Data collection documentation

Usage

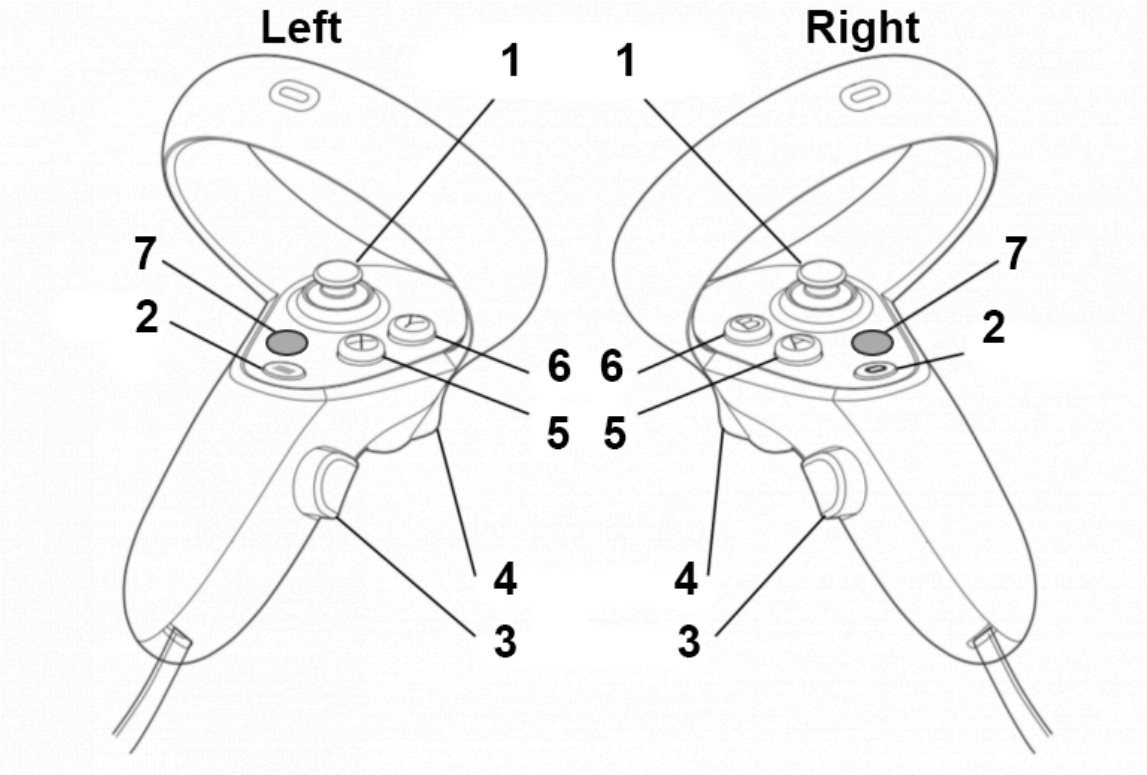
Hardware list

- Quest2/3 + controller
- Ubuntu 20.04 laptop
- Fetch robot

Steps

1. Inside the quest headset, there is an app `Teleoperator`. Open it
2. Setup fetch:
    a. `ssh fetch@fetch48.local` open two terminal
    b. In the first terminal, run the necessary drivers by `roslaunch low_level_planning fetch_control_drivers.launch`
    c. Go to the path of `RobotIL` and run the controller `python RobotIL/core/fetch_robot/fetch_controller.py [--light_data_collection]`
    d. Plugin the hard drive on head (if you want to save data directly from fetch)
    e. For more arguments and details, please read the README on GitHub
3. Make sure the robot state estimation is correct. In the host machine, run `set_rosmaster_fetch` and check by `roslaunch low_level_planning rviz.launch`.
4. On host machine, launch the main data collection script: `robotil-record --task_name ${task_name} [--use_gazebo] [--controller_index 0] [--num_episodes 35] [--light_data_collection]`
5. For install, train, and inference, please read the README on GitHub

