python相关

python虚拟环境中的包不会影响到其他虚拟环境中的配置，虚拟环境中的包位于envs/env\_name/lib/site-packages下

Override:覆盖父类所继承的类函数

list的成员，可以是class，也可以是list，如pid类的实例组成了一个list

\_\_call\_\_()的功能类似于在类中重载 () 运算符，使得类实例对象可以像调用普通函数那样，以“对象名()”的形式使用

字典（dictionary）数据类型，不同于其他由数字索引的序列，字典是用”键”（key）来索引的。通常表示为dict(key: val, …)，有以下特征：

键可以是任何不可变（immutable）数据类型（不可变数据类型：数字，字符串、元组）（也就是说key不能为列表和字典类型）；

每个键必须是唯一的；

字典中每一项的顺序是任意的

defaultdict是Python内建dict类的一个子类，第一个参数为default\_factory属性提供初始值，默认为None。它覆盖一个方法并添加一个可写实例变量。它的其他功能与dict相同，但会为一个不存在的键提供默认值，从而避免KeyError异常。

defaultdict接受一个工厂函数作为参数，如下来构造：

dict =defaultdict( factory\_function)

这个factory\_function可以是list、set、str等等，作用是当key不存在时，返回的是工厂函数的默认值，比如list对应[]，str对应的是空字符串，set对应set( )，int对应0

类名定义后面的object可写可不写，python3默认都有

python函数多个返回值使用元组

在Python中，元组是以逗号分隔的项目序列；它是一系列不可变的Python对象。元组与列表相似，但元组一旦声明就无法更改（元组是不可变的）。元组通常比列表更快。

def fun():

str = "你好！"

x = 2019

return str, x;

str, x = fun() # Assign returned tuple

print(str)

print(x)

四元数

二元数：复数

三元数：乘法不收敛，无用

四元数：标量+3维向量

八元数

机器人基础

驱动器空间，关节空间，笛卡尔空间

0.0 关节空间

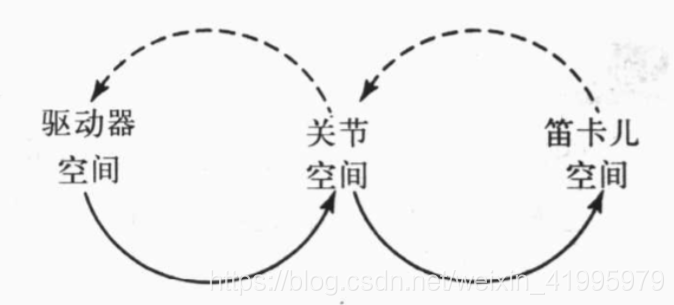
对于一个具有n个自由度的的操作臂来说，它的所有连杆位置可由一组n个关节变量来确定。这样的一组变量通常被称为n x 1的关节矢量。所有关节矢量组成的空间称为关节空间。

0.1 笛卡尔空间（操作空间operation space,，task-oriented space）

当位置是在空间相互正交的轴上的测量，且姿态是按照空间描述章节中任一一种规定测量的时候，我们称这个空间为笛卡尔空间，有时也称为任务空间或者操作空间。简单地理解成在空间直角坐标系。

0.2 驱动器空间

到目前为止我们一直假设每个运动关节都是由某种驱动器直接驱动的，然而对于许多机器人来说并非如此，比如利用两个驱动器以差分驱动方式来驱动一个关节，或者使用四连杆机构来驱动关节，或者有些关节存在耦合关系，如两个关节角度限制相等，或者关节直接锁死。这时就需要考虑驱动器的细节，由于测量操作臂的传感器常常安装在驱动器上，因此当我们在使用驱动器时就需要把关节矢量转换到驱动器矢量，驱动器矢量组成的空间就称为驱动空间。



驱动器空间到关节空间的映射：根据驱动器位置计算关节角；

关节空间到笛卡尔空间的映射：根据关节角计算机械臂末端执行器的笛卡尔位置和姿态.

