上海海洋大學

毕业设计

（2018 届本科）

题 目：_____智能机器狗步态控制研究_____

_____

学 院：工程学院

专 业：电气工程及其自动化

班 级：电气一班

姓 名：姚逸凡

学 号：1822318

指导教师：匡兴红

2022 年 4 月

目 录

1 绪论 ..... 三
1.1 课题背景及研究目的 ..... 三
1.2 国内外研究现状 ..... 三
1.3 本论文研究主要内容 ..... 四
2 设计背景 ..... 四
2.1 平台介绍 ..... 四
2.2 设计思路和模型介绍 ..... 五
3 设计原理 ..... 六
3.1 运动的逆推导 ..... 六
3.2 足端摆线轨迹 ..... 十
3.3 基本步态设计 ..... 十二
3.4 步态程序的搭建 ..... 十五
4 仿真与程序 ..... 十六
4.1 Matlab 和 CoppeliaSim 联合仿真环境搭建 ..... 十六
4.2 Matlab 和 CoppeliaSim 联合仿真环境搭建设计过程中相关问题举例 ..... 二十
5 结论和展望 ..... 二十
5.1 人机交互界面的设计 ..... 二十
5.2 足尖轨迹不平衡 ..... 二十
5.3 其他功能性的拓展 ..... 二十一
谢辞 ..... 二十一
附录 A:主程序 ..... 二十二
附录 B:内嵌函数 ..... 二十七
附录 C: CoppeliaSim 内嵌脚本 ..... 二十九

智能机器狗步态控制研究

摘要：目前，市场上和研究中常用的于机器人的移动方式分为两大类：轮式机器人和步行机器人。轮式机器人采用轮胎或履带的方式进行平滑滚轮移动，因此，在面对地面情况复杂，环境崎岖，落差较大的地理背景下，轮式机器人的移动具有局限性。步行机器人采用仿生学的方式，通过模仿人类双足移动，或动物四足移动的方式，极大的提高了机器人移动的范围，能力和自由度。本次本科毕业论文的研究主题是基于步行机器人中的四足行走模式进行研究探讨，通过 Matlab 进行程序编码，Solidwork 进行机器狗的建模，以及机器人仿真平台 CoppeliaSim（V-REP）进行仿真模拟，模仿犬类的移动方式设计“智能机器狗”这一概念的机器设备，并通过仿真和程序模拟其四足行走方式，包括稳定性的分析和速度的调节，从而达到对机器狗步态控制的初步探讨和研究。

关键词：机器狗；步态；仿真；CoppeliaSimMatlabb

Research on gait control of intelligent robot dog

Abstacts： At present, the movement methods commonly used in robots in the market and research can be divided into two categories: wheeled robots and walking robots. Wheeled robots use tires or tracks to move smoothly with rollers. Therefore, under the geographical background of complex ground conditions like rugged environment and large drop, the movement of wheeled robots has limitations. The walking robot adopts the method of bionics, which greatly improves the range, ability and degree of freedom of the robot's movement by imitating the way of human bipedal movement or animal quadrupedal movement. The research topic of this undergraduate dissertation is based on the research and discussion of the four-legged walking mode in the walking robot. The program coding is carried out through Matlab, the modeling of the robot dog is carried out by Solidwork, and the robot simulation platform is CoppeliaSim (V-REP) imitating the dog. We will design the machine equipment of the concept of "intelligent robot dog", and simulate its four-legged walking mode through simulation and program, including stability analysis and speed adjustment, so as to achieve a preliminary discussion on the gait control of robot dog and research.

Key words： intelligent robot dog；gait control；simulation；CoppeliaSim；Matlab

1 绪论

1.1 课题背景及研究目的

研究四足机器人的首要目的是探索动物四足运动的原理，尤其是那些对于模拟关节的控制和平衡原理的探索。 这些原理可以帮助我们理解动物运动的过程并利用仿生学构建类似的四足机器人。四足机器人的一大特点是机动性。 他们能够通过导航系统的辅助进入现有车辆（轮式或者履带式）无法进入的复杂地形。轮式车辆在轨道和道路等预先铺好的路面上表现良好，但事实上世界上大部分地区都没有铺好的道路。 现有的轮式和履带式车辆只能到达地球上大约一半的土地，而足式机器人和动物的分布一样可以到达绝大部分的陆地区域。 因此，有必要研究和进化出这种“有腿”的车辆，以便到达动物可以到达的地方开展救援，土地勘测，运送物资等在复杂地形下需要完成的任务。如果这项研究成功，它将极大促进足式机器人的发展，使其现有车辆无法移动的松软、倾斜或受阻地形中高效、快速地移动。除了救援任务，这种运载工具将在工业、农业和军事应用中发挥巨大作用。是我认为这是我们未来的研究重点之一。

1.2 国内外研究现状

1.2.1 国外研究

最早具有四足机器人模型的史料记载是 Chebyshev 在 1870 年设计的行走机械，该机械将四足的肢体根部关节进行周期性运动，并将对角线上的两腿作为一组进行同步运动，这样可以进行简单的直立，行走和小跑。但是其局限性在于只能适应平坦的地面，对于较为崎岖的路面没有很好的运动稳定性。

1893 年，美国专利局出现了一款“骑马机”的机器，由 L.A.Rygg 设计。虽然其具有了较为自由的移动范围，但是该机器需要一名人员骑在上面进行人工运转，并不能算是严格意义上的自主行动机器人。

到了 20 世纪 40 年代，英美开始致力于研究四足机器人的在军事和航天领域上的应用，并得到了航天局和陆军官方的支持。60 年代，出现了步行车“Waling Truck”类似的设计，推动了四足机器人的进一步发展。

除了英美，日本大学也在进行类似的科技研究。日本东京工业大学的福田机器人研究实验室在 1976 年设计出名为 KUMO 的现代足行机器人，这被认为是世界上第一款具有自主行走能力的四足机器人，随后，又推出了更进一步的设计 PV-Ⅱ，具有上下楼梯的功能，被誉为四足机器人的里程碑。

现今，在科技飞速发展的 21 世纪，依靠着网络信息技术，生物仿真技术，计算机技术和人工智能，四足机器人已经得到了十分飞跃性的发展。其中最著名的是美国波士顿动力公司（Boston Dynamics）设计的“大黄”机器狗 Spot。除了基本的四足行走，跑步，跳跃功能外，Spot 甚至可以模仿真实的狗

的行为，例如伸懒腰，匍匐等十分具有仿生意义的动作，甚至必要时，它还可以为主人端茶送水，开门拉货。

1.2.2 国内研究

国内有关四足机器人的研究起于 20 世纪 80 年代，尽管起步较晚，基础薄弱，但是收到了国家科技研发的重点关照，被列入“863 计划”。一开始主要的研究小组分布在大学，据了解，上海交通大学，清华大学，吉林大学，北京大学等高校有关于四足机器人的研究。1991 年，上海交通大学以马培荪教授为主的研究人员研制出了关节式哺乳动物型四足机器人 JTUMM—III，整体有 12 个自由度，采用直流伺服电机进行驱动，利用它的足端压力传感器，通过位置和力的混合控制，实现了机器人的低速动态行走运动。

到了 21 世纪，国内一些科技企业也开始了机器狗的研究，华为公司推出了一款机器狗，由华为和宇树科技合作推出，具有前进、后退、左跳跃、右跳跃，后空翻等各种动作，非常灵活。它还可以智能识别物体，人等不同目标，会自动定位目标，能够动态跟踪多个目标，以及主动追随主人。看起来科技感十足。目前可以用应用于跳舞、智能识别、主动追踪、运载物品等场景。

1.3 本论文研究主要内容

本论文将研究四足机器人进行基本运动时的基本原理，对 Walk（行走）步态和 Trot（小跑）步态进行逆运动的分析，利用几何关系计算出四足机器人运动时的腿部位置与关节电机参数间的关系，然后研究足尖的摆线轨迹以计算腿部位置的路径，并进行循环。从而达到模拟运动的效果。本次研究是基于上述原理的电脑仿真模型，利用 matlab 进行编程控制和 CoppeliaSim（原 V-REP）进行仿真环境模拟，最终通过仿真的动画实现对四足机器人步态控制的研究

2 设计背景

2.1 平台介绍

目前市场上主流的机器人仿真平台有 Gazebo 和 CoppeliaSim（原 V-REP）。Gazebo 作为目前最流行的平台虽然在功能性上更胜一筹，但是其最佳运行系统平台是 Linux 中的 Ubuntu 系统，而不是我们常见的 Windows 或 MacOS 系统。因此在使用面上有些许限制。本人选择的 CoppeliaSim 平台完美兼容 Windows 系统，并且在 MacOS 和 Linux 上都能适用，可谓应用面广泛。另外 CoppeliaSim 支持多种编辑方法和编辑语言，方法包括嵌入式脚本、插件，附加组件、ROS 节点等，语言包括 C/C++、Python、Java、

