Assignment - ground vehicles localization - Visual Localization

Overview

Design, implement, and test a **visual localization approach** leveraging the provided data streams from an UGV system. The aim is to evaluate your ability to:

- Analyze available data
- Propose a technically localization method using vision
- Prototype or tune part of the system
- Document your reflection, architecture choices, and results

Note: This assignment focuses on demonstrating your **problem-solving approach** and **engineering judgment** rather than achieving perfect results.

Context

You're joining the **Autonomous UGV Localization team**. A robot “mbuggy” operates in **outdoor environments**. You’re given a ROS bag with about **5 minutes** of driving data that includes:

- A front monocular camera (rectified stream)
- GNSS/INS-based odometry
- IMU
- Transform trees for coordinate frame management

Available ROS Bag Topics

Link: visual-localization

Topic	Type	Description
`/mbuggy/camera_front/camera_info`	`sensor_msgs/CameraInfo`	Camera intrinsics
`/mbuggy/camera_front/image_rect`	`sensor_msgs/Image`	Rectified front camera stream
`/mbuggy/imu_ins`	`sensor_msgs/Imu`	IMU data
`/mbuggy/odom`	`nav_msgs/Odometry`	Ground truth odometry
`/tf`, `/tf_static`	`tf2_msgs/TFMessage`	Transform tree

Note: As with real-world deployments, the image stream may exhibit occasional frame drops or slight irregularities. These are typical of field-collected data and should be taken into account in your analysis and system design.

Your Tasks

1. Data Exploration

Goal: Get familiar with the dataset and assess its quality.

- Check message frequencies and synchronization
- Inspect camera and IMU signal quality
- Plot the ground truth trajectory to understand the motion

Deliverable: A short summary with key plots and findings

---

2. Strategy & Method Selection

Goal: Choose a visual localization approach based on the available data.

- Briefly describe your chosen method (e.g., VO, VIO, hybrid)
- You can use an open-source SLAM/VIO package or build a basic custom pipeline
- Justify your choice based on practicality, robustness, and time constraints

Deliverable: A short write-up explaining your approach and reasoning

---

3. Implementation

Goal: Build a minimal working setup on the provided bag.

Setup:

- Install and configure your chosen tool or framework
- Create basic launch and config files

Integration:

- Adjust frame IDs and coordinate transforms
- Set initial parameters and input topics
- Log the estimated trajectory in a usable format

Deliverable: A working ROS pipeline with launch/config files
(Optional: include a Dockerfile or environment setup for reproducibility)

---

4. Testing & Basic Analysis

Goal: Evaluate the initial performance and identify strengths/weaknesses.

- Run the full bag and log the estimated trajectory
- Compare it to `/mbuggy/odom` (reference)
- Plot both trajectories for visual comparison

- Estimate simple drift or position error metrics
- Identify a few success/failure moments (e.g., where it diverges or holds well)

Deliverable: Plots, metrics, and 1–2 paragraphs of commentary

---

5. Documentation

Goal: Summarize your process and results clearly.

- What method did you try, and why?
- What worked well? What didn't?
- What would you improve next?

Deliverable: A concise technical report (PDF or Markdown)

Hints & Tools

- Use `rqt`, `rviz`, `tf_echo`, `image_view`, `plotjuggler`, or any tools you prefer like evo
- Time synchronization can be checked via topic timestamps or rqt_bag
- If you use SLAM packages, adapting them to rectified camera stream and frame IDs is crucial
- Consider recording output into another bag or CSV for comparison plots
- If you include a `Dockerfile` or setup instructions, it'll make your work easier to test and review.