

D-O AIO32 v2

ESP32-S3-basierter AIO-Controller

Benutzerhandbuch

Vollstaendiges Benutzerhandbuch fuer das D-O AIO32 v2 Controller-Board. Abdeckung: Firmware-Upload, Hardware-Uebersicht, IMU-Setup, RC (iBus + SBUS), Button-Interface, Serial-CLI, Balance-PID-Tuning, Sound, Display, Akku-Monitoring, Fehlersuche und Sicherheit. Zielgruppe: Maker, die ihren eigenen D-O bauen.

Version 2.1.1 — 2026-04-23

Haftungsausschluss

D-O AIO32 v2 ist ein DIY-Projekt. Aufbau, Verkabelung, Konfiguration und Betrieb umfassen Arbeiten, die bei unsachgemässer Ausführung Sach- oder Personenschaden verursachen koennen.

Mit der Nutzung dieser Dokumentation, Hard- und Software erkennst du an:

- Du bist alleine verantwortlich fuer alle Arbeiten an deinem System.
- Du verfuegst ueber die noetigen Grundkenntnisse oder holst dir qualifizierte Hilfe.
- Du pruefst Spannung, Polung, Sicherungen und Steckverbindungen vor jedem Einschalten.
- Du verwendest nur passende Komponenten.
- Du beachtest die oertlichen Vorschriften.

Der Autor und alle Beteiligten uebernehmen keinerlei Haftung fuer Schaeden durch falsche Verkabelung, ungeeignete Komponenten, unsachgemaeße Bedienung oder unzureichende Sicherheitsmassnahmen.

Die Dokumentation und Software werden "**wie besehen**" ohne jede Garantie bereitgestellt. Im Zweifel qualifizierte Hilfe einholen bevor am System gearbeitet wird.

Inhaltsverzeichnis

Haftungsausschluss	2
Inhaltsverzeichnis	3
Einfuehrung — Was ist D-O AIO32 v2?	6
Hauptfunktionen	6
Verhaeltnis zur Mega-AIO-Linie	6
Repository und Support	7
Schnelleinstieg	8
Hardware-Uebersicht	10
Hauptkomponenten	10
Pin-Belegung	11
Servo-Beschriftung auf dem Board	11
Firmware-Upload	13
Board-Einstellungen	13
Benoetigte Libraries	13
Upload-Prozedur	14
IMU-Setup und Kalibrierung	15
IMU-Modi	15
Kalibrier-Prozedur	15
Madgwick-Beta	15
RC-Empfaenger-Setup (iBus + SBUS)	17
Kanalbelegung	17
iBus-Setup (Default)	17
SBUS-Setup (ohne Inverter)	17
Button-Interface	19
Button-Labels und Funktionen	19
Button-Events debuggen	19
Serial-CLI-Referenz	20
Top-Level-Kommandos	20

pid — PID-Tuning	21
imu — IMU, Kalibrierung, Madgwick	21
drive — Fahrdynamik	22
rc — Protokoll + Kanäle	22
sound — DFPlayer	23
display — TFT	23
balance — State-Steuerung	23
feature — Runtime-Flags	24
test — Manueller Hardware-Test	24
debug — Dumps + Live-Streams	25
Balance und PID-Tuning	26
Empfohlene Startwerte	26
Tuning-Prozedur	26
Sound (DFPlayer Mini + SD)	27
Track-Belegung (identisch zur Mega-Linie)	27
SD-Karten-Format + Dateinamen	27
Lautsprecher-Anschluss	28
Volume und Mute	28
Display und Status-LED	29
Display-Modi	29
Always-On vs Auto-Sleep	29
NeoPixel-State-Muster	29
Akku-Monitoring	30
Konfiguration	30
Kalibrierung	30
Fehlersuche	32
Sicherheit	34

Teil I

Erste Schritte

Einfuehrung — Was ist D-O AIO32 v2?

Das D-O AIO32 v2 ist ein ESP32-S3-basierter All-in-One-Controller fuer den selbstbalancierenden D-O-Droiden aus Star Wars: Der Aufstieg Skywalkers. Es ist das juengere Geschwister des Arduino-Mega-basierten "Control & Power Board" und bedient den gleichen Droiden, gleichen Sender und die gleichen Sound-Files — nutzt aber einen modernen Mikrocontroller, zwei IMUs mit Madgwick-Quaternion-Fusion, ein Farbdisplay, Buttons zur Konfiguration und eine vollstaendige interaktive CLI. Beide Produktlinien werden parallel gepflegt.

Hauptfunktionen

- **ESP32-S3 All-in-One:** TENSTAR TS-ESP32-S3-Modul mit nativem USB-C und 4 MB Flash.
- **Dual-IMU-Fusion:** QMI8658C (auf dem TENSTAR-Modul) plus LSM6DS3 (auf der AIO32-Platine) kombiniert durch Madgwick-6-DoF-AHRS mit runtime-anpassbarem Beta.
- **Zwei RC-Protokolle:** FlySky iBus (Default) und Futaba/FrSky SBUS, beide zur Laufzeit per CLI umschaltbar — SBUS ohne externen Inverter dank ESP32-UART-Hardware-Invert.
- **Vier Servos** ueber native LEDC-PWM (50 Hz, 14-bit) — glatte Idle-Animationen auf Mainbar + 3 Kopfachsen.
- **DFPlayer Mini** Sound (Makuna-Library, non-blocking) mit Mood-Triggern, Idle-Sounds, Tilt-Warnungen.
- **1,14" ST7789 Farb-TFT** mit adaptivem Portrait-/Landscape-Layout, Live-Anzeige von Pitch / Akku / PID / Mode.
- **NeoPixel-Status-LED** mit Change-Detection-Mustern fuer INIT / CALIBRATING / READY / RUNNING / ERROR / E-STOP.
- **6 beschriftete Tasten (B1-B6)** fuer Start/Stop, Calibrate, Display, Debug/Select, Mode, Back/Emergency — alle haeufigen Aktionen ohne CLI.
- **Vollstaendige Serial-CLI** (~80 Kommandos): PID-Tuning, Adaptive-Baender, IMU-Modi, RC-Protokoll, Fahrdynamik, Debug-Streams.
- **Adaptive 3-Band-PID** mit linearem Blending zwischen Slow/Medium/Fast-Gains.
- **Arcade-/Tank-Mixing-Modi** mit Deadband, Expo, Motor-Ramping und dynamischem Lean-Winkel.
- **Auto-Baseline-Kalibrierung** (B1-long): 2-Sekunden-Pitch-Mittelwert wird neuer Target-Angle, in NVS persistiert.
- **NVS-Persistenz** fuer jeden Parameter: RC-Protokoll, IMU-Mount, Madgwick-Beta, Target-Angle.

