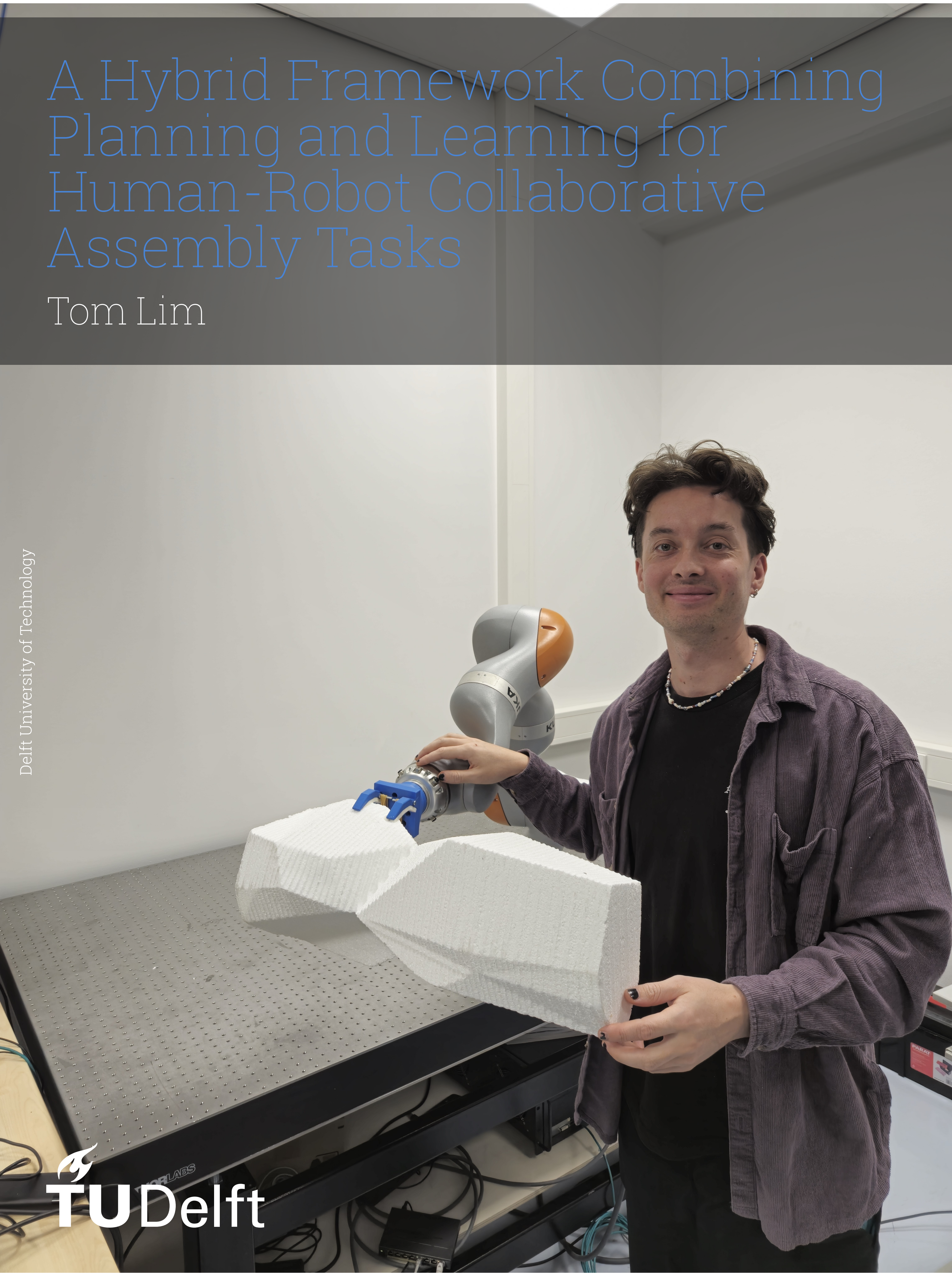
A Hybrid Framework Combining Planning and Learning for Human-Robot Collaborative Assembly Tasks

Tom Lim

Delft University of Technology



A Hybrid Framework Combining Planning and Learning for Human-Robot Collaborative Assembly Tasks

by

Tom Lim

to obtain the degree of
Master of Science

At the:
Department of Cognitive Robotics
Deft University of Technology
to be defended publicly on Wednesday November 20, 2024 at 15:00 PM.

Student number:	4442520	
Project Duration:	May, 2024 - November, 2024	
Supervisors and Assessment committee:	Dr. L. Peternel,	TU Delft, supervisor, chair
	Dr. J. Alonso-Mora,	TU Delft, member
	ir. A. Hidding,	TU Delft, supervisor

Preface

This thesis marks the conclusion of my Master's in Robotics at Delft University of Technology. Over the past seven months, I have developed a hybrid framework combining planning and learning for human-robot collaborative assembly tasks, exploring the potential of combining state-of-the-art motion planning and learning-based trajectory generation.

I would like to express my sincere gratitude to my supervisors, Luka Peternel and Arwin Hidding, for their professional guidance and invaluable support throughout the course of this project. Their expertise, constructive feedback, and encouragement were instrumental in refining my work and keeping me motivated.

A special thanks goes to my girlfriend, who, despite having no personal interest in robotics, has always shared in my excitement. Her enthusiasm for my work, even when she didn't have to, has meant the world to me. I am also deeply grateful to my parents for their unconditional support and patience throughout my (quite long) academic journey. Their belief in me and consistent encouragement have been a cornerstone of my success.

Tom Lim
Delft, November 2024

Contents

A Hybrid Framework Combining Planning and Learning for Human-Robot Collaborative Assembly Tasks

Tom Lim[1]
Supervised by:
Arwin Hidding[2], and Luka Peternel[1]

***Abstract*—This paper proposes a novel framework that combines both planning and learning-based trajectory generation methods to handle complex robotic assembly tasks. The framework utilizes MoveIt! for planning large-scale reaching motions and Dynamic Movement Primitives (DMPs) for precise grasping and placing movements, with both methods integrated into a single system controlled by a behavior tree. An impedance controller is employed to ensure smooth and safe execution of the generated trajectories, particularly in scenarios that involve human interaction.**

The proposed framework was evaluated within the context of the European Space Agency-funded Rhizome project, which focuses on off-earth habitat construction. The project involves assembling habitats using custom-designed Voronoi-shaped building blocks, which were also utilized in experiments to test the framework. The results showed that combining planning for large-reaching motions with DMPs for detailed movements effectively addressed the limitations of each individual method, delivering a flexible and robust solution to the challenges of robotic assembly.

I. Introduction

Collaborative robots are the result of significant advancements in robotics. Positioned at the forefront of the fourth industrial revolution [1], these robots enable companies to integrate the strength and precision of robots with the dexterity and decision-making capabilities of humans [2]. This collaboration allows robots to handle complex tasks that are challenging to fully automate and assists humans in performing physically demanding or tedious repetitive tasks.

A major advantage of collaborative systems is their flexibility; human-robot collaborations are designed such that they can share a workspace without the need for rigid safety systems. This flexibility allows for easier and quicker re-allocation of robots within production plants. Consequently, a single robot can perform a variety of tasks [3], making robotic automation accessible and cost-effective for smaller companies.

Assembly tasks present one of the most challenging areas to automate, requiring consideration of both position trajectories and task dynamics [4]. This complexity makes assembly a prime candidate for human-robot collaboration [3], [5], [6], resulting in various implemented methods. The main difference between these methods lies in the trajectory generation.

For example, trajectories can be generated using planning algorithms. In [7], the authors employ computer vision to locate a part and plan a trajectory towards it, positioning the end-effector near the grasping location. Once positioned, the human physically guides the more intricate grasping action. Similarly, authors of [8] use a tablet to indicate the location of a part after which the robot plans a trajectory towards the specified part.

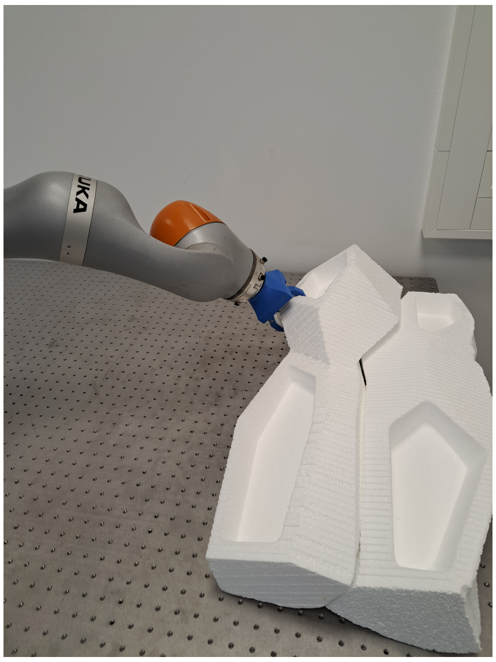


Fig. 1. Photo of the robot executing the assembly task using a DMP-based approach. The setup is part of the Rhizome 2.0 project, focused on off-earth habitat construction. The task involves grasping and placing custom-designed, Voronoi-shaped building blocks, which are used to simulate components for modular habitat structures.

