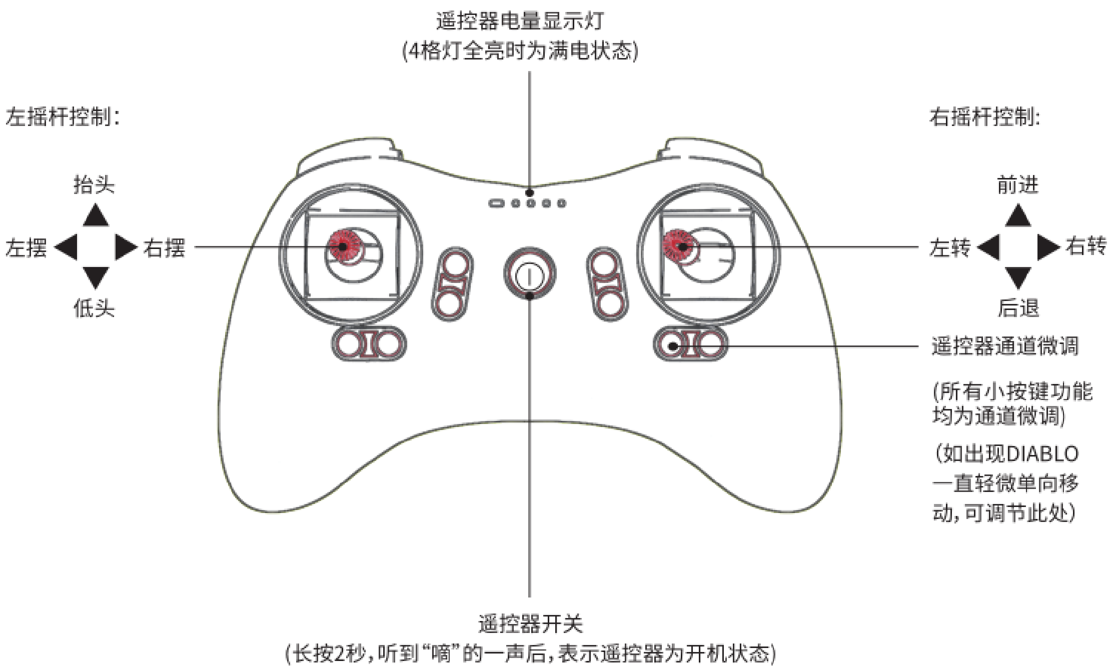
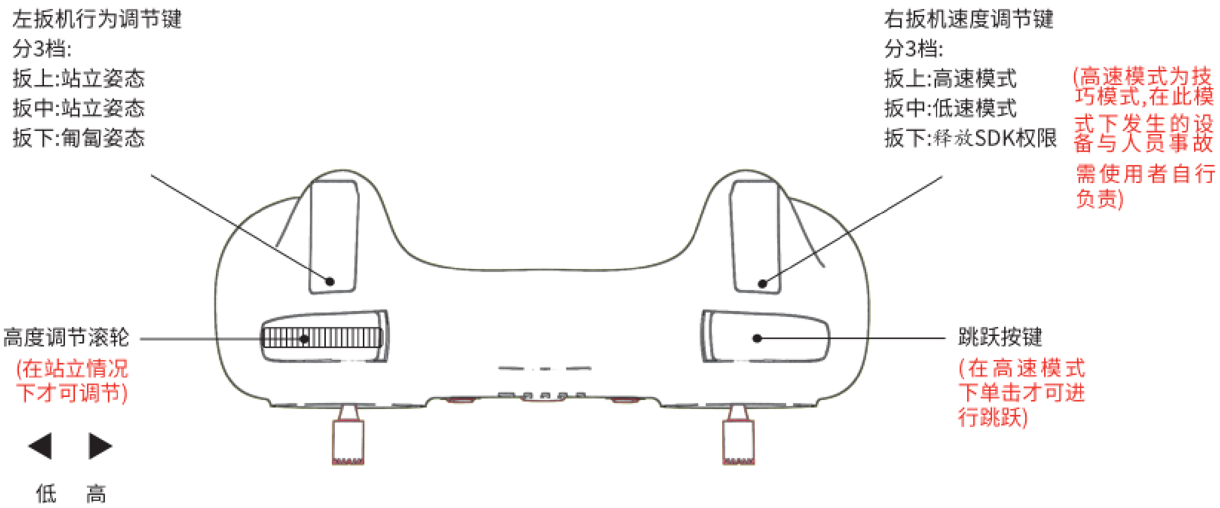
diabloA1.1参考文本

1.快速使用指南

1.1 机器人遥控器使用

可参考说明书中的遥控器指南，具体如下所示，也可参考 视频 。注意视频中为A1.0版本的操作，与A1.1的不同在于急停方式。 在A1.1中，需要先按下跳跃键，再把两个扳手扳下，才可让电机失能。





注:为确保用户的人身安全,左扳机行为调节键默认设置在中间位置,如需使DIABLO站立,用户需先将左扳机行为调节键往下扳,再往中间扳回,DIABLO即可站立。

扳机按键位置切换

单键状态简介

左扳机		右扳机	
扳下	闸刹状态	扳下	遥控器禁用，给SDK释放权限
扳中	站立状态	扳中	低速模式
扳上	站立状态	扳上	高速模式

1.1组合使用简介

左扳机	右扳机	跳跃键	状态	备注
扳下	扳下	禁用	闸刹状态，遥控器禁用	给SDK释放权限
扳下	扳下	点击2秒	电机处于失能状态	该状态可使机器人电机失能，在站立状态下请谨慎使用
扳下	扳中	禁用	闸刹状态，低速运动	左摇杆功能禁用，右摇杆正常使用
扳下	扳上	禁用	闸刹状态，高速运动	左摇杆功能禁用，右摇杆正常使用
扳中	扳下	禁用	站立状态，遥控器禁用	给SDK释放权限
扳中	扳中	禁用	站立状态，低速运动	左摇杆和右摇杆正常使用
扳中	扳上	机器跳跃	站立状态，高速运动	左摇杆和右摇杆正常使用
扳上	扳下	禁用	站立状态，遥控器禁用	给SDK释放权限
扳上	扳中	机械舞	站立状态，低速运动	左摇杆功能禁用，右摇杆正常使用
扳上	扳上	机械舞	站立状态，高速运动	左摇杆功能禁用，右摇杆正常使用

1.2 机器人调速

1.2.1 通过修改最大限速进行调速

打开以下文件：

复制代码

```
diablo-chibios_rt/param/param_M15.hpp
```

在变量 `param_core` 中修改成员 `.FORWARD_MAX_VEL` 。该成员为机器人速度，单位为m/s 。目前达到过的理论最高时速是2.2m/s，有摔跤的风险，请谨慎提速。

1.2.2 通过修改speed_psc进行调速

打开以下文件

复制代码

```
Core\Function\Task.cpp
```

修改以下函数中的变量 `speed_psc`

复制代码

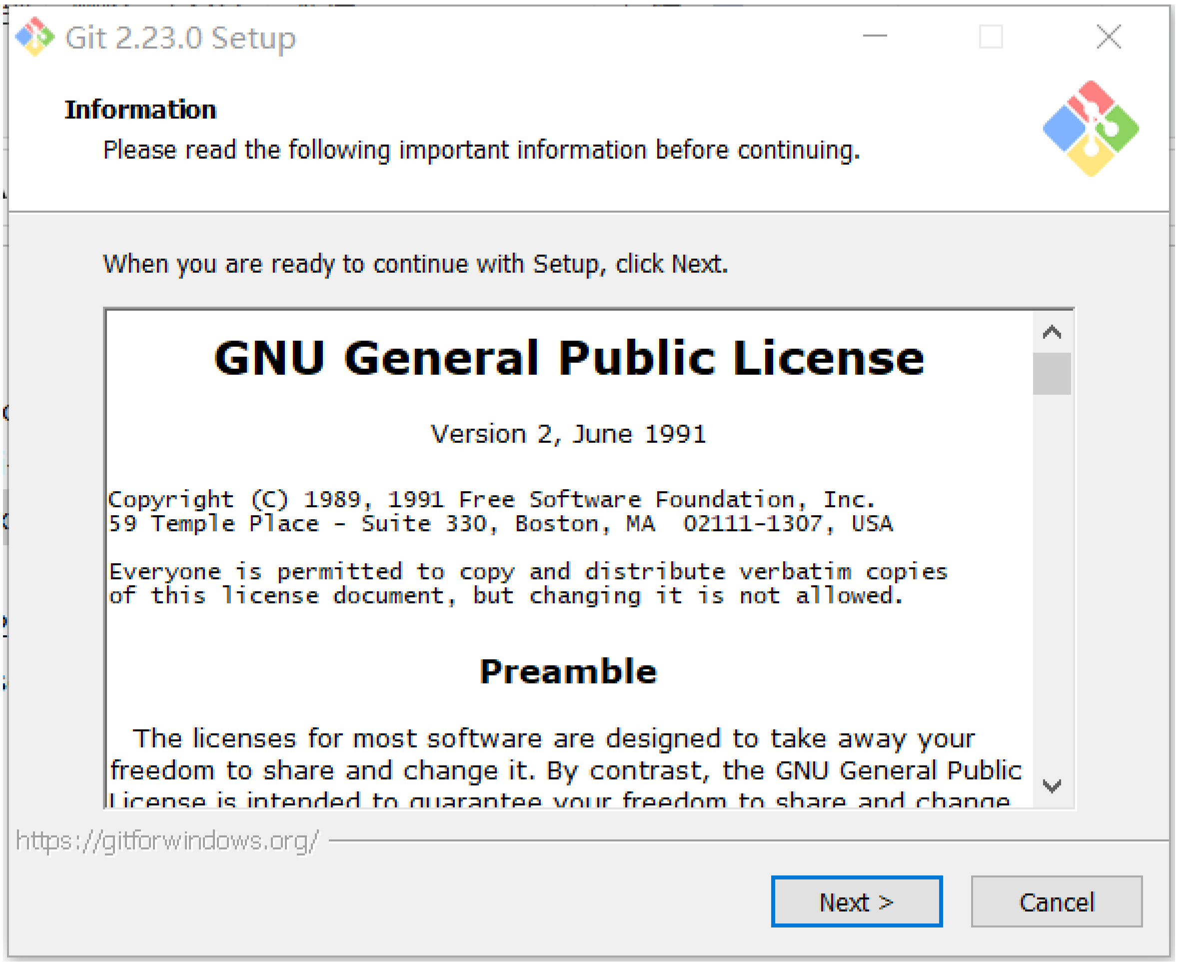
```
void WB6_Task::cmd_update(float dt) //指令更新，接收来自遥控器RC或SDK的指令更新
float speed_psc = rc->speed_mode ? 1.f : 0.5f;  //speed_psc速度分频，用于区分机器人高低速挡
```

