Otso Pudas

Selvitys Controllino-mikrokontrollerin käytöstä ja kasvatuskaappiprojektin toteutus

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Sähkö- ja automaatiotekniikka

Insinöörityö

31.10.2022

Tiivistelmä

Tekijä: Otso Pudas

Otsikko: Selvitys Controllino-mikrokontrollerin käytöstä ja kasvatuskaappiprojektin toteutus

Sivumäärä: 45 sivua + 1 liite
Aika: 31.10.2022

Tutkinto: Insinööri (AMK)

Tutkinto-ohjelma: Sähkö- ja automaatiotekniikka

Ammatillinen pääaine: Automaatiotekniikka
Ohjaajat: Lehtori Timo Tuominen
Projektipäällikkö Tiina Vuorijärvi

Insinöörityön tilaaja on Big Flash-hanke ja se toteutettiin Little Garden -yritykselle. Työn tavoitteena on tuottaa prototyyppi itsenäisestä kasvatuskaapista ja luoda selvitys projektissa käytetyn Controllino-mikrokontrollerin käytöstä ja ominaisuuksista.

Controllino on Arduino Mega2560-mikroprosessoriin pohjautuva mikrokontrolleri. Se on luotu olemaan välimuoto harrastetason Arduino-mikrokontrollereiden ja teollisten PLC-tietokoneiden välillä. Suurin haaste Controllinon käytössä on siihen liittyvien ohjeiden vähäinen määrä, joka vaikeuttaa sen tehokasta käyttöä projekteissa. Insinöörityö pyrkii helpottamaan Controllinon käyttöä ja esittelemään sen ominaisuuksia.

Projektissa luotu kasvatuskaappi on prototyyppi kasvatusyksiköstä, joka mahdollistaa erilaisten kasvien, kuten yrttien tai itujen kasvatuksen myös siihen soveltumattomassa ympäristössä. Prototyypin tarkoituksena on olla kompakti, hiljainen, pystyä ohjaamaan sekä ylläpitämään sisäistä ilmastoaan erilaisissa olosuhteissa. Insinöörityössä käydään läpi kasvatuskaapin komponentit, kokoaminen ja ohjelmointi.

Insinöörityön tuloksena on kattava selvitys Controllino-mikrokontrollerin ominaisuuksista ja käytöstä, minkä tarkoituksena on helpottaa kontrollerin kanssa työskentelyä. Työssä luotu prototyyppi toimii lähtökohtana Little Gardenin tulevaisuuden bisnesmallille ja ohjaa alaa omaksumaan ilmastotietoisempia ratkaisuja tulevaisuudessa.

Avainsanat: Arduino, mikrokontrolleri, ohjelmointi, C++

Abstract

Author:	Otso Pudas
Title:	Use of the Controllino Microcontroller and the Implementation of the Growing Cabinet Project
Number of Pages:	45 pages + 1 appendix
Date:	31 October 2022
Degree:	Bachelor of Engineering
Degree Programme:	electrical and automation engineering
Professional Major:	Automation technology
Supervisors:	Tiina Vuorijärvi, Project Manager Timo Tuominen, Senior Lecturer

---

This thesis work was done in cooperation with Big Flash-enterprise and was carried out for company called Little Garden. The goal of the project was to make a prototype of an independently operating plant growing cabinet and to produce a comprehensive explanation of the use and characteristics of the Controllino-microcontroller that was used in the project.

Controllino is a microcontroller based on the Arduino Mega2560 microprocessor. It was created to fill the market between hobbyist microcontrollers and industry-level PLC computers. The biggest challenge in using Controllino is the small number of related instructions, which makes it difficult to use it effectively in projects. The thesis work aims to ease the use of Controllino and to present its features more clearly.

The growing cabinet build int the project is a prototype of a growing unit capable of growing different plants like herbs and sprouts in environments normally unsuitable for plant growth. The cabinet is built to be compact, quiet and able to control and maintain its internal climate in different conditions. The thesis work reviews the components, programming and assembly of the growing cabinet.

The result of this thesis work is a comprehensive information-package on the use and characteristics of the Controllino-microcontroller with the aim to make future use of the controller easier for other projects. The growing cabinet prototype is to be a starting point for Little Gardens new business model and a steppingstone towards more climate conscious future.

Keywords: Arduino, microcontroller, programming, C++

Sisällys

Liitteet

Liite 1: Laitteen ohjelma

Lyhenteet

Arduino: Arduino viittaa samannimisen yrityksen valmistamiin yhden piirilevyn mikrokontrollereihin.

RH/T: Suhteellinen ilmankosteus jaettuna lämpötilalla; laskutoimitus antaa absoluuttisen ilmankosteuden.

PLC: Ohjelmoitava mikrokontrolleri (Programmable logic controller) on teollisuuden käyttöön suunniteltu tietokone, jolla ohjataan tuotantoprosesseja, -linjoja tai robotiikkaa.

Open Source: Avoimen lähdekoodin teknologiat ovat avoimesti ja julkisesti kehitettyjä ja saatavilla olevia teknologioita, joita voidaan käyttää vapaasti ilman lisenssiä [1].

I/O moduuli: I/O moduuli, eli tulo/lähtömoduuli on laite, joka mahdollistaa siihen liitetyn logiikan lähettää ja vastaanottaa signaaleja.

SELV: Sähköjärjestelmä, jossa jännite ei voi ylittää pienoisjännitettä (ELV) normaaliolosuhteissa ja yhden vian tapauksessa, mukaan luettuna maasulut toisissa piireissä [2].

LAN: Lähiverkko on rajoitetun alueen laitteiden muodostama tietoliikenneverkko. Esimerkiksi samaan reitittimeen liitetyt laitteet muodostavat lähiverkon.

RTC: Real-time clock eli reaaliaikakello on sähköpiiri, joka mittaa aikaa mittaamalla sen sisältämän kristallin aiheuttamaa oskillaatiota.

Ethernet: Ethernet on pakettipohjainen tiedonsiirtoprotokolla.

PWM: Pulssinleveysmodulaatio on modulointitapa, jossa signaalin jännitettä säädellään säätämällä sen pulssileveyttä.

A/D: Analog to digital -piiri muuttaa saamansa analogisen signaalin digitaaliseksi.

GND: Sähkön maadoituspisteestä käytettävä lyhenne.

RPM: Pyörimisnopeuden mittayksikkö, joka kertoo pyörähdysten määrän minuutissa.

TFT LCD: Ohutfilmitransistori nestekristallinäyttö. Ohutfilmitransistorit sallivat oman transistorin jokaiselle näytön pikselille.

1 Johdanto

Tämän opinnäytetyön tavoitteena on toteuttaa projekti Big Flash-hankkeelle, jossa rakennetaan prototyyppi kasvatuskaapista sekä luodaan kattava selvitys työssä käytetystä Controllino-mikrokontrollerista. Projekti toteutetaan Little Garden -yritykselle, missä se toimii pilottina uudelle liiketoimintamallille. Kasvatuskaappi on suljettu siirrettävä tila, joka pystyy itsenäisesti kasvattamaan erilaisia kasveja säätämällä sisäisiä olosuhteita. Anturit mittaavat kaapin lämpötilaa sekä ilmankosteutta kaapin sisällä ja ulkopuolella, ja mikrokontrolleri ohjaa ilmastointia niiden mukaisesti. Kasvit saavat ravinteita veden mukana, jota pumppu kierrättää kasvien ja vesisäiliön välillä, jolloin vesi, jota kasvit eivät käytä, voidaan käyttää myöhemmin uudelleen.

Kasvatuskaapin konsepti on olla pieni ja tyylikäs kasvatusratkaisu, joka voidaan sijoittaa kauppoihin tai muihin julkisiin tiloihin ilman muutoksia infrastruktuuriin. Kaappi mahdollistaa ruokakasvien loppukasvattamisen kaupassa, jossa ne voidaan myydä suoraan kuluttajalle. Tämä pidentää kasvien hyllyaikaa huomattavasti.

Työssä käytetty Controllino on Arduino-pohjainen mikrokontrolleri, joka on tarkoitettu korvaamaan tai simuloimaan PLC-tietokonetta. Kontrollerin etuja on hinta, helppokäyttöisyys ja avoimen käyttöoikeuden mukainen suunnittelu. Tämän työn tarkoituksena on avata Controllinon käyttöä ja ohjelmointia tuleville tuotteen käyttäjille.

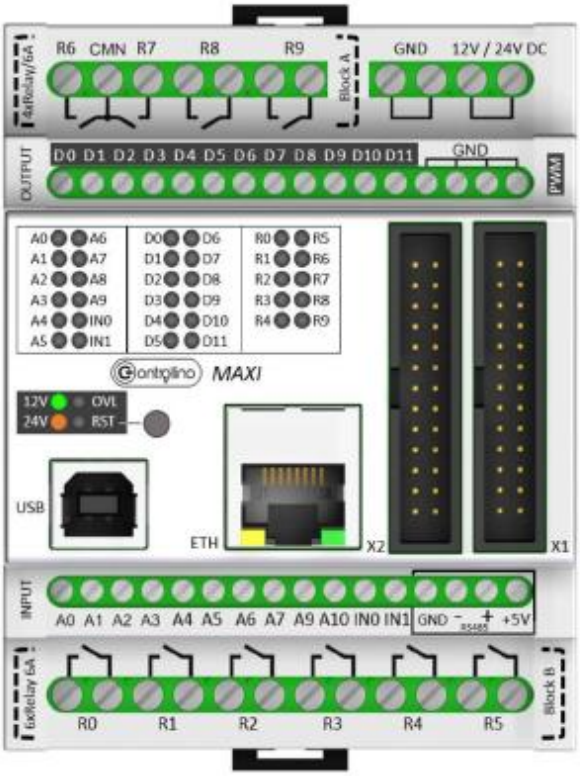
2 Controllino

Controllino PLC on open source Arduino-alustaan perustuva pienikokoinen tietokone. Se on välimuoto Arduino-pohjaisten harrastetason mikrokontrollerien ja teollisuuden käyttämien PLC-ohjainten välillä. Tämä on toteutettu yhdistämällä Arduino-mikrokontrolleri teollisuudelle tyypilliseen I/O-mooduuliin. Näin Controllino mahdollistaa nopean ja notkean ohjelmoinnin ja käytön, ja samalla mahdollistaen liitännän ja käytön teollisuudelle tyypillisissä sovelluksissa. Tässä luvussa keskitytään Controllino Maxi -ohjaimeen. Tästä eteenpäin käytetään termejä controllino, ohjain tai mikrokontrolleri ja niillä viitataan Controllino Maxi -ohjaimeen.

2.1 Rakenne ja tekniset tiedot

Controllinossa on 11 digitaalista ulostuloa, 10 analogista sisääntuloa, kaksi interrupt-terminaalia, 10 Relettä, 12 ja 24 voltin virran sisääntulo, neljä maaterminaalia, RS485-sarjakommunikaatioterminaalit, USB-portti ja Ethernet-portti. Lisäksi ohjaimessa on suorat pinnit Arduino Mega2560-mikroprosessessoriin terminaaleissa X1 ja X2 (kuva 12).

Ohjaimessa voidaan käyttää joko 12 tai 24 voltin käyttöjännitettä, ja eri terminaalien maksimijännite määräytyy käytetyn jännitteen mukaan. Ohjaimessa on kaksi sulaketta, yksi 20 ampeerin sulake ulos- ja sisääntuloille sekä 10 ampeerin sulake jokaisessa releessä [3].



Kuva 1. Controllino Maxi-mikrokontrolleri [3].

Digitaaliset lähdöt ja releet

Ohjaimen lähtö tarkoittaa terminaalia tai pinniä, joka lähettää signaaleja toisille laitteille. Usein mikrokontrollereissa on erikoistuneita lähtöjä, jotka pystyvät lähettämään erilaisia signaaleja, tai kestävät suuremman kuorman niihin liitetyistä laitteista.

Controllinon lähdöt ovat digitaalisia, eli ne lähettävät signaaleja tietyiltä jännitevälеiltä, jotka vastaanottava laite tulkitsee signaaleiksi 0 tai 1, mutta ne pystyvät emuloimaan analogisia signaaleja, eli tietyn jännitealueen sijaan lähtö pystyy lähettämään kaikki arvot tiettyjen ääripäiden väliltä. Näin ne pystyvät PWM-ohjaukseen sekä ajamaan pieniä tasasähkömoottoreita omalla virrallaan.

Jokainen ohjaimen lähdöistä kestää kahden ampeerin kuorman joko 12 tai 24 voltin jännitteellä, ohjaimeen kytketystä käyttöjännitteestä riippuen. Lähtöjen yhteinen kuorma ei kuitenkaan saa ylittää 6 ampeeria, eli jos kaikki lähdöt ovat käytössä, on niiden maksimaalinen kuorma 0,5 A per lähtö [3]. Ulostulot eivät ole myöskään potentiaalivapaita, vaan niiden antama signaali pitää kytkeä maahan. Tiettyjä lähtöjä on mahdollista rinnastaa tietyissä olosuhteissa, jos on tarve ajaa suurempaa kuormaa.

Controllinon releet toimivat sekä tasa- että vaihtovirralla. Niiden nimellinen maksimikuormitus on 6 ampeeria, mutta ne kestävät hetkellisesti korkeamman kuormituksen 10 ampeeriin asti. Releissä on 250V 10A pääpiiristä ulkoinen sulake. Maksimijännite on vaihtovirtaa 230V +10 % ja tasavirtaa 30V [3]. Releet on jaettu kahteen lohkoon, koska niitä voidaan käyttää SELV-järjestelmissä. Samaa lohkoa ei saa käyttää samaan aikaan normaaleille ja EVL-piireille.

