Investigate the misalignment between Point Cloud and perspective image

Jingya Xun Boyu Wang

Objectives

- In this project, we are asked to investigate the misalignment between camera and point cloud images.
- We are given camera images taken from cameras mounted at various locations within a vehicle, namely the front, back, left, and right position.
- At the same time we are given data that can be processed into point cloud images taken from the same cameras, with the possibility of misalignment between the two sets of images.

Methodology - Convert Point Cloud Data into Image Form

We are given point cloud data described by (latitude, longitude, altitude, intensity)

- Step1 - Convert (latitude, longitude, altitude) points into earth-cencerted, earth-fixed (ECEF) coordinates. The following calculations are performed during this step.
  - Calculate ellipsoid flattening f: $f = \frac{a-b}{a}$ where a is the equatorial radius and b is the polar radius
  - Calculate eccentricity e: $e = \sqrt{f * (2-f)}$
  - Calculate the distance from the surface of the earth to z-axis along ellipsoid normal as a function of the latitude Φ.

$$N(\Phi) = \frac{a}{\sqrt{1-e^2 sin^2(\Phi)}}$$

  - Subsequently, the ECEF coordinates can be calculated as

$$X = \big(h + N(\Phi)\big)\cos(\lambda)\cos(\Phi)$$
$$Y = \big(h + N(\Phi)\big)\cos(\lambda)\sin(\Phi)$$
$$Z = \big(h + (1-e^2)N(\Phi)\big)\sin(\lambda)$$

where Φ is the given latitudes, λ is the given longitude, and h is the given altitude

Methodology - Convert Point Cloud Data into Image Form

We are given point cloud data described by (latitude, longitude, altitude, intensity)

- Step 2 - Convert ECEF coordinates into East North Up coordinates (ENU).
  The following calculations are performed during this step.
- Calculate rotation matrix R:

$$R = \begin{matrix} 1 & 0 & 0 \\ 0 & -\sin(\Phi) & \cos(\Phi) \\ 0 & \cos(\Phi) & sin(\Phi) \end{matrix} \cdot \begin{matrix} -\sin(\lambda) & \cos(\lambda) & 0 \\ \cos(\lambda) & \sin(\lambda) & 0 \\ 0 & 0 & 1 \end{matrix}$$

- Calculate ENU coordinates as:

P(e, n, u) = R * (P(X, Y, Z) - O(X_o,Y_o,Z_o), where P(X, Y, Z) are ECEF coordinates of the point cloud and O(X_o,Y_o,Z_o) are ECEF coordinates of the camera

We can rewrite P(e,n,u) further as:

$$P(e,n,u) = \begin{matrix} -\sin(\lambda) & \cos(\lambda) & 0 \\ -\cos(\lambda)\sin(\Phi) & -\sin(\Phi)\sin(\lambda) & \cos(\Phi) \\ \cos(\Phi)\cos(\lambda) & \cos(\Phi)\sin(\lambda) & \sin(\Phi) \end{matrix} \cdot \begin{bmatrix} X & X_o \\ Y & -Y_o \\ Z & Z_o \end{bmatrix}$$

Methodology - Convert Point Cloud Data into Image Form

We are given point cloud data described by (latitude, longitude, altitude, intensity)

- Step 3 - Convert ENU coordinates into camera coordinates.
  The following calculations are performed during this step.
- First calculate matrix Rq:

$$R_q = \begin{matrix} {q_s}^2 + {q_a}^2 - {q_y}^2 - {q_z}^2 & 2q_xq_y - 2q_sq_z & 2q_xq_z + 2q_sq_y \\ 2q_xq_y + 2q_sq_z & {q_s}^2 - {q_a}^2 + {q_y}^2 - {q_z}^2 & 2q_yq_z - 2q_sq_x \\ 2q_xq_z - 2q_sq_y & 2q_yq_z + 2q_sq_x & {q_s}^2 - {q_a}^2 - {q_y}^2 + {q_z}^2 \end{matrix}$$

- Camera coordinates can be calculated as:

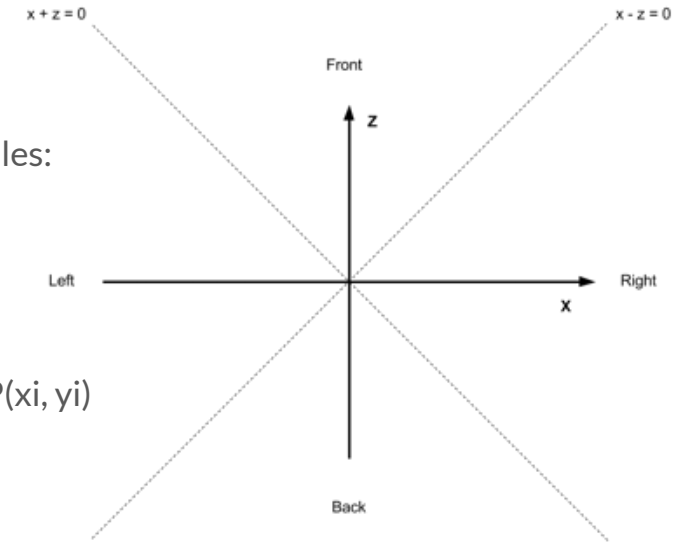
$$P(x, y, z) = R_q \cdot P(n, e, -u)$$

Methodology - Convert Point Cloud Data into Image Form

We are given point cloud data described by (latitude, longitude, altitude, intensity)

- Step 4 - Convert camera coordinates into image coordinates.
  The following calculations are performed during this step.
- We determine the direction of the image with the following rules:
  Front: z > 0, z > |x|, z > |y|
  Back: z < 0, z < -|x|, z < -|y|
  Left: x < 0, x < -|z|, x < -|y|
  Right: x > 0, x > |y|, x> |z|
- For each direction, we calculate the pixel points of the image P(xi, yi)

$$x_i = \frac{y}{z} \cdot \left(\frac{R_s - 1}{2}\right) + \left(\frac{R_s + 1}{2}\right)$$
$$y_i = \frac{x}{z} \cdot \left(\frac{R_s - 1}{2}\right) + \left(\frac{R_s + 1}{2}\right)$$



Methodology - Finding misalignments

Method 1 - Brute force matcher with ORB descriptors

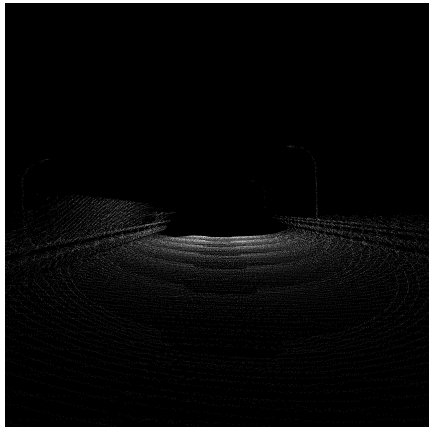
- The brute force matcher takes in a set of descriptors of one feature and compare with all the other features in the second set using distance calculation.
- We use Matcher.match() to get the best matches in the two given images.
- We sort the results in ascending order based on their distances.
- The top 50 matches are printed as shown in Fig. 1.

Methodology - Finding misalignments

Method 2 - Brute force matcher with SIFT Descriptors

- The brute force matcher takes in a set of descriptors of one feature and compare with all the other features in the second set using distance calculation.
- We use Matcher.knnmatch() to get the best matches in the two given images.
- We sort the results in ascending order based on their distances.
- The top 50 matches are printed as shown in Fig. 2.

