seKwal

Absolut! Hier ist eine umfassende Projektdokumentation, die alle bisher gesammelten Informationen zusammenfasst und versucht, eine detaillierte Anleitung für Ihr DIY-Segway-Projekt zu geben.

—

Probleme

- BNO085 nicht auf 100Hz gestellt → Test mit SerialPlot → Pitch immer 4-5 Werte gleich in der Kurve
- Garbage Collector Probleme bei zu komplexen print Anweisungen → Loop hat immer Spikes → SerialPlot
- Motoren verdreht
- 1 Motor invertiert
- Looptime falsch berechnet (es war nicht der ganze Code der Loop in der Zeitberechnung)
- sleep_ms anstelle von sleep_us → ungenaue Verzögerung der Looptime !

Tuning The final Step is to Tune the PID loop Kp, Ki & Kd parameters.

A good starting point is to slowly increase Kp until the robot oscillates around the balance point and can catch a fall. Next, start Kd at around 1% the value of Kp and increase slowly until the oscillations disappear and the robot glides smoothly when pushed. Finally, start with Ki around 20% of Kp and vary until the robot “overshoots” the setpoint to actively catch a fall and return to vertical.

Links

- https://learn.pimoroni.com/article/overclocking-the-pico-2
- https://github.com/dobodu/BOSCH-BNO085-I2C-micropython-library

Projektdokumentation: DIY Segway mit RP2350 und MessKi-Integration

Version: 0.1 (Entwurf) **Datum:** 26. Mai 2025

1. Projektübersicht und Ziele

1.1. Projektidee Entwicklung eines selbstbalancierenden, zweirädrigen Fahrzeugs (Segway-ähnlich) unter Verwendung eines Raspberry Pi RP2350 Mikrocontrollers für die Echtzeit-Regelung und der bestehenden “MessKi”-Software für übergeordnete Steuerung, Parametrierung, Datenerfassung und Visualisierung.

1.2. Kernziele * Realisierung eines funktionierenden, selbstbalancierenden Fahrzeugs. * Nutzung von MicroPython auf dem RP2350 für die Regelung. * Nahtlose Integration mit der MessKi-Software über eine serielle/USB-Schnittstelle. * Implementierung einer sicheren und intuitiven Steuerungsmethode. * Modularer Aufbau für einfache Wartung und Erweiterung.

1.3. Angestrebte Funktionen * Automatische Balance im Stand und während der Fahrt. *

Steuerung von Geschwindigkeit und Richtung durch den Fahrer. * Fahrererkennung als Sicherheitsmerkmal (Dead Man's Switch). * Übertragung von Telemetriedaten (Neigungswinkel, Geschwindigkeit, Batteriestatus etc.) an MessKi. * Empfang von Steuerparametern (z.B. PID-Werte, Geschwindigkeitslimits) von MessKi. * Live-Visualisierung der Segway-Daten über das MessKi-Webinterface. * Logging von Fahr- und Sensordaten für Analyse und Optimierung. * Not-Aus-Funktion.

2. Systemarchitektur

2.1. Komponentenübersicht * **Fahrzeugbasis:** Rahmen, Räder (Hoverboard), Standplattform. * **Antriebseinheit:** 2x Hoverboard-Motoren mit integrierten Encodern, 2x Hoverboard-Motorcontroller. * **Sensorik:** IMU (BNO085), Fahrererkennungssensoren. * **Steuereinheit (Low-Level):** Raspberry Pi RP2350. * **Steuereinheit (High-Level & UI):** PC/Server mit laufender MessKi-Software. * **Kommunikationslink:** USB-Kabel (RP2350 als CDC-Device ↔ MessKi-Host). * **Stromversorgung:** Hoverboard-Batterie, Step-Down-Konverter für RP2350 & Sensorik.

