第一章 关节电机

A1 机器人上安装有 12 个关节电机，这些关节电机就是机器人实现复杂运动的基础。我们后续开发的所有算法，最终都是生成了发送给这 12 个关节电机的命令。因此我们需要熟悉关节电机及其控制。

1.1 适用于四足机器人的动力系统

从四足机器人控制系统的角度来看，这个系统的输入量是每个关节的力矩。因此一个理想的机器人关节能够准确地输出我们希望的力矩，也就是力矩源 [1]。这个需求看似简单，但是实际上许多电机并不能做精确的控制（如异步电机），或者只能对角度做精确控制（如步进电机）。因此在四足机器人上，我们选用了一种特殊的电机：永磁同步电机（Permanent-Magnet Synchronous Motor, PMSM）。

1.1.1 永磁同步电机概述

关节电机中的永磁同步电机的定子是一个三相对称正弦波绕组，转子上粘贴有永磁体。我们知道，在一个固定的磁场中，永磁体会旋转并固定在一个平行于磁场的方向。一个典型的例子就是在地球磁场下的指南针，指南针会转动到指向地磁南北极的方向。同样的，如果这个固定的磁场开始转动，那么永磁体也会跟着转动，尽量跟随平行磁场方向，这样我们就可以通过旋转磁场来令永磁体旋转到指定角度。同时，永磁体在磁场中产生的力矩大小和永磁体与磁场方向的夹角有关，所以我们也可以通过控制磁场和永磁体的夹角来控制永磁体产生的力矩。

回到电机的角度来说，就是我们可以通过控制定子上三个绕组电压的大小与通断，来控制转子的角度位置和输出力矩。而这种控制方法就是永磁同步电机的矢量控制（Field-Oriented Control，以下简称 FOC）。

1.1.2 永磁同步电机的 FOC 控制简介

FOC 控制有许多独特的优势，它可以让我们对永磁同步电机进行“像素级”的控制，实现很多传统电机控制方法所无法达到的效果 [2]：

1. 可以在低转速下保持精确控制
2. 可以很好地实现电机换向旋转
3. 能够对电机进行力矩、速度、位置三个闭环控制
4. FOC 控制的永磁同步电机噪音较小

正如1.1.1节所述，FOC 控制的基础就是旋转磁场。在图（1.1）中，我们看到电机定子的三个线圈 a、b、c 可以产生三个方向的磁场 B_a、B_b、B_c，它们能合成电机中的磁场 B。FOC 控制器根据当前永磁体转子的角度、角速度、期望的输出力矩以及采样测量得到的 a、b、c 三个线圈的电流，计算得到三个线圈的电压通断状态与通断时间，进而通过

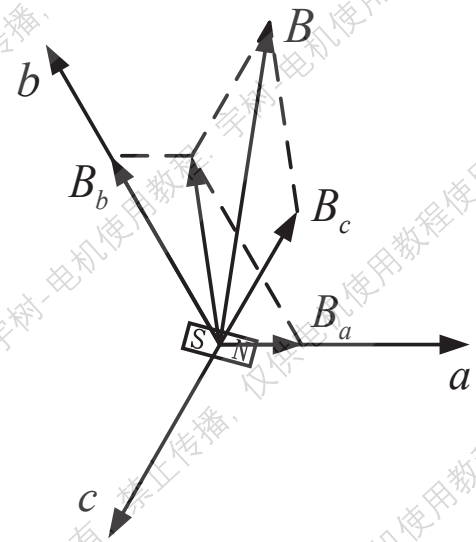

图 1.1: FOC 的磁场矢量

MOS 管来控制三个线圈的电压通断。这样就可以合成我们期望的磁场 B，进而拖动电机转子按照我们期望的方式运动。

1.2 A1 关节电机硬件概述

1.2.1 关节电机基本结构

关节电机的核心构件是电机驱动板、定子、转子和行星减速器。因为电机适合在高转速低力矩的工况下工作，而我们的机器人需要的是低转速高力矩，因此电机转子需要通过一个减速器减速之后，再输出力矩。A1 关节电机的大致结构如图（1.2）所示：

