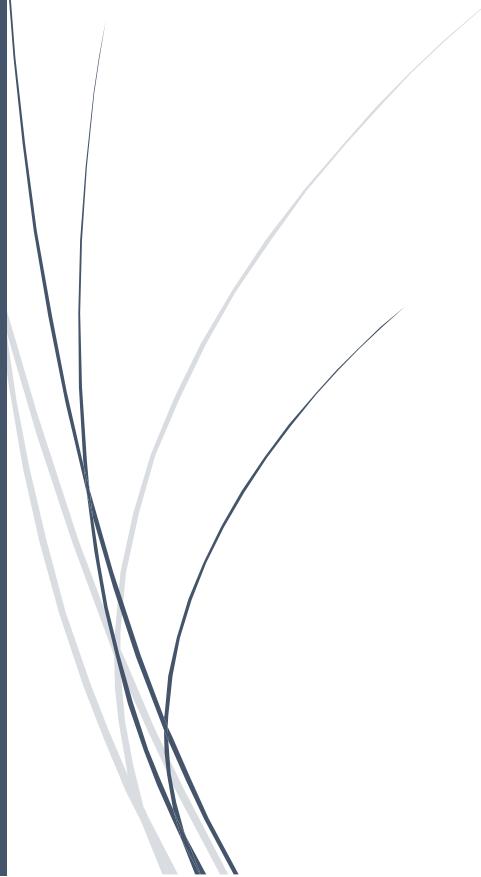


05.05.2023

# Smart City Prosjekt

Prosjektoppgave i IELET1002



Asbjørn Sørensen, Håkon Malmbekk, Kato Abrahamsen  
og Petter Fuglestrand  
GRUPPE B18

KANDIDATER (ETTERNAVN, FORNAVN):

Abrahamsen, Kato

Fuglestrand, Petter

Malmbekk, Håkon

Sørensen, Asbjørn

DATO:	FAGKODE:	GRUPPE (NAVN/NR):	ANT SIDER/BILAG:	BIBL. NR:
05/05.2023	IELET1002	B18	33 /	NA

TEAMET:

Arne Midjo (ansvarlig)

Even J. Christiansen (labingeniør)

Torje Johansen (teamleder veiledning)

TITTEL:

Prosjektoppgaven i IELET1002 DataTeknikk

SAMMENDRAG:

Prosjektet går ut på å lage en Smart City løsning på liten skala. Vi har satt sammen prosjektet ved å dele det opp i ulike moduler. Vi benyttet oss av en Zumo32U4 belterobot. Zumo'en blir programmert gjennom Arduino-IDE til å utføre en rekke oppgaver. Den har også mulighet til kommunikasjon gjennom en ESP-32 som er montert på toppen av Zumo'en.

Hovedoppgaven til Zumo-roboten er at den skal ha mulighet til å følge en linje, samt utføre utfordringer langs denne linjen. I tillegg har den et software-basert batteri, som tømmer seg mens bilen kjører. Når dette batteriet faller under 20% eller en bruker forespør opplading skal bilen automatisk følge teipen til den kommer til ladestasjonen som skal simulere at batteriet blir ladet. Alle modulene er sammenkoblet gjennom Node-RED og MQTT, og data blir vist på et brukergrensesnitt.

# Innhold

1	Innledning .....	4
2	Terminologi .....	5
3	Teoretisk grunnlag .....	6
3.1	Smart City .....	6
3.1.1	Smart City på mindre skala.....	6
3.2	Arduino IDE.....	6
3.3	God programmeringsstruktur.....	6
3.4	Zumo32U4 .....	7
3.4.1	Linjesensorer.....	7
3.5	ESP32 .....	7
3.6	Raspberry PI.....	8
3.7	Node-RED .....	9
3.8	MQTT .....	9
3.9	Termistor .....	9
3.10	Fotoresistor .....	10
3.11	IR break-beam sensor.....	10
4	Metode.....	11
4.1	Organisering av gruppearbeid .....	11
4.2	Versjonskontroll .....	11
4.3	Platform.io.....	11
4.4	Zumo32U4 og linjefølging-modulen .....	12
4.5	SW-Batteri .....	13
4.6	Bank .....	15
4.6.1	ESP32.....	15
4.6.2	Node-RED .....	15
4.7	Sensornode .....	16
4.7.1	Termistor .....	16
4.7.2	Fotoresistor .....	16
4.7.3	IR break-beam sensor .....	17
4.7.4	Node-RED .....	18
4.8	Ladestasjon .....	20
4.8.1	Opplasting ved lavt batteri .....	20
4.8.2	Brukerdefinert opplasting.....	21

4.8.3	Batteribytte .....	22
4.8.4	Strømpris hentet fra API .....	22
4.9	Skyserver (Node-RED).....	23
4.9.1	Port-viderekobling.....	23
4.9.2	Sikkerhet .....	23
4.10	Zumo kontroll .....	23
5	Resultater.....	24
5.1	Sluttresultat .....	24
5.1.1	Linjefølging-modulen .....	24
5.1.2	Sensornode .....	24
5.1.3	Ladestasjon .....	25
5.1.4	Software batteri / bank .....	26
5.2	Kompetanseoverføring.....	26
6	Drøfting .....	27
6.1	Arbeidet med prosjektet .....	27
6.2	Utfordringer og problemer.....	27
6.2.1	Kommunikasjon mellom Zumo32U4 og ESP-32 .....	27
6.2.2	Bruk av sensorverdier til å styre bilen .....	27
6.2.3	Kommunikasjon gjennom MQTT.....	28
6.3	Forbedringer .....	28
6.3.1	Garasje som alternativ ladeplass og verksted .....	28
6.3.2	Nettverks-sikkerhet.....	29
6.3.3	Kamera på Zumo .....	29
6.3.4	Bruke JSON for å redusere antall MQTT-topics.....	29
6.3.5	Mer universell bank-funksjon .....	29
6.3.6	Forbedret SW-Batteri .....	29
6.3.7	Integrering av brukergrensesnitt til Zumo-Control .....	29
6.4	FNs Bærekraftmål.....	30
7	Konklusjon.....	31
8	Referanser .....	32
9	Vedlegg .....	33

## 1 Innledning

I vårt stadige utviklede samfunn, blir det viktigere å finne gode bærekraftige løsninger for problemene som medfølger. Ett av de mest lovende konseptene som dukker opp i denne sammenhengen er Smart City. Smart City er en by som tar i bruk avansert teknologi for å forbedre levekår, øke effektivitet og forminske forbruk. En av de sentrale teknologiene for å gjennomføre dette er IoT, et nettverk hvor forskjellige enheter er sammenkoblet for å måle, prosessere og dele data for å sikre avgjørelser.

I dette prosjektet skal vi se nærmere på hvordan mikrokontrollere kan brukes til å realisere en Smart City på mindre skala. Vi skal bruke en Zumo32U4 belterobot som skal følge en sort teip-linje rundt en bane og utføre oppgaver langs banen. Roboten skal ha konstant kommunikasjon med en skyserver, hvor man kan styre hva roboten skal gjøre. I skyen skal man også ha oversikt over hvor mye batteri og penger på konto roboten har.

## 2 Terminologi

IDE	<i>Integrated Development Environment.</i> Programmeringsverktøy som forenkler programmering, debugging og opplasting av kode.
Åpen-kildekode/ <i>Open-source</i>	Alle lisenser til kildekoden er gratis og ligger fritt ute på nettet for at hvem som helst kan laste ned og bruke
OLED	<i>Organic Light Emitting Diode.</i> Skjerm hvor hver piksel produserer sitt eget lys
Topic	Brukes innen MQTT, navnet på temaet en melding blir sendt med
Subscribe/Publish	Brukes innen MQTT, subscribe for å abonnere på meldinger fra et topic, publish for å sende meldinger over et topic
Millis	Innebygget klokke i mikrokontrolleren som starter samtidig som programmet. Teller i millisekunder
VS Code	Visual Studio Code. En IDE med mulighet for utvidelser, slik at den kan brukes til alle programmeringsspråk
RPi	Forkortelse for ettkorts-datamaskinen Raspberry Pi
Aktuator	Omgjør styresignaler til bevegelse, eksempelvis motor
JSON	Java Script Object Notation. Bygd opp på nesten samme måte som et dictionary, der du har key/value par. JSON-objekter kan Serialises, slik at de kan sendes over blant annet MQTT og Serial
/keyboard	Refererer til brukergrensesnittet i Node-RED som ligger under «Node-RED IP:1880/keyboard»
IoT	Forkortelse for Internet of Things
bCrypt	Krypteringsalgoritme brukt for å kryptere passord
Brukergrensesnitt	Kontaktflaten mellom brukeren og system
/ui	Refererer til brukergrensesnittet i Node-RED som ligger under «Node-RED IP:1880/ui»

## 3 Teoretisk grunnlag

### 3.1 Smart City

En Smart City er en by som bruker informasjon- og kommunikasjons teknologi for å optimalisere byens funksjoner og bidra til økonomisk vekst. En Smart City skal følge en bærekraftig utvikling. Det skal føre til bedre levekår hos innbyggerne, øke effektivitet og forminske forbruk. For å oppnå dette kreves en infrastruktur bygd på teknologi. IoT er en sentral teknologi som brukes i en Smart City. Det går ut på å koble flere enheter opp mot ett nettverk. Enhetene brukes til å måle, behandle og sende data. I en Smart City kan dette eksempelvis brukes til trafikkstyring, intelligente transport muligheter, offentlig sikkerhetsovervåkning eller smart grids.

#### 3.1.1 Smart City på mindre skala

En Smart City kan lages på mindre skala ved bruk av de samme prinsippene og teknologiene en Smart City anvender. Mikrokontrollere som ESP-32 eller ettkorts-datamaskinen RPi, sammenkoblet med sensorer og aktuatorer gjør det mulig å lage en liten Smart City som tilnærmer samme funksjonalitet.

