1 What I've done

- Installed ROS, vicon_bridge, etc. on desktop.
- Updated recording code in quadcopter so everything (commands, odometry, colour images, depth images) are recorded into one rosbag.
- Flew quadcopter, but it crashed a few times so I still don't have good data.
- Plotted trajectory for 20 frames of external dataset.
- Discarded matches for which the depth of one of the points is recorded as 0 (or less) for use with Kabsch.
- Started investigating what's going wrong in my Kabsch implementation

2 Parts of report to look at

- Introduction (section 2, page 2)

3 Questions

- What did you think of the gantt chart?
- When do you want me to send you drafts of report sections (day, time so that you will have time to read it before the meeting)?

4 Comments

- There's definitely something wrong with the conversion of points from image to camera frame (or at least the scaling). See Figure 2a.
- It also seems like I'm working out the rotation properly for the points, but not the position. Actually when no translation is applied it seems to align well, which also indicates that there is probably an issue with the conversion to 3D points in the camera frame. The fact that Kabsch is giving the wrong translation is s abit of a worry, but it could just be due to numerical errors. See Figures 2b and 2c.

  - I'm currently working out the translation by summing up all of the points in each point cloud (component-wise) and dividing by the number of points to get the centroid of each, and then subtracting the centroid of the first point cloud from the centroid of the second point cloud.

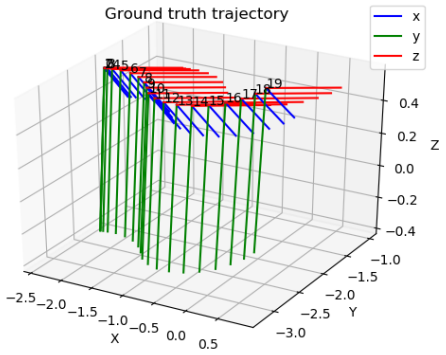
- For some reason the ethernet cable wasn't letting the desktop connect to the Vicon or the quadcopter. The wifi worked ok though.

- Something weird is going on with the SD card in the TX2. It works fine when the TX2 is connected to a monitor, but when connecting via SSH it says I don't have permission to cd into it or write anything to it. I tried adding permission with chmod +xr, but then when I cd into it it says there is nothing there and it still won't let me write to it.

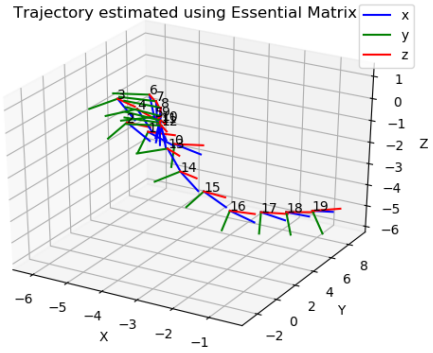
  - We tried re-formatting the SD card, and that didn't work. We also tried swapping to a different SD card, and that also didn't work.
  - For now I'm just going to record rosbags on the TX2 and not use the SD card.

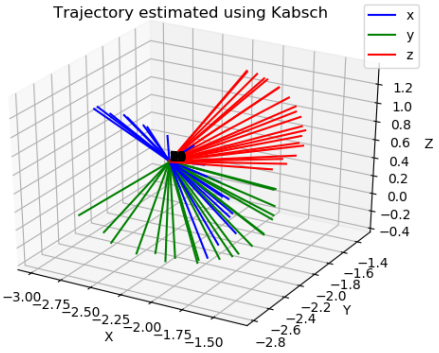
- When I was flying yesterday, the quad crashed in roughly the same spot it did before (far left corner). I tried to fix the Vicon, but I think I ended up flipping the orientation as it flew straight into the far wall (Jean Luc told me that he and Pieter changed which way was forward, I think I may have unknowingly changed it back).

- I'm planning to try fly again after today's meeting. I need to grab the locking pliers off Alex as I need to swap out one of the propellers that got chipped in the last crash yesterday.