1.3 给机器人debug

1.3.1 机器人debug环境配置

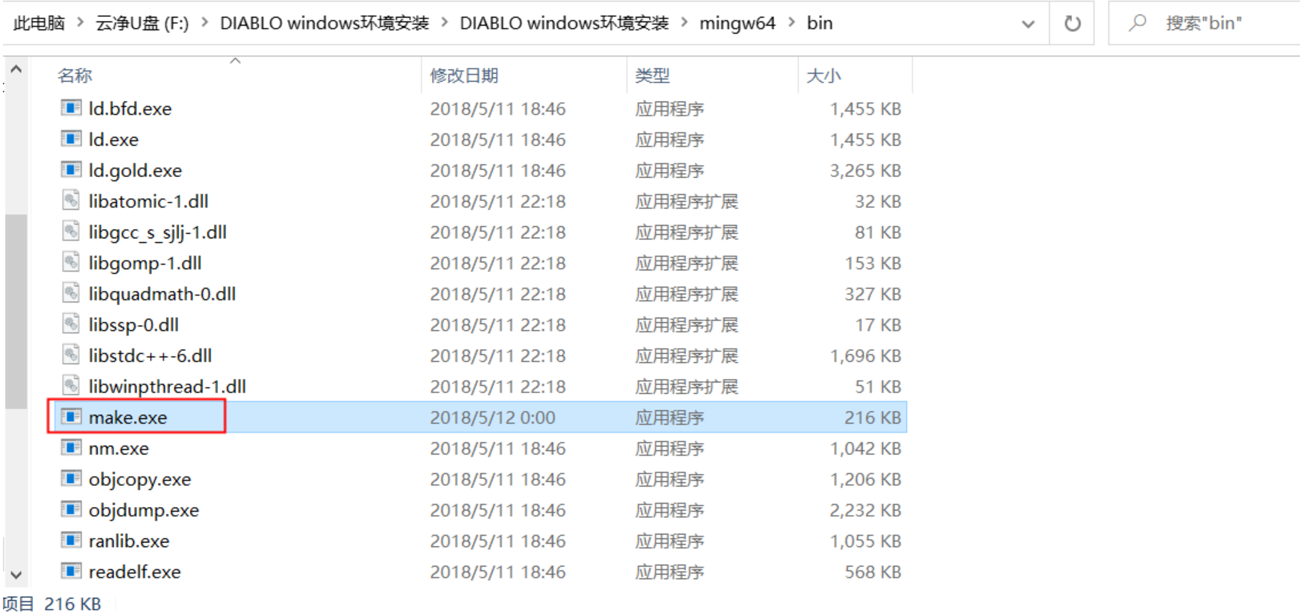
按照以下步骤进行：

1. 安装git 和 sourcetree

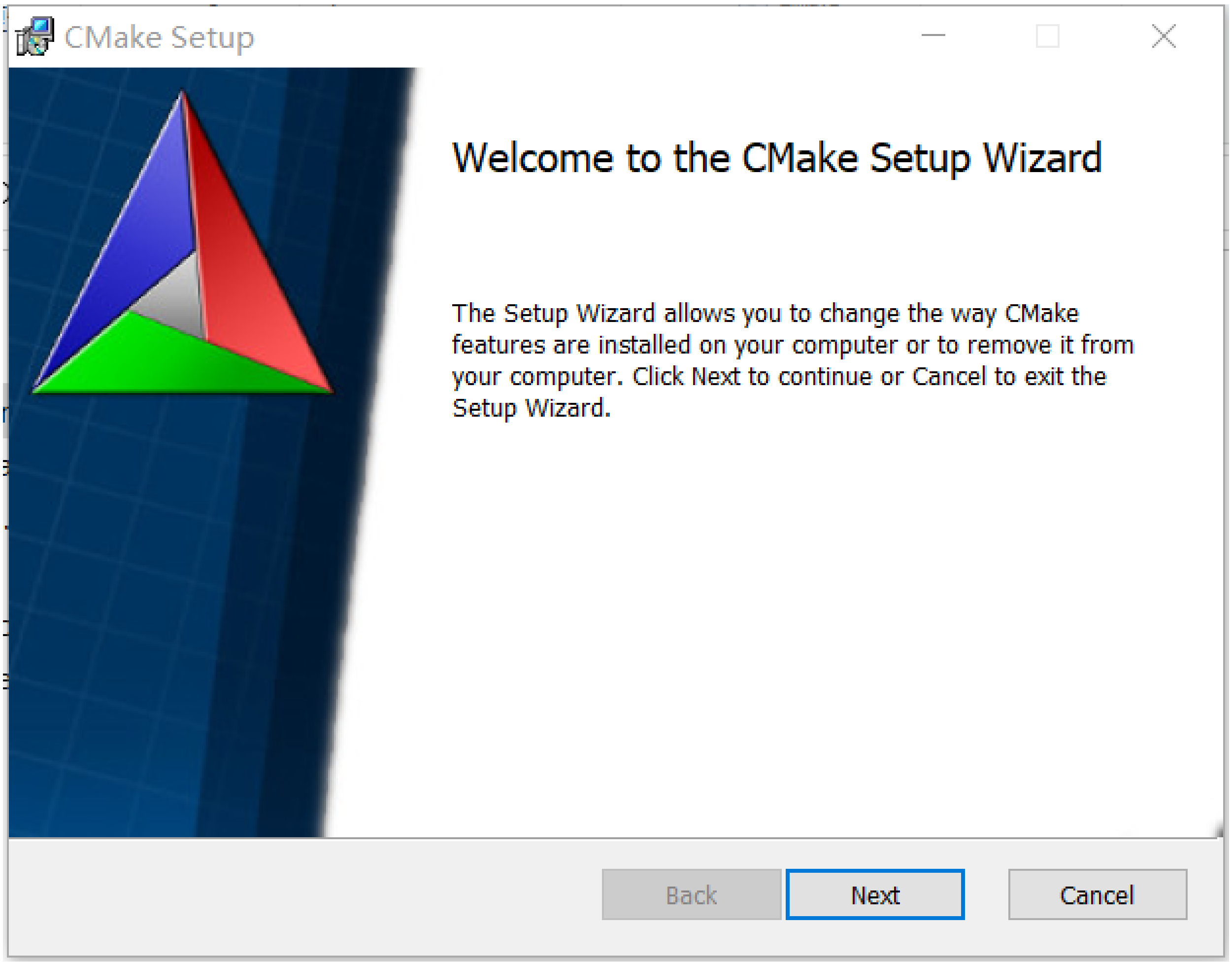


2. 安装gcc-arm-none-eabi ， 将bin文件并添加到PATH环境变量中（一定要清楚您安装软件的位置）

3. 解压缩minGW64 ，将mingw32-make.exe 改为make.exe，将bin文件并添加到PATH环境变量中

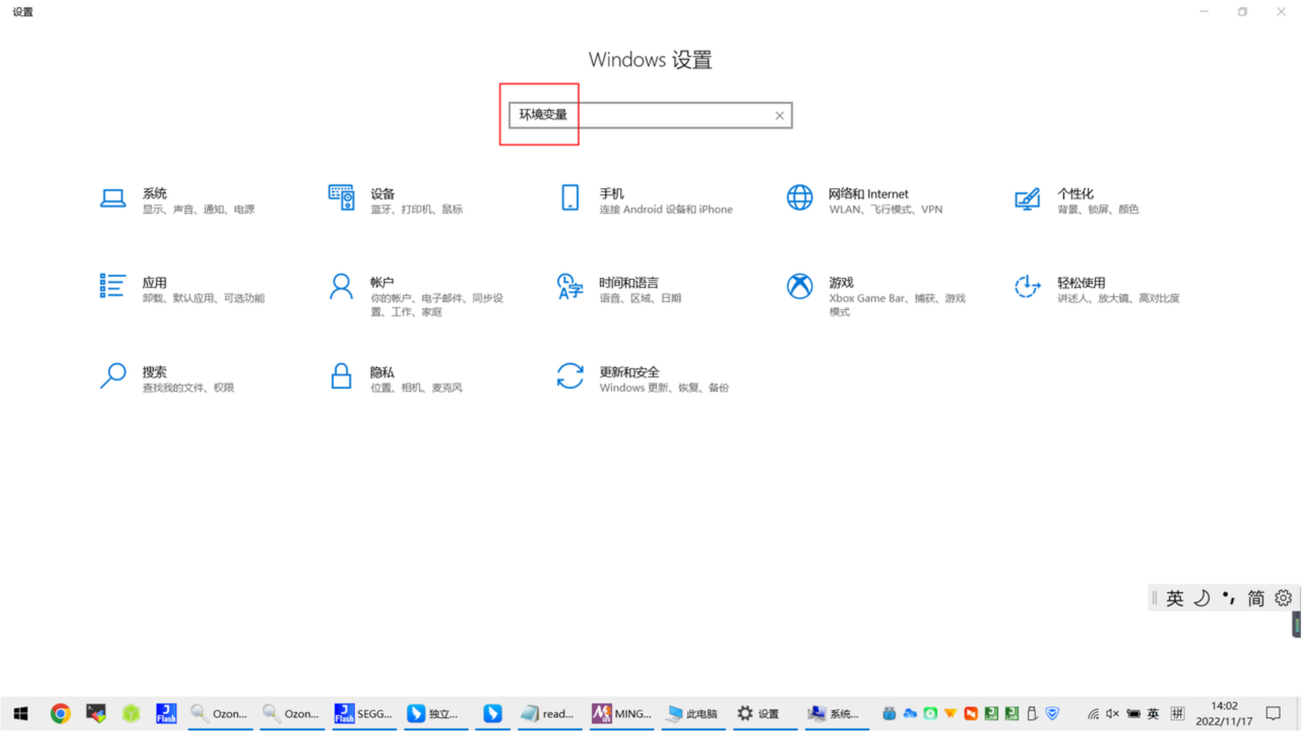


4. 安装Cmake，将bin文件添加到PATH环境变量中

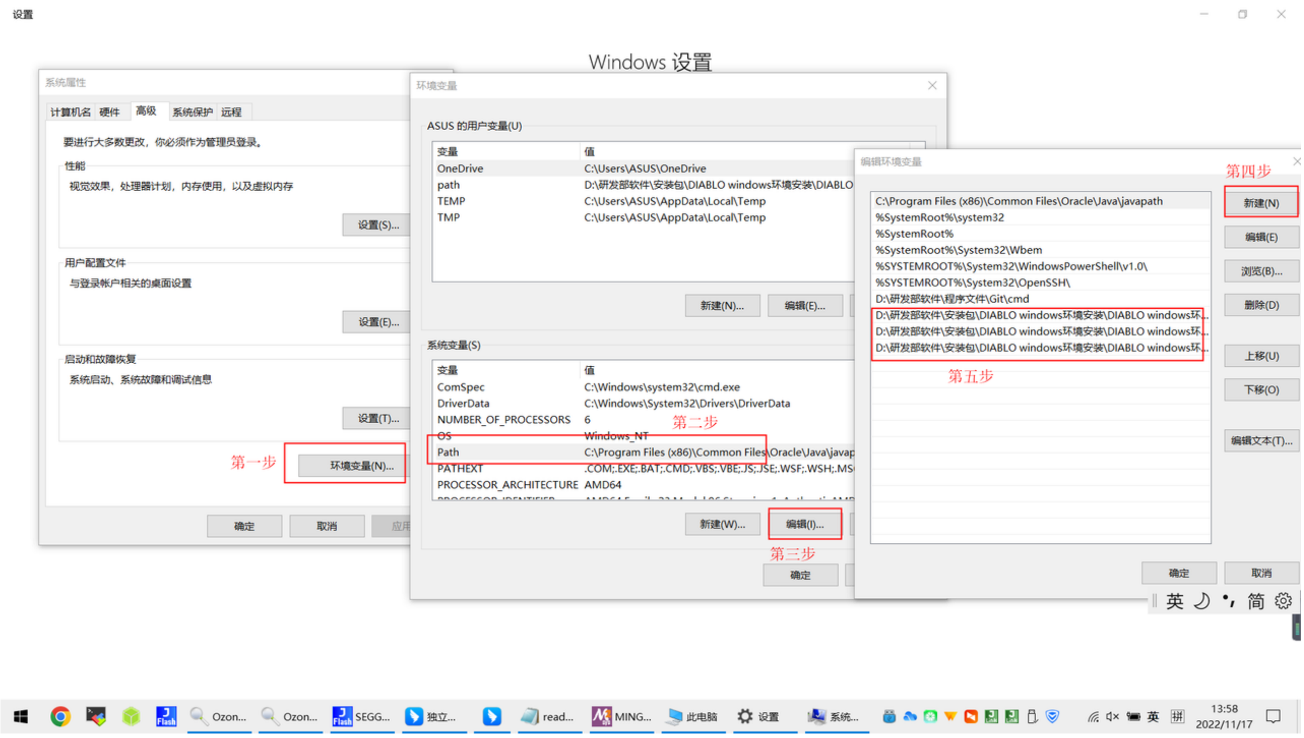


将bin文件添加到PATH环境变量中的操作流程如下：

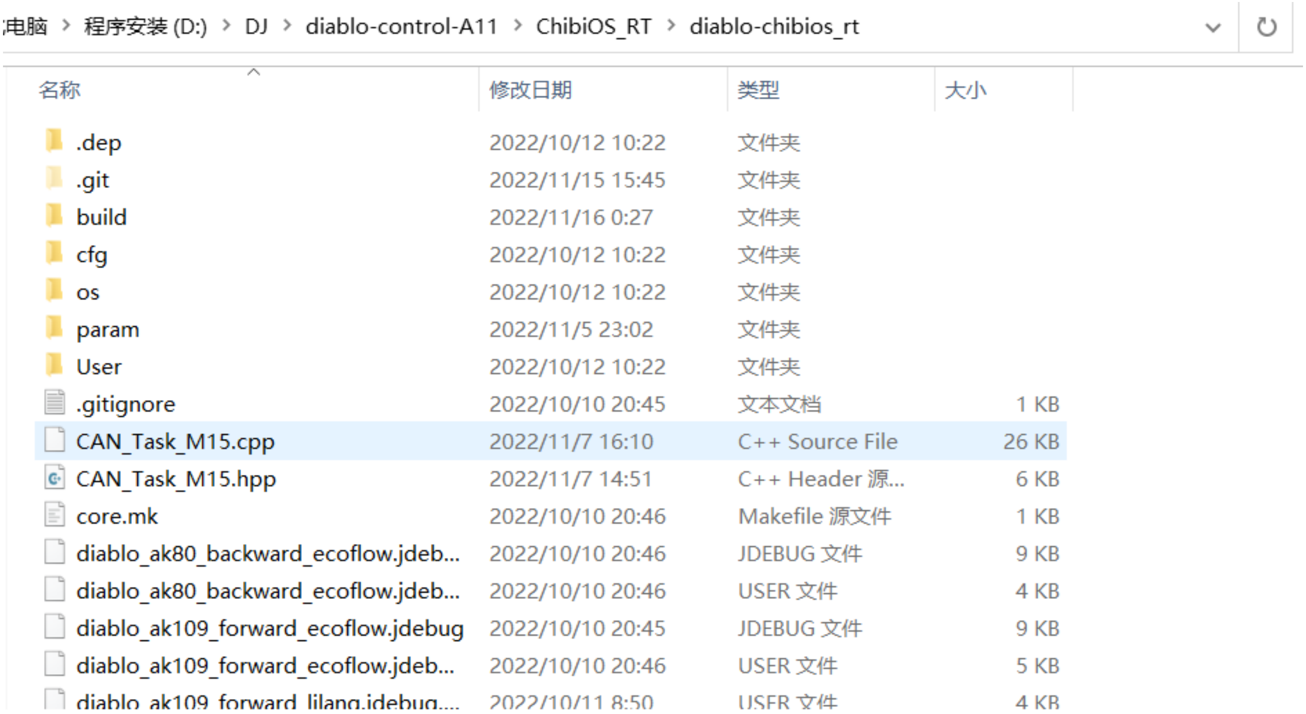
1、打开windows设置（控制面板），搜索栏输入环境变量，找到系统环境变量；



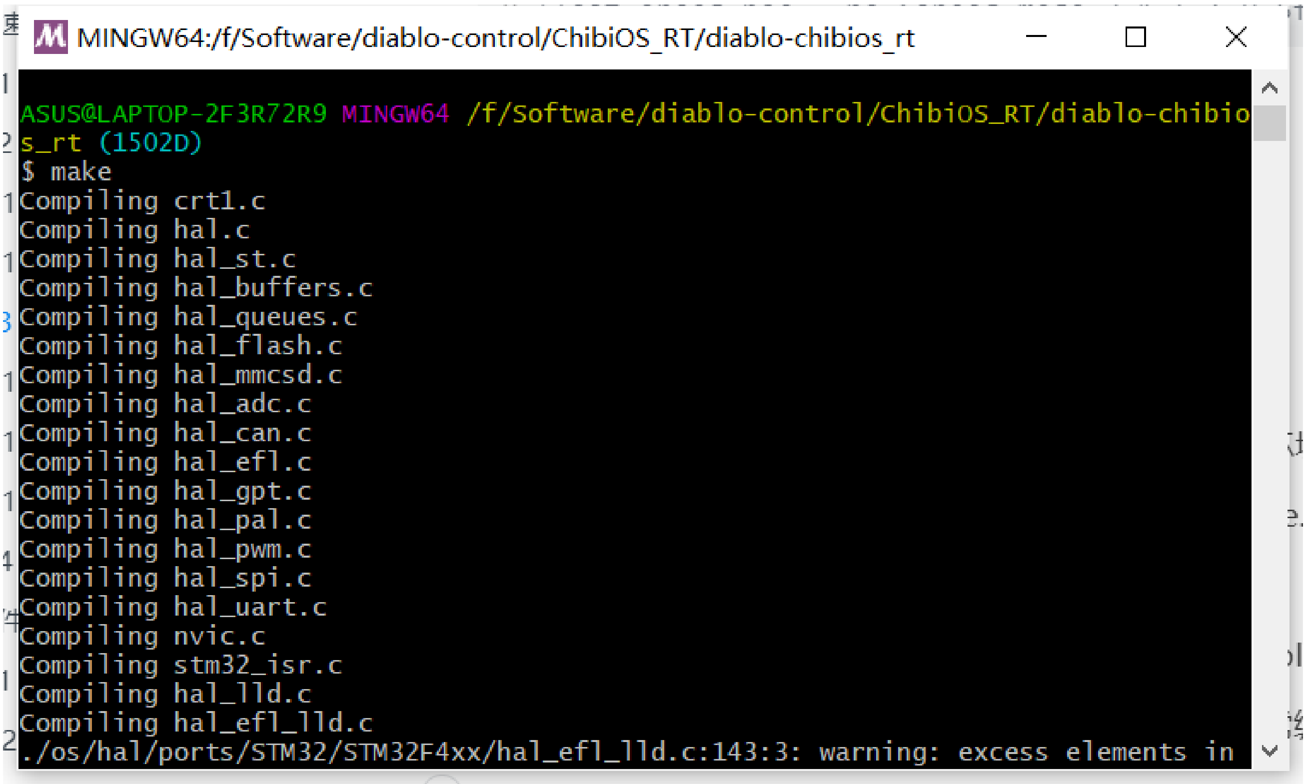
2、把对应的bin文件添加到环境变量中，添加完成后，请务必点击确认及应用。



完成以上步骤之后，打开代码所在的文件夹 diablo-control\ChibiOS_RT\diablo-chibios_rt 位置

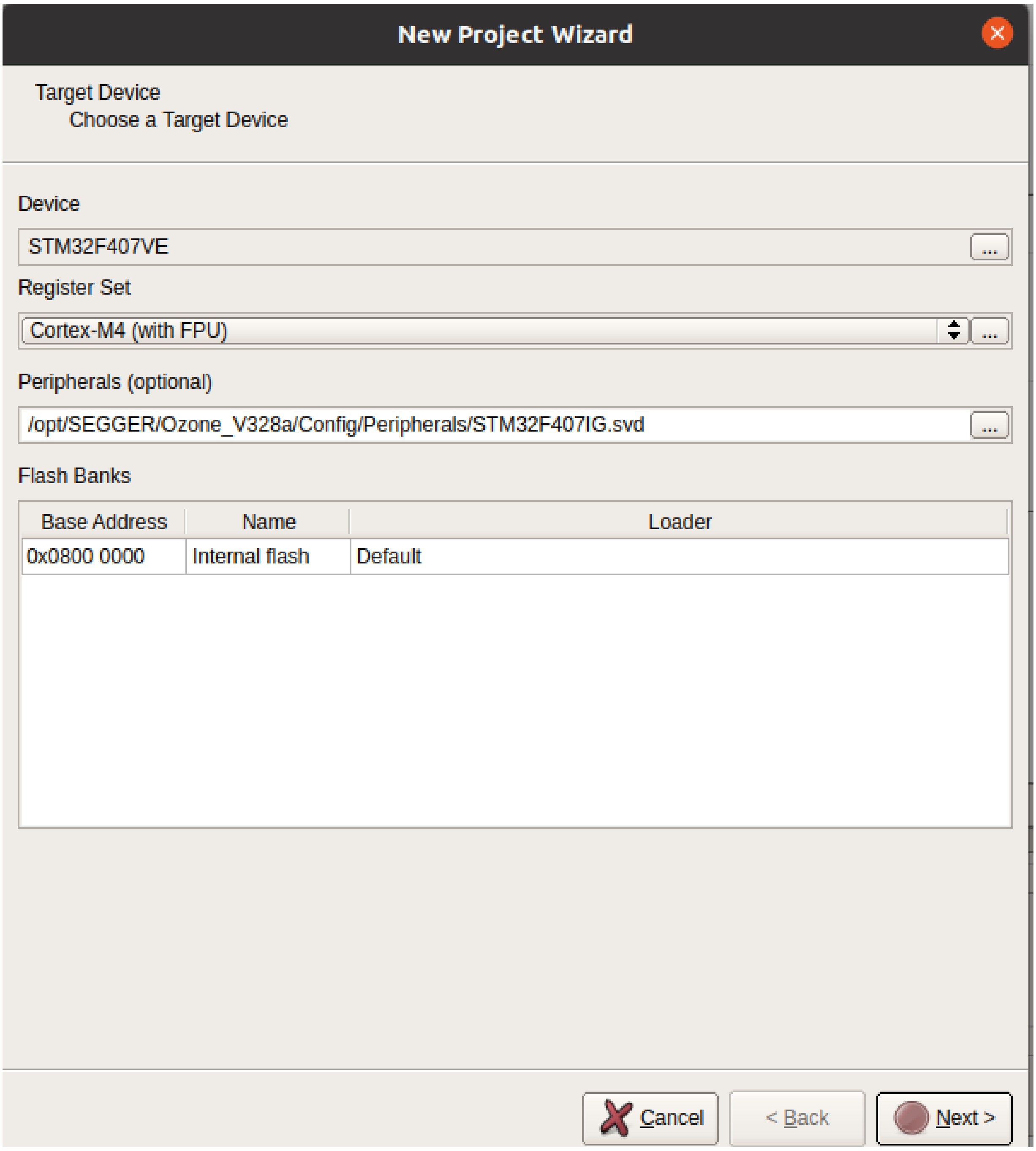


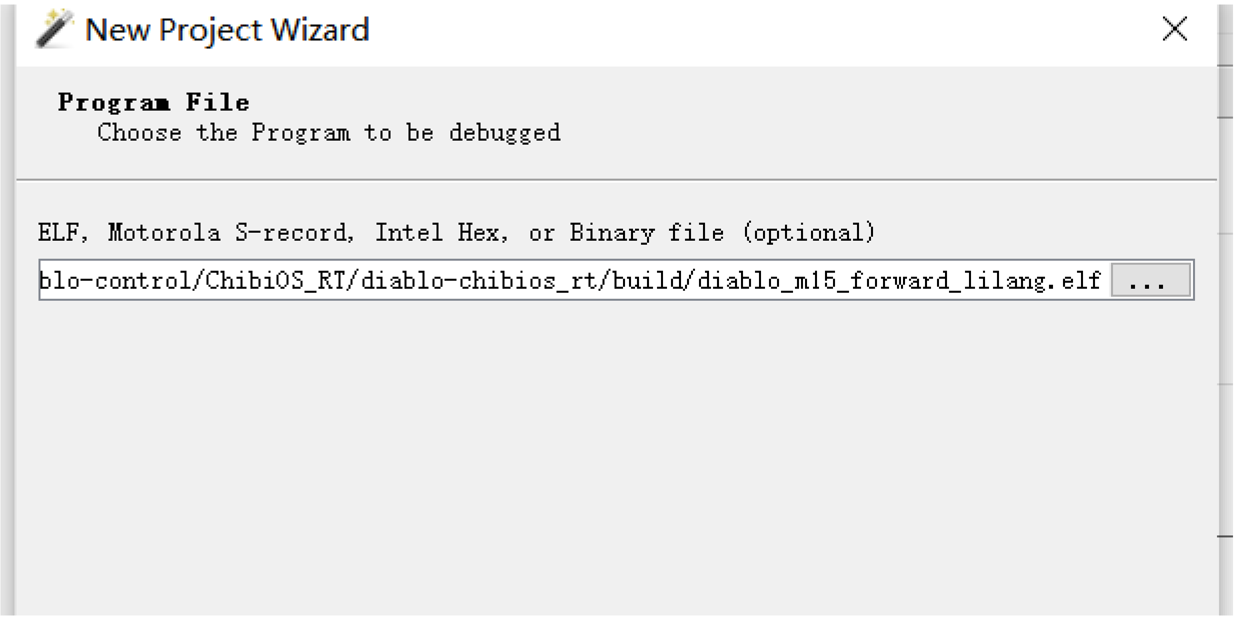
鼠标右键打开Git Bash Here，进行make编译，代码能正常编译既环境安装完成。



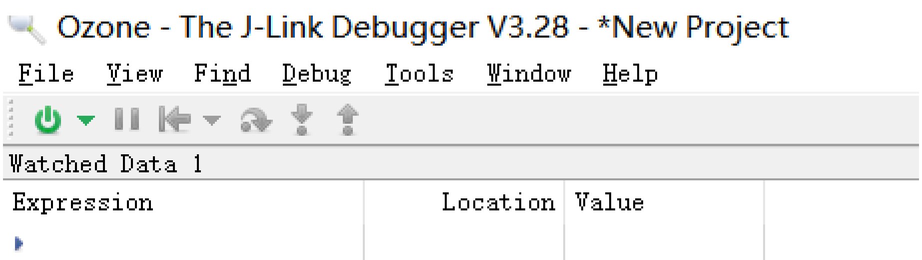
1.3.2 给机器人debug

通过makefile编译代码后，在ozone导入 `diablo_m15_forward_lilang.elf` 文件如下图所示。

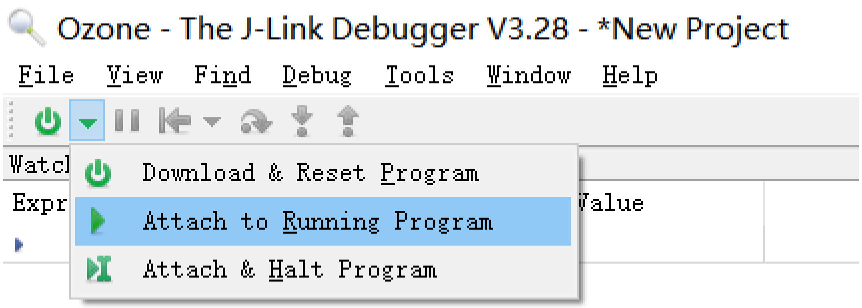

New Project Wizard
Target Device
Choose a Target Device
Device
STM32F407VE
Register Set
Cortex-M4 (with FPU)
Peripherals (optional)
/opt/SEGGER/Ozone_V328a/Config/Peripherals/STM32F407IG.svd
Flash Banks
Base Address
Name
Loader
0x0800 0000
Internal flash
Default
Cancel
< Back
Next >



点击左上角绿色开机键下载和烧录代码



如不想停止代码运行，可以选择attach 代码，如下图所示。



可在 `Watched Data` 中输入 `p_diablo` ，查看机器人中robot实例中的各项参数。具体参数可参考**章节3**

注意在使用ozone进行debug过程中，只有被实例化的对象是可以查看数值的。

1.3.3 修改并保存机器人的y轴偏置

如果发现机器人站立时，机器人有向前或向后偏移的情况出现，可修改 `p_diablo.param.imu_bias.y` 。如果站起后机器人往前飘，可增大数值；若机器人往后退，可减小数值。可以0.01为刻度进行调整。

