# CADRE MoonDB: Distributed Database for Multi-Robot Information-Sharing and Map-Merging for Lunar Exploration

Maíra Saboia
Jet Propulsion Laboratory, California Institute of Technology
Pasadena, CA, U.S.A
maira.saboia.da.silva@jpl.nasa.gov

Federico Rossi
Jet Propulsion Laboratory, California Institute of Technology
Pasadena, CA, U.S.A
federico.rossi@jpl.nasa.gov

Viet Nguyen
Jet Propulsion Laboratory, California Institute of Technology
Pasadena, CA, U.S.A
viet.t.nguyen@jpl.nasa.gov

Grace Lim
Jet Propulsion Laboratory, California Institute of Technology
Pasadena, CA, U.S.A
grace.lim@jpl.nasa.gov

Dustin Aguilar
Jet Propulsion Laboratory, California Institute of Technology
Pasadena, CA, U.S.A
dust.aguilar@gmail.com

Jean-Pierre de la Croix
Jet Propulsion Laboratory, California Institute of Technology
Pasadena, CA, U.S.A
jean-pierre.de.la.croix@jpl.nasa.gov

## Abstract

We introduce MoonDB, a distributed database designed to support cooperative robotic exploration and distributed measurements for NASA's upcoming Cooperative Autonomous Distributed Exploration Rovers (CADRE) mission. MoonDB stores, shares, and fuses information from multiple robots, providing multi-agent planning algorithms with a consistent view of the robotic team. It does so without assuming continuous communication, and significantly limiting bandwidth use through judicious selection of the state variables to share and of the sharing policy and frequency. Further, MoonDB integrates with a pose graph optimization module, allowing mapping information collected by individual robots to be re-localized a posteriori based on refined localization information. Map-merging and reconciliation of inconsistent mapping information uses OpenGL acceleration, resulting in excellent performance on embedded systems. Overall, MoonDB provides a spaceflight-quality solution to the problem of information-sharing for CADRE, addressing one of the key challenges in coordination of multi-agent systems. This paper presents MoonDB's design, the underlying assumptions guiding its development, and implementation details to enhance comprehension. Additionally, we offer preliminary experiments and analyses to validate our approach.

## Keywords

Multi-Agent Systems, Space Robots, Distributed Databases, Data Management Systems

**ACM Reference Format:**
Maíra Saboia, Federico Rossi, Viet Nguyen, Grace Lim, Dustin Aguilar, and Jean-Pierre de la Croix. 2024. CADRE MoonDB: Distributed Database for Multi-Robot Information-Sharing and Map-Merging for Lunar Exploration. In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), Auckland, New Zealand, May 6 – 10, 2024*, IFAAMAS, 9 pages.

*Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), N. Alechina, V. Dignum, M. Dastani, J.S. Sichman (eds.), May 6 – 10, 2024, Auckland, New Zealand.* 

## 1 Introduction

Multi-robot systems hold promise to unlock the answers to a number of hitherto-unexplored questions in planetary science, providing the unique opportunity to collect simultaneous measurements at spatially-distributed locations. This unique capability enables for new kinds of instruments, including multi-static ground-penetrating radars for high-resolution imaging of the Lunar surface, seismic networks to localize seismic activity, and distributed weather instruments to study atmospheric circulations on planetary bodies with atmospheres.

The CADRE mission [4, 5], funded by NASA's Game-Changing Development program, serves as a technology demonstration platform for autonomy technologies that will power such future multi-robot scientific explorers. Set to deploy three autonomous robots to the Moon's Reiner Gamma region, CADRE's objectives include the *autonomous exploration* of a region near the landing site and the execution of *a multi-static ground-penetrating radar (GPR) survey* with the least possible ground intervention. CADRE's novel multi-agent autonomy solution translates high-level commands from ground operators, such as "explore this region", into commands for the rovers' individual surface navigation stacks, e.g., "go to this location by this time", accounting for the agents' available resources and ensuring that the system avoids a single point of failure that could compromise the mission [4].

CADRE's multi-agent autonomy architecture components are distributed across multiple agents in the system. Some of these components utilize information about the whole system, while others utilize information from a single agent [4]. (a) *The leader election* module ensures continuous leadership, with a leader mediating all coordination among agents, and immediate reelection in case of leader loss [1]. (b) *The strategic planner* on the leader plans cooperative activities, considering available resources, generating coordinated task sequences for each rover. A pair of team planners on the leader refine the strategic planner's commands by adding spatial information: (c) a sampling-based motion planner computes obstacle-free trajectories that satisfy inter-rover separation constraints required for *distributed measurements* (i.e., the multi-static GPR survey) and (d) an *exploration* planner employs a divide-and-conquer approach to assign agents to regions they should explore

[8]. The (e) *agent planner* on each leader further refines instructions received from the team planners. Finally, the (f) *shared state database* samples, distributes, and aggregates various types of data collected and produced by the robots, enabling multi-agent situational awareness and effective planning. We refer the reader to [4] for a detailed discussion of CADRE's multi-agent architecture. This paper discusses the *shared state database*, or **MoonDB**, and any relevant aspects of above components (a)-(e) that influenced the design of MoonDB.

The MoonDB component not only oversees data storage and sharing across multiple robots but also performs data fusion, integrating multiple data sources (in particular, maps) to generate more consistent and useful information than that provided by any individual robot. MoonDB supplies the leader with the necessary knowledge of rovers' location, temperature, and power states, as well as individual and fused localization and mapping data. It also allows data synchronization and replication, with a specialized sharing policy for each data type. The capabilities provide by this component are crucial in scenarios where multiple robots need to work together to achieve a common goal. There are many key aspects to be considered in designing a multi-robot data management solution. The adoption and design of a solution depend on the application's requirements and will vary significantly from one system to another due to factors such as the size of the robot teams, the system's tolerance to faults, the amount of data shared, the available communication protocols, to name a few. The aspects of MoonDB were primarily shaped by the system resources, limitations, and the CADRE mission objectives.