Analogiset ja interrupt-tulot

Ohjaimen tulo tarkoittaa terminaalia tai pinniä, joka vastaanottaa toisista laitteista tulevia signaaleja. Tulot voivat olla digitaalisia tai analogisia. Controllinon tulot ovat analogisia, mutta niihin tulevat signaalit muutetaan digitaalisiksi ohjaimen sisällä A/D-muuntopiirissä. Muunnettujen arvojen resoluutio on 10 bittiä, eli arvoja välillä 0–1023. Yhtä yksikköä vastaava jännite riippuu ohjaimen käyttöjännitteestä:

- 1 yksikkö = 0,015V (15 mV) jos käyttöjännite on 12 volttia.
- 1 yksikkö = 0,03V (30 mV) jos käyttöjännite on 24 volttia.

Jokaista Controllinon tuloa voidaan käyttää myös digitaalisena tulona. Tällöin logiikan jännitevälit ovat riippuvaisia käyttöjännitteestä (taulukko 1).

Taulukko 1. Digitaalisten tulojen Loogiset jännitevälit [3].

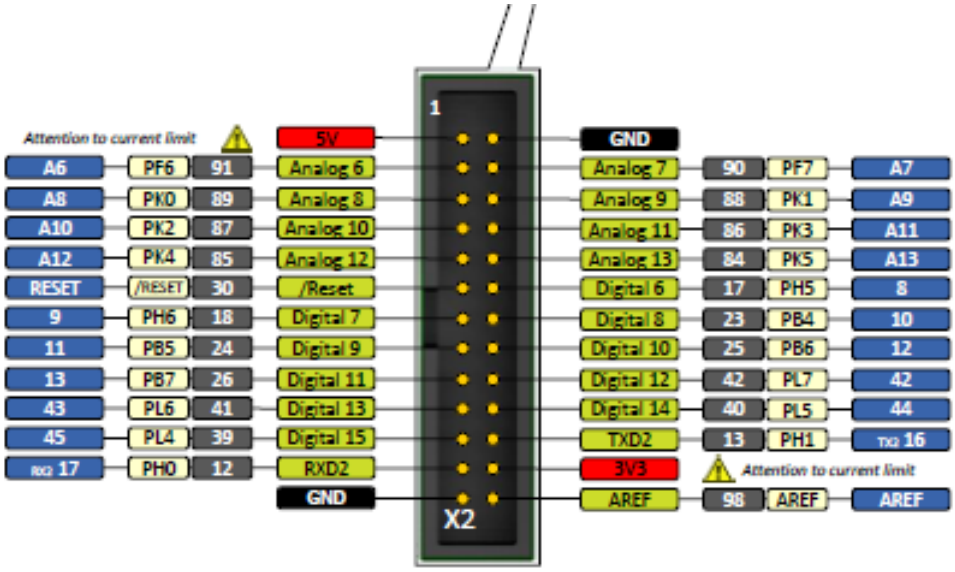
Looginen arvo	Käyttöjännite	Jänniteväli
0	12V	0–3,6V
1	12V	9–13,2V
0	24V	0–7,2V
1	24V	18–26,4V

Interrupt-tulot ovat terminaaleja, jotka voivat ottaa vastaan äkillistä huomiota vaativia signaaleja. Ne käyttävät omaa ohjelmoitua rutiinia, joka saadessaan signaalin ajaa sen muiden yli. Käytännössä näillä porteilla voidaan ottaa vastaan signaaleja, jotka voidaan käsitellä erillään pääohjelmistosta, ja ohjain pystyy vastaanottamaan näitä signaaleja milloin vain. Hyviä esimerkkejä ovat erilaiset turvapiirit, kuten hätäseis-nappi. Jos se kytketään normaaliin tuloon, signaali luetaan vasta, kun ohjelma on käynyt itsensä kerran läpi, jolloin voi olla liian myöhäistä. Jos signaali tulee Interrupt-tuloon, huomaa ohjelma sen heti [4].

X1- ja X2-pinnit

Controllinon terminaalit X1 ja X2 ovat suorat pinnit ohjaimen prosessoriin. Niiden tarkoituksena on sallia pinnien käyttö samalla tavalla kuin Arduino Mega2560-mikrokontrolleria käyttäessä. Toisin kuin Controllinon muut tulot ja lähdöt, X1- ja X2-pinnit toimivat viiden voltin jännitteellä ja niissä on oma maa (kuva 13 ja 14). Pinnit eivät ole erillisiä ohjaimen muista terminaaleista, vaan sama terminaali yhdistyy prosessoriin samaan paikkaan kuin vastaava pinni. Ainoastaan jännite on eri. Pinnit on suojattu sähköstaattisilta purkauksilta [3].

Kuva 2. Controllino X1 pinnien nimet [5].



Kuva 3.Controllino X2 pinnien nimet [5].

USB, Ethernet ja sarjakommunikaatio

Controllino on mahdollista liittää osaksi tietoverkkoa usealla eri tavalla. Tyypillisin on yhdistää ohjain USB-johdolla tietokoneeseen. Tämä mahdollistaa ohjaimen ohjelmoimisen ja tiedon siirron ohjaimelta tietokoneelle.

Controllinossa on Ethernet-portti, jolla Controllino voidaan liittää osaksi LAN-verkkoa. Se voi kommunikoida suoraan muiden samassa verkossa olevien laitteiden kanssa.

Ohjaimen RS485 terminaalit ovat sarjakommunikaatiota varten. RS485 on käyttöliittymästandardi sarjakommunikaatiolle, joka mahdollistaa ohjaimen liittämisen korkeintaan 32 muun laitteen kanssa.

RTC-piiri ja OVL-ledi

OVL- ledi ja lähtö PE7 (sijaitsee laitteen kotelon sisällä) on liitetty laitteen sisällä oleviin lämpötila-antureihin. Ne antavat signaalin, kun Controllinon sisäinen lämpötila ylittää sallitun raja-arvon.

Ohjaimessa on sisäinen RCT-piiri, jolla ohjainta voidaan aikaohjata ilman ulkoista kelloa tai internetyhteyttä. Piirissä on oma akku, jonka avulla se pitää asetetun ajan 2 viikkoa ilman ulkoista virtaa.

2.2 Controllinon ohjelmointi ja käyttö

Controllino ohjelmoidaan käyttämällä Arduino IDE -ohjelmistoa. Se sallii ohjelman kirjoittamisen, koostamisen ja lataamisen Arduino-ympäristöä käyttäviin ohjaimiin. Controllinon ominaisuuksien käyttö vaatii oman koodikirjaston käyttöä, jonka komentoja ja ominaisuuksia käydään tässä luvussa läpi.

Controllinon yksinkertaisin komento on tulon tai lähdön määrittäminen. Toisin kuin normaalia Arduinoa käytettäessä, jolloin voidaan vain käyttää pinnin numeroa, Controllinossa pelkkä numero määrittää käyttöön numeroa vastaavan pinnin X1- tai X2-terminaalista. Ruuviterminaali määritellään koodissa lisäämällä terminaalin nimen eteen komento, joka kertoo määritettävän terminaalin olevan Controllinon ruuviterminaali (esimerkkikoodi 1).

```
#define CONTROLLINO_D0
```

Esimerkkikoodi 1. Controllino-ruuviterminaalin D0 määrittäminen [6].

Riippuen siitä onko kyseessä tulo vai lähtö, joudutaan portit määrittelemään eri tavalla. Portin käytöstä riippuen on siihen eri määrittelyjä. Esimerkiksi kun halutaan lähdön lähettävän PWM-signaalia, määritellään se kuten esimerkkikoodissa 2.

```
#define CONTROLLINO_SCREW_TERMINAL_PWM_00
```

Esimerkkikoodi 2. Controllino-ruuviterminaali 0 määrittäminen lähettämään PWM-signaalia [6].

Esimerkkikoodi 2 määrittää, että kyseessä on ruuviterminaali, joka lähettää PWM-signaalia. Sama mikrokontrollerin pinni voitaisiin määritellä lähettämään PWM-signaalia suoraan X1-terminaalissa olevasta vastaavasta pinnistä. Tämä on tärkeää ohjelmoidessa, sillä pinnin määrittely myös muuttaa sen antamaa jännitettä. Jos pinni on määritelty koodissa väärin, voi siihen liitetty laite vahingoittua saadessaan liikaa virtaa tai liian suuren jännitteen. Varsinkin erilaiset mittauslaitteet ja anturit ovat herkkiä ylivirralle.

Controllinon pinnit voidaan koodissa asettaa joko lähdöiksi tai tuloiksi. Kaikkia pinnejä ei voida vaihtaa tulon ja lähdön välillä. Pinni määritellään koodissa setup loopissa (esimerkkikoodi 3).

```
const int fan_control_pin1 = CONTROLLINO_PIN_HEADER_PWM_07;

void setup() {
     pinMode( fan_control_pin1, OUTPUT);
}
```

Esimerkkikoodi 3. Määritetään pinni 7 lähettämään PWM signaalia ja määritetään se lähdöksi.

Muita Controllinon ominaisuuksia, joita tulee määrittää koodissa, ovat RTC-piiri, Ethernet-portti ja siihen liitetty tietoliikennepiiri, RS485-käyttöliittymä ja releet.

Releet toimivat samalla tavalla koodissa kuin lähdöt, mutta signaalin lähettämisen sijaan ne menevät kiinni tai auki. Jos looginen signaali on 0, on rele auki, ja kun se on 1, on rele kiinni (taulukko 1). Releille on kaksi määrittelyä koodissa (esimerkkikoodi 4).

```
#define CONTROLLINO_RELAY_00

#define CONTROLLINO_R0
```

Esimerkkikoodi 4. Ylhäällä määritetään rele 0 toimimaan releenä ja alhaalla määritellään rele 0 toimimaan digitaalisena lähtönä. Molemmat komennot toimivat käytännössä samalla tavalla koodissa [6].

RTC-piiri tulee ottaa käyttöön ja määrittää aika setup loopissa (esimerkkikoodi 5).

```
void setup() {

Controllino_RTC_init(0); //initialize Real Time Clock module on Con-
trollino
  Controllino_SetTimeDate(5,1,9,22,14,19,50);// (Day of the month, Day
of the week, Month, Year, Hour, Minute, Second) //Set date
}
```

Esimerkkikoodi 5. Aktivoidaan RTC-piiri ja asetetaan siihen aika.

RCT-piiriä käytettään ajastamaan toimintoja. Sen tarkoitus on kertoa prosessorille siihen asetettu aika ja ilman sitä prosessien nopeuksia tai ajastuksia on hyvin vaikeaa rinnastaa 24 tunnin sykliin. Projektissa ajastusta tarvitaan vesipumpun ja valojen säätämiseen oikeina kellonaikoina (esimerkkikoodi 6). Mikrokontrollerin ajastaminen ilman RCT-piiriä tapahtuu liittämällä ohjain verkkoon ja lukemalla ajan tietokonepalvelimelta.

```
if(currentHour==9&&currentMin>=1&&currentMin<=3){
            analogWrite(pumpPin, 255);
            Pumppu = 1; //set pump status to 1 for printing
        } else{
          analogWrite(pumpPin, 0);
          Pumppu = 0;       //set pump status to 0 for printing
        }
           if(currentHour>=5&&currentHour<=8){
            analogWrite(lightPin, 0);
            Valo = 0;          //set light status to 0 for printing
        } else{
          analogWrite(lightPin, 255);
          Valo = 1; //set light status to 1 for printing

        }
```

Esimerkkikoodi 6. Ajastetaan pumppu päälle kolmeksi minuutiksi klo 9.01 ja valo pois päältä neljäksi tunniksi klo 5.00–9.00

3 Komponentit

3.1 Virtalähde

Työssä käytetään MEAN WELL NDR-120-sarjan 10A yhden ulostulovirran virtalähdettä. NDR-120-12 on teollinen DIN-kiskoon liitettävä virtalähde, joka muutta sähköverkosta saatavan kolmivaihevirran kahdentoista voltin tasavirraksi. Siinä on yksi sisääntulovirta ja yksi ulostulovirta. Se on tarkoitettu käytettäväksi teollisissa prosenssinhallinta- ja automaatiojärjestelmissä [7] .

NDR-120 sarjan virtalähde valittiin työhön, koska se täyttää kolme vaadittavaa kriteeriä: mahdollisuus liittää DIN-kiskoon, 12 voltin jännite ja yli 100 wattia tehoa. Tärkeää oli myös tarpeeksi pieni hinta, jottei se ylitä varattua budjettia. NDR-120 valikoitui myös saatavuuden perusteella, sillä se oli ainut kriteerit täyttävä virtalähde Partco-verkkokaupassa, joka toimii Big Flash-hankkeen toimittajana.

Kuva 4. MEAN WELL NDR-120-sarjan virtalähde

NDR-120-virtalähteen voisi korvata millä tahansa virtalähteellä, joka täyttää kriteerit. NDR-120:n huonoja puolia on sen koko, joka tässä työssä rajoitti suojakaapin valintaa. Muita hyödyllisiä lisäominaisuuksia olisivat useammat ulostulojännnitteet.

3.2 Suojakotelo

Suojakotelo elektronisille laitteille työssä on FIBOX-yrityksen EKPK 230 T. Se on polykarbonaatista valmistettu suojakotelo läpinäkyvällä kannella. Kannen tiivisteet on valmistettu polyuretaanista. Kotelo täyttää tunkeutumissuojausluokituksen IP66 ja iskunkestävyysluokituksen IK08. Kotelo on myös suojaeristetty sähköltä [8].

Kuva 5. EKPK 230 T -suojakotelo

EKPK 230 T valittiin työhön, koska siinä on tarvittava suojaus vedeltä. Siinä on läpinäkyvä kansi, joka mahdollisti näytön asentamisen kotelon sisään, se on tarpeeksi suuri kaikille asennettaville komponenteille ja koteloon on mahdollista asentaa DIN-kiskoja komponenttien asennukseen.

