Rapid Lightweight Firmware Architecture of the Mobile Metamaterial Internal Co-Integrator Robot

Damiana Catanoso, In Won Park, Taiwo Olatunde
KBR, Inc.
NASA Ames Research Center
Moffett Field, CA 94035
damiana.catanoso@nasa.gov

Olivia Formoso, Greenfield Trinh, Christine Gregg, Elizabeth Taylor, Megan Ochalek, Kenneth Cheung
NASA Ames Research Center
Moffett Field, CA 94045
oliviairene.b.formoso@nasa.gov

***Abstract*— The Mobile Metamaterial Internal Co-Integrator (MMIC-I) is a structure assembly and servicing robot for in-space servicing, assembly, and manufacturing of primary structures and infrastructure. MMIC-I is a battery-powered crawling robot that can travel through periodic structures such as trusses and open framework mechanical metamaterials. It does this through sequences of component extension, contraction, and gripping. This paper provides a detailed discussion of MMIC-I's lightweight and rapidly developed firmware architecture, to enable demonstration of robot locomotion, secondary operations, and communications with a central command source. The rationale for the lightweight rapid development approach is to allow for assessment of long term system requirements in parallel with the mechatronics development, including optimization of system and subsystem power densities, to inform a future choice of flight ready software frameworks. MMIC-I system computing and I/O requirements are much lower than what is provided by proven baseline computing hardware for existing flight ready software frameworks such as the core Flight System, F prime, and the Robot Operating System. Development of earth gravity ground demonstration of the robotic systems is greatly benefited by limited power and mass factors for computing hardware. Here, we implement inter-process communication, commanding, and telemetry with the Espressif ESP32 module running the Arduino OS.**

TABLE OF CONTENTS

1. INTRODUCTION

In-space robotic servicing, assembly, and maintenance can allow space structures to bypass the limitations of single-vehicle launches. Early examples used astronaut extra-vehicular activities (EVA) and human-operator controlled dexterous manipulators to assemble trusses and the International Space Station [1]. Since EVAs are hazardous and crew-time is extremely valuable, recent efforts have focused on robotic assembly to conduct construction tasks [2][3][4][5][6]. However, despite these research investments, the challenges of reliable autonomous robotic assembly of complex and precise structures in unstructured environments remains a significant technology challenge [7].

Large space structures are a key capability for both next generation science and human exploration of the solar system and beyond. Sustained human presence on the Moon and Mars will require larger scale habitats, infrastructure, and space-hubs [8], while larger aperture telescopes are needed to capture better and more diverse snapshots of the universe [9]. Though current space assets rely on a "pack, launch, deploy" paradigm that relies on complex deployment with high risk of failure, the size of deployed structures is still subject to single-launch restrictions.

The Automated Reconfigurable Mission Adaptive Digital Assembly Systems (ARMADAS) project, currently under development in the Coded Structures Lab at NASA Ames Research Center, proposes implementation of large space structures using relatively small robots that operate in and on a modular structural system itself [10], to simplify the autonomy and robotic requirements. The strategy is based on breakthroughs in high-performance mechanical metamaterial lattice structures that are constructed by discrete assembly from multiple materials [11][12][13][14][15]. By controlling the geometry, density, and constituent material of an individual unit cell (termed "voxel" for volumetric pixel), a wide variety of mechanical properties can be realized at scale [16][13][14][15] with an unlimited build envelope [16]. This type of robot is termed a "relative robot" [17][18][19], and uses a highly-structured environment for alignment and localization. This allows for very simple robotic agents with minimal sensing (no vision for instance) to autonomously assemble complex structures.

Relatively mature examples of robotic servicing for space applications include full suites of functionality per robot, with navigation and manipulation supported by machine vision over fully autonomous and teleoperation modes. Control architecture is typically handled with multiple processing threads over multiple devices, with centralized coordination of tasks that are distributed over child processes according to the distribution of hardware systems or by task criticality [20][21]. Modern conventional software practices and tools, such as the Robot Operating System (ROS), are sometimes incorporated [20].

The ARMADAS system distributes physical sub-tasks across different robot types [22], with a system architecture as a whole having similar overall complexity to prior robotic servicing demonstrations. This differentiation or specialization allows each individual robot to be a relatively simple system with low numbers of actuated degrees of freedom and control states. Modular scalability to very large systems can leverage a distributed architecture [23], but for development purposes we implemented a centralized architecture that allows for simulated hybrid and distributed control [24]. Autonomy is achieved through specifically designed algorithms that produce a "plan file", that include a sequence of concurrent robot motions. An operations control and telemetry user interface and logging software (opsUI) coordinates the execution of these motions based on regular feedback received from the robots.

This paper focuses on the firmware architecture of the relative robot assembly agent type called the Mobile Meta-Material Interior Co-Integrator (MMIC-I). Multi-robot autonomy and other robot types are subjects of companion articles [25] [26].

We first provide an overview of MMIC-I's avionics. Section 3 presents the overall firmware architecture. Internal communication between controller boards, as well as their interaction and coordination, is presented in Section 4. Section 5 describes the controller strategy. Section 6 describes the motion generation, primitives and robot states. Section 7 presents MMIC-I's operating modes and the autonomous fault detection implementation. Conclusions and future work follow.

System Description

MMIC-I is designed to locomote internally throughout a structure, and actuate fasteners that are captive in the structural unit. MMIC-I has a symmetric structure about the hip module. Each of the two symmetric sides has an extension arm, a gripper, and a bolter module. These modules are highlighted in Figure 1. For a full description of the kinematics and hardware design, please see our companion article on this subject [27].

MMIC-I exercises no localization in the global reference frame of the structure, but it is able to calculate its current configuration (including rotational orientation) among a total of 15 discrete states it can occupy and execute reconfiguration commands. One side is designated as the primary board (board A). It embeds the motion generation, motion primitives and coordinates operations of each of the robot modules that result in the desired motion. It handles the WiFi client and directly controls side A of the robot, as well as the hip. The B side of the robot receives low level instructions through a wired serial connection with the A side, to move the modules connected to it.

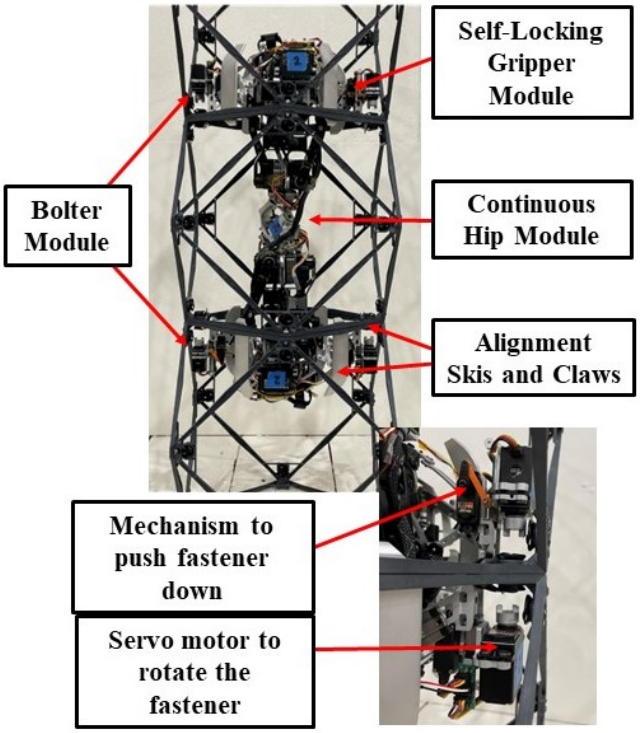

Figure 1. MMIC-I's modules.

