Homework 4

- SIFT feature extraction
- SIFT feature matching
- Fundamental matrix
- RANSAC

SIFT Feature Extraction

(SIFT visualization) Use `VLFeat` to visualize SIFT features with scale and orientation as shown in Figure 1. You may want to plot up to 500 feature points. You may want to follow the following tutorial:
`http://www.vlfeat.org/overview/sift.html`

```
% load image
I = imread('left.bmp');
I = single(rgb2gray(I));

% run SIFT
[f,d] = vl_sift(I);
```

SIFT Feature Extraction

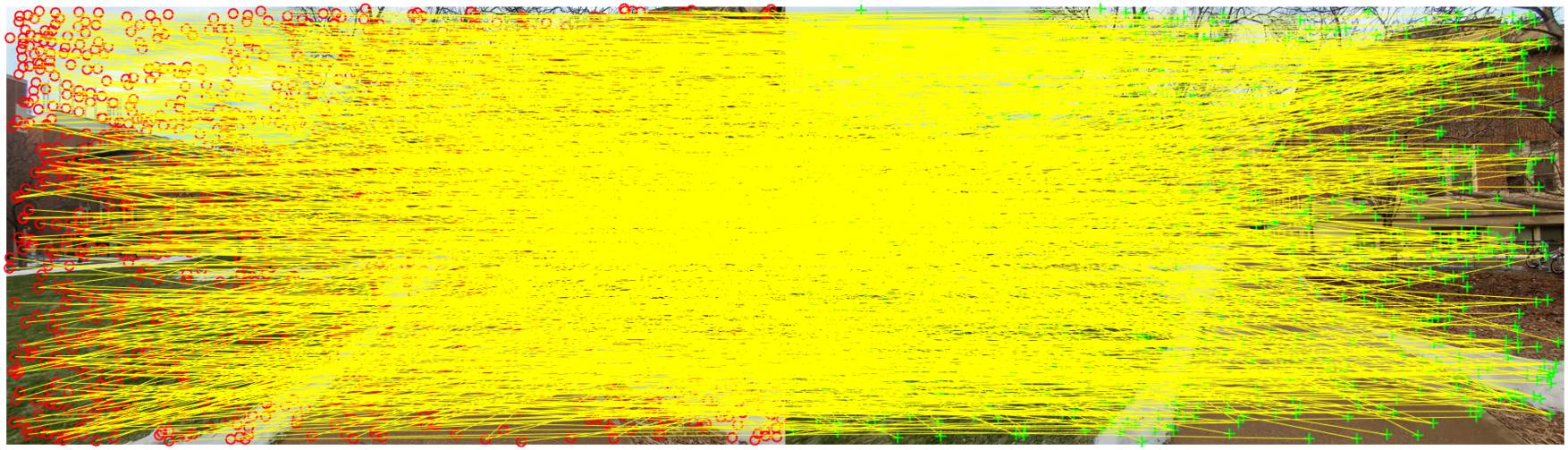
(SIFT visualization) Use `VLFeat` to visualize SIFT features with scale and orientation as shown in Figure 1. You may want to plot up to 500 feature points. You may want to follow the following tutorial:
`http://www.vlfeat.org/overview/sift.html`

```
% load image
I = imread('left.bmp');
I = single(rgb2gray(I));

% run SIFT
[f,d] = vl_sift(I);

% visualize
figure, imshow('left.bmp'), hold on
perm = randperm(size(f,2));
sel  = perm(1:500);
h1   = vl_plotframe(f(:,sel)) ; set(h1,'color','k','linewidth',3);
h2   = vl_plotframe(f(:,sel)) ; set(h2,'color','y','linewidth',2);
```

SIFT Feature Matching

(1) (Nearest neighbor search) Let two sets of features be $\{\mathbf{f}_1, \cdots, \mathbf{f}_{N_1}\}$ from $\mathcal{I}_1$ and $\{\mathbf{g}_1, \cdots, \mathbf{g}_{N_2}\}$ from $\mathcal{I}_2$ where N_1 and N_2 are the number of features in image 1 and 2, respectively. Compute nearest neighbor per feature and visualize ($\{\mathbf{f}_1, \cdots, \mathbf{f}_{N_1}\} \rightarrow \{\mathbf{g}_1, \cdots, \mathbf{g}_{N_2}\}$ and $\{\mathbf{g}_1, \cdots, \mathbf{g}_{N_2}\} \rightarrow \{\mathbf{f}_1, \cdots, \mathbf{f}_{N_1}\}$) as shown in Figure 2(a) and Figure 2(b). Note that the distance between two features is defined as $d = \|\mathbf{f} - \mathbf{g}\|$. You may use `knnsearch` function in MATLAB.

```
% nearest neighbor
[idx12, dist] = knnsearch(dg', df', 'K', 2);

% visualize
im1 = imread('left.bmp');
im2 = imread('right.bmp');
im1 = imresize(im1, 0.5);
im2 = imresize(im2, 0.5);
figure, showMatchedFeatures(im1,im2,f(1:2,:)',g(1:2,idx12(:,1))','montage');
```

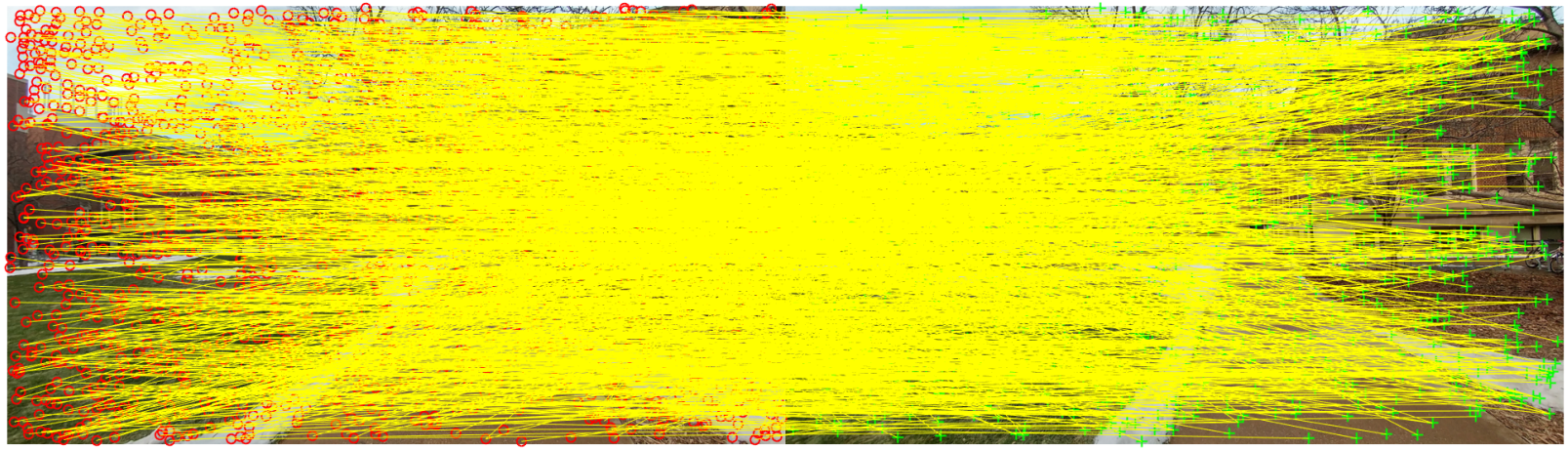


SIFT Feature Matching

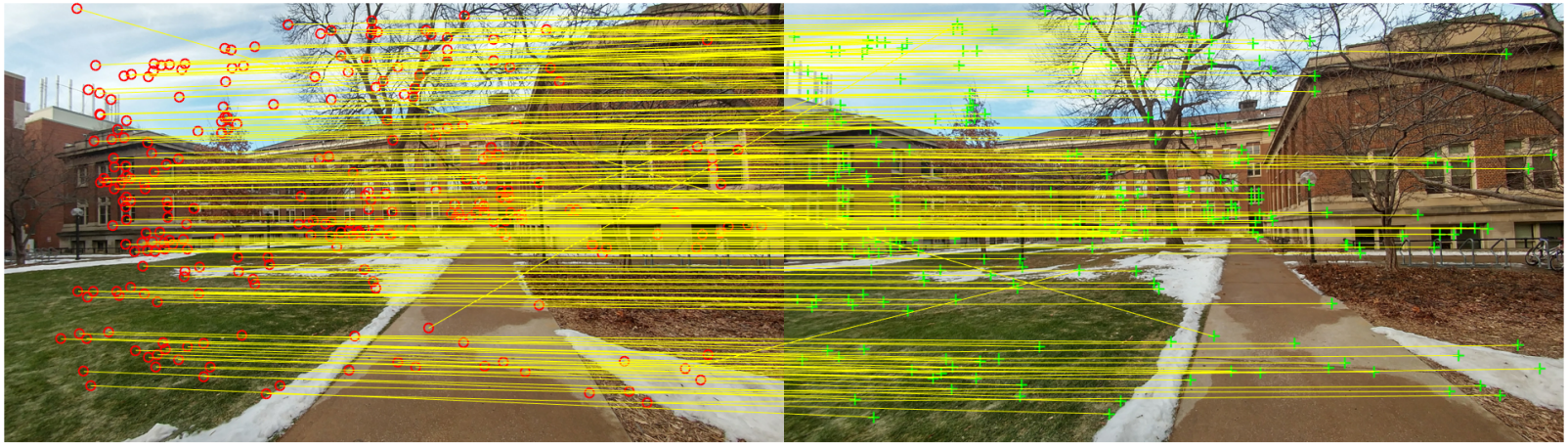
(2) (Ratio test) Filter out matches using the ratio test, i.e., keep the match if $d_{ij_1}/d_{ij_2} < 0.7$ and discard otherwise, where d_{ij_1} and d_{ij_2} are the first and second nearest neighbors for the i^{th} feature, respectively. Visualize the matches after the ratio test as shown Figure 2(d) and Figure 2(c).

```
% nearest neighbor
[idx12, dist] = knnsearch(dg', df', 'K', 2);

% ratio test
id = find(dist(:,1)./dist(:,2) < 0.7);

% visualize
im1 = imread('left.bmp');
im2 = imread('right.bmp');
im1 = imresize(im1, 0.5);
im2 = imresize(im2, 0.5);
figure, showMatchedFeatures(im1,im2,f(1:2,id)',g(1:2,idx12(id,1))','montage');
```

Lowe, David G. "Distinctive image features from scale-invariant keypoints." *International journal of computer vision* 60.2 (2004): 91-110.



