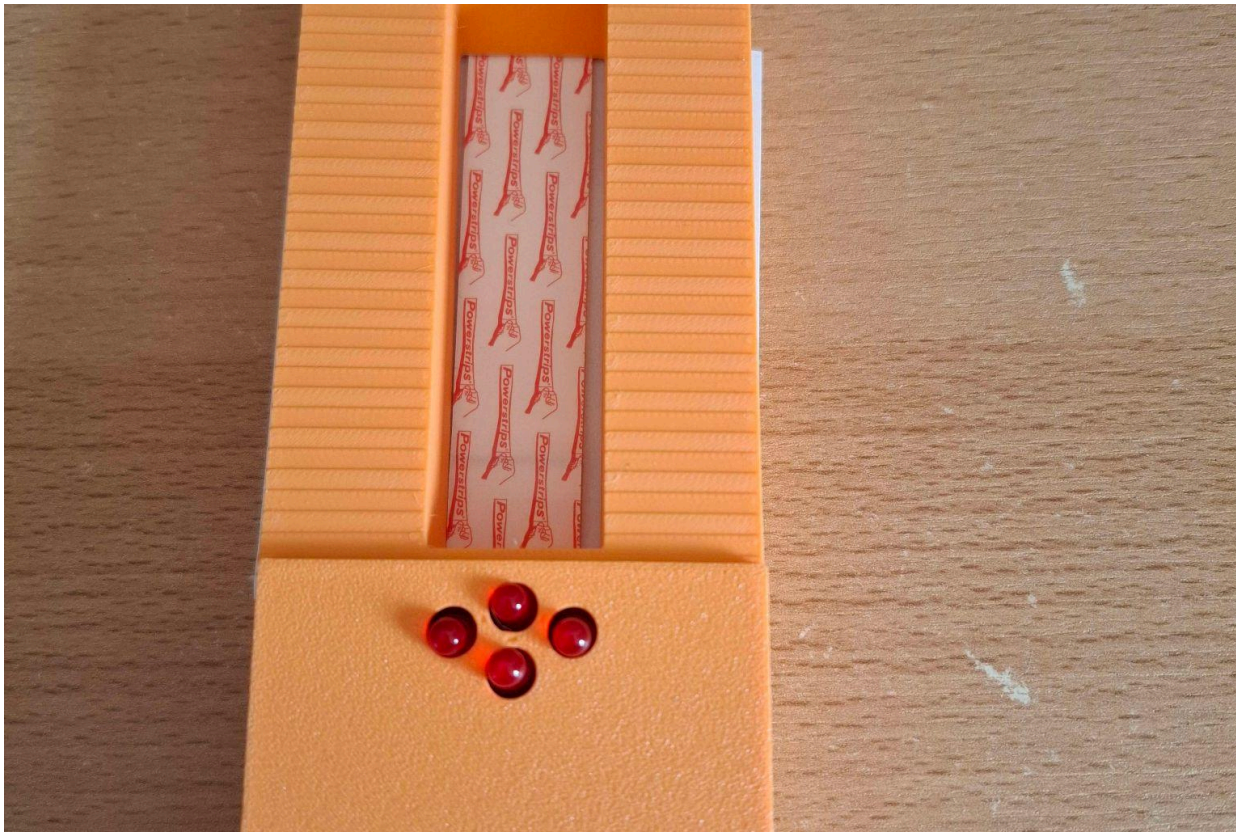


Projekt Cross Innovation Class

Tesa Mounting Tool: Orientierungserkennung und Bewegungskontrolle

05.08.2025
Projekt Intelligente Umgebungen

Dennis La - 105807
Studiengang Smart Technology
Sommersemester 2025
Fachhochschule Wedel

Einleitung

Projektbeschreibung

Im Rahmen der Cross Innovation Class wurde neben dem ModRail noch ein intelligentes Orientierungs- und Bewegungskontrollsystem für das Unterstützen des präzisen Aufklebens von Tesa Powerstrips entwickelt. Das System basiert auf einem ESP32-Wroom-32 in Kombination mit einem BNO055-Orientierungssensor. Diese arbeiten im Zusammenspiel auf einer eigenen PCB.

Das Gerät bietet zwei Modi an und fungiert als digitale Wasserwaage mit erweiterten Funktionen zur Bewegungsführung. Es erkennt Abweichungen von der gewünschten Orientierung und gibt über LEDs visuelle Korrektursignale aus. Ein integrierter Button ermöglicht dabei den Wechsel zwischen den Modi und das Zurücksetzen der Referenzorientierung.

