

MOTION CORE

USB Motion Controller for Differential-Drive Mobile Robots

Getting Started — ROS 2

APPLIES TO: Motion Core firmware 1.9.6 (CONFIG 23) · ORCP v1.1, conformance Level 2 · ROS 2 Jazzy

AUDIENCE: Integrators bringing a new two-wheeled robot platform up under ROS 2

YOU WILL FINISH WITH: A configured Motion Core publishing odometry and TF, driven by /cmd_vel from keyboard teleop.

COMPANION GUIDE

Motion Core — Getting Started (Python) covers the same board setup and then drives the platform directly from Python. If you are not committed to ROS 2, start there — it is the shorter path to a moving robot.

READ THIS FIRST

The ROS 2 driver deliberately does NOT expose the board's configuration interface. Sections 5–7 configure the board — using the web configurator or a serial connection — before ROS 2 enters the picture. You cannot skip them, and you cannot do them from ROS 2.

CONTENTS

1. What you need	3
2. Wiring it up	4
3. First contact	9
4. The safety model — read before you move anything	11
5. Configuring for your platform	13
6. Tuning the velocity loop	18
7. Save it and prove it.....	19
8. Installing the ROS 2 driver.....	20
9. The driver interface	22
10. Driving the platform	24
11. Fault reference.....	26
12. Troubleshooting	27
Where to go next	29
Appendix A — Trying it without hardware	30

1. WHAT YOU NEED

Hardware

ITEM	NOTES
Motion Core controller	Ships pre-flashed and factory-calibrated
Battery pack, 9–20V DC	Factory thresholds assume 3S Li-ion/LiPo; reconfigurable for other chemistries
Two brushed DC gearmotors with quadrature encoders	Up to 4A continuous per side. Check whether your encoders need 5V or 3.3V — see §2
Normally-closed e-stop switch	Mandatory — or a jumper link fitted to J3. See §2
Normally-closed charger-detect switch	Mandatory — or a jumper link fitted to J2. See §2
USB-C cable	USB 2.0 rated, data-capable. Does not power the board — see §2
A chassis	Two driven wheels plus a castor or equivalent
XT60 battery lead	To match the board's power input

Host machine

- **ROS 2 Humble or later, on Ubuntu 22.04 LTS or later.** Jazzy on Ubuntu 24.04 is the configuration the driver is built and tested against
- Python 3.9 or newer
- macOS users: see the Docker path in §8, and read the USB warning there **before** buying into that approach
- A Chromium-based browser (Chrome or Edge) to run the web configurator — see §5

Bench space

Section 6 drives the robot about a metre under power. Keep a clear, flat area of roughly 3 m × 3 m, and keep the e-stop within reach.

2. WIRING IT UP

All connectors are labelled on the board silkscreen.

Connectors

REF	SILKSCREEN	FUNCTION	CONNECTOR TYPE
J1	VBAT 9V-20V DC	Battery / DC supply input	2-pin XT60
J2	CHARGER	Charger-detect interlock — mandatory	2-pin JST XH
J3	ESTOP	E-stop interlock — mandatory	2-pin JST XH
J4	—	Regulated 5V auxiliary output, 6A	2-pin Molex Mini-Fit Jr.
J5	CAN	CAN bus — factory-fit variant only	4-pin JST XH
J6	EXPANSION	Reserved — no function at this release	6-pin JST XH
J7	USB	USB-C host interface	USB-C receptacle
J8	ENCODER LEFT	Left encoder input	5-pin JST XH
J9	MOTOR LEFT	Left motor output	2-pin Molex Mini-Fit Jr.
J10	ENCODER RIGHT	Right encoder input	5-pin JST XH
J11	MOTOR RIGHT	Right motor output	2-pin Molex Mini-Fit Jr.
JP1	ENC VOLTAGE	Encoder supply select, 5V / 3.3V	3-pin header + jumper
JP2	120Ω	CAN bus termination	Header + jumper
CN1	PGM	SWD programming — factory / service use	6-way header
—	RESET	Reset pushbutton	—

Mating connectors, jumper links and reference cabling are supplied with the Rapid Integration Kit variant (FLR-MC-1-2M12-4-RIK) if you would rather not source them yourself.

NOTE

Reading the pinouts below. Signal names are given by **pin number**, which is authoritative. On the multi-way connectors the silkscreen abbreviations are printed in the opposite order to the pin numbering, so identify pin 1 from the board marking rather than assuming left-to-right.

Step 1 — the two mandatory interlocks (J2 and J3)

CAUTION

Read this before anything else. A board with J2 or J3 left open will not drive its motors, no matter what your software does — and nothing on the software side will look broken.

Motion Core has two independent hardware interlocks, both wired for normally-closed (NC) contacts, and both built into the drive-enable logic at hardware level. Neither can be overridden by the host.

	J3 — E-STOP	J2 — CHARGER DETECT
Intended switch	NC emergency-stop pushbutton	NC switch on the charging connector
Loop closed	Motors can be enabled	Motors can be enabled

	J3 — E-STOP	J2 — CHARGER DETECT
Loop open	Relay de-energises — motor supply physically disconnected from the H-bridges	Drive-enable inhibited at hardware level
Reported to host	estop=1, fault=ESTOP	fault=CHARGER

The e-stop is a de-energise-to-trip relay chain: opening the loop physically disconnects the H-bridge outputs from the motor terminals. It works even if the processor has crashed. The charger interlock is a parallel hardware path that stops the robot driving away while it is plugged in to charge.