参数调整完毕后。可在diablo_main.cpp的main函数下，修改 `ctrl_flag = 1` ，即可将参数写入MCU Flash中。

1.4 SDK使用

参考SDK [使用文档]。

SDK的协议的固件端在以下文件中查看：

```
diablo-chibios_rt\User\src\Onboard_SDK_Telemetry.cpp，*.hpp // SDK操作文件，包括SDK线程调度
diablo-control\Core\Interface\Onboard_SDK_UART_Protocol.h//  SDK协议文档
```

2.文件模块说明

2.1 User模块

2.1.1 ChibiOS使用说明

2.1.2 diablo_main.cpp

命令线程，在进入规划之前，先完成电机端的初始化，也就是Diablo_Ctrl.status == 8时，开始执行控制更新时，才能进入规划，否则无法进行正常的控制。

```
void Cmd_Task::main(void)
{
    task->plan_start();

    systime_t time = System::getTimeX();
    float dt;
    while(!this->shouldTerminate())
    {
```

```
        time += TIME_MS2I(1000/freq);
        if(System::getTimeX() < time)// cmd task time period
        {
            this->sleepUntil(time);
            dt = 1.f/freq;
        }
        else
        {
            systime_t curr = System::getTimeX();
            dt = TIME_I2US(curr - time)/1e6f;
            time = curr;
        }

        battry_level_display.update();
        if(Diablo_Ctrl.status == 8){
        task->cmd_update(dt);
        task->plan_update(dt);
        }
        //calibration finish, save parameter
        if(task->calibration.finish_flag)
        {
            p_param->save();
            task->calibration.finish_flag = false;
        }
    }
}
```

控制线程，配置电机

复制代码

```
void Ctrl_Task::main(void)
{

    static float init_stamp = 0;
    static uint8_t init_cnt = 0;
    task->ctrl_start();
    status = 0;
    systime_t time = System::getTimeX();
    float dt;
    while(!this->shouldTerminate())
    {
        time += TIME_MS2I(1000/freq);
```

```
        if(System::getTimeX() < time)
        {
            this->sleepUntil(time);
            dt = 1.f/freq;
        }
        else
        {
            systime_t curr = System::getTimeX();
            dt = TIME_I2US(curr - time)/1e6f;
            time = curr;
        }
        stamp += dt;
        init_stamp += dt;
        //whether init finish
        switch(status)
        {
            case 100://重置电流环参数用CASE
                if(init_stamp < 3.f) // 10ms
                {
                    for(uint8_t motor_id = 1;motor_id <= 6;motor_id++ )
                    {

//CAN.Set_Control_Param(CAN.CAN_CURRENT_LOOP,100,5,5,9,motor_id);
                        CAN.Set_Control_Reset(motor_id);
                    }
                    for(uint8_t motor_id = 1;motor_id <= 6;motor_id++ )
                    {
                        CAN.Set_Current_Filter(118,motor_id);
                    }
                }
                else if(init_stamp >= 3.f && init_stamp <= 3.2f)
                {
                    // CAN.setMechOffset(CAN.CAN_MECH_OFFSET_LEG_RIGHT);
                    // CAN.setMechOffset(CAN.CAN_MECH_OFFSET_LEG_LEFT);
                    for(uint8_t motor_id = 1;motor_id <= 6;motor_id++ )
                    {

CAN.Set_Control_Param(CAN.CAN_CURRENT_LOOP,100,5,5,9,motor_id);
                        // CAN.Set_Control_Reset(motor_id);
                    }
                    for(uint8_t motor_id = 1;motor_id <= 6;motor_id++ )
                    {
                        // CAN.Set_Control_Param_Save(motor_id);
                    }
                }
```

```
                else
                {

                    CAN.setMode();
                    CAN.sendCmd();
                }
                break;

            case 0: // set protection
                if(init_stamp > 0.1f) // 10ms
                {
                 for(uint8_t motor_id = 1;motor_id <= 6 ;motor_id++ )
                    {
                     CAN.set_temp_Mode_init(motor_id,0);
                    }
                    init_stamp = 0;
                    status = 1;
                }
                break;
            case 1: // set motor baud rate

                if(init_stamp >= 0.1f) // 10ms
                {

                     CAN.setMode();
                     init_cnt++;
                    init_stamp = 0;
                }


                if(CAN.motors[2]->mode == CAN.motors[2]->CTRL_CURRENT)
                {
                    for(uint8_t motor_id = 1;motor_id <= 6;motor_id++ )
                    {
                        // CAN.set_temp_Mode_init(motor_id);

//CAN.Set_Control_Param(CAN.CAN_CURRENT_LOOP,100,5,5,9,motor_id);
                        CAN.Set_Control_Reset(motor_id);
                    }
                    init_stamp = 0;
                    init_cnt = 0;
                    status = 2;
                }
                break;
            case 2: // set protection
```

```
            if(init_stamp > 0.1f) // 10ms
            {
                for(uint8_t motor_id = 1;motor_id <= 6;motor_id++ )
                {
                    CAN.Set_Protect_Switch(CAN.CAN_PROTECT_OFF,motor_id);
                }
                init_stamp = 0;
                status = 3;
            }
            break;
        case 3: // set filter
            if(init_stamp > 0.1f) // 10ms
            {
                for(uint8_t motor_id = 1;motor_id <= 6;motor_id++ )
                {
                    CAN.Set_Current_Filter(118,motor_id);
                }
                init_stamp = 0;
                status = 4 ;
            }
            break;
        case 4: // set control parameter
            if(init_stamp > 0.1f) // 10ms
            {
                for(uint8_t motor_id = 1;motor_id <= 6;motor_id++ )
                {

CAN.Set_Control_Param(CAN.CAN_CURRENT_LOOP,100,5,5,9,motor_id);
                }
                init_stamp = 0;
                status = 5;
            }
            break;
        case 5: // set last parameter feedback
            if(init_stamp > 0.1f) // 10ms
            {
                for(uint8_t motor_id = 1;motor_id <= 6;motor_id++ )
                {
                    CAN.Set_Stall(20000,10,10,motor_id);
                }
                init_stamp = 0;
                status = 6;
            }
            break;
        case 6:
```

```
                if(init_stamp > 0.1f) // 10ms
                {
                    for(uint8_t motor_id = 1;motor_id <= 6 ;motor_id++ )
                    {
                        CAN.Set_Time_Out(100,motor_id);
                    }
                    init_stamp = 0;
                    status = 7;
                }
                break;
            case 7:
                if(init_stamp > 0.1f) // 10ms
                {
                    for(uint8_t motor_id = 1;motor_id <= 6 ;motor_id++ )
                    {
                        CAN.set_temp_Mode_init(motor_id,0x01);
                    }
                    init_stamp = 0;
                    status = 8;
                }
                break;
            case 8:
                if(user_kill)
                   task->ctrl.kill();
                else
                     task->ctrl_update(stamp);


                CAN.sendCmd();
                break;


        }



    }
}
```

```
void Ctrl_Task::start(WB6_Task* task, const bool kill)
```

status	备注

case 0	设置电机最后一位为电机模式
case 1	设置电机模式，重置电机PID参数
case 2	设定过温过流保护开关发送指令
case 3	设置电流滤波
case 4	修改电机PI参数
case 5	设置阻塞时间
case 6	设置超时时间
case 7	将电机最后一位设置为温度反馈
case 8	控制更新

1.看门狗线程，监控机器人状态，错误码提示，严重警告提示，警告提示，优先级126；
2.姿态线程，获取BMI088角速度，角加速度数据，优先级128；
3.电池线程初始化，ADC采集电池电压，串口6输出电池电压数据，优先级128；
4.遥控器sbus线程，获取遥控器通道数据，优先级129；
5.can1，can2通信初始化，机器人腿部左侧和右侧电机分别挂在在can1和can2上，对应id分别是97，98，99，左腿和右腿id一致，优先级135，can线程monitor管理线程优先级131；
6.串口3，串口4线程，用于SDK通信，发送线程优先级126，接受线程优先级129；
7.SDK线程管理，monitor1优先级126,monitor2优先级126；
8.Telemetry.start(&robot, &CAN, &UART, &SDKCTRL, &SBUS, &boardLed);
9.命令线程，优先级127；
10.控制线程，优先级126；

2.1.3 Attitude_Task.cpp

1.获取imu角速度， 更新机器人角速度在初始化时采集yaw、pitch、roll的角速度做平均值为角速度偏置， `atti->update_gyro(bmi088.gyroData, gyro_stamp, dt);`
2.获取角速度 atti->update_accl(bmi088.accelData, accl_stamp, dt);
3.获取imu温度 bmi088_get_temperature(&bmi088);

2.1.4 DDJ_M15.cpp

1.can数据更新，can接收一帧有8个byte,data8[0]为电机角度高8位，data8[1]为电机角度低8位，data8[2]为电机电流高8位，data8[3]为电机电流低8位，data8[4]为电机速度高8位，data8[5]为电机速度低8位，data8[6]为电机错误码，data8[7]为电机模式或者电机温度，在初始化出设置。 DDJ_M15::update_can(const CANRxFrame* frame, const float V_in)
2.获取电机模式 DDJ_M15::translateMode(const uint8_t rx_temp)
3.获取电机错误码 DDJ_M15::translateErrorCode(uint16_t error_code) 4.获取电机温度 void DDJ_M15::Translate_Temp(const uint8_t rx_temp)

2.1.5 Control

controller.cpp

设置各项pid参数以及限幅

```
bool WB6_Controller::setup(void)
{
    legL.pid_angle.kp = legR.pid_angle.kp = param.core.LEG_TILT_KP;
    legL.pid_angle.ki = legR.pid_angle.ki = param.core.LEG_TILT_KI;
    legL.pid_angle.kd = legR.pid_angle.kd = param.core.LEG_TILT_KD;
    legL.pid_angle.max_int = legR.pid_angle.max_int =
param.core.LEG_TILT_INT_MAX;
    legL.pid_angle.max_out = legR.pid_angle.max_out =
param.core.LEG_TILT_OUT_MAX;

    drive_pid.kp = param.core.DRIVE_KP;
    drive_pid.ki = 0;
    drive_pid.kd = param.core.DRIVE_KD;
    drive_pid.max_int = 0;
    drive_pid.max_out = robot.legL.wheel->max_output_torque;

    drive_lock_pid.kp = 5.f;
    drive_lock_pid.ki = 100.f;
    drive_lock_pid.kd = 0.f;
    drive_lock_pid.max_int = 6.f;
    drive_lock_pid.max_out = robot.legL.wheel->max_output_torque;

    legL.pid_len.kp = legR.pid_len.kp = param.core.LEG_LEN_KP;
    legL.pid_len.ki = legR.pid_len.ki = param.core.LEG_LEN_KI;
    legL.pid_len.kd = legR.pid_len.kd = param.core.LEG_LEN_KD;
    legL.pid_len.max_int = legR.pid_len.max_int = robot.legL.knee-
>max_output_torque / 2;
    legL.pid_len.max_out = legR.pid_len.max_out = robot.legL.knee-
>max_output_torque;

    leg_diff_pid.kp = param.core.LEG_LEN_DIFF_KP;
    leg_diff_pid.kd = param.core.LEG_LEN_DIFF_KD;

    roll_pid.kp = param.core.ROLL_KP;
    roll_pid.ki = param.core.ROLL_KI;
    roll_pid.kd = param.core.ROLL_KD;

    return true;
}
```

切换机器人模式函数

robot.mode参考值	含义	备注

CAR	CAR模式是机器人趴下姿态	关闭腿长robot.leg_on = false；
TRANSFORM_UP	TRANSFORM_UP机器人站起来过程状态	重置lqr头部和腿部的LQR参数，重置roll轴积分项输出，使能腿长；
TRANSFORM_DOWN	TRANSFORM_DOWN是机器人蹲下过程状态	重置左右腿tilt积分项输出，也就是腿部内侧的积分项输出；
STAND	机器人站立状态	drive_pid是轮子速度pid, drive_lock_pid是摔倒时用到的锁住轮子的pid,pid_len是控制腿长的pid,切换到站立姿态时将这4项pid积分项输出重置为0.

复制代码

```
void WB6_Controller::transform(const WB6_Base::robot_mode_t mode)
{
    if (mode == robot.mode)
        return;

    switch (mode)
    {
    case WB6_Base::CAR:
        robot.leg_on = false;
        break;
    case WB6_Base::TRANSFORM_UP:
        lqr2.reset();
        force.reset();
        roll_pid.reset();
        robot.leg_on = true;
        break;
    case WB6_Base::TRANSFORM_DOWN:
        legL.pid_angle.reset();
        legR.pid_angle.reset();
    case WB6_Base::STAND:
        drive_lock_pid.reset();
```

```
        drive_pid.reset();
        legL.pid_len.reset();
        legR.pid_len.reset();
        break;
    }
    robot.mode = mode;
}
```

腿长更新函数，限制关节力矩输出，左右腿长度更新

复制代码

```
void WB6_Controller::leg_len_update(const float left_setLen, const float
right_setLen, const float dt)
{
    this->knee_limit_update();
    legL.ctrl_update_drive(left_setLen,  dt);
    legR.ctrl_update_drive(right_setLen, dt);
}
```

Leg.cpp

腿部有关角度和长度解算

复制代码

```
void Leg2::update(const float stamp, const float pitch, const float d_pitch)
{
    static float angle_obs = 0;

    this->alpha = -(hip->angle - init_angle_hip) + pitch;
    this->beta = knee->angle - init_angle_knee - this->alpha;

    this->d_alpha  = -hip->angle_vel + d_pitch;
    this->d_beta   =  knee->angle_vel - this->d_alpha;
    this->dd_alpha = this->diff1.update(this->d_alpha, stamp);
    this->dd_beta  = this->diff2.update(this->d_beta, stamp);
    angle_obs   = this->knee_angle ;
    switch (config)
    {
        case LEG_CONFIG_FORWARD:
            this->wheel_pos  = wheel->round_angle + alpha;
            this->wheel_vel  = wheel->angle_vel + d_alpha;
            break;
        case LEG_CONFIG_BACKWARD:
```

```
            this->wheel_pos  = wheel->round_angle - beta;
            this->wheel_vel  = wheel->angle_vel - d_beta;
            break;
    }

    this->knee_angle = (knee->angle - init_angle_knee)/2;
    this->sin_knee = sinf(knee_angle);//knee_angle is the knee motor's angle,
each leg has one.
    float sinPsc = sin_knee;
        if(sinPsc < 0.2f) sinPsc = 0.2f;// why is this?
    this->sin_knee_psc = sinPsc;

    this->angle   = -(hip->angle - init_angle_hip) - knee_angle;
    this->d_angle = -hip->angle_vel - knee->angle_vel/2;
    this->len     =  cosf(knee_angle) * param->core.LEG_LEN_STRAIGHT;
    this->d_len   = -sinf(knee_angle) * param->core.LEG_LEN_STRAIGHT * knee-
>angle_vel/2;

    this->dd_len = param->core.LEG_LEN_STRAIGHT*
        -(sinf(knee_angle)*(dd_alpha + dd_beta)/2 + cosf(knee_angle)*(knee-
>angle_vel)*(knee->angle_vel)/4);
}void Leg2::update(const float stamp, const float pitch, const float d_pitch)
{
    static float angle_obs = 0;

    this->alpha = -(hip->angle - init_angle_hip) + pitch;
    this->beta = knee->angle - init_angle_knee - this->alpha;

    this->d_alpha  = -hip->angle_vel + d_pitch;
    this->d_beta   =  knee->angle_vel - this->d_alpha;
    this->dd_alpha = this->diff1.update(this->d_alpha, stamp);
    this->dd_beta  = this->diff2.update(this->d_beta, stamp);
    angle_obs  = this->knee_angle ;
    switch (config)
    {
        case LEG_CONFIG_FORWARD:
            this->wheel_pos  = wheel->round_angle + alpha;
            this->wheel_vel  = wheel->angle_vel + d_alpha;
            break;
        case LEG_CONFIG_BACKWARD:
            this->wheel_pos  = wheel->round_angle - beta;
            this->wheel_vel  = wheel->angle_vel - d_beta;
            break;
    }
```

```
    this->knee_angle = (knee->angle - init_angle_knee)/2;
    this->sin_knee = sinf(knee_angle);//knee_angle is the knee motor's angle,
each leg has one.
    float sinPsc = sin_knee;
        if(sinPsc < 0.2f) sinPsc = 0.2f;// why is this?
    this->sin_knee_psc = sinPsc;

    this->angle   = -(hip->angle - init_angle_hip) - knee_angle;
    this->d_angle = -hip->angle_vel - knee->angle_vel/2;
    this->len     =  cosf(knee_angle) * param->core.LEG_LEN_STRAIGHT;
    this->d_len   = -sinf(knee_angle) * param->core.LEG_LEN_STRAIGHT * knee-
>angle_vel/2;

    this->dd_len = param->core.LEG_LEN_STRAIGHT*
        -(sinf(knee_angle)*(dd_alpha + dd_beta)/2 + cosf(knee_angle)*(knee-
>angle_vel)*(knee->angle_vel)/4);
}
```

机器出厂腿部校准条件：电机关节杆合上，腿部往后摆顶到限位块，分别对外侧电机和内侧电极进行校准。因为外侧电机转子连接内侧电极定子，所以当机器进行升蹲时，内侧电机会和外侧电机转动**相反且相等**的角度

init_angle_hip	init_angle_hip则为前后的一个中值，腿部往后是内侧电机（hip）的0点，往前是5.2（大概数值），单位是弧度单位，程序初始化init_angle_hip = （0 + 5.12）/ 2 ≈ 2.6；
alpha	因为外侧腿部短连杆与地面法线的夹角，以机器抬头为例，逆时针pitch角度减小，-hip值增加，变化的角度一样，所以alpha不变，同理，低头也一样；
beta	以机器蹲下时为例，外侧电机会带动内侧电机转子转动，导致hip值发生变化，beta = knee->angle - alpha；
d_alpha	hip的角速度 + pitch角速度
d_beta	knee角速度-d_alpha

dd_alpha	利用d_alpha做差分求得
dd_beta	利用d_beta做差分求得
轮子圈数this->wheel_pos	轮子电机当前角度 + alpha
轮子速度wheel_vel	轮子电机当前角速度 + alpha角速度
knee_angle	外侧电机张角的一半

knee_angle	外侧电机张角的一半
机器腿长len	两连杆长度 * knee_angle；
伸缩腿速度d_len	对len求导；
伸缩腿加速度dd_len	

调节腿长长度

复制代码

```
    float ctrl_len(float set_len, float set_d_len, const float dt)
    {
        if (set_len > param->core.LEG_MAX_LEN)
            set_len = param->core.LEG_MAX_LEN;

        float set_angle = 2 * acosf(set_len / param->core.LEG_LEN_STRAIGHT) +
this->init_angle_knee,
                set_vel = 2 / sqrtf(param->core.LEG_LEN_STRAIGHT * param-
>core.LEG_LEN_STRAIGHT - set_len * set_len) * set_d_len;

        return pid_len.update(set_angle, set_vel, this->knee->angle, this-
>knee->angle_vel, dt);
    }
```

关节（外侧电机）动力学

```
float Leg2::ctrl_update_lift(const float set_accl, const bool offground, const float cos_roll)
{
    const float L = param->core.LEG_LEN_STRAIGHT / 2;
    float accl_out = set_accl * param->core.LEG_INERTIA/(4*L*L*this->sin_knee_psc*this->sin_knee_psc);

    if(offground)
    {
        const float mass_const = (param->core.WHEEL_MASS)*
            param->core.UPPER_MASS/(param->core.UPPER_MASS + param->core.WHEEL_MASS*2);
        accl_out += set_accl * mass_const * GRAVITY;
    }
    else
        accl_out += (set_accl + cos_roll) * param->core.UPPER_MASS * GRAVITY * load_psc;

    return accl_out;
}
```

计算膝关节力矩输出

复制代码

```
float Leg2::knee_output(const float accl_out, const float roll_out,
    const float high_limit, const float low_limit)
{
    const float L = param->core.LEG_LEN_STRAIGHT / 2;
    float out = (roll_out * sin_knee_psc - accl_out * sin_knee) * L;

    switch (config)
    {
        case LEG_CONFIG_FORWARD:
            out += (hip->torque + wheel->torque)/2;
            break;
        case LEG_CONFIG_BACKWARD:
            out += (hip->torque - wheel->torque)/2;
            break;
    }

    out += pos_limit(high_limit, low_limit);

    return out;
```

```
}
```

PID.cpp

PI控制器

```
float PID_Controller::update(const float target, const float input, const
float dt)
{
    float output, err = target - input;
    if(!isfinite(err))  return 0; //TODO CATCH BUG

    output = this->kp * err;
    if(this->ki)
    {
        this->err_int += this->ki * err * dt;
        bound((this->err_int), this->max_int);
        output += this->err_int;
    }

    bound(output, this->max_out);
    this->output = output;
    return output;
}
```

PID控制器

```
float PID_Controller::update(const float target, const float d_target,
    const float input, const float d_input, const float dt)
{
    float output,
          err   = target - input,
          d_err = d_target - d_input;
    if(!isfinite(err) || !isfinite(d_err))  return 0; //TODO CATCH BUG

    output = this->kp * err + this->kd * d_err;
    if(this->ki)
    {
        this->err_int += this->ki * err * dt;
        bound((this->err_int), this->max_int);
        output += this->err_int;
```

```
    }

    bound(output, this->max_out);
    this->output = output;
    return output;
}
```

pid控制器积分加误差阈值，目标超过一定值时，进行ki积分，否则ki积分输出为0.

