

# 四足机器人随笔

by: Wenboxing

2021.2.18

## 写这份材料的目的：

笔者目前从事的工作是腿足机器人运动控制，在一开始学习腿足机器人相关的入门资料时，被《机器人学导论》，《legged robots that balance》，《仿人机器人》等资料弄得有点糊涂，满篇公式看得人头疼，看懂公式后实际写代码又感觉无从下手。一直想基于自己的理解写一份比较简短的笔记，本材料的重点不是去推机器人学相关的一堆堆公式，而是将相关理论的用途和代码实现过程作为重点，笔者将用尽可能简短的语言说明每一处计算的用途以及代码实现步骤，材料中所提到的计算部分将提供源代码，或者可作为参考的开源示例代码。个人水平有限，眼光难免比较局限，如发现错误或者引用遗漏之处还请麻烦联系笔者 [twyang@zju.edu.cn](mailto:twyang@zju.edu.cn)，也欢迎大家与笔者讨论。

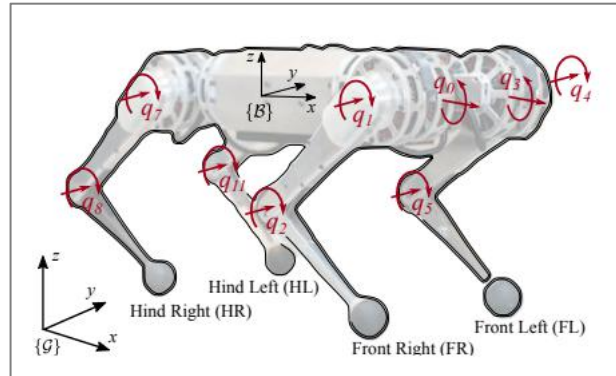
## 目录

1 一些基础计算.....	1
1.1 正运动学.....	1
1.2 逆运动学.....	2
1.3 雅可比矩阵.....	4
1.3.1 3 维雅可比.....	5
1.3.2 6 维雅可比.....	5
1.4 常用插值轨迹.....	5
1.5 独立 PD 控制应用.....	6
1.6 动力学求解.....	7
1.6.1 动力学作用.....	7
1.6.2 常见动力学用法.....	7
1.6.3 计算方法.....	8
2 基于简化模型的控制.....	15
2.1 单刚体模型介绍.....	16
2.2 基于单刚体模型的控制方法.....	16
2.2.1 虚拟模型控制 (VMC) .....	16
2.2.2 直接优化法.....	17
2.2.3 基于单刚体模型的模型预测控制方法 (MPC) .....	18
3 引入全身动力学模型的控制.....	21
3.1 基于零空间.....	21
3.2 基于优化.....	23
4 一些与 MIT Cheetah 相关的算法小知识.....	26
4.1 关节设计思想.....	26
4.2 花式动作.....	27
4.2.1 太空步.....	28
4.2.2 跳上斜面/V 型板行走/两个斜面间跳跃.....	29

## 1 一些基础计算

在实际写机器人代码过程中，难免会需要解决运动学、雅可比矩阵、轨迹等基础计算问题，此处以四足为例进行相关的计算。

对四足建立如下的坐标系，



各个关节角度为零时即为四条腿绷直状态，此时各关节坐标系与身体坐标系各轴平行。

### 1.1 正运动学

正运动学 (FK) 的作用是求末端某点相对于某个坐标系的位置，实际中，FK 往往用在对末端点轨迹规划过程中根据各关节角求末端点此刻的位置。

四足 4 条腿具有相同的结构，此处以左前腿为例。左前腿 3 个自由度，根据 DH 参数写出每个关节坐标系的齐次变换矩阵如下：

```
%hipx rt. hipx固定系(与hipx关节坐标系共原点，各坐标轴与身体坐标系平行)
T01 = [1      0      0      0;
       0  cos(q0) -sin(q0)  0;
       0  sin(q0)  cos(q0)  0;
       0      0      0      1];

%hipy rt. hipx
T12 = [cos(q1)  0  sin(q1)  0;
       0        1  0        hipy_to_hipx;
       -sin(q1)  0  cos(q1)  0;
       0        0  0        1];
```

```

%knee rt. hipy
T23 = [cos(q2)    0    sin(q2)    0;
        0        1        0        0;
       -sin(q2)    0    cos(q2)  -knee_to_hipy;
        0        0        0        1];

%foot rt. knee
T34 = [1    0    0    0;
        0    1    0    0;
        0    0    1  -foot_to_knee;
        0    0    0    1];

T04 = T01*T12*T23*T34;
xyz = T04(1:3, 4:4);

```

xyz 即为腿末端（足）相对 hipx 固定系的位置，此时要求足相对身体坐标系的位置，加上身体系原点到对应 hipx 固定系原点的常数向量即可。

## 1.2 逆运动学

逆运动学（IK）的作用是已知末端某点相对某坐标系的 xyz 位置，求此时的各个关节角。实际使用中，当需要对机械臂末端/腿足机器人末端做轨迹插值时，就可以使用 IK 求得每个控制周期的对应关节角位置。

IK 的求解稍微麻烦一点，数值迭代法算是一种通用解法，但为了可以可靠的进行例如 1kHz 的控制周期，最好还是求出它的解析解（也叫封闭解）。求解析解没有统一的方法，笔者觉得推导 IK 过程如果用的是代数法那干的更多的是因式变换这类的工作，几何法则干的更多的是三角形构造几何变换等。

Matlab 的求 IK 解析解函数 `ikine6s()`，只适用于关节数量为 6，且腕部三个旋转关节的轴相较于一个点的情况，其余构型可以使用 `ikine()` 求数值解。笔者也尝试了用 matlab 的 `solve()` 函数根据 FK 的表达式求关节角，仅 3 自由度 matlab 已经解不出来了，实际还是得自己推导。

还是 1.1 中的左前腿，计算出的正运动学结果如下：

```

xyz =
- foot_to_knee*(cos(q1)*sin(q2) + cos(q2)*sin(q1)) - knee_to_hipy*sin(q1)
hipy_to_hipx*cos(q0) - foot_to_knee*(sin(q0)*sin(q1)*sin(q2) - cos(q1)*cos(q2)*sin(q0)) + knee_to_hipy*cos(q1)*sin(q0)
hipy_to_hipx*sin(q0) - foot_to_knee*(cos(q0)*cos(q1)*cos(q2) - cos(q0)*sin(q1)*sin(q2)) - knee_to_hipy*cos(q0)*cos(q1)

```

根据经验，我们使用 matlab 求一下  $x*x + y*y + z*z$

```

— simplify(xyz(1,1)^2 + xyz(2,1)^2 + xyz(3,1)^2)
命令窗口
ans =

foot_to_knee^2 + 2*cos(q2)*foot_to_knee*knee_to_hipy + hipy_to_hipx^2 + knee_to_hipy^2

```

于是可以直接求出  $q_2$ 。

为了看着简洁点，做如下变量替换后得到：

```

hipy_to_hipx = L1;
knee_to_hipy = L2;
foot_to_knee = L3;
s1 = sin(q1);
c1 = cos(q1);
s12 = sin(q1+q2);
c12 = cos(q1+q2);
% xyz---->
x = -L3*s12 - L2s1;
y = L1c0 + (L3c12+L2c1)*s0;
z = L1s0 - (L3c12+L2c1)*c0;

```

观察  $y$ 、 $z$  表达式，做如下处理：

$$\frac{y - Lc_0}{z - Ls_0} = \frac{-s_0}{c_0}$$

$$\Rightarrow Ls_0^2 - zs_0 = yc_0 - Lc_0^2$$

$$\Rightarrow \textcircled{A} yc_0 + zs_0 = L$$

化成这种形式后我们可以直接使用《机器人学导论》（原书第 3 版）第 87 页的结论求解  $q_0$ 。

将超越方程

$$a \cos \theta + b \sin \theta = c \quad (4-36)$$

变换成含有半角正切的一次多项式，以求解 $\theta$ 。

采用式(4-35)的变换式，将上式乘以 $1+u^2$ ，得

$$a(1-u^2) + 2bu = c(1+u^2) \quad (4-37)$$

取 $u$ 的幂函数形式为

$$(a+c)u^2 - 2bu + (c-a) = 0 \quad (4-38)$$

由一元二次方程求解公式解出：

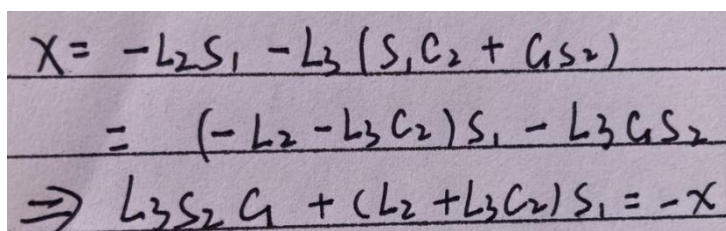
$$u = \frac{b \pm \sqrt{b^2 + a^2 - c^2}}{a+c} \quad (4-39)$$

因此，

$$\theta = 2 \tan^{-1} \left( \frac{b \pm \sqrt{b^2 + a^2 - c^2}}{a+c} \right) \quad (4-40)$$

(机器人学导论第 87 页截图)

$q_1$  的求解同样也能化成相同的超越方程形式求取。



Handwritten equations showing the derivation of  $q_1$  from the position equations:

$$\begin{aligned} X &= -L_2 S_1 - L_3 (S_1 C_2 + C_1 S_2) \\ &= (-L_2 - L_3 C_2) S_1 - L_3 C_1 S_2 \\ \Rightarrow L_3 S_2 C_1 + (L_2 + L_3 C_2) S_1 &= -X \end{aligned}$$

至此，IK 已求解完毕。当然，这只是相对简单的 IK 求解，若要做到鲁棒性，还需要注意奇异点规避，因笔者没有很深入的去做过这块，此处就无法详述了。

### 1.3 雅可比矩阵

雅可比矩阵主要有两个用途：

1) 已知各关节速度，求末端相对某坐标系的速度。在规划末端速度时便可以用它求得当前实际的末端速度。

$$\mathbf{v} = \mathbf{J} \dot{\mathbf{q}}$$

2) 已知末端相对某坐标系的力，求对应的关节力矩。在机器人控制中，规划末端的作用力，便可以通过它求得各关节该发出的力矩。

$$\boldsymbol{\tau} = \mathbf{J}^T \mathbf{F}$$

这里的速度  $\mathbf{v}$ ，末端力  $\mathbf{F}$  根据情况可以是 xyz 方向的 3 维量（线速度/力），也可以是包含角速度/转矩的 6 维量。

