

机器鼠仿真实验报告

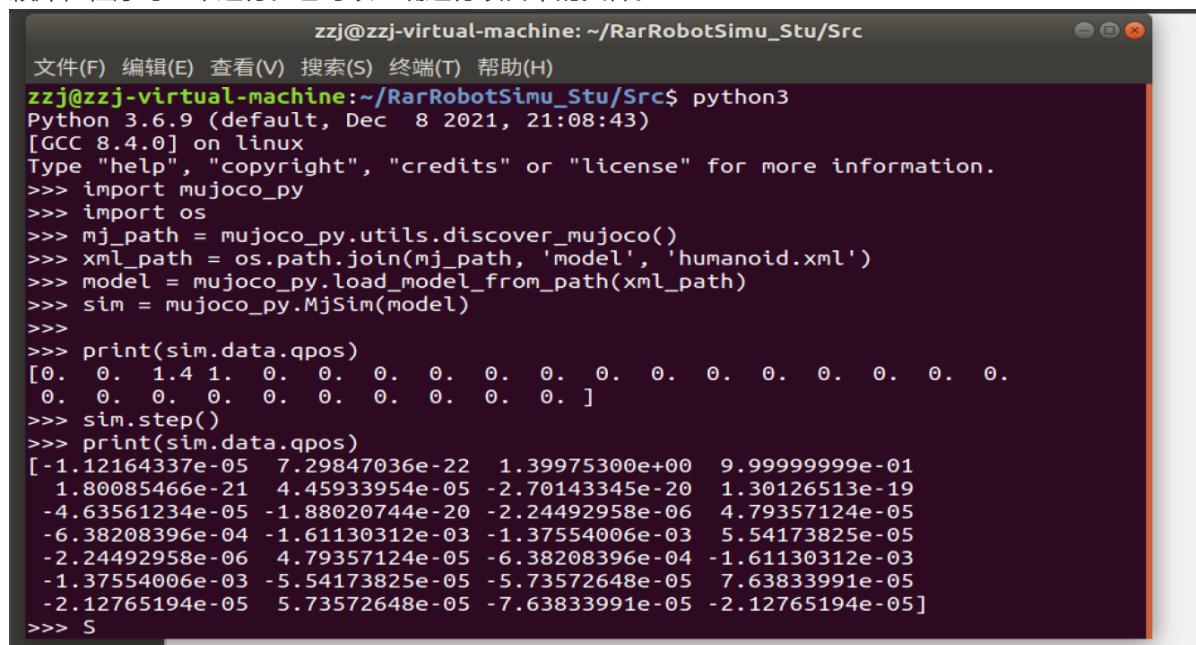
姓名：赵子健

学号：20308269

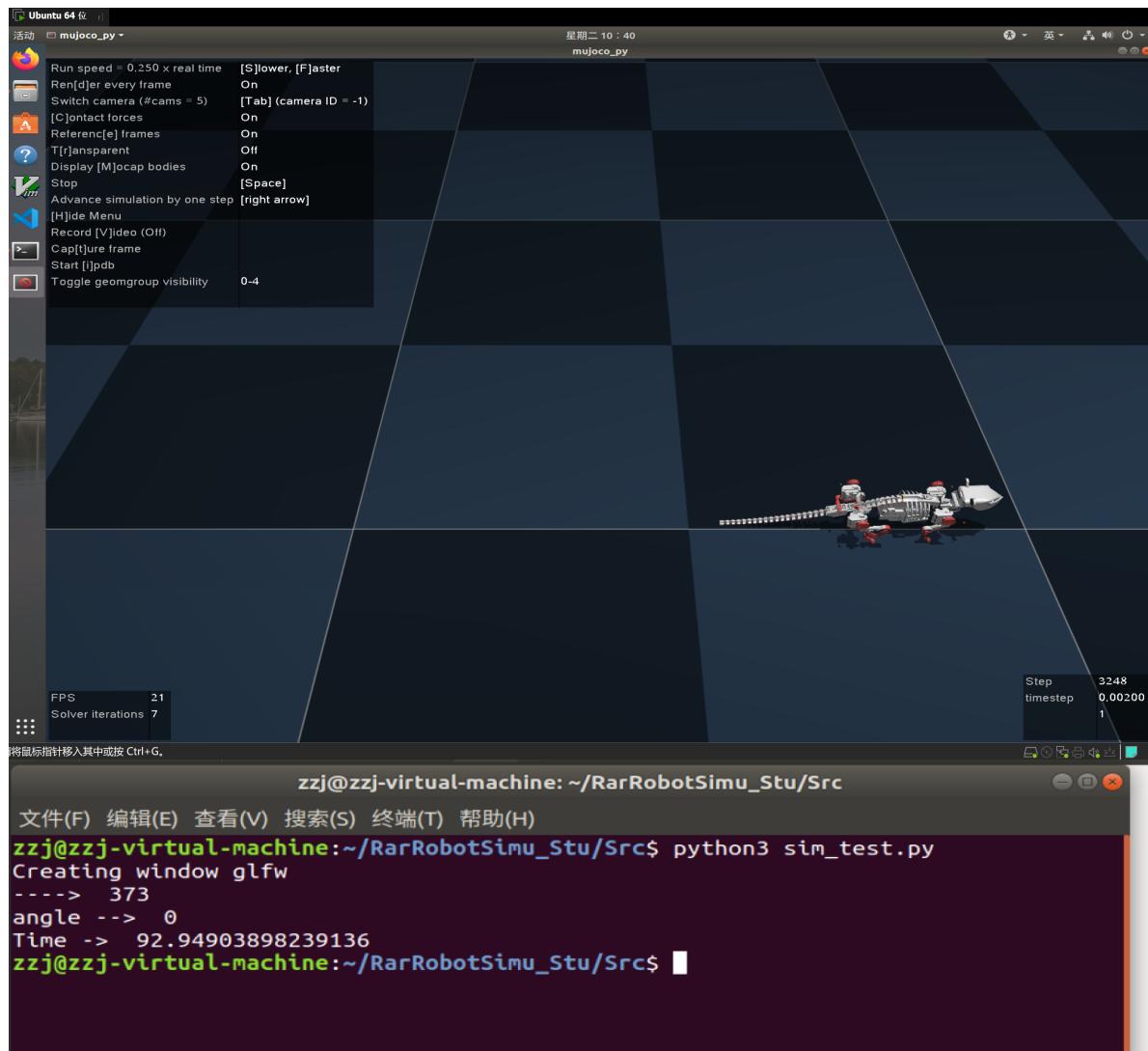
Task1：热身运动

1.1 环境安装

在Ubuntu虚拟机下安装mujoco，根据[GitHub - openai/mujoco-py: MuJoCo is a physics engine for detailed, efficient rigid body simulations with contacts. mujoco-py allows using MuJoCo from Python 3](https://github.com/openai/mujoco-py)的流程进行安装。在首次导入mujoco包后报错，后根据网上的教程更新了python与相关的依赖库，程序可正常运行。也可以正确运行项目中的文件。



```
zzj@zzj-virtual-machine: ~/RarRobotSimu_Stu/Src$ python3
Python 3.6.9 (default, Dec  8 2021, 21:08:43)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import mujoco_py
>>> import os
>>> mj_path = mujoco_py.utils.discover_mujoco()
>>> xml_path = os.path.join(mj_path, 'model', 'humanoid.xml')
>>> model = mujoco_py.load_model_from_path(xml_path)
>>> sim = mujoco_py.MjSim(model)
>>>
>>> print(sim.data.qpos)
[0.  0.  1.4 1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0. ]
>>> sim.step()
>>> print(sim.data.qpos)
[-1.12164337e-05  7.29847036e-22  1.39975300e+00  9.99999999e-01
 1.80085466e-21  4.45933954e-05  -2.70143345e-20  1.30126513e-19
 -4.63561234e-05  -1.88020744e-20  -2.24492958e-06  4.79357124e-05
 -6.38208396e-04  -1.61130312e-03  -1.37554006e-03  5.54173825e-05
 -2.24492958e-06  4.79357124e-05  -6.38208396e-04  -1.61130312e-03
 -1.37554006e-03  -5.54173825e-05  -5.73572648e-05  7.63833991e-05
 -2.12765194e-05  5.73572648e-05  -7.63833991e-05  -2.12765194e-05]
>>> S
```



1.2 仿真程序

实验任务

(1) 机器鼠由哪几个部分组成? 主要的控制器是什么? 有几个自由度?

机器鼠由头、脊柱、尾巴、四条腿 (每条腿包括2个活动关节) 组成

主要的控制器为MouseController类, 每次通过Controller获取控制数据, 将数据输入SimModel类中以在仿真环境中控制小老鼠

```
for i in range(RUN_STEPS):
    tCtrlData = theController.runStep()                      # No Spine
    #tCtrlData = theController.runStep_spine()               # With Spine
    ctrlData = tCtrlData
    theMouse.runStep(ctrlData)
```

自由度：10个（4条腿，每条腿2个自由度；脖子、头、尾巴各一个自由度）

```
def runStep(self, ctrlData):
    # -----
    # ID 0, 1 left-fore leg and coil
    # ID 2, 3 right-fore leg and coil
    # ID 4, 5 left-hide leg and coil
    # ID 6, 7 right-hide leg and coil
    # Note: For leg, it has [-1: front; 1: back]
    # Note: For fore coil, it has [-1: leg up; 1: leg down]
    # Note: For hide coil, it has [-1: leg down; 1: leg up]
    # -----
    # ID 08 is neck      (Horizontal)
    # ID 09 is head      (vertical)
    # ID 10 is spine     (Horizontal) [-1: right, 1: left]
    # Note: range is [-1, 1]
    # -----
```

(2) 绘制出机器人质心的运行轨迹。

```
def drawPath(self):
    import matplotlib.pyplot as plt
    ax = plt.axes(projection='3d')
    ax.plot3D(self.movePath[0], self.movePath[1], self.movePath[2])
    plt.show()
    for i in range(4):
        plt.plot(self.legRealPoint_x[i], self.legRealPoint_y[i])
        plt.show()
    distance=0 #计算走过的路径长度
    for i in range(len(self.movePath[0])):
        point = np.array([self.movePath[0][i], self.movePath[0][i],
        self.movePath[0][i]])
        if i==0:
            line_point1 = np.array([0, 0, 0])
            line_point2 = np.array([0, 0, 1])
        else:
            line_point1 = np.array([self.movePath[0][i - 1],
            self.movePath[0][i - 1], self.movePath[0][i - 1]])
            line_point2 = np.array([self.movePath[0][i - 1],
            self.movePath[0][i - 1], self.movePath[0][i - 1]+1])
        distance+=self.point_distance_line(point,line_point1,line_point2) #以
        相邻两次位置相邻在xoy平面上的投影作为移动距离
        print(distance)
    return distance
```

在主函数中发现了DrawPath函数应当返回距离，于是在代码中也加入了计算距离的部分，其中调用了movePath函数，这个函数的作用是计算point到直线line_point1->line_point2的距离

但同时发现原函数中point_distance_line求distance代码也存在错误，distance表达式应为
np.linalg.norm(np.cross(vec1,vec2)) / np.linalg.norm(line_point1-line_point2)而不是
np.abs(np.cross(vec1,vec2)) / np.linalg.norm(line_point1-line_point2)，在numpy中abs是求向量的绝对值，norm才能求模长。

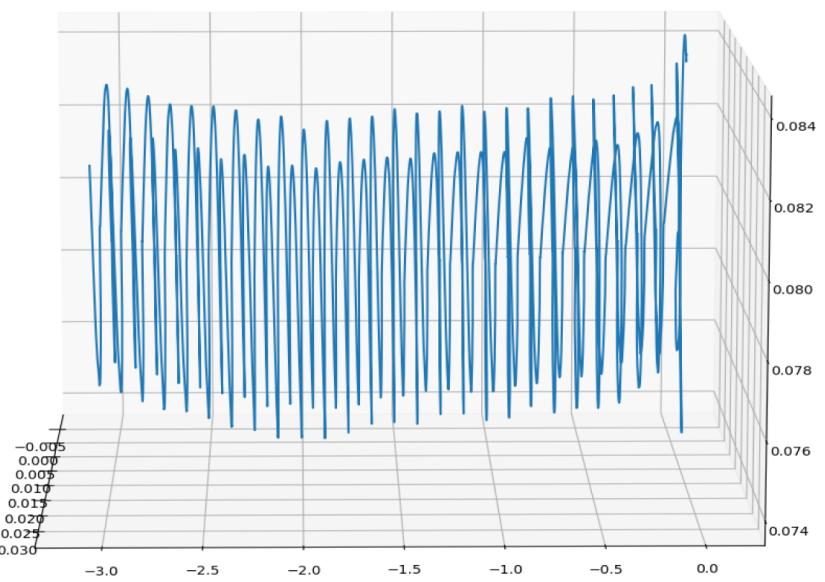
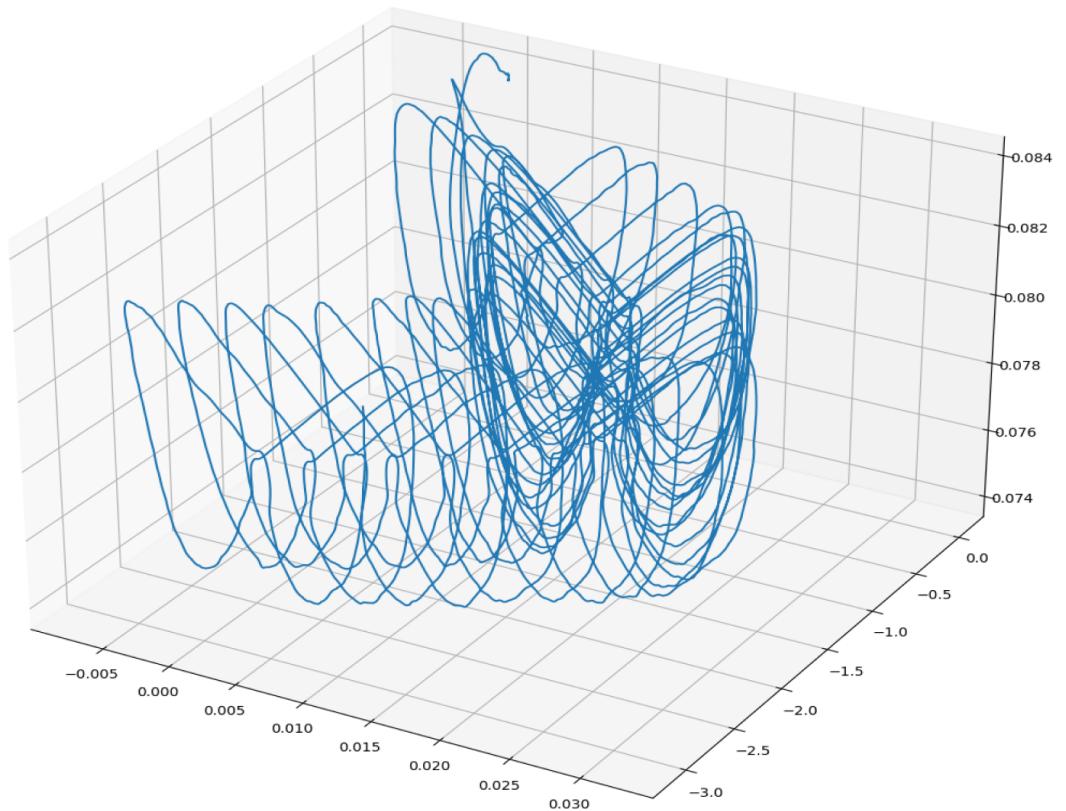
```
def point_distance_line(self, point, line_point1, line_point2): #计算point到直线line_point1->line_point2的距离
    vec1 = line_point1 - point
    vec2 = line_point2 - point
    distance = np.linalg.norm(np.cross(vec1, vec2)) / np.linalg.norm(line_point1 - line_point2)
    return distance
```

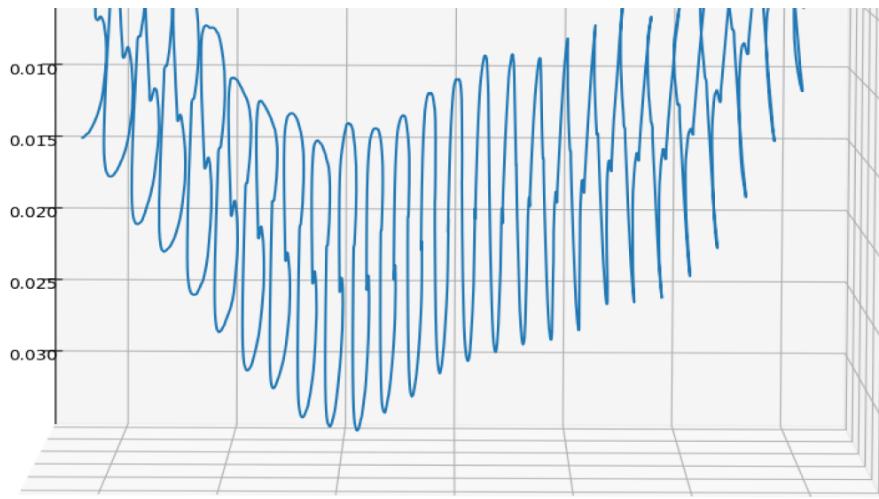
设vec1与vec2构成三角形面积为S，point到直线line_point1->line_point2的距离为h，则
np.abs(np.cross(vec1,vec2))的值等于2S，同时S也等于h*np.linalg.norm(line_point1-line_point2)/2，
所以h=np.abs(np.cross(vec1,vec2)) / np.linalg.norm(line_point1-line_point2)