2.2. Aufgabenverteilung

* **RP2350 (Echtzeit-Controller, MicroPython):**

- Auslesen des BNO085 IMU-Sensors (Neigung, Winkelgeschwindigkeit).
- Implementierung des Balance-Algorithmus (PID-Regler) mit hoher Frequenz (Ziel: >100 Hz).
- Auslesen der Fahrererkennungssensoren.
- Auslesen der Lenk-Input-Sensoren (Drucksensoren oder Potentiometer).
- Berechnung der Motorsteuerbefehle basierend auf Balance, Fahrer-Input und MessKi-Befehlen.
- Ansteuerung der Hoverboard-Motorcontroller über UART.
- Implementierung von Sicherheitslimits (max. Neigung, max. Geschwindigkeit).
- Serielle Kommunikation mit MessKi:
  - Senden von Telemetriedaten (Winkel, Geschwindigkeit, Batteriestatus, Motorströme, etc.).
  - Empfangen und Verarbeiten von Steuerbefehlen (Soll-Geschwindigkeit) und Parametern (PID-Werte, Limits) von MessKi.
  - Durchführen von Kalibrierungsroutinen.

* **MessKi (High-Level Steuerung, Monitoring, UI, Konfiguration):**

- Benutzerschnittstelle (Web-Frontend) für:
  - Anzeige von Live-Telemetriedaten (Winkel, Geschwindigkeit, etc.).
  - Einstellen von Parametern (PID-Gains, Max-Speed, Max-Tilt).
  - Auslösen von Kalibrierungsroutinen auf dem RP2350.
  - Gamepad-Interface zur optionalen, indirekten Steuerung (z.B. Setzen einer Soll-Geschwindigkeit, *nicht* direkte Balance-Lenkung).
- Neues MessKi-Gerät “SegwayInterface”:
  - Nutzt `SerialInput` oder `SCPIUsbInput` für die Kommunikation mit dem RP2350.
  - Definiert Channels und Measures für alle relevanten Segway-Daten.
  - Implementiert `@device_action`-Methoden zum Senden von Befehlen/Parametern an den RP2350.
- Datenausgabe über `WebSocketOutputHandler` an das Frontend.
- Datenspeicherung über `CsvOutputHandler` für Analysen.
- Logging von Systemereignissen und Fehlern.

3. Hardwarekomponenten

3.1. Mikrocontroller: Raspberry Pi RP2350 * **Board:** Raspberry Pi Pico W (oder ein anderes

RP2350-Board). * **Vorteile:** Ausreichend Rechenleistung (Dual Cortex-M0+ mit FPU), genug GPIOs, ADC-Eingänge, I2C- und UART-Schnittstellen, gute MicroPython-Unterstützung. * **Anschluss:** Über USB an den MessKi-Host-PC.

3.2. Inertial Measurement Unit (IMU): Bosch BNO085 * **Modul:** Breakout-Board mit BNO085. * **Vorteile:** Hochintegrierte Sensor-Fusion-Algorithmen (Hillcrest/CEVA SH-2), liefert stabile Orientierungsdaten (Quaternionen, Eulerwinkel), entlastet den RP2350. Höhere Update-Raten und oft bessere Kalibrierung als BNO055. * **Anschluss:** Über I2C an den RP2350. * **Bibliothek:** [dobodu/BOSCH-BNO085-I2C-micropython-library](https://github.com/dobodu/BOSCH-BNO085-I2C-micropython-library)

3.3. Motorisierung: Hoverboard-Motoren und -Controller * **Komponenten:** 2x Standard Hoverboard-Radnabenmotoren (meist BLDC mit Hallsensoren) und die zugehörigen Hoverboard-Motorcontroller-Platinen (oft zwei separate oder eine kombinierte). * **Ansteuerung:** Die Motorcontroller werden typischerweise über UART mit einem spezifischen seriellen Protokoll angesteuert. Das Protokoll beinhaltet oft Befehle für Geschwindigkeit/Drehmoment für jedes Rad.

- Ihr `HopfNRoll`-Projekt ist hier eine wertvolle Referenz für das Protokoll.

* **Verbindung zum RP2350:** Über eine UART-Schnittstelle des RP2350.