## 1.1 Related Work

Networked robotic systems commonly involve the transfer of diverse data types. In [9], the authors categorize data transferred between robots into three classes: *key*, *mission-critical* and *time-sensitive*. Key data refers to information that requires to be shared periodically, which is crucial for nominal multi-robot mission control (e.g., telemetry). Mission-critical data includes crucial asynchronous information, where low-latency is desired, but the total and correct transfer of this information are of higher priority than its transmission time (e.g., maps). *Time-sensitive* data refers to data whose absence could potentially harm and put our vehicle integrity at risk, such as relative positioning between neighboring robots in a collision trajectory.

A number of approaches and software packages are available for merging raster maps from multiple robots (e.g., [7, 12**?** ]). However, existing approaches generally (i) offer simple merging heuristics that provide suboptimal performance in presence of noisy maps and do not accommodate multi-resolution maps, (ii) do not allow post-hoc repositioning of local maps, which is required for integration with SLAM, and (iii) rely on continuously-available communication between robots, such as in consensus-based distributed databases [**?** ] - limitations that our approach aims to overcome.

The problem of multi-agent, communication-aware merging of point cloud maps has been addressed in [3]; in contrast, our approach merges raster traversability maps, which can readily accommodate variable-resolution maps but requires complex merging policies for reconciling disagreeing information.

Our mapping methodology is inspired by the one introduced in [6], where pose-graphs and local maps were used to generate 3D scene reconstructions, specifically in a single robot scenario. Notably, our application diverges in two key aspects. Firstly, we deploy multiple robots, requiring our system to handle dynamic objects in the environment. Secondly, instead of a single resolution, our rovers generate robocentric multi-resolution local maps [10], where resolution decreases with distance. Unlike scenarios expecting identical resolutions for multiple views of the same map area, our concept of operation anticipates varied resolutions due to robots exploring different regions. Consequently, our merging policy incorporates time and resolution factors, differing from their approach, which relies on a probability function where all identical observations carry equal weight.

## 1.2 Contributions

Our contributions focus on distributed data management for multi-agent systems under intermittent, and unreliable communication, as well as limited bandwidth constraints. These contributions include:

- Data storage: enabling persistent storage of data on each robot;
- Data distribution: enabling specialized mechanisms for sharing data, ensuring accessibility of a robot's data by the entire team; and
- Data fusion: extracting knowledge from the data collected from the robot team.

Figure 1 illustrates MoonDB operations. Each rover has a database used to persist data. MoonDB controls how the data is stored, shared, and presented to the CADRE high-level planners.

## 1.3 Organization

The paper is organized as follows. In Section 2, we present the constraints and requirements that drove the design of MoonDB. In Section 3, we discuss the types of data that are stored within individual agents' databases. Section 4 presents the synchronization and replication strategy adopted to share information across the agents. Section 5 details the synchronization and merging strategy for maps generated by multiple agents, and discusses integration with pose graph optimization. Finally, in Section 6, we draw our conclusions and outline directions for future work.

# 2 Problem Formulation

We consider a small team of robots operating autonomously in an unknown environment, devoid of continuous connectivity. The primary goal is to store and relay valuable information to other robots within the system, with emphasis on presenting only the data essential for high-level multi-agent autonomy.

The CADRE team comprises three identical lunar rovers and one base station, as depicted in top image of Figure 2. The base station shares similarities with the rovers in terms of computing and wireless communication capabilities, but lacks sensors and mobility. It is specifically hardwired for communication and power to the lander, playing a crucial role in system downlink and uplink. The lunar rovers have dimensions of approximately $0.75 \times 0.5 \times 0.2m^3$ when the solar panels are deployed, and weigh less than 10 kg each. Each rover is equipped with a ModalAI VOXL sporting a Qualcomm Snapdragon 821 SoC, 4GB of RAM, and 32GB of flash

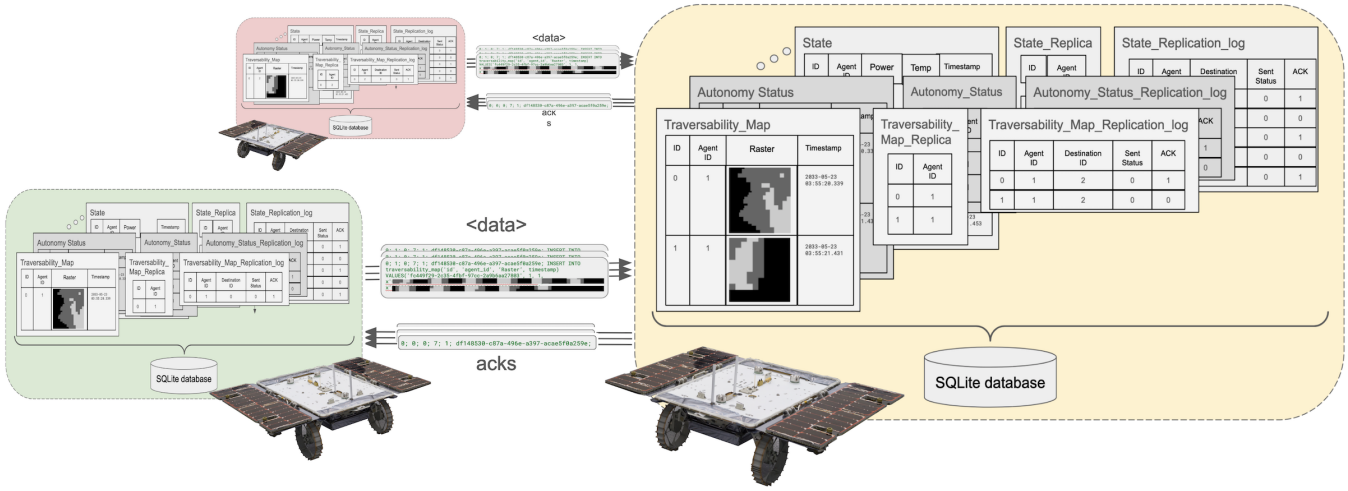



**Figure 1: CADRE MoonDB is a distributed data management solution designed for autonomous lunar exploration. MoonDB offers capabilities for data storage, sharing, and fusion, facilitating multi-agent situational awareness and effective planning.**