### 3.2 Arduino IDE

Arduino er en åpen-kildekode plattform som gjør det enkelt å utvikle mikrokontroller-baserte prosjekter. Arduino inneholder både maskinvare og programvare, der maskinvaren må bestilles på nettet, mens programvaren Arduino IDE kan lastes ned gratis fra internett.

Arduino IDE bruker C++ som programmeringsspråk, men legger til en rekke mikrokontroller-spesifikke funksjoner. Hovedkoden består av en void setup() som kjører én gang når programmet starter og en void loop() som kjører kontinuerlig.

### 3.3 God programmeringsstruktur

Når man arbeider med å skrive programmer er det ofte flere som skal jobbe med eller feilsøke koden. Dette fører med seg en del standarder for god programmeringsstruktur, slik at andre har mulighet til å forstå koden man skriver.

Godt beskrivende variabelnavn er et eksempel på dette. Man skal i utgangspunktet skjønne hva en variabel gjør uten å måtte lese kommentarene som er koblet til den.

Kommentarene man bruker skal i hovedsak forklare hva funksjonene gjør og hva de brukes til.

Hovedfunksjonen i programmet, vanligvis kalt main(), skal som hovedregel bestå av en tilstandsmaskin som kjører en rekke funksjonskall. Funksjonskallene gjør det enklere å dele opp koden i mindre biter som er mer oversiktlige og gjør det enkelt å gjenbruke funksjoner som kan utføre samme oppgaver flere steder i programmet.

### 3.4 Zumo32U4

Zumo32U4 er en beltebilrobot utviklet av Pololu Robotics & Electronics. Kjernen av roboten er en ATmega32U4 AVR mikrokontroller.

Montert på Zumo'en er diverse sensorer og to separate motorer, som brukes til å bestemme hvordan roboten skal oppføre seg.

Zumo32U4 er spesielt egnet for linjefølgingsoppgaver, og den kan også brukes til å utføre andre oppgaver, for eksempel kollisjondeteksjon.

For programmering i Arduino IDE, bruker Zumo'en et ferdigskrevet bibliotek (Zumo32U4.h), som inneholder både funksjoner og ferdigskrevet eksempelkode.

#### 3.4.1 Linjesensorer

Zumo32U4 er utstyrt med fem linjesensorer, montert foran, på undersiden av roboten.

Hver sensor består av et par med én infrarød sender, som emitterer infrarødt lys ned mot banen, og en fotoresistor som måler hvor mye av lyset som reflekteres tilbake.

Linjesensorene har markering DN 1-5, hvor DN1 er sensoren plassert lengst til venstre og DN5 er lengst til høyre, sett ovenfra.



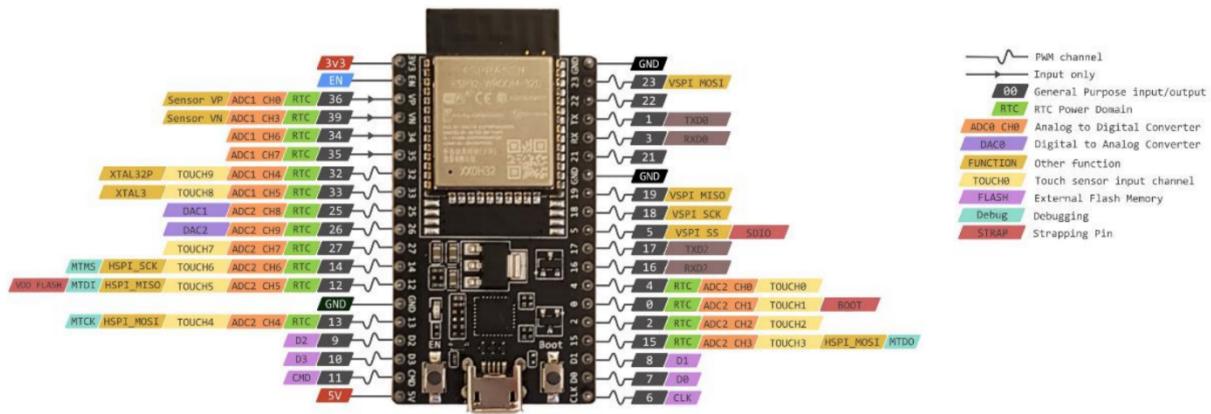
FIGUR 1: BILDE AV UNDERSIDEN TIL ZUMO32U4. LINJESENSORENE ER MARKERT MED RØD SIRKEL

### 3.5 ESP32

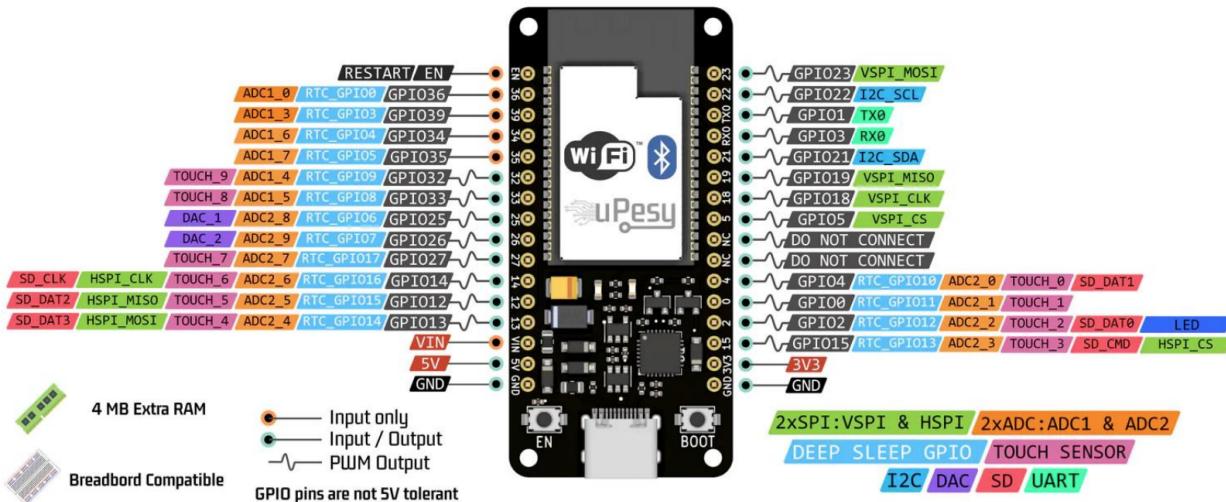
ESP32 er en 32-bits mikrokontroller med innebygd WiFi og Bluetooth. For programmering i Arduino IDE må man installere en ekstrapakke: Arduino core for the ESP32 [4]. Det er mange varianter av ESP32 med forskjellig funksjonalitet og pin-out. ESP32 finnes i både en-kjernet og to-kjernet utgave. Med den to-kjernede utgaven kan man kjøre opptil to funksjoner parallelt med hverandre på hver sin kjerne. I dette prosjektet bruker vi ESP32-WROVER-E, ESP32-WROOM-32E, og ESP32-WROOM-32, som alle er to-

kjernede utviklerkort produsert av Espressif. WROOM 32 er kortere og har færre pinner enn WROVER-E og WROOM-32E.

## ESP32 DevKitC V4 - Pinout



FIGUR 2: PIN-OUT DIAGRAM ESP-32 WROVER-E / WROOM-32E



FIGUR 3: PIN-OUT DIAGRAM ESP32-WROOM-32

## 3.6 Raspberry Pi

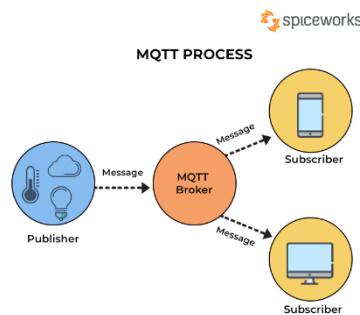
Raspberry Pi er en ett-korts datamaskin som kjører Raspberry Pi OS, som er et operativsystem basert på Debian.

### 3.7 Node-RED

Node-RED er et programmeringsverktøy i nettleseren som brukes til å behandle data og vise det fram på et dashbord på en nettside. Raspberry Pi blir ofte brukt til å kjøre Node-RED.

### 3.8 MQTT

MQTT er en meldingsprotokoll for IoT. Meldinger sendes over protokollen via publish/subscribe som et JSON-objekt, kalt payload. Payloaden blir sendt over et bestemt topic som mottakeren må abonnere på for å kunne motta meldingen. Dette gjør det mulig å sende informasjon mellom ulike enheter over det samme nettverket.



FIGUR 4: ILLUSTRASJON AV MQTT

### 3.9 Termistor

Termistor er en halvleder som er designet for å merke endring i temperatur, dette gjenspeiles i halvlederen som endring i resistans. Det finnes to typer termistorer: NTC (*negative temperature coefficient*) og PTC (*positive temperature coefficient*). NTC termistoren fungerer på den måten at resistansen blir lavere når temperaturen øker og PTC termistoren fungerer på den måten at resistansen blir høyere når temperaturen øker. For å finne temperaturen med de målte verdiene fra termistoren bruker man Steinhart-Hart ligningen.



FIGUR 5: BILDE AV TERMISTOR

### 3.10 Fotoresistor

Fotoresistor er en halvleder som er designet for å merke endring i lysstyrke/lumen, dette gjenspeiles i halvlederen som endring i resistans. Fotoresistoren har høy resistans når det er mørkt og lav resistans når det er lyst.



FIGUR 6: BILDE AV FOTORESISTOR

### 3.11 IR break-beam sensor

IR break-beam sensor er en bevegelses detektor, som består av to deler. Dette er en transmitter som sender ir-signal (infrarødt-signal) og en mottaker som fanger opp ir-signal (infrarødt-signal). I det tilfellet signalet mellom transmitteren og mottakeren blir brutt, vil dette bli registrert.

