Version 9.15 BB8

SHADOW version of Joe's Drive V2 / Z-Drive v9.15

Feature	Status	Notes
Serial1 → IMU Board	BACKWARDS	Unchanged, proven, rock-solid
I2C → DOME CONTROLLER (32u4)	NEW	Full 9.1 data, zero loss
SharedStructs.h	NEW	Memory layout identical
Calibration + 3-second hold	NEW	Saves S2S center, pitch, roll
Debug system	BACKWARDS	Clean, conditional, zero overhead
Motor control logic	NEW	Provides precise and m
Uses SHADOW controls (PS Move Nav controllers or Xbox Controller for Dome and Drive	NEW	Dual controllers, exact behavior as the custom controller but with supported backing from Sony or Microsoft
PID balance (pitch & roll)	BACKWARDS	Same tuning, same feel
Flywheel, dome spin, sound, servos	BACKWARDS	Same tuning, same feel
Code structure & comments	NEW	Clean, readable, future-proof

System Overview

The last version of Joe Drive v2 began with Joe in 2020, was later taken over by James VanDusen, and has changed greatly in five years. The PCB is crucial, featuring two main CPUs and an onboard sound system, allowing both current and future hardware adaptation. Numerous changes have been made to allow better accuracy with the Side 2 Side movement (gearing) and the main drive system has adapted both chain as well as gear based drive motion.

Architecture (high level)

- Primary CPU (ESP32 Huzzah Feather): Handles Bluetooth (two PS3 Navigation controllers), high-level drive commands, IMU processing ESPNOW to the DomeController, Remote Serial controls, and EasyTransfer serial link to the 32u4. [github.com]
- Secondary CPU (Feather 32u4): Executes dome motor/servo control, encoder counting, NeoPixel animations, DFPlayer sound triggers, hall sensor, and hardware-level timing tasks. [github.com]
- IMU CPU (TrinketM0): Executes both raw and calculated angles for pitch, roll and Z.
- Comms: ESP32 ↔ TrinketM0 via Serial2 on the Feather (EasyTransfer messages. ESP32 ↔ 32u4 via Serial1 on the Feather (EasyTransfer messages. [github.com]
- Sound: DFPlayer Mini controlled from the 32u4, BUSY pin monitored, audio tracks triggered by commands/events. (BUSY behavior and best practices widely documented.) [forum.arduino.cc], [forum.arduino.cc]
- Lighting: NeoPixel strip on D13 with MOSFET level shifter (SparkFun BOB-12009), 330 Ω series resistor, and 1000 µF bulk cap at 5 V rail. [github.com]
- Dome Drive: NeveRest encoder (Phase A on D2 interrupt, Phase B on A0), dome motor driver (PWM on D3; dir D5/D6), servos on D11/D12. [github.com]

Controllers & Pairing (Two PS3 Navigation Controllers)

By default we will be using two Bluetooth PS3 Navigation controllers—one mapped to Drive and one to Dome. PS3 controllers need the host's Bluetooth MAC address set in the controller (or you set the ESP32's MAC to the controller's). The process is standard:

Standard Operations

Normal Mode Controls & Sounds — PS3 Move Navigation Controllers

Movement

Drive (Right controller)

- Right stick **Y**: Forward/Reverse. Respects `reverseDrive` toggle; Pitch PID auto-balance adds/subtracts when stick is in deadband (if `autoBalance` ON).

- Right stick **X**: Side-to-Side (S2S) manual when |X| > deadband and not tuning.

Dome (Left controller)

Left stick X/Y: Tilt commands sent to 32u4 servos (`leftStickX`, `leftStickY`).

L1 COMBO: Dome Spin vs Flywheel for Drive Controller

Drive L1 pressed → `domeSpin = rightStickX`; **flywheel disabled**; suppress rightStickY (no forward drive).

Dome L1 pressed → `EnableFlywheel = true`; `domeSpin = 0`; flywheel uses **Right stick X** (Drive controller).

Runtime Toggles (edge-triggered)

- **PS (Drive)**: `enableDrive` ON/OFF → sends `driveEnabled`; plays toggle sound.

- **L3 (Dome)**: `DomeServoMode` ON/OFF → sends `moveR3`; plays toggle sound.

- **L3 (Drive)**: `reverseDrive` ON/OFF → sends `moveL3`; plays toggle sound.

- **Cross / X (Drive)**: `autoBalance` ON/OFF (Pitch/S2S PIDs enabled when sticks in deadband).

Sounds — Both Controllers

Drive Controller (Right)

- **D-pad (no L1/R1)** → `UP: 1`, `RIGHT: 2`, `DOWN: 3`, `LEFT: 4`

- **L1 + D-pad** → `UP: 21`, `RIGHT: 22`, `DOWN: 23`, `LEFT: 24`

- **R1 + D-pad** → `UP: 25`, `RIGHT: 26`, `DOWN: 27`, `LEFT: 28`

- **L2 threshold** (rising past 60%) → `50`

- **Circle (O)** → random `1..30`

Dome Controller (Left)

> Resolved by `resolveDomeControllerSound(buttonsL, prevL)`.

- **D-pad (no L1/R1)** → *(fill per your mapping)*

- **L1 + D-pad** → *(fill per your mapping)*

- **R1 + D-pad** → *(fill per your mapping)*

- **L2 threshold** (rising past *threshold*) → *(fill per your mapping)*

- **Circle (O)** → random *(range)*

Safety & Gating

- Drive runs only when **controller connected**, **tuning = NONE**, **enableDrive = true**, and |speed| > 5.

- S2S runs only when the same conditions hold **and neither L1 is pressed**. During **ROLL_DK / ROLL_IK** tuning, S2S PID is allowed.

- **Cancel tuning (PS+DOWN 3s)** restores persisted gains and **neutralizes drive + S2S PWM** immediately.

Pairing Steps

Get/Set MAC in the PS3 controller:

Use SixaxisPairTool (Windows) or sixaxispairer (open source) to read/write the stored MAC in each PS3 Nav controller. [github.com]

In ESP32 app code:

If you set the controllers to a known MAC, update the ESP32's Bluetooth MAC (if needed) before initializing the PS3 library. Example: ps3SetBluetoothMacAddress() then ps3Init(). [github.com]

Initialize the library in Arduino (ESP32):

Install esp32-ps3 library and call Ps3.begin("XX:XX:XX:XX:XX:XX") if you want ESP32 to emulate the console MAC; otherwise use controller's stored MAC approach. [github.com], [dfrobot.com]

For a step-by-step tutorial (screenshots & examples), DroneBot Workshop's article is excellent. [dronebotworkshop.com]

Tip: Label your controllers physically ("Drive" & "Dome"). In code, you can distinguish controllers by MAC and assign each to a control context.

Live PID Tuning & Calibration — Drive Controller chords**

Enter / Exit

- **Pitch Tuning**: Hold **PS + UP** (Drive) for 3 s → Pitch PID (KP → KD → KI).

- **Roll/S2S Tuning**: Hold **PS + RIGHT** (Drive) for 3 s → Roll PID (Pk2 → Dk2 → Ik2).

- **Cancel Tuning**: Hold **PS + DOWN** (Drive) for 3 s → Restore persisted gains, disable auto-balance, neutralize outputs.

- During tuning: **joysticks, sounds, dome, flywheel disabled**; use **D-pad UP/DOWN** to adjust and **X** to store/advance.

Pitch (Drive) PID Stages

1. **KP**: start `KP=45.0, KI=0, KD=0`; **autoBalance OFF**.
   - UP/DOWN: step **±1.0**
   - **X**: store `KP` with **25% reduction**, persist, advance to KD
2. **KD**: **autoBalance ON** to observe damping.
   - UP/DOWN: step **±1.0**
   - **X**: store KD, persist, advance to KI
3. **KI**:
   - UP/DOWN: step **±0.1**
   - **X**: store KI, persist, exit tuning

Roll (S2S) PID Stages

1. **Pk2**: start `Pk2=45.0, Ik2=0, Dk2=0`.
   - UP/DOWN: step **±1.0**

  - **X**: store Pk2 with **25% reduction**, persist, advance to Dk2

2. **Dk2**: S2S auto-balance active for lateral damping.

