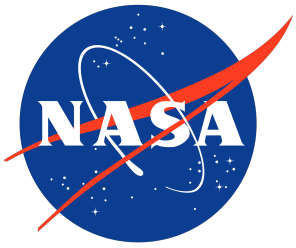
NASA/TM–20250009477



Development of a JPL Open Source Rover into an Autonomous Research Platform

Zane Brotherton
Rensselaer Polytechnic Institute

September 2025

NASA STI Program Report Series

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI Program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI Program provides access to the NTRS Registered and its public interface, the NASA Technical Report Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

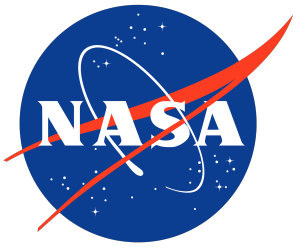
- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers, but having less stringent limitations on manuscript length and extent of graphic presentations.
- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.
- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.
- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.
- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.
- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

For more information about the NASA STI Program, see the following:

- Access the NASA STI program home page at `http://www.sti.nasa.gov`
- Help desk contact information: `https://www.sti.nasa.gov/sti-contact-form/` and select the "General" help request type.

NASA/TM–20250009477



Development of a JPL Open Source Rover into an Autonomous Research Platform

Zane Brotherton
Rensselaer Polytechnic Institute

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

September 2025

Acknowledgments

The author wishes to thank Dr. Natalia Alexandrov for her guidance and mentorship, and NASA Langley Research Center's Autonomous Integrated Systems Research Branch (D332) for their continued support.

The use of trademarks or names of manufacturers in this report is for accurate reporting and does not constitute an official endorsement, either expressed or implied, of such products or manufacturers by the National Aeronautics and Space Administration.

Available from:

NASA STI Program / Mail Stop 150
NASA Langley Research Center
Hampton, VA 23681-2199

Abstract

This report details the implementation of autonomous navigation capabilities on the Jet Propulsion Laboratory (JPL) Open Source Rover (OSR). The enhanced system performs real-time trajectory generation, obstacle avoidance, and simultaneous localization and mapping (SLAM), converting the manually controlled platform into a fully autonomous ground vehicle. The report provides practical development insights for students and beginning developers.

Contents

1 Introduction **4**

2 Problem Definition and Analysis **5**
2.1 Objective Specification 5
2.2 Development Constraints 5

3 Process Outline **5**

4 System Architecture and Sensor Simulation **6**
4.1 Initial System State 6
4.2 Software Framework Updates 6
4.3 Sensor Configuration and Exploratory Simulation 6

5 SLAM Framework Implementation **7**
5.1 Framework Selection 7
5.2 RTAB Mapping 7
5.3 Navigation Framework (Nav2) 9

6 Hardware Integration **9**
6.1 Computational Requirements 9
6.2 Package Distribution 10
6.3 Containerization and Deployment 11
6.4 Hardware Configuration 11

7 Testing and Analysis **12**
7.1 Physical Testing 12
7.2 Performance Analysis 12
7.3 Cost Analysis 12

8 Conclusions & Future Work **13**

References **14**

A OSR Low-Level Motor Abstraction **15**
A.1 Terminology 15
A.2 Variable Definitions 15
A.3 Robot Dimensions 16
A.4 Motor Calculations 17

B Sensor Simulation in Gazebo **20**
B.1 RealSense Description Package 20
B.2 RGB Camera 20
B.3 Depth Camera 21

C RTAB-Map Configuration **22**

D Nav2 Configuration **23**

E Additional Hardware Configuration **24**
E.1 RealSense D435i 24
E.2 Madgwick Filter 24
E.3 Twist Mux 25

F Enumerated bringup instructions **26**
F.1 Simulation bringup 26
F.2 Hardware bringup 27

G Bill of Materials **28**

List of Figures

1 Simulated model with mounted D435i 7
2 Point cloud generated by RTAB-Map using a singular frame 8
3 An example of a geometrically sparse environment 8
4 Computing Platform Diagram. Solid lines denote power connections, dotted lines denote wired data connections, and dotted lines with arrows denote wireless data connections. 10
5 Package Distribution Diagram 10
A1 OSR Dimension Illustration (Credit to JPL) 16
A2 OSR Motor Control Scheme 17
B1 RGB Camera Simulation 20
B2 Depth Camera Simulation 21

1 Introduction

This paper shares insights gained from building a functional autonomous system. Designing such a system from scratch may be a difficult task for a beginner. However, a judicious integration of existing software and hardware can provide students and other beginning developers with a practical platform for experimentation and future developments.

The JPL Open Source Rover (OSR) [1] is a subscale implementation of the six-wheel rover design used to explore the surface of Mars. Constructed entirely from consumer-off-the-shelf (COTS) components, the OSR is intended to function as both an educational platform for robotic exploration, and a research platform for use on rugged terrain. The open source architecture of the rover ensures complete accessibility to the design and is designed to be relatively inexpensive, costing approximately $1,600 USD at the time of this report. The platform supports expandability through attachment points positioned at various locations across the rover chassis.

Key features include:

- **Rocker-Bogie Suspension System**: Ensures continuous ground contact across all six wheels when traversing obstacles.
- **Differential Pivot**: Distributes weight between the sides of the rover while climbing.
- **Six Wheel Ackermann Steering**: Enables reduced turning radius and in-place rotation capabilities.

The rover has ten motors: six for propulsion and four vertically mounted servos for steering. Its sensing capabilities are limited to motor encoders, allowing for basic wheel odometry.

The acquisition of an OSR as a research platform was motivated by several operational requirements. The OSR has enhanced performance in challenging environments compared to existing research platforms and aligns with space exploration mission requirements. These advantages, combined with cost effectiveness and modularity, support rapid prototyping and multi-agent autonomous operations.

The OSR is the focus of this report; however, the general principles of conversion are applicable to other devices. Readers are expected to possess basic understanding of programming and robotic systems.