WARNING

If you are not fitting one of these switches, you must fit a jumper link. Both loops are mandatory for the drive to operate:

- No e-stop fitted → **link J3 pin 1 to pin 2**
- No charger switch fitted → **link J2 pin 1 to pin 2**

Leaving either connector open holds the motor outputs disabled at hardware level, regardless of host commands. This is the most common reason a new board appears to do nothing.

Both connectors share the same pinout:

PIN	SIGNAL	NOTES
1	GND	Ground
2	ESTOP / CHGDET	Dry-contact loop input, internal ~40 kΩ pull-up to 3.3V

These are **dry-contact inputs** — connect a switch, not a voltage. Maximum applied voltage is 3.6 V.

On a robot people will stand near, fit a real e-stop. The link is for bench bring-up, and for machines where the e-stop lives elsewhere in a larger safety chain.

Recovery from either interlock needs the loop closed **and** an explicit ENABLE ON from the host. Closing the loop alone does not restart the robot.

Step 2 — set the encoder supply voltage (JP1)**WARNING**

Do this before you plug an encoder in. JP1 selects whether the encoder supply pin carries **5V or 3.3V**. Applying 5V to a 3.3V-only encoder can destroy it.

PIN	SIGNAL
1	5V
2	ENC_VCC — routed to both encoder connectors
3	3.3V

Fit the jumper between pin 2 and whichever supply your encoders need.

A jumper must be fitted for the encoders to work at all — with JP1 open, ENC_VCC is unpowered. JP1 feeds both J8 and J10, so you cannot mix a 5V encoder on one side with a 3.3V encoder on the other.

Step 3 — wire the encoders (J8, J10)

J8 pairs with Motor Left, J10 with Motor Right. Both share this pinout:

PIN	SIGNAL	SILKSCREEN	NOTES
1	GND	GND	Ground
2	ENC_VCC	VIN	Encoder supply — 5V or 3.3V per JP1
3	ENC_A	A	Quadrature channel A
4	ENC_B	B	Quadrature channel B
5	Shield	SHIELD	Optional cable-shield ground

Points worth knowing:

- **A and B channels only.** There is no index/Z input; index pulses are not used.
- Inputs are level-shifted and accept **3.3V or 5V** logic, and **open-collector or push-pull** outputs.
- Maximum input frequency is **100 kHz per channel** (400 kHz decoded at ×4).
- If your cable has no shield, leave pin 5 unconnected. Do not bond the shield at both ends.

A/B order determines the reported direction of travel. Getting it backwards is normal and expected — **§5 step 2 fixes it in software, so do not rewire to correct direction.**

Step 4 — wire the motors (J9, J11)

J9 is Motor Left, J11 is Motor Right.

PIN	SIGNAL	NOTES
1	M+	Motor output, positive — forward direction
2	M–	Motor output, negative

Rated **4A continuous per channel** across the full 0–40 °C ambient range, and **8A peak** for short acceleration events of a few seconds.

As with the encoders, polarity sets which way the wheel turns — **do not rewire to fix direction**, §5 step 2 corrects it in configuration.

Step 5 — power (J1) and the auxiliary output (J4)

J1 — battery input. 9–20V DC, XT60, polarity marked with + on the PCB.

WARNING

USB-C is data-only. The board draws no operating power from USB VBUS and supplies none to the host; USB Power Delivery is not implemented in either direction. **Motion Core will not power up or enumerate from USB alone — it must be supplied from J1.** If the board does not appear as a serial port, check the battery before you check the cable.

J4 — regulated 5V auxiliary output. For sensors, an SBC, or other 5V loads.

PIN	SIGNAL	NOTES
1	+5V	Regulated 5V, 6A maximum
2	GND	Ground

Output holds 4.8–5.2V across the full load and temperature range and is short-circuit protected. Its current is monitored on-board and reported to the host, so an overloaded rail or a failing sensor is visible in software. Warning thresholds are configurable (`aux5v.i_warn`, `aux5v.v_warn`) and raise `! WARN AUX5V`.

The 5V pins on J5 and J6 are limited to **500 mA each** and are not a substitute for J4.

Step 6 — power up

1. Fit JP1 for your encoder voltage.
2. Connect motors (J9/J11) and encoders (J8/J10).
3. Fit the e-stop (J3) and charger-detect (J2) loops — switch or link.
4. Connect the battery (J1).
5. Connect USB (J7).

Allow **1.5–2.5 s** from applying battery power to the board appearing as a serial port on the host.

Safe-start behaviour

Motor outputs are held disabled at power-up, regardless of anything the host is doing. They enable only after the host has connected and issued an explicit command. A board that resets mid-drive comes back stopped — it will not resume motion on its own.

Status LEDs

LED	MEANING
Red, blinking ~1 Hz	Application firmware running normally
Green, blinking ~5 Hz	Board is sitting in the bootloader

If you see the green bootloader blink on a board you have not deliberately put into update mode, see §12.

