

* @Author: Chen Ningzhan (2024)
* @Last edit time: 2023-10-22
* @brief: 平衡步兵技术文档
* @attention

版本更新说明

引言

参考文献以及资料汇总

文献博客
控制理论
Simscape
Webots
ADRC (了解即可)

RM2024研发迭代记录

技术要点记录
问题记录

经典倒立摆数学模型

数学模型
Simscape 仿真建模

平衡步兵动力学模型

车轮模型
车体模型
 正向运动
 转向运动
系统状态方程以及解耦

平衡步兵模型系统分析

稳定性分析
能控性分析
能观性分析

平衡步兵控制系统设计

LQR 控制算法基本原理
LQR 控制仿真

Simscape Multibody 仿真建模

Simscape搭建模型步骤
Matlab脚本文件

Webots 两轮自平衡小车仿真

搭建模型步骤
IMU
 添加InertialUnit读取代码
GYRO
 添加 Gyro 读取代码
控制代码

伺服电机驱动

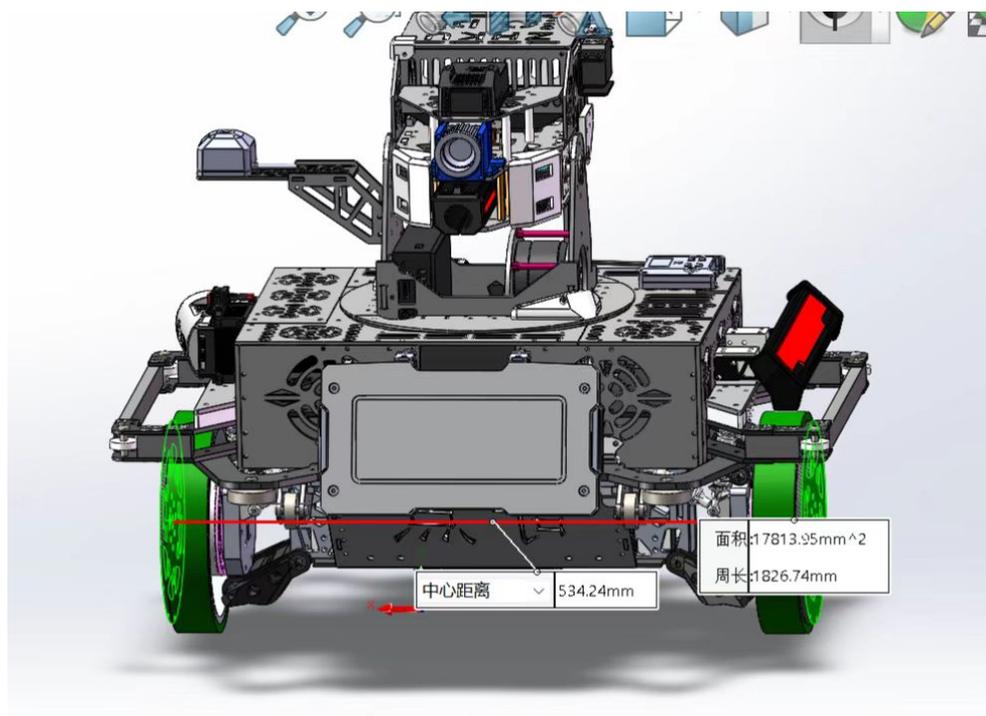
转矩闭环与实际情况分析
 参数分析
 转矩电流常量计算
驱动封装
 电机信息结构体定义
 底盘电机ID命令结构体定义
 CAN协议接收解析

版本更新说明

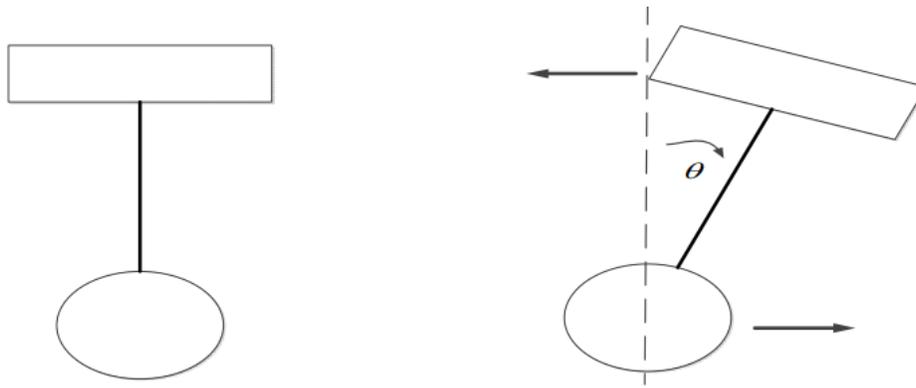
改动记录	完成日期
首次发布	2023/10/2
新增瓴控电机特性分析	2023/10/12
增加两轮自平衡动力学建模	2023/10/19
增加Webots总结	2023/10/17
增加系统状态方程以及解耦	2023/10/19
增加系统能控、能观和稳定分析	2023/10/21
增加matlab下的LQR的控制仿真	2023/10/22

引言

平衡步兵所属的两轮自平衡机器人是非常复杂的智能的控制系统，在控制过程中，通过IMU检测它的姿态信息，经过处理器处理之后会对机器人的电机施加控制，使机器人能够达到平稳状态。它的整体构造是由软件部分和硬件部分两方面组成的。软件部分就是要研究的控制系统和控制算法的设计等等。硬件部分主要是控制系统中所需要的硬件，如控制单元、驱动单元、传感器单元、电源和一些外围的电路等等。还有一部分是机械结构的系统，如用来支持控制系统的车体、车轮、支架、外壳等等。



本质上这是一种基于两轮式倒立摆控制原理的自平衡控制系统，原理上是模拟人的平衡技能。可以做前进、后退、前倾、后倾等各种运动。从侧面观察可以看到机器人平衡状态和倾斜状态。



平衡步兵机器人由车体和车轮两部分组成，机器人可以沿电机轴心转动。姿态传感器检测机器人姿态判断其是否处于倾斜状态，控制器计算出控制量并将控制量输出给驱动器，驱动电机转动使机器人保持平衡姿态。两轮机器人运动主要考虑两个问题：姿态平衡控制以及运动与转向控制。

两轮机器人通过控制左右两个电机来实现自平衡控制，当机器人后仰时，此时电机向后转动使两轮自平衡机器人重新恢复平衡状态。同样的道理，当机器人向前倾斜，此时电机正向转动使两轮自平衡机器人恢复平衡。机器人的姿态通过姿态传感器检测出来，控制控制电机的正反转，实现两轮机器人的自平衡。

对于平衡步兵这样一个系统，原理与倒立摆相似，它是一个不稳定的系统，而且它也是非线性的系统。根据它的数学模型可以看出，在实际应用中，我们首先会想到应用 PID 控制算法到机器人上，因为 PID 控制算法原理简单，操作方便，比较容易实现，所以采用 PID 算法的较多。再有就是极点配置反馈控制，基于线性化模型，得到状态空间方程，这种控制较稳定。由此，每一个算法都会有优点，同时也会存在一定的不足。在目前来说，在对两轮自平衡机器人进行控制时，这两种方法比较常见。当然近些年来自适应控制和模糊控制算法也出现在机器人的应用中，他们具有较好的控制性和鲁棒性。

平衡步兵底盘为多变量、强耦合的不稳定系统。其控制方法主要分为自平衡控制和转向控制两部分，怎样保证在不同的状况下机器人能够自动保持平衡是控制的关键问题，且为了适应比赛时打滑处理，飞坡等情况，对控制算法要求较高，传统 PID 算法应用存在较多局限性。

最优控制作为现代控制理论的发展成果，对此系统较为适用。相较于 MPC 等控制最优算法，LQR 只需离线求解 Ricatti 方程，无需在线求解优化，计算量较小，适合嵌入式 MCU 运行，通过求得的 K 矩阵在代码里使用数组计算即可完成 LQR 算法的嵌入式端实现。平衡步兵云台仍采用串级 PID 控制。云台及底盘姿态解算使用 C 板开源程序中的 AHRS 库函数加上校准零漂，效果良好，满足控制精度要求。

参考文献以及资料汇总

文献博客

- [1]李志超. 两轮自平衡机器人LQR-模糊控制算法研究[D].哈尔滨理工大学,2014.
- [2]张慧慧,侯伯杰,高建设等.基于LQR对直线倒立摆的稳摆控制研究及实现[J/OL].机械设计与制造:1-6[2023-09-21].DOI:10.19356/j.cnki.1001-3997.20230718.007.
- [3]魏芬,王素青,邓海琴等.基于模糊控制算法的一级倒立摆控制研究[J].计算机仿真,2023,40(03):320-325.
- [4]李修宇,李金凤.两轮自平衡机器人重心偏移自适应控制研究[J].齐齐哈尔大学学报(自然科学版),2023,39(04):1-5+10.
- [5]解宝彬. 两轮自平衡机器人控制算法的研究[D].哈尔滨理工大学,2014.
- [6]乔林. 两轮自平衡机器人控制策略研究[D].哈尔滨工程大学,2019.

一阶倒立摆的PID控制和LQR控制 - 廖洽源的文章 - 知乎 <https://zhuanlan.zhihu.com/p/54071212>

【RM 2023 技术答辩分享会——南京航空航天大学RoboMaster长空御风战队】 https://www.bilibili.com/video/BV1dg411z7bR/?share_source=copy_web&vd_source=876c5a71e38d5d87451539bdf1bdd9d6

控制理论

推荐DR_CAN的视频，以下是哔哩哔哩网页

工程数学: <https://space.bilibili.com/230105574/channel/seriesdetail?sid=1569595>

动态系统的建模与分析: <https://space.bilibili.com/230105574/channel/seriesdetail?sid=1569598>

自动控制理论: <https://space.bilibili.com/230105574/channel/seriesdetail?sid=3357900>

现代控制理论: <https://space.bilibili.com/230105574/channel/seriesdetail?sid=1569601>

Simscape

【matlab的simscape对PID控制下的倒立摆模型建模】 https://www.bilibili.com/video/BV1BL4y1J79y/?share_source=copy_web&vd_source=876c5a71e38d5d87451539bdf1bdd9d6

【倒立摆控制仿真】 https://www.bilibili.com/video/BV1ro4y1K73a/?share_source=copy_web&vd_source=876c5a71e38d5d87451539bdf1bdd9d6

【小车倒立摆最优控制教程 - Part2 动态推导, LQR控制器设计以及在Simulink Simscape仿真实验验证】 https://www.bilibili.com/video/BV1AF411R7T6/?share_source=copy_web&vd_source=876c5a71e38d5d87451539bdf1bdd9d6

【LQR倒立摆 从建模到控制 零基础都能复现】 https://www.bilibili.com/video/BV1yU4y1J7dT/?share_source=copy_web&vd_source=876c5a71e38d5d87451539bdf1bdd9d6

【小车倒立摆的仿真】 https://www.bilibili.com/video/BV1uP4y1T7xi/?share_source=copy_web&vd_source=876c5a71e38d5d87451539bdf1bdd9d6

Webots

<https://cyberbotics.com/doc/reference/menu?tab-language=c++>

<https://cyberbotics.com/doc/guide/robots?version=R2021a>

【Webots Full Courses for beginners -超详细入门教程 (2020)】 https://www.bilibili.com/video/BV11V411f7ko/?p=5&share_source=copy_web&vd_source=876c5a71e38d5d87451539bdf1bdd9d6

【实例 手把手搭建Webots四足机器人 (机器狗)】 https://www.bilibili.com/video/BV1u44y1f7oU/?share_source=copy_web&vd_source=876c5a71e38d5d87451539bdf1bdd9d6

【【Webots入门教程-1用户界面(UI)】 Webots用户界面介绍和基本使用】 https://www.bilibili.com/video/BV1f1F411F7D9/?share_source=copy_web&vd_source=876c5a71e38d5d87451539bdf1bdd9d6

CSDN教程: https://blog.csdn.net/crp997576280/category_9855084.html

Webots控制器函数详解 - 追梦小公子的文章 - 知乎<https://zhuanlan.zhihu.com/p/406419561>

ADRC (了解即可)

【ADRC】跟踪微分器 - 龙猫的文章 - 知乎 <https://zhuanlan.zhihu.com/p/511275301>

【ADRC】扩张状态观测器(ESO) - 龙猫的文章 - 知乎 <https://zhuanlan.zhihu.com/p/513560110>

【ADRC】根据ADRC的思想改进PID - 龙猫的文章 - 知乎 <https://zhuanlan.zhihu.com/p/514691923>

自抗扰控制: <https://zhuanlan.zhihu.com/p/515622797>

CSDN: <http://t.csdn.cn/f1Gft>

【ADRC基本概念】 https://www.bilibili.com/video/BV1hY411c7Lq/?share_source=copy_web&vd_source=876c5a71e38d5d87451539bdf1bdd9d6

【ADRC ESO】 https://www.bilibili.com/video/BV1TY4y1r7MY/?share_source=copy_web&vd_source=876c5a71e38d5d87451539bdf1bdd9dPIF6

RM2024研发迭代记录

技术要点记录

来源	是否完成	要点
论文	否	ADRC
论文	否	模糊控制算法
论文	否	ICPA-LQR
论文	否	重心偏移自适应控制

问题记录

时间	是否解决	事项
2023-10-02	是	第一版LQR控制代码测试完成, 运动正常, 位移收敛快, 有一定的重心自适应效果。整体基本达到预期
2023-10-21	是	对车体和双轮进行完全的建模
2023-10-25	是	遥控期望速度平滑处理
2023-11-2	是	改进转向控制部分以及底盘跟随, 解决原地静止时yaw转向原地晃动问题
2023-10-25	是	对车体速度, 位移滤波或者找寻更准确的计算处理方法

经典倒立摆数学模型

倒立摆系统是一种多变量、非线性、不稳定的耦合欠驱动机械系统，被广泛地应用于控制理论的教学、实验和验证控制算法的有效性。倒立摆的稳摆控制方法在军工、航天、机器人领域和工业工程上具有很广泛的用途，如人型机器人行走过程中的平衡控制、火箭发射中的垂直度控制和卫星的姿态控制等这些类似于倒立摆的重心在上，支点在下的倒置控制问题。

