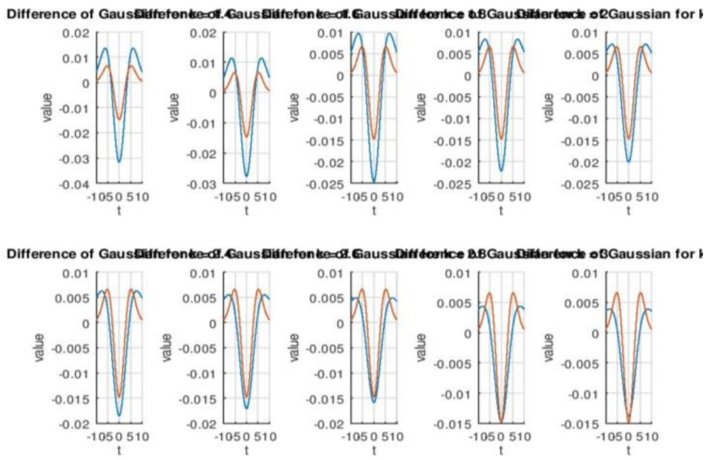
Question 1

Scale-Invariant Feature Detection



```
gauss = gaussian(t,sigma);
first_derivative_of_gauss = -t/power(sigma,2).*gauss;

hold on
plot(t,gauss);
plot(t,first_derivative_of_gauss);
hold off

second_derivative_of_gauss = (t.*t-power(sigma,2)).*gauss/power(sigma,4);

figure,
plot(t,second_derivative_of_gauss);
title("Difference of Gaussian for different k values");
xlabel("t");
ylabel("value");
grid on
```

Figure: Second derivative (instead of LoG for 2D) and Difference of Gaussian function similarity

Pseudo Code

Define sigma = sqrt(2) and kernal_size = 6*sqrt(2)*sigma

Create blurred images

Use different sigma values at each level of the octave. Sigma is change sqrt(2) between levels.

Downsample by 2 to create the next octave and choose sigma = sigma value corresponding to 2 levels from the top level in the previous octave

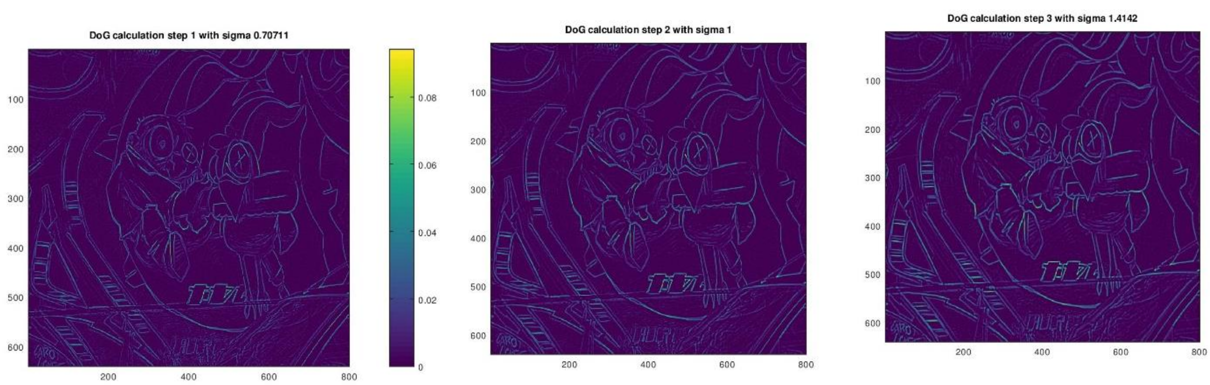
Create Difference of Gaussian (DoG) scale-space by substracting consecutive blurred images. Now these are somewhat similar to getting the Laplacian of Gaussian (LoG).