(a) Trajectory, ground truth

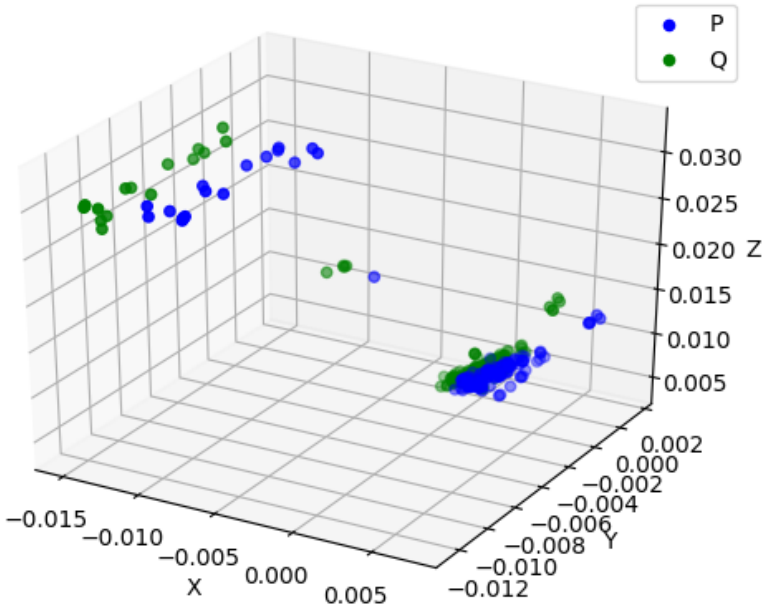




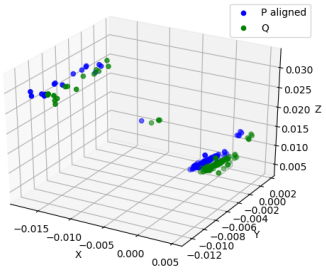
(b) Estimated trajectory for Essential Matrix method

(c) Estimated trajectory for Kabsch method

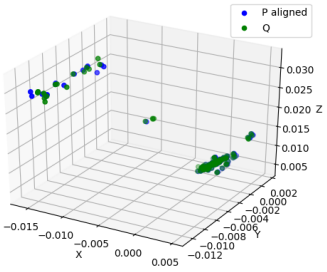
Figure 1: Trajectory visualizations for external dataset freiburg2/pioneer_slam



(a) Point clouds used for one frame of Kabsch. Note the axes scales (in metres) are much smaller than they should be



(b) Point clouds used for one frame of Kabsch, first point cloud P aligned with second point cloud Q using found rotation and translation



(c) Point clouds used for one frame of Kabsch, first point cloud P aligned with second point cloud Q using found rotation with no translation

Figure 2: Investigating problems with this report's Kabsch implementation

5 Algorithm Description

5.1 Shared

For each approach, the reconstructed trajectory of the camera is stored in two arrays: one with the x,y and z components of the position, and another with the w, x, z and z components of a Quaternion that gives the orientation. These formats were chosen as it is how the ground truth is stored.

The initial position and orientation are initialized as the first values in the ground truth.

Next, the frames are looped over. Right now the timestamps are not aligned, so I'm looping over the number of depth images as there are less of them. The following will describe what happens in each loop.

The current RGB image and the one from the next timestamp are both loaded. The keypoints of each image are found using SIFT, they are then matched using a brute force method. The matches are then ordered in terms of their distance (i.e. how good a match they are).

The matches are then iterated over, and matches with a distance of less than 180 (found via trial and error) are kept. If there are less than 6 points the Essential Matrix cannot be found, so the loop is restarted with the next frame. If there are more than 6 points, two different methods are used to recover the camera pose: finding the Essential Matrix (see Section 5.2), and aligning matched 3D points using Kabsch (see Section 5.3).

5.2 Essential Matrix

The Essential Matrix E can be found in opencv using matched points and the focal length and principal point of the camera. This is done using Nister's five-point algorithm [1]. Note that in practice more than five points may be required to ensure the estimation is robust and accurate. With more than five points, the fve-point algorithm can be used as a hypothesis generator within a random sample consensus scheme (RANSAC). That is, five point correspondences are chosen at random, then the five-point algorithm is used to estimate the Essential matrix from them, this estimate is a hypothesis. This process is repeated to get a number of hypotheses, which are scored using a robust statistical measure over all points. This gives an overall estimate of the best hypothesis, which can be improved iteratively as more sets of five point correspondences are used. In opencv this process does

not necessarily use all possible sets of five point correspondences; it stops once the probability of the hypothesis being correct has reached the specified probability threshold.