## 4 Metode

### 4.1 Organisering av gruppearbeid

Prosjektgruppen fordele arbeidsoppgavene etter interesse. Hvert medlem var ansvarlig for én hovedmodul hver. I løpet av prosjektperioden har gruppen møttes regelmessig for å gjennomgå hverandres kode og implementeringer. Under møtene har vi også satt egne tidsfrister for arbeidsoppgavene. Vi har hjulpet hverandre og samarbeidet når problemer har oppstått.

### 4.2 Versjonskontroll

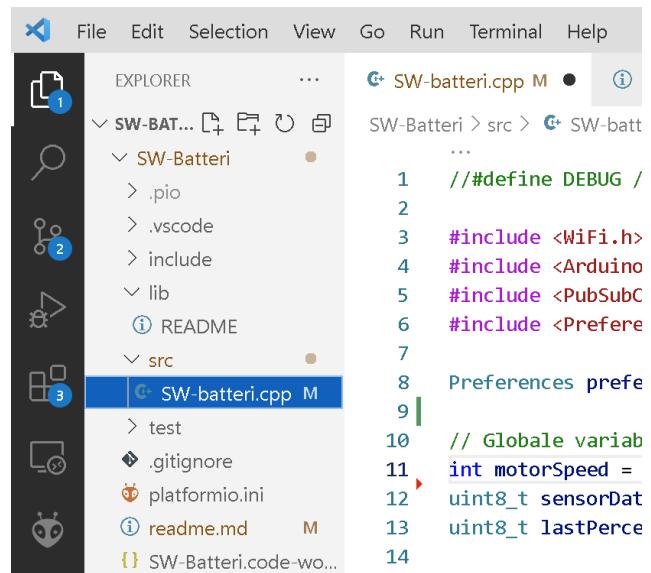
Vi startet tidlig å bruke GIT for å holde oversikt og struktur over prosjektet. GIT-mappen er delt inn i fire mapper, en del til hver modul. GIT-prosjektet er lagret på GitHub [5].

### 4.3 Platform.io

Platform.io er en utvidelse til VS Code som brukes til å programmere mikrokontrollere. Ved å bruke Platform.io i kombinasjon med GitHub utvidelser kan versjonskontroll gjøres enkelt ved hjelp av kun et par tastetrykk, og vi slipper å kopiere og lime inn kode hele tiden. Samtidig får vi raskt oversikt over hvilke endringer som er gjort.

For å bruke Platform.io til ESP-programmering må man lage et nytt Platform.io prosjekt i GIT-mappen til modulen man skal jobbe på. Når prosjektet er opprettet vil det se slik ut i VS Code:

Man jobber da på en .cpp fil i stedet for en .ino fil, og da vil Intellisense og auto-fullføring fungere. Når man laster opp koden til Platform.io automatisk gjøre om filen slik at den vil kjøre på mikrokontrolleren. Om man trenger ekstra biblioteker går man inn i platform.io og legger til bibliotek til prosjektet der. Bibliotekene vil da bli inkludert i mappen, så andre som laster opp koden med platform.io ikke trenger å legge til bibliotek på nytt.



The screenshot shows the VS Code interface with the Platform.io extension installed. The Explorer sidebar on the left displays a project structure for 'SW-Batteri'. It includes a root folder 'SW-Batteri' containing subfolders '.pio', 'include', 'lib', 'README', and 'src'. Inside 'src', there is a file named 'SW-batteri.cpp'. The code editor on the right shows the contents of this file:

```
//#define DEBUG /  
#include <WiFi.h>  
#include <Arduino>  
#include <PubSubC>  
#include <Preferences>  
// Globale variabler  
int motorSpeed =  
uint8_t sensorData  
uint8_t lastPerce
```

FIGUR 7: VS CODE ETTER AT DET HAR BLITT OPPRETTET ET PLATFORM.IO PROSJEKT

## 4.4 Zumo32U4 og linjefølging-modulen

Zumo32U4 bruker et eget bibliotek i Arduino IDE, Zumo32U4.h, for å programmere mikrokontrolleren.

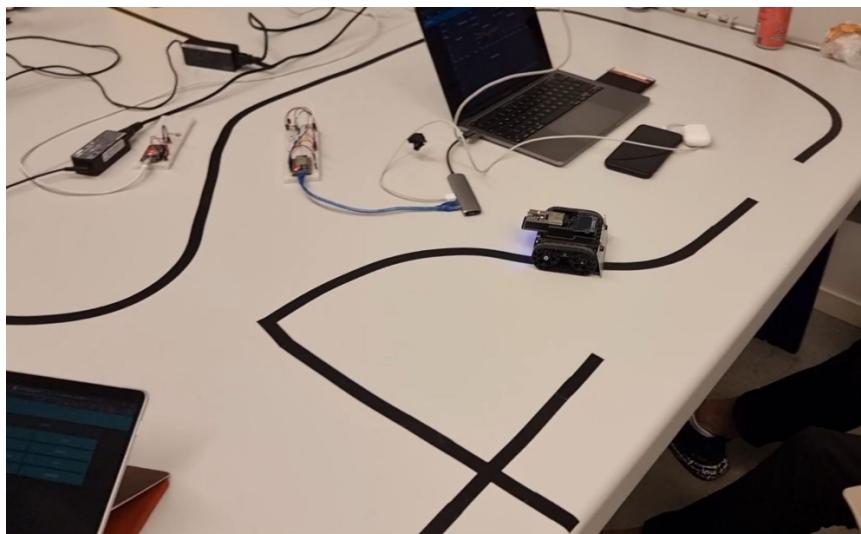
Dette gjør det mulig å fin-justere nøyaktig hva bilen skal gjøre ved bruk av motorene og sensorene. I dette prosjektet har vi brukt de fem linjesensorene til å få bilen til følge en linje. Verdiene man mottar fra sensorene brukes i funksjonen `.readLine(uint[])`, som returnerer en verdi som forteller oss hvor bilen er plassert i forhold til linjen.

Motorene programmeres ved å sette en fart den skal følge frem til den får beskjed om en ny hastighet.

Dette gjøres gjennom funksjonen `.setSpeeds(*venstre motorfart*, *høyre motorfart*)`.

```
160 /*Funksjon som sjekker hva bilen skal gjøre
161 * når den kommer til en teipbit som dekker begge yttersensorene
162 * Når deadEndRoad == true skal den kjøre rett frem,
163 * inn på blindveien og snu når den kommer til enden
164 Når deadEndRoad == false skal den ta en høyresving*/
165 void checkIntersection(){
166     if(deadEndRoad){
167         motors.setSpeeds(150, 150);
168         timeNow = millis();
169         while(millis() < timeNow + 300){}
170         lineSensors.readCalibrated(lineSensorValues);
171         while(!lostTrack()){
172             motors.setSpeeds(150, 150);
173         }
174         deadEnd();
175         deadEndRoad = false;
176     }
177     else{
178         rightTurn();
179         deadEndRoad = true;
180     }
181 }
```

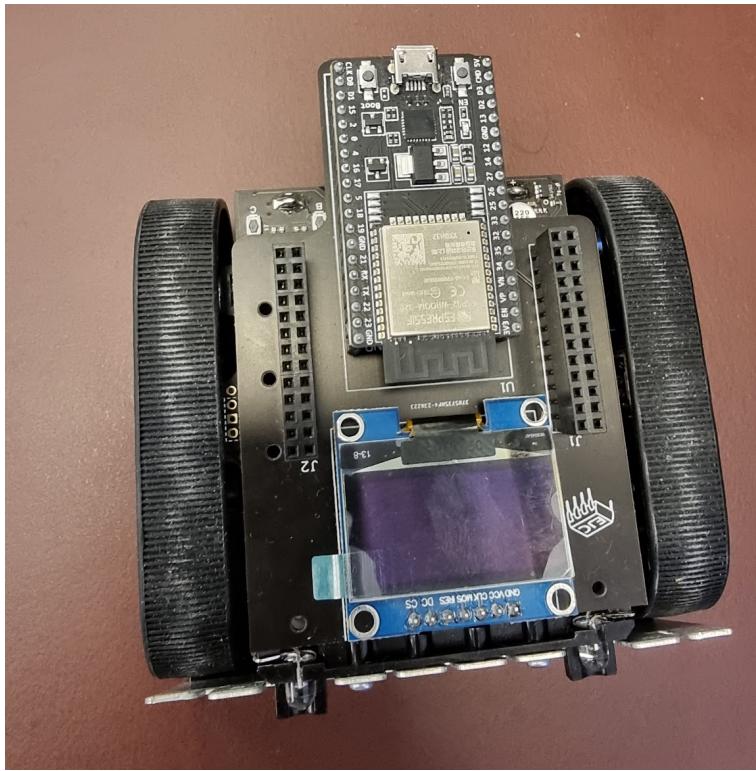
FIGUR 8: EKSEMPEL PÅ HVORDAN ZUMO-FUNKSJONENE KAN BENYTTE FOR Å STYRE BILEN



FIGUR 9: BANEN SOM ZUMO'EN FULGTE, SAMT LADESTASJONEN

## 4.5 SW-Batteri

Etter hvert som Zumo'en kjører rundt banen, vil den bruke strøm. Zumo'en har ikke oppladbare batterier, så vi simulerer et oppladbart batteri med et software-batteri. Vi bestemte oss for at kapasiteten på batteriet skal være på 20kWh. Software-batteri-koden kjører på en ESP32-WROOM-32 som sitter på et utvidelseskort oppå Zumo'en.



