Autonomous Robotic Manipulator Software

Collin J. Cresta* , Radhika Rajaram†, A. Kyle McQuarry‡, and Thea V. Avila§
Analytical Mechanics Associates, Hampton, VA, 23666

Matthew P. Vaughan¶ and John R. Cooper‖
NASA Langley Research Center, Hampton, VA, 23681

Autonomous robotic manipulation requires a deep and wide stack of supporting software. This paper presents Autonomous Robotic Manipulator Software (ARMS), a software suite designed at NASA Langley Research Center to support research and development of different algorithms for In-space Servicing, Assembly and Manufacturing (ISAM). ARMS solves common challenges along the autonomous manipulation software stack. Various challenges, such as integration with commercial hardware, simulation, and path planning, are solved through the use of Robot Operating System 2 and its community-developed packages. Other challenges, such as configuration management and task definition, and execution are solved in software built on those tools. The result is a modular approach to robotic system definition, agent actions, and assembly task definitions. ARMS has been used in two ISAM projects at NASA Langley Research Center, the Precision Assembled Space Structures project and the Built On-orbit Robotically assembled Gigatruss project.

I. Nomenclature

ARMS	=	Autonomous Robotic Manipulation Software
BORG	=	Built On-orbit Robotically assembled Gigatruss
GUI	=	Graphical User Interface
ISAM	=	In-space Servicing Assembly and Manufacturing
LaRC	=	Langley Research Center
PASS	=	Precision Assembled Space Structure
ROS 2	=	Robot Operating System 2
SRDF	=	Semantic Robotic Description Format
URDF	=	Universal Robotic Description Format

II. Introduction

This paper describes Autonomous Robotic Manipulator Software (ARMS), a software system used for autonomous assembly operations. ARMS helps facilitate the definition of robotic assembly systems and assembly tasks. The software system is designed to be generic and extensible to diverse assembly mission configurations. This extensibility requires that the software system should be compatible with different manipulators, end-effectors, and other actuators. Additionally, the definition of actions at various levels of complexity should be intuitive.

In-Space Servicing, Assembly, and Manufacturing (ISAM) is a critical capability needed for NASA Artemis mission goals. ISAM mission tasks require parts to be assembled to create structures larger than can be carried by a single launch vehicle. NASA Langley Research Center (LaRC) has multiple ongoing projects in the ISAM domain including Precision Assembled Space Structure (PASS), Tall Lunar Tower, and Built On-orbit Robotically assembled Gigatruss (BORG) [1, 2]. Among these ISAM projects there are common processes that occur during assembly of various

*Robotics Engineer, Analytical Mechanics Associates, resident at NASA Langley Research Center
†Robotics Engineer, Analytical Mechanics Associates, resident at NASA Langley Research Center
‡Lead Software Developer, Analytical Mechanics Associates, resident at NASA Langley Research Center
§Software Engineer, Analytical Mechanics Associates, resident at NASA Langley Research Center
¶Research Computer Scientist, Autonomous Integrated System Research Branch
‖Research Engineer, Autonomous Integrated System Research Branch

structures. A strut or module needs to be retrieved from a stowed location, delivered to an insertion point, and joined to the main structure. These projects share common obstacles and solutions, from hardware/software integration to high-level task planning and execution. This work is an extension and iteration of previous work at LaRC [3, 4].

ARMS is built on top of the Robot Operating System 2 (ROS 2). ROS 2 provides a modular and extensible environment for robotics software development through a suite of core infrastructure and a large ecosystem of community-developed packages (e.g. hardware drivers, planning and control algorithms, and other utilities) [5]. ARMS leverages the motion planning pipeline of the MoveIt 2 package as a foundation for advanced manipulation behaviors [6].

ARMS solves the problem of configuration management with a tool called the Robot Factory and the problem of task definition and execution with high level abstractions of agent actions and tasks. The Robot Factory is used to build the requisite collection of configuration files from user input to represent the necessary assembly systems. ARMS defines abstract agent types that are used to encode functionality of components in the assembly system. Tasks are compositions of agent behaviors that achieve high level functionality, such as picking up a complex object and installing it into a larger system. The PASS project is the first use case and primary driver of ARMS development, where it has been used to assemble modules for the backbone structure of a 20m space telescope mirror.

III. System Architecture

LaRC developed three main components that make up ARMS. The first is the Robot Factory, where the configuration is input and what will start all ROS 2 nodes required by this definition. The second are the Agents, these include manipulator and end-effector agents that can interact with the environment. The third are High Level Tasks, which coordinate action between multiple agents.

A. Robot Factory

Robot Factory is the subsystem of ARMS responsible for handling input configuration files, generating robot configuration files, and starting all nodes that are specified for the mission. Traditional robot application development often involves the creation of highly customized packages tailored to specific robot configurations, resulting in a lack of flexibility and interoperability. This approach necessitates the creation of unique robot descriptions, controller definitions, MoveIt 2 configurations, launch procedures, etc. for each application. The space of possible custom package definitions grows combinatorially in the number of components making it difficult to iterate on large systems. In contrast, the approach of the Robot Factory enables the seamless composition of diverse robotic components such as manipulators, bases, tools, and end-effectors. The approach taken here aligns with the core principals of ROS 2, emphasizing modularity and reusability of robotic systems [5].