数学模型

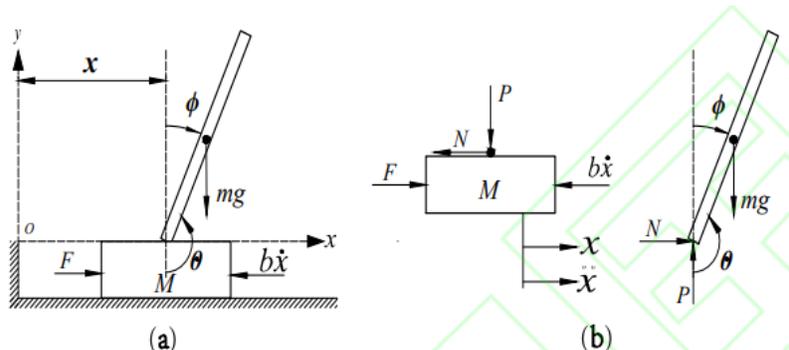


图3.1 (a) 为倒立摆系统简化模型图，图3.1 (b) 为倒立摆系统受力分析图。

在图 1 中，

字母表示	物理参数
F	施加在小车上的驱动力
x	小车的位移
M	小车的质量
m	摆杆质量
b	为小车移动的摩擦系数
I	摆杆绕其质心的转动惯量 ($I = 1/3ml^2$)
η	摆杆转动阻力矩系数
θ	摆杆与竖直向下方向的夹角
ϕ	摆杆与竖直向上方向的夹角 (逆时针为正)

依据牛顿力学与运动定理，推导直线倒立摆系统的动力学方程的过程如下：

分析小车在水平方向上受力平衡

$$M\ddot{x} + b\dot{x} + N = F \quad (1)$$

分析摆杆在水平方向上受力平衡，可得

$$N = m \frac{d^2}{dt^2} (x + l \sin \theta) \quad (2)$$

(1)、(2) 方程中的 N 为滑块与摆杆水平方向间的相互作用力, 联立 (1)、(2) 两个方程消除 N , 得到一级直线倒立摆的第一个非线性动力学模型方程

$$(M + m)\ddot{x} + b\dot{x} + ml\ddot{\theta}\cos\theta - ml\dot{\theta}^2\sin\theta \quad (3)$$

分析摆杆在竖直方向上受力平衡, 可得

$$P - mg = m\frac{d^2}{dt^2}(l\cos\theta) \quad (4)$$

分析摆杆绕其质心的力矩平衡, 可得

$$I\ddot{\theta} = -Pl\sin\theta - nlc\cos\theta - \eta\dot{\theta} \quad (5)$$

(2)、(4)、(5) 方程中的 N 、 P 为滑块与摆杆间水平、竖直方向间的相互作用力, 联立 (2)、(4)、(5) 三个方程消除 P 、 N , 得到一级直线倒立摆第二个非线性动力学模型方程

$$(ml^2 + I)\ddot{\theta} + mgl\sin\theta + \eta\dot{\theta} = -ml\ddot{x}\cos\theta \quad (6)$$

倒立摆稳摆控制是在 $-0.3\text{rad} < \phi < 0.3\text{rad}$, 此时摆角 ϕ 很小, 则有:

$$\begin{aligned} \cos\theta &= \cos(\phi + \pi) = -1 \\ \sin\theta &= \sin(\phi + \pi) = -\phi \\ \dot{\theta}^2 &= \dot{\phi}^2 = 0 \end{aligned}$$

因此, 可对倒立摆系统非线性动力学方程 (3) 与 (6) 进行线性化处理, 得到一级直线倒立摆系统线性化动力学方程

$$\begin{aligned} (I + ml^2)\ddot{\phi} - mgl\phi + \eta\dot{\phi} &= ml\ddot{x} \\ (M + m)\ddot{x} + b\dot{x} - ml\ddot{\phi} &= F \end{aligned} \quad (7)$$

记状态向量

$$X = [x \quad \dot{x} \quad \phi \quad \dot{\phi}]^T$$

与 $u=F$

根据方程组 (7) 可得到一级直线倒立摆系统的状态空间方程如下:

$$\begin{aligned} \dot{X} &= AX + Bu \\ y &= CX + Du \end{aligned} \quad (8)$$

式中:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{-4b}{4M+m} & \frac{3mg}{4M+m} & \frac{3\eta}{4M+m} \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-3b}{4M+ml} & \frac{3(M+m)g}{4M+ml} & \frac{-3(M+m)\eta}{4Ml^2+m^2l^2} \end{bmatrix} \quad (9)$$

$$B = \begin{bmatrix} 0 & \frac{4}{4M+m} & 0 & \frac{3}{4Ml+ml} \end{bmatrix}^T \quad (10)$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad (11)$$

$$D = [0 \quad 0]^T \quad (12)$$

对自主设计的平衡步兵结构进行测量，得到系统的物理参数及物理意义见表

符号	物理意义	单位	数值
M	小车质量	kg	1.6
m	摆杆质量	kg	15
l	摆杆质心到转轴距离	m	0.15
g	重力加速度	m's^-2	9.8
b	小车运动摩擦力系数		0.1
η	摆杆旋转阻力矩系数		0.01

把表 1 中倒立摆系统物理参数代入式 (9) 与式 (10) (使用matlab计算)

```

clc
m = 1.6;    %车轮的质量
M = 15;    %车体的质量
l = 0.15;  %摆杆质心到转轴距离
g = 9.8;   %重力加速度
b = 0.1;   %小车运动摩擦力系数
n = 0.01;  %摆杆旋转阻力矩系数

A = [0 1 0 0;
     0 -4*b/(4*M+m) 3*m*g/(4*M+m) 3*n/(4*M+m);
     0 0 0 1;
     0 -3*b/(4*M*l+m*l) 3*g*(M+m)/(4*M*l+m*l) -3*n*(M+m)/(4*M*l+m*m*l)]

B = [0 4/(4*M+m) 0 3/(4*M*l+m*l)].'

C = [1 0 0 0;
     0 0 1 0]

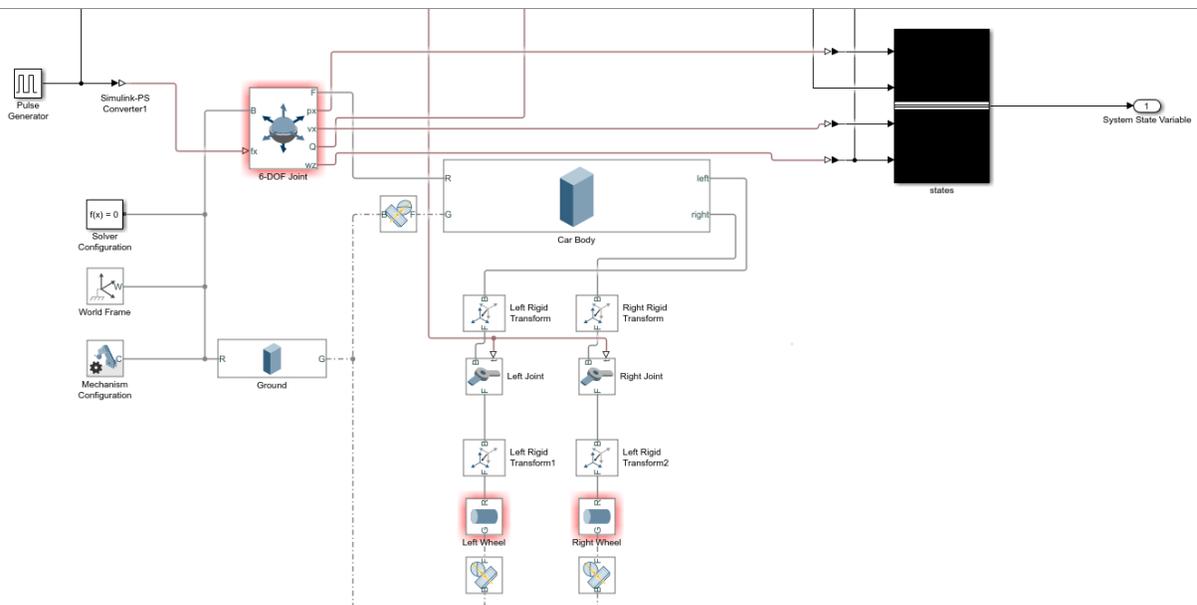
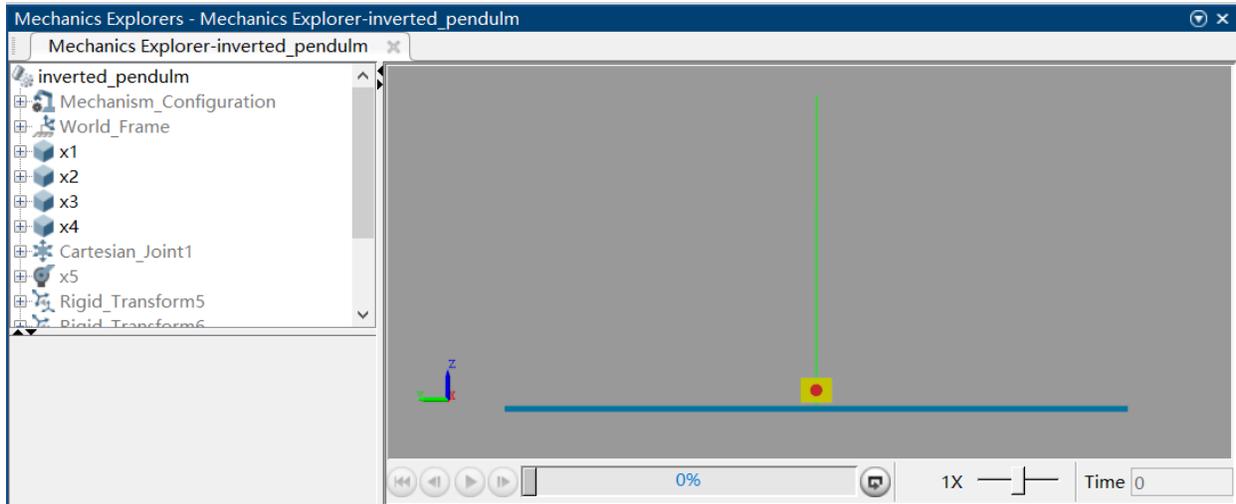
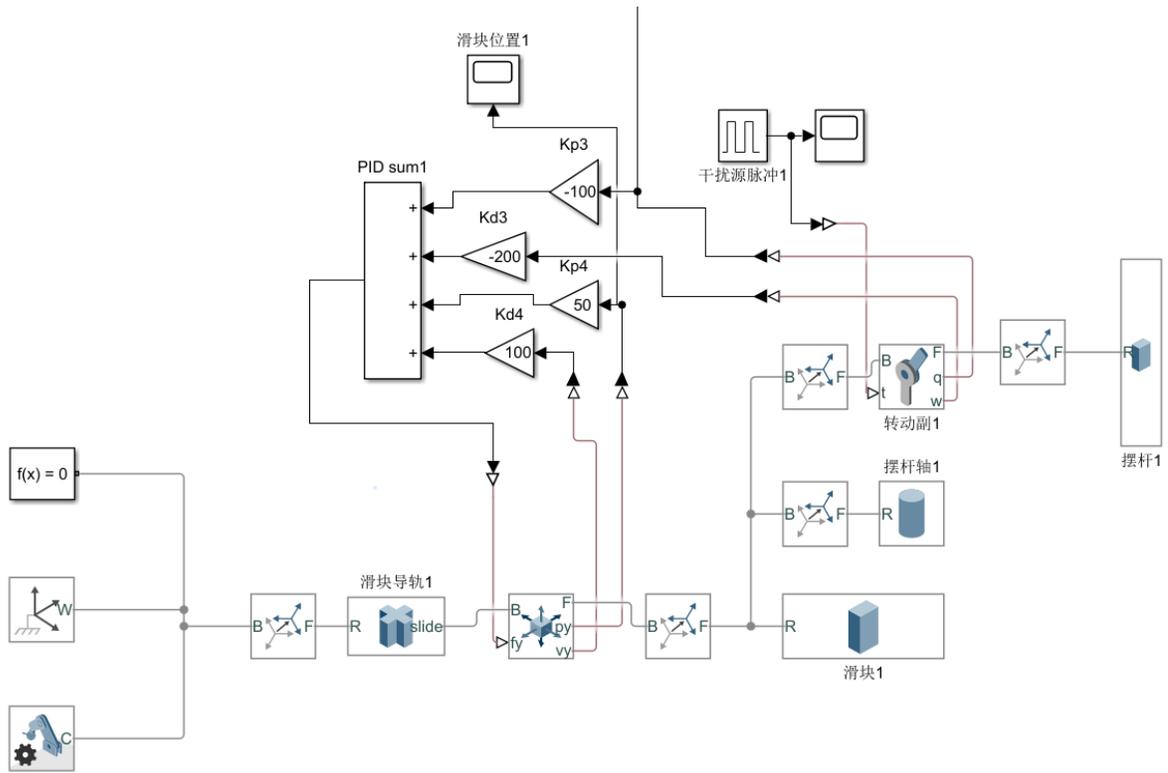
D = [0 0].'
```

