Autonomous Drone Georeferencing via Factor Graphs

CS6365/RTES Final Project Report

Vladyslav Havriutkin

November 30, 2025

Abstract

This report details the design and implementation of an autonomous drone algorithm capable of georeferencing points of interest (POI) using a factor graph optimization approach. Utilizing ROS2, MAVROS, ArduPilot SITL, and the GTSAM library, the system performs a flight mission, captures Lidar and GPS data from multiple vantage points, and solves an optimization problem to estimate target coordinates with high precision.

Contents

1 Introduction

This work is part of the larger project in RoboJackets club of Georgia Tech for the University Rover Challenge (URC). The URC [Mar25] is the robotics competition for university students, tasking teams with designing autonomous rovers for harsh environments. A critical component of the competition involves the "Autonomous Navigation" and "Delivery" missions, where rovers must traverse unstructured terrain with obstacles to pick up and deliver specific objects (e.g., toolbox, rocks, etc.). Ground-based rovers encounter major difficulties in such missions because the environment is unknown and the terrain creates line-of-sight blockages. To address these issues, the concept of deploying Unmanned Aerial Vehicles (UAVs) has emerged, as they offer the possibility to locate Points of Interest (POIs) from the air before rover even starts the mission.

This project focuses on a specific competency required for such drone operations: **autonomous georeferencing**. While identifying an object in a camera frame can be done manually or using computer vision algorithms, translating that detection into precise global coordinates (Latitude/Longitude) remains a complex challenge. In low-cost drones, onboard sensors—specifically GPS and Magnetometers—are prone to drift and noise. A naive approach, such as simple geometric ray-casting from the drone to the ground, is highly sensitive to these errors; a pitch error of merely 2° at an altitude of 30 meters can result in a ground position error of over 1 meter.

To address the stochastic nature of sensor noise, this project implements a probabilistic estimation approach using **factor graphs** (discussed in section 4). Instead of relying on a single instantaneous measurement, the system utilizes a "Tri-View" data collection strategy. The drone autonomously navigates to three distinct vantage points around the target, collecting Lidar range data and gimbal orientation at each step. These measurements are fused using the **GTSAM** (Georgia Tech Smoothing and Mapping) library [DC22], which solves a non-linear optimization problem to determine the most probable location of the target. This approach shifts the paradigm from deterministic calculation to probabilistic inference, significantly increasing robustness against sensor noise.

The main contribution of this work is a modular software stack built on ROS2 that allows a drone to autonomously circle a target and estimate its position with high accuracy. In addition, a realistic Gazebo simulation environment is configured and employed for testing. This report details the system architecture, the factor graph formulation, and the simulation results, demonstrating a reusable capability that can be integrated into the RoboJackets software ecosystem for future URC competitions.

2 Motivation

2.1 Sensitivity of Geolocation to Sensor Noise

The primary motivation for employing a probabilistic factor graph approach rather than deterministic triangulation lies in the non-linear propagation of sensor errors. In a standard "Ray-Casting" approach, the drone estimates the ground target's position by projecting a vector from its current GPS position based on the gimbal's orientation and the Lidar's range measurement.

Consider a simplified 2D scenario where a drone hovers at altitude h and detects a target at a pitch angle θ. The horizontal distance to the target, d, is given by the trigonometric relation:

$$d = h \cdot \tan(\theta) \tag{1}$$

In real-world scenarios Lidar reading θ contains measurement noise $\delta\theta$. To quantify how this angular noise impacts the final position estimate, we can look at the sensitivity of d with respect to θ by taking the partial derivative:

$$\frac{\partial d}{\partial \theta} = h \cdot \sec^2(\theta) \tag{2}$$

The error scales with the *square* of the secant of the look angle. As the drone looks further away (increasing θ), the error multiplier grows exponentially.

For example, consider a drone at a typical survey altitude of $h = 30$ meters looking at a target at $\theta = 45^\circ$. If the gimbal stabilization has an error of $\delta\theta = 2^\circ$ (approx. 0.035 radians), the resulting positional error δd is:

$$\delta d \approx \left(30 \cdot \sec^2(45^\circ)\right) \cdot 0.035 = 30 \cdot 2 \cdot 0.035 \approx 2.1 \text{ meters} \tag{3}$$

A 2-meter error is significant for a rover attempting to navigate to a specific tool or rock; it could mean the difference between the rover finding the object or driving past it. Since our single-ray Lidar provides only a 1D depth measurement, it cannot inherently correct this angular error. This motivates adopting a probabilistic framework that optimizes jointly over multiple measurements taken from different viewpoints $(\theta_1, \theta_2, \theta_3)$, thereby effectively averaging out the Gaussian sensor noise. I use GTSAM and factor graphs as the in-house state-of-the-art solution for SLAM inference tasks.