memory. Communication between the rovers is provided by Microhard mesh network radios providing a minimum of 1Mbps of aggregate bandwidth shared between all participating stations.

In addition to a low data rate of 1Mbps (hundred of times slower than a typical robotics lab's WiFi network), MoonDB is also not able to rely on continuous data transmission across the mesh network. The primarily reason is due to the electromagnetic interference (EMI) generated by motors causing significant interference with radio communication signals. This interference results in degraded signal quality and packet loss, posing challenges for reliable data transmission, especially during rover movement. MoonDB is intentionally designed to address this communication limitation by implementing a mechanism that selectively transfers only essential data. This strategy incorporates aspects of data stratification, prioritization and synthesis.

Using the data categorization outlined in Section 2, MoonDB is responsible for handling CADRE's *key* and *mission-critical* data. The constraints imposed by the described EMI results in MoonDB being effectively capable of transferring data only within specific time windows. Accordingly, MoonDB does not manage *time-sensitive* data; rather, this data (namely, commands to execute tasks and task sequences, and status reports regarding these tasks) is handled by a separate software that able to transmit data more reliably even under EMI disturbances, thanks primarily to lower overhead. The design of the component for time-sensitive data falls beyond the scope of this paper.

The robots within the system can assume one of four roles in the context of data distribution: base station (BS), leader, designated survivor (DS), and a standard rover. The leader is responsible for aggregating all shareable information from the robots and also hosts the active team planners. Meanwhile, the designated survivor serves as a backup to the leader. In the event of a catastrophic failure where the leader role needs to be reassigned to another robot, the designated survivor contains all team information available in the leader, enabling quicker recovery.

Locally, MoonDB addresses the challenge of persisting and managing data storage. This capability is crucial in scenarios where rovers undergo periodic shutdowns, and the ability to access data from previous wake cycles is essential. As a data distribution solution, MoonDB enables the team to share essential data among themselves and provides backup mechanisms to ensure that all vital data is readily accessible to the team planners. Additionally, MoonDB integrates local maps collected by individual rovers with their corresponding poses fetched from the pose graph to create global maps that can be used by the team planners for an effective multi-agent mission.

MoonDB is a storage, replication system for CADRE as well as a distributed mapping module. The storage and replication system supports any number of sources and targets and is asynchronous and trigger based. It allows for persistent storage of data on each robot; enables specialized mechanisms for sharing data; ensures accessibility of a robot's data by the entire team; and also facilitates the extraction of knowledge from the fused information collected from the team of robots.

## 3 Data Storage

The storage capability is implemented using an SQLite database, a widely-used relational database management system (RDBMS). In addition to essential features like being open source and stored in a single, cross-platform file on the disk, SQLite has a small memory footprint, making it well-suited for embedded systems with limited resources. MoonDB also benefits from SQLite's relational approach by giving our application the ability to create meaningful information by joining the tables. This capability is explored when leveraging pose graphs and maps to create global maps, described in Section 5.

Each data type within MoonDB is represented as a table in the database. The data classes described below are listed in Table 1.

*Robot States.* The robot state data type includes temperature, power status, operational mode (awake or sleeping), autonomous

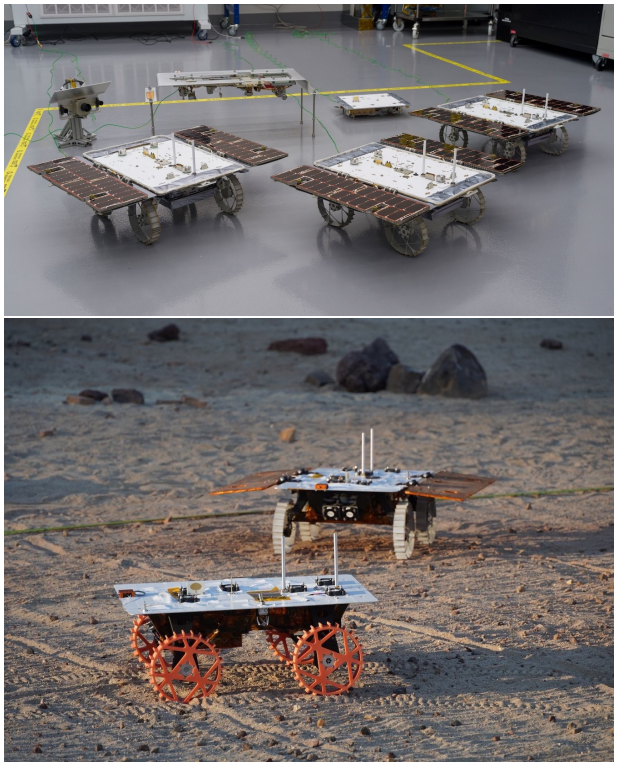


**Figure 2: CADRE teams used for testing. In the top image, from left to right there are the Situational Awareness Camera Assembly, Deployer (one of three), Base Station; and the flight rovers located in JPL's clean room. In the bottom image, two CADRE development models are shown in JPL's Mars Yard.**

**Table 1: MoonDB Database Types and Sharing Policy**

| Category | Data Type | Sharing Policy |
|---|---|---|
| **Robot State** | Temperature | latest |
| | Power | latest |
| | Awake status | latest |
| | Mode | latest |
| | Health status | latest |
| **Autonomy Status** | Exploration | latest |
| | Distributed Measurements | latest |
| **Localization** | Pose Graph: Nodes | All |
| | Pose Graph: Edges | All |
| **Mapping** | Local Traversability Map | All |

mode (current activity: exploring, executing distributed measurements, or idle), and overall system health. It is important to note that the robot state stored in MoonDB is intended for decision-making within high-level autonomy layers, and represents only a subset of the robot telemetry transmitted to the ground in terms of both data type and data volume.

