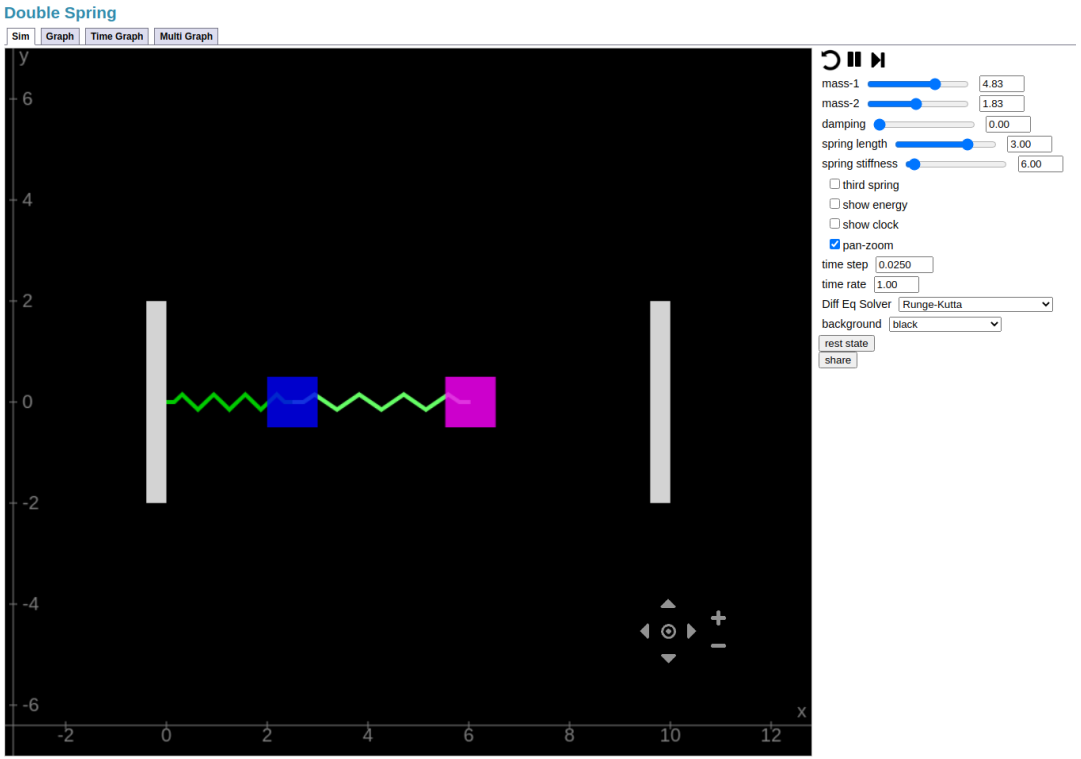
Physics simulator

2021/10/13 ray wang

What is Pysics simulator?

- 物理模擬器主要的目的是用虛擬環境來模擬真實的環境，讓虛擬的物體在虛擬的空間中模擬真實的運動。
  （ https://www.myphysicslab.com/ ）



Behavior-base robotics

- 使用者可以自訂空間,物件,光線，並且可以透過程式來控制物件讓物件與環境互動，可以有回饋。通常simulator 會再加上physics engines 讓整體模擬更貼近實際狀況。

常見的物理模擬器

- Gazebo：可以進行碰撞偵測，主要是建立在ROS上面且沒有python api，與pybullet比就沒有那麼好用，只能透過ROS topic進行資料的獲取。
- Webots：使用者付費（半開源）
- V-REP：使用者付費（教育版免費）
- Pybullet ：號稱是專門給neural network的模擬器，底下的物理引擎預設使用Bullet，可以替換成別的。可以進行碰撞偵測，有python的API可以直接使用，支援URDF、SDF和MJDF三大类模型文件。
- PyRep / RL Bench ： 原本設計出來就與RL無關，較老的虛擬環境模擬器，基於V-REP開發出來的，主要是模擬機器手臂，效率差編程難
- (https://zhuanlan.zhihu.com/p/300541709) <比較不同模擬器的優劣>

常見的物理引擎

- 常見的物理引擎（physics engines）分成兩大類：（https://www.guyuehome.com/8218）
  - 專門給遊戲仿真使用，要的是表面擬真並且資源消耗小
  - 專門給高精確度仿真使用，著重再表面材質或是各種物理特性的表現，精確計算