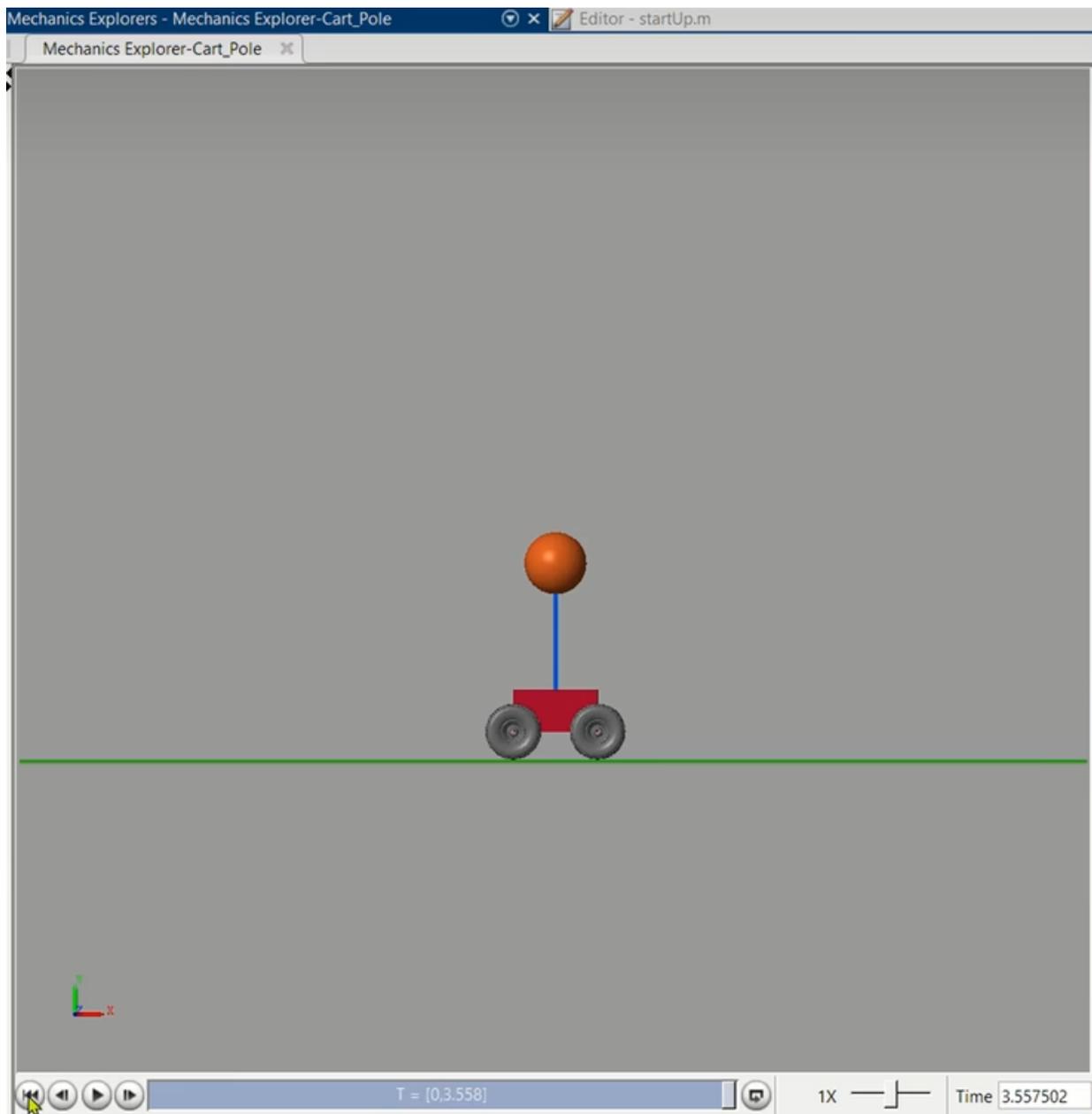
得到倒立摆系统矩阵 A，输入矩阵 B 如下：

$$A = \begin{bmatrix} 0 & 1.0000 & 0 & 0 \\ 0 & -0.0065 & 0.7636 & 0.0005 \\ 0 & 0 & 0 & 1.0000 \\ 0 & -0.0325 & 52.8182 & -0.3538 \end{bmatrix} \quad (13)$$

$$B = [0 \quad 0.0649 \quad 0 \quad 0.3247]^T \quad (14)$$

Simscape 仿真建模





平衡步兵动力学模型

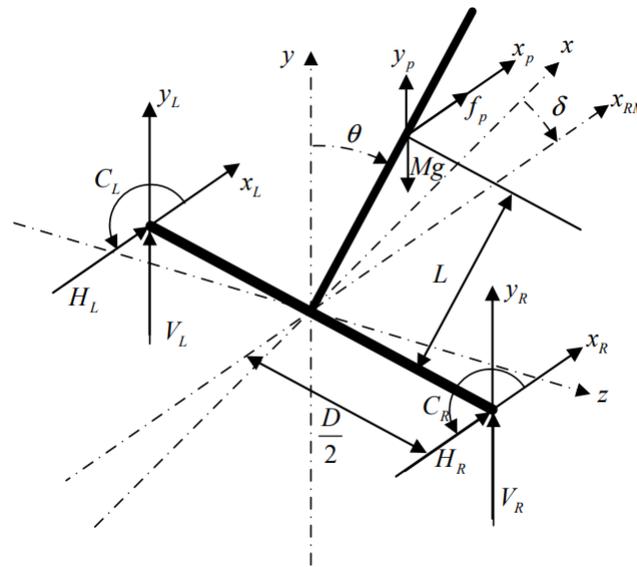
机器人的动力学和运动学模型是实现机器人控制策略的基础。想要完成对两轮自平衡机器人控制系统的设计，需要根据两轮自平衡机器人的运动特点和结构特点进行分析，建立准确可靠的数学模型。机器人动力学建模有两种具有代表性的方法：牛顿力学法和拉格朗日函数法。

拉格朗日函数法依据 Hamilton 原理，利用标量代替矢量，对总动量和总势能进行分析，建立动力学模型。这种方法运用能量方式建模，不需要对内力进行分析。

牛顿力学法运用牛顿定律和动量矩定理对各部分刚体的受力情况进行隔离分析，然后建立相邻刚体间的内力项，最终得到系统的动力学模型。牛顿力学建模法可以表达出系统完整的受力关系，有明确的物理意义，该方法建立的模型易于被控对象控制策略的设计。本节针对两轮机器人采用牛顿力学法建立动力学模型。

由于机器人不是线性的，并且具有不稳定的性质，而且具有一定耦合性的系统。机器人的构造结构和它的运动的方式是很复杂的，这样难以精确地去建立其数学模型，所以为了简化难度去分析系统，建立可行性的近似的系统模型，在一定范围内，我们允许忽略掉系统的弹性误差、信号干扰、机体和车轮之间的作用等。为了简化难度去分析系统，做了一些简化，简化的思想如下：

1. 在机器人运动的过程中不会发生跳跃，也就是不会离开地面，左右轮不会产生滑动，无论是左右滑动还是前后滑动，只能是滚动；
2. 使用机器人时，电机会有转动的摩擦，电机内部会有电感，电机在不加负载的情况下产生的阻碍转矩，这些我们都要忽略。在这种情况下电机的转矩就是电磁转矩；
3. 使用机器人时，齿轮间空隙和倾角仪、陀螺仪等引起的噪声都不关心；
4. 只关心由摩擦产生的力和力矩，忽略其它



平衡步兵机器人系统车体重心位于两轮转轴轴线之上，若不对其进行任何控制，那么机器人车体将会向前或向后倾倒。为了保护机器人，在旁边安装了保护机器人的支架，安装与机器人本体的夹角大约为 25° ，转化为弧度即为 0.43rad 。

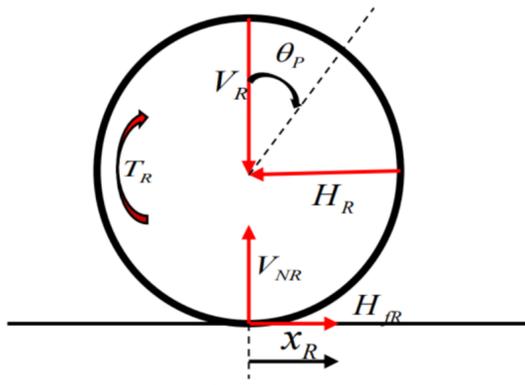
平衡步兵机器人类似于倒立摆结构，由左右两个驱动轮和车体两部分组成。左右两轮由电机独立驱动并且在同一轴线上。如图所示建立两轮自平衡机器人空间直角坐标系。以两轮轮轴中心为坐标原点，机器人水平移动方向为 x 轴， z 轴为机器人的两轮轴向方向并从左轮指向右轮， y 轴为经过两轮轴中心点的竖直向上方向，坐标系满足右手法则。

其底盘结构主要由车体和双轮两部分组成，可以看成是一个移动的倒立摆。下面分别对平衡步兵的车轮和车体进行力学分析，建立动力学模型，最后，通过对两者的分析给出系统的状态空间表达式。

车轮模型

两轮机器人左右两轮受力分析图如图所示，机器人水平方向的合力即车轮与地面的摩擦力以及车身与车轮的水平作用力的矢量和。

平衡步兵的运动是通过车轮转动来实现的，我们选用的是一对同轴安装，参数(质量、转动惯量、半径)相同的车轮。以右轮为例进行受力分析：



车轮的运动可分解为平动和转动，则由牛顿第二定律可得

$$m\ddot{x}_R = H_{fR} - H_R \quad (1)$$

由刚体定轴转动定律可得

$$I\dot{\omega}_R = T_R - H_{fR}R \quad (2)$$

式中：

公式量	意义
m	左轮、右轮各自的质量(kg)
r	左轮、右轮各自的半径(m)
x_L, x_R	左轮、右轮的水平位移(m)
x	车体中心的水平位移(m)
H_{fL}, H_{fR}	左轮、右轮受到地面的摩擦力的大小(N)
H_L, H_R	左轮、右轮受到车体作用力的水平分力的大小(N)
T_L, T_R	左轮、右轮电机输出转矩的大小(N·m)
I	车轮的转动惯量($\text{kg} \cdot \text{m}^2$)
ω_L, ω_R	左轮、右轮的角速度的大小(rad/s)

联立(1)和(2)，消去 H_{fL} ，可得

$$m\ddot{x}_R = \frac{T_R}{r} - \frac{I\dot{\omega}_R}{r} - H_R \quad (3)$$

在车轮不打滑的情况下，车轮移动速度的大小和转动速度的大小成比例关系，即

$$\begin{cases} \omega_R = \frac{\dot{x}_R}{r} \\ \dot{\omega}_R = \frac{\ddot{x}_R}{r} \end{cases} \quad (4)$$

将方程(4)代入(3)中，可得

$$\left(m + \frac{I}{r^2}\right)\ddot{x}_R = \frac{T_R}{r} - H_R \quad (5)$$

由于左右轮参数相同，则对左轮也可以得到相似的结果，即

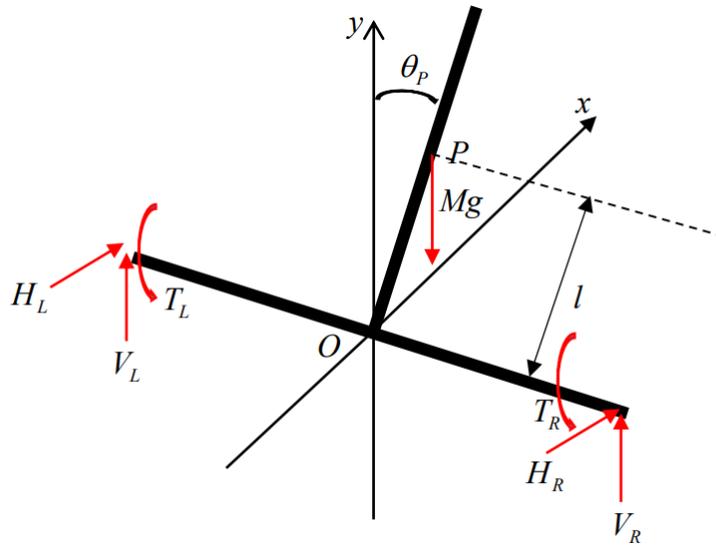
$$\left(m + \frac{I}{r^2}\right)\ddot{x}_L = \frac{T_L}{r} - H_L \quad (6)$$

车体模型

与车轮的运动类似，车体的运动也可以分解为正向运动（前向、俯仰）和侧向运动（转向、偏航）。其中，偏航运动可以看成是转向运动的特殊情况，因此，主要分析车体的正向运动和转向运动。

正向运动

为了易于分析，对车体模型进行简化，简化后的模型如图所示



小车的正向运动可以分解为前向运动和绕车体质心 P 的相对转动（俯仰）。小车底盘中心 O 的水平位移为

$$x = \frac{x_L + x_R}{2} \quad (7)$$

将方程(5)和(6)相加后，等式两边除以 2 可得

$$\left(m + \frac{I}{r^2}\right)\frac{\ddot{x}_L + \ddot{x}_R}{2} = \frac{T_L + T_R}{2r} - \frac{H_L + H_R}{2} \quad (8)$$

联立方程(7)(8)可得

$$\left(m + \frac{I}{r^2}\right)\ddot{x} = \frac{T_L + T_R}{2r} - \frac{H_L + H_R}{2} \quad (9)$$

对车体，由牛顿第二定律可得

在水平方向上，有

$$M\frac{d(x + l\sin\theta_p)}{dt^2} = H_L + H_R \quad (10)$$

在竖直方向上, 有

$$M \frac{d(l \cos \theta_p)}{dt^2} = V_L + V_R - Mg \quad (11)$$

对车体, 由刚体定轴转动定律可得

$$J_p \ddot{\theta}_p = (V_L + V_R) l \sin \theta_p - (H_L + H_R) l \cos \theta_p - (T_L + T_R) \quad (12)$$

式中

公式量	意义
M	整个机器人的总质量(kg)
l	机器人机体重心到 z 轴的距离(m)
Jp	车体绕质心转动时的转动惯量(俯仰)(kg · m ²)=(1/3)*Ml ²
θp	机器人机体与 y 轴的夹角(rad)

联立方程(9)(10)

$$\left(M + 2m + \frac{2I}{r^2} \right) \ddot{x} - \frac{T_L + T_R}{2} + Ml \ddot{\theta}_p \cos \theta_p - Ml \dot{\theta}_p^2 \sin \theta_p = 0 \quad (13)$$

方程(13)就是机器人的非线性数学模型, 根据牛顿力学方程得到的。

目前, 由于对非线性控制系统没有一个实用的成熟的非线性控制理论来分析它的性能, 所以我们的目标转向了线性控制理论, 因为它的应用已经十分广泛, 并且已经有着有一套十分成熟的科学体系, 能够为我们解决非线性系统提供帮助。因此, 我们要用线性化之后的系统模型去代替原来的非线性系统的模型, 然后利用线性分析的成熟的体系简化非线性系统的模型, 降低解决非线性系统复杂模型的难度, 但是还是要在合理的效果之内, 若是简化之后不能够达到控制要求, 那么简化就是失去了意义。根据线性化数学模型设计出来的控制器应用在原来的非线性数学模型中, 理论上都会有比较好的控制效果, 这样就简化了非线性系统的难度, 为解决非线性系统遇到的问题提供了一种可行性方案。

下面是如何简化系统的非线性系统模型, 具体的线性化过程如下:

在平衡点周围可以近似的认为有 $\theta \approx 0$, 则有 $\sin \theta \approx \theta, \cos \theta \approx 1$, 则有

$$\begin{cases} \cos \theta_p = 1 \\ \sin \theta_p = \theta_p \\ \dot{\theta}_p^2 = 0 \end{cases}$$

故方程(13)变为

$$\ddot{x} = \frac{T_L + T_R}{\left(M + 2m + \frac{2I}{r^2} \right) r} - \frac{Ml}{\left(M + 2m + \frac{2I}{r^2} \right)} \ddot{\theta}_p \quad (14)$$

将方程(10)和(11)代入方程(12)中, 可得

$$\left(\frac{J_p}{Ml} + l \right) \ddot{\theta}_p + \ddot{x} \cos \theta_p - g \sin \theta_p + \frac{T_L + T_R}{Ml} = 0 \quad (15)$$

类似的，对方程(15)进行线性化可得

$$\ddot{\theta}_p = \frac{Mlg}{(J_p + Ml^2)}\theta_p - \frac{Ml}{(J_p + Ml^2)}\ddot{x} - \frac{T_L + T_R}{(J_p + Ml^2)} \quad (16)$$

