A ROS-Based Modular Action Server for Efficient Motion Planning in Robotic Manipulators

Pedro A. Dias*† , João C. Souza* , Luís F. Rocha* , Daniel Figueiredo ‡, Manuel F. Silva*†
*INESC TEC - Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência,
Porto, Portugal
†Instituto Superior de Engenharia do Porto, Instituto Politécnico do Porto,
Porto, Portugal
‡ Palbit S.A., P.O. Box 4, Branca, Portugal
Albergaria-a-Velha, Portugal

***Abstract*—This paper discusses the emerging field of robotics, particularly focusing on motion planning for robotic manipulators. It highlights the need for simplification and standardization in robot implementation processes. Among several tools available, the paper focuses on the MoveIt tool due to its compatibility, popularity, and community contributions. However, the paper acknowledges some resistance in developing new applications with MoveIt, especially for researchers and beginners. To address this, the paper introduces an efficient, modular action server for interacting with the MoveIt framework. This pipeline simplifies parameter reconfiguration and provides a general solution for the motion planning problem. It can calculate trajectories for robotic manipulators without environmental collisions using a single server request and supports operation in different modes. The server was tested on an Universal Robots UR10 manipulator, demonstrating its ability to quickly plan paths for two test operations: an object pick-and-place mission and a collision avoidance test. The results were positive, achieving the set goals with minimal user-server interaction. This work represents a significant step towards more efficient and user-friendly robotic manipulation.**

***Index Terms*—Path Planning, Robotic Manipulators, MoveIt, Textual Configurable Pipeline Paradigm**

I. Introduction

It is widely recognized that robotic manipulators are transforming operations across various industrial sectors. The adoption of these technologies is driven by the potential benefits they offer, such as enhanced productivity and reduced operator risk. However, these systems operate on the basis of complex robotic concepts. In this context, motion planning is a critical aspect of robotics, applicable to a wide range of applications from picking up objects for aerospace construction factories [1] or in car painting in the automotive area [2]. Several solutions for the motion planning problem are currently available for use. One of them is the Open Motion Planning Library (OMPL), which includes several state-of-the-art sampling-based path planning algorithms [3]. However, OMPL does not contain the ability to detect collisions or for visualization, which arises from a deliberate design choice to avoid linking it to a specific collision checker or visualization front end. Contrasting with OMPL, there are some complete solutions, that already include extra functionalities apart from the planning. One such solution is Open Robotics Automation Virtual Environment (OpenRAVE), which provides a platform for studying, designing, and implementing motion planning algorithms in practical robotics scenarios [4]. Its focus is on simulating and verifying kinematic and geometric data relevant to motion planning. Another similar solution is Robotics Library (RL), a standalone C++ library focusing on robot kinematics, motion planning, and control [5]. It covers multiple areas, such as mathematics, kinematics and dynamics, hardware abstraction, motion planning, collision detection, and visual representation. MoveIt is another framework that isolates the user from the low-level code for planning, collision checking, and optimization problems. Although all the presented solutions can address most of the challenges researchers encounter, OpenRAVE and RL, despite being comprehensive solutions to control robotic manipulators, they do not receive as much community input as MoveIt.

Despite being a comprehensive solution, MoveIt is not a user-friendly framework, because of its integration with several planning, collision, and optimization libraries resulting in a large number of configurable parameters. These parameters often need to be adjusted multiple times in real or simulated scenarios, which is not a straightforward task. Many researchers use MoveIt for various movements with robotic manipulators, with or without environmental awareness, to avoid collisions. However, the high level of parametrization and the diversity of libraries used can pose challenges for researchers and beginners.

Considering this, the paper presents an efficient and modular pipeline for interacting with the MoveIt framework, allowing an easier parameter reconfiguration and a general pipeline solution for the motion planning problem, retrieving collision-free trajectories from a single action server request.

This work is co-financed by Component 5 - Capitalization and Business Innovation, integrated in the Resilience Dimension of the Recovery and Resilience Plan within the scope of the Recovery and Resilience Mechanism (MRR) of the European Union (EU), framed in the Next Generation EU, for the period 2021 - 2026, within project Hi_reV, with reference 64.

