Erlend Joa
Patrick Lund Haugen
Linn Srongyoo Hjulstad
Markus Terjesen Evanger
Noah Nikolaj Klavestad-Christensen

Teknisk Dokumentasjon

Prosjektoppgave i IN1060 Institutt for Informatikk

Lenke til video: https://vimeo.com/831863437

1. Mål for prosjektet

Målet vårt for dette prosjektet har vært å utforske temaet “More than a human world” ved målgruppen konkurrerende idrettsutøvere i alderen 15-20, for så å bruke tilnærmingen “design med bruker” til å designe et produkt som dekker eller bidrar til å dekke et behov hos målgruppen . Vi landet på å inkludere tre brukere i målgruppen. Sammen med disse brukerne har vi gjennomført iterasjoner med datainnsamling, prototyping og evaluering, der hver iterasjon forbedret aspekter ved design og brakte oss nærmere et ferdig sluttprodukt.

Etter runder med datainnsamling landet vi på å ville lage noe for brukernes selvrealisering. Vi fant at mye av motivasjonen for å konkurrere i idretter hos brukerne kommer av å oppleve mestring ved å se progresjon på egen utvikling.Vi observerte at friidrettsutøvere gjorde dette ved å ta tiden ved sprintintervaller med en trener som tar tiden. Sluttproduktet vårt er til for å hjelpe friidrettsutøvere som driver med sprint og hekkeløp med å få tilbakemelding på egen tid i en brukssituasjon der en trener ikke er til stede.

2. Sluttprodukt: “Motion-Finish”

Sluttproduktet vårt har fått navnet “Motion-Finish”. Navnet skal bringe konnotasjoner til photofinish, en metode brukt i enkelte profesjonelle idretter, og som innebærer å bruke et kamera til å nøyaktig måle bevegelse over en målstrek. På samme måte har vi ved utvikling av vårt produkt vært opptatt av nøyaktige målinger med bruk av bevegelsessensorer. Dette får vi til ved å sende signaler over trådløs kommunikasjon mellom modulene via WiFi, oppfulgt med visuell presentasjon av data på en skjerm.

Løsningen er designet slik at den kan brukes på treningsøkt alene, der bruker kan få presis tilbakemelding på egne sprintintervaller. Produktet er knyttet opp til temaet “More than a human world” ved muligheten for eksplorerende interaksjon via signaler oppfanger av bevegelsessensorer og sendt med WiFi signaler. Vi mener at produktet vårt i brukssituasjonen (1 bruker på sprint-trening) tilbyr en løsning som ikke hadde vært mulig uten sensorbasert teknologi.

Gjennom iterasjonene med gjensidig læring fikk vi mye kunnskap fra brukerne for et best mulig utgangspunkt. Vi satte med det disse kravene til sluttproduktet vårt:

1. Produktet skal kunne plukke opp raske forbi-løp og gi feedback med lyd
2. Produktet skal være sammenleggbar og passe inn i en treningssekk
3. Produktet skal gi brukeren mulighet til å fritt bestemme avstand mellom start og slutt.
4. Resultatet skal presenteres etter at brukeren har fullført intervallet i form av tall

3. Komponent-tabell

Antall	Komponent	Funksjon
2	NodeMCU V3 Lua CH340G ESP8266 Development Board F5D3	Fungerer som hoved-arduino brett og sender WiFi signaler
2	HC-SR501 Infrared PIR Motion Sensor Bevegelsessensor Pyroelectric Module For Arduino Raspberry Pi	Registrerer bevegelse og sender en HIGH til arduino
1	0.96 / 0.91 / 1.3 OLED Display Module for Arduino, Raspberry	Viser en timer på en skjerm ved mål til bruker
6	Neodym-magneter 12 mm	Holder begge moduler sammen for å være praktisk å ta med
2	Piezo	Lager lyd ved i en brukssituasjon
2	Batteri 9V	Gi strøm til begge arduino
2	Switch	Starter modulene / fungerer som av og på switch

4. Komponenter



Sluttproduktet vårt med to PIR-sensorer, to ESP8266-mikrokontrollere, to Piezo og OLED-Skjerm gir en teknisk løsning som muliggjør nøyaktig timerfunksjonalitet, trådløs kommunikasjon og visuell presentasjon av data. Kombinasjonen av komponentene er ment for å gi en brukervennlig og effektiv prototypeløsning for presentasjon av data gjennom eksplorerende interaksjon. Komponentene ESP8266, PIR sensorer og OLED display er alle kjøpt fra elkim.no. Piezo er fra Arduino starter-kit, og batterier og magneter er kjøpt på dagligvarebutikk.