FIGUR 10: ZUMO32U4 MED ESP32 PÅMONTERT

ESP'en benyttes til all kommunikasjon mellom Zumo og omverdenen. ESP'en snakker med Zumo'en gjennom Serial. Meldingene fra Zumo'en er i JSON-format, da den både sender hastighet og sensordata. Når det blir sendt som JSON kan vi enkelt skille mellom hva som er hva. Meldingene til Zumo'en er kun String, da Zumo kun mottar tilstandsendringer, og String er enklere å behandle enn JSON. Zumo'en har så lav fart rundt banen at vi ser bort fra luftmotstand og friksjon i beregningene. Vi bruker derfor en enkel lineær formel som kun baserer seg på fart for å beregne batteriforbruk.

Batteriprosenten blir beregnet en gang i sekundet. Først blir batterinivået beregnet ut fra kapasitet og hvor mange prosent som er på batteriet. Batterinivået reduseres, og en ny batteriprosent blir beregnet. Batteriet tar skade om batteriet utlades under 5% eller om det lades opp. Batterikapasiteten vil dermed

reduseres ved bruk. Batteri-prosenten, -kapasiteten, og antall ladesykluser blir lagret i flash ved hjelp av preferences.h, slik at de forblir på samme nivå om ESP'en slås av.

```

344     // Oppdaterer batterinivået hvert sekund
345     if (now - lastBatteryUpdate > 1000 && motorSpeed > 0)
346     {
347         lastBatteryUpdate = now;
348         float batteryLevel = batteryPercentage / 100 * batteryCapacity; // Regner ut batterinivået i kWh
349         batteryLevel -= motorSpeed * powerConstant ; // P=mv
350         batteryPercentage = batteryLevel / batteryCapacity * 100; // Regner ut prosent
351
352         preferences.putDouble("batPrcnt", batteryPercentage); // Lagrer batterinivået i flash
353     }

```

FIGUR 11: EKSEMPEL PÅ HVORDAN BATTERIPROSENT BLIR BEREGNET

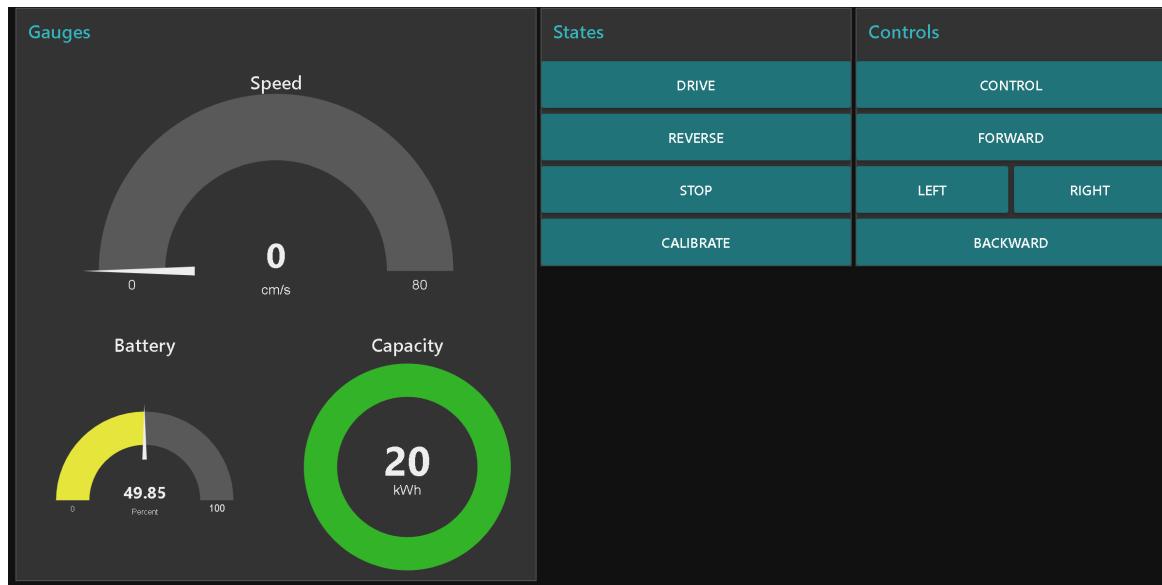
```

365     else if (batteryPercentage <= 10 && lastPercentage > 10) // Kjører bare en gang per utlading
366     {
367         Serial.println("Battery low");
368         lastPercentage = batteryPercentage;
369         batteryCapacity -= 0.5; // 0,5kWh av batterikapasiteten er tapt på grunn av lavt batterinivå
370         preferences.putDouble("batCap", batteryCapacity);
371     }

```

FIGUR 12: EKSEMPEL PÅ HVORDAN BATTERIKAPASITET BEREGNES

Hastighet, batteriprosent, og kapasitet sendes over MQTT til Node-RED, slik at det kan overvåkes fra brukergrensesnittet. I brukergrensesnittet kan man også endre tilstand på Zumo'en.



FIGUR 13: TELEMETRIVISNING FOR SW-BATTERI MODULEN. KNAPPENE LENGST TIL HØYRE ER FOR KONTROLL-MODULEN.

## 4.6 Bank

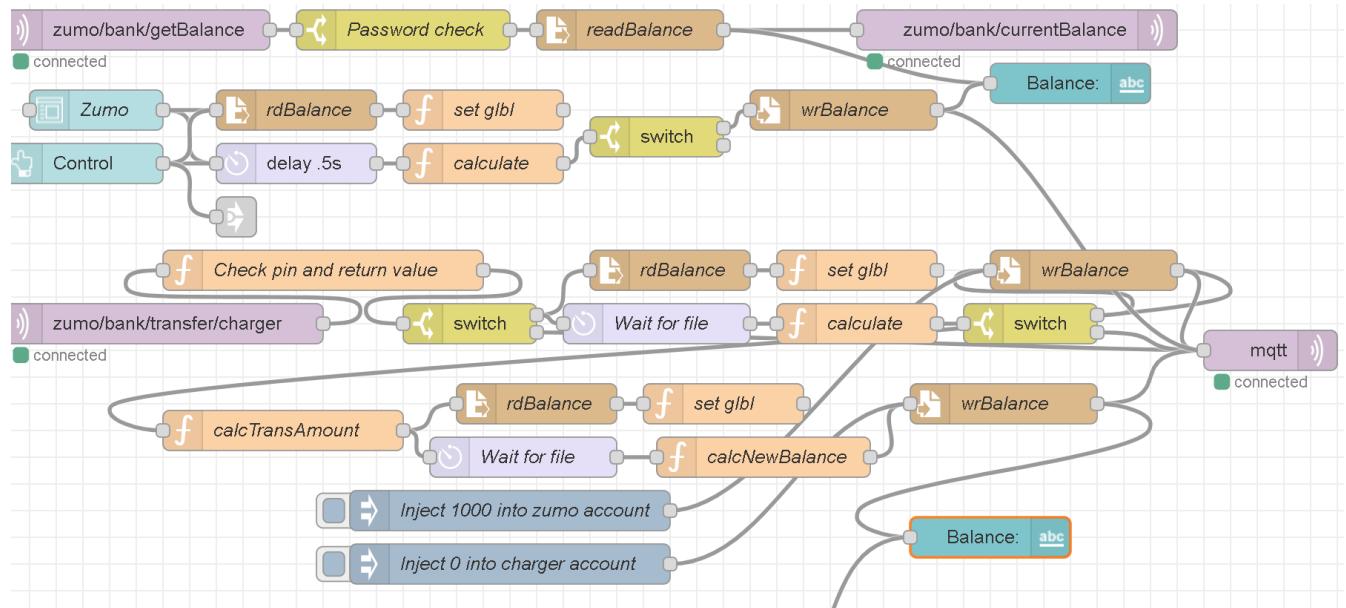
I utgangspunktet var bank-funksjonaliteten underlagt SW-batteri, men på grunn av omfanget valgte vi å gjøre det til en egen modul. Beløpet på kontoene blir lagret i en fil på Node-RED, og kan ikke direkte endres på fra ESP'ene.

### 4.6.1 ESP32

For å sjekke saldo må det sendes en pin-kode gjennom et bestemt topic over MQTT. For å overføre penger til en annen konto må det sendes en *String* på et annet emne gjennom MQTT der pin-koden er de siste fire sifrene, mens resten av meldingen er beløpet. Slik hindres det at uvedkommende kan sjekke saldo og overføre penger uten pin-koden. Zumo'en og ladestasjonen har hver sine kontører. For å forenkle prosessen med å implementere bank-kontoer på andre enheter har vi laget en klasse med innebygde metoder for å overføre, forespørre saldo, og motta ny saldo fra Node-RED.

### 4.6.2 Node-RED

På Node-RED er det satt opp en MQTT-in-node for overføring av penger, og en for å sjekke saldo. Det blir kontrollert om pin-koden er korrekt, saldoer blir oppdatert ved endringer, og ny saldo returneres til alle kontører som har endret saldo. Det må settes opp en ny flyt i Node-RED for hver konto.



FIGUR 14: BANK-FLYTN FOR ZUMO OG OVERFØRING TIL LADESTASJONEN I NODE-RED

## 4.7 Sensornode

Vi valgte de tre sensorene termistor (temperatur-sensor), fotoresistor (lys-sensor) og IR break-beam sensor (infrarød-sensor). Disse var ikke de eneste sensorene vi var innom. Vi hadde satt opp en tilt-sensor som vi valgte å skrote. Alle sensorene sender nyttig informasjon med MQTT som videre bearbeides og vises i Node-RED brukergrensesnittet.