ARMS uses Universal Robot Description Format (URDF) files, which encode the physical properties of the robot. When combined with xacro, an XML scripting language, URDF files are highly reconfigurable. Xacro files can be filled with variables and programmatically modified to generate new URDF files on the fly. Robot Factory uses the versatility of xacro to allow the user to compose a configuration that creates a robotic system including bases, manipulators, end-effectors, sensors, and other actuators. Robot Factory handles switching between hardware, and two levels of simulation, one using Gazebo as a dynamic simulation and another using 'mock hardware' from ROS 2 Control to create a kinematic simulation. The goal of the Robot Factory is to allow for rapid mission configuration prototyping, by making all elements composable, so the user can easily reconfigure system level hardware definitions.

A user can build a robot description in a launch file by defining components necessary for the robotic assembly system required for the mission. The result is a configuration that includes all dependencies as in Figure 1. Table 1 shows a list of parameters that can be passed in for a component. Additionally, if a component's xacro file has additional arguments that are needed, they can be passed through this description.

Robot Factory handles launching all controllers, interfaces, and a move group node for all manipulators in the configuration specified. The completion of the launch process allows for the action servers to handle requests as described in Section III.B. ARMS has a configuration library of over 30 components that can be composed into any number of robotic system configurations. These components include robotic manipulators, bases, end-effectors, sensors, and payloads.

B. Agent Definitions

ARMS defines abstractions of assembly system agents to assist in launching elements and determine what capabilities they may have. These abstractions are used in the formulation of higher level tasks. ARMS uses the ROS 2 action

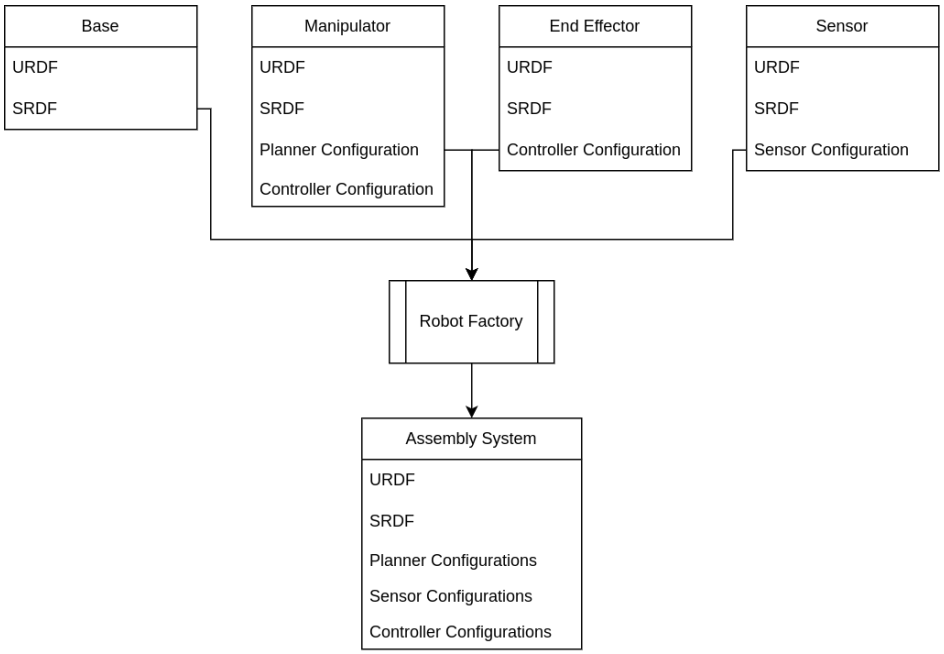


Fig. 1 The Robot Factory takes multiple specified component packages and produces a system description.

Variable Name	Required	Description
package	[x]	package name where URDF, Semantic Robot Description Format (SRDF), config, and mesh files are located
group name	[x]	the name by which this agent/segment will be referred
parent link	[x]	link to attach this element's base link to
filename	[ ]	name of xacro file in package name or URDF and package name or SRDF, needed if it differs from the package name
namespace	[ ]	ROS 2 namespace where MoveIt 2, robot action/status, and driver nodes are launched, used to contain a system if multiple are used
RViz	[ ]	Boolean dictating whether an RViz node should be launched
x	[ ]	x location of component's base link relative to parent link
y	[ ]	y location of component's base link relative to parent link
z	[ ]	z location of component's base link relative to parent link
roll	[ ]	roll of component's base link relative to parent link
pitch	[ ]	pitch of component's base link relative to parent link
yaw	[ ]	yaw of component's base link relative to parent link

Table 1 Variables that can be passed into single component definitions to the Robot Factory. Other variables passed in will be forwarded to that component's xacro scripts.

communication pattern, which allows for asynchronous request/response in addition to providing feedback to the requester.

