ChatGPT

LunaBot: Autonomous Navigation Robot for Lunar Habitats (SIH 2025)

Introduction

Future lunar missions like NASA's Artemis program and ISRO's planned lunar projects are aiming for a sustained human presence on the Moon. In such missions, **autonomous robots** will be essential partners for astronauts, ensuring habitat safety and taking over routine tasks. The Moon presents a harsh and unique environment: there is **no GPS for navigation**, lighting can be extreme (blinding glare or total darkness), and lunar dust and rough terrain pose constant challenges [1] [2]. An autonomous "LunaBot" can support astronauts by navigating the habitat (both **indoors and outdoors**), monitoring environmental conditions, and performing maintenance tasks without continuous human control. The goal is to reduce astronauts' workload and enhance safety – for example, a robot can patrol for hazards or respond to emergencies while the crew focuses on critical scientific work [3]. By leveraging the Robot Operating System (ROS) and state-of-the-art sensors, it's possible to **prototype a LunaBot** that meets these needs even within a short timeline. In this document, we discuss how to build such a system, the pros and cons of various design choices, and what is needed to make the solution successful and future-ready.

Challenges and Requirements

Designing a lunar habitat robot comes with several key challenges and requirements:

- **GPS-Denied Navigation:** The Moon has no GPS satellites, so the robot must localize itself using on-board sensors and maps. It needs to create its own reference frame for position and navigation [1]. This requires sophisticated **Simultaneous Localization and Mapping (SLAM)** algorithms to build maps and track its position in real-time. High-precision mapping (within a few centimeters) is crucial for the robot to reliably navigate tight habitat modules or cluttered terrain [4].

- **Harsh Environment:** The robot must handle extreme conditions. Outdoor temperatures on the Moon swing from blistering hot in sunlight to extreme cold in shadow. Fine lunar dust (regolith) is abrasive and electrostatically sticky, which can clog mechanisms and obscure sensors [5] [6]. Lighting conditions can be challenging – near the lunar poles the sun stays low, causing long shadows and glare that can blind cameras [7]. Even indoors, the robot may transition between light and dark areas (airlocks, sunlit habitats, etc.). The design should use sensors and materials that tolerate dust and low-pressure/vacuum, and algorithms that handle variable lighting (e.g. LiDAR which works in darkness). While our prototype will be in simulation (so we don't model dust and temperature directly), it's important to keep these constraints in mind for future real deployment.

- **Constrained Spaces:** Inside a habitat, space is limited. Early lunar habitats might be the size of a large module or a couple of connected modules – "no larger than a mobile home and packed to the brim with equipment" [8]. The robot must navigate **tight corridors and cluttered rooms** without

bumping into sensitive equipment. This demands precise locomotion control, a compact form-factor, and excellent obstacle detection. Soft obstacles (like cables or tools on the floor) could be present, so a variety of sensors (not just laser scanners, but possibly cameras or bump sensors) can help detect all hazards. The robot's software should limit its speed and plan paths carefully when in confined indoor areas.

- **Rough Terrain:** Outside the habitat, the terrain will be uneven with craters, rocks, and slopes. The robot needs a robust mobility system (e.g. all-wheel drive, rocker-bogie suspension, or flexible wheels) to avoid getting stuck. Even a 20-degree slope of loose regolith can be challenging – prototypes like the MoonBot have demonstrated climbing such slopes in lunar regolith simulant when properly designed [9]. In simulation, we can model uneven ground and test the rover's stability. Hazard detection is critical outdoors: the robot should identify steep drop-offs (e.g. into craters), large rocks, or soft sand pits and plan around them.

- **Communication Delays:** While not explicitly in the problem statement, it's worth noting that if the robot were controlled from Earth, there's a ~1.3 second one-way delay. That makes real-time remote control impractical. Thus, autonomy is needed – the robot should make decisions on its own (with perhaps high-level supervision from astronauts or mission control). Autonomous navigation and fault handling are required so that the robot doesn't require constant teleoperation.

- **Power and Duration:** A practical concern (beyond simulation) is that lunar robots will run on battery or solar power, needing to conserve energy. For our prototype, energy management algorithms (like returning to a charging station) are optional, but in a real scenario this is a factor. The robot might need to shut down non-essential functions to save power or schedule tasks when power is available (e.g., solar charging during lunar day).

Understanding these challenges helps us shape the solution. We now discuss how to address navigation, mapping, obstacle avoidance, and maintenance tasks with a ROS-based approach, given a short development timeline.

ROS-Based Development Approach

Robot Operating System (ROS) will be the backbone of our LunaBot prototype. ROS is not actually an "operating system" but a middleware collection of software libraries and tools that make it easier to build complex robot behavior. We choose ROS because it provides a modular framework: we can integrate sensor drivers, mapping algorithms, and control logic as separate nodes that communicate with each other. This modularity aligns perfectly with the problem requirements of combining navigation, mapping, and monitoring functions. Importantly, ROS (and its newer version ROS 2) has a vast ecosystem of pre-built packages for SLAM, path planning, sensor fusion, and more, which accelerates development [10].

Using ROS also enables **simulation integration**. We can develop our algorithms and test them in a virtual environment using tools like Gazebo (a 3D physics simulator) or Unity-based simulators [11]. In fact, NASA and Open Robotics have demonstrated a full lunar rover simulation using Gazebo and ROS for an end-to-end mission scenario [12]. This shows that our plan to use simulation is very feasible – *"Simulators make it possible to operate in environments that have the physical characteristics of target locations without the expense of extensive physical tests"* [13]. With ROS, the same code we run in simulation can later be used on real robot

hardware with minimal changes, which makes our prototype a good stepping stone to a future physical rover.

