

## 个人信息



张聿韬 / Male

手机: 13777588250  
微信号: yutao\_1999  
邮箱: yutaov5@outlook.com  
华东理工大学 控制工程

Github: <https://github.com/YuTaoV5>  
技术博客: <https://yutaov5.github.io/>

### 擅长领域

脑机接口    机器人    自动驾驶

欢迎各位老师与我交流~  
如果有工作或博士Offer, 请联系我要简历

## 环境准备: Ubuntu20.04

### Nvidia驱动相关

加入官方ppa源,根据自己的n卡可选驱动下载显卡驱动

```
1 sudo add-apt-repository ppa:graphics-drivers/ppa
2 sudo apt update
3 apt list --upgradable
4 sudo apt upgrade
5 ubuntu-drivers devices
6 sudo apt install nvidia-driver-535
```

### CUDA

```
sudo apt install nvidia-cuda-toolkit
```

### Ros安装

```
wget http://fishros.com/install -O fishros && . fishros
```

### Miniconda下载

官网地址 <https://docs.anaconda.com/miniconda/install/#quick-command-line-install>

```
1 mkdir -p ~/miniconda3
2 wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh -O ~/miniconda3/miniconda.sh
3 bash ~/miniconda3/miniconda.sh -b -u -p ~/miniconda3
4 rm ~/miniconda3/miniconda.sh
5 source ~/miniconda3/bin/activate
6 conda init --all
```

友情提醒: 请安装22.04版本直接选择重装系统, 由于您很难在22.04版本上面直接安装ROS1, 使用Docker技术的话, 由于X11显示服务, docker内的ROS1无法完整显示RVIZ等可视化应用

## Linux小技巧

### 命令行

<code>ctrl + alt + t</code>	唤出命令行
<code>cmd-&gt; pwd</code>	返回当前目录路径
<code>cmd-&gt; ls</code>	返回当前目录下所有文件及文件夹

cmd-> <code>cd ..</code>	返回上一级目录
cmd-> <code>cd xxx</code>	进入xxx文件夹
cmd-> <code>sudo gedit xxx</code>	超级权限用记事本编辑文件, 如果您不熟悉vim

## 双系统同步时间

```
1 sudo apt install ntpdate
2 sudo ntpdate time.windows.com
3 sudo hwclock --localtime --systohc
```

## 软件安装

deb安装包	<code>sudo dpkg -i linuxqq_3.1.2-13107_amd64.deb</code>
setup.py安装包	终端输入 <code>pip install -e .</code>
.sh安装	<code>chmod +x install.sh</code>
	<code>./install.sh</code>
一键安装缺失依赖	<code>sudo apt -f install</code>

## ROS项目编译

进入工作空间根目录

进入src文件夹, 如果没有可以把现有文件移入src文件夹,再 `catkin_init_workspace` 一下  
这一步会生成 `CMakeLists`

```
catkin_init_workspace
```

终端进入src上级文件夹, 初始化工作空间

```
catkin_make
```

编译完配置环境变量 `source` 一下

```
source devel/setup.bash
```

然后就可以 `roslaunch` 了

# 1. 强化学习基础-机器人走迷宫

## 1.1. 强化学习基本概念

强化学习是一种通过与环境的交互来学习实现目标的计算方法, 主要涉及以下几个方面:

- **感知:** 代理 (Agent) 通过观察环境来获取当前状态 (State) 。
- **行动:** 代理选择一个动作 (Action) 来执行。
- **目标:** 代理的目标是最大化累积奖励 (Reward) 。

### 1.1.1. 关键概念总结

概念	描述
智能体 (Agent)	在环境中执行动作并学习的实体。
环境 (Environment)	代理所处的外部世界, 提供状态和奖励。
状态 (State)	环境当前的情况, 代理的观察输入。
动作 (Action)	代理在当前状态下可选择的行为。

概念	描述
奖励 (Reward)	环境对代理动作的反馈，用于指导学习。
策略 (Policy)	代理选择动作的规则，可以是确定性的或概率性的。
Q值 (Q-value)	给定状态下采取某个动作的预期累计奖励。

### 1.1.2. Q-Learning算法基本原理

Q-Learning是一种无模型的强化学习算法，用于学习状态-动作值函数（Q函数）。其基本原理如下：

- **Q表:** 维护一个表格，记录所有状态-动作对的Q值。
- **Q值更新公式:**

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[ r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

其中：

- $\alpha$  是学习率，控制新知识和旧知识的权重。
- $r$  是即时奖励。
- $\gamma$  是折扣因子，控制未来奖励的权重。
- $s'$  是下一步的状态。
- **算法步骤:**
  - 初始化Q表，所有Q值设为0。
  - 代理在当前状态 $s$ 选择一个动作 $a$ （通常使用 $\epsilon$ -贪心策略）。
  - 执行动作 $a$ ，观察下一步状态 $s'$ 和奖励 $r$ 。
  - 更新Q值，使用上述Q值更新公式。
  - 设置 $s = s'$ ，重复步骤2-4，直到达到终止条件。

Q-Learning的目标是通过不断迭代更新Q值，最终收敛到最优策略，使得代理在任何状态下都能选择最优动作，以最大化累积奖励。

## 1.2. 实操记录：DQN算法部署

### 1.2.1. 操作指南

#### 创建环境

```
1 conda create -n DQN python=3.8
2 conda activate DQN
3 pip install torch==2.0.0 torchvision==0.15.1 torchaudio==2.0.1 -i https://pypi.tuna.tsinghua.edu.cn/simple
4 pip install -r requirements.txt
```

#### 将conda环境加入jupyter

```
python -m ipykernel install --user --name=DQN
```

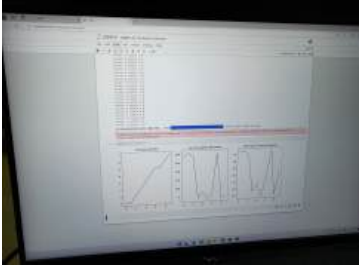
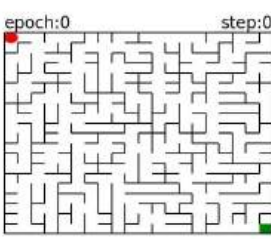
#### 启动jupyter

```
jupyter notebook
```

- 选择 `main.ipynb` 进入，kernel选择 `DQN` 环境
- 逐步运行 `main.ipynb`

该项目代码注释相当详细，本文不再做过多注解，请直接参考注解

### 1.2.2. 照片记录

DQN	效果
	

### 1.3. 应用拓展：可视化Q-Learning走迷宫

#### 📄 Q-Learning-UI

参考所给例程，通过PyQt5设计UI界面，将部分参数可视化，支持重复运行，自动生成随机地图

#### 使用方法

```

1 git clone https://github.com/YuTaoV5/DQN-ui.git
2 cd DQN-ui
3 conda activate DQN
4 pip install PyQt5
5 python main.py

```

#### 运行效果



Designed by YuTaoV5

## 2. 仿生机器人强化学习中的DWL算法与Humanoid-Gym项目

### 2.1. 算法与仿真的基本概念

#### 2.1.1. DWL算法概念

DWL (Denoising World Model Learning) 是一种用于仿生机器人强化学习的先进算法，旨在通过去噪世界模型学习，提升机器人在复杂环境中的适应能力。其核心特点包括：

- **去噪机制**：通过消除仿真环境与真实环境之间的噪声差异，缩小“仿真到现实” (Sim2Real) 的差距。
- **端到端学习**：利用单一神经网络实现从仿真到真实环境的零样本迁移，减少对复杂模型调整的需求。
- **鲁棒性与泛化能力**：在多种复杂地形（如雪地、楼梯、不规则表面）中表现出稳定的运动能力，并能够抵抗外部干扰。

#### 2.1.2. Humanoid-Gym开源项目

Humanoid-Gym 是一个基于NVIDIA Isaac Gym的强化学习框架，专门用于训练人形机器人的运动技能，并实现从仿真到真实环境的零样本迁移。其主要特点包括：

- **仿真到现实迁移**：支持从仿真环境（如Isaac Gym）到真实环境的无缝迁移，验证策略的鲁棒性和泛化能力。
- **仿真到仿真支持**：允许在Isaac Gym和Mujoco等仿真引擎之间进行策略迁移，进一步提升训练效率。
- **奖励函数设计**：提供专门为人形机器人设计的奖励函数，简化训练过程。
- **多机器人支持**：已在RobotEra的XBot-S (1.2米) 和XBot-L (1.65米) 人形机器人上成功验证。

### 2.1.3. 仿真对于强化学习的重要性

仿真在强化学习中扮演着至关重要的角色，主要体现在以下几个方面：

- **低成本与高效率**：仿真环境提供了低成本、高效率的训练平台，避免了真实环境中反复试错的高成本和风险。
- **可观测性与可控性**：仿真环境允许研究人员完全控制实验条件，便于调试和优化算法。
- **复杂场景模拟**：仿真可以模拟真实环境中难以复现的复杂场景（如极端地形、动态障碍物等），为算法提供多样化的训练数据。

### 2.1.4. 仿真引擎对比及其优劣

以下是常用仿真引擎的对比及其优劣：

仿真引擎	优点	缺点
Isaac Gym	- 高性能，支持大规模并行仿真 - 与NVIDIA硬件深度集成，计算效率高	- 对硬件要求较高 - 学习曲线较陡峭
Mujoco	- 物理仿真精度高 - 适合精细控制任务	- 商业软件，需付费 - 计算效率较低
Gazebo	- 开源免费 - 与ROS兼容性好，适合多机器人协同仿真	- 图形渲染和物理仿真精度较低 - 对复杂场景支持有限
PyBullet	- 开源免费 - 支持多种物理引擎，灵活性高	- 物理仿真精度和稳定性不如Mujoco
Webots	- 用户友好，适合初学者 - 支持多种机器人模型和传感器仿真	- 计算效率较低 - 对复杂场景的支持有限

### 2.1.5. 总结

- **DWL算法**通过去噪机制和端到端学习，显著提升了仿生机器人在复杂环境中的适应能力。
- **Humanoid-Gym**项目为仿生机器人强化学习提供了高效的训练框架，支持从仿真到现实的零样本迁移。
- **仿真是**强化学习的重要工具，不同仿真引擎各有优劣，需根据具体任务需求选择合适的平台。

## 2.2. 实操记录：Humanoid-gym部署

### 2.2.1. 操作指南

#### 创建环境

```
1 conda create -n isaac python=3.8
2 conda activate isaac
3 pip install torch==2.0.0 torchvision==0.15.1 torchaudio==2.0.1 -i https://pypi.tuna.tsinghua.edu.cn/simple
```

#### 安装isaac

解压"IsaacGym\_Preview\_4\_Package.tar.gz"  
请确保终端此时在conda环境内

```
cd isaacgym/python && pip install -e .
```

#### 测试isaac

```
cd examples && python 1080_balls_of_solitude.py
```

#### 安装humanoid-gym项目

```
1 git clone https://github.com/roboterax/humanoid-gym.git
2 cd humanoid-gym && pip install -e .
```

如果 `git clone` 出现让你输入账号密码，你输入后他仍然报错拒绝访问的话，排除网络原因也可能因为是你的git版本太高了，git新版本规定要密钥登陆，旧版本没有这限制