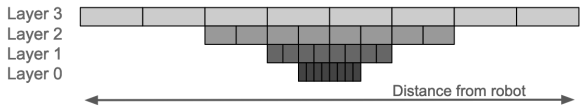



**Figure 3: Layout of multi-size, multi-resolution local traversability map**

*Autonomy Status.* The autonomy status provides information about the progress of the autonomous operations being executed. The exploration progress includes the percentage of the region explored by the robot team and the timestamp when this percentage was last measured. It also includes details about the formation completion percentage and its corresponding timestamp.

*Pose Graph.* The CADRE system utilizes a pose graph to represent the pose and observation histories of all agents. In this graph, nodes correspond to poses from individual rovers and the base station, and edges represent either the distance between consecutive nodes on the same agent, determined by visual-inertial odometry, or the distance between different agents, measured using the on-board Ultrawide-band (UWB) system.

Periodic UWB range measurements, taken before and during a drive, allow pose graph optimization (PGO) in each team planning cycle to mitigate pose drift from the previous cycle. The optimized poses obtained from PGO play a crucial role in reconstructing the global map and contribute to enhanced pose estimates used for initialization in subsequent drives.

This component requires from MoonDB the ability of storing, sharing a pose graph inserted by the agent as well as updating the same graph with information coming from PGO, and propagatingthese updates when replication (described in Section 4) occurs.

*Local Traversability Maps.* The local mapping module running within each robot uses inputs from the on-board camera to perform a 3D reconstruction and the assembly of a local traversability map [11]. This map is a pyramid representation of a multi-resolution Digital Elevation Model (DEM). The DEM pyramid structure comprises four layers, as depicted in Figure 3. The resolution of each layer is half of the resolution of the layer below it, meaning that the terrain footprint covered by one cell at the top layer is equivalent to the footprint of four cells in the layer below. Each layer of a traversability map is represented as a grid map with values representing either free (traversable) space, hazards/untraversable space (rocks of a certain size and slopes beyond a certain inclination - e.g., inside craters) or unknown. Each local traversability map is timestamped and associated with a node in the pose graph, providing the latest map pose information for that specific map.

## 4 Data Distribution

The goal of data distribution is consistently to share only the data essential for subsequent planning steps. While this means that sharing all the local maps is necessary to reconstruct a global map, it also implies that outdated data, such as the state of a rover at time $t$, when we have collected information about the state at time $t'$ with $t < t'$, must be excluded from the upcoming sharing tasks.

MoonDB uses two tables, namely `replica` and `replication_log`, to manage the data distribution. Upon insertion, data explicitly marked for sharing is included in the `replica` table. This means that not all inserted data is automatically marked to be shared. A data record marked for sharing is recorded in the `replica` table as a tuple, (`replica_id`, `agent_id`), where `replica_id` is the primary key of the table containing the data corresponding to the record, and `agent_id` represents the identifier of the agent that initially produced the data.

The record of the data already shared by an agent is logged in the `replication_log` table. In addition to the columns `replica_id` and `agent_id`, which retain their meanings from the `replica` table, three supplementary columns are used to monitor the shared data, its destination (column `destination_id`), confirmation of data transmission (column `sent_status`) to the destination, and acknowledgment of data receipt (column `ack`) by the destination.

Due to EMI concerns, data sharing does not occur continuously. Robot state and autonomy status are shared periodically with the leader agent using the "latest" sharing policy, which only shares the latest available data point for each information type; if messages are lost, the leader uses older data until synchronization succeeds. For localization and mapping information, the strategic planner initiates sharing only when the leader requires information for replanning. Replanning occurs at times when the rovers are not driving, which mitigates EMI concerns.

When the strategic planner requires MoonDB to distribute currently stored data to a specified destination (i.e., the leader), records found in the `replica` table but not yet found in `replication_log` for the specific destination are transmitted to the destination, and a record is added to the `replication_log` table with `sent_status` set to TRUE and `ack` set to FALSE. Records already present in the `replication_log` table with both `sent_status` and `ack` equal FALSE, indicating a previous attempt to share the data with no acknowledgment received, are resent, and the `sent_status` is set to TRUE. After all the data slated for sharing has been sent, the `sent_status` is reset to FALSE. Acknowledgment messages are continuously monitored, and upon receiving an acknowledgment for a particular record, that record will no longer be subject to further sharing with the same destination. On the receiving end, a received record will be appended to the data table, as well as to the `replica` and `replication_log` tables. In these tables, the `agent_id` will represent the ID of the agent that originally generated the data, and the `destination_id` will denote the ID of the agent receiving the data. The acknowledgment is transmitted back to the sender.

MoonDB offers two data distribution functionalities, *synchronization* (sync) and *replication*. Sync involves transferring the data of an individual agent to a specific destination, typically the leader, to keep it synchronized with all other agents. Replication is utilized for backing up the leader to a designated survivor, ensuring rapid and transparent recovery. When sync is performed, only agent generated by the source agent is transferred to the destination; in contrast, in replication, all the data, not only that generated by the source agent, is transferred. The table `replication_log` is used to keep track of shared information for both syncing and replication, avoiding duplication.