驱动器空间下的力与关节空间下的力：关节空间力=驱动器空间力×减速比

力的维度问题：驱动器空间=驱动器motor个数，关节空间=关节joint个数

mujoco入门

机器人仿真器=物理引擎+渲染引擎

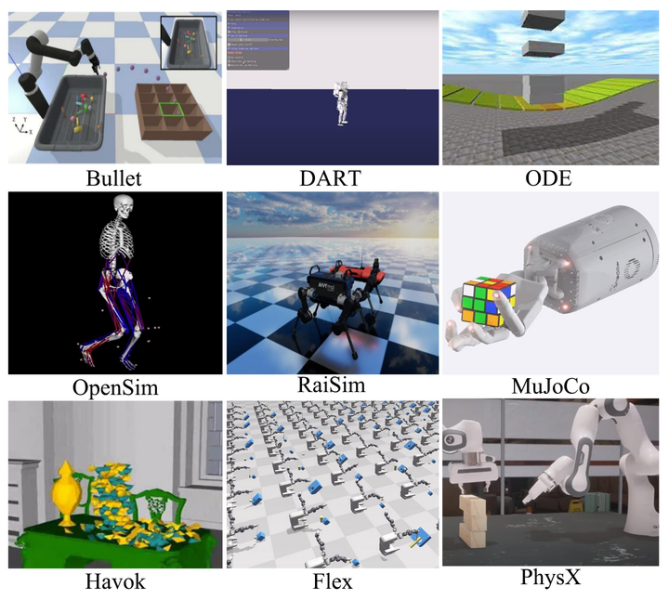
机器人仿真器 游戏引擎

主要钢体系统 柔性材料视觉效果，海量粒子运动模拟

追求精确性与速度 追求视觉效果

高效的空间向量法 简单的牛顿力学

不同的物理引擎



基于MuJoCo目前用的比较多的开发环境就是OpenAI开源的gym[3]和mujoco\_py[4], 其次是DeepMind开源的dm\_control[5], 主要都是一些连续控制和机器人相关的task与benchmark,但是两个环境不是直接兼容的. dm\_control可以算是将gym和[mujoco\_py](https://www.zhihu.com/search?q=mujoco_py&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"article","sourceId":151107285}" \t "/home/yzk/Documents\\x/_blank)进行了融合, 既包括RL的环境设置, 又包括了MuJoCo的python接口以及常用的task定义.

MuJoCo的核心特点

结合先进接触力学、广义坐标系统

游戏引擎(如ODE、Bullet、Physx)一般通过数值优化的方法来处理关节约束，会造成多刚体系统的不稳定和不精确。MuJoCo则采用广义坐标系和基于优化方法的接触力学方法。

柔性、凸模型并且解析可逆的接触力学

现代物理引擎大多数[求解线性互补问题](https://zhuanlan.zhihu.com/p/378361153)来处理约束，MuJoCo允许软体接触和其他约束，并包含一个独有的逆动力学模型来做数据分析。提出了新的摩擦力模型，支持滚动摩擦、扭转摩擦等多种摩擦力仿真。

肌腱仿真

MuJoCo支持3D的肌腱模型，这个模型跟OpenSim的肌肉模型相关，不过做了进一步优化，使MuJoCo支持人工肌肉仿真、线驱动灵巧手仿真等实用功能。

广义驱动器模型

可以配置高达3阶的驱动器模型，从而实现气缸、液压、肌肉等各种非线性驱动器仿真。

可灵活配置的仿真流程

可能灵活将仿真步骤拆开执行，或者只执行仿真流程的一部分(如不计算逆动力学)。

先进的模型、编译方法

采用IMG_256分离模型和运行时数据，有助于并行。模型可以编译成MJB二进制格式文件，提高加载速度。

强大的环境建模语言

独创了MJCF建模格式，相比URDF模型具有易读性、灵活配置等优点。

高效的软体材料仿真

支持绳、布料、可变3D物体的稳定可靠仿真。

Mujoco中关节的作用是增加自由度，如果没有关节，则默认为焊接在一起

关节空间与笛卡尔空间

关节坐标：

最适合复杂的运动结构，如机器人；

接头增加了默认情况下焊接在一起的实体之间的自由度；

关节约束在表示中是隐含的，不能违反；

模拟体的位置和方向由广义坐标通过正运动学获得，不能直接操纵（根体除外）。

笛卡尔坐标：

最适合于许多相互弹跳的物体，如分子动力学和盒子堆积；

关节移除了在默认情况下可以自由浮动的物体之间的自由度；

关节约束在数值上是强制的，可以被违反；

模拟实体的位置和方向是显式表示的，并且可以直接操纵，尽管这会导致进一步的关节约束冲突。

LCP问题

建模

分为低级模型mjModel和高级模型mjCModel，其中高级模型是为了用户设计的，只为了生成低级模型，高级模型mjCModel→MuJoCo compiler)→mjModel

高级模型存为xml文件MJCF/URDF (XML)，mj\_loadXML加载，格式的区分由开头的关键字

低级模型存为二进制文件MJB (binary)，mj\_loadModel加载，使用mj\_saveModel存

XML格式的urdf文件或者MJCF文件，那些都是格式

然后，软件可以在不同的媒体(文件或内存)和不同的描述级别(高或低)上创建同一个模型的多个实例。

运动链

MuJoCo一般用于约束刚体的仿真。系统状态定义在关节广义坐标系中，刚体被组织成运动链。除了worldbody外其他刚体都只有一个父节点，并且运动环结构的定义不被允许。如果想要定义闭环结构话，需要通过[公式]属性来定义环关节。

因此MuJoCo模型的骨架由一到多个运动链组成(自由浮动刚体也可以定义成一个运动链)，下面列出的元素依附于具体某个刚体(下一节中的元素则不依附任何刚体)。

Body: 刚体body有质量和惯量特性，但是不包含任何几何信息(几何信息包含在geoms中)。例如一个body叫desk，里面包含多个geom如长方体，但是这些长方体都没有名字，只有大小，位置，材料

每个刚体有两个坐标系:用于定义刚体和其他元素位姿关系的坐标系，和一个位于刚体质心并且对其主惯量轴的惯量坐标系(此坐标系中惯量矩阵是对角形式)。在每一步仿真中MuJoCo递推计算前向动力学，得到刚体在全局坐标系中的位姿。