## 训练, 测试与移植

在4096个环境中启动'v1'的PPO策略训练  
此命令启动基于PPO算法的人形机器人任务训练。

```
python scripts/train.py --task=humanoid_ppo --run_name v1 --headless --num_envs 4096
```

如果显存不够, 请降低尾部数字

评估训练好的PPO策略'v1'  
此命令加载'v1'策略以在其环境中评估性能。

```
1 cd humanoid
2 python scripts/play.py --task=humanoid_ppo --run_name v1
```

实施仿真到仿真模型转换  
此命令使用导出的'v1'策略进行仿真到仿真转换。

```
python scripts/sim2sim.py --load_model /path/to/logs/XBot_ppo/exported/policies/policy_1.pt
```

运行训练好的策略

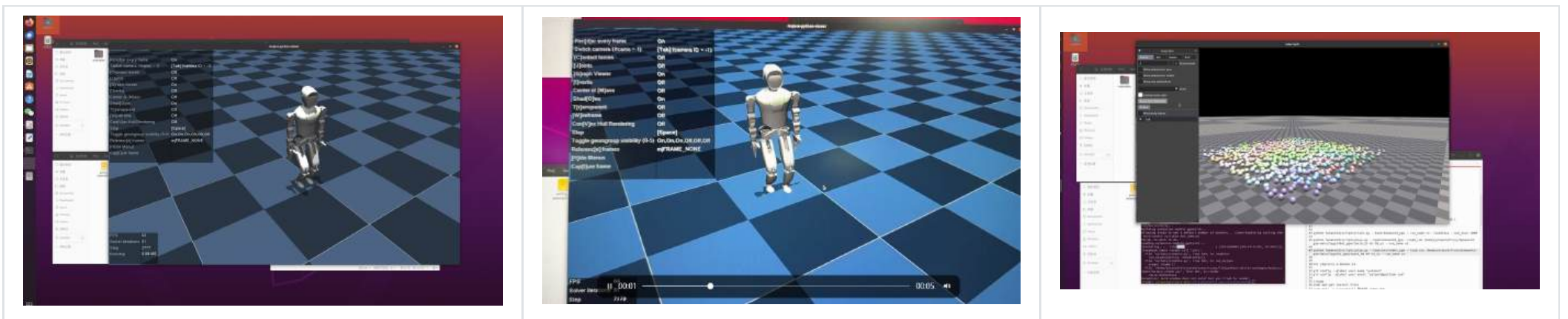
```
python scripts/sim2sim.py --load_model /path/to/logs/XBot_ppo/exported/policies/policy_example.pt
```

/path/to/logs 是指您电脑上指向logs的绝对地址 如:

```
python humanoid/scripts/play.py --task=humanoid_ppo --load_run /home/yutaov5/Projs/humanoid-gym-main/logs/XBot_ppo/Jan14_21-01-39_v1 --run_name v1
```

训练好的默认模型储存在 logs/{experiment\_name}/exported/policies

### 2.2.2. 照片记录



### 2.3. 应用拓展: 移植宇树H1机器人到Humanoid-gym

#### 📌 移植要点

在 resources -> robots 文件夹里面添加 H1的urdf文件夹  
在 humanoid -> envs 里面的 \_\_init\_\_ 添加任务,修改如下:

```
from humanoid import LEGGED_GYM_ROOT_DIR, LEGGED_GYM_ENVS_DIR
from .base.legged_robot import LeggedRobot

from .custom.humanoid_config import XBotLCfg, XBotLCfgPPO
from .custom.humanoid_env import XBotLFreeEnv
from .custom.unitreeh1_config import Unitreeh1Cfg, Unitreeh1CfgPPO
from .custom.unitreeh1_env import Unitreeh1Env

from humanoid.utils.task_registry import task_registry

task_registry.register( "unitreeh1_ppo", Unitreeh1Env, Unitreeh1Cfg(), Unitreeh1CfgPPO() )
task_registry.register( "humanoid_ppo", XBotLFreeEnv, XBotLCfg(), XBotLCfgPPO() )
```

## 📌 移植要点

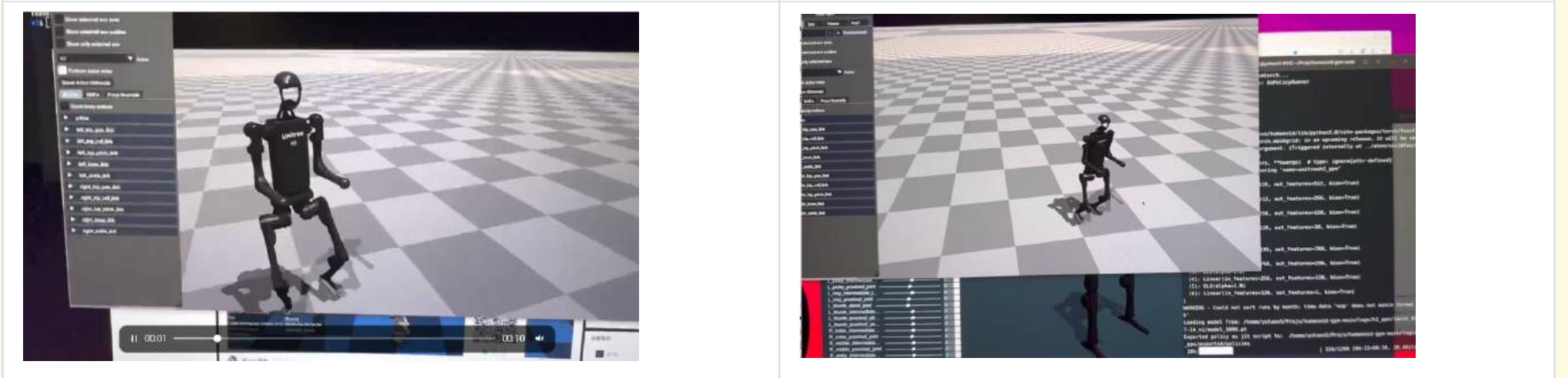
在 humanoid -> envs -> custom 里面的仿照原任务复制添加两个文件: unitreeh1\_config.py 和 unitreeh1\_env.py , 并修改相关参数  
在 humanoid -> scripts 里面的仿照原 sim2sim.py 文件复制添加一个 sim2sim\_h1.py 文件, 并修改相关参数

## 运行指令

```
1 python scripts/train.py --task=unitreeh1_ppo --run_name v1 --headless --num_envs 1000
2 python humanoid/scripts/play.py --task=unitreeh1_ppo --load_run /home/yutaov5/Projs/humanoid-gym-main/logs/h1_ppo/Jan14_01-07-14_v1 --run_name v1
```

请注意指令相应地址替换成自己的路径

## 运行效果



# 3. 仿生机器人强化学习中的PPO算法与Unitree-RL-Gym项目

## 3.1. PPO算法概念

PPO (Proximal Policy Optimization) 是一种强化学习算法, 旨在通过限制策略更新的幅度, 保证训练的稳定性和效率。其核心思想是通过**裁剪机制** (Clipping Mechanism) 限制策略更新的幅度, 避免策略更新过大导致训练不稳定。PPO的主要特点包括:

- **稳定性**: 通过裁剪机制限制策略更新幅度, 防止策略崩溃。
- **高效性**: 支持多次小批量更新, 提高数据利用率。
- **普适性**: 适用于连续动作空间和高维状态空间, 广泛应用于机器人控制、游戏AI等领域。

### 3.1.1. PPO的核心公式

PPO的目标函数如下:

$$L^{CLIP}(\theta) = \mathbb{E}_t \left[ \min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t) \right]$$

其中:

- $r_t(\theta) = \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ : 新旧策略的概率比率。
- $\hat{A}_t$ : 优势函数, 表示动作的相对优劣。
- $\epsilon$ : 裁剪阈值, 通常设为0.1或0.2。

## 3.2. Unitree-RL-Gym开源项目

Unitree-RL-Gym 是宇树科技开源的强化学习框架, 专注于仿生机器人 (如四足机器人Go2、人形机器人H1) 的运动控制。其特点包括:

- **支持多仿真引擎**: 包括NVIDIA Isaac Gym和MuJoCo, 支持从仿真到真实环境的无缝迁移。
- **完整训练流程**: 提供从数据采集、模型训练到真机部署的全流程支持。
- **开源代码与教程**: 包含详细的代码和教程, 帮助用户快速上手。

### 3.2.1. 项目结构

```
1 unitree_rl_gym/
2 |— legged_gym/      # 核心代码
3 |   |— scripts/     # 训练和测试脚本
```

```
4 | |─ config/          # 配置文件
5 | |─ resources/       # 机器人模型和资源
6 | └─ LICENSE         # 开源许可证
7 | └─ README.md       # 项目说明
8 | └─ setup.py        # 安装脚本
```

## 3.3. 实操记录 Unitree-rl-gym部署

### 3.3.1. 操作指南

#### 安装环境

```
1 | conda create -n unitree python=3.8
2 | conda activate unitree
3 | pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple
4 | pip install torch==2.4.1 torchvision==0.19.1 torchaudio==2.4.1 -i https://pypi.tuna.tsinghua.edu.cn/simple
```

#### 安装isaac

解压"IsaacGym\_Preview\_4\_Package.tar.gz"  
请确保终端此时在conda环境下

```
cd isaacgym/python && pip install -e .
```

#### 测试isaac

```
cd examples && python 1080_balls_of_solitude.py
```

#### 下载项目

请回到主目录，不要在isaac目录里面下载新项目

```
1 | git clone https://github.com/leggedrobotics/rs1_rl.git
2 | cd rs1_rl
3 | git checkout v1.0.2
4 | pip install -e.
```

rs1\_rl 请注意使用 `git clone` 拉下来项目，不要去网站下载zip

```
1 | git clone https://github.com/unitreerobotics/unitree_rl_gym.git
2 | cd unitree_rl_gym
3 | pip install -e.
```

```
1 | git clone https://github.com/unitreerobotics/unitree_sdk2_python.git
2 | cd unitree_sdk2_python
3 | pip install -e.
```