### 1.3.1 3 维雅可比

雅可比矩阵求解主要有旋量法（几何法）和解析法。解析法即求导法，这个可以使用 matlab 的 `jacobian()` 函数很方便的求出，如下所示：

```
%-----Jacobian-----%
% 3维雅可比
Jac = jacobian(T04(1:3, 4:4), [q0;q1;q2]);
```

可能有人会纠结这两种方法求出来的结果会不会不一样啊，这里给出明确的结论：在笛卡尔坐标系下这两种方式求出的结果是相同的<sup>1</sup>。因此可以放心的使用 `jacobian()` 函数快速的求解出 3 维雅可比矩阵。

### 1.3.2 6 维雅可比

6 维雅可比可以认为是在 3 维雅可比基础上加上 3 行表示末端相对某坐标系的角速度，这个雅可比比较好求，每一列就是该关节相对某坐标系（或者这里我们把它叫做基坐标系）的旋转矩阵的对应转轴的  $3 \times 1$  向量，关于这个求法的更详细的讲解可以参考 [https://mp.weixin.qq.com/s/UBhr\\_sIP7J238HM1cTk9qQ](https://mp.weixin.qq.com/s/UBhr_sIP7J238HM1cTk9qQ)，此处不再赘述，最终左前腿的 6 维雅可比在 matlab 中的求法如下：

```
%-----Jacobian-----%
% 3维雅可比
Jac = jacobian(T04(1:3, 4:4), [q0;q1;q2]);
% 6维雅可比
T02 = T01*T12;
T03 = T01*T12*T23;
Jw_trans = [[T01(1:3, 1:1)]'; [T02(1:3, 2:2)]'; [T03(1:3, 2:2)]'];
Jac_6D = [jacobian(T04(1:3, 4:4), [q0;q1;q2]);
          Jw_trans'];
```

## 1.4 常用插值轨迹

一般情况下，我们所说的轨迹是一个与时间相关的函数，比如对于一个关节，指定了期望的关节位置，如何使关节可以平滑地移动到期望的位置呢？此时便可以在初始位置和期望位置之间进行插值，得到平滑连续的位置、速度轨迹，然后在每个控制周期里使用 PD 控制律  $\tau = K_p(\theta_d - \theta) + K_d(\dot{\theta}_d - \dot{\theta})$  下发关节力矩指令

<sup>1</sup> ETH 动力学讲义《Robot Dynamics Lecture Notes》第 41 页



使关节可以稳定连续地摆动到期望位置。实际中笔者较常用的插值轨迹是三次样条。

三次样条公式：

$$\begin{aligned}\theta(t) &= a_0 + a_1 t + a_2 t^2 + a_3 t^3 \\ a_0 &= \theta_0 \\ a_1 &= \dot{\theta}_0 \\ a_2 &= \frac{3}{t_f^2}(\theta_f - \theta_0) - \frac{2}{t_f} \dot{\theta}_0 - \frac{1}{t_f} \dot{\theta}_f \\ a_3 &= -\frac{2}{t_f^3}(\theta_f - \theta_0) + \frac{1}{t_f^2}(\dot{\theta}_f + \dot{\theta}_0)\end{aligned}$$

其中带 0 下标的量是初始位置的位置/速度，带 f 下标的是期望终点位置/速度， $t_f$  表示初始到终点的总时间。该曲线满足任何起始、终止位置以及任何起始速度、终止速度的连续性。在每个控制周期里，下一时刻的期望位置与当前期望位置的差除以控制周期便是期望的轨迹速度。

## 1.5 独立 PD 控制应用

求解完正逆运动学、雅可比和三次样条公式后，我们便可以初步的控制一个多轴机械臂/四足某条腿末端点摆到期望的位置。一般步骤为，给定期望的末端位置——>使用正运动学和雅可比矩阵计算出初始的末端位置、速度——>对初始点、期望终点进行三次样条插值得到当前时刻和下一时刻的期望末端位置——>逆运动学求出当前时刻和下一时刻的各关节期望位置——>得到当前时刻期望的关节位置、速度，使用 PD 控制律计算关节力矩。

这是最基础的定点控制（控制末端到达一固定的期望位置）方式，这种 PD 控制方式也早已经被证明了其稳定性<sup>2</sup>。也许有人会问，这样感觉好麻烦，那我可不可以直接规划各个关节的轨迹而不用正逆运动学和雅可比呢？如果你只需要控制关节到达一个已知的位置，那用不着正逆运动学，直接对关节插值就行。上述这套流程主要用于需要规划末端点位置的场景，例如使机械臂末端移动到某个固定点，或者控制四足的摆动腿摆到期望的落脚点。

<sup>2</sup> Arimoto S. Stability and Robustness of PID Feedback Control for Robot Manipulators of Sensory Capability[C]// Robotics Research: First International Symposium. MIT Press, 1984.

## 1.6 动力学求解

动力学部分相对来说难度就开始增加了，这部分笔者尽量避免大段公式推导，只是简述笔者对动力学作用的理解和计算方法。

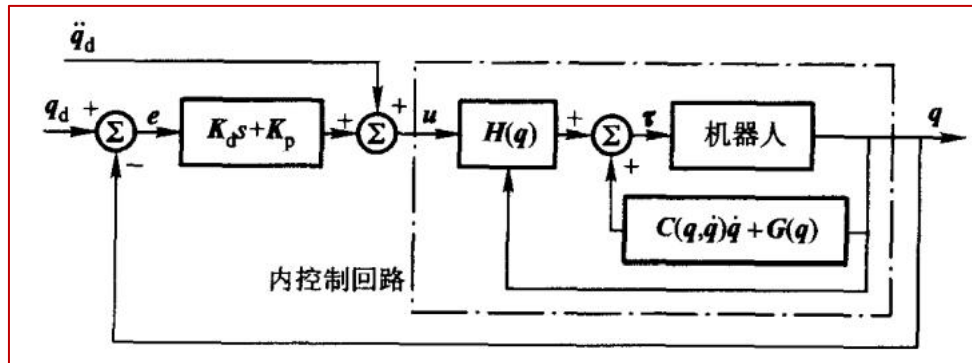
### 1.6.1 动力学作用

在机械臂/腿足机器人腿摆动过程中，各个关节实际是会受到惯性力、离心力、哥氏力以及自身重力等多种额外力的影响，1.5 中的 PD 控制方法实际可以认为是一种简单粗暴的方法，是完全忽略了这些额外力的，不过在笔者实际开发过程中我对动力学的理解是，对轨迹跟随精度、速度要求不太高的场景下使用 1.5 的 PD 方法已经足够了。动力学真正发挥作用的场景往往是要求对轨迹实现快速、高精度跟随的场景，这种情境下单靠调节 PD 参数已经不能抵消这些额外力的扰动了，类推到四足，在快速运动情况下引入动力学可以更好的补偿掉惯性力等带来的扰动，提高机器人的鲁棒性。

### 1.6.2 常见动力学用法

#### 1) 逆动力学法（计算力矩法）

该方案控制框图如下：



即在内控制回路中引入非线性补偿

$$\begin{aligned} u &= \ddot{q}_d + K_d(\dot{q}_d - \dot{q}) + K_p(q_d - q) \\ &= \ddot{q}_d + K_d\dot{e} + K_p e \end{aligned}$$

最终的控制律为

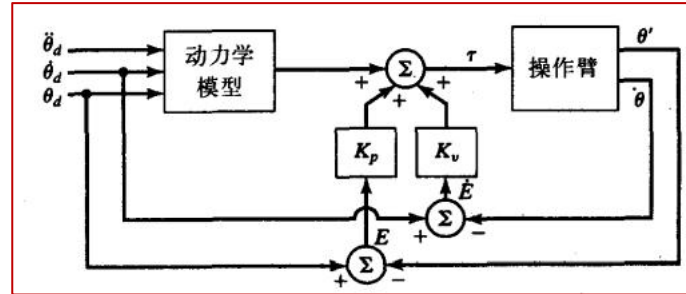
$$\tau = H(q)(\ddot{q}_d + K_d\dot{e} + K_p e) + C(q, \dot{q})\dot{q} + G(q)$$

这是一种很常用的引入动力学的控制方案，当然这种方案使得需要计算形如拉格朗日表达形式的动力学方程，比较复杂，同时也需要动力学部分的计算速度

跟上伺服速率，对计算性能要求较高。

## 2) 前馈控制法

该方案的控制框图如下：



基于模型的控制部分在伺服回路之外，如果主控板计算性能跟不上，可以让外部的动力学部分以较低的速率运行。不过这种方法并不能向逆动力学法那样实现完全解耦<sup>3</sup>。其实现在的板子性能已经提升了很多了，要使用逆动力学法其实也没问题了。

### 1.6.3 计算方法

这一块我自己当初觉得是比较折磨人的，《机器人学导论》等这些参考书尽是满篇的公式，例子呢一般也就只是个 2 连杆机械臂，自由度上去之后写代码还是会不知道从哪下手。因此这块笔者不讲太多公式推导，只着重讲具体代码计算方法。

首先，动力学计算结果与运动学类似，分为解析解和数值迭代解，实际两者是共通的，因为将数值迭代法中的变量都用 matlab 的符号变量表示的话，最终也能得到解析解。不过在笔者自己的实践中发现，当自由度多了后（例如笔者所算的机器人有 13 个驱动自由度+6 个欠驱动自由度）matlab 生成的解析解是没法看的，表达式太复杂，竟然有 10 多 w 行，放到代码里计算时间很拉胯……

```
1 #include "Dynamics_Calculate.h"
2
3 static void output1(Eigen::Matrix<double, 19, 19> &p_output1, const Eigen::Matrix<double, 19, 1> &var1) { ... }
11736
11737 static void output1(Eigen::Matrix<double, 19, 19> &p_output1, const Eigen::Matrix<double, 19, 1> &var1, const Eigen::Mat
125076
125077 static void output1(Eigen::Matrix<double, 19, 1> &p_output1, const Eigen::Matrix<double, 19, 1> &var1) { ... }
```

实际常见的除了动力学辨识步骤或者基于解析式分析控制特性需要用到解析解表达式，其余时间做控制完全可以使用迭代法求数值解，以今天的较好的工控板的计算性能来说，跑迭代法绰绰有余。

<sup>3</sup> 《机器人学导论》（原书第 3 版）第 237 页

接下来讲计算方法，笔者实践过的主要有两种方案——用库或者自己算，接下来将分别阐述。

### 1.6.3.1 用库

现有的机器人动力学计算库是比较多的，基本都是导入机器人的 URDF 模型后使用迭代法求动力学。这块就我所知的比较好用的有 Frost, Drake, Pinocchio, RBDL 等，这些库的使用示例都存放于笔者的该篇博客：<https://zhuanlan.zhihu.com/p/231351878>，感兴趣的读者可以按照博客中的例程自己跑跑代码试试。