Motivation

Das Projekt entstand aus der praktischen Notwendigkeit, Tesa Powerstrips präzise und gerade an Wänden zu befestigen. Herkömmliche Wasserwaagen bieten nur statische Levelinformationen, während dieses System zusätzlich dynamische Bewegungsführung für horizontale Verläufe ermöglicht. Dies erhöht die Präzision beim Aufkleben und reduziert Fehlplatzierungen. Im Gespräch bei Tesas Zentrale in Norderstedt ist genau das Problem im Gespräch mit dem Cross Innovation Class Team aufgeworfen worden und es wurde bestätigt, dass es ein Produkt dieser Art noch nicht gibt. Die Entwicklung erfolgte im Kontext der Cross Innovation Class mit dem Ziel, moderne Sensortechnologie für alltägliche Anwendungen nutzbar zu machen und dabei praxistaugliche Lösungen zu entwickeln.

Benutzerhandbuch

Bedienungsanleitung

Inbetriebnahme

1. Einschalten: Das Gerät über USB-Anschluss mit Strom versorgen
2. Kalibrierung: Das Gerät zwei Sekunden in der gewünschten Ausgangsposition halten
3. Bereitschaft: Automatische Referenzsetzung und Start im LEVEL_MODE

Betriebsmodi

LEVEL_MODE (Wasserwaage):

- Zeigt Abweichungen von der horizontalen/vertikalen Ausrichtung an
- Ideal für statisches Leveling vor der Montage
- Toleranz: ±5° für LED-Aktivierung

GUIDE_MODE (Bewegungsführung):

- Führt den Benutzer bei horizontalen Bewegungen entlang der Wand
- Erkennt vertikale Bewegungstendenzen und korrigiert diese
- Toleranz: ±2° für erhöhte Präzision bei der Bewegungsführung

Bedienung

Button-Funktionen:

- Kurzer Druck(<1s): Referenz neu setzen in aktueller Position
- Langer Druck(>1s): Wechsel zwischen LEVEL_MODE und GUIDE_MODE

LED-Anzeigen:

- LED_UP: Gerät nach oben neigen/ Bewegung nach oben korrigieren
- LED_DOWN: Gerät nach unten neigen/ Bewegung nach unten korrigieren
- LED_LEFT: Gerät nach links neigen/ nach Links korrigieren

Anwendungsszenarien

Szenario 1: Einzelne Powerstrip-Montage

1. Gerät in gewünschter Position an die Wand halten
2. Kurzen Button-Druck für Referenzsetzung
3. LEDs zeigen Korrekturen für perfekte Ausrichtungen an
4. Powerstrip aufkleben, wenn alle LEDs aus sind

Szenario 2: Horizontale Reihe von Powerstrips

1. Ersten Powerstrip mit LEVEL_MODE montieren
2. Langen Button-Druck für GUIDE_MODE
3. Gerät horizontal entlang der Wand bewegen
4. LEDs führen zu gerader horizontaler Bewegung
5. Weitere Powerstrips in gleicher Linie montieren

Software Implementation

Button Handling

Erklärung des Button-Handlings:

- Unterscheidet zwischen Drücken und Loslassen
- Zeitbasierte Funktion: Druckdauer bestimmt die Aktion
- Entprellung: 300ms Wartezeit verhindert mehrfache Auslösung
- Pull-up Logik: LOW = gedrückt, HIGH = nicht gedrückt

Modularisierung:

- Button-Handling: Kurz-/Langdruck-Erkennung mit Entprellung
- Sensor-Interface: Quaternion-Verarbeitung und Euler-Konversion
- LED-Controller: Modi-spezifische Steuerungslogik
- Bewegungsanalyse: Trend-Erkennung und Filterung

```
// Button-Status-Variablen (global)
unsigned long buttonPressTime = 0;
bool buttonPressed = false;

// In loop()-Funktion:
bool buttonState = digitalRead(BUTTON_PIN) == LOW; // Invertiert wegen
Pull-up

if (buttonState && !buttonPressed) {
  // Button wurde gerade gedrückt
  buttonPressTime = millis();
  buttonPressed = true;
}

if (!buttonState && buttonPressed) {
  // Button wurde losgelassen - Aktion bestimmen
  unsigned long pressDuration = millis() - buttonPressTime;

  if (pressDuration < 1000) {
    // Kurzer Druck: Referenz neu setzen
    imu::Vector<3> eulerRef = getRemappedEuler();
```

```
    refYaw = eulerRef.x();
    refRoll = eulerRef.y();
    refPitch = eulerRef.z();
    Serial.println("Referenz zurückgesetzt.");
  } else {
    // Langer Druck: Modus wechseln
    currentMode = (currentMode == LEVEL_MODE) ? GUIDE_MODE : LEVEL_MODE;
    Serial.println(currentMode == LEVEL_MODE ? "Modus: LEVEL_MODE" :
"Modus: GUIDE_MODE");
  }

  buttonPressed = false;
  delay(300); // Entprellzeit
}
```

Orientierungsberechnung

Eulerwinkel Extraktion

Der Sensor liefert seine absolute Orientierung im Raum als Quaternion. Diese wird in Eulerwinkel umgerechnet, um die Rotation entlang der drei Hauptachsen anschaulich darzustellen:

- Yaw: Rotation um vertikale Achse (Uhrzeigersinn/Gegenuhrzeigersinn)
- Roll: Rotation um horizontale Achse (seitliche Neigung)
- Pitch: Rotation um Tiefenachse (oben/unten Neigung)

Die Funktion **getRemappedEuler()** liest das aktuelle Quaternion, passt es an die vertikale Montageposition des Sensors an und gibt daraus berechnete Eulerwinkel in Grad zurück:

```
imu::Vector<3> getRemappedEuler() {
  // Quaternion vom Sensor lesen und normalisieren
  imu::Quaternion q = bno.getQuat();
  q.normalize();

  // Achsen-Remapping für vertikale Sensor-Montage
  float temp = q.x();
  q.x() = -q.y();      // X-Achse wird zu negativer Y-Achse
  q.y() = temp;        // Y-Achse wird zu X-Achse
  q.z() = -q.z();      // Z-Achse wird negiert

  // Quaternion zu Euler-Winkeln konvertieren
  imu::Vector<3> euler = q.toEuler();
```

```
  // Radiant zu Grad konvertieren und Vorzeichen anpassen
  return imu::Vector<3>(
    -180 / M_PI * euler.x(),  // Yaw in Grad
    -180 / M_PI * euler.y(),  // Roll in Grad
    -180 / M_PI * euler.z()   // Pitch in Grad
  );
}
```

Warum Quaternions?

Quaternionen sind eine mathematische Repräsentation von Rotation im dreidimensionalen Raum. Der BNO055 verwendet intern Quaternions statt Eulerwinkel, weil sie ein wichtigen Vorteil bietet: Kein Gimbal Lock.

Bei der Verwendung von Eulerwinkel wurde in der Entwicklung deutlich, dass bei der vertikalen Nutzung des BNO055 ein Problem entsteht. Der sogenannte Gimbal Lock. Eine Achse wird durch eine vorherige Rotation so ausgerichtet, dass zwei Rotationen um verschiedene Achsen den gleichen Effekt haben, sodass genau ein Freiheitsgrad verloren geht. Das führt zu komischen Sprungeffekte und unzuverlässiger Orientierung. Zur Weiterverarbeitung wird die Quaternion anschließend in Eulerwinkel umgerechnet, da so die Interpretation und Weiterverarbeitung vereinfacht und verdeutlicht wird.

Erklärung des Remappings

Das Achsen-Remapping ist notwendig, da der BNO055-Sensor vertikal montiert ist, aber standardmäßig für horizontale Montage kalibriert ist. Durch die Transformation werden die Sensorachsen an die physische Orientierung angepasst:

Sensor-Achse Standard	**Nach Remapping**
X (rechts/links)	-Y (oben/unten)
Y (vor/zurück)	X (rechts/links)
Z (oben/unten)	-Z (invertierte Tiefe)

