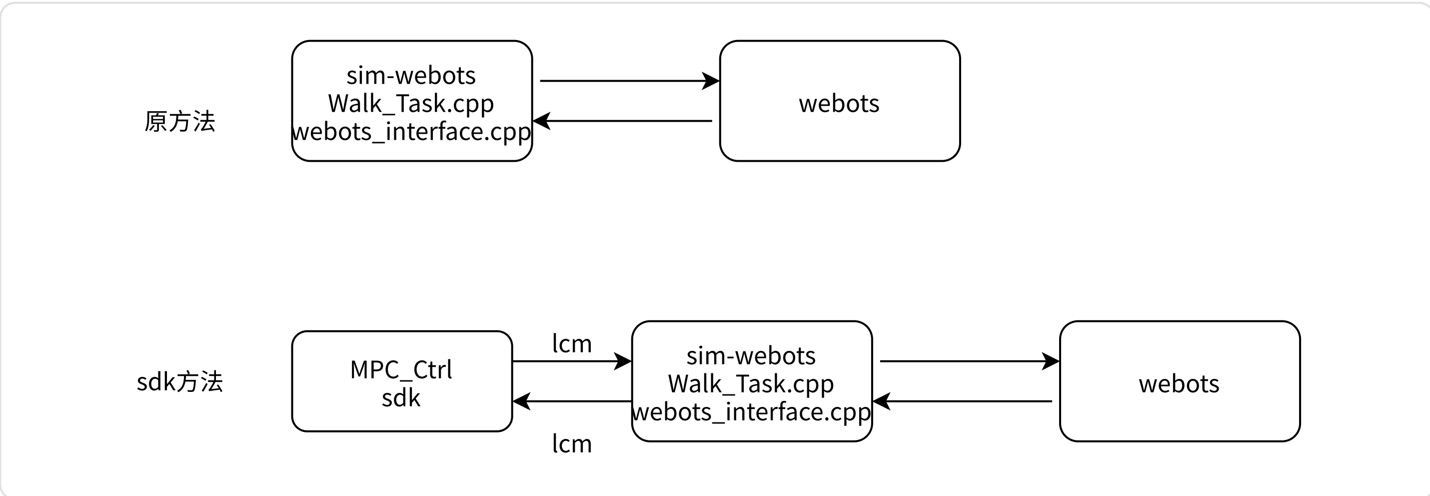
# hxt--基于cheetah-software的“半只狗”控制sdk

## 0.思路：把A2简化成半只狗，将cheetah-software的四条腿按两条腿处理。

## 1.用lcm作为状态和输出传递的sdk方法

### 1.0 方法结构比较




### 1.1 sim-webots的处理

**A2_walking**分支：CAR状态表示伸腿，使机器人在几个solid的帮助下站立，3s之后进入TRANSFORM_UP，这时开始执行基于cassie的运控，在此基础上，做如下处理。

如下列图片所示，在更新完机器人状态之后，将机器人的状态通过lcm发送，MPC_Ctrl中会有相应的订阅函数

```
float out_right[3] = {0.f,0.f,0.f};
void WB6_Task::Walk_Plan_Task(const float dt)
{
        // constexpr std::chrono::nanoseconds dT = 10ms;
        // std::chrono::nanoseconds stamp = std::chrono::nanos
        robot.Robot_Update(dt);
        // legL_ptr._q1(_q2/_q3)
        // legR_ptr
        // posture_ptr._quaternion.w(x/y/z)
        ctrl_data ctrl_data_in;
        ctrl_data_in.quat[0] = robot.posture_ptr->_quaternion.
        ctrl_data_in.quat[1] = robot.posture_ptr->_quaternion.
        ctrl_data_in.quat[2] = robot.posture_ptr->_quaternion.
        ctrl_data_in.quat[3] = robot.posture_ptr->_quaternion.
        for(int i = 0; i < 3; i++) {
            ctrl_data_in.gyro[i] = robot.posture_ptr->rotation
            ctrl_data_in.acc[i] = robot.posture_ptr->accl[i];
        }
        ctrl_data_in.motor_pos[0] = robot.legL_ptr->_q1;
        ctrl_data_in.motor_pos[1] = robot.legL_ptr->_q2;
        ctrl_data_in.motor_pos[2] = robot.legL_ptr->_q3;
        ctrl_data_in.motor_pos[3] = robot.legR_ptr->_q1;
        ctrl_data_in.motor_pos[4] = robot.legR_ptr->_q2;
        ctrl_data_in.motor_pos[5] = robot.legR_ptr->_q3;
        ctrl_data_in.motor_vel[0] = robot.legL_ptr->_dq1;
        ctrl_data_in.motor_vel[1] = robot.legL_ptr->_dq2;
        ctrl_data_in.motor_vel[2] = robot.legL_ptr->_dq3;
        ctrl_data_in.motor_vel[3] = robot.legR_ptr->_dq1;
