Boston Dynamics

Final Presentation

Team 4 : WeiJen Lu, Lan Qing, Sekito Shinjo, Kota Teshima, Atsuki Abe

Aug. 27th, 2025

Agenda

1. Introduction
2. Method
3. Results
4. Conclusion

Introduction

Objective

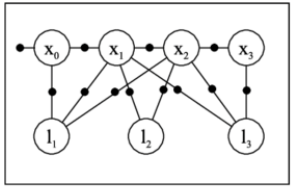
1. Application of Graph-SLAM to **KITTI model dataset**
2. Application of Graph-SLAM in 1. to **2D images and videos** (Simple Demo)
3. Real-time demonstration of feature point extraction and trajectry estimation

Motivation

1. Understand Graph-SLAM procedure more deeply by applying it to real situation
2. To evaluate **how precise** the model can perform Graph-SLAM with real data
3. To know the **possibility and limitation** of real-time trajectory estimation

Introduction

What is Graph SLAM ?



1. Estimate **odometry** (several types of data)

2. Observation constrains and optimization

https://www.researchgate.net/figure/Factor-graph-representation-of-the-Full-SLAM-problem-for-both-the-simple-example-and-the_fig3_221344652

Method

How to gain the necessary data

	Groud Truth	Odometry	Observation
Visual Odom on KITTI	Already Given (GPS)	LK Feature tracking & matching	-
Simple Demo	Based on ArUco (didn't go well)	From image feature points and ArUco	Based on ArUco
Real-time Demo	-	LK optical flow + PnP Ransac	ORB Matghing

Dataset (Visial Odometry on KITTI)