Before ratio test (#pairs = 2076)



Before ratio test (#pairs = 299)

SIFT Feature Matching

(3) (Bidirectional match) Visualize bidirectionally consistent matches as shown Figure 2(e). Compare the number of matches from (1) to (3).

```
% match from image1 to image2
[idx12, dist] = knnsearch(dg', df', 'K', 2);
id1 = find(dist(:,1)./dist(:,2) < 0.7);

% match from image2 to image1
[idx21, dist] = knnsearch(df', dg', 'K', 2);
id2 = find(dist(:,1)./dist(:,2) < 0.7);
```

```
% bidirectional match
idx = 1 : size(idx12, 1);
bi = find(idx21(idx12(id1,1), 1) == idx(id1)'); % bidirectional check
[~, ia] = intersect(idx12(id1(bi),1), id2); % image2 to image1 ratio check
idx_bi = bi(ia);
figure, showMatchedFeatures(im1,im2, ...
        f(1:2,id1(idx_bi))',g(1:2,idx12(id1(idx_bi),1))','montage');
```

SIFT Feature Matching

(3) (Bidirectional match) Visualize bidirectionally consistent matches as shown Figure 2(e). Compare the number of matches from (1) to (3).

```
% match from image1 to image2
[idx12, dist] = knnsearch(dg', df', 'K', 2);
id1 = find(dist(:,1)./dist(:,2) < 0.7);

% match from image2 to image1
[idx21, dist] = knnsearch(df', dg', 'K', 2);
id2 = find(dist(:,1)./dist(:,2) < 0.7);

% bidirectional match
idx = 1 : size(idx12, 1);
bi = find(idx21(idx12(id1,1), 1) == idx(id1)'); % bidirectional check
[~, ia] = intersect(idx12(id1(bi),1), id2); % image2 to image1 ratio check
idx_bi = bi(ia);
figure, showMatchedFeatures(im1,im2, ...
        f(1:2,id1(idx_bi))',g(1:2,idx12(id1(idx_bi),1))','montage');
```

SIFT Feature Matching

(3) (Bidirectional match) Visualize bidirectionally consistent matches as shown Figure 2(e). Compare the number of matches from (1) to (3).

```
% match from image1 to image2
[idx12, dist] = knnsearch(dg', df', 'K', 2);
id1 = find(dist(:,1)./dist(:,2) < 0.7);

% match from image2 to image1
[idx21, dist] = knnsearch(df', dg', 'K', 2);
id2 = find(dist(:,1)./dist(:,2) < 0.7);

% bidirectional match
idx = 1 : size(idx12, 1);
bi = find(idx21(idx12(id1,1), 1) == idx(id1)'); % bidirectional check
[~, ia] = intersect(idx12(id1(bi),1), id2); % image2 to image1 ratio check
idx_bi = bi(ia);
figure, showMatchedFeatures(im1,im2, ...
        f(1:2,id1(idx_bi))',g(1:2,idx12(id1(idx_bi),1))','montage');
```

SIFT Feature Matching

(3) (Bidirectional match) Visualize bidirectionally consistent matches as shown Figure 2(e). Compare the number of matches from (1) to (3).

```
% match from image1 to image2
[idx12, dist] = knnsearch(dg', df', 'K', 2);
id1 = find(dist(:,1)./dist(:,2) < 0.7);

% match from image2 to image1
[idx21, dist] = knnsearch(df', dg', 'K', 2);
id2 = find(dist(:,1)./dist(:,2) < 0.7);

% bidirectional match
idx = 1 : size(idx12, 1);
bi = find(idx21(idx12(id1,1), 1) == idx(id1)'); % bidirectional check
[~, ia] = intersect(idx12(id1(bi),1), id2); % image2 to image1 ratio check
idx_bi = bi(ia);
figure, showMatchedFeatures(im1,im2, ...
        f(1:2,id1(idx_bi))',g(1:2,idx12(id1(idx_bi),1))','montage');
```

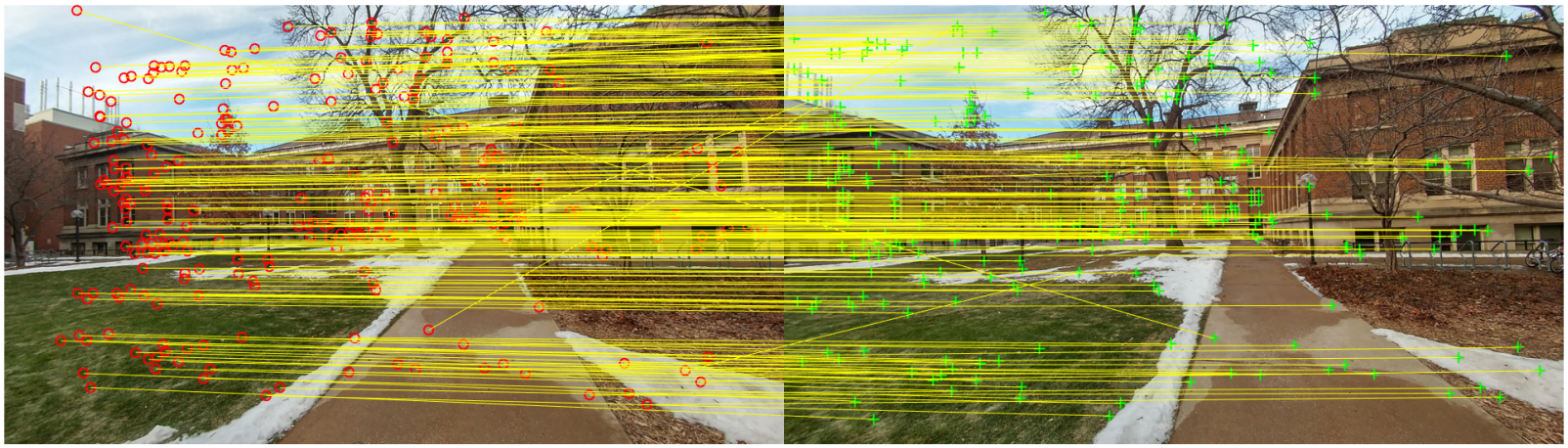
$\mathbf{P_1} == \mathbf{P_3}$?

SIFT Feature Matching

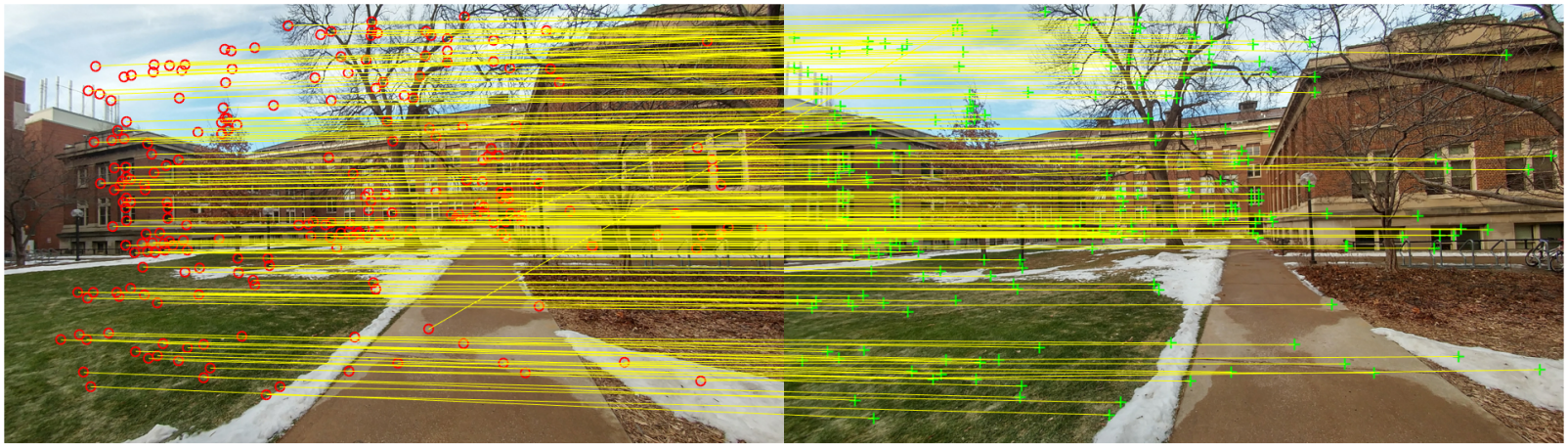
(3) (Bidirectional match) Visualize bidirectionally consistent matches as shown Figure 2(e). Compare the number of matches from (1) to (3).

```
% match from image1 to image2
[idx12, dist] = knnsearch(dg', df', 'K', 2);
id1 = find(dist(:,1)./dist(:,2) < 0.7);

% match from image2 to image1
[idx21, dist] = knnsearch(df', dg', 'K', 2);
id2 = find(dist(:,1)./dist(:,2) < 0.7);

% bidirectional match
idx = 1 : size(idx12, 1);
bi = find(idx21(idx12(id1,1), 1) == idx(id1)'); % bidirectional check
[~, ia] = intersect(idx12(id1(bi),1), id2); % image2 to image1 ratio check
idx_bi = bi(ia);
figure, showMatchedFeatures(im1,im2, ...
        f(1:2,id1(idx_bi))',g(1:2,idx12(id1(idx_bi),1))','montage');
```

P_2 passes ratio test?



Before bidirectional matching (#pairs = 299)

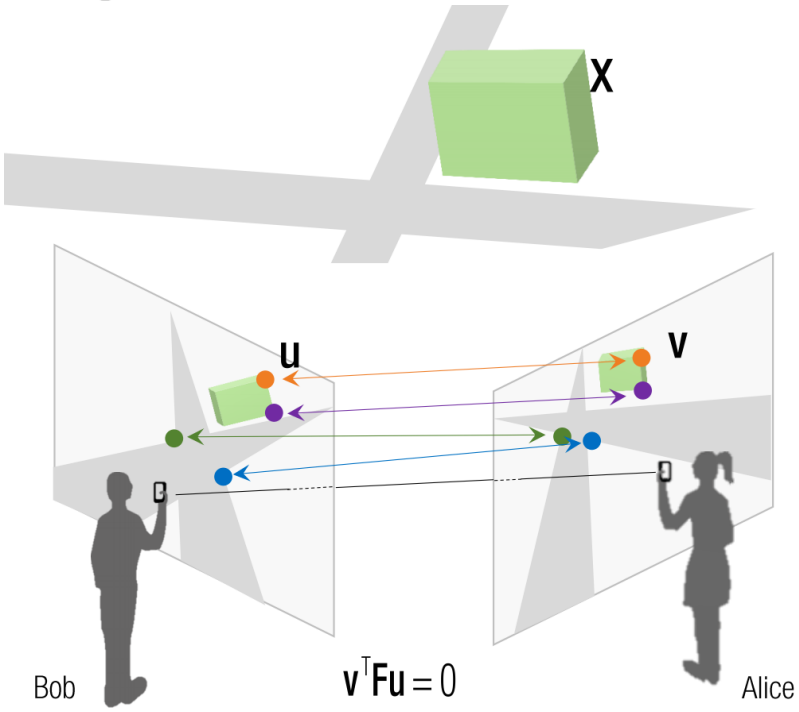


After bidirectional matching (#pairs = 260)

Fundamental Matrix

(1) (Fundamental matrix) Complete the following function to compute a fundamental matrix, linearly:

F = `ComputeFundamentalMatrix(u, v)`



$$\mathbf{v}^T\mathbf{F}\mathbf{u} = \begin{bmatrix} v^x & v^y & 1 \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{bmatrix} u^x \\ u^y \\ 1 \end{bmatrix}$$

$$= f_{11}u^xv^x + f_{12}u^yv^x + f_{13}v^x + f_{21}u^xv^y + f_{22}u^yv^y + f_{23}v^y + f_{31}u^x + f_{32}u^y + f_{33}$$

$$= 0$$

Linear in **F**.

$$\begin{bmatrix} u_1^xv_1^x & u_1^yv_1^x & v_1^x & u_1^xv_1^y & u_1^yv_1^y & v_1^y & u_1^x & u_1^y & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_m^xv_m^x & u_m^yv_m^x & v_m^x & u_m^xv_m^y & u_m^yv_m^y & v_m^y & u_m^x & u_m^y & 1 \end{bmatrix} \mathbf{A} \quad \mathbf{X} \quad \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{bmatrix} = \mathbf{0}$$

What is minimum m?

