Assignment 4 (12pt)

In this assignment you will need to complete the combined task consisting a quadruped robot and a humanoid robot.

Environment

You can install the environment as follows:

```
conda create -n hw4 python=3.10
conda activate hw4
conda install conda-forge::roboticstoolbox-python==1.0.3

# if you can use Nvidia GPU
conda install pytorch==2.5.1 torchvision==0.20.1 torchaudio==2.5.1 pytorch-cuda=12.4 -c pytorch -c nvidia
# you might need to change the cuda version here
# if you can only use CPU
conda install pytorch==2.5.1 torchvision==0.20.1 torchaudio==2.5.1 cpuonly -c pytorch

pip install tqdm h5py wandb plotly pygments ikpy transforms3d pyyaml pillow matplotlib trimesh dm_control
pip install numpy==1.26.4 scipy==1.11.4 sympy==1.13.1 opencv-python==4.11.0.86
pip install onnxruntime pyapriltags
```

Simulation Task Details

The task requires you to implement the policies both for the quadruped robot (go2) and the humanoid robot (galbot). The whole pipeline consists of three steps: 1.guiding the quadruped to a position near table, 2.controlling the humanoid to grasp the object (a driller) and drop in the box which is on the quadruped's back and 3.guiding the quadruped going back to its initial position.

An apriltag is attached on quadruped's head for location. You don't need to implement the policy of quadruped's locomotion. Instead, you just need to give command of its velocity in xy plane and angular velocity around z axis. For tracking the apriltag in case it runs out of sight, you are allowed to adjust the humanoid head pose by controlling the qpos of head joints.

The grasping driller is almost the same as Assignment 2. But you cannot directly set the arm qpos to pre-grasp state, since it is not feasible in real world. Please implement a whole policy of planning to grasp and lift, and moving the driller to drop it in the box. Note that the simulation environment this time is different from that in Assignment2, and your may have to generate your own dataset and re-train the detection network if it no longer works.

The simulation task will judge your whole policy by three metrics. See metrics below for details.

We have provided a demo code for the simulation guidance, you can run `python demo.py` to see the demo scene. The main code template is in `main.py`. You will implement your code here.

Full pipeline

We have separate the task into 5 steps in `main.py`

initialization

Upon the simulation environment is launched, you may need to assign some initialization. The `head_init_qpos` in `main.py` is used to move humanoid head to track the quadruped. The `observing_qpos` is used to initialize the humanoid's arm for taking rgbd image of the driller.

step1: move quadruped to dropping position

You will detect the Apriltag using d435 camera on humanoid's head, and calculate the pose of tag and the box. The relative pose from tag to box is given in `main.py`.
Then you use the pose of box to guide the quadruped to your desired dropping position. If you worry about losing track of the Apriltag, you can change `head_qpos` when the quadruped is walking.

step2: detect driller pose

After the quadruped stops at dropping position, you need to detect driller pose. Metric I is judged when you finish detection. Pay attention, the camera used in this Assignment has resolution of 1280*720, which is different from Assigment 2.

step3: plan grasp and lift

When you acquired driller pose, we have provided some feasible grasping pose in object frame (offcourse you can use your own design). Use calculated grasping pose to plan grasp and lift trajectory and execute them.

step4: plan to move and drop

Using the grasp plan result and the detected box pose in step1, plan a moving trajectory and drop the driller into box. Since the workspace is limited, you have better choose the dropping pose of gripper. The gripper can be dropped in air and it will not break.
If the driller falls into the box, the simulation will record the Metric II as true. In this situation, the driller will attach to the box. So, do not worry about driller poping out of box when the quadruped is walking.

step5: move quadruped backward to initial position

This step is similar to step1. You can copy and change the code of step1 to guide the quadruped back to its initial position. After all the steps is completed, Metric III will judge whether the quadruped returns precisely.

Metric I: Object Pose Estimation (4pt)

You are required to estimate the pose of driller which is placed on the table. The ground truth of driller is the pose of the object in simulation. You can access only rgb and depth of the wrist camera. In `main.py`, `Metric['obj_pose']` will judge the precision of pose estimation.

You will get one score if the mean translation error is smaller than 0.025 m and the mean rotation error is smaller than 0.25 rad. There will be four test cases.

Metric II: Dropping Precision (4pt)

The dropping precision metric is whether the humanoid drops the driller into the box. The process is judged success if the position of driller enters the box area in simulation. In `main.py`, `Metric['drop_precision']` will judge the precision of pose estimation.

You will get one score for each success case. There will be four test cases.

Metric III: Quadruped Returning (4pt)

This metric will detect whether the quadruped robot take the box to the initial position. The judging is triggered after the dropping success. In `main.py`, `Metric['quad_return']` will judge the precision of pose estimation. The process is judged success if the box's position distance from its initial position is smaller than 0.1m.

You will get one score for each success case. There will be four test cases.

Precautions

You are NOT allowed to: (1) read the pose of object in simulation, (2) read the pose of quadruped robot or the box in the simulation, (3) diretly set pose of object, set pose of robot root or set qpos of robot in simulation

Testing & Uploading

We provided four example testing cases in simulation, the final testing cases will not vary far from these. For testing, run `python main.py --test_id id` to get your task metrics. Id can be chosen from [0,1,2,3].

For uploading your homework, zip `main.py` as `{your_name}_{your_id}.zip` and upload to the website.