常見的物理引擎

- 有很多不同種的物理引擎（http://element-ui.cn/python/show-137945.aspx）
  - ODE：開源物理引擎，全稱Open Dynamics Engine，它是一款模擬剛體運動力學的基于C/C++高性能庫，功能稳定，常被用於遊戲和虛擬現實技術；
  - Bullet：開源物理引擎，世界三大物理模擬引擎之一，由C++編寫，與ODE相同，被廣泛應用與遊戲開發、電影製作作中
  - Simbody：開源物理引擎，由C++编寫，為多體運動力學模擬的高性能庫
  - DART：開源物理引擎，全稱Dynamic Animation and Robotics Toolkit，以准确性和稳定性著稱；
  - Newton：開源物理引擎，精确的3D物理庫，用于物理環境的即時仿真；
  - Vortex：商業物理引擎，由CM-Labs開發，計算精度上可以說是上面介绍这幾个物理引擎中的佼佼者，由于这是一款商業物理引擎，因此教育版的V-REP只能用来仿真20秒的過程。

什麼事機器人描述檔

- 都是XML格式的文件
- URDF
  - 在ROS中是一種功能強大且標準化的機器人描述格式，但依然缺少許多功能
- SDF
  - SDF完整描述了從世界級到機器人級的所有內容。它具有良好的可擴展性。專門給Gazebo使用來改善URDF的不足
- MJDF
  - OpenAI Gym & Deepmind control suite（Mujoco）

<https://zhuanlan.zhihu.com/p/129660153>比較URDF SDF的差別

Pybullet 安裝

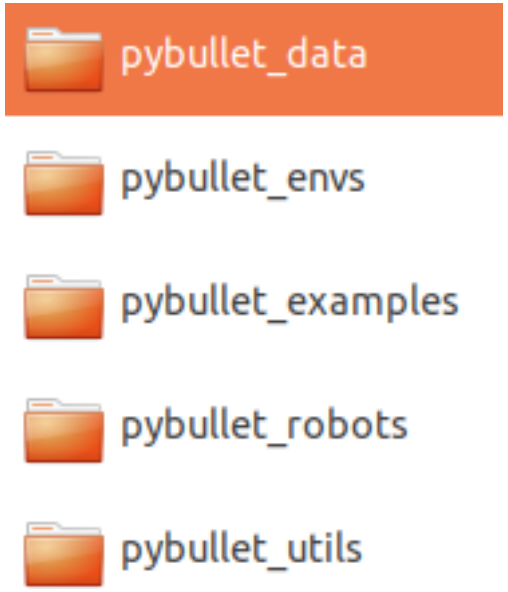
- pip install pybullet
- pip install gym #很多套件需要用到這個所以先裝起來

特別注意：如果有install gym 但是執行時找不到module，需要在init.py的地方加入

```
import sys
sys.path.append('location found above')
```

學習網站
https://zhuanlan.zhihu.com/p/347177629

Pybullet 初步使用



安裝完後會有5個新的資料夾

pybullet_data裏面放了很多URDF文件可以使用
pybullet_env裏面放了很多現成的環境
pybullet_robots裏面放了很多現成的機器手臂(kuka)

基本程式

```
p.disconnect()
```

```
physicsCilent = p.connect(p.GUI) #分成有GUI模式跟沒有GUI模式，後者執行比較快  p.DIRECT
print('physicsCilent = ',p.getConnectionInfo(physicsCilent))

# pybullet.GUI {'isConnected': 1, 'connectionMethod': 1} or pybullet.DIRECT  {'isConnected': 1, 'connectionMethod': 2}
# 'isConnected' = ID of client

# 添加將要使用的URDF檔案的路徑
p.setAdditionalSearchPath(pybullet_data.getDataPath())

# 设置环境重力加速度
p.setGravity(0, 0, -10)
```

Connect = 連結物理引擎,可以設定開啟GUI模式或是隱藏模式（結束時要斷開連結）
GetConnectionInfo = 模擬器是否有連線,是否有連線到GUI
SetADDitionalSearchPath = 將需要用的資料的位置讀入程式中，後面可以直接取用資料夾中的檔案
SetGracity = 設置重力