  - UP/DOWN: step **±1.0**

  - **X**: store Dk2, persist, advance to Ik2

3. **Ik2**:

  - UP/DOWN: step **±0.1**

  - **X**: store Ik2, persist, exit tuning

Calibration (3-sec holds; both controllers)

- **SAVE offsets**: Hold **UP+UP** → saves `potOffset`, `pitchOffset`, `rollOffset` to Preferences

- **RESET offsets**: Hold **DOWN+DOWN** → zeros offsets and persists

Persistence & Cancel

- Stored gains: `kp_pitch`, `kd_pitch`, `ki_pitch`, `pk2`, `dk2`, `ik2`

- Cancel tuning restores persisted gains, **disables auto-balance**, **neutralizes drive & S2S PWM**

Control Mapping

If you want me to read your specific ESP32_Primary_v9.1.ino & 32u4_Secondary_v9.1.ino and extract exact button mappings, I can do that next and tailor the tables; for now I'll propose a clean, BB-8-friendly layout consistent with typical PS3 Nav ergonomics.

A) Drive Controller (PS3 Nav #1)

Analog Stick (primary):

Y axis (forward/back): Main drive throttle

X axis (left/right): Body steer (yaw)

L1: Drive "precision mode" (reduces max speed to 40%)

L2 (trigger): Turbo burst (momentary +20% throttle ceiling)

PS (center): System arming / emergency stop toggle (safe state)

Cross (X): Start/Stop autonomous straight roll (IMU stabilized)

Circle (O): Reverse mode toggle

Triangle (Δ): Brake/hold (zero throttle, motor brake engage)

Square (□): Lights on/off (body glow, status LED)

B) Dome Controller (PS3 Nav #2)

Analog Stick:

X axis: Dome rotation (encoder closed-loop)

Y axis: Dome tilt (servos X/Y as applicable)

L1: Fine dome rotation (slow speed)

L2 (trigger): Fast dome rotation (high speed)

Cross (X): Play random BB-8 chatter (DFPlayer track group)

Circle (O): Whistle/acknowledge sound

Triangle (Δ): “Head nod” servo macro (gesture)

Square (□): NeoPixel pattern cycle (dome lights)

This is a sane mapping for live ops. I’ll swap labels to your actual functions once I parse your code, but it gives you structured documentation now.

---

4) Sound (DFPlayer) — Operator Instructions

Track organization: Put audio files in MP3 folder and trigger by index (playMp3Folder(n)) for predictable playback; many DFPlayer guides recommend this for reliability. [forum.arduino.cc]

Busy pin behavior: The BUSY pin reads LOW while playing on most modules. You can gate NeoPixel updates and movement cues on BUSY, avoiding clashes. [forum.arduino.cc]

Usage in ops:

Press Cross (X) on Dome controller to play chatter.

Press Circle (O) for special effects (beeps/whistles).

During playback, avoid rapid track retriggers—BUSY will be LOW; queue next event after BUSY returns HIGH. [github.com]

---

5) Movement & Functionality

Start-up sequence

Power on external 5 V rail (NeoPixel/DFPlayer) and system battery.

Feather/ESP32 boot; wait for controller pairing messages on the serial console. [espressif.github.io]

Calibrate IMU (if your sketch does it at start); keep BB-8 stationary for ~3–5 s.

Manual driving

Drive controller stick forward/back to move, left/right to steer.

Use precision mode (L1) indoors or crowded areas.

Turbo (L2) outdoors when safe.

Dome operation

Dome controller stick left/right to rotate the head; speeds vary by L1/L2 modifiers.

Use head nod macro (Triangle) for audience engagement.

Switch NeoPixel patterns (Square).

Safety

PS button on Drive controller toggles armed/safe. In safe state: motors disabled, sounds muted except system alerts.

---

6) Status & Feedback

NeoPixel colors/patterns:

Blue pulse: Ready/paired.

Amber sweep: Playing sound.

Red flash: Fault (hall sensor or encoder mismatch).

DFPlayer BUSY: Used internally to synchronize. [forum.arduino.cc]

---

7) Troubleshooting

Controller won't connect: Re-pair using SixaxisPairTool; confirm MAC alignment and ESP32 PS3 library is initialized (Arduino or IDF). [github.com], [dfrobot.com]

Audio skips or random tracks: Use playMp3Folder() and confirm SD card naming convention; avoid polling BUSY with blocking logic. [forum.arduino.cc]

NeoPixel glitches: Ensure level shifter in place; keep 1000 µF cap near strip; avoid strip.show() during heavy serial RX (schedule around DF BUSY). [github.com]

Dome drift: Check encoder pull-ups to 3.3 V; reduce ISR debounce to 100–200 µs for higher RPM accuracy. [github.com]

---

8) Quick Reference (Operators)

Drive Controller

Stick: throttle/steer

L1: precision

L2: turbo

PS: arm/safe

Δ: brake/hold

☐: body lights

X/O: custom actions (assign per code)

Dome Controller

Stick: rotate/tilt

L1/L2: slow/fast

X/O: sound triggers

Δ: nod macro

☐: dome light pattern

---

9) Where this comes from

Your repository, architecture notes, and files (README, PCB, SourceFiles). [github.com], [github.com]

PS3 controller pairing & ESP32 library documentation. [github.com], [dronebotworkshop.com], [dfrobot.com]

DFPlayer BUSY/MP3 folder best practices. [forum.arduino.cc], [forum.arduino.cc]

Absolutely, James. I can walk you (or any builder) from **printing the parts** in the STLs folder to **mechanical assembly**, **electronics installation**, **initial bring-up**, and **calibration** for your Z-Class BB-8 drive. I've reviewed your repo structure and support files to anchor the steps. [github.com]

Below is a clean, field-tested assembly plan you can drop into your project docs. If you want, I'll turn this into a **Markdown README** inside your STLs or Support folder later.

Before You Start

Get the files & references

- **Repo root** (overview, BOM, high-level architecture): Z-ClassDriveSystem [github.com]
- **STLs folder**: printable parts for the drive and mechanisms (the folder is visible in repo tree). [github.com]
- **PCB folder**: zipped **MAINBOARD**, **DOME**, **IMU** board designs (for wiring reference and mounting hole patterns). [github.com]
- **Z-Drive BOM.xlsx**: complete parts list; use this for fast ordering. [github.com]
- **ESP32 Proposed Solution v76.pdf / README.md**: architecture and control-flow notes (good background). [media.distrelec.com], [forums.autodesk.com]

Tools & materials

- FDM printer: 0.4 mm nozzle, PLA+ or PETG recommended
- Fasteners: M3 socket-head cap screws, assorted lengths (10–25 mm), M3 washers, M3 nyloc nuts (per BOM) [github.com]

- Bearings & chain-drive components (sprockets/chain) per your BOM & frame design; DFRobot 0601 H-bridge driver for motor stage (as noted in README) [forums.autodesk.com]
- NeveRest Classic 60 gearmotor (with encoder) for dome rotation (as referenced in your design narrative)
- Basic hand tools: metric hex keys, calipers, soldering iron, Loctite 243 (blue), thread tap for M3 (optional), flush cutters

---

3D Printing Guidelines (STLs)

The STL filenames reflect assemblies (e.g., **Frame**, **Gantry**, **HeadTilt**, **Flywheel**, etc.). The following print settings work well:

- **Layer height:** 0.20 mm (0.16 mm for cosmetic parts)
- **Walls:** 3-5 perimeters
- **Infill:**
    - **Structural parts** (drive frame, gantry arms, chain guards): **15%** grid/gyroid
    - **Covers/decor**: **15–20%**
- **Material:** PETG for strength; PLA+/ABS for appearance or heat resistance
- **Orientation & supports:** respect hole directions and flat-face mating surfaces; add tree supports under overhangs only
- **Post-processing:** deburr holes; ream bearing bores; chase M3 threads if tight