ESP8266 (NodeMCU V3 Lua CH340G ESP8266 Development Board F5D3)

Prototypen har to ESP8266 som fungerer som mikrokontroller for arduino kode og har alle de andre komponentene koblet til seg. Brettet har male ports hvor vi bruker female-to-male kabler som kobles i et breadboard, og som videre kobles til de andre komponentene. Måten modulene kommuniserer med hverandre er at de begge har innebygd WiFi funksjonalitet. Det er flere måter å bruke WiFi funksjonalitetene til NodeMCU biblioteket, men vi har valgt å gå for nettverksarkitekturen peer-to-peer, hvor vi har satt opp enveiskommunikasjon fra startsmodulen (modul A) til sluttmodulen (modul B). Vi kom fram til at denne løsningen passet oss veldig bra, ettersom det ikke krever kommunikasjon gjennom for eksempel mobilnett. Når mikrokontrollerene får i seg strøm er de klare til å sende og motta signaler uten noe mer oppsett. Rekkevidden har vi målt til å være på rundt 45 meter mellom brettene, men distansen kan variere ut fra forskjellige faktorer.

PIR (HC-SR501 Infrared PIR Motion Sensor

Det brukes to PIR-sensorer som begge har som funksjon å fange opp bevegelse fra bruker under løping forbi disse sensorene. De sender begge signaler som ved hjelp av kode modifiserer andre komponenter i prototypen. De er helt essensielle i å få løsningen til å fungere, ettersom det er signalene disse registrerer som vi faktisk sender videre over WiFi og/eller bestemmer tallet vi presenterer for bruker. PIR-sensorer er passive infrarøde sensorer som oppdager bevegelse ved å registrere endringer i infrarød stråling som genereres av objekter med varme.

OLED (0.96 / 0.91 / 1.3 OLED Display Module for Arduino)

Det brukes en OLED 0.96 tommer skjerm som gir brukeren visuell tilbakemelding i form av et sett med tall som representerer tiden det tok i millisekunder fra brukeren passerte den første PIR sensoren til bruker passerte den andre. Vi valgte denne skjermen fordi den passet med mikrokontrolleren, den viser tydelige piksler ute i sollys og det var denne modellen som var tilgjengelig å få tak i med rask levering over nett.

Piezo

Piezo vi har brukt er de som følger med i Arduino-uno startkit vi ble anbefalt å kjøpe av universitetet. Vi valgte å ta i bruk piezo for å kunne gi brukeren feedback på registrering av PIR sensorene. På den måten vet brukeren at timeren er satt i gang, samt at timeren har stoppet, istedenfor å måtte stole på dette blindt.

Batteri 9V

Batteriene vi har brukt er 9 volts batterier som er loddet til en av/på switch. Med denne løsningen er det lett å skru av og på maskinen.

5. Kode, overordnet

Kode i GitHub: https://github.com/markusevanger/SuperCircuit-1060
Modulene har hver sin kode skrevet i Arduino IDE med likheter og forskjeller. Modul A sin hovedfunksjon er å sende et WiFi signal til modul B ved registrert bevegelse, mens modul B sine funksjoner er å ta imot WiFi signal, starte en timer, vise timer på skjerm og registrere bevegelse for å stoppe timeren.

Etter begge modulene er skrudd på, er alt klart for å begynne treningsøkten. Timeren starter når brukeren har passert modul A. Et WiFi signal sendes når PIR sensor registrerer HIGH. Signalet blir fanget opp av modul B, som starter en timer og søker etter bevegelse. Når modul B sin PIR registrerer bevegelse, vil timeren stoppe, lyd vil spilles av og det som står igjen på skjermen er resultatet i minutter. Ved passering av begge PIR vil piezo lage lyd for å indikere at timeren har startet/stoppet. En ny timer vil starte når modul A sin PIR igjen registrerer HIGH. På den måten trenger ikke brukeren å ta stilling til en restart-funksjon. Skjermen vil vise brukeren sitt forrige forsøk, slik at hen kan se progresjon fra sist løp, og se om hen har løpt på en bedre tid. Hvis det er tilfellet, vil en egen lydsekvens spilles av fra piezo, for å gi bruker umiddelbar tilbakemelding på et bedre løp.

