYSTEM_RESET**
   - Atiliek sistēmu uz sākuma stāvokli.
   - Gaidīt laika un vietas datus.

**Seriālā izvade**

Cikla beigās ik **pēc 5 sekundēm** tiek izsaukta `sendSystemData()` funkcija, kas nosūta aktuālos datus uz Raspberry Pi.

### Raspberry Pi — `ESPSerialSender.py`

#### Programmas mērķis

Šī Python programma darbojas uz **Raspberry Pi** un kalpo kā **saziņas un datu apmaiņas tilts** starp ESP32 un datubāzes serveri. Tās uzdevumi ir:

- **Iegūt reāllaika laiku un atrašanās vietas datus** no interneta, izmantojot iebūvēto sistēmas pulksteni un `geocoder` bibliotēku.
- Nosūtīt šo informāciju ESP32, lai tas spētu aprēķināt saules pozīciju.
- **Saņemt sistēmas datus** no ESP32 seriālajā formātā un **saglabāt tos MySQL datubāzē**.

Programma darbojas **nepārtrauktā ciklā**, regulāri atjauninot informāciju un uzraugot ESP32 izsūtītos datus.

#### Galvenās funkcijas

##### `send_time_and_location()`

- Iegūst **lokālo laiku**.
- Mēģina noteikt **ģeogrāfiskās koordinātas** (platumu un garumu) ar IP balstītu geolokāciju.
- Ja tas neizdodas, izmanto **rezerves vērtības**.
- Nosūta datus uz ESP32 divās atsevišķās komandās: `TIME:` un `LOC:`.

##### `read_esp_data()`

- Klausās seriālajā portā.
- Ja saņemta rinda, kas sākas ar `DATA:`, tā tiek parsēta 8 laukos:
    - Laika zīmogs
    - Platums
    - Garums
    - Saules azimuts
    - LDR kreisais sensors
    - LDR labais sensors
    - Bāzes pozīcija
    - Sistēmas stāvoklis

- Pēc veiksmīgas parsēšanas dati tiek **ierakstīti MySQL datubāzē** ar `insert_data_to_mysql(...)`.

##### `insert_data_to_mysql(...)`

- Veic SQL `INSERT` vaicājumu tabulā `sun_tracking_data`, saglabājot visu ienākošo telemetriju strukturētā veidā.

#### Galvenais cikls

Programma darbojas bezgalīgā `while True` ciklā:

- Ik **pēc 5 sekundēm** tiek izsaukta `send_time_and_location()`, lai nodrošinātu ESP32 ar aktuāliem datiem.
- **Nepārtraukti** tiek klausīts ESP32 seriālais izvads, un saņemtie dati tiek apstrādāti.
- Starp cikliem programma īslaicīgi "guļ", lai mazinātu CPU noslodzi (`sleep(0.9)`).

**Kopsavilkums**

Šī programma ir vienkārša, bet būtiska — tā **sinhronizē ESP32 ar reālo pasauli** (laiku un atrašanās vietu) un **nodrošina telemetrijas plūsmu** uz datubāzi.

### Datu bāzes ierīce — `Sundial_Plots.py`

**Programmas mērķis**

Šī Python programma darbojas uz **vietējās datubāzes ierīces** un nodrošina **reāllaika vizualizāciju** par saules pulksteņa sistēmas darbību. Tās uzdevumi ir:

- **Savienoties ar MySQL datubāzi** un nolasīt jaunākos telemetrijas ierakstus.
- **Attēlot datus vizuāli**, izmantojot `matplotlib`, sniedzot lietotājam skaidru priekšstatu par sistēmas stāvokli, saules pozīciju, sensoru rādījumiem u.c.
- Atjaunot grafikus **ik pēc 5 sekundēm**

**Vizualizācijas paneļi**

Programma izveido četrus galvenos skatījumus (`subplot`) vienā logā:

**1. Saules azimuts (polārais grafiks)**

- Attēlo saules virzienu uz kompasa diska.
- 0° = Ziemeļi; grafiks orientēts ar pulksteņrādītāja virzienu.
- Dati tiek iegūti no pēdējā `sun_azimuth` ieraksta datubāzē.

**2. Gnomona slīpums (polārais grafiks)**

- Attēlo platuma leņķi lietotāja lokācijā.