Joint: 关节定义在刚体中，来创建刚体和其父节点刚体的自由度(当关节缺失时两个刚体固接)。有四种类型的关节:球关节ball,滑动关节slide,旋转关节hinge,自由关节free。其中球关节和自由关节通过单位四元数表示。

DOF: 自由度跟关节数相关，但不是一一对应，这是由于球关节和自由关节有多个自由度。自由度有速度相关的性质如摩擦损失、阻尼、电机惯量等。广义力在自由度空间内表达；关节具有位置相关的性质如范围限制、弹簧刚度等。自由度不直接由用户定义，而是通过编译器仿真指定关节时创建。

Geom: geom是固接到刚体上的3D形状。一个刚体可以有多个geoms。geom可以用于碰撞检测、渲染、自动推断刚体质量惯量。MuJoCo支持多种基元几何形状:plane, sphere, capsule, ellipsoid, cylinder, box。geom有大量的材质相关特性用于仿真和可视化。

Site: Site可以认为是零质量的geom。其定义了刚体坐标系中感兴趣的位置点，通常用于定义一些对象（如传感器、肌腱路径点、滑块终点）的位置。不参与碰撞检测。

Camera: MuJoCo的默认存在一个相机，用户可以在可视化界面中自由移动鼠标来切换视角。另外用户也可以自己定义一些相机视角用于立体渲染等。

Light: 光源可以固接在世界坐标系下或者移动的刚体上。MuJoCo实现的可视化界面可以访问OpenGL中所有的光源模型如环境光、漫反射、镜面反射、衰减截止、方向光、雾、阴影等等。默认的主光源跟相机一起移动，其特性可以通过mjVisual选项来改变。

刚体、几何形状和虚拟点(bodies, geoms, sites)

这三个元素大致对应了现实世界中的刚体。为什么要用三个元素而不是合成一个?

相同点:三者都有一个空间坐标系固接在上面，坐标系通过前向动力学计算坐标系的位姿，body的位姿在mjData的xpos、xquat、xmat,geoms的位姿在geom\_xpos 和 geom\_xmat，site的位姿在site\_xpos and site\_xmat。

不同点:body用于构建运动链和做为其他元素的容器(包括geom和site)。body具有惯量，但是没有几何外观，因为在物理仿真中(除了碰撞检测)不需要外观。我们在任何机器人动力学书籍中看到的抽象运动链均不含几何外观。geom用于可视化和碰撞检测，通过固接的方式连接到body上。geom也可以有质量和惯量，但是只是用于计算body的质量和惯量，实际上mjModel中geom并不包含惯量信息。site一般用于指定用户感兴趣的点，这些点没有质量、不会和物体实体碰撞，如用来指定触觉传感器的体积、IMU的安装位置等等。

Body用于慣量

Geom用于外观，碰撞

两个geoms和sites属于同一个body:

各种预设值，如assert，default，class，type

在

mujoco动力学建模分析

接触动力学分析

切向、扭转和滚动摩擦，以及椭圆形和锥形摩擦锥

x为接触正方向，yz为切平面

摩擦锥变为锥形变为椭圆形，用于软接触动力学

Group

margin, gap

Priority(默认0)：谁的高就用谁的condim类型

Condim(默认3)：接触力，力矩的维数，1，3，4，6维

1维：无摩擦接触，只产生法向接触力

3维：法向力以及切向摩擦力

4维：法向力、切向力、摩擦力矩，用于软手指建模

6维：两个geom之间所有运动，用于建模球、轮胎在地上的滚动

不能是2，5因为两个切向与法向被视为成对的

friction：摩擦系数，维数为condim-1

触点最多5个摩擦系数：2个切向，2个滚动，1个扭转

几何体只有3个系数：切向（2个相等），滚动（2个相等），扭转

a1 + d \* (b v + k r) = (1 - d) \* a0

a1为有约束加速度，v为有约束速度，r为位置或残差（在摩擦尺寸中定义为0），个人觉得是位置偏差

a0为无约束加速度，

b，k分别为虚拟弹簧的刚度、阻尼（缩放后，实际效果为b，k与d相乘）

d为约束阻抗，属于【0-1】，越大则约束越强，可为常数，或者为r的函数，形状由solimp设置

b，k，d通过solref与solimp间接设置

solref：若全为正数，则为(timeconst, dampratio)

刚度b = 2 / (dmax \* timeconst)

阻尼k = d(r) / (dmax \* dmax \* timeconst \* timeconst \* dampratio \* dampratio)

否则为（-stiffness，-damping）

刚度b = damping / dmax

阻尼k = stiffness / (dmax \* dmax)

timeconst至少比仿真步长大一倍

dampratio通常设为1为临界阻尼，小了就欠阻尼，大了就过阻尼

solimp:设置d的形状，(dmin, dmax, width, midpoint, power)

dmin, dmax限制d范围

width为曲线范围

midpoint为曲线拐点

power为幂需要大于1

solmix

属性的覆盖

o\_margin, o\_solref, o\_solimp attributes of option, applies only to contacts

三角摩擦锥、椭圆摩擦锥：接触力与约束违反距离之间的关系，简单的约束是胡克定律中的线性的，类似于近似化的弹簧，复杂的是非线性关系，变成了椭圆

金字塔摩擦锥和椭圆摩擦锥之间的选择是一种建模选择，而不是算法选择，即它导致使用相同算法解决不同的优化问题。椭圆锥更接近于物理现实。然而，金字塔锥可以提高算法的性能，但不一定。虽然默认设置为金字塔形，但我们建议尝试使用椭圆锥。当接触滑移是一个问题时，抑制接触滑移的最佳方法是使用椭圆锥、较大的不规则度和公差非常小的牛顿算法。如果这还不够，请启用Noslip解算器。