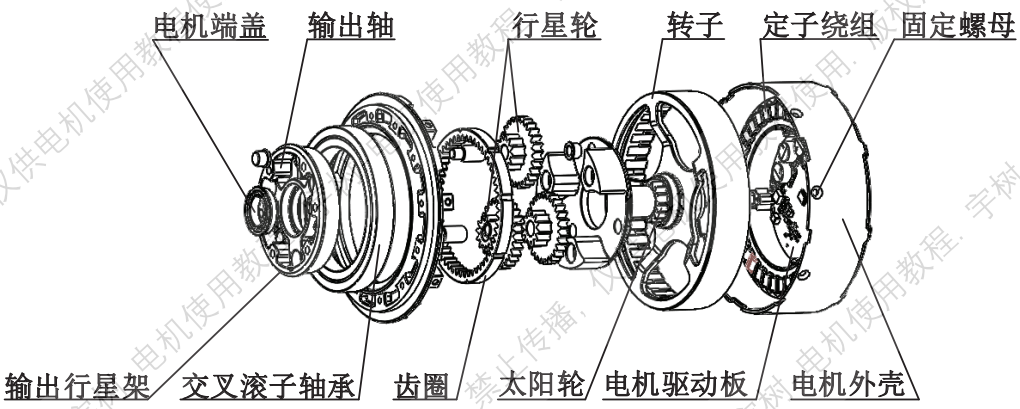

图 1.2: A1 关节电机主要构件

1.2.2 单圈绝对位置编码器

编码器（encoder）是用来测量旋转角度的传感器。编码器分为增量编码器、多圈绝对位置编码器与单圈绝对位置编码器等多种。我们在此只详细讨论关节电机实际应用的单圈绝对位置编码器。

关节电机的单圈绝对位置编码器安装在电机转子上。对于单圈绝对位置编码器（以下简称编码器），可以将它当做一个“时钟表盘”。我们每次看表的时候，都可以读到当前的日期和时间，例如 4 月 1 日 23 点。如果时间经过了 2 个小时，那么时钟转过了 4 月 1 日 24 点，日期会增加 1 天变成 4 月 2 日，时间重新从 0 点开始计时，变成了 4 月 2 日 1 点。为了方便计算经过的时间，我们也可以说现在是 4 月 1 日 25 点。看上去 25 点超过了一天 24 个小时的范围，实际上是因为日期增加了 1 天。

单圈绝对位置编码器也是同样的道理。每次开机上电后，我们的转子可能处于任意位置，编码器会告诉我们转子所处的角度位置（0 至 2π 之间的某个值）。如果转子旋转转过了 2π 这个角度位置，编码器也能够记录转过的圈数增加了 1，从而输出一个超出 0 至 2π 范围的角度位置。看上去编码器也能够输出超过一圈的角度位置，那么为什么叫它“单圈”绝对位置编码器呢？原因在于这种编码器在断电之后不能够储存之前旋转的圈数，我们以下面这个例子来说明。

假设当前编码器输出的角度值为 2.3π，这意味着编码器在开机之后经过了 2π，旋转圈数从 0 变为 1，同时编码器当前位于 0.3π 这个位置。此刻我们将编码器关机，不做任何旋转，再将编码器开机，那么此时编码器输出的角度值就会变为 0.3π。因为关机之后旋转圈数重置为 0，所以编码器只会输出当前位置 0.3π。

1.3 关节电机的混合控制

关节电机作为一个高度集成的动力单元，其内部已经封装了电机底层的控制算法。作为用户，只需要给关节电机发送相关的命令，电机就能完成从接收命令到关节力矩输出的全部工作。

对于电机的底层控制算法，唯一需要的控制目标就是输出力矩。可是对于机器人，我们通常需要给关节设定位置、速度和力矩。这时就需要对关节电机进行混合控制。

宇树科技的关节电机包含如下 5 个控制指令：

1. 前馈力矩：τ_{ff}
2. 期望角度位置：p_{des}
3. 期望角速度：ω_{des}
4. 位置刚度：k_p
5. 速度刚度（阻尼）：k_d

在关节电机的混合控制中，使用 PD 控制器将电机在输出位置的偏差反馈到力矩输出上：

$$\tau = \tau_{ff} + k_p \cdot (p_{des} - p) + k_d \cdot (\omega_{des} - \omega) \tag{1.1}$$

式 (1.1) 中，τ 为关节电机的电机转子输出力矩，p 为电机转子的当前角度位置，ω 为电机转子的角速度。在实际使用关节电机时，需要注意将电机输出端的控制目标量与发送的电机转子的指令进行换算。

1.4 宇树科技关节电机的线路连接

为了将上述 5 个控制指令发送给宇树科技制造的关节电机，我们需要通过串口将指令下发。宇树科技的关节电机通过 RS-485 接口与上位机进行通信，固定波特率 (Baud) 为 4.8MBd。为了方便用户使用，我们会提供 USB 转 RS-485 的转接口。

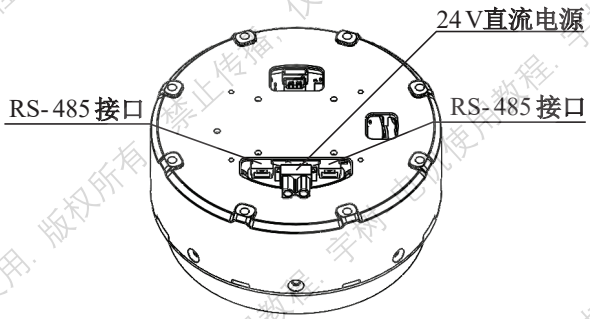

图 1.3: 关节电机线路连接

以四足机器人 A1 的关节电机为例，如图 (1.3) 所示。提供给用户的接口共有 3 个，其中中间的接口为 24V 直流电源接口。两侧的为 RS-485 接口，且这两个 RS-485 接口完全等价，因此可以使用 RS-485 线将电机串联（最多串联 3 个）同时控制。