Such planning-based approaches can be effectively achieved using state-of-the-art open-source robotics manipulation platforms, like MoveIt! [9], which generates high-degree of freedom trajectories through cluttered environments while avoiding local minimums.

Another widely used approach in robotic assembly is learning from demonstration (LfD). In robotics, LfD is a method where robots learn new skills by imitating the actions of an expert [10]–[14]. A key advantage of LfD is that it makes robot programming accessible to nonexperts. Through demonstrations, robots can learn the constraints and requirements of a task, enabling them to adapt their behavior. This means robots are not limited to repeating predefined actions in controlled settings, but can learn to make optimal decisions in more

[1]Cognitive Robotics, Faculty of Mechanical Engineering, Delft University of Technology, The Netherlands.
[2]Robotic Building, Faculty of Architecture, Delft University of Technology, The Netherlands.

complex environments. As a result, LfD has the potential to bring significant benefits to industries like manufacturing [15].

Learning trajectories for tasks such as robotic assembly is often done via the use of dynamical motion primitives (DMPs) [14]. DMPs, first stated in [16], represent an elegant mathematical formulation of the motor primitives as stable dynamical systems and are well suited to generate motor commands for artificial systems like robots [4]. This process usually involves a human that demonstrates a movement, after which a DMP can be fitted to reproduce the demonstration.

For example, authors of [17] utilize kinesthetic guiding to demonstrate trajectories for the Cranfield assembly benchmark [18], after which they encode the position and orientation trajectories as DMPs. In another example, authors of [19] recognize that assembly tasks often fail due to unforeseen situations. In order to resolve this issue they propose a LfD framework which models exception strategies as DMPs.

Complementary to assembly tasks, disassembly is also challenging by solely using the demonstrated trajectories [4]. Classical DMPs repel the idea of reversibility because they have a unique point attractor in the specified goal parameter of the movement. In order to tackle the disassembly challenge, authors of [20] propose a method that learns two DMPs from a single demonstration; one forward and one backward.

Both planning and Learning from Demonstration (LfD) frameworks have their own advantages. Planning approaches, which rely on a predefined goal position, are only as effective as the perception system providing the goal [14]. The inherent complexity of assembly tasks makes it challenging to fully automate these processes using planning methods alone. Consequently, as shown in the literature, planning frameworks are often employed to position the end-effector near a target part, as specifying such a goal location is relatively straightforward. Planning methods are preferred where applicable because they generate optimal trajectories, including the effective control of the arm's null-space.

Conversely, LfD methods, particularly DMPs, excel in situations requiring small, precise movements that are difficult to define through coding. Their strength lies in the ability to replicate demonstrated movements, eliminating the need for sophisticated perception systems or complex programming. Because demonstrations typically involve recording end-effector data, they have proven to be particularly well suited for overactuated systems, such as redundant manipulators, for which kinematic feasibility is relatively easier to achieve [14]. It is worth noting that there are also approaches that do teach null-space motion [21], however, these approaches require additional steps beyond the classical DMP framework.

So far we have only discussed trajectory generation methods used in assembly tasks. However, the execution method of such trajectories is just as important to consider. Classically trajectories were executed using (possibly dangerous) position-controlled rigid robots [22]. However, in the context of human-robot interaction such methods are inadequate because the unavoidable modeling errors and uncertainties may cause a rise of the contact force, ultimately leading to an unstable behavior during the interaction, especially in the presence of rigid environments [23].

In the context of safe human-robot interaction, impedance and the related admittance control, defined by [24], form a paradigm to treat robotic systems from an energetic point of view such that motion and force can be controlled in a unified manner [22]. The impedance has flow (i.e., motion) input and effort (i.e., force) output, while admittance is the opposite, having effort input and flow output. This means that in a physical interaction, one must physically complement the other. It means if one system is regarded as admittance, the other must be treated as impedance and vice versa [25].

Especially impedance control has proven effective for manipulation tasks. The reason being that the fact that the environment can always accept force input, but sometimes cannot be moved, admittance is a proper role for environment. Based on the complementary theory, the manipulator should be regarded as impedance, allowing it to safely interact with its environment (i.e. human worker).

To summarize, it can thus be concluded that planning-based approaches generally excel in scenarios requiring simpler movements, whereas learning-based approaches are more effective for complex movements. Given that assembly tasks often require both types of movements, it is beneficial to consider integrating both planning and learning into a single framework. However, based on the extensive DMP survey [4], such a combination has not yet been implemented.

This paper proposes a novel assembly framework that incorporates both planning and learning trajectory generation methods in combination with an impedance controller. The hypothesis is that combining trajectory planning for large reaching movements with trajectory learning for precise, small-scale movements will result in an effective assembly framework. This proposed framework will be tested within the context of the Rhizome 2.0 project[1], which aims to provide a proof of concept for assembling habitats in empty lava tubes on Mars using Voronoi-shaped building blocks.

The following sections describe the methods used in the proposed framework. After which, the experiments conducted to test the framework are described in the experiments section. Finally, the results are analyzed and discussed in the discussion section.

II. Methods

The aim of this paper is to integrate planning and learning-based trajectory generation into a unified assembly framework. Figure 2 provides a system overview with the main software blocks, signals and apparatus. Assembly tasks can usually be broken down into two types of movements: the first is moving the parts between grasping and assembly locations and the second is the more precise grasping and placing of the parts themselves. As described in section I, a planning approach is a suitable option for the first type of movement and a learning approach suits the second type of movement. For the second, more precise sub-task, a learning approach based on Dynamic Movement Primitives (DMPs) is proposed. Both methods are implemented as ROS packages allowing smooth integration within a single framework.

[1]http://cs.roboticbuilding.eu/index.php/Rhizome2

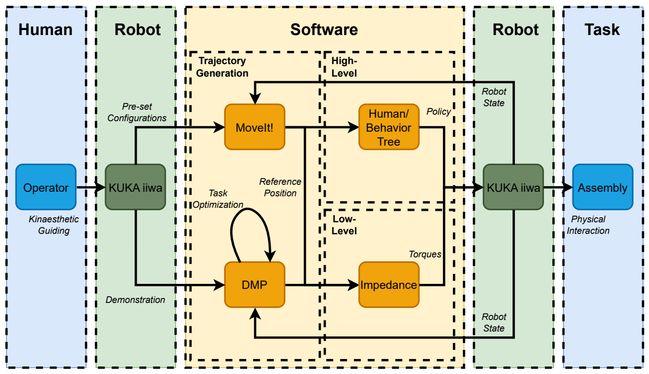


Fig. 2. System overview depicting core software blocks, signals and apparatus. The human operator (blue) interacts with the robot (green) to provide it with both a set of predefined configurations as well as a demonstration that is used to create the initial DMP. After the initial setup, the software components (yellow) provide the robot with commanded torques in a feedback loop. The behavior tree component manages the use of the MoveIt! and DMP trajectory.

To manage the assembly task at a high level a behavior tree is utilized. Behavior trees provide a method for describing a policy for an agent such as a robot. In this context, the behavior tree mainly coordinates when to use MoveIt! for the larger movements and when to switch to DMPs for the finer grasping and placing tasks. This ensures that the appropriate method is used at the right time during the assembly process. For a more detailed explanation of behavior trees and their theoretical foundations, refer to Appendix A.

Trajectory execution will be handled by an impedance controller. As described in section I, such a controller ensures smooth trajectory execution and enhances safety during human-robot interactions, which is crucial for assembly tasks involving precise movements and contact with objects.

A custom gripper was designed for physically manipulating the building blocks. As this design falls outside the primary scope of this paper, it will not be detailed here. For a more comprehensive description of the gripper, please refer to Appendix B.

A. *State Management using Behavior Trees*

As mentioned, the behavior tree is used to control the task at a high level. The behavior tree implementation used in this work is based on the BehaviorTree.CPP library[2]. This library is well-maintained, thoroughly documented, written in C++, and supports ROS integration, making it an ideal choice for this work.

