Carnegie Mellon Airlab Problem Statement

Arka Sadhu - 140070011
3rd Year Btech. Electrical Engineering

December 2016

Problem Statement for Q2

Q1. Name one advantage and one disadvantage of using Euler angles (e.g. Roll-Pitch-Yaw), unit quaternions, rotation matrices and axis/angle representations of rotations.
A1.

- Euler Angles:
  - Advantages:
    - Only 3 parameters need to be stored.
    - It is more human understandable and good for decomposing rotations into individual degrees of freedom.
  - Disadvantages:
    - Interpolation is difficult.
    - There are ambiguities in the order of Matrix Rotations.
    - If the middle rotation is by 90°, it results in a Gimbal Lock.
- Unit Quaternions:
  - Advantages:
    - There is no ambiguity, no gimbal lock, and interpolation can be done using slerp, producing smooth rotations.
    - It requires only 4 parameters to be stored.
    - Even translations can be accomodated using Dual Quaternions.
    - Computation is efficient for computers.
    - Consecutive rotation is simply multiplication of two quternions.
  - Disadvantages:
    - In general it is not easy to visualize Quaternions, and need to be converted to Axis Angle Representation.
    - The Quaternion-Algebra is quite involved.
    - There exists redundancy, ie two quaternions may imply the same rotation.
- Rotation Matrices
  - Advantages:
    - Rotations can be easily concatenated.
    - They directly give the new axis of rotations, which may be required in some applications.

    * With homogenous coordinates it is very easy to incorporate Translation too.
  - Disadvantages:
    * Rotation Matrices need 9 parameters to be stored.
    * The matrices must be orthogonal, and often due to floating point inaccuracy, they do not remain orthogonal, and the elements of the matrix may need to be recomputed.

- Axis Angle Representation
  - Advantages:
    * Very easy to visualize.
    * Can be very easily converted to quaternions.
  - Disadvantages:
    * If $\theta = 0$ then axis is arbitrary.
    * For θ and $\theta + 2 * k * \pi$ produce the same result.

Q2. Let $R^a = (\theta^a_{roll}, \theta^a_{pitch}, \theta^a_{yaw})$ and $R^b = (\theta^b_{roll}, \theta^b_{pitch}, \theta^b_{yaw})$ be a rotation corresponding to the following Roll-Pitch-Yaw angles (ZYX conventions):

$$\theta^a_{roll} = \pi/4 \qquad \theta^b_{roll} = -\pi/3$$

$$\theta^a_{pitch} = 0 \qquad \theta^b_{roll} = 0$$

$$\theta^a_{yaw} = \pi/3 \qquad \theta^b_{roll} = 0$$

Compute 3x3 rotation matrices corresponding to R^a and R^b. Then compare R^aR^b and R^bR^a and give physical meaning to these two.
A2.

$$R^a = \begin{bmatrix} 0.707107 & -0.353553 & 0.612372 \\ 0.707107 & 0.353553 & -0.612372 \\ 0 & 0.866025 & 0.5 \end{bmatrix}$$

$$R^b = \begin{bmatrix} 0.5 & 0.866025 & 0 \\ -0.866025 & 0.5 & -0 \\ -0 & 0 & 1 \end{bmatrix}$$

$$R^aR^b = \begin{bmatrix} 0.65974 & 0.435596 & 0.612372 \\ 0.0473672 & 0.789149 & -0.612372 \\ -0.75 & 0.433013 & 0.5 \end{bmatrix}$$

$$R^bR^a = \begin{bmatrix} 0.965926 & 0.12941 & -0.224144 \\ -0.258819 & 0.482963 & -0.836516 \\ 0 & 0.866025 & 0.5 \end{bmatrix}$$

We note that even though matrices are not commutative in general, they are associative. Let $R^{ab} = R^aR^b$. Clearly R^{ab} is inturn a Rotation matrix. When R^{ab} is applied to some other matrix,i.e. pre-multiplied to a matrix (say $R^{ab} * M$) the multiplication can be viewed in two steps, first multiplication with R^b and then with R^a. Similarly with R^{ba}, the transformation can be viewed as successive rotations.

