1 What I've done

- Installed ROS on TX1, got new RealSense working with ROS
- Worked on registration algorithm: better graphs, trying different things for the rotations
- Wrote plan for remaining weeks

2 Parts of report to look at

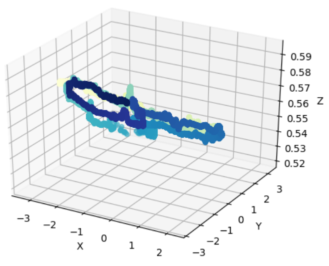
- Nothing since last week

3 Questions

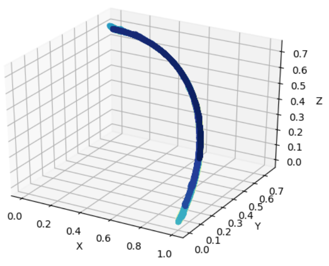
-

4 Comments

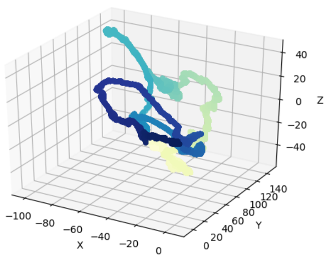
- I should be able to use the TX1 for ROS and write a script to extract data from the ROS bags and save it in a different format so that I can process it on my computer.
- I took the positive axis of rotation for all of the trajectories.
- I removed frames where there weren't enough matches with a small "distance" (measure of how good estimate is). But I think this actually made it worse.
- It does look like the ground truth trajectory is global.
- I tested the alignment of the images using ginput in MATLAB. I used two corners I could see fairly well on both the RGB and depth images. I got the coordinates as (281.0000, 173.0000), (40.0000, 174.0000) for the RGB and (281.0000, 169.0000), (36.0000, 174.0000) for the depth. They don't match up perfectly, but that's likely because (a) the timesteps aren't the same and (b) possible errors in my clicking on the points. Thus the RGB and depth images appear to be aligned.



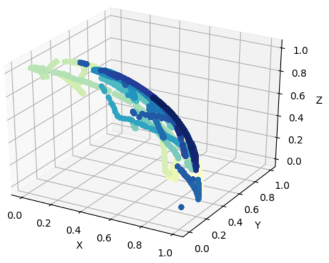
(a) Position trajectory, ground truth



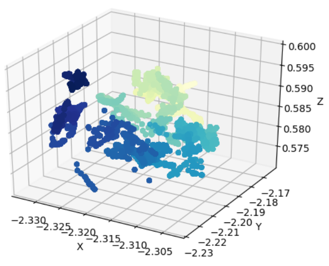
(b) Axis of rotation, ground truth



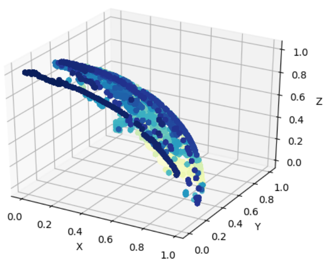
(c) Position estimated trajectory for Essential Matrix method



(d) Axis of rotation estimation for Essential Matrix method

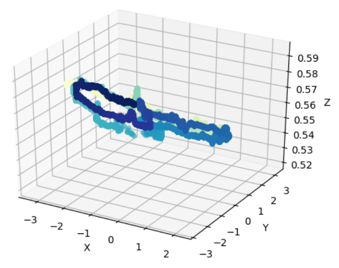


(e) Position estimated trajectory for Kabsch method

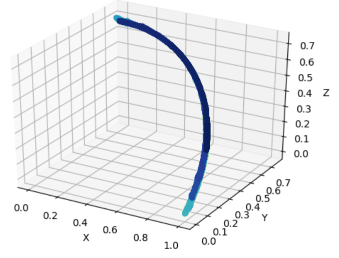


(f) Axis of rotation estimation for Kabsch method

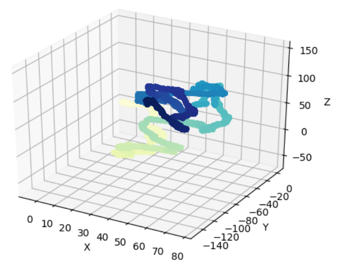
Figure 1: Trajectory visualizations. Begins at yellow, then goes into green and then blue



