02 July 2025

Manual

for

Automated Cap Detection and Alignment on Drum Surfaces
with UR5 using CoppeliaSim and ROS 2

Introduction

This manual presents the process to get started with the simulation and control of a UR5 robotic arm equipped with a high-resolution (420x1) perspective type visionSensor modified into a laser scanner for automated inspection of metallic drums containing used heavy water. Using CoppeliaSim EDU as the simulation environment and ROS2 as the middleware, the system performs precise scanning of drum surfaces to detect and localize the cap. Upon accurate identification of the cap's center, the UR5 is autonomously guided to lower its end-effector to this location while adjusting for any perturbations on the surface, demonstrating a safe and efficient approach for handling hazardous materials. The project showcases seamless integration of advanced robotics, sensor processing, and simulation tools to address real-world industrial challenges in nuclear waste management.

List of Main Components

- **CoppeliaSim:** A powerful robot simulation platform used for modeling, simulating, and visualizing robotic systems in a 3D environment. It supports integration with various programming languages and features a hierarchical structure for managing robot components.
- **ROS2:** Middleware for modular communication and control, enabling integration between the simulation environment and external control or perception nodes.
- **Python Scripts:** Custom code for implementing forward kinematics, interfacing with CoppeliaSim, and automating simulation tasks. Python is chosen for its ease of use and rich ecosystem of scientific libraries.
- **STL Drum and Cap Model:** The robot's mechanical design, created using CAD software and exported as a STL file for import into CoppeliaSim.
- **Simulation Scene Files:** CoppeliaSim scene files (.ttt) that define the virtual environment, robot, and any sensors or objects involved in the simulation.

System and Softwares used:

OS:	Ubuntu 24.04.2 LTS
Hardware Model:	Intel Corporation DB75EN
Memory:	8.0 GiB
Processor:	Intel® Core™ i7-3770 × 8
Graphics:	NVC3
Disk Capacity:	500.1 GB
Simulation:	CoppeliaSim EDU
Middleware:	ROS2 – Rolling Ridley

Installations

Installing CoppeliaSim:

- Download the latest CoppeliaSim Edu version from the official website.
- Extract the downloaded archive to your preferred directory.
- Add CoppeliaSim to your PATH or create an alias for easy launching
- After getting inside the source directory, test the installation by launching CoppeliaSim using the command ./coppeliaSim or ./coppeliaSim.sh

Set the Python Environment

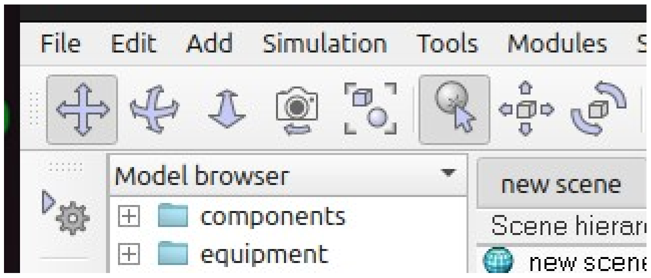
- Install Python 3 if not already present.
- Install required Python packages using pip install requirements.txt once you are inside the project directory.

Installing and Configuring ROS2

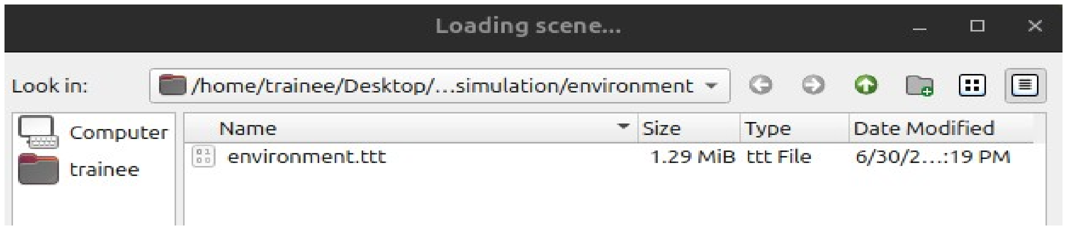
- Install the recommended ROS2 distribution for your OS (I used Rolling Ridley).
- Set up your ROS2 workspace and source the environment.
- Install any required ROS2 packages and dependencies.

