ÉCOLE NATIONALE SUPÉRIEURE DE TECHNIQUES AVANCÉES

ROB311

Daniel Carvalho Frulane de Souza

Diego Bernardes Rafael Pincer

Neng Xu

Shikun Wei

Palaiseau, France 2024

Contents

List of Figures

1 Introduction

This project aims to develop an automated grasping system using a Husky AGV equipped with a UR5 robotic arm, guided by human voice commands. The primary objective is to enable the robot to execute simple grasping tasks based on voice input, such as “place the green cylinder to the right of the blue cylinder." By leveraging vision-based object recognition and a simplified voice processing technique, the system will detect objects and perform the desired actions using a gripper attached to the end-effector of the robotic arm.

1.1 Motivation

Robotics appliances are gradually taking over our day-to-day environment, with appliances such as autonomous vehicles, delivery drones, and soon-to-be humanoid home assistants. Within this context, handling human-to-robot interactions presents itself as an important side of their conception, once it is necessary to avoid any type of disparity between what would be expected of a humanized robot in comparison with its actual behavior. This way, robots can be inserted without rejection into human lives to better aid us with our daily activities.

1.2 Problem Statement

The project involves creating a system that allows an autonomous mobile robot to execute tasks based on human voice input. Specifically, the robot should be able to detect objects in its surroundings, point to them, grasp them, and place them in locations as specified by the user’s command. Due to time constraints and limited access to the robot, the system must be simplified and optimized for rapid development and deployment. Once traditional natural language processing techniques for voice command recognition are time-consuming and complex, a more straightforward approach to voice recognition will be employed, relying on keyword extraction rather than full sentence processing.

1.3 Objectives

The main objectives of this project are as follows:

1. Develop a system that enables the Husky AGV with a UR5 robotic arm to respond to human voice commands and perform grasping tasks.
2. Implement a voice command processing module that extracts key information (such as object color, type, and relative positioning) from user input.
3. Use ArUco Tags attached to objects for accurate object recognition and pose estimation.
4. Control the robotic arm’s gripper to pick up the desired object and place it in the correct position based on the user’s instructions.

2 Methodology

The project workflow begins with Voice Recognition, which converts spoken commands into text. The text is then processed by the NLP module to extract actionable elements, such as objects, actions, and spatial relations. Next, Object Recognition identifies and localizes the relevant objects in the environment. Using this information, the UR5 Control via MoveIt module plans and executes precise movements of the robotic arm. Finally, the Task Execution module carries out the desired actions, such as grasping or placing objects, ensuring the robot fulfills the user's command accurately.

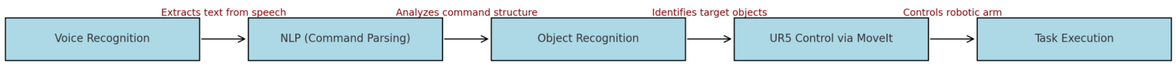


Figure 1: Workflow

The following approach will be adopted to achieve the project objectives:

2.1 Voice Command Processing

Instead of using a full-scale natural language processing (NLP) model, the system will employ a simplified method of keyword extraction. Each voice command provided by the user will contain keywords that specify the task. For example, in the command "place the green cylinder to the right of the blue cylinder," the keywords are "green," "cylinder," "right," and "blue." These extracted keywords will then be mapped to specific task instructions and object identifiers, allowing the robot to understand the required action.

2.2 Object Recognition Using ArUco Tags

For accurate and simplified object recognition, ArUco Tags will be attached to the surface of the objects to be manipulated. The robot's camera system will detect these tags to identify the objects and calculate their positions relative to the robotic arm. This method offers a reliable and simple way to recognize the objects while avoiding the complexities of using deep learning-based object detection models.

2.3 Task Execution by Robotic Arm

After processing the voice input and identifying the objects using the camera, the robot will initiate the appropriate task. The UR5 robotic arm will move to the location of the object, using the visual feedback from the camera to align the gripper accurately with the target. The gripper will grasp the object and move it to the specified location as per the command (e.g., to the right of another object). The entire task will be carried out in a coordinated manner, with the Husky AGV providing mobility and the UR5 arm handling manipulation tasks.

2.4 Simplifications for Time Constraints

Given the limited project time, several simplifications will be introduced:

1. **Keyword Extraction for Voice Processing** Instead of using advanced language models for voice command interpretation, a rule-based keyword extraction method will be used. This drastically reduces the complexity of natural language understanding while still allowing the robot to perform basic tasks based on human input.

2. **ArUco Tags for Object Recognition:** Instead of relying on complex visual recognition algorithms, ArUco Tags will be used for object detection. These tags are easy to detect, and their precise positioning helps the robot calculate the relative pose of the objects, simplifying the task of grasping and placement.

3. **Predefined Task Codes** Each keyword extracted from the voice command will correspond to predefined task codes that are easier for the robot to interpret and execute. This reduces the time required for developing more sophisticated mapping and decision-making algorithms.

3 Expected Outcome

By the end of the project, the system should be able to:

1. Process basic voice commands using keyword extraction.

2. Detect objects accurately via the camera using ArUco Tags.

3. Perform simple grasping tasks such as picking up an object and placing it in a specific location.

This proof-of-concept system will serve as a foundation for further development, where more complex tasks and advanced natural language processing techniques can be incorporated.

