Hovedkode:

```
#include "setupFunctions.h"
#include "buttonFunctions.h"
#include "scoreFunctions.h"
#include "oledFunctions.h"

/* Denne lagrer "context" som dynamisk minne på
størrelse på 50 bytes. "malloc" reserverer minnnet
til heapen som er minne som en kan bruke under kjøring
samt er dynamisk som gjør det mulig å endre verdien under
kjøring av programmet, dette brukes til å lagre
lokasjonen man får ut */
char* context = (char*)malloc(sizeof(char) * 50);

/* konverterer koordinatene fra grader og minutter om til
desimal grader */
double konverterTilDesimalgrader(double koordinat) {
  // henter ut koordinatene og fjerner desimaler, deretter deles det på 100 for å
få ut antall hele grader
  int grader = static_cast<int>(koordinat) / 100;
  // fmod deler koordinatet på 100 og gir ut restene fra dette, dette er da
desimalminutter
  double desimalMinutter = fmod(koordinat, 100.0);
  // adderer sammen helgradene og minuttene, først gjøre om desimalminutter -->
desimalgrader ved å dele på 60
  return grader + (desimalMinutter / 60.0);
}

void setup()
{
  pinMode(buttonPin, INPUT_PULLUP);
  Wire.begin();
  Serial.begin(115200);
  initialize_GPS();
  initialize_sensor();
  calibrate_accelerometer();
  internet_connection();
  ubidots_setup();
  u8g2.begin();
  // ubidots.setCallback(score_callback);
}


void ubidots_connection()
{
```

```
  //om forbindelsen mistes koble til på nytt
  if (!ubidots.connected())
  {
    ubidots.reconnect();
    ubidots.subscribeLastValue(DEVICE_LABEL_akselerometer,
VARIABLE_akselerometer_verdi);
    ubidots.subscribeLastValue(DEVICE_LABEL_gps, VARIABLE_gps);
    ubidots.subscribeLastValue(DEVICE_LABEL_gps, VARIABLE_satelitt);
    ubidots.subscribeLastValue(DEVICE_LABEL_gps, VARIABLE_satelitt_kvalitet);
    ubidots.subscribeLastValue(DEVICE_LABEL_akselerometer, VARIABLE_score);
    ubidots.subscribeLastValue(DEVICE_LABEL_gps, VARIABLE_gps_speed);
    ubidots.subscribeLastValue(DEVICE_LABEL_gps, VARIABLE_gps_speed_breach_lat);
    ubidots.subscribeLastValue(DEVICE_LABEL_gps, VARIABLE_gps_speed_breach_long);
  }
}

void gps_connection()
{
  //nye gps data mottat og leses av
  GPS.read();

  //hvis gps ikke finner posisjonen
  if (GPS.fix == false)
  {
    static unsigned long previousMillis = 0;
    const unsigned long interval = 2500; // Intervallet mellom hvert utskrift, i
millisekunder
    unsigned long currentMillis = millis();

    if (currentMillis - previousMillis >= interval) {
      // printer ut hvert 2.5 sekund dersom den ikke har en posisjon
      Serial.println("Finner ikke posisjon");
      previousMillis = currentMillis;
    }
  }
}

float acceleration_total(){ //Funksjon som henter ut akselerasjonsdata i x,y,z
retning for å lage en akselerasjonsvektor som logger total akseleraajon.