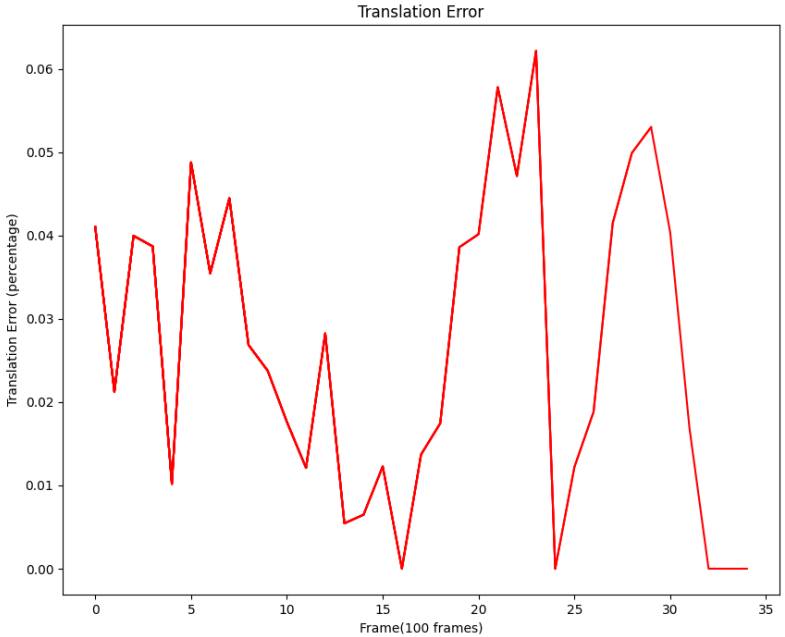
KITTI Odometry Dataset

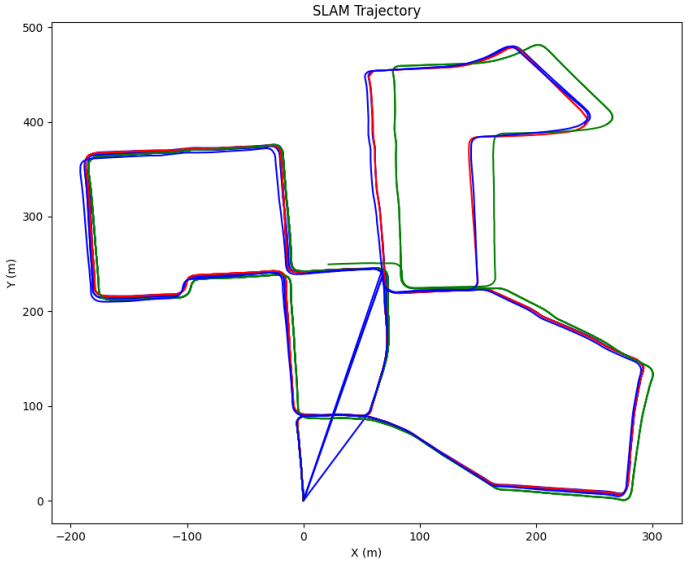
Large-scale benchmark widely used in computer vision and robotics research.

1. Photos taken by left and right camera with provided intrinsics and extrinsics

2. time of taken photos
3. ground truth trajectory
    a. collected using Real-Time Kinematic GPS (cm level percision) and IMU

Result (Visial Odometry on KITTI)



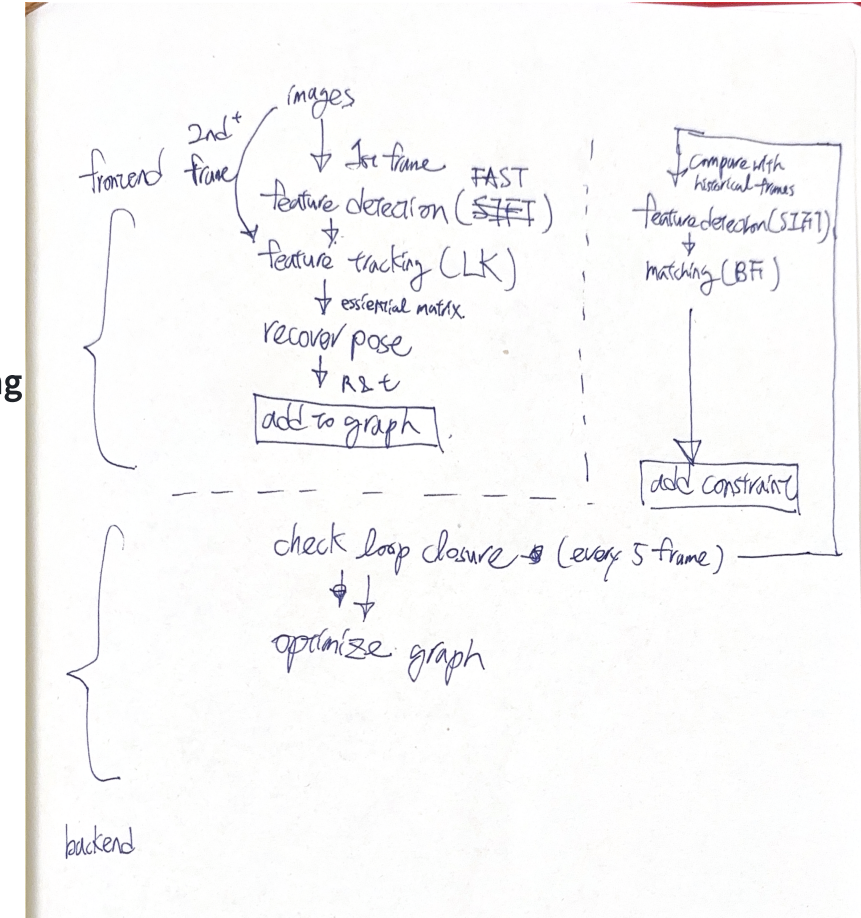


Method (Visial Odometry on KITTI)