Select local maxima or minima points considering 3x3x3 neighbourhood of a point. These points are taken as key-points in SIFT

Use Harris critera to eliminate key-points related to edge responsed

The principal curvatures are computed from a 2x2 Hessian matrix, H, computed at the location and scale of the key-point. First calculate the gradients in x and y directions in the keypoint. Then calculate the second moment matrix. $\frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} < \frac{(r+1)^2}{r}.$ Criteria is used to eliminate outliers. r=10 is used as suggested in Lowe's paper.

Then calculate gradient magnitude and directions near a region at selected keypoints, assign the nearby points depending on their direction to a histogram by weighting gradient magnitude and gaussian kernal with 1.5*sigma of the scale of that keypoint. Use 36 bins in the histogram. A second orientataion is assigned to keypoints if the second highest bin value is greater than 80% of the maximum bin value.



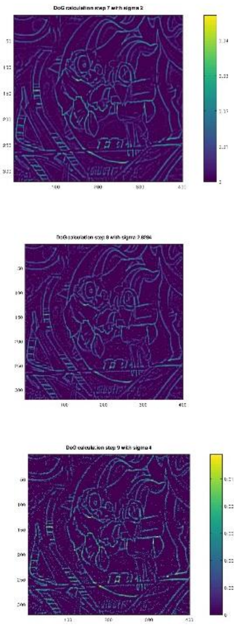


Figure: First 3 levels in the first octave and first 3 levels in the second octave in DoG space

Figure: Selected key-points are shown on the image

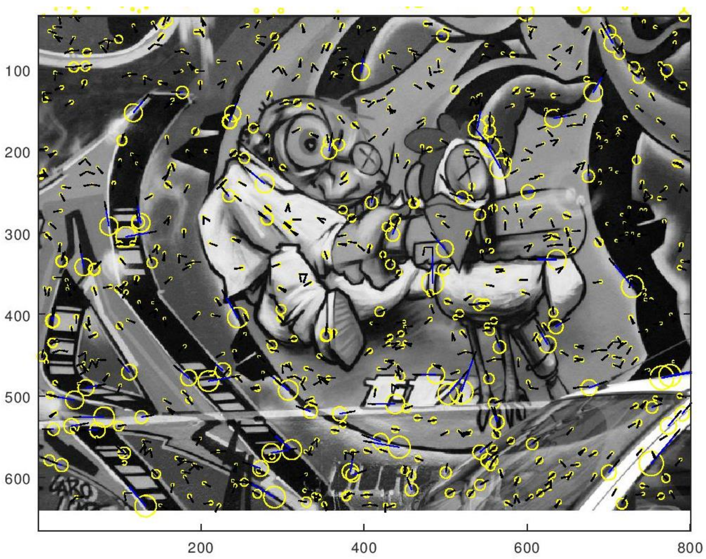


Figure: Assigned orientations are shown in blue arrows.

```
for oct=0:octaves-1

  I = J;
  disp(['Sigma at the start of octave ' num2str(oct+

  for k = 1:stepsPerOctave

      step = oct*(stepsPerOctave)+k;
      disp(['With sigma = ' num2str(sigma) ' creatin

      gauss = fspecial('gaussian', kernelSize, sigma
      blur{step} = imfilter(I, gauss, 'replicate', '
      sigma = sigma * mult;

      tau = 0.06;
      num_of_points = 500;
      [Ix, Iy] = gradient(blur{step});
      Ixx = imfilter(Ix.*Ix, gauss);   % second deri
      Iyy = imfilter(Iy.*Iy, gauss);   % second deri
      Ixy = imfilter(Ix.*Iy, gauss);   % second deri
      har = Ixx.*Iyy - Ixy.*Ixy - tau*(Ixx+Iyy).^2;
      harris{step} = har;

      % check that center is strict max
      maxv = ordfilt2(har, 49, ones(7)); % maximum f
      maxv2 = ordfilt2(har, 48, ones(7));   % second
      ind = find(maxv==har & maxv~=maxv2);  % check

      indices{step} = ind;
  end

  sigma = sigma / power(mult,3);
  gauss = fspecial('gaussian', kernelSize, sigma);
  J = imfilter(I, gauss, 'replicate', 'same');
  J = imresize(J, 0.5);

end
```