#### 运行项目

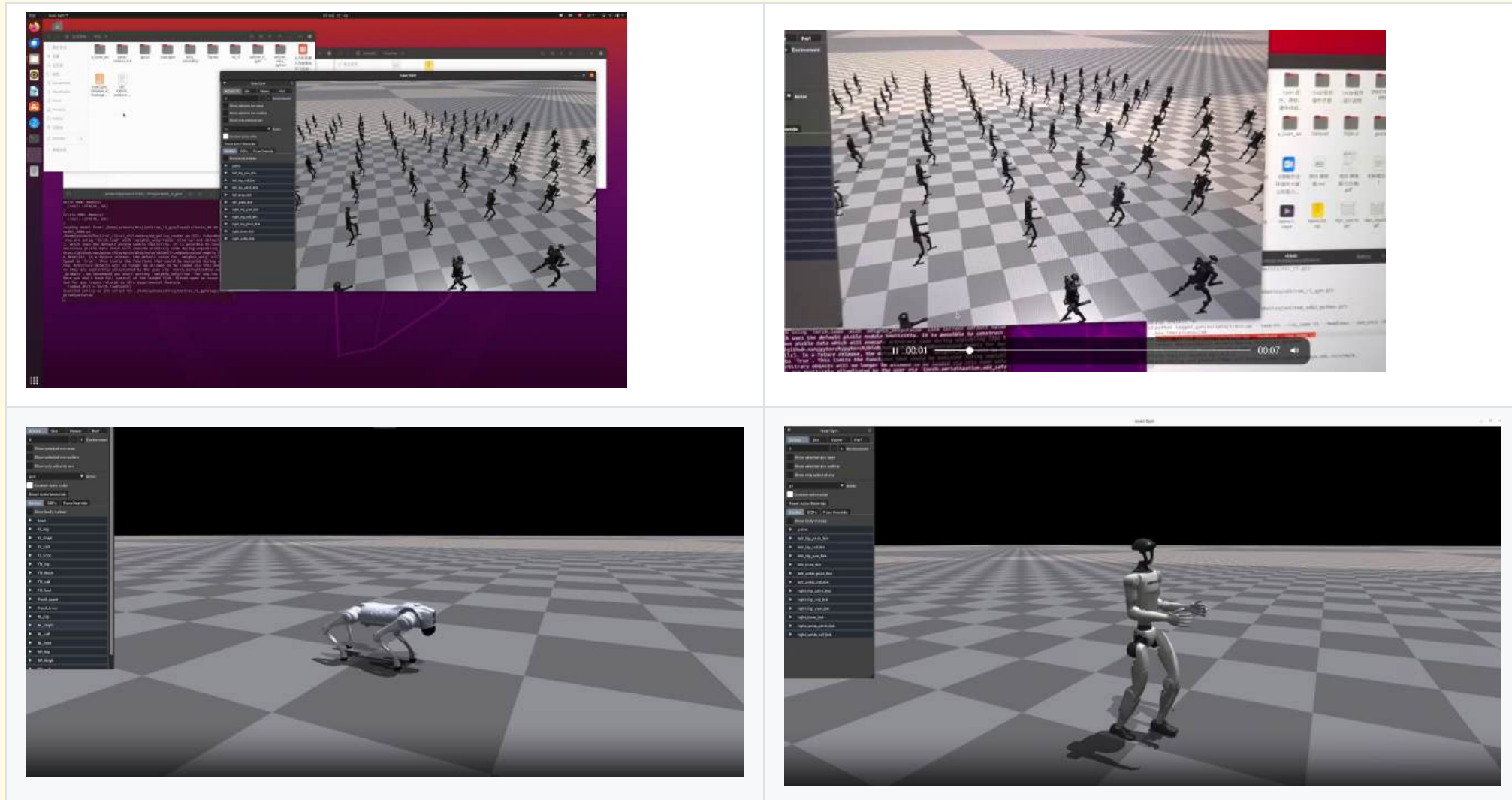
回到主目录，进入unitree\_rl\_gym文件夹

```
1 | python legged_gym/scripts/train.py --task=h1 --run_name V1 --headless --num_envs 1000 --max_iterations=5000
2 | python legged_gym/scripts/play.py --task=h1 --run_name v1
```

### 3.3.2. 照片记录



## 运行效果



## 4. 需求驱动导航DDN

环境安装以及指令比较简单，不在此多赘述

### CLIP报错

我将给出我是怎么排查这个错误的流程

#### Step 1

这是报错的位置，我发现这里是直接import clip  
既然没有同级clip文件或文件夹，说明这是安装的包

```
1 import gc
2 from copy import deepcopy
3 import math
4 import clip
5 from utils.util import get_valid_transforms, _convert_image_to_rgb
6 from model.detr import DETRModel
7 # from thortils.navigation import (
8 #     get_shortest_path_to_object,
9 # )
10 from utils.util import make_env_fn
11 from vector_env import VectorEnv
```

#### Step 2

直接pip发现，是一个小的安装包，但是 src 文件夹里面直接有 clip 文件夹，里面有setup.py,那么直接安装本地包即可,更加方便

```
1 cd src/clip
2 pip install -e .
```

## 运行结果



就比较小一个窗口

## 5. 环境定位算法A-LOAM算法概念及其开源项目

### 5.1. A-LOAM算法

A-LOAM (Advanced Lidar Odometry and Mapping) 是一种基于激光雷达的3D SLAM算法，旨在通过高效的点云处理和优化技术，实现机器人的实时定位与建图。其核心特点包括：

- **特征提取**：从激光雷达点云中提取边缘点 (edge points) 和平面点 (planar points) 作为特征点，用于位姿估计。
- **运动补偿**：通过时间戳对齐和插值技术，消除激光雷达在运动过程中产生的点云畸变。
- **优化框架**：使用Ceres库进行非线性优化，替代了传统LOAM算法中的手推高斯牛顿法，简化了代码实现并提高了可读性:cite[1]:cite[6]。

#### 5.1.1. A-LOAM开源项目

A-LOAM 是一个开源的激光SLAM项目，基于ROS (Robot Operating System) 框架开发，适用于自动驾驶、机器人导航等场景。其主要特点包括：

- **易用性**：提供详细的安装和运行指南，适合初学者快速上手。
- **高效性**：利用Ceres和PCL库，能够高效处理激光雷达数据，生成高质量的点云地图。
- **灵活性**：支持多种运行模式，包括直接运行数据包和与机器人平台集成:cite[6]:cite[8]。

#### 5.1.2. A-LOAM与Point-LIO、Fast-LIO的优劣比较

以下是A-LOAM、Point-LIO和Fast-LIO的对比：

特性	A-LOAM	Point-LIO	Fast-LIO
核心算法	基于特征点的优化方法，使用Ceres库进行非线性优化。	基于点云的优化方法，结合IMU数据进行高精度定位。	基于关键帧的优化方法，通过IMU预积分加速计算。
计算效率	较高，适合实时应用，但对大规模点云处理效率较低。	较低，计算复杂度高，适合高精度场景。	高，通过算法优化实现快速计算，适合实时性要求高的场景。
硬件要求	中等，依赖Ceres和PCL库，适合普通计算平台。	高，需要较强的计算资源处理大量点云数据。	低，优化后的算法对硬件要求较低，适合资源受限的设备。
精度与鲁棒性	较高，适合复杂环境下的定位与建图。	高，基于点云的优化方式在复杂环境下表现优秀。	较高，但在复杂环境中可能略逊于Point-LIO。
应用场景	自动驾驶、机器人导航、室内外定位。	精细化环境建图、高精度定位。	自动驾驶、机器人导航等实时性要求高的场景。

#### 5.1.3. 激光SLAM算法对人形机器人自主导航的重要性

激光SLAM算法在人形机器人自主导航中扮演着关键角色，主要体现在以下几个方面：

- **环境感知**：激光雷达能够提供高精度的3D环境信息，帮助机器人实时感知周围环境，避免碰撞。
- **定位与建图**：通过SLAM算法，机器人能够在未知环境中实现自主定位与建图，为路径规划和导航提供基础。
- **鲁棒性**：激光SLAM对光照和纹理不敏感，适合复杂多变的动态环境，如家庭、商场等场景:cite[3]:cite[9]。

#### 5.1.4. 总结

- A-LOAM 是一种高效、易用的激光SLAM算法，适合初学者和实时应用场景。
- Point-LIO 和 Fast-LIO 分别在高精度和实时性方面具有优势，适合不同的应用需求。
- 激光SLAM 是人形机器人实现自主导航的核心技术，为其提供了高精度的环境感知和定位能力。

## 5.2. 实操记录: A-LOAM部署

### 5.2.1. 操作指南

#### Ros安装

```
wget http://fishros.com/install -O fishros && . fishros
```

#### 下载ceres2.0.0

<http://ceres-solver.org/ceres-solver-2.0.0.tar.gz>

#### 安装依赖

```
1 sudo apt-get install liblapack-dev libsuitesparse-dev libcxsparse3 libgflags  
2 dev libgoogle-glog-dev libgtest-dev
```

#### 编译与安装ceres

```
1 cd ceres-solver  
2 mkdir build && cd build  
3 cmake ..  
4 make -j  
5 sudo make install
```

注意 `make -j` 表示全核心运行, 若您是使用云主机, 请切换为 `make -j4`  
切换为主目录

## 下载安装A-LOAM

如果你没有a\_loam\_ws文件夹

建立a\_oam\_ws文件夹, 建立并进入src文件夹  
这一步会生成 `cmakelist`

```
1 cd a_loam_ws/src  
2 git clone https://github.com/YuehaoHuang/A-LOAM.git  
3 catkin_init_workspace
```

终端进入src上级文件夹, 初始化工作空间

```
catkin_make
```

编译完配置环境变量 `source` 一下

```
source devel/setup.bash
```

然后就可以 `roslaunch` 了

如果你有a\_loam\_ws文件夹, 有devel等文件夹

```
1 cd a_loam_ws/src  
2 git clone https://github.com/YuehaoHuang/A-LOAM.git  
3 cd ../  
4 catkin_make
```

请注意, 之后每次 `roslaunch` 都需要 `source` 一下

## 运行VLP-16

### 启动ROS

`ctrl + alt + t` 唤出终端，输入

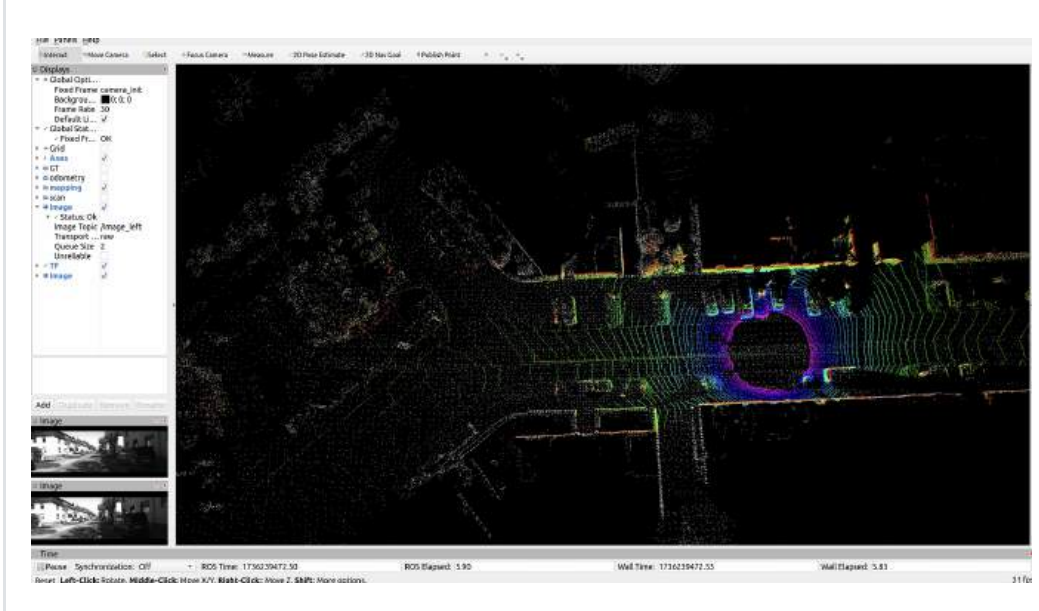
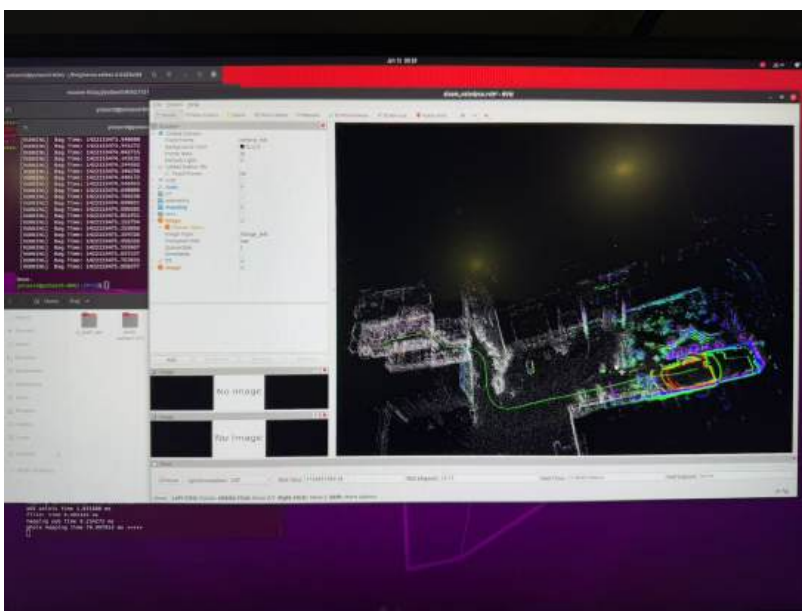
```
roscore
```