Given our two-week timeline and lack of prior ROS experience, we will likely use high-level ROS packages rather than writing everything from scratch. The approach can be summarized as follows:

- **Platform Selection:** Use a ready robot model (for simulation) that has LiDAR, camera, and IMU sensors. For example, a **wheeled rover** model such as the Husky (Clearpath Robotics) or a TurtleBot-like platform can be used in Gazebo. This saves time in modeling dynamics. If available, use any provided model from the SIH dataset (the problem links a dataset that might include a lunar habitat model or robot URDF file). The robot model will be equipped (virtually) with:
- A **LiDAR sensor** (for distance scanning, e.g. a 2D 360° laser scanner for indoor mapping and a 3D LiDAR or multiple 2D lidars angled for outdoor terrain).
- One or more **cameras** (RGB camera for vision, possibly depth camera or stereo pair for 3D perception).
- An **IMU** (Inertial Measurement Unit for detecting acceleration and rotation, helpful for odometry).
- Wheel encoders (to measure how far wheels have turned, for dead-reckoning odometry).
- **Software Architecture:** Develop ROS nodes for key functionalities:
- **SLAM Node:** to perform simultaneous localization and mapping using sensor data (described in detail below).
- **Localization & Sensor Fusion Node:** to fuse data from wheel odometry, IMU, and possibly visual odometry into a stable pose estimate of the robot. ROS has a package called `robot_localization` that implements an Extended Kalman Filter to combine these sources.
- **Path Planning and Navigation Node:** using ROS Navigation Stack (if ROS1, the `move_base` package; if ROS2, the Nav2 stack) which handles path planning and movement. This will require a *global planner* (for finding a route on the map from point A to B) and a *local planner* (for obstacle avoidance in real-time).
- **Obstacle Detection:** This can be partly handled by the navigation stack (which uses a **costmap** that marks obstacles on the map using sensor data). We may also implement custom hazard detectors if needed (e.g., a node that analyzes LiDAR for sudden drop-offs or a camera-based classifier for terrain types).
- **Environmental Monitoring:** a node to read environmental sensors (or simulation data) such as temperature, oxygen level, pressure, etc. This node can simply subscribe to topics that provide these readings. If using Gazebo, we might simulate these sensors via plugins or by scripted events (for instance, at some point in the simulation, drop the oxygen level reading to simulate a leak).
- **Maintenance Task Coordinator:** a high-level state machine or logic node that manages routines like patrol schedules, anomaly detection, and alert signaling. This could be as simple as a Python script in ROS that sends the robot on a predefined patrol route every X hours and checks sensor thresholds to raise alarms.
- **Simulation Environment:** Set up a Gazebo simulation world that represents a section of a lunar habitat and its surroundings. For example, we can model an **indoor habitat module** (a few corridors or rooms, perhaps using simple geometry or any provided CAD model) and an **outdoor terrain** area next to it (with regolith-like ground, some rocks, small craters). Gazebo allows importing heightmaps

or 3D models for terrain – NASA's virtual moon terrain used fractal-generated heightmaps for realism [14], but for our prototype we might use a simpler heightmap or even a flat plane with obstacles for concept demonstration. The dataset link provided (`sac.gov.in/sih2025`) might contain relevant models – if it includes a lunar base CAD or a specific terrain, we will incorporate that. In absence of that, we could use existing open models (for example, a simplified lunar lander or habitat model from Open Robotics).

- **Testing and Demonstration:** We will iteratively test each component in simulation. First, get the SLAM working so the robot can map the environment. Next, test navigation by sending the robot to random goals and ensuring it avoids obstacles. Introduce some simulated obstacles or moving objects to test re-planning. Then integrate the monitoring: simulate an abnormal condition (like temperature rising beyond normal) and verify the robot's system flags an alert (this could simply log a warning or blink a light on the robot model). Finally, record a **demo video** from the simulation: this video will showcase the LunaBot autonomously driving around (both inside habitat and outside on regolith), building a map, avoiding obstacles, and responding to an anomaly (e.g., stopping and raising an alarm upon detecting a hazard or abnormal sensor reading).

By following this ROS-based plan, even with limited experience we can utilize community resources and examples. ROS has extensive documentation and there are similar projects (for Mars rovers, etc.) from which we can learn. In fact, other researchers have demonstrated ROS 2 based simulators for lunar rovers that provide visual and physics realism, confirming that our approach is on the right track [15]. Next, we delve deeper into each major subsystem: mapping & localization, navigation & obstacle avoidance, environmental monitoring, and maintenance tasks.

Mapping and Localization (SLAM in a GPS-Denied Environment)

Mapping and localization are at the heart of enabling autonomous navigation on the Moon. Since the robot cannot get its position from GPS satellites, it must build its own map of the environment and continuously figure out where it is on that map. This is done through SLAM – *Simultaneous Localization and Mapping*.

In our LunaBot, we will implement SLAM to allow the robot to traverse unknown areas and gradually produce a map (which can be used for path planning). There are several ROS-compatible SLAM algorithms we could use: - **LiDAR-based 2D SLAM:** For example, **GMapping** (GMAP), **Hector SLAM**, or **Cartographer**. These take 2D laser scan data to create a floor-plan-like map. GMapping is classic and easy to use, Hector SLAM can leverage the IMU for better accuracy (useful if no wheel odometry), and Cartographer (by Google) can do 2D or 3D mapping with loop closure. Since our habitat likely has mostly flat floors indoors, a 2D map is sufficient for indoor navigation; we can mount a 2D LiDAR at a certain height to capture walls and obstacles. - **Visual SLAM (V-SLAM):** Using camera data to map. Algorithms like ORB-SLAM2 or RTAB-Map can use a monocular or stereo camera to map and localize. Visual SLAM can even build 3D point clouds of the environment and detect visual features (like textures on walls) as landmarks. This might complement LiDAR in areas where LiDAR range is limited or when identifying specific objects. RTAB-Map is appealing because it can fuse LiDAR and camera data (RGB-D SLAM) to create dense 3D maps and has a ROS package readily available. - **3D SLAM:** If we want the outdoor mapping to include uneven terrain, a 2D map might not capture it well (e.g., a rock or a hill isn't represented in a flat 2D grid). 3D mapping using a 3D LiDAR (or a tilted 2D LiDAR scanning vertically over time) can produce a 3D occupancy grid or a point cloud map. However, 3D SLAM is computationally heavier and might be overkill for an initial prototype unless the

simulation environment demands it. We might approximate outdoor mapping with 2.5D (e.g., a costmap with elevation or just treat everything as obstacles on a 2D plane for navigation purposes).