6. Kode, steg for steg

```
    // P2P COMMUNICATION SETUP:
// MAC Addresse til receiver
  uint8_t broadcastAddress[] = {0xE0, 0x98, 0x06, 0x05, 0xED, 0xC6};
//E0:98:06:05:ED:C6 = Boardet MED flekk på siden
//10:52:1C:E5:51:9F = Boardet UTEN flekk på siden

typedef struct structMessage {
  int signal;
};

structMessage myData;
bool startSignal = false; // start signal for opptelling
```

Hver mikrokontroller er programmert med hver sin arduino kodefil, med likheter og forskjeller. Oppsettet til WiFi bibliotekene er å først importere <ESP8266WiFi.h> og <espnow.h> . Koden over er lik i begge filer, bortsett fra innholdet i broadcastAddress. I hver sin kodefil deklarerer vi det andre brettet sin broadcast adresse i en liste broadcastAddress[]. Signalet som skal sendes er et int signal, og den lagrer vi i en struct: structMessage. Vi har også satt boolean startSignal til å være False (finnes bare i modul B). På nederste bilde til høyre under ser vi hvordan startSignal bestemmer om vi kjører metoden startLoop() eller ikke.

```
    // PIR CODE SETUP:
int inputPin = D5; // vel pin for PIR
int pirState = LOW; // vi starter, og antar at motion ikke er merket
int val = 0; // variabel for å lese pin status
    // PIEZO CODE SETUP
int b = 720; // frekvens for noten B
```

```
void pirSetup() {
  pinMode(inputPin, INPUT);
}
```

(definering og pirSetup() for A og B).

```
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64

#define OLED_MOSI   13
#define OLED_CLK   14
#define OLED_DC    5
#define OLED_CS    15
#define OLED_RESET 4

    // OLED CODE SETUP:
// gir oss et Display objekt med riktige rammer, som heter display.
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &SPI, OLED_DC, OLED_RESET, OLED_CS);
unsigned long minTeller;
unsigned long sekTeller;
unsigned long milTeller;
unsigned long elapsedTime;
String minString;
String sekString;
String milString;
String tellerString;
unsigned long forrigeTime = 0;
String forrigeString = "";
```

```
void oledSetup() {
  // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
  if(!display.begin(SSD1306_SWITCHCAPVCC)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Don't proceed, loop forever
  }
  display.clearDisplay(); // fjenrer det som var på skjermen før.
  display.setTextColor(WHITE); // vi må definere textfargen
  display.setTextSize(3);

  display.print(" SUPER ");
  display.print("CIRCUIT");
  display.display(); // oppdater skjermen med det vi definerete over.
}
```

(definering og oledSetup() i B. Vi importerer i tillegg <Adafruit_GFX.h> og <Adafruit_SSD1306.h>)

```
void setup() {
  Serial.begin(9600);
  pirSetup();
  wifiSetup();
}

void loop() {
  pirLoop();
}
```

(Modul A).

```
void setup() {
  Serial.begin(9600);
  pirSetup();
  wifiSetup();
  oledSetup();
  pinMode(D6, OUTPUT);  //PIEZO
}

void loop() {
  pirLoop();
  if (startSignal) {
    startLoop();
  }
}
```

(Modul B).