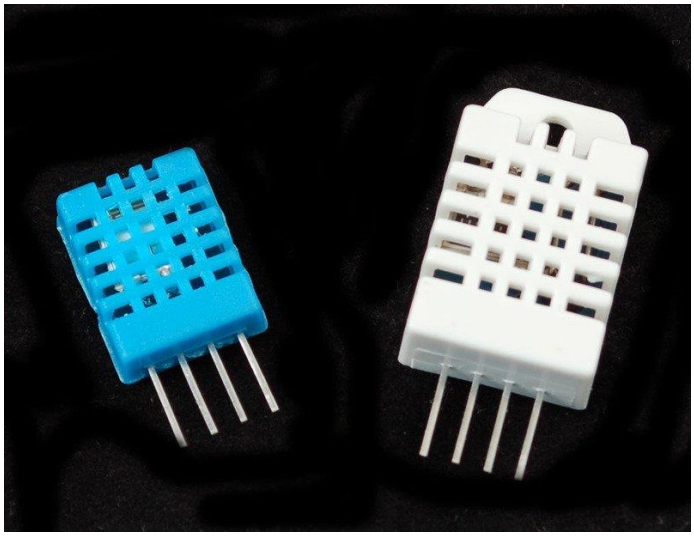
EKPK 230 T -suojakotelolle ei ole juurikaan vaihtoehtoisia tuotteita. Suurin osa asennuskoteloista on tarkoitettu komponenteille kuten ylivirtasuojille, eikä niissä ole tilaa työn vaatimille komponenteille. Suuri liitettävien komponenttien määrä vaati myös suuren määrän riviliittimiä, jotka eivät olisi mahtuneet kilpaileviin tuotteisiin. Muissa tuotteissa ei myöskään ollut usein vaihtoehtoa läpinäkyvälle kannelle.

3.3 Anturit

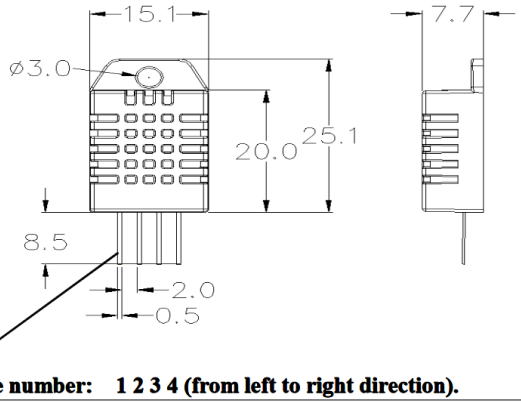
DHT22

Työssä käytetään neljää DHT22-sarjan lämpötila- ja kosteusanturia. DHT22 on halpa harrastustason anturi, joka koostuu kapasitiivisestä kosteusanturista, termistorista sekä yksinkertaisesta piirilevystä, joka kääntää analogiset signaalit digitaalisiksi. Anturi lähettää kaiken mittausdatan yhden johtimen kautta kahden sekunnin välein. Anturi mittaa ilmankosteutta 100 prosenttiin asti 2- 5 % virhemarginaalilla ja lämpötilaa välillä -40 - +80 celsiusastetta puolen prosenttiyksiön virhemarginaalilla. Mittaustarkkuus on ilmankosteudelle prosentin kymmenesosa ja lämpötilalle celsiusasteen kymmenesosa. Anturin käyttöjännite on 3.3–6 V DC [9].

Anturin hyviä puolia ovat sen matala hinta, yksinkertainen käyttö valmiiden ohjelmointikirjastojen avulla, yksinkertainen liittäminen yhdellä datajohtimella ja hyvä saatavuus. Huonoja puolia ovat huono mittaustarkkuus ja hidas mittausnopeus. Anturi valittiin projektiin, koska niitä oli hankkeen varastossa valmiina.



Kuva 6. DHT11 (vasen) ja DHT22 (oikea) lämpötila- ja ilmankosteusanturit [9].



Pin sequence number: 1 2 3 4 (from left to right direction).

Pin	Function
1	VDD----power supply
2	DATA--signal
3	NULL
4	GND

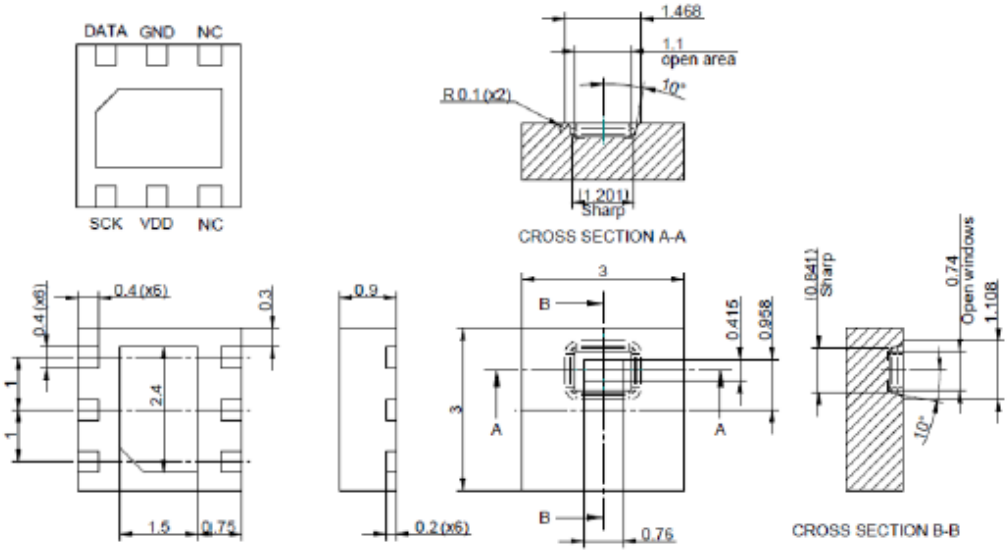
Kuva 7. DHT22-lämpötila- ja ilmankosteusanturin mitat ja pinout [10].

HTU21D

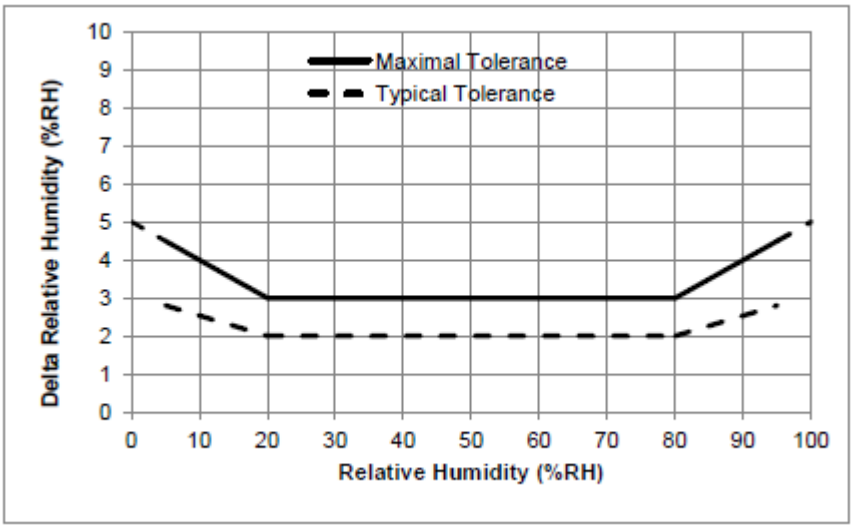
HTU21D on digitaalinen kosteus- ja lämpötila-anturi. Kuten DHT22, HTU21D on myös halpa, harrastetason anturi. HTU21D ei vaadi erillistä piirilevyä kääntämään analogisen signaalin digitaaliseksi vaan lähettää datan aina I2C-formaatilla. Anturin tarkkuus riippuu, millä resoluutiolla anturi mittaa dataa. Resoluutiota voidaan säätää välillä 8–14 bittiä per RH/T lähettämällä koodissa anturille komennon. Mitä suurempi mittausdatan bittiluku on, sitä tarkempi mittaustulos on ja hitaampi mittausaika. Anturin mittausväli ilmankosteudelle on 0–100 %, 2–5 % virheellä, jossa 2 % virhe on tyypillinen, mutta tarkkuus huononee, kun mitataan alle 20 % tai yli 80 % ilmankosteutta. Mittaustarkkuus on 12 bitin resoluutiolla prosentin neljän sadasosan tarkkuudella ja 8 bitin resoluutiolla seitsemän kymmenesosan tarkkuudella. Mittausnopeus ilmankosteudelle on 2–14 millisekuntia riippuen mittausresoluutiosta.

Lämpötilan mittausväli on -40–125 celsiusastetta 0,3–1,6 asteen virheellä, jossa tarkkuus huononee, kun mitattava lämpötila on yli 60 °C tai alle 2 °C. Mittaus-

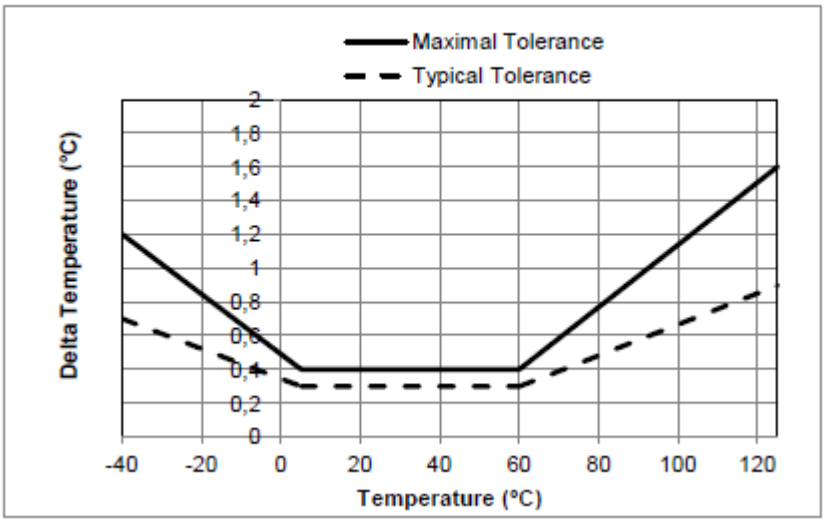
tarkkuus on asteen yksi sadasosa tai neljä sadasosaa, mikä riippuu valitusta resoluutiosta. Mittausaika on 6–50 millisekuntia riippuen resoluutiosta. Anturin käyttöjännite on 5 volttia [11].



Kuva 8. HTU21D-anturin mittasuhteet [11].



Kuva 9. Suhteellisen ilmankosteuden virhemarginaaliolosuhteet, 25 °C [11].



Kuva 10. Lämpötilan virhemarginaali [11].

HTU21D olisi hyvä vaihtoehto korvaamaan DHT22-anturit projektissa. Sen nopeampi mittausaika ja parempi mittaustarkkuus parantaisivat saatavaa dataa huomattavasti. Anturit ovat melkein samanhintaisia ja ominaisuuksiltaan vastaavia.

3.4 Ilmastointi

Projektin ilmastointi toimii kuuden tuulettimen avulla. Kolme matalan tehon tuuletinta ovat päällä jatkuvasti, millä huolehditaan, että kaapin sisällä oleva ilma vaihtuu. Kolme korkean tehon tuuletinta ovat päällä ainoastaan, jos kaapin sisäinen ilmankosteus nousee liian korkeaksi. Tuulettimet ovat tehokkaita poistamaan ylimääräistä ilmankosteutta, mutta ovat kovaäänisiä eikä ilmankosteuden poistamiselle ole tarvetta kuin tietyissä ympäristöissä.

P12 PWM PST

P12 on 120 millinen tuuletin PWM-säädöllä ja tachometrillä. Tuulettimen nopeus on 200–1800 rpm eli se pyörii hitaasti, vaikka saisi alimman mahdollisen säätöpulssin. Tuulettimen ilmavirtaus on vahvimmillaan 95,7 m^3/h ja staattinen ilmanpaine 2,20 mm H_2O. Käyttövirta tuulettimelle on 0,1 A ja jännite 12 V. Suurin äänenvoimakkuus on 28 dB [12].

P12 PWM PST valittiin projektiin sen saatavuuden, PWM-ohjauksen ja hiljaisen äänen perusteella. P12 on suunniteltu käytettäväksi tietokoneissa kotelotuulettimina ja voidaan olettaa, että se voitaisiin korvata melkein millä vain tietokoneen kotelotuulettimella, missä on PWM-säätö.

Kuva 11. P12 PWM PST -kotelotuuletin eri väreissä

NF-F12 industrialPPC-3000 PWM

Noctuan NF-F12 on korkean nopeuden tuuletin, joka on tarkoitettu teolliseen käyttöön. Sen nopeus on 750–3000 rpm ja lopettaa pyörimisen kokonaan, jos ei saa tarpeeksi korkeaa ohjauspulssia. Tuulettimen luoma ilmavirtaus on korkeimmillaan 186,7 m^3/h ja staattinen paine 7,63 mm H_2O. Käyttövirta on korkeimmillaan 0,3 A ja jännite on 12 V. Tuulettimen kovin luoma ääni on 43,5 dB [13].

NF-F12 valittiin projektiin korvaamaan ensin suunniteltua ilmanvaihtoa, missä yksi voimakas tuuletin olisi hoitanut ilmanvaihdon. Tästä luovuttiin muun muassa kustannus- ja sähköteknisistä syistä. NF-F12 pystyy kuitenkin tuottamaan tarvittavan ilmanpaineen ja -virtauksen alentamaan kaapin ilmankosteutta.

