

# 目录

- 项目背景与项目内容
  - 项目背景
  - 需求分析
  - 项目内容
- 轮腿机器人相关说明
  - 整机图片
  - 正方向定义
  - 底盘结构体从属关系
  - 控制方式说明
- 项目系统与功能设计
  - 硬件设计
  - 软件设计
    - 系统架构
    - 运行流程
    - 底盘任务流程
    - 离线检测任务流程
  - 功能设计
- 项目关键技术与实现
  - 机器人运动和控制
    - 系统建模
      - 模型定义
      - 经典力学分析
      - 状态空间模型
    - 控制器设计
      - LQR
      - VMC
    - 控制逻辑及代码

# 项目背景与项目内容

## 项目背景

随着科技的不断发展，机器人技术在各个领域得到了广泛应用。轮腿机器人兼具轮式结构和腿部能力，轮式结构移动快、效率高；腿部能力让机器人适应不平地面、完成跳跃台阶等动作，适用于各种复杂环境的运动与操控。在多个领域，包括工业、服务、军事等，轮足机器人展现了强大的潜力。

在影视制作、视频拍摄等领域，机器人搭载云台进行拍摄任务已成为一种趋势。这种机器人搭载云台拍摄的方式为拍摄提供了更广阔的视角和更灵活的运动轨迹，但是目前市场上仍存在一些痛点，例如传统摄影设备进行预设轨迹拍摄需要预先进行轨道铺设、轨迹不易更改、摄影机自由度较少的问题，而且往往需要多人操作。而使用无人机等飞行设备进行拍摄在近地面时容易受到气流干扰，并且容易产生碰撞，还需考虑飞行限制和法律法规。本项目希望用轮腿机器人给拍摄过程带来机动性和适应性。

## 需求分析

针对项目背景和对背景中痛点的分析，总结出以下几点需求：

**移动功能**：为满足拍摄需求，机器人需满足能够前往场地中的任意地点，并且能够保证拍摄设备的搭载平台在移动过程中能够保持平稳。

**拍摄角度调整功能**：机器人三轴方向都需要有自由度，才能让拍摄设备朝向各个方向，并且能利用五连杆腿部来升降拍摄设备。

**地形适应功能**：机器人需要在各种不平坦路面上稳定自己，能够完成从高处跌落并保持稳定，在高速转向时能够有相关处理。

**易操作性**：希望拍摄过程不需要过多人力，并且操作简单。便于携带，能检测机器人状态，稳定启动。

**抗干扰能力**：在遇到障碍发生碰撞时能保持稳定，遭遇一定程度撞击能够快速恢复。

**跳跃能力**：在机器人移动路径中遇到台阶等单独依靠轮子无法跨越的障碍时，能够通过跳跃进行越障。

## 项目内容

针对以上需求，将需求总结为以下技术点：

**移动**：完成机器人前后、旋转运动，并且在机器人运动时机体保持Pitch轴稳定。

**自由度**：Pitch、Yaw、Roll三轴自由度调节，并配合腿部控制完成Z轴升降。

**适应性**：复杂路面上的自适应，能完成Roll轴自稳，下台阶冲击抵抗，转向重心偏移。

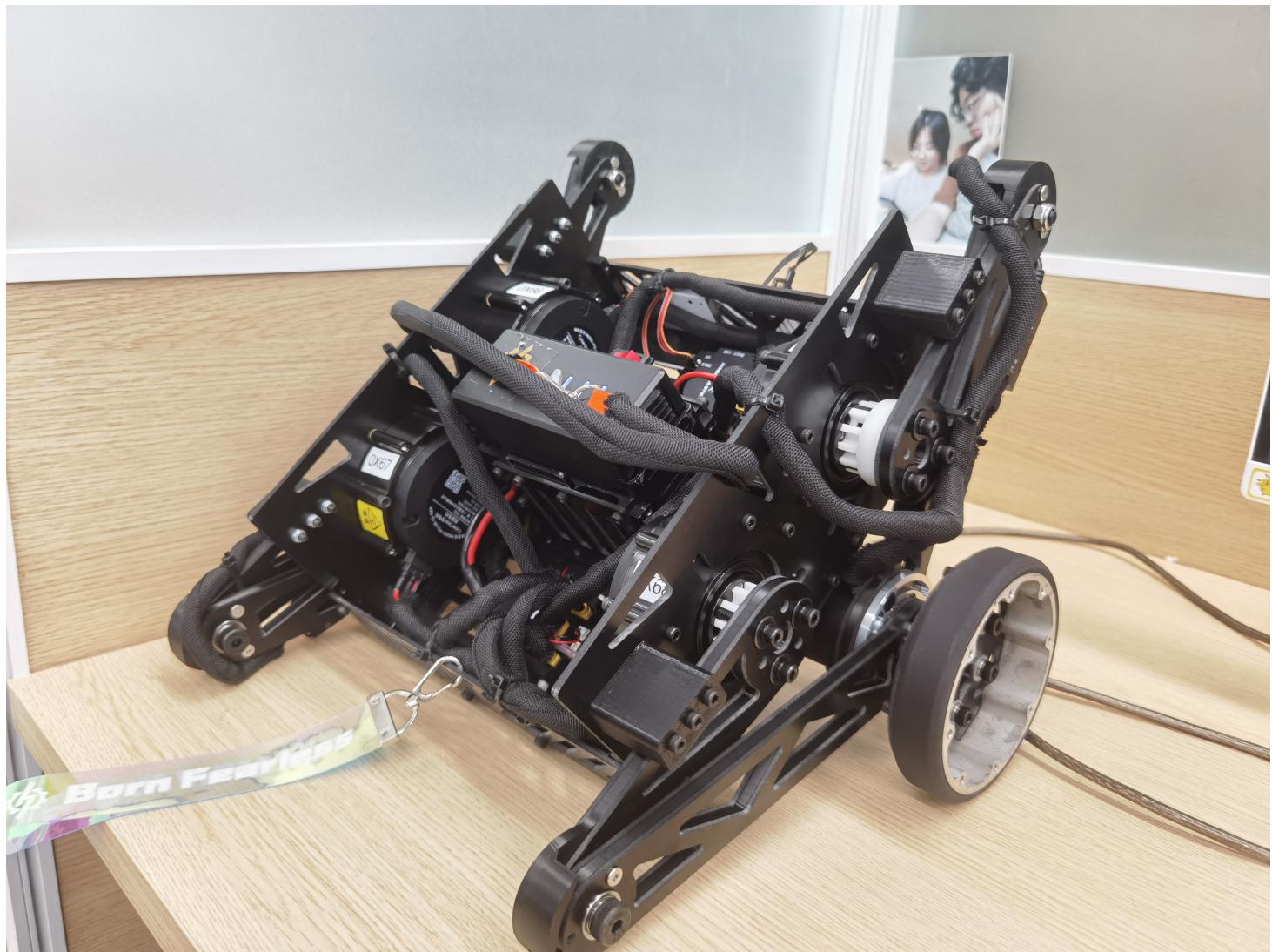
**易操作**：能在一个遥控器上实现所有操作，能够检测机器人是否离地，能稳定自启。