Copyright notice: 979-8-3503-7636-4/24/$31.00 ©2024 IEEE

Following this introduction, the article is structured as follows: Section II presents a review of some related works, and Section III describes the structure of the pipeline developed and its working principle. Section IV presents all the tests and consecutive results of the implementation of the pipeline in a real robot. Finally, Section V presents the main conclusions and key findings from this paper.

II. Related Work

In the topics of robotics and motion planning, significant advancements have been made to simplify the programming process and enhance the autonomy of robotic systems. A common thread among these advancements is the use of Robot Operating System (ROS)-based frameworks and integrated with MoveIt for motion planning.

Kingston and Kavraki [6] introduced Robowflex, a tool designed to streamline interactions with MoveIt for general motion planning applications. Robowflex serves as an Application Programming Interface (API) that facilitates the manipulation of robots, collision environments, planning requests, and motion planners. It offers the capability to develop self-contained scripts for evaluating motion planning algorithms, which is particularly beneficial for researchers and developers testing and comparing different motion planning algorithms in a controlled and isolated environment. Its main focus is to simplify certain steps in the robotic motion planning offered by MoveIt, allowing it to describe several lines of code in a single command. Furthermore, Robowflex's compatibility with other libraries like OMPL, DART, and ROS Industrial Tesseract enhances its versatility, making it a valuable tool that can be integrated into various robotics projects. This library was successfully implemented in the NASA Robonaut 2.

Building on this foundation, Villagrossi *et al.* [7] presented a comprehensive ROS-based framework that integrates various functionalities crucial to robotics. This framework allows for the inclusion of functionalities to control various robotic arms, thereby providing a high level of autonomy in the execution of different tasks. The framework's ability to autonomously process manipulation actions, such as picking and placing objects, and managing the kinematics model of the robotic system, represents a significant advancement in the field. Notably, the framework also embeds motion planning functionalities, provided by MoveIt, for generating collision-free trajectories, by querying a planning scene that can be dynamically updated and connected to a perception system, a feature that resonates with the capabilities of Robowflex.

Faroni *et al.* [8] introduced a novel approach to human-robot collaboration tasks. They proposed a two-layered control approach divided into task planning and action planning layers. Each layer operates at a specific level of abstraction, with the task planning layer considering high-level operations without taking into account movement properties, and the action planning layer optimizing the execution of high-level operations based on human state and geometric reasoning. This layered approach echoes the structure of both Robowflex and Villagrossi *et al*.'s framework, where high-level operations are optimized based on specific parameters.

The works presented focus on creating frameworks that include MoveIt, or simplifying certain aspects of it, while still maintaining dependencies with other functionalities. However, not all of these solutions allow for the easy configuration of the parameters associated with MoveIt. In contrast, the solution proposed in this text aims to isolate MoveIt from other functionalities, thus removing the need for users to manipulate specific parameters in the code, encapsulating the whole task of motion planning, besides just simplifying the code writing. This is achieved by simplifying the configuration through a YAML Ain't Markup Language (YAML) file, which can be modified without the need to recompile the project. The goal of this solution is to provide a high-level interaction between the user and the robot manipulator.

III. Implementation

The ROS1-based action server [9] described in this paper is based on a modular textual configurable pipeline paradigm structure which identifies different heuristics, in a YAML file.

These heuristics define the processing workingflow of the system besides configurations parameters according to specific demands. Two other works were developed using this modular structure, implemented in this paper, being one for estimating grasp positions for several objects in picking tasks in aerospace construction factories [1] and other for bin-picking for ship-building logistics [10].

The developed pipeline aims to simplify the interaction of researchers and enthusiasts with the MoveIt tool. This tool was designed to meet the following requirements:

- Simplified parameter reconfiguration: This feature is crucial for quickly testing new configurations and adapting to new implementation scenarios.
- General use pipeline: This involves creating a pipeline composed of different groups of metrics for abstraction and adaptation to a wide variety of scenarios imposed by different applications.
- Reduced user interaction: The server allows the planning and movement of a robotic manipulator with minimal user interaction.

