Software Design for the Supervised Autonomous Assembly of a Tall Lunar Tower

Jacob T. Cassady*, Matthew K. Mahlin†, Emma L. Kravets‡, Matthew P. Vaughan§
NASA Langley Research Center, Hampton, Virginia, 23666

Matthew R. Rodgers¶
Analytical Mechanics Associates, Hampton, Virginia, 23666

Carl J. Nicklas‖
Guardians of Honor, Washington, D.C., 20006

Tall towers enable a wide-ranging set of capabilities on the lunar surface including communication, navigation, surveillance, power generation, and more. The Tall Lunar Tower project at NASA Langley Research Center is focused on the design, modeling, fabrication, and testing of an engineering development unit to assemble a tall tower through supervised autonomous operations. In this paper, the software design for the supervised autonomous assembly of a tall lunar tower is presented. The paper includes a high-level description of the concept of operations, the agents, and an overview of the software architecture.

I. Nomenclature

AES	=	Advanced Exploration Systems
ARS	=	Assistant Robot System
BT	=	Behavior Tree
CAD	=	Computer-aided Design
$CCSDS$	=	Consultative Committee for Space Data Systems
CNN	=	Convolutional Neural Network
$ConOps$	=	Concept of Operations
CRS	=	Construction Robot System
$C-3PO$	=	Certifiable 3D Object Pose Estimation
EDU	=	Engineering Development Unit
EE	=	End Effector
GSW	=	Ground Software
GUI	=	Graphical User Interface
$ISRU$	=	In-situ Resource Utilization
$NASA$	=	National Aeronautics and Space Administration
RGB	=	Red, Green, and Blue
ROS	=	Robot Operating System
TLT	=	Tall Lunar Tower Project
TRL	=	Technology Readiness Level
UR	=	Universal Robotics
$V\&V$	=	Verification and Validation

*Computer Engineer, Flight Software Systems Branch, Mail Stop 064, AIAA Member.
†Research Engineer, Structural Mechanics and Concepts Branch, Mail Stop 190, AIAA Member.
‡Research Engineer, Autonomous Integrated Systems Research Branch, Mail Stop 233.
§Research Engineer, Autonomous Integrated Systems Research Branch, Mail Stop 233.
¶Flight Software Engineer, Flight Software Systems Branch, Mail Stop 064.
‖Student Trainee, Structural Mechanics and Concepts Branch, Mail Stop 190.

II. Introduction

Tall towers are important infrastructure for enabling a wide-ranging set of capabilities on the lunar surface including communication, navigation, surveillance, power generation, and more [1]. The Tall Lunar Tower (TLT) project at NASA Langley Research Center aims to design, fabricate, and test the supervised autonomous assembly of a tall tower by an Engineering Development Unit (EDU). The TLT project is developing a terrestrial demonstration with a payload mass analogous to a comparable lunar system. Power generation is being targeted in TLT's reference mission because towers 25-meters tall, located at specific areas near the Shackleton Crater of the lunar south pole, would be sufficient for achieving near-continuous power generation [2]. TLT's tower design has been shown to be favorable when compared to other tower types such as telescoping towers in the parameters of mass, packaging volume, and material volume [1]. Furthermore, the design presented here is compatible with future lunar in-situ resource utilization (ISRU) efforts while requiring a fraction of the mass presented in analysis of other ISRU tall lunar towers [1, 3].

This paper begins with a description of the concept of operations (ConOps) in Section III focusing on the EDU assembly system. Next, we present a high-level overview of the EDU system architecture including a description of the primary agents and their interfaces with each other in Section IV. Section V includes a description of the implementation with a focus on Robot Operating System (ROS) 2 and the frameworks leveraged from the ROS2 ecosystem. Subsection V.E describes the computer vision systems. Section VI discusses the current Technology Readiness Level (TRL) of key technologies related to the TLT EDU and the proposed TRL levels of the key technologies after a successful TLT EDU demonstration. The paper concludes in Section VII with a summary of the topics presented in this paper as well as some concluding remarks. Additional work has been done by the TLT Software team to facilitate the design and analysis of the tower itself [4] as well.

III. Concept of Operations

The ConOps presented here focuses on the assumptions of a terrestrial demonstration fo the EDU and the steps of the tower assembly. A description of the TLT EDU task specification and execution of the assembly is presented in subsection V.C.

The TLT project's EDU reference mission concept begins with the systems shown in Figure 1 on top a lander at the South Pole of the Moon. The lander is expected to have a leveling system, space for TLT's component dispensers, and provide power and communication to TLT's systems. For the reference mission, the tower would be built on top of the lander and remain there. TLT's reference mission also assumes a solar panel payload can be attached, stored within two bays of its tower, and deployed after the construction of the tower.

For the terrestrial demonstration of the EDU, construction will take place on a level surface in an indoor environment. Power to the system is provided by the facility and communication with ground support equipment will take place on a local network. The floor will be level and operators will have access to both hardware and software stops. Additional vision systems will monitor the construction to ensure the environment remains safe.

Tower construction begins by raising the payload using a lifter. A tower bay is then built under the payload by robotic manipulators with rivet guns on their end effector (EE). Part localization is performed using cameras on the EEs. Parts for the assembly are presented to the robotic manipulators using dispensers. Before placement, each part is inspected using cameras on the lifter. After a bay is assembled, the assembled set of bays is then raised again. The process is continued until the desired number of bays. At the end, the final bay is riveted to the foundation of the system.

To build a bay, a square cross frame of preassembled struts is first placed at the bottom of the EDU's lifter. Four vertical struts are added to connect the top and bottom cross frames. To add a vertical strut, the robotic manipulators load rivets and grab a vertical strut. The vertical strut is then added to connect the top and bottom cross frames using the end effector's rivet gun before another rivet is added to the other side of the vertical. Verification of proper strut placement is performed by the vision system. Next, four diagonal struts are added. The diagonal struts are longer than the verticals. To accommodate the differences in sizes between the two struts, the EE for each robot is designed to expand before placing diagonal struts and retract after. To add a diagonal strut a robotic manipulator first loads a rivet. It then extends the width of its EE to grab a diagonal strut. Finally, the manipulator places the diagonal strut using the EE's rivet gun and verifies with the vision system.

IV. System Architecture

The software for TLT's EDU is designed around three main agent types: Ground Software (GSW), Construction Robot System (CRS), and Assistant Robot System (ARS). The GSW, described in subsection IV.A, is the primary

interface to monitor telemetry from and send commands to the CRS. The CRS, described in subsection IV.B, is the central system responsible for receiving commands from GSW and giving commands to ARSs. The ARS, described in subsection IV.C, is a duplicated system. Each ARS is responsible for controlling a robotic manipulator, EE, and vision system. The interfaces between agents are shown in Figure 2. Descriptions of hardware interfaces and third-party ROS2 frameworks are described in detail throughout Section V. Figure 1 includes an image of the agents and their associated hardware.

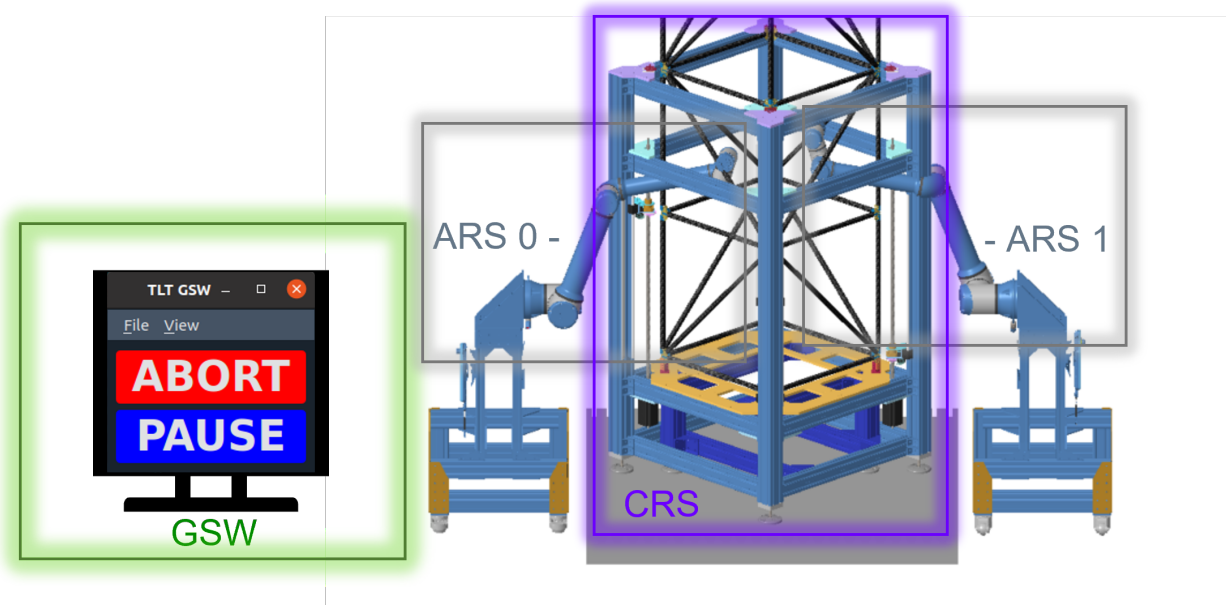

Fig. 1 TLT EDU

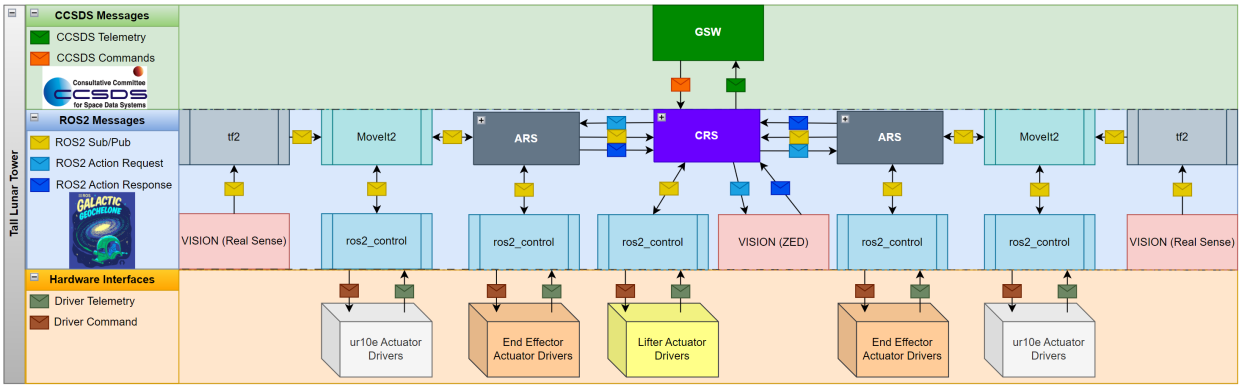

Fig. 2 Agent Interfaces

A. Ground Software

The GSW runs on a Linux laptop and is the primary interface to monitor telemetry from and send commands to the CRS. The GSW enables the supervised autonomous functions by providing an interface for mission operators to update the task execution of the CRS and ARS as well as alter the tower model maintained by the CRS. For TLT's EDU, the GSW is connected to the same network as the CRS and ARS but maintains the communication protocols found in space flight systems. The communication between the CRS and GSW is described in detail in subsection V.A. Figure 3 displays the interface between the GSW and CRS.

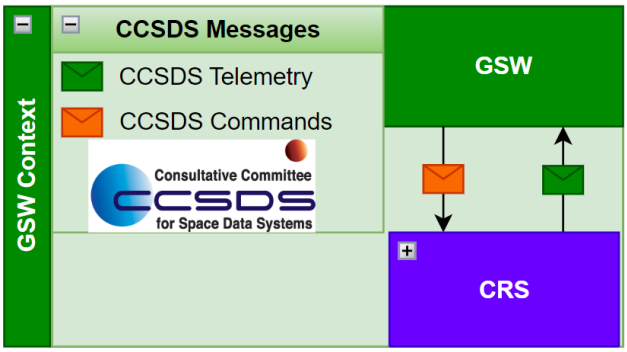

Fig. 3 Ground Software Interfaces

B. Construction Robot System

The CRS is the central software system. For TLT's EDU, the CRS runs on an NVIDIA Jetson Orin AGX Developer Kit using Ubuntu 20.04 operating system. The CRS's responsibilities include communciating with GSW, tracking the state of the tower, tracking the available components, commanding the ARSs, and controlling actuators on the Lifter. The CRS's communication with GSW is described in subsection V.A. Communication between the CRS and ARS is expanded upon in subsection V.C.1. The CRS's low-level actuator control is presented in subsections V.B.1, V.B.2, and V.B.3. Lastly, task scheduling and execution for the CRS is described in subsection V.C.2. Figure 4 shows the interfaces of the CRS with the GSW, ARSs, and Lifter.

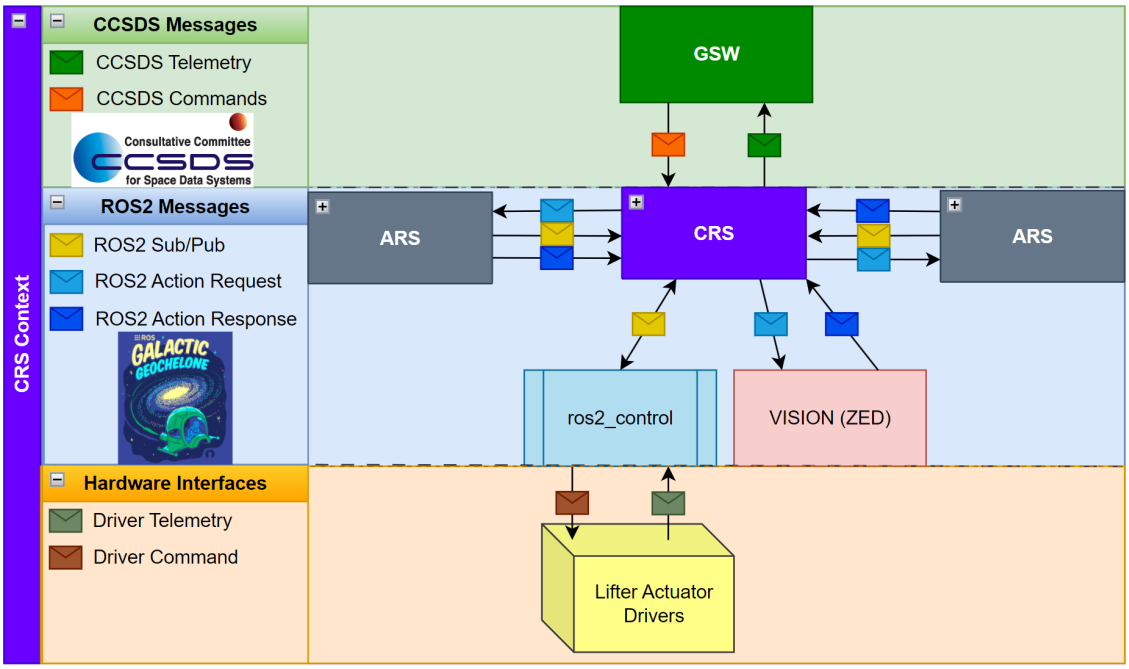

Fig. 4 Construction Robot System Interfaces

C. Assistant Robot System

The ARS is a repeated system. Each ARS controls a single robotic manipulator, EE and vision system to perform tasks designated by the CRS. The ARS acts as an action server for the CRS's commands. These commands are received through ROS2 as action requests and described in detail in subsection V.C.1. Task scheduling and execution for the ARS is described in subsection V.C.3. Strut positions are provided to the ARS through the vision system as described in subsection V.E. For TLT's EDU, the ARS runs on an NVIDIA Jetson Orin AGX Developer Kit using Ubuntu 20.04

operating system. The robotic manipulator for each ARS EDU is a UR10e made by Universal Robotics (UR). The EE was designed by the TLT Hardware team. Figure 5 shows the interfaces of the ARS with the manipulator, EE, and vision system.

