```
%========================
%  Name:                hw3_1.m
%
%  Author:              Kairi Kozuma
%
%========================

% Transformation matrices
T_WC = [-10;15;-5];
R_WC = [0.836516303737808,0.224143868042013,0.500000000000000;0.0173375885302536,0.901221065013438,-0.433012701892219;-0.547667674420164,0.37089097912

% a) Find camera frame points from world frame points

    % Points in world frame
    p1_W = [6.60000000000000,-7.20000000000000,-41.1000000000000,-13.5000000000000;-1.70000000000000,10.5000000000000,10.7000000000000,18.300000000000

    % Convert to camera frame
    qc1 = transformToCamera(p1_W, R_WC', -R_WC'*T_WC);

    % Print out values
    fprintf('a) Points in camera frame:\n');
    disp (qc1(1:3,:));

% b) Determine which points are in field of view

    % Field of view +- the following value
    horiFOV = 45;
    vertFOV = 30;

    % Determine if in field of view
    inView = inFOV(qc1(1:3,:),horiFOV, vertFOV);

    fprintf('b) Points in field of view:\n');
    count = 0;
    for n = 1:length(inView)
        if (inView(n))
            fprintf('Point q%d in field of view\n', n);
            count = count + 1;
        end
    end

    if (count == 0)
        fprintf('\tNo points in field of view\n\n');
    end


% c) Find world frame points from camera frame points

    % Points in camera frame
    p2_C = [2.80000000000000,13.1000000000000;1.40000000000000,-11.3000000000000;16,28];

    % Conver to world frame
    qw2 = transformToWorld(p2_C, R_WC, T_WC);

    % Print out values
    fprintf('c) Points in world frame:\n');
    disp (qw2(1:3,:));

%======================= transformToCamera ==============================
%
%  qc = transformToCamera(pw, R_CW, T_CW)
%
%  INPUTS:
%    pw         - point in 3 dimension, world frame
%    R_CW       - rotation matrix
%    T_CW       - translation vector
%
%  OUTPUTS:
%    qc             - point in 3 dimensions, camera frame
%
%======================= transformToCamera ==============================
function [pc] = transformToCamera(pw, R_CW, T_CW)

transformMatrix = [R_CW,T_CW;0,0,0,1];

dim = size(pw);
lastRow = ones([1,dim(2)]);

qw = [pw; lastRow];

pc = transformMatrix * qw;

end

%=================================== inFOV ==============================
%
%  inView = inFOV(pc, horiFOV, vertFOV)
%
%  INPUTS:
%    pc         - point in 3 dimensions, camera frame
```

```
%    horiFOV    - horizontal field of view, +- value
%    vertFOV    - vertical field of view, +- value
%
%  OUTPUTS:
%    inView             - boolean vector of whether points are in FOV
%
%=================================== inFOV ==============================
function [inView] = inFOV(pc, horiFOV, vertFOV)

angleY = (180 / pi) * atan2(pc(2,:), pc(3,:));
angleX = (180 / pi) * atan2(pc(1,:), pc(3,:));

angles = [angleX; angleY];

inHoriView = (angles(1,:) >= -horiFOV & angles(1,:) <= horiFOV);
inVertView = (angles(2,:) >= -vertFOV & angles(2,:) <= vertFOV);

inView = inHoriView & inVertView;

end

%======================= transformToWorld ==============================
%
%  qw = transformToWorld(pc, R_WC, T_WC)
%
%  INPUTS:
%    pc         - point in 3 dimensions, camera frame
%    R_WC       - rotation matrix
%    T_WC       - translation vector
%
%  OUTPUTS:
%    qw             - point in 3 dimensions, world frame
%
%======================= transformToWorld ==============================
function [pw] = transformToWorld(pc, R_WC, T_WC)

transformMatrix = [R_WC,T_WC;0,0,0,1];

dim = size(pc);
lastRow = ones([1,dim(2)]);

qc = [pc; lastRow];

pw = transformMatrix * qc;

end
```

```
a) Points in camera frame:
    6.6960   -8.4153  -21.4898   -1.5562
   -6.6564    3.8045  -13.9616    1.2994
   24.9813   17.9736  -19.9880   -4.9789

b) Points in field of view:
Point q1 in field of view
Point q2 in field of view
c) Points in world frame:
    0.6560   12.4255
    9.3821   -7.0810
    5.9858    4.6345
```

```matlab
%=========================
%  Name:                hw3_2.m
%
%  Author:              Kairi Kozuma
%
%=========================

% Transformation matrix
R_WL = [0.913545457642601,-0.063627629171822,0.401729040058774;0.287606238475951,0.799453749866612,-0.527405302792764;-0.287606238475951,0.59734849680
R_WR = [0.994521895368273,-0.016351854232753,0.103241544429788;0.073912785203567,0.808411029059454,-0.583959337863936;-0.073912785203567,0.58839121760
T_WL = [-8.659258262890683;2.169872981077807;4.830127018922193];
T_WR = [10.659258262890683;5.830127018922193;1.169872981077807];

% Homogenous points in world frame
q_W = [16,78.3000000000000,33.1000000000000;-25.4000000000000,-20.9000000000000,-39.1000000000000;19.1000000000000,7.30000000000000,38.5000000000000;1

% a) Transformation giving camera's R frame relative to L frame

    G_WR = [R_WR,T_WR;0,0,0,1];
    G_WL = [R_WL,T_WL;0,0,0,1];
    G_LW = [R_WL',-R_WL'*T_WL;0,0,0,1];
    G_LR = G_LW * G_WR;

    fprintf('a) Transformation matrix of R frame relative to L frame:\n');
    disp(G_LR);

% b) Coordinates of points given in both frames

    % Convert to camera R frame
    qcR1 = transformToCamera(q_W(1:3,:), R_WR', -R_WR'*T_WR);
    fprintf('b1) Points in camera R frame:\n');
    disp (qcR1(1:3,:));

    % Convert to camera R frame