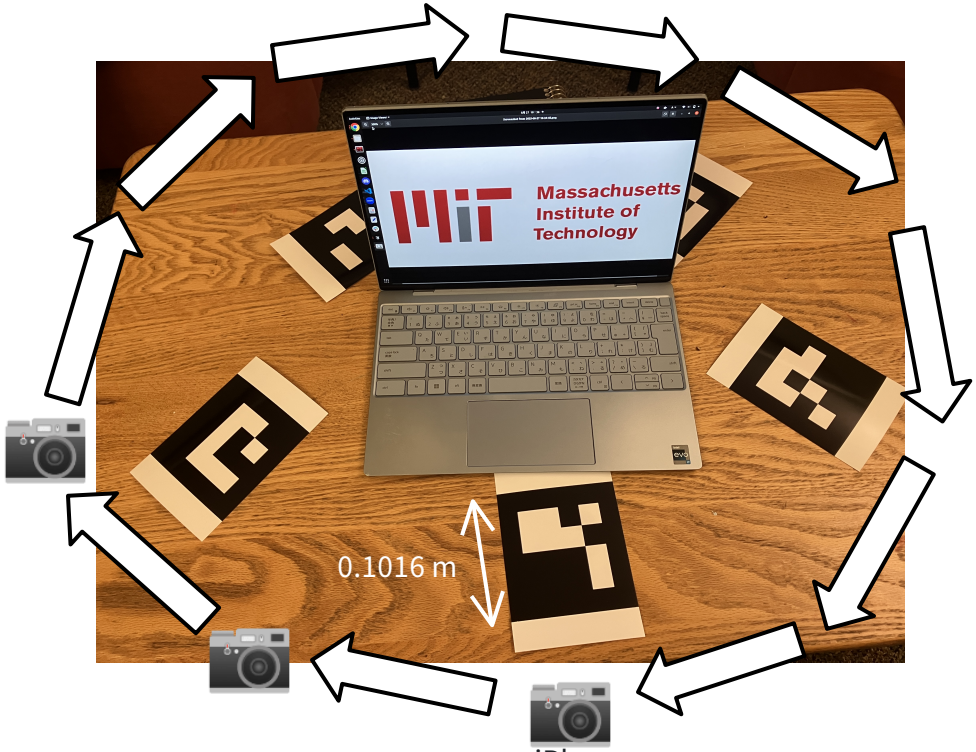
Odometry
continous frame **FAST+ LK feature tracking** to save up resources
->essiential matrix -> recover pose-> R & t

Loop Closure
Trigers checking every five frames
check current frame with historical frame using **SIFT+BFMatching**
if it is triggered then optimized

Graph Implementation
through package g2o
use SparseOptimizer & Levenberg-Marquardt (LM) Solver



Method (Simple Demo)

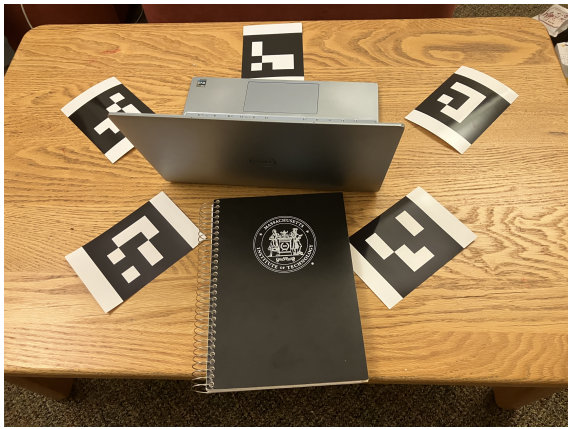


iPhone

(intrinsic *K* was calibrated with checker-board)

