

Ryhmä 14 - Hiiri

Jäsenet/Members:

Nimi	Pääaine
Lauri Suominen	Teknillinen fysiikka (SCI)
Otto Salmela	Teknillinen fysiikka (SCI)
Valtteri Silde	Teknillinen fysiikka (SCI)
Henrik Purokoski	Matematiikka ja systeemitieteet (SCI)

SISÄLLYSLUETTELO

- [SISÄLLYSLUETTELO](#)
- [1. Johdanto](#)
- [2. Työn suunnittelu](#)
- [3. Työn toteutus](#)
 - [3.1 Kytkenäkaavio](#)
 - [3.2 Prosessi](#)
 - [3.3 Haasteet ja ratkaisut](#)
- [4. Valmis työ](#)
 - [4.1 Hiiri](#)
 - [4.2 Video](#)
 - [Valmis hiiri](#)
 - [4.3 Lähdekoodi](#)
- [5. Yhteenveto ja kehitysideat](#)
- [6. Lähteet](#)

1. Johdanto

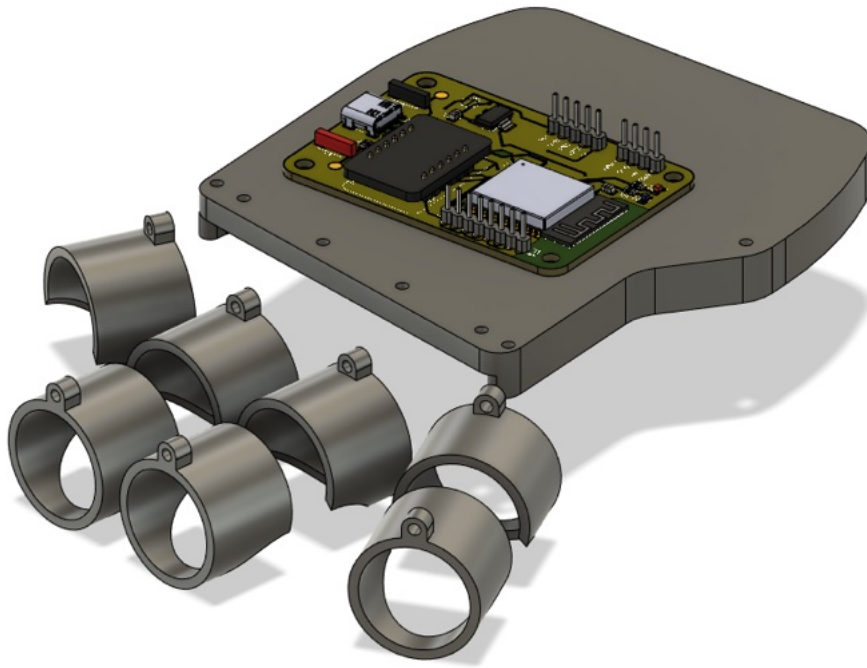
Lähdimme ideoimaan ryhmän kanssa projektia heti ensimmäisen luennon jälkeen. Tavoitteenamme oli tehdä kiinnostava projekti, joka voisi olla myös käytännöllinen - ainakin jossain määrin. Päädyimme lopulta tekemään tietokonehiiren, jota käyttäjä voisi ohjata oman kätensä liikkeellä. Järjestelmän voisi toteuttaa muutamalla eri tavalla (esimerkiksi optisella sensorilla Wii-ohjaimen tapaan), mutta päätimme yrittää toteutusta kiihtyvyysanturin avulla. Kuten pian huomasimme, projektiin liittyi monta ongelmaa, eikä alkuperäinen haaveemme käytännönläheisyydestä toteutunut täydellisesti. Opimme kuitenkin projektia tehdessämme paljon, ja lopputulos on melko hyvin toimiva (ainakin verrattaessa ensimmäisiin versioihin).

2. Työn suunnittelu

Aloitimme suunnittelun etsimällä tietoa aiheesta useasta eri lähteestä, ja löysimme useita artikkeleita ja keskusteluja missä puhuttiin paikan mittaamisesta kiihtyvyysanturilla. Tiesimme projektin toteuttamisen olevan mahdollista kiihtyvyysanturilla ja ESP32- mikrokontrollerilla, mutta epäilimme heti alkuun, saisimmeko käden liikkeen muutettua hiiren liikkeeksi tarpeeksi hyvällä tarkkuudella. Kiihtyvyysanturi mittaa vain kiihtyvyyttä, ja dataa pitää käsitellä matemaattisesti, johon liittyy epätarkkuuksia. Netistä löytämämme tieto tuki oletuksiamme: noin puolet lähteistä väitti sen olevan mahdollista vaihtelevalla tarkkuudella, ja toinen puoli sanoi sen olevan käytännössä mahdotonta juuri tarkkuuden takia. Päätimme kuitenkin yrittää projektin toteuttamista, ja otimme tavoitteeksemme tehdä hiirestä mahdollisimman tarkka. Mikäli tarkkuus olisi käyttökelvottoman huono, teimme myös varasuunnitelman: yksinkertainen peliohjain käyttämällä gyroskooppia.

