FIRMWARE MANUAL

DlaSteer-100

电动转向执行器 · 出厂固件说明书

副标题	DlaSteer-100 · 出厂固件 FOC_steer · 使用说明书
品牌	Daliang Auto
产品型号	DlaSteer-100
固件名称	FOC_steer
固件版本	0.1.10（构建标识见随包 `MANIFEST.txt`）
文档版本	Rev. 1.1
发布日期	2026-09-10
控制台	USB Type-C 虚拟串口，115200 8N1，无流控
对外接口	控制台命令行 · CAN（转向指令帧 / 状态帧）

本书与规格书的分工

- **《DlaSteer-100 产品规格书》描述硬件**：机械传动、电气参数、接口、限值、外形尺寸。 硬件不改版，规格书就不改。
- **本书描述出厂固件**：命令、出厂标定流程、限位与保护逻辑、CAN 协议、升级方式。 **固件每发布一版，本书跟着走一版**，两者互不牵动。
- 两本书里都出现的量（例如机械行程、齿条速度），**以本书为准的只有"固件现在 用的是哪个数"**；硬件事实以规格书为准。

板子上跑的是哪一版，用 `ver` 问它，不要从出厂日期推。 而且 `ver` 报的 `cmdhash` 必须和随包 `COMMANDS.json` 里的相等 —— 那是唯一一条把这份包和这台执行器绑在一起的证据，版本号和提交号都不是。

---

1. 这套固件是什么

出厂固件把这台执行器变成一台**可以直接接进整车的闭环转向机**： 上位机（或整车控制板）只发一个**目标位置**，执行器自己走过去并停住。 限位、减速、越程保护、失联保持，全部在它内部完成。

⚠ **它不是一台"电机驱动器"。** 不要在外面再套一个位置环 —— 两个位置环互相追的样子是"抖"，而不是一条报错。

1.1 功能地图

层	固件提供什么
位置伺服	位置环 200 Hz → 速度环 1 kHz → 电流环 20 kHz FOC，三级级联
绝对位置	单圈绝对式磁编码器经同步带接转轴，**上电即知位置，不需要回零**
纯软件限位	没有任何限位开关。软限位、减速带、越程锁存、翻圈余量、传感器故障检测共同保证齿条不撞机械止点
停止包络	接近限位和目标时按 `v ≤ √(2·a·d)` 限速，保证剩下的距离里一定停得住
越程可恢复	越程锁存之后可以经受限通道（限速、只许朝中位）自己退回来，不需要拆机
出厂验收	一条 `prodcal` 跑完十二道门，不合格的自己说下一步做什么
自标定	电机换相参数（霍尔映射、相电阻电感、电流环增益）一条命令学出来；齿条位置标定一条命令做完
CAN	收 `0x230` 转向指令、发 `0x231` 状态，失联自动保持位置

1.2 两条对外通路

通路	对面是谁	物理层	本书章节
控制台命令行	人，或上位机软件	USB Type-C（USART1），115200 8N1	§2 – §5
CAN	整车控制板 / 上位机	经典 CAN，1 Mbps	§6

⚠ **这台执行器不接收整车行驶请求帧 `0x210`，也不发送 `0x211` / `0x212`。** 那些 ID 属于牵引驱动板，本节点发它们会在整车总线上和左牵引板撞车。 ⇒ 它只认一条指令帧： `0x230` 。

⚠ 这条设计有一个**代价**，写在这里而不是留给人去发现：因为不收 `0x210`， 这台执行器**拿不到任何整车层面的信息**。"整车已经撤防"这件事只能通过 `0x230[2]` 的 bit0（使能位）传进来。⇒ **整车撤防时，控制板必须把那一位放掉。**

1.3 这台执行器和裸板不是一回事

DlaSteer-100 是**装好的总成**：电机、行星减速、伞齿轮、齿轮齿条、绝对编码器和 FOC 控制器在一个壳子里。控制器用的是 `FOC_single` 那块 PCB， 但**跑的是不同的固件**（`FOC_steer`），两者不能互换：

⚠⚠ **把 `FOC_single` 的镜像烧进这台执行器，它会正常启动、正常在串口上说话、 通过每一道校验 —— 而齿条没有任何限位。** 认固件只认发布包 `MANIFEST.txt` 里的 `name: FOC_steer`，不要按目录名、也不要按"哪一版最新"挑。

---

2. 接上控制台

2.1 接线与串口参数

项	值
物理口	USB Type-C（板载 USB-UART 桥，接 USART1）
波特率	115200，8 数据位，无校验，1 停止位，无流控
换行	板子回 `CR LF`；发过去 `CR` 或 `LF` 都认

敲 `help` 会列出全部 83 条命令。

2.2 每条命令都标了它会不会让机器动

随包 `COMMANDS.json` 给每条命令标了一个 `danger`：

标记	含义	例子
`none`	只读，回答一个问题，什么都不改	`ver` `steer`（不带参数） `prot` `enc`
`gates`	改状态或改一个设置，但**转不动任何东西**	`save` `clear` `stop` `steerpid` `steertmo`
`motion`	**可能让齿条动或让轴转**	`drive` `steermm` `steercal` `learn` `prodcal`