**抗干扰**：遇到持续外力和突发冲撞能保持稳定。

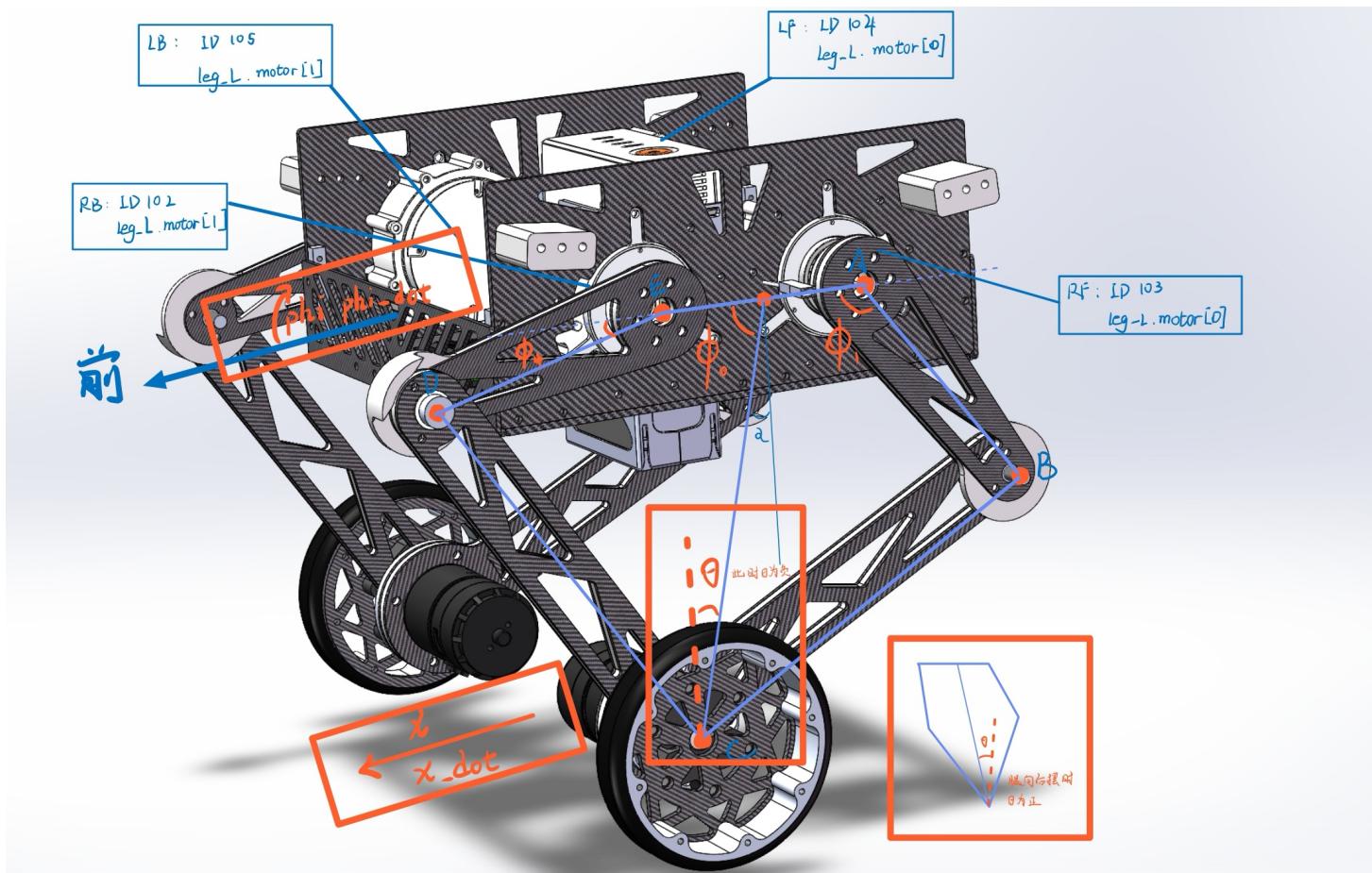
**跳跃**：能够稳定跳跃。

# 轮腿机器人相关说明

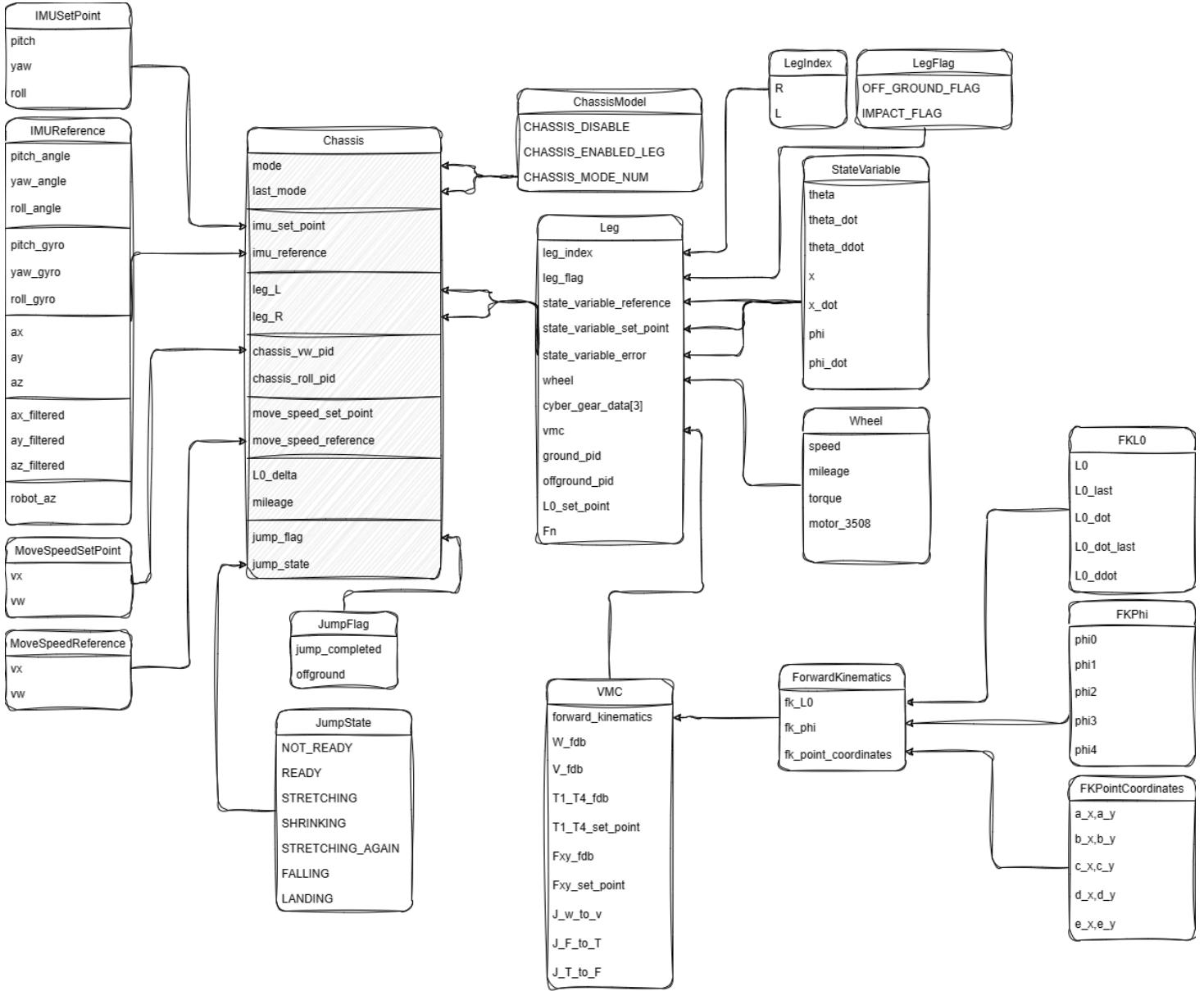
## 整机图片



# 正方向定义

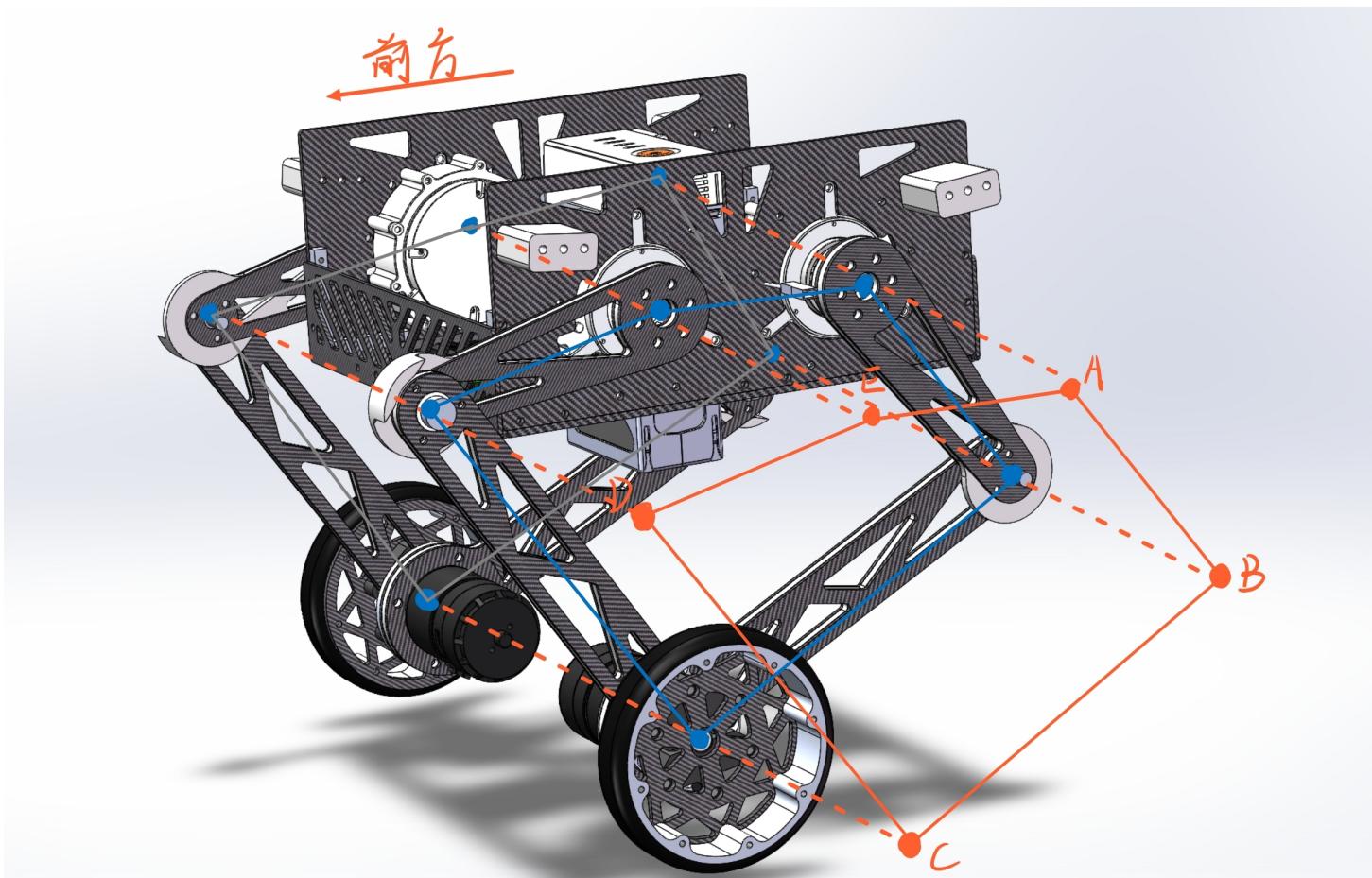


## 底盘结构体从属关系



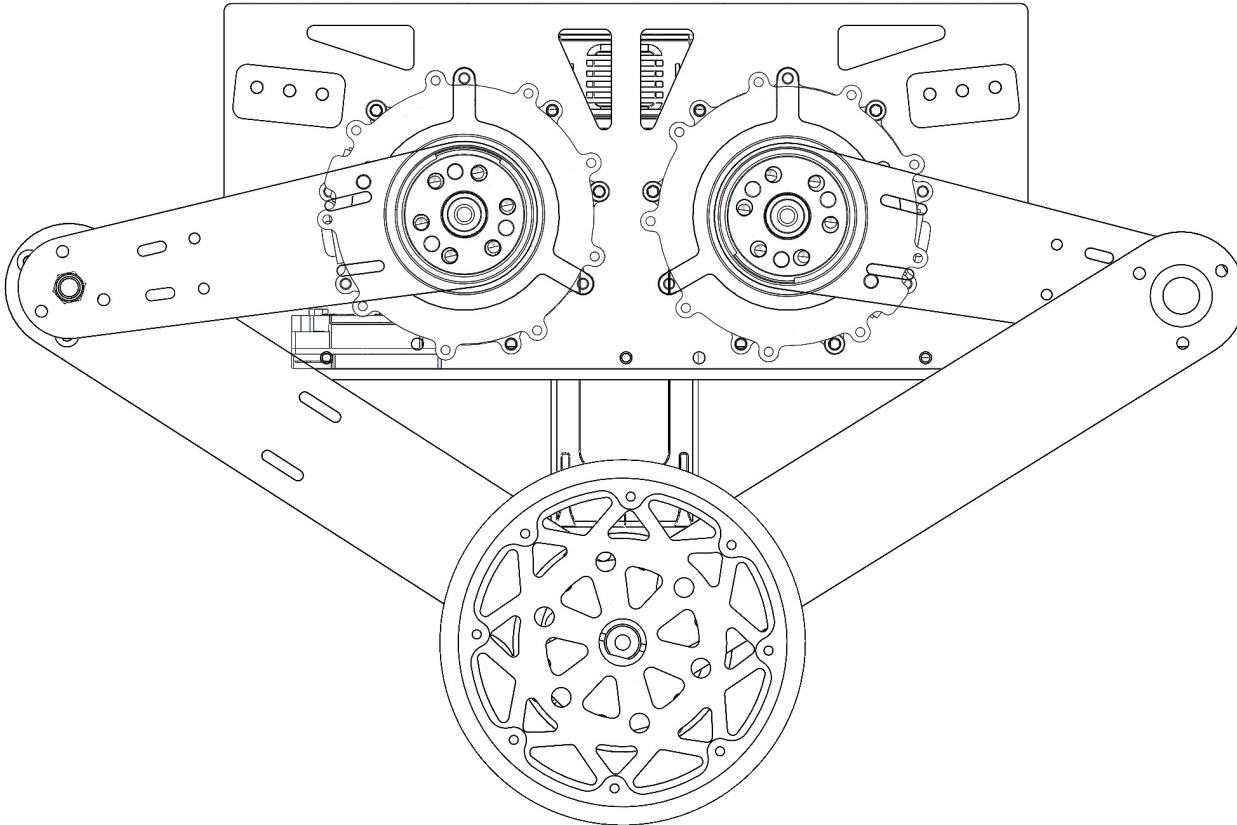
## 控制方式说明

机器人左右两腿皆从左视图参照正方向定义中控制两腿，参考示意图：



# 项目系统与功能设计

## 机械设计



五连杆机器人腿是一种常见的机械结构，具有一些优势，特别适用于复杂的应用场景。

**稳定性：**五连杆结构可以提供较好的稳定性，使得机器人在行走或运动时更加平稳。这对于需要在不同地形或工作环境中移动的机器人尤为重要。

**适应性强：**五连杆机器人腿的设计使得机器人能够适应不同高度和倾斜角度的地面。这种灵活性使得机器人能够在多样化的环境中行动，从而拓展了其应用领域。

**精准控制：**五连杆结构提供了较好的关节控制，使得机器人能够在移动和定位过程中实现更为精准的运动。这对于执行复杂任务或避开障碍物非常有帮助。

**有效载荷能力：**由于五连杆机器人腿的结构稳定，它通常具有较高的有效载荷能力。这使得机器人能够携带更多的设备、传感器或负载，扩展了其应用范围。

**节省空间：**五连杆腿结构通常能够以较小的空间实现较大的运动范围，这对于有限空间内的机器人应用尤为重要，如在狭窄的通道中行走或执行任务。

**可编程性：**五连杆机器人腿的关节可编程性较高，使得开发人员可以更容易地实现各种运动模式和行为，从而适应不同的任务需求。

总体而言，五连杆机器人腿的优势在于其结构稳定性、适应性强、精准控制和较高有效载荷能力。

