Weatherstation (WS)

TEKNISK DOKUMENTATION

19 APRIL 2017

PROGRAMMERING AV INBYGGDA SYSTEM

MÖLK UTBILDNINGAR

FILIP LINDSTRÖM

FILLE044@GMAIL.COM

0768-831184

MÖLK UTBILDNING

Innehållsförteckning

1. Bakgrund och syfte

Väderstationer finns idag i alla möjliga former och storlekar, med funktioner från det allra mest basala till riktigt avancerade och ståtliga pjäser. De mäter allt från temperatur till hög och lågtryck för att förutse kommande väderprognos.

Min tanke med projektet var att skapa en väderstation utifrån de behov jag har av en sådan. Jag har idag en billig termometer i köksfönstret a'la Kjell&Co för under hundralappen, som enbart mäter temperatur inne och ute. Enheten i sig är helt batteriförsörjd och har inte behövt nya batterier på över ett år.

Istället för att lägga flera hundra, kanske till och med tusenlappar på en väderstation med fler funktioner tog jag tillfället i akt när det nalkades projekt i kursen Programmering av Inbyggda System.

I vår bostad har vi som många andra, växter i köksfönstret bredvid vår termometer. Vi har haft flertalet kryddor som alla sloknar och ger upp efter bara några veckor, detta såklart för att vi glömmer vattna, eller snarare, att vi glömmer bort att de står där. Detta problemet vet jag att många i min närhet har, och att dessa gärna vill se ett hjälpmedel för detta.

Olika kryddor vill ha olika mycket vatten, så att kunna välja vilken krydda som står i krukan som mäts verkar vara en viktig funktion. Säg att en basilika växt står i krukan, vars sensor är inställd efter den torra kaktusen, kommer systemet inte hjälpa många.

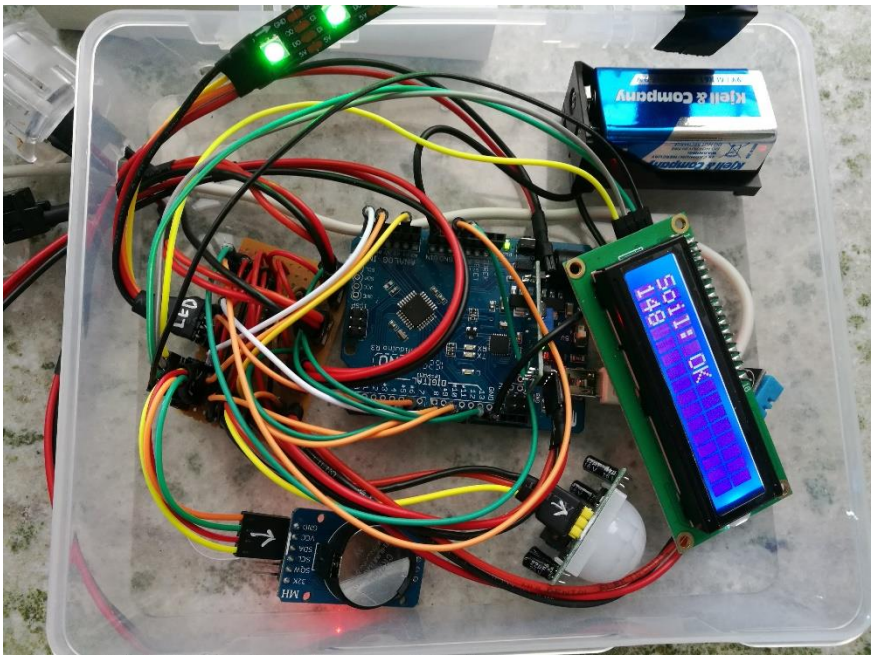
2. Produkt

Weatherstation (WS) är skapad för personer som söker en lättanvänd väderstation, och som inte gärna lägger ut flera hundra på en produkt av denna rang.

WS erbjuder en smidig formfaktor i bulkformat, där man som kund själv får komplettera med strömförsörjning, som sker via den klassiska Mini-USB standarden och en strömadapter. Detta för att hålla nere kostnaden samtidigt som de allra flesta hushåll innehar denna typ av utrustning i mängder.

WS har en tydlig och ljusstark LCD-skärm som säkerställer synbarhet dygnet runt. Denna ackomanjeras av två återfjädrande knappar av hög kvalitet som låter dig styra runt i menysystemet. I detta menysystem finns val att se tid och datum i realtid, se temperatur och luftfuktighet både inomhus och utomhus, samt möjligheten att granska fuktigheten i växtkrukan i detaljnivå. För att inte behöva ha den tredje sidan med jordfuktigheten uppe hela tiden finns en ljusstark LED-slinga som lyser upp din växt när den behöver vattnas.

För att inte slösa på strömmen i onödan sitter en IR-sensor monterad vars uppgift är att släcka ner skärmen och sensorer när ingen är i rummet. När rörelse känns av håller sig skärmen tänd i 50 sekunder för att sedan släckas, eller omedelbart tändas igen om rörelse fortfarande känns av.