2 Problem Definition and Analysis

2.1 Objective Specification

The primary objective was to demonstrate autonomous navigation capabilities for the OSR. To address this task, the system would have to satisfy the following operational requirements:

1. **Autonomous Navigation**: The rover must navigate from its current position to any arbitrary two-dimensional pose, using real-time trajectory generation.
2. **Obstacle Avoidance**: The system must avoid collisions with static and dynamic obstacles by maintaining continuous environmental perception.
3. **Scalability Requirements**: Solutions must be transferable across multiple similar rover systems.

2.2 Development Constraints

Several development constraints shaped the implementation approach.

1. **Timeline Limitation**: The duration of this project was approximately ten weeks, which imposed the requirement of rapid prototyping and failure recovery.
2. **Cost Effectiveness**: The development was to minimize the amount of funds spent to ensure reproducibility both within NASA and in the larger open source community for the OSR.

These objectives and constraints required integration of established navigation and Simultaneous Localization and Mapping (SLAM) frameworks to achieve reliable autonomous operation in ten weeks.

3 Process Outline

This section outlines the high-level process, following the order of implementation. Individual items are described in the subsequent sections.

1. Conduct literature search and assess the existing and affordable software and hardware necessary for the implementation.
2. Simulate proposed software and sensor configuration to determine viability.
3. Determine edge computing resources required by the software solution.
4. Expand hardware with additional resources, integrating into the rover's power and network systems.
5. Determine the package distribution across computing resources, as well as a deployment strategy.

6. If necessary, construct non-simulation versions of software.

7. Implement a viable solution on existing rover hardware and test it in a practical environment.

4 System Architecture and Sensor Simulation

4.1 Initial System State

The baseline OSR software [2] did not include sensors beyond motor encoders. The existing code base offered an abstraction for lower-level motor control (described in Appendix A), interpreting commanded linear and angular velocities into individual motor commands—a standard abstraction for ground vehicles.

The baseline system supported direct remote control functionality, serving as the foundation for autonomous capabilities to be developed.

4.2 Software Framework Updates

The original code base was built using Robot Operating System 2 (ROS2) [3], version Foxy. The JPL team also provided Unified Robot Description Format (URDF) files, 3D models, and a simulation implementation using the `ros2_control` framework. Migration to ROS2 Jazzy, the latest long-term supported version, required modifications to ensure compatibility with Gazebo Harmonic [4], the corresponding simulation framework. The modifications included updates to package and plugin names, as well as some parameters used in simulation.

A fundamental limitation of URDF resides in its strictly tree-based structure, resulting in an inability to describe closed-loop systems. This prevented accurate modeling of the differential linkage present on the OSR. Other description formats, such as Gazebo's native SDF, could be used to amend this; however, such a conversion was not within the scope of the project, and the differential and rocker-bogie joints were set as fixed constraints for simulation.

4.3 Sensor Configuration and Exploratory Simulation

The available sensors included Intel RealSense D435i [5] and D405 [6] cameras, both stereo depth cameras. D435i incorporates a six-axis Inertial Measurement Unit (IMU), an active infrared dot projector, two infrared stereo cameras, and a dedicated RGB camera. D405 is optimized for short-range precision, with two infrared stereo cameras positioned closely together. While further sensor acquisition of modules such as LiDAR was possible, it was crucial to verify whether other sensors were explicitly necessary. We decided to explore sensor configurations via simulation prior to mounting on the physical robot.

Sensor simulation in Gazebo (detailed in Appendix B) allowed for rapid validation and positioning optimization without manufacturing overhead. The simulation used Gazebo camera plugins for the color and depth sensors and an Intel RealSense description package for positioning and model information. Color and depth cameras were configured with matching specifications to the RealSense data sheets, anchored

to the same transform frame to simulate the `align_depth_to_color` parameter necessary for Red, Green, Blue, Depth (RGBD) SLAM frameworks.

The IMU simulation was also conducted through Gazebo sensor plugins, positioned at the RealSense IMU transform frame to maintain hardware fidelity.

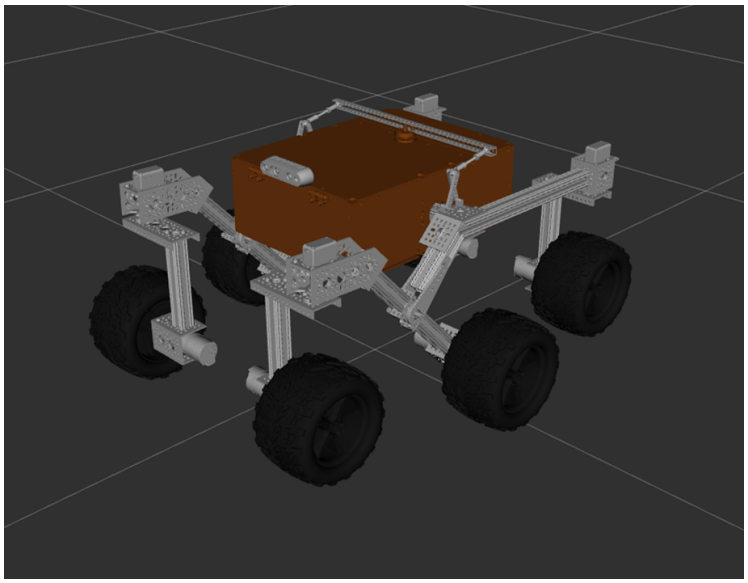


Figure 1: Simulated model with mounted D435i

With sensor simulation validated, the focus shifted to implementing SLAM capabilities for autonomous navigation.

5 SLAM Framework Implementation

5.1 Framework Selection

The `slam_toolbox` [7] package can effectively convert RGBD point clouds into a LiDAR scan format. However, this option was rejected because of the framework's limitation in the use of RGBD cameras. Instead, we selected Real-Time Appearance-Based Mapping (RTAB-Map) [8], a dedicated RGBD SLAM framework.

