

# Pinocchio库使用说明——以双足模型为例

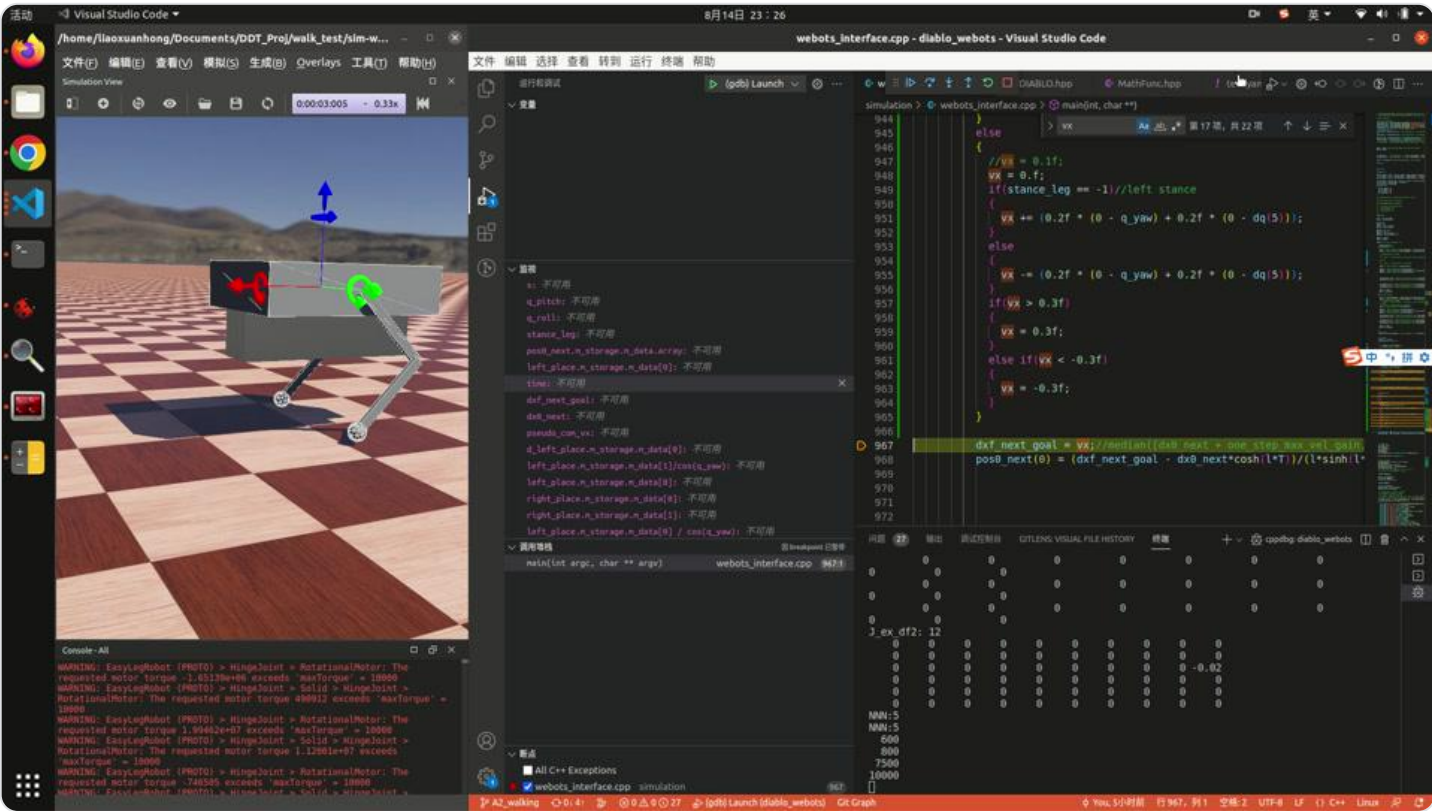
此文档同时关联以下文档：

[Cassie HZD双足控制方法说明](#)

以及以下project:

[https://git.ddt.dev:9281/rbt/alg/pinnocchio\\_bipedal\\_example](https://git.ddt.dev:9281/rbt/alg/pinnocchio_bipedal_example)

将搭建一个简易的双足模型，并使用Cassie的HZD方法进行控制，最终效果如下视频所示



## 背景

### 为什么要用动力学库

我们常常在论文中能看到以下形式的动力学模型：

$$M(q)\ddot{q} + C(q, \dot{q}) + G(q) = \tau$$

上述为机器人的逆动力学模型，他描述了机器人广义坐标下的加速度与关节力矩之间的关系。简而言之，如果我们拥有了机器人当前的“配置”（广义位置 $q$ 、广义速度 $\dot{q}$ ），给定广义加速度 $\ddot{q}$ ，即可算出广义力 $\tau$ 。这个 $\tau$ 可以作为我们的机器人前馈项代入到控制器当中，从而获得更快的关节响应。因而，我们要做的事情则是想办法获得这个动力学模型，也就是去求解动力学模型。

求解机器人的动力学方法有两种：推算解析解 或者 使用牛顿欧拉法求解。

使用解析解的优点在于：速度快，有效避免矩阵求逆带来的计算量激增。如果拥有了动力学模型的解析解，通过一定的简化后，我们甚至可以在一块单片机上实现动力学算法。这将大大降低我们的产品成本。