In a typical interaction with the pipeline, which structure is illustrated in Figure 1, users can choose to include a perception of the robot's surroundings in the planning. This is achieved by activating and defining the topic in the YAML configuration file where the point cloud is published. Once this is done, the pipeline processes the point cloud until it is included as a collision object in the planning scene. The processing of the point cloud involves several heuristics (highlighted in yellow). It begins with *PointCloudRetrieval*, which receives the point cloud from the topic. This is followed by *PointCloudDownsamplingFilter*, which reduces the number of points in the point cloud. Next, *PointCloudOutliersFilter* removes points from the point cloud that are distant from zones with a higher concentration of points. *PointCloudCentroid* then finds the centroid of the point cloud and calculates a set of shifted

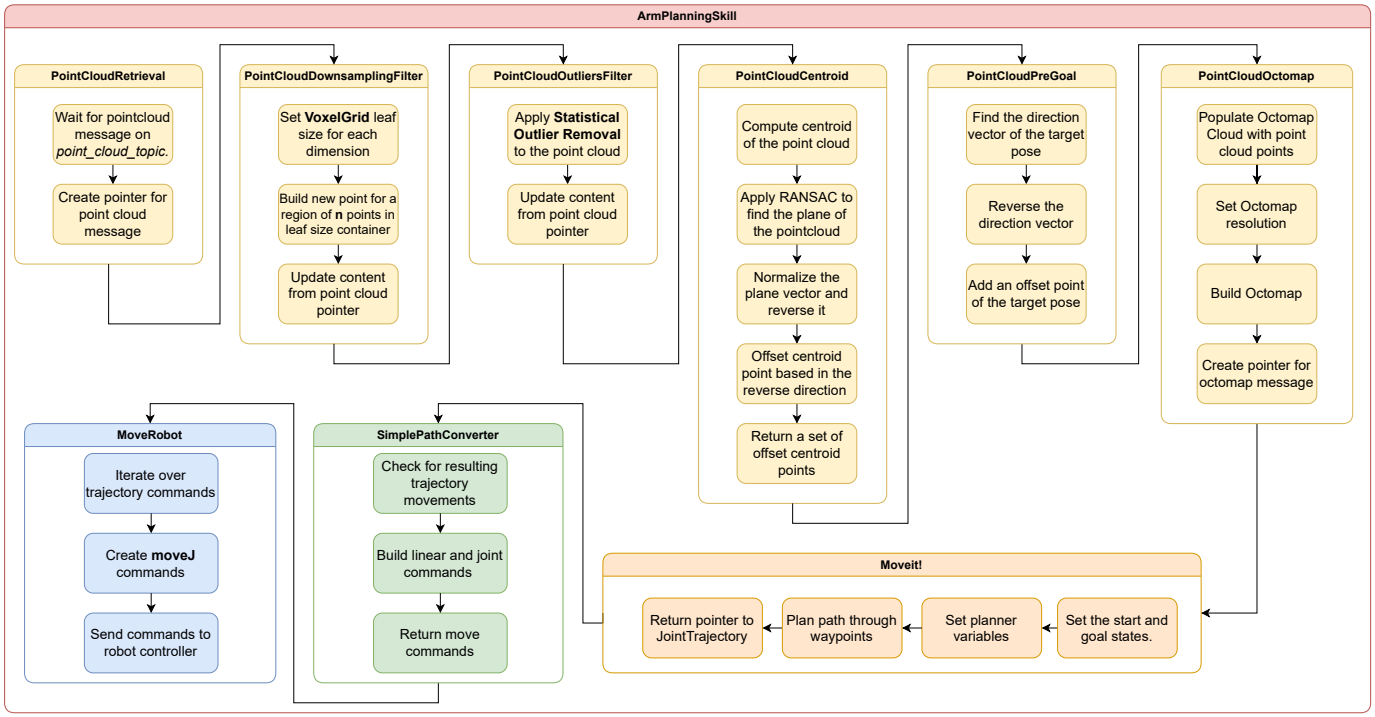


Fig. 1: Pipeline block structure of the action server.