## 机械参数

### 尺寸相关

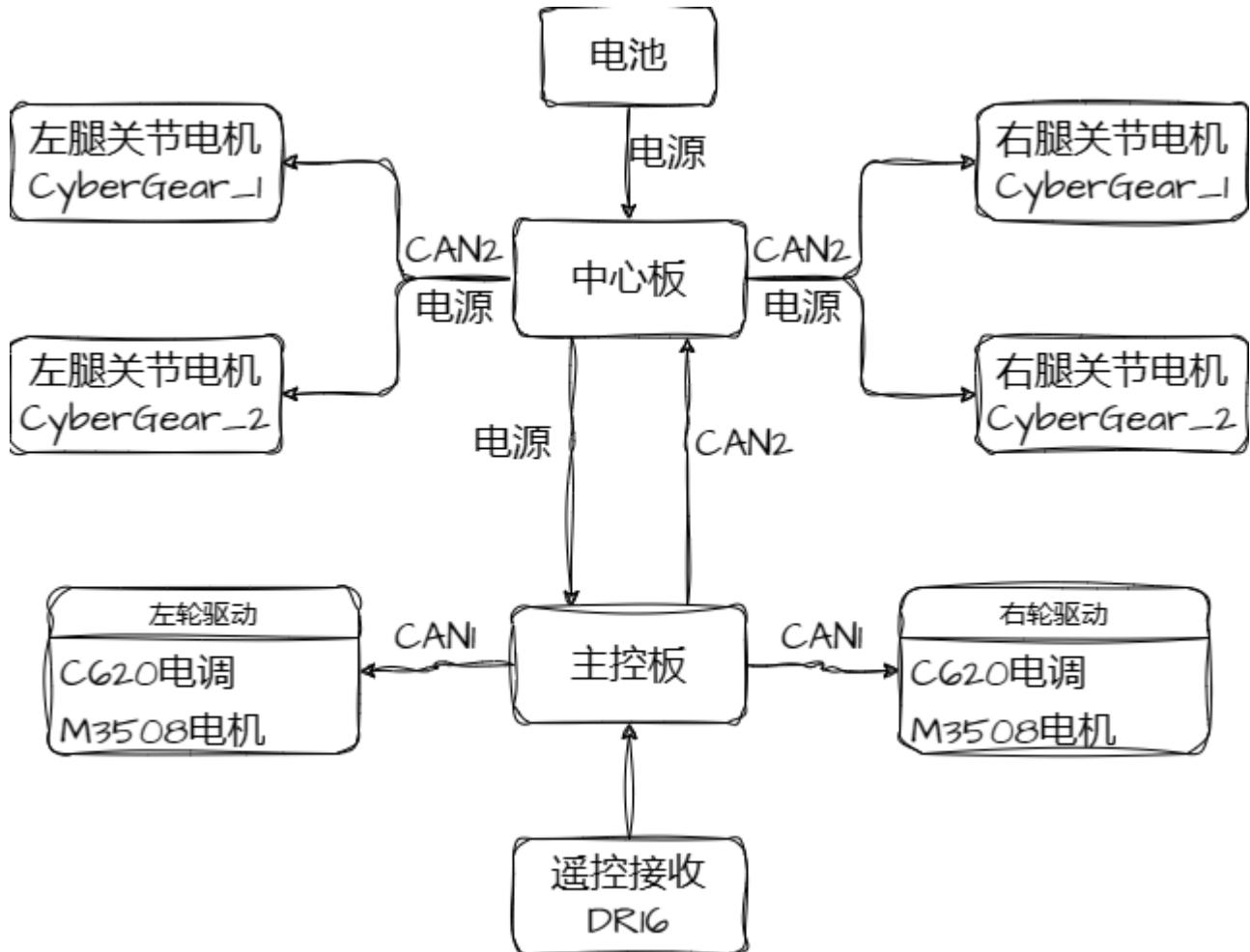
部件名称	数值	单位
杆L1	0.11	m
杆L2	0.18	m
杆L3	0.18	m
杆L4	0.11	m
杆L5	0.09	m
驱动轮半径	0.05	m
底盘绕Z轴旋转半径	0.1577	m
大腿趴下机械限位	0.10472	rad

### 质量相关

部件名称	数值	单位
整车	4.635	kg
轮子	0.183	kg
单腿	0.715	kg
车体	2.839	kg

# 硬件设计

## 整机硬件框图



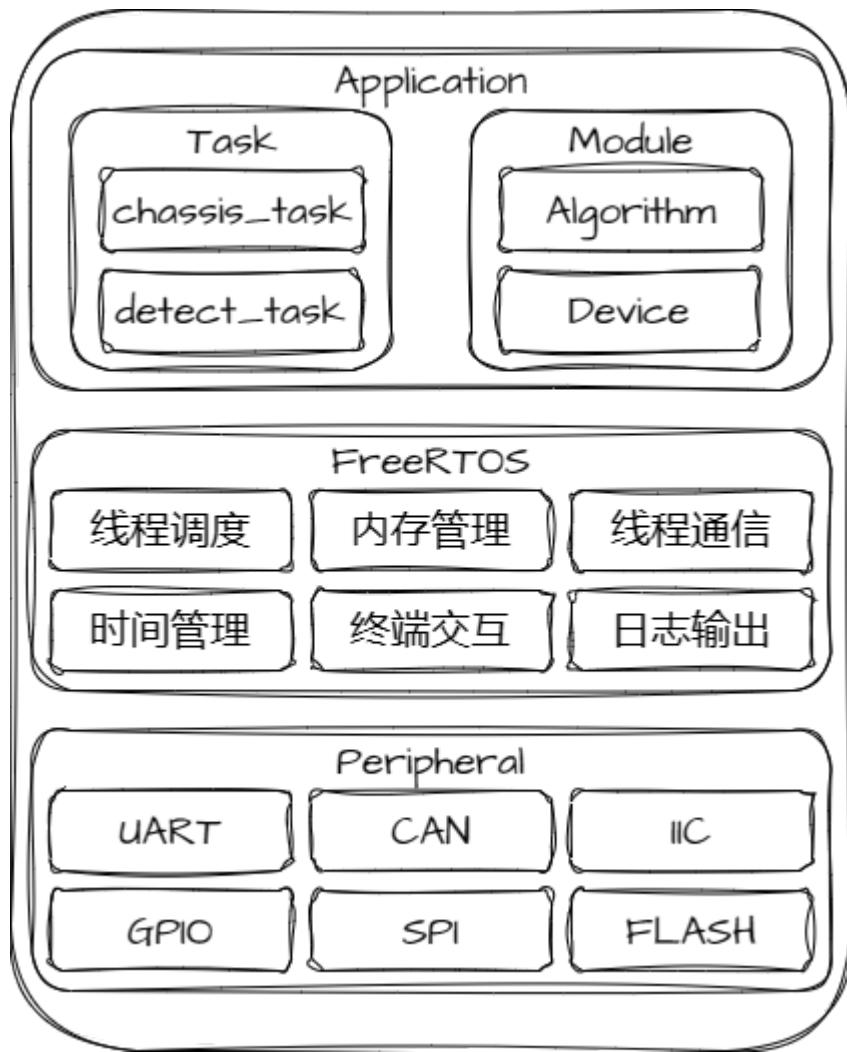
整车电源由电池提供，电池直接连接中心板再分流到各电机以及主控板，可以有效保护主控板。机器人主要执行器都是用CAN通信，整机一共使用2个CAN网络。

轮子驱动电机电子调速器CAN总线直接连接主控板，接入CAN1网络；关节电机发送扩展帧优先级较低，为了避免丢失数据，将四个关节电机单独接入CAN2网络，并通过中心板连接主控板。

# 软件设计

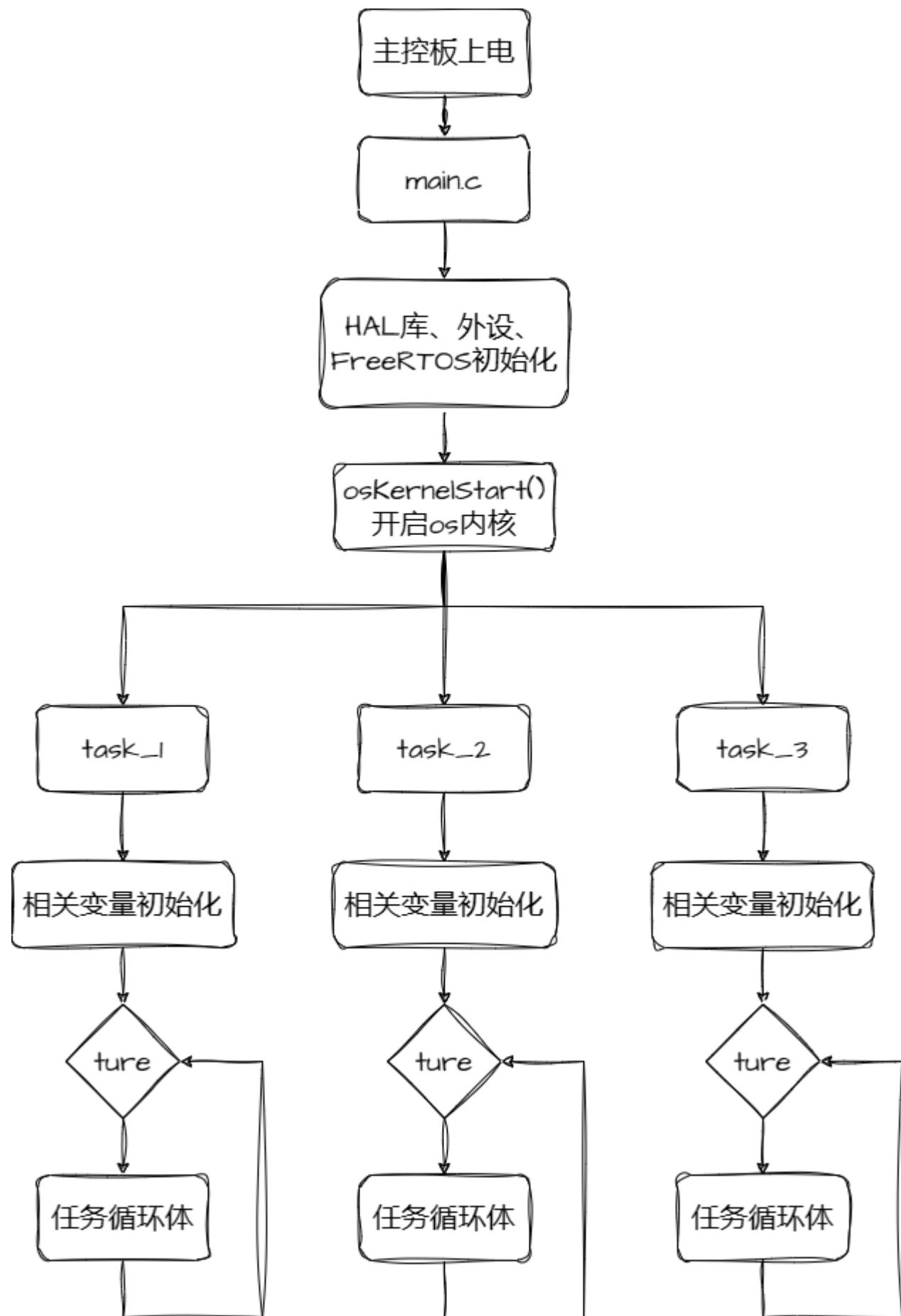
## 系统架构

机器人的主控程序采用相同的驱动层和整体架构。主控程序以 FreeRTOS 作为片上操作系统，实现多线程间的调度和通信。FreeRTOS 操作系统是一个轻量级的操作系统，提供了线程管理、时间管理、内存管理等。