### 4.7.1 Termistor

Termistoren har vi valgt å bruke for å måle temperaturen. Dette er gjort med å koble opp en 10k NTC termistor i serie med en 10k motstand, med 3,3 volt tilførsel. Vi mäter termistorverdien mellom termistoren og motstanden. Verdien vi får på esp32'en må gjøres om til grader celsius, først bruker vi ohms-lov og referanse motstanden, deretter utføres Steinhart-Hart ligningen som gir oss grader celsius, som sendes over MQTT.

```
107 // Read thermistor sensor
108 int temp = analogRead(temp_sensor);
109 // Convert thermistor value to celsius with Steinhart-Hart equation
110 float voltage = temp * (3.3 / 4096.0);
111 float resistance = REFERENCE_RESISTANCE * voltage / (3.3 - voltage);
112 float temperature = 1.0 / (1.0 / (NOMINAL_TEMPERATURE + 273.15) + log(resistance / NOMINAL_RESISTANCE) / B_COEFFICIENT) - 273.15;
113 // Publish to mqtt
114 mqttClient.publish(temp_state_topic, String(temperature).c_str());
```

FIGUR 15: EKSEMPEL PÅ BEREGNING AV GRADER

### 4.7.2 Fotoresistor

Fotoresistoren er en sensor vi har valgt å bruke i den grad at den registrerer hvor lyst eller hvor mørkt det er på banen. Denne er koblet opp til en esp32 mikrokontroller og gjennom denne vil den styre om lyset skal være på eller av. Vi har etter gjentatte forsøk funnet lysverdier vi synes ble passende for hvor mørkt det var i rommet. Om lyset er på sendes over MQTT, sammen med lysverdien.

```
129 // If the light value is below the threshold and override is not on, turn on the light
130 if (light_value <= 2300 && !override_on) {
131     digitalWrite(light_diode, HIGH);
132     // Publish state change to MQTT
133     mqttClient.publish(light_state_topic, "ON");
134 }
135 // If the light value is over the threshold and override is not on, turn off the light
136 if (light_value > 2300 && !override_on){
137     digitalWrite(light_diode, LOW);
138     // Publish state change to MQTT
139     mqttClient.publish(light_state_topic, "OFF");
140 }
```

FIGUR 16: EKSEMPEL PÅ BEARBEIDING AV LYSVERDI

Over ser man at det ikke er bare lysverdien som styrer om lyset skal være på eller av, det er også lagt inn en overstyrelses funksjon, denne funksjonen er koblet opp mot MQTT. Når man får rett signal så vil lyssensoren overstyres og lysene vil bli skrud på.

```

199   // When you receive the right message start light override
200   if (message == "on") {
201     override_on = true;
202     // Turn on the light
203     digitalWrite(light_diode, HIGH);
204     // Publish state change to MQTT
205     mqttClient.publish(light_state_topic, "ON");
206   }else{
207     override_on = false;
208 }
```

FIGUR 17: EKSEMPEL PÅ OVERSTYRELSE AV FOTORESISTOR

#### 4.7.3 IR break-beam sensor

Den infrarøde break-beam sensoren er brukt til å registrere passeringer på et fast punkt. Break-beam sensoren valgte vi å sette opp som en infrarød-diode som sender et infrarødt signal. Det infrarøde signalet mottas av en infrarød-sensor. De to forskjellige komponentene valgte vi å sette på egne esp32'er, dette gjorde vi for å få mer stabilitet og mindre avhengighet i forhold til plassering.

I koden valgte vi å bruke biblioteket IRremoteESP8266, siden vi støtte på problemer med ingen og uten bibliotek. Vi sender et signal fra den infrarøde-dioden hvert 50ms, dette for å skape ett stabilt og forutsigbart signal. Dette signalet blir plukket opp av den infrarøde-sensoren på den andre esp32'en. Vi vet om signalet er brutt om det ikke kommer gjennom et nytt signal innen 110ms. Gjennom gjentatte forsøk så fant vi ut at dette ble den korteste, men fortsatt stabile tiden for å oppdage ett signal brudd, som man ser under. Det oppdagede bruddet blir sent videre med MQTT.

```

155   // Checks if the signal is broken
156   else if((millis() - lastIRTime > 110) && (counter_1 == 0)) {
157     counter_1 = 1;
158     counter_2 = 0;
159     Serial.println("No IR signal detected");
160     mqttClient.publish(ir_state_topic, "1");
```

FIGUR 18: EKSEMPEL PÅ INFRARØDT BRUDD

Denne sensoren har også ansvar for å måle rundetid på banen dette gjøres med å registrere når signalet brytes og neste gang signalet brytes. Rundetid telleren er satt opp med et debounce system for å ikke få falske doble signaler som resetter telleren, deretter blir rundetiden sendt med MQTT.

```

160 // Start timer for lap 1
161 if (lapStartTime == 0) {
162     lapStartTime = millis();
163 }
164 // Check if debounce time is done
165 else if ((millis() - lapStartTime) > 1500) {
166     // Record lap time and start timer for next lap
167     lapTime = millis() - lapStartTime;
168     lapStartTime = millis();
169     Serial.println(lapTime);
170     // Publish labtime to mqtt
171     mqttClient.publish(lap_time_topic, String(lapTime).c_str());
172 }

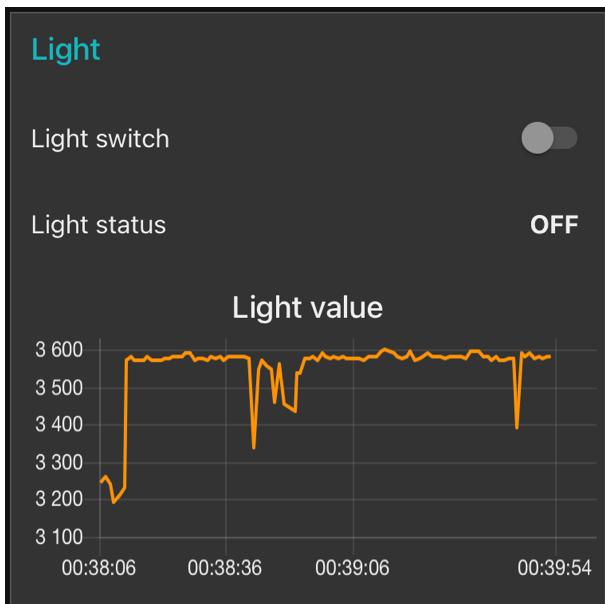
```

FIGUR 19: EKSEMPEL PÅ DEBOUNCE

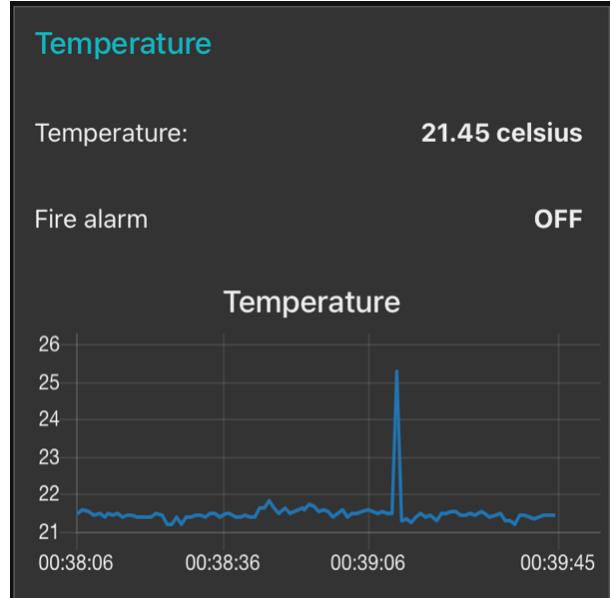
#### 4.7.4 Node-RED

All sensor dataen og andre sensor verdier blir sendt med MQTT til Node-RED hvor de fremstilles og bearbeides. I Node-RED brukergrensesnittet vises alle sensor dataene, de vises sammen i en nåtidsgraf, men også hver for seg for å få bedre oversikt over de ulike dataene.

Termistor verdiene blir vist som grader celsius, de vises i en graf som viser de siste 3 minuttene og direkte i nåtid. Temperatur delen av brukergrensesnittet har også ett varslingssystem som sier ifra hvis temperaturen er over 100 grader celsius.



FIGUR 21: LYS I BRUKERGRENSESNITT



FIGUR 20: TEMPERATUR I BRUKERGRENSESNITT

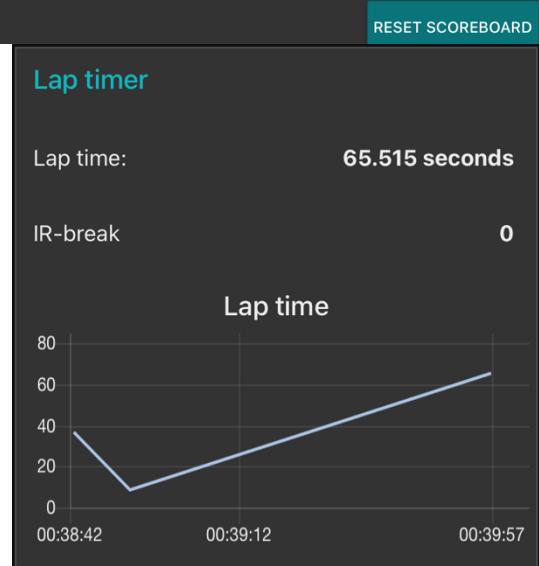
Fotoresistor verdiene blir vist som lysverdi og lys status. Lysverdien vises i en graf som viser de siste 3 minuttene. Lys statusen vises som enten av eller på. Man har også muligheten til å overstyre og skru lyset på direkte fra brukergrensesnittet.