使用牛顿欧拉法求解的优点在于：形式简单，可以做成一种通用的方法，对不同的模型进行求解。在学术研究中，

使用动力学库能帮助我们快速验证某一算法的可行性。由于并非所有的物理模型都能快速得到动力学模型的解析解，往往需要经过大量的验证和测试，甚至还会存在隐式解的情况。因而使用牛顿欧拉法求解便成了很多工程与科研工作的首选。然而，手写牛顿欧拉法也要遵照一定的基本规则以及写法，否则会导致运算量激增，有一定的难度。因此出现了很多开源的动力学库，如Pinocchio（本文着重介绍），Frost（Cassie用的动力学库，Mathematica的一个库），RBDL等，**来帮助我们更快地地上手机器的动力学控制。**

动力学模型的数值求解是一门学问，较为有名的是Roy Featherstone 的神书 Rigid Body Dynamics Algorithms，能看到很多的动力学库都参考了此书进行编写，并进行了算法优化，**这些库的运算速度会比我们自己手写的可能更快。**这也是为什么我们要使用动力学库的另一个重要原因。

此外。使用动力学库，一方面能帮助我们快速地推进对某个模型的研究，另一方面还可以辅助我们来验证我们自己写的动力学模型的解析解是否确实可行，是否有纰漏的地方。

## 为什么要用Pinocchio

参考回答：<https://www.zhihu.com/question/437857717>

另外这篇[paper](#)也对pinocchio做各种类型的动力学计算的时候的表现做了测算。

总的来说使用pinocchio的原因有2点：

1. 容易安装
2. 算得快，因而容易部署。
3. 支持好。Pinocchio同时支持c++和python。此外，网上学习的资料也比较多。
4. 开源协议是BSD-2 clause的，可以进行商用。

推荐在以下网站查询pinocchio相关的资料：

[Pinocchio的官方documentation](#)。个人认为官方的documentation写的一般，但能帮助我们快速找到对应的类中（比如modelTpl、dataTpl）的成员以及用例。

[Pinocchio的官方github](#)。在使用pinocchio库过程中遇到的大部分具体到案例的问题，都可以通过查询pinocchio的issue来找到答案，最早的回答可以追溯到2019年之前，很多基本的动力学问题，比如浮动基座计算、雅格比矩阵计算、逆运动学等，都可以在上面找到答案。但相对来说信息比较零散。本文档会对其中一些信息进行探索

## 环境配置

### Pinocchio安装

一般有两种安装方式：用ROS安装或者直接安装。直接安装的话参考以下网址：

<https://stack-of-tasks.github.io/pinocchio/download.html>

我在安装的时候出现了一直卡着的情况，这个时候中断安装，尝试一下**sudo apt upgrade**升级一下自己的包，然后再重新安装。

pinocchio的软件包安装路径为：`/opt/openrobots/./pinocchio/`，在这里你可以查看所有的pinocchio包的头文件以及库文件。

在使用makefile编译pinocchio的时候，记得在Makefile文件中添加库文件libpinocchio.so以及头文件的文件夹

```
1 LIBRARIES += -L"/opt/openrobots/lib" -lpinocchio#libpinocchio.so
2 INCLUDE += -I"/opt/openrobots/include"
```

同时，为了使pinocchio库生效，打开以下文件夹，并添加库文件路径。

```
1 nano /etc/ld.so.conf
2 > /opt/openrobots/lib
```