Referenzsystem

Beim Start des Systems erfolgt eine automatische Kalibrierung der Sensoren durch die

interne Firmware des BNO055. Da der Sensor jedoch vertikal montiert ist, wurde eine initiale Kalibrierung in genau dieser vertikalen Ausrichtung durchgeführt und dauerhaft im EEPROM gespeichert. So kann der Sensor nach einem Neustart direkt mit gültigen Kalibrierwerten arbeiten, ohne dass erneut eine Kalibrierung notwendig ist. Dies stellt sicher, dass die ermittelten Orientierungsdaten auch bei wiederholtem Hochfahren des Systems stabil und präzise sind. Zusätzlich verfügt das System über eine manuelle Referenzrücksetzung, die per Tastendruck ausgelöst wird. Durch diesen Knopfdruck werden Position und Geschwindigkeit zurückgesetzt, sodass alle weiteren Bewegungsdaten relativ zur aktuellen Lage erfasst werden. Auf diese Weise lassen sich wiederholbare Messungen durchführen, bei denen die Ausgangsposition exakt definiert ist.

Bewegungsanalyse

Trend Erkennung

- Lineare Beschleunigung der Y-Achse (vertikal) wird überwacht
- Gleitender Durchschnitt mit 70%/30% Gewichtung für Glättung
- Schwellwerte zur Unterscheidung zwischen Bewegung und Stillstand

```
// Beschleunigungswerte vom Sensor lesen
imu::Vector<3> accel = bno.getVector(Adafruit_BNO055::VECTOR_LINEARACCEL);
float accelY = accel.y(); // Y-Achse = vertikal bei vertikaler Haltung

// Gleitender Durchschnitt für Trendglättung
movementTrend = movementTrend * 0.70 + accelY * 0.30;
```

Erklärung des Filters

Der gleitende Durchschnitt kombiniert 70% des vorherigen Trends mit 30% des aktuellen Messwerts. Dies bewirkt:

- Glättung von Rauschen: Kurzzeitige Schwankungen werden herausgefiltert
- Responsive Trenderfassung: Echte Bewegungen werden schnell erkannt
- Stabilität: Verhindert Flackern der LED-Anzeigen bei kleinen Bewegungen

Schwellwert Logik

```
bool verticalMovement = abs(movementTrend) > 0.05; // m/s² Schwelle

if (verticalMovement) {
```

```
  // Vertikale Bewegung erkannt - Korrektur anzeigen
  if (movementTrend > 0.08) {
    digitalWrite(LED_DOWN, HIGH);  // Nach oben bewegt -> nach unten
korrigieren
  } else if (movementTrend < -0.08) {
    digitalWrite(LED_UP, HIGH);    // Nach unten bewegt -> nach oben
korrigieren
  }
}
```

Bewegungsintegration: Die 0.05 m/s² Schwelle unterscheidet zwischen Stillstand und aktiver Bewegung, während die 0.08 m/s² Schwelle bestimmt, wann eine Korrektur-LED aktiviert wird. Diese Hysterese verhindert unstete Anzeigen bei Bewegungen nahe der Schwelle.

LED-Steuerungslogik und Modi

LEVEL_MODE Prioritäten

1. Pitch (oben/unten): Höchste Priorität
2. Roll (links/rechts): Mittlere Priorität, überschreibt Yaw
3. Yaw (Drehung): Niedrigste Priorität, nur wenn Roll im Toleranzbereich

```
if (currentMode == LEVEL_MODE) {
  // Pitch = Neigung nach oben/unten (höchste Priorität)
  if (pitchDiff > ANGLE_TOLERANCE_DEG)
    digitalWrite(LED_UP, HIGH);
  else if (pitchDiff < -ANGLE_TOLERANCE_DEG)
    digitalWrite(LED_DOWN, HIGH);

  // Roll = Neigung nach links/rechts (überschreibt Yaw)
  if (rollDiff > ANGLE_TOLERANCE_DEG)
    digitalWrite(LED_RIGHT, HIGH);
  else if (rollDiff < -ANGLE_TOLERANCE_DEG)
    digitalWrite(LED_LEFT, HIGH);
  // Yaw = Drehung um vertikale Achse (nur wenn Roll im Toleranzbereich)
  else if (yawDiff > ANGLE_TOLERANCE_DEG)
    digitalWrite(LED_LEFT, HIGH);
  else if (yawDiff < -ANGLE_TOLERANCE_DEG)
```

```
    digitalWrite(LED_RIGHT, HIGH);
}
```

Erklärung der Prioritätslogik

- Pitch wird immer angezeigt, da oben/unten-Neigung am kritischsten ist
- Roll überschreibt Yaw durch else if-Struktur, da seitliche Neigung wichtiger als Drehung ist
- Yaw wird nur angezeigt, wenn Roll im Toleranzbereich liegt