Verhaeltnis zur Mega-AIO-Linie

Die AIO32 ist **kein Ersatz** fuer das Mega-AIO-Board — beide werden als aktive Produktlinien gepflegt. Die Mega-Linie ist einfacher, bewaehrt, und hat ein eigenes umfangreiches Handbuch. Die AIO32 richtet sich an Nutzer, die modernere Hardware, ein Display, farbige LED-Rueckmeldung, Button-Konfiguration und die saubere Architektur dank 240 MHz und 2 MB PSRAM wollen. Beide nutzen die gleiche iBus/SBUS-Kanalbelegung und das gleiche Sound-File-Layout auf SD.

Repository und Support

Quellcode, KiCad-Hardware, Handbuch-PDFs und Issue-Tracker liegen im gemeinsamen PrintedDroid-Repository auf GitHub — gleiche Repo wie die Mega-AIO-Linie, andere Unterordner:

- <https://github.com/PrintedDroid/D-O-Printed-Droid>

Kommerzieller Support, Fertigboards und Droiden-Kits: <https://www.printed-droid.com/>

Schnelleinstieg

Von einer verkabelten AIO32 zum fahrenden Droiden in rund 30 Minuten, Hardware vorausgesetzt montiert. Wenn etwas klemmt: ins Troubleshooting-Kapitel springen.

Raeder bei erstem Einschalten in der Luft. Die Balance-PID kann den Droiden waehrend der Kalibrierung heftig bewegen.

1. Arduino IDE + ESP32-Core installieren

Arduino IDE 2.x. Datei → Einstellungen → zusätzliche Boardverwalter-URL: https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json.

Boardverwalter: **esp32 von Espressif** in Version 3.3.7 installieren.

2. Libraries installieren

Library-Manager: **Adafruit ST7789**, **Adafruit NeoPixel**, **DFMiniMp3 von Makuna**. (Madgwick ist eine eigene Implementierung im Sketch, keine Library noetig.)

3. Board waehlen

Werkzeuge → Board → **Adafruit Feather ESP32-S3 TFT** (die AIO32 nutzt das TENSTAR-Modul mit kompatibelem Pinout). Werkzeuge → USB CDC On Boot: **Enabled**, PSRAM: **Disabled**, Flash Size: **4MB**, Partitionierung: **Default 4MB with spiffs**.

4. Sketch oeffnen

D-O_AIO32_v2.1/D-O_AIO32_v2.1.ino oeffnen.

5. Upload

BOOT am AIO32 halten, RESET einmal druecken, BOOT nach einer Sekunde loslassen. Upload druecken. Nach dem Upload RESET fuer Start.

6. Serial Monitor 115200 Baud oeffnen

Es sollte `=== D-O AIO32 v2.1.x ===` erscheinen, dann `Init-Diagnose.help` eintippen fuer CLI-Uebersicht.

7. IMUs kalibrieren

D-O aufrecht und perfekt still auf einer ebenen Flaechе aufstellen. **B2 (Calibrate)** kurz druecken, oder `imu cal` eintippen. 5 Sekunden still halten. Offsets landen im NVS.

8. Baseline-Target-Angle setzen

Nach der IMU-Kalibrierung **B1 (Start/Stop)** lang druecken (oder `balance baseline` senden). 2-Sekunden-Pitch-Mittelwert wird neuer Target-Angle. Bleibt in NVS.

9. RC pruefen

`debug stream rc` eintippen. Sticks bewegen — Werte muessen glatt 1000..2000 wandern. Beliebige Taste stoppt den Stream.

10. Erster Balance-Versuch

Raeder auf Boden, Abstand halten. **B1 (Start/Stop)** kurz druecken fuer Balance (oder `balance start`). Wenn der Droid beim Start kippt: Target-Angle anpassen mit `pid target -0.5` und neu baseline-en.

Tipp: jeder tuneable Wert im Sketch ist aus der CLI erreichbar, im NVS gespeichert und ueberlebt Reboots. Fuer normales Tuning kein Reflash noetig.

Hardware-Uebersicht

Das AIO32-v2-Board ist eine einzelne Platine mit TENSTAR-ESP32-S3-Modul, einem **Cytron MDD10A** Dual-Motortreiber (10 A pro Kanal — gleicher wie auf der Mega-AIO-Linie), einer zweiten IMU, DFPlayer-Sockel, vier Servo-Headern, einem 6-Tasten-ADC-Ladder und einem Akku-Spannungsteiler. Gleiche Droiden-Mechanik, gleiche Sound-Library, gleiche Sender-Belegung wie die Mega-Linie — nur das Controller-Hirn ist anders.