Wiring practice

These are not style preferences. **CE/UKCA compliance is conditional on following them**, and departures may require system-level EMC mitigation.

- **Twist the motor pairs** (M+/M-) to reduce radiated EMI. Near sensitive electronics, use shielded motor cable with the shield bonded to chassis ground.
- **Use twisted-pair shielded encoder cable**, particularly for runs longer than 0.5 m. Encoder noise shows up as velocity jitter and phantom stall faults.
- **Route the e-stop loop away from motor wiring**, and through any external safety circuits before it reaches J3.
- **Use a USB 2.0-rated cable**. Beyond 1 m use a cable with active or passive signal conditioning; 3 m is the specification limit.
- Size motor conductors for your actual motor current with derating headroom — gauge selection is the integrator's responsibility.
- Motion Core uses a **single ground reference**; battery, signal and shield grounds are joined on-board. In multi-controller installations, avoid creating ground loops through secondary chassis connections.

Power budget

CONTRIBUTOR	DRAW
Motors	Up to $2 \times 4A \times V_{BAT}$ — roughly 96W at 12V
5V auxiliary	Up to $6A \times 5V = 30W$, drawn from V_{BAT} via the on-board regulator
Host SBC, if powered from J4	Typically 5–15W

Mounting and cooling

The board is designed for **natural convection with no enclosure restriction**. Mount it on **5 mm standoffs with both sides clear for airflow**. Installing it in a sealed enclosure, or at elevated ambient temperature, may require derating the motor current below 4A per channel.

Board outline is 115.0 × 90.0 mm, 1.6 mm thick, with four mounting holes on a 105.0 × 80.0 mm pitch. Maximum component height is 16 mm above the top surface.

CAN (JP2)

CAN is a **factory-fit variant** (FLR-MC-1-2M12-4-C) and is **mutually exclusive with USB** — on CAN-variant boards the firmware brings up the CAN peripheral in place of the USB CDC interface. Standard boards leave the transceiver, connector and termination unpopulated.

If you have a CAN board sitting at one **end** of the bus, fit JP2 for the 120 Ω termination; omit it mid-bus. This guide uses USB throughout.

3. FIRST CONTACT

Before ROS 2, confirm the board talks. Find its serial port.

NOTE

The board must be powered from the battery (J1). USB-C is data-only — it neither powers Motion Core nor draws from it. Allow **1.5–2.5 s** after applying battery power before the serial port appears.

Linux

```
ls /dev/ttyACM*
```

If you get a permission error, add yourself to the dialout group and log back in:

```
sudo usermod -aG dialout $USER
```

This matters more under ROS 2 than elsewhere — a driver node that cannot open the port fails at startup with an unhelpful traceback.

macOS

```
ls /dev/cu.usbmodem*
```

Use the `/dev/cu.*` entry, not `/dev/tty.*`.

Say hello

The board speaks line-oriented ASCII at **115200 baud, 8N1**. Any serial terminal works. Type PING:

```
PING
OK PONG
```

Then:

```
INFO
OK INFO fw=1.9.6 bl=1.4.0 hw=MC1 proto=ORCP/1.1 level=2 uuid=...
      build=... serial=MC1-2626-00001 vbus_v=5.02 vbus_present=1 vdda=3.31
```

- `fw` — firmware version
- `proto` — protocol version. **This is the number that matters for software compatibility**, not `fw`
- `level` — conformance level; Level 2 is what enables configuration commands
- `hw=MC1` — the board's internal hardware identifier for Motion Core

Two notes on the link:

- **The baud rate is nominal.** This is a USB CDC device, so baud, parity and flow control do not affect the physical link and are not enforced by the board. 115200 8N1 is recommended only because most host libraries require *some* valid setting before they will open a port.
- **Command lines are limited to 128 characters** including the newline; longer lines are rejected with `ERR TOO_LONG`. Commands are terminated with `\n`, and a preceding `\r` is accepted. Responses come back terminated with `\r\n`.

Host OS support: Linux (kernel 3.0 or later, no driver installation needed), macOS 10.15 or later, and Windows 10 or later.

Keep this terminal handy — sections 5 to 7 use it. If PING returns nothing, jump to §12.

4. THE SAFETY MODEL — READ BEFORE YOU MOVE ANYTHING

Motion Core will not let you drive a robot at full power by accident. Three mechanisms are worth understanding before you send a motion command.

Presets

The board boots in **SLOW** every time. It never remembers that you were in NORMAL.

	SLOW	NORMAL
Duty limit	30 %	90 %
Motors enabled	Automatically	Only after ENABLE ON
Stale-command timeout	Off by default	250 ms
Heartbeat required	No	Yes, every 500 ms

SLOW is the bring-up default: you can drive immediately, you cannot go fast, and nothing times out while you think. NORMAL is the operating mode and demands that the host prove it is alive.

The ROS 2 driver handles the heartbeat for you — setting its preset parameter to NORMAL automatically starts a 100 ms heartbeat, and setting it back to SLOW stops it. You never send heartbeats by hand. There is no "FULL" preset.

The hardware interlocks

Two of them — e-stop (J3) and charger-detect (J2) — and both sit below software. If either loop is open, the motor outputs are disabled in hardware, whatever the host asks for; the e-stop physically disconnects the motor supply via a de-energised relay. See §2 — if you have not fitted switches, you need jumper links on both.

