

# Teknisk rapport

---

**Gruppe Syvende Sans**

IN1060 vår 2023

Gruppemedlemmer: Andreas Klæboe, Christian Fjeld Thorkildsen, Viktoriia Olonova

# **Innholdsfortegnelse**

---

<b>Kapittel 1</b>	
Elektronikken	2
<b>Kapittel 2</b>	
Generell kodelogikk	3
<b>Kapittel 3</b>	
Termistor sensoren og kalibrering	4
<b>Kapittel 4</b>	
Skjermen	8
<b>Kapittel 5</b>	
Knappene	9
<b>Kapittel 6</b>	
Klokken	10
<b>Kapittel 7</b>	
Sending av data fra UNO-en	12
<b>Kapittel 8</b>	
Digital loggføring	13
<b>Kapittel 9</b>	
Video	21
<b>Kildeliste</b>	22

---

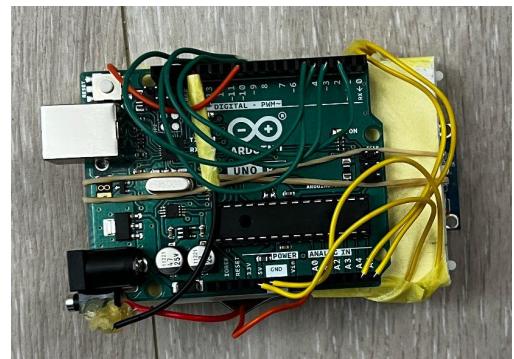
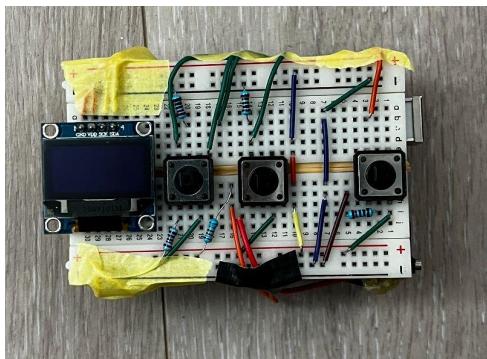
# Kapittel 1 – Elektronikken

Følgende komponenter ble brukt:

- Arduino uno R3
- Arduino ESP32 MH-ET Live Devkit
- Termistor (termometer)
- 2.5mm input jack (input til termistoren)
- Breadboard
- Adafruit ssd1306 128×64 (Skjermen)
- 3x pushbuttons
- 2x blå LED-er
- 5x 220 $\Omega$  resistorer
- 1x 1k $\Omega$  resistor

Vi bruker hovedsakelig en Arduino Uno R3 til å kjøre kode. Den er festet på baksiden av et breadboard for å gjøre artefakten så kompakt som mulig. I tillegg har vi en ESP32. Den bruker vi til å gi Arduinoen en sannidsklokke ved å bruke RX og TX portene<sup>[1]</sup>. ESP-en får også strømforsyning fra Arduinoen. ESP-en sender i tillegg informasjon til et Google-Sheets dokument hver gang en temperatur registreres. Mer om dette står lenger ned.

Skjermen, knappene og LED-ene er alle koblet opp på breadboardet. Knappene og LED-ene har hver sin resistor, mens skjermen har det innebygd. Skjermen er koblet til 5V, GND og to analoge pins. 2.5mm inngangen til termistoren er koblet til GND på breadboardet og én analog pin med en 1k $\Omega$  resistor i mellom. Den ble også festet til Arduinoen (uno R3) med epoxy lim, rett ved strøminngangen til Arduinoen. Alle ledningene ble festet ekstra godt til breadboardet med teip for å unngå at de faller av og for å gjøre artefakten mest mulig kompakt.



Bildene over viser ikke LED-ene eller ESP-en, da disse ble koblet til senere etter breadboardet ble dekket til med papp

## Kapittel 2 – Generell kodelogikk

I loop funksjonen til UNO-en blir det først sjekket om på/av knappen blir trykket. I void setup blir prototypen initialisert som av med `bool strom = false;`. Når på/av knappen blir trykket endres denne variabelen. I loop funksjonen blir det så gjort en if-sjekk som sjekker om den skal være på eller av. Vi skrev to funksjoner; `void paa()` og `void av()` som pakker inn koden som forteller hvordan UNO-en skal oppføre seg ut i fra om den er på eller av.

```
void loop() {
    // sjekker om den skal være på eller av
    strom = sjekkKnapp(paaKnapp, strom);

    // om den er skrudd på
    if(strom) {
        paa();
    }
    else {
        av();
    }
}
```

Når den er av ønsker vi at skjermen skal være tom og begge LED-ene skal være av. Når den er på ønsker vi at skjermen skal vise temperatur og klokkeslett, og at LED-ene skal indikere modus. For å få til dette bruker vi funksjonene:

- `float sjekkTemp()` – Returnerer et flyttall som tilsvarer temperaturen fra termistoren
- `void oppdaterKlokke()` – Ser tidspunktet sendt fra ESP-en og oppdaterer klokke variabelen
- `void displayPrint(String tekst, int storrelse)` – Forteller skjermen hva som skal stå på den.

I paa() funksjonen kjøres funksjonene over, og en if-sjekk av modus gjør at riktig LED vil lyse.