将方程(16)代入方程(14)中，消去 $\ddot{\theta}_p$ 可得

$$\ddot{x} = -\frac{M^2l^2g}{Q_{eq}}\theta_p + \frac{J_p + Ml^2 + Mlr}{Q_{eq}r}(T_L + T_R) \quad (17)$$

将方程(14)代入方程(16),消去 \ddot{x} ,可得

$$\ddot{\theta}_p = \frac{Mlg\left(M + 2m + \frac{2I}{r^2}\right)}{Q_{eq}}\theta_p - \frac{\left(\frac{Ml}{r} + M + 2m + \frac{2I}{r^2}\right)}{Q_{eq}}(T_L + T_R) \quad (18)$$

综上所述，对于正向运动有

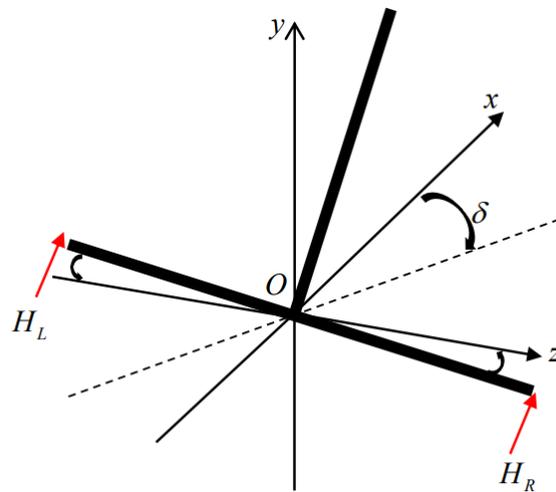
$$\begin{cases} \ddot{x} = -\frac{M^2l^2g}{Q_{eq}}\theta_p + \frac{J_p + Ml^2 + Mlr}{Q_{eq}r}(T_L + T_R) \\ \ddot{\theta}_p = \frac{Mlg\left(M + 2m + \frac{2I}{r^2}\right)}{Q_{eq}}\theta_p - \frac{\left(\frac{Ml}{r} + M + 2m + \frac{2I}{r^2}\right)}{Q_{eq}}(T_L + T_R) \end{cases} \quad (19)$$

式中

$$Q_{eq} = J_pM + (J_p + Ml^2)\left(2m + \frac{2I}{r^2}\right)$$

转向运动

与正向运动类似，我们也可以建立简化后的转向运动模型，如图所示



转向运动是由于左右两车轮从水平方向上施加给车体的反作用力的大小 H_L 和 H_R 不相等引起的，则由刚体定轴转动定律可得

$$J_\delta \ddot{\delta} = \frac{d}{2}(H_L - H_R) \quad (20)$$

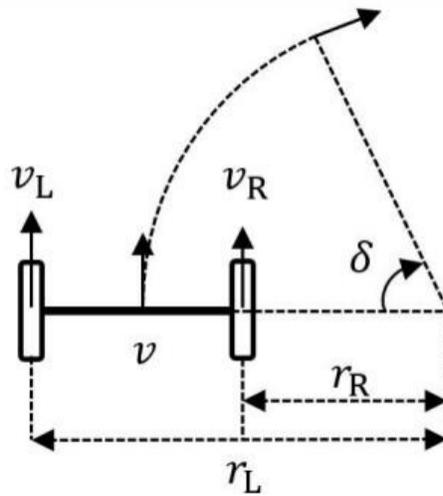
式中

公式量	意义
d	左轮、右轮两个轮子间的距离(m)
$J\delta$	车体绕 y 轴转动时的转动惯量($\text{kg} \cdot \text{m}^2$)= $(1/12)*MD^2$
δ	机器人机体与 x 轴的夹角(小车的偏航角)(rad)

将方程(5)和(6)相减后可得

$$\left(m + \frac{I}{r^2}\right)(\ddot{x}_L - \ddot{x}_R) = \frac{T_L - T_R}{r} - (H_L - H_R) \quad (21)$$

当左右两轮运动速度不相等时，小车身转向，如图所示。由几何关系可得



位移的表达式

$$x = \frac{x_L + x_R}{2}$$

转角的表达式

$$\delta = \frac{x_L - x_R}{d}$$

并且

$$\begin{cases} x_L = r\theta_l \\ x_R = r\theta_r \end{cases}$$

得到

$$\begin{cases} \dot{x}_L = \dot{\delta}r_L \\ \dot{x}_R = \dot{\delta}r_R \\ r_L = r_R + d \end{cases} \quad (22)$$

解得

$$\dot{\delta} = \frac{\dot{x}_L - \dot{x}_R}{d} \quad (23)$$

对方程(23)两边同时求导得

$$\ddot{\delta} = \frac{\ddot{x}_L - \ddot{x}_R}{d} \quad (24)$$

联立方程(20)(21)(24)可得

$$\ddot{\delta} = \frac{1}{r \left(md + \frac{Id}{r^2} + \frac{2J_\delta}{d} \right)} (T_L - T_R) \quad (25)$$

系统状态方程以及解耦

由方程(19)和(25)可得系统的状态方程为

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta}_p \\ \ddot{\theta}_p \\ \dot{\delta} \\ \ddot{\delta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & A_{23} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & A_{43} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \theta_p \\ \dot{\theta}_p \\ \delta \\ \dot{\delta} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ B_{21} & B_{22} \\ 0 & 0 \\ B_{41} & B_{42} \\ 0 & 0 \\ B_{61} & B_{62} \end{bmatrix} \begin{bmatrix} T_L \\ T_R \end{bmatrix} \quad (26)$$

由方程(19)(25)可得，矩阵中的元素为

$$A_{23} = -\frac{M^2 l^2 g}{Q_{eq}}$$

$$A_{43} = \frac{M l g \left(M + 2m + \frac{2I}{r^2} \right)}{Q_{eq}}$$

$$B_{21} = \frac{J_p + M l^2 + M l r}{Q_{eq} r}$$

$$B_{22} = \frac{J_p + M l^2 + M l r}{Q_{eq} r}$$

$$B_{41} = -\frac{\left(\frac{M l}{r} + M + 2m + \frac{2I}{r^2} \right)}{Q_{eq}}$$

$$B_{42} = -\frac{\left(\frac{M l}{r} + M + 2m + \frac{2I}{r^2} \right)}{Q_{eq}}$$

$$B_{61} = \frac{1}{r \left(md + \frac{Id}{r^2} + \frac{2J_\delta}{d} \right)}$$

$$B_{62} = -\frac{1}{r\left(md + \frac{Id}{r^2} + \frac{2J_\delta}{d}\right)}$$

由机器人系统模型的状态方程可知系统输入为左右两轮控制扭矩，是双输入系统，为了方便分析与控制器的设计，现在把该系统解耦成为平衡控制和转向控制两个单输入的系统，根据两轮机器人运动学模型可知：

$$\begin{bmatrix} T_L \\ T_R \end{bmatrix} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & -0.5 \end{bmatrix} \begin{bmatrix} T_\theta \\ T_\delta \end{bmatrix} \quad (27)$$

注意：这里可推出TL=0.5Tθ,即Tθ=2TL

由方程(26)和(27)可得：

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta}_p \\ \ddot{\theta}_p \\ \dot{\delta} \\ \ddot{\delta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & A_{23} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & A_{43} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \theta_p \\ \dot{\theta}_p \\ \delta \\ \dot{\delta} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ B_{21} & 0 \\ 0 & 0 \\ B_{41} & 0 \\ 0 & 0 \\ 0 & B_{62} \end{bmatrix} \begin{bmatrix} T_\theta \\ T_\delta \end{bmatrix} \quad (28)$$

分解上式可以得到平衡子系统和转向子系统，并且两者之间是相互独立的

平衡子系统：

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta}_p \\ \ddot{\theta}_p \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & A_{23} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & A_{43} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \theta_p \\ \dot{\theta}_p \end{bmatrix} + \begin{bmatrix} 0 \\ B_2 \\ 0 \\ B_4 \end{bmatrix} [T_\theta] \quad (29)$$

假定 $T_l = T_r = T_{lr}$ ，替换 T_l ， T_r ，为 T_{lr} 可以得到机器人的二自由度的线性数学模型为：

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta}_p \\ \ddot{\theta}_p \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & A_{23} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & A_{43} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \theta_p \\ \dot{\theta}_p \end{bmatrix} + \begin{bmatrix} 0 \\ 2B_2 \\ 0 \\ 2B_4 \end{bmatrix} [T_l] \quad (30)$$

式中

公式量	意义
T_l, T_r	二自由度子系统 1 左边、右边电机控制时的各自输出转矩

以速度和倾角为输出，那么输出方程为：

$$y = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \theta_p \\ \dot{\theta}_p \end{bmatrix} \quad (31)$$

下面的研究对象主要就是这个二自由度数学模型

转向子系统:

$$\begin{bmatrix} \dot{\delta} \\ \ddot{\delta} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \delta \\ \dot{\delta} \end{bmatrix} + \begin{bmatrix} 0 \\ B_{62} \end{bmatrix} [T_{\delta}] \quad (32)$$

以偏航角为输出, 那么输出方程为:

$$y = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \delta \\ \dot{\delta} \end{bmatrix} \quad (33)$$

注意: 方程(26)中的B62=-B61, 方程(28)和(32)中的B62=B61

从上面的方程(29)和(32)可知, 原来的系统是有两个输入, 经过处理后可以得到两个子系统, 方程(29)为子系统1, 这个系统是用 T_{θ} 控制机器人的位移 x 和倾角 θ , T_{θ} 即是系统1的相应的输入转矩。同理方程(32)为子系统2, 这个系统用 T_{δ} 控制机器人的转角 δ 。

平衡步兵模型系统分析

稳定性分析

系统稳定性指的是系统受到外界干扰而偏离原来的状态, 当去掉扰动后系统可以恢复到平衡状态的一种能力。稳定性是平衡机器人系统重要特性之一。对于任何一个系统, 都要先考虑系统是否是稳定系统, 对于稳定系统要怎么样去提高系统的稳定性, 能够更好的稳定。对于不稳定系统怎样去控制使得系统能够稳定。

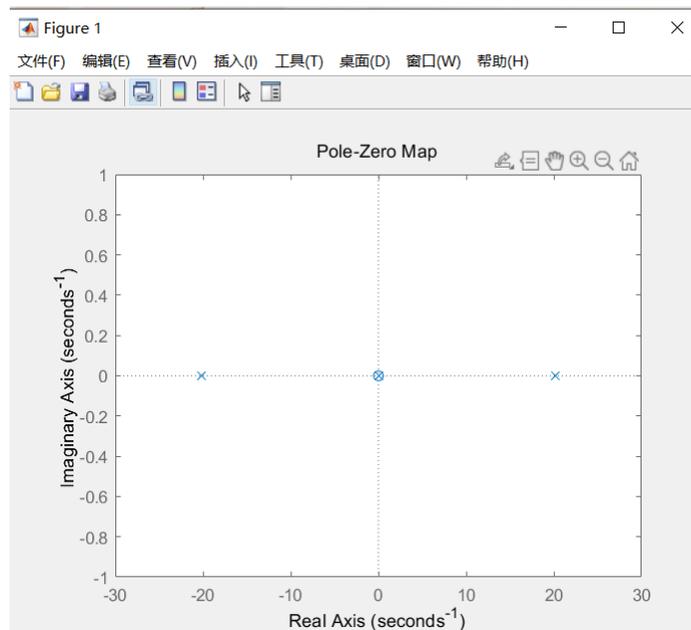
根据李雅普诺夫第一法则, 在线性定常系统中, 如果系统矩阵A的所有特征值都具有负实部, 那么系统的平衡状态都是渐近稳定的。由上面方程 (30), 可知系统的状态矩阵为

A =

$$\begin{bmatrix} 0 & 1.0000 & 0 & 0 \\ 0 & 0 & -52.5869 & 0 \\ 0 & 0 & 0 & 1.0000 \\ 0 & 0 & 408.4246 & 0 \end{bmatrix}$$

经过计算可以得到矩阵A的特征值:[0 0 20.2095 -20.2095], 其中含有正实根, 说明系统是不稳定的, 只有对系统加以控制, 才能使其保持平稳的运动。

应用 MATLAB 函数, 根据线性方程(30)和线性方程(31)可以制作出系统的极点分布图



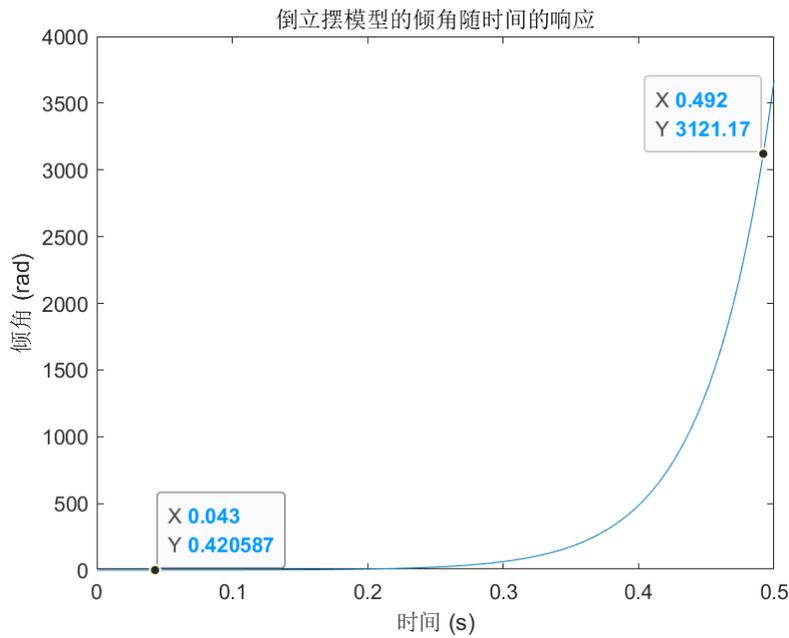
从图中可以知道，极点不仅存在于以零点为界的左边这半部分，并且在以零点为界的右半部分也存在。分析可以知道若不对机器人进行控制，它就不会自己站立，即它是个不稳定系统。所以十分有必要去设计控制器来对该系统进行控制，能够让该系统在没有外力的情况下自主的去保持到平衡稳定的状态。

利用 MATLAB 对于上述模型进行分析，当初始条件为 $x_0 = [0 \ 0 \ 0.1 \ 0]^T$ ，观察系统在 $u = 0$ ，即零输入下的响应情况：

```

%% 稳定性分析，生成极点图和系统响应
% 初始条件
x0 = [0; 0; 0.3; 0];
% 生成状态空间模型
sys = ss(A, B, C, D);
% 计算特征向量
[V, D] = eig(A); % V是特征向量矩阵，D是特征值矩阵
figure;
% 画极点分布图
pzmap(sys);
title('系统极点图');