        ctrl_data_in.motor_vel[4] = robot.legR_ptr->_dq2;
        ctrl_data_in.motor_vel[5] = robot.legR_ptr->_dq3;
        lcm_pub.publish("control_data",&ctrl_data_in);
        if (!initHandle.started)
        {
```

如下列图片所示，在TRANSFORM_UP状态，用接收来自MPC_Ctrl的output取代原来的output，这个写法比较不正式，但是快，正式的写法应该是再建一个状态，比如RobotClass::LCM_MPC

```
}

void WB6_Task::handleMessage(const lcm::ReceiveBuffer *rbuf, const std::string &chan,
                             const ctrl_cmd *msg)
{
    memcpy(&ctrl_cmd_out, msg, sizeof(ctrl_cmd));
    static int count = 0;
    if(count++ % 100 == 0) {
        printf("ctrl_cmd_out_l: %f,%f,%f\n",
               ctrl_cmd_out.swing_out[0],
               ctrl_cmd_out.tilt_out[0],
               ctrl_cmd_out.knee_out[0]);
        printf("ctrl_cmd_out_r: %f,%f,%f\n",
               ctrl_cmd_out.swing_out[1],
               ctrl_cmd_out.tilt_out[1],
               ctrl_cmd_out.knee_out[1]);
    }
}
```

```
    break;
case Robot_Class::TRANSFORM_UP:

    ctrl.walk_ctrl.Ctrl_Update(dt,false);
    ctrl.left_swing_out  = ctrl_cmd_out.swing_out[0];//ctrl.walk_ctrl.u_out[0];
    ctrl.left_tilt_out   = ctrl_cmd_out.tilt_out[0];
    ctrl.left_knee_out   =  ctrl_cmd WB6_Task::ctrl_cmd_out
    ctrl.right_swing_out = ctrl_cmd_out.swing_out[1];
    ctrl.right_tilt_out  = ctrl_cmd_out.tilt_out[1];
    ctrl.right_knee_out  = -ctrl_cmd_out.knee_out[1];
```

经过上述处理得到了分支**A2_walking_sdk**

## 2.MPC_Ctrl的软件结构

2.1 common文件夹

AttitudeData.h：更新和存储来自lcm的四元数，局部加速度和局部角速度信息，处理得到旋转矩阵等信息

LegData.h：更新和存储来自lcm的腿部六个关节的位置和速度，处理得到足端相对于abad关节的局部笛卡尔坐 标系等信息

orienation_tools.h：Cheetah-software定义的旋转矩阵和四元数处理函数

其他头文件：定义一些数据类型之类的

2.2 Estimator文件夹

此文件包含Kalman观测器的实现

StateEstimatorContainer.h: 定义父类GenericEstimator，public变量_stateEstimatorData变量用于传递结果