GUIDE_MODE Prioritäten

1. Vertikale Bewegung: Höchste Priorität für horizontale Führung
2. Yaw-Stabilität: Richtung halten während Bewegung
3. Roll/Pitch: Orientierungsstabilität bei ruhiger Bewegung

```
else {
  // GUIDE_MODE: Horizontale Bewegungsführung

  // Hauptziel: Horizontale Bewegung (kein vertikaler Trend)
  bool verticalMovement = abs(movementTrend) > 0.05; // m/s² Schwelle

  if (verticalMovement) {
    // Vertikale Bewegung erkannt - Korrektur anzeigen
    if (movementTrend > 0.08) {
      digitalWrite(LED_DOWN, HIGH);  // Nach oben bewegt -> nach unten
korrigieren
    } else if (movementTrend < -0.08) {
      digitalWrite(LED_UP, HIGH);    // Nach unten bewegt -> nach oben
korrigieren
    }
  }

  // Orientierungs-Stabilität (sekundär)
  if (!verticalMovement) {
    // Nur wenn keine vertikale Bewegung, dann Orientierung prüfen
```

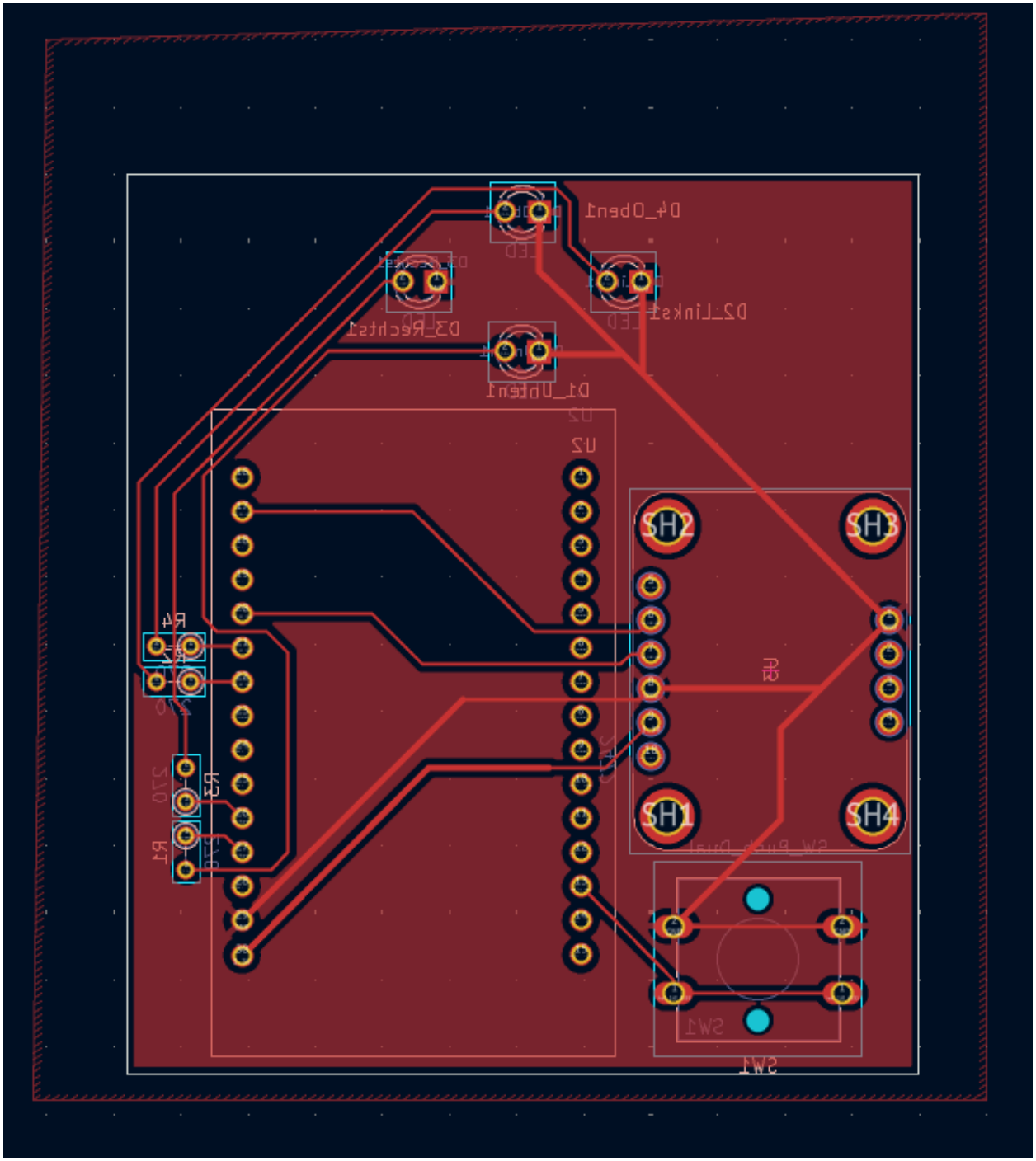
```
    // Yaw-Kontrolle (Richtung halten) - höchste Priorität bei Orientierung
    if (abs(yawDiff) > MOVEMENT_TOLERANCE_DEG) {
      if (yawDiff > MOVEMENT_TOLERANCE_DEG)
        digitalWrite(LED_LEFT, HIGH);
      else if (yawDiff < -MOVEMENT_TOLERANCE_DEG)
        digitalWrite(LED_RIGHT, HIGH);
    }
    // Roll-Kontrolle (seitliche Neigung)
    else if (abs(rollDiff) > MOVEMENT_TOLERANCE_DEG) {
      if (rollDiff > MOVEMENT_TOLERANCE_DEG)
        digitalWrite(LED_RIGHT, HIGH);
      else if (rollDiff < -MOVEMENT_TOLERANCE_DEG)
        digitalWrite(LED_LEFT, HIGH);
    }
    // Pitch-Kontrolle (vertikale Neigung) - niedrigste Priorität
    else if (abs(pitchDiff) > MOVEMENT_TOLERANCE_DEG) {
      if (pitchDiff > MOVEMENT_TOLERANCE_DEG)
        digitalWrite(LED_UP, HIGH);
      else if (pitchDiff < -MOVEMENT_TOLERANCE_DEG)
        digitalWrite(LED_DOWN, HIGH);
    }
  }
}
```

