**Robotics and AI**

**Milestone Three**

December 19th, 2024

Khalil, Vishal & Harsha

The following document represents the Final Demonstration and Writeup for out Group project:

1. Git Link: <https://github.com/HAR5HA-7663/Stretch2_Cluster_navigation_grasp>
2. Responsibilities:
   1. Khalil: Handle clustering failures by adjusting detection thresholds and/or scanning location
   2. Harsha: Implement/Improve the path planning algorithm
   3. Vishal: Implement/Improve the grabbing algorithm
3. Individual Component:
   1. Khalil:   
      The robot in our class is designed to perform clustering tasks to analyze data from its environment, but it sometimes fails in the process. These failures may occur due to factors such as improper detection thresholds, insufficient number of cloud points, or challenging environmental conditions. My task in this group project was to improve the software of the robot such that, in case of failure of clustering, it automatically tries to adjust and change parameters such as detection thresholds or the number of cloud points, and moreover if the issue is not solved using different parameters, then the robot should move to a new location and try the clustering again. This approach allows the robot to adapt in improving the reliability and efficiency of the robot's clustering process so that even in non-ideal conditions, the robot could still finish the end goal.

Existing Component:

My focus was on ros\_object\_query.py file, specifically in the “top1\_cluster\_service” method. The existing code looked like this before my changes:

A computer screen shot of a program

Description automatically generated

In this context:  
  
- request.main\_query is a parameter of the GetClusterRequest service request.  
- It acts as a key or label to identify which clusters to process or query.  
- The method create\_and\_publish\_clusters uses request.main\_query to filter and generate clusters based on the query.

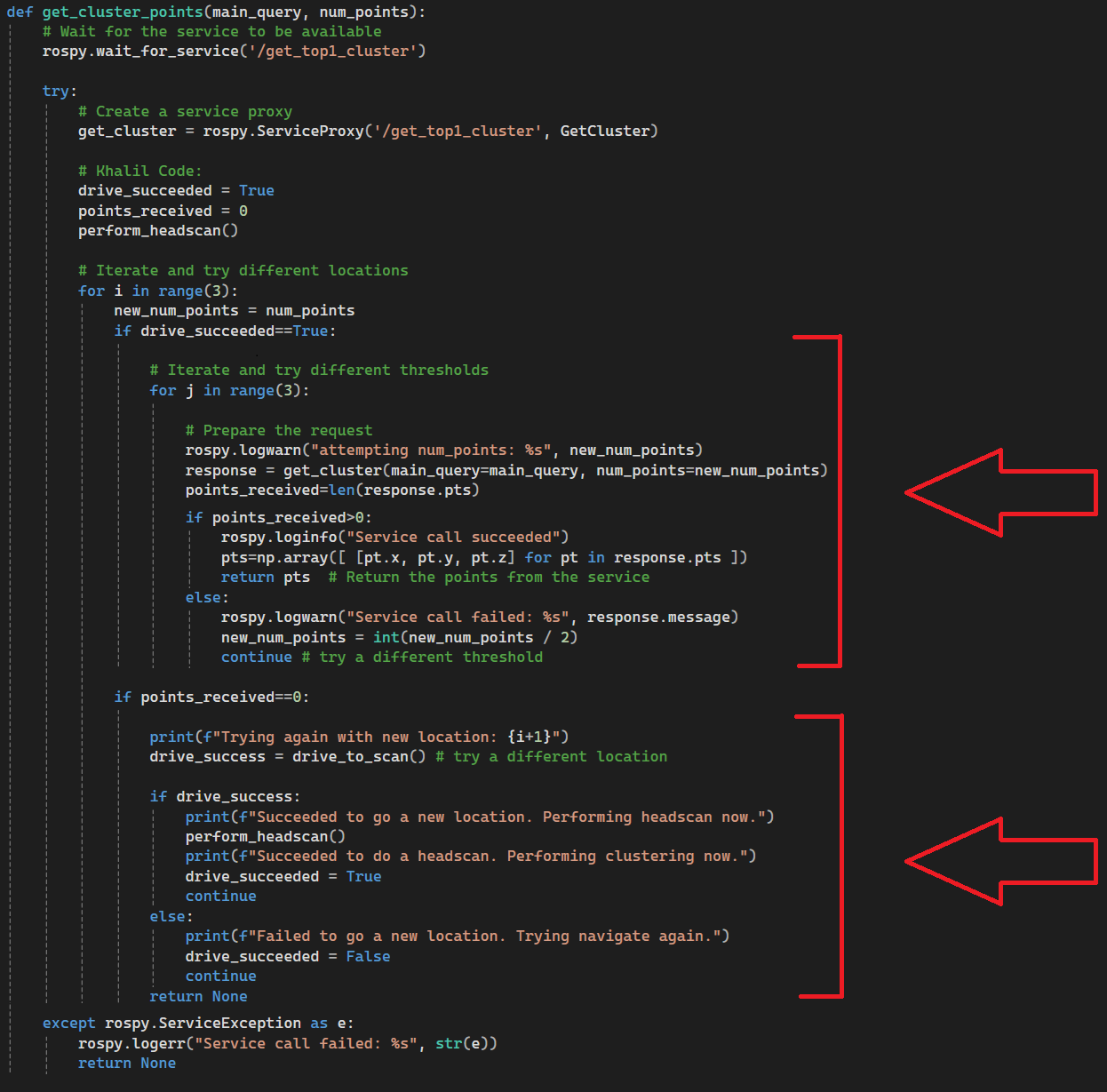
​In poke.py, the function call\_cluster\_service makes a call to the /get\_top1\_cluster service where main\_query is passed to the service as request.main\_query:

response = get\_cluster(main\_query=main\_query, llm\_query=llm\_query, num\_points=num\_points)

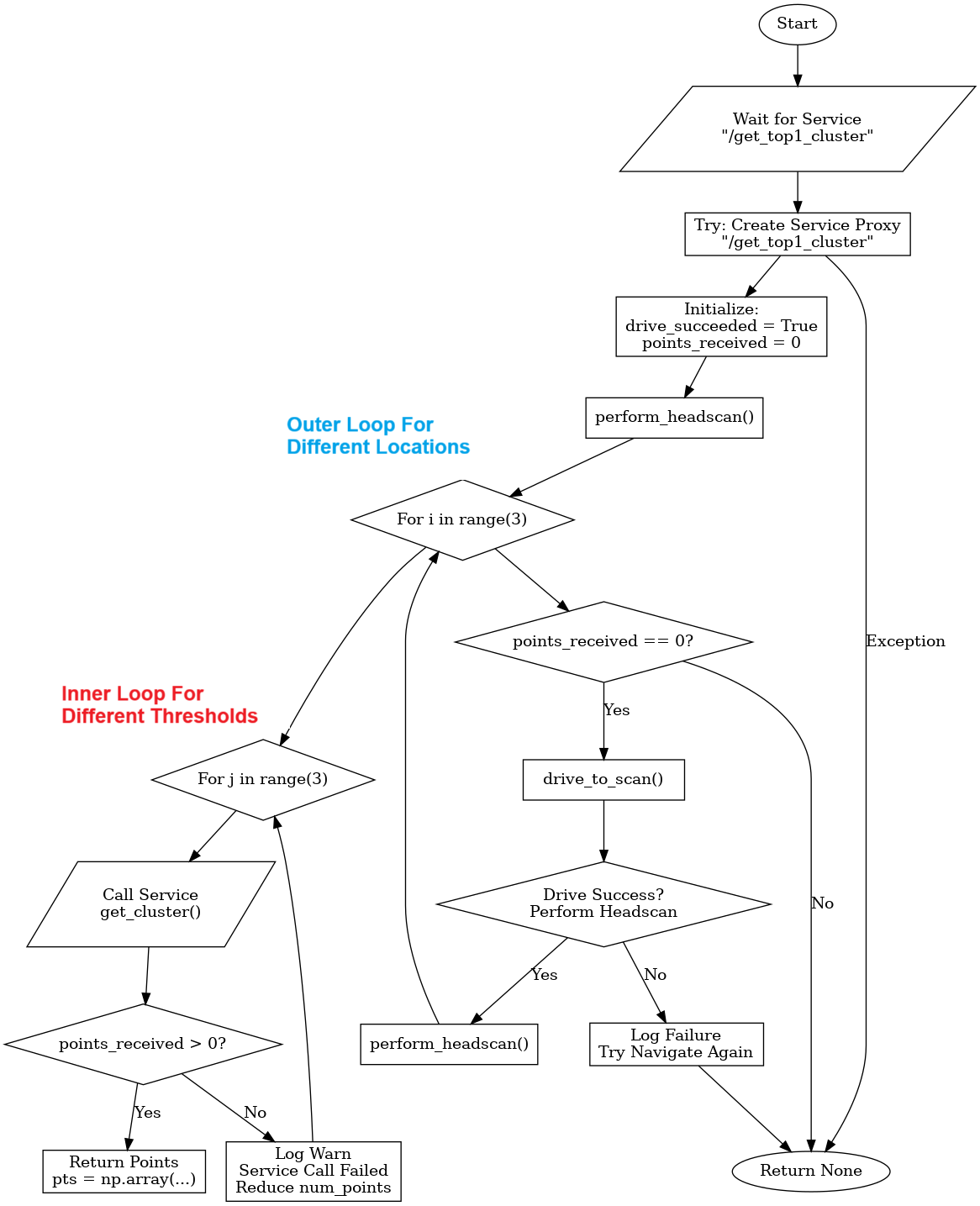
So I modified the code in such a way that the software performs:

* An initial complete head scan
* Attempt three different locations in case of no clusters found
* Attempt three different thresholds of “number of points” on each location, where if one attempt fails, it tries again with “half the previous threshold”

Below is the code after and was demonstrated in class which can be found in “poke\_khalil.py” of the Github repository:



Here is the flowchart that presents the above changes:

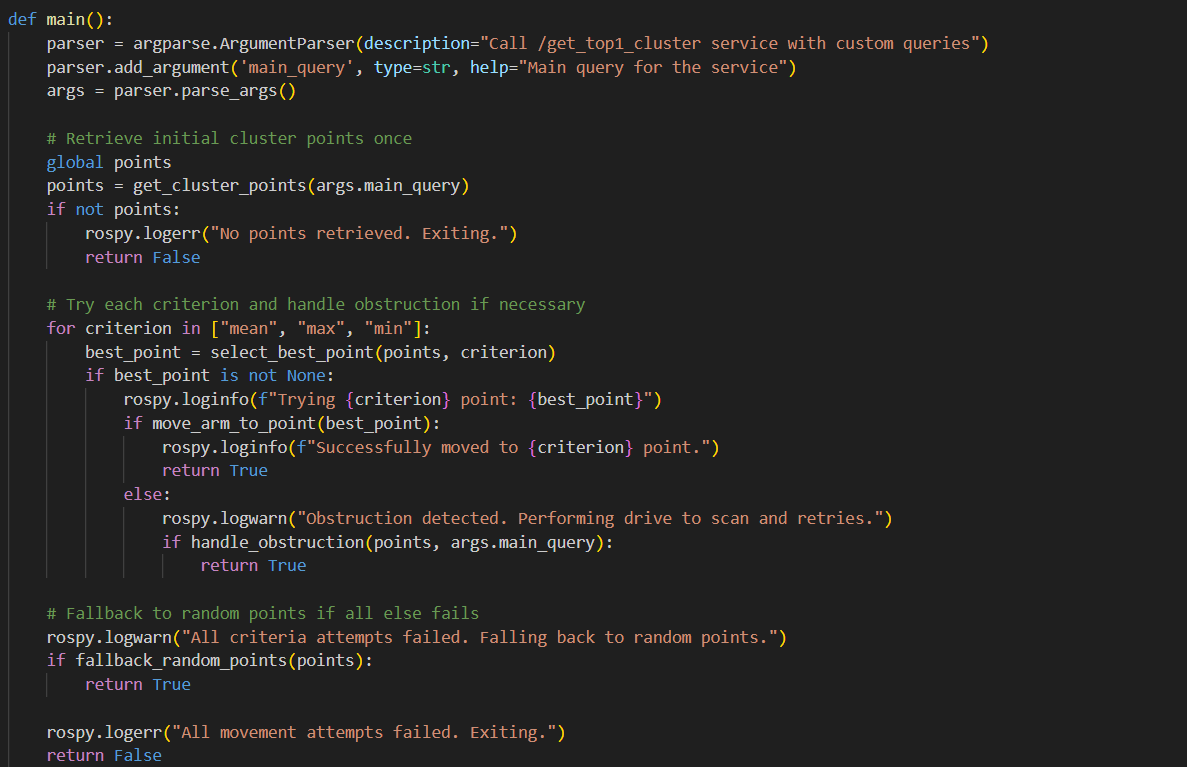


* 1. **Harsha:**

My primary focus in this project was to enhance the **robot’s navigation and object interaction capabilities** by integrating robust path planning, cluster re-evaluation, and manipulation strategies. Below are the key aspects of my contributions:

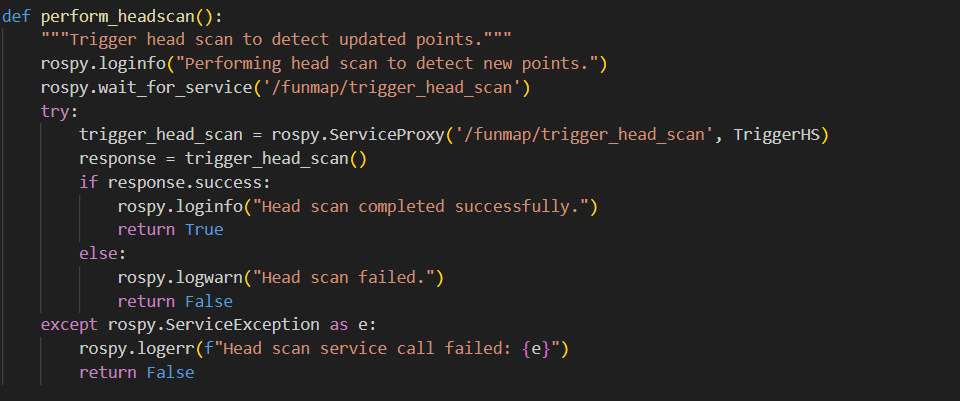
**Path Planning Logic Enhancements**:

* Improved the robot’s ability to calculate trajectories to detected clusters by implementing fallback strategies for handling navigation failures.
* Introduced dynamic error recovery by retrying with updated cluster data and adjusting target points.



**Head Scan Integration**:

* Added a mechanism to perform head scans (/funmap/trigger\_head\_scan) when navigation failures occur.
* TODO: Optimized the workflow to avoid redundant scans by checking the robot’s positional changes, ensuring head scans are only initiated when necessary.



**Cluster Navigation and Fallbacks**:

* Modified the poke.py base code to better handle cluster selection using criteria like mean, max, and min distances from the robot.
* Introduced a retry mechanism for failed navigation attempts, incorporating randomized fallback strategies to explore alternate cluster points effectively.

**Advanced Object Interaction**:

* Developed poke\_copy(Harsha).py as an enhanced version of poke.py to:
  + Seamlessly integrate navigation (drive\_to\_scan) with object interaction.
  + Add functionalities for grasping objects using the arm and gripper via precise joint control.
* Improved the interaction flow to handle obstructions by combining navigation recovery and cluster re-evaluation.

**Key Challenges and Resolutions**:

* Debugged critical service communication issues, ensuring smooth interactions with /get\_top1\_cluster and /funmap/move\_arm.
* Streamlined the integration of path planning, head scans, and cluster evaluation to maintain a consistent workflow across various failure scenarios.
  1. **Vishal:**