2. AVIONICS

MMIC-I is powered by two 1000 mAh lithium polymer batteries connected in parallel, providing between 10 V and 12.6 V. Below 10 V, the battery is considered out of charge and must be replaced and recharged. Figure 2 shows the top and bottom view of the MMIC-I controller board, which has been appositely designed and fabricated. The 8-layers board includes the ESP32-WROOM-32E [28] as the dual core processor, supporting WiFi and Bluetooth. The board is equipped with a reset and boot button, as well as some probing points useful for board testing and debugging. The board has two DF3 power connectors (see Figure 2). The four-pin one connects the board to the adjacent battery through a switch. The three pin one activates the in-parallel connection with the battery mounted on the symmetrically opposite side, close to the other board. The micro USB connector doesn't provide power, it only provides serial connection and firmware upload capability. The board has a voltage regulator that lowers the voltage provided by the batteries to 7.4 V to all the modules - servo motors adopted in the modules have an operational voltage of 7.4 V. The board has a connector dedicated to the board A-board B serial communication. The controller board incorporates the MPU-6000 Inertial Measurement Unit (IMU) [29], used by the firmware to calculate the current state. The 8-channels ADS7828 analog-to-digital converter with I2C interface [30] is used for voltage and current sensing. These readings are used for autonomous faults/warning detection and operating mode change. For instance, the unregulated voltage reading is used to trigger

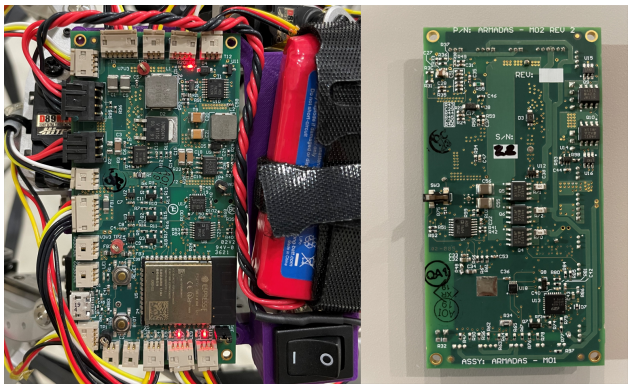

Figure 2. MMIC-I's custom made controller board - top view mounted on the robot (left), bottom view (right).

the low-battery warning, the locomotion current reading is used to determine whether the robot motion is being impeded for any reason, and so on. Chapter 7 provides a detailed description of the fault detection implementation.

Because of the scarcity of space on the board to include all the bolter pins, the ADG1604 4-channels multiplexer [31] was chosen to activate the bolter pair to be used, through selecting one among the four possible combinations of the two input channel A0-A1 (low-low, low-high, high-low, high-high). The hip sensor is not located on the board. Although the hip servo is the same servo adopted by the arms, it has been modified to be a continuous servo. This means that it requires a velocity closed loop controller, which uses a position sensor. The magnetic encoder is mounted on the hip, and connected to board A through an I2C line. All the raw sensor data is sent to a moving average calculator, with a window size of 50 samples.

The Espressif ESP32 dual core capability is utilized in board A only, where core 1 runs the control loop and core 0 hosts the WiFi Client.

3. FIRMWARE ARCHITECTURE

Figure 3 provides a high level representation of MMIC-I's main components, their mutual interaction and the interaction with the opsUI. When the opsUI reads in the plan file that a command has to be sent to MMIC-I, it sends the proper command ID to the server board through serial port.

Table 1 summarizes the types of commands available for MMIC-I. The first four rows describe commands to move a single module. These are used for debugging and tuning of motions. The following commands are adopted during operation. MMIC-I exchanges information with the opsUI over a WiFi bridge that includes a server board on the opsUI side. This server board is an ESP32 WROOM devkit module, and is connected to the opsUI software via RS232 over USB.

The primary function of the server board is as a dedicated network interface to allow opsUI to maintain multiple persistent WiFi connections with many robots, utilizing as many server boards as robots. It also performs limited formatting and pre-processing of command and feedback packets, for development reasons. In Figure 4 the block diagram relative to the server operation is shown. When the server detects that there is information available on the serial port, it identifies the command and fills the command packet. The command ID is stored in another variable as well, called "command byte". Command byte stores the command ID until the corresponding command byte gets successfully sent to the client, after which it resets to a zero value. The server establishes and terminates connection with the given client at every loop. The reason is to support a single server-multiple clients operation. After connection is established, if the value of command byte indicates that there is a pending packet to be sent, it sends the packet. The server, then, checks if any packet has been received from the client. If there is a packet ready to be sent to the client but the server and client are not connected, the server keeps looping, trying to connect and send the packet at each loop. If the lack of connection lasts for a significant amount of time, the opsUI will detect it without using any information from the server. Every time the opsUI sends a command to MMIC-I through the server, it waits for a feedback heartbeat packet as acknowledgement. If the opsUI doesn't receive a feedback packet within a given time span, it will show a message on the opsUI and send the packet repeatedly until it gets a response.

Table 2 summarizes the content of the 14-bytes long command packet. The packet includes the client board number, in the event there are multiple robots attached to a single server, and the the command identifier. Few bytes in the packet are dedicated to target PWM commands for arms and hip, used only when a arm or hip position command is sent.

The WiFi client, which runs on core 0 of MMIC-I's board A, receives the command packet and responds with a heartbeat packet. The server-client connection type is IP. Figure 5

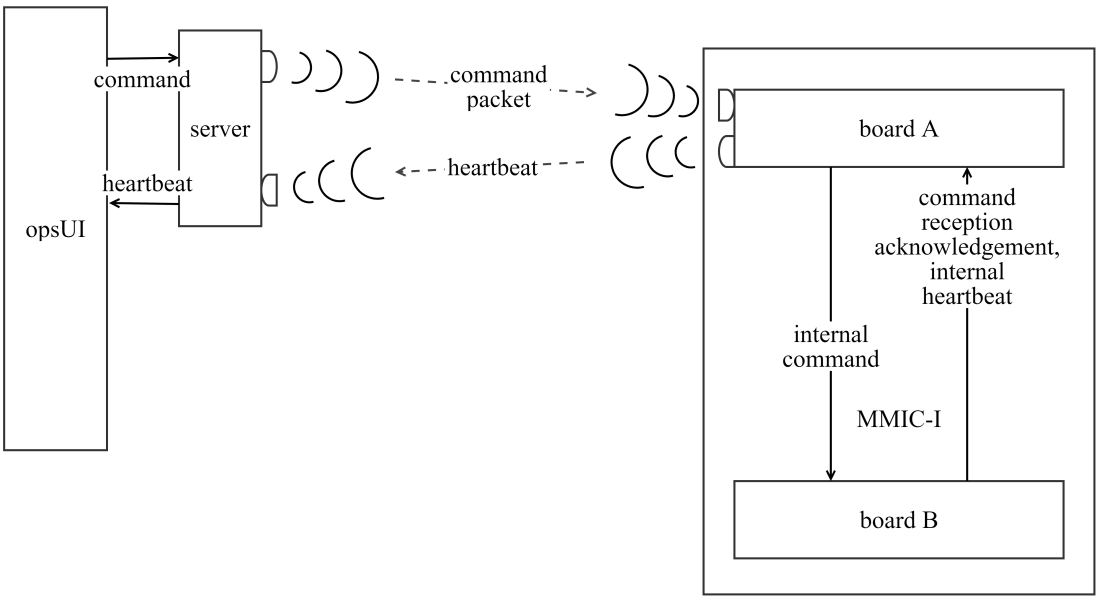

Figure 3. Conceptual diagram of MMIC-I's firmware operation highlighting the opsUI, the WiFi server-client communication, and the board to board wired communication.

Table 1. **A summary of MMIC-I commands.**

Command Type	Modules Engaged	Explanation
Single module	Arm	Totally/partially contract/extend arm on side A or B
Single module	Gripper	Totally/partially contract/extend grippers on side A or B
Single module	Hip	Rotate hip to most common positions or any other value
Single module	Bolter	bolt/unbolt a single fastener or all four, move bolters down
Primitive motion	All	Execute the commanded primitive motion
Stop	All	Emergency stop of current motion
Reset	All	After a stop command, resume motion
Power Control	All	Turns on-off power to individual/all modules
Reconfiguration	All	Reconfigure from an initial to a target state
Mode	All	Changes operating mode
System fault	All	Triggers the assembly system operation fault
Heartbeat	All	Requests a heartbeat packet to server

shows the block diagram of the WiFi function looping on core 0. After confirming there is an active connection, the client receives the packet and sends back to the server the updated heartbeat packet as acknowledgement. The content of MMIC-I's heartbeat packet is summarized in Table 3. The packet reports the target and current angle values for the locomotion-related modules, the battery voltage (V_{bat}), battery current draw (I_{bat}), arm and hip current draw (I_{loc}), gripper current draw (I_{grip}), regulated voltage (V_{reg}), bolter current draw (I_{bo}). The packets includes the last commanded target state, the command complete flag, current active bolter number and bolter attempt number, faults and operating mode information.