Figure 3 illustrates the behavior tree utilized in this work. The tree operates by assigning robot functionalities to various nodes, within the framework provided by the selected library. For details regarding the implementation, please refer to the iiwa_bt package in the associated Git repository[3]. For a more in-depth, theoretical discussion of behavior trees, refer to Appendix B.

[2]https://github.com/BehaviorTree/BehaviorTree.CPP
[3]https://github.com/TomLim210/thesis

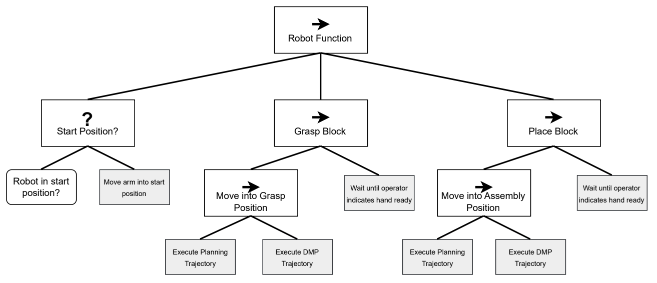

Robot Function
?
Start Position?
Grasp Block
Place Block
Robot in start position?
Move arm into start position
Move into Grasp Position
Wait until operator indicates hand ready
Move into Assembly Position
Wait until operator indicates hand ready
Execute Planning Trajectory
Execute DMP Trajectory
Execute Planning Trajectory
Execute DMP Trajectory

Fig. 3. Visual representation of the behavior tree used to describe the assembly task.

B. *Trajectory Planning using MoveIt!*

MoveIt! will be used for the planning part of the framework. MoveIt! is an good fit for this framework due to its advanced motion planning capabilities and ease of implementation. As one of the most widely used and powerful planning tools in the robotics community, MoveIt! allows for the generation of highly efficient trajectories, making it well-suited for tasks involving large, reaching motions. Given that MoveIt! is integrated with the Robot Operating System (ROS), makes for a relatively easy development process.

Another benefit of MoveIt! is its support for custom controllers using the ROS control framework. As mentioned, the impedance controller used in this work, also has a ROS control integration on top of it. This allows us to launch MoveIt! using the specified impedance controller. The benefit of this setup is that after configuration, MoveIt! will execute any given commands with the provided impedance controller.

Given that MoveIt! is so widely used, many robotics manipulators already have existing MoveIt! configurations as part of their public repository. The KUKA iiwa is no exception and it also comes with out of the box MoveIt! functionality. However, the default setup can not be used for much more than manually showcasing it's capabilities. In order to use it in the proposed framework, custom code was implemented via the MoveIt! API (Application programming interface) which allows users to access MoveIt! functionalities via Python or C++ code.

Going back to the proposed framework, MoveIt! will be responsible to generate trajectories between known configurations. The first objective is thus to create a method for storing those configurations. The second objective is to provide the user with an interface that allows them to command the robot to move into these configurations. These functionalities are located in the iiwa_planning package in the associated Git repository[3]

A python script starts a ROS node and allows the user to store different robot configurations in a yaml file. After moving the robot into a desired configuration, the user enters a name for that configuration and the node checks if there already exists a configuration with that name and if so, overwrites it and if not adds a new configuration to the list.

The same package also provides a ROS node, iiwa_planning_node, that provides the user with an interface to select a configuration. After the user has entered a valid configuration name, the node uses the MoveIt! API to command the robot accordingly.

C. Dynamical Movement Primitives

As outlined in the literature, Dynamical Movement Primitives (DMPs) are one of the most commonly used methods for learning trajectories [14]. They are especially effective for tasks that are difficult to program manually or require frequent adaptation, making them an ideal solution for handling the intricate grasping and placing movements involved in assembly tasks.

The basic concept behind dynamical movement primitives (DMPs) is to model movement as a combination of dynamical systems. The state variables of these systems represent trajectories for controlling elements such as the 7 joints of a robot arm or the position and orientation of its end-effector. The goal of the movement is captured by an attractor state, which is the endpoint of the trajectory.

One of the main benefits of DMPs is that they retain the desirable features of linear dynamical systems, such as guaranteed convergence to the goal, robustness against disturbances, and independence from time. At the same time, DMPs can represent more complex and smooth movements by introducing a non-linear forcing term. This forcing term is typically learned from demonstrations and can be further refined using reinforcement learning.

The DMPs notation follows that of a spring-damper model, as shown in [26] they can be described as:

$$\tau \ddot{\mathbf{y}} = \alpha(\beta(g - y) - \dot{y}) + f, \tag{1}$$

which has first-order notation:

$$\tau \dot{\mathbf{z}} = \alpha_z(\beta_z(g - y) - z) + f, \tag{2}$$
$$\tau \dot{\mathbf{y}} = z, \tag{3}$$

where τ is a time constant and α_z and β_z are positive constants. If the forcing term $f = 0$, these equations represent a globally stable second-order linear system with $(z, y) = (0, g)$ as a unique point attractor. With appropriate values of α_z and β_z, the system can be made critically damped (with $\beta_z = \alpha_z/4$) in order for y to monotonically converge toward g [26].

Since the goal is to replicate a demonstrated trajectory, the forcing term is non-zero. This makes that it is no longer guaranteed that the system will converge towards the goal state g. In order to solve this, a gating term is added to the forcing function, which is 1 at the beginning of the movement and 0 at the end. Authors of [27] suggest to use an exponential system to formulate the gating system.

While solving the converging issue, adding the gating term makes the forcing function depended on time. When the system depends on time, the movement is tied to a fixed timeline, making it less adaptable to variations. For example, if the movement needs to be executed faster or slower, a time-dependent system would require significant adjustments to maintain the quality of the motion.

By making the system autonomous, the movement's progression is determined by the internal state of a dynamical system, often referred to as the phase variable. This phase variable governs the progression of the movement from start to finish, regardless of how fast or slow the movement needs to be. Authors of [27] suggested to use the same dynamical system for the gating and phase. Thus the phase of the movement starts at 1, and converges to 0 towards the end of the movement, just like the gating system. This allows the same motion to be scaled in time without altering the underlying dynamics, making the system more adaptable to different conditions.

1) ROS Implementation: Given the DMPs popularity, there are many public repositories that implement them. However, to the best of our knowledge, there is not one that has integrated DMPs with ROS. In order to solve this, another ROS package is created that is basically a ROS wrapper around the DMP repository described in [28].

This repository implements DMPs as described in section II-C based on the work [26]. Besides providing a method for implementing DMPs, the authors also provide a framework to optimize the DMP for a given task. Their workflow, which is adopted in this work is as follows:

1) Train the DMP with a demonstration.
2) Define the task and implement executing DMPs on the robot.
3) Tune the exploration noise for the optimization.
4) Prepare the optimization.
5) Run the optimization update-per-update.
    a) Executing the DMPs.
    b) Update the distribution.
    c) Plotting intermediate results.

Important to note is that the first step consists of learning the forcing function. The optimization itself refers to further fine tuning the weights of the DMP based on a secondary defined task. During the execution of a given DMP, a cost variables file is created which is used to asses the performance of that iteration. The reason for this extra optimization is that an initial human demonstration might not be the optimal solution, in fact it most likely is not. Instead the demonstration provides the robot with a solid starting point which immediately points the robot towards the optimal solution, thus saving a lot of time.

Integrating this into ROS meant creating a method for recording and saving trajectories and a method for executing DMP iterations. The rest of the process can be done offline and thus does not require changing. As for recording the demonstrated trajectories it is important to consider what data to record. For a robotic manipulator this usually comes down to either recording the end-effector states or the joint states.

In our case, the demonstrations are done via kinesthetic guiding of the end-effector. During such demonstrations the individual joints simply follow the end-effector, in other words they are not actively controlled. Therefore a recording in joint space might cause the resulting DMP to try to reproduce certain joint configurations that are not intended at all. It is more logical to record in the end-effector space. Another reason for recording end-effector data is that the used impedance controller provides functionality to control the end-effector state, making it more straightforward to execute DMPs that generate trajectories in this same end-effector space.

For the execution of the DMPs, a straightforward ROS node is implemented. This node integrates a given DMP within a feedback loop, using robot state information to generate the desired state, which is then directly published to the ROS-integrated impedance controller.