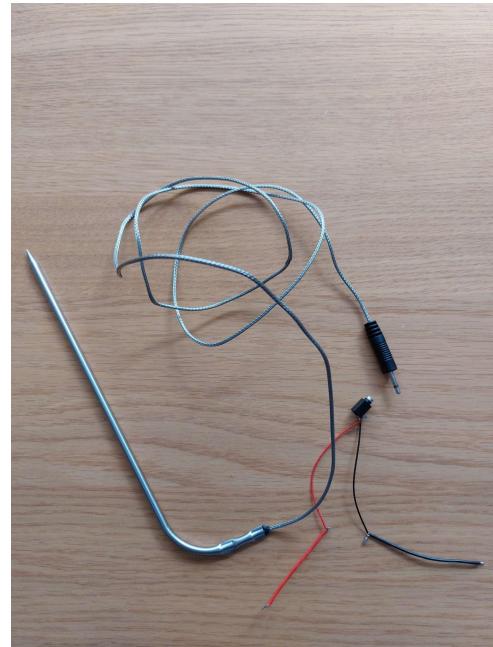
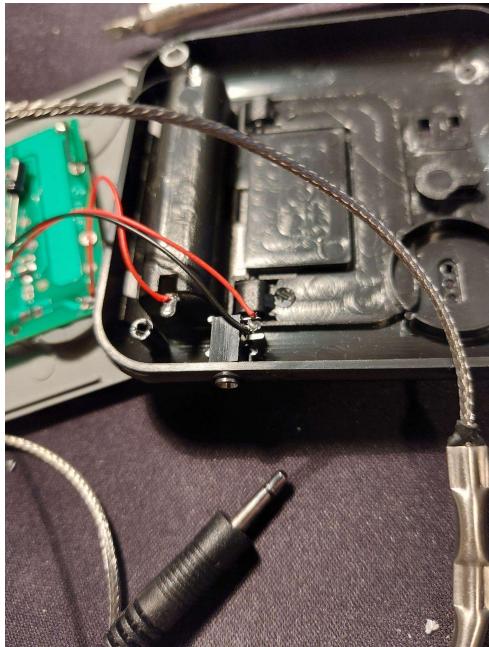
Prototypen vår har tre knapper.

- På/av knappen – skrur prototypen av eller på (slik beskrevet over).
- Modus knappen – endrer modus variabelen.
- Registrer knappen – Sender informasjon (temperatur og modus) til ESP-en med I2C kommunikasjonsprotokoll.

ESP-en, som er koblet til wi-fi, har to oppgaver; laste opp data til et regneark og fortelle UNO-en hva klokken er. ESP-en oppdager når UNO-en sender informasjon med I2C protokoll med funksjonen `void receiveEvent(int bytes)`. som også sender informasjonen videre til et Google sheets regneark. For å sende klokkeslettet til UNO-en bruker den funksjonen `void sendCurrentTime()` som sender gjennom Serial kommunikasjon med TX og RX portene.

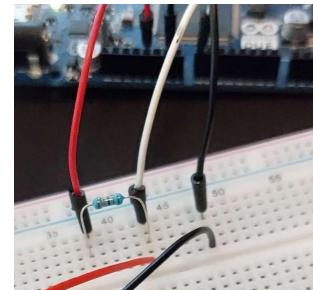
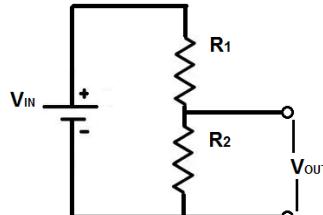
## Kapittel 3 – Termistor sensoren og Kalibrering

Sensoren vi brukte for prosjektet er en termistor tatt ut fra et INVITE 5-volt Elektronisk Steketermometer fra Jernia<sup>[2]</sup> som vi hadde liggende til overs. Vi hadde vurdert flere typer sensorer for å måle temperaturer, men kom fram til at de fleste termistorer solgt som en del av Arduino-sett er treige, kan ikke kobles til og fra prototypen ved behov mens den er slått på, og har heller ingen spiss slik at de kan stikkes inn i matvarer. En termistor er i praksis bare en resistor med et elektrisk-ledende materiale mellom hvert endestykke av resistoren som endrer på den molekulære strukturen basert på hvilken temperatur den befinner seg i. Dette gjør at elektroner beveger seg mer eller mindre fritt ved ulike temperaturer. Så lenge man vet nøyaktig hvor mye strøm som blir sendt til den, kan man regne ut hvilken motstand termistoren har ved å lese strømmen som er til overs. Termistoren som fulgte med produktet fra Jernia kom både med en 2.5mm port for å koble til og fra termistoren, var laget for 5v slik som Arduino UNO kan tilby og var perfekt egnet for vårt formål i utforming.



Etter en del graving på nettet fant vi ut at steketermometeret er produsert under et eget merkevarenavn hos Jernia, og er bygget på en masseprodusert modell. Denne blir solgt med små ulike variasjoner i utseende i flere ulike butikker, men ingen av dem kommer med tekniske detaljer om hvordan kretsen er bygget opp eller hvilken motstand termistoren er basert på ved en gitt temperatur. Det eksisterte heller ingen biblioteker vi kunne inkludere i koden for å automatisk lese temperaturer, så den neste oppgaven ble å finne en måte vi kunne kontinuerlig lese motstanden. Dermed kan vi lage en spenningsdeler<sup>[3]</sup> til Arduinoen for å finne motstanden til termistoren.

$$R_2 = R_1 \cdot \frac{1}{\left( \frac{V_{in}}{V_{out}} - 1 \right)}$$



Vår spenningsdeler vil bruke Arduino UNO sin 5v, GND og den analoge pin A0 som leser inn en verdi vi kan bruke til å regne ut Vout. Vi valgte å bruke en  $1k\Omega$  resistor som R1, og testet den med et multimeter og leste at den er  $999\Omega$ . Dette blir vår referansevariabel, Rref.

```
float Rref = 999;
```

“Vin” er 5, div. 5v fra UNO sin 5v pin. Vout er den vi må regne ut for å finne R2.

```
float Vout;
float Vin = 5;
```

Vi definerer A0 som sensorPin og sensorValue initialiseres som 0.

```
int sensorPin = A0;
int sensorValue = 0;
```

Vi kaller flyttall-variabelen som tilsvarer temperaturen for “T”, og lager en tom string-variabel “temp” som skal brukes til å konvertere temperaturen til en printbar tekst for skjermen.

```
String temp;
float T;
```

For å regne ut Vout må vi først finne ut hvilken analoge verdi som blir lest på A0. Denne blir deretter ganget med Vin, og produktet blir delt på 1023, som er hvor mange ulike verdier vi kan lese (eksklusivt 0) gitt at 5v er høyeste verdi en analog pin vil lese som 1024. Hvis for eksempel A0 gir oss verdien “800” kan vi regne ut at med en forventet maks volt på 5v ganget med 800, delt på antall ulike verdier vi kan lese, er Vout 3.91v.

Nå som vi har Vout, kan vi bruke formelen tidligere hvor vi bruker ohms lov til å regne ut hvilken motstand R2 (i vårt tilfelle termistoren), har ved en gitt temperatur.