⚠ **一条命令按它最危险的那种形式分类。** `prodcal` 不带参数确实只读，但 `prodcal run` 会驱动齿条、`prodcal motor` 会转轴 —— 所以整条命令是 `motion`。同理 `steermot`：它大部分键是几何参数， 但其中一个是 `inv`，而 `inv` 反转的是"一条指令往哪边推齿条"。

2.3 任何时候都可以打断的一条

```
stop
```

`stop` 立刻关掉栅极总使能（TIM1 的 MOE）并把 DRV8323 的 ENABLE 拉低 —— 两条互相独立的通路，任何一条成立齿条就不会再被驱动。

⚠ `stop` 之后齿条是**自由的**：这台机器没有抱闸，也没有自锁蜗杆。 拉杆上有力的时候它会被推着走。

2.4 最先该敲的三条

```
ver         这是哪一版固件、哪块板、cmdhash 是多少
prodcal     十二道出厂门现在过了几道，没过的下一步做什么
steer       齿条现在在哪、标定过没有、有没有故障
```

`ver` 的回答里有一行是给程序读的：

```
=tlm 1 build debug=0 stamped=1 crc=1 major=0 minor=1 patch=0 board=foc_steer scm=<commit> hw=1.0 cmdhash=1235599930
```

⚠ `debug=1` 表示这是 **Debug 构建，不要拿它控制真实硬件。**

`stamped=` 和 `crc=` 是两件不同的事，别当成一个看：

	意思	怎么办
`stamped=1` `crc=1`	镜像带着校验和，而且板子里的字节现在仍然对得上	正常

	意思	怎么办
`stamped=0`	这份镜像里**没有**校验和——它是从编译器直接烧进去的开发版，不是发布包里的	不是故障；但上车请用发布包
`stamped=1 crc=0`	有校验和，**而且对不上了**	重新烧录；反复出现就是 flash 坏了

⭐ 板子每被问一次就把整段镜像重算一遍，不是报一个开机时存下的结果—— 开机时算出来的那个答案，正是发现不了后来才发生的损坏的那个。

---

3. 出厂标定：一条 `prodcal`

3.1 三种形式

```
prodcal          只读。什么都不改，随时可以敲
prodcal run      标定齿条 + 写 flash + 复核全部十二道门（减速比没填会拒绝）
prodcal motor    会转轴的一步（= learn）。单独一个词，故意的
prodcal sweep    会推齿条的一步：先探 3 mm 定方向，再到两端软限位前 5 mm 处、回中
```

⚠ `prodcal sweep` 是 0.1.1 起加的，**方向符号 `st_invert` 由它探出来**，不再靠人填： 起步先朝正方向走 3 mm，位置反着走 2 mm 以内就判"接线反了"，自己把符号翻过来 （第一次）或者锁 `DIR` 故障停轴（翻过之后还反）。它会自己武装轴，扫完自己撤防并在 控制台打一行；扫描期间 CAN 上有整车指令的话它会拒绝起步。

3.2 输出格式（人和程序读同一份）

```
=prodcal begin 1 count=12
=prodcal 1 fw     state=pass ver=0.1.10 scm=… board=foc_steer hw=1.0 cmdhash=… uid=…
=prodcal 1 geom   state=pass gear=1000 rpm=400 gearxrpm=4000 bevel=100 potr=171 mod=150 teeth=20 pp=2 gear_set=1
=prodcal 1 limits state=pass ibus_ma=4500/4500 iqmax_ma=8000/8000 …
=prodcal 1 vbus   state=pass mv=24160 min=18000 max=30000
=prodcal 1 drv    state=pass ocp=0x019 csa=0x243
=prodcal 1 enc    state=pass reads=200 bad=0 statuserr=0 ang=8504
=prodcal 1 hall   state=pass raw=6
=prodcal 1 motor  state=pass hallmap=1 r_mohm=365 l_uh=493
=prodcal 1 phase  state=pass src=calibrated-centre raw=8194 target=8192 off=2 tol=…
=prodcal 1 rack   state=pass min=14954 ctr=8194 max=1434 store=bank1-last-page dir=-1 want_dir=-1
=prodcal 1 wrap   state=pass plus_mm10=… minus_mm10=… hard_mm10=550 …
=prodcal 1 sweep  state=pass reached_mm10=430/-432 slow_at_mm10=420/-420 want_mm10=420
=prodcal 1 dir    state=pass verified=1 inv=0 inv_set=1
=prodcal end verdict=pass fail=0 pending=0
```

- `state` ∈ `pass` / `fail` / `pending`
- `verdict` ∈ `pass` / `fail` / `blocked` （有 pending 就是 blocked）
- **值不含空格、不含浮点。** 毫米一律 `*_mm10` （整数，单位 0.1 mm）
- ⚠ `count=12` 数的是**门**， `fw` 那一行是抬头不算在内 ⇒ 一共 13 行。
- `rack` 行的 `min > max` 是**正常的**（0.1.9 起）： 正方向按产品约定定成"齿条向右"， 而编码器计数在这个方向上是减小的； `dir=` 是这份记录的计数方向， `want_dir=` 是产品的。
- `dir` 行的 `inv_set=` 说的是"方向符号有没有人定过"—— `inv=0` 既可能是"探过了、不反"， 也可能是"从没探过"，光看 `inv` 分不出来。