复制代码

```
float PID_Controller::update(const float target, const float d_target,
    const float input, const float d_input, const float int_thresh_err,
    const float dt = 0)
{
    float output,
          err   = target - input,
          d_err = d_target - d_input;
    if(!isfinite(err) || !isfinite(d_err))  return 0; //TODO CATCH BUG

    output = this->kp * err + this->kd * d_err;
    if(this->ki && abs(err) > int_thresh_err)
    {
        this->err_int += this->ki * err * dt;
        bound((this->err_int), this->max_int);
        output += this->err_int;
    }
    else
    {
         this->err_int = 0;
    }
    bound(output, this->max_out);
    this->output = output;
    return output;
}
```

pid输出最大最小值限幅

复制代码

```
float PID_Controller::update(const float target, const float d_target,
    const float input,   const float d_input,
    const float Max_int, const float Min_int,
    const float Max_out, const float Min_out,
    const float dt)
```

```
{
    float output,
          err   = target - input,
          d_err = d_target - d_input;
    if(!isfinite(err) || !isfinite(d_err))  return 0; //TODO CATCH BUG

    output = this->kp * err + this->kd * d_err;
    if(this->ki)
    {
        this->err_int += this->ki * err * dt;
        bound((this->err_int), Max_int, Min_int);
        output += this->err_int;
    }

    bound(output, Max_out, Min_out);
    this->output = output;
    return output;
}
```

2.1.6 Estimation

Attitude_Comp.cpp

角速度更新，在初始化时会检验1500个周期角速度偏置，如果此时间内imu偏置yaw轴角速度超过0.04，则判断imu出现移动（1.出于安全考虑，机器正在初始化，防止人为移动机器，机器启动误伤人；2.若机器在初始化时，机器人在移动，当机器启动后，偏置是有误的，机器在初始化完之后，将会出现yaw轴漂移的情况），机器人将初始化失败，检测将重新开始，直到imu初始化成功。

复制代码

```
void Attitude_Complementary::update_gyro(const float gyro_meas[3], const float
stamp, float dt)
{
static float yaw_last_gyro = 0, yaw_gyro = 0,start_init = 1;
static uint32_t stop_count = 0;

    if(!data.init)
    {

    if(mabs(gyro_meas[2])>0.04)
    {

     memset(gyro_bias, 0, 12);
     memset(init_accl, 0, 12);
```

```
     data.update_cnt = 0;
     average_cnt = 0;
     return ;
    }
        if(average_cnt < 1500)
        {
            gyro_bias[0] += gyro_meas[0] / 1500.f;
            gyro_bias[1] += gyro_meas[1] / 1500.f;
            gyro_bias[2] += gyro_meas[2] / 1500.f;
             average_cnt++;
        }

        return;
    }

    data.rotation[0] = gyro_meas[0] - gyro_bias[0];
    data.rotation[1] = gyro_meas[1] - gyro_bias[1];
    data.rotation[2] = gyro_meas[2] - gyro_bias[2];

    deadzone(data.rotation[2],0.01f);

    if(!(dt > 0)) dt = init_dt_gyro;

    Quaternion dq(data.rotation, dt);
    data.q *= dq;
}
```

角加速度更新

复制代码

```
bool Attitude_Complementary::update_accl(const float accl_meas[3], const float
stamp, float dt)
{
    memcpy(data.accl, accl_meas, 12);

    if(!data.init)
    {
        float norm = vector_norm<float>(accl_meas, 3);
        if(norm > 1.05f || norm < 0.95f)  return false;

        init_accl[0] += accl_meas[0];
        init_accl[1] += accl_meas[1];
        init_accl[2] += accl_meas[2];
```

```
        data.update_cnt++;

        if(data.update_cnt < 20) return true;

        float Reb[3][3] = {0};
        memcpy(&(Reb[2][0]), init_accl, 12);
        vector_normalize<float>(Reb[2], 3);

        Reb[0][0] = Reb[2][2];
        Reb[0][1] = 0;
        Reb[0][2] = -Reb[0][2];
        vector_normalize<float>(Reb[0], 3);
        vector3_cross<float>(Reb[2], Reb[0], Reb[1]);

        data.q = Quaternion(Reb);
        if(data.update_cnt > 2000)
        {
            data.init = true;
            data.update_cnt = 2000;
        }


        return true;
    }
    else
    {
        if(!(dt > 0)) dt = init_dt_accl;

        float corr[3];
        float n_accl = vector_norm<float>(accl_meas, 3);

        float v2[3];
        Quaternion& q = data.q;
        v2[0] = 2*(q.x*q.z - q.w*q.y);
        v2[1] = 2*(q.y*q.z + q.w*q.x);
        v2[2] = 1 - 2*(q.x*q.x + q.y*q.y);
        for (uint8_t i = 0; i < 3; i++)
             data.linear_accl[i] = accl_meas[i] - v2[i];

        if(n_accl < 1.05f && n_accl > 0.95f)
        {
            float accel_corr[3], norm_accel[3];
            for (uint8_t i = 0; i < 3; i++)
                norm_accel[i] = accl_meas[i]/n_accl;
```

```
            vector3_cross(norm_accel, v2, accel_corr);
            for (uint8_t i = 0; i < 3; i++)
                corr[i] = accel_corr[i] * w_accl;

            if(data.rotation[0] < 0.1f && data.rotation[0] > -0.1f &&
               data.rotation[1] < 0.1f && data.rotation[1] > -0.1f &&
               data.rotation[2] < 0.1f && data.rotation[2] > -0.1f)
                for (uint8_t i = 0; i < 3; i++)
                {
                    gyro_bias[i] -= corr[i] * (w_gyro * dt);
                    bound<float>(gyro_bias[i], gyro_bias_max);
                }
        }
        else
            return false;

        Quaternion dq(corr, dt);
        data.q *= dq;
        data.update_cnt++;
        if(data.update_cnt > 5000)
        {
           data.update_cnt = 5000;
        }
        return true;
    }
}
```

Attitude_Estimation.cpp

陀螺仪pitch轴角度和角速度解算，用于机器头部点头和抬头姿态检测。

复制代码

```
void WB6_Base::pitch_estimate(void)
{
    float t0 = attitude.q.w * attitude.q.y - attitude.q.z * attitude.q.x;

    float c1 = 1.f - 2.f * (attitude.q.x * attitude.q.x + attitude.q.y *
attitude.q.y),
          c2 = 2.f * t0;
    float t1 = sqrtf(c1 * c1 + c2 * c2);

    float t2 = 2 * (attitude.q.w * attitude.q.x + attitude.q.y * attitude.q.z)
/ c1;
```

```
    float t3 = sqrtf(1 - 4 * t0 * t0);

    head_pitch = t0 < 0 ? -acosf(c1 / t1) : acosf(c1 / t1);

    if(mabs(head_pitch) > 2*M_PI)
    {
        head_pitch = pitch_last;
    }


    d_head_pitch = 2 * t2 * t0 * attitude.rotation[0] + attitude.rotation[1] -
t2 * t3 * attitude.rotation[2];

    if(mabs(d_head_pitch) > 100)
    {
        d_head_pitch = d_pitch_last;
    }

    pitch_last = head_pitch;
    d_pitch_last = d_head_pitch;
}
```

陀螺仪roll轴角度和角速度解算，用于机器左右倾斜姿态检测。

```
void WB6_Base::roll_estimate(void)
{
    float cos = cosf(leg_angle),
          sin = sinf(leg_angle);

    float t0 = 2 * (attitude.q.w * attitude.q.y + attitude.q.x *
attitude.q.z),//1/2s{\theta}+sin
          t1 = 1 - 2 * (attitude.q.y * attitude.q.y + attitude.q.z *
attitude.q.z),
          t2 = 2 * (attitude.q.w * attitude.q.x - attitude.q.y * attitude.q.z),
          t3 = 2 * (attitude.q.w * attitude.q.z + attitude.q.x * attitude.q.y);

    this->roll = -(sin * t0 - cos * t1) * (cos * t2 - sin * t3) + (cos * t0 +
sin * t1) * (sin * t2 + cos * t3);
    float t4 = (sin * 2 * (attitude.q.w * attitude.q.y - attitude.q.x *
attitude.q.z) - cos * (1 - 2 * (attitude.q.x * attitude.q.x + attitude.q.y *
attitude.q.y)));
```

```
    roll_err = (-roll - set_roll * t4) / sqrtf(set_roll * set_roll + 1);
//sine value of err angle, linearize to err angle
    ground_tilt = atanf(-roll / t4) - atanf((legL.len - legR.len) /
param.core.WHEEL_DISTANCE);

    float d_angle = cos * attitude.rotation[0] - sin * attitude.rotation[2];
    d_roll_err = set_d_roll - d_angle;

    //force arm estimation
    float tan_theta, cos_theta;
    if (this->offground)
    {
        tan_theta = 0;
        cos_theta = 1;
    }
    else
    {
        float curve_roll = -(this->vel_forward * this->vel_yaw) / GRAVITY;
        float curve_roll_err = (-roll - curve_roll * t4) / sqrtf(curve_roll *
curve_roll + 1);

        tan_theta = tanf(curve_roll_err);
        cos_theta = 1 / sqrtf(1 + tan_theta * tan_theta);
    }

    float temp_l = (param.core.WHEEL_DISTANCE / 2 + legL.len * tan_theta),
          temp_r = (param.core.WHEEL_DISTANCE / 2 - legR.len * tan_theta);

    if (temp_l < 0.02f)
        temp_l = 0.02f;
    if (temp_r < 0.02f)
        temp_r = 0.02f;

    legL.force_arm = temp_l * cos_theta;
    legR.force_arm = temp_r * cos_theta;

    float leg_force_arm_psc = legL.force_arm / legR.force_arm;//force arm
    legL.load_psc = 1 / (1 + leg_force_arm_psc);//L1 = / (L1 + L2)
    legR.load_psc = leg_force_arm_psc / (1 + leg_force_arm_psc);
}
```

轮子估计：轮子估计主要用于机器里程估计、速度估计、角速度估计、超速检测，此外还有一个驻坡功能，驻坡功能还能人为的牵引机器人前进和后退。

```
void WB6_Base::wheel_estimate(const float dt)
{
    static float pos_err_limit = 0.5f,err_cnt = 0,err_rec = 0;/*record the pos
when no input*/;
    static uint8_t pos_limit_state = 0,jump_status_last = 0;
    static float jump_status_cnt = 0;
    static float detect_ground_cnt = 0.f;
    static uint8_t offground_last = 0;
    static int8_t dir_flag = 1;
    float pos = (legL.wheel_pos + legR.wheel_pos) / 2 + (legL.wheel->rev +
legR.wheel->rev) * M_PI;
    //get the cmd direction
    pos_act = pos;

        if(dir_flag != 1u && cmd.forward.d_val > 0.1f && jump_status == 0)
        {
            dir_flag = 1;
        }
        else if(dir_flag != -1u &&  cmd.forward.d_val < -0.1f && jump_status
== 0)
        {
            dir_flag = -1;
        }
        if(mode != WB6_Base::STAND)
        {
            dir_flag = 0u;
        }

        if((legL.load < 40.f && legR.load < 40.f) || offground)
        {
            detect_ground_cnt = 0;
            forward_push_status = 0;
        }
        else if(offground == 0)
        {
            detect_ground_cnt += dt;
            if(detect_ground_cnt > 10.f)
            {
                detect_ground_cnt = 10.f;
            }
        }

        //when jumping, the pos err should remain
```

```
        //slide the robot
        if(mabs(cmd.forward.d_val) > 0.1f)
        {
            this->pos_rec = pos;
        }

        switch(forward_push_status)
        {
            case 0:
                if(dir_flag != 0u && mabs(cmd.forward.d_val) <= 0.1f &&
jump_status == 0 && detect_ground_cnt  >= 3.f)
                {
                    this->pos_rec = pos + vel_forward * 6.f; // slide the
overshoot according to the vel
                    dir_flag = 0;
                }
                else if(mabs(cmd.forward.d_val) <= 0.1f && dir_flag == 0
                        && mabs(this->pos_rec - pos) > 12.f
                        && mabs(vel_forward) > 0.2f
                        && jump_status == 0
                        && detect_ground_cnt  >= 3.f
                        && cmd.height.val > 0.18f) // robot have been pushed
                {
                    forward_push_status = 1;
                }
                break;
            case 1:
                this->pos_rec = pos + vel_forward * 6.f;
                if(mabs(vel_forward) < 0.2f || mabs(cmd.forward.d_val) > 0.1f)
                {
                    forward_push_status = 0;
                }
                break;
        }
    // if(mabs(cmd.forward.d_val) >= 0.01f)
    // {
    //     this->pos_rec = pos;
    // }
    // if(vel_forward >= 0.5f && dir_flag == 1)
    // {
    //     this->pos_rec = pos + 0.2f;
    // }
    // else if(vel_forward < -0.5f && dir_flag == -1)
    // {
    //     this->pos_rec = pos - 0.2f;
```

```
    // }
    // else
    // {
    //     this->pos_rec = pos;
    // }
    float err = (this->pos_rec  + cmd.forward.d_val * dt - pos) *
param.core.WHEEL_RADIUS; /*((cmd.forward.rev - (legL.wheel->rev + legR.wheel-
>rev)) * M_PI +
                 (cmd.forward.val - pos)) *
                param.core.WHEEL_RADIUS;*/


    if (err > pos_err_limit)
        err = pos_err_limit;
    else if (err < -pos_err_limit)
        err = -pos_err_limit;

    this->pos_forward_err = err / param.core.WHEEL_RADIUS;
    if(jump_status_last == 1 && jump_status > 1)
    {
        err_rec = pos_forward_err;
    }
    else if(jump_status_last != 0 && jump_status == 0)
    {
        this->pos_rec = pos + err_rec;
    }
    jump_status_last = jump_status;
    this->vel_forward = (legL.wheel_vel + legR.wheel_vel) / 2 *
param.core.WHEEL_RADIUS;

    // this->pos_yaw_err = (cmd.yaw.rev - 2 * attitude.yaw_rev) * M_PI +
cmd.yaw.val - attitude.yaw;

    this->vel_yaw = attitude.d_yaw;

    static float overspeed_dt = 0;

    float max_vel = (legL.wheel->max_vel + legR.wheel->max_vel)/2;
    if(mabs(vel_forward) > max_vel * param.core.WHEEL_RADIUS * 0.8f)
    {
        overspeed_dt += dt;
        if(overspeed_dt > 0.5f)
            wdg.setFlag(WHEEL_OVERSPEED, Watchdog::SEVERE);
        else
        {
```

```
            overspeed_dt = 0;
            float param_max_speed = this->client_max_speed_flag ? this-
>client_max_speed : param.core.FORWARD_MAX_VEL;
            if(mabs(vel_forward) < param_max_speed)
                wdg.clearFlag(WHEEL_OVERSPEED, Watchdog::SEVERE);
        }
    }
}
```

机器人里程 = 计算左轮 + 右轮的当前的里程+之前滚过的轮子的里程

复制代码

```
    float pos = (legL.wheel_pos + legR.wheel_pos) / 2 + (legL.wheel->rev +
legR.wheel->rev) * M_PI;
```

通过拨动遥控器遥感，判别前进和后退的标志位，前进dir_flag = 1，后退dir_flag = -1，在遥控器遥感没有输入，非跳跃的状态下，机器人触地一段时间内才会触发驻坡功能进行里程误差计算，机器人直行速度 = （左轮速度 + 右轮速度）/ 2，转向角速度 = imu反馈的角速度。