限制约束与接触约束

rbound: adius of bounding sphere

sameframe

For collision detection, a height field is treated as a union of triangular prisms. Collisions between height fields and other geoms (except for planes and other height fields which are not supported) are computed by first selecting the sub-grid of prisms that could collide with the geom based on its bounding box, and then using the general convex collider. The number of possible contacts between a height field and a geom is limited to 9; any contacts beyond that are discarded. To avoid penetration due to discarded contacts, the spatial features of the height field should be large compared to the geoms it collides with.

Detailed collision checking is performed by functions that depend on the geom types in the pair. MuJoCo supports several primitive geometric shapes: plane, sphere, capsule, cylinder, ellipsoid, box. It also supports triangulated meshes and height fields.

We have chosen to limit collision detection to convex geoms. All primitive types are convex. Height fields are not convex but internally they are treated as unions of triangular prisms (using custom collision pruning beyond the filters described above). Meshes specified by the user can be non-convex, and are rendered as such. For collision purposes however they are replaced with their convex hulls. Mesh collisions are based on the Minkowski Portal Refinement (MPR) algorithm as implemented in libccd. It has tolerance and maximum iteration parameters exposed as mjModel.opt.mpt\_tolerance and mjModel.opt.mpr\_iterations respectively. MPR operates on the convex hull implicitly, however pre-computing that hull can substantially improve performance for large meshes. The model compiler does that by default, using the qhull library.

In order to model a non-convex object other than a height field, the user must decompose it into a union of convex geoms (which can be primitive shapes or meshes) and attach them to the same body. Tools such as the HACD library can be used outside MuJoCo to automate this process. Finally, all built-in collision functions can be replaced with custom callbacks. This can be used to incorporate a general-purpose “triangle soup” collision detector for example. However we do not recommend such an approach. Pre-processing the geometry and representing it as a union of convex geoms takes some work, but it pays off at runtime and yield both faster and more stable simulation.

asset/ hfield (\*)

This element creates a height field asset, which can then be referenced from geoms with type “hfield”. A height field, also known as terrain map, is a 2D matrix of elevation data. The data can be specified in one of three ways

不要用这个来设计奇奇怪怪的桌子来执行接触任务，老老实实用sw画转为urdf，这个的效果是像素值的点的连接（存疑），因此加载不同的像素值png效果会不一样，哪怕他们实现的高度场是长相相同的，接触力也会不一样。

Plane and height field collisions are indeed different, even if the height field is perfectly flat. Height fields are always treated as a union of triangular prisms, and each prism is collided using convex collisions. So the number of configuration of contact points will generally be different, in turn resulting in somewhat different contact forces.

加载mesh 的注意：

sw画图单位为m，导出stl单位系统更改为m，坐标系同样也可以选择

mujoco只建立凸的，所以凹下去的样子想都不要想，除非慢慢分割逼近

凹面的唯一解决为凸面的和

Default关键字

使得可以在开头统一修改，能给后面的未赋值的属性进行赋值

类

可定义无数种类的名字，除了顶级类，未定义则为main，也可以定义为其他名字

在一个类的内部包含另一个类时，性质由子类将继承的父类部分或全部覆盖

顶级类没有父类，因此使用默认初始化

坐标

用户编写MJCF文件时，可以是局部坐标，也可以是全局坐标，由coordinate确定，但编译后都以局部坐标表示

全局坐标更直观，而且可以不指定身体的位置和方向

关于为啥找不到mujoco的python文件：

PyMjData and related classes are automatically generated from the MuJoCo C header files. For more information on this process, see [Autogenerated wrappers](https://openai.github.io/mujoco-py/build/html/internals.html" \l "genwrapper). Their structure therefore directly follows the MuJoCo structs.

使用mujoco的三个层级

低级：直接用cython文件

中级：pymjdata，用cython将c文件转换为cython文件

高级：mujoco\_py.MjSim，用mujoco\_env

MjSim: Basic simulation详细，仿真部分，加载模型

内部包含PyMjModel与PyMjData

PyMjModel由xml的文件建立，然后用于MjSim初始化，xml文件读取body的inertia属性存在一个选择位，选择读取文件还是自行生成

model里面包含一大堆函数，如何找到函数的详细解释？

Mjmodel里面的东西在mujoco.h里面都有

mjdata在mujoco-py的官方文档里

Mjmodel趋向于是model的各种信息

mjdata不止信息，还有各种方法api function

两个东西都是可修改并产生作用的

attribute 'qpos' of 'mujoco\_py.cymj.PyMjData' objects is not writable

设了速度，是不是要用那个controller

一定要用gym.mujoco\_env?

