实时调节lqr参数workflow

📌 现在的控制逻辑，是在Robot_Class::STAND状态下启用新lqr相关算法，在其他模态下并不启用此lqr算法

```
void legWheelDrive::Output_Mix(float dt)
{
  if (robot.mode != Robot_Class::CAR) {
    ctrl.set_lqr_yaw(dt, initHandle.get_init_val());
  }

  if (robot.flag_ptr->balance) {
    ctrl.set_lqr(dt);
    // this->lwd_calc_tau(dt);
  }
  switch (robot.mode) {
    case Robot_Class::CAR:

      break;
    case Robot_Class::STAND:
      if (robot.flag_ptr->offground && planner.jumpHandle.phase == planner.jumpHandle.OFF) {
        ctrl.left_forward_out = ctrl.right_forward_out = ctrl.drive(dt);
      }
      static bool JUMPs = false;

      if (robot.cmd.enable_jump && lwd_get_mode() == STAND || JUMPs) {
        lwd_set_mode(JUMP);
        cjump.lwd_jump_set_phase(lwdJump::LIFT);
        cjump.lwd_jump_set_ycmd(ycmd);
        cjump.lwd_jump_set_jump(0.0, 0.25);
      }

      switch (this->lwd_mode)
      {
        case STAND:
          break;
        case JUMP:
        this->lwd_ctrl_jump(dt);
          JUMPs = false;
          break;
        default:
          break;
      }
      this->lwd_calc_tau(dt);
      break;
```

状态量

lqr相关的控制参数共为12个，分别为：

pitch, d_pitch,int_pitch, tilt, p, d_tilt, d_p, split, delta_p, d_split, d_delta_p, int_p

下面来做出解释：

对于 倒立摆及lqr控制 文档中的平面3dof模型，存在pitch，tilt，和p三个状态量。由于tita两条腿，将两条腿融合成一条腿，便产生其他衍生状态量：

设置左腿状态量为：pitch_l，tilt_l，和p_l，右腿的状态量为：pitch_r，tilt_r，和p_r pitch = pitch_l + pitch_r;

tilt = tilt_l + tilt_r;

p = p_l + p_r;

split = tilt_l - tilt_r;

delta_p = p_l - p_r;