- 11 RGB images with 0.1016 m squared ArUco Markers

Opposite side

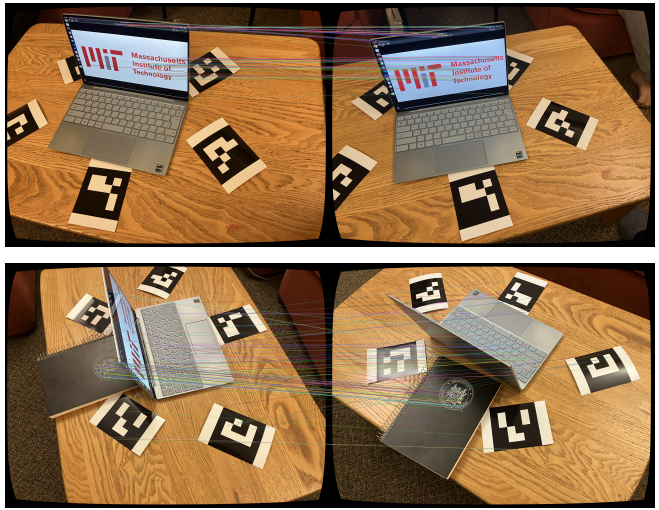


Method1 (Simple Demo)

Ground Truth (GT) construction
Define a 2D world frame centered at the centroid of the detected markers.
→ Detect visible ArUco markers and run PnP.

Odometry
Feature points (ORB or SIFT) matching
→ Relative pose estimation (recoverPose ~~or PnP~~)
→ Scale adjustment (with ArUco Markers)

SIFT Matching



Observations (landmark measurements)
Treat detected ArUco markers as absolute landmarks with known world positions.

Compute distance and bearing from the current GT pose to each visible landmark, and create HalfEdge observations tied to the current guessed pose.
→ Observations sharing the same landmark ID are paired to create graph Edges for optimization.

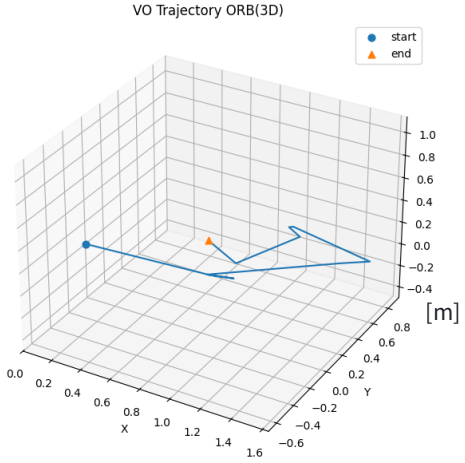
Results1 (Simple Demo)

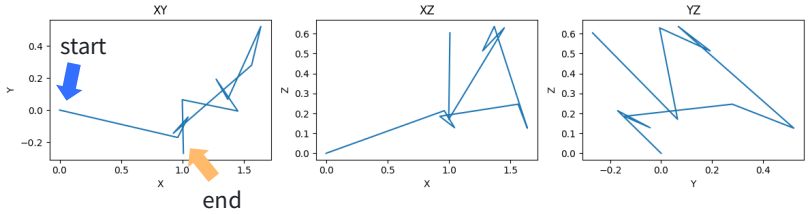
Odometry

Comparison between ORB and SIFT

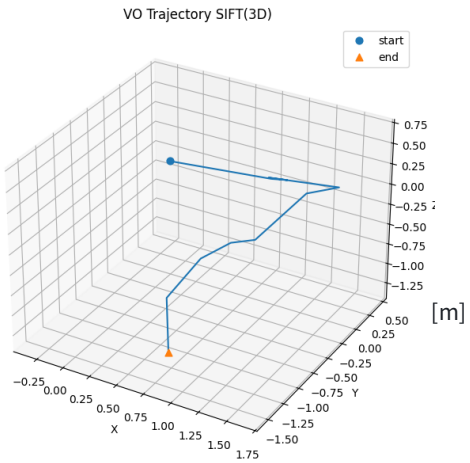
- Odometry with both feature extraction methods drifted
- The reason is probably the cyclic motion was difficult to estimate relative pose -> Tried PnP, but didn't work well