Fundamental Matrix

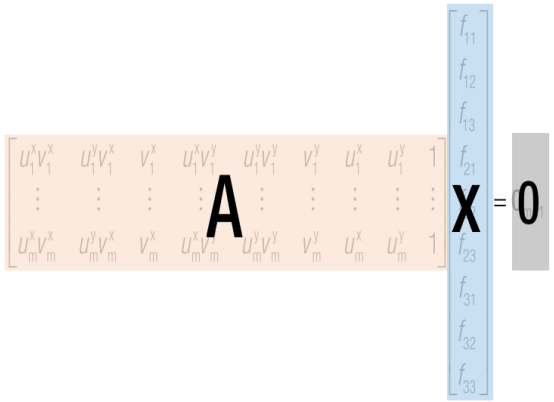
(1) (Fundamental matrix) Complete the following function to compute a fundamental matrix, linearly:

F = ComputeFundamentalMatrix(**u**, **v**)

```
function F = ComputeFundamentalMatrix(x1, x2)
A = [];
for i = 1 : size(x1,1)
    A =[A; x1(i,1)*x2(i,1) x1(i,2)*x2(i,1) x2(i,1) ...
        x1(i,1)*x2(i,2) x1(i,2)*x2(i,2) x2(i,2) x1(i,1) x1(i,2) 1];
end
```

construct A

```
% solve F
[u, d, v] = svd(A);
f = v(:,end);

% rectify F
F = [f(1:3)'; f(4:6)'; f(7:9)'];
[u, d, v] = svd(F);
d(3,3) = 0;
F = u*d*v';
```



Fundamental Matrix

(1) (Fundamental matrix) Complete the following function to compute a fundamental matrix, linearly:

F = ComputeFundamentalMatrix(**u**, **v**)

```
function F = ComputeFundamentalMatrix(x1, x2)
A = [];
for i = 1 : size(x1,1)
    A =[A; x1(i,1)*x2(i,1) x1(i,2)*x2(i,1) x2(i,1) ...
        x1(i,1)*x2(i,2) x1(i,2)*x2(i,2) x2(i,2) x1(i,1) x1(i,2) 1];
end
```

```
% solve F
[u, d, v] = svd(A);
f = v(:,end);
```

solve F

```
% rectify F
F = [f(1:3)'; f(4:6)'; f(7:9)'];
[u, d, v] = svd(F);
d(3,3) = 0;
F = u*d*v';
```

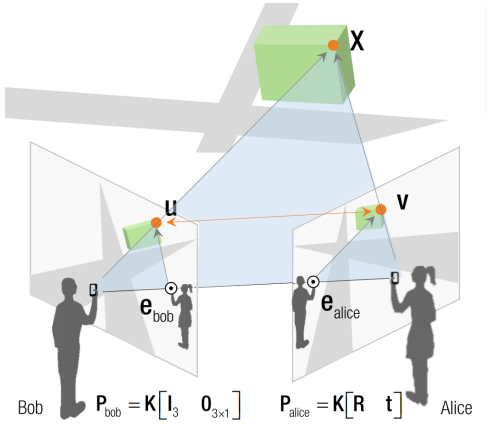
rectify F

$$\begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} = U \; D \; V^{\top}$$

Fundamental Matrix

(2) (Epipole and epipolar line) Pick 8 random correspondences, $\mathbf{u}_r \leftrightarrow \mathbf{v}_r$, compute the fundamental matrix, $\mathbf{F}_r$ and visualize epipole and epipolar lines for the rest of feature points in both images as shown in Figure 3. Pick different sets of correspondences and visualize different epipolar lines.

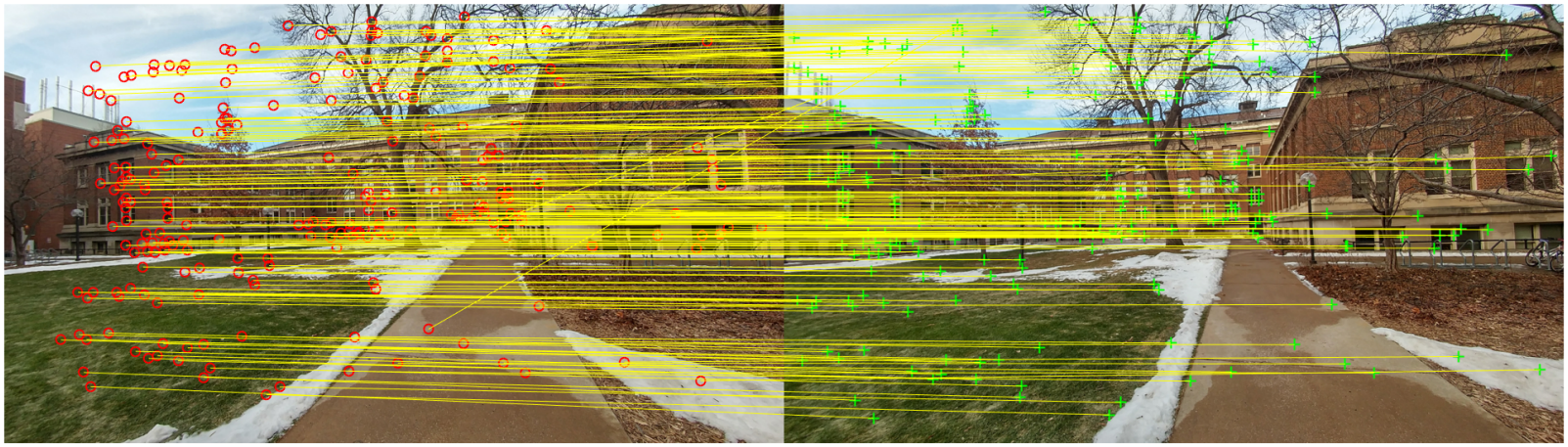
```
l = F'*[v(j,:)';1];
u0 = 0;
u1 = size(im1,2);
v0 = (-l(1)*u0-l(3))/l(2);
v1 = (-l(1)*u1-l(3))/l(2);
hold on
plot([u0 u1], [v0 v1], 'r-', 'LineWidth', 1);
plot(u(j,1), u(j,2), 'go', 'MarkerFaceColor', 'g');
```



- Epipolar line: $\mathbf{l}_u = \mathbf{F}\mathbf{u}$ $\mathbf{l}_v = \mathbf{F}^T\mathbf{v}$

Robust Fundamental Matrix Estimation

(1) (RANSAC with fundamental matrix) Write a RANSAC algorithm for the fundamental matrix estimation given N matches from Section 3 using the following pseudo code:



Robust Fundamental Matrix Estimation

```
max_inlier = 0;
for i = 1 : 50
    rand_idx = randperm(length(id_new));
    x1 = f1(1:2,id_new(rand_idx(1:8)))';
    x2 = f2(1:2,idx12(id_new(rand_idx(1:8)),1))';
    F_r = ComputeFundamentalMatrix(x1, x2);

    u = f1(1:2,id_new)';
    v = f2(1:2,idx12(id_new,1))';

    nInlier = 0;
    for j = 1 : length(id_new)
        e = abs([v(j,:) 1] * F_r * [u(j,:)';1]) ...
            / norm(F_r(1:2,:) * [u(j,:)';1]);
        if (e < 1)
            nInlier = nInlier + 1;
        end
    end

    if max_inlier < nInlier
        max_inlier = nInlier;
        F = F_r;
    end

end
```

Algorithm 1 GetInliersRANSAC

1: $n \leftarrow 0$
2: **for** $i = 1 : M$ **do**
3: Choose 8 correspondences, $\mathbf{u}_r$ and $\mathbf{v}_r$, randomly from $\mathbf{u}$ and $\mathbf{v}$.
4: $\mathbf{F}_r$ = `ComputeFundamentalMatrix(`$\mathbf{u}_r$`,` $\mathbf{v}_r$`)`
5: Compute the number of inliers, n_r, with respect to `F`.
6: **if** $n_r > n$ **then**
7: $n \leftarrow n_r$
8: $\mathbf{F} = \mathbf{F}_r$
9: **end if**
10: **end for**

Robust Fundamental Matrix Estimation

```
max_inlier = 0;
for i = 1 : 50
    rand_idx = randperm(length(id_new));
    x1 = f1(1:2,id_new(rand_idx(1:8)))';
    x2 = f2(1:2,idx12(id_new(rand_idx(1:8)),1))';
    F_r = ComputeFundamentalMatrix(x1, x2);

    u = f1(1:2,id_new)';
    v = f2(1:2,idx12(id_new,1))';

    nInlier = 0;
    for j = 1 : length(id_new)
        e = abs([v(j,:) 1] * F_r * [u(j,:)';1]) ...
            / norm(F_r(1:2,:) * [u(j,:)';1]);
        if (e < 1)
            nInlier = nInlier + 1;
        end
    end

    if max_inlier < nInlier
        max_inlier = nInlier;
        F = F_r;
    end

end
```

Algorithm 1 GetInliersRANSAC

1: $n \leftarrow 0$
2: **for** $i = 1 : M$ **do**
3: Choose 8 correspondences, $\mathbf{u}_r$ and $\mathbf{v}_r$, randomly from $\mathbf{u}$ and $\mathbf{v}$.
4: $\mathbf{F}_r$ = `ComputeFundamentalMatrix(`$\mathbf{u}_r$, $\mathbf{v}_r$`)`
5: Compute the number of inliers, n_r, with respect to `F`.
6: **if** $n_r > n$ **then**
7: $n \leftarrow n_r$
8: $\mathbf{F} = \mathbf{F}_r$
9: **end if**
10: **end for**

Robust Fundamental Matrix Estimation

```
max_inlier = 0;
for i = 1 : 50
    rand_idx = randperm(length(id_new));
    x1 = f1(1:2,id_new(rand_idx(1:8)))';
    x2 = f2(1:2,idx12(id_new(rand_idx(1:8)),1))';
    F_r = ComputeFundamentalMatrix(x1, x2);

    u = f1(1:2,id_new)';
    v = f2(1:2,idx12(id_new,1))';

    nInlier = 0;
    for j = 1 : length(id_new)
        e = abs([v(j,:) 1] * F_r * [u(j,:)';1]) ...
            / norm(F_r(1:2,:) * [u(j,:)';1]);
        if (e < 1)
            nInlier = nInlier + 1;
        end
    end

    if max_inlier < nInlier
        max_inlier = nInlier;
        F = F_r;
    end

end
```

Algorithm 1 GetInliersRANSAC

$n \leftarrow 0$
for $i = 1 : M$ **do**
  Choose 8 correspondences, $\mathbf{u}_r$ and $\mathbf{v}_r$, randomly from $\mathbf{u}$ and $\mathbf{v}$.
  $\mathbf{F}_r$ = `ComputeFundamentalMatrix(`$\mathbf{u}_r$, $\mathbf{v}_r$`)`
  Compute the number of inliers, n_r, with respect to `F`.
  if $n_r > n$ **then**
    $n \leftarrow n_r$
    $\mathbf{F} = \mathbf{F}_r$
  end if
end for

Epipolar line: $l_u = Fu$

Distance: $d = \frac{|au_x + bu_y + c|}{\sqrt{a^2 + b^2}} = \frac{|Fu|}{\sqrt{(F_{1,}u)^2 + (F_{2,}u)^2}}$