Test the e-stop before every session — §5 step 1.

Safe-start

Motor outputs are held disabled at power-up regardless of host state, and enable only after the host connects and issues an explicit command. A board that resets mid-drive comes back stopped — it does not resume motion on its own.

Faults latch

This is the single most important behaviour to internalise:

A fault stays latched until the board is explicitly re-enabled.

Motion Core does not silently recover. If the battery dips, or a wheel jams, or the e-stop is pressed, the board stops, records *why*, and stays stopped. The condition clearing is not enough — you must acknowledge it.

In ROS 2 that acknowledgement is the `~/enable` service:

```
ros2 service call /orcp_driver/enable std_srvs/srv/SetBool "{data: true}"
```

Full fault list and recovery steps in §11.

5. CONFIGURING FOR YOUR PLATFORM

READ THIS FIRST

This section is done outside ROS 2 — with the web configurator, or over the serial terminal from §3. The driver deliberately does not expose configuration, and it holds the serial port exclusively while running, so the node must be **stopped** while you do this.

Motion Core ships knowing nothing about your robot. This section teaches it.

Configuration splits cleanly in two, and the distinction matters:

- **Robot configuration** — geometry, motor limits, battery chemistry. Yours to set. This section.
- **Factory calibration** — current-sense scaling and offsets, the battery divider ratio, the MCU temperature coefficients. Measured per board at manufacture. **Do not change these.** They are `current.scale`, `current.offset_left`, `current.offset_right`, `batt.divider_ratio`, `mcu.temp_v25`, `mcu.temp_slope` and `aux5v.shunt_ohms`.

The `DEFAULTS` command respects that split — it resets your robot configuration but **preserves the factory calibration group**. It is safe to use when a configuration experiment goes wrong.

WARNING

SAVE before you power-cycle. Configuration changes take effect immediately but live only in RAM. A power cycle before `SAVE` discards everything you configure in this section — the single most common bring-up mistake. Full save/load workflow in §7.

Two ways to configure — the web configurator is preferred

The **Motion Core Web Configurator** is the recommended way to work through this section. It runs in a browser, connects to the board over USB, and gives you:

- the full configuration surface as a grouped, validated form — no key names to type and no `SET` syntax to get wrong
- live status and telemetry while you work
- battery chemistry applied as a single profile selection rather than four separate values
- fault codes decoded into plain language

Use it unless you specifically need scripted or headless configuration.

NOTE

Browser requirement. The configurator uses the **Web Serial API**, which is supported in **Chrome, Edge and other Chromium-based browsers**. **Firefox and Safari do not implement Web Serial** and cannot connect to the board.

How to connect

The configurator is at <https://firstlayerrobotics.com/motion-core/configurator>.

Plug your Motion Core into a USB port, then click the connect button on the page. Your browser will show a picker; choose the entry labelled **FLR Motion Core** (also shown alongside a `usbmodemXXXXXX` or `usbserialXXXXXX` device path).

Only one program can hold the serial port at a time. Close the configurator before running the tuning tool in §6, or any of the software in §8 onward.

Two things the configurator does *not* cover: **velocity-loop tuning** (§6, which uses a host-side tool) and **firmware updates**.

The rest of this section gives the equivalent serial commands. They are the authoritative reference, they are what the configurator sends underneath, and they are what you need for scripted setup — so they are worth reading even if you never type one.

Command syntax

Read a value, write a value, write all values to flash:

```
GET kin.track_width
SET kin.track_width=0.28
SAVE
```

NOTE

SET takes key=value with no spaces around the =. SET kin.track_width 0.28 is rejected. This trips up nearly everyone once.

Changes take effect immediately but live only in RAM until you SAVE. That is deliberate: you can experiment freely, and a power cycle undoes anything you have not committed.

Step 0 — snapshot what you started with

```
GET ALL
```

Copy the output somewhere. It is one long line containing all 54 keys, and it is your undo button.

Step 1 — prove the e-stop works

Before any wheel turns. Press the e-stop, then:

```
STATUS
```

You should see `estop=1` and `fault=ESTOP`. Release it, send `ENABLE ON`, and confirm the fault clears. **If this does not behave exactly as described, stop and fix the wiring** — everything downstream assumes you can stop the robot.

Step 2 — get the directions right

Put the robot on blocks so the wheels spin free. Command both wheels forward gently:

```
WHEEL l=0.2 r=0.2
```

Fix the two possible sign errors **in this order** — motor direction first, then encoder direction. Reversing them in the wrong order will chase its own tail.

First, motor direction. Does each wheel physically drive the robot forward? For any wheel that spins backwards, flip its motor invert:

```
SET motor.invert_left=1
SET motor.invert_right=0
```

Defaults are `motor.invert_left=0`, `motor.invert_right=1` — asymmetric because the two motors face opposite ways on a typical chassis.

Then, encoder direction. With both wheels now physically driving forward, check that the reported velocities are positive:

```
STATUS
```

Look at `v1` and `vr`. A wheel moving forward must report a **positive** velocity. If one is negative, flip its encoder invert:

```
SET encoder.invert_left=1
SET encoder.invert_right=0
```

Defaults are `encoder.invert_left=0`, `encoder.invert_right=1`.