movePath中存储了质心移动的轨迹，只要将相邻两点间在xoy面上投影的距离相加，即可得到总的路径长度

注：此时只能计算机器鼠在水平平面运动的距离，后续根据场景不同会对代码进行修改

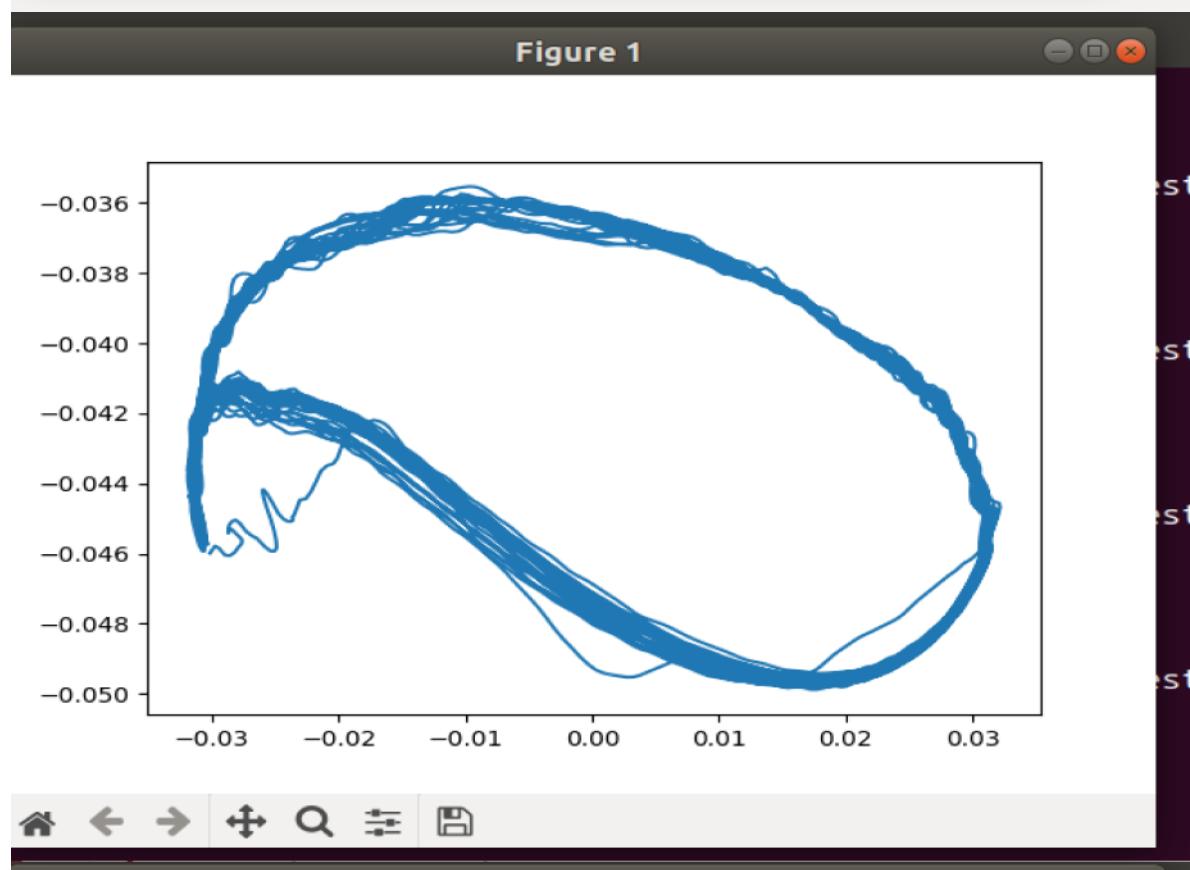
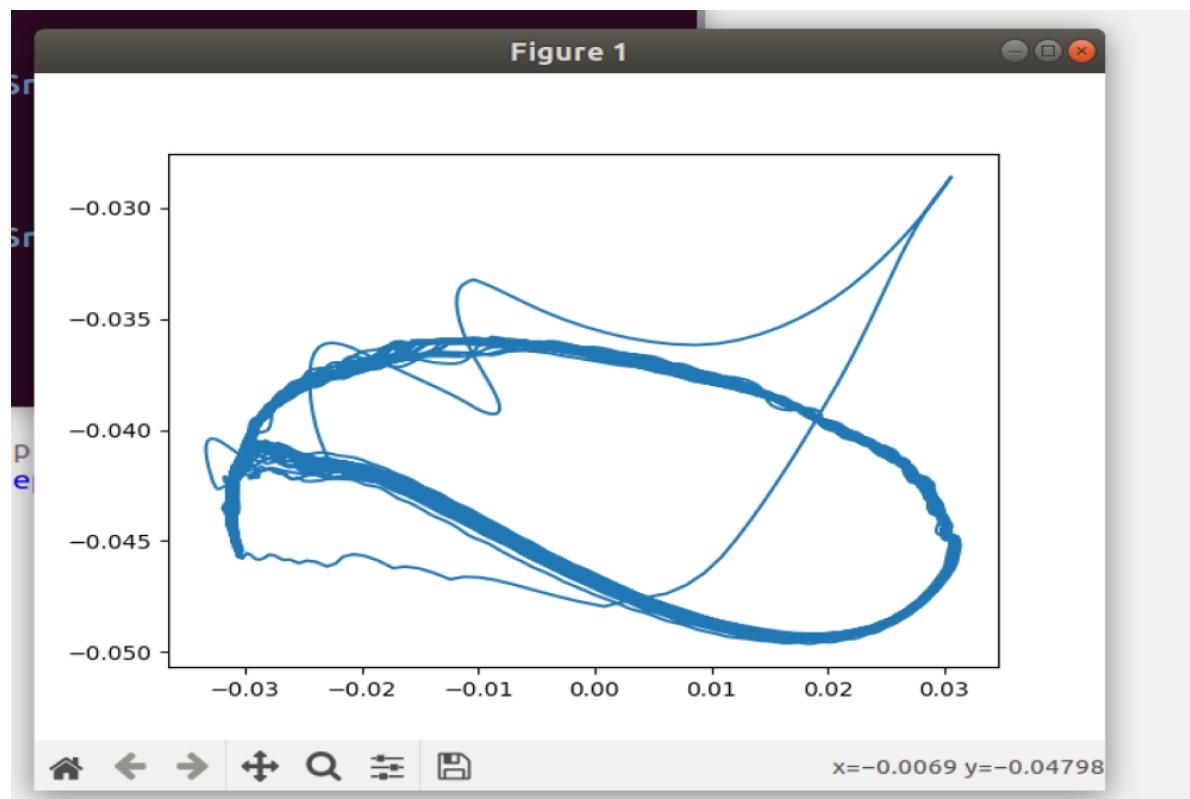
在程序结束后调用上面的函数，首先画出机器人的质心（即代码中的body_ss）运动轨迹，之后画出四条腿在自身坐标系下的运动轨迹

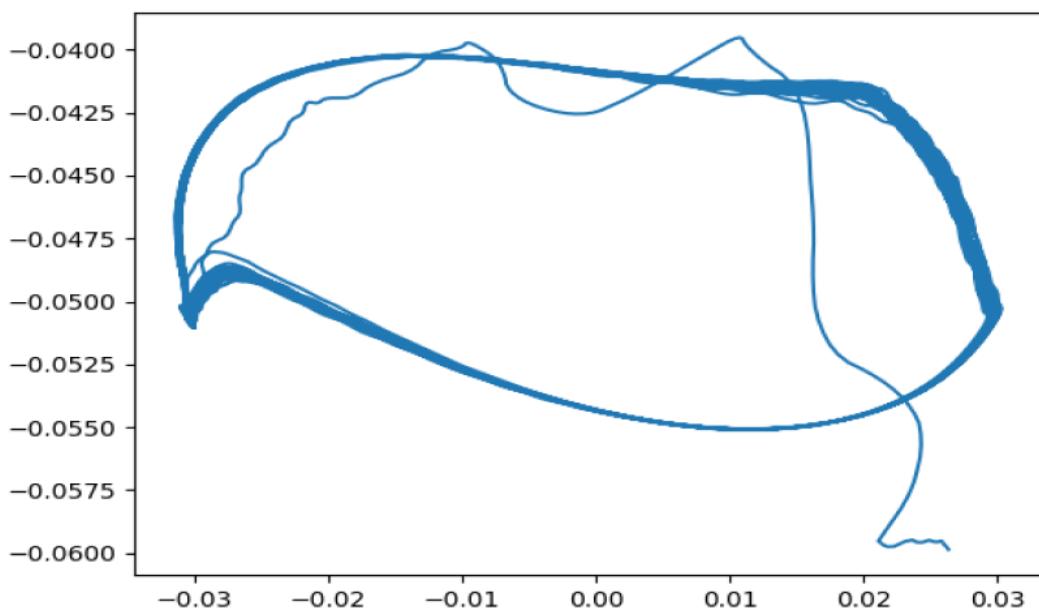
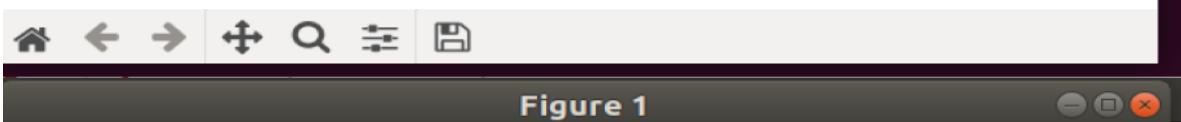
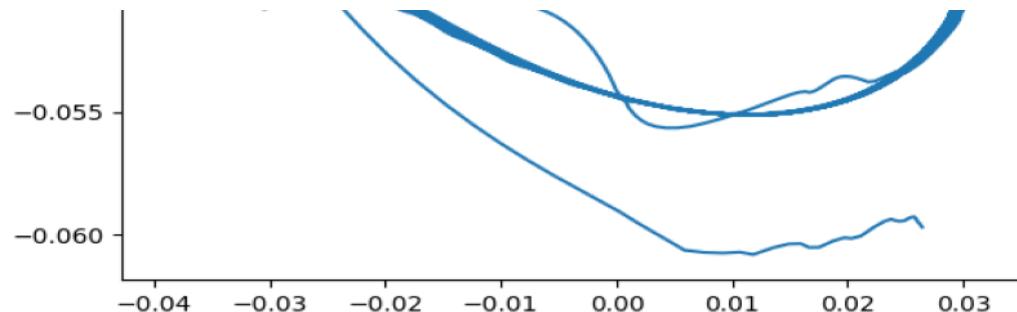




几个不同视角下的运动轨迹，右侧是起点，左侧是终点

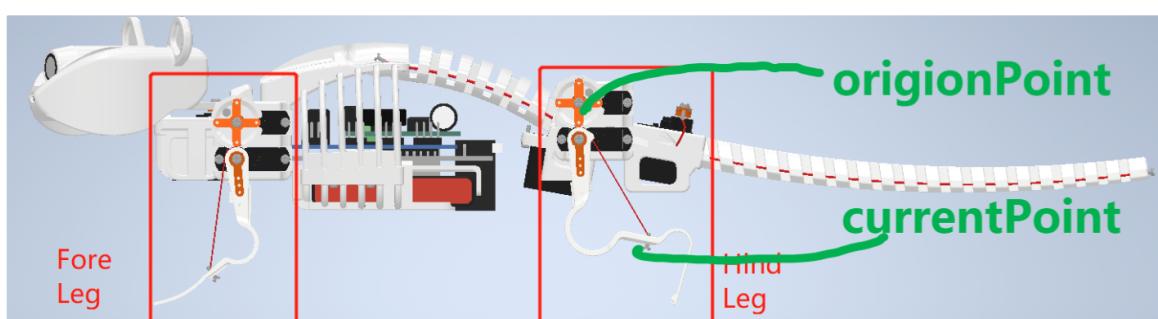
(3) 绘制出前腿和后腿在自身坐标系下的运动轨迹，并说明轨迹由哪些参数定义？





通过下图代码可知轨迹参数由legPosName中各组件位置定义，每条腿的相对运动轨迹即每条腿的foot相对于router的运动轨迹。在代码中tx、ty为绘图时用到的相对坐标，其中的originPoint对应腿部上方电机(router)中心，他的位置相对于机器鼠是固定的，下方currentPoint对应foot最边缘的位置，二者坐标作差即可得到相对坐标。

发现得到相对运动轨迹的横坐标大概在-0.3到0.3间，而纵坐标都是负值，可以判断出腿前后摆动但不会运动到router的上方；每张图像中都有两条非常粗的线（很多轨迹组合的结果），推断腿抬起的相对轨迹几乎相同、放下的轨迹也几乎相同并且与抬起的轨迹不同。



```

self.sim_state = self.sim.get_state() #返回模拟器状态的副本
self.sim.set_state(self.sim_state) #设置模拟器状态
self.legPosName = [ #shoulder对应front left/right leg(fl/fr), hip对应rear left/right leg(rl/rr),
    ["router_shoulder_fl", "foot_s_fl"],
    ["router_shoulder_fr", "foot_s_fr"],
    ["router_hip_rl", "foot_s_rl"],
    ["router_hip_rr", "foot_s_rr"]]
for i in range(4):
    originPoint = self.sim.data.get_site_xpos(self.legPosName[i][0])
    currentPoint = self.sim.data.get_site_xpos(self.legPosName[i][1])
    #print(originPoint, currentPoint)
    tX = currentPoint[1]-originPoint[1]
    tY = currentPoint[2]-originPoint[2]
    self.legRealPoint_x[i].append(tX)
    self.legRealPoint_y[i].append(tY)

```

(4) 机器鼠四条腿的运动轨迹之间存在什么关系?

由 (3) 图可发现两条前腿轨迹相似、两条后腿轨迹相似；通过仿真场景观察到左前腿与右后腿运动几乎同步、右前腿与左后腿运动几乎同步；通过代码可知在运动过程中四条腿相位差保持不变。

(5) (Optional) 画出程序架构图。

代码分析

程序包中的models文件夹中是各个场景和机器鼠模型的文件；sim_test.py中的是主代码文件，用于整个程序的控制；ToSim.py中的是与mujoco环境交互的代码；Controller.py中的是计算控制数据的代码，ToSim中使用的控制数据就是从Controller中计算得到的。所以想要理解机器鼠控制过程需要主要理解Controller代码：Controller主要调用LegModel文件夹中的代码，其中foreLeg.py和hindLeg.py分别是用来计算前腿和后腿控制参数q1、q2的代码，具体实现的方法参考Task1.3实验问题部分的推导过

程。所以接下来我将分析Controller.py和ForPath.py中的代码。

```
class LegPath(object):
    """docstring for ForeLegPath"""
    def __init__(self, pathType="circle"):
        super(LegPath, self).__init__()
        # Trot #0维表示初始位置坐标, 1维为椭圆运动轨迹的两个参数
        self.para_FU = [[-0.00, -0.045], [0.03, 0.01]]
        self.para_FD = [[-0.00, -0.045], [0.03, 0.005]]
        self.para_HU = [[0.00, -0.05], [0.03, 0.01]]
        self.para_HD = [[0.00, -0.05], [0.03, 0.005]]
    def getOvalPathPoint(self, radian, leg_flag, halfPeriod):
        pathParameter = None
        cur_radian = 0
        if leg_flag == "F":
            if radian < halfPeriod*math.pi:
                pathParameter = self.para_FU
                cur_radian = radian/halfPeriod
            else:
                pathParameter = self.para_FD
                cur_radian = (radian)/(2-halfPeriod)
        else:
            if radian < halfPeriod*math.pi:
                pathParameter = self.para_HU
                cur_radian = radian/halfPeriod
            else:
                pathParameter = self.para_HD
                cur_radian = (radian)/(2-halfPeriod)
        originPoint = pathParameter[0]
        ovalRadius = pathParameter[1]
        #根据椭圆方程和角度求坐标位置
        trg_x = originPoint[0] + ovalRadius[0] *math.cos(cur_radian)
        trg_y = originPoint[1] + ovalRadius[1] *math.sin(cur_radian)
        return [trg_x, trg_y]
```

forPath中代码的实质就是根据椭圆方程以及参数计算点的坐标，推导过程如下

LegPath部分：圆心 $(X_0, Y_0) = (\text{para}[0][0], \text{para}[0][1]) = \text{originPoint}$
椭圆参数 $\begin{cases} a = \text{para}[1][0] \\ b = \text{para}[1][1] \end{cases}$ $\alpha = \text{cur_radian}$

点坐标为 $(X_0 + a\cos\alpha, Y_0 + b\sin\alpha)$

这是椭圆方程： $\frac{(x-X_0)^2}{a^2} + \frac{(y-Y_0)^2}{b^2} = 1$

其中前腿、后腿、抬腿、落腿对应的半圆形状/位置不同，也就对应着（3）绘制出的4张图片（前两张是前腿，图像类似；后两张是后腿，图像类似），每张图有2中轨迹，对应了这四个半圆

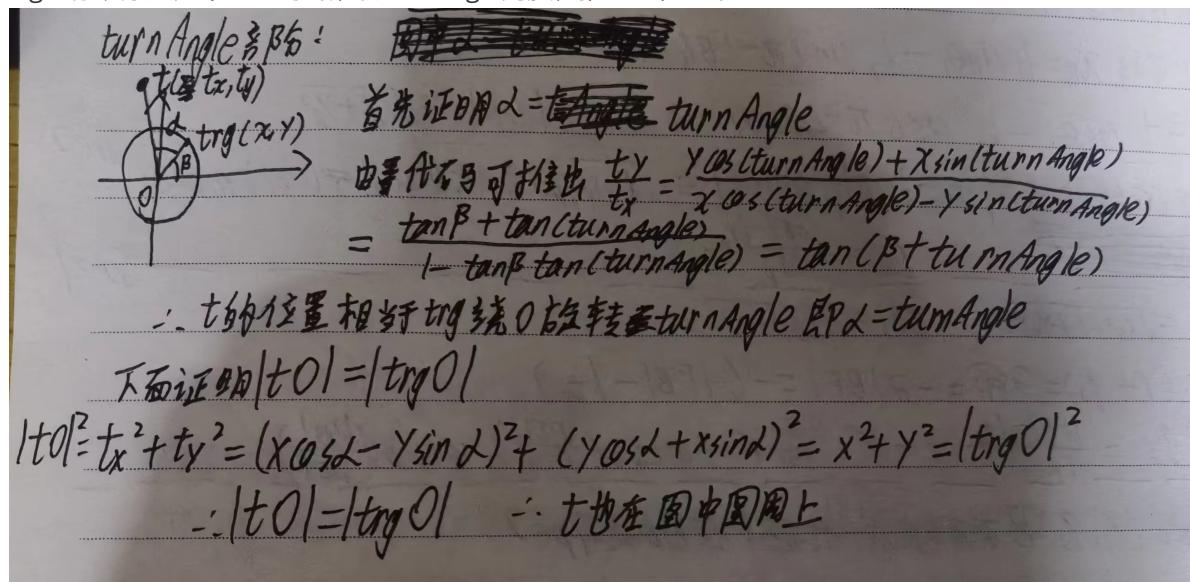
（代码中根据radian取值对应抬腿与落腿过程，按目前参数radian在 $[0, \pi]$ 对应抬腿、 $[\pi, 2\pi]$ 对应落腿）

下面我将根据我的理解简述Controller.py中各主要变量的含义

turnF、turnH分别代表前后腿的一个转动角度常量，phaseDiff则是每条腿的相位。代码中phaseDiff设定为 $[0, \pi, \pi, 0]$ 正好和（4）观察到的情况相符，置于turnF与turnH推导情况如下

```
#逆时针旋转turnAngle
tx = math.cos(turnAngle)*trg_x - math.sin(turnAngle)*trg_y;
ty = math.cos(turnAngle)*trg_y + math.sin(turnAngle)*trg_x;
```

上图为getLegCtrl中最后的部分代码，其中trg_x与trg_y为经过forPath.py（即上述计算椭圆上坐标的过程）计算出的坐标，turnAngle为turnF或turnH决定（取决于是前腿还是后腿），两行代码的作用是将trg坐标以原点为中心逆时针旋转turnAngle角度，推导过程如下



此外，SteNum是由freq决定的（freq类似步长，SteNum类似周期长度，二者成反比），虽然我没有理解具体的计算过程，但看起来这是将完整的运动过程离散化，curStep代表了当前的运动阶段，每次运行runStep会将curStep加1，具体更新规则如下

```
self.curStep = (self.curStep + 1) % self.SteNum
```

最后需要对SteNum取模也证明了上述离散化的想法应该是正确的，即1个周期有SteNum步；之后上面提到的PhaseDiff也被转化为StepDiff，即用步数差替换相位差，在Ctrl部分代码Step类数据会再次被转化为角度数据，两次转化过程如下图

```
self.stepDiff = [0, 0, 0, 0]
for i in range(4):
    self.stepDiff[i] = int(self.SteNum * self.phaseDiff[i]/(2*PI))
self.stepDiff.append(int(self.SteNum * self.spinePhase/(2*PI)))
```

```
curStep = curStep % self.SteNum
radian = 2*np.pi * curStep/self.SteNum
```

剩余参数包括前后腿的控制类、脊柱的控制参数

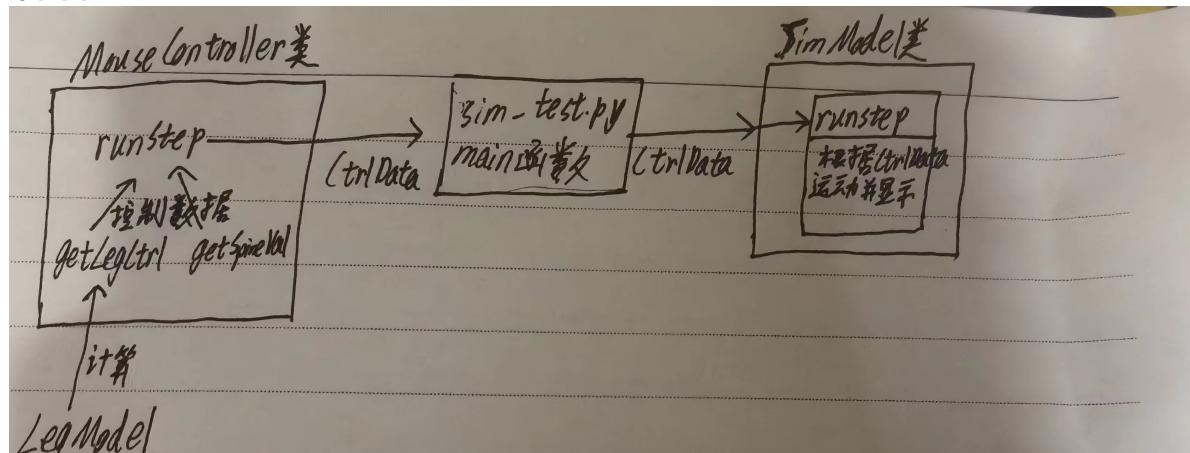
下面我将简述Controller中各函数的作用及其工作原理

runStep是主要控制代码，首先根据之前提到的相位stepDiff与当前的curStep调用getLegCtrl(二者共同控制转动角度)以获取控制参数q1、q2，再调用getSpineVal获取脊柱的参数，之后更新curStep，再将所有的控制参数封装到ctrlData中并返回。

getLegCtrl则根据传入的curStep调用forPath和foreLeg/hindLeg中的代码计算两个电机的转动角度q1、q2。

最终得到的ctrlData传给SimModel中的runStep函数，将其放入模型的ctrl参数中，之后模型根据新参数产生相应的动作并显示在屏幕中。

架构图



遇到的问题

在进行绘图时，首先在drawPath函数中添加matplotlib.pyplot的绘图命令进行测试，但发现程序会报错无法打开matplotlib的figure窗口，于是为了调试这一错误，我先没有使用drawPath函数，而是在sim_test.py结尾添加了一段绘图命令进行测试