I setup() har vi definert egne metoder for å sette opp komponentene, hvor alle har fått navnet ***setup(). På samme måte kjøres metoder med navn ***loop() i kodenes loop(). Modul A og B har nesten likt setup, bare at i wifiSetup() i modul B bruker vi esp_now_register_recv_cb(OnDataRecv), mens i modul A bruker vi esp_now_register_recv_cb(OnDataSend). Dette for å deklarere hvilke metoder som skal kjøres når mikrokontrolleren sender eller tar imot et signal.

```
void wifiSetup() {
  // Set device as a Wi-Fi Station
  WiFi.mode(WIFI_STA);

  // Init ESP-NOW
  if (esp_now_init() != 0) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }
  // Set ESP-NOW Role
  esp_now_set_self_role(ESP_NOW_ROLE_CONTROLLER);
  // Once ESPNow is successfully Init, we will register for Send CB to
  // get the status of Trasnmitted packet
  esp_now_register_send_cb(OnDataSent);
  // Register peer
  esp_now_add_peer(broadcastAddress, ESP_NOW_ROLE_COMBO, 1, NULL, 0);
}
```

```
void wifiSetup() {
  // Set device as a Wi-Fi Station
  WiFi.mode(WIFI_STA);

  // Init ESP-NOW
  if (esp_now_init() != 0) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }
  // Set ESP-NOW Role
  esp_now_set_self_role(ESP_NOW_ROLE_CONTROLLER);
  // Register peer
  esp_now_add_peer(broadcastAddress, ESP_NOW_ROLE_COMBO, 1, NULL, 0);
  // Register for a callback function that will be called when data is received
  esp_now_register_recv_cb(OnDataRecv);
}
```

```
// Callback when data is sent
void OnDataSent(uint8_t *mac_addr, uint8_t sendStatus) {
  Serial.print("Last Packet Send Status: ");
  if (sendStatus == 0){
    Serial.println("Delivery success");
  }
  else{
    Serial.println("Delivery fail");
  }
}
```

```
// Callback when data is received
void OnDataRecv(uint8_t * mac, uint8_t *incomingData, uint8_t len) {
  memcpy(&myData, incomingData, sizeof(myData));
  Serial.print("Bytes received: ");
  Serial.print(len);
  Serial.print(myData.signal);
  Serial.print(" === START TIMER  ");

  // lagrer forrigeString som den forrige elapsedTime strenger:
  forrigeString = minString + ":" + sekString + ":" + milString;
  forrigeTime = elapsedTime; // tar vare på forrige elapsedTime
  elapsedTime = 0; // restarter timer
  startSignal = true; // starter opptelling
}
```

(modul A). → (modul B).

Metodene OnDataSent() og OnDataRecv() kjøres hver gang ESP sender eller mottar signal. Vi har derfor valgt å ikke fokusere på hva slags signal vi sender, ettersom metodene vil kjøre uansett, og hva vi sender fra A til B ikke vil ha noe å si for logikken vi bruker i kodene. Vi har gjort det heller slik at når OnDataRecv() kjører settes int elapsedTime til å være 0 (aka restarter timeren), og bool startSignal blir satt til True. På den måten, som man ser i modul B sin loop(), vil programmet hele tiden sjekke om startSignal er True eller False, og om den er True (aka. om OnDataRecv() har blitt kjørt) vil startLoop() kjøre.

I tillegg i OnDataRecv() for å sette den forrige timeren som forrige på skjermen, setter vi forrigeTime som elapsedTime, og DERETTER elapsedTime til 0. Slik tar vi vare på forrige løp, og kan sjekke om neste løp var bedre eller ikke.

Nedenfor ser vi startLoop() i kodefil til modul B.

```
void startLoop() {
  display.clearDisplay(); // fjern alt innhold fra forrige loop. Skjermen er nå svart
  display.setCursor(0, 0); // Velger hvor vi starter å skrive i kordinat systemet,
  // standard er 0, 0. Om vi ikke definerer dette hver loop, vil teksten
  // "forstette å skrive" fra der forrige tekst avsluttet.
  // i arduino kan vi ikke konkatinere, dermed skriver vi uten linjeskift.

  oppdaterTeller(); // setter nye verdier for timer variablene

  display.setTextSize(3);
  display.print(tellerString);
  display.setCursor(20,40);
  display.setTextSize(2);
  display.println(forrigeString);
  display.display(); // oppdater skjermen med det vi definerete over.
  delay(100); // venter 1 millisekund.
}

void oppdaterTeller() {
  elapsedTime += 100;  // øk total tid med 100 millisekunder
  unsigned long minTeller = (elapsedTime / 60000) % 60;
  unsigned long sekTeller = (elapsedTime / 1000) % 60;
  unsigned long milTeller = elapsedTime % 1000;

  minString = String(minTeller);
  sekString = (sekTeller < 10) ? "0" + String(sekTeller) : String(sekTeller);
  milString = "0" + String(milTeller / 100);
  tellerString = minString + ":" + sekString + ":" + milString;
}
```

(startLoop() i B).