centroid points to guarantee a linear trajectory when planning in constrained workspaces. Optionally, the user can choose to calculate a pregoal point using the *PointCloudPregoal*, which is a shifted goal point, to guarantee the right positioning of the end effector for grasping/placing tasks. Finally, *PointCloudOctomap* converts the point cloud into an octomap, *i.e.* a collision object representing the point cloud. Once the point cloud has been processed, the server moves on to the planning phase. This involves the MoveIt framework (highlighted in orange), which plans a path for the manipulator that avoids collisions, in joint space. If the planning is successful, the path is converted into commands compatible with the robot controller and also a format that can be interpreted by another action server responsible for moving the robot manipulator. These commands are also published in ROS parameter server for future use. This is done by the *SimplePathConverter* heuristic (highlighted in green). Suppose the action server responsible for controlling the robot manipulator is unavailable. In that case, the *MoveRobot* heuristic (highlighted in blue) sends movement commands directly to the robot controller, for execution of the planned movement. All the parameters associated with the heuristics explained, can be modified before launching the action server. By disassociating these parameters from their local definition in the source code, users can change the configuration parameters without re-compiling the project, making execution and testing more efficient.

The user can send a request to the pipeline, asking for one of two modes:

- Mode No-PointCloud (NPC): This mode allows the user to plan paths without considering the surrounding environment, skipping the yellow group in Figure 1. This mode will only take into consideration the robot Unified Robot Description Format (URDF), guaranteeing no self-collision.
- Mode With-PointCloud (WPC): This mode is responsible for processing the surrounding environment, creating a collision map for the path planning phase. This mode will consider all the heuristics in the pipeline.

For a more simplified approach, this server loads up into the parameter server an YAML file, which describes common movements in a Transform Frame (TF) format, like the one described in Listing 1.

```
Move_01:
      {BaseFrameIdTF: reference_frame,
       ToolFrameIdTF: child_frame,
       Goal: [x, y, z, qx, qy, qz, qw]}
```

Listing 1: Example of move types used in the pipeline

All TFs are either directly retrieved from the TF tree by the user or dynamically supplied by another action server, depending on the specific objective, like for grasping and placing tasks. For example, a static position is obtained by moving the robot to the desired location and then reading the TF between two arbitrary links, namely the reference frame and child frame. To ensure the server can interpret any TF between any two links, each server request triggers an analysis of these movements. If the reference frame differs from the *base link* or the child frame deviates from *tool0*, they are converted to these standard values. This ensures that the movement executed on the actual robot aligns with

expectations. To modify the reference base of the TF, it's essential to multiply the TF between the desired base and the current base with the existing TF. This process is outlined in Equation 1:

$$^{A}\mathbf{T}_{C} = ^{A}\mathbf{T}_{B} * ^{B}\mathbf{T}_{C} \quad (1)$$

This allows the user to reference different positions to different frames depending on the application, without worrying about if it is the correct frame or not. This conversion is necessary since MoveIt uses one specific frame for planning, which is normally the top frame in the URDF file. Therefore, it is mandatory that all target points are referenced to the same base frame.

IV. Results

This section explores the testing of the pipeline in a simulated version of the robot, the precision acquired by the pipeline compared to the real robot controller, and the implementation in the real robot in the execution of different missions. For reference, this pipeline was tested on a mobile manipulator, with a coupled UR10 from Universal Robots, a 2F-140 gripper from Robotiq, and a Phoxi 3D scanner.

A. Pipeline testing in simulation scenario

Before testing in a real robot, the pipeline was tested using Rviz [11] and several simulation scenarios, to guarantee a safe transition to the real robot. In an initial test, a purple object was created. For the planning phase, two points were selected: a starting state and a goal state, indicated by a red and yellow arrow, respectively, in Figure 2. These points were determined by a linear translation along the *x*-axis, with slight variations in orientation. In this case, the path can't be achieved by linearly moving the manipulator's end effector, and to reach the goal state, MoveIt will have to plan a trajectory that avoids a direct collision with the obstacle. Figure 2 shows the result of this planning, described by a green line.

A second test, illustrated in Figure 3, included the perception of the environment. Two points were chosen for the planning, as depicted in Figure 3: the first point of interest is the current reference point of the point cloud. This is the position at which the octomap, *i.e.* the collision object representing the point cloud, would be calculated and established; the second point is chosen such that it would be obscured by the octomap itself. This implies that for collision-free planning, a deviation of the arm would be necessary.

As shown in Figure 3, the skill can plan a collision-free trajectory between the two points.