5.2 RTAB Mapping

RTAB-Map was selected based on several key capabilities:

1. **RGBD Optimization**: Native support for RGBD cameras, generating a dense 3D point cloud that includes obstacles at various elevations.
2. **Visual-Inertial Odometry (VIO)**: Precise positioning via camera feed analysis, tracking the displacements in detected features.
3. **Sensor Flexibility**: Optional LiDAR and multi-camera support for future sensor suite enhancements.

4. **Native ROS2 Integration**: RTAB-Map was designed to work with ROS2, with existing launch files and a node-based architecture.

The single-camera implementation presents limitations, including a restricted field of view, leading to incomplete environmental perception without a previously generated map or manual exploration. However, RTAB-Map successfully generated dense and accurate mapping of high-fidelity environments, verified using the Gazebo Ionic World model provided by OSRF.

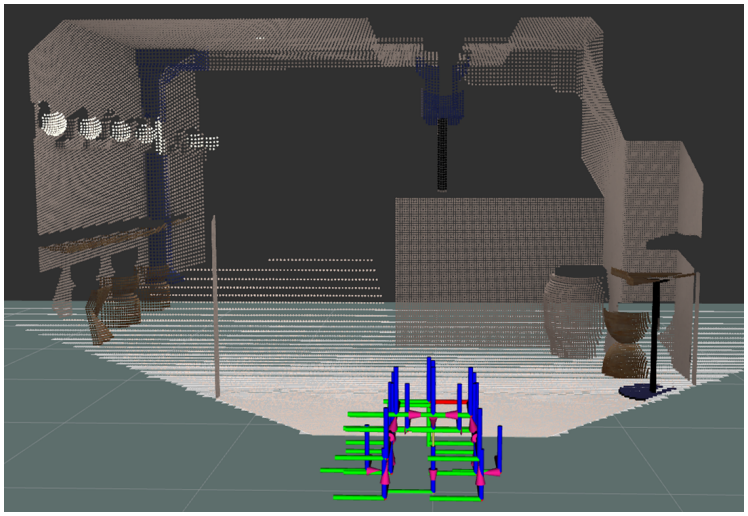


Figure 2: Point cloud generated by RTAB-Map using a singular frame

Testing revealed that RGBD frameworks that use VIO require environments with sufficient visual complexity for reliable operation. Simple geometric environments created in simulation proved insufficient for a successful mapping.

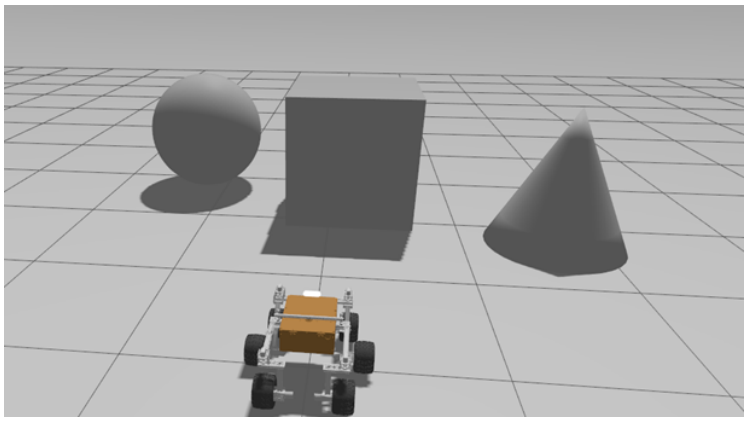


Figure 3: An example of a geometrically sparse environment

A detailed RTAB-Map configuration can be found in Appendix C.

5.3 Navigation Framework (Nav2)

Despite significant performance overhead, Navigation 2 [9] (Nav2) was selected as the navigation stack based on its industry standard capabilities, expandability, and implementation efficiency.

The map generated using RTAB-Map was continuously integrated into Nav2 as a cost map. This enabled successful navigation through the Ionic World environment, with pose commands provided via RViz, a ROS2 live visualization utility.

Despite field-of-view limitations, this sensor configuration proved sufficiently functional to enable hardware development: the rover was able to reliably navigate in our environment to arbitrary poses while avoiding obstacles. Following successful verification in simulation, we proceeded to implement the autonomous navigation system on the physical rover.

Detailed Nav2 configuration can be found in Appendix D.

6 Hardware Integration

6.1 Computational Requirements

Guidance from experts and lessons from previous projects indicated the existing Raspberry Pi hardware would be inadequate for simultaneous RTAB-Map, Nav2, and control system operation. A distributed computing architecture was required.

An NVIDIA Jetson Nano and a network switch were integrated into the existing power distribution network, with the stock 5V regulator providing adequate current capacity for the additional computing power. A RealSense D435i was connected to the Jetson Nano via USB and affixed to the front of the rover with a 3D printed mount.

To optimize computational resource distribution, visualization software was offloaded to a 'control laptop' with a discrete graphics card. The XBOX controller used for manual control maintained direct connection to the Raspberry Pi, enabling operation without laptop dependency during non-autonomous operations.

