I. Folder Description

remote_control-x86-can-v2

This folder contains the low-level driver code for the robot, including the drivers and control logic for two master arms and two follower arms. The system is modified from the cobo_magic platform and is divided into two parts: the **robot side** and the **teleoperation side**, although both share the same codebase.

Mechanical

This folder contains the mechanical drawings of the robot and the experimental platform, provided for reference.

script

This folder contains the main script programs for the project. See the following sections for detailed descriptions.

LLM_agent

This folder contains the LLM (Large Language Model) agent scripts used for data analysis.

II. Program Dependencies

- Main Program: `actiongui.py` (Main interface program, used to start the robot, record trajectories, path points, and execute automated processes)
  - Dependency 1: `robot.py` (Robot startup node, gripper control, etc.)
    - Dependency 1.1: `claw.py` (Gripper control)
  - Dependency 2: `gun.py` (Robotic pipette control)
  - Dependency 3: `gui1.py` (Base class for the main interface program)
    - Dependency 3.1: `rosbagrecord.py` (Used for trajectory recording)
  - Dependency 4: `BLScontroller.py` (Tracking control)
  - Dependency 5: `GunPntRcord.py` (Records points into trajectory)
  - Dependency 6: `runGunPose2B.py` (Point-to-point control)
- Tool Programs: `ysmooth.py` (Trajectory optimization)

Note: This program relies on a customized robotic system and its software and hardware configuration. The system is re-made based on the cobo_magic platform from AgileX Robotics (known as "松灵机器人"). Therefore, it may not run properly without the corresponding robotic system.Users can refer to the algorithms and the architecture of it to build their own systems for reproduction.

III. System Usage

(A) Configuration

• Robot Host is configured as the ROS Master, with IP address set to:

192.168.1.5 (example)

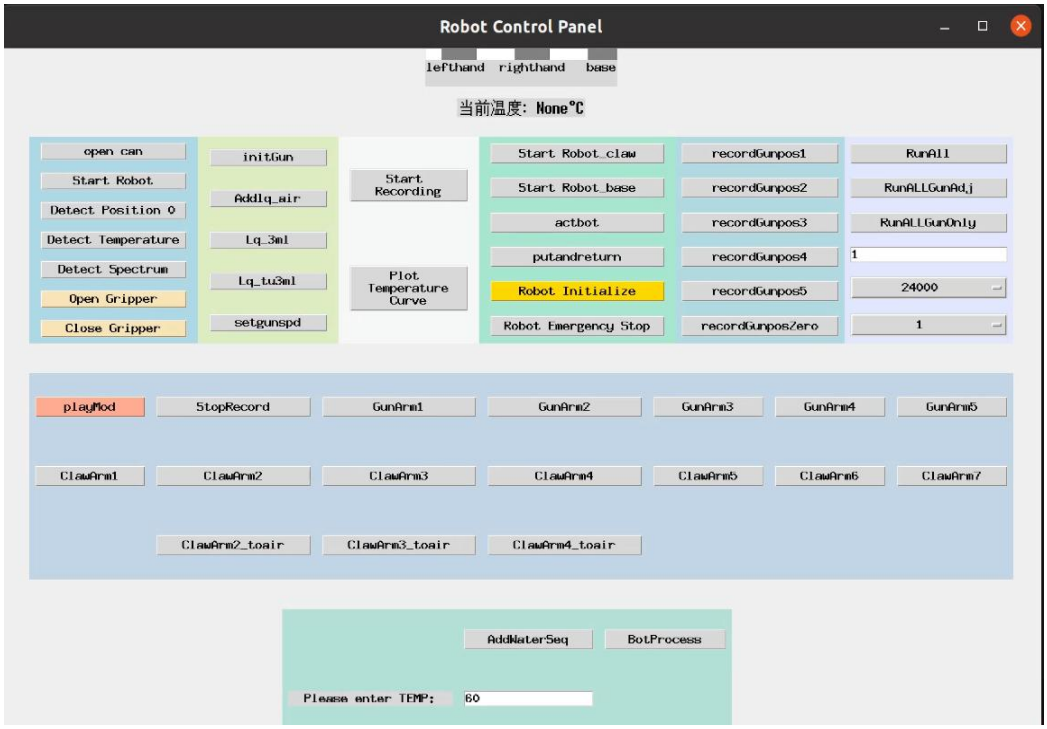
• Teleoperation Host is configured as the ROS Slave, with IP address set to:

192.168.1.7 (example)

(B) Teaching (Operation)

1、Launch the program actiongui.py on the robot side.

