unitree_legged_sdk LowLevel Interfaces

unitree_legged_sdk LowLevel Interfaces

LowLevel Control是四足机器人的底层控制模式，可以理解为控制对象是狗的12个关节电机，通过sdk将电机指令发送给12个电机。

1. LowCmd

1.1 LowCmd结构体中主要的控制命令

MotorCmd **motorCmd**[20]	LowCmd中主要控制命令就是20个MotorCmd结构体数组，其中前12个有效，个电机的编号可以参考quadruped.h MotorCmd结构体参考1.2 MotorCmd

1.2 MotorCmd

uint8_t **mode**	期望的电机工作模式，默认初始化为伺服模式
float **q** (unit: rad)	期望的角度
float **dq** (unit: rad/s)	期望的角速度
float **tau** (unit: N·m)	期望的输出力矩
float **Kp** (unit: N·m/rad)	位置刚度
float **Kd** (unit: N·m/(rad/s))	速度刚度

1539 字

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。对于机器人，我们通常需要给关节设定位置、速度和力矩，就需要对关节电机进行混合控制。

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 + Kp * (q - q') + Kd * (dq - dq')

其中，tau' 为电机输出力矩，q' 为电机当前角度，dq' 为电机当前角速度。

SDK中已经做了电机减速比的转换，客户不需要关心电机减速问题。

2. LowState

2.1 LowState结构体中有效的状态反馈

MotorState **motorState**[20]	电机状态反馈，20个MotorState结构体数组，其中前12个有效，各个电机的编 可以参考quadruped.h MotorState结构体参考2.2 MotorState
IMU **imu**	参考2.3 LowState IMU
int16_t **footForce**[4]	足端传感器数值，触地检测。 这个值是飘的，每个气囊的值不一样，需要实际测试，通常是通过变化量来检 是否触地。
int16_t **footForceEst**[4]	电机电流估计的足端力，单位应该是N
uint32_t **tick** (unit: ms)	主控板的实时参考，数值没有实际意义，通过取差值来获得时间。
uint8_t **wirelessRemote**[40]	遥控器键值的反馈，可参考提供的手柄例程源码

2.2 MotorState

uint8_t **mode**	电机当前的工作模式
float **q** (unit: rad)	电机当前的角度

1539 字

float **dq** (unit: rad/s)	电机当前的角速度
float **ddq** (unit: rad/s²)	电机当前的角加速度
float **tauEst** (unit: N·m)	电机当前估计的输出力矩
float **q_raw** (unit: rad)	电机当前的角度的原始数值
float **dq_raw** (unit: rad/s)	电机当前的角速度的原始数值
float **ddq_raw** (unit: rad/s²)	电机当前的角加速度的原始数值
int8_t **temperature**	电机温度，但是由于温度传导缓慢导致滞后

2.3 LowState IMU

float **quaternion**[4]	归一化的四元数 quaternion[0] = w quaternion[1] = x quaternion[2] = y quaternion[3] = z
float **gyroscope**[3] (unit: rad/s)	陀螺仪，角速度，原始数据 gyroscope[0] = x gyroscope[1] = y gyroscope[2] = z

1539 字

float **accelerometer**[3] (unit: m/s²)	加速度计，加速度，原始数据 accelerometer[0] = x accelerometer[1] = y accelerometer[2] = z
float **rpy**[3] (unit: rad)	欧拉角 rpy[0] = Roll rpy[1] = Pitch rpy[2] = Yaw
int8_t **temperature**	IMU温度
在加速运动时，由 IMU 计算出的机器人姿态会发生漂移。	

3. LowLevel Interfaces中的safety.h

safety.h中定义了宇树SDK unitree_legged_sdk底层模式里的保护函数，主要包含功率保护和位置保护。保护是通过软件的方式，减少异常力矩导致设备故障的概率，建议新手都加上，后续熟练了以后，可以不使用保护函数。
safety.h里的函数只是一种软件的保护手段，并不能保证都能有效保护，我们的底层控制需要注意不要输出异常大力矩。

3.1 PowerProtect(LowCmd&, LowState&, int)

仅在底层控制中生效；
输入参数取值1-10，表示10%~100%功率限制；
如果您是新手，请使用1；如果您熟悉之后，那么可以尝试更大的取值甚至注释掉此功能。
PowerProtect主要是功率保护，利用的原理是p=fv，力矩*角速度，也就是12个电机力矩乘各自角速度的总和，按经验值均匀划分成了1-10档，使用时可以慢慢加。（经验值，总功率1000w，5档就是500w）
功率保护，手去摇晃腿也能被动触发保护。
触发后函数返回-1，函数不会对命令做修改。

3.2 PositionProtect(LowCmd&, LowState&, double limit = 0.087)

位置保护，表示关节电机离两侧限位的值，默认限制5°。

3.3 PositionLimit(LowCmd&)

仅在底层控制中的位置模式生效，把位置指令限制在最大限位里。

4. example

4.1 example_position.cpp

位置模式底层示例。

motiontime < 400时，初始化一条腿的位姿，用到了线性插值，防止初始位置与目标位置差距太大导致异常力矩。

motiontime > 400时，通过两个正弦函数叠加算出小腿关节的位置，使小腿关节来回摆动。

4.2 example_torque.cpp

力矩模式底层示例。

```
53      if( motiontime >= 500){
54          float torque = (0 - state.motorState[FR_1].q)*10.0f + (0 - state.motorState[FR_1].dq)*1.0f;
55          if(torque > 5.0f) torque = 5.0f;
56          if(torque < -5.0f) torque = -5.0f;
57          // if(torque > 15.0f) torque = 15.0f;
58          // if(torque < -15.0f) torque = -15.0f;
59
60          cmd.motorCmd[FR_1].q = PosStopF;
61          cmd.motorCmd[FR_1].dq = VelStopF;
62          cmd.motorCmd[FR_1].Kp = 0;
63          cmd.motorCmd[FR_1].Kd = 0;
64          cmd.motorCmd[FR_1].tau = torque;
65      }
66      int res = safe.PowerProtect(cmd, state, 1);
67      if(res < 0) exit(-1);
```

首先用一个PD公式，计算让这个关节保持0位时需要的力矩（随后通过限幅保证安全），然后通过力矩模式发给改关节。机器狗的表现为该关节保持零位不动。

为了保证安全，发送之前过了一下功率保护。

这里的PosStopF和VelStopF是力矩控制时，推荐设置的位置和速度值。

4.3 example_velocity.cpp

速度模式底层示例。

```
53      if( motiontime >= 500){
54          float speed = 2 * sin(3*M_PI*Tpi/1500.0);
55
56          cmd.motorCmd[FR_1].q = PosStopF;
57          cmd.motorCmd[FR_1].dq = speed;
58          cmd.motorCmd[FR_1].Kp = 0;
59          cmd.motorCmd[FR_1].Kd = 4;
60          // cmd.motorCmd[FR_1].Kd = 6;
61          cmd.motorCmd[FR_1].tau = 0.0f;
62          Tpi++;
63      }
64
65      if(motiontime > 10){
66          int res1 = safe.PowerProtect(cmd, state, 1);
67          // You can uncomment it for position protection
68          // int res2 = safe.PositionProtect(cmd, state, 10);
69          if(res1 < 0) exit(-1);
70      }
71
72      udp.SetSend(cmd);
```

首先是通过一个正弦函数算出变化的速度值，然后通过速度模式发送给该关节。机器狗的表现为该关节来回移动。

为了保证安全，发送之前过了一下功率保护。

1539 字