3. Metod

3.1 Komponentlista för färdig produkt

Arduino Nano 3.0 – *Utvecklingskort* *Kjell.com*

Luxorparts DS3231 – *Realtidsklocka* *Kjell.com*

16x2 LCD I^2C skärm - *Amazon.com*

Luxorparts Rörelsedetektor - *Kjell.com*

Arkadknappar - *Kjell.com*

10k Ω resistor - *Kjell.com*

AdaFruit DotStar - *LED-strip* *Adafruit.com*

DHT11 – *Temperatur och luftfuktighetssensorer* *Kjell.com*

Experimentkort banor - *Kjell.com*

Kablar 0,5mm^2 - *Kjell.com*

Krympslang - *Kjell.com*

Stiftlist 20x2 - *Kjell.com*

Hylslist 40x1 - *Kjell.com*

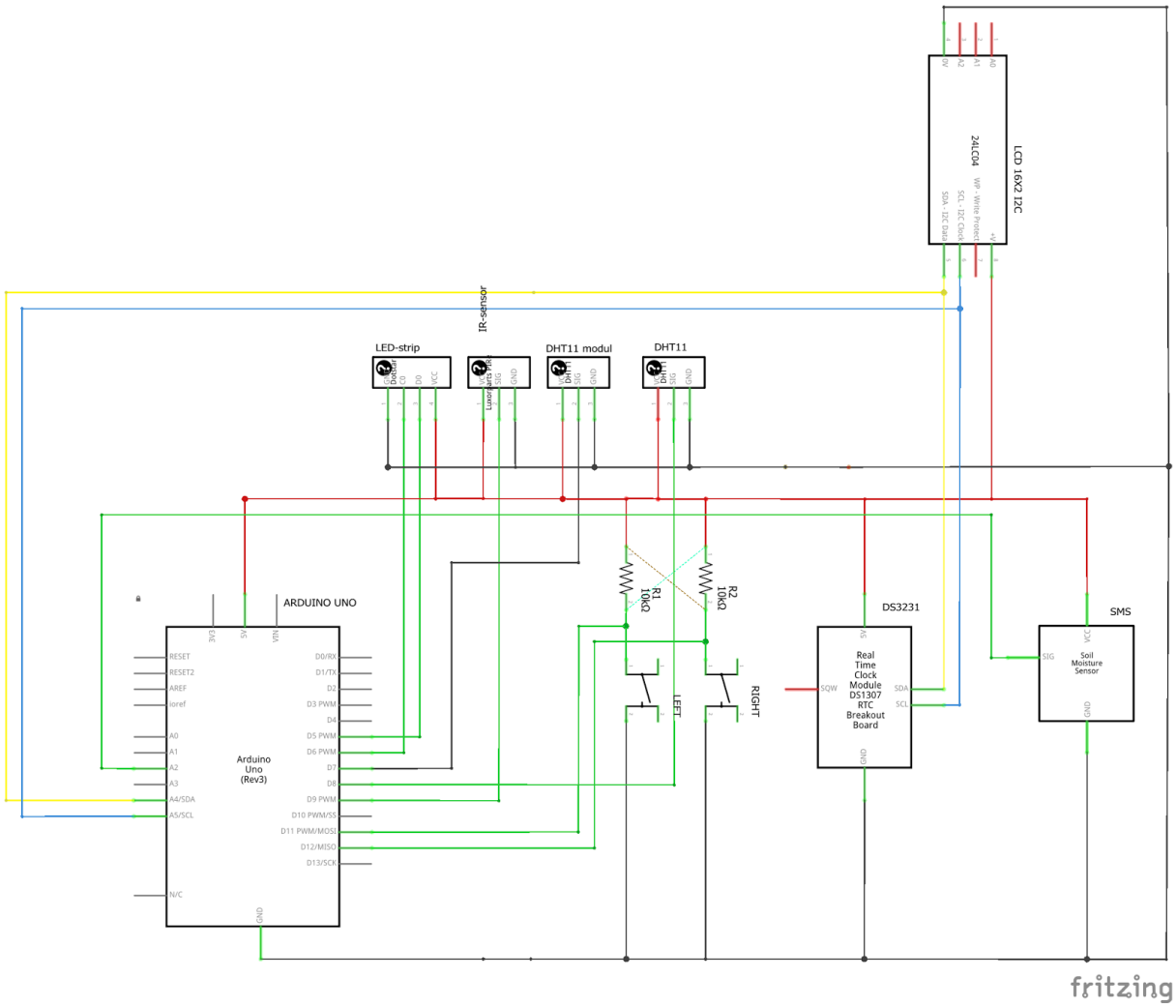
Plexiglas 300x300x3 - *Biltema.se*

3.2 Kretschema

Så här såg kretsen ut vid färdigställande.