  sensor.read();
  float ax = sensor.getAccelX(); //Henter akselerasjon i x retning
  float ay = sensor.getAccelY(); //Henter akselerasjon i y retning
  float az = sensor.getAccelZ(); //Henter akselerasjon i z retning
  float accelerationValue = sqrt(sq(ax)+sq(ay)+sq(az)); // Akselerasjonsvektoren
blir laget
```

```
  return accelerationValue;
}

void ubidots_publish() {
  static unsigned long previousMillis = 0;
  const unsigned long interval = 2500; // Intervallet mellom hvert utskrift, i
millisekunder
  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;

    // Dekrypterer NMEA-setningen slik at vi får ut informasjon vi ønsker
    if (GPS.parse(GPS.lastNMEA())) {
      // fortelles oss om kvalitet, fra 1 - 6
      Serial.print("Kvalitet: ");
      Serial.println(GPS.fixquality);

      // sier hvor mange satelitter den er koblet til
      Serial.print("Antall satellitter: ");
      Serial.println((int)GPS.satellites);

      if (GPS.fix == true) {
        // Konverterer grader og minutter til desimalgrader
        double breddegrader = konverterTilDesimalgrader(GPS.latitude);
        double lengdegrader = konverterTilDesimalgrader(GPS.longitude);

        // print ut lokasjon i decimal grader
        Serial.print(breddegrader, 8);
        Serial.print(", "); // Legg til komma her
        Serial.print(lengdegrader, 8);
        Serial.println(".");

// Konverterer desimalgrader til strenger
        // lager en char med plass til 20 tegn for å lagre lokasjonen i
        char str_lat[20];
        char str_lng[20];
        // sprintf er en funksjon som formaterer data til en string
        // "%f" sier at stringen skal være et flyttall (desimaler og alt)
        // breddegrader og lengdegrader er dataen som skal formateres
        sprintf(str_lat, "%f", breddegrader);
        sprintf(str_lng, "%f", lengdegrader);

        // Kvalitet og antall satelitter
        int satelitt = (int)GPS.satellites;
```

```
        int kvalitet = GPS.fixquality;
        int gps_speed = GPS.speed;

        // Henter ut akselerasjon verdiene
        acceleration_total();
        float akselerasjon = acceleration_total(); // Example temperature value

        calculateScore(accelerationValue, elapsedGoodTime, elapsedBadTime,
elapsedVeryBadTime, elapsedStandStillTime);

        // Legge inn knappetrykk her
        if (toggleState == 1) {
          Serial.println(toggleState);
          // legger til context som "lat" og "Lng" fra breddegrad og lengdegrad
          ubidots.addContext("lat", str_lat);
          ubidots.addContext("lng", str_lng);
          // Sender context som er gps data til ubidots
          ubidots.getContext(context);
          // sender opp data til variablene inne på ubidots
          ubidots.add(VARIABLE_akselerometer_verdi, akselerasjon);
          ubidots.add(VARIABLE_score, score);
          ubidots.add(VARIABLE_gps, 1, context);
          ubidots.add(VARIABLE_satelitt, satelitt);
          ubidots.add(VARIABLE_satelitt_kvalitet, kvalitet);
          ubidots.add(VARIABLE_gps_speed, gps_speed);

          //lagre lokasjon dersom akselersjonen (i g krefter) overstiger limiten
          if (akselerasjon > 0.35)
          {
          ubidots.add(VARIABLE_gps_speed_breach_lat, GPS.latitude);
          ubidots.add(VARIABLE_gps_speed_breach_long, GPS.longitude);
          }