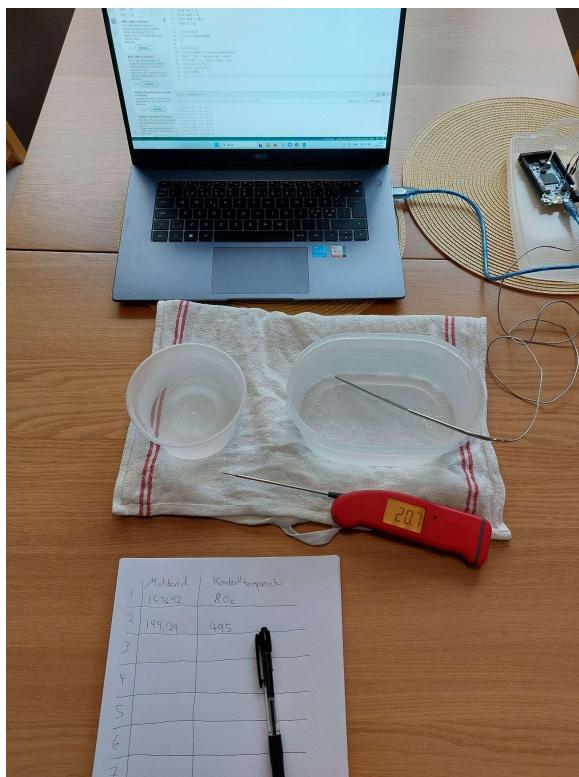
$$Y = 999 \cdot \frac{1}{\left( \frac{5}{\left( \frac{5 \cdot x}{1023} \right)} \right) - 1}$$

$$R_2 = R_1 \cdot \frac{1}{\left( \frac{V_{in}}{V_{out}} - 1 \right)}$$

Her er "x" representert som den analoge verdien, og Y er R2. I kodefilen er dette skrevet som;

```
sensorValue = analogRead(sensorPin);
Vout = (Vin * sensorValue) / 1023;
int res = Rref * (1 / ((Vin / Vout) - 1)); // formel som regner ut resistansen
```

## Kalibrering

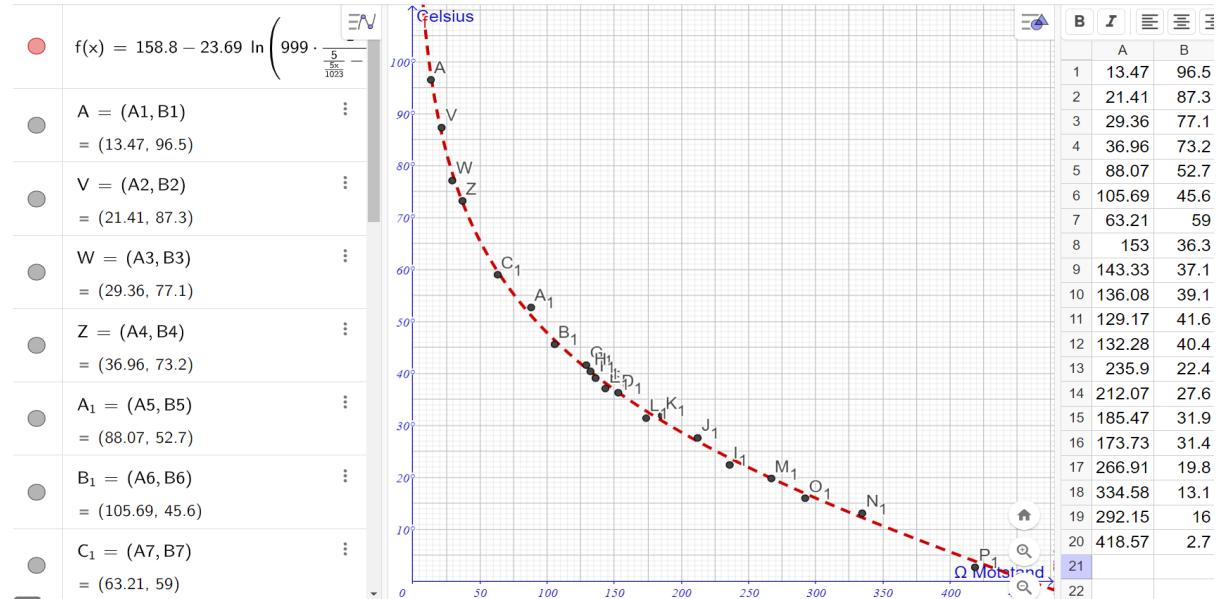


Vi lager variabelen "res" som vi kan bruke til å printe ut motstanden som måles kontinuerlig til Serial. Nå har vi muligheten til å se motstanden endre seg direkte mens vi varmer opp eller kjøler ned termistoren. For å finne korrelasjonen mellom motstand og temperatur kan vi gjøre en rekke tester i et kontrollert vannbad mot kjente temperaturer.

For hver måling lar vi vannet i en spring renne i et halvt minutt for å være sikker på at temperaturen på vannet er den samme. Ved noen av testene ble kokende vann i en termos målt. Deretter setter vi termistoren og et termometer med 1 millimeter avstand fra hverandre i vannbadet i ett minutt. Dette er for å være sikker på at de har stabilisert seg.

Deretter noterer vi i et skjema motstanden som printes i Serial samtidig som vi noterer hvilken temperatur som ble målt. Denne prosessen ble gjentatt for hele prototypen hver gang vi la til ekstra komponenter eller funksjoner, for å være sikker på at termistoren ble kalibrert korrekt når den totale spenningen fra Arduino UNOen endret seg.

Tallene fra skjemaet ble deretter lagt inn i en tabell i GeoGebra<sup>[4]</sup> og konvertert til en liste med punkter. Ved å bruke “**FitLog(Liste1)**”-kommandoen i GeoGebra kunne vi skape en logaritmisk f(x) funksjon i base e som konverterer hvilken tilsvarende temperatur f(x) er lik ved en gitt motstand.



Nå som vi har funksjonen  $f(x) = 158.8 - 23.69 * \ln(x)$ , og vi allerede har laget vår egen funksjon med ohms lov for å finne x, kan vi slå de sammen til én funksjon som regner ut temperatur ved en målt motstand.

