Lab 6 Report: EXERCISES

Yilin Zhang [1]. Lab Group: 31.

I. Individual

Deliverable 1 - Nister's 5-point Algorithm

Read the paper [1] and answer the questions below

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

Sol.

Construct a 5×9 matrix from the five point correspondences using the epipolar constraint and compute four vectors that span its nullspace (e.g., via QR factorisation).

Express the essential matrix as a linear combination of four basis matrices derived from the nullspace.

Substitute this expression into the cubic constraints that characterize an essential matrix, leading to a system of nine equations.

Perform Gauss-Jordan elimination to reduce the system, then form two 4×4 polynomial matrices and compute their determinants to obtain a tenth-degree polynomial.

Extract the real roots of this polynomial (e.g., using Sturm sequences or a companion matrix).

For each real root, recover the essential matrix, decompose it via singular value decomposition to obtain the rotation matrix R and the translation vector t (up to scale), and resolve the four-fold ambiguity using the cheirality constraint with triangulation of one point.

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)

Sol.

In the calibrated setting, the algorithm does not fail completely under special configurations. For two views of a planar scene, there exists at most a two-fold ambiguity in the solution for the essential matrix, which is generally resolved by incorporating a third view. The algorithm continues to operate correctly even with coplanar points, unlike uncalibrated methods that may fail or require model switching. Thus, while planar scenes introduce an ambiguity, the algorithm remains robust and does not exhibit a degeneracy that leads to failure.

3 When used within RANSAC, what is the expected number of iterations the 5-point algorithm requires to find an outlier-free set?

Hint: take same assumptions of the lecture notes

Sol.

Let e be the outlier ratio among the point correspondences. The probability that a random sample of five points contains no outliers is $(1-e)^5$. Therefore, the expected number of RANSAC iterations required to obtain an outlier-free sample is $1/(1-e)^5$.

Deliverable 2 - Designing a Minimal Solver

You are required to solve the following problems:

1 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.

Hint: take same assumptions of the lecture notes

Sol.

Given the relative rotation R from the IMU, the remaining unknowns are the translation direction $\mathbf{t}$ (a 3-vector defined up to scale, hence 2 degrees of freedom). Each point correspondence $(\mathbf{q}_1, \mathbf{q}_2)$ in normalized image coordinates provides one linear constraint on $\mathbf{t}$ via the epipolar equation:

$$\mathbf{q}_2^\top [\mathbf{t}]_\times R\, \mathbf{q}_1 = 0.$$

Define $\mathbf{u} = R\mathbf{q}_1$ and $\mathbf{v} = \mathbf{q}_2$. The constraint simplifies to $\mathbf{t} \cdot (\mathbf{u} \times \mathbf{v}) = 0$. With two point correspondences, we obtain two vectors $\mathbf{a} = \mathbf{u}_1 \times \mathbf{v}_1$ and $\mathbf{b} = \mathbf{u}_2 \times \mathbf{v}_2$ that are both orthogonal to $\mathbf{t}$. Provided $\mathbf{a}$ and $\mathbf{b}$ are linearly independent, $\mathbf{t}$ is parallel to $\mathbf{a} \times \mathbf{b}$. The minimal solver proceeds as follows:

1) For each of the two point correspondences, compute $\mathbf{u}_i = R\mathbf{q}_{1,i}$ and $\mathbf{v}_i = \mathbf{q}_{2,i}$.
2) Compute $\mathbf{a} = \mathbf{u}_1 \times \mathbf{v}_1$ and $\mathbf{b} = \mathbf{u}_2 \times \mathbf{v}_2$.
3) If $\mathbf{a}$ and $\mathbf{b}$ are nearly collinear, the configuration is degenerate; discard the sample.
4) Otherwise, compute $\mathbf{t} = \mathbf{a} \times \mathbf{b}$ and normalize to unit length.

The translation direction $\mathbf{t}$ is recovered up to a sign ambiguity, which can be resolved later by cheirality checks using additional points.

2 *OPTIONAL* A: 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).

Sol.

The pseudo-code in appendix C-A outlines a RANSAC scheme that uses the above minimal solver to estimate the translation direction in the presence of outliers. This RANSAC procedure efficiently rejects outliers while exploiting the known rotation to reduce the minimal sample size to two points, thereby decreasing the required number of iterations compared to the standard five-point algorithm.

[1] Yilin zhang, OUC id: 23020036094, is with Faculty of Robotics and Computer Science, Ocean University of China and Heriot-Watt University, China Mainland zyl8820@stu.ouc.edu.cn

II. Team

Deliverable 3 - Initial Setup

This part deals with camera calibration for the drone, specifically obtaining the intrinsic matrix and distortion coefficients. The goal is to transform detected pixel coordinates into undistorted 3D bearing vectors that can be used for geometric vision tasks.