4 Connection to the robot

Establishing a connection between the computer and the robotic components of the system was a challenging and iterative process. Initially, we attempted to connect the Husky robot and use it to send commands to the UR5 robotic arm. However, due to version compatibility issues with certain ROS libraries, this integration was not successful. Subsequently, while the Husky robot was under maintenance, we decided to directly connect the computer to the UR5 robotic arm, which enabled successful control of the arm. The detailed steps are outlined below:

1. **Install Required ROS Libraries:** The necessary ROS libraries, `ur-robot-driver` and `ur-calibration`, were installed to enable communication and control of the UR5 robotic arm.

2. **Establish Network Connection:** The UR5 robotic arm was connected to the computer via an Ethernet cable. The computer's Ethernet IP address was configured to ensure compatibility with the UR5 robot's network. On the UR5 teach pendant, the `External Control` program was set up with a host IP matching the computer's Ethernet IP. For Docker-based ROS setups, the container was configured to use the host network mode.

3. **Launch UR5 Driver:** The `ur-robot-driver`'s `bringup.launch` file was executed with the correct IP address of the UR5 robot. This step established a communication link between ROS and the UR5 robotic arm.

4. **Control Using MoveIt:** To control the UR5 robotic arm in real-time, the planning_execution.launch and demo.launch files from the `MoveIt` library were executed. These files allowed for trajectory planning and execution. In the later stages of the project, custom commands replaced the demonstration scripts to achieve specific task requirements.

5 Voice Command Processing

5.1 Voice Recognition

The voice recognition module is responsible for converting human speech into textual commands that the robot can interpret. This is achieved using the speech_recognition library and a microphone configured with specific audio parameters. The system records audio for a fixed duration and processes the recorded audio file to extract text.

The recording process utilizes the pyaudio library to capture audio from a selected microphone device. Audio data is saved as a .wav file, ensuring compatibility with speech recognition tools. Once the audio is captured, the system uses Google's Web Speech API to convert the speech into text. This approach offers reliable recognition for simple commands, minimizing the need for extensive model training or customization.

To integrate with the robot, the recognized text is published to a ROS topic (recognized_speech). This modular design ensures that the voice recognition module operates independently, providing real-time textual input for subsequent processing. Additionally, the system logs any recognition issues, such as unclear speech or API errors, allowing for easy debugging and system monitoring.

5.2 Natural Language Processing

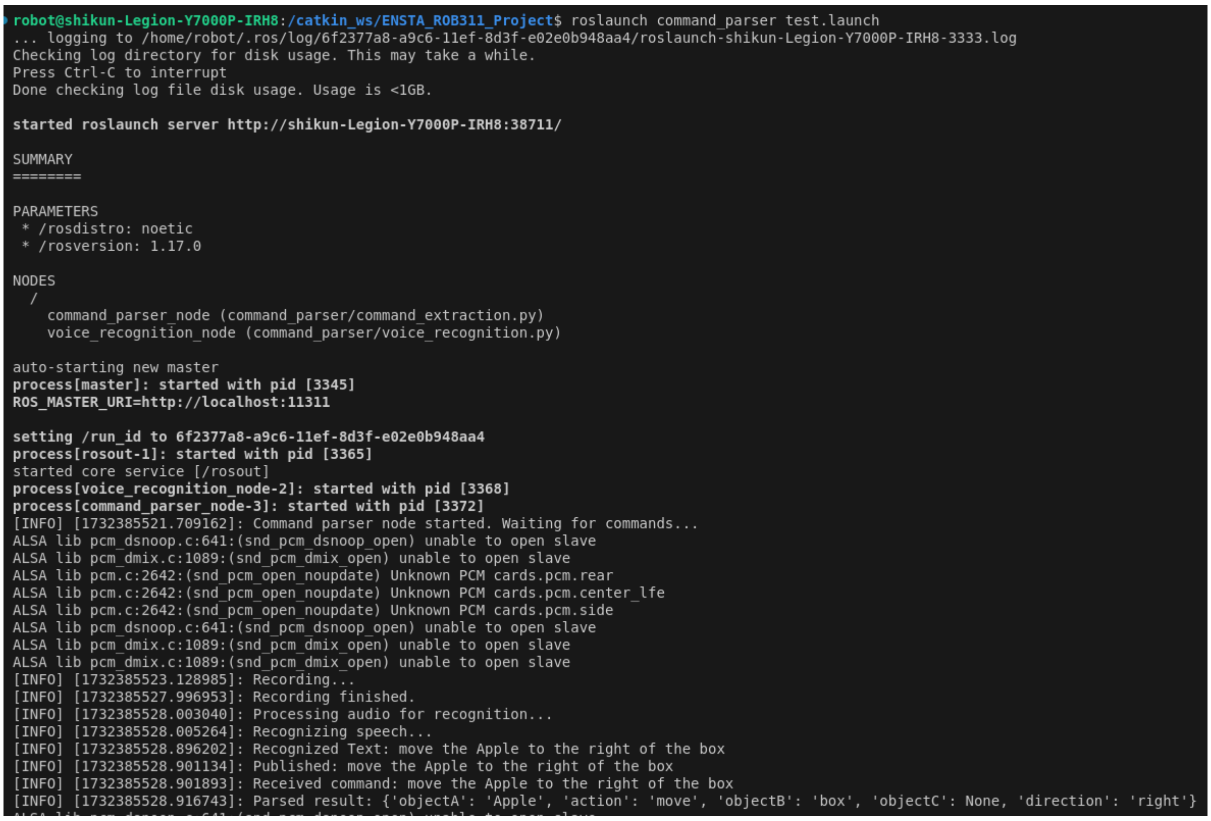
The NLP module processes the recognized text to extract structured information for robotic tasks. Instead of employing advanced and computationally intensive language models, this project leverages a rule-based parsing approach using the **spacy** library. This method efficiently identifies essential components of the command while maintaining simplicity and reliability.