Each type of data has a requirement indicating whether all (not yet shared) records of that data type need to be transferred, if the most recent record is preferred, or if no data should be transferred at all. Specifically, for each data type, we can establish a *distribution policy*, with options including `none`, `latest`, and `all`. When sharing starts, data is shared based on its assigned distribution policy. While `none` and `all` policies are straightforward, indicating sharing nothing or everything, respectively, the `latest` policy alters the semantics of the *marked to be shared* option at insertion time to *candidate to be shared*, recognizing that older records for this data category are obsolete. Table 1 lists the distribution policy for each type of data managed by MoonDB.

All data exchanged between agents is encapsulated in a C++ data structure named `Message`, outlined with the following components:

- `agent_id`: an integer identifier specifying the data originating agent.
- destination_id: an integer identifier indicating the intended destination agent.
- `remaining`: an integer value representing the remaining amount of data to be transmitted in the current sharing cycle.
- `data_type`: an integer code specifying the type of the data being communicated (e.g., Traversability Map, Robot State).
- `ms_type`: an integer code identifying the type of payload attached to the message (acknowledgment or data).
- record_id: a string containing an identifier associated with a specific record.
- `sql`: a string representing a populated SQL statement holding the data to be transmitted. A SQL statement could be either an `INSERT` or an `UPDATE` statement.

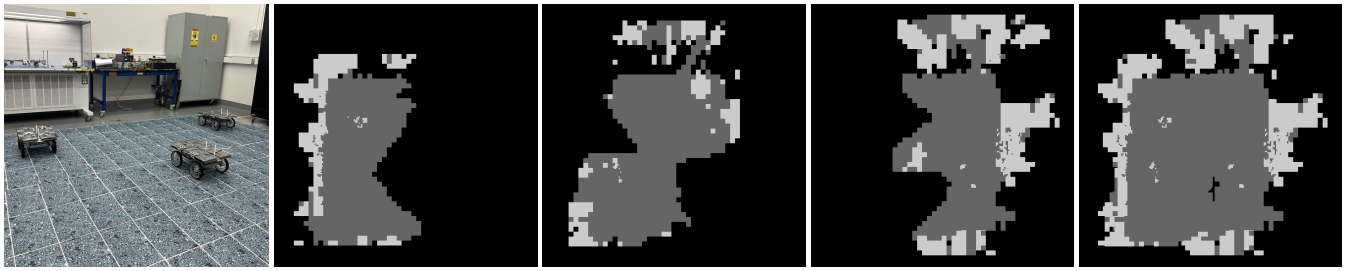
This `Message` structure offers a standardized format for packaging and exchanging data between agents, facilitating data interactions within the multi-agent system.

## 5 Distributed Mapping

Distributed Mapping integrates the local maps collected by individual robots to create a unified global map, covering the entire area explored by the robot team.

Figure 4 shows the results of merging the local traversability maps from three rovers. In the experiment displayed, robots navigated within a rectangular area inside JPL's clean room (Figure 4(a)), with various objects surrounding the region. Rover-specific merged traversability maps, utilizing local maps from single individual rovers, are shown in 4(b)-(d). The merged traversability map that combines maps collected from all the rovers is presented in 4(e). Colors represent occupancy values, with black representing *unknown*, darker gray representing *free*, and light gray representing *occupied*. The merged traversability map contains only a single layer; each cell assumes values *unknown*, *free*, and *occupied*, like the local traversability map.

CADRE's mapping architecture [11] results in significantly higher resolution information for areas closer to the rovers' paths. This crucial feature is leveraged by the merging policy, described next.



(a) (b) (c) (d) (e)

**Figure 4: Map merging. Image (a) shows the JPL clean room where the maps were collected. In images (b)-(e), colors indicate occupancy values, with black representing UNKNOWN, darker gray representing FREE, and light gray representing OCCUPIED. Images (b)-(d) depict merged maps from individual agents, while image (e) shows the combined map in the leader.**

## 5.1 Map Merging

The merging policy determines how local traversability maps are integrated into a single map. The primary challenges involve reconciling divergent information, where two or more layers of the same traversability maps or different traversability maps present conflicting values for the same region.

*5.1.1 Map Merging Challenges* We have identified three main causes in which the same region is labeled with different traversability values in different local maps:

(a) The first cause is inherent in the multi-size, multi-resolution structure of the map. Layers with varying resolutions may assign different values to the same region, namely, lower-resolution maps attribute a single value to regions that are covered by multiple cells in higher-resolution maps.

(b) The second cause involves rovers appearing within another rover's field of view. Since rovers do not directly share their locations with each other in real time [4], they cannot differentiate between rocks, craters, and other rovers in their field of view, leading to the inclusion of the latter in the local traversability maps as obstacles. Although temporarily adding the rover as an obstacle is expected behavior, this information may persist in the map if a rover does not spend sufficient time observing a region until the other rover has moved. An example of a rover marked as an obstacle in a map is illustrated in Figure 5. In the top row, the left image shows that rover 1 in rover 2's camera view, and on the right, an obstacle is added to the map (shown as the magenta cluster). In the second-row images, the obstacle is removed from the maps after rover 1 has moved out of the field of view of rover 2.