这里的mujoco\_env

是所有基于mujoco引擎的项目都共用的一个东西，我们找到源码，打开看一下

可以看到，大多是关于渲染的内容，中间有一行：methods to override

就是说，别的我都写好了，你别动，只要你动这俩函数就行了。

那么ok,回到自己的pusher里来：

除了要我们override的东西之外，还有什么呢？

有一个特别重要的东西，就是step函数

step函数就对应一些人说的，物理模型。前面都在提mujoco，提渲染，那么这里提供的就是这个环境与外界交互的规则。

当输入了action a，我应该回应一些什么。这里就看有什么具体需求了，代码长，就不截上来了。

int nq;// number of generalized coordinates = dim(qpos)关节位置个数

int nv;// number of degrees of freedom = dim(qvel)关节速度个数

int nu;// number of actuators/controls = dim(ctrl)驱动器个数

int na;// number of activation states = dim(act)

mujoco\_py.PyMjData简略，由c转换而来，内部包含一堆数据，以及数据的提取函数（get），主要问题是。。。这些数据的含义是什么

MjSimPool: Batched simulationclass简略，被移除了

mujoco\_py.MjViewerBasic(sim)详细

PyMjModel：关于PyMjModel没有很好的文档说明，只能从[源码（点这里）](https://link.zhihu.com/?target=https://github.com/openai/mujoco-py/blob/master/mujoco_py/generated/wrappers.pxi)中查看相关的接口。

只包含mjmodel和sim，导致其他的结构体在python中莫得

如何在python中导入mjmodel.h的其他东西

If qpos or qvel have been modified directly, the user is required to call forward() before step() if their udd\_callback requires access to MuJoCo state set during the forward dynamics.

基于mujoco-py的机器人仿真优质代码库推荐：

gym-fetch：机械臂系列：slide，push，pick and place。

robosuite：如果做机械臂相关的必看！斯坦福开源的Surreal机器人项目，基于Mujoco，包含单/双臂与多种手爪组合配置，task包括pick&place、装配协作等。

flow-rl：如果做击球相关的，可以参考一下

UR5+GRIPPER

10个关节：1个底座，2个大臂，3个手腕 3根手指，1个剪刀

Mujoco的能力不清楚，能不能轨迹规划，估计不能

KDL

Mocap：移动末端执行器都是靠这个实现的？代码里面把他和末端执行器焊在一起了

直接设置速度？

前向运动学就是设置关节速度、位置，然后读出来末端执行器位置？

Ctrl即力矩

感觉就是设计好所有东西，然后step跑一步

mocap似乎会锁住末端执行器

mocap（motion capture system）在mojoco中视为约束，虽然能在运动中改变

两大控制方式：mocap，或者ctrl，一个是设计末端执行器位置，一个是控制关节力矩

动力学问题

接触任务中的接触力，无法从运动学中恢复，除非采用软接触模型，如可形变墙体，弹簧阻抗模型等，但是这些没有考虑施加的力，因此会引入误差

现代模型往往在计算接触力时，考虑施加的力（内力），但是复杂

mujoco在连续时间中计算前向动力学与逆向运动学

雅克比矩阵建立关节坐标系与约束坐标系之间的关系

雅可比矩阵：关节空间速度转到约束空间速度

雅可比矩阵的逆：约束力转到关节空间力

M*v*˙+c=τ+*J**T*f

τ：被动力，流体力学，关节施加的力，用户指定的力

c：科氏力，离心力，重力

M：惯性矩阵，复合刚体 (CRB) 算法the Composite Rigid-Body (CRB) algorithm，通常是稀疏矩阵，经常分解来保持器稀疏性

他所谓的约束空间估计就是笛卡尔空间

控制器模型：单输入单输出

输入u：用户指定的标量mjData.ctrl

输出p：标量关节力矩 mjData.actuator\_force

实际torque：gear \* mjData.actuator\_force

状态： mjData.act

组成：转换，激活动力学，力生成

transmission转换：标量驱动器力到关节力

activation dynamics激活动力学：驱动器如气动、液压等，需要激活

force generation力生成a,b：

力=function（控制信号，激活动力学，转换），实现直接力控制、位置伺服、速度伺服

p = f(au, activation dynamics, force generation)

mjModel.actuator\_gainprm (nu x mjNGAIN)，mjModel.actuator\_biasprm (nu x mjNBIAS)

mjData.qfrc\_actuation: the net force in generalized coordinates contributed by all actuators

mjData.qfrc\_applied: user-defined forces in joint coordinates

mjData.xfrc\_applied: user-defined forces in Cartesian coordinates

mjData.qfrc\_bias:bias force: Coriolis, centrifugal, gravitational

mjData.actuator\_force: ctrl

mjData.qfrc\_actuator: torque

被动力：只取决于位置速度

mjData.qfrc\_passive

在关节和肌腱中的弹簧阻尼器和流体动力学

关节力矩输出τ = actuator\_gear × env.sim.data. actuator\_force

其中env.sim.data. actuator\_force

= env.mjc\_model.actuator\_gainprm × env.sim.data.ctrl

+ env.mjc\_model.actuator\_biasprm[:][0]