In board A, while core 0 is handling the WiFi task, core 1 handles the loop task. The two processes run independently from each other, with their own timing, but they share the same information, e. g. before sending the heartbeat packet, the WiFi task updates the packet with the latest values, and then sends it. The values it uses to update the packet come from the loop task, since it's the controller loop that reads the current/voltage sensors and angles encoders/interpolation values. In this process there's the risk that the two tasks try to access the same memory location at the same time, which would result in loss of communication efficiency. This issue was addressed implementing a communication system between tasks based on queues. When the WiFi task receives the command packet, it copies the packet content into a rx-queue, that is read by the loop task. The loop knows that after receiving a command packet, a heartbeat packet will be sent, so it updates its own copy of the heartbeat packet and sends it to a tx-queue. Then, the WiFi task receives the queue heartbeat information and sends it to the server.

4. BOARDS

MMIC-I's two controller boards work in a coordinated fashion to ensure synchronous execution of motion commands. Although the two boards control two symmetric and identical parts of the robot (with the exception of the hip that is only controlled by board A), board B is subordinated to A, in which it only serves to move the single modules attached to it, whenever commanded by A. Board A embeds all the motion primitives, the motion planning, the fault detection, operating mode handling, the bolter state machine. A block diagram of the firmware running on board A is shown in Figure 6.

Boards A and B are placed at the two far ends. They are connected by a serial cable that traverses the entire body of the robot. The cable allows for bi-directional serial communication, that is used by board A to send commands to B. Board B also uses the connection to send a periodic heartbeat to A.

Board B's firmware architecture is shown in Figure 7. This runs on ESP32-WROOM-32E's core 1 on MMIC-I's board B. The firmware includes the controller for each module, sensors reading, communication with board A. The heartbeat packet is shown in Table 4. The packet contains status variables relative to the B side, a series of Boolean variables needed for the synchronous operation of the bolters side A and B, an acknowledgement of received command from board A, the direction of the gravity vector in the reference frame of the IMU mounted on board B, overcurrent-related Boolean's, switch sensor pushed information after arm contraction command.

Table 2. **MMIC-I's command packet.**

0	**1**	**2**	**3**	**4**	**5**	**6**
header	header	header	header	board number	command	lowByte hip PWM
7	**8**	**9**	**10**	**11**	**12**	**13**
highByte hip PWM	lowByte arm A PWM/ final state	highByte arm A PWM/ init state	lowByte arm B PWM	highByte arm B PWM	lowByte checksum	highByte checksum

Table 3. MMIC-I's WiFi heartbeat packet.

0	**1**	**2**	**3**	**4**	**5**	**6**
header	header	header	header	board number	lowByte hip-target	highByte hip-target
7	**8**	**9**	**10**	**11**	**12**	**13**
lowByte hip-now	highByte hip-now	lowByte arm A-target	highByte arm A-target	lowByte arm A-now	highByte arm A-now	lowByte arm B-target
14	**15**	**16**	**17**	**18**	**19**	**20**
highByte arm B-target	lowByte arm B-now	highByte arm B-now	lowByte gripper A-target	highByte gripper A-target	lowByte gripper A-now	highByte gripper A-now
21	**22**	**23**	**24**	**25**	**26**	**27**
lowByte gripper B-target	highByte gripper B-target	lowByte gripper B-now	highByte gripper B-now	lowByte V_{batA}	highByte V_{batA}	lowByte I_{batA}
28	**29**	**30**	**31**	**32**	**33**	**34**
highByte I_{batA}	lowByte I_{locA}	highByte I_{locA}	lowByte I_{gripA}	highByte I_{gripA}	lowByte I_{3v3A}	highByte I_{3v3A}
35	**36**	**37**	**38**	**39**	**40**	**41**
lowByte V_{regA}	highByte V_{regA}	lowByte I_{boA}	highByte I_{boA}	lowByte V_{batB}	highByte V_{batB}	lowByte I_{batB}
42	**43**	**44**	**45**	**46**	**47**	**48**
highByte I_{batB}	lowByte I_{locB}	highByte I_{locB}	lowByte I_{gripB}	highByte I_{gripB}	lowByte I_{3v3B}	highByte I_{3v3B}
49	**50**	**51**	**52**	**53**	**54**	**55**
lowByte V_{regB}	highByte V_{regB}	lowByte I_{boB}	highByte I_{boB}	target state	command complete	active bolter
56	**57**	**58**	**59**	**60**	**61**	**62**
bolter attempt	empty	empty	empty	fault byte0	fault byte1	fault byte2
63	**64**	**65**				
operating mode	lowByte checksum	highByte checksum				

5. CONTROLLER

The gripper and arm modules operate through an open loop controller. Given a target state, the trajectory is calculated through simple interpolation and the interpolated values fed directly to the servos at a given frequency. The servos are individually programmed to have an internal velocity limit. The interpolation interval on the firmware side and the velocity limit on the servo side are tuned together to ensure a smooth motion at the desired speed. The only sensing associated with the arm and gripper motion is the current monitoring and a switch that confirms when the arm is contracted. The "current PWM" values included in the heartbeat correspond to the last interpolated value sent to the servos. This doesn't necessarily correspond to the actual PWM, but it's an accurate enough representation.

The hip module uses a custom proportional-integral (PI) feedback controller to reach its commanded position. To guarantee smooth motion during a rotation, a combination of velocity and position controller is implemented. When the target and current hip position differ more than 3°, the hip follows a PI velocity controller with a target speed of 10°/sec, using a smoothing technique to estimate the velocities from the angle readings over time. Once the hip position is within 3° of the commanded position, the controller transitions to a position controller to accurately reach its target, again estimating velocity from the servo's position readings. The utilization of a PI position controller for the final 3° allows the hip to minimize position error regardless of the robot's orientation with respect to gravitational force.

The bolter state machine controls the bolters operation. The diagram in Figure 8 shows bolter states. For each bolter pair that sequence of states gets executed once. Each face has four fasteners. To bolt a single face, the state machine has to run four times. MMIC-I can operate each bolter pair one at a time because the bolter PWM is sent to the multiplexer, which sends the signal to the bolter number identified by the analog high-low of two variables A0 and A1. There are four possible high-low combinations of A0 and A1, corresponding to the four bolter pair numbers. So, before executing the bolter state machine, the proper values for A0 and A1 are set and the desired bolter number activated. After setting the bolter pair (1 to 4), the bolters start their translation towards

Table 4. **MMIC-I internal heartbeat packet.**

0	**1**	**2**	**3**	**4**	**5**	**6**
header	header	header	header	lowByte arm B-target	highByte arm B-target	lowByte arm B-now
7	**8**	**9**	**10**	**11**	**12**	**13**
highByte arm B-now	lowByte gripper B-target	highByte gripper B-target	lowByte gripper B-now	highByte gripper B-now	lowByte V_{batB}	highByte V_{batB}
14	**15**	**16**	**17**	**18**	**19**	**20**
lowByte I_{batB}	highByte I_{batB}	lowByte I_{locB}	highByte I_{locB}	lowByte I_{gripB}	highByte I_{gripB}	lowByte I_{3v3B}
21	**22**	**23**	**24**	**25**	**26**	**27**
highByte I_{3v3B}	lowByte V_{regB}	highByte V_{regB}	lowByte I_{boB}	highByte I_{boB}	bolter B operation auxiliary variables	
28	**29**	**30**	**31**	**32**	**33**	**34**
bolter B operation auxiliary variables						
35	**36**	**37**	**38**	**39**	**40**	**41**
command received	g direction sign	g direction axis	arm B over-idle	gripper B over-idle	bolter B over-idle	arm B overcurrent
42	**43**	**44**	**45**	**46**		
gripper B overcurrent	bolter B overcurrent	arm B contracted	lowByte checksum	highByte checksum		

their respective fastener, push them towards each other, and go back to their initial configuration. The decision on when to stop pushing and start moving the opposite direction is based on current sensing. States whose transition is regulated by current sensing are indicated with an asterisk in Figure 8. The bolters are commanded to descend more than they physically can. So they will keep pushing even after they come in contact with the fastener until the I_{boA} and I_{boB} current threshold is met, and the state machine will switch to the next state. The same motion is repeated, but this time, when I_{bo} hits the threshold while pushing, it rotates the bolter head, that engages the fastener. At this point both bolter heads are rotating the opposite directions until the pair of fasteners is locked. Current sensing determines if it's locked. After detecting that the fasteners are locked, the bolters rotate the opposite direction and go back to their original configuration.