2.2 Gimbal Control Abstraction

In this project, we utilize a custom 3-axis gimbal configuration that does not map 1-to-1 with off-the-shelf ArduPilot supported hardware.

However, we aim to maintain compliance with the **MAVROS** standard [**?**], which serves as the layer between ROS2 high-level control and the flight controller. We require the on-board computer to publish orientation targets without knowing whether the underlying hardware is a simulated joint in Gazebo or a physical servo on the drone. Directly controlling the joints from the mission manager would break the MAVROS abstraction.

To resolve this, I implement a gimbal bridge. ArduPilot is configured to treat the mount as a generic servo-driven gimbal. However, instead of wiring this to physical outputs, we intercept the processed orientation quaternion emitted by MAVROS. The bridge then acts as a translator:

- **In Simulation:** It translates the quaternion into Gazebo `/cmd_pitch` topics to drive the physics engine.
- **In Deployment:** It translates the same quaternion into PWM signal values for a driver board.

This ensures that my R&D efforts in simulation are directly portable to the physical support without rewriting the control logic.

3 System Architecture

3.1 Drone Characteristics

The simulation utilizes the `Iris` quadcopter model, a standard reference platform in the ArduPilot ecosystem[cite]. While the physical URC drone features a custom architecture, the software model abstracts these mechanical details, relying on the generic multi-rotor dynamics provided by the simulator. The drone is equipped with a custom 3-axis gimbal mechanism. This gimbal is actively actuated, allowing the camera and Lidar to track ground targets.

The following sensors are mounted on the drone:

1. GPS. Provides global latitude/longitude (simulated with realistic noise, $\pm 2m$ accuracy).

2. Lidar Rangefinder. A single-ray laser rangefinder co-aligned with the camera axis. It provides a scalar depth measurement d to the ground target. The standard deviation of 0.2 is used for noise simulation.

3. IMU. Provides high-frequency attitude estimation.

3.2 Software Stack

The system operates on Ubuntu 24.04 (Noble Numbat) and is built upon the ROS2 Jazzy [MFG+22].

The core flight dynamics, navigation filters (EKF), and motor mixing are handled by ArduPilot [Ard25]. This provides a "black box" flight controller that runs the exact same firmware binaries as the physical Pixhawk hardware, ensuring 1:1 portability. Additionally, ArduPilot provides out-of-the-box solutions for standard mission planning and integrations with GPS modules.

We use state-of-the-art MAVROS [Mav25] as the primary translation bridge between flight computer (ROS2) and flight controller (ArduPilot). It subscribes to ROS2 topics (e.g., `/mavros/setpoint_position/local`) and serializes them into MAVLink packets over UDP. It is utilized for arming the vehicle, switching flight modes, and reporting telemetry.

The Gazebo physics engine [KH04] renders the world and simulation. It handles the rigid body dynamics and sensor emulation. Crucially, it interfaces with ArduPilot via the `ardupilot_gazebo` plugin to synchronize motor speeds and sensor data. We choose Gazebo due to good reputation and excessive documentation.

3.3 Mission Architecture

The autonomous capabilities are encapsulated within the `drone_control` package, primarily distributed between the `mission_manager` and `auto_collector` nodes. These components orchestrate the mission state machine, managing the transition between manual operator oversight and autonomous execution.

3.3.1 Mission State Machine

The georeferencing mission is represented as a state machine, detailed in Figure 1.

1. **Phase 1: Transit (Autonomous Navigation)**
   The cycle begins with the transition from `IDLE` to `TRANSIT`. The drone receives global GPS coordinates for the Area of Operations (AO) and utilizes ArduPilot's `GUIDED`

mode to navigate to the site. MAVROS handles the conversion of global coordinates into local NED (North-East-Down) setpoints. Upon arrival, the system automatically transitions back to `IDLE` to await operator input.

