COMP0250 Coursework 2: Tic-Tac-Toe Shape Detection, Classification, and Pick & Place Manipulation

This repository contains the implementation of the three advanced tasks for Coursework 2 using ROS and C++ on the Franka Emika Panda robot arm. Building on our previous work from CW1, this solution integrates advanced perception, planning, and grasping strategies to perform:

- **Task 1:** Precise pick and place operations with shape-specific grasp planning.
- **Task 2:** Robust classification of shapes on a variable-height table.
- **Task 3:** Detection, classification, and manipulation of shapes within a clutter environment.

The solution uses point cloud processing with PCL, computer vision techniques with OpenCV, and robust motion planning via MoveIt. Detailed visualisations are provided, including a custom RViz configuration that shows the perception pipeline's visualisation. It adds a marker topic and disables robot models, you can find the config file inside the submission package.

Authors

- Ziya Ruso
- Lorenzo Uttini
- Amlan Sahoo

Each author contributed equally (33.33%) to all tasks and worked around 60 hours each, with extensive work on integrating perception, motion planning, and grasping into a robust execution pipeline.

Building the Project

1. Clone the entire repository (COMP0250_S25_LABS) into your workspace and replace the `cw1_team_x` package with our `cw1_team_7` package.
2. **Install dependencies** Run the following command to install ROS and system dependencies:

```
rosdep install --from-paths src --ignore-src -r -y
```

3. **Build the package** Make sure you are inside the workspace and run:

```
cd ~/comp0250_s25_labs
catkin build cw2_team_7
```

Alternatively, you can also build the entire workspace with `catkin build`.

Note: You can ignore the `jump_threshold` parameter deprecation warning and `C++17 Structured Bindings` warning during the build (see Troubleshooting section below)

4. **Source your terminal** After building, source your terminal to use the newly built packages:

```
source devel/setup.bash
```

Running the Tasks

1. Launch the **simulation environment** in one sourced terminal:

```
roslaunch cw2_team_7 run_solution.launch
```

2. Call the desired task service from another sourced terminal:

```
# Task 1
rosservice call /task 1

# Task 2
rosservice call /task 2

# Task 3
rosservice call /task 3
```

Each task is triggered via the `/task` service followed by its corresponding task number. Make sure both terminals are sourced with:

```
source devel/setup.bash
```

Dependencies

- **ROS Noetic** — Main middleware for robot control.
- **MoveIt** — Motion planning framework.
- **PCL (Point Cloud Library)** — For point cloud processing.
- **OpenCV** — For computer vision and image processing tasks.
- **tf2** — For handling coordinate transformations.
- **cw2_world_spawner** — Provides custom service messages and simulation environment support.

Code Structure

Header Files

- `include/cw2_class.h`: Header file containing class definitions and method declarations used for the tasks
- `include/cw2_grasp.h`: Header file for grasp pose computation and gripper control

- `include/cw2_movement.h`: Header file for motion planning
- `include/cw2_perception.h`: Header file for perception

Source Files

- `src/cw2_class.cpp`: Main integration of perception, motion, and grasping
- `src/cw2_grasp.cpp`: Implementation of grasp pose computation and gripper control functions
- `src/cw2_movement.cpp`: Implementation of motion planning and manipulation strategies
- `src/cw2_perception.cpp`: Implementation of point cloud processing, cluster extraction, and perception strategies
- `src/cw2_node.cpp`: Entry point for the ROS node that initializes the CW2 class and handles service callbacks

Custom Rviz Config

- `cw2_team_7_config.rviz`: Our custom rviz config file for better perception visualization

Troubleshooting & Debugging

Restart Gazebo Setup

We have observed from 100+ test runs that sometimes if there's a slight failure in terms of movement planning or anything else, calling the tasks successively can result in multiple failures. Although this doesn't happen in most cases, if you start getting a string of failures one after another, please restart Gazebo completely and/or rebuild our package. This helps clear any residual states or errors, ensuring the system resets to a clean baseline. In our testing, the tasks execute precisely in the vast majority of runs.

Image Debugging

Debug images (binary projections, orientation images) are stored in the `/src/images` folder. These images are for development purposes only and do not impact runtime functionality.

Additional RViz Setup

A sample RViz config file is provided with already adjusted topic subscriptions and disable unnecessary visualisation topics. Refer to the config file in the repository for optimal settings.

C++17 Structured Bindings Warning

Warning: Structured bindings are only available with -std=c++17 or -std=gnu++17. The code still compiles and works correctly because ROS Noetic's default compiler supports C++17 features even if the project is set to C++11/14.