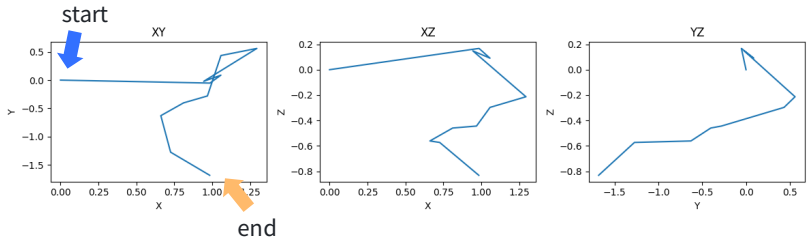
- ORB (Light)





- SIFT (Precise but severer)



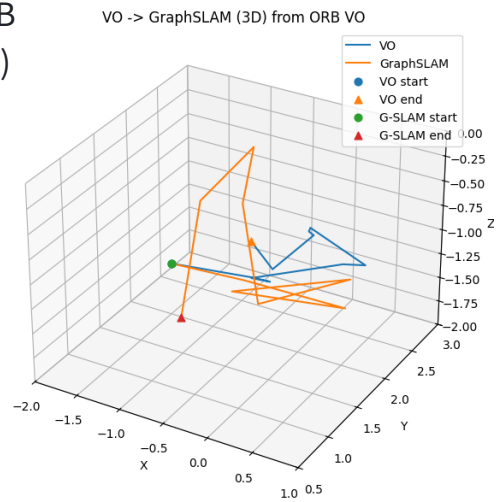


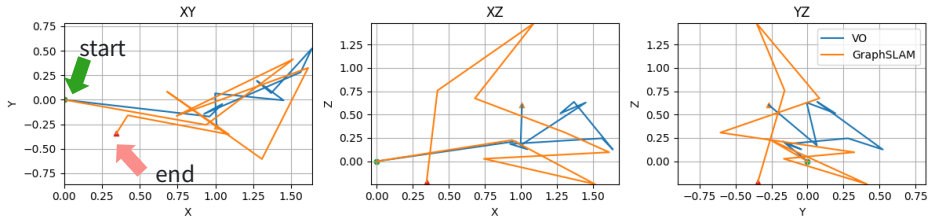
Results1 (Simple Demo)

Graph-SLAM

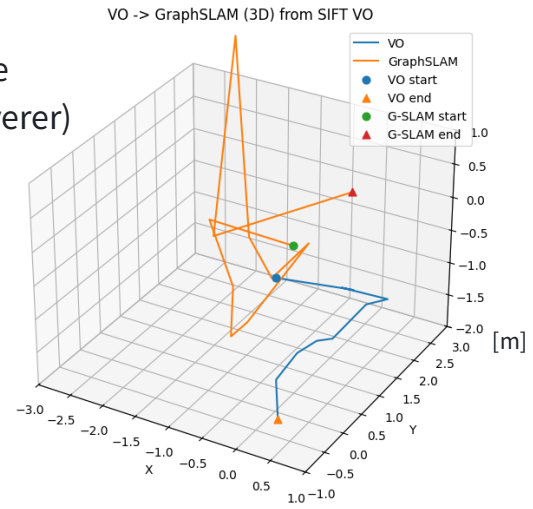
The endpoint barely more converged, but it exhibited behavior suggesting forced convergence, likely due to **drift in the VO**. Seeking a better pose estimation through methods like bundle adjustment (BA) might be useful.
ORB was somewhat better. WHY?

