

Lab 6 Report

Robotics Integration Group Project I

Yuwei ZHAO (23020036096)
Group #31 2025-12-24

Abstract

...

See Resources on github.com/RamessesN/Robotics/MIT.

1 Introduction

2 Procedure

2.1 Individual Work

2.1.1 Nister's 5-point Algorithm

Read the following paper.

[1] Nistér, David. “An efficient solution to the five-point relative pose problem.” 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Vol. 2. 2003. link [here](#).

Questions:

1. Outline the main computational steps required to get the relative pose estimate (up to scale) in Nister's 5-point algorithm.

Step I — Extract Nullspace Extraction:

Build a 5×9 matrix, which consists of an epipolar constraint of 5 pairs of points, and calculate the zero-space vector $\tilde{X}, \tilde{Y}, \tilde{Z}, \tilde{W}$ of the matrix. This step usually utilizes QR decomposition or SVD to achieve.

Step II — Expand cubic constraints:

Using the properties of the essential matrix, namely the trace constraint $2EE^T E - \text{trace}(EE^T)E = 0$. Express the essential matrix as a linear combination of null space vectors $E = xX + yY + zZ + \omega W$, and then expand these cubic constraints.

Step III — Gauss-Jordan Elimination:

Build a 9×20 matrix A , and implement “Gauss-Jordan Elimination” to it.

Step IV — Build & Solve Polynomial Formulation:

Expand two 4×4 determinant polynomial of matrixes B and C . Then get the 10th degree polynomial about the variable z through elimination.

Step V — Root Extraction:

Solve for the real roots of this 10th-degree polynomial. This can be done by the Sturm sequence or by the eigenvalue decomposition of the adjoint matrix.

Step VI — Recover Pose & Disambiguation:

For each real root, the corresponding essential matrix E is recovered, and then the rotation matrix R and translation vector t are decomposed. The points are triangulated using Chirality constraint (i.e. points must be in front of the camera), and the only correct physical and geometric solution is selected from a maximum of 10 possible mathematical solutions.

2. Does the 5-point algorithm exhibit any degeneracy? (degeneracy = special arrangements of the 3D points or the camera poses under which the algorithm fails)

Conclusion — In the main context discussed in the paper (i.e. compared to uncalibrated methods), the 5-point algorithm overcomes Planar Structure Degeneracy, which is one of the big advantages over 7 or 8-point algorithms.

Explanation:

1. **Non-degradation of planar scenes:** Section 4 of the paper makes it clear that for uncalibrated methods, the algorithm fails when the scene points are coplanar (there is a continuous solution). However, under the calibrated 5-point method setting, coplanar points lead to at most 2-fold ambiguity, which can usually be resolved by a third view, or excluded by chiral constraints. Therefore, the paper concludes that the algorithm still works well in planar scenes.
2. **General degradation:** While the paper focuses on its robustness in planar scenarios, according to the basics of epipolar geometry (and the implicit assumption in the paper), the essential matrix is not definable (baseline $t = 0$) without translational motion (pure rotation), at which point the algorithm degrades. In addition, multiple or infinite solutions also exist if 5 points and two optical centers lie on a specific critical surface such as twisted cubic, but it is considered robust for practical applications and planar scene comparisons highlighted in the paper.
3. **When used within RANSAC, what is the expected number of iterations the 5-point algorithm requires to find an outlier-free set?**

From the standard theory of RANSAC, we have the expected number of iterations N is determined by

$$N = \frac{\log(1 - p)}{\log(1 - \omega^s)}$$

which $\begin{cases} s=5: \text{The minimum number of points required by the 5-point algorithm, i.e. sample size} \\ \omega(\text{Inlier Ratio}): \text{Proportion of inliers (i.e. the probability of selecting an inlier point in a sample)} \\ p(\text{Confidence}): \text{The confidence that we want to find at least one full set of interior points} \end{cases}$

Since the term ω^s decreases exponentially with s , a smaller sample size s significantly reduces the required number of iterations. For the 5-point algorithm ($s = 5$), the probability of selecting an outlier-free set is ω^5 , which is much higher than for the 7-point ($s = 7$) or 8-point ($s = 8$) algorithms. This reduction is critical for real-time structure and motion estimation.

E.g. Assuming a typical inlier ratio $\omega = 0.5$ and a desired confidence $p = 0.99$, the required iterations N are:

Method	Sample Size (s)	Required Iterations (N)
5-point algorithm	5	$\left\lceil \frac{\log(0.01)}{\log(1 - 0.5^5)} \right\rceil = 145$
7-point algorithm	7	$\left\lceil \frac{\log(0.01)}{\log(1 - 0.5^7)} \right\rceil = 588$
8-point algorithm	8	$\left\lceil \frac{\log(0.01)}{\log(1 - 0.5^8)} \right\rceil = 1177$

Additionally, if we consider the strictly theoretical mean number of trials $E[k]$ required to encounter the first outlier-free set (geometric distribution expectation), it is given by:

$$E[k] = \frac{1}{\omega^s}$$

For $\omega = 0.5$, the 5-point algorithm requires on average 32 trials, whereas the 8-point algorithm requires 256 trials. This efficiency allows the algorithm to be executed for hundreds of samples within a small time budget.

2.1.2 Designing a Minimal Solver

Can you do better than Nister? Nister's method is a minimal solver since it uses 5 point correspondences to compute the 5 degrees of freedom that define the relative pose (up to scale) between the two cameras (recall: each point induces a scalar equation). In the

presence of external information (e.g., data from other sensors), we may be able use less point correspondences to compute the relative pose.

Consider a drone flying in an unknown environment, and equipped with a camera and an Inertial Measurement Unit (IMU). We want to use the feature correspondences extracted in the images captured at two consecutive time instants t_1 and t_2 to estimate the relative pose (up to scale) between the pose at time t_1 and the pose at time t_2 . Besides the camera, we can use the IMU (and in particular the gyroscopes in the IMU) to estimate the relative rotation between the pose of the camera at time t_1 and t_2 .

You are required to solve the following problems:

- Assume the relative camera rotation between time and is known from the IMU. Design a minimal solver that computes the remaining degrees of freedom of the relative pose.**

For the matching points p_1, p_2 on the normalized img plane, the epipolar constraint is

$$p_2^T E p_1 = 0$$

which essential matrix is $E = [t]_x R$.

Put E into the known rotation, we have

$$p_2^T ([t]_x R) p_1 = 0$$

Let $p'_1 = Rp_1$, then we have $p_2^T [t]_x p'_1 = 0$ ①

∴ ① is equivalent to scalar triple product

∴ $t \cdot (p'_1 \times p_2) = 0$, which means the translation vector t must be perpendicular to the vector

$$v = p'_1 \times p_2$$

∴ 1 point can only ensure t is on some plane

∴ we need 2 points to ensure the direction of t

$$\begin{cases} \text{For 1^{st} pair points: normal vector } v_1 = (Rp_{1,a}) \times p_{2,a} \\ \text{For 2^{nd} pair points: normal vector } v_2 = (Rp_{1,b}) \times p_{2,b} \end{cases}$$