Deprecated Parameter for MoveIt's Cartesian Planner

The jump_threshold parameter is deprecated but the code compiles and operates correctly. The deprecation is due to improved, automatic methods for handling point cloud discontinuities that no longer require manual threshold tuning.

Task 1: Pick and Place Shape Manipulation

Overview

Task 1 focuses on basic pick and place operations with precise object manipulation. The robot receives coordinates for a shape (cross or nought) and a goal location, and must successfully grasp the object and place it in the designated target position while maintaining stable orientation throughout the movement.

Implementation Approach

1. Observation and Object Detection

The system begins with a structured perception process:

- **Strategic Positioning & Stabilise**: The camera moves to an optimal viewing angle above the target object and pauses for stabilization, ensuring clear point cloud data.
- **Dynamic Cluster Extraction**: Filters the point cloud by height, XY ranges and performs Euclidean clustering
- **Shape Identification**: Uses the specified shape type from the service request for customized grasping

2. Shape Orientation Detection

Accurate orientation detection is crucial for proper grasping:

- **Binary Image Projection**: Converts 3D point cloud to 2D binary projection
- **OpenCV Orientation & Contour Detection**: Determines the contour, primary axes and orientation angle
- **Visual Debugging**: Generates annotated images showing detected orientation
- **Grasp Alignment**: Uses orientation data to optimize the grasp approach angle

3. Grasp Planning

The system incorporates grasp planning based on shape characteristics:

- **Shape-Specific Grasping**: Applies different strategies for crosses and noughts
- **Orientation-Aware Approach**: Aligns gripper approach with optimal grasping points
- **Pre-grasp Positioning**: Ensures secure positioning before final grasp execution

4. Movement Execution

Robust pick and place execution with carefully controlled movements:

- **Pre-grasp Approach**: Precise positioning above the target object
- **Controlled Descent**: Smooth lowering to grasp position with collision avoidance
- **Stable Lifting**: Careful raising of the object to avoid tipping or dropping
- **Transport Trajectory**: Smooth path planning to the goal location
- **Gentle Placement**: Controlled release at the designated target position

5. Error Recovery and Robustness

Multiple strategies ensure reliable task completion:

- **Visual Verification**: Confirms object detection before attempting grasp
- **Movement Validation**: Checks successful execution of each motion phase
- **Adaptive Parameters**: Adjusts grasp and movement parameters based on object properties
- **Visualisation Tools**: Real-time visualisation of planned grasp poses for monitoring

Results

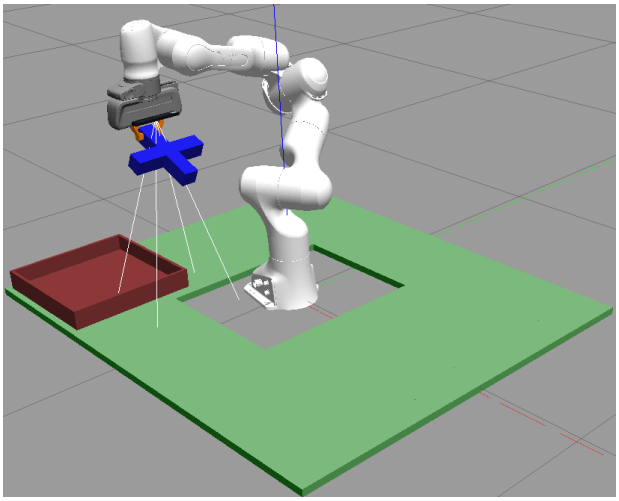
The system detects and analyzes objects precisely, determines optimal grasp points, and executes smooth pick-and-place operations.

Key Parameters

- **Observation Height**: 0.7m - Optimal camera position for object detection
- **Pre-grasp Offset**: 0.125m - Pre-grasp height offset above object pose for approach
- **Lift Height**: 0.6m - Safe height for transporting objects
- **Drop Height**: 0.4m - Appropriate height for object release

Future Improvements

- Integration of force feedback for adaptive grasping
- More precise drop location to ensure object completely fits inside basket



The following figure represents an example of Task 1 while the robot is transporting the object to basket.

Task 2: Shape Classification with Variable Table Height

Important Note:

For Task 2 to correctly compute variable table heights, the robot must be in its default 'ready' position at startup. Occasionally, Gazebo may improperly spawn the robot arm (falling to ground), preventing accurate table height calculation.