在通过自己的电脑控制关节电机时，为了将指令从上位机发送到关节电机，需要将 RS-485 接口通过 USB 转 RS-485 转接口连接到上位机。在接通 24V 直流电源后，电机绿色指示灯开始闪烁，说明电机已开机。

1.5 关节电机的配置

由于所有的电机底层控制算法都已经整合在电机内部，因此上位机（通常为用户的计算机）只需要完成上层控制和 RS-485 串口的数据收发。为了方便用户对关节电机的操作，宇树科技提供了 RS-485 串口收发的软件开发工具包（Software Development Kit，以下简称 SDK），即`unitree_actuator_sdk`，该 SDK 位于配套代码的同名文件夹下。

`unitree_actuator_sdk`支持以下平台与系统：

1. x86/x64 平台下的 Linux 系统
2. ARM32/ARM64 平台下的 Linux 系统
3. x64 平台下的 Windows 系统

在每一个支持的系统下，`unitree_actuator_sdk`都提供 C、C++、Python 以及 ROS 的代码实例。用户只需要仿照实例就能完成对电机的控制。下面我们以 Linux 系统为例，演示如何控制电机。

1.5.1 查看串口名

将 USB 转 RS-485 转接口连接在上位机上时，上位机会为这个串口分配一个串口名。在 Linux 系统中，这个串口名一般是以“ttyUSB”开头，在 Windows 系统中，串口名往往以“COM”开头。

在 Linux 系统中，一切外接设备都是以文件形式存在的。USB 转 RS-485 转接口也可以被视为/dev 文件夹下的一个“文件”。打开任意一个终端窗口（在 Linux 下快捷键为 Ctrl+Alt+t 组合键），运行如下命令：

```
cd /dev
ls |grep ttyUSB
```

其中cd /dev命令将当前文件夹切换为/dev，ls |grep ttyUSB 命令显示当前文件夹下所有文件名包含ttyUSB的文件。运行如上命令后，即可得到上位机当前连接的串口名。例如图 (1.4) 中所示，当前上位机连接的串口名为ttyUSB0。考虑到串口所在的文件夹路径，其完整的串口名为 /dev/ttyUSB0。

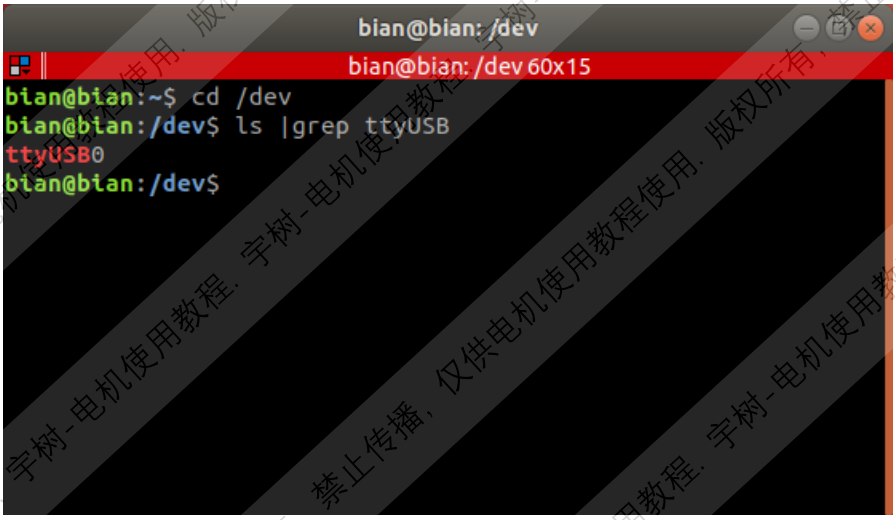

图 1.4: Linux 系统查看串口名

1.5.2 电机 ID 修改

每一个电机都需要分配一个 ID，同时上位机发送的每一条控制命令也包含一个 ID。电机只会执行 ID 与自己一致的控制命令。因此，当多个关节电机串联在同一条 RS-485 线路中时，为了分别控制其中的每一个电机，必须给每一个电机分配一个唯一的 ID。

在此说明一下电机修改 ID 的流程。首先通过上位机的广播模式，将 RS-485 串口下的所有关节电机切换到修改 ID 模式。此时所有电机的输出轴都变成了电子棘轮，所谓电子棘轮，即在旋转输出轴时，能感受到类似棘轮的明显的顿挫感。这时通过旋转输出轴，就能够将电机的 ID 设置为对应的值。最后再由上位机给电机发送保存 ID 的指令，即可完成电机 ID 的修改。

为了在上位机发送上述指令，我们提供了电机 ID 修改程序。该程序在unitree_actuator_sdk中的 ChangeID_Tools文件夹下。电机 ID 修改程序有 Linux 版与 Windows