Kuva 12. Noctua NF-F12 industrialPPC-3000 PWM

3.5 Muut komponentit

Muut työssä käytetyt komponentit ovat:

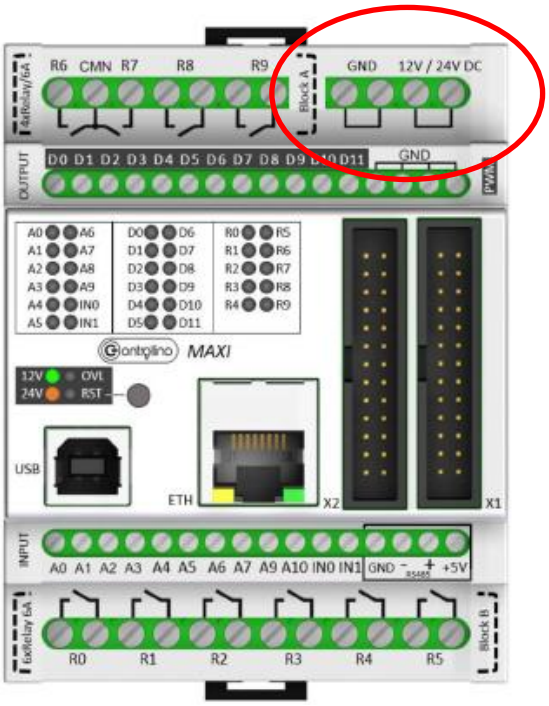
- Metallikauluksella varustettu palautuva painike sulkevalla toiminnalla.
- 1,44 tuuman TFT LCD -näyttö SPI käyttöliittymällä.
- 6 x Parus PFL-m 600mm:n valaisimet omalla virtalähteellään.
- SEAFLO 33 series DC -kalvopumppu.

Projektin tilannut yritys toimitti ja asensi käytettävät valot ja pumpun, eikä niiden hankinta ollut osa projektia. Painonapin ja näytön käyttöä käydään läpi samalla ohjelmoinnin kanssa.

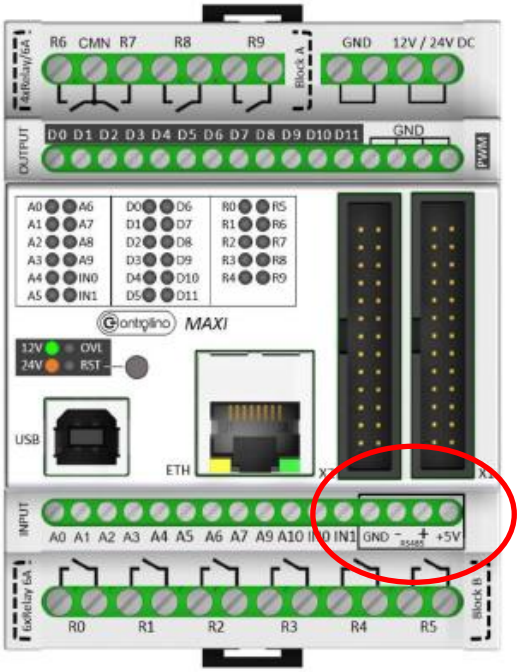
4 Projektin toteutus

4.1 Virtalähteen ja mikrokontrollerin yhdistäminen

Virtalähde ja Controllino asennettiin suojakotelon sisään (kuva 5) DIN-kiskoon riviliittimien kanssa. Riviliittimistä muodostettiin kolme yhtenäistä liitinryhmää, joista yksi liitettiin virtalähteen maa- terminaaliin ja toinen virtalähteen 12V-terminaaliin (kuva 4). Controllinon virtaliittimet (kuva 13) liitettiin samoihin riviliitinryhmiin. Kolmas liitinryhmä kytkettiin Controllinon RS485-käyttöliittymän virtaterminaaliin (kuva 14), josta saadaan 5V jännite. Kaikki projektin laitteet on kytketty, ja ne saavat virran riviliitinryhmistä. Näin varmistetaan, että kaikki laitteet on kytketty samaan maahan. Suojakoteloon asennettiin toinen DIN-kisko ja siihen tarpeeksi riviliittimiä laitteiden kytkemisen helpottamiseksi.



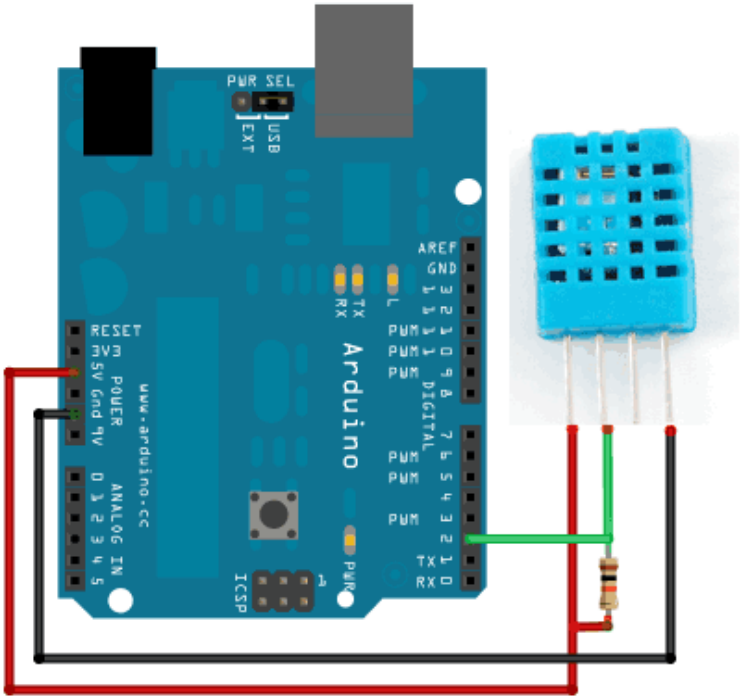
Kuva 13. Controllino-ohjaimen virtaliittimet



Kuva 14. Controllino-ohjaimen RS485-terminaali

4.2 Antureiden liittäminen

Antureihin liitettiin ruuviterminaalit kytkemisen helpottamiseksi. Ne liitettiin kotelon asennettuihin riviliittimiin. DHT22-antureissa on kolme liitettävää johdinta (kuva 7), joista maajohdin liitettiin yhteiseen maahan ja virtajohdin 5V liitinryhmään. Antureita on neljä ja niiden dataliittimet yhdistettiin Controllinon X1 ja X2 terminaalien analogisiin tuloihin A0, A1, A2 ja A7 (kuvat 2 ja 3).



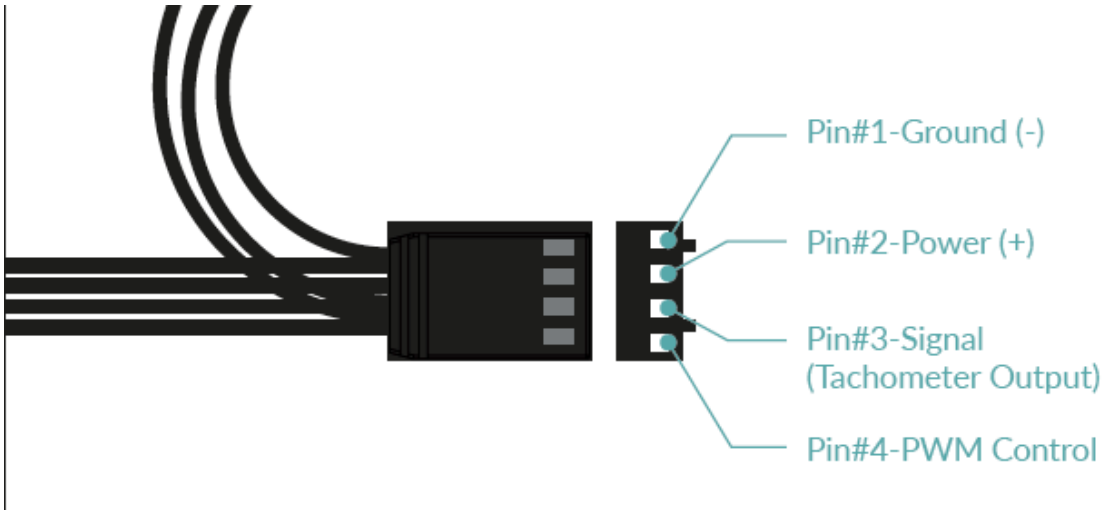
Kuva 15. DHT22-anturin kytkeminen Arduino-mikrokontrolleriin [14].

4.3 Ilmastoinnin liittäminen

Projektin kuusi tuuletinta liitettiin koteloon riviliittimiin. Jokaisessa tuulettimessa on neljä johdinta (kuva 16):

- GND-maajohdin
- 12V johdin
- Tachometrin datajohdin
- PWM ohjauksen datajohdin.

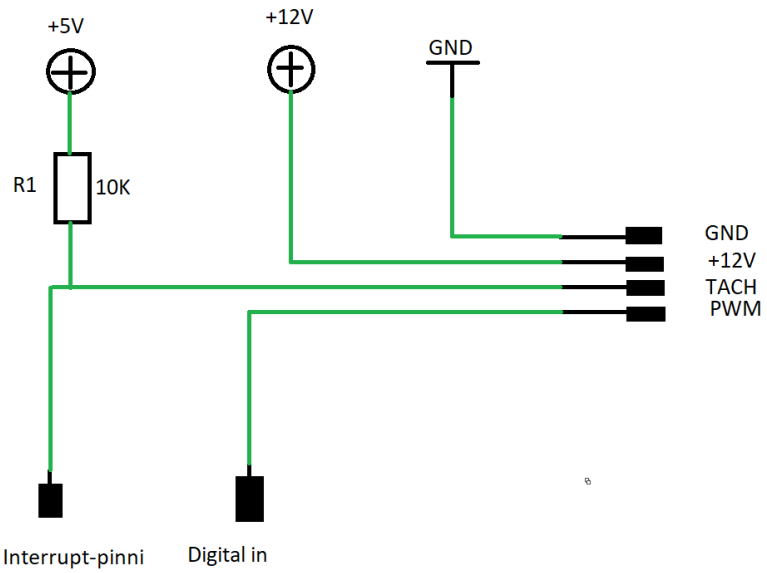
Kaikkien tuulettimien maa ja 12V johtimet liitettiin vastaaviin riviliitinryhmiin. PWM-johtimet liitettiin Controllinon digitaalisiin lähtöihin 2, 7, 8, 9, 10 ja 11 (kuvat 2 ja 3).



Kuva 16. Tuulettimen liittimen pinnien nimet

Ainoastaan kolmen tuulettimen tachometrit liitettiin rajallisten interrupt-pinnien määrän vuoksi. Tachometrit vaativat pullup-piirin toimiakseen, eli johdin pitää yhdistää sekä 10 kilo-ohmin vastuksella 5 voltin virtaan, että ohjaimen interrupt-pinniin (kuva 17). Pullup-piiri ”vetää” loogisen signaalin ylemmälle loogiselle jännitealueelle, kun anturi ei lähetä signaalia. Tämä estää ohjaimen pinnin ”kelluMisen” eli tilan, jossa ohjaimen saama signaali on loogisten alueiden välissä. Kelluminen voi antaa ohjaimelle vääriä signaaleja.

Tachometrit liitettiin riviliittimien kautta pinneihin interrupt 0 ja digitaalisiin lähtöihin 2 ja 3. Lähdöt muutettiin ohjelmassa tuloiksi.



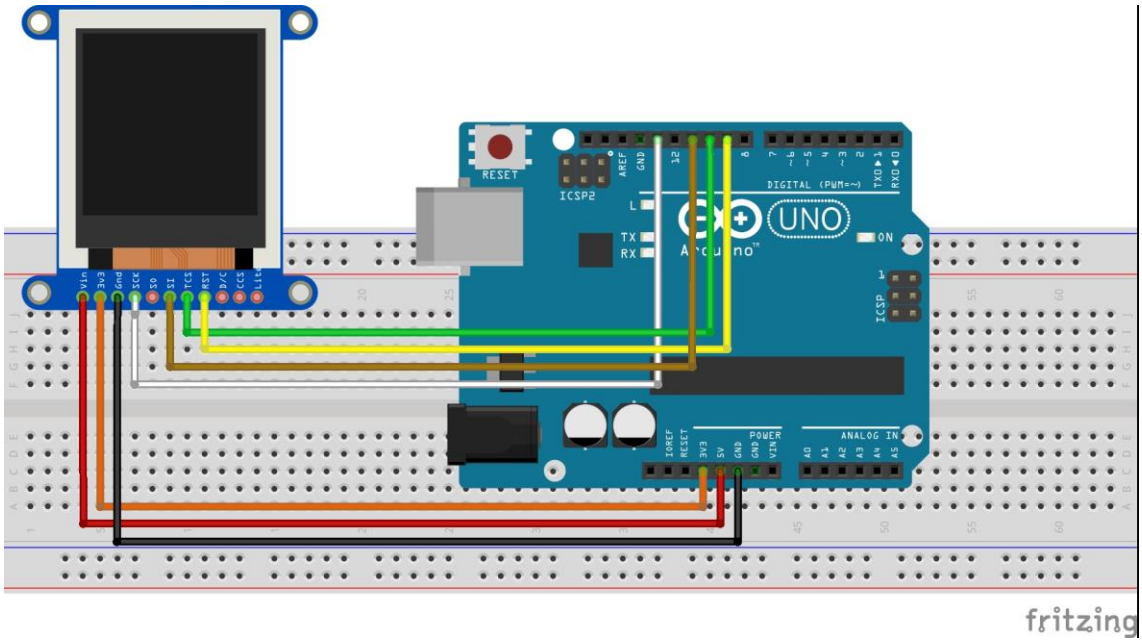
Kuva 17. Tuulettimen kytkentä

4.4 Näytön ja painonapin liittäminen

Painonappi on liitetty suoraan interrupt-pinniin 19 (kuva 2).