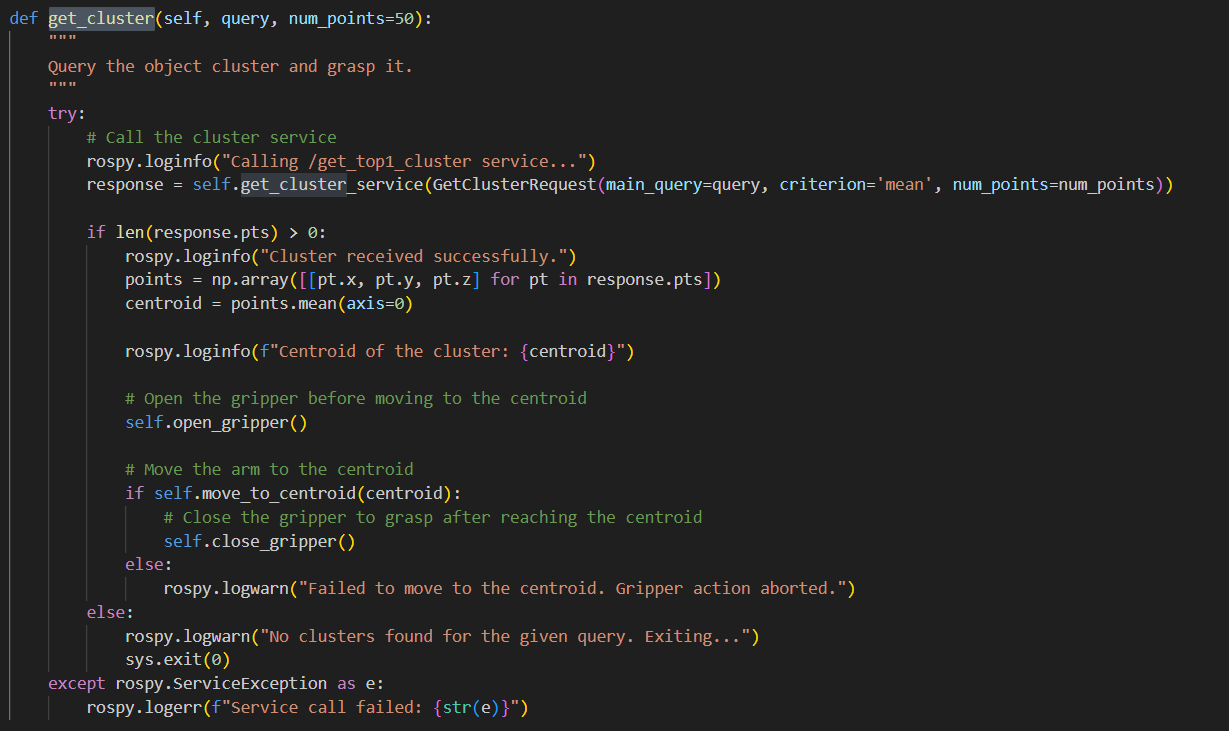
My python file implements a pipeline for autonomous robotic object grasping using ROS services. It focuses on object cluster detection, robotic arm manipulation, and gripper control to perform precise and effective grasping tasks.

### **Object Cluster Detection(get\_cluster)**

Identifying the target object cluster provides the information needed to compute the moving arm and grasping position.

**Functionalities**:

* Queries the /get\_top1\_cluster service with:
  + **main\_query**: Describes the object to grasp.
  + **criterion**: Defines how clusters are evaluated (e.g., by mean position).
  + **num\_points**: Specifies the number of points in the cluster.

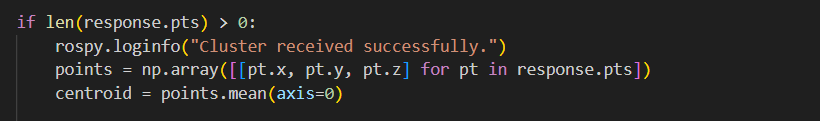


### **Centroid Calculation ( move\_to\_centroid)**

The centroid provides a target position for the arm to move towards during the grasping operation.

**Functionalities**:

* Processes the points received from /get\_top1\_cluster.
* Calculates the average position (centroid) of the cluster using NumPy:

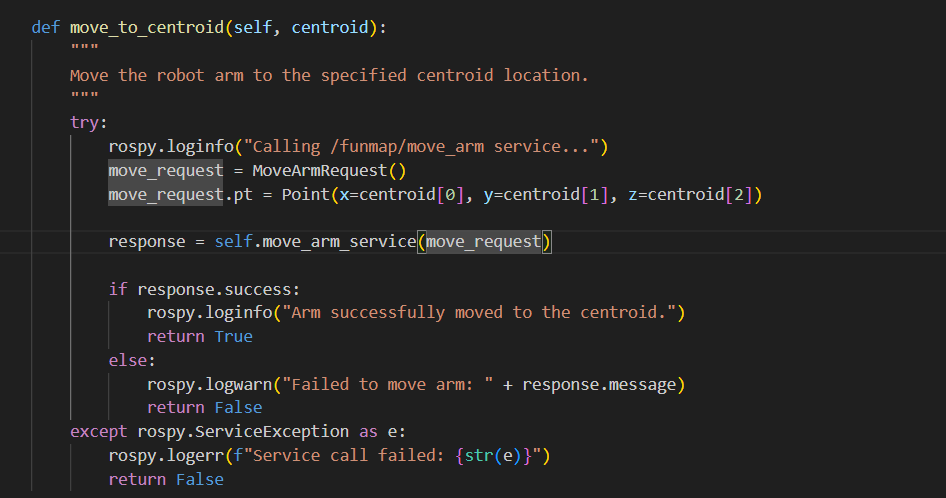


### **Arm Movement**

Moving the arm is critical to position it near the target object or to perform stowing tasks.