A wheel that drives forward but reports negative velocity creates a positive feedback loop in closed-loop control — the robot will run away, and your `/odom` will be wrong in a way that is very hard to debug from inside ROS 2. This step is not optional.

Step 3 — teach it your scale

These three numbers determine whether /odom and TF are correct. Everything in the ROS 2 navigation stack inherits errors made here. Set them in this order, because each measurement depends on the one before it.

`kin.counts_per_rev` — encoder counts per wheel revolution, after gearing.

```
counts_per_rev = gearbox_ratio × encoder_CPR × 4
```

The $\times 4$ is quadrature decoding. The factory default of **2249** corresponds to the reference platform: a 46.85:1 gearmotor with a 48-count encoder ($46.85 \times 48 = 2249$). For a 30:1 gearbox with a 12-count encoder you would use $30 \times 12 \times 4 = 1440$.

```
SET kin.counts_per_rev=1440
```

If your platform has **no encoders**, set this to 0. The board switches to open-loop and velocity commands return `NO_FEEDBACK` — you can still drive in duty mode, but nothing in §6 applies.

`kin.wheel_radius` — verify by driving, not by measuring the wheel. Command the robot to travel 1.0 m in a straight line, then measure what it actually did. If it travelled 0.92 m, your radius is 8 % too small:

```
SET kin.wheel_radius=0.053
```

Repeat until commanded and actual agree. This catches tyre compression and counts-per-rev errors together.

`kin.track_width` — the distance between the two wheel contact patches. Verify by rotation: command a full 360° spin in place and measure the heading error. Overshooting means the configured track width is too small.

```
SET kin.track_width=0.28
```

Get these three right and everything downstream — odometry, `cmd_vel`, teleop speeds — is correct. Get them wrong and every layer above inherits the error.

NOTE

The driver reads `kin.wheel_radius` and `kin.track_width` from the board once at startup and uses them for odometry. Change them later and you must restart the node.

Step 4 — set your motion envelope

```
SET kin.max_v=1.0      # max linear speed, m/s
SET kin.max_w=3.0      # max angular speed, rad/s
SET kin.max_accel=25.0 # acceleration limit, rad/s² (0 disables)
SET motor.max_rads=20.9 # max motor speed, rad/s – from your motor's rating
```

`kin.max_v` and `kin.max_w` are the envelope the board clamps incoming `/cmd_vel` to. A navigation stack asking for more will silently get less — set them to what the robot can actually do, not what you hope for.

Step 5 — set your motor limits

```
SET motor.v_max=12.0    # nominal motor voltage
SET motor.i_max=4.0     # rated continuous current per side, A
SET motor.i_fault=5.0   # absolute hard trip per side, A
SET motor.i_max_time_ms=500 # how long i_max may be exceeded before tripping
```

Current protection is two-tier. Exceeding `motor.i_max` for longer than `motor.i_max_time_ms` trips an overcurrent fault — that is the thermal protection. Exceeding `motor.i_fault` trips immediately — that is the "something is badly wrong" limit. Set `i_max` from your motor's continuous rating, and keep `i_fault` above it but within the board's 4A per-side continuous rating.

Step 6 — set your battery

Defaults assume a 3S Li-ion/LiPo pack. Set four values to match your chemistry:

```
SET batt.full_v=12.6    # 100% reference
SET batt.low_v=10.2     # warning threshold
SET batt.critical_v=9.6 # LOWBATT fault – motors stop
SET batt.hyst_v=0.2     # hysteresis, stops threshold chatter
```

`batt.critical_v` is a real cutoff: crossing it stops the motors and latches LOWBATT, and the board will refuse ENABLE ON until the voltage recovers. Set it above the point where your pack is damaged by over-discharge. These values also drive the `/battery_state` topic, so getting them right makes ROS-side battery monitoring meaningful.

Do **not** touch `batt.divider_ratio` — that is factory calibration of the voltage sensing itself, not a property of your battery.

What you should leave alone

Beyond the factory calibration group, these are advanced or bench-only and are correctly set to their defaults for normal use:

- `current_loop.enable`, `pid.kp_i`, `pid.ki_i` — the inner current cascade
- `ff.static`, `ff.visc`, `ff.inertia`, `pid.kp_c`, `pid.ki_c` — cascade feed-forward
- `encoder.obs_enable`, `encoder.obs_alpha`, `encoder.obs_beta` — velocity observer
- `motor.stall_duty`, `motor.stall_vel`, `motor.stall_ms` — stall detection thresholds

The cascade is **disabled by default and is not yet validated for production use**. Enabling it with zero gains raises a CASCADE_NOGAIN fault that blocks all motion. Leave `current_loop.enable=0`.

A complete per-parameter reference, including valid ranges, is in `CONFIG_PARAMETERS.md`.

6. TUNING THE VELOCITY LOOP

With geometry and directions correct, tune the velocity controller to your motors. The board ships with `pid.kp=0.5`, `pid.ki=7.0`, `pid.kd=0.0`, tuned for the reference platform — a good starting point, rarely optimal for yours.