6. MOTION GENERATION

MMIC-I has 15 allowed configuration states [27]. Descriptive names are listed in Table 5. Each state is defined with respect to a reference voxel, considering the global coordinate frame of the structure. To define the state, let's imagine to have the coordinate frame at the center of the reference structural unit. The name of the state is determined by what axis of the reference frame each MMIC-I side aligns with, and what face each side is gripped to, negative if gripped on the negative coordinates, positive otherwise. As an example, the "d" state has "Xneg Xpos" in its name, because the entire robot lies along the X axis of the structure reference frame: side A is placed on the negative side of the X axis, side B on the positive. A state can be a match, a stretch or a turn. It is a match when MMIC-I is contracted and gripped onto a single face, as in "a". For matched states, the name includes only one alignment because the two sides are gripped on one face. The stretch configuration has two sides gripped onto different faces, as in "h". The turn state has A and B along different axes.

Table 5. **MMIC-I's allowed configurations.**

state	**name**
a	Xneg match
b	Yneg match
c	Zneg match
d	Xneg Xpos stretch
e	Xneg Ypos turn
f	Xneg Zpos turn
g	Yneg Xpos turn
h	Yneg Ypos stretch
i	Yneg Zpos turn
j	Zneg Xpos turn
k	Zneg Ypos turn
l	Zneg Zpos stretch
m	Xpos match
n	Ypos match
o	Zpos match

The matched states are used in the re-configuration commands, since they are considered final states. The other ones are transitional, i. e. they serve as intermediate states during re-configuration between two matched states.

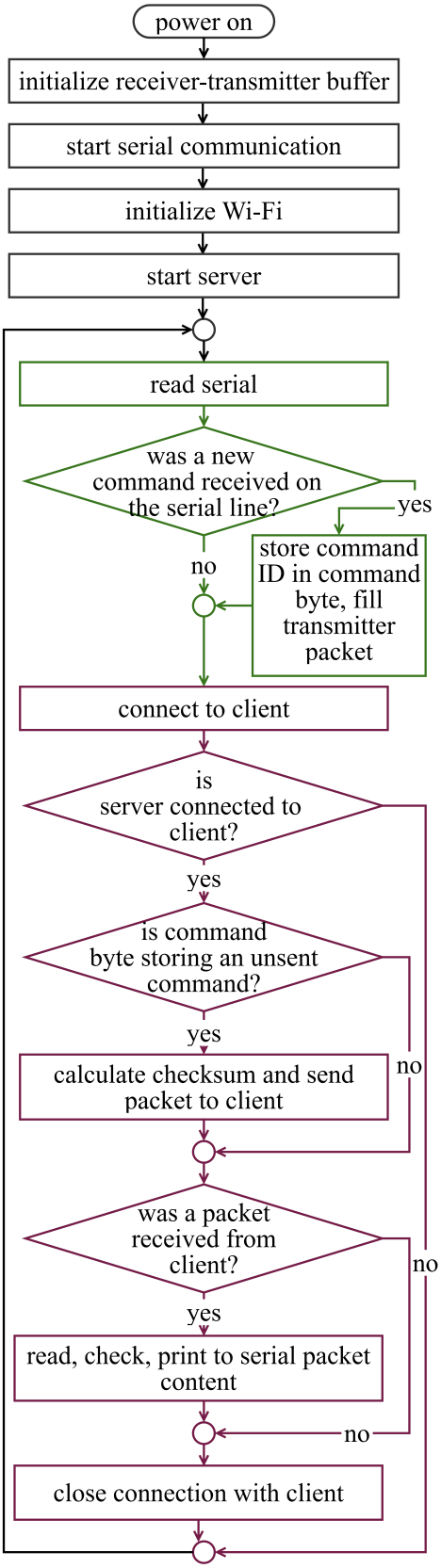

Figure 4. MMIC-I's WiFi server block diagram.

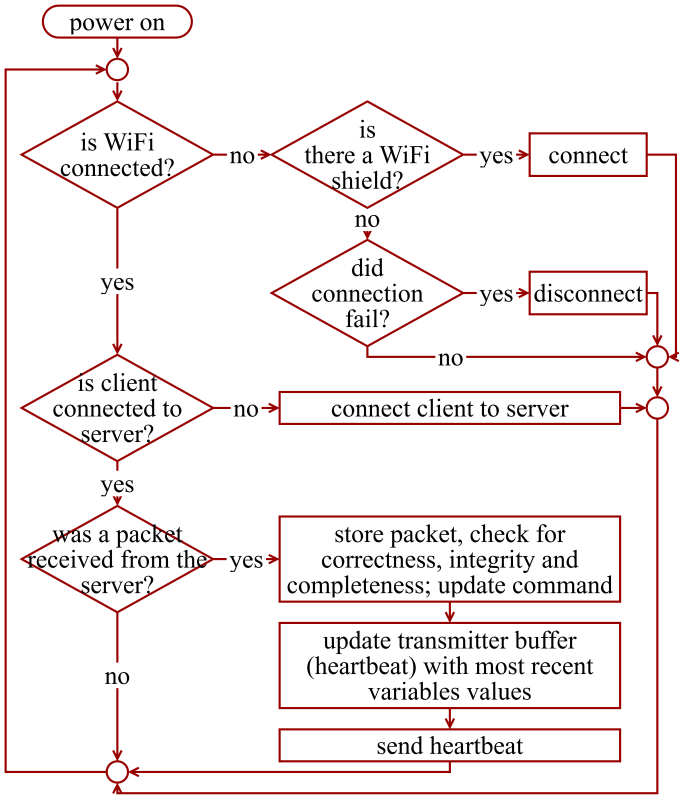

Figure 5. MMIC-I's WiFi client block diagram.

Some reconfigurations require passing by two intermediate states, some require to pass by one intermediate state, some others do not require an intermediate state. For the latter case, the reconfiguration requires a single motion primitive. While the other reconfigurations use a combination of motion primitives. All MMIC-I motion primitives are listed in Table 6. Re-configurations between matched-stretched states use Primitives from 1 to 4. The turns use from 4 to 8. The "fwd" and "back" in the primitive naming refers to towards which side, A - fwd, B - back, the motion happens. To identify which reconfiguration can be accomplished using a single primitive and what that primitive is, the reconfiguration map shown in Figure 9 can be used. The color of the intersection cell between a chosen initial state on the left column and a target state chosen on the upper row, provides this information. If the cell is grey, a single-primitive reconfiguration is not possible, though it might be possible to use multiple primitives. If the cell is colored, the transition is possible using the primitive number written in the cell. The green identifies intermediate states, while the blue final states. The black cells identify reconfiguration between same states, which is not a reconfiguration. Some cells have asterisks that indicate possible moves that the robot is capable of executing, but not necessary or used during motion generation for the demo build sequence.

To determine the sequence of primitives needed to go from a matched state to another it is sufficient to perform an iterative search on the reconfiguration map.

