## 1 What I've done

- Added axis-angle representation to the registration evaluation, still not really sure what a good way to visualize it is
- Tried to install ROS on my ubuntu dual boot, but it wasn't working (was failing on some security updates that aren't ROS specific, which it's looking for in the wrong place, I think)

## 2 Parts of report to look at

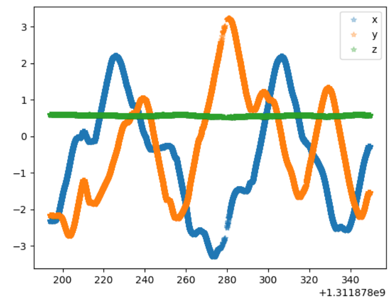
- Nothing since last week

## 3 Questions

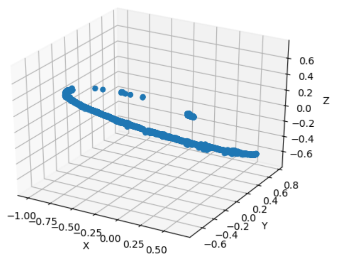
-

## 4 Comments

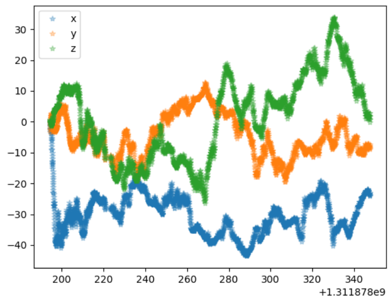
- A few different options for the timestamps:
  - Ty fix my ubunutu installation (probably the best option)
  - Just use the TX1 (might be a bit slow)
  - Try get ROS working in windows
  - Don't use ROS for the timestamps (do-able, but I'll need a laptop with ROS later anyway)
- I need to sort out what some way of reliably connecting my laptop to wifi when it's in the ubunutu boot so I can SSH to the quadcopter.
- Something is going very wrong with the Kabsch method, the translation is basically constant. The rotation is jumping around a lot.
- I think I mentioned it last week, but the depth and RGB images aren't aligned, I still haven't fixed this.
- I left the photos from the new RealSense and the algorithm descriptions from last week in this report.



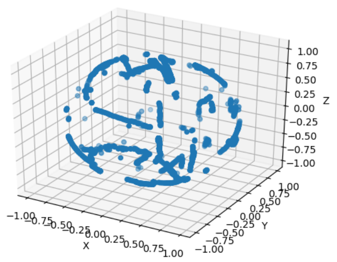
(a) Position trajectory, ground truth



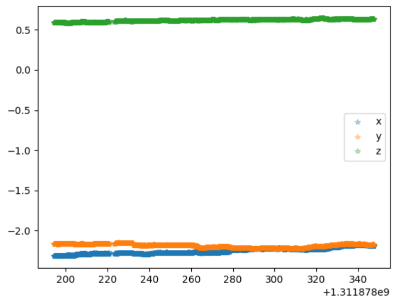
(b) Axis of rotation, ground truth



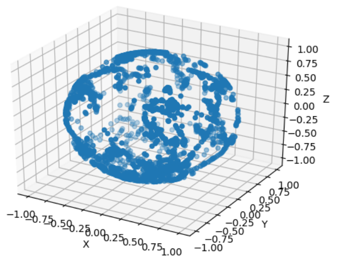
(c) Position estimated trajectory for Essential Matrix method



(d) Axis of rotation estimation for Essential Matrix method

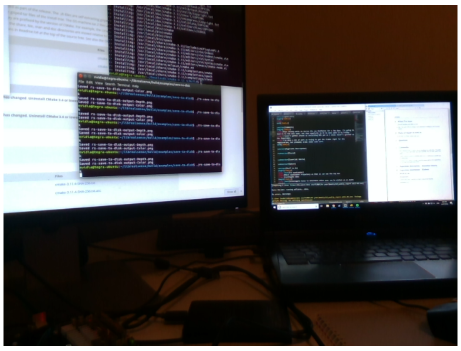


(e) Position estimated trajectory for Kabsch method



(f) Axis of rotation estimation for Kabsch method

Figure 1: Trajectory visualizations


(a) Colour Image (640 x 480)

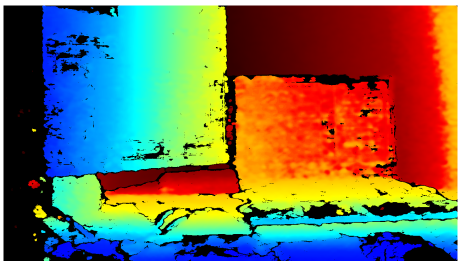

(b) Depth Image (1280 x 720)

Figure 2: Images captured from RealSense RD415, note that the image dimensions are different and the depth image captures more of the scene to the right

## 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 can be found in opencv using matched points and the focal length and principal point of the camera. Opencv can then be used to recover the pose difference between the two set of matched points using the above information and the Essential Matrix. I think this pose is given in the camera frame (before it moves) but I need to double check that I'm giving it the points in the right order.

## 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).

## 6 Stuff to do

- Adjust quadcopter trajectory so that it can see the top box
- Write acquisition code for D415
- Investigate data
  - ~~Investigate boxes to determine which ones can be picked up as point clouds~~
  - ~~Investigate RealSense cameras, compare the two cameras and using them on TX1/TX2 or Windows/Ubuntu~~
- Investigate registration algorithm
  - Generate 3D object in MATLAB that I can get points from from various camera poses (may also need to get RGB and depth images if we're going with that approach?)
  - Apply registration algorithm to generated point clouds (ground truth known) – without noise first, then add noise. Get error in true and estimated translation and rotation.
- Try registration on external dataset
  - Feature-only method (Essential Matrix)
  - Feature and depth method (Kabsch)
  - Install ROS and use it to align timestamps
  - Ensure that every pose is given with respect to the appropriate frame (ground truth is given in global frame, so each registered pose should be stored in global frame)
  - Incorporate IMU data
- Reading
  - ~~Incorporating RGB and depth images – feature matching (papers you sent me)~~
  - Write up summary of main points (parts of the algorithm, choices with advantages/disadvantages)
  - See if more recent papers have made advancements