### 1.6.3.2 基于 Roy 理论直接算

用了段时间库后，为了搞清这些库的具体计算过程，方便后期查错，笔者也尝试自己写了下动力学计算代码。

动力学计算代码主要参考资料为 Roy Featherstone 的《Rigid Body Dynamics Algorithms》，这本书我觉得很好的一点是可操作性很高，可以照着书里的伪代码直接写出动力学计算代码，书本身也有配套代码：

<http://royfeatherstone.org/>

下面笔者将根据自己的使用体验把一些关键点写一下，有关书中更详细的点还请参考原书籍。

首先先把笔者开始觉得有点含糊的几个公式解释下。

#### 1. 角速度与旋转矩阵的关系

大部分书上都给出了个统一的结论：

$$[A\omega_{AB}]_{\times} = \dot{C}_{AB} \cdot C_{AB}^T$$

笔者开始对这个结论咋来的感到过疑惑，这里用比较直观的话解释下。

假定旋转矩阵  ${}^A_R(t)$  表示 B 坐标系相对 A 坐标系的旋转矩阵，它的列向量就是 B 坐标系在 A 坐标系中的方向向量，因此  ${}^A\dot{R}(t)$  可以理解为刚体旋转时 B 坐标系中 xyz 轴相对 A 的“速度”，于是旋转矩阵的导数为（下面的角速度  $\omega$  和  $r$  向量均是 B 相对 A 的）：

$$\dot{R}(t) = \begin{pmatrix} \omega(t) \times \begin{pmatrix} r_{xx} \\ r_{xy} \\ r_{xz} \end{pmatrix} & \omega(t) \times \begin{pmatrix} r_{yx} \\ r_{yy} \\ r_{yz} \end{pmatrix} & \omega(t) \times \begin{pmatrix} r_{zx} \\ r_{zy} \\ r_{zz} \end{pmatrix} \end{pmatrix}$$

角速度叉乘矩阵按照叉乘矩阵定义写为：

$$[w]_{\times} = \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix}$$

于是对任意向量  $P$  都有  $w \times P = [w]_{\times} P$ ，所以：  $\dot{R}(t) = [w]_{\times} R(t) \Rightarrow [w]_{\times} = \dot{R}(t) R^T(t)$

## 2. 欧拉方程

转矩与角速度的关系许多书上都能看到这样的一个结论：

$$\frac{d}{dt}(I\omega) = I\dot{\omega} + \omega \times I\omega$$

这个方程笔者开始不太理解第二项的叉乘是咋来的，这里也尝试用比较直观的语言描述下。

第一项很好理解，主要是第二项，有的地方将第二项解释为是非惯性系带来的影响，实际这个式子是通用的。

以相对惯性系的欧拉方程推导为例：

物理量下标  $I$  表示该量在惯性参考系（Inertia frame）中描述，这里刚体的惯性张量  $I$  不是一个常数，而是随时间变化的量，因此：

$$\tau_I = \frac{d}{dt}(I_I \omega_I) = I_I \dot{\omega}_I + \dot{I}_I \omega_I$$

$\dot{I}_I$  又等于啥呢？直观的说，根据张量的定义，张量是刚体上每个点相对某一坐标系的位置向量做一些运算后的体积分，即：  $I = [\int_V -[r]_{\times}[r]_{\times} \rho dv]$ ，所以  $\dot{I}_I$  可以理解为因为刚体转动导致的  $r$  向量变化产生的惯量变化律，可以写出：

$$\dot{I}_I = \omega_I \times I_I \omega_I$$

若要更严谨的推导过程可以参考：

<https://www.cnblogs.com/21207-iHome/p/9196997.html>

把这两个一开始可能会使人感到有点疑惑的地方解释清楚后，现在可以参照 Roy 的书开始写动力学代码，这部分笔者不会讲太多的公式推导过程，所引用的结论公式会注明在书中的页码，需要看详细推导的读者可以阅读原书。

Roy 的《Rigid Body Dynamics Algorithms》一大特点是将以往表征力和力矩的两个方程统一为一个方程。

以往表征力和力矩方程：

$$\mathbf{f} = m \mathbf{a}_C \quad \text{and} \quad \mathbf{n}_C = \mathbf{I} \dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \mathbf{I} \boldsymbol{\omega}$$

将线速度、角速度信息都写在一个矩阵里的欧拉方程形式：

$$\mathbf{f} = \mathbf{I} \mathbf{a} + \mathbf{v} \times^* \mathbf{I} \mathbf{v}$$

这样在代码编写过程上就会很简洁，书里重要的基础公式都在第2章，这里只贴几个重要的矩阵：

6D 速度 (P12)，6D 力 (P15)

$$\hat{\mathbf{v}}_O = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \\ v_{Ox} \\ v_{Oy} \\ v_{Oz} \end{bmatrix} = \begin{bmatrix} \underline{\boldsymbol{\omega}} \\ \underline{\mathbf{v}}_O \end{bmatrix} \quad \hat{\mathbf{f}}_O = \begin{bmatrix} n_{Ox} \\ n_{Oy} \\ n_{Oz} \\ f_x \\ f_y \\ f_z \end{bmatrix} = \begin{bmatrix} \underline{\mathbf{n}}_O \\ \underline{\mathbf{f}} \end{bmatrix}$$

6D 下的转换阵 (P23)：

$$\begin{aligned} \text{rx}(\theta) &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & c & s \\ 0 & -s & c \end{bmatrix} & \text{ry}(\theta) &= \begin{bmatrix} c & 0 & -s \\ 0 & 1 & 0 \\ s & 0 & c \end{bmatrix} & \text{rz}(\theta) &= \begin{bmatrix} c & s & 0 \\ -s & c & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ (c = \cos(\theta), s = \sin(\theta)) & & & \\ \text{rot}(\mathbf{E}) &= \begin{bmatrix} \mathbf{E} & \mathbf{0} \\ \mathbf{0} & \mathbf{E} \end{bmatrix} & \text{xlt}(\mathbf{r}) &= \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ -\mathbf{r} \times & \mathbf{1} \end{bmatrix} & \text{rotx}(\theta) &= \text{rot}(\text{rx}(\theta)) \\ & & & & \text{roty}(\theta) &= \text{rot}(\text{ry}(\theta)) \\ & & & & \text{rotz}(\theta) &= \text{rot}(\text{rz}(\theta)) \end{aligned}$$

6D 速度叉乘矩阵 (P37)

$$\begin{bmatrix} \boldsymbol{\omega} \\ \mathbf{v}_O \end{bmatrix} \times = \begin{bmatrix} \boldsymbol{\omega} \times & \mathbf{0} \\ \mathbf{v}_O \times & \boldsymbol{\omega} \times \end{bmatrix} = \begin{bmatrix} 0 & -\omega_z & \omega_y & 0 & 0 & 0 \\ \omega_z & 0 & -\omega_x & 0 & 0 & 0 \\ -\omega_y & \omega_x & 0 & 0 & 0 & 0 \\ 0 & -v_{Oz} & v_{Oy} & 0 & -\omega_z & \omega_y \\ v_{Oz} & 0 & -v_{Ox} & \omega_z & 0 & -\omega_x \\ -v_{Oy} & v_{Ox} & 0 & -\omega_y & \omega_x & 0 \end{bmatrix}$$

质心 C 相对某一点 O 的向量记作  $\mathbf{c}$ ，则在 O 坐标系中的 6D 惯量形式为 (P33)：

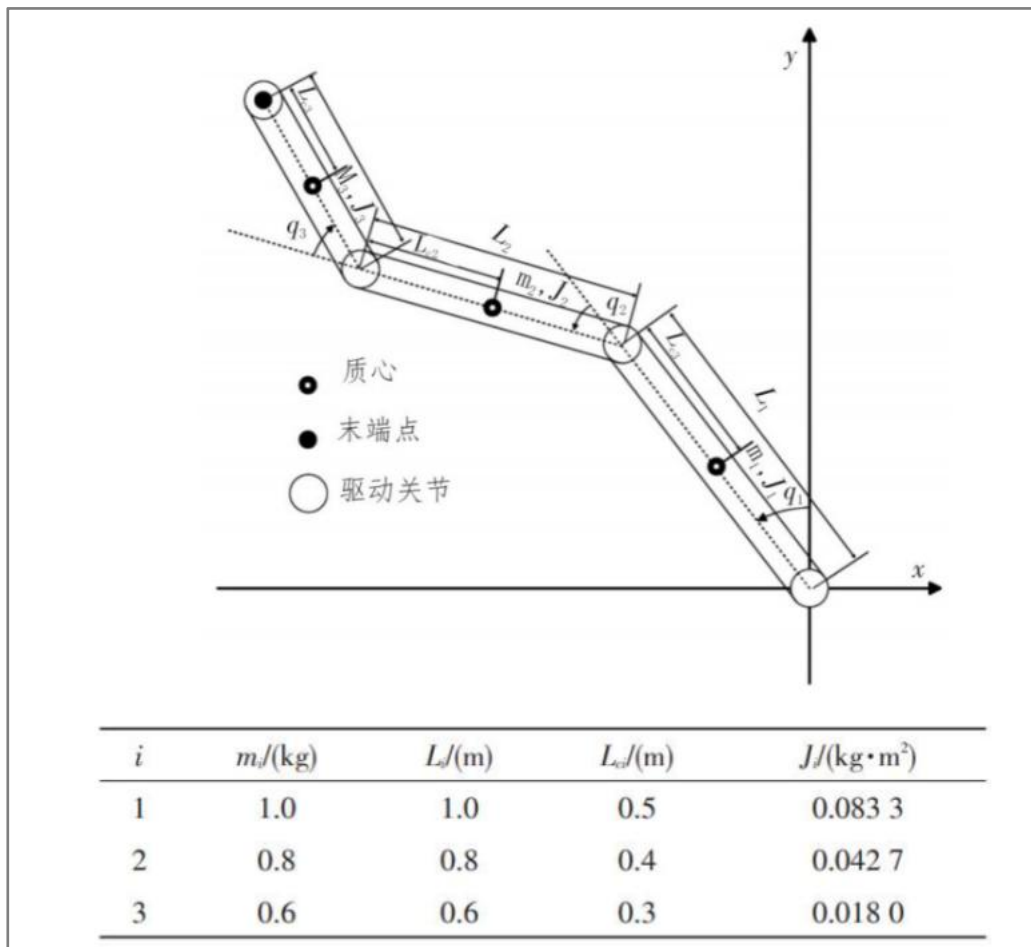
$$\mathbf{I}_O = \begin{bmatrix} \bar{\mathbf{I}}_C + m \mathbf{c} \times \mathbf{c}^T & m \mathbf{c} \times \\ m \mathbf{c} \times^T & m \mathbf{1} \end{bmatrix}$$