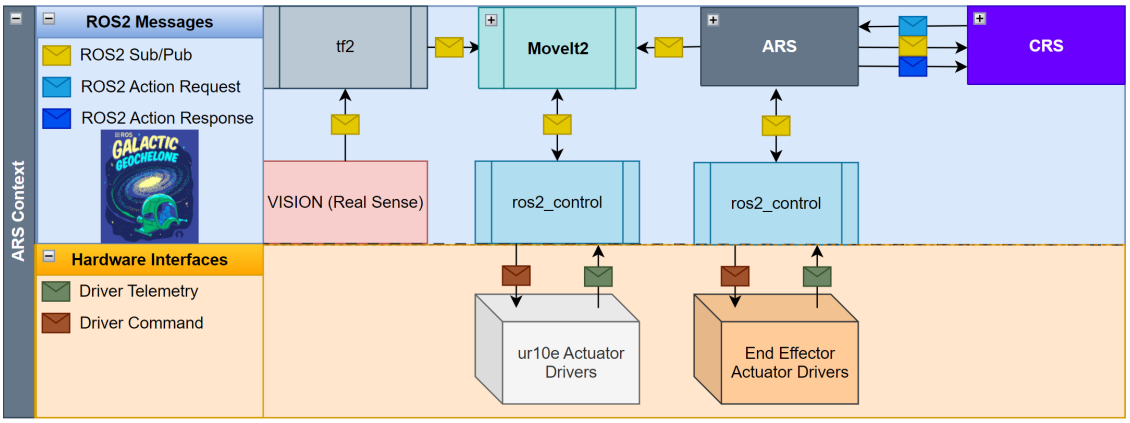

Fig. 5 Assistant Robot System Interfaces

V. System Overview

Software for the CRS and ARS agents was developed using ROS2 [5] as the underlying framework. ROS2 provides core infrastructure for distributed communication using the Data Distribution Service (DDS) [6] standard which enables dynamic, modular, and flexible robotic systems architectures through well-defined message interfaces. The DDS standard specifies a one-way publish/subscribe communication protocol over channels called topics, and ROS2 adds two-way client/server communications through services and action servers. Due to its popularity in industry and academia, ROS2 also has a large ecosystem of hardware drivers, algorithms, and other utilities to support complex robotic applications such as TLT.

The system overview section begins with a description of the ground software design and its Consultative Committee for Space Data Systems (CCSDS) communication protocol with the CRS in subsection V.A. Then we present the low-level control of actuators in the EE and Lifter using a framework from the ROS2 ecosystem, ros2_control, in subsection V.B before describing the Behavior Trees (BTs) used to perform task specification and execution in subsection V.C. Next, we provide a description of the motion planning of the manipulators associated with each ARS agent using MoveIt2, a framework within the ROS2 ecosystem in section V.D. The section ends with the vision system's algorithms for part detection and fault detection in subsection V.E.

A. Ground Software Design

The GSW's GUI was developed using the Qt framework. Qt is cross-platform software for creating GUIs and is available under multiple open-source licenses [7]. Qt provides native-looking interfaces with native capabilities and speed. Qt has history of usage in live ground software such as NASA's Low-Earth Orbit Flight Test of an Inflatable Decelerator (LOFTID) in 2022 and contains sufficient compatible extensions for live plotting and other data display formats. Trade studies were performed comparing our own GUI solution with that of open-source solutions such as Ball Aerospace's COSMOS but found these solutions to contain too much overhead for development of dynamic displays that fit TLT's operator needs.

TLT's GSW utilizes the CCSDS "Space Packet Protocol" for communication with the CRS. The CCSDS protocol was developed by an international management council for the purpose of standardizing transfer of space application data between any sending and receiving entities. Use of the CCSDS protocol enhances governmental and commercial interoperability, and reduces risk, development time, and project cost [8]. The CCSDS standardized header is recognizable by payload software, spacecraft software, ground software, and transmission software. Implementation of the CCSDS protocol is important to receive support and ensure compatibility with existing government and commercial space transmission and database systems.

Figure 6 shows the high-level design of the GSW. The GSW receives CCSDS telemetry from the CRS, then records

the raw packets and parsed/converted data fields to a local SQLite database [9]. Then, the GSW GUI performs queries on the database to display telemetry within Qt windows, and additional windows provide buttons to initiate commands which are sent to the CRS as CCSDS packets. The Qt GUI also provides an interface for operators to send commands with CCSDS packets.

There are six commands from the GSW to the CRS, as shown in Table 1. TLT Commands have three fields: a CCSDS header, the Command ID, and an optional command parameter. For safety purposes, the CRS must receive a Start command before beginning. Also, commands have been made to pause the system and stop or shutdown. The Resume command brings the system out of a pause state. There is a Set Blackboard Value command that is used to manipulate the state of agents' behavior trees and the CRS's model of the mission at runtime. Lastly, the Set Actuator State command can be used to drive subsystem actuators from the ground.

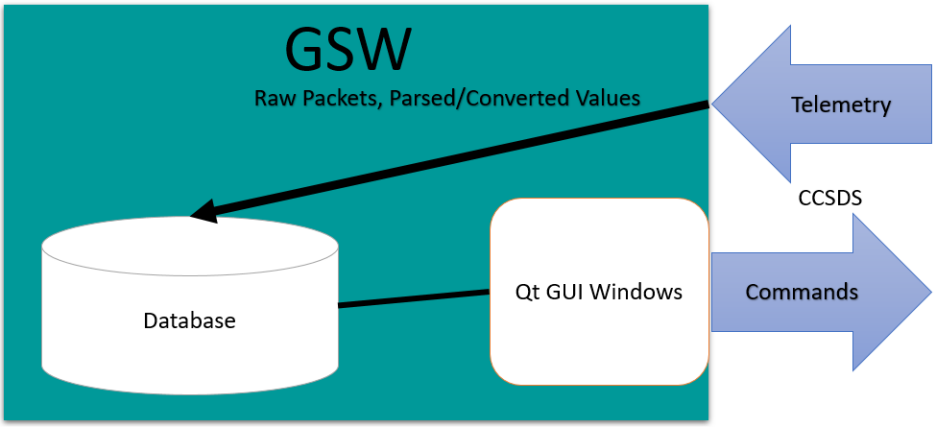

Fig. 6 Ground Software High-Level Design

Command	Description
Start	Start behavior tree execution.
Stop	Stop behavior tree. Shutdown.
Pause	Pause beahvior tree execution.
Resume	Resume behavior tree execution.
Set Blackboard Value	Set blackboard value.
Set Actuator State	Sets the state of an actuator.

Table 1 GSW Commands to CRS

B. Low Level Control

TLT's EDU includes six mechanisms that are controlled with actuators. The CRS and ARS use four types of actuators across their systems: actuators on UR Manipulators, Dynamixel servos, MightyZap linear actuators, and stepper motors. The low-level control of all of these actuators is handled through hardware drivers that expose interfaces to the ROS2 network using the ros2_control framework [10]. The ros2_control architecture defines interfaces for implementing and managing hardware components, sensor data, and controllers. The TLT software stack makes use of existing ros2_control interfaces for the UR manipulators and the Dynamixel servos. In addition to utilizing open source drivers, custom interfaces have been developed for the MightyZap actuators and lifter stepper motors.

The CRS controls four mechanisms as shown in Figure 7. Table 2 provides a legend for Figure 7. The lifter is used to increment the bays of the tower; low-level control of the lifter is described in subsection V.B.1. The Top Grippers hold the tower while a new bay is built; low-level control of the Top Grippers is described in subsection V.B.2. The Active Grippers hold the tower while the lifter is incrementing the bays; low-level control of the Active Grippers is

described in V.B.3. The CRS has two ZED camera mounts. Each ZED camera mount uses a single Dynamixel servo for positioning. Control of the Dynamixel is done using ros2_control's built in JointTrajectoryController through an open-source ros2_control hardware interface [11].

The ARS controls two mechanisms: the UR10e robot and the EE. The ros2_control implementation for UR robots is open-source and was created by Universal Robotics [12]. The EE is used for manipulating struts and retrieving/placing rivets. The low-level control of the EE is described in subsection V.B.4.

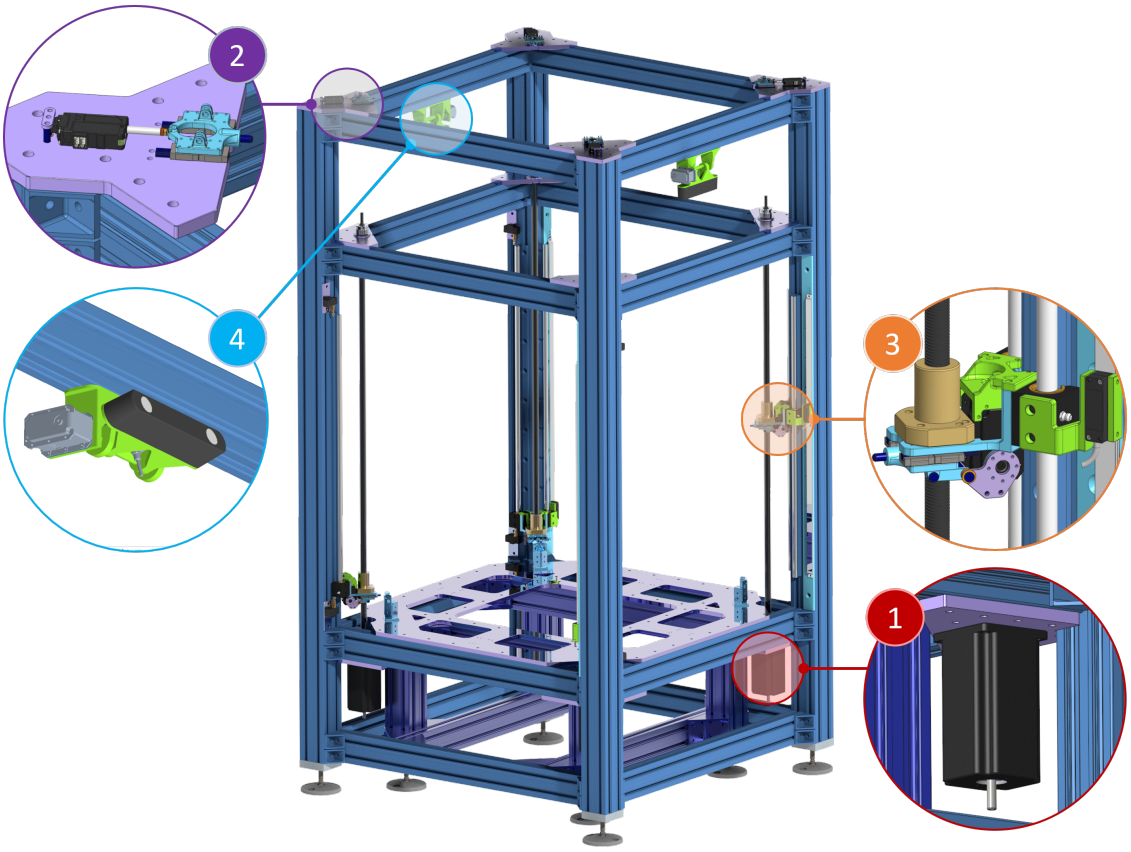

Fig. 7 CRS Mechanisms

Label	Mechanism
1	Lifter Stepper Motor
2	Top Gripper
3	Active Gripper
4	ZED Camera Mount

Table 2 CRS Mechanisms Legend

1. CRS Lifter

The CRS controls a lifting mechanism to index the tower during construction. The CRS Lifter's context, including components and interfaces, is shown in Figure 8. The electronics for the lifter include a Teensy 4.1 microcontroller, four linear encoders, four STR2 stepper motor drivers, four stepper motor actuators, eight limit switches, ten hardware push buttons, and a potentiometer. The linear encoders are used to report the position of the Active Grippers. The stepper motors, shown in Figure 7, and motor drivers are used to drive the position of the Active Grippers. The limit

switches, four upper and four lower, are used to stop the stepper motors from driving the Active Grippers too far one direction. The hardware push buttons are used to drive the stepper motors without the CRS control. Also, there are push buttons to drive each and all motors clockwise and push buttons to drive each and all motors counterclockwise. Lastly, a potentiometer is used to set the speed of the hardware push button motor control.

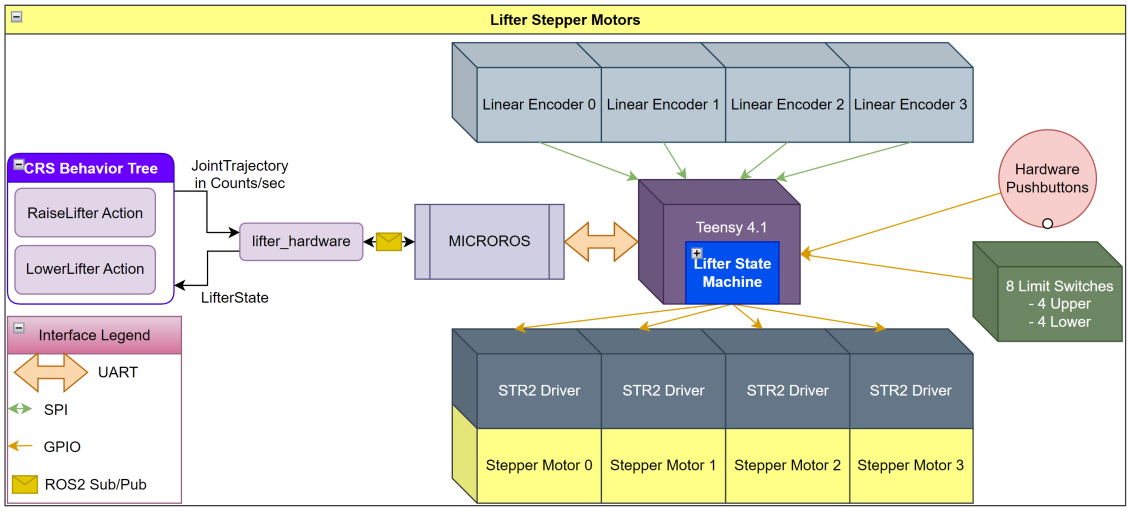

Fig. 8 CRS Lifter Context

The interface between the CRS BT and the firmware on the Teensy 4.1 is ros2_control and micro-ROS Arduino. Micro-ROS Arduino is an open-source library for bridging the gap between Arduino microcontrollers and larger processors in robotics applications [13]. The ROS2 messages that interface with the CRS Lifter are shown in Figure 9. The CRS Lifter subscribes to a joint trajectory topic on the ROS network. The joint trajectory contains a list of joint names and a list of corresponding points. The points are generated by a custom ros2_control controller and sent through a custom ros2_control hardware interface as a velocity with units of counts/sec. The CRS Lifter publishes a LifterState message with the state of the Lifter's state machine, the position of the linear encoders for each joint, and the state of each limit switch. The loop for the limit switches is closed inside of the state machine but published for monitoring purposes.

Fig. 9 CRS Lifter Interface

The state machine for the CRS Lifter is shown in Figure 41. Note the state machine diagram is simplified to convey the intention of the architecture. The actual implementation is threaded for each motor controller. ROS2 publication and subscription occurs in parallel to the motor control threads.

There are four CRS Lifter states: Initializing, Hardware Control, CRS Control, and Safe. The system begins in an Initializing state and enters a Hardware Control state when it is finished. The hardware control state is useful for testing purposes and is not used for the EDU assembly. While in the Hardware Control state, if a push button is pressed, the limit switch in the direction of the push button is checked. If the limit switch is not pressed, the motors are driven in the direction of the push button at a speed defined by the potentiometer's resistance. If no buttons are pressed, the CRS can request the lifter enter CRS Control state.