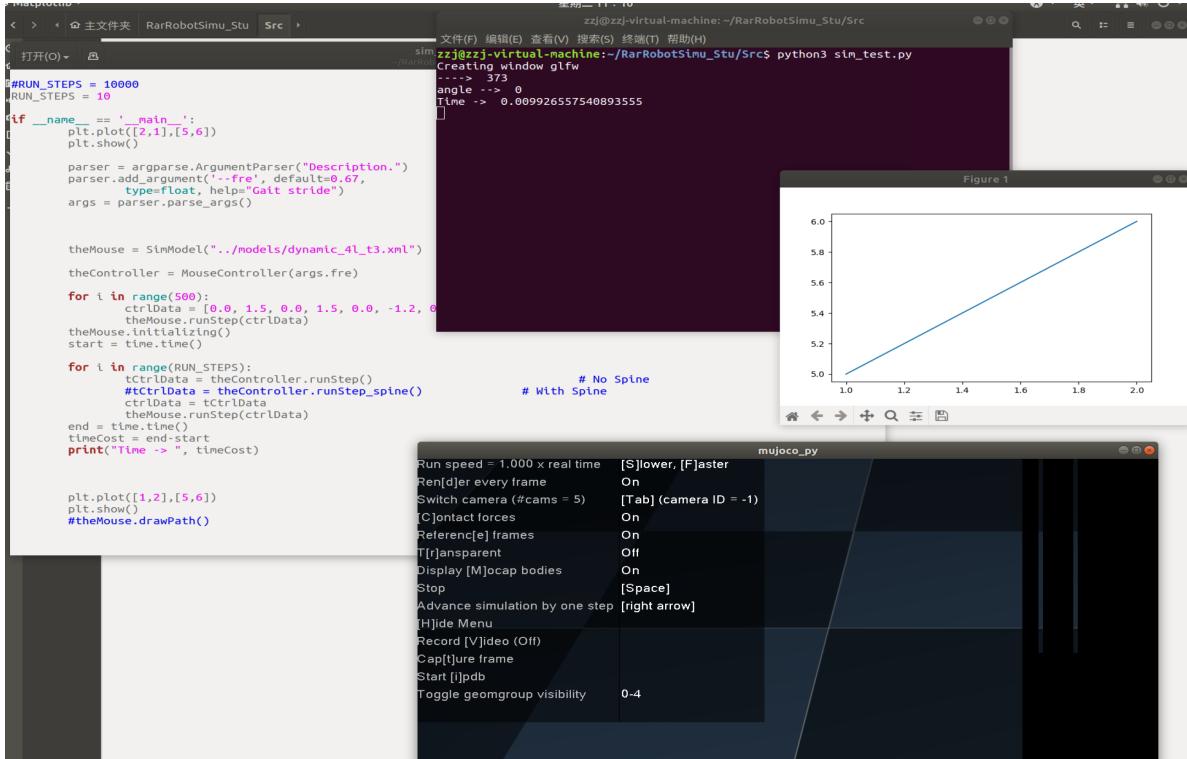
```
theMouse.runStep(ctrlData)
end = time.time()
timeCost = end-start
print("Time -> ", timeCost)
```

```
plt.plot([1,2],[5,6])
plt.show()
#theMouse.drawPath()
```

```
zzj@zzj-virtual-machine:~/RarRobotSimu_Stu/Src$ python3 sim_test.py
Creating window glfw
----> 373
angle --> 0
Time -> 0.009447097778320312
段错误 (核心已转储)
```

发现程序仍然报错，我又将这一段代码加入到程序首部，发现可以正常运行。最终反复测试发现，若在机器鼠仿真执行前执行绘图语句，则出现在程序中任何位置的绘图语句都可以正常运行（即若在机器鼠仿真开始前进行一次绘图，则在小老鼠仿真结束后再进行一次绘图则可以正常绘制）；而若在机器鼠仿真开始后执行绘图指令，无论在哪个位置执行都会报错（无论是机器鼠仿真执行过程中还是结束以后），

于是我使用gdb进行调试



上图为发现实验中只要在小老鼠仿真前启动plt.show，程序任何部位的绘图指令都可正确运行的截图

```
zzj@zzj-virtual-machine:~/RarRobotSimu_Stu/Src$ gdb python3
GNU gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from python3...(no debugging symbols found)...done.
(gdb) r sim_test.py
Starting program: /usr/bin/python3 sim_test.py
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[New Thread 0x7fffff2df4700 (LWP 3056)]
[Thread 0x7fffff2df4700 (LWP 3056) exited]
Creating window glfw
----> 373
angle --> 0
Time -> 0.037400245666503906
[New Thread 0x7fffff2df4700 (LWP 3063)]
[New Thread 0x7fffc242b700 (LWP 3064)]

Thread 1 "python3" received signal SIGSEGV, Segmentation fault.
__GI__pthread_mutex_lock (mutex=0x0) at ../nptl/pthread_mutex_lock.c:67
67      ..../nptl/pthread_mutex_lock.c: 没有那个文件或目录.
(gdb)
```

通过gdb调试只在结尾加入绘图命令的sim_test.py，发现程序首先创建了一个新线程1，线程1退出，之后创建了线程2、3，之后发生了signal sigsev（网上解释为内存访问错误）。后经过测试发现，运行py程序会不显示地创建一个线程0，当导入matplotlib.pyplot后会创建一个线程1并退出，当调用plt.show后会新建2个线程（2-4），当程序结束后程序会先退出线程4-2，接着退出python线程0。证明过程如下：

```
Reading symbols from python3...(no debugging symbols found)...done.
(gdb) r test.py
Starting program: /usr/bin/python3 test.py
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[Inferior 1 (process 3162) exited normally]
(gdb)
```

1+1

```
(gdb) r test.py
Starting program: /usr/bin/python3 test.py
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[New Thread 0x7ffff064f700 (LWP 3174)]
[Thread 0x7ffff064f700 (LWP 3174) exited]
[Inferior 1 (process 3171) exited normally]
(gdb)
```

```
import matplotlib.pyplot as plt
#plt.plot([1,2,3],[4,5,6])
#plt.show()
```

1+1

```
(gdb) r test.py
Starting program: /usr/bin/python3 test.py
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[New Thread 0x7ffff2cb2700 (LWP 3500)]
[Thread 0x7ffff2cb2700 (LWP 3500) exited]
[New Thread 0x7ffff2cb2700 (LWP 3502)]
[New Thread 0x7ffffdb831700 (LWP 3503)]
[New Thread 0x7ffffd8c4a700 (LWP 3504)]
[Thread 0x7ffffd8c4a700 (LWP 3504) exited]
[Thread 0x7ffffdb831700 (LWP 3503) exited]
[Thread 0x7ffff7fb0740 (LWP 3499) exited]
[Inferior 1 (process 3499) exited normally]
(gdb)
```

```
import matplotlib.pyplot as plt
plt.plot([1,2,3],[4,5,6])
plt.show()
```

后来我上网查找了matplotlib.pyplot的源码[matplotlib/pyplot.py at 30b320a535d47e3adf054e845766218ff8b7cf1b · matplotlib/matplotlib · GitHub](#)

，他的运行流程大致为：首先创建线程1初始化一些函数需要的变量，之后当调用show函数时会新创建三个线程，其中用到了线程1初始化的某些变量。

原程序在创建线程4前发生signal sigsegv错误（错误的访问了某些地址），我怀疑是程序中某些函数修改了matplotlib线程1初始化的变量而导致的，于是我查阅了mujoco_py中的MjViewer的源码[mujoco-py/mjviewer.py at master · openai/mujoco-py · GitHub](#)（由于show函数与MjViewer都是与绘图有关的，所以首先想到查阅这一函数），之后发现MjViwer中会调用glfw库，我怀疑是glfw与matplotlib产生了某些冲突，于是阅读了glfw的源码[GitHub - glfw/glfw: A multi-platform library for OpenGL, OpenGL ES, Vulkan, window and input](#)，其中的说明文档提到在glfw使用完毕后要使用terminate函数对其进行关闭以释放一些变量，于是在sim_test.py结尾先添加了glfw.terminate()函数，之后再添加有关绘图的函数，程序依旧报错。

之后为了检查是否是glfw与matplotlib产生了冲突，我进行了如下测试：

```
打开(O)  盘
(gdb) r test.py
Starting program: /usr/bin/python3 test.py
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[New Thread 0x7ffff2cb2700 (LWP 3598)]
[Thread 0x7ffff2cb2700 (LWP 3598) exited]
[New Thread 0x7ffff2cb2700 (LWP 3601)]
[New Thread 0x7ffffda4c3700 (LWP 3602)]

Thread 1 "python3" received signal SIGSEGV, Segmentation fault.
  _GI__pthread_mutex_lock (mutex=0x0) at ../nptl/pthread_mutex_lock.c:67
  67  .../nptl/pthread_mutex_lock.c: 没有那个文件或目录.
(gdb)  ■
```

发现果然是glfw导致的问题出现，我怀疑是glfw修改了matplotlib产生的线程1初始化的某些变量，而python中glfw的terminate函数存在某些问题，导致当glfw退出后没有完全的恢复现场，所以导致了段错误的产生。于是想到，在glfw关闭后再导入matplotlib包可能会解决这一问题，进行测试后问题果然解决

(gdb) r test.py
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /usr/bin/python3 test.py
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[New Thread 0x7ffffe8b49700 (LWP 3621)]
[Thread 0x7ffffe8b49700 (LWP 3621) exited]
[New Thread 0x7ffffe8b49700 (LWP 3623)]
[New Thread 0x7ffffd3c35700 (LWP 3624)]
[Thread 0x7ffffd3c35700 (LWP 3624) exited]
[Thread 0x7ffffe8b49700 (LWP 3623) exited]
[Inferior 1 (process 3619) exited normally]

test.py
~/RarRobotSimu_Stu/Src

```
import glfw
glfw.init()
glfw.terminate()

import matplotlib.pyplot as plt
plt.plot([1,2,3],[4,5,6])
plt.show()
```

之后我将项目代码中的matplotlib包导入部分转移到drawPath函数中，解决了这一问题；但我的室友在实验时并没有遇到我的问题，我猜测是我的某些库版本较老存在bug导致的这一问题。

1.3熟悉四足运动机制

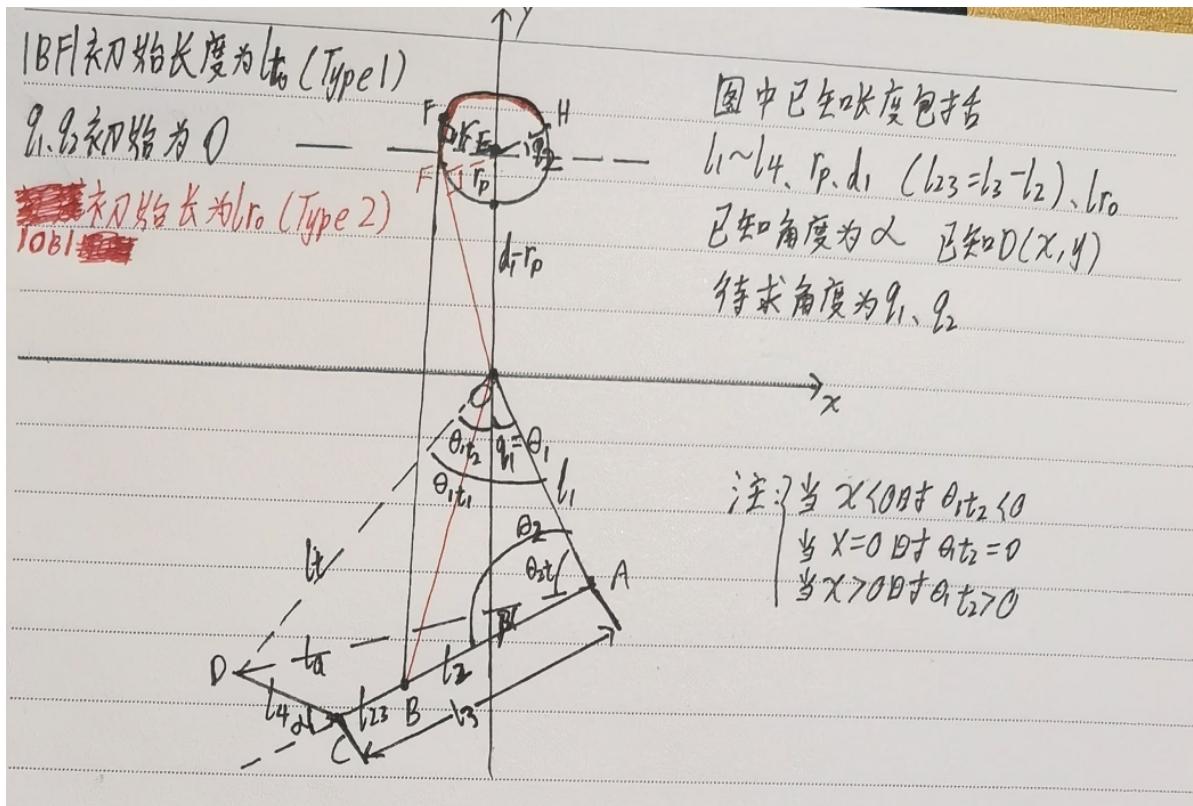
实验任务

- (1) 阅读参考文档，回答问题：机器鼠的执行电机如何实现对四足的控制？

我在实验问题部分重新梳理了代码中所用的Type2控制方式，控制代码将想要D到达的相对位置 (x,y) 转化为两个电机的转动角度(q1,q2)，进而实现四足控制

- (2) 根据文档，结合仿真程序代码，计算出前腿或后腿末端点（足尖）能够到达的所有区域，并绘图表示。

下面为推导前腿可到达范围的过程，后腿同理



下面求解 (x, y) 的范围，当 (x, y) 未知时：

当 H 到达图最下方位时，OB 最长，此时 $|OB|_{\max} = l_0 - (d_1 - r_p)$

此时 θ_2 最大，有 $\theta_{2\max} = \arccos \frac{l_1^2 + l_2^2 - |OB|_{\max}^2}{2l_1l_2}$

同时有 $\theta_{2t\max} = \theta_{2\max} - \beta$

而 θ_{2t} 最小时接近 0，为了方便后续计算取 $\theta_{2t\min} = 10^{-5}$

于是可得 $\theta_{2t} \in [10^{-5}, \theta_{2\max} - \beta]$

$$l_t = \sqrt{l_1^2 + l_2^2 - 2l_1l_2\cos\theta_{2t}} \quad \theta_{1t_1} = \arccos \frac{l_1^2 + l_t^2 - l_2^2}{2l_1l_t}$$

$$\theta_{1t_2} = \theta_1 - \theta_{1t_1} \quad (\theta_{1t_2} 为负值表示 D 在轴左侧)$$

$$(X, Y) = (l_t \sin \theta_{1t_2}, -l_t \cos \theta_{1t_2})$$

从而根据 θ_2 和 θ_{2t} 的取值便可以得到 (x, y) 的取值范围

下面为上述过程的代码实现

```
def compute_frontleg_range(params):
    l0 = params['l0']
    rp = params['rp']
    d1 = params['d1']
    l1 = params['l1']
    l2 = params['l2']
    l3 = params['l3']
    l4 = params['l4']
```

```

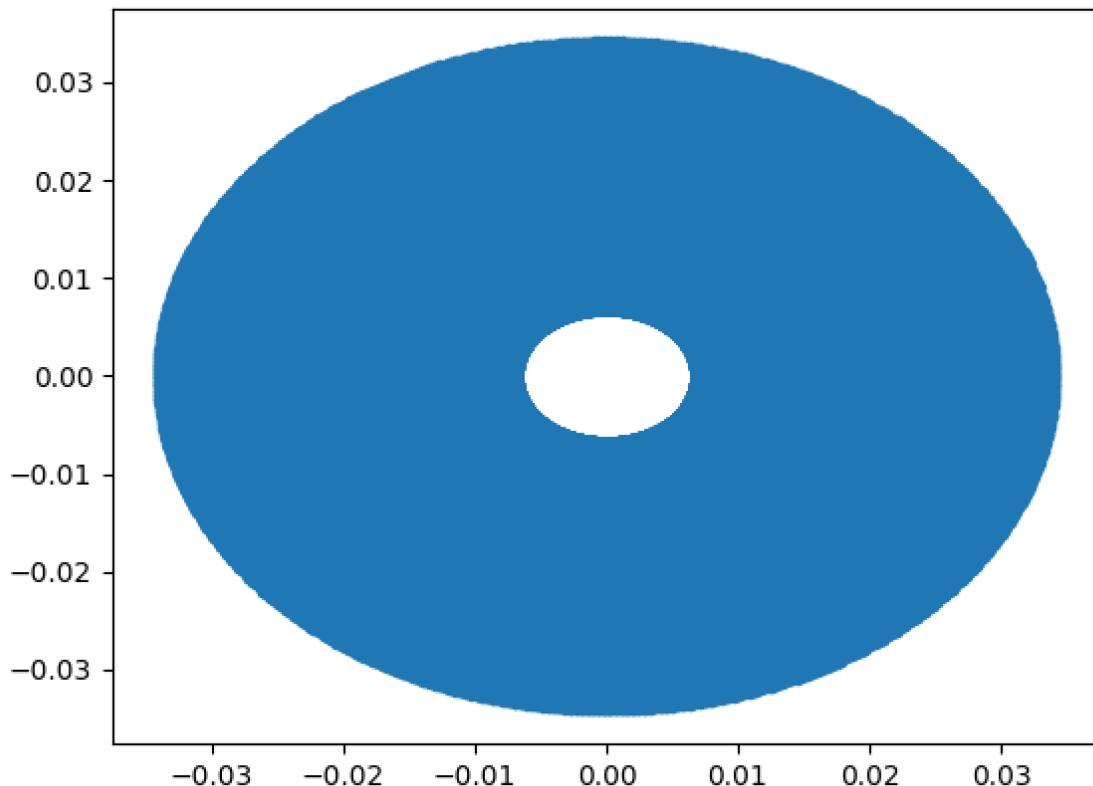
alpha = params['alpha']
l_a = LawOfCosines_edge(13, 14, (math.pi - alpha))
beta = LawOfCosines_angle(l_a, 13, 14)

l_b_max = lr0 - (d1 - rp)
theta2_t_min = 10**(-5)
theta2_t_max = LawOfCosines_angle(l1, l2, l_b_max) - beta
x=[]
y=[]
for i in range(500):
    for j in range(500):
        theta2_t=theta2_t_min+i/499*(theta2_t_max-theta2_t_min)
        q1=j/499*math.pi
        D=compute_D(l1,l_a,theta2_t,q1)
        x.append(D[0])
        y.append(D[1])
plt.plot(x,y,'.')
plt.show()

def compute_D(l1,l_a,theta2_t,q1):
    lt=LawOfCosines_edge(l_a, l1, theta2_t)
    theta1_t1=LawOfCosines_angle(l1, lt, l_a)
    theta1_t2=q1-theta1_t1
    x=lt*math.sin(theta1_t2)
    y=-lt*math.cos(theta1_t2)
    return [x,y]

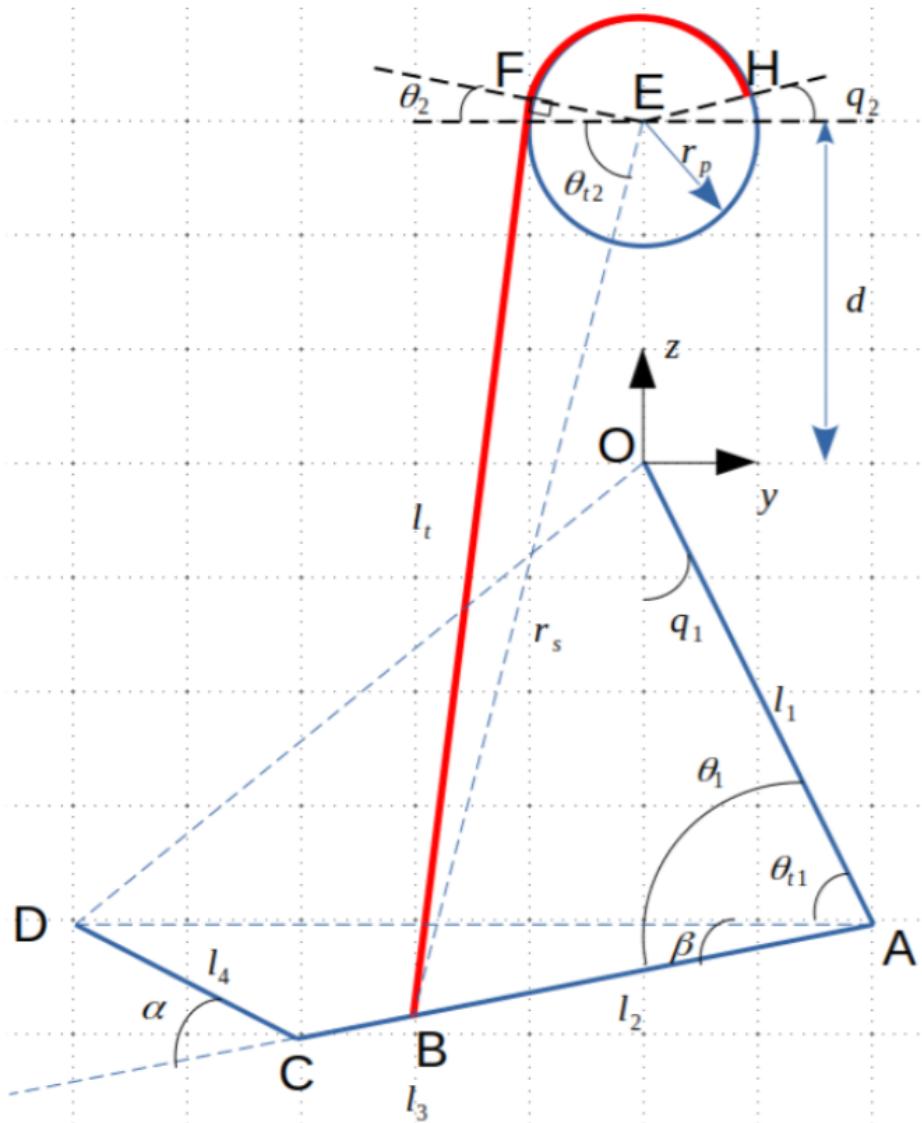
```

q1取值范围为 $[0, 2\pi]$, 前腿可到达的区域如下图所示 (与1.2 (5) 中提到的椭圆对应)