在 1.6.2 中提到了常用的引入动力学的控制方法——前馈法和逆动力学法，两种方法所需要的动力学方程形式分别为带入加速度、速度、位置直接求得力矩和拉格朗日形式的动力学方程形式。

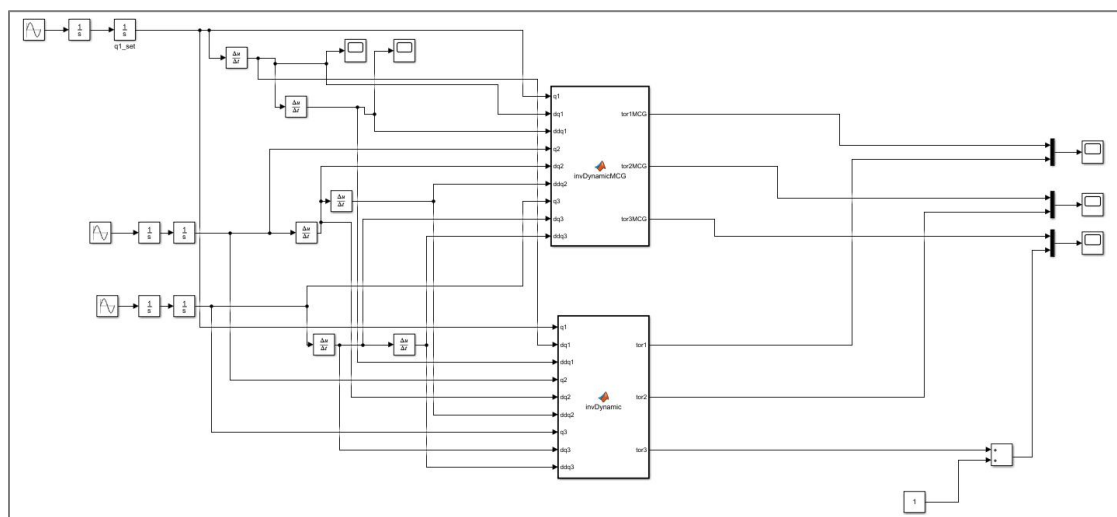
带入加速度、速度、位置直接求得力矩的动力学方程求解伪代码在 P96，对应的 Cheetah 开源代码是 FloatingBaseModel.cpp 中的 inverseDynamics() 函数；拉格朗日的动力学方程形式在 P183 和 P185 页，对应 Cheetah 代码 FloatingBaseModel.cpp 中的 massMatrix()，generalizedCoriolisForce()，generalizedGravityForce() 函数。

书中任何一种形式的动力学计算伪代码均可用于固定基座（如机械臂）和浮动基座（例如腿足机器人、无人机）的动力学求解。其实在笔者自己的理解中，所谓的浮动基动力学模型其实就是假定世界系与身体间有个假想的 6 自由度关节，然后浮动基动力学代码就是负责算出实际驱动关节的力矩和这假想的 6 自由度关节的力+力矩（论文中常叫“wrench”）。其中这欠驱动的 6 自由度关节所需要的力+力矩在实际中一般是假定足末端作用力/力矩直接作用在质心。关于两种形式的动力学计算笔者已经基于如下的 3 自由度机械臂为例编写了对应的代码，代码文件为 threeDofArm\_MCGForm.slx，其中笔者忽略了重力矩这一项，其余代码均是按照书中提到的伪代码编写的。其实以笔者自己读代码的结果来看，MIT 开源的 Cheetah 控制代码的动力学部分以及 RBDL 的动力学代码部分都是参照 Roy 的伪代码写的。



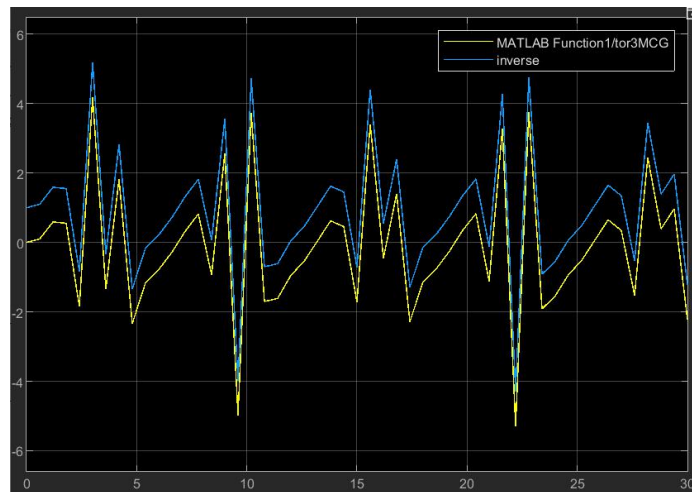


为了看着方便，给其中一种方法力矩结果加个常数偏置 1，以关节 3 为例，可以看出两种方法计算出的最终力矩结果是相同的。



两种方法的关节 3 力矩图（其中一条曲线加了常数偏置 1）：





注：simulink 中初始  $\sin$  曲线之所以加两重积分，是为了微分求得的速度、加速度平滑没有突变。

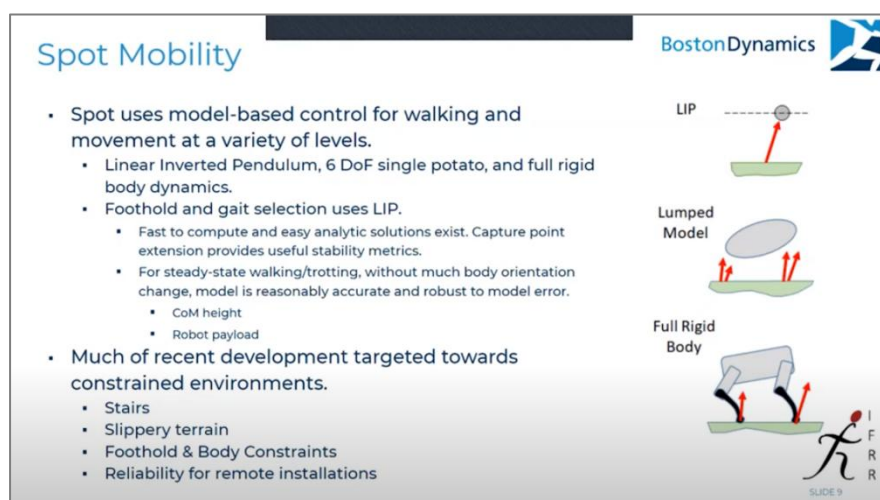
## 2 基于简化模型的控制

本部分笔者将根据自己的实践体验粗浅的谈谈与四足机器人控制相关的方面，因个人水平有限，权当抛砖引玉了。

四足机器人的控制，可以理解为分为两块：支撑腿需要维持身体平衡，比如保证身体高度不掉，姿态角稳定机器人不倒向一边；同时摆动腿在低速情况就用 1.5 中的 PD 控制法让其摆到期望位置即可，摆动腿的期望终点位置常基于倒立摆模型计算，详细可参考笔者博客：<https://zhuanlan.zhihu.com/p/190028074>。

最原始简单的应当属于位置控制，虽说是一种很局限的方法，为防止有读者含糊位控和力控，这里也稍微提一下位控。位置控制我的理解是最终实质控制的其实就是各个关节跟随一条轨迹，全程只是在规划位置轨迹，不会去考虑比如需要给身体怎样的力/力矩才能使其保持平衡这种问题。举个例子，四足机器人从趴在地上到站起这个过程大多都是直接给每个关节一个期望终点角度，然后位置控制让关节动到这个定点，PD 调好了，机器人也就站稳了。至于位控最大的局限性，举个例子，如果要求四足能在草地、石子路等不同地形运动，单靠给轨迹位控，这轨迹怎么给，基本搞不了。所以必须直接考虑给怎样的力/力矩才能使机器人平衡，这就是力控，这样才能使机器人能应对不同地形，也才能让机器人更灵活敏捷。

常用的四足机器人控制模型可总结为下图的 3 种，其中倒立摆模型主要用于摆动腿落脚点位置的计算。



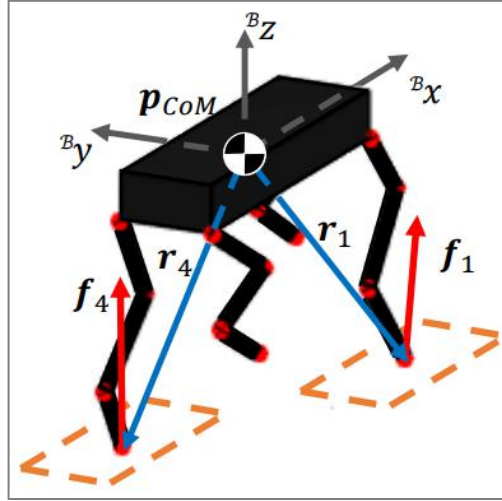
(摘自 <https://www.bilibili.com/video/BV1Tt4y1z7uW> 第 41min)

直接用四足全身动力学模型做控制模型太过复杂，而且也存在动力学参数辨

识问题，因此常见的是先基于简化的模型做控制，集中质量模型（lumped mass），也称单刚体模型，就是一种非常常用的四足简化模型。

## 2.1 单刚体模型介绍

单刚体模型直观的说就是忽略四足的腿的影响，把机器人直接看作一个整体，也即一个刚体，地面对腿末端的作用力直接等效作用在质心，如下图所示。



其运动学方程可以很容易的写出：

$$\begin{aligned} \ddot{\mathbf{p}} &= \frac{\sum_{i=1}^n \mathbf{f}_i}{m} - \mathbf{g} \\ \frac{d}{dt}(\mathbf{I}\boldsymbol{\omega}) &= \sum_{i=1}^n \mathbf{r}_i \times \mathbf{f}_i \end{aligned}$$

## 2.2 基于单刚体模型的控制方法

介绍了单刚体模型的定义，这部分讲下基于它的常见控制方法

### 2.2.1 虚拟模型控制（VMC）

个人觉得 VMC 应该算是很早的基于单刚体模型的方法，笔者对 VMC 的理解就是把支撑腿与地面的接触点看作基座，然后与身体连接处看作“机械臂”末端，对身体的 XYZ 位置、RPY 角度作 PD 控制律算出末端期望的力/力矩，然后根据  $\boldsymbol{\tau} = \mathbf{J}^T \mathbf{F}$  算出各关节力矩执行即可。当然在具体工程化过程中可能会存在不可控自由度的问题，也即雅可比的秩  $< 6$  的情况，这时还得把一些自由度直接指定常数才能保证全控。因笔者没有具体开发过四足 VMC，这里就不能细讲了，但 VMC 相关的资料是很多的。