Idean löytymisen jälkeen lähdimme suunnittelemaan mahdollisia muita ominaisuuksia. Jokaisessa hiiressä täytyy olla jokin tapa ohjata konetta (klikkaukset, scrollaus jne.), jotka päätimme toteuttaa sormenpäiden kosketuksilla. Alkuperäinen suunnitelma oli upottaa hanskan sormenpäihin metallipaloja, jotka yhdistäisimme ESP:hen. Sormenpäiden kosketuksesta syntyisi jännite, joka vastaisi eri sormia yhdistämällä eri hiiren toimintoa. Myöhemmin vaihdoin metallipalat paineantureihin, sillä ajattelimme sen olevan toimivampi ja siistimpi ratkaisu, kun löysimme tarpeeksi pieniä paineantureita. Työssä tarvittavat kytkennät ovat suhteellisen yksinkertaisia, joten niitä ei tarvinnut suunnitella pitkään. Suurin osa suunnitteluajasta kului 3D-printattavien osien piirtämiseen ja testaamiseen, sekä kiihtyvyysanturien selailuun. Päädyimme lopulta usean tunnin tiedonhaun jälkeen tilaamaan ICM-20948- anturin, sillä siinä oli kiihtyvyysanturin lisäksi gyroskooppi ja kompassi, ja myös sisäänrakennettu prosessori, joka tuotti suoraan lineaarista kiihtyvyydataa, josta on poistettu putoamiskiihtyvyys akselista ja orientaatiosta huolimatta. Tämän datan voimme muuttaa käden nopeudeksi, joka voidaan muuttaa hiiren liikkeeksi. Anturin mitta-alue oli myös tarpeeksi herkkä tarkoitukseemme (pienimmillään ± 2g), ja gyroskoopin ansiosta se soveltuisi myös varasuunnitelmaamme.

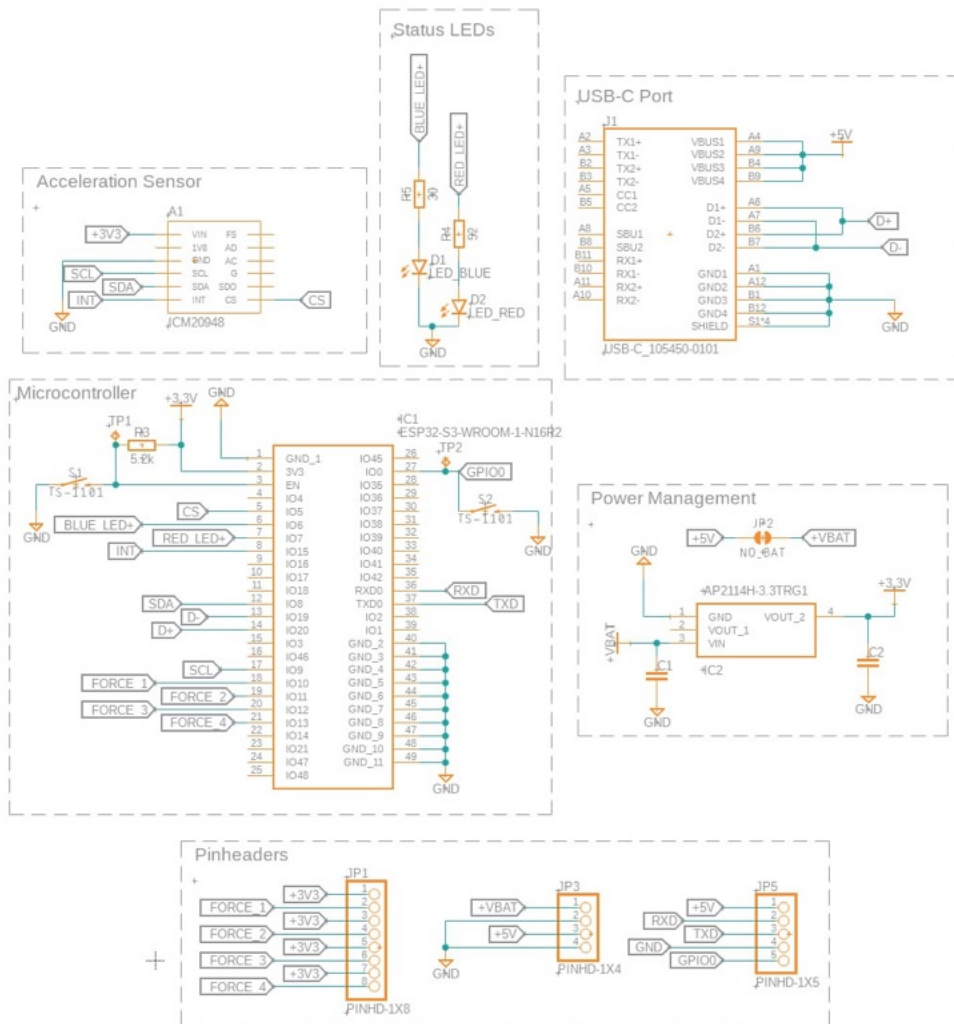
Halusimme hiirestä täysin langattoman, joten osille (anturi, ESP32 ja virtalähde) täytyi keksiä jonkinlainen alusta kädelle. Päädyimme nopeasti 3D-printaukseen, koska ryhmällämme oli siitä kokemusta, ja kaareva muoto on helpointa toteuttaa printtaamalla. Päädyimme printtaamaan yhden muovilevyn kämmenselälle, johon komponentit kiinnitetään, sekä sormille renkaat, joista johdot menisivät siististi sormenpäihin painesensoreita varten. Jotta lopputulos olisi mahdollisimman kevyt, oli suunniteltuna tilata piirilevy, jolle komponentit juotettaisiin, sillä leipälevy olisi kämmenselällä epäkäytännöllisen kokoinen. Myöhemmin alustalle lisättiin myös tuki akulle, joka antaisi tarvittavan virran hiiren toiminnalle.