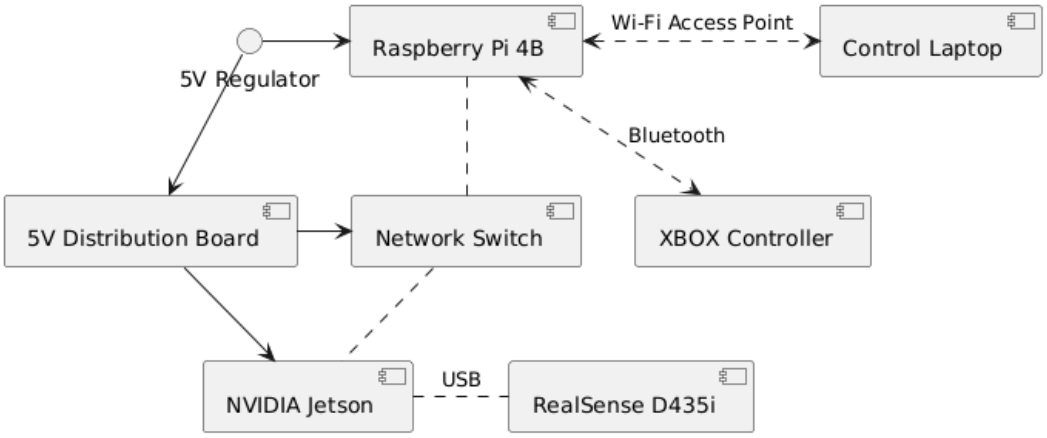


Figure 4: Computing Platform Diagram. Solid lines denote power connections, dotted lines denote wired data connections, and dotted lines with arrows denote wireless data connections.

6.2 Package Distribution

The OSR system now consists of six Colcon packages distributed across three computing platforms.

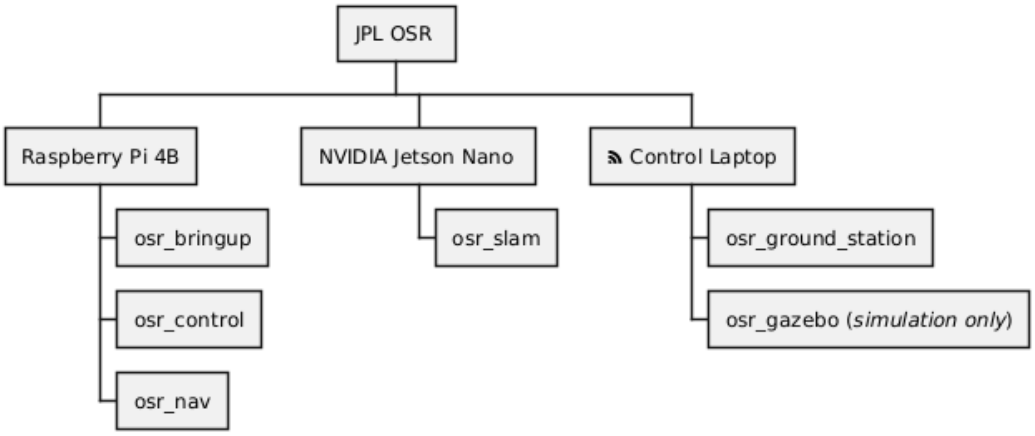


Figure 5: Package Distribution Diagram

Raspberry Pi 4B:

- `osr_bringup`: ROS2 initialization and launch sequence.
- `osr_control`: Robot control and low-level motor commands.
- `osr_nav`: Navigation 2 framework functionality.

NVIDIA Jetson Nano:

- `osr_slam`: RealSense processing, IMU filtering, RTAB-Map, and URDF publishing.

Control Laptop:

- `osr_ground_station`: Visualization software including RViz for autonomous motion commands.
- `osr_gazebo`: Gazebo and simulation functions. (simulation only)

6.3 Containerization and Deployment

Docker[10] implementation was selected to optimize development cycles and ensure a consistent hardware configuration across platforms. The system employs dedicated Docker images for each computing platform, built via a shared Docker Compose file and deployed via bash scripts, using SCP and SSH for the startup configuration. An additional docker image was created for simulation purposes. All packages run independently of the rover hardware.

Docker containerization provides several advantages:

- Consistent development environment across different hardware architectures.
- Simplified debugging through standardized testing configurations.
- Cross-compilation support for ARM64 packages on AMD64 development systems.
- Offline operation capability through SSH-based deployment.

Docker containers operate with privileged access and host networking to ensure device detection and network communication functionality. Automatic container start on rover boot minimizes the requirement for manual intervention.

6.4 Hardware Configuration

The Raspberry Pi was configured as a wireless access point for laptop connectivity, enabling ROS2 topic routing for visualization and Nav2 pose commands. This configuration has the potential to support extended-range operation and swarm management applications in the future.

Additional hardware configuration steps can be found in Appendix E.

7 Testing and Analysis

7.1 Physical Testing

The time limitations did not allow for extensive platform testing in a practical environment. However, under limited testing, the OSR was able to complete its objectives. It successfully and reliably generated a point cloud and occupancy grid of its surrounding area; and it navigated around obstacles to arbitrary points.

A surprising strength of Nav2 was its robustness against severe hardware problems. During testing, the rover's steering servos were not zeroed correctly, so that the wheels could not accurately determine their orientation. The Nav2 behavior tree was able to compensate for this, with the robot eventually reaching its goal position. However, orientation matching lacked reliability, and Nav2 would abort the desired goal when detecting excessively incorrect movements.

As addressed in the future work section, the design implementation is feasible. However, the rover requires further development and more extensive testing to meet its potential as a platform. Detailed instructions for booting the platform are in Appendix F.

7.2 Performance Analysis

During nominal autonomous operation, the Raspberry Pi and NVIDIA Jetson Nano were found to use approximately `52.4`% and `67.3`% of their respective CPU capabilities, as well as `61.6`% and `83.8`% of their respective memory capabilities. This was averaged over a 15 minute period of operation, with performance metrics collected via the Linux `top` utility.

Given these metrics, it is plausible that a few additional packages could be run on the Raspberry Pi. However, this estimate may vary in the future with advanced Nav2 techniques. The NVIDIA Jetson Nano lacks the necessary performance overhead to support further packages.

Performance on the NVIDIA Jetson Nano could possibly be improved by utilizing its GPU resources, which were not used in the current implementation.

7.3 Cost Analysis

The total cost of the system remained within the target budget of approximately $1,000 USD for additional components beyond the base OSR platform. A detailed material bill and cost breakdown are shown in Appendix G.