Hauptkomponenten

Komponente	Rolle	Notizen
TENSTAR TS-ESP32-S3	Hauptcontroller	240 MHz, 4 MB Flash, natives USB-C, ST7789-TFT 1,14" 135x240 integriert, QMI8658C-IMU onboard
LSM6DS3TR-C	Zweite IMU	I2C 0x6A (SA0 am Modul per Solder-Jumper auf GND)
Cytron MDD10A	Motortreiber	Dual-Kanal, 10 A Dauerstrom pro Kanal. Gleicher Treiber wie in der Mega-Linie.
DFPlayer Mini	Sound	UART via Makuna-DFMiniMp3-Library, spielt /mp3/NNNN.mp3 von SD
NeoPixel (auf TENSTAR)	Status-LED	GPIO 33, WS2812, 1 LED
6 Tasten (B1-B6)	Konfig + Sicherheit	Auf der Platine mit B1..B6 beschriftet, jeweils Short- und Long-Press
Batterie-Teiler	Akku-Spannungsmessung	Auf GPIO 16 (ADC2). Faktor ~5,74 passt zum 47k/10k-Teiler

Pin-Belegung

GPIO	Funktion	Notizen
2	iBus / SBUS RC-Eingang	Serial1-RX. UART-Invert wird bei SBUS zur Laufzeit aktiviert
13	Motor 1 DIR (Cytron MDD10A Kanal 1 Richtung)	
11	Motor 1 PWM (Cytron MDD10A Kanal 1 Speed)	LEDC
10	Motor 2 DIR (Cytron MDD10A Kanal 2 Richtung)	
9	Motor 2 PWM (Cytron MDD10A Kanal 2 Speed)	LEDC
6	Servo Mainbar — Board-Label "Servo 3" (CH3)	LEDC 50 Hz 14-bit
15	Servo Kopf Pitch — Board-Label "Servo 4" (CH4)	LEDC 50 Hz 14-bit
14	Servo Kopf Yaw — Board-Label "Servo 5" (CH5)	LEDC 50 Hz 14-bit
8	Servo Kopf Roll — Board-Label "Servo 6" (CH6)	LEDC 50 Hz 14-bit
42	I2C SDA	Beide IMUs + optional BMP280
41	I2C SCL	
33	NeoPixel Status-LED	WS2812 onboard
5	Buttons B1-B3	
12	Buttons B4-B6	Gesperrt wenn WiFi aktiv (ADC2-Pin)
16	Akku-Spannungsmessung	ADC2-CH5 — auch bei WiFi gesperrt. Teiler 47k/10k
18	DFPlayer TX (an Modul-RX via 1-k Ω -Widerstand)	UART2-TX
17	DFPlayer RX (vom Modul-TX)	UART2-RX

ADC2-Einschraenkung: GPIO 12 (Button-Ladder 2) und GPIO 16 (Akku-Messung) sind ADC2-Pins. Wenn eine zukuenftige Firmware WiFi aktiviert, sind diese Pins fuer Analog-Reads blockiert. Die aktuelle Firmware nutzt kein WiFi — Konflikt ist theoretisch. Vor Aktivierung von OTA sollten die Pins auf ADC1 migriert werden.

Servo-Beschriftung auf dem Board

Die vier Servo-Header auf der AIO32 sind mit **"Servo 3"** bis **"Servo 6"** bedruckt. Die Beschriftung entspricht 1:1 den FlySky-iBus-/SBUS-Kanalnummern — der Mainbar-Servo kommt an den Header "Servo 3" und bekommt RC-Kanal 3, der Kopf-Pitch-Servo an "Servo 4" = CH4, und so weiter. Es gibt kein "Servo 1" oder "Servo 2" auf dem Board, weil die Kanale 1 und 2 fuer die beiden Antriebsmotoren reserviert sind.

Board label / Board-Label	Function / Funktion	FlySky / iBus CH
Servo 3	Mainbar	CH3
Servo 4	Head Pitch / Kopf Pitch	CH4

Board label / Board-Label	Function / Funktion	FlySky / iBus CH
Servo 5	Head Yaw / Kopf Yaw	CH5
Servo 6	Head Roll / Kopf Roll	CH6

Firmware-Upload

Die AIO32-Firmware liegt in `D-O_AIO32_v2.1/D-O_AIO32_v2.1.ino`. Es ist ein Multi-File-Arduino-Sketch; alle `.h`- und `.cpp`-Dateien in diesem Ordner gehören zur gleichen Compile-Unit. Compile-Ziel: Arduino IDE 2.x mit dem Espressif-ESP32-Core.

Board-Einstellungen

Setting / Einstellung	Value / Wert
Board	Adafruit Feather ESP32-S3 TFT
USB CDC On Boot	Enabled
CPU Frequency	240 MHz
Flash Mode	QIO 80 MHz
Flash Size	4 MB
Partition Scheme	Default 4MB with spiffs (1.2MB APP / 1.5MB SPIFFS)
PSRAM	Disabled
Upload Mode	UART0 / Hardware CDC
Upload Speed	921600 (lower to 460800 if uploads fail)
Core Version	3.3.7 (verified working; later 3.x versions should work too)

Benoetigte Libraries

Library	Author	Why
Adafruit GFX	Adafruit	ST7789 display primitives
Adafruit ST7789	Adafruit	1.14" colour TFT driver
Adafruit NeoPixel	Adafruit	WS2812 status LED
DFMiniMp3	Makuna	DFPlayer Mini (non-blocking, supports playMp3FolderTrack)
SensorLib QMI8658C	Xinyuan LilyGO / TENSTAR	QMI IMU low-level driver
-- Madgwick --	-- included --	Own implementation in <code>madgwick_ahrs.h/cpp</code> , no external dep
-- iBus/SBUS --	-- included --	Inline parsers in <code>rc_receiver.cpp</code> , no external dep
-- LSM6DS3 --	-- included --	Direct I2C register access in <code>imu_handler.cpp</code>

Upload-Prozedur

- AIO32 per USB-C verbinden.
- COM-Port waehlen (Geraet erscheint als "USB JTAG/serial debug unit" oder "USB Serial").
- Wenn das Board nicht automatisch in den Bootloader geht: **BOOT** halten, **RESET** druecken, BOOT nach einer Sekunde loslassen.
- Upload druecken. Erster Upload ~90 Sekunden (esptool schreibt Bootloader + Partitionstabelle + App).
- Nach Upload **RESET** einmal druecken, um die neue Firmware zu starten.
- Serial Monitor bei **115200 Baud** oeffnen. Boot-Log zeigt I2C-Scan, IMU-Init, NVS-Load, RC-Protokoll, TFT-Init.

IMU-Setup und Kalibrierung

Die AIO32 hat zwei IMUs: **QMI8658C** auf dem TENSTAR-Modul (I2C 0x6B) und **LSM6DS3TR-C** auf der AIO32-Platine (I2C 0x6A, SA0 per Jumper auf GND). Beide liefern Accel und Gyro an ein **Madgwick-6-DoF-AHRS**, das eine Quaternion erzeugt. Daraus werden Pitch/Roll/Yaw extrahiert. Beim Boot wählst du welche IMU aktiv ist, oder lässt beide mitteln.