Figure: Gaussian Filtered images and calculation
Of Harris criterion for all the levels. Sqrt (2) sigma difference

```
sigma = sqrt(2)/2;
% Create DoG
for oct = 0:octaves-1
  for k = 1:stepsPerOctave-1
      step = oct*(stepsPerOctave)+k;
      disp(['DOG calculation step: ' num2st
      dog{step} = blur{step+1}-blur{step};
      sigma = sigma * mult;
  end
  sigma = sigma / power(mult,2);
end
```

Figure: DoG space-scale generation using gaussian filtered images

```
hw = 2;
gauss = fspecial('gaussian', [2*hw+1,2*hw+1], sig*1.5);

m = 0;  theta = 0;
for k=-hw:hw
    for l=-hw:hw
        #[m, theta] = point_grad(I, i+k, j+l);
        x = i+k;     y = j+l;

        m = sqrt( (I(x,y-1)-I(x,y+1))^2 + (I(x-1, y)-I(x+1, y))^2 );
        theta = atan( (I(x+1, y)-I(x-1, y))/(I(x, y+1)-I(x, y-1)) );
        if theta<0
            theta = 2*pi+theta;
        end
        theta = theta*180/pi;

        idx = idivide(theta, 10);   % find the corresponding bin of
        hist(idx+1) = hist(idx+1) + m * gauss(k+hw+1, l+hw+1);  # we
    end
end
%     hist
[max_grad,orientation] = max(hist);

D_harris(4,p) = (orientation-1)*10;
D_harris(5,p) = max_grad*1000*sig+sig;

hist(orientation) = -Inf;
[second_max,orient] = max(hist);

if(second_max > max_grad*0.8)
  d = [i; j; sig; (orient-1)*10; second_max*1000*sig+sig];
  disp([" A SECOND ORIENTATAION FOUND " num2str((orient-1)*10)]);
  D_new = [D_new d];
endif
```

Figure: Code for orientation assignment and second orientation identification using 80% criteria

```
for i=2:m-1
  for j=2:n-1

    mat(:,:,1) = lower_level(i-1:i+1,j-1:j+1);
    mat(:,:,2) = middle_level(i-1:i+1,j-1:j+1);
    mat(:,:,3) = upper_level(i-1:i+1,j-1:j+1);

    val = mat(2,2,2);                  # replace middl
    mat(2,2,2) = mat(1,1,1);

    mat_min = min(min(min(mat)));
    mat_max = max(max(max(mat)));

    if (val>mat_max || val<mat_min)
        d = [i; j; sigma];
        D = [D d];
        D_all = [D_all d];

        if ( ismember((i-1)*n + j, ind) && i-3>0
          disp([num2str(i) '  ' num2str(j) '  ' r
          D_new = [D_new d];
          D_harris = [D_harris d];
        endif
    end

  endfor
endfor
```

Figure: Select key-points using maxima minima criterion followed by Harris criterion

References:

[1] David G. Lowe, Distinctive Image Features from Scale-Invariant Key-points
http://www.cs.jhu.edu/~hager/Public/teaching/cs461/LoweIJCV.pdf

