Carrybox with Ros Vacuum Gripper Plugin

Team 6: Jiyuan(Leo) Luo, Siqi Ye, Huiwen(Ella) Xue

Agenda

1. Introduction
2. Design
3. Solutions
4. Demo
5. Q&A
6. Reference

Introduction - Project Background and Motivation

The Need for Automation:

- the pressing need for flexible, intelligent automation in sectors such as **logistics**, **warehousing**, and **manufacturing**, where **efficiency**, **cost reduction**, and **operational safety** are top priorities.
- With the **rise of e-commerce** and the increasingly complex demands placed on supply chain infrastructure, there is a **growing demand** for systems that can seamlessly adapt to dynamic workloads and diverse handling requirements.

Challenges in Current Systems:

- Conventional material-handling technologies often lack the necessary **flexibility** to handle a variety of objects without complex reconfiguration, making it challenging to maintain **throughput** and **reliability** in high-mix environments.

Project Motivation:

- Develop a system to **seamlessly adapt** to dynamic workloads.
- Address the growing demand for systems with **universal handling capabilities**.

Introduction - Project Solution Overview

System Design Goals:

- Designing a system capable of securely **grasping**, **lifting**, and **transporting** a range of box sizes and shapes with **minimal setup** and **consistent accuracy**.

Key Features:

- **Vacuum Gripper:**
  - The need for a universal handling tool
- **ROS Framework:**
  - Enhances **control**, **modularity**, and **scalability**
  - Allows the system to evolve as requirements change.

Impact:

- Improved **efficiency** and **reliability**.
- Reduced **operational costs** and enhanced **safety**.

Design - Robot Design and Implementation

Robot Arm Control:

- Utilized the **Universal Robot GitHub repository** to access the control package.
- Enabled **basic motion** and **control functionalities** for the robotic arm.

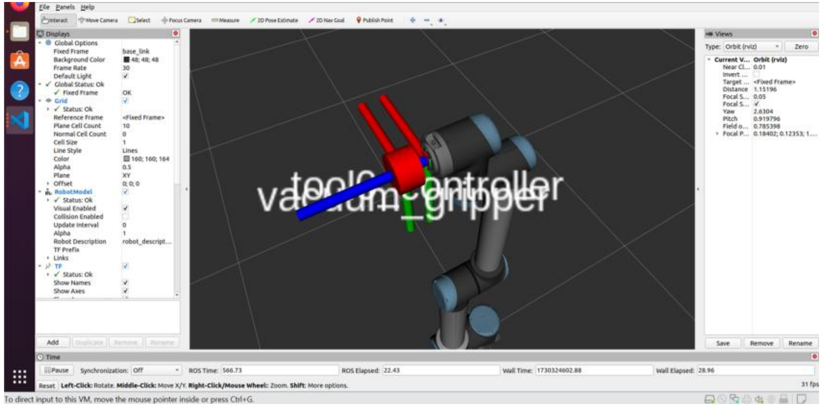
Vacuum Gripper:

- Found a **ready-to-use plugin on GitHub**, simplifying development.
- Designed a **cylindrical structure** for the gripper:
  - Connected to the end of the robotic arm.
  - Facilitated efficient grasping and transportation of boxes.

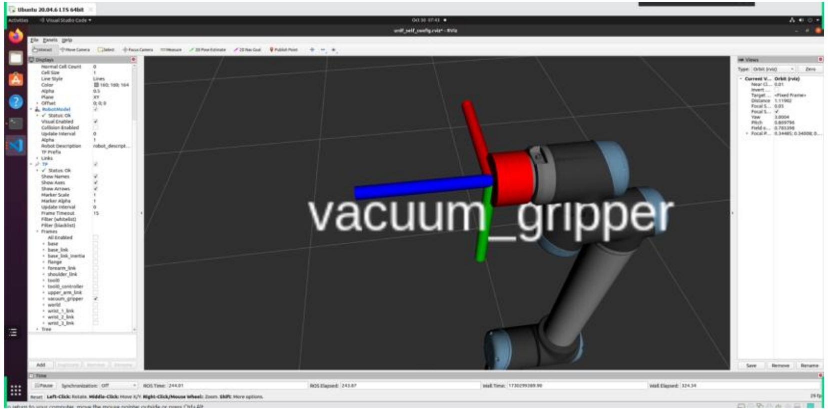
Design - Robot Design and Implementation

Offset Issue and Solution:

- Initial setup revealed a **gap** between the gripper and arm.
- Research identified the **gripper's coordinate frame** at the center of the 0.05m cylinder.
- Added a **-0.025m Z-axis offset** to align the gripper properly.
- Result: Stable connection and **accurate simulations**.