+ env.mjc\_model.actuator\_biasprm[:][1] \* env.sim.data.actuator\_length

+ env.mjc\_model.actuator\_biasprm[:][2] \* env.sim.data.actuator\_velocity

动力学τ= mjData.qfrc\_passive

+ env.sim.data.actuator\_force

+ mjData.qfrc\_applied / mjData.xfrc\_applied

xml文件里面指明了驱动器的输出范围以及减速比

为啥改质量和惯量没有影响，改力矩有影响

现在无法从xml文件中正确加载inertia与mass属性

g=9.81

cpp文件与py的api对应关系:py文件由cpp文件wrapper而来，所以应当都是有的

x：笛卡尔空间？

model和data里面的数据有什么区别，一个set一个read？

pos

body\_pos body\_xpos get\_body\_xpos

jnt\_pos qpos get\_joint\_qpos, set\_joint\_qpos

geom\_pos geom\_xpos get\_geom\_xpos

site\_pos site\_xpos get\_site\_xpos

cam\_pos cam\_xpos get\_camera\_xpos

body\_pos与xpos

body与body frame

mj\_kinematics

PyKDL入门

PyKDL：KDL的python版本，可以在python中开发机器人的运动学和动力学

三维向量：vector()

旋转矩阵：Rotation()

坐标系：Frame

一级结构：segment，由关节和连杆组成的最小单元

四个成员变量：

（1）name

（2）Joint

name, type

scale, inertia, damping, stiffness

axis,origin

joint\_pose

get, pose, twist

（3）RigidBodyInertia

质量m，double类

质心oc，vector类， reference坐标系下reference到cog位置矢量

转动惯量Tc，rotationalinertia类，cog下转动惯量

（4）Frame

DH坐标系

a Rotation M for the rotation of the object/frame wrt the reference frame

a Vector p for the position of the origin of the object/frame in the reference frame

四个坐标系：

（1）reference坐标系（输入坐标系，上一个segment的输出坐标系）

（2）joint坐标系（通常与reference坐标系重合）

（3）cog坐标系（reference坐标系原点平移到质心）

（4）tip坐标系（输出坐标系，下一个segment的输入坐标系）

四个成员函数：

对 name，Joint，RigidBodyInertia进行set或get

三个方法：

(1)getFrameToTip()获得初始状态时，root坐标系（关节初始坐标系）到tip坐标系位姿

(2)pose()关节旋转后，root坐标系（关节初始坐标系）到tip坐标系位姿

(3)twist()关节旋转后，root坐标系（关节初始坐标系）到tip坐标系速度

初始化

joint()

二级结构：chain，由segment组成，也是后续求解运动学和动力学的单元

创建

构造函数法

堆叠法（加segment，加chain）

成员变量

关节个数nrOfJoint

单元个数nrOfSegment

单元体segment

成员函数，对成员变量进行get

三级结构：tree，由chain组成的最高单元

创建

构造函数法

堆叠法（加segment，加chain，加tree）

成员变量

关节个数nrOfJoint

单元个数nrOfSegment

单元体segment

成员函数，对成员变量进行get

关于各种坐标之间的关系

坐标是相对关系还是绝对关系

mujoco里面是相对关系，需要不断累积才能知道世界坐标系中的位置

mujoco没有逆运动学，利用kdl，kdl安装要把pybind11 gitclone下来，放到pyxxxx的文件夹下

要是在pycharm下运行，还要复制PyKDL和build下的.so文件到env——name/lib/python3.6/site-packages

pykdl\_utils比pykdl人性化好多，baselink，endeffctor都有，还有joint limit

运动学、动力学命名方式

ChainXXSolver

FK, IK

FD, ID

通用性说明

机器人本体

实物机器人与UR5的结构完全相同，关节设置相同，连杆、关节属性、质量、惯量参数关系未知，末端连杆的长度可能有一定出入

末端执行器

实物机械爪robotiq 2f-85, 插销

仿真机械爪robotiq 3finger, robotiq 2f-85

关于古月局的代码分析

控制器初始化：

model,sim,viewer

create\_list

create\_group,

状态位:reached\_target, image\_counter

actuated\_joint\_ids

current\_target\_joint\_values, current\_output

controller\_list, groups, actuators

create\_list

self.controller\_list：关节pid控制器list

self.current\_target\_joint\_values：关节当前目标值

self.current\_output：关节pid当前输出

self.actuators：关节控制器信息。驱动器id、名称，关节id、名称，关节控制器配置

create\_group

self.groups: defaultdict类型

group[all],group[arm]

move\_group\_to\_joint\_target（关节空间控制）

若未到达

逐个利用pid设置控制器力输出

sim.step()