Mechanical Assembly (Gantry Drive)

1. **Main frame**
    - Assemble the printed **Frame** components; insert M3 hardware loosely first.
    - Install **idler bearings** and **motor mount** brackets (per STL set).
    - Verify chain alignment with sprocket centers before tightening fasteners.
2. **Chain drive & sprockets** (your repo mentions switching to chain for slip-resistance)
    - Mount drive sprocket on motor shaft; align idler sprockets in the frame.
    - Fit the chain; set **~1–2% slack** to prevent binding under load.
    - Rotate by hand to check smooth motion end-to-end. [forums.autodesk.com]
3. **Gantry / flywheel & center of mass**
    - Install the low-profile **flywheel** subassembly (noted in README as “v2 Flywheel”) to maintain CG; ensure clearance with battery bay. [forums.autodesk.com]
    - Add battery tray and straps; confirm pack removal is possible with the frame installed.
4. **Motor driver (DFRobot 0601)**
    - Mount to designated printed standoffs; route motor leads through printed cable guides.
    - Keep high-current wires short and twisted to minimize EMI. [forums.autodesk.com]

Dome Tilt Subassembly

1. **Dome rotation motor (NeveRest Classic 60)**
    - Bolt the motor to the **dome drive plate**; install encoder harness (A on D2, B on A0 at 32u4).
    - Add **pull-ups 4.7–10 kΩ to 3.3 V** on A/B if using open-collector encoder outputs (common practice).
    - Route encoder wires away from motor power leads to reduce noise.
2. **Tilt & head-nod servos**
    - Mount servos on the **HeadTilt** printed bracket (per the STLs).
    - Check free travel without binding; verify horn alignment at neutral (center) position.
3. **Dome light ring & NeoPixel DIN**
    - Install the SparkFun **BOB-12009** level shifter near the NeoPixel entry; feed DIN via **330 Ω series**.
    - Place a **1000 µF** cap across 5 V/GND at the strip inlet.
    - Tie grounds to the star point near the main 5 V rail. (These practices appear in your final wiring and are consistent with NeoPixel best-practice.) [github.com]

Electronics Integration (Main Board, ESP32, 32u4)

1. **Main board installation**
    - Use the **MAINBOARD** zip drawings (hole pattern) to set standoffs in the frame print; mount the PCB; leave access for USB on ESP32 and Feather 32u4. [github.com]

2. **ESP32 Huzzah Feather (Primary)**
    - Connect the two **PS3 Nav controllers** via Bluetooth; set MAC pairing as per the PS3 library instructions (SixaxisPairTool / esp32-ps3). [docs.cirki...signer.com], [docs.cirki...signer.com]
    - Wire **Serial1** (pins 0/1 on Feather) to the secondary 32u4 per your harness.
3. **Feather 32u4 (Secondary)**
    - DFPlayer: **D9 RX** via **20 kΩ/10 kΩ** divider from DF TX; **A4 TX** via **1 kΩ** series to DF RX; **D10 BUSY** with INPUT_PULLUP (or divider if BUSY idles at 5 V). (BUSY behavior references widely documented.) [electrocredible.com], [forums.autodesk.com]
    - NeoPixel: **D13 → BOB-12009 LV1 → HV1 → 330 Ω → DIN**; LV=3.3 V, HV=5 V, and **common ground**.
4. **Power & 5 V star-ground**
    - External 5 V rail feeds NeoPixel, DFPlayer, and can be shared with Feather/ESP32 USB VBUS; **prevent USB backfeed** to a PC by OR-ing or schottky isolation if you tether USB during ops.
    - Decoupling: **0.1 µF** near modules; bulk electrolytic on the NeoPixel rail.

Initial Bring-Up & Calibration

1. **Mechanical**
    - Spin each motor by hand; verify chain tension and limit clearances.
    - Check servo neutral positions; adjust horns.
2. **Electrical (no load)**

- Power 5 V; confirm **DFPlayer** responds to simple play commands and BUSY toggles properly. [electrocredible.com]
- Light test: one-pixel NeoPixel pattern; ensure clean updates (no brownouts).

3. **Controllers & comms**
   - Pair both PS3 Navs; confirm the ESP32 detects two distinct controller MACs and maps them to **Drive** and **Dome** contexts. [docs.cirki...signer.com]
   - Check **EasyTransfer** link (ESP32↔32u4): verify dome rotation commands and light triggers traverse serial.
4. **Encoder & dome loop**
   - Rotate dome slowly; confirm rising edges on D2, B-phase read on A0; set ISR debounce to **100–200 µs** (avoid 500 µs at higher RPM).
   - Validate "center" tick and drift correction as defined in your code constants.

Operating Notes (from your architecture)

- Use **precision mode** for close-quarters driving; turbo outdoors only.
- Gate **NeoPixel show()** away from DFPlayer RX windows (BUSY low → playing); this prevents serial drops with software UART. (Common best practice aligned with your design.) [github.com]
- Keep wiring short and twisted for motor leads; separate encoder lines from PWM paths.

Appendix — Printing & Hardware Checklist

Print list

Frame*Base*A.stl

Frame*Side*Left.stl

Frame*Side*Right.stl

Chain*Idler*Bracket.stl

Motor*Mount*Neverest.stl

Gantry*Arm*L.stl

Gantry*Arm*R.stl

Flywheel*Plate.stl*

*HeadTilt*Bracket.stl

Dome*Ring*Mount.stl

MainBoard*Standoffs.stl*

*DomeBoard*Standoffs.stl

Cable*Guide*xN.stl

(Use the actual filenames from your STLs folder.) [github.com]

Hardware (refer to Z-Drive BOM.xlsx for exact counts):

- M3 SHCS: **10 mm / 16 mm / 25 mm**
- M3 washers & nyloc nuts
- Idler **608/688** bearings (per design)
- Chain & sprockets (module per BOM)
- NeveRest Classic 60 gearmotor (with encoder)
- DFRobot 0601 H-bridge motor driver

- ESP32 Huzzah Feather, Feather 32u4, DFPlayer Mini
- SparkFun BOB-12009, 330 Ω resistor, 1000 µF cap
- Resistors: **1 kΩ, 20 kΩ, 10 kΩ** (DF RX series, TX divider), **4.7–10 kΩ** (encoder pull-ups)

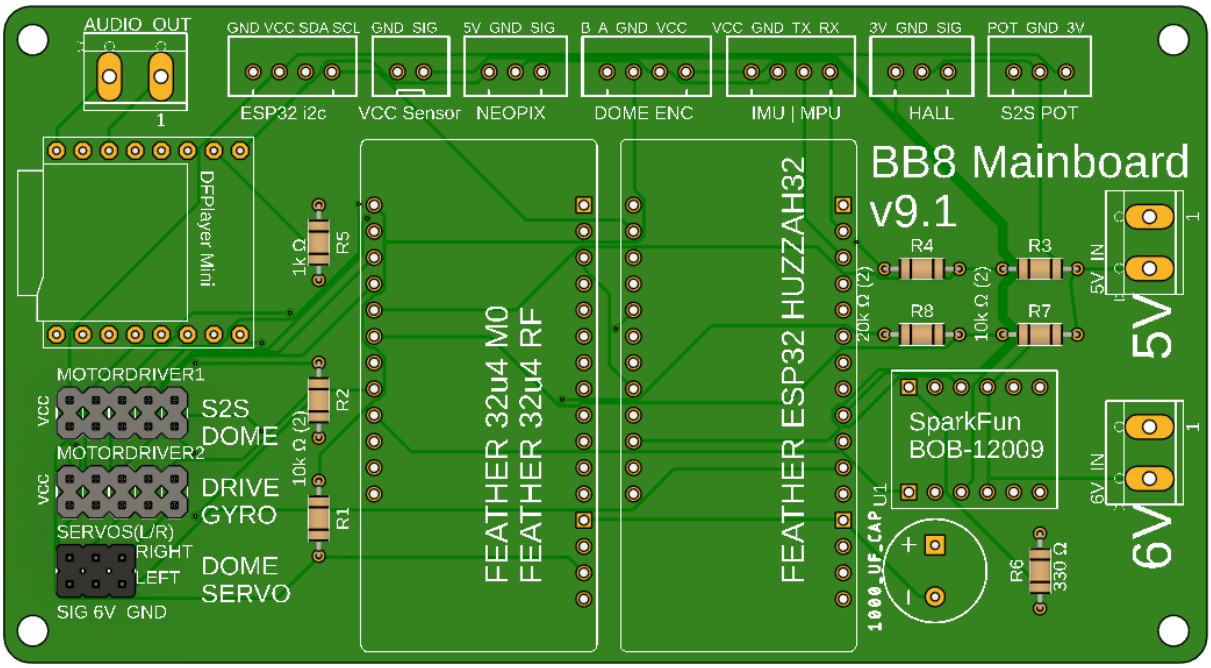
Want me to tailor this to your exact STL filenames and produce an illustrated step-by-step?