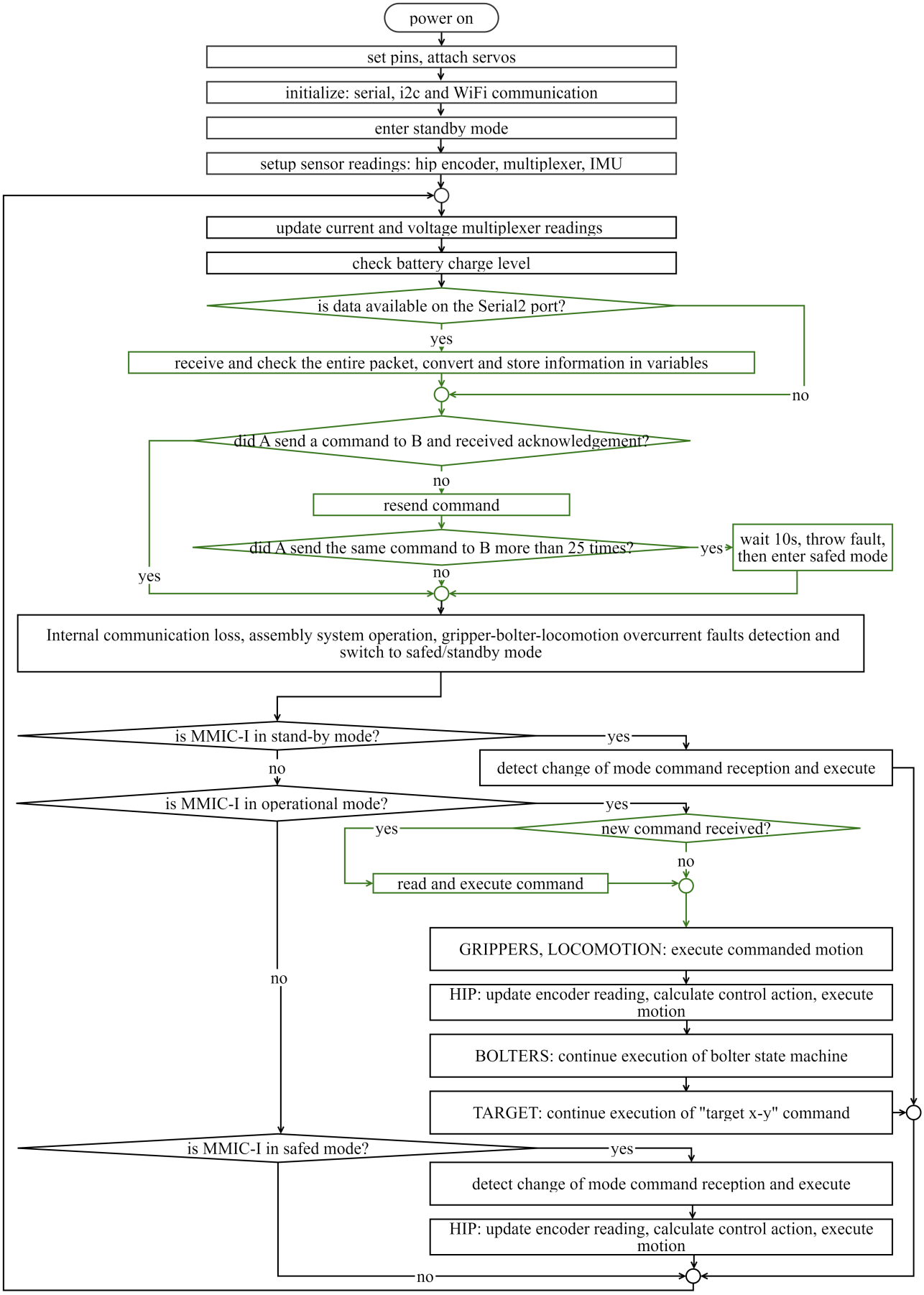

Figure 6. MMIC-I's board A control loop block diagram.

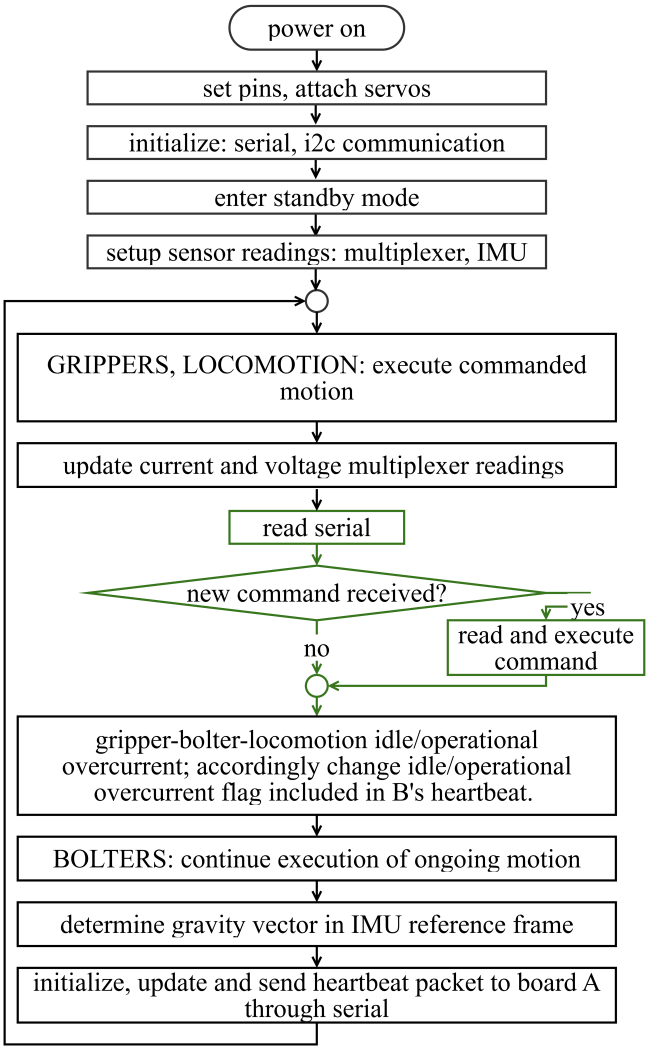

Figure 7. MMIC-I's board B control loop block diagram.

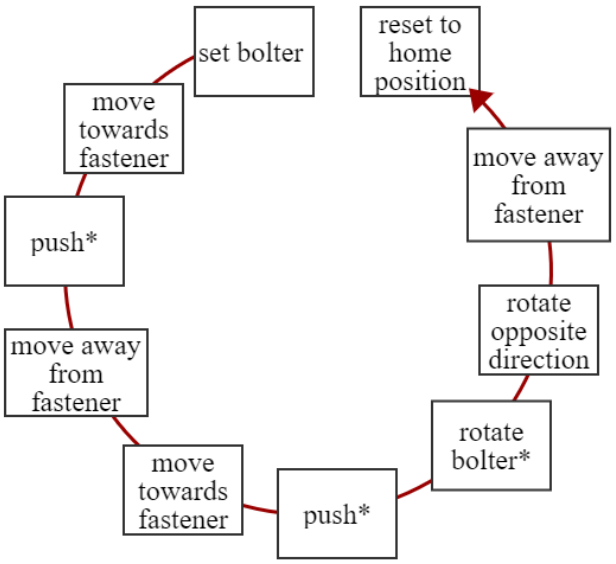

Figure 8. Bolter state machine diagram representing the series of states, from start to completion of the bolting process.

Table 6. MMIC-I's motion primitives. The first four primitives entail motion along the same direction, while the last four include a rotation of the hip, thus a change of direction.

	Primitive
1	fwdContract
2	backContract
3	fwdExtend
4	backExtend
5	fwdTurnCW
6	fwdTurnCCW
7	backTurnCW
8	backTurnCCW

7. Operating Modes And Autonomous Fault Detection

MMIC-I has three possible operating modes: standby, safed and operational mode, as shown in Table 7. Upon startup, the robot enters standby mode. When in standby mode, the voltage regulator is not powered, thus all the modules are not powered. In these conditions, MMIC-I can be manually extended or contracted to position it onto a face, since the servo motors are back-drivable. MMIC-I stays typically in standby mode when waiting for other robots to complete their motion or when it's done with a task. In this mode, MMIC-I does not accept motion commands, it only accepts commands to change operating mode. When it's time to move or bolt, the opsUI sends a command to transition to operational mode. While in operational mode, all actuators are powered, the voltage regulator is on and MMIC-I is able to accept all types of commands.