- **Level Shifter:** Unbedingt erforderlich, wenn die Hoverboard-Controller mit 5V Logikpegel arbeiten und der RP2350 mit 3.3V. Ein bidirektionaler Level-Shifter für TX/RX ist notwendig.

3.4. Fahrererkennung: Fußschalter / Drucksensoren * **Variante A (einfach): Fußschalter**

- 2x robuste Mikroschalter oder Endschalter.
- Montage unter den Fußpads, sodass sie bei Belastung schließen.
- Anschluss an digitale GPIO-Pins des RP2350 (mit internem oder externem Pull-up/-down Widerstand).

* **Variante B (fortgeschritten): Drucksensoren (FSRs)**

- 2x oder 4x Force Sensitive Resistors (FSRs).
- Montage unter den Fußpads.
- Anschluss an ADC-Eingänge des RP2350 (oft über einen Spannungsteiler).
- Ermöglicht eine feinere Erkennung und potenziell eine gewichtsbasierte Steuerung.

* **Ziel:** Motoren nur aktivieren, wenn beide Fußpads belastet sind.

3.5. Lenkmechanismus (Optionen) * **Option A: Drucksensoren (wie 3.4B, erweitert für Lenkung)**

- Mindestens ein FSR pro Fußpad, besser zwei (vorne/hinten) pro Pad.
- Software interpretiert differentielle Druckverteilung zwischen linkem/rechtem Fuß und vorderem/hinterem Bereich der Pads als Lenkimpuls.

* **Option B: Lenkstange mit Potentiometer**

- Mechanische Konstruktion einer neigbaren Lenkstange.
- Ein Dreh-Potentiometer (z.B. 10kΩ linear) erfasst den Neigungswinkel der Stange.
- Potentiometer an einen ADC-Eingang des RP2350.

* **Option C: Joystick direkt am RP2350 (weniger empfohlen für die Hauptlenkung)**

- Ein kleiner analoger Thumbstick, montiert an einer Festhalte-Struktur.
- X-Achse des Joysticks an einen ADC-Eingang.

3.6. Stromversorgung * **Hauptbatterie:** Standard Hoverboard Li-Ion Akku (typ. 36V, 10S). * **RP2350-Versorgung:** Step-Down (Buck) Konverter-Modul, das die 36V der Hauptbatterie auf stabile 5V (für USB-Power-Input des Pico) oder direkt 3.3V (für VSYS-Pin des Pico) reduziert.

- Der Konverter muss den Strombedarf des RP2350, der IMU und ggf. anderer 3.3V-Komponenten decken können.

* **Verkabelung:** Geeignete Kabelquerschnitte, Sicherungen für den Hauptstromkreis und den Step-Down-Konverter.

3.7. Optionale Komponenten * Status-LEDs (für Betriebszustand, Fehler, Kalibrierung) direkt am RP2350. * Ein kleiner Summer für akustische Signale. * Not-Aus-Schalter (Hardware), der die Stromzufuhr zu den Motoren unterbricht.

3.8. Materialliste (geschätzt, ohne Hoverboard-Basis) * Raspberry Pi RP2350 (Pico W oder ähnliches) * BNO085 Breakout-Board * 2-4x Fußschalter oder FSRs * Step-Down Konverter (36V → 5V/3.3V) * Bidirektionaler Level Shifter (3.3V ↔ 5V, falls nötig für Hoverboard UART) * Potentiometer (für Option 3.5B) oder Joystick-Modul (Option 3.5C) * Kabel, Steckverbinder (JST, Dupont etc.), Schrumpfschlauch * Lochrasterplatine oder kleine Prototyping-Platine * Gehäuse für Elektronik * Material für Rahmen/Plattform (Holz, Aluminiumprofile)

4. Software-Implementierung: RP2350 (MicroPython)

4.1. Entwicklungsumgebung * Thonny IDE oder VS Code mit MicroPython-Erweiterung (z.B. Pico-W-Go). * REPL-Zugriff über USB-Serial für direktes Testen und Debugging.