The handwritten derivation shows the simplification of the logarithmic expression from the previous slide:

$$f(x) = 158.8 - 23.69 \cdot \ln \left( 999 \cdot \frac{1}{\left( \frac{5}{1023} \right)} - 1 \right)$$

I programmet skrev vi en egen funksjon som returnerer temperaturen som måles.

Merk: I biblioteket `<math.h>` er `log()` i basen “e”, ikke 10. Derfor er `log()` her det samme som `ln()`.

```
float sjekkTemp(){
    sensorValue = analogRead(sensorPin);
    Vout = (Vin * sensorValue) / 1023;
    int res = Rref * (1 / ((Vin / Vout) - 1));
    return 158.8 - 23.69 * log(res);
}
```

Vi kan nå skrive `T = sjekkTemp();` for å sette variabelen T til å tilsvare temperaturen som blir målt.

## Kapittel 4 – Skjermen

Skjermen vi brukte er en Adafruit SSD 1306. Den er på 128 x 64 piksler. For å finne ut av hvordan vi skulle ta den i bruk lette vi på nettet etter en god guide. Vi fant en veldig god en fra electrocredible.com<sup>[5]</sup> som forklarte hvordan man skulle koble den til arduinoen og hvilket bibliotek man måtte laste ned. Den ble koblet til 5V, GND, A4 og A5. Koden under er koden som er nødvendig for å starte å ta i bruk skjermen:

```
// For skjerm
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define SCREEN_WIDTH 128      // OLED display width
#define SCREEN_HEIGHT 64       // OLED display height
#define OLED_RESET -1         // Reset pin number, -1 means Arduino reset pin
#define SCREEN_ADDRESS 0x3C // Address may be either 3C or 3d
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

void setup() {
    // sjekker om skjermen er koblet opp riktig
    if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
        Serial.println(F("SSD1306 allocation failed"));
        for(;;); // Looper forevig om noe er galt, programmet stopper
    }
}
```

Kode som ikke er relevant for skjermen er ikke tatt med i koden over.

Artikkelen inkluderte også et enkelt “Hello world!” program. Vi modifiserte dette litt og lagde en funksjon som inneholder koden som trengs for å vise tekst på skjermen. Den ser slik ut:

```
void displayPrint (String tekst, int storrelse){
    display.clearDisplay();           // tømmer skjermen
    display.setTextSize(storrelse);   // setter riktig skriftstørrelse
    display.setTextColor(SSD1306_WHITE);
    display.setCursor(0,0);          // starter fra øverst til venstre
    display.println(tekst);          // skriver ut tekst
    display.display();               // starter å vise
}
```

Denne funksjonen blir brukt hver gang skjermen skal vise noe nytt.

## Kapittel 5 – Knappene

Vi brukte til sammen tre knapper; på/av, modus og registrer. Vi ga alle knappene hver sin setup med `int knapp = pin;` og `pinMode(knapp, INPUT_PULLUP);`.

På/av knappen og modus knappen har hver sin boolske verdi som de skal endre på. For å unngå å kopiere kode, lagde vi en felles funksjon som registerer et knappetrykk og endrer den relevante boolske variabelen. Den funksjonen ser slik ut:

```
bool sjekkKnapp(int knapp, bool variabel){
    int trykk = digitalRead(knapp);

    if(trykk == HIGH){           // dersom knappen blir trykket
        while(trykk == HIGH){   // venter til knappen blir sluppet
            trykk = digitalRead(knapp);
            delay(50);          // for å unngå debounce
        }
        return !variabel;
    }
    return variabel;
}
```

For å benytte samme funksjon til registrer-knappen lagde vi en midlertidig boolsk variabel vi kunne sende inn i sjekkKnapp() funksjonen. Dette puttet vi inn i en ny funksjon som sjekker om registrer-knappen blir trykket. Den funksjonen ser slik ut:

```
void sjekkRegistrer(){
    bool trykk = true;

    // om registrer knappen blir trykket
    if (trykk != sjekkKnapp(registrerKnapp, trykk) ){
        registrer();
    }
}
```

## Kapittel 6 – Klokken

I tillegg til å vise temperatur, ønsket vi at skjermen også skulle vise klokkeslettet. UNO-en har ikke tilgang på internett, så til å hente klokkeslettet brukte vi ESP-en. Hvordan vi skulle gi ESP-en en klokke var vi usikre på, så vi ga ChatGPT<sup>[6]</sup> følgende prompt: “write a simple function for ESP32 that prints the time. I want the clock to be 24h and only show hours and minutes”. Den ga oss en god kode som fungerte slik den skulle. Vi modifiserte den litt, slik at den skulle fungere til våre formål. Følgende kode måtte være med i setup:

```
#include <NTPClient.h>
#include <WiFiUdp.h>
#include <WiFiUdp.h>
const char* ssid = nettverk-SSID;
const char* password = nettverk-passord;

WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP);

void setup() {
    Serial.begin(9600);

    // kobler seg på nett
    WiFi.mode(WIFI_STA); //Optional
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(100);
    }

    // initialiserer og synkroniserer klokke
    timeClient.begin();
    timeClient.setTimeOffset(2 * 3600); // setter riktig tidssone
    timeClient.forceUpdate();
}
```

*Kode som ikke er relevant til å vise klokke er ikke tatt med i koden over*

Videre lagde vi en funksjon som skulle printe ut klokkeslettet til Serial. Den ser slik ut:

```
void sendCurrentTime() {
    timeClient.update();

    // Henter tid
    int hours = timeClient.getHours();
    int minutes = timeClient.getMinutes();

    // Printer tiden
    Serial.printf("%02d:%02d\n", hours, minutes);
}
```

Funksjonen kjøres i void loop() på ESP-en.