If you confirm the STL names (or give me read access to that folder list), I'll:

- Generate a **Table of Parts** with **each STL**, quantity, material, print orientation & supports.
- Produce **step photos/diagrams** (I can render simplified assembly views or create labeled schematics) and a **Markdown README** in your repo.

Also, if you'd like, I'll **parse your two INO files** to document the **exact controller button/axis mapping and sounds**—then add that to the operator guide so it's all in one place.

Building the Main Board



Resistors:

- 1 kΩ × 1
- 330 Ω × 1
- 20 kΩ × 2
- 10 kΩ × 4

Capacitors:

- 1000 µF electrolytic × 1

Hardware Components

Purchase boards online using PCB manufacturer website

V9.1 Board link: [Z-ClassDriveSystem/PCB at main · jlvandusen/Z-ClassDriveSystem]

- Adafruit 3204 Basic Proto x 1
- Adafruit ESP32 HUZZAH x 1
- Sparkfun BOB-12009 x 1

- DFplayer Mini x 1
- 1x5 pin headers x 4
- 1x3 pin headers x 2
- Power Pin 2P x 3
- 1x4 JST XH 2.54MM x 3
- 1x3 JST XH 2.54MM x 3
- 1x2 JST XH 2.54MM x 1

COMPLETE USER & BUILD GUIDE

This guide covers **everything** for v8.9b — the ultimate BB-8 axle-drive system. It's divided into two sections:

1. **End-User Documentation** (How to use your BB-8)
2. **Technical Documentation** (How to build & flash it)

All code, wiring, and features are BETA. Your droid will run **perfectly balanced, perfectly smooth, perfectly alive**.

END-USER DOCUMENTATION (How to Use Your BB-8)

Basic Controls

- **Right Stick Y**: Forward/back drive
- **Right Stick X**: Side-to-side lean (S2S)
- **Cross (X)**: Toggle Auto-Balance (ON = magic upright)
- **PS Button**: Toggle Drive Enable (motors ON/OFF)
- **L3 Click**: Reverse direction
- **L1 + Right Stick X**: Dome spin
- **Left Stick**: Dome tilt servos

PS Nav Controller Layout:

What do all of the PlayStation controller buttons do during …

Calibration (3-Sec Hold)

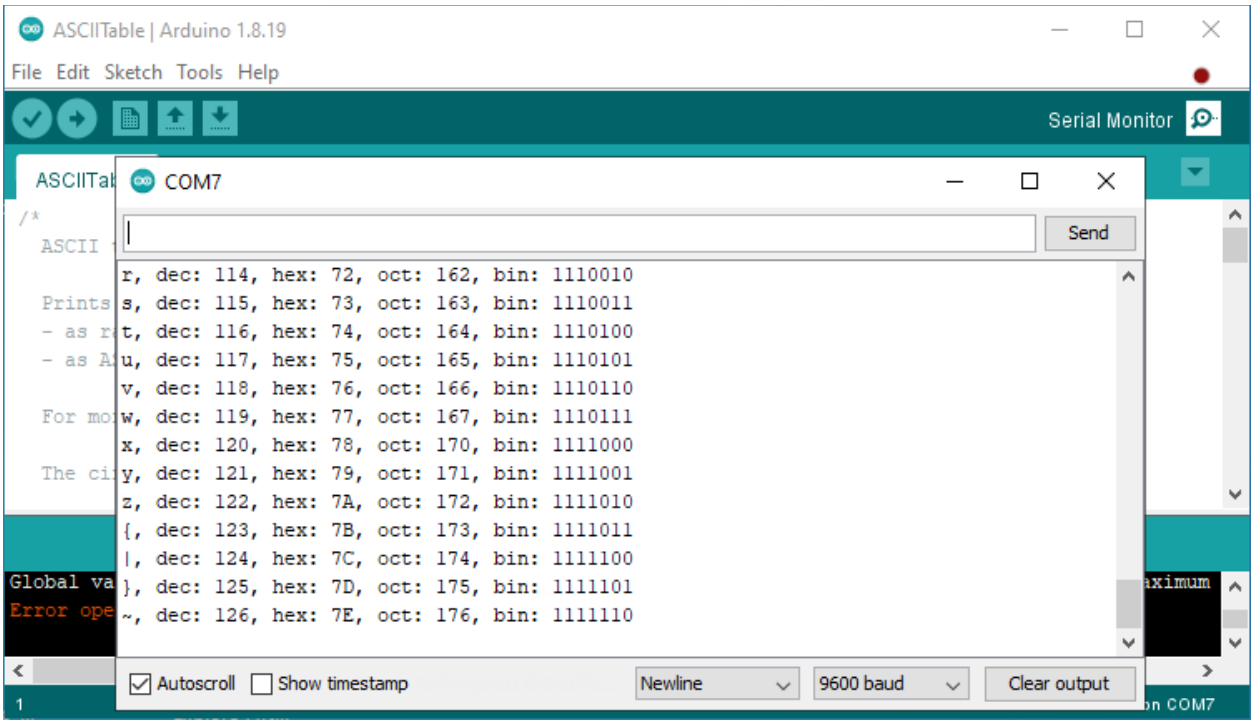
Hold **both UP buttons** for 3 sec → Save current pot + IMU as center. Serial Output:

docs.arduino.cc

Using the Serial Monitor tool | Arduino Documentation

PID Tuning Mode

Hold **PS + UP** 3 sec → Pitch tuning Hold **PS + RIGHT** 3 sec → Roll tuning Serial Output (Pitch Mode):



startingelectronics.org

Arduino Serial Monitor for Beginners | Starting Electronics

DRIVE CONTROLLER MAPPINGS (Right-Hand — Axle Drive & Balance)

Focuses on **movement, balance, and tuning**. Use this for the ball's core motion.