Our approach will likely start with **2D LiDAR SLAM** for simplicity. As the robot drives, the SLAM algorithm will create a map of obstacles and free space. At the same time, SLAM provides an estimate of the robot's pose (position and orientation) within that map. This pose can then be refined by sensor fusion (combining IMU and wheel encoder data).

Sensor fusion for localization is important because each source has its strengths and weaknesses: - Wheel odometry can estimate movement but drifts over time (wheels can slip on lunar dust, or on any surface, causing odometry error). - The IMU gives orientation and acceleration data; integrating accelerations can track movement but also drifts without external reference. - LiDAR or camera-based localization (from SLAM) can correct drift by recognizing previously seen features (loop closure) or matching scan data to the map.

By fusing these, the robot gets a more robust estimate. We will use an **Extended Kalman Filter (EKF)** or Unscented Kalman Filter via ROS's `robot_localization` package to fuse (odometry + IMU + possibly visual odometry). The output is a smooth pose estimation for the robot that takes into account all sensor inputs. This fused pose is used by the navigation system to know where the robot is on the map at any time.

It's worth noting that modern developments in lunar navigation use cutting-edge LiDAR technology to achieve high precision. For example, NASA's KNaCK (Kinematic Navigation and Cartography Knapsack) is a LiDAR-based system that can create "**ultra-high-resolution 3D maps at centimeter-level precision**" and provide precise positioning in real time, explicitly to overcome the Moon's lack of GPS [1] [4]. While our robot will use more off-the-shelf sensors, the principle is the same – a dense 3D point cloud from LiDAR can serve as a local "GPS" by matching the shapes around the rover to previously mapped data. We will try to incorporate at least a basic 3D mapping for the outdoor portion (perhaps by storing a point cloud map of rocks and terrain).

In summary, the mapping and localization subsystem will enable the LunaBot to **know where it is and what the environment looks like**, a foundational capability for all other functions. Once a map is built (or as it is being built), the robot can move on to intelligent path planning and obstacle avoidance.

Obstacle Detection and Safe Path Planning

Safe navigation requires the robot to perceive obstacles and plan paths around them. In our scenario, obstacles could be: - Inside: walls, doors, habitat furniture, cables, equipment racks, or even astronauts (the robot should not run into people). - Outside: rocks, boulders, craters, slopes, equipment like solar panels or scientific instruments placed on the surface, etc. There may also be "hazards" like regolith pits or cliffs that aren't exactly obstacles (solid objects) but dangerous terrain features to avoid.

Obstacle Detection: Our LunaBot will use a combination of LiDAR and cameras to detect obstacles and hazards: - The LiDAR sensor provides a direct way to sense obstacle distances in 360° (for a 2D Lidar) or in a swath (for a 3D Lidar). Any returns that are within a certain range (say, closer than a threshold) can be considered obstacles for path planning. ROS Navigation Stack will take laser scans and mark occupied cells in a grid-based **costmap** automatically. This means as the robot drives, any obstacle the laser sees will

appear on its local map as an area to avoid. - Cameras can be used to detect obstacles that LiDAR might miss or classify the type of obstacle. For example, a camera could detect if something is a small object on the ground that the LiDAR beam passed over (like a low profile object), or to detect visual hazards like a patch of rough terrain (through texture analysis) or a drop (through stereo vision disparity or structure-from-motion). We could implement a simple vision-based classifier for outdoor terrain: perhaps identify darker patches as shadows (potential holes) or use disparity from a stereo camera to see if the ground falls away ahead. - We will also account for **negative obstacles** (like holes or drop-offs). A downward-pointing LiDAR or ultrasonic sensor could help detect if there's suddenly no ground under the robot's front – a classic problem for cliff detection. Another method is to analyze the 3D point cloud from a forward-looking depth camera to see changes in elevation. Given limited time, we might not implement a perfect cliff detector, but we should at least mark known dangerous zones on the map (e.g., if the simulation world has a crater, we can include a boundary or have the robot map it out by scanning from a safe distance).

Once obstacles are detected and mapped, the **path planning** module takes over. ROS's planning pipeline works as follows: - A **global planner** operates on the static or slowly-updated global map. We can use algorithms like A *or Dijkstra's algorithm to find the shortest path from the robot's current location to the goal location on the grid map, avoiding known obstacles. This yields a global path (a sequence of waypoints) for the robot to follow. - A local planner (trajectory planner)** runs more frequently, taking into account the latest sensor data and the dynamic state of the robot. For instance, we could use the Dynamic Window Approach (DWA) or the Time Elastic Band (TEB) planner (both available in ROS). These local planners generate velocity commands that try to follow the global path but also avoid any sudden obstacles in front of the robot. If a new obstacle appears (say a previously closed door is now open or a rock was moved by a prior action), the local planner will see it in the costmap and adjust the trajectory to avoid collision. If it cannot avoid (path blocked), it can signal the global planner to replan a new route.

The robot's speed and acceleration limits are considered in planning to ensure it can stop in time if something is detected. For example, indoors it will move slowly and cautiously. Outdoors, it might go a bit faster on open ground, but still must be ready to halt if a boulder appears. We might implement a simple **hazard stop** behavior: if any sensor (LiDAR or camera) detects an obstacle closer than a minimum safety distance, override and stop the robot immediately.

To verify safe path planning, we will test scenarios like: - Place a random obstacle (e.g., a box) in the habitat corridor and see if the robot's navigation stack dynamically goes around it. - Simulate a rock on the path outside and ensure the robot chooses a path around the rock. - If an obstacle suddenly appears (we can spawn an object in Gazebo in front of a moving robot), check that the robot stops or reroutes.

By integrating these obstacle detection and planning capabilities, the LunaBot can navigate autonomously without hitting things, which is fundamental for operating in a real habitat. The use of ROS here again gives us a big advantage: the ROS Navigation Stack is well-tested and has been used on countless robots, so we can largely reuse its logic, tuning parameters for our robot's specifics. This lets us achieve robust path planning behavior within our short timeframe.