When switching to "opmode", actuators are set to go to their home position. The home position corresponds to a contracted configuration, gripped to a face. It is hence advisable to have MMIC-I already in this configuration before sending the opmode command. The autonomous fault detection is implemented in such a way that, if any fault is detected, MMIC-I goes to standby or safed mode. Safed mode differs from standby mode in that the hip and arm modules do not loose power upon the automatic switch of mode. Safed mode is designed for those faults that need immediate intervention, but the intervention shouldn't compromise the integrity of the robot or structure. For the rest, standby mode is used.

Table 8 lists the faults currently implemented on MMIC-I and the respective mode that the robot enters upon detection. After a bolting or unbolting failure is detected, MMIC-I enters standby mode because the bolting action always happens when the robot is fully gripped to a face. So cutting power to the actuators doesn't let the robot fall. Same applies to the initial state mismatch fault. This is triggered when, upon reception of a reconfiguration command, the initial state communicated within the command and the current state, calculated using IMU measurements and PWM values, do not coincide. This check is important because it validates the opsUI belief of MMIC-I's current location in the structure. The low battery fault triggers when either of the two V_{batA} or V_{batB}, fall below 10 V for more than 0.5 seconds. The assembly system operation fault is the only fault that is externally triggered. It is used when there is a fault in the entire armadas system and it is wanted that the robots enter a

Figure 9. MMIC-I's reconfiguration map. It determines if it's possible to transition from an initial state (column on the left) to a target state (row above) using a single primitive. A colored-background cell filled with a number identifies a possible transition through that specific primitive number.

Table 7. MMIC-I's operating modes and their features: how to enter and exit, what modules are powered and what commands can be received in each mode.

	Standby mode	**Operational mode**	**Safed mode**
Entry	Upon startup, fault, command	Command only	Fault, command
Power availability to modules	No	Yes	Limited: hold hip/arms position only
Command availability	Limited: transition to other mode only	Yes	Limited: to hip/arms actuators only
Exit	Command	Fault, command	Command

fault mode too. If the assembly system operation or/and the low battery faults trigger during a motion, the robot completes the motion first and then enters standby mode. All the other faults switch the mode to safed. The arm not fully contracted fault is triggered when after a contract arm command, the physical switch doesn't get pushed. This fault can only happen in the middle of a motion, so the resulting mode is safed.

With regards to overcurrent faults, there are two types: the idle check and the operational check. The idle check happens at the beginning and end of a reconfiguration motion and consists of verifying that the current draw for this module is lower that a maximum allowed when the robot is in a resting - gripped to a face - configuration. This check is aimed at spotting situations in which the robot is not properly gripping to a face, or the arm is not properly contracted because stuck, or a bolter is for some reason stuck in a fastener. If any of these conditions are true and MMIC-I starts a motion, the consequence would be that the robot falls or motors pull high current for a long time and get damaged. The operational overcurrent check is performed at all times. The threshold relative to this check is the maximum absolute allowed current value for each module. Events that might trigger the operational overcurrent are short-circuit due to cables wearing or damage, presence of an obstacle that impedes locomotion, etc.

The hip mismatch fault wants to catch malfunctioning or damage of the hip encoder, or its connection, when the robot

enters operational mode. It triggers when the reading values differ substantially from the known value of the hip angle in the contracted-gripped configuration.

Finally, the internal communication loss fault detects if the integrity of the board A-board B communication was compromised. The fault is triggered if board A stops receiving B's heartbeat for a given amount of time or if A sent the same command to B multiple times, but never got acknowledgement. Before integration with the opsUI, the firmware included an "external communication loss" to handle the WiFi malfunctioning case. In this version, MMIC-I client would send a periodic heartbeat to the server, similarly to how B sends its heartbeat to A through serial. The periodic WiFi heartbeat packet was replaced with a periodic request from the opsUI, so now this WiFi loss check is done by the opsUI itself, and not the firmware.

8. Validation Data

Validation of MMIC-I's firmware performance was performed during the ARMADAS ground demonstration of autonomous robotic assembly. For more information and data collected ducting the ARMADAS ground demo please refer to [26], [27] and [25]. The data included in this section aims at showing the efficacy of the bolter state machine and the efficiency of the WiFi communication. Figure 10 shows how many times, in percentage, it was sufficient to run the bolter state machine respectively once, twice and three times, to result in the successful bolting of a given pair of fasteners. Figure 10 also shows the percentage of failure, which indicates the number of fasteners that MMIC-I wasn't able to bolt. During the ground demo, the total number of engaged fasteners was 2524, which means that the number of fastener pairs successfully bolted after one attempt was 2307, 103 after two attempts, 28 after three, and 86 failed. One of

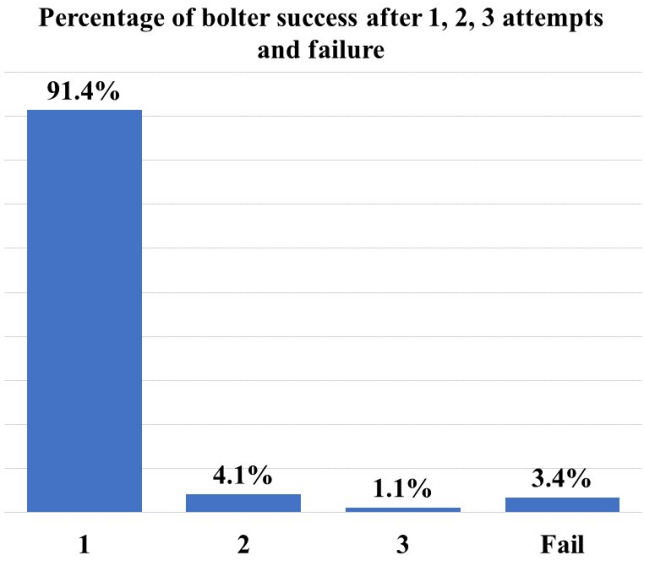

Figure 10. Percentage of bolting success after first, second, third attempt, and failure.

the main factors influencing the success of MMIC-I's bolting action is the placement accuracy of the structural unit that MMIC-I is tasked to attach to the structure. If the voxel is not well placed and there is a gap between the two faces MMIC-I is trying to connect, the fasteners are not able to interlock, even when correctly engaged. During the ground demo, this was the main reason of bolting failure.

During the ground demo, the opsUI would send through the server a command packet to MMIC-I every 1.2 seconds and wait for a feedback heartbeat packet as acknowledgement.

Table 8. MMIC-I's autonomously detectable faults and the operating mode resulting from the detection.

Fault	Triggers when	Mode
(Un)Bolting failure	After trying three consecutive times, the bolter module wasn't able to bolt successfully	Standby
Arm not fully contracted	After executing an arm contraction command, the contraction is not sensed by hardware	Safed
Bolter module overcurrent	Bolter electrical current rises above a given threshold	Safed
Gripper module overcurrent	Gripper electrical current rises above a given threshold	Safed
Locomotion module overcurrent	Locomotion electrical current rises above a given threshold	Safed
Hip mismatch	Entering operational mode, the hip angle reading differs substantially from its supposed value	Safed
Initial state mismatch	Receiving a reconfiguration command, the detected and communicated robot states do not match	Standby
Internal communication loss	Board A stops receiving B's heartbeat for a given time interval and/or when B doesn't acknowledge reception of a command repeatedly sent by A, through wired connection internal to the robot	Safed Safed
Locomotion module overcurrent	Locomotion electrical current rises above a given threshold	Safed
Low battery	Battery level falls below a certain threshold for a given time interval	Standby
Assembly system operation	Externally injected through command	Standby

If the heartbeat packet is not received within 0.7 s, the opsUI counts that as a failed communication attempt, in which MMIC-I is not responding. Figure 11 shows WiFi communication efficiency. The percentage, calculated using a moving window approach, measures how many communication attempts were successful over the last 100 attempts.