          // Forteller oss om data kom trygt frem til ubidots
          int publishResult = ubidots.publish(DEVICE_LABEL_gps);
          if (publishResult == 1) {
            Serial.println("Publisert på ubidots");
            ubidots.publish(DEVICE_LABEL_akselerometer);
          } else {
            Serial.print("Fikk ikke til å publisere, feilkode: ");
            Serial.println(publishResult);
          }
        }
        else
        {
```

```
          Serial.print("Publiserer ikke til Ubidots");
        }
      }
    }
  }
}



void loop()
{
  //knapp funksjon
  toggleState = toggleStateFunction();  // Update toggleState based on button
press
  // ubidots tilkobling
  ubidots_connection();
  //akselerasjon
  acceleration_total();
  //score
  restrainScore();
  calculateTime();
  calculateScore(accelerationValue, elapsedGoodTime, elapsedBadTime,
elapsedVeryBadTime, elapsedStandStillTime);
  //oled
  OLEDSkjerm();
  //gps og ubidots
  gps_connection();
  ubidots_publish();
  ubidots.loop();
}
```

Setupfunksjoner header:

```
/* Header fil som inneholder alle
de ulike setup - funksjonene */

#include <Adafruit_GPS.h>
#include <Arduino.h>
#include <UbidotsEsp32Mqtt.h>
#include "GY521.h"

#define GPSSerial Serial2
Adafruit_GPS GPS(&GPSSerial);

GY521 sensor(0x68);//I2C port for kommunikasjon

// Ubidots credentials
const char *UBIDOTS_TOKEN = "BBUS-4GTYhGckwBnhENTwIxRPtELvX1wWsR";
const char *WIFI_SSID = "NTNU-IOT";
const char *WIFI_PASS = "";

// GPS ubidots labels
const char *DEVICE_LABEL_gps = "gps_position";
const char *VARIABLE_gps = "gps";
const char *VARIABLE_satelitt = "satelitter";
const char *VARIABLE_satelitt_kvalitet = "kvalitet";
const char *VARIABLE_gps_speed = "gps_speed";
const char *VARIABLE_gps_speed_breach_lat = "gps_speed_breach_lat";
const char *VARIABLE_gps_speed_breach_long = "gps_speed_breach_long";

// akselerometer ubidots labels
const char *DEVICE_LABEL_akselerometer= "akselerometer";
const char *VARIABLE_akselerometer_verdi = "aks_verdi";
const char *VARIABLE_score = "score";

Ubidots ubidots(UBIDOTS_TOKEN);

void ubidots_setup()
{
  // ubidots setup, innebygde funksjoner som etablerer kommunikasjon med ubidots
  ubidots.setup();
  ubidots.reconnect();
  // Abonnerer på utvalgte topics inne på ubidots
  ubidots.subscribeLastValue(DEVICE_LABEL_akselerometer,
VARIABLE_akselerometer_verdi);
  ubidots.subscribeLastValue(DEVICE_LABEL_akselerometer, VARIABLE_score);
  ubidots.subscribeLastValue(DEVICE_LABEL_gps, VARIABLE_gps);
```

```
  ubidots.subscribeLastValue(DEVICE_LABEL_gps, VARIABLE_satelitt);
  ubidots.subscribeLastValue(DEVICE_LABEL_gps, VARIABLE_satelitt_kvalitet);
  ubidots.subscribeLastValue(DEVICE_LABEL_gps, VARIABLE_gps_speed);
  ubidots.subscribeLastValue(DEVICE_LABEL_gps, VARIABLE_gps_speed_breach_long);
  ubidots.subscribeLastValue(DEVICE_LABEL_gps, VARIABLE_gps_speed_breach_lat);
}

void internet_connection()
{
  // koble til internettet
  Serial.println("Initializing Wi-Fi...");
  ubidots.connectToWifi(WIFI_SSID, WIFI_PASS);
  if (ubidots.connected()) {
    // tilbakemelding på om tilkoblingen ble gjennomført
    Serial.println("Wi-Fi connected!");
  } else {
    Serial.println("Failed to connect to Wi-Fi!");
  }
}

// Funksjon for å kalibrere akselerometeret
void calibrate_accelerometer() {
  Serial.println("Kalibrerer akselerometer"); // Skriv en melding til seriell