TFT LCD -näyttö on liitetty ohjaimeen kahdeksalla johtimella:

- VCC johdin on liitetty Controllino X1 terminaalin +5V pinniin.
- GND johdin on liitetty Controllino X1 terminaalin GND pinniin.
- CS johdin on liitetty Controllino X1 terminaalin 5 pinniin.
- RST johdin on liitetty Controllino X1 terminaalin 7 pinniin.
- A0 johdin on liitetty Controllino X1 terminaalin 6 pinniin.
- SDA johdin on liitetty Controllino X1 terminaalin 20 pinniin.
- SCK johdin on liitetty Controllino X1 terminaalin 52 pinniin.
- LED johdin on liitetty Controllino X1 terminaalin +3,3V pinniin.

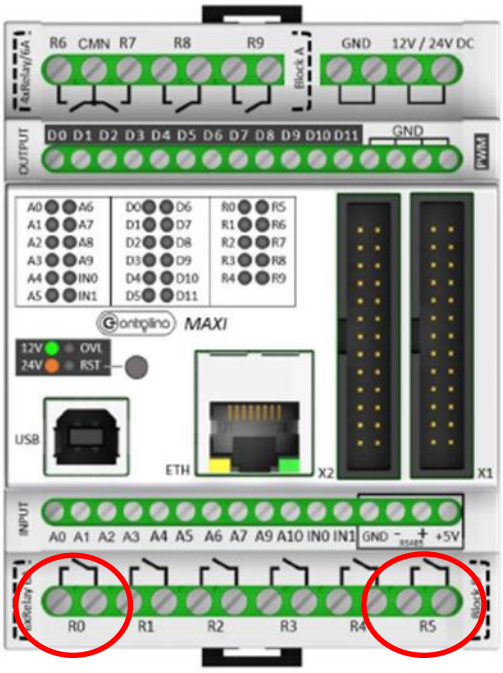


Kuva 18. TFT LCD -näytön liitäminen ohjaimeen [15].

4.5 Pumpun ja valojen liittäminen

Pumppu on liitetty releen R5 (kuva 19) kautta 12 voltin riviliitinryhmään ja suoraan maadoitettuun ryhmään. Tämä sallii pumpun ohjauksen releellä sen katkaistessa pumpun sähkönsyötön, jolloin pumppu on päällä ainoastaan, kun rele on kiinni.

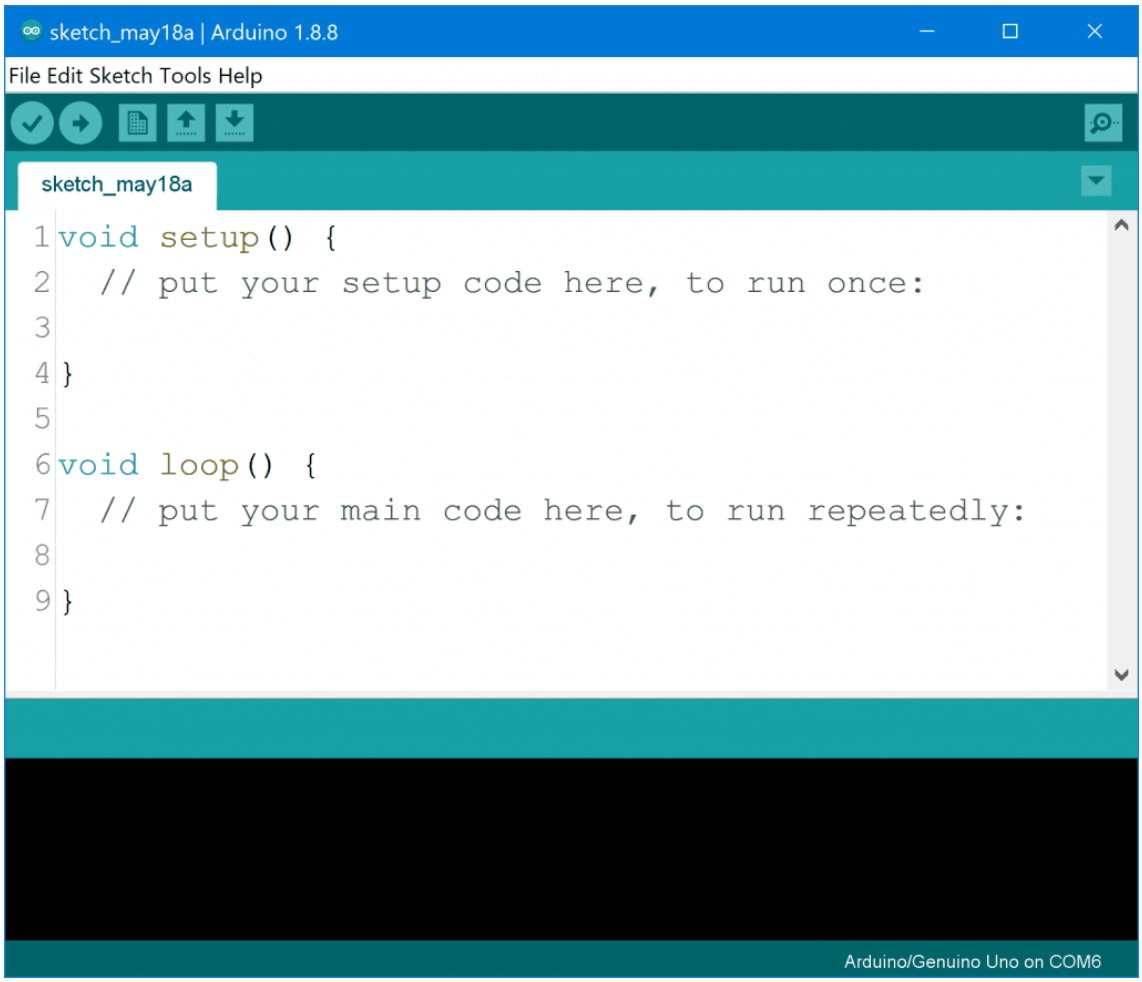
Valoilla on oma virtalähde, joka on kytketty releen R0 (kuva 19) kautta suoraan verkkovirtaan. Samalla tavalla kuin pumppu, valot saavat virtaa vain, kun rele on kiinni.



Kuva 19. Projektissa käytettävät releet R0 ja R5.

5 Ohjelmointi

Ohjelmointi on tehty Arduino IDE -ohjelmointiympäristössä (kuva 20). Arduino IDE mahdollistaa ohjelman kirjoittamisen ja lataamisen prosessoriin. Koodin kirjoituskieli on C++, mutta Arduino IDE mahdollistaa erilaisten valmiiden ohjelmakirjastojen lataamisen ja sisältää oman Arduino-kirjaston, joka yksinkertaistaa ohjelmointia. Arduino IDE -koodista käytetään nimitystä Sketch.



Kuva 20. Arduino IDE -pääikkuna, jossa Setup- ja Main-silmukat.

5.1 Yleinen ohjelma-arkkitehtuuri

Arduino-ohjelma koostuu yksinkertaisimmillaan Setup-silmukasta, joka sisältää koodin asetukset ja asiat, joiden halutaan toistuvan vain kerran prosessorin käynnistyessä, ja Main-silmukasta, koodin pääohjelmasta, joka toistuu jatkuvasti ylhäältä alas prosessorin ollessa päällä (kuva 20). Koska Arduinossa on vain yksi prosessoriydin, pystyy se suorittamaan vain yhden komennon kerrallaan.

Projektin ohjelma koostuu useammasta silmukasta. Näitä Setup- tai Main-silmukan ulkopuolisia silmukoita kutsutaan funktioiksi. Ne ovat koodipätkiä, jotka suorittavat tiettyä funktiota ja kirjoittamalla koodin omaan funktioon, voidaan se kutsua pääohjelmassa yksinkertaisemmin (esimerkkikoodi 7).

```
int ledpinni = 5;

Void loop(){
     sytytaled();
}
Void sytytaled (){
     digitalWrite(ledpinni, HIGH);
}
```

Esimerkkikoodi 7. Kutsutaan pääohjelmassa sytytaled-funktio, joka sytyttää ledin prosessorin pinnissä 5.

Yleisessä ohjelmarakenteessa ensiksi määritetään käytettävät muuttujat, pinnit, kirjastot ja muut vaadittavat määritelmät. Tämän jälkeen tulee Setup-silmukka, jota seuraa pääohjelma. Pääohjelman alle kirjoitetaan ohjelman funktiot. Mikään ei kuitenkaan estä funktioiden kirjoittamista ennen Setup- tai Main-silmukoita (esimerkkikoodi 8).

```
void button_interrupt_handler(){
} bool read_button() {
}
 void setup() {
}
void loop() {
}
void temphumRead() {
}
void displayprint() {
}
void counter1() {
}
void counter2() {
}
void counter3() {
}
void buttonPress() {
}
void dataPrint() {
}
void excelPrint() {
}
void rpmCount(){
}
```

Esimerkkikoodi 8. Projektin ohjelman rakenne.

5.2 Antureiden ohjelmointi

Projektissa käytettävät DHT22-anturit vaativat kaksi kirjastoa. Ensimmäinen kirjasto on yleinen kirjasto kaikille Adafruit Industriesin tuottamille antureille. Toinen kirjasto on DHT-antureille tarkoitettu kirjasto. DHT-kirjasto vaatii ylemmän tason Adafruit sensors -kirjaston toimiakseen (esimerkkikoodi 9).

```
#include <Adafruit_Sensor.h>
#include <DHT.h>
```

Esimerkkikoodi 9. Anturien käyttöön vaadittavat ohjelmakirjastot.

Ohjelmassa tulee määrittää omat pinnit jokaiselle anturille. Koska anturit on liitetty Controllinon X1- ja X2-pinniterminaaleihin, ei pinnien nimien eteen tarvitse

laittaa CONTROLLINO-liitettä. Jokaiselle määritellylle pinnille pitää määrittää anturin tyyppi, joka on DHT22 (esimerkkikoodi 10).

```
#define DHTPIN1 A0   //Pin for sensor
#define DHTTYPE DHT22 //definition of the sensor type (AM2302)
#define DHTPIN2 A1
#define DHTTYPE DHT22
#define DHTPIN3 A2
#define DHTTYPE DHT22
#define DHTPIN4 A7
#define DHTTYPE DHT22
```

Esimerkkikoodi 10. Anturien pinnien ja anturityypin määrittäminen.

Anturien pinnien määrittämisen jälkeen jokaisesta anturista luodaan instanssi, jotta jokaisella anturilla on uniikki nimitys (esimerkkikoodi 11).

```
DHT dht1(DHTPIN1, DHTTYPE); //create instances for sensors
DHT dht2(DHTPIN2, DHTTYPE);
DHT dht3(DHTPIN3, DHTTYPE);
DHT dht4(DHTPIN4, DHTTYPE);
```

Esimerkkikoodi 11. Instanssien luonti antureille.

Setup silmukassa jokainen anturi pitää alustaa, jotta ohjelma osaa alkaa vastaanottamaan niiden dataa (esimerkkikoodi 12). Tämän jälkeen pääohjelmassa data voidaan lukea erityisellä komennolla, joka löytyy anturien ohjelmakirjastosta (esimerkkikoodi 13). Kun tätä komentoa kutsutaan ohjelmassa, antaa se anturin datan suoraan muunnettuna käyttökelpoiseen muotoon ja ohjelmassa ei tarvitse tehdä erikseen datalle laskutoimituksia. Projektin ohjelmassa kaikkien anturien data luetaan ja asetetaan omiin muuttujiin, jotta sen käsittely on helpompaa (esimerkkikoodi 13). Tätä varten on ohjelmaan luotu oma funktio.

```
dht1.begin(); //initialize sensors
dht2.begin();
dht3.begin();
dht4.begin();
```

Esimerkkikoodi 12. Antureiden alustaminen.

```
void temphumRead() { //self explanatory, put sensor data into varia-
bles

  hum1 = dht1.readHumidity();
  temp1 = dht1.readTemperature();
  hum2 = dht2.readHumidity();
  temp2 = dht2.readTemperature();
  hum3 = dht3.readHumidity();
  temp3 = dht3.readTemperature();
  ulkohum = dht4.readHumidity();
  ulkotemp = dht4.readTemperature();
}
```

Esimerkkikoodi 13. Funktio, joka lukee antureiden arvot ja laittaa ne muuttujiin, joita muut funktiot voivat lukea.

5.3 Ilmastoinnin ohjelmointi

Ilmastoinnista vastaavien tuulettimien ohjaaminen vaatii PWM-signaalia. Signaalia lähettävät lähdöt tulee Controllino-kirjaston avulla määrittää PWM-lähdöiksi (esimerkkikoodi 14 ja 15).

```
#include <Controllino.h>
```

Esimerkkikoodi 14. Controllinon käyttöön vaadittava kirjasto.

```
const int fan_control_pin1 = CONTROLLINO_PIN_HEADER_PWM_07; //PWM Fan1
const int fan_control_pin2 = CONTROLLINO_PIN_HEADER_PWM_09;//PWM Fan2
const int fan_control_pin3 = CONTROLLINO_PIN_HEADER_PWM_11;//PWM Fan3
const int fan_control_pin4 = CONTROLLINO_PIN_HEADER_PWM_08; //PWM Fan4
const int fan_control_pin5 = CONTROLLINO_PIN_HEADER_PWM_10;//PWM Fan5
const int fan_control_pin6 = CONTROLLINO_PIN_HEADER_PWM_02;//PWM Fan6
```

Esimerkkikoodi 15. Tuulettimien PWM-lähtöjen määrittäminen.