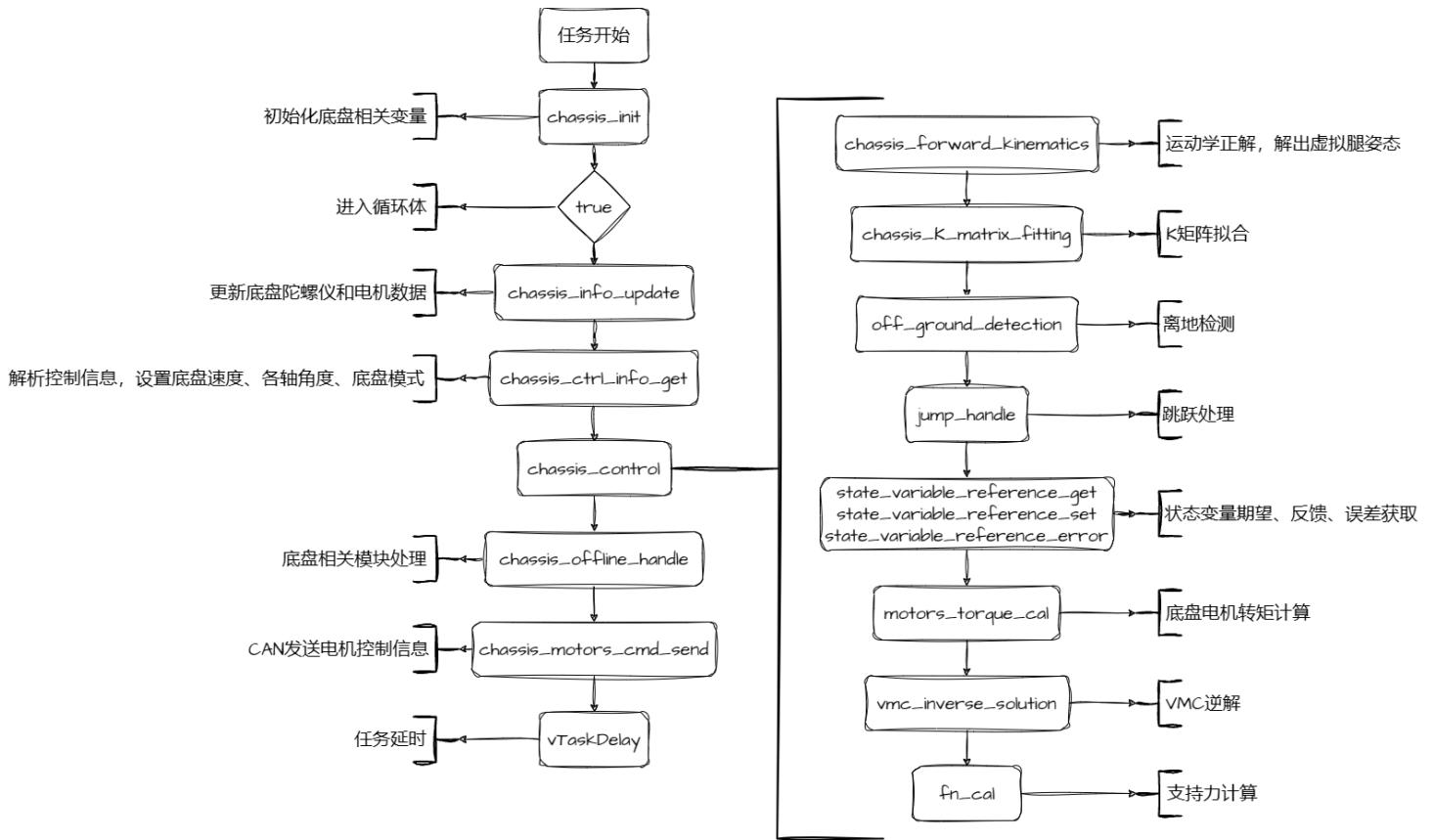


系统以FreeRTOS为片上操作系统，底层的外设驱动涉及 UART、CAN、IIC、GPIO、SPI、FLASH 等。顶层应用层包含 Task 和 Module 两方面，Task 中主要为与系统运行相关的线程，Module 主要为系统运行过程所需要使用的算法(控制算法、解算算法、滤波算法等)和工具（遥控器、传感器等）。

# 运行流程



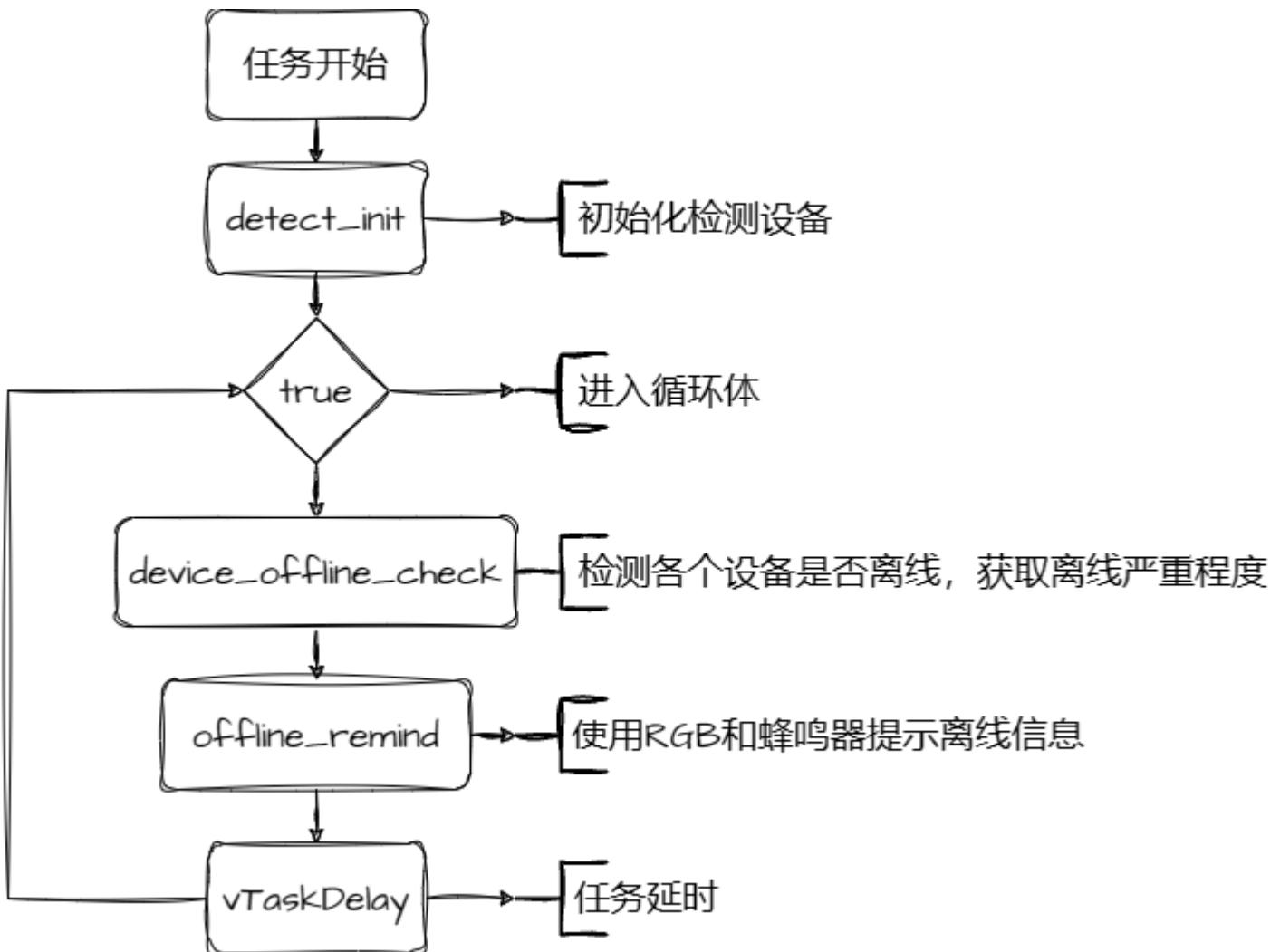
# 底盘任务流程



底盘运行任务开始后，会进行一段时间延时，再进行底盘相关变量的初始化。初始化完毕后进入循环体。循环体中，`chassis_info_update()`更新底盘陀螺仪和电机相关数据；`chassis_ctrl_info_get()`解析遥控发送的底盘控制信息；`chassis_control()`根据底盘控制命令对底盘进行运动学正逆解算，并进行LQR和VMC计算出电机的期望转矩，并根据电机说明转换为期望电流；`chassis_device_offline_handle()`检测底盘相关设备的是否离线，若离线，则使相关电机失能，防止底盘失控；`chassis_motors_can_send()`发送can数据控制电机。

腿部控制处理中，`chassis_forward_kinematics()`先结算处虚拟腿姿态；`chassis_K_matrix_fitting()`根据虚拟腿长进行LQR的K矩阵拟合；`off_ground_detection()`根据腿部支持力判断是否离线以及进行离线处理；`jump_handle()`在接到跳跃命令之后进行跳跃处理；`state_variable_reference`相关函数进行状态变量的期望、反馈、误差获取并在`motors_torque_cal()`中计算电机期望转矩；`vmc_inverse_solution()`进行运动学和动力学逆解；最后`fn_cal()`进行支持力计算。