8 Conclusions & Future Work

In the process of this experiment, the JPL OSR was found to be a promising cost-effective research platform for autonomous ground vehicles. The following describes future work for this project, including both the changes to the code described in this document and general modules that do not directly require that code.

1. **Extensive Physical Testing and Validation**: Numerical simulation can serve only to verify an implementation. Extensive testing in a physical environment based on a sound design of experiments would be required to ensure the platform's robustness.

2. **RTAB-Map GPU utilization**: Computing resources can be used more efficiently if RTAB-Map is configured to use the GPU present on the Jetson Nano.

3. **Nav2 Expansion**: Implementing Nav2 advanced capabilities, including a route follower for multiple waypoints, would further strengthen the functionality of the platform.

4. **Sensor Suite Expansion**: The minimal sensor configuration used in the present experiment provided basic functionality. Experimenting with a variety of sensor suites on this platform would be necessary to ensure an optimal sensor suite and configuration for specific environments of interest.

5. **Time Synchronization**: The current implementation of the platform does not automatically synchronize time across its computing resources.

6. **Differential Linkage Simulation**: Due to limitations of the URDF, the differential linkage present in the OSR cannot be accurately modeled. Another description format, such as the Simulation Description Format (SDF) native to Gazebo, could be used to address this problem.

7. **Multi-Agent Operation**: Due to its cost effectiveness, the OSR is well suited for multi-agent operation, where multiple platforms can collaborate to achieve tasks. Multi-agent collaboration would be the next overall development target.

References

[1] Jet Propulsion Laboratory. "JPL Open Source Rover Project." (2023), [Online]. Available: https://github.com/nasa-jpl/open-source-rover.

[2] Jet Propulsion Laboratory. "JPL Open Source Rover Code." (2025), [Online]. Available: https://github.com/nasa-jpl/osr-rover-code.

[3] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, eabm6074, 2022. DOI: 10.1126/scirobotics.abm6074. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[4] Open Robotics. "Gazebo Harmonic Documentation." (2023), [Online]. Available: https://gazebosim.org/docs/harmonic/getstarted/.

[5] Intel RealSense. "Intel® RealSense™ Depth Camera D435i Specifications." (2018), [Online]. Available: https://www.intel.com/content/www/us/en/products/sku/190004/intel-realsense-depth-camera-d435i/specifications.html.

[6] Intel RealSense. "Intel® RealSense™ Depth Camera D405 Specifications." (2022), [Online]. Available: https://www.intel.com/content/www/us/en/products/sku/229218/intel-realsense-depth-camera-d405/specifications.html.

[7] S. Macenski and I. Jambrecic, "SLAM toolbox: SLAM for the dynamic world," *The Journal of Open Source Software*, 2021. DOI: http://dx.doi.org/10.21105/joss.02783.

[8] M. Labbé and F. Michaud, "RTAB-Map as an open-source Lidar and visual SLAM library for large-scale and long-term online operation," *Journal of Field Robotics*, vol. 36, no. 2, pp. 891–921, 2019. DOI: https://doi.org/10.48550/arXiv.2403.06341.

[9] S. Macenski, F. Martín, R. White, and J. Ginés Claver, "The Marathon 2: A Navigation System," *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020. DOI: https://doi.org/10.48550/arXiv.2003.00368.

[10] Docker Inc. "Docker Documentation." (2025), [Online]. Available: https://docs.docker.com/.

[11] Ivan Dryanovski. "Madgwick Filter." (2022), [Online]. Available: https://wiki.ros.org/imu_filter_madgwick.

Appendix A

OSR Low-Level Motor Abstraction

This appendix details how high-level linear and angular velocity commands are mathematically interpreted into low-level motor outputs.

A.1 Terminology

Steer axis A vertical axis projected through the center of a wheel.

Rotation axis A horizontal axis projected through the center of a wheel.

Corner wheels The four wheels with the ability to rotate about their steer axes.

Middle wheels The two wheels without the ability to rotate about their steer axes.

A.2 Variable Definitions

The use of an asterisk (*) with a variable denotes desired or commanded. Assume actual/measured quantities in all other cases. *Letters for ϕ_{XX} and ω_{XX} denote Front/Middle/Back and Left/Right.*

Wheel steer (Φ_{XX}): Current angular position of a corner wheel about its steer axis.

Wheel angular velocity (ω_{XX}): Current angular velocity of a wheel about its rotation axis.

Center angular velocity (ω_C): Angular velocity of the chassis about its center (or projected center of rotation).

Linear velocity (V_C): Linear velocity of the chassis.

Turning radius (ρ): The turning radius drawn horizontally from the center of the robot.

