# Universal Robots UR5 Smart Outliner

A ROS2 Simulation inside a Containerized environment

**Student:** Tommaso Cosimi
**ID:** 191739

Università degli Studi di Modena e Reggio Emilia
Dipartimento di Ingegneria "Enzo Ferrari"

# Table of Contents


Project Overview

- Why this Toolset

Docker Setup

Robot Setup

Path Predictor Setup

Usage

# Project Overview

# Project Overview

The Project explores the **Motion Planning** capabilities of a simulated **Universal Robots UR5** manipulator aided by **Computer Vision** techniques.

The **Robot simulation** runs under **ROS2 Humble** accompanied by **MoveIt2** and exploiting **Gazebo 11** for a physically-accurate simulation environment, while the **Computer Vision applications** make use of both **ROS Nodes** and standalone **Python scripts**.

All of this is running inside **Docker Containers**.

# Why ROS2?

The **discontinuation** and **deprecation** of the **classic ROS distributions** is the main driving factor behind the move to ROS2.

Other advantages of ROS2 compared to ROS:

- **DDS** Approach, no more Master Node
  - Decentralization
  - Lower communication latency (Real Time applications)
  - Support for QoS
  - Easier Scalability
- Standardization of a **Security Model**
- More **modern building** and **packaging** (`colcon`)
- Initial **support** for different Operating Systems

# Why MoveIt2?

MoveIt2 is a mostly complete porting of the MoveIt framework to ROS2.
Being compatible with ROS2 it brings in all of its advantages.
APIs:

- C++ $\rightarrow$ Complete w.r.t. MoveIt for ROS
- Python $\rightarrow$ Almost complete w.r.t. MoveIt for ROS

# Why Docker?

Containerizing applications means:

- **Isolation**
- **Security**
- **Reproducibility**
- **Portability**

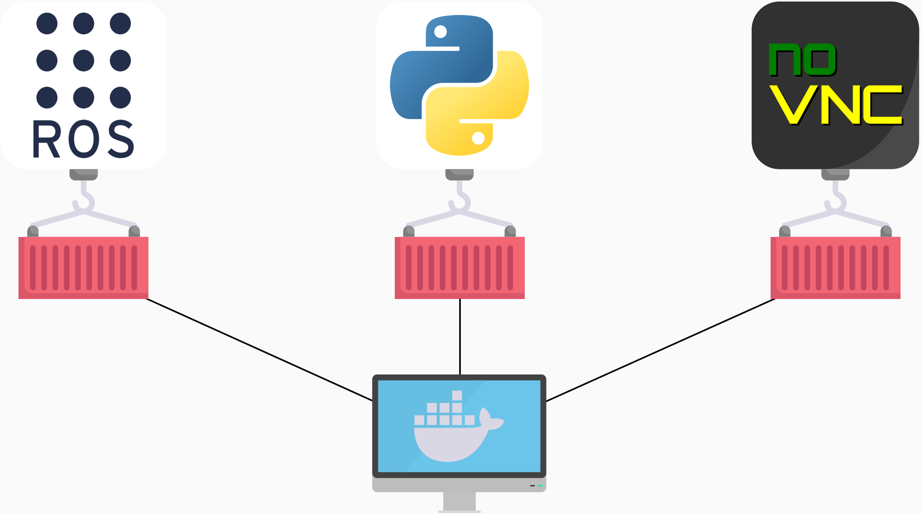
This project provides a way to develop ROS and Python applications inside specific **Host-OS-agnostic Containers** which ensure a **standard development environment**, all while offering **less overhead** than using a full Virtual Machine.

# Why DevContainers?

Using **DevContainers** is helpful while developing containerized applications because it **allows** for **development** work to happen **directly inside** of **the target container** in a declared environment (even for the IDE).