The algorithm itself uses the fact that a real non-zero 3×3 matrix E is an essential matrix if and only if it satisfies

$$EE^T E - \frac{1}{2} trace(EE^T)E = 0$$

There are additional constraints imposed by each point correspondence: $\tilde{q}^t \tilde{E} = 0$ where

$$\tilde{q} \equiv \begin{bmatrix} q_1 q'_1 & q_2 q'_1 & q_3 q'_1 & q_1 q'_2 & q_2 q'_2 & q_3 q'_2 & q_1 q'_3 & q_2 q'_3 & q_3 q'_3 \end{bmatrix}^T$$
$$\tilde{E} \equiv \begin{bmatrix} E_{11} & E_{12} & E_{13} & E_{21} & E_{22} & E_{23} & E_{31} & E_{32} & E_{33} \end{bmatrix}^T$$

The $\tilde{q}^T$ vectors for the five points can then be stacked to get a 5×9 matrix. Then four vectors $\tilde{X}, \tilde{Y}, \tilde{Z}, \tilde{W}$ that span the right nullspace of this matrix are found (when there are multiple possibilities, the ones corresponding to the smallest singular values are used). These vectors correspond to four 3×3 matrices X, Y, Z, W, which can be used to find the Essential Matrix as

$$E = xX + yY + zZ + wW$$

for some scalars x, y, z, w, it is assumed $w = 1$ as these scalars are only defined up to a common scale factor. This equation can then be inserted into the 9×5 constraint matrix. This can then be used to find x, y and z (see [1] for details), and thus E can be found from $E = xX + yY + zZ + wW$.

Opencv can then be used to recover the relative pose difference of the camera using the above information and the Essential Matrix. The process for doing this is also described in [1]. At least one point is needed to resolve ambiguities by imposing a cheirality constraint (i.e. all scene points should be in front of the cameras).

5.3 Kabsch

Once the matched points have been found, a depth needs to be associated with each one. Currently the pixel with the same indices as the RGB image is taken from the depth image. However upon inspecting the data more closely it seems like the RGB and depth images are not aligned, thus more processing will be necessary to find the corresponding point in the depth image.

Once the corresponding depth has been found, a 3D point is defined with the x and y components from the RGB image width and height, respectively and the depth as the z component. As these points are matched, the Kabsch algorithm can be used to align them.

However, before this can be done the points need to be converted out of the image frame and into the camera frame. To do this, the following equations are used

$$\begin{aligned} z^c &= z^i/sf \\ x^c &= (x^i - cx) * z^c/fx \\ y^c &= (y^i - cy) * z^c/fy \end{aligned}$$

Where the superscript indicates the frame (c for camera, i for image), sf is a scaling factor provided by the makers of the dataset, cx and cy are the x and y components of the optical center, and fx and fy are the x and y components of the focal length.

Then Kabsch is applied. First, a P and Q matrix are defined, with each row being a point (first column x, second y, third z).The points are then centered, this is done by subtracting from each row in the matrix the centroid of that matrix. (The centroid is the sum of all the elements divided by the number of elements, done separately for each component). The translation is then given by the centroid of Q minus the centroid of P.

Next, the cross-covariance matrix, H, is found:

$$H = P^T Q$$

The rotation matrix from P to Q is given by

$$R = (H^T H)^{1/2} H^{-1}$$

To actually calculate this, the singular value decomposition (SVD) is used:

$$\begin{aligned} H &= USV^T \\ d &= \det(VU^T) \\ R &= V \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & d \end{bmatrix} U^T \end{aligned}$$

5.4 Updating global trajectory

After the pose is acquired (as a translation and rotation matrix), the global trajectory needs to be updated. We have pose in the camera frame, so we can update the trajectories as follows:

$$Q_m^{g,n} = Q_m^{g,n-1} * Q_m^{c,n}$$
$$P_m^{g,n} = P_m^{g,n-1} + Q_m^{g,n-1}(P_m^{c,n})$$

Where Q are the orientations (in quaternions), and P are the positions. The subscript is for the method used for registration (rgb for using only the RGB images via the Essential Matrix, and d for also using the depth images via Kabsch). The superscript starts with the frame the measurement is in (g for global, c for camera, i for image) and then gives the timestep the measurement is for (where n is the current timestep).

Note that the first position is not rotated, as it is already in the global frame. The second position is rotated, as it is in the camera frame (this is the translation found by registration). This position is rotated by the global rotation from the last timestep, as it is in the camera frame (i.e. with respect to the camera orientation at the end of the last timestep).

References

[1] D. Nister. An efficient solution to the five-point relative pose problem. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 2, pages II–195. IEEE, 2003.