B. Pipeline vs real controller

A key aspect of validating the proposed solution is assessing the errors between two known positions in space reached by the robot, whether through its controller or planning algorithm. This evaluation is crucial to ensure that the planned movements have adequate precision for executing tasks that require high accuracy. Two arbitrary positions were compared: Position A, and Position B, respectively illustrated in Figure 4b and in Figure 4c. For reference, the robot's home position is illustrated in Figure 4a.

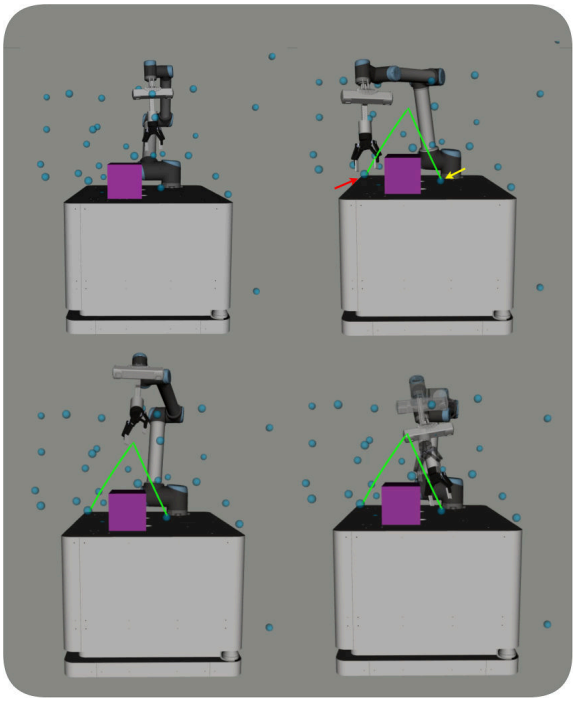


Fig. 2: Path planning for a robotic manipulator between two points with one obstacle in between.

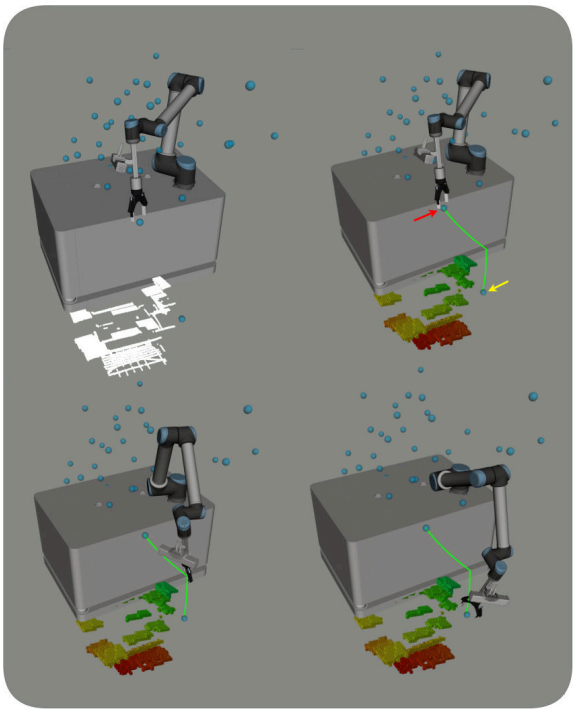


Fig. 3: Path planning for a robotic manipulator between two points with perception of the environment.

Once the planning has been completed using both methods, it is then necessary to extract the robot's final poses, through

(a) Robot in home position. (b) Robot in position A. (c) Robot in position B.

Fig. 4: Robot positions for precision comparison.

the TF tree in ROS, when the movements are executed by the controller and the planning pipeline. To accomplish this, the TF between two links, namely *base_link* and *tool0*, *i.e.* robot's base and its phalange, respectively, was extracted. The choice of these two links is completely arbitrary. The only consideration when choosing the links is to ensure that, at least, one of them is fixed and that both belong to the robot's kinematic chain. Table I shows the robot's translation and rotation values for the different positions.