PositionVelocityEstimator.h：子类LinearKFPositionVelocityEstimator

与一个典型kalman filter对应，LinearKFPositionVelocityEstimator的成员变量包括：

输入　中间变量　常量　输出

x: 状态向量
P: 协方差

预测

$$x_{k\_pred} = Ax_{k-1} \cancel{+ Bu_k}$$
$$P_{k\_pred} = AP_{k-1}A^T + Q$$

A: 转移矩阵
Q: 过程噪声

更新

$$\tilde{y} = z_k - Hx_{k\_pred}$$
$$S = HP_{k\_pred}H^T + R$$
$$K = P_{k\_pred}H^T S^{-1}$$
$$x_k = x_{predicted} + K\tilde{y}$$
$$P_k = (I - KH)P_{k\_pred}$$

z: 观测向量
H: 转换矩阵
y: 新息（残差）
S: 新息协方差
R: 观测噪声
K: 卡尔曼增益



Eigen::Matrix<T, 12, 1> _xhat：状态向量：质心在世界坐标系下的位置(3X1),速度(3X1),左右褪**足端**在世界坐标系下位置(6X1)

Eigen::Matrix<T, 12, 12> _A：模型转移矩阵

Eigen::Matrix<T, 12, 12> _Q0：过程噪声

Eigen::Matrix<T, 12, 12> _P：协方差矩阵

Eigen::Matrix<T, 14, 14> _R0：测量噪声

Eigen::Matrix<T, 12, 3> _B：输入矩阵，输入为3X1的全局加速度

Eigen::Matrix<T, 14, 12> _C：观测矩阵, 将状态向量映射到测量向量

测量向量：14X1，(左腿右腿**足端**在**全局body坐标系**下的位置（6X1)，左腿右腿**足端**在**全局body坐标系**下的局部速度(6X1)，左腿足端和右腿足端的离地高度(2X1))

**全局body**坐标系(容易混淆)：姿态与世界坐标系重合，位置与局部坐标系重合，或者说世界坐标系平移到body的com

PositionVelocityEstimator.cpp

A_,B_,C_的构成细节以后再补充，**基本原理是用支撑腿的局部足端速度补偿加速度计的积分得到的速度**

$$\hat{x} = \begin{bmatrix} x \\ y \\ z \\ vx \\ vy \\ vz \\ x_{leftfoot} \\ y_{leftfoot} \\ z_{leftfoot} \\ x_{rightfoot} \\ y_{rightfoot} \\ z_{rightfoot} \end{bmatrix}$$

$$A = \begin{bmatrix} I_3 & dt * I_3 & 0 & 0 \\ 0 & I_3 & 0 & 0 \\ 0 & 0 & I_3 & 0 \\ 0 & 0 & 0 & I_3 \end{bmatrix}$$

$$B = \begin{bmatrix} 0_3 \\ dt * I_3 \\ 0_3 \\ 0_3 \end{bmatrix}$$

$$C = \begin{bmatrix} I_3 & 0_3 & -I_3 & 0_3 \\ I_3 & 0_3 & 0_3 & -I_3 \\ 0_3 & I_3 & 0_3 & 0_3 \\ 0_3 & I_3 & 0_3 & 0_3 \\ 0,0,0 & 0,0,0 & 0,0,1 & 0,0,0 \\ 0,0,0 & 0,0,0 & 0,0,0 & 0,0,1 \end{bmatrix}$$

$$y = \begin{bmatrix} x_{leftfoot} \\ y_{leftfoot} \\ z_{leftfoot} \\ x_{rightfoot} \\ y_{rightfoot} \\ z_{rightfoot} \\ vx_{leftfoot} \\ vy_{leftfoot} \\ vz_{leftfoot} \\ vx_{rightfoot} \\ vy_{rightfoot} \\ vz_{rightfoot} \\ h_{leftfoot} \\ h_{rightfoot} \end{bmatrix}$$

状态空间的迭代(比较好理解)

$\hat{x} = A\hat{x} + B * [a_w - g]$

状态向量映射到测量向量的原理

原理1：足端在全局body坐标系下的位置=世界坐标系下的com位置 - 世界坐标下的足端位置

原理2：**支撑**足端在全局body坐标系的速度 = 世界坐标下的com速度(摆动足的影响会由Q和R矩阵抵消,但这里遵照支撑腿的关系形式)