`ctrl + alt + t` 唤出新终端，输入

```
1 cd a_loam_ws
2 source devel/setup.bash
3 roslaunch aloam_velodyne aloam_velodyne_VLP_16.launch
```

打开 `nsh_indoor_outdoor.bag` 所在文件夹，右键打开新终端，输入

```
rosbag play nsh_indoor_outdoor.bag
```



## 5.3. 课后作业



第三四节课助教

25/1/15

这个留着当作业了，提示一下和这部分代码有关，有兴趣的学员自行调试



第三四节课助教

25/1/15



请找到 `A-LOAM -> src -> kittiHelper.cpp` 这个文件，找到 `main` 函数，观察如下片段：

```
1 while (std::getline(timestamp_file, line) && ros::ok()) {
2   float timestamp = stof(line);
3   std::stringstream left_image_path, right_image_path;
4   left_image_path << dataset_folder
5     << "sequences/" + sequence_number + "/image_0/"
6     << std::setfill('0') << std::setw(6) << line_num << ".png";
7   cv::Mat left_image =
8     cv::imread(left_image_path.str(), cv::IMREAD_GRAYSCALE);
9
10  right_image_path << dataset_folder
11    << "sequences/" + sequence_number + "/image_1/"
12    << std::setfill('0') << std::setw(6) << line_num << ".png";
13
14  lidar_data_path << dataset_folder
15    << "sequences/" + sequence_number + "/velodyne/"
16    << std::setfill('0') << std::setw(6) << line_num << ".bin";
17  if (to_bag) {
18    bag_out.write("/image_left", ros::Time::now(), image_left_msg);
```



```

19 bag_out.write("/image_right", ros::Time::now(), image_right_msg);
20 bag_out.write("/velodyne_points", ros::Time::now(), laser_cloud_msg);
21 bag_out.write("/path_gt", ros::Time::now(), pathGT);
22 bag_out.write("/odometry_gt", ros::Time::now(), odomGT);
23 }

```

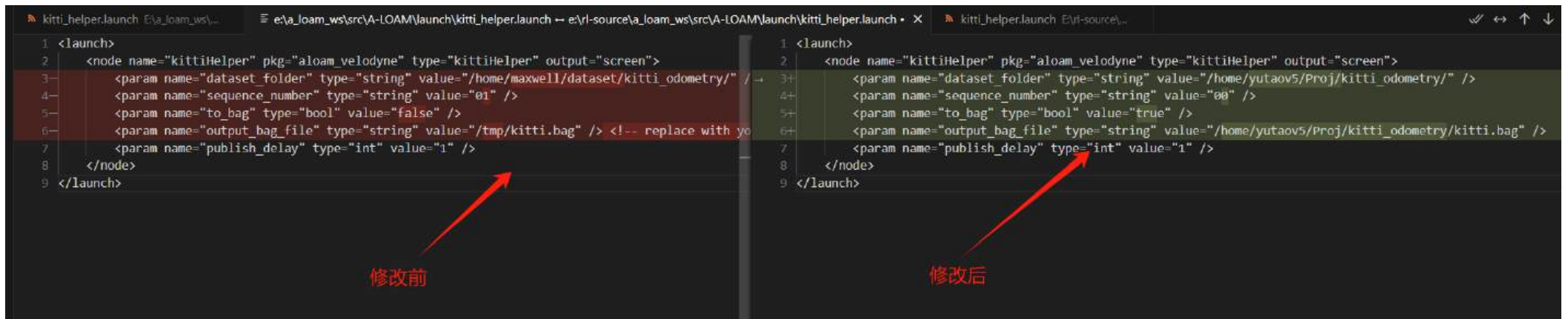
这就是3类文件的读取路径，所以选择保存 bag 后，他的路径就在下面这段代码，这样就好理解他的保存逻辑了，主要就在文件结构上

请找到 A-LOAM -> launch -> kitti\_helper.launch 修改如下：

```

1 <launch>
2   <node name="kittiHelper" pkg="aloam_velodyne" type="kittiHelper" output="screen">
3     <param name="dataset_folder" type="string" value="/home/yutaov5/Proj/kitti_odometry/" />
4     <param name="sequence_number" type="string" value="00" />
5     <param name="to_bag" type="bool" value="true" />
6     <param name="output_bag_file" type="string" value="/home/yutaov5/Proj/kitti_odometry/kitti.bag" />
7     <param name="publish_delay" type="int" value="1" />
8   </node>
9 </launch>

```



然后我的 kitti\_odometry 数据集的目录在 /home/yutaov5/Proj/kitti\_odometry/，我的文件结构如下

```

| kitti_odometry \
|-----| sequences \
|-----| | 00 \
|-----| | image_0 \
|-----| | image_1 \
|-----| | image_2 \
|-----| | image_3 \
|-----| | velodyne \
|-----| | calib.txt
|-----| | times.txt

```

## 运行指令

ctrl + alt + t 唤出新终端，输入

```

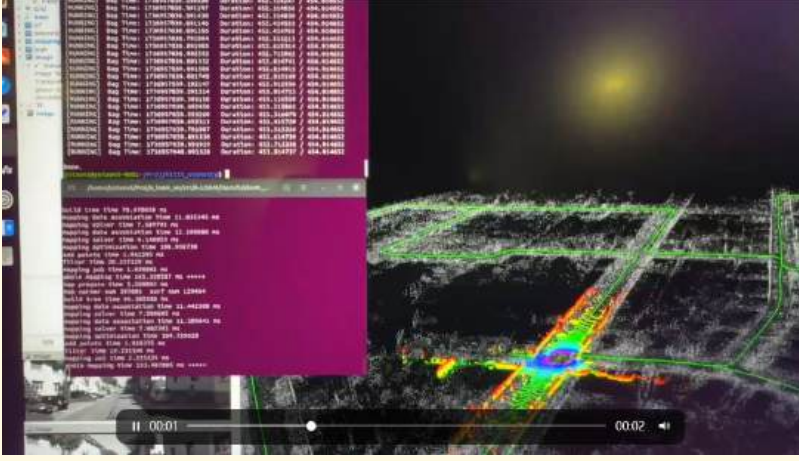
1 source devel/setup.bash
2 roslaunch aloam_velodyne aloam_velodyne_HDL_64.launch
3 roslaunch aloam_velodyne kitti_helper.launch

```

等待他结束即可~



## 运行截图



大概20G左右的 bag

## 6. ORCA算法概念与开源项目介绍

### 6.1. ORCA算法概念

ORCA (Optimal Reciprocal Collision Avoidance) 是一种用于多机器人动态避障的运动规划算法，旨在通过局部优化实现高效、无碰撞的路径规划。其核心思想是：

- **速度障碍法 (Velocity Obstacle, VO)**：通过计算每个机器人的速度障碍区域，避免与其他机器人发生碰撞。
- **互惠性 (Reciprocity)**：假设所有机器人均遵循相同的避障规则，通过共享避障责任，实现公平的避障行为。
- **优化目标**：在满足动力学约束的前提下，选择最优速度，使机器人尽可能接近目标速度。

#### 6.1.1. ORCA的核心公式

ORCA通过以下公式计算每个机器人的可行速度集合：

$$ORCA_{A|B}^{\tau} = \{v | (v - v_A^{opt}) \cdot n \geq 0\}$$

其中：

- $v_A^{opt}$ ：机器人A的最优速度。
- $n$ ：速度障碍区域的法向量。
- $\tau$ ：时间窗口，表示避障的时间范围。

### 6.2. ORCA开源项目

ORCA 是一个开源的动态避障算法实现，广泛应用于多机器人系统、自动驾驶和游戏AI等领域。其主要特点包括：

- **高效性**：能够在实时环境中快速计算避障速度。
- **灵活性**：支持多种机器人模型和运动学约束。
- **易用性**：提供详细的文档和示例代码，适合快速集成到机器人系统中。

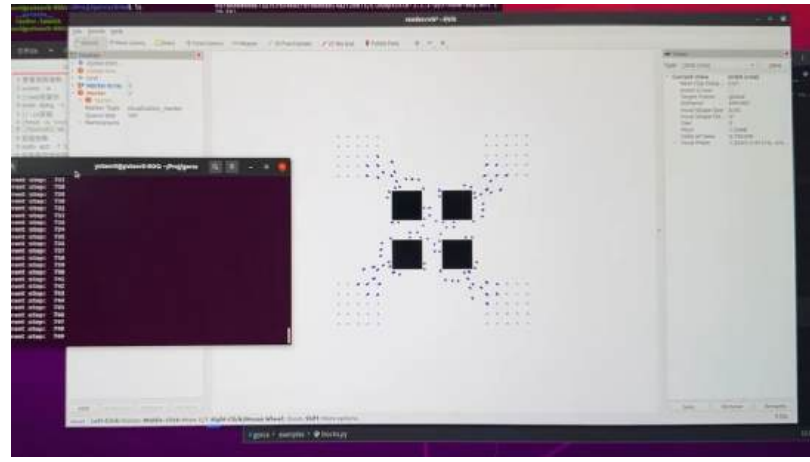
#### 6.2.1. 项目结构

```
gorca/  
├─draw  
├─examples  
├─figs  
├─rvo  
├─前沿人形机器人仿真环境下轨迹规划算法研发与实践-1.16  
├─.idea  
└─.git
```

环境安装比较简单

### 6.3. 运行结果

运行 blocks\_new.py



## 7. PCA算法用于点云图像生成Demo3D项目

### 7.1. PCA算法概念

PCA (Principal Component Analysis, **主成分分析**) 是一种降维算法, 通过线性变换将高维数据投影到低维空间, 同时保留数据的主要特征。在点云图像生成中, PCA用于提取点云的主要方向 (如法向量、主方向等), 从而辅助三维感知和机器人抓取位姿估计。