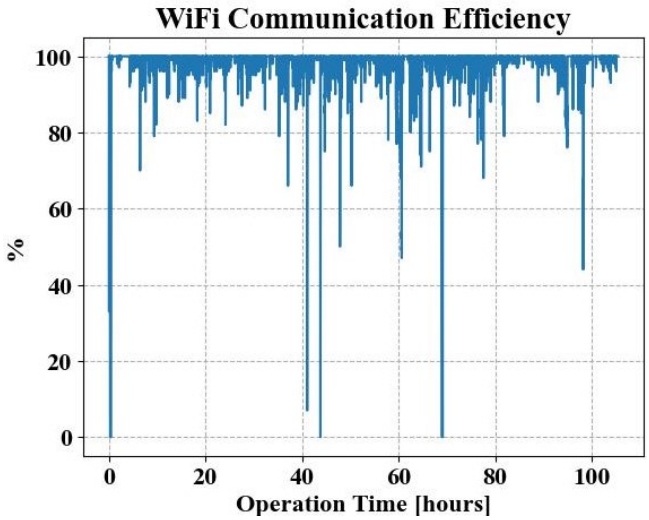

Figure 11. MMIC-I's WiFi communication efficiency.

9. Conclusions and Future Work

The Automated Reconfigurable Mission Adaptive Digital Assembly System (ARMADAS) is a project being developed in the Coded Structures Lab at NASA Ames Research Center. The Mobile Metamaterial Internal Co-Integrator (MMIC-I) is one of the relative robots, responsible for securing, or bolting, to the structure all the contact faces of a voxel that has just been placed by another robot. MMIC-I is able to locomote to a given location in the structure through contraction/extension of its two arms, hip rotation and gripping. When it reaches a face and both its gripper modules are extended, it activates the bolter module, that engages the fasteners and and locks the new voxel face to the structure.

MMIC-I's avionics includes two custom made controller boards, mounted on the robot, and a server board responsible for communication with the operation User Interface (opsUI). All the boards mount the dual-core ESP32 chip, which allows to run the WiFi communication task on a separate and dedicated core. The two controller boards on the robot mount: an Inertial Measurement Unit to detect the orientation of the robot, and a 8-channel ADC converter to measure the battery current and voltage, the regulated voltage, and all the currents drawn by the actuation modules. Other sensors are the magnetic absolute hip encoder and a switch that confirms when the arms are contracted. The sensors data are received through the i2c protocol.

MMIC-I's boards are identified by the side they belong to: A and B. Board A hosts the WiFi client, the motion primitives and motion planning, the bolter state machine, operating modes and autonomous fault detection. The WiFi client-server implementation allows the opsUI to send command packets and receive heartbeat packets containing the overall status of the robot. Board B only executes commands sent by A through a wired serial connection that passes through the robot. B sends a continuous heartbeat to A, sharing the latest current/target angles and the voltage/current measurements. Whenever A sends a command, B acknowledges reception.

MMIC-I has 15 possible states. Each motion consists of reconfiguration between states, which can happen executing between one and three motion primitives. The number and types of primitives required for each reconfiguration is dictated by a reconfiguration map, which is embedded in the firmware.

MMIC-I has three operating modes and autonomous fault detection. Upon startup, the robot automatically enters standby mode, in which all actuators are off. When ready for operation, the robot is commanded to enter operational mode. When faults that need immediate attention are detected, MMIC-I automatically enters safed mode, which has all actuators disabled except for the hip and arms, to avoid crashing. Upon detection of faults that can be addressed after completing what is currently being executed, MMIC-I automatically enter standby mode. Implemented faults include: overcurrent detection, low battery, not allowed state detection, internal communication loss, etc.

With the autonomous robotic assembly ground demonstration, the ARMADAS project is transitioning to a new phase aimed at preparing for a flight demo. This phase includes redesign of MMIC-I and its firmware to increase their flight readiness. For the flight demo, the team will consider replacing the herein described communication protocol with an international standard, such as CCSDS space packets or ECSS PUS services, and add the capability of receiving housekeeping data from the robots without the need of a heartbeat request by the opsUI. Furthermore, the team will work on increasing the overall system autonomy level.

The ARMADAS team has additionally demonstrated autonomous reconfiguration of structures trough detachment, transportation and re-attachment of voxels previously locked onto the lattice, in a different location. MMIC-I's unbolting capability allowed for this demonstration.

Acknowledgments

The authors thank the NASA Space Technology Mission Directorate's Game Changing Development program for supporting the ARMADAS project.

References

[1] J. Watson, T. Collins, and H. Bush, "A history of astronaut construction of large space structures at NASA Langley Research Center," in *Proceedings, IEEE Aerospace Conference*, vol. 7. Big Sky, MT, USA: IEEE, 2002, pp. 7–3569–7–3587. [Online]. Available: http://ieeexplore.ieee.org/document/1035334/

[2] W. Doggett, "Robotic assembly of truss structures for space systems and future research plans," in *Proceedings, IEEE Aerospace Conference*, vol. 7. Big Sky, MT, USA: IEEE, 2002, pp. 7–3589–7–3598. [Online]. Available: http://ieeexplore.ieee.org/document/1035335/

[3] R. Doggett, J. T. Dorsey, and D. S. Kang, "State of the Profession Considerations: NASA Langley Research Center Capabilities and Technologies for Large Space Structures, In-Space Assembly and Modular Persistent

Assets," Mar. 2021.

[4] R. P. Hoyt, "SpiderFab: An Architecture for Self-Fabricating Space Systems," in *AIAA SPACE 2013 Conference and Exposition*. San Diego, CA: American Institute of Aeronautics and Astronautics, Sep. 2013. [Online]. Available: http://arc.aiaa.org/doi/10.2514/6.2013-5509

[5] H. Thronson, R. Mukherjeee, L. Bowman, M. Greenhouse, H. MacEwen, R. Llc, B. Peterson, and R. Polidan, "Building the Future: In-Space Assembled Telescopes Future Assembly and Servicing Space Telescopes (FASST) Status Report," Sep. 2018.

[6] R. Mukherjee, N. Siegler, and H. Thronson, "The Future of Space Astronomy will be Built: Results from the In-Space Astronomical Telescope (iSAT) Assembly Design Study," in *70th International Astronautical Congress (IAC)*. Washington, D. C., United States: International Astronautical Federation (IAF), Oct. 2019.

[7] C. E. Gregg, B. Jenett, and K. C. Cheung, "Assembled, Modular Hardware Architectures - What Price Reconfigurability?" in *2019 IEEE Aerospace Conference*. Big Sky, MT, USA: IEEE, Mar. 2019, pp. 1–10. [Online]. Available: https://ieeexplore.ieee.org/document/8741533/

[8] N. Aeronautics and S. Administration, "Moon to mars objectives september 2022," 2022.

[9] E. National Academies of Sciences, Medicine *et al.*, "Decadal survey on astronomy and astrophysics 2020," 2021.

[10] NASA, "Automated reconfigurable mission adaptive digital assembly systems (armadas)," 2022, [Online; accessed 14-December-2022]. [Online]. Available: https://www.nasa.gov/directorates/spacetech/game_changing_development/projects/armadas

[11] K. C. Cheung and N. Gershenfeld, "Reversibly assembled cellular composite materials," *science*, vol. 341, no. 6151, pp. 1219–1221, 2013.

[12] C. E. Gregg, J. H. Kim, and K. C. Cheung, "Ultra-light and scalable composite lattice materials," *Advanced Engineering Materials*, vol. 20, no. 9, p. 1800213, 2018.