4.2. Softwarestruktur (RP2350) * `main.py`: Hauptprogramm, Initialisierung, Start der asynchronen Tasks. * `bno085_handler.py` (oder ähnlich): Kapselt die Logik für die BNO085-Bibliothek, liefert aufbereitete Winkeldaten. * `hoverboard_protocol.py`: Implementiert das Senden von Befehlen an die Hoverboard-Controller (inkl. Checksumme). * `pid_controller.py`: Klasse oder Funktionen für den PID-Regler. * `config.py`: Speichert Kalibrierwerte, PID-Konstanten (ggf. aus Flash geladen). * `utils.py`: Hilfsfunktionen (z.B. Quaternion-zu-Euler-Umrechnung, Mapping-Funktionen). * Asynchrone Tasks (`uasyncio`):

- `balance_loop()`: Hauptregelkreis (IMU lesen, PID, Motoren ansteuern).
- `messki_communication_loop()`: Kommunikation mit MessKi.
- Optional: `sensor_read_loop()`: Wenn IMU-Auslesung oder andere Sensorik komplexer ist und separat laufen soll.

4.3. IMU-Integration (BNO085) * Einbindung der `BOSCH-BNO085-I2C-micropython-library`. * Konfiguration der I2C-Schnittstelle auf dem RP2350. * Initialisierung des BNO085. * Aktivierung des benötigten Sensor-Reports (z.B. `BNO_REPORT_GAME_ROTATION_VECTOR` für Neigung ohne Magnetfeldeinfluss oder `BNO_REPORT_ROTATION_VECTOR` für 9-Achsen-Fusion). * Implementierung einer Funktion `get_raw_bno_pitch_from_sensor()`, die den rohen Pitch-Winkel (oder den relevanten Neigungswinkel) vom Sensor liefert. Dies erfordert das Parsen der Daten aus dem Report der Bibliothek. * Implementierung der Kalibrierungsroutine (siehe 4.10).

4.4. Balance-Algorithmus (PID-Regler) * Implementierung einer PID-Reglerfunktion oder -klasse.

- Eingang: `current_calibrated_tilt_angle`, `target_angle` (sollte 0.0 nach Kalibrierung sein).
- Ausgang: Korrekturwert für die Motorgeschwindigkeit.

* Parameter `KP`, `KI`, `KD` (Proportional, Integral, Derivative Anteile). * Berechnung des Fehlers: `error = target_angle - current_calibrated_tilt_angle`. * Berechnung des Integral-Terms (Summe der Fehler über Zeit), inkl. Anti-Windup. * Berechnung des Differential-Terms (Änderungsrate des Fehlers). * Loop-Zeit (`dt`) muss berücksichtigt werden. * `pid_output = (KP * error) + (KI * integral_error) + (KD * derivative_error)`. * **Tuning:** Die PID-Werte müssen experimentell ermittelt werden. Starten Sie mit kleinem P, dann D, dann I.

4.5. Motoransteuerung (Hoverboard-Controller) * Konfiguration der UART-Schnittstelle des RP2350. * Implementierung der Funktion `send_hoverboard_command(left_speed, right_speed)`:

- Nimmt Soll-Geschwindigkeitswerte für linken und rechten Motor (z.B. -1000 bis +1000).
- Erzeugt das korrekte serielle Datenpaket gemäß Hoverboard-Protokoll.
- Berechnet und fügt die erforderliche Checksumme hinzu.
- Sendet das Paket über UART an die Motorcontroller.

4.6. Fahrererkennung * Funktion `is_rider_present()`:

- Liest die Zustände der Fußschalter-GPIOs oder die Werte der FSR-ADCs.
- Gibt `True` zurück, wenn ein Fahrer erkannt wird (z.B. beide Schalter gedrückt / ausreichender Druck auf FSRs).
- Wichtig: Der `balance_loop` darf die Motoren nur aktivieren, wenn ein Fahrer präsent UND das System kalibriert ist.