For displayet importerer vi <SPI.h>, <Wire.h>, <Adafruit_GFX.h> og <Adafruit_SSD1306.h>, som alle kommer med mange metoder. startLoop() starter med å fjerne hva som var på displayet, for å så displaye det vi har printet ved display.print()) og display.display() senere i startLoop(). Vi bruker delay(100) for å kjøre startLoop() hvert millisekund startSignal fortsatt er True. På den måten oppdaterer vi displayet og elapsedTime med 100 hvert millisekund, og displayer det på OLED skjermen for hver kjøring. tellerString blir definert av kode kjørt i oppdaterTeller(), som tar i bruk logikk for å ta vare på hvor mange minutter, sekunder og millisekunder som har gått. Etter kjøringer av startLoop() vil displayet kunne se noe slikt ut:

Nedenfor ser vi hvordan pirLoop() fungerer i modul A og B:

```
void pirLoop() {
  val = digitalRead(inputPin);
  if (val == HIGH) {
    // MENS MOTION ER MERKET
    Serial.println("Motion detected!");
    // Sender WiFi signal:
    esp_now_send(broadcastAddress, (uint8_t *) &myData, sizeof(myData));
    // Spiller av lyd
    tone(D6, b);
    delay(1000);
    noTone(D6);
    delay(9000);

  } else {
    pirState = LOW;
  }
}
```

(Modul A). →

```
void pirLoop() {
  val = digitalRead(inputPin);
  if (val == HIGH) {
    // MENS MOTION ER MERKET
    Serial.println("Motion detected!");
    startSignal = false; // stopper timeren
    piezoBuzz(); // bestemmer hvilken lyd vi vil spille av
  }
  else {
    pirState = LOW;
  }
}
```

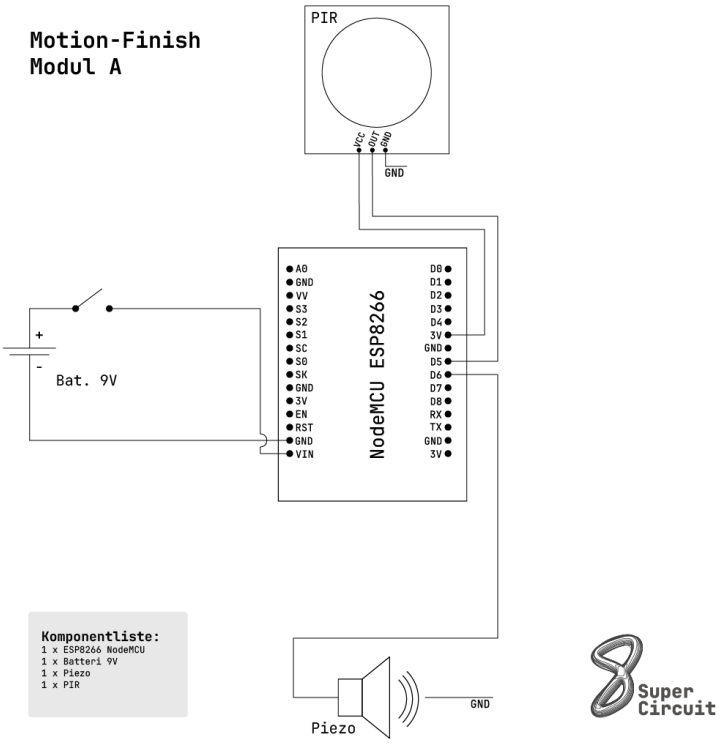
(Modul B).