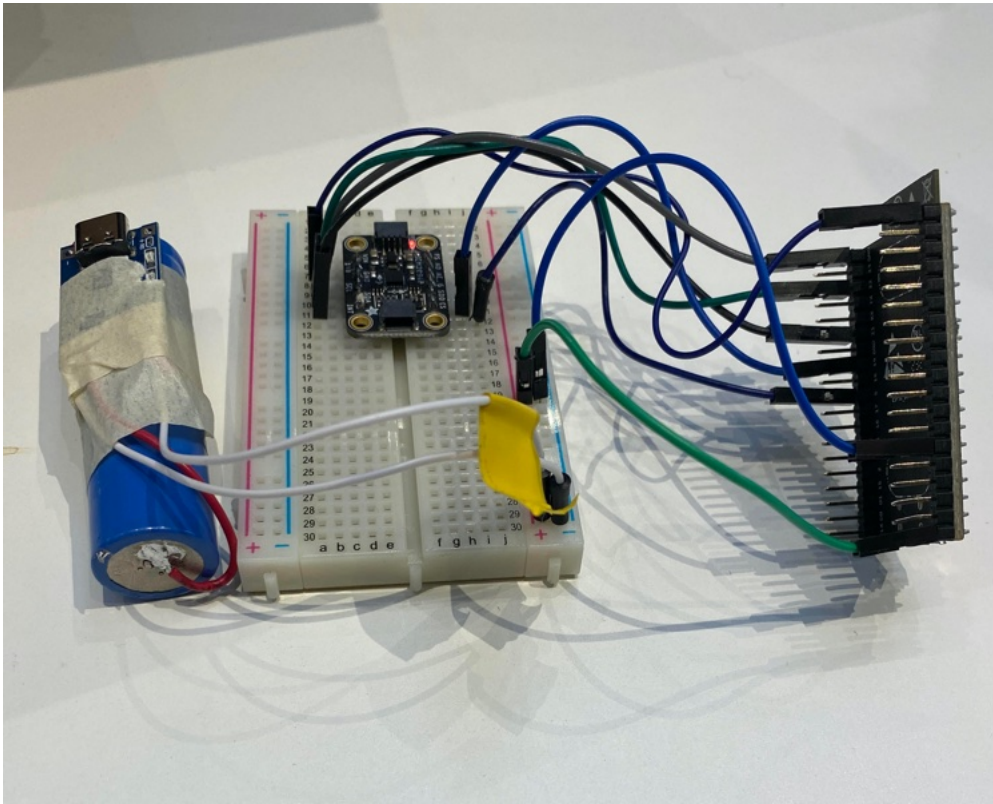
Viimeinen 3D-versio ennen toteutusta.