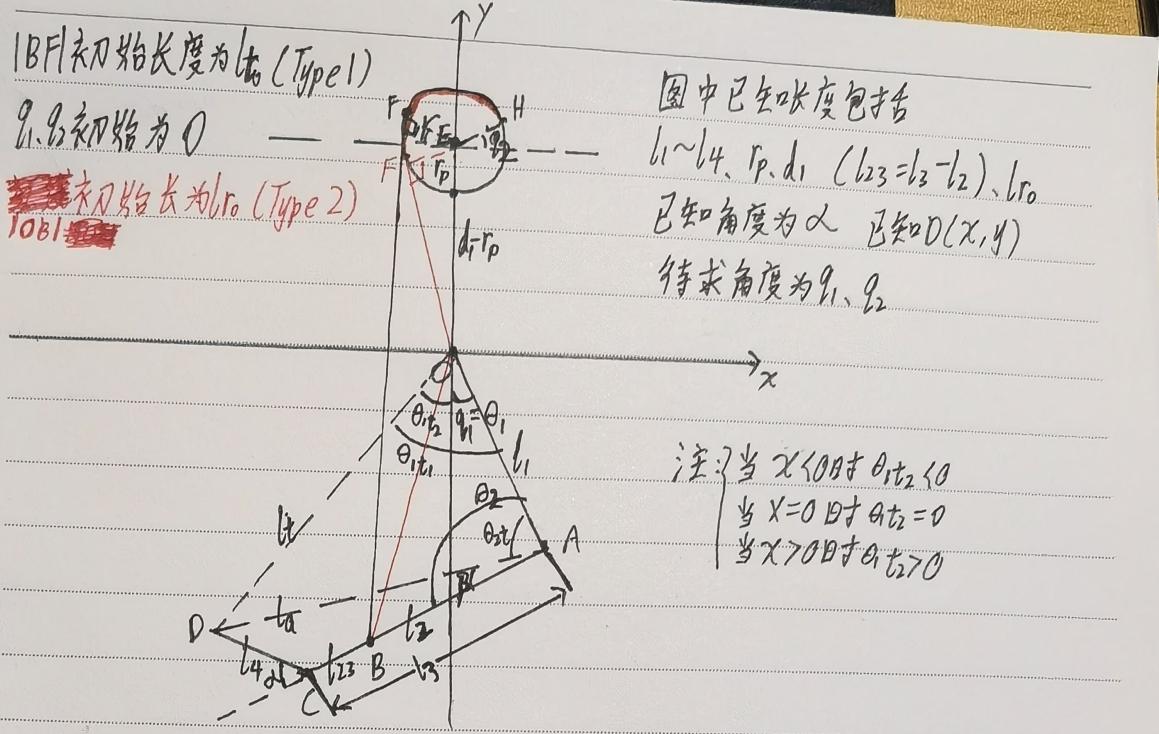


实验问题

在阅读参考文档和foreLeg.py时，我发现文档中部分推导是错误的（代码中是正确的），文档中默认AD是水平的，导致许多计算是不正确的（如对B点坐标的计算），此外代码中的字母标识与参考文档中也有所不同且使用的类型为Type2，我根据代码重新梳理了推导过程



上图为参考文档中的图片，在推导过程中默认了AD是水平的，下图为梳理的推导过程



$$l_a = |AD| = \sqrt{l_3^2 + l_4^2 - 2l_3l_4 \cos(\pi - \alpha)}$$

$$\theta_{2t} = \arccos\left(\frac{l_1^2 + l_4^2 - l_3^2}{2l_1l_4}\right)$$

$$\theta_{1t} = \arccos\left(\frac{l_1^2 + l_4^2 - l_3^2}{2l_1l_4}\right)$$

$$\text{arc cos}\left(\frac{l_1^2 + l_2^2 - l_4^2}{2l_1l_2}\right) = \beta$$

$$\theta_2 = \theta_{2t} + \beta$$

$$\theta_{1t} = \text{sign}(x) \cdot \arccos\left(\frac{y}{l_t}\right)$$

↓
图中的θ_{1t}为负值 ⇒ q₁ = θ₁ = θ_{1t} + θ_{1t2}

如图，设AB与Y轴夹于L，则∠BGO = θ₁ + θ₂

$$\text{由此可得 } x_B = l_1 \sin \theta_1 - l_2 \sin(\pi - \angle BGO)$$

$$y_B = l_1 \cos \theta_1 - l_2 \cos(\pi - \angle BGO) \quad |OB| = \sqrt{x_B^2 + y_B^2} \quad \text{Type 2}$$

q₂ 对应的弧长为 |l₀ - |OB|| (由F1OF1 ≡ √(d² - r_p²), 故弓形长变化量 = |OB| 变化量 = 0)

$$\frac{q_2}{r_p} = \frac{|OB| - l_0}{r_p} \quad \text{Type 1}$$

$$|EB| = \sqrt{x_B^2 + (d - y_B)^2} \quad |FB| = \sqrt{|EB|^2 - r_p^2}$$

$$-r_p(r + q_2) = \partial \widehat{FH} = -\partial |BF| = -(|FB| - l_{t0})$$

$$q_2 = \frac{|FB| - l_{t0}}{r_p} - r \quad \text{而 } r = \arccos\left(\frac{|FE|}{|EB|}\right) - \arccos\left(\frac{|FB|}{|EB|}\right)$$

至此 (q₁, q₂) 已求解完成，代码中使用的是 Type 2

Task2：运动实验

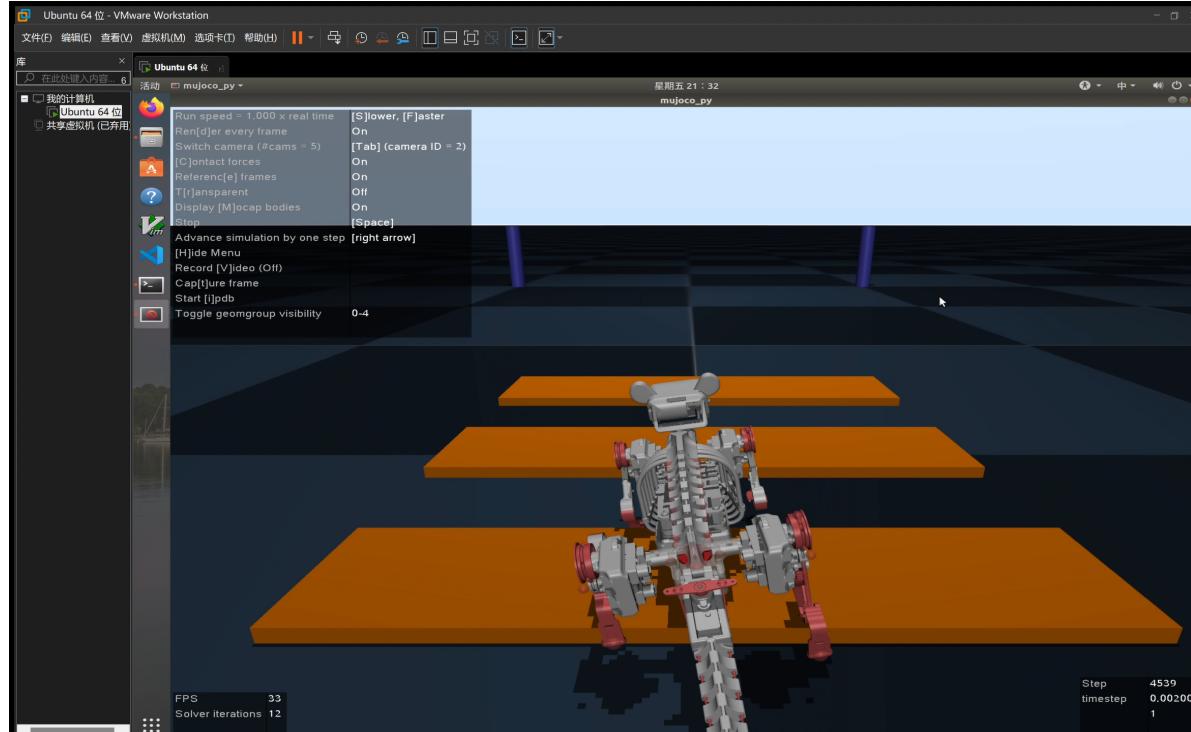
2.1场景运动实验

实验任务

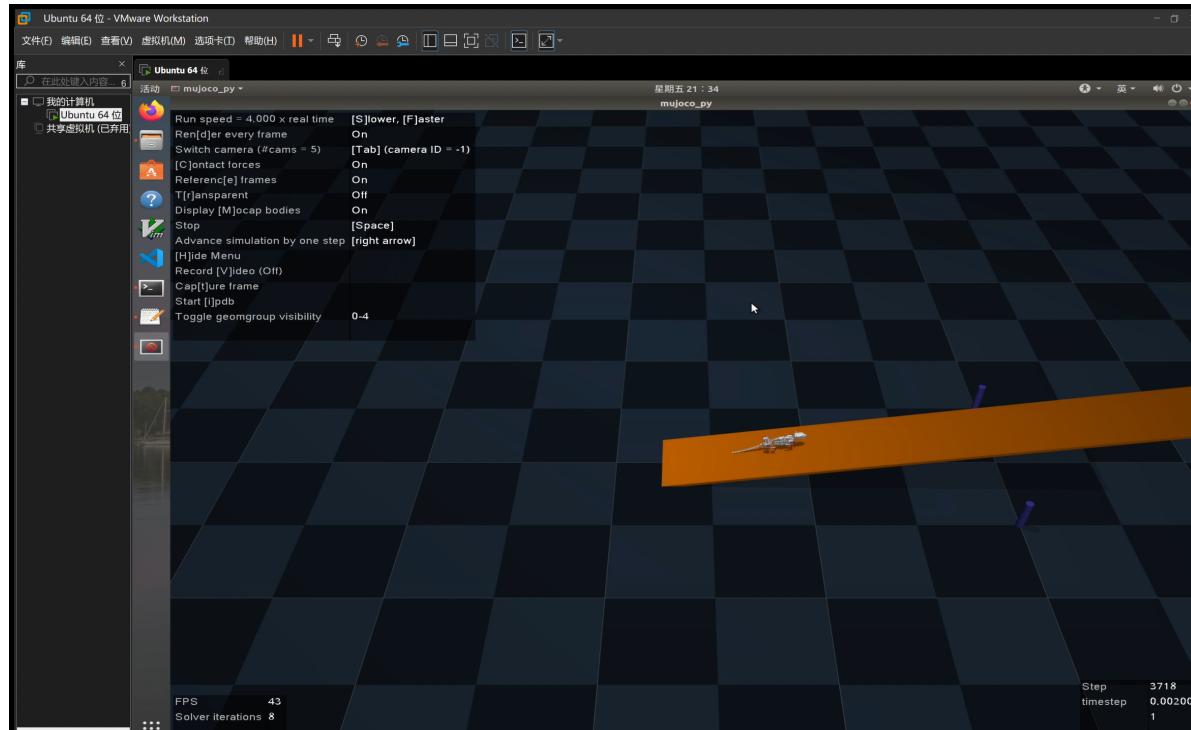
(1) 请使用仿真程序加载不同的场景，分别录屏记录。

若视频不显示，请点击链接<https://www.aliyundrive.com/s/KXCjMbp6aQk>

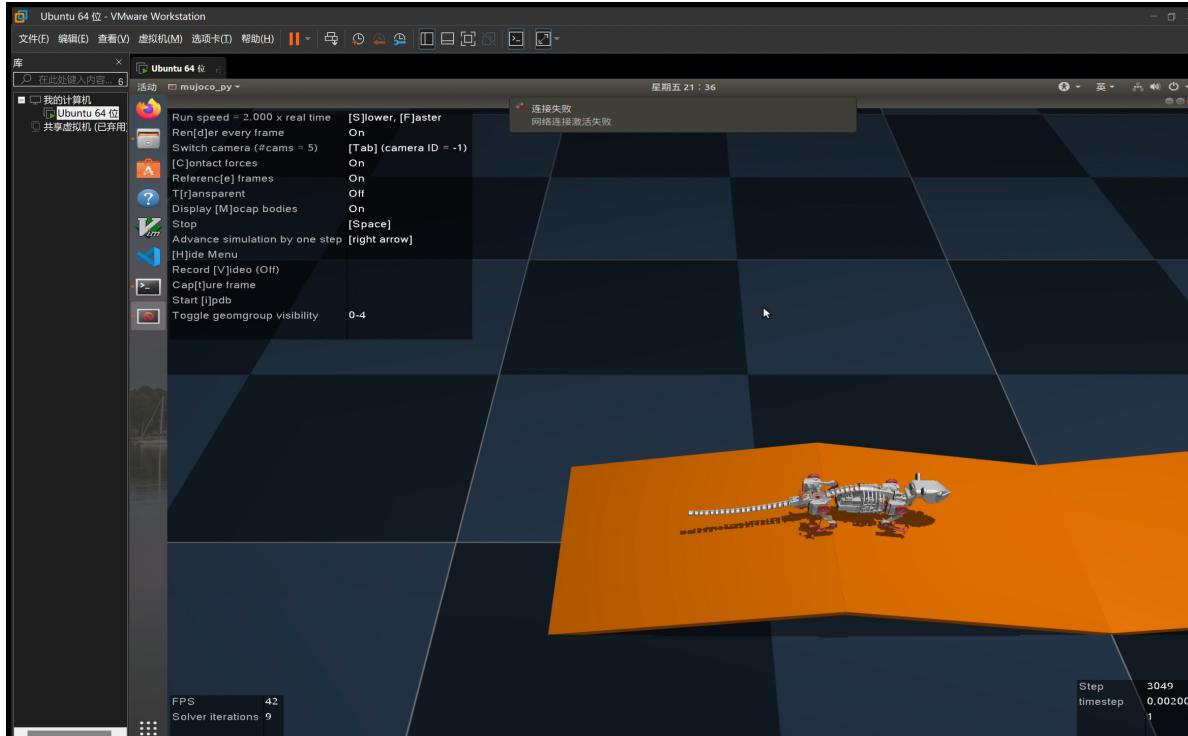
Scene1: 木板



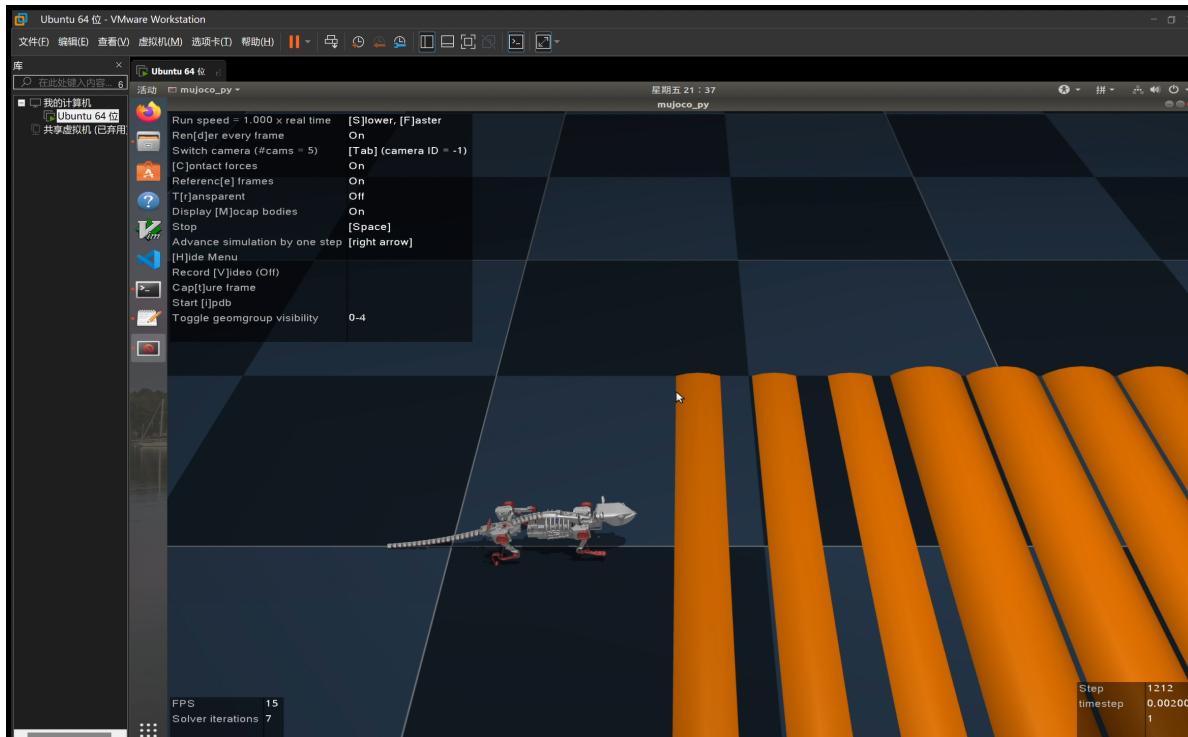
Scene2: 上坡



Scene2pro: 起伏地形



Scene3: 弧面障碍地形



(2) 总结各个场景的运行情况。

Scene1木板场景中机器鼠前腿正常通过第一个木板但后腿被卡住。视频中前腿踏入木板前离木板距离较远，因此有足够的空间让前腿充分抬起；但后腿要跨越木板时距离过近，没有充足的空间让机器鼠完全抬起后腿。