保存并关闭后，使用如下指令使得ld.so.conf生效：

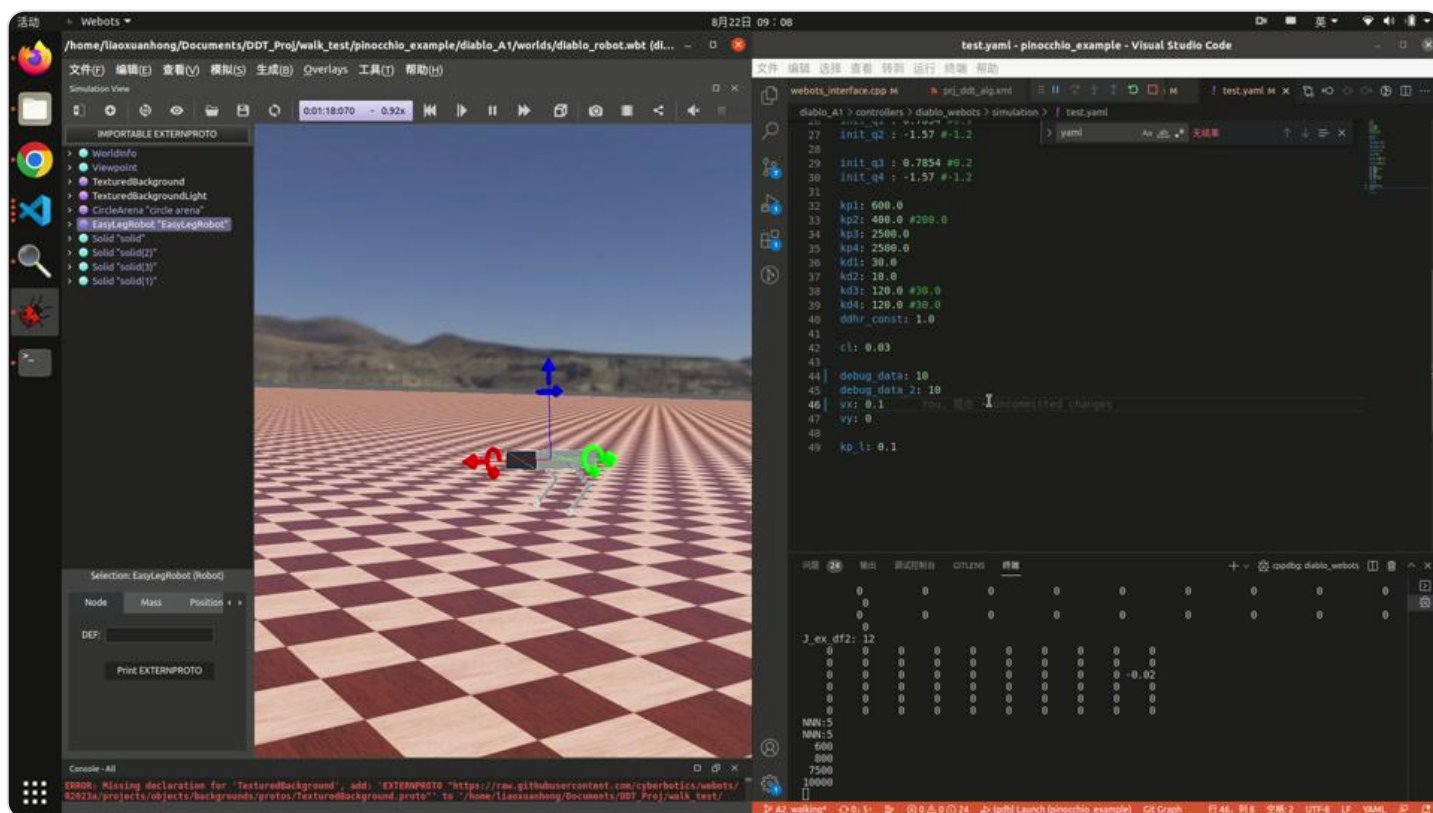
```
1 sudo ldconfig
```

### Eigen库安装

Eigen库的安装在此不赘述，参考[这个链接](#)。

## yaml-cpp安装

yaml是一种较为人性化的数据序列化语言，可以简单理解为一种格式化的txt。使用yaml-cpp可以直接读取.yaml中的具体内容，与cpp程序进行交互，且避免了.hpp或者宏定义的缺点：每次修改后都要对程序重新进行编译。在仿真中还可以做到实时更新控制参数，效果如下。



yaml-cpp的安装和编译参考[这里](#)。同样的，要记得将yaml的库文件和头文件包含在Makefile路径中。

```
1 LIBRARIES += -L"/opt/openrobots/lib" -lyaml-cpp#libyaml-cpp.so
2 INCLUDE += -I"/usr/include/yaml-cpp"
```

## Gnuplot安装

如果要使用gnuplot进行绘图，需要先安装gnuplot，参考[这里](#)

## 以双足控制为例

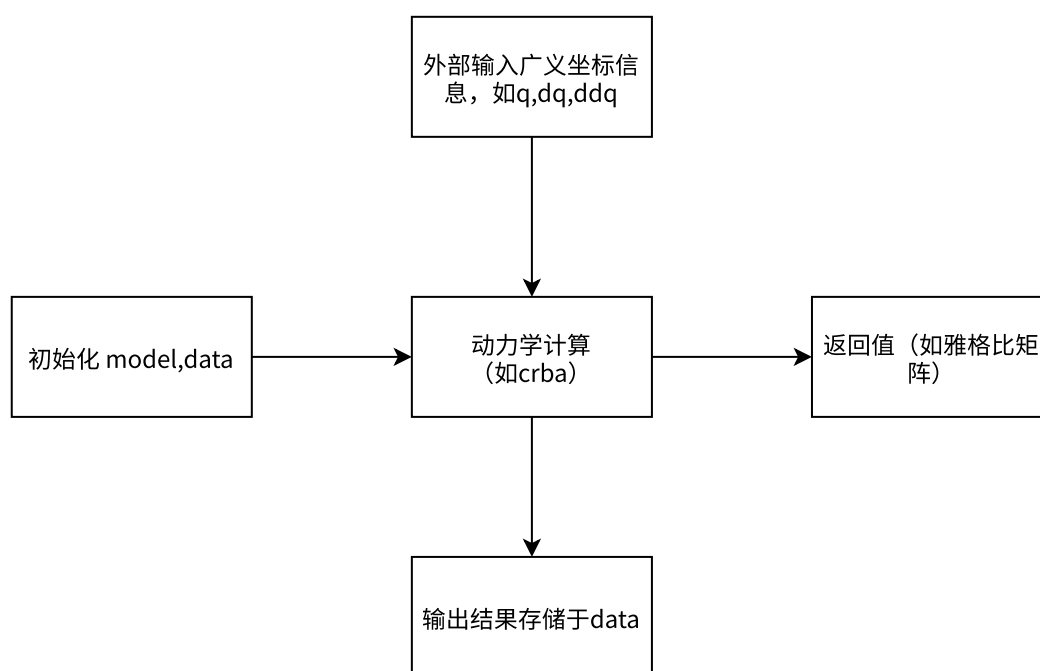
### 快速使用

默认大家已经大致掌握了如何使用webots。

#### 1. 克隆工程

2. 打开webots，打开diablo\_robot.wbt。
3. 打开vscode工程。
4. 修改launch.json中的"programm"，路径更改为你路径下的可执行文件"diablo\_webots"
5. 修改yaml\_addr为test.yaml所在的地址
6. 修改urdf\_addr为easy\_leg\_robot.urdf所在的地址
7. 按F5运行脚本，在main函数的case 2中打一个断点。
8. 待机器人运行到断电后，在webots界面中移开机器人正下方物块。
9. 取消断点后继续运行。能看到机器人在仿真中运行起来，具体效果看第一章节。

