Workspaces

There are 2 workspaces necessary to successfully launch all the required nodes: ws_moveit and ur5_ws. The MoveIt workspace handles all of the nodes related to the MoveIt simulation. The ur5 workspace handles all nodes related to motion planning, object detection, and ur5 drivers. These can simply exist in the home directory.

Package and Node Descriptions

launch_ur5: Handles launching of full system

- **launch/launch_ur5.launch.py:** launch file to run all necessary nodes for full system

camera: Handles the webcam

- **gather_workspace**: finds ArUco tags and objects, publishes tag distances to /table_size and object positions of msg type CollisionObject to /object_pose
    - Run with --ros-args -p port:={PORT} to specify the camera port. Defaults to 0 if no argument is passed
- **view_camera:** displays camera view

custom_interfaces: ros2 services

- **srv/GetObjectPose.srv**: service so that CollisionObject msg can be fetched from other nodes

moveit_scene_modifier: Handles objects in MoveIt planning scene

- **object_detector:** creates objects in MoveIt. This is currently only set up to create cubes but functionality can easily be changed. Subscribed to /object_pose. Publishes the average of the last 5 object poses to /planning_scene_diff
- **ur5_workspace:** creates table in MoveIt. Listens for 10 Float32MultiArray messages on /table_size then publishes to /planning_scene_diff before killing the node
- **scene_aggregator:** merges all objects in the planning scene into one contiguous scene. Ensures objects don't get deleted when others are updated. Subscribed to /planning_scene_diff and published everything to /planning_scene

object_world_state: Handles object dictionary and access service

- **object_world_state:** hold a dictionary of all objects in the workspace. Associates each pose with a string object id. Subscribes to /avg_object_pose and creates a service of type GetObjectPose on /get_object_pose. Object poses can be fetched by their string id

ur5_control: Handles UR5 motion planning/execution

- **motion_executor:** plans and executes motion to a target pose. Subscribed to /target_pose

- **pose_publisher:** user input target pose based on a desired object. Publishes pose to /target_pose. Accesses the dictionary of workspace objects via the GetObjectPose service. Input for motion planning can simply be the name of the object (i.e. “cube”)

Universal_Robots_ROS2_Driver: Contains all nodes for UR5 control via ros2

Other Files

- **single_camera_calibration:** extracts camera intrinsics based on a set of chessboard calibration images
- **get_calibration_images:** helper script to gather images used for calibration
- **freedrive:** puts the UR5 into freedrive. Locks the quaternion at [1, 0, 0, 0]. Good for calibrating the end effector (make sure it is close the the calibration tag before running).

Connect UR5 to PC (Ubuntu 24.04) via Ethernet

1. Ensure UR5 has a static IP address
    a. Navigate to “Setup Robot” → “Network”
    b. Select static address
    c. Set IP address (something like 10.168.18.17)
    d. Set Subnet mask to 255.255.255.0, leave everything else blank
2. On the PC, find the interface name for ethernet connection with **$ ip link**
    a. Ethernet cable adapter will need to be plugged in if PC does not have a built in ethernet port
    b. Name may be something like enxc8a362c504d0 and will have a DOWN status
3. On the PC, navigate to /etc/netplan
4. Make a file called 01-robot-link.yaml with **$ sudo vim 01-robot-link.yaml**
5. Paste in the following contents. Interface name should be collected from step 2 and host IP can be anything (for example 10.168.18.10/24). /24 is necessary.

```
network:
  version: 2
  renderer: networkd
  ethernets:
   {INTERFACE_NAME}:
     dhcp4: no
     addresses:
       - {HOST_IP}
     optional: true
```

6. Ensure that systemd-networkd is running with **$ sudo systemctl enable --now systemd-networkd.service**
7. Apply the network configuration with **$ sudo netplan apply**
8. If you get the permissions warning, run **$ sudo chmod 600 /etc/netplan/01-robot-link.yaml**
9. Plug in the ethernet cable and ensure the connection is UP with **$ sudo ip link set {INTERFACE_NAME} up**
10. Test the connection with **$ ping {UR5_STATIC_IP}**

Calibrate end effector

1. Manually move the UR5 close to ArUco tag ID = 1 using the teach pendant (run **$ ros2 run camera gather_workspace** and **$ ros2 run camera view_camera** to see which tag is ID 1)
2. Navigate to ur5_ws/src/util. For first time using:
    a. Make virtual environment with **$ python -m venv rtde-venv**
    b. Activate with **$ source rtde-venv/bin/activate**
    c. Install RTDE with **$ pip install ur-rtde**
3. Run **$ source rtde-venv/bin/activate**
4. Ensure mass is set to 1 kg on teach pendant
5. Run freedrive.py and move the end effector to the center of the tag (this freedrive script locks the quaternion at 1, 0, 0, 0 so the end effector is always pointed down. Seems to cause some resistance in the UR5 which is why it needs to be close to the tag before you run freedrive)
6. Gather the x and y coordinates
7. Manually enter the magnitudes into the motion_executor node via the parameter lines
8. Rebuild

Launch

1. Ensure that the External Control program is running on the UR5 via the teach pendant
    a. Navigate to “Run Program” → “File” → “Load Program”
    b. Select “externalControl.urp” and run
    c. If motion planning fails, check that the host IP in “Program Robot” → “Installation” → “External Control” is set to the Host IP from 01-robot-link.yaml
2. Run all required nodes with **$ ros2 launch launch_ur5 launch_ur5.launch.py robot_ip:={UR5_STATIC_IP} port:={CAMERA_PORT} view_camera:={TRUE/FALSE}**
    a. Launches the UR5 drivers, MoveIt simulation, motion planning nodes, camera
    b. robot_ip should be the static IP set on the ur5

    c. port should be the port of the webcam. This was made an argument as it sometimes changes when you unplug the webcam (usually is 0 or 4)
    d. view_camera determines whether the webcam view is displayed or not
3. Motion planning is active and listening on /target_pose. Run the manual pose_publisher node with **$ ros2 run ur5_control pose_publisher** to move to an object id

If needed, processes and nodes can be launched individually:

1. Spin up the UR5 driver with **$ ros2 launch ur_robot_driver ur_control.launch.py ur_type:=ur5 robot_ip:=192.168.56.101 launch_rviz:=false**
2. Launch UR5 MoveIt configuration with **$ ros2 launch ur_moveit_config ur_moveit.launch.py ur_type:=ur5 launch_rviz:=true**
3. All other nodes can be launched manually via **$ ros2 run {PKG_NAME} {NODE}**