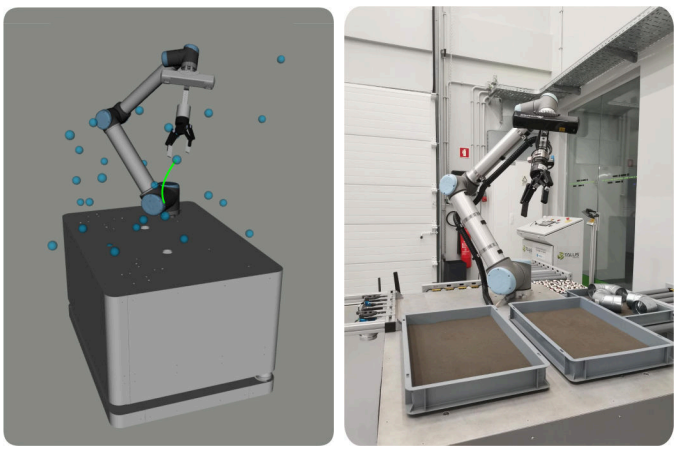
TABLE I: Comparison between skill planning and robot controller for known poses.

Poses	Translation (m)			Euler (rad)		
	x	y	z	roll	pitch	yaw
Controller (A)	-0.265	-0.177	0.567	3.061	-0.090	-0.044
Pipeline (A)	-0.265	-0.177	0.568	3.055	-0.084	-0.048
Controller (B)	-0.425	0.479	0.614	3.097	0.013	1.499
Pipeline (B)	-0.428	0.481	0.606	3.099	0.021	1.505

In Table I, the thresholds for significant differences are set at 3 millimeters for translation and 0.005 radians (0.72 degrees) for orientation. The translation pair for Position B is highlighted in red, indicating a difference of approximately 8 millimeters in the Z-axis translation value between the controller and the pipeline. For the positions' orientation, pairs with differences greater than the 0.005 radians threshold are highlighted in orange. The two highlighted orientation pairs are for Position A (Pos. A) in the *roll* orientation, showing a difference of 0.006 radians (0.34 degrees), and for Position B (Pos. B) in the *pitch* orientation, with a difference of 0.008 radians (0.46 degrees). It is possible to assume that the positions achieved by the planning pipeline differ in max 8 millimeters, being an average error, of less than 3 millimeters for pose translation and a pose rotation error less than 0.72 degrees, being 0.46 degrees the max detected. This accuracy is only possible thanks to the calibration of the URDF models, using a script provided by Universal Robots [1].

C. Skill implementation in real robot

After the simulated tests, the proposed system was deployed in the real robot. Therefore, several tasks were performed to check the capability of the system. The first real test consisted of moving the robot manipulator, between two pre-defined points. The execution of the trajectory in the simulated robot is illustrated in Figure 5a. The execution in the real robot can also be seen in Figure 5b.



(a) Trajectory execution on the simulated robot. (b) Trajectory execution on the real robot.

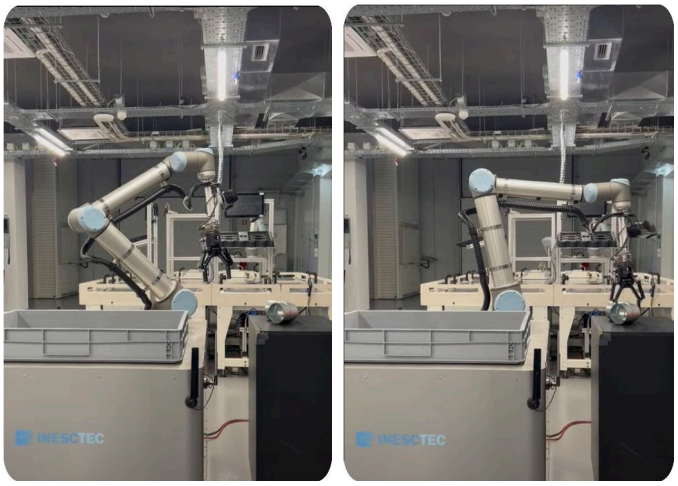
Fig. 5: First test on the real robot.

[1]More information at Universal Robots Calibration.

After testing various movements with the robot and proving the high precision of the developed system, it was possible to carry out more complex tasks. To do this, two different tasks were considered: one was to pick up and place an object and the other was to avoid an obstacle barrier.

The initial task involved positioning the robotic manipulator to perform a scan. After this, the point cloud retrieved, was used to identify the part using 3D perception and segmentation, and to determine a grasp position with a separate action server. Finally the robot manipulator could perform the task of pick and place, considering all the intermediate movements to reach the goal position.