## workflow

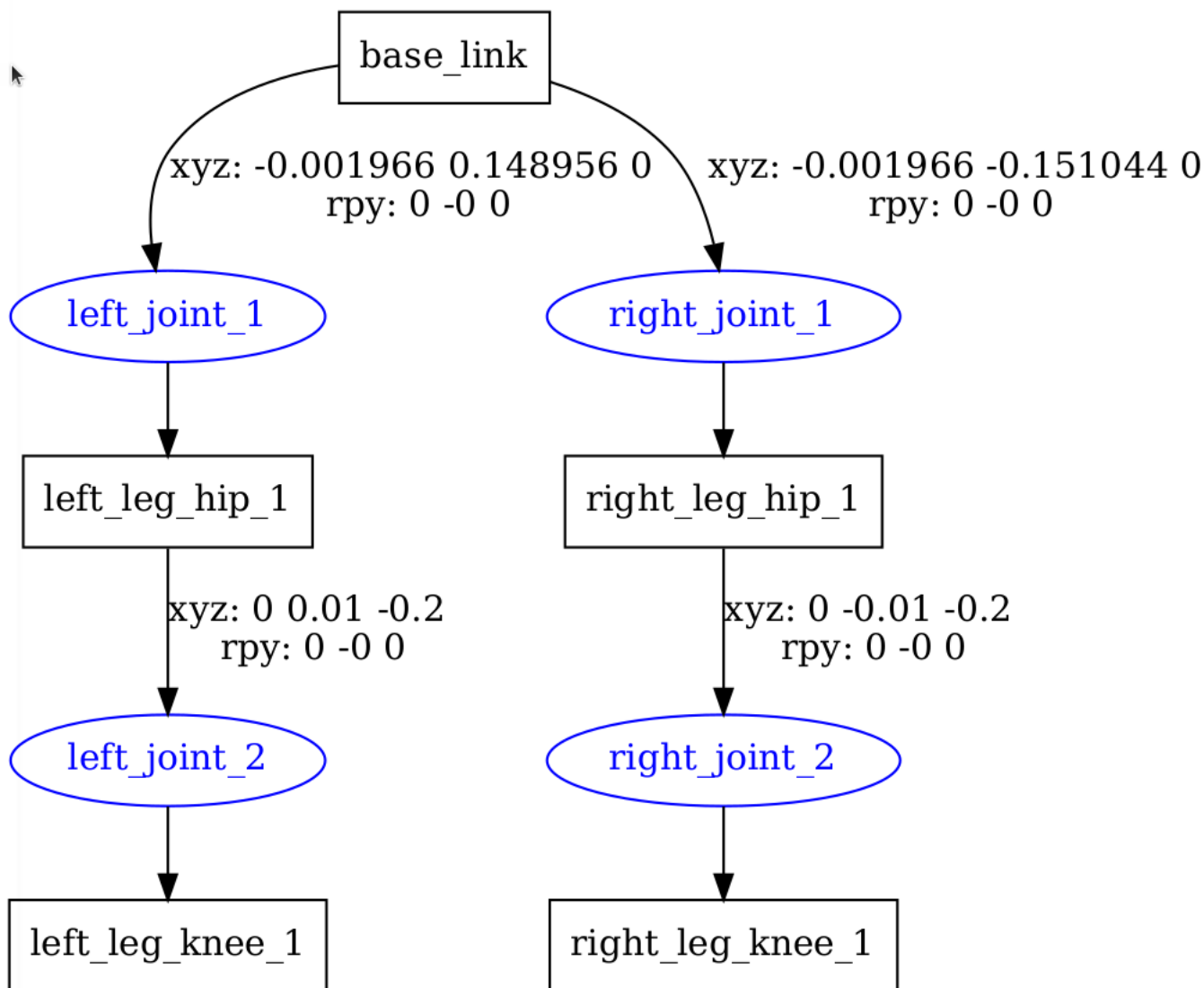


## 关键变量

- |                  |                                       |
|------------------|---------------------------------------|
| 1 ModelTpl model | 模型，一般用于存储模型的基本信息，如广义坐标数量，杆件质量，等       |
| 2 DataTpl data   | 数据，一般用于存储动力学计算后的结果，如雅格比矩阵、科氏矩阵、惯量矩阵等。 |

## 模型说明

我们建立的是一个最简单的双足模型：一个基座和两个串连的二连杆，如下图所示：



## 浮动基座模型建立

关于浮动基座模型的相关知识可以参考 [Cassie HZD双足控制方法说明](#) 以及 [ETH动力学讲义](#)。简而言之，浮动基座模型就是把基座的x,y,z,roll,pitch,yaw也作为广义坐标的一部分，对机器人进行建模。使用浮动基座进行建模，能够让我们更好地对机器人的基座（base）进行约束。但我们在建urdf模型的时候，往往并没有考虑这个模型是浮动基座还是固定基座。万幸的是，pinocchio提供了让我们在固定基座和浮动基座间切换的方法。

pinocchio的浮动基座模型建立方法为：

```

1  pinocchio::JointModelFreeFlyer root_joint;
2  pinocchio::urdf::buildModel(urdf_filename,root_joint,model);