## 2.2.2 直接优化法

知道其质心的运动学方程，那给出期望的 XYZ、RPY 加速度，然后直接用二次规划使得实际质心状态逼近期望，则可以算出期望的足末端力，然后雅可比算到关节力矩即可<sup>4</sup>。

**注意：**此处的腿雅可比足是末端，而 2.2.1 中的雅可比足是基座，方向一变，二者雅可比是完全不同的，也不存在正负号什么的转换关系，得分别算。

上述单刚体模型写为矩阵形式：

$$\underbrace{\begin{bmatrix} \mathbf{I}_3 & \dots & \mathbf{I}_3 \\ [\mathbf{p}_1 - \mathbf{p}_c] \times & \dots & [\mathbf{p}_4 - \mathbf{p}_c] \times \end{bmatrix}}_{\mathbf{A}} \mathbf{F} = \underbrace{\begin{bmatrix} m(\ddot{\mathbf{p}}_c + \mathbf{g}) \\ \mathbf{I}_G \dot{\boldsymbol{\omega}}_b \end{bmatrix}}_{\mathbf{b}}$$

其中  $\mathbf{p}_c$  是质心位置， $\mathbf{p}_1$  为腿 1 末端位置

期望的质心加速度用下式计算：

$$\begin{bmatrix} \ddot{\mathbf{p}}_{c,d} \\ \dot{\boldsymbol{\omega}}_{b,d} \end{bmatrix} = \begin{bmatrix} \mathbf{K}_{p,p}(\mathbf{p}_{c,d} - \mathbf{p}_c) + \mathbf{K}_{d,p}(\dot{\mathbf{p}}_{c,d} - \dot{\mathbf{p}}_c) \\ \mathbf{K}_{p,\omega} \log(\mathbf{R}_d \mathbf{R}^T) + \mathbf{K}_{d,\omega}(\boldsymbol{\omega}_{b,d} - \boldsymbol{\omega}) \end{bmatrix}$$

$$\mathbf{b}_d = \begin{bmatrix} m(\ddot{\mathbf{p}}_{c,d} + \mathbf{g}) \\ \mathbf{I}_G \dot{\boldsymbol{\omega}}_{b,d} \end{bmatrix}$$

则目标函数定义为：

$$\mathbf{F}^* = \min_{\mathbf{F} \in \mathbb{R}^{12}} (\mathbf{A}\mathbf{F} - \mathbf{b}_d)^T \mathbf{S}(\mathbf{A}\mathbf{F} - \mathbf{b}_d)$$

可以将对力大小、摩擦力等做出约束，同时摆动腿的末端力  $\mathbf{F}$  应为 0。需要注意，该目标函数写法只是为了看着直观，实际要用二次规划求解库例如 qpOASES 等解还得化为求解库的标准形式，下文将会详细讲这块。

代码实现具体过程：

qpOASES 求解的二次规划形式为：

$$\begin{array}{ll} \min_{\mathbf{x}} & \frac{1}{2} \mathbf{x}^T \mathbf{H} \mathbf{x} + \mathbf{x}^T \mathbf{g}(w_0) \\ \text{s. t.} & \text{lb} \mathbf{A}(w_0) \leq \mathbf{A} \mathbf{x} \leq \text{ub} \mathbf{A}(w_0), \\ & \text{lb}(w_0) \leq \mathbf{x} \leq \text{ub}(w_0), \end{array}$$

<sup>4</sup> Focchi M, Del Prete A, Havoutis I, et al. High-slope Terrain Locomotion for Torque-Controlled Quadruped. RoNguyen Q, Powell M J, Katz B, et al. Optimized Jumping on the MIT Cheetah 3 Robot.

因此上述的目标函数得化成求解库支持的标准形式,

$$\begin{aligned} & (AF - b_d)^T S (AF - b_d) \\ &= (F^T A^T - b_d^T) S (AF - b_d) \\ &= F^T A^T S A F - F^T A^T S b_d - b_d^T S A F + b_d^T b_d \end{aligned}$$

因为  $(F^T A^T S b_d)^T = b_d^T S^T A F$ , 而  $S$  矩阵为权重矩阵, 为对角矩阵, 本问题中取单位阵,  $\min F^T A^T S b_d$  就是个线性规划, 对线性规划来说,  $\min F^T A^T S b_d$  等效于  $\min (F^T A^T S b_d)^T$ , 而  $b_d^T b_d$  是个常数, 因此该问题最终带入 qpOASES 中  $H, g$  矩阵分别为:

$$\begin{aligned} H &= 2A^T S A \\ g &= -2A^T S b_d \end{aligned}$$

约束部分有:

- 1) 力上下限约束      2) 摆动腿末端力  $F = 0$       3) 摩擦力约束

提到摩擦力约束, 这里将论文中常见的 3 种摩擦力公式总结如下<sup>5</sup>, 实际为了计算方便, 常用线性摩擦力约束。

**friction cone**

$$\mathcal{C} = \left\{ (f_x, f_y, f_z) \in \mathbb{R}^3 \mid f_z \geq 0; \sqrt{f_x^2 + f_y^2} \leq \mu f_z \right\}$$

**pyramidal friction**

$$\mathcal{P} = \left\{ (f_x, f_y, f_z) \in \mathbb{R}^3 \mid f_z \geq 0; |f_x|, |f_y| \leq \frac{\mu}{\sqrt{2}} f_z \right\}$$

**linear friction**

$$\begin{aligned} f_{\min} &\leq f_z \leq f_{\max} \\ -\mu f_z &\leq \pm f_x \leq \mu f_z \\ -\mu f_z &\leq \pm f_y \leq \mu f_z \end{aligned}$$

至此, 这种方法已经可以写代码实现了。

### 2.2.3 基于单刚体模型的模型预测控制方法 (MPC)

MPC 与优化控制类似, 都是很多年前的产物, MIT 的 mini Cheetah 用 MPC

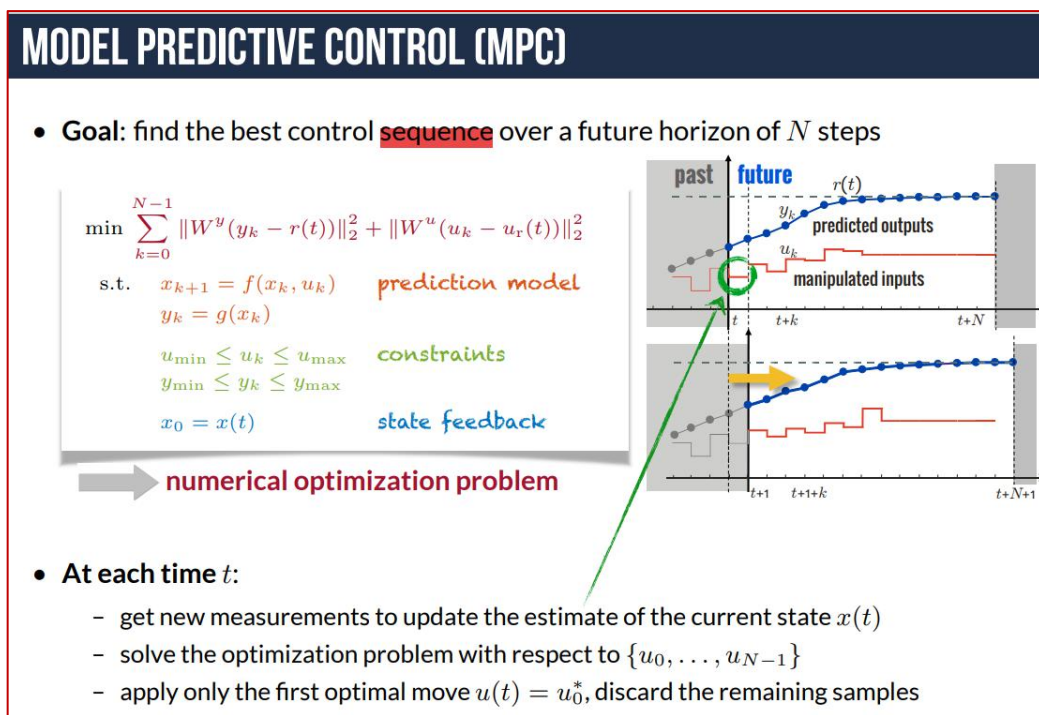
<sup>5</sup> Apgar T, Clary P, Green K, et al. Fast Online Trajectory Optimization for the Bipedal Robot Cassie[C]// Robotics: Science and Systems 2018.

实现了 trot 步态最高 2.4m/s<sup>6</sup>的速度，后 MPC 也引起了腿足界的关注。笔者上半年接触了段时间的 MPC，这里结合自身体验简单谈一下 MPC 如何用于四足。

### 1) MPC 简单介绍

为了读者更好的理解，首先简单的介绍下 MPC。此处用线性模型的 MPC 进行直观的理解，笔者觉得 MPC 说白了可以理解为已知一个系统的输入-输出模型方程，则根据初始状态  $x_0$  就可以按照模型表达式求出  $N$  个 MPC 周期后的系统状态，那么我就可以给定一段  $N$  个周期内的期望轨迹，例如  $N$  步内的位置、速度轨迹，然后依旧还是用二次规划逼近根据初始  $x_0$  算出的轨迹和期望轨迹，即求得  $N$  个周期里每个周期的解，然后让机器人执行第一个解序列即可。

核心点可用下图总结：



(摘自 Alberto Bemporad 教授 “Linear MPC” ppt 第 4 页)

这里可能有几个小细节有读者会感到疑惑，简单解释下。

1) MPC 有啥独特的优势？ 笔者觉得 MPC 最大的优势是基于当前状态求出未来一段时间内的系统状态，然后求出的最优解序列是考虑了一段时间的；而上面提到的直接优化法或者 VMC 只是求解当前时刻的最优解，只考虑了一个周期。

<sup>6</sup> Katz B, Carlo J D, Kim S. Mini Cheetah: A Platform for Pushing the Limits of Dynamic Quadruped Control[C]// 2019 International Conference on Robotics and Automation (ICRA). 2019.