对应的d_ 和int_分别指对应的微分和积分。

见对应代码ctrl_lqr.cpp 中的此处

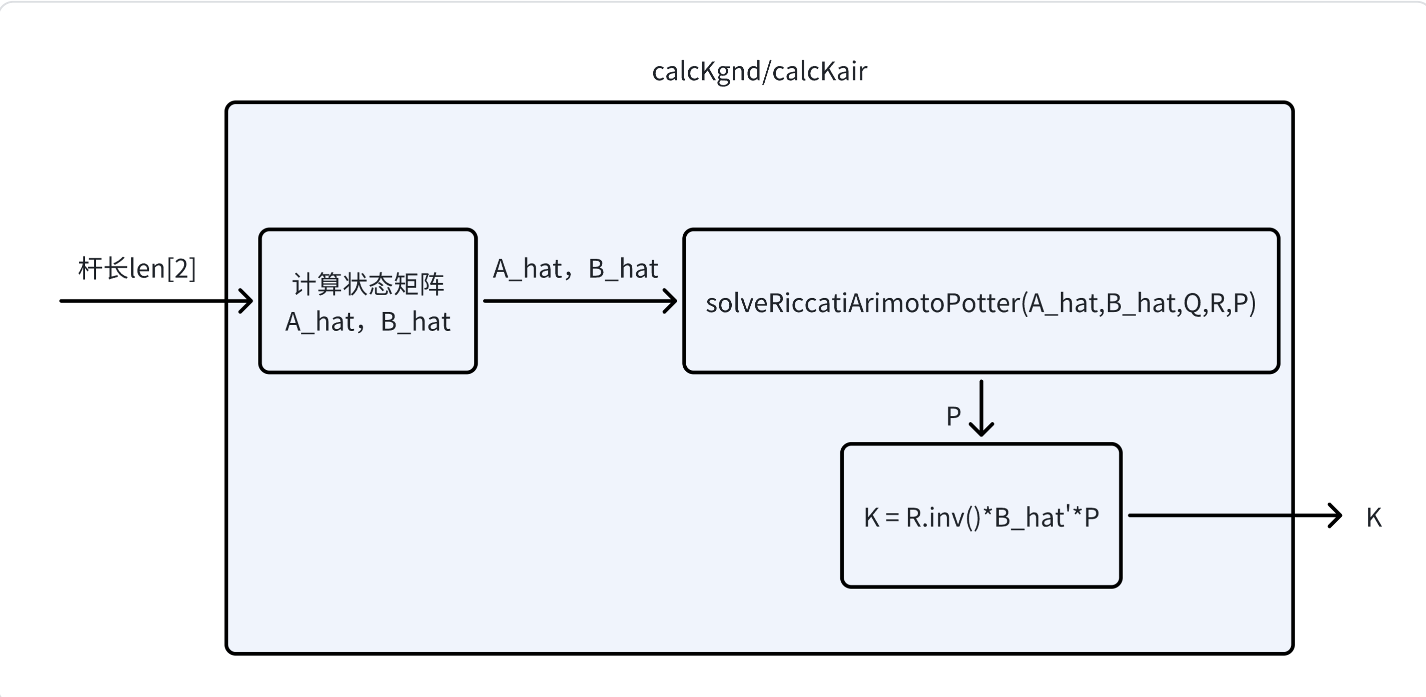
```
void Lqr::calcTau4dof(
        bool sign, const float dt, const float *y, const float *y_dot,
        const float *yd, const float *yd_dot, float *tau)
{
    float x[12];
    x[0] = y[0] - yd[0];
    x[1] = y_dot[0] - yd_dot[0];

    x[3] = y[1] - yd[1];
    x[4] = y[2] - yd[2];
    x[5] = y_dot[1] - yd_dot[1];
    x[6] = y_dot[2] - yd_dot[2];

    x[7] = y[3] - yd[3];
    x[8] = y[4] - yd[4];
    x[9] = y_dot[3] - yd_dot[3];
    x[10] = y_dot[4] - yd_dot[4];

    this->_err_int[0] += x[0] * dt;
    x[2] = this->_err_int[0];
    this->_err_int[1] += (x[4] + x[8]) * dt;
    x[11] = this->_err_int[1];

    bound(x[2], _int_max[0]);
    bound(x[11], _int_max[1]);
    tau[0] = tau[1] = tau[2] = tau[3] = 0;
    if (sign)
    {
        for (uint8_t i = 0; i < U_DIM; i++)
            for (uint8_t j = 0; j < X_DIM_GND; j++)
                tau[i] -= this->k_psc.k_gnd[X_DIM_GND * i + j] * x[j];       You, 4 days ago • move
    }
    else
    {
        for (uint8_t i = 0; i < U_DIM; i++)
            for (uint8_t j = 0; j < X_DIM_AIR; j++)
                tau[i] -= this->k_psc.k_air[X_DIM_AIR * i + j] * x[j];
    }
}
```

引入积分项可消除静态误差，但积分项累计会存在饱和，因此做出限幅来防止积分项累计的误差过大。修改此限幅相关在对应.hpp这里

```cpp
class Lqr
{
public:

  Lqr(){};

  void calcTau4dof(
      bool sign, const float dt, const float *y, const float *y_dot,
      const float *yd, const float *yd_dot, float *tau);

  void reset(void);

  void setKgnd(float *k);

  void setKair(float *k);

private:
  // leg int is pitch and position
  float _err_int[2] = {0.0f, 0.0f};

  float _int_max[2] = {0.10f, 0.00f}; // pitch, wheel

  struct lqr_k k_psc;
};

#endif
```