# Docker Setup

# Docker Setup - Architecture



# Docker Setup - Docker Compose

The Compose file sports three services, each of which spins one of the three containers.

```yaml
services:
  ros-humble-sim:
    # ROS Humble Container and its configurations
    [...]

  path-predictor:
    # Python Container and its configurations
    [...]

  novnc_server:
    # noVNC Server Container and its configurations
    [...]
```

# Docker Setup - Docker Compose - ROS Service

```yaml
ros-humble-sim:
  container_name: ros-humble-sim
  privileged: true
  build:
    context: ./.devcontainer/ros-humble-sim/
    dockerfile: Dockerfile
  volumes:
    # ROS Workspace
    - ./ros-humble-sim_workspace:/ros_workspace
    # Common Workspace
    - ./common_workspace:/common
    # ROS Utils
    - ./utils/ros-humble-sim:/utils
    # X11 Socket
    - ./utils/novnc_server/desktop-x11:/tmp/.X11-unix
    - ./utils/novnc_server/desktop-x11/.Xauthority:/tmp/.Xauthority:ro
  environment:
    # X11 Setup
    - DISPLAY=:1
    - XAUTHORITY=/tmp/.Xauthority
  command: sleep infinity
  depends_on:
    - novnc_server
```

# Docker Setup - Docker Compose - Python Path Predictor Service

```yaml
path-predictor:
  container_name: path-predictor
  build:
    context: ./.devcontainer/path-predictor/
    dockerfile: Dockerfile
  volumes:
    # Python Workspace
    - ./path-predictor_workspace:/py_workspace
    # Common Workspace
    - ./common_workspace:/common
    # Python Utils
    - ./utils/path-predictor:/utils
    # X11 Socket
    - ./utils/novnc_server/desktop-x11:/tmp/.X11-unix
    - ./utils/novnc_server/desktop-x11/.Xauthority:/tmp/.Xauthority:ro
  environment:
    #X11 Setup
    - DISPLAY=:1
    - XAUTHORITY=/tmp/.Xauthority
  command: sleep infinity
  depends_on:
    - novnc_server
```

# Docker Setup - Docker Compose - noVNC Server Service

```yaml
novnc_server:
  container_name: novnc_server
  ports:
    # Allow connections to the noVNC Host from the Docker host only
    - 127.0.0.1:6080:6080
    # Allow connections to the noVNC Host from every interface of the Docker Host
    # - 6080:6080
  build:
    context: ./utils/novnc_server/
    dockerfile: Dockerfile
  volumes:
    # noVNC Server utilities and scripts
    - ./utils/novnc_server/scripts:/scripts
    # X11 Socket
    - ./utils/novnc_server/desktop-x11:/tmp/.X11-unix
    - ./utils/novnc_server/desktop-x11/.Xauthority:/tmp/.Xauthority
  environment:
    # X11 Setup
    - DISPLAY=:1
    - XAUTHORITY=/tmp/.Xauthority
  command: [ "/scripts/startup.sh" ]
```

# Docker Setup - DevContainers - ROS Humble

```json
{
    "name": "ROS Humble",
    "dockerComposeFile": [
        "../../docker-compose.yml"
    ],
    "service": "ros-humble-sim",
    "shutdownAction": "none",
    "workspaceFolder": "/ros_workspace",
    "postStartCommand": "bash /utils/install_deps_and_build.sh",
    "customizations": {
        "vscode": {
            "extensions": [
                "ms-python.vscode-debugpy",
                "ms-python.vscode-pylance",
                "ms-python.vscode-python",
                "ms-python.vscode-python-envs",
                "ms-vscode.cpptools-extension-pack",
                "ranch-hand-robotics.rde-pack",
                "redhat.vscode-xml",
                "redhat.vscode-yaml"
            ]
        }
  }
}
```

# Docker Setup - DevContainers - Python Path Predictor

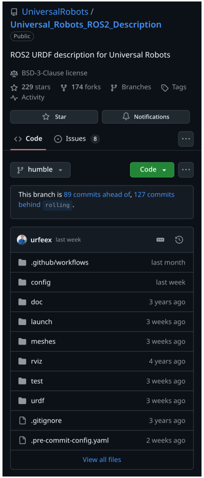
```json
{
    "name": "Path Predictor",
    "dockerComposeFile": [
        "../../docker-compose.yml"
    ],
    "service": "path-predictor",
    "shutdownAction": "none",
    "workspaceFolder": "/py_workspace",
    "customizations": {
        "vscode": {
            "extensions": [
                "ms-python.python",
                "ms-python.debugpy",
                "ms-python.vscode-pylance",
                "ms-python.vscode-python-envs"
            ]
        }
  }
}
```

# Robot Setup

# Manipulator Description

The Manipulator Description comes from the “Humble” branch of the official GitHub repository.