$\hat{y} = C * \hat{x}$

K增益矩阵由Q，R，P矩阵决定，一个宏观的理解，Q矩阵越大，说明模型粗糙，更信任测量值，R矩阵越大，说明测量噪声大，则更信任模型得到的值。

$\hat{x} = \hat{x} + K(y - C\hat{x})$：对于x中的速度项估计，K矩阵会增加支撑足段速度的影响，削弱摆动足足端速度和加速度计积分的影响

2.3 convexMPC

2.3.0 该类主要包含基于slip模型的落足点规划和convexMPC控制，其中每个dt(仿真5ms)，更新一次落足点，

每6个dt称为每个MPC周期，每个MPC周期会进行一次MPC优化，得到下一MPC周期的支撑腿支撑力。(反直觉的地方：MPC不是每次dt都执行，因为要兼容真实硬件的算力，采取几次dt执行一次的方案)

2.3.1 一些重要的成员变量

ConvexMPCLocomotion.h

世界坐标下身体质心的期望位置，期望速度以及期望加速度(WBC需要)

```
Vec3<float> pBody_des;
Vec3<float> vBody_des;
Vec3<float> aBody_des;
```

世界坐标系下身体的期望姿态转角以及转角速度(WBC需要)

```
Vec3<float> pBody_RPY_des;
Vec3<float> vBody_Ori_des;
```

世界坐标系下足端的位置以及足端坐标系

```
Vec3<float> pFoot_des[2];
Vec3<float> vFoot_des[2];
Vec3<float> aFoot_des[2];
```

生成contact_schedule的gait类，给定当前时刻，决定褪的支撑与摆动

```
OffsetDurationGait trotting:
```

摆动腿足端轨迹规划类，每个腿摆动开始时，给定足端起始位置，和期望步高。

摆动开始以及支撑之前，每个时刻根据根据slip模型更新足端期望结束位置，之后输出当前时刻的期望轨迹(反直觉的地方：每个时刻期望结束位置是更新的，因为要控制速度，增加抗扰动)

FootSwingTrajectory<float> footSwingTrajectories[4];

以下为世界坐标系下身体质心的期望位置

Vec3<float> world_position_desired；//当前位置+dt*当前速度

float trajAll[12 * 36];//12表示期望状态(12X1),期望姿态(3X1)，期望位置(3X1)，期望角速度(3X1)，期望速度(3X1)

//36表示积分步长，一般不会用那么多，只用前10个，每个期望速度维持不变，期望位置由上一个位置+期望速度*dtMPC得到

void ConvexMPCLocomotion::updateMPCIfNeeded(int *mpcTable, bool omniMode, AttitudeData* attitude_, LinearKFPositionVelocityEstimator<float>* posvelest_) {