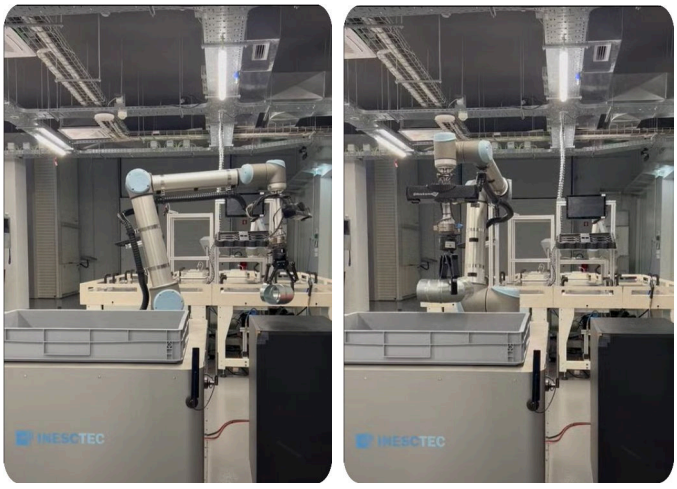
In the past, these intermediate movements were carried out directly with the controller, without taking the robot cell into account for self-collision. In these tests, all the movements executed by the robot were planned by the planning skill. Figure 6 shows the movements performed by the real robot when it scans (Figure 6a) and picks up the detected object (Figure 6b). The mission continues with the lifting movements of the object (Figure 7a) and the return to the starting position (Figure 7b).



(a) Robot scanning the object. (b) Robot picking the object.

Fig. 6: Robot executing scan and pick of an object.

A second test with the real robot was carried out to check the pipeline ability to plan paths for the robotic manipulator, considering the surrounding environment. In this test, a barrier was built using cardboard boxes between the robot's scanning pose and its final pose. This barrier prevented linear movement between the two points, so it was necessary to plan either above the barrier or around it. The constructed barrier is similar to the one illustrated in Figure 8a. The position of the robot's gripper in relation to the barrier is illustrated in Figure 8b, to prove that it is close, but not colliding, with it. The final result is illustrated in Figure 9, where the robot successfully plans a trajectory between its current pose and the final pose along the side of the barrier.



(a) Robot lifting the object. (b) Robot placing the object.

Fig. 7: Robot executing lift and place of an object.

(a) Barrier used for collision avoidance test. (b) Distance between gripper and barrier.

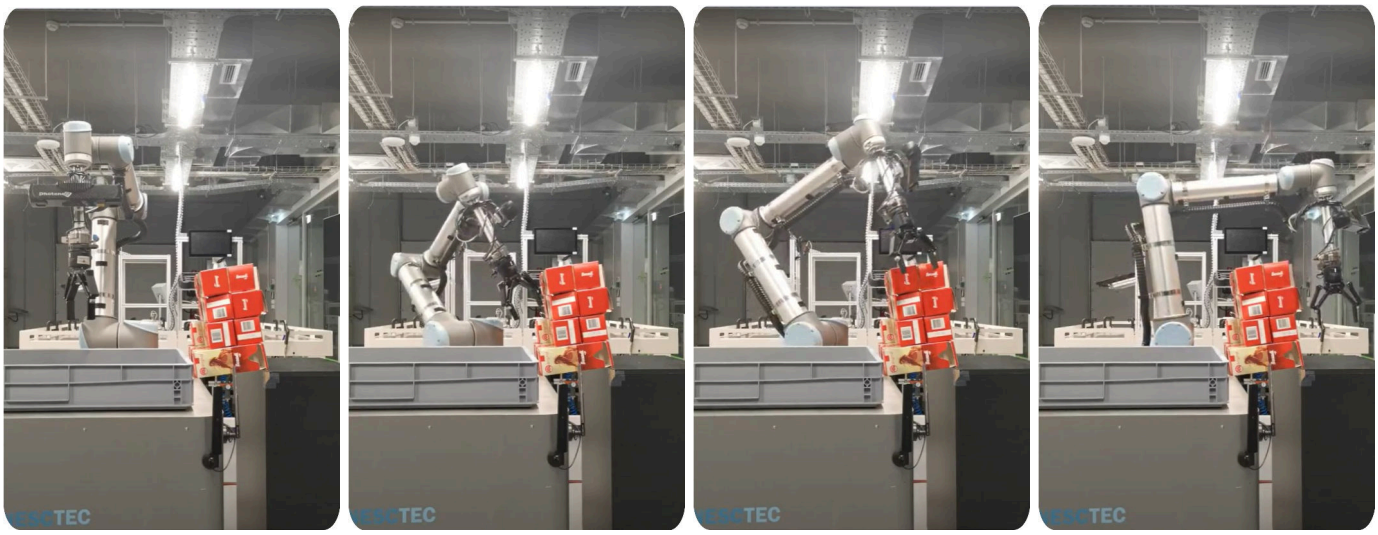
Fig. 8: Barrier setup and distance between gripper and barrier.