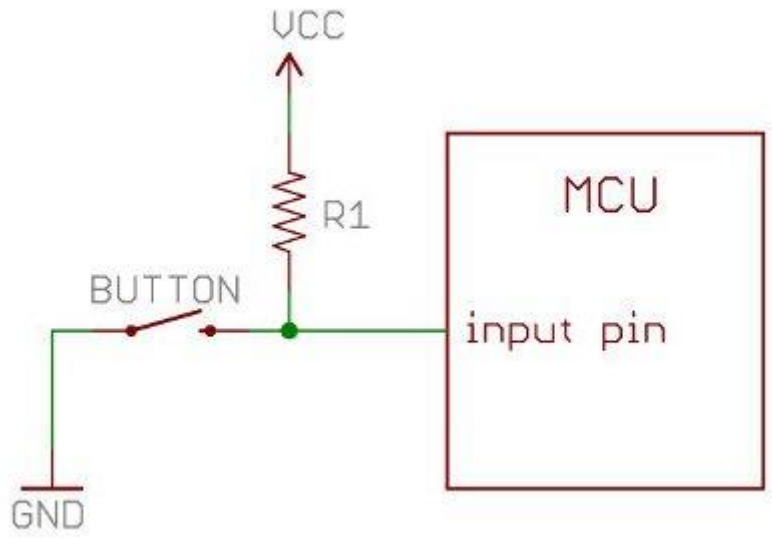
Samtliga kablar drogs till en lödplatta där allt sattes ihop, men i grund och botten var allt kopplat som på bilden nedan.

Svart linje är jord, Röd linje är 5V, grön linje är digital eller analog input eller output, gul linje är SDA och blå linje är SCL (I^2C).



Figur 3.2.1

Koppling av knapparna sker med såkallade pull-up resistorer, vilket innebär att 5V går genom en resistor och drar upp signalpinnen till en etta. När knappen trycks ner går strömmen den kortaste vägen till jord, vilket då gör att signalpinnen blir strömlös och således skickar tillbaka en nolla. Fenomenet visas i figur 3.2.2 nedan.



Figur 3.2 2

3.3 Byggprocess

I det här projektet började jag som vanligt med en Arduino Uno som utvecklingsplattform ackompanjerad av den klassiska breadboarden (Figur 3.3.1) för att enkelt och lättöverskådligt skapa en prototyp.

För att göra det så lätt som möjligt för mig själv kopplade jag in en komponent i taget för att testa funktionen och hur komponenten fungerade och kommunicerade med systemet.

Jag började med att koppla upp knapparna i pull-up konfiguration och testade dessa mot den inbyggda Serial Monitorn. När dessa visade noll vid knapptryck och ett annars gav jag mig på de analoga sensorerna. Samma sak här kopplades dessa till prototypbrädan för att skriva ut sina värden till monitorn.

Efter det börjar härvan med bibliotek, där en temperatursensor kopplades in först. Vid implentation av nästkommande komponenter underlättar det att använda de redan utvecklade biblioteken, till denna användes ett bibliotek för DHT11 (DHT11.h).

Realtidsklockan som används, DS3132, kommunicerar likt LCD-skärmen via I^2C och kan då kopplas på samma slinga på prototypbrädan. Här används ett bibliotek för realtidsklockan (DS3231.h) och ett för skärmen (LiquidCrystal_I2C.h). I^2C komponter behöver även biblioteket Wire.h för att kunna kommunicera korrekt.

Senare i projektet byttes två svaga LED-lampor ut mot en ljusstark LED-slinga från Adafruit. För att smidigt koda till denna används Adafruit_DotStar.h. Här har jag implementerat 7 stycken färger i hexadecimal form, dessa finns definierade i functions.h.

IR-sensorn sköter sig själv i princip, när den känner rörelse sätts signal-pinnen till hög och man får en etta utskriven i monitorn. Det sitter två rattar på baksidan av kretskortet som används för att ställa in tiden mellan avläsningar samt känsligheten.

När alla komponenter har sin baskod och är kopplat rätt kan man börja lägga samman dessa i en fil och få de att fungera tillsammans.