版，下面我们来演示如何在 Linux 系统下修改电机的 ID。

首先在管理员权限下运行Linux文件夹下的可执行文件ChangeID：

```
sudo ./ChangeID
```

其中sudo的含义为以管理员权限运行，因此在按下回车之后需要输入管理员密码。请注意在 Linux 中，输入密码时并不会显示“***”等标识，只需要在密码输入完毕后按下回车即可。后面的 ./ChangeID表示执行当前文件夹下（./）的可执行文件 ChangeID。程序执行过程参见图 (1.5)。

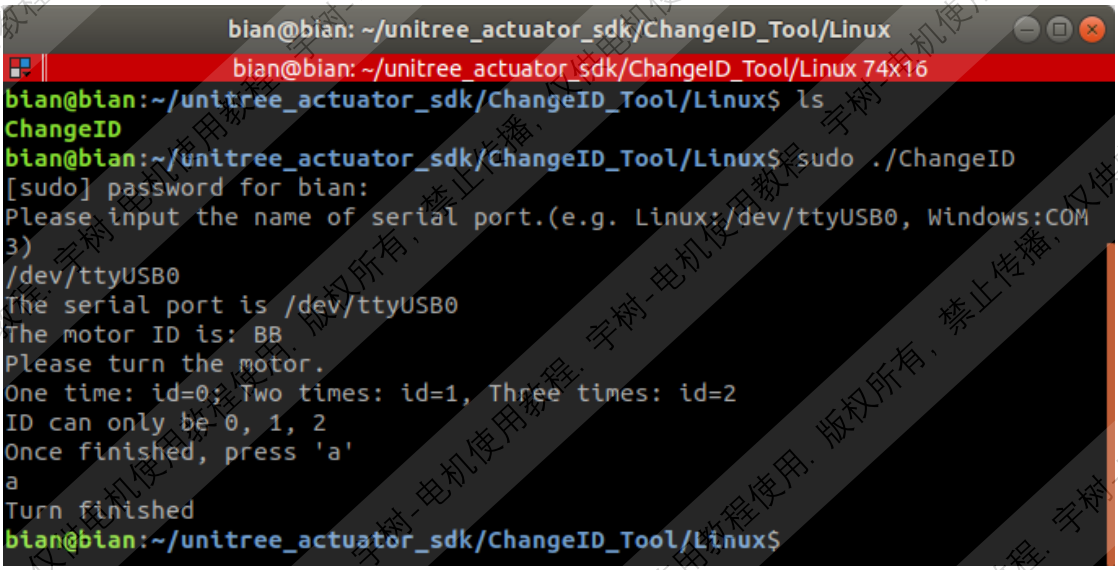

图 1.5: Linux 下执行修改 ID 程序

开始执行ChangeID程序后，第一步为输入当前的串口名。正如上文所述，在本实例中该串口名为 /dev/ttyUSB0，输入后按下回车，所有电机都会进入修改 ID 模式。

由于电机输出轴的电子棘轮刚度较大，所以建议操作者按照图 (1.6) 所示安装工装螺丝，并使用杠杆旋转电机输出轴。需要注意，工装螺丝为 M4 螺丝，且旋入深度不可超过 5mm。正如图 (1.5) 所示，转动输出轴一次，则电机的 ID 被设置为 0，以此类推，且 ID 只能为 0，1 和 2。在转动完每个电机的输出轴后，在终端窗口输入 a 并且按下回车，即完成电机 ID 的修改。

1.6 Python 范例试运行

让我们检验一下电机 ID 修改的结果，让电机转起来。因为 Python 是一种脚本语言，修改源代码后，不需要编译即可直接运行。所以为了操作简便，我们使用unitree_actuator_sdk 中的 Python 示例代码。

首先打开script文件夹下的check.py文件，第一步要做的就是修改串口名。如果只在一个系统下运行，那么只修改这一系统下的串口名即可：

```
fd = c.open_set(b'\\\\.\\COM4')  \\ Windows系统下
...
```

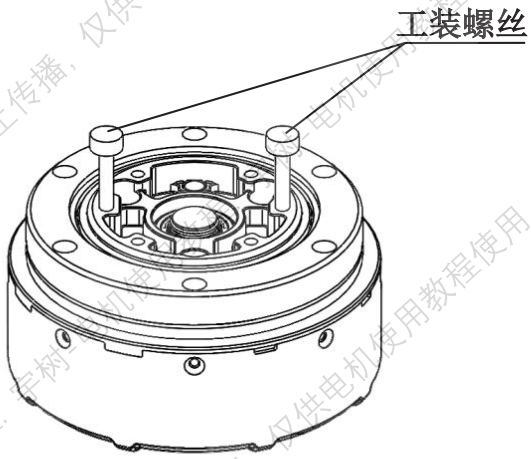

图 1.6: 电机输出轴安装工装螺丝

```
fd = c.open_set(b'/dev/ttyUSB0') \\ Linux系统下
```

接下来是修改给电机发送和接收的命令：

```
motor_s = MOTOR_send()
motor_s1 = MOTOR_send()
motor_r = MOTOR_recv()
```