∴ translation vector t is perpendicular to v_1, v_2 , thus

$$t \sim v_1 \times v_2$$

Then after normalizing, we have

$$t \leftarrow \frac{t}{\|t\|}$$

- Describe the pseudo-code of a RANSAC algorithm using the minimal solver developed in point a) to compute the relative pose in presence of outliers (wrong correspondences).**

```
1 Algorithm: RANSAC for 2-point Relative Pose with Known Rotation Typst
2
3 Input:
4 Matches: A set of N feature correspondences  $S = \{(p_{1,i}, p_{2,i})\}$ 
5 Rotation: Relative rotation matrix  $R$  (from IMU)
6 Threshold: Inlier threshold  $\epsilon$  (e.g., epipolar distance)
7 Confidence: Desired confidence probability  $p$  (e.g., 0.99)
8 InlierRatio: Estimated ratio of inliers  $w$  (e.g., 0.5)
9
10 Output:
11 Best_t: The estimated translation direction
12 Best_Inlier_Set: The set of consistent matches
13
14 1. Max_Iterations =  $\log(1 - p) / \log(1 - w^2)$ 
15 2. Best_Score = 0
16 3. Best_t = [0, 0, 0]
17
18 4. For  $k = 1$  to Max_Iterations:
19
20     a. // Sampling
21         Select 2 random correspondences  $\{(p_{1,a}, p_{2,a}), (p_{1,b}, p_{2,b})\}$  from  $S$ .
22
23     b. // Model Generation
24          $p_{1\text{prime},a} = R * p_{1,a}$ 
25          $p_{1\text{prime},b} = R * p_{1,b}$ 
26          $v_1 = \text{cross\_product}(p_{1\text{prime},a}, p_{2,a})$ 
27          $v_2 = \text{cross\_product}(p_{1\text{prime},b}, p_{2,b})$ 
28          $t_{\text{candidate}} = \text{cross\_product}(v_1, v_2)$ 
29         Normalize  $t_{\text{candidate}}$ 
30
31     c. // Scoring (Count Inliers)
32         Current_Score = 0
33         Current_Inliers = {}
34          $E_{\text{candidate}} = [t_{\text{candidate}}]_{\text{cross}} * R$  // Essential Matrix
35
36     For each match  $(p_{1,i}, p_{2,i})$  in  $S$ :
37         // Calculate Sampson error or simple algebraic error
38         error =  $\text{abs}(p_{2,i}^T * E_{\text{candidate}} * p_{1,i})$ 
39
40         If error < epsilon:
41             Current_Score = Current_Score + 1
42             Add  $(p_{1,i}, p_{2,i})$  to Current_Inliers
43
44     d. // Update Best Model
45     If Current_Score > Best_Score:
46         Best_Score = Current_Score
```

```

47     Best_t = t_candidate
48     Best_Inlier_Set = Current_Inliers
49
50 5. // Refinement
51     Re-estimate t using all points in Best_Inlier_Set via Linear Least
52     Squares or SVD.
53 6. Return Best_t

```

which

- Sample Size — It merely needs to sample 2 points, as opposed to 5 for the Nister algorithm, which means that the number of cycles of RANSAC will be greatly reduced ($N \propto \omega^{-2}$ vs $N \propto \omega^{-5}$), which greatly improves the computation speed.
- Model Validation — epipolar geometric error $p_2^T E p_1$ is used for validation for R is known and t is assumed so that E is ensured.

2.2 Team Work

2.2.1 Initial Setup

This section is to calibrate the camera of the drone, namely to obtain the camera intrinsics and distortion coefficients.

Therefore, we need to check the implementation of `calibrateKeypoints` at first — Using the camera's intrinsic parameters and distortion coefficients, convert the pixel coordinates into undistorted 3D direction vectors. The corresponding code is as follows:

```

1 void calibrateKeypoints(const std::vector<cv::Point2f>& pts1,           C++
2                               const std::vector<cv::Point2f>& pts2,
3                               opencv::bearingVectors_t& bearing_vector_1,
4                               opencv::bearingVectors_t& bearing_vector_2) {
5     //
6     // For this part, we perform:
7     //   1. Use the function cv::undistortPoints to rectify the keypoints.
8     //   2. Return the bearing vectors for each keypoint.
9     //
10
11     std::vector<cv::Point2f> points1_rect, points2_rect;
12     cv::undistortPoints(pts1, points1_rect, camera_params_.K,
13                         camera_params_.D);
14     cv::undistortPoints(pts2, points2_rect, camera_params_.K,
15                         camera_params_.D);

```

```

14
15     for (auto const& pt: points1_rect){
16         opengv::bearingVector_t bearing_vector(pt.x, pt.y, 1); // focal
17         length is 1 after undistortion
18         bearing_vector_1.push_back(bearing_vector.normalized());
19     }
20
21     for (auto const& pt: points2_rect){
22         opengv::bearingVector_t bearing_vector(pt.x, pt.y, 1); // focal
23         length is 1 after undistortion
24         bearing_vector_2.push_back(bearing_vector.normalized());
25     }

```

Then, call this function in `cameraCallback` with just one line code:

```
calibrateKeypoints(pts1, pts2, bearing_vector_1, bearing_vector_2);
```

After calling `calibrateKeypoints`, `bearing_vector_1` and `bearing_vector_2` will contain the undistorted direction vectors. Subsequently, the line of code

```
Adapter adapter_mono(bearing_vector_1, bearing_vector_2);
```

in the code can correctly feed these data to the **RANSAC** algorithm.

2.2.2 2D-2D Correspondences

In this section, we implemented and evaluated three different algorithms for estimating the relative camera motion between consecutive frames using 2D-2D feature correspondences.

Experimental Methodology

1. **Feature Tracking:** Utilizing the SIFT-based tracker developed in the previous lab to obtain matched keypoints between the previous and current frames.
2. **Keypoint Calibration:** Since the geometric algorithms assume a calibrated pinhole camera model, we implemented `calibrateKeypoints` to undistort the raw pixel coordinates using the provided camera intrinsics (K) and distortion coefficients (D), converting them into normalized bearing vectors.
3. **Motion Estimation:** We utilized the OpenGV library to solve for the relative pose (R, t) using:
 - **Nistér’s 5-point Algorithm:** A minimal solver for the essential matrix.
 - **Longuet-Higgins 8-point Algorithm:** A linear solver for the essential matrix.

- **2-point Algorithm (Known Rotation):** A minimal solver for translation, assuming the relative rotation is known (provided by ground truth in this experiment to simulate a high-precision IMU).
4. **Scale Correction:** As monocular vision inherently suffers from scale ambiguity, we normalized the estimated translation vector and rescaled it using the magnitude of the ground truth translation (`scaleTranslation`) to allow for meaningful visualization and comparison in Rviz.

Implementation Details

The core logic was implemented in the `cameraCallback` function. Before passing data to the solvers, we verified that the feature tracker returned a sufficient number of correspondences. We used `cv::undistortPoints` to map pixel coordinates to the normalized image plane.

For the pose estimation, we employed `opencv::sac::Ransac` to robustly estimate the model in the presence of outliers.

- For the **5-point** and **8-point** methods, we used `CentralRelativePoseSacProblem`.
- For the **2-point** method, we calculated the relative rotation from the ground truth odometry ($R_{\{GT\}} = R_{\{\text{prev}\}}^T R_{\{\text{curr}\}}$) and used `TranslationOnlySacProblem` to solve solely for the translation direction.

(*Note:* We need to add two more parameters to the launch file — `use_ransac` & `pose_estimator:=0` `use_ransac:=<True/False>`, we have:

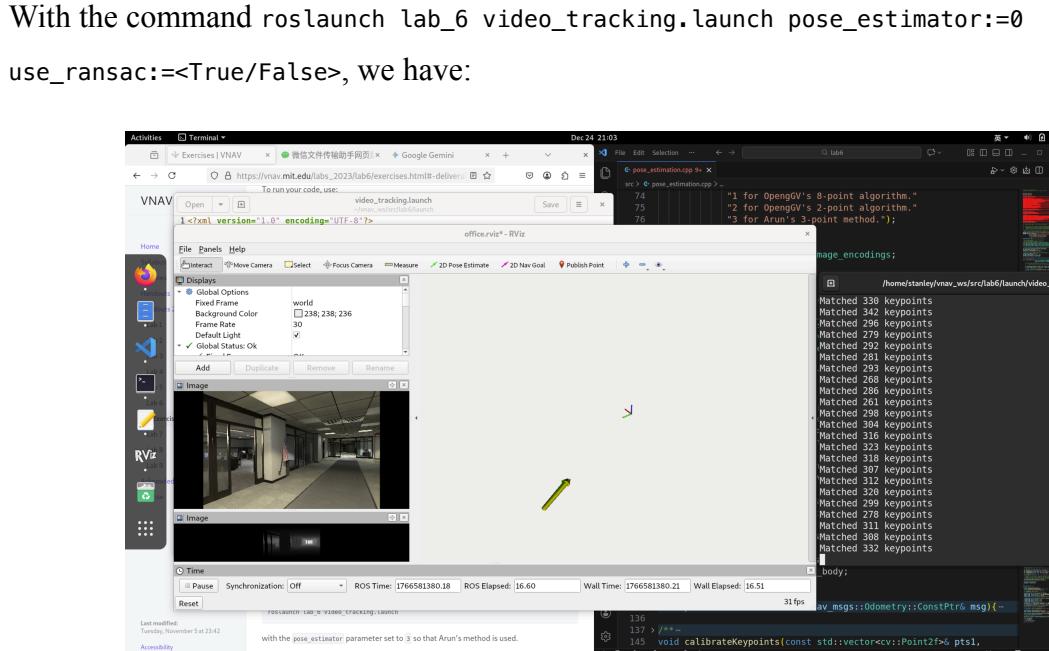


Figure 1: Running Snapshot

Performance Evaluation & Analysis

- Impact of RANSAC on Estimation Accuracy To validate the necessity of outlier rejection in Visual Odometry (VO), we compared the performance of the 5-point algorithm with and without RANSAC. The detailed implementation is [here](#) and the results are visualized in the figure below:

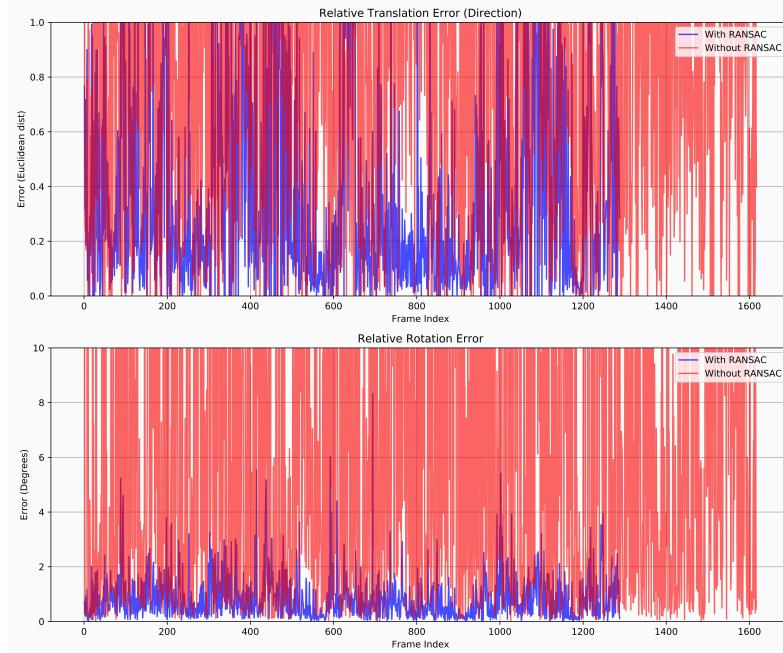


Figure 2: Visualization of the 5-points-based RPE Translation and Rotation Errors Comparison (With vs. Without RANSAC)

- Without RANSAC (Red Line):** The error metrics exhibit significant instability, with frequent spikes in both rotation (exceeding 10 degrees) and translation errors. This confirms that even a small ratio of outliers (mismatched features) can severely corrupt the algebraic solution of the minimal solver.
 - With RANSAC (Blue Line):** The errors are drastically reduced and remain stable over the trajectory. The rotation error is consistently low (< 2 degrees), and the translation error (direction) is bounded, proving that the RANSAC framework effectively filters out spurious matches and selects the best geometric model.
- Comparison of Different Algorithms We further compared the performance of the three geometric algorithms (all using RANSAC). The corresponding python script is [here](#) and the Relative Pose Errors (RPE) for translation and rotation are plotted below:

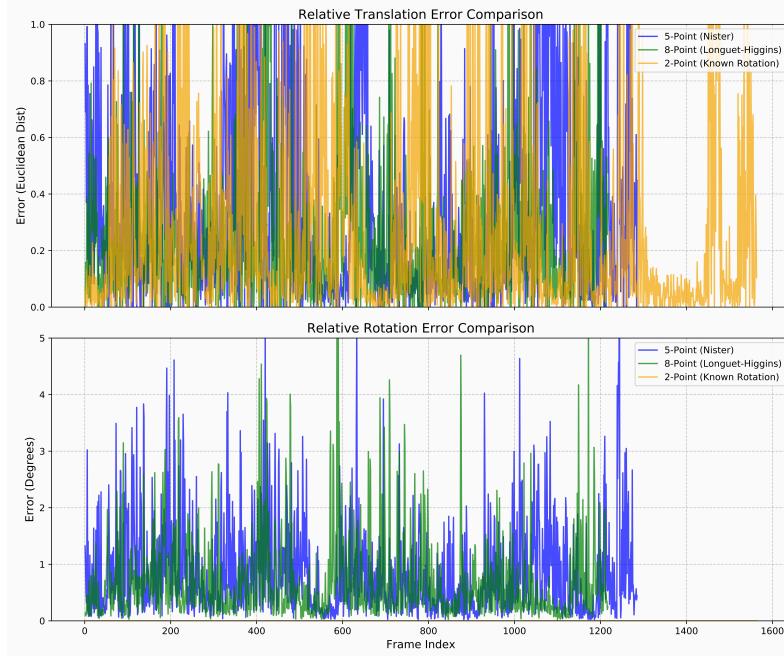


Figure 3: Visualization of the 5-points, 8-points, 2-points Algorithms Translation and Rotation Errors Comparison

1. Rotation Error:

- The **2-point algorithm (Orange)** shows zero rotation error. This is expected and validates our implementation, as we fed the ground truth rotation into the solver.
- The **5-point (Blue)** and **8-point (Green)** algorithms show comparable performance in rotation estimation, generally keeping the error under 3 degrees. The 5-point algorithm appears slightly more stable in certain segments, likely because it enforces the internal constraints of the essential matrix more strictly than the linear 8-point method.

2. Translation Error:

- All three methods exhibit fluctuations in translation error. This is typical for monocular VO, especially when the camera motion is small (low parallax) or the scene structure is degenerate (e.g., planar scenes).
- The **2-point algorithm**, despite having perfect rotation, still shows translation errors. This indicates that translation estimation is highly sensitive to feature noise, even when rotation is known. However, it generally maintains a lower error baseline compared to the 5-point and 8-point methods, demonstrating the benefit of reducing the degrees of freedom when reliable rotation data (e.g., from IMU) is available.

2.2.3 3D-3D Correspondences

This section is to estimate the relative camera motion by aligning two sets of 3D points (3D-3D registration), which allows for the recovery of the trajectory with **absolute scale**, eliminating the need for the ground-truth scaling trick used in Deliverable 4.

Experimental Methodology

We move beyond monocular vision constraints by incorporating depth information provided by the RGB-D sensor.

The workflow proceeds as follows:

1. **3D Point Generation:** For each matched keypoint (u, v) in the RGB image, we queried the corresponding value d from the registered depth image. The 2D keypoints were first converted to normalized bearing vectors (using camera intrinsics) and then scaled by their respective depth values to obtain 3D coordinates in the camera frame: $P_{\{\text{cam}\}} = d \cdot K^{\{-1\}} \cdot [u, v, 1]^T$.
2. **Point Cloud Registration:** We utilized **Arun's Method** (a closed-form solution based on Singular Value Decomposition) to find the rigid body transformation (R, t) that aligns the 3D points from the previous frame ($P_{\{\text{prev}\}}$) to the current frame ($P_{\{\text{curr}\}}$).
3. **Robust Estimation:** To handle noisy depth measurements and mismatches, the algorithm was wrapped in a RANSAC loop provided by OpenGV.