1. Manipulators

Manipulators are the most complicated elements in the system. They are agents with multiple serial degrees of freedom and are typically robotic arms. Manipulators must be capable of collision-free trajectory generation for

themselves and attached end-effectors, sensors, and payloads. The manipulator action server has various actions that it can perform to allow for motion around the work area. The capabilities of the manipulator are discussed in the following paragraphs.

Move To Pose The Move To Pose action is used to move the manipulator and its end-effector to specific points in the workspace. It takes a target pose as input and will plan a path of the manipulator to reach the target pose with its end link. It can be specified to do Cartesian planning, which will linearly translate the end-effector to the specified position. This action will perform motion planning and execution.

Move Link To Pose The Move Link To Pose action allows a link on the manipulator to be specified along with a target pose for that link. This action will formulate and forward the request to MoveIt 2 for planning and execution.

Move To Joint The Move To Joint action allows a specified target joint state (i.e. all joint angles in the manipulator) to be specified. This action will formulate and forward the request to MoveIt 2 for planning and execution.

Move To Named Pose The Move To Named Pose will move the robot to a pre-defined joint state defined in the SRDF for that component.

Move To Joint The Move To Joint action allows a specified target joint state (i.e. all joint angles in the manipulator) to be specified.

Save Planning Scene This saves the Planning Scene state to a file to be used for recovery. All objects in the scene will be saved, with their locations, to the file. This file can be used to reload the Planning Scene on startup of ARMS. The reload feature is useful for tests that take multiple days since it allows persistence of the state between days of testing.

Move With Approach Arbitrary selection of a feasible joint state may position the manipulator in a poor configuration to allow further planned motion as demonstrated in Figures 2a and 2b. This problem has been seen most commonly during Pick and Install tasks (discussed in Section III.C) where motion is constrained during approach of an end-effector to a part or assembly.

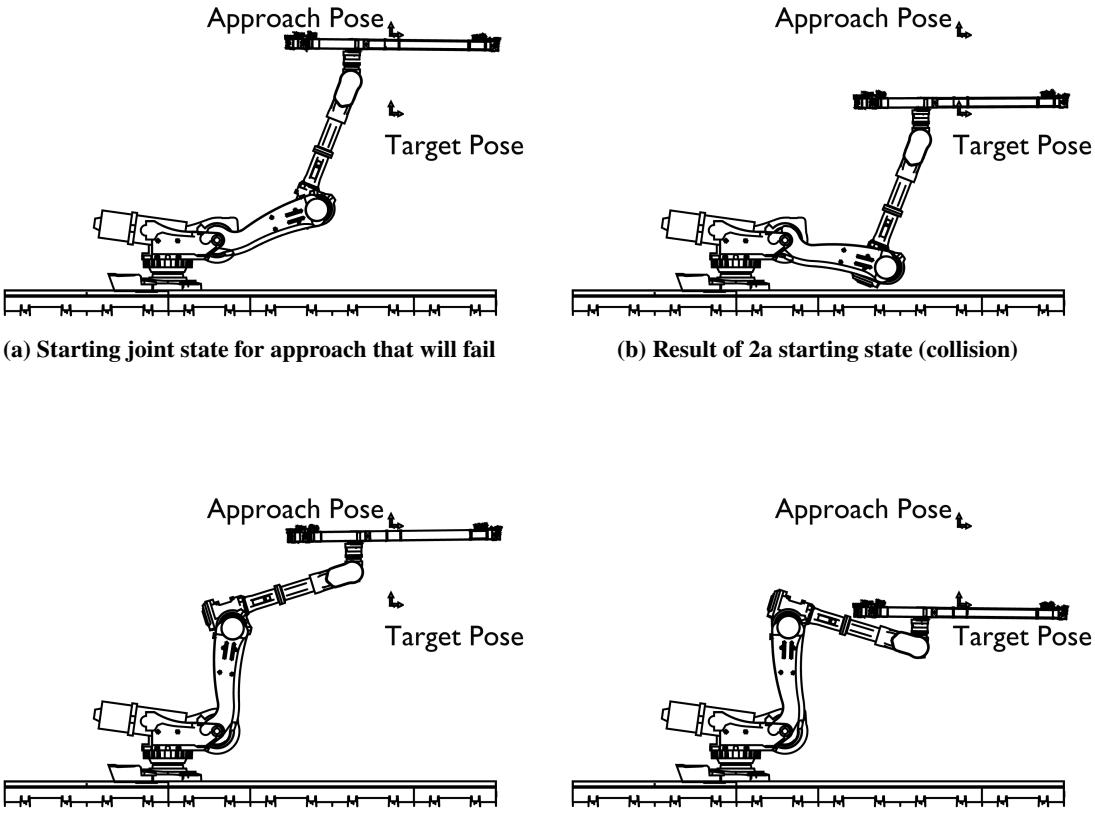
The solution used in ARMS to get around the problem of disjointed planning and infeasible states is to plan the Cartesian approach and non-Cartesian motion together. First, a joint state at the target position (e.g. pick truss 1 in Figure 4) is found, then a Cartesian path from the approach point (e.g. pre pick truss 1 in Figure 4) is planned. The ending joint state of that Cartesian path becomes the target joint state for the non-Cartesian planning. After execution of the non-Cartesian motion, ARMS can either execute the Cartesian path that was generated or perform visual servoing to the final position. A similar solution to this type of problem is solved by [7].