//MPC的期望输入：Gait类产生的mpctable(接下来各个褪的摆动和支撑情况，接下来10个时刻的期望状态trajAll

//MPC的测量输入：当前观测器传来的实际位置，实际速度，实际角速度，实际速度，实际足端到质心的位置矢量

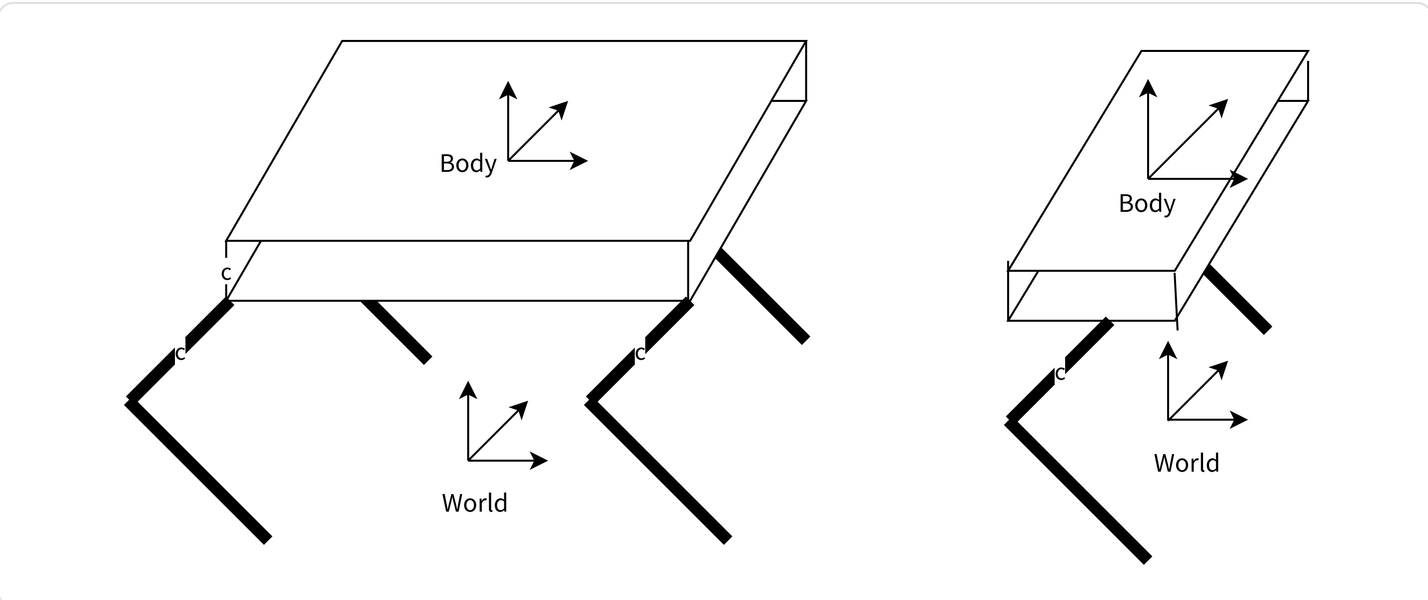
//MPC的权重:Q[12] = {0.5, 0.25, 0, 0, 0, 100, 0.0, 0.0, 0.0, 0.2, 0.2, 0.1};

//有了上述期望输入和测量输入，构造MPC问题，优化求解得到期望支撑力，作为控制输入(如何构造mpc问题，由2.3.2叙述)

}

2.3.2结合MIT硕士论文Software and Control Design for the MIT Cheetah

Quadruped Robots描述一下mpc问题的构造



Carlo_Jared_2020_Software and control design for the MIT Cheet⋯

25.55MB

下列状态方程的量均在世界坐标系下：

$$\frac{d}{dt}\begin{bmatrix}\theta\\ p\\ \omega\\ \dot{p}\end{bmatrix}=\begin{bmatrix}0_3 & 0_3 & R_z(\psi) & 0_3\\ 0_3 & 0_3 & 0_3 & I_3\\ 0_3 & 0_3 & 0_3 & 0_3\\ 0_3 & 0_3 & 0_3 & 0_3\end{bmatrix}\begin{bmatrix}\theta\\ p\\ \omega\\ \dot{p}\end{bmatrix}+\begin{bmatrix}0_3 & \dots & 0_3\\ 0_3 & \dots & 0_3\\ I^{-1}[r_1]_\times & \dots & I^{-1}[r_n]_\times\\ 1_3/m & \dots & 1_3/m\end{bmatrix}\begin{bmatrix}f_1\\ \dots\\ f_n\end{bmatrix}+\begin{bmatrix}0\\ 0\\ 0\\ g\end{bmatrix}$$

欧拉角的时间导数与角速度的近似关系：

$$\frac{d}{dt}\theta=\frac{d}{dt}\begin{bmatrix}roll\\ pitch\\ yaw\end{bmatrix}\approx R_z(yaw)\omega_w$$

角加速度与支撑腿到com的转矩的关系：

$$\frac{\mathrm{d}}{\mathrm{dt}}(I_w\omega)\approx I_w\frac{d}{dt}\omega=\sum_{i=1}^{n}r_i\times f_i$$