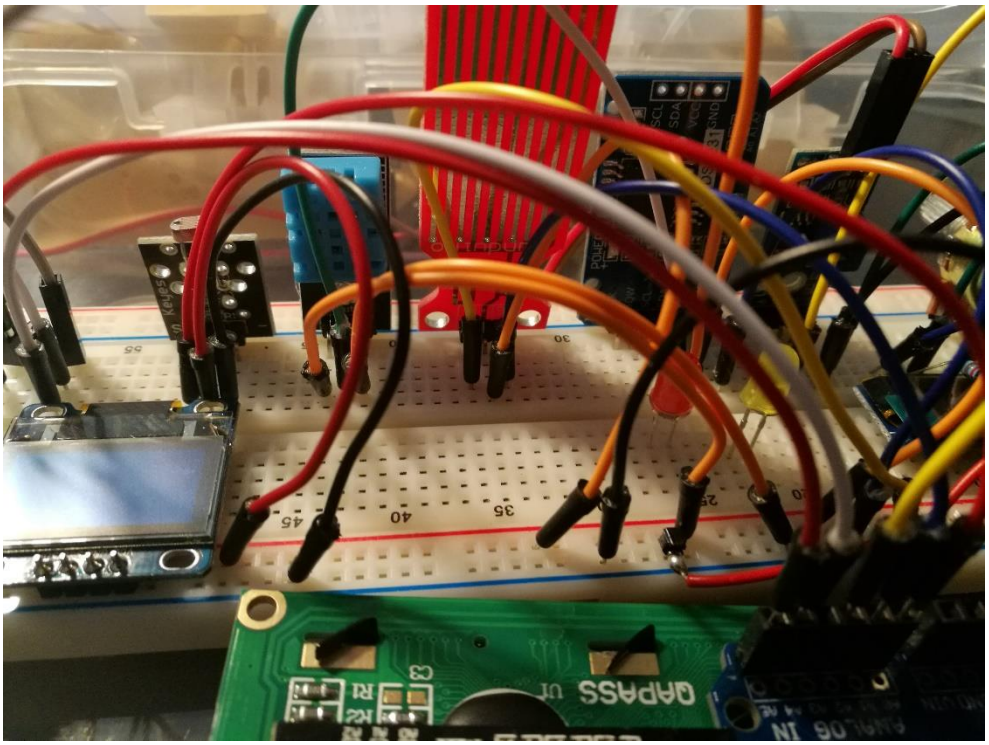


Bild 3.3.1

När allt lirar ihop är det dags för det tidsödande arbetet med att löda alla kontakter för samtliga komponenter. Jag valde att koppla alla komponeter med kontakter för att kunna byta ut dom, alternativt använda dom till andra projekt i framtiden, istället för att löda dit dom direkt. Jag använde mig av ett så kallat experimentkort med banor för att spara lite på tennet och kablar. Jag placerade ut mina kontakter som jag ville ha dom först (Figur 3.3.3), därefter drog jag plus och minus från alla kontakter till den banan med plus och minus från Arduinon. Jag försökte hålla en standard där minus alltid satt på en kant, och plus bredvid den, för att sedan koppla signal och kommunikations pinnarna bredvid dessa. På detta sätt blev det lätt vid montering, där jag bara behövde ta hänsyn till att den svarta kabeln hamnade på kanten där jord var placerad.

På knapparnas plus-kabel lödde jag fast en 10k resistor direkt och fasade ihop denna med signalkabeln under en krympslang.

Alla komponter går att frigöra från systemet helt förutom en av temperatursensorerna och jordfukthetsmätaren som är tänkta att användas antingen utomhus eller i en växtkruka där kyla och fukt är ett faktum (Figur 3.3.4).

Här har jag lött direkt på komponenterna och täckt dessa med krympslang, och sedan använt limpistol för att skydda de nakna lödningarna på kretskorten.

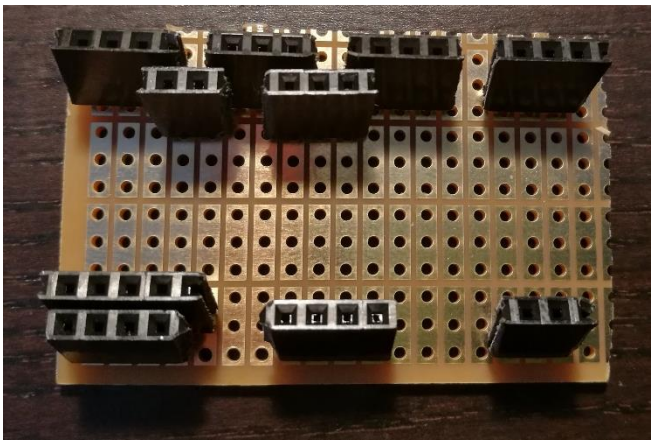


Bild 3.3.2

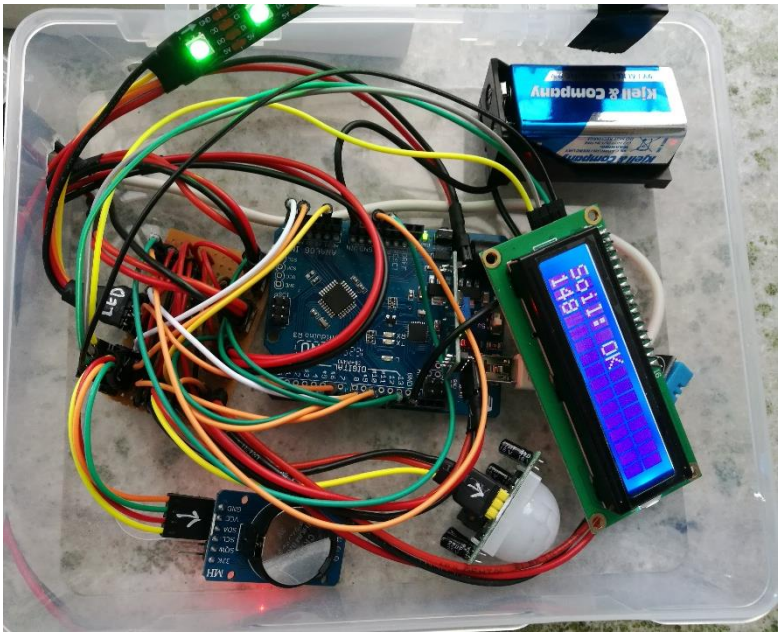


Bild 3.3.3

Bild 3.3.4

3.4 Programmering

Om man som användare skapar ett arduino-projekt i Arduino IDE och klistrar in den kod som finns bifogad under WS20_arduino.ino, skapar tre nya flikar och klistrar in resterande kod i dessa och döper flikarna till rubrikerna som ges nedan, så bör ditt utvecklingskort fungera på samma vis som min. Förutsatt att allt kopplats enligt kretsschemat som givits ovan bör du ha en lika komplett väderstation som jag.