Environmental Monitoring Capabilities

Beyond navigation, the LunaBot will serve as a **mobile environmental monitor** for the habitat. This means it will carry sensors (or have access to data) for key habitat parameters and be able to respond to

anomalies. The problem statement mentions monitoring things like *temperature and O_2 level*. These are critical for astronaut life support, so early detection of any issue is vital.

Sensor Integration: In a real scenario, the robot could be outfitted with environmental sensors: - **Temperature sensors** (ambient temperature, perhaps multiple to detect localized hot spots which might indicate equipment overheating or fire). - **Gas sensors** for **oxygen levels**, CO_2 levels, etc., to check air quality. For example, a drop in O_2 or rise in CO_2 could mean a leak or ventilation problem. - Possibly **pressure sensors** (to detect if the habitat is losing pressure), though that would likely be noticed by stationary sensors in the habitat too. - **Radiation sensor** (since lunar habitats need to monitor radiation, a robot could carry one to map radiation levels inside or outside, especially after solar events). - **Cameras** for visual inspection (the robot's cameras can double as inspection devices – e.g., checking if a gauge on an equipment panel is in the green zone, or if there's condensation on a wall, etc. This would require either an algorithm or a human looking at the camera feed remotely).

In our prototype, we can simulate one or two such sensors. Gazebo allows simulating sensors by publishing data to ROS topics. If the SIH dataset provides a structured way to simulate environment data (for instance, a file with time-series of temperature readings), we can use that. Otherwise, we can script it ourselves: for example, assume normal room temperature ~20°C and normal O_2 concentration ~21%, then introduce an anomaly event where temperature climbs to 50°C (simulate an electrical fire) or O_2 falls to 15% (simulate a leak). The robot's monitoring node will constantly check these values against thresholds.

Anomaly Detection and Alerts: The robot needs to recognize when a parameter goes out of the safe range – this triggers it to take action or at least notify humans: - If a dangerous change is detected (like O_2 dropping), the robot can raise an **alarm**. In a real system, that might be an audible siren or flashing lights on the robot, and sending a message to the habitat's central computer. In our simulation, we might simply log a warning message (and we could simulate a buzzer sound or a light by toggling a model's LED if available). - Optionally, the robot could autonomously respond. For example, if temperature is high in one area, it could go to that location to provide visual feed or try to find the cause. However, fully automating the response might be complex; initially, we at least have it notify and maybe home in on the problem area (if we can localize where the anomaly is strongest, like move around to find the source of a leak).

A basic demonstration might be: during a routine patrol, the robot's O_2 sensor reading suddenly drops abnormally (we simulate this). The robot detects this, stops its patrol, and triggers an **alert mode** – in the demo video we could show it turning on a red light (if our robot model has one) or printing "ALERT: Oxygen level low!" on screen, and then moving to a safe zone or returning to a home position. This would show that the LunaBot can serve as an automated safety system.

Monitoring also includes **regular data collection**. The robot can periodically log readings at various points in the habitat (maybe some spots have different microclimates). For instance, it might report that the oxygen is evenly distributed, or that a particular corner is hotter (maybe a machine there is running). Over time, such data helps track the habitat's status. In future expansions, this data could be fed into an AI system to predict issues (predictive maintenance), but that is beyond our initial scope.

In sum, environmental monitoring is relatively straightforward to implement in ROS (just reading sensor topics and comparing to thresholds), but it is a vital component that turns our robot from just a navigator into a **maintenance and safety assistant**. It aligns with the vision that *"you'd really like the habitat to handle*

as much as possible on its own, which means robots doing that work" [3] – including checking life support systems so that astronauts don't have to manually inspect every gauge daily.

Maintenance Tasks: Routine Patrols and Alerts

To tie everything together, we will implement some **maintenance task behaviors** on the robot. Two specific tasks mentioned are routine patrol and alert signaling:

- **Routine Patrol:** The idea is that the robot will autonomously do rounds of the habitat, much like a night watchman or a roving sensor platform. We can program a patrol route consisting of multiple waypoints that cover key areas of the habitat. For example, a simple patrol might be: Start at charging station → Go through Corridor A → Check Airlock entrance → Circle around the exterior habitat module → Return to start. Using the navigation capabilities we discussed, the robot can follow this route automatically. We can schedule this patrol at certain intervals (say once every few hours, or triggered by certain events). ROS could utilize a scheduler node or we could simply loop it in code with delays. On each patrol, the robot can perform checks: read sensors (as discussed), maybe use its camera to take pictures of certain equipment (for later human inspection), etc. This kind of routine task demonstrates the robot's ability to operate **independently over long durations**, which is crucial for reducing astronaut workload. As one example from research, autonomous robots could replace astronauts in tasks like checking filters or looking for any out-of-place items [3]. Our prototype will showcase a simplified version of that concept.

- **Alert Signaling and Response:** When the robot detects an anomaly or hazard, it must signal an alert. There are a few types of alerts:

- **Environmental Alert:** e.g., atmosphere issues, as described. The robot should signal to humans if something is wrong. In a fully developed system, this might tie into the habitat's alarm system. In our demo, as noted, we can simulate it via on-screen messages or a siren sound.
- **Obstacle/Hazard Alert:** Suppose during navigation the robot encounters an unexpected hazard it cannot navigate (say a large unseen pit or a blocked path). It should inform operators that it's stuck or that a path is blocked. This can be as simple as logging "Path blocked, please assist" in our prototype, but an advanced robot could, for instance, deploy a drone or camera to further inspect, or call for help to remove the obstacle (if humans are around).
- **Maintenance Alert:** If the robot was checking some equipment and finds a reading out of normal range (like a pressure gauge via camera OCR, or an electrical panel reading), it could flag that for the crew. This is somewhat speculative for our prototype, but conceptually possible with image recognition.