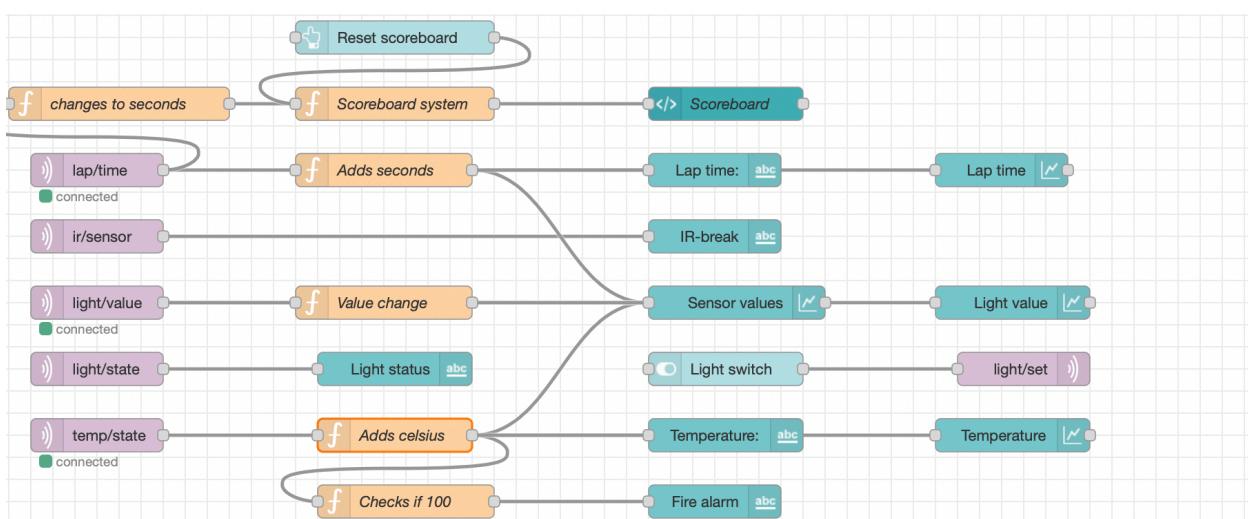
## FIGUR 22: RESULTATTAVLEN

IR break-beam verdiene vises som rundertid og om det infrarøde signalet er brutt eller ikke. Rundetiden blir vist som siste runde tid i sekunder og rundetider de siste 3 minuttene. I tillegg til dette så blir rundetidene rangert etter hvilken som er raskest. De 5 raskeste tidene blir lagret i resultattavlen som du ser over. Denne blir oppdatert for hver runde som går og hvis en av de nye tidene er raskere så blir den lagt til, ellers vil det ikke skje noe. Tavlen kan resettes med et enkelt knappetrykk.

Under ser man flyten som er satt opp i Node-RED for å styre innsamling av sensor dataene og distribuere de til Node-RED brukergrensesnittet.



**FIGUR 23: RUNDETID I BRUKERGBENSESNITT**



FIGUR 24: SENSORNODE FLYTEN I NODE-RED

## 4.8 Ladestasjon

Ladestasjonen er en viktig del av Smart City implementeringen. Den håndterer oppladingen Zumo'en trenger etter å ha kjørt og tapt batteriprosent. Den håndterer også bytte av batteriet dersom batterikapasiteten blir nedsatt. Ladestasjonen er en ESP-WROOM-32 tilkoblet en OLED skjerm med oppløsning på 64x128 piksler. Ladestasjonen er en simulering av funksjonaliteten til en reel ladestasjon, da Zumo'en ikke benytter seg av oppladbare batterier.

Vi bestemte at ladestasjonen skal ha to hovedfunksjonaliteter for opplading, samt batteribytte. Den første ladetypen er når Zumo'ens batteri faller under 20%. Den vil da kjøre til ladestasjonen og lade til 100%. Den andre ladetypen er brukerdefinert opplading. Her velger brukeren ladetype i Node-RED brukergrensesnittet. Begge ladetypene og batteribytte krever at Zumo'en har nok penger på kontoen.

### 4.8.1 Opplading ved lavt batteri

For å unngå at Zumo'en skal stoppe på grunn av tomtt batteri, vil Zumo'en automatisk kjøre til ladestasjonen og lade ved lavt batteri. ESP'en som er festet på Zumo'en vil automatisk gå i tilstanden «CHARGE» dersom det er under 20% på batteriet. Neste gang Zumo'en passerer break-beam sensoren vil sensornode ESP'en sende en beskjed over MQTT til ESP'en festet på Zumo'en. Deretter vil ESP'en sende beskjed via seriell kommunikasjon til Zumo'en at den skal stoppe. Det blir samtidig sendt en ny melding over MQTT til ladestasjonen at oppladingen skal igangsettes. Ladestasjonen vil dermed starte en «standard» opplading, som er en fullading.

#### 4.8.2 Brukerdefinert opplading

Helt sentralt for denne modulen har vi valgt at opplading kan bestilles fra brukergrensesnittet i Node-RED. Vi så dette som en relevant løsning i en Smart City. Man har forskjellige valg for hvilke oppladingstyper som er ønskelig. Det kan velges mellom fullading, eller opplading til bestemt prosent. Videre kan kunden også velge om den ønsker hurtigladning, som gjør at oppladingen skjer dobbelt så fort. Det kan også bestilles opplading til et senere tidspunkt. Ønsket oppladingstype velges og ladeknappen trykkes.



FIGUR 25, BRUKERGRENSESNITT FOR LADESTASJONEN I NODE-RED

Det blir sendt melding over MQTT til ladestasjonens ESP når en knapp eller bryter blir endret. ESP'en vil lagre informasjonen som variabler, eksempelvis «fullCharge» og «fastCharge». Dersom «Lad» blir trykket i brukergrensesnittet vil ESP'en sjekke følgende variabler og sette ladestasjonen i en «waitingForZumo»-tilstand. Her blir det publisert melding over MQTT til ESP'en på Zumo'en, som vil sette den i tilstand «CHARGE». På samme måte som ved opplading ved lavt batteri, vil oppladingen starte når den ankommer ladestasjonen.

```
224 //Funksjonen returnerer tilstanden ladestasjonen er i
225 chargeProgress initializeCharge() {
226     if (topicVar.chargeButton == 1) {           //Skjekker om ladeknappen er trykket på
227         if (topicVar.currentBalance > 5) {       //Skjekker om det er nok penger på konto til å starte lading
228             if (topicVar.readyForCharge == 1) {    //Skekker om Zumo er klar til å lade
229                 return userDefinedChargeStart;
230             }
231             //Skjekker at kun fullading eller lading til prosent er valgt.
232             if ((topicVar.fullCharge == 1 && topicVar.chargeToPercentage == 0) || (topicVar.fullCharge == 0 && topicVar.chargeToPercentage > 0)) {
233                 //Skjekker for lading innstilt etter tid
234                 if ((topicVar.chargeTime == -1) || (getTimeInMilliseconds() > topicVar.chargeTime)) {
235                     return waitingForZumo;
236                 }
237                 return waitingForChargeTime;
238             }
239         }
240     }
241 }
```

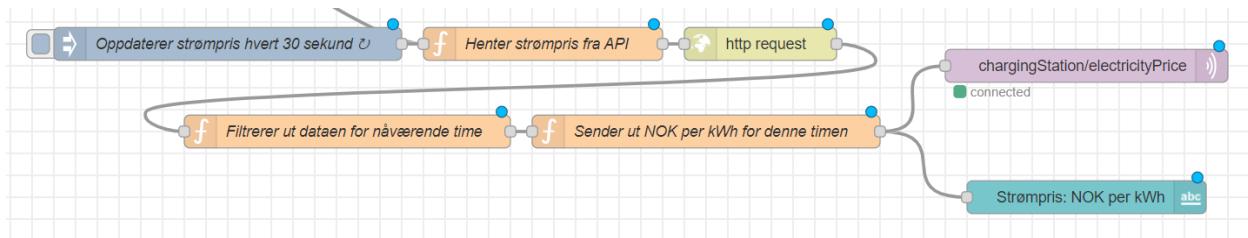
FIGUR 26, KODESNUTT FRA INITIALISERINGSFUNKSJONEN PÅ LADESTASJON ESP'EN

#### 4.8.3 Batteribytte

Dersom man trykker på «*Swap battery*» i brukergrensesnittet sendes det en melding direkte til ESP'en festet på Zumo'en at batteribytte er forespurt. Zumo'en vil på samme måte stoppe ved ladestasjonen og et batteribytte vil utføres i stedet for en opplading.

#### 4.8.4 Strømpris hentet fra API

For å gjøre ladestasjonen mer realistisk kom gruppen frem til å forsøke å hente ut den faktiske strømprisen i Midt-Norge for å beregne prisen for en opplading. For å oppnå dette brukte vi en API fra [www.hvakosterstrommen.no](http://www.hvakosterstrommen.no), sammen med Node-RED.



FIGUR 27, STRØMPRIS FLYTEN I NODE-RED

I Node-RED flyten bruker vi en inject-node som oppdaterer informasjonen hvert 30 sekund. I funksjonsnoden «Henter strømpris fra API» definerer vi url-en vi skal bruke når vi forespør strømprisen fra API-en. For å hente ut dagens strømpris for Midt-Norge måtte lenken modifiseres med dagens dato, samt NO3 for Midt-Norge.

```
1 const now = new Date();
2 const year = now.getFullYear();
3 const month = String(now.getMonth() + 1).padStart(2, '0');
4 const day = String(now.getDate()).padStart(2, '0');
5
6 const url = `https://www.hvakosterstrommen.no/api/v1/prices/${year}/${month}-${day}_NO3.json`;
7 msg.url = url;
8
9 return msg;
10
```

FIGUR 28, API-FORESPØRSEL FUNKSJON UNDER STRØMPRIS FLYTEN I NODE-RED

Fra API'en mottar vi pris og estimert pris for strømmen for alle timer på forespurt dag som JSON-objekt, dette filtreres bort i påfølgende funksjon og sendes til slutt ut som NOK per kWh til brukergrensesnittet og til ladestasjon ESP'en.

## 4.9 Skyserver (Node-RED)

### 4.9.1 Port-viderekobling

Alle modulene snakker sammen gjennom skyserveren. Mye av prosjektarbeidet har foregått individuelt, og vi møtte tidlig på et problem med at modulene ikke kunne snakke med Node-RED om RPi'en ikke var på samme nettverk. Ved å ha RPi'en hjemme hos noen og viderekoble port 1880 og 1883 kan man koble til både Node-RED og MQTT uten å være på samme nettverk.