Before



After

Design - World Design and Customization

Initial Setup:

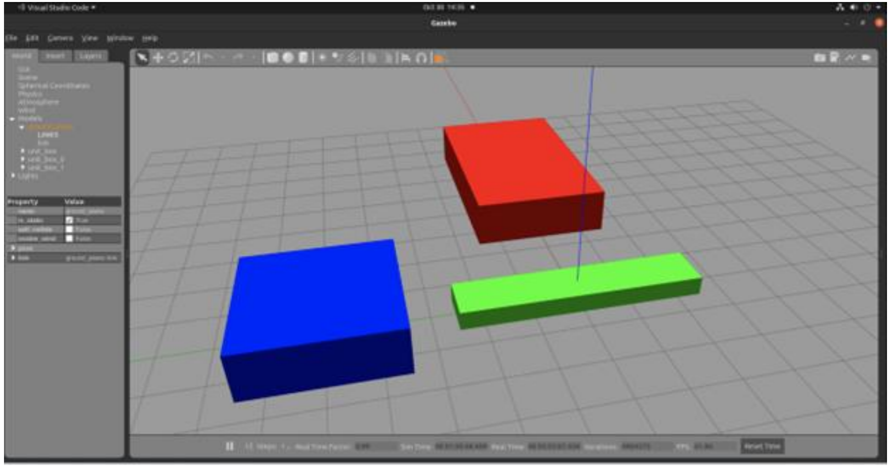
- Used objects from the **Gazebo simulation library**.
- **Drag-and-drop** placement of objects to create the initial environment.

Customization:

- Modified the setup to meet project requirements.
- Created the **first iteration** of the simulation environment.

Outcome:

- A functional environment for testing and validating the robot's **grasping** and **transportation** tasks.

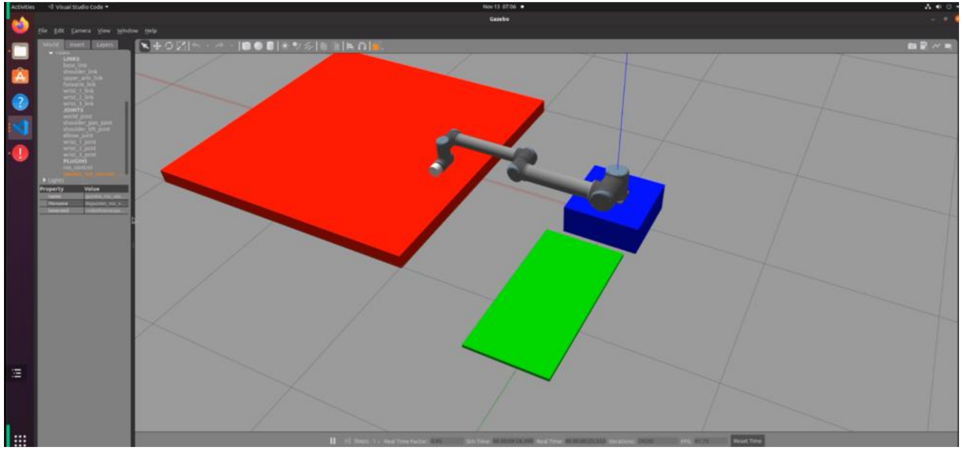


Continue - box design and better world

Customization:

- **improved initial testing world file**
- **add box model into gazebo world**

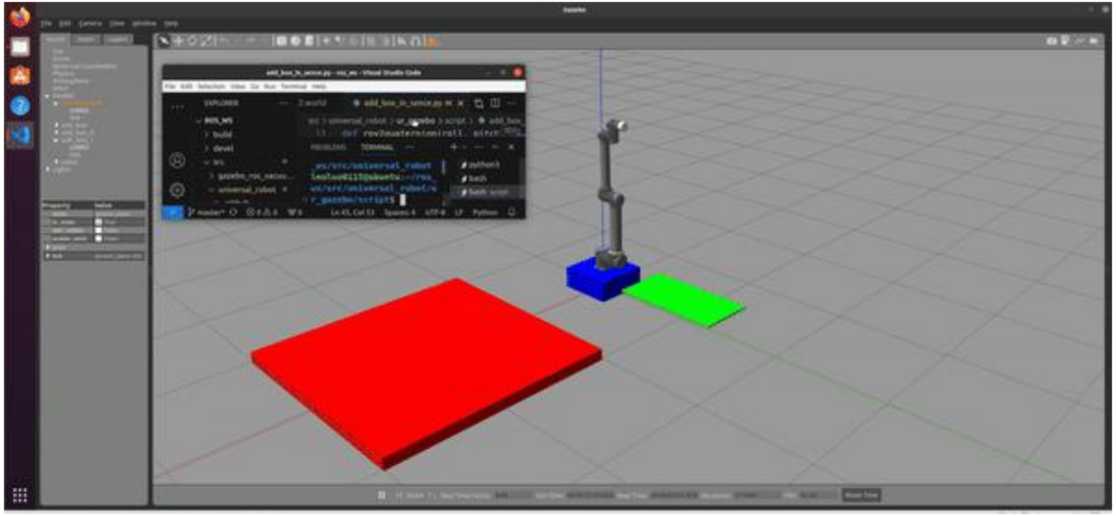
```
<!-- world define -->
<state world_name='default'>
  <sim_time>544 136000000</sim_time>
  <real_time>150 636984561</real_time>
  <wall_time>1730302602 110928810</wall_time>
  <iterations>136521</iterations>
  <model name='ground_plane'>
    <pose>0 0 0 0 -0 0</pose>
    <scale>1 1 1</scale>
    <link name='link'>
      <pose>0 0 0 0 -0 0</pose>
      <velocity>0 0 0 0 -0 0</velocity>
      <acceleration>0 0 0 0 -0 0</acceleration>
      <wrench>0 0 0 0 -0 0</wrench>
    </link>
  </model>
  <!-- red destination -->
  <model name='unit_box'>
    <pose>2 0 0.05 0 -0 0</pose>
    <scale>2 2 0.1</scale>
    <link name='link'>
      <pose>2 0 0.05 0 -0 0</pose>
      <velocity>0 0 0 0 -0 0</velocity>
      <acceleration>-0 -9.8 -0 2.23746 -0 0</acceleration>
      <wrench>-0 -9.8 -0 0 -0 0</wrench>
    </link>
  </model>
  <!-- blue base -->
  <model name='unit_box_0'>
    <pose>0 0 0.1 0 0 0</pose>
    <scale>0.5 0.5 0.2</scale>
    <link name='link'>
      <pose>0 0 0.1 0 0 0</pose>
```



Add box model into gazebo

Initial Setup:

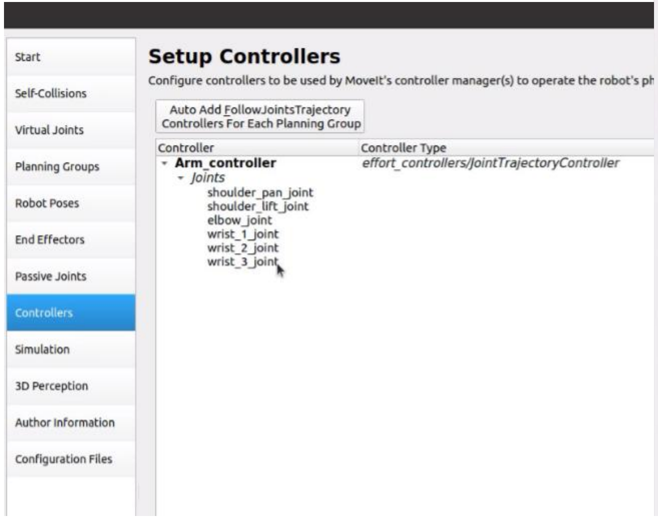
- Make your box model SDF file
- Use program to add box model into gazebo world