The interface is described as follows:



After entering the above interface, the interface description is as follows:

The "Open Can" button in the top-left corner, when clicked, will execute the following procedure:

```
def open_can():
    print("Opening CAN")
    """运行给定路径的 shell 脚本"""
    script_path = "../remote_control-x86-can-v2/tools/can.sh"
    # 对于 GNOME Terminal，你可以使用如下命令
    gnome_command = ["gnome-terminal", "--", "bash", "-c", f"{script_path}; exec bash"]
    subprocess.Popen(gnome_command)
    print("CAN is openned")
```

——Its purpose is to execute the script ../remote_control-x86-can-v2/tools/can.sh, which is used to initiate the CAN connection for the robotic arm.

- The "Start Robot" button, located as the second button in the top-left corner, will execute the

following procedure when clicked:

```
def start_robot():
    global process_robotstart
    print("Starting robot")
    script_path = "../remote_control-x86-can-v2/tools/jgl_2follower.sh"
    gnome_command =  ["gnome-terminal", "--", "bash", "-c",
f"{script_path}; exec bash"]
    process_robotstart= subprocess.run(["bash", script_path])
    time.sleep(2)
    if process_robotstart is not None and process_robotstart.poll() is None:
        print("robot is started")
```

——Its purpose is to execute the script `../remote_control-x86-can-v2/tools/jgl_2follower.sh`, which is used to launch two follower robotic arms.

——Through the above steps, both robotic arms are started and ready to receive control commands.

2. Start the Program on the remote Side

Execute the script `../remote_control-x86-can-v2/tools/jgl_Master.sh`, which contains the following content. Purpose: To start the two teleoperation master arms.

```
#!/bin/bash
workspace=$(pwd)
gnome-terminal -t "master1" -- bash -c "cd ${workspace}/master1; source
devel/setup.bash && roslaunch arm_control arx5.launch; exec bash;"
sleep 1
gnome-terminal -t "master2" -- bash -c "cd ${workspace}/master2; source
devel/setup.bash && roslaunch arm_control arx5.launch; exec bash;"
```

3. Teleoperation Recording

In the main interface of `actiongui.py`, the central area is used to record the teleoperated trajectory and actions of the robot. This area provides record/playback for 5 actions of the left gun_arm and 7 actions of the right claw_arm, respectively. It is recommended to decompose experimental operations into smaller segments for easier switching and combination.

For example, a liquid aspiration task can be decomposed into the following steps:

(1). Move from the initial position to the bottle mouth

(2). Insert into the bottle

(3). Aspirate and retract

(4). Move to the injection position

(5). Injection

(6). Return to the initial position

Among these, steps (1), (4), and (6) can be taught via human teleoperation. Steps (2) and (3)can use fixed motions based on inverse kinematics, such as vertical movement, to ensure accuracy and avoid collisions.

4. Trajectory Optimization and Control

After recording different actions of the experiment, the `ysmooth.py` script can be used for trajectory optimization. The optimized `rosbag` file can then be used directly for playback-based control.

For actions requiring stability, the control method provided in `BLScontroller.py` (tracking control) can be applied to suppress disturbances and improve robustness.

5. Experimental Workflow

In `actiongui.py`, we provide two example implementations of experimental workflows:

- `RunAll_1217(self)` corresponds to a workflow using tracking control.
- `RunAll(self)` corresponds to a workflow using direct rosbag playback for control.

Since both the actions and execution logic are modularized, we plan to further integrate graphical programming, LLM assistance, and other methods in the future to enable more user-friendly experimental workflow configurations.