Running the Simulation

1. Open a new terminal and go to the directory where CoppeliaSim is downloaded.
   - Once inside the directory, run the command ./coppeliaSim.
   - After the software opens, go to the 'File' option in the menu and then select 'Open Scene' or use 'New Scene' to create your own environment.



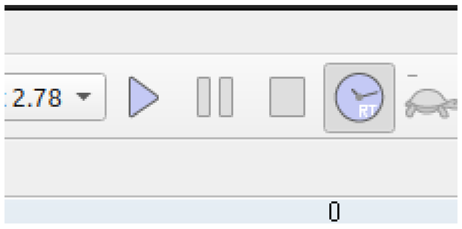
   - After clicking on 'Open Scene', from the project directory choose the file 'environment.ttt' which would be inside the directory environment.



/home/trainee/Desktop/...simulation/environment

Name	Size	Type	Date Modified
environment.ttt	1.29 MiB	ttt File	6/30/2...:19 PM

   - Once chosen the scene should be loaded in the simulator with the drum, cap, UR5 and the visionSensor already in place.
   - Click on the Play button to start the simulation and use the square button to stop and reset the simulation.



2. Now, open a new Terminal parallelly and go to the project directory. Once inside, run the command python vision_depth_publisher.py. This will start a ZMQ Remote API Server on port 24000.
3. Now keep the above terminal running and open a new Terminal and paste the command - python sequence.py. This will begin the first sweep which will approximate the center of the cap and will adjust the visionSensor(connected to the target dummy) to reach it. After that, it will start the second sweep to get the exact center of the cap and orient itself perpendicular to the direction of the ridge.

Modification and Tinkering

Fine-tuning variables are located at the beginning of each script:

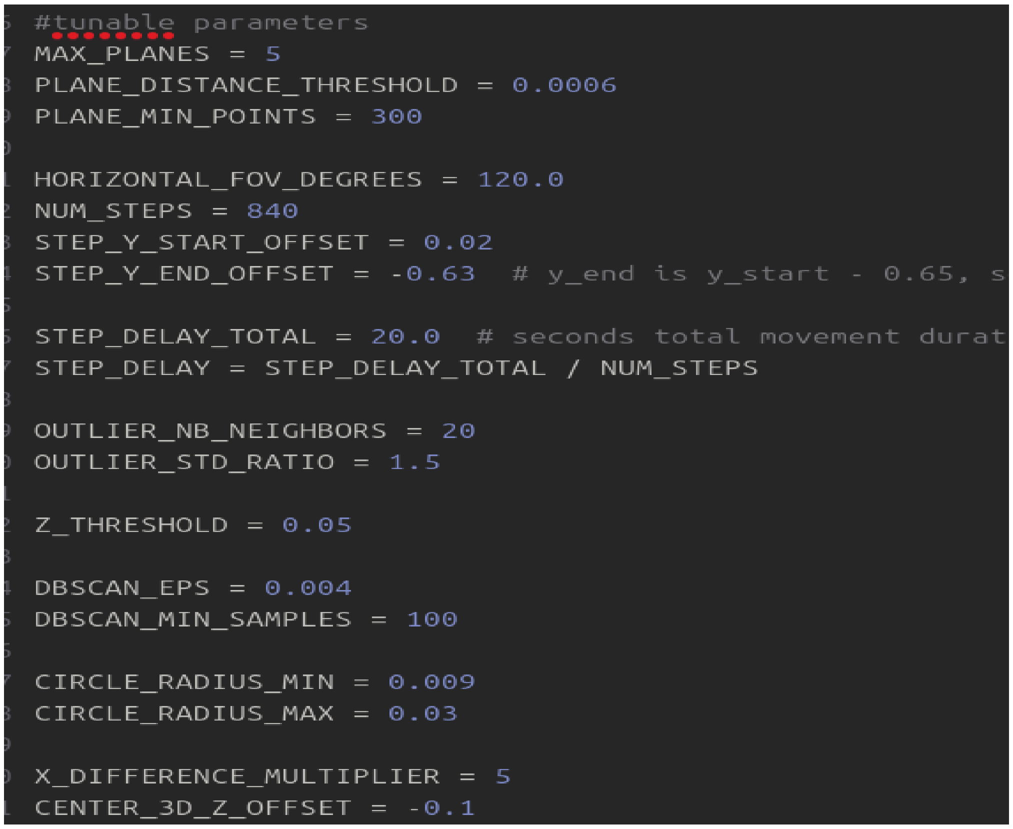
- first_sweep.py
- second_sweep.py
- adjust_dummy_1.py
- adjust_dummy_2.py
- drum_manipulator.py

To optimize performance:

1. **Edit these values** directly in the scripts to match your simulation requirements.
2. **Go through the official documentation** for function-specific parameter definitions before modification.
3. **Refer to the report** for algorithm use and its behavior.