Servo This action will put the manipulator in a mode where it will listen for a joint velocity or Cartesian velocity (i.e. twist) commands. This command will be finished when a stop signal is sent to the manipulator.

Modifiers Modifiers can be sent with the requests to change how the requested action will be performed. Options are shown in Table 2.

Planning Scene and Padding Considerations The Planning Scene in MoveIt 2 is what makes the manipulator actions possible and it is where collision checking occurs for motion planning. The manipulator has actions available to interact with the Planning Scene, including adding or removing objects and attaching objects to the manipulator. Error between the real object or manipulator states and the representation in the Planning Scene can cause collisions in path execution. There are many sources for state error such as sensor error, discrete state distance sampling, and scene construction error. As such, an important element of the Planning Scene is the ability to add padding to objects. Padding is the process of adding a buffer around a mesh in order to minimize the potential for collisions.

The padding technique employed by the ROS 2 geometric shapes package, and consequently MoveIt 2, at the time of writing is not a comprehensive solution. It will achieve a desirable result on meshes that are mostly convex. When padding is applied to an object that has concavity the effect is scaling rather than padding, since holes in the model are



(a) Starting joint state for approach that will fail

(b) Result of 2a starting state (collision)

(c) Starting joint state for approach that can succeed

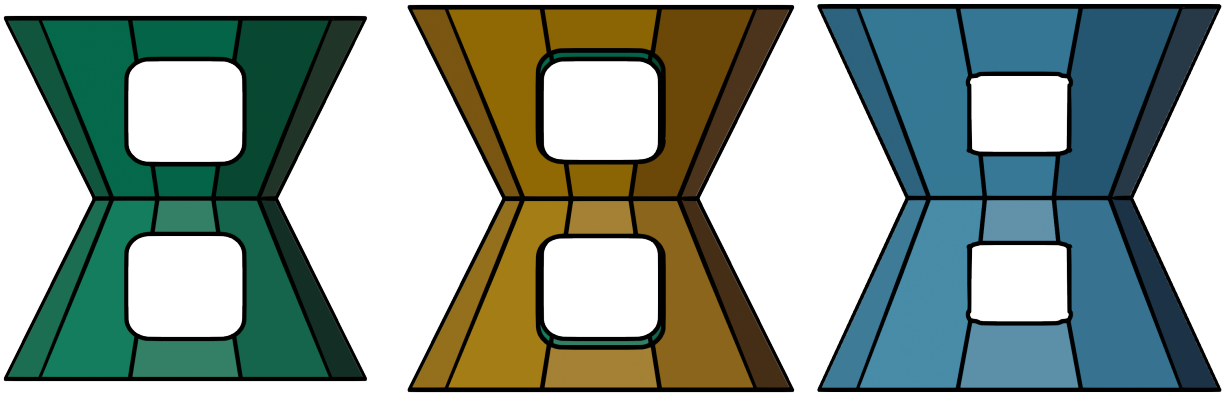
(d) Result of 2c starting state

Fig. 2 Selection of a gross motion goal state to allow for approach/retreat of the target is required to be able to complete the action.

Modifier	**Description**
Cartesian	Used when Cartesian motion from the current position to the target is required.
Ignore Collision With	A list of links that should have collisions ignored, adds all permutations of pairs in the list. This is useful for alignment operations where contacts are required.
Path Preview	Boolean specifying that the path should be visualized and approved by an operator before execution.
Plan Only	Boolean specifying to only plan and return the plan to the requester.

Table 2 All manipulator action modifiers that are available to send with a request. Note that not all modifiers apply to all request types.

also increased in size. This can lead to an inaccurate representation of the work area. A more desirable result is shown by the Blender Fatten operation [8]. The difference between the effect of these mesh operations can be seen in Figure 3. This shows that concave portions will get blown up into areas that are occupied by the original mesh, while the Fatten operation from Blender preserves all original occupied volumes and moves faces relative to their starting location.



(a) Unmodified shape **(b) Geometric Shapes 'padd' result** **(c) Blender 'Fatten' result**

Fig. 3 The difference between padding mesh operations overlaid on the original model.

Pre-padded meshes are used for collision checking on payloads that require it, along with the Ignore Collision With modifier.

2. End-effectors

End-effectors are agents that interact directly with assembly components. End-effectors grab parts for manipulators and mobile bases to move to another location. They may also be capable of performing an alignment operation between two parts and/or join two or more parts together. Additionally, they may be able to do the inverse of any of these actions to undo certain steps.