    qcL1 = transformToCamera(q_W(1:3,:), R_WL', -R_WL'*T_WL);
    fprintf('b2) Points in camera L frame:\n');
    disp (qcL1(1:3,:));

% c) Both cameras have horizontal FOV of 60deg, vertical FOV of 40deg
%    Specify if each point is visible by L only, R only, or both

    % Field of view +- the following value
    horiFOV = 30;
    vertFOV = 20;

    % Determine if in field of view
    inViewR = inFOV(qcR1(1:3,:),horiFOV, vertFOV);
    inViewL = inFOV(qcL1(1:3,:),horiFOV, vertFOV);

    inView = inViewR & inViewL;

    fprintf('b) Points in both fields of view:\n');
    count = 0;
    for n = 1:length(inView)
        if (inView(n))
            fprintf('Point q%d in field of view\n', n);
            count = count + 1;
        end
    end

    if (count == 0)
        fprintf('\tNo points in field of view\n\n');
    end

%======================== transformToCamera ===============================
%
%  qc = transformToCamera(pw, R_CW, T_CW)
%
%  INPUTS:
%    pw         - point in 3 dimension, world frame
%    R_CW       - rotation matrix
%    T_CW       - translation vector
%
%  OUTPUTS:
%    qc             - point in 3 dimensions, camera frame
%
%======================== transformToCamera ===============================
function [pc] = transformToCamera(pw, R_CW, T_CW)

transformMatrix = [R_CW,T_CW;0,0,0,1];

dim = size(pw);
lastRow = ones([1,dim(2)]);

qw = [pw; lastRow];

pc = transformMatrix * qw;

end
```

```
%==================================== inFOV ===============================
%
%  inView = inFOV(pc, horiFOV, vertFOV)
%
%  INPUTS:
%    pc         - point in 3 dimensions, camera frame
%    horiFOV    - horizontal field of view, +- value
%    vertFOV    - vertical field of view, +- value
%
%  OUTPUTS:
%    inView             - boolean vector of whether points are in FOV
%
%==================================== inFOV ===============================
function [inView] = inFOV(pc, horiFOV, vertFOV)

angleY = (180 / pi) * atan2(pc(2,:), pc(3,:));
angleX = (180 / pi) * atan2(pc(1,:), pc(3,:));

angles = [angleX; angleY];

inHoriView = (angles(1,:) >= -horiFOV & angles(1,:) <= horiFOV);
inVertView = (angles(2,:) >= -vertFOV & angles(2,:) <= vertFOV);

inView = inHoriView & inVertView;

end
```

```
a) Transformation matrix of R frame relative to L frame:
    0.9511    0.0483   -0.3052   19.7538
   -0.0483    0.9988    0.0076   -0.4894
    0.3052    0.0076    0.9523    3.0902
         0         0         0    1.0000

b1) Points in camera R frame:
    1.6779   64.8414   16.2377
  -14.7842  -19.1081  -14.7242
   33.2257   27.5286   58.6121

b2) Points in camera L frame:
   10.4940   72.0958   16.5958
  -15.0858  -22.5009  -15.5377
   35.1298   48.9503   63.7484

b) Points in both fields of view:
Point q3 in field of view
```

```matlab
%===========================
%  Name:                hw3_4.m
%
%  Author:              Kairi Kozuma
%
%===========================

fprintf('a)\n');
    fprintf('Distance transform is an operator applied to binary images that\n');
    fprintf('that results in a grayscale image, where the distance from the\n');
    fprintf('closest boundary determines the intensities of the gray scale.\n');
    fprintf('\n\n');
    fprintf('Two distance types are Manhattan distance and Euclidean distance:\n');
    fprintf('\tIn Manhattan distance, the distance between two points is the\n');
    fprintf('\tsum of the difference in absolute differences of the Cartesian\n');
    fprintf('\tcoordinates.\n');
    fprintf('\tIn Euclidean distance, the distance between two points is the\n');
    fprintf('\tstraight line distance between two points in Euclidean space.\n');
    fprintf('\tIn other words, it is the square root of the sum of the squares\n');
    fprintf('\tof its Cartesian coordinates.\n');

fprintf('b)\n');
    % Set threshold values
    lowerThresh = 959;
    upperThresh = 978;

    % Apply threshold to range image
    binImage = (range > lowerThresh) & (range < upperThresh);

    % bwdistgeodesic to pick out person
    distGray = bwdistgeodesic(binImage, [300], [261]);

    figure(1);
    distGray(isnan(distGray)) = 255;
    imshow(distGray,[0,255]);
    title('Distance image of original binary image');

 fprintf('c)\n');
    threshold = 164;
    distGray2 = distGray;
    distGray2(distGray2 > threshold) = 255;
    figure(2);
    imshow(distGray2,[0,255]);
    title('Distance image of original binary image with threshold');

 fprintf('d)\n');
 fprintf('\tThe distance transform allows filtering of objects that are far\n');
 fprintf('\tfrom the seed points. This was effective in removing the unnecessary\n');
 fprintf('\tbottom board that was attached to the filtered person in the bwselect\n');
 fprintf('\tfiltering method. However, the feet were chopped off the filtere person.\n');
```

```
a)
Distance transform is an operator applied to binary images that
that results in a grayscale image, where the distance from the
closest boundary determines the intensities of the gray scale.


Two distance types are Manhattan distance and Euclidean distance:
        In Manhattan distance, the distance between two points is the
        sum of the difference in absolute differences of the Cartesian
        coordinates.
```