4.7. Lenkungslogik * Funktion `read_steering_input_locally()`:

- Liest die Werte der Lenksensoren (Drucksensoren oder Potentiometer).
- Wandelt die Rohwerte in einen normalisierten Lenkbefehl um (z.B. -1.0 für voll links, 0.0 für geradeaus, +1.0 für voll rechts).

* Integration in `balance_loop()`:

- `steer_effect = normalized_steer_input * STEER_SENSITIVITY`.
- `left_motor_cmd = base_speed_command - steer_effect`.
- `right_motor_cmd = base_speed_command + steer_effect`.
- `base_speed_command` kommt vom PID-Regler und dem Geschwindigkeits-Sollwert von MessKi.
- `STEER_SENSITIVITY` muss experimentell abgestimmt werden.

4.8. Kommunikation mit MessKi (Protokoll) * RP2350 agiert als serielles Gerät (USB CDC). * **RP2350 → MessKi (Telemetrie):**

- Regelmäßiges Senden von Datenpaketen, z.B. als ASCII-Zeilen oder JSON-Strings.
- Format: `KEY1=VALUE1,KEY2=VALUE2\n` oder `{“tilt”: 1.23, “vbat”: 35.8}\n`
- Beispiele für Daten: `calibrated_tilt_angle`, `raw_pitch`, `motor_left_cmd`, `motor_right_cmd`, `battery_voltage` (falls messbar), `rider_present_status`, `current_steer_input`, `pid_error`, `pid_integral`, `pid_derivative`, `pid_output`.

* **MessKi → RP2350 (Befehle/Parameter):**

- Format: `CMD:ACTION=VALUE\n` oder `SET:PARAM=VALUE\n`

- Beispiele:
  - `CMD:SPEED=200` (Soll-Geschwindigkeit für Vorwärts/Rückwärts)
  - `SET:KP=12.5`
  - `SET:MAX_TILT=20.0`
  - `CMD:CALIBRATE_NOW`
  - `CMD:EMERGENCY_STOP`
- Funktion `process_messki_command(command_str)` auf dem RP2350 zum Parsen und Anwenden.

4.9. Sicherheitsfunktionen und Fehlerbehandlung * **Maximale Neigung:** Wenn `calibrated_tilt_angle` einen kritischen Wert überschreitet (z.B. > 25-30 Grad), Motoren sofort abschalten oder sanft herunterregeln. * **IMU-Fehlererkennung:** Wenn `get_tilt_angle()` `None` zurückgibt oder Fehler signalisiert, Motoren stoppen. * **Fahrer nicht präsent:** Motoren sofort stoppen. * **Kommunikationsverlust zu MessKi:** Nach einer bestimmten Zeit ohne “Heartbeat” oder Befehl von MessKi in einen sicheren Zustand gehen (z.B. Stopp). * **Batterie-Unterspannung:** Motoren abschalten, um Tiefentladung zu verhindern (falls Batteriespannung gemessen wird).

4.10. Kalibrierungsroutinen * **Winkel-Offset-Kalibrierung:**

- Funktion `calibrate_level_procedure()` wie zuvor besprochen.
- Auslösbar über Taster am Segway oder Befehl von MessKi.
- Speichern des `ANGLE_OFFSET` im Flash (`config.py` oder separate Datei).
- Laden des Offsets beim RP2350-Start.

* **Lenkungs-Kalibrierung (falls nötig):**

- Für Potentiometer: Mittelstellung und Maximalausschläge erfassen.
- Für Drucksensoren: Ruhewerte und Werte bei maximaler Belastung/Neigung erfassen.