The text is tokenized and analyzed to identify grammatical roles, such as actions, objects, and spatial relationships. The parser extracts key elements, including the primary object (objectA), the action (move), the relative direction (left, right), and additional objects (objectB and objectC) required for spatial instructions. For example, a command like "Move the green cylinder to the right of the blue box" is parsed into structured data indicating the object to be moved, the action, and its target position relative to another object.

This design ensures that the robot receives a clear and actionable command structure, simplifying downstream task execution. The system also supports more complex spatial instructions, such as identifying multiple objects for relations like "between." The parsed commands are logged for validation and sent to the robot's control modules via ROS topics.

By focusing on keyword-based extraction and rule-driven grammar interpretation, the NLP module strikes a balance between accuracy and computational efficiency, making it well-suited for real-time applications with limited processing resources.

For example, if we give a command "move the apple to the right of the box", the NLP module will finally return a dictionary which contains "action: move, objectA: Apple, objectB: box, objectC: None, direction: right", as shown in the figure below.



```
robot@shikun-Legion-Y7000P-IRH8:/catkin_ws/ENSTA_ROB311_Project$ roslaunch command_parser test.launch
... logging to /home/robot/.ros/log/6f2377a8-a9c6-11ef-8d3f-e02e0b948aa4/roslaunch-shikun-Legion-Y7000P-IRH8-3333.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://shikun-Legion-Y7000P-IRH8:38711/

SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.17.0

NODES
  /
    command_parser_node (command_parser/command_extraction.py)
    voice_recognition_node (command_parser/voice_recognition.py)

auto-starting new master
process[master]: started with pid [3345]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to 6f2377a8-a9c6-11ef-8d3f-e02e0b948aa4
process[rosout-1]: started with pid [3365]
started core service [/rosout]
process[voice_recognition_node-2]: started with pid [3368]
process[command_parser_node-3]: started with pid [3372]
[INFO] [1732385521.709162]: Command parser node started. Waiting for commands...
ALSA lib pcm_dsnoop.c:641:(snd_pcm_dsnoop_open) unable to open slave
ALSA lib pcm_dmix.c:1089:(snd_pcm_dmix_open) unable to open slave
ALSA lib pcm.c:2642:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.rear
ALSA lib pcm.c:2642:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.center_lfe
ALSA lib pcm.c:2642:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.side
ALSA lib pcm_dsnoop.c:641:(snd_pcm_dsnoop_open) unable to open slave
ALSA lib pcm_dmix.c:1089:(snd_pcm_dmix_open) unable to open slave
ALSA lib pcm_dmix.c:1089:(snd_pcm_dmix_open) unable to open slave
[INFO] [1732385523.128985]: Recording...
[INFO] [1732385527.996953]: Recording finished.
[INFO] [1732385528.003040]: Processing audio for recognition...
[INFO] [1732385528.005264]: Recognizing speech...
[INFO] [1732385528.896202]: Recognized Text: move the Apple to the right of the box
[INFO] [1732385528.901134]: Published: move the Apple to the right of the box
[INFO] [1732385528.901893]: Received command: move the Apple to the right of the box
[INFO] [1732385528.916743]: Parsed result: {'objectA': 'Apple', 'action': 'move', 'objectB': 'box', 'objectC': None, 'direction': 'right'}
```

Figure 2: Command detection

6 Object Recognition Using ArUco Tags

For the purposes of this project, there are 4 different type of objects whose position can be detected by image recognition tools. For a straightforward recognition of their positions, ArUco markers were used with central positions equal to their respective object's central position. This way, the system will be able to detect without error where each component is localized in order to perform an action:

1. **Reference Plane:** 4 markers that delimit the coordinate plane used as a reference for the surface where the objects will be placed.

2. **Object:** 1 marker for each object that the robot can interact with.

3. **Object Placement:** 1 marker for each position where an object can be placed.

4. **Robot Arm:** 1 marker placed at the tip of the robot's arm.