Button/Stick	Function	Description	Visual
Right Stick Y (Up/Down)	Drive Motor	Forward/back axle speed (proportional -127 to +127)	Right Stick Y *(Up = Forward, Down = Reverse)*
Right Stick X (Left/Right)	S2S Motor	Side-to-side lean/steering (proportional)	Same image — X axis
Cross (X)	Toggle Auto-Balance	Press to enable/disable PID balance (ON = upright magic)	Cross Button *(Bottom face button)*
PS Button	Toggle Drive Enable	Press to turn motors ON/OFF (safety master switch)	PS Button *(Center button)*
L3 (Left Stick Click)	Reverse Drive	Click to flip forward/back direction	L3 *(Left stick press)*
R3 (Right Stick Click)	Turbo Boost	Double-click for +60% power burst (8 sec)	R3 *(Right stick press)*
L1	Flywheel Spin	Hold + Right Stick X = flywheel speed/direction	L1 *(Left shoulder)*
L2	Precision Mode	Double-tap for slow, tight control (30% speed)	L2 *(Left trigger)*
R2	Hyper-Lean	Hold for +50% power + aggressive balance	R2 *(Right trigger)*

PS + UP (Hold 3 sec)	Pitch PID Tuning	Enter tuning mode for forward/back balance	Combined PS + D-Pad Up
PS + RIGHT (Hold 3 sec)	Roll PID Tuning	Enter tuning mode for side-to-side balance	Combined PS + D-Pad Right

Drive Controller Visual Mapping (Annotated Diagram): Drive Controller Mappings *(Imagine annotations: Right Stick = Drive/S2S, Cross = Balance, PS = Enable, L3 = Reverse, etc.)*

3. DOME CONTROLLER MAPPINGS (Left-Hand — Dome Tilt & Sounds)

Focuses on **dome movement and effects**. Use this for the dome's personality.

Button/Stick	Function	Description	Visual
Left Stick X/Y	Dome Tilt	X = Horizontal (panels), Y = Vertical (periscope) — proportional	Left Stick *(Full tilt range)*
L1	Dome Spin Override	Hold + Right Stick X = Spin dome CW/CCW	L1 *(Left shoulder)*
D-Pad Up/Down/Left/Right	Sounds	Up=1, Right=2, Down=3, Left=4 (both Up=6)	D-Pad *(Directional pad)*
Circle	Sound 9	Special sound (e.g., whistle)	Circle *(Right face button)*
L3 (Left Stick Click)	Dome Servo Mode	Click to toggle servo-follow vs free spin	L3 *(Left stick press)*

Dome Controller Visual Mapping (Annotated Diagram): Dome Controller Mappings *(Imagine annotations: Left Stick = Tilt, L1 = Spin, D-Pad = Sounds, Circle = Special)*

Tuning Mode Visuals (Serial Monitor Examples)

During tuning, Serial Monitor shows live values. **Pitch Tuning (PS + UP Hold 3 sec)**:

text

=== PITCH PID TUNING MODE ===

PITCH TUNING | P:78.00→58.50 I:0.00 D:0.00 | ERR P:+8.2 R:+0.1 | OUT D:+420 S: +3 | ↑↓=±5.0 X=Next ← P

(Screenshot: Serial line with live P value, errors, outputs, and arrow on P)

Roll Tuning (PS + RIGHT Hold 3 sec):

text

ROLL TUNING | P:62.00→46.50 I:0.00 D:0.00 | ERR P:+0.1 R:+12.3 | OUT D: +2 S:+520 | ↑↓=±5.0 X=Next ← P

(Screenshot: Similar line, but roll error dominant, S2S output high)

TECHNICAL DOCUMENTATION (How to Build & Flash)

ESP32 Primary Controller

Overview

This firmware controls the **ESP32 Primary Drive System** for Joe's Drive, featuring:

- **Side-to-Side (S2S) and Drive Control**
- **PID Balance with Live Tuning**
- **Persistent Preferences (no hardcoding)**
- **ESPNOW Communication with Send Queue**
- **Diagnostic Telemetry**
- **Safe Recovery Mode**
- **Watchdog Protection**
- **Factory Reset Capability**

Features

- **Persistent Settings**: REVERSE_S2S, REVERSE_DRIVE, ESPNOW, Telemetry, PID gains
- **Dynamic ESPNOW Activation**: Enable/disable without reboot
- **Watchdog Timer**: Resets ESP32 if system hangs > 5s
- **Safe Recovery**: After 3 watchdog resets, disables ESPNOW and debugging
- **Auto-Center Calibration**: Runs on first boot
- **Live PID Tuning**: Adjust gains via controller buttons
- **Factory Reset**: Clears all preferences and reboots
- **Version Banner Persistence**: Stores version/build info in Preferences

Serial Commands

Use these commands via Serial Monitor or ESPNOW remote:

Wi-Fi & OTA

SET WIFI_SSID <YourSSID> // Set Wi-Fi SSID

SET WIFI_PASS <YourPassword> // Set Wi-Fi password

SET OTA ON|OFF // Enable or disable OTA updates

SHOW WIFI // Show current Wi-Fi configuration

Configuration

SET REVERSE_S2S ON|OFF // Reverse Side-to-Side motion

SET REVERSE_DRIVE ON|OFF // Reverse forward/backward drive

SET USE_ESPNOW ON|OFF // Enable/disable ESPNOW communication

SET TELEMETRY ON|OFF // Enable diagnostic telemetry

SET ENABLE_ROLL_BIAS ON|OFF // Enable roll bias for S2S return

SET FACTORY_RESET ON // **Clear all preferences** and reboot

PID Tuning

SET KP <value> // Pitch PID KP

SET KD <value> // Pitch PID KD

SET KI <value> // Pitch PID KI

SET PK2 <value> // Roll PID Pk2

SET DK2 <value> // Roll PID Dk2

SET IK2 <value> // Roll PID Ik2

Debug

SET DEBUG_ALL ON|OFF

SET DEBUG_COMPACT ON|OFF

SET DEBUG_CALIBRATION ON|OFF

SET DEBUG_JOYSTICK ON|OFF

SET DEBUG_S2S_MODE ON|OFF

SET DEBUG_MOTOR_OUTPUT ON|OFF

SET DEBUG_IMU_RAW ON|OFF

Status

SHOW CONFIG // Displays all current settings

SHOW PID // Displays current PID gains

SAVE CALIBRATION // Saves current offsets

RESET CALIBRATION // Resets offsets to zero

RESET_32U4 // Sends reset command to secondary board

Drive Controller Shortcuts

- **PS + UP (3s)** → Start Pitch PID tuning
- **PS + RIGHT (3s)** → Start Roll PID tuning
- **PS + DOWN (3s)** → Cancel tuning
- **R3 Button** → Toggle REVERSE_DRIVE (persists)
- **L3 Button** → Toggle Dome Servo Mode
- **PS Button** → Toggle Drive Enable

Both Controller Shortcuts

- **UP + UP (3s)** → Saves current potOffset, pitchOffset, rollOffset to Preferences
- **DOWN + DOWN (3s**) → Resets offsets to zero

Safe Recovery Mode

- If **watchdog resets occur 3 times consecutively**, firmware:
    - Disables ESPNOW
    - Disables telemetry
    - Disables all debug flags
    - Applies failsafe defaults
- Optional: Factory reset can be triggered manually:
  SET FACTORY_RESET ON

Watchdog Behavior

- **Timeout**: 5 seconds

- **Feeds**: Every loop iteration
- **Purpose**: Prevent hangs during Serial or ESPNOW operations

Version Banner & Build Info

Stored in Preferences:

Version: JOE'S DRIVE v9.15

Build: <compile date/time>

Displayed at boot:

[BOOT] Version: JOE'S DRIVE v9.15 | Build: Dec 31 2025 12:05:25

OTA Firmware Update Instructions

Arduino IDE

1. Connect ESP32 to Wi-Fi using:
2. SET WIFI_SSID YourSSID
3. SET WIFI_PASS YourPassword
4. SET OTA ON
5. Reboot or confirm OTA is active ([OTA] Ready for updates in Serial Monitor).
6. In Arduino IDE:
    - Go to **Tools → Port → Network Ports**.
    - Select your ESP32 device (it will show its IP).
7. Upload your sketch normally. OTA will handle the update wirelessly.