(c) The third cause is the appearance of phantom obstacles. These are false-positive obstacles generated by the local mapping components (such as caused by pose or extrinsic calibration issues, or data gaps in the inertial measurement unit [IMU] data) and are added to the maps of individual rovers. Since these are not actual obstacles, they will not appear in the local traversability maps of any other rover covering the same area, leading to conflicts regarding which traversability value should be assigned to those map regions. This issue is illustrated in Figure 6. In the top row, a small cluster represents rover 2 as an obstacle in the map, and no other obstacles are visible in the camera view. In the second row, a large phantom obstacle is added to the map, which does not correspond to any obstacle the real scene. After a while, as the robot collects more local traversability maps, the older maps with the obstacle are replaced with new ones that do not contain the phantom obstacle. Additionally, the cluster representing the real rover 1 as an obstacle has disappeared since rover 1 is no longer in the field of view of rover 2.

Issue (a) will be resolved by the merging policy, which places more trust in high-resolution information than in lower-resolution information. Both of issues (b) and (c) can be resolved by subsequent maps if the region with obstacles remains within the rover's field of view long enough to acquire new maps. This mitigation can happen when the problem generating the phantom obstacles is resolved or when the rover marked as an obstacle has moved. Importantly, the successful resolution of these issues relies on the condition that the updated maps are shared with the team.

*5.1.2 Map Merging Policy* Our merging policy is formulated on two primary assumptions. The first assumption is that high-resolution information is more accurate than data at lower resolutions. Given that the range error of stereo reconstruction scales with distance, and since the local traversability map is robocentric, the resolution is proportional to the increasing range error. The second assumption is that newer information at the same resolution is more accurate than older data, which helps remove other rovers's footprints and phantom obstacles.

Consider $v_t$ and $v_{t'}$, where time $t' > t$, as potential values for the same cell in a map; the resulting value $v_r$ in the merged map is defined as:

$$v_r = \begin{cases} v_{t'} & \text{if } v_t = \text{ unknown or } v_t = v'_t \\ v_{t'} & \text{if } v_{t'} \text{ has the same or higher resolution than } v_t \\ v_t & \text{if } v_t \text{ has higher resolution than } v_{t'} \end{cases} \quad (1)$$

This policy yields the following important behavior:

- If a region has been marked as an `obstacle` at time $t$ but is `free` at time $t' > t$ at the same or higher resolution, we mark it as free. This allows us to clear obstacles that get cleared in newer maps (e.g., rover footprints and phantom obstacles).

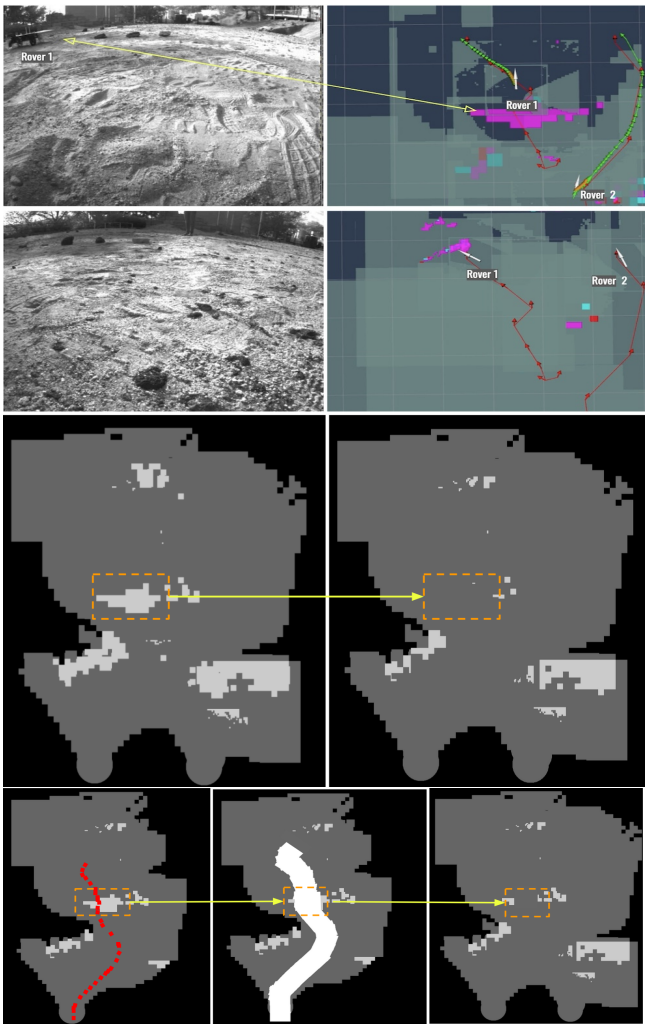



**Figure 5: Rover within the field of view of another rover. The top row of images shows how one rover is marked as an obstacle in the local traversability map of another robot (magenta). In the second row of images, the obstacle is cleared after the robot leaves the field of view. The third row shows that the obstacle is cleared from the merged map after new local maps are received; the bottom row shows the obstacle been partially cleared using knowledge of the robots' paths.**

- If a region has been marked as free at a lower resolution, but it is marked as an `obstacle` at a higher resolution (even at an older time), we conservatively mark it as an obstacle.

Additionally, we mark the footprint of all rovers' past poses (which are stored in MoonDB) as free space. If a rover has safely passed through a location, even if MoonDB marked the area as an obstacle (e.g., because the rover was observed and labeled as an obstacle by another rover), the location is marked as traversable space once pose information is synchronized.