Workarounds:

Run and discard the first task execution, then run Task 2 again Alternatively, enable the commented code in the Task 2 callback (lines 315-317):

```
// NOTE TO EVALUATORS: Enable this line to move robot to 'ready' position
// Make sure robot is in default pose
// cw2_movement::moveToReadyPosition(arm_group_, 30.0, 0.6, 0.6, 0.01);
```

This issue is intermittent and difficult to reproduce, so we've left this as an optional configuration.

Overview

Task 2 involves the classification of shapes (noughts and crosses) placed on a variable-height table. The robot must analyze multiple reference shapes and identify which one matches the mystery shape, demonstrating robustness to environmental variations.

Implementation Approach

1. Dynamic Table Height Detection

The system dynamically adapts to varying table heights through a robust detection pipeline:

- **Color-based Detection**: Identifies the green table surface using RGB color analysis
- **Statistical Analysis**: Applies 10th percentile filtering to determine reliable table height
- **Automatic Adaptation**: Adjusts perception parameters based on detected height (0-50mm variation)

2. Shape Observation Sequence

For reliable shape perception, the system follows a methodical observation process:

- **Strategic Positioning & Stabilise**: The camera moves to an optimal viewing angle above the target object and pauses for stabilization, ensuring clear point cloud data.
- **Height-Adjusted Filtering**: Dynamically sets z-filtering range based on detected table height
- **Matching**: Analyzes reference shapes first and then matches the mystery shape

3. Shape Classification Pipeline

Shapes are classified through an advanced computer vision workflow:

- **Point Cloud Clustering**: Isolates individual shapes from the environment
- **Binary Image Projection**: Converts 3D clusters to 2D binary images for analysis
- **Contour Analysis**: Extracts feature information from shape outlines
- **Hole Detection**: Identifies interior contours to distinguish noughts from crosses
- **Topological Classification**: Uses contour hierarchy to detect shape characteristics

4. Orientation Detection

The system determines shape orientation for comprehensive analysis:

- **OpenCV Orientation & Contour Detection**: Determines the contour, primary axes and orientation angle
- **Visual Debugging**: Generates annotated images showing detected orientation
- **Rotation-Invariant Classification**: Ensures consistent identification regardless of orientation

5. Reference Shape Matching

To identify the mystery shape, the system:

- **Feature Comparison**: Compares classification results between mystery and reference shapes
- **Reference Indexing**: Returns the 1-based index of the matching reference shape

Results

The system adapts to table heights (0–50mm), reliably classifies shapes, identifies the matching reference, and provides detailed visualisation for robust performance under varying conditions. Appropriate responses are returned.

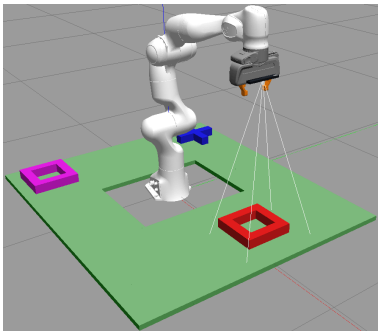
Key Parameters

- **Table Color Detection**: RGB values around (168, 198, 168) with adaptable thresholds
- **Height Processing**: 10th percentile filtering for robust surface detection
- **Hole Ratio Threshold**: 0.2 for distinguishing between noughts and crosses
- **Cluster Distance Threshold**: 0.15m for identifying clusters near target points

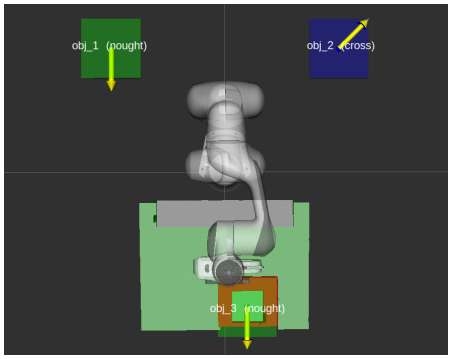
Future Improvements

- Multi-view fusion for more reliable classification
- Shape recognition using machine learning for greater robustness
- 3D analysis techniques that don't require projection to 2D

The following figure represents an example of Task2 while the robot completes scanning the scene:



Gazebo



Rviz

Task 3: Most Common Shape Detection and Manipulation in Cluttered Environment

Overview

Task 3 involves the detection, classification, and manipulation of shapes in a Tic-Tac-Toe-like cluttered environment. The robot must scan the workspace, identify objects (crosses and noughts), detect obstacles, locate the basket, and then pick one instance of the most common shape type and place it in the basket.