Since rotations in 3D are in general not commutative, here too, $R^{ab} \neq R^{ba}$.

Q3. Compute the quaternions q^a and q^b equivalent to the matrices R^a and R^b . Are these

quaternions a unique representation of these rotations?
A3.
Representing Quaternions in the form of [w x y z]

$$q^a = \begin{bmatrix}0.800103 & 0.46194 & 0.191342 & 0.331414\end{bmatrix}$$

$$q^b = \begin{bmatrix}0.866025 & 0 & 0 & -0.5\end{bmatrix}$$

For Quaternions we note that q and $-q$ both give the same rotation. Hence these Quaternions are not a unique representation of these rotations. Other than this the relationship is unique.

Q4. Compute two compositions of the rotations using quaternions, $q^c = q^a q^b$ and $q^d = q^b q^a$. Are q^c and q^d the same? Compute the relative rotation between q^a and q^b as $q^e = q^a (q^b)^{-1}$. Then compute the composition $q^f = q^e q^b$. Verify that q^f and q^a are the same.
A4.

$$q^c = \begin{bmatrix}0.858616 & 0.304381 & 0.396677 & -0.113039\end{bmatrix}$$

$$q^d = \begin{bmatrix}0.858616 & 0.495722 & -0.0652631 & -0.113039\end{bmatrix}$$

As expected q^c and q^d are not the same, since rotations in 3d are in general not commutative.

$$q^e = \begin{bmatrix}0.527203 & 0.495722 & -0.0652631 & 0.687064\end{bmatrix}$$

$$q^f = \begin{bmatrix}0.800103 & 0.46194 & 0.191342 & 0.331414\end{bmatrix}$$

As expected $q^f = q^a$.

Q5. How does axis/angle representation relate to quaternion? For example, how to convert between quaternion q = (x, y, z, w) and angle/axis (n,)? Is the conversion unique or not? If not, how many?
A5.
Considering the representation of the quaternion as $q = [v \quad s]$, where v is the complex part and s is the scalar part. Therefore $v = [x \quad y \quad z]$ and $s = w$.

The axis angle and quaternions are closely related. Assumption is that the quaternion is a unit quaternion.

- Quaternion $[xyzw]$ to Angle Axis (n, θ):

$$\begin{aligned} \theta &= 2 * \cos^{-1} w \\ n_x &= \frac{x}{\sqrt{1-w^2}} \\ n_y &= \frac{y}{\sqrt{1-w^2}} \\ n_z &= \frac{z}{\sqrt{1-w^2}} \end{aligned}$$

- Angle Axis to Quaternion:

$$\begin{aligned} w &= \cos\frac{\theta}{2} \\ x &= \sin\frac{\theta}{2} \\ y &= \sin\frac{\theta}{2} \\ z &= \sin\frac{\theta}{2} \end{aligned}$$

Except for the fact that anti-podal Quaternions represent the same rotation there is no other ambiguity, i.e. q and $-q$ represent the same axis angle rotation. But this is to be expected since (n, θ) and $(-n, -\theta)$ also represent the same rotation.