D. *Impedance Control*

As mentioned, an impedance controller is essential for ensuring safe human-robot interaction. This control mechanism works by simulating a virtual spring between the end-effector and the reference position, effectively creating a spring-damper-mass system. The behavior of this system can be described by the following equation:

$$\mathbf{F_{ext}} = K(\mathbf{x_r} - \mathbf{x}) - D\dot{\mathbf{x}}, \tag{4}$$

where $\mathbf{F_{ext}}$ represents the force applied to move the end-effector towards the reference position, K is the spring stiffness, $\mathbf{x_r}$ and $\mathbf{x}$ are the reference and actual positions, respectively, and D is the damping term that stabilizes the system.

The impedance controller used in this paper is described in [29]. From their Git repository: "The controller is developed using the seven degree-of-freedom (DoF) robot arm LBR iiwa by KUKA AG .... This controller is used and tested with ROS 1 melodic and noetic. ... a ROS control integration on top of it."

These properties in combination with clear documentation make it relatively straightforward to use the same setup for executing both MoveIt!- and DMP-generated trajectories.

III. Experiments & Analysis

This section details experiments conducted to assess the proposed framework. Initial experiments evaluate the planning and learning methods independently in assembly tasks, emphasizing their respective limitations. The final experiment then demonstrates the complete framework, showcasing how combining both methods addresses these limitations. All experiments are performed using both KUKA iiwa 7 and iiwa 14 robots equipped with a custom 3D-printed gripper.

These experiments are designed within the context of the Rhizome 2.0 project, aimed at developing methods for robotic habitat construction on Mars. Given the project's focus on efficiency and resource constraints, the experiments evaluate the proposed framework's ability to handle assembly tasks effectively in a simulated remote environment.

A. *Planning-Based Trajectory Generation*

This experiment aims to evaluate the planning framework's effectiveness in quickly positioning the robot's end effector near grasping and assembly points, while identifying limitations in handling complex orientation requirements which are essential for precise assembly tasks. The hypothesis is that the planning framework can achieve efficient and accurate movements to predefined configurations but may struggle with orientation adjustments when provided with simplified goal data.

To examine this, the experiment will first involve setting the robot in gravity compensation mode, enabling manual placement in two key configurations: "grasp" and "assembly." These configurations will be defined to match exact positions and orientations necessary to pick up and place blocks located at known positions. After defining and storing these configurations, the planning framework will be used to command the robot to move between them. The initial trajectory will be recorded to showcase the planning framework's accuracy in reaching stored configurations with minimal error, even over multiple executions.

To simulate a more limited perception system, an offset "perceived goal" will then be provided, which includes only x, y, and z coordinates, excluding orientation data. This choice of a simple perception model reflects the resource limitations common in many real-world applications, where high-end, orientation-detecting perception systems may not be feasible. Particularly in the context of this paper, which involves the Rhizome 2.0 project and the goal of building habitats on Mars, minimizing equipment requirements is crucial. With fewer resources required, the system could be more resilient and practical in such a remote, resource-constrained environment.

This setup will allow us to observe how the end-effector orientation deviates from the exact "grasp" and "assembly" configurations. The hypothesis is that the end effector will reach the desired position but not the correct orientation, thus highlighting a key limitation in using only a planning-based framework with limited perception systems for assembly.

The main metrics for this experiment are the ability to handle goal variations and the efficiency of setup. Specifically, we will measure the deviation in end-effector orientation when the robot is provided with an offset goal compared to the original configuration, as well as the time required to set up the planning framework. These metrics will help evaluate how well the planning approach can achieve accurate positioning under simplified perception constraints and highlight any limitations in assembly tasks that lack advanced orientation detection capabilities.

B. *DMP-Based Trajectory Generation and Optimization*

This experiment evaluates the effectiveness of Dynamic Movement Primitives (DMPs) in performing a full assembly movement, which includes both precise orientation for grasping and larger, reaching motions between grasp and assembly points. The hypothesis is that, while DMPs can accurately replicate demonstrated movements, setting up a DMP for a complex, large-scale assembly task will be cumbersome and time-intensive due to the challenge of demonstrating an entire assembly motion manually. Given the extensive workspace, initial demonstrations may include unintended deviations and oscillations.

The experiment begins by placing the robot in gravity compensation mode, which allows for kinesthetic guidance of the end-effector. A complex movement that encompasses both grasping and placing actions will be demonstrated and recorded, providing a baseline trajectory for the DMP. The number of basis functions for the DMP will be tuned to achieve a balance between accuracy and computational efficiency, and the trained DMP will be saved as the initial model for evaluation.

The primary metrics for this experiment are the DMP's ability to adapt to variations in goal positions and the ease of setup. To evaluate adaptability, a "perceived goal" will be introduced as a new target, simulating a perception system capable of specifying an approximate position but lacking precise orientation data. It is expected that the DMP will adjust its trajectory to accommodate the new goal while maintaining the learned orientation from the original demonstration. This test will reveal whether the DMP framework can handle position variations effectively, addressing a key challenge identified in the planning-based experiment.

In terms of expected results, the system should show robustness by maintaining proper end-effector orientation even when adapting to altered goal positions. The experiment will highlight how well the learning-based method can handle goal variations and provide insight into the overall efficiency of setting up the DMP framework.

C. Framework Demonstration in Rhizome 2.0 project

The purpose of this experiment is to demonstrate the integrated functionality of both the planning and DMP-based trajectory generation frameworks in the context of an assembly task. The hypothesis is that combining planning for large-scale motions with DMP for fine, precise manipulation will result in a flexible and effective framework that uses the strengths of both the planning and learning frameworks to address each others limitations.

Two key functionalities will be demonstrated. First, the hybrid trajectory generation will show how planning is used for large-reaching motions, while the DMP framework is applied to handle the precise grasping and placing movements. Second, the completion of the assembly task will validate the system's ability to execute both types of movements seamlessly in a single workflow.

The experiment protocol involves equipping the KUKA iiwa arm with the custom designed gripper and storing the required configurations for the assembly task using the planning framework. A demonstration of the grasping and placing tasks will then be recorded using the learning framework. Once both configurations are set, the behavior tree will execute the task in several stages: it will first use planning to move into the grasping position, then use the DMP to perform the actual grasp. Afterward, the robot will move the block to the assembly position using planning, and finally, the DMP will control the precise placement of the block. This sequence will be repeated multiple times to evaluate repeatability.

To evaluate the complete system, the experiment will focus on assessing how the integrated framework addresses scenarios that highlight the limitations of using planning and learning methods independently. Specifically, the system will be tested for its ability to adapt to variations in goal positions, addressing a known constraint in planning-only approaches that require precise goal definitions and orientations. Additionally, the setup time and complexity of large demonstrations, challenges often associated with learning-based methods, will be examined. By observing the system's performance under these conditions, we aim to confirm that the combined approach mitigates these individual limitations, providing a more robust and adaptable solution for complex assembly tasks.

D. Analysis

This section presents the analysis of the planning and learning methods for robotic assembly, focusing on the main metrics: the ability to handle goal variations and the time required for setup.

The planning framework was efficient in setting up predefined grasping and assembly configurations, requiring only around five minutes to establish endpoint positions. This minimal setup time highlights the ease and convenience of using a planning-based approach for large-scale, reaching motions. However, the framework struggled with goal variations that lacked orientation data. When presented with altered goal positions that only included x, y, and z coordinates, the planning method accurately reached the designated position but failed to achieve the necessary orientation. This limitation emphasizes the framework's dependence on advanced perception systems to ensure precise end-effector orientation.

In contrast, the learning-based approach, using Dynamic Movement Primitives (DMPs), showed a strong ability to adapt to variations in goal positions. The DMP framework maintained the correct end-effector orientation even when the goal position was modified, demonstrating its robustness and flexibility in handling positional offsets. However, this adaptability came at the cost of a more time-consuming and labor-intensive setup process. Manually demonstrating and recording complex assembly movements, especially those spanning a large workspace, proved to be inconvenient and physically demanding, requiring significant effort to ensure accurate trajectories.

The final experiment evaluated the integrated framework, combining both planning and learning methods using a behavior tree. This hybrid approach successfully addressed the limitations observed when each method was used independently. The planning framework managed large, reaching motions efficiently, while the DMPs handled the intricate grasping and assembly tasks with precise orientation control. The integration allowed for seamless transitions between methods, resulting in a robust assembly process that did not require advanced perception systems. The combined approach demonstrated the benefits of leveraging the strengths of both methods to achieve efficient and adaptable robotic assembly.