Luau、Matlab、Octave 等。因此这个平台适用于任何背景的基础者使用，不用花费太多时间研究语言和界面，只需要通过程序自带的将编程平台和仿真平台联合的语言模块即可。除此之外，CoppeliaSim 自带四款物理引擎的模拟（ODE、Bullet、Vortex 和 Newton），因此可以让我们在仿真过程中更真实的发现问题和进行研究。鉴于以上优点，CoppeliaSim 已经广泛应用于快速算法开发、工厂自动化仿真、快速原型与验证、机器人相关教育、远程监控、安全复核等领域。

2.2 设计思路和模型介绍

“智能机器狗”又叫“四足机器人”，是通过控制一个主题上的四个机械臂完成类似于四足动物行为的仿生机器。其基本设计理念有很多种，有对称式：及交叉对称的脚（左前-右后，右前-左后）用同一关节控制，进行同轴运动。另一种是四足分别式：每个腿都由独立的关节控制。本人选择的是独立式。本人的设计是一条腿上由三个关节组成（三台电机），因此整个机器狗有 12 个关节（12 个电机）。由于本人选择的是电脑仿真进行模拟，因此本人通过 Solidwork 搭建了一个简易的机器狗模型，由 13 个部件十二个关节组成，这些关键可分为四部分：主躯干部件×1，肩部件×4，大腿部件×4，小腿部件×4。机器狗的每一条腿由一个肩部件，一个大腿部件和一个小腿部件组成，在每个部件的连接处放置一个电机作为关节控制各个部位的运动。因此 13 个部件由 12 个电机关节连接而成，形成我们的机器狗的拟态模型。而我们需要设计的就是通过往关节电机里写入程序，控制各个关节运行的速度，时间和方向，从而实现机器狗模拟行走，达到拟态步伐的目的。主躯干与肩部间的电机只能上下旋转，达到平衡和微调躯干高度的目的，肩部和大腿的关节只能前后旋转，达到控制腿部前进后退的目的。大腿和小腿之间的电机只能前后旋转，达到控制整体高度的目的。

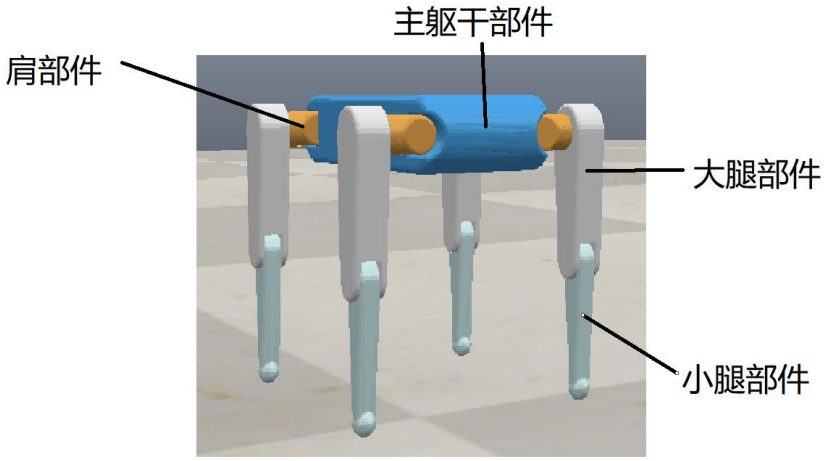

图 3-1 四足机器人模型各部件示意图

3 设计原理

3.1 运动的逆推导

3.1.1 视角分解

如图 3-1 所示，我们假设机器狗面朝的方向为 x 轴，垂直机器狗的方向为 z 轴，横向水平机器狗的方向为 y 轴。图 3-1 和 3-2 所展示的是机器狗正常站立时的单腿的模型分解。我们假设大腿长度为 hu，小腿长队为 hl，肩部长度为 h。足尖点的初始位置记为 P 点。

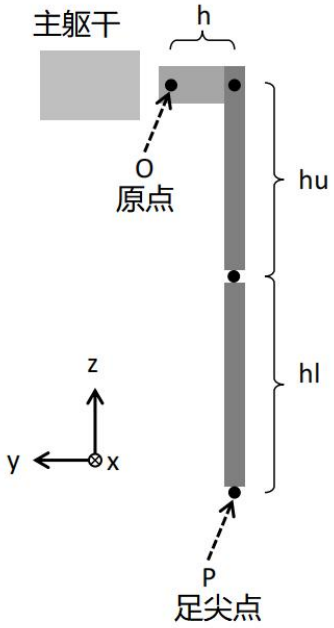

图 3-1 单腿逆解直立 x 轴示意图

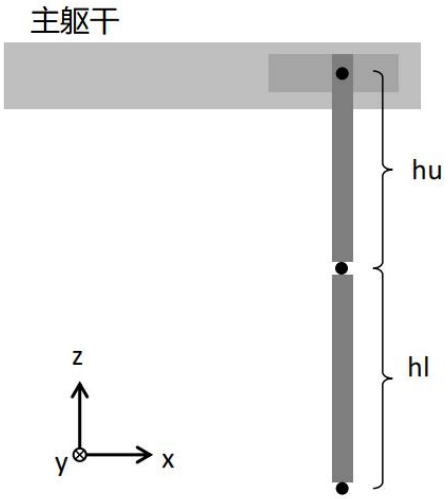

图 3-2 单腿逆解直立 y 轴示意图

然后我们控制各个关节上的电机，使各个部位随意扭曲一些角度，就会达到图 3-3 和 3-4 的效果。假设主躯干与肩部的关节旋转了 gamma 角，肩部和大腿间得关节旋转 alfa 角度，大腿和小腿间的关节旋转 beta 角度。此时足尖点的位置发生了变化，我们将前后视角下的位置记为 Pyz，水平视角下的位置记为 Pxz'。 我们的目标是在已知机器狗的相关参数：肩长 h，大腿长 hu 以及小腿长度 hl 的条件下，通过不同部件的相对位置运动时的几何关系，计算出此时的 alfa、beta 和 gamma 角，从而将这些数据输入到指定的电机里，让其进行角度的旋转。

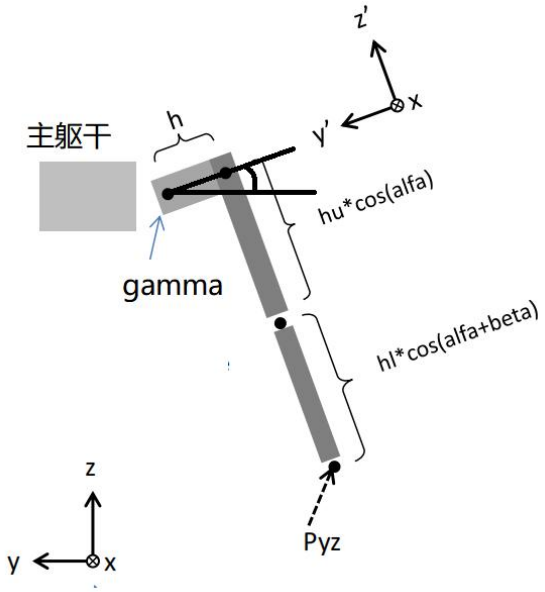

图 3-3 单腿逆解运动 x 轴示意图

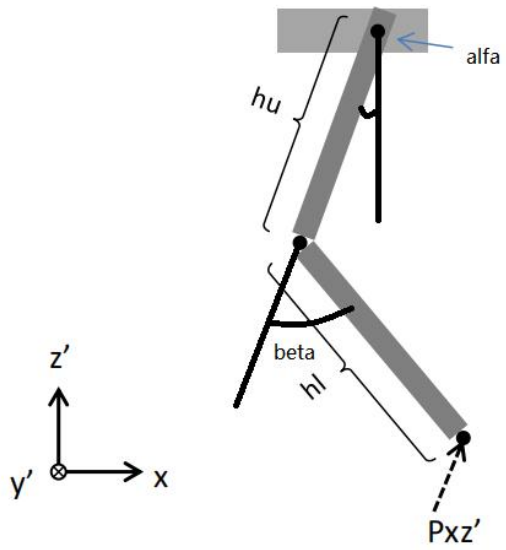

图 3-4 单腿逆解运动 y 轴示意图

3.1.2 gamma 角求解

设 dyz 为原点 O 到足尖点 Pyz 的前后视角下的长度，lyz 为此时大腿和小腿的相对长度。我们读取 CoppeliaSim 中的坐标系此时记录的 Pyz 的相对坐标得出 y 值和 z 值。因此根据几何关系勾股定理可以得出：