Robust Fundamental Matrix Estimation

```
max_inlier = 0;
for i = 1 : 50
    rand_idx = randperm(length(id_new));
    x1 = f1(1:2,id_new(rand_idx(1:8)))';
    x2 = f2(1:2,idx12(id_new(rand_idx(1:8)),1))';
    F_r = ComputeFundamentalMatrix(x1, x2);

    u = f1(1:2,id_new)';
    v = f2(1:2,idx12(id_new,1))';

    nInlier = 0;
    for j = 1 : length(id_new)
        e = abs([v(j,:) 1] * F_r * [u(j,:)';1]) ...
            / norm(F_r(1:2,:) * [u(j,:)';1]);
        if (e < 1)
            nInlier = nInlier + 1;
        end
    end

    if max_inlier < nInlier
        max_inlier = nInlier;
        F = F_r;
    end

end
```

Algorithm 1 GetInliersRANSAC

1: $n \leftarrow 0$
2: **for** $i = 1 : M$ **do**
3: Choose 8 correspondences, $\mathbf{u}_r$ and $\mathbf{v}_r$, randomly from $\mathbf{u}$ and $\mathbf{v}$.
4: $\mathbf{F}_r$ = `ComputeFundamentalMatrix(`$\mathbf{u}_r$, $\mathbf{v}_r$`)`
5: Compute the number of inliers, n_r, with respect to `F`.
6: **if** $n_r > n$ **then**
7: $n \leftarrow n_r$
8: $\mathbf{F} = \mathbf{F}_r$
9: **end if**
10: **end for**

Robust Fundamental Matrix Estimation

(3) (Camera pose estimation) Compute 4 configurations of relative camera poses:

`[R1 C1 R2 C2 R3 C3 R4 C4] = CameraPose(F, K)`

$$E = [t]_\times R = U \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} U^T R = U \begin{bmatrix} 1 & & \\ & 1 & \\ & & 0 \end{bmatrix} V^T$$

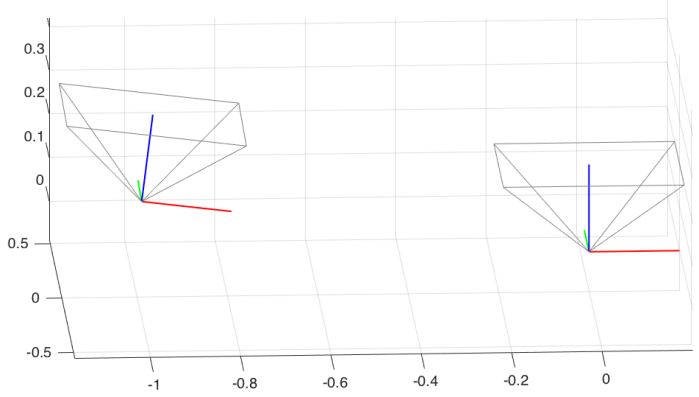
$$t = \pm u_3$$

$$R = U \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} V^T, \text{ or } U \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} V^T$$

```
% solve and rectify E
E = K'*F*K;
[U, D, V] = svd(E);

W = [0 -1 0;
     1 0 0;
     0 0 1];
```

```
% pose 1
t1 = U(:,3);
R1 = U * W * V';
if det(R1) < 0
  t1 = -t1;  R1 = -R1;
end
```

→ Four configurations



Robust Fundamental Matrix Estimation

(3) (Camera pose estimation) Compute 4 configurations of relative camera poses:

`[R1 C1 R2 C2 R3 C3 R4 C4] = CameraPose(F, K)`

$$E = [t]_\times R = U \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} U^T R = U \begin{bmatrix} 1 & & \\ & 1 & \\ & & 0 \end{bmatrix} V^T$$

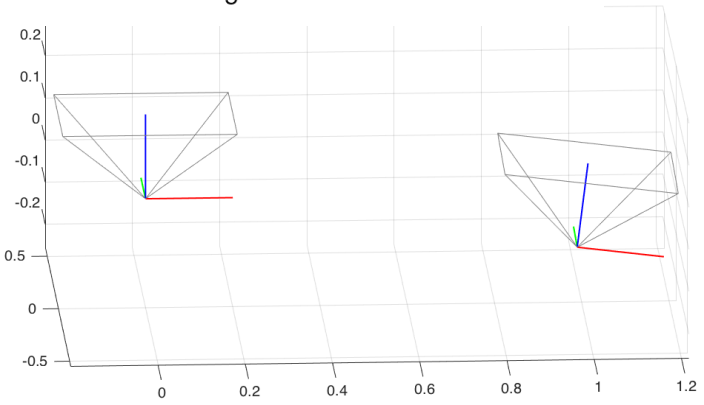
$$t = \pm u_3$$

$$R = U \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} V^T, \text{ or } U \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} V^T$$

```
% solve and rectify E
E = K'*F*K;
[U, D, V] = svd(E);

W = [0 -1 0;
     1 0 0;
     0 0 1];
```

```
% pose 2
t2 = -U(:,3);
R2 = U * W * V';
if det(R2) < 0
  t2 = -t2;  R2 = -R2;
end
```

→ Four configurations



Robust Fundamental Matrix Estimation

(3) (Camera pose estimation) Compute 4 configurations of relative camera poses:

`[R1 C1 R2 C2 R3 C3 R4 C4] = CameraPose(F, K)`

$$E = [t]_\times R = U \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} U^T R = U \begin{bmatrix} 1 & & \\ & 1 & \\ & & 0 \end{bmatrix} V^T$$

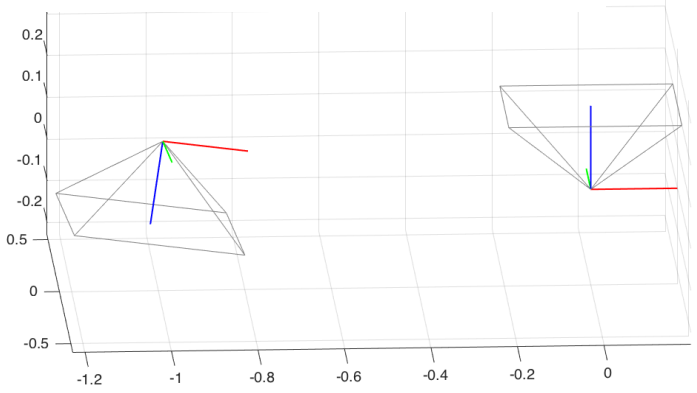
$$t = \pm u_3$$

$$R = U \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} V^T, \text{ or } U \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} V^T$$

```
% solve and rectify E
E = K'*F*K;
[U, D, V] = svd(E);

W = [0 -1 0;
     1 0 0;
     0 0 1];
```

```
% pose 3
t3 = U(:,3);
R3 = U * W' * V';
if det(R3) < 0
  t3 = -t3;  R3 = -R3;
end
```

→ Four configurations



Robust Fundamental Matrix Estimation

(3) (Camera pose estimation) Compute 4 configurations of relative camera poses:

`[R1 C1 R2 C2 R3 C3 R4 C4] = CameraPose(F, K)`

$$E = [t]_\times R = U \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} U^T R = U \begin{bmatrix} 1 & & \\ & 1 & \\ & & 0 \end{bmatrix} V^T$$

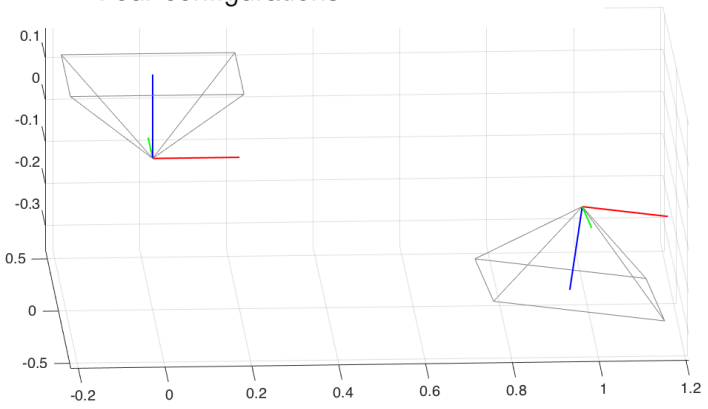
$$t = \pm u_3$$

$$R = U \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} V^T \text{, or } U \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} V^T$$

```
% solve and rectify E
E = K'*F*K;
[U, D, V] = svd(E);

W = [0 -1 0;
     1 0 0;
     0 0 1];
```

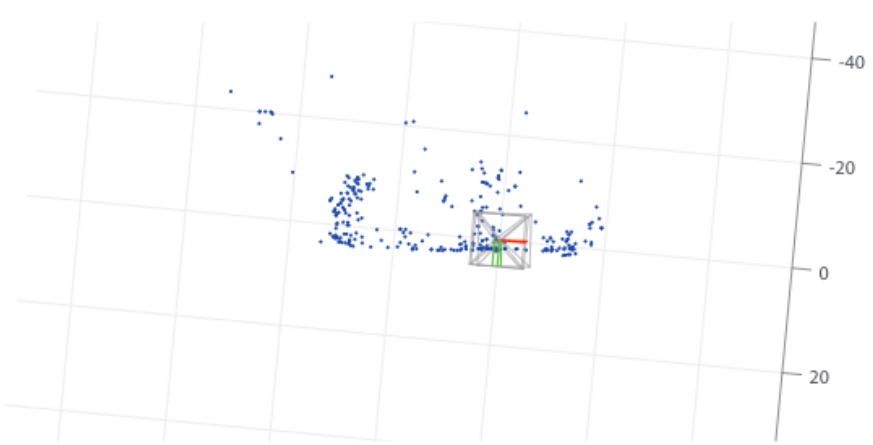
```
% pose 4
t4 = -U(:,3);
R4 = U * W' * V';
if det(R4) < 0
  t4 = -t4;  R4 = -R4;
end
```

→ Four configurations

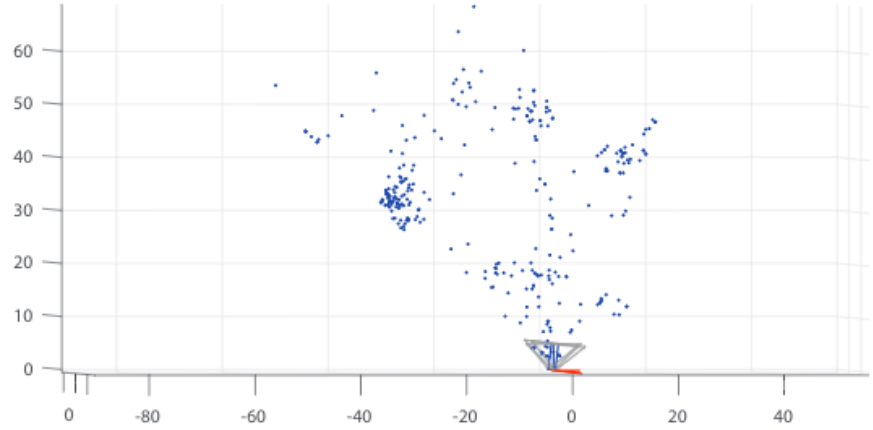


Homework 5

- Triangulation
- Stereo
- PnP



(a) Front view



(b) Top view

Data Capture and Pose Estimation

(1) Visualize epipolar lines after fundamental matrix computation via RANSAC.

(2) Visualize four configurations in 3D.