IMU-Modi

Modus	Verhalten
QMI_ONLY	Nur QMI8658C wird ausgelesen. Nuetzlich wenn LSM nicht bestueckt oder defekt ist.
LSM_ONLY	Nur LSM6DS3 wird ausgelesen. Nuetzlich wenn der QMI auf einer bestimmten TENSTAR-Charge instabil ist.
BOTH_AVG (Default)	Beide IMUs pro Tick. Accel und Gyro werden vor Madgwick gemittelt. Bestes Rauschverhalten.

Zur Laufzeit wechseln: `imu mode qmi|lsm|both`. Ueberpruefen mit `imu status`.

Kalibrier-Prozedur

- D-O auf ebene Flaechen stellen, perfekt aufrecht, perfekt still.
- `imu cal` in CLI senden, oder B4 lang druecken (abhaengig von Button-Binding).
- 5 Sekunden still bleiben. Die Firmware zaehlt live herunter und mittelt 500 Samples pro IMU.
- Gyro-Bias (3 Achsen pro IMU) wird von allen weiteren Readings abgezogen. Accel wird nicht bias-korrigiert (Schwerkraft ist Referenz).
- Nach der Kalibrierung `imu bias show` fuer die Offsets. Werden automatisch in NVS geschrieben.
- Die Kalibrierung ueberlebt Reboots. Nur neu kalibrieren wenn der Droid an einen neuen Ort kommt oder IMUs ausgetauscht werden.

Madgwick-Beta

Madgwick-Beta ist die Gradient-Descent-Gain, die Accel in die gyro-integrierte Quaternion mischt. Niedriger = glatter, mehr Gyro-Vertrauen, Drift-Risiko. Hoeher = snappier, mehr Accel-Gewicht, rauschiger. D-O-Default ist **0,1**. Sinnvoller Bereich 0,05..0,20.

Zur Laufzeit anpassen: `imu beta 0.12` dann `save`. Wirkt ab dem naechsten IMU-Tick (kein Reboot).

Teil II

Kernfunktionen

RC-Empfaenger-Setup (iBus + SBUS)

Die AIO32 unterstuetzt sowohl FlySky iBus (Default) als auch Futaba/FrSky SBUS. Der grosse Vorteil gegeneuber der Mega-Linie: die ESP32-UART kann ihre RX-Leitung in Hardware invertieren, **SBUS laeuft ohne externen Inverter**. Protokoll wechselt man zur Laufzeit per CLI — die Wahl wird in NVS gespeichert und ueberlebt Reboots.

Kanalbelegung

Die Kanalbelegung ist identisch zum Mega-AIO-Handbuch — gleicher Sender-Config funktioniert auf beiden Controller-Boards.

CH	Function / Funktion	FlySky default
CH1	Drive 1 (Steering in Arcade, Motor 1 in Tank)	Right Stick L/R
CH2	Drive 2 (Throttle in Arcade, Motor 2 in Tank)	Right Stick U/D
CH3	Mainbar Servo	not assigned — map to VrA
CH4	Head Servo 1 (Pitch)	Left Stick U/D
CH5	Head Servo 2 (Yaw)	Left Stick L/R
CH6	Head Servo 3 (Roll)	VrB dial
CH7	Sound Mute (2-pos)	SwA
CH8	Sound Mode / Trigger (2-pos)	SwB
CH9	Sound Mood (3-pos, neg/mid/pos)	SwC
CH10	Sound Squeak (2-pos)	free — map to any unused control

CH3-Luecke: FlySky-Sender (FS-i6, i6X, i6S) belegen CH3 standardmaessig nicht. CH3 unter "Functions → Aux. Channels" auf VrA oder einen Schalter mappen, sonst reagiert der Mainbar-Servo nicht.

iBus-Setup (Default)

- FlySky-Empfaenger mit Sender binden (Bind-Taste am Empfaenger beim Einschalten halten, am Sender Bind-Mode aktivieren).
- Empfaenger-iBus-Ausgang (einzelner Draht, meist "iBus" oder "SENS" beschriftet) an AIO32-**GPIO 2** legen.
- Ausserdem 5V und GND des Empfaengers anschliessen.
- AIO32 booten. CLI `rc status` zeigt iBus als aktives Protokoll; `rc channels` zeigt Live-Kanalwerte.

SBUS-Setup (ohne Inverter)

- FrSky-/Futaba-Empfaenger mit seinem Sender binden (Bind-Prozedur ist sender-spezifisch).
- Empfaenger-SBUS-Ausgang direkt an AIO32-**GPIO 2** legen. **Kein Inverter noetig** — der ESP32 invertiert in Firmware.

- Ausserdem 5V und GND des Empfängers.
- AIO32 booten. Protokoll per CLI umstellen: `rc protocol sbus`, dann `save`.
- Prüfen: `rc status` zeigt "SBUS" als aktiv; `rc channels` zeigt Live-Werte die sich mit den Sticks bewegen.
- Rollback: `rc protocol ibus` dann `save`.

Tipp: beide Empfängertypen kannst du im Aufbau behalten und durch Umstecken + ein CLI-Kommando wechseln. Der Droid verhaelt sich nach dem Wechsel identisch.

Button-Interface

Die AIO32 hat sechs physische Tasten **B1 bis B6**. Jede Taste deckt einen Themenbereich ab — Balance, Kalibrierung, Display, Debug, Modi, Sicherheit. Alle Tasten unterstützen Short-Press und Long-Press; Short-Press ruft die Hauptfunktion des Labels auf, Long-Press eine verwandte Zweitfunktion, wo sinnvoll. Jeder Druck wird zusätzlich im Serial-Monitor als `[btn] BN short|long` protokolliert — Verdrahtungs-Check ohne die CLI zu verlassen.

