Project2: Investigate the Misalignment between Point Cloud and Perspective Image

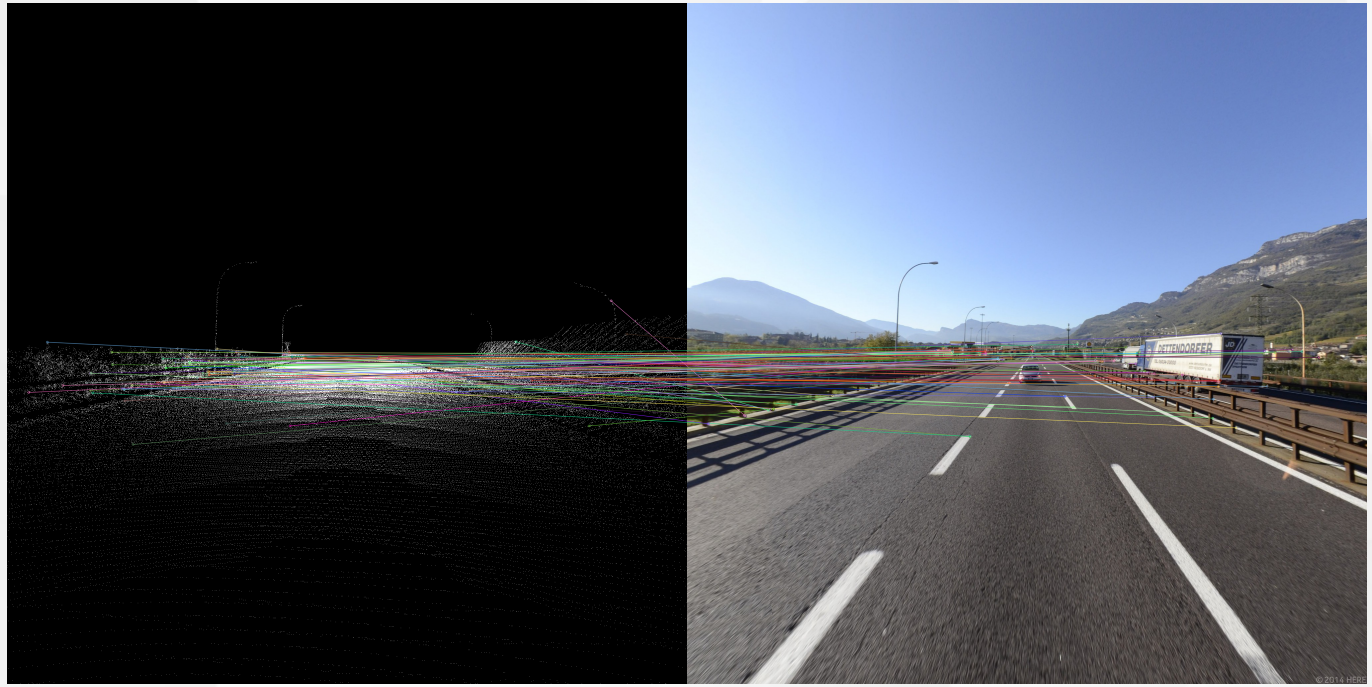
EECS 495: Geospatial Vision and Visualization

Lingfei Cui
Dan Wu
Sheng Ma

CONTENTS

PART 1: GOAL

- Transform the given data from final_project_point_cloud.fuse to visualize point cloud in image coordinate on four directions, front, back, left and right.
- Find misalignment between the visualized point cloud and original images in four directions.



PART 2: METHODOLOGY

- **Coordinate Transformations**

  Process: Input → LLA → ECEF → ENU → Camera Coordinate → Image Coordinate

  Output: Point cloud(without intensity information and with intensity information)

- **Calculate Misalignment**

  Feature-based technique:

  1. Brute-Force Matching with ORB Descriptors
  2. Brute-Force Matching with SIFT Descriptors and Ratio Test

  Quantify misalignment:

  1. Angle Misalignment
  2. Euclidean Distance Misalignment

PART 2: METHODOLOGY – Coordinate Transformation

- Step1: LLA → ECEF
- Parameter Statement

  Φ : geodetic altitude

  λ : longitude

  h : altitude

  a : equatorial radius

  b : polar radius

  e : eccentricity

  $N(\Phi)$: distance from the surface to z- axis along ellipsoid normal

- The ECEF coordinates X, Y and Z can be calculated from:

$$X = \big(h + N(\Phi)\big)\cos(\lambda)\cos(\Phi)$$

$$Y = \big(h + N(\Phi)\big)\cos(\Phi)\sin(\lambda)$$

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

PART 2: METHODOLOGY – Coordinate Transformation

- Step2: ECEF → ENU
- Parameter Statement

  $P(X, Y, Z)$: input location in ECEF coordinate

  $O(X_o, Y_o, Z_o)$: origin location in ECEF coordinate

  Φ_o : geodetic altitude of origin location

  λ_o : longitude of origin location

- East North Up Coordinate $P(e, n, u)$ can be calculated from:

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

PART 2: METHODOLOGY – Coordinate Transformation

- Step3: ENU → Camera Coordinate
- Parameter Statement

  $Q(q_s, q_x, q_y, q_z)$: camera unit quaternion input

  $P(n, e, -u)$: input from ENU coordinate

  R_q : orthogonal matrix

$$R_q = \begin{matrix} 1 - 2{q_y}^2 - 2{q_z}^2 & 2q_xq_y + 2q_sq_z & 2q_xq_z - 2q_sq_y \\ 2q_xq_y - 2q_sq_z & 1 - 2{q_x}^2 - 2{q_z}^2 & 2q_yq_z + 2q_sq_x \\ 2q_xq_z + 2q_sq_y & 2q_yq_z - 2q_sq_x & 1 - 2{q_x}^2 - 2{q_y}^2 \end{matrix}$$

- Camera coordinate $P(x, y, z)$ can be calculated from:

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

PART 2: METHODOLOGY – Coordinate Transformation

- Step4: Camera Coordinate → Image Coordinate
- Parameter Statement

  $P(x, y, z)$: input from camera coordinate

  R_s : resolution
- Image coordinate $P(x_i, y_i)$ can be calculated from:

- **Front:**

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

- **Back:**

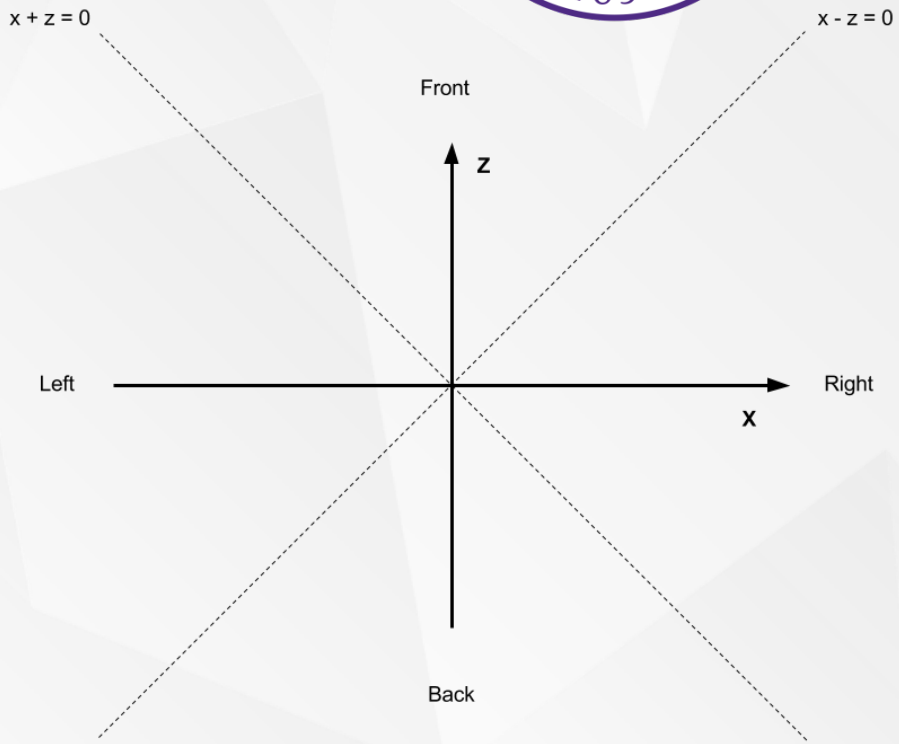
$$x_i = -\frac{y}{z} \cdot \left(\frac{R_s - 1}{2}\right) + \left(\frac{R_s + 1}{2}\right)$$
$$y_i = \frac{x}{z} \cdot \left(\frac{R_s - 1}{2}\right) + \left(\frac{R_s + 1}{2}\right)$$

- **Left:**

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

- **Right:**

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



PART 2: METHODOLOGY – Calculate Misalignment

- Method I: Brute-Force Matching with ORB Descriptors

1. Find the keypoints and descriptors with ORB
2. Create a BFMatcher object with distance measurement **cv2.NORM_HAMMING** and **crossCheck** is switched **on** for better results.
3. Use Matcher.match() method to get the best matches in two images.
4. Sort them in ascending order of their distances. (We only draw the best 100 matches)

```
# Calculate misalignment using ORB feature extraction
orb = cv2.ORB_create(1500)
# Create matcher
bf1 = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

# Front matching
(kp_f_orb, des_f_orb) = orb.detectAndCompute(img_front, None)
(kp_f0_orb, des_f0_orb) = orb.detectAndCompute(img_front0 , None)

matches1_orb = bf1.match(des_f_orb, des_f0_orb)
matches1_orb = sorted(matches1_orb, key=lambda val: val.distance)

front_matching_orb = cv2.drawMatches(img_front, kp_f_orb, img_front0, kp_f0_orb, matches1_orb[:100], None, flags = 2)
cv2.imwrite('front_matching_orb.png',front_matching_orb)
front_result_orb1 = calculate_misalignment_by_angle(kp_f_orb, kp_f0_orb, matches1_orb)
front_result_orb2 = calculate_misalignment_by_distence(kp_f_orb, kp_f0_orb, matches1_orb)
print ("The misalignment result for front image calculated by angle is ", front_result_orb1)
print ("The misalignment result for front image calculated by distence is ", front_result_orb2)
```

PART 2: METHODOLOGY – Calculate Misalignment

- Method II: Brute-Force Matching with SIFT Descriptors and Ratio Test

1. Use BFMatcher.knnMatch() to get k best matches. (We use k = 2 in order to apply ratio test explained by D.Lowe in his paper)
2. Other steps are similar to the previous method.

```python
# Calculate misalignment using SIFT feature extraction
# create sift extracter to extract sift features
sift = cv2.xfeatures2d.SIFT_create()
# create bf matcher
bf2 = cv2.BFMatcher()

# Front matching
kp_f_si, des_f_si = sift.detectAndCompute(img_front, None)
kp_f0_si, des_f0_si = sift.detectAndCompute(img_front0, None)
matches1_si = bf2.knnMatch(des_f_si, des_f0_si, k=2)

good1 = []
for m,n in matches1_si:
    if m.distance < 0.85 * n.distance:
        good1.append([m])

front_matching_si = cv2.drawMatchesKnn(img_front,kp_f_si,img_front0,kp_f0_si,good1, None, flags=2)
cv2.imwrite('front_matching_sift.png',front_matching_si)
```

PART 2: METHODOLOGY – Quantify Misalignment

- Method I: Angle Misalignment

Calculate the average angle differences between keypoints1 and keypoints2.

```
def calculate_misalignment_by_angle2(keypoints1, keypoints2, matches1to2):
    angles = []
    for m, n in matches1to2:
        img1_idx = m.queryIdx
        img2_idx = m.trainIdx
        (x1, y1) = keypoints1[img1_idx].pt
        (x2, y2) = keypoints2[img2_idx].pt
        x2 = x2 + 2048
        diff_point1_point2 = math.atan((float)(y2 - y1) / (x2 - x1)) * (180 / math.pi)
        diff_point1_point2 = abs(diff_point1_point2)
        angles.append(diff_point1_point2)
    return (sum(angles) / len(angles))
```

PART 2: METHODOLOGY – Quantify Misalignment

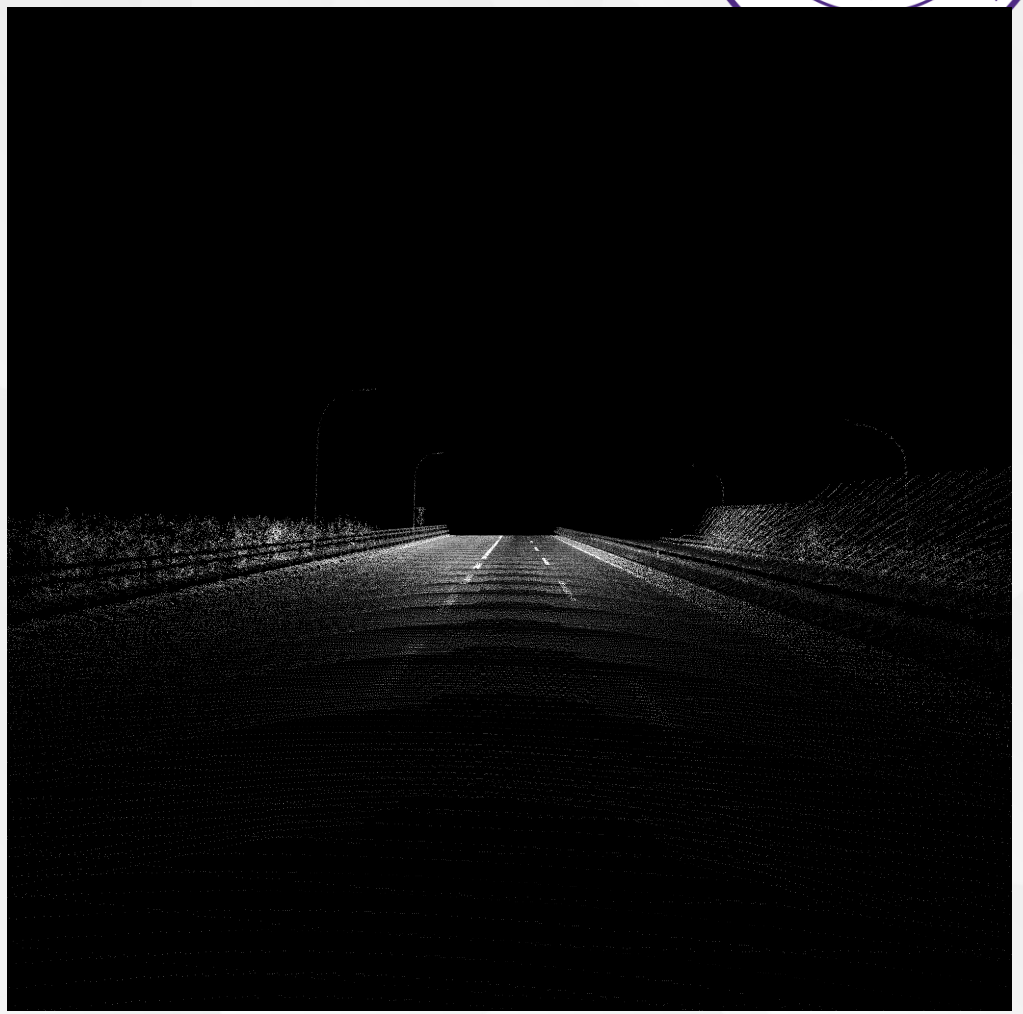
- Method II: Euclidean Distance Misalignment