2. **Phase 2: Acquisition (Human-in-the-Loop)**
   While in the `IDLE` state, the operator manually controls the gimbal orientation to center the Lidar on the Point of Interest (POI). We assume that the rangefinder is oriented correctly in this phase. Once the target is visually acquired, the operator triggers the `/start_collection` service, initiating the transition into the **Active Collection State**. This phase has a potential to be fully autonomous by utilizing computer vision alorithms. We discuss that possibility in section 8.

3. **Phase 3: Scanning (Composite Active State)**
   As illustrated by the dashed region in Figure 1, this phase is a composite state containing the "Stop-and-Stare" control loop.

   - **Initialization:** The system first computes the relative vectors for the three required vantage points based on the initial aim.
   - **Execution Loop:** The state machine cycles through `MOVING`, `STABILIZING`, and `SNAPSHOT` states. A timer guard (`[timer > 5s]`) in the stabilization state ensures the IMU settles before data recording.
   - **Error Handling:** To ensure mission robustness, a specific timeout transition (`[Timeout] / Skip WP`) allows the system to bypass a waypoint if navigation logic hangs or encounters obstacles, proceeding directly to the index check to preserve mission continuity.

4. **Phase 4: Return and Inference**
   Upon completing the loop (Index ≥ 3), the system exits the composite state and transitions to `RETURN_START`, navigating back to the initial hover position. Finally, the system enters the `INFERENCE` state, where the collected JSON log is transmitted to the factor graph optimization node before returning to `IDLE` for the next target.