The CRS Control state is used for the CRS to autonomously control the position of the Active Grippers. If a button is pressed while the Lifter is in the CRS Control state, the motor velocities are zeroed and the system re-enters Hardware

Control state. While in CRS Control state, the system is looking for joint trajectories on the ROS network. Once a joint trajectory is found, the Lifter updates its motor velocities to match the joint trajectory. If the motor velocities are non-zero, and the limit switch isn't pressed in the direction of the motor velocities, the motors will drive in the direction and at the speed of the joint trajectory. There is an additional check on the time since a command has been received. If no command has been sent to the Lifter in twice the period of the CRS command frequency, it is assumed communication has been lost with the CRS and the motor velocities are zeroed out.

Lastly, the CRS can put the Lifter in a Safe state. While in a Safe state, the hardware buttons and joint trajectory signals are ignored. The Safe state was added for safety. Commands from the CRS can move the Lifter from a Safe state to either Hardware Control or CRS Control states. GSW can also tell the CRS to change the state of the Lifter using a Set Actuator State command.

2. CRS Top Grippers

The CRS controls push pins on the top of the CRS's frame, as shown in Figure 7, for holding the tower while a new bay is being built. For simplicity, TLT refers to these mechanisms as the "Top Grippers". The mechanism for the Top Gripper was developed by the TLT team. Each Top Gripper is designed around MightyZAP linear actuators.

Figure 10 shows the hardware and software interfaces of the Top Grippers with the CRS. A MightyZAP USB Board is used to facilitate the communication between the CRS's Nvidia Jetson Orin computer and the MightyZAPs. Each MightyZAP is connected to an IR-USB02 MightyZAP USB board using daisy chaining of both power and TTL serial communication. Each MightyZAP can still be communicated with separately. Control of the MightyZAPs is done using ros2_control's built in JointGroupPositionController through a custom ros2_control hardware interface.

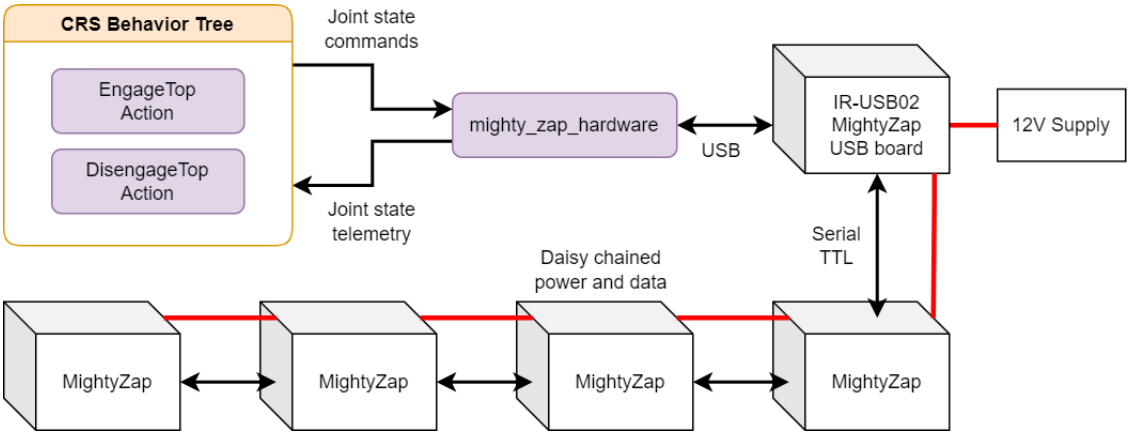

Fig. 10 CRS Top Gripper Interface

3. CRS Active Grippers

The CRS controls Active Grippers that are used to hold the tower while the CRS Lifter raises it. The Active Grippers are attached to lead screws, as shown in Figure 7. As the lead screws rotate, the Active Grippers move up and down. The mechanism for the Active Gripper was developed by the TLT team. Each Active Gripper is designed around Dynamixel MX-106T servos.

Figure 11 shows the hardware and software interfaces of the Active Grippers with the CRS. The CRS's Nvidia Jetson Orin computer is connected over USB to a U2D2 communication converter. The U2D2 then connects through TTL serial to a Dynamixel Power Hub Board. Each Dynamixel MX-106T is connected to the Dynamixel Power Hub Board using serial daisy chaining of both power and TTL serial communication. Each Dynamixel can still be communicated with separately. Control of the Dynamixels is done using ros2_control's built in JointTrajectoryController through an open-source ros2_control hardware interface [11].

4. ARS End Effector

The ARS controls an End Effector attached to the UR10e robotic manipulator. The EE was developed by the TLT team. The ARS EE is used to pick up parts and place them using rivet guns mounted on each side. The ARS EE also

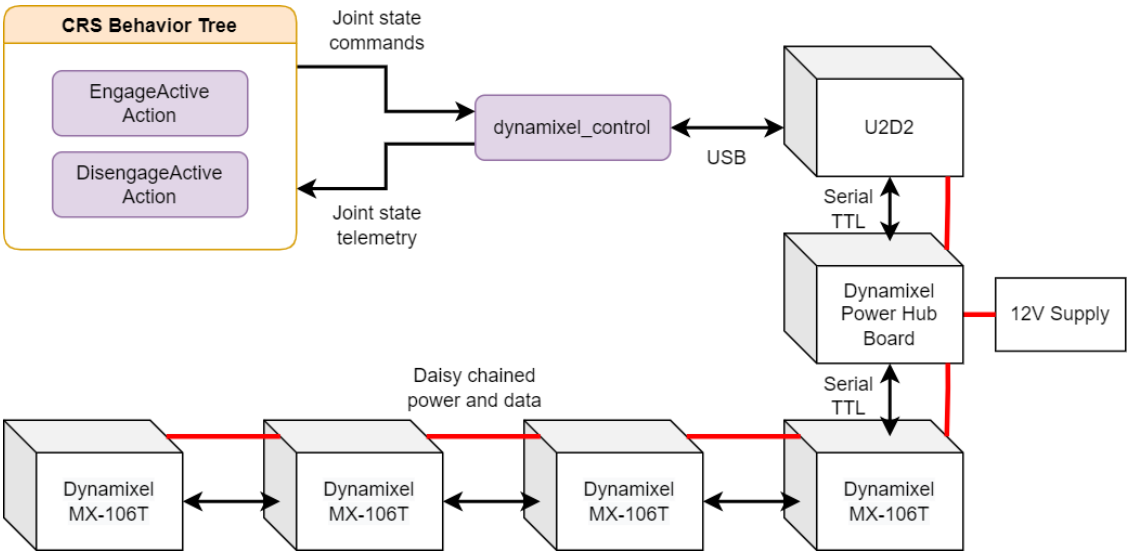

Fig. 11 CRS Active Gripper Interface

has the capability to extend or retract the position of the EE grippers to pick up struts of different sizes. Verification of EE grasps can be done using feedback from the EE servos.

The EE electronics include a Dynamixel Power Hub Board, three Dynamixel MX-106T servos, two rivet guns, and a limit switch. Figure 12 shows the hardware and software interfaces of the EE with the ARS. The ARS's Nvidia Jetson Orin computer is connected over USB to a U2D2 communication converter. The U2D2 then connects through TTL serial to a Dynamixel Power Hub Board. Each Dynamixel MX-106T is connected to the Dynamixel Power Hub Board using serial daisy chaining of both power and TTL serial communication. Each Dynamixel can still be communicated with separately.

Control of the Dynamixels that are used to grasp objects is done using ros2_control's built in JointTrajectoryController through an open-source ros2_control hardware interface [11]. Control of a single Dynamixel that is used to extend or contract the EE is done using ros2_control's built in JointGroupPositionController through the same hardware interface [11]. Modifications had to be made to the dynamixel driver to support series communication with dynamixels in different control modes. The ARS can trigger the rivet guns using a GPIO out pin and can check the state of the limit switch using GPIO in pin. A limit switch was added to the ARS EE to zero out the position of the Dynamixels. Limit switches are needed because the Dynamixels only retain the absolute position of their current rotation upon power reset. A calibration step is therefore required.

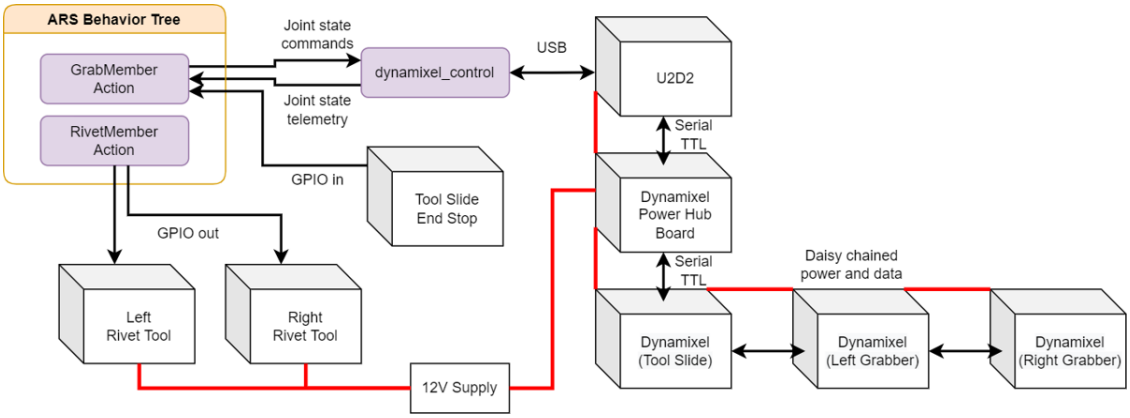

Fig. 12 ARS EE Interface

C. Task Specification and Execution

Task specification and execution for the CRS and ARS is done using BTs. Bts were chosen due to their modularity and reactivity [14]. Additionally, being able to validate branches of trees from the bottom up is a power testing and integration strategy. Furthermore, a trade study was done between a hierarchical state machine and a BT implementation of our system, and we found the BT implementation required fewer lines of code after leveraging 3rd party libraries. Communication between the BT of CRS and ARS is described in subsection V.C.1.

The py_trees library is an open-source python implementation of a BT model [15]. The library includes sequence, selector (also known as fallback), and parallel nodes as well as a general node implementation called a behavior. Additionally, py_trees uses a “blackboard” concept to store data shared across nodes throughout the BT. The EDU’s py_trees implementation has been extended by subclassing Behavior to implement Action and Condition nodes to match the node types described in Colledanchise and Ögren’s book, "Behavior Trees in Robotics and AI" [14]. These node types and their corresponding symbols used throughout this paper are shown below in Table 3. Additional encode/decode methods were added to describe the status of a BT as a bounded set of bytes for transmission purposes. The CRS and ARS are both designed around the concept of BTs using our modified version of py_trees. The BT models for the CRS and ARS are described in detail in subsections V.C.2 and V.C.3 respectively.

Node type	Symbol	Succeeds	Fails	Running
Fallback	?	If one child succeeds	If all children fail	If one child returns Running
Sequence	-->	If all children succeed	If one child fails	If one child returns Running
Action	Action	Upon completion	If impossible to complete	During Completion
Condition	condition	If true	If false	Never

Table 3 BT Node Types

1. ROS2 Action Client/Server Relationship

Commands from the CRS to the ARS are done through ROS2 Actions. Actions were chosen because they work well when the response to a message may occur in a significant amount of time [16]. Actions in ROS2 are designed around action servers and action clients. Action servers provide an interface to an action. Action clients send goals to an action server. Actions in ROS2 have three components: request, feedback, and result. Action clients use request to provide a goal to action servers. Action clients monitor the progress of requests through feedback sent by action servers. Action servers send the outcome of a goal to action clients using result.

In TLT’s case, the CRS acts an action client and each ARS acts as an action server. The CRS sends command to the ARS in the form of action requests. The ARS provides action feedback to the CRS during execution. When the ARS finishes, it provides a result to the CRS. The result can be a success or a failure. The CRS will update the model of the mission and its behavior tree accordingly. Interactions between the CRS and ARS are contained in the “Build Bay” and “Add Foundation Rivets” branches described in the following subsection.

2. Construction Robot System Behavior Tree

The CRS’s software architecture was designed around BTs. Figure 13 includes a graphical representation of the BT that governs the CRS. Table 3 includes a legend of nodes and their matching symbols used in Figure 13.

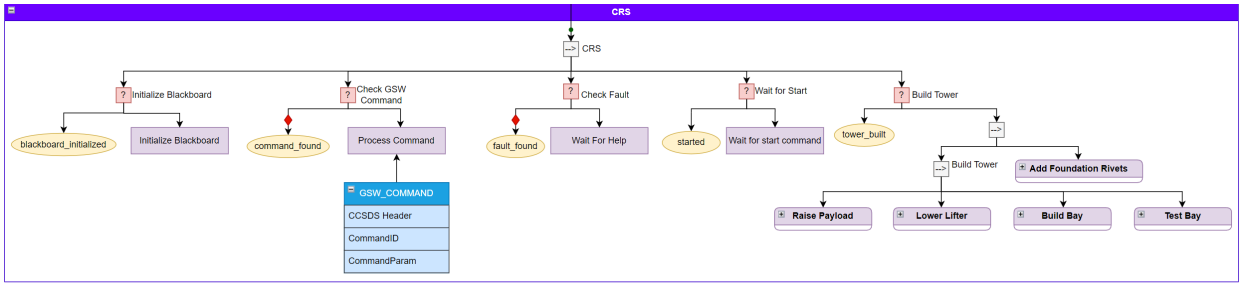

Fig. 13 CRS BT

The CRS's BT begins by initializing a blackboard of data, which includes loading mission definition files. Next, it checks for commands from ground software and processes a command if one is available. The CRS then checks to see if a fault has been found and waits for help if a fault has been found. A started condition has been added for pausing the system and begins false for safety purposes. If the previous conditions are nominal, the build tower sequence begins.

Building the tower begins by first raising the payload. The subtree for raising the payload can be found in Figure 18. The first step to raise the payload is to engage the Active Grippers and then disengage the Top Grippers. The CRS then places the Lifter into a CRS control state and raises the lifter. The next step is lowering the lifter so the next bay can be built. The subtree for lowering the lifter can be found in Figure 19. The "Lower Lifter" substree begins with engaging the Top Grippers before disengaging the Active Grippers. The CRS then places the Lifter into a CRS control state and lowers the lifter.

Now that the payload has been indexed up, a new bay is ready to be built. Before building a bay the Top Grippers are engaged and the Active Grippers are disengaged and at a lowered state. The subtree for building a new bay can be found in Figure 20. The build bay behavior has a sequence with five large subtrees for each step of the assembly. The first step is adding a cross frame at the bottom of the lifter. Next, each manipulator adds far and near vertical struts to the tower. Finally, each manipulator adds diagonal struts to the tower.

The subtree for adding a cross frame can be found in Figure 21. The "Add Cross Frame" subtree begins by sending a cross frame request to an ARS and waiting for an acknowledgement. Details for the ARS retrieving the cross frame is presented in subsection V.C.3. The CRS then performs an inspection check of the cross frame and waits for the ARS to finish placing the cross frame. The final step for adding the cross frame is engaging the Active Grippers to hold the cross frame in place.

The subtree for performing the inspection check can be found in Figure 22. The "Inspection Check" substree starts by requesting an inspection report from the Vision system. The vision system uses the ZED camera mounted on the lifter to check for faults in the part. Once the vision system returns a result to the CRS, the CRS then forwards the results of the inspection to the ARS. More details on the vision system are presented in subsection V.E.