Det finns givetvis ett färdigt Arduino-projekt i samma mapp som du hittade detta dokument, för att hjälpa dig komma igång snabbare.

Koden har en central variabel, som jag döpt till ”state” (figur 3.4.1), vars uppgift är att hålla koll på vilken sida man befinner sig på. Eftersom det finns en undermeny men inställningar kändes detta som ett givet val.

```
delay(200);
return state;
```

Figur 3.4.1

Jag har annars modulerat koden så gott jag kunnat för att göra den så lättförståelig som möjligt. Jag hade gärna flyttat ut samtliga funktioner som innehåller lcd, led, rtc eller dht anrop men dessa verkar vara fast i huvudfilen på grund av att initsieringen sker där.

4. C-kod

4.1 WS20_arduino.ino :

```
#include <LiquidCrystal_I2C.h>

#include <DS3231.h>

#include <DHT.h>

#include <Arduino.h>

#include <Wire.h>

#include ”functions.h”

#include <Adafruit_DotStar.h>

#include <SPI.h>


Adafruit_DotStar LED = Adafruit_DotStar(LED_NUMPIXELS,

LED_DATAPIN, LED_CLOCKPIN, DOTSTAR_BRG);

DHT dht_indoors(DHTPIN_INDOORS, DHTTYPE);

DHT dht_outdoors(DHTPIN_OUTDOORS, DHTTYPE);

DS3231  rtc(SDA, SCL);

// ADRESS OF LCD, DS, RW, E, D0, D1, D2, D3, D4

// This works for the LCD that was used in this project

LiquidCrystal_I2C lcd(LCD_ADDRESS, 2, 1, 0, 4, 5, 6, 7);

void setup()

{

        Serial.begin(9600);

        dht_indoors.begin();

        dht_outdoors.begin();

        rtc.begin();

        Wire.begin();

        lcd.begin (16,2); //  My LCD is 16x2 characters
```

```
        lcd.setBacklightPin(3, POSITIVE);
        LED.begin(); // Initialize pins for output
        LED.show();  // Turn all LEDs off ASAP


  /*
Uncomment to set date and time
14 seconds to upload
  */
  //rtc.setDOW(FRIDAY);         // Set Day-of-Week to SUNDAY
  //rtc.setTime(11, 49, 30);     // Set the time to 12:00:00 (24hr format)
  //rtc.setDate(6, 4, 2017);    // Set the date to 16-03-2017
Serial.println("60 second calibration started");
delay(600);
Serial.println("Calibration finished");
}


/*
int state is for the navigation that has been implemented into the project
If the user is on page 2, state will be set to 2, and will show the content on that page.
*
ex. if state = 1, print_clock will run and output time and date to LCD
*/


void print_Clock(int state)
{   // Print date and time to large LCD screen
  if (state == 1) {
      lcd.setCursor (0,0);
```

```
      lcd.print(rtc.getDateStr(FORMAT_LONG));
      lcd.print(" ");
      lcd.print(rtc.getDOWStr(FORMAT_SHORT));
      lcd.setCursor (0,1);
      lcd.print(rtc.getTimeStr());
      lcd.print(" ");
  }
}

int read_temp(int state)
{   // Print temp to large LCD screen
  if (state == 2) {
      int temp_in = dht_indoors.readTemperature();
      int temp_out = dht_outdoors.readTemperature();
      lcd.setCursor (0,0);
      lcd.print(temp_in);
      lcd.print("%C, ");
      lcd.setCursor (0,1);
      lcd.print(temp_out);
      lcd.print("%C, ");
      return temp_in;
  }
}

int read_humidity(int state)
{   // Print humidity to large LCD screen
  if (state == 2) {
```

```
      int humidity_in = dht_indoors.readHumidity();
      int humidity_out = dht_outdoors.readHumidity();
      lcd.setCursor (6,0);
      lcd.print(humidity_in);
      lcd.print("% RH in");

      lcd.setCursor (6,1);
      lcd.print(humidity_out);
      lcd.print("% RH out");
      return humidity_in;
  }
}

void light_LED(int LEDs)
{
  // Lights up the LED-strip in the correct colour.
  // Call 1 for RED, 2 for yellow and 3 for black.
  for (int x = 0; x < 6; x++ ){
      LED.setBrightness(100);
      if (LEDs == 1){
LED.setPixelColor(x, RED); // 'On' pixel at head
LED.setBrightness(255);
      }
else if (LEDs == 2)
LED.setPixelColor(x, YELLOW); // 'On' pixel at head
else if (LEDs == 3)
LED.setPixelColor(x, BLACK); // 'On' pixel at head
```

```
LED.show(); // Refresh strip
  }
}

// Lag-free display that prints level and value without any lcd.clear() functions
void print_soil_LCD(int state, int current_soil, int critical_soil, int warning_soil)
{
// Light up RED LED if very dry