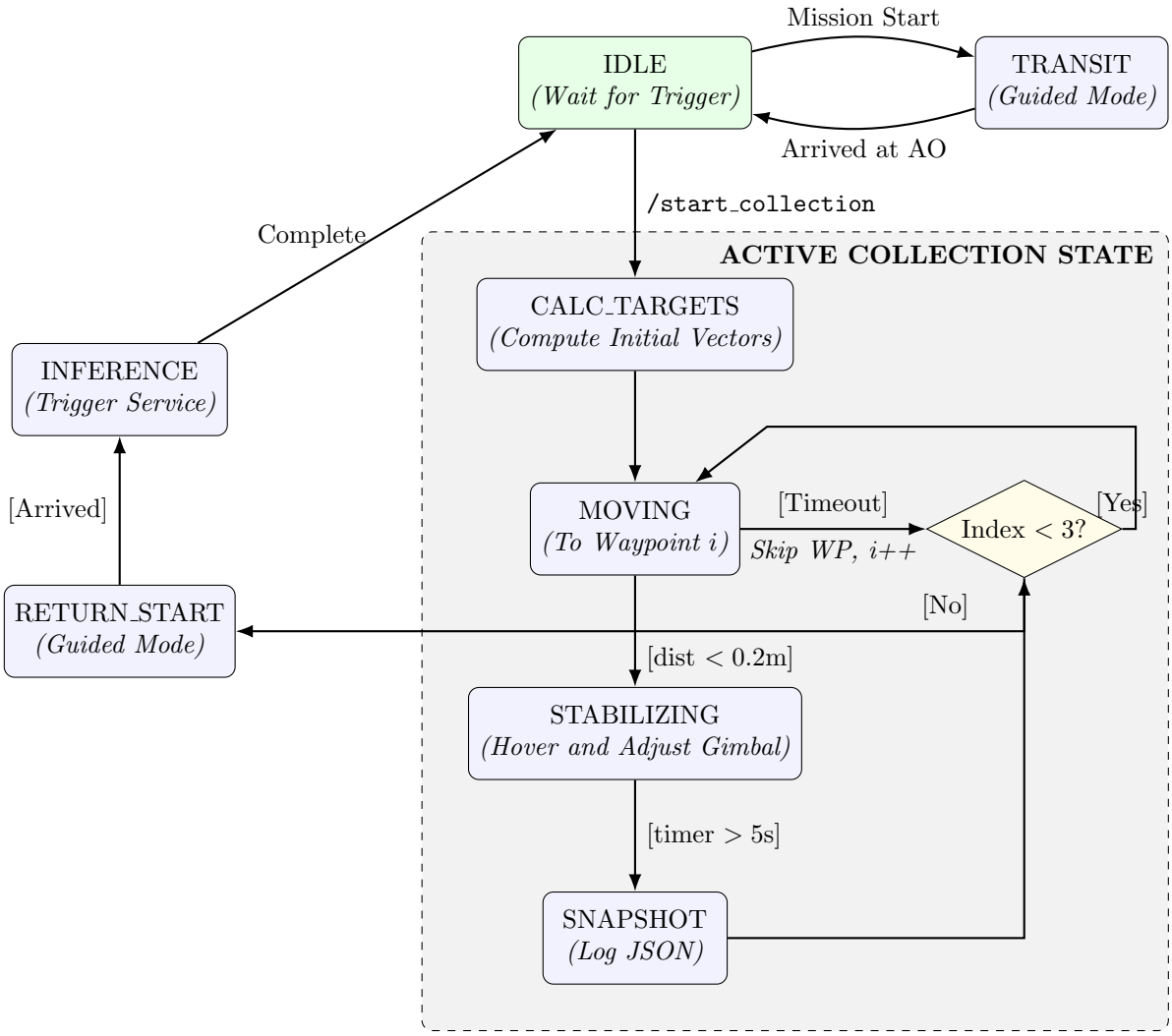

Figure 1: State Machine Diagram of the Mission.

3.3.2 Gimbal Control Logic

Gimbal Adjustments. To maintain gimbal lock on the target during the Phase 3 maneuvers, we implement a relative pointing update scheme. Rather than recalculating the global target position at every time step—which would introduce recursive sensor noise—we rely on drone odometry to compute displacement relative to the initial pose.

Let the state of the drone at the start of the mission (t_0) be defined by position $\mathbf{p}_0 \in R^3$, orientation $\mathbf{R}_0 \in SO(3)$, and the initial looking vector $\mathbf{v}_0$. The target position $\mathbf{p}_{target}$ is implicitly defined as:

$$\mathbf{p}_{target} = \mathbf{p}_0 + \mathbf{R}_0\mathbf{v}_0 \tag{4}$$

As the drone traverses to a new waypoint at time t_k, we calculate the translational displacement $\Delta\mathbf{p} = \mathbf{p}_k - \mathbf{p}_0$. The new required looking vector $\mathbf{v}'_{world}$ is computed by subtracting this displacement from the initial target vector:

$$\mathbf{v}'_{world} = (\mathbf{R}_0\mathbf{v}_0) - \Delta\mathbf{p} \tag{5}$$

Finally, this vector is transformed into the drone's current body frame to extract control angles (θ_k, ψ_k):

$$\mathbf{v}'_{body} = \mathbf{R}_k^T\mathbf{v}'_{world} \tag{6}$$

The Gimbal Bridge. Since ArduPilot's native driver stack does not support our custom gimbal mechanism, we established a hardware abstraction layer to maintain standard MAVLink compatibility. The bridge node acts as a middleware, subscribing to high-level MAVROS commands and translating them into specific actuation signals. In the current simulation environment, these are mapped to Gazebo joint controllers. Crucially, this architecture decouples the flight control logic from the hardware; to deploy on the physical drone, only this low-level translation step needs to be updated to drive physical pins, while the high-level path planning remains unchanged.

4 SLAM via Factor Graphs

To refine the target's estimated position, we formulate the georeferencing task as a probabilistic inference problem. Simultaneous Localization and Mapping (SLAM) is traditionally used to construct a map of an unknown environment while tracking an agent's location. In our specific application, the "map" consists of the static target (landmark), and the "agent" is the drone capturing range measurements. We model this system using a factor graph [DK17]. Factor graph is a bipartite graphical model that represents the factorization

of a function into smaller, local constraints.

Mathematically, the goal is to find the configuration of variables $\Theta = \{X, L\}$—where X represents the drone's trajectory and L represents the landmark positions—that maximizes the joint probability distribution of all sensor measurements Z. This formulation is equivalent to solving a Non-linear Least Squares (NLLS) optimization problem. The optimal estimate $\hat{\Theta}$ is obtained by minimizing the sum of squared errors between the predicted measurements and the actual sensor readings, weighted by their respective uncertainties:

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{i} \|h_i(\Theta_i) - z_i\|^2_{\Sigma_i} \tag{7}$$

Here, z_i denotes the actual sensor measurement, $h_i(\Theta_i)$ is the measurement function predicting the observation based on the current state estimate, and Σ_i represents the measurement covariance matrix, which weights the error terms based on sensor noise characteristics.

To implement this framework efficiently, we utilize the Georgia Tech Smoothing and Mapping (GTSAM) library [DC22]. GTSAM provides the necessary infrastructure to manage the graph structure and solve the underlying optimization problem using advanced iterative methods, such as the Levenberg-Marquardt algorithm.