In the following image, used to test the algorithm, there are present the markers used for the Reference Plane and the Object Placement:

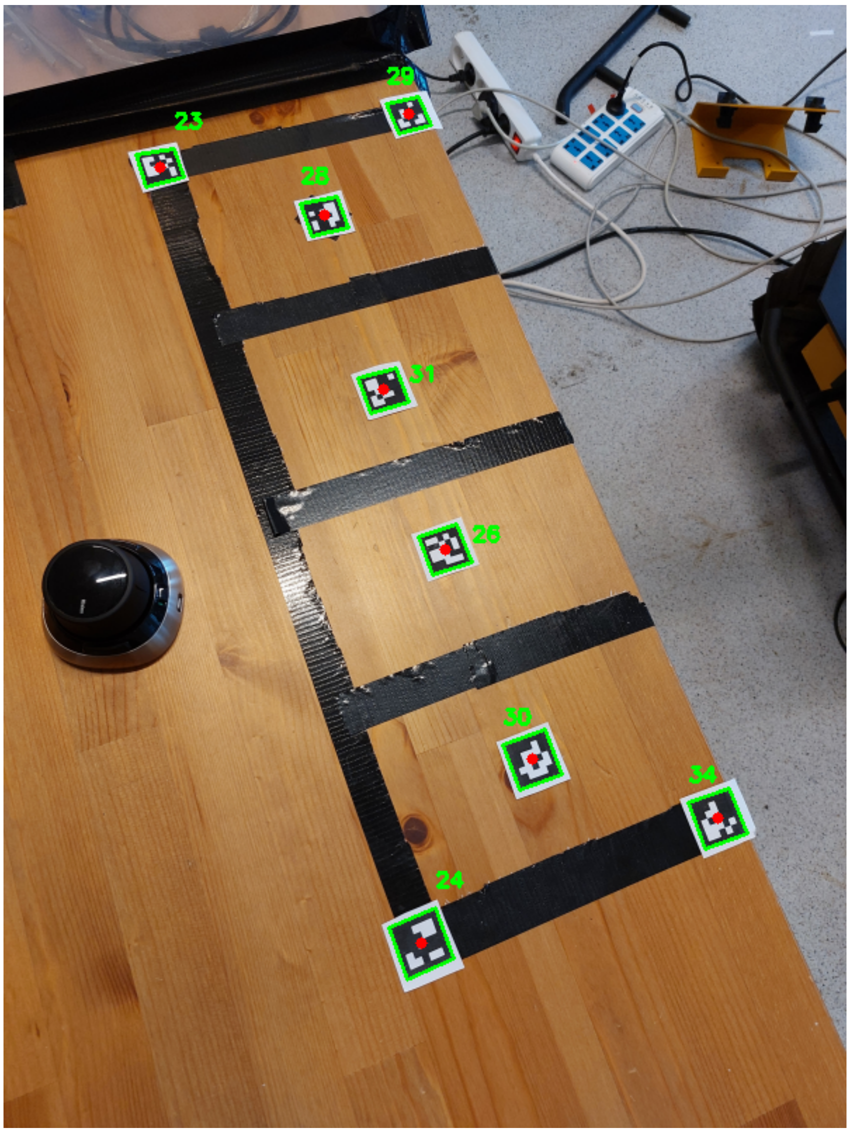


Figure 3: ArUco application in the project.

Considering, also, that the camera used for image processing has a perspective projection, as opposed to an orthographic projection (as seen in the image 5), and that the chosen objects are tridimensional and have heights that cannot be factored only by the superior view provided by the camera, there should be an additional processing dedicated to correcting the object's real base position's distortion that is generated by the perspective projection. Given that this distortion is proportional to the distance from the center of the camera, and that the considered objects have roughly the same height, we can approximate the displacement necessary to correct this distortion by a fixed linear factor, that can be calculated from trigonometry once the camera distance from the reference surface is known.

base position = top position - distortion coefficient * top distance from center

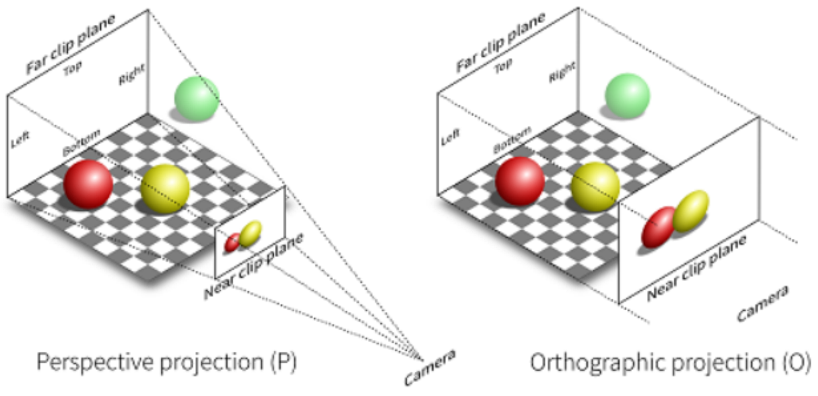


Figure 4: Projection types.