Controller button combination



- The end effector follows your hand and record data when holding 3
- 3 + 4 to close robot gripper
- 5 to stop and save the data
- 6 to stop and reset

Add new task

Some notes

- When collecting the data, don't move too fast
- When collecting the data, stay as close to the headset as possible
- There is no synchronization, but try to keep the VR controller in the same rotation with the end effector will make human easier to control the robot arm
- The argument `-light_data_collection` is used when you want to save the rgb and depth directly from fetch. During our test, passing the rgb and depth through ros will significantly increase the network traffic and lead to a high delay (around 0.8s). A good practice is don't use this argument when finding the pre-pose (to make sure the robot could see an informative scene) and enable it when collecting the data.
- Inference using the desktop with 4090 will have a higher success rate due to the low compute delay.
- If the fetch joystick cannot connect to the robot, you can try `sudo systemctl restart robot`.

Q&A

Where to modify the path to save depth and image?
https://github.com/RobotIL-rls/RobotIL/blob/a0b7d288b24449934df29f2ca95f0f9b60e88b67/RobotIL/core/fetch_robot/fetch_controller.py#L300

Performance

Just by increasing the data quality, open the fridge success rate: 70% -> 90%
By changing to predicting end effector pose, success rate: 100%

Data format

- Depth Images
- RGB Images
- acml_poses.npy: list of position and rotation
- controller_btn_states.npy: controller button states
- frequency.npy: frequency for saving data
- joint_states.npy: list of joint states
- abs_ee_pose.npy: list of absolute end effector poses under robot frame
- delta_ee_pose.npy: list of delta end effector poses, details are in next section
- point_clouds.npy: array of point cloud for each frame
- stop_signal.npy: 00000…0001 with the length equals to `len(acml_pose)`

Development

Access the source code here

Fetch

How it works

When the fetch robot is opened, it will run `robot.service` by default. The service will run the `~/start_script.sh` and launch the `fetch_bringup.launch` to load the necessary drivers and controllers. The `start_script.sh` will source the default ros setup.bash and `catkin_ws/devel/setup.bash`.
Do not modify the `start_script.sh` unless you know what you are doing.

How the current pd controller works

The PD controller is implemented in `~/catkin_ws/src/robot_controllers` ros package. It operates by computing the velocity commands for the robot's joints based on the error between the desired and actual end-effector poses. It uses proportional and derivative

gains to adjust joint velocities dynamically, enabling the end-effector to track a target trajectory smoothly and precisely in Cartesian space.

How to tune the parameters

Tune the parameters in `fetch_bringup/config/default_controllers.yaml`:

```
cartesian_pose_velocity_controller:
  type: "position_controllers/CartesianPoseVelController"
  fb_trans:
        p: 0.8   # Proportional gain for translation
        i: 0.0  # Integral gain for translation
        d: 0.1  # Derivative gain for translation
  fb_rot:
        p: 0.5  # Proportional gain for rotation
        i: 0.0  # Integral gain for rotation
        d: 0.1   # Derivative gain for rotation
```

Some notes

- There are some issues with the emergency button, now we need to keep pressing it until you are sure it is safe.

Possible improvements:

- [ ] Implement a more general quadratic programming controller
- [ ] We can save all the velocity data and effort data, and explore the policy/control over those values
- [ ] Add a velocity limit for the joints/EEF.

VR Controller

Source code: Teleoperator.zip

How the delta pose works

Check it here

The Unity app

Open the source code in unity hub. Please use Unity `2022.3.47f1`
Connect the headset to computer, build and run

Possible improvements

- [ ] Now the target ip address is hard coded in the unity app. Maybe we can make it configurable in headset

System pipeline

Possible improvements:

- [ ] Whole body control
- [ ] Save current pose as task's pre-pose by a controller key combination
- [ ] The task pre-pose data should only take role in host machine, so that we do not need to modify the fetch controller repo every time
- [ ] Add a debugging feature. Visualize the trajectory predicted by policy before running the step.
- [ ] Currently we hardcode the path to save data in the fetch. Maybe we can make it as an argument