#### 7.1.1. PCA的核心步骤

- 中心化**: 将点云数据平移至原点, 使其均值为零。
- 协方差矩阵**: 计算点云的协方差矩阵, 描述点云在不同方向上的分布。
- 特征值分解**: 对协方差矩阵进行特征值分解, 得到特征值和特征向量。
- 主成分提取**: 选择最大的特征值对应的特征向量作为点云的主方向。

#### 7.1.2. PCA在点云图像生成中的应用

PCA在点云图像生成中的主要应用包括:

- 法向量估计**: 通过PCA提取点云的局部表面法向量, 用于三维重建和表面分析。
- 主方向提取**: 通过PCA提取点云的主方向, 用于物体位姿估计和机器人抓取规划。
- 降维与压缩**: 通过PCA将高维点云数据降维, 减少计算复杂度。

#### 7.1.3. PCA算法在三维感知与机器人抓取位姿估计中的重要性

PCA是三维感知和机器人抓取位姿估计的基本算法, 主要体现在以下几个方面:

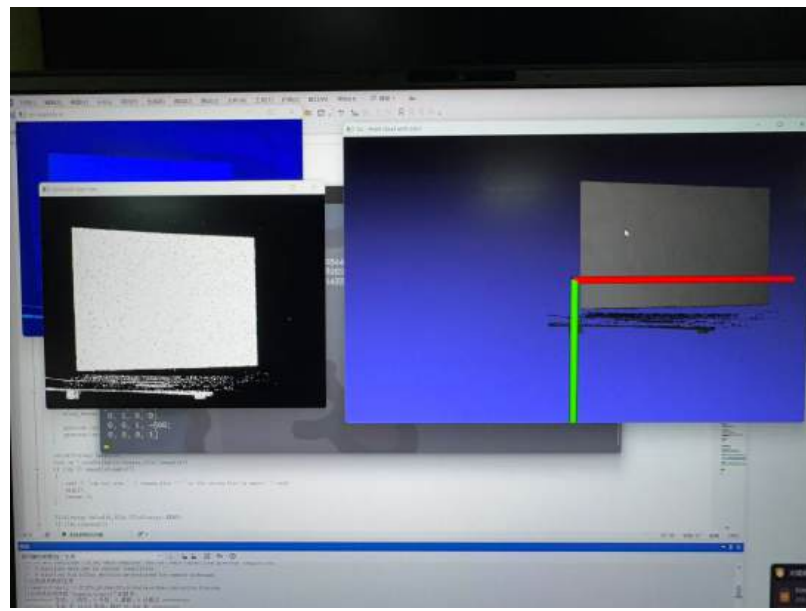
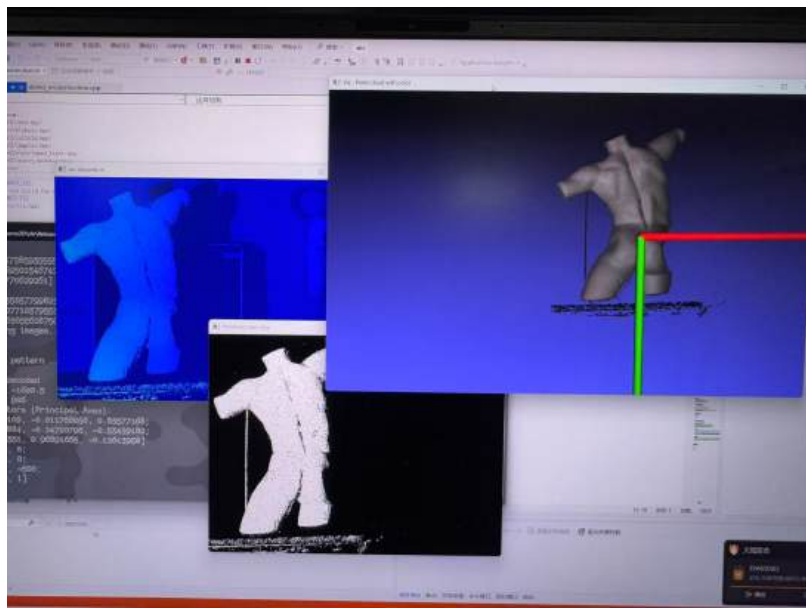
应用场景	描述
三维感知	通过PCA提取点云的主方向和法向量, 辅助物体识别和环境建模。
抓取位姿估计	通过PCA提取物体的主方向, 确定抓取点的位置和姿态。
点云降维	通过PCA将高维点云数据降维, 减少计算复杂度, 提高实时性。
点云压缩	通过PCA保留点云的主要特征, 减少存储空间和传输带宽。

### 7.2. PCA算法的优势与局限性

以下是PCA算法在点云图像生成中的优势与局限性总结:

特性	描述
优势	<ul style="list-style-type: none"><li>- 计算简单, 易于实现</li><li>- 能够有效提取点云的主要特征</li><li>- 适合实时应用</li></ul>
局限性	<ul style="list-style-type: none"><li>- 对噪声敏感, 需预处理去噪</li><li>- 仅适用于线性数据分布</li><li>- 无法处理非线性结构</li></ul>

### 7.3. 运行结果



## 8. LFG-NAV算法与多模态大模型GPT-4在视觉语言导航中的应用

### 8.1. LFG-NAV算法概念

LFG-NAV (Language-Guided Navigation) 是一种结合多模态大模型（如GPT-4）的视觉语言导航算法，旨在通过自然语言指令和视觉感知实现智能导航。其核心思想是：

- **多模态融合**：将视觉信息（如图像、点云）与语言指令（如目标描述、路径规划）结合，生成导航策略。
- **思维链推理**：通过大语言模型的推理能力，将复杂的导航任务分解为多个子任务，逐步生成导航策略。
- **启发式策略**：将大语言模型的输出转化为规划过程中的启发式策略，而非直接执行其建议，从而提高导航的鲁棒性和适应性。

#### 8.1.1. GPT-4在视觉语言导航中的作用

GPT-4 是一种多模态大语言模型，能够处理文本、图像等多种输入形式。在视觉语言导航中，GPT-4的作用包括：

- **指令理解**：解析自然语言指令，理解用户需求。
- **任务分解**：将复杂的导航任务分解为多个子任务，生成思维链。
- **策略生成**：结合视觉感知信息，生成导航策略。

#### 8.1.2. 思维链推理在导航中的应用

思维链 (Chain-of-Thought, CoT) 是一种通过逐步推理解决复杂问题的方法。在视觉语言导航中，思维链的作用包括：

- **任务分解**：将导航任务分解为多个子任务（如目标识别、路径规划、避障等）。
- **推理引导**：通过大语言模型的推理能力，逐步生成导航策略。
- **启发式策略**：将大语言模型的输出转化为规划过程中的启发式策略，提高导航的鲁棒性。

### 8.2. LFG-NAV算法的优势与局限性

以下是LFG-NAV算法在视觉语言导航中的优势与局限性总结：

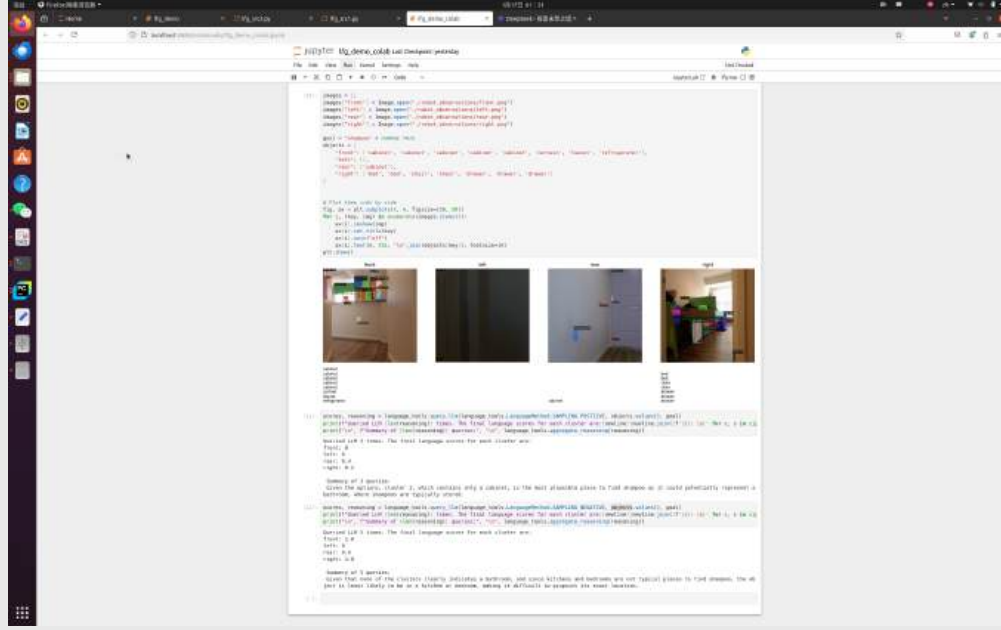
特性	描述
优势	<ul style="list-style-type: none"><li>- 结合多模态信息，提高导航精度</li><li>- 通过思维链推理，增强策略的合理性</li><li>- 支持复杂环境下的智能导航</li></ul>
局限性	<ul style="list-style-type: none"><li>- 对计算资源要求较高</li><li>- 依赖高质量的多模态数据</li><li>- 需要复杂的模型训练和调优</li></ul>

- **LFG-NAV算法** 通过结合多模态大模型（如GPT-4）和思维链推理，实现了智能视觉语言导航。
- **思维链推理** 能够充分发挥大语言模型的推理作用，将复杂任务分解为多个子任务，生成合理的导航策略。
- **启发式策略** 通过将大语言模型的输出转化为规划过程中的启发式策略，提高了导航的鲁棒性和适应性。

### 8.3. 运行结果



使用jupyter notebook,环境安装与运行都比较简单



## 9. SolidWorks在机器人结构设计中的使用

### 9.1. SolidWorks简介

SolidWorks 是一款广泛应用于机械设计和工程的三维CAD软件，特别适合机器人结构设计。其核心功能包括：

- **参数化建模**：通过参数驱动设计，快速修改和优化机器人结构。
- **装配体设计**：支持多零件装配，模拟机器人运动学行为。
- **仿真与分析**：提供运动学、动力学和有限元分析工具，验证设计的可行性。

#### 9.1.1. SolidWorks在机器人结构设计中的使用操作要点

以下是SolidWorks在机器人结构设计中的关键操作步骤：

操作步骤	描述
1. 零件设计	使用草图工具绘制机器人零件的2D轮廓，通过拉伸、旋转等操作生成3D模型。
2. 装配体设计	将多个零件组装成完整的机器人结构，定义零件之间的约束关系（如配合、对齐）。
3. 运动学仿真	使用Motion Study工具模拟机器人运动，验证关节运动范围和运动学性能。
4. 有限元分析	使用Simulation工具进行应力、应变分析，确保结构强度和刚度满足要求。
5. 生成工程图	创建2D工程图，标注尺寸和公差，用于制造和装配。