Implementation Details

The implementation was carried out in `cameraCallback` (Case 3).

- **Data Preparation:** We iterated through the matched keypoints, retrieved depth values using `depth.at<float>(y, x)`, and constructed two point clouds (`opengv::points_t`).
- **OpenGV Integration:** We used the `PointCloudAdapter` and `PointCloudSacProblem` classes from OpenGV to interface with the RANSAC solver. The threshold for RANSAC was set in meters (e.g., 0.1m) to reject points with large re-projection errors.
- **Scale Handling:** Unlike the previous 2D-2D experiments, we explicitly disabled the `scaleTranslation` parameter in the launch file. This ensures that the translation output is derived purely from the visual data, verifying the system's ability to recover the true physical scale of the motion.

(*Note:* We need to add one more parameter to the launch file — `scale_translation`.)

With the command `roslaunch lab_6 video_tracking.launch pose_estimator:=3 scale_translation:=0`, we have the running result.

Performance Evaluation & Analysis

We evaluated the accuracy of the 3D-3D estimation by comparing it against the ground truth. The code is [here](#) and the translation and rotation errors are visualized below:

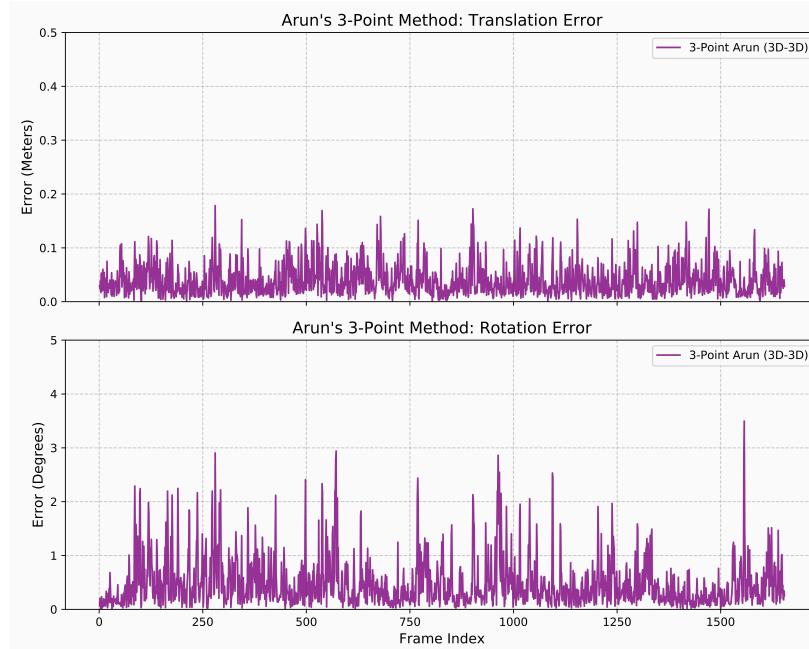


Figure 4: Visualization of the Translation and Rotation Errors of Arun's Algorithm with RANSAC

- **Absolute Scale Recovery:** The most significant observation is the translation error plot (top). Unlike Deliverable 4, where the error was unitless (direction only), here the error is measured in **meters**. The error remains low — typically bounded within $0.05 \sim 0.2$ meters — confirming that Arun's method successfully recovered the absolute scale of the drone's movement using the depth map.
- **Rotation Accuracy:** The rotation error (bottom) is extremely low, consistently staying below 1.0 degree. This indicates that 3D-3D registration provides a very strong constraint on orientation, often outperforming 2D-2D methods which can suffer from rotation-translational ambiguity in certain motion configurations.
- **Stability:** The trajectory estimation is robust and does not exhibit the scale drift issues common in monocular VO systems, demonstrating the advantage of RGB-D sensors for indoor navigation.

3 Reflection and Analysis

4 Conclusion

5 Source Code

- *pose_estimation.cpp*

```
1 #include <stdio.h>
2 #include <iostream>
3 #include <fstream>
4 #include <memory>
5
6 #include <gflags/gflags.h>
7 #include <glog/logging.h>
8
9 #include <ros/ros.h>
10 #include <message_filters/subscriber.h>
11 #include <message_filters/synchronizer.h>
12 #include <message_filters/sync_policies/approximate_time.h>
13 #include <image_transport/image_transport.h>
14 #include <image_transport/subscriber_filter.h>
15 #include <sensor_msgs/image_encodings.h>
16 #include <cv_bridge/cv_bridge.h>
17 #include <sensor_msgs/Image.h>
18 #include <sensor_msgs/CameraInfo.h>
19 #include <geometry_msgs/PoseStamped.h>
20 #include <tf/tf.h>
21 #include <tf/transform_datatypes.h>
22 #include <tf/conversions/tf_eigen.h>
23 #include <tf/transform_broadcaster.h>
24 #include <nav_msgs/Odometry.h>
25
26 #include <opencv2/imgcodecs.hpp>
27 #include <opencv2/highgui/highgui.hpp>
28
29 #include <Eigen/Eigen>
30
31 #include <opencv2/core.hpp>
32 #include <opencv2/calib3d.hpp>
33 #include <opencv2/core/eigen.hpp>
34
35 // OpenGV
36 #include <opengv/sac/Ransac.hpp>
37 #include <opengv/sac_problems/relative_pose/TranslationOnlySacProblem.hpp>
38 #include <opengv/sac_problems/relative_pose/
39 CentralRelativePoseSacProblem.hpp>
40 #include <opengv/sac_problems/point_cloud/PointCloudSacProblem.hpp>
41 #include <opengv/relative_pose/CentralRelativeAdapter.hpp>
42 #include <opengv/relative_pose/methods.hpp>
```

```

42 #include <opengv/point_cloud/PointCloudAdapter.hpp>
43 #include <opengv/point_cloud/methods.hpp>
44
45 #include "lab6_utils.h"
46 #include "pose_estimation.h"
47 #include "tracker_shim.h"
48
49 DEFINE_bool(use_ransac, true, "Use Random Sample Consensus.");
50 DEFINE_bool(scale_translation, true, "Whether to scale estimated translation
51 to match ground truth scale");
52 DEFINE_int32(pose_estimator, 0,
53             "Pose estimation algorithm, valid values are:"
54             "0 for OpengV's 5-point algorithm."
55             "1 for OpengV's 8-point algorithm."
56             "2 for OpengV's 2-point algorithm."
57             "3 for Arun's 3-point method.");
58
59 using namespace std;
60 namespace enc = sensor_msgs::image_encodings;
61
62 using RansacProblem =
63 opengv::sac_problems::relative_pose::CentralRelativePoseSacProblem;
64 using Adapter = opengv::relative_pose::CentralRelativeAdapter;
65 using RansacProblemGivenRot =
66 opengv::sac_problems::relative_pose::TranslationOnlySacProblem;
67 using AdapterGivenRot = opengv::relative_pose::CentralRelativeAdapter;
68 using Adapter3D = opengv::point_cloud::PointCloudAdapter;
69 using RansacProblem3D =
70 opengv::sac_problems::point_cloud::PointCloudSacProblem;
71
72 std::unique_ptr<TrackerWrapper> feature_tracker_;
73 ros::Publisher pub_pose_estimation_, pub_pose_gt_;
74 ros::Subscriber sub_cinfo_;
75 geometry_msgs::PoseStamped curr_pose_;
76 geometry_msgs::PoseStamped prev_pose_;
77
78 CameraParams camera_params_;
79 cv::Mat R_camera_body, t_camera_body;
80 cv::Mat T_camera_body;
81 geometry_msgs::Pose pose_camera_body;
82 tf::Transform transform_camera_body;
83
84 void poseCallbackTesse(const nav_msgs::Odometry::ConstPtr& msg){
85     curr_pose_.pose = msg->pose.pose;
86
87     tf::Transform current_pose;