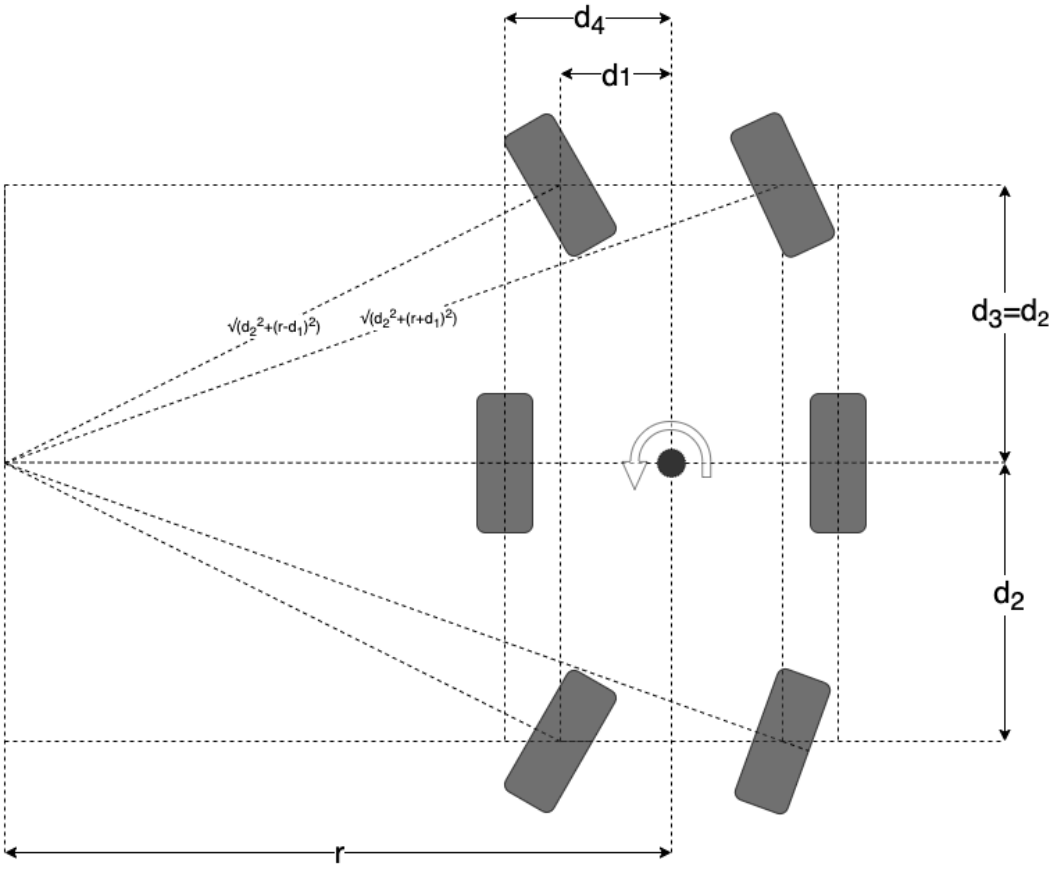


Figure A1: OSR Dimension Illustration (Credit to JPL)

A.3 Robot Dimensions

d_1: Horizontal distance from the rover centerline to the corner wheel steer axes
d_2: Vertical distance from the back wheel centerline to the middle wheel centerline
d_3: Vertical distance of the front wheel centerline from the middle wheel centerline
d_4: Horizontal distance from the rover centerline to the middle wheel steer axes
r_w: Wheel radius
ρ_{max} : Maximum achievable turning radius
ρ_{min} : Minimum achievable turning radius
V_{max} : Maximum achievable velocity taking turn radius into account (* Refers to maximum straight-line velocity)

A.4 Motor Calculations

This flowchart models the OSR steering control logic at a high level. Calculations for each step can be found in the associated subsections.

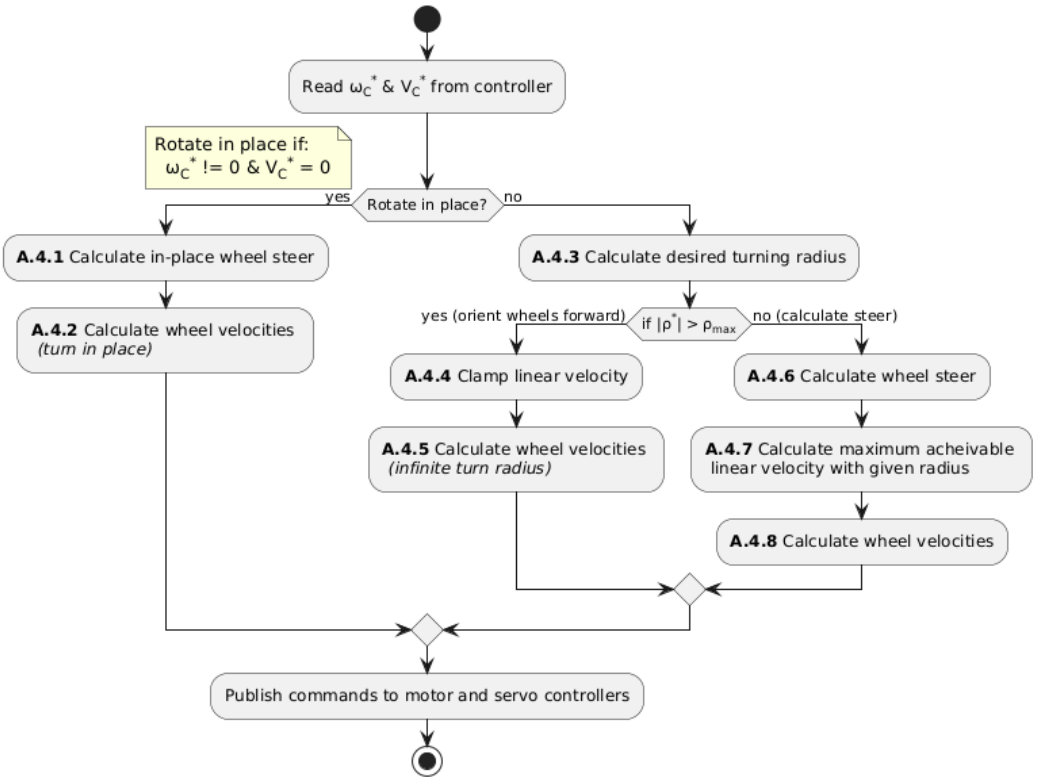


Figure A2: OSR Motor Control Scheme

A.4.1 Calculate in-place wheel steer

$$\phi^*_{FL} = \arctan(\frac{d_3}{d_1})$$
$$\phi^*_{BL} = -\arctan(\frac{d_3}{d_1})$$
$$\phi^*_{BR} = \arctan(\frac{d_2}{d_1})$$
$$\phi^*_{FR} = -\arctan(\frac{d_2}{d_1})$$

A.4.2 Calculate wheel velocities (turn in place)

$$\omega_{FL}^* = \omega_{FR}^* = \sqrt{{d_1}^2 + {d_3}^2} \times \frac{\omega_C^*}{r_w}$$
$$\omega_{ML}^* = \omega_{MR}^* = d_4 \times \frac{\omega_C^*}{r_w}$$
$$\omega_{BL}^* = \omega_{BR}^* = \sqrt{{d_1}^2 + {d_2}^2} \times \frac{\omega_C^*}{r_w}$$