Setup silmukassa PWM-lähdöt määritellään lähdöiksi ja luodaan ohjeet, miten ohjelma käsittelee tuulettimien tachometrien signaalit. Tachometrit on liitetty interrupt-tuloihin, jotta ohjelma pystyy vastaanottamaan niiden signaalia jatkuvasti pääohjelman ulkopuolella. AttachInterrupt-komennolla määritetään Interrupt-tulon pinni, funktio jonka pinnin aktivaatio kutsuu (esimerkkikoodi 17) ja milloin pinni aktivoituu. Aktivaatiotapoja on neljä:

- LOW, jossa pinni aktivoituu aina, kun sen looginen signaali on matala.
- CHANGE, jossa pinni aktivoituu aina, kun sen arvo muuttuu.
- RISING, jossa pinni aktivoituu aina, kun sen looginen signaali menee matalasta korkeaksi.
- FALLING, jossa pinni aktivoituu aina, kun sen looginen arvo menee korkeasta matalaksi.

Tachometrien kanssa halutaan käyttää Rising-aktivaatiota, jotta pinni aktivoituu aina, kun anturi lähettää arvon. Tachometrin arvon laskeminen pohjautuu sen lähettämien signaalien määrään sekunnissa (esimerkkikoodi 18). Kun arvo muutetaan kierroksiksi minuutissa, tulee anturin antama arvo jakaa kahdella, koska se antaa kaksi sykäystä sekunnissa, ja kertoa 60:lla, jotta arvo muutetaan minuutteihin. Muunnetut arvot tallennetaan muuttujiin, jotta eri tuulettimien pyörimisnopeuksia on helppo käsitellä.

```
void setup() {
  pinMode( fan_control_pin1, OUTPUT); //Set fan control pin pinmode
  pinMode( fan_control_pin2, OUTPUT);
  pinMode( fan_control_pin3, OUTPUT);
  pinMode( fan_control_pin4, OUTPUT);
  pinMode( fan_control_pin5, OUTPUT);
  pinMode( fan_control_pin6, OUTPUT);
  analogWrite(fan_control_pin4, 0); //set fanspeed to zero
  analogWrite(fan_control_pin5, 0);
  analogWrite(fan_control_pin6, 0);
  attachInterrupt(digitalPinToInterrupt(18), counter1, RISING); //at-
tachinterrupt functions for fan tachiometers
  attachInterrupt(digitalPinToInterrupt(2), counter2, RISING);
  attachInterrupt(digitalPinToInterrupt(3), counter3, RISING);
  analogWrite(fan_control_pin1, 255); //set noctua fans to max to
"wake them up"
  analogWrite(fan_control_pin2, 255);
  analogWrite(fan_control_pin3, 255);
  analogWrite(fan_control_pin1, 170); //noctua fans on but not full on
(lowest values that keep them spinning
  analogWrite(fan_control_pin2, 168);
  analogWrite(fan_control_pin3, 168);
}
```

Esimerkkikoodi 16. Tuulettimien Setup-koodi. Ensiksi tuulettimien ohjauslähdöt määritetään lähdöiksi. Tachometrien interrupt-tulot määritetään. Tuulettimet käynnistetään ja asetetaan alkuarvoihin.

```
void counter1() { //Tachiometer count for first(highest) noctua fan.
  count1++;
}
void counter2() {//Tachiometer count for second noctua fan.
  count2++;
}
void counter3() {//Tachiometer count for third noctua fan.
  count3++;
}
```

Esimerkkikoodi 17. Funktiot tachometrien signaalien vastaanottamiseksi

```
void rpmCount(){ //counting for tachiometer, gives two pulses per sec-
ond so we add the interrupt pulses for a second and then multiply by
30(60/2 because TWO pulses per second)
  rpm1 = count1 * 30; // accurate enough
  count1 = 0;
  rpm2 = count2 * 30;
  count2 = 0;
  rpm3 = count3 * 30;
  count3 = 0;
}
```

Esimerkkikoodi 18. Funktio tachometrin arvon muuntamiseksi rpm-muotoon.

Tuulettimien nopeuden ohjaus perustuu ilmankosteuteen. Jokaisen kaapin kerroksen tuulettimet säätyvät itsenäisesti kerroksen anturin antamien arvojen mukaisesti. Säätö tapahtuu käyttäen if-silmukoita, joissa tuulettimelle annetaan uusi nopeus, jos kosteusprosentti siirtyy pois asetetulta väliltä (esimerkkikoodi 19). Ohjauksen tarkoitus on rajoittaa kaapin sisäinen ilmankosteus tietyn rajan alapuolelle nostamalla tuulettimien tehoa asteittain ilmankosteuden noustessa.

```
if(hum1>80){ //simple if loop controls for fans the air humidity dic-
tates
          analogWrite(fan_control_pin1, 255);
          analogWrite(fan_control_pin4, 255);
        } else if(80>hum1&&hum1>75){
          analogWrite(fan_control_pin1, 170);
          analogWrite(fan_control_pin4, 170);
        }else if(75>hum1&&hum1>70){
          analogWrite(fan_control_pin1, 0);
          analogWrite(fan_control_pin4, 150);
        }else if(hum1<70){
          analogWrite(fan_control_pin1, 0);
          analogWrite(fan_control_pin4, 120);
        }
        if(hum2>80){
          analogWrite(fan_control_pin2, 255);
          analogWrite(fan_control_pin5, 255);
        } else if(80>hum2&&hum2>75){
          analogWrite(fan_control_pin2, 170);
          analogWrite(fan_control_pin5, 170);
        }else if(75>hum2&&hum2>70){
          analogWrite(fan_control_pin2, 0);
          analogWrite(fan_control_pin5, 150);
        }else if(hum2<70){
          analogWrite(fan_control_pin2, 0);
          analogWrite(fan_control_pin5, 120);
        }
        if(hum3>80){
          analogWrite(fan_control_pin3, 255);
          analogWrite(fan_control_pin6, 255);
        } else if(80>hum3&&hum3>75){
          analogWrite(fan_control_pin3, 170);
          analogWrite(fan_control_pin6, 170);
        }else if(75>hum3&&hum3>70){
          analogWrite(fan_control_pin3, 0);
          analogWrite(fan_control_pin6, 150);
        }else if(hum3<70){
          analogWrite(fan_control_pin3, 0);
          analogWrite(fan_control_pin6, 120);
        }
     }
```

Esimerkkikoodi 19. Ohjelma tuulettimien nopeuden säädölle ilmankosteuden mukaan.

5.4 Pumpun ja valojen ohjelmointi

Pumppua ja valoja ohjataan mikrokontrollerin releillä R0 ja R5 (esimerkkikoodi 20). Tämä yksinkertaistaa ohjelmaa huomattavasti, mutta rajaa säädön päälle/pois-tyyliseksi. Controllinon RTC-piiriä käyttämällä voidaan pumpulle ja valoille asettaa ajoitus. Tämä on toteutettu if-silmukalla, missä silmukan ehtona on tietty aikaväli (esimerkkikoodi 21).

```
int lightPin = CONTROLLINO_R0; //light relay
int pumpPin = CONTROLLINO_R5; //pump Relay
```

Esimerkkikoodi 20. Pumppua ja valoja ohjaavien releiden määrittäminen.

```
        if(currentHour==9&&currentMin>=1&&currentMin<=3){//timing for
the pump to pump water for the plants
            analogWrite(pumpPin, 255);
            Pumppu = 1; //set pump status to 1 for printing
        } else{
          analogWrite(pumpPin, 0);
          Pumppu = 0;       //set pump status to 0 for printing
        }
           if(currentHour>=5&&currentHour<=8){ //timing for lights to
shine, in my world the night only lasts for 4 hours
            analogWrite(lightPin, 0);
            Valo = 0;          //set light status to 0 for printing
        } else{
          analogWrite(lightPin, 255);
          Valo = 1; //set light status to 1 for printing

        }
```

Esimerkkikoodi 21. if-silmukka pumpun ja valojen aikaohjaamiseen RCT-piirin avulla.

5.5 Näytön ohjelmointi

TFT LCD -näytön ohjelmointi pohjaa suuresti ohjelmakirjastojen käyttöön. Ilman kirjastoja näytön käyttö on monimutkaista, koska datan kirjoittaminen tapahtuu lähettämällä Hex-koodeja, jotka vastaavat näytön pikseleitä. Kirjaston valmis

funktio osaa automaattisesti pikseleiden kutsumisen näytölle kirjoitettaessa (esimerkkikoodi 22). Näytölle määritetään, mitä väriä tietty väriä kuvaava muuttuja vastaa. Näyttö käyttää RGB-järjestelmää väreille ja ne määritetään väriä vastaavalla Hex-koodilla. Osa käytettävistä pinneistä tulee määrittää koodissa, koska Controllinon pinnit eivät täysin vastaa näytölle tyypillistä Arduino-mikrokontrolleria.

```
#include <SPI.h>
#include <Adafruit_GFX.h>
#include <TFT_ILI9163C.h>


// Color definitions
#define BLACK   0x0000 //color designations
#define BLUE    0x001F
#define RED     0xF800
#define GREEN   0x07E0
#define CYAN    0x07FF
#define MAGENTA 0xF81F
#define YELLOW  0xFFE0
#define WHITE   0xFFFF


#define __CS 5 //pin definitions for the display as we use Arduino
mega
#define __DC 6
#define __RST 7


TFT_ILI9163C display = TFT_ILI9163C(__CS, __DC, __RST); //tell our
custom pins to the display
float p = 3.1415926; //define pi
```

Esimerkkikoodi 22. TFT LCD -näytön kirjastot ja määritykset. Määritetään näytön värit, pii ja määritellään pinnit, joita näyttö käyttää datan lähettämisen ja vastaanottamiseen.

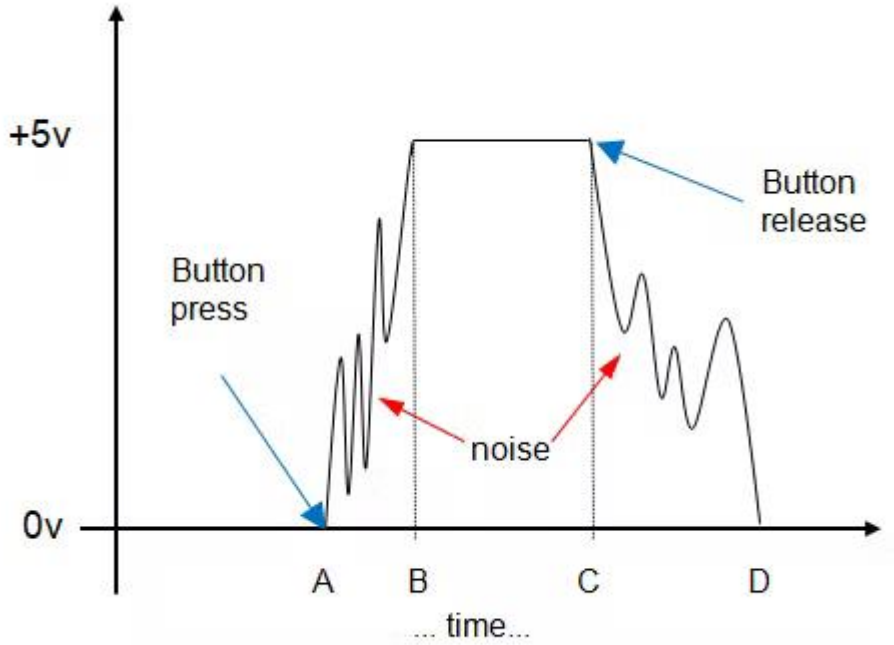
Näytölle kirjoittaminen tapahtuu kuten esimerkissä (esimerkkikoodi 23). Tyhjennetään ruutu, asetetaan teksti alkamaan näytön vasemmasta yläkulmasta (0,0 vastaa ensimmäistä pikseliä näytön pikseleitä kuvaavassa matriisissa), valitaan tekstin koko ja väri ja lopuksi kirjoitetaan haluttu teksti printtauskomennon sisään.

```
display.clearScreen();//clear the display
  display.setCursor(0, 0);//set cursor to first line on first row
  display.setTextSize(1); //set text size
  display.setTextColor(CYAN); //set text color(check start of the code
for color Hexcodes
  display.println("Growing chamber 1.0."); //writing on display
```

Esimerkkikoodi 23. Näytölle kirjoittamisessa käytettyjä komentoja. Näytön ohjelmakirjasto sallii yksinkertaisten komentojen käytön.

5.6 Painonapin ohjelmointi

Ongelma painikkeen ohjelmoinnissa on sen painamisesta aiheutuva häiriö (kuva 21). Painikkeen painaminen sulkee sen läpi kulkevan piirin, mutta ennen piirin sulkeutumista aiheutuu häiriötä signaaliin. Mikrokontrolleri voi havaita nämä signaalipiikit, jolloin pinni aktivoituu monia kertoja ennen ja jälkeen painikkeen painamisen.



Kuva 21. Palautuvan painonapin painamisesta aiheutuva häiriö.

Ongelman ratkaisu on jakaa painikkeen painaminen kahteen osaan ohjelmassa. Ensimmäinen osa havaitsee ja tallentaa Interrupt-tulon aktivoitumisen ja toinen

osa hoitaa painikkeen palautumisen ja siitä aiheutuvan häiriön. Ohjelmassa on käytetty valmista koodia painikkeen käsittelyä varten [16].