2) 为什么用解序列中的第一个解? 笔者的理解是, 该解序列是基于初始  $x_0$  算出来的, 开始的控制输入相比后几步的, 是更关键的; 而且因为系统模型难免会有误差, 误差会累积, 再或者机器人中途受到扰动打乱了实际机器人状态, 所以需要尽力提高 MPC 频率, 就是为了可以在受到扰动时及时更新状态, 算出新一轮的最优解序列。综合来说, 取最优解序列第一个是更合理的。

3) 有什么好的参考资料吗? 笔者当初这块主要是看了下 Alberto Bemporad 教授的 ppt, Matlab MPC 工具箱主要贡献者就是他。ppt 链接:

[http://cse.lab.imtlucca.it/~bemporad/mpc\\_course.html](http://cse.lab.imtlucca.it/~bemporad/mpc_course.html)

## 2) 四足 MPC 代码实现

这部分的代码实现步骤笔者已经在以往的博客中详细说明了, 感兴趣的读者可以参考该博客:

<https://zhuanlan.zhihu.com/p/190044338>



### 3 引入全身动力学模型的控制

基于全身动力学模型的控制方法很多，有直接基于动力学模型进行优化的；也有像 MIT 那样将 MPC 作为基础控制器，然后在 MPC 求得的地面作用力  $F$  基础上引入全身动力学模型实现 Whole Body Control(WBC)。很多论文里都会提到 WBC，这里简要介绍下什么是 WBC。

《Humanoid Robotics: A Reference》中对 WBC 的定义如下：

WBC structures represent a wide range of sensor-based feedback control methods, exploiting the full mobility of the entire body of redundant, floating-base robots in compliant multi-contact interactions with the environment, consisting of task descriptors projected onto a robot's actuator space, thereby, to execute a coordinated behavior that usually combines mobility and manipulation.

说白了可以理解为四足/双足等腿足机器人是一个高度冗余的系统，总自由度数目为实际机构自由度+浮动基座的 6 自由度，然后四足机器人通常有多个控制任务——身体姿态平衡、身体高度、摆动腿等，相应的任务可以用雅可比矩阵将机器人的关节映射到这些属于操作空间的任務。然后就可以根据各个控制任务的重要性使更重要的任务有更高的优先级，后基于多任务优先级的零空间理论或者直接优化法求得满足任务优先级的关节位置、速度、加速度信息；最终基于动力学模型可求得各关节力矩。

实际 WBC 并不是什么新鲜的理论，最早在 1980 左右就开始用于机械臂控制了<sup>7</sup>，下面也将基于机械臂阐述相关计算和代码编写，四足机器人与机械臂相比无非就是自由度、控制任务的改变，计算理论依旧是共通的。

WBC 可以和传统的分解控制组合在一起理解，《legged robots that balance》中是将姿态、高度、摆动腿当作 3 个独立的控制任务，各自进行 PD 控制。据笔者的实践经验，实际它们是耦合的，这种分解控制的思想在低速情况下是 OK 的，但当速度上去之后，就会发现彼此的耦合干扰影响显著变大，单调 PD 参数已经不太管用了。而 WBC 就是重点考虑这种冗余、耦合、多任务的一种方法。

#### 3.1 基于零空间

有优先级的多任务零空间求解实质就是基于数学直接求出对任务满足程度按照优先级排列的解。详细推导可参考 ETH 动力学讲义第 44 页，这里只列出主

<sup>7</sup> Vukobratovic M, Kircanski M. A dynamic approach to nominal trajectory synthesis for redundant manipulators[J]. Systems Man & Cybernetics IEEE Transactions on, 1984, SMC-14(4):580-586.



要公式：

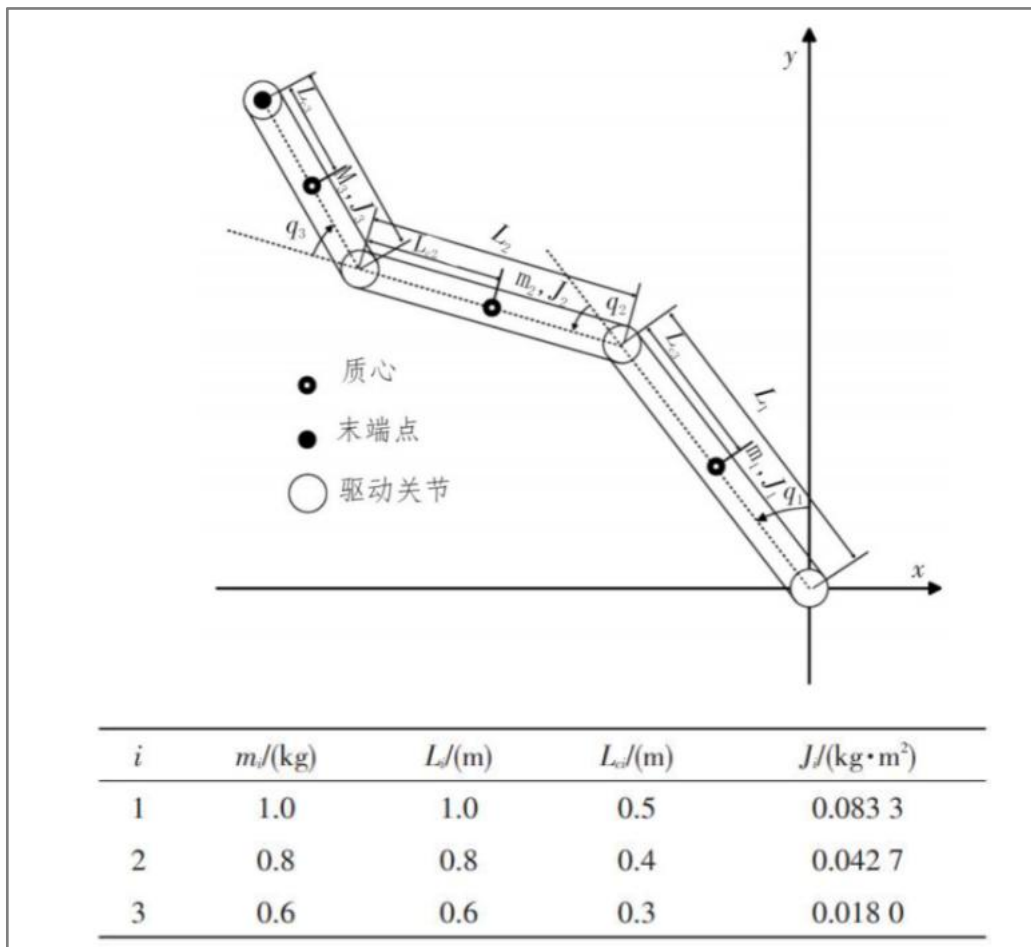
$$\dot{\mathbf{q}} = \mathbf{J}_1^+ \mathbf{w}_1^* + \mathbf{N}_1 (\mathbf{J}_2 \mathbf{N}_1)^+ (\mathbf{w}_2^* - \mathbf{J}_2 \mathbf{J}_1^+ \mathbf{w}_1^*)$$

注：此处的  $\mathbf{N}_1(\mathbf{J}_2 \mathbf{N}_1)^+ = (\mathbf{J}_2 \mathbf{N}_1)^+$ ，MIT 的开源代码中就省略了前面的  $\mathbf{N}_1$ ，为防止有读者感到困惑特此说明下。

$n$  个任务满足如下递推式子。

$$\dot{\mathbf{q}} = \sum_{i=1}^{n_t} \bar{\mathbf{N}}_i \dot{\mathbf{q}}_i, \quad \text{with} \quad \dot{\mathbf{q}}_i = (\mathbf{J}_i \bar{\mathbf{N}}_i)^+ \left( \mathbf{w}_i^* - \mathbf{J}_i \sum_{k=1}^{i-1} \bar{\mathbf{N}}_k \dot{\mathbf{q}}_k \right)$$

下面依旧以 1.6.3 中的机械臂例子具体说明。



假定控制任务为末端速度为  $\begin{bmatrix} 1 \\ 0.5 \end{bmatrix}$ ，且整个过程中要尽可能保证第二个关节

的速度为 0。则这就是一个多任务有优先级的问題。

任务	优先级	雅可比	末端期望状态
末端速度	1	正运动学求导	末端速度 $\begin{bmatrix} 1 \\ 0.5 \end{bmatrix}$
第二关节速度为 0	2	$\begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$	0

在 Matlab 中的计算结果如下：

```

%末端速度
w1 = [1; 0.5];
J1 = eval(subs(Jac, ...
    {l1,l2,l3, ...
    q1,q2,q3},...
    {1, 0.8, 0.6, deg2rad(30), deg2rad(45), deg2rad(75) ...
    }));
%第2关节速度=0
w2 = 0;
J2 = [0 1 0];
N1 = eye(3,3) - pinv(J1)*J1;
q_desired = pinv(J1)*w1 + pinv(J2*N1)*(w2 - J2*pinv(J1)*w1);
error = norm((q_desired - (pinv(J1)*w1 + N1*pinv(J2*N1)*(w2 - J2*pinv(J1)*w1))), 2);
error1 = norm((J1*q_desired - w1), 2);
error2 = norm((J2*q_desired - w2), 2);
fprintf('the error of add N1 or not is: %e\n',error)
fprintf('the error of endpoint_vel is: %e\n',error1)
fprintf('the error of add second joint vel is: %e\n',error)

```

命令行窗口

```

the error of add N1 or not is: 1.665335e-16
the error of endpoint_vel is: 0.000000e+00
the error of add second joint vel is: 1.665335e-16

```

可以看出采用零空间理论计算期望的关节速度时，会先满足优先级最高的任务，位置、加速度也同理。在得到位置、速度、加速度后，就可以将其带入动力学方程求得关节力矩实现多任务的 WBC 控制。

## 3.2 基于优化

零空间是基于数学理论直接求多任务的解，在实际中，也可以通过对不同的任务赋予不同的权重然后求得解。据笔者的经验来看，基于优化的 WBC 比较头疼的一点是调参比较麻烦，当控制任务变多，调各个控制任务的权重参数是比较费时间的。

依旧以上述的 3 自由度机械臂控制任务为例，此处采用优化的方法求两个任务情况下其期望的关节速度。

按照 3.1 中 Matlab 代码的各个量的定义,上述问题可转化为如下的优化问题:

$$\min (J\dot{q} - w_d)^T Q (J\dot{q} - w_d)$$

$$\text{where: } J = \begin{bmatrix} J1 \\ J2 \end{bmatrix}, w_d = \begin{bmatrix} w1 \\ w2 \end{bmatrix}$$

$Q$  即为权重矩阵,对优先级更高的任务赋予更大的权重数值。Matlab 的二次规划标准形式为:

#### 语法

```
x = quadprog(H,f)
x = quadprog(H,f,A,b)
x = quadprog(H,f,A,b,Aeq,beq)
x = quadprog(H,f,A,b,Aeq,beq,lb,ub)
x = quadprog(H,f,A,b,Aeq,beq,lb,ub,x0)
x = quadprog(H,f,A,b,Aeq,beq,lb,ub,x0,options)
x = quadprog(problem)
[x,fval] = quadprog(____)
[x,fval,exitflag,output] = quadprog(____)
[x,fval,exitflag,output,lambda] = quadprog(____)
```