其中motor_s与motor_s1为给电机发送的数据包，它们都是MOTOR_send类型的结构体。所谓结构体，即包含了许多不同类型数据的数据包，我们马上就会展示针对结构体的操作。同理motor_r是接收电机返回命令的数据包，它是一个MOTOR_recv类型的结构体。关于这两个结构体的具体内容，可以参考 script/typedef.py文件，在此不再赘述。

接下来我们修改motor_s与motor_s1。首先解释一下MOTOR_send 类型结构体包含的数据：

1. id: 当前控制命令的目标电机 ID
2. mode: 目标电机运行模式。0 为停转，5 为开环缓慢转动，10 为闭环伺服控制
3. T: 前馈力矩 τ_{ff}
4. W: 指定角速度 ω_{des}
5. Pos: 指定角度位置 p_{des}
6. K_P: 位置刚度 k_p
7. K_W: 速度刚度（阻尼）k_d

当mode的值为 0 或 5 时，后面的 5 个控制参数并没有任何作用。为了安全起见，我们的首次运转令mode = 5。当我们给电机发送这一命令后，电机会持续执行这一命令，即开环缓慢转动。为了让电机停下来，我们还需要创建一个令电机停止运转的命令：

```
motor_s.id = 0
motor_s.mode = 5
...
```

```
motor_s1.id = 0
motor_s1.mode = 0
```

在上述代码中，motor_s命令电机开环缓慢转动，motor_s1命令电机停止运转。在将motor_s与motor_s1发送给电机之前，需要先将它们的数据进行处理：

```
c.modify_data(byref(motor_s))
c.modify_data(byref(motor_s1))
```

下面我们就可以给电机发送命令了，下列代码会通过send_recv()函数每秒给电机发送一次motor_s命令，并且用motor_r接收电机当前状态，持续 5 秒：

```
i = 0
while(i < 5):
    c.send_recv(fd, byref(motor_s), byref(motor_r))
    c.extract_data(byref(motor_r))
    print('********************')
    print('Motor torque: ', motor_r.T)
    print('Motor position: ', motor_r.Pos)
    print('Motor velocity: ', motor_r.W)
    time.sleep(1)
    i = i + 1

c.send_recv(fd, byref(motor_s1), byref(motor_r))
```

实际上如果我们只是要让电机持续运转，并不需要持续发送同一条命令。之所以这么做，是为了持续获取电机当前的状态，如力矩、位置、速度等。因为电机只有在接收到命令后，才会返回自身的状态，所以需要持续发送命令。电机返回的状态是经过压缩编码的，所以需要通过函数extract_data() 将其解码。解码之后就可以通过 Python 的print()函数将各个状态打印出来。在完成循环之后，向电机发送motor_s1命令，停止电机运转。

图 (1.7) 即为check.py运行后的结果。因为对串口进行操作需要管理员权限，所以我们需要在script文件夹下用sudo python3 check.py命令来执行 check.py。

1.7 电机控制模式

在 (1.3) 节我们提到，可以修改电机的 5 个控制参数。这 5 个参数的不同搭配组合就能够形成不同的控制模式。下面我们继续使用示例代码check.py来进行讲解。