Button-Labels und Funktionen

Taste	Label	Short-Press	Long-Press
B1	Start/Stop	Balance toggeln — Start in READY, Stop in RUNNING. Aus ERROR / NOT-AUS: Fehler quittiert und Soft-Reboot.	Auto-Baseline Target-Angle — 2-Sek.-Pitch-Mittelwert im aufrechten Stand wird neuer Nullpunkt.
B2	Calibrate	IMU-Kalibrierung (Droid ~5 Sek. still halten). Gyro-Bias wird neu gelernt.	PID-Tuning-Menue am TFT oeffnen (B3 navigiert Eintraege, B4 passt Werte an, B6 schliesst).
B3	Display	Display-Modus durchschalten: Telemetry → Diagnostics → Display-Off → zurueck. Im PID-Menue: zum naechsten Eintrag.	"Always-On"-Backlight toggeln (Display schlaeft nie ein).
B4	Debug Select	Serielle Debug-Ausgabe an/aus toggeln. Im PID-Menue: den markierten Wert anpassen.	System-Info-Seite am TFT anzeigen.
B5	Mode	Drive-Mixing durchschalten (Arcade ↔ Tank).	IMU-Quelle durchschalten (QMI → LSM → FUSION).
B6	Back / Emergency	In Menues zurueck (PID-Menue verlassen). Ausserhalb von Menues: System-Info-Seite als "nichts zum Zurueck"-Bestaetigung.	NOT-AUS — Motoren + Servos sofort aus, Wechsel zu ERROR bis B1 quittiert.

Button-Events debuggen

Wenn eine Taste nicht reagiert oder die falsche Aktion ausloest: Serial-Trace aktivieren mit `debug stream buttons`. Jeder Druck wird mit dekodierter Button-Nummer und Event-Typ geloggt. Beliebige Taste im Serial-Monitor stoppt den Stream.

Serial-CLI-Referenz

Die AIO32 exponiert eine vollstaendige interaktive CLI auf USB-Serial bei **115200 Baud**. Jeder runtime-tuneable Parameter erreichbar, plus manuelle Test-Kommandos fuer Motoren, Servos, LED und Sound, plus Live-Streaming-Debug-Dumps. `help` listet die Kategorien, `help <category>` liefert Detail-Syntax in der Console. Dieses Kapitel dokumentiert jedes Kommando mit mindestens einem Beispiel — du musst nie in den Quellcode schauen.

Top-Level-Kommandos

Command	Purpose	Example
<code>help</code>	Show category list	<code>help</code>
<code>help <cat></code>	Detailed help for one category	<code>help pid</code>
<code>status / st</code>	Full system snapshot: state, IMU, RC, battery, servos	<code>status</code>
<code>version / ver</code>	Firmware version + build info	<code>version</code>
<code>save</code>	Write all runtime values to NVS (survives reboot)	<code>save</code>
<code>load</code>	Re-read NVS values, overwriting runtime state	<code>load</code>
<code>reset factory</code>	Wipe all NVS keys, reboot with defaults	<code>reset factory</code>
<code>restart / reboot</code>	Software restart (same as RESET button)	<code>restart</code>

pid — PID-Tuning

Command	Purpose	Example
pid show	Print all gains (base + 3 bands) + target + autotune state	pid show
pid kp <v>	Base Kp (used when adaptive off)	pid kp 28
pid ki <v>	Base Ki	pid ki 0.0
pid kd <v>	Base Kd	pid kd 0.8
pid slow kp <v>	Slow-band Kp ($ \text{throttle} < 0.2$)	pid slow kp 30
pid slow kd <v>	Slow-band Kd	pid slow kd 0.9
pid med kp <v>	Medium-band Kp (0.2..0.5)	pid med kp 22
pid med kd <v>	Medium-band Kd	pid med kd 0.7
pid fast kp <v>	Fast-band Kp ($ \text{throttle} > 0.8$)	pid fast kp 18
pid fast kd <v>	Fast-band Kd	pid fast kd 0.5
pid target <deg>	Balance target angle (pitch offset)	pid target -0.3
pid autotune	Start bounded slow-band Kp sweep, pick min-RMS	pid autotune

imu — IMU, Kalibrierung, Madgwick

Command	Purpose	Example
imu show	Active mode + live pitch/roll/yaw + health	imu show
imu mode qmi	Use only the QMI8658C sensor	imu mode qmi
imu mode lsm	Use only the LSM6DS3 sensor	imu mode lsm
imu mode fusion	Average both sensors (default)	imu mode fusion
imu mode auto	Fall back to whichever sensor is healthy	imu mode auto
imu cal	Run 5-second calibration (keep droid still)	imu cal
imu bias show	Print calibration offsets currently in use	imu bias show
imu bias reset	Clear offsets (next imu cal will write new ones)	imu bias reset
imu beta <v>	Madgwick gradient-descent gain (0.03..0.5)	imu beta 0.12
imu beta show	Current beta value	imu beta show

drive — Fahrdynamik

Command	Purpose	Example
drive show	All driving-dynamic settings at once	drive show
drive mode arcade	Arcade mixing (CH1 = steering, CH2 = throttle)	drive mode arcade
drive mode tank	Tank mixing (CH1 = motor1, CH2 = motor2)	drive mode tank
drive mode cycle	Toggle arcade <-> tank (same as B5 long)	drive mode cycle
drive shaping on off	RC deadband + expo enable	drive shaping on
drive deadband <v>	Stick deadzone (normalised 0..0.5)	drive deadband 0.05
drive expo <v>	Expo curve (0 = linear, 1 = heavy)	drive expo 0.35
drive ramping on off	Motor-output low-pass enable	drive ramping on
drive ramp <v>	Ramp rate (0..1; lower = smoother acceleration)	drive ramp 0.15
drive lean on off	Dynamic forward-lean enable (lean into acceleration)	drive lean on
drive leanmax <deg>	Max lean angle at full throttle (0..15)	drive leanmax 6

rc — Protokoll + Kanaele

Command	Purpose	Example
rc	Shortcut for <code>rc status</code>	rc
rc status	Active + stored protocol, connection, packet counts	rc status
rc protocol	Show currently active protocol	rc protocol
rc protocol ibus	Switch to FlySky iBus (default, no inverter needed)	rc protocol ibus
rc protocol sbus	Switch to SBUS (ESP32 HW invert, no inverter needed)	rc protocol sbus
rc channels / rc ch	One-shot dump of all channel values	rc channels