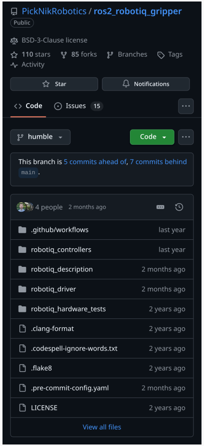
Several modifications were made to the original URDF to allow for the Gazebo Simulation and inclusion of a Depth Camera and Gripper.



# Gripper Description

The Robotiq 2f85 Gripper Description comes from the “Humble” branch of the PickNikRobotics’ GitHub repository.

Several modifications were made to the original URDF to allow for the Gazebo Simulation, which was initially not supported.



# Robot Controllers

The controllers used were the ones from the description repositories, grouped in a single file for ease of use.

This means that the MoveIt2 dummy controllers were to be replaced in the final MoveIt Configuration Package.

Especially the Joint Trajectory Controller and the Gripper Action Controller were useful for the project purposes.

# Depth Camera Description

The Depth camera is entirely simulated:

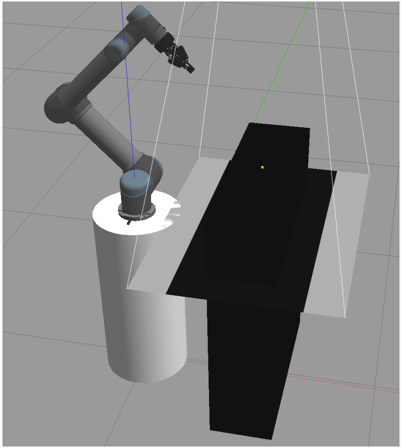
- $1024px \times 768px$ Resolution
- $2.24\mu m \times 2.24\mu m$ pixels
- Minimum perceived depth of $0.10m$
- Maximum perceived depth of $2.25m$

It is positioned $2m$ above and $50cm$ in front of the Base Link of the Robot, while pointing down.

# Gazebo World

The world consists in three objects:

- Robot Base
- Target Base
- Target



# MoveIt Setup - Move Groups and Controllers

Two move groups:

1. "`ur_manipulator`" → The UR5 Manipulator
2. "`ur_gripper`" → The Robotiq 2f85 Gripper

Two ROS2 controllers:

1. "`joint_trajectory_controller`" → `JointTrajectoryController`
2. "`gripper_position_controller`" → `GripperActionController`

Two MoveIt2 controllers:

1. "`joint_trajectory_controller`" → `FollowJointTrajectory`
2. "`gripper_position_controller`" → `GripperCommand`

# MoveIt Setup - Motion Planners

Adapted using the correct move groups from the MoveIt2 tutorials repository:

- "chomp_planning.yaml"
- "lerp_planning.yaml"
- "ompl_planning.yaml"
- "pilz_industrial_motion_planner_planning.yaml"
- "trajopt_planning.yaml"

# Running the Inverse Kinematics

The Inverse Kinematics is ran using the MoveIt2 APIs in a C++ ROS2 node.

```cpp
// Create MoveGroup Interface and set it up
using moveit::planning_interface::MoveGroupInterface;
MoveGroupInterface move_group(node, "ur_manipulator");
move_group.setPoseReferenceFrame("world");
move_group.setPlanningTime(10.0);
move_group.setMaxVelocityScalingFactor(0.2);
move_group.setMaxAccelerationScalingFactor(0.2);
```

# Running the Inverse Kinematics

The world objects geometries are added to the planning scene for the Motion Planner to avoid them.

```
// Description of the Robot Base
collision_objects[0].id = "robot_base";
collision_objects[0].header.frame_id = "world";
collision_objects[0].primitives.resize(1);
collision_objects[0].primitives[0].type = shape_msgs::msg::SolidPrimitive::CYLINDER;
collision_objects[0].primitives[0].dimensions = {0.750, 0.175};
collision_objects[0].primitive_poses.resize(1);
collision_objects[0].primitive_poses[0].position.x = 0.000;
collision_objects[0].primitive_poses[0].position.y = 0.000;
collision_objects[0].primitive_poses[0].position.z = 0.375;
collision_objects[0].operation = moveit_msgs::msg::CollisionObject::ADD;
// Description of the Target Base
collision_objects[1].id = "target_base";
collision_objects[1].header.frame_id = "world";
collision_objects[1].primitives.resize(1);
collision_objects[1].primitives[0].type = shape_msgs::msg::SolidPrimitive::BOX;
collision_objects[1].primitives[0].dimensions = {0.25, 0.75, 1.00};
collision_objects[1].primitive_poses.resize(1);
collision_objects[1].primitive_poses[0].position.x = 0.50;
collision_objects[1].primitive_poses[0].position.y = 0.00;
collision_objects[1].primitive_poses[0].position.z = 0.50;
collision_objects[1].operation = moveit_msgs::msg::CollisionObject::ADD;
// Add the objects
planning_scene_interface.applyCollisionObjects(collision_objects);
```