```



两轮平衡机器人初始很小的弧度，在没有外界控制的情况下会导致系统无法稳定，从而验证了两轮机器人是个天然不稳定系统，需要增加外界的控制才能使其保持稳定。

能控性分析

对于线性连续定常系统：

$$\dot{x} = Ax + Bu$$

式中：X ----- n 维状态向量，n ----- p维输入向量，A、B-----分别为n n × 和n p × 常值矩阵。

如果在有限的时间 $t \in [t_0, t_f]$ 里，存在一个控制向量 $u(t)$ ，能够从初始状态 $x(t_0)$ 转移到指定状态 $x(t_f)$ ，则称在 t_0 时刻状态 $x(t_0)$ 能控。如果系统中所有的非零状态都为能控的，则称系统是可控的。

系统完全可控的充分必要条件是：

$$\text{rank} = [B \quad AB \quad A^2B \quad \dots \quad A^{n-1}B] = n$$

其中，n 为矩阵A的维数，若 $\text{rank}(S) = n$ ，即满秩时，系统状态能控；否则系统不可控

$$S = [B \quad AB \quad A^2B \quad \dots \quad A^{n-1}B]$$

其中，S称为系统的可控判别阵

根据系统的平衡子系统状态方程(30)，使用 MATLAB 函数 $S = \text{ctrb}(A, B)$ ，得到平衡子系统的能控性判别矩阵：

```
S =
1.0e+04 *
     0     0.0012     0     0.4369
    0.0012     0     0.4369     0
     0    -0.0083     0    -3.3929
   -0.0083     0    -3.3929     0
```

再根据 $\text{rank}(S)$ 命令求得 $\text{rank}(S) = 4$ ，能控判别矩阵满秩可知平衡子系统是能控的。

```

%% 能控性分析
Mat_ctrb=ctrb(A,B);
if rank(Mat_ctrb)==4 % 判断是否满秩，这里的系统是4阶的，于是判断是否等于4
    disp('Mat_ctrb=');
    disp(Mat_ctrb); % 打印可控性矩阵
    disp('原系统可控');
else
    disp('原系统不可控');
end

```

```

Mat_ctrb=
1.0e+04 *
    0    0.0012    0    0.4369
    0.0012    0    0.4369    0
    0   -0.0083    0   -3.3929
   -0.0083    0   -3.3929    0

```

原系统可控

能观性分析

对于线性连续定常系统：

$$\begin{aligned}\dot{x} &= Ax + Bu \\ y &= Cx\end{aligned}$$

式中：x ----- n 维状态向量，u ----- p 维输入向量，y ----- q 维输出向量，A、B、C ----- 分别为 $n \times n$ ， $p \times n$ ， $n \times q$ 常值矩阵。

对于任意的输入 $u(t)$ ，根据 $t \in [t_0, t_f]$ ($t_f > t_0$) 期间的输出 $y(t)$ 可以唯一地确定系统在初始时刻的状态 $x(t_0)$ ，则称系统在 $t \in [t_0, t_f]$ 内是能观测的。如果对于所有 $t_f > t_0$ 系统都是可观测的，那么称系统在 $[t_0, \infty)$ 完全可观测。

系统完全可观的充分必要条件是：

$$\text{rank} \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix} = n$$

其中

$$V = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix}$$

称为系统的可观判别矩阵。

在 MATLAB 中利用函数 $V=obsv(A, C)$ 命令可以得到能观性判别矩阵 V_0 ，求得 $rank(V_0)=4$ ，所以平衡子系统系统的线性化数学模型是可观测的。同样，对于转向子系统状态方程得到该系统能观性判别矩阵 V_δ ，根据 $rank(V_\delta)=2$ ，可以判断转向子系统是能观的。

```

%% 能观性分析
Mat_obsv=obsv(A,C);
if rank(Mat_obsv)==4 % 判断是否满秩，这里的系统是4阶的，于是判断是否等于4
    disp('Mat_obsv=');
    disp(Mat_obsv);
    disp('原系统可观测');
else
    disp('原系统不可观测');
end

```

```

Mat_obsv=
    1.0000         0         0         0
         0         0    1.0000         0
         0    1.0000         0         0
         0         0         0    1.0000
         0         0   -52.5869         0
         0         0   408.4246         0
         0         0         0   -52.5869
         0         0         0   408.4246

```

原系统可观测

平衡步兵控制系统设计

直到目前为止，最优控制是现代控制理论的一个十分重要的组成部分，主要研究是为了使控制系统的性能指标能够达到最优化的最基本的条件和综合的方法，是关于怎么样在一些可能的控制方式中寻找最优控制的理论。若已知被控对象的数学模型，为了实现某种需要达到的控制功能，则需要一种控制策略去实现，并且这种控制策略实现的控制对象的性能能够最好，也就是让某一项性能最大或者最小，这就是最优控制问题。最优控制的理论来源于极值原理，用最大或者最小的状态量使得其中的某一性能最优。

最优控制解决的问题为被控对象的数学模型在一定的约束条件下，被控对象按照预期要求完成任务，使得系统状态变量的性能指标达到最小。线性二次型 (Linear Quadratic) 指的是性能指标函数是状态变量和控制变量的二次型，并且被控对象的模型是线性的。LQR 控制是一种比较经典的控制方法，在实际的倒立摆系统中表现出了很好的实用性，因此将其运用在平衡机器人中是完全适用的。在 MATLAB 软件中可以方便求出最优反馈矩阵，这使得 LQR 控制器的设计更加方便。

LQR 控制算法基本原理

LQR 控制器的基本原理是：对于可控的线性时不变系统，其状态空间方程的形式为

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned} \quad (1)$$

状态反馈控制律：

$$u = -Kx \quad (2)$$

其中，A 为系统矩阵，B 为输入矩阵，C 为输出矩阵，X 为状态向量，u 为控制率，y 为输出向量。设控制率 u 不受约束寻找最优，使式 (3) 性能指标最小

$$J = \frac{1}{2} \int_0^{\infty} [X^T Q X + u^T R u] dt \quad (3)$$

其中，Q 为半正定对称常数的状态向量加权矩阵，R 正定对称常数的控制率加权矩阵

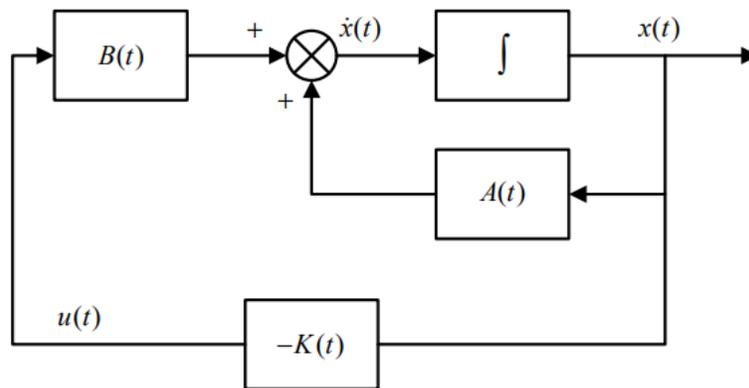
通过求解黎卡提代数方程：

$$PA + A^T P - PBR^{-1}B^T P + Q \quad (4)$$

可以得到 P 与最优状态反馈增益矩阵 K 的值，使性能指标 J 最小的控制率 u，控制率 u 为：

$$u = R^{-1}B^T P X = -K X \quad (5)$$

LQR 控制器的控制框图如图所示



Q 对角线上权值系数决定了各个指标误差的相对重要性，当 Q 矩阵中系数增加时候系统响应变快。选取不同的 Q 值，状态反馈矩阵 K 也会随之发生变化，进而系统的动态性能和稳态性能均会受到影响，权阵 Q 的取值和被控制系统的抗扰性密切相关。权阵 Q 的取值增大，则动态过程的超调量和调整时间将减小，但相应的会导致控制输入消耗的能量增加。取值过大甚至还会引起系统不稳定，所以权阵 Q 的取值应该限定在一定的范围之内。

R 矩阵系数增加时可以减少系统的输入变量，但同时会降低系统的响应速度，如果要减小系统的控制能量消耗，可以适当的增大权阵 R 的取值，但是如果权阵 R 的取值过大，会导致控制能量过小，同样不利于对系统的控制。

因此应该协调 Q 和 R 的权值得到最佳的控制方案。

对于 LQR 控制器的设计，在选取加权矩阵 Q、R 后，在 MATLAB 中调用 lqr 函数，即可得到系统的状态反馈增益矩阵 $K = \text{lqr}(A, B, Q, R)$ ，完成 LQR 控制器设计。

LQR 控制仿真

根据两轮自平衡机器人的系统的状态空间方程的线性模型，应用 MATLAB 搭建 LQR 仿真控制。

```
clear;
clc;
global K;
%% 定义小车倒立摆物理性质
R = 0.0925;    %车轮的半径
D = 0.55;     %左轮、右轮两个轮子间的距离
l = 0.15;     %摆杆质心到转轴距离
m = 0.88;     %车轮的质量
```

```

M = 13;          %摆杆质量
I = (1/2)*m*R^2; %车轮的转动惯量
Jz = (1/3)*M*l^2; %机器人机体对 z 轴的运动时产生的转动惯量(俯仰方向)
Jy = (1/12)*M*D^2; %机器人机体对 y 轴的运动时产生的转动惯量(偏航方向)
g = 9.8;        %重力加速度b = 1e-4;

%% 状态空间矩阵
Q_eq = Jz*M + (Jz+M*l*l) * (2*m+(2*I)/R^2);
% A为系统矩阵
A_23=-(M^2*l^2*g)/Q_eq;
A_43=M*l*g*(M+2*m+(2*I/R^2))/Q_eq;
A = [0 1 0 0;
     0 0 A_23 0;
     0 0 0 1;
     0 0 A_43 0];

% B为输入矩阵
B_21=(Jz+M*l^2+M*l*R)/Q_eq/R;
B_41=-((M*l/R)+M+2*m+(2*I/R^2))/Q_eq;
B = [0 ;
     2*B_21;
     0 ;
     2*B_41];

% C为输出矩阵
C = [1 0 0 0
     0 0 1 0];
D = [0;0];

%% 求LQR增益矩阵
Q = diag([10 25 50 30]); % x dx q dq
R = 10;
K = lqr(ss(A, B, C, D),Q,R)

%% LQR闭环控制
% 初始条件
x0=[0 0 0.43 0];
% 设置时间范围
t = 0:0.01:10;
% 设置输入
u =0.2*ones(size(t));
Ac = (A-B*K);
Bc = B;
Cc = C;
Dc = D;
% 生成状态空间模型
sys_close = ss(Ac,Bc,Cc,Dc);
% 对输入函数的响应
[y,t,x]=lsim(sys_close,u,t,x0);

% 绘图
figure;

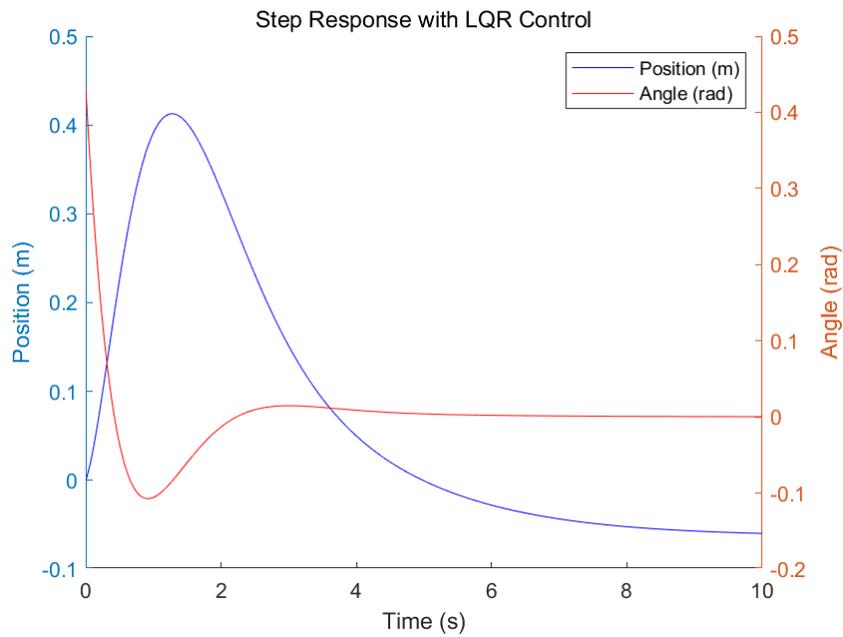
```

```

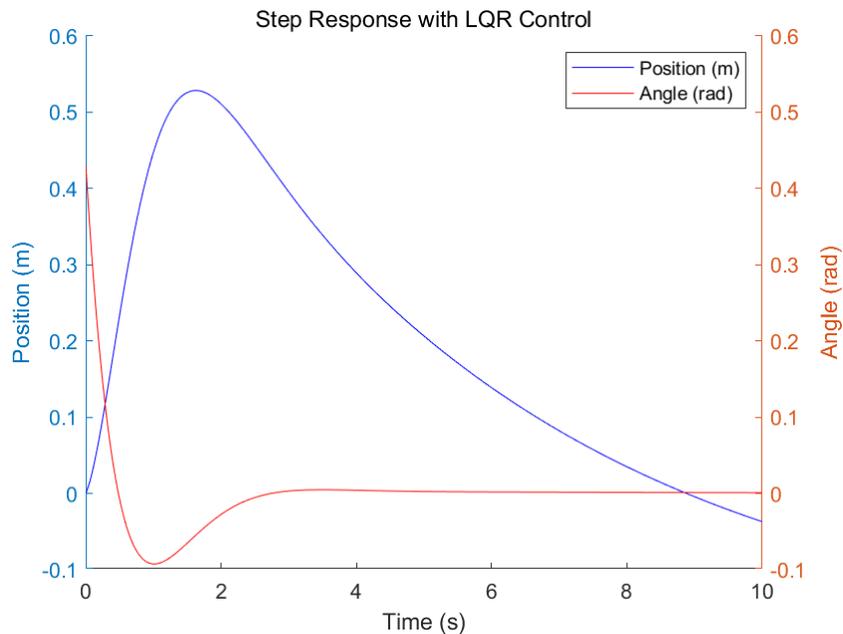
hold on;
yyaxis left;
plot(t, y(:,1), 'b', 'Linewidth', 0.5); % 位移
ylabel('Position (m)');
yyaxis right;
plot(t, y(:,2), 'r', 'Linewidth', 0.5); % 倾角
ylabel('Angle (rad)');
hold off;
legend('Position (m)', 'Angle (rad)');
xlabel('Time (s)');
title('Step Response with LQR Control');