For the SIH competition deliverables, we would script a scenario in the simulation to **demonstrate both patrol and alert**: 1. The robot starts a patrol at time 0. It navigates through a few pre-set checkpoints, showcasing its autonomous navigation in both indoor and outdoor settings. We ensure the video captures it avoiding an obstacle or two. 2. During the patrol, an anomaly is triggered – e.g., the habitat's temperature sensor goes above threshold or a section of the habitat is marked as "leaking" (we could simulate a drop in O_2). The robot's monitoring node catches this. 3. Immediately, the robot stops its normal routine. Possibly it says (via a text-to-speech or a console message) "Alert: Environment anomaly detected." It then either returns to a safe base or moves toward the affected area (depending on what behavior we choose to implement). In the demo, it might be more visual to have it move to a certain location (like going to the

airlock if a leak is at the airlock) and then shining a light (if the model can do that) or just indicating this is the location of the problem. 4. The video would end with the robot in alert mode, proving that it not only can do tasks when all is normal but can also handle abnormalities proactively.

Such a demonstration would fulfill the requirement of showing **"the robot performing autonomous navigation and anomaly detection in the simulated habitat"** (as the problem expects in the demo video).

It's important to note that making a habitat robot truly autonomous for maintenance is a **future-facing challenge** – researchers are actively working on robots that can repair habitat components, replace filters, etc., which requires manipulation abilities in addition to navigation [3] [16]. Our LunaBot at this stage is mainly a mobile base with sensors (no advanced manipulators), so its maintenance tasks are limited to inspection and monitoring. In the future, adding a robotic arm would allow it to physically fix issues (e.g., tighten a bolt, swap an air filter) which is part of the vision for resilient extraterrestrial habitats. For now, we ensure it can at least inform humans or other systems about issues, which is the first step toward a fully autonomous maintenance regime.

A prototype lunar rover navigating a simulated lunar surface environment. Test facilities like LUNA (run by DLR/ESA) allow robots to practice mobility and autonomy under Moon-like conditions on Earth. Such testing validates designs for rough terrain, dust, and slopes before actual deployment [17] [9].
Figure: Example of a lunar robot maneuvering in a Moon simulation hall.

Simulation and Testing Strategy

To achieve all the above within two weeks, a clear simulation and testing strategy is essential. We plan to take an incremental approach: 1. **Unit Testing of Components:** First, test each subsystem in isolation. For instance, run the SLAM node with the robot driving around a single room to ensure it produces a coherent map. Test the navigation stack in a simple world to tune parameters (for example, tune the obstacle avoidance radius, max velocity, etc., by having the robot move toward a wall and verifying it stops or turns). Test the monitoring by feeding fake data to the monitoring node (e.g., simulate a rising temperature) and see that it triggers an alert message. 2. **Integration Testing:** Once individual parts work, integrate SLAM, localization, and navigation. Start in a known map if needed (to simplify early tests). We might initially give the robot a predefined map of the indoor area to see if localization alone works, then introduce SLAM later. We'll use RViz (ROS's visualization tool) extensively here – it can display the robot's perceived map, the planned path, and sensor readings in real time, which helps debug issues (for example, if the LiDAR isn't registering an obstacle due to configuration, we'd see it). 3. **Simulation Environment Setup:** Build the full simulated environment with both indoor and outdoor sections. This might involve combining models: e.g., place a habitat module model adjacent to a patch of rocky terrain. Ensure that the coordinate system and units are consistent (Gazebo uses meters, gravity can be set to Moon's gravity ~1.62 m/s^2 if we want to be fancy, though that mainly affects driving physics like jumps or traction). We will likely slow the simulation speed if needed to observe behavior, and then can run in real-time for the final video. 4. **Scenario Testing:** Run through the demonstration scenario multiple times to fine-tune. We might discover, for example, that the robot gets confused at the transition of indoor to outdoor (because lighting or terrain change causes SLAM to lose track). If that happens, possible solutions include using fiducial markers at the transition (like an AR tag on the airlock door that the robot recognizes to reset its position) or simply driving slower and allowing the SLAM to catch up. Another scenario: the robot might identify a false obstacle due to sensor noise (common in simulation), so we adjust filtering parameters. 5. **Recording the Demo:** Use Gazebo's

built-in recording or an external screen capture to get the footage. We'll annotate the video to highlight when the robot is autonomously deciding (perhaps overlay text like "Mapping in progress" or "Alert triggered"). It's crucial the video clearly shows the robot *acting on its own*: e.g., show the RViz view side-by-side or insets that illustrate the robot's map and planned path, to convince judges that it's autonomous and not remote-controlled.

Throughout testing, we'll document any limitations or assumptions. For instance, maybe our SLAM map isn't globally accurate (common in short-term SLAM runs); we might note that longer-term, the robot should use loop closure or better algorithms to minimize drift. Or if we find the robot can't climb a steep slope in simulation, we acknowledge that and plan a route that avoids slopes >15°. These details show understanding of the system's capabilities and limits.

It's encouraging to note that others have successfully validated similar systems in simulation. Researchers report that high-fidelity simulators can realistically model phenomena like wheel slip, sensor noise, and even the visual appearance of lunar terrain, which is promising for our testing [18] [15]. While our simulation might be simpler, the core idea is the same: test as much as possible virtually, because it's far cheaper and safer than a physical test, especially for a Moon environment which we can't physically replicate easily. NASA's own use of Gazebo-based lunar simulators underscores the value of this approach [12].

Pros and Cons of the Proposed Solution

Like any engineering approach, our ROS-based LunaBot design has its advantages and disadvantages. It's important to evaluate these, as they inform how we might improve the system and what trade-offs we're accepting.

Pros

- **Autonomy Reduces Human Workload:** The primary benefit is that an autonomous robot can handle routine and risky tasks so that astronauts don't have to. This directly addresses the problem statement's goal of reducing human workload for maintenance. Robots can patrol at all hours, including lunar night or when the crew is resting, providing continuous monitoring of the habitat's status. This enhances safety – for example, a robot might catch an issue in the middle of the night before it becomes a catastrophe, whereas humans might be asleep. Over the long term, such robots are *"indispensable pioneers"* for sustaining a lunar base [19].