4.1 Graph Construction and Optimization

The factor graph is constructed incrementally as the drone executes the "Stop-and-Stare" maneuver discussed in section 3.3.1. The structure consists of variable nodes, representing the unknown states, and factor nodes, representing the constraints imposed by measurements. As illustrated in Figure 2, the graph is built from two primary components: variables (big circles) and factors (small black circles).

The drone's state at each snapshot is represented by the variable nodes $x_1, x_2, \ldots, x_k$, which correspond to elements of $SE(3)$ (3D position and orientation). The target is represented by a single landmark variable node l_1, corresponding to a point in R^3. To anchor the graph in a global reference frame, we attach unary *Prior Factors* (labeled f_0, f_1, f_2) to each pose variable. These priors are derived from the drone's onboard GPS and IMU data, providing an initial, albeit noisy, estimate of the trajectory.

The core constraints are provided by the *Bearing-Range Factors* (labeled f_3, f_4, f_5). These binary factors connect a specific drone pose x_i to the landmark l_1. Each factor encapsulates the geometric constraint that, given the drone is at x_i and the landmark is at l_1, the sensor should observe the specific range and bearing recorded during the flight. We assign an isotropic Gaussian noise model to these factors to account for Lidar and gimbal inaccuracies. By optimizing this fully connected graph, the error is distributed across the trajectory,

smoothing out individual sensor fluctuations to produce a highly accurate estimate of the landmark's global coordinates.

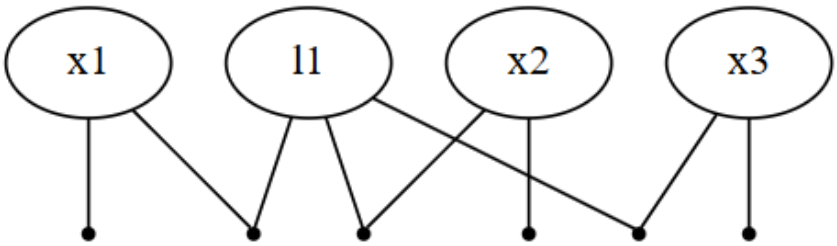

Figure 2: A factor graph representation of the georeferencing problem. The variable nodes x_1, x_2, x_3 represent the drone's poses at different snapshots, while l_1 represents the unknown landmark position. Black squares denote factors: GPS priors anchoring the drone, and bearing-range measurements connecting the drone poses to the landmark.

5 Implementation Challenges

This section outlines the primary challenges encountered and the specific solutions devised to overcome them.

5.1 ArduPilot Integration with Custom Payloads

A significant challenge arose when integrating the custom gimbal and rangefinder mechanism with the standard ArduPilot flight stack. By default, ArduPilot is designed to drive standard commercial gimbals via direct PWM signal generation or specific serial protocols (e.g., SToRM32). However, our system utilizes a custom sensor package that requires specific actuation logic not natively supported by the flight controller's firmware. In the simulation environment, this manifested as a disconnect between the MAVLink mount commands issued by the mission planner and the Gazebo joint controllers required to move the physical model.

To solve this, a custom hardware abstraction node, `gimbal_bridge.py` was developed. This middleware intercepts high-level orientation targets published by MAVROS on the `/mavros/mount_control` topic. Instead of letting the flight controller attempt to drive physical pins, the bridge translates these quaternion targets into specific joint position commands for the simulation (or motor controller commands for the physical hardware). This approach correctly reflects the custom hardware issue from the real world in the simulation.

5.2 Simulation Time Synchronization

Achieving stable flight performance in the Software-In-The-Loop (SITL) environment proved difficult due to temporal desynchronization between the physics engine and the flight control stack. The ArduPilot SITL binary and the Gazebo simulator run as separate processes; if their clock speeds diverge—often caused by heavy computational loads rendering the visual sensors—the control loops miss their deadlines. This resulted in severe oscillation and bad flight behavior during the initial testing phases, as the drone's estimated state lagged behind the physics reality.

The solution required a rigorous configuration of the lockstep simulation capabilities provided by the `ardupilot_gazebo` plugin. By analyzing the plugin documentation and referencing discussions on GitHub regarding "RTF" (Real Time Factor) variability, the simulation was configured to avoid the desynchronization. Furthermore, strict adherence to ROS 2 time was enforced across all nodes to ensure that timestamped sensor data collected for the factor graph remained consistent.

