1. **Instructions for use of the motor**

1.1 Driver Board Information

- Operating voltage: 12V-38V
- Dual encoder: integrated dual encoder at the bottom
- Communication interface: supports CAN and CAN-FD protocols
- Power indicator

1.2 Hardware wiring of the motor

1. The motor CAN-FD interface is connected to the FDCAN module.
2. Connect the power supply, the indicator light is on when the power supply is powered on indicating that the driver board is working normally.

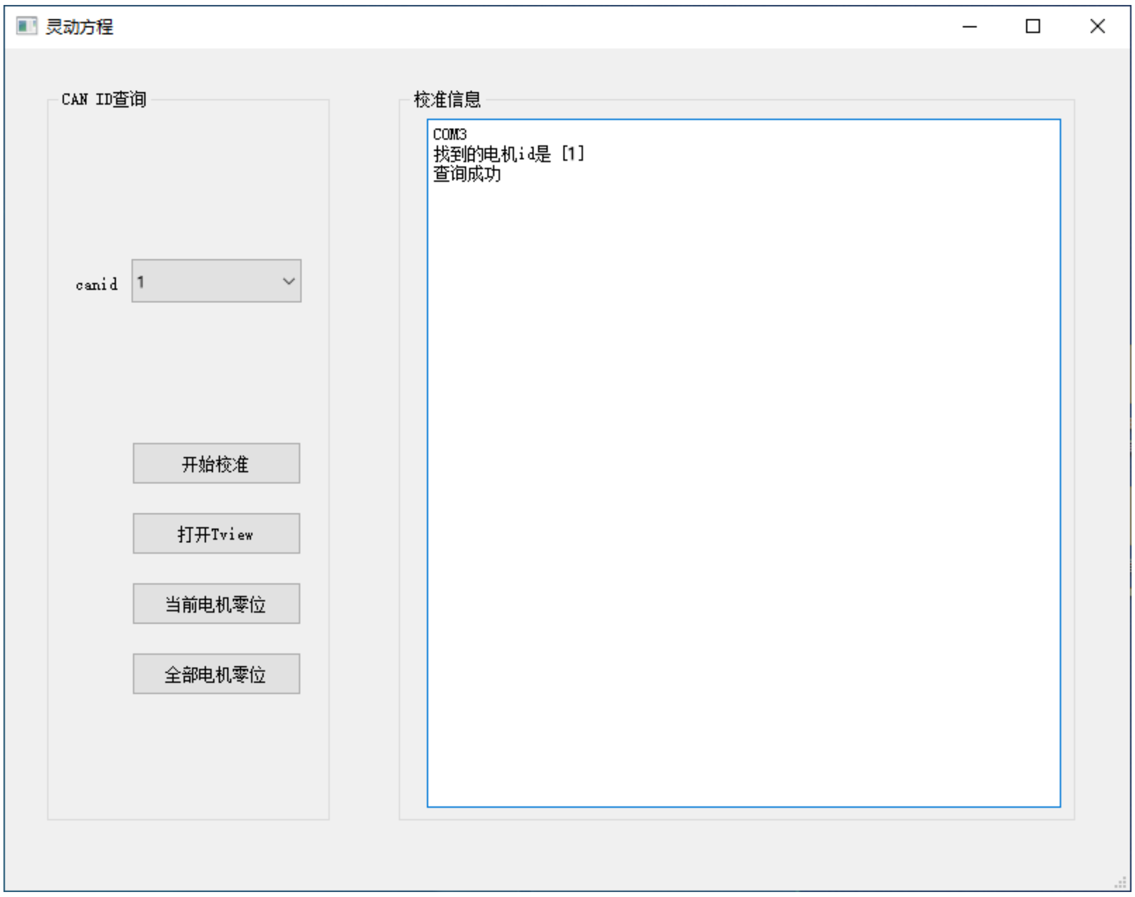
Note: Please make sure that the motor can cable wiring sequence is correct with the FDCAN module wiring sequence, the can module light will blink when the communication with the upper computer is turned on.

2. Relevant methods of use

2.1 Motor Calibration Process

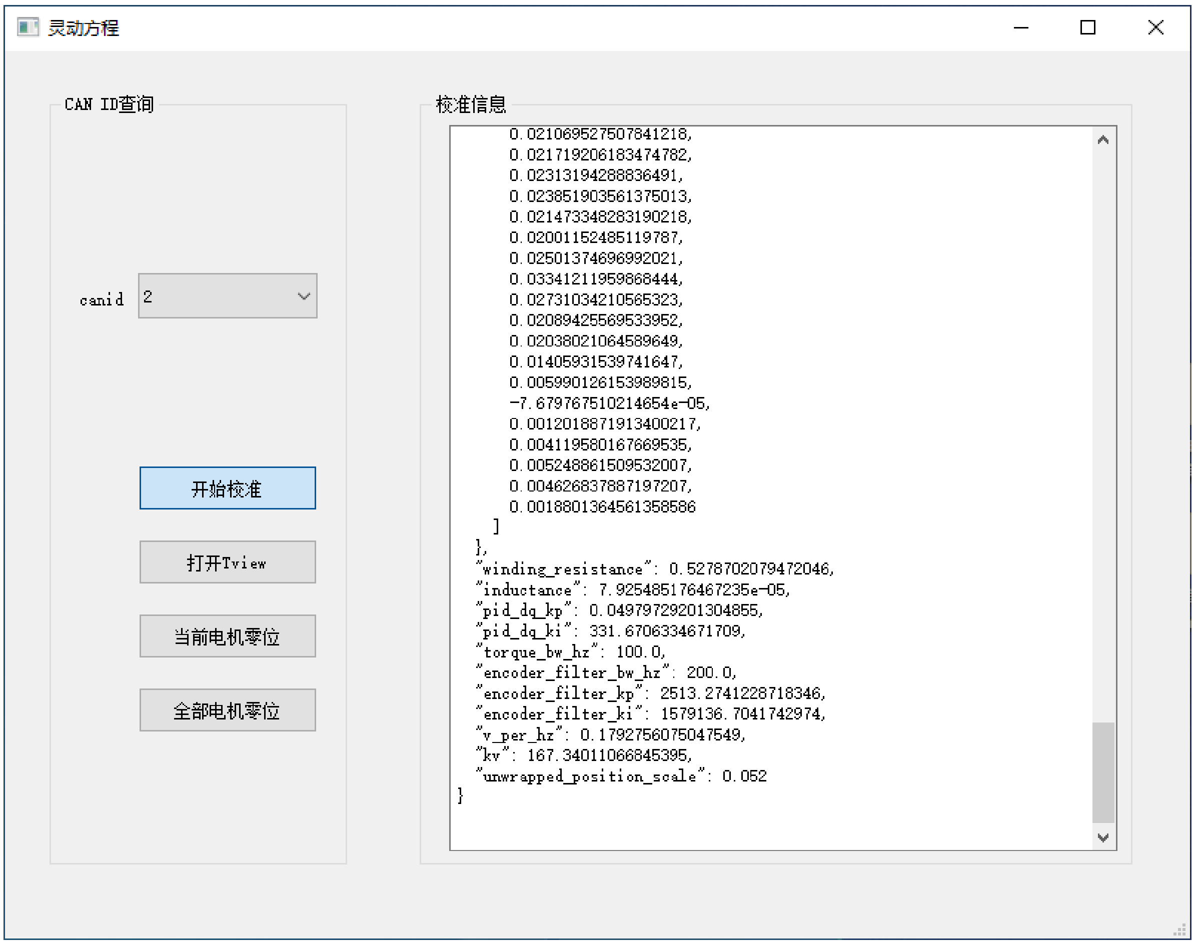
1. After the hardware wiring is correct, open the upper computer software, the upper computer will automatically identify the motor ID, the default initial ID is 1, click start

calibration. (The motor from the factory is already calibrated)



```
COM3
找到的电机id是 [1]
查询成功
```

2. Wait for calibration, the upper computer will output calibration information, the successful calibration interface is shown in the following figure.



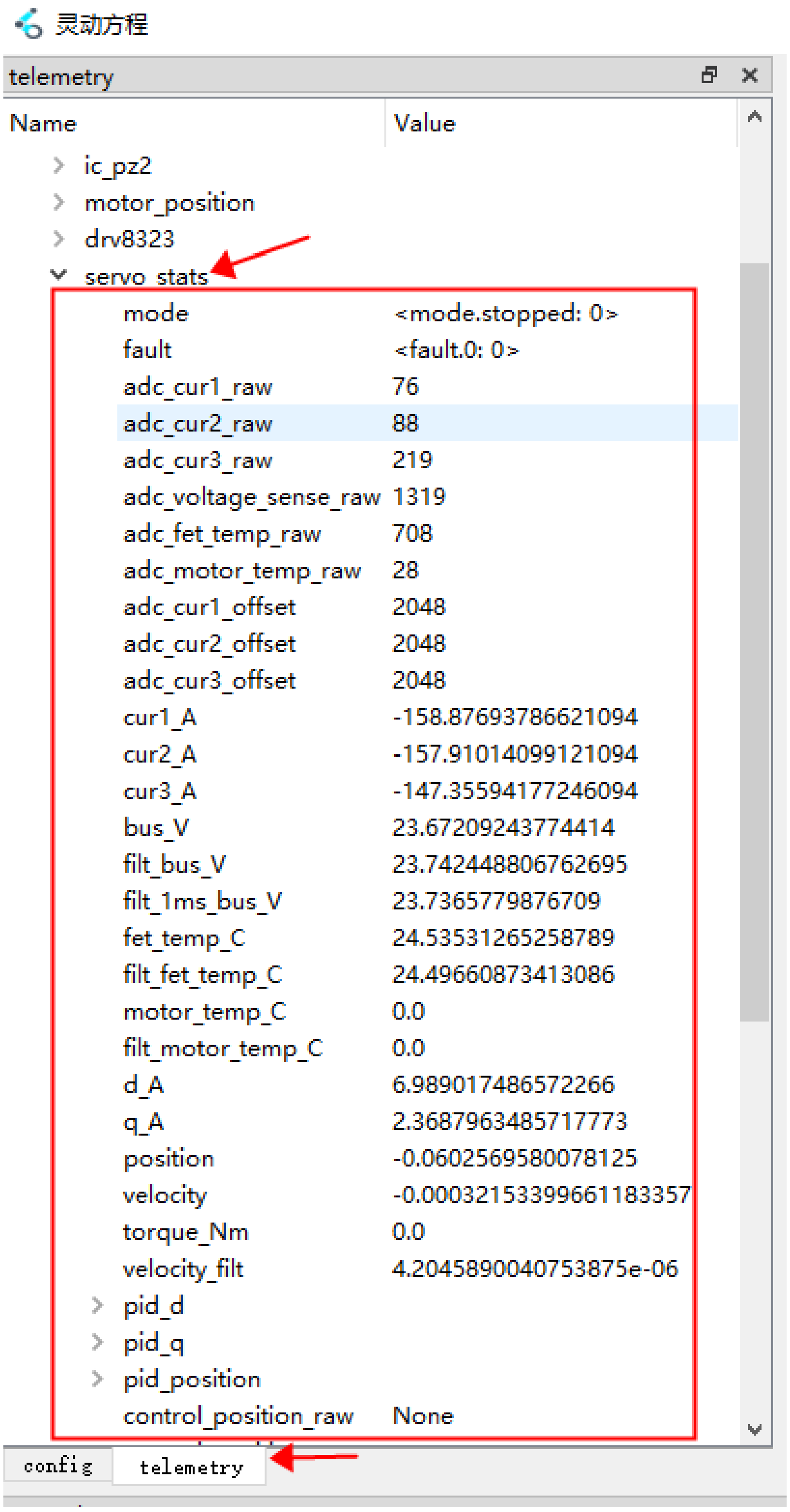
```
        0.0210695275078412l8,
        0.021719206183474782,
        0.02313194288836491,
        0.023851903561375013,
        0.021473348283190218,
        0.02001152485119787,
        0.02501374696992021,
        0.03341211959868444,
        0.02731034210565323,
        0.02089425569533952,
        0.02038021064589649,
        0.01405931539741647,
        0.005990126153989815,
        -7.679767510214654e-05,
        0.0012018871913400217,
        0.004119580167669535,
        0.005248861509532007,
        0.004626837887197207,
        0.0018801364561358586
      ]
    },
    "winding_resistance": 0.5278702079472046,
    "inductance": 7.925485176467235e-05,
    "pid_dq_kp": 0.04979729201304855,
    "pid_dq_ki": 331.6706334671709,
    "torque_bw_hz": 100.0,
    "encoder_filter_bw_hz": 200.0,
    "encoder_filter_kp": 2513.274122871834б,
    "encoder_filter_ki": 1579136.7041742974,
    "v_per_hz": 0.1792756075047549,
    "kv": 167.3401106684539б,
    "unwrapped_position_scale": 0.052
}
```

2.2 Motor Parameter Status View

State information view: After successful calibration, open the host computer and click "**Open Tvie**", inside the **telemetry** item, click on the **servo_stats** tab, which can view the current status of the motor, etc., as shown in the figure below.

Example:

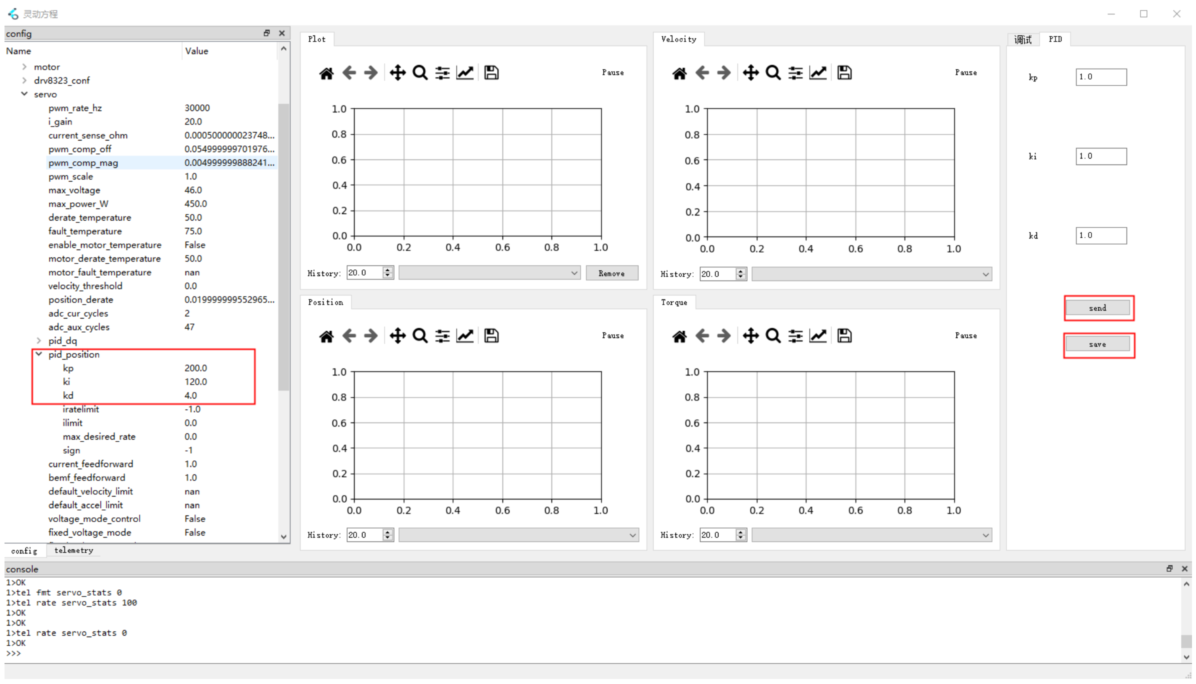
- Position information: Inside **telemetry**, tap **servo_stats**, look for **position**, right click and select **plot right/left**.
- Velocity information: Inside **telemetry**, click on **servo_stats**, look for **velocity**, right click and select **plot right/left**.
- Torque info: inside **telemetry**, tap **servo_stats**,look for **torque_Nm**, right click and select **plot right/left**.



Name	Value
› ic_pz2	
› motor_position	
› drv8323	
⌄ servo_stats	
mode	<mode.stopped: 0>
fault	<fault.0: 0>
adc_cur1_raw	76
adc_cur2_raw	88
adc_cur3_raw	219
adc_voltage_sense_raw	1319
adc_fet_temp_raw	708
adc_motor_temp_raw	28
adc_cur1_offset	2048
adc_cur2_offset	2048
adc_cur3_offset	2048
cur1_A	-158.87693786621094
cur2_A	-157.91014099121094
cur3_A	-147.35594177246094
bus_V	23.67209243774414
filt_bus_V	23.742448806762695
filt_1ms_bus_V	23.7365779876709
fet_temp_C	24.53531265258789
filt_fet_temp_C	24.49660873413086
motor_temp_C	0.0
filt_motor_temp_C	0.0
d_A	6.989017486572266
q_A	2.3687963485717773
position	-0.0602569580078125
velocity	-0.00032153399661183357
torque_Nm	0.0
velocity_filt	4.2045890040753875e-06
› pid_d	
› pid_q	
› pid_position	
control_position_raw	None

2.3 PID Tuning

After configuring the parameters in the config item, you need to save them to take effect :command **conf write** or set the parameters in the PID parameter on the right hand side and then click **Send** and **save in order** to save them to take effect, as shown in the following figure.



Name	Value
motor	
drv8323_conf	
servo	
pwm_rate_hz	30000
i_gain	20.0
current_sense_ohm	0.000500000023748...
pwm_comp_off	0.054999999701976...
pwm_comp_mag	0.004999999888241...
pwm_scale	1.0
max_voltage	46.0
max_power_W	450.0
derate_temperature	50.0
fault_temperature	75.0
enable_motor_temperature	False
motor_derate_temperature	50.0
motor_fault_temperature	nan
velocity_threshold	0.0
position_derate	0.019999999552965...
adc_cur_cycles	2
adc_aux_cycles	47
pid_dq	
pid_position	
kp	200.0
ki	120.0
kd	4.0
iratelimit	-1.0
ilimit	0.0
max_desired_rate	0.0
sign	-1
current_feedforward	1.0
bemf_feedforward	1.0
default_velocity_limit	nan
default_accel_limit	nan
voltage_mode_control	False
fixed_voltage_mode	False

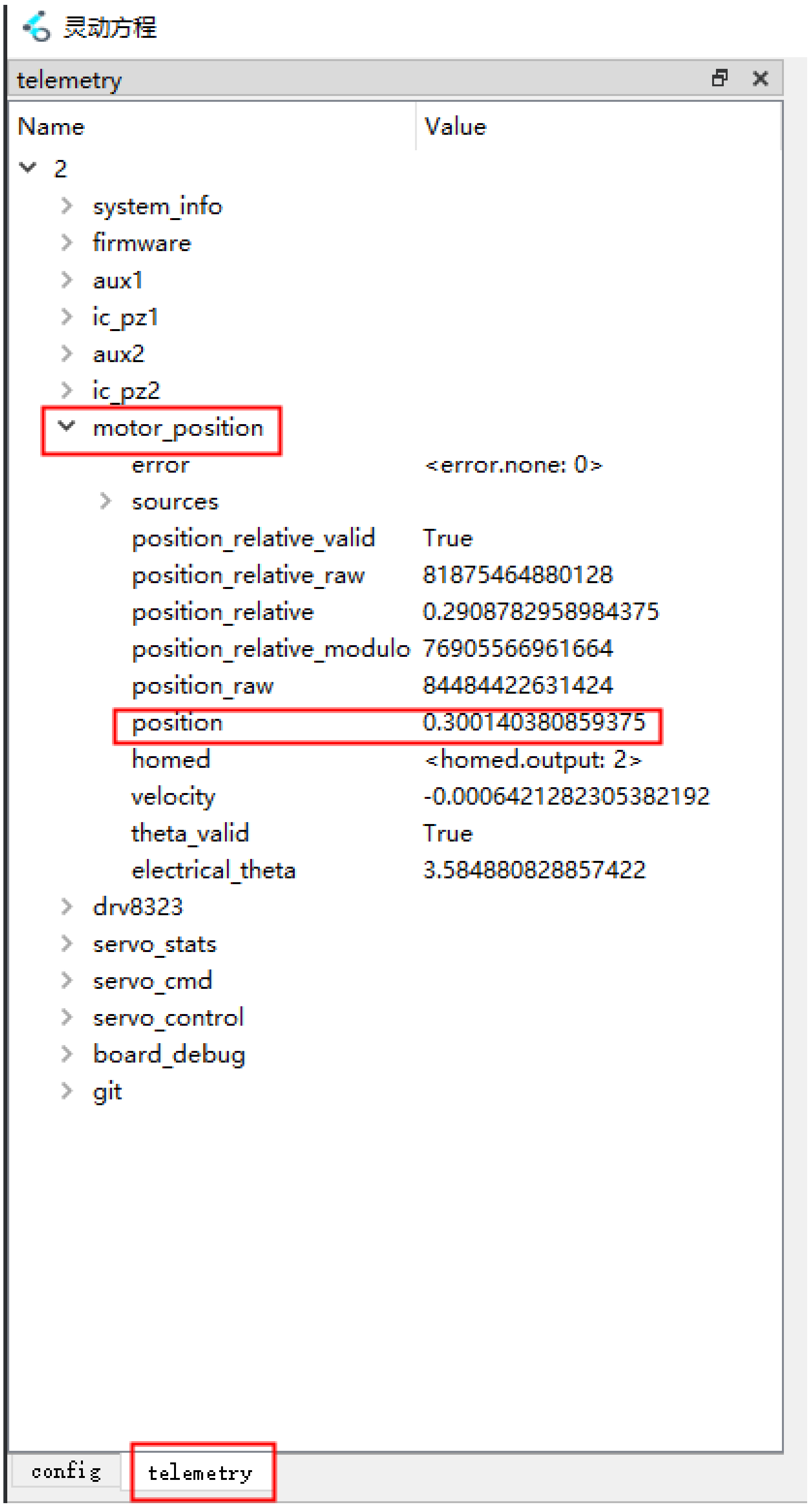
```
1>OK
1>tel fmt servo_stats 0
1>tel rate servo_stats 100
1>OK
1>OK
1>tel rate servo_stats 0
1>OK
>>>
```

2.3.1 PID speed parameterization process

1. Call up the speed profile.
2. **d pos (d pos nan 0.6 2)** command, so that the motor first move up, it is recommended that the speed first set a little smaller can first give 0.6, observe the motor speed curve and the target curve gap.
3. Start tuning the PID and watch the speed profile change.
4. Input the command d stop to make the motor stop, then re-input the command d pos to make the motor rotate,observe the difference between the motor speed curve and the target curve,and keep repeating step 3 until it is optimal.
5. After determining the final parameters, you need to enter the save command **conf write** to save the parameters.

2.4 Motor zero point

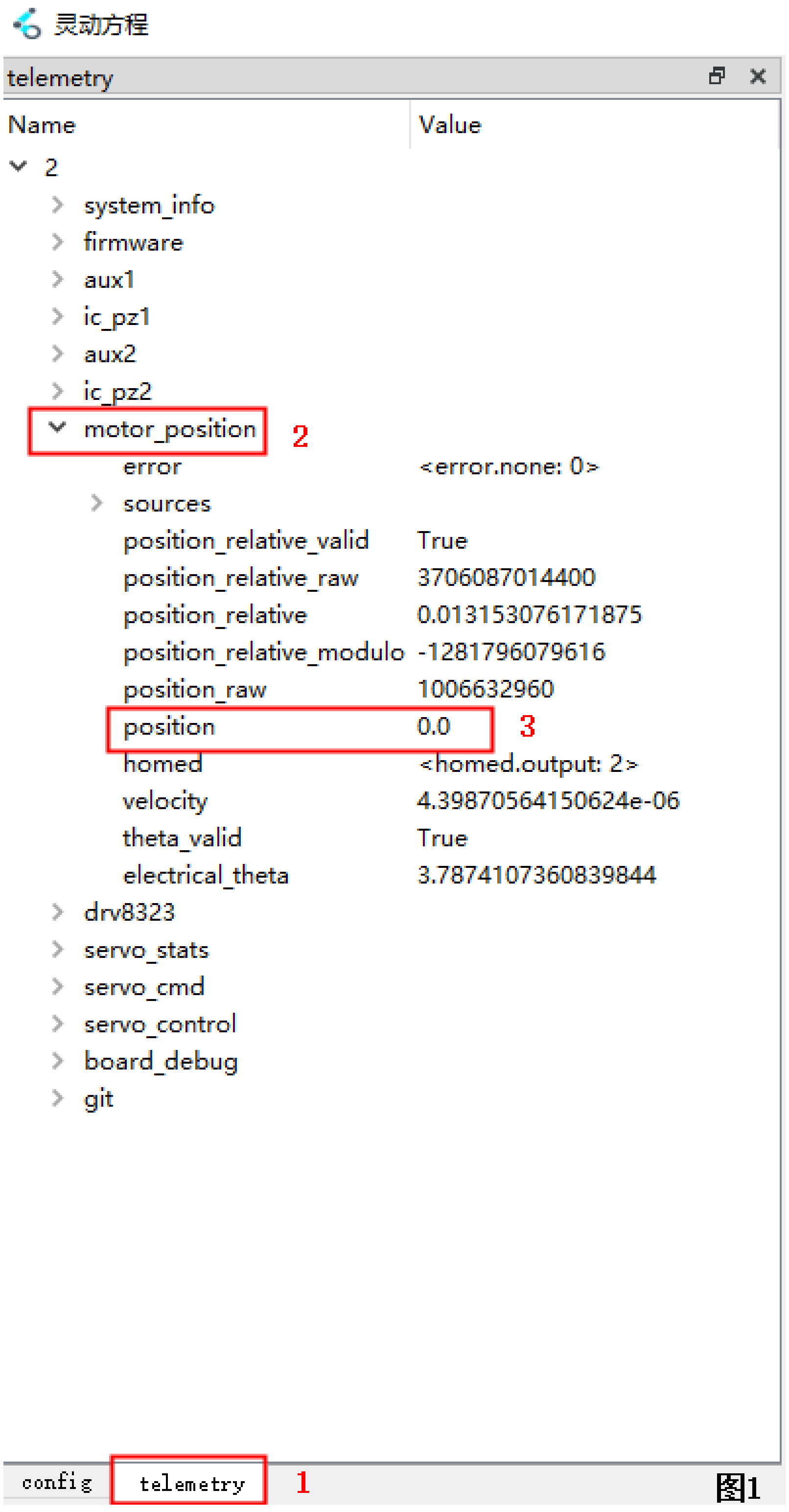
The motor zero recorded position is -0.5-+0.5 turns, the position beyond this range will be counted as a single turn position after re-powering up. The current position value can be viewed in the upper computer **motor_position.position**, the current position value is the position value of the output encoder, as shown in the following figure. To test the zero point of the motor, you can twist the motor to a position of -0.5 to +0.5, and then re-power on the motor after power failure to see if the position is the position before power failure. You can also record the position before the power failure, then power off, twist the motor, re-power on, twist back to the position recorded before the power failure to observe whether it is the same.



```
Name                          Value
2
  system_info
  firmware
  aux1
  ic_pz1
  aux2
  ic_pz2
  motor_position
    error                     <error.none: 0>
    sources
    position_relative_valid   True
    position_relative_raw     81875464880128
    position_relative         0.2908782958984375
    position_relative_modulo  76905566961664
    position_raw              84484422631424
    position                  0.300140380859375
    homed                     <homed.output: 2>
    velocity                  -0.0006421282305382192
    theta_valid               True
    electrical_theta          3.584880828857422
  drv8323
  servo_stats
  servo_cmd
  servo_control
  board_debug
  git
```

2.5 Reset zero position

Motor calibration will have an absolute zero position, open the host computer in **telemetry** items, motor **motor_position** tab, the parameter **position** for the current position of the motor, as shown in Figure 1 below, the position parameter is 0, that is, the zero position of the motor rotation. If you want to specify the motor rotation position for the zero position, you can click on the host computer "**all motor zero position**" can be, as shown in Figure 2 below, at this time the motor zero position change.

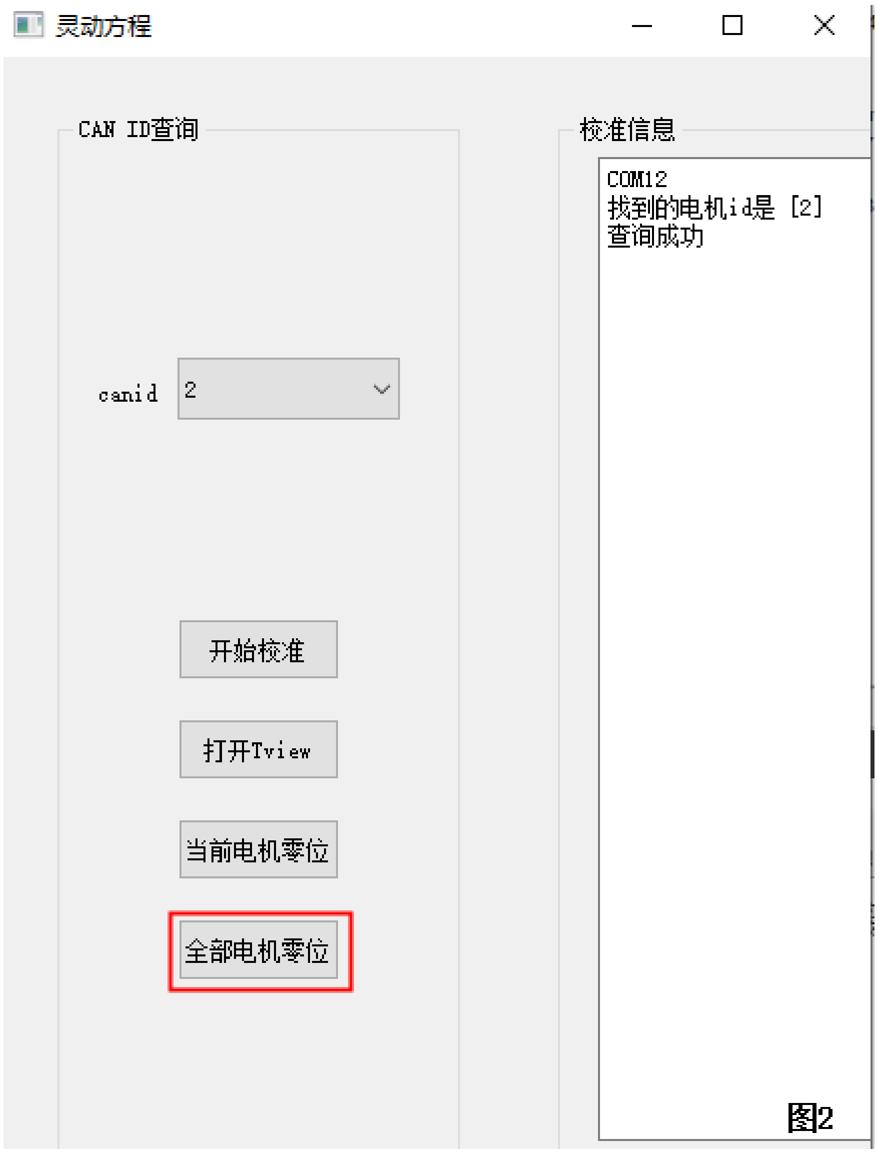


灵动方程

telemetry

Name	Value
2	
system_info	
firmware	
aux1	
ic_pz1	
aux2	
ic_pz2	
motor_position	
error	<error.none: 0>
sources	
position_relative_valid	True
position_relative_raw	3706087014400
position_relative	0.013153076171875
position_relative_modulo	-1281796079616
position_raw	1006632960
position	0.0
homed	<homed.output: 2>
velocity	4.39870564150624e-06
theta_valid	True
electrical_theta	3.7874107360839844
drv8323	
servo_stats	
servo_cmd	
servo_control	
board_debug	
git	

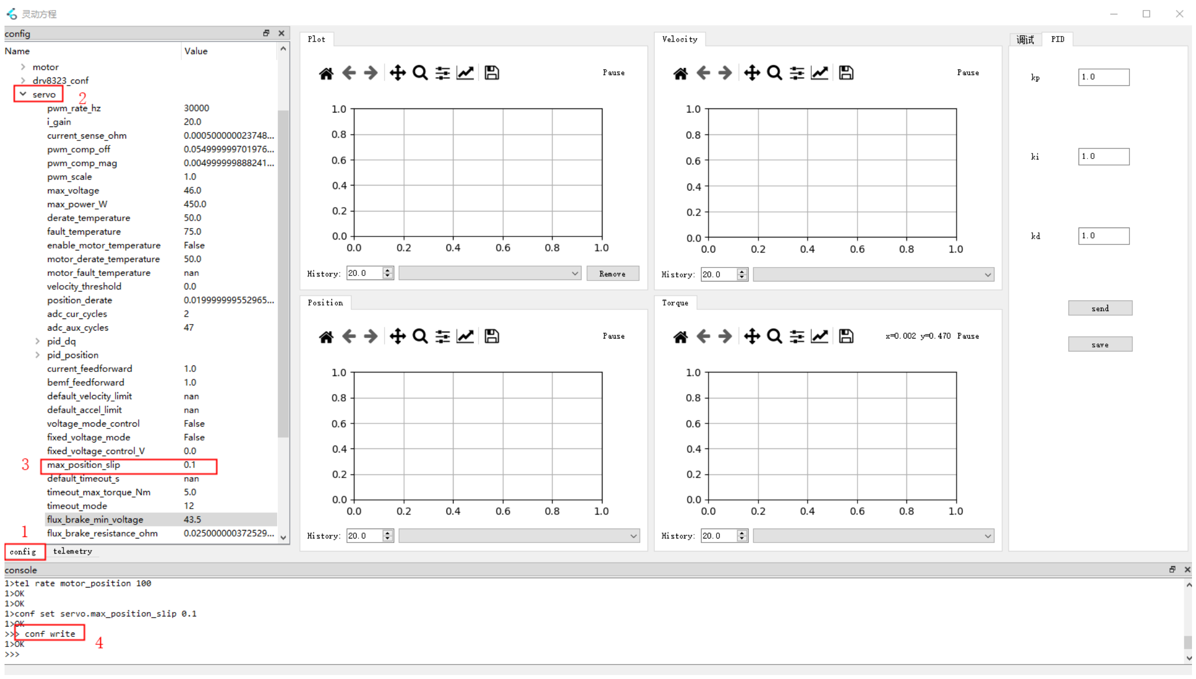
图1



```
COM12
找到的电机id是 [2]
查询成功
```

2.6 Position integral

To set the position integral of the motor, you can find the parameter **max_position_slip in the config item-servo** tab of the host computer, change the original parameter **nan** to **0.1**, after that, enter **conf write** in the command line to save the instruction, and the host computer will return to OK, as shown in the following figure. Modify this parameter is to prevent the motor in the use of speed control by the external force card stop, withdraw the external force will be quickly rotated without the previous speed control.



config

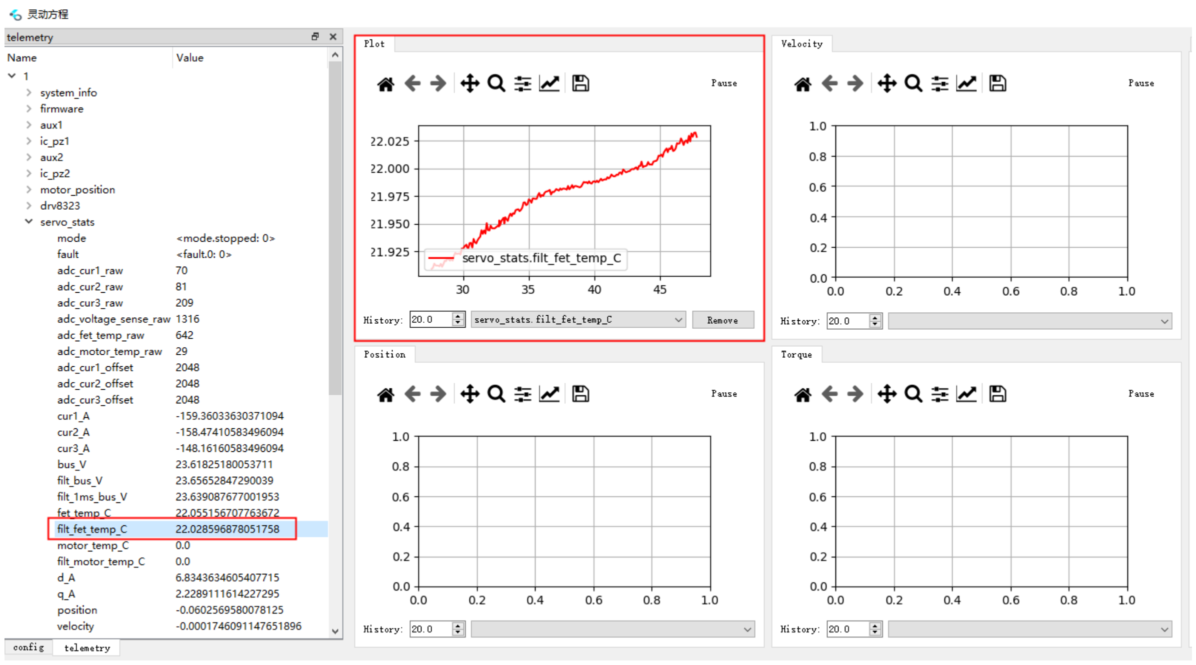
Name	Value
motor	
drv8323_conf	
servo	
pwm_rate_hz	30000
i_gain	20.0
current_sense_ohm	0.000500000023748...
pwm_comp_off	0.054999999701976...
pwm_comp_mag	0.004999999888241...
pwm_scale	1.0
max_voltage	46.0
max_power_W	450.0
derate_temperature	50.0
fault_temperature	75.0
enable_motor_temperature	False
motor_derate_temperature	50.0
motor_fault_temperature	nan
velocity_threshold	0.0
position_derate	0.019999999552965...
adc_cur_cycles	2
adc_aux_cycles	47
pid_dq	
pid_position	
current_feedforward	1.0
bemf_feedforward	1.0
default_velocity_limit	nan
default_accel_limit	nan
voltage_mode_control	False
fixed_voltage_mode	False
fixed_voltage_control_V	0.0
max_position_slip	0.1
default_timeout_s	nan
timeout_max_torque_Nm	5.0
timeout_mode	12
flux_brake_min_voltage	43.5
flux_brake_resistance_ohm	0.025000000372529...

console

```
1>tel rate motor_position 100
1>OK
1>OK
1>conf set servo.max_position_slip 0.1
1>OK
>>> conf write
1>OK
>>>
```

2.7 Temperature Limit Setting

The drive plate may limit the torque output due to high temperatures, which can be improved by physical and software means. To view the visualization of the temperature profile, inside the **telemetry** item, open the **servo_stats** tab, find the parameter **filt_fet_temp_C**, right-click and select **plot right/left**, as shown in the figure below.



telemetry

Name	Value
1	
system_info	
firmware	
aux1	
ic_pz1	
aux2	
ic_pz2	
motor_position	
drv8323	
servo_stats	
mode	<mode.stopped: 0>
fault	<fault.0: 0>
adc_cur1_raw	70
adc_cur2_raw	81
adc_cur3_raw	209
adc_voltage_sense_raw	1316
adc_fet_temp_raw	642
adc_motor_temp_raw	29
adc_cur1_offset	2048
adc_cur2_offset	2048
adc_cur3_offset	2048
cur1_A	-159.36033630371094
cur2_A	-158.47410583496094
cur3_A	-148.16160583496094
bus_V	23.61825180053711
filt_bus_V	23.65652847290039
filt_1ms_bus_V	23.639087677001953
fet_temp_C	22.055156707763672
filt_fet_temp_C	22.028596878051758
motor_temp_C	0.0
filt_motor_temp_C	0.0
d_A	6.8343634605407715
q_A	2.2289111614227295
position	-0.0602569580078125
velocity	-0.0001746091147651896

1. **Physical approach:**

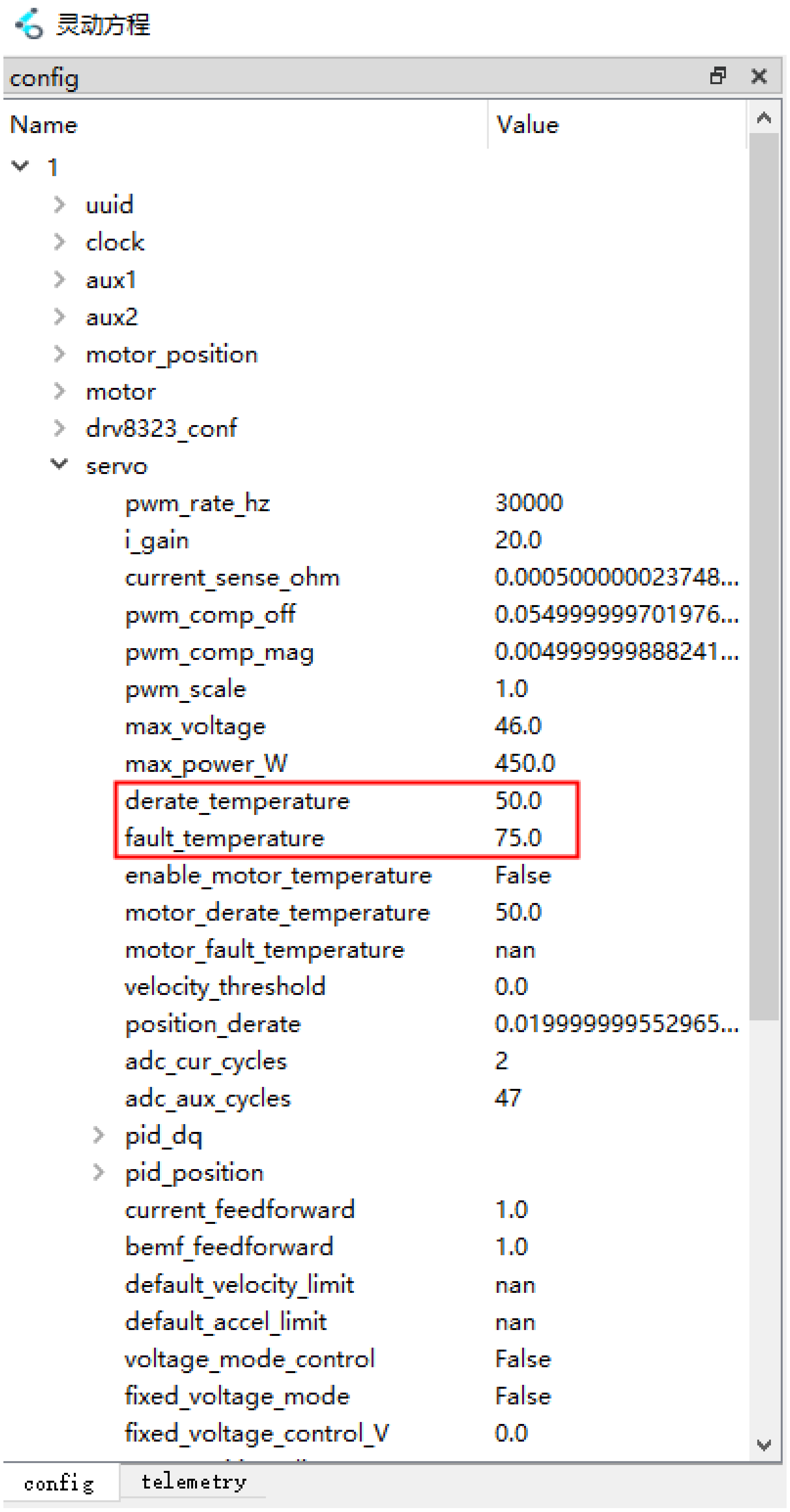
-
    - Add heat sink
    - heat sink
    - cold air

1. **Software Methods:**

You can set the following parameters:

parameters	functionality
servo.derate_temperature	When the temperature reaches this value, the torque starts to be limited
servo.fault_temperature	When the temperature reaches this value, a fault is triggered and all torque is stopped

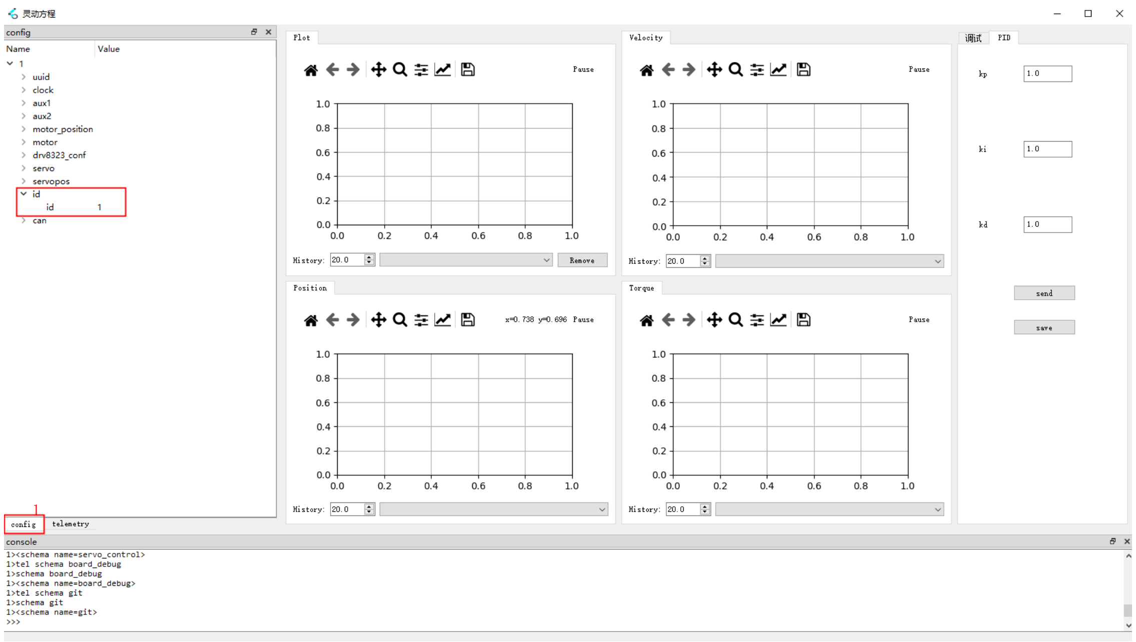
To ensure that the temperature coefficient is not affected, first of all, it is necessary to operate in a good heat dissipation environment (add heat sink, heat dissipation aluminum shell, low temperature), etc.. Secondly, you can modify its torque parameter affected by temperature to improve its performance **(must be carried out under the heat dissipation aluminum shell)**, raise the parameter **servo.derate_temperature**, **servo.fault_temperature** limit can be, as shown in the figure below.



Name	Value
1	
uuid	
clock	
aux1	
aux2	
motor_position	
motor	
drv8323_conf	
servo	
pwm_rate_hz	30000
i_gain	20.0
current_sense_ohm	0.000500000023748...
pwm_comp_off	0.054999999701976...
pwm_comp_mag	0.004999999888241...
pwm_scale	1.0
max_voltage	46.0
max_power_W	450.0
derate_temperature	50.0
fault_temperature	75.0
enable_motor_temperature	False
motor_derate_temperature	50.0
motor_fault_temperature	nan
velocity_threshold	0.0
position_derate	0.019999999552965...
adc_cur_cycles	2
adc_aux_cycles	47
pid_dq	
pid_position	
current_feedforward	1.0
bemf_feedforward	1.0
default_velocity_limit	nan
default_accel_limit	nan
voltage_mode_control	False
fixed_voltage_mode	False
fixed_voltage_control_V	0.0

2.8 Modifying the motor ID

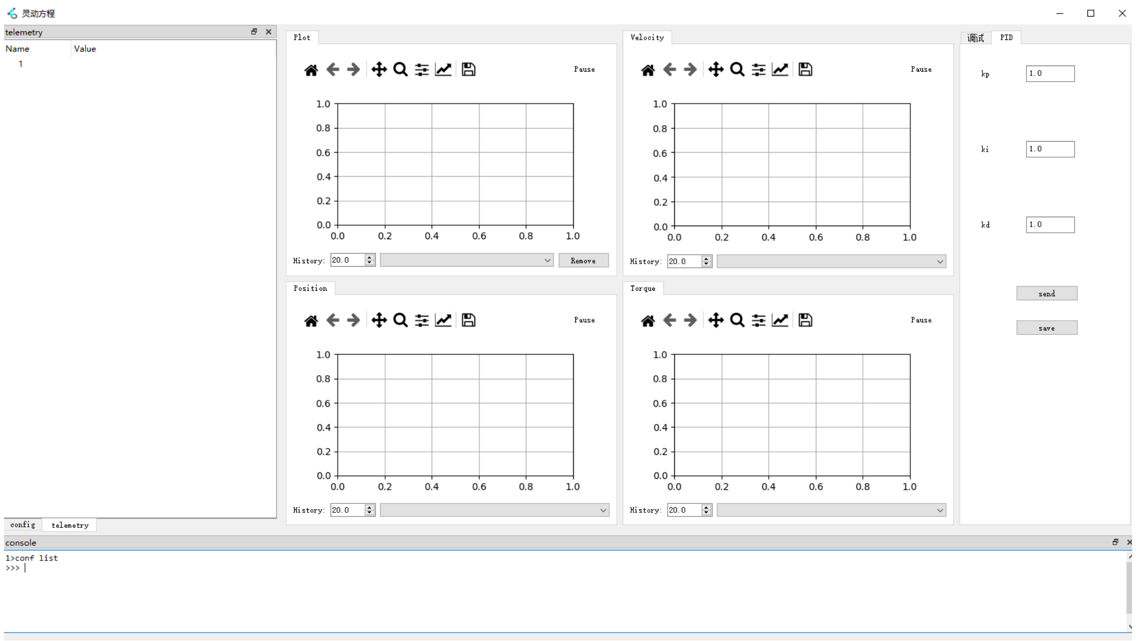
1. Connect the motor, open the host computer to read the motor ID, click on "**Open Tview**" as shown in the following figure, select the **config** item, open the id tab, modify the id number, enter or click on the blank space, the console to print the ID number has been modified. **Note: Do not disconnect the power during the process of modifying ID!**



```
Name                Value
1
  uuid
  clock
  aux1
  aux2
  motor_position
  motor
  drv8323_conf
  servo
  servopos
  id
    id              1
  can
```

```
1><schema name=servo_control>
1>tel schema board_debug
1>schema board_debug
1><schema name=board_debug>
1>tel schema git
1>schema git
1><schema name=git>
>>>
```

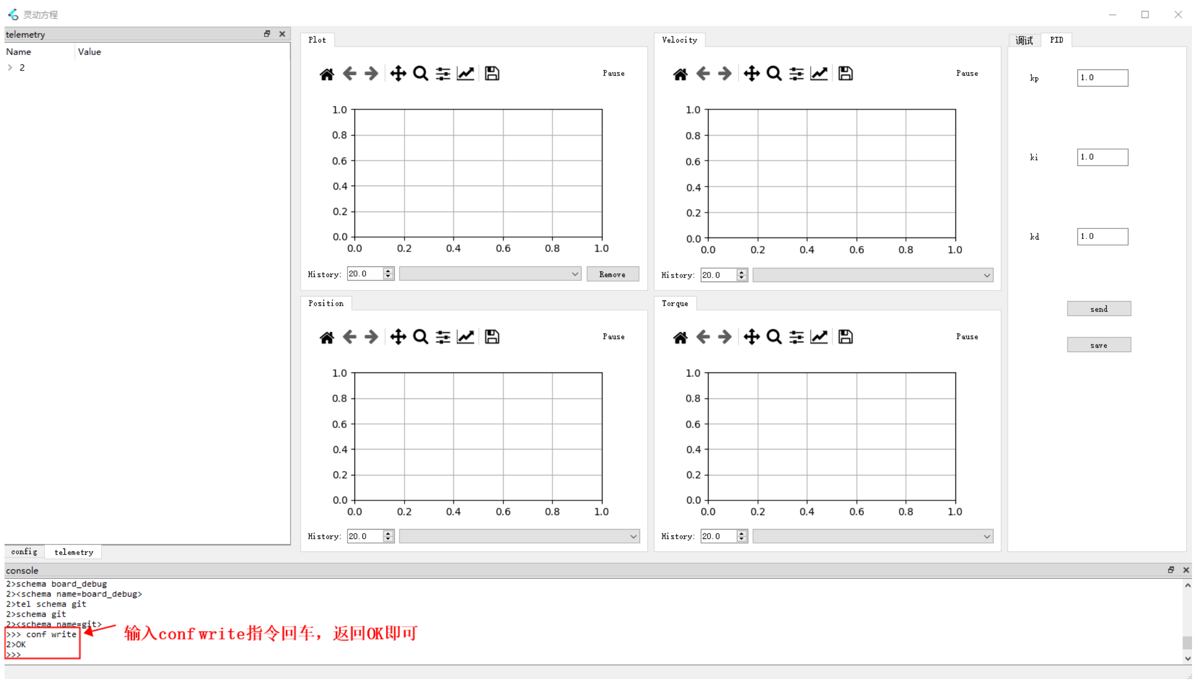
2. Directly after the modification, close the current window and reopen it, you can find that there is no driver board information in the upper computer as shown in the following figure, because the driver board id has been modified to 2.



```
Name        Value
  1
```

```
1>conf list
>>> |
```

3. Next, we close both windows of the host computer, re-open the host computer, the host computer will automatically recognize the changed ID, click "**Open Tview**", in the command line of the host computer, enter the save command **conf write** to save the ID, the feedback is **OK** that is, the ID is successfully modified, as shown in the figure below.



```
2>schema board_debug
2><schema name=board_debug>
2>tel schema git
2>schema git
2><schema name=git>
>>> conf write
2>OK
>>>
```

2.9 Common Control Instructions

2.9.1 FOC position control commands

- Syntax: **d pos [pos] [vel] [max_torque] [options...]**

Each optional element consists of a prefix character and a value:

p - kp Proportional: Sets the steel parameter for motor rotation.

d - kd scale: sets the damping parameter for motor rotation.

s - stop position: when given a non-zero speed, motion stops when the control position reaches this value.

f - Feedforward torque in Nm.

t - timeout: if another command is not received within so many seconds, enter timeout mode.

v - Speed Limit: The given value will override the global speed limit for the duration of this command.

a - Acceleration Limit: The given value will override the global acceleration limit for the duration of this command.

o - Fixed Voltage Override: when in effect, the control is considered to be enabled using the fixed voltage mode for the given voltage.

Example:

directives	functional explanation

directives	functional explanation
d pos nan 1 3	Torque of 3n/m, speed of 1 revolution/s.
d pos nan 1 3 s10	Move to position 10 with a torque of 3n/m and a speed of 1 revolution/s, then stop.

2.9.2 Stop command

directives	functional explanation
d stop	Put the controller into the stop state, thus controlling the motor to stop

2.9.3 Voltage mode (d pwm)

- Syntax: **d pwm [phase] [magnitude] [phase_rate]**

Functional explanation:

- **Phase** is the direction of the electric vector (radians)
- **mangnitdue** is strength (voltage volts)
- (Optional) **phase_rate** parameter controls the vector speed (radians per second)

Note: Give the voltage volt when the value from 0 to give, amplitude 0.1-0.2 or so increase, too large a voltage can lead to motor hot or even smoke damage to the motor, use caution!

2.9.4 Moment modes

directives	functional explanation
d pos nan 0 nan p0 d0 f[value]	The parameter after f is the adjustable torque parameter

2.9.5 Brake command

directives	functional explanation
d brake	Enter the "Brake" state. In this mode, all motor phases are shorted to ground, resulting in a passive "braking" action.

2.9.6 Saving instructions

directives	functional explanation
conf write	Writes the current values of all configurable parameters from RAM to memory

2.9.7 Position limits (d within)

- Syntax: **d within [value] [value] [value] [value]**

Example:

directives	functional explanation
d within -0.4 0.5 0.3	-0.4,0.5 are the upper and lower limits of the set position When the set limit is exceeded, a torque of 0.3 will be used to limit the torque.
d within nan 0.5 0.3	No upper position limit and 0.5 lower limit, when the set limit is exceeded, a torque of 0.3 will be used to limit the torque.

3 Lively board communication protocol description

3.1 CAN-FD Protocol Analysis

3.1.1 CAN-FD related description

1. CAN-FD baud rate:

- - Arbitration segment: 1 Mbps
  - Data Segment: Supports up to 5 Mbps, also available in 1Mbps.

1. ID: consists of 16 bits, where 0x7F is the broadcast address.

- - High 8 bits: indicates **the source address**:
  - - The highest bit is 1: a response is required.
    - Highest bit 0: No response required.
    - The remaining 7 bits: signal source address.
  - Lower 8 bits: indicates **the destination address**:
  - - Up to 0.
    - The remaining 7 bits indicate the destination address.

Example:

1. ID: 0x8001

- - The signal source address is 0.
  - The destination address is 1.
  - The highest bit is 1, indicating that a response is required.

1. ID: 0x100

- - The signal source address is 1.
  - The destination address is 0.
  - The highest bit is 0, indicating that no response is required.

3.1.2 Basic description of the protocol

1. The smallest unit is a subframe, and each subframe can write values to, or read data from, **one or more registers**.
2. A CAN-FD frame can consist of **one or more** subframes.
3. The individual functions of the motor are realized by writing the values of **one or more registers** corresponding to the function in an FDCAN frame.
4. The four basic data types, int8_t, int16_t, int32_t, and float, can be written to any register.
5. The data types in the **same subframe** must be the same, and the data types in **different subframes** can be different.
6. All basic types of transmission are in **small-end mode**, i.e., the low byte of data is sent first, followed by the high byte.
7. A CAN-FD that requires a **trailing padding** byte should use 0x50 for no operation (NOP).
8. **Unlimited** for various data types:

- - int8_t: 0x80
  - int16_t: 0x8000
  - int32_t: 0x80000000
  - float: NAN

3.1.3 Subframe parsing

3.1.3.1 Transmission protocol

① Mode 1

uint8_t tdata[] = {cmd, addr, a1, a2, b1, b2...} ;

1. cmd: indicates read/write, data type and number.
2. 1. The high four bits cmd[7:4] indicate read, write, and readback.
3. 1. 1. 0x0x: Write
      2. 0x1x: read
      3. 0x2x: Reply
4. 1. The lower four bits cmd[3:0] indicate the data type and number:
5. 1. 1. The high two cmd[3:2] indicate the data type:
6. 1. 1. 1. 00: int8_t

            2. 01: int16_t
            3. 10: int32_t
            4. 11: float
7. 1. 1. The lower two bits cmd[1:0] indicate the number of data.
8. 1. 1. 1. 01: A data.
            2. 10: Two figures.
            3. 11: Three figures.
            4. 00: **Mode 2** flag.
9. addr: start register address.
10. a1, a2, b1, b2... : Data written to the register, note: It should fulfill the 4 and 5 items of 1.2 Protocol Basics.
11. 1. a1, a2: data to be written to addr.
    2. b1, b2: data to be written to addr + 1.
    3. ... : The data to be written to add + n.

② Model II

uint8_t tdata[] = {cmd, num, addr, a1, a2, b1, b2...} ;

1. cmd: indicates read/write, data type and number.
2. 1. The high four bits cmd[7:4] indicate read, write, and readback.
3. 1. 1. 0000: Write
       2. 0001: Read
       3. 0010: Response
4. 1. The lower four bits cmd[3:0] indicate the data type and number:
5. 1. 1. The high two cmd[3:2] indicate the data type:
6. 1. 1. 1. 00: int8_t
            2. 01: int16_t
            3. 10: int32_t
            4. 11: float
7. 1. 1. The lower two bits cmd[1:0] are fixed to 00, indicating mode two, and the latter byte indicates the number of data.
8. num: number of data.
9. addr: start register address.
10. a1, a2, b1, b2... : Data written to the register, note: It should fulfill the 4 and 5 items of 1.2 Protocol Basics.
11. 1. a1, a2: data to be written to addr.
    2. b1, b2: data to be written to addr + 1.
    3. ... : The data to be written to add + n.

Mode 2 Substance: When the number of data in xxx of mode 1 is written as 0, the last byte is used to indicate the number of data, and the rest of the bytes are shifted back one bit.

3.1.3.2 Receiving protocol

- **The receive protocol mode is determined by the mode of the send protocol.**
- **The receive protocol mode is the same as the transmit protocol mode.**

① Mode 1

Assuming the acquired data is uint16_t

uint8_t rdata[] = {cmd, addr, a1, a2, b1, b2, ... , cmd1, addr1, c1, c2, c3, c4}

- cmd:
-   - Higher four bits cmd[7, 4]: 0010 Indicates a reply.
    - 2~3 bits cmd[3, 2]: indicate the type.
    -   - 00: int8_t Type.
        - 01: int16_t Type.
        - 10: int32_t Type.
        - 11: float type.
    - Lower 2 bits cmd[1, 0]: indicates the number.
    -   - 00: Indicates mode two.
        - 01: A data.
        - 10: Two figures.
        - 11: Three figures.
- addr: the address to start fetching.
- a1, a2: Data 1, small end mode.
- b1, b2: Data 2, small end mode.

②Model II

uint8_t rdata[] = {cmd, addr, num, a1, a2, b1, b2, ... , cmd1, addr1, c1, c2, c3, c4}

- cmd:
-   - Higher four bits cmd[7, 4]: 0010 Indicates a reply.
    - 2~3 bits cmd[3, 2]: indicate the type.
    -   - 00: int8_t Type.
        - 01: int16_t Type.
        - 10: int32_t Type.
        - 11: float type.
    - Lower 2 bits cmd[1, 0]: indicates the number.
    -   - 00: Indicates mode two, and the last byte indicates the number of data.
- num: number of data.
- addr: the address to start fetching.
- a1, a2: Data 1, small end mode.
- b1, b2: Data 2, small end mode.

3.1.3.3 Examples

Ⅰ, send protocol example

① Mode 1

uint8_t cmd[] = {0x01, 0x00, 0x0A, 0x0A, 0x20, 0x00, 0x00, 0x00, 0x80, 0x10, 0x27, 0x00, 0x00, 0x50, 0x50, 0x50};

The CAN-FD frame consists of two subframes:

1. Subframe 1: The overall meaning is that the motor enters position mode.

- - 0x01: beginning of the first subframe
  - - The high four bits are 0000, indicating a write operation.
    - Four places lower:
    - - The high 2 bits are: 00, indicating the int8_t type.
      - The lower 2 bits are: 01, indicating 1 data.
  - 0x00: start register address: checking the table shows that 0x00 register indicates the motor mode setting.
  - 0x0A: Write 0x0A to the 0x00 register.

1. Subframe 2: the overall meaning is that the position is not limited and the speed is 0.1 revolutions/sec.

- - 0x0A: Beginning of the 2nd subframe
  - - The high 8 bits are 0x0, indicating a write operation.
    - 8 bits lower:
    - - The high 2 bits are: 10, indicating the int32_t type.
      - The lower 2 bits are: 10, indicating 2 data.
  - 0x20: start register address: looking up the table, we can see that 0x20 register indicates the position and 0x21 register indicates the speed.
  - 0x00, 0x00, 0x00, 0x80: small end mode, i.e. 0x80000000 is written to 0x20 register, which means that the motor position is unlimited.
  - 0x10, 0x27, 0x00, 0x00: Small end mode, i.e., 0x2710 is written to the 0x21 register, indicating that the motor speed is set to 0.1 rpm.

1. 0x50, 0x50, 0x50: Since the closest CAN-FD transmit bytes are 12 and 16, a 3 byte placeholder byte is added.

② Model II

uint8_t cmd[] = {0x01, 0x00, 0x0A, 0x08, 0x02, 0x20, 0x00, 0x00, 0x00, 0x80, 0x10, 0x27, 0x00, 0x00, 0x50, 0x50};

The CAN-FD frame consists of two subframes:

1. Subframe 1: The overall meaning is that the motor enters position mode.

- - 0x01: beginning of the first subframe
  - - The high 8 bits are 0001, indicating a write operation.
    - 8 bits lower:

- - - The high 2 bits are: 00, indicating the int8_t type.
      - The lower 2 bits are: 01, indicating 1 data.
  - 0x00: start register address: checking the table shows that 0x00 register indicates the motor mode setting.
  - 0x0A: Write 0x0A to the 0x00 register.

1. Subframe 2: the overall meaning is that the position is not limited and the speed is 0.1 revolutions/sec.

- - 0x08: Beginning of the 2nd subframe
  - - The high 8 bits are 0x0, indicating a write operation.
    - 8 bits lower:
    - - The high 2 bits are: 10, indicating the int32_t type.
      - The lower 2 bits are: 00, indicating mode 2, and the last byte indicates the number of data.
  - 0x02: 2 data.
  - 0x20: start register address: looking up the table, we can see that 0x20 register indicates the position and 0x21 register indicates the speed.
  - 0x00, 0x00, 0x00, 0x80: small end mode, i.e. 0x80000000 is written to 0x20 register, which means that the motor position is unlimited.
  - 0x10, 0x27, 0x00, 0x00: Small end mode, i.e., 0x2710 is written to the 0x21 register, indicating that the motor speed is set to 0.1 rpm.

1. 0x50, 0x50: Since the closest CAN-FD transmit bytes are 12 and 16, a 2 byte placeholder byte is added.

II. Examples of reception protocols

- **The receive protocol mode is determined by the mode of the send protocol.**
- **The receive protocol mode is the same as the transmit protocol mode.**

① Mode 1

uint8_t rdata[] = {0x28, 0x04, 0x00, 0x0A, 0x00, 0x00, 0x00, 0x96, 0x1D, 0x04, 0x00, 0x54, 0x40, 0x02, 0x00, 0x86, 0x01, 0x00, 0x00, 0x50};

The frame consists of one subframe:

1. Subframe 1: The overall meaning is that the motor enters position mode.

- - 0x28:
  - - The high 8 bits are 0010, indicating a reply operation.
    - 8 bits lower:
    - - The high 2 bits are: 10, indicating the int32_t type.
      - The lower 2 bits are: 00, indicating mode 2, and the last byte indicates the number of data.
  - 0x04: 4 data.
  - 0x00: Start register address
  - 0x0A, 0x00, 0x00, 0x00: 0x00 register value, look up table for motor mode, decimal is 10, look up table for position mode.

- 0x96, 0x1D, 0x04, 0x00: small end mode, decimal 269718, i.e. the current position of the motor is 2.69718 revolutions.
- 0x54, 0x40, 0x02, 0x00: Small end mode, decimal 147540, i.e. current motor speed is 1.4754 rpm.
- 0x86, 0x01, 0x00, 0x00: Small end mode, decimal 390, i.e. current actual output torque: 0.0039 NM.
- 0x50: Placeholder.

The first three characters show that this CAN-FD is replying to 0x18, 0x04, 0x00.

② Model II

uint8_t rdata[] = {0x2B, 0x01, 0x0A, 0x00, 0x00, 0x00, 0x96, 0x1D, 0x04, 0x00, 0x54, 0x40, 0x02, 0x00, 0x86, 0x01, 0x00, 0x00, 0x50};

The frame consists of one subframe:

1. Subframe 1: The overall meaning is that the motor enters position mode.

-
  - 0x2B:
  -
    - The high 8 bits are 0x2, indicating a reply operation.
    - 8 bits lower:
    -
      - The high 2 bits are: 10, indicating the int32_t type.
      - The lower 2 bits are: 11, indicating 3 data.
  - 0x01: Start register address
  - 0x0A, 0x00, 0x00, 0x00: 0x00 register value, look up table for motor mode, decimal is 10, look up table for position mode.
  - 0x96, 0x1D, 0x04, 0x00: small end mode, decimal 269718, i.e. the current position of the motor is 2.69718 revolutions.
  - 0x54, 0x40, 0x02, 0x00: Small end mode, decimal 147540, i.e. current motor speed is 1.4754 rpm.
  - 0x86, 0x01, 0x00, 0x00: Small end mode, decimal 390, i.e. current actual output torque: 0.0039 NM.
  - 0x50: Placeholder.

The first three characters show that this CAN-FD is replying to 0x1B, 0x01.

Example of frame parsing

A single CAN-FD frame can be used to command the servo and initiate queries to certain registers. Example frames are shown below, encoded in hexadecimal with comments.

For example the entire CAN-FD message would be (in hexadecimal): (microcontroller sends) **0x01000a07206000200150ff140400130d**

digital	descriptive
01	Write to a single int8 register (number of registers encoded in 2 LSBs)

digital	descriptive
00	Start register number "mode"
0a	"Location" mode
07	Write 3 int16 registers (number of registers encoded in 2 LSBs)
20	Register 0x020
6000	Position = 0x0060 = 96 = 3.456 degrees
2001	Speed = 0x0120 = 288 = 25.92 dps
50ff	Feedforward torque = 0xff50 = -176 = 1.76 N*m
14	Read int16 register
04	Read 4 registers
00	Starting at 0x000 (so 0x000 mode, 0x001 position, 0x002 speed, 0x003 torque)
13	Read 3x int8 register
0d	Starting at 0x00d (so 0x00d voltage, 0x00e temperature, 0x00f fault code)

Therefore, to use **the fdcanusb converter to** send it to a device configured with default address 1, you can write.

- can send **8001 01000a07206000200150ff140400130d**

This 80 is used for two purposes in ID. The high bit of the setting forces the device to respond (otherwise it will not respond even if a query command is sent). The remaining bits are the "ID" of the response. The possible responses from the servo in response to this command are shown below:

- rcv **100 2404000a005000000170ff230d181400**

Decode, which means:

digital	descriptive
100	From device "1" to device "0"
24	Re: int16 value
04	4 registers
00	Starting from register 0
0a00	In Mode 10 - Position
5000	The position is 0x0050 = 80 = 2.88 degrees.
0001	The speed is 0x0100 = 256 = 23.04 dps

digital	descriptive
70ff	The torque is 0xff70 = -144 = -1.44 Nm.
23	Re 3 int8 values
0d	Starting at register 0x00d
18	Voltage 12V
14	Temperature of 20°C
00	faultless

3.1.4 Description of common types (units)

3.1.4.1 Current (A)

data type	LSB	Actual (A))
int8	1	1
int16	1	0.1
int32	1	0.001
float	1	1

3.1.4.2 Voltage (V)

data type	LSB	Actual (V))
int8	1	0.5
int16	1	0.1
int32	1	0.001
float	1	1

3.1.4.3 Torque (Nm)

- **True torque = k * tqe + d**

① 5046 Torque (Nm)

data type	Slope (k)	Offset (d)
int8	0.225234	-0.457642
int16	0.004497	-0.448998
int32	0.000439	-0.410969
float	0.433600	-0.519366

② 4538 Torque (Nm)

data type	Slope (k)	Offset (d)
int8	0.002024	-0.293455
int16	0.003813	-0.280438
int32	0.000368	-0.225920
float	0.414104	-0.472467

3.1.4.4 Temperature (°C)

data type	LSB	Actual (°C)
int8	1	1
int16	1	0.1
int32	1	0.001
float	1	1

3.1.4.5 Time (s)

data type	LSB	Actual (s)

data type	LSB	Actual (s)
int8	1	0.01
int16	1	0.001
int32	1	0.000001
float	1	1

3.1.4.6 Position (turn)

data type	LSB	actual	Actual (°)
int8	1	0.01	3.6
int16	1	0.0001	0.036
int32	1	0.00001	0.0036
float	1	1	360

3.1.4.7 Speed (rpm)

data type	LSB	Actual (rpm)
int8	1	0.01
int16	1	0.0001
int32	1	0.00001
float	1	1

3.1.4.8 Acceleration (rpm^2)

data type	LSB	Actual (rpm^2)
int8	1	0.5
int16	1	0.01
int32	1	0.001

data type	LSB	Actual (rpm^2)
float	1	1

3.1.4.9 PWM scale (unitless)

data type	LSB	practice
int8	1	1/127 - 0.007874
int16	1	1/32767 - 0.000030519
int32	1	(1/2147483647) - 4.657^10
float	1	1

3.1.4.10 Kp, Kd scale (unitless)

data type	LSB	practice
int8	1	1/127 - 0.007874
int16	1	1/32767 - 0.000030519
int32	1	(1/2147483647) - 4.657^10
float	1	1

3.1.5 Examples of functions

Description:

- **A brief description of the motor control modes provided in the sample program is presented below.**
- **See the example project provided for details.**
- **This motor can achieve more than that, if you need to customize special functions, you can use your imagination according to the register table.**

3.1.5.1 DQ Voltage control

Description:

- D Phase voltage defaults to 0.
- Q Phase voltage is controlled by the user.

Provide functions:

void set_dq_volt_float(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, float volt);
 void set_dq_volt_int32(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int32_t voltage);
 void set_dq_volt_int16(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int16_t volt);

3.1.5.2 DQ current control

Description:

- D phase current defaults to 0.
- Q Phase current is user controlled.

Provide functions:

void set_dq_current_float(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, float current);
 void set_dq_current_int32(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int32_t current);;
 void set_dq_current_int16(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int16_t current);

3.1.5.3 Position control

Description:

- Rotate to the specified position with maximum speed and torque.
- Positive and negative indicate direction.

Provides functions:

void set_pos_float(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, float pos);
 void set_pos_int32(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int32_t pos);; void set_pos_int32(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int32_t pos);
 void set_pos_int16(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int16_t pos); void set_pos_int16(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, float pos).

3.1.5.4 Speed control

Description:

- Rotate at the set speed.
- Positive and negative indicate direction.

Provides functions:

void set_val_float(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, float val);
 void set_val_int32(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int32_t val);
 void set_val_int16(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int16_t val); void set_val_int16(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int16_t val).

3.1.5.5 Torque control

Description:

- Rotate with a given torque.

Provides functions:

void set_torque_float(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, float torque);
 void set_torque_int32(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int32_t torque);;
void set_torque_int32(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int32_t torque)
 void set_torque_int16(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int16_t torque);
void set_torque_int16(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int16_t torque).

3.1.5.6 Position, speed, maximum torque control

Description:

- Rotates to a specified position at a given speed and limits the maximum torque output.

Provide functions:

void set_pos_vel_tqe_float(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, float pos, float val, float torque);
 void set_pos_vel_tqe_int32(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int32_t pos, int32_t val, int32_t torque);;
 void set_pos_vel_tqe_int16(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int16_t pos, int16_t val, int16_t torque);

3.1.5.7 Position, speed, torque, PD control

Description:

- Rotates to a specified position at a given speed and limits the maximum torque output.
- The internal Kp, Kd ratio can also be adjusted.

Provide functions:

void set_pos_val_tqe_pd_float(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, float pos, float val, float tqe, float kp, float kd);
 void set_pos_val_tqe_pd_int32(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int32_t pos, int32_t val, int32_t tqe, float rkp, float rkd);
 void set_pos_val_tqe_pd_int16(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int16_t pos, int16_t val, int16_t tqe, float rkp, float rkd);

3.1.5.8 Speed, speed limit control

Description:

- Rotates at the specified speed, or at the speed limit if the speed is greater than the speed limit.

Provide functions:

void set_val_valmax_int16(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int16_t val, int16_t vel_max);;

3.1.5.9 Position, velocity, acceleration control (trapezoidal control)

Description:

- Specifies that the motor rotates to a certain position and limits the maximum speed and acceleration during rotation.

Provides functions:

void set_pos_valmax_acc_float(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, float pos, float vel_max, float acc);
void set_pos_valmax_acc_int32(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int32_t pos, int32_t vel_max, int32_t acc);
void set_pos_valmax_acc_int16(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int16_t pos, int16_t vel_max, int16_t acc);

3.1.5.10 Velocity, acceleration control

Description:

- Accelerates to the specified speed with the specified acceleration.

Provides functions:

void set_val_acc_float(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, float val, float acc);
void set_val_acc_int32(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int32_t val, int32_t acc);;
void set_val_acc_int16(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor, int16_t val, int16_t acc);

3.1.5.11 Resetting the zero point

Description:

- Sets the current position to zero.

Provides functions:

void set_pos_rezero(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor).

3.1.5.12 Save settings

Description:

- Saves the motor configuration information.
- Currently it is only used to reset the zero point.

Provide functions:

void set_conf_write(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor).

3.1.5.13 Motor brake

Description:

- All phases of the motor are shorted to ground, creating a passive "braking" effect.

Provide functions:

void set_motor_brake(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor).

3.1.5.14 Motor stop

Description:

- Go to stop.

Provides functions:

void set_motor_stop(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor);

3.1.5.15 Reading the motor status

Description:

- Provides routines to read four types of data: motor operation status, position, speed, and torque.
- The function here is for the motor to send these four kinds of data back, see the sample code for the specific routine to parse the data.

Provides functions:

void read_motor_state_float(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor);
 void read_motor_state_int32(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor); void read_motor_state_int32(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor);
 void read_motor_state_int16(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor); void read_motor_state_int16(FDCAN_HandleTypeDef *fdcanHandle, motor_e motor);

3.1.6 Description of the example program

- **The sample motor functions are in the libelybot.c and libelybot.h files.**
- **The sample program will not run any control code by default, if you need to test a certain function, please uncomment it in the main.c file or write it by yourself.**

Macro definitions to watch out for:

1. MOTOR_MODEL: Used to select the motor model for correcting the input torque.
2.
    1. MOTOR_MODEL = 5046: 5046 Motor.
    2. MOTOR_MODEL = 4538: 4538 Motor.
    3. MOTOR_MODEL = 50471: 5047 Unipolar motor.
    4. MOTOR_MODEL = 50472: 5047 bipolar motor.
3. POS_FLAG: used to **test the position mode**, there are three modes, the three modes are just different data types, the effect is -0.5 ~ 0.5 rotations back and forth.
4.
    1. POS_FLAG = 1: float
    2. POS_FLAG = 2: int32
    3. POS_FLAG = 3: int16
5. READ_MOTOR_FLAG: Used to change the **type of data used to read the motor status**.
6.
    1. READ_MOTOR_FLAG = 1: float
    2. READ_MOTOR_FLAG = 2: int 32
    3. READ_MOTOR_FLAG = 3: int 16
7. POS_REZERO: **reset zero** function for testing the motor.
8.
    1. The effect is to set the current position to zero 3 seconds after the motor is powered up.
    2. Note: You need to let the motor stop before resetting the zero position, otherwise it is not effective
    3. To test this feature uncomment POS_REZERO.
9. MOTOR_STOP: Used to test the motor **stop** function.
10.
    1. To test this function, uncomment MOTOR_STOP.
    2. The effect is that the motor will stop 3 seconds after powering up.
    3. Note: To be used in conjunction with a motor control function, enabling the MOTOR_STOP macro does not change any of the motor control functions.
11. MOTOR_BRAKE: Used to test the motor **brake** function function.
12.
    1. To test this function uncomment MOTOR_BRAKE.

3.1.7 Description of Register Functions

register address	Register Name	r/w	Register Description
0x000	paradigm	R/W	Motor operation mode (see 3.2.8 Motor operation mode for specific modes)
0x001	placement	R	Motor output shaft position
0x002	tempo	R	Motor output shaft speed

register address	Register Name	r/w	Register Description
0x003	torque	R	Output shaft torque of the motor
0x004	Q Phase current	R	Q Phase current
0x005	D Phase current	R	D Phase current
0x006	reservations		reservations
0x00d	input voltage	R	Input Voltage
0x00e	temp	R	temp
0x00f	error code	R	Specific error codes are shown in Table 3 Error Code Descriptions
0x010	PWM phase A	R/W	Controls the raw PWM value of phase A in PWM mode
0x011	PWM phase B	R/W	Controls the raw PWM value of phase B in PWM mode
0x012	PWM phase C	R/W	Raw PWM value of control C phase in PWM mode
0x014	Voltage phase A	R/W	In voltage mode, controls the voltage applied to phase A
0x015	Voltage phase B	R/W	In voltage mode, controls the voltage applied to phase B
0x016	Voltage phase C	R/W	In voltage mode, controls the voltage applied to phase C
0x018	Voltage FOC Angle	R/W	Voltage mode to control the desired electrical angle (not multiplied by the number of pole pairs)
0x019	Voltage FOC Voltage	R/W	Voltage focus mode controls the desired applied phase voltage
0x01a	D Voltage	R/W	DQ Voltage Mode, Controls D Phase Voltage
0x01b	Q Voltage	R/W	DQ Voltage Mode, Controls Q Phase Voltage
0x01c	Q Current	R/W	DQ current mode, controls Q-phase current
0x01d	D Current	R/W	DQ current mode, controls D-phase current
0x020	position command	R/W	Position mode, control position

register address	Register Name	r/w	Register Description
0x021	speed command	R/W	Control speed in position mode
0x022	Feedforward torque	R/W	Position mode to control feed-forward torque
0x023	Kp ratio	R/W	In position mode, scaling control items down by a given factor
0x024	Kd ratio	R/W	Positional mode shrinks the differential control term by a given factor
0x025	maximum torque	R/W	Maximum torque controlled in position mode
0x026	stop position	R/W	In position mode and when a non-zero speed is commanded, motion stops when a given position is reached.
0x027	reservations		reservations
0x028	speed limit	R/W	Global Speed Limit
0x029	acceleration limit	R/W	Global acceleration limit
0x030	proportional torque	R	Torque of proportional term in PID controller
0x031	Integral torque	R	Torque of integral term in PID controller
0x032	Differential torque	R	Torque of differential term in PID controller
0x033	Feedforward torque	R	Feedforward in PID controllers
0x034	Total control torque	R	Total control torque in position mode
0x040	lower limit	R/W	In range mode, it controls the minimum allowable position
0x041	limit	R/W	In range mode, it controls the maximum allowable position
0x042	Feedforward torque		0x022 Register Mapping
0x043	Kp ratio		0x023 Register Mapping
0x044	Kd ratio		0x024 Register Mapping

register address	Register Name	r/w	Register Description
0x045	maximum torque		0x025 Register Mapping

3.1.8 Motor operation mode

Mode number	Model name
0	Stop, clear error
1	incorrect
2, 3, 4	ready run
5	PWM Mode
6	Voltage Mode
7	foc voltage mode
8	DQ Voltage Mode
9	DQ Current Mode
10	Position Mode
11	timeout mode
12	Zero Speed Mode
13	Scope Mode
14	Measurement inductance mode
15	Brake Mode
16	reservations
17	reservations

3.2 CAN Protocol Analysis

3.2.1 CAN-related descriptions

1. CAN baud rate:

- - Arbitration segment: 1 Mbps
  - Data segment: 1 Mbps.

1. ID: consists of 16 bits, where 0x7F is the broadcast address.

- - High 8 bits: indicates **the source address**:
  - - The highest bit is 1: a response is required.
    - Highest bit 0: No response required.
    - The remaining 7 bits: signal source address.
  - Lower 8 bits: indicates **the destination address**:
  - - Up to 0.
    - The remaining 7 bits indicate the destination address.

Example:

1. ID: 0x8001

- - The signal source address is 0.
  - The destination address is 1.
  - The highest bit is 1, indicating that a response is required.

1. ID: 0x100

- - The signal source address is 1.
  - The destination address is 0.
  - The highest bit is 0, indicating that no response is required.

3.2.2 Description of the model

3.2.2.1 Normal mode (position and speed cannot be controlled simultaneously)

uint8_t cmd[] = {0x07, 0x07, pos1, pos2, val1, val2, tqe1,tqe2};

- The common protocol consists of: command bits (2 bytes) + position (2 bytes) + speed (2 bytes) + torque (2 bytes) for a total of 8 bytes.
- 0x07 0x07: Normal mode with control of speed and torque, position and torque (see 3.1.5.1 Normal mode).
- **The position, velocity, and torque** data in the protocol are in **small-end mode**, i.e., the low byte is sent first and the high byte is sent second.
- - For example, in pos = 0x1234, pos1 = 0x34 and pos2 = 0x12.
- This mode can be divided into **two** types of control:
- - Position, torque control (at this point val=0x8000 for unlimited).
  - Speed, torque control (at this time pos=0x8000, indicating no limit).

3.2.2.2 Moment mode

uint8_t cmd[] = {0x05, 0x13, tqe1, tqe2};

- Torque mode protocol consists of: command bits (2 bytes) + torque (2 bytes).
- 0x05 0x13: Pure torque mode, followed by two bytes of torque data. (See 3.1.5.2 Moment Mode).
- The torque data in the protocol is in small end mode, i.e., the low byte is sent first and the high byte is sent second.
- - As in tqe = 0x1234, tqe1 = 0x34 and tqe2 = 0x12.

3.2.2.3 Co-control mode (position, speed and torque can be controlled simultaneously)

uint8_t cmd[] = {0x07, 0x35, val1, val2, tqe1, teq2, pos1,pos2};

- Co-control mode protocol: Command bits (2 bytes) + speed (2 bytes) + torque (2 bytes) + position (2 bytes) consists of 8 bytes.
- 0x07 0x35: Co-control mode, has been specified speed rotation to the specified position, and limit the maximum torque.
- The parameter 0x8000 in this mode means **unlimited** (unlimited speed and torque are the maximum values).
- - e.g. val = 5000, tqe = 1000, pos = 0x8000: means that the motor rotates all the time at 0.5 rpm with a maximum torque of 0.1 NM.
- The position, velocity, and torque data in the protocol are in small-end mode, i.e., the low byte is sent first and the high byte is sent second.
- - For example, in pos = 0x1234, pos1 = 0x34 and pos2 = 0x12.

3.2.3 Motor status data reading

1. The protocol for reading the motor status section is the same as in CAN-FD, the only difference being that CAN is limited to 8 byte data segments.
2. For register addresses and function descriptions, please see the Register **Functions, Motor Operation Modes, and Error Reporting Code Descriptions. xlsx** file.
3. Due to the 8-byte data segment limitation of CAN, the maximum motor information returned in one CAN frame is limited:
4. 1. Motor information for a register of type float or int32_t.
   2. 3 motor messages of type int16_t with consecutive addresses.
   3. 6 motor messages of type int8_t with consecutive addresses.
5. The routine provides the int16_t example function to query the motor position, speed, torque information and the motor information parsing (the routine uses the C common body to directly copy the data from byte 3 to byte 8 of the CAN).

3.2.3.1 Transmission protocol description

uint8_t tdata[] = {cmd, addr, cmd1, addr1, cmd2, add2};

The **approximate meaning is:** read cmd[0, 1] cmd[3, 2] type data from addr.

- cmd:
-
  - Higher four bits [7, 4]: 0001 Indicates reading.
  - 2~3 bits [3, 2]: Indicates the type.
  -
    - 00: int8_t Type.
    - 01: int16_t Type.
    - 10: int32_t Type.
    - 11: float type.
  - Lower 2 bits [1, 0]: Indicates the number.
  -
    - 01: A data.
    - 10: Two figures.
    - 11: Three figures.
- addr: the address to start fetching.
  Multiple cmd , addr can be concatenated together to read data with non-contiguous addresses and different types at once.

3.2.3.2 Acceptance of the description of the agreement

Assuming the acquired data is uint16_t

uint8_t rdata[] = {cmd, addr, a1, a2, b1, b2, ... , cmd1,addr1, c1, c2, c3, c4}

- cmd:
- Higher four bits [7, 4]: 0010 Indicates a reply.
-
  - 2~3 bits [3, 2]: Indicates the type.
  - 00: int8_t Type.
  -
    - 01: int16_t Type.
  - 10: int32_t Type.
  -
    - 11: float type.
- Lower 2 bits [1, 0]: Indicates the number.
-
  - 01: A data.
  - 10: Two figures.
  -
    - 11: Three figures.
- addr: the address to start fetching.
- a1, a2: Data 1, small end mode.
- b1, b2: Data 2, small end mode.

3.2.3.3 Examples

1. We need to read position, speed and torque data.
2. From the register excel sheet, we can see that the data addresses for position, speed and torque are 01, 02 and 03 respectively.
3. From this we can see that we can read 3 consecutive data from address 01. Considering that CAN transfers a maximum of 8 bytes of data at a time and cmd + addr takes up 2 bytes, the data type can be up to the int16_t type.
4. From the above, the binary of cmd is: 0001 1011 and the hexadecimal is: 0x17.
5. It is necessary to read from address 01, so addr is 0x01.
6. The total data to be sent is uint8_t tdata[] = {0x17, 0x01}.

The sample code is as follows:

```
/**
 * @brief Read the motor
 * @param id
 */
 void motor_read(uint8_t id)
 {
 static uint8_t tdata[8] = {0x17, 0x01};
 CAN_Send_Msg(0x8000 | id, tdata, sizeof(tdata));
 }
```

uint8_t cmd[] = {0x17, 0x01};

The overall meaning is: start from address 0x01, read 3 int16_t registers (check the table to know that the registers at address 0x01~0x03 represent position, speed and torque, respectively), so the command is to query the position, speed and torque information of the motor.

- 0x17:
-
    - The binary of 0x17[7:4] is 0001: indicates a read.
    - The binary of 0x17[3:2] is 01: indicating that the data type is int16_t.
    - The binary of 0x17[1:0] is 11: indicating that the number of data is 3.
- 0x01:
-
    - Starts at address 0x01.

Accept data examples accordingly:

uint8_t rdata[] = {0x27, 0x01, 0x38, 0xf6, 0x09, 0x00, 0x00,0x00};

- 0x27: corresponds to 0x17 sent.
- 0x01: Starts at address 0x01.
- 0x38 0xf6: Position data: 0xf638, i.e. -2505.
- 0x09 0x00: Speed data: 0x0009, i.e. 9.
- 0x00 0x00: Torque data: 0x0000, i.e. 0.

3.2.4 Motor stop

Description:

1. Stop the motor.
2. Corresponds to the host computer command d stop

```
/**
 * @brief Motor stop
 /*
 void motor_stop(uint8_t id)
 {
 uint8_t tdata[] = {0x01, 0x00, 0x00};

 CAN_Send_Msg(0x8000 | id, tdata, sizeof(tdata));
 }
```

3.2.5 Example functions

3.2.5.1 General mode

1. position control

   ```
   /**
    * @brief Position Control
    * @param id Motor ID
    * @param pos Position: unit 0.0001 turns, e.g. pos = 5000 means turn to 0.5 turns position.
    * @param torque
    */
    void motor_control_Pos(uint8_t id,int32_t pos,int16_t tqe)
    {
    uint8_t tdata[8] = {0x07, 0x07, 0x0A, 0x05, 0x00.
    0x00, 0x80, 0x00};
    *(int16_t *)&tdata[2] = pos.
    *(int16_t *)&tdata[6] = tqe;
    uint32_t ext_id = (0x8000 | id);
    CAN_Send_Msg(ext_id, tdata, 8);
    }
   ```

2. speed control

   ```
   /**
    * @brief Position Control
    * @param id Motor ID
    * @param pos Position: unit 0.0001 turns, e.g. pos = 5000 means turn to 0.5 turns position.
    * @param torque
    */
    void motor_control_Pos(uint8_t id,int32_t pos,int16_t tqe)
    {
    uint8_t tdata[8] = {0x07, 0x07, 0x0A, 0x05, 0x00.
    0x00, 0x80, 0x00};
    *(int16_t *)&tdata[2] = pos.
   ```

```
        *(int16_t *)&tdata[6] = tqe;
        uint32_t ext_id = (0x8000 | id);
        CAN_Send_Msg(ext_id, tdata, 8);
        }
```

3.2.5.2 Moment mode

```
/**
 * @brief Torque mode
 * @param id motor id
 * @param tqe Torque
 /*
 void motor_control_tqe(uint8_t id,int32_t tqe)
 {
 uint8_t tdata[8] = {0x05, 0x13, 0x00, 0x80, 0x20.
 0x00, 0x80, 0x00};
 *(int16_t *)&tdata[2] = tqe;
 CAN_Send_Msg(0x8000 | id, tdata, 4);
 }
```

3.2.5.3 Collaborative control models

```
/**
 * @brief motor position-speed-feedforward torque (max torque) control, int16 type
 * @param id motor id
 * @param pos Position: unit 0.0001 laps, e.g. pos = 5000 means turn to 0.5 laps position.
 * @param val speed: unit 0.00025 revolutions/second, such as val = 1000 that 0.25 revolutions/second
 * @param tqe Maximum torque.
 /*
 void motor_control_pos_val_tqe(uint8_t id, int16_t pos.
 int16_t val, int16_t tqe)
 {
 static uint8_t tdata[8] = {0x07, 0x35, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00
 0x00, 0x00, 0x00};
 *(int16_t *)&tdata[2] = val.
 *(int16_t *)&tdata[4] = tqe;
 *(int16_t *)&tdata[6] = pos;
 CAN_Send_Msg(0x8000 | id, tdata, 8);
 }
```

3.2.6 STM32H730 Full Sample Code

libelybot_can. h

```
#ifndef _LIBELYBOT_CAN_H
 #define _LIBELYBOT_CAN_H

 #include "main.h"

 /* NAN means not limited */
 #define INI8_NAN 0x80
 #define INT16_NAN 0x8000
 #define INT32_NAN 0x80000000

 typedef struct
 {
 uint32_t id; int16_t position; uint32_t
 int16_t position; int16_t velocity; int16_t
 int16_t velocity; int16_t torque; int16_t
 int16_t velocity; int16_t torque
 }motor_state_s.

 typedef struct
 union
 int16_t velocity; int16_t torque; }motor_state_s
 motor_state_s motor; typedef {
 motor_state_s motor; uint8_t data[24]; uint8_t
 uint8_t data[24]; }; { motor_state_s motor
 }; }
 }motor_state_t; extern motor_state_t motor_state

 extern motor_state_t motor_state; }
 extern uint8_t motor_read_flag;

 uint8_t CAN_Send_Msg(uint32_t id, uint8_t *msg, uint8_t len);; }
 void fdcan_filter_init(FDCAN_HandleTypeDef *fdcanHandle);

 void motor_control_Pos(uint8_t id, int32_t pos, int16_t tqe);
 void motor_control_Vel(uint8_t id, int16_t vel, int16_t tqe); void motor_control_tqe(uint8_t id,
int16_t vel, int16_t tqe).
 void motor_control_tqe(uint8_t id, int32_t tqe);; void motor_control_pos_tqe(uint8_t id, int32_t
tqe)
 void motor_control_pos_val_tqe(uint8_t id, int16_t pos, int16_t val, int16_t tqe); void
motor_read(uint8_t id, int16_t vel, int16_t tqe)

 void motor_read(uint8_t id).

 #endif
```

libelybot_can.c

```
#include "libelybot_can.h"
 #include "fdcan.h"
 #include <string.h
```

```
FDCAN_RxHeaderTypeDef fdcan_rx_header1; fdcan_rx_header1; fdcan1_rdata[24] = {0
uint8_t fdcan1_rdata[24] = {0};

motor_state_t motor_state; uint8_t motor_read
uint8_t motor_read_flag = 0;

uint8_t CAN_Send_Msg(uint32_t id, uint8_t *msg, uint8_t len)
{
FDCAN_TxHeaderTypeDef TxHeader; uint8_t TxData; uint8_t TxData; uint8_t TxData
uint8_t TxData[8] = {0};

TxHeader.Identifier = id; // set the extension ID
TxHeader.IdType = FDCAN_EXTENDED_ID; //Use the extension ID
TxHeader.TxFrameType = FDCAN_DATA_FRAME; //data frame
TxHeader.DataLength = FDCAN_DLC_BYTES_8; //data length
TxHeader.ErrorStateIndicator = FDCAN_ESI_ACTIVE; // Error indication state
TxHeader.BitRateSwitch = FDCAN_BRS_OFF; // Bit rate switching off, not applicable to classic CAN
TxHeader.FDFormat = FDCAN_CLASSIC_CAN; // Classical CAN format
TxHeader.TxEventFifoControl = FDCAN_NO_TX_EVENTS; // not applicable to send event FIFOs
TxHeader.MessageMarker = 0; // Message Marker

// Copy data to send buffer
for(int i = 0; i < len; i++)
{
TxData[i] = msg[i];
}

// Send the CAN command
if(HAL_FDCAN_AddMessageToTxFifoQ(&hfdcan1, &TxHeader, TxData) ! = HAL_OK)
{
// Send failure handling
Error_Handler(); return 1; // return a non-zero value.
return 1; // Return a non-zero value to indicate a failure to send.
}
return 0; // Successful send
}

void fdcan_filter_init(FDCAN_HandleTypeDef *fdcanHandle)
{
if (HAL_FDCAN_ConfigGlobalFilter(fdcanHandle, FDCAN_ACCEPT_IN_RX_FIFO0,
FDCAN_ACCEPT_IN_RX_FIFO0, FDCAN_FILTER_REMOTE, FDCAN_FILTER_ REMOTE) ! = HAL_OK)
{
Error_Handler();
}

if (HAL_FDCAN_ActivateNotification(fdcanHandle, FDCAN_IT_RX_FIFO0_NEW_MESSAGE |
FDCAN_IT_TX_FIFO_EMPTY, 0) ! = HAL_OK)
{
Error_Handler();
}
HAL_FDCAN_ConfigTxDelayCompensation(fdcanHandle, fdcanHandle->Init.DataPrescaler *
fdcanHandle->Init.DataTimeSeg1, 0);
HAL_FDCAN_EnableTxDelayCompensation(fdcanHandle);

if (HAL_FDCAN_Start(fdcanHandle) ! = HAL_OK)
{
Error_Handler();
}

//HAL_FDCAN_Start(fdcanHandle).
}

/**
* @brief Position control
* @param id motor id
* @param pos Position: unit: 0.0001 turns, e.g. pos = 5000 means turn to 0.5 turns position.
* @param torque: unit: 0.01 NM, e.g. torque = 110 means the maximum torque is 1.1NM
*/
void motor_control_Pos(uint8_t id, int32_t pos, int16_t tqe)
{
uint8_t tdata[8] = {0x07, 0x07, 0x0A, 0x05, 0x00, 0x00, 0x80, 0x00};

*(int16_t *)&tdata[2] = pos.
*(int16_t *)&tdata[6] = tqe;

uint32_t ext_id = (0x8000 | id);
CAN_Send_Msg(ext_id, tdata, 8);
}

/**
* @brief Speed control
* @param id Motor ID
* @param vel velocity: unit 0.00025 rpm, e.g. val = 1000 means 0.25 rpm
* @param tqe torque: unit: 0.01 NM, e.g. torque = 110 means maximum torque is 1.1NM
*/
void motor_control_Vel(uint8_t id, int16_t vel, int16_t tqe)
{
uint8_t tdata[8] = {0x07, 0x07, 0x00, 0x80, 0x20, 0x00, 0x80, 0x00};

*(int16_t *)&tdata[4] = vel.
*(int16_t *)&tdata[6] = tqe;

uint32_t ext_id = (0x8000 | id);
CAN_Send_Msg(ext_id, tdata, 8);
}

/**
* @brief Torque mode
* @param id Motor ID
* @param tqe Torque: unit: 0.01 NM, e.g. torque = 110 means max torque is 1.1NM.
/*
void motor_control_tqe(uint8_t id, int32_t tqe)
{
uint8_t tdata[8] = {0x05, 0x13, 0x00, 0x80, 0x20, 0x00, 0x80, 0x00};
```

```
*(int16_t *)&tdata[2] = tqe;

CAN_Send_Msg(0x8000 | id, tdata, 4);
}

/**
* @brief motor position-speed-feedforward torque (max torque) control, int16 type
* @param id Motor ID
* @param pos Position: unit 0.0001 turns, e.g. pos = 5000 means turn to 0.5 turns position.
* @param val speed: unit 0.00025 revolutions/second, such as val = 1000 that 0.25 revolutions/second
* @param tqe Maximum torque: unit: 0.01 NM, such as torque = 110 that the maximum torque
is 1.1NM
/*
void motor_control_pos_val_tqe(uint8_t id, int16_t pos, int16_t val, int16_t tqe)
{
static uint8_t tdata[8] = {0x07, 0x35, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};

*(int16_t *)&tdata[2] = val.
*(int16_t *)&tdata[4] = tqe;
*(int16_t *)&tdata[6] = pos;

CAN_Send_Msg(0x8000 | id, tdata, 8);
}

/**
* @brief Read motor
* @param id
*/
void motor_read(uint8_t id)
{
static uint8_t tdata[8] = {0x17, 0x01};

CAN_Send_Msg(0x8000 | id, tdata, sizeof(tdata));
}

static uint8_t Fdcan_Dlc_To_Len(uint32_t dlc)
{
uint8_t len = 0;
uint8_t tab_dlc_to_len[] = {12, 16, 20, 24, 32, 48, 64};

if (dlc <= FDCAN_DLC_BYTES_8)
{
len = dlc >> 16;
}
else
{
len = tab_dlc_to_len[(dlc >> 16) - 9]; }
}

return len; }
}
```

void HAL_FDCAN_RxFifo0Callback(FDCAN_HandleTypeDef *hfdcan, uint32_t RxFifo0ITs) *// FDCAN FIFO 0 callback function*
{
uint8_t len = 0;
if(hfdcan->Instance == FDCAN1)
{

```
  HAL_FDCAN_GetRxMessage(hfdcan, FDCAN_RX_FIFO0, &fdcan_rx_header1,
 fdcan1_rdata);
  if (fdcan_rx_header1.DataLength ! = 0)
  {
```

len = Fdcan_Dlc_To_Len(fdcan_rx_header1.DataLength);
motor_state.motor.id = fdcan_rx_header1.Identifier; *// get the motor id*
memcpy(&motor_state.data[4], &fdcan1_rdata[2], len - 2); *// get motor state data*
motor_read_flag = 1; }
}
}
}

main. c

int main(void)
{
/ USER CODE* BEGIN *1 */*
uint32_t tick_500ms = 0;
/ USER CODE* END *1 */*

/ MPU Configuration--*/*
MPU_Config();

/ MCU Configuration--*/ /* MPU_Config()*

/ Reset of all peripherals, Initializes the Flash interface and the Systick. */*
HAL_Init(); /* USER CODE BEGIN

/ USER CODE* BEGIN *Init */ /* USER CODE END Init */ /* HAL_Init()*

HAL_Init(); / USER CODE BEGIN Init */; /* USER CODE* END *Init */*

/ Configure the system clock */*
SystemClock_Config();

/ USER CODE* BEGIN *SysInit */ /* USER CODE END SysInit*

/ USER CODE* END *SysInit */ /* Configure the system clock */ SystemClock_Config()*

/ USER CODE END SysInit */ /* Initialize all configured peripherals */*
MX_GPIO_Init().
MX_GPIO_Init(); MX_FDCAN1_Init(); /* USER CODE END SysInit
MX_GPIO_Init(); MX_FDCAN1_Init(); MX_USART1_UART_Init(); MX_USART3_UART_Init()
MX_USART3_UART_Init();
/ USER CODE* BEGIN *2 */*

```
fdcan_filter_init(&hfdcan1).
// if(HAL_FDCAN_Start(&hfdcan1) ! = HAL_OK)
// {
// // Startup failure management
// Error_Handler(); // if(HAL_FDCAN_Start(&hfdcan1) !
// }
motor_control_pos_val_tqe(1, INT16_NAN, 1000, 100);
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
if (HAL_GetTick() - tick_500ms >= 2000)
tick_500ms = HAL_GetTick( - tick_500ms >= 2000) {
tick_500ms = HAL_GetTick(); //motor_control_pos_tick() - tick_500ms >= 2000) {
//motor_control_pos_val_tqe(1, INT16_NAN, 1000, 100);
HAL_GPIO_TogglePin(LED1_GPIO_Port, LED1_Pin);
DEBUG_PRINT("AAA");
motor_read(1);
}

if (motor_read_flag == 1)
motor_read_flag = 0; } if (motor_read_flag == 1)
motor_read_flag = 0; } if (motor_read_flag == 1) { motor_read_flag = 0; }
DEBUG_PRINT("motor %x %d %d %d %d",motor_state.motor.id, motor_state.motor.position,
motor_state.motor.velocity, motor_state.motor.torque);
}
/* USER CODE END WHILE */
```

```
*/\* USER CODE* BEGIN *3 \*/*
```

```
}
/* USER CODE END 3 */
}
```