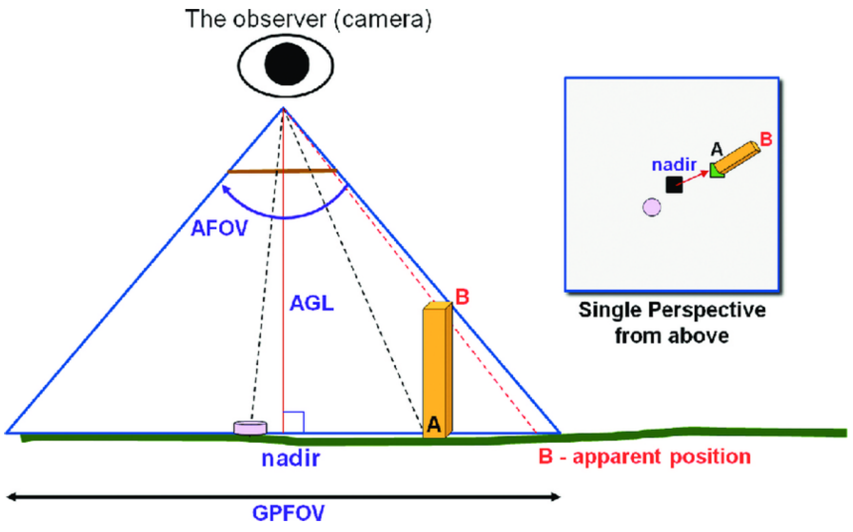


Figure 5: Perception of perspective projection distortion.

Considering that the robot's connections were unavailable, the developed image treatment routine was not implemented along with the rest of the features. Nonetheless, the only modification required to do so would be constantly sending the detected coordinates and their associated references through the ROS interface, making use of a broadcaster node.

A more performant algorithm should consider the individual object heights for calculating the projection distortion, and could eliminate the need of knowing the camera placement by analyzing the inclination of the ArUco markers, though these alternatives still could present some variations when compared to the real distortion values.

7 Path Planning of UR5 Robotic Arm

This section describes the implementation of path planning tasks of robotic grasping using the UR5 robotic arm, mounted on the Husky robot.

7.1 Task description

The task is structured into two distinct and interdependent phases, each focusing on a specific aspect of the system's operational workflow: the Grasp-and-Place Phase and the

Transportation Phase. These phases are designed to address the challenges associated with precise object manipulation and efficient path planning.

- **Grasp-and-Place Phase:** During this phase, the system employs Resolved-Rate Motion Control to handle objects with high precision. This approach allows for smooth and accurate manipulation of objects by continuously adjusting the robot's end-effector position and orientation in response to real-time feedback. The precise control ensures reliable grasping, careful placement, and minimized errors during the interaction with the environment, making this phase crucial for achieving the desired level of operational accuracy.
- **Transportation Phase:** Once the object has been successfully grasped and positioned, the task transitions to the Transportation Phase, where the system leverages the RRT-Connect (Rapidly-exploring Random Trees Connect) algorithm for path planning. This algorithm is chosen for its efficiency in exploring high-dimensional spaces and its ability to find feasible paths quickly, even in complex and constrained environments. The RRT-Connect algorithm ensures that the object is transported to its target location along an optimized and collision-free trajectory, maintaining the overall efficiency and safety of the operation.

7.2 Grasp-and-Place Phase

When executing the grasp-and-place phase, the end-effector of robotic arm needs to approach the target pose in a linear motion (including the process of approaching the target object versus placing the object smoothly). If the robotic arm approaches the object to be gripped in a non-linear motion, it may not be possible to perform the gripping action in a suitable position. Similarly, when placing an object, non-linear motion may result in an inability to place the object smoothly on a table. And in this phase, there are not many obstacles in such a short distance. We neglect the process of obstacle avoidance here.

7.2.1 Resolved-rate Motion Control

Resolved-rate motion control (RRMC) is an elegant method to generate straight-line motion of the end effector. RRMC is a direct application of the first-order differential equation:

$$^{0}\nu = {}^{0}\mathbf{J}(\mathbf{q})\,\dot{\mathbf{q}}$$

where $^{0}(\cdot)$ denotes quantities expressed in the base frame. In this formulation, $^{0}\nu$ represents the desired velocity of the end effector in the base frame, $^{0}\mathbf{J}(\mathbf{q})$ is the robot's Jacobian matrix in the base frame, and $\dot{\mathbf{q}}$ represents the vector of joint velocities. RRMC utilizes the inverse of the above equation:

$$\dot{\mathbf{q}} = {}^{\mathbf{0}}\mathbf{J}(\mathbf{q})^{\mathbf{-1}}\ {}^{\mathbf{0}}\nu$$

This equation can only be solved when the Jacobian matrix $\mathbf{J}(\mathbf{q})$ is square. These conditions are satisfied when the robot has exactly 6 degrees of freedom (DOF), as is the case for the UR5 robot.

To plan and execute straight-line paths for the UR5 robot using RRMC, the desired velocity of the end effector $^{0}\nu$ is specified. The corresponding joint velocities $\dot{\mathbf{q}}$ are computed by solving the above equation. By integrating the joint velocities over time, the UR5 can achieve smooth and accurate end-effector motion along the desired trajectory.

7.2.2 Position-Based Servoing with RRMC

To accomplish the grasp-and-place phase, RRMC is integrated into a closed-loop pose control framework, referred to as Position-Based Servoing (PBS). Using PBS, the end-effector is guided along a straight-line trajectory in the robot's task space toward a desired pose. This method leverages an error vector **e**, which quantifies the translational and rotational discrepancies between the current and desired end-effector poses.