基本程式

```
# 加载URDF模型，此处是加载蓝白相间的陆地
planeId = p.loadURDF("plane.urdf")

# # 加载机器人，并设置加载的机器人的位姿
startPos = [0, 0, 5]
startOrientation = p.getQuaternionFromEuler([0, 0, 0])
boxId = p.loadURDF("r2d2.urdf", startPos, startOrientation)

# 按照位置和朝向重置机器人的位姿，由于我们之前已经初始化了机器人，所以此处加不加这句话没什么影响
# p.resetBasePositionAndOrientation(boxId, startPos, startOrientation)

p.configureDebugVisualizer(p.COV_ENABLE_RENDERING, 0)
p.configureDebugVisualizer(p.COV_ENABLE_RENDERING, 1)

# 讓CPU不會參與影像渲染工作
p.configureDebugVisualizer(p.COV_ENABLE_TINY_RENDERER, 0)
# # 輸入物件ID來獲取位置與方向四元数
cubePos, cubeOrn = p.getBasePositionAndOrientation(boxId)
```

LoadURDF = 加載URDF，後面可以另外設定起始位置以及旋轉角度
p.configureDebugVisualizer(p.COV_ENABLE_RENDERING, 0) = 在場景還沒加載完之前不會進行渲染
p.configureDebugVisualizer(p.COV_ENABLE_RENDERING, 1) = 開始渲染
p.configureDebugVisualizer(p.COV_ENABLE_TINY_RENDERER, 0) = 讓CPU不會參與影像渲染工作
getBasePositionAndOrientation(boxID) = 可以得到特定物件的位置以及旋轉量（四元）

基本程式

```
# 输出基本信息
joint_num = p.getNumJoints(robot_id)  #獲得對應物件的結點數量
print("r2d2的节点数量为：", joint_num)

print("r2d2的信息：")
for joint_index in range(joint_num):
    info_tuple = p.getJointInfo(robot_id, joint_index) #獲得該物件指定結點詳細資料
    print(f"關節序號：{info_tuple[0]}\n\
          關節名稱：{info_tuple[1]}\n\
          關節類型：{info_tuple[2]}\n\
          機器人第一個位置的變量索引：{info_tuple[3]}\n\
          機器人第一個速度的變量索引：{info_tuple[4]}\n\
          保留参数：{info_tuple[5]}\n\
          關節的阻尼大小：{info_tuple[6]}\n\
          關節的摩擦系数：{info_tuple[7]}\n\
          slider和revolute(hinge)类型的位移最小值：{info_tuple[8]}\n\
          slider和revolute(hinge)类型的位移最大值：{info_tuple[9]}\n\
          關節驅動的最大值：{info_tuple[10]}\n\
          關節的最大速度：{info_tuple[11]}\n\
          結點名稱：{info_tuple[12]}\n\
          局部框架中的關節軸系：{info_tuple[13]}\n\
          父節點frame的關節位置：{info_tuple[14]}\n\
          父節點frame的關節方向：{info_tuple[15]}\n\
          父節點的索引，若是基座返回-1：{info_tuple[16]}\n\n")
```

- getNumJoints(robot_id)= 可以獲得對應物件的節點數量
- getJointInfo(robot_id,joint_index)= 可以獲得物件中對應結點的詳細參數

碰撞箱子模型設計（1/3）

載入物件以及碰撞的箱子並且結合再一起,這邊用到的檔案格式是.obj
這個檔案格式只包含了對於物體3D的幾何描述，其餘質量或是動力學相關的特徵都沒有
結合表述物體的視覺模型以及物體碰撞的幾何來設計碰撞的箱子

```
# 创建视觉形状
# shape_type 可以調整成以下 : GEOM_SPHERE, GEOM_BOX, GEOM_CAPSULE, GEOM_CYLINDER, GEOM_PLANE, GEOM_MESH
visual_shape_id = p.createVisualShape(
    shapeType=p.GEOM_MESH,
    fileName="lego/lego.obj",
    rgbaColor=[1, 1, 1, 1],
    specularColor=[0.4, 0.4, 0],
    visualFramePosition=shift,
    meshScale=scale
)
```