机器人tilt估计

tilt估计获取tilt的角度和角速度

复制代码

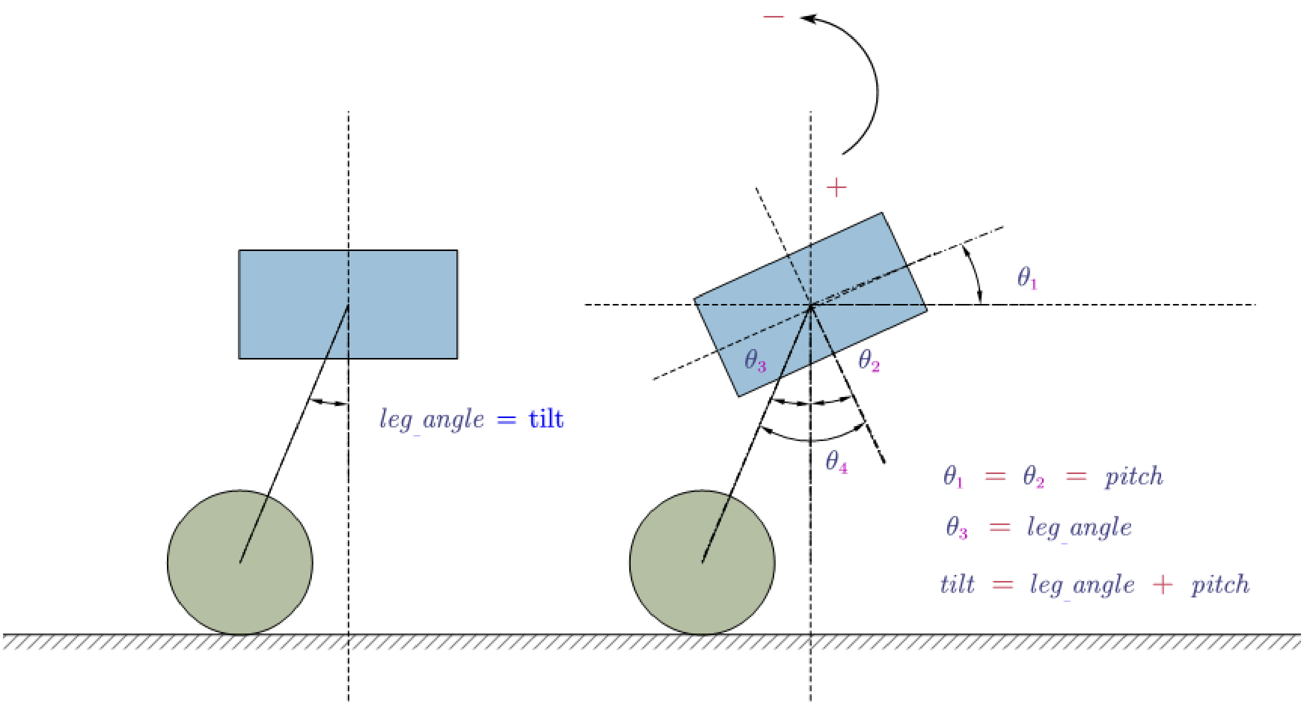
```
void WB6_Base::tilt_estimate(void)
{
    float ang0 = head_pitch + param.mass.body_ang,
          angl = legL.angle + head_pitch,
          angr = legR.angle + head_pitch;

    float xl = -legL.len * sinf(angl) - param.mass.body_d * cosf(ang0),
          xr = -legR.len * sinf(angr) - param.mass.body_d * cosf(ang0),
          zl =  legL.len * cosf(angl) - param.mass.body_d * sinf(ang0),
          zr =  legR.len * cosf(angr) - param.mass.body_d * sinf(ang0);

    legL.tilt = tilt_angle_update(atan2f(-xl, zl), legL.round_tilt,
legL.tilt_rev);
    legR.tilt = tilt_angle_update(atan2f(-xr, zr), legR.round_tilt,
legR.tilt_rev);
    tilt = tilt_angle_update(atan2f(-xl - xr, zl + zr), round_tilt, tilt_rev);

    legL.cos_tilt = zl / sqrtf(xl * xl + zl * zl);
```

```
    legR.cos_tilt = zr / sqrtf(xr * xr + zr * zr);
    cos_tilt = (zl + zr) / sqrtf((zl + zr) * (zl + zr) + (xl + xr) * (xl +
xr));

    if (legL.cos_tilt < 0.1f)
        legL.cos_tilt = 0.1f;
    if (legR.cos_tilt < 0.1f)
        legR.cos_tilt = 0.1f;
    if (cos_tilt < 0.1f)
        cos_tilt = 0.1f;

    legL.d_tilt = legL.d_tilt_lpf.update(legL.d_angle + attitude.rotation[1]);
    legR.d_tilt = legR.d_tilt_lpf.update(legR.d_angle + attitude.rotation[1]);
    d_tilt = (legL.d_tilt + legR.d_tilt) / 2;
}
```

leg_angle为机器腿部和头部法线的的夹角，tilt为腿部和地面法线的夹角，头部θ1为机器头部pitch角度，逆时针旋转方向为-号，顺时针旋转方向为+号。

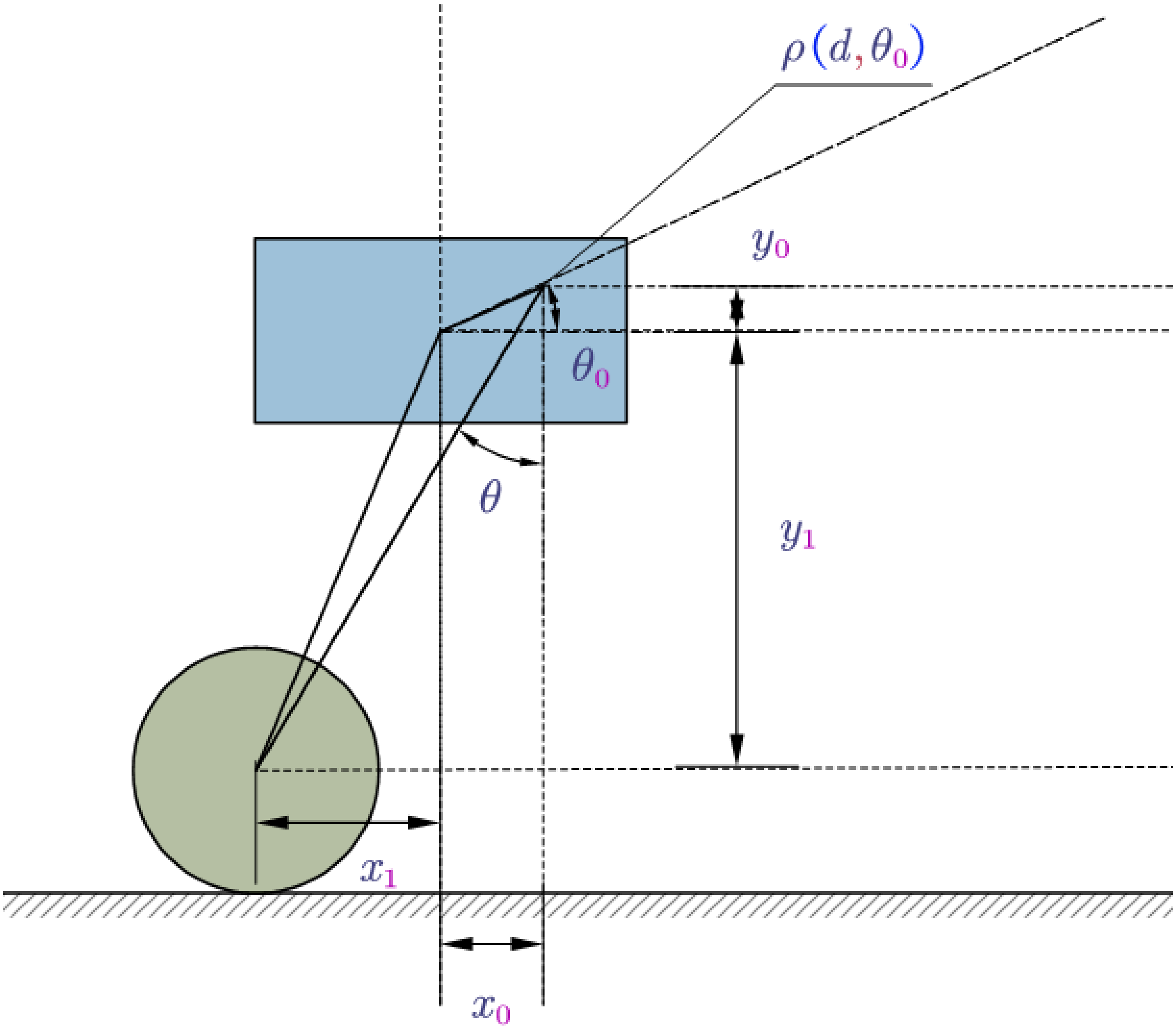
假设机器人质心在腿部和头部的连接处

1. 当机器人头部水平时，leg_angle = tilt.
2. 当机器人抬头时，θ1 = θ2 = pitch，tilt = leg_angle - θ2，∵pitch逆时针为-号，∴tilt = leg_angle + pitch;



但是机器人重心并不在腿部和头部的连接处，我们给头部质心在头部的某一点，以腿部和头部的连接处为原点，用极坐标ρ =（param.mass.body_d， param.mass.body_ang）表示头部质心的位置，这里的param.mass.body_d和param.mass.body_ang在初始化时设定的数值，这并不是准确的质心坐标，所以，目前的机器人质心是有问题的，后续需要优化。

所以实际的tilt = atan((x1 + x0) / (y0+y1))



2.1.7 Function

Automation_App.cpp

太空步由一个振幅为0.5，周期为1s的正弦波输入，结束太空步时，需要经历一个完整的输入周期才能结束，也就是fmodf(2.f*split_amp*M_PI*T,2*M_PI) < 0.05f。

复制代码

```
void Automation_App::SpaceWalk::update(Cmd& cmd, bool *start_flag,const float
dt)
{
    // if(!rc.automation) //exit automation
    // {
    //     mode = 0;
    //     return;
    // }

    cmd.forward_input = forward_val * cmd.forward_input;
    cmd.rotate_input  = cmd.rotate_input * 1.f;
    cmd.roll_input    = 0;
    cmd.split_input   = split_freq * sinf(2.f*split_amp*M_PI*T);
    cmd.pitch_input   = 0;
    //stop the action smoothly.
    if(cmd.automation == 0 && mabs(fmodf(2.f*split_amp*M_PI*T,2*M_PI)) < 0.05f)
    {
        *start_flag = false;//stop automation
    }

    Automation_App::Base::update(cmd, dt);
}
```

Init.cpp

复制代码

```
void Init_Handle::start(const float init_d_len, const float init_d_ang)
{
    this->len_ramp.val = this->ang_ramp.val = 0;

    this->init_d_len = init_d_len;
    this->init_d_ang = init_d_ang;

    robot->hip_activate = robot->leg_on =
    robot->balance = this->init = false;
    this->started = true;

    this->reset_trial = 0;
    robot->wdg.clearFlag(LEG_OVERLOAD, Watchdog::SEVERE);
    robot->wdg.clearFlag(RESET_FAIL,   Watchdog::CRITICAL);

    ctrl->transform(WB6_Base::CAR);
    this->reset(init_d_len, init_d_ang);
```

```
}
```

腿部收缩速度和角度斜坡速度置0；

在头文件Init.hpp处初始化，设置腿部缩腿速度**init_d_len = 0.4** 和回中角速度**init_d_ang = π / 2**；

将臀部标志位**robot->hip_activate**和伸腿标志位**robot->leg_on**平衡状态标志位**robot->balance**和机器人初始化标志位**this->init**置**false**；

将开始初始化标志位**this->started**置**true**；

将复位计数器**this->reset_trial = 0**;

清除过载标志位**robot->wdg.clearFlag(LEG_OVERLOAD, Watchdog::SEVERE)**；

清除复位标志位**robot->wdg.clearFlag(RESET_FAIL, Watchdog::CRITICAL)**；

切换模式到CAR，**ctrl->transform(WB6_Base::CAR)**；

复位设置腿部收缩速度和角度斜坡速度，**this->reset(init_d_len, init_d_ang)**；

复制代码

```
void Init_Handle::reset(const float init_d_len, const float init_d_ang)
{
    robot->wdg.clearFlag(FALLOVER_MASK, Watchdog::CRITICAL);

    this->init_d_len = init_d_len;
    this->init_d_ang = init_d_ang;
    this->init = false;

    if (this->reset_trial > 1)
    {
        this->reset_trial = 0;
        robot->wdg.setFlag(RESET_FAIL, Watchdog::CRITICAL);
        return;
    }
    else if (!this->reset_trial)
    {
        switch (robot->mode)
        {
        case WB6_Base::CAR:
        case WB6_Base::TRANSFORM_DOWN:
            this->reset_mode = WB6_Base::CAR;
            break;
        case WB6_Base::TRANSFORM_UP:
        case WB6_Base::STAND:
            this->reset_mode = WB6_Base::STAND;
```

```
            break;
        }
    }

    ctrl->transform(WB6_Base::CAR);
    this->reset_time = 0;
    this->reset_trial++;

    // reset leg length ramp
    init_left_len = robot->legL.len;
    init_right_len = robot->legR.len;

    float init_len = robot->param.core.LEG_LEN_RETRACT;

    float len_ramp_t = init_left_len > init_right_len ? (init_left_len - 
init_len) / init_d_len : (init_right_len - init_len) / init_d_len;
    if (len_ramp_t < 0.5f)
        len_ramp_t = 0.5f;

    len_ramp.reset(len_ramp_t);
    ang_ramp.reset(0.5f);
}
```

若已经进行复位，复位计数器this->reset_trial累加，计数器清0，复位标志位置1；

若解除复位，记录当前的机器人模式robot->mode，若robot->mode是CAR或者TRANSFORM_DOWN，this->reset_mode为CAR，若robot->mode是TRANSFORM_UP或者STAND，this->reset_mode = STAND；

len_ramp.reset(len_ramp_t)设置收腿的斜坡周期;
ang_ramp.reset(0.5f)设置回中的斜坡周期;

复制代码

```
void Init_Handle::update(const float dt)
{
    if(this->init)
    {
        reset_time += dt;
        if(reset_time > 0.5f)
            this->reset_trial = 0;
    }
    else
    {
        len_ramp.update(dt);
```

```
        //if(robot->leg_len < robot->param.core.LEG_LEN_RETRACT + 0.02f)
return;

        if(!robot->hip_activate)
        {
            //reset leg angle ramp
            init_diff =  robot->legL.angle - robot->legR.angle;
            init_ang  = (robot->legL.angle + robot->legR.angle)/2;

            float ang_err_l  = mabs(robot->legL.angle),
                  ang_err_r  = mabs(robot->legR.angle);
            float ang_ramp_t = ang_err_l > ang_err_r ? ang_err_l/init_d_ang :
ang_err_r/init_d_ang;
            if(ang_ramp_t < 0.05f)   ang_ramp_t = 0.05f;

            ang_ramp.reset(ang_ramp_t);

            robot->hip_activate = true;
        }

        ang_ramp.update(dt);
        if(len_ramp.finish() && ang_ramp.finish())
        {
            if(this->reset_mode == WB6_Base::STAND)
            {
                ctrl->transform(WB6_Base::TRANSFORM_UP);
            }

            this->init = true;
            robot->wdg.clearFlag(FALLOVER_MASK, Watchdog::CRITICAL);
        }
    }
}
```

若机器已经经过初始化，每经过0.5s将复位计数置0；

若没有经过初始化，跑腿部收缩和腿部回中的斜坡函数，腿部回中；

Jump.cpp

设置离地状态

复制代码

```
void Jump_Handle::set_offground(const bool result)
{
```

```
    if(result)
    {
        this->init_diff = (robot.mode == WB6_Base::CAR) ? 0 : robot.legL.len - robot.legR.len;
        this->init_tilt = robot.tilt;
        // this->dest_tilt = -robot.vel_forward * 0.1f; //TODO
       //if(mabs(robot.cmd.forward.d_val) > 0.1f )
        {
            if(phase == OFF)
            {
                robot.set_tilt = robot.param.imu_bias.y;
            }
            else
            {
                if(mabs(robot.cmd.forward.d_val) >= 0.1f)
                    robot.set_tilt     =   robot.param.imu_bias.y - robot.vel_forward * 0.1f;// this->init_tilt - 0.05f;
                else
                    robot.set_tilt     =   robot.param.imu_bias.y;
            }

            // if(robot.set_tilt  > 0.1f)
            // {
            //     robot.set_tilt =  0.1f;
            // }
            // else if(robot.set_tilt  < -0.1f )
            // {
            //     robot.set_tilt =  - 0.1f;
            // }
        }
        // else
        // {
        //     robot.set_tilt = robot.param.imu_bias.y;
        // }
        // else if(robot.vel_forward < 0)
        // {
        //     robot.set_tilt     = -robot.vel_forward * 0.1f;
        // }
        robot.set_leg_diff = this->init_diff;

        robot.offground = true;
        this->offground_time = 0;
        robot.legL.pid_angle.reset();//reset the integrals
        robot.legR.pid_angle.reset();
        ctrl.leg_diff_pid.reset();
```

```
        // this->tilt_ramp.reset(1.f);
        // this->lift_handle();
        // robot.pos_forward_err = 0;
    }
    else
    {
        this->init_pitch = robot.head_pitch;
        this->init_tilt = robot.tilt;
        this->init_damp = robot.force_d;

        robot.offground = false;
        robot.offground_stable = false;
        float t = mabs(robot.set_pitch - init_pitch)/M_PI*2;
        robot.set_d_pitch = 0;
        robot.set_pitch    = init_pitch;
        robot.set_tilt     = robot.param.imu_bias.y;


        this->pitch_ramp.reset(t);
        this->landing_handle();
    }
}
```

若result为true，左右腿长差init_diff = 左腿长度 - 右腿长度；

站立状态时，set_tilt = imu的y轴偏置，否则有前进或者后退指令时默认set_tilt和轮子速度有关，set_tilt为增大，机器人腿部往后摆，set_tilt变小，机器往前摆，例如驱使机器向前走，前面有掉落的阶梯，此时腿部需要向前摆，轮子速度向前，所以，set_tilt跟轮速负相关；

重置腿部、roll积分项输出为0；

若result为false,

设置头部回到设置位置斜坡函数的时间；

重置加速度积分项；

负载保护，机器站起来，超过负载时间大于8s则判断机器人过载，一般运动时超过负载时间比站起时的负载时间小；

负载大于一定值一定时间表示机器负载正常。

复制代码

```
void Jump_Handle::load_protection(const float dt)
{
```

```
    static const float MAX_LOAD = 150.f;
    float sin1 = sinf(robot.legL.knee_angle),
          sin2 = sinf(robot.legR.knee_angle);
    float sin = sin1 > sin2 ? sin1 : sin2;
    if(sin < 0.1f) sin = 0.1f;
    float max_load =  MAX_LOAD/sin,max_load_2 = MAX_LOAD * 0.4f/sin;

    static float overload_dt0 = 0,
                 overload_dt1 = 0;
    const float over_load_time_0 = 0.25,over_load_time_1 = 8;
    if(load_kf.load_ext_z > max_load)
    {
        overload_dt0 += dt;
        overload_dt1 += dt;
        if(robot.mode == WB6_Base::TRANSFORM_UP)

        {
            if(overload_dt0 > over_load_time_1)
                robot.wdg.setFlag(LEG_OVERLOAD, Watchdog::SEVERE);
        }
        else
        {
            if(overload_dt0 > over_load_time_0    )
                robot.wdg.setFlag(LEG_OVERLOAD, Watchdog::SEVERE);
        }

    }
    else if(load_kf.load_ext_z > max_load_2)
    {
        overload_dt0 = 0;
        overload_dt1 += dt;
        if(overload_dt1 > 0.25f)
            robot.wdg.setFlag(LEG_OVERLOAD, Watchdog::NORMAL);
    }
    else
    {
        robot.wdg.clearFlag(LEG_OVERLOAD, Watchdog::NORMAL);
        overload_dt1 = 0;
    }
}
```

负载估计

复制代码

```
void Jump_Handle::load_estimate(const float dt)
{

    load_kf.estimate_load(-robot.legL.hip->torque, robot.legL.knee->torque,
robot.legL.wheel->torque,
                          -robot.legR.hip->torque, robot.legR.knee->torque,
robot.legR.wheel->torque);

    this->load_protection(dt);

    static float state_cntL = 0,
                 state_cntR = 0,
                 stable_cnt = 0;
    float       lift_detect_th   = param.estimation.LIFT_DETECT_FORCE_TH,
                ground_detect_th = param.estimation.GROUND_DETECT_FORCE_TH;
    const float max_load   = param.estimation.GROUND_DETECT_MAX_FORCE_TH;

    const float      up_dt = param.estimation.LIFT_DETECT_DT,
                 ground_dt = param.estimation.GROUND_DETECT_DT,
                 stable_dt = 2.f;

    bool ground_detect = this->phase < RETRACT ||
        (this->phase == EXTEND &&
         this->offground_time > this->estimate_jump_duration
            - param.jump.DECEND_T + param.jump.EXTEND_DT +
param.jump.GROUND_DETECT_DT);

    if(robot.offground && phase == OFF)
    {
        stable_cnt += dt;
        if(stable_cnt > stable_dt)
        {
            robot.offground_stable = true;
        }
    }
    else
    {
        robot.offground_stable = false;
        stable_cnt = 0;
    }

    if(robot.mode == WB6_Base::CAR)
    {
        lift_detect_th = param.estimation.LIFT_DETECT_FORCE_TH_CAR;
        ground_detect_th = param.estimation.GROUND_DETECT_FORCE_TH_CAR;
```

```
    }

    if(!robot.left_offground && robot.mode != WB6_Base::TRANSFORM_UP &&
robot.mode != WB6_Base::TRANSFORM_DOWN)
    {
        if(robot.legL.load < lift_detect_th )
        {
            state_cntL += dt;
            if(state_cntL > up_dt || this->phase == RETRACT)
            {
                robot.left_offground  = true;
                state_cntL = 0;
            }
        }
        else
            state_cntL = 0;
    }
    else if(robot.left_offground && ground_detect /*&&
mabs(robot.leg_len_err) < 0.01*/)
    {
        if(robot.legL.load > ground_detect_th)
            state_cntL += dt;
        else
            state_cntL = 0;

        if(robot.legL.load > max_load || state_cntL > ground_dt)
        {
            robot.left_offground = false;
            state_cntL = 0;
        }
    }



    if(!robot.right_offground && robot.mode !=WB6_Base::TRANSFORM_UP &&
robot.mode != WB6_Base::TRANSFORM_DOWN)
    {
        if(robot.legR.load < lift_detect_th)
        {
            state_cntR += dt;
            if(state_cntR > up_dt || ground_detect)
            {
                robot.right_offground = true;
                state_cntR = 0;
            }
```

```
        }
        else
            state_cntR = 0;

    }
    else if(robot.right_offground )
    {
        if(robot.legR.load > ground_detect_th  /*&& mabs(robot.leg_len_err) < 
0.01*/)
            state_cntR += dt;
        else
            state_cntR = 0;

        if(robot.legR.load > max_load || state_cntR > ground_dt )
        {
            robot.right_offground = false;
            state_cntR = 0;
        }
    }
    //leave time for retract
}
```