1) Dealing with Goal Variations: The ability to adjust to changes in goal positions is crucial for assembly tasks, especially in environments where resources for advanced perception systems are limited. The planning framework performed well when tasked with moving the end-effector to predefined configurations, as it efficiently reached target positions with minimal error. However, a limitation emerged when goal variations were introduced. As shown in Figure 4, the planning

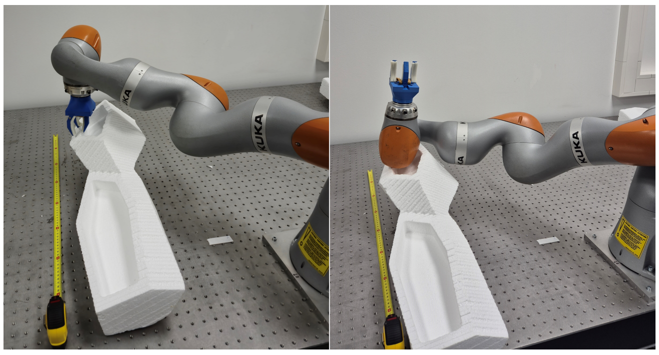


Fig. 4. Images of the robot in the original demonstrated grasp configuration (left) and in a configuration with the goal position modified by an offset of 16 cm along the x-axis (right) and no orientation data.

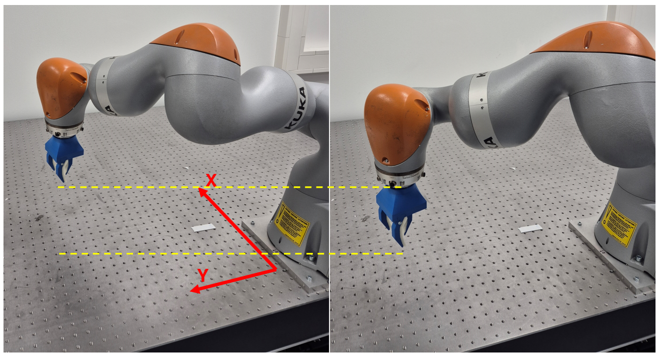


Fig. 6. End-effector configurations at the goal with the original demonstration goal (left) and the x-offset "perceived" goal (right), showing maintained orientation.

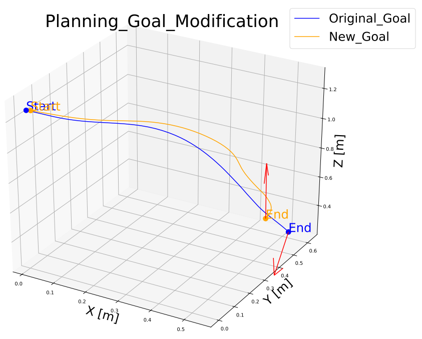


Fig. 5. End-effector trajectory when moving to the original grasp configuration and to the configuration with a 16 cm offset along the x-axis. Red arrows indicate the end-effector orientation at the end of each movement.

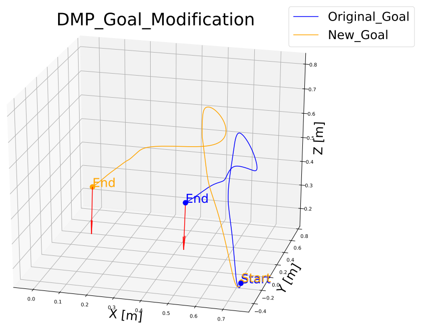


Fig. 7. Comparison of end-effector trajectories for the original and x-offset goals, with red arrows indicating consistent orientation across both trajectories.

approach could not properly handle changes to the goal's orientation when only x, y, and z coordinates were provided. The end-effector maintained a default upright orientation, demonstrating the method's dependency on having complete orientation data for accurate performance. Figure 5 makes this dependency even more clear by plotting the end-effector orientation at the end of a trajectory.

Although one might suggest augmenting the planning approach by extracting end-effector orientation from an initial configuration, it's important to note that the stored configurations in this framework are managed in joint space. This is intentional, as controlling the robot's null space during configuration storage ensures smooth transitions between preset configurations and allows operators to manage these transitions more effectively. The planning method benefits from this setup for large-scale, efficient movements but struggles with orientation control in scenarios lacking precise perception data.

In contrast, the learning-based approach, using Dynamic Movement Primitives (DMPs), excelled in adapting to goal variations. As shown in Figure 6, when presented with a new "perceived" goal with a 40 cm offset, the DMPs preserved the correct end-effector orientation. The DMP framework proved capable of maintaining learned orientation details while adjusting to positional changes, making it more adaptable and flexible than planning methods for tasks requiring precise orientation. Figure 7 further illustrates this adaptability, highlighting the consistency of end-effector orientation even when goal positions were modified.

2) Ease of Setup: The ease and speed of setup are critical in resource-constrained environments, where minimizing time and effort is essential. The planning framework demonstrated significant advantages in this regard. Setting up the system with pre-set "grasp" and "assembly" configurations took only about five minutes. This simplicity comes from the fact that only endpoint positions need to be defined; MoveIt! then calculates the trajectories automatically, making it efficient for large-scale movements. Figure 8 illustrates the trajectories into these pre-set configurations, with the end-effector orientations appropriately oriented for grasping on one side and assembly on the other. This setup efficiency is particularly valuable in off-earth scenarios, where astronauts may have limited time and need to minimize exposure outside safe habitats.

On the other hand, the learning framework posed significant challenges during setup. Demonstrating and recording the entire assembly movement required manual kinesthetic guidance, which forced the operator into awkward and non-ergonomic positions to control all seven joints of the robot. As depicted in Figure 9, this process was both physically demanding and prone to inconsistencies, even after five repeated attempts. The full setup for DMPs averaged around 20 minutes, highlighting the method's time-consuming nature. Although the learning framework can generalize from a noisy demonstration, Figure

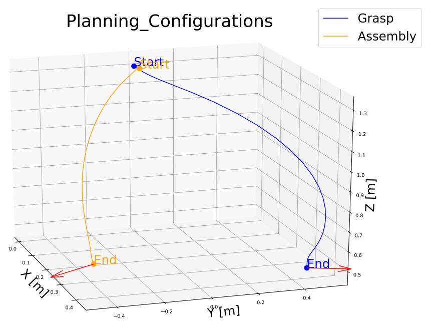


Fig. 8. Pre-set 'grasp' and 'assembly' configurations with end-effector orientation indicated by red arrows.

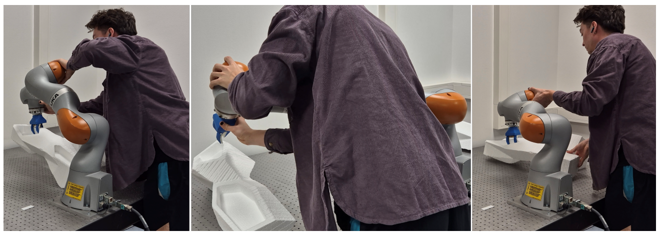


Fig. 9. Images capturing some of the awkward operator positions during the DMP demonstration process.

10 illustrates the noticeable differences between the original demonstration and the learned DMP trajectory. This deviation is problematic, as the DMP should closely replicate the demonstration, particularly at critical points like grasping or assembly. Moreover, the DMP does not inherently account for workspace or joint limits, meaning that if the learned trajectory diverges too much from the original demonstration, it could result in joint limit errors, even if the initial demonstration adhered to those constraints.

3) Sub-Conclusion: In summary, the comparison between planning and learning frameworks reveals a clear trade-off between setup efficiency and adaptability. The planning method offers quick and straightforward setup for large, reaching movements but struggles with goal variations, particularly in orientation. In contrast, the DMP-based learning approach, while more time-consuming to set up, provides the flexibility needed to adapt to positional changes without losing precision in orientation. This analysis highlights the complementary nature of these methods, which justifies the need for an integrated approach in scenarios requiring both efficient setup and adaptability.

4) Framework Demonstration: The last experiment involved testing the complete framework in the context of the Rhizome 2.0 project, more specifically the assembly of the Vonroi-shaped building blocks. The first steps involved configuring both the planning and learning frameworks similar to the previously described experiments. After configuration the behavior tree is used to execute the complete assembly task. It must be noted that due to the current limitations of the used gripper, a human operator was needed to actuate the gripper during grasping and placing.

Figure 11 shows the complete assembly process, starting with the robot grasping a block on its left side, followed by moving into the assembly configuration. At that point the assembly DMP is executed to perform the final step of the process. For a video of the complete assembly please refer to: https://youtu.be/XIbV7j7TgrY.