3. Työn toteutus

3.1 Kytcentäkaavio



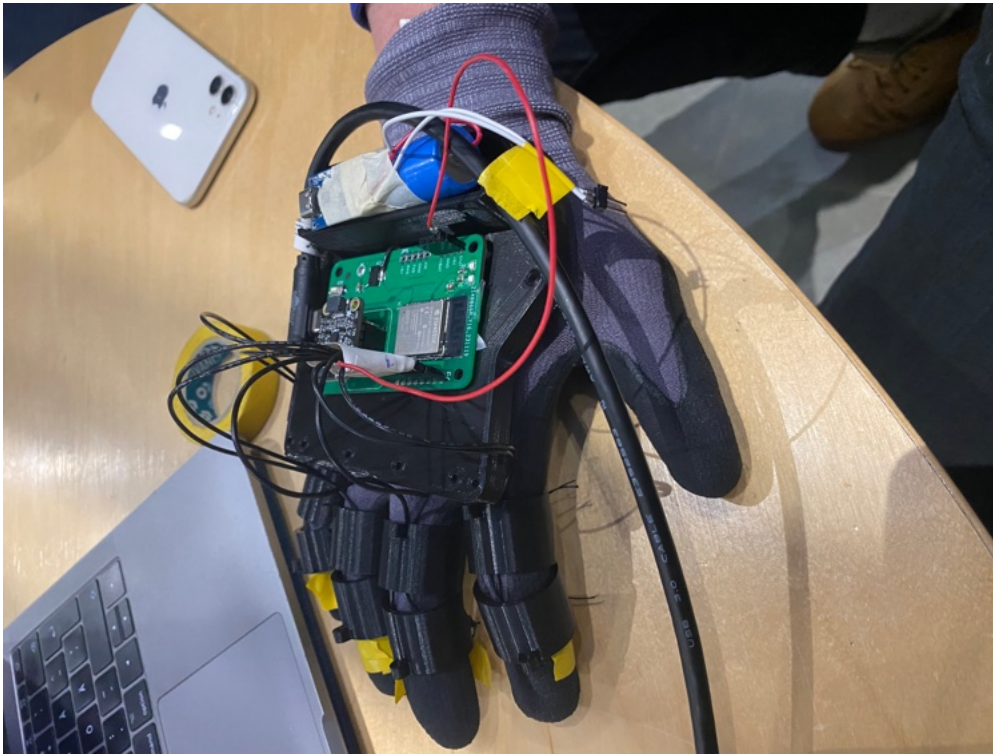
3.2 Prosesi



Ensimmäinen prototyyppi leipälaudalla

Työn rakentaminen alkoi heti kun anturi saapui pajalle. Olimme löytäneet netistä valmisteluissa muutaman koodikirjaston, joita käytimme ensimmäisen prototyypin tekemiseen. Ensimmäinen prototyyppi oli yksinkertainen leipälaudalle kytketty anturi, joka yhdistettiin ESP32:seen. Lautaa liikuttamalla pystyimme testaamaan hiirtä, ja huomasimme sen toimivan melko lailla oletetusti: yhteyden kanssa ei ollut ongelmia, mutta tarkkuus oli todella huono. Tästä alkoi monen viikon ohjelmointihelvetti, kun kokeilimme useita eri vaihtoehtoja signaalin tasoittamiselle.

Piirilevyn osat saapuivat pajalle vasta pari viikkoa ennen näytöspäivää, joten projekteille tyypillisesti suurin osa työstä kasaantui viimeiselle viikolle. Piirilevyn käyttöönotto ei onnistunut aivan niin helposti kuin olimme ajatelleet, mikä viivästytti lopullista testausta ja kasaamista. 3D-tulostuksen osalta projekti sujui ilman ongelmia: otimme mittoja ja hioimme prototyyppejä tarpeen mukaan. Lopulta muut ongelmat saatiin selätettyä, ja lopputuloksena oli (johtojen ja teippien määrää lukuunottamatta) siisti ja suhteellisen onnistunut langaton hiiri.