```

而固定基座模型的建立方法为：

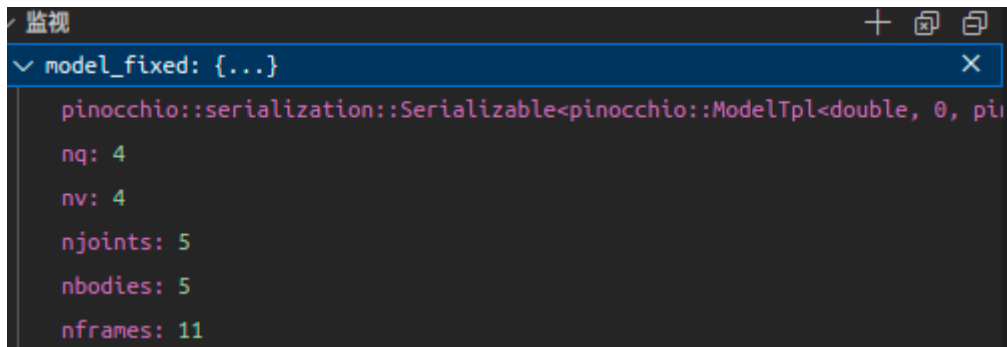


```
1 pinocchio::urdf::buildModel(urdf_filename,model);
```

可以看到，在浮动基座模型建立的时候，会多输入一个JointModelFreeFlyer 类型的root\_joint，将我们建立的模型标注为浮动基座模型。

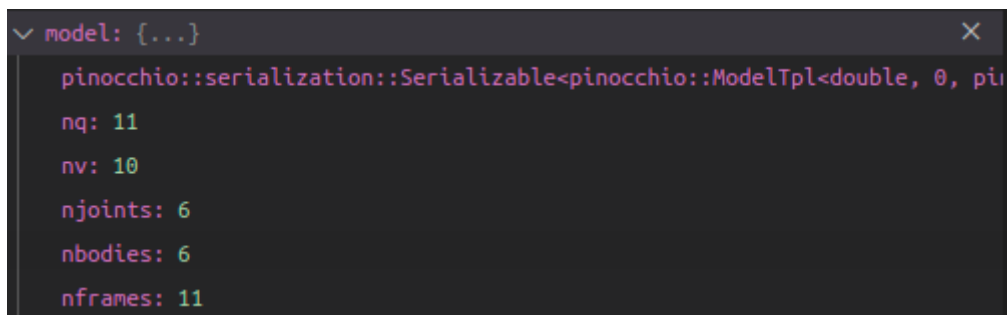
通过观察model.joints，能够观察到两种构建方式的广义坐标数量是不同的。

对于固定基座：



```
监视
model_fixed: {...}
  pinocchio::serialization::Serializable<pinocchio::ModelTpl<double, 0, pi
  nq: 4
  nv: 4
  njoints: 5
  nbodies: 5
  nframes: 11
```

对于浮动基座：



```
监视
model: {...}
  pinocchio::serialization::Serializable<pinocchio::ModelTpl<double, 0, pi
  nq: 11
  nv: 10
  njoints: 6
  nbodies: 6
  nframes: 11
```

其中 `model.nq` 文档定义为：“Dimension of the configuration vector representation”，意思是配置向量的维数,实际上就是广义坐标向量 $q$ 的维数。类似的，`model.nv` 为广义速度向量 $dq$ 的维数。对于easy\_robot，关节数量为4，算上浮动基座的维数为6（x y z,roll pitch yaw）,广义坐标向量的维数应该为10。对于固定基座，广义坐标向量的维数等于关节向量的维数，应该为4。

而在pinocchio中，情况略有不同。对浮动基座。由于使用了四元数w,x,y,z来代替roll,pitch,yaw，因此 `model.nq = 11`。而对于广义速度，前六个速度为 dx,dy,dz,droll,dpitch,dyaw，因此，`model.nv = 10`。

固定基座则沿用上一段的分析，`model.nq=model.nv=4`。

此外，在浮动基座中，`model.njoints=6`；在固定基座中，`model.njoints=5`。他们都比关节的实际数量：4，要更大。这是因为，两者都把世界“universe”算作了一个joint。在浮动基座中，则又添加了一个root\_joint。

在更新浮动基座的模型的各项广义坐标时，如果我们的代码里面对于质心的相对于世界坐标系的速度 dx dy dz和x y z进行了估计，则可以同步更新到广义坐标 $q$ 与 $dq$ 中。但在大多数情况下，我们可能不能

直接获得 $x y z$ 与 $dx dy dz$ ，此时我们直接把上述数值设置为0即可。将上述广义坐标设置为0的代价是，当我们想要使用 `computeCentroidalMomentum` 计算基座质心/几何中心的动量的时候，会认为当前质心的线速度接近于0，因而对应的线动量 `m*c_dot` 也会接近于0。pinocchio的在计算正逆运动学的过程中也不会帮我们推算质心的线速度。

对于其他的广义坐标，四元数部分直接从imu处获取，关节广义坐标直接从关节传感器中获取。

```
1  qua = wb_inertial_unit_get_quaternion(IMU);
2  accl = wb_accelerometer_get_values(ACCL);
3  gyro = wb_gyro_get_values(GYRO);
4  //Quaternion update
5  q(0) = 0;
6  q(1) = 0;
7  q(2) = 0;
8  for(int i = 0; i < 4; i++)
9  {
10     q(i+3) = qua[i];
11 }
12 //leg update
13 dq(0) = 0;
14 dq(1) = 0;
15 dq(2) = 0;
16 dq(3) = gyro[0];
17 dq(4) = gyro[1];
18 dq(5) = gyro[2];
19 for(int i = 0; i < 4; i++)
20 {
21     q(i+7) = position[i];
22     dq(i+6) = velocity[i];
23 }
```