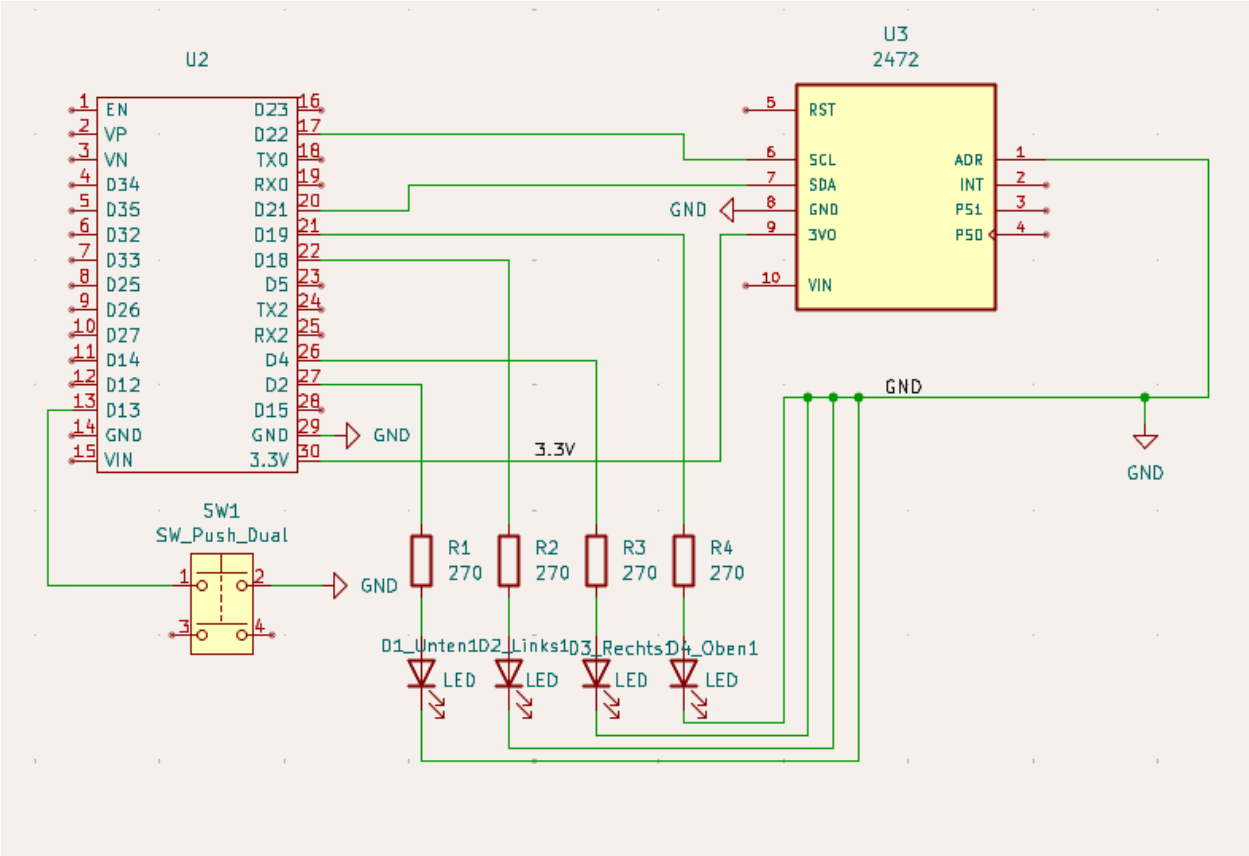
Erklärung der GUIDE_MODE Logik:

- Wenn die vertikale Bewegung erkannt wird, werden nur Bewegungskorrektur-LEDs angezeigt
- Nur bei ruhiger Bewegung wird die Orientierung geprüft
- Yaw-Priorität: Richtung halten ist wichtiger als kleine Neigungen
- Präzisere Toleranz: MOVEMENT_TOLERANCE_DEG (2°) statt ANGLE_TOLERANCE_DEG (5°) für genauere Führung

Hardware Implementierung

PCB





Komponenten:

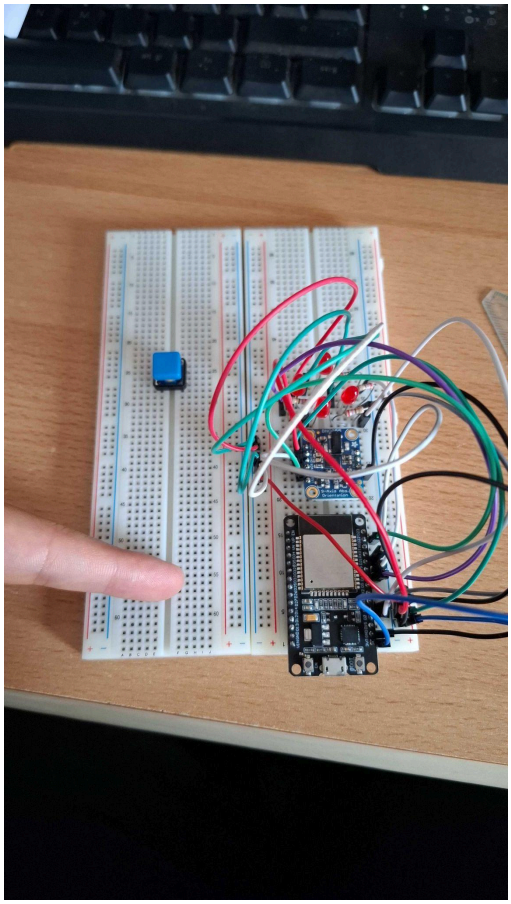
- ESP32-Wroom-32
- GPIO-Pins für LED-Ansteuerung
- I²C-Schnittstelle für BNO055-Kommunikation
- Internal Pull-up für Button-Eingang