Viimeisiä versioita ennen deadlinea

3.3 Haasteet ja ratkaisut

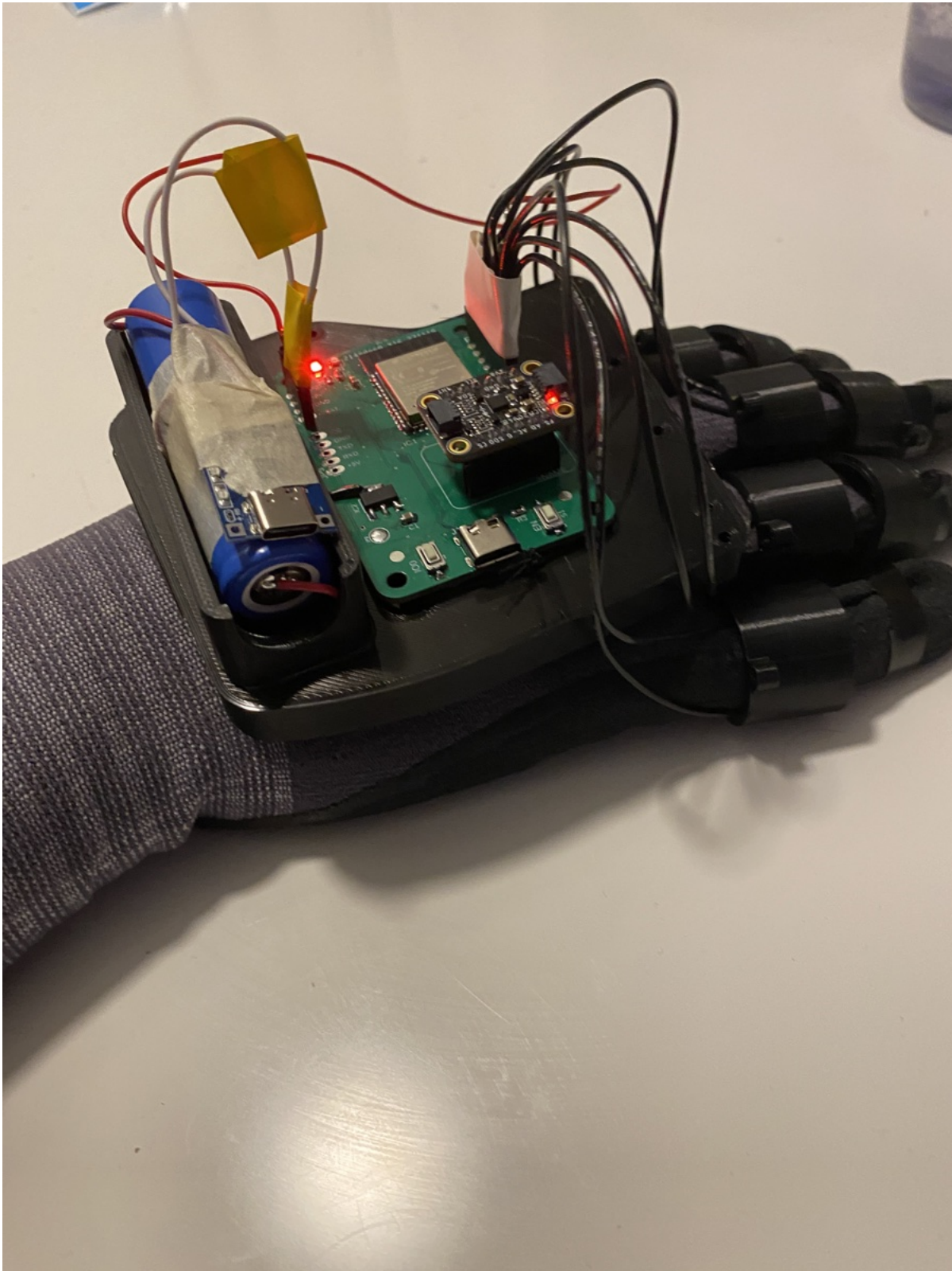
Työn haasteet olivat oletettuja: kun anturi yhdistettiin tietokoneeseen ja sitä käytettiin hiirenä, liike välittyi ajoittain hyvin, mutta ajoittain todella huonosti. Hiiri saattoi esimerkiksi välillä pysähtymisen jälkeen liikkua selvästi vastakkaiseen suuntaan, vaikka anturia ei enää liikutettu. Joskus hiiri jäi tasaiseen liikkeeseen, ja joskus se ei liikkunut ollenkaan. Kokeilimme montaa erilaista ratkaisua koodaamalla (ala- ja ylärajoja mitatuille kiihtyvyyksille, mittaustaajuuden pienentämistä ja kasvattamista, ns. rolling average- tasoitusta), kunnes lopulta pääsimme tyydyttävään lopputulokseen. Lopullisessa koodissa on erityisesti kaksi tärkeää osaa jolla driftaamista minimoidaan: jos hiiren (ohjelman laskeman) nopeuden merkki vaihtuu, ja anturi on mitannut kiihtyvyyttä molempiin suuntiin, nopeus nollataan ja mittauksia jätetään huomioimatta, kunnes hiiren nopeus on taas nollassa. Lisäksi, jos nopeuden nollautumisen jälkeen uusi nopeus on lähellä vanhaa useammassa mittauspisteessä, nopeus nollataan.

Huomasimme myös, että hiiri toimii aina aluksi melko huonosti, mutta kun sitä on käytetty vähän aikaa niin liike muuttuu tarkemmaksi. Tälle ei ole muuta selitystä kuin, että kiihtyvyyssanturin antaman datan tarkkuus on riippuvainen sen lämpötilasta, jolloin kun käytössä tämä nousee, ohjelma (joka on tehty ja kalibroitu, kun anturi on ollut pidempään käytössä) alkaa toimimaan. Tälle ei ole oikeastaan muuta ratkaisua, kuin laittaa virtaa kulkemaan anturin läpi ja odottaa, jolloin lämpötila nousee ja ohjelma alkaa toimimaan halutulla tavalla.

Piirilevyn kanssa ongelmana oli asetusten säätäminen. Koska käytimme pelkkää mikrokontrolleria, kesti pitkään löytää oikeat asetukset Arduino IDE:stä, sillä niissä oli pieniä eroja riippuen ESP:n tyypistä. Jälkiviisaana olisimme käyttäneet valmista devboardia, jolloin asetuksia ei olisi tarvinnut säätää. Jostain syystä ensimmäinen piirilevy myös käynnistyi aina uudelleen, kun yritimme käyttää I2C:tä. Ongelman syytä ei ikinä löydetty, mutta juotimme toisen levyn, joka toimi normaalisti. Tämän lisäksi oli useita muita pienempiä ongelmia (esimerkiksi USB-C toimi vain toisinpäin laittuna piirilevyn porttiin, luultavasti hieman virheellisen juottamisen takia).

4. Valmis työ

4.1 Hiiri



Valmiissa hiiressä kaikki toimii lähestulkoon täydellisesti. Alla on video, jossa esittelemme hiiren toiminnot ja tarkkuuden lyhyesti.