# 离线检测任务流程



detect\_task是用来检测离线设备的任务。任务开始时, detect\_device\_init()对被检设备进行初始化, 设置离线判断的时间阈值并默认设备离线。初始化完毕后进入循环体。每次循环开始会初始化最高警告等级和离线设备数量, 接着检测每个设备离线间隔时间是否超过对应设备的间隔阈值, 超过的标记为离线, 并且记录最高警告等级。offline\_remind()中根据最高离线等级和离线设备数量, 利用蜂鸣器和RGB灯发出提醒信息。

## 功能设计

### 移动控制

机器人可以在保持机体Pitch轴稳定的情况下进行移动和旋转运动。



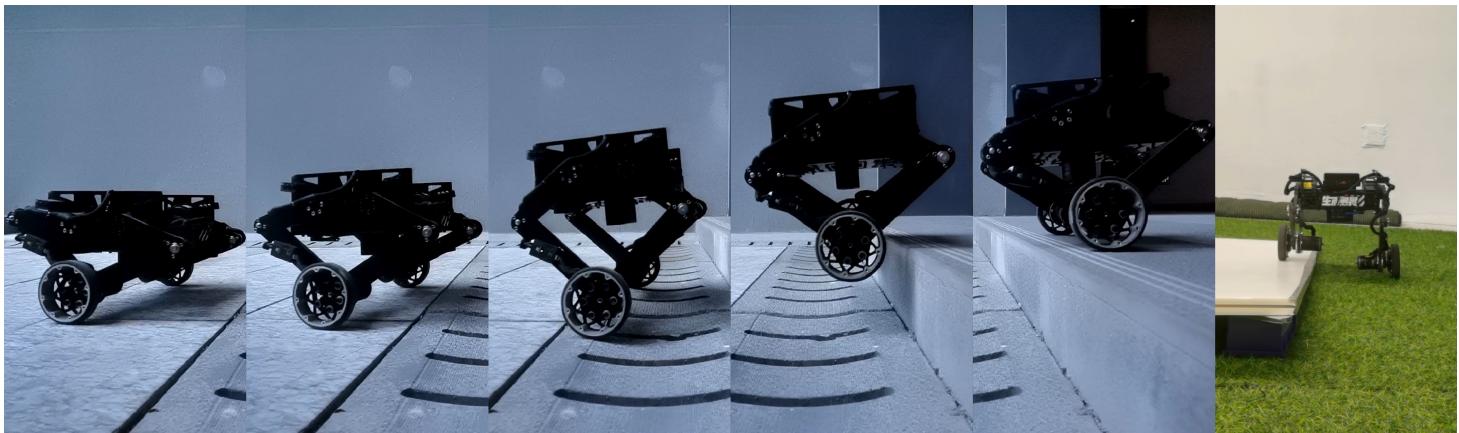
## 自由度控制

Pitch、Yaw、Roll三轴自由度调节，并配合腿部控制完成Z轴升降。



## 适应性控制

复杂路面上的自适应，能完成Roll轴自稳，下台阶冲击抵抗，转向重心偏移。



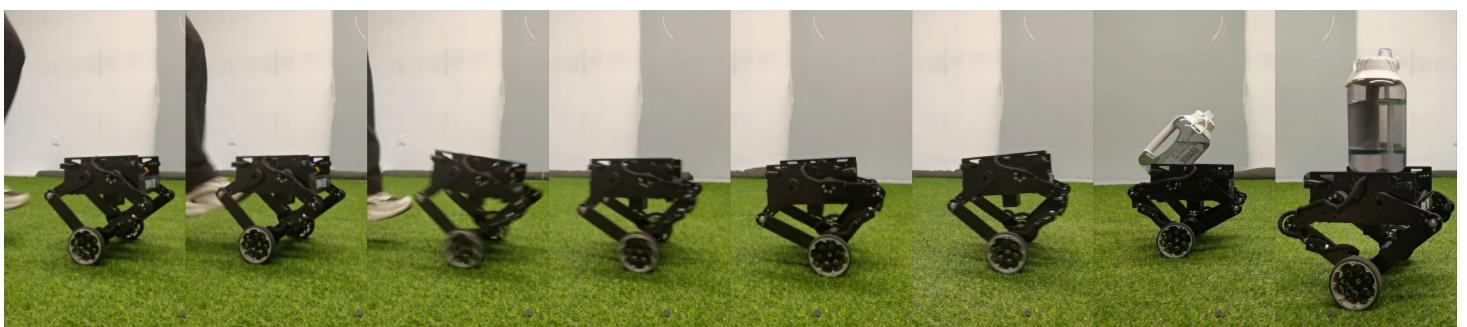
## 操作控制

能在一个遥控器上实现所有操作，能够检测机器人是否离地，能稳定自启。



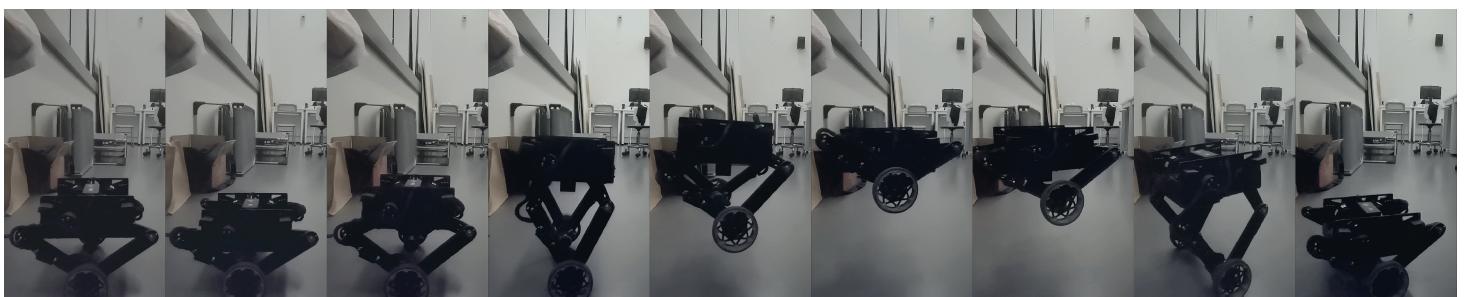
## 抗干扰能力

遇到持续外力和突发冲撞能保持稳定。



## 跳跃

能够稳定跳跃，实现原地跳跃和向前跳跃。



## 项目关键技术与实现

本部分将主要介绍机器人的平衡与运动控制和跳跃功能的实验，平衡与运动控制中分为系统建模和控制器设计，控制器设计中涉及到LQR和VMC。

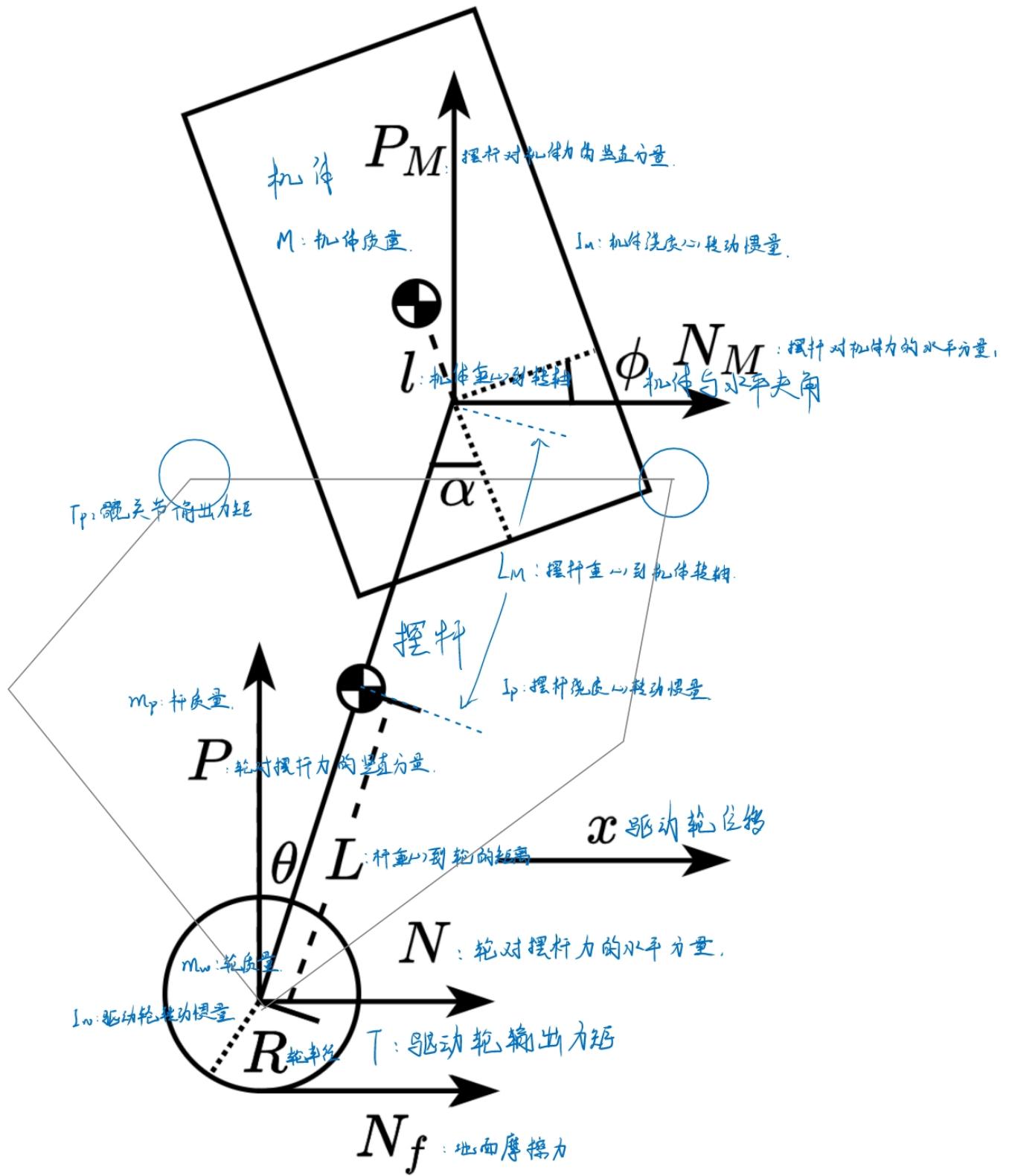
# 机器人运动和控制

## 系统建模

### 模型定义

以下只对单腿模型做分析。

轮腿倒摆模型如下图：



## 模型变量

符号	含义	正方向	单位
$\theta$	腿部和竖直方向的夹角	向后摆腿时为正	rad

符号	含义	正方向	单位
$x$	驱动轮位移	前进方向	m
$\varphi$	机体和水平夹角	上仰为正	rad
T	驱动轮输出力矩	向前转动为正	Nm
$T_p$	虚拟关节输出力矩	向后摆腿时为正	Nm
N	驱动轮对腿的水平分力	前进方向	N
P	驱动轮对腿的竖直分力	竖直向上	N
$N_M$	腿对机体的水平分力	前进方向	N
$P_M$	腿对机体的竖直分力	竖直方向	N
$N_f$	地面对驱动轮的摩擦力	前进方向	N

## 参数

符号	含义	单位
R	驱动轮半径	m
L	腿重心到驱动轮转轴距离	m
$L_M$	腿重心到机体转轴距离	m
I	机体重心到机体转轴距离	m
$m_w$	驱动轮质量	kg
$m_p$	腿质量	kg
M	机体质量	kg
$I_w$	驱动轮转动惯量	$kg \cdot m^2$
$I_p$	腿绕质心转动惯量	$kg \cdot m^2$
$I_M$	机体绕质心转动惯量	$kg \cdot m^2$