A.4.3 Calculate desired turning radius

$$\rho^* = \frac{V_C^*}{\omega_C^*};\ \rho_{min} \leq |\rho^*| \leq \rho_{max} \qquad \text{if } \omega_C^* \neq 0$$
$$\rho^* = \infty \qquad \text{if } \omega_C^* = 0$$

A.4.4 Clamp linear velocity

$$V_{max} = {V_{max}}^*$$
$$|V_C^*| \leq V_{max}$$

A.4.5 Calculate wheel velocities (infinite turn radius)

$$\omega_{FL}^* = \omega_{ML}^* = \omega_{BL}^* = \frac{V_C^*}{r_w}$$
$$\omega_{FR}^* = \omega_{MR}^* = \omega_{BR}^* = -\frac{V_C^*}{r_w}$$

A.4.6 Calculate wheel steer

$$\begin{cases} \phi_{FL}^* = -\arctan_2(d_3, |p^*| - d_1) \\ \phi_{BL}^* = \arctan_2(d_3, |p^*| - d_1) \\ \phi_{BR}^* = \arctan_2(d_2, |p^*| + d_1) \\ \phi_{FR}^* = -\arctan_2(d_2, |p^*| + d_1) \end{cases} \quad \text{if } \rho^* > 0$$

$$\begin{cases} \phi_{FL}^* = \arctan_2(d_2, |p^*| + d_1) \\ \phi_{BL}^* = -\arctan_2(d_2, |p^*| + d_1) \\ \phi_{BR}^* = -\arctan_2(d_3, |p^*| - d_1) \\ \phi_{FR}^* = \arctan_2(d_3, |p^*| - d_1) \end{cases} \quad \text{if } \rho^* \leq 0$$

A.4.7 Calculate maximum achievable linear velocity with given radius

$$V_{max} = \frac{|\rho^*|}{|\rho^*| + d_1} \times {V_{max}}^*$$
$$|V_C^*| \leq V_{max}$$

A.4.8 Calculate wheel velocities

Adjust ω_C^* for constraints:

$$\omega_C^* = \frac{V_C^*}{|\rho^*|}$$

Calculate placeholder velocities in terms of middle/corner, close/far side:

$$\begin{aligned}
\text{Let MC} &= (|\rho^*| - d_4) \times \frac{\omega_C^*}{r_w} \\
\text{Let CC} &= \sqrt{(|\rho^*| - d_1)^2 + {d_3}^2} \times \frac{\omega_C^*}{r_w} \\
\text{Let CF} &= \sqrt{(|\rho^*| + d_1)^2 + {d_3}^2} \times \frac{\omega_C^*}{r_w} \\
\text{Let MF} &= (|\rho^*| + d_4) \times \frac{\omega_C^*}{r_w}
\end{aligned}$$

$$\begin{cases}
\omega_{FL}^* = \omega_{BL}^* = \text{CC} \\
\omega_{FR}^* = \omega_{BR}^* = -\text{CF} \\
\omega_{ML}^* = \text{MC} \\
\omega_{MR}^* = -\text{MF}
\end{cases} \quad \text{if } \rho^* > 0$$

$$\begin{cases}
\omega_{FL}^* = \omega_{BL}^* = \text{CF} \\
\omega_{FR}^* = \omega_{BR}^* = -\text{CC} \\
\omega_{ML}^* = \text{MF} \\
\omega_{MR}^* = -\text{MC}
\end{cases} \quad \text{if } \rho^* \leq 0$$

Appendix B

Sensor Simulation in Gazebo

B.1 RealSense Description Package

RealSense has published a description package for use in sensor simulation, under the name `realsense2_description`. By including the `/urdf/_d435i.urdf.xacro` file under the package shared directory in a Gazebo URDF file, the `xacro:sensor_d435i` tag can be used to construct a camera model. Supply the `parent` transform link as a parameter and an `origin` sub-tag containing `xyz` and `rpy` parameters to describe its relative three-dimensional pose.

To construct the necessary optical link with the ROS2 standard image coordinate system, create a joint in the URDF file linking `camera_link` generated by the description to a new link used for the optical frame. Provide a rotation of $\{-\frac{\pi}{2}, 0, -\frac{\pi}{2}\}$ using the `rpy` parameter in the new joint's `origin` sub-tag.

B.2 RGB Camera



Figure B1: RGB Camera Simulation

reference	`camera_link`
type	`camera`
gz_frame_id	`camera_link_optical`
topic	`../color/image_raw`
image format	`R8G8B8`
near clip	`0.05m`
far clip	`100.0m`

Table B1: RGB camera parameters

B.3 Depth Camera



Figure B2: Depth Camera Simulation

reference	camera_link
type	depth
gz_frame_id	camera_link_optical
topic	../aligned_depth_to_color/image_raw
image format	R_FLOAT32
near clip	0.28m
far clip	10.0m

Table B2: Depth camera parameters

Appendix C

RTAB-Map Configuration

RGBD sync is used to unify RGB and depth frames before they are sent to RTAB-Map. This is because the camera does not output frames at an exact synchronized rate. Note that while the camera does publish a synchronized RGBD topic, this topic is not currently of the type expected by RTAB-Map, causing errors.

package	rtabmap_launch
file	launch/rtabmap.launch.py
args	--delete_db_on_start
depth_topic	../aligned_depth_to_color/image_raw
rgb_topic	../color/image_raw
camera_info_topic	../color/camera_info
frame_id	base_footprint
publish_tf_odom	true
odom_topic	/odom
odom_frame_id	odom
approx_sync	true
rgbd_sync	true
approx_rgbd_sync	true
subscribe_rgbd	true
visual_odometry	true
qos	1
imu_topic	/rtabmap/imu
wait_imu_to_init	true
rtabmap_viz	true
map_topic	/map

Table C1: RTAB-Map Launch Arguments

Appendix D

Nav2 Configuration

To use Nav2, install both the `navigation2` and `nav2_bringup` packages.