In Euclidean distance, the distance between two points is the straight line distance between two points in Euclidean space. In other words, it is the square root of the sum of the squares of its Cartesian coordinates.

b)

c)

d)

The distance transform allows filtering of objects that are far from the seed points. This was effective in removing the unnecessary bottom board that was attached to the filtered person in the bwselect filtering method. However, the feet were chopped off the filtere person.

Distance image of original binary image

Distance image of original binary image with threshold

```
%===========================
%  Name:                hw3_5.m
%
%  Author:              Kairi Kozuma
%
%===========================

fprintf('a)Inverse matrix result\n');

x1 = [4.900000000000000;0.200000000000000];
y1 = [66;54.660000000000004];
x2 = [-2.600000000000000;1.800000000000000];
y2 = [-41.120000000000000;-21.760000000000000];

yvec = [y1;y2];
xmat = [x1',0,0;0,0,x1';x2',0,0;0,0,x2'];
avec = inv(xmat)*yvec;
solutionA = reshape(avec,2,2)';
disp(solutionA);

fprintf('b)Singular Value Decomposition result\n');

mat = [xmat, -yvec;0,0,0,0,0];
[UU, SS, VV] = svd(mat);
avecSVD = VV(:,5)./VV(5,5);
avecSVD = avecSVD(1:4);
solutionB = reshape(avecSVD,2,2)';
disp(solutionB);
```

```
a)Inverse matrix result
   13.6000   -3.2000
   11.0000    3.8000

b)Singular Value Decomposition result
   13.6000   -3.2000
   11.0000    3.8000
```