Subtrees for adding vertical struts follow two steps: adding and affixing. The two steps are needed because each vertical strut requires two sets of rivets to be placed. The first step, adding, includes getting a rivet, getting a part, performing inspection, placing the part and installing the rivet. The second step, affixing, includes getting a rivet and performing the rivet. The subtree for adding a far vertical can be found in Figure 23. Adding a far vertical begins by sending an Add Member Request to ARS X with parameters describing the far vertical. The CRS then performs an inspection check and waits for the ARS to finish the Add Member Request. When ARS X finishes, ARS Y is then sent an Affix Member Request with parameters describing the far vertical ARS X just placed. Using both manipulators for placing far verticals is required due to the reach limitations of the manipulators used for the EDU. The far vertical subtree is repeated for (x=0, y=1) and (x=1, y=0) to add the far vertical for each manipulator. The subtree for adding the near vertical, found in Figure 24, is like that of adding the far vertical except the same ARS that adds the strut does the affixing.

The subtree for adding a diagonal can be found in Figure 25. Adding a diagonal begins by sending an Add Member Request with parameters describing the diagonal to an ARS. The CRS then performs an inspection and waits for the ARS to finish. The subtree for adding the other diagonal is the same with different parameters supplied to the Add Member Request.

After building a bay, the CRS performs a compression test on the bay to ensure structural stability. The subtree for testing a bay can be found in Figure 26. The first step to testing a bay is engaging both the Top Gripper and the Active Grippers. The CRS then places the Lifter into a CRS control state and commands the lifter to supply a small amount of

compression force. The CRS then rotates the ZED camera inward and uses the computer vision system to look for faults in the alignment. After bay inspection, the ZED camera rotates outward again to prepare for future inspection checks.

The final step of the CRS BT is the Add Foundation Rivets behavior shown in Figure 27. The "Add Foundation Rivets" subtree begins with sending a Foundation Rivet Request to an ARS and waiting for the ARS to complete. Once the foundation has been riveted, tower_built becomes true and the system enters an idle state. It is important to note that some conditions such as tower_built are driven by the model of the tower and the mission definition while others are driven by sensor feedback such as top_gripper_engaged. TLT's design BT philosophy was to use conditions driven by sensor feedback as much as possible.

3. Assistant Robot System Behavior Tree

The ARS's software architecture was also designed around BTs. Figure 14 includes a graphical representation of a representative ARS BT. Table 3 includes a legend of nodes and their matching symbols used in Figure 14.

The ARS's BT begins by waiting for a command from the CRS. The commands come in the form of an Action Request or as forwarded ground commands including Start, Stop, Pause, and Resume. The Action Request names match branches in the CRS and including Add Cross Frame Request, Add Member Request, Affix Member Request, and Foundation Rivet Request. When a request is received, the behavior tree is updated with a matching branch: Add Cross Frame, Add Member, etc. The ARS then sends an acknolwedgement to the CRS and begins to carry out the updated branch. After the branch is complete, the ARS reports the result to the CRS in the form of a matching result: Add Cross Frame Result, Add Member Result, etc.

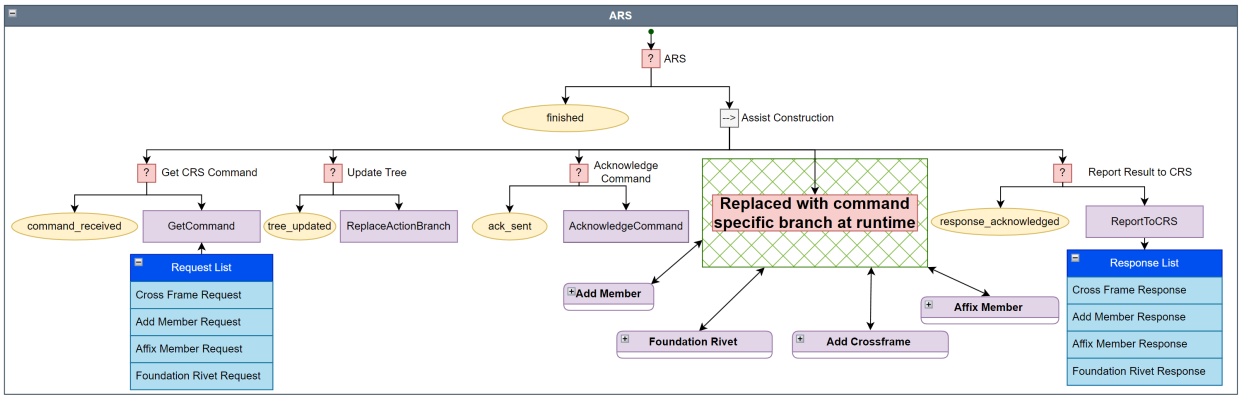

Fig. 14 ARS BT

The ARS subtrees follow similar patterns. The ARS subtrees often have a coarse approach to a dispenser or placement location that is hard coded. Then the ARS subtrees have a fine approach using either an April tag and OpenCV [17], in the case of a dispenser, or the neural network described in subsection V.E, in the case of a part placement. After the fine approach the ARS subtrees have an action, such as acquiring an object or placing an object. After the action, the ARS subtrees perform verification of the action which is done through actuator feedback when possible or by the vision system.

The subtree for adding a cross frame can be found in Figure 28. The Add cross frame subtree has three parts: getting a cross frame, inspecting the cross frame, and placing the cross frame. The subtree for getting a cross frame can be found in Figure 29. Getting the cross frame requires a coarse approach to the cross frame dispenser, then a fine approach using an April tag and OpenCV. Next, the ARS grasps the cross frame and verification is done using feedback from the Dynamixel servos on the EE. The subtree for inspecting a cross frame can be found in Figure 30. The first step for ARS cross frame inspection is moving to a hard coded cross frame inspection position. The ARS then requests an inspection report from the CRS and discards the part if the inspection does not pass. If the inspection passes, the ARS continues to place the cross frame following the subtree shown in Figure 31. Placing a cross frame is done with a hard coded coarse approach before using the neural network for fine alignment. The placement of the cross frame is verified using the ARS camera.

The Add Member subtree is used for both vertical and diagonal struts but with different parameters in the request. The subtree for adding a member can be found in Figure 32. The Add Member subtree has five parts: loading a rivet, grabbing a member, inspecting the member, placing the member, and riveting the member. The subtree for loading a

rivet can be found in Figure 33. There is an initial coarse approach to the dispenser that is hard coded before using an April tag and OpenCV for fine alignment. Next a rivet is loaded, and the camera system is used to verify the rivet is loaded properly. The subtree for grabbing a member can be found in Figure 34. Grabbing a member is done with another coarse approach to a dispenser using a hard coded value and using an April tag and OpenCV for fine alignment. The member is then grasped, and acquisition of the member is confirmed using feedback from the EE. After acquisition of the part, the ARS performs an inspection check of the member with the CRS, as shown in Figure 35, that is like that of the cross frame. Once the inspection check is done the ARS places the member, following the subtree shown in Figure 36. The ARS then rivets the member following the subtree shown in Figure 37. Confirmation of a well placed rivet is not easily attainable and has to be supplemented with additional vision systems outside of the EDU. Further discussion of risk acceptance related to the rivet joining will occur in Section VII.

The subtree for affixing a member can be found in Figure 39. The "Affix Member" substree entails loading a rivet and placing a rivet, as described above. The second rivet placement is required for vertical struts because they require two sets of rivets. The subtree for placing a rivet can be found in Figure 40. The subtree includes a coarse approach to the target member before following the same rivet member subtree described in the previous paragraph. The subtree for riveting the tower to the foundation is like that of affixing a member and can be found in Figure 38.

D. Motion Planning

TLT uses the MoveIt 2 motion planning framework to generate collision-free paths for the robotic manipulators [18]. MoveIt 2 integrates motion planning, collision detection, trajectory execution, and environmental modeling in a highly configurable ROS2 interface. Figure 15 shows the MoveIt2 pipeline and the interfaces with the ARS, vision, and UR10e manipulator. Support for MoveIt 2 motion planning for the UR10e robot is supported by the open-source UR ROS2 Driver [12] made by Universal Robotics. ARS's BT interacts with MoveIt2 framework by implementing semantically relevant low-level behaviors such as "Approach Dispenser", seen in Figure 34, or "Move To Member Inspection", seen in Figure 35, as sequences of motion planning requests and executions through MoveIt 2.

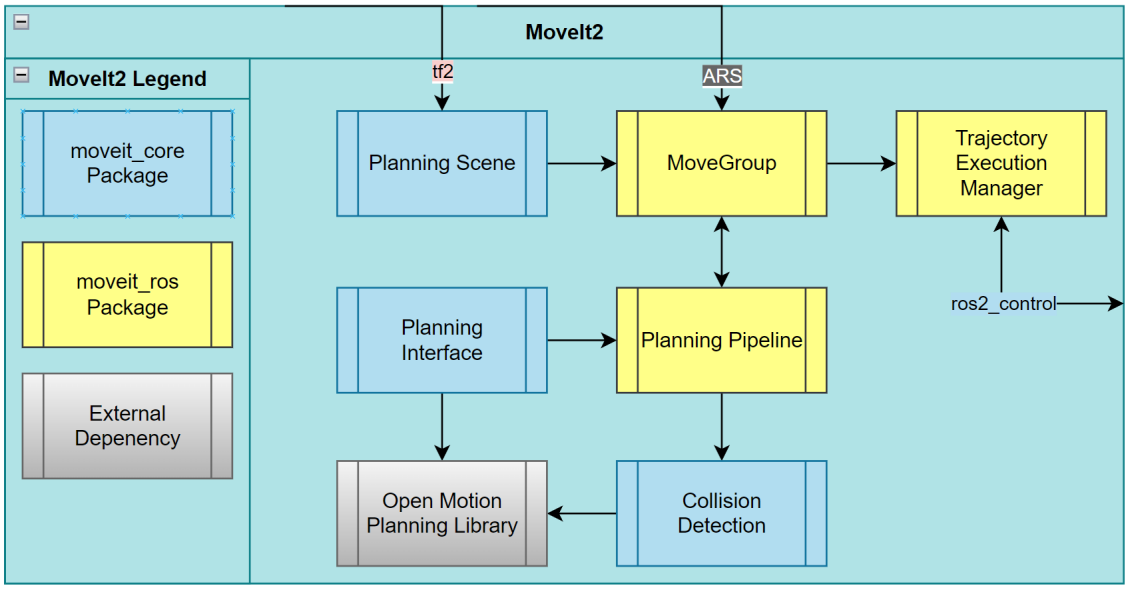

Fig. 15 ARS Planning and Manipulation

E. Vision

The vision system is responsible for accurate pose estimation for the purpose of part and fault detection. The system relies on two different camera systems. Two ZED 2i cameras are mounted to the top of the Lifter, and are used for inspecting parts and checking the truss bay assembly. On each ARS EE, an Intel RealSense D405 camera is mounted at each end, and these are used for part alignment checks. The context for each camera and the vision system is described in Figures 16 and 17.

To ensure a reliable and accurate pose estimate, the vision system utilizes an approach based on Certifiable 3D Object Pose Estimation (C-3PO) [19]. First, the camera's red, green, and blue (RGB) channel is used as input to a Mask-RCNN instance segmentation model [20], to obtain instance masks for each object in the camera scene. The instance mask is then applied to the RGB and depth channels, and the output is converted to a point cloud. The point cloud serves as input to the C-3PO model, which returns the detected point cloud, rotation and translation vectors, and a certificate of correctness for the pose.

C-3PO uses a self-supervised training approach, wherein a supervised model is trained on synthetically generated point clouds and the object's computer-aided design (CAD) model and annotated semantic key points, and the resulting weights are provided to a self-supervised model which trains on real data. The self-supervised model applies a corrector to overcome the sim-to-real gap between the models, correcting the key point detections, and providing a certificate of correctness for the pose [19]. For a pose to be considered certifiably correct, it must meet both an observably correct requirement and a non-degeneracy requirement. For observable correctness, the 90th percentile of the Chamfer distances between the obtained point cloud points and the ground truth CAD points must be within a set threshold, which is specific to each object, and for non-degeneracy, a minimum set of key points must be included in the point cloud such that enough of the object is visible to make an accurate determination about its pose [19].

The vision system is used in several parts of the system. First, the ARS uses fiducial tags to assist in locating the part dispenser, where it collects a part. Each part is then presented to a ZED camera mounted to the Lifter. The CRS requests an inspection of the part from the vision system. Here, the vision system relies on the segmentation and pose models to determine whether the part is intact. For example, for each strut, the vision system expects to find a strut tube and two end fittings, each in the correct locations. When a passing inspection report is returned to the CRS, the ARS can continue construction by placing the part.

As parts are added to the assembly, the EE cameras are used to obtain poses to determine part alignment before the parts are riveted. The poses obtained in the camera frame are converted to the planning scene frame using ROS2 TF2 transforms. TF2 is a ROS2 package which allows for easier coordinate transformations, as it maintains a structure tree of the system, such that the position of an object relative to any other object can be quickly computed [21]. These poses are then passed to the ARS, which can confirm that the parts are properly aligned relative to each other, and proceed with construction.

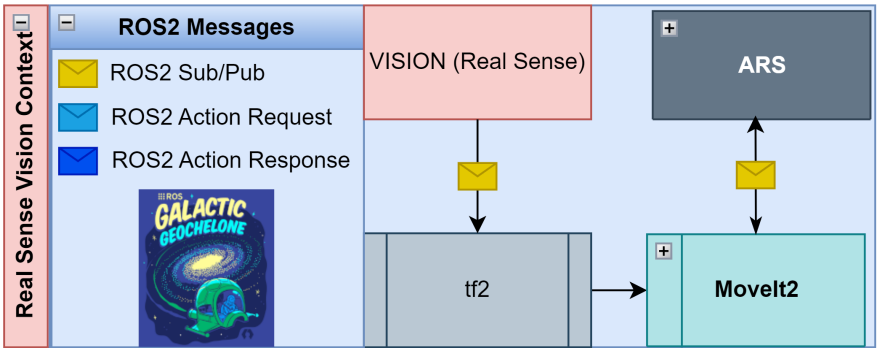

Fig. 16 RealSense Vision Context

Once a truss bay is completed, the CRS requests an inspection of the bay from the vision system. The ZED cameras are rotated inward, and each obtains the poses of the opposite four corner joints. If the corner joints are in the correct positions relative to each other, the truss bay is assembled correctly, and construction can proceed to the next bay.

The vision system was trained in multiple phases. First, the CAD models were set up in random poses with appropriate textures and random lighting conditions, and 5,000 RGB and depth images were generated at random camera positions, along with ground truth instance masks and poses. The RGB images and masks were used to train Mask R-CNN. Keypoints for each CAD model were annotated manually. The supervised portion of C-3PO was trained using only the CAD models, with 50,000 synthetically generated point clouds in the training set and 10,000 in the validation set. Next, the ground truth masks and pose data were applied to the RGB and depth images, in order to generate ground truth point clouds for the self supervised model. The self supervised model was trained for each object until either the model reached a minimum of 95% certifiable poses, or learning stopped improving.