首先必须切换到mode = 10的伺服模式下：

```
motor_s.mode = 10
```

接下来我们将展示不同的控制模式的参数设置。

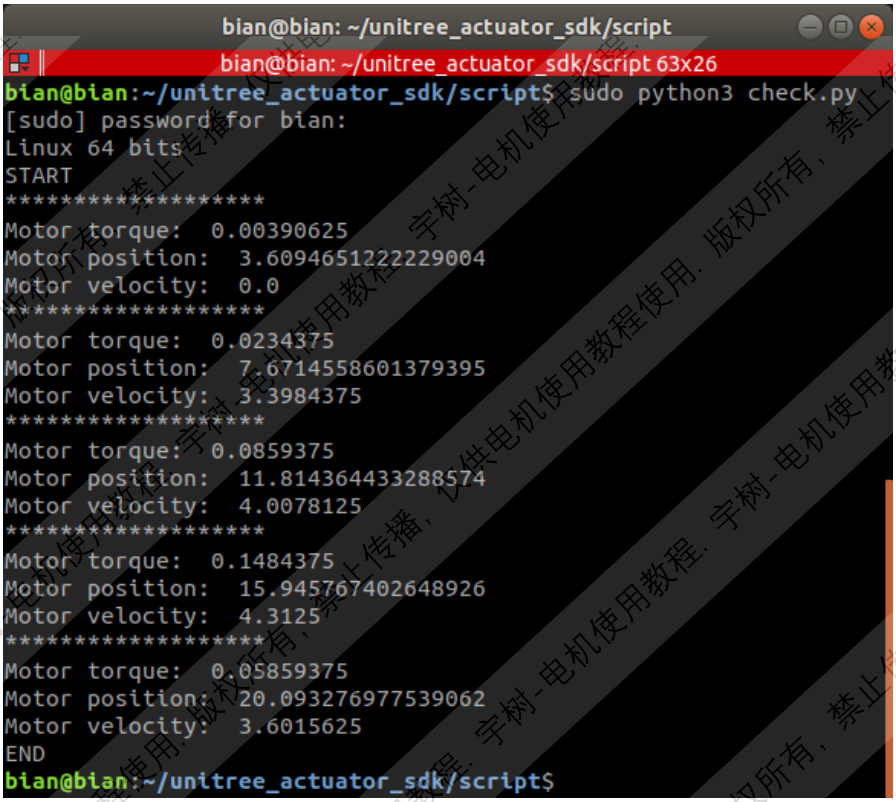

图 1.7: 运行 check.py

笔记 此处需要特别注意的是，给电机发送的命令都是针对减速器之前的电机转子，即图 (1.8) 中的转轴 1。而不是经过减速之后的输出轴 2。所以在进行实际控制的过程中，一定要注意考虑电机的减速比。在 A1 的电机中，减速比为 9.1。

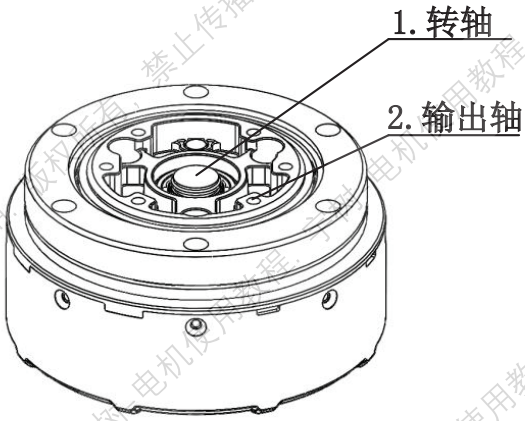

图 1.8: 电机输出端

1.7.1 位置模式

在位置模式下，电机的输出轴将会稳定在一个固定的位置。例如，如果我们希望电机输出端固定在 3.14 弧度的位置，可以将控制参数如下设置：

```
motor_s.T = 0.0          # 单位: Nm, |T|<128
motor_s.W = 0.0          # 单位: rad/s, |W|<256
motor_s.Pos = 3.14*9.1   # 单位: rad, |Pos|<823549
motor_s.K_P = 0.2        # 0<K_P<16
motor_s.K_W = 3.0        # 0<K_W<32
```

在上述参数设置中，将T与W设置为 0，即可成为针对Pos的 PD 控制 [1]。其中K_P为比例系数，K_W为微分系数，9.1 为减速比。各个参数的单位如注释（“#” 后的内容）所示，并且由于参数压缩算法的要求，各个参数的绝对值必须小于限制值。

完成上述修改后，运行check.py，即可从电机返回的状态看出，电机转子的位置稳定在 3.14×9.1 弧度，即电机输出端固定在 3.14 弧度的位置。根据式1.1可知，如果目标位置和当前位置之间差距很大，那么电机产生的力矩 τ 也会很大，从而产生一个很大的电流。如果给电机供电的电源的输出电流上限较小，可能会出现电源保护，即电机停止旋转。此时就需要考虑让motor_s.Pos 缓慢变化，避免产生瞬间的极大力矩。

1.7.2 速度模式

在速度模式下，电机的输出轴将会稳定在一个固定的速度。令电机输出轴转速稳定在 5rad/s：

```
motor_s.T = 0.0          # 单位: Nm, |T|<128
motor_s.W = 5.0*9.1      # 单位: rad/s, |W|<256
motor_s.Pos = 0.0        # 单位: rad, |Pos|<823549
motor_s.K_P = 0.0        # 0<K_P<16
motor_s.K_W = 3.0        # 0<K_W<32
```

速度模式下T和K_P必须为 0，这样就构成了对W的 P 控制 [2]。其中K_W为速度的比例系数。

1.7.3 阻尼模式

阻尼模式是一种特殊的速度模式。当我们令W = 0.0时，电机会保持转轴速度为 0。并且在被外力旋转时，产生一个阻抗力矩。这个力矩的方向与旋转方向相反，大小与旋转速度成正比。当停止外力旋转后，电机会静止在当前位置。因为这种状态和线性阻尼器类似，所以被成为阻尼模式。

1.7.4 力矩模式

在力矩模式下，电机会持续输出一个恒定力矩。但是当电机空转时，如果给一个较大的目标力矩，电机会持续加速，直到最大速度，这时也仍然达不到目标力矩。

[1]即比例-微分控制

[2]即比例控制

下面提供一个无负载情况下比较安全的力矩模式参数设置:

```
motor_s.T = 0.05        # 单位: Nm, |T|<128
motor_s.W = 0.0         # 单位: rad/s, |W|<256
motor_s.Pos = 0.0       # 单位: rad, |Pos|<823549
motor_s.K_P = 0.0       # 0<K_P<16
motor_s.K_W = 0.0       # 0<K_W<32
```