## 连杆末端坐标系(frame)插入

pinocchio在建立模型的时候，其模型的末端只会建立到最末端的关节（joint），而不会建立到连杆的末端。因此，如果我们想通过正运动学方法 `forwardKinematics` 来计算末端位置、速度和雅格比矩阵的时候，就只能计算到末端的joint。为了计算连杆的末端，需要给模型插入连杆末端的坐标系。代码如下所示：

```
1  Eigen::Matrix3d Id = Eigen::Matrix3d::Identity(3,3);
2  Eigen::Vector3d zd(3);
3  zd << 0.0,0,-0.2;
4
5  pinocchio::SE3 left_se3(Id,zd); //构建SE3实例
```



```

6  pinocchio::Frame
   left_frame("left_toe",model.getJointId("left_joint_2"),model.getFrameId("left_j
   oint_2"),left_se3,pinocchio::FIXED_JOINT);
7  pinocchio::Frame
   right_frame("right_toe",model.getJointId("right_joint_2"),model.getFrameId("rig
   ht_joint_2"),left_se3,pinocchio::FIXED_JOINT);
8
9  model.addFrame(left_frame);
10 model.addFrame(right_frame);

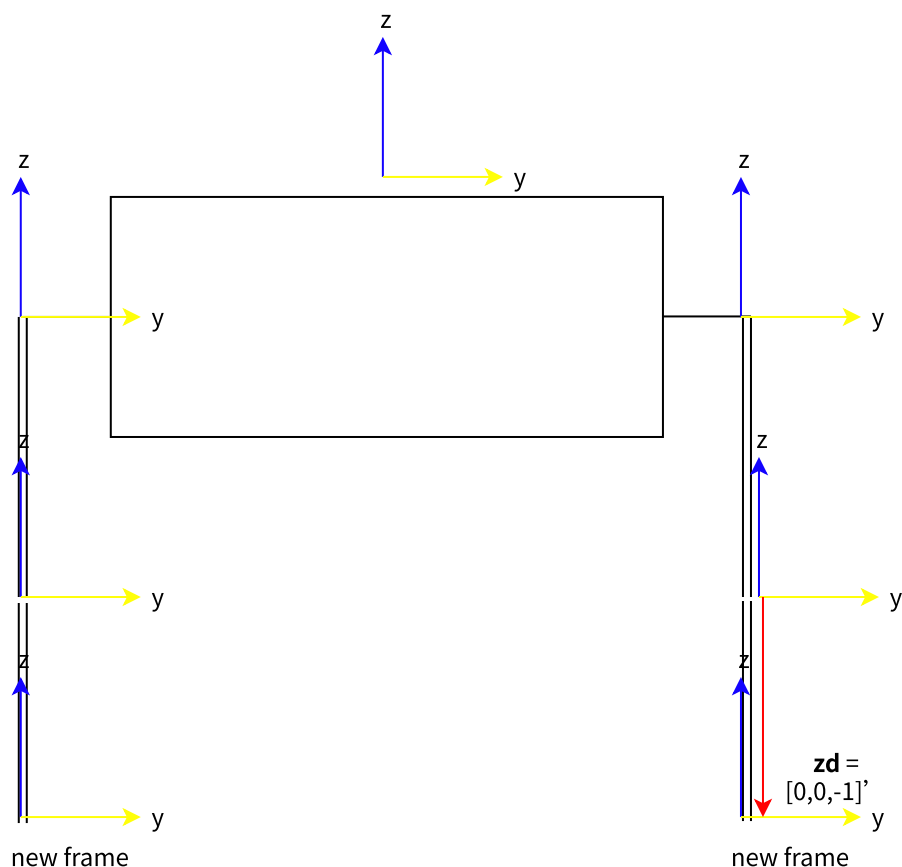
```

SE3矩阵是一个4x4的矩阵，如下图所示。其中R.block(0,0,3,3) 是3x3的旋转矩阵，R.block(0,3,3,1)是3x1的位移向量。

$$\begin{bmatrix} c & s & 0 & x \\ -s & c & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

通过 `Frame` 构建坐标系的时候，第一个形参为新坐标系的命名，第二个形参为该坐标系的父关节坐标系joint的ID，第三个形参为该坐标系的父坐标系frame的id，第四个形参是新坐标系相对于父坐标系的位置变换，第五个形参的含义是新坐标系随着父坐标系发生变化的。

在easy\_robot中，坐标系如下图所示。zd代表新坐标系相对于旧坐标系之间的平移。



在插入新的坐标系后可以看到，nframes的数量从11增加到了13，新增了两个坐标系。

```

20. {...}
监视
model: {...}
pinocchio::serialization::Serializable<pinocchio::ModelTpl<double, 0
nq: 11
nv: 10
njoints: 6
nbodies: 6
nframes: 13

```

## 获取浮动基座下的雅格比矩阵

浮动基座下，雅格比矩阵依旧符合原始定义，即：

$$p = h(q)$$

$$v = \frac{\partial h(q)}{\partial q} = J\dot{q}$$

其中， $h(\cdot)$ 描述的是广义坐标和末端坐标之间的关系，也可以简单称为基座原点和末端之间的关系。可以看到，雅格比矩阵右乘广义速度坐标后，依然是等于末端速度的。但由于我们有两个末端（也就是两条腿），因此，从基座到左腿的雅格比矩阵JF\_l，和基座到右腿的雅格比矩阵JF\_r是不一样的。具体的区别大家可以运行过程中print出来看一看。