sound — DFPlayer

Command	Purpose	Example
sound show	Current volume, mute flag, track count on SD	sound show
sound volume <0..30>	Set DFPlayer volume	sound volume 22
sound mute	Mute output (transmitter can toggle too)	sound mute
sound unmute	Unmute	sound unmute
sound test <n>	Play /mp3/NNNN.mp3 immediately	sound test 3
sound stop	Stop current playback	sound stop

display — TFT

Command	Purpose	Example
display show	Current rotation + mode	display show
display rotate <0..3>	Set fixed rotation (persists)	display rotate 1
display rotate cycle	Next rotation (same as B3 short)	display rotate cycle
display mode telemetry	Live balance / battery / PID page	display mode telemetry
display mode diag	Diagnostic page with raw IMU + RC values	display mode diag
display mode off	Blank the display (backlight stays)	display mode off
display alwayson on off	Keep display on while running (same as B3 long)	display alwayson on

balance — State-Steuerung

Command	Purpose	Example
balance start	Enter STATE_RUNNING (PID active, motors live)	balance start
balance stop	Back to STATE_READY (motors idle)	balance stop
balance estop	Emergency stop: motors cut, STATE_ERROR (same as B6 long)	balance estop
balance baseline	Auto-null target angle — 2-second pitch average becomes new target (same as B1 long)	balance baseline

feature — Runtime-Flags

Vier Flags sind zur Laufzeit umschaltbar. Zwei weitere (`adaptive_pid` und `madgwick`) sind Compile-Time-Flags in `config.h` — `feature show` zeigt sie an, damit du den aktuellen Build kennst, aber sie koennen ohne Re-Flash nicht umgeschaltet werden.

Command	Purpose	Example
<code>feature show</code>	All flags (runtime + compile-time) and their current state	<code>feature show</code>
<code>feature state_rx on off</code>	Tilt-warn + recovery sound triggers	<code>feature state_rx on</code>
<code>feature idle_sound on off</code>	Idle mood sound triggers	<code>feature idle_sound on</code>
<code>feature idle_anim on off</code>	Idle servo animations (mainbar + head)	<code>feature idle_anim on</code>
<code>feature sound on off</code>	Master sound enable (also toggles DFPlayer mute)	<code>feature sound on</code>

test — Manueller Hardware-Test

Command	Purpose	Example
<code>test motor1 <-255..255></code>	Spin motor 1 at signed PWM value	<code>test motor1 120</code>
<code>test motor2 <-255..255></code>	Spin motor 2 at signed PWM value	<code>test motor2 -80</code>
<code>test motors stop</code>	Stop both motors immediately	<code>test motors stop</code>
<code>test servo <name> <0..180></code>	Drive a servo: mainbar, head1, head2, head3	<code>test servo mainbar 120</code>
<code>test servo center</code>	All four servos back to 90 deg	<code>test servo center</code>
<code>test led <r> <g> </code>	Set NeoPixel RGB (0..255 each)	<code>test led 0 255 0</code>
<code>test sound <n></code>	Alias for <code>sound test n</code>	<code>test sound 7</code>

debug — Dumps + Live-Streams

Command	Purpose	Example
debug rc	One-shot RC channel dump	debug rc
debug imu	One-shot raw IMU dump (accel + gyro per sensor)	debug imu
debug buttons	One-shot button status dump	debug buttons
debug batt	One-shot battery voltage + divider factor	debug batt
debug stream rc [ms]	Live RC stream (default 200 ms). Any key stops.	debug stream rc 100
debug stream imu [ms]	Live IMU angles + gyro rates	debug stream imu 200
debug stream buttons [ms]	Live button events (default 50 ms)	debug stream buttons
debug stream batt [ms]	Live battery voltage (default 1000 ms)	debug stream batt 500
debug stream off	Stop the stream (any keystroke also stops)	debug stream off

Tip: am Ende einer Tuning-Session jeden Wert mit eigenem Kommando setzen und mit `save` abschliessen, um alle Aenderungen persistent zu machen. Die CLI verarbeitet ein Kommando pro Zeile; zusammengesetzte Ketten werden nicht unterstuetzt, aber mehrere Zeilen hintereinander reinpasten funktioniert.

Balance und PID-Tuning

D-O balanciert auf zwei Raedern. Ein PID-Regler nimmt den Pitch-Error (gemessener Pitch minus Target-Pitch) und erzeugt eine Motor-Korrektur, die beide Motoren gleich anwenden. Die AIO32 nutzt eine **adaptive 3-Band-PID**: separate kp/kd-Gains fuer Slow / Medium / Fast Rad-Geschwindigkeit, linear gebendet. So ist der Droid im Stand stabil und bei Fahrt agil ohne einen Kompromiss-Wert.

Empfohlene Startwerte

Parameter	Default	Range
kp_slow	28.0	20..40
kp_medium	22.0	15..35
kp_fast	18.0	10..30
kd_slow	0.9	0.3..1.5
kd_medium	0.7	0.3..1.5
kd_fast	0.5	0.2..1.2
ki (global)	0.0	0..0.5 (leave 0 unless you see a steady lean)
target_angle	-0.3 deg	-2.0 .. +2.0 (baseline via B1-long)
max_integral	400	200..800
madgwick_beta	0.10	0.05..0.20

Tuning-Prozedur

- **Auf einem Stand anfangen**, Raeder frei in der Luft, 4S-Pack dran.
- **Baseline** zuerst: **B1 (Start/Stop)** lang, oder `balance baseline`. Pitch muss in Ruhe nahe 0 Grad zeigen.
- **Balance einschalten** mit **B1 (Start/Stop)** kurz, oder `balance start`. Droid sollte kleine Zuckungen machen aber nicht wild oszillieren.
- Wenn er schnell oszilliert: `kp_slow` in 2er-Schritten reduzieren bis stabil.
- Wenn er absackt und dann zurueckkommt: `kp_slow` erhoehen.
- Wenn er beim Anstossen nachschwingt: `kd_slow` in 0,1er-Schritten erhoehen.
- Wenn Band *slow* stabil ist: *medium* und *fast* entsprechend. Fuer *fast* auf den Boden stellen und fahren; Oszillation beim Abbiegen beobachten.
- Mit `save` sichern.
- Wenn der Droid nach Baseline noch im Stand vor/zurueck driftet: `pid target` in 0,1-Grad-Schritten anpassen. Klein negativ (-0,3 bis -0,8) ist typisch.