Videre ønsket vi at UNO-en skulle plukke opp det som ble printet av ESP-en. Vi koblet da TX porten til ESP-en til RX porten til UNO-en. UNO-en vil da motta alt som blir printet av ESP-en. Vi lagde så en funksjon til UNO-en som setter en String klokke variabel til å være det som blir printet av ESP-en. Også til denne funksjonen baserte vi på kode vi fikk fra ChatGPT med prompten “write a simple function for Arduino UNO that stores a string from the RX port in a variable”. Vi oppdaget at ESP-en av og til kunne sende to klokkeslett samtidig. Vi antar at dette er fordi TX og RX portene ikke er veldig stabile. Vi løste det med en if sjekk, som passer på at Stringen som blir sendt ikke er lengre en 5 chars. Funksjonen ser slik ut:

```
void oppdaterKlokke(){
    // ser om data blir sendt
    if (Serial.available()) {
        // Lager en variabel tilsvarer stringen som blir sendt
        String mottatt = Serial.readStringUntil('\n');

        int lengde = mottatt.length();
        if(lengde == 5){
            klokke = mottatt;
        }
    }
}
```

## Kapittel 7 – Sending av data fra UNO-en

Når UNO-en er skrudd på vil den konstant sjekke hva temperaturen i termistoren er, og “lytte” til om registrerings-knappen blir trykket. Når knappen blir trykket, sendes temperatur og modus over til ESP-en med I2C kommunikasjonsprotokoll. ESP-en er koblet til UNO-ens 5V og GND for å få strøm, og SDA- og SCL-portene for å kommunisere. For å bruke I2C protokoll tar vi i bruk *Wire.h*<sup>[7]</sup> biblioteket.

For å starte denne kommunikasjonen bruker vi følgende kode i UNO-en:

```
#include <Wire.h>
void setup() {
    // Starts I2C communication with the wifi-connected ESP32
    Wire.begin();
}
```

*Kode som ikke er relevant for kommunikasjonen er ikke tatt med i koden over*

Vi skrev en funksjon som sender den relevante dataen til ESP-en når registrer-knappen blir trykket:

```
void registrer(){
    // Starter tilkobling til ESP
    Wire.beginTransmission(9); // transmit to device #9

    // sjekker modus og sender riktig data
    if(modus) {
        char signal = 'S';

        // sender først modus som signal
        Wire.write((byte *) &signal, sizeof (signal));

        // sender så temperatur
        Wire.write((byte *) &T, sizeof (T));
    }

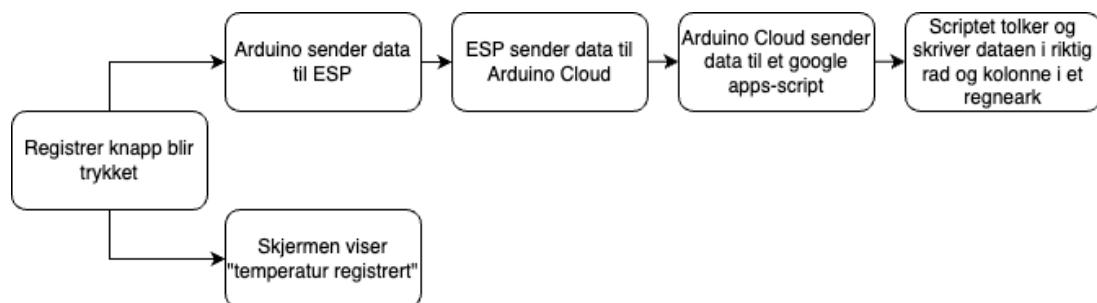
    else {
        char signal = 'V';
        Wire.write((byte *) &signal, sizeof (signal));
        Wire.write((byte *) &T, sizeof (T)); // sender temperatur
    }
    Wire.endTransmission();
}
```

*Kode som ikke er relevant for kommunikasjonen er ikke tatt med i koden over*

## Kapittel 8 – Digital loggføring

Siden Arduino Uno ikke har WiFi tilgang, kan vi ikke bruke den direkte til å sende eller loggføre data til et regneark. Til dette formålet bruker vi en MH-ET Live ESP32 DevKit modul, som vi kobler opp via WiFi mot Arduino Cloud. Men først må måledataene sendes over fra Uno til ESP32. Dette gjøres over kabel ved hjelp av en standard kommunikasjonsprotokoll - I2C implementert i Arduino-biblioteket *Wire*<sup>[7]</sup> (slik beskrevet i kapittel 7). Vi sender over dataene byte for byte, og ESP32 setter dem sammen igjen til tall. Før hver variabel blir sendt, sender vi først en char som beskriver typen variabel som sendes over (steke eller vannbad). Tallene vi får inn blir så omgjort til JSON-strenger og sendt videre til Arduino Cloud via WiFi automatisk. Dette innebærer at man trenger en stabil WiFi tilkobling i kjøttdisken og at innloggingsdetaljene hardkodes i ESP32. Dersom innloggingsdetaljene endres, må også koden i ESP32 endres.

### Sekvensdiagram for digital loggføring:



### *ESP\_code.ino*

Under finner man koden for å sette opp ESPen. I begynnelsen kobler WiFi-biblioteket seg på nett og muliggjør klokke-funksjonaliteten. Etterpå kobles ESP til Arduino IOT Cloud, med et høyt nivå av logging. Til slutt starter man en I2C tilkobling i slavemodus, med samme device-kode (9) som blir referert til av UNOen. I2C kobles opp mot callback-funksjonen *receiveEvent()*.

```
#include <WiFi.h>
#include <NTPClient.h>
#include <WiFiUdp.h>
#include "Wire.h"
#include "thingProperties.h"
const char* ssid = "***";           // brukernavn på WiFi
const char* password = "***";       // Passordet til WiFi

WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP);

void setup() {
    Serial.begin(9600);
```

```

// kobler seg på nett
WiFi.mode(WIFI_STA); //Optional
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
    delay(100);
}

// initialiserer og synkroniserer klokke
timeClient.begin();
timeClient.setTimeOffset(2 * 3600); // setter riktig tidssone
timeClient.forceUpdate();

// Defined in thingProperties.h
initProperties();
// Connect to Arduino IoT Cloud
ArduinoCloud.begin(ArduinoIoTPreferredConnection);

// Print debug info
setDebugMessageLevel(2);
ArduinoCloud.printDebugInfo();

// Start the I2C Bus as Slave on address 9.
// The sender of the data also has this address in its code to communicate
Wire.begin(9);
// Attach a function to trigger when something is received.
Wire.onReceive(receiveEvent);
}