monitor for å indikere at kalibrering starter
  int resolution = 100; // Antall målinger som skal brukes for kalibreringen
  float calx = 0, caly = 0, calz = 0; // Variabler for å samle opp
akselerometerdata

  // Les akselerometerdata 100 ganger
  for (int i = 0; i < resolution; i++) {
    sensor.read(); // Les data fra sensoren
    calx += sensor.getAccelX(); // Legg til X-aksel data
    caly += sensor.getAccelY(); // Legg til Y-aksel data
    calz += sensor.getAccelZ(); // Legg til Z-aksel data
  }
  // Beregn gjennomsnittet for hver akse og inverter det for kalibrering
  sensor.axe = -calx / resolution; // Kalibreringsverdi for X-aksen
  sensor.aye = -caly / resolution; // Kalibreringsverdi for Y-aksen
  sensor.aze = -calz / resolution; // Kalibreringsverdi for Z-aksen
  Serial.println("Akselerometer er ferdig kalibrert"); // Skriv en melding til
seriell monitor for å indikere at kalibreringen er fullført
}

// Funksjon for å initialisere GPS
void initialize_GPS() {
```

```
  GPS.begin(9600);
  /* setter opp GPS tilkobling, RMC setter opp informasjon om
  posisjon og hastighet, mens GGA setter opp informasjon om
  GPS har en fix, antall satelitter og kvalitet */
  GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
  // setter oppdateringsfrekvensen på 1Hz
  GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ);
  Serial.println("GPS er ferdig initialisert");
}

// Funksjon for å initialisere akselerometeret
void initialize_sensor() {
  delay(100); // Vent i 100 millisekunder for å sikre at sensoren er klar
  // Prøv å vekke sensoren til den svarer
  while (!sensor.wakeup()) {
    Serial.println("Kan ikke koble til GY521: Sjekk GY521 addresse (0x68/0x69)");
   // Hvis sensoren ikke svarer, skriv en feilmelding til seriell monitor
    delay(1000); // Vent i 1 sekund før du prøver igjen
  }
  sensor.setAccelSensitivity(0);   // Sett akselerometersensitiviteten til 0
  sensor.setThrottle();  // Aktiver akselerometerets throttle-funksjon
  Serial.println("Akselerometer er ferdig initialisert");  // Skriv en melding
til seriell monitor for å indikere at initialiseringen er fullført
}
```

Knappefunksjoner header:

```
const int buttonPin = 33;  // Utgang for knapp
unsigned long lastDebounceTime = 0;  // den siste gangen pin ble endret
unsigned long debounceDelay = 50;    // debouncing tid
int lastButtonState = LOW;  // den forrige avlesningen fra buttonPin
int buttonState = LOW;      // den nåværende avlesningen fra buttonPin


int toggleState = 1;  // Variabel for å holde styr på toggle tilstanden (1 eller
0)

// Funksjon for å sjekke etter gyldig knappetrykk og endre tilstanden
int toggleStateFunction() {
  static int internalToggleState = toggleState;  // Lagrer toggle tilstanden
  int reading = digitalRead(buttonPin);  // Leser tilstanden til knappen

  if (reading != lastButtonState) {
    // Tilbakestiller debounce tid
    lastDebounceTime = millis();
  }

  if ((millis() - lastDebounceTime) > debounceDelay) { //Sjekker etter gyldig
knappetrykk gitt av debounce kriterie
    if (reading != buttonState) {
      buttonState = reading;         // Endrer nåværende knappestatus
      lastDebounceTime = millis();  // Oppdater tiden for knappe endringen

      if (buttonState == HIGH) {
        internalToggleState = 1 - internalToggleState;  // Endrer tilstanden
mellom 1 og 0.