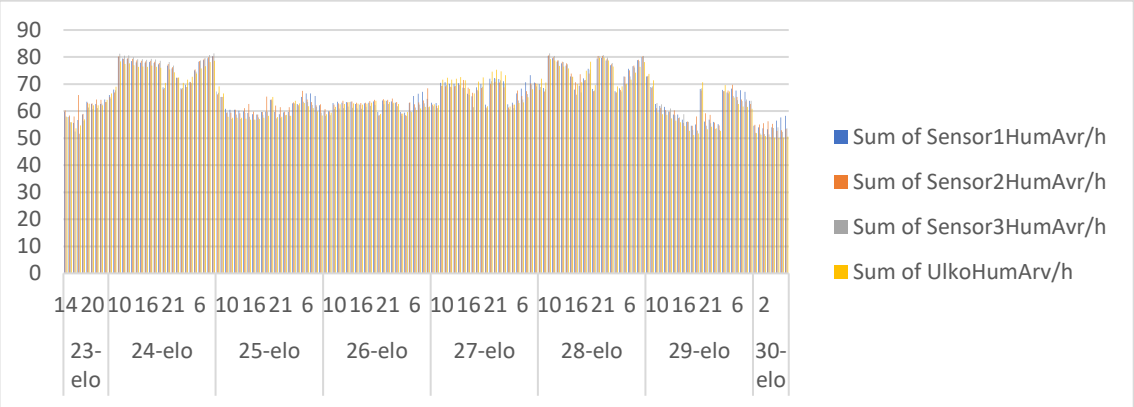
6 Yhteenveto

Projektin tavoitteena oli luoda kasvatuskaappi, joka kykenee itsenäisesti luomaan kasvien kasvattamiseen tarvittavan ympäristön. Projekti onnistui odotusten mukaisesti ja prototyyppi on käytössä Little Garden-yrityksen tiloissa kasvattamassa ituja yrityksen käyttöön.

Projekti vaati monipuolista osaamista kone-, automaatio- ja sähkötekniikan aloilta, ja suurimmat esteet työn valmistumiselle olivat ohjelmointi ja Controllino-mikrokontrollerin käyttö. Työn aihe Controllinon ominaisuuksien ja käytön selvittämisestä tuli tarpeesta ohjeistukselle mikrokontrollerin käytöstä.

Kasvatuskaapin toiminta testaaminen toteutettiin 23–30. elokuuta välisenä aikana, jolloin kaappi kasvatti kasveja onnistuneesti (taulukko 2).

Taulukko 2. Keskiarvoinen ilmankosteus kaapin sisällä viikon kestävän testin aikana.



Testin tulokset osoittivat kasvatuskaapin ominaisuuksien toimivan suunnitellusti ja pitkiä ajanjaksoja. Kasvatuskaappi on ollut testin jälkeen onnistuneesti jatkuvassa käytössä.

Prototyypin jatkokehityksessä se voidaan yhdistää internetiin ja luoda siitä laite IoT-verkkoon. Tämä mahdollistaisi kaapin monitoroinnin etänä ja mahdollistaisi kasvatuksen optimoinnin suuremman datamäärän avulla. Internet-yhteyden avulla voitaisiin hallita monia samanlaisia kaappeja samanaikaisesti. Jatkokehitys ei kuitenkaan ole osa projektia.

Lähdeluettelo

[1]	Red Hat, ”What is open source?,” Red Hat, 24 Lokakuu 2019. Verkkoaineisto. Available: https://www.redhat.com/en/topics/open-source/what-is-open-source. Luettu 23.10.2022.
[2]	Tieteen termipankki, ”SELV-järjestelmä,” Tieteen termipankki, 8 Helmikuu 2014. Verkkoaineisto. Available: https://tieteentermipankki.fi/wiki/S%C3%A4hk%C3%B6tekniikka:SELV-j%C3%A4rjestelm%C3%A4. Luettu 23.10.2022.
[3]	CONELCOM, ”CONTROLLINO Instruction Manual,” CONELCOM, 2016. Verkkoaineisto. Available: https://www.controllino.com/downloads/. Luettu 23.10.2022.
[4]	Arduino Project Hub, ”Interrupts basics,” MIT, 10 Huhtikuu 2020. Verkkoaineisto. Available: https://create.arduino.cc/projecthub/rafitc/interrupts-basics-f475d5. Luettu 23.10.2022.
[5]	CONELCOM, ”Controllino Maxi Pinout Table v1.1,” 14 Heinäkuu 2015. Verkkoaineisto. Available: https://www.controllino.com/downloads/. Luettu 23.10.2022.
[6]	CONTROLLINO-Support, ”CONTROLLINO_Library,” CONELCOM, 21 Huhtikuu 2021. Verkkoaineisto. Available: https://github.com/CONTROLLINO-PLC/CONTROLLINO_Library. Luettu 24.10.2022.
[7]	MEAN WELL, ”NDR-120-SPEC,” MEAN WELL, 20 09 2022. Verkkoaineisto. Available: https://www.meanwell.com/productPdf.aspx?i=137. Luettu 7.10.2022.
[8]	FIBOX, ”EKPK 230 T 2538158,” FIBOX, Verkkoaineisto. Available: https://www.fibox.com/fi/products/ek/ekpk-ekpl/2538158. Luettu 7.10.2022.

[9]	Adafruit Industries, ”DHT11, DHT22 and AM2302 Sensors,” Adafruit Industries, 29 Heinäkuu 2012. Verkkoaineisto. Available: https://learn.adafruit.com/dht. Luettu 19.10.2022.
[10]	Electroschematics, ”AM2302 / DHT22 Datasheet,” Verkkoaineisto. Available: https://www.electroschematics.com/am2302-dht22-datasheet/. Luettu 23.10.2022.
[11]	MEAS, ”HPC199_3 HTU21D(F) Sensor Datasheet,” Lokakuu 2013. Verkkoaineisto. Available: www.meas-spec.com. Luettu 19.10.2022.
[12]	Arctic, ”P12 PWM PST,” Verkkoaineisto. Available: https://www.arctic.de/en/P12-PWM-PST/ACFAN00170A. Luettu 21.10.2022.
[13]	Noctua, ”nf-f12-industrialppc-3000-pwm,” Noctua, Verkkoaineisto. Available: https://noctua.at/en/nf-f12-industrialppc-3000-pwm/specification. Luettu 21.10.2022.
[14]	Adafruit Industries, ”Connecting to a DHTxx Sensor,” Adafruit Industries, 29 Heinäkuu 2012. Verkkoaineisto. Available: https://learn.adafruit.com/dht/connecting-to-a-dhtxx-sensor. Luettu 25.10.2022.
[15]	Electronics-lab, ”Using the 1.44″ Color TFT display (ILI9163C) with Arduino,” Verkkoaineisto. Available: https://www.electronics-lab.com/project/using-1-44-color-tft-display-ili9163c-arduino/. Luettu 25.10.2022.
[16]	R. Bentley, ”Button Switch Using An External Interrupt,” hackster.io, 3 Helmikuu 2021. Verkkoaineisto. Available: https://www.hackster.io/ronbentley1/button-switch-using-an-external-interrupt-7879df. Luettu 25.10.2022.
[17]	Arduino, ”attachInterrupt(),” Arduino, 21 Toukokuu 2021. Verkkoaineisto. Available: https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/. Luettu 25.10.2022.