4.2 Video

[Valmis hiiri](#)

4.3 Lähdekoodi

Lähdekoodi

```
#include <Arduino-ICM20948.h>
#include <Arduino.h>
#include <BleMouse.h>
#include <Wire.h>

#define SDA0_Pin 8 // select ESP32 I2C pins
#define SCL0_Pin 9 // select ESP32 I2C pins

#define RED_LED_PIN 7
#define BLUE_LED_PIN 6
#define RED_LED 0
#define BLUE_LED 1
#define RED_HIGH 30
#define BLUE_HIGH 40

// Initializing the sensor and bluetooth mouse and also the settings for the sensor:
ArduinoICM20948 icm20948;
BleMouse bleMouse;
bool ICM_found = true;
ArduinoICM20948Settings icmSettings = {
    .i2c_speed = 400000,          // i2c clock speed
    .is_SPI = false,              // Enable SPI, if disable use i2c
    .cs_pin = 5,                  // SPI chip select pin
    .spi_speed = 7000000,          // SPI clock speed in Hz, max speed is 7MHz
    .mode = 1,                    // 0 = low power mode, 1 = high performance mode
    .enable_gyroscope = true,      // Enables gyroscope output
    .enable_accelerometer = true,  // Enables accelerometer output
    .enable_magnetometer =
        true,                      // Enables magnetometer output // Enables quaternion output
    .enable_gravity = true,        // Enables gravity vector output
    .enable_linearAcceleration = true, // Enables linear acceleration output
    .enable_quaternion6 = true,    // Enables quaternion 6DOF output
    .enable_quaternion9 = true,    // Enables quaternion 9DOF output
    .enable_har = false,           // Enables activity recognition
    .enable_steps = false,         // Enables step counter
    .gyroscope_frequency = 225,    // Max frequency = 225, min frequency = 1
    .accelerometer_frequency = 225, // Max frequency = 225, min frequency = 1
    .magnetometer_frequency = 70,  // Max frequency = 70, min frequency = 1
    .gravity_frequency = 225,      // Max frequency = 225, min frequency = 1
    .linearAcceleration_frequency =
        225,                      // Max frequency = 225, min frequency = 1
    .quaternion6_frequency = 225,  // Max frequency = 225, min frequency = 50
    .quaternion9_frequency = 225,  // Max frequency = 225, min frequency = 50
    .har_frequency = 225,          // Max frequency = 225, min frequency = 50
    .steps_frequency = 225         // Max frequency = 225, min frequency = 50
};

// Variables needed for the rolling average:
const int numReadings = 10;
int readIndex = 0;
float totalX = 0;
float totalY = 0;
float totalZ = 0;
float readingsX[numReadings];
float readingsY[numReadings];
float readingsZ[numReadings];

// In the setup we initialize the pins, the sensor and the rolling average data.
void setup() {
    Serial.begin(115200);
    ledcSetup(RED_LED, 5000, 8);
    ledcSetup(BLUE_LED, 5000, 8);
    ledcAttachPin(RED_LED_PIN, RED_LED);
    ledcAttachPin(BLUE_LED_PIN, BLUE_LED);
    ledcWrite(BLUE_LED, BLUE_HIGH);
}
```

```

Serial.println("Starting Wire");
Wire.begin(SDA0_Pin, SCL0_Pin);
Wire.setClock(400000);

Serial.println("Starting ICM");
delay(10);
pinMode(10, INPUT);
pinMode(11, INPUT);
pinMode(12, INPUT);
pinMode(13, INPUT);

adcAttachPin(12);

bleMouse.begin();

if (ICM_found) {
  Serial.println("ICM found");
  icm20948.init(icmSettings);
  ledcWrite(RED_LED, RED_HIGH);
  ledcWrite(BLUE_LED, 0);
}
for (int thisReading = 0; thisReading < numReadings; thisReading++) {
  readingsX[thisReading] = 0;
  readingsY[thisReading] = 0;
  readingsZ[thisReading] = 0;
}
}

// Different variables needed for the program:
long double xSpeed = 0;
long double ySpeed = 0;
int countx = 0;
int county = 0;
bool skipY = true;
bool skipZ = true;
double xLim = 0.04;
double yLim = 0.05;
double zLim = 0.05;
bool isoAccelPosX = false;
bool isoAccelNegX = false;
bool isoAccelPosY = false;
bool isoAccelNegY = false;
bool isOn = false;
int prevOnOff = 0;

void loop() {
  // Get the value from the pressure sensor in the pinky finger (on/off)
  int onOffValue = analogRead(10);
  if (onOffValue > 3500 && prevOnOff < 3500) {
    isOn = !isOn;
    if (isOn) { // If we have turned the glove on, we move the mouse to the center of the screen. This is
computer specific.
      if (bleMouse.isConnected()) {
        for (int i = 0; i < 100; i++) {
          bleMouse.move(100, 100);
        }
        delay(100);
        for (int i = 0; i < 100; i++) {
          bleMouse.move(-32, -20);
        }
      }
    }
  } else { // If we have turned the glove off, we reset the rolling average data
    for (int thisReading = 0; thisReading < numReadings; thisReading++) {
      readingsZ[thisReading] = 0;
      totalZ = 0;
    }
    for (int thisReading = 0; thisReading < numReadings; thisReading++) {
      readingsY[thisReading] = 0;
      totalY = 0;
    }
  }
}