Figure 5, in the third image-row, and Figure 6, in the bottom image-row, show comparisons between the outcomes (left images)

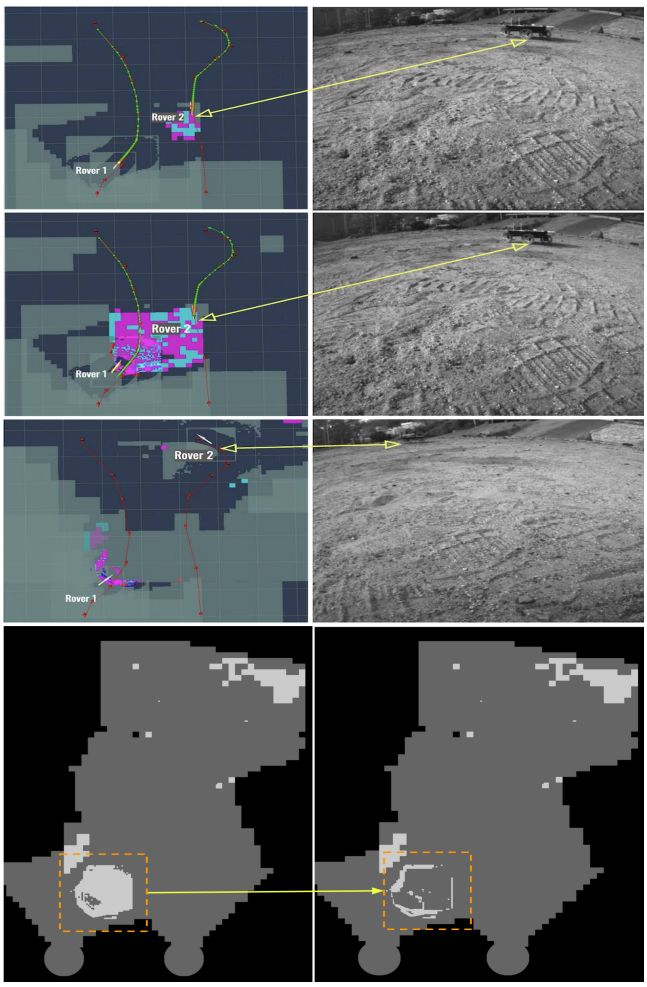



**Figure 6: Phantom obstacles in maps. The top three rows show the overlaid local traversability maps from the two robots (i.e., the inputs to MoonDB) at three consecutive time steps. The top-left image in the first row displays rover 2 as an obstacle (magenta and cyan). The second row reveals a large "phantom obstacle", likely caused by a noisy estimate of rover 1's attitude. In the third row, most of the phantom obstacle is cleared from local traversability maps as the robot's attitude estimate is corrected. In the bottom-left image, we observe a merged map with a conservative policy that only considers resolution (but not time) in its logic. Despite being cleared in the local maps, the phantom obstacle persists in the merged map. In the bottom-right image, both resolution and timestamp are considered in the merge policy, in accordance with Eq. (1). Including the timestamp allows the robots to reject older, inaccurate information, clearing most of the phantom obstacle.**

of a very conservative merging policy, where any obstacle placed in the map persists without considering time and resolution, and the merged map (right images) generated by the policy described above. The presence and absence of the obstacles, in the left and in the right merged maps, respectively, are clearly evident and highlighted with a dashed square in each merged map.

In the bottom images of Figure 5, the red path represents the trace of the rover's movement, referred to as the rover footprint. This path is converted into a free path in the merged map, based on the rover having safely passed through it. Using the previously mentioned conservative merging policy, the three images represent merged local traversability maps. Notably, an obstacle present in the left image is cleared in the right image by applying the step to remove the rover's footprint, which is shown in white in the middle image. This additional step ensures that, even if the rover's field of view shifts before new data arrives to clear an ephemeral obstacle, those obstacles can still be removed.

## 5.2 Pose Graph Optimization

MoonDB is designed for integration with a Pose Graph Optimization (PGO) module, described in [2]. The PGO module builds a graph where rovers' poses, recorded at prescribed intervals that are synchronized across all robots, are represented as nodes; visual inertial odometry (VIO) measurements connect poses collected by the same robot; and range measurements collected through ultra-wide-band (UWB) radios connect poses collected by different robots at the same time. VIO measurements and UWB ranges are collected on each rover, stored in MoonDB, and synchronized to the leader. Whenever a local map is stored in MoonDB, a corresponding PGO node is created. The pose of the map in the global frame is recorded through two transforms: one from the global frame to the robot's pose (represented by the PGO node), and another from the robot's pose to the map origin (which is determined by the mapping module, and is not altered by PGO). The PGO module periodically recomputes the set of past and current robot poses that maximizes the likelihood of the observed VIO and UWB measurements. When a rover pose node is modified by PGO, the updated pose is sent to MoonDB, which, in turn, updates the location of the map corresponding to that node (if any). This approach allows maps collected by multiple agents to be reconciled: in particular, incorporating UWB range measurements is critical to counter drift between individual robots' pose estimates, which would otherwise create multiple copies of the same obstacle whenever an obstacle is observed by multiple robots.

## 5.3 OpenGL-accelerated map merging

The map merging policy described in Section 5.1.2 is implemented in OpenGL, which allows the operation to take advantage of GPU and multi-core CPU acceleration. Individual maps are represented as monochromatic textures. A vertex shader computes the location of the maps in the world frame based on the location, orientation, and size of the map. A custom fragment shader then implements the map merging policy described in Equation (1). Maps are sorted by resolution and (for a given resolution) by time of creation before being processed by the shaders; this way, the fragment shader can implement the operations in Equation (1) with no explicit knowledge of time or resolution, relying solely on ordering of the maps. Figure 7 compares the performance of the OpenGL-accelerated implementation with a naive map merging policy. The GPU-accelerated implementation results in an order-of-magnitude speedup compared to a naive CPU implementation when merging 100 or more maps; even when executed on a CPU, the OpenGL-accelerated policy retains a five-to-tenfold advantage compared to the naive implementation.