# 经典力学分析

对**驱动轮**进行受力分析

牛顿第二定律：

$$m_w \ddot{x} = (N_f - N) \quad (1.1)$$

刚体定轴转动定律：

$$I_w \frac{\ddot{x}}{R} = T - N_f R \quad (1.2)$$

合并(1.1)(1.2)，解得 $\ddot{x}$ ：

$$\ddot{x} = \frac{T - NR}{\frac{I_w}{R} + m_w R} \quad (1.3)$$

对**腿部**进行受力分析

牛顿第二定律水平方向：

$$N - N_M = m_p \frac{\partial^2}{\partial t^2} (x + L \sin \theta) \quad (1.4)$$

牛顿第二定律竖直方向：

$$P - P_M - m_p g = m_p \frac{\partial^2}{\partial t^2} (L \cos \theta) \quad (1.5)$$

刚体定轴转动定律：

$$I_p \ddot{\theta} = (PL + P_M L_M) \sin \theta - (NL + N_M L_M) \cos \theta - T + T_p \quad (1.6)$$

对**机体**进行受力分析

牛顿第二定律水平方向：

$$N_M = M \frac{\partial^2}{\partial t^2} (x + (L + L_M) \sin \theta - l \sin \phi) \quad (1.7)$$

牛顿第二定律竖直方向：

$$P_M - Mg = M \frac{\partial^2}{\partial t^2} ((L + L_M) \cos \theta + l \cos \theta) \quad (1.8)$$

刚体定轴转动定律：

$$I_M \ddot{\phi} = T_p + N_M l \cos \phi + P_M l \sin \theta \quad (1.9)$$

## 状态空间模型

定义机器人的状态向量

$$x = \begin{bmatrix} \theta \\ \dot{\theta} \\ x \\ \dot{x} \\ \phi \\ \dot{\phi} \end{bmatrix} \quad u = \begin{bmatrix} T \\ T_p \end{bmatrix}$$

根据(1.4)、(1.5)、(1.7)、(1.8)消去中间变量 $P, N, P_M, N_M$ ,之后可以用Jacobian函数求AB阵。

## 控制器设计

### LQR

LQR (外文名linear quadratic regulator)即线性二次型调节器，LQR可得到状态线性反馈的最优控制规律，易于构成闭环最优控制。LQR最优控制利用廉价成本可以使原系统达到较好的性能指标(事实也可以对不稳定的系统进行整定)，而且方法简单便于实现。

在求出A、B矩阵之后，使用MATLAB算出K矩阵：

$$K = lqrd(A_{ballanced}, B_{ballanced}, Q, R, T)$$

在实际工作中，腿部长度是不断变化的，在最大腿长和最小腿长中平均分成50份，对系统模型进行一次线性化，并求解系统的反馈增益矩阵 $K$ 。对矩阵 $K$ 的每个元素随虚拟腿长变化拟合成多项式方程：

$$K_{ij}(L_0) = p_{0|ij} + p_{1|ij}L_0 + p_{2|ij}L_0^2 + p_{3|ij}L_0^3$$

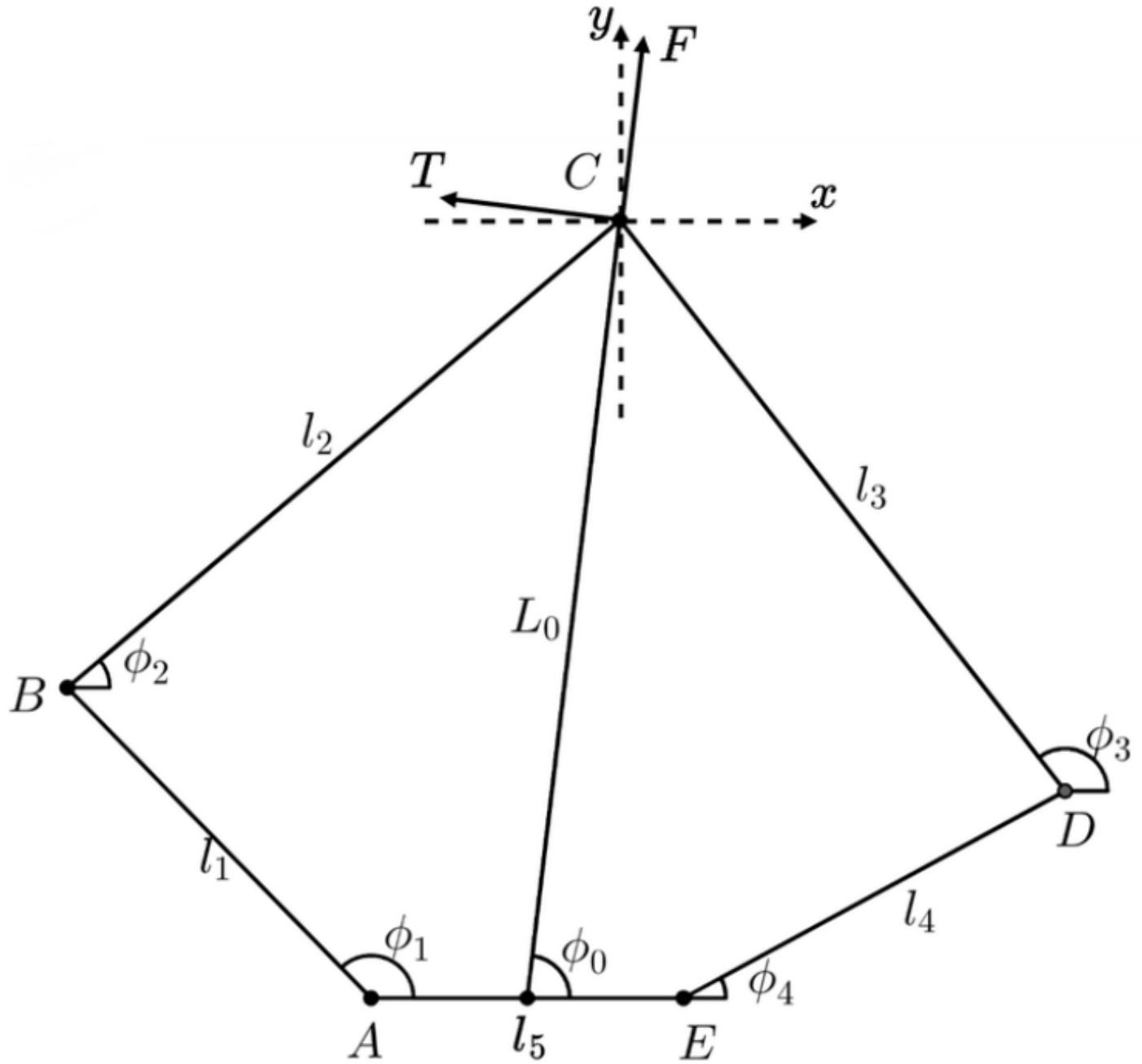
实现通过获取虚拟腿长来直接得出不同虚拟腿长对应的 $K$ 矩阵。

### VMC

VMC (virtual model control) 是一种直觉控制方式，其关键是在每个需要控制的自由度上构造恰当的虚拟构件以产生合适的虚拟力。虚拟力不是实际执行机构的作用力或力矩，而是通过执行机构的作用经过

机构转换而成。对于一些控制问题，我们可能需要将工作空间 (Task Space) 的力或力矩映射成关节空间 (Joint Space) 的关节力矩。

将腿部五连杆机构进行简化：



其中A、E两转动副为髋关节，由CyberGear电机驱动，B、D两点为膝关节，无动力。VMC控制的关键在于与五连杆机构末端驱动轮C点的位置，和沿虚拟腿方向的推力和沿重心轴的力矩。

### 解出C点极坐标

$\phi_1$ 、 $\phi_4$ 角度可以由关节电机编码器直接获取

根据角度和边长的几何关系，列出C点坐标等式：

$$\begin{cases} x_B + l_2 \cos \phi_2 = x_D + l_3 \cos \phi_3 \\ y_B = l_2 \sin \phi_2 = y_D + l_3 \sin \phi_3 \end{cases} \quad (2.1)$$

解得 $\phi_2$ :

$$\phi_2 = 2 \arctan \frac{2l_2(y_D - y_B) + \sqrt{(2l_2(x_D - x_B))^2 + (2l_2(y_D - y_B))^2 - (l_2^2 + l_{BD}^2 - l_3^2)^2}}{2l_2(x_D - x_B) + (l_2^2 + l_{BD}^2 - l_3^2)} \quad (2.2)$$

其中 $l_{BD}$ :

$$l_{BD} = \sqrt{(x_D - x_B)^2 + (y_D - y_B)^2}$$

用 $\phi_2$ 表示 $C$ 点直角坐标:

$$\begin{cases} x_C = l_1 \cos \phi_1 + l_2 \cos \phi_2 \\ y_C = l_1 \sin \phi_1 + l_2 \sin \phi_2 \end{cases} \quad (2.3)$$

则 $C$ 点极坐标:

$$\begin{cases} L_0 = \sqrt{(x_C - \frac{l_5}{2})^2 + y_C^2} \\ \phi_0 = \arctan \frac{y_C}{x_C - \frac{l_5}{2}} \end{cases} \quad (2.4)$$

### 雅可比矩阵

该部分将获得沿虚拟腿方向的推力 $F$ 和沿重心轴的力矩 $T_p$ ,与两关节电机转矩 $T_1$ 、 $T_2$ 的关系。  
分别定义:

$$x = \begin{bmatrix} L_0 \\ \phi_0 \end{bmatrix} q = \begin{bmatrix} \phi_1 \\ \phi_4 \end{bmatrix}$$

则正运动学模型:

$$x = f(q)$$

对该模型求全微分:

$$\delta x = \begin{bmatrix} \frac{\delta f_1}{\delta \phi_1} & \frac{\delta f_1}{\delta \phi_4} \\ \frac{\delta f_2}{\delta \phi_1} & \frac{\delta f_2}{\delta \phi_4} \end{bmatrix} \delta q$$

则雅可比矩阵为:

$$J = \begin{bmatrix} \frac{\delta f_1}{\delta \phi_1} & \frac{\delta f_1}{\delta \phi_4} \\ \frac{\delta f_2}{\delta \phi_1} & \frac{\delta f_2}{\delta \phi_4} \end{bmatrix}$$