#### 9.1.2. 结构设计对机器人运动控制的巨大影响

机器人结构设计直接影响其运动控制性能，主要体现在以下几个方面：

- **运动学性能**：结构设计决定了机器人的关节类型、运动范围和自由度，影响运动规划的复杂性。
- **动力学性能**：结构设计影响机器人的质量分布和惯性特性，决定其动态响应速度和稳定性。
- **精度与刚度**：结构设计影响机器人的刚度和精度，决定其在高负载或高速运动中的表现。
- **能量效率**：合理的结构设计可以减少能量损耗，提高机器人的续航能力。

### 9.2. 生成URDF文件

URDF (Unified Robot Description Format) 是描述机器人模型的XML格式文件，用于ROS (Robot Operating System) 中的仿真和控制。以下是使用SolidWorks生成URDF文件的步骤：

#### 9.2.1. 4.1 安装插件

- 安装SolidWorks到URDF导出插件 ( `sw_urdf_exporter` ) 。

#### 9.2.2. 4.2 导出URDF文件

1. **定义关节和连杆**：在SolidWorks中为每个关节和连杆添加参考坐标系。
2. **设置运动学参数**：定义关节类型（如旋转、平移）和运动范围。
3. **导出URDF**：使用插件将装配体导出为URDF文件，包含机器人的几何、运动学和视觉信息。

## 9.3. 作业展示

### 9.3.1. 课堂作业



### 9.3.2. 家庭作业

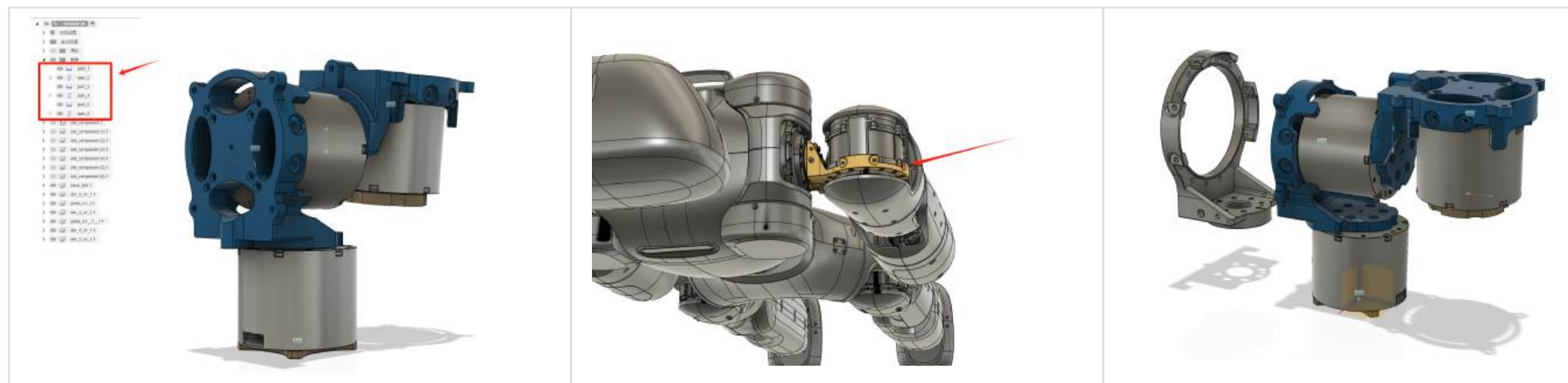


#### 关键设计考虑

- 运动学分析：该关节结构具有3个自由度，确保运动范围和精度满足要求。
- 负载能力：考虑关节结构的负载能力，选择合适的结构。
- 装配孔设计：除了预留电机安装螺孔外，还设计了多个3mm的装配孔，方便与其他部件连接。
- 结构优化：通过优化结构设计，减小关节结构的重量和体积，提高运动灵活性。

#### 9.3.2.1. 结构展示以及3D打印

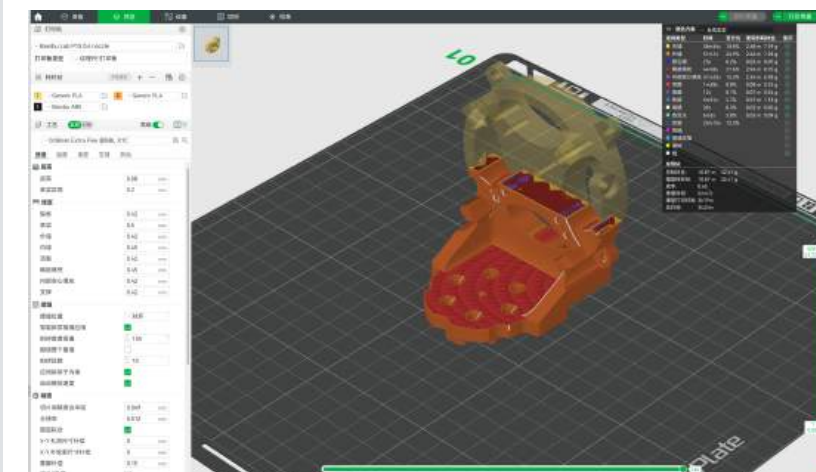
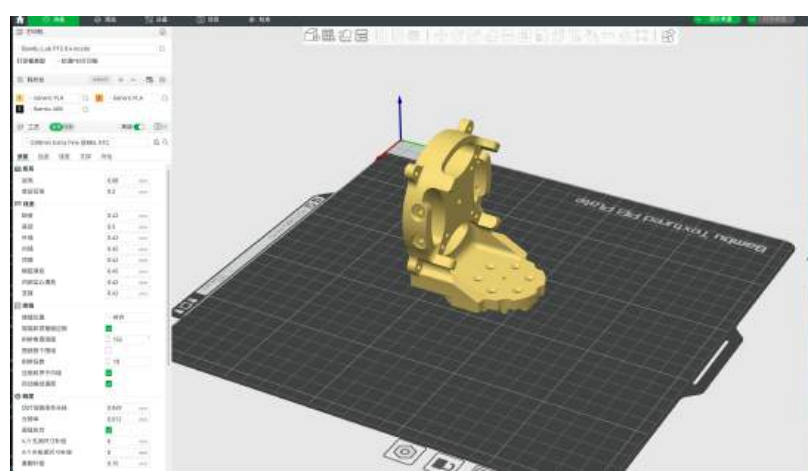
##### 关节联结展示，参考的灵犀肩关节以及对比



3D打印软件Bambu Lab



本设计为一个3电机关节结构，参考灵犀机器人肩关节设计，具有类似的运动模式和结构特点，并预留了电机安装螺孔和其他装配孔。该关节结构由三个电机和两个连接件组成，能够实现多自由度运动。



### 文件分享

<https://a360.co/4gWmcGL>

## 10. 导纳控制在机械臂控制中的应用与优势

### 10.1. 导纳控制概念

**导纳控制 (Admittance Control)** 是一种基于力-位置关系的控制方法，通过调整机械臂的动态响应特性，使其能够适应外部力的作用。其核心思想是：

- **力-位置映射**：将外部力（如接触力）转化为位置或速度调整，实现柔顺控制。
- **动态响应调节**：通过调整导纳参数（如质量、阻尼、刚度），控制机械臂的动态行为。

#### 10.1.1. 导纳控制在机械臂控制中的应用

导纳控制广泛应用于机械臂的柔顺控制和人机交互场景，主要包括以下应用：

- **力控操作**：在装配、打磨等任务中，通过导纳控制实现精确的力控制。
- **人机协作**：在协作机器人中，通过导纳控制实现安全的人机交互。
- **环境适应**：在未知环境中，通过导纳控制实现机械臂的自适应运动。

#### 10.1.2. 导纳控制的优势

以下是导纳控制在机械臂控制中的主要优势：

优势	描述
柔顺性	通过力-位置映射实现柔顺控制，适应外部力的作用。
安全性	在人机协作中，通过导纳控制避免机械臂对操作者造成伤害。
适应性	在未知环境中，通过动态响应调节实现自适应运动。
精确性	在力控操作中，通过导纳控制实现高精度的力控制。

#### 10.1.3. 导纳控制的核心公式

导纳控制的核心公式如下：

$$F = M\ddot{x} + D\dot{x} + Kx$$

其中：

- $F$ ：外部力。
- $M$ ：虚拟质量。
- $D$ ：虚拟阻尼。
- $K$ ：虚拟刚度。
- $x$ ：位置偏差。

通过调整  $M$ 、 $D$ 、 $K$  参数，可以控制机械臂的动态响应特性。

### 10.1.4. 导纳控制与阻抗控制的对比

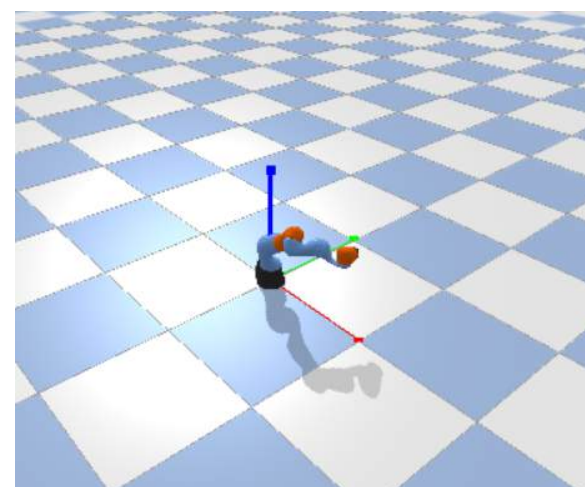
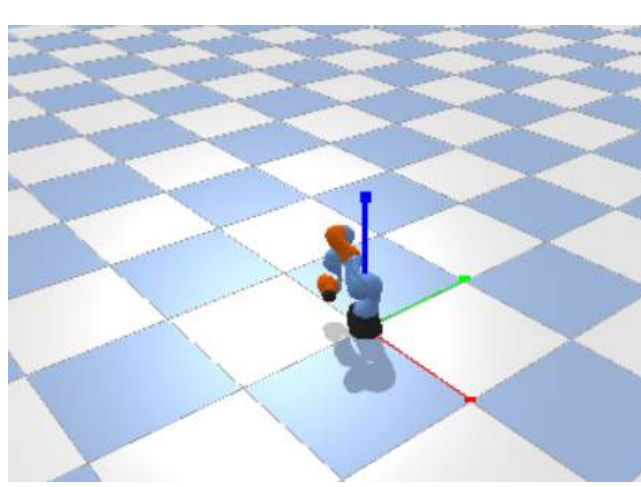
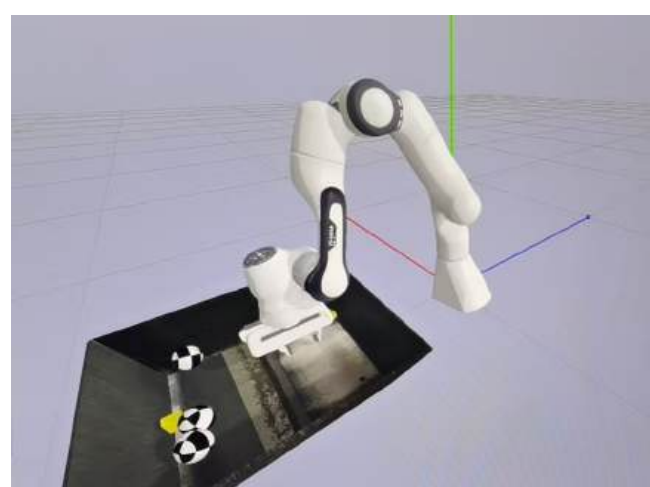
以下是导纳控制与阻抗控制的对比：