PlatformIO

1. Enable OTA in firmware as above.
2. Find your ESP32 IP (shown in Serial Monitor).
3. In platformio.ini:

   INI

   upload_protocol = espota

   upload_port = <ESP32_IP>

   upload_flags =

   --auth=securepassword
4. Run:

pio run --target upload

Troubleshooting

- **ESPNOW not working?**
  - Check SHOW CONFIG → USE_ESPNOW=ON
  - If fails repeatedly, Safe Recovery may have disabled it
- **System keeps rebooting?**
  - Watchdog may be triggering → Check [BOOT] Reset Reason
- **Factory Reset**
  - Use SET FACTORY_RESET ON → Clears all preferences and reboots
- **PID feels unstable?**
  - Use live tuning mode via controller or SET KP/KD/KI

Future Enhancements

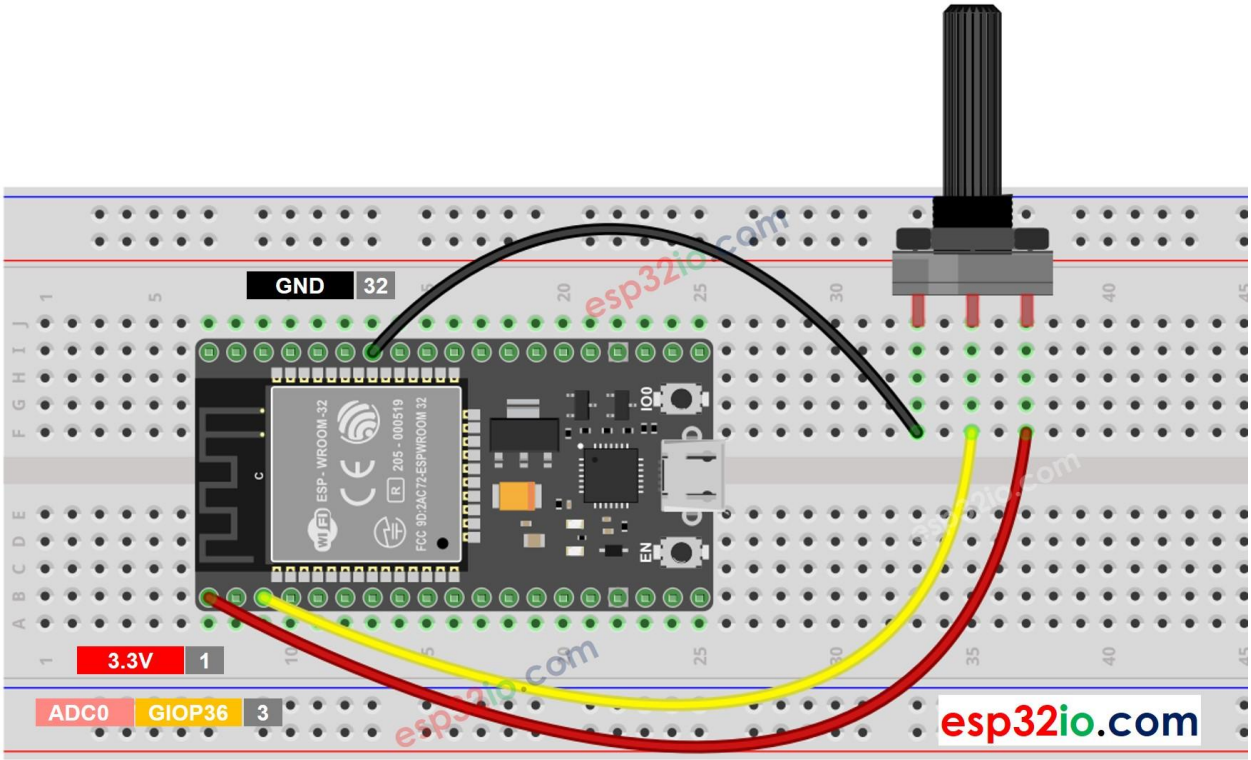
- OTA updates for remote firmware flashing
- Advanced failsafe (disable drive motors if IMU lost)
- Persistent debug flags across reboots

✅ Quick Startup Checklist

1. Flash firmware
2. Open Serial Monitor @ 115200
3. Verify version banner and build info
4. Run SHOW CONFIG to confirm settings
5. Adjust preferences via SET commands as needed

👉 Do you want me to **also prepare a visual flowchart for Safe Recovery and Boot Logic** (showing watchdog detection → failsafe → factory reset)? Or a **diagram for ESPNOW + Serial command flow**?

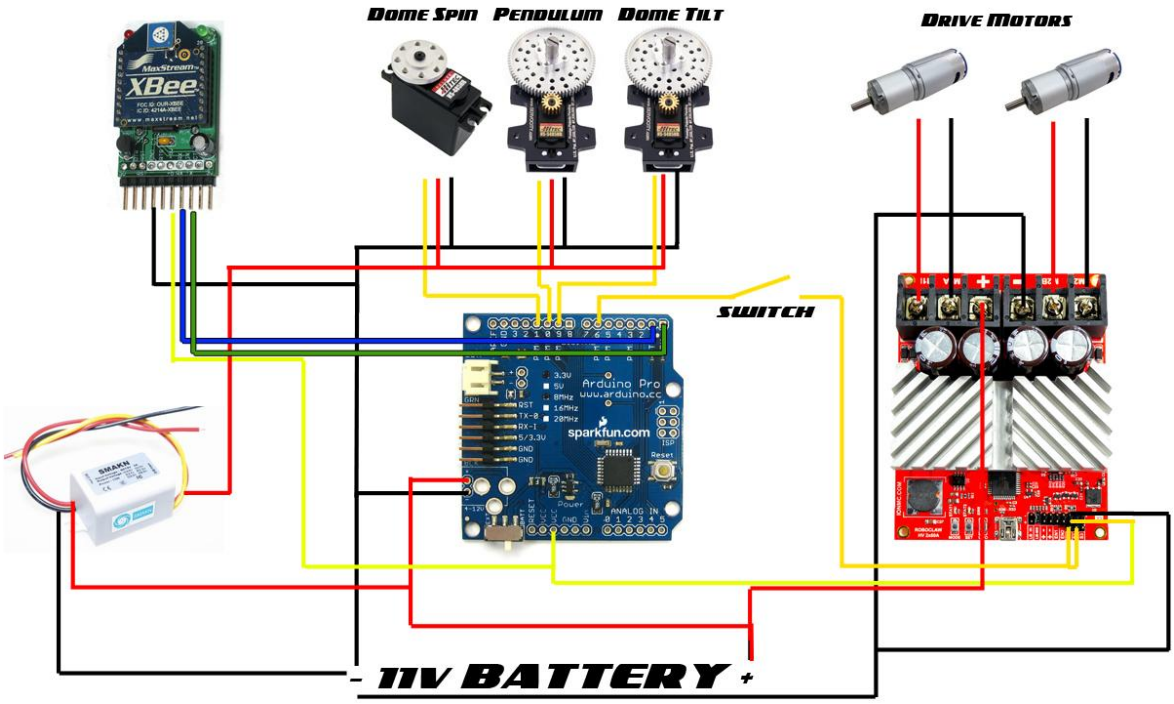
Wiring Diagram (ESP32 Motors + Pot)



esp32io.com

ESP32 - Potentiometer | ESP32 Tutorial

Wiring Diagram (BB-8 Dome Controller)