机器人判断离地一段时间后，触发offground_stable，这个标志位目前主要用于检测机机器人是否被提起，使轮子停转。

左腿和右腿离地检测

在非离地和非升起和蹲下的状态，左腿负载低于一定值lift_detect_th时间**state_cntL > up_dt 或者蹬腿 之后收腿状态也会被判断离地this->phase == RETRACT。**

跳跃分段

复制代码

```
void Jump_Handle::update(const float dt)
{
    this->tilt_ramp.update(dt);
    if(robot.offground)
    {
        switch(phase)
        {
            case OFF:

                robot.force_p = 1.5f * param.core.IMPEDANCE_KP_LIFT;
                robot.force_d = param.core.IMPEDANCE_KD_LIFT;
                break;
```

```
            case EXTEND:
                robot.force_p = 1.9f * param.core.IMPEDANCE_KP;
                robot.force_d = 1.5f *  param.core.IMPEDANCE_KD;
                break;
            default:
                robot.force_p = 3.5f * param.core.IMPEDANCE_KP;
                robot.force_d = 2.1f *  param.core.IMPEDANCE_KD;
                break;
        }

        this->offground_time += dt;

    }
    else
    {
        this->offground_time = 0;
        this->pitch_ramp.update(dt);
        robot.force_p =  1.5f * param.core.IMPEDANCE_KP;
        robot.force_d =  1.2f * param.core.IMPEDANCE_KD;
        // robot.force_p = 1.5f * param.core.IMPEDANCE_KP;
        // robot.force_d = 1.2f *  param.core.IMPEDANCE_KD;
        switch(phase)
        {
            case RETRACT:
                robot.force_p = 6.5f * param.core.IMPEDANCE_KP;
                robot.force_d = 1.0f *  param.core.IMPEDANCE_KD;
                break;
            case EXTEND:
                robot.force_p = 4.5f * param.core.IMPEDANCE_KP;
                robot.force_d = 3.5f *  param.core.IMPEDANCE_KD;
                break;
        }
    }

    float setHeight  = this->set_height,
          setDHeight = this->set_d_height;
    switch(phase)
    {
        case OFF://not in jump mode
            if(!robot.offground)
            {

                setHeight = cmd.height.val;
            }
            else
```

```
            {
                    setHeight = cmd.height.val;
            }
            setDHeight = robot.set_dd_leg_len = 0;
            if(cmd.jump == Command::JUMP_CHARGE)// go to charge mode
                phase = CHARGE;
            break;
        case CHARGE://robot body down
            if(robot.cmd_mode == Command::LOST){
                        cmd.jump = Command::JUMP_STOP;

            }
            else if(cmd.jump == Command::JUMP_START)
            {
                if(setHeight > this->dest_height - 0.045f) //charge height too
small, abort jumping
                    cmd.jump = Command::JUMP_STOP;
                else
                    phase = LIFT;
            }

            robot.set_dd_leg_len = 0;
            //don't kneel too fast to avoid shrinking on the floor
            setDHeight = (cmd.jump == Command::JUMP_CHARGE) ? -
param.core.MAX_HEIGHT_VEL/1.2f :

param.core.MAX_HEIGHT_VEL / 4.f;

            setHeight  += setDHeight * dt;



            if(cmd.jump == Command::JUMP_STOP && setHeight >= cmd.height.val)
            {
                setHeight  = cmd.height.val;
                setDHeight = 0;
                phase = OFF;
            }
            else if(setHeight <= param.core.MIN_HEIGHT)
            {
                setHeight  = param.core.MIN_HEIGHT;
                setDHeight = 0;
            }
            break;
        case LIFT:
```

```
            robot.set_dd_leg_len = 0;
            setHeight  = param.core.LEG_LEN_STRAIGHT;
            setDHeight = 6.0f;
            if(robot.legL.len > this->dest_height || robot.legR.len > this-
>dest_height)//if any of the leg  have exeeded the height
            {
                const float wheel_mass = param.core.WHEEL_MASS;
                this->lift_vel = (robot.legL.d_len + robot.legR.d_len)/2;
                this->lift_len = robot.leg_len + 0.03;

                float vel = this->lift_vel *
param.core.UPPER_MASS/(param.core.UPPER_MASS + 2*wheel_mass);
                float diff_len = (param.jump.DECEND_PT - this-
>lift_len)*param.core.UPPER_MASS/(param.core.UPPER_MASS + 2*wheel_mass);
                this->estimate_jump_duration = (vel + sqrtf(vel * vel -
2*GRAVITY*diff_len))/GRAVITY;

                ascend.plan_ascend(lift_len, lift_vel);
                phase = RETRACT;
            }
            break;
        case RETRACT: // the lefg goes back
            ascend.update(dt);
            setHeight  = ascend.SetPos();
            setDHeight = ascend.SetVel();
            robot.set_dd_leg_len = ascend.SetAccl()/GRAVITY;

            if(this->offground_time > this->estimate_jump_duration
                - param.jump.DECEND_T + param.jump.EXTEND_DT)
            {
                decend.plan_decend(robot.leg_len+0.03, 0);
                phase = EXTEND;
            }
            break;
        case EXTEND://the leg goes back  to the ground
            decend.update(dt);
            setHeight  = decend.SetPos();
            setDHeight = decend.SetVel()/1.2f;
            robot.set_dd_leg_len = decend.SetAccl()/GRAVITY;

            if(!robot.offground)    phase = FINISH;
            break;
        case FINISH:
            ascend.reset();
            decend.reset();
```

```
            robot.set_dd_leg_len = 0;
            if(cmd.jump != Command::JUMP_START)
                phase = OFF;
            break;
    }

    this->set_height   = setHeight;
    this->set_d_height = setDHeight;
}
```

phase 参考值	含义	备注
OFF	不在跳跃状态	机器人触地后即视为不在跳跃状态
LIFT	上升状态	
CHARGE	下蹲	
RETRACT	蹬腿后收腿的状态	
EXTEND	蹬腿	
FINISH	完成跳跃	

这里使用了分段PID对跳跃的每一个阶段对腿部的控制。每个阶段设置机器人不同的高度，表示机器人升起和蹲下，

在LIFT阶段表示机器人起跳，设置伸腿长度和伸腿速度；

Planner.cpp

轮子命令复位

复制代码

```
void WB6_Planner::wheel_cmd_reset(const float forward_err, const float yaw_err)
{
    cmd.forward.val    = (legL.wheel_pos + legR.wheel_pos)/2 + forward_err;
```

```
    cmd.forward.d_val = (legL.wheel_vel + legR.wheel_vel)/2;
    cmd.forward.rev   = legL.wheel->rev + legR.wheel->rev;

    while(cmd.forward.val > M_PI)
    {
        cmd.forward.val -= 2*M_PI;
        cmd.forward.rev += 2;
    }

    while(cmd.forward.val < -M_PI)
    {
        cmd.forward.val += 2*M_PI;
        cmd.forward.rev -= 2;
    }

    this->yaw_cmd_reset();
}
```

yaw命令复位

复制代码

```
void WB6_Planner::yaw_cmd_reset(void)
{
    cmd.yaw.val   = robot.attitude.yaw;
    cmd.yaw.rev   = robot.attitude.yaw_rev * 2;
    cmd.yaw.d_val = robot.attitude.d_yaw;
}
```

轮子命令

复制代码

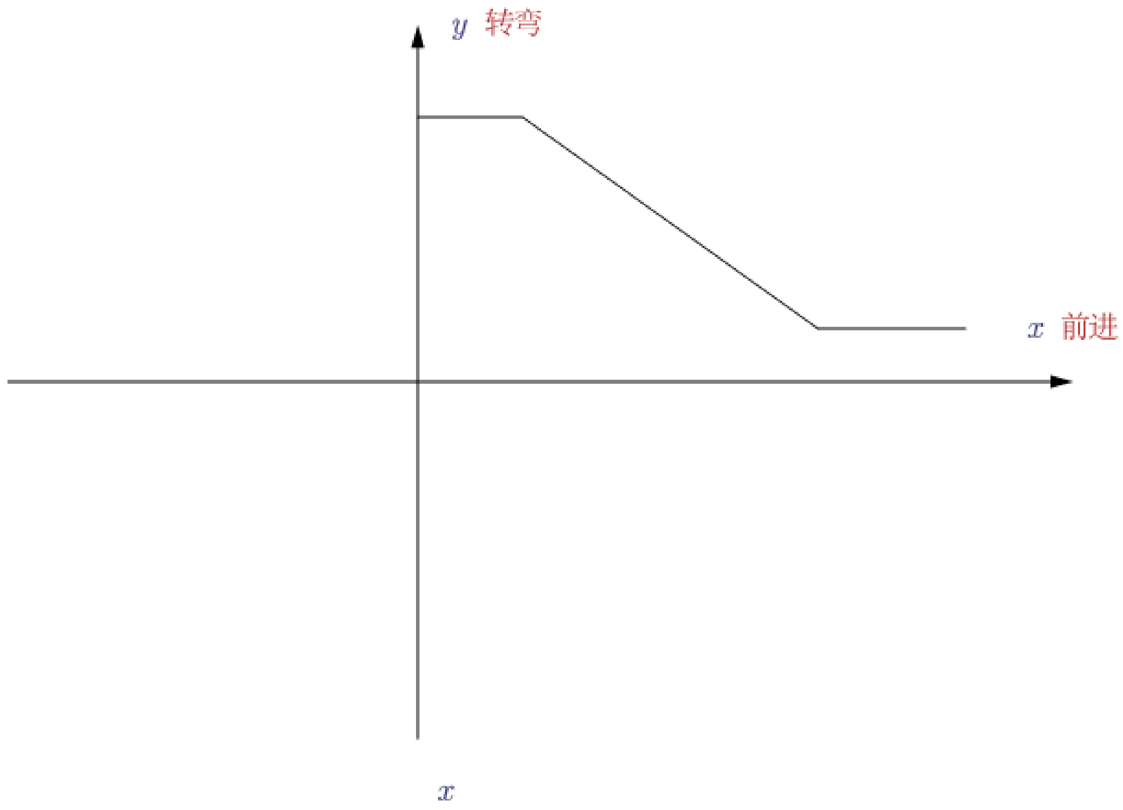
```
void WB6_Planner::wheel_cmd(float forward_input/*velocity m/s*/, float
rotation_input/*yaw, rad/s*/, const float dt)
{
    if(robot.offground && !cmd.cross_mode)
    {
        this->wheel_cmd_reset();
        return;
    }
    float accl_limit = 0,accl_limit_minus = 0;
    if(robot.balance)
    {
```

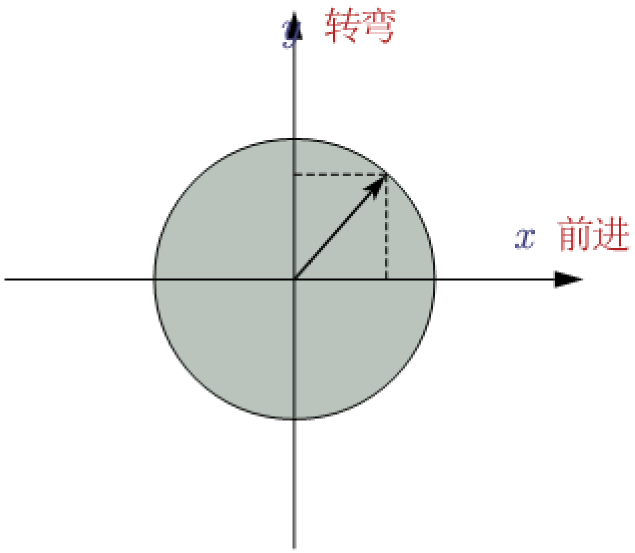
```
        accl_limit = 0.5f * param.core.ACCL_LIMIT * (robot.leg_len - 
param.core.LEG_MIN_LEN) / (param.core.LEG_LEN_STRAIGHT - 
param.core.LEG_MIN_LEN) +  0.5f * param.core.ACCL_LIMIT ; //accelerate from 
0.5a ~ 1a
        accl_limit_minus = -(accl_limit / param.core.FORWARD_MAX_VEL) * 
cmd.forward.d_val*param.core.WHEEL_RADIUS - accl_limit;
        accl_limit = -(accl_limit / param.core.FORWARD_MAX_VEL) * 
cmd.forward.d_val*param.core.WHEEL_RADIUS + accl_limit;
    }
    else
    {
        accl_limit =  0.5f * param.core.ACCL_LIMIT;
        accl_limit_minus = -accl_limit;
    }
    //31V as full
    float max_vel_psc = robot.power->Vin / 31.f;
    bound(max_vel_psc,1.f,0.6f);


    bound(accl_limit,param.core.ACCL_LIMIT);
    bound(accl_limit_minus,param.core.ACCL_LIMIT);
    //bound the rotate input
    bound(rotation_input,param.core.ROTATE_MAX_VEL);
    //bound the forward input
    bound(forward_input,param.core.FORWARD_MAX_VEL * max_vel_psc);

    float rotation_input_linear = rotation_input * param.core.WHEEL_DISTANCE / 
2.f * 2.f;//get the yaw linear speed

    bound(rotation_input_linear,param.core.FORWARD_MAX_VEL * 
max_vel_psc);//avoid sqrt a minus number

    float max_forward_vel = sqrt(param.core.FORWARD_MAX_VEL * 
param.core.FORWARD_MAX_VEL * max_vel_psc * max_vel_psc
                                 - rotation_input_linear * 
rotation_input_linear);//get the maximum forward vel

    float min_forward_vel = -max_forward_vel;


    float forward_input_del = (param.core.FORWARD_MAX_VEL*max_vel_psc - 
mabs(forward_input) + 0.5f);
    if(forward_input_del > 1.f)
    {
        forward_input_del = 1;
```

```
    }
      float rotate_limit = param.core.ANG_ACCL_LIMIT *
          (1 - 0.5f*robot.height/param.core.LEG_LEN_STRAIGHT) *
forward_input_del;
    //slope for forward
    float max_vel_cmd = cmd.forward.d_val*param.core.WHEEL_RADIUS + accl_limit
* dt,//d_val :angle velocity, not linear
          min_vel_cmd = cmd.forward.d_val*param.core.WHEEL_RADIUS +
accl_limit_minus * dt;
    bound(forward_input, max_vel_cmd, min_vel_cmd);//m/s
    //bound the final forward input
    bound(forward_input,max_forward_vel,min_forward_vel);

    //slope for yaw
    float max_ang_cmd = cmd.yaw.d_val + rotate_limit * dt,
          min_ang_cmd = cmd.yaw.d_val - rotate_limit * dt;

    //(param.core.FORWARD_MAX_VEL,3)   (x1,y1)
    //(param.core.FORWARD_MAX_VEL / 2.f,5)   (x2,y2)
    float forward_vel_x = robot.vel_forward;
    //get the slope of the maximum angle input
    bound(forward_vel_x,param.core.FORWARD_MAX_VEL *
max_vel_psc,param.core.FORWARD_MAX_VEL / 2.f);
    float max_ang_k = (2.f - param.core.ROTATE_MAX_VEL) /
                        (param.core.FORWARD_MAX_VEL * max_vel_psc -
param.core.FORWARD_MAX_VEL / 2.f) ; //obtain the k of the slope

    float max_ang_y =  max_ang_k * (forward_vel_x - param.core.FORWARD_MAX_VEL
* max_vel_psc) + 2.f;

    bound(rotation_input, max_ang_cmd, min_ang_cmd);//rad/s
    bound(rotation_input,max_ang_y);

    cmd.forward.input_rev(forward_input / param.core.WHEEL_RADIUS, dt);//rad/s

  //  if(robot.left_offground && robot.right_offground)
   //     this->yaw_cmd_reset();
  //  else
        cmd.yaw.input_rev(rotation_input, dt);//yaw.d_val update in here
}
```

1. 设定加速度最大最小值
2. 速度比例系数max_vel_psc和电池电压有关

3. 对速度比例系数max_vel_psc限幅，对加速度限制accl_limit、accl_limit_minus、rotation_input、forward_input进行限幅
4. 机器人在高速转弯的时候，电机速度达到最大速度，电机无法提供多余的速度给机器人，导致机器人摔倒，为了解决这个问题，我们对前进的速度进行了限制，通过遥控器遥感的输入，如下图，将前进的速度x进行限制，转弯和前进的速度合成在圆形限制范围内。
5. 在高速状态下，转弯角速度在低速状态下会有最大转弯速度，在高速状态下会有最小转弯速度，呈现阶梯-斜坡分布。





高度命令输入

```
void WB6_Planner::height_cmd(float input, float dt)
{
    if(robot.set_pitch > robot.pitch_limit_angle)   robot.pitch_limit = true;
    if(robot.pitch_limit && robot.set_pitch < robot.pitch_limit_angle && input
> robot.pitch_limit_height)
        robot.pitch_limit = false;

    float min = robot.pitch_limit ? robot.pitch_limit_height :
param.core.MIN_HEIGHT;
    cmd.height.input_limit(input, dt, param.core.MAX_HEIGHT_VEL, -
param.core.MAX_HEIGHT_VEL, param.core.MAX_HEIGHT, min);
}
```

机器人pitch过大，下蹲会触碰地面，限制了蹲下的高度。为了防止头部触碰到地面，当机器蹲下的低于某一高度，pitch增大时，机器人提升高度。

pitch命令输入

```
void WB6_Planner::pitch_cmd(float input, float dt)
{
```

```
    float max = robot.set_leg_len < robot.pitch_limit_height ?
robot.pitch_limit_angle + 0.01 : param.core.MAX_PITCH;
     if(cmd.jump == Command::JUMP_STOP)
     {
            if(robot.last_flag)
            {
             robot.last_pitch_val = robot.set_pitch ;
            }
            else
            {
                robot. last_flag = 1;
                cmd.pitch.val = robot.last_pitch_val;
            }
     }
     if(cmd.jump == Command::JUMP_CHARGE || robot.mode ==
WB6_Base::TRANSFORM_DOWN)
     {
        robot.last_flag = 0;
        cmd.pitch.input_limit(input, dt, param.core.MAX_PITCH_VEL, -
param.core.MAX_PITCH_VEL, 0.1, -0.5);
     }

     else
     {
        cmd.pitch.input_limit(input, dt, param.core.MAX_PITCH_VEL, -
param.core.MAX_PITCH_VEL, max, -param.core.MAX_PITCH);
     }
}
```