if (current_soil >= critical_soil) {
light_LED(1);
if (state == 3) {
lcd.setCursor (0,0);
lcd.print(”Soil: CRITICAL”);
lcd.setCursor (0,1);
lcd.print(current_soil);
      }
  }
//Light up YELLOW LED if  dry
else if (current_soil >= warning_soil && current_soil < critical_soil) {
light_LED(2);
if (state == 3) {
lcd.setCursor (0,0);
lcd.print(”Soil: WARNING ”);
lcd.setCursor (0,1);
//Prints current soil value
lcd.print(current_soil);
if(current_soil < 1000) {
```

```
lcd.setCursor (3,1);
lcd.print(" ");
        }
    }
  }
// If moister than warning level, it says ok, and no LED
else if (current_soil < warning_soil) {
light_LED(3);
if (state == 3) {
lcd.setCursor (0,0);
lcd.print("Soil: OK      ");
lcd.setCursor (0,1);
//Prints current soil value
lcd.print(current_soil);
if(current_soil < 1000) {
lcd.setCursor (3,1);
lcd.print(" ");
        }
    }
  }
}

// Navigation system that let´s the user choose which plant to adjust the critical level to.
// Set plant and levels in functions.h
int set_critical_soil(int state)
{
if (state == 71) {
```

```
lcd.setCursor(0,0);
lcd.print(PLANT_ONE);
return PLANT_ONE_CRITICAL;
  }
else if (state == 72) {
lcd.setCursor(0,0);
lcd.print(PLANT_TWO);
return PLANT_TWO_CRITICAL;
   }
else if (state == 73) {
lcd.setCursor(0,0);
lcd.print(PLANT_THREE);
return PLANT_THREE_CRITICAL;
    }
else if (state == 74) {
lcd.setCursor(0,0);
lcd.print(PLANT_FOUR);
return PLANT_FOUR_CRITICAL;
     }
}

// Navigation system that let´s the user choose which plant to adjust the critical level to.
int set_warning_soil(int state)
{
if (state == 71) {
return PLANT_ONE_WARNING;
  }
```

```
else if (state == 72) {
return PLANT_TWO_WARNING;
    }
else if (state == 73) {
return PLANT_THREE_WARNING;
    }
else if (state == 74) {
return PLANT_FOUR_WARNING;
    }
}

// Print the UI for the settings menu, accessed by pressing both buttons
void print_setting_LCD(int state)
{
if (state == 5) {
lcd.setCursor(0,0);
lcd.print(”Settings”);
lcd.setCursor(0,1);
lcd.print(”Double for back”);
   }
if (state == 6) {
lcd.setCursor(0,0);
lcd.print(”Set clock, date”);
lcd.setCursor(0,1);
lcd.print(”Not until v2.0”);
   }
if (state == 7) {
```

```
lcd.setCursor(0,0);
lcd.print(”Choose plant”);
  }
}

int main_function(int state, int critical_soil, int warning_soil)
{
if (digitalRead(RIGHT_BUTTON) == 0) {
lcd.clear();
state = right_button_pressed(state);
  }
if (digitalRead(LEFT_BUTTON) == 0) {
lcd.clear();
state = left_button_pressed(state);
  }
if (digitalRead(LEFT_BUTTON)==0 &&(digitalRead(RIGHT_BUTTON)==0)) {
state = both_buttons_pressed(state);
  }
// If something goes wrong with the statements above, they will return -1
// Giving the user a hint that the system needs a reboot.
if (state < 1) {
state_fails(state);
lcd.setCursor(0,0);
lcd.print(”Contact someone smart”);
  }

print_Clock(state);
```

```
int temp = read_temp(state);

int humidity = read_humidity(state);

int soil = read_soil();

print_soil_LCD(state, soil, critical_soil, warning_soil);

print_setting_LCD(state);

//  Uncomment to print all valuable numbers to Serial monitor //print_to_serial(state, soil, temp,
humidity, critical_soil, warning_soil);

delay(200);

return state;

}


void loop()

{

int value = digitalRead(IR_SENSOR);

Serial.println(value);

delay(200);

static int critical_soil = 1000;

static int warning_soil = 300;

// If the IR_sensor senses movement, it will run the main_function

// which contains everything about the LCD.

if (value == 1){

lcd.setBacklight(HIGH);

// 100 laps equals 23 seconds of watching the screen

for (int timer = 0; timer < 100; timer++){

static int state = 1;

state = main_function(state, critical_soil, warning_soil);

if (state > 70) {

critical_soil = set_critical_soil(state);
```

```
warning_soil = set_warning_soil(state);
          }
       }
    }
// If no movement, only the soil sensor will update and show via the LED-strip.
else{
int soil = read_soil();
print_soil_LCD(1, soil, critical_soil, warning_soil);
lcd.clear();
lcd.setBacklight(LOW);
   }
}
```