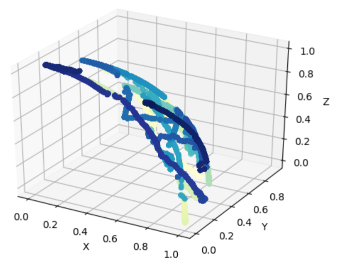
(a) Position trajectory, ground truth



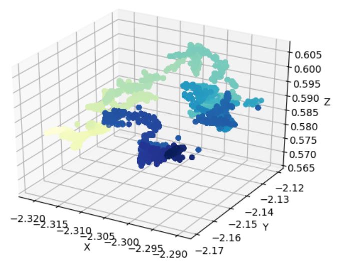
(b) Axis of rotation, ground truth



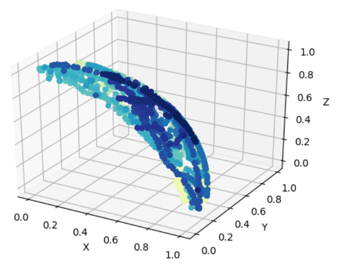
(c) Position estimated trajectory for Essential Matrix method



(d) Axis of rotation estimation for Essential Matrix method

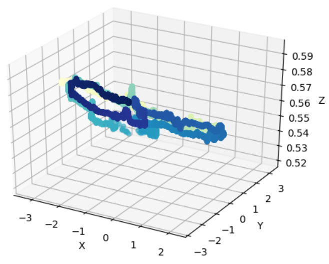


(e) Position estimated trajectory for Kabsch method

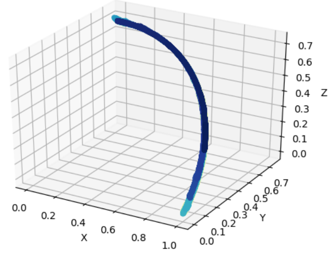


(f) Axis of rotation estimation for Kabsch method

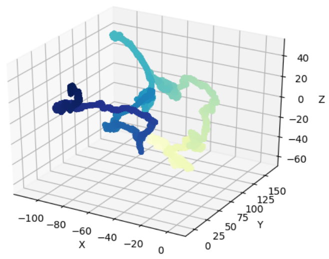
Figure 2: Trajectory visualizations, inverted rotation and translation. Begins at yellow, then goes into green and then blue



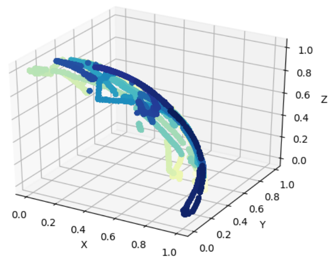
(a) Position trajectory, ground truth



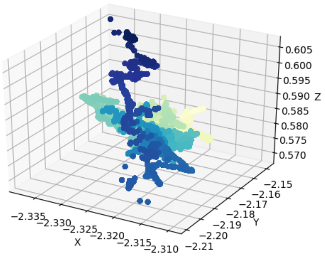
(b) Axis of rotation, ground truth



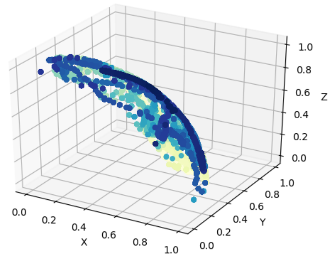
(c) Position estimated trajectory for Essential Matrix method