Tipp: `pid autotune` macht ein begrenztes Slow-Band-kp-Sweep und waehlt den Wert mit minimalem RMS-Pitch-Error. Guter Startpunkt fuer einen neuen Droid, aber immer noch per Hand nachtunen fuer den konkreten mechanischen Aufbau.

Sound (DFPlayer Mini + SD)

Die AIO32 steuert einen DFPlayer Mini ueber UART (GPIO 9 TX, GPIO 10 RX) mit der Makuna-**DFMiniMp3**-Library. File-Adressierung ist **filename-basiert** via `playMp3FolderTrack(N)` — die SD-Karten-Kopierreihenfolge ist egal, der Chip sucht `/mp3/NNNN.mp3` direkt.

Track-Belegung (identisch zur Mega-Linie)

Track	Function / Funktion
0001	Startup ("battery charged")
0002	Default ("I am D-O")
0003–0005	Greetings
0006–0009	Negative moods
0010–0014	Positive moods
0015–0020	Squeaky wheel
0021	Tilt warning (droid leaning >15 deg)
0022	Recovery (droid re-centred after tilt)
0023	Low battery
0024–0030	Idle sounds (played while in READY state for N seconds)
0031	System ready (post-boot)
0032	Signal lost

SD-Karten-Format + Dateinamen

- Dateisystem: **FAT16** oder **FAT32**. Aktuelles Windows formatiert gerne exFAT — DFPlayer liest KEIN exFAT.
- Ordner: `/mp3/` (klein). Bei Bedarf manuell im Root anlegen.
- Dateiformat: **MP3** oder **WAV**. Beides laeuft. MP3 ist kleiner und reicht fuer Sprach-Clips; WAV vermeidet jegliche Decoder-Eigenheiten bei sehr kurzen Effekten.
- Dateinamen-Regel: nur die ersten **vier Ziffern** sind die Track-Nummer, der Rest ist frei. `0001.mp3`, `0001_startup.mp3`, `0001_battery_charged.wav` sind alle Track 1 — nimm was du lesbar findest.
- Fuehrende Nullen sind Pflicht: `1.mp3` funktioniert NICHT; es muss `0001.mp3` sein (oder aehnlich mit dem 4-stelligen Praefix).
- Kartengroesse: 2 GB..32 GB gut. 64 GB+ oft defaultmaessig exFAT — neu als FAT32 formatieren.
- Bitrate: 128..192 kbps Mono-MP3 reicht. Hoeher geht, verschwendet aber Platz.

Lautsprecher-Anschluss

Das AIO32-Board hat **zwei Lautsprecher-Ausgaenge**, jeweils mit eigenem **+** und **--**-Anschluss. Beide werden **direkt vom DFPlayer Mini getrieben** — kein externer Verstaerker auf der Platine, und keiner noetig. Die zwei Ausgaenge sind parallel am Mono-Audio-Teil des DFPlayers, sie tragen also dasselbe Signal. Du kannst nutzen:

- **Einen Lautsprecher** an einem der beiden Ausgaenge — der andere bleibt leer.
- **Zwei Lautsprecher parallel** (je einer pro Ausgang), z.B. einen im Kopf und einen im Koerper. Beide spielen dasselbe Mono-Signal.

Empfohlene Daten pro Lautsprecher: **8 Ω , 2-3 W, 28-40 mm Durchmesser**. Zwei 8- Ω -Typen parallel ergeben 4 Ω Gesamtlast — noch im Rahmen, aber die Lautstaerke sollte dann nicht bis Maximum gehen. Bei parallelem Betrieb die +/- Markierung unbedingt beachten, sonst loeschen sich die beiden Lautsprecher phasenmaessig aus.

Volume und Mute

Volume 0..30. Default 25. Per CLI `sound volume 22`, `sound mute`, `sound unmute`, `sound test 5` (spielt Track 0005). Mute auch vom Sender (CH7 — SwA Default) oder per CLI `sound mute`.

Display und Status-LED

Die AIO32 hat zwei visuelle Indikatoren: ein 1,14" Farb-TFT fuer Live-Telemetrie und eine einzelne NeoPixel, die den System-State durch Farbe und Muster anzeigt.

Display-Modi

Das Display orientiert sich automatisch nach `imu mode` und letzter physischer Orientierung, oder bleibt fix wenn du willst. Drei Rotationen per **B5-kurz** (oder `display rotation auto|portrait|landscape`):

Modus	Zweck
auto	Display folgt der Droid-Orientierung — nuetzlich bei Bau/Test auf der Bench
portrait	Fix Hochformat (240 hoch, 135 breit). Haupt-Runtime-Modus
landscape	Fix Querformat (240 breit, 135 hoch). Nuetzlich bei Wartung im Halter

Always-On vs Auto-Sleep

B5-lang toggelt Always-On. Wenn aus, schlaeft das Display nach 5 Minuten Inaktivitaet in READY (nie in RUNNING). Langfristiges Always-On kann bei OLED-aehnlichen Displays Burn-In verursachen; das ST7789 ist unanfaellig, aber die Backlight-LEDs leben laenger wenn sie bei Idle aus sind.