```

首先，假设初始角度为 $\theta = 0.43\text{rad}$ ，初始状态就是为 $x_0 = [0, 0, 0.43, 0]^T$ ，最后机器人的位移和倾角控制仿真曲线如下：



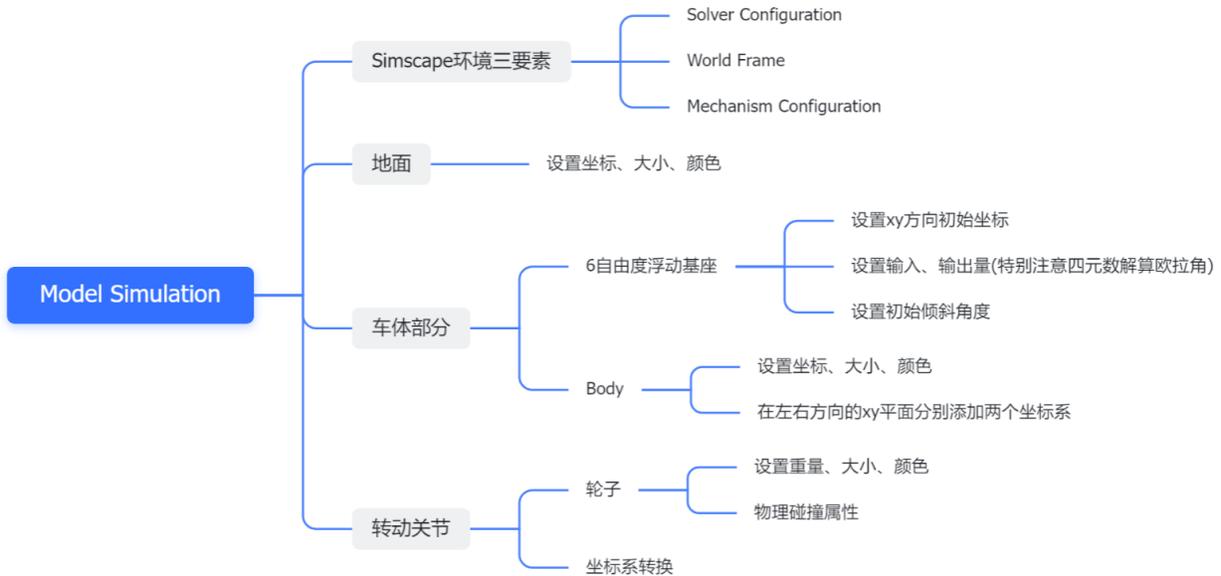
其次，增加Q矩阵中倾角控制的权重，减少位移控制的权重，最后机器人的位移和倾角控制仿真曲线如下：



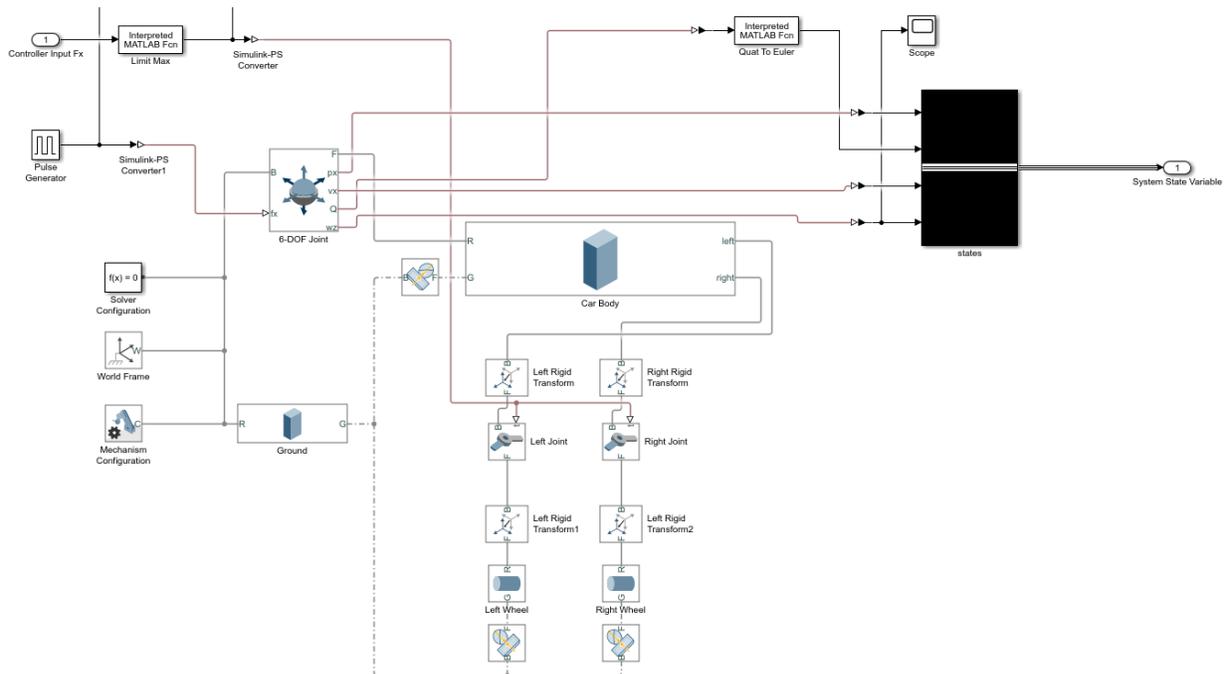
从上边的分析系统仿真实验我们可以知道，根据 LQR 控制理论设计出来的 LQR 控制对机器人系统能够让两轮自平衡机器人独立的达到平稳状态，可以看出 LQR 控制器对于机器人的控制是有效果的，能够在无外力的情况下让机器人能够自主的恢复到平衡情况。

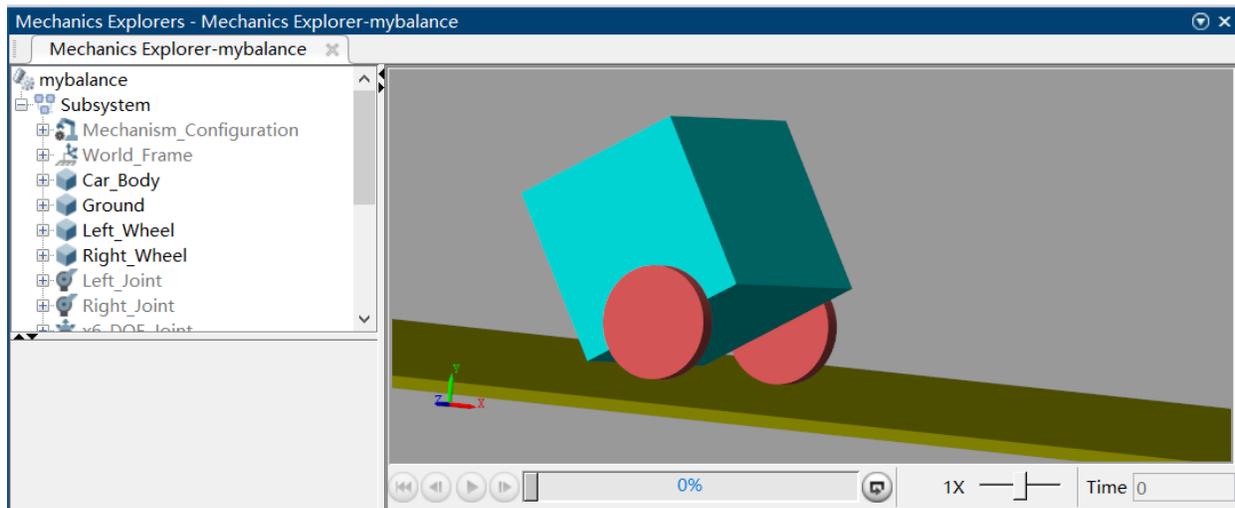
Simscape Multibody 仿真建模

Simscape可在 Simulink环境中迅速创建物理系统的模型，Simscape库中的块代表实际的物理组件。因此，可以构建复杂的多体动力学模型，而无需通过物理原理来合成数学方程。接下来总结下如何使用 Simscape Multibody的物理建模模块来建立平衡步兵模型。



Simscape搭建模型步骤





Matlab脚本文件

物理模型定义

```

clear;
clc;

%% 定义小车倒立摆物理性质
global m_wheel m M wheel_r l car_y g b n newrad gain_K Fx
wheel_r = 0.0925;    %车轮的半径
l = 0.15;            %摆杆质心到转轴距离
car_y = wheel_r+l-0.01; %simscape车体模型初始y坐标
m_wheel = 0.8;      %车轮的质量
m = m_wheel*2;      %倒立摆车体的质量
M = 15;             %摆杆质量
g = 9.8;            %重力加速度
b = 1e-4;           %小车运动摩擦系数
n = 1e-4;           %摆杆旋转阻力矩系数

%% 状态空间矩阵
A = [0 0 1 0;
     0 0 0 1;
     0 m*g/M 0 0;
     0 (m*g + M*g)/(M*l) 0 0];
B = [0;0;1/M;1/(M*l)];
C = eye(4);
D = 0;

%% LQR:
Q = diag([0.001 20 10 300]); % x q dx dq
R = 1; % fx
gain_K = lqr(A,B,Q,R)
gain_K = [0 -100 0 -2]

```

将自由度关节输出的四元数转化为欧拉角，用来获取车身倾角

```

function Euler_Angles = Quat_To_Euler(quat)
    % Rearrange the quaternion into the correct form [w, x, y, z]
    Quat = [quat(1,1), quat(2,1), quat(3,1), quat(4,1)];

    % Convert the quaternion to Euler angles using 'XYZ' order
    Euler = quat2eul(Quat, 'XYZ');

    % Extract the roll, pitch, and yaw angles from the Euler angles
    Roll = Euler(1);
    Yaw = Euler(2);
    Pitch = Euler(3);

    % Return the yaw angle as the output
    Euler_Angles = Pitch;
end

```

限幅函数

```

function out=LimitMax(input)
    out = input;
    if out>50
        out = 50;
    else if out<-50
        out = -50;
    end
end

```

LQR计算函数

```

function Fx = LQRcalculate(states)
    % Declare gain_K as a global variable
    global gain_K

    % Set the desired state
    X_des = [0; 0; 0; 0];

    % Calculate the control input using LQR control law
    Fx = gain_K * (X_des - states);
end

```

Webots 两轮自平衡小车仿真

webots是一款开源的机器人仿真软件，可以在Windows、linux、mac os 三种系统上使用。主要功能是机器人的建模、控制与仿真，用于开发、测试和验证机器人算法。

Webot支持C/C++、Python、MATLAB、Java、ROS和TCP/IP等多种方式实现模型的仿真控制。Webot内置了接近100种机器人模型，包括轮式机器人、人形机器人、爬行移动机器人、单臂移动机器人、双臂移动机器人、无人机、大狗、飞艇等等，其中就包括大家比较熟悉的Boston Dynamics Atlas、DJI Mavic 2 PRO、Nao、PR2、YouBot、UR、Turtlebot3 Burger等机器人。

官方教程: <https://cyberbotics.com/doc/guide/tutorials>

网上教程很多,这方面的介绍也很详细,所以接下来的建模过程就不赘述了,不理解的可以看看前面提到的教程学习下。

搭建模型步骤

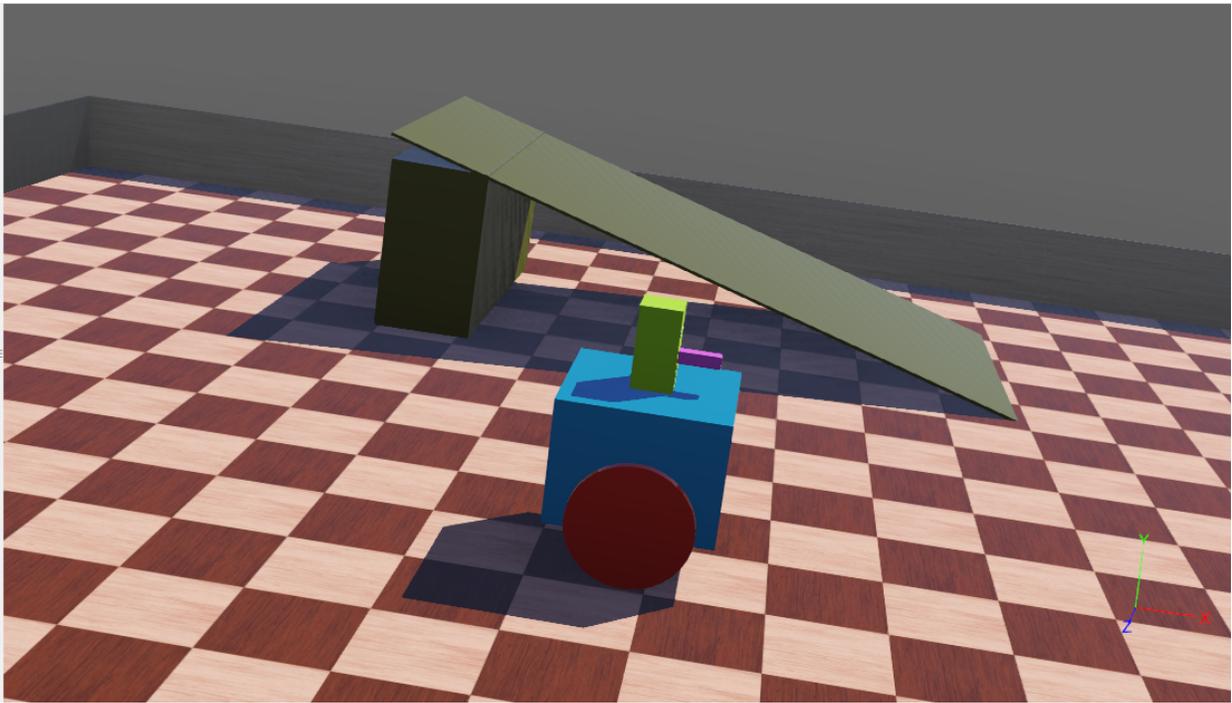
世界树结构:

- > ● WorldInfo
- > ● Viewpoint
- > ● TexturedBackground
- > ● TexturedBackgroundLight
- > ● RectangleArena "rectangle arena"
- > ● DEF BLANCE Robot
- > ● DEF RAMP Solid
- > ● DEF WALL Solid

Robot结点示意:

- ▼ ● DEF BLANCE Robot
 - translation 0.771 0.237 0.545
 - rotation 0.976 -0.156 -0.15 0.0128
 - scale 0.901 0.901 0.901
 - ▼ ■ children
 - > ● DEF EYE_L Solid
 - > ● DEF EYE_R Solid
 - > ● DEF GYRO Gyro
 - > ● DEF BMI088 InertialUnit
 - > ● DEF FILE Solid
 - > ● DEF GIMBAL Solid
 - > ● DEF joint_r HingeJoint
 - > ● DEF joint_l HingeJoint
 - > ● DEF BODY Shape
 - name "solid"

效果展示

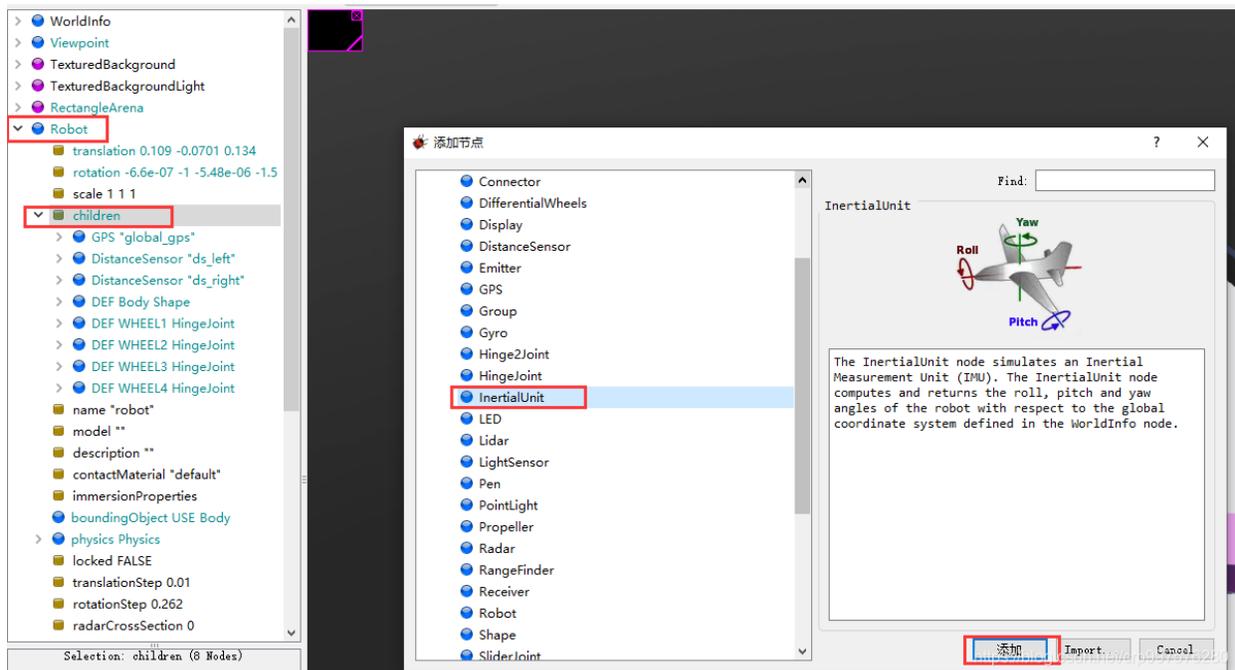


IMU

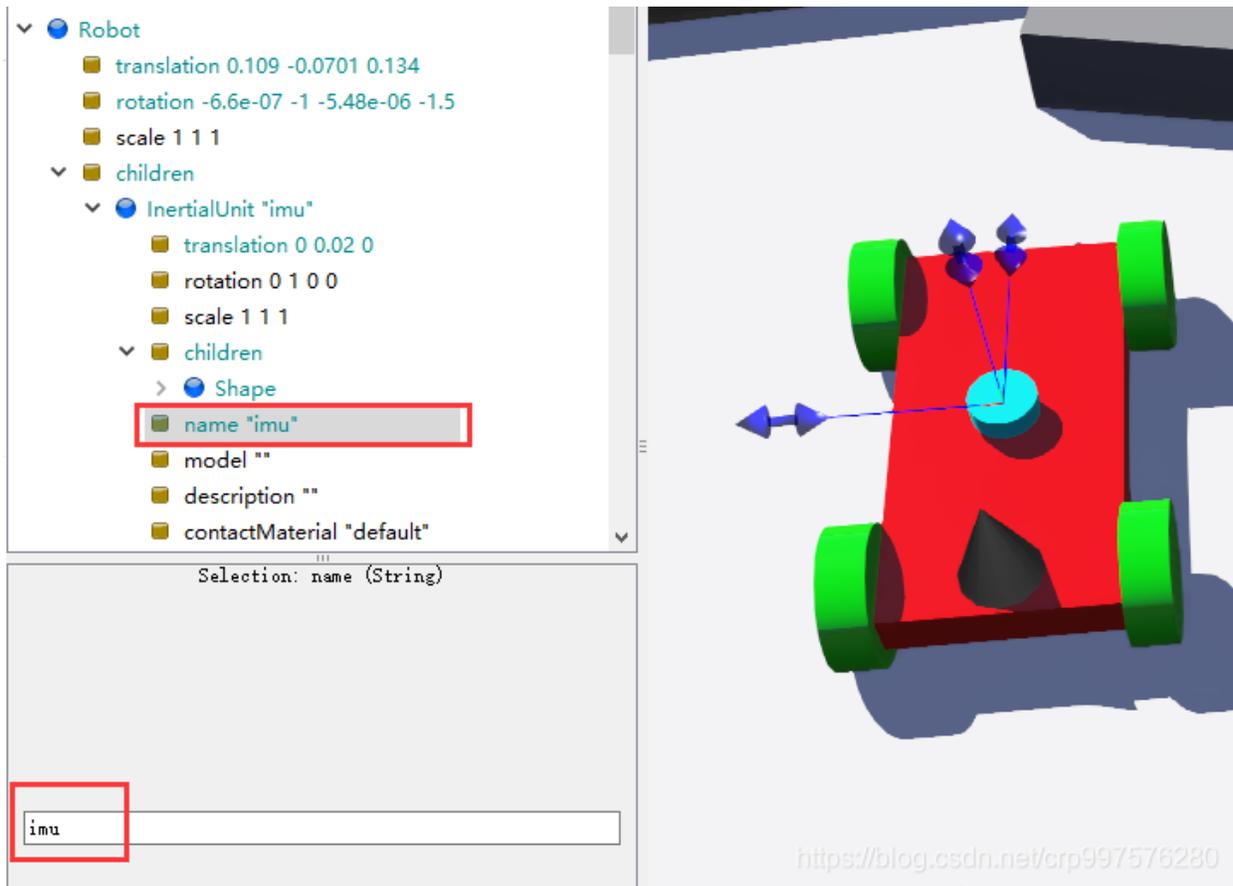
在webots中 InertialUnit 只对外提供pitch、roll、yaw三个角度，如需要加速度、陀螺仪和磁力计的数据还需要添加 Accelerometer、Gyro 和 Compass 组件。这几个传感器添加实体的方式基本相同，只有数据读取的接口差别较大，其次要注意 InertialUnit 输出的姿态角度（单位是 rad） Accelerometer 输出的加速度值（单位是 m/s^2 ）、Gyro 输出的是车体旋转速度（单位是单位是 rad/s ）。

step1: 在Robot中添加 InertialUnit（IMU传感器）

step2: 设置这个InertialUnit传感器的children中添加shape节点，并设置外观和形状



设置IMU传感器的名称（这个在控制器中会用到）



注意：添加IMU的过程中不设置boundingObject属性和 physics属性

添加InertialUnit读取代码

初始化

```
#include <webots/InertialUnit.hpp> //头文件

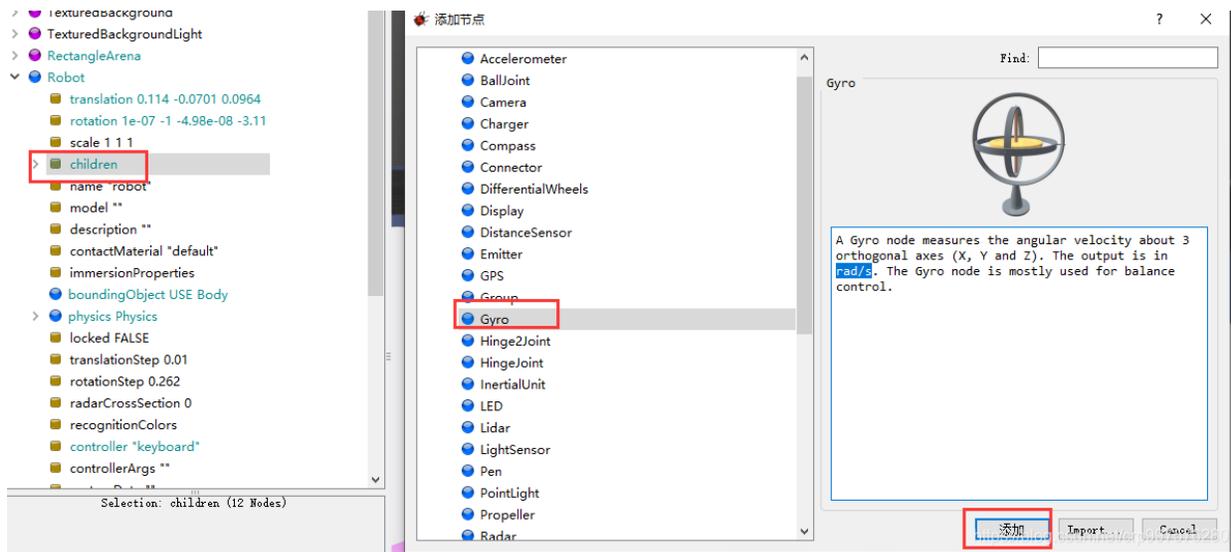
InertialUnit *imu;
imu== robot->getInertialUnit("imu");
imu->enable(TIME_STEP);
```

IMU角度数据读取代码段：（单位是弧度/**rad**）

```
std::cout<< "IMU roll: " <<imu->getRollPitchYaw()[0]<< " pitch: " <<imu->
getRollPitchYaw()[1]
<< " yaw: " <<imu->getRollPitchYaw()[2] <<std::endl;
```

GYRO

添加陀螺仪 Gyro 实体的步骤与添加 InertialUnit 实体的操作方式一样，只需要把 **step1**: 中选中Gyro 实体，后续的操作一样



添加 Gyro 读取代码

初始化代码段：

```
Gyro *gyro;
gyro=robot->getGyro("gyro");
gyro->enable(TIME_STEP);
```

陀螺仪数据读取代码段：（单位是 rad/s）

```
std::cout<< "Gyro X: " <<gyro->getValues()[0]
<< " Y: " <<gyro->getValues()[1]<< " Z: " <<gyro->getValues()[2] <<std::endl;
```

控制代码

这里我们分析webots中的基础控制器，这是由于这其中涉及到很多的基础控制函数（比如读取传感器数据、控制电机），通过了解这些基础控制器中的操作函数，为我们后面控制做基础。

官方文档：<https://cyberbotics.com/doc/reference/menu?tab-language=c++>

```
#include <stdio.h>
#include <stdlib.h>
#include "alg_pid.h"
#include <webots/Robot.hpp>
#include <webots/InertialUnit.hpp>
#include <webots/Motor.hpp>
#include <webots/Keyboard.hpp>
#include <webots/Gyro.hpp>

#define TIME_STEP 2
#define MAX_SPEED 15
using namespace webots;

PID_Loop_t angle_loop,gyro_loop,loop_angle_yaw,loop_gyro_yaw,loop_speed;
```

```

PIDElem_t angle_pitch,angle_yaw;
PIDElem_t gyro_pitch,gyro_yaw;
PIDElem_t speed_target=0,speed_real;

void pid_limitparameter_init(PID_Loop_t *pid)
{
    pid->Limit_Output = MAX_SPEED;
    pid->Limit_IntegralValue = 2;
    pid->Limit_DiffValue = 2;
}
void pid_parameter_reset(PID_Loop_t *pid, PIDElem_t new_Kp, PIDElem_t new_ki,
PIDElem_t new_kd)
{
    pid->PID_Param.Kp = new_Kp;
    pid->PID_Param.Ki = new_ki;
    pid->PID_Param.Kd = new_kd;

    if (pid->PID_Param.Ki == 0)
        pid->Iout = 0;
}
PIDElem_t pid_calculate(PID_Loop_t *pid, PIDElem_t target, PIDElem_t measure)
{
    pid->Measure = measure;
    pid->Target = target;
    pid->Error = pid->Target - pid->Measure;

    pid->Pout = pid->PID_Param.Kp * pid->Error;

    pid->ITerm = pid->PID_Param.Ki * ((pid->Error + pid->Last_Error) / 2);

    pid->Dout = pid->PID_Param.Kd * (pid->Error - pid->Last_Error);

    pid->Iout += pid->ITerm;
    if (pid->Iout > 2)
        pid->Iout = 2;
    else if (pid->Iout < -2)
        pid->Iout = -2;
    pid->Output = pid->Pout + pid->Iout + pid->Dout;

    pid->Last_Measure = pid->Measure;
    pid->Last_Output = pid->Output;
    pid->Last_Dout = pid->Dout;
    pid->Last_Error = pid->Error;

    return pid->Output;
}

int main(int argc, char **argv)
{
    // define the robot object
    Robot *robot = new Robot();
}

```

```

// defines a keyboard object
Keyboard kb;
kb.enable(TIME_STEP);

// defines a imu object
InertialUnit *imu;
imu = robot->getInertialUnit("bmi088");
imu->enable(TIME_STEP);

// defines a gyro object
Gyro *gyro;
gyro=robot->getGyro("gyro");
gyro->enable(TIME_STEP);

// defines two motor object
Motor *wheels[2];
char wheels_names[2][20] = {"motor_l", "motor_r"};
for (int i = 0; i < 2; i++)
{
    wheels[i] = robot->getMotor(wheels_names[i]);
    wheels[i]->setPosition(INFINITY);
    wheels[i]->enableTorqueFeedback(2);
}
// defines two motor object
pid_limitparameter_init(&angle_loop);
pid_parameter_reset(&angle_loop,30,0,0);

pid_limitparameter_init(&gyro_loop);
pid_parameter_reset(&gyro_loop,5,0,0);

pid_limitparameter_init(&loop_angle_yaw);
pid_parameter_reset(&loop_angle_yaw,5,0,0);

pid_limitparameter_init(&loop_gyro_yaw);
pid_parameter_reset(&loop_gyro_yaw,1,0,0);

pid_limitparameter_init(&loop_speed);
pid_parameter_reset(&loop_speed,1,0.001,0);

// Init Successd ...
printf("Init Successd ...\n");

while (robot->step(TIME_STEP) != -1)
{
    float RealSpeed_L,RealSpeed_R,RealTorque_L,RealTorque_R;
    float EndTorque_L,EndTorque_R,Balance_Torque,Turn_Torque,walk_Torque;

    angle_pitch= imu->getRollPitchYaw()[1];
    angle_yaw  = imu->getRollPitchYaw()[2];
    gyro_pitch = gyro->getValues()[2];
    gyro_yaw   = gyro->getValues()[1];