(a) Image 1, $\mathcal{I}_1$

(b) Image 2, $\mathcal{I}_2$

(c) Image 3, $\mathcal{I}_3$

Triangulation

(1) (Linear triangulation) Write a code that computes the 3D point given the correspondence, $\mathbf{u} \leftrightarrow \mathbf{v}$, and two camera projection matrices:

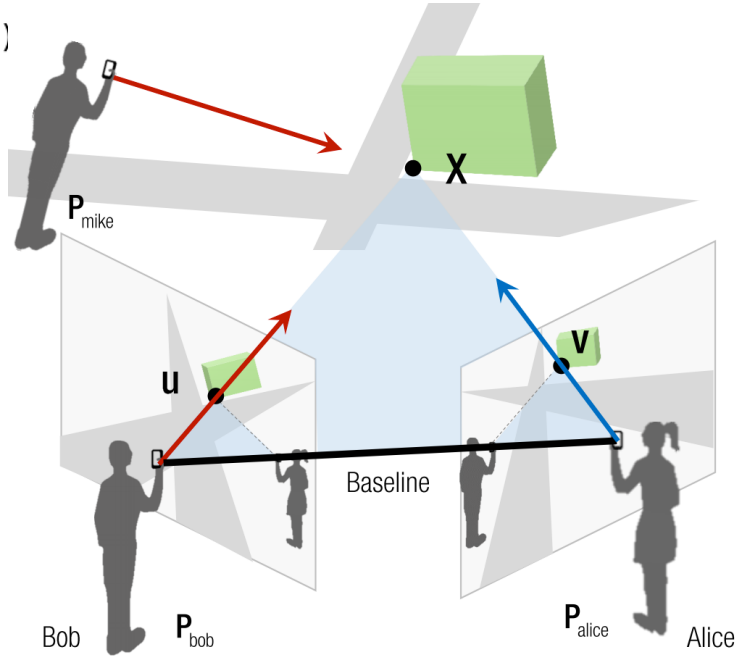
$[\mathbf{X}]$ = LinearTriangulation($\mathbf{P}_1, \mathbf{u}, \mathbf{P}_2, \mathbf{v}$)

$$\begin{bmatrix} \begin{bmatrix} \mathbf{u} \\ 1 \end{bmatrix}_\times \mathbf{P}_1 \\ \begin{bmatrix} \mathbf{v} \\ 1 \end{bmatrix}_\times \mathbf{P}_2 \end{bmatrix} \begin{bmatrix} \mathbf{X} \\ 1 \end{bmatrix} = \mathbf{0}$$

```
function skew = Vec2Skew(v)
skew = [0 -v(3) v(2);
        v(3) 0 -v(1);
        -v(2) v(1) 0];
```

[U,D,V] = svd(A)

X = V(1:3,end) / V(end,end)



Triangulation

(2) (Cheirality) Write a code that computes the number of 3D points in front of two cameras. The condition of a 3D point being in front of camera is called *cheirality*:

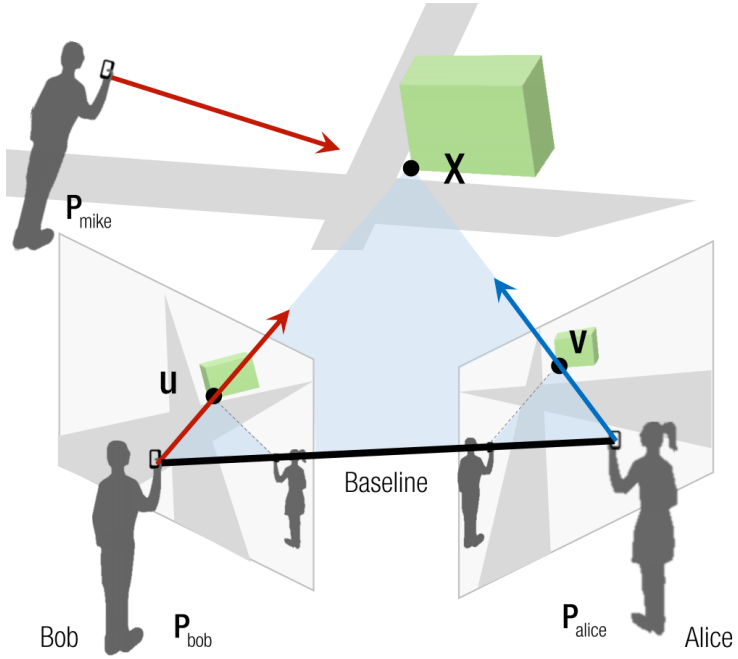
```
idx = CheckCheirality(Y,C1,R1,C2,R2)
```

Cheirality condition:

$$r_3^T (X - C) > 0$$

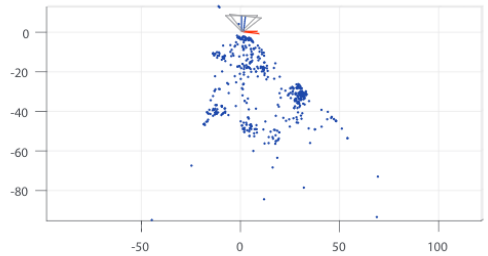
where

$$R = \begin{bmatrix} r_1^T \\ r_2^T \\ r_3^T \end{bmatrix}$$

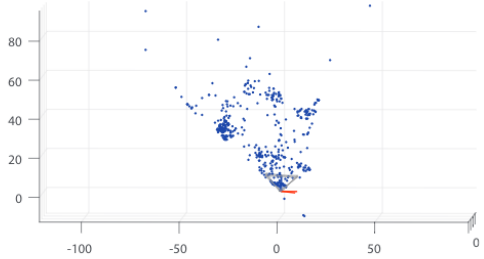


Triangulation

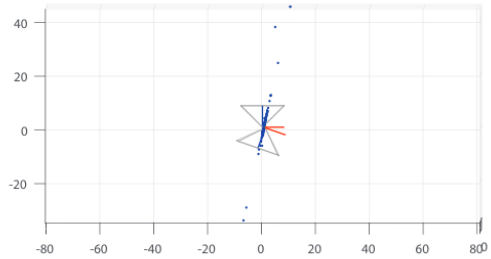
(3) (Camera pose disambiguation) Based on cheirality, find the correct camera pose. Visualize 3D camera pose and 3D points together as shown in Figure 2.
Hint: Use `plot3` and `DisplayCamera.m` to visualize them.



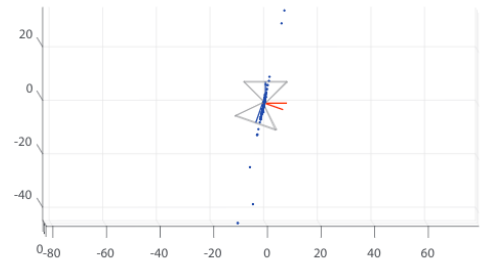
(a) nValid = 10



(b) nValid = 488



(c) nValid = 0



(d) nValid = 0

```
DisplayCamera(zeros(3,1), eye(3), 5);
hold on
c = -R'*t;
DisplayCamera(c, R, 5);
hold on
c = -R_new'*t_new;
DisplayCamera(c, R_new, 5);
hold on;
Y(:,abs(Y(1,:))>100) = [];
Y(:,abs(Y(2,:))>100) = [];
Y(:,abs(Y(3,:))>100) = [];
plot3(Y(1,:), Y(2,:), Y(3,:), 'b.')
axis equal
```

Triangulation

(4) (Reprojection) Project 3D points to each camera and visualize reprojection, $\hat{\mathbf{u}}$ and measurement, $\mathbf{u}$ onto the image as shown in Figure 3, i.e.,

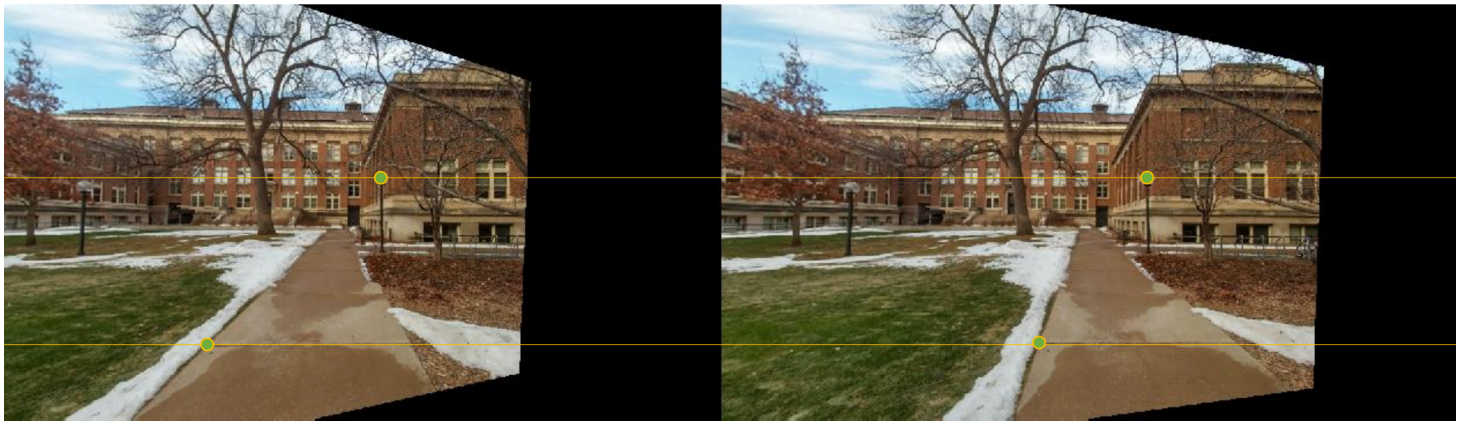
$$\lambda \begin{bmatrix} \hat{\mathbf{u}} \\ 1 \end{bmatrix} = \mathbf{K}\mathbf{R} \begin{bmatrix} \mathbf{I} & -\mathbf{C} \end{bmatrix} \begin{bmatrix} \mathbf{X} \\ 1 \end{bmatrix}$$

where $\mathbf{X}$ is the reconstructed point.

Figure 3: Visualization of measurements and reprojection.

Stereo

(1) (Stereo rectification) Given the disambiguated camera pose, find the rectification rotation matrix, $\mathbf{R}_{\text{rect}}$ such that the x-axis of the images aligns with the baseline. Find the rectification homography $\mathbf{H} = \mathbf{K}\mathbf{R}_{\text{rect}}\mathbf{R}^{\mathsf{T}}\mathbf{K}^{-1}$ where $\mathbf{R}$ is the rotation matrix of the camera. Rectify two images as shown in Figure 4. Visualize the epipolar lines of the sparse point correspondences. Check if they form horizontal line. Note that the epipolar line needs to be transformed by the homography, i.e., $\mathbf{l}_{\text{rect}} = \mathbf{H}^{-\mathsf{T}}\mathbf{l}$.



Stereo

(1) (Stereo rectification) Given the disambiguated camera pose, find the rectification rotation matrix, $\mathbf{R}_{\text{rect}}$ such that the x-axis of the images aligns with the baseline. Find the rectification homography $\mathbf{H} = \mathbf{K}\mathbf{R}_{\text{rect}}\mathbf{R}^{\mathsf{T}}\mathbf{K}^{-1}$ where $\mathbf{R}$ is the rotation matrix of the camera. Rectify two images as shown in Figure 4. Visualize the epipolar lines of the sparse point correspondences. Check if they form horizontal line. Note that the epipolar line needs to be transformed by the homography, i.e., $\mathbf{l}_{\text{rect}} = \mathbf{H}^{-\mathsf{T}}\mathbf{l}$.