The implementation resides in the function `calibrateKeypoints`. It takes two sets of 2D keypoints and converts them into normalized bearing vectors using the camera parameters. Here's what the code does:

```
void calibrateKeypoints(
  const std::vector<cv::Point2f>& pts1,
  const std::vector<cv::Point2f>& pts2,
  opengv::bearingVectors_t& bearing_vector_1,
  opengv::bearingVectors_t& bearing_vector_2
) {
  std::vector<cv::Point2f> points1_rect,
   ↪  points2_rect;
  cv::undistortPoints(
    pts1, points1_rect,
    camera_params_.K, camera_params_.D
  );
  cv::undistortPoints(
    pts2, points2_rect,
    camera_params_.K, camera_params_.D
  );

  for (auto const& pt: points1_rect){
    opengv::bearingVector_t
     ↪  bearing_vector(pt.x, pt.y, 1);
    bearing_vector_1.push_back(bearing_vector⌋
     ↪  .normalized());
  }

  for (auto const& pt: points2_rect){
    opengv::bearingVector_t
     ↪  bearing_vector(pt.x, pt.y, 1);
    bearing_vector_2.push_back(bearing_vector⌋
     ↪  .normalized());
  }
}
```

After undistorting the points with `cv::undistortPoints`, each corrected 2D point (x, y) is lifted to a 3D direction vector $(x, y, 1)$. Since the intrinsic matrix has already been accounted for during undistortion, the effective focal length becomes 1, and the resulting vectors are normalized to unit length.

This function is called inside `cameraCallback` with a single line:

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

Once populated, `bearing\_vector\_1` and `bearing\_vector\_2` are passed to an OpenGV adapter:

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

This prepares the data for the RANSAC-based relative pose estimation that follows.

III. Some words

References

[1] D. Nistér, "An efficient solution to the five-point relative pose problem," in *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, 2003. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.86.8769&rep=rep1&type=pdf

Appendix A
Individual

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.

Appendix B
Team

Deliverable 3 - Initial Setup

Before we go to motion estimation, an important task is to calibrate the camera of the drone, i.e., to obtain the camera intrinsics and distortion coefficients. Normally you would need to calibrate the camera yourself offline to obtain the parameters.

However, in this lab the camera that the drone is equipped with has been calibrated already, and calibration information is provided to you! (If you are curious about how to calibrate a camera, feel free to check this ROS package)

As part of the starter code, we provide a function `calibrateKeypoints` to calibrate and undistort the keypoints. Make sure you use this function to calibrate the keypoints before passing them to RANSAC.

Deliverable 4 - 2D-2D Correspondences

Given a set of keypoint correspondences in a pair of images (2D - 2D image correspondences), as computed in the previous lab 5, we can use 2-view (geometric verification) algorithms to estimate the relative pose (up to scale) from one viewpoint to another.

To do so, we will be using three different algorithms and comparing their performance.

We will first start with the 5-point algorithm of Nister. Then we will test the 8-point method we have seen in class. Finally, we will test the 2-point method you developed in Deliverable 2. For all techniques, we use the feature matching code we developed in Lab 5 (use the provided solution code for lab 5 if you prefer - download it here). In particular, we use SIFT for feature matching in the remaining of this problem set.

We provide you with a skeleton code in `lab6` folder where we have set-up ROS callbacks to receive the necessary information.

We ask you to complete the code inside the following functions:

1 **`cameraCallback`: this is the main function for this lab.**
2 **`evaluateRPE`: evaluating the relative pose estimates**
3 **Publish your relative pose estimate**

Deliverable 5 - 3D-3D Correspondences

The rosbag we provide you also contains depth values registered with the RGB camera, this means that each pixel location in the RGB camera has an associated depth value in the Depth image.

In this part, we have provided code to scale to bearing vectors to 3D point clouds, and what you need to do is to use Arun's algorithm (with RANSAC) to compute the drone's relative pose from frame to frame.

1 **`cameraCallback`: mplement Arun's algorithm**

Performance Expectations: What levels of rotation and translation errors should one expect from using these different algorithms?

Summary of Team Deliverables: For the given dataset, we require you to run **all algorithms** on it and compare their performances.

Appendix C
Source Code

A. RANSAC Algorithm

The pseudo-code answered question in section I.

```
Input:
  - Set of point correspondences {(q1_i, q2_i)} (normalized coordinates)
  - Known rotation matrix R
  - Error threshold eps
  - Number of iterations N

Output:
  - Best estimate of translation direction t

Procedure:
  best_t = None
  best_inlier_count = 0

  for iteration = 1 to N:
    // 1. Randomly sample 2 correspondences
    sample = random_sample(correspondences, size=2)
    // 2. Compute translation direction using the minimal solver
    a = (R * sample[0].q1) x sample[0].q2
    b = (R * sample[1].q1) x sample[1].q2
    if norm(a x b) < small_value:   // degenerate case
      continue
    t_candidate = normalize(a x b)
    // 3. Form essential matrix E = [t_candidate]_x * R
    E = skew(t_candidate) * R
    // 4. Evaluate model on all correspondences
    inlier_count = 0
    for each correspondence (q1, q2):
      error = |q2^T * E * q1|   // or symmetrized epipolar distance
      if error < eps:
        inlier_count += 1
    // 5. Update best model
    if inlier_count > best_inlier_count:
      best_inlier_count = inlier_count
      best_t = t_candidate

  // (Optional) Refine using all inliers
  if best_t is not None:
    Gather all inliers according to best_t and E
    Solve for t by minimizing ||A t|| subject to ||t||=1,
        where each row of A is ( (R*q1) x q2 )^T
    Update best_t with the refined direction

  return best_t
```