机器人太空步（腿部劈叉）输入

复制代码

```
void WB6_Planner::split_cmd(float input, float dt)
{
    cmd.leg_split.input_limit(input, dt, 3.0f, -3.0f, 1.0f, -1.0f);
    robot.set_leg_split   = cmd.leg_split.val;
    robot.set_d_leg_split = cmd.leg_split.d_val;
}
```

roll角度命令

复制代码

```
void WB6_Planner::roll_cmd(const float input, const float dt)
{
    const float LEG_MAX_DIFF_ANGLE = atanf((param.core.LEG_MAX_LEN -
param.core.LEG_MIN_LEN)/param.core.WHEEL_DISTANCE);

    if(!robot.balance || robot.mode  == WB6_Base::TRANSFORM_UP)
    {
        robot.set_roll   = robot.ground_tilt;
        robot.set_d_roll = 0;

        return;
    }

    float cmd_forward = cmd.forward.d_val * param.core.WHEEL_RADIUS,
          cmd_rotate  = cmd.yaw.d_val;
    float result = -(cmd_forward * cmd_rotate)/GRAVITY;
    deadzone(result, 0.1f);
    result = 0;
    robot.roll_curve = result;
    float Cmd   = input + robot.roll_curve,
          d_Cmd = 0;

    float max_cmd = tanf(robot.ground_tilt + LEG_MAX_DIFF_ANGLE),
          min_cmd = tanf(robot.ground_tilt - LEG_MAX_DIFF_ANGLE);
    if(max_cmd >  param.core.MAX_ROLL) max_cmd =  param.core.MAX_ROLL;
    if(min_cmd < -param.core.MAX_ROLL) min_cmd = -param.core.MAX_ROLL;

    if(Cmd < min_cmd)
    {
        Cmd   = min_cmd;
        d_Cmd = 0;
    }
    else if(Cmd > max_cmd)
    {
        Cmd   = max_cmd;
        d_Cmd = 0;
    }

    float maxRoll = robot.set_roll + param.core.MAX_ROLL_RATE*dt,
          minRoll = robot.set_roll - param.core.MAX_ROLL_RATE*dt;

    if(Cmd > maxRoll)
    {
        robot.set_roll   = maxRoll;
        robot.set_d_roll = 0;
```

```
    }
    else if(Cmd < minRoll)
    {
        robot.set_roll   = minRoll;
        robot.set_d_roll = 0;
    }
    else
    {
        robot.set_roll   = Cmd;
        robot.set_d_roll = d_Cmd;
    }
}
```

腿长命令

复制代码

```
void WB6_Planner::leg_len_cmd(const float cmd, const float d_cmd, const bool
jump)
{
    float cmd_len = cmd/robot.cos_tilt;

    float d_length = mabs(legL.len - legR.len)/2;
    float max_len = (param.core.LEG_MAX_LEN - d_length), min_len = 0;
    if(robot.mode == WB6_Base::STAND)
        min_len = (param.core.LEG_MIN_LEN + d_length);

    if(robot.mode ==  WB6_Base::TRANSFORM_UP)
    {
         min_len = (param.core.LEG_MIN_LEN  + 0.01);
    }
    else
    {
        min_len = (param.core.LEG_LEN_RETRACT);
    }
    if(!jump && cmd_len > max_len)
    {
        robot.set_leg_len   = max_len;
        robot.set_d_leg_len = 0;
    }
    else if(!jump && cmd_len < min_len)
    {
        robot.set_leg_len   = min_len;
        robot.set_d_leg_len = 0;
    }
```

```
    else
    {
        robot.set_leg_len    = cmd_len;
        robot.set_d_leg_len = d_cmd/robot.cos_tilt +
        cmd*sinf(robot.tilt)/(robot.cos_tilt*robot.cos_tilt)*robot.d_tilt;
    }
}
```

Task.cpp

命令更新

复制代码

```
void WB6_Task::cmd_update(float dt)
{
    static float cross_cnt = 0;
    static uint8_t automation_last = 0;
    bound(dt, 0.05f, 0.f);
    robot.cmd_enable[Command::cmd_mode_t::RC] = RC.connect_flag;
    robot.cmd_enable[Command::cmd_mode_t::SDK_MOVEMENT] =
sdk_movement.connect_flag;
    if(robot.wdg.check(POWER_CHARGING, Watchdog::NORMAL))//robot charging,
stop moviong robot
    {
        robot.cmd_enable[Command::cmd_mode_t::RC] = 0;
        robot.cmd_enable[Command::cmd_mode_t::SDK_MOVEMENT] = 0;
    }
    // cmd_mode is set in here
    robot.cmd_switch();

    RC_Cmd* rc = NULL;
    Movement_Cmd* movement = NULL;

    rc = &RC;

    if (rc->kill)
    {
        wdg.setFlag(RC_KILL, Watchdog::CRITICAL);
        cmd.kill = true;
    }
    else
        wdg.clearFlag(RC_KILL, Watchdog::CRITICAL);
```

```
    float param_max_speed = this->robot.client_max_speed_flag ? this-
>robot.client_max_speed : param.core.FORWARD_MAX_VEL;
    if(this->robot.client_max_speed > 2.f)// the maximum speed is limited to
2m/s
    {
        this->robot.client_max_speed = param.core.FORWARD_MAX_VEL;
    }

    switch (robot.cmd_mode)
    {
        case Command::cmd_mode_t::LOST:
            cmd.forward_input = 0;
            cmd.rotate_input = 0;
            cmd.pitch_input = 0;
            cmd.jump = Command::JUMP_STOP;
            rc->enable_jump = Command::JUMP_STOP;
            rc ->jump = Command::JUMP_STOP;
            break;
        //Remote control control
        case Command::cmd_mode_t::RC:
        {
            cmd.pitch.ctrl_mode = Command::INPUT_VEL;
            //move the robot forward when crossing

            float speed_psc = rc->speed_mode ? 1.f : 0.7f;
            float speed_psc_rotate = rc->speed_mode ? 1.f : 0.5f;
            {


             if(rc->move_forward<0)
             {
                speed_psc = 0.7;
                speed_psc_rotate = 0.5f;
             }
             if(rc->automation > 0)
             {
                speed_psc = 0.7;
                speed_psc_rotate = 0.5f;
             }
            if(robot.mode == WB6_Base::CAR)
            {
                speed_psc_rotate = 0.5f;
                // speed_psc = 0.5f;
            }

```

```
            cmd.forward_input =  rc->move_forward * param_max_speed *
speed_psc;
            cmd.rotate_input  =  rc->move_left * param.core.ROTATE_MAX_VEL *
speed_psc_rotate * param_max_speed / param.core.FORWARD_MAX_VEL;// adjust the
rotation from forward speed
            }

            cmd.height_input  =  param.core.MIN_HEIGHT + rc->height *
(param.core.MAX_HEIGHT - param.core.MIN_HEIGHT);
            cmd.pitch_input   =  rc->head_up * param.core.MAX_PITCH_VEL;//vel
of pitch, not the angle
            cmd.roll_input    = -rc->tilt_left * param.core.MAX_ROLL_SET;
            cmd.split_input   = 0;
            cmd.head_ctrl_mode = rc->head_ctrl_mode;
            cmd.automation = rc->automation;
            automation_last = cmd.automation;

            //when crash flag is on, take away the control
            if (rc->enable_transform && !cmd.crash_flag)
                cmd.transform = rc->transform;
            break;
        }
        //SDK movement control
        case Command::cmd_mode_t::SDK_MOVEMENT:
        {
            movement = &sdk_movement;
            if(movement->ctrl_mode_change)
            {
                cmd.head_ctrl_mode =      movement->head_ctrl_mode;
                cmd.yaw.ctrl_mode =       movement->yaw_ctrl_mode;
                cmd.hip_split.ctrl_mode = movement->split_ctrl_mode;
                cmd.pitch.ctrl_mode =     movement->pitch_ctrl_mode;
                cmd.roll.ctrl_mode =      movement->roll_ctrl_mode;
                cmd.height.ctrl_mode =    movement->height_ctrl_mode;
                cmd.automation = movement->automation;
                movement->ctrl_mode_change = false;
            }

            float max_speed = movement->head_ctrl_mode ? 1.0f *
param_max_speed : 0.75f * param_max_speed;
            if(movement->forward > max_speed)
                cmd.forward_input = max_speed;
            else if(movement->forward < -max_speed)
                cmd.forward_input = -max_speed;
            else
```

```
                cmd.forward_input = movement->forward;
            // cmd.forward_input = 0.0f;

            if(movement->left > param.core.ROTATE_MAX_VEL)
                cmd.rotate_input = param.core.ROTATE_MAX_VEL;
            else if(movement->left < -param.core.ROTATE_MAX_VEL)
                cmd.rotate_input = -param.core.ROTATE_MAX_VEL;
            else
                cmd.rotate_input = movement->left;

            if(cmd.height.ctrl_mode == Command::INPUT_VEL)
            {
                if(movement->up > param.core.MAX_HEIGHT_VEL)
                    cmd.height_input = param.core.MAX_HEIGHT_VEL;
                else if(movement->up < -param.core.MAX_HEIGHT_VEL)
                    cmd.height_input = -param.core.MAX_HEIGHT_VEL;
                else
                    cmd.height_input = movement->up;
            }
            else
                cmd.height_input = param.core.MIN_HEIGHT + movement->up *
(param.core.MAX_HEIGHT - param.core.MIN_HEIGHT);

            if(movement->pitch > param.core.MAX_PITCH_VEL)
                cmd.pitch_input = param.core.MAX_PITCH_VEL;
            else if(movement->pitch < -param.core.MAX_PITCH_VEL)
                cmd.pitch_input = -param.core.MAX_PITCH_VEL;
            else
                cmd.pitch_input = movement->pitch;

            if(movement->roll > param.core.MAX_ROLL_SET)
                cmd.roll_input = param.core.MAX_ROLL_SET;
            else if(movement->roll < -param.core.MAX_ROLL_SET)
                cmd.roll_input = -param.core.MAX_ROLL_SET;
            else
                cmd.roll_input = movement->roll;

            cmd.transform = movement->transform;
            // cmd.forward_input = 0.0f;
            // cmd.rotate_input = 0.0f;
            // cmd.height_input = 0.26f;
            // cmd.pitch_input = 0.0f;
            //  cmd.roll_input = 0.0f;
            //  cmd.transform = Command::TRANSFORM_IDLE;
            break;
```

```
        }
        case Command::CALIBRATION_APP:
            calibration.update(cmd, dt);
            break;
        default:
            break;
    }

    switch (cmd.jump)
    {
        case Command::JUMP_STOP:
            //启动跳跃的条件：无warning,不在CAR，按下跳跃键
            if(jumpHandle.phase != jumpHandle.OFF)
            {
                cmd.roll_input = 0;
            }
            if (!wdg.Warning() &&
                ((rc->jump == Command::JUMP_CHARGE && rc->enable_jump) ||
movement->jump == Command::JUMP_CHARGE ) &&
                robot.mode != WB6_Base::CAR && robot.mode !=
WB6_Base::TRANSFORM_DOWN)
                {
                    cmd.jump = Command::JUMP_CHARGE;
                }
            break;
        case Command::JUMP_CHARGE:
            if (wdg.Warning())
                cmd.jump = Command::JUMP_STOP;
            else
            {
                if (rc->jump == Command::JUMP_STOP && movement->jump ==
Command::JUMP_STOP)
                {
                    cmd.jump = Command::JUMP_START;
                }
            }
            break;
        //allow another jump only after prev jump finishes
        case Command::JUMP_START:
           cmd.roll_input = 0;
            if (jumpHandle.finish())
            {
                cmd.jump = Command::JUMP_STOP;
                cmd.cross_mode = false;
            }
```

```
            break;
    }

    if(robot.mode == WB6_Base::STAND  && mabs(robot.vel_forward) < 1.0f &&
mabs(robot.vel_yaw) < 0.5f &&  !cmd.cross_mode ) //do not start
    {
        automation.cmd_app(cmd, jumpHandle.phase == Jump_Handle::OFF);
    }
    if(!cmd.cross_mode)
    {
        automation.update(cmd, dt);
    }


    if (cmd.calibration)
        calibration.start();
}
```

robot.cmd_mode参考值	含义	备注
Command::cmd_mode_t::LOST	遥控器丢失命令	
Command::cmd_mode_t::RC	遥控器模式	
Command::cmd_mode_t::SDK_MOVEMENT	SDK模式	
Command::CALIBRATION_APP	校准模式	
Command::JUMP_START	跳跃控制	

规划更新

复制代码

```
void WB6_Task::plan_update(float dt)
{
    static uint8_t int_flag = 1;
    bound(dt, 0.05f, 0.f);

    if ( robot.attitude.init){
```

```
        if(!robot.initialize_check(this->stamp))
       return;
    }


    if (!initHandle.started)
        initHandle.start();

    bool init = initHandle.init;
    initHandle.update(dt);

    {
        System_Lock lock;

        float setPitch, setDPitch,
            setTilt = param.imu_bias.y,
            setDTilt = 0;
            if (robot.head_pitch > robot.param.core.PITCH_PROTECT_MAX)
                wdg.setFlag(PITCH, Watchdog::CRITICAL);
            if (robot.head_pitch < -robot.param.core.PITCH_PROTECT_MAX)
                wdg.setFlag(PITCH << 1, Watchdog::CRITICAL);
        if (!initHandle.init)//init not finish
        {
            robot.set_tilt = initHandle.leg_angle(setTilt);
            /*if(cmd.crash_flag && cmd.crash_count >4000 ){
                cmd.crash_count = 0;
                robot.set_d_tilt = 0;
                cmd.crash_flag = 0;
            }
            else*/
                robot.set_d_tilt = initHandle.d_leg_angle(setDTilt);
        }
        else if(jumpHandle.phase <=  jumpHandle.CHARGE && robot.offground )
        {

            // robot.set_tilt = jumpHandle.tilt(setTilt);
            // robot.set_d_tilt = jumpHandle.d_tilt(setTilt, setDTilt);
        }


        if (robot.hardware_check() || !initHandle.init)
        {
            planner.wheel_cmd_reset();
            robot.set_leg_split  = initHandle.leg_diff();
            robot.set_d_leg_diff = 0;
```

```
            return;
        }
        //change the robot.mode
        this->transform_handler(cmd.transform);
        //处理跳跃与腿长
        jumpHandle.update(dt);


        planner.split_cmd(cmd.split_input, dt);
        planner.wheel_cmd(cmd.forward_input, cmd.rotate_input, dt);
        planner.roll_cmd(cmd.roll_input + param.imu_bias.x, dt);
        if (robot.cmd_mode != Command::LOST)
        {
            planner.height_cmd(cmd.height_input, dt);
        }
//check the static point first
        switch (robot.mode)
        {
        case WB6_Base::CAR:
            robot.balance = false;
            break;
        case WB6_Base::STAND:
            robot.balance = true;
/*
            if(robot.offground && !jumpHandle.start_jump_flag())// off ground
and give a slope
            {
                downHandle.set_height_dest(robot.param.jump.DAMP_LEN);
                downHandle.update(dt);

planner.leg_len_cmd(downHandle.height(jumpHandle.set_height),0);//give a slope
            }
            else*/

            planner.leg_len_cmd(jumpHandle.set_height,
                                jumpHandle.set_d_height,
                                jumpHandle.onJump());
                //downHandle.start();
                //downHandle.height_ramp.reset(3);
            planner.pitch_cmd(cmd.pitch_input, dt);
            setPitch  = cmd.pitch.val;
            setDPitch = cmd.pitch.d_val;
            robot.jump_status = jumpHandle.phase;
            break;
        case WB6_Base::TRANSFORM_UP:
```

```
        robot.balance = true;
       // upHandle.wheel_ramp.setVal(5,0);
       if(!cmd.crash_flag)
       {
            if(robot.roll_err <= 0.1)
            {
            upHandle.update(dt);
            planner.leg_len_cmd(upHandle.height(cmd.height.val),0);
            }
            setPitch = upHandle.head_angle(cmd.pitch.val);
            setDPitch = upHandle.d_head_angle(cmd.pitch.val);
            robot.set_roll = upHandle.roll(robot.set_roll);

            if (upHandle.finish())
            {
                ctrl.transform(WB6_Base::STAND);//ctrl mode change
            }
       }

        break;
    case WB6_Base::TRANSFORM_DOWN:


        downHandle.update(dt);
        planner.leg_len_cmd(downHandle.height(cmd.height.val),0);
        // if(cmd.height.val > 0.15)
        //     setPitch = downHandle.head_angle(cmd.pitch.val)-0.1;
        // else
        setPitch = downHandle.head_angle(cmd.pitch.val);
        // setDPitch = 0;
        // setDPitch = downHandle.d_head_angle(cmd.pitch.val);

        robot.set_roll = downHandle.roll(robot.set_roll);

        if (downHandle.finish())
        {
            robot.time_cnt = 0;
            ctrl.transform(WB6_Base::CAR);
            planner.wheel_cmd_reset();
           jumpHandle.tilt_cmd_reset();
        }
        break;
    case WB6_Base::REVERSE_TRANSFORM_UP:
        Reverse_Transform_Up_Plan(dt);
        break;
```

```
        }

        robot.set_pitch = jumpHandle.pitch(setPitch);

//          robot.set_d_pitch = 0;
    }
}
```

1. 初始化检测
2. 机器人模式this->transform_handler(cmd.transform);
3. 处理跳跃与腿长jumpHandle.update(dt);
4. 腿部劈叉指令更新planner.split_cmd(cmd.split_input, dt);
5. 轮子更新
6. roll轴更新
7. 高度更新
8. 站立姿态下有腿长更新，pitch角度更新
9. 站起规划
10. 趴下规划

控制更新

复制代码

```
bool WB6_Task::ctrl_update(const float curr_stamp)
{
    float dt = curr_stamp - this->stamp;
    if (dt > 0.01f)
        dt = 0.01f;
    this->stamp = curr_stamp;
    obs_dt = dt;

    //update the posture of the robot
    Robot_Estimation(dt);
    //check whether to kill the robot
    if(Ctrl_Error_Handler() == false)
    {
        return false;
    }
```

```
    Knee_Ctrl(dt);

    Tilt_Ctrl(dt);

    Wheel_Ctrl(dt);



    Reverse_Transform_Up_Ctrl(dt);

    Crash_Handler(dt);

    //wheel controller, calculate the output via the load
    ctrl.Wheel_Output_Calc(robot.offground && jumpHandle.offground_time >
0.5f);

    Output_Mix(dt);
    //all output
    ctrl.output();

    return true;
}
```

负载估计

复制代码

```
void WB6_Task::Robot_Estimation(float dt)
{

    static float over_time = 0;
    //update the status of the leg
    robot.legL.update(stamp, robot.head_pitch, robot.d_head_pitch);
    robot.legR.update(stamp, robot.head_pitch, robot.d_head_pitch);

    robot.roll_estimate();
    robot.pitch_estimate();
    robot.wheel_estimate(dt);
    robot.tilt_estimate();
    //update some control parameters
    robot.leg_len    = (robot.legL.len + robot.legR.len) / 2;
    robot.leg_angle = (robot.legL.angle + robot.legR.angle) / 2;
    robot.leg_split =  robot.legL.angle - robot.legR.angle;
    robot.height     =  (robot.legL.len * robot.legL.cos_tilt + robot.legR.len
* robot.legR.cos_tilt)/2.f;//robot.leg_len * robot.cos_tilt;
```

```
    jumpHandle.load_estimate(dt);


    if(
        !robot.offground &&
        ((robot.left_offground || robot.right_offground  || jumpHandle.phase
==jumpHandle.RETRACT))
    )
     over_time += dt;

    if(over_time>0.3f){
    // jumpHandle.set_offground(true);

    }
    //robot.set_tilt will be changed in here when offground
    if(
        !robot.offground &&
        ((robot.left_offground && robot.right_offground ) || jumpHandle.phase
==jumpHandle.RETRACT)
    )
    {
        jumpHandle.set_offground(true);
    }




    else if(robot.offground && jumpHandle.phase != jumpHandle.RETRACT &&
            !robot.left_offground && !robot.right_offground)
    {
        jumpHandle.set_offground(false);
        over_time = 0;
    }

}
```