#### 说明

具有线性约束的二次目标函数的求解器。

quadprog 求由下式指定的问题的最小值

$$\min_x \frac{1}{2} x^T H x + f^T x \text{ such that } \begin{cases} A \cdot x \leq b, \\ Aeq \cdot x = beq, \\ lb \leq x \leq ub. \end{cases}$$

将上述优化问题化为此标准形式:

$$\min (J\dot{q} - w_d)^T Q (J\dot{q} - w_d)$$

$$= \dot{q}^T J^T Q J \dot{q} - 2(J^T Q w_d)^T \dot{q}$$

$$\text{where: } J = \begin{bmatrix} J1 \\ J2 \end{bmatrix}, w_d = \begin{bmatrix} w1 \\ w2 \end{bmatrix}$$

设  $Q = \begin{bmatrix} 100 & 0 & 0 \\ 0 & 100 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ , 则将其带入 Matlab 求解可得:

```

- J = [J1; J2];
- W = [w1; w2];
- Q = [100 0 0;
      0 100 0;
      0 0 1];
- H = 2*J'*Q*J;
- f = -2*(J'*Q*W)';
- qdot = quadprog(H, f);
- error1 = norm((J1*qdot - w1), 2);
- error2 = norm((J2*qdot - w2), 2);
- fprintf('the error of endpoint_vel is: %e\n', error1)
- fprintf('the error of add second joint vel is: %e\n', error2)

```

命令行窗口

```

the error of endpoint_vel is: 5.698389e-13
the error of add second joint vel is: 4.201903e-11

```

可以看出求得的解优先满足权重更大的解。则将其扩展到腿足机器人的控制可以定义如下的优化问题<sup>8</sup>:

<b>minimize</b>	$\ A\ddot{q} + \dot{A}\dot{q} - \ddot{x}_{cmd}\ _W^2$	(25a)
$\ddot{q}, \tau, f$		
<b>subject to</b>	$M\ddot{q} + N_s^T h + \gamma = N_s^T S_a^T \tau + N_s^T J_c^T f$	(25b)
	$f \in \mathcal{P}^p.$	(25c)
	$S_f f = 0$	(25d)
	$\underline{\tau} \leq \tau \leq \bar{\tau}$	(25e)

二者基本道理是一样的，可以根据基础例子举一反三。

将此优化思想拓展到更广的范围，可以定义不同的目标函数，然后在动力学约束、摩擦力约束、力矩约束等你需要的约束下求得控制量从而控制机器人。笔者觉得这种纯优化的方法用起来不太舒服的地方主要是目标函数的设计，其次是调权重参数的确比较费时间，可能调很久都没调出让人满意的效果。

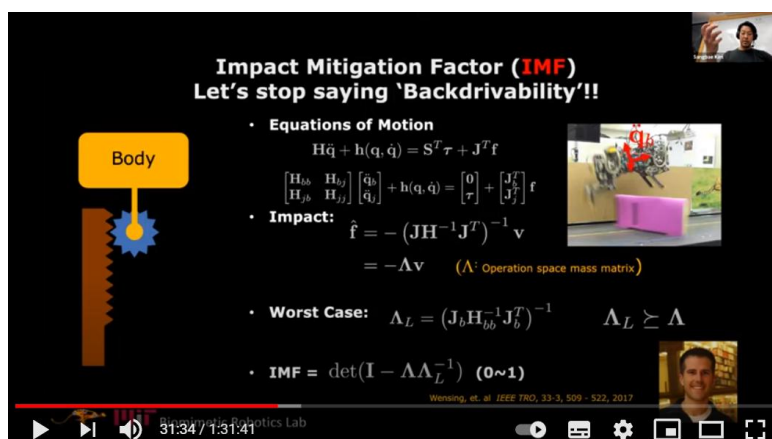
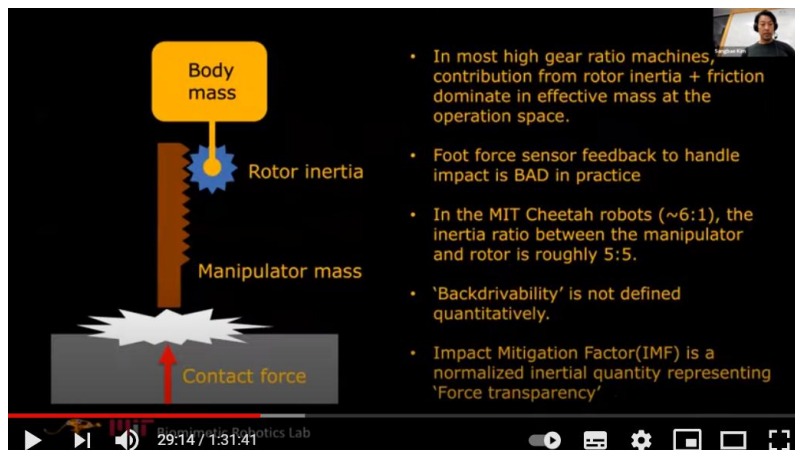
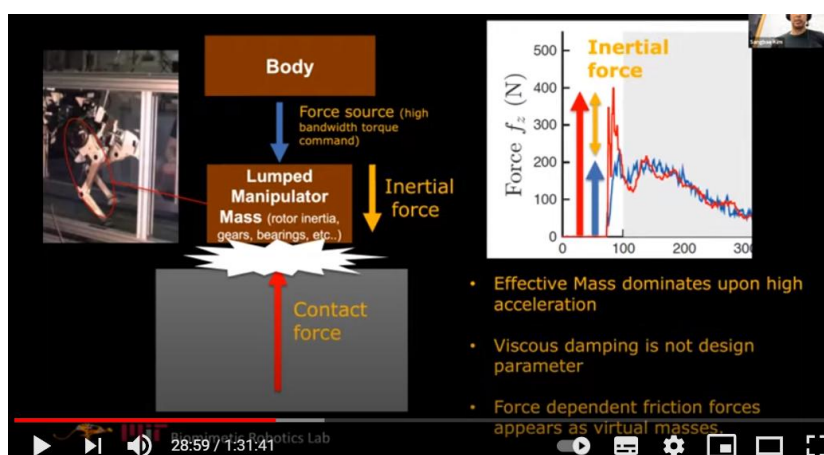
<sup>8</sup> Apgar T, Clary P, Green K, et al. Fast Online Trajectory Optimization for the Bipedal Robot Cassie[C]// Robotics: Science and Systems 2018.

## 4 一些与 MIT Cheetah 相关的算法小知识

这部分没有什么核心思想，主要就是将笔者觉得的一些有意思的与 MIT Cheetah 相关的算法知识罗列下。权且给自己记录下，方便日后查找。

### 4.1 关节设计思想

正如 Sangbae Kim 常在公开场合所讲的，如下截图所示。MIT 的高力矩密度+低减速比的关节驱动方案可以对四足机器人腿部与地面的碰撞起到很好的缓冲作用，并且也能实现基于电流的直接力控，很适合腿足机器人。



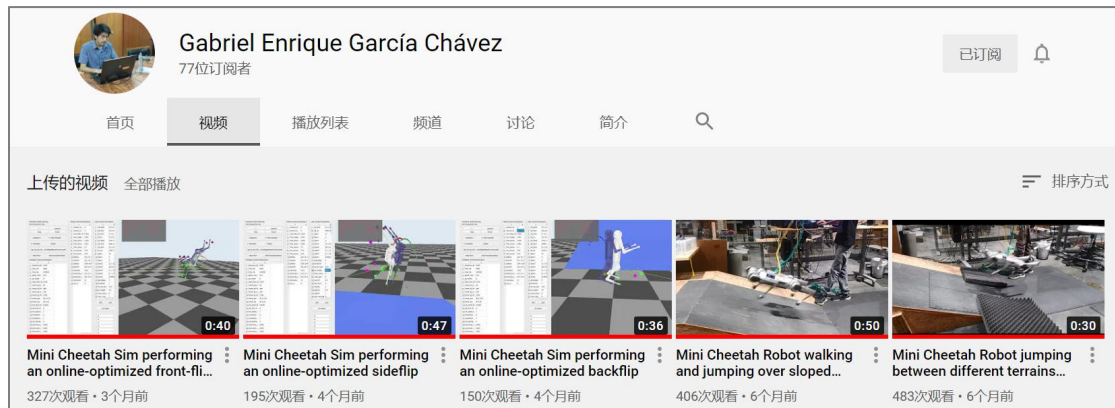
(摘自 <https://www.youtube.com/watch?v=9XTyWR1l-9E&t=2595s>)

详细论文为《Proprioceptive Actuator Design in the MIT Cheetah: Impact Mitigation and High-Bandwidth Physical Interaction for Dynamic Legged Robots》，感兴趣的读者可对照论文自行推导相关公式。

## 4.2 花式动作

在实际做四足机器人的倒地爬起、后空翻、侧空翻等花式动作时，根据笔者的实践经验，最简单粗暴的方法就是基于时间直接硬调轨迹，在哪个时间段执行一段什么样的位置轨迹，然后又硬调 PD，比如笔者曾硬调的四足“爬”过挡板（捂脸）：<https://b23.tv/4BSXZD>。总的来说，这种方法费时费力，且完全没有泛化能力，适合固定平台的单一任务。

而基于一定的算法理论去做花式动作的工作笔者目前主要注意到 IHMC 一位老哥，他是基于 mini cheetah 实现了一系列花式动作，视频号截图如下：

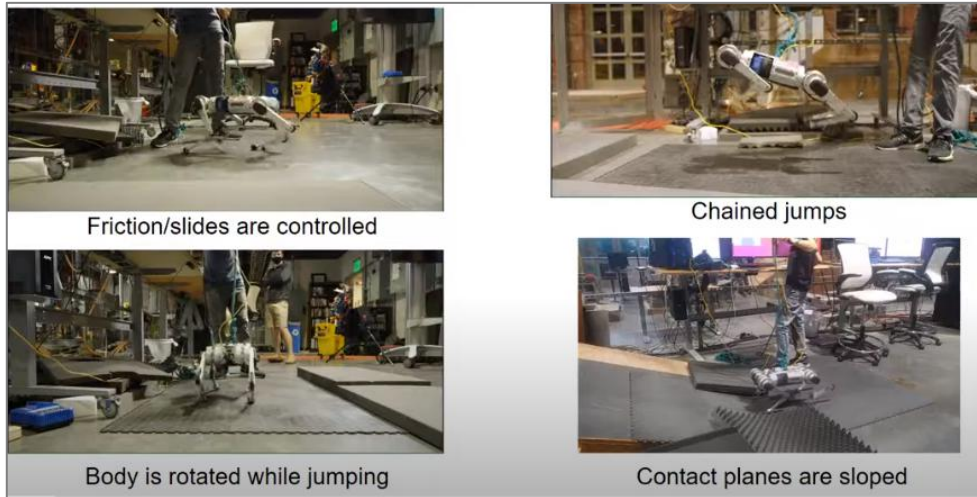


他自己所用的方案部分没有论文，但在 MIT Mini-Cheetah Workshop 2020 中他详细讲解了一些动作的技术点，完整视频见油管：

<https://www.youtube.com/watch?v=Rm2JMUBDUUnU&list=PLLIBX525Tpqjk2nhUir71HN-b7qK34EC&index=3&t=1108s>,