在该参数下，可以观察到电机在恒定力矩下逐渐加速的过程。因为各个电机之间存在细微差异，如果电机无法顺利旋转，可以适量增大T的数值。

1.7.5 零力矩模式

零力矩模式是一种特殊的力矩模式。当我们令`T = 0.0`时，电机会保持转轴的力矩为0。此时电机并不是停止运转，而是主动产生力矩来抵抗自身的摩擦力矩。因此在零力矩模式下，尝试转动输出轴，会感觉输出轴的阻力明显小于未开机时的阻力。

1.7.6 混合模式

在四足机器人的实际控制时，机器人运动控制器往往会给关节同时发送前馈力矩τ_{ff}、目标角度p_{des}和目标角速度ω_{des}。这时的控制模式就是混合控制，这也是我们在后面的实际应用中使用最多的一种控制模式。

1.8 * 移植到其他上位机平台

考虑到部分用户会使用特殊的上位机平台来控制电机，在此我们也对如何编写自己的电机控制程序进行讲解。根据本节所述的方法，读者可以在任意满足硬件要求的平台上给电机发送控制命令并接收电机状态。这部分内容只面向有需求的少数读者，大部分读者可以直接略过。

1.8.1 通信配置

A1 机器人的电机采用串口通信，通信标准为 RS-485，波特率为 4.8MBd。串口的数据位为 8 bit，无奇偶校验位，停止位为 1 bit。需要注意的是为了提高电机的通信频率，我们使用了 4.8MBd 这一很高的波特率，用户需要检查自己的硬件是否支持这么高的波特率。

根据 RS-485 标准，串口的通讯线需要有两根，分别为 A 线与 B 线，同时我们还增加了一根地线 GND。

1.8.2 电机收发报文格式

在控制电机时，我们会通过串口给电机发送一个长度为 34 字节的命令，之后电机会返回一个长度为 78 字节的状态。如果不给电机发送命令，那么电机也不会返回状态。给

电机发送命令的格式如表1.1所示，电机返回的状态格式如表1.2所示。这两个表中详细介绍了收发报文中各个字节表示的含义，下面我们会解释其中的部分细节。

首先是发送给电机命令中的第 13、14 字节，这两个字节表示了电机前馈力矩 τ_{ff}，显然前馈力矩 τ_{ff} 是一个浮点数，即`float`型，但是在大多数平台下，一个`float`型浮点数需要占用 4 个字节。为了使用 2 个字节来表示浮点数，我们使用的方法是移位操作，在此不对具体原理展开讲解。从应用的角度，读者可以认为我们对前馈力矩乘了 256，之后赋值给一个 2 个字节的`signed short int`型，即带符号的短整形变量。在这个赋值过程中会强制取整，这样我们就可以只用两个字节来发送前馈力矩 τ_{ff}，当电机接收到这个数据后，只需要除以 256，就能获得前馈力矩 τ_{ff} 的数值。这种操作虽然会丧失一些精度，但是对于实际应用是完全足够的。另外需要注意的一点就是，对于一个长度为 2 字节的变量，一共有 16 个字符，即 16 位。其中 1 位用来表示正负，是符号位，所以只有 15 位用于表示数值的大小，这意味着命令中`T`的数值不能大于 2^{15}。考虑到我们曾经对原始数据 τ_{ff} 乘了 256，所以其绝对值存在上限：

$$|\tau_{ff}| < \frac{2^{15}}{256} = 128 \tag{1.2}$$

同时需要注意的是，由于在赋值过程中存在强制取整，所以 τ_{ff} 的数值越大，保存的小数精度越低。表1.1中变量`T` 的说明里所说的 ×256 倍描述指的就是上文中所说的乘 256，其他变量中所说的某某倍描述也与之同理。

在发送命令和接收状态的末尾，我们可以看到一个 4 字节的 CRC 校验，如果在数据传输过程中出现了数据错误，那么 CRC 校验就无法通过，从而帮助我们避免错误数据。在此我们不对 CRC 校验的算法展开具体介绍，读者可以直接参考如下代码：

```
uint32_t crc32_core(uint32_t* ptr, uint32_t len){
    uint32_t xbit = 0;
    uint32_t data = 0;
    uint32_t CRC32 = 0xFFFFFFFF;
    const uint32_t dwPolynomial = 0x04c11db7;
    for (uint32_t i = 0; i < len; i++){
        xbit = 1 << 31;
        data = ptr[i];
        for (uint32_t bits = 0; bits < 32; bits++){
            if (CRC32 & 0x80000000){
                CRC32 <<= 1;
                CRC32 ^= dwPolynomial;
            }
            else
                CRC32 <<= 1;
            if (data & xbit)
                CRC32 ^= dwPolynomial;
            xbit >>= 1;
        }
```

```
    }
  return CRC32;
}
```