计算增益K

对应文件为lqr_ksolver.hpp及lqr_ksolver.cpp，计算采用eigen来进行，相关计算如下：



```cpp
void solver::calcKgnd(float *len)
{
    float m_H = robot_param.m_H;
    float I_H = robot_param.I_H;
    float m_w = robot_param.m_w;
    float I_w = robot_param.I_w;
    float m_L = robot_param.m_L;
    float I_L = robot_param.I_L;
    float r = robot_param.r;
    float g = robot_param.g;

    float a[4] = {0}, b[10] = {0};
    float gnd1 = I_w + (m_H + m_L + m_w) * r * r;
    float gnd2 = I_L * gnd1 + I_w * len[1] * len[1] * (m_H + m_L) + len[1] * len[1] * m_w * r * r * (m_H + m_L);
    float gnd3 = I_L * gnd1 + I_w * len[0] * len[0] * (m_H + m_L) + len[0] * len[0] * m_w * r * r * (m_H + m_L);

    a[0] = len[0] * g * (m_H + m_L) * gnd1 / gnd3;
    a[1] = -len[0] * len[0] * g * r * r * (m_H + m_L) * (m_H + m_L) / gnd3;
    a[2] = len[1] * g * (m_H + m_L) * gnd1 / gnd2;
    a[3] = -len[1] * len[1] * g * r * r * (m_H + m_L) * (m_H + m_L) / gnd2;
    b[0] = -1 / I_H;
    b[1] = -1 / I_H;
    b[2] = gnd1 / gnd3;
    b[3] = -(gnd1 + len[0] * r * (m_H + m_L)) / gnd3;
    b[4] = -len[0] * r * r * (m_H + m_L) / gnd3;
    b[5] = r * (I_L + len[0] * (len[0] + r) * (m_H + m_L)) / gnd3;
    b[6] = gnd1 / gnd2;
    b[7] = -(gnd1 + len[1] * r * (m_H + m_L)) / gnd2;
    b[8] = -len[1] * r * r * (m_H + m_L) / gnd2;
    b[9] = r * (I_L + len[1] * (len[1] + r) * (m_H + m_L)) / gnd2;

    Matrix<float, X_DIM_GND, X_DIM_GND> A_hat = Matrix<float, X_DIM_GND, X_DIM_GND>::Zero();
    Matrix<float, X_DIM_GND, U_DIM> B_hat = Matrix<float, X_DIM_GND, U_DIM>::Zero();
    A_hat(0, 1) = A_hat(2, 0) = A_hat(3, 5) = A_hat(4, 6) = A_hat(7, 9) = A_hat(8, 10) = 1;
    A_hat(11, 4) = 2;
    A_hat(5, 3) = A_hat(9, 7) = (a[0] + a[2]) / 2;
    A_hat(5, 7) = A_hat(9, 3) = (a[0] - a[2]) / 2;
    A_hat(6, 3) = A_hat(10, 7) = (a[1] + a[3]) / 2;
    A_hat(6, 7) = A_hat(10, 3) = (a[1] - a[3]) / 2;
    B_hat(1, 0) = b[0];
    B_hat(1, 2) = b[1];

    B_hat(5, 0) = B_hat(9, 0) = b[2] / 2;
    B_hat(5, 1) = B_hat(9, 1) = b[3] / 2;
    B_hat(6, 0) = B_hat(10, 0) = b[4] / 2;       You, 4 weeks ago • change jump and lqr and others
    B_hat(6, 1) = B_hat(10, 1) = b[5] / 2;

    B_hat(5, 2) = b[6] / 2;
```

相关Q，R参数在lqr_ksolver.hpp中的结构体qr_psc_s中，_gnd和_air分别为在地面和空中。

```cpp
struct qr_psc_s
{
// #ifndef DEBUG_IN_SIM
// real param
    const float q_gnd[X_DIM_GND] =
    {2.0e2, 2.0e0, 1.0e0, 1.0e3, 5.0e1, 1.0e1, 5.0e-5, 1.0e4, 1.0e4, 1.0e-1, 1.0e-1, 1.0e-2};
    const float r_gnd[U_DIM] = {1.0, 1.0, 1.0, 1.0};

    const float q_air[X_DIM_AIR] =
    {10.0e2, 5.0e1, 1.0e0, 4.0e1, 1.0e-4, 1.0e1, 1.0e-5, 1.0e3, 1.0e-4, 1.0e1, 1.0e-5, 1.0e-5};
    const float r_air[U_DIM] = {1.0, 1.0, 1.0, 1.0};
```

调试

在MathConst.h文件内定义了几个宏，如下

```c
#define MOTOR_NUM 8
#define Y_STATE_DIM 7

// #define REAL_TIME_K
// #define GENERATE_K_PARAM_LISTS
```