**Functionalities**:

* **Move to Centroid**:
  + Computes the centroid of a detected object cluster.
  + Uses the /funmap/move\_arm service to move the robot arm to the calculated 3D position.

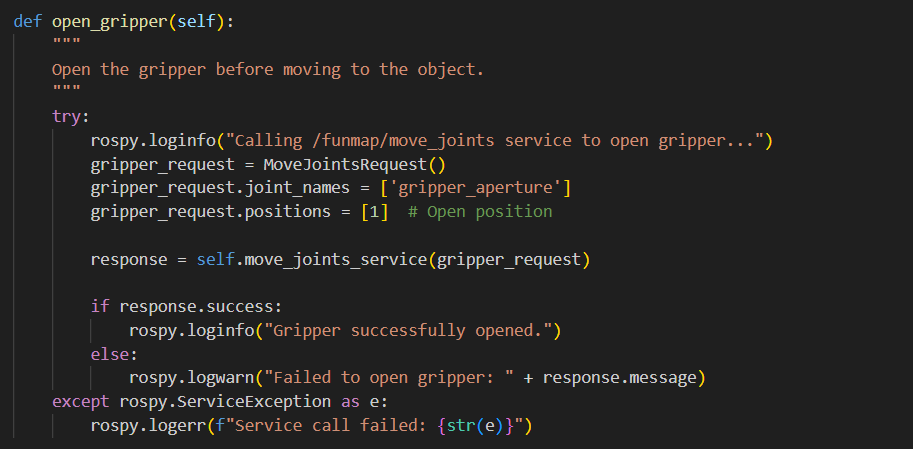


### **Gripper Control**

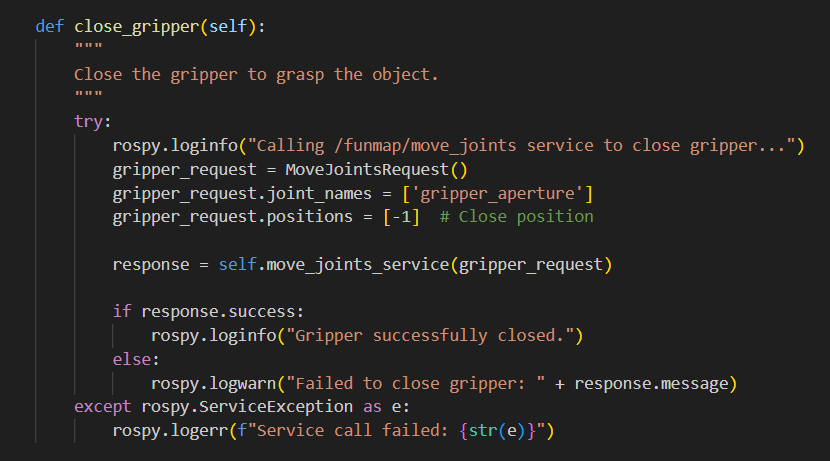
The gripper is essential for interacting with objects (grasping and releasing). Proper control ensures a successful grasping operation.

**Functionalities**:

* **Open Gripper**:
  + Uses the /funmap/move\_joints service.
  + Moves the gripper\_aperture joint to an open position (positions = [1]).



* **Close Gripper**:
  + Uses the /funmap/move\_joints service.
  + Moves the gripper\_aperture joint to a closed position (positions = [-1]).



**Key Challenges**:

* Critical service communication issues, ensuring smooth interactions with /get\_top1\_cluster , /funmap/move\_arm. and /funmap/move\_joints
* Invalid or noisy cluster points are returned by the service.
* Inconsistent arm calibration or incorrect position mapping.
* Gripper hardware issues, such as calibration errors.