```

```

85     tf::poseMsgToTF(curr_pose_.pose, current_pose);
86
87     tf::poseTFToMsg(current_pose * transform_camera_body, curr_pose_.pose);
88
89     curr_pose_.header.frame_id = "world";
90     pub_pose_gt_.publish(curr_pose_);
91 }
92
93 /**
94  * @brief      Compute 3D bearing vectors from pixel points
95  *
96  * @param[in]  pts1          Feature correspondences from camera 1
97  * @param[in]  pts2          Feature correspondences from camera 2
98  * @param      bearing_vector_1 Bearing vector to pts1 in camera 1
99  * @param      bearing_vector_2 Bearing vector to pts2 in camera 2
100 */
101 void calibrateKeypoints(const std::vector<cv::Point2f>& pts1,
102                         const std::vector<cv::Point2f>& pts2,
103                         opencv::bearingVectors_t& bearing_vector_1,
104                         opencv::bearingVectors_t& bearing_vector_2) {
105     std::vector<cv::Point2f> points1_rect, points2_rect;
106     cv::undistortPoints(pts1, points1_rect, camera_params_.K,
107                         camera_params_.D);
108     cv::undistortPoints(pts2, points2_rect, camera_params_.K,
109                         camera_params_.D);
110
111     for (auto const& pt: points1_rect){
112         opencv::bearingVector_t bearing_vector(pt.x, pt.y, 1); // focal length
113         is 1 after undistortion
114         bearing_vector_1.push_back(bearing_vector.normalized());
115     }
116
117     for (auto const& pt: points2_rect){
118         opencv::bearingVector_t bearing_vector(pt.x, pt.y, 1); // focal length
119         is 1 after undistortion
120         bearing_vector_2.push_back(bearing_vector.normalized());
121     }
122 }
123
124 /**
125  * @brief      Update pose estimate using previous absolute pose and estimated
126  *             relative pose
127  *
128  * @param[in]  prev_pose      ground-truth absolute pose of previous frame
129  * @param[in]  relative_pose  estimated relative pose between current
130  *                           frame and previous frame
131  * @param      output        estimated absolute pose of current frame

```

```

126  /*
127   void updatePoseEstimate(geometry_msgs::Pose const& prev_pose,
128   geometry_msgs::Pose const& relative_pose, geometry_msgs::Pose& output) {
129     tf::Transform prev, relative;
130     tf::poseMsgToTF(prev_pose, prev);
131     tf::poseMsgToTF(relative_pose, relative);
132     tf::poseTFToMsg(prev*relative, output);
133   }
134 /**
135 * @brief      Given an estimated translation up to scale, return an absolute
136 *             translation with correct scale using ground truth
137 *
138 * @param[in]  prev_pose          ground-truth absolute pose of previous frame
139 * @param[in]  curr_pose          ground-truth absolute pose of current frame
140 * @param      translation        estimated translation between current frame
141 *             and previous frame
142 */
143 void scaleTranslation(geometry_msgs::Point& translation,
144 geometry_msgs::PoseStamped const& prev_pose, geometry_msgs::PoseStamped
145 const& curr_pose) {
146   if (!FLAGS_scale_translation) return;
147   tf::Transform prev, curr;
148   tf::poseMsgToTF(prev_pose.pose, prev);
149   tf::poseMsgToTF(curr_pose.pose, curr);
150   tf::Transform const relative_pose = prev.inverseTimes(curr);
151   double const translation_scale = relative_pose.getOrigin().length();
152   if (isnan(translation_scale) || isinf(translation_scale)) {
153     ROS_WARN("Failed to scale translation");
154     return;
155   }
156   double const old_scale = sqrt(pow(translation.x, 2) + pow(translation.y, 2)
157   + pow(translation.z, 2));
158   translation.x *= translation_scale / old_scale;
159   translation.y *= translation_scale / old_scale;
160   translation.z *= translation_scale / old_scale;
161
162 /**
163 * @brief      (TODO) Compute Relative Pose Error (RPE) in translation and
164 *             rotation.
165 * @param[in]  gt_t_prev_frame  ground-truth transform for previous frame.
166 * @param[in]  gt_t_curr_frame  ground-truth transform for current frame.
167 * @param[in]  est_t_prev_frame estimated transform for previous frame.
168 * @param[in]  est_t_curr_frame estimated transform for current frame.
169 */
170 void evaluateRPE(const tf::Transform& gt_t_prev_frame,

```

```

166             const tf::Transform& gt_t_curr_frame,
167             const tf::Transform& est_t_prev_frame,
168             const tf::Transform& est_t_curr_frame) {
169     tf::Transform const est_relative_pose =
170         est_t_prev_frame.inverseTimes(est_t_curr_frame);
171     tf::Transform const gt_relative_pose =
172         gt_t_prev_frame.inverseTimes(gt_t_curr_frame);
173
174     tf::Vector3 t_est = est_relative_pose.getOrigin();
175     tf::Vector3 t_gt = gt_relative_pose.getOrigin();
176
177     if (FLAGS_pose_estimator < 3) {
178         if (t_est.length() > 1e-6) t_est.normalize();
179         if (t_gt.length() > 1e-6) t_gt.normalize();
180     }
181
182     double translation_error = t_gt.distance(t_est);
183
184     tf::Quaternion q_est = est_relative_pose.getRotation();
185     tf::Quaternion q_gt = gt_relative_pose.getRotation();
186
187     tf::Quaternion q_diff = q_gt.inverse() * q_est;
188     double rotation_error_rad = q_diff.getAngle();
189
190     double rotation_error_deg = rotation_error_rad * 180.0 / M_PI;
191
192     static std::ofstream log_file;
193
194     if (!log_file.is_open()) {
195         std::string output_path = "/home/stanley/vnav_ws/src/lab6/log/";
196         std::string filename;
197
198         if (!FLAGS_use_ransac) {
199             filename = output_path + "rpe_with_no_ransac.csv";
200         } else {
201             switch(FLAGS_pose_estimator) {
202                 case 0: filename = output_path + "rpe_5pt.csv"; break;
203                 case 1: filename = output_path + "rpe_8pt.csv"; break;
204                 case 2: filename = output_path + "rpe_2pt.csv"; break;
205                 case 3: filename = output_path + "rpe_3pt.csv"; break;
206                 default: filename = output_path + "rpe_unknown.csv"; break;
207             }
208         }
209
210         log_file.open(filename, std::ios::out | std::ios::trunc);

```

```

210     if (log_file.is_open()) {
211         log_file << "frame,trans_error,rot_error_deg" << std::endl;
212         std::cout << "[Lab6] Saving RPE logs to: " << filename <<
213         std::endl;
214     } else {
215         std::cerr << "[Lab6] Failed to open file: " << filename <<
216         std::endl;
217     }
218     static int frame_count = 0;
219     if (log_file.is_open()) {
220         log_file << frame_count << "," << translation_error << "," <<
221         rotation_error_deg << std::endl;
222     }
223     frame_count++;
224 }
225 /**
226 * @brief (TODO) This function is called when a new image is published. This
227 * is
228 * where all the magic happens for this lab
229 * @param[in] rgb_msg    RGB Camera message
230 * @param[in] depth_msg Depth Camera message
231 */
232 void cameraCallback(const sensor_msgs::ImageConstPtr &rgb_msg, const
233 sensor_msgs::ImageConstPtr &depth_msg) {
234     cv::Mat bgr, depth;
235
236     try {
237         bgr = cv_bridge::toCvShare(rgb_msg, "bgr8")->image;
238         depth = cv_bridge::toCvShare(depth_msg, depth_msg->encoding)->image;
239     } catch (cv_bridge::Exception &e) {
240         ROS_ERROR("Could not convert rgb or depth images.");
241     }
242
243     cv::Mat view = bgr.clone();
244
245     static cv::Mat prev_bgr = bgr.clone();
246     static cv::Mat prev_depth = depth.clone();
247
248     std::pair<std::vector<cv::KeyPoint>, std::vector<cv::KeyPoint>>
249     matched_kp_1_kp_2;
250     feature_tracker_->track(prev_bgr, bgr, &matched_kp_1_kp_2);
251
252     int N = matched_kp_1_kp_2.first.size();
253     std::cout << "Matched " << N << " keypoints" << std::endl;
254     std::vector<cv::Point2f> pts1, pts2;