move\_group\_along\_trajectory

并不是任意轨迹输入

也只是单纯的控制每个关节的转动角度，辣鸡

主要控制环

读取当前位置self.sim.data.qpos

pid计算控制器输出current\_output

加到控制器上sim.data.ctrl

不是所有的控制器属性都在init里面写了出来

关于轨迹规划部分

古月居如何实现任意轨迹输入

如何将预期速度、位置转换为控制力矩

学长代码分析

模型加载部分

避开了pykdl\_utils，urdf包的加载方式，就无法一步到位

需要利用堆叠法得到机器人tree

但是为什么代码中堆叠到chain层就结束了

好像没必要到tree层，加载生成的可能是tree，也可能是chain？

kdl中的运动学都进行了重新封装

机器人

存在许多关节未使用，如躯干与机械臂之间的抬升关节，机械臂内部许多旋转关节

robot0:slide0：桌子前后

robot0:slide1：桌子左右

robot0:slide2：桌子上下

shoulder\_lift\_joint：肩上下抬手关节1

elbow\_flex\_joint：肘上下抬手关节2

wrist\_flex\_joint：腕上下抬手关节2

黑盒

object0:joint：物体的位置，但为啥是7位

getchain

getchain里面的joint，frame也都是7个，xml文件中也只给了这7个关节驱动

对应关节

shoulder\_pan\_joint

shoulder\_lift\_joint

upperarm\_roll\_link

elbow\_flex\_link

forearm\_roll\_link

wrist\_flex\_link

wrist\_roll\_link

getchain中的数据来源为xml，相当于没有直接使用urdf与xml，而是在xml文件里面翻找数据，主要是数据还对不上

tree和chain的堆叠法

joint初始化，frame初始化

segment初始化，chain初始化，tree初始化：Segment(jnts, frms)， Chain(), Tree()

segment=joint+frame append

chain=segment1+segment2+…… addSegment

tree=chain1+chain2+…… addChain

main函数效果

跑一次正向运动学，输出末端执行器位置，对比kdl与mujico

跑一次逆向运动学，输出关节角度，对比实际角度与迭代结果

正向动力学的kdl输出4x3矩阵，上三行为旋转，最后一行为平移，与mj相当接近

逆向动力学输出从底盘开始，到末端手腕旋转的关节角度

对于扩展至ur5中出现的差异的解释：

原robot里面对body的描述只有一个pos，而在ur5中还包含四元数，旋转了45°

关于初始角度的设计。。。

这么水平的状态为什么会有四元数

学长的机器人全为0的状态下，也是趴着的

我怀疑是所有的坐标系只存在平移关系，都没有旋转，因此有的关节绕y转，有的绕x转，并不是将z轴统一设置为旋转轴

难道robot.xml里面也是只存在平移关系？

mujoco的body和joint关系，joint到底是连接在body的哪个地方

body不具有几何属性，就当做是一根线，然后joint就是

joint是加在body上面的，因此应当是高一级的body的属性，而不是下一级

总之就是joint是为了给上一级body提供自由度

关于body2的pos，是body1和body2结合的地方，body2相对于body1的位置，也是body2的joint的地方

编写：body1, body2 joint2

效果：body1, joint2, body2

segment的效果，指明joint和tip frame

frame的输入参数含义

frame是tip相对于reference/joint的位姿

三个quat是需要的，不加连z都对不上

四元数顺序对不上，kdl里面实数放在最后面

就没有ee的文件

变成fixed就没返回值了

那个四元数意味着什么旋转方式

关节一个一个加，看从哪个关节开始出问题

关节的转动方向问题？

关节0,1都是好使的

到了关节2就不对了

动的关节不一样，一个是1，一个是2

改变23号关节角度

输入joint参数与具体输出的是哪个栏杆的最小关节

num pos

1 shoulder\_link 怎么都不变

2 upper\_arm\_link qpos[0]影响m，k

3 forearm\_link qpos[0，1]影响m，2影响k

4 wrist\_1\_link

5 wrist\_2\_link

6 wrist\_3\_link

7 ee\_link

中间的坐标系存在问题

起始不是base的坐标，应当是001转轴下一个坐标

正逆运动学已解决

但是path，trajectory好像只有c没有python

速度

显示

segment的说明

mujoco的速度控制

为啥sim.data里面的数据都不变

路径规划，还涉及速度以及雅可比矩阵

涉及加速度吗？

不同的速度的确会影响姿态，但为何都是只动一下

力矩分析？关节力矩？

去掉爪子和不去掉的力矩值打印出来看一下

在同一个位姿的时候

看看一样不一样

再看看里面写的爪子的质量是多少，能不能自己算一个负载值直接加到末端，不用再增加一个link

约束动力学

约束坐标系？

雅可比矩阵如何搭建起两个坐标之间的关系

改变质量并不能改变自然下落的加速度

这个十列是什么意思

step1 查出来关节的配置

step2 理解关节配置

step3 设置关节配置

回调的使用

把palm加到tool0上

kdl对tool0的处理

n=7

不加连杆0，则有7个连杆

但是只有6个关节，还有一个fix

kdl里面的iner没有旋转效果

要是加关节，会导致正逆运动学求解无法锁死关节

改末端连杆的质心位姿没用，改质量有用

关节摩擦？

实际输出与关节输入

为啥都是0了还会有输出

[0 -65.5508 -20.646 -1.62932 -0.147937 0.000271859]

[0 -6.55508158e+01 -2.06460117e+01 -1.62932179e+00 -3.49604198e-02 2.71856834e-04]

iner.append(kdl.RigidBodyInertia(0.634, kdl.Vector(0, 0, 0.0968) - kdl.Vector(0, 0, 0.263),

kdl.RotationalInertia(0.0052, 0.0052, 0.0041)))

