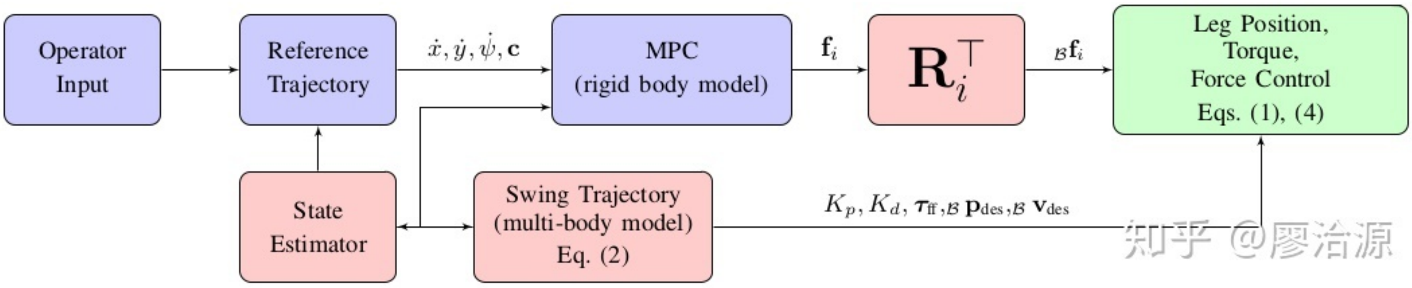
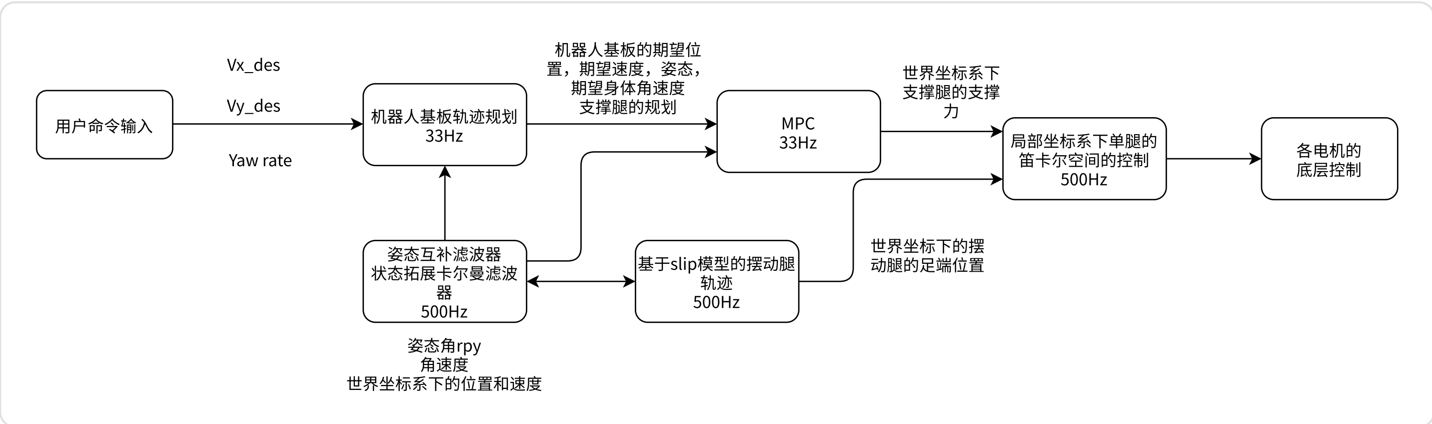
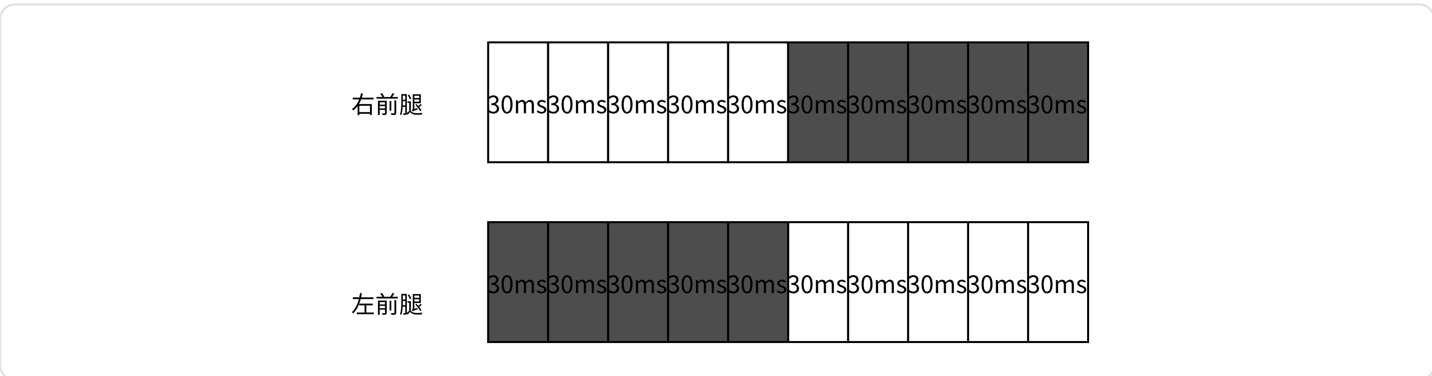
hxt--入门MPC