The error vector is defined as:

$$\mathbf{e} = \begin{pmatrix} \tau\left({}^{0}\mathbf{T}_{e^*}\right) - \tau\left({}^{0}\mathbf{T}_{e}\right) \\ \alpha\left(\rho\left({}^{0}\mathbf{T}_{e^*}\right)\rho\left({}^{0}\mathbf{T}_{e}\right)^{\top}\right) \end{pmatrix} \in \mathbb{R}^{\mathbf{6}},$$

where:

- The top three rows represent the translational error in the world (base) frame.
- The bottom three rows represent the rotational error in the world (base) frame.

The function $\rho(\cdot)$ extracts the rotational component of a homogeneous transformation matrix, returning the 3D rotation as a 3×3 matrix. The function $\tau(\cdot)$ extracts the translational component, returning the 3D position as a vector. Both functions are used to separate rotation and translation information encoded in the matrix. ${}^{0}\mathbf{T}_{e}$ denotes the current end-effector pose derived from the robot's forward kinematics, while ${}^{0}\mathbf{T}_{e^*}$ is the desired end-effector pose. The function $\alpha(\cdot) : \mathbf{SO(3)} \mapsto \mathbb{R}^{\mathbf{3}}$ transforms a rotation matrix into its equivalent Euler vector.

We note r_{ij} the elements of the rotation matrix $\mathbf{R} = \rho\left({}^{\mathbf{0}}\mathbf{T}_{\mathbf{e}^*}\right)\rho\left({}^{\mathbf{0}}\mathbf{T}_{\mathbf{e}}\right)^{\top}$. For the special case where $(r_{11}, r_{22}, r_{33}) = (1, 1, 1)$, the Euler vector is:

$$\alpha(\mathbf{R}) = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}^{\top}.$$

Otherwise, it is:

$$\alpha(\mathbf{R}) = \frac{\pi}{2}\begin{pmatrix} r_{11}+1 \\ r_{22}+1 \\ r_{33}+1 \end{pmatrix}.$$

The PBS method is implemented by calculating the error vector **e** from two poses, **T** (current pose) and $\mathbf{T_d}$ (desired pose) which is given by the detection of Aruco tag from the camera. The 'angle_axis' function encapsulates these computations to generate the required error vector, enabling precise motion control in the robot's task space.

7.2.3 Proportional Gain in PBS

To construct the PBS scheme, the error term **e** is used to define the end-effector velocity v in the RRMC equation at each time step as:

$$v = \mathbf{ke},$$

where k is a proportional gain term that controls the rate of convergence to the desired pose. This gain determines how aggressively the system corrects errors, balancing speed and stability in trajectory execution. The error vector **e** is non-homogeneous because its elements have different physical units—translations are measured in meters, while rotations are measured in radians. To account for this disparity, k can be extended to a diagonal matrix, allowing for separate gain values for translational and rotational components:

$$\mathbf{k} = \mathrm{diag}(\mathbf{k_t}, \mathbf{k_t}, \mathbf{k_t}, \mathbf{k_r}, \mathbf{k_r}, \mathbf{k_r}),$$

where k_t is the translational motion gain, and k_r is the rotational motion gain.

7.2.4 Results of grasp-and-place phase

The results of grasp-and-place phase are shown in figure 6. Figure 6a represents the actions of the robot during the grasping phases and figure 6b represents the actions of the robot during the placing phases. The end of the robot arm moves in a uniform linear motion. The movement of the robotic arm is magnified in the figure to make the effect look more obvious. In the real grasping task the robot arm movement is 1/10 of its amplitude.