      }
    }
  }

  lastButtonState = reading;  // Oppdater siste knappestatus
  return internalToggleState;
}
```

Scorefunksjoner header:

```
//////////////////////////////////////////////////////////////////
/////////////////////////Score-Funksjon/////////////////////////
//////////////////////////////////////////////////////////////////
const float AccelerationThreshold = (0.3);  //Dette er øvre grense på hva god akselerasjon er, det vil si er du over det så blir det sett på som "dårlig" kjøring

int score = 0;  //Sjølv forklarande, dette er poengen du har til no

float accelerationValue;  //Dette er veriden til akselerasjonen som vi for fra akselerometeret

unsigned long elapsedGoodTime = 0;  //Variablen som lagrer hvor lenge bilen har kjørt "bra"
unsigned long elapsedBadTime = 0;  //Variablen som lagrer hvor lenge bilen har kjørt "dårlig"
unsigned long elapsedVeryBadTime = 0;  //Variablen som lagrer hvor lenge bilen har kjørt "ekstremt dårlig"
unsigned long elapsedStandStillTime = 0;  //Variablen som lagrer hvor lenge bilen har stått i ro

unsigned long lastGoodTime = 0;  //Variabel som husker når den sist kjørte "bra"
unsigned long lastBadTime = 0;  //Variabel som husker når den sist kjørte "dårlig"

unsigned long currentTime;  //Variabel for millis
unsigned long lastTime = 0;

bool standstill;

void gps_speed() {
  if (GPS.speed > 5) {
    standstill = false;
  }

  else{
    standstill = true;
  }
}


void restrainScore() {  //Funksjon som begrenser variabelen score slik at den ikke overstig 100 eller 0
  if (score >= 100) {
```

```
    score = 100;
  }

  if (score <= 0) {
    score = 0;
  }
}

void calculateTime(){  //Funksjon som teller hvor lenge bilen har kjørt hverken
"bra", "dårlig" eller "ekstremt dårlig"
  currentTime = millis();

  if (toggleState == 1) {  //Ser først om boksen er på
    if ((accelerationValue < AccelerationThreshold) && standstill == false) {
 //Viss bilen er under grensen så teller den hvor lenge den kjører "bra"
      elapsedGoodTime += (currentTime - lastTime);
      lastTime = currentTime;
      lastGoodTime = currentTime;

      if (elapsedGoodTime > 120031) {  //Nullstiller tellingen etter den har
pasert 2 minutter
        elapsedGoodTime = 0;
      }
    }

    else if ((accelerationValue > AccelerationThreshold) && standstill == false)
{  //Viss bilen er over grensen så teller den hvor lenge den har kjørt "dårlig"
      elapsedBadTime += (currentTime - lastTime);
      lastTime = currentTime;
      lastBadTime = currentTime;

      if (elapsedBadTime > 5031) {  //Nullstiller tellingen etter den har pasert
5 sekunder
        elapsedGoodTime = 0;
      }
    }

    else if ((accelerationValue > (AccelerationThreshold * 1.5)) && standstill ==
false) {  //Viss bilen er veldig mye over grensen så teller den hvor lenge den
har kjørt "veldig dårlig"
      elapsedVeryBadTime += (currentTime - lastTime);
      lastTime = currentTime;

      if (elapsedVeryBadTime > 2031) {  //Nullstiller tellingen etter den har
pasert 2 sekunder
```

```
        elapsedVeryBadTime = 0;
      }
    }

    else if (standstill == true) {  //Viss bilen står i ro så vil den begynne å
telle hvor lenge den står i ro
      elapsedStandStillTime = (currentTime - lastTime);
      lastTime = currentTime;

      if (elapsedStandStillTime > 240031) {  //Nullstiller telleren etter den har
pasert 4 minutter
        elapsedStandStillTime = 0;
      }
    }

    else if ((accelerationValue < AccelerationThreshold) && ((currentTime -
lastBadTime) > 90000)) {  //Viss bilen kjører "bra" og den har kjørt "bra" i 90
sekunder så vil den nullstille teller til dårlig tid
      elapsedBadTime = 0;
      elapsedVeryBadTime = 0;
    }

    else if ((accelerationValue > AccelerationThreshold) && ((currentTime -
lastGoodTime) > 10000)) {  //Viss bilen kjører "dårlig" og den har kjørt "dårlig"
i 10 sekunder så vil den nullstille teller til bra tid
      elapsedGoodTime = 0;