```

Funksjonen `sendCurrentTime()` leser klokkeslett (fra internett) og sender nåværende tid til UNO ved hjelp av UART-protokollen og Serial-porten, formatert som en streng bestående av timer og minutter. Standardfunksjonen `loop()` gjør ingenting annet enn å sikre at tiden sendes til UNO-en kun én gang i sekundet ved hjelp av en `delay`.

```

void loop() {
    ArduinoCloud.update();
    sendCurrentTime();
    // Here we just wait, because this increases the stability of this loop
    delay(1000);
}
void sendCurrentTime() {
    timeClient.update();

    // Henter tid
    int hours = timeClient.getHours();
    int minutes = timeClient.getMinutes();

    // Printer tiden
    Serial.printf("%02d:%02d\n", hours, minutes);
}

```

*ReceiveEvent()* funksjonen under kjører hver gang UNO sender data (når register() funksjonen blir kjørt i UNO-en) og starter en runde med kommunikasjon over I2C. I2C stopper midlertidig alle andre prosesser for å la *receiveEvent()* kjøre. Funksjonen leser først inn én bokstav ('s' for steke, 'v' for vannbad) som forteller hva temperaturen måles for (steke eller vannbad), og deretter leser den inn temperaturverdien. Temperaturen blir lest inn som et sett med bytes (ikke bokstaver) tatt direkte fra en binær 32-bit float sendt fra UNO, som deretter blir satt sammen tilbake til en float ved hjelp av pointer casting. Dermed spares mye kommunikasjons- og prosesseringstid sammenliknet med å sende tall som strenger. Funksjonen gjør så om verdiene til JSON, og lagrer dem i en variabel *data*, som synkroniseres med Arduino Cloud. Selve JSON-strengen består alltid av to felter der det ene er temperatur i vannbad og den andre er steketemperatur. Avhengig av om man starter sendingen med 'v' eller 's' blir enten det ene eller det andre feltet fylt med et tall, mens det gjenværende forblir tomt.

```

void receiveEvent(int bytes) {
    Serial.println("test:");
    // First receive the signal char
    char sig = Wire.read();

    // Assemble bytes into a float
    float t;
    byte * p = (byte*) &t;
    unsigned int i;
    for (i = 0; i < sizeof t; i++) *p++ = Wire.read();

    // JSON skeleton to insert values into
    String prefix = "{ \"Temperatur vannbad\": \"";
    String midfix = "\", \"Temperatur steke\": \"";
    String suffix = "\" }";

    // Send data over to google sheets
    char temp_arr[16];
    dtostrf(t, 4, 2, temp_arr);
    if(sig=='V') data = prefix+String(temp_arr)+midfix+" "+suffix;
    else if(sig=='S') data = prefix+" "+midfix+String(temp_arr)+suffix;

    Serial.println(data);
}

```

### *thingProperties.h*

Filen under er en konfigurasjonsfil for Arduino Cloud. Den inneholder informasjon om wifi-navn og passord som Arduino skal bruke, samt navn og passord for selve Arduinoens tilkobling til IOT Cloud. Innloggingsdata er selvfølgelig ikke tatt med, av hensyn til personvern.

```
// Code generated by Arduino IoT Cloud, DO NOT EDIT.

#include <ArduinoIoTCloud.h>
#include <Arduino_ConnectionHandler.h>

const char DEVICE_LOGIN_NAME[] = "****";

const char SSID[] = SECRET_SSID; // Network SSID (name)
const char PASS[] = SECRET_OPTIONAL_PASS; // Network password
const char DEVICE_KEY[] = SECRET_DEVICE_KEY; // Secret device password

String data;

void initProperties(){

    ArduinoCloud.setBoardId(DEVICE_LOGIN_NAME);
    ArduinoCloud.setSecretDeviceKey(DEVICE_KEY);
    ArduinoCloud.addProperty(data, READ, ON_CHANGE, NULL);

}

WiFiConnectionHandler ArduinoIoTPreferredConnection(SSID, PASS);
```

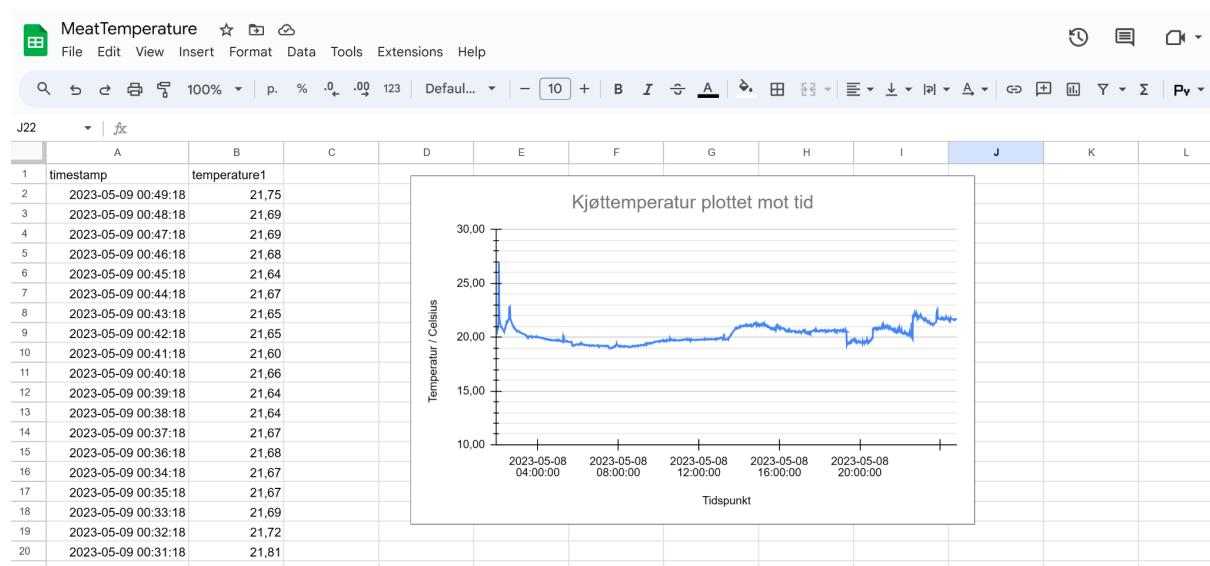
Arduino Cloud kan i seg selv brukes for både lagring og visualisering av data, men det er ikke en plattform folk vanligvis bruker eller er villige til å lære seg. Derfor tenkte vi at det ville vært mer brukervennlig å heller lagre og visualisere temperaturene i Google Sheets.