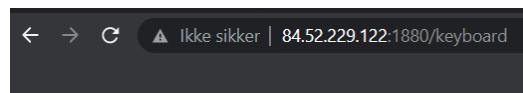
### 4.9.2 Sikkerhet

For å sikre Node-RED slik at ingen uvedkommende kommer seg inn har vi lagt til innlogging i både Node-RED flytene og på brukergrensesnittet [6]. For å oppnå dette måtte vi først «Hashe» et passord ved hjelp av bCrypt algoritmen. Vi opprettet også sikkerhetskopiering av Node-RED til GitHub i tilfelle det skjer noe [7].

## 4.10 Zumo kontroll

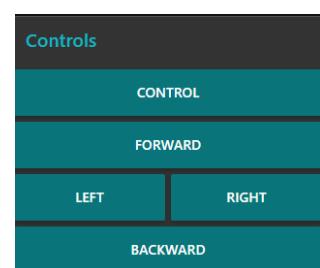
Som tilleggsmodul valgte vi å utvikle et kontrollsysten for Zumo via brukergrensesnittet i Node-RED. Denne modulen fungerer som en inntektsmodul, der kunder betaler for å leie Zumo'en. Siden serveren som Zumo ESP'en er koblet opp mot er port-viderekoblet, gir det muligheten til å styre Zumo'en fra hele verden.

I brukergrensesnittet i Node-Red satt vi opp ulike knapper man kan trykke på for å styre Zumo'en. Når en knapp blir trykket sendes en melding over MQTT med topic'et «zumo/control/direction». Dette mottar ESP'en montert på Zumo32U4 og sender det videre til Zumo'en via seriell kommunikasjon. For å få en mer naturlig måte å styre Zumo'en over Node-Red opprettet vi et nytt brukergrensesnitt som ligger under adressen «/keyboard». Dette brukergrensesnittet tar i bruk knappene på tastaturet, hvor 'w', 'a', 's', 'd' vil korrespondere til 'framover', 'venstre', 'bakover', 'høyre'.



Fjernstyr zumoen med knapper

- backward
  - left
  - left
  - forward
  - forward
- FIGUR 29, BRUKERGRENSESNITTET FOR Å STYRE ZUMO'EN MED KNAPPER



FIGUR 30, BRUKERGRENSESNITTET UNDER UI

## 5 Resultater

### 5.1 Sluttresultat

#### 5.1.1 Linjefølging-modulen

Banen som Zumo-roboten kjører på må bestå av svart elektrikerteip festet på et lyst bord for at bilen skal kunne følge den pålitelig. Banen består av diverse utfordringer som bilen utfører. Disse utfordringene er:

- Rettvinklede og slake høyre- og venstresvinger.
- Et veikryss der bilen først kjører rett frem til den mister banen, snur seg 180 grader, kjører tilbake til krysset og forsetter videre på banen, uten å kjøre tilbake der den kom fra.
- Et stykke med bane som mangler teip, der bilen kjører rett frem med konstant fart til den finner banen igjen.

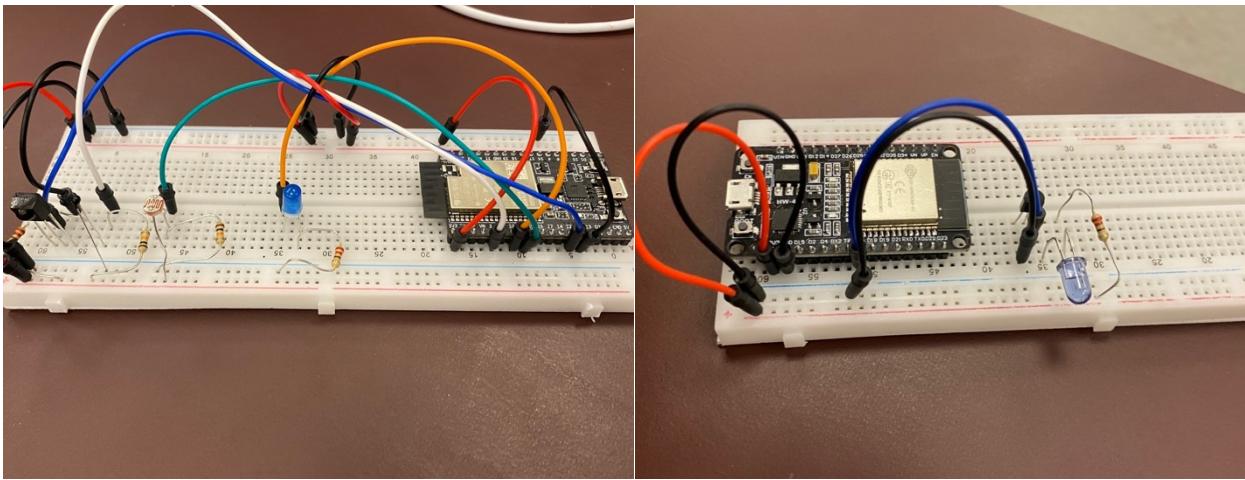
I sluttpunktet vårt klarer bilen å utføre alle disse utfordringene, men veikrysset krever at banen er satt opp slik at bilen ikke kommer skjevt på krysset. Dersom den kommer skjevt på krysset får bilen feil sensoravlesning og skjønner ikke hvor på banen den er.

Banen er også utstyrt med en break-beam-sensor som fungerer som en start- og mållinje og ladestasjon. Når bilen kjører flere runder på rad, får man opp forrige rundetid og de fem beste rundetidene på dashbordet i Node-RED.

Over dashbordet i Node-RED har man mulighet til å styre hva man ønsker at Zumo'en skal gjøre, som styres av én knapp for hver tilstand. Her får man også opp hastigheten til bilen.

#### 5.1.2 Sensornode

Sensornoden består av 3 forskjellige sensorer som alle ble velfungerende i sluttpunktet. Alle sensornodene viser dataene sine i Node-RED brukergrensesnittet, brukergrensesnittet er lett å navigere og viser fram dataene på forskjellige måter som gir god oversikt.



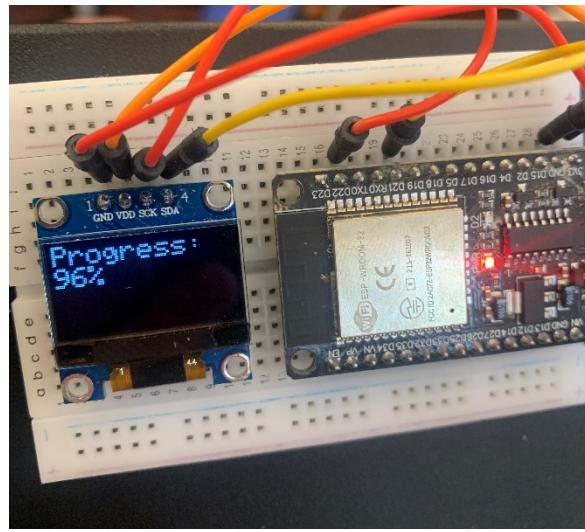
FIGUR 31: SENSORNODE

FIGUR 32: INFRARØD TRANSMITTER

I sensornode brukergrensesnittet ligger også resultatattavlen som blir nevnt ovenfor for å rangere de beste rundetidene.

### 5.1.3 Ladestasjon

Ladestasjonen ble satt opp i teip banen, etter break beam sensoren. I sluttresultatet fikk vi en fungerende ladestasjon, som ladet opp batteriet, både når SW-batteriet gikk under 20% og når det ble sendt ladeforespørsel. De ulike typene ladning fikk vi til å fungere, samt hurtiglading som gikk dobbelt så



FIGUR 33, LADESTASJONEN

fort. Prisen for en opplading ble batterikapasiteten i kWh ganget med strømprisen i Midt-Norge i øyeblikket den ladet, pluss eventuelt tillegg på 35% for hurtiglading. Bytte av batteriet fikk vi også til å fungere.

#### 5.1.4 Software batteri / bank

Software-batteriet skal simulere et ekte oppladbart batteri, ved at det har variabel utladningshastighet, og batterikapasiteten reduseres utover batteriets levetid. I løpet av testingen mistet batteriet kapasitet slik som forventet, og batteriprosenten gikk ned ved bruk av Zumo'en.

Ved oppladning skal Zumo'en overføre penger til ladestasjonen, og Zumo'en skal tjene penger ved bruk av kontroll-funksjonaliteten. I tillegg skal en banktjeneste være sikker. Banktjenesten har fungert akkurat slik den skal.

## 5.2 Kompetanseoverføring

Alle i gruppen har god innsikt i hvordan de forskjellige modulene fungerer, da dette er noe vi har vært helt avhengige av for å kunne implementere de forskjellige funksjonalitetene på tvers av både moduler og enheter. Zumo-modulen ville for eksempel ikke fungert dersom ikke alle i gruppen hadde oversikt over hvordan Zumo'en oppførte seg og hvordan den kommuniserte med NodeRed gjennom ESP'en som er montert på den. Det samme gjelder også ladestasjonen, da denne er avhengig av informasjon fra alle de forskjellige hovedmodulene for å kunne starte ladingen.

## 6 Drøfting

### 6.1 Arbeidet med prosjektet

Vi startet tidlig med å jobbe med planlegging og arbeidsfordelingen for prosjektet. Det ble en naturlig fordeling på hvem som gjorde hva basert på tidligere kunnskap og hva hver av oss syntes virket interessant. Samarbeidet i gruppen var svært bra og alle hadde god kontroll på hva egen modul gjorde, noe som gjorde kompetanseoverføringen i gruppen svært effektiv.

De dagene vi endte opp med å jobbe utover kvelden ble samarbeidet dårligere enn det var når vi jobbet på dagen. Dette var sannsynligvis fordi vi alle begynte å bli slitne og ønsket å dra hjem. De gangene vi merket at dette skjedde bestemte vi oss for å ta kvelden og heller komme tilbake dagen etterpå.

