移动六足机器人与自动姿态调整的实现

李崇珊 盛李杰

摘要：本项目完成了移动六足机器人的关节数学建模，平移和旋转步态的步态曲线规划，机身姿态调整的运动学建模，并使用Simulink进行了几种移动步态的仿真。同时基于数学建模完成了实机的机电设计，在实机上成功实现了几种步态，并且通过陀螺仪的反馈，实现了自动姿态调整的功能。

关键词：六足机器人；移动步态；姿态调整；自稳；运动学

中图分类号：TG156

**A Mobile Hexapod Robot with Basic Movement Functions and Auto Posture Stabilization**

LI Chongshan SHENG Lijie

**Abstract：**This project has completed the joint mathematical modeling of the mobile hexapod robot, the gait curve planning of the translation and rotation, the mathematical modeling of the body posture adjustment, and has used Simulink to simulate the gaits above. This project also covered an electromechanical design of a real hexapod robot, on which we successfully verified the gait of basic movement functions (translation and rotation). In addition, with a gyroscope and its feedback, we accomplished auto posture stabilization function.

Key words**：Hexapod robot; Gait planning; Posture adjustment; Self stabilization; Kinematics**

# 研究背景

随着科学技术的不断发展和进步，为了获得所需的矿物能源和信息材料，人类不断探索外行星和深海。这些地区对人类来说很困难或无法进入，需要使用外力来取代人类的勘探活动。因此，机器人应运而生。通过对自然界生物的研究和分析，研究人员开发了一种能够在崎岖复杂地形中行走的多足机器人。与传统的轮式和履带式机器人相比，行走步态可以轻松跨越障碍物，可以调整姿势以保持身体的稳定性，可以根据地形选择行走方向，优化行走路线，降低能耗，提高作业效率。作为多足机器人的典型结构之一，六足机器人具有高承载能力和高稳定性。它以不连续支撑模式行走。当遇到崎岖复杂的地形时，它可以依靠自身的冗余自由度来调整腿部关节角度和身体姿势。其运动姿态灵活多变。此外，六足机器人可以同时配备摄像头、雷达等检测设备，机器人在矿山和废墟中行走时可以获取更多信息，还可以携带机械臂获取所需的矿物能量。这种多足机器人还可用于自然灾害、爆炸、火灾和化学品泄漏等救援任务。在这种情况下，它将危及受影响人员和救援人员的生命，或者救援人员很难在这种环境中到达。因此，有必要开发一种能够在如此复杂的地形中执行救援和探测任务的机器人。该六足机器人在行驶时稳定性高，可全方位移动，越障能力突出。它可以解决灾后地形复杂多变的问题，可以快速应用于救援任务。六足机器人具有更灵活的步态和较强的地形适应性。与两足机器人和四足机器人相比，它有着更广泛的应用场景，并且可以依赖更多的自由度来维持机器人故障后的运行。

20 世纪 80 年代，麻省理工学院的布鲁克斯研制出机器人 Genghis。该六足机器人基于视觉技术，通过装载 150 多个传感器，可在复杂的陆地上行走，可以很好地执行在火星等危险环境的探测任务。

不久之后，美国 NASA 为了支持探月作业研制出月球车 ATHLETE。如这款由美国喷气推进实验室（JPL）开发的六足机器人有 6 条关节型腿，腿部相应装有轮子，目的在于使得 ATHLETE 采集信息之后可以顺利地从泥坑里面爬出。同时针对月球表面的特征，ATHLETE 不仅可以负载 15t 的货物，而且可以在岩石以及斜坡上稳定地行走。

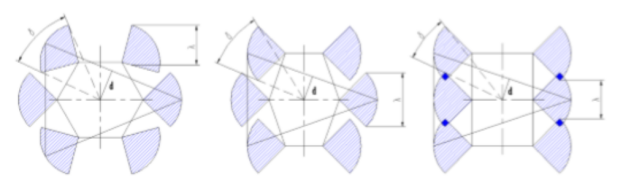
二十世纪六十年代以来，随着计算机和传感器技术的发展，以前研发的六足机器人难以适应的场景可以利用针对性的改进技术来解决。比较典型的有用于排雷的 SILO-6 机器人。该机器人尺寸为 0.88m×0.45m×0.26m，重量为 5kg。它分为五个主要子系统：传感器头，机械手，定位器，步行机器人本身和控制器。传感器头由一个探雷装置组成，可以检测带有金属零件的地雷。机械手携带地雷探测器。定位器子系统由电磁罗盘、GPS 天线和 WiFi 线组成。该机器人项目由西班牙工业自动化研究所于 2004 年完成，其主要目的是利用SILO6 探测地雷，被认为是解决几十年前部署的排雷所造成的全球性问题的一项举措。

在 2000-2010 年期间，日本千叶大学历经十余年的时间成功研发出一系列六足机器人——COMET。该六足机器人主要用于特殊灾害救援、地震探测等极端工况。该款机器人的最大的外轮廓尺寸约为 2.8m×3.3m×2.5m，机身质量约为 2120kg，其最大负载能力高达 400kg。COMET的腿部构型呈对称式分布，每一条腿均含有四个液压驱动关节，COMET 机器人足端采用了较大支撑面积的平板足构型，使其能够在松软地形移动并且具有攀爬斜坡的功能。

# 数学建模

## 尺寸和布局

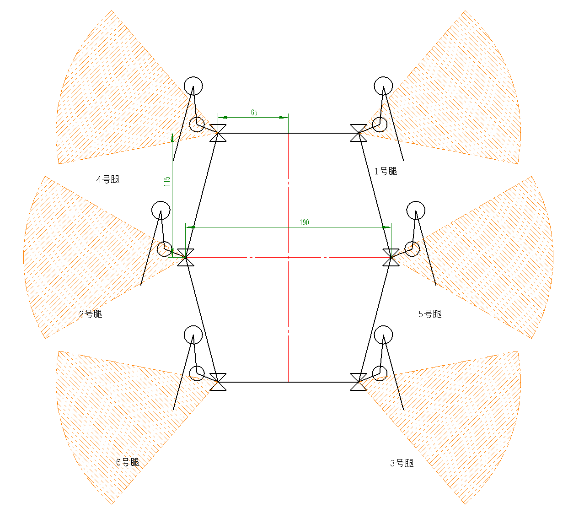
常见的六足机器人六条腿的布局有圆形，椭圆形和矩形。其布局会影响机器人的稳定性和腿部工作空间大小。



根据前人研究[1]：

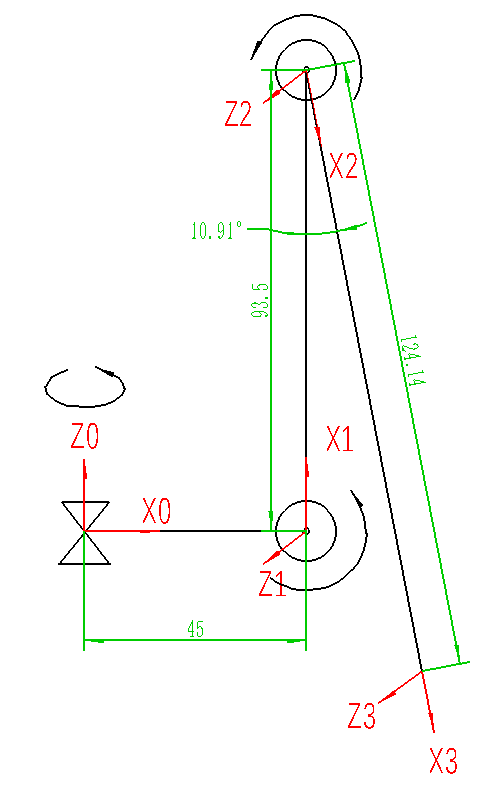
*（1）腿部在行进时的干涉情况：矩形布局的腿部摆动范围重合度最高，圆形布局的重合度最小，即在保证相邻腿部不干涉的情况下，圆形布局腿部摆动空间最大，椭圆布局次之，矩形布局最差；*

*（2）直线行走的有效行进距离：设定单腿迈步可以完成的最大迈步距离为λ，标出每种布局的迈步距离，可见圆型布局的迈步距离最小，椭圆较大，矩形最大。*

综合防止工作空间干涉和获得更大的前进距离的目的，我们选用了椭圆形的布局，底盘左右关节距离（宽）190mm，前后总长为230mm。

## 关节建模

### DH参数与尺寸



DH参数表

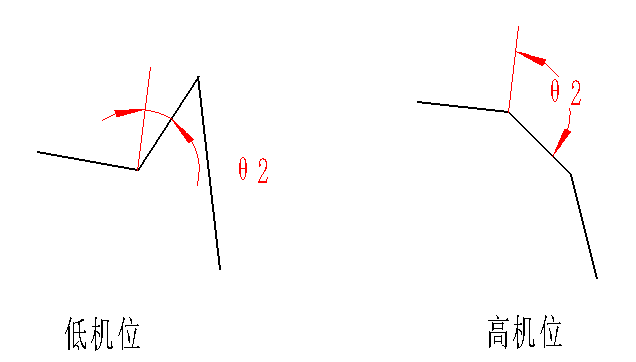
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | a | α | d | θ |
| 1 | 45 | 90° |  | 90°+θ1 |
| 2 | 93.5 |  |  | -169.09°+θ2 |
| 3 | 124.76 |  |  | θ3 |

人为规定关节角的限制：θ1∈[-30°, 30°]; θ2∈[-180°, 0]; θ3∈[79.39°, 169.09°]

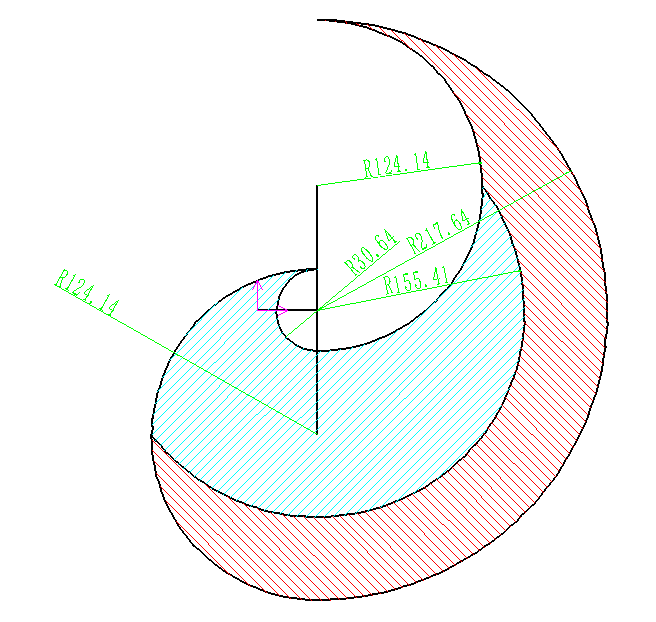
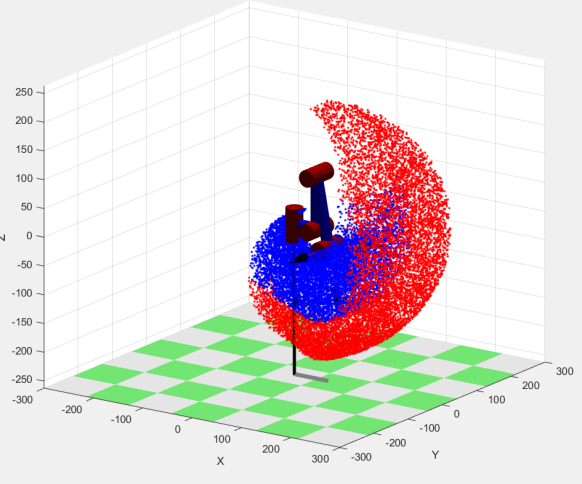
其中根据θ2的取值范围继续细分为两种状态，这两种状态的逆运动学求解公式会不一样，把这腿的两种状态分为高机位和低机位。

低机位：θ2∈[-90°, 0]；

高机位：θ2∈[-180°, -90°]

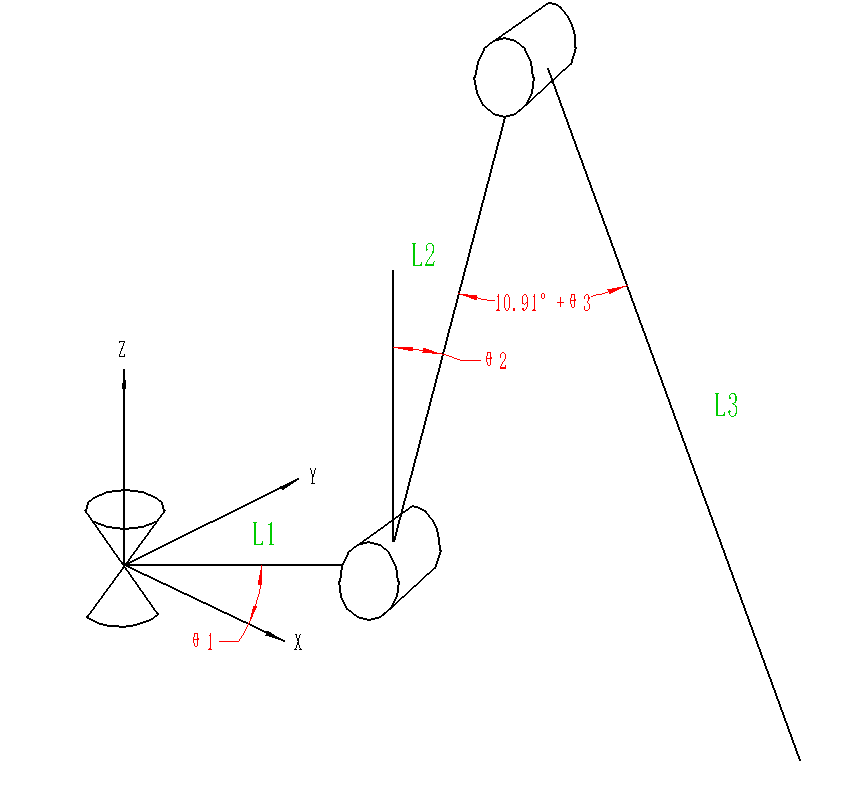


工作空间如图：（蓝色为低机位工作空间，红色为高机位工作空间）

### 正运动学

以第一旋转关节处为原点，建坐标系如下：



其中L1，L2，L3为杆长，θ1，θ2，θ3为关节角；第三关节角有10.91°的初始偏置，与实物的机械设计有关，详见后文机械设计部分。

正运动学公式：

### 逆运动学

已知末端坐标为 [x, y, z] ，令：

低机位：

高机位：

## 行走步态规划

### 步态曲线

根据前人研究的相关定义[2]：

步态：是指机器人的每条腿按一定的顺序和轨迹的运动过程，正是因为这一运动过程实现了机器人的步行运动。

步态周期T：步态周期是指多足机器人完成一个步态所需的时间，也就是所有腿轮番完成一次“提起-摆动-放下”的动作所花费的时间，在此过程中机器人机体也完成过渡过程。本系统的步态周期参数可调，经过实际测试，最小步态周期是300ms，这个参数受限于舵机的性能。

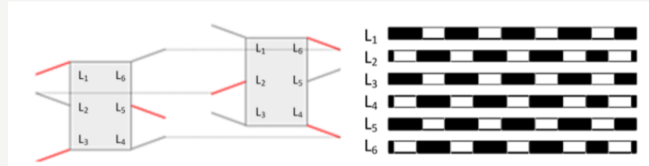
占地系数β：占地系数是指每条腿接触地面的时间和整个步态周期的比值。当占地系数等于0.5时，机器人是用两组腿交替摆动，这种步态称为小跑步态；当占地系数小于0.5时，机器人任何瞬间只有不足三条腿支撑地面，称为跳跃步态，当占地系数大于0.5时，机器人轮番有三条腿以上支撑地面，这种步态俗称慢爬行步态。

步幅L：机器人的重心在一个步态周期中的平移为步幅。

六足机器人常见步态：

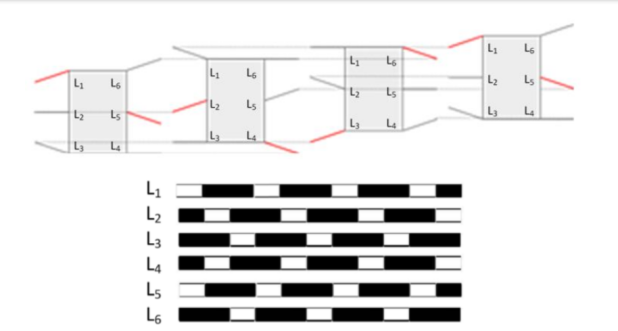
（1）二步态

　　二步态也叫三足步态、三角步态。支撑状态为 L1、L3、L5 和 L2、L4、L6腿 交替支撑。行走示意图和步态相图如图所示，示意图中红色连杆表示支撑腿，步态相图中黑色实心部分表示支撑相。



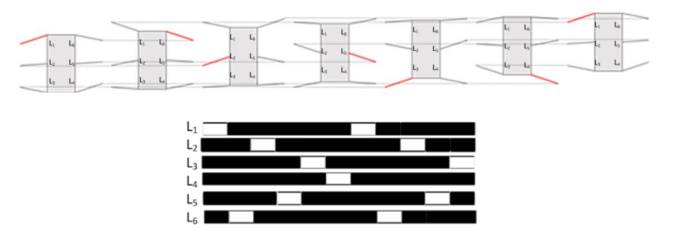
（2）三步态

　　三步态情况下始终有四条腿作为支撑相。通常将 L1、 L5 分为一组，L2、 L4 分为一组，L3、 L6 分为一组，也可按照其它方式两两分组。移动时每组腿依次抬起摆动。因支撑平面为四边形，因而又称之为四足步态或四角步态。其行走示意图如下：



（3）六步态

　　六步态情况下每条腿各自为一个分组，依次抬起进行摆动。其抬起顺序变化相比于三步态更多，可以为 L1 -L6 -L2 -L5 -L3 -L4 也可以是 L1 -L4 -L2 -L6 -L3 -L5不同的迈步方式使得机器人在整个运动周期内的支撑平面以不同的方式变化，以便适应不同的越障或爬坡需求。其行走示意图如下：

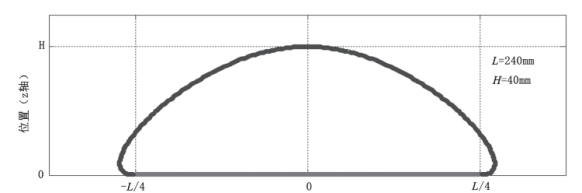
****

我们此处采用和验证的是二步态，即三角步态。

我们参考了前人对步态曲线的研究[3]：

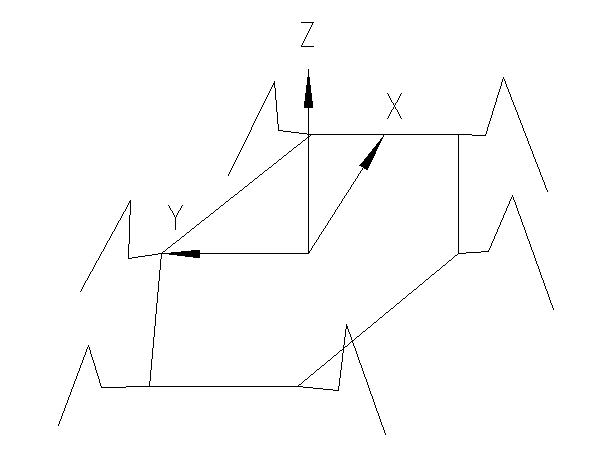
令H为最高抬腿高度，L为步幅长度。对于占空比β=0.5的情况，在机器人移动的一个运动周期内，选取运动轨迹上的7个点的 位置作为约束条件，将其中的速度连续性作为另外的3个约束条件，最后得到在以腿末端落点为原点的坐标系中机器人腿部足端轨迹规划函数：

机器人整个腿部足端在单周期内的运动封闭位移曲线:



### 整机步态坐标

机身坐标系以底盘的几何中心为原点，前进方向为x轴正方向，竖直上方为z轴正方向。

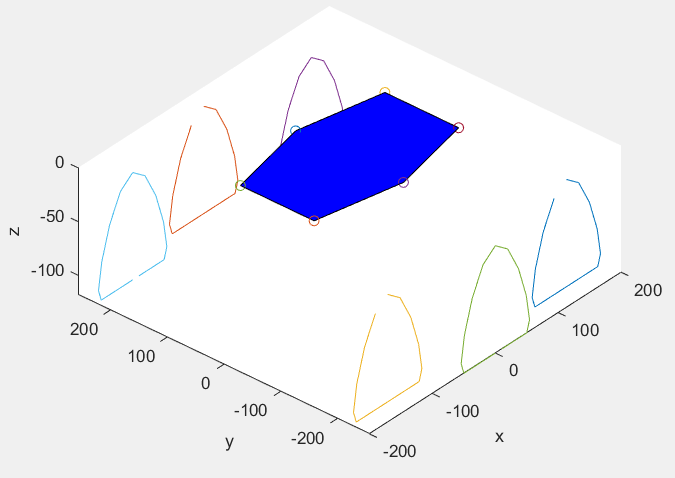


以占空比β=0.5的三角前进步态为例：

占空比β=0.5，1，2，3号腿和4，5，6号腿的步态刚好相差半个周期。

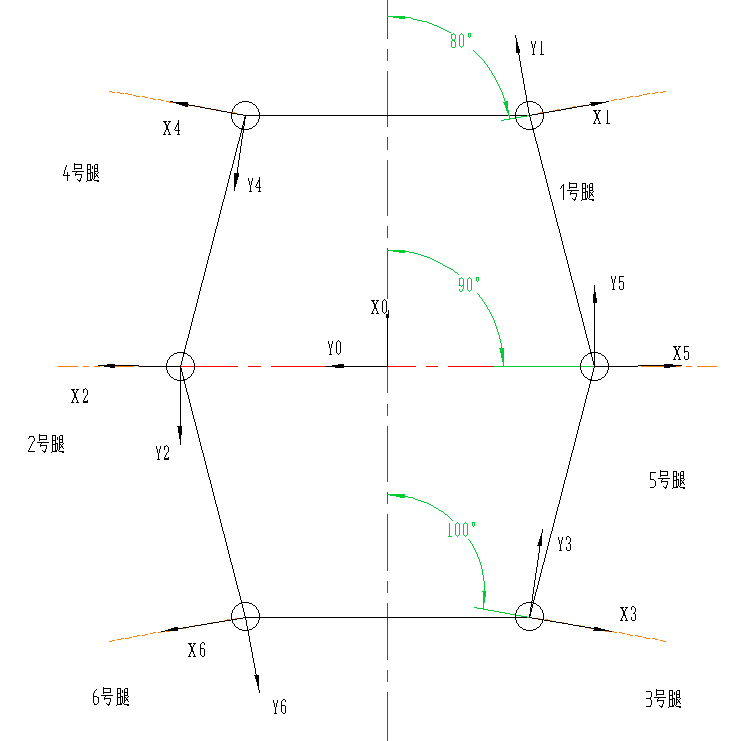
将生成的单脚落点曲线按照六只脚的位置加上xyz坐标的偏置，即得到机身坐标系下六条腿前进步态的坐标。记xyz轴偏置的量为Sx，Sy，Sz。

此处前进步态只需要平移即可的所需坐标，如果是其他步态，如让机身旋转的步态，则需要加上一些旋转变换，这个变换同样可以用一个齐次变换矩阵表示。



再由机身坐标系下的步态曲线坐标转换到各腿分别坐标系下：

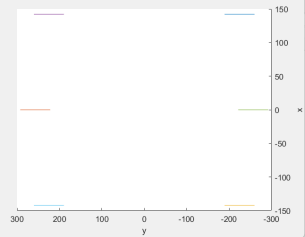
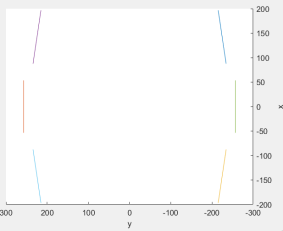
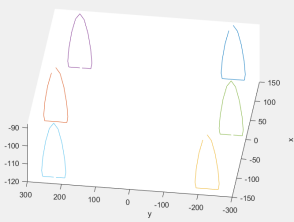
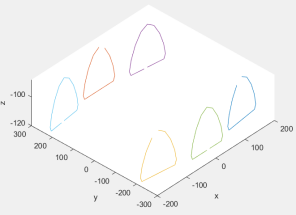
各条腿的坐标系的x轴方向为腿的初始静止摆放方向。同时六条腿具有不同的初始角度偏差，以使工作空间扇形不重叠。



第x（x=1,2,…6）条腿的基座坐标系（{x}）原点在机身坐标系（{0}）中的坐标即为{0}到{x}的齐次变换的偏移量，记为Dx，Dy，Dz。齐次变换的绕z轴旋转的角度也可由机械布局图可得，记为γ。

由此齐次变换矩阵得到各腿基座坐标系下的各腿步态曲线的坐标，再由逆运动学公式，得到每条腿三个关节的关节角。

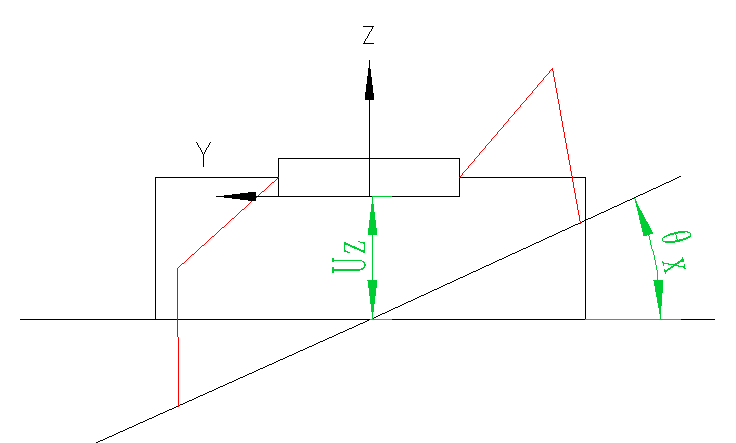
其他步态的生成同前进步态，都是先得到落脚点坐标系的曲线，再经过不同步态得到的齐次变换得到机身坐标系下的坐标，然后再经过一次由机身机械尺寸得到的齐次变换得到各腿基座坐标系下的坐标，最后由逆运动学公式得到各关节角。



## 静态姿态调整与算法优化

### 静态姿态调整

实现对于在有倾角的斜面上的机身稳定和角度调整，即使机器人在斜面上保持机身与原地面的水平。假设已用传感器测得未平衡时机身的roll（x轴）和pitch（y轴）角，已知旋转前后六条腿的落点都在一个平面上，要保持机身平稳，相当于六个脚的目标落点在原落脚平面旋转相反角度后形成的平面上。



我们令静止/水平时关节角θ2=90°，及刚好处于低机位和高机位的边界（这样旋转一定角度后一定一侧的3条腿和另一侧的3条腿机位相反，有利于后续算法优化）。由机械布局已知静态时的初始平面（z=Uz）和六个落脚点的坐标数据。令某条腿初始落脚点世界坐标为pi=[x0, y0, z0]，为了使机身平面旋转至roll角为θroll，pitch角为θpitch处，记该腿落脚点需要到达的新世界坐标为pn。

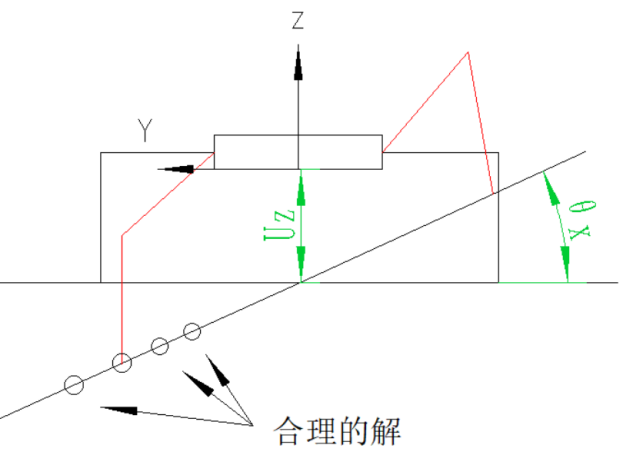
matlab代码：

rotXYAng2jointAng.m

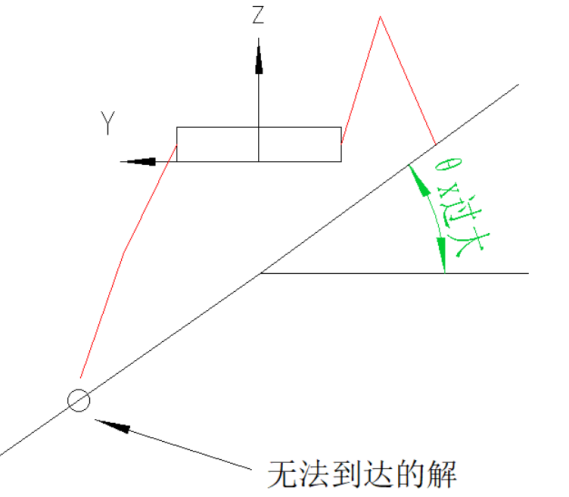
|  |
| --- |
| %rotXYAng2jointAng 由机身要转动的roll和pitch角算出六个脚的关节角  %  %[k]=rotXYAng2jointAng(rx,ry)  %k=3\*6的矩阵，包含六个脚的所有角度，可以用k(i)索引（i=1-18）；六列对应1-6号脚  %k=[r1;r4;...;r16  % r2;r5;...;r17  % r3;r6;...;r18]  % rx, ry 是角度制（degree）的旋转角  % roll pitch  function [k]=rotXYAng2jointAng(rx,ry)  k=zeros(3,6);  global yyy1 yyy2 xxx1 zzz1 % 导入全局变量  %机身脚位置初始偏置  xo=[xxx1,0,-xxx1,xxx1,0,-xxx1];  yo=[-yyy1,yyy2,-yyy1,yyy1,-yyy2,yyy1];  zo=[zzz1,zzz1,zzz1,zzz1,zzz1,zzz1];  %先由rx，ry算到x0y0z0（世界坐标系的目标位置）  p0=zeros(3,6);  p0(1,:)=xo.\*cos(ry\*(pi/180));  p0(2,:)=yo.\*cos(rx\*(pi/180));  p0(3,:)=zo-xo.\*sin(ry\*(pi/180))+yo.\*sin(rx\*(pi/180));  %p0->px（关节坐标系的目标位置）  jointCo=zeros(3,6);  for i=1:6  jointCo(:,i)=worldCo2jointCo(i,p0(:,i));  end  %px->q（求三个关节角）  for i=1:6  k(:,i)=jointCo2jointAng(jointCo(:,i));  end  end |

### 算法优化

实际上上述pn的算法只是一种合理的算法，实际上要达成使机身旋转的目标，可以有多种不同的落脚点算法，只要新的坐标能使新的六个脚的落脚点都在旋转后的目标平面内，且在机器人脚的工作空间内就行了。如下图情景，在图示直线上的许多点都可以是一个合理的解。



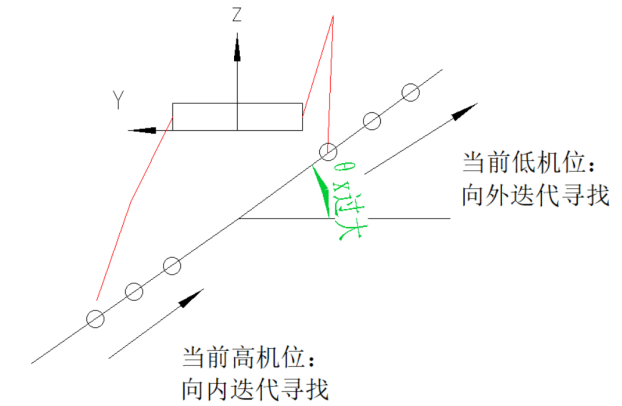
当目标到达角度过大时，上述算法可能会给出一个在脚工作空间以外的解：



但是这种情况并不是完全不能解决，旋转角在一定范围内时，腿的落脚点可以到达目标平面的其他点。一种简单的迭代求解法图示如下：当发现上述算法的解在脚的工作空间外时，根据这个无解的坐标向内（靠近原点方向）或向外（远离原点方向）迭代几次，然后检验新坐标是否在脚的工作空间内，以此可能得到一个合法的解。



前面提到，静止/水平的时候使关节角θ2=90°，及刚好处于低机位和高机位的边界处，这样旋转后一侧的3条腿和另一侧的3条腿的机位相反。在进行迭代寻求合法解的时候，我们会依靠当前腿的机位来决定迭代的方向。规则为高机位向内迭代，低机位向外迭代。



优化后matlab代码：

rotXYAng2jointAng.m

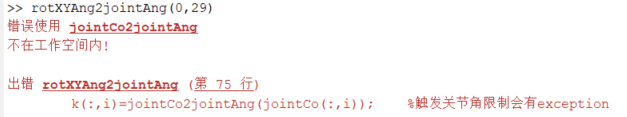
|  |
| --- |
| function [k]=rotXYAng2jointAng(rx,ry)  k=zeros(3,6);  global yyy1 yyy2 xxx1 zzz1  %机身脚位置初始偏置  xo=[xxx1,0,-xxx1,xxx1,0,-xxx1];  yo=[-yyy1,yyy2,-yyy1,yyy1,-yyy2,yyy1];  zo=[zzz1,zzz1,zzz1,zzz1,zzz1,zzz1];  %先由rx，ry算到x0y0z0（世界坐标系的目标位置）  p0=zeros(3,6);  p0(1,:)=xo.\*cos(ry\*(pi/180));  p0(2,:)=yo.\*cos(rx\*(pi/180));  p0(3,:)=zo-xo.\*sin(ry\*(pi/180))+yo.\*sin(rx\*(pi/180));  %p0->px（关节坐标系的目标位置）  jointCo=zeros(3,6);  for i=1:6  jointCo(:,i)=worldCo2jointCo(i,p0(:,i));  end  **function qn=findNext(ii,p00)**  **cnt=0;**  **pn=p00;**  **flag=0;**  **while(cnt<=3)**  **pn(3)=pn(3)-zzz1;**  **if(p00(3)>-117.9)**  **% disp("低机位");**  **pn=pn.\*1.2; %迭代规则比较简单，以原点为中心乘一个系数，找到的下一个点是到落脚点到平面原点连线上的一个点**  **else**  **% disp("高机位");**  **pn=pn.\*0.8;**  **end**  **pn(3)=pn(3)+zzz1;**  **cnt=cnt+1;**  **pxn=worldCo2jointCo(ii,pn);**    **try**  **qn=jointCo2jointAng(pxn);**  **flag=1;**  **break**  **catch**  **continue**  **end**  **end**    **assert(flag~=0,"无解");**  **end**  **%px->q（求三个关节角）**  **for i=1:6**  **try**  **k(:,i)=jointCo2jointAng(jointCo(:,i)); %触发关节角限制会有exception**  **catch**  **k(:,i)=findNext(i,p0(:,i));**  **end**  **end**  end |

### 优化结果

优化前：

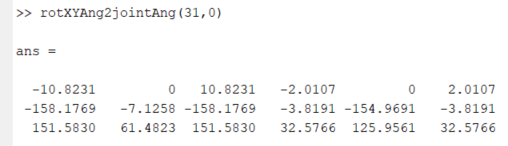
单边旋转极限：roll角为16°以下，pitch角为29°以下



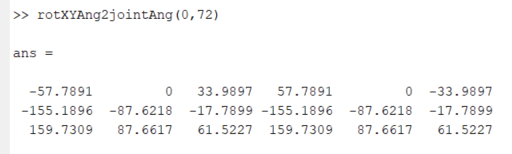


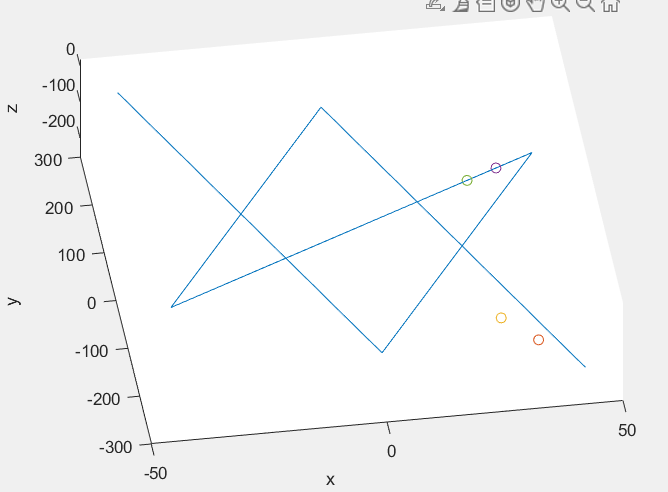
优化后：

单边旋转极限：roll角为31°以下，pitch角为72°以下。两个极限被显著的提高了，当然实际上受到底盘离地高度的限制，在一个更小的角度范围时机器人的底盘可能已经会与地面接触，过大的角度极限实际上没有实际用处，即优化后算法的有解范围已经超过了实机的机械限位范围。



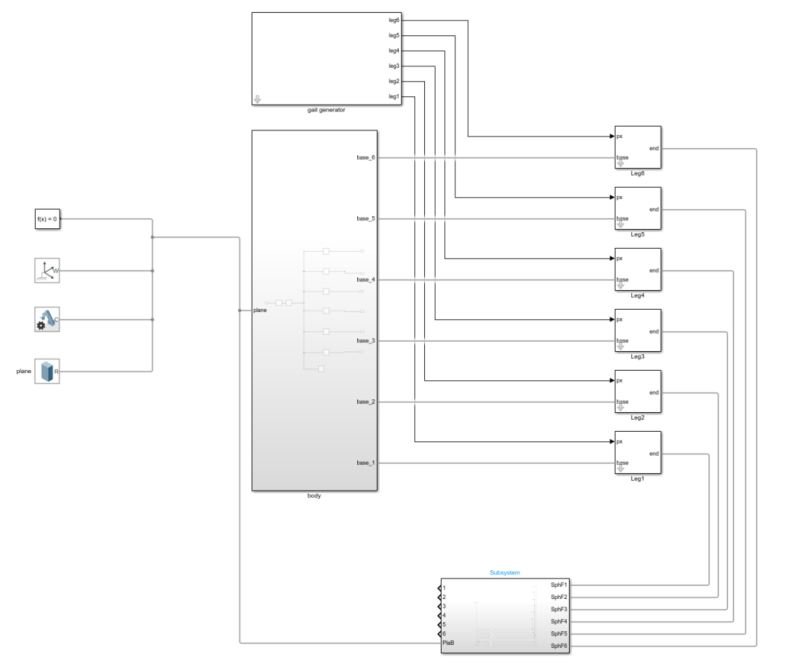






图中圆圈为迭代过程的的新落脚点坐标。

## Simulink仿真



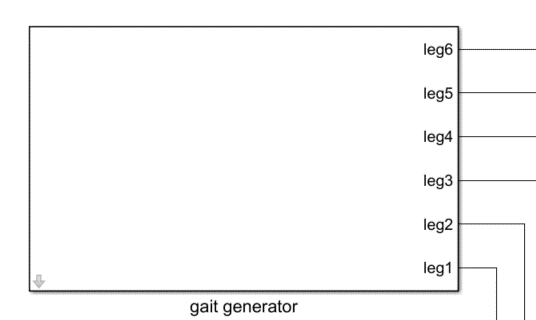
基于上述算法，在simulink中构建机器人的模型来验证各种步态的可行性。

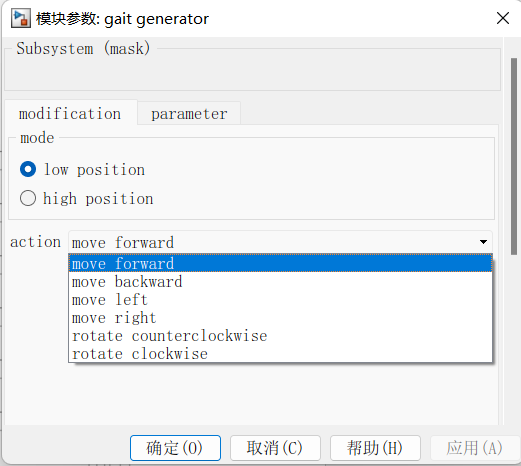
步态生成部分和每条腿分别封装。

求解器采用定步长的ode2。



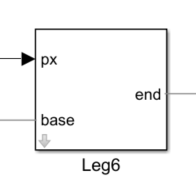
步态信号生成器：

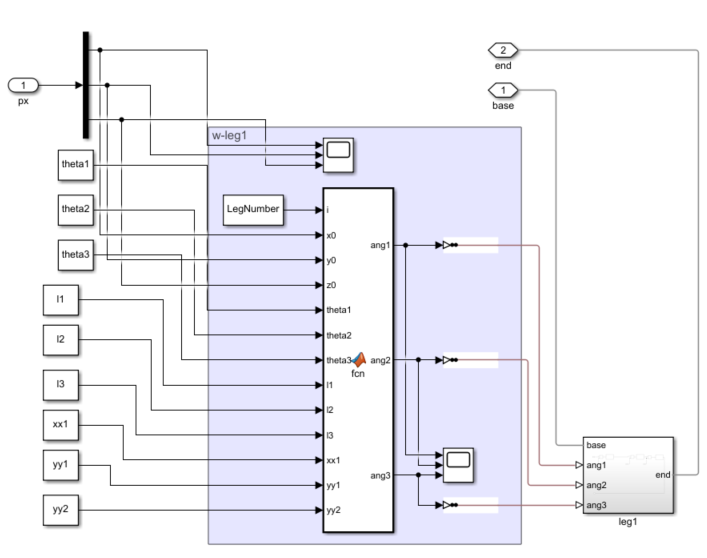


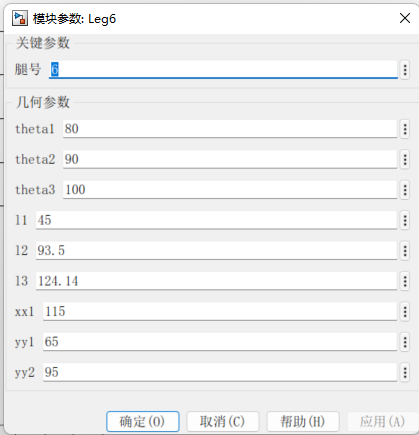


从封装可以修改步距，步高等参数，以及选择预置的几种步态类型来进行验证。

单条腿：

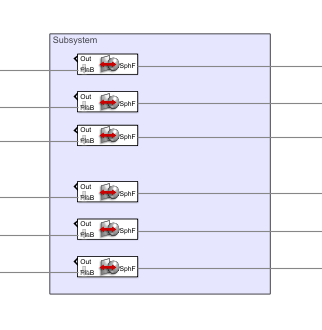


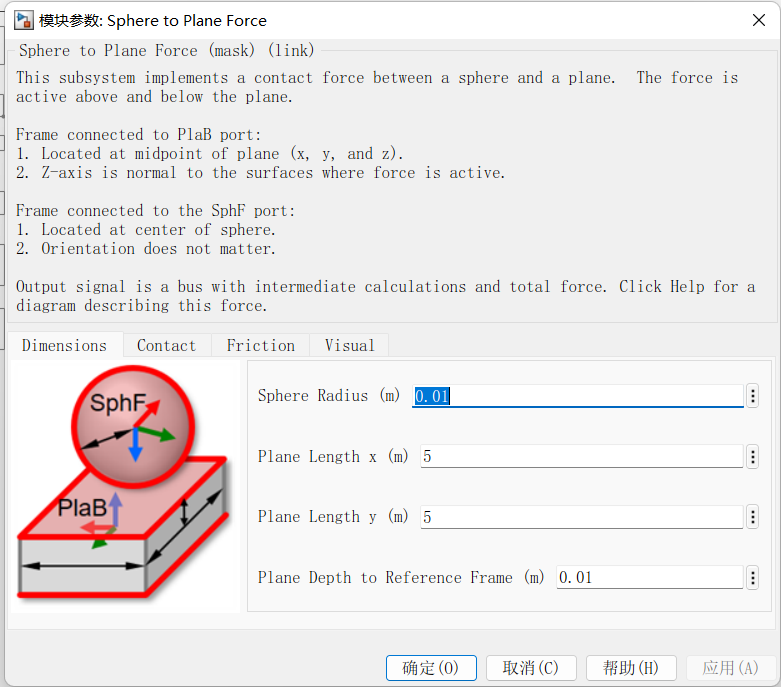




从顶层封装可以修改每条腿的关键参数和几何参数，应保证六条腿的几何参数相同。

地面反馈：





使用了Simscape Multibody Contact Forces Library的Sphere to Plane Force模块，调整后的参数如图上。

# 基础方案

## 项目目标

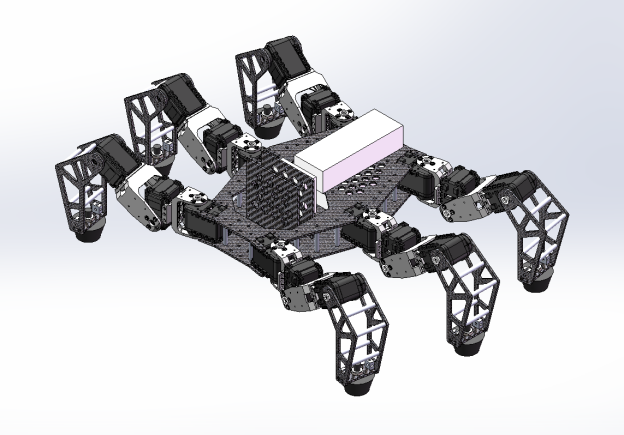
完成六组机器人的步态，自稳等功能的原理数学建模。

设计合理的机械结构支撑我们的硬件软件平台。

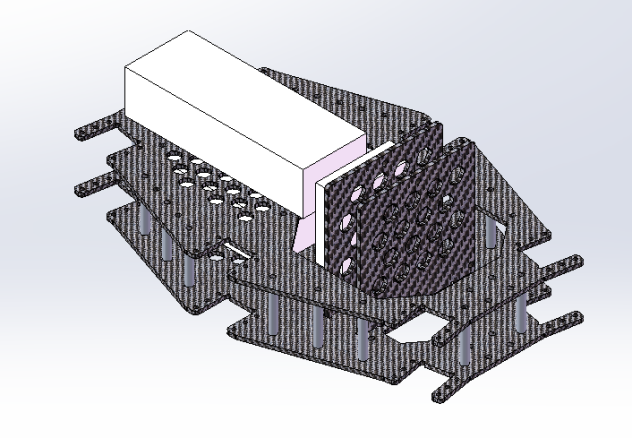
完成六足机器人的搭建和硬件安装，在上面实现基本步态和自稳功能。

在完成以上所有目标后设计合理的实验测量评估六组机器人的各项参数。

## 机械设计

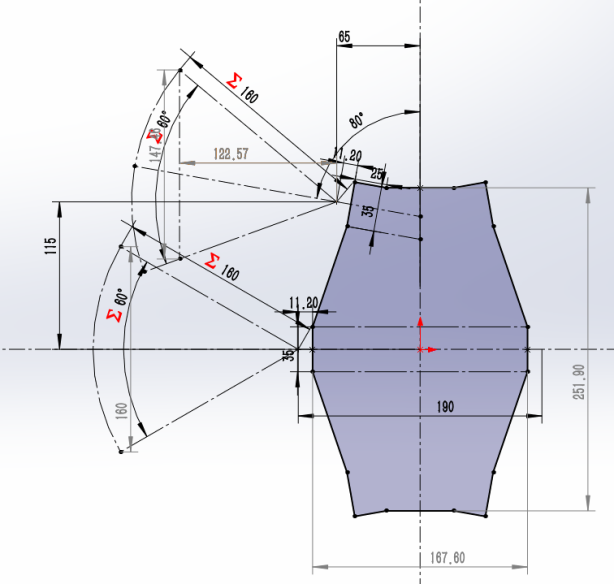


### 主体结构及基本尺寸



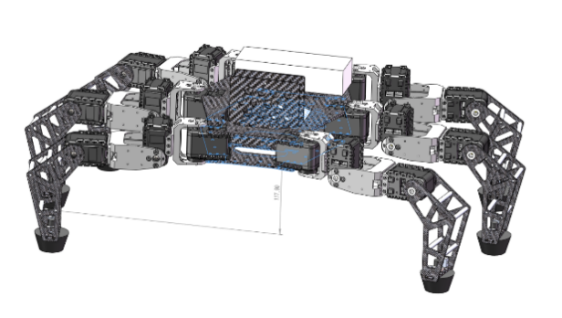
主体结构由碳纤维板和铝柱结合的结构组成，底盘本身质量很轻，约200g。夹层和背部有用来放置硬件和电池的空间，前部设计有用于拓展的竖置的放置硬件的空间和固定孔位。

与数学建模相关的底盘机械参数如图：



通过关节角的变化，分为高机位和低机位。下图是设定的高低机位的起止位置，从机械模型中可以得到起始的六个落脚点的坐标数据，用于后续计算。

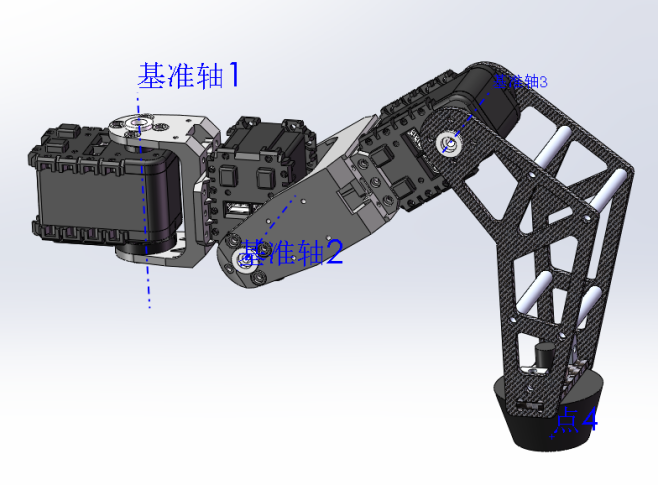
高机位：

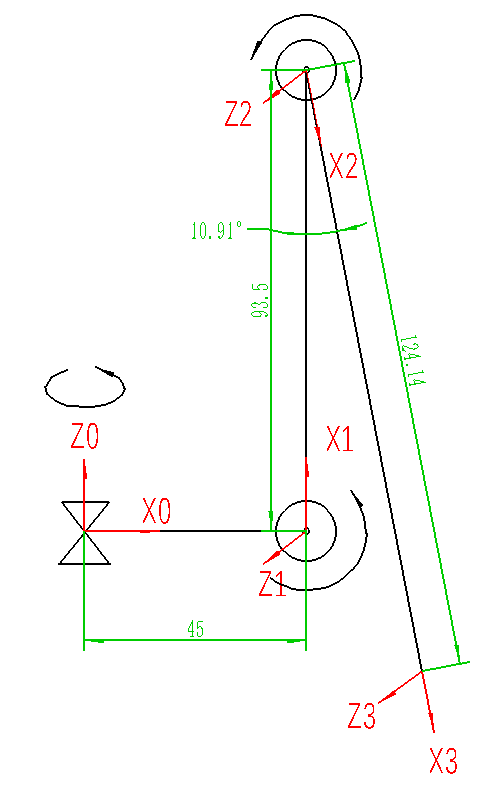


低机位：



### 单腿设计和尺寸

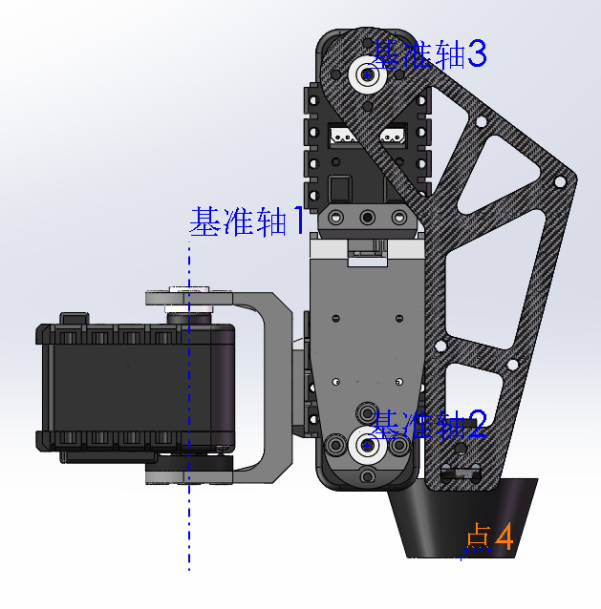




DH参数表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | a | α | d | θ |
| 1 | 45 | 90° |  | 90°+θ1 |
| 2 | 93.5 |  |  | -169.09°+θ2 |
| 3 | 124.76 |  |  | θ3 |

机械建模与前文的DH建模相一致。



当腿收缩到机械限位时，从图中轴3到点4的连线与轴2到轴3的连线中有10.91°的基础偏移角，与前文DH建模中加上的偏移相同。

## 机电方案：

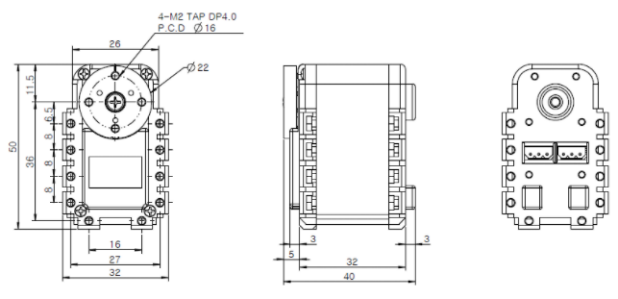
### 需求分析

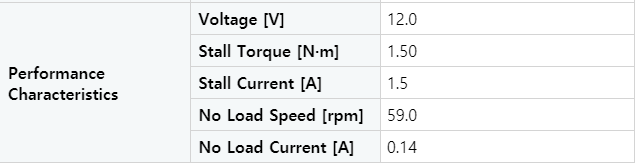
制造出能完成基本移动，自稳控制的六足机器人，为了保证每条腿3个关节的自由度和整个机器人六条腿的运动需求，我们总共需要6\*3一共18个舵机。传感器层面，为了完成自稳要求，我们需要实时读取机器人姿态，所以我们需要姿态传感器来返回机器人的姿态信息。最后实现数据处理的控制器，需要完成传感器数据读取和对舵机控制，且要保证一定的数据处理速度，六足机器人的运动学解算需要一定的时间。

### 硬件选型

执行器：

#### DYNAMIXEL AX-12A



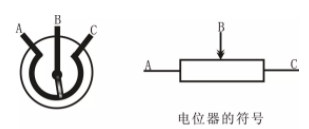


该款舵机支持9V到12V的工作电压，在减速比为254:1的前提下提供最大1.5N\*m的扭矩，最佳状态下满足空载59rpm的转速。

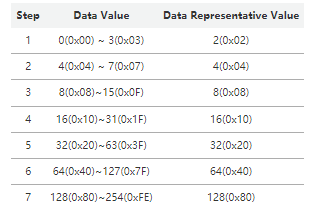
该舵机的基本工作原理是基于TTL通讯的前提下，接收外界满足协议的通讯包，来满足外界需求。外界接口API包括设置运动模式,目标位置，目标速度，运动速度，控制ID等,这些数据构成了驱动舵机的控制表：



而更底层的运动控制，包括位置修正的底层算法是有舵机内嵌硬件控制。对于位置控制AX-12A使用了电位器作为位置控制器，

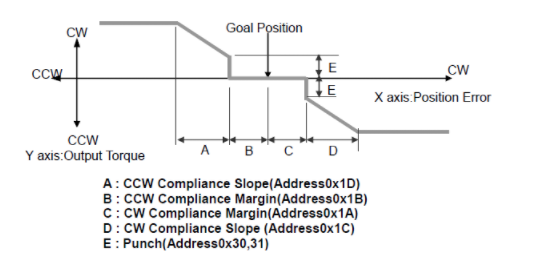


它是由一个电阻体和一个转动或滑动系统组成。当电阻体的两个固定触点之间外加一个电压时，通过转动或滑动系统改变触点在电阻体上的位置，在动触点与固定触点之间便可得到一个与动触点位置成一定关系的电压。来映射旋转位置的具体数值。

相较于传统的PID控制，其底层的控制原理比较简单，为了防止欠调和超调AX-12A定义了参数Compliance Slope，

针对不同的调节范围定义了7个级别，这个值越大为了纠正Position Error的反向电流越大，而为了防止小震荡，AX\_12A同时定义了另一个参数Compliance Margin ，当Position Error处于这个区间时不进行误差调整。而切所有的驱动电流都满足一个最小电流Punch的倍数。

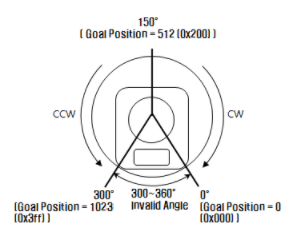
整个过程如下图所示：



而AX-12A给予我们的操作空间就是这5个参数。

区别于AX-12A的speed mode我们使用的position mode同时为了区分存在参数CW/CCW Angle Limit，来限制舵机的旋转角度

，即使给定数值为100，其仍只有300度的范围

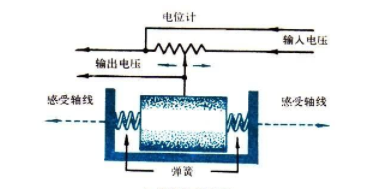


我们整个项目对舵机的需求在于关节控制，使用的是AX-12A的位置模式，关注的参数是精确度，最大扭矩和工作范围，考虑到我们使用的机身结构为碳板和铝柱组成，质量不大，且支撑脚之间的距离不是过大，所以1.5N\*m的最大扭矩复合我们的要求，300的工作范围和精度基本上能满足我们的基本步态，

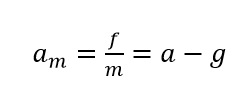
传感器：

世面上的姿态传感器基本上三种传感器的结合

#### 加速度计



加速度传感器[4]是一种能够测量加速度的传感器。通常由质量块、阻尼器、弹性元件、敏感元件和适调电路等部分组成。传感器在加速过程中，通过对质量块所受惯性力的测量，利用牛顿第二定律获得加速度值。



具体的实现方式有以下四种

①参考刚体由弹簧与壳体相连，它和壳体的相对位移反映出加速度分量的大小，这个信号通过[电位器](https://www.eefocus.com/baike/500767)以电压量输出

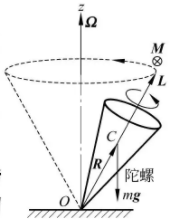
②参考刚体由弹性细杆与壳体固连，加速度引起的动载荷使杆变形，用应变电阻丝感应变形的大小，其输出量是正比于加速度分盘大小的电信号

③参考刚体通过压电元件与壳体固连，质量的动载荷对压电元件产生压力，压电元件输出与压力即加速度分量成比例的电信号

④参考刚体由弹簧与壳体连接，放在线圈内部，反映加速度分量大小的位移改变线圈的电感，从而输出与加速度成正比的电信号。

#### 陀螺仪(角速度传感器)

陀螺仪是用高速回转体的[动量](https://link.zhihu.com/?target=https://baike.baidu.com/item/%E5%8A%A8%E9%87%8F" \t "https://zhuanlan.zhihu.com/p/_blank)矩敏感壳体相对[惯性](https://link.zhihu.com/?target=https://baike.baidu.com/item/%E6%83%AF%E6%80%A7)空间绕正交于自转轴的一个或二个轴的角运动检测装置。



在一定的[初始条件](https://link.zhihu.com/?target=https://baike.baidu.com/item/%E5%88%9D%E5%A7%8B%E6%9D%A1%E4%BB%B6" \t "https://zhuanlan.zhihu.com/p/_blank)和一定的外在[力矩](https://link.zhihu.com/?target=https://baike.baidu.com/item/%E5%8A%9B%E7%9F%A9)作用下，陀螺会在不停自转的同时，环绕着另一个固定的转轴不停地旋转，这就是陀螺的旋进(precession)，又称为回转效应(gyroscopic effect)。即一个旋转物体的旋转轴所指的方向在不受外力影响时，是不会改变的。

陀螺仪的基本部件

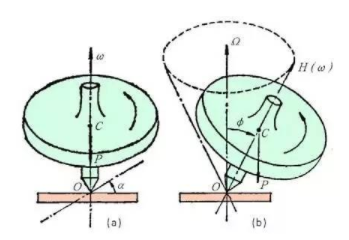


1、陀螺转子（常采用同步电机、磁滞电机、三相交流电机等拖动方法来使陀螺转子绕自转轴高速旋转，并见其转速近似为常值）。

2、内、外框架（或称内、外环，它是使陀螺自转轴获得所需角转动自由度的结构）。

3、附件（是指力矩马达、信号传感器等）。

##### 陀螺仪的两个重要特性



定轴性

当陀螺转子以高速旋转时，在没有任何外力矩作用在陀螺仪上时，陀螺仪的自转轴在惯性空间中的指向保持稳定不变，即指向一个固定的方向；同时反抗任何改变转子轴向的力量。这种物理现象称为陀螺仪的定轴性或稳定性。

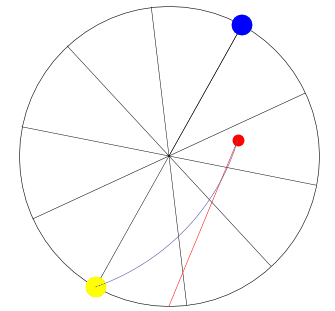
进动性

当转子高速旋转时，若外力矩作用于外环轴，陀螺仪将绕内环轴转动；若外力矩作用于内环轴，陀螺仪将绕外环轴转动。其转动角速度方向与外力矩作用方向互相垂直。这种特性，叫做陀螺仪的进动性。

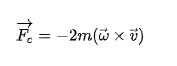
##### 陀螺仪测量敏感角速度原理

陀螺仪的基本原理是牛顿第二定律。如果在牛顿方程中增加旋转，就会出现虚拟力，数学术语称之为科里奥利力。陀螺仪的工作原理是把旋转转换为科里奥利力。陀螺仪需要主运动或进给运动，科里奥利力是两种正交运动的结果，并且与二者正交。

##### 科里奥利力

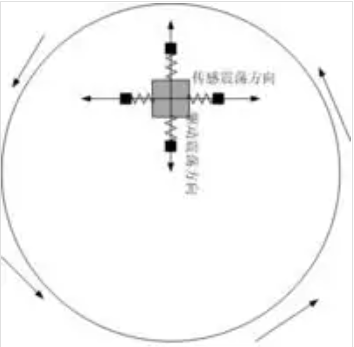


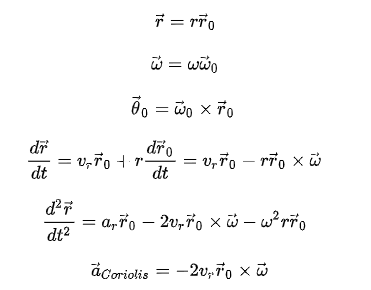
对旋转体系中进行直线运动的质点由于[惯性](https://baike.baidu.com/item/%E6%83%AF%E6%80%A7/3595487" \t "https://baike.baidu.com/item/%E7%A7%91%E9%87%8C%E5%A5%A5%E5%88%A9%E5%8A%9B/_blank)相对于旋转体系产生的直线运动的偏移的一种描述。科里奥利力来自于物体运动所具有的惯性。



如果我们可以控制物体做直线运动的速度，知道物体的质量，通过测量它在旋转时受到的科里奥利力大小，就可以计算出角速度的大小，从而知道物体的旋转状态。

陀螺仪利用陀螺的定轴性与进动性来敏感某一方向的角速度。

在空间设立动态坐标系（图一）。用以下方程计算加速度可以得到三项，分别来自径向加速、[科里奥利加速度](https://baike.baidu.com/item/%E7%A7%91%E9%87%8C%E5%A5%A5%E5%88%A9%E5%8A%A0%E9%80%9F%E5%BA%A6/4383598" \t "https://baike.baidu.com/item/%E5%BE%AE%E6%9C%BA%E6%A2%B0%E9%99%80%E8%9E%BA%E4%BB%AA/_blank)和[切向加速度](https://baike.baidu.com/item/%E5%88%87%E5%90%91%E5%8A%A0%E9%80%9F%E5%BA%A6/895020)。



为了产生[科里奥利力](https://baike.baidu.com/item/%E7%A7%91%E9%87%8C%E5%A5%A5%E5%88%A9%E5%8A%9B/1255543" \t "https://baike.baidu.com/item/%E5%BE%AE%E6%9C%BA%E6%A2%B0%E9%99%80%E8%9E%BA%E4%BB%AA/_blank)，[陀螺仪](https://baike.baidu.com/item/%E9%99%80%E8%9E%BA%E4%BB%AA/84317)的设计上，这个物体被驱动，不停地来回做径向运动或者震荡，与此对应的科里奥利力就是不停地在横向来回变化，并有可能使物体在横向作微小震荡，相位正好与驱动力差90度，陀螺仪通常有两个方向的可移动电容板。径向的电容板加震荡电压迫使物体作径向运动，横向的电容板测量由于横向科里奥利运动带来的电容变化。因为科里奥利力正比于[角速度](https://baike.baidu.com/item/%E8%A7%92%E9%80%9F%E5%BA%A6/1532689)，所以由电容的变化可以计算出角速度。

传统的姿态传感器有两种组合一种是陀螺仪加加速度计组成的6轴传感器这种组合在pitch轴和roll轴上的表现优秀（具体原因会在之后分析），但在yaw轴上会出现误差过大的情况，另一种组合是陀螺仪加加速度计加地磁传感器组成的九轴传感器，虽然能保证yaw轴的修正但计算也会更加复杂，考虑到我们的自稳明显体现在和pitch和roll轴保持水平而yaw轴的角度与自稳之间的联系不大，所以我们选择用六轴结构的姿态传感器。

控制器：

对于AX-12A的控制方案核心在于产生TTL电平，这里有两种方案

#### U2D2

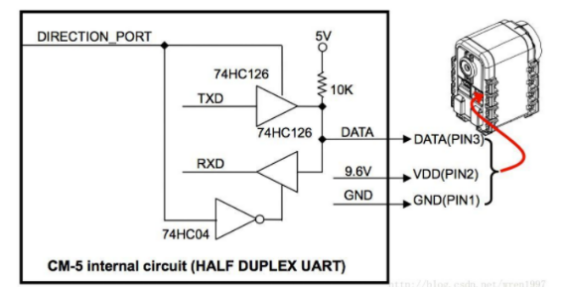


U2D2是一个USB通信转换器，其核心功能在于筛选信息：当数据从PC中的USB端口传输到外围设备（TX）时，所有连接的设备将接收相同的数据。另一方面，当从外围设备之一（RX）收到数据时，它将自动转换为USB通信并发送到PC。

#### UART

除了使用U2D2，Dynamixel官方给我们另一种方式：

与其他数字舵机不同的是，Dynamixel系列的舵机并不是通过PWM信号来控制，而是使用一种读写RAM的通信协议。这种通信基于半双工UART端口，只需要一根总线便可以同时接受和发送信号。（半双工的意义在于使用一条主线完成收发操作）



上述结构完成了对全双工串口到半双工的改造。（当DIRECTION\_PORT高电平使，TXD可使用，低电平时，RXD可使用）

### 控制方案

控制方案一：

使用官方的Dynamixel shield



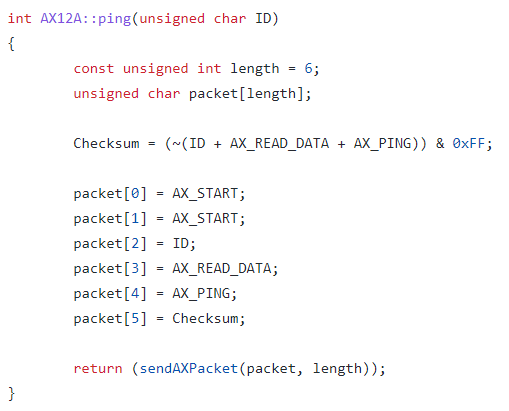
使用Arduino的硬串口实现与shield的串口通信再通过shield自己的计算电路转化为TTL电平信息，舵机供电从shield上走，Arduino只做信息交换处理不过大电流

控制方案二：

利用Arduino自带的硬串口的read，write函数使用单TX串口在函数代码层面完成全双工到半双工的转变，来模拟TTL电平。







这种控制同时保留只做处理不过大电流的优点

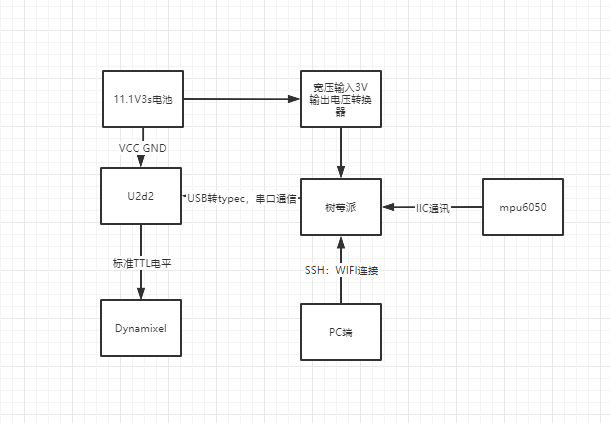
控制方案三：

树莓派+U2D2



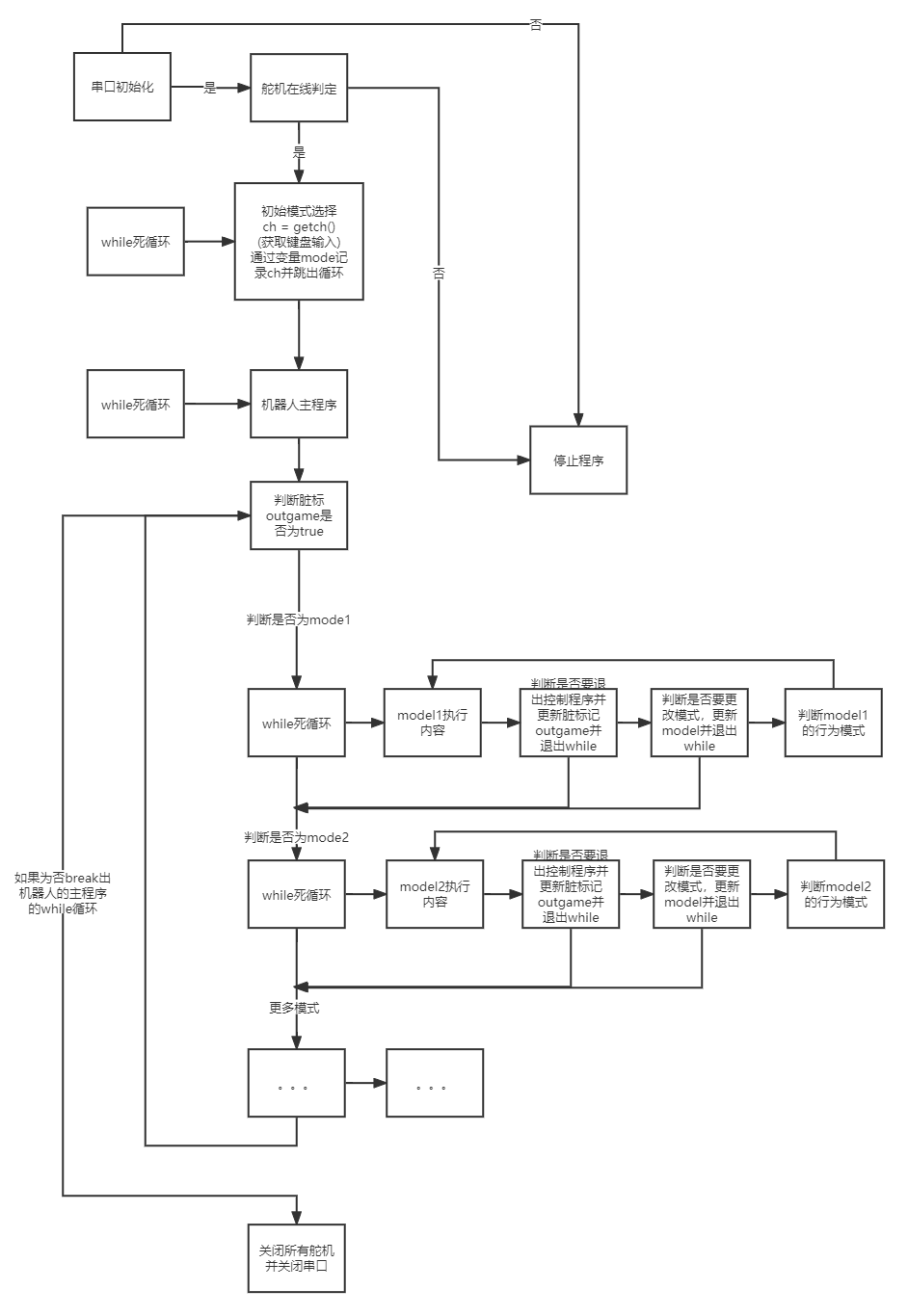
在保证了传感器接口的前提下，相较于Arduino数据处理能力大大增加。

机电连线图：



### 控制逻辑

#### 代码框架



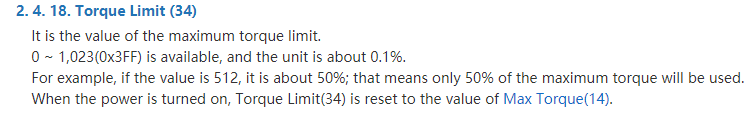
#### 运动模式

代码框架细节

注意点：

1.程序代码知识给舵机一个指令，如果代码运行速度超过舵机的运行速度，会导致舵机运动不到位就执行下一个动作导致动作幅度小等问题，所以我们需要做位置检验。

2.如果需要更精细的控制我们可以设定pid位置控制（由于Dynamixel留给我们的外部接口很少我们只能做到伪的位置控制）因为我们不能直接影响舵机内部的电流控制，但我们改变舵机的力矩上限，在control table 上有个参数 Max torque



这个参数能保证单次调整不会超过的扭矩上限，我们虽然无法直接控制电流，但能通过扭矩上限间接设置电流，假设我们的功率最大的情况下扭矩上限合适，那么电流的设定值应该是和扭矩上限是一一对应的线性关系，这样的话可以改善过冲现象，做到精准调控。

|  |
| --- |
| if curP > targetPosition + margin:  servo.goal\_position = minP  servo.torque\_limit = finalTorque  elif curP < targetPosition - margin:  servo.goal\_position = maxP  servo.torque\_limit = finalTorque |

而为了得到扭矩限制finalTorque，我们可以读取currentposition来实现PID计算

误差计算

|  |
| --- |
| error = targetPosition - curP |

KP计算

|  |
| --- |
| # Proportional torque  propTorque = error \* propGainConstant  # Limit to allowed servo range  propTorque = min( (abs(propTorque), 1023) ) |

KI计算

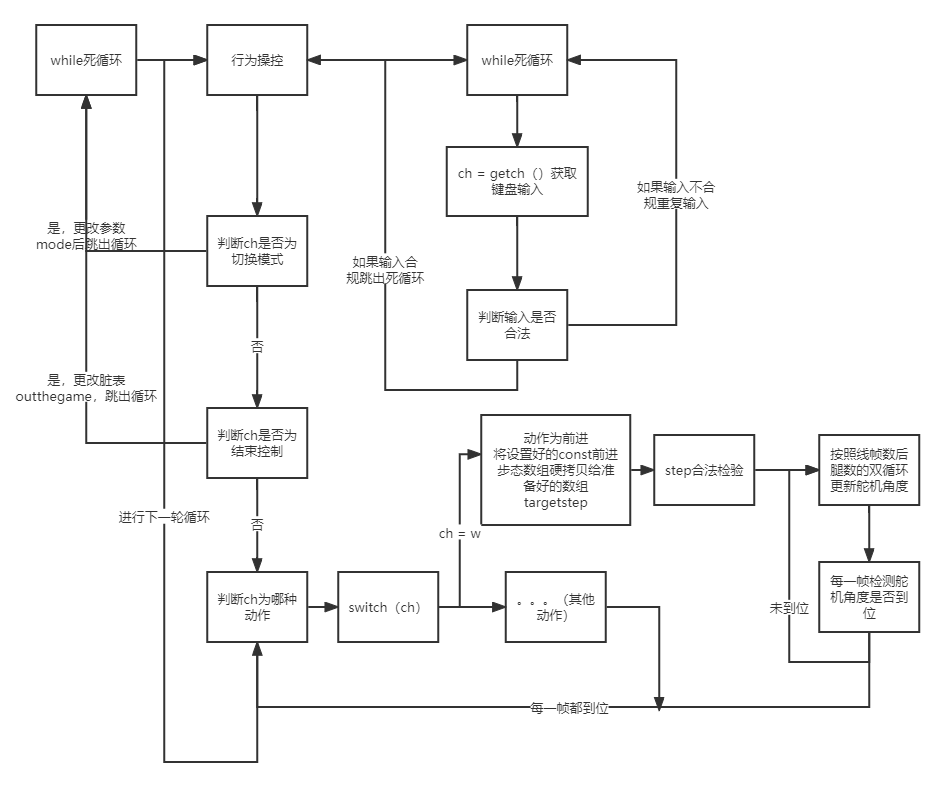
|  |
| --- |
| # The sum of errors \* their time delta in our window  integralTotal = sum([e\*td for e, td in window])  # Here's where the integral tuning parameter comes in.  integralContribution = integralTotal \* integralGain |

KD计算：

|  |
| --- |
| if tDelta:  eDelta = absError - ePrev  # Update for next iteration  ePrev = absError  errorSlope = eDelta / tDelta  derivativeContribution = derivativeGain \* errorSlope |

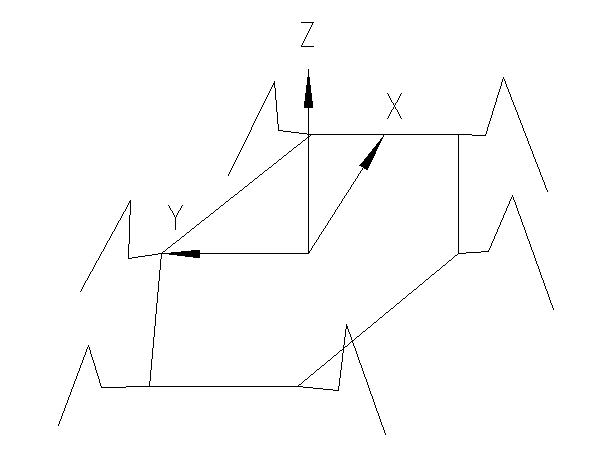
FinalTorque求和

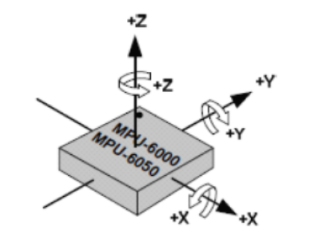
|  |
| --- |
| # Sum the contributions  finalTorque = propTorque + integralContribution + derivativeContribution  # Convert to a whole number  finalTorque = int(round(finalTorque)) |



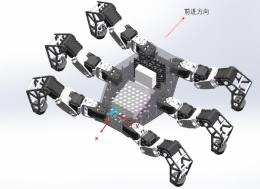
#### 平衡模式细节

我们的目标是实现机体的水平，而我们返回



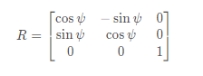


考虑到安装要求，我们最后选择将IMU安装在如下图所示位置。

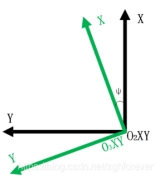


这样机体坐标系和姿态传感器坐标系的转变需要两部分：由于安装位置的旋转矩阵和安装误差产生的矩阵。

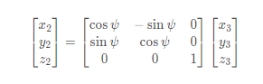
绕Z轴逆时针旋转的[旋转矩阵](https://so.csdn.net/so/search?q=%E6%97%8B%E8%BD%AC%E7%9F%A9%E9%98%B5&spm=1001.2101.3001.7020)定义为:



考虑到车体与传感器之间只存在绕Z轴的旋转，故坐标系旋可以只考虑OXY平面内的旋转变换。

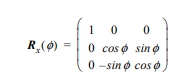


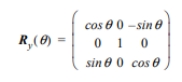
变换关系可以简化表示如下：



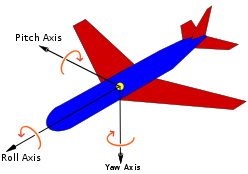
对于平衡机体的目的，我们只需要roll和pitch的旋转角度，而OXY坐标系相对位置的改变并不会改变这两个角的角度，所以我们并不需要对坐标系进行平移操作。

对于安装的误差我们只需要对X轴和Y轴进行旋转修正，我们给机体设置一个初始状态，保证机体的水平（可以使用水平仪判断或者记录误差），接着我们读取IMU的roll，pitch数值，这两个值就是机体与传感器的角度误差，结合上述的误差，我们通过两个旋转矩阵



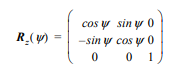
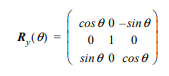
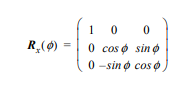


就能完成坐标系的转化。



我们使用的加速度计返回的数据为x，y，z三轴的线加速度，而这里的x，y，z轴并不是我们广义的roll，pitch，yaw轴，这三个轴是随着传感器的姿态实时旋转的。初始状态下，我们只受地心引力的印象，考虑到我们姿态传感器的安装位置不可能完全水平，三轴上会有初始角度，这就导致重力会在三个轴上产生加速的分量，由这三个分量，我们就可以得到初始的三轴旋转角。

我们定义旋转矩阵如下：



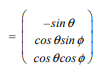
定义重力加速度如下：

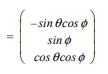
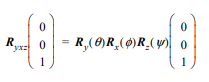


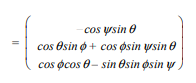
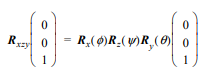
我们就可以得到关系式

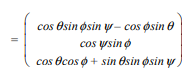
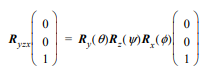


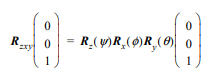
除了这样的顺序，我们同时会有这样的形式一共6种顺序的矩阵乘法，不同顺序的矩阵乘法，做得到的结果就不同，六种结果如下所示：

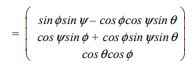
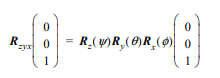






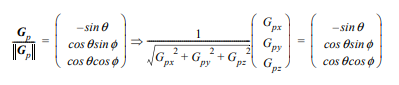


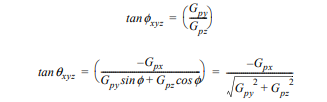


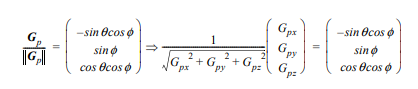


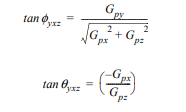
分析上述结果，我们可以直接排除4种结果，形如后四种重力的分量是由三个角度决定，这就说明每个对应的一组三个值会得到一个确定的重力分量向量。而实际情况下，重力加速度分量满足一个等式向量模长为1g，这就导致这个角度向量其实只有2个自由度，而后四者其实有三个自由度，换句话说，后四者的不一定满足模长为1g的限制条件。而前两者分量只取决于[θ和φ，这很符合实际情况，对于ψ轴的旋转并不印象重力加速度在x，y，z三轴上的分量。](https://zhidao.baidu.com/question/79232975.html" \t "https://www.google.com.hk/_blank)

[接着我们分析前两种情况：](https://zhidao.baidu.com/question/79232975.html" \t "https://www.google.com.hk/_blank)

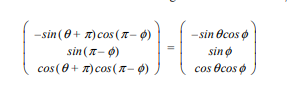
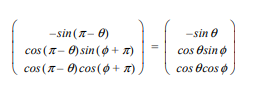
[](https://zhidao.baidu.com/question/79232975.html" \t "https://www.google.com.hk/_blank)







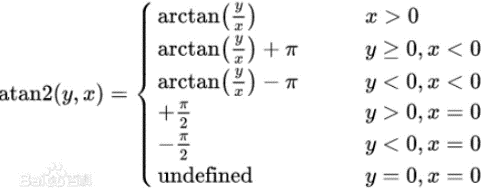
虽然确定了具体θ和φ的反三角函数，但我们仍需要对他们进行角度限制，以防出现



这种情况。

解决方案是限制roll或pitch（但不是两者都位于-90°和 +90°之间，这样会大幅限制姿态）。航空航天序列中使用的惯例是，roll角可以在-180°至 +180°之间范围，但pitch角限于-90°至 +90°。智能手机和Microsoft在其Windows 8传感器平台上使用的约定是相反的，roll角限制在-90°至90°之间，但pitch角度的范围可以在-180°和180°之间。

而我们限制-180到180的角度，我们则需要atan2函数



而对于剩下的-90到90的角度，则使用普通atan函数限制角度即可

在具体代码上用以下方式实现：

|  |
| --- |
| if (RestrictPitch):     roll = math.atan2(accY,accZ) \* radToDeg     pitch = math.atan(-accX/math.sqrt((accY\*\*2) +(accZ\*\*2))) \* radToDeg  else:     roll = math.atan (accY/math.sqrt((accX\*\*2) +(accZ\*\*2))) \* radToDeg     pitch = math.atan2(-accX,accZ) \* radToDeg |

解算完了加速度计的数据处理，接下来我们需要处理陀螺仪的数据，陀螺仪返回的速度是三个轴的角速度，如果我们的时间很短的话，积分角速度和时间的乘积，那我我们就能得到roll，pitch的角度。

但考虑的微分时间的误差，传感器的测量误差我们这样计算的结果不是很准确，我们有两种解决方案。

互补滤波器：

|  |
| --- |
| compAngleX = 0.93 \* (compAngleX + gyroXRate \* dt) + 0.07 \* roll  compAngleY = 0.93 \* (compAngleY + gyroYRate \* dt) + 0.07 \* pitch |

我们可以使用加速度计解算出来的值和陀螺仪积分所得的值加权求平均值来得到相较准确的答案，互补滤波器的系数一般选为0.93和0.07

卡尔曼滤波[5, 6]：

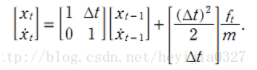
我们可以使用状态空间方程来描述预测值和观测值：



我们假设Xk为陀螺仪在k时刻的状态，我们能使用这样的方程来描述上一刻k-1时刻与这一刻的关系。

B矩阵 表示控制输入到当前状态的转换矩阵，A 表示上一状态到当前状态的转换矩阵，w表示过程噪声，主要是从上一状态进入到当前状态时，会有许多外界因素的干扰

一般可以套用运动方程（将状态表示为角度位置和角速度）



但为了和稳态误差联系更加直接，我们可以将状态表示为角度位置和角速度误差





和真实的状态方程一致，预测状态同样有这样的效果，但由于我们无法估计出过程误差所以我们引入一个新的概念先验估计值



接着我们来表示预测值，可以使用方程：



相对的我们同样无法准确表示出测量误差，所以我们引入另一个变量忽略噪声的先验测量值

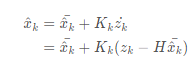


那么就可以用实际得到的测量值减去无噪声由上一状态预测的测量值



这个值表示预测的测量值和实际测量值的差异即数字化过程噪声和测量噪声对当前测量值的影响。

有了这两个值，卡尔曼滤波中采用按照比例组合的方式来结合无噪声估计先验估计和噪声变量。



为了得到最优的Kk值，我们的评定准则是使误差最小，这里我们定义两种误差：先验误差和估计误差



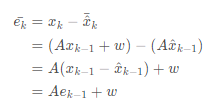
目标就成最小化后验状态估计的误差协方差矩阵的迹，误差协方差矩阵的对角线表示了误差相对于过程误差和测量误差的方差



经过一系列数学变化后我们能计算出最优解



为了计算



计算协方差矩阵



我们假设的估计误差与过程误差的均值都为0



化简得到：

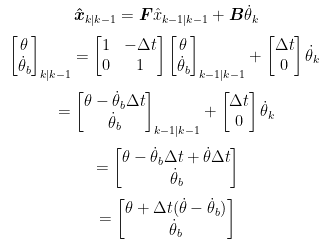


至此卡尔曼滤波的5个过程结束

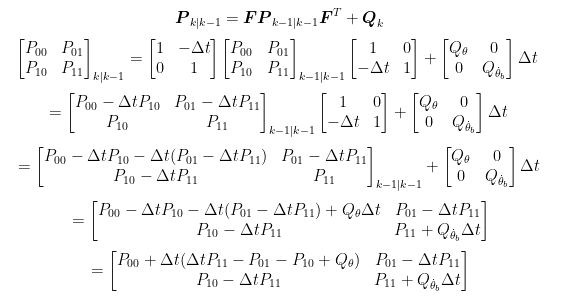


与传统的卡尔曼滤波不同由于我们使用的描述状态为角速度误差，5处的具体矩阵乘法如下：

Step1：



Step2：

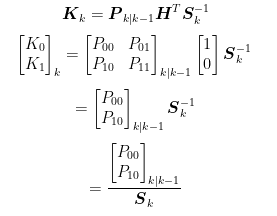
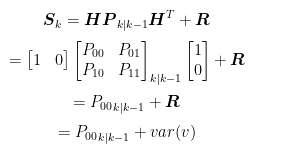


Q\_K是过程噪声协方差矩阵,考虑到dt越大，更新时间越长，过程噪声会更明显。所以：

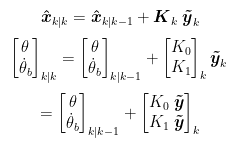


这里的角度误差和角速度误差都是一个假定的常数，是可以更改的，如果更相信角度的估计值可以减小角度误差，如果角度开始漂移，则需要增加角速度误差

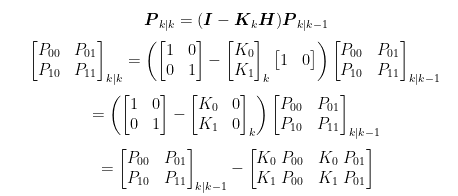
Step3：



Step4:



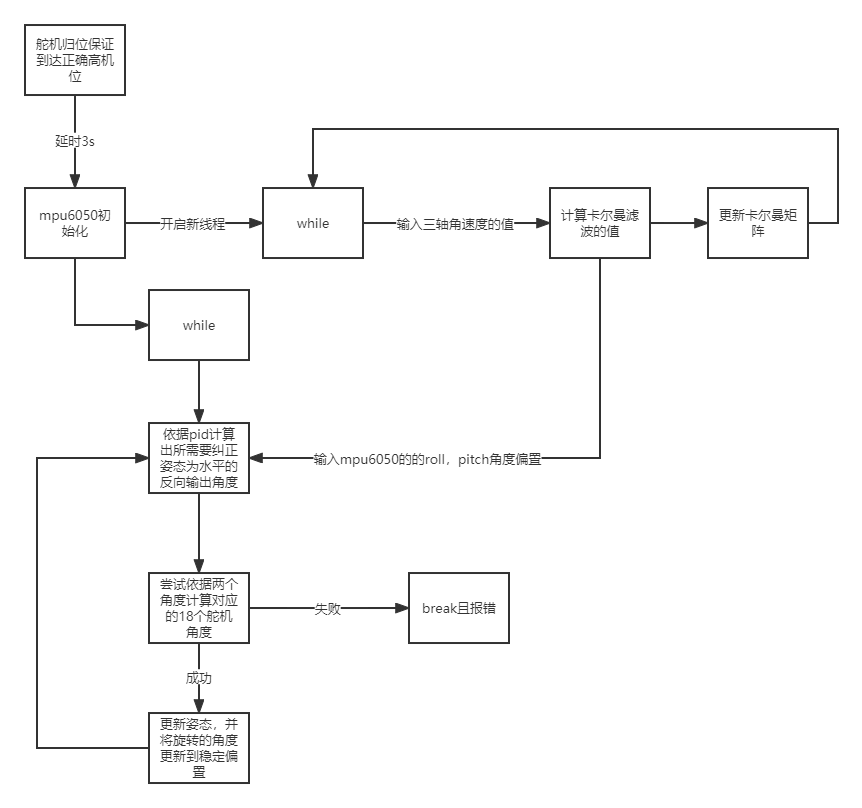
Step5:



这样我们就完成了一轮的循环。

需要注意的是，我们不希望其他多余的计算出现在mpu的计算上导致dt的增加而造成误差，所以我们直接选则为其单独开设一个线程。

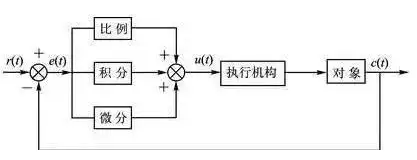
|  |
| --- |
| def measure(self):  angleThread = threading.Thread (target=self.measureAngles)  angleThread.start() |



Roll，pitch角度的pid控制：

首先我们需要明白pid的本质：

当被控对象的结构和参数不能完全掌握，或得不到精确的数学模型时，系统控制器的结构和参数必须依靠经验和现场调试来确定，这时使用PID控制可以使用系统自身对输出进行修正。



我们有两种方式去理解这个环节

第一种方法是将r（t）理解为离散值，我们有一个规定的目标输出r（t），但往往一次输出并不能达到目标值，可能会出现欠调和超调的情况。所以我们需要使用我们的输出值c（t）来计算误差去修正我们输出，而调节p，i，d的意义在于缩短或者稳定这个调节的过程。

第二种方法是在频域上理解，将r（t）理解为连续的输入函数，这个函数可以是高频的也可以是低频的，高频反应在六足机器人上就是角度的快速来回改变，而低频反应的就是平稳的改变，而我们的目的就是使e（t）为0，就是让这个系统的输出约等于r（t），且通过复杂化这个系统来增强这个系统的稳定性。（为了保证输出最简单的方法就是将传递函数定义为1的零阶系统，但零阶系统虽然可以实现高阶次的无差，但其最大缺点在于抗扰动能力太 差，另外由于其对于参数的变化太敏感，在实践上也不容易实现真正的零阶系统，所以在实际上不把调节系统校正为零阶。）当然在增加系统复杂性的同时，我们也不能让信号失真。

实现方式：

我们现在的情况是不能准确描述从翻转角到实际舵机角度的映射，这两者由于计算误差和舵机移动误差难以稳定。所以我们用pid优化控制。

首先我们使用第一种方式来理解这个问题：

不同于电机转动的pid，我们的目标输入是人定义的控制值，这里我们假设输入一直是0，而由于外界一直的扰动（平板的偏转），我们难以保持0，所以我们需要更改pid增加系统的响应效率，对于任意一个偏转值很快就能归0。

或者用第二种方法来理解，板子的晃动相当于是一个输入函数，我们要输出一个与原函数相似的函数。

|  |
| --- |
| #pidcontroll  err\_pitch = 0 - pitch  err\_roll = 0 - roll |

我们定义p项，i项，d项如下：

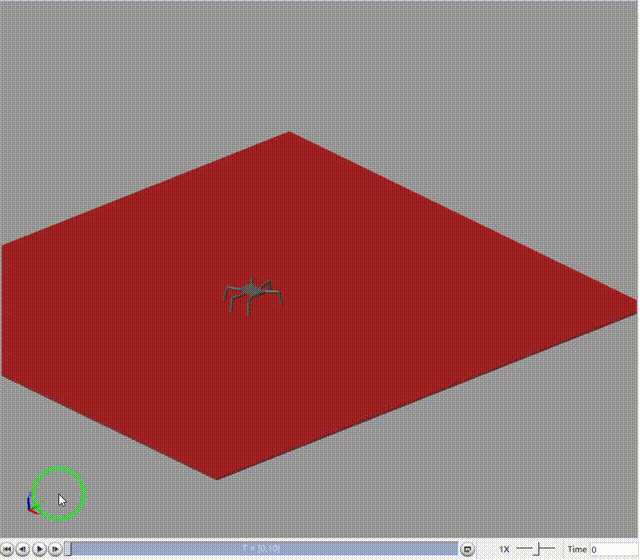
|  |
| --- |
| #累计项  if(err\_pitch > -SPE\_INTEGRAL\_START\_ERR and err\_pitch < SPE\_INTEGRAL\_START\_ERR):      err\_pitch\_integral += err\_pitch      if(err\_pitch\_integral > SPE\_INTEGRAL\_MAX\_VAL):          err\_pitch\_integral = SPE\_INTEGRAL\_MAX\_VAL      if(err\_pitch\_integral < -SPE\_INTEGRAL\_MAX\_VAL):          err\_pitch\_integral = -SPE\_INTEGRAL\_MAX\_VAL    if(err\_row > -SPE\_INTEGRAL\_START\_ERR and err\_row < SPE\_INTEGRAL\_START\_ERR):      err\_row\_integral += err\_roll      if(err\_row\_integral > SPE\_INTEGRAL\_MAX\_VAL):          err\_row\_integral = SPE\_INTEGRAL\_MAX\_VAL      if(err\_row\_integral < -SPE\_INTEGRAL\_MAX\_VAL):          err\_row\_integral = -SPE\_INTEGRAL\_MAX\_VAL |

|  |
| --- |
| #结果计算  output\_pitch = Kp \* err\_pitch + Ki\*err\_pitch\_integral + Kd\*(err\_pitch-err\_pitch\_last)  output\_row = Kp \* err\_row + Ki\*err\_row\_integral + Kd\*(err\_row-err\_row\_last)  #误差传递  err\_row\_last = err\_row  err\_pitch\_last = err\_pitch |

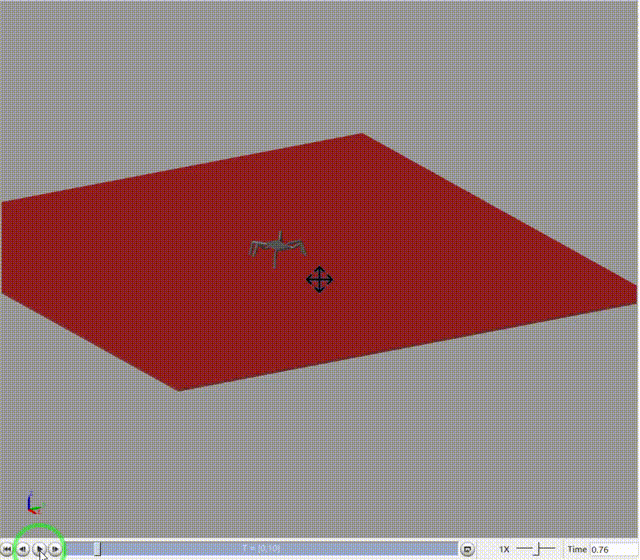
# 实验结果

## 仿真结果

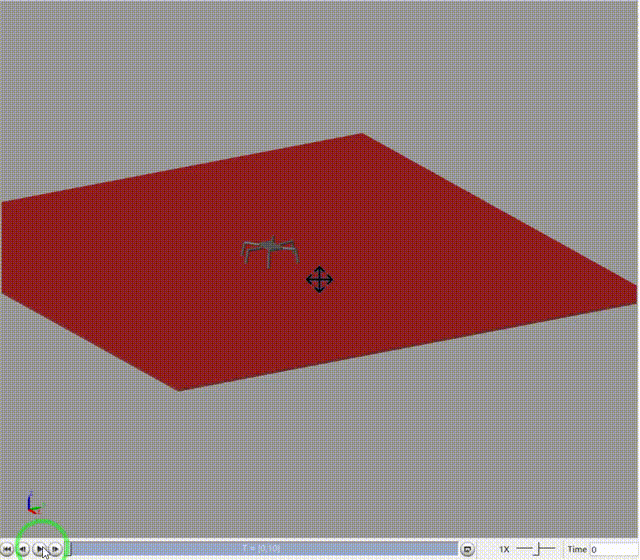
步距L=100，抬腿高度H=50，前进步态：



步距L=50，抬腿高度H=60，左移步态：



单腿单次旋转角θ=45°，抬腿高度H=50，顺时针旋转步态：



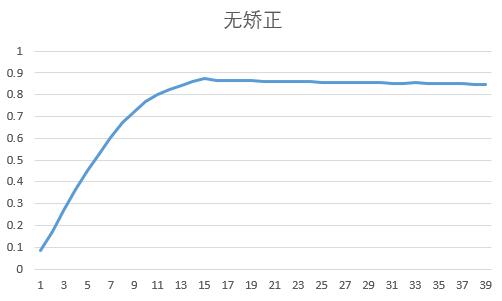
## 基本移动功能的实现

见附件视频演示。

## 自动角度适应/斜面自稳的实现和优化

自稳功能见附件视频演示。

我们根据实机性能绘制了阶跃响应曲线如下：



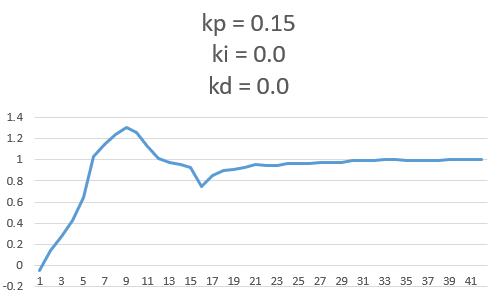
无PID调控的性能如图所示，响应时间较慢，且每次调节后，由于舵机的精度限制，机身并不能达到完全水平，存在一定稳态误差。

我们尝试增加PID环节来提高性能。

同时试图使用我们搭建的六足硬件平台可视化PID调节的作用。

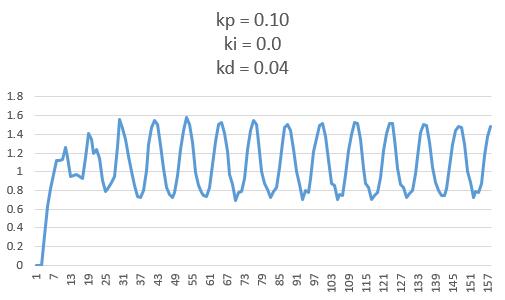
我们使用机身的实际反馈角度来做参考值。一开始的想法，是不断修正机器人的姿态，运用纯比例环节尝试将需调整的角度细分化，角度差距越大调节程度越大，并通过细分整个过程缩短调节时间，这也有利于减少IMU的滞空时间防止其数据漂移。

在我们引入PID环节后首先加入KP，结果如下图所示：



可见响应速度明显变快，伴随着Kp的增加，响应速度的继续增大，超调也越来越明显。

于是我们尝试引入Kd，结果如下图所示：



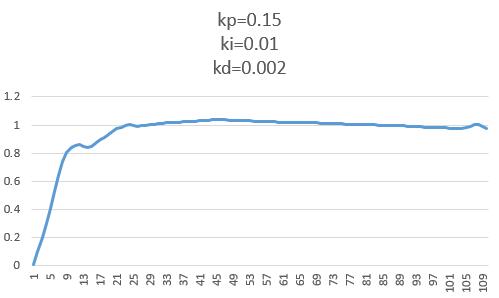
一开始的超调确实有所衰减，但由于我们的舵机精度，传感器精度较低，关机柔度较大，导致之后的系统出现了严重的抖动。

同时我们也尝试引入KI：



由于KI的取值过大且我们的机械结构和硬件平台散度过大，导致我们没有统一的稳态误差，所以系统发生了失稳震荡，且超调量也随之增大。

最后我们统一了所有参数，经过长时间调试得到的一组性能比较好的参数图如下所示：



参 考 文 献

[1] MANOONPONG P, PATANE L, XIONG X, et al. Insect-Inspired Robots: Bridging Biological and Artificial Systems[J]. Sensors (Basel), 2021, 21(22)

[2] SONNY\_XBY. 基于MATLAB步态算法仿真的六足仿生机器人[J/OL] 2019, https://www.cnblogs.com/Sonny-xby/p/11145366.html

[3] 陈瑞晓. 六足机器人步态规划与运动仿真[J]. 现代制造技术与装备, 2020, No.283(06)

[4] PEDLEY M. Tilt Sensing Using a Three-Axis Accelerometer[J]. 2013

[5] C+侦探DE裤子. MPU6050使用一阶互补和卡尔曼滤波算法平滑角度数据[J/OL] 2021, https://www.cnblogs.com/qq70565912/articles/15259892.html

[6] WELCH G, BISHOP G. An Introduction to the Kalman Filter[J]. 2006