Etter at dataene er mottatt i Arduino Cloud, blir de automatisk videresendt til et Google Regneark via webhooks til et Google Apps Script. Begge deler er eksisterende funksjonalitet på disse plattformene. Dataene blir sendt over som en JSON-kode, som blir sendt til riktig kolonne (steke eller vannbad) i regnearket. I tillegg kommer et datostempel med hver sending (generert av Arduino Cloud), som brukes direkte i loggingen. Koden for denne omgjøringen er en adaptasjon av en kode vi fant på på GitHub av brukeren marcopass<sup>[8]</sup>. Koden inneholder en funksjon **doPost()** som automatisk gjør om en Arduino-formatert JSON-fil til et sett med felter i et rteark, og legger dem inn linje for linje med siste oppdatering først. Koden fungerte ikke helt til vårt formål slik den var i git, siden den la til en ny rad hver gang. Vi ønsket kun å legge til to celler (tidsstempel og temperatur) i en gitt kolonne (steke eller vannbad). I tillegg måtte flere av funksjonene moderniseres, og noen slettes. Arduino Cloud sender ofte de samme JSON-kodene flere ganger pga. interne feil. Vi adaptere derfor en

funksjon ***RemoveDuplicates()***, til å passe våre formål, ved at man i stedet for å slette dupliserte linjer, slettet linjer med dupliserte datoer/tid. Dette ga oss en renere loggfil. Funksjonen er hentet fra Google workspace<sup>[9]</sup>.

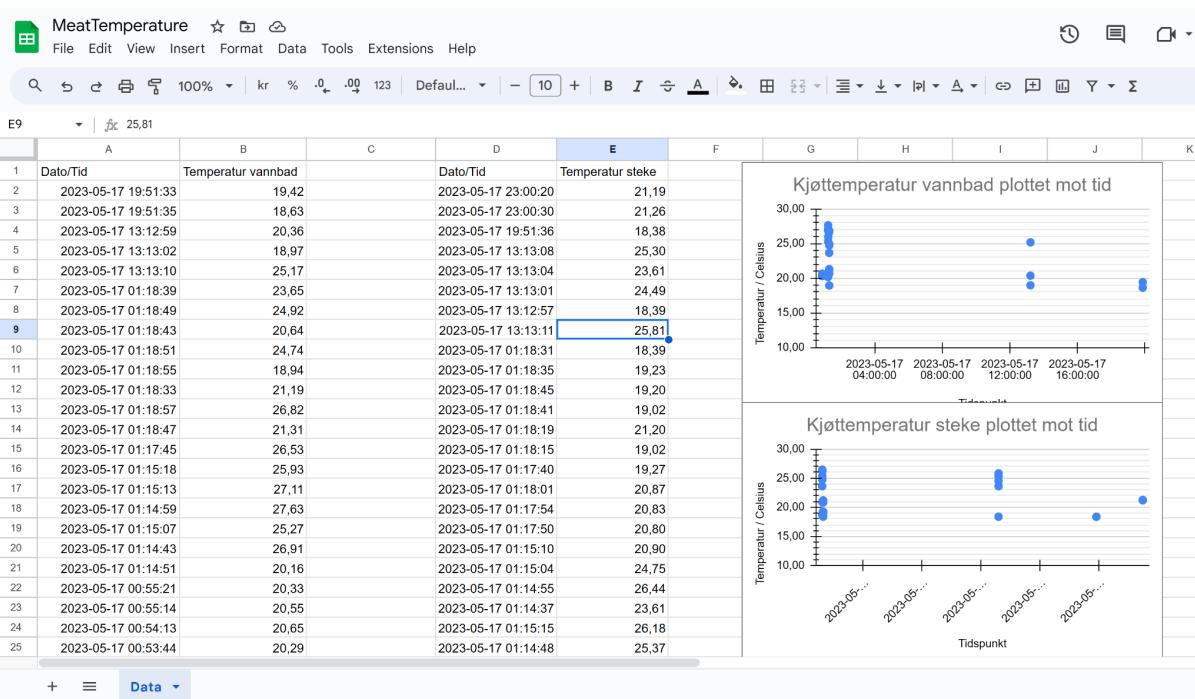
Under er to eksempler på hvordan loggen ser ut. Det første eksempelet er fra en prototype der Arduino sendte temperaturen automatisk hvert minutt, og sendte bare 1 type temperatur. En graf viser trenden over tid. Med en slik graf og en så høy oppdateringsfrekvens, kunne man til og med sett kortvarige forandringer i temperaturen. Dette kunne brukes til å for eksempel se når en medarbeider åpnet kjøleskapet, men glemte å lukke det igjen siden temperaturen gikk opp kritisk mye pga dette. Man kunne også brukt dette til å se hvor fort kjøtt faktisk kjøles eller varmes opp på innsiden.

### ***Tidlig prototype av kontinuerlig temperaturmåling:***



Det andre eksempelet er sluttproduktet, der loggen brukes til å lagre målinger som brukeren foretar når brukeren ønsker det selv. I tillegg er det to kolonner der hver står for sin egen målingstype.

## Sluttproduktet med brukerdefinert tidspunkt for logging, og 2 forskjellige temperaturer:



Under finner man koden som vi brukte for å lage og oppdatere regneark-loggen vår. Den er skrevet i Javascript, med kommentarer på engelsk, der en del av kommentarene er våre egne, mens noen er fra den originale koden. Funksjonaliteten er fullt definert av JSON-innholdet som man får fra Arduino Cloud, så både feltinnhold og kolonnenavn er generert automatisk. Hvis JSON-en ville kommet inn med flere enn to felter, ville disse automatisk også blitt gjort om til nye par med kolonner av datoer og målinger. Grafene derimot, er ikke automatisk generert, men holder mål så lengre målingnavn og kolonnenummer ikke forandres.