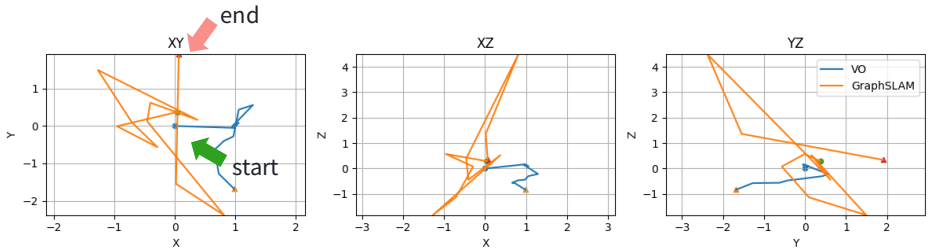
- ORB (Light)





- SIFT (Precise but severer)

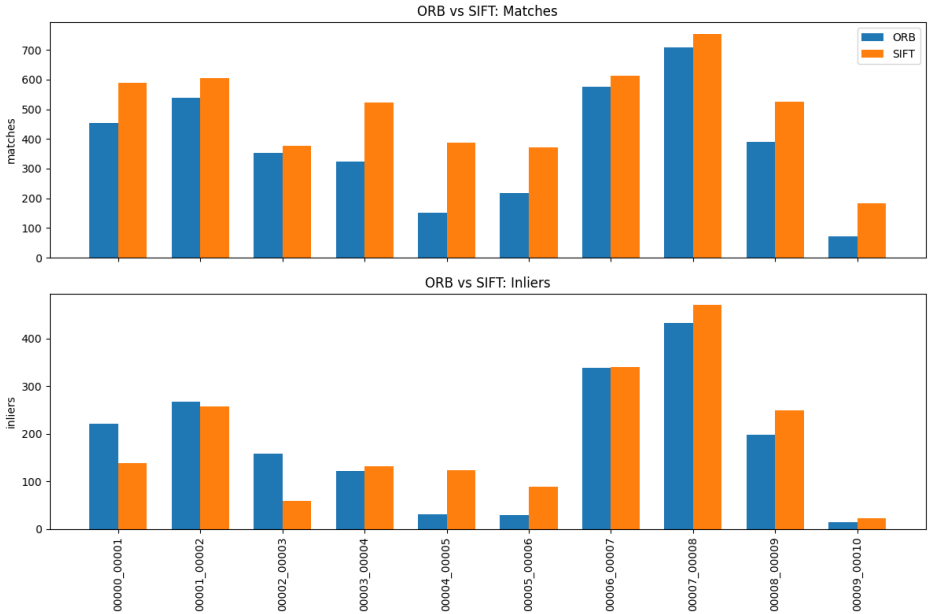




Results1 (Simple Demo)

Graph-SLAM

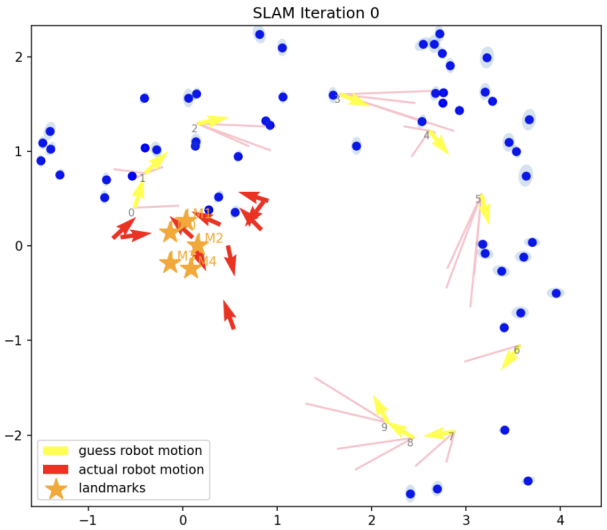
- Number of matches -> There are not large differences