5. Software-Implementierung: MessKi-Integration

5.1. Neues MessKi-Gerät: “SegwayInterface” * **Config (`SegwayConfig(DeviceConfigBase)`):**

- `device_class: Literal[“SegwayInterface”] = “SegwayInterface”`
- `name: str = “RP2350 Segway Controller”`
- `input_configs`: Liste, die eine `SerialInputConfig` für den USB-Port des RP2350 enthält (Port muss vom Benutzer eingestellt werden).
- `active_input_uuid`: UUID der `SerialInputConfig`.
- Felder für Standard-PID-Werte, Max-Speed, Max-Tilt, die beim Start an den RP2350 gesendet werden können.
- `fetch_interval_sec`: Wie oft MessKi aktiv Daten vom RP2350 anfordert (falls Pull-Mechanismus) oder Verarbeitungsintervall für empfangene Daten. Eher gering halten, da RP2350 von sich aus senden sollte.

* **Handler (`SegwayHandler(DeviceBase)`):**

- `_activate()`: Stellt sicher, dass der serielle Port geöffnet ist. Sendet ggf. initiale Konfigurationsparameter an den RP2350.
- `_run_device_task()`:
  - Liest kontinuierlich Daten vom seriellen Input (`await self._read_from_active_input()`).
  - Parst die empfangenen Telemetrie-Strings/JSONs vom RP2350.
  - Aktualisiert die entsprechenden Measures in den MessKi-Channels.

    - Holt Gamepad-Daten vom `GamepadHandler`.
    - Berechnet daraus eine Soll-Geschwindigkeit (z.B. Vorwärts/Rückwärts basierend auf einem Stick). **Wichtig: Die feine Balance-Lenkung geschieht auf dem RP2350! MessKi liefert nur den "Wunsch" des Fahrers.**
    - Sendet die Soll-Geschwindigkeit (und ggf. grobe Lenkrichtung, falls gewünscht, aber eher nicht für Balance) an den RP2350.
- `@device_action` Methoden:
    - `set_pid_gains(kp: float, ki: float, kd: float)`: Sendet `SET:KP=...` etc. an RP2350.
    - `set_max_speed(speed: int)`
    - `set_max_tilt_angle(angle: float)`
    - `trigger_calibration()`: Sendet `CMD:CALIBRATE_NOW` an RP2350.
    - `send_emergency_stop()`: Sendet `CMD:EMERGENCY_STOP`.
    - `send_target_speed(speed: int)` (intern von `_run_device_task` genutzt oder als Action).

5.2. Input-Handler Konfiguration (Serial/USB) * In der `SegwayConfig` wird eine `SerialInputConfig` definiert. * Der `port` muss vom Benutzer in MessKi auf den korrekten virtuellen COM-Port / `/dev/ttyACMx` des RP2350 eingestellt werden. * Baudrate, etc. müssen mit den Einstellungen auf dem RP2350 übereinstimmen. * `read_mode` in MessKi sollte "readline" sein, wenn der RP2350 mit Newline terminiert. `terminator` entsprechend setzen (z.B. `\n`).

5.3. Channel- und Measure-Definitionen in MessKi (für `SegwayConfig`) * **Channel "Segway Telemetry":**

- `TiltAngle` (NumericMeasure, Einheit: Grad)
- `AngularVelocity` (NumericMeasure, Einheit: Grad/s)
- `BatteryVoltage` (NumericMeasure, Einheit: V)
- `MotorLeftSpeed` (NumericMeasure, Einheit: RPM oder Einheit vom RP2350)
- `MotorRightSpeed` (NumericMeasure, Einheit: RPM oder Einheit vom RP2350)
- `RiderPresent` (StringMeasure oder NumericMeasure: "JA"/"NEIN" oder 1/0)
- `SteeringInputRaw` (NumericMeasure, Rohwert vom Lenksensor)
- `PIDError` (NumericMeasure)
- `PIDOutput` (NumericMeasure)
- `RP2350Status` (StringMeasure, z.B. "BALANCING", "CALIBRATING", "ERROR")

* **Channel "Control Outputs":**

- `TargetSpeedToRP` (NumericMeasure, von MessKi an RP2350 gesendet)
- `TargetSteerToRP` (NumericMeasure, falls MessKi auch Lenkimpulse sendet)

* **Channel "PID Parameters":**

- `Param_KP` (NumericMeasure, spiegelt den Wert auf dem RP2350 wider, lesend)
- `Param_KI` (NumericMeasure)
- `Param_KD` (NumericMeasure)

* Outputs: `WebSocketOutputHandler` für alle relevanten Channels, `CsvOutputHandler` für Telemetrie.