Scene2上坡场景可正常通过。

Scene2pro起伏地形场景机器鼠在第一次下坡到第二次上坡之间前腿被卡住，前腿在夹缝中无法完全抬起。

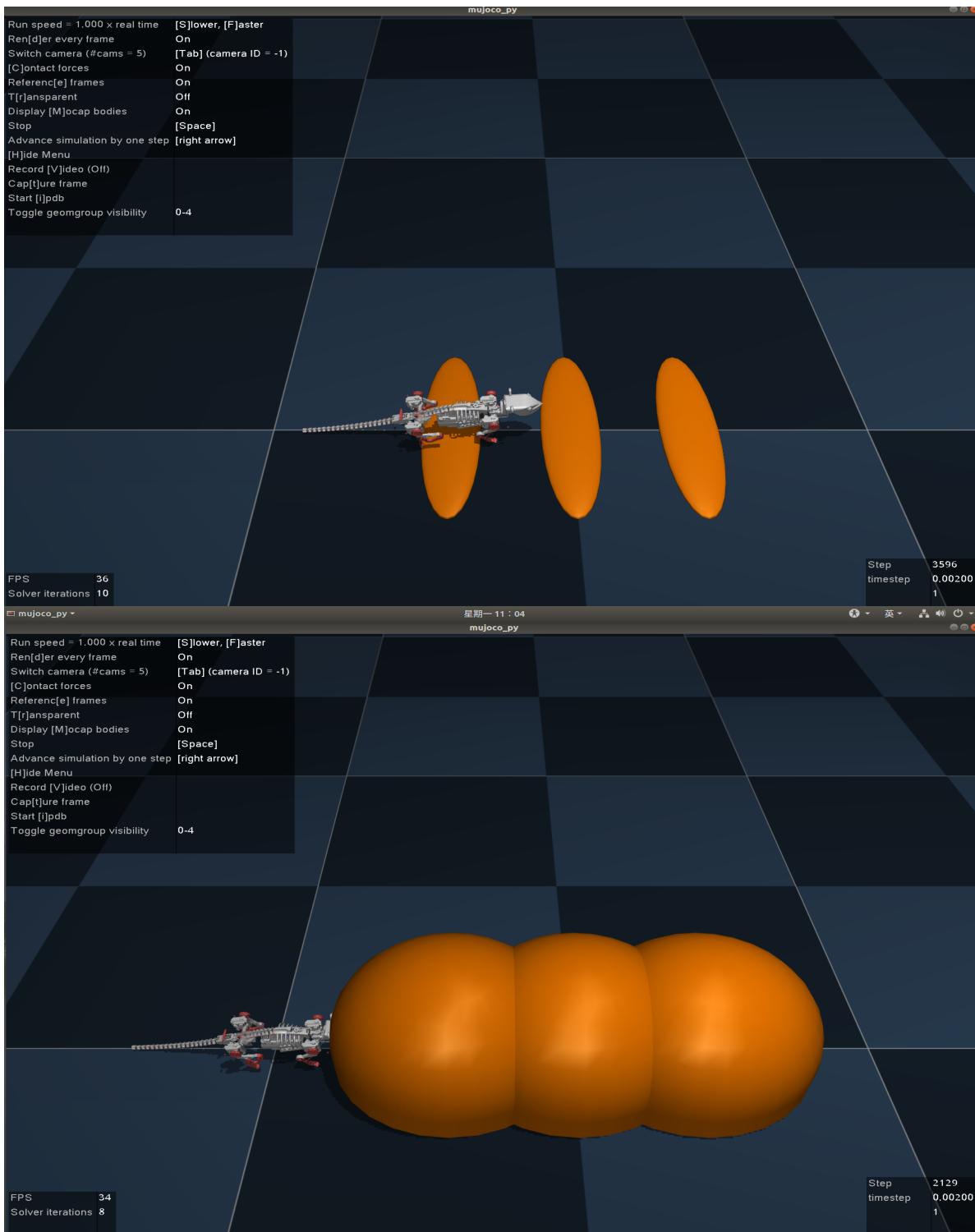
Scene3弧面障碍地形场景和Scene1无法通过的情况类似，都是后腿被障碍卡住。

(3) (Optional) 你能构造更多有意思的场景吗？对于无法通过的场景，你有什么想法能让机器鼠通过吗？

通过修改xml文件中worldbody的geom模块可以对场景进行修改，我使用ellipsoid和capsule构造了两个新的场景椭球形障碍和球形障碍，放在了models文件夹中的scene_test_new1.xml和scene_test_new2.xml文件中，修改的代码部分如下

```
<geom type="ellipsoid" pos="0 -0.3 0" size="0.2 0.05 0.010" rgba="1 0.5 0 1"/>
<geom type="ellipsoid" pos="0 -0.5 0" size="0.2 0.05 0.010" rgba="1 0.5 0 1"/>
<geom type="ellipsoid" pos="0 -0.7 0" size="0.2 0.05 0.010" rgba="1 0.5 0 1"/>
```

```
<geom type="capsule" pos="0 -0.3 0" size="0.2 0.05 0.010" rgba="1 0.5 0 1"/>
<geom type="capsule" pos="0 -0.5 0" size="0.2 0.05 0.010" rgba="1 0.5 0 1"/>
<geom type="capsule" pos="0 -0.7 0" size="0.2 0.05 0.010" rgba="1 0.5 0 1"/>
```



实验发现机器鼠无法通过扁椭球障碍的原因和木板类似，都是后腿没有足够的空间抬起；而球形障碍则是由于机器鼠没有提前调整前腿导致的被卡在图中的场景

至于如何通过障碍，我有一个想法是让机器鼠遇到障碍时后退适当距离（或提前检测到障碍，在障碍前留出合适的距离），让机器鼠的腿有足够的空间抬起以跨越障碍。

2.2运动数据收集

实验任务

(1) Mujoco 仿真环境中，机器人及其环境的模型可以由 XML 文件定义，阅读参考资料中对于 Sensor 的说明，结合 model/dynamic_4l_t3.xml 文件等，写出机器鼠上定义了哪些传感器？

文件中 sensor 模块代码如下：

```
<sensor>
<!-- jointpos: 1D; touch 1D-->
<jointpos name="neck" joint="neck"/>
<jointpos name="head" joint="head"/>
<jointpos name="spine" joint="m_ss"/>
<!--jointpos name="knee_f1" joint="knee1_f1"/>
<jointpos name="knee_fr" joint="knee1_fr"/>
<jointpos name="knee_r1" joint="knee1_r1"/>
<jointpos name="knee_rr" joint="knee1_rr"/-->
<touch name="f1_t1" site="foot_s_f1"/>
<touch name="fr_t1" site="foot_s_fr"/>
<touch name="r1_t1" site="foot_s_r1"/>
<touch name="rr_t1" site="foot_s_rr"/>
<framepos name="com_pos" objtype="site" objname="imu"/>
<framequat name="com_quat" objtype="site" objname="imu"/>
<framelinvel name="com_vel" objtype="site" objname="imu"/>
<accelerometer name="imu_acc" site="imu"/>
<gyro name="imu_gyro" site="imu"/>
</sensor>
```

此外，在腿部和尾部的 xml 文件中也有传感器定义，这里腿部以左前腿为例

```
<sensor>
<!-- tendonpos name="tendon_length" tendon="m2_tendon"/-->
<jointpos name="m1_f1" joint="m1_f1"/>
<jointpos name="m2_f1" joint="m2_f1"/>
<!--actuatorfrc name="m1_F_f1" actuator="m1_f1"/>
<actuatorfrc name="m2_F_f1" actuator="m2_f1"/-->
</sensor>
```

```
<sensor>
<jointpos name="m1_tail" joint="m1_tail"/>
<!--tendonpos name="sensor_m1_tail_1" tendon="m1_tail_tendon1"/>
<tendonpos name="sensor_m1_tail_2" tendon="m1_tail_tendon2"/-->
</sensor>
```

通过参考文档我大致了解了这些传感器的含义：

touch 是触觉传感器，功能是测量一定区域内的直接受力大小（不包括由于软接触导致的力）

accelerometer 是加速度传感器，用于记录空间中三个方向的加速度

gyro 是一个三轴陀螺仪，可以得到三个方向的角速度

jointpos 会创建一个关节位置或角速度传感器并输出一个标量（测试发现标量为 1 维）

framepos 可以返回三维坐标位置

framequat 返回代表对象方向的四维数组

framelinvel 返回三维线速度

(2) 在原代码基础上增加一个数据导出的模块，导出机器鼠的运动数据，包括位置、速度、加速度、角速度。

在参考文档API reference — mujoco-py 1.50.1.0 documentation (openai.github.io)有关PyMjData的属性与方法中，和传感器有关的只有一个sensordata，我在程序中将这一属性输出，发现所有sensor的数据混在一起无法分辨，于是我又查阅了mujoco的源码[GitHub - openai/mujoco-py: MuJoCo is a physics engine for detailed, efficient rigid body simulations with contacts. mujoco-py allows using MuJoCo from Python 3.](#)，也没有发现相关内容，只知道mjData.sensordata的维度为mjModel.nsensordata。后来经过手动测试（删减传感器，观察输出结果），发现sensordata的结果顺序与xml文件中的定义顺序相同。于是我通过以下代码导出所需的数据：

```
self.index=
{"m1_f1":0,"m2_f1":1,"m1_fr":2,"m2_fr":3,"m1_rl":4,"m2_rl":5,"m1_rr":6,"m2_rr":7
,"m1_tai":8,
"neck":9,"head":10,"spine":11,"f1_t1":12,"fr_t1":13,"rl_t1":14,"rr_t1":15,
"com_pos":16,"com_quat":19,"com_vel":23,"imu_acc":26,"imu_gyro":29}#设置
传感器的序号
#存储运动过程的位置、线速度、角速度、加速度
self.pos=[]
self.linvel=[]
self.angvel=[]
self.acc=[ ]
```

首先在初始化函数中定义index方便后续修改，并添加存储位置、线速度、角速度、加速度的列表

```
def get_sensors(self):
    sensors=self.sim.data.sensordata
    pos=sensors[self.index["com_pos"] : self.index["com_pos"]+3].copy() #位置
    linvel=sensors[self.index["com_vel"] : self.index["com_vel"]+3].copy() #线
    speed
    acc=sensors[self.index["imu_acc"] : self.index["imu_acc"]+3].copy() #加速
    degree
    angvel=sensors[self.index["imu_gyro"] : self.index["imu_gyro"]+3].copy() #角速度
    result={'pos':pos,'linvel':linvel,'acc':acc,'angvel':angvel}
    return result
```

之后定义get_sensors函数从sensordata中提取所需的信息

```
result=self.get_sensors()
self.pos.append(result['pos'])
self.linvel.append(result['linvel'])
self.angvel.append(result['angvel'])
self.acc.append(result['acc'])
```

最后在runstep中添加如下代码用于记录每一步传感器的数据

实验问题

在最初get_sensors代码中没有加入copy(), 结果在最后得到的self.pos、self.linvel、self.angvel、self.acc中数据都是相同的。后来得知python即使是切片返回的数据所在地址也是原list地址中的一部分，所以他们的值都随着sensordata的改变而一起更改。最后通过copy方法解决了这一问题

Task3：运动数据分析

实验任务：任选一个【不能正常通过】的实验场景，将其运动数据导出，连同平地上正常通行的运动数据一起进行分析

(1) 相比于平地上的运动数据，不能通过的场景的运动数据不同在哪里？为什么不同？

首先定义记录数据的函数，在机器鼠运动结束后将传感器数据写入文件

```
def write_log(self): #记录传感器数据
    f1 = open("position.txt", mode='w')
    f2 = open("linear velocity.txt", mode='w')
    f3 = open("angular velocity.txt", mode='w')
    f4 = open("acceleration.txt", mode='w')
    f5 = open("log.txt", mode='w')
    for i in range(len(self.pos)):
        f1.writelines(str(self.pos[i])+"\n")
        f2.writelines(str(self.linvel[i])+"\n")
        f3.writelines(str(self.angvel[i])+"\n")
        f4.writelines(str(self.acc[i])+"\n")
        f5.writelines("位置:"+str(self.pos[i]))
        f5.writelines(" 线速度:"+str(self.linvel[i]))
        f5.writelines(" 角速度:"+str(self.angvel[i]))
        f5.writelines(" 加速度:"+str(self.acc[i])+"\n")
    f1.close()
    f2.close()
    f3.close()
    f4.close()
    f5.close()
```

结果已上传到<https://www.aliyundrive.com/s/KXCjMbp6aQk>，记录了基本平地场景与Scene1木板的运动情况，但直接的数据形式非常不直观，我通过如下代码绘制了二者的对比图，分别对比各传感器数据在三个维度的大小关系以及各传感器三维数据做矢量和后的大小关系

```
import matplotlib.pyplot as plt

#target="linear velocity"
#target="angular velocity"
#target="position"
target="acceleration"

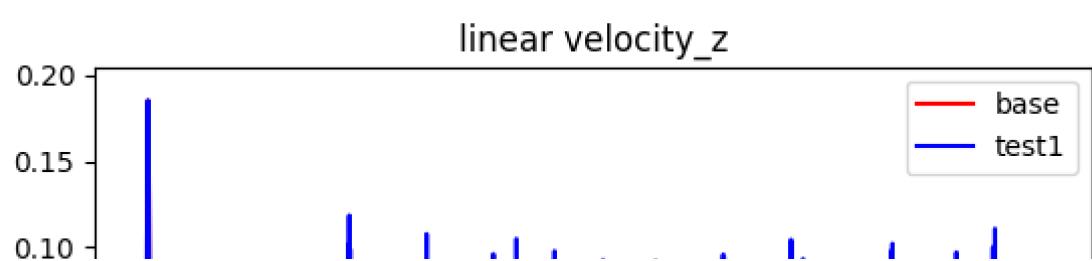
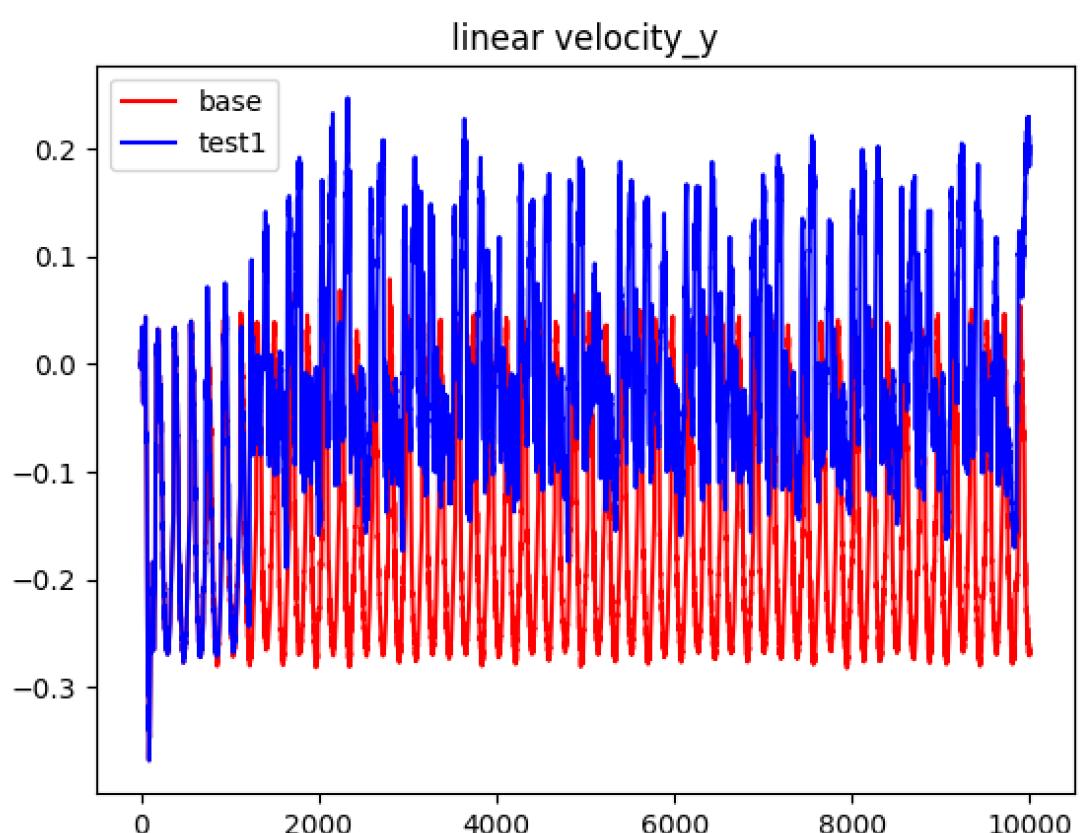
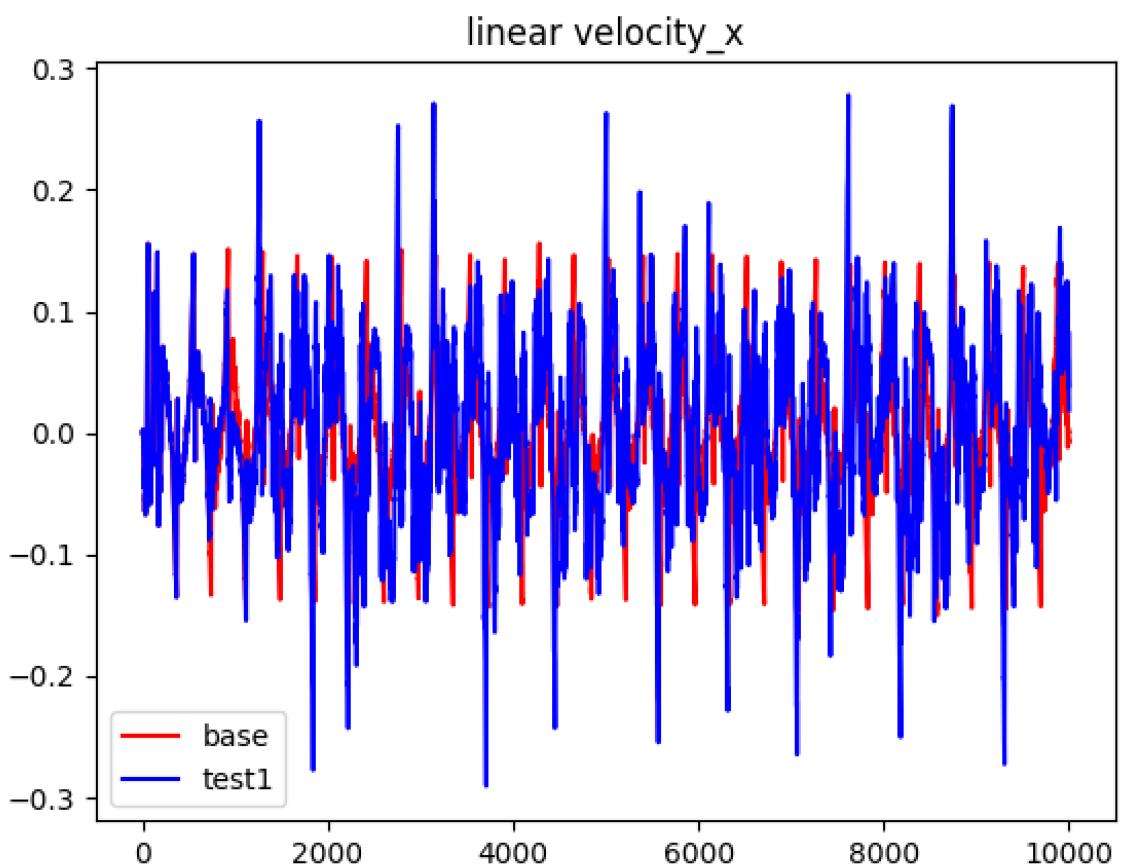
f1=open("./data/scene0/"+target+".txt")
data1=[[[],[],[],[]]]
for line in f1:
    str=line[1:]
    idx=str.index(':')
    str1=str[:idx-1].split()
```