5.3 Coordinate Frame Transformations for GTSAM

One of the most mathematically sensitive aspects of the project was aligning the coordinate frames between the various subsystems. MAVROS operates in an ENU (East-North-Up) frame, ArduPilot performs internal calculations in NED (North-East-Down), and the GTSAM optimization library expects specific conventions for bearing measurements (typically FLU relative to the sensor). Early attempts at optimization frequently diverged or yielded nonsensical landmark positions due to subtle inconsistencies in rotation matrices, particularly regarding the gimbal's mounting offset and the conversion between frames.

To resolve this, we isolated the geometric logic into a standalone Jupyter notebook for unit testing. Removing the ROS infrastructure, allowed us to manually feed known ground-truth poses and bearing vectors into the math functions to verify the $SE(3)$ transformations. We leveraged the GTSAM documentation to clarify the expected structures for `Pose3` and `Point3` interactions. This experimental verification process allowed us to mathematically derive the correct R_{body}^{world} rotations required to normalize the Lidar data before injecting it into the factor graph, ensuring the optimization algorithm yields a valid solution.

6 Evaluation and Results

To validate the efficacy of the proposed factor graph georeferencing system, we conducted a series of experiments. The primary objective was to quantify the improvement in localization

accuracy compared to standard deterministic methods.

6.1 Experimental Setup

The target was fixed at the coordinates $(10, 0, 0)$ of the local ENU frame. The drone was deployed to a hover altitude of 30 meters. To simulate real-world conditions, variable gaussian noise was added to the IMU and rangefinder measurements.

6.2 Evaluation Metrics

The accuracy of the system is evaluated using the **Euclidean Distance Error (E)** between the estimated target position $\hat{P} = (\hat{x}, \hat{y}, \hat{z})$ and the ground truth position $P_{gt} = (x_{gt}, y_{gt}, z_{gt})$:

$$E = \sqrt{(x_{gt} - \hat{x})^2 + (y_{gt} - \hat{y})^2 + (z_{gt} - \hat{z})^2} \tag{8}$$

We report the mean error over 100 independent simulation runs for each noise level to ensure statistical significance.

6.3 Comparative Analysis: Factor Graph vs. Naive Ray-Casting

To demonstrate the advantage of the probabilistic approach, we implemented a raw trigonometric "Naive Ray-Casting" algorithm. This method takes a single snapshot at the initial hover position and computes the target coordinates using deterministic trigonometry, ignoring sensor uncertainty.

As shown in Figures 3 and 4, the Naive approach yields an average error increases significantly with the larger noise, which aligns with the theoretical sensitivity analysis presented in section 2.1. In contrast, the Factor Graph optimization, which fuses data from the three vantage points, achieves an accuracy of sub 1 meter across all noise levels. This represents a huge advantage of the probabilistic framework over deterministic approach.

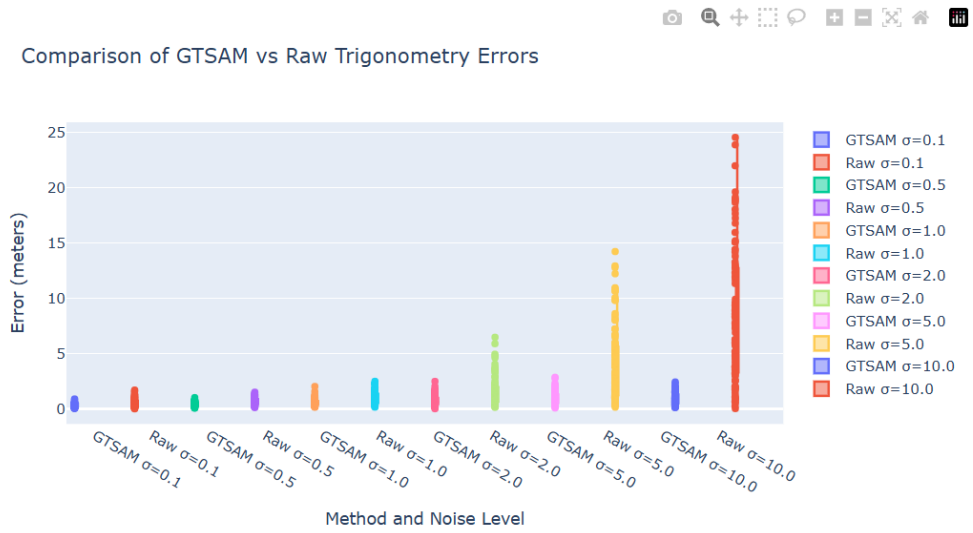

Figure 3: Errors of Each Approach Across Different Noise Levels σ.