Implementation Approach

1. Environment Scanning

The system performs a thorough scan of the workspace using a predefined set of observation poses:

- 8 strategic positions around the workspace perimeter
- Each position offers an optimal viewing angle to detect objects
- There are overlaps between the poses to ensure robust detections

2. Object Detection Pipeline

Objects are detected through a multi-stage perception pipeline:

- **Point Cloud Processing**: Filtering by height range, removing noise, and downsampling
- **Euclidean Clustering**: Segmenting point clouds into individual object clusters
- **Object Classification**:
    - Obstacles (black objects)
    - Crosses (X shapes)
    - Noughts (O shapes, square with hole)
    - Basket (brown target destination)

3. Shape Classification

Our shape classifier uses advanced computer vision techniques:

- **2D Projection**: Converting 3D point clouds to binary images
- **Contour Analysis**: Extracting shape contours and features
- **Hole Detection**: Detecting interior contours to distinguish noughts
- **Orientation Detection**: Determining rotational alignment of shapes

4. Object Selection Strategy

The system employs a sophisticated scoring algorithm to select the optimal object to pick:

- **Distance Score**: Preference for objects at an ideal distance (~0.4m)
- **Clearance Score**: Higher values for objects with more free space around them
- **Orientation Score**: Preference for orientations that facilitate easier grasping
- **Size Score**: Larger objects are prioritized for more reliable grasping

5. Grasp Planning

Dynamic grasp pose generation customized for each shape type:

- **For Crosses**:
    - Identifies the best arm to grasp based on orientation

    - Applies shape-specific lateral offsets (0.02m-0.05m) based on cross size
    - Aligns gripper perpendicular to the selected cross arm

- **For Noughts**:

    - Applies a 45° orientation adjustment to align with edges
    - Uses edge-centered positioning for optimal grip stability
    - Applies size-appropriate lateral offsets (0.05m-0.07m)

6. Robust Motion Planning

The system includes multiple fallback strategies to ensure reliable motion execution:

- **Pre-grasp Positioning**: Precise positioning above target objects with collision avoidance
- **Controlled Descent**: Careful lowering to grasp position
- **Robust Lifting**: Secure lifting with configurable height parameters
- **Transport Planning**: Safe navigation to the basket avoiding obstacles with collision avoidance
- **Error Recovery**: Automatic recovery strategies for failed motion attempts

7. Visualisation Tools

The implementation includes comprehensive visualisation for debugging:

- **Marker Arrays**: 3D visualisation of detected objects and their properties
- **Color Coding**: Different colors for crosses (blue), noughts (green), and obstacles (red)
- **Orientation Arrows**: Displaying detected object orientations
- **Grasp Pose Markers**: Visualisation of planned grasp positions

Results

The system successfully:

- Detects and classifies different shapes with high accuracy
- Handles objects of varying sizes and orientations
- Avoids obstacles while planning paths
- Performs reliable pick and place operations
- Correctly identifies and reports the most common shape type

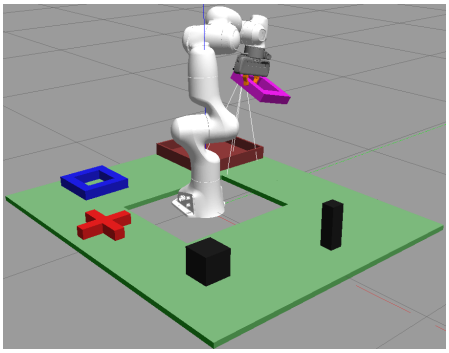
Key Parameters

- `grasp_offset`: 0.15m - Safe height for picking objects
- `lift_height`: 0.5m - Height to lift objects above the workspace
- `drop_height`: 0.4 - Height for releasing objects into the basket

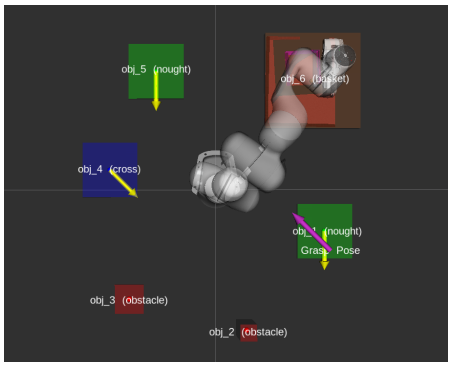
Future Improvements

- Enhanced collision checks after one instance is picked
- Dynamic replanning based on scene

The following figure represents an example of Task2 while the robot is nearing execution completion:



Gazebo



Rviz

License

MIT License