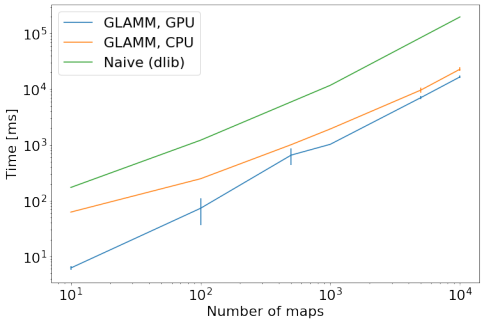



**Figure 7: Performance of OpenGL-accelerated map merging compared to a naive CPU implementation.**

## 6 Conclusions and Future Work

We introduced MoonDB, a new distributed database for storage, sharing, and merging of key and mission-critical data in multi-robot systems. MoonDB is designed to store state information from multiple robots; synchronize it to other agents over intermittent communication links with comparatively low bandwidth; and efficiently merge mapping information from multiple robots, integrating with pose graph optimization algorithms and handling noisy maps and other robots' footprints through an ad-hoc merging policy that is amenable to GPU acceleration. Tests on CADRE flight hardware and engineering models show that MoonDB performs well on embedded flight hardware, including low-bandwith communication links; is able to handle noisy mapping information produced in field tests; and, overall, successfully addresses the challenge of multi-agent, communication-aware information sharing for the upcoming CADRE lunar mission. Alongside the upcoming CADRE Mission deployment, our current emphasis is on experimentally testing and evaluating MoonDB across diverse environments and configurations (e.g., dynamically switching roles or disabling a rover), while also conducting simulated stress tests. Looking further ahead, our goal is to enhance the system's versatility and scalability, as well as integrate it with ROS (Robot Operating System).

## Acknowledgments

The research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration (80NM0018D0004). The authors thank all the staff at the Jet Propulsion Laboratory working on CADRE for their contributions to the work described in this paper. The authors also acknowledge the contributions to CADRE by the various partners, such as Motiv Space Systems, SolAero, and Dr. Laura Remond and her students at Clemson University.

## References


[1] Keenan Albee, Sriramya Bhamidipati, Joshua Vander Hook, and Federico Rossi. 2024. Lunar Leader: Persistent, Optimal Leader Election for Multi-Agent Exploration Teams. In *International Workshop on Autonomous Agents and Multi-Agent Systems for Space Applications (MASSpace)*. IEEE, Auckland, NZ. submitted.

[2] Elizabeth R. Boroson, Robert Hewitt, Nora Ayanian, and Jean-Pierre de la Croix. 2020. Inter-Robot Range Measurements in Pose Graph Optimization. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 4806–4813. https://doi.org/10.1109/IROS45743.2020.9341227

[3] Gerasimos Damigos, Nikolaos Stathoulopoulos, Anton Koval, Tore Lindgren, and George Nikolakopoulos. 2024. Communication-Aware Control of Large Data Transmissions via Centralized Cognition and 5G Networks for Multi-Robot Map merging. *Journal of Intelligent & Robotic Systems* 110, 1 (2024), 22.

[4] Jean-Pierre de la Croix, Federico Rossi, Roland Brockers, Dustin Aguilar, Keenan Albee, Elizabeth Boroson, Abhishek Cauligi, Jeff Delaune, Robert Hewitt, Dima Kogan, Grace Lim, Benjamin Morrell, Yashwanth Nakka, Viet Nguyen, Pedro Proenca, Gregg Rabideau, Joseph Russino, Maira Saboia, Guy Zohar, and Subha Comandur. 2024. Multi-Agent Autonomy for Space Exploration on the CADRE Lunar Technology Demonstration. In *2024 IEEE Aerospace Conference*. IEEE, 1–12.

[5] Game Changing Development. 2020. Cooperative Autonomous Distributed Robotic Exploration. Retrieved Jan 18, 2024 from https://www.nasa.gov/cooperative-autonomous-distributed-robotic-exploration-cadre/

[6] Bing-Jui Ho, Paloma Sodhi, Pedro Teixeira, Ming Hsiao, Tushar Kusnur, and Michael Kaess. 2018. Virtual occupancy grid map for submap-based pose graph SLAM and planning in 3D environments. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2175–2182.

[7] Jiří Hörner. 2016. *Map-merging for multi-robot system*. Bachelor's thesis. Charles University in Prague, Faculty of Mathematics and Physics, Prague. https://is.cuni.cz/webapps/zzp/detail/174125/

[8] Sharan Nayak, Federico Rossi, Grace Lim, Michael Otte, and Jean-Pierre de la Croix. 2024. Multi-Robot Exploration for the CADRE Mission. (2024). submitted.

[9] Maira Saboia, Lillian Clark, Vivek Thangavelu, Jeffrey A Edlund, Kyohei Otsu, Gustavo J Correa, Vivek Shankar Varadharajan, Angel Santamaria-Navarro, Thomas Touma, Amanda Bouman, et al. 2022. Achord: Communication-aware multi-robot coordination with intermittent connectivity. *IEEE Robotics and Automation Letters* 7, 4 (2022), 10184–10191.

[10] Pascal Schoppmann, Pedro F Proença, Jeff Delaune, Michael Pantic, Timo Hinzmann, Larry Matthies, Roland Siegwart, and Roland Brockers. 2021. Multi-resolution elevation mapping and safe landing site detection with applications to planetary rotorcraft. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 1990–1997.

[11] Lennart Werner, Pedro Proenca, Andreas Nuechter, and Roland Brockers. 2024. Covariance Based Terrain Mapping for Autonomous Mobile Robots. In *Proc. IEEE Conf. on Robotics and Automation*. IEEE, Yokohama, Japan. In Press.

[12] Zhi Yan, Luc Fabresse, Jannik Laval, and Noury Bouraqadi. 2014. Team Size Optimization for Multi-robot Exploration. In *In Proceedings of the 4th International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR 2014)*. Bergamo, Italy, 438–449.