0. MInicheetah的MPC控制架构

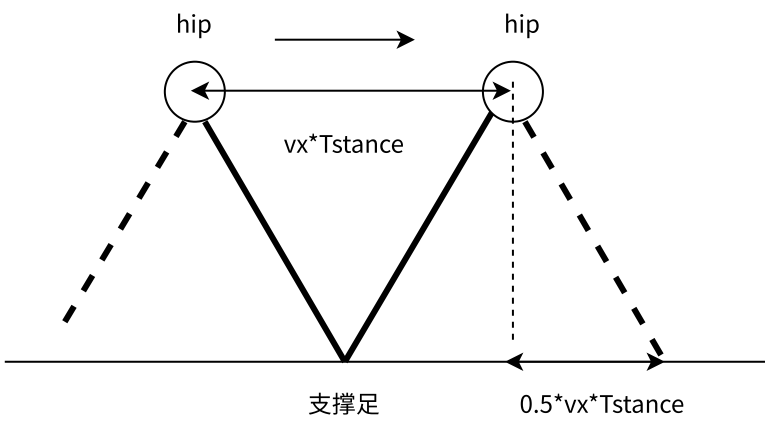




0.1支撑与摆动的规划



0.2摆动腿的步态规划

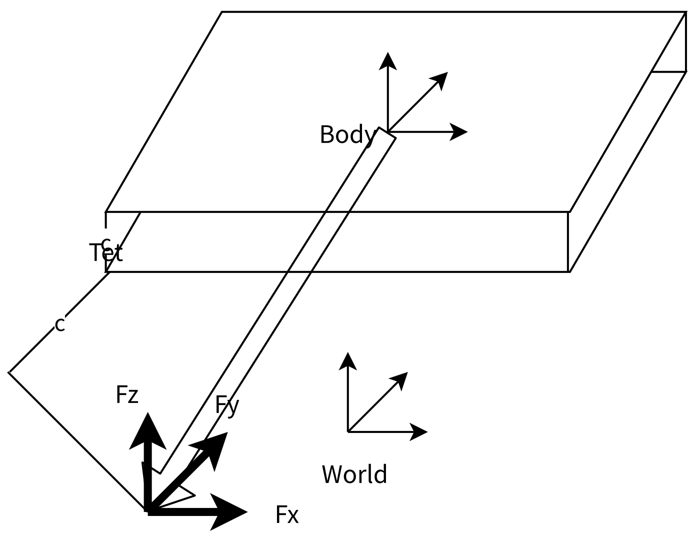


$$x_{sw} = x_{hip} + 0.5 * v_x * t_{stance} + k(v_x - v_{dx}) + 0.5 * h/g * v_y * w_{dz}$$

0.3状态观测器或者足式里程计

　mpc均需要在世界坐标下进行规划和控制，需要EKF的方法用来得到机器人质心在世界坐标系下的位置和速度，继而得到足端在世界坐标系下的位置和速度

1.建模



欧拉角的时间导数与角速度的近似关系：

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

注释：

θ :Body的欧拉角

$R_z(yaw)$:旋转矩阵

ω_w :世界坐标系下的角速度

角加速度与支撑腿到com的转矩的关系：

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

注释:

I_w :base转动惯量

r_i :世界坐标系下质心到足端的位置矢量

f_i :世界坐标系下末端受力

加速度与支撑腿受力的关系

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

2.预测

2.1最简单的单步预测

$$\frac{d}{dt}X = A_cX + B_cu$$

可以通过**离散化**的方法转化为一个离散时间方程。常见的离散化方法包括**前向欧拉法（Forward Euler method）**。对于离散化间隔 T 足够小的情况，我们可以使用线性近似的公式来进行预测：

$$X(t_0+T) \approx (A_cT+I)X(t_0) + B_cTu$$

这个公式的推导基于前向欧拉法的近似离散化方法。这里 A_cT+I 是对系统矩阵 A_c 的离散化近似，而不是 A_c 本身，我们使用前向欧拉法来近似状态变化时，可以认为当时间间隔 T 足够小时，系统的状态变化可以被线性地逼近。因此， A_cT+I 是对指数矩阵 e^{A_cT} 的一个近似。它在 T 足够小时有效，因为系统的状态不会在极小的时间间隔内发生剧烈变化。

单位矩阵 I 表示状态系统在当前时刻的贡献。

A_cT+I 是状态矩阵 A_c 的离散化，乘以时间间隔 T来表示从 t_0 到 t_0+T 的状态变化。

如果不做近似，精确的离散化状态更新方程应该是：

$$X(t0+T) = e^{A_cT}X(t_0) + \int_0^T e^{A_c(T-\tau)}B_cu(\tau)d\tau$$

这里， e^{A_cT} 是矩阵的指数运算。

在T非常小时，我们可以用泰勒展开来近似 e^{A_cT} ：

$$e^{A_cT} \approx A_cT + I$$

期望

$$X_{des} = (A_cT+I)X + B_cTu$$

基于单步预测的控制a

$$\min_{u} ||X_{des} - (A_cT + I)X(t_0) - B_cTu||$$

$$u = (B_cT)^{-1}(X_{des} - (A_ct + I)X)$$

2.2多步预测的情况

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

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

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

基于多步预测的控制

$$\min_{U} ||A_{qp}X(0) + B_{qp}U - x_{ref}||_L + ||U||_K$$

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

2.3 一些假设和细节

状态方程A与输入方程B均要做线性化处理，并且默认在预测步长内(10个MPC周期)不变化

对于多步预测的情况，所得到U，其实包含预测步长n步的所有控制输入，但是实际控制只用u(0)，其他u(1)~u(n-1)**算出来但不用**。下一个MPC周期，更新当前的A，B，xref，结算得到U，依然只用u(0)，输入到系统。

3.控制的解算

3.1 c++中描述矩阵和矢量运算的库eigen

```
 Eigen::Matrix<float, 3, 1> X;
 X << 0,0,0;
 Eigen::Matrix<float, 3, 3> A;
 A << 1,0,0,0,1,0,0,0,1;
 A.setIdentity();
X=A*X;
X=X.cross(X);
A=A.inverse();
float x = X.transpose()*A*X;
```

3.2二次规划问题的求解

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

改写成标准的二次规划问题

$$\begin{aligned}
&\min_{U} \quad \tfrac{1}{2}U^T HU+U^T g\\
&s.t. \quad c_l \le CU \le c_u\\
&H=2(B^T LB+K)\\
&g=2B^T L(AX(0)-x_{ref})
\end{aligned}$$

qpOASES库

```
        qpOASES::QProblem problem_red (new_vars, new_cons);
        qpOASES::Options op;
        op.setToMPC();
        op.printLevel = qpOASES::PL_NONE;
        problem_red.setOptions(op);
        //int_t nWSR = 50000;

        int rval = problem_red.init(H_red, g_red, A_red, NULL, NULL, lb_red, ub_red, nWSR);
        (void)rval;
        int rval2 = problem_red.getPrimalSolution(q_red);
        if(rval2 != qpOASES::SUCCESSFUL_RETURN)
          printf("failed to solve!\n");
```