Compute rotation matrix

$$\mathbf{r}_x = \frac{\mathbf{c}}{\|\mathbf{c}\|}$$

$$\mathbf{r}_z = \frac{\tilde{\mathbf{r}}_z - (\tilde{\mathbf{r}}_z \cdot \mathbf{r}_x)\mathbf{r}_x}{\left\|\tilde{\mathbf{r}}_z - (\tilde{\mathbf{r}}_z \cdot \mathbf{r}_x)\mathbf{r}_x\right\|}$$

$$\mathbf{r}_y = \mathbf{r}_z \times \mathbf{r}_x$$

: Orthogonal projection

where $\tilde{\mathbf{r}}_z = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

Stereo

(1) (Stereo rectification) Given the disambiguated camera pose, find the rectification rotation matrix, $\mathbf{R}_{\text{rect}}$ such that the x-axis of the images aligns with the baseline. Find the rectification homography $\mathbf{H} = \mathbf{K}\mathbf{R}_{\text{rect}}\mathbf{R}^{\mathsf{T}}\mathbf{K}^{-1}$ where $\mathbf{R}$ is the rotation matrix of the camera. Rectify two images as shown in Figure 4. Visualize the epipolar lines of the sparse point correspondences. Check if they form horizontal line. Note that the epipolar line needs to be transformed by the homography, i.e., $\mathbf{l}_{\text{rect}} = \mathbf{H}^{-\mathsf{T}}\mathbf{l}$.

Compute homography

- Same orientation

$$\mathbf{R}_{\text{rect}} = \begin{bmatrix} \mathbf{r}_x^T \\ \mathbf{r}_y^T \\ \mathbf{r}_z^T \end{bmatrix}$$

Homography by pure rotation: $\mathbf{R}_{\text{rect}}$

$$\mathbf{H}_{\text{bob}} = \mathbf{K}\mathbf{R}_{\text{rect}}\mathbf{K}^{-1}$$

$$\mathbf{H}_{\text{mike}} = \mathbf{K}\mathbf{R}_{\text{rect}}\mathbf{R}^{\mathsf{T}}\mathbf{K}^{-1}$$

Stereo

(1) (Stereo rectification) Given the disambiguated camera pose, find the rectification rotation matrix, $\mathbf{R}_{\text{rect}}$ such that the x-axis of the images aligns with the baseline. Find the rectification homography $\mathbf{H} = \mathbf{K}\mathbf{R}_{\text{rect}}\mathbf{R}^{\mathsf{T}}\mathbf{K}^{-1}$ where $\mathbf{R}$ is the rotation matrix of the camera. Rectify two images as shown in Figure 4. Visualize the epipolar lines of the sparse point correspondences. Check if they form horizontal line. Note that the epipolar line needs to be transformed by the homography, i.e., $\mathbf{l}_{\text{rect}} = \mathbf{H}^{-\mathsf{T}}\mathbf{l}$.

Warp images

Stereo

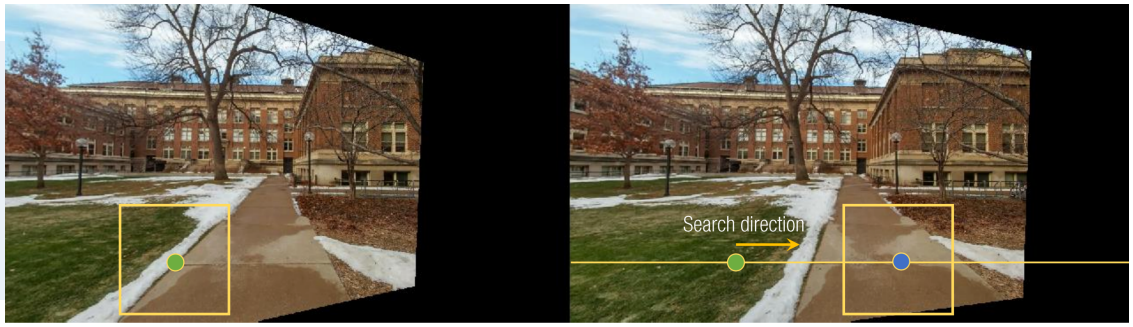
(2) (Dense feature extraction) Download MATLAB SIFT Flow code from `https://people.csail.mit.edu/celiu/SIFTflow/` and extract sift descriptors for each image. Unlike David Lowe's SIFT of HW #4 that produces sparse feature points, this code computes SIFT descriptor for each pixel, i.e., $W \times H$ SIFT descriptors will be computed if your image resolution is $W \times H$.

```
% compile and add path
addpath(fullfile(sift_flow_path,'mexDenseSIFT'));
addpath(fullfile(sift_flow_path,'mexDiscreteFlow'));

% parameters
cellsize=3;
gridspacing=1;

% compute SIFT
sift1 = mexDenseSIFT(im1,cellsize,gridspacing);
sift2 = mexDenseSIFT(im2,cellsize,gridspacing);
```

Stereo

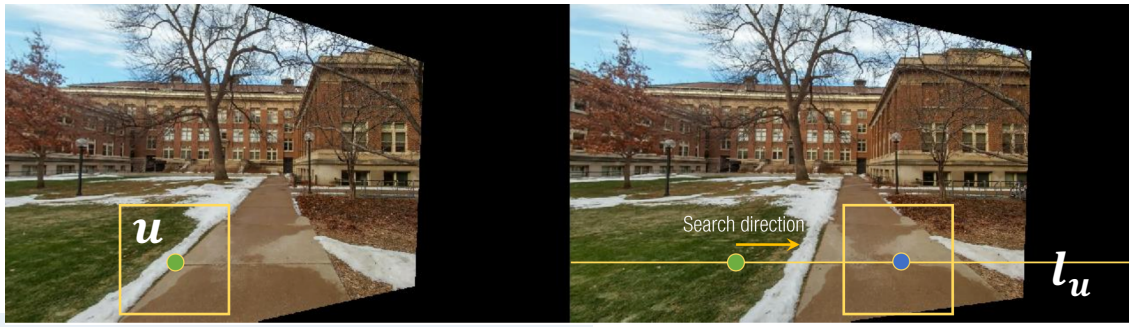
(3) (Dense stereo matching) Compute the dense matches across all pixels. Given a pixel, $\mathbf{u}$ in the left image, sweep along its epipolar line, $\mathbf{l_u}$, and find the disparity, d, that produces the best match

For each valid point in image1



Stereo

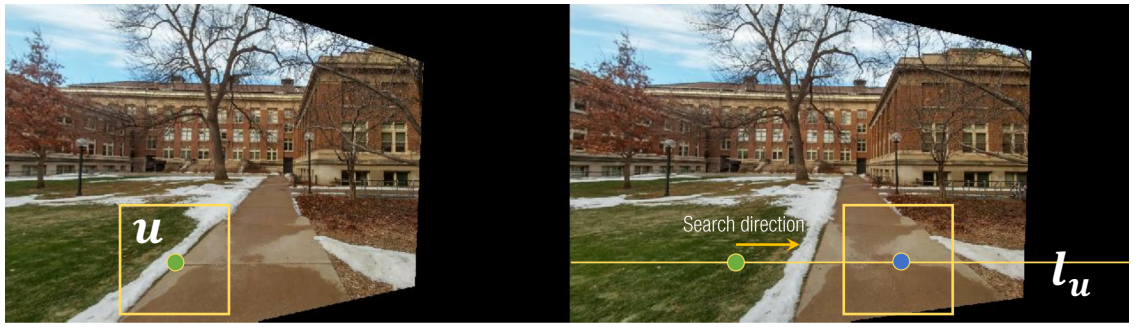
(3) (Dense stereo matching) Compute the dense matches across all pixels. Given a pixel, $\mathbf{u}$ in the left image, sweep along its epipolar line, $\mathbf{l_u}$, and find the disparity, d, that produces the best match



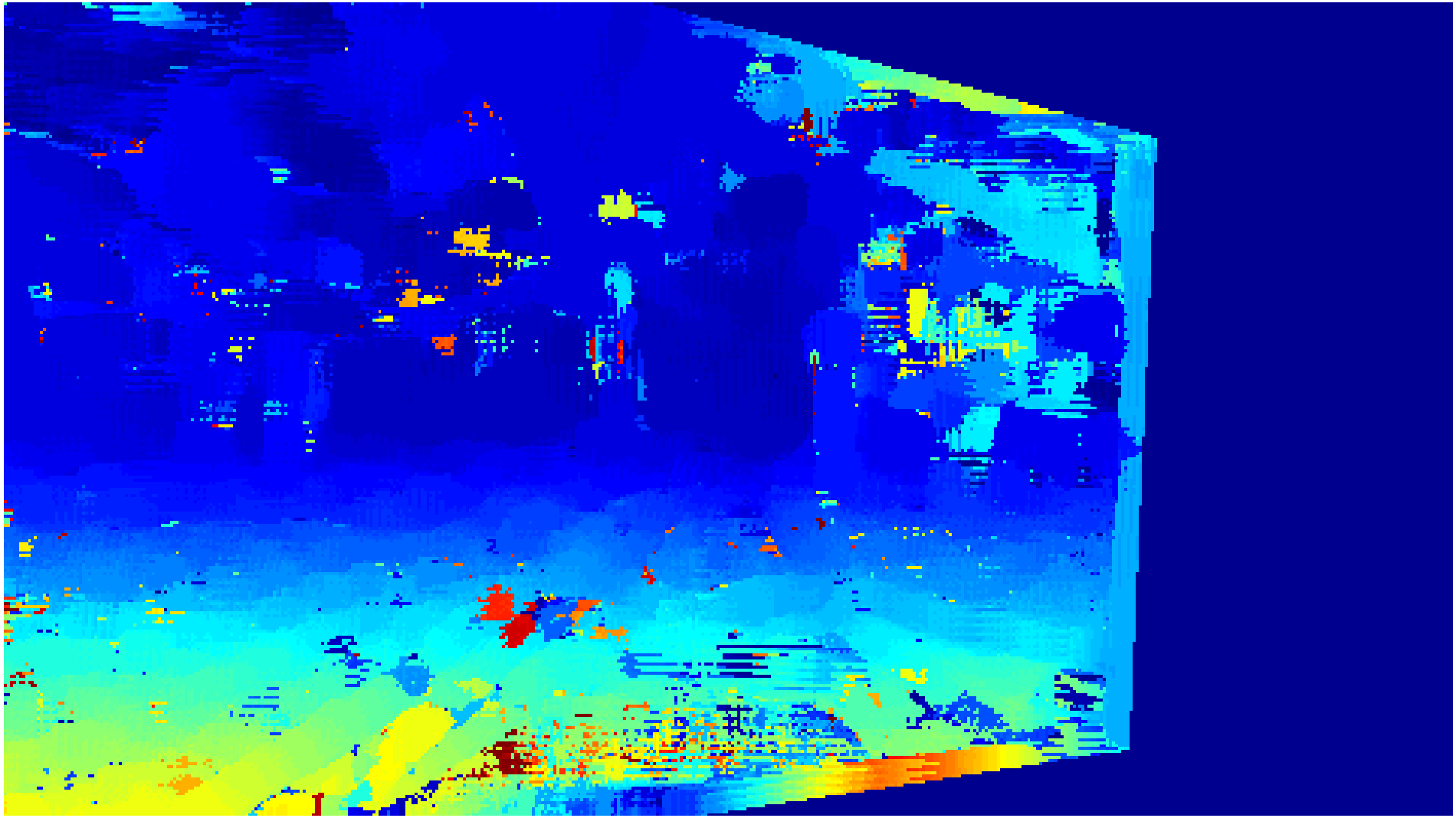
Take SIFT feature of point u and point on the corresponding epipolar line l_u