- **Modular and Rapid Development via ROS:** By using ROS and existing algorithms (SLAM, Navigation Stack, etc.), we dramatically shorten development time. We don't need to reinvent path planning or basic mapping; instead we tune and integrate them. This is a huge pro given our 2-week timeline. ROS also ensures **modularity** – each component (navigation, monitoring, etc.) can be developed and tested somewhat independently, and we can swap out algorithms if something isn't working (for instance, try a different SLAM package if one fails). In the future, this modularity means we can add capabilities (like a manipulator arm or swarm coordination) by plugging in new ROS nodes.

- **Simulation-First Testing:** Developing in simulation is cost-effective and safe. We can test edge cases (like a sudden obstacle or sensor failure) easily in software. This means our prototype will be relatively well-validated for a concept demo. It also allows flashy demonstrations (like showing a 3D

map building in real-time) that make our submission more convincing. And since ROS code from simulation usually works on real hardware, our solution is *future-proof* in that sense – it could be deployed on a real rover platform with minimal changes once hardware is available [12].

- **Sensor Fusion Increases Reliability:** By combining LiDAR, cameras, and IMU data, the robot is not reliant on one sensor alone. This is a pro because on the Moon any single sensor can have issues (cameras can be blinded by glare or darkness, LiDAR can potentially be confused by certain reflective surfaces, etc.). Fusion means if one data source is momentarily bad, the others keep the robot on track. For example, if dust obscures the camera, the LiDAR still guides the robot; if the LiDAR loses track on a featureless flat plain, the visual SLAM might pick up lunar landmarks in the distance. This robustness is crucial for real missions and improves our simulation as well (less likely to get "lost").

- **Scalability to Multi-Robot Systems:** Our design could be extended to multiple robots working together. While not in the initial scope, ROS can handle multi-robot communication (especially ROS 2 with DDS which is built for distributed systems). In future, a team of LunaBots could coordinate – some could specialize in scouting outside, others in internal cleaning – and share data. This **swarm or team approach** has been noted as a way to cover more ground and provide redundancy [20] [21]. Starting with a ROS-based system sets the stage for that because we can use ROS multi-master or similar setups later.

- **Supports Future Enhancements (AI, Vision):** Once the basic navigation works, we can integrate more advanced AI algorithms relatively easily. For example, we could add a neural network for object detection (to identify equipment or tools lying around), or integrate a reinforcement learning model for improved path planning. ROS has packages for connecting to machine learning frameworks. This means our architecture is flexible for future research and improvements, ensuring it can evolve into a sophisticated system – essentially making it **"a part of the future"** lunar habitat operations, not just a one-off demo.

Cons and Challenges

- **Steep Learning Curve:** ROS and autonomous robotics have a steep learning curve, especially for newcomers (and we have no prior ROS experience). The two-week timeframe is very tight for learning, developing, and integrating everything. We risk spending a lot of time debugging ROS setup issues (like dependency conflicts, understanding ROS topics/services, TF coordinate frames, etc.) instead of actual functionality. Mitigation: use as many existing demos and templates as possible (for instance, use the TurtleBot3 navigation demo as a starting point, then adapt to our scenario).

- **Complexity and Reliability:** An autonomous system is complex, and things can go wrong. SLAM might fail (e.g., the robot might get lost if it revisits a place and the map doesn't align), or the navigation stack might oscillate or get the robot stuck in a corner (known issues like the "corner trap" when the local planner can't get out). Tuning these is tricky and might be hard to perfect in two weeks. In a real lunar mission, reliability is paramount – any software crash or misbehavior could be mission-threatening. Our prototype likely won't be as reliable as needed for a real deployment, though that's expected at this stage.

- **Sensor Limitations:** Each sensor has downsides which could be cons in design:

- LiDAR works great for mapping, but high-end LiDARs are heavy and power-hungry (though newer ones are better). Also, standard LiDAR can be affected by fine dust and by harsh sunlight (like traditional LiDAR might get "noise" from sun interference). The Aeva 4D LiDAR addresses that with FMCW technology [7], but that's cutting-edge (and expensive). In simulation we don't have noise by default; in reality, it's a concern.
- Cameras need light; in dark lunar craters or during lunar night, they may need infrared or not work at all. Also, processing camera data (for V-SLAM or object detection) requires significant computing power and can introduce latency.
- IMUs drift over time (especially affordable ones). On a long traverse, if SLAM fails to correct, the position could be off.

- These limitations mean the robot might have difficulty in certain scenarios (e.g., featureless areas with no visual or Lidar features). We accept that our solution might struggle if the environment is too sparse or too dynamic. Careful setting of the environment in simulation can avoid exposing these weaknesses in the demo, but they exist inherently.

- **Computational Load:** Running SLAM, path planning, sensor fusion, and monitoring together can be heavy for an onboard computer. In simulation on a PC it's fine, but a real rover would need a robust computer or even multiple processors (one for vision, one for control, etc.). High power computing generates heat and consumes more power – a problem in space. If our code isn't efficient, a small real robot might not handle it in real-time. This is something to consider for the future; one may need to optimize algorithms or use specialized hardware (like FPGAs or edge AI chips) for things like vision processing [11].

- **Navigation Limits (No Dynamic Obstacle Handling beyond avoid):** Our current plan handles static or slowly moving obstacles by avoidance. If something truly dynamic occurs (imagine another robot or astronaut moving quickly in front of our robot), ROS navigation might not react fast enough, or could even collide if the obstacle doesn't show in the laser in time. More advanced techniques (like predictive avoidance or cooperation) are not implemented due to time constraints. This is a con for real-world safety. However, in our simulation we can control the scenario to avoid high-speed interactions.

- **Dependency on Simulation Accuracy:** For the prototype, we rely on simulation fidelity. If the simulation physics or sensor models are not accurate, our testing might miss some issues. For example, Gazebo's default wheel friction might not simulate wheel slip on slopes well; so our robot might appear to climb fine in sim, but a real rover might struggle in powdery regolith. There's also the chance of "simulation bias" – overfitting our solution to the simulated environment specifics. We have to be careful to keep the logic general. We might even intentionally add noise to sensors in simulation to test robustness.