```

```

251
252     cv::KeyPoint::convert(matched_kp_1_kp_2.first, pts1);
253     cv::KeyPoint::convert(matched_kp_1_kp_2.second, pts2);
254
255     opencv::bearingVectors_t bearing_vector_1, bearing_vector_2;
256
257     if (!pts1.empty()) {
258         calibrateKeypoints(pts1, pts2, bearing_vector_1, bearing_vector_2);
259     }
260
261     Adapter adapter_mono (bearing_vector_1, bearing_vector_2);
262
263     geometry_msgs::PoseStamped pose_estimation;
264     tf::poseTFToMsg(tf::Pose::getIdentity(), pose_estimation.pose);
265
266     geometry_msgs::Pose relative_pose_estimate = pose_estimation.pose;
267
268     switch(FLAGS_pose_estimator) {
269     case 0: {
270         // 5-points Algorithm
271         static constexpr size_t min_nr_of_correspondences = 5;
272         if (adapter_mono.getNumberCorrespondences() >= min_nr_of_correspondences)
273         {
274             if (!FLAGS_use_ransac) {
275                 std::shared_ptr<RansacProblem> ransac_problem =
276                     std::make_shared<RansacProblem>(adapter_mono, RansacProblem::NISTER);
277                 opencv::sac::Ransac<RansacProblem> ransac;
278                 ransac.sac_model_ = ransac_problem;
279
280                 ransac.max_iterations_ = 1;
281                 ransac.threshold_ = 100000.0;
282
283                 if(ransac.computeModel())
284                     relative_pose_estimate = eigen2Pose(ransac.model_coefficients_);
285                 } else {
286                     ROS_WARN("5-point non-RANSAC failed");
287                 }
288             } else {
289                 std::shared_ptr<RansacProblem> ransac_problem =
290                     std::make_shared<RansacProblem>(adapter_mono, RansacProblem::NISTER);
291
292                 opencv::sac::Ransac<RansacProblem> ransac;
293                 ransac.sac_model_ = ransac_problem;
294
295                 ransac.threshold_ = 0.00001;
296                 ransac.max_iterations_ = 100;
297             }
298         }
299     }
300 }
```

```

295         ransac.computeModel();
296
297         relative_pose_estimate = eigen2Pose(ransac.model_coefficients_);
298     }
299 } else {
300     ROS_WARN("Not enough correspondences to compute pose estimation using"
301             " Nister's algorithm.");
302 }
303 break;
304 }
305 case 1: {
306     // 8-points Algorithm
307     static constexpr size_t min_nr_of_correspondences = 8;
308
309     if (adapter_mono.getNumberCorrespondences() >= min_nr_of_correspondences)
310     {
311         if (!FLAGS_use_ransac) {
312             std::shared_ptr<RansacProblem> ransac_problem =
313             std::make_shared<RansacProblem>(adapter_mono,
314             RansacProblem::EIGHTPT);
315             opencv::sac::Ransac<RansacProblem> ransac;
316             ransac.sac_model_ = ransac_problem;
317
318             ransac.max_iterations_ = 1;
319             ransac.threshold_ = 100000.0;
320
321             if(ransac.computeModel()) {
322                 relative_pose_estimate = eigen2Pose(ransac.model_coefficients_);
323             }
324         } else {
325             std::shared_ptr<RansacProblem> ransac_problem =
326             std::make_shared<RansacProblem>(adapter_mono,
327             RansacProblem::EIGHTPT);
328
329             opencv::sac::Ransac<RansacProblem> ransac;
330             ransac.sac_model_ = ransac_problem;
331
332             ransac.threshold_ = 0.00001;
333             ransac.max_iterations_ = 100;
334
335             ransac.computeModel();
336
337             relative_pose_estimate = eigen2Pose(ransac.model_coefficients_);
338         }
339     } else {
340         ROS_WARN("Not enough correspondences to compute pose estimation using"
341                 " Longuet-Higgins' algorithm.");

```

```

337     }
338     break;
339 }
340 case 2: {
341     // 2-point Algorithm
342     static constexpr size_t min_nr_of_correspondences = 2;
343     if (adapter_mono.getNumberCorrespondences() >= min_nr_of_correspondences)
344     {
345         tf::Transform curr_frame, prev_frame;
346         tf::poseMsgToTF(curr_pose_.pose, curr_frame);
347         tf::poseMsgToTF(prev_pose_.pose, prev_frame);
348         Eigen::Matrix3d rotation;
349         tf::matrixTFToEigen(prev_frame.inverseTimes(curr_frame).getBasis(),
350                             rotation);
351         adapter_mono.setR12(rotation);
352
353         if (!FLAGS_use_ransac) {
354             std::shared_ptr<RansacProblemGivenRot> ransac_problem =
355             std::make_shared<RansacProblemGivenRot>(adapter_mono);
356             opencv::sac::Ransac<RansacProblemGivenRot> ransac;
357             ransac.sac_model_ = ransac_problem;
358
359             ransac.max_iterations_ = 1;
360             ransac.threshold_ = 100000.0;
361
362             if(ransac.computeModel()) {
363                 relative_pose_estimate = eigen2Pose(ransac.model_coefficients_);
364             }
365             } else {
366                 std::shared_ptr<RansacProblemGivenRot> ransac_problem =
367                 std::make_shared<RansacProblemGivenRot>(adapter_mono);
368
369                 opencv::sac::Ransac<RansacProblemGivenRot> ransac;
370                 ransac.sac_model_ = ransac_problem;
371
372                 ransac.threshold_ = 0.00001;
373                 ransac.max_iterations_ = 100;
374
375                 ransac.computeModel();
376
377                 relative_pose_estimate = eigen2Pose(ransac.model_coefficients_);
378
379                 // **** end solution
380                 ****
381             }
382         } else {

```

```

378     ROS_WARN("Not enough correspondences to estimate relative translation
379     using 2pt algorithm.");
380 }
381 }
382 case 3: {
383     // Arun's 3-point algorithm
384     for (int i=0; i<N; i++) {
385         double d1 = double( prev_depth.at<float>(std::floor(pts1[i].y),
386                             std::floor(pts1[i].x)) );
387         double d2 = double( depth.at<float>(std::floor(pts2[i].y),
388                             std::floor(pts2[i].x)) );
389
390         bearing_vector_1[i] /= bearing_vector_1[i](2,0);
391         bearing_vector_2[i] /= bearing_vector_2[i](2,0);
392
393     }
394
395     opencv::points_t cloud_1, cloud_2;
396     for (auto i = 0ul; i < bearing_vector_1.size(); i++) {
397         cloud_1.push_back(bearing_vector_1[i]);
398         cloud_2.push_back(bearing_vector_2[i]);
399     }
400
401     Adapter3D adapter_3d(cloud_1, cloud_2);
402
403     static constexpr int min_nr_of_correspondences = 3;
404     if (adapter_3d.getNumberCorrespondences() >= min_nr_of_correspondences) {
405         if (!FLAGS_use_ransac){
406             std::shared_ptr<RansacProblem3D> ransac_problem =
407                 std::make_shared<RansacProblem3D>(adapter_3d);
408
409             opencv::sac::Ransac<RansacProblem3D> ransac;
410             ransac.sac_model_ = ransac_problem;
411
412             ransac.max_iterations_ = 1;
413             ransac.threshold_ = 1000.0;
414
415             if(ransac.computeModel()) {
416                 relative_pose_estimate = eigen2Pose(ransac.model_coefficients_);
417             }
418         } else{
419             std::shared_ptr<RansacProblem3D> ransac_problem =
420                 std::make_shared<RansacProblem3D>(adapter_3d);
421
422             if(ransac_problem->computeModel()) {
423                 relative_pose_estimate = eigen2Pose(ransac_problem->model_coefficients_);
424             }
425         }
426     }
427 }