[13] B. Jenett, C. Cameron, F. Tourlomousis, A. P. Rubio, M. Ochalek, and N. Gershenfeld, "Discretely assembled mechanical metamaterials," *Science Advances*, vol. 6, no. 47, p. eabc9943, Nov. 2020. [Online]. Available: https://advances.sciencemag.org/lookup/doi/10.1126/sciadv.abc9943

[14] B. Jenett, D. Cellucci, C. Gregg, and K. Cheung, "Meso-Scale Digital Materials: Modular, Reconfigurable, Lattice-Based Structures," in *Volume 2: Materials; Biomanufacturing; Properties, Applications and Systems; Sustainable Manufacturing*. Blacksburg, Virginia, USA: American Society of Mechanical Engineers, Jun. 2016, p. V002T01A018. [Online]. Available: https://asmedigitalcollection.asme.org/MSEC/proceedings/MSEC2016/49903/Blacksburg,%20Virginia,%20USA/269061

[15] B. Jenett, N. Gershenfeld, and P. Guerrier, "Building Block-Based Assembly of Scalable Metallic Lattices," in *Volume 4: Processes*. College Station, Texas, USA: American Society of Mechanical Engineers, Jun. 2018, p. V004T03A053. [Online]. Available: https://asmedigitalcollection.asme.org/MSEC/proceedings/MSEC2018/51388/College%20Station,%20Texas,%20USA/277119

[16] N. B. Cramer, J. Kim, C. Gregg, K. C. Cheung, and S. S.-M. Swei, "Modeling of tunable elastic ultralight aircraft," in *AIAA Aviation 2019 Forum*, 2019, p. 3159.

[17] M. Carney and B. Jenett, "Relative Robots: Scaling Automated Assembly of Discrete Cellular Lattices," in *Volume 2: Materials; Biomanufacturing; Properties, Applications and Systems; Sustainable Manufacturing*. Blacksburg, Virginia, USA: American Society of Mechanical Engineers, Jun. 2016, p. V002T01A019. [Online]. Available: https://asmedigitalcollection.asme.org/MSEC/proceedings/MSEC2016/49903/Blacksburg,%20Virginia,%20USA/268940

[18] B. Jenett and D. Cellucci, "A mobile robot for locomotion through a 3D periodic lattice environment," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. Singapore: IEEE, May 2017, pp. 5474–5479. [Online]. Available: http://ieeexplore.ieee.org/document/7989644/

[19] B. Jenett, A. Abdel-Rahman, K. Cheung, and N. Gershenfeld, "Material–Robot System for Assembly of Discrete Cellular Structures," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4019–4026, Oct. 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8769886/

[20] L. Fluckiger, K. Browne, B. Coltin, J. Fusco, T. Morse, and A. Symington, "Astrobee robot software: A modern software system for space," in *iSAIRAS (International Symposium on Artificial Intelligence, Robotics and Automation in Space)*, no. ARC-E-DAA-TN55483, 2018.

[21] A. Donnan and J. Holley, "Robonet: A data bus for distributed control systems," in *2018 IEEE Aerospace Conference*. IEEE, 2018, pp. 1–6.

[22] B. Bernus, G. Trinh, C. Gregg, O. Formoso, and K. Cheung, "Robotic specialization in autonomous robotic structural assembly," in *2020 IEEE Aerospace Conference*. IEEE, 2020, pp. 1–10.

[23] E. Niehs, A. Schmidt, C. Scheffer, D. E. Biediger, M. Yannuzzi, B. Jenett, A. Abdel-Rahman, K. C. Cheung, A. T. Becker, and S. P. Fekete, "Recognition and reconfiguration of lattice-based cellular structures by simple robots," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 8252–8259.

[24] A. Costa, A. Abdel-Rahman, B. Jenett, N. Gershenfeld, I. Kostitsyna, and K. Cheung, "Algorithmic approaches to reconfigurable assembly systems," in *2019 IEEE Aerospace Conference*. IEEE, 2019, pp. 1–8.

[25] I.-W. Park, D. Catanoso, O. Formoso, C. Gregg, T. Olatunde, E. Taylor, G. Trinh, and K. Cheung, "Solle: A module transport and placement robot for autonomous assembly of discrete lattice structures (submitted paper)."

[26] C. Gregg, D. Catanoso, O. Formoso, M. Ochalek, T. Olatunde, I. W. Park, E. Taylor, G. Trinh, and K. Cheung, "Stiff and strong robotically programmable materials (submitted)."

[27] O. Formoso, G. Trinh, D. Catanoso, and K. Cheung, "Mmic-i: A robotic platform for assembly integration and internal locomotion through mechanical metamaterial structures (submitted paper)."

[28] E. Systems, "ESP32-WROOM-32E/ESP32-WROOM-

32UE Datasheet," https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf, 2022, [Online; accessed 12-October-2022].

[29] InvenSense, "MPU-6000/MPU-6050 EV Board User Guide," https://media.digikey.com/pdf/Data%20Sheets/TDK%20PDFs/MPU-6000_6050_EvalBrd_UG.pdf, 2011, [Online; accessed 12-October-2022].

[30] T. Instruments, "12-Bit, 8-Channel Sampling ANALOG-TO-DIGITAL CONVERTER with I2C Interface Datasheet," https://www.ti.com/lit/ds/sbas181c/sbas181c.pdf?HQS=TI-null-null-EDS-df-pf-null-eu, 2005, [Online; accessed 12-October-2022].

[31] A. Devices, "ADG1604 Datasheet," https://www.analog.com/media/en/technical-documentation/data-sheets/ADG1604.pdf, 2016, [Online; accessed 12-October-2022].

BIOGRAPHY

Damiana Catanoso *is a Space Robotics Software Engineer in the Coded Structures Lab, and KBR employee, working within the Intelligent Systems Division at NASA Ames Research Center. She received her B.S. degree in Aerospace Engineering from La Sapienza University of Rome in 2015 and a double M.S. degree in Space Automation and Control from Wuerzburg University and Lulea University of Technology (SpaceMaster) in 2019.*

Greenfield Trinh *is a research engineer in the Coded Structures Lab at NASA Ames Research Center. His current research activities include automated assembly of digital material structures and robotics. He received his B.S. in Physics from UC Riverside and M.S. in Aerospace Engineering from San Jose State University.*

Olivia Formoso *is a research engineer at the Coded Stuctures Lab at NASA Ames Research Center. Her research is focused on digital material structures and robotics. She received her B.S. degree in Chemical Engineering from the University of Florida and M.S. in Mechanical Engineering from San Jose State University.*

In Won Park *is a senior robotics engineer working for KBR at NASA Ames Research Center. His previous work includes design of robotic manipulators, development of avionics for space systems. He received his Ph.D. in electrical engineering at Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea.*

Taiwo Olatunde *is a Research Engineer, and KBR employee, working in the Intelligent Systems Division at NASA Ames Research Center. Her research is focused on robotics. She received her Ph.D in Aerospace Engineering from the University of Florida. She currently works in the Coded Structures Lab.*

Christine Gregg *received her Ph.D. from the Department of Mechanical Engineering at UC Berkeley, where she was a NASA Space Technology Research Fellow. Her thesis focused on digital lattice structures and lattice fracture mechanics. She works in the ARC Coded Structures Laboratory (CSL).*

Elizabeth Taylor *is the Deputy Program Manager for Game Changing Development. Just prior to this she was the Associate Chief of Mission Systems for NASA Ames Research Center's Intelligent Systems Division, as well as the Project Manager for the ARMADAS project. Ms. Taylor previously worked on numerous missions to assemble and operate the International Space Station, helped with the design for autonomous operations of the Gateway vehicle.*

Megan Ochalek *is currently pursuing her PhD degree at Stanford university with the Aeronautics and Astronautics department. She received her MS from Stanford and her BS from MIT. She is part of the pathway program at NASA Ames Research center, working in the Coded Structures Lab..*

Kenneth Cheung *is the Principal Investigator for the ARMADAS Project. He helps to run the NASA ARC Coded Structures Laboratory (CSL), which conducts research on the application of robotically programmable meta-materials to aeronautical and space systems. As a member of the NASA ARC Intelligent Systems Division and affiliate of the office of the Center Chief Technologist, he serves as a technical lead on autonomy, robotics, advanced materials, and manufacturing. He received his Ph.D. from the Massachusetts Institute of Technology.*