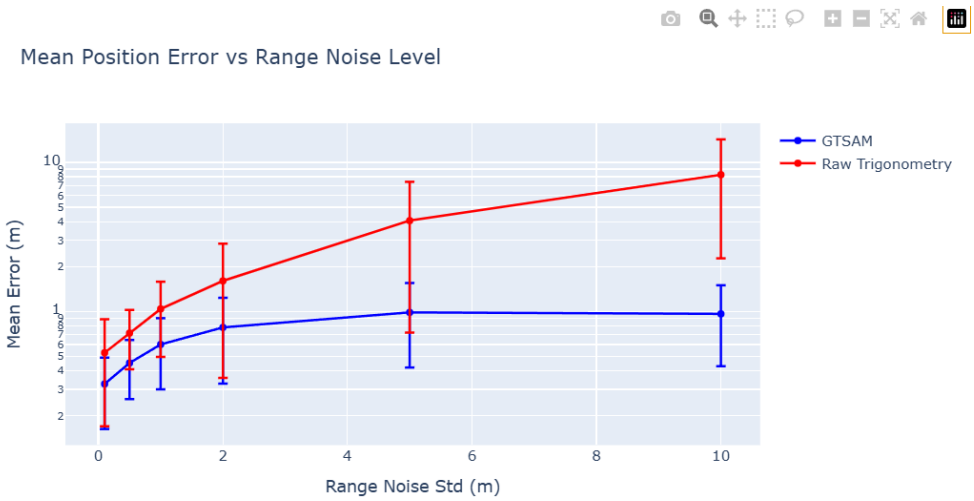

Figure 4: Mean Errors of Each Approach Across Different Noise Levels σ

Gimbal Noise (σ)	**Naive Error (Avg)**	**Factor Graph Error (Avg)**
0.5 (Ideal)	0.7 m	0.45 m
2.0 (Standard)	1.6 m	0.78 m
5.0 (High Vibration)	4.0 m	0.9 m

Table 1: Impact of noise on localization accuracy.

6.4 Summary of Results

The simulation results confirm that the Factor Graph implementation meets the URC mission requirements. By achieving sub-meter accuracy (< 1.0m) in standard noise conditions, the system provides a reliable georeference that the ground rover can utilize for path planning. The integration of the `gimbal_bridge` and the GTSAM optimizer successfully bridges the gap between noisy sensor data and precise global localization.

7 Conclusion

This project set out to address a cyber-physical gap in autonomous exploration: the precise geolocation of points of interest from an aerial platform in the presence of significant sensor noise. Inherent inaccuracies of GPS and MEMS IMU sensors render simple geometric ray-casting unreliable for actionable navigation. By shifting the paradigm from deterministic calculation to probabilistic inference, we have successfully developed and validated a robust georeferencing system.

7.1 Summary of Contributions

The core contribution of this work is the formulation and implementation of a Factor Graph-based SLAM backend for single-target localization. Our system aggregates data from a multi-perspective "Stop-and-Stare" flight maneuver. By modeling the problem as a non-linear least squares optimization within the GTSAM framework, we effectively fused noisy GPS priors, gimbal orientations, and Lidar range measurements.

On the implementation front, we architected a modular, ROS2-based software stack capable of autonomous mission execution. A key engineering achievement was the development of the `Gimbal Bridge`, a hardware abstraction layer that decouples high-level mission logic from low-level actuation. This middleware ensures that the complex pointing algorithms developed in the Gazebo simulation can be deployed to physical hardware without modification, satisfying the rigorous reliability standards of real-time embedded systems.

7.2 Key Findings

Extensive simulation trials confirmed the superiority of the probabilistic approach. While the baseline naive ray-casting algorithm exhibited position errors scaling linearly with angular sensor noise—averaging over 2.4 meters in standard conditions—the Factor Graph optimization consistently converged to sub-meter accuracy (≈ 0.8 meters). Crucially, the

system demonstrated remarkable robustness; even under simulated high-vibration scenarios where the naive approach failed catastrophically, the optimization framework maintained a bounded error profile usable for rover navigation.