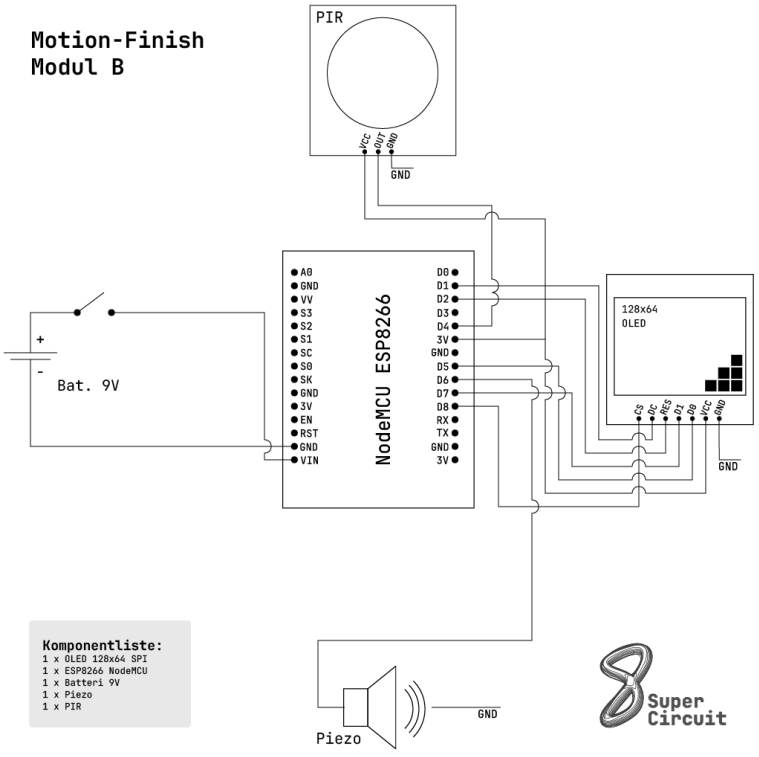
PIR-loopen i modul A sørger for å sende signalet som skal starte timeren i modul B.
Val er satt som digitalRead(inputPin) og vil være HIGH dersom PIR sensor merker bevegelse.
Når val er HIGH i modul A vil metoden esp_now_send() bli kjørt. Dette er metoden som initierer sending av signal til broadcast-adressen vi har definert tidligere, sammen med structen myData som også er definert tidligere. For å unngå at flere enn et signal blir sendt ved bevegelse fra modul A bruker vi en delay() på 10 sekunder etter at signalet er sendt. Når esp_now_send() kjøres i modul A vil OnDataSend() kjøres, som initierer esp_now_send() metoden, og deretter i modul B om signalet har kommet fram, vil OnDataRecv() kjøres.

PIR-loopen i modul B sørger for å stoppe timeren og deretter vise resultatet på OLED skjermen. Hvis PIR merker bevegelse skal det bety at bruker har nådd mål, så da setter vi boolen startSignal til å være False. På den måten, i modul B sin loop() vil ikke lenger startLoop() kjøre, siden startSignal nå ikke er True vil ikke if-sjekken gå i gjennom. Vi bruker også noen metoder som tone() og noTone() for å lage en lyd med Piezo, som signaliserer til bruker at bevegelse er oppdaget. For å få resultatet på skjermen, trenger vi ikke skrive noe ekstra kode, ettersom display.clear() kun blir kjørt i startLoop(). Resultatet vil derfor stå stillestående på skjermen helt til et nytt signal blir mottatt. piezoBuzz() er en lang metode som bestemmer et sett med toner for piezo og ser noe slikt ut:

```
void piezoBuzz() {
  if (elapsedTime < forrigeTime) {
    tone(D6, c);
    delay(50);
    noTone(D6);
    delay(50);

    tone(D6, c);
    delay(50);
    noTone(D6);
    delay(50);

    tone(D6, c);
    delay(50);
    noTone(D6);
    delay(50);

    tone(D6, c);
    delay(250);
    noTone(D6);
    delay(50);

    tone(D6, g);
    delay(300);
```