- DEBUG_IN_SIM表示代码是实机代码还是仿真代码，注释掉就表示为实机（主要区别在于低通滤波，仿真不用低通滤波）
- REAL_TIME_K表示实时计算lqr增益K，开启表示启用实时计算增益
- GENERATE_K_PARAM_LISTS表示用来生成的离线K参数的.hpp文件相关宏定义

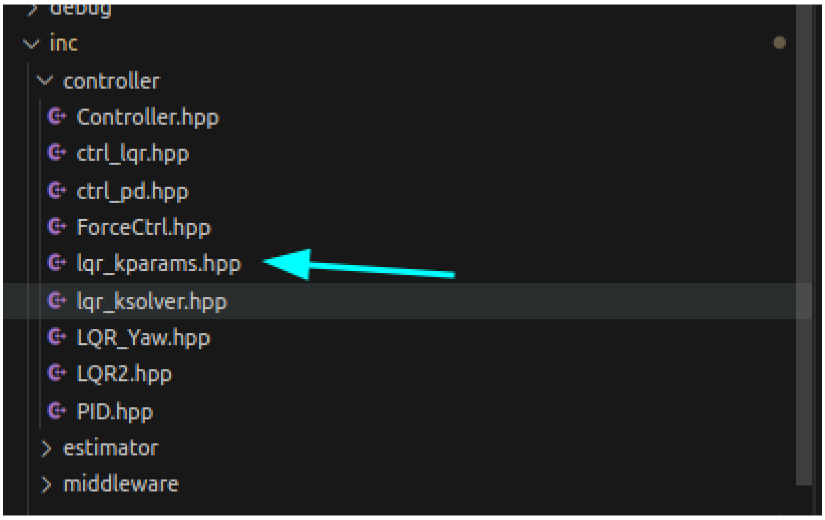
> ✍️ 最新的dev在makefile中添加了宏定义，不用每次注释掉DEBUG_IN_SIM来使用**仿真**和**实机**
>
> 仿真代码，在diablo_webots文件夹内使用 `make debug` 来进行编译
>
> 实机代码，在cerebellum文件夹内用 `west build -b diablo_apollo` 来进行编译

调试过程如下：

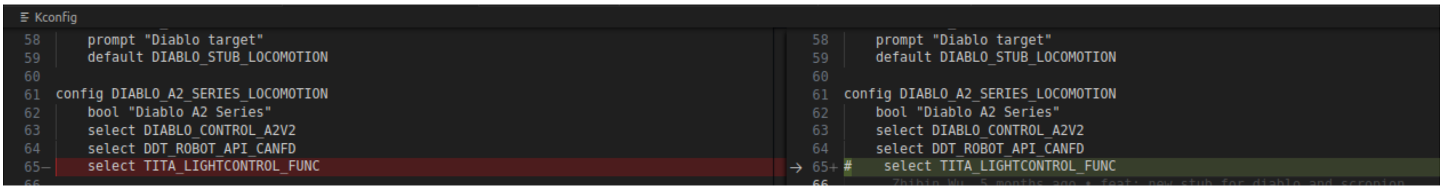
`#define REAL_TIME_K` `// #define GENERATE_K_PARAM_LISTS`	无论仿真或者实机，目前都采用同样的lqr参数，此宏定义来进行lqr的qr参数调节，在ozone中调节使用变量ksolver，调节Q_gnd, R_gnd调节**触地**时的lqr增益，调节Q_air, R_air调节**离地**时的lqr增益，见图1
`// #define REAL_TIME_K` `#define GENERATE_K_PARAM_LISTS`	调好的增益后，修改lqr_ksolver.hpp中的qr_psc_s中的q_gnd，r_gnd, q_air, r_air，保存hpp文件后在/diablo_webots文件夹中make debug后运行 `./diablo_webots` ，会在inc/controller中生成lqr_kparams.hpp文件
`// #define REAL_TIME_K` `// #define GENERATE_K_PARAM_LISTS`	此时无论在./cerebellum文件夹内west编译还是在./diablo_webots文件夹内make debug，跑的代码都是离线算好的增益K