特性	导纳控制	阻抗控制
控制对象	位置或速度	力
适用场景	需要柔顺性和安全性的场景（如人机协作）。	需要精确力控制的场景（如装配、打磨）。
动态响应	通过调整导纳参数实现动态响应调节。	通过调整阻抗参数实现动态响应调节。
实现复杂度	相对简单，适合实时控制。	相对复杂，需要高精度力传感器。

- **导纳控制** 是一种基于力-位置关系的控制方法，广泛应用于机械臂的柔顺控制和人机交互场景。
- 通过调整导纳参数，可以实现机械臂的动态响应调节，提高其柔顺性、安全性和适应性。
- 导纳控制在力控操作和人机协作中表现出显著优势，是机械臂控制的重要技术之一。

## 10.2. 实操结果

pybullet



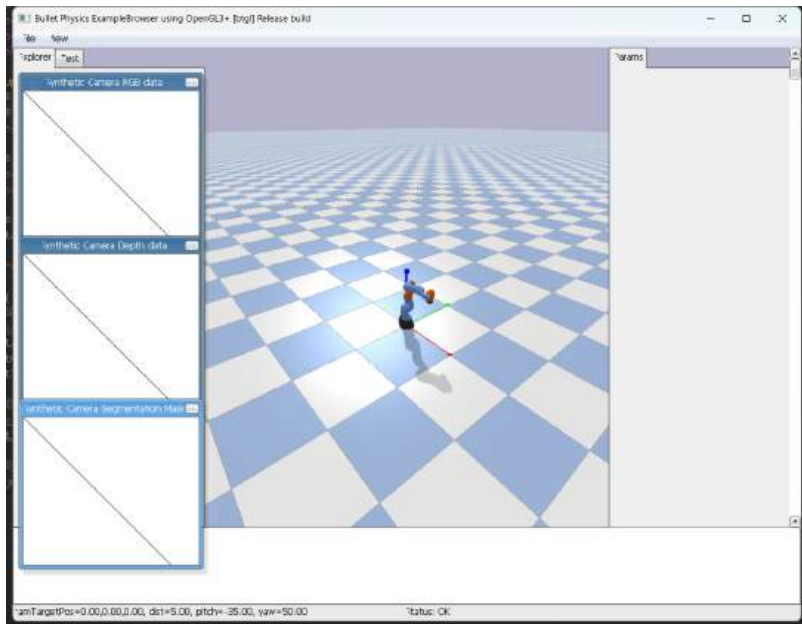
具备一定抗扰能力

## 10.3. 作业

### 部分代码

```
1 class TorqueAdmittanceController:
2     def __init__(self, j, b, k, dt):
3         self.J = j # 虚拟转动惯量
4         self.B = b # 虚拟阻尼
5         self.K = k # 虚拟刚度
6         self.dt = dt # 时间步长
7
8         # 状态变量
9         self.theta = 0.0 # 角度偏差
10        self.dtheta = 0.0 # 角速度
11        self.ddtheta = 0.0 # 角加速度
12
13    def update(self, measured_torque, desired_angle):
14        # 导纳控制方程:  $\tau = J * \ddot{\theta} + B * \dot{\theta} + K * \theta$ 
15        # 求解  $\ddot{\theta}$ 
16        self.ddtheta = (measured_torque - self.B * self.dtheta - self.K * self.theta) / self.J
17
18        # 更新状态
19        self.dtheta += self.ddtheta * self.dt
20        self.theta += self.dtheta * self.dt
21
22        # 计算目标角度 (期望角度 + 偏差)
23        target_angle = desired_angle + self.theta
24
25        return target_angle
```

## 运行结果



# 11. MPC与WBC在双足机器人控制中的应用与优势

## 11.1. MPC与WBC算法简介

### 11.1.1. MPC (Model Predictive Control, 模型预测控制)

MPC 是一种基于模型的最优控制方法，通过预测未来一段时间内的系统行为，优化当前控制输入。其核心特点包括：

- **预测模型**：利用系统动力学模型预测未来状态。
- **滚动优化**：在每个控制周期内求解优化问题，生成最优控制输入。
- **反馈校正**：通过实时反馈校正预测误差，提高控制精度。

### 11.1.2. WBC (Whole-Body Control, 全身控制)

WBC 是一种基于优化算法的控制方法，通过协调机器人全身关节的运动，实现复杂任务（如行走、平衡）。其核心特点包括：

- **任务优先级**：将任务分为高优先级（如平衡）和低优先级（如姿态调整），通过优化实现任务协调。
- **动力学约束**：考虑机器人动力学约束（如关节力矩限制），确保控制输入的可行性。
- **实时性**：通过高效求解器实现实时控制。

## 11.2. MPC与WBC在双足机器人控制中的应用

### 11.2.1. MPC的应用

- **步态生成**：通过预测未来状态，生成稳定的步态轨迹。
- **平衡控制**：在外部扰动下，通过滚动优化实现动态平衡。
- **路径跟踪**：在复杂环境中，通过MPC实现精确的路径跟踪。

### 11.2.2. WBC的应用

- **全身协调**：通过任务优先级和动力学约束，实现全身关节的协调运动。
- **复杂任务**：支持多任务并行（如行走、抓取、避障），提高机器人灵活性。
- **实时控制**：通过高效求解器实现实时全身控制。

## 11.3. MPC与WBC的对比与优势

以下是MPC与WBC在双足机器人控制中的对比与优势总结：

特性	MPC	WBC
核心思想	基于模型预测和滚动优化，实现最优控制。	基于任务优先级和动力学约束，实现全身协调控制。
适用场景	步态生成、平衡控制、路径跟踪。	全身协调、复杂任务、实时控制。
计算复杂度	较高，需在线求解优化问题。	较高，但可通过高效求解器实现实时控制。
优势	- 预测未来状态，提高控制精度 - 适应动态环境变化	- 支持多任务并行 - 考虑全身动力学约束



## 11.4. 使用Webots进行算法可视化结果的优势

Webots 是一款开源的机器人仿真平台，支持多种机器人模型和传感器仿真。使用Webots进行MPC和WBC算法可视化结果的优势包括：

优势	描述
快速验证	通过仿真快速验证算法性能，减少硬件调试时间。
可视化调试	提供丰富的可视化工具，便于分析算法效果和问题定位。
多场景支持	支持多种环境（如室内、室外、复杂地形），验证算法的鲁棒性。
低成本	无需硬件设备，降低开发成本。
可扩展性	支持自定义机器人模型和控制算法，便于扩展和二次开发。

- MPC 和 WBC 是双足机器人控制中的两种重要算法，分别适用于步态生成、平衡控制和全身协调任务。
- Webots 提供了强大的仿真和可视化功能，能够显著提高算法开发和调试的效率。
- 通过仿真验证和可视化分析，可以快速优化算法性能，为实际硬件部署奠定基础。

## 11.5. 实操过程

下载webots安装包。注意版本需要是 R2021a: <https://github.com/cyberbotics/webots/releases>

新建环境变量 `WEBOTS_HOME`，变量值为 Webots 安装的文件夹，例如此处是 E 盘的 Webots 文件夹 `E:\Webots`

新建环境变量 `PYTHONPATH`，变量值为 `%WEBOTS_HOME%\lib\controller\python39`

将路径 `%WEBOTS_HOME%\lib\controller` 添加到 `PATH` 环境变量中

打开 `Windows Powershell`，使用下列命令进行验证，确保环境变量正确配置

注意不是CMD, 而是Powershell, 因为Powershell权限会更高

```
1 echo $env:WEBOTS_HOME
2 echo $env:PYTHONPATH
3 $env:Path -split ';' | Select-String "webots"
```

### 配置环境

```
1 conda create -n MPC-WBC-Python-Demo python=3.9.18
2 conda activate MPC-WBC-Python-Demo
3 conda install pinocchio=3.3.1 -c https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge/
4 conda install cvxopt=1.3.0 -c https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
5 conda install matplotlib=3.9.2 -c https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/mai
```