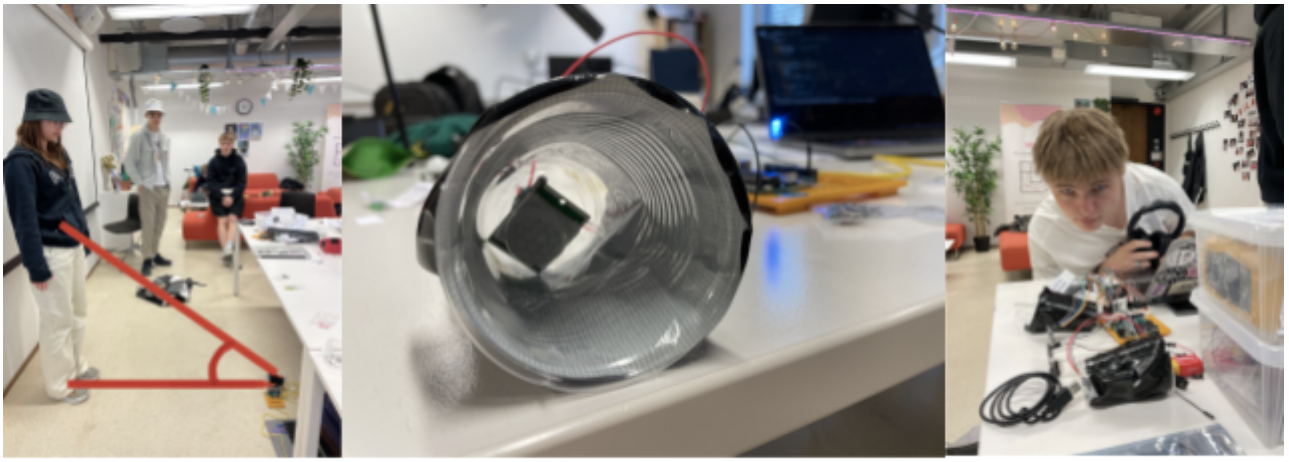
7. Circuit


Motion-Finish
Modul A
PIR
VCC
OUT
GND
GND
A0
GND
VV
S3
S2
S1
SC
S0
SK
GND
3V
EN
RST
GND
VIN
NodeMCU ESP8266
D0
D1
D2
D3
D4
3V
GND
D5
D6
D7
D8
RX
TX
GND
3V
+
-
Bat. 9V
Komponentliste:
1 x ESP8266 NodeMCU
1 x Batteri 9V
1 x Piezo
1 x PIR
Piezo
GND
Super
Circuit


Motion-Finish
Modul B
PIR
VCC
OUT
GND
GND
A0
GND
VV
S3
S2
S1
SC
S0
SK
GND
3V
EN
RST
GND
VIN
NodeMCU ESP8266
D0
D1
D2
D3
D4
3V
GND
D5
D6
D7
D8
RX
TX
GND
3V
128x64
OLED
CS
DC
RES
D1
D0
VCC
GND
GND
+
-
Bat. 9V
Komponentliste:
1 x OLED 128x64 SPI
1 x ESP8266 NodeMCU
1 x Batteri 9V
1 x Piezo
1 x PIR
Piezo
GND
Super
Circuit

8. Utfordringer

Vi sto overfor noen tekniske utfordringer underveis, både med å få komponenter til å virke grunnleggende, for å få dem til å fungere sammen samtidig og å skrive robust kode.



Ulike aktiviteter i implementasjon av kode og utforming

WiFi utfordringer

Et stort spørsmål i gruppa og en utfordring vi skjønte tidlig at vi kanskje kom til å stå over var å i det hele tatt få wifien til å fungere. Heldigvis valgte vi mikrokontroller-modellen ESP8266 som vi med hjelp av tutorials og en god del research på nett fikk til å fungere uten å bruke alt for mye tid. Den mest utfordrende delen med WiFi for oss var å gjøre research på hvordan bibliotekene og alle metodene fungerte i praksis, hvordan modul A og B skulle være forskjellige, og hvilke metoder som skulle være med i hvilken fil.