Figure 12 illustrates the 3D end-effector trajectory during the execution of the complete assembly task. The red-marked path represents the movement generated by the planning framework. A primary limitation of using the planning framework alone is that it requires an advanced perception system to manage changing goal positions effectively. In this integrated setup, however, the planning framework only needs two pre-set configurations, 'grasp' and 'assembly,' which are used as starting points for the learning framework. By keeping these configurations constant, the robot can efficiently and accurately navigate between them from any initial position, leveraging the strengths of the planning approach. The initial part of the trajectory in Figure 12 demonstrates the robot moving into its starting configuration from an upright position.

For the finer grasping and assembly tasks, the learning framework was implemented. In Figure 12, these movements are marked by yellow trajectories. The main limitation with the learning framework is that demonstrating and fitting DMPs to large reaching movements can be cumbersome and time-consuming. In this combined approach, however, the learning framework only manages two small complex motions: grasping and assembly.

Figure 13 further highlights the difference between demonstrating the full assembly movement and only the grasping motion. By focusing the learning framework on these specific actions, demonstrations became more ergonomic and efficient, reducing the time required. Additionally, this approach preserved the benefits of the learning framework's adaptability to goal variations, making the combined framework particularly robust.

Overall, the experiment proceeded smoothly, with no significant issues observed. Both the planning and learning methods were executed sequentially using the same impedance controller without conflicts, and the system demonstrated robustness in handling variations in starting positions without performance degradation. Consequently, the complete framework

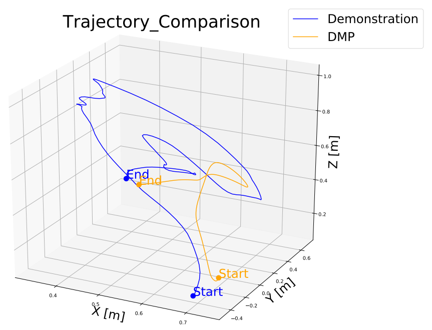


Fig. 10. Initial demonstration trajectory vs. the smoothed initial DMP trajectory after fitting.

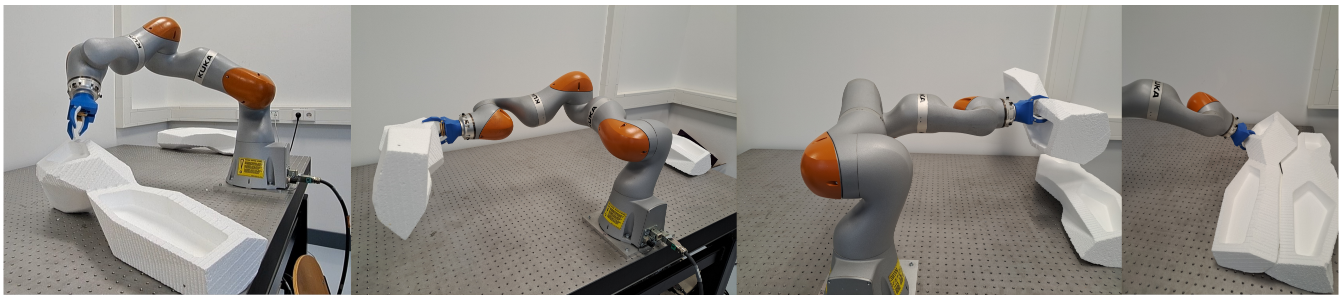


Fig. 11. Figures showing the different stages of the assembly task. Grasping is done on the left and assembly is done on the right. For a video of the complete assembly please refer to https://youtu.be/XIbV7j7TgrY.

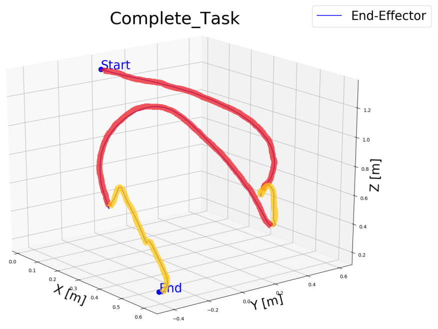


Fig. 12. 3D end-effector position during the execution of the complete assembly task starting in an upright position. Red marked trajectory is executed using the planning framework and yellow marked trajectory is the result of the learning framework.

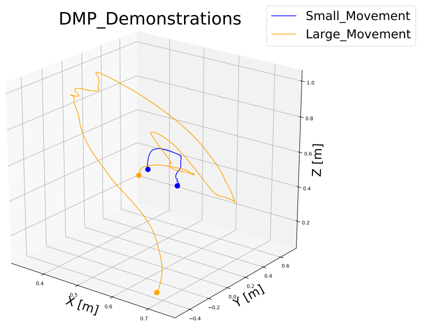


Fig. 13. Comparison between demonstrations of the complete assembly movement and a focused grasping motion.

successfully carried out the entire assembly task, underscoring its effectiveness in this complex scenario.

The key takeaway is that by combining planning and learning frameworks, the strengths of each approach compensate for the limitations of the other. The planning framework efficiently handles large, reaching motions, which would be challenging to set up with learning-based methods. Meanwhile, the learning framework addresses intricate grasping and assembly tasks, which would otherwise require an advanced perception system if only the planning approach were used.

IV. Discussion

The experiments demonstrated that the proposed framework, combining planning and learning-based trajectory generation, was capable of successfully performing a complex assembly task. One of the key challenges discussed in robotic assembly is the need to address both large-scale movements and fine, precise motions. The integration of planning for larger reaching motions and learning-based methods for more intricate tasks provided a flexible solution. The planning framework handled the large motions well, producing smooth and repeatable trajectories.

The learning-based component of the framework, specifically Dynamical Movement Primitives (DMPs), performed effectively in replicating precise grasping and placing movements. This approach demonstrated the ability of learning-based systems to adapt to complex tasks without the need for extensive programming. However, optimizing the DMPs to minimize acceleration did not result in significant change beyond the initial demonstration, suggesting that optimization might not be necessary for movements in which the main goal is simply achieving the goal configuration. While the learning approach worked well for detailed motions, future work could explore whether incorporating additional optimization parameters would result in further improvements.

Another challenge encountered was the performance of the custom gripper. While it functioned adequately during the experiments, there were moments when the blocks shifted during transportation, which affected the accuracy of the assembly. This suggests that task-specific hardware, such as grippers, plays a critical role in the overall performance of robotic assembly tasks. Improving the design of the gripper or incorporating additional control measures could enhance reliability in future applications.

A key factor in the framework's success was the use of the impedance controller, which enabled smooth trajectory execution and safe human-robot interaction. This flexibility allowed for manual adjustments, especially important with the custom gripper, which required manual actuation during the assembly process. The ability to interact safely with the robot while maintaining precise control of the task proved to be an essential aspect of this framework, particularly in the context of human-robot collaboration.

A broader question arises as to whether combining planning and learning truly provides a significant advantage over using either method alone. While this integrated framework allowed for flexible handling of both large and small-scale motions, it is worth considering if the added complexity of a hybrid approach is always justified. In this work, the combination was essential to address specific limitations of each method within the given scenario. However, in contexts where these limitations are less pronounced, relying on just one of the frameworks may prove sufficient and more efficient.

Looking forward, a key improvement to consider is implementing the framework on a mobile manipulator. In scenarios where objects need to be transported between locations, such as in the Rhizome 2.0 project, mobility is essential for the system to function autonomously. A mobile platform would enable the robot to navigate its environment and perform tasks without human intervention, making the framework more applicable to real-world scenarios where flexibility and movement are required. This addition would significantly enhance the capabilities of the system and broaden its applicability in dynamic environments.

V. Conclusion

This paper presented a novel assembly framework that integrates both planning and learning-based trajectory generation methods to handle complex assembly tasks. The planning approach, utilizing MoveIt!, was employed for large-scale movements between predefined locations, while Dynamic Movement Primitives (DMPs) were applied to manage fine, precise movements such as grasping and placing. The entire system was controlled by a behavior tree and executed using an impedance controller to ensure smooth and safe robot operation, particularly during manual interventions.

Through a series of experiments, the framework was evaluated in terms of its ability to perform an assembly task involving the manipulation of custom Voronoi-shaped building blocks. The results demonstrated that both the planning and learning methods functioned effectively in their respective roles, with the impedance controller proving essential in ensuring safe operation and adaptability. The combination of planning for large movements and learning for fine manipulation allowed the system to handle the full assembly process without errors, confirming the validity of the proposed approach.