加速度与支撑腿受力的关系

$$\ddot{p}=\sum_{i=1}^{n}f_i/m+g$$

将上述状态方程标准化

$$\frac{d}{dt}\begin{bmatrix}\theta\\ p\\ \omega\\ \dot{p}\\ -9.8\end{bmatrix}=\begin{bmatrix}0_3 & 0_3 & R_z(\psi) & 0_3 & \begin{bmatrix}0\\0\\0\end{bmatrix}\\ 0_3 & 0_3 & 0_3 & I_3 & \begin{bmatrix}0\\0\\0\end{bmatrix}\\ 0_3 & 0_3 & 0_3 & 0_3 & \begin{bmatrix}0\\0\\0\end{bmatrix}\\ 0_3 & 0_3 & 0_3 & 0_3 & \begin{bmatrix}0\\0\\1\end{bmatrix}\\ [0,0,0] & [0,0,0] & [0,0,0] & [0,0,0] & 0\end{bmatrix}\begin{bmatrix}\theta\\ p\\ \omega\\ \dot{p}\\ -9.8\end{bmatrix}+$$

$$\begin{bmatrix}0_3 & \dots & 0_3\\ 0_3 & \dots & 0_3\\ I^{-1}[r_1]_\times & \dots & I^{-1}[r_n]_\times\\ 1_3/m & \dots & 1_3/m\\ [0,0,0] & \dots & [0,0,0]\end{bmatrix}\begin{bmatrix}f_1\\ \dots\\ f_n\end{bmatrix}$$

$$\frac{d}{dt}X=A_cX+B_cu\Longrightarrow\frac{d}{dt}\begin{bmatrix}X\\ u\end{bmatrix}=\begin{bmatrix}A_c & B_c\\ 0 & 0\end{bmatrix}\begin{bmatrix}X\\ u\end{bmatrix}=A_b\begin{bmatrix}X\\ u\end{bmatrix}$$

$$\frac{d}{dt}\begin{bmatrix}X\\ u\end{bmatrix}=A_b\begin{bmatrix}X\\ u\end{bmatrix}\Longrightarrow\begin{bmatrix}X(t_0+T)\\ u(t_0+T)\end{bmatrix}=e^{A_bT}\begin{bmatrix}X(t_0)\\ u(t_0)\end{bmatrix}\Longrightarrow\begin{bmatrix}X(n+1)\\ u(n+1)\end{bmatrix}=e^{A_bT}\begin{bmatrix}X(n)\\ u(n)\end{bmatrix}$$

$$\begin{bmatrix} X(n+1) \\ u(n+1) \end{bmatrix} = e^{A_bT} \begin{bmatrix} X(n) \\ u(n) \end{bmatrix} \rightarrow \begin{bmatrix} X(n+1) \\ u(n+1) \end{bmatrix} = \begin{bmatrix} e_{11}^{A_bT} & e_{12}^{A_bT} \\ e_{21}^{A_bT} & e_{22}^{A_bT} \end{bmatrix} \begin{bmatrix} X(n) \\ u(n) \end{bmatrix}$$

经过上述推导，我们得到了如下的离散状态转换方程，上述过程在SolverMPC.cpp中有代码级的实现

$$X(n) = \begin{bmatrix} \theta(n) \\ p(n) \\ \omega(n) \\ \dot{p}(n) \\ -9.8 \end{bmatrix} u(n) = \begin{bmatrix} f_1(n) \\ f_2(n) \end{bmatrix}$$

$$X(n+1) = e_{11}^{A_bT} X(n) + e_{12}^{A_bT} u(n)$$

$$X(n+1) = \hat{A}X(n) + \hat{B}u(n)$$

如果MPC仅有两步预测，则为如下形式

$$\begin{bmatrix} X(1) \\ X(2) \end{bmatrix} = \begin{bmatrix} A \\ AA \end{bmatrix} X(0) + \begin{bmatrix} B & 0 \\ AB & B \end{bmatrix} \begin{bmatrix} u(0) \\ u(1) \end{bmatrix}$$