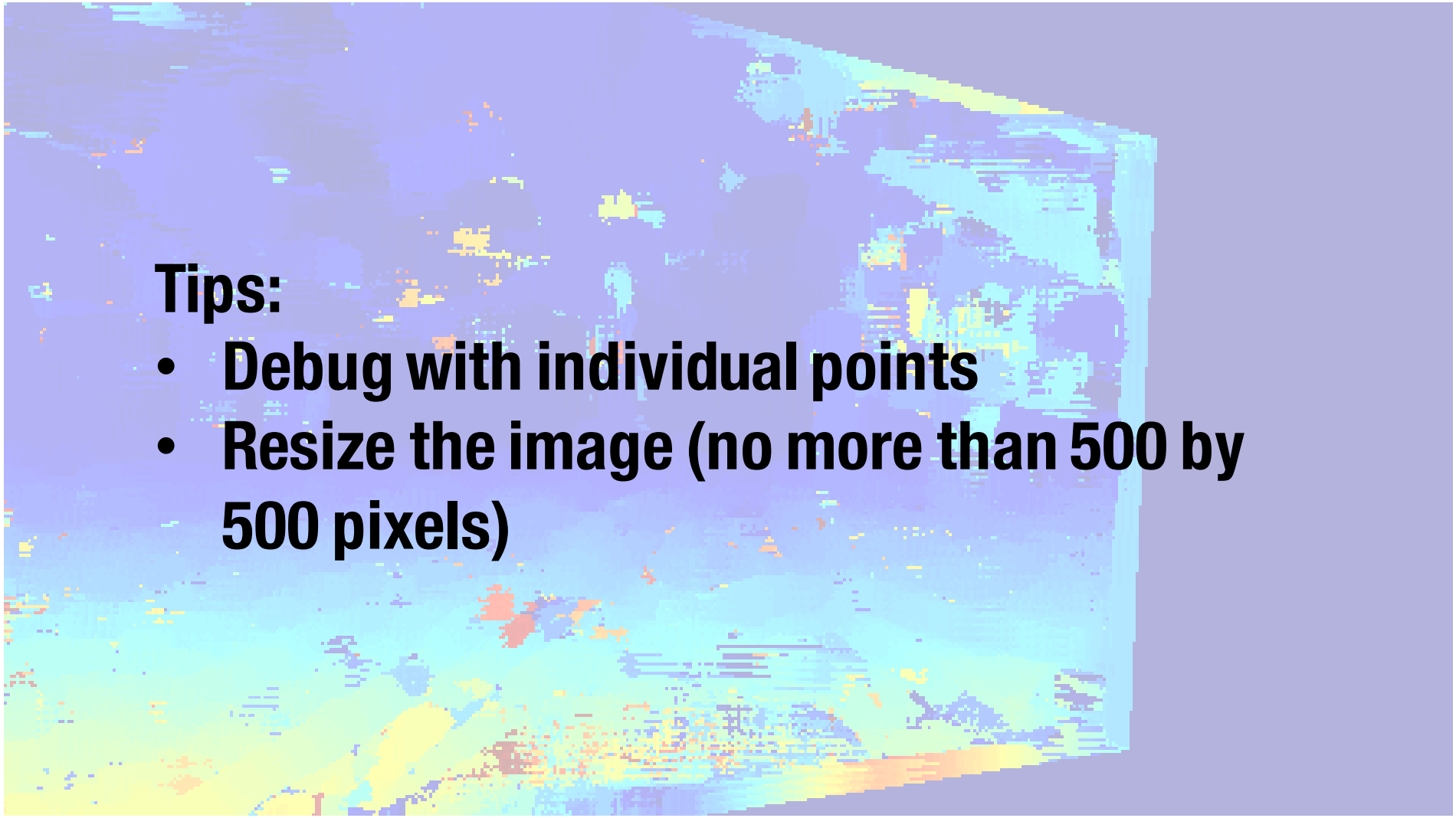
Stereo

(3) (Dense stereo matching) Compute the dense matches across all pixels. Given a pixel, $\mathbf{u}$ in the left image, sweep along its epipolar line, $\mathbf{l_u}$, and find the disparity, d, that produces the best match



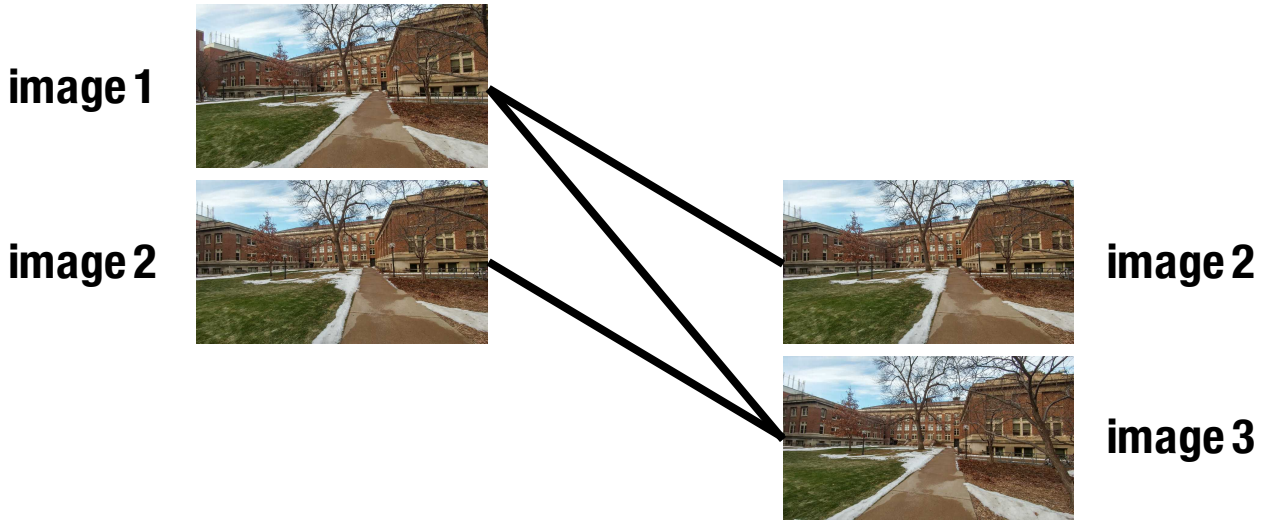
Find the match and set it as disparity




Tips:
• Debug with individual points
• Resize the image (no more than 500 by 500 pixels)

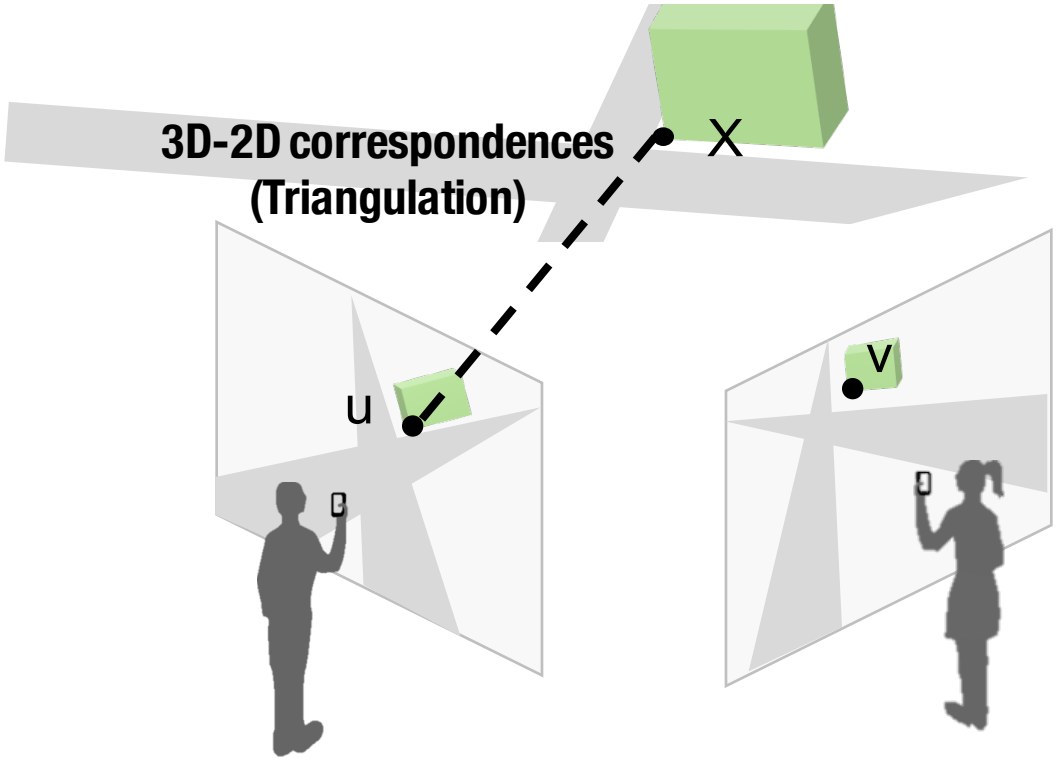
Additional Image Registration

(1) (New image matching) Similar to SIFT descriptor match in HW #4, find matches across three images, $\mathcal{I}_1$, $\mathcal{I}_2$, and $\mathcal{I}_3$. Compute matches for all possible pairs of images, i.e., $\mathcal{I}_1 \leftrightarrow \mathcal{I}_2$, $\mathcal{I}_1 \leftrightarrow \mathcal{I}_3$, $\mathcal{I}_2 \leftrightarrow \mathcal{I}_3$. Visualize matches after fundamental matrix based RANSAC for all matches.



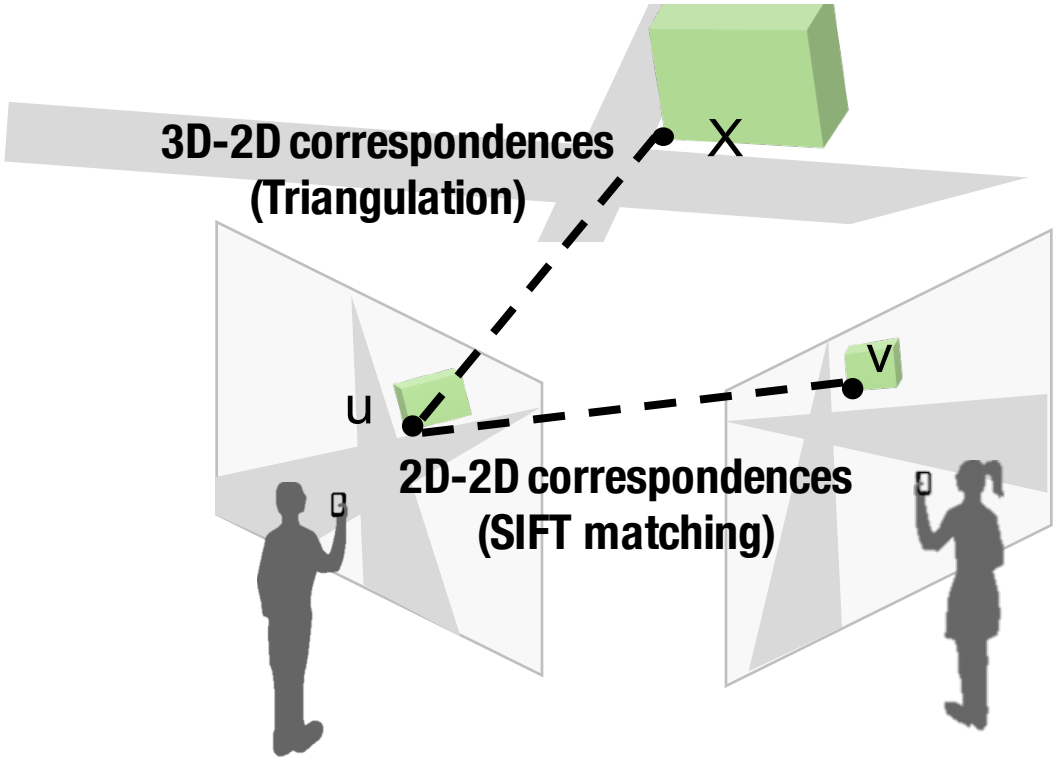
Additional Image Registration

(2) (3D-2D correspondences) Given 3D triangulated points, find 3D-2D matches, $\mathbf{X} \leftrightarrow \mathbf{w}$. Visualize the 3D reconstructed points visible to $\mathcal{I}_3$.