Best practices:

- Adjust one parameter at a time and validate changes.
- Leverage simulation visualization to observe parameter effects. For eg.



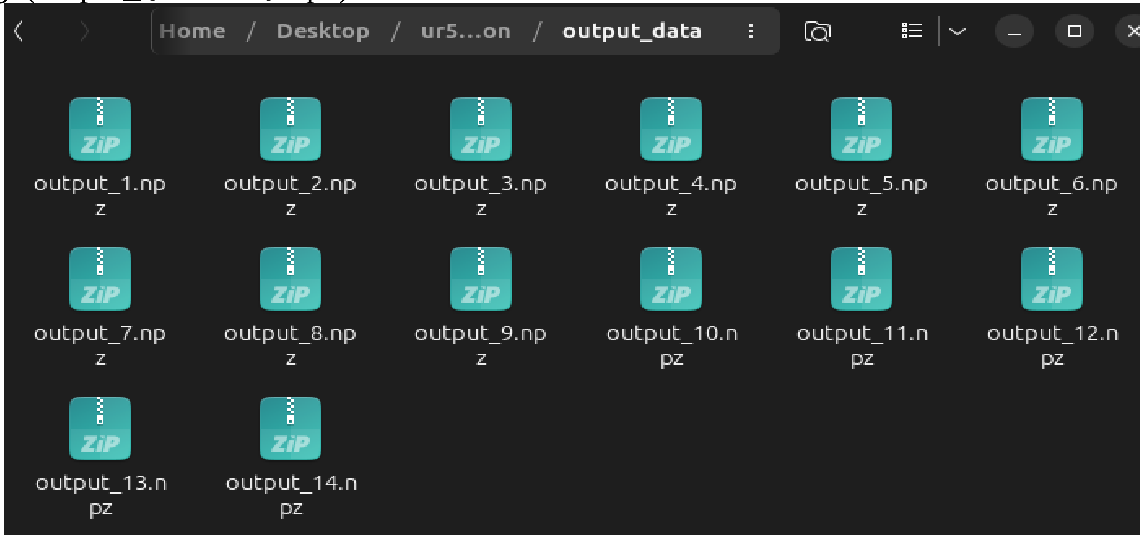
```
#tunable parameters
MAX_PLANES = 5
PLANE_DISTANCE_THRESHOLD = 0.0006
PLANE_MIN_POINTS = 300

HORIZONTAL_FOV_DEGREES = 120.0
NUM_STEPS = 840
STEP_Y_START_OFFSET = 0.02
STEP_Y_END_OFFSET = -0.63  # y_end is y_start - 0.65, s

STEP_DELAY_TOTAL = 20.0  # seconds total movement durat
STEP_DELAY = STEP_DELAY_TOTAL / NUM_STEPS

OUTLIER_NB_NEIGHBORS = 20
OUTLIER_STD_RATIO = 1.5

Z_THRESHOLD = 0.05

DBSCAN_EPS = 0.004
DBSCAN_MIN_SAMPLES = 100

CIRCLE_RADIUS_MIN = 0.009
CIRCLE_RADIUS_MAX = 0.03

X_DIFFERENCE_MULTIPLIER = 5
CENTER_3D_Z_OFFSET = -0.1
```

Note

Upon completing a successful sweep and cap identification, the system generates an output package with individual .npy files containing:

1. **Core Data**
   - Full 3D point cloud of the scanned cap surface (ZXY coordinates + intensity values)
   - Detected cap position (X, Y, Z coordinates)
   - Cap orientation (quaternion/Euler angles)
2. **Visualization Assets**
   - Circle fitting overlay on cap edge
   - Ridge line fitting visualization
   - Coordinate axes reference
   - Color-mapped point cloud highlighting curvature features

To visualize it later again, open main.py and edit the path of the file to the required output package(output_{number}.npz)



after that, save the contents of main.py and run the command python main.py to visualize it.

For any doubts regarding this project, feel free to email me at ashwinmenon502@gmail.com