$$dyz = \sqrt{y^2 + z^2}$$

$$gamma\_yz = -\arctan\frac{y}{z}$$

此时：

$$lyz = \sqrt{dyz^2 - h^2}$$

$$gamma\_h\_offset = -\arctan\frac{h}{lyz}$$

因此可以得出：

$$gamma = gamma\_yz - gamma\_h\_offset$$

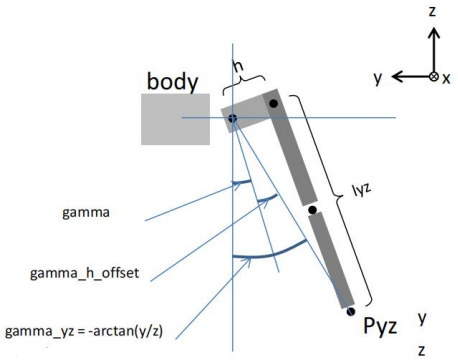

图 3-5 单腿逆解运动 x 轴角度示意图

3.1.3 beta 角求解

如图 3-6 所示，通过读取此时 CoppeliaSim 中的坐标系此时记录的 Pxz’的相对坐标得出足尖点距离主躯干的相对 z 值 lyz 和 x 值，通过勾股定理得出足尖点距离原点在水平视角下的相对距离 lxz'：

$$lxz' = \sqrt{x^2 + lyz^2}$$

之后我们可以构建出如图 3-7 所示的三角形，并且可以得出如下方程组：

$$(hu + n)^2 + m^2 = lxz'^2$$
$$n^2 + m^2 = hl^2$$

因为 hu 和 hl 为已知参数，我们可以计算得出 m 和 n 的值，从而得出 beta 的大小：

$$beta = -\arccos\frac{n}{hl}$$

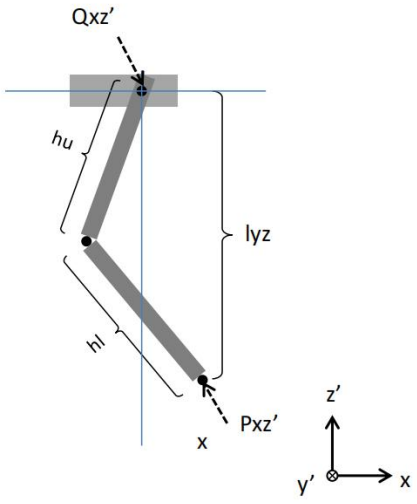

图 3-6 beta 角求解方法

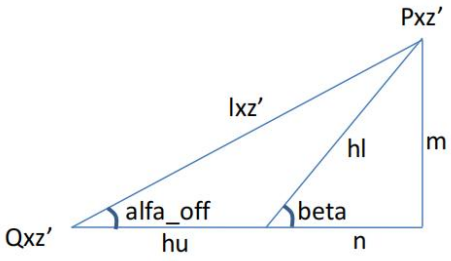

图 3-7 beta 角求解三角示意图

3.1.4 alfa 角求解

同样通过 CoppeliaSim 中的坐标系此时记录的坐标计算 alfa_xz’的大小：

$$alfa\_xz' = -\arctan\frac{x}{lyz}$$

根据图 3-7 的三角形关系得出 alfa_off 的大小：

$$alfa\_off = \arccos\frac{hu+n}{lxz'}$$

所以得出：

$$alfa = alfa\_off + alfa\_xz'$$

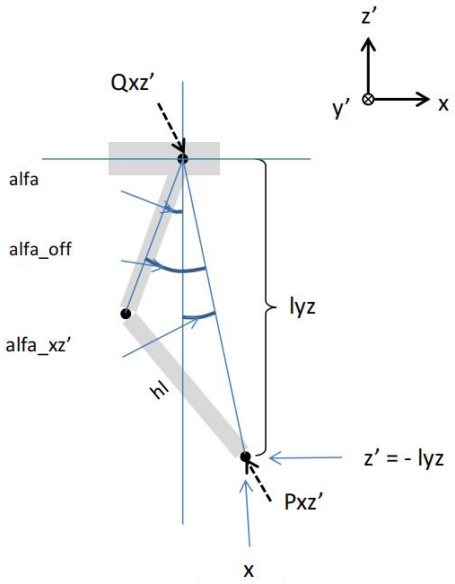

图 3-8 alfa 角求解示意图

3.1.5 Matlab 程序

我们因此可以在 matlab 中定义函数 xyz.m，以此来计算各个关节在不同位置下需要旋转的角度：

```
function  [gamma,alfa,beta]=xyz(x,y,z)
h=0.15; %模型参数
hu=0.35; %模型参数
hl=0.382; %模型参数
dyz=sqrt(y.^2+z.^2);
lyz=sqrt(dyz.^2-h.^2);
gamma_yz=-atan(y/z);
gamma_h_offset=-atan(h./lyz);
```

```
gamma=gamma_yz-gamma_h_offset;
%
lxzp=sqrt (lyz.^2+x.^2);
n=(lxzp.^2-hl.^2-hu.^2)/(2*hu);
beta=-acos(n/hl);
%
alfa_xzp=-atan(x/lyz);
alfa_off=acos((hu+n)/lxzp);
alfa=alfa_xzp+alfa_off;
%输出角度为弧度
end
```

3.2 足端摆线轨迹

3.2.1 摆线定义及基本方程

摆线又称圆滚线，是一个圆沿一条直线运动时圆上的一点所形成的轨迹。如图 3-9 所示，假设圆的半径为 a，在某个时刻 t 这一点与竖直方向的角度为θ，则摆线基本方程为：

$$x = a(\theta - \sin\theta)$$
$$y = a(1 - \cos\theta)$$

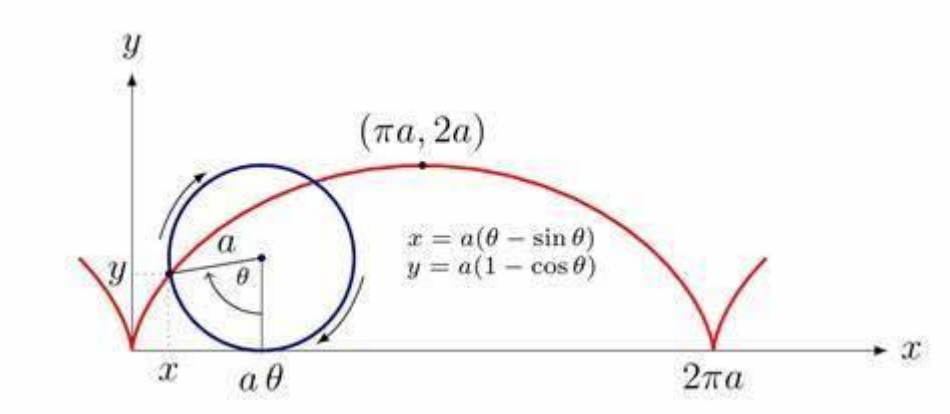

图 3-9 摆线原理及方程

3.2.2 足尖摆线方程

如果我们假设一个弧为一个周期 Ts，当我们已知足尖的初始位置和最终位置时，我们可以通过上述的摆线关系得出在摆动过程中足尖的相对位置，然后再把这个位置输入逆运动方程中得出电机旋转角

度，最终达成运动的目的。由此一来，只要我们设定四足机器狗在一个周期内每条腿的开始位置和结束位置，就可以达到四足机器人的拟态步伐控制。组件的相对位置可以通过如下公式计算：

$$\sigma = \frac{2\pi t}{T_s}, 0 < t < T_s$$

$$x_t = (x_f - x_s)\frac{\sigma - \sin\sigma}{2\pi} + x_s$$

$$z_t = h\frac{1-\cos\sigma}{2} + z_s$$

其中足尖初始位置坐标为（xs，ys，zs），最终位置坐标为（xf，yf，zf）。由于机器狗在前进过程中基本不存在左右横向平移的情况，因此我们可以设定一个恒定 y 值，并且：

$$y = y_s = y_f = y_t$$

所以足尖在摆动过程中的相对坐标为（xt，y，zt）。

3.2.3 Matlab 程序及实例

通过上述公式，我们可以建立如下摆线函数 baixian.m：

```
clc
clear
Ts=1; %周期
xs=-0.1; %起点 x 位置
xf=0.1;   %终点 x 位置
zs=-0.582; %z 起点位置
h=0.1; %抬腿高度
x=[];
z=[];
for t=0:0.01:Ts
    sigma=2*pi*t/(Ts);
    xep=(xf-xs)*((sigma-sin(sigma))/(2*pi))+xs;
    zep=h*(1-cos(sigma))/2+zs;
    x=[x,xep];
    z=[z,zep];
end
plot(x,z,'r','LineWidth',3)
```

如果我们假设初始位置坐标为（-0.1，y，-0.582），终点坐标位置为（0.1，y，-0.582），则该足尖的摆动轨迹如下图所示：

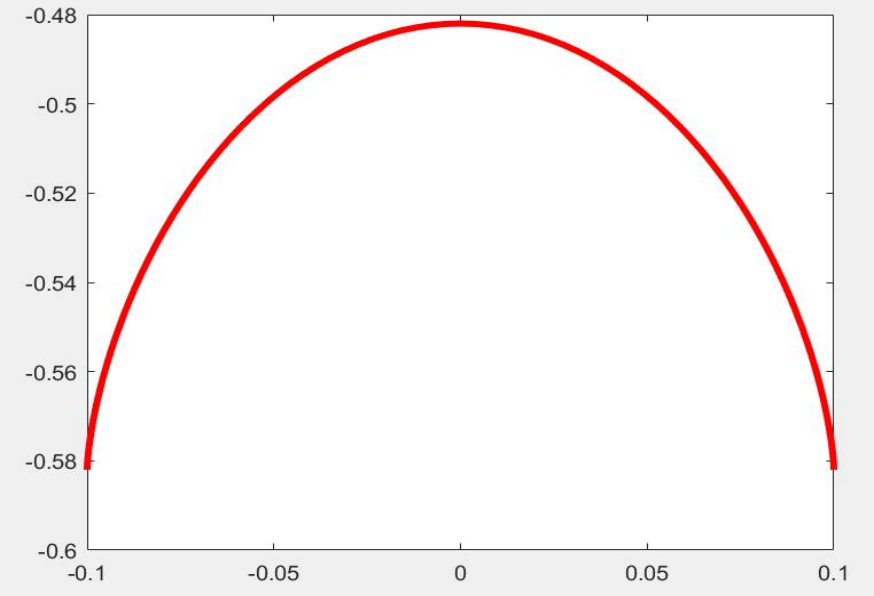

图 3-10 足尖摆线示意图

3.3 基本步态设计

3.3.1 基本步态分析

如果将四足生物运动的过程逐帧拍下来并进行分析，我们可以发现可以将单个腿的动作状态分为两大类，静止时的支撑状态和运动时的摆动状态。摆动状态可以用我们上述分析的逆运动分解通过分析各个足尖的相对位置计算得出电机的旋转角度，静止时的支撑状态可以看作足尖不动，原点发生移动（主躯干进行了运动），因此也可以看作足尖发生相对运动的形式，通过逆运动分解得出相应数据。如此一来，一个复杂的运动就变成了一个个单腿的运动循环，而步态的形成则是每个单腿运动开始的时间的不同所导致的，因此给我们一种视觉上每条腿运动不一样的状态。实际上他们的动作是一致的。

3.3.2 Walk 步态（行走步态）分析

如图 3-11 所示，如果我们将四条腿分别编号为 1，2，3，4。前腿编号为 1，4，后腿编号为 2，3。那么如果我们让机器狗向前进行 Walk 步态的运动，我们假设先让 1 号腿先动，那么在 1 号腿进行摆动的时候，2，3，4 号腿都处于支撑状态。在经过一段时间后，Walk 步态中的 3 号腿也开始摆动，而此时 1 号腿已经完成摆动运动，回到支撑状态。2，4 依旧处于支撑状态。再过一段时间，4 号发生摆动，，此时 1，2，3 都处于支撑状态。最后再过一段时间，2 号摆动，而 1，3，4 处于支撑状态。如此一个周期就是四足的一个完整的运动周期。Walk 步态可以看成是不断重复上述周期的一个运动过程，轮换顺序是 1→3→4→2→1 的“8”字型循环。

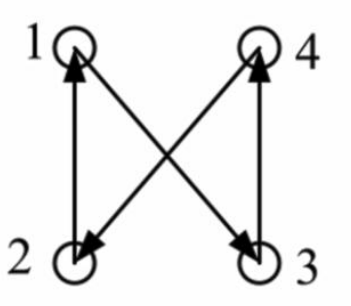

图 3-11 行走步态腿部顺序示意图

上述描述的一个周期可以分成四个阶段：1 号摆动，3 号摆动，4 号摆动，2 号摆动。在每个阶段都只有一条腿在进行摆动动作。我们只需要在每个阶段给相应的运动腿输入初始值，让它摆动到相应的相对坐标，通过函数自动计算出各个电机旋转的角度，从而达到运动的目的。我们假设一个运动周期为 T，那么我们可以让 1 号腿在 0T 的时候输入初始值，假设腿的反馈时间和运动时间为（1-ρ）T，我们令 1-ρ 远小于 0.25T，那么在第二条腿开始摆动时，第一条腿已经完成了摆动状态并回到支撑位置。由此一来，第二条腿的输入时间为 0.25T，第三条腿为 0.5T，第四条腿为 0.75T。具体运动时间和运动状态如下图 3-12 所示。所有腿在一个周期里的支撑时间从图中可以得出都是ρT。

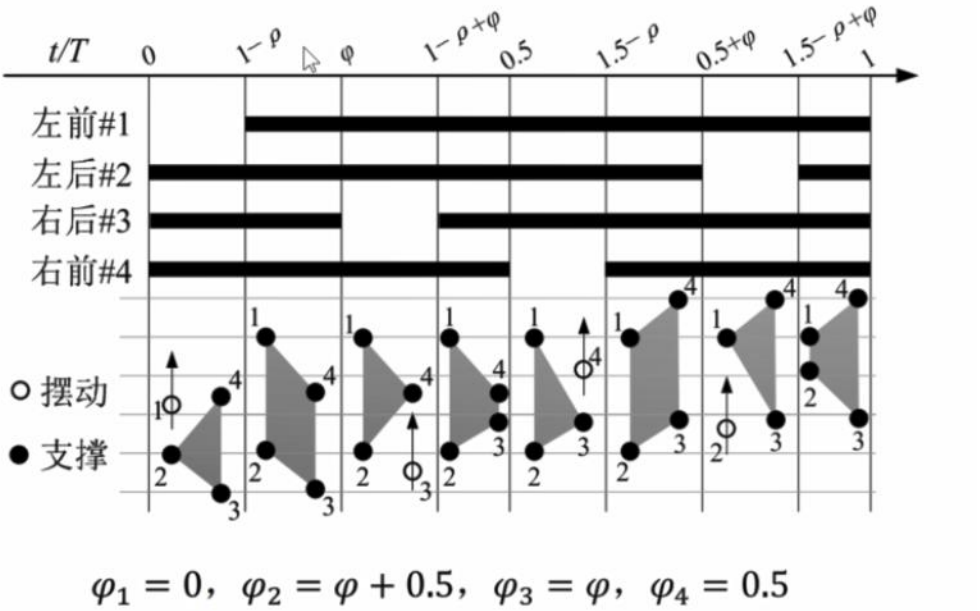

图 3-12 行走步态相位示意图

如果我们令ρ=0.75，此时为一种最理想的临界状态，即四条腿同时为支撑状态的时间为理想 0。这种状态是 Walk 步态下的最快运行。因此我们在设计的时候如果想要调速，可以通过改变ρ的大小来实现频率的变化。

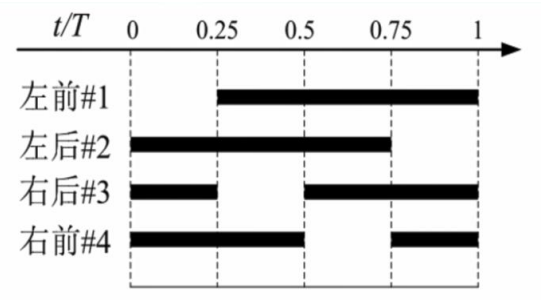

图 3-12 行走步态理想相位示意图

3.3.3 Trot 步态（小跑步态）分析

Trot 步态是机器狗运动中最常用的步态之一，因其具有速度快，耗能低的特点被广泛使用。Trot 步态与 Walk 步态的最大不同是 Trot 步态将对角的两条腿的运动状态绑定在了一起，使相对的两条腿进行同样的运动，即 1 号腿和 3 号腿运动一致，2 号腿和 4 号腿运动一致。这种状态下一个运动周期就只用分成两个阶段：1，3 摆动阶段和 2，4 摆动阶段。如图 3-13 所示，在 1，3 处于摆动阶段时，2，4 处于支撑状态。因此我们可以在 0T 的时候同时向 1，3 号腿输入初始值，在 0.5T 时同时向 2，4 号腿输入，以此达到。如此循环，形成 trot 步态的运动效果。

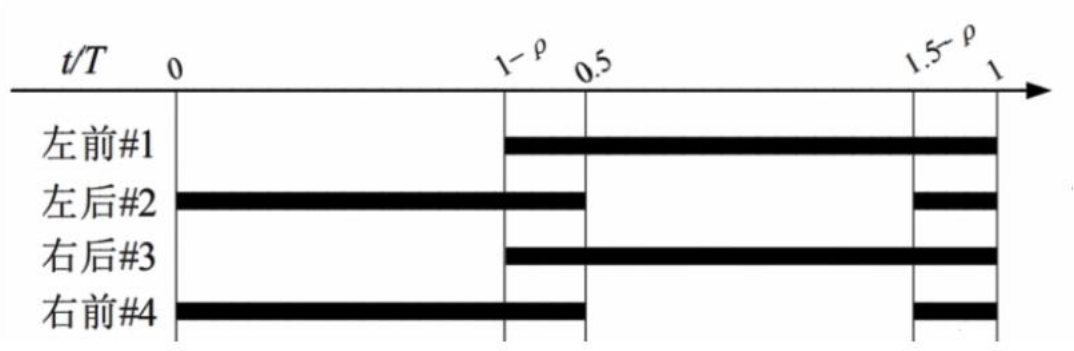

图 3-13 Trot 步态相位示意图

同样 trot 步态也有理想状态，如果我们令ρ=0.5，此时 trot 步态处于临界状态。四条腿同时接触地面的时间为 0。如图 3-14 所示。

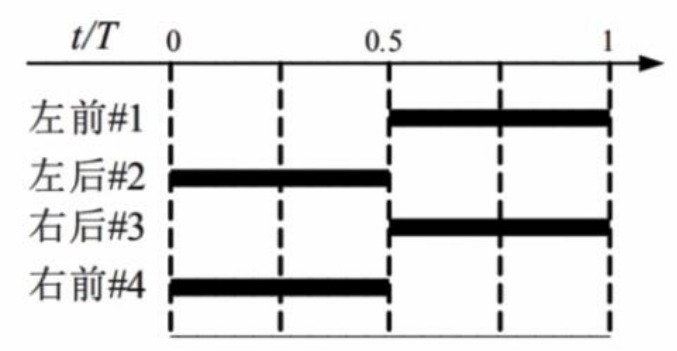

图 3-14 Trot 步态理想相位示意图

3.4 步态程序的搭建

从上述原理可以得出，只要我们给程序设定不同时间点开始相同的足尖摆线运动，就可以完成机器狗的四足拟态步伐模拟。具体程序如下：

3.4.1 Trot 步态 Matlab 方程

```
function [x,z]=gait_plan1(t,T)
Ts=T/2; %周期为 0.5s
xs=-0.1; %起点 x 位置
xf=0.1; %终点 x 位置
zs=-0.482; %z 起点位置
h=0.1; %抬腿高度
if(t<=Ts)
    sigma=2*pi*t/Ts;
    x=(xf-xs)*((sigma-sin(sigma))/(2*pi))+xs;
    z=h*(1-cos(sigma))/2+zs; %执行足尖摆线运动
else
    x=(-(xf-xs)/(T-Ts))*(t-Ts)+xf;
    z=-0.482;
end
end
```

方程中 T 的值可以在主程序中进行调节，用于调节机器狗步伐的频率和前进的速度。

3.4.2 Walk 步态 Matlab 方程

```
function [x,z]=gait_plan2(t,T)
Ts=T/4; %周期为 0.25s
xs=-0.1; %起点 x 位置
xf=0.1; %终点 x 位置
zs=-0.482; %z 起点位置
h=0.15; %抬腿高度
if(t<=Ts)
    sigma=2*pi*t/Ts;
    x=(xf-xs)*((sigma-sin(sigma))/(2*pi))+xs;
    z=h*(1-cos(sigma))/2+zs;
else
    x=(-(xf-xs)/(T-Ts))*(t-Ts)+xf;
    z=-0.482;
end
end
```

同样，方程中 T 的值可以在主程序中进行调节，用于调节机器狗步伐的频率和前进的速度。其余参数都可以在程序中进行调节。可以看出，其最大的不同时运动周期的不同，而具体是哪一条腿进行上述函数的运动需要在主程序中设定。

4 仿真与程序

4.1 Matlab 和 CoppeliaSim 联合仿真环境搭建

前面已经提到过，之所以选择 CoppeliaSim 作为我的仿真平台，是因为其与其他软件和语言强大的交互性。Matlab 是电气专业的笔者经常用到的软件，因此上手简单，非常适合笔者使用。那么现在的问题就是如何将两个平台进行联合仿真。我们可以通过阅读 CoppeliaSim 软件自带的用户手册获取相关的说明，具体操作步骤如下：

首先我们需要进入 CoppeliaSim 下载的源文件夹处，即打开 CoppeliaRobotics 命名的文件夹

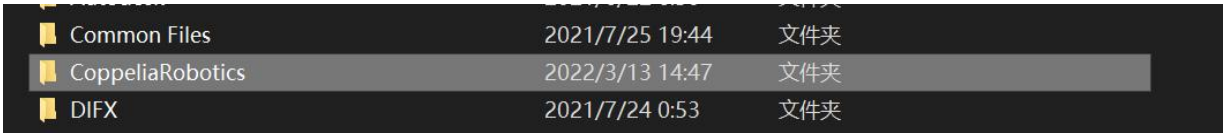

图 4-1 CoppeliaRobotics 文件夹

然后选择其中第一个文件夹

图 4-2 CoppeliaSim 文件夹

选择里面的 programming 文件

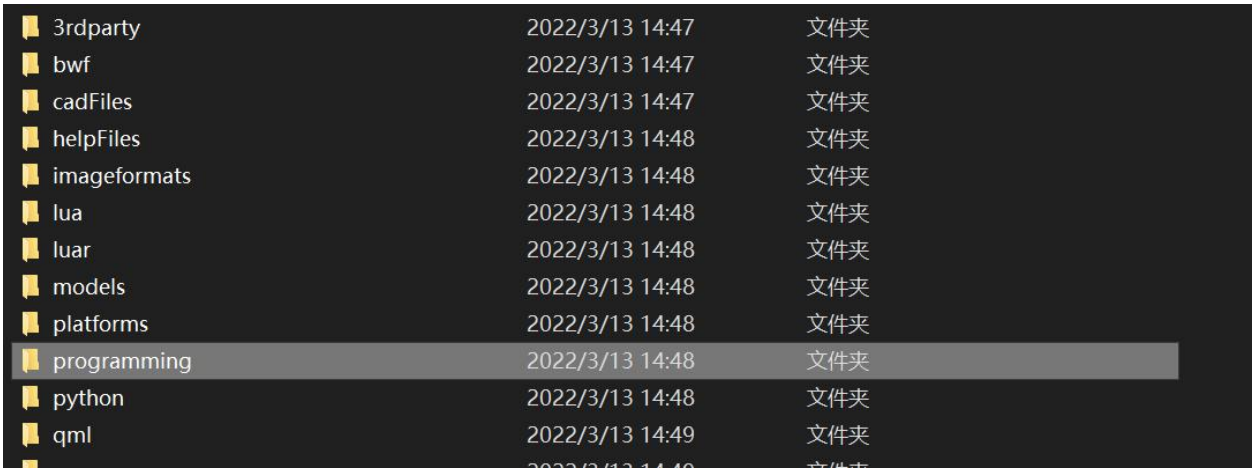

图 4-3 Programming 文件夹

选择 remoteApiBindings 文件

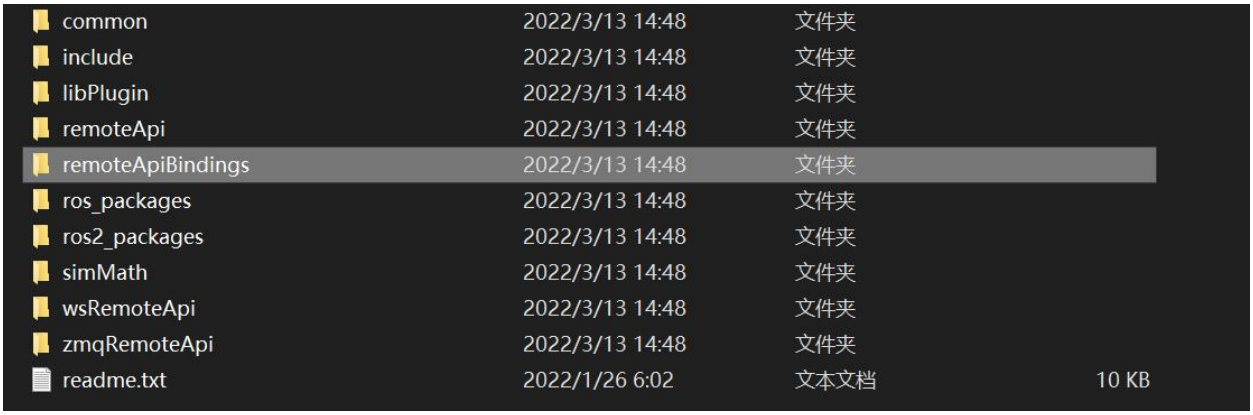

图 4-4 remoteApiBindings 文件夹

点开我们就可以看到这里面有很多我们熟悉的软件和语句，如果我们以后要选择其他的语言或者软件和 CoppeliaSim 进行联合仿真，就可以通过上述步骤找到相应的联合文件即可。这里我们选择 matlab

文件夹。

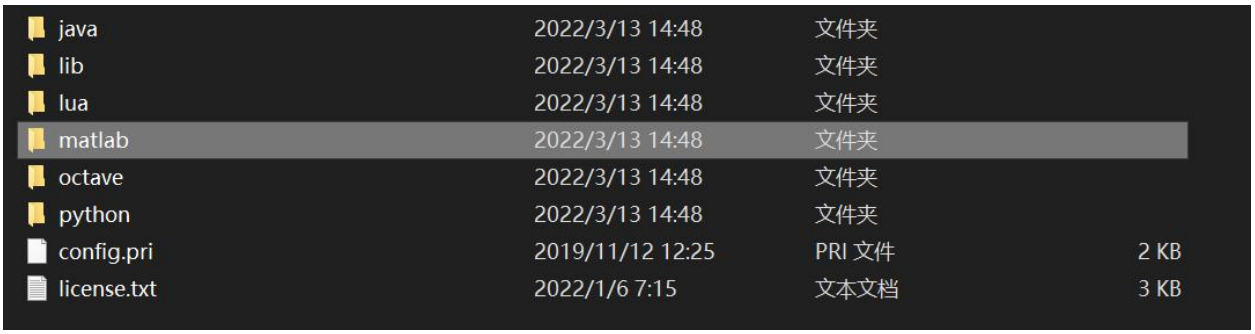

图 4-5 Matlab 文件夹

然后将里面的全部文件复制到我们 matlab 的程序文件夹中。

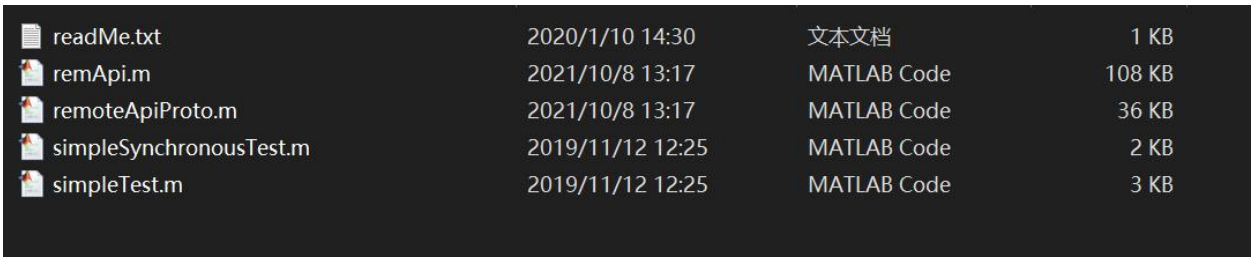

图 4-6 matlab 联合程序

接下来我们需要在 CoppeliaSim 中输入一个内嵌脚本，该脚本需要附着在我们导入的 Solidwork 模型的文件主题下，如下图所示：

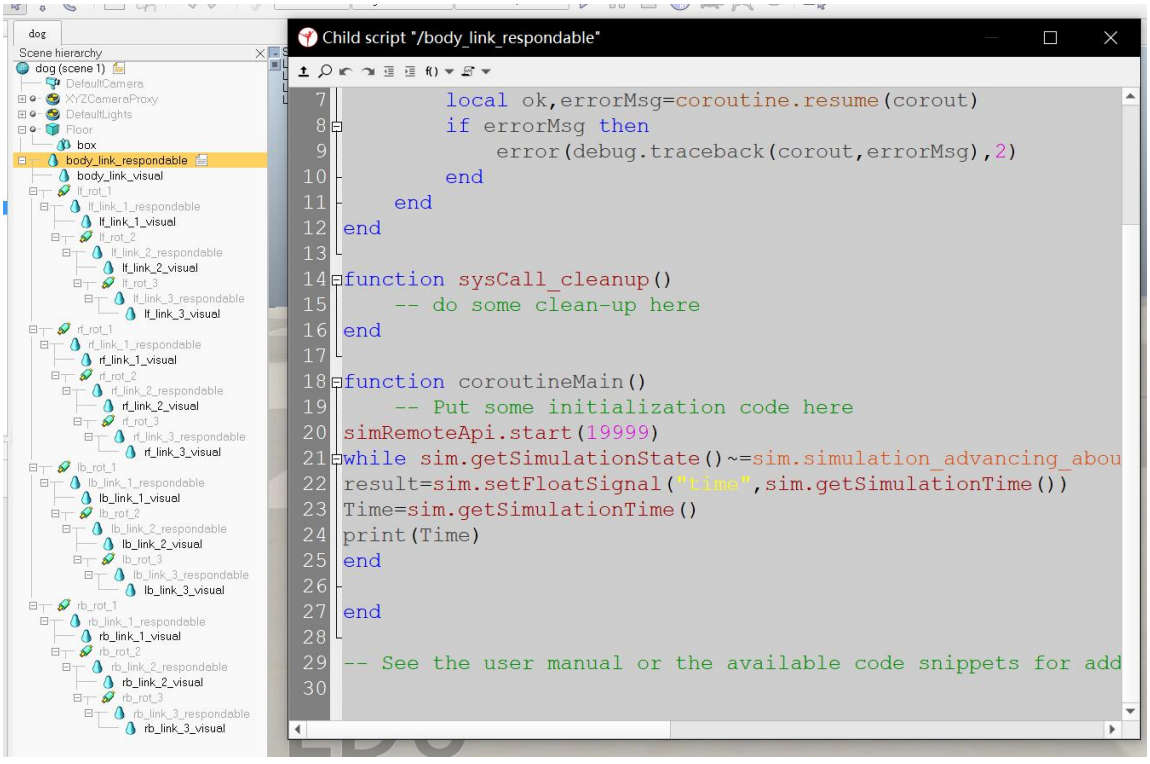

图 4-7 CoppeliaSim 脚本位置示意图

具体脚本内容如下：

```
function coroutineMain()
simRemoteApi.start(19999)
while sim.getSimulationState()~=sim.simulation_advancing_abouttostop do
result=sim.setFloatSignal("time",sim.getSimulationTime())
Time=sim.getSimulationTime()
print(Time)
end
```

这段脚本的语句是从 CoppeliaSim 自带的用户手册上得来的，具体的格式要求按照手册上的要求写即可。有了这个脚本文件后，我们还需要一个 Matlab 主程序的程序段进行连接声明和测试，具体代码如下：

```
%通讯初始化
clear
clc
disp('Program started');
vrep=remApi('remoteApi');
vrep.simxFinish(-1);
clientID=vrep.simxStart('127.0.0.1',19999,true,true,5000,5);%如果返回值为-1，则代表通讯不成功，返回值为 0 代表通讯成功
disp(clientID);
%%
if (clientID>-1)
    disp('Connected to remote API server');
    vrep.simxStartSimulation(clientID,vrep.simx_opmode_oneshot);%开始仿真
……%主程序部分
  vrep.simStopSimulation(clientID,vrep.simx_opmode_blocking);%仿真停止
    vrep.simxDinishi(clientID);
else
    disp('Failed connecting to remote API server');
end
```

有了上述这段操作和代码，我们的 matlab 程序就可以和 CoppeliaSim 中导入的 Solidwork 模型进行联合仿真和控制了。

4.2 Matlab 和 CoppeliaSim 联合仿真环境搭建设计过程中相关问题举例

4.2.1 Matlab 无法与 CoppeliaSim 进行联合仿真怎么办？

即 Matlab 里写的程序仿真平台读不出来。这是一个大问题，因为代码不认识机器狗根本不听话，后来通过查找资料发现了问题所在。CoppeliaSim 自身文件中是有 Matlab 联合文件的，只需将这些文件复制到 matlab 目录下，通过 matlab 打开，再在仿真平台中写一段读取命令的脚本就行。但是官方给的联合文件缺了一个 dll 文件，导致其与 matlab 无法顺利联合仿真，因此在从其他地方找到这个 dll 文件和放入联合文件里就可以顺利反馈代码了。

4.2.2 关节无法按照期望运动到目标角度（点）怎么办？

在仿真过程中经常会有十分搞笑的现象，像机器狗软绵绵的站不起来啊，机器狗腿和腿打架然后起飞啊之类的。这次的问题是抬腿后它不会停在目标角度，而是会来回抽搐。这是因为关节（电机）设置的最大力矩不够，导致其无法支撑自身的重力达到平衡状态。其他两个小问题很简单，一个是没有关节力矩，自然无法设置，另一个是碰撞体积设置的不对，调下参数就行。

5 结论和展望

5.1 人机交互界面的设计

目前程序设计的功能一个最大的问题是无法做到快速的人机交互界面的应用。目前如果我们想控制四足机器人从 Walk 步态还原到站立或者初始状态时，无法通过外部程序和人机交互界面进行调换，也无法作答从 Walk 步态切换到 Trot 步态的功能。这一问题的根本原因是写程序时选用了 Switch 语句，该语句对后面的输入值进行判断时是通过 case 1，2 等数字进行识别的。因此目前如果需要切换步态方式需要到程序中修改选择的 case 号。目前笔者仍在研究 Matlab 的 GUI 功能，该功能类似于 App 应用的开发，可以通过设计人机交互界面控制机器狗的行为，一键发出命令，十分简单清楚。但是目前仍在研究当中。

5.2 足尖轨迹不平衡

考虑到真实的物理引擎的效果下，只要是有质量的物体在以一定速度落向地面时，都会产生动能。

这种动能会反馈给机器本身造成四足机器人腿部落地时无法达到完美的摆线模型的状态，会出现类似的抖动或者偏移，如下图所示

图 5-1 足尖摆线波动示意图

如何减轻这种滑行的效果是目前时很多公司和研究项目研究的主题之一，笔者暂时也在研究当中。

5.3 其他功能性的拓展

笔者目前仍在研究其他步态的姿态解算和逆运动过程。类似的比如 Bound（跳跃）步态和快速跑步态，该种步态由于会产生一段四足都不处于支撑状态的时间，使得机器狗在空中完全悬空，那么此时足部向下产生的加速度和相应的动能都会对机器狗本身产生较大的影响。目前仍是研究的主题之一。

谢辞

特别鸣谢上海海洋大学相关部门和指导老师在疫情影响下的特殊时期依旧提供帮助和支持，感谢古月居机器人研究论坛对本次设计提供思路的指导，感写指导老师匡兴红老师的帮助以及霍海波老师在其他沟通方面对我提供的支持。

参考文献：

[1] 林德龙. 四足机器人行走步态及控制研究[D]. 西南科技大学, 2011.

[2] Santos P G D， Garcia E， Estremera J . Quadrupedal locomotion. An introduction to the control of four-legged robots. Springer-Verlag New York, Inc. 2006.

[3] 阿芬. 国内外四足机器人的发展历史、发展趋势 [EB/OL]. 2020[1 月 6 日]. https://www.robot-china.com/news/202001/06/60371.html.

[4] 迷你数智. 华为推出有趣的机器狗，和波士顿的机器狗有什么不同？ [EB/OL]. 2020[8 月 23 日]. https://baijiahao.baidu.com/s?id=1675823105794737325&wfr=spider&for=pc.

[5] A. Shkolnik and R. Tedrake, "Inverse Kinematics for a Point-Foot Quadruped Robot with Dynamic Redundancy Resolution," Proceedings 2007 IEEE International Conference on Robotics and Automation, 2007, pp. 4331-4336, doi: 10.1109/ROBOT.2007.364146.

[6] Raibert M. Legged Robots That Balance[M]. MIT Press, 1986.

附录 A:主程序

```
%通讯初始化
clear
clc
disp('Program started');
vrep=remApi('remoteApi');
vrep.simxFinish(-1);
clientID=vrep.simxStart('127.0.0.1',19999,true,true,5000,5);%如果返回值为-1，则代表通讯不成功，返回值为 0 代表通讯成功
disp(clientID);
%%
if (clientID>-1)
    disp('Connected to remote API server');
    vrep.simxStartSimulation(clientID,vrep.simx_opmode_oneshot);%开始仿真
    %声明节点，关节初始化，12 个关节
    [rec ,rb_rot_3]=vrep.simxGetObjectHandle (clientID,'rb_rot_3',vrep.simx_opmode_blocking);
    [rec ,rf_rot_3]=vrep.simxGetObjectHandle (clientID,'rf_rot_3',vrep.simx_opmode_blocking);
    [rec ,rb_rot_2]=vrep.simxGetObjectHandle (clientID,'rb_rot_2',vrep.simx_opmode_blocking);
    [rec ,rf_rot_2]=vrep.simxGetObjectHandle (clientID,'rf_rot_2',vrep.simx_opmode_blocking);
    [rec ,rb_rot_1]=vrep.simxGetObjectHandle (clientID,'rb_rot_1',vrep.simx_opmode_blocking);
    [rec ,rf_rot_1]=vrep.simxGetObjectHandle (clientID,'rf_rot_1',vrep.simx_opmode_blocking);
    [rec ,lb_rot_3]=vrep.simxGetObjectHandle (clientID,'lb_rot_3',vrep.simx_opmode_blocking);
    [rec ,lf_rot_3]=vrep.simxGetObjectHandle (clientID,'lf_rot_3',vrep.simx_opmode_blocking);
    [rec ,lb_rot_2]=vrep.simxGetObjectHandle (clientID,'lb_rot_2',vrep.simx_opmode_blocking);
    [rec ,lf_rot_2]=vrep.simxGetObjectHandle (clientID,'lf_rot_2',vrep.simx_opmode_blocking);
    [rec ,lb_rot_1]=vrep.simxGetObjectHandle (clientID,'lb_rot_1',vrep.simx_opmode_blocking);
    [rec ,lf_rot_1]=vrep.simxGetObjectHandle (clientID,'lf_rot_1',vrep.simx_opmode_blocking);
    %12 个电机力矩参数
    rb_rot_1_force=500; rb_rot_2_force=500; rb_rot_3_force=500;   %右后腿
```

```
rf_rot_1_force=500; rf_rot_2_force=500; rf_rot_3_force=500;   %右前腿
lb_rot_1_force=500; lb_rot_2_force=500; lb_rot_3_force=500;   %左后腿
lf_rot_1_force=500; lf_rot_2_force=500; lf_rot_3_force=500;   %左前腿
%12 个电机角度参数
rb_rot_1_pos=0; rb_rot_2_pos=0; rb_rot_3_pos=0;   %右后腿
rf_rot_1_pos=0; rf_rot_2_pos=0; rf_rot_3_pos=0;   %右前腿
lb_rot_1_pos=0; lb_rot_2_pos=0; lb_rot_3_pos=0;   %左后腿
lf_rot_1_pos=0; lf_rot_2_pos=0; lf_rot_3_pos=0;   %左前腿
%设置电机力矩
rec=vrep.simxSetJointForce(clientID, rb_rot_3,rb_rot_3_force,vrep.simx_opmode_blocking);
rec=vrep.simxSetJointForce(clientID, rf_rot_3,rf_rot_3_force,vrep.simx_opmode_blocking);
rec=vrep.simxSetJointForce(clientID, rb_rot_2,rb_rot_2_force,vrep.simx_opmode_blocking);
rec=vrep.simxSetJointForce(clientID, rf_rot_2,rf_rot_2_force,vrep.simx_opmode_blocking);
rec=vrep.simxSetJointForce(clientID, rb_rot_1,rb_rot_1_force,vrep.simx_opmode_blocking);
rec=vrep.simxSetJointForce(clientID, rf_rot_1,rf_rot_1_force,vrep.simx_opmode_blocking);
rec=vrep.simxSetJointForce(clientID, lb_rot_3,lb_rot_3_force,vrep.simx_opmode_blocking);
rec=vrep.simxSetJointForce(clientID, lf_rot_3,lf_rot_3_force,vrep.simx_opmode_blocking);
rec=vrep.simxSetJointForce(clientID, lb_rot_2,lb_rot_2_force,vrep.simx_opmode_blocking);
rec=vrep.simxSetJointForce(clientID, lf_rot_2,lf_rot_2_force,vrep.simx_opmode_blocking);
rec=vrep.simxSetJointForce(clientID, lb_rot_1,lb_rot_1_force,vrep.simx_opmode_blocking);
rec=vrep.simxSetJointForce(clientID, lf_rot_1,lf_rot_1_force,vrep.simx_opmode_blocking);


%row=0;     pitch=0; yaw=0;
%pos_x=0; pos_y=0.2; pos_z=-0.2;
pause (1); %延时 1s
t=clock; %获取 matlab 系统当前时间
startTime=t(5)*60+t(6); %当前时间[年 月 日 时 分 秒]
currentTime=0; %当前时间
gait_state=1;   %步态标志位
 %[rb_x,rb_y,rb_z,rf_x,rf_y,rf_z,lb_x,lb_y,lb_z,lf_x,lf_y,lf_z]=pose_control(row,pitch,yaw,pos_x,pos_y,p
```

```
os_z);
        % [lb_rot_1_pos, lb_rot_2_pos, lb_rot_3_pos]=xyz(lb_x,lb_y,lb_z);
        % [rb_rot_1_pos, rb_rot_2_pos, rb_rot_3_pos]=xyz(rb_x,rb_y,rb_z);
      %    [lf_rot_1_pos, lf_rot_2_pos, lf_rot_3_pos]=xyz(lf_x,lf_y,lf_z);
        % [rf_rot_1_pos, rf_rot_2_pos, rf_rot_3_pos]=xyz(rf_x,rf_y,rf_z);
    %%
    while (currentTime < 100)
        t=clock;
        currentTime=t(5)*60+t(6)-startTime;
        if (currentTime <5)
            if(gait_state==2)% walk 步态
                lb_x=-0.1;     rb_x=-0.1;     lf_x=0.1;       rf_x=0.1;
                lb_z=-0.482; rf_z=-0.482; lf_z=-0.482; rb_z=-0.482;
            end
            if(gait_state==1)% trot 小跑步态
                lb_x=-0.1;     rb_x=0.1;     lf_x=0.1;       rf_x=-0.1;
                lb_z=-0.482; rf_z=-0.482; lf_z=-0.482; rb_z=-0.482;
            end
            [rec,vrep_time]=vrep.simxGetFloatSignal(clientID,'time',vrep.simx_opmode_oneshot);
        else
            [rec,vrep_realtime]=vrep.simxGetFloatSignal(clientID,'time',vrep.simx_opmode_oneshot);

            switch gait_state
                case 2 % walk 步态
                    T=1;
                    time1=vrep_realtime-vrep_time;
                    time2=vrep_realtime-vrep_time+0.25;
                    time3=vrep_realtime-vrep_time+0.5;
                    time4=vrep_realtime-vrep_time+0.75;
                    T1=mod(time1,T); T2=mod(time2,T);
```

```
                T3=mod(time3,T); T4=mod(time4,T);
                [lb_x,lb_z]=gait_plan2(T1,T);
                [rf_x,rf_z]=gait_plan2(T2,T);
                [rb_x,rb_z]=gait_plan2(T3,T);
                [lf_x,lf_z]=gait_plan2(T4,T);
            case 1 % trot 小跑步态
                T=0.4;
                time1=vrep_realtime-vrep_time;
                time2=vrep_realtime-vrep_time+0.2;
                T1=mod(time1,T); T2=mod(time2,T);
                [lb_x,lb_z]=gait_plan1(T1,T);
                [rf_x,rf_z]=gait_plan1(T1,T);
                [rb_x,rb_z]=gait_plan1(T2,T);
                [lf_x,lf_z]=gait_plan1(T2,T);
        end
    end
    %单腿逆运动学， Y 值设定为默认值
[lb_rot_1_pos, lb_rot_2_pos, lb_rot_3_pos]=xyz(lb_x,-0.15,lb_z);
[rb_rot_1_pos, rb_rot_2_pos, rb_rot_3_pos]=xyz(rb_x,-0.15,rb_z);
[lf_rot_1_pos, lf_rot_2_pos, lf_rot_3_pos]=xyz(lf_x,-0.15,lf_z);
[rf_rot_1_pos, rf_rot_2_pos, rf_rot_3_pos]=xyz(rf_x,-0.15,rf_z);
%电机控制函数
    rec=vrep.simxSetJointTargetPosition(clientID,lb_rot_1,-lb_rot_1_pos,vrep.simx_opmode_oneshot);
    rec=vrep.simxSetJointTargetPosition(clientID,lb_rot_2,lb_rot_2_pos,vrep.simx_opmode_oneshot);
    rec=vrep.simxSetJointTargetPosition(clientID,lb_rot_3,lb_rot_3_pos,vrep.simx_opmode_oneshot);
    rec=vrep.simxSetJointTargetPosition(clientID,rb_rot_1,-rb_rot_1_pos,vrep.simx_opmode_oneshot);
    rec=vrep.simxSetJointTargetPosition(clientID,rb_rot_2,rb_rot_2_pos,vrep.simx_opmode_oneshot);
    rec=vrep.simxSetJointTargetPosition(clientID,rb_rot_3,rb_rot_3_pos,vrep.simx_opmode_oneshot);
    rec=vrep.simxSetJointTargetPosition(clientID,lf_rot_1,lf_rot_1_pos,vrep.simx_opmode_oneshot);
    rec=vrep.simxSetJointTargetPosition(clientID,lf_rot_2,lf_rot_2_pos,vrep.simx_opmode_oneshot);
```

```
        rec=vrep.simxSetJointTargetPosition(clientID,lf_rot_3,lf_rot_3_pos,vrep.simx_opmode_oneshot);
        rec=vrep.simxSetJointTargetPosition(clientID,rf_rot_1,rf_rot_1_pos,vrep.simx_opmode_oneshot);
        rec=vrep.simxSetJointTargetPosition(clientID,rf_rot_2,rf_rot_2_pos,vrep.simx_opmode_oneshot);
        rec=vrep.simxSetJointTargetPosition(clientID,rf_rot_3,rf_rot_3_pos,vrep.simx_opmode_oneshot);
    end
    vrep.simStopSimulation(clientID,vrep.simx_opmode_blocking);%仿真停止
    vrep.simxDinishi(clientID);
else
    disp('Failed connecting to remote API server');
end
```

附录 B:内嵌函数

1. 逆运动解

```
function [gamma,alfa,beta]=xyz(x,y,z)
h=0.15;
hu=0.35;
hl=0.382;
dyz=sqrt(y.^2+z.^2);
lyz=sqrt(dyz.^2-h.^2);
gamma_yz=-atan(y/z);
gamma_h_offset=-atan(h./lyz);
gamma=gamma_yz-gamma_h_offset;
%
lxzp=sqrt (lyz.^2+x.^2);
n=(lxzp.^2-hl.^2-hu.^2)/(2*hu);
beta=-acos(n/hl);
%
alfa_xzp=-atan(x/lyz);
alfa_off=acos((hu+n)/lxzp);
alfa=alfa_xzp+alfa_off;
%输出角度为弧度
end
```

2. Walk 步态

```
function [x,z]=gait_plan2(t,T)
Ts=T/4; %周期为 0.25s
xs=-0.1; %起点 x 位置
xf=0.1; %终点 x 位置
zs=-0.482; %z 起点位置
h=0.15; %抬腿高度
if(t<=Ts)
    sigma=2*pi*t/Ts;
    x=(xf-xs)*((sigma-sin(sigma))/(2*pi))+xs;
    z=h*(1-cos(sigma))/2+zs;
else
    x=(-(xf-xs)/(T-Ts))*(t-Ts)+xf;
    z=-0.482;
end
end
```

3. Trot 步态

```
function [x,z]=gait_plan1(t,T)
Ts=T/2; %周期为 0.5s
xs=-0.1; %起点 x 位置
xf=0.1; %终点 x 位置
zs=-0.482; %z 起点位置
h=0.1; %抬腿高度
if(t<=Ts)
    sigma=2*pi*t/Ts;
    x=(xf-xs)*((sigma-sin(sigma))/(2*pi))+xs;
    z=h*(1-cos(sigma))/2+zs;
else
    x=(-(xf-xs)/(T-Ts))*(t-Ts)+xf;
    z=-0.482;
end
end
```

附录 C: CoppeliaSim 内嵌脚本

```
function sysCall_init()
    corout=coroutine.create(coroutineMain)
end
function sysCall_actuation()
    if coroutine.status(corout)~='dead' then
        local ok,errorMsg=coroutine.resume(corout)
        if errorMsg then
            error(debug.traceback(corout,errorMsg),2)
        end
    end
end
function sysCall_cleanup()
end
function coroutineMain()
simRemoteApi.start(19999)
while sim.getSimulationState()~=sim.simulation_advancing_abouttostop do
result=sim.setFloatSignal("time",sim.getSimulationTime())
Time=sim.getSimulationTime()
print(Time)
end
end
```