可见crc32_core函数的第一个参数是uint32_t型的指针 ptr，uint32_t型即是 4 字节长度的无符号整形数，ptr即表示需要进行 CRC 校验的数据指针。而另一个参数len则表示需要进行 CRC 校验的数据长度，由于我们发送的命令去掉最后的 CRC 校验位之后还有 30 个字节，也就是包含 7 个完整的uint32_t型，所以在计算发送命令的 CRC 时需要令len=7。

	字节	变量名	说明
数据包头 COMHead	1	start[0]	包头，固定为 0xFE
	2	start[1]	包头，固定为 0xEE
	3	motorID	电机编号，可以为 0、1、2、0xBB，0xBB 代表向所有电机广播
	4	reserved	预留位，可忽略
数据体 Master ComdV3	5	mode	电机运行模式，可为 0（停转）、5（开环缓慢转动）、10（闭环伺服控制）
	6	ModifyBit	电机内部控制参数修改位，可忽略
	7	ReadBit	电机内部控制参数发送位，可忽略
	8	reserved	预留位，可忽略
	9	Modify	电机参数修改数据，可忽略
	10		
	11		
	12		
	13	T	电机前馈力矩 τ_{ff}，×256 倍描述
	14		
	15	W	电机速度命令 ω_{des}，×128 倍描述
	16		
	17	Pos	电机位置命令 p_{des}，$\times\frac{16384}{2\pi}$ 倍描述
	18		
	19		
	20		
	21	K_P	电机位置刚度 k_p，×2048 倍描述
	22		
	23	K_W	电机速度刚度 k_d，×1024 倍描述
	24		
	25	LowHzMotor CmdIndex	电机低频率控制命令的索引，可忽略

	26	LowHzMotor CmdByte	电机低频率控制命令，可忽略
	27	Res[0]	预留位，可忽略
	28		
	29		
	30		
CRC 校验	31	CRCdata	CRC 校验位
	32		
	33		
	34		

表 1.1: 发送给电机命令的报文格式

	字节	变量名	说明
数据包头 COMHead	1	start[0]	包头，固定为 0xFE
	2	start[1]	包头，固定为 0xEE
	3	motorID	电机编号，可以为 0、1、2。不会为 0xBB，因为广播模式下电机不会返回状态
	4	reserved	预留位，可忽略
数据体 Servo ComdV3	5	mode	电机当前的运行模式
	6	ReadBit	表示电机内部控制参数是否修改成功，可忽略
	7	Temp	电机当前平均温度
	8	MError	电机报错信息
	9	Read	读取电机内部控制参数，可忽略
	10		
	11		
	12		
	13	T	当前电机输出力矩，×256 倍描述
	14		
	15	W	当前电机实际转速，×128 倍描述
	16		
	17	LW	也表示当前电机实际转速，但是已经过滤波，所以会有延迟，不推荐使用
	18		
	19		

20		
21	W2	为关节编码器预留，可忽略
22		
23	LW2	为关节编码器预留，可忽略
24		
25		
26		
27	Acc	当前电机转动加速度，×1 倍描述
28		
29	OutAcc	为关节编码器预留，可忽略
30		
31	Pos	当前电机角度位置，$\times\frac{16384}{2\pi}$ 倍描述
32		
33		
34		
35	Pos2	为关节编码器预留，可忽略
36		
37		
38		
39	gyro[0]	电机控制板上 IMU 在 x 轴的角速度，乘 $\frac{2000}{2^{15}}\cdot\frac{2\pi}{360}$ 后可以换算为 rad/s
40		
41	gyro[1]	电机控制板上 IMU 在 y 轴的角速度
42		
43	gyro[2]	电机控制板上 IMU 在 z 轴的角速度
44		
45	acc[0]	电机控制板上 IMU 在 x 轴线加速度，乘 $8\cdot\frac{9.80665}{2^{15}}$ 后可以换算为 $\mathrm{m/s^2}$
46		
47	acc[1]	电机控制板上 IMU 在 y 轴的线加速度
48		
49	acc[2]	电机控制板上 IMU 在 z 轴的线加速度
50		
51	Fgyro[0]	
52		
53	Fgyro[1]	
54		
55	Fgyro[2]	
56		
57	Facc[0]	为足端 IMU 预留，可忽略

	58		
	59	Facc[1]	
	60		
	61	Facc[2]	
	62		
	63	Fmag[0]	
	64		
	65	Fmag[1]	
	66		
	67	Fmag[2]	
	68		
	69	Ftemp	足端传感器温度，可忽略
	70	Force16	足端力传感器高 16 位数据，可忽略
	71		
	72	Force8	足端力传感器低 8 位数据，可忽略
	73	FError	足端力传感器错误标识，可忽略
	74	Res[0]	预留位，可忽略
CRC 校验	75	CRCdata	CRC 校验位
	76		
	77		
	78		

表 1.2: 从电机接收状态的报文格式