In answering the research question, this work demonstrates that the integration of planning and learning into a single framework offers a flexible and efficient method for handling complex assembly tasks. The hybrid approach, while potentially more complex than using either planning or learning alone, provided notable advantages in terms of adaptability to different sub-tasks. However, further exploration could be conducted to determine the specific contexts in which this combination significantly outperforms single-method systems.

Future work could improve the framework's versatility by incorporating a perception system to dynamically detect goal positions, thus reducing the need for manual configuration. Additionally, implementing the framework on a mobile manipulator would enhance its applicability, particularly in scenarios such as the Rhizome 2.0 project, where parts must be moved between different locations. Overall, the proposed framework has shown promise as an efficient and adaptable solution for robotic assembly tasks in both static and dynamic environments.

References

[1] J. Bloem, M. Van Doorn, S. Duivestein, D. Excoffier, R. Maas, and E. Van Ommeren, "The fourth industrial revolution," *Things Tighten*, vol. 8, no. 1, pp. 11–15, 2014.

[2] A. Bauer, D. Wollherr, and M. Buss, "Human–robot collaboration: a survey," *International Journal of Humanoid Robotics*, vol. 5, no. 01, pp. 47–66, 2008.

[3] E. Matheson, R. Minto, E. G. Zampieri, M. Faccio, and G. Rosati, "Human–robot collaboration in manufacturing applications: A review," *Robotics*, vol. 8, no. 4, p. 100, 2019.

[4] M. Saveriano, F. J. Abu-Dakka, A. Kramberger, and L. Peternel, "Dynamic movement primitives in robotics: A tutorial survey," *The International Journal of Robotics Research*, vol. 42, no. 13, pp. 1133–1184, 2023.

[5] V. Villani, F. Pini, F. Leali, and C. Secchi, "Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications," *Mechatronics*, vol. 55, pp. 248–266, 2018.

[6] H. Bier, S. Khademi, C. van Engelenburg, J. Prendergast, and L. Peternel, "Computer vision and human–robot collaboration supported design-to-robotic-assembly," *Construction Robotics*, pp. 1–7, 2022, green Open Access added to TU Delft Institutional Repository 'You share, we take care!' - Taverne project https://www.openaccess.nl/en/you-share-we-take-care Otherwise as indicated in the copyright section: the publisher is the copyright holder of this work and the author uses the Dutch legislation to make this work public.

[7] H. Bier, S. Khademi, C. van Engelenburg, J. M. Prendergast, and L. Peternel, "Computer vision and human–robot collaboration supported design-to-robotic-assembly," *Construction Robotics*, vol. 6, no. 3, pp. 251–257, 2022.

[8] D. Di Prima, "Human-robot collaboration through a new touch emulation system for assembly tasks," Ph.D. dissertation, Politecnico di Torino, 2020.

[9] S. Chitta, I. Sucan, and S. Cousins, "Moveit![ros topics]," *IEEE robotics & automation magazine*, vol. 19, no. 1, pp. 18–19, 2012.

[10] S. Schaal, "Is imitation learning the route to humanoid robots?" *Trends in cognitive sciences*, vol. 3, no. 6, pp. 233–242, 1999.

[11] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, "Survey: Robot programming by demonstration," *Springer handbook of robotics*, pp. 1371–1394, 2008.

[12] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.

[13] S. Chernova and A. L. Thomaz, *Robot learning from human teachers*. Morgan & Claypool Publishers, 2014.

[14] H. Ravichandar, A. S. Polydoros, S. Chernova, and A. Billard, "Recent advances in robot learning from demonstration," *Annual review of control, robotics, and autonomous systems*, vol. 3, no. 1, pp. 297–330, 2020.

[15] Z. Zhu and H. Hu, "Robot learning from demonstration in robotic assembly: A survey," *Robotics*, vol. 7, no. 2, p. 17, 2018.

[16] S. Schaal, "Dynamic movement primitives-a framework for motor control in humans and humanoid robotics," in *Adaptive motion of animals and machines*. Springer, 2006, pp. 261–280.

[17] A. Kramberger, R. Piltaver, B. Nemec, M. Gams, and A. Ude, "Learning of assembly constraints by demonstration and active exploration," *Industrial Robot: An International Journal*, vol. 43, no. 5, pp. 524–534, 2016.

[18] K. Collins, A. Palmer, and K. Rathmill, "The development of a european benchmark for the comparison of assembly robot programming systems," in *Robot Technology and Applications: Proceedings of the 1st Robotics Europe Conference Brussels, June 27–28, 1984*. Springer, 1985, pp. 187–199.

[19] B. Nemec, M. Simonič, and A. Ude, "Learning of exception strategies in assembly tasks," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 6521–6527.

[20] B. Nemec, L. Žlajpah, S. Šlajpa, J. Piškur, and A. Ude, "An efficient pbd framework for fast deployment of bi-manual assembly tasks," in *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*. IEEE, 2018, pp. 166–173.

[21] M. Saveriano, S.-i. An, and D. Lee, "Incremental kinesthetic teaching of end-effector and null-space motion primitives," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 3570–3575.

[22] S. Haddadin and E. Croft, "Physical human–robot interaction," *Springer handbook of robotics*, pp. 1835–1874, 2016.

[23] L. Villani and J. De Schutter, "Force control," *Springer handbook of robotics*, pp. 195–220, 2016.

[24] N. Hogan, "Impedance control: An approach to manipulation," in *1984 American control conference*. IEEE, 1984, pp. 304–313.

[25] P. Song, Y. Yu, and X. Zhang, "Impedance control of robots: an overview," in *2017 2nd international conference on cybernetics, robotics and control (CRC)*. IEEE, 2017, pp. 51–55.

[26] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: learning attractor models for motor behaviors," *Neural computation*, vol. 25, no. 2, pp. 328–373, 2013.

[27] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Movement imitation with nonlinear dynamical systems in humanoid robots," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, vol. 2. IEEE, 2002, pp. 1398–1403.

[28] F. Stulp and G. Raiola, "Dmpbbo: A versatile python/c++ library for function approximation, dynamical movement primitives, and black-box optimization," *Journal of Open Source Software*, 2019. [Online]. Available: https://www.theoj.org/joss-papers/joss.01225/10.21105.joss.01225.pdf

[29] M. Mayr and J. M. Salt-Ducaju, "A c++ implementation of a cartesian impedance controller for robotic manipulators," *Journal of Open Source Software*, vol. 9, no. 93, p. 5194, 2024. [Online]. Available: https://doi.org/10.21105/joss.05194

A

Appendix - Behavior Trees

This information is based on the work [1], which is an extensive survey of behavior trees in robotics.

A BT is a directed tree where we apply the standard meanings of root, child, parent, and leaf nodes. The leaf nodes are called execution nodes and the non-leaf nodes are called control flow nodes.

The execution of a BT starts from the root node, that generates signals called Ticks with a given frequency. These signals enable the execution of a node and are then propagated to one or several of the children of the ticked node. A node is executed if, and only if, it receives Ticks. The child immediately returns Running to the parent, if its execution is under way, Success if it has achieved its goal, or Failure otherwise.

Sequences are used when some actions, or condition checks, are meant to be carried out in sequence, and when the success of one action is needed for the execution of the next. The Sequence node routes the ticks to its children from the left until it finds a child that returns either Failure or Running, then it returns Failure or Running accordingly to its own parent. It returns Success if and only if all its children return Success.

Fallbacks are used when a set of actions represent alternative ways of achieving a similar goal. Thus, the Fallback node routes the ticks to its children from the left until it finds a child that returns either Success or Running, then it returns Success or Running accordingly to its own parent. It returns Failure if and only if all its children return Failure.

Parallel nodes tick all the children simultaneously. Then, if out of the children return Success, then so does the parallel node. If more than return Failure, thus rendering success impossible, it returns Failure. If none of the conditions above are met, it returns running.

Action nodes typically execute a command when receiving ticks, such as e.g. moving the agent. If the action is successfully completed, it returns Success, and if the action has failed, it returns Failure. While the action is ongoing it returns Running.

Condition nodes check a proposition upon receiving ticks. It returns Success or Failure depending on if the proposition holds or not. Note that a Condition node never returns a status of Running. Conditions are thus technically a subset of the Actions, but are given a separate category and graphical symbol to improve readability of the BT and emphasize the fact that they never return running and do not change the world or any internal states/variables of the BT.

B

Appendix - Custom Gripper

To accommodate the unique shape of the building blocks used for demonstrating the assembly capabilities of the framework, a custom gripper was designed. Since the focus of this work lies more in the development of the robotics framework rather than the gripper design, a fixed amount of time was allocated for its development. Although the gripper could benefit from further improvements, it functioned adequately for the purposes of the current experiments.