### 6.2 Utfordringer og problemer

Under arbeidet med prosjektet har vi møtt på flere utfordringer og problemer, noen gikk relativt raskt å fikse, mens andre måtte vi bruke flere dager på før vi til slutt klarte å fikse dem. Disse utfordringene har ført til en bratt læringskurve, og samtlige i gruppa har kommet ut av prosjektet med en stor grad av mestringsfølelse. I dette underkapittelet skal vi ta for oss noen av de større utfordringene vi hadde og hvordan vi klarte å løse dem.

#### 6.2.1 Kommunikasjon mellom Zumo32U4 og ESP-32

Vi hadde lenge problemer med å få til pålitelig kommunikasjon mellom Zumo-roboten og ESP-32 som var montert på den. Dette viste seg å være på grunn av måten vi opprinnelig hadde lagt opp funksjonskallet som leste av seriell overvåkeren. Vi hadde lagt for mye av koden til kommunikasjon inn i hovedløkken, noe som skapte gjorde at funksjonen ikke fungerte som den skulle.

Vi løste dette ved å opprette et eget testprogram der vi kun fokuserte på å få til kommunikasjon mellom de to enhetene. I dette programmet kunne vi se på hva om fungerte og ikke, og bruke det vi lærte i hovedkoden. Et verktøy som ble svært viktig her var seriell overvåkeren mellom PC og Zumo. Ved å printe alt vi mottok med en gang vi fikk det, samt printe endringene som skjedde med Zumo'en, kunne vi få en oversikt over når vi faktisk klarte å kommunisere og om beskjedene førte til de riktige endringene i tilstand.

#### 6.2.2 Bruk av sensorverdier til å styre bilen

Det var flere ganger i koden der vi ønsket at bilen skulle oppføre seg annerledes basert på gitte sensorverdier, for eksempel når Zumo'en ikke fant banen. Måten vi opprinnelig prøvde å gjøre dette på

var ved å se på posisjonen til Zumo'en, der vi feilaktig hadde antatt at ved en posisjon lik null betyddet dette at den ikke fant banen.

Dette problemet ble også løst ved å bruke seriell overvåkeren mellom PC og Zumo'en i et eget testprogram. Vi printet verdiene fra sensorene og posisjonen, deretter bevegde vi bilen over en teipbit for å se på hvordan verdiene endret seg i sanntid. Det vi fant ut var at posisjon lik null betyddet at teipen befant seg rett over venstre sensor og 4000 betyddet høyre sensor. Når bilen mistet banen ble posisjonen satt til det den var rett før den mistet banen, noe som gjorde det å bruke posisjonen verdiløst.

Videre hadde vi også antatt at når bilen mistet banen ville alle sensorverdiene være null, men grunnet små fargeforskjeller i bordet vi brukte, fikk sensorverdiene en verdi mellom 0-100. Dette løste vi ved å sette opp en funksjon som sjekket når alle sensorene hadde verdier under hundre samtidig og bestemte at dette betyddet at bilen hadde mistet banen.

### 6.2.3 Kommunikasjon gjennom MQTT

Vi har hatt en del problemer med at noen meldinger ikke går gjennom når de skal. Vi mistenker at dette har med nettverksstøy, da det har vært flere andre grupper i samme rom da problemene har oppstått. Feilsøkingen på dette har vært svært tidkrevende og frustrerende, da det er vanskelig å si om det er feil på grunn av eksterne kilder, eller eventuelt hvilken modul som har feil i koden.

## 6.3 Forbedringer

På grunn av forsinkelser og utfordringene nevnt ovenfor var det dessverre noen ideer vi ikke fikk tid til å realisere.

### 6.3.1 Garasje som alternativ ladeplass og verksted

Vi hadde lenge en plan om å ha en egen garasje som Zumo'en skulle kjøre til som alternativ ladeplass og verksted for batteribytte. Den skulle kjøre forbi break-beam sensoren og ta av til høyre på banen, før den kjørte til garasjen, snudde seg 180 grader og stanset. Dette var noe som ble lagt opp til under hele arbeidet med Zumo-modulen, men under de siste testene av prosjektet stanset den ved break-beam sensoren i stedet for å ta av til høyre. Siden dette var noe vi fant ut av helt på slutten av arbeidsperioden rakk vi ikke å fikse dette og valgte derfor å droppe det som en egen tilstand.

En mulig årsak til at dette skjedde kan ha vært at funksjonene vi brukte til tilstandene «Garage» og «Charge» var svært like, noe som kan ha ført til at bilen oppførte seg annerledes enn forventet. Hvis

dette var tilfellet kunne dette problemet blitt løst ved å endre på måten Zumo'en registrerte når den skulle svinge av til garasjen.

### 6.3.2 Nettverks-sikkerhet

Ved å port-viderekoble åpner vi nettverket for mulige trusler. Dette kan løses på flere måter, blant annet ved å opprette et sikkert domene. Vi gjorde et forsøk på det, men enten så koster et sikkert domene en del, ellers så krever det mye kunnskap og arbeid å opprette det.

### 6.3.3 Kamera på Zumo

I starten av prosjektet var det mye snakk om å sette opp en kamera-sensor, slik at «Zumo kontroll» modulen kunne gjøre at en hadde muligheten til styre Zumo'en og se hvor den var. Vi så oss nødt til å skrote denne ideen grunnet vanskeligheter med å skaffe kamera-sensor til esp32.

### 6.3.4 Bruke JSON for å redusere antall MQTT-topics

Underveis i prosjektet lærte vi at JSON-objekter kan sendes som payload over MQTT. Ved implementering av dette kan man unngå å bruke mange topics for publish/subscribe og heller lagre dataen som et JSON-objekt og sende over et topic. Dette vil effektivisere koden.

### 6.3.5 Mer universell bank-funksjon

Slik bank-modulen er skrevet nå er det ganske mye arbeid å legge til nye kontoer, da flyten må kopieres og endres på for alle nye kontoer. Man må da også legge til enda flere overføringsfunksjoner, noe som vil øke eksponentielt med antall kontoer. Utøver prosjektet har vi lært at man kan sende JSON-objekter over MQTT, noe som hadde hjulpet veldig med å gjøre bankfunksjonen både mer universell og mindre komplisert.

### 6.3.6 Forbedret SW-Batteri

SW-Batteri modulen ble ikke utført helt etter kravene, da bank-funksjonaliteten tok mer tid enn forventet, og vi hadde problemer med både Seriell og MQTT-kommunikasjon. Vi kunne inkludert lading ved rygging og nødlading, og hatt flere faktorer for degradering av batteriet.

### 6.3.7 Integrering av brukergrensesnitt til Zumo-Control

Dersom man skal styre Zumo'en kan det gjøres ved å trykke på musetasten i «/ui», eller så kan det brukes tastatur i «keyboard». Dette kunne blitt slått sammen med siden for Zumo-Control. Grunnet begrensete ferdigheter innen web utvikling ble dette ikke realisert.

## 6.4 FNs Bærekraftmål

FNs bærekraftmål er 17 mål FN har satt for å stoppe klimaendringer, utrydde fattigdom, og bekjempe ulikhet innen 2030. Med utgangspunkt i dette prosjektet ser vi spesielt to mål relevante for Smart City prosjektet. Mål 9 innebærer å bygge en solid infrastruktur som fremmer bærekraftig utvikling. I en Smart City hvor vi kontinuerlig tar målinger, behandler og sender data, vil infrastrukturen spille en viktig rolle for å oppnå bærekraftig utvikling. Ikke bare infrastruktur som i bygninger og veier, men også teknologiske inngrep. Ved sensormålinger og bearbeiding av denne dataen vil vi ha bedre muligheter for å optimalisere ressursbruk og forvalte energi på en bærekraftig måte.

Mål 11 innebærer å gjøre byer og lokalsamfunn inkluderende, trygge, robuste og bærekraftige. Med IoT teknologi kan sikkerheten i byer forbedres. Sensorer som sender dataen over nettet, vil kunne varsle om farlige situasjoner for å minske skadeomfang. I sensornode modulen bruker vi en termistor for å ta temperaturmålinger og sende dataen over MQTT. Dette kunne vært brukt for å forsikre at temperaturen til Zumo'en ikke overskridet en gitt temperatur, for å forsikre trygg drift og unngå farlige situasjoner.

## 7 Konklusjon

Prosjektet har stått til de forventningene vi hadde da vi først begynte med arbeidet. Deler av det vi planla når vi startet måtte droppes, men gruppen er godt fornøyd med Smart Cityen vi kom frem til. Som sluttprodukt oppnådde vi fungerende løsninger for de ulike modulene.

Vi har lært mye om innsamling og bearbeiding av sensordata, noe som også har ført til økt interesse for hvordan man videreutvikle disse ferdighetene. Samarbeidsevnene våre også noe som har utviklet seg under prosjektet, da arbeidet krevde stor grad av samarbeid på tvers av modulene.

Dersom vi skulle gjort prosjektet på nytt ville vi startet med intensivarbeidet tidligere, slik at vi kunne realisert flere av de idéene vi hadde da vi startet.

## 8 Referanser

1. <https://www.arduino.cc/en/about>
2. <https://www.pololu.com/docs/0J63>
3. <https://www.twi-global.com/>
4. <https://github.com/espressif/arduino-esp32>
5. <https://github.com/Mr-Putin/DataTek-prosjekt>
6. <https://nodered.org/docs/user-guide/runtime/securing-Node-RED#editor--admin-api-security>
7. <https://nodered.org/docs/user-guide/projects/>

Figur 4: <https://pimages.toolbox.com/wp-content/uploads/2022/06/11060901/MQTT-Process.png>

## 9 Vedlegg

- Vedlegg 1: Bidragserklæring
- Vedlegg 2: Videopresentasjon
- All kode ligger og Node-RED flow ligger på [GitHub](#)