3.3 不合格的门自己说下一步

不合格的步骤会多打一行 `need=`，键里不含空格：

```
=prodcal 1 geom   need=set-the-gearbox-ratio-this-unit-was-built-with-steermot-gear-1000-or-1500-or-2000-then-save why=…
=prodcal 1 enc    need=replug-the-encoder-connector why=…
=prodcal 1 motor  need=free-the-shaft-then-run-prodcal-motor
=prodcal 1 phase  need=slacken-the-belt-turn-the-encoder-pulley-to-8192-counts-then-re-tension
=prodcal 1 rack   need=push-the-rack-to-mechanical-centre-then-prodcal-run why=this-record-was-taken-with-…
=prodcal 1 limits need=set-the-production-limits-then-save
=prodcal 1 wrap   need=re-phase-the-belt why=…
=prodcal 1 sweep  need=run-prodcal-sweep why=nothing-has-driven-the-rack-across-its-travel-since-reset
=prodcal 1 dir    need=command-one-move-of-at-least-20mm-s-and-…
```

拒绝执行的命令也是同一个口径，一行 `xxx: refused -- <原因>`，原因同样是连字符句，例如 `prodcal sweep: refused -- direction-sign-never-set-on-this-unit-run-steermot-inv-0-or-1-then-save-see-dir-gate`。

⇒ `need=` **就是操作员要做的那件事。** 把连字符换成空格显示即可。 ⚠ **不要自己另写一套提示** —— 另写的那套会和固件分叉， 而这个项目已经为这件事付过一次账：一份作业指导书描述了板子上已经不存在的 传感器，整整六周，什么都没有失败。

3.4 装配与标定的十步

```
1  烧录固件
2  ⚠ 完整断电重启（断开电源，确认掉电后再上电）
3  填配置：
     · 极对数：默认 2，界面展示出来给人确认即可，不要求填
     · 减速比：⭐ 必填，10 / 15 / 20，操作员从减速箱铭牌读（steermot gear，额定转速跟着自动填）
       ⚠ 没填 prodcal run / sweep 都会拒绝—默认值是一个"合法但没人选过"的数
4  齿条摆到机械中位并卡住 → 转编码器皮带轮到 8192 counts → 张紧皮带
     → prodcal run          ← 记录中位并写 flash
5  ⭐ 到这一步转向机就可以完整装配了（装电机）
6  prodcal motor            ← 电机自标定，会转轴
7  prodcal sweep            ← 探方向、全行程扫描，会推齿条；扫完控制台自己说一声
8  save
9  prodcal                  ← 复核全部十二道门
10 verdict=pass 放行
```

台架实测（2026-09-10，0.1.9）：从整片擦除到 `verdict=pass` 13 分钟。

⭐ **第 4 步先记中位、第 6 步再 learn，是安全的**： `steercal` 存下来的是 「编码器计数 ↔ 毫米」这个**映射**， `learn` 之后齿条被挪到哪里都不改变这个映射。

⚠ 3.4.1 第 4 步的 `verdict` 一定不是 pass，这是正常的

第 4 步电机还没装、霍尔线没接 ⇒ `hall` 读到 0 或 7 ⇒ `hall state=fail`， `motor state=pending`。

⇒ **第 4 步只看 `rack` 和 `wrap` 两行**，别看 `verdict`。 `verdict` 只在第 9 步（电机装好、扫描做完之后）才有意义。

⚠ 固件不打算把「电机没装」和「霍尔坏了」分开报 —— **板子分不出来**， 而把分不出来的两件事报成同一件，比报成两件安全。

⚠ 3.4.2 第 4 步「卡住齿条」不是客套

皮带松开之后齿条是自由的。台架实测两次读数之间它自己滑了 **4.9 mm**， 而滑掉的每一毫米会**一比一地**吃掉翻圈余量（见 §5.4）。

⚠ 3.4.3 减速比只能靠人填

同一台电机配三种行星减速箱（10 / 15 / 20，PB61010391 / PB61015391 / PB61020391）， **电机本体完全相同** ⇒ 板子没有任何办法测出来。

```
steermot gear 20
save
```

⚠ **减速比不影响齿条标定**（`steercal` 只用编码器和齿轮齿条几何）， 它影响的是**速度换算**和 `encmeas` 的校验。 ⇒ 不要写成「不填就标不了」，要写成「**不填速度是错的**」。

⚠⚠ 3.4.4 烧录之后必须完整断电重启

烧完固件、上电之前，**必须完整断一次电**（断开电源，确认完全掉电后再上电）， 然后 `prodcal` 的 `enc` 那一行必须是 `state=pass`。

⇒ 这一步在上位机里应该是一个**必须点确认的门**，不是一行提示文字。

3.5 极对数是 2，不是 4

⚠⚠ 厂家规格书那一行写的是 `Poles 4`，而 **4 极 = 2 对极。**

编译期默认已经是 2。但**如果界面允许改，默认值要给 2**，并把这句话写在旁边： 写成 4 的时候 `encmeas` 会报出一句**自信的、关于硬件的错话** （"编码器在行程内会翻圈"），而不翻圈正是当初选这套皮带轮的全部理由。

⚠ 而且用极对数换算过的**每一个历史台架速度数都作废了** —— "命令 120 mm/s 时电流环饱和"实际上是在要 240 mm/s。饱和是真的，它证明的东西不是。

---

4. 运行模式与位置环

4.1 三种模式

```
mode            查询
mode idle       空转，不驱动
mode learn      电机自标定
mode steer      转向伺服（正常工作模式）
```

⚠ **牵引相关的模式在这台执行器上是关掉的。** 这块 PCB 的固件家族里有转矩、 速度、开环等模式，那些留给驱动板用；这里只保留 `idle` / `learn` / `steer`。

⭐ **换模式本身产不出转矩**： `Mode_Set()` 会先重置电流调节器并把两个参考清零， 必须有别的东西随后写一个非零参考。⇒ 换模式是"要问一次"级别， 而**给桥上电（`drive on`）是最高一档** —— 上电会施加此刻已经立在那里的参考值， 那个值可能是上一条命令留下的。

4.2 让它动起来

```
mode steer
drive on
steermm 150          目标 +15.0 mm（单位 0.1 mm）
steer 500            或者按千分比：+500‰ = 半行程
steer                看现在到哪了
steer off            撤销使能，目标不再被追
```

- `steer <‰>` 把 ±1000‰ 映射到 ±软限位。⇒ `steerlim` **改的不只是保护限值， 也是这条输入的量程。**
- `steer on` 是**无扰使能**：它把目标先设成当前位置，所以使能那一下齿条不会跳。
- `steer` 第二行的 `src=spi-encoder` 说的是位置从哪来（0.1.9 起；以前打的是电位器的 ADC 脚名）。
- `steermm` **不夹目标**：超过软限位加硬余量会锁 `OVERTRAVEL`。给人看"左右转几下"用 ±200~300（±20–30 mm）足够。
- 编码器插好之后读数会立刻跟上，不需要 `steer clear`（0.1.6 起；以前读数会冻在故障前的值）。

4.3 ⚠ 每条指令只驱动 0.3 秒

`st_timeout_ms` 默认 **300 ms**：收不到新指令，执行器就保持当前位置。 这是失联保护，不是 bug。

⇒ 手敲命令做一整段移动时（约 2–3 mm 就停），临时放大：

```
steertmo 3000
… 做完整段移动 …
steertmo 300         ⚠ 跑完记得改回来
```

⚠ 这一项**不进 flash**，掉电自动回到 300 ms —— 会被人忘掉的那一个是临时的那一个。

4.4 三级级联

环路	频率	输出
位置环	200 Hz	齿条速度指令（PD + 停止包络）
速度环	1 kHz	Iq 参考
电流环	20 kHz	SVPWM 占空（Id/Iq PI，圆形限幅 + 抗饱和）

---

5. 齿条限位与保护

5.1 这台机器没有限位开关

每一条端点保证都来自编码器加固件里的检查。 没有微动开关，也没有机械缓冲行程 可以挥霍：软限位到机械止点之间只有 **4.9 mm**。

5.2 四个区

区	条件	行为
`NORMAL`	行程内	正常伺服
`SLOW`	离软限位 ≤ 减速带宽度	按距离缩放速度
`LIMIT`	到达/越过软限位	**只允许朝中位的运动**
`FAULT`	锁存故障	完全不动

5.3 越程锁存与恢复

越过软限位再加"硬余量"就锁存 `OVERTRAVEL`。锁存之后普通的 `clear` 清不掉 （条件还成立），所以有一条受限的恢复通道：

```
mode steer
drive on
steer recover
```

⚠ **顺序不能反。** 先发 `steer recover` 再发 `mode steer` 的话， `mode steer` 会重新 seat 目标并解除武装，恢复就白发了。

恢复通道的性质： - 速度被压到 10 mm/s - 目标强制为中位（只许朝里走） - 回到行程内**自动解除武装** - ⚠ **有时限**（10 s）：它不能无限期地压住三道保护 - ⚠ **它自己盯着位移**（0.1.8 起）：回程开始后 |位置| 只许变小，往外超过 2 mm 就停轴、锁 `DIR` 故障—— 一台方向符号反了的机，回程会把齿条往止点推，这一条就是拦它的 - 方向符号从没定过的机（ `st_inv_set=0` ）**拒绝回程**，理由和 `prodcal sweep` 拒绝扫描是同一句

⚠⚠ **只有 `OVERTRAVEL` 能走这条路。** `SENSOR` / `JUMP` / `DIR` 三种故障意味着 **位置读数本身不可信**，而一条"朝中位走"的命令在读数不可信时不知道中位在哪边。

5.4 ⚠⚠ 翻圈：唯一一条三道检查都不响的失效

单圈绝对编码器转过一圈之后，一个物理位置会读出**另一个位置的合法数值**。

上电那一刻： - 跳变检测**不响** —— 没有前一个样本可比 - 开路/短路检测**不响** —— 数值在标定带里 - 越程保护**不响** —— 数值在软限位里

⇒ 三道检查一起保持沉默，而执行器会朝着它已经顶住的那个止点继续推。

这个隐患软件修不了 —— 信息在传感器那里就已经丢了。修法是机械的： 编码器经 24/14 同步带接转轴，使**翻圈点落在机械止点之外**。

量	值
编码器可测范围	±60.6 mm
机械止点（实测，两端手推）	±55 mm
全部余量（几何决定，改不了）	11.7 mm，完美相位时两侧各 5.9 mm

⚠ **皮带相位每偏一毫米，一侧的余量就整个搬到另一侧。** 这就是 `prodcal` 里 `phase` 和 `wrap` 两道门存在的原因，也是第 4 步要求把齿条卡住的原因。

`wrap` 那道门按实测止点 55 mm 算余量（0.1.1 起；0.1.0 按 51.5 mm 算，偏乐观 3.4 mm）。 `prodcal sweep` 的两端停在软限位之前 5 mm（0.1.7 起）——减速带之后只剩 1.5 mm 余量时 齿条曾冲到 51.9 mm 锁越程，那是台架撞出来的。

---

6. CAN

经典帧，1 Mbps。

6.1 转向指令帧 `0x230` （控制板 → 执行器）

字节	内容
0:1	目标 int16 LE：千分比，或 mm×10（当 bit1 置位）
2	标志：bit0 使能 / bit1 单位=mm / bit2 回中 / bit3 清故障
3	速度限制，0..255 映射为已配置上限的 0..100%（0 = 用默认）

字节	内容
4	序号（在状态帧里回显）
5	指令超时 ×10 ms（0 = 用默认）

⚠ 第 3、5 字节的覆盖值**只作用于本帧**，不会写进持久化配置 —— 否则一帧走偏的报文可以把齿条速度永久压低，而随后一次 `save` 会把它固化进 flash。

⚠⚠ 6.1.1 清故障是按「清故障位本身」的边沿触发的

契约规定 `0x230[4]` 的序号**每帧自增**。所以一个盯着序号变化的边沿检测器 等于一个电平检测器 —— 一个一直把 bit3 举着的控制器会**每 20 ms 清一次故障**， 而越程故障在齿条静止之后不会重新锁存 ⇒ 翻圈保护就真的没有了，而且没有任何地方会说出来。

⇒ 现在的判据是 **bit3 本身的上升沿**。

⚠⚠ 6.1.2 一帧同时带「使能 + 清故障」不会上电

只要 bit3 还举着，轴就上不了电。控制器必须**放掉 bit3，再发使能** —— 两帧，这本来就是"一次新的使能"的意思。

⭐ 而且它**自己会说话**：控制器做错了，轴就是不上电。

6.2 状态帧 `0x231`（执行器 → 控制板）

字节	内容
0:1	当前位置，千分比 int16 LE
2:3	当前位置，mm×10 int16 LE
4	状态位，见下表
5	转向故障码（ `steer_fault_t` ）
6	电机电流 Iq，A×10 有符号 —— 上位机据此看路面负载 / 有没有卡住
7	序号回显

字节 4 的位：

位	含义
0	已标定
1	已使能
2	到位
3	在左（负向）软限位
4	在右（正向）软限位
5	**转向**故障（详见字节 5）
6	助力模式（本版编译关闭）
7	⚠⚠ **驱动侧**锁存了故障：过流 / DRV nFAULT / 欠压过压 / 电流采样不可信

⚠⚠ **位 5 和位 7 是两种不同的失效，答案也不同：** - 位 5：轴**不相信自己的位置** - 位 7：**桥关了，齿条根本不会动**

⇒ 位 7 不是可有可无的。没有它，一次锁存的过流会让这一帧的每个字段都显示健康： 故障位是 0、故障码是 0、位置照报（传感器是好的，关掉的是电机）—— **整车控制器最需要知道的那件事，恰好是这一帧说不出来的那件事。**

6.3 失联

超过 `st_timeout_ms` 没有新指令，执行器**保持当前位置**（不是回中、不是断电）。

7. 保护与故障

7.1 驱动侧（`prot` / `fault` / `clear`）

位	名字	含义
0	`OV`	过压
1	`UV`	欠压
2	`OC`	软件过流（20 kHz ISR 判）
3	`NFAULT`	DRV8323 报故障（经 EXTI 立即切栅极）
4	`IMEAS`	三相电流残差不合理 —— 采样链本身有问题

- 过压/欠压门限**随上电检测到的电池节数自动缩放**，另有绝对上下限兜底。
- I^2t 发热模型按 `iq_cont_ma` 折返连续电流；它和 `iq_max_ma` 是两回事。
- ⚠ **切桥一律锁存**，要 `clear` 才解除，而且原因还在的时候清不掉。

⚠ **换完电池、或者主电断过一次之后，敲一次 `clear`。** 调试时 USB 供着逻辑电、主电断开再回来，栅极驱动会掉电复位。 本版在 `clear` 和放电流之前**各查一次**栅极驱动的配置， 所以这条比早期版本可靠 —— 但先敲一次 `clear` 仍然是对的。