Laitteen ohjelma

```
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <SPI.h>
#include <Adafruit_GFX.h>
#include <TFT_ILI9163C.h>
#include <Arduino.h>
#include <Controllino.h>


// Color definitions
#define BLACK   0x0000 //color designations
#define BLUE    0x001F
#define RED     0xF800
#define GREEN   0x07E0
#define CYAN    0x07FF
#define MAGENTA 0xF81F
#define YELLOW  0xFFE0
#define WHITE   0xFFFF


#define __CS 5 //pin definitions for the display as we use Arduino
mega
#define __DC 6
#define __RST 7


#define DHTPIN1 A0   //Pin for sensor
#define DHTTYPE DHT22 //definition of the sensor type (AM2302)
#define DHTPIN2 A1
#define DHTTYPE DHT22
#define DHTPIN3 A2
#define DHTTYPE DHT22
#define DHTPIN4 A7
#define DHTTYPE DHT22

TFT_ILI9163C display = TFT_ILI9163C(__CS, __DC, __RST); //tell our
custom pins to the display
float p = 3.1415926; //define pi

DHT dht1(DHTPIN1, DHTTYPE); //create instances for sensors
DHT dht2(DHTPIN2, DHTTYPE);
DHT dht3(DHTPIN3, DHTTYPE);
DHT dht4(DHTPIN4, DHTTYPE);
int lightPin = CONTROLLINO_R0; //light relay
int pumpPin = CONTROLLINO_R5; //pump Relay
float hum1;  //variables for sensors
float temp1;
float hum2;
float temp2;
float hum3;
float temp3;
float ulkotemp;
float ulkohum;

const int fan_control_pin1 = CONTROLLINO_PIN_HEADER_PWM_07; //PWM Fan1
const int fan_control_pin2 = CONTROLLINO_PIN_HEADER_PWM_09;//PWM Fan2
const int fan_control_pin3 = CONTROLLINO_PIN_HEADER_PWM_11;//PWM Fan3
const int fan_control_pin4 = CONTROLLINO_PIN_HEADER_PWM_08; //PWM Fan4
const int fan_control_pin5 = CONTROLLINO_PIN_HEADER_PWM_10;//PWM Fan5
const int fan_control_pin6 = CONTROLLINO_PIN_HEADER_PWM_02;//PWM Fan6
const int button_switch = 19; //pin for button, interrupt pin
```

```
long previousMillis = 0;  //variables to use in the three state ma-
chines
long previousMillis1 = 0;
long previousMillis2 = 0;
long previousMillis3 = 0;
long previousMillis4 = 0;
long previousMillis5 = 0;
long previousMillis6 = 0;
int count1 = 0; //variables for tachiometer counts
int count2 = 0;
int count3 = 0;
int rpm1; //variables to save rpm count to
int rpm2;
int rpm3;
volatile int button_press = 1; //set button to 1
int button_read; //variable to put button state into
int n; //for printing timing setting in setup loop
int currentHour =0; //variables for saving time info to
int currentMin=0;
int currentSec=0;
int Valo = 0; //variable for light state
int Pumppu = 0; //variable for pump state

#define switched                              true // value if the but-
ton switch has been pressed
#define triggered                             true // controls interrupt
handler
#define interrupt_trigger_type            RISING // interrupt trig-
gered on a RISING input
#define debounce                                10 // time to wait in
milli secs

volatile  bool interrupt_process_status = {
  !triggered                                      // start with no
switch press pending, ie false (!triggered)
};
bool initialisation_complete =            false; // inhibit any inter-
rupts until initialisation is complete
//
// ISR for handling interrupt triggers arising from associated button
switch
//
void button_interrupt_handler()
{
  if (initialisation_complete == true)
  { //  all variables are initialised so we are okay to continue to
process this interrupt
    if (interrupt_process_status == !triggered) {
      // new interrupt so okay start a new button read process -
      // now need to wait for button release plus debounce period to
elapse
      // this will be done in the button_read function
      if (digitalRead(button_switch) == HIGH) {
        // button pressed, so we can start the read on/off + debounce
cycle wich will
        // be completed by the button_read() function.
        interrupt_process_status = triggered;  // keep this ISR 'qui-
et' until button read fully completed
      }
    }
  }
```

```
} // end of button_interrupt_handler

bool read_button() {
  int button_reading;
  // static variables because we need to retain old values between
function calls
  static bool     switching_pending = false;
  static long int elapse_timer;
  if (interrupt_process_status == triggered) {
    // interrupt has been raised on this button so now need to com-
plete
    // the button read process, ie wait until it has been released
    // and debounce time elapsed
    button_reading = digitalRead(button_switch);
    if (button_reading == HIGH) {
      // switch is pressed, so start/restart wait for button relealse,
plus end of debounce process
      switching_pending = true;
      elapse_timer = millis(); // start elapse timing for debounce
checking
    }
    if (switching_pending && button_reading == LOW) {
      // switch was pressed, now released, so check if debounce time
elapsed
      if (millis() - elapse_timer >= debounce) {
        // dounce time elapsed, so switch press cycle complete
        switching_pending = false;             // reset for next but-
ton press interrupt cycle
        interrupt_process_status = !triggered; // reopen ISR for busi-
ness now button on/off/debounce cycle complete
        return switched;                       // advise that switch
has been pressed
      }
    }
  }
  return !switched; // either no press request or debounce period not
elapsed
} // end of read_button function


void setup() {
  Serial.begin(9600); //start serial connection
  Controllino_RTC_init(0); //initialize Real Time Clock module on Con-
trollino
  Controllino_SetTimeDate(5,1,9,22,14,19,50);// (Day of the month, Day
of the week, Month, Year, Hour, Minute, Second) //Set date
  display.begin(); //Initialize the display
  uint16_t time = millis(); // for button
  time = millis() - time; // sets new time
  dht1.begin(); //initialize sensors
  dht2.begin();
  dht3.begin();
  dht4.begin();
  pinMode( fan_control_pin1, OUTPUT); //Set fan control pin pinmode
  pinMode( fan_control_pin2, OUTPUT);
  pinMode( fan_control_pin3, OUTPUT);
  pinMode( fan_control_pin4, OUTPUT);
  pinMode( fan_control_pin5, OUTPUT);
  pinMode( fan_control_pin6, OUTPUT);
  analogWrite(fan_control_pin4, 0); //set fanspeed to zero
  analogWrite(fan_control_pin5, 0);
```

```
  analogWrite(fan_control_pin6, 0);
  attachInterrupt(digitalPinToInterrupt(18), counter1, RISING); //at-
tachinterrupt functions for fan tachiometers
  attachInterrupt(digitalPinToInterrupt(2), counter2, RISING);
  attachInterrupt(digitalPinToInterrupt(3), counter3, RISING);
  pinMode(button_switch, INPUT);
  attachInterrupt(digitalPinToInterrupt(button_switch), //attachinter-
rupt for button
                  button_interrupt_handler, //see code for button be-
fore setup loop
                  interrupt_trigger_type);
  initialisation_complete = true; // open interrupt processing for
business
  analogWrite(fan_control_pin1, 255); //set noctua fans to max to
"wake them up"
  analogWrite(fan_control_pin2, 255);
  analogWrite(fan_control_pin3, 255);
  display.clearScreen();//clear the display
  display.setCursor(0, 0);//set cursor to first line on first row
  display.setTextSize(1); //set text size
  display.setTextColor(CYAN); //set text color(check start of the code
for color Hexcodes
  display.println("Growing chamber 1.0."); //writing on display
  delay(2000); //waiting to be cool
  display.println("Starting up."); //more text
  delay(2000); //more waiting
  display.print("Day: ");n = Controllino_GetDay(); display.println(n);
//Print date and time setting on display
  display.print("WeekDay: ");n = Controllino_GetWeekDay();dis-
play.println(n);
  display.print("Month: ");n = Controllino_GetMonth();dis-
play.println(n);
  display.print("Year: ");n = Controllino_GetYear();dis-
play.println(n);
  display.print("Hour: ");n = Controllino_GetHour();dis-
play.println(n);
  display.print("Minute: "); n = Controllino_GetMinute();dis-
play.println(n);
  display.print("Second: ");n = Controllino_GetSecond();dis-
play.println(n);
  delay(2000); //waiting for the rupture
  analogWrite(fan_control_pin1, 170); //noctua fans on but not full on
(lowest values that keep them spinning
  analogWrite(fan_control_pin2, 168);
  analogWrite(fan_control_pin3, 168);
  display.setTextSize(3);
  display.setTextColor(GREEN);
  display.clearScreen();
  delay(500);
  display.setCursor(0, 40);
  display.println("w");
  delay(100);
  display.setCursor(15, 40);
  display.println("e");
  delay(100);
  display.setCursor(30, 40);
  display.println("l");
  delay(100);
  display.setCursor(45, 40);
  display.println("c");
  delay(100);
```

```
  display.setCursor(60, 40);
  display.println("o");
  delay(100);
   display.setCursor(75, 40);
  display.println("m");
  delay(100);
   display.setCursor(90, 40);
  display.println("e");
  delay(2000);
  display.clearScreen();
  analogWrite(lightPin, 255); //set lights on so the plants know it's
day
  //analogWrite(pumpPin, 255);
  //delay(2000);
  //analogWrite(pumpPin, 0);
}

void loop() {

  buttonPress(); //check if someone pushed the button (button doesn't
do anyting yet
  unsigned long currentMillis = millis(); //set current millis with
millis function
        currentHour =Controllino_GetHour(); //put current time into
variables for printing and checking
        currentMin=Controllino_GetMinute();
        currentSec=Controllino_GetSecond();
  switch(button_press){ //switch function for the button to change the
mode
  case 1: //The mode 1
     if (currentMillis - previousMillis >= 1000) { //basic state ma-
chine set to a second.
          previousMillis = currentMillis; // we tell ourselves how
long we have been on
          temphumRead(); //read the sensors
          rpmCount(); //count the rpm from tachiometers and put it
into variables
          displayprint();//print on display
        if(hum1>80){ //simple if loop controls for fans the air humid-
ity dictates
          analogWrite(fan_control_pin1, 255);
          analogWrite(fan_control_pin4, 255);
        } else if(80>hum1&&hum1>75){
          analogWrite(fan_control_pin1, 170);
          analogWrite(fan_control_pin4, 170);
        }else if(75>hum1&&hum1>70){
          analogWrite(fan_control_pin1, 0);
          analogWrite(fan_control_pin4, 150);
        }else if(hum1<70){
          analogWrite(fan_control_pin1, 0);
          analogWrite(fan_control_pin4, 120);
        }
        if(hum2>80){
          analogWrite(fan_control_pin2, 255);
          analogWrite(fan_control_pin5, 255);
        } else if(80>hum2&&hum2>75){
          analogWrite(fan_control_pin2, 170);
          analogWrite(fan_control_pin5, 170);
        }else if(75>hum2&&hum2>70){
          analogWrite(fan_control_pin2, 0);
          analogWrite(fan_control_pin5, 150);
```

```
        }else if(hum2<70){
          analogWrite(fan_control_pin2, 0);
          analogWrite(fan_control_pin5, 120);
        }
        if(hum3>80){
          analogWrite(fan_control_pin3, 255);
          analogWrite(fan_control_pin6, 255);
        } else if(80>hum3&&hum3>75){
          analogWrite(fan_control_pin3, 170);
          analogWrite(fan_control_pin6, 170);
        }else if(75>hum3&&hum3>70){
          analogWrite(fan_control_pin3, 0);
          analogWrite(fan_control_pin6, 150);
        }else if(hum3<70){
          analogWrite(fan_control_pin3, 0);
          analogWrite(fan_control_pin6, 120);
        }
     }
        if (currentMillis - previousMillis1 >= 30000) { //machine for
printing to serial, we don't want to know every second
          previousMillis1 = currentMillis;
          //dataPrint(); //print data to serial for testing purposes
          excelPrint(); //print in excel form to extention tool to en-
able data logging
         }

        if(currentHour==9&&currentMin>=1&&currentMin<=3){//timing for
the pump to pump water for the plants
            analogWrite(pumpPin, 255);
            Pumppu = 1; //set pump status to 1 for printing
        } else{
          analogWrite(pumpPin, 0);
          Pumppu = 0;       //set pump status to 0 for printing
        }
           if(currentHour>=5&&currentHour<=8){ //timing for lights to
shine, in my world the night only lasts for 4 hours
            analogWrite(lightPin, 0);
            Valo = 0;          //set light status to 0 for printing
        } else{
          analogWrite(lightPin, 255);
          Valo = 1; //set light status to 1 for printing

        }

        break; //break the case like they broke my spirit

   case 2:
         analogWrite(pumpPin, 255);
         if (currentMillis - previousMillis2 >= 1000) { //basic state
machine set to a second.
          previousMillis2 = currentMillis; // we tell ourselves how
long we have been on
          temphumRead(); //read the sensors
          rpmCount(); //count the rpm from tachiometers and put it
into variables
          displayprint();//print on display
        if(hum1>80){ //simple if loop controls for fans the air humid-
ity dictates
          analogWrite(fan_control_pin1, 255);
          analogWrite(fan_control_pin4, 255);
        } else if(80>hum1&&hum1>75){
```

```
          analogWrite(fan_control_pin1, 170);
          analogWrite(fan_control_pin4, 170);
        }else if(75>hum1&&hum1>70){
          analogWrite(fan_control_pin1, 0);
          analogWrite(fan_control_pin4, 150);
        }else if(hum1<70){
          analogWrite(fan_control_pin1, 0);
          analogWrite(fan_control_pin4, 120);
        }
        if(hum2>80){
          analogWrite(fan_control_pin2, 255);
          analogWrite(fan_control_pin5, 255);
        } else if(80>hum2&&hum2>75){
          analogWrite(fan_control_pin2, 170);
          analogWrite(fan_control_pin5, 170);
        }else if(75>hum2&&hum2>70){
          analogWrite(fan_control_pin2, 0);
          analogWrite(fan_control_pin5, 150);
        }else if(hum2<70){
          analogWrite(fan_control_pin2, 0);
          analogWrite(fan_control_pin5, 120);
        }
        if(hum3>80){
          analogWrite(fan_control_pin3, 255);
          analogWrite(fan_control_pin6, 255);
        } else if(80>hum3&&hum3>75){
          analogWrite(fan_control_pin3, 170);
          analogWrite(fan_control_pin6, 170);
        }else if(75>hum3&&hum3>70){
          analogWrite(fan_control_pin3, 0);
          analogWrite(fan_control_pin6, 150);
        }else if(hum3<70){
          analogWrite(fan_control_pin3, 0);
          analogWrite(fan_control_pin6, 120);
        }
     }
/*if (currentMillis - previousMillis1 >= 30000) { //machine for print-
ing to serial, we don't want to know every second
          previousMillis1 = currentMillis;
          //dataPrint(); //print data to serial for testing purposes
          excelPrint(); //print in excel form to extention tool to en-
able data logging
         }*/



        if (currentMillis - previousMillis4 >= 180000) {
          previousMillis4 = currentMillis;
          analogWrite(pumpPin, 0);
          button_press = 1;
        }
        break;
      case 3:
          button_press = 1;
          break;


  }
}
```

```
void temphumRead() { //self explanatory, put sensor data into varia-
bles

  hum1 = dht1.readHumidity();
  temp1 = dht1.readTemperature();
  hum2 = dht2.readHumidity();
  temp2 = dht2.readTemperature();
  hum3 = dht3.readHumidity();
  temp3 = dht3.readTemperature();
  ulkohum = dht4.readHumidity();
  ulkotemp = dht4.readTemperature();


}
void displayprint() { // again read the function name, we print to
display

  display.fillScreen();
  display.setCursor(0, 0);
  display.setTextColor(CYAN);
  display.setTextSize(1);
  display.print("current setting: "); //shows setting
  display.println(button_press);
  display.print("Time: "); //shows time, hour:minute:second
  display.print(currentHour);
  display.print(":");
  display.print(currentMin);
  display.print(":");
  display.println(currentSec);
  display.println("Outer temp & hum:");
  display.print(ulkotemp);
  display.print(" c & ");
  display.print(ulkohum);
  display.println(" %");
  display.println("Sensor 1: Humidity:"); //we print our sensor data
  display.print(hum1);
  display.print(" %,");
  display.println("Temp:");
  display.print(temp1);
  display.println(" Celsius");

  display.println("Sensor 2: Humidity:");
  display.print(hum2);
  display.print(" %,");
  display.println("Temp:");
  display.print(temp2);
  display.println(" Celsius");

  display.println("Sensor 3: Humidity:");
  display.print(hum3);
  display.print(" %,");
  display.println("Temp:");
  display.print(temp3);
  display.println(" Celcius.");


  display.print("Fan1 Speed = "); // we print our fan data
  display.print(rpm1);
  display.println(" rpm");

  display.print("Fan2 Speed = ");
```

```
  display.print(rpm2);
  display.println(" rpm");
  display.print("Fan3 Speed = ");
  display.print(rpm3);
  display.println(" rpm");


}

void counter1() { //Tachiometer count for first(highest) noctua fan.
  count1++;
}
void counter2() {//Tachiometer count for second noctua fan.
  count2++;
}
void counter3() {//Tachiometer count for third noctua fan.
  count3++;
}
void buttonPress() {
  if (read_button() == switched) { //function to register buttonpress
and change the Mode (switch function in main loop).
    button_press++;

  }
}
void dataPrint() { //function to print to Serial, mainly for testing



  Serial.print("Fan1 Speed = ");
  Serial.print(rpm1);
  Serial.print(" rpm");
  Serial.print("% , Fan2 Speed = ");
  Serial.print(rpm2);
  Serial.print(" rpm");
  Serial.print("% , Fan3 Speed = ");
  Serial.print(rpm3);
  Serial.println(" rpm");
  Serial.print("Mode: ");
  Serial.println(button_press);
  Serial.print("Time: ");
  Serial.print(currentHour);
  Serial.print(":");
  Serial.print(currentMin);
  Serial.print(":");
  Serial.println(currentSec);
  if (Valo == 1){
    Serial.println("Valo päällä");
  } else {
    Serial.println("Valo Pois päältä");
  }
 if (Pumppu == 1){
    Serial.println("Pumppu päällä");
  } else {
    Serial.println("Pumppu Pois päältä");
  }

  Serial.print("Sensor 1: Humidity: ");
  Serial.print(hum1);
  Serial.print(" %, Temp: ");
  Serial.print(temp1);
```

```
  Serial.println(" Celsius");

  Serial.print("Sensor 2: Humidity: ");
  Serial.print(hum2);
  Serial.print(" %, Temp: ");
  Serial.print(temp2);
  Serial.println(" Celsius");

  Serial.print("Sensor 3: Humidity: ");
  Serial.print(hum3);
  Serial.print(" %, Temp: ");
  Serial.print(temp3);
  Serial.println(" Celcius.");
  Serial.println();
}
void excelPrint() { //function to print to Serial, mainly for testing



  Serial.print("FanSpeed 1");
  Serial.print('\t'); //change column
  Serial.print(rpm1);
  Serial.print('\t');
  Serial.print("FanSpeed 2");
  Serial.print('\t');
  Serial.print(rpm2);
  Serial.print('\t');
  Serial.print("FanSpeed 3");
  Serial.print('\t');
  Serial.print(rpm3);
  Serial.print('\t');
  Serial.print('\t');
  Serial.print(currentHour);
  Serial.print(":");
  Serial.print(currentMin);
  Serial.print(":");
  Serial.print(currentSec);
  Serial.print('\t');
  if (Valo == 1){
    Serial.print("Valo paalla");
    Serial.print('\t');
  } else {
    Serial.print("Valo Pois paalta");
    Serial.print('\t');
  }
 if (Pumppu == 1){
    Serial.print("Pumppu paalla");
    Serial.print('\t');
    Serial.print('\t');
  } else {
    Serial.print("Pumppu Pois paalta");
    Serial.print('\t');
    Serial.print('\t');
  }

  Serial.print(hum1);
  Serial.print('\t');
  Serial.print(temp1);
  Serial.print('\t');
  Serial.print('\t');
  Serial.print(hum2);
```

```
  Serial.print('\t');
  Serial.print(temp2);
  Serial.print('\t');
  Serial.print('\t');
  Serial.print(hum3);
  Serial.print('\t');
  Serial.print(temp3); //println changes row
  Serial.print('\t');
  Serial.print("ulkoarvot");
  Serial.print('\t');
  Serial.print(ulkohum);
  Serial.print('\t');
  Serial.println(ulkotemp);


}


void rpmCount(){ //counting for tachiometer, gives two pulses per sec-
ond so we add the interrupt pulses for a second and then multiply by
30(60/2 because TWO pulses per second)
  rpm1 = count1 * 30; // accurate enough
  count1 = 0;
  rpm2 = count2 * 30;
  count2 = 0;
  rpm3 = count3 * 30;
  count3 = 0;
}
```