```

```

420     opencv::sac::Ransac<RansacProblem3D> ransac;
421     ransac.sac_model_ = ransac_problem;
422
423     ransac.threshold_ = 0.1;
424     ransac.max_iterations_ = 100;
425
426     if(ransac.computeModel()) {
427         relative_pose_estimate = eigen2Pose(ransac.model_coefficients_);
428     }
429     }
430 } else {
431     ROS_WARN("Not enough correspondences to estimate absolute transform
432         using Arun's 3pt algorithm.");
433     }
434     break;
435 }
436 ROS_ERROR("Wrong pose_estimator flag!");
437 }
438 }
439 if (FLAGS_pose_estimator < 3) {
440     scaleTranslation(relative_pose_estimate.position, prev_pose_,
441     curr_pose_);
442 }
443 updatePoseEstimate(prev_pose_.pose, relative_pose_estimate,
444 pose_estimation.pose);
445 tf::Transform gt_t_prev_frame, gt_t_curr_frame;
446 tf::Transform est_t_prev_frame, est_t_curr_frame;
447 tf::poseMsgToTF(pose_estimation.pose, est_t_curr_frame);
448 tf::poseMsgToTF(curr_pose_.pose, gt_t_curr_frame);
449 tf::poseMsgToTF(prev_pose_.pose, est_t_prev_frame);
450 tf::poseMsgToTF(prev_pose_.pose, gt_t_prev_frame);
451
452 evaluateRPE(gt_t_prev_frame,
453     gt_t_curr_frame,est_t_prev_frame,est_t_curr_frame);
454 pose_estimation.header.frame_id = "world";
455 pub_pose_estimation_.publish(pose_estimation);
456
457 prev_bgr = bgr.clone();
458 prev_depth = depth.clone();
459 prev_pose_ = curr_pose_;
460 }
461
462 int main(int argc, char** argv) {

```

```

463     google::InitGoogleLogging(argv[0]);
464     google::ParseCommandLineFlags(&argc, &argv, true);
465     google::InstallFailureSignalHandler();
466
467     ros::init(argc, argv, "keypoint_trackers");
468     ros::NodeHandle local_nh("~");
469
470     camera_params_.K = cv::Mat::zeros(3, 3, CV_64F);
471     camera_params_.K.at<double>(0,0) = 415.69219381653056;
472     camera_params_.K.at<double>(1,1) = 415.69219381653056;
473     camera_params_.K.at<double>(0,2) = 360.0;
474     camera_params_.K.at<double>(1,2) = 240.0;
475     camera_params_.D = cv::Mat::zeros(cv::Size(5,1),CV_64F);
476
477     T_camera_body = cv::Mat::zeros(cv::Size(4,4),CV_64F);
478     T_camera_body.at<double>(0,2) = 1.0;
479     T_camera_body.at<double>(1,0) = -1.0;
480     T_camera_body.at<double>(1,3) = 0.05;
481     T_camera_body.at<double>(2,1) = -1.0;
482     T_camera_body.at<double>(3,3) = 1.0;
483
484     R_camera_body = T_camera_body(cv::Range(0,3),cv::Range(0,3));
485     t_camera_body = T_camera_body(cv::Range(0,3),cv::Range(3,4));
486     pose_camera_body = cv2Pose(R_camera_body,t_camera_body);
487
488     tf::poseMsgToTF(pose_camera_body, transform_camera_body);
489
490     feature_tracker_.reset(new TrackerWrapper());
491
492     auto pose_sub = local_nh.subscribe("/ground_truth_pose", 10,
493                                         poseCallbackTesse);
494
495     image_transport::ImageTransport it(local_nh);
496     image_transport::SubscriberFilter sf_rgb(it, "/rgb_images_topic", 1);
497     image_transport::SubscriberFilter sf_depth(it, "/depth_images_topic", 1);
498
499     typedef message_filters::sync_policies::ApproximateTime<sensor_msgs::Image,
500     sensor_msgs::Image> MySyncPolicy;
501     message_filters::Synchronizer<MySyncPolicy> sync(MySyncPolicy(10), sf_rgb,
502     sf_depth);
503     sync.registerCallback(cameraCallback);
504
505     pub_pose_gt_ = local_nh.advertise<geometry_msgs::PoseStamped>("/gt_camera_pose", 1);
506     pub_pose_estimation_ = local_nh.advertise<geometry_msgs::PoseStamped>("/camera_pose", 1);
507

```

```

505     while (ros::ok()) {
506         ros::spinOnce();
507         cv::waitKey(1);
508     }
509     cv::destroyAllWindows();
510
511     return EXIT_SUCCESS;
512 }
```

- *rpe_comparison.py*

```

1  import matplotlib.pyplot as plt
2  import pandas as pd
3  import os
4
5  path_ransac = '/home/stanley/vnav_ws/src/lab6/log/rpe_with_ransac.csv'
6  path_no_ransac = '/home/stanley/vnav_ws/src/lab6/log/rpe_with_no_ransac.csv'
7
8  df_ransac = pd.read_csv(path_ransac) if os.path.exists(path_ransac) else None
9  df_no_ransac = pd.read_csv(path_no_ransac) if os.path.exists(path_no_ransac)
else None
10
11 fig, axs = plt.subplots(2, 1, figsize=(12, 10))
12
13 # --- 1. 绘制平移误差 ---
14 if df_ransac is not None:
15     axs[0].plot(df_ransac['frame'].to_numpy(),
16                  df_ransac['trans_error'].to_numpy(), label='With RANSAC', color='blue',
17                  alpha=0.7)
18
19 if df_no_ransac is not None:
20     axs[0].plot(df_no_ransac['frame'].to_numpy(),
21                  df_no_ransac['trans_error'].to_numpy(), label='Without RANSAC',
22                  color='red', alpha=0.6)
23
24 axs[0].set_title('Relative Translation Error (Direction)')
25 axs[0].set_ylabel('Error (Euclidean dist)')
26 axs[0].set_xlabel('Frame Index')
27 axs[0].legend(
28     loc='upper right',
29     bbox_to_anchor=(1.0, 1.0)
30 )
31 axs[0].grid(True)
32 axs[0].set_ylim(0, 1.0)
33
34 # --- 2. 绘制旋转误差 ---
35 if df_ransac is not None:
36     axs[1].plot(df_ransac['frame'].to_numpy(),
37                  df_ransac['rot_error'].to_numpy(), label='With RANSAC', color='blue',
38                  alpha=0.7)
39
40 if df_no_ransac is not None:
41     axs[1].plot(df_no_ransac['frame'].to_numpy(),
42                  df_no_ransac['rot_error'].to_numpy(), label='Without RANSAC',
43                  color='red', alpha=0.6)
44
45 axs[1].set_title('Relative Rotation Error (Degree)')
46 axs[1].set_ylabel('Error (Degree)')
47 axs[1].set_xlabel('Frame Index')
48 axs[1].legend(
49     loc='upper right',
50     bbox_to_anchor=(1.0, 1.0)
51 )
52 axs[1].grid(True)
53
54 # --- 3. 绘制总误差 ---
55 total_error = df_ransac['trans_error'].to_numpy() + df_ransac['rot_error'].to_numpy()
56 total_no_error = df_no_ransac['trans_error'].to_numpy() + df_no_ransac['rot_error'].to_numpy()
57
58 if df_ransac is not None:
59     axs[2].plot(df_ransac['frame'].to_numpy(),
60                  total_error, label='With RANSAC', color='blue',
61                  alpha=0.7)
62
63 if df_no_ransac is not None:
64     axs[2].plot(df_no_ransac['frame'].to_numpy(),
65                  total_no_error, label='Without RANSAC',
66                  color='red', alpha=0.6)
67
68 axs[2].set_title('Total Error')
69 axs[2].set_ylabel('Error')
70 axs[2].set_xlabel('Frame Index')
71
72 # --- 4. 绘制帧数 ---
73 frame_nums = np.arange(1, len(df_ransac['frame']) + 1)
74
75 if df_ransac is not None:
76     axs[3].plot(frame_nums, df_ransac['frame'], label='With RANSAC', color='blue',
77                 alpha=0.7)
78
79 if df_no_ransac is not None:
80     axs[3].plot(frame_nums, df_no_ransac['frame'], label='Without RANSAC',
81                 color='red', alpha=0.6)
82
83 axs[3].set_title('Frame Index')
84 axs[3].set_ylabel('Frame Index')
85 axs[3].set_xlabel('Frame Index')
86
87 # --- 5. 绘制帧数 ---
88 frame_nums = np.arange(1, len(df_ransac['frame']) + 1)
89
90 if df_ransac is not None:
91     axs[4].plot(frame_nums, df_ransac['frame'], label='With RANSAC', color='blue',
92                 alpha=0.7)
93
94 if df_no_ransac is not None:
95     axs[4].plot(frame_nums, df_no_ransac['frame'], label='Without RANSAC',
96                 color='red', alpha=0.6)
97
98 axs[4].set_title('Frame Index')
99 axs[4].set_ylabel('Frame Index')
100 axs[4].set_xlabel('Frame Index')
```

```

30 # --- 2. 绘制旋转误差 ---
31 if df_ransac is not None:
32     axs[1].plot(df_ransac['frame'].to_numpy(),
33                  df_ransac['rot_error_deg'].to_numpy(), label='With RANSAC', color='blue',
34                  alpha=0.7)
35
36 if df_no_ransac is not None:
37     axs[1].plot(df_no_ransac['frame'].to_numpy(),
38                  df_no_ransac['rot_error_deg'].to_numpy(), label='Without RANSAC',
39                  color='red', alpha=0.6)
40
41 axs[1].set_title('Relative Rotation Error')
42 axs[1].set_ylabel('Error (Degrees)')
43 axs[1].set_xlabel('Frame Index')
44 axs[1].legend(
45     loc='upper right',
46     bbox_to_anchor=(1.0, 1.0)
47 )
48 plt.tight_layout()
49 plt.savefig('rpe_comparison.png', dpi=1200)
50 plt.show()