4.2 functions.h

```
#define DHTPIN_INDOORS 8

#define DHTPIN_OUTDOORS 5

#define DHTTYPE DHT11

#define POSITIVE 0

#define LCD_ADDRESS 0x3F

#define RIGHT_BUTTON 12

#define LEFT_BUTTON 11

#define YELLOW_LED 2

#define RED_LED 3

#define ON HIGH

#define OFF LOW

#define LED_NUMPIXELS 4 // Number of LEDs in strip

#define LED_DATAPIN    7

#define LED_CLOCKPIN   6

#define IR_SENSOR 9

#define SOIL_SENSOR A2


// Colors in hex

#define BLACK 0x000000

#define WHITE 0xFFFFFF

#define RED 0x00FF00

#define BLUE 0x0000FF

#define GREEN 0xFF0000

#define YELLOW 0xFFFF00

#define PINK 0x00FFFF
```

```
// Plant names and levels
#define PLANT_ONE ”Kaktus”
#define PLANT_ONE_WARNING 700
#define PLANT_ONE_CRITICAL 850
#define PLANT_TWO ”Basilika”
#define PLANT_TWO_WARNING 500
#define PLANT_TWO_CRITICAL 700
#define PLANT_THREE ”Timjan”
#define PLANT_THREE_WARNING 200
#define PLANT_THREE_CRITICAL 800
#define PLANT_FOUR ”Valfri”
#define PLANT_FOUR_WARNING 900
#define PLANT_FOUR_CRITICAL 1080

int right_button_pressed(int state);
int left_button_pressed(int state);
int both_buttons_pressed(int state);
void state_fails(int state);
int read_soil();
void print_to_serial(int state, int soil, int temp,
int humidity, int critical_soil, int warning_soil);
```

4.3 functions.cpp

```
#include ”functions.h”

#include <Arduino.h>


/*

Simple functions that reads the value of all analog sensors and returns the value

Also the troubleshooting functions is here

*/


int read_soil()

{

int soil = analogRead(SOIL_SENSOR);

return soil;

}


void print_to_serial(int state, int soil, int temp, int humidity,

int critical_soil, int warning_soil)

{

Serial.print(”State: ”);

Serial.println(state);

Serial.print(”Temp: ”);

Serial.println(temp);

Serial.print(”Humidity: ”);

Serial.println(humidity);

Serial.print(”Soil: ”);

Serial.println(soil);

Serial.print(”Critical Soil level: ”);
```

```
Serial.println(critical_soil);

Serial.print(”Wardning Soil level: ”);

Serial.println(warning_soil);

Serial.println(” ”);

Serial.println(” ---------- ”);

delay(2000);

}
```

4.4 navigation.cpp

```
#include ”functions.h”

#include <Arduino.h>


/*

Contains all the steps that is needed to provide a secure navigation via two buttons.

1-3 contains main level that shows values.

5-7 contains settings menu

61-65 is supposed to contain change of time and date

71-74 contains the ability to change plant.

*/


int right_button_pressed(int state)

{

    if (state == 1) return 2;

    else if (state == 2) return 3;

    else if (state == 3) return 1;

    else if (state == 5) return 6;

    else if (state == 6) return 7;

    else if (state == 7) return 5;

    else if (state == 61) return 62;

    else if (state == 62) return 63;

    else if (state == 63) return 64;

    else if (state == 64) return 65;

    else if (state == 65) return 61;

    else if (state == 71) return 72;

    else if (state == 72) return 73;
```

```
    else if (state == 73) return 74;
    else if (state == 74) return 71;
    else return (-1);
}
int left_button_pressed(int state)
{
if (state == 1) return 3;
else if (state == 2) return 1;
else if (state == 3) return 2;
else if (state == 5) return 7;
else if (state == 6) return 5;
else if (state == 7) return 6;
else if (state == 61) return 65;
else if (state == 62) return 61;
else if (state == 63) return 62;
else if (state == 64) return 63;
else if (state == 65) return 64;
else if (state == 71) return 74;
else if (state == 72) return 71;
else if (state == 73) return 72;
else if (state == 74) return 73;
else return (-1);
}

int both_buttons_pressed(int state)
{
if (state <= 3) return 5;
```

```
else if (state == 6) return 61;

else if (state == 7) return 71;

else if (state == 61) return 611;

else if (state == 62) return 621;

else if (state >= 71) return 1;

else if (state > 3) return 1;

}


// If state is -1 this will run until reset

void state_fails(int state)

{

Serial.println(”Something went wrong, state is out of bounds: ”);

Serial.println(state);

}
```

5. Diskussion

5.1 Hårdvara

Det fanns från början en tanke att bara använda Arduino för protypandet, för att sedan gå över till en ATMega328p och använda mig av Atmel Studio för programmerandet. Det kräver dock att koden skrivs i lågnivå-C, vilket innebär att inga av de bibliotek som använts i detta projektet hade fungerat. Jag hade helt enkelt fått granska samtliga bibliotek och gjort om dom till att passa mitt projekt skrivet i C. En sådan konvertering var inget jag varken hade tid eller lust med i detta projektet. Det är svårt att läsa in analoga värden till en MCU, vilket var lite av tanken med väderstationen.

Jag hade gärna bytt ut DHT11 mot DHT22 för noggrannare mätning och för ett större intervall. DHT22 kostar dock mångdubbelt mer än DHT11 i Sverige, men en Ebay beställning blev lagd men hann inte fram innan färdigställande av denna produkt.

Från början hade jag en ljussensor monterad som skulle släcka skärmen vid dagsljus och tända den på kvällen. Problemet med den var att skärmen blev helt oläslig när den var släckt. Hade man använt en skärm utan I^2C hade man kunnat sätta skärmens bakgrundsbelysning som analog output och reglerat den på så vis. Den skärmen jag använt dock har en påbyggnadsmodul för konvertering till I^2C vilken låser skärmens ljusstyrka till antingen max eller inget.