```



```

    for (int thisReading = 0; thisReading < numReadings; thisReading++) {
        readingsX[thisReading] = 0;
        totalY = 0;
    }
}
delay(200);
}
prevOnOff = onOffValue;
long start = micros();
if (ICM_found && isOn) {
    // The variables that hold the rolling average linear acceleration data.
    // x-axis of the sensor is ultimately not used in this program
    float x = 0;
    float y = 0;
    float z = 0;
    if (!skipY && !skipZ) { // If we don't want to skip y or z data, we read new data
        totalX = totalX - readingsX[readIndex];
        totalY = totalY - readingsY[readIndex];
        totalZ = totalZ - readingsZ[readIndex];

        // read from the sensors:
        icm20948.task();
        while (!icm20948.linearAccelDataIsReady()) {
            icm20948.task();
            continue;
        }
        float xa, ya, za;
        icm20948.readLinearAccelData(&xa, &ya, &za);
        if (xa < xLim && xa > -xLim) {
            xa = 0;
        }
        if (za < zLim && za > -zLim) {
            za = 0;
        }
        if (ya < yLim && ya > -yLim) {
            ya = 0;
        }
        readingsX[readIndex] = xa;
        readingsY[readIndex] = ya;
        readingsZ[readIndex] = za;

        // add the reading to the total:
        totalX = totalX + readingsX[readIndex];
        totalY = totalY + readingsY[readIndex];
        totalZ = totalZ + readingsZ[readIndex];

        // advance to the next position in the array:
        readIndex++;

        // if we're at the end of the array...
        if (readIndex >= numReadings) {
            // ...wrap around to the beginning:
            readIndex = 0;
        }

        x = totalX / numReadings; // the average
        y = totalY / numReadings; // the average
        z = totalZ / numReadings; // the average
    } else { // If we want to skip y or z data, we reset the rolling average data and read linear
        // acceleration data until it is sufficiently close to zero
        for (int thisReading = 0; thisReading < numReadings; thisReading++) {
            readingsZ[thisReading] = 0;
            totalZ = 0;
        }
        for (int thisReading = 0; thisReading < numReadings; thisReading++) {
            readingsY[thisReading] = 0;
            totalY = 0;
        }
        icm20948.task();
        while (!icm20948.linearAccelDataIsReady()) {
            icm20948.task();

```

```

        continue;
    }
    float xa, ya, za;
    icm20948.readLinearAccelData(&xa, &ya, &za);
    if (za < zLim * 3 && za > -zLim * 3) { // the times 3 coefficient was found through testing
        for (int thisReading = 0; thisReading < numReadings; thisReading++) {
            readingsZ[thisReading] = 0;
            totalZ = 0;
        }
        skipZ = false;
    }
    if (ya < yLim * 3 && ya > -yLim * 3) {
        for (int thisReading = 0; thisReading < numReadings; thisReading++) {
            readingsY[thisReading] = 0;
            totalY = 0;
        }
        skipY = false;
    }
}

// We check if there has been a "big" acceleration, such as moving your hand
// so that the cursor moves from the left side of the screen to the right. This is also computer specific
double isoAccelRaja = 0.05;
if (y > isoAccelRaja) {
    isoAccelPosX = true;
} else if (y < -isoAccelRaja) {
    isoAccelNegX = true;
}

if (z > isoAccelRaja) {
    isoAccelPosY = true;
} else if (z < -isoAccelRaja) {
    isoAccelNegY = true;
}

// Integration of acceleration data to speed:
double time = (micros() - start) / 1000000.0;
long double xNewSpeed = xSpeed + y * time; // The y-axis of the sensor gives us x-axis cursor movement on
the screen
long double yNewSpeed = ySpeed + z * time; // The z-axis of the sensor gives us y-axis cursor movement on
the screen

/*
Here we check if the speed has changed signs and there has been a big acceleration and a big deceleration,
and we reset the speed, and skip data until it is sufficiently close to zero. We do this because the sensor
almost
always gives a bigger deceleration curve, resulting in drifting to the opposite way of the movement at the
end.
*/
if ((xNewSpeed > 0 && xSpeed < 0) || (xNewSpeed < 0 && xSpeed > 0) && (isoAccelNegX && isoAccelPosX)) {
    skipY = true;
    xNewSpeed = 0;
    isoAccelNegX = false;
    isoAccelPosX = false;
}

if ((yNewSpeed > 0 && ySpeed < 0) || (yNewSpeed < 0 && ySpeed > 0) && (isoAccelNegY && isoAccelPosY)) {
    skipZ = true;
    yNewSpeed = 0;
    isoAccelNegY = false;
    isoAccelPosY = false;
}

long double speedLim = 0.0000000001;
// Here we check if the new speed is sufficiently close to the old speed, and if that has happened enough
times
// in a row, we reset the speed. If not, we set the speed as the new speed. The speed limit and
coefficients were
// found through testing and seem to be computer specific.
if ((xNewSpeed < (xSpeed + 1000 * speedLim)) && (xNewSpeed > (xSpeed - 1000 * speedLim))) {
    countx++;
} else {