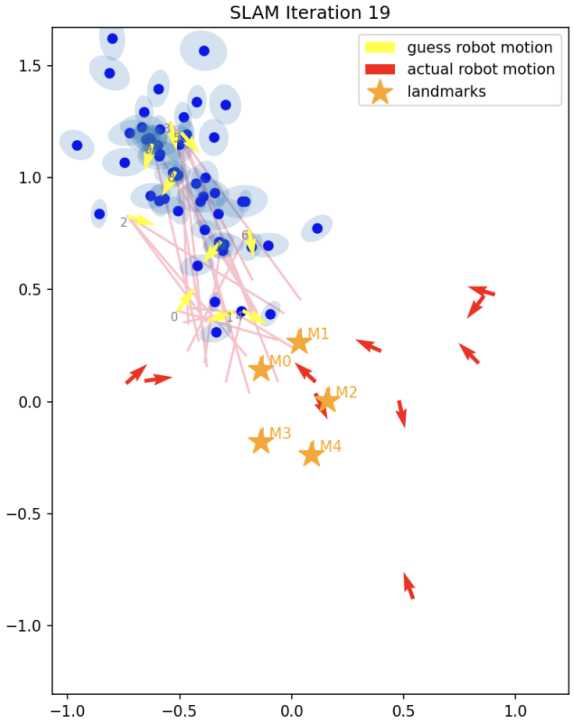
For improvement (couldn't test due to time constraints...)

- Ground truth (!!)
- Bandle adjustment before Graph-SLAM
- Parameter tuning in SIFT matching

Results1 (Simple Demo)



Work not well...



At Iteration 0, the trajectory and landmarks were largely misaligned, but by Iteration 19, they had **converged into a coherent map.**

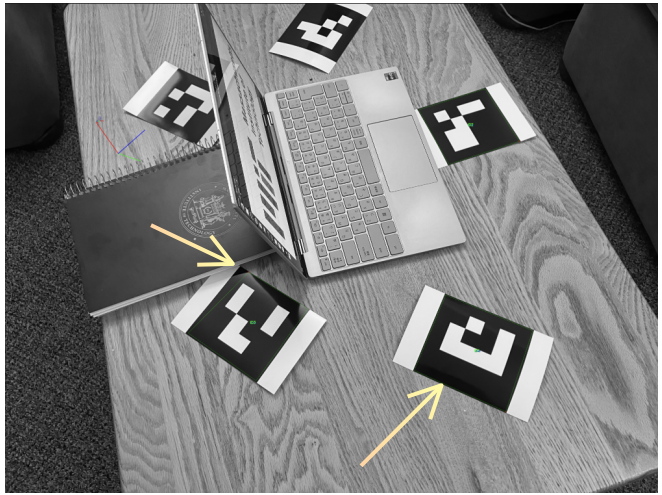
Conclusion / Acknowledgement (Simple Demo)

.

Through the updates of Graph-SLAM, the robot trajectory and landmark positions gradually become **consistent.**

This demonstrates that the errors in odometry and observation information (landmark detection) can be reconciled through optimization.

On the other hand, the ground truth itself is not accurately obtained, and although the trajectory has become more consistent, it is still **NOT** precise.
→ This is likely because the ArUco markers were printed as **photographs,** which often led to poor recognition.
(Reflection, curved, etc...)



Results (Realtime Demo)

A lightweight, real-time Visual SLAM pipeline in Python for Intel RealSense RGB-D cameras.
It tracks camera motion using ORB features + LK optical flow, estimates poses via PnP / Essential matrix, performs (optional) loop-closure pose-graph optimization, and renders a 2D top-down trajectory.

Results (Realtime Demo) _ Algorithm

1) Input & Preprocessing
Fetch synchronized color (BGR8) and depth (meters) frames from RealSense.
2) Tracking Frontend
Keyframes: Run ORB detection (depth-valid mask) and compute descriptors.
Between frames: Track keypoints via LK optical flow with forward–backward error check to keep only reliable tracks.
3) Motion Estimation
Primary: 3D–2D PnP (RANSAC) using 3D points from the previous frame (back-projected by depth) and current 2D tracks.
Fallback: Essential matrix (RANSAC) + recoverPose when depth-based PnP is insufficient.
4) Keyframe Decision
Create a keyframe when (a) track count low, (b) rotation exceeds threshold, or (c) a minimum time passes.