- Grab - Moves actuators to grab a part.
- Release - Moves actuators to release a part.
- Close Alignment - Moves actuators to close alignment devices to bring multiple parts into alignment.
- Open Alignment - Moves actuators to open alignment devices to release parts.
- Join - Joins multiple parts together.
- Unjoin - Disconnects part from the assembly.

C. High Level Tasks

ARMS uses ROS 2 action servers to implement high level tasks, which can coordinate and/or sequence agent actions or other tasks. These tasks servers will take a request message, parse the inputs, and populate the information that it will need to send to lower level tasks and agents. Then the task enters the execution sequence where it completes each step in the task. In a step, the task sends the requisite actions to agents or other lower-level tasks to complete this step. Each step can be a single request or multiple in parallel. Each step that is successfully completed is sent as feedback to the requester. If a step results in an error, that step number is sent as feedback to the requester which they can act on. In the use case discussed in Section IV.A, the mission executor will pause the script and inform the operator. The operator can perform some manual recovery process and resend the request specifying the step to resume from.

1. Pick

The Pick task coordinates a manipulator and end-effector together to pick an object from a location. The user needs to give a target frame, located in the work area, where the end-effector needs to be located to grab the object. Typically, there is a frame defined on the end-effector that will be coincident with the frame of the object being picked, when it can be grasped. Additionally, there may be a desired pre-pick and post-pick position. This will define the approach and

retreat vectors from the pick point as shown in Figure 4.

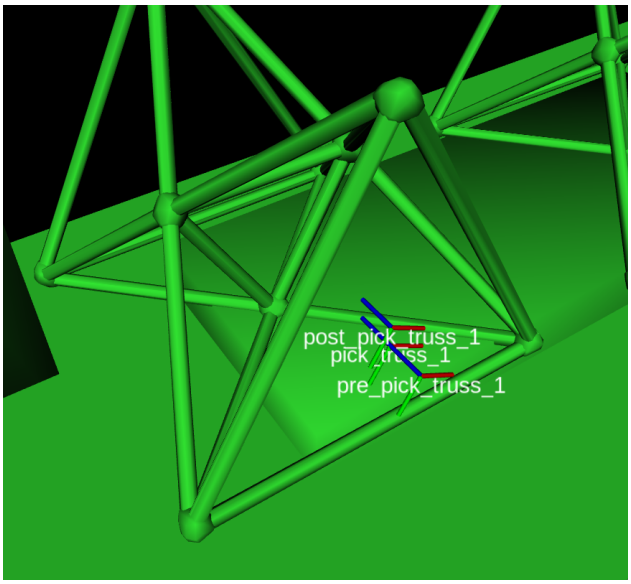


Fig. 4 TriTruss with target and approach/retreat positions specified. pick_truss_1 is the target position, pre_pick_truss 1 is the approach target and post_pick_truss_1 is the retreat target.

The user specifies the end-effector and manipulator agents that will be used for a pick task and the following steps will happen:

1) Release - Open end-effectors to prepare for tool approach of payload.
2) Move With Approach - Move the robot into the pre-defined position of the payload, where grasping is intended to occur.
3) Visual Servoing - A coordinated action between a manipulator and a pose estimation node, if specified.
   - Servo - Send a command to the manipulator to start listening for end-effector Cartesian velocity commands.
   - Pose Estimation - Send a command for the pose estimation service to start publishing the servoing commands.
4) Grab - Grab the part using the specified end-effector.
5) Attach Object - Attach the object to the manipulator and end-effector in the Planning Scene.
6) Move To Pose (Cartesian) - Move the manipulator away from the stowed location of the payload to a user defined position, to minimize potential collisions.

2. Install

The Install task coordinates a manipulator and end-effector together to move a held part into an assembly. The user needs to give a target frame, where the part needs to be located for the part to be installed. Typically, this frame will be coincident with the part at the pose where it can be joined to the assembly. Additionally, there may be a desired pre-install and post-install pose specified. This will define the approach and retreat vectors from the target point. The user specifies the end-effector and manipulator agents that will be used for an Install task and the following steps will happen:

1) Open Alignment - Open end-effectors to prepare for approach of end-effector to assembly
2) Move With Approach - Move the robot into the pre-defined position of the part's position in the assembly, where joining is intended to occur.
3) Visual Servoing - A coordinated action between a manipulator and a pose estimation node, if specified.
   - Servo - Send a command to the manipulator to start listening for end-effector Cartesian velocity commands.
   - Pose Estimation - Send a command for the pose estimation service to start publishing the servoing commands.
4) Force Servoing - A force based servoing that will allow .
   - Switch the manipulator from a joint trajectory controller to a force controller that will listen for desired force commands.
   - Force command of 0 will be streamed.

- Start the engagement of alignment features to pull the part into the assembly.