Q6. Assuming Euler angles are small, it is very useful to know the approximation: $R \approx I + [\omega]$, where $[.]_x$ is the skew-symmetric operator and ω is the small rotation. Now can you write down $\frac{\partial Rv}{\partial w}|_{\omega=0}$ where $v \in R^3$
A6.
Given that $R \approx I + [\omega]$ we deduce :

$$Rv \approx v + [\omega]_x v$$

$$Rv \approx v + \omega \times v$$

$$Rv \approx v - v \times \omega$$

$$Rv \approx= v - [v]_x \omega$$

Now we can differentiate easily, using the fact

$$\frac{d(a^T x)}{dx} = a^T$$

Hence we get

$$\frac{\partial Rv}{\partial w}|_{\omega=0} = -[v]_x$$

Problem Statement for Q4 : 3D Perception

Q1. Plane Segmentation Tutorial
A1.
Plane Segmentation is done using SAC segmentation. First the model type is selected as a plane, and then computation method is selected as RANSAC. The code basically chooses points which satisfy the equation of a plane, using RANSAC method. Once it has found the plane with maximum number of inliers,it computes the equation of the plane in the form

$$ax + by + cz + d = 0$$

. It then colors those points, and then removes all those inliers, and repeats the method to get a new set of inliers. This way red, green and blue points are shown.

The problem asked us to use display the normal vector arrows of the plane in the same colors as those belonging to the plane.

I have done this using the Marker Arrow. I have computed the centroid of the points in a particular plane, and chosen this point to show the arrow. I also had the model coefficients for the plane in the form of

$$ax + by + cz + d = 0$$

and hence I knew the direction of the normal is

$$v = (a, b, c)$$

I simply used the Eigen library to get the quaternion form of this using the function FromTwoVectors, which gave me the orientation in the quaternion form, and I used these values to set the orirentation of the marker arrow, and also set the color same as that of the points.

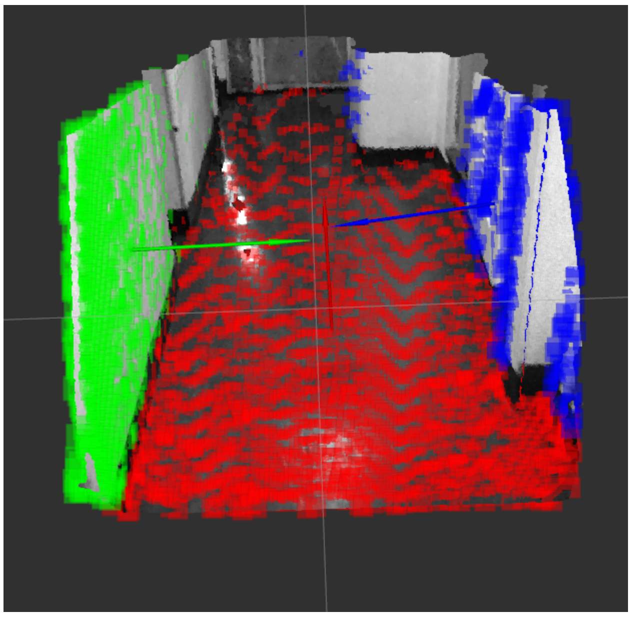

Figure 1: Plane Segmentation Normal Vectors

Q2. 3D reconstruction with Stereo
A2.
This was considerably tougher problem. The code was written following the guidelines provided in the problem statement. In brief the following steps were followed:

- Loading the data from data.json, left_calib.json and right_calib.json to the program. The default function recorded the filenames, and poses, which basically gives us Translation and Rotation in a matrix format. In addition to this, I also loaded the Translation and Rotation (in quaternion form) separately.
-
- For disparity map computation, a class **Disp_map** is constructed, which stores the disparity map, a pointer for the Opencv implementation of Semi Global Block Matching (SGBM). The code for disparity computation is similar to that used in an example code in OpenCV cpp, for stereo matching. The disparity map is stored as a float. The class **Disp_map** also has a funciton to store the disparity map in png format. Object oriented approach is used since it simplifies the code, and different algorithms can be implented.
- After the disparity map is computed, another class **Depth_map** is constructed which stores two point clouds generated from different computations, one using my implementation of reprojection and the other using default opencv implementation of reprojection. Both give similar point cloud results, but opencv implementation gives slightly better results. The class also includes methods to compute the transfomration of a point cloud, and methods to save the point clouds in pcd or ply format.
- Once the point cloud is formed, it is saved in PCD format. The point cloud has several artifacts at places with ambiguous textures.
- Transformation into the same coordinate system is done using the pcl implementation of **tranformPointCloud**. Concatenation of point clouds is simply adding all the transformed point clouds, which is trivial. The resulting point cloud is also saved.