5) Loop Closure
For a new keyframe, search a top-1 candidate in the DB using BF-Hamming + Lowe's ratio;
Enforce both index gap and time gap (e.g., ≥5 s) to avoid trivial/early matches;
Geometric verification via PnP; optional ICP refinement (Open3D);
If accepted, add a loop edge and run Open3D global optimization ("Loop Closer Detected").
6) Robustness: Interruption & Re-localization
If pose estimation fails several consecutive frames, pause localization to avoid drift ("Localization Interrupted").
Try to re-localize by matching the current frame to keyframes (PnP verification).
If the estimated pose is close to the pre-interruption pose (position & yaw thresholds), resume from there ("Localization Restart").

Results (Realtime Demo) _ Key features & Parameters

Key Features

RGB-D input (RealSense): Depth aligned to color.

Lightweight frontend: ORB only on keyframes + LK optical flow with forward–backward check → per-frame cost dominated by LK + PnP, not feature detection.

Depth-aware processing: Valid-depth masks for ORB and back-projection (higher inlier ratio, lower compute).

Pose estimation: 3D–2D PnP (RANSAC) as primary; Essential matrix fallback when depth is insufficient.

Keyframe policy: Insert by track drop, rotation threshold, or minimum time gap.

Loop closure: BF-Hamming + Lowe's ratio (top-1), index & time gap gating, PnP verification, ICP refinement, Open3D pose-graph optimization.

Robustness: Localization interruption on repeated failures; re-localization via keyframe matching. Status via overlay or console.

Parameters

RealSense: width, height, fps, depth_min/max.

Features/Tracking: orb_nfeatures (default 1200), orb_fast_threshold, lk_win_size, lk_max_level, lk_fb_thresh_px.

PnP/Essential: pnp_reproj_px, pnp_iter, pnp_conf, pnp_min_inliers, ess_thresh_px, ess_conf, ess_min_inliers.

Keyframe Policy: kf_min_tracks, kf_min_rot_rad, kf_min_dt_sec.

Loop Closure: loop_min_gap (index gap), loop_min_time_gap_sec (time gap), loop_ratio, loop_min_good, loop_pnp_min_inliers.

ICP: use_icp, icp_voxel, icp_max_corr_dist, icp_max_iter, icp_min_fitness.

Pose Graph: information diagonal and optimizer iterations.

Visualization: viz2d_*, show_windows.

Results (Realtime Demo) _ Insight

Works well

Textured, static scenes: Rich corners/edges; limited dynamic objects.

Moderate inter-frame motion: Pyramidal LK can track; low motion blur and rolling-shutter distortion.

Valid, stable depth: Surfaces within depth_min–depth_max, few holes; good depth–color alignment.

Planar-ish, smooth trajectories: Small per-frame rotations; occasional keyframes.

Distinctive places for loops: Repeated views with consistent lighting; enough ORB matches → reliable loop closure + pose-graph optimization.

Struggles / failure modes

Low texture / glossy / transparent surfaces: Weak gradients; LK drifts or breaks; ORB yields few stable matches.

Fast motion / heavy blur / large rotations: LK linearization breaks despite pyramid; PnP inliers drop → interruption triggers.

Depth issues: IR interference, sunlight, speculars → noisy/invalid depth; back-projection degrades PnP.

Strong dynamics: Many moving objects violate static-scene assumption; RANSAC inlier ratio falls.

Degenerate geometry: Nearly co-planar or tiny baseline scenes weaken PnP/Essential constraints.

Sparse or false loops: Few distinctive features or lookalikes → loop closure rare or rejected by PnP/ICP.

Built-in mitigations (in this code)

CPU-friendly front-end: ORB only on keyframes; per-frame cost dominated by LK+PnP.

Forward–backward check & interruption: Drop bad tracks; pause localization after repeated failures; re-localize via KF matching.

Thank you for listening!