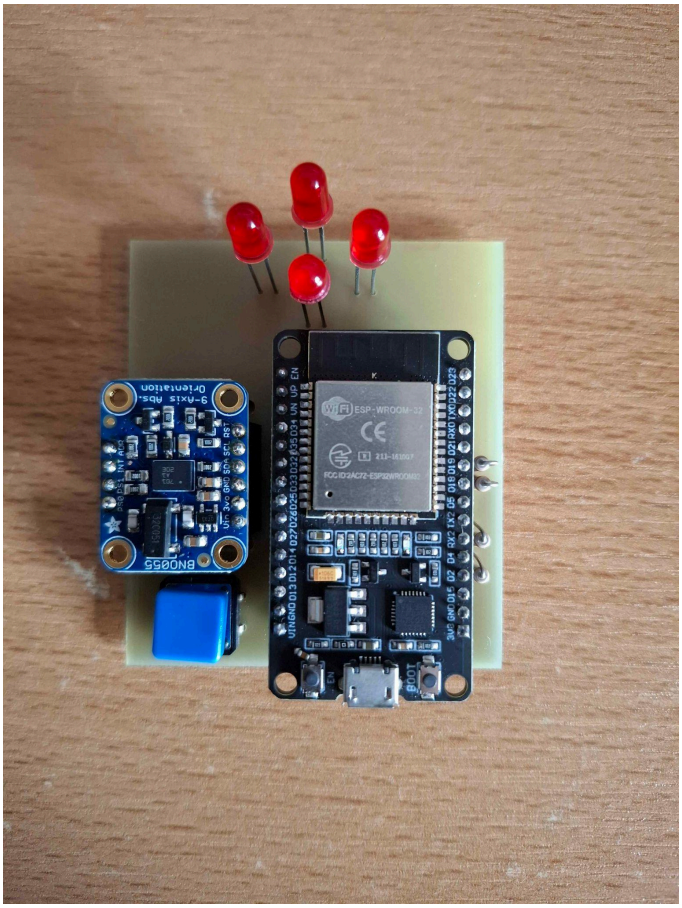
Pinbelegung

Komponente	Pin	Funktion
LED_UP	19	Anzeige "nach oben korrigieren"
LED_DOWN	2	Anzeige "nach unten korrigieren"
LED_LEFT	18	Anzeige "nach links korrigieren”

LED_RIGHT	4	Anzeige "nach rechts korrigieren"
BUTTON	13	Modus-Wechsel/Referenz-Reset
BNO055 SDA	SDA	I^2C Datenleitung
BNO055 SCL	SCL	I^2C Taktleitung

Entwicklung der PCB





Gehäuse

Das Gehäuse wurde von der Industrie- und Produktdesignerin Juline Müller von der AMD Akademische Mode & Design Hamburg entwickelt.

Fazit und Ausblick

Das entwickelte Mounting Tool erfüllt erfolgreich die Ziele einer präzisen Orientierungshilfe für die Montage von Tesa Powerstrips. Das Projekt vermittelte wertvolle Erfahrungen in der Sensordatenverarbeitung und der Implementierung von Bewegungsalgorithmen. Rückblickend bestand eine zentrale Herausforderung darin, dass der BNO055 trotz integrierter Sensorfusion nicht vollständig frei von Gimbal Lock ist, weswegen die Recherche nach der Lösung mit Quaternions geprägt von Stress und Glück ist. Zusätzlich hat der BNO055 angeblich ein Problem mit UART bei diversen ESP32 Modellen, weswegen die Adafruit Libraries nicht funktioniert hätten. Der Anschluss über I^2C hat dies nochmal gerettet. Eine gründlichere Recherche im Vorfeld hätte diesen Umstand frühzeitig erkennen lassen und möglicherweise zu einer alternativen Strategie oder Sensorwahl geführt. Ein Beispiel wäre die Lösung mittels LED wie bei der Unterseite einer Maus und einem Sensor, der das reflektierte Licht aufnimmt und so die Bewegungsinformation umsetzt. Bei der Überlegung zu Erweiterungen spielen akustische Feedbacks für die Ausrichtung eine große Rolle. Ein Display war bei der Entwicklung des Mounting Tools ebenfalls in Überlegung, jedoch wurde die Idee verworfen, da der Einsatz für diese Aufgabe und im Rahmen der Geschäftsidentität überdimensioniert wäre.

Ein großer Teil der Cross Innovation Class war die einzigartige, interdisziplinäre Arbeit mit Studenten verschiedener Studiengangsrichtungen. Schwer gefallen ist das verständliche Erklären der technischen Konzepte für Nicht-Techniker und die verschiedenen Ansätze und Erfahrungen mit Entwicklungszyklen von der Idee bis zum Prototyp. Die Arbeit mit Tesa als Industriepartner brachte Learnings über den Umgang mit Stakeholdern, da eine Balance zwischen technischer Machbarkeit, praktischen Bedürfnissen und Erwartungen erreicht werden musste. Zusammenfassend lässt sich sagen, dass die Arbeit trotz einiger Herausforderungen und Erkenntnisse im Nachhinein einen hohen Lern- und Erfahrungswert hatte.