在 presentation 里他主要介绍了 4 种动作的技术方法，如下图所示：控制摩擦力使支撑腿打滑实现“太空步”的效果、连续跳跃、身体旋转跳跃、在斜面运动。





笔者这里暂且基于自己的理解将他的 presentation 总结一下，因为我还没有去复现他提到的方法，因此难免有不严谨错误之处，仅供大家参考。

#### 4.2.1 太空步

四足太空步过程中，支撑腿实际是打滑状态，需要控制支撑腿受到的摩擦力。视频作者将以往的 xyz 共 3 维的接触力换成：接触面法向力 \* 一个固定的向量，

按照作者意思，这个固定向量是由法向矢量（应该就是图中的  $\hat{n}$ ，推测是  $\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$  这

类矢量）和接触点瞬时速度决定的（图中的  $-\mu_k \hat{r}_p$ ，这一项笔者还没太搞懂……），然后动力学方程里的接触力项变成下图所示：

### Controlling friction

- After the jump, the robot achieved the desired final orientation, but it slipped down.
- Force applied by an sliding contact:
$$\mathbf{f} = \mathbf{f}_{\hat{n}} + \mathbf{f}_t = f_{\hat{n}} \hat{n} - \mu_k f_{\hat{n}} \hat{r}_p(\mathbf{q}, \dot{\mathbf{q}}) = f_{\hat{n}} (\hat{n} - \mu_k \hat{r}_p)$$
- Add a sliding contact to the dynamics. It applies a force  $\mathbf{f}$  at a point  $\mathbf{r}(\mathbf{q}) = \mathbf{r}_p$ :
$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \mathbf{S}\mathbf{r}_g + \frac{\partial \mathbf{r}^T}{\partial \mathbf{q}} (\hat{n} - \mu_k \hat{r}_p) f_{\hat{n}}, \mathbf{s}, \mathbf{t}$$

$$\mathbf{r}_{\hat{n}}(\mathbf{q}) = 0$$
- Note that only the normal component of  $\mathbf{r}(\mathbf{q})$  is a constraint, but the full jacobian of the contact is present in the dynamics.

ihmc

之后便将 MIT 的 WBC 方案里的多任务零空间的接触雅可比换成新的只考虑

法向力的雅可比。

### Controlling friction

- Adding a sliding contact to the WBC [1]

$$A \begin{pmatrix} \ddot{q}_f \\ \ddot{q}_j \end{pmatrix} + b + g = \begin{pmatrix} 0_6 \\ \tau \end{pmatrix} + J_c^T f_r, \quad (3)$$

where  $A$ ,  $b$ ,  $g$ ,  $\tau$ ,  $f_r$ , and  $J_c$  are the generalized mass matrix, Coriolis force, gravitation force, joint torque, augmented reaction force and contact Jacobian, respectively.  $\ddot{q}_f \in \mathbb{R}^6$

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) = S\tau_g + \frac{\partial \phi^T}{\partial q}(\hat{n} - \mu_k \hat{r}_p) f_{\hat{n}}$$

$$\Delta q_i = \Delta q_{i-1} + J_{i,pre}^{-1} (e_i - J_i \Delta q_{i-1}), \quad (16)$$

$$\ddot{q}_i^{cmd} = \ddot{q}_{i-1}^{cmd} + J_{i,pre}^{-1} (\ddot{x}_i^{des} - J_i \ddot{q}_{i-1}^{cmd}), \quad (17)$$

$$\ddot{q}_i^{cmd} = \ddot{q}_i^{cmd} + J_{i,pre}^{-1} (\ddot{x}_i^{cmd} - J_i \ddot{q}_i - J_i \ddot{q}_{i-1}^{cmd}), \quad (18)$$

where

$$J_{i,pre} = J_i N_{i-1}, \quad (19)$$

$$N_{i-1} = N_0 N_{10} \dots N_{i-10-2}, \quad (20)$$

$$N_0 = I - J_0 J_c^T, \quad (20)$$

$$N_{i-1} = I - J_{i-1}^{-1} J_{i-1} J_c^T.$$

Here,  $i \geq 1$ , and

$$\Delta q_0, \ddot{q}_0^{cmd} = 0, \quad (21)$$


$$\ddot{q}_0^{cmd} = J_0^{-1} (-J_0 \ddot{q}_0).$$

$\Rightarrow e_i$  is the position error defined by  $x_i^{des} - x_i$  and  $\ddot{x}_i^{cmd}$  is the acceleration command of  $i$ -th task defined by

$$\ddot{x}_i^{cmd} = \ddot{x}_i^{des} + K_p (\ddot{x}_i^{des} - \ddot{x}_i) + K_d (\dot{x}_i^{des} - \dot{x}_i), \quad (22)$$

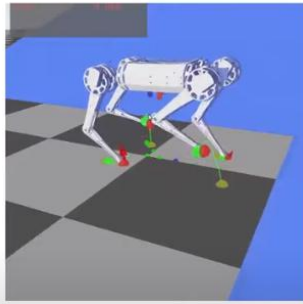
Only the normal components of the jacobians of the sliding contacts are used for defining  $J_c$

$$\phi_{\hat{n}}(q) = 0$$


 [1] D. Kim, J. Di Carlo, B. Katz, G. Bledt, and S. Kim "Highly Dynamic Quadrupedal Locomotion via Whole-Body Impulse Control and Model Predictive Control," in arXiv preprint arXiv:1909.0658, Sep 2019

最终通过控制支撑腿的打滑实现了仿真和实物的“太空步”运动，部分截图如下：

- Adding a sliding contact to the WBC [1]
- Contact sequence for each leg is:  
 $SC \rightarrow FC \rightarrow SC \rightarrow SW \rightarrow FC$
- FC: Fixed Contact
- SC: Sliding Contact
- SW: Swing Trajectory



### Testing with friction control

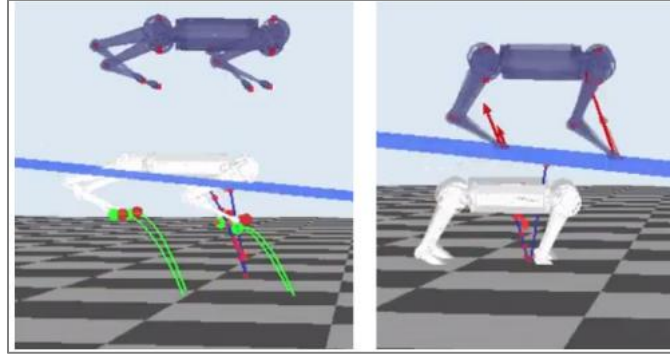


Contact sequence for each leg is: Slide → Fix → Slide → Swing → Fix

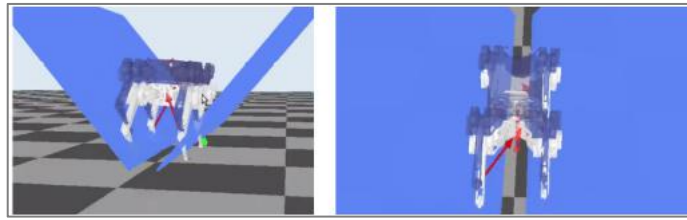
#### 4.2.2 跳上斜面/V 型板行走/两个斜面间跳跃

这一部分在原作者的《Time-Varying Model Predictive Control for Highly Dynamic Motions of Quadrupedal Robots》中有详细的讲解。主要实现了跳上斜面，或者在 V 型板上行走，或者在两个斜面间跳跃。

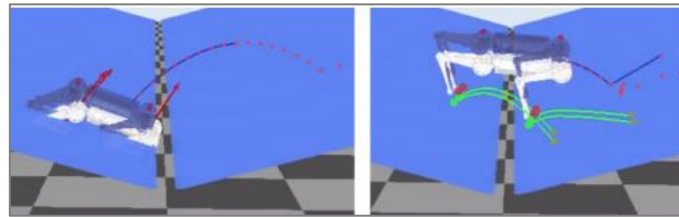




跳上斜面



V 型板行走



两个斜面间跳跃

控制方法是在 MIT 的 Cheetah 开源代码的 MPC 基础上改的。前提条件主要改变点总结如下：

- 1) 不再将 roll、pitch 假定为 0，使机器人可以在任意方向旋转
- 2) 不再将 roll、pitch 速度假定为 0，使机器人可以快速旋转

基于这两点前提条件的改变，将 MIT 原有的 MPC 模型做出如下改变：

- 1) 旋转矩阵改用四元数表示，避免奇异点
- 2) 不再忽略陀螺力矩  $w \times Iw$ ，将单刚体模型改为如下形式：

$$\begin{bmatrix} \ddot{\mathbf{r}} = \sum_{i=1}^n \mathbf{f}_i + \mathbf{g} \\ \dot{\mathbf{L}}_2 = \sum_{i=1}^n \mathbf{r}_{pi} \times \mathbf{f}_i + m\mathbf{r} \times \mathbf{g} \\ \dot{\mathbf{q}}_r = \mathbf{Q}(\mathbf{q}_r)\mathbf{I}^{-1}(\mathbf{q}_r)(\mathbf{L}_2 - m\mathbf{r} \times \dot{\mathbf{r}}) \end{bmatrix}$$

状态变量取  $[\mathbf{r}, \dot{\mathbf{r}}, \mathbf{L}_2, \mathbf{q}_r]^T$

- 3) 因为是在斜面上，所以摩擦力约束引入一个平地到斜面的变换阵：

$$\begin{aligned}
& -\mu f_z \leq f_x \leq \mu f_z \\
& -\mu f_z \leq f_y \leq \mu f_z \\
& 0 \leq f_z \leq f_{zmax} \\
& \mathbf{W}_i \mathbf{f}_i \leq \mathbf{b}_i \\
& \downarrow \\
& \mathbf{W}_i \mathbf{R}_{plai}^T \mathbf{f}_i \leq \mathbf{b}_i
\end{aligned}$$

这部分代码作者开源了，详见：

<https://github.com/GabrielEGC/IHMC-Robotics/tree/master/MIT%20Mini-Cheetah>