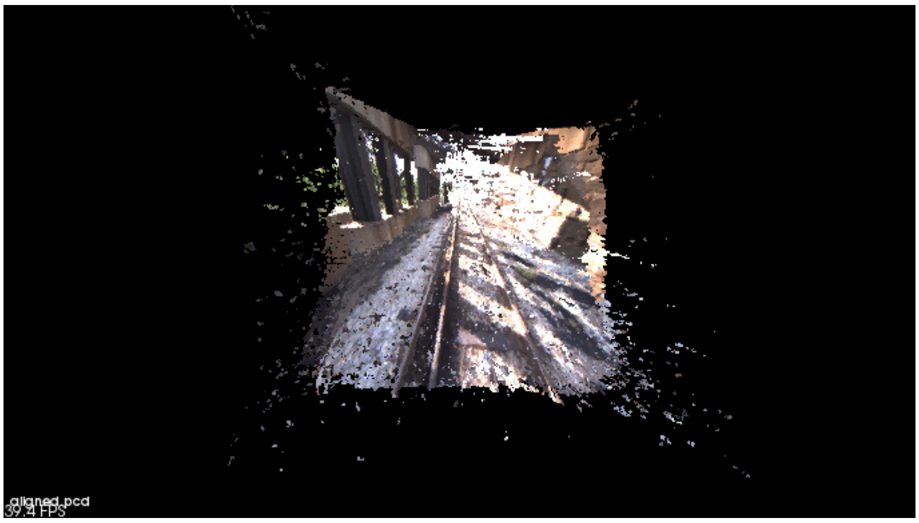

Figure 2: Initial Aligned Result

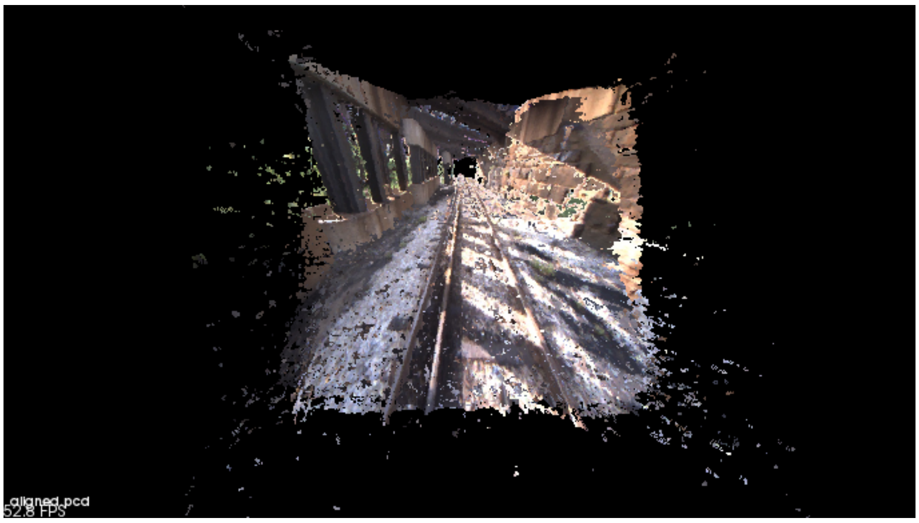

Figure 3: Extra Aligned Result

- There are a few problems with the alignment of the Point Cloud. Some are corrected in the Extra Aligned, but one important problem is not resolved, i.e. exact alignment. I have used the Pose data directly, but it seems that all the point clouds are directly overlapping on each other, which is giving incorrect 3d point cloud. The point clouds are supposed to be more away from each other, which is not happening, and I haven't been able to identify the source of error.