全微分式表示为：

$$\delta x = J \delta q$$

即可用雅可比矩阵  $J$  得到关节速度  $\dot{q}$  和五连杆虚拟腿长和虚拟关节的变化率  $\dot{x}$ ，根据虚功原理：

$$T^T \delta q + (-F)^T \delta x = 0$$

其中  $T = \begin{bmatrix} T_1 \\ T_2 \end{bmatrix}$ ,  $F = \begin{bmatrix} T \\ T_p \end{bmatrix}$ 。结合  $\delta x = J \delta q$ , 得到：

$$T = J^T F$$

则得到了沿虚拟腿方向的推力  $F$  和沿重心轴的力矩  $T_p$ , 与两关节电机转矩  $T_1$ 、 $T_2$  的关系。

## 控制逻辑及代码

### 运行学正解

该部分可以通过两个关节电机的角度反馈解出虚拟腿长  $L_0$  以及虚拟关节角度  $\phi_0$ ，以左腿为例：

```

chassis.leg_L.vmc.forward_kinematics.fk_point_coordinates.b_x =
    cosf(chassis.leg_L.vmc.forward_kinematics.fk_phi.phi1) * L1;
chassis.leg_L.vmc.forward_kinematics.fk_point_coordinates.b_y =
    sinf(chassis.leg_L.vmc.forward_kinematics.fk_phi.phi1) * L1;
chassis.leg_L.vmc.forward_kinematics.fk_point_coordinates.d_x =
    cosf(chassis.leg_L.vmc.forward_kinematics.fk_phi.phi4) * L4
    +
    L5;
chassis.leg_L.vmc.forward_kinematics.fk_point_coordinates.d_y =
    sinf(chassis.leg_L.vmc.forward_kinematics.fk_phi.phi4) * L4;

fp32 L_A0 = (chassis.leg_L.vmc.forward_kinematics.fk_point_coordinates.d_x
    -
    chassis.leg_L.vmc.forward_kinematics.fk_point_coordinates.b_x) * 2.f * L2;
fp32 L_B0 = (chassis.leg_L.vmc.forward_kinematics.fk_point_coordinates.d_y
    -
    chassis.leg_L.vmc.forward_kinematics.fk_point_coordinates.b_y) * 2.f * L2;
fp32 L_BD_sq = (chassis.leg_L.vmc.forward_kinematics.fk_point_coordinates.d_x
    -
    chassis.leg_L.vmc.forward_kinematics.fk_point_coordinates.b_x)
    *
    (chassis.leg_L.vmc.forward_kinematics.fk_point_coordinates.d_x
        -
        chassis.leg_L.vmc.forward_kinematics.fk_point_coordinates.b_x)
    +
    (chassis.leg_L.vmc.forward_kinematics.fk_point_coordinates.d_y
        -
        chassis.leg_L.vmc.forward_kinematics.fk_point_coordinates.b_y)
    *
    (chassis.leg_L.vmc.forward_kinematics.fk_point_coordinates.d_y
        -
        chassis.leg_L.vmc.forward_kinematics.fk_point_coordinates.b_y);
fp32 L_C0 = L2 * L2 + L_BD_sq - L3 * L3;

fp32 temp = L_A0 * L_A0 + L_B0 * L_B0 - L_C0 * L_C0;
fp32 y = L_B0 + sqrtf(ABS(temp));
fp32 x = L_A0 + L_C0;
chassis.leg_L.vmc.forward_kinematics.fk_phi.phi2 = 2.0f * atan2f(y, x);

chassis.leg_L.vmc.forward_kinematics.fk_point_coordinates.c_x =
    L1 * cos(chassis.leg_L.vmc.forward_kinematics.fk_phi.phi1)
    +
    L2 * cos(chassis.leg_L.vmc.forward_kinematics.fk_phi.phi2);
chassis.leg_L.vmc.forward_kinematics.fk_point_coordinates.c_y =
    L1 * sin(chassis.leg_L.vmc.forward_kinematics.fk_phi.phi1)
    +
    L2 * sin(chassis.leg_L.vmc.forward_kinematics.fk_phi.phi2);
y = chassis.leg_L.vmc.forward_kinematics.fk_point_coordinates.c_y
    -
    chassis.leg_L.vmc.forward_kinematics.fk_point_coordinates.d_y;
x = chassis.leg_L.vmc.forward_kinematics.fk_point_coordinates.c_x
    -
    chassis.leg_L.vmc.forward_kinematics.fk_point_coordinates.d_x;
chassis.leg_L.vmc.forward_kinematics.fk_phi.phi3 = atan2f(y, x);

temp = (chassis.leg_L.vmc.forward_kinematics.fk_point_coordinates.c_x - L5 * 0.5f)
    *
    (chassis.leg_L.vmc.forward_kinematics.fk_point_coordinates.c_x - L5 * 0.5f)

```

```

+ chassis.leg_L.vmc.forward_kinematics.fk_point_coordinates.c_y
    * chassis.leg_L.vmc.forward_kinematics.fk_point_coordinates.c_y;
chassis.leg_L.vmc.forward_kinematics.fk_L0.L0_last = chassis.leg_L.vmc.forward_kinematics.fk_L
chassis.leg_L.vmc.forward_kinematics.fk_L0.L0 = sqrtf(ABS(temp));
chassis.leg_L.vmc.forward_kinematics.fk_L0.L0_dot_last = chassis.leg_L.vmc.forward_kinematics.
chassis.leg_L.vmc.forward_kinematics.fk_L0.L0_dot =
    (chassis.leg_L.vmc.forward_kinematics.fk_L0.L0 - chassis.leg_L.vmc.forward_kinematics.fk_L
     / (CHASSIS_PERIOD * MILLISECOND_TO_SECOND));
chassis.leg_L.vmc.forward_kinematics.fk_L0.L0_ddot =
    (chassis.leg_L.vmc.forward_kinematics.fk_L0.L0_dot - chassis.leg_L.vmc.forward_kinematics.
     / (CHASSIS_PERIOD * MILLISECOND_TO_SECOND));
y = chassis.leg_L.vmc.forward_kinematics.fk_point_coordinates.c_y;
x = chassis.leg_L.vmc.forward_kinematics.fk_point_coordinates.c_x - L5 * 0.5f;
chassis.leg_L.vmc.forward_kinematics.fk_phi.phi0 = atan2f(y, x);

```

## K矩阵拟合

该通过运动学正解得到的实时虚拟腿长拟合出该状态下的K矩阵：

```

void chassis_K_matrix_fitting(fp32 L0, fp32 K[6], const fp32 KL[6][4]) {
    for (int i = 0; i < 6; i++) {
        K[i] = KL[i][0] * powf(L0, 3) + KL[i][1] * powf(L0, 2) + KL[i][2] * powf(L0, 1) + KL[i][3] :
    }
}

```

## 状态变量反馈获取

该部分获取LQR控制所需的状态变量实时反馈值、期望值、误差值：

```

static void leg_state_variable_reference_get(struct Leg *leg) {
    if (leg == NULL) {
        return;
    }
    leg->state_variable_reference.theta_last = leg->state_variable_reference.theta;

    if (leg->leg_index == L) {
        leg->state_variable_reference.theta =
            cal_leg_theta(leg->vmc.forward_kinematics.fk_phi.phi0, -chassis.imu_reference.pitch_ang);
    } else if (leg->leg_index == R) {
        leg->state_variable_reference.theta =
            cal_leg_theta(leg->vmc.forward_kinematics.fk_phi.phi0, -chassis.imu_reference.pitch_ang);
    }

    leg->state_variable_reference.theta_dot_last = leg->state_variable_reference.theta_dot;
    leg->state_variable_reference.theta_dot =
        (leg->state_variable_reference.theta - leg->state_variable_reference.theta_last)
        / (CHASSIS_PERIOD * MILLISECOND_TO_SECOND);
    leg->state_variable_reference.theta_ddot =
        (leg->state_variable_reference.theta_dot - leg->state_variable_reference.theta_dot_last)
        / (CHASSIS_PERIOD * MILLISECOND_TO_SECOND);

    if (leg->leg_index == L) {
        leg->state_variable_reference.x = -leg->wheel.mileage;
        leg->state_variable_reference.x_dot = -leg->wheel.speed;
    } else if (leg->leg_index == R) {
        leg->state_variable_reference.x = leg->wheel.mileage;
        leg->state_variable_reference.x_dot = leg->wheel.speed;
    }
}

if (leg->leg_index == L) {
    leg->state_variable_reference.phi = -chassis.imu_reference.pitch_angle;
    leg->state_variable_reference.phi_dot = -chassis.imu_reference.pitch_gyro;
} else if (leg->leg_index == R) {
    leg->state_variable_reference.phi = -chassis.imu_reference.pitch_angle;
    leg->state_variable_reference.phi_dot = -chassis.imu_reference.pitch_gyro;
}
//todo 对位移处理，提高刹车性能
}

static void leg_state_variable_set_point_set(struct Leg *leg, fp32 vx) {
    if (leg == NULL) {
        return;
    }

```

```

}

leg->state_variable_set_point.x = 0;
leg->state_variable_set_point.x_dot = vx;
leg->state_variable_set_point.theta = 0;
leg->state_variable_set_point.theta_dot = 0;
leg->state_variable_set_point.phi = chassis.imu_set_point.pitch;
leg->state_variable_set_point.phi_dot = 0;
}

static void leg_state_variable_error_get(struct Leg *leg) {
    if (leg == NULL) {
        return;
    }

    leg->state_variable_error.x = leg->state_variable_reference.x - leg->state_variable_set_point.x;
    leg->state_variable_error.x_dot = leg->state_variable_reference.x_dot - leg->state_variable_set_point.x_dot;
    leg->state_variable_error.theta = leg->state_variable_reference.theta - leg->state_variable_set_point.theta;
    leg->state_variable_error.theta_dot =
        leg->state_variable_reference.theta_dot - leg->state_variable_set_point.theta_dot;
    leg->state_variable_error.phi = leg->state_variable_reference.phi - leg->state_variable_set_point.phi;
    leg->state_variable_error.phi_dot = leg->state_variable_reference.phi_dot - leg->state_variable_set_point.phi_dot;
}

```

## 电机期望转矩获取

该部分计算电机期望转矩，其中关节电机加入了腿长控制，将腿长的期望和反馈输入pid控制器中，输出沿腿方向的力，并加入车体重力前馈：

```

static void wheel_motors_torque_set_point_cal(struct Leg *leg) {
    if (leg == NULL) {
        return;
    }
    if (leg->leg_index == L) {
        leg->wheel.torque = 0;
        leg->wheel.torque += leg->state_variable_error.theta * wheel_K_L[0];//theta
        leg->wheel.torque += leg->state_variable_error.theta_dot * wheel_K_L[1];//theta_dot
        leg->wheel.torque += leg->state_variable_error.x * wheel_K_L[2];
        leg->wheel.torque += leg->state_variable_error.x_dot * wheel_K_L[3];
        leg->wheel.torque += leg->state_variable_error.phi * wheel_K_L[4];//phi
        leg->wheel.torque += leg->state_variable_error.phi_dot * wheel_K_L[5];
    } else if (leg->leg_index == R) {
        leg->wheel.torque = 0;
        leg->wheel.torque += leg->state_variable_error.theta * wheel_K_R[0];//theta
        leg->wheel.torque += leg->state_variable_error.theta_dot * wheel_K_R[1];//theta_dot
        leg->wheel.torque += leg->state_variable_error.x * wheel_K_R[2];
        leg->wheel.torque += leg->state_variable_error.x_dot * wheel_K_R[3];
        leg->wheel.torque += leg->state_variable_error.phi * wheel_K_R[4];//phi
        leg->wheel.torque += leg->state_variable_error.phi_dot * wheel_K_R[5];
    }
    if (leg->leg_index == R) {
        leg->wheel.torque *= -1;
    }
}

leg->wheel.torque += chassis.chassis_move_speed_set_point.vw;

VAL_LIMIT(leg->wheel.torque, MIN_WHEEL_TORQUE, MAX_WHEEL_TORQUE);

}

static void joint_motors_torque_set_point_cal() {
    chassis.leg_L.vmc.Fxy_set_point.E.Tp_set_point = 0;
    chassis.leg_R.vmc.Fxy_set_point.E.Tp_set_point = 0;
//R
    chassis.leg_R.vmc.Fxy_set_point.E.Tp_set_point += chassis.leg_R.state_variable_error.theta * ;
    chassis.leg_R.vmc.Fxy_set_point.E.Tp_set_point += chassis.leg_R.state_variable_error.theta_dot * ;
    chassis.leg_R.vmc.Fxy_set_point.E.Tp_set_point += chassis.leg_R.state_variable_error.x * joint;
    chassis.leg_R.vmc.Fxy_set_point.E.Tp_set_point += chassis.leg_R.state_variable_error.x_dot * ;
    chassis.leg_R.vmc.Fxy_set_point.E.Tp_set_point += chassis.leg_R.state_variable_error.phi * jo;
    chassis.leg_R.vmc.Fxy_set_point.E.Tp_set_point += chassis.leg_R.state_variable_error.phi_dot * ;
//L
    chassis.leg_L.vmc.Fxy_set_point.E.Tp_set_point += chassis.leg_L.state_variable_error.theta * ;
}

