Technical Challenge Report: Autonomous Robot Logging with ROS 2 and Blockchain

1. Introduction

This report documents the design, implementation, and evaluation of the technical challenge outlined in `TechnicalChallenge.txt`. The goal was to simulate a mobile robot in ROS 2 and Gazebo, enable autonomous navigation on a pre-defined map, record key operational data, visualize the results, and log selected events on a blockchain.

The implementation demonstrates an integrated system that achieves these objectives and provides a foundation for further improvements.

---

2. Experiment Goal and Scenario

The chosen experimental goal was **autonomous waypoint navigation**. Specifically, the robot was tasked with visiting **three fixed waypoints** defined in a CSV file:

```
(-2.0, -1.0)
( 4.0, -1.8)
( 3.0,  1.7)
```

All waypoints are defined in the `map` frame. The expected behavior is that the robot sequentially navigates to each waypoint, reporting progress and logging operational data.

---

3. ROS 2 System Architecture

The simulation was implemented in **ROS 2 Humble** with the following components:

Functionality	Implementation
Simulation	Gazebo world: `warehouse.world`
Map	`warehouse.yaml` with PGM occupancy grid
Robot Model	URDF: `sdnt_robot_description.urdf`
Launch System	`simulation.launch.py` orchestrating Gazebo, robot state publishers, EKF, map server, AMCL, Nav2, TFs, and RViz2
Navigation	Nav2 stack with configuration in `nav2_params.yaml`
Localization	Extended Kalman Filter (EKF) with AMCL
Waypoint Control	`odom_follower.py` (sequential goals), with alternatives (`go_through_poses.py`, `gps_follower.py`)

Functionality	Implementation
Status Topic	`/nav2_status` published by `odom_follower.py` (`waiting`, `get target`, `moving`, `reached target`)

This architecture enables reproducible autonomous navigation while maintaining modularity for experimentation.

---

4. Data Collection and Logging

Operational data was recorded as follows:

- **Odometry (`/odom`)**: Robot position (x, y), logged at 1 Hz if movement exceeded 0.1 m.
- **Status (`/nav2_status`)**: Discrete string values reflecting navigation progress.
- **Timestamp**: Recorded using wall clock time (`time.time()`).

A logging script (`blockchain_logger.py`) structured data as JSON:

```
{"timestamp": <float>, "robot": {"time": <float>, "x": <x>, "y": <y>,
"status": "<string>"}}
```

Data was sent to the blockchain in real time. In case of blockchain unavailability, records were written locally to `movement_log.json`.

---

5. Blockchain Integration

A local Ethereum blockchain was simulated using **Ganache CLI**. Parameters included deterministic accounts, block time, and a persistent database.

Transactions were generated via `web3.py` with the following characteristics:

- **Account**: The first Ganache account used as both sender and receiver.
- **Payload**: JSON-encoded robot state, stored as a UTF-8 hex string in the transaction data field.
- **Trigger**: Status changes or significant movement events.
- **Extractor**: `extract_blockchain_data.py` retrieves, decodes, and compiles all stored records into `decoded_blockchain_logs.json`.

This setup ensures transparent and immutable storage of robot events.

---

6. Experiment Methodology

The experiment followed a reproducible procedure scripted in `run.sh`:

1. Clean previous build and logs.
2. Build workspace with `colcon`.
3. Launch Ganache blockchain.

4. Start simulation (Gazebo + Nav2 + RViz2).
5. Run blockchain logger.
6. Start waypoint follower node.
7. After navigation, extract and decode blockchain transactions.

Expected behavior: the robot sequentially visits all three waypoints, generating blockchain transactions whenever status changes or significant movements occur.

---

7. Visualization

Two forms of visualization were provided:

- **Interactive**: RViz2 displays the map, robot pose, and navigation layers during execution.
- **Static**: `visualization.py` plots the decoded blockchain logs over the map background, including:
  - Waypoints (numbered)
  - Start (green) and end (red) positions
  - Actual path polyline

The resulting figure (`path_plot.png`) confirms the robot visited the defined waypoints in the correct order.

---

8. Reproducibility

Reproducibility was ensured by:

- Deterministic blockchain configuration (`--deterministic` flag in Ganache).
- Fixed waypoint definitions in CSV.
- Orchestrated setup via `setup.sh` and single-command execution (`./run.sh`).

This guarantees consistent experimental outcomes across runs.

---

9. Evaluation Against Requirements

Challenge Requirement	Implementation Evidence
Simulate robot in ROS 2 + Gazebo	`simulation.launch.py`, warehouse world
Autonomous navigation on map	Nav2 stack with waypoints
Record timestamps, position, status	`blockchain_logger.py`
Define single clear goal	Three fixed waypoints
Describe methodology & expected behavior	Documented in this report and `run.sh`
Visualization	RViz2 + static plot
Blockchain setup	Ganache CLI integration
web3.py transactions	Implemented in logger

Challenge Requirement	Implementation Evidence
Transaction payload includes required fields	Timestamp, position, status
Store data on blockchain	JSON payload in transaction input

10. Limitations

- Transactions are raw self-transfers (no smart contract for structured querying).
- Status messages are minimal and contain a typo (`waitin`).
- No explicit waypoint indices included in the payload.
- Orientation and velocity are not recorded.
- Gas estimation is heuristic and not fully aligned with Ethereum cost models.

11. Future Improvements

- Replace self-transactions with a **dedicated smart contract**.
- Expand logged data (orientation, velocity, waypoint index).
- Improve status messages for clarity (e.g., “Reached waypoint 2”).
- Add recovery behaviors and dynamic waypoint handling in navigation.
- Provide headless, CI-friendly launch configuration without GUI dependencies.

12. Conclusion

The implemented system successfully meets the objectives of the technical challenge:

- A ROS 2 mobile robot autonomously navigates a pre-defined map.
- Timestamps, positions, and statuses are logged and recorded on a local blockchain.
- Results are visualized interactively and through static plots.
- The methodology is documented and reproducible.

While several improvements are possible, the delivered implementation fulfills all required evaluation criteria and provides a strong basis for further research and development.