I denne delen av koden skriver man først ut kolonnenavnene hvis de ikke allerede finnes. Deretter legger man inn en ny blank linje på topp i en kolonne hvis en ny verdi med tilhørende kolonnenavn har blitt funnet, og timestampen for denne verdien ikke er samme som den forrige timestampen som allerede var i kolonnen.

```
// Write out the dato/tid column labels
for (var i = 0; i < incLength; i++) {
    sheet.getRange(H HEADER_ROW, 1+i*3).setValue('Dato/Tid');
}

// Write the data column labels
for (var i = 0; i < incLength; i++) {
    var col = 2+3*i;
    // Check if the name is already in header
    if (sheet.getRange(H HEADER_ROW, col).getValue() != incNames[i]) {
```

```

        sheet.getRange(HDR_ROW, col).setValue(incNames[i]);
    }
}

// insert new row at the top in the columns where non-empty data is inserted
if not duplicated date
var duplicate = false;
for (var i = 0; i < incLength; i++) {
    if(incValues[i]!="") {
        if(Date(sheet.getRange(HDR_ROW+1, 1+3*i).getValue())!=date) {
            sheet.getRange(HDR_ROW+1,
1+3*i).insertCells(SpreadsheetApp.Dimension.ROWS);
            sheet.getRange(HDR_ROW+1,
2+3*i).insertCells(SpreadsheetApp.Dimension.ROWS);
        }
        else duplicate = true;
    }
}

```

Hvis verdien ikke har en tidligere brukt timestamp, skriver man ned den nye timestampen og verdien i de tomme cellene som er blitt åpnet opp. Her formaterer man også de nye cellene og sjekker hvilken type verdien man legger inn tilhører, slik at boolske verdier blir omgjort til 0 eller 1.

```

if(!duplicate) {
    // reset style of the new row, otherwise it will inherit the style of the
header row
    var range = sheet.getRange('A2:Z2');
    range.setFontColor('#000000');
    range.setFontSize(10);
    range.set FontWeight('normal');

    // write the timestamp where required
    for (var i = 0; i < incLength; i++) {
        if (incValues[i]!="") {
            sheet.getRange(HDR_ROW+1,
1+i*3).setValue(date).setNumberFormat("yyyy-MM-dd HH:mm:ss");
        }
    }

    // write the values where required
    for (var i = 0; i < incLength; i++) {
        if (incValues[i]!="") {
            var col = 2+3*i;
            // first copy previous values
            // this is to avoid empty cells if not all properties are updated at the
same time
            sheet.getRange(HDR_ROW+1, col).setValue(sheet.getRange(HDR_ROW+2,
col).getValue()).setNumberFormat("0.00");
        }
    }
}

```

```

        // turn boolean values into 0/1, otherwise google sheets interprets them
        // as labels in the graph
        if (incValues[i] == true) incValues[i] = 1;
        else if (incValues[i] == false) incValues[i] = 0;
        // fill the cell finally
        sheet.getRange(H HEADER_ROW+1, col).setValue(incValues[i].replace(".", ","));
        .setNumberFormat("0.00");
    }
}
}

for (var i = 0; i < incLength; i++) if (incValues[i]!="") removeDuplicates(i);
}

```

Dessverre skjer det ofte at Arduino Cloud sender samme JSON flere ganger på rad, og noen ganger kommer de seg gjennom filteret over. Da må man ha en metode som man kan bruke til å fjerne dupliserte målinger i ettertid. Funksjonen *removeDuplicates()* går gjennom en kolonne, og fjerner dupliserte verdier basert på timestamp, ved å lage en JS ordbok, som bare kan holde en verdi per nøkkel (timestamp). Denne funksjonen kjøres etter hver mottakelse av ny JSON for sikkerhets skyld.

```

function removeDuplicates(i) {
    var lastRow = sheet.getLastRow();
    const data = sheet.getRange(2, 1+i*3, lastRow-1, 2).getValues();
    const uniqueData = {};
    for (let row of data) {
        const key = row[0];
        uniqueData[key] = [key, row[1]];
    }
    sheet.getRange(2, 1+i*3, lastRow-1, 2).clearContent();
    var newData = Object.values(uniqueData);
    sheet.getRange(2, 1+i*3, newData.length, 2).setValues(newData);
}

```

## Kapittel 9 – Video

Link til video: <https://youtu.be/uq5j4pqnLsg>

Kort beskrivelse av video: Videoen viser først loggføring av temperatur slik brukerne utfører denne oppgaven uten vår prototype. Det blir så beskrevet hvorfor brukeren ønsker en digital løsning. Vår prototype blir så presentert. Det blir først beskrevet hvordan prototypen fungerer, og så vises prototypen i brukskontekst. Til slutt viser videoen de tekniske detaljene ved prototypen. All funksjonalitet som blir demonstrert på videoen er implementert i prototypen.

Videoen ble filmet med et mobilkamera og redigert i Final Cut Pro<sup>[10]</sup>



## Kildeliste

- De fullstendige kode filene: [GitHub - 010nova/syvende\\_sans](https://GitHub - 010nova/syvende_sans)
- Video: <https://youtu.be/uq5j4pqnLsg>
- [1] Programming electronic academy. (2021). “Using Serial.read() with Arduino | Part 2”.  
<https://www.youtube.com/watch?v=nSGnCT080d8>
- [2]  
<https://www.jernia.no/kj%C3%B8kken/matlaging/kj%C3%B8kkenutstyr/steketermometer/inne-steketermometer-elektrisk-sort-med-timer/p/00174560?queryID=3828ae9ee2fed82d0e0522cf54cf0d57>
- [3] Spenningsdeler [https://en.wikipedia.org/wiki/Voltage\\_divider](https://en.wikipedia.org/wiki/Voltage_divider)
- [4] GeoGebra Classic 6 <https://www.geogebra.org/download>
- [5] electrocredible.com. (2022). “Arduino OLED Display Tutorial”  
<https://electrocredible.com/arduino-oled-display-tutorial-ssd1306-i2c/>
- [6] Open AI. ChatGPT. <https://chat.openai.com/>
- [7] Wire kommunikasjonsprotokoll  
<https://reference.arduino.cc/reference/en/language/functions/communication/wire/>
- [8] marcopass. (2019). <https://github.com/arduino/arduino-iot-google-sheet-script>
- [9] <https://developers.google.com/apps-script/quickstart/library>
- [10] Final Cut Pro versjon 10.6.2 <https://www.apple.com/final-cut-pro/>