5) Join - End-effector starts the joining process specified for that tool.
6) Test Load - Apply a load to the assembled part to test the joint to determine if it was successful.
   - Switch the manipulator from a joint trajectory controller to a force controller that will listen for desired force commands.
   - Force commands will be streamed in the direction of the approach vector, of a magnitude specified by the request.
7) Release - Releases the end-effectors holding the part.
8) Close Alignment - Opens the alignment features on the end-effector.
9) Move To Pose (Cartesian) - Move the manipulator away from the stowed location of the payload to a user defined position, to minimize potential collisions.

D. Use

The use of ARMS involves setting up a launch file and configuration files. There are two types of configuration files to load information into the environment. The first describes named frames to load into the work area that will get put into ROS 2's Transform Tree. These will be points of interest in the work area, including places where parts are staged, or where they need to be inserted into an assembly. The second defines the meshes to be loaded into MoveIt 2's planning scene, including the work area mesh and any part file meshes. The launch file contains the robotic assembly system description. The description is passed into the Robot Factory, which handles launching the robotic assembly system components.

Once the robot components are configured and the assembly system is launched, action interfaces will accept commands from command line input, mission execution scripts or graphical user interfaces (GUIs). These commands include low-level commands such as moving end-effectors or manipulators to specific states, or high level commands such as installing a component. Figure 5 shows a communication diagram indicating the flow of information through the system.

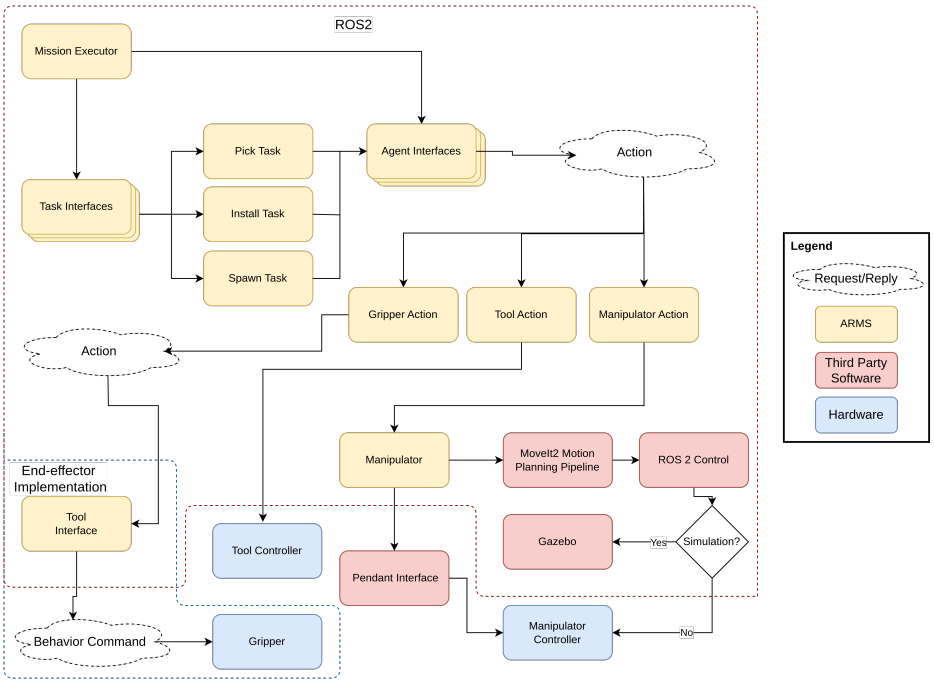


Fig. 5 The software node diagram showing how nodes communicate with each other.

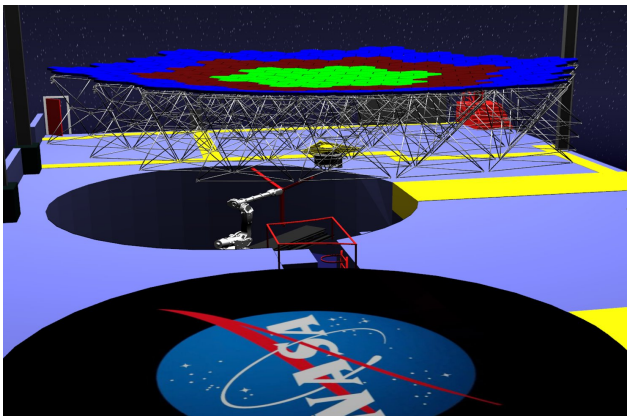


Fig. 6 The Precision Assembled Space Structure assembled in Gazebo simulation.

IV. Application

ARMS has been applied to two projects at LaRC, PASS and BORG. These projects have served as the drivers and testbeds for the software.

A. PASS

The PASS project seeks to assemble 37 TriTruss modules to create a backbone structure supporting a 20 meter telescope mirror as shown in Figure 6. An end-effector mechanism was designed and manufactured at LaRC to allow an industrial manipulator to grab the TriTrusses. The end-effector is capable of joining modules together via a screwdriver, and has actuators that are used to aid in fine alignment between multiple TriTruss modules for installation. The end-effector is mounted to a commercially available robot arm from Asea Brown Boveri, the ABB6650s-125 robotic manipulator.