如果MPC有n步，则为如下形式

$$\begin{bmatrix} X(1) \\ X(2) \\ \dots \\ X(n) \end{bmatrix} = \begin{bmatrix} A \\ AA \\ \dots \\ A^n \end{bmatrix} X(0) + \begin{bmatrix} B & 0 & \dots & 0 \\ AB & B & \dots & 0 \\ \dots & \dots & \dots & \dots \\ A^{n-1}B & A^{n-2}B & \dots & B \end{bmatrix} \begin{bmatrix} u(0) \\ u(1) \\ \dots \\ u(n-1) \end{bmatrix}$$

至此我们得到n步mpc的状态预测方程

$$\begin{bmatrix} X(1) \\ X(2) \\ \dots \\ X(n) \end{bmatrix} = A_{qp}X(0) + B_{qp} \begin{bmatrix} u(0) \\ u(1) \\ \dots \\ u(n-1) \end{bmatrix}$$

$$x = \begin{bmatrix} X(1) \\ \dots \\ X(n) \end{bmatrix}, x_{ref} = \begin{bmatrix} X_{ref}(1) \\ \dots \\ X_{ref}(n) \end{bmatrix}, U = \begin{bmatrix} u(0) \\ \dots \\ u(n-1) \end{bmatrix}$$

优化问题的代价方程(cost function)描述如下：

$$(A_{qp}X(0) + B_{qp}U - x_{ref})^T L(A_{qp}X(0) + B_{qp}U - x_{ref}) + U^T KU$$

第一项表示达到期望，第二项表示代价最小

L 矩阵：控制状态误差的权重矩阵，影响状态 X 偏离参考状态 $x_{ref}$ 的惩罚力度。L 决定了预测控制中状态误差的惩罚力度。通常来说，选择 L会影响预测控制中对误差最小化的优先级，较大的 L 元素会对状态误差施加更大的惩罚，从而影响优化的结果。

K 矩阵：控制输入 U 的权重矩阵，影响控制输入大小的惩罚，通常用于限制控制能量或抑制控制信号过大。K 矩阵控制的是对控制输入大小的惩罚。较大的 K 元素会抑制控制输入的大小，从而实现控制能量或控制力度的约束。这种设置通常用于防止系统过大的控制输入。

标准的二次规划形式

$$\begin{array}{ll} \underset{U}{min} & \frac{1}{2}U^THU+U^Tg \\ s.t. & c_l \le CU \le c_u \end{array}$$

在这里我们先对目标函数展开

$$(A_{qp}X(0)+B_{qp}U-x_{ref})^TL(A_{qp}X(0)+B_{qp}U-x_{ref})$$

展开这个二次型：

$$= \ (A_{qp}X(0))^TL(A_{qp}X(0))+(B_{qp}U)^TLB_{qp}U-2x_{ref}^TL(A_{qp}X(0)+B_{qp}U)$$

这三项中，第一项与U无关，可以忽略。第二项和第三项与U有关，需要继续处理。

第二项，

$$(B_{qp}U)^TLB_{qp}U \ = \ B_{qp}^TU^TLB_{qp}U$$

第三项，

$$-2x_{ref}^TL(A_{qp}X(0)+B_{qp}U) \ = \ -2(x_{ref}^TLA_{qp}X(0)+x_{ref}^TLB_{qp}U)$$

因此，整个目标函数可以写为：

$$\underset{U}{min}\ U^TB_{qp}^TLB_{qp}U+2U^TB_{qp}^TL(A_{qp}X(0)-x_{ref})+U^TKU$$

由此，我们算得H和g

二次项 $U^THU$ :

$$H=2(B_{qp}^TLB_{qp}+K)$$

线性项 $U^Tg$ :

$$g=2B_{qp}^TL(A_{qp}X(0)-x_{ref})$$

2.4 腿部笛卡尔空间控制与lcm的接收和发送