摔倒站立控制

复制代码

```
void WB6_Task::Crash_Handler(float dt)
{
 //strategy in different mode of the robot.
    switch (robot.mode)
```

```
    {
    //Robot to the ground
        case WB6_Base::CAR:

        /*机器人摔倒复位回到原位之后再让它升起*/
        if(cmd.crash_flag == 1&& mabs(robot.vel_forward) < 0.1 )
        {
            cmd.forward.d_val = 0;
            cmd.crash_count ++;
            if(cmd.crash_count > 5000)
            {
                {
                    reset_handler();
                    cmd.crash_count = 0;
                    robot.set_d_tilt = 0;
                    cmd.crash_flag = 0;
                }
            }
        }
        // if crash, wheel speed goes to 0
        else if(cmd.crash_flag == 1)
        {
            cmd.forward.d_val = 0;
        }
        break;
    //The robot stands up

        case WB6_Base::STAND:

        /*判断机器人摔倒*/


        if(robot.height < 0.07f ||
            ((mabs(robot.legL.tilt) > 1.2f || mabs(robot.legR.tilt) > 1.2f)
            /*|| ( mabs(robot.leg_split) > 1.2f) */
            || (mabs(robot.legL.wheel_vel) > robot.legL.wheel->max_vel * 0.8f
                && mabs(robot.legR.wheel_vel) > robot.legR.wheel->max_vel *
0.8f

                && robot.tilt > 0.8f))
            && cmd.crash_flag == 0)
        {
            cmd.crashing_count++;
            if(cmd.crashing_count > 200)
            {
                ctrl.transform(WB6_Base::CAR);
```

```
                planner.wheel_cmd_reset();
                jumpHandle.tilt_cmd_reset();
                cmd.transform = Command::TRANSFORM_IDLE;
                cmd.crashing_count = 0;
                cmd.crash_flag  = 1;
                cmd.crash_count = 0;
            }
        break;
        }
        else
        {
            cmd.crashing_count = 0;
        }
        break;
    case WB6_Base::TRANSFORM_UP:
    case WB6_Base::TRANSFORM_DOWN:
        if((mabs(robot.legL.tilt) > 1.2f || mabs(robot.legR.tilt) > 1.2f))
        {
                ctrl.transform(WB6_Base::CAR);
                planner.wheel_cmd_reset();
                jumpHandle.tilt_cmd_reset();
                cmd.transform = Command::TRANSFORM_IDLE;
                cmd.crashing_count = 0;
                cmd.crash_flag  = 1;
                cmd.crash_count = 0;

        }
        else
        {
            cmd.crashing_count = 0;
        }
    break;

     }
     if(cmd.crash_flag == 1)
     {
        robot.legL.tilt_out = 0;
        robot.legR.tilt_out  = 0;

        robot.legL.split_out = 0;
        robot.legR.split_out = 0;
        robot.legL.knee_out = 0;
        robot.legR.knee_out = 0;
        robot.legR.yaw_out = 0;
        robot.legL.yaw_out = 0;
```

```
    }
}
```

Transform.cpp

机器人升起规划

复制代码

```
void Transform_Up_Handle::update(const float dt)
{
    if(height_ramp.val == 0 &&
        (robot.tilt > 0.2f     || robot.tilt < -0.2f ||
         robot.leg_split > 0.4f || robot.leg_split < -0.4f)
      ) return; //do not start if tilt angle too big

    height_ramp.update(dt);
//     height_ramp.val    = 1;
    if(robot.height > robot.param.core.LEG_LEN_RETRACT + 0.03f) robot.balance
= true;

    if(balance_init)    wheel_ramp.update(dt);
}
```

机器人趴下规划

复制代码

```
void Transform_Down_Handle::update(const float dt)
{

    if(!robot.offground )
    {
        this->dest_height = robot.param.core.LEG_LEN_RETRACT;

        if(robot.height < robot.param.core.LEG_LEN_RETRACT + 0.03f)
        {
            if(robot.balance)
                planner.wheel_cmd_reset();

            robot.balance = false;
            this->reversible = false;
            this->retract_time += dt;
        }

```

```
        if(robot.roll_err <= 0.1 && (robot.time_cnt > 500 ||
mabs(robot.head_pitch) < 0.3f) )
        {
            height_ramp.update(dt);
            // time_cnt = 0;
        }
        else
        {
        pitch_ramp.update(dt);
        robot.time_cnt++;
        }
    }
    else
    {
        height_ramp.update(dt);
        pitch_ramp.update(dt);
    }


}
```

2.1.8 Interface

cmd.cpp

记录过去的数值

复制代码

```
void Command::Channel::input_rev(float val_in, const float dt)//reverse
{
    this->d_val = val_in;
    this->val += val_in*dt;
    if(val > M_PI)
    {
        val -= 2*M_PI;
        rev += 2;
    }
    else if(val < -M_PI)
    {
        val += 2*M_PI;
        rev -= 2; //used internally, increment by 2
    }
}
```

输入限幅

复制代码

```
void Command::Channel::input_limit(float val_in, const float dt,
    const float up_limit_d, const float low_limit_d, const float up_limit,
const float low_limit)
{
    float max_val_in, min_val_in;
    switch (ctrl_mode)
    {
        case INPUT_POS:

            if(val_in > up_limit)
            {
                val = up_limit;
                d_val = 0;
                return;
            }
            else if(val_in < low_limit)
            {
                val = low_limit;
                d_val = 0;
                return;
            }

            max_val_in = val + up_limit_d*dt;
            min_val_in = val + low_limit_d*dt;
            if(val_in > max_val_in)
            {
                d_val = up_limit_d;
                val = max_val_in;
            }
            else if(val_in < min_val_in)
            {
                d_val = low_limit_d;
                val = min_val_in;
            }
            else
            {
                d_val = (val_in - val)/dt;
                val = val_in;
            }

            break;
```

```
        case INPUT_VEL:
            if(val_in > up_limit_d)         val_in = up_limit_d;
            else if(val_in < low_limit_d)  val_in = low_limit_d;
            this->val += val_in * dt;
            if(this->val > up_limit)
            {
                this->val = up_limit;
                this->d_val = 0;
            }
            else if(this->val < low_limit)
            {
                this->val = low_limit;
                this->d_val = 0;
            }
            else
                this->d_val = val_in;

            break;
    }
}
```

Joint_motor.cpp

输出物理单位转换

复制代码

```
void Joint_motor::setVoltage(const float voltage)
{
    this->vq_set = voltage;
}

void Joint_motor::setCurrent(const float current)
{
    this->iq_set = current;

    if(mode != CTRL_CURRENT)
        this->setVoltage(current*phase_resistance + angle_vel*bemf_constant);
}

void Joint_motor::setTorque(const float torque)
{
    float torque_set = torque;
    bound(torque_set, this->max_torque, this->min_torque);
```

```
    this->torque_set = torque_set;

    if(mode != CTRL_TORQUE)
        this->setCurrent(this->torque_set*torque_constant);
}
```

电机数据更新

复制代码

```
// maximum output update
void Joint_motor::update(const float angle_vel, const float V_in)//Can try to change into current loop for a better control in leg
{
    this->max_vel     = V_in/bemf_constant;

    this->max_torque = ( V_in - bemf_constant * angle_vel)/phase_resistance / torque_constant;//calculate the actual maximum output  based on the motor model.
    if(this->max_torque > this->max_output_torque)
        this->max_torque =  this->max_output_torque;

    this->min_torque = (-V_in - bemf_constant * angle_vel)/phase_resistance / torque_constant;
    if(this->min_torque < -this->max_output_torque)
        this->min_torque = -this->max_output_torque;
}
// actual status output
void Joint_motor::update(const float angle, const float angle_vel, const float iq, const float V_in)
{
    this->iq = iq;
    this->angle_vel = angle_vel;
    this->torque = iq/torque_constant;

    if(angle - this->round_angle < -M_PI)       this->rev++;
    else if(angle - this->round_angle > M_PI)   this->rev--;
    this->round_angle = angle;
    this->angle = this->rev*2*M_PI + this->round_angle;

    this->update(angle_vel, V_in);// update the maximum output value.

    // TODO: UPDATE PROTOCOL OR NOT
}
```

2.1.9 Utility

crc16_IBM.c

查表法crc16计算

复制代码

```
#include  <stdint.h>
#include "CRC16_IBM.h"

/*
 * CRC lookup table for bytes, generating polynomial is 0x8005
 * input: reflexed (LSB first)
 * output: reflexed also...
 */
static const uint16_t crc_ibm_table[256] = {
  0x0000, 0xc0c1, 0xc181, 0x0140, 0xc301, 0x03c0, 0x0280, 0xc241,
  0xc601, 0x06c0, 0x0780, 0xc741, 0x0500, 0xc5c1, 0xc481, 0x0440,
  0xcc01, 0x0cc0, 0x0d80, 0xcd41, 0x0f00, 0xcfc1, 0xce81, 0x0e40,
  0x0a00, 0xcac1, 0xcb81, 0x0b40, 0xc901, 0x09c0, 0x0880, 0xc841,
  0xd801, 0x18c0, 0x1980, 0xd941, 0x1b00, 0xdbc1, 0xda81, 0x1a40,
  0x1e00, 0xdec1, 0xdf81, 0x1f40, 0xdd01, 0x1dc0, 0x1c80, 0xdc41,
  0x1400, 0xd4c1, 0xd581, 0x1540, 0xd701, 0x17c0, 0x1680, 0xd641,
  0xd201, 0x12c0, 0x1380, 0xd341, 0x1100, 0xd1c1, 0xd081, 0x1040,
  0xf001, 0x30c0, 0x3180, 0xf141, 0x3300, 0xf3c1, 0xf281, 0x3240,
  0x3600, 0xf6c1, 0xf781, 0x3740, 0xf501, 0x35c0, 0x3480, 0xf441,
  0x3c00, 0xfcc1, 0xfd81, 0x3d40, 0xff01, 0x3fc0, 0x3e80, 0xfe41,
  0xfa01, 0x3ac0, 0x3b80, 0xfb41, 0x3900, 0xf9c1, 0xf881, 0x3840,
  0x2800, 0xe8c1, 0xe981, 0x2940, 0xeb01, 0x2bc0, 0x2a80, 0xea41,
  0xee01, 0x2ec0, 0x2f80, 0xef41, 0x2d00, 0xedc1, 0xec81, 0x2c40,
  0xe401, 0x24c0, 0x2580, 0xe541, 0x2700, 0xe7c1, 0xe681, 0x2640,
  0x2200, 0xe2c1, 0xe381, 0x2340, 0xe101, 0x21c0, 0x2080, 0xe041,
  0xa001, 0x60c0, 0x6180, 0xa141, 0x6300, 0xa3c1, 0xa281, 0x6240,
  0x6600, 0xa6c1, 0xa781, 0x6740, 0xa501, 0x65c0, 0x6480, 0xa441,
  0x6c00, 0xacc1, 0xad81, 0x6d40, 0xaf01, 0x6fc0, 0x6e80, 0xae41,
  0xaa01, 0x6ac0, 0x6b80, 0xab41, 0x6900, 0xa9c1, 0xa881, 0x6840,
  0x7800, 0xb8c1, 0xb981, 0x7940, 0xbb01, 0x7bc0, 0x7a80, 0xba41,
  0xbe01, 0x7ec0, 0x7f80, 0xbf41, 0x7d00, 0xbdc1, 0xbc81, 0x7c40,
  0xb401, 0x74c0, 0x7580, 0xb541, 0x7700, 0xb7c1, 0xb681, 0x7640,
  0x7200, 0xb2c1, 0xb381, 0x7340, 0xb101, 0x71c0, 0x7080, 0xb041,
  0x5000, 0x90c1, 0x9181, 0x5140, 0x9301, 0x53c0, 0x5280, 0x9241,
  0x9601, 0x56c0, 0x5780, 0x9741, 0x5500, 0x95c1, 0x9481, 0x5440,
  0x9c01, 0x5cc0, 0x5d80, 0x9d41, 0x5f00, 0x9fc1, 0x9e81, 0x5e40,
  0x5a00, 0x9ac1, 0x9b81, 0x5b40, 0x9901, 0x59c0, 0x5880, 0x9841,
```

```
    0x8801, 0x48c0, 0x4980, 0x8941, 0x4b00, 0x8bc1, 0x8a81, 0x4a40,
    0x4e00, 0x8ec1, 0x8f81, 0x4f40, 0x8d01, 0x4dc0, 0x4c80, 0x8c41,
    0x4400, 0x84c1, 0x8581, 0x4540, 0x8701, 0x47c0, 0x4680, 0x8641,
    0x8201, 0x42c0, 0x4380, 0x8341, 0x4100, 0x81c1, 0x8081, 0x4040,
};

uint16_t crc16_ibm_update_byte(uint16_t crc, const uint8_t c)
{
    const unsigned char lut = (crc ^ c) & 0xFF;
    return (crc >> 8) ^ crc_ibm_table[lut];
}

/**
 * crc_ibm - recompute the CRC for the data buffer
 * @crc - previous CRC value
 * @buffer - data pointer
 * @len - number of bytes in the buffer
 */
uint16_t crc16_ibm_calc(const uint8_t *buffer, size_t len)
{
    uint16_t crc = 0;
    while (len--)
        crc = crc16_ibm_update_byte(crc, *buffer++);
    return crc;
}

void crc16_ibm_update(uint8_t *const buffer, const size_t len)
{
    uint16_t crc = crc16_ibm_calc(buffer, len-2);
    *(uint16_t*)(&buffer[len-2]) = crc;
}

uint8_t crc16_ibm_verify(const uint8_t *const buffer, const size_t len)
{
    uint16_t crc = crc16_ibm_calc(buffer, len-2);
    return crc == *(uint16_t*)(&buffer[len-2]);
}
```

watchdog.cpp

看门狗标志位检查、标志位设立、清除标志位、电机错误码标志位设立

复制代码

```cpp
#include "Watchdog.hpp"

bool Watchdog::check(uint32_t flag, error_level_t level)
{
    switch (level)
    {
        case CRITICAL:
            return *error          & flag;
        case SEVERE:
            return *severe_warning & flag;
        case NORMAL:
            return *warning        & flag;

    }
    return false;
}


void Watchdog::setFlag(uint32_t flag, error_level_t level)
{
    if(!this->enable)   return;

    switch (level)
    {
        case CRITICAL:
            *error          |= flag;
            break;
        case SEVERE:
            *severe_warning |= flag;
            break;
        case NORMAL:
            *warning        |= flag;
            break;
    }
}

void Watchdog::clearFlag(uint32_t flag, error_level_t level)
{
    switch (level)
    {
        case CRITICAL:
            *error          &= (~flag);
            break;
        case SEVERE:
            *severe_warning &= (~flag);
```

```
            break;
        case NORMAL:
            *warning        &= (~flag);
            break;
    }
}

void Watchdog::setMotorError(const uint8_t error, const uint8_t id)
{
    if(!this->enable)   return;
    motor_error[id] = error;
}
```

3. A1.1机器人参数

3.1 刹车距离

实验环境如下:

机器向前行走致1.58m/s时，forward_input为遥控器输入的期望速度，vel_forward为机器实际前进速度，robot_distance为机器人滑行距离，当forward_input<=0时开始计算robot_distance。

实验方法	实测结果	备注
到达指定目标点后，松开摇杆，测量机器人滑行距离，测量3次取平均值	滑行图1 滑行图2 滑行图3,测量3次滑行距离求得平均值=0.97m.实距离3次求平均值(90+100+120)/3=1.0m	
到达指定目标点后，反拉摇杆，测量机器人滑行距离，测量3次取平均值	遥感反拉图1 遥感反拉图2 遥感反拉图3 测量3次反拉刹车滑行距离求得平均值=0.83m.实际距离3次求平均值(80+84+75)/3=0.80m	
到达指定目标点后，斜推摇杆，测量机器人滑行距离，测量3次取平均值	斜推遥感图1 斜推遥感图2 斜推遥感图3 测量3次斜推遥感刹车滑行距离求得平均值=0.35m.实际距离3次求平均值(50+50+50)/3=0.50m	车子会旋转， 实际车最外侧电机为最

3.2 越障能力

低速档测量

双轮直接过坎:4cm

单轮过坎:4cm

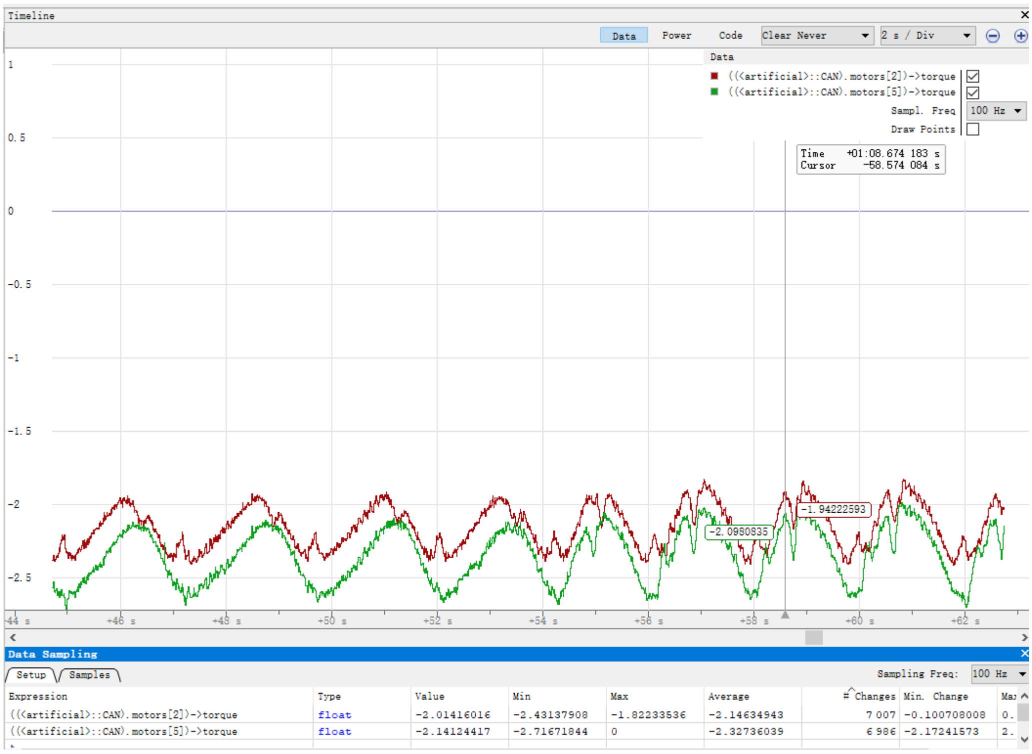
跳跃过坎:7.5cm

冲刺上坡：高度20cm,滑行距离60cm~90cm

3.3 静态负载站 33°坡，并推算 35kg下能否站坡

主要测站坡时的轮子的电流 / 扭矩

1.在15°坡时，静态电流是多少，反推摩擦系数



在15°坡时，两个轮子的静态电流的平均值为2.2Nm,机器在斜坡38°时发生滑动，摩擦系数tan38° = 0.78129 。

2.在33°坡时，能否稳在坡上，转矩是多少?

在33°坡时，能稳在坡上,两个轮子转矩平均值4.5Nm.