NeoPixel-State-Muster

State	Farbe und Muster
INIT	Gelb konstant
CALIBRATING	Cyan-Puls (1,2 s Periode)
READY	Gruen konstant (oder Orange-Puls bei <code>batteryWarnActive</code>)
RUNNING	Blau-Puls (2 s Periode)
ERROR	Rot konstant
EMERGENCY_STOP	Rot langsames Blinken (400 ms)
<code>batteryCriticalActive</code>	Rot schnelles Blinken (200 ms) — ueberschreibt alle anderen Muster

Akku-Monitoring

Die AIO32 misst die Pack-Spannung ueber einen 47k/10k-Teiler auf GPIO 16 (ADC2). Warn- und Critical-Schwellen sind konfigurierbar; bei Critical uebernimmt ein schnelles Rotblinken der NeoPixel, Motoren werden gesperrt — LiPo-Schutz. Akku-Monitoring ist per Default aus — per Compile-Flag `BATTERY_MONITOR_ENABLED` in `config.h` aktivieren, sobald du den Teiler-Faktor gegen deinen Pack verifiziert hast.

Konfiguration

Parameter	Default	Bereich
<code>BATTERY_MONITOR_ENABLED</code>	false	true nach aktivem Teiler
<code>battery_warning</code>	13,6 V	3,4 V pro Zelle x 4
<code>battery_critical</code>	12,8 V	3,2 V pro Zelle x 4
<code>voltage_divider</code>	5,74	$(R1+R2)/R2$ berechnen; Default passt zu 47k/10k

Kalibrierung

Zur genauen Teiler-Kalibrierung: Pack-Spannung mit Multimeter messen, dann in CLI `debug batt`, um den ADC-Wert zu sehen. Wenn die Anzeige um X% daneben liegt: `voltage_divider` um den gleichen % anpassen. Save. Nochmal pruefen.

Teil III

Referenz & Anhaenge

Fehlersuche

Erster Schritt bei jedem Problem: `status` in der CLI liefert einen vollstaendigen Snapshot von IMU, RC, Akku, PID, State. Zweiter Schritt: `debug stream <kind>` fuer Live-Daten.

Boot haengt bei "DFPlayer init..."

- SD-Karte fehlt oder nicht FAT32 — formatierte Karte einlegen und rebooten
- DFPlayer TX/RX vertauscht — GPIO 9 ist AIO32-TX (geht an DFPlayer-RX via 1k-Widerstand), GPIO 10 ist AIO32-RX (von DFPlayer-TX direkt)
- Workaround fuer Boot-Durchkommen: `SOUND_CONTROLLER_ENABLED=false` in `config.h`, neu flashen, Verdrahtung fixen, wieder aktivieren

Droid kippt in Ruhe auch nach Kalibrierung

- `balance baseline` (oder B1 lang) mit perfekt aufrechtem Droid laufen
- Wenn immer noch schief: `pid target` in 0,1-Grad-Schritten anpassen
- Physische Rad-Symmetrie pruefen — unterschiedliche Raddurchmesser kippen den Target-Angle

RC zeigt keine Kanaele / alles Null

- `rc status` muss `connected=Yes` zeigen. Wenn No: Verdrahtung an GPIO 2 pruefen.
- iBus: Empfaenger gebunden? Sender an?
- SBUS: `rc protocol sbus` aktiviert? `save` gemacht?
- `debug stream rc` fuer Live-Bytes. Kanaele muessen 1000..2000 mit Sticks wandern

Buttons loesen falsche Aktionen aus

- `debug stream buttons` zeigt Live-Button-Events
- Wenn eine Taste nie ausloest: Verdrahtung am B1..B6-Header pruefen
- Wenn die falsche Button-Nummer gemeldet wird: mit dem Debug-Output an den Support wenden — die Dekodierungs-Schwellen muessen ggf. fuer deine Charge angepasst werden

Servo zuckt beim Boot heftig

- LEDC-Channel-Kollision oder Stromunterbrechung. 5V-BEC muss Strom fuer alle vier Servos bei Blockade liefern koennen
- Alle Servo-Massen muessen an BEC-GND UND AIO32-GND haengen

Display zeigt Muell oder schwarz

- ST7789 nutzt SPI auf AIO32-internen Pins; wahrscheinlich schlechte Verbindung am TENSTAR-Modul-Header
- `display mode splash` fuer Test-Muster

Tilt-Warn feuert beim falschen Winkel (z.B. 15 Grad als 5 Grad erkannt)

- Madgwick-Quaternion daneben — `imu cal` neu laufen
- `imu beta` temporaer auf 0,05 reduzieren und Stabilitaet beobachten

- Anderen IMU-Modus probieren: `imu mode qmi` vs `imu mode lsm` vs `imu mode both` — ein Sensor koennte Rauschen beisteuern

NVS liest Unsinn nach Factory-Reset

- Sollte nicht passieren — Factory-Reset wischt die NVS-Partition. Wenn doch: volle Firmware neu flashen (nicht nur App-Partition) und IMU neu kalibrieren

Sicherheit

Der vollstaendige Haftungsausschluss steht auf Seite 2. Dieses Kapitel listet die praktischen Sicherheitsregeln fuer den Betrieb von D-O.

Regel	Details
LiPo-Handhabung	LiPo-faehiges Ladegeraet. Nicht unbeaufsichtigt laden. Aufgeblaehnte Packs ersetzen.
Polung	Jede Stromverbindung vor dem Einschalten verifizieren. Verkehrte Polung kann ESP32 oder Motortreiber sofort beschaedigen.
Spannung	4S LiPo: 16,8 V voll, 14,8 V nominal, 12,0 V leer. Nicht unter 3,0 V pro Zelle entladen.
Servo-Stromversorgung	Separater 5-6V-BEC fuer Servos (3-5 A). Servos NICHT ueber die 3,3-V-Schiene des ESP32 speisen.
Erstes Einschalten	Raeder in der Luft. Die PID kann den Droiden waehrend Kalibrierung / Tuning heftig bewegen.
Not-Aus	Long-Press B6 (Back/Emergency) schaltet Motoren und Servos sofort ab. Vor der ersten Bodenfahrt den Griff dahin ueben.
Umgebung	Bereich frei halten. Ein umkippende Droid mit 4S-Strom kann Haende, Tiere oder Hardware verletzen.
SD-Karten-Handling	DFPlayer liest nur FAT16/FAT32. Vor dem Neu-Formatieren sauber auswerfen — kaputte SD-Karten verursachen Boot-Hangs.