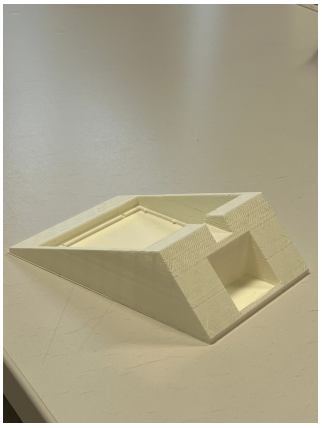
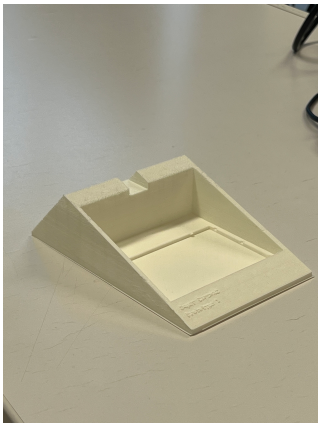
PIR utfordringer

En større utfordring enn WiFi for oss viste seg å være å få pir-sensorene til å fungere stabilt. Komponenten har i utgangspunktet ikke så mye kode relatert til seg, men den har en del innstillinger som kan justeres på selve komponenten, som sensitivitet og avstand. Det var en utfordring å justere dette til det optimale for vårt prosjekt. Vi fant også mange eksempler på PIR koder vi prøvde ut, men som ikke fungerte. Utgangspunkter som skulle etter researchen virke fungerte ikke for oss, noe som gjorde en del av oss i gruppa demotivert. Heldigvis fant vi etter hvert en kode som fungerte, uten at vi egentlig kom fram til hva som var galt med vår tidligere kode. Vi opplevde under testing at sensorene var ustabile til tider, noe vi løste med å bruke delay(), som tidligere vist i koden.

9. 3D-Printing

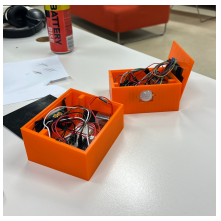
Første iterasjon

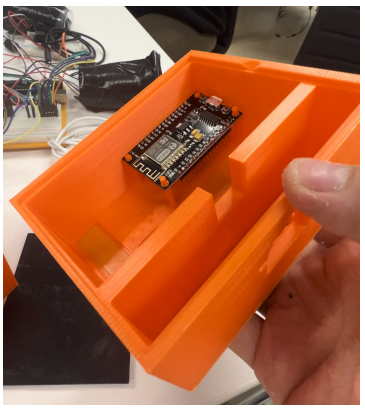
I løpet av prosjektet har vi hatt tre prototyper 3D printet. Den første var mer tenkt som et forsøk på å få til 3D-printingen og se hva som funket og ikke. Her prøvde vi å holde formen så lik ideene som vi opprinnelig ville ha dem ut fra skissene våre. Vi skjønte derimot fort at vi ville trenge mer plass i høyden om vi skulle få plass til alle komponentene.

Andre iterasjon

Selv om vi ønsket et mer kreativt formkonsept, skjønte vi fort at for å få plass til alle komponenter måtte vi øke størrelsen på kabinettet. Vi beholdt enda skråningen på taket slik at boksene kunne settes sammen og spare plass. Vi endret fargen til å være lett å se under bruk, og denne prototypen var komplett nok til å ta med ut og teste i felt. Der lærte vi fort at boksen ga PIR sensoren alt for bredt synsfelt og stoppet timeren en god stund før brukeren egentlig hadde passert den. Vi skjønte også at alle fester og skillevegger mest sto i veien for komponenter mer enn det hjalp, som vi endte med å sage bort.

Tredje iterasjon

Vi tok med oss mye kunnskap fra tidligere iterasjoner inn i den tredje og siste 3d-printer. Denne prototypen løste mange av problemene vi står overfor i andre prototype. Plass til ledninger og komponenter nådde vi gjennom å fjerne unødvendige aspekter som skillevegg og festehull. Problemet med å registrere bevegelse for tidlig løste vi med å utvide form-konseptet til å ha en utstikker/rund vegg rundt bevegelsessensorene, som en slags "kameralense". Dette designet hindret synsfeltet til PIR å være for bred. Prototypen har lokk med bedre hull på siden, som gjør lokkene enklere å ta av og på, noe som gjorde det enklere å teste og jobbe med dem. Ikke minst var denne printen lagt til rette for å sette inn magneter, og har hull på siden for implementasjonen av en av/på switch. Med generelle "quality of life" endringer og større krav-orienterte endringer endte vi opp med å bruke denne 3d-printen til vårt sluttprodukt.