    }
  }

  else {
    lastTime = currentTime;  //Viss boksen er av så vil den bare oppdatere siste
tid til no tid
  }
}

void calculateScore(float accelerationValue, unsigned long elapsedGoodTime,
unsigned long elapsedBadTime, unsigned long elapsedVeryBadTime, unsigned long
elapsedStandStillTime) {  //Funksjon som oppdaterer poengen til brukeren etter
tid den har kjørt bra eller dårlig
  if (toggleState == 1) {  //Ser om boksen skal vere på
      if ((accelerationValue < AccelerationThreshold) && (elapsedGoodTime >
120000)) { //Viss bilen er under grensen og har kjørt "bra" i 2 minutter til
sammen så vil poengen til brukeren go opp med 1
        score ++;
      }
```

```
      else if (((accelerationValue > AccelerationThreshold)) && (elapsedBadTime > 5000)) {  //Viss bilen er over grensen og har kjørt "dårlig" i 5 sekunder til sammen så vil poengen til brukeren go ned med 1
        score --;
      }

      else if (((accelerationValue > (AccelerationThreshold * 1.5))) && (elapsedVeryBadTime > 2000)) {  //Viss bilen er veldig mye over grensen og har kjørt "veldig dårlig" i 2 sekunder så vil pengen til brukeren go ned med 5
        score = (score - 5);
      }
      else if (standstill == true && elapsedStandStillTime > 240000) {  //Viss bilen står i ro og har stått i ro i 4 minutter så vil poengen til brukeren go ned med 5
        score = (score - 5);
      }
  }
}
```

Oledfunksjoner header:

```
#include <U8g2lib.h>

// Setup for SH1106 display ved bruk av U8G2 biblioteket
// Initialiserer displayet SH1106 med I2C kommunikasjon og ingen reset pin
U8G2_SH1106_128X64_NONAME_F_HW_I2C u8g2(U8G2_R0, /* reset=*/ U8X8_PIN_NONE);

// Funksjon som håndterer oppdatering av OLED-skjermen
void OLEDSkjerm() {
  unsigned long currentMillis = millis(); // Får den nåværende tiden i
millisekunder
  unsigned long previousMillis = 0;  // Lagrer sist gang displayet ble oppdatert
  const long interval = 1000;  // Interval som displayet skal oppdateres på
(millisekunder)

  // Sjekker om toggleState er satt til 1
  if (toggleState == 1) {
    // Sjekker om nok tid har passert siden forrige oppdatering
    if (currentMillis - previousMillis >= interval) {
      previousMillis = currentMillis; // Oppdaterer previousMillis til nåværende
tid

      u8g2.clearBuffer(); // Tømmer bufferet for å kunne skrive ny tekst
      u8g2.setFont(u8g2_font_ncenB08_tr); // Setter fonten som skal brukes

      char displayText[30];  // Buffer for å holde tekst som skal vises
      sprintf(displayText, "Score: %d", score);  // Formaterer tekst med score-
verdi

      int textWidth = u8g2.getUTF8Width(displayText); // Får bredden på teksten
      int x = (u8g2.getDisplayWidth() - textWidth) / 2; // Beregner X-posisjon
for å sentrere teksten
      int y = u8g2.getDisplayHeight() / 2; // Beregner Y-posisjon for å plassere
teksten i midten

      u8g2.drawStr(x, y, displayText); // Tegner teksten på skjermen
      u8g2.sendBuffer(); // Sender bufferet til skjermen for visning
    }
  }
  else {
    // Hvis toggleState ikke er 1, viser en annen melding
    if (currentMillis - previousMillis >= interval) {
      previousMillis = currentMillis; // Oppdaterer previousMillis til nåværende
tid
```

```
      u8g2.clearBuffer(); // Tømmer bufferet for å kunne skrive ny tekst
      u8g2.setFont(u8g2_font_ncenB08_tr); // Setter fonten som skal brukes

      char displayText[30];  // Buffer for å holde tekst som skal vises
      sprintf(displayText, "ForceMap er av.");  // Formaterer tekst

      int textWidth = u8g2.getUTF8Width(displayText); // Får bredden på teksten
      int x = (u8g2.getDisplayWidth() - textWidth) / 2; // Beregner X-posisjon
for å sentrere teksten
      int y = u8g2.getDisplayHeight() / 2; // Beregner Y-posisjon for å plassere
teksten i midten

      u8g2.drawStr(x, y, displayText); // Tegner teksten på skjermen
      u8g2.sendBuffer(); // Sender bufferet til skjermen for visning
    }
  }
}
```