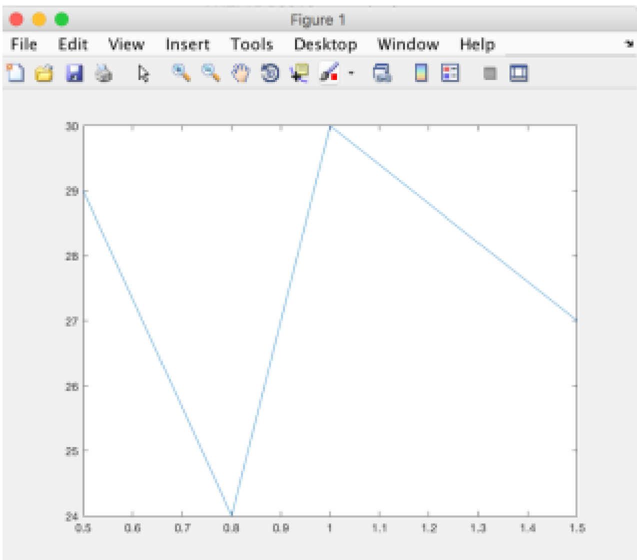
SIFT AND MOPS OF IMAGE DATABASE

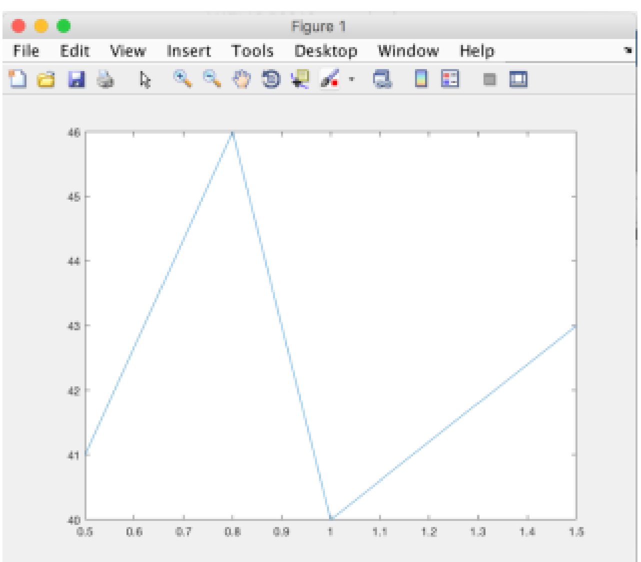
Shreyas Pattabiraman UIN: 674434423

MOPS:

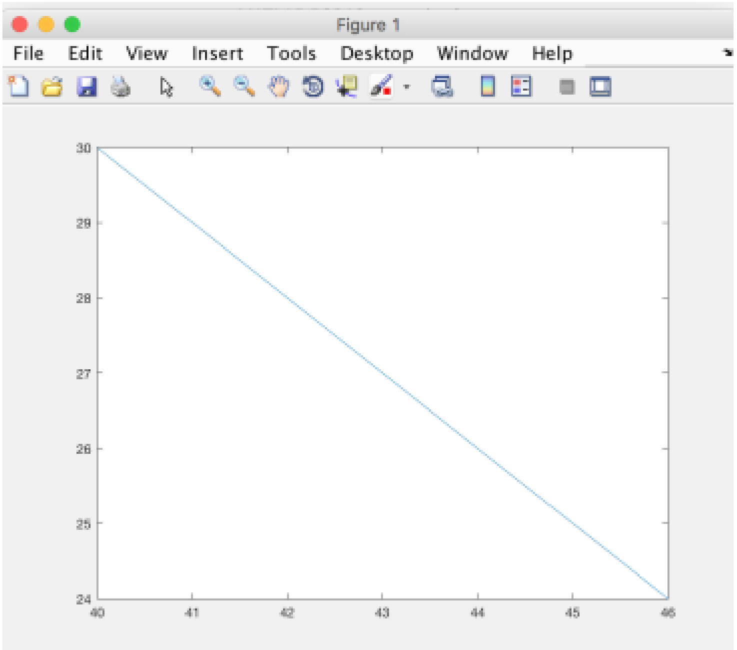
Plot for Threshold vs False Positive



Plot for Threshold vs True Positive



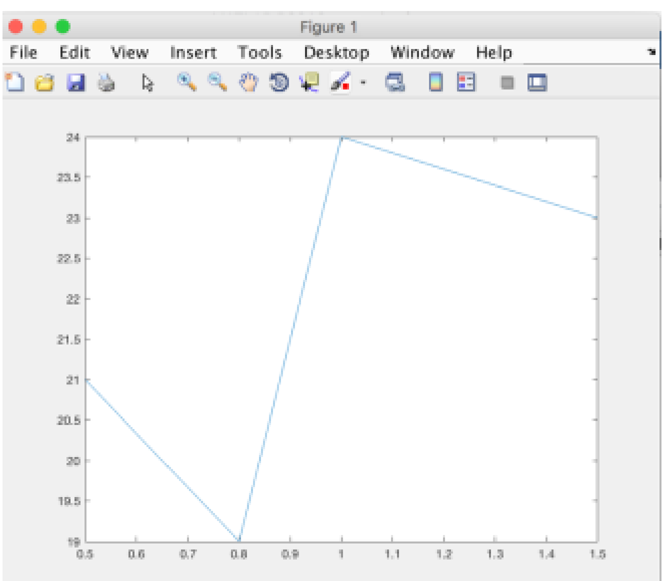
Plot for True Positive vs False Positive



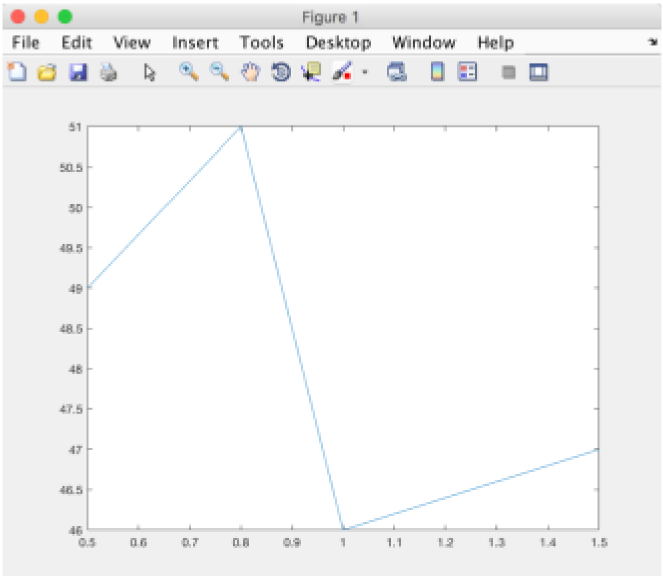
The above three plots are for MOPS descriptor feature extraction. The Thresholds used were 0.5, 0.8, 1, 1.5. The best threshold turned out to be 0.8. The true positives turned out to be maximum at threshold= 0.8 and they were less when the value varied. The plots for true positive and threshold was a pretty good evidence that 0.8 was the best threshold to use. The other plot was false positive vs true positive also showed pretty good evidence that the best threshold was 0.8.

SIFT:

