

## 目录

### 目录 1

一、新建一个 ROS 工作空间 .....	2
1. 创建 catkin 工作空间 .....	2
2. 编译 catkin 工作空间 .....	2
3. 配置环境变量 .....	2
4. 创建工作包 .....	2
二、拷贝电机控制相关功能包并重新编译 .....	2
1. 替换 catkin_ws/src 文件夹 .....	2
2. 编译整个功能包 .....	2
三、修改并运行示例程序 .....	2
1. 修改 catkin_ws/src/drempower/src/motor_test_publisher.cpp 文件 .....	3
2. 修改 catkin_ws/src/drempower/src/xx_subscriber.cpp 文件 .....	3
3. 修改 catkin_ws/src/drempower/src/motor_control_test.launch 文件 .....	4
四、重新编译整个功能包 .....	4
五、启动 launch 文件 .....	4
六、使用 rqt_plot 查看电机运行曲线图 .....	5
七、使用 rqt_graph 查看节点及消息图 .....	7
八、电机控制相关 Canopen 对象字典表 .....	8
1. 自定义对象字典表 .....	8
2. 控制模式与 RPDO 配置 .....	10
3. 控制模式与 TPDO 配置 .....	12
4. PDO 配置步骤流程 .....	13

## 一、新建一个 ROS 工作空间

### 1. 创建 catkin 工作空间

```
$ mkdir -p ~/catkin_ws/src
```

### 2. 编译 catkin 工作空间

```
$ cd ~/catkin_ws
```

```
$ catkin_make
```

### 3. 配置环境变量

```
$ echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
```

### 4. 创建工作包

```
$ cd ~/catkin_ws/src
```

```
$ catkin_create_pkg drempower std_msgs roscpp rospy
```

## 二、拷贝电机控制相关功能包并重新编译

### 1. 替换 catkin\_ws/src 文件夹

●将电机 ROS 库函数文件夹下的 DrEmpower\_ws/src 文件夹整个复制到 ~/catkin\_ws 路径下，替换原有的 src 文件夹；

### 2. 编译整个功能包

```
$ cd ~/catkin_ws/
```

```
$ catkin_make
```

## 三、修改并运行示例程序

●在示例程序中提供了单个电机 7 种控制模式测试，实际使用时根据实际情况修改测试的电机 ID 号和测试模式：

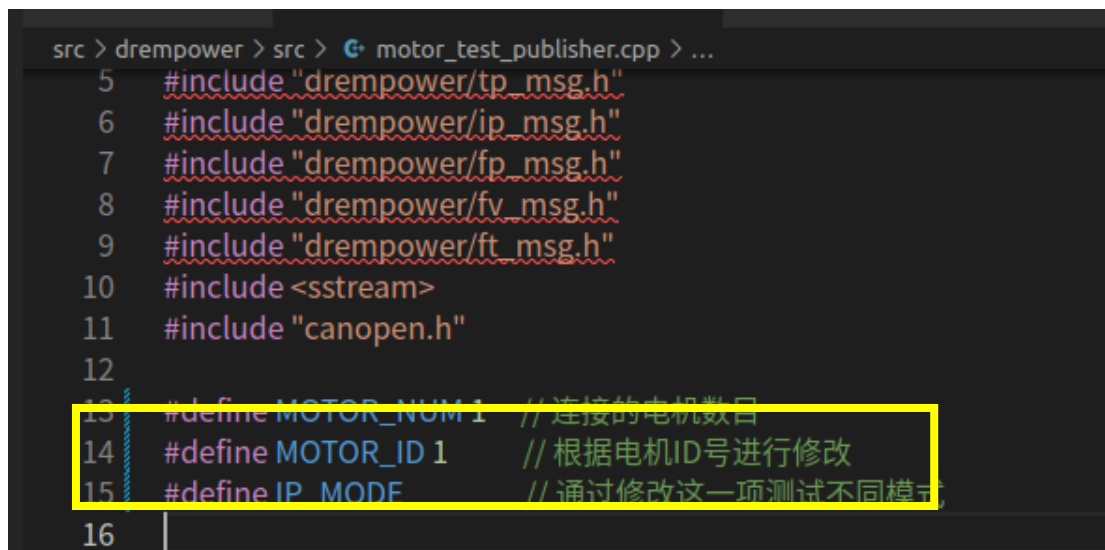
	模式名称	模式简称	数字代号	注释
位置控制	轨迹跟踪模式	TP	7	
	梯形轨迹模式	PP	1	
	前馈控制模式	FP	-1	
速度控制	速度爬升模式	PV	3	
	速度前馈模式	FV	-3	

扭矩控制	扭矩爬升模式	PT	4	
	直接控制模式,	FT	-4	
	阻抗控制模式	IP	-7	

以上 7 种控制模式更详细的介绍请参考产品手册及使用说明书中的《DrEmpower 系列电机使用手册 v1.0》

## 1. 修改 catkin\_ws/src/drempower/src/motor\_test\_publisher.cpp 文件

●根据实际电机 ID 号及想要测试的模式修改第 14、15 行，如下图所示：



```

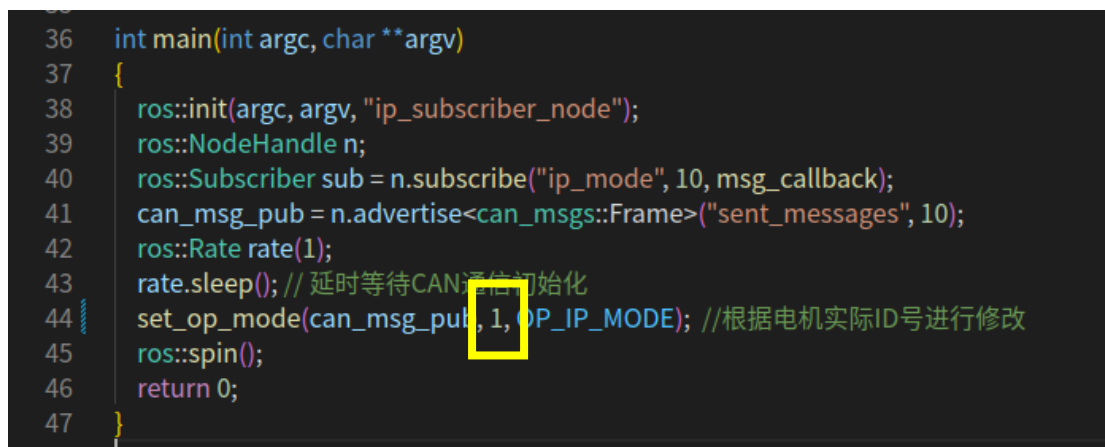
src > drempower > src > motor_test_publisher.cpp > ...
5  #include "drempower/tp_msg.h"
6  #include "drempower/ip_msg.h"
7  #include "drempower/fp_msg.h"
8  #include "drempower/fv_msg.h"
9  #include "drempower/ft_msg.h"
10 #include <sstream>
11 #include "canopen.h"
12
13 #define MOTOR_NUM 1 // 连接的电机数目
14 #define MOTOR_ID 1 // 根据电机ID号进行修改
15 #define IP_MODE // 通过修改这一项测试不同模式
16

```

图 1 修改测试节点文件中的电机 ID 及控制模式

## 2. 修改 catkin\_ws/src/drempower/src/xx\_subscriber.cpp 文件

●根据上图中选择的模式，修改对应的 xx\_subscriber.cpp 文件，例如上图中我们选择测试 IP\_MODE(阻抗控制模式)，则这一步需要修改 ip\_subscriber.cpp 文件，修改处位于倒数第 4 行，将其中的数字改成实际的电机 ID 号；



```

36 int main(int argc, char **argv)
37 {
38     ros::init(argc, argv, "ip_subscriber_node");
39     ros::NodeHandle n;
40     ros::Subscriber sub = n.subscribe("ip_mode", 10, msg_callback);
41     can_msg_pub = n.advertise<can_msgs::Frame>("sent_messages", 10);
42     ros::Rate rate(1);
43     rate.sleep(); // 延时等待CAN通信初始化
44     set_op_mode(can_msg_pub, 1, OP_IP_MODE); //根据电机实际ID号进行修改
45     ros::spin();
46     return 0;
47 }

```

图 2 修改对应控制模式协议解析节点中的电机 ID 号

### 3. 修改 catkin\_ws/src/drempower/src/motor\_control\_test.launch 文件

●根据图 1 中选择的模式，将对应控制模式协议解析节点 xx\_subscriber\_node 节点注释删掉，同时注释掉其他控制模式协议解析节点(快捷键 Ctrl+/-)；



```

13
14  -->
15  <node pkg="socketcan_bridge" type="socketcan_bridge_node" name="socketcan_bridge_node" output="screen">
16  </node>
17  <node pkg="drempower" type="state_subscriber_node" name="state_subscriber_node" output="screen">
18  </node>
19  <!-- <node pkg="drempower" type="tp_subscriber_node" name="tp_subscriber_node" output="screen">
20  </node> -->
21  <!-- <node pkg="drempower" type="pp_subscriber_node" name="pp_subscriber_node" output="screen">
22  </node> -->
23  <!-- <node pkg="drempower" type="fp_subscriber_node" name="fp_subscriber_node" output="screen">
24  </node> -->
25  <!-- <node pkg="drempower" type="pv_subscriber_node" name="pv_subscriber_node" output="screen">
26  </node> -->
27  <!-- <node pkg="drempower" type="fv_subscriber_node" name="fv_subscriber_node" output="screen">
28  </node> -->
29  <!-- <node pkg="drempower" type="pt_subscriber_node" name="pt_subscriber_node" output="screen">
30  </node> -->
31  <!-- <node pkg="drempower" type="ft_subscriber_node" name="ft_subscriber_node" output="screen">
32  </node> -->
33  <node pkg="drempower" type="ip_subscriber_node" name="ip_subscriber_node" output="screen">
34  </node>
35  <node pkg="drempower" type="motor_test_node" name="motor_test_node" output="screen">
36  </node>
37  </launch>

```

图 3 在 launch 文件中启动对应控制模式协议解析节点

■**注意：**一次只能同时测试一种控制模式，且不同模式协议解析节点只能同时启动一个；

### 四、重新编译整个功能包

```
$ cd ~/catkin_ws/
```

```
$ catkin_make
```

### 五、启动 launch 文件

```
$ roslaunch drempower motor_control_test.launch
```

■**注意：**启动 launch 文件前需要先将电机供电，同时将 USB 转 CAN 模块连接好电脑和电机，如图 4 所示（必须使用图中所示的 USB 转 CAN 模块）；

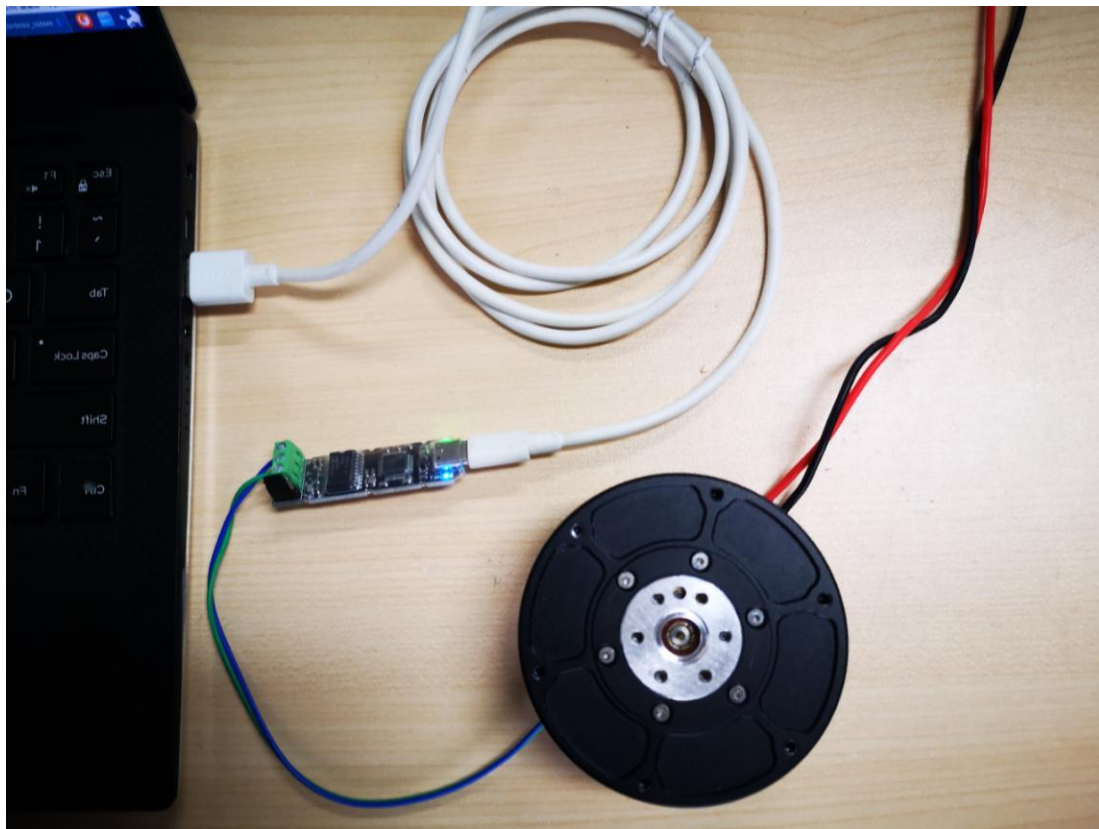
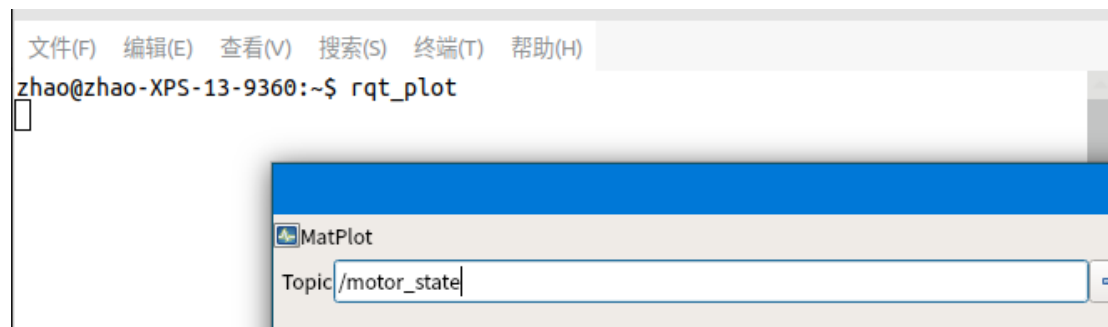


图 4 电机接线图

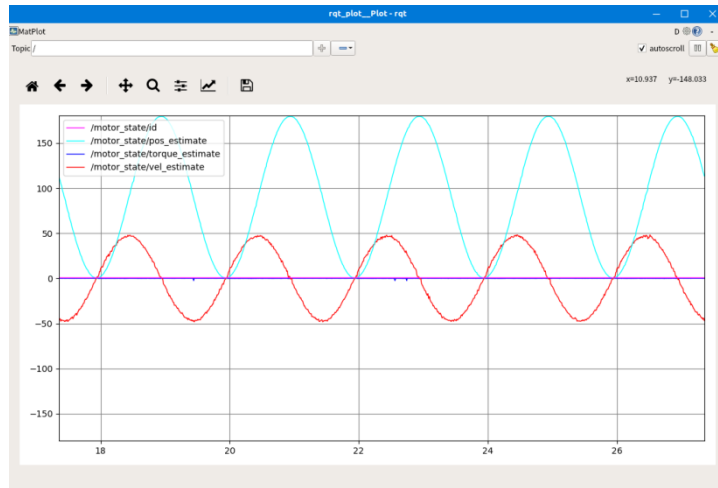
## 六、使用 `rqt_plot` 查看电机运行曲线图

- 轨迹跟踪模式 (TP) 下，TPDO1 配置为循环同步模式，每接收到一个 SYNC 返回一次电机状态；
- 其他 6 种模式下，TPDO1 配置成异步模式，每 10ms 返回一次电机状态；
- 在示例程序中，电机自动返回电机状态数据，会被 `state_subscriber_node` 接收解析，并向外发送 Topic(`motor_state`)，该 topic 采用 `state.msg`，包含电机 id 号，实时电机位置 (度)、电机速度 (r/min)、电机扭矩(Nm)；
- 重新开启一个 terminal (快捷键 `Ctrl+alt+T`)，输入 `rqt_plot` 指令，在弹出的窗口里选择 topic(`motor_state`)即可，如图 5 所示；

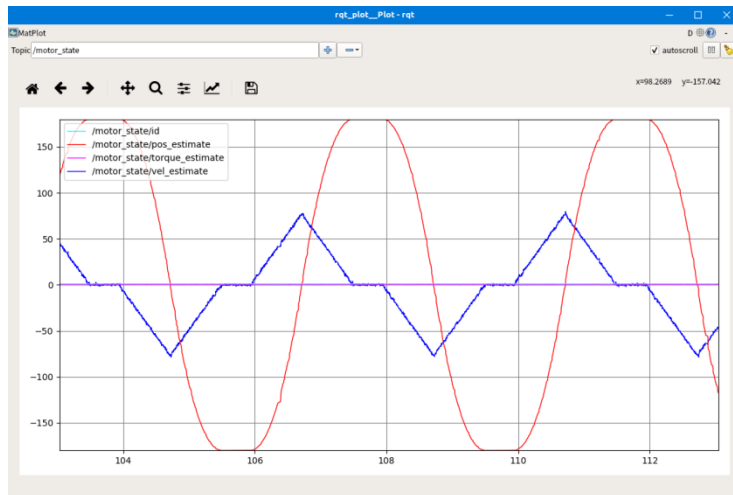
```
$ rqt_plot
```

图 5 启动 `rqt_plot` 并选择 `motor_state` 话题

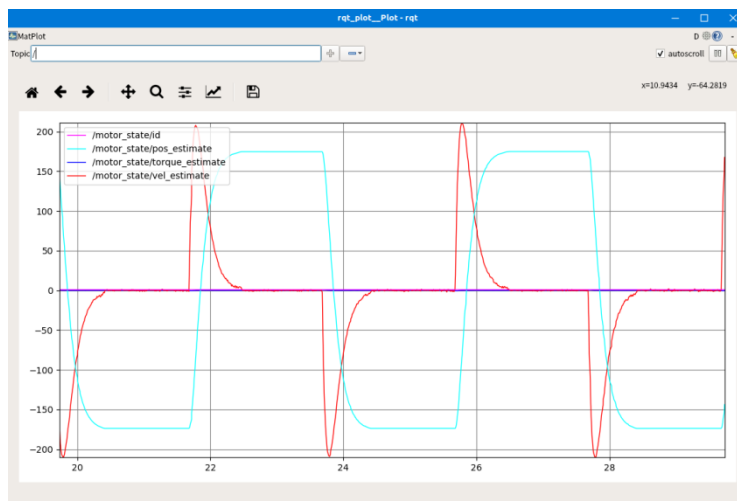
●轨迹插补模式、梯形轨迹模式、阻抗控制模式下电机运行曲线图如图 6 所示：



(a) 轨迹跟踪模式



(b) 迹跟踪模式



(c) 阻抗控制模式

图 6 三种控制模式下电机运行状态曲线图

## 七、使用 rqt\_graph 查看节点及消息图

●重新开启一个 terminal（快捷键 Ctrl+alt+T），输入 rqt\_graph 指令即可得到当前 ROS 节点及节点间消息交互动态图形，如图 6 所示

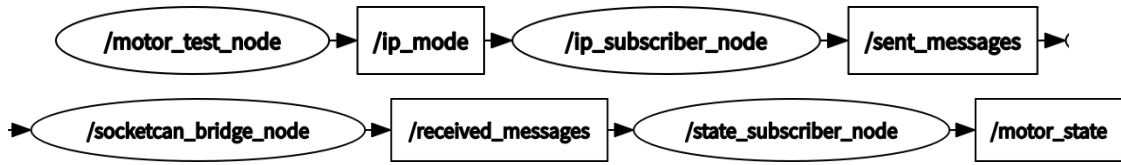


图 6 示例程序下 ROS 节点及消息动态图（有改动）

●从图中可以看出整个示例程序中存在 4 个节点：motor\_test\_node、ip\_subscriber\_node、socketcan\_bridge\_node 及 state\_subscriber\_node。其中：

1. motor\_test\_node 节点对应库文件 motor\_test\_publisher.cpp，其主要作用发送对应的控制指令，图中测该节点按照 ip\_msg.msg 中规定消息类型发送控制指令 ip\_node 消息；

2. ip\_subscriber\_node 节点对应库文件中 ip\_subscriber.cpp，其主要作用是将 ip\_msg.msg 格式的 ip\_node 消息根据 canopen 通信协议解析成 can\_msgs/Frame.msg 格式的 sent\_message 消息；

3. socketcan\_bridge\_node 节点是 ROS 标准库 Ros\_canopen 库里的子节点，主要有两个作用，一是将 can\_msgs/Frame.msg 格式的 sent\_message 消息通过 socketcan 设备发送到 CAN 总线上；另外一个作用是将 socketcan 设备接收的 CAN 数据转换成 can\_msgs/Frame.msg 格式的 received\_message 消息；

4. state\_subscriber\_node 节点对应库文件中 state\_subscriber.cpp，其主要作用是解析由 socketcan\_brideg\_node 节点转换后的 received\_message 消息，这些消息是由电机发送给上位机，包括电机运行状态等信息。state\_subscriber\_node 节点将解析后的电机运行状态转换成 state\_msg.msg 格式的 motor\_state 消息，该消息可以用 rqt\_plot 工具绘制实时电机运行曲线，也可以用在用户程序里来了解电机实时运行状态等；

●其他控制模式也类似，只需将前两个节点对应换成对应的控制模式即可，后面两个节点无须做任何改变；每种控制模式都有一个对应的协议解析节点 xx\_subscriber\_node(对应 xx\_subscriber.cpp),当需要使用某种控制模式时，只需在 launch 文件中打开对应 xx\_subscriber\_node 节点，同时屏蔽掉其他控制模式的 xx\_subscriber\_node 节点，然后将 motor\_test\_publisher.cpp 文件中的模式宏定义（第 15 行）相应修改即可；

## 八、电机控制相关 Canopen 对象字典表

●电机 CAN 通信协议有两个版本，一个为标准版本，另一个为 Canopen 版本；

本说明书针对 Canopen 协议版本进行说明；

●Canopen 协议包含两部分内容，一部分是标准的 DS301（又名 CIA301），这部分本说明书不做过多解释，用户可参考《CANopen 轻松入门》；

●与电机控制直接相关部分，原本也有一套标准协议 CIA402，但是由于其支持的控制模式及数据类型比较固定，下面自定义了一套对象字典，用于实现电机支持的 7 种控制模式，具体介绍如下：

### 1. 自定义对象字典表

Index 索引	Name 名称	Data type 数据类型	只读/ 读写	单位	参数范围	参数说明
0x2101	input_pos 用户指令位置	Float32	读/写	°	软件限位 范围	
0x2102	input_vel 用户指令速度	Int16	读/写	r/min 0.01	-327~327	
0x2103	input_torque 用户指令扭矩	Int16	读/写	Nm 0.01	-327~327	
0x2111	pos_setpoint 目标位置	Float32	只读	°	软件限位 范围	
0x2112	vel_setpoint 目标速度	Int16	只读	r/min 0.01	-327~327	
0x2113	torque_setpoint 目标扭矩	Int16	只读	Nm 0.01	-327~327	
0x2121	pos_estimate 当前实际位置	Float32	只读	°	软件限位 范围	
0x2122	vel_estimate 当前实际速度	Int16	只读	r/min 0.01	-327~327	
0x2123	torque_estimate 当前实际扭矩	Int16	只读	Nm 0.01	-327~327	
0x2131	impedance_kp 阻抗位置增益	Uint16	读写	Nm/rad 0.01	0~655	IP 模式
0x2132	impedance_kd 阻抗速度增益	Uint16	读写	Nm/rad/s 0.01	0~655	IP 模式
0x2141	tt_vel_limit	Int16	读写	r/min	0~327	PP 模式



	轮廓规划速度			0.01		
0x2142	tt_accel_limit 轮廓规划加速度	Int16	读写	r/min/s 0.01	0~327	PP 模式
0x2143	tt_decel_limit 轮廓规划减速度	Int16	读写	r/min/s 0.01	0~327	PP 模式
0x2151	vel_ramp_rate 速度斜率	Int16	读写	r/min/s 0.01	0~327	PV 模式
0x2152	torque_ramp_rate 扭矩斜率	Int16	读写	Nm/s 0.01	0~327	PT 模式
0x2161	interpolation_freq 轨迹插补频率	Uint16	读写	Hz 0.01	0~655	TP 模式
0x6040	Controlword 控制字	Uint16	读写	-	0~65535	
0x6041	Statuword 状态字	Uint16	只读	-	-	
0x6060	Modes of operation 模式选择	Int8	读写	-	-7~7	
0x6061	Modes of operation display 模式显示	Int8	只读	-	-7~7	

**注:**

1. 单位列中部分单位下 0.01 表示最终传输值=实际值/0.01，例如需要将 vel\_ramp\_rate 设置为 10.5r/min/s，则实际传输的为 int16 类型，这时会将 10.5 转换成 1050（10.5/0.01=1050）进行传输，电机收到后也会将 1050\*0.01=10.5（因为 CAN 标准帧中数据长度只有 64 位，如果需要同时传输三个浮点数，后两位需要通过 16 位 signed int 进行传输）；
2. 上述所有字典对象中的参数都是针对一体化关节输出轴，即所有参数都是过了减速器之后的参数，用户无需再进行二次换算；

## 2. 控制模式与 RPDO 配置

- 为了提高不同控制模式 CAN 通信效率, 建议使用 PDO 来实现各种模式控制;
- PP 模式和 IP 模式参数比较多, 无法使用 RPDO1 传输完, 故单独额外分配了一个 RPDOx 用于传输模式参数, 实际使用是需**先发送 RPDOx, 再发送 RPDO1**;
- RPDO4 固定用于传输控制模式 (modes of operation, Index = 0x6040), 配置好所有 RPDO 后, 需要首先发送一条 RPDO4 设置控制模式 (Mode of operation);
- RPDO 映射配置需要和设置的控制模式 (Mode of operation) 对应, 否则电机无法正常转动; 例如使用 TP 模式, 则需要将 RPDO1 映射到 input\_pos 和 interpolation\_freq, 并将 Modes of operation 设置为 7;
- 所有控制模式下 RPDO1 传输模式均被配置成循环同步模式, 即电机接收到 RPDO1 后, 不会立即响应, 必须再接收到一帧同步报文后才会响应这条 RPDO1 指令 (主要是用于同步多个电机控制);

表 2 控制模式-RPDO 配置表

TP 模式-轨迹跟踪模式(Modes of operation = 7)				
变量名称	映射索引值	映射地址 Index-SubIndex	RPDO 分配	传输模式
input_pos	0x21010020	0x1600-0x01	RPDO1	0x01
interpolation_freq	0x21610010	0x1600-0x02		(循环同步)
PP 模式-梯形轨迹模式(Modes of operation = 1)				
变量名称	映射索引值	映射地址 Index-SubIndex	RPDO 分配	传输模式
input_pos	0x21010020	0x1600-0x01	RPDO1	0x01
tt_vel_limit	0x21410010	0x1600-0x02		(循环同步)
tt_accel_limit	0x21420010	0x1601-0x01	RPDO2	0xFF
tt_accel_limit	0x21430010	0x1601-0x02		(异步)
FP 模式-前馈控制模式(Modes of operation = -1)				
变量名称	映射索引值	映射地址 Index-SubIndex	RPDO 分配	传输模式
input_pos	0x21010020	0x1600-0x01	RPDO1	0x01 (循环同步)
input_vel	0x21020010	0x1600-0x02		
input_torque	0x21030010	0x1600-0x03		
PV 模式-梯形轨迹模式(Modes of operation = 3)				
变量名称	映射索引值	映射地址 Index-SubIndex	RPDO 分配	传输模式

input_vel	0x21020020	0x1600-0x01	RPDO1	0x01
vel_ramp_rate	0x21510010	0x1600-0x02		(循环同步)
FV 模式-梯形轨迹模式(Modes of operation = -3)				
变量名称	映射索引值	映射地址 Index-SubIndex	RPDO 分配	传输模式
input_vel	0x21020020	0x1600-0x01	RPDO1	0x01
input_torque	0x21030010	0x1600-0x02		(循环同步)
PT 模式-梯形轨迹模式(Modes of operation = 4)				
变量名称	映射索引值	映射地址 Index-SubIndex	RPDO 分配	传输模式
input_torque	0x21030020	0x1600-0x01	RPDO1	0x01
vel_torque_rate	0x21520010	0x1600-0x02		(循环同步)
FT 模式-梯形轨迹模式(Modes of operation = -4)				
变量名称	映射索引值	映射地址 Index-SubIndex	RPDO 分配	传输模式
input_torque	0x21030020	0x1600-0x01	RPDO1	0x01 (循环同步)
IP 模式-阻抗控制模式(Modes of operation = -7)				
变量名称	映射索引值	映射地址 Index-SubIndex	RPDO 分配	传输模式
input_pos	0x21010020	0x1600-0x01	RPDO1	0x01 (循环同步)
input_vel	0x21020010	0x1600-0x02		
input_torque	0x21030010	0x1600-0x03		
impedance_kp	0x21310010	0x1602-0x01	RPDO3	0xFF
impedance_kd	0x21320010	0x1602-0x02		(异步)
所有控制模式下 RPDO4 配置				
变量名称	映射索引值	映射地址 Index-SubIndex	RPDO 分配	传输模式
Modes of operation	0x60400008	0x1603-0x01	RPDO4	0xFF (异步)

### 3. 控制模式与 TPDO 配置

●如果需要实时获取电机运行状态，可以配置 TPDO 将电机状态（当前位置、当前速度、当前扭矩）发送到上位机；如果不需要获取电机运行状态，也可以不配置 TPDO（默认关闭）。

●TPDO 传输模式常用的主要有两种：循环同步模式(0x01)和异步模式(0xFF)；

●循环同步模式（0x01）表示每接收到一帧同步报文，发送一次 TPDO(也可以配置成 0x0N,表示接收到 N 帧同步报文后，发送一次 TPDO)；

●异步模式(0xFF)一般结合 TPDO 通信参数中的 Event timer (Subindex = 0x05) 用于实现电机定时反馈电机状态，例如异步模式下，将 Event timer 设置成 0x000A，则电机每 10ms(100Hz)发送一次 TPDO；

●TPDO 传输模式(Transmission type)具体使用哪种模式，取决于上位机控制发送控制指令的频率。如果是轨迹跟踪（插补）模式（TP），一般上位机会按照较高频率（100Hz 以上），这时可以配置成循环同步模式，每接收到一帧同步帧，往上位机发送一次 TPDO；如果是梯形轨迹模式（PP），一般发送指令频率不会太高，这时如果想要得到电机运行状态曲线，需要将 TPDO 配置成异步模式（0xFF），设置合适的 Event timer（一般不能小于 3），让电机较高频率定时发送 TPDO，反馈电机状态；

表 3 控制模式 TPDO 配置表

TP 模式-轨迹跟踪模式(Modes of operation = 7) – TPDO 循环同步				
变量名称	映射索引值	映射地址 Index-SubIndex	RPDO 分配	传输模式
pos_estimate	0x21210020	0x1600-0x01	TPDO1	0x01 (循环同步)
vel_estimate	0x21220010	0x1600-0x02		
torque_estimate	0x21230010	0x1600-0x03		
其他控制模式 – TPDO 异步				
变量名称	映射索引值	映射地址 Index-SubIndex	RPDO 分配	传输模式
pos_estimate	0x21210020	0x1600-0x01	TPDO1	0xFF (异步)
vel_estimate	0x21220010	0x1600-0x02		
torque_estimate	0x21230010	0x1600-0x03	Event timer = 0x000A	

#### 4. PDO 配置步骤流程

- 配置 PDO 一般有以下几个步骤：

1. 将 NMT 节点状态机切换到 PreOP 状态(此状态可用 SDO 来配置 PDO 映射)；
2. 失能 PDO (将 PDO 的通信参数中的 COB-ID 最高位置为 1)；
3. 清空 PDO 映射对象 (将 PDO 的映射参数个数 (SubIndex=0) 置为 0)；
4. 设置 PDO 映射对象(参照表 2 和表 3 将映射地址的值设置为对应索引值)；
5. 设置传输模式(如果是 TPDO 且为异步传输模式,还需要设置 Event timer)；
6. 使能 PDO 映射对象 (将 PDO 的映射参数个数 (SubIndex=0) 设置为实际映射参数个数)；
7. 使能 PDO (将 PDO 的通信参数中的 COB-ID 最高位置为 0)；
8. 将 NMT 节点状态机切换到 OP 状态 (配置的 PDO 映射将有效)；

- 下图是一个 RPDO 配置函数，可进行参考：

(该函数位于 Ros 库函数 DrEmpower\_ws/src/drempower/src/canopen.h 文件中)；

```
void config_rpdo(ros::Publisher pub, uint16_t id_num, uint8_t pdo_num, uint32_t *map_list, uint8_t len, uint8_t trans_type)
{
    set_nmt_state(pub, id_num, 0x7F); // 进入预操作模式-// "Pre-operational"
    uint8_t data[] = {uint8_t(id_num), pdo_num + 1, 0x00, 0x80};
    send_sdo(pub, id_num, 0x1400 + pdo_num - 1, 0x01, data, 4); // 失能PDO
    data[0] = 0x00;
    send_sdo(pub, id_num, 0x1600 + pdo_num - 1, 0x00, data, 1); // 清空PDO映射
    for (uint8_t i = 0; i < len; i++)
    {
        for (uint8_t j = 0; j < 4; j++)
        {
            data[j] = (map_list[i] >> (8 * j)) & 0xFF;
        }
        send_sdo(pub, id_num, 0x1600 + pdo_num - 1, i + 1, data, 4); // 设置PDO映射
    }
    data[0] = trans_type;
    send_sdo(pub, id_num, 0x1400 + pdo_num - 1, 0x02, data, 1); // 设置PDO传输类型
    data[0] = len;
    send_sdo(pub, id_num, 0x1600 + pdo_num - 1, 0x00, data, 1); // 使能PDO映射
    data[0] = uint8_t(id_num);
    data[1] = pdo_num + 1;
    data[2] = 0x00;
    data[3] = 0x00;
    send_sdo(pub, id_num, 0x1400 + pdo_num - 1, 0x01, data, 4); // 使能PDO
    set_nmt_state(pub, id_num, 0x05); // 进入操作模式- // "Operational"
}
```

图 8 RPDO 配置示例函数代码