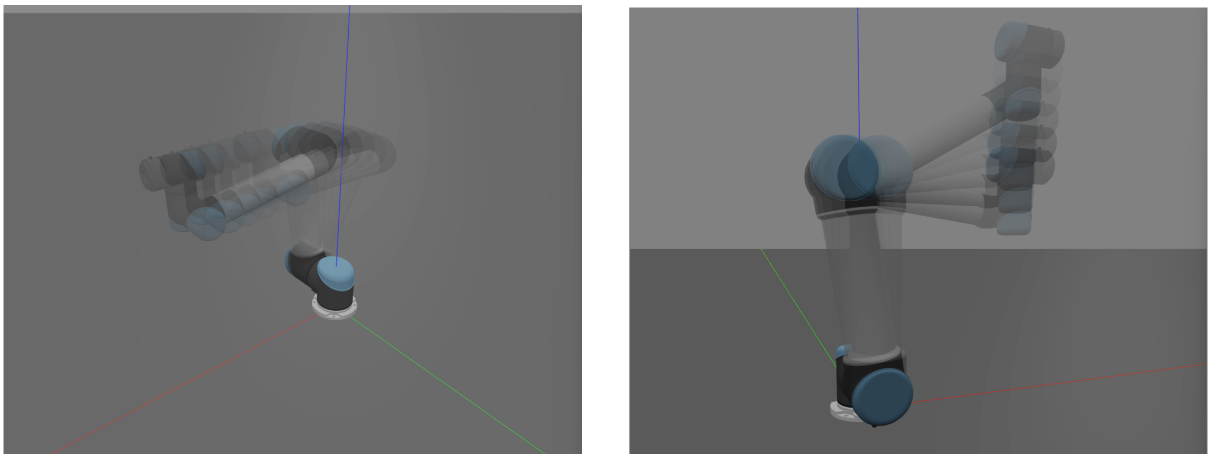

(a) moving forward (b) moving downward

Figure 6: Results of RRMC control

7.3 Transportation Phase

During the transportation phase, one of the most critical objectives is to ensure that the robot avoids obstacles within its workspace. These obstacles may include other manipulable objects, the workspace surface (e.g., a table), or parts of the robot itself. To achieve effective obstacle avoidance, the Rapidly-Exploring Random Tree (RRT) algorithm is utilized as the path planning method. RRT is particularly advantageous in this context due to its ability to handle high-dimensional configuration spaces efficiently, making it ideal for robots with multiple degrees of freedom, just as UR5 in this task. By considering the obstacles explicitly during path generation, RRT ensures that the planned trajectories are collision-free.

7.3.1 Obstacle Definition in the Environment

In this task, all areas in the environment that could potentially contain obstacles are explicitly labeled as 'obstacle'. Specifically, the table, the UR5 robotic arm, the Husky chassis, the ground, and the positions on the table designated for placing graspable objects are marked as obstacles.

To simplify the task, there are four predefined positions where graspable objects may be placed (some positions may remain unoccupied). Regardless of whether an object is present at these locations, each position is marked as an 'obstacle' based on the maximum volume of the object that could potentially occupy that position. This approach ensures that the robotic arm avoids all possible collision zones during the execution of the transportation phase.

During the grasp-and-place phase, the robotic arm utilizes RRMC, which allows for precise and direct control of its motion. As a result, the predefined 'obstacle' definitions in the environment do not affect the grasp-and-place phase, since the focus of this phase is on the controlled interaction between the robot and the target object rather than global path planning. This separation of concerns ensures efficient and collision-free task execution across both phases.

7.3.2 The RRT and RRT-Connect Algorithms

The Rapidly-exploring Random Trees (RRT) algorithm works by incrementally building a tree rooted at the robot's starting configuration. The primary objective is to explore the configuration space efficiently by random sampling. The key steps of the RRT algorithm are:

1. **Random Sampling:** A random configuration is sampled from the robot's configuration space.
2. **Nearest Node Selection:** The closest node in the existing tree to the sampled configuration is identified.
3. **Tree Expansion:** A new node is added to the tree by extending from the nearest node toward the sampled configuration within a predefined step size.
4. **Collision Checking:** The new node and the connecting edge are checked for collisions to ensure feasibility.
5. **Termination:** The process continues until the goal configuration is reached or a specified computational budget is exceeded.

RRT's strength lies in its ability to rapidly explore large configuration spaces. However, its paths are often suboptimal, and convergence to a solution can be slow in some cases.

7.3.3 Enhancements in the RRT-Connect Algorithm

The RRT-Connect algorithm is a significant enhancement of the original RRT algorithm. RRT-Connect improves upon the original RRT algorithm by growing two trees simultaneously: one from the start configuration and the other from the goal configuration. These trees aim to connect in the configuration space to form a feasible path. The steps involved in RRT-Connect are:

1. **Tree Initialization:** Two trees are initialized, one at the start configuration ($\mathtt{T}_{\mathtt{start}}$) and one at the goal configuration ($\mathtt{T}_{\mathtt{goal}}$).
2. **Random Sampling:** A random configuration is sampled from the configuration space.

3. **Tree Growth:** The sampled configuration is used to extend one tree ($\mathtt{T}_{\mathtt{active}}$) by adding a new node and edge, subject to collision checks.

4. **Attempt Connection:** The newly added node in $\mathtt{T}_{\mathtt{active}}$ attempts to connect to the nearest node in the other tree ($\mathtt{T}_{\mathtt{passive}}$). If successful, a complete path is formed.

5. **Tree Switching:** The roles of the active and passive trees are alternated in each iteration.

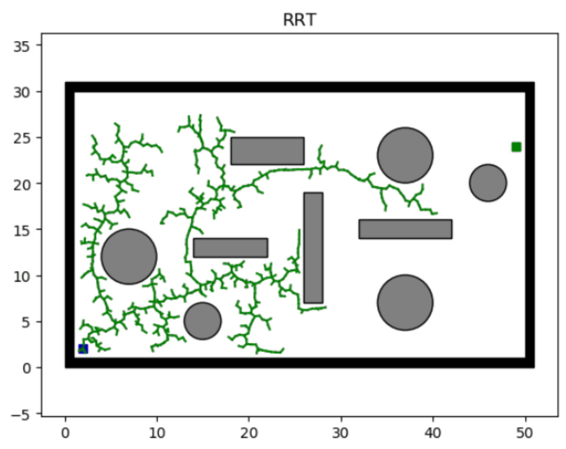
6. **Termination:** The process terminates when the two trees connect or a computational limit is reached.