V. Conclusion

This paper addresses the escalating interest among researchers in simplifying the process of motion planning for robotic manipulators. While several tools have been developed to this end, they often lack intuitive usability.

In this paper was presented a streamlined approach to path planning for robotic manipulators, offering a solution for researchers and enthusiasts alike. The developed pipeline handles the majority of complex tasks associated with path planning, freeing users to concentrate on other aspects. The results demonstrate the feasibility of maneuvering the robotic manipulator using a simple server request, applicable across a range of scenarios. These include pick-and-place tasks for various objects and obstacle avoidance.

Looking ahead, it will be necessary to explore additional libraries, such as the RL and OpenRAVE mentioned in this paper. The goal is to develop a pipeline compatible with all



(a) Robot in home position. (b) Robot scanning the barrier. (c) Robot on the side of the barrier. (d) Robot reaching the goal.

Fig. 9: Robot executing a path avoiding collision with the barrier illustrated in Figure 8a.

these libraries, enabling users to select the one that best suits their needs. Furthermore, it would be beneficial to execute benchmarks on each of the paths returned by the planners. This would allow for a comparison and selection of the optimal path. This paper thus serves as a stepping stone towards more efficient and user-friendly robotic manipulation.

REFERENCES

[1] J. P. Carvalho de Souza, C. M. Costa, L. F. Rocha, R. Arrais, A. P. Moreira, E. S. Pires, and J. Boaventura-Cunha, "Reconfigurable grasp planning pipeline with grasp synthesis and selection applied to picking operations in aerospace factories," *Robotics and Computer-Integrated Manufacturing*, vol. 67, p. 102032, Feb. 2021. [Online]. Available: http://dx.doi.org/10.1016/j.rcim.2020.102032

[2] H. N. Ghafil and K. Jármai, "Research and application of industrial robot manipulators in vehicle and automotive engineering, a survey," in *Lecture Notes in Mechanical Engineering*, ser. Lecture notes in mechanical engineering. Cham: Springer International Publishing, 2018, pp. 611–623.

[3] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, https://ompl.kavrakilab.org.

[4] R. Diankov and J. Kuffner, "Openrave: A planning architecture for autonomous robotics," Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34, July 2008.

[5] M. Rickert and A. Gaschler, "Robotics library: An object-oriented approach to robot applications," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 733–740.

[6] Z. Kingston and L. E. Kavraki, "Robowflex: Robot motion planning with moveit made easy," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 3108–3114.

[7] E. Villagrossi, N. Pedrocchi, and M. Beschi, "Simplify the robot programming through an action-and-skill manipulation framework," in *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2021, pp. 1–6.

[8] M. Faroni, M. Beschi, S. Ghidini, N. Pedrocchi, A. Umbrico, A. Orlandini, and A. Cesta, "A layered control approach to human-aware task and motion planning for human-robot collaboration," in *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, 2020, pp. 1204–1210.

[9] Robotic Operating System, "actionlib - ROS Wiki — wiki.ros.org," https://wiki.ros.org/actionlib, [Accessed 16-07-2024].

[10] A. Cordeiro, J. P. Souza, C. M. Costa, V. Filipe, L. F. Rocha, and M. F. Silva, "Bin picking for ship-building logistics using perception and grasping systems," *Robotics*, vol. 12, no. 1, p. 15, Jan. 2023. [Online]. Available: http://dx.doi.org/10.3390/robotics12010015

[11] Robotic Operating System, "rviz - ROS Wiki — wiki.ros.org," https://wiki.ros.org/rviz, [Accessed 16-07-2024].