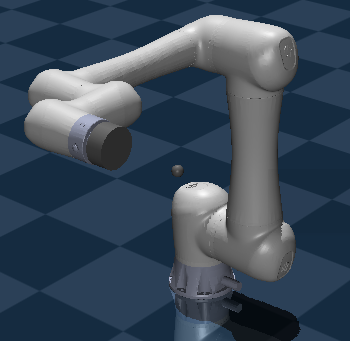
m= 0.634

cog = 0, 0, 0.0968

iner(xx,yy,zz) = 0.0052, 0.0052, 0.0041

kdl: 0 0.2145901 0.2145901 0.2150928 0 0

mjc: 0 2.1450197e-01 2.1450197e-01 2.150923e-01 0 0



ur5gripper

[-5.14498109e-15 -7.41232876e-01 -2.30539137e-01 -1.29872294e-03 1.05034922e-16 -1.29872294e-03]

[0, -0.74123, 0.00129, 0, -0.0012987]

ur5palm

[0.00000000e+00 -6.89120838e-01 -2.04851888e-01 3.51753830e-17 0.00000000e+00 1.75876915e-17]

[0, -0.68912, -0.20485, 0, 0, 0]

ur5

[0.00000000e+00 -5.85928866e-01 -1.55323529e-01 0.00000000e+00 0.00000000e+00 2.19846143e-18]

[0, -0.58592, -0.15532, 0, 0, 0]

ur5mini

[3.24747946e-48 -5.85929837e-01 -1.55324501e-01 -9.71287129e-07 -1.14754636e-51 -9.71287129e-07]

mjc

正常

[ 0.00000000e+00 -6.89041460e-01 -2.04813789e-01 -3.51753830e-17 0.00000000e+00 1.75876915e-17]

改长度

[ 0.00000000e+00 -6.89041460e-01 -2.04813789e-01 -3.51753830e-17 0.00000000e+00 1.75876915e-17]

[ 0.00000000e+00 -6.89041460e-01 -2.04813789e-01 -3.51753830e-17 0.00000000e+00 1.75876915e-17]

[0.00000000e+00 -6.89120838e-01 -2.04851888e-01 3.51753830e-17 0.00000000e+00 1.75876915e-17]

改质量

[ 0. -1.61776921 -0.65056902 0. 0. 0. ]

kdl

[ 1.32844e-31 9.86076e-32 -107.651 -23.8366-3.07489e-16 2.55977e-16 -6.6751e-33 0]

[0 -107.651 -23.8366 0 0 0]

预期

mjc

[0 x x x 0 0]

kdl

[0 0 x x x 0 0 0]

圆盘 1/2mr^2 3/2mr^2

棍 1/12mr^2 1/3mr^2

xx yy zz m r

爪子 0.00418 0.00508 0.00125 0.921 0.06 顶点坐标系

sw 0.00024 0.00024 0.00024 0.45271 0.069871 重心坐标系

估计 0.00800113 0.00972386 0.00176438 1.3 0.069871

0.00418\*(1.3/0.921)\*(0.069871/0.06)^2

0.00508\*(1.3/0.921)\*(0.069871/0.06)^2

0.00125\*(1.3/0.921)

mujoco中设计fixed很容易

kdl里面会出bug

urdf很容易

ur5gripper,ur5palm,ur5三个共用一套ur5\_gripper\_env

ur5mini是对ur5palm的模仿，kdl加入惯量参数，加入底座与末端连杆关节，使用ur5\_mini\_env

ur5nftgripper在ur5上加入了nftgripper，kdl加入惯量参数，加入底座与末端连杆关节，使用ur5\_nftgripper\_env

关于建模的说明

kdl

segment指定当前连杆近端joint与远端坐标系，远端坐标系用于描述link位置

mujoco

指定link近端位置

因此均无法有效表达工具坐标系

mujoco与kdl重复的部分

mujoco没有前向，逆向运动学、动力学，但是可以求出力来

同步问题的设置

如何让kdl中模型与mujoco中模型一起动

kdl有模型吗，能动态吗，还是只能按照当前状态进行计算

应当只是一个库

因此利用mujoco仿真得到的数据，作为kdl的输入，然后再把结果施加于mujoco

考虑把kdl包起来

类的方法总是将mjc数据转换为kdl再计算，转换为mjc再输出

把kdl属性private

为啥每个link设置了两个geom

自动生成的

弹起来还是因为刚度

要是两个都不加，就是默认的硬环境，所以会弹起来

要是加了一个，则加的那个因为比较小被采用

要是两个都加了,不该弹起来的，为啥弹起来了

不恰当的摩擦设计会导致弹起，要是不改摩擦，效果很好，可以兼容两个都加

影响来源包括：friction, solimp, solref

参数对系统的影响，振荡太久了

hfield是二維的，geom是三維，導致接觸模型不一樣？

参数设计的方式不一样？，连接难道是tendon？

不同的图的效果都不一样

想办法把mesh去掉

图会对接触力产生影响，哪怕生成的桌面渲染效果一样，实际仍然存在差异

如何把mesh去掉

阻抗的动态调节

阻抗的设置

算法转代码

kdl只使用他的运动学，去他的动力学，动力学用mujoco

改qpos不好使，引起其他关节变化

不同actuator效果区别