Q3. Extra Credit
A3.
For extra credit the following has been implemented:

- **Preprocessing the images using illumination equalization**.
  This is done using CLAHE method in opencv. The disparity map turned out to be slightly better but the improvement is very nominal. Laplacian of Gaussian was also tried, but this result in a worse disparity map possibly because the disparity map also got blurred because of the gaussian.

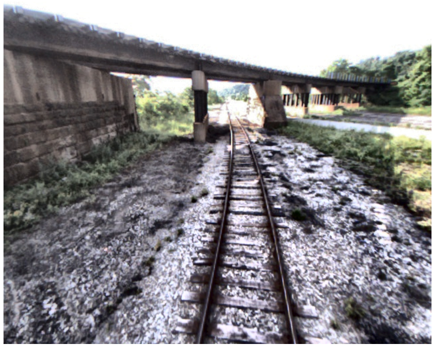

Figure 4: Illumination Equalized using Clahe

- **Removal of Background Noise in Depth Map**. There were some pixels with very small value of disparity. These constituted parts of the image where disparity map gave absurd results. To remove these, the initial disparity map is multiplied by a factor(25 in the code). This new disparity map is then passed for reprojection. Now we consider only points with $z > 0$. Other points have bad disparity and hence discarded.

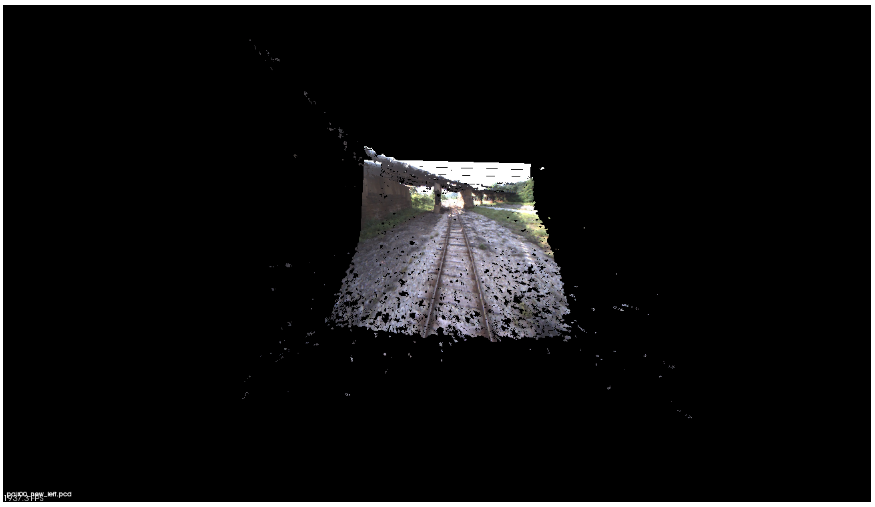

Figure 5: Initial pair00 Point Cloud

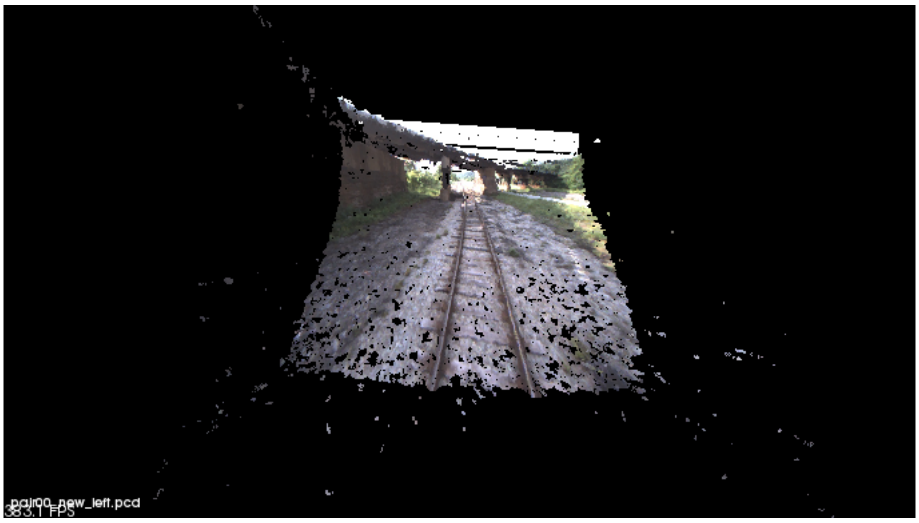

Figure 6: Background noise removed

- **Post processing. Removal of noise in disparity map due to sky regions.** This is done using a rather simple algorithm. We consider an element to be a sky element if it is present in the upper half of the image, and its blue channel color is greater than a certain threshold. The threshold is chosen to be $0.8 * gmax$ where $gmax = max(max(R), max(B), max(G))$. For such pixels the disparity is made 0. This gave suprisingly very good results.

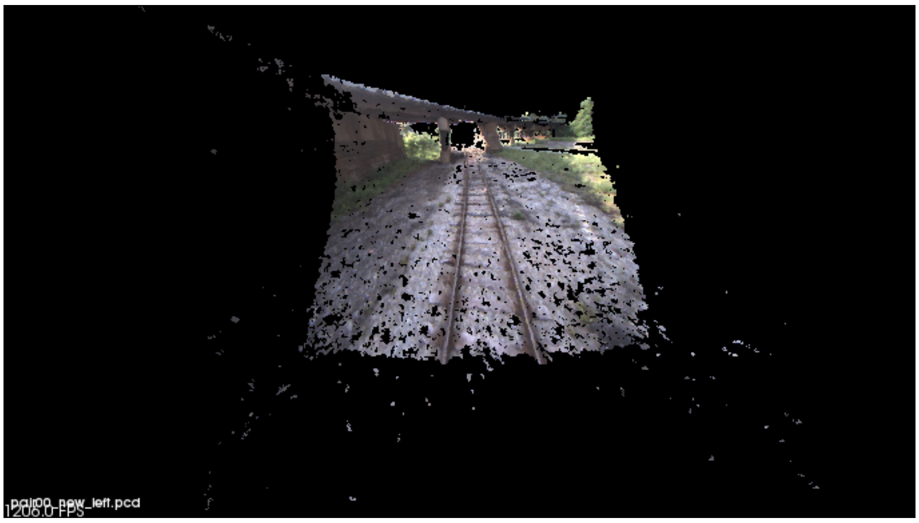

Figure 7: Sky Noise Removed

- **Attempt at census transform**. OpenCV implementation of disparity Computation using census transform resulted in error due to some Computation problem in hamming distance, and was not resolved.