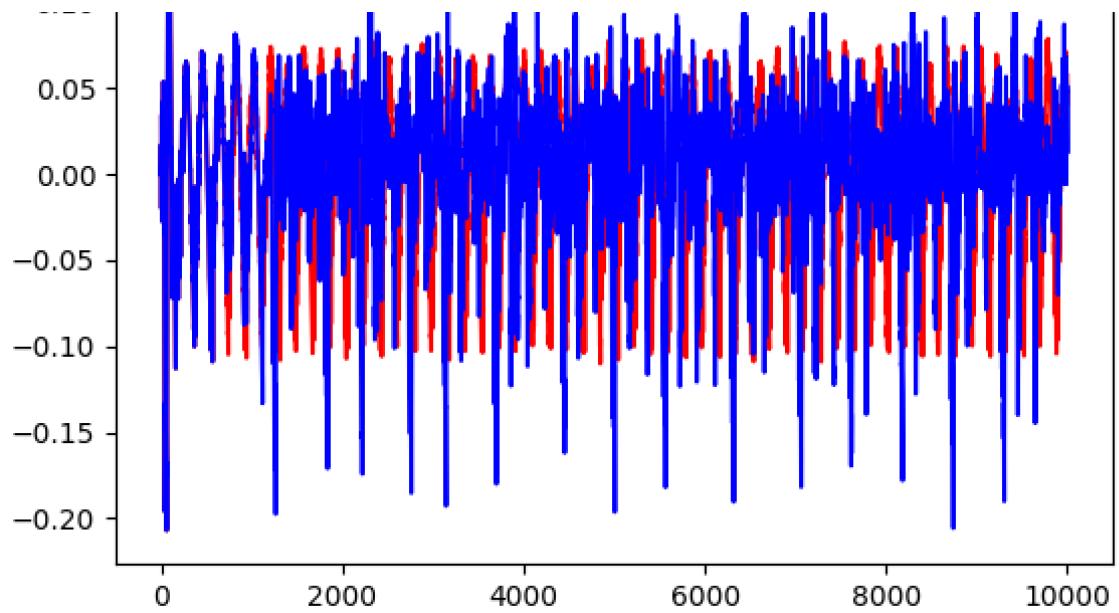
```
str2=str[idx+1:].split()
for i in range(3):
    data1[i].append(eval(str1[i]))
data1[-1].append(eval(str2[0]))
f1.close()

f2=open("./data/scene_test1/"+target+".txt")
data2=[[],[],[],[]]
for line in f2:
    str=line[1:]
    idx=str.index(':')
    str1=str[:idx-1].split()
    str2=str[idx+1:].split()
    for i in range(3):
        data2[i].append(eval(str1[i]))
    data2[-1].append(eval(str2[0]))
f2.close()

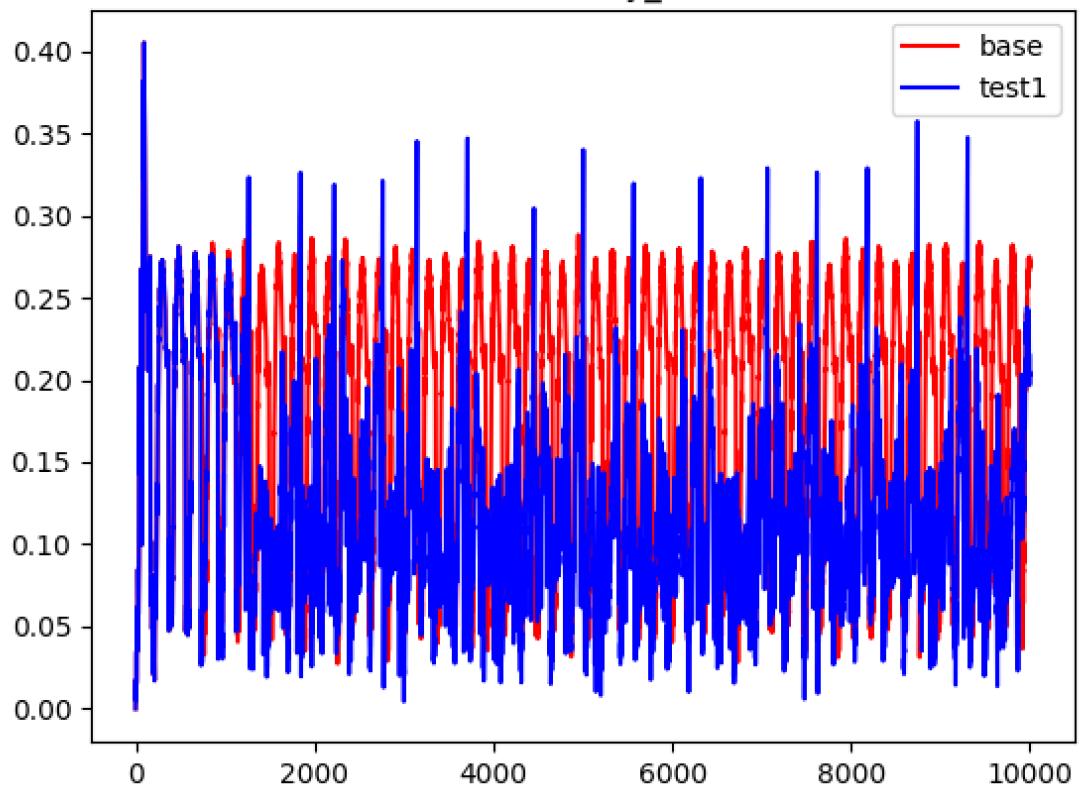
for i in range(4):
    plt.plot(data1[i],'r',label="base")
    plt.plot(data2[i],'b',label="test1")
    if i<3:
        temp=chr(ord('x')+i)
    else:
        temp="value"
    plt.title(target+"_"+temp)
    plt.legend()
    plt.show()
```

线速度



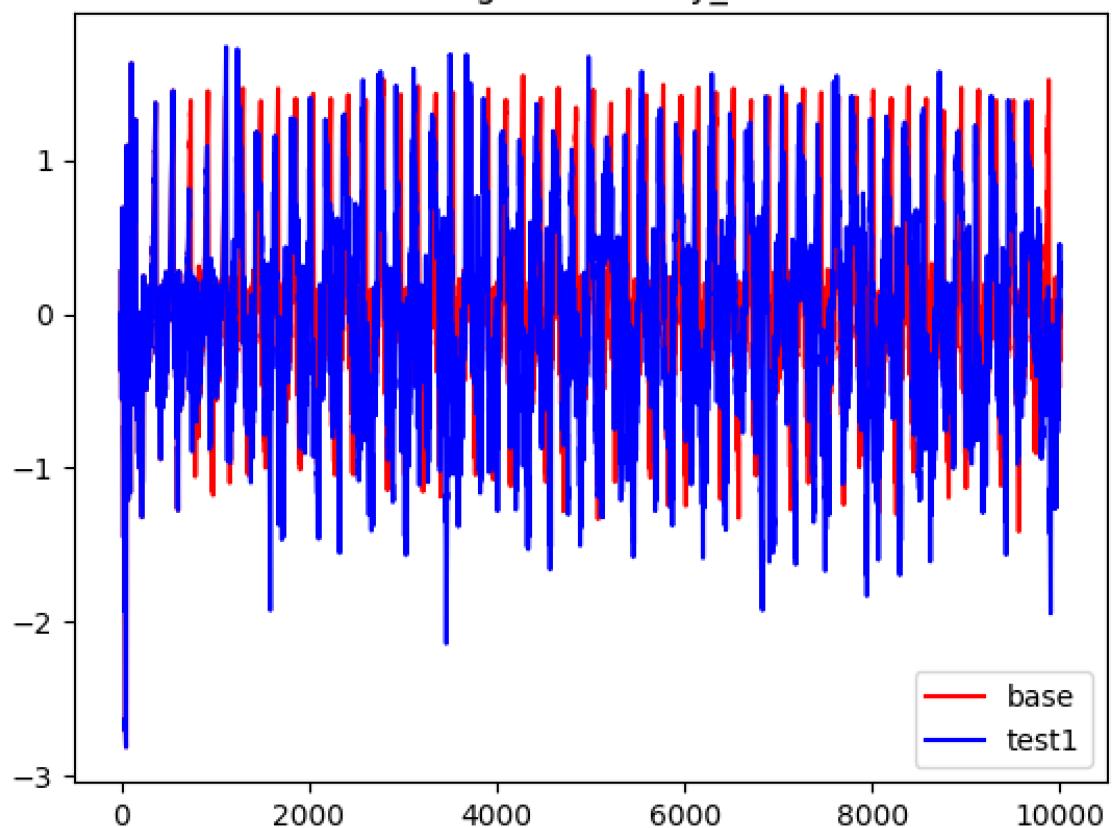


linear velocity_value

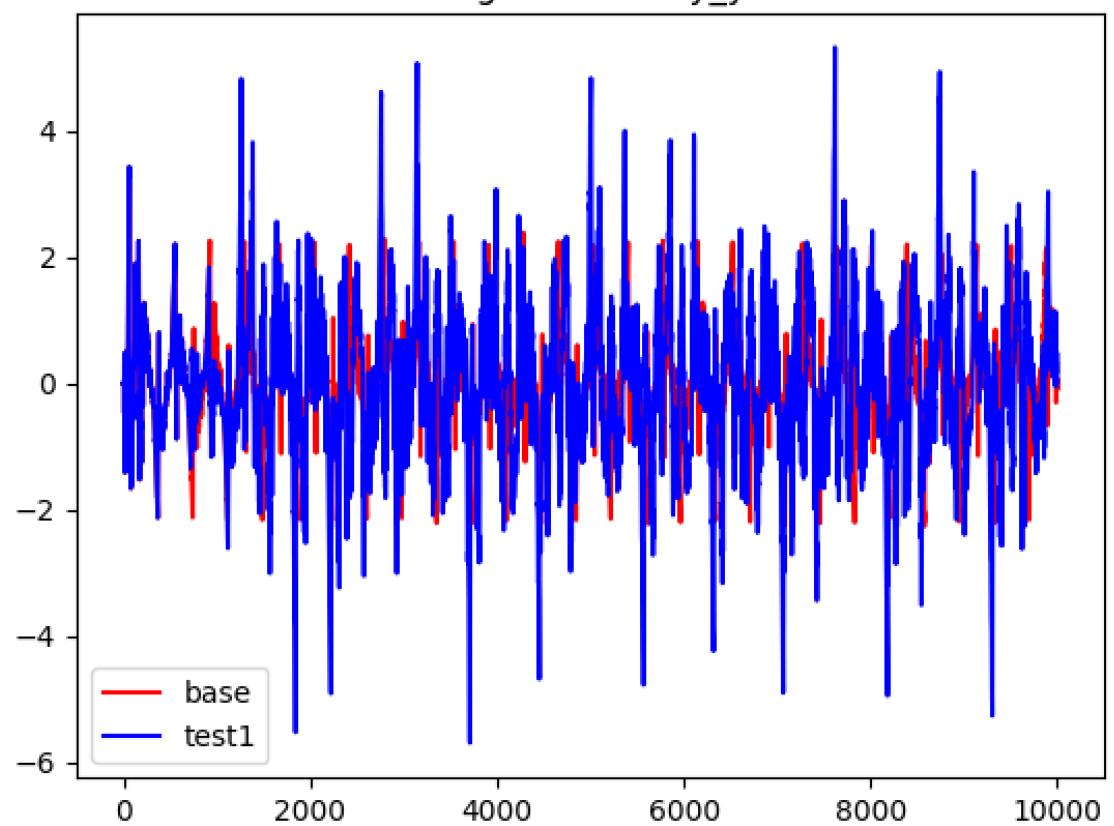


角速度

angular velocity_x

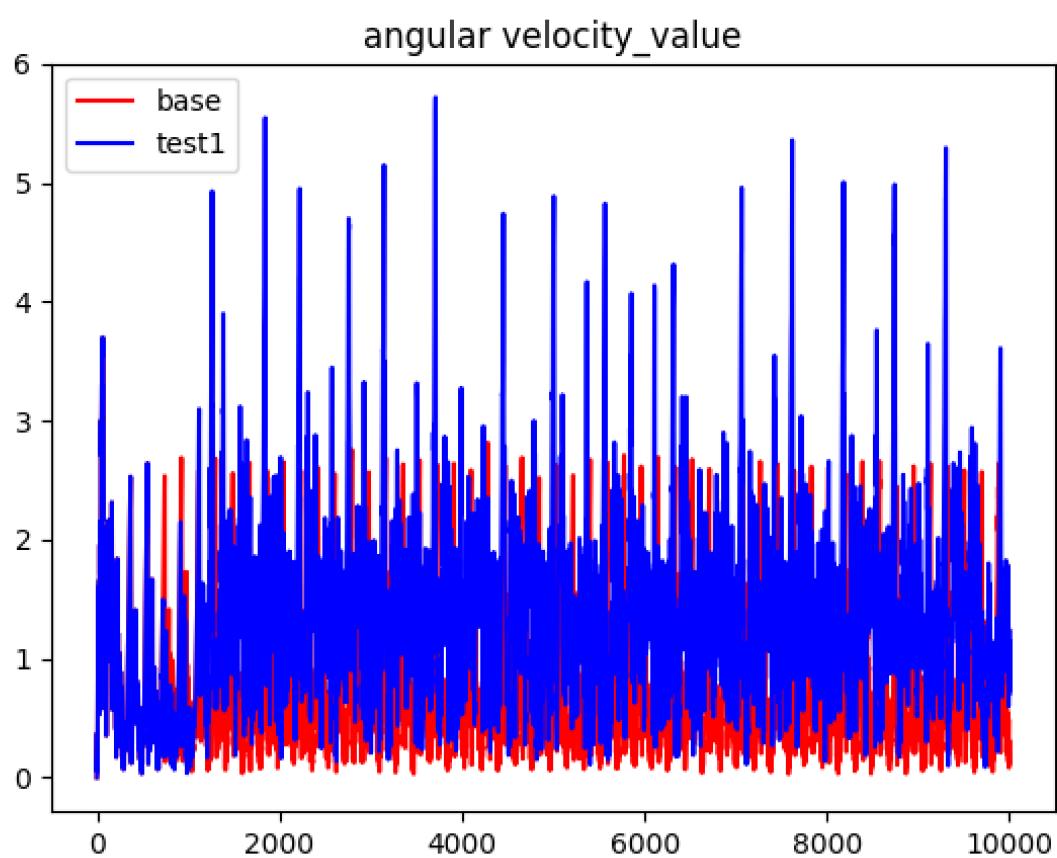
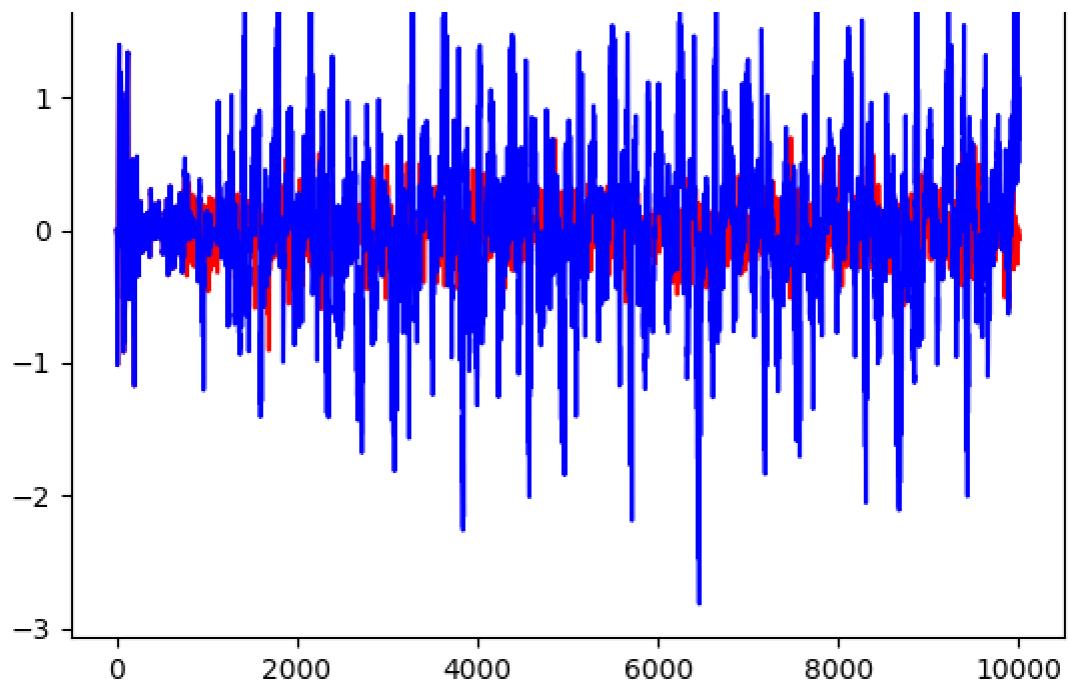