5.4. Steuerung über MessKi (Gamepad, API) * Das Gamepad in MessKi wird *nicht* für die direkte Links/Rechts-Balance-Lenkung verwendet. * Es kann verwendet werden, um eine **Soll-Geschwindigkeit** (vorwärts/rückwärts) an den RP2350 zu senden. Der `_run_device_task` im `SegwayHandler` würde z.B. den Y-Achsenwert eines Sticks lesen und als `CMD:SPEED=...` an den

RP2350 senden. * Die API-Endpunkte (`@device_action`) ermöglichen das Setzen von Parametern (PID, Limits) und das Auslösen von Aktionen (Kalibrierung, Not-Aus).

5.5. Datenvisualisierung und -logging * MessKi-Webfrontend zeigt Live-Werte aus den “Segway Telemetry”-Measures an. * Grafische Darstellung von Neigungswinkel, Geschwindigkeiten etc. * CSV-Logging für spätere Analyse der Fahrdynamik und PID-Abstimmung.

6. Mechanische Konstruktion

6.1. Rahmen und Plattform * Stabiler Rahmen, der die Hoverboard-Achsen/Motoren aufnimmt. * Eine Plattform für den Fahrer, die ausreichend Platz bietet und die Montage der Fußschalter/Drucksensoren ermöglicht. * Materialien: Aluminiumprofile, Multiplex-Holz, Stahl (je nach verfügbaren Mitteln und Fähigkeiten). * Schwerpunkt beachten: Der Gesamtschwerpunkt (Fahrer + Segway) sollte sich möglichst über der Radachse befinden.

6.2. Montage der Komponenten * RP2350, BNO085, Step-Down-Konverter, Level-Shifter in einem geschützten Gehäuse unterbringen. * BNO085 möglichst nahe am Drehzentrum (Radachse) und vibrationsarm montieren. Die genaue Ausrichtung (X, Y, Z Achsen) muss in der Software berücksichtigt werden. * Sichere Montage der Batterie.

6.3. Lenkmechanismus (falls zutreffend) * **Drucksensoren:** Sauber unter den Fußpads integrieren, sodass sie zuverlässig auf Gewichtsverlagerung reagieren. * **Lenkstange:** Stabile Lagerung der Drehachse, spielfreie Verbindung zum Potentiometer/Encoder. Endanschläge für die Lenkstange vorsehen.

6.4. Verkabelung und Stromverteilung * Sorgfältige Verkabelung mit ausreichenden Querschnitten, besonders für die Motorcontroller und die Hauptbatterie. * Zugentlastung für alle Kabel. * Übersichtliche Stromverteilung mit Sicherungen. * Gute Abschirmung für Signalleitungen (IMU, UART), um Störungen zu vermeiden.

7. Testplan und Inbetriebnahme

SEHR WICHTIG: Bei allen Tests mit Motorkraft äußerste Vorsicht walten lassen! Das Segway muss immer gesichert sein, um unkontrolliertes Wegfahren oder Umkippen zu verhindern! Beginnen Sie mit aufgebockten Rädern.

7.1. Modultests (RP2350) * **BNO085:** Daten auslesen, Winkelberechnung prüfen, Kalibrierung testen. Ausgabe über Serial an PC. * **Hoverboard-Controller:** Serielle Befehle senden, Motoren manuell drehen lassen (langsam!), Drehrichtung prüfen. * **Fahrererkennung:** Funktion der Schalter/FSRs prüfen. * **Lenksensorik:** Rohwerte der Drucksensoren/Potis auslesen und prüfen.