(d) Axis of rotation estimation for Essential Matrix method

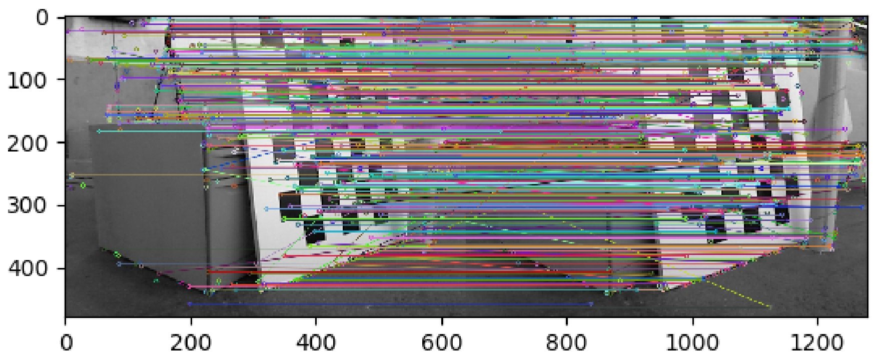


(e) Position estimated trajectory for Kabsch method



(f) Axis of rotation estimation for Kabsch method

Figure 3: Trajectory visualizations, skipped frames with not enough good matches. Begins at yellow, then goes into green and then blue


(a) Before RANSAC

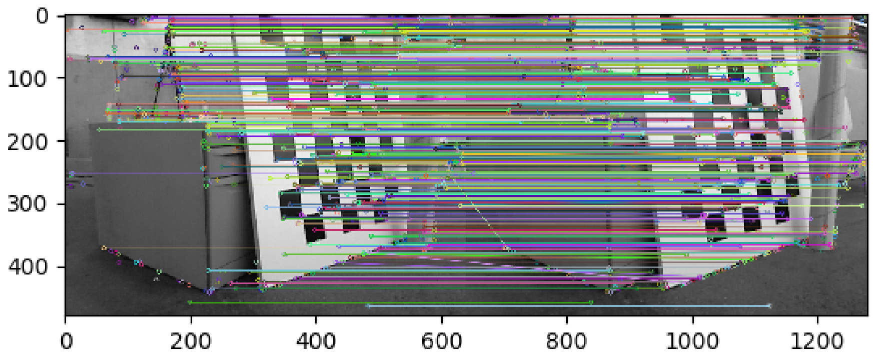

(b) After RANSAC

Figure 4: Feature matches before and after RANSAC with prob=0.999, threshold=1.0. Note that RANSAC has not removed all outliers

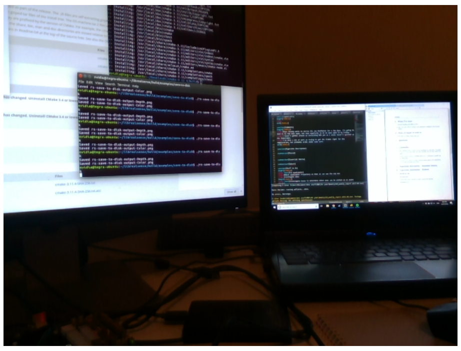
5 Algorithm Description

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,


(a) Colour Image (640 x 480)

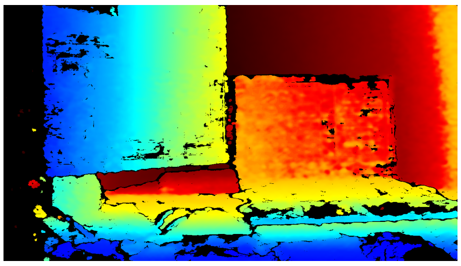

(b) Depth Image (1280 x 720)

Figure 5: Images captured from RealSense RD415, note that the image dimensions are different and the depth image captures more of the scene to the right

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

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

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

$$\tilde{q} \equiv \begin{bmatrix} q_1q'_1 & q_2q'_1 & q_3q'_1 & q_1q'_2 & q_2q'_2 & q_3q'_2 & q_1q'_3 & q_2q'_3 & q_3q'_3 \end{bmatrix}^T$$
$$\tilde{E} \equiv \begin{bmatrix} E_{11} & E_{12} & E_{13} & E_{21} & E_{22} & E_{23} & E_{31} & E_{32} & E_{33} \end{bmatrix}^T$$

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:

$$\begin{aligned} Q_m^{g,n} &= Q_m^{g,n-1} * Q_m^{c,n} \\ P_m^{g,n} &= P_m^{g,n-1} + Q_m^{g,n-1}(P_m^{c,n}) \end{aligned}$$

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.