Jag gick då till Kjell och köpte en rörelsedetektor och monterade den så att den har fri sikt över rummet. När sensorn känner rörelse skickar den en etta och programmet kickar igång skärmen och börja läsa av temperatur medmera. Även när ingen rörelse noteras läser systemet av jordfuktssensorn och justerar LED-slingans färg efter fuktigheten i krukan.

Jag valde att byta ut Arduino Uno mot Nano helt enkelt för att spara plats. Uno är smidig att använda vid protoypande tack vare honkontakterna och den stora och tydliga layouten. Vid färdig produkt däremot finns ingen anledning att inte gå över till en Nano då de hårdvarumässigt är precis likadana, medan storleken mellan de är enorm.

Knapparna som används är helt vanliga tvåvägs brytare som helt enkelt skapar en sluten krets när knappen trycks ner. Jag valde dessa arkadknappar på grund av känsla och utseende, det vill säga ett personligt val och kan bytas ut mot vilken tvåvägsbrytare som helst.

LED-listen är en överbliven komponent från ett annat projekt som gjorts i klassen. Listen är av hög kvalitet och har bra mjukvarustöd i form av bibliotek. Den är ljusstark och lätt att koppla in och programmera till. Nackdelen med denna komponent är att det är relativt dyr, men kommer förhoppningsvis hålla längre än Ebay´s dito för en bråkdel av priset. En väderstation ska gärna hålla länge, det är inte en förbrukningsvara.

Jag har nått upp till de allra flesta av de delmålen jag satte upp innan projektets start, dock finns några kvar. Att koppla upp stationen med WiFi var ett av de mål jag inte nådde till, dock var detta ett distansmål som jag enbart hade tagit mig an om resten av projektet hade gått för lätt, vilket det inte har gjort.

Att använda en liten skärm för att visa vart i navigeringen man befinner sig var också ett delmål, men inget som jag lagt någon direkt tid på. Jag hade en I^2C OLED skärm som hade passat perfekt till detta projekt men den hade ingen given adress (som alla I^2C komponenter kräver) så jag kunde inte använda den.

Ett mål jag gärna hade nått dock var att göra systemet batteriförsörjande. Innan jag kopplade in IR-sensorn och koden stod och loopade hela tiden med skärmen tänd höll sig stationen igång ungefär en timme på en nytt 9V batteri. Efter att IR-sensorn kopplats in höll sig systemet igång närmare 4-5 timmar på ytterligare ett nytt 9V batteri. Min implementation av sensorn gjorde alltså systemet 400–500% mer energi effektivt, men inte i närheten av vad som krävs av en sådan produkt.

Resultatet av experimentet innebär alltså att stationen måste vara inkopplad till strömkälla konstant. Detta sker via Nanons Mini-USB port till en nätadapter.

En framtida uppgift är helt klart att koppla detta på en MCU istället, som har funktionen att den kan gå i sömnläge, och på så vis i princip inte belasta batteriet någonting när ingen är i rummet.

5.2 Mjukvara

Den mesta delen av koden är skriven i texteditorn Atom för att sedan kompileras och skrivas in på Arduinon genom Arduinos utvecklingsmiljö. Atom är ett lättvikts program som gör översikt och färgkodning på ett mycket bättre sätt än Arduinos IDE. Skribenten av denna dokumentation rekomenderar starkt att överväga detta sätt att skriva kod på (https://atom.io/).

När det gäller mjukvaran finns där utrymme för utbyggnad, och troligtvis även finslipning. I det stora hela känns produkten stabil och säker att använda. Navigationssystemet som jag skapat verkar fungera som det ska.

När man går in i settings finns där en sida för att kunna ställa tid och datum. Jag gav mig på den funktionen en dag, men det blev snabbt stort och gjorde inte riktigt som jag ville. Med hjälp av bara två knappar och realtidsklockan kan man skapa denna koden och få den att fungera i seriella monitorn, och sedan implementera den i väderstationen. Ett bra uppdrag för nästkommande studenter som behöver få in känsla för hur variabler fungerar och hur de kan användas i redan befintliga bibliotek. Uppdaterar jag mjukvaran i väderstationen någon gång lär jag implementera möjligheten att göra just detta. Det är ändå en funktion som inte kräver några hårdvarumodifikationer utan är helt och hållet möjligt att göra med enbart mjukvara och USB-porten som finns tillgänglig.

I funktionen main_function ligger det som tidigare låg i loopen innan rörelsedetektorn implementerades. För att göra den processen med överskådlig valde jag att lägga den delen i en extern funktion som kallas på när rörelse känns av. Personligen föredrar jag så lite rader som möjligt i varje if/for sats, och detta är ett sätt att för mig göra koden mer lättläst. Det kanske kan vara svårare att sätta sig in i koden vid första blick, men när man satt sig in i den anser jag att det är lättare att modifiera funktioner var för sig.

6. Bilagor

Datablad för DHT11

Datablad för DS3231

Datablad för Luxorparts Rörelsedetektor

Datablad för Arduino Uno

Datablad för Arduino Nano

Datablad för 16x2 LCD display

Datablad för jordfuktsensor *Den jag använt har även en analog output, annars samma*

Datablad för Adafruit DotStar LED-slinga