An automated tuner is provided:

```
python3 tools/mc1_optimise.py <PORT> --scenario autotune-all
```

This sweeps `pid.kp`, then `pid.ki`, and **saves both** to flash. It is the customer-facing tuning path. Expect it to drive the wheels through a range of speeds for a minute or two. **Run it with the ROS 2 driver stopped** — the tool needs exclusive access to the port.

Useful diagnostic scenarios, which change nothing:

SCENARIO	WHAT IT DOES
open	Open-loop duty ramp. Exposes cogging, deadband, and supply sag
closed	Closed-loop velocity ramp. Exposes oscillation and tracking error
all	Both, in sequence (the default)

The tuner temporarily raises the SLOW duty limit so it can reach test speeds, and restores it when it exits.

NOTE

Bench-only scenarios. `autotune-inner` and `autotune-vehicle` tune the current cascade, which is not validated for production. Do not run them on a customer platform.

Judge the result by ear and by eye. A well-tuned loop holds commanded speed without audible hunting and without visible oscillation when you push against the wheel. If the tuner leaves the robot buzzing or twitching, reduce `pid.ki` and re-test.

7. SAVE IT AND PROVE IT

Nothing so far survives a power cycle. Commit it:

```
SAVE
```

Then **prove it stuck** — power the board down completely, bring it back up, and verify:

```
GET ALL
```

Compare against what you configured. This step catches the single most common bring-up mistake, which is a session of careful configuration that was never saved.

Three commands manage persisted configuration:

COMMAND	EFFECT
SAVE	Write current settings to flash
LOAD	Discard RAM changes, reload from flash
DEFAULTS	Reset robot configuration to factory values — preserves calibration

WARNING
Close the serial terminal before continuing. The ROS 2 driver needs the port, and only one process can hold it.

Your board is now configured. Everything below is ROS 2.

8. INSTALLING THE ROS 2 DRIVER

Prerequisite: the ORCP Python library

WARNING

The driver imports `orcp`, but does not declare it as a ROS dependency. `rosdep` will not install it and `colcon` will not warn you. Install it yourself, into the same Python interpreter ROS 2 uses:

```
pip install orcp
```

Miss this and the node fails at startup with `ModuleNotFoundError: No module named 'orcp'`.

Native Linux (Ubuntu 24.04 + Jazzy)

```
mkdir -p ~/orcp_ws/src && cd ~/orcp_ws/src
git clone https://github.com/orcp-protocol/orcp-ros2.git
cd ~/orcp_ws
colcon build --symlink-install
source install/setup.bash
```

Source both ROS 2 and the workspace in every new shell:

```
source /opt/ros/jazzy/setup.bash
source ~/orcp_ws/install/setup.bash
```

macOS via Docker

WARNING

Read this before committing to the Docker route. Docker Desktop on macOS **cannot pass a USB serial device into a container**, and Colima requires explicit device passthrough that is fiddly at best. There is no reliable way to reach `/dev/cu.usbmodem*` from inside a container on macOS.

Practical options:

- **Run ROS 2 on a Linux machine** — the robot's onboard computer is the natural place
- **Use a network bridge** and point the driver at `socket://host:port` instead of a serial device
- **Use Docker for simulation only** (Appendix A), and drive real hardware from the Python guide instead

With that understood, the container path is:

```
docker build -t orcp-ros2-dev .
docker run -dit --init --name orcp -p 8765:8765 \
    -v /path/to/orcp:/work orcp-ros2-dev sleep infinity
docker exec orcp pip install --break-system-packages orcp orcp-sim
docker exec orcp bash -lc \
    'source /opt/ros/jazzy/setup.bash && cd /work/orcp-ros2 && colcon build --symlink-
install'
```

Mount the **parent** directory so sibling repositories are visible inside the container. Port 8765 is published for Foxglove (§10). The `--break-system-packages` flag is required by PEP 668 on Ubuntu 24.04.

Verify the build

```
ros2 pkg list | grep orcp_ros2
```

9. THE DRIVER INTERFACE

Start the node against real hardware:

```
ros2 run orcp_ros2 orcp_driver --ros-args -p port:=/dev/ttyACM0
```

NOTE

The bundled launch files take no arguments and hard-code /tmp/orcp for the simulator. For hardware, use `ros2 run` as above, or write your own launch file.

Parameters

PARAMETER	TYPE	DEFAULT	NOTES
port	string	/tmp/orcp	Device path, or socket://host:port
preset	string	SLOW	SLOW or NORMAL. Runtime-settable
odom_frame	string	odom	
base_frame	string	base_link	
publish_rate	double	20.0	Odometry and TF rate, Hz
status_rate	double	4.0	Diagnostics rate, Hz
cmd_timeout	double	0.0	Seconds; 0 disables. Runtime-settable

Two things to know:

- **There is no baud parameter.** The connection is always 115200.
- **Only preset and cmd_timeout take effect at runtime.** The rest are read once at construction; changing them requires a node restart.

Setting preset to NORMAL also enables the motors and starts a 100 ms heartbeat automatically. Setting it back to SLOW stops the heartbeat.