Calculate the average euclidean distances between keypoints1 and keypoints2

```
def calculate_misalignment_by_distance2(keypoints1, keypoints2, matches1to2):
    euclidean = []
    for m, n in matches1to2:
        img1_idx = m.queryIdx
        img2_idx = m.trainIdx
        (x1, y1) = keypoints1[img1_idx].pt
        (x2, y2) = keypoints2[img2_idx].pt
        euclidean_distance = math.sqrt((float)(y2 - y1)**2 + (x2 - x1)**2)
        euclidean.append(euclidean_distance)
    return (sum(euclidean) / len(euclidean))
```

PART 3: RESULTS AND ANALYSIS

Visualization of point cloud with intensity information. (front and back)

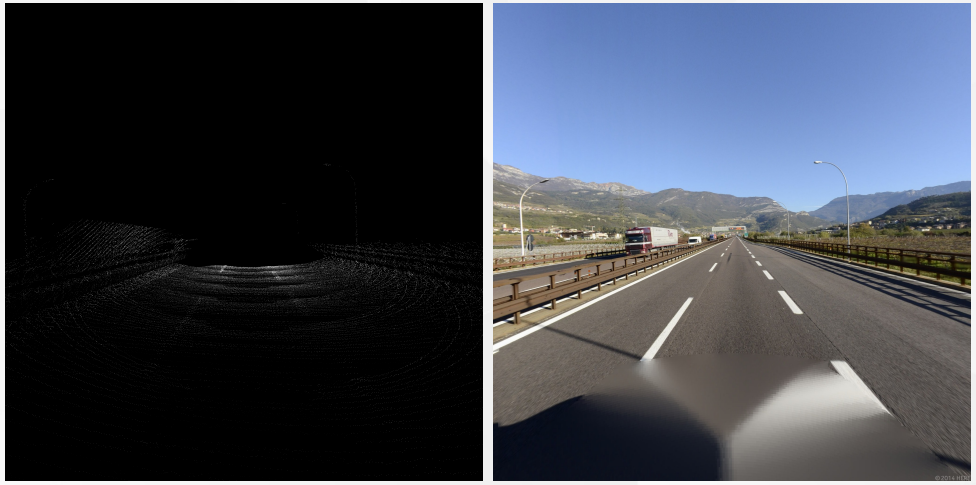


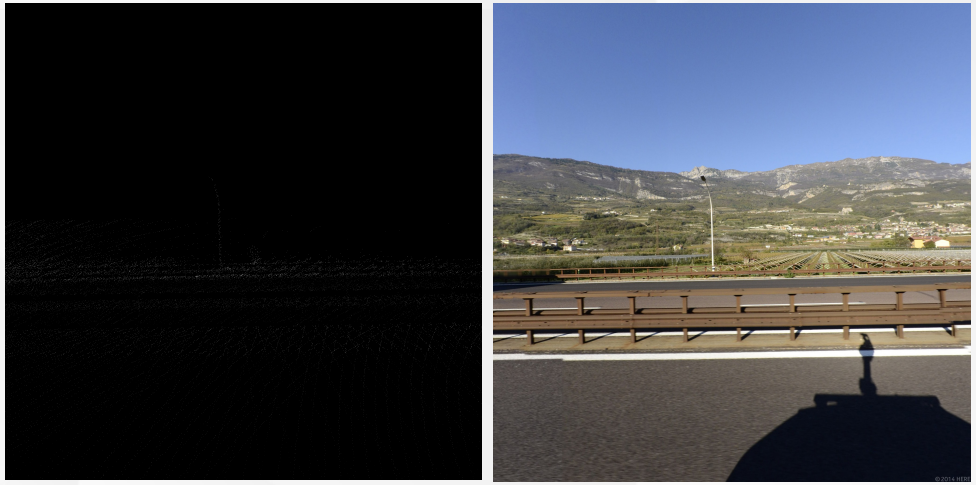
PART 3: RESULTS AND ANALYSIS

Visualization of point cloud with intensity information. (left and right)