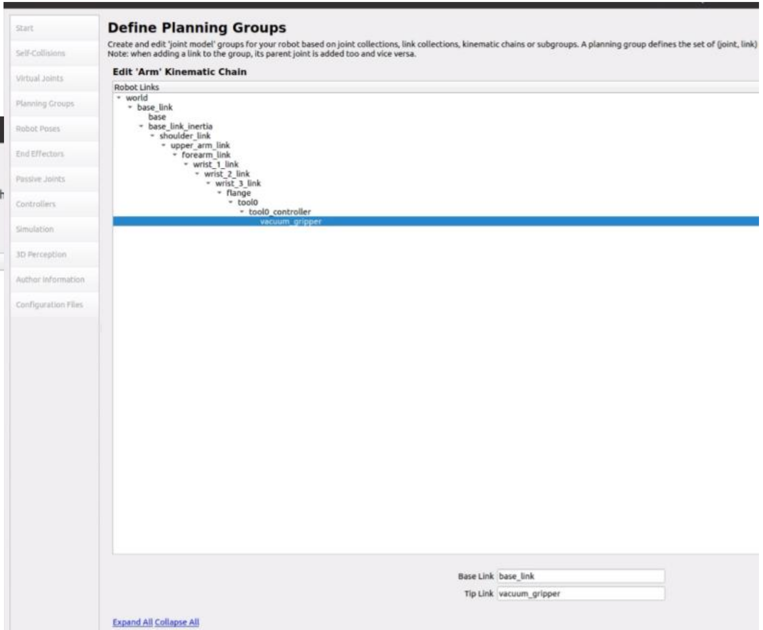


```python
# Function to spawn a model
def spawn_aruco_cube_hover(name='6'):
    model_name = "box" + name
    model_path = "/home/leoluo0115/ros_ws/src/universal_robot/ur_gazebo/sdf/box/model.sdf"
    initial_pose = Pose(position=Point(x=-0.0, y=1, z=0.6), orientation=rpy2quaternion(0, 0, 1.57))

    # Load the model from file
    with open(model_path, "r") as f:
        model_xml = f.read()

    # Call the SpawnModel service in Gazebo
    spawn_model = rospy.ServiceProxy('/gazebo/spawn_sdf_model', SpawnModel)
    resp_sdf = spawn_model(model_name, model_xml, "", initial_pose, "world")

    if resp_sdf.success:
        rospy.loginfo("Model '{}' spawned successfully.".format(model_name))
    else:
        rospy.logerr("Failed to spawn model '{}'.".format(model_name))


# Call the function to spawn the model
if __name__ == '__main__':
    try:
        # Initialize ROS node
        rospy.init_node('spawn_box', anonymous=True)
        spawn_aruco_cube_hover()
    except rospy.ROSInterruptException:
        pass
```

Config moveit

- **Setup base / tip link**
- **check number of joints in your group**
- **This will generate a whole new packages**





Code and File

```
universal_robot
  .github
  universal_robots
  ur_description
  ur_gazebo
    config
    launch
    script
    sdf / box
      model.config
      model.sdf
    tests
    urdf
      ur_macro.xacro
      ur.xacro
    worlds
      my_world_v1.world
      my_world_v2.world
      my_world.world
```

```
vacuum_config
  config
    cartesian_limits.yaml
    chomp_planning.yaml
    fake_controllers.yaml
    gazebo_controllers.yaml
    gazebo_ur10_robot.u...   M
    joint_limits.yaml
    kinematics.yaml
    ompl_planning.yaml
    ros_controllers.yaml
    sensors_3d.yaml
    simple_moveit_controllers....
    stomp_planning.yaml
    ur10_robot.srdf
  launch
    chomp_planning_pipeline.l...
    default_warehouse_db.lau...
    demo_gazebo.launch
    demo.launch
    fake_moveit_controller_ma...
    gazebo.launch
    joystick_control.launch
```

```python
def go_to_joint_state(self, joints):
    group = self.group
    joint_goal = group.get_current_joint_values()
    for i in range(6):
        print("joint "+str(i)+" angle is ", joint_goal[i])
    # joint_goal[0] = 1.57
    # joint_goal[1] = -1.57
    # joint_goal[2] = 1.57
    # joint_goal[3] = 1.57
    # joint_goal[4] = -1.57
    # joint_goal[5] = 0.0

    joint_goal = joints

    group.go(joint_goal, wait=True)
    group.stop()

    current_joints = self.group.get_current_joint_values()
    return all_close(joint_goal, current_joints, 0.01)
```

How to Control the Vacuum Gripper in ROS

Overview:

- ROS Services are defined by srv files, which contains a *request* message and a *response* message. These are identical to the messages used with ROS Topics (see rospy message overview).
- You call a service by creating a rospy.ServiceProxy with the name of the service you wish to call. You often will want to call rospy.wait_for_service() to block until a service is available.

Key Steps to Control the Vacuum Gripper:

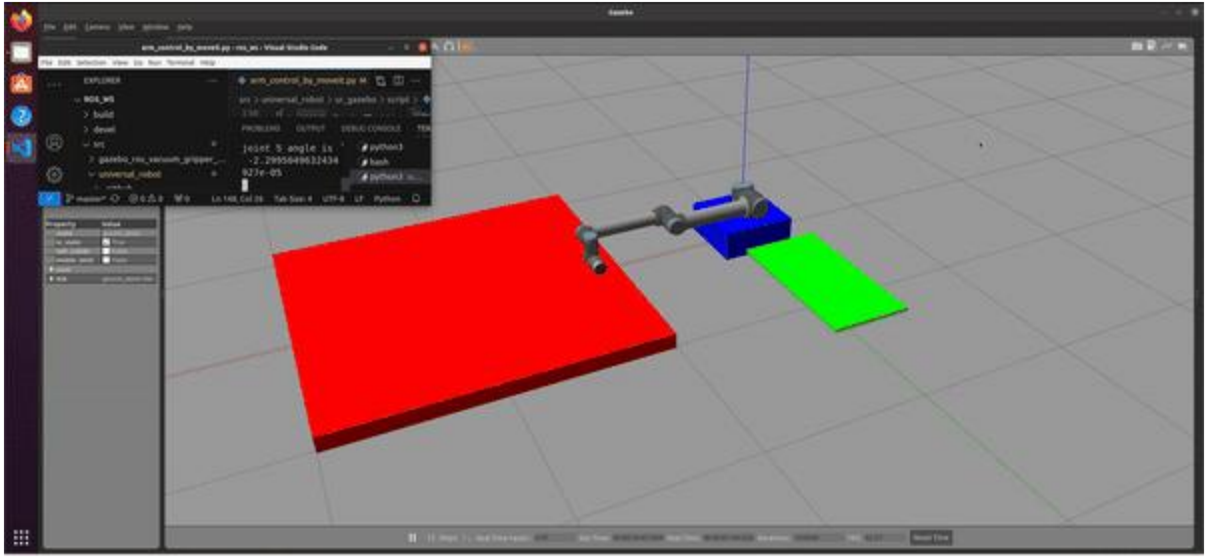
1. **Service Availability Check**:
    - Before using a service, ensure it is active and available to avoid errors.
    - This is similar to "subscribing" in the sense that you wait until the service is ready.
2. **Service Invocation**:
    - Once the service is available, you invoke it using p
        - **Turn On**: Activates the vacuum gripper.
        - **Turn Off**: Deactivates the vacuum gripper.

```
rospy.wait_for_service('/vacuum_gripper/on', 1)
rospy.wait_for_service('/vacuum_gripper/off', 1)
_on = rospy.ServiceProxy('/vacuum_gripper/on', Empty)
_off = rospy.ServiceProxy('/vacuum_gripper/off', Empty)
```

demo: current state

What we have **accomplished** so far includes completing the initial phase of **motion planning**, successfully **dropping the box** into the correct location, and **activating the vacuum gripper**. However, we are encountering issues with the plugin.

We believe the joint tag is currently set to **fixed**, which may not be the most suitable option. Using a **revolute** joint could be more appropriate. However, the revolute joint requires a **limit** tag, where the lower and upper bounds must be properly configured.



Error

```
ocess[robot_state_publisher-7]: started with pid [16263]
RROR] [1733089137.045141529]: Joint [gripper_joint] is of type REVOLUTE but it does not specify limits
RROR] [1733089137.046524721]: joint xml is not initialized correctly
INFO] [1733089137.383423481]: Stereo is NOT SUPPORTED
INFO] [1733089137.383639447]: OpenGL device: SVGA3D; build: RELEASE;  LLVM;
INFO] [1733089137.384105294]: OpenGl version: 4.1 (GLSL 4.1) limited to GLSL 1.4 on Mesa system.
obot_state_publisher-7] process has died [pid 16263, exit code 1, cmd /opt/ros/noetic/lib/robot_state_publisher/robot_state_publisher __name:=robot_st
e_publisher __log:=/home/leoluo0115/.ros/log/a9312730-b02c-11ef-8ef5-6b059b154def/robot_state_publisher-7.log].
g file: /home/leoluo0115/.ros/log/a9312730-b02c-11ef-8ef5-6b059b154def/robot_state_publisher-7*.log
obot_state_publisher-7] restarting process
ocess[robot_state_publisher-7]: started with pid [16317]
RROR] [1733089137.454895922]: Joint [gripper_joint] is of type REVOLUTE but it does not specify limits
RROR] [1733089137.456840926]: joint xml is not initialized correctly
INFO] [1733089137.663126078]: waitForService: Service [/gazebo/set_physics_properties] is now available.
```

Next Steps:

- **Debug the vacuum gripper to ensure it successfully grips the object.**
- **Verify the properties of the revolute joint and set appropriate limit boundaries.**
- **Complete the final step by moving to the target position and placing the object in the designated drop box.**

Lessons and Learnings

- **Ensure your virtual machine has sufficient RAM to maintain a real-time factor as close to 1 as possible.**
- **position control is mush easier than effort control since we do not need to set force or torque for each joint**
- **Ensure the design is simple and easy to test, allowing for efficient focus on motion planning and testing**

Q & A?