Subscribed topics

TOPIC	TYPE	NOTES
/cmd_vel	geometry_msgs/Twist	Uses linear.x and angular.z
/wheel	std_msgs/Float64MultiArray	data: [left_rad_s, right_rad_s] — direct per-wheel control

Published topics

TOPIC	TYPE	RATE	NOTES
/odom	nav_msgs/Odometry	publish_rate	Dead-reckoned from wheel velocities
/battery_state	sensor_msgs/BatteryState	publish_rate	Voltage, percentage, presence
/diagnostics	diagnostic_msgs/DiagnosticArray	status_rate	Preset, fault, velocities, duties, battery

Transforms

The driver broadcasts **odom** → **base_link** at `publish_rate`.

Two caveats worth knowing before you build a navigation stack on this:

- **Odometry is dead-reckoned from wheel velocities only.** There is no IMU fusion and no slip compensation. It drifts.
- **Covariances are published as zero.** If you feed `/odom` into a filter that respects covariance, supply your own values.

There is no `joint_states` publisher — the reference URDF models the wheel joints as fixed, so `robot_state_publisher` alone is sufficient.

Services

SERVICE	TYPE	NOTES
<code>/orcp_driver/stop</code>	<code>std_srvs/Trigger</code>	Immediate stop
<code>/orcp_driver/enable</code>	<code>std_srvs/SetBool</code>	true = enable. This is how you clear a latched fault

The command watchdog

`cmd_timeout` is a ROS-side dead-man switch: if no `/cmd_vel` arrives within the window, the driver sends a stop.

Leave it at 0 (disabled) for keyboard teleop — `teleop_twist_keyboard` sends one message per keypress, so a watchdog turns driving into hold-and-repeat. Enable it for autonomous operation, where a silent planner should stop the robot.

What is deliberately absent

Configuration is not exposed to ROS 2 — no parameters, topics, or services for GET/SET/SAVE. Kinematics and PID must be configured over serial (§5–7) before the node starts.

The driver holds the serial port **exclusively**. To change configuration:

1. Stop the node
2. Configure over serial, SAVE
3. Restart the node

10. DRIVING THE PLATFORM

Keyboard teleop

The driver has no teleop of its own — use the standard ROS 2 package.

Terminal 1, the driver:

```
ros2 run orcp_ros2 orcp_driver --ros-args -p port:=/dev/ttyACM0
```

Terminal 2, teleop:

```
ros2 run teleop_twist_keyboard teleop_twist_keyboard
```

The board boots in SLOW with motors enabled, so this drives immediately. Keep `cmd_timeout` at 0 for comfortable keyboard driving.

To go faster, switch to NORMAL — which enables the motors and starts the heartbeat in one step:

```
ros2 param set /orcp_driver preset NORMAL
```

Verify it is working

```
ros2 node list                # /orcp_driver present
ros2 topic hz /odom           # ≈ publish_rate
ros2 topic echo /odom --field pose.pose.position
ros2 topic echo /diagnostics  # preset, fault, velocities
ros2 run tf2_tools view_frames # odom → base_link
```

Drive forward a measured distance and compare against `/odom`. If they disagree, your kinematics are wrong — go back to §5 step 3. **Fix it there, not with a scale factor in ROS.**

Visualising with Foxglove

```
ros2 launch orcp_ros2 viz.launch.py
```

This starts the simulator, the driver, `robot_state_publisher` with the reference URDF, and `foxglove_bridge` on port 8765.

Connect Foxglove to `ws://localhost:8765` using the **"Foxglove WebSocket"** connection type — *not* Rosbridge. Add a 3D panel and set the fixed frame to `odom`.

NOTE

Browser limitation: `app.foxglove.dev` is served over HTTPS and will block a plain `ws://localhost` connection as mixed content. Use the Foxglove desktop application.

Headless alternative:

```
ros2 topic echo /odom --field pose.pose.position
```

Recovering from a fault

The robot stops and `/diagnostics` reports the reason. Clear the physical cause if there is one, then:

```
ros2 service call /orcp_driver/enable std_srvs/srv/SetBool "{data: true}"
```

Fault codes are also logged as ROS warnings by the driver, so they appear in your console and in `rqt_console`.

11. FAULT REFERENCE

Every fault latches. The `~/enable` service is the universal recovery unless noted otherwise.

CODE	MEANING	WHAT TO DO
OK	No fault	—
TIMEOUT	No motion command within the preset's timeout window	Resume publishing <code>/cmd_vel</code> , then enable
HEARTBEAT	No heartbeat within <code>hb.timeout_ms</code>	The driver manages this — check the node is alive, then enable
NOT_ENABLED	Motion commanded while disabled	Call <code>~/enable</code> with true
ESTOP	E-stop loop open — button pressed, wiring broken, or J3 has no switch and no link fitted	Release the e-stop and confirm wiring, then enable. Motor supply is physically disconnected until the loop closes
LOWBATT	Pack fell below <code>batt.critical_v</code>	Charge or replace. Enable is refused until voltage recovers
CHARGER	Charger-detect loop open — charger connected, or J2 has no switch and no link fitted	Disconnect the charger, or fit a link across J2 pins 1–2, then enable. Drive is inhibited in hardware until the loop closes
MOTOR_LEFT / MOTOR_RIGHT / MOTOR_BOTH	Motor driver reported a hardware fault	Check motor wiring for shorts; look for overheating. Then enable
OVERCURRENT_LEFT / _RIGHT / _BOTH	Current exceeded <code>motor.i_max</code> for longer than <code>motor.i_max_time_ms</code> , or exceeded <code>motor.i_fault</code> outright	Find the mechanical cause. Verify <code>motor.i_max</code> matches your motor. Then enable
ENCODER_STALL	Duty applied but the wheel is not turning	Check for a jam, disconnected encoder, or broken wiring. Then enable
CASCADE_NOGAIN	<code>current_loop.enable=1</code> with both inner gains at zero	Stop the node, SET <code>current_loop.enable=0</code> , SAVE, restart
BOOTLOADER_MISMATCH	Firmware requires a newer bootloader	Motion blocked until the bootloader is updated. Contact support