PART 3: RESULTS AND ANALYSIS

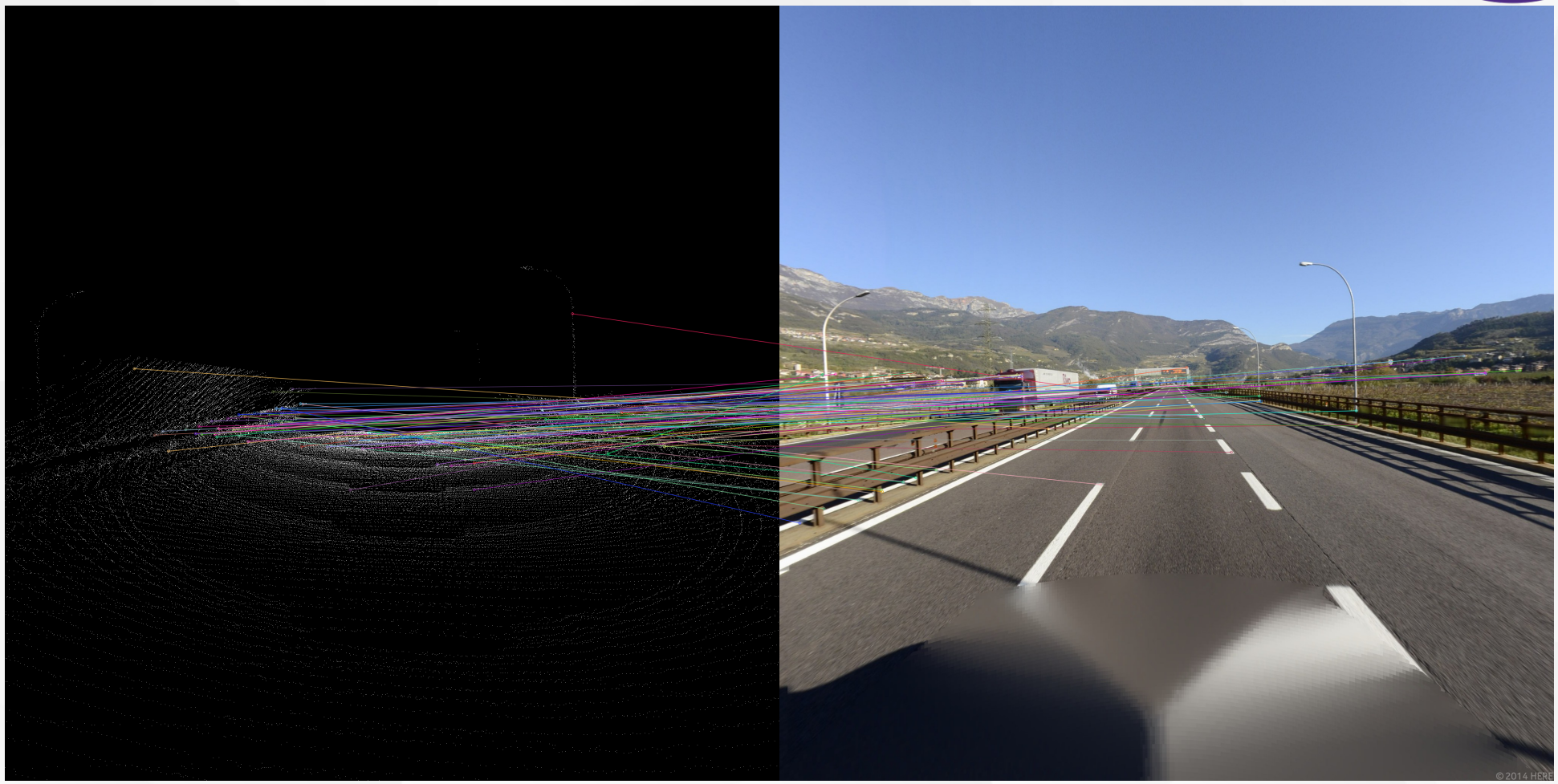
Visualization of point cloud and original images





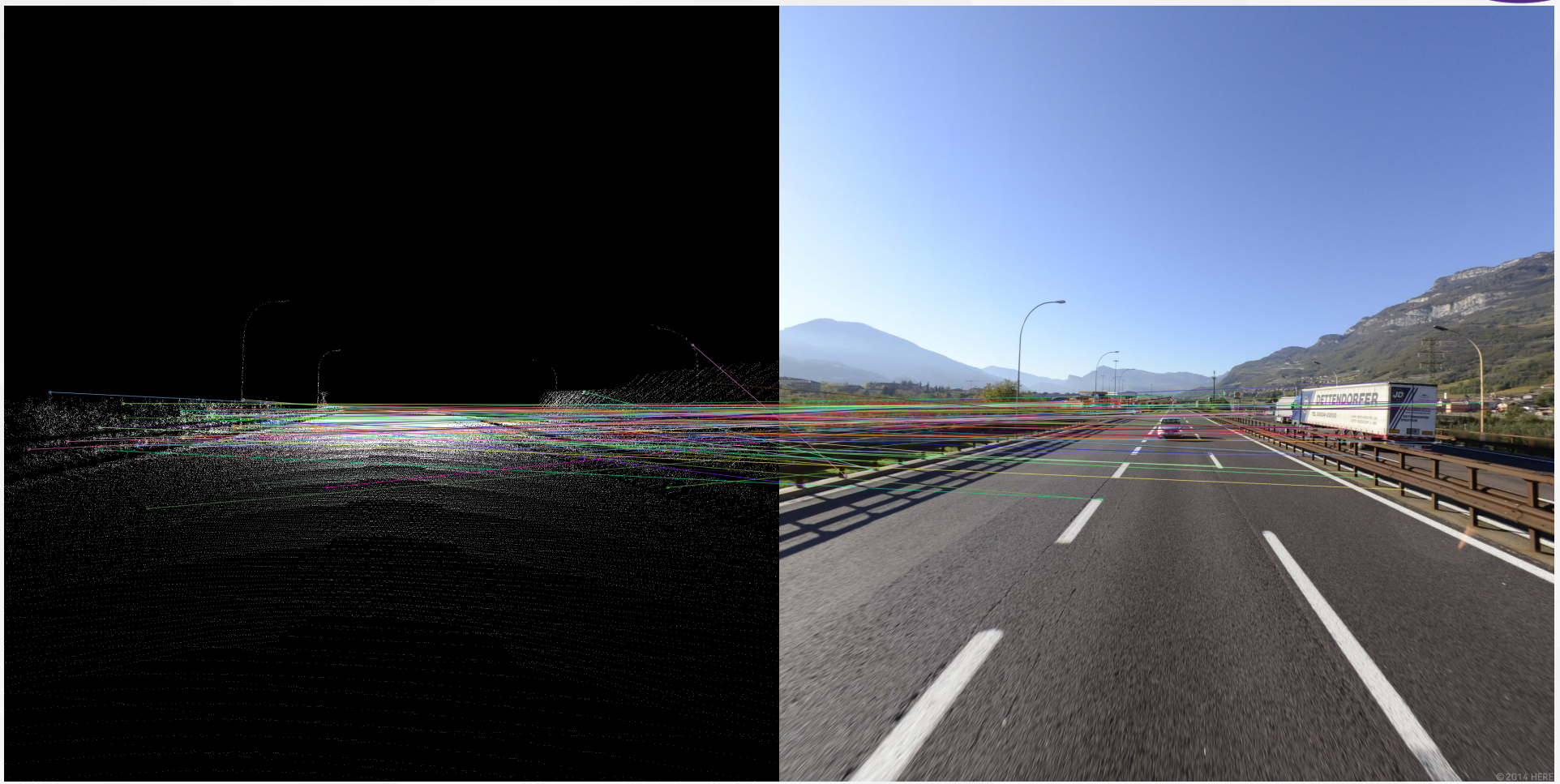
PART 3: RESULTS AND ANALYSIS

- front_matching_orb:



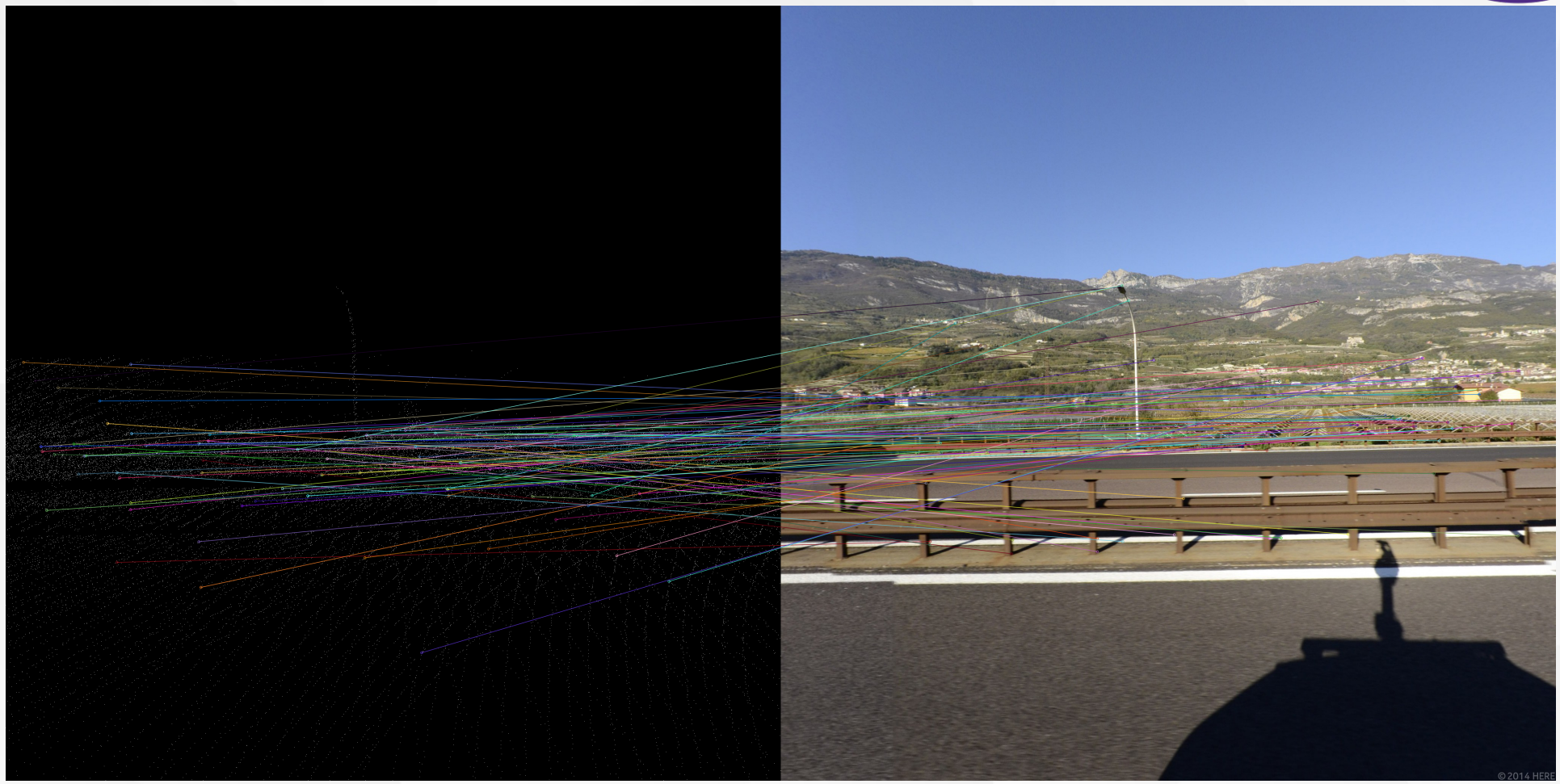
PART 3: RESULTS AND ANALYSIS

- back_matching_orb:



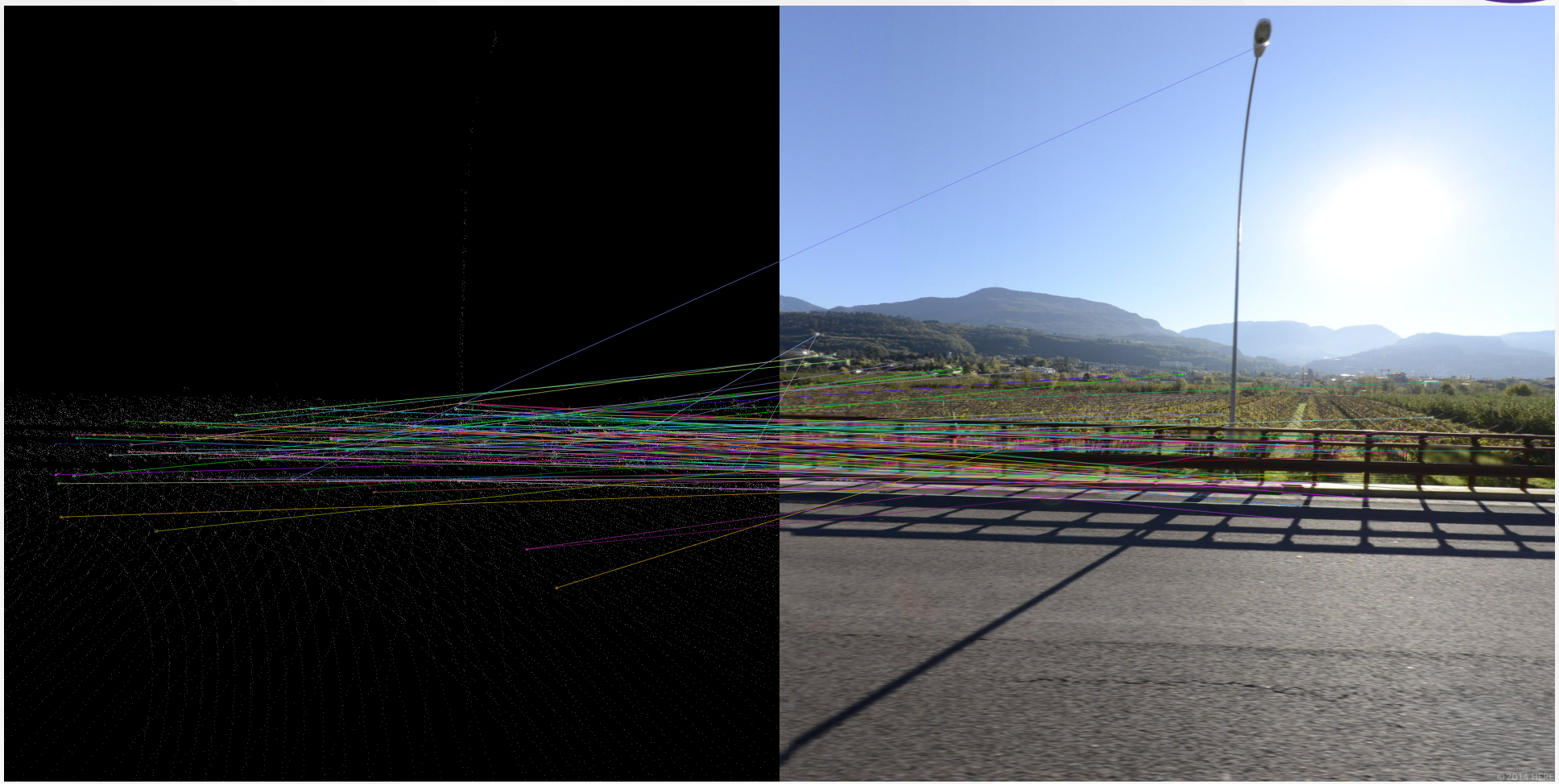
PART 3: RESULTS AND ANALYSIS

- left_matching_orb:



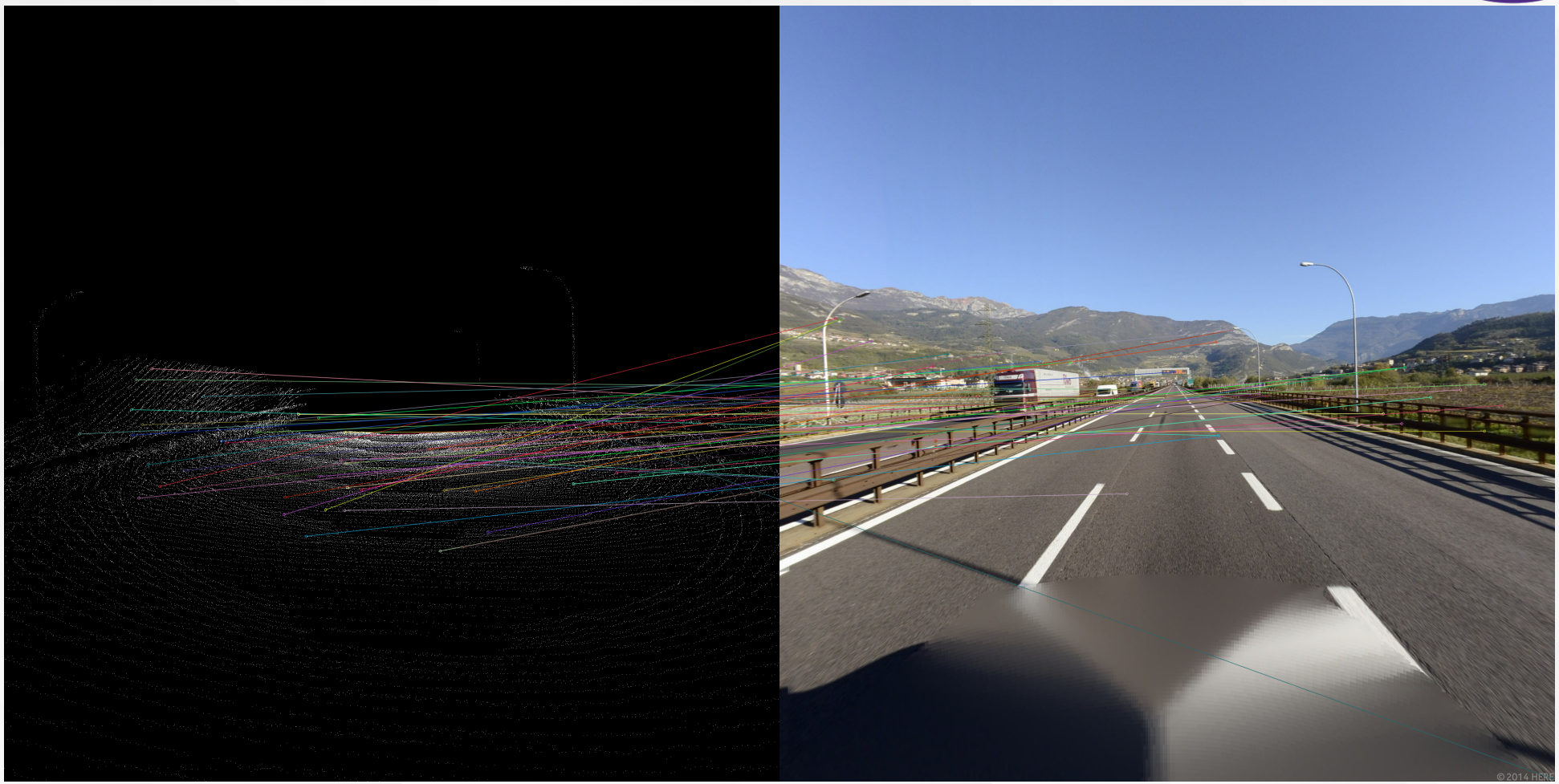
PART 3: RESULTS AND ANALYSIS

- right_matching_orb:



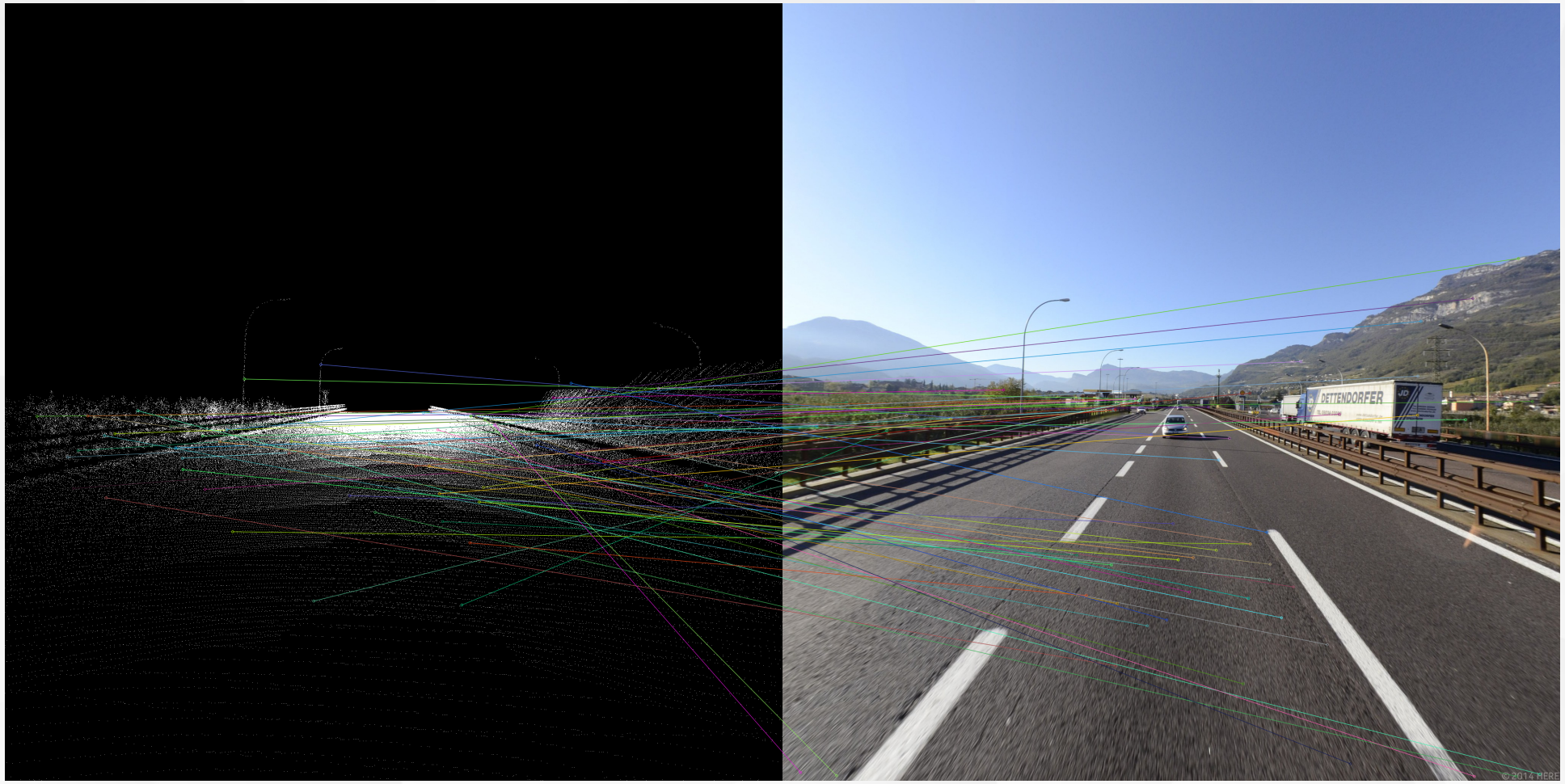
PART 3: RESULTS AND ANALYSIS

- front_matching_sift:



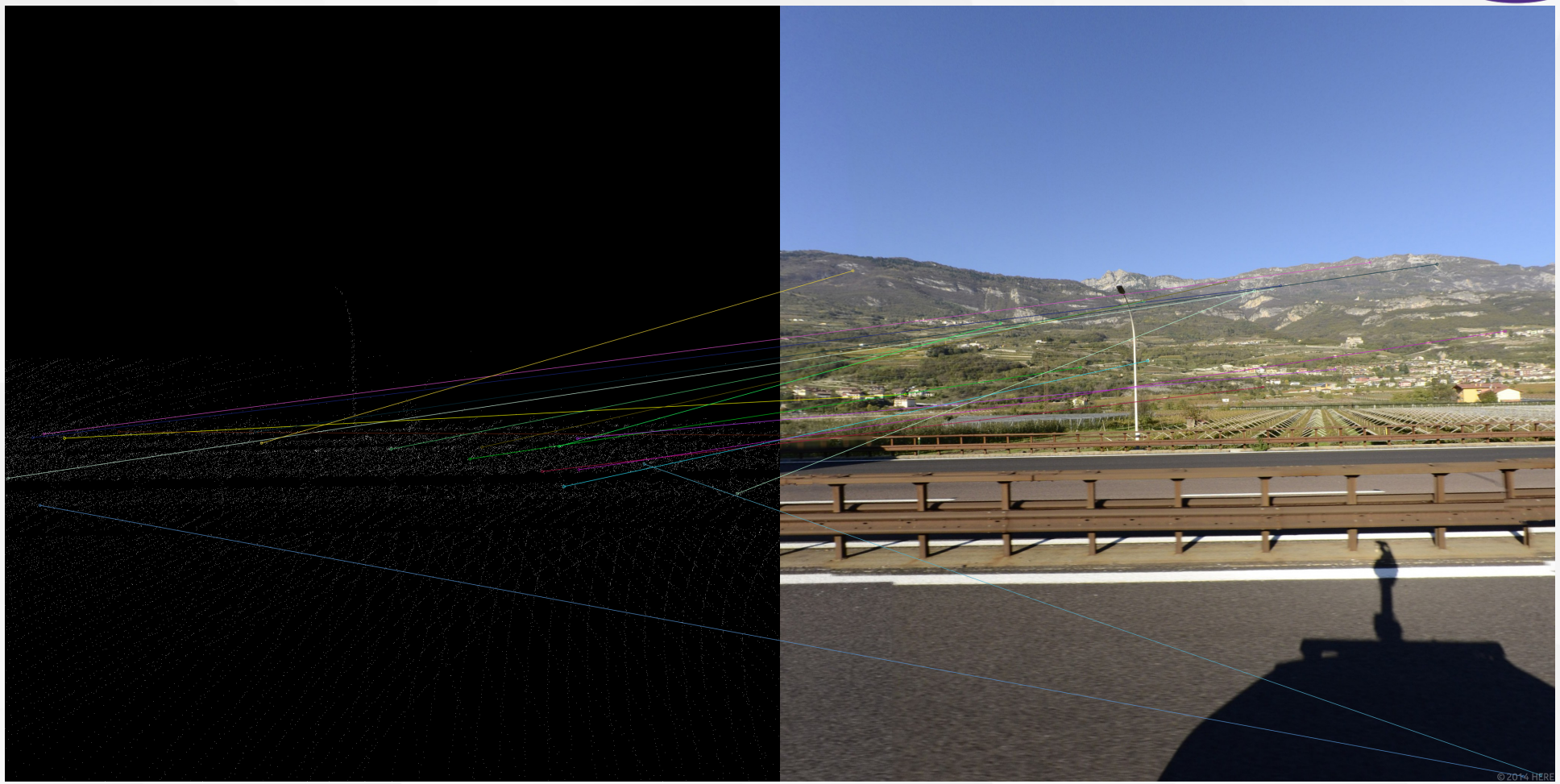
PART 3: RESULTS AND ANALYSIS

- back_matching_sift:



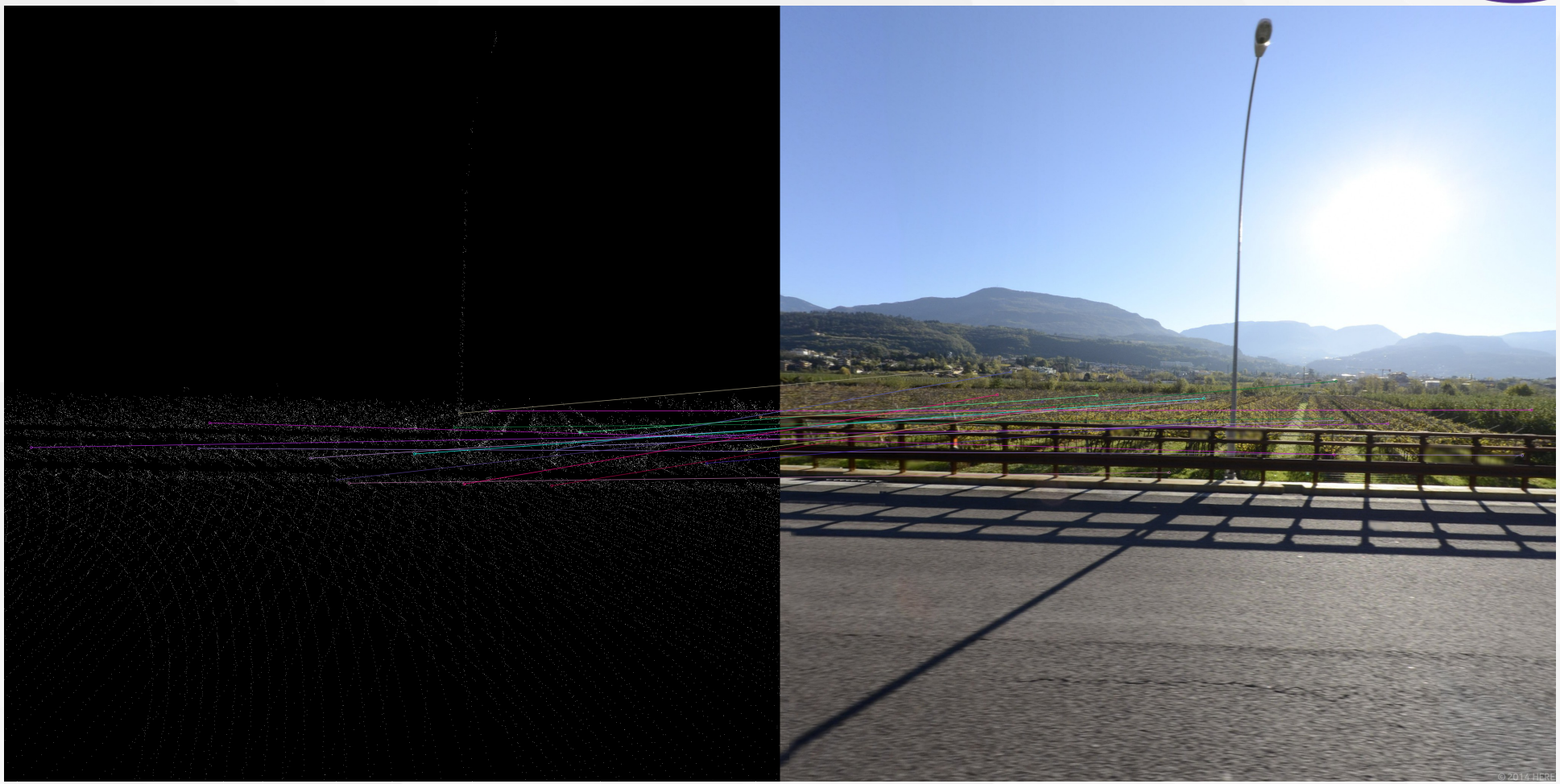
PART 3: RESULTS AND ANALYSIS

- left_matching_sift:



PART 3: RESULTS AND ANALYSIS

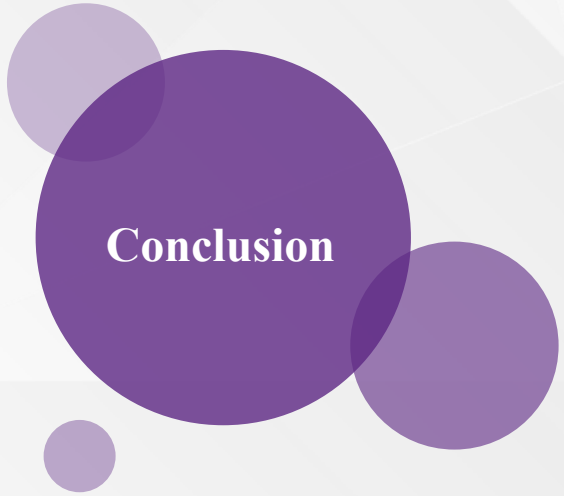
- right_matching_sift:



PART 3: RESULTS AND ANALYSIS

- Quantify misalignment:
  angle misalignment and euclidean distance misalignment

```
(Python36) bash-3.2$ python misalignment.py
--------------------------------------------------
Method1: Brute-Force Matching with ORB Descriptors
[ INFO:0] Initialize OpenCL runtime...
front
The misalignment result for front image calculated by angle is 3.7580994682540108
The misalignment result for front image calculated by distance is 588.4596764972877
back
The misalignment result for back image calculated by angle is 2.0925080115365646
The misalignment result for back image calculated by distance is 488.39558330701215
left
The misalignment result for left image calculated by angle is 5.584154870461994
The misalignment result for left image calculated by distance is 687.5257764616058
right
The misalignment result for right image calculated by angle is 3.97102094708418
The misalignment result for right image calculated by distance is 604.2796016945551
--------------------------------------------------------------------
Method2: Brute-Force Matching with SIFT Descriptors and Ratio Test
front
The misalignment result for front image calculated by angle is 7.082026354311747
The misalignment result for front image calculated by distance is 633.926099515695
back
The misalignment result for back image calculated by angle is 8.257248777233603
The misalignment result for back image calculated by distance is 652.1124033271328
left
The misalignment result for left image calculated by angle is 8.986099324396482
The misalignment result for left image calculated by distance is 835.1852914515773
right
The misalignment result for right image calculated by angle is 5.4262073451705355
The misalignment result for right image calculated by distance is 674.9901372760687
```



Misalignment Result

ORB

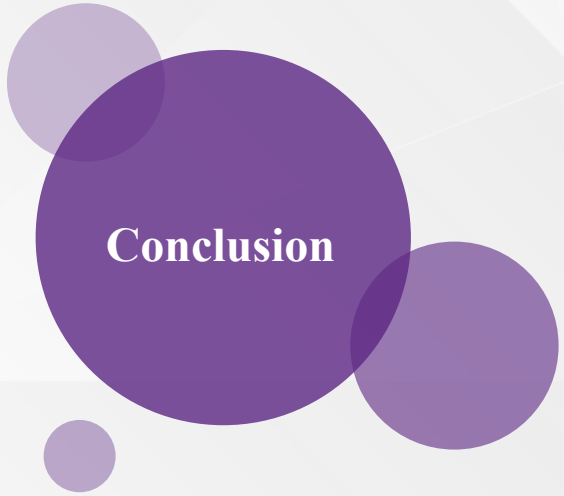
Angle

Front:3.7580994682540108
Back:2.0925080115365646
Left:5.584154870461994
Right:3.971020947084818

SIFT

Angle

Front:7.082026354311747
Back:8.257248777233603
Left:8.986099324396482
Right:5.4262073451705355



Misalignment Result

ORB

Distance

Front:588.4596764972877
Back:488.39558330701215
Left:687.5257764616058
Right:604.2796016945551

SIFT

Distance

Front:633.926099515695
Back:652.1124033271328
Left:835.1852914515774
Right:674.9901372760687

PART 4: FUTURES

Future Work

In this project, we use feature-based technique to investigate the misalignment between point cloud and perspective image. However, we found that the feature-based approach is not very intuitive and the result may changes with the extraction of different feature points. So in future work, we want to use direct-based technique to investigate the misalignment.

REFERENCES

Revised_Coordinate_Transformations.pptx

https://ww2.mathworks.cn/matlabcentral/fileexchange/7942-covert-lat--lon--alt-to-ecef-cartesian

https://ww2.mathworks.cn/help/map/ref/ecef2enu.html

https://docs.opencv.org/3.0-beta/modules/features2d/doc/drawing_function_of_keypoints_and_matches.html

https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html

https://www.programcreek.com/python/example/89342/cv2.drawMatchesKnn

Questions?

Thank You!

Lingfei Cui
Dan Wu
Sheng Ma