首先創建視覺形狀：
CreateVisualShape
shapeType = 可以決定物件顯示的種類
（GEOM_SPHERE, GEOM_BOX, GEOM_CAPSULE, GEOM_CYLINDER, GEOM_PLANE, GEOM_MESH）
rgbColor = 可以設計物件的顏色
SpecularColor = 可以設計反光的顏色
VisualFramePosition = 起始位移
MashScale = 物件縮放

碰撞箱子模型設計（2/3）

```
#創建碰撞箱形狀
collision_shape_id = p.createCollisionShape(
    shapeType=p.GEOM_MESH,
    fileName="lego/lego_vhacd.obj",
    collisionFramePosition=shift,
    meshScale=scale
)
```

創建碰撞的箱子形狀：
CreateCollisionShape
ShapeType = 跟視覺形狀一樣可以選擇很多種的呈現方式
CollisionFramePosition = 起始位置
MeshScale = 起始縮放

碰撞箱子模型設計（2/3）

```
p.createMultiBody(
    baseMass=10,
    baseCollisionShapeIndex=collision_shape_id,
    baseVisualShapeIndex=visual_shape_id,
    basePosition=[0, 0, 2*i],
    useMaximalCoordinates=True
)
```

將建立好的視覺模型以及碰撞模型結合在一起
CreateMultiBody：
BaseMass = 質量
BaseCollisionShapeIndex= 建立好的碰撞箱子
BaseVisualShapeIndex= 建立好的視覺模型
BasePosition = 初始位置

物件控制

- 以URDF的文件為例，整個物件由2個重要的元件構成
  - Base
  - Link
- 而兩個元件透過joint(關節)連接，也就是說可動的部份就是有關節的地方
- <https://mymodelrobot.appspot.com/5629499534213120>

物件控制

available_joints_indexes = [i for i in range(p.getNumJoints(robot_id)) if p.getJointInfo(robot_id, i)[2] != 4]
從物件中的所有元件中找到關節類型不為4的（固定不動的）
如果以r2d2為例子要找到輪子的link才可以讓他前進
wheel_joints_indexes = [i for i in available_joints_indexes if "wheel" in str(p.getJointInfo(robot_id, i)[1])]
target_v = 設置最後想到達的速度
max_force = 設置運動的加速度

```
p.setJointMotorControlArray(
    bodyUniqueId=robot_id_2,
    jointIndices=wheel_joints_indexes_2,
    controlMode=p.VELOCITY_CONTROL,
    targetVelocities=[target_v_2 for _ in wheel_joints_indexes_2],
    forces=[max_force_2 for _ in wheel_joints_indexes_2]
)
```

開始移動 setJointMotorControlArray()
bodyUniqueId=robot_id_2 #物件ID
jointIndices=wheel_joints_indexes_2 #要移動之關節ID可以一次驅動很多個關節點所以這邊輸入是一個list
controlMode=p.TORQUE_CONTROL #設定移動的模式 POSITION_CONTROL, VELOCITY_CONTROL, TORQUE_CONTROL PD_CONTR
targetVelocities=[target_v for _ in wheel_joints_indexes] #加速度因為一次驅動很多個關節點因此每個關節點要賦予一個速度
forces=[max_force for _ in wheel_joints_indexes] #加速度

物件控制

```
p.setJointMotorControl2(
    robot_id,
    2,
    p.VELOCITY_CONTROL,
    target_v,
    max_force
)
```

也是控制關節運動的一個方法但是跟前者不一樣的是一次只能控制一個關節運動

SetJointMotorControl2
robot_id #物件ID
2 #關節ID
p.VELOCITY_CONTROL#設定移動模式
target_v #期望速度
max_force #加速度

鏡頭追蹤

物體移動時會需要有一個鏡頭可以跟著移動中的物體進行紀錄或是觀看

```
location, _ = p.getBasePositionAndOrientation(robot_id_2)
p.resetDebugVisualizerCamera(
    cameraDistance=5,
    cameraYaw=110,
    cameraPitch=-30,
    cameraTargetPosition=location
)
```

getBasePositionAndOrientation(robot_id_2)#獲得物體的位置
ResetDebugVisualizerCamera()#設置相機
cameraDistance=5
cameraYaw=110
cameraPitch=-30
cameraTargetPosition=location#設置相機要拍攝的位置