因此，我们在实际使用过程中，需要将两个雅格比矩阵分离出来。在pinocchio中的做法如下：

1. 调用函数 `computeJointJacobians(model,data,q)`

2. 获取基座到左腿的雅格比矩阵

```

getFrameJacobian(model,data,model.getFrameId("left_toe"),pinocchio::
LOCAL_WORLD_ALIGNED,JF_l)

```

以及右腿的雅格比矩阵

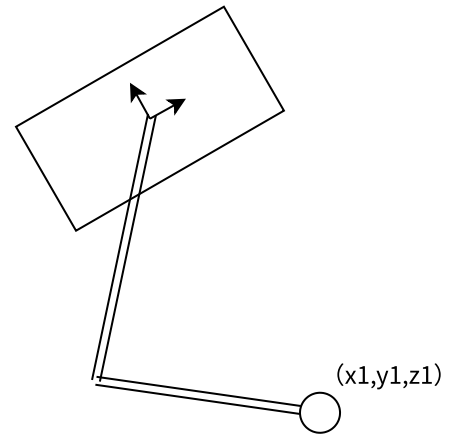
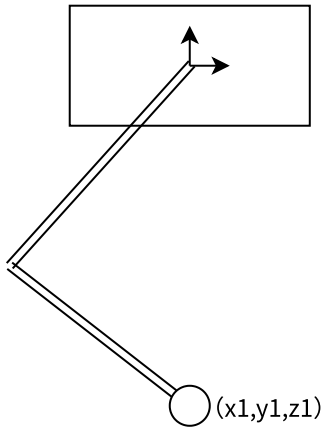
```

getFrameJacobian(model,data,model.getFrameId("right_toe"),pinocchio::LO
CAL_WORLD_ALIGNED,JF_r)

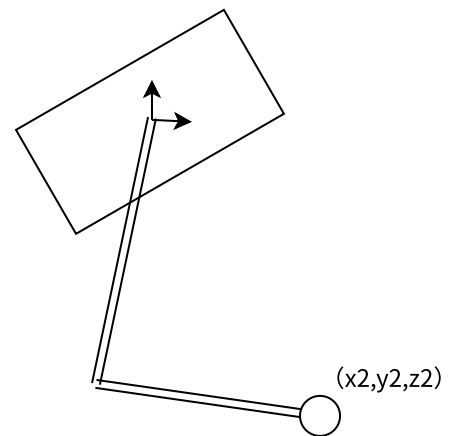
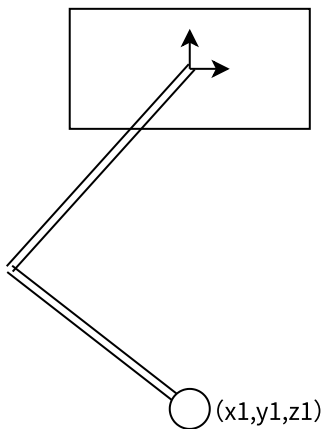
```

这里需要注意，其中一个形参为 `pinocchio::LOCAL_WORLD_ALIGNED`，其实这里还有两个选项 `pinocchio::WORLD` 以及 `pinocchio::LOCAL`。前者的意思是雅格比矩阵是相对于世界坐标系 universe建立的，换言之其雅格比矩阵是相对世界坐标原点建立的。在github中有提过，`WORLD` 一般不常用。而 `LOCAL` 的意思则是相对于基座建立雅格比矩阵，这和相对**基座原点**建立雅格比矩阵是有区别的；在固定基座下，两者的效果相同，因而基座原点会随着基座变化而变化，比如整个机器人绕着原点发生旋转，末端坐标不会随着机器人旋转而发生变化；但在浮动基座下，如果整个机器人绕着原点发生旋转，末端相对于原点的坐标系会发生变化。参考下图。

## Fixed base and LOCAL Jacobian



## Floating base and LOCAL\_WORLD\_ALIGNED Jacobian



## 逆动力学模型关键计算

`crba(model, data, q)` : crba方法计算惯量矩阵 $M(q)$ ，计算结果存储在`data.M`中。pinocchio中为了简化 $M(q)$ 的计算，只算了惯量矩阵的上三角部分，这是因为 $M(q)$ 本身就是一个对称矩阵。要获得完整的矩阵惯量矩阵，可直接把上三角矩阵复制到下三角即可：

```
1 data.M.triangularView<Eigen::StrictlyLower>() =  
  data.M.transpose().triangularView<Eigen::StrictlyLower>();
```