- **Maintenance Task Scope:** Currently, our robot doesn't *fix* things, it only reports or observes. This is a limited form of maintenance. In a sense, one could argue it's more of a mobile inspector than a repairman. The real value in a habitat might come when robots can actually swap out filters, tighten bolts, deploy emergency patches on walls, etc. [3]. To do that, you need robotic arms, dexterous manipulators, and very advanced planning (for manipulation tasks). We don't have that in this project (due to time and complexity). So one could cite that as a "con" or limitation – the solution addresses navigation and monitoring, but not physical repair actions. However, since the problem

statement didn't explicitly require a manipulator, we are focusing on the mandated scope. It's just something to note for future enhancement.

In summary, our solution's pros lie in addressing the problem effectively with current tools and setting up a foundation for future expansion, while the cons are mostly about the difficulty of perfecting such a system quickly and the inherent limits of sensors and autonomy in unstructured environments. Being aware of these helps ensure we communicate realistic expectations in our SIH submission (e.g., we'll clarify which challenges are solved in prototype vs. which would need more R&D). The pros and cons analysis ultimately reinforces why our approach (ROS-based autonomous robot) is **promising** for lunar habitat support, yet also why careful engineering and future iterations will be needed to make it mission-ready.

Future Outlook: LunaBot's Role in Lunar Habitats

Looking beyond the initial prototype, it's clear that autonomous robots like our LunaBot will play a **pivotal role in the future of lunar exploration and habitation**. Space agencies and researchers envision a Moon where robots and humans work hand-in-hand (or rather, hand-in-end-effector) to build and maintain a sustainable presence.

In the near future, as Artemis and other programs establish base camps, we expect to see specialized robots for tasks such as construction, mining, transport, and habitat upkeep [22] [23]. Our LunaBot concept is directly aligned with this trajectory: - **Habitat Maintenance and Repair:** Projects like NASA's RETHi (Resilient ExtraTerrestrial Habitats Institute) are actively developing robots that can perform repairs and maintenance when astronauts are away or occupied [3]. The logical extension of our LunaBot is to add manipulators so it can physically intervene – for example, patch a minor meteorite puncture or swap out a clogged air filter autonomously. This could be life-saving in scenarios where humans can't respond in time. - **Infrastructure Deployment:** As mentioned in the Amphenol Aerospace review, robots will help assemble habitats, set up solar panels, and do the heavy lifting of construction [22]. A navigation robot with environmental awareness could serve as a **mobile supervisor or assistant** on a construction site – carrying tools, holding parts, scanning the area for alignment and safety. If we equip future LunaBots with appropriate tools, they could assist in assembling modular habitat components delivered from Earth or built via 3D printing from regolith. - **Exploration and Science Support:** While astronauts focus on high-level science, robots can scout ahead. A LunaBot might roam beyond the base perimeter to map terrain, measure radiation in different locations, or even collect samples from just outside the airlock for astronauts to analyze. Multi-robot systems like NASA's **CADRE rovers (Cooperative Autonomous Distributed Robotic Exploration)** are being designed to work as a team to survey large areas without direct human control [21]. In the future, our habitat robot could network with such scouting robots – for instance, receiving alerts from an external rover about incoming dust storms or interesting findings, and then helping to secure the habitat or notify the crew.

- **Human-Robot Interaction:** For robots to be part of daily life in a lunar base, they must safely coexist with humans. This means good interaction design – possibly voice commands, gesture recognition, or simply predictable and transparent behavior so astronauts trust the robot. Our prototype doesn't delve deeply into HRI (Human-Robot Interaction), but we foresee adding features like voice alerts ("Oxygen level dropping, please check life support!") or the ability for a crew member to summon the robot ("LunaBot, come to Module 2"). NASA has already put free-flying robots (like **Astrobee** drones) on the ISS to monitor inventory and do inspections; a wheeled or crawling robot in

a Moon base could be similarly invaluable. We should keep interfaces open (for example, a ROS bridge to a speech recognition system or a tablet control app) to accommodate such enhancements.

- **Reliability and Autonomy Improvements:** Over time, the autonomy algorithms will be refined. Things like **machine learning** could allow the robot to better detect anomalies (e.g., using anomaly detection AI on sensor data to catch subtle deviations), or to adapt to changing environment (like adjusting its navigation policy if wheels keep slipping on a certain slope – perhaps learning to avoid that path). The current state-of-the-art already includes AI that allows robots to make real-time decisions in complex environments [24]. As this tech matures, a future LunaBot might have a high-level "brain" that optimizes habitat operations – deciding when to recharge, when to do tasks, and coordinating with other systems, all with minimal human input.

- **Collaboration with Space Agencies:** If we look at organizations like ISRO (the sponsor of this problem) and NASA, there is a clear push towards robotics in space. ISRO has already experimented with robotic rovers in its Chandrayaan missions (though those were teleoperated). The next logical step is autonomous rovers for upcoming lunar missions or even for orbital outposts like Gateway. Our project could potentially align with those efforts. For example, the technologies we integrate (SLAM, sensor fusion) could feed into how ISRO designs navigation for a future lunar rover or how they manage indoor robotics for a planned habitat module. Demonstrating a working prototype in SIH 2025 could catch the interest of ISRO's Department of Space, possibly opening doors to further development or collaboration.

To truly be "part of the future," LunaBot must evolve from a prototype to a robust system tested in analog environments on Earth and eventually on the Moon itself. Testing on Earth might involve lunar analog sites (like volcanic fields or desert habitats) – similar to how DLR's LUNA facility tests robots on simulated regolith [17], or how researchers take robots to lava tubes on Earth to mimic lunar caves [25]. These tests will validate the robot's performance in real-world conditions of dust, gravity (partial gravity testing is tricky on Earth but there are ways like parabolic flights or suspension rigs), and communication limits. Our current simulation would need to be complemented by such field tests to prove the concept fully.