```

```

int userkey = kb.getKey();
if(userkey == 65)
{
    speed_target = 5;
}
else if (userkey == 68)
{
    speed_target = -5;
}
else
{
    speed_target = 0;
}
std::cout << " Torque: " << EndTorque_L << " Speed: " << RealSpeed_L
<<std::endl;

pid_calculate(&angle_loop,0,angle_pitch);
pid_calculate(&gyro_loop,0,gyro_pitch);
pid_calculate(&loop_angle_yaw,0,angle_yaw);
pid_calculate(&loop_gyro_yaw,0,gyro_yaw);
pid_calculate(&loop_speed,speed_target,speed_real);

Balance_Torque = - angle_loop.Output - gyro_loop.Output ;
Turn_Torque = loop_angle_yaw.Output + loop_gyro_yaw.Output;
walk_Torque = loop_speed.Output;

EndTorque_L = Balance_Torque + Turn_Torque;
EndTorque_R = Balance_Torque - Turn_Torque;

wheels[0]->setTorque(EndTorque_L);
wheels[1]->setTorque(EndTorque_R);
RealTorque_L = wheels[0]->getTorqueFeedback();RealTorque_R = wheels[1]-
>getTorqueFeedback();
    RealSpeed_L = wheels[0]->getVelocity();RealSpeed_R = wheels[1]-
>getVelocity();speed_real = (RealSpeed_L+RealSpeed_R)/2;
}
delete robot;
return 0;
}
#endif __ALG_PID_H
#define __ALG_PID_H

typedef float PIDElem_t;    //定义PID相关变量，参数的数据类型
#define myabs(x) ((x)>0? (x):(-(x)))

/**
 * @brief PID参数结构体
 */
typedef struct
{
    PIDElem_t Kp;    // 比例系数
    PIDElem_t Ki;    // 积分系数

```

```

    PIDElem_t Kd;      // 微分系数
} PidParamTypeDef;

/* PID环数据 */
typedef struct
{
    PidParamTypeDef PID_Param;

    PIDElem_t Target;
    PIDElem_t Measure;      // 测量值
    PIDElem_t Last_Measure; // 上一次的测量值

    PIDElem_t Error;      // 误差
    PIDElem_t Last_Error; // 上一次的误差
    PIDElem_t DeadBand_Error; // 死区误差
    PIDElem_t Integral_Error; // 积分误差

    PIDElem_t Pout;      // 比例输出
    PIDElem_t Iout;      // 积分输出
    PIDElem_t Dout;      // 微分输出
    PIDElem_t Last_Dout; // 上一次的微分输出
    PIDElem_t ITerm;     // 积分项

    PIDElem_t Output;      // 输出值
    PIDElem_t Last_Output; // 上一次的输出值

    PIDElem_t Limit_Output; // 输出值的限幅
    PIDElem_t Limit_IntegralValue; // 积分的限幅
    PIDElem_t Limit_DiffValue; // 微分值的限幅

}PID_Loop_t;

void pid_limitparameter_init(PID_Loop_t *pid);
void pid_parameter_reset(PID_Loop_t *pid, PIDElem_t new_Kp, PIDElem_t new_ki,
PIDElem_t new_kd);
PIDElem_t pid_calculate(PID_Loop_t *pid, PIDElem_t target, PIDElem_t measure);
PIDElem_t LimitMax(PIDElem_t input,PIDElem_t max);

#endif

```

伺服电机驱动

转矩闭环与实际情况分析

文件（LK-TECH电机CAN协议说明）中说明了此款9025电机可以使用三种闭环控制（位置、速度、力矩）指令实现精准伺服控制。实际在平衡步兵上使用力矩闭环控制是最优选择。

1. 一方面是关节电机不同于RM的所有直流无刷电机，关节电机偏向于专业精准的伺服控制，成熟的驱动方案均有（位置、速度、力矩）控制模式，且驱动板均内置PID参数，力矩控制更精准，在RM或者专业机器人行业控制机械臂或者各种轮足、人型机器人是使用最多的控制模式。

2. 一方面是针对复杂的机器人，业界越来越倾向于使用先进的控制理论，例如LQR, MPC,自适应控制，滑模控制等。这些现代控制理论往往需要建模与分析稳定性等等，最后输出的力往往是力矩（扭矩）。

所以掌握关节电机的力矩控制是至关重要的

参数分析

由于控制算法最终输出的结果是扭矩（单位：Nm），而电机的控制是通过发送转矩电流指令给驱动板，驱动板转化成电机输出的转矩来实现的，所以，得到的目标转矩，该怎么转化为准确的转矩电流呢？

1. 淘宝电机性能参数

产品型号	Item Name	MF9025v2	
匝数	Turns	16	35
额定电压	Rated Voltage (V)	24	24
空载转速	Max Speed (rpm)	710	280
额定扭矩	Rated Torque (N.m)	2.42	2.79
额定转速	Rated Speed (rpm)	490	130
额定电流	Rated Current (A)	7.45	3.46
峰值功率	Max Power (W)	170	38
峰值扭矩	Max Torque (N.m)	4.5	5.8
转速常数	Speed Constant (rpm/V)	20	5.4
扭矩常数	Torque Constant (N.m/A)	0.32	0.81
绕组类型	Winding Type	Y	Y
相电阻	Phase Resistance (Ω)	0.5	1.9
相间电感	Phase Inductance (mH)	0.96	4.71
磁极数	Motor Poles	28	
转子惯量	Rotor Inertia (gcm ²)	4656	
电机温度监测	Motor Temperature	YES	
轴承额定静载荷	Bearing Rated Load (N)	650	
电机重量	Motor Weight (g)	880	

在图中，我们主要用到：额定电流，额定扭矩、峰值扭矩、扭矩参数

1. 电机CAN协议说明

(20) 转矩闭环控制命令（1 帧）该命令仅在 MF 和 MG 上实现

主机发送该命令以控制电机的转矩电流输出，控制值 iqControl 为 int16_t 类型，数值范围-2000~ 2000，对应实际转矩电流范围-32A~32A（母线电流和电机的实际扭矩因不同电机而异）。

数据域	说明	数据
DATA[0]	命令字节	0xA1
DATA[1]	NULL	0x00
DATA[2]	NULL	0x00
DATA[3]	NULL	0x00
DATA[4]	转矩电流控制值低字节	DATA[4] = *(uint8_t *)&iqControl
DATA[5]	转矩电流控制值高字节	DATA[5] = *((uint8_t *)&iqControl)+1
DATA[6]	NULL	0x00
DATA[7]	NULL	0x00

备注：

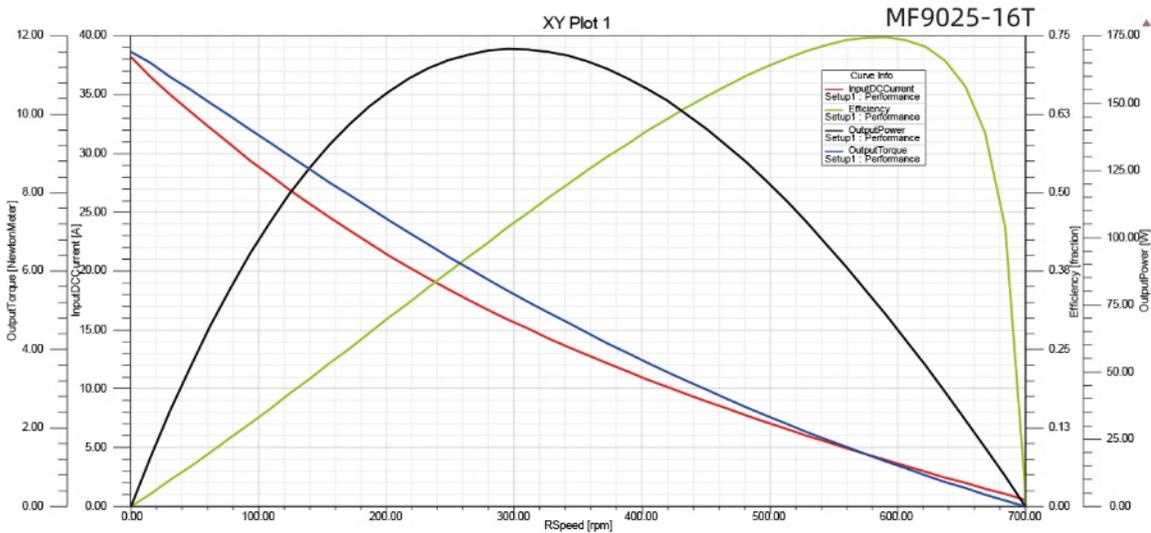
1. 该命令中的控制值 iqControl 不受上位机(LK-Motor Tool)中的 Max Torque Current 值限制。

可以看到，Iq范围是-2000~2000，对应实际转矩电流是-32A~32A。

可以看到，Iq范围是-2000~2000，对应实际转矩电流是-32A~32A。

1. 电机特性曲线

— 电流(Inputdc Current) — 效率(Efficiency) — 功率(Output Power) — 转矩(Output Torque)



需要注意的是，实际上这款MF9025实测最大爆发是能达到10Nm的扭矩的，这也印证了为什么一个平衡步兵动辄15KG起步，但是按照性能参数峰值扭矩4.5也能让车体正常瞬间起身。

转矩电流常量计算

我们最终目的是将求得的扭矩转化为Iq发送给电机，保持正确的对应关系来输出想要的扭矩。假设扭矩->Iq的这个比例为K

1. 确定最大转矩电流

为了不影响平衡步兵的正常起身瞬间的爆发且考虑到电机实际情况，为了寿命以及防止极限情况下电机出现的高频振动，将最大转矩电流限制在30A,最大转矩大约9.6Nm，最大Iq为1875。

1. 求得K

$$K = (1875/30/0.32)=195.3125$$

实际使用代码里要对扭矩输出限制9Nm,最大Iq为1875。

驱动封装

电机信息结构体定义

```
/**
 * @brief DJI电机CAN接收到的数据结构体
 */
typedef struct
{
    uint16_t Location;        // 机械角度值, 范围: 0-8191
    int16_t Speed;           // 电机转速, 单位为 rpm
    int16_t Current;         // 电机扭矩电流, 单位为 A
    uint8_t Temperature;     // 电机温度, 单位为 °C
} DJImotorFeedbackTypeDef;

/**
 * @brief DJI电机CAN发送的数据结构体
 */
typedef struct
{
    PIDElem_t angleLoopOutput; // 位置环输出
    PIDElem_t speedLoopOutput; // 速度环最终输出
} DJImotorSendTypeDef;

/**
 * @brief 瓩控电机CAN接收到的数据结构体
 */
typedef struct
{
    uint8_t cmd;              // 命令字节
    int8_t Temperature;       // 电机温度, 1°C/LSB
    int16_t Torque_iq;        // 扭矩电流值iq, 范围-2048~2048, 对应实际扭矩电流范围-33A~33A
    int16_t Speed;           // 电机转速, 1dps/LSB
    uint16_t Location;        // 编码器的数值, 范围 0~16383(14bit)
} LKmotorFeedbackTypeDef;

/**
 * @brief 瓩控电机CAN扭矩电流发送的数据结构体
 */
typedef struct
{
    int16_t Control_iq;       // 数值范围 -2000~ 2000, 对应实际扭矩电流范围-32A~32A
    int32_t Speed;
} LKmotorSendTypeDef;
```

底盘电机ID命令结构体定义

```
/**
 * @brief 底盘电机ID枚举类型枚举值
 * @attention 注意底盘不一样，电调ID不一样，这里要改
 */
typedef enum
{
    CMD_Multiple = 0x280, // 多电机控制
    CMD_iqLoop = 0xA1, // 单个电机转矩电流闭环
    CMD_speedLoop = 0xA2, // 单个电机速度闭环
    LEFT_ID = 0x141, // 左电机标识符
    RIGHT_ID = 0x142, // 右电机标识符

    /**
     * @brief 底盘电机数组编号枚举类型枚举值
     * @note 数组编号和ID编号一一对应
     */
    LEFT = 0, // 左电机数组编号
    RIGHT = 1, // 右电机数组编号
} ChassisMotor_Id_TypeDef;
```

CAN协议接收解析

```
/**
 * @brief 大疆电机数据解码、转移
 * @param[in] CanRxMsg, 符合电机类的结构体变量
 * @retval 无
 */
void can_DJImotor_data_decode(CanRxMsg *Rx , DJImotorFeedbackTypeDef *str)
{
    DJImotorFeedbackTypeDef Data;

    Data.Location = (Rx->Data[0]<<8) | Rx->Data[1];
    Data.Speed = (Rx->Data[2]<<8) | Rx->Data[3];
    Data.Current = (Rx->Data[4]<<8) | Rx->Data[5];
    Data.Temperature = Rx->Data[6];

    memcpy(str, &Data, sizeof(DJImotorFeedbackTypeDef));
}

/**
 * @brief 瓴控电机数据解码、转移
 * @param[in] CanRxMsg, 符合电机类的结构体变量
 * @retval 无
 */
void can_LKmotor_data_decode(CanRxMsg *Rx , LKmotorFeedbackTypeDef *str)
{
    LKmotorFeedbackTypeDef Data;
```

```

Data.cmd          = Rx->Data[0];
Data.Temperature = Rx->Data[1];
Data.Torque_iq   = (Rx->Data[3]<<8) | Rx->Data[2];
Data.Speed       = (Rx->Data[5]<<8) | Rx->Data[4];
Data.Location    = (Rx->Data[7]<<8) | Rx->Data[6];

memcpy(str,&Data,sizeof(LKmotorFeedbackTypeDef));
}
/**
 * @brief      底盘电机数据更新
 * @param[in]  CanRxMsg
 * @retval    无
 */
void Chassis_Motor_Receive(CanRxMsg* RxMessage)
{
    switch(RxMessage->StdId)
    {
        case LEFT_ID:
            can_LKmotor_data_decode(RxMessage,&Chassis.motor[LEFT].Feedback);
            break;

        case RIGHT_ID:
            can_LKmotor_data_decode(RxMessage,&Chassis.motor[RIGHT].Feedback);
            break;
    }
}

```