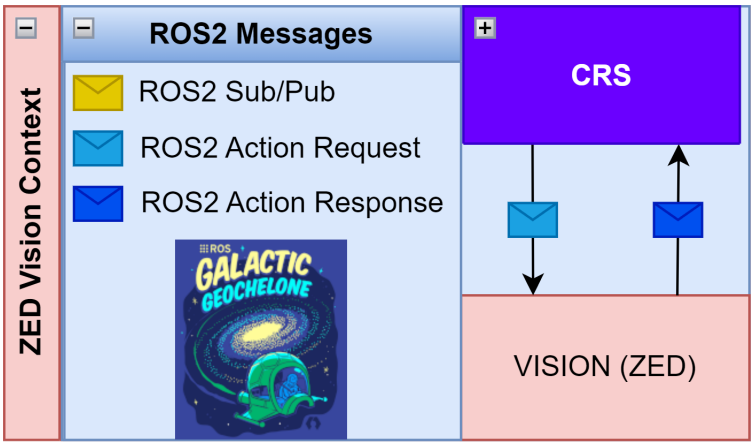

Fig. 17 ZED Vision Context

VI. Technology Readiness Level

TRL is a measurement system used to assess the maturity level of a particular technology. NASA uses a TRL system with levels 1 through 9; each TRL has a definition, description, and exit criteria [22]. TLT has identified four key technologies that may be advanced through a successful demonstration of the EDU: the application of behavior trees to in-space assembly, the application of ROS2 to in-space assembly, the application of machine learning based vision systems in in-space assembly, and Automated Local Structural V&V in in-space assembly. All four of the identified key technologies enter the demonstration at TRL 2; they have peer reviewed publications of the underlying concept, and the practical application has been identified but no experimental proof, matching the operational environment, is available. A successful TLT EDU demonstration would advance the TRL of these key technologies to TRL 4 meeting the exit criteria of TRL 2, documented description of the application, and the exit criteria of TRL 3, documented experimental results validating the technologies. With a successful TLT EDU demonstration, the four key technologies will be functionally validated in a laboratory environment.

The first two key technologies: the application of BTs and ROS2 to in-space assembly, are important for utilizing these powerful technologies in future in-space missions. Raising the TRL levels will be critical for making arguments during future trade studies on the architecture of a new system. To reach exit criteria for TRL 3, a successful demonstration of the CRS with a simulated ARS would be sufficient. A subsystem demonstration of the CRS would show BTs can be used to dictate the actions of a supervised autonomous system and the CRS's control of its actuators would demonstrate the use of ROS2 for in-space assembly applications in a laboratory environment.

The other two key technologies: the application of machine learning based vision systems to in-space assembly and Automated local V&V of in-space assembly structures, will be demonstrated by the vision system. To validate the machine learning based vision system, TLT will demonstrate object localization in all relevant operating modes. To advance the TRL of automated local V&V, TLT will demonstrate the identification of joint faults and success joining. The key technologies related to the vision system can be advanced in parallel with the demonstration of the autonomous agents and if needed, these systems can be supplemented with analogous systems during the demonstration of the TLT EDU. Additional work is being done by TLT to determine the efficacy of using vision systems in lunar lighting but is outside the scope of this paper [23].

VII. Concluding Remarks

The software design for the supervised autonomous assembly of a tall lunar tower was presented. The publication began with a high-level description of the concept of operations including the tower assembly process. Next, an overview of the system architecture was provided with a focus on the main agents of the system. After the overview of the main agents, an overview of the system implementation was given with a focus on ROS2 and the frameworks leveraged from the ROS2 ecosystem. Lastly, a discussion of TLT EDU's key technologies and the TRL entry and exit levels of these technologies was addressed. If successful in future lunar missions, the system presented here would enable a wide-ranging set of capabilities on the lunar surface including communication, surveillance, power generation, and more.

At the start of the project, TLT performed a trade study between two types of state machines: BTs and Hierarchical State Machines. Although, we expected Hierarchical State Machines to be preferred, BTs were chosen due to the large amount of states of the system and the development cost of maintaining the complex transition structure. BTs are not the best solution for every system; the CRS Lifter for instance has a small number of states and implementing the behavior tree for it would have had a higher development cost. A metric that wasn't in our initial trade but should have been, is testability. One large benefit from using BTs is the ability to verify and validate (V&V) branches independently from the rest of the system. Furthermore, being able to perform independent V&V of subtrees allows the project to make a test plan starting from the leaves of the tree. As low-level actions are tested and validated, you are able to continue to climb the tree while increasing the complexity of your tests moving from unittests, to functional tests, to integration tests, and finally to system level tests.

In subsection V.C.3, it was noted that verification of a well-placed rivet had a high development cost for our team and due to time constraints on our project we had to mitigate the risk using additional cameras and an operator in the loop. An in-depth trade between different joining technologies is outside of the scope of this paper but it is important to note TLT accepted a reduction in autonomy, secondary to the lack of rivet V&V, for the EDU because of how unlikely it is for a rivet to be chosen as the joining technology for a lunar system. Although, the joining technique may not be applicable to future lunar systems, the assembly approach and software is applicable to analogous joining methods that are easier to verify.