**3. LDR sensoru vērtības (līniju grafiks)**

- Attēlo kreisā un labā LDR sensora vērtības laika gaitā.
- Dati tiek izlīdzināti ar vienkāršu `delta` koeficientu (`SMOOTHING_DELTA`), lai iegūtu gludāku līkni.
- Rāda līdz **100** pēdējiem punktiem (`MAX_POINTS`).

**4. Sistēmas stāvoklis (teksta logs)**

- Vizualizē, kādā stāvoklī šobrīd atrodas sistēma (piemēram, `SYSTEM_ON`, `SYSTEM_PAUSED`, utt.).
- Teksts centrēts, ar palielinātu fontu ērtai nolasīšanai.

**Datu iegūšana**

### `fetch_latest_data()`

- Veic SQL pieprasījumu uz `sun_tracking_data` tabulu.
- Iegūst jaunākos ierakstus (pēc `timestamp`), apgriež to kārtību hronoloģiskai attēlošanai.
- Atgriež rezultātus kā sarakstu ar vārdotiem lauciņiem (`dictionary=True`).

**Grafiku atjaunošana**

### `update_plot(frame)`

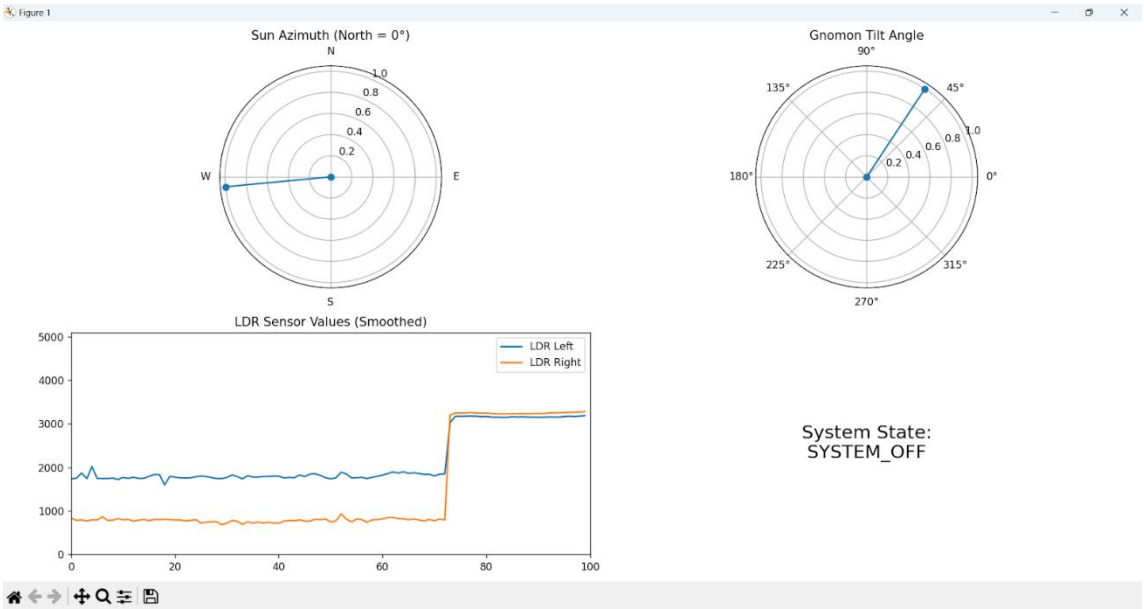
- Tiek izsaukta ik sekundi, izmantojot `matplotlib.animation.FuncAnimation`.
- Atjauno visus četrus vizualizācijas paneļus ar jaunākajiem datiem no datubāzes.
- Veic LDR datu izlīdzināšanu, pārkonvertē leņķus uz radiāniem polārajiem grafikiem.

**Kopsavilkums**

*Sundial_Plots.py* ir **lietotāja interfeiss** sistēmas uzraudzībai. Tas piedāvā vizuālu un intuitīvu skatījumu uz sensoriem, mehānismiem un saules pozīciju, un ļauj nekavējoties redzēt, kā sistēma reaģē uz vidi un ievaddatiem.