Testing for PASS has taken three forms; simulation testing, 1 meter TriTruss testing, and 2.8 meter TriTruss testing. Simulation testing using ARMS was performed using Gazebo simulation software. Here, it was shown that the data and algorithms were routing as desired by Figure 5. After this, hardware testing was completed using 1 meter TriTrusses. In the 1 meter testing, a UR10e was used to manipulate TriTrusses, with a smaller end-effector compatible with the 1 meter TriTrusses. This testing highlighted the need for padding as discussed in Section III.B.1. Pre-padded meshes were used to reduce the likelihood of collisions during path execution. A three module assembly was completed and results are discussed in [9].

The only necessary changes between the 1 meter and 2.8 meter tests were the configurations that were fed into the Robot Factory. Specifically, the robotic assembly system components for the ABB6650s manipulator and 2.8 meter end-effector. The PASS Phase 1A Test was completed in January 2024 through the assembly of three TriTruss modules.

B. BORG

The BORG concept aims to test robotic operations for assembling a cube-like truss structure using a strut-by-strut approach. Individual struts, made from densely stackable angle irons with pre-drilled holes, are grabbed, positioned, and riveted together using custom end-effectors. The manipulator for this project is a UR10e industrial collaborative robot arm mounted on a linear track. The assembly process takes place on a turntable, which allows the robot to access different sides of the cube from optimal orientations.[2] Figure 7 shows the lab setup used for assembly testing.

Two lengths of struts are used, corresponding to the sides and diagonals of the cube, requiring two different end-effectors. Quick-change mechanisms enable the end-effectors to be swapped out, enhancing the robot's reusability. The robot is supplied with dispensers for struts and rivets, allowing it to reload as needed, which ensures a continuous and efficient assembly process. Once a cube is assembled, it can be pushed out and rotated, making space for the next cube to be assembled. This process allows for the three-dimensional expansion of the gigatruss structure.

ARMS simulation was employed from the outset to assess the feasibility of the laboratory setup. Initial simulation tests revealed that certain assembly poses would be very close to robot singularities, so the setup was modified to avoid such poses. In order to maximize the stiffness of intermediate states of assembly, the cube must be assembled in specific

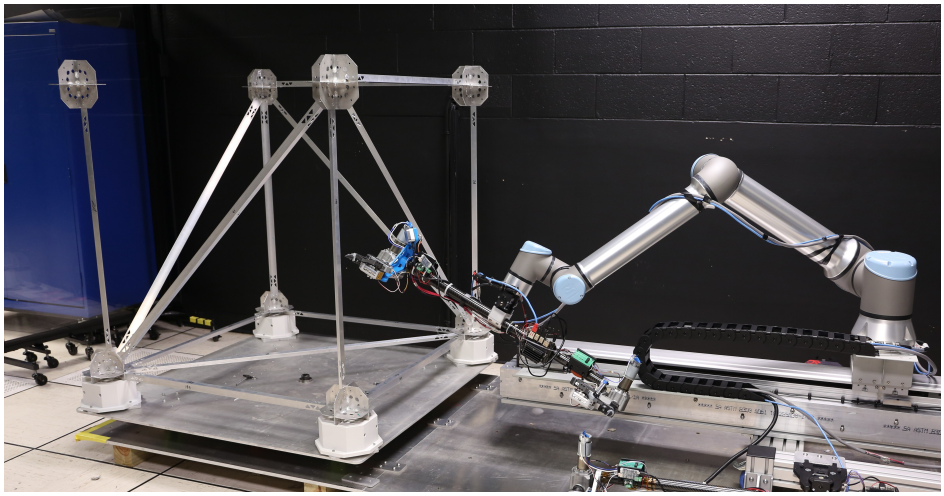


Fig. 7 Shown above is the robotic assembly system used for BORG. Pictured are a partially assembled cube (left) and the manipulator with end-effector (right).

sequences. Using the simulations, reachability issues and scenarios where the robot might become entrapped in the structure after assembling a strut were found early. The assembly sequence and grasp poses were changed, and several end-effector designs changes occurred to overcome these challenges. The ease of composing different robot models, coordinating assembly steps for multiple agents, ability to swap end-effectors, spawn, attach and detach objects in ARMS was very helpful in both exploration and implementation stages of the project. In October 2023, a hardware demonstration of the assembly of one tetrahedron of the cube was completed using ARMS.

C. Discussion

The Robot Factory has proven to be an invaluable asset for the autonomous assembly system projects. Its use has saved countless hours by preventing the need for reconfiguration of files when changes to the assembly system occurred. This has been particularly beneficial for both the PASS project and other similar initiatives. The ability to avoid unnecessary change to mission scripts between runs as streamlined the process significantly.