7.2 转向侧（`steer` 的 `fault=` 字段 / CAN `0x231[5]`）

码	名字	含义	出路
0	`NONE`	没有故障	—
1	`UNCAL`	没有有效标定 ⇒ 拒绝驱动	跑 `prodcal run`
2	`SENSOR`	原始读数超出合理带（开路 / 短路）	查编码器接线，然后 `steer clear`
3	`JUMP`	位置跳变不合理（翻圈或接触不良）	同上
4	`OVERTRAVEL`	超出软限位 + 硬余量	`steer recover`（§5.3）
5	`DIR`	齿条朝指令的反方向走 —— 反馈或电机极性反了	重跑 `prodcal sweep`（它会探方向并翻符号），然后 `save`

⚠ `DIR` 是正反馈的症状：位置环会朝**误差变大**的方向推，一路推到越程锁存。 所以三处都在拦它：`prodcal sweep` 起步先探 3 mm，反了在 2 mm 内停；`steer recover` 回程往外 2 mm 就停； `prodcal` 的 `dir` 门要一次至少 20 mm/s 的实际移动作证据，"没动"不算合格。 方向符号不要手填——0.1.7 起它是探出来的，手填错过一次把齿条撞出了软限位。

7.3 固件做不到的那一件

⚠⚠ **这台执行器没有抱闸，也没有自锁机构。** 停机之后齿条是自由的， 断电时人用手就能推动它。整车侧如果需要"断电保持转向角"，那是机械件的事， 固件给不了。

8. 参数

8.1 板子自己是参数表的真源

```
param list                    整张表：id / 名字 / 单位 / 范围 / 默认 / 权限 / 当前值
param get st_travel_um
param set st_travel_um 50000
param save                    写进 flash（等价于 save）
```

⚠ **不要把范围和默认值抄进别的文档。** 它们跟着固件走，抄本第二天就是错的， 而且抄本看起来完全正常 —— 一个合理的数字，在一个合理的字段里。

⚠ `param list` 的回复带 `begin` / `count=` / `end` 三件套。 **数出来不等于 `count` 就把整份丢掉** —— 一张被心跳打断的四十行表， 每一行仍然格式完美，而少掉的那一行看起来就像"这个参数不存在"。

8.2 三条使用规矩

1. **桥活着的时候写不进去。** 几何、标定、方向这些行在电机使能时直接拒绝 （回 `wrong-state` ） —— 在一个转着的电机底下改增益，没有人看得见它落地的那一刻。
2. `param set` **回的是板子事后真正持有的值**，不是 OK，也不是把你发的值抄回来。 被限幅了、或者存不住，回复里会同时带值和拒绝码。
3. **改完要存**： `save` 。⚠ 有几行**故意不进 flash**（见下）。

8.3 两类特殊的行

类	例子	为什么
只读的测量结果	`st_cal_min_cnt` / `st_cal_ctr_cnt` / `st_cal_max_cnt` / `st_cal_travel_um`	它们是一次测量的结果。一个能填的框会让人去调答案，而不是重做测量
故意不持久化	`ibus_limit_ma` / `st_timeout_ms` / `dt_comp_u` / `delay_comp_m`	它们描述**这次台架**，不描述这台执行器。掉电回到量产值 —— 会被人忘掉的那一个是临时的那一个

⚠ 方向符号 `st_invert` 能通过 `param set` 写，写了也算"定过了"（0.1.10 起）； 只读行 `st_inv_set` 说的是它到底定没定——备份/还原一台机的参数时，先看这一行。

⚠⚠ `ibus_limit_ma` 尤其要注意：接限流电源时敲 `ibus 1700` 压低， **跑完不用记得改回来**，掉电自己就回去了。反过来做（把台架值存进 flash） 会让一台 100 W 的电机每次上电只拿到 41 W，而它表现得完全正常。

8.4 ⚠ 错了没有任何症状的那一组

几何参数（齿数、模数、减速比、伞齿比、编码器速比、编码器满量程、极对数、 额定转速）**填错了执行器照样动、读数照样合理，只有卷尺不同意。** `param list` 会在这些行上带 `silent=1` 。

⇒ 界面上要把它们标出来。判据不是"重不重要"，是"**错了会不会有人发现**"。

---

9. 给程序用的接口

9.1 机器可读的记录

记录	什么时候出	内容
`=tlm 1 build …`	`ver`	版本、提交、板型、硬件版本、Debug 与否、 `cmdhash`
`=tlm 1 persist …`	启动 / `ver` ，仅当发生过迁移	标定页从旧地址迁过来了，**下一次** `save` **之前它靠的是未声明的硅片**

记录	什么时候出	内容
`=param …`	`param list` / `get` / `set`	见 §8.1，带 `begin` / `count` / `end`
`=prodcal …`	`prodcal`	见 §3.2，带 `begin` / `count` / `end`
`=hall …`	`hall dump`	学到的霍尔角度表一行备份（24 位十六进制 + 8 位键，键绑本板 UID），`hall set <blob> <key>` 放回去再 `save`。给"升级擦掉了标定页、不想再转轴重学"用（0.1.9 起）

⚠⚠ **本版没有周期性的 `=tlm` 遥测记录。** `mon` 打开的是一条给人看的 4 Hz 文字流（母线电压 / 三相原始值 / 霍尔），**不是结构化记录。** ⇒ 上位机要连续位置就轮询 `steer`，或者收 CAN `0x231`（20 ms 一帧，而且不占串口）。**`0x231` 是这台执行器上唯一一条真正的周期性机器接口。**

9.2 命令表 `COMMANDS.json`

发布包里带一份 `COMMANDS.json`：每条命令的名字、参数形状、作用域、前置条件（`needs`）和危险等级（`danger`）。

⚠⚠ **用它之前先比 `cmdhash`**：板子 `ver` 报的和文件里的必须相等。不等就整份不要用 —— 它们是把这份包和这台执行器绑在一起的**唯一**证据，版本号不是，提交号也不是。

---

10. 固件升级

10.1 这台执行器没有应用 Bootloader

发布包里只有应用镜像，没有 loader。升级有两条路：

路	怎么走	说明
串口 ISP	USB Type-C，芯片自带的系统 Bootloader	板上有双三极管一键下载电路，自动控制 BOOT0 + NRST，免按键
SWD	ST-Link	产线和研发用

⚠ **ST-Link 烧录或复位时，不要开着占用同一个串口的工具。** 若串口工具在复位瞬间同时拉低 DTR + RTS，BOOT0 可能被采样为高而进入 DFU。

10.2 ⚠⚠ 烧完必须完整断电重启

见 §3.4.4。烧完之后不断电就跑，`prodcal` 的 `enc` 那道门不作数。

10.3 ⚠ 升级之后重新标定

2026-09-08 定的升级路径：**升级之后让客户重新走一遍 `prodcal`。** 标定页的位置在 0.1.0 里挪过（从 `0x0803F800` 到 `0x0801F800`），旧固件写下的标定不保证能被新固件正确读出。

⇒ 升级后第一件事：`prodcal`，看 `verdict`。

⚠ **从 0.1.8 及更早升上来的执行器，`rack` 那道门会报 `fail`，`need=push-the-rack-to-mechanical-centre-then-prodcal-run`。** 这不是坏了：0.1.9 起正方向按产品约定定死（安装口朝人，齿条向右为正），而旧版标定记录里"正"指向另一端。把齿条手推到机械中点，再 `prodcal run`，它会自己重标；方向符号 `st_invert` 也会重新探出来，结果和旧版相反是正常的。

10.4 ⚠ 默认构建是 Debug

`scripts/flash.sh` 不带参数烧的是 `Debug/`。**上位机烧录必须显式选 Release**，不要靠默认。判据不是文件名 —— 是板子 `ver` 报的 `debug=` 那一位。

---

11. 版本历史

11.1 固件

版本	日期	变化
0.1.10	2026-09-10	`param set st_invert` 也算方向符号已定；只读行 `st_inv_set`
0.1.9	2026-09-10	**正方向按产品约定定死**（安装口朝人、齿条向右为正）；旧标定记录在 `rack` 门报 fail，`prodcal run` 自己重标；`hall dump` / `hall set`；`steer` 那行 `ch=` 改成 `src=`
0.1.8	2026-09-10	`steer recover` 回程自己盯位移，往外 2 mm 停轴锁 `DIR`；方向符号没定过拒绝回程
0.1.7	2026-09-10	`prodcal sweep` 两端停在软限位前 5 mm；`geom` 门带 `gear_set`，减速比没填 `run` / `sweep` 拒绝；`steermot gear` 自动填额定转速
0.1.6	2026-09-10	编码器开机时没插、插好之后读数不再冻在旧值
0.1.5	2026-09-10	`prodcal sweep` 起步先探 3 mm 方向，反了 2 mm 内锁故障停轴；方向符号没定过拒绝扫描
0.1.4	2026-09-10	扫描结束控制台自己说一声；`param set` 回整行；清单带 `name:`；发布包带 `source.zip`
0.1.3	2026-09-10	没整备的板 `prodcal` 输出尾巴不再丢（按发送环空位发）
0.1.2	2026-09-10	扫描每周期刷新目标；`dir` 门不再把"没动"判成合格；记下方向符号有没有人定过
0.1.1	2026-09-09	标定加减速比这一步；全行程验证改成机器做（`prodcal sweep`，自己武装轴）；`geom` 门（减速比 × 额定转速 ≈ 4000）；`wrap` 按实测止点 55 mm 算；`MANIFEST` 和另外两块板同一行式
0.1.0	2026-09-08	首个带完整出厂验收的版本。`prodcal` 十二道门；栅极驱动掉电丢配置已修（`clear` 与放电流前各查一次）；编码器 MOSI strap 提到复位后第一条 C 语句；出厂限值按电机铭牌定死；机械止点实测 ±55 mm；标定页从 `0x0803F800` 挪到 `0x0801F800`；CAN 清故障改成按清故障位本身的边沿触发；`ENABLE|CLR_FAULT` 同帧不再重新上电；`0x231` 增加驱动侧故障位；参数表的默认值改为从头文件推导（`st_pole_pairs` 曾经写着 4，而板子按 2 启动）

11.2 本书

版本	日期	变化
Rev. 1.1	2026-09-10	跟上固件 0.1.1–0.1.10：十二道门、`prodcal sweep`、方向符号由探测定、正方向产品约定、升级后重标的样子、`hall dump/set`、`steer recover` 盯位移；§13 加"名字会误导的几条"和装车方向约定
Rev. 1.0	2026-09-08	首版。此前这个产品**没有出厂固件说明书**

12. 已知限制（固件侧）

#	限制	影响
1	~~`wrap` 那道门的判据偏乐观~~ 0.1.1 起按实测止点 55 mm 算	—
2	**没有周期性 `=tlm` 遥测记录**	上位机连续读位置要轮询 `steer` 或收 CAN `0x231`

#	限制	影响
3	**齿条推力没有实测过**	规格书上的推力是按齿轮几何和电机铭牌算的；真正的约束是固件的电流上限
4	**停止包络用的减速度只量过纯滑行**（约 260 mm/s²）	开主动制动之后能到多少没有量过
5	**没有抱闸 / 自锁**	停机和断电时齿条自由，可被外力推动
6	**Debug 构建的 CPU 占用没有在这台执行器上量过**	不要引用驱动板的那个数
7	**助力模式编译关闭**（`steerasst`）	命令在，功能不在

13. 附录：命令一览

⚠ 这张表是给人快速查的。**权威的一份是随包** `COMMANDS.json`， 它带参数形状、前置条件和危险等级，而且能和板子的 `cmdhash` 对上。

只读（`none`）

`help` `ver` `iq` `vbus` `iabc` `hspd` `isr` `clk` `thr` `pot` `din` `enc` `hmap` `prot` `therm` `fault` `drv` `scst` `scope` `txdrop` `steerraw` `adcscan`

改状态 / 改设置，但转不动（`gates`）

`stop` `clear` `save` `cal` `hseen` `mon` `scarm` `pi` `spi` `cloff` `hcorr` `haff` `hnavg` `dcomp` `dtcomp` `iqslew` `ibus` `ovset` `ocset` `steerpid` `steertmo` `steerch` `steerphs` `buzz` `can` `hall`（不带参数只读； `hall dump` 备份霍尔表、 `hall set` 还原，轴使能时拒绝）

可能让机器动（`motion`）

`pwm` `drive` `mode` `learn` `calib` `rl` `calibol` `cl` `clc` `spd` `idq` `if` `spin` `brake` `iqmax` `srcmap` `simthr` `thrcal` `thrmin` `thrmax` `potcal` `potmin` `potmax` `ascpu` `side` `param` `steer` `steermm` `steercal` `steerlim` `steermot` `encmeas` `steerasst` `prodcal`

⚠ 名字会误导的几条

这套固件和单轮驱动固件 FOC_single 是同一块板、同一套核心代码， 所以命令表里有几条是**从驱动轮那边带过来的，在执行器上没有作用。** 它们能敲、会应答，但不改变执行器的任何行为：

命令	在驱动轮固件上是什么	在执行器上
`side left|right`	**车轮身份**：这块板驱动的是左轮还是右轮，决定它用哪一对 CAN 状态帧 ID（左 `0x211/0x212`，右 `0x213/0x214`）	**无作用。** 执行器只发 `0x231`，不分左右；装在车上哪一侧、"正方向"对应车轮往哪边转，**不是用这条命令定的**
`pot` `potcal` `potmin` `potmax` `steerch` `steerraw` `adcscan`	电位器 / 模拟油门那一代传感器的读数和标定	**无作用。** 这台执行器的位置来自 SPI 绝对编码器（`enc`）， `steer` 那行的 `src=spi-encoder` 就是在说这件事

> **正方向的产品约定（2026-09-10 定）：以电机安装口朝向自己为参照，齿条朝右移动为正。** "正"就是位置数值变大、 `prodcal sweep` 第一段走的方向。这是这个型号的产品事实， 同款编码器和机械每台一样；每台不同的只有电机接线的方向符号，那个由出厂标定 （`prodcal`）自己探出来，**不问**操作员"车轮是不是向右"。 **装车默认：执行器正方向 = 车辆右转。** 这套转向系统的安装方式决定了绝大多数 情况下方向是固定的，装车流程里没有"设左右"这一步。只有极少数改变用途、装反了的 情况才需要反过来，那时在**控制板**上改一个参数（转向正负号），执行器这边什么都不改。

常用的十条

命令	做什么
`ver`	这是哪一版固件、哪块板、`cmdhash` 多少
`prodcal`	十二道出厂门现在过了几道
`steer`	齿条在哪、标定过没有、有没有故障
`steermm <mm×10>`	命令一个目标位置
`steertmo <ms>`	临时放大指令过期时间（台架用）
`steermot gear <10|15|20>`	填减速比（额定转速跟着自动填）
`prodcal sweep`	探方向 + 全行程扫描，会推齿条
`hall dump`	一行备份学到的霍尔表，升级后 `hall set` 放回去
`enc`	直接读一次绝对编码器
`prot`	保护状态和门限
`clear`	清一条锁存的驱动故障
`stop`	立刻关栅极

Daliang Auto · DlaSteer-100 出厂固件说明书 Rev. 1.1 · 2026-09-10