```

```

    countx = 0;
}

if ((yNewSpeed < (ySpeed + speedLim)) && (yNewSpeed > (ySpeed - speedLim))) {
    county++;
} else {
    county = 0;
}
if (countx > 5) {
    xSpeed = 0;
    countx = 0;
} else {
    xSpeed = xNewSpeed;
}
if (county > 10) {
    ySpeed = 0;
    county = 0;
} else {
    ySpeed = yNewSpeed;
}

// Read the pressure sensor values from the ring, middle and index fingers.
int scrollValue = analogRead(11);
int rightClickValue = analogRead(12);
int leftClickValue = analogRead(13);
if (bleMouse.isConnected()) {
    if (leftClickValue == 4095) { // If the pressure sensor is pressed sufficiently, left click.
        bleMouse.press();
    } else {
        if (bleMouse.isPressed()) {
            bleMouse.release();
            delay(200);
        }
    }
    if (rightClickValue == 4095) { // If the pressure sensor is pressed sufficiently, right click.
        bleMouse.press(2); // 2 = Right click
    } else {
        if (bleMouse.isPressed(2)) {
            bleMouse.release(2);
            delay(200);
        }
    }
}

// Converting the speeds to cursor movement:
int mouseMultiplier = 800;
double mouseXSpeed = -xSpeed;
double mouseYSpeed = -ySpeed;
(mouseXSpeed > 0.0005 || mouseXSpeed < -0.0005) ? mouseXSpeed
                                                : mouseXSpeed = 0;
(mouseYSpeed > 0.0005 || mouseYSpeed < -0.0005) ? mouseYSpeed
                                                : mouseYSpeed = 0;
if (scrollValue != 4095) { // If we don't want to scroll, we move the mouse
    bleMouse.move(mouseXSpeed * mouseMultiplier,
                  mouseYSpeed * mouseMultiplier);
} else {
    // If we want to scroll, we check normal acceleration data giving us the orientation of the hand.
    // If the magnitude of the acceleration is sufficiently large, we scroll up or down.
    while (scrollValue > 3000) { // We do this until the pressure sensor is released
        icm20948.task();
        if (icm20948.accelDataIsReady()) {
            float gx, gy, gz;
            icm20948.readAccelData(&gx, &gy, &gz);
            if (gx < 0.2 && gx > -0.2) {
                gx = 0;
            }
            if (gx > 0) {
                bleMouse.move(0, 0, -1);
                delay(60);
            } else if (gx < 0) {
                bleMouse.move(0, 0, 1);
                delay(60);
            }
        }
    }
}

```

```

    }
  }
  // New scroll value
  scrollValue = analogRead(11);
  icm20948.task();
}
// After scrolling ends, we reset the rolling average data, so that the cursor is stationary
for (int thisReading = 0; thisReading < numReadings; thisReading++) {
  readingsZ[thisReading] = 0;
  totalZ = 0;
}
for (int thisReading = 0; thisReading < numReadings; thisReading++) {
  readingsY[thisReading] = 0;
  totalY = 0;
}
for (int thisReading = 0; thisReading < numReadings; thisReading++) {
  readingsX[thisReading] = 0;
  totalX = 0;
}
delay(1000);
}
}
}
}

```

5. Yhteenveto ja kehitysideat

Alun vaikeuksien jälkeen olimme yllättyneitä siitä, kuinka hyvin hiiri toimi lopulta. Liike on enimmäkseen tasaista, ja vaikka tarkkuus ei riitä osumaan kaikista pienimpiin kohtiin näytöllä, esimerkiksi Windows- koneen asetusten muuttaminen onnistui ilman vaikeuksia. Isommat liikkeet ovat tarkkoja ja niissä ei tapahdu juuri ollenkaan ylimääristä liikettä, mikä oli suuri ongelma kehityksen alkuvaiheessa. Toisaalta ohjelmoitujen rajojen takia pienet ja hitaat liikkeet ovat hieman epätarkkoja, tai eivät rekisteröidy ollenkaan. Huomasimme myös, että hiiren parametreja täytyy myös muuttaa riippuen koneesta, johon se on yhdistettynä.

Hiirestä voisi todennäköisesti saada vielä jonkin verran paremman käyttämällä ohjelmointiin useita tunteja lisää, mutta suurin virhe tulee kiihtyvyyssdatan mittauksessa ja prosessoinnissa. Tätä on vaikeaa välttää, ellei hankkisi vielä tarkempaa kiihtyvyyssanturia (ja mahdollisesti kehittäisi omaa algoritmia, mutta on vaikea sanoa, toimisiko se paremmin). Meidän tiedoilla ja taidoillamme hiiren tarkkuus ei kuitenkaan enää parane, joten täydelliseen käytännöllisyyteen emme päässeet, mutta lopullinen versio täyttää tavoitteemme hyvin.

6. Lähteet

<https://github.com/T-vK/ESP32-BLE-Mouse> - Hiiren koodin pohjana toimiva kirjasto

https://github.com/isouriadakis/Arduino_ICM20948_DMP_Full-Function - ICM-20948 komponentille valmiiksi tehty kirjasto, josta saimme lineaarikiihtyvyyssdatan.

Lisäksi useita nettilähteitä hyödynnetty kiihtyvyyssanturin etsinnässä ja piirilevyn teossa