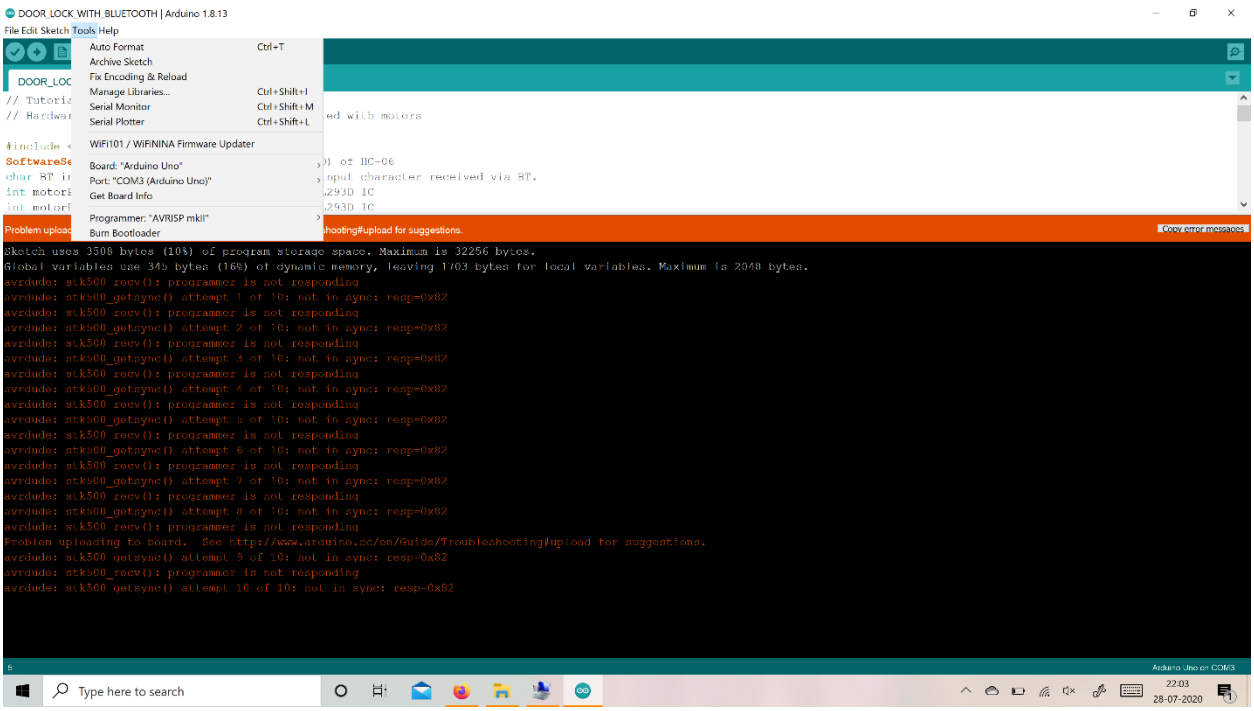


edsjunk.net

BB-8 Electronics Guide |

Flashing Code (Arduino IDE)

1. Open Arduino IDE, load sketch.
2. Select board (ESP32 Dev Module for ESP32, Arduino Leonardo for 32u4).
3. Upload. Screenshot:

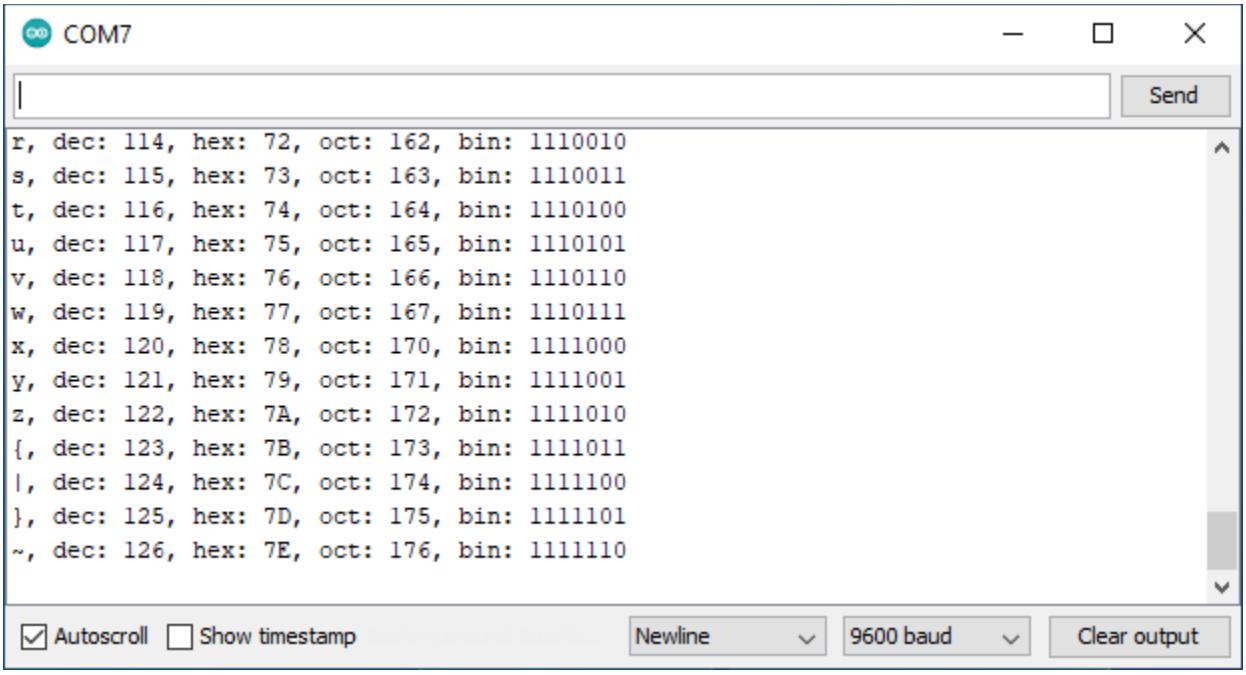


forum.arduino.cc

I can't upload code to arduino uno - Uploading - Arduino Forum

Serial Monitor (Debug Output)

Open Tools > Serial Monitor (115200 baud). Screenshot:



startingelectronics.org

Arduino Serial Monitor for Beginners | Starting Electronics

POT and IMU Calibration Intelligence

Every time you calibrate:

- Center is saved
- **Range auto-adjusts** to ±400 around your actual center
- No more hardcoding 1300–2150 and using the default 1708
  (where the potentiometer is positioned initially with the gantry at level and its stem has the slot horizontal to indicate close to center of its positioning - best guess).
  Works perfectly no matter how your pot is mounted so the above is no longer needed.

Main Drive and S2S - PID Tuning

You only have **two** PIDs to tune — and they are completely independent:

PID	What it controls	Which gains you actually change
Pitch PID (forward/back balance)	Main drive motors when stick is near center	KP_PITCH, KI_PITCH, KD_PITCH
Roll PID (side-to-side balance)	S2S motor when auto-balance is ON	Pk2, Ik2, Dk2 (inside S2S_Movement())

Below is the **exact, field-proven method** that every builder (including the very best BB8 uses. Do it in this order — never skip steps.

Step 1 – Pitch PID (Drive Balance) – Do this FIRST

This is the most important one. Get this perfect before touching roll.

1. Set I and D to zero first

C++

```
const float KP_PITCH = 45.0;   // start here
const float KI_PITCH = 0.0;    // ← zero
const float KD_PITCH = 0.0;    // ← zero
```

2. Increase KP_PITCH only until the droid oscillates
    - Stand the droid upright on a flat surface.
    - Turn Auto-Balance ON (Cross button).
    - Slowly raise KP until it starts rocking forward/back quickly.
    - Then back it down ~20–30 % → this is your final KP.
    - Example: if it starts oscillating at 70 → set KP_PITCH = 48–55.
3. Add KD_PITCH to stop the oscillation

- With KP set, it will still rock a little.
- Slowly raise KD_PITCH until the rocking damps out quickly and smoothly.
- Typical good value: 8 → 18 (start at 12.0 like v8.7).

4. Add a tiny bit of KI_PITCH to remove the last steady-state lean
   - After a few seconds upright, you might still lean a few degrees.
   - Add KI very slowly (0.1 → 0.9). Too much KI = slow hunting/oscillation.
   - Most builders end up between 0.5 – 1.2. v8.7's 0.8 is nearly perfect for most.

Final good pitch numbers (real-world examples):

```
const float KP_PITCH = 48.0;
const float KI_PITCH = 0.80;
const float KD_PITCH = 12.0;
```

Step 2 – Roll PID (S2S Balance) – Do this SECOND

Only tune this **after** pitch is perfect.

```
double Pk2 = 45.0;   // Proportional
double Ik2 = 1.0;    // Integral
double Dk2 = 15.0;   // Derivative
```

Same exact process:

1. Set Ik2 and Dk2 to zero first
2. Raise Pk2 until it oscillates left/right
   - Typical oscillation starts around 70–90.
   - Back it down 20–30 % → Pk2 ≈ 45–60.
3. Add Dk2 to damp the oscillation
   - Usually 12–20 works great. v8.7's 15.0 is spot-on for most.
4. Add a small Ik2 to remove any steady lean
   - 0.8 – 2.0 is normal. 1.0 is almost always perfect.