### 关键参数分享

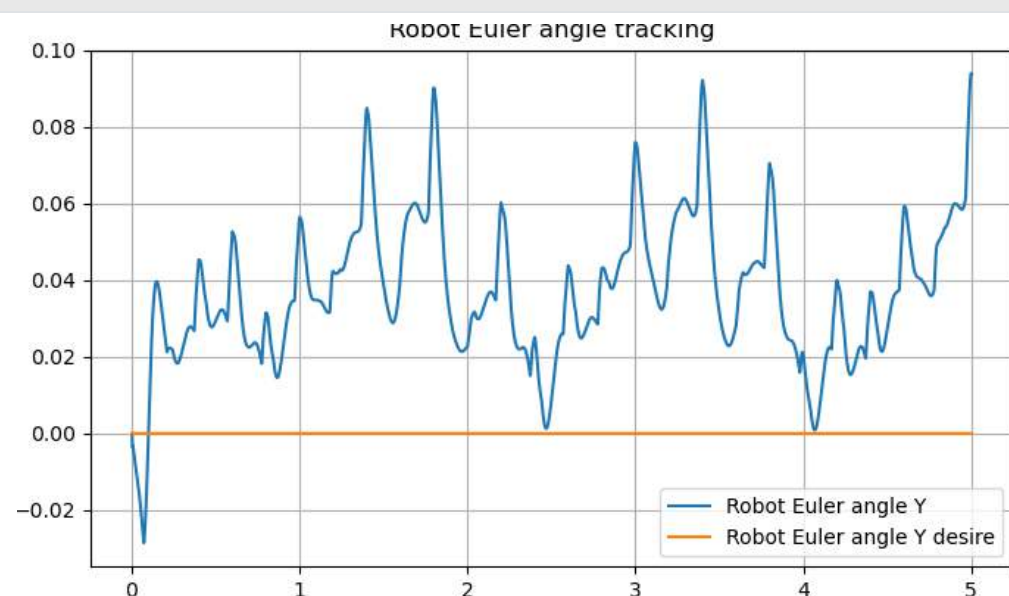
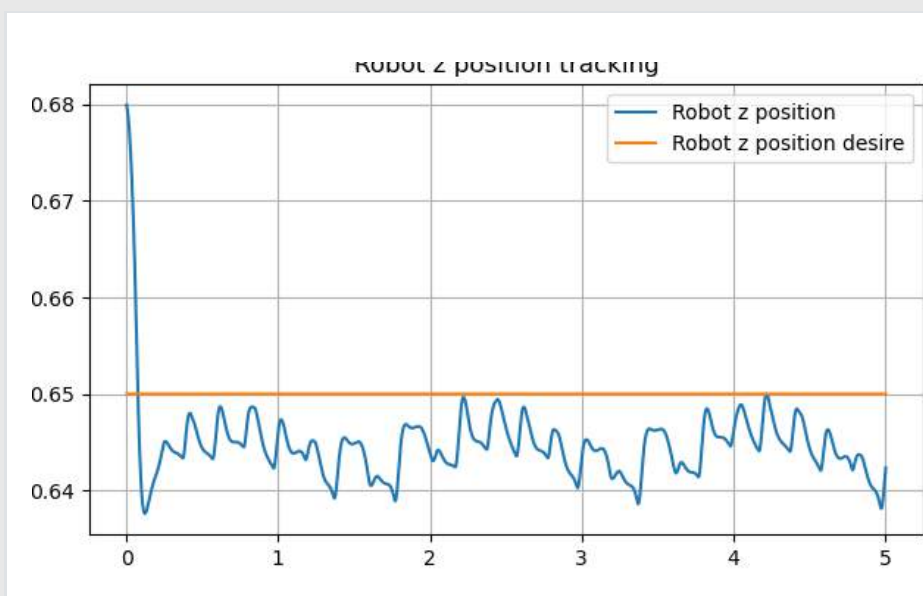
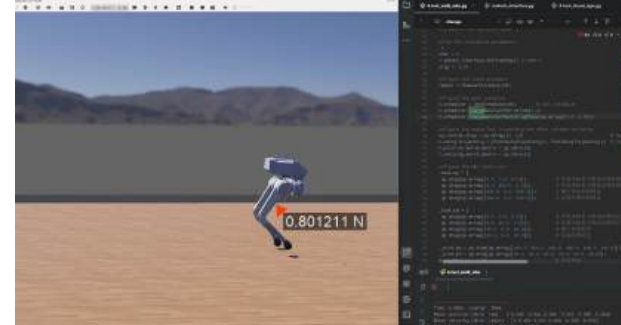
```
1 # Configure the gait scheduler
2 gait_scheduler = GaitScheduler(dt) # Gait scheduler
3 gait_scheduler.ChangeWalkGaitPeriodTime(0.4)
4 gait_scheduler.ChangeWalkGaitSwitchingPhase(np.array([0.6, 0.6]))
5
6 # Configure the swing foot trajectory and other related variables
7 swing_initial_flag = np.array([0, 0]) # Used to determine whether the swing phase is just
   entered
8 foot_swing_trajectory = [FootSwingTrajectory(), FootSwingTrajectory()] # Swing foot trajectory generators, they are
   independent of each other
9 foot_position_world_desire = np.zeros(6) # Swing foot expected trajectory
10 foot_velocity_world_desire = np.zeros(6)
11
12 # Configure the wbc controller
13 wbc_task_kp = [
14     np.diag(np.array([0.0, 0.0, 0.0])), # 浮基坐标系下的角动量控制增益
15     np.diag(np.array([0.0, 400.0, 0.0])), # 浮基坐标系下的线动量控制增益
16     np.diag(np.array([400.0, 0.0, 600.0])), # 躯干姿态控制增益
17     np.diag(np.array([500.0, 0.0, 500.0])) # 足端位置控制增益
18 ]
19 wbc_task_kd = [
20     np.diag(np.array([0.0, 0.0, 0.0])), # 浮基坐标系下的角动量阻尼
```

```

21     np.diag(np.array([0.0, 40.0, 0.0])),           # 浮基坐标系下的线动量阻尼
22     np.diag(np.array([40.0, 0.0, 40.0])),       # 躯干姿态阻尼
23     np.diag(np.array([20.0, 0.0, 20.0]))       # 足端位置阻尼
24 ]
25 wbc_joint_kp = np.diag(np.array([400.0, 400.0, 400.0, 400.0, 400.0, 400.0])) # 关节位置控制增益
26 wbc_joint_kd = np.diag(np.array([20.0, 20.0, 20.0, 20.0, 20.0, 20.0])) # 关节速度阻尼
27 step_length_k_p = 0.08 # 步长增益
28 wbc_controller = WBCController(wbc_task_kp, wbc_task_kd, wbc_joint_kp, wbc_joint_kd)

```

## 实现效果



## 2025/1/20 实验一操作步骤

```

1 conda create -n ur_grasp python=3.8
2 conda activate ur_grasp

```

### 能翻墙

```
git clone https://github.com/JeroenOudeVrielink/ur5-robotic-grasping
```

### 不能翻墙

```
git clone https://gitclone.com/github.com/JeroenOudeVrielink/ur5-robotic-grasping
```

### 下载依赖

```

1 cd ur5-robotic-grasping
2 pip config set global.index-url https://mirrors.tuna.tsinghua.edu.cn/pypi/web/simple
3 pip install -r requirements.txt

```

## 修改代码

### 打开要修改的文件

也可以直接用我在群里发的文件，直接替换这个 `grasp.py`  
 gedit是直接用文本文档编辑，十分适合新手的编辑指令



```
sudo gedit network/utils/dataset_processing/grasp.py
```

## 第一处

找到

```
).astype(np.float))
```

替换为

```
).astype(np.float64))
```

```
387 [y2 - self.width / 2 * xo, x2 - self.width / 2 * yo],
388 [y2 + self.width / 2 * xo, x2 + self.width / 2 * yo],
389 [y1 + self.width / 2 * xo, x1 + self.width / 2 * yo],
390 ).astype(np.float))
391
392 def max_iou(self, grs):
393     """
394     Return maximum IOU between self and a list of GraspRectangles
395     """
```

## 第二处

找到

```
ax.plot(points[:, 1], points[:, 0], color=color, linewidth=3)
```

替换为

```
ax.plot(points[:, 1], points[:, 0], color=color, linewidth=3)
```

```
326 self.points = factor
327
328 def plot(self, ax, q, color=None):
329     """
330     Plot grasping rectangle.
331     :param ax: existing matplotlib axis
332     :param q: Grasp quality
333     :param color: matplotlib color code (optional)
334     """
335     points = np.vstack((self.points, self.points[0]))
336     ax.plot(points[:, 1], points[:, 0], color=color, linewidth=3)
337     ax.plot(self.center[1], self.center[0], 'o')
338     ax.legend(['score: {0:.2f}'.format(q)])
339
340 def zoom(self, factor, center):
341     """
```

## 第三处

找到

```
return self.points.mean(axis=0).astype(np.int)
```

替换为

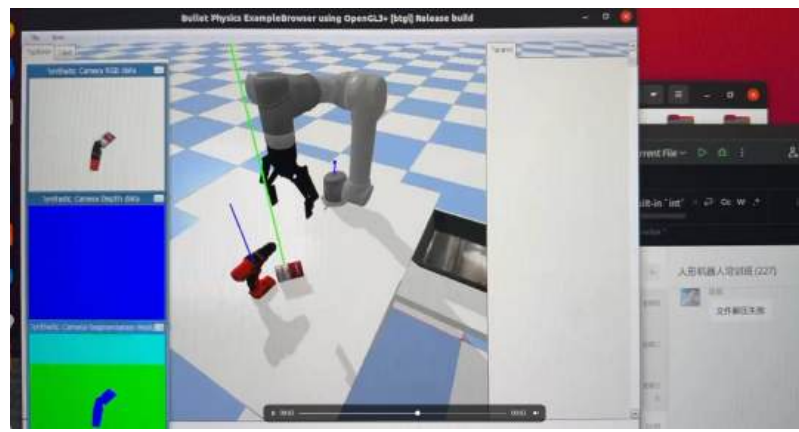
```
return self.points.mean(axis=0).astype(int)
```

```
221 """
222 return Grasp(self.center, self.angle, self.length, self.width)
223
224 @property
225 def center(self):
226     """
227     :return: Rectangle center point
228     """
229     return self.points.mean(axis=0).astype(np.int) # use Python's built-in 'int'
230
231 @property
232 def length(self):
233     """
234     :return: Rectangle length (i.e. along the axis of the grasp)
235     """
```

## 运行

```
python demo.py --scenario=pack --runs=1 --show-network-output=False
```

## 结果



## 2025/1/20 实验二操作步骤

- 1 `conda create -n unidegrasp python=3.8 -y`
- 2 `conda activate unidegrasp`

安装CUDA11.3, 官方下载地址<https://developer.nvidia.com/cuda-11.3.0-download-archive>

### CUDA Toolkit 11.3 Downloads

Select Target Platform  
Click on the green buttons that describe your target platform. Only supported platforms will be shown. By downloading and using the software, you agree to fully comply with the terms and conditions of the [CUDA EULA](#).

Operating System	Linux	Windows					
Architecture	x86_64	ppc64le	arm64-sbsa				
Distribution	CentOS	Debian	Fedora	OpenSUSE	RHEL	SLES	Ubuntu
Version	16.04	18.04	20.04				
Installer Type	deb (local)	deb (network)	runfile (local)				

Download installer for Linux Ubuntu 20.04 x86\_64

The base installer is available for download below.

> Base installer

Installation instructions

```
$ wget https://developer.download.nvidia.com/compute/cuda/11.3.0/local_installers/cuda_11.3.0_465.19.01_linux.run
$ sudo sh cuda_11.3.0_465.19.01_linux.run
```

在终端里面输入下面两条指令

- 1 `wget https://developer.download.nvidia.com/compute/cuda/11.3.0/local_installers/cuda_11.3.0_465.19.01_linux.run`
- 2 `sudo sh cuda_11.3.0_465.19.01_linux.run`

如果此时终端输入`nvcc -V`

发现没有CUDA11.3或者版本不对, 请输入以下这条指令添加CUDA11.3进系统环境

```
export PATH=/usr/local/cuda-11.3/bin:$PATH
```

安装对应版本pytorch

- 1 `conda install pytorch==1.10.0 torchvision==0.11.0 torchaudio==0.10.0 -c https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/pytorch/linux-64/ -y`
- 2 `conda install -y https://mirrors.bfsu.edu.cn/anaconda/cloud/pytorch3d/linux-64/pytorch3d-0.6.2-py38_cu113_py1100.tar.bz2`
- 3 `pip install tensorboard==2.11.2`
- 4 `pip install yyaml`

安装本地pip包

- 1 `cd dexgrasp_generation/dexgrasp_generation/thirdparty/pytorch_kinematics`
- 2 `pip install -e .`
- 3 `cd ../nflows`
- 4 `pip install -e .`
- 5 `cd ../CSDF`
- 6 `pip install -e .`
- 7 `cd ../..`

如果报缺失什么, 就`pip install xxx`安装那个包

修改`visualize_result_l20.py`文件的153行, 改成自己的目录

```
sudo gedit tests/visualize_result_l20.py
```

```
140 #         cta_hand_pose = [
141 #             torch.tensor([tx, ty, tz, rx, ry, rz, q1, q2, q3, ...]),
142 #             torch.tensor([tx, ty, tz, rx, ry, rz, q1, q2, q3, ...]),
143 #             ...
144 #         ]
145 #     }
146
147 # hand model
148 builder = L20HandBuilder()
149
150 # object mesh
151 object_code = result['object_code'][args.num]
152 object_scale = result['scale'][args.num].item()
153 object_mesh = tm.load(os.path.join('/home/mike/UniDexGrasp/dexgrasp_generation/dexgrasp_generation/data/mjc/meshes/'))
154 object_pc = result['canon_obj_pc' if args.canonical_frame else 'obj_pc'][args.num].numpy()
155 object_pc_plotly = go.Scatter3d(
156     x=object_pc[:, 0],
157     y=object_pc[:, 1],
158     z=object_pc[:, 2],
159     mode='markers',
160     marker=dict{
```

## 运行代码

```
python ./tests/visualize_result_l20.py --exp_dir "eval_mypc" --num 2
```

## 结果

### 显示抓取位置