Appendix

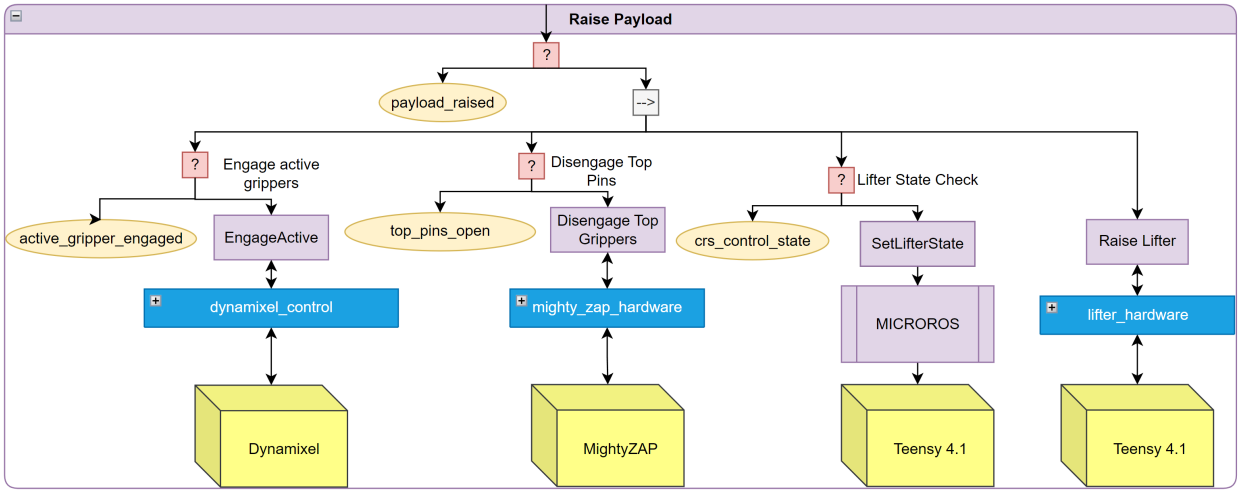

Fig. 18 CRS Raise Payload Behavior

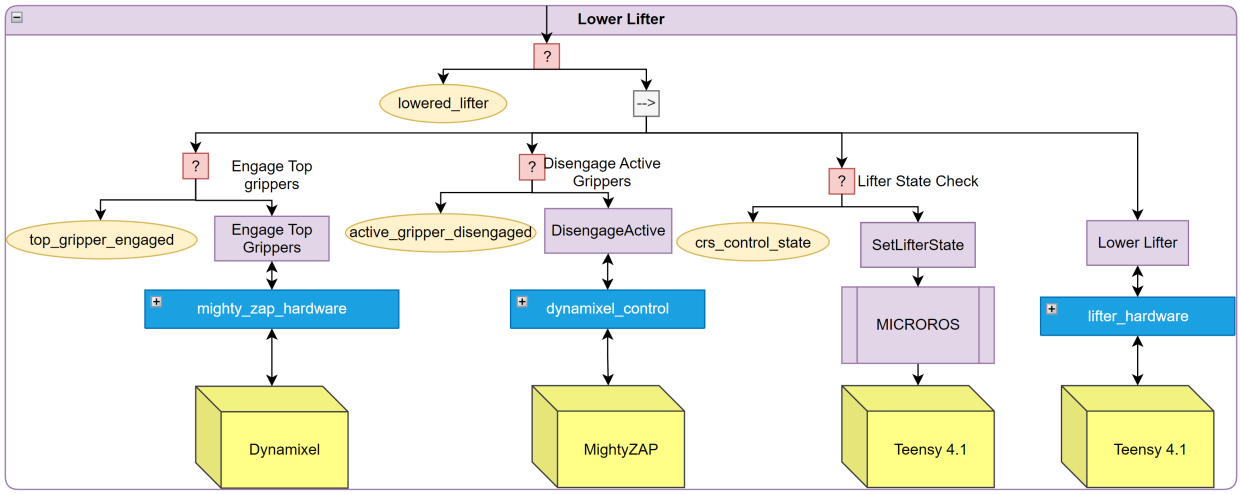

Fig. 19 CRS Lower Lifter Behavior

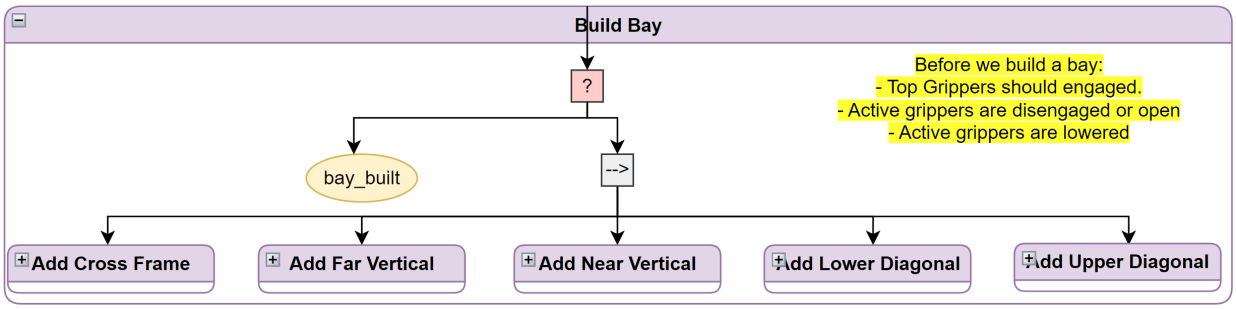

Fig. 20 CRS Build Bay Behavior

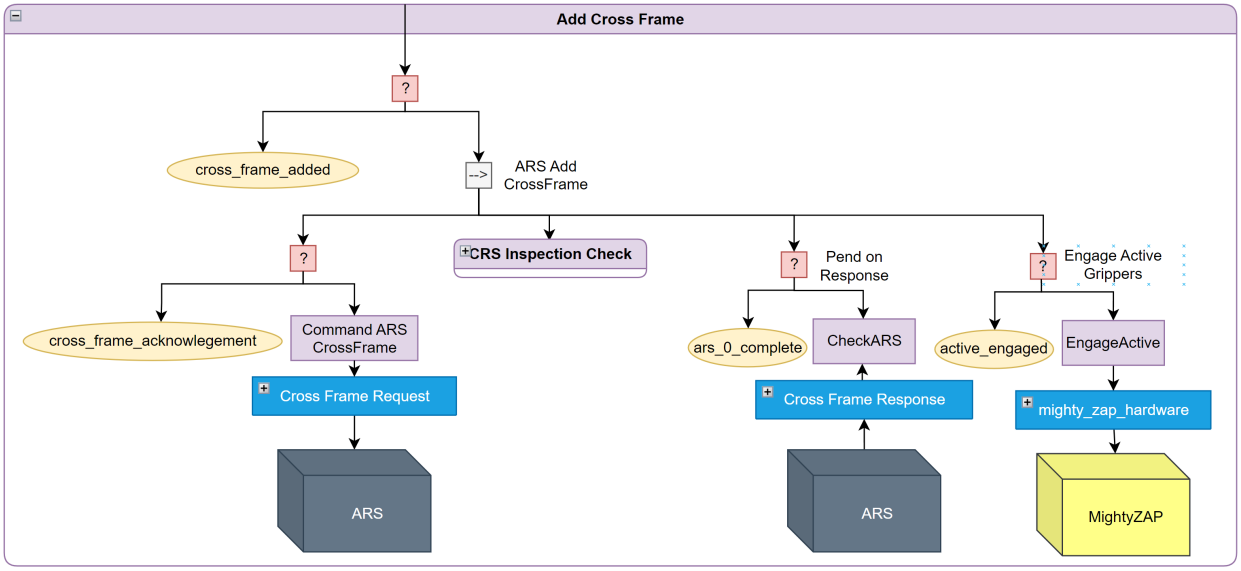

Fig. 21 CRS Add Cross Frame Behavior

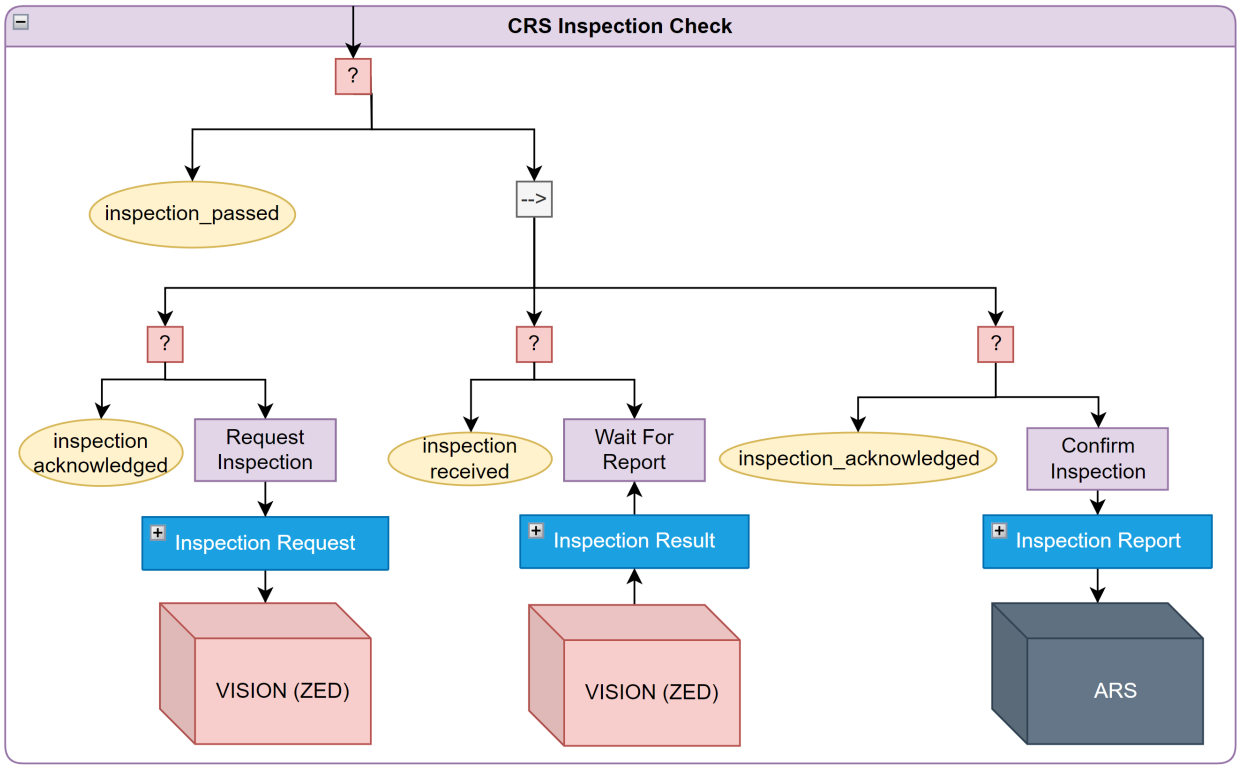

Fig. 22 CRS Inspection Check Behavior

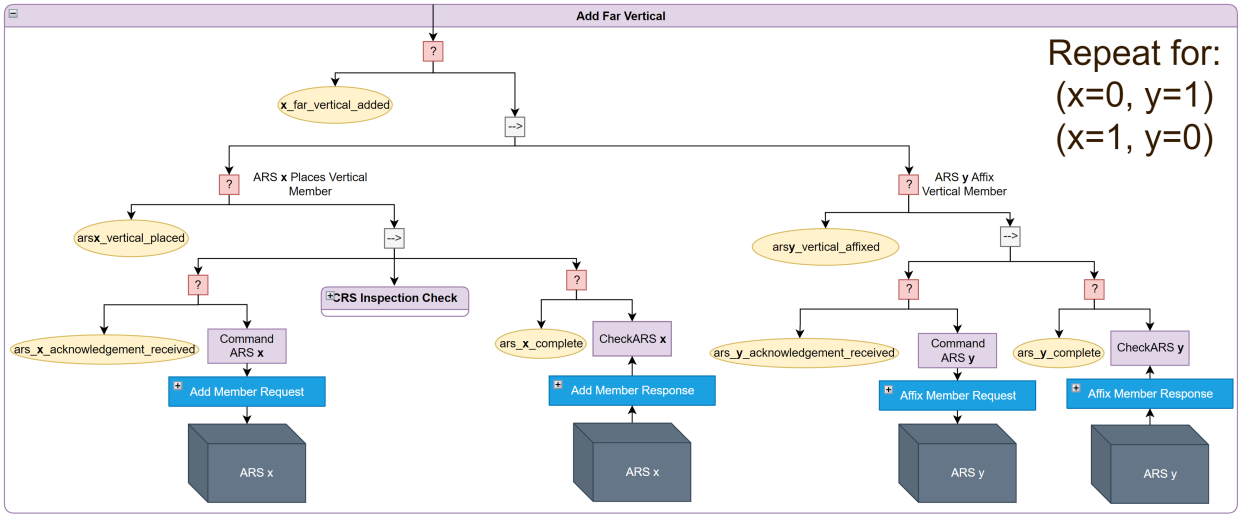

Fig. 23 CRS Add Far Vertical Behavior

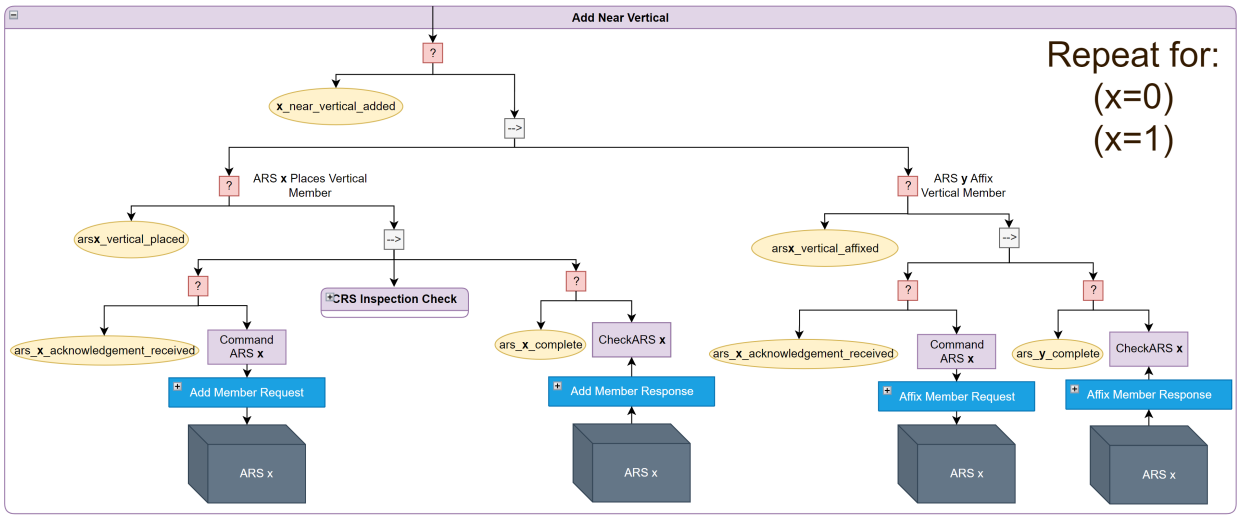

Fig. 24 CRS Add Near Vertical Behavior

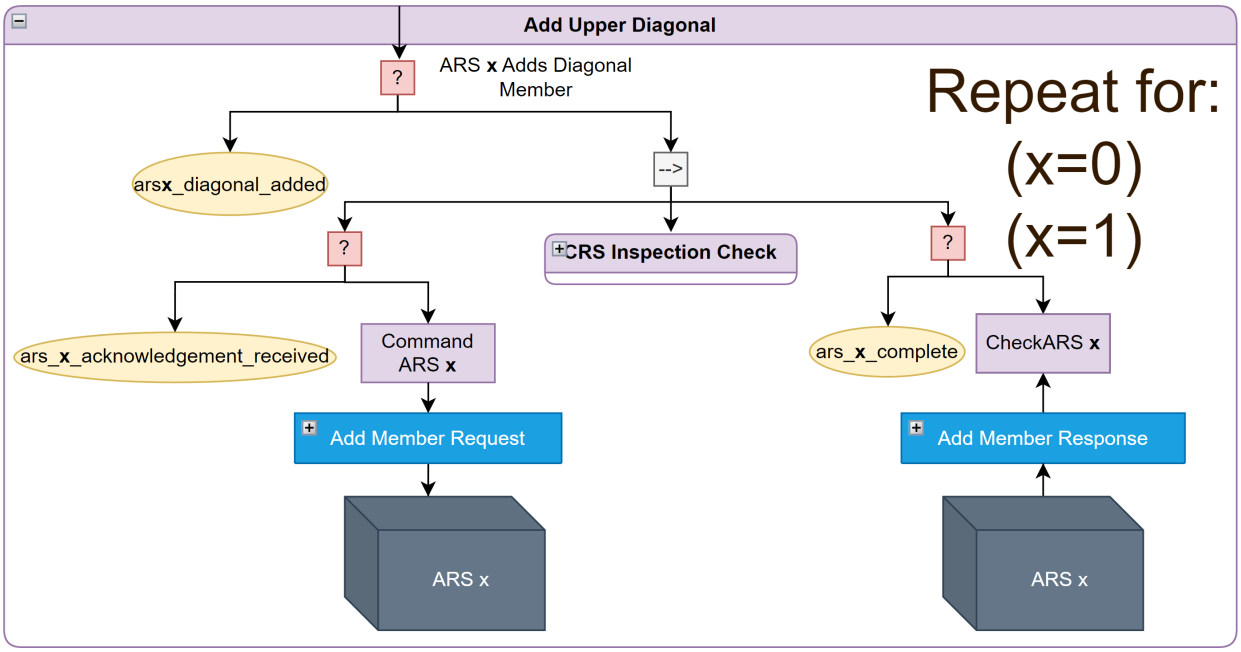

Fig. 25 CRS Add Upper Diagonal Behavior

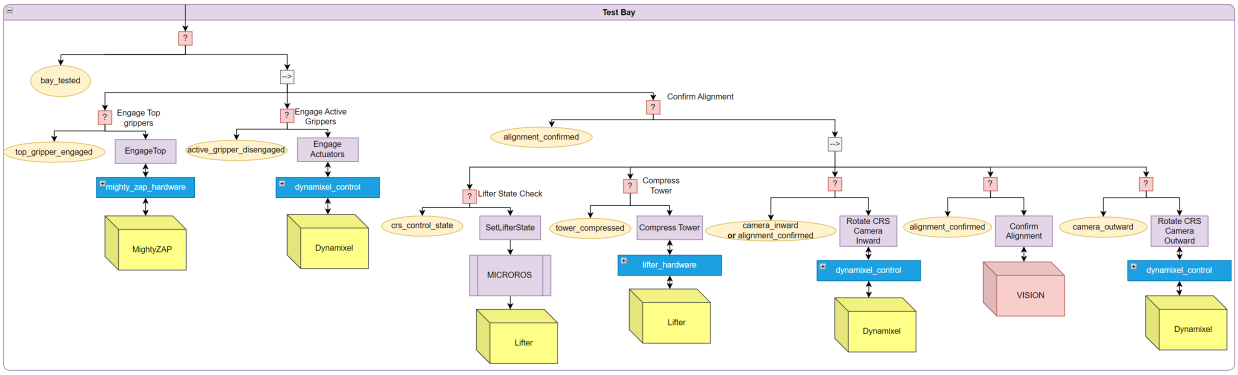

Fig. 26 CRS Test Bay Behavior

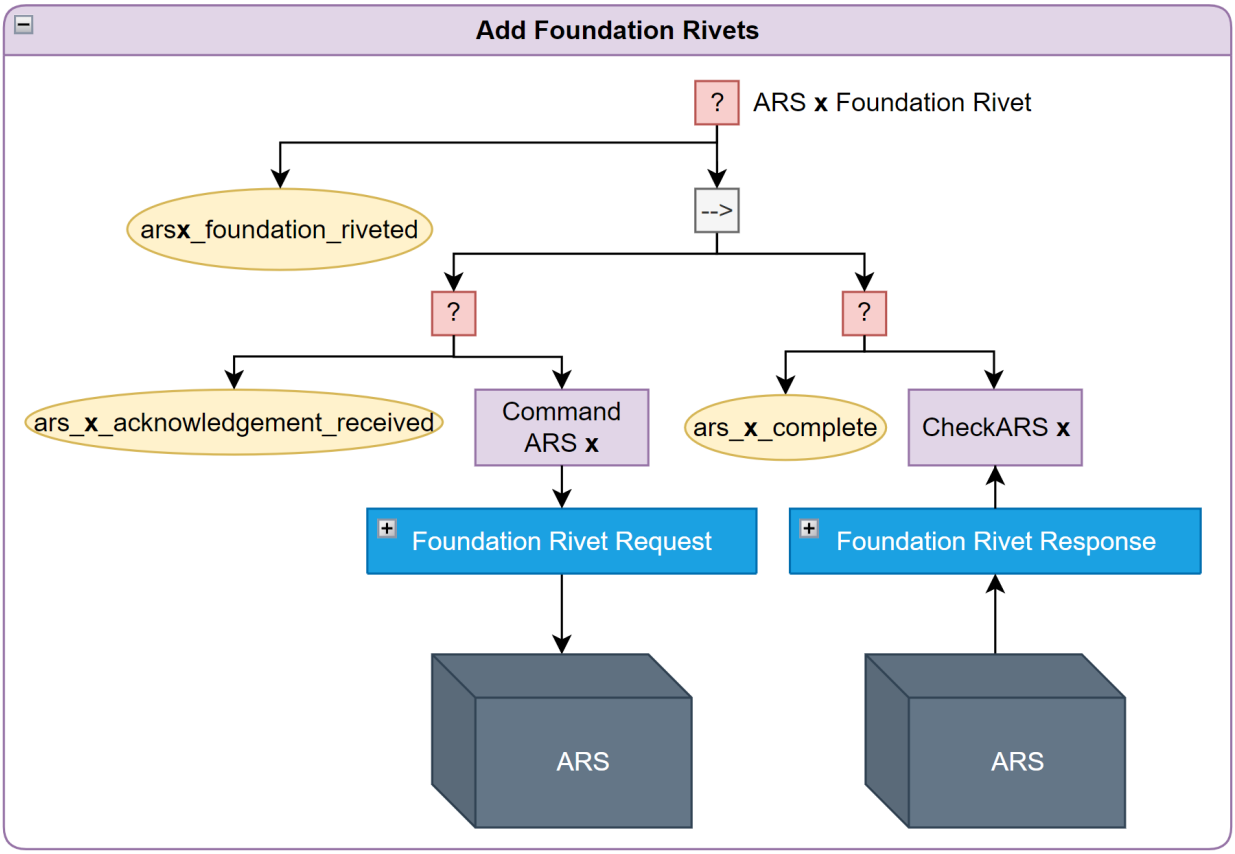

Fig. 27 CRS Add Foundation Rivets Behavior

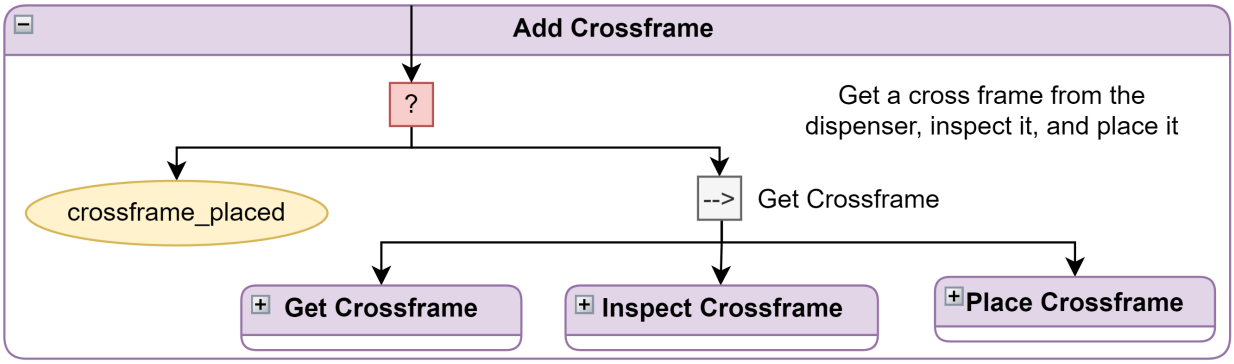

Fig. 28 ARS Add Crossframe Behavior

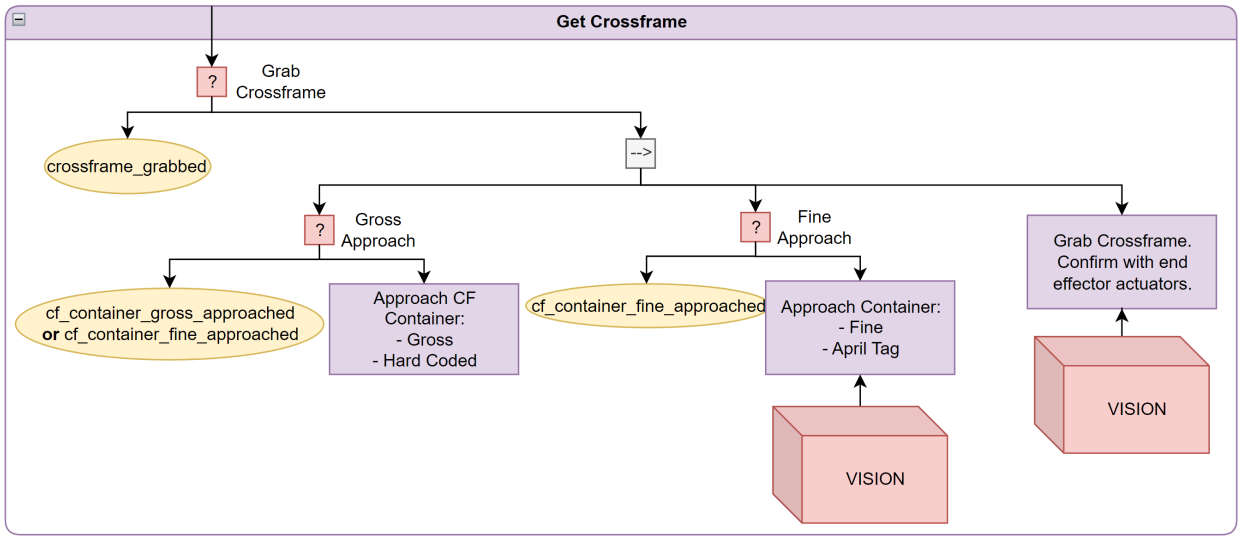

Fig. 29 ARS Get Crossframe Behavior

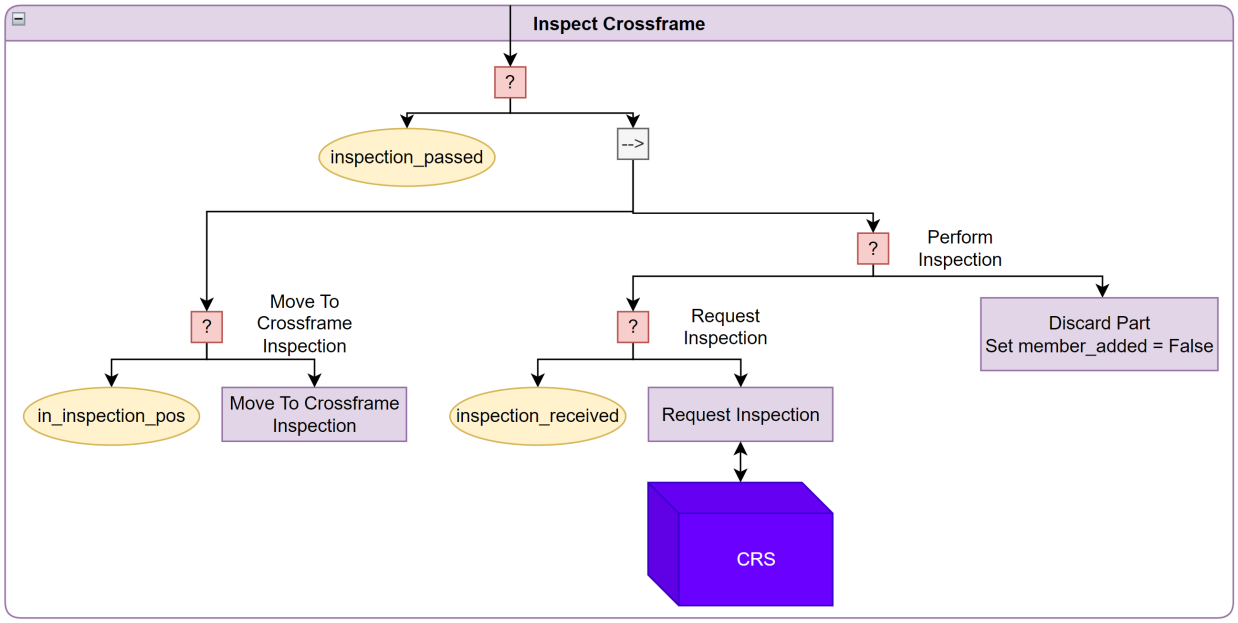

Fig. 30 ARS Inspect Crossframe Behavior

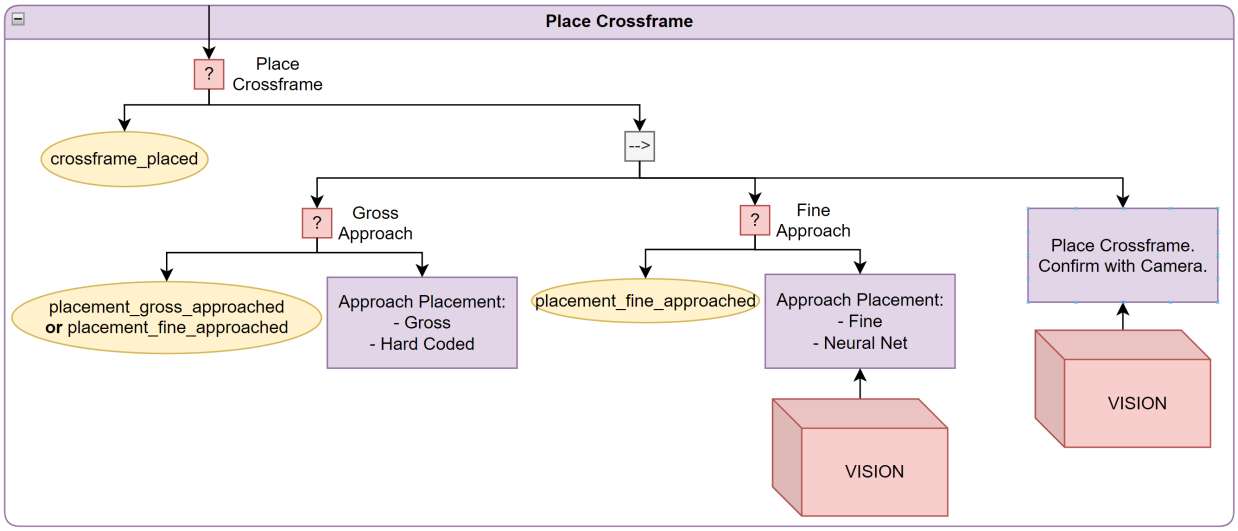

Fig. 31 ARS Place Crossframe Behavior

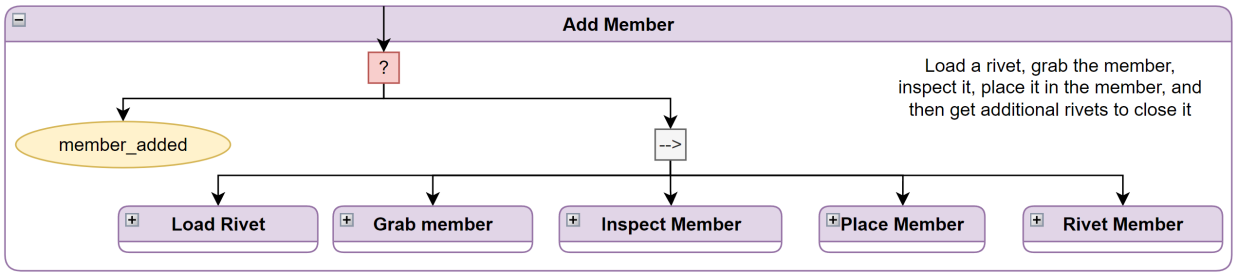

Fig. 32 ARS Add Member Behavior

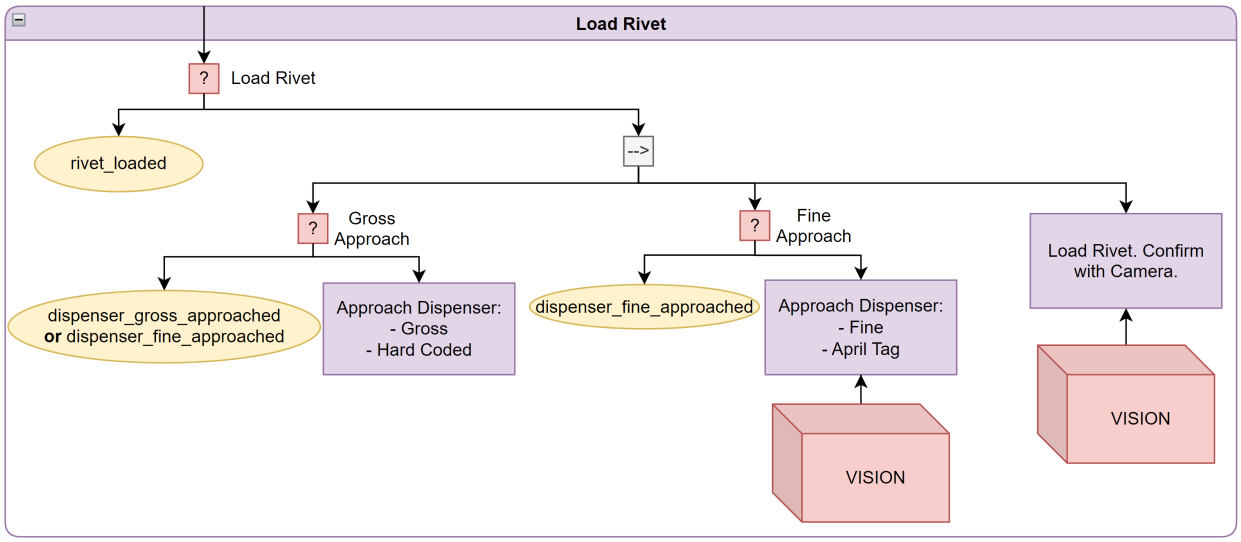

Fig. 33 ARS Load Rivet Behavior

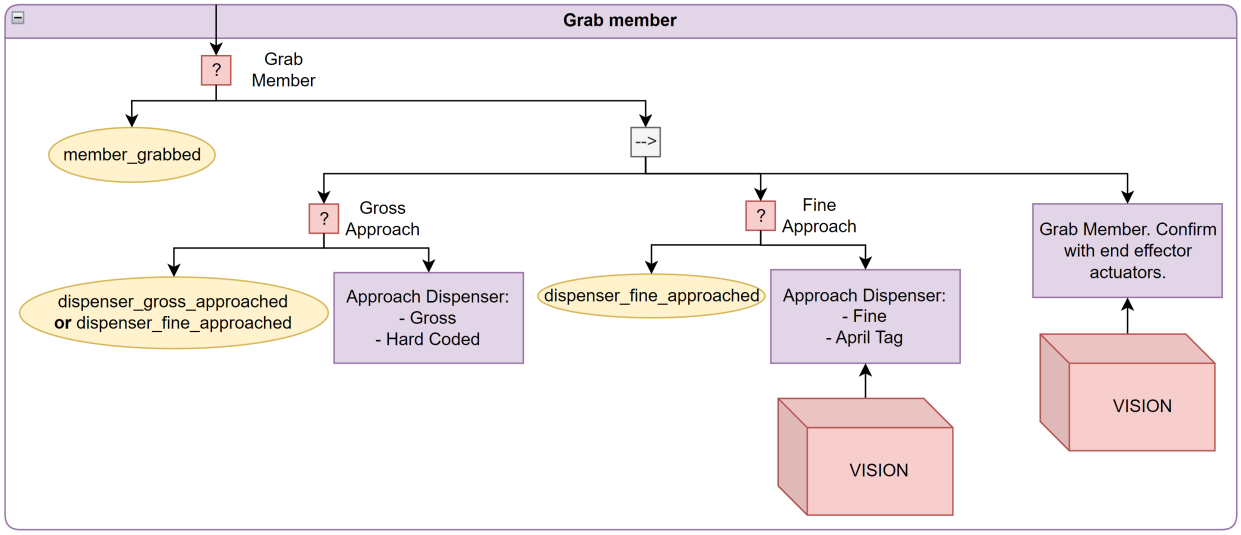

Fig. 34 ARS Grab Member Behavior

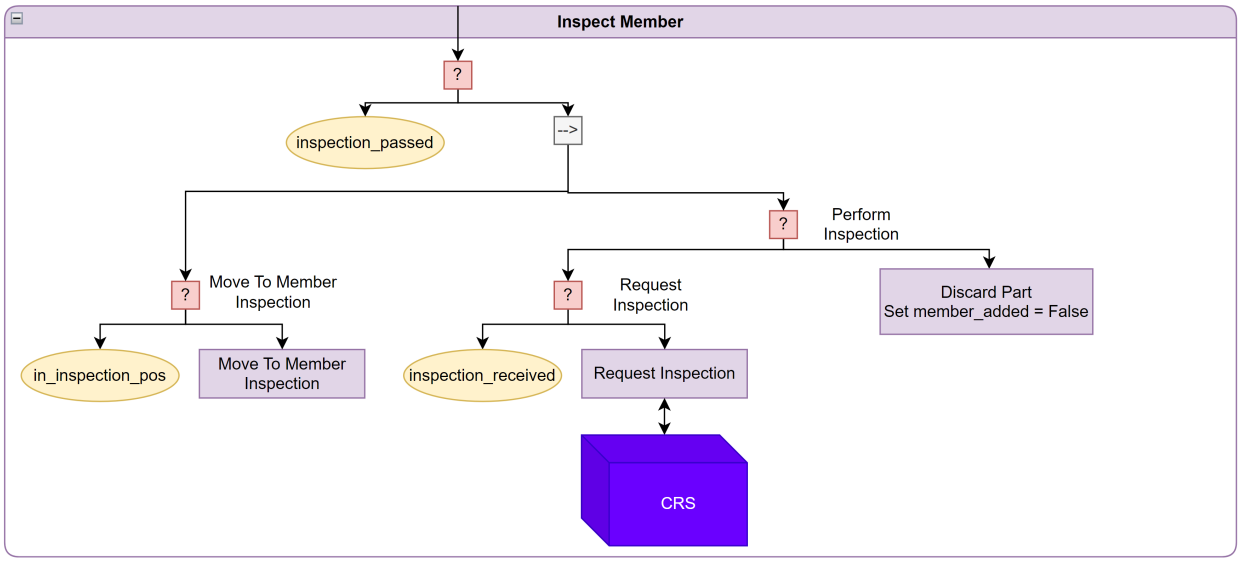

Fig. 35 ARS Inspect Member Behavior

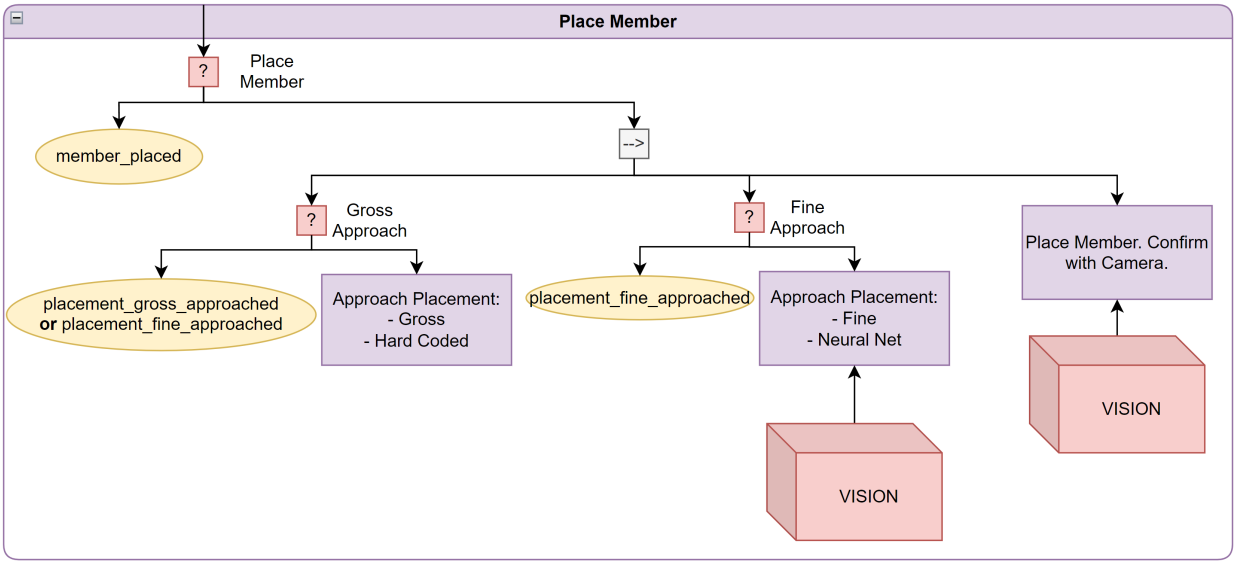

Fig. 36 ARS Place Member Behavior

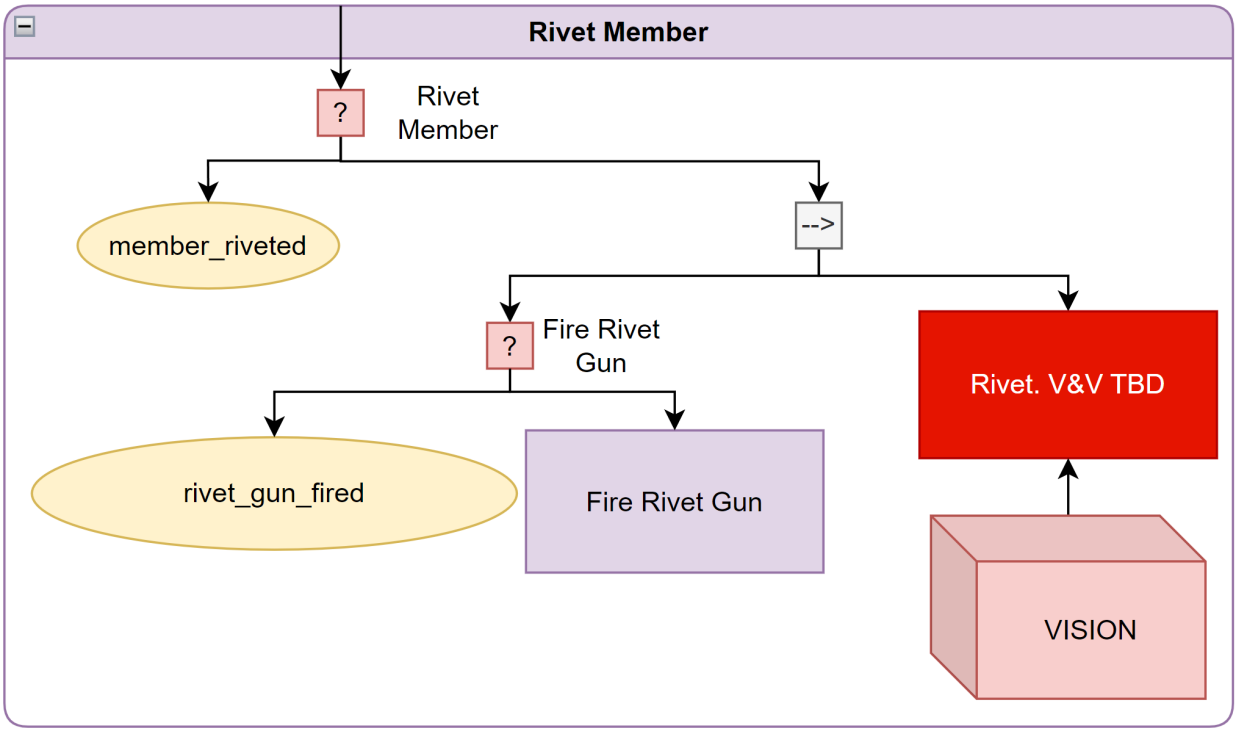

Fig. 37 ARS Rivet Member Behavior

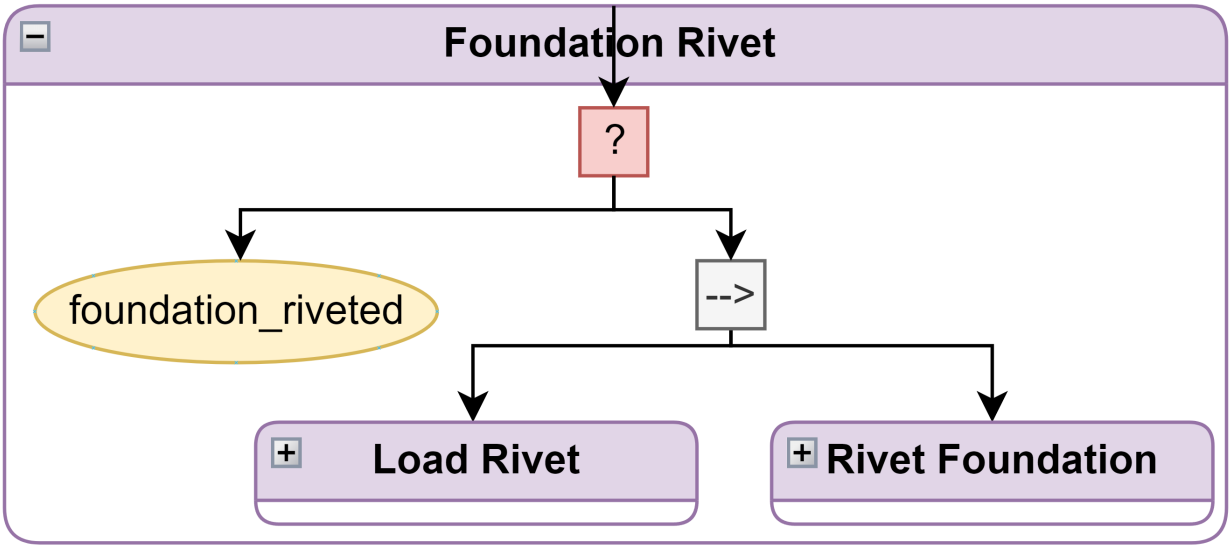

Fig. 38 ARS Foundation Rivet Behavior

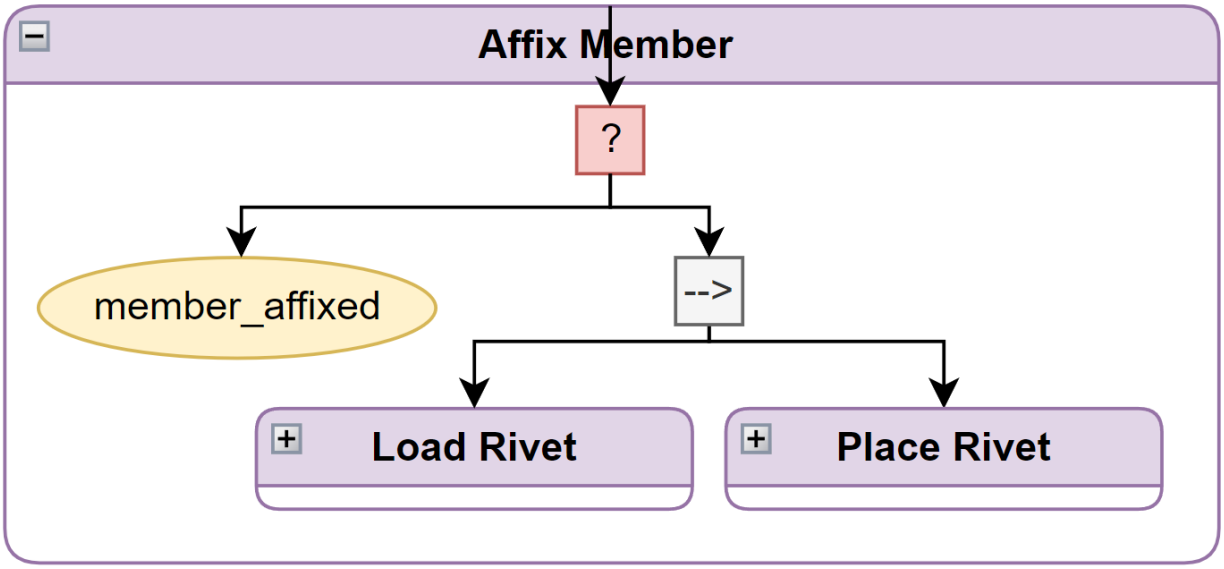

Fig. 39 ARS Affix Member Behavior

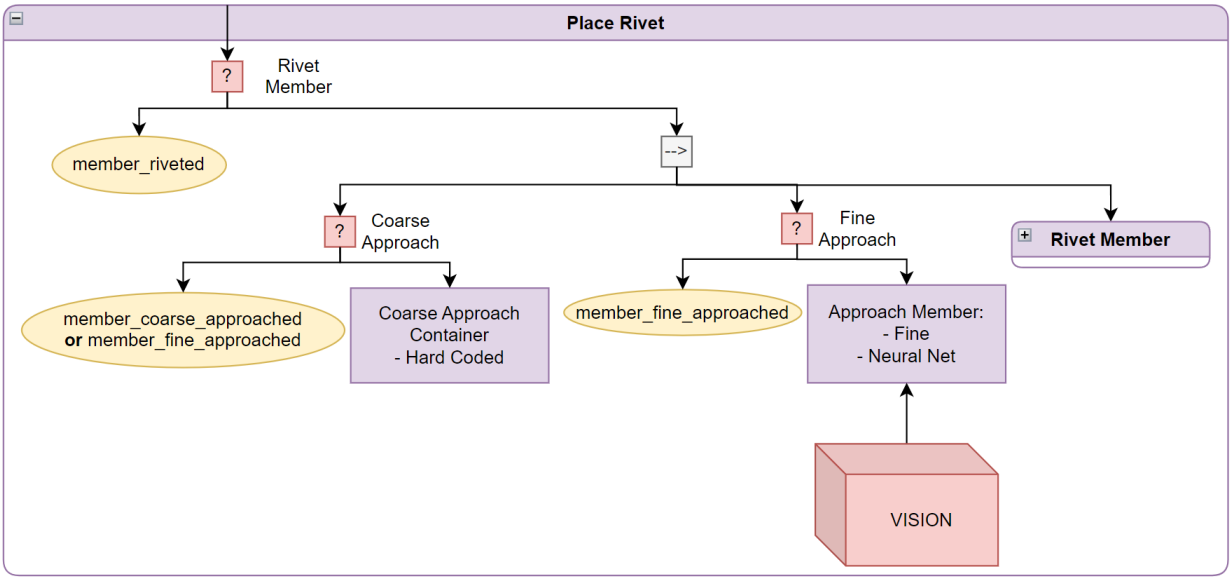

Fig. 40 ARS Place Rivet Behavior

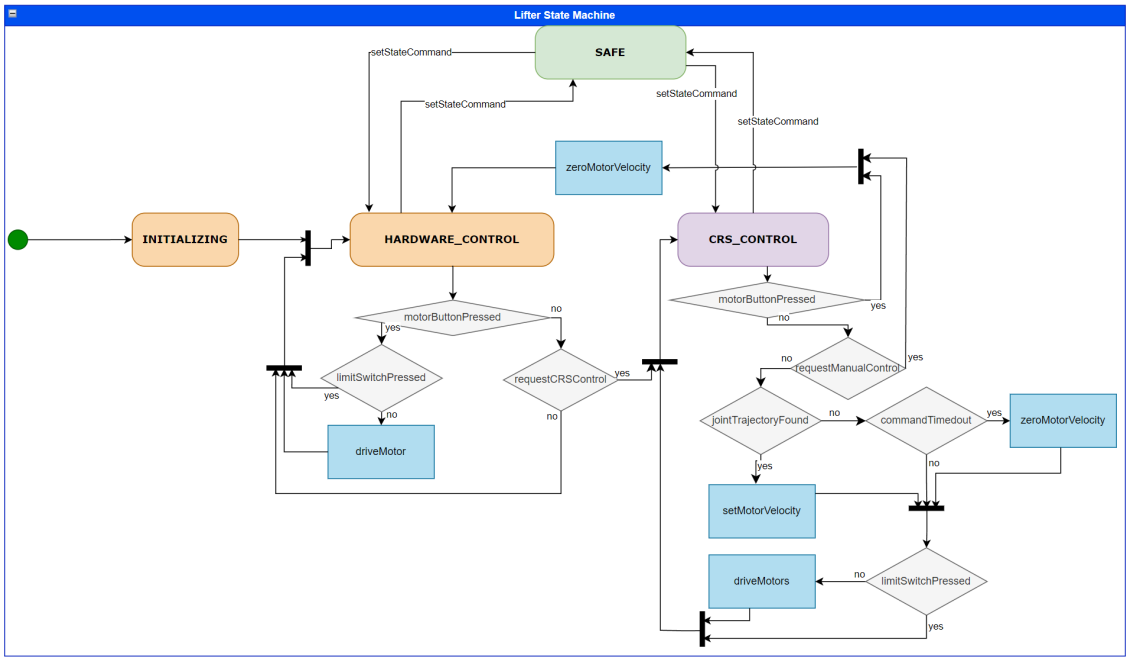