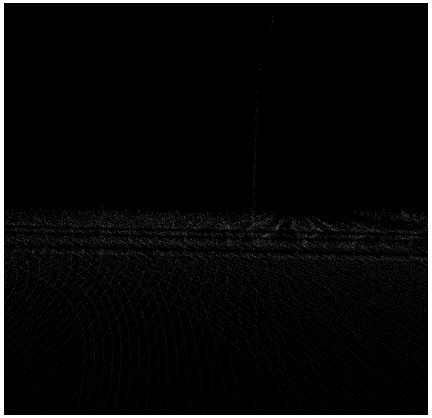
Results - Converted Point Cloud



Front

Back

Left



Right

Results-front image with orb

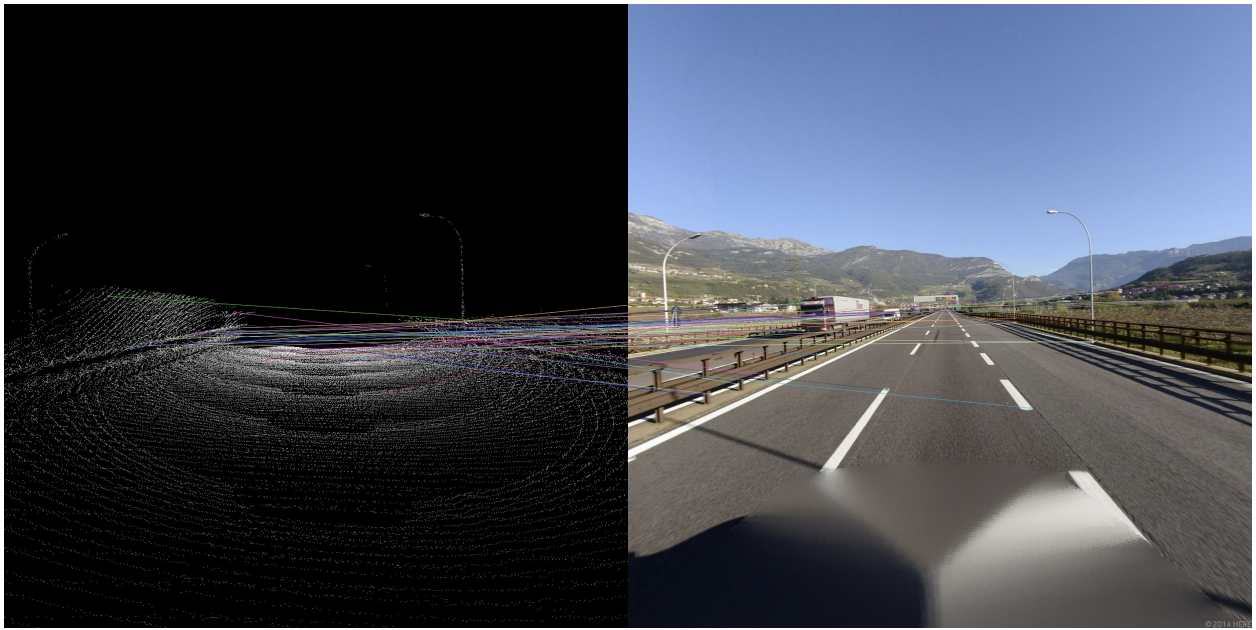


Fig. 1

Results-front image with SIFT

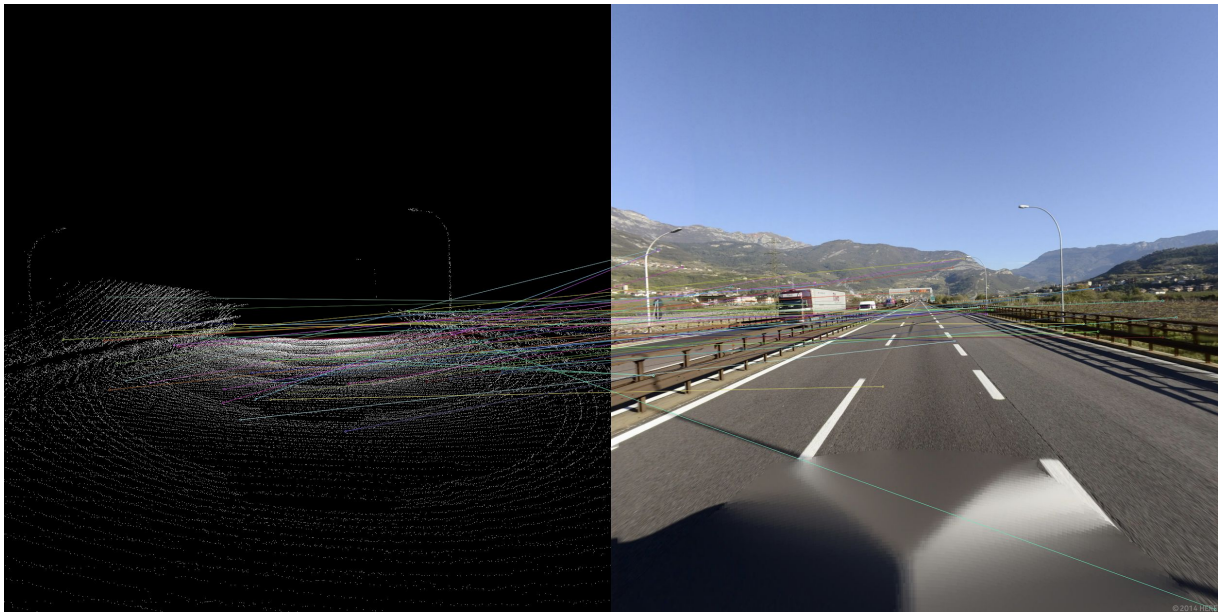


Fig. 2

Results-statistical misalignment by orb

Angle

Front image:3.7881

Back image:1.8836

Left image:5.3000

Right image:4.4313

Distance

Front image:524.240

Back image:615.850

Left image:686.336

Right image:622.037

Results-statistical misalignment by SIFT

Angle

Front image:7.0778

Back image: 8.5427

Left image:8.6217

Right image:5.3267

Distance

Front image:642.524

Back image:656.661

Left image:830.083

Right image:636.565

Discussion

- We are able to detect misalignments between point cloud and camera images using two different methods.
- The results of the misalignment detection are presented in the previous two slides.
- From the results of both methods, we notice some mismatches between the two images and suspect that is caused by the low resolution of the camera images and the generated point cloud images.

References

- https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html
- Coordinate Transformations lecture slides shared by the instructor.