# Running the Inverse Kinematics

The object contours coordinates are loaded into a "pts" vector, and the waypoints are computed by combining such coordinates and a predefined target orientation.

```
// Set the tool0 Reference Frame target orientation to rpy = [0, -pi, -pi]
tf2::Quaternion q;
q.setRPY(0.0, -M_PI, -M_PI);
geometry_msgs::msg::Quaternion orientation = tf2::toMsg(q);

// Build the waypoints from the pts vector coming from the file
// NOTE: the target z position will be offset by the length of the end
↪  effector in the closed postion

const double z_offset = 0.163;
std::vector<geometry_msgs::msg::Pose> waypoints;
waypoints.reserve(pts.size());
for (const auto &pp : pts) {
    geometry_msgs::msg::Pose p;
    p.position.x = pp[0];
    p.position.y = pp[1];
    p.position.z = pp[2] + z_offset;
    p.orientation = orientation;
    waypoints.push_back(p);
}
```

# Running the Inverse Kinematics

Finally, a cartesian path is computed and the trajectory with its timing law is executed.

```
// Compute Cartesian path
moveit_msgs::msg::RobotTrajectory trajectory_msg;
const double eef_step = 0.001;
const double jump_threshold = 0.0;
double fraction = move_group.computeCartesianPath(waypoints, eef_step,
↪  jump_threshold, trajectory_msg);
RCLCPP_INFO(logger, "Computed Cartesian path fraction: %.3f", fraction);
if (fraction < 0.99) {
    RCLCPP_ERROR(logger, "Cartesian path incomplete (fraction < 0.99).
    ↪  Aborting.");
    rclcpp::shutdown();
    spinner_thread.join();
    return 1;
}

// Execute trajectory
moveit::planning_interface::MoveGroupInterface::Plan plan;
plan.trajectory_ = trajectory_msg;
```

# Path Predictor Setup

# ROS Package

Used to gather the image and depth data from the Depth Camera Node, which publishes:

- Color image on "`/depth_camera/image_raw`"
- Greyscale depth image on "`/depth_camera/depth/image_raw`"
- PointCloud on "`/depth_camera/points`"

And store such data in files in the Common Workspace.

# Python Application - Bootstrapper

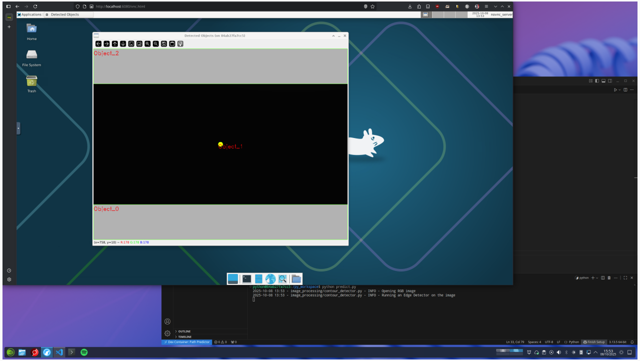
The bootstrapper launches three scripts:

1. Contour detector
2. Pixel to World coordinates Converter
3. Duplicate Remover

```python
if __name__ == "__main__":
    contour_detector.detect_contours(
        images_dir="/common/image_processing",
        trajectory_computation_dir="/common/trajectory_planning",
        rgb_image_filename="camera_image.png",
        obj_contour_pixels_filename="relevant_contours_pixels.json"
    )
    pixel_to_world.pixel_to_coordinates(
        images_dir="/common/image_processing",
        trajectory_computation_dir="/common/trajectory_planning",
        depth_data_filename="depth_data.npy",
        obj_contour_pixels_filename="relevant_contours_pixels.json",
        obj_contour_coordinates_filename="relevant_contours_coordinates.json",
        camera_intrinsics=camera_intrinsics,
        camera_extrinsics=camera_extrinsics
    )
    duplicate_remover.remove_duplicates(
        trajectory_computation_dir="/common/trajectory_planning",
        obj_contour_coordinates_filename="relevant_contours_coordinates.json",
        obj_contour_coordinates_cleared_filename="relevant_contours_cleared_coordinates.json"
    )
```