Fig. 41 Lifter State Machine

Acknowledgments

The work presented here was funded by Advanced Exploration Systems (AES) Project Polaris. The Project Polaris initiative seeks to fill high-priority capability gaps for deep space missions while involving early-career employees from across NASA.

References

[1] Doggett, W. R., Heppler, J., Mahlin, M. K., Pappa, R. S., Teter, J., Song, K., White, B., Wong, I., and Mikulas, M., "Towers: Critical Initial Infrastructure for the Moon," *AIAA SciTech 2023 Forum*, 2023.

[2] Henrickson, J. V., and Stoica, A., "Reflector placement for providing near-continuous solar power to robots in Shackleton Crater," *2017 IEEE Aerospace Conference*, 2017, pp. 1–10. https://doi.org/10.1109/AERO.2017.7943944.

[3] Ruppert, S., Ross, A., Vlassak, J., and Elvis, M., "Tall Towers on the Moon," , 2021. https://doi.org/10.48550/ARXIV.2103.00612, URL https://arxiv.org/abs/2103.00612.

[4] Song, K., Mikulas, M., Mahlin, M. K., and Cassady, J. T., "Sizing and Design Tool for Tall Lunar Tower," *AIAA SciTech 2023 Forum*, 2023.

[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] Pardo-Castellote, G., "Omg data-distribution service: Architectural overview," *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.*, IEEE, 2003, pp. 200–206.

[7] The Qt Company, "Qt," , 2023. URL https://www.qt.io/.

[8] *CCSDS Recommended Standard for Space Packet Protocol*, Consultative Committee for Space Data Systems, 2020. URL https://public.ccsds.org.

[9] Hipp, R. D., "SQLite," , 2020. URL https://www.sqlite.org/index.html.

[10] Chitta, S., Marder-Eppstein, E., Meeussen, W., Pradeep, V., Rodríguez Tsouroukdissian, A., Bohren, J., Coleman, D., Magyar, B., Raiola, G., Lüdtke, M., and Fernández Perdomo, E., "ros_control: A generic and simple control framework for ROS," *The Journal of Open Source Software*, 2017. https://doi.org/10.21105/joss.00456, URL http://www.theoj.org/joss-papers/joss.00456/10.21105.joss.00456.pdf.