The Robot Factory has also enabled quick iterations for lab layout validation. The system can be configured easily, and simulation tests can be re-run without any major hurdles. This was particularly evident during the PASS project when mounting positions for the ABB6650s manipulator, candidate positions were set up in a launch file, as discussed in Section III.D, which expedited the process.

A simulation of the assembly for the assembly was run, and the number of modules that could feasibly be assembled was determined. This method allowed the evaluation of several configurations in just a few days. The efficiency and speed of this process underscores the benefits of the Robot Factory.

Hardware configuration changes are easily changed due to the Robot Factory and component library. For instance, during the 3-meter Phase 1A test for PASS, a force/torque sensor was added to the system during initial hardware testing. A package containing URDF, SRDF, and sensor configuration already existed for the sensor. Adding the sensor between the manipulator and end-effector require modifying only a few lines of the configuration.

The Robot Factory would also be helpful for other areas where reconfiguration is helpful, such as camera mount selection. The ease with which changes can be made to the hardware configuration is a testament to the versatility and utility of the Robot Factory.

Currently, the high level tasks perform some calculations and decisions based on the input of the request that was sent. The software begins sending requests to agents or other task servers. The execution and control flow used in ARMS is similar to a Behavior Tree, a structure that route abstract actions and are used in robotics [1, 10]. Adopting a Behavior Tree structure for control would open up possibilities for better user interfaces and introspection capabilities, further enhancing the functionality and making the system more user-friendly.

V. Conclusion

Future development efforts for ARMS include creation of GUIs for definition of robotic assembly systems, work area layout, and assembly tasks. These features would further streamline the process of configuring and deploying robotic systems for different applications. Additionally, future research will explore the integration of alternative perception, path planning and joint control algorithms. New high level tasks can be defined with ARMS for other ISAM problems, examples include cable routing, work area scanning, and multi-robot planning for object handoff.

ARMS provides a way to modularize and quickly integrate new hardware and algorithms with existing robotic system descriptions and software. Additionally, it provides ROS 2 action server interfaces for robotic manipulators and end-effectors. These manipulator and end-effector interfaces provide a base layer of autonomous actions to allow for higher level tasks as demonstrated in the Pick and Install tasks. The use of these elements in the PASS and BORG projects has shown the benefits of its modularity and composability in reducing integration time.

References

[1] Cassady, J., Mahlin, M., Kravets, E., Vaughan, M., and Rodgers, M., "Software Design for the Supervised Autonomous Assembly of a Tall Lunar Tower," *ASCEND*, 2023. https://doi.org/10.2514/6.2023-4682.

[2] Chapin, S., Everson, H., Chapin, W., Quartaro, A., and Komendera, E., "Built On-orbit Robotically assembled Gigatruss (BORG): A mixed assembly architecture trade study," *Frontiers in Robotics and AI*, Vol. 10, 2023. https://doi.org/10.3389/frobt.2023.1109131.

[3] Kelley, B. N., Cooper, J. R., Navarro, J. P., Vaughan, M., Waltz, W. J., Allen, B. D., Doggett, W., Avila, T. V., McQuarry, A. K., Shazly, S. A., Slick, R. M., and Williams, R. A., "Designing a Software Architecture for the Precision Assembly of Space Structures," *AIAA Journal*, Vol. 24, No. 11, 2022, pp. 1872, 1873. https://doi.org/10.2514/6.2022-2077.

[4] L., H. C., Grantham, C., Allen, C. L., Doggett, W. R., and Will, R. W., "Software design for automated assembly of truss structures," , 1992. URL https://ntrs.nasa.gov/citations/19920019132.

[5] Macenski, S., Foote, T., Gerkey, B., Lalancette, C., and Woodall, W., "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, Vol. 7, No. 66, 2022, p. eabm6074. https://doi.org/10.1126/scirobotics.abm6074, URL https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[6] Coleman, D., Şucan, I. A., Chitta, S., and Correll, N., "Reducing the Barrier to Entry of Complex Robotic Software: a MoveIt! Case Study," *Journal of Software Engineering for Robotics*, Vol. 5, 2014, pp. 3–16. https://doi.org/10.6092/JOSER_2014_05_01_p3.

[7] Görner, M., Haschke, R., Ritter, H., and Zhang, J., "MoveIt! Task Constructor for Task-Level Motion Planning," *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 190–196. https://doi.org/10.1109/ICRA.2019.8793898.

[8] Team, B. D., "The Blender 4.1 Manual," , 2024. Licensed under a CC-BY-SA v4.0.

[9] Cooper, J. R., Cresta, C. J., Avila, T. V., Rajaram, R., McQuarry, A. K., Martin, J., and Stohlman, O. R., "Test Results for Autonomous Assembly of Modular Space Structures," *ASCEND*, 2023. https://doi.org/10.2514/6.2023-4699.

[10] Ghzouli, R., Berger, T., Johnsen, E. B., Dragule, S., and Wąsowski, A., "Behavior trees in action: a study of robotics applications," *Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering*, ACM, 2020. https://doi.org/10.1145/3426425.3426942.