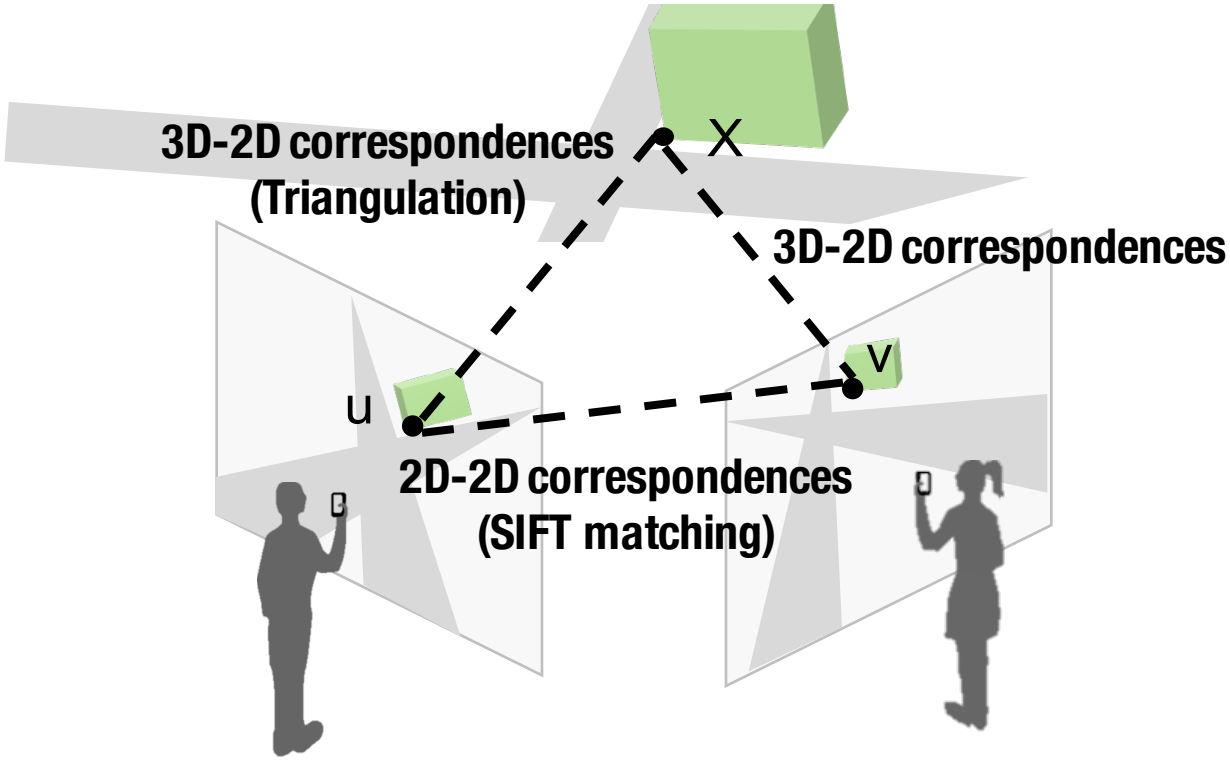
Additional Image Registration

(2) (3D-2D correspondences) Given 3D triangulated points, find 3D-2D matches, $\mathbf{X} \leftrightarrow \mathbf{w}$. Visualize the 3D reconstructed points visible to $\mathcal{I}_3$.



Additional Image Registration

(2) (3D-2D correspondences) Given 3D triangulated points, find 3D-2D matches, $\mathbf{X} \leftrightarrow \mathbf{w}$. Visualize the 3D reconstructed points visible to $\mathcal{I}_3$.

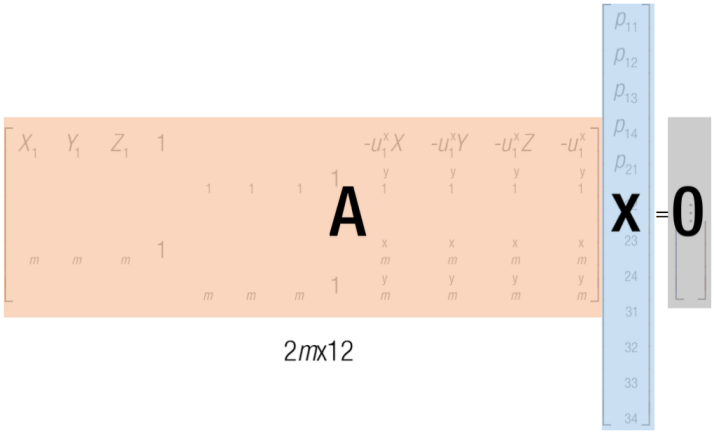


Additional Image Registration

(3) (Perspective-n-Point algorithm) Write a code that computes 3D camera pose from 3D-2D correspondences:

`[R, t] = LinearPnP(X, w, K)`

Set up A from 2D coordinates



Additional Image Registration

(3) (Perspective-n-Point algorithm) Write a code that computes 3D camera pose from 3D-2D correspondences:

`[R, t] = LinearPnP(X, w, K)`

If **K** is given,

$$\mathbf{K}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix} = \gamma\begin{bmatrix}\mathbf{p}_1 & \mathbf{p}_2 & \mathbf{p}_3 & \mathbf{p}_4\end{bmatrix}$$

$$\longrightarrow \gamma\mathbf{R} = \mathbf{K}^{-1}\begin{bmatrix}\mathbf{p}_1 & \mathbf{p}_2 & \mathbf{p}_3\end{bmatrix}$$

Solve R and t given K

Additional Image Registration

(3) (Perspective-n-Point algorithm) Write a code that computes 3D camera pose from 3D-2D correspondences:

`[R, t] = LinearPnP(X, w, K)`

$$\mathbf{K}^{-1}\begin{bmatrix}\mathbf{p}_1 & \mathbf{p}_2 & \mathbf{p}_3\end{bmatrix} = \mathbf{U}\begin{bmatrix} d_{11} & & \\ & d_{22} & \\ & & d_{33} \end{bmatrix}\mathbf{V}^{\mathsf{T}}$$

$\longrightarrow \gamma \approx d_{11}$

$\mathbf{R} = \mathbf{U}\mathbf{V}^{\mathsf{T}}$: SVD cleanup

$\longrightarrow \mathbf{t} = \dfrac{\mathbf{K}^{-1}\mathbf{p}_4}{d_{11}}$: Translation and scale recovery

SVD clean up and scale t

Additional Image Registration

(4) (RANSAC PnP) Write a RANSAC algorithm for the camera pose registration (PnP) given n matches using the following pseudo code:

Algorithm 1 PnP RANSAC

$nInliers \leftarrow 0$
for $i = 1 : M$ **do**
    Choose 6 correspondences, $\mathbf{X}_r$ and $\mathbf{w}_r$, randomly from $\mathbf{X}$ and $\mathbf{w}$.
    `[`$\mathbf{R}_r$`,` $\mathbf{t}_r$`] = LinearPnP(`$\mathbf{X}_r$`,` $\mathbf{w}_r$`,` $\mathbf{K}$`)`
    Compute the number of inliers, n_r, with respect to $\mathbf{R}_r$, $\mathbf{t}_r$.
    if $n_r > nInliers$ **then**
        $nInliers \leftarrow n_r$
        $\mathbf{R} = \mathbf{R}_r$ and $\mathbf{t} = \mathbf{t}_r$
    end if
end for

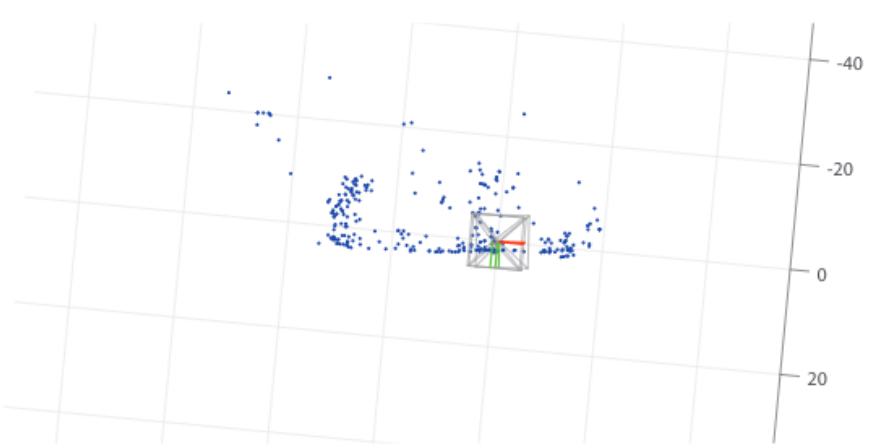
Additional Image Registration

(4) (RANSAC PnP) Write a RANSAC algorithm for the camera pose registration (PnP) given n matches using the following pseudo code:

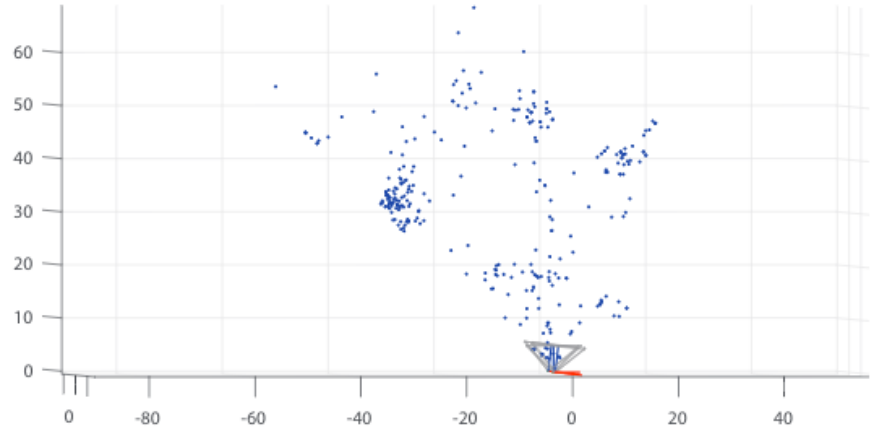
Algorithm 1 PnP RANSAC

```
nInliers ← 0
for i = 1 : M do
    Choose 6 correspondences, X_r and w_r, rand
    [R_r, t_r] = LinearPnP(X_r, w_r, K)
    Compute the number of inliers, n_r, with respect to R_r, t_r.
    if n_r > nInliers then
        nInliers ← n_r
        R = R_r and t = t_r
    end if
end for
```

Reprojection error $e = \sqrt{\left(u_j - \frac{\mathbf{P}_1\widetilde{\mathbf{X}}_j}{\mathbf{P}_3\widetilde{\mathbf{X}}_j}\right)^2 + \left(v_j - \frac{\mathbf{P}_2\widetilde{\mathbf{X}}_j}{\mathbf{P}_3\widetilde{\mathbf{X}}_j}\right)^2}$



(a) Front view



(b) Top view