`computeCoriolisMatrix(model, data, q, dq)` : 计算科氏矩阵 $C$ ，计算结果存储在`data.C`中。

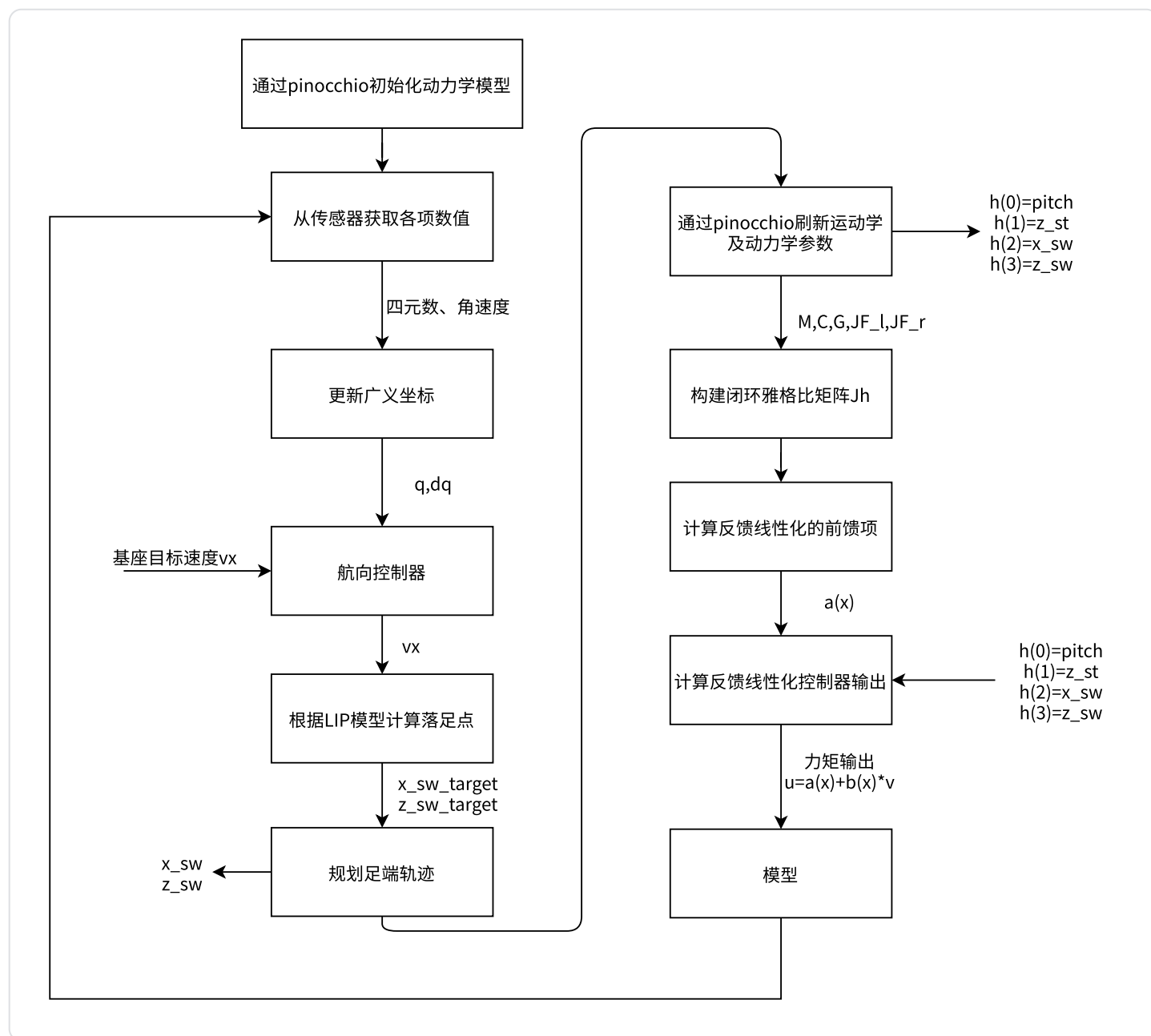
`computeGeneralizedGravity(model, data, q)` : 计算重力补偿向量 $G$ ，计算结果存储在`data.g`中。

`updateFramePlacements(model,data)`：刷新每一个坐标系的状态。由于我们插入了新的坐标系left\_toe和right\_toe，因此在计算前向运动学之前，需要通过该函数更新所有frame的状态。

`forwardKinematics(model,data,q)`：顾名思义，计算整机的运动学状况。

## 双足控制模型

和Cassie一样，基于混合零动态理论（Hybrid Zero Dynamics），根据机器人的动力学模型，进行机器人的全阶控制。具体控制理论参考文章[Cassie HZD双足控制方法说明](#)。以下给出了整个控制流程：



在做计算的过程中实际上还做了一些处理和简化：

1. 没有计算雅格比矩阵的导数dJ，这项简化在Cassie的代码中也有体现，目前看来对控制的影响不大
2. 多了个旋转矩阵Rb，如下所示：

```

1    q_euler_convert.toEulerAngle(&q_roll,&q_pitch,&q_yaw); //注意q是四元数，要转换
    成欧拉角q_euler_convert
2    q_euler_convert.toRotationMatrix(rotate);
3    //此处用for赋值的方法比较低效，Eigen应该有更好的做法
4    for(int i = 0; i < 3; i++)
5    {
6        for(int j = 0; j < 3; j++)
7        {
8            Rb(i,j) = rotate[i][j];
9        }
10   }

```

这个Rb矩阵的用途是处理腿长和浮动基座之间的关系。如“获取浮动基座下的雅格比矩阵”一节所述，末端坐标是会随着基座旋转而发生变化的。这样的变化有时可能会导致机身晃动过大的时候，为了跟踪末端坐标，控制器会让腿部运动超过其操作空间。使用Rb进行旋转变换以后，可以一定程度上把末端坐标由浮动基座转到固定基座表达，进而避免出现超过操作空间的情况，在两个模型之间的切换也更加灵活。

3. 同样是关于Rb，由于  $p_{\text{new}} = Rb * p$ ,  $v_{\text{new}} = dRb * p + Rb * v$ , 为了简化运算，直接采用了  $v_{\text{new}} = Rb * v$ ，以简化一部分运算。这种简化在MPC流派也有用到。

## TODO:

1. 目前简化计算做的还不够，需要裁剪并简化相关计算。
2. 基于此模型其实可以展开很多相关的研究，比如增加自由度、增加步长步高的控制等等。欢迎大家对双足相关课题进行持续性研究。

参考文献：

[ETH动力学讲义](#)

[🇬🇧 Cassie HZD双足控制方法说明](#)