The `nav2_params.yaml` that can be found in the shared directory for `nav2_bringup` under the `params` directory was found to function with minimal modification. This file was copied into the OSR navigation package's `config` directory. It was verified that Nav2 was subscribed to the correct ROS2 topics, so the RTAB-Map generated map was correctly provided to the navigation stack.

A modification to the default Nav2 configuration was to add a `VelocityDeadbandCritic` to the Nav2 `controller_server` critic list, as the OSR drive motors were initially unable to overcome static friction with the output provided by the navigation controller. This critic mainly used the default values recommended in the Nav2 documentation. Deadband velocities were arbitrarily selected.

enabled	`true`
cost_weight	`35.0`
cost_power	`1`
deadband_velocities	`[0.25, 0, 0.25]`

Table D1: VelocityDeadbandCritic parameters

Furthermore, the maximum velocity and acceleration parameters used by the `FollowPath` plugin were tuned to the environment the rover was operating in. These will vary depending on the intended rover behavior.

The following arguments were used to launch nav2 with this configuration.

package	`nav2_bringup`
file	`launch/navigation_launch.py`
params_file	`config/nav2_params.yaml`

Table D2: Nav2 Launch Arguments

Appendix E

Additional Hardware Configuration

E.1 RealSense D435i

This table contains the arguments supplied to the D435i launch file.

`align_depth.enable` is used to align the RGD and depth feeds such that RTAB-Map can combine and utilize them as RGBD data.

`unite_imu_method 2` specifies that linear interpolation is to used to synchronize IMU data.

package	`realsense2_camera`
file	`launch/rs_launch.py`
enable_color	`true`
enable_depth	`true`
align_depth.enable	`true`
pointcloud.enable	`false`
enable.sync	`false`
unite_imu_method	`2`
enable_gyro	`true`
enable_accel	`true`
publish_tf	`true`

Table E1: D435i Launch Arguments

E.2 Madgwick Filter

A Madgwick [11] IMU filter was applied on raw IMU data to reduce noise. The specific filter was selected due to its ease of use and community success using it with the D435i camera.

package	`imu_filter_madgwick`
executable	`imu_filter_madgwick_node`
use_mag	`False`
publish_tf	`True`
fixed_frame	`camera_link`
Remap: `/imu/data_raw`	`/camera/camera/imu`
Remap: `/imu/data`	`/rtabmap/imu`

Table E2: Madgwick ROS Node information

E.3 Twist Mux

The `twist_mux` package is used to switch between remote and autonomous control. Remote commands are given higher priority, so that they are capable of interrupting autonomous commands. The configuration is contained in a `twist_mux_params` file under the `/config` directory of the `osr_bringup` package.

package	`twist_mux`
executable	`twist_mux`
params-file	`config/twist_mux_params.yaml`

Table E3: Twist Mux Launch Arguments

name	`navigation`
topic	`cmd_vel_nav`
timeout	`0.5`
priority	`10`

Table E4: Twist Mux `navigation` Parameters

name	`navigation`
topic	`cmd_vel_joy`
timeout	`0.5`
priority	`100`

Table E5: Twist Mux `joystick` Parameters

Appendix F

Enumerated bringup instructions

F.1 Simulation bringup

These instructions assume that the code and images have already been built. The instructions presume access to the referenced code, which is currently available to internal NASA users.

1. Change directory to repository directory

   ```
   cd osr-ws
   ```

2. Start using docker compose

   ```
   docker compose up -d sim
   ```

3. Attach to container

   ```
   docker attach sim
   ```

4. Run simulation launch file

   ```
   ros2 launch osr_gazebo empty_world.launch.py
   ```

5. Open a new terminal (This will need to be done every time a new ROS2 command is run to view nodes simultaneously)

   ```
   docker exec -it sim bash
   ```

6. (Optional) Run visualization launch file

   ```
   ros2 launch osr_gazebo rviz.launch.py
   ```

7. Run RTAB-Map launch file

   ```
   ros2 launch osr_slam osr_sim_rtabmap.launch.py
   ```

8. Run Nav2 launch file

   ```
   ros2 launch osr_nav osr_nav.launch.py
   ```

9. Run joystick

   ```
   ros2 launch teleop_twist_joy teleop-launch.py
   ```

F.2 Hardware bringup

These instructions assume that the code and images have already been built and deployed. The instructions presume access to the referenced code, which is currently available to internal NASA users.

1. SSH into Raspberry Pi

   ```
   ssh user@$RPI_IP
   ```

2. Change directory to launch directory

   ```
   cd osr-init
   ```

3. Run launch file

   ```
   source launch_osr.bash
   ```

4. SSH into Jetson Nano

   ```
   ssh user@$JETSON_IP
   ```

5. Change directory to launch directory

   ```
   cd osr-init
   ```

6. Run launch file

   ```
   source launch_osr.bash
   ```

Appendix G

Bill of Materials

Sensors

Activity	Rate/Unit	Count	Amount (USD)
Intel RealSense D435i	339.27	1	339.27
Subtotal Sensors			339.27

Computing Platform

Activity	Rate/Unit	Count	Amount (USD)
NVIDIA Jetson Orin Nano Super Developer Kit	249.00	1	249.00
TEG-S5g Network Switch	14.99	1	14.99
CRIUS Power Distribution	14.70	1	14.70
Subtotal Fees			278.69
Subtotal Computing Platform			278.69
Sum Fees			617.96
Total			617.96

All prices shown are the approximate quotes from the vendor at the time of writing. The NVIDIA Jetson Nano used has been discontinued by the manufacturer; the above table reflects a more capable model at a similar price.