The gripper consists of a main body and three separate fingers. The main body is designed to be mounted onto the end plate of the KUKA iiwa robots. Two static fingers are press-fitted into the main body, while the third finger is movable, sliding within a dovetail channel. Due to the time constraints, the gripper was manually actuated.

Figure B.1 provides a close-up view of the designed gripper on the left. The holes in the body allow access to the mounting screws. The movable finger is attached to the static fingers using elastic bands to apply sufficient pressure when holding the building blocks. A foam layer is added to the fingers to increase the contact area, ensuring a more secure grip.

On the right, the figure shows the gripper holding a Voronoi-shaped block in a horizontal position. This position was the most stable, although the gripper was also able to hold the blocks vertically, occasional slipping occurred in this position.

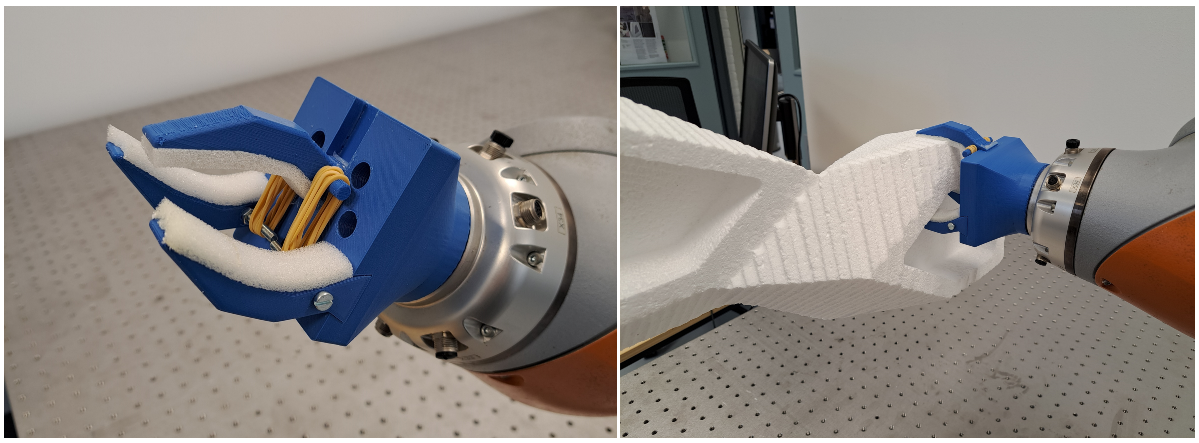


Figure B.1: Closeups of the gripper.

C

Appendix - DMP Setup

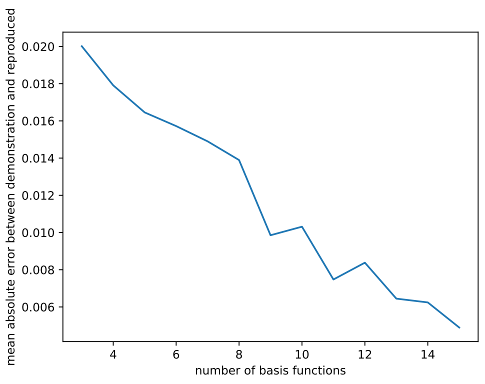
The implementation of DMPs in this work uses the publicly available DMP repository described in [2].

Before analyzing the performance of the DMPs, it was necessary to make some decisions about its configuration. The first decision involved selecting the number of basis functions to use for generating the forcing function. As the number of basis functions increases, the DMP's ability to follow the demonstrated trajectory improves significantly. This is confirmed in figure C.1a, which plots the mean error against the number of basis functions.

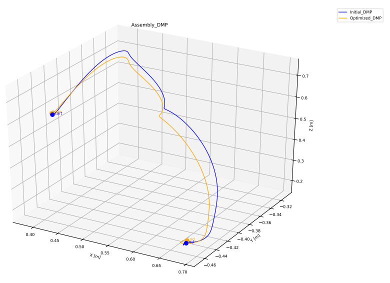
Choosing the appropriate number of basis functions is important. While a higher number of basis functions requires more computational resources, the increase in accuracy outweighs this cost. Since the DMP execution is implemented in C++ code, the additional computational load is negligible. Based on these findings, using 13 to 15 basis functions was found to provide sufficient accuracy for both the grasping and assembly DMPs.

The next step in the experiment was to optimize the chosen initial DMP based on a defined task. The idea behind this is that a human demonstration might not be optimal for a given start but can be used as a very good starting point. For example, authors of the used DMP library, use this to optimize the DMP for throwing a ball towards a certain location. During the execution of the DMP the position of the ball is recorded and saved as a cost variable which can then be used to asses the performance of a given DMP.

This work differs in that case because while the human input does contain unwanted jerky movements, in general it is considered as the actual desired trajectory and the robot should not deviate too much from it. Therefore the optimization process in this work only minimizes acceleration during the execution.

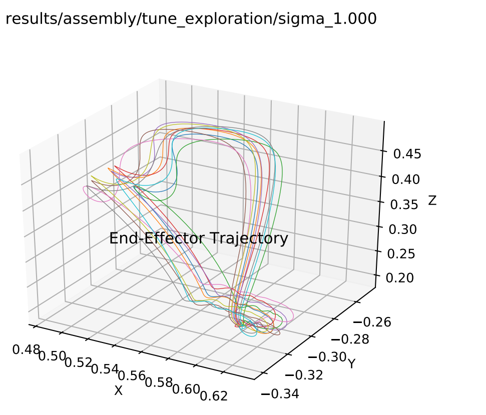


(a) Mean absolute errors for different numbers of basis functions used



(b) 3D end-effector position during the execution of the initial and optimized DMP.

Figure C.1



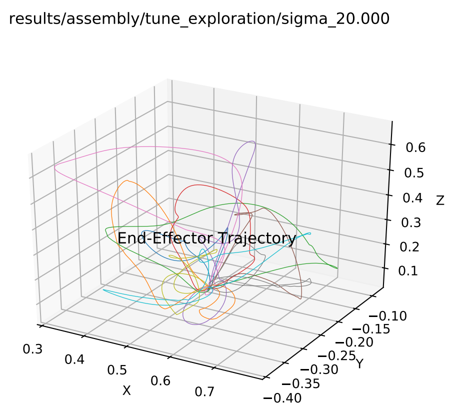


Figure C.2: Plotted exploration rollouts with sigma 1.0 on the left and 20.0 on the right.

During the stochastic optimization process, the parameters of the DMP are sampled from a Gaussian distribution. The mean of this distribution corresponds to the parameters obtained from training the DMP on the initial demonstration.

The covariance matrix of the sampling distribution controls the exploration magnitude, which is defined by sigma. The diagonal of the covariance matrix is initialized with $sigma^2$. If sigma is set too low, the exploration will be limited, potentially less than the inherent variability in the robot's movements, making learning ineffective. Conversely, if sigma is set too high, it could lead to unsafe behavior, such as the robot exceeding acceleration or joint limits, or colliding with its surroundings. Figure C.2 illustrates the results of exploration using sigma values of 1.0 and 20.0.

In this experiment, it is important that the DMP does not deviate significantly from the demonstrated trajectory. As shown in figure C.2, a sigma value of 20.0 leads to the exploration of highly different trajectories. While this is expected in general optimization, for the purposes of this work, we aim to maintain close adherence to the initial demonstration with only limited exploration. Therefore, a sigma value of 1.0 was chosen.

In the final step of the DMP experiment, the optimization process aimed to minimize joint accelerations. Interestingly, as shown in figure C.1b, the optimized DMP closely resembled the initial DMP obtained from training. This could be due to the relatively low sigma value used during the optimization, which restricted the exploration of alternative trajectories. Another possible reason is that the optimization was focused solely on minimizing accelerations, without incorporating a secondary task beyond simply reaching the goal configuration. As a result, the DMP did not significantly deviate from the initial demonstration.

References

[1] Matteo Iovino et al. “A survey of behavior trees in robotics and ai”. In: *Robotics and Autonomous Systems* 154 (2022), p. 104096.

[2] Freek Stulp and Gennaro Raiola. “DmpBbo: A versatile Python/C++ library for Function Approximation, Dynamical Movement Primitives, and Black-Box Optimization”. In: *Journal of Open Source Software* (2019). DOI: `10.21105/joss.01225`. URL: `https://www.theoj.org/joss-papers/joss.01225/10.21105.joss.01225.pdf`.