# Python Application - Contour Detector

The detector uses `OpenCV` to find object contours by thresholding the image's pixels RGB values and finding closed paths.
After some candidates are found, the program shows a window with the image and the different objects with their contours highlighted.
The user is finally asked which one is the object of interest before dumping its contour's pixels into the output file.




```
python@84ab27fa7cc5:/py_workspace$ python predict.py
2025-10-08 13:53 - image_processing/contour_detector.py - INFO - Opening RGB image
2025-10-08 13:53 - image_processing/contour_detector.py - INFO - Running an Edge Detector on the image
Select the object to choose by its index: 1
```

# Python Application - Pixel to World coordinates Converter

Given the simulated camera intrinsic and extrinsic parameters it computes the correspondance between the pixel coordinates and the simulated world 3D coordinates, using both the contour's pixels coordinates and the depth data coming from the PointCloud published by the depth camera itself.

```python
k = compute_intrinsic_matrix(
    camera_intrinsics,
    logger
)

rotation_matrix, translation_vector = compute_extrinsic_matrix(
    camera_extrinsics,
    logger
)

object_contours_coordinates = []
k_inv = np.linalg.inv(k)
for point in object_contours:
    u, v = point
    w = depth_data[v, u]
    coordinates_wrt_camera = w * (k_inv @ np.array([u, v, 1.0]))
    coordinates_wrt_worldframe = rotation_matrix @ coordinates_wrt_camera +
    ↪   translation_vector
    ## Round the results to have millimetric precision
    np.round(
        a=coordinates_wrt_worldframe,
        decimals=3,
        out=coordinates_wrt_worldframe
    )
    object_contours_coordinates.append(coordinates_wrt_worldframe.tolist())
```

# Python Application - Duplicate Remover

Since there's the possibility that different pixels map to the same coordinates after the rounding to millimetric precision, the consecutive duplicates are dropped from the final object contour coordinates file.

```python
# Remove Duplicates
cleared_points = []
for idx, point in enumerate(object_contours):
    if idx > 0:
        if point == object_contours[idx-1]:
            logger.info(f"Duplicate point found")
        else:
            cleared_points.append(point)
    else:
        logger.info(f"Adding first point")
        cleared_points.append(point)
```

# Usage

# Usage Pipeline

1. **Spawn the Robot with the depth camera and take pictures**

   ```
   ros@ros_devcontainer:~$ ros2 launch ur_gripper_sim get_data_w{i}.launch.py
   ```

2. **Predict the contours of the object**

   ```
   py@python_devcontainer:~$ python ./predict.py
   ```

3. **Spawn the Robot with the MoveIt2 Controllers and run the Inverse Kinematics**

   ```
   # Terminal Window 1
   ros@ros_devcontainer:~$ ros2 launch ur_gripper_sim sim_w{i}.launch.py
   # Terminal Window 2
   ## Run the node once the simulation environment is ready
   ros@ros_devcontainer:~$ ros2 run ur_gripper_sim run_ik
   ```

# Demo

**Thank you.**

Open Robotics.
**ROS2 Humble Art.**
Image available at: https://github.com/openrobotics/artwork/blob/master/distributions/humble/HumbleHawksbillTransparent.png.

PickNik Robotics.
**MoveIt2 Humble Art.**
Image available at:
https://moveit.picknik.ai/humble/_static/humble-small.png.

Docker Inc.
**Docker Logo.**
Image available at:
https://www.docker.com/company/newsroom/media-resources/.

Open Robotics.
**ROS Logo.**
Image available at:
https://commons.wikimedia.org/wiki/File:Ros_logo.svg.

Python Software Foundation.
**Python Logo.**
Image available at: https://www.python.org/static/community_logos/python-logo-inkscape.svg.

noVNC Authors and Contributors.
**noVNC Logo.**
Image available at: https://github.com/novnc/noVNC/blob/master/app/images/icons/novnc-icon.svg.

kerismaker.
**Container.**
Image available at:
https://www.reshot.com/free-svg-icons/item/container-SH5MJB9DK7/.

IconBunny.
**Computer.**
Image available at:
https://www.reshot.com/free-svg-icons/item/computer-EK8XD9YUQS/.