ENCODER_STALL firing when you bump a wall is **correct behaviour** — it is motor protection. If it trips too eagerly, the thresholds are tunable via `motor.stall_duty`, `motor.stall_vel` and `motor.stall_ms`.

Warnings

Warnings do not stop the robot. They surface as ROS log warnings and in `/diagnostics`.

WARNING	MEANING
BATT	Pack fell below <code>batt.low_v</code> — still running, but plan to charge
AUX5V	The auxiliary 5V rail exceeded <code>aux5v.i_warn</code> or dropped below <code>aux5v.v_warn</code>

12. TROUBLESHOOTING

`ModuleNotFoundError: No module named 'orcp'.`

The library is not installed in the interpreter ROS 2 is using. `pip install orcp`. In Docker, add `--break-system-packages`. See §8.

The node starts, then exits immediately.

Almost always the serial port. Check the path, check nothing else holds it (a serial terminal from §3–7 is the usual culprit), and check permissions — `sudo usermod -aG dialout $USER`, then log out and back in.

The robot does nothing — no motion at all, and no fault that explains it.

Check the two hardware interlocks first (§2). If J3 (e-stop) or J2 (charger-detect) is open — no switch and no jumper link — the motor outputs are disabled in hardware and no amount of software will move the robot. `/diagnostics` will report ESTOP or CHARGER.

No serial port appears at all.

Check the battery first — USB does not power the board, so an unpowered Motion Core never enumerates no matter how good the cable is. Try another USB-C cable — charge-only cables are the usual culprit. Confirm the board is powered and the red LED is blinking.

Check `dmesg | tail` after plugging in.

/odom publishes but never changes.

The robot is not moving, or the encoders are not reporting. Check `/diagnostics` for a latched fault, confirm the e-stop loop is closed, and verify `v1/vr` are non-zero while driving.

/odom distances and rotations are wrong.

Your kinematics are wrong. §5 step 3, in order: `counts_per_rev`, then `wheel_radius` from a measured straight line, then `track_width` from a 360° spin. Restart the node afterwards — those values are read once at startup.

The robot runs away on a small /cmd_vel.

An encoder is reading backwards. §5 step 2. This is the classic symptom.

The robot only moves while I hold a key.

`cmd_timeout` is enabled. Set it to 0 for keyboard teleop: `ros2 param set /orcp_driver cmd_timeout 0.0`.

Nothing moves and /diagnostics says not enabled.

You are in NORMAL without having enabled the motors. Call `~/enable`, or set `preset` to NORMAL (which enables automatically).

The simulator launch file races and the driver cannot connect.

`sim_bringup.launch.py` waits a fixed 2.5 s for the pseudo-terminal to appear. On a loaded machine that can be too short — relaunch.

Foxglove will not connect.

Use the **"Foxglove WebSocket"** connection type, not Rosbridge. Use the desktop application — the browser blocks `ws://localhost` from an HTTPS page. Confirm port 8765 is published if you are in Docker.

I cannot reach `/dev/ttyACM0` from Docker on macOS.

You cannot. See the warning in §8 — run ROS 2 on Linux, or use a `socket://` bridge.

My board settings vanished after a power cycle.

You did not SAVE (§7).

WHERE TO GO NEXT

- `CONFIG_PARAMETERS.md` — complete per-parameter reference with valid ranges
- `BATTERY_PROFILES.md` — supported battery chemistries and their threshold values
- *Motion Core — Getting Started (Python)* — driving the board directly from Python, and the tuning workflow in more depth

APPENDIX A — TRYING IT WITHOUT HARDWARE

The repository ships a launch file that starts the ORCP reference simulator and the driver together, so you can exercise the whole ROS 2 graph before a board arrives.

```
pip install orcp orcp-sim
ros2 launch orcp_ros2 sim_bringup.launch.py
```

That starts a simulated Motion Core on `/tmp/orcp` and connects the driver to it. In a second terminal:

```
ros2 topic echo /odom --field pose.pose.position
```

Drive it with keyboard teleop (§10) and watch the position change.

Two limitations to know about:

- **The simulator requires macOS or Linux.** It uses a pseudo-terminal, which Windows does not provide.
- **The bundled `mc1` profile models firmware 1.8.0 with 46 configuration keys**, while current firmware is 1.9.6 with 54. The ROS 2 interface is identical either way — the driver only reads two config keys — but treat the simulator as a way to learn the graph, not to rehearse configuration.