**Koda pielikumu saraksts:**

- **Pielikums A** – *main.pdf (css)*
  ESP32 mikrokontrollera programma: **Autonomous Sundial**
- **Pielikums B** – *ESPSerialSender.pdf (py)*
  Raspberry Pi skripts: **laika/atrašanās vietas sinhronizācija** un **datu uzglabāšana**
- **Pielikums C** – *Sundial_Plots.pdf (py)*
  Vizualizācijas rīks, kas lasa no **MySQL datubāzes** un **attēlo datus reāllaikā**




*Attēls 5, Vizualizācija Sundial_plots.py*

# Testēšana

| Nr. | Testēšanas elements | Testēšanas mērķis | Rezultāts |
|---|---|---|---|
| 1 | Sistēmas barošana | Pārbaudīt, vai ESP32 un visi komponenti saņem atbilstošu barošanu | Sistēma ieslēdzas pareizi un LED indikatori darbojas |
| 2 | Stepper draivera strāvas iestatījums | Optimizēt motora strāvu, lai novērstu pārkaršanu vai zemu griezes momentu | Iestatīts uz 0.3 V ref, motors darbojas stabili |
| 3 | Pogas (Start, Stop, Reset, Emergency) | Pārbaudīt pogu darbību un sistēmas reakciju uz tām | Sistēma korekti pārslēdz stāvokļus, Emergency aptur visu darbību |
| 4 | Limit switch | Pārbaudīt nulles pozīcijas noteikšanu un aizsardzību | Limit switch nostrādā un sistēma korekti nolasa pozīciju |
| 5 | Gnomona motors | Pārbaudīt motora darbību un virziena kontroli | Motors kustas pareizi un tiek kontrolēts caur stepper draiveri |
| 6 | Bāzes motors | Pārbaudīt rotācijas kustību un pozicionēšanu | Motors rotē 360°, tiek veikta korekta pozicionēšana |
| 7 | AS5600 sensori | Pārbaudīt sensora datus un to reakciju uz kustību | Abi sensori strādā bez problēmām |
| 8 | LDR sensori | Pārbaudīt saules gaismas intensitātes nolasīšanu un reakciju uz to | Viens LDR bojāts — nomainīts; pēc tam abi darbojas korekti |
| 9 | Gaismas izsekošanas algoritms | Pārbaudīt, vai tiek izsekots pareizais gaismas avots | Sākotnēji izsekoja nepareizus avotus; tika ieviests algoritms ar spilgtuma pārbaudi un ilguma filtru |
| 10 | Motora vadības algoritmu testēšana | Pārbaudīt, vai kustības tiek pareizi aprēķinātas un veiktas (stepper vadība, leņķu kompensācija, nulles pozīcija) | Motori kustas paredzētajā virzienā un ātrumā, tiek ievērota tolerances zona un tiek veikta korekta pozicionēšana |
| 11 | Datu nosūtīšana uz Raspberry Pi | Pārbaudīt, vai ESP32 pareizi nosūta telemetrijas datus uz RPi | Seriālā komunikācija strādā, dati tiek nosūtīti ik sekundi |
| 12 | SQL datubāze | Pārbaudīt, vai dati tiek ierakstīti datubāzē | Veiksmīgi tiek glabāti dati par pozīcijām, gaismas intensitāti, sistēmas statusu |
| 13 | Datu vizualizācija Python | Reālā laikā attēlot azimutu, lokācijas platuma leņķi, LDR vērtības un sistēmas statusu | Grafiki tiek atjaunoti ik sekundi, rādījumi ir vizuāli korekti |
| 14 | Gaismas avota simulācija ar datora ekrānu | Sākotnējā testēšana bez RS-35 luktura | Sistēma reaģēja uz ekrāna gaismu, bet netika sasniegta pietiekama spilgtuma uzticamība |
| 15 | Mehāniskā bāzes savienojuma izturība | Pārbaudīt, vai motors reāli pagriež pulksteni | Bāzes savienojums sākumā slīdēja; tika nostiprināts mehāniski ar līmi |
| 16 | Sistēmas gala testēšana | Pārbaudīt visu sistēmu kā vienotu bloku – no datu saņemšanas līdz vizualizācijai un reakcijai uz gaismu | Sistēma darbojas kā paredzēts, tiek veikta sinhrona datu apmaiņa, kustības, vizualizācija un datu uzglabāšana |