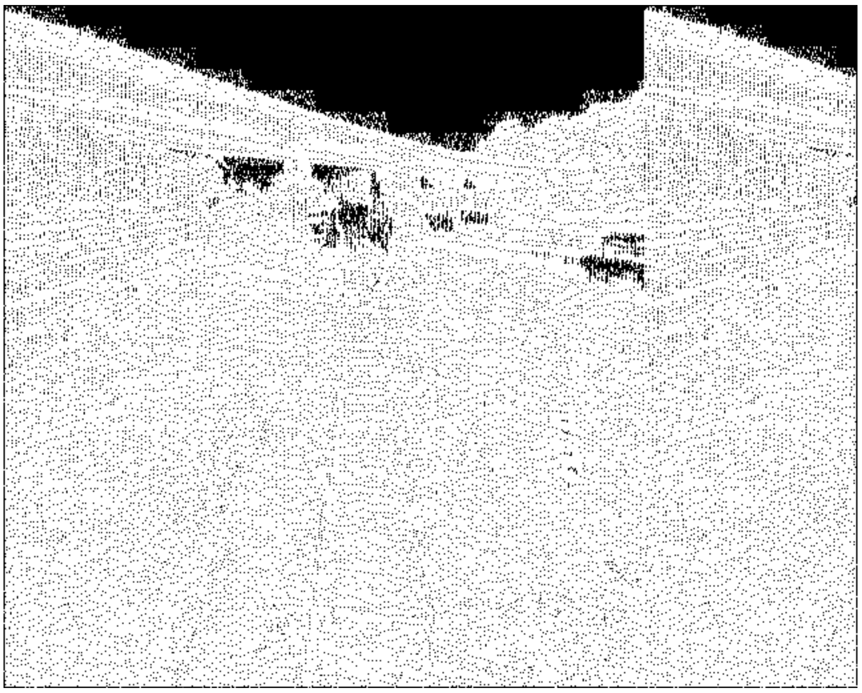

Figure 8: Census Transform

- **Attempt at post processing using WLS Filtering**. The OpenCV implementation of WLS filter in ximgproc only accepts CV_16S. This leads to loss of info in the disparity map at the cost of better filtering. Overall the disparity map was worse. The primary reason for this is conversion to CV_16S resulted in loss of information.

Figure 9: Actual Disparity Map

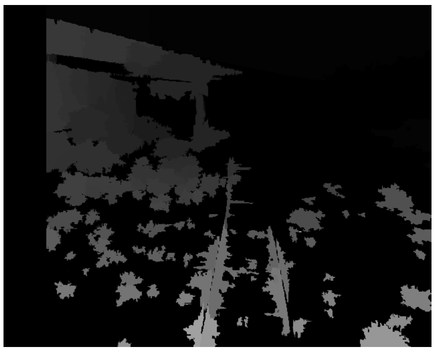

Figure 10: Post Filtered Image (WLS filter)

- **Interpolation using Inpainting**. A mask for image was created denoting good and bad pixels in the disparity map. But even then the disparity map was nearly the same. Not very sure why this didn't work.
- **Attempt at Alignment using Iterated Closest Point (ICP)**. There is some error in the code, since the euclidean fitness score is in the range of 1e6. Since this too takes up a lot of time, this has been commented.

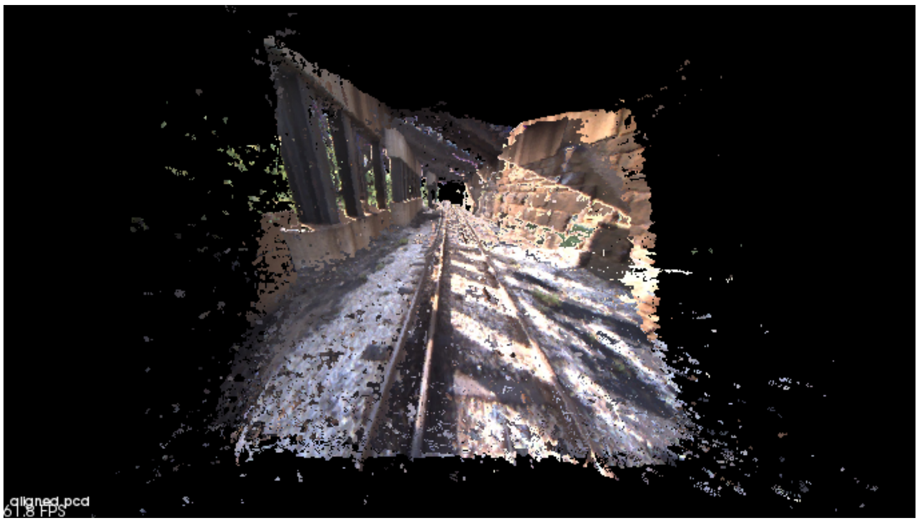

Figure 11: ICP Extra Aligned