In conclusion, the LunaBot we design in 2025 could be the forerunner of a class of autonomous lunar habitat robots that, within the next decade, **ensure that lunar bases are safe, efficient, and sustainable**. The work we do now – even as a simulation prototype – contributes to that vision by tackling the integration of navigation, environment sensing, and autonomy. As one article aptly put it, *"as NASA seeks to return humans to the moon this decade, robotics will be fundamental to the success of future lunar missions… Their use will only continue to grow as we expand our footprint on the moon."* [26] Our LunaBot is a step toward that reality, demonstrating how an autonomous robot can navigate without GPS, perceive its environment, and act as the tireless guardian of a human outpost on another world.

Conclusion

Building a ROS-based autonomous robot for lunar habitats is an ambitious but achievable goal – even on a compressed timeline – thanks to modern robotics software and a clear understanding of the challenges. We have outlined how to construct **"LunaBot"** by integrating SLAM mapping, sensor fusion for localization, obstacle avoidance, and environmental monitoring into a cohesive system. In just two weeks, focusing on simulation and reusing proven ROS components will allow us to create an initial prototype and demo video showcasing the concept in action.

The proposed solution navigates both indoor habitat modules and the lunar surface outside, avoiding obstacles and hazards while carrying out routine patrols. It monitors vital environmental parameters and can alert the crew to anomalies, thereby enhancing safety. We discussed the **pros** – such as reduced astronaut workload, use of ROS for fast development, and the potential for future scalability – as well as the **cons** – including technical complexity and sensor limitations that must be managed.

Crucially, this project is not just a one-off demo; it is aligned with the future direction of space exploration. Autonomous robotic assistants are **poised to become part of the standard toolkit for lunar (and Martian) habitats**, taking on everything from scouting and science to construction and emergency response [22] [3]. By succeeding in this problem statement, we contribute to that forward-looking vision. Our LunaBot prototype, if developed and refined beyond the hackathon, could evolve into a real system safeguarding astronauts and expanding humanity's reach on the Moon.

In summary, it is indeed possible to build the LunaBot using current technology: **sensor fusion and SLAM algorithms to substitute for GPS, robust planning for navigation, and smart integration of monitoring tools for maintenance** – all tied together under the flexible ROS framework. With careful planning, testing, and an eye on both the technical details and the broader mission goals, we can deliver a solution that not only addresses the hackathon requirements but also stands as a step toward the future of autonomous space robotics. The success of this project will lie in thoughtful execution and iteration, and if done right, LunaBot could very well become an inspiration for how we manage and live in extraterrestrial habitats in the years to come.

References:

1. Allan, M. et al., *"Planetary Rover Simulation for Lunar Exploration Missions,"* NASA Ames Research Center & Open Robotics – *IEEE Aerospace Conference 2019*. (Describes a high-fidelity Gazebo-based lunar rover simulator using ROS, emphasizing the value of simulation for mission planning) [12] [27].

2. Aeva Press Release (2022), *"Aeva 4D LiDAR Helps NASA Map the Moon,"* – Discusses NASA's KNaCK LiDAR backpack enabling precise navigation and mapping on the Moon in absence of GPS [1] [4].

3. Eric Hastings (2025), *"NASA's AI Robots Scout Lunar Sites for Artemis Moon Bases,"* WebProNews – Highlights the importance of autonomous robots with advanced sensors in Artemis, for scouting and mapping in harsh lunar terrains [19] [20].

4. Amphenol Aerospace Blog (2023), *"Lunar robotics will shape the exploration of the Moon,"* – Provides an overview of the expected roles of robotics in upcoming lunar missions, from exploration rovers (e.g., CADRE) to construction and maintenance, and notes challenges like lunar dust [21] [23].

5. Harvard SEAS News (Leah Burrows, 2024), *"Robots to help with human habitation in space,"* – Describes the RETHi project on resilient habitats, focusing on robots doing maintenance tasks so that habitats can recover from disruptions. Notably mentions using robots for routine tasks (replacing filters, cleaning) to free astronaut time [3].

6. DLR/ESA News (2025), *"Robots train in LUNA for their work on the Moon,"* – Discusses tests of lunar robots (MoonBot, Scout) in a simulated lunar environment (LUNA facility). It demonstrates

capabilities like modular reconfiguration and rugged mobility (climbing 20° slopes, surviving drops) which inform design considerations for lunar rovers [17] [9].

7. Pieczyński et al. (2023), *"LunarSim: Lunar Rover Simulator Focused on High Visual Fidelity and ROS 2 Integration,"* Applied Sciences Journal – Introduces a ROS 2-based simulator with Unity for vision-based lunar rover development, showing the use of realistic simulation for SLAM, visual odometry, and hardware-in-loop testing [15] [10].

8. Additional references from ROS documentation and NASA materials on autonomy (not explicitly cited above) were consulted to validate the technical approach (e.g., ROS Navigation Stack Guide, NASA's Lunar Surface Innovation Initiative roadmap, etc.), ensuring our solution aligns with current best practices and future needs.

---

[1] [4] [7] Aeva 4D LiDAR Helps NASA Map the Moon – Aeva
https://www.aeva.com/press/aeva-4d-lidar-helps-nasa-map-the-moon/

[2] [19] [20] [24] [25] NASA's AI Robots Scout Lunar Sites for Artemis Moon Bases
https://www.webpronews.com/nasas-ai-robots-scout-lunar-sites-for-artemis-moon-bases/

[3] [8] [16] Robots to help with human habitation in space
https://seas.harvard.edu/news/2024/01/robots-help-human-habitation-space

[5] [6] [21] [22] [23] [26] Lunar robotics will shape the exploration of the Moon | Amphenol Aerospace
https://www.amphenol-aerospace.com/blog/lunar-robotics-will-shape-the-exploration-of-the-moon

[9] [17] Robots train in LUNA for their work on the Moon
https://www.dlr.de/en/latest/news/2025/robots-train-in-luna-for-their-work-on-the-moon

[10] [11] [15] [18] LunarSim: Lunar Rover Simulator Focused on High Visual Fidelity and ROS 2 Integration for Advanced Computer Vision Algorithm Development
https://www.mdpi.com/2076-3417/13/22/12401

[12] [13] [14] [27] Planetary Rover Simulation for Lunar Exploration Missions - NASA Technical Reports Server (NTRS)
https://ntrs.nasa.gov/citations/20190027571