[11] dynamixel-community, "dynamixel_hardware," , 2023. URL https://github.com/dynamixel-community/dynamixel_hardware.

[12] Universal Robotics, "Universal_Robots_ROS2_Driver," , 2023. URL https://github.com/UniversalRobots/Universal_Robots_ROS2_Driver.

[13] micro-ROS, "micro_ros_arduino," , 2023. URL https://github.com/micro-ROS/micro_ros_arduino.

[14] Colledanchise, M., and Ögren, P., *Behavior Trees in Robotics and AI: An Introduction*, CRC Press, 2020.

[15] Stonier, D., "py_trees," , 2023. URL https://github.com/splintered-reality/py_trees.

[16] Biggs, G., Perron, J., and Loretz, S., "Actions," , 2020. URL https://design.ros2.org/articles/actions.html.

[17] Bradski, G., "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[18] Coleman, D., Sucan, I., Chitta, S., and Correll, N., "Reducing the Barrier to Entry of Complex Robotic Software: a MoveIt! Case Study," , 2014. https://doi.org/10.48550/ARXIV.1404.3785, URL https://arxiv.org/abs/1404.3785.

[19] Talak, R., Peng, L. R., and Carlone, L., "Certifiable Object Pose Estimation: Foundations, Learning Models, and Self-Training," *IEEE Transactions on Robotics*, 2023, pp. 1–20. https://doi.org/10.1109/TRO.2023.3271568.

[20] He, K., Gkioxari, G., Dollár, P., and Girshick, R., "Mask R-CNN," , 2017. https://doi.org/10.48550/ARXIV.1703.06870, URL https://arxiv.org/abs/1703.06870.

[21] Foote, T., "tf: The transform library," *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*, 2013, pp. 1–6. https://doi.org/10.1109/TePRA.2013.6556373.

[22] NASA, "Technology Readiness Level Definitions," , 2023. URL https://www.nasa.gov/pdf/458490main_TRL_Definitions.pdf.

[23] Notosubagyo, B., Mahlin, M. K., and Cassady, J. T., "Unreal Engine Testbed for Computer Vision of Tall Lunar Tower Assembly," *ASCEND 2023*, 2023.