```

```

chassis.leg_L.vmc.Fxy_set_point.E.Tp_set_point += chassis.leg_L.state_variable_error.theta_dot;
chassis.leg_L.vmc.Fxy_set_point.E.Tp_set_point += chassis.leg_L.state_variable_error.x * joint;
chassis.leg_L.vmc.Fxy_set_point.E.Tp_set_point += chassis.leg_L.state_variable_error.x_dot * joint;
chassis.leg_L.vmc.Fxy_set_point.E.Tp_set_point += chassis.leg_L.state_variable_error.phi * joint;
chassis.leg_L.vmc.Fxy_set_point.E.Tp_set_point += chassis.leg_L.state_variable_error.phi_dot * joint;

pid_calc(&chassis.leg_L.ground_pid, chassis.leg_L.vmc.forward_kinematics.fk_L0.L0, chassis.leg_L.vmc.Fxy_set_point.E.Fy_set_point);
chassis.leg_L.vmc.Fxy_set_point.E.Fy_set_point = chassis.leg_L.ground_pid.out + BODY_WEIGHT * gravity;

pid_calc(&chassis.leg_R.ground_pid, chassis.leg_R.vmc.forward_kinematics.fk_L0.L0, chassis.leg_R.vmc.Fxy_set_point.E.Fy_set_point);
chassis.leg_R.vmc.Fxy_set_point.E.Fy_set_point = chassis.leg_R.ground_pid.out + BODY_WEIGHT * gravity;

VMC_positive_dynamics(&chassis.leg_R.vmc);
VMC_positive_dynamics(&chassis.leg_L.vmc);

chassis.leg_L.cyber_gear_data[2].torque = chassis.leg_L.vmc.Fxy_set_point.E.Tp_set_point;
chassis.leg_L.cyber_gear_data[0].torque = chassis.leg_L.vmc.T1_T4_set_point.E.T1_set_point;//I
chassis.leg_L.cyber_gear_data[1].torque = chassis.leg_L.vmc.T1_T4_set_point.E.T4_set_point;//II

chassis.leg_R.cyber_gear_data[2].torque = chassis.leg_R.vmc.Fxy_set_point.E.Tp_set_point;
chassis.leg_R.cyber_gear_data[0].torque = chassis.leg_R.vmc.T1_T4_set_point.E.T1_set_point;//I
chassis.leg_R.cyber_gear_data[1].torque = chassis.leg_R.vmc.T1_T4_set_point.E.T4_set_point;//II

VAL_LIMIT(chassis.leg_R.cyber_gear_data[0].torque, MIN_JOINT_TORQUE, MAX_JOINT_TORQUE);
VAL_LIMIT(chassis.leg_R.cyber_gear_data[1].torque, MIN_JOINT_TORQUE, MAX_JOINT_TORQUE);
VAL_LIMIT(chassis.leg_L.cyber_gear_data[0].torque, MIN_JOINT_TORQUE, MAX_JOINT_TORQUE);
VAL_LIMIT(chassis.leg_L.cyber_gear_data[1].torque, MIN_JOINT_TORQUE, MAX_JOINT_TORQUE);
}

```

## 动力学逆解

该部分通过雅可比矩阵求得沿虚拟腿方向的推力 $F$ 和沿重心轴的力矩 $T_p$ ,与两关节电机转矩 $T_1$ 、 $T_2$ 的关系,求出关节电机的期望转矩:

```

void Vmc_Negative_Dynamics(struct VMC *vmc, fp32 T1, fp32 T4) {
    if (vmc == NULL) {
        return;
    }
    vmc->T1_T4_fdb.E.T1_fdb = T1;
    vmc->T1_T4_fdb.E.T4_fdb = T4;

    vmc->J_T_to_F.E.x1_1 =
        vmc->forward_kinematics.fk_L0.L0 * sinf(vmc->forward_kinematics.fk_phi.phi0 - vmc->forward_kinematics.fk_phi.phi1)
        / (L1 * sinf(vmc->forward_kinematics.fk_phi.phi1 - vmc->forward_kinematics.fk_phi.phi2));
    vmc->J_T_to_F.E.x1_2 =
        vmc->forward_kinematics.fk_L0.L0 * sinf(vmc->forward_kinematics.fk_phi.phi0 - vmc->forward_kinematics.fk_phi.phi4)
        / (L4 * sinf(vmc->forward_kinematics.fk_phi.phi4 - vmc->forward_kinematics.fk_phi.phi3));
    vmc->J_T_to_F.E.x2_1 = cosf(vmc->forward_kinematics.fk_phi.phi0 - vmc->forward_kinematics.fk_phi.phi1)
        / (L1 * sin(vmc->forward_kinematics.fk_phi.phi2 - vmc->forward_kinematics.fk_phi.phi1));
    vmc->J_T_to_F.E.x2_2 = cosf(vmc->forward_kinematics.fk_phi.phi0 - vmc->forward_kinematics.fk_phi.phi3)
        / (L4 * sinf(vmc->forward_kinematics.fk_phi.phi3 - vmc->forward_kinematics.fk_phi.phi4));

    Matrix_multiply(2, 2, vmc->J_T_to_F.array, 2, 1, vmc->T1_T4_fdb.array, vmc->Fxy_fdb.array);
}

void Vmc_Negative_Kinematics(struct VMC *vmc, fp32 w1, fp32 w4) {
    if (vmc == NULL) {
        return;
    }
    vmc->W_fdb.E.w1_fdb = w1;
    vmc->W_fdb.E.w4_fdb = w4;

    vmc->J_w_to_v.E.x1_1 = (L1 * sinf(vmc->forward_kinematics.fk_phi.phi0) * sinf(vmc->forward_kinematics.fk_phi.phi1)
        * sinf(vmc->forward_kinematics.fk_phi.phi1 - vmc->forward_kinematics.fk_phi.phi2)
        - L4 * cosf(vmc->forward_kinematics.fk_phi.phi0) * sinf(vmc->forward_kinematics.fk_phi.phi1)
        * sinf(vmc->forward_kinematics.fk_phi.phi3 - vmc->forward_kinematics.fk_phi.phi4))
        / sinf(vmc->forward_kinematics.fk_phi.phi2 - vmc->forward_kinematics.fk_phi.phi3);
    vmc->J_w_to_v.E.x1_2 = (L1 * cosf(vmc->forward_kinematics.fk_phi.phi0) * sinf(vmc->forward_kinematics.fk_phi.phi1)
        * sinf(vmc->forward_kinematics.fk_phi.phi1 - vmc->forward_kinematics.fk_phi.phi2)
        + L4 * sinf(vmc->forward_kinematics.fk_phi.phi0) * sinf(vmc->forward_kinematics.fk_phi.phi1)
        * sinf(vmc->forward_kinematics.fk_phi.phi3 - vmc->forward_kinematics.fk_phi.phi4))
        / sinf(vmc->forward_kinematics.fk_phi.phi2 - vmc->forward_kinematics.fk_phi.phi3);
    vmc->J_w_to_v.E.x2_1 = (-L1 * sinf(vmc->forward_kinematics.fk_phi.phi0) * cosf(vmc->forward_kinematics.fk_phi.phi1)
        * sinf(vmc->forward_kinematics.fk_phi.phi1 - vmc->forward_kinematics.fk_phi.phi2)
        + L4 * cosf(vmc->forward_kinematics.fk_phi.phi0) * cosf(vmc->forward_kinematics.fk_phi.phi1)
        * sinf(vmc->forward_kinematics.fk_phi.phi3 - vmc->forward_kinematics.fk_phi.phi4))
        / sinf(vmc->forward_kinematics.fk_phi.phi2 - vmc->forward_kinematics.fk_phi.phi3);
}

```

```

vmc->J_w_to_v.E.x2_2 = -(L1 * cosf(vmc->forward_kinematics.fk_phi.phi0) * cosf(vmc->forward_k:
    * sinf(vmc->forward_kinematics.fk_phi.phi1 - vmc->forward_kinematics.fk_phi.phi2) * L4
    * sinf(vmc->forward_kinematics.fk_phi.phi0) * cosf(vmc->forward_kinematics.fk_phi.phi2)
    * sinf(vmc->forward_kinematics.fk_phi.phi3 - vmc->forward_kinematics.fk_phi.phi4))
    / sinf(vmc->forward_kinematics.fk_phi.phi2 - vmc->forward_kinematics.fk_phi.phi3);

Matrix_multiply(2, 2, vmc->J_w_to_v.array, 2, 1, vmc->W_fdb.array, vmc->V_fdb.array);
vmc->V_fdb.E.w0_fdb /= vmc->forward_kinematics.fk_L0.L0;
}

```

## 支持力计算

该部分计算腿部受到的支持力：

```

void leg_fn_cal(struct Leg *leg, fp32 az) {
    if (leg == NULL) {
        return;
    }
    fp32 P;
    P = leg->vmc.Fxy_fdb.E.Tp_fdb * sinf(leg->state_variable_reference.theta) / leg->vmc.forward_l
        + leg->vmc.Fxy_fdb.E.Fy_fdb * cosf(leg->state_variable_reference.theta);

    leg->wheel.imu_reference.az =
        az - leg->vmc.forward_kinematics.fk_L0.L0_ddot * cosf(leg->state_variable_reference.theta)
            + 2 * leg->vmc.forward_kinematics.fk_L0.L0_dot * leg->state_variable_reference.theta_
                * sinf(leg->state_variable_reference.theta)
            + leg->vmc.forward_kinematics.fk_L0.L0 * leg->state_variable_reference.theta_ddot
                * sinf(leg->state_variable_reference.theta)
            + leg->vmc.forward_kinematics.fk_L0.L0 * leg->state_variable_reference.theta_dot
                * leg->state_variable_reference.theta_dot * cosf(leg->state_variable_reference.the

    leg->Fn = P + WHEEL_WEIGHT * 9.8f + WHEEL_WEIGHT * leg->wheel.imu_reference.az;
}

```