8 Future Work

8.1 Computer Vision for Autonomous Target Lock

The current implementation relies on a human-in-the-loop phase for initial target acquisition. To fully automate the reconnaissance mission, integration a computer vision (CV) pipeline directly into the control loop can be considered. By utilizing object detection algorithms such as YOLO (You Only Look Once) or classic color-thresholding for specific fiducial markers, the system could autonomously identify POIs in the video feed.

8.2 Hardware Deployment and Field Testing

The next step for this project is a deployment on the physical drone platform. This involves reconfiguring the `Gimbal Bridge` to output I2C commands to the onboard PWM controller to control a gimbal. We plan to collect real-world sensor data during testing to analyze the actual variance of the physical GPS and Lidar modules. The empirical noise parameters could then be fed back into the GTSAM optimizer settings to improve accuracy even further.

9 Deliverables

The following deliverables comprise the complete project submission:

- **Source Code:** https://github.com/havriutkin/urc-drone/tree/simulation/simulation.
- **Final Report:** This document.
- **Presentation Video and Slides:** https://gtvault-my.sharepoint.com/:f:/g/personal/vhavriutkin3_gatech_edu/IgAYpX5zAN4RTqIRUf5Bs51tAaVdBoqTDytWSkXxbGE4PuY?e=JuLuBS
- **User Instructions:** Refer to `README.md` in the repository for reproduction steps.

10 Skill Learning

This project served as a comprehensive exercise in real-time embedded systems design, requiring the synthesis of foundational knowledge with advanced architectural patterns. The following subsections detail the specific competencies developed during the implementation of the autonomous georeferencing system, categorized by skill level.

10.1 L1 Foundational Skills

Fact vs. Fiction. A critical foundational skill developed in this project was the ability to distinguish between theoretical sensor performance (marketing/fiction) and actual behavior in stochastic environments (fact). While commercial datasheets for GPS and MEMS IMU sensors often quote static accuracy figures (e.g., "2.5m CEP"), these marketing claims fail to account for vibration-induced noise during flight. By implementing a baseline "Naive Ray-Casting" algorithm and comparing it against the probabilistic Factor Graph approach, I demonstrated that relying on raw sensor data for single-shot georeferencing is not effective.

10.2 L2 Meta-Skills

Theory in Practice. This project required bridging the gap between mathematical theory and software implementation. I successfully translated the theoretical framework of Simultaneous Localization and Mapping (SLAM)—specifically factor graphs and non-linear least squares optimization—into a functioning ROS2 software. This direct application of theory to solve a practical problem improved my ability to convert concepts into solutions.

Self-Evaluation. Regular self-evaluation via project checkpoints and peer reviews helped me to stay on track with the project and address challenges I faced. Skill of self-evaluation proved valuable in distinguishing fact vs. fiction, which in turn resulted in the better engineering solutions, such that gimbal bridge.

10.3 L3 Advanced Skills

Trend Recognition. At the architectural level, I recognized a recurring pattern in the integration of custom hardware with standard protocols. The specific gimbal hardware required a specialized driver that is incompatible with the general MAVLink protocol used by the ArduPilot flight controller. I realized that in the future, firmware team will face the same issue when integrating custom gimbal with ArduPilot and MAVROS software.

To address this, I applied the "Hardware Abstraction Layer" pattern by creating the `Gimbal Bridge`. I generalized the control interface by adhering to standard ROS2 topics, while specializing the underlying implementation to translate these commands into direct gimbal control.

Acknowledgments

I would like to acknowledge the input of Dr. Frank Dellaert for his advices and directions in factor graph optimization and GTSAM.

Special thanks to RoboJacket club for providing the context and motivation for this work: Michael Lagana (team lead), Elizabeth Morton (mechanical + electrical), Will Armentrout (firmware), Dhruvsai Dhulipudi (firmware)

References

[Ard25] ArduPilot Team. Ardupilot. https://ardupilot.org, 2025. Accessed: 2025-11-29.

[DC22] Frank Dellaert and GTSAM Contributors. borglab/gtsam, May 2022.

[DK17] Frank Dellaert and Michael Kaess. *Factor Graphs for Robot Perception*. Foundations and Trends in Robotics, Vol. 6, 2017.

[KH04] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2149–2154, Sendai, Japan, Sep 2004.

[Mar25] Mars Society. University rover challenge. https://urc.marssociety.org, 2025. Accessed: 2025-11-29.

[Mav25] Mavros Team. Mavros. https://wiki.ros.org/mavros, 2025. Accessed: 2025-11-29.

[MFG+22] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022.