Question 2

Kanade-Lukas-Tomashi (KLT) Tracker Algorithm

Pseudo Code

Select key-points from the first image (m, n) using Harris Criteria

- Blur image with gaussian filter
- Compute Ix and Iy (Image gradients)
- Find Ixx, Iyy, Ixy | second moment matrix (2, 2) convoluted with gaussian filter
- Calculate Harris matrix (m, n)
- Find local maxima in the Harris criterion matrix

```
%%
% detect keypoints
% calculate harris corners
tau = 0.06;
num_of_points = 500;
il = images{1};

gauss_filter = fspecial('gaussian', [7 7], 1);
imblur = imfilter(il, gauss_filter);
[Ix, Iy] = gradient(imblur);
Ixx = imfilter(Ix.*Ix, gauss_filter);   % second derivative w.r.t. x
Iyy = imfilter(Iy.*Iy, gauss_filter);   % second derivative w.r.t. y
Ixy = imfilter(Ix.*Iy, gauss_filter);   % second derivative w.r.t. x and y
har = Ixx.*Iyy - Ixy.*Ixy - tau*(Ixx+Iyy).^2;   % Harris criterion

% check that center is strict max
maxv = ordfilt2(har, 49, ones(7)); % maximum filter
maxv2 = ordfilt2(har, 48, ones(7));   % second max after ordering in ascendi
ind = find(maxv==har & maxv~=maxv2);  % check if it is a local maximum

% get top N points
[sv, sind] = sort(har(ind), 'descend');
sind = ind(sind);
[pty, ptx] = ind2sub(size(il), sind(1:min(num_of_points, numel(sind))));
```

Figure: Harris criterion MATLAB code

- Select the most prominent maxima depending on the number of key-points needed
- Get key-point locations are the prominent maxima locations

The next part of the algorithm is to track these points iteratively using KLT tracker, let's consider using the first and second images for tracking the movement

Let initial positions of the points to track be selected key-points
Calculate Ix, Iy gradients of the first image, use gaussian filter before gradient calculation
Create matrix with all the tracked points and their neighbors (neighbors are pixels those pixels which falls inside 5x5 or 7x7 window used in the algorithm.
Interpolate gradients to find the gradients at the tracked point locations and their neighbors (tracked point location is not an integer after the first image), cubic interpolation is used in the code. Bilinear interpolation also yields good results.
Interpolate pixel brightness values for all the image patches for considered point locations in the first image and the second image.

Now, using those gradients find the pixel movement using

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

Here, only the neighborhood is considered.

```
for iter = 1:5

    % need to make sure point hasn't drifte
    valid2 = valid & x2>=hw & y2>=hw & x2+h
    x2(~valid2) = nan;
    y2(~valid2) = nan;

    x2w = repmat(x2, [1 numel(winx)])'+repm
    y2w = repmat(y2, [1 numel(winy)])'+repm
    patch2_all = zeros(size(patch1_all));
    patch2_all(:, valid2) = interp2(im2, x2

    % for each point, step towards the matc
    for p = 1:numel(x)

        if ~valid2(p)
            continue;
        end

        patch1 = patch1_all(:, p);
        Ix = Ix_all(:, p);
        Iy = Iy_all(:, p);
        patch2 = patch2_all(:, p);

        It = patch2-patch1;
        A = [sum(Ix.*Ix) sum(Ix.*Iy) ;
             sum(Ix.*Iy) sum(Iy.*Iy)];
        b = -[sum(Ix.*It) ; sum(Iy.*It)];
        d = A\b;
        x2(p)=x2(p)+d(1);
        y2(p)=y2(p)+d(2);
```

Figure: KLT algorithm – iterative calculation

Iterate through all the points while updating the point locations according to x = x + u and y = y + v;

Do this until the point has converged (x(t+1)-x(t)<threshold) or a few iterations.

Then go to the next 2 frames, i.e. 2[nd] frame and the 3[rd] frame, follow the same procedure iteratively.