The pseudocode for the RRT-Connect algorithm is presented below:

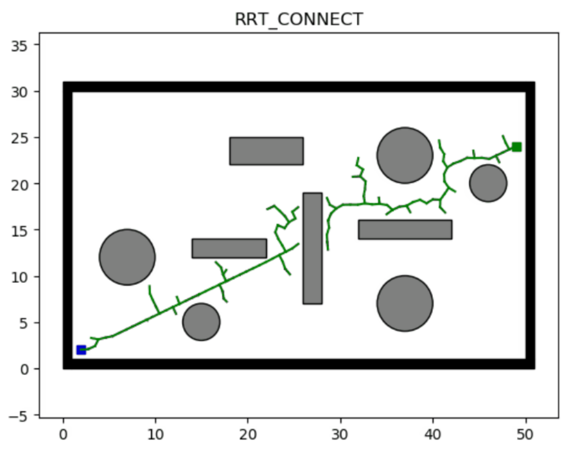
Algorithm 1 RRT-Connect Algorithm

```
1:  Initialize trees: $\mathtt{T}_{\mathtt{start}}$ at start configuration, $\mathtt{T}_{\mathtt{goal}}$ at goal configuration
2:  **while** no connection between trees and computational limit not reached **do**
3:      Sample a random configuration $\mathtt{q}_{\mathtt{rand}}$
4:      Extend $\mathtt{T}_{\mathtt{active}}$ toward $\mathtt{q}_{\mathtt{rand}}$ to add new node $\mathtt{q}_{\mathtt{new}}$
5:      **if** $\mathtt{q}_{\mathtt{new}}$ successfully added and collision-free **then**
6:          Attempt to connect $\mathtt{q}_{\mathtt{new}}$ to $\mathtt{T}_{\mathtt{passive}}$
7:          **if** connection successful **then**
8:              Return complete path
9:          **end if**
10:     **end if**
11:     Swap roles of $\mathtt{T}_{\mathtt{active}}$ and $\mathtt{T}_{\mathtt{passive}}$
12: **end while**
13: Return failure if no connection found
```

The figure 7 shows the the effect of RRT and RRT-Connect algorithms for path planning in 2D space. It is clear from the picture that the RRT-Connect algorithm significantly optimises the computation time by sampling from the start and end points separately.



(a) RRT



(b) RRT-Connect

Figure 7: Exemples of RRT algorithm

7.3.4 Controlling the UR5 Robot using MoveIt

In the context of controlling the UR5 robotic arm, the RRT-Connect algorithm is implemented within the MoveIt motion planning framework in ROS Noetic. MoveIt provides a robust and versatile platform for motion planning, collision checking, and kinematic solutions.

To utilize RRT-Connect for the UR5 arm, the following procedure is applied:

1. The UR5's configuration space, encompassing all six degrees of freedom, is defined. This includes specifying the joint limits and ensuring that the robot's kinematic chain is accurately modeled.

2. The workspace is modeled in MoveIt using a combination of URDF (Unified Robot Description Format) for the robot's physical structure and a collision environment map.

3. All obstacles in the environment are defined with 'moveit_msgs.msg.CollisionObject()', thus added into the environment.

4. A planning scene is established in MoveIt to integrate the robot's kinematic model with the environment. This includes specifying the start and goal poses for the end effector of UR5 arm.

5. MoveIt's built-in motion planning pipeline invokes the RRT-Connect algorithm as the planner.

6. Once a feasible path is found, the trajectory is executed on the UR5 arm via ROS controllers, ensuring precise motion.

7. MoveIt enables real-time monitoring of the robot's state and the environment in the RVIZ GUI.

7.4 Application of UR5 Motion Planning

This project integrates global path planning in controlling the UR5 robotic arm. The repository is available at: `https://github.com/Nelx-SJTU/ENSTA_ROB311_Project.git`

Environment description: For grasp-and-place tasks, RRMC ensures precise straight-line motion by controlling the end-effector in task space. The RRT-Connect algorithm is employed to compute collision-free paths within the MoveIt framework.

Obstacles in the environment, including tables, other objects, and robot components, are defined in the file located at: `src/ur5_move/config/environments/obstacles.yaml`. Each obstacle is described by an ID, frame of reference, dimensions (as a box), and pose (position and orientation). For example:

- `id: "table"`
- `frame: "world"`
- `dimensions: [0.5, 1.0, 0.2]` (length, width, height)
- `pose: [x, y, z, roll, pitch, yaw].`

Target poses for these operations are configured in: `src/ur5_move/config/poses/grab_poses.yaml`, where positions are given in terms of `x, y, z` coordinates, and orientations in Euler angles (converted to quaternions during execution). An example definition includes:

- `position: [x, y, z]`
- `orientation: [roll, pitch, yaw].`

8 Conclusion

The proposed system components were validated and the expected functionalities were correctly implemented, though some interactions between modules could not be performed, even if they should not be difficult to implement. Using the UR5 robotic arm of the Husky robot is a powerful tool for simple object interaction applications, and further progress can be done to enhance its capabilities in all development topics of this project. This way, a completely flexible application for grabbing and identifying objects could be developed to attain a real case human-to-robot interaction demand.