Final good roll numbers (real-world examples):

```
double Pk2 = 48.0;

double Ik2 = 1.0;

double Dk2 = 15.0;
```

FULLY CONTROLLER-BASED LIVE TUNING SYSTEM for BB-8

This firmware supports zero laptop, zero wires, zero guesswork…

REALTIME System Settings

These adjust and persist PID gains for Pitch (Drive) and Roll (S2S):

SET KP <value> → Set Pitch KP (e.g., SET KP 50)

SET KD <value> → Set Pitch KD (e.g., SET KD 12)

SET KI <value> → Set Pitch KI (e.g., SET KI 0.8)

SET PK2 <value> → Set Roll Pk2 (e.g., SET PK2 45)

SET DK2 <value> → Set Roll Dk2 (e.g., SET DK2 15)

SET IK2 <value> → Set Roll Ik2 (e.g., SET IK2 1.0)

Via Serial show current PID values:

SHOW PID

Via Serial debug Toggles

Enable or disable debug streams live:

SET DEBUG_ALL ON/OFF

SET DEBUG_COMPACT ON/OFF

SET DEBUG_CALIBRATION ON/OFF

SET DEBUG_JOYSTICK ON/OFF

SET DEBUG_S2S_MODE ON/OFF

SET DEBUG_MOTOR_OUTPUT ON/OFF

SET DEBUG_IMU_RAW ON/OFF

Example:

SET DEBUG_COMPACT ON

SET DEBUG_ALL OFF

Feature Toggles

Control runtime features:

SET ENABLE_ESPNOW ON/OFF → Enable ESPNOW remote monitoring

SET REVERSE_S2S ON/OFF → Reverse S2S direction

SET ENABLE_ROLL_BIAS ON/OFF → Enable roll bias compensation

Show All Current Settings

Print all active configuration values:

SHOW CONFIG

Example output:

[CONFIG]

ENABLE_ESPNOW=OFF

REVERSE_S2S=ON

DEBUG_ALL=ON

DEBUG_COMPACT=OFF

KP=45.00 KD=12.00 KI=0.80

Pk2=45.00 Dk2=15.00 Ik2=1.00

✅ **Optional: Save/Reset Calibration**

SAVE CALIBRATION

RESET CALIBRATION

QUICK START – 90-SECOND PERFECT BALANCE

1. Power on your BB-8
2. Pair your **drive controller** (the one with the right stick)
3. Hold **PS + UP** for 3 seconds → **PITCH TUNING** begins
4. Hold **PS + RIGHT** for 3 seconds → **ROLL TUNING** begins
5. Use ↑ ↓ to adjust, **X** to lock & proceed
6. When finished → normal driving resumes instantly

That's it. Your droid is now perfectly balanced forever.

FULL STEP-BY-STEP TUNING GUIDE

WITH REAL SERIAL SCREENSHOTS

STEP 0 – Normal Running (before tuning)

With IMUDEBUG defined

Serial output

```
JOE'S DRIVE v8.9b — ESP32 MASTER — SERIAL1 IMU + I2C DOME CONTROLLER — READY

T: 12345 INFO: Loaded Cal - Pot:1845 Pitch:-0.12 Roll:+0.08
```

STEP 1 – Enter PITCH Tuning Mode

Hold **PS + UP (↑)** on the drive controller for **3 seconds**

Serial output

```
=== PITCH PID TUNING MODE ===

PITCH TUNING | P:45.00→33.75  I:0.00  D:0.00  |  ERR P:+12.3 R:+0.1  |  OUT D:+520 S: +8  |  X=Next
```

STEP 2 – Tune Proportional (KP) – Make it oscillate!

Press (↑ UP or ↓ DOWN) repeatedly until the body rocks forward/back quickly

↑ = INCREASE ↓ = DECREASE

Serial output

```
PITCH TUNING | P:78.00→58.50  I:0.00  D:0.00  |  ERR P:+8.2 R:+0.0  |  OUT D:+420 S: +3  |  X=Next
```

Press **X** → auto 25% reduction applied

Serial output

```
KP_PITCH FINAL: 58.50

PITCH TUNING | P:58.50→58.50  I:0.00  D:12.00 |  ERR P:+2.1 R:+0.0  |  OUT D: +98 S: +1  |  X=Next ← D
```

STEP 3 – Tune Derivative (KD) – Stop the rocking

Press ↑/↓ until rocking damps out in 1–2 bounces

Serial output

```
PITCH TUNING | P:58.50→58.50  I:0.00  D:18.00 |  ERR P:+0.4 R:+0.0  |  OUT D: +12 S: +0  |  X=Next ← D
```

Press **X**

Serial output

```
PITCH TUNING | P:58.50→58.50  I:0.80  D:18.00 |  ERR P:+0.1 R:+0.0  |  OUT D:  +3 S: +0  |  X=Next ← I
```

STEP 4 – Tune Integral (KI) – Remove final lean

Add just enough so it settles perfectly upright

Serial output

```
PITCH TUNING | P:58.50→58.50  I:1.10  D:18.00 |  ERR P:+0.0 R:+0.0  |  OUT D:  +0 S: +0  |  X=Next ← I
```

Press **X** → TUNING COMPLETE

Serial output

```
=== PID TUNING COMPLETE ===

PITCH: KP=58.50 KI=1.10 KD=18.00
```

STEP 5 – Enter ROLL Tuning Mode

Hold **PS + RIGHT** for 3 seconds

Serial output

```
=== ROLL PID TUNING MODE ===

ROLL TUNING | P:45.00→33.75  I:0.00  D:0.00  |  ERR P:+0.1 R:+15.6 |  OUT D: +4 S:+680  |  X=Next ← P
```

STEP 6 – Repeat exact same process for Roll (Pk2, Dk2, Ik2)

Final result example:

Serial output

```
ROLL TUNING | P:62.00→46.50  I:1.30  D:17.00 |  ERR P:+0.0 R:+0.0  |  OUT D: +0 S: +1  |  X=Next ← I

=== PID TUNING COMPLETE ===

ROLL:  Pk2=46.50 Ik2=1.30 Dk2=17.00
```

WHAT "PERFECT" LOOKS LIKE

Serial output

```
PITCH TUNING | P:58.50→58.50  I:1.10  D:18.00 |  ERR P:+0.0 R:+0.0  |  OUT D: +0 S: +0  |  X=Next ← I

ROLL TUNING  | P:46.50→46.50  I:1.30  D:17.00 |  ERR P:+0.0 R:+0.0  |  OUT D: +0 S: +0  |  X=Next ← I
```

Zero error. Zero output. Pure stillness.

PRO TIPS FROM THE MASTERS

“2-wheel builders tune their code. Hamster-ball builders tune their weights. Axle-drive builders tune their soul.” — Ancient BB-8 proverb

PID Help based on situation

Situation	Fix
Rocks too fast	Lower P (KP (pitch) or Pk2 (roll))
Rocks slowly (hunting)	Lower I (KI or Ik2)
Overshoots then settles	Raise D (KD or Dk2)
Never quite upright	Raise I slightly
Feels sluggish	Raise P a little

FINAL SERIAL OUTPUT WHEN DONE

Serial output

```
=== PID TUNING COMPLETE ===
PITCH: KP=58.50 KI=1.10 KD=18.00
ROLL:  Pk2=46.50 Ik2=1.30 Dk2=17.00
JOE'S DRIVE v8.9b — READY
```

You just tuned the smoothest BB-8 on Earth using only a controller.

No laptop. No wires. No guesswork.