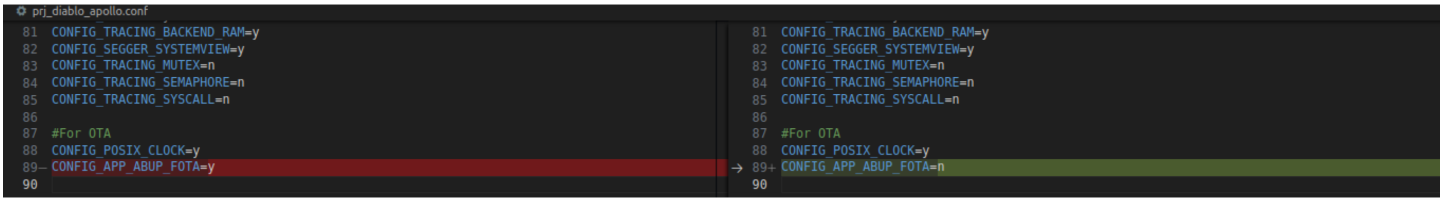
Expression	Location	Refresh	Access
JUMPs	2000 C699	Off	private
⊞ debug_robot	2000 6D08	2 Hz	
⊟ ksolver	2000 8DE0	1 Hz	
f solver::solver(class solver*)		1 Hz	public
f solver::init(class solver*)		1 Hz	public
f solver::refresh(class solver*)		1 Hz	public
f solver::calcKgnd(class solver*, float*)		1 Hz	public
f solver::calcKair(class solver*, float*)		1 Hz	public
f solver::getKgnd(class solver*)		1 Hz	public
f solver::getKair(class solver*)		1 Hz	public
f solver::setMode(class solver*, enum l(		1 Hz	public
f solver::reset(class solver*)		1 Hz	public
f solver::solveRiccatiArimotoPotter(clas		1 Hz	private
⊟ Q_gnd	2000 8DE0	1 Hz	private
[0]	2000 8DE0	1 Hz	
[1]	2000 8DE4	1 Hz	
[2]	2000 8DE8	1 Hz	
[3]	2000 8DEC	1 Hz	
[4]	2000 8DF0	1 Hz	
[5]	2000 8DF4	1 Hz	
[6]	2000 8DF8	1 Hz	
[7]	2000 8DFC	1 Hz	
[8]	2000 8E00	1 Hz	
[9]	2000 8E04	1 Hz	
[10]	2000 8E08	1 Hz	
[11]	2000 8E0C	1 Hz	
⊟ R_gnd	2000 8E10	1 Hz	private
[0]	2000 8E10	1 Hz	
[1]	2000 8E14	1 Hz	
[2]	2000 8E18	1 Hz	
[3]	2000 8E1C	1 Hz	

```
lqr_ksolver.hpp ×
inc > controller > lqr_ksolver.hpp > qr_psc_s > r_gnd
You, 3 days ago | 1 author (You)
struct qr_psc_s
{
// #ifndef DEBUG_IN_SIM
// real param
    const float q_gnd[X_DIM_GND] =
    {2.0e2, 2.0e0, 1.0e0, 1.0e2, 5.0e1, 1.0e-1, 5.0e-5, 1.0e4, 1.0e4, 1.0e-1, 1.0e-1, 1.0e-2};
    const float r_gnd[U_DIM] = {1.0, 1.0, 1.0, 1.0};     You, 4 days ago • sim rm sim_to_real

    const float q_air[X_DIM_AIR] =
    {10.0e2, 5.0e1, 1.0e0, 4.0e1, 1.0e-4, 1.0e1, 1.0e-5, 1.0e3, 1.0e-4, 1.0e1, 1.0e-5, 1.0e-5};
    const float r_air[U_DIM] = {1.0, 1.0, 1.0, 1.0};
```



注：由于计算增益K时占用Flash空间过大，编译后程序基本占用flash大小为95%左右，可以在调试时通过关闭OTA和灯板控制相关代码来降低Flash占用，在./cerebeleum文件夹中的Kconfig和prj_diablo_apollo.conf中，如下：





其次，如果west build时不关DEBUG_IN_SIM或者GENERATE_K_PARAM_LISTS都有可能报超出flash大小的问题