```

- *algorithm_comparison.py*

```

1 import matplotlib.pyplot as plt
2 import pandas as pd
3 import os
4 import numpy as np
5
6 base_path = '/home/stanley/vnav_ws/src/lab6/log/'
7 files = {
8     '5-Point (Nister)': os.path.join(base_path, 'rpe_5pt.csv'),
9     '8-Point (Longuet-Higgins)': os.path.join(base_path, 'rpe_8pt.csv'),
10    '2-Point (Known Rotation)': os.path.join(base_path, 'rpe_2pt.csv')
11 }
12
13 colors = {
14     '5-Point (Nister)': 'blue',
15     '8-Point (Longuet-Higgins)': 'green',
16     '2-Point (Known Rotation)': 'orange'
17 }
18

```

```

19 data_frames = {}
20 for name, path in files.items():
21     if os.path.exists(path):
22         print(f"Loading {name} from {path}...")
23         data_frames[name] = pd.read_csv(path)
24     else:
25         print(f"[WARNING] File not found: {path}")
26
27 if not data_frames:
28     print("No data found! Please check file paths.")
29     exit()
30
31 fig, axs = plt.subplots(2, 1, figsize=(12, 10), sharex=True)
32
33 # --- 1. 绘制平移误差 (Translation Error) ---
34 for name, df in data_frames.items():
35     frames = df['frame'].to_numpy()
36     trans_err = df['trans_error'].to_numpy()
37
38     axs[0].plot(frames, trans_err, label=name, color=colors[name], alpha=0.7,
39                  linewidth=1.5)
40
41 axs[0].set_title('Relative Translation Error Comparison', fontsize=14)
42 axs[0].set_ylabel('Error (Euclidean Dist)', fontsize=12)
43 axs[0].grid(True, which='both', linestyle='--', alpha=0.7)
44 axs[0].legend(
45     loc='upper right',
46     bbox_to_anchor=(1.0, 1.0)
47 )
48
49 # --- 2. 绘制旋转误差 (Rotation Error) ---
50 for name, df in data_frames.items():
51     frames = df['frame'].to_numpy()
52     rot_err = df['rot_error_deg'].to_numpy()
53
54     axs[1].plot(frames, rot_err, label=name, color=colors[name], alpha=0.7,
55                  linewidth=1.5)
56
57 axs[1].set_title('Relative Rotation Error Comparison', fontsize=14)
58 axs[1].set_ylabel('Error (Degrees)', fontsize=12)
59 axs[1].set_xlabel('Frame Index', fontsize=12)
60 axs[1].grid(True, which='both', linestyle='--', alpha=0.7)
61 axs[1].legend(
62     loc='upper right',
63     bbox_to_anchor=(1.0, 1.0)

```

```

63 )
64 axs[1].set_ylim(0, 5.0)
65
66 plt.tight_layout()
67
68 output_file = 'algorithm_comparison.png'
69 plt.savefig(output_file, dpi=1200)
70 plt.show()

```

- *arun_3d.py*

```

1 import matplotlib.pyplot as plt
2 import pandas as pd
3 import os
4 import numpy as np
5
6 file_path = '/home/stanley/vnav_ws/src/lab6/log/rpe_3pt.csv'
7
8 if os.path.exists(file_path):
9     print(f"Loading 3D-3D results from {file_path}...")
10    df = pd.read_csv(file_path)
11 else:
12    print(f"[ERROR] File not found: {file_path}")
13    print("Please make sure you ran the simulation with pose_estimator=3")
14    exit()
15
16 fig, axs = plt.subplots(2, 1, figsize=(10, 8), sharex=True)
17
18 frames = df['frame'].to_numpy()
19 trans_err = df['trans_error'].to_numpy()
20 rot_err = df['rot_error_deg'].to_numpy()
21
22 # --- 1. 绘制平移误差 (Translation Error) ---
23 axs[0].plot(frames, trans_err, label='3-Point Arun (3D-3D)', color='purple',
24               alpha=0.8, linewidth=1.5)
25
26 axs[0].set_title('Arun\'s 3-Point Method: Translation Error', fontsize=14)
27 axs[0].set_ylabel('Error (Meters)', fontsize=12)
28
29 axs[0].grid(True, which='both', linestyle='--', alpha=0.7)
30 axs[0].legend(
31     loc='upper right',
32     bbox_to_anchor=(1.0, 1.0)
33 )

```

```
34 axs[0].set_ylim(0, 0.5)
35
36 # --- 2. 绘制旋转误差 (Rotation Error) ---
37 axs[1].plot(frames, rot_err, label='3-Point Arun (3D-3D)', color='purple',
38               alpha=0.8, linewidth=1.5)
39
40 axs[1].set_title('Arun\'s 3-Point Method: Rotation Error', fontsize=14)
41 axs[1].set_ylabel('Error (Degrees)', fontsize=12)
42 axs[1].set_xlabel('Frame Index', fontsize=12)
43 axs[1].grid(True, which='both', linestyle='--', alpha=0.7)
44 axs[1].legend(
45     loc='upper right',
46     bbox_to_anchor=(1.0, 1.0)
47 )
48
49 plt.tight_layout()
50 plt.savefig('arun_3d.png', dpi=1200)
51 plt.show()
```