7.2. Statische Balance-Tests (gesichert) * Segway aufbocken, sodass die Räder frei drehen können, aber das Gestell nicht umkippen kann. * Fahrererkennung aktivieren (Gewichte auf Pads). * Balance-Regler (PID) aktivieren. * **Ziel:** Die Räder sollten versuchen, die Plattform horizontal zu halten, wenn sie manuell leicht geneigt wird. * PID-Tuning beginnen:

```
  1.  Nur P-Anteil (I und D auf 0): P erhöhen, bis leichte Oszillationen
auftreten. Dann P etwas reduzieren.
  2.  D-Anteil hinzufügen: D erhöhen, um Oszillationen zu dämpfen.
  3.  I-Anteil hinzufügen (vorsichtig): I erhöhen, um statische Fehler
auszugleichen.
  *   Iterativ vorgehen!
```

7.3. Dynamische Balance-Tests (mit Fahrer, SEHR GUT GESICHERT!) * Personen zum Sichern bereitstellen! Langsam beginnen. * In einem Bereich mit viel Platz und ohne Hindernisse. * Schutzausrüstung tragen (Helm!). * Feintuning der PID-Werte. * Testen der Reaktion auf leichte Störungen.

7.4. Lenkungstests * Wenn Balance grundlegend funktioniert, Lenklogik aktivieren. * Zuerst bei sehr langsamer oder keiner Vorwärtsbewegung testen. * `STEER_SENSITIVITY` anpassen für ein angenehmes Lenkverhalten.

7.5. MessKi-Integrationstests * RP2350 mit MessKi verbinden. * Prüfen, ob Telemetriedaten korrekt in MessKi angezeigt werden. * Testen der Parameteränderung über MessKi (PID-Werte, Limits). * Testen der Soll-Geschwindigkeitsvorgabe von MessKi an RP2350.

7.6. Sicherheitsüberprüfungen * Funktion der Fahrererkennung unter allen Bedingungen. * Reaktion auf maximale Neigung. * Not-Aus-Funktion. * Verhalten bei niedrigem Batteriestand.

8. Mögliche Erweiterungen und Zukünftige Arbeiten * Verbesserte Sensor-Fusion-Algorithmen (falls nicht BNO085). * Adaptiver PID-Regler. * Energierückgewinnung beim Bremsen (Hoverboard-Controller unterstützen das oft). * Integration von GPS für Tracking. * Fortgeschrittenere Fahrmodi (z.B. Sport, Eco). * Hinderniserkennung.

9. Sicherheitshinweise (SEHR WICHTIG!)

* **Dies ist ein potenziell gefährliches Projekt!** Ein unkontrolliertes Segway kann schwere Verletzungen verursachen oder Sachschäden anrichten. * **Arbeiten Sie immer mit größter Vorsicht!** * **Tragen Sie immer geeignete Schutzausrüstung** (Helm, Knie-/Ellbogenschützer) bei Testfahrten. * **Sichern Sie das Segway bei Tests immer**, besonders in der Anfangsphase (z.B. aufbocken, Haltevorrichtungen, zweite Person). * **Implementieren Sie mehrere Sicherheitsebenen** (Fahrererkennung, Neigungslimits, Not-Aus). * **Beginnen Sie mit niedrigen Geschwindigkeiten und geringer Leistung.** * **Testen Sie in einem sicheren, freien Bereich ohne Hindernisse oder andere Personen.** * **Seien Sie sich der Grenzen Ihrer Fähigkeiten und der Hardware bewusst.** * **Lithium-Ionen-Akkus erfordern sorgfältigen Umgang!** Kurzschlüsse und Überladung vermeiden. Brandschutz beachten. * **Übernehmen Sie die volle Verantwortung für Ihr Projekt und dessen Betrieb.**

—

Diese Dokumentation ist ein umfangreicher Leitfaden. Beginnen Sie mit kleinen, überschaubaren Schritten und testen Sie jede Komponente gründlich, bevor Sie sie integrieren. Das Debugging und Tuning des Balance-Reglers wird die meiste Zeit in Anspruch nehmen.

Viel Erfolg bei diesem spannenden Vorhaben!