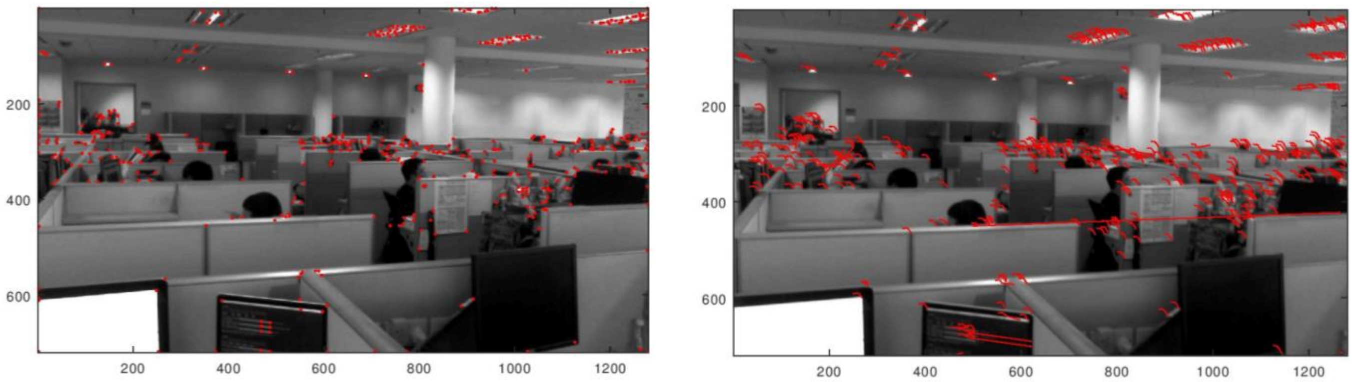


Figure: Results of KLT tracker. Image on the left shows the initially detected key-points. Image on the right shows the movement of those points found using KLT tracking. Images are taken from LiveLabs Urban Lifestyle Innovation Platform, Singapore.

References:
[1] https://en.wikipedia.org/wiki/Bicubic_interpolation
[2] http://dhoiem.cs.illinois.edu/

Question 3

Affine Transformation

```
subplot(1,2,1)
imagesc(images{1}), hold on, axis image
[x1,y1] = ginput(number_of_points);
plot(x1, y1, 'g'); drawnow; hold off;

subplot(1,2,2)
imagesc(images{2}), hold on, axis image
[x2,y2] = ginput(number_of_points);
plot(x2, y2, 'g'); drawnow; hold off;

a2 = [x2'; y2'];
a1 = [x1'; y1'; ones(1, numel(x1))];

#a2 = trans_mat * a1;
trans_mat = a2 * pinv(a1);
trans_mat(3,:) = [0, 0, 1];

a2
trans_mat * a1

#J = imwarp(images{1},trans_mat);
J = imperspectivewarp(images{1}, trans_mat, "bicubic", "same");
figure,
imagesc(J), hold on, axis image
```

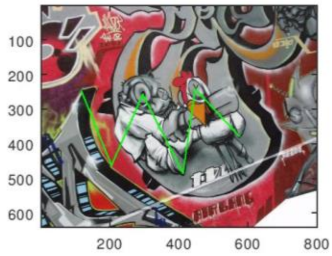


Figure: First image in the left. Second image in the middle. Chosen points are shown in green colour using ginput funstion. Transformed image after calculating the affine transformation is shown in the right and is identical to the middle image.

Computing Homography

```
subplot(1,2,1)
imagesc(images{1}), hold on, axis image
[x1,y1] = ginput(number_of_points);
plot(x1, y1, 'g'); drawnow; hold off;

subplot(1,2,2)
imagesc(images{2}), hold on, axis image
[x2,y2] = ginput(number_of_points);
plot(x2, y2, 'g'); drawnow; hold off;

a2 = [x2'; y2'; ones(1, numel(x1))];
a1 = [x1'; y1'; ones(1, numel(x1))];

#a2 = trans_mat * a1;
trans_mat = a2 * pinv(a1);

J = imperspectivewarp(images{1}, trans_mat, "bicubic", "same");
figure,
imagesc(J), hold on, axis image
```

Calculating homography is somewhat same as calculating the affine transformation, method is the same but homography transformation has 8 DoF while affine transformation has only 6 DoF.