angular velocity_y



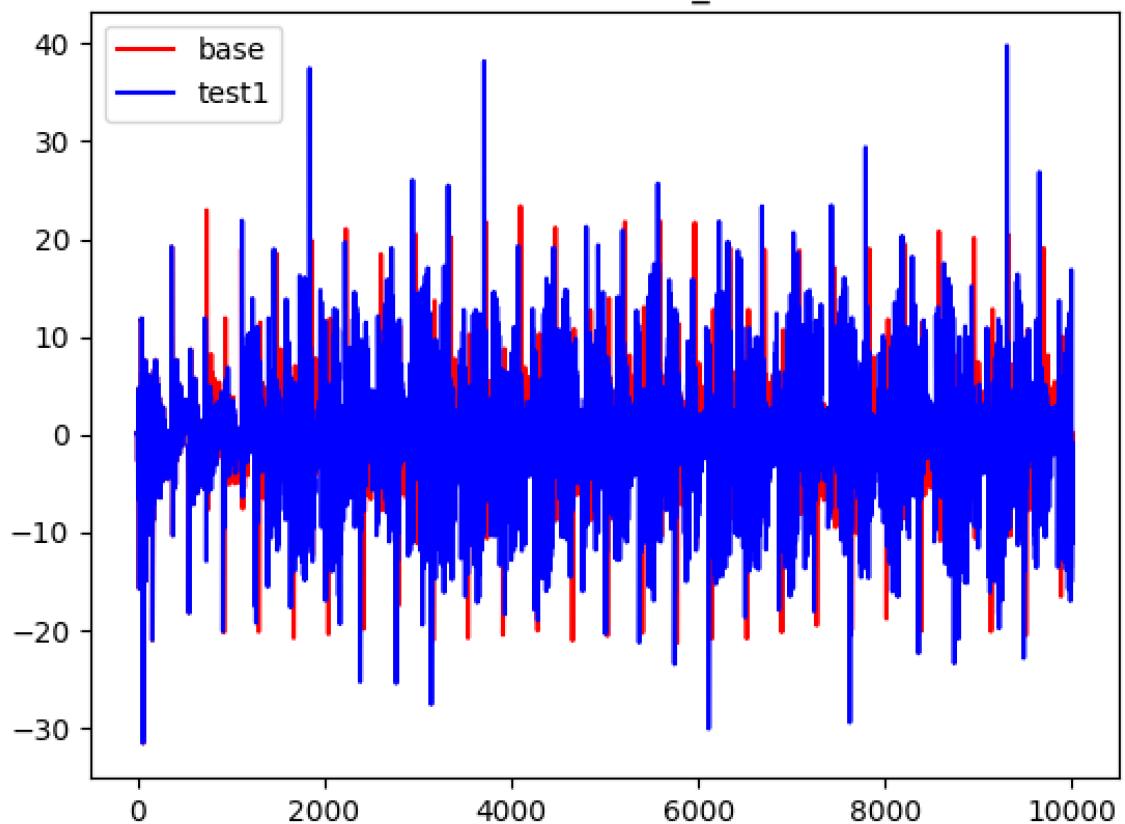
angular velocity_z



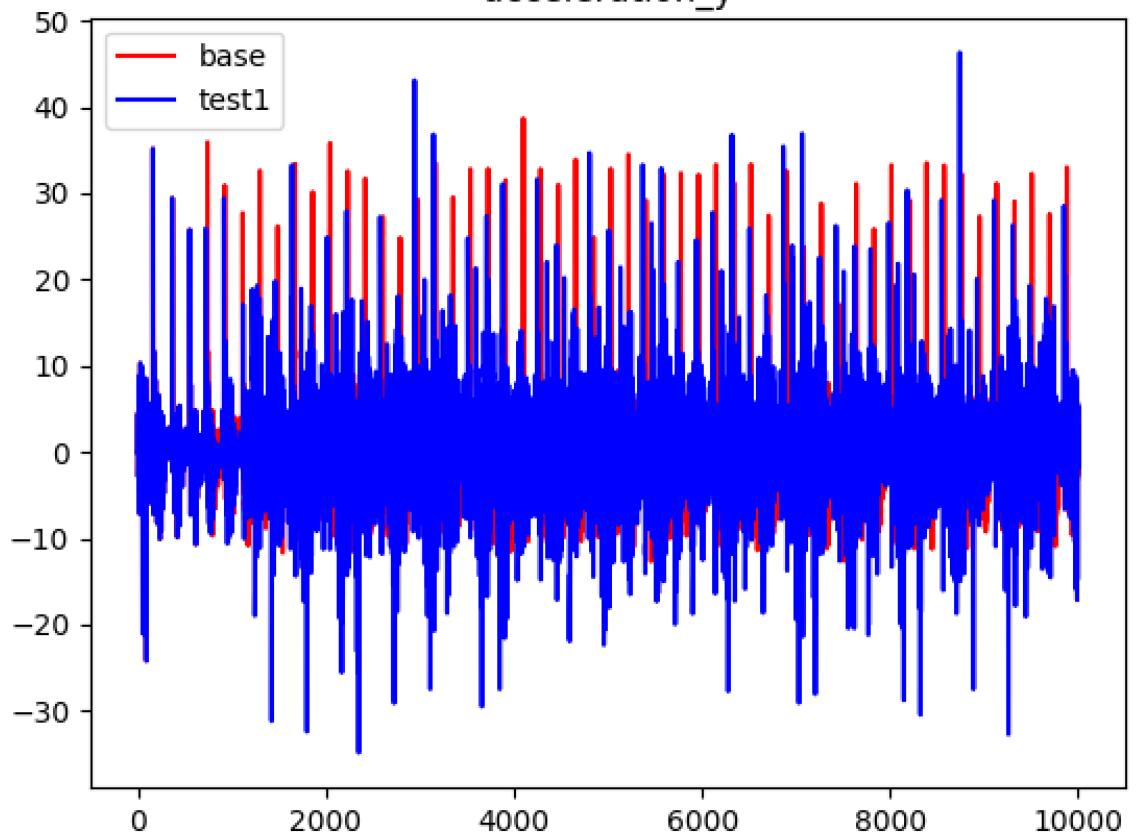


加速度

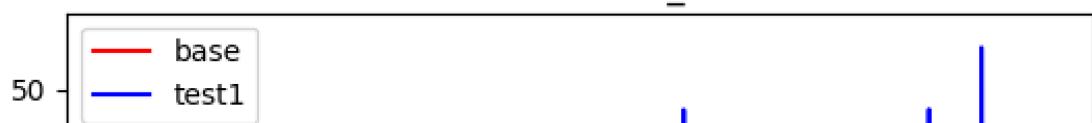
acceleration_x

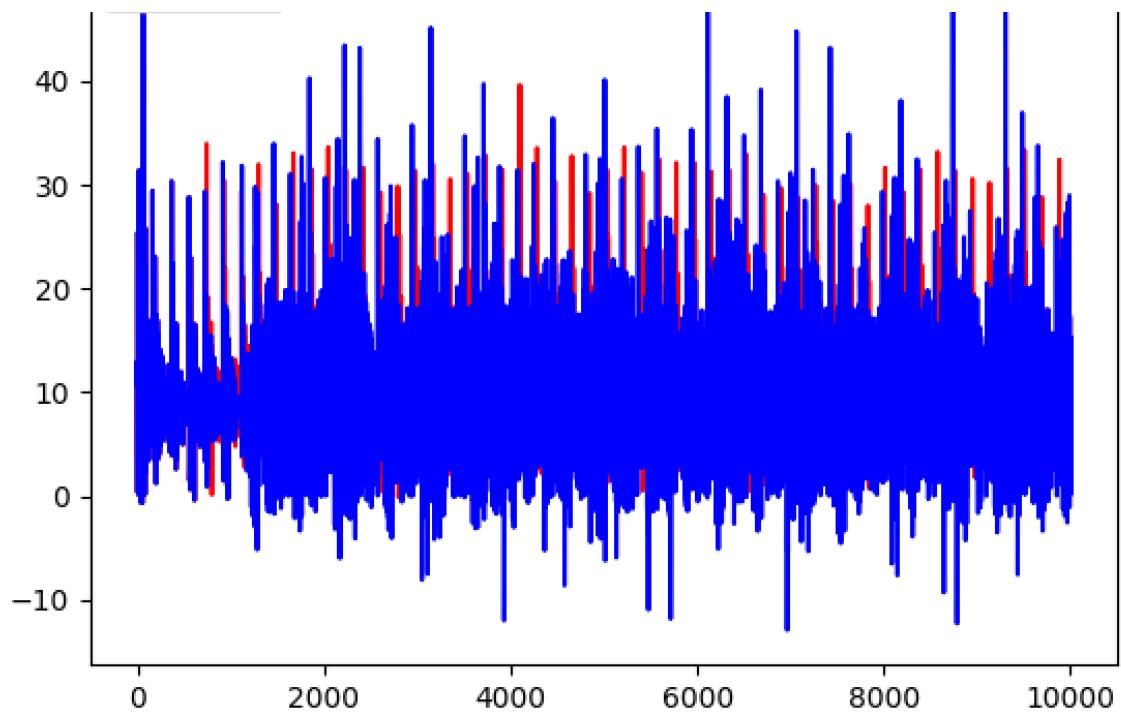


acceleration_y

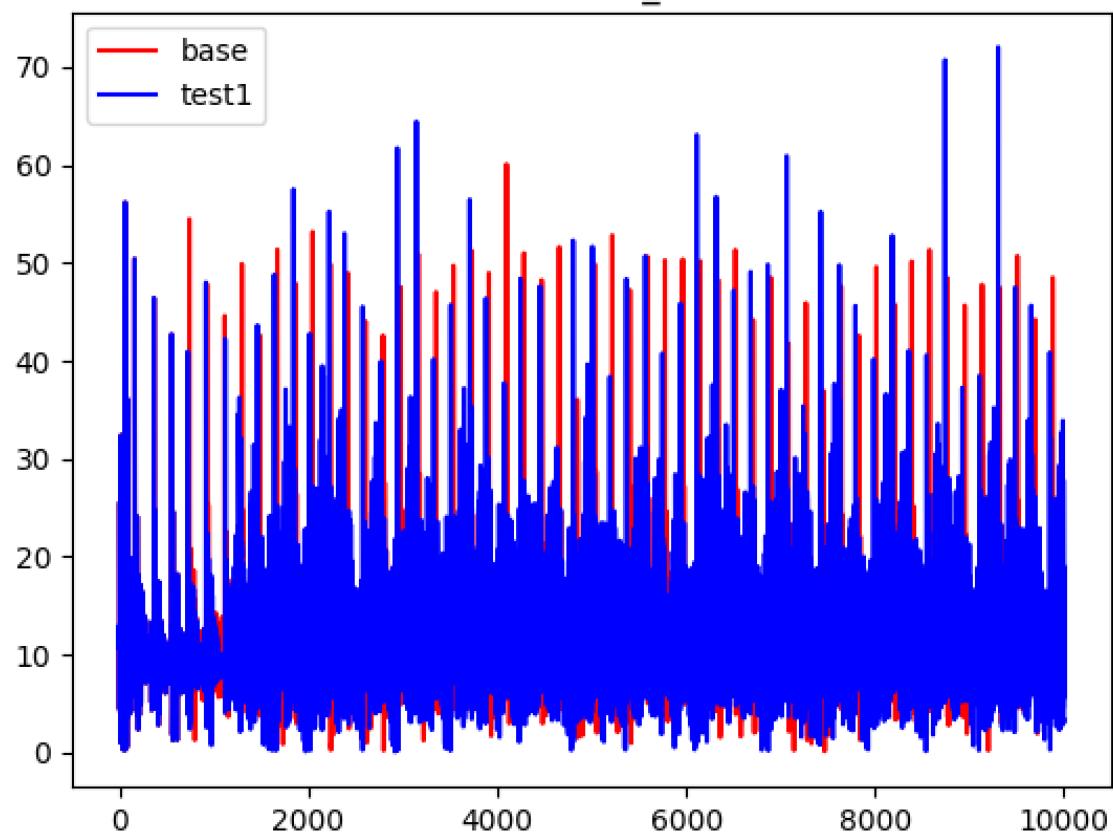


acceleration_z



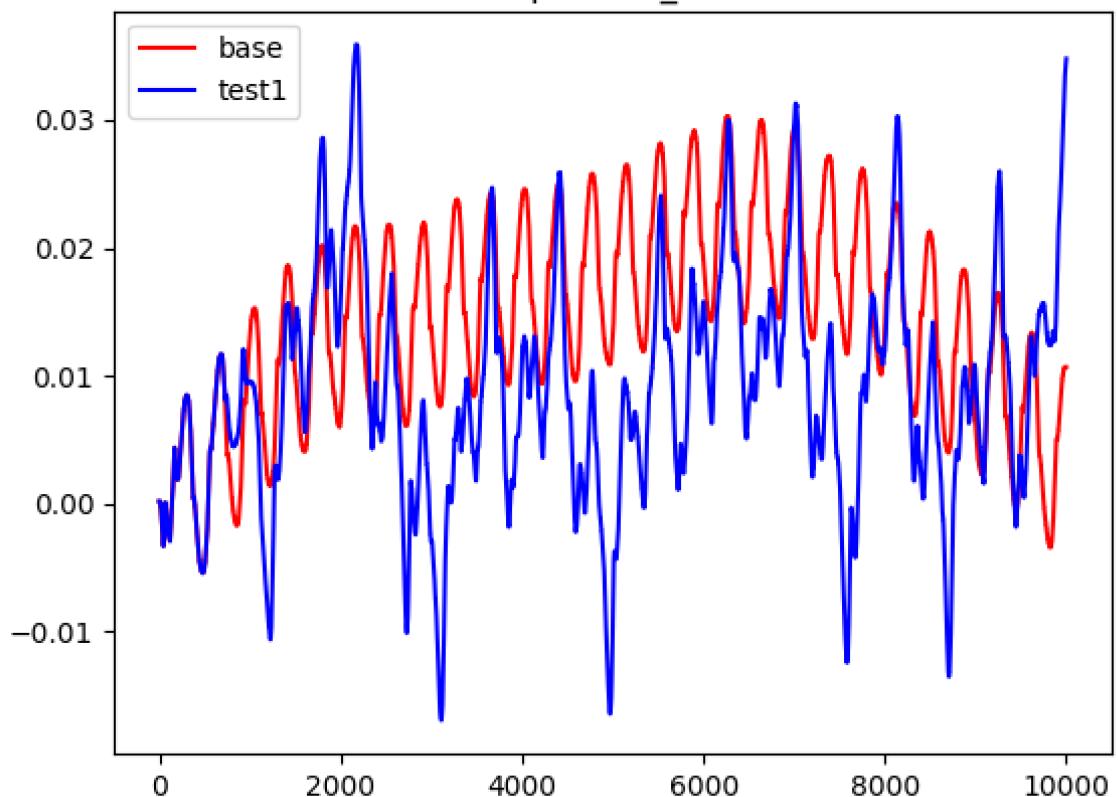


acceleration_value

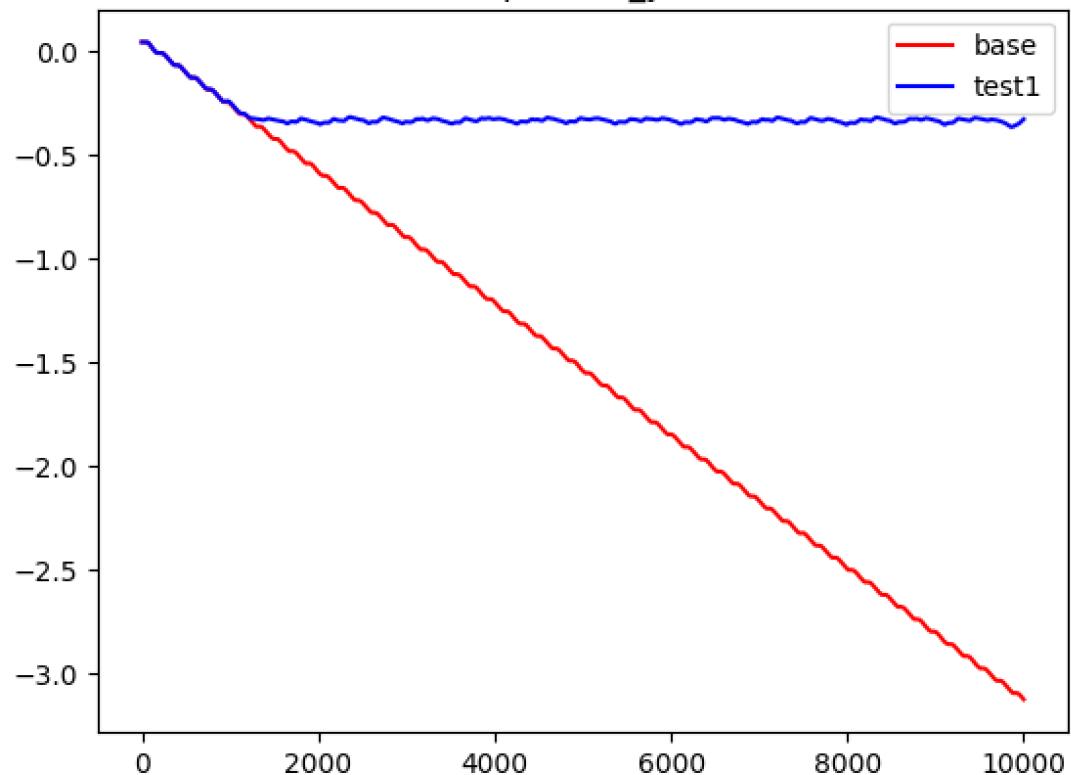


位置

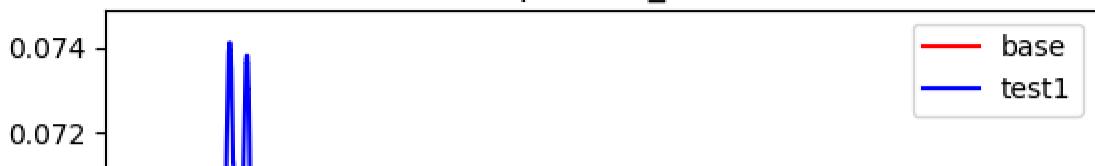
position_x

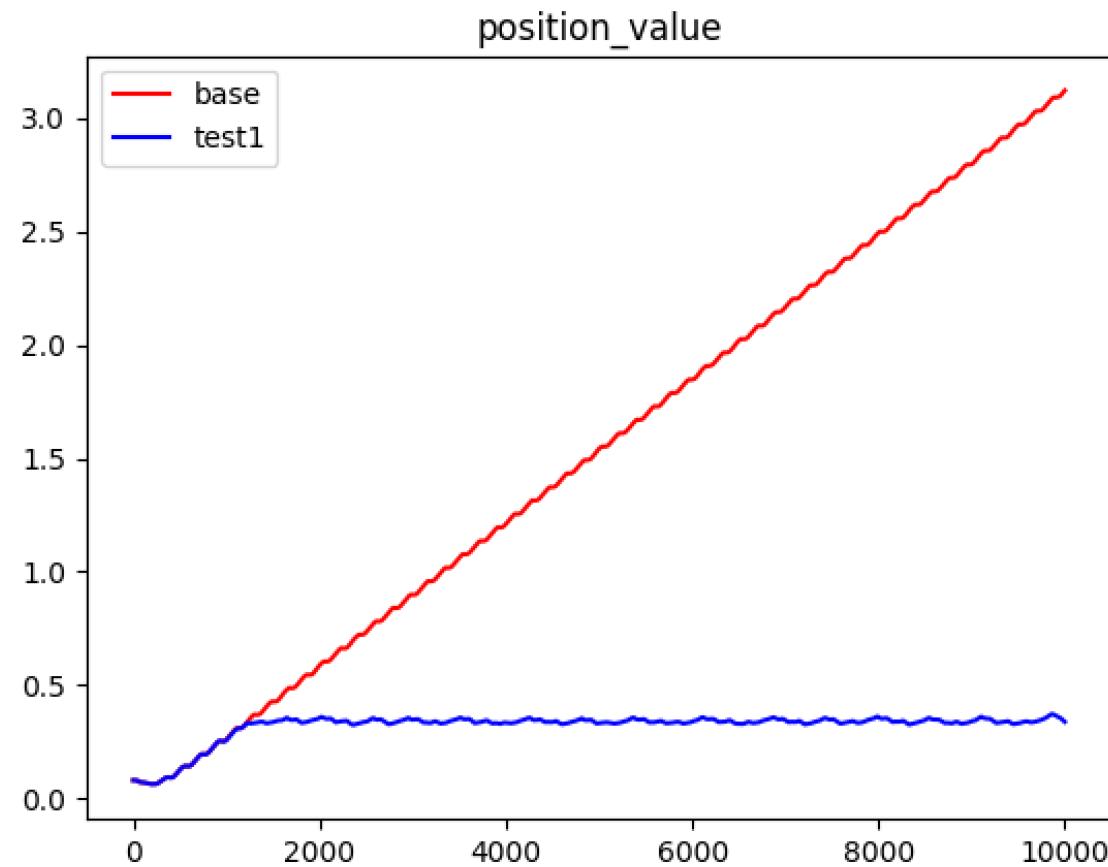
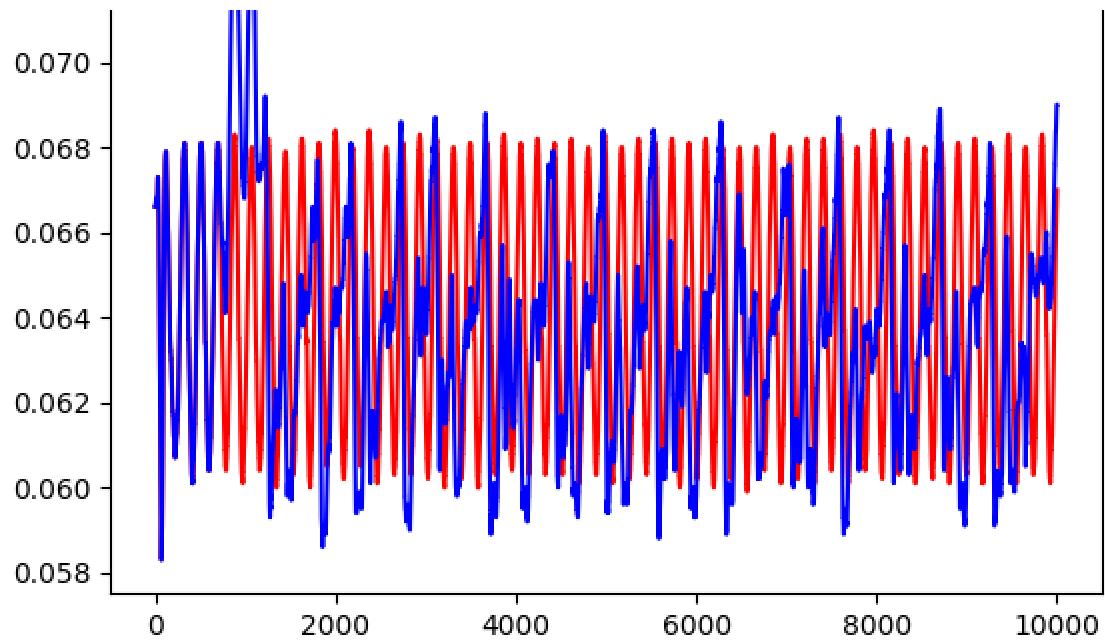


position_y



position_z





通过图像可以观察出，在正常的平地运动中，各项数据指标（除位置）均呈近似周期性的变化，当产生障碍时周期性几乎未发生改变，但是在幅度方面出现了变化，具体如下：

a.在线速度方面：在x、z轴两个方向变化的幅度变大，在y轴方向幅度变化不大但图像整体向上平移。机器鼠原本朝着y轴负方向移动，在y轴方向的线速度图像积分应为负值，在遇到障碍图像平移后积分得到的位移值应该会变为接近0.最后从线速度矢量和上看，遇到障碍后的平均速度会变小，但波动的幅度会变大。

b.在角速度方面：在y、z轴两个方向的角速度变化幅度增加明显、x轴方向略有增加。整体的角速度波动幅度也变大，且均值略有增加。

c.加速度方面则变化不是很明显，各个维度的变动幅度都有所增加，其中延y轴负方向（原机器鼠运动方向）的加速度在每个周期达到的极值下降叫明显（即绝对值变大）。

d.位置的变化和实际运动情况相同，值得注意的是在遇到障碍后z轴方向出现了两个周期的突变，之后又恢复到和正常状态接近的图像。

为了验证上述变化是否具有普遍性，我又导出了其他障碍场景的数据绘图比较。发现由于模型场景卡住的位置并不是平地（如scene2pro）。最后发现在各场景卡住后较为共性的变化有如下几条：

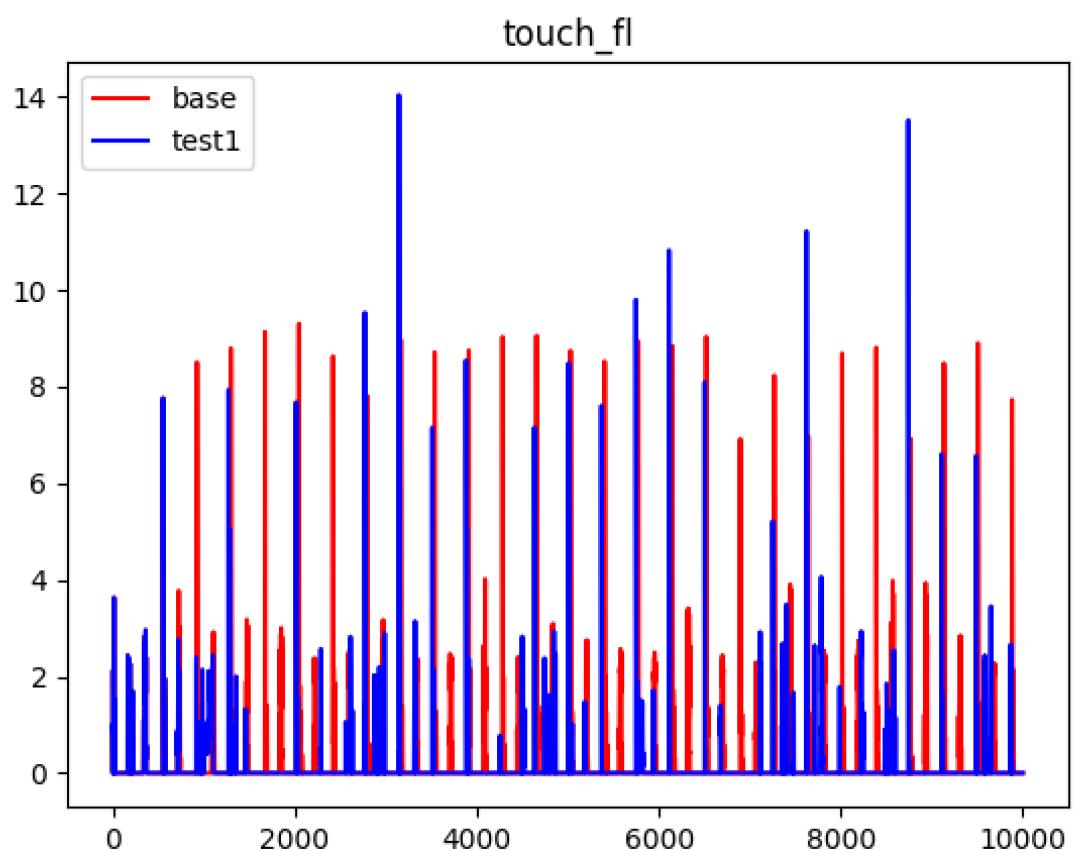
a.线速度y轴方向的图像整体上移

b.平均线速度降低、平均角速度变大

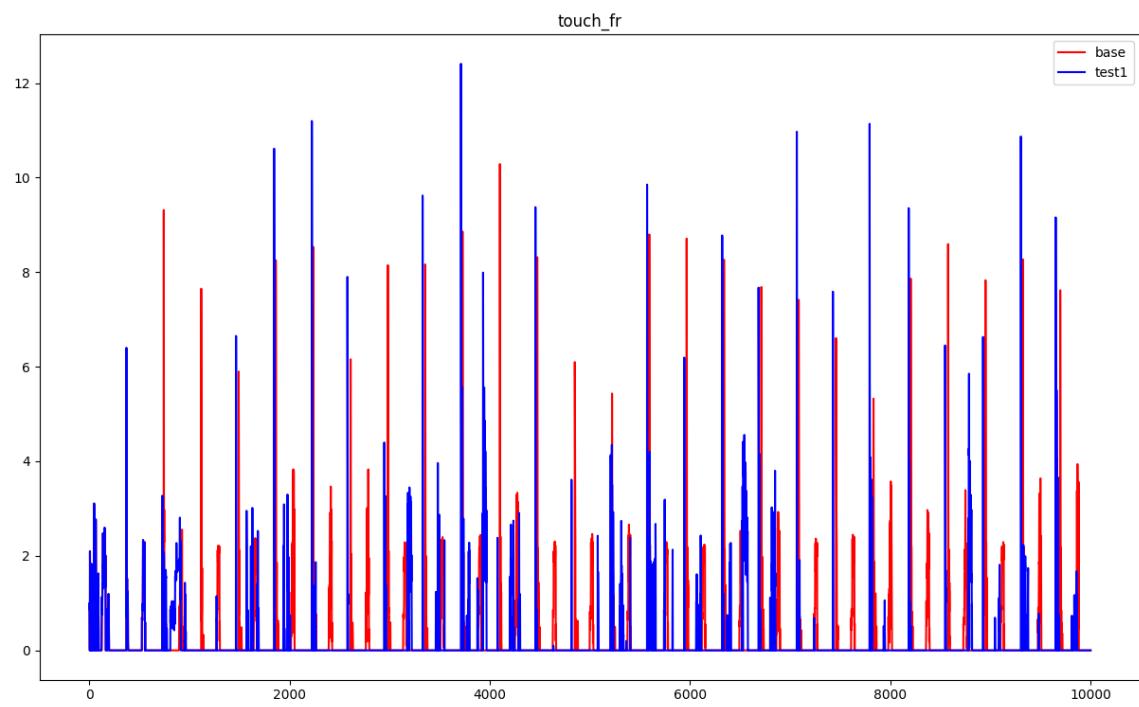
c.加速度、线速度、角速度在各个方向波动幅度变大（但发现scene2斜坡能正常通过的情况，这些数值波动也有所变大）

之后我想到在机器鼠的定义中还有一些touch传感器，我通过代码将他们的数据也进行了导出记录，在scene_test1与平地的对比图如下：

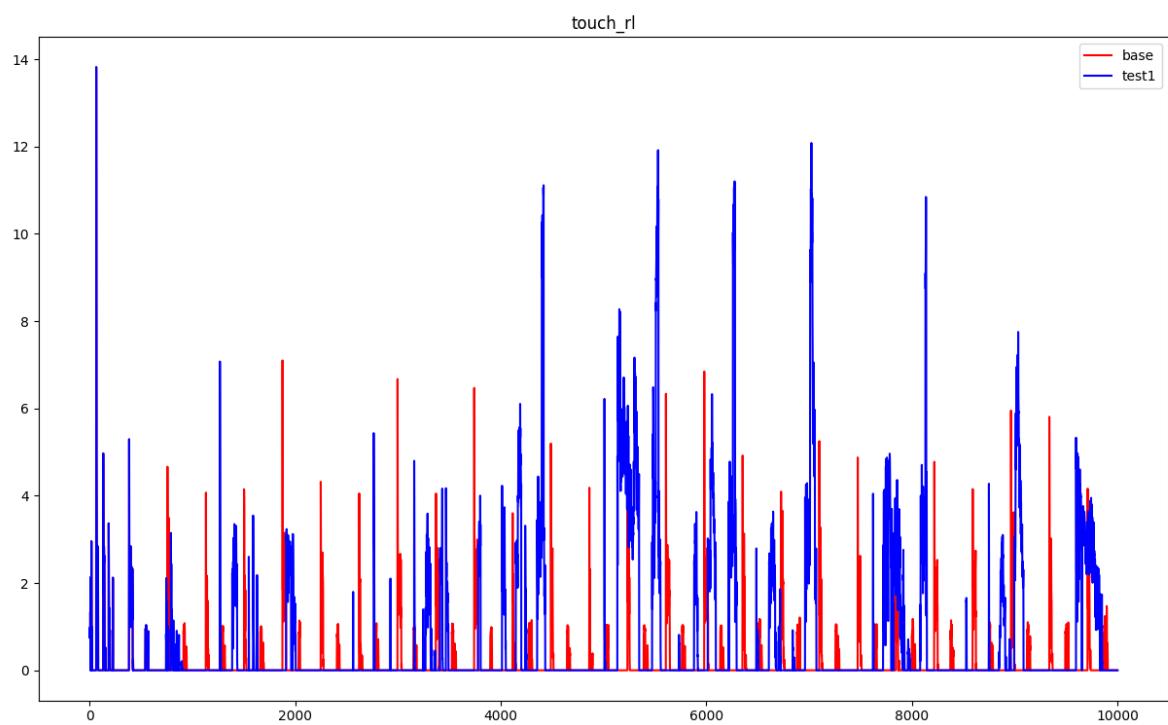
左前腿



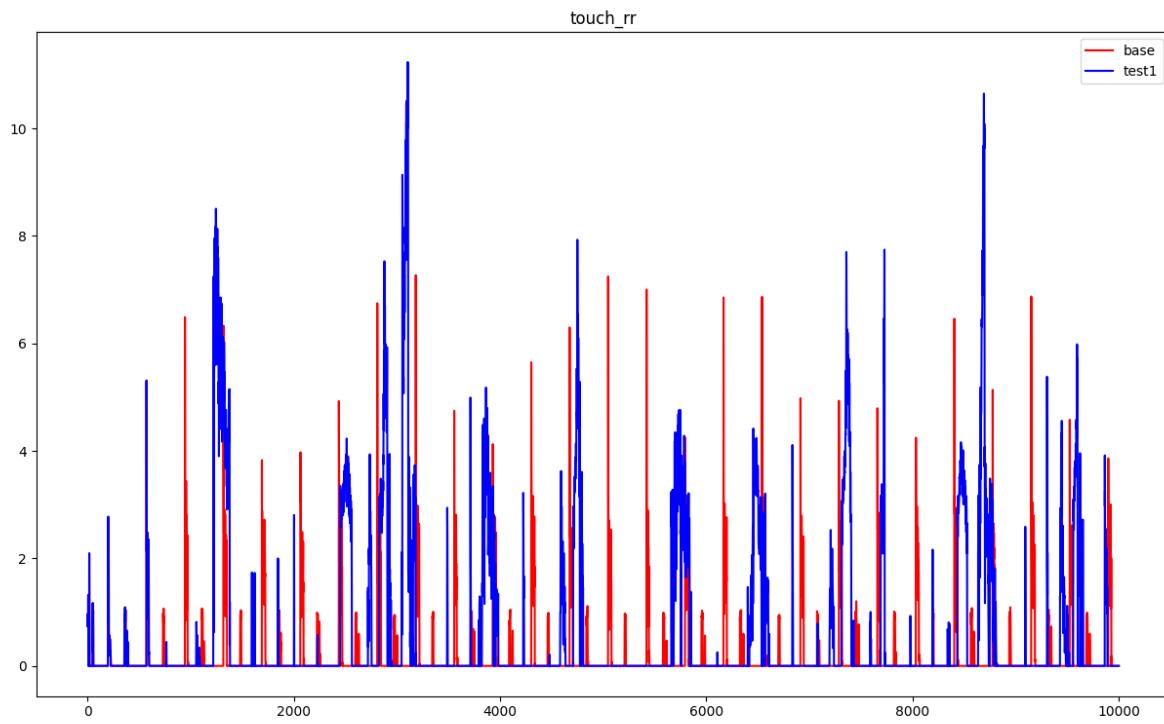
右前腿



左后腿



右后腿



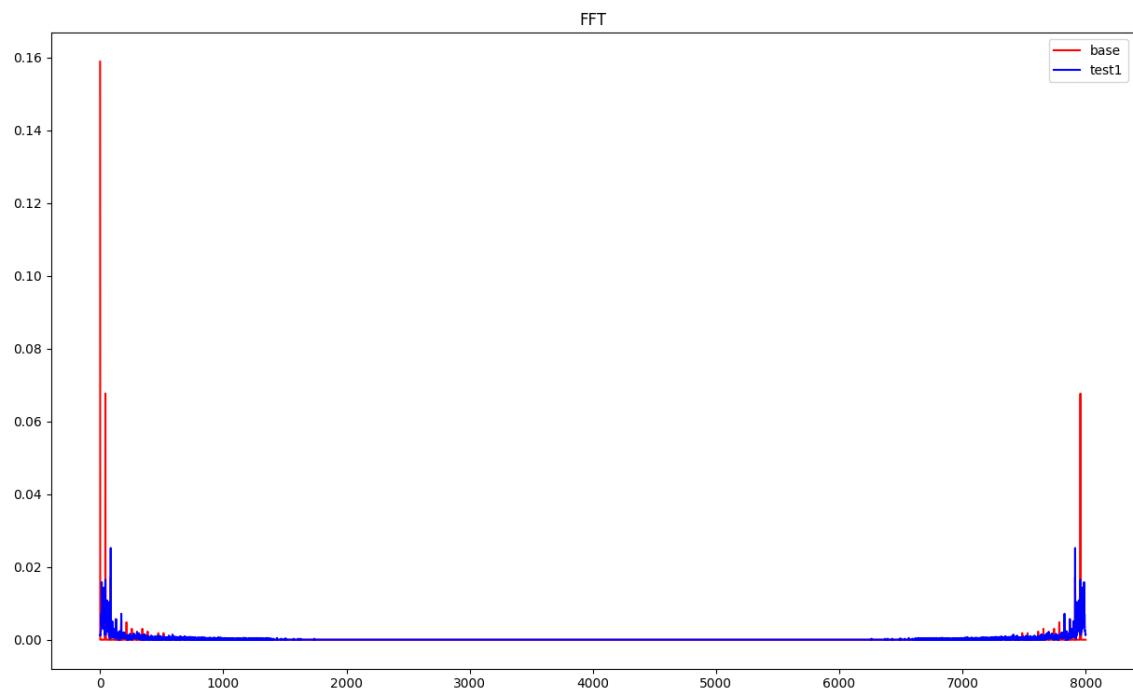
但是综合对比所有障碍场景后，暂时没有找出受力上的规律

(2) 如何用一种数学形式，简洁地表示它们之间的不同？

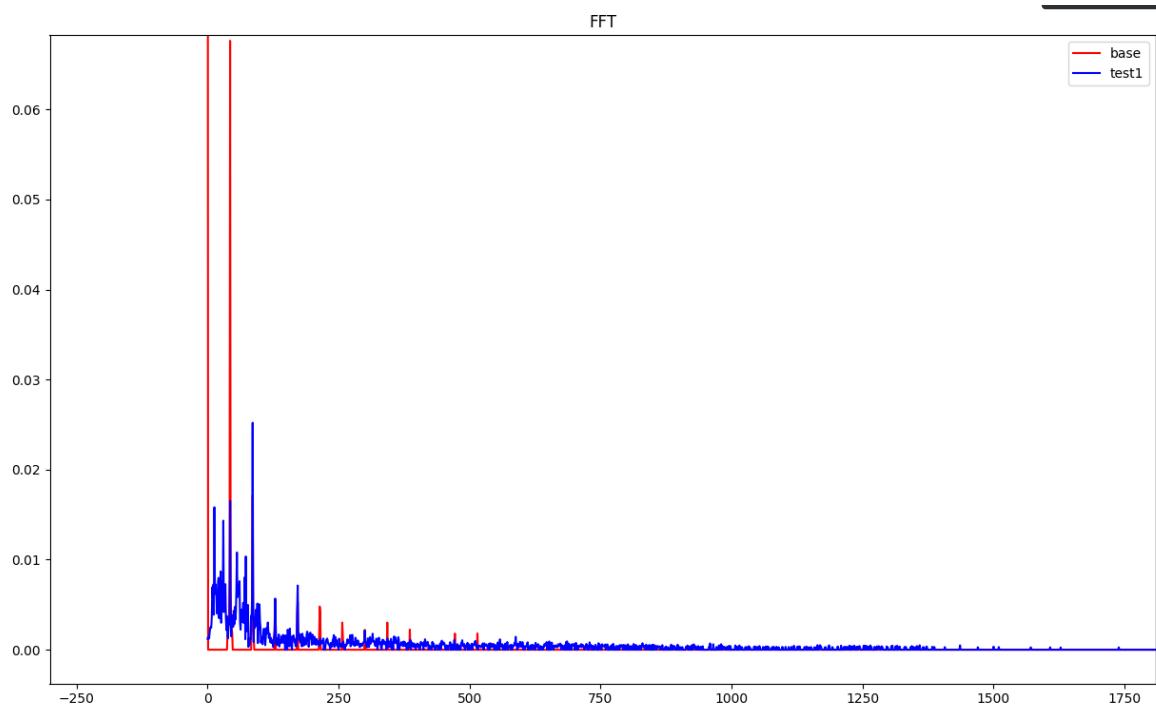
由上面的图片可以看出各个量都具有一定的周期性，所以想到通过离散傅里叶变换对图像进行函数拟合。这里选择变化最大的y轴上的线速度与z轴上的角速度进行数学表示：代码通过numpy.fft的fft函数实现，FFT中求得的x(n)进行图像可视化表示

$$X(m) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi nm/N}$$

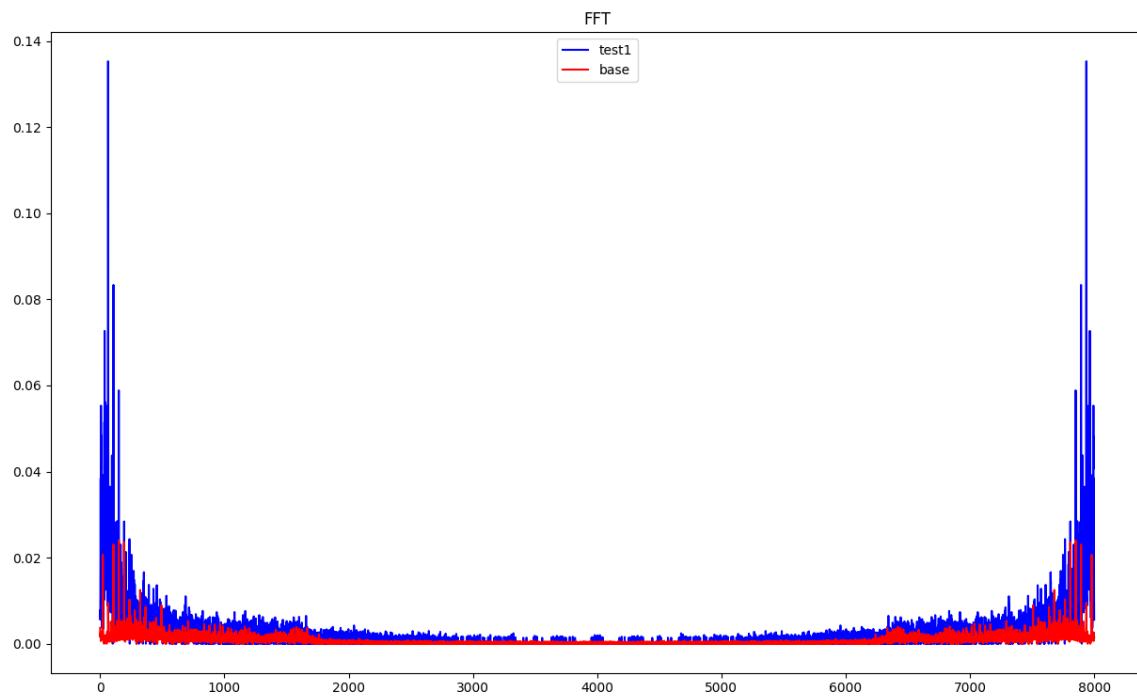
线速度y轴方向：



局部放大图如下



角速度z轴方向：



虽然角速度的周期性不如线速度明显，但明显可以看出蓝线对应的障碍情况在z轴方向角速度的幅值时大于平地无障碍状况的

(3) 考虑一种“强化学习”的框架，机器人通过持续性的数据反馈学习自己的行为，这种连续性要求数据的表征不仅需要简洁，还要考虑计算量。

要构建强化学习框架，首先要定义状态空间S、动作空间A、奖励值R，之后要定义决策方法 π 、需要的其他函数如动作函数Q、值函数V等

a.首先状态空间S可以使用传感器数据sensordata（可以只选择线速度、角速度、加速度，也可以加入受力传感器数据观察这类数据是否对结果有影响）以及当前的控制数据ctrlidata定义。由于状态空间各维度的值都是连续的，所以可以采用神经网络来计算值函数V

b.动作空间A可以考虑三种方法：

(1) 第一种是连续的动作空间，每次的动作都是 $\Delta ctrlidata$ ，采取动作后直接更新 $ctrlidata \leftarrow ctrlidata + \Delta ctrlidata$ 。策略 π 就可以考虑使用策略学习的方法获得。和之前的值函数学习结合在一起即 Actor-Critic Method

(2) 第二种方法是离散的动作空间，即预先设置好一些 $\Delta ctrlidata$ ，仍采取 $ctrlidata \leftarrow ctrlidata + \Delta ctrlidata$ 的方法更新状态

(3) 第三种方法也是离散的动作空间，不同的是此时的A是预先设定好的一些运动策略（可以根据现有的的一些障碍情况，通过数学建模设计一些运动方案）。当采取某运动策略后，就按照动作 a_i 运动n步。(2)(3)的动作空间离散、状态空间连续，可以考虑使用DQN及其变体

c.最后要考虑的是对奖励值R的定义，可以用计算机器鼠每一步在目标方向（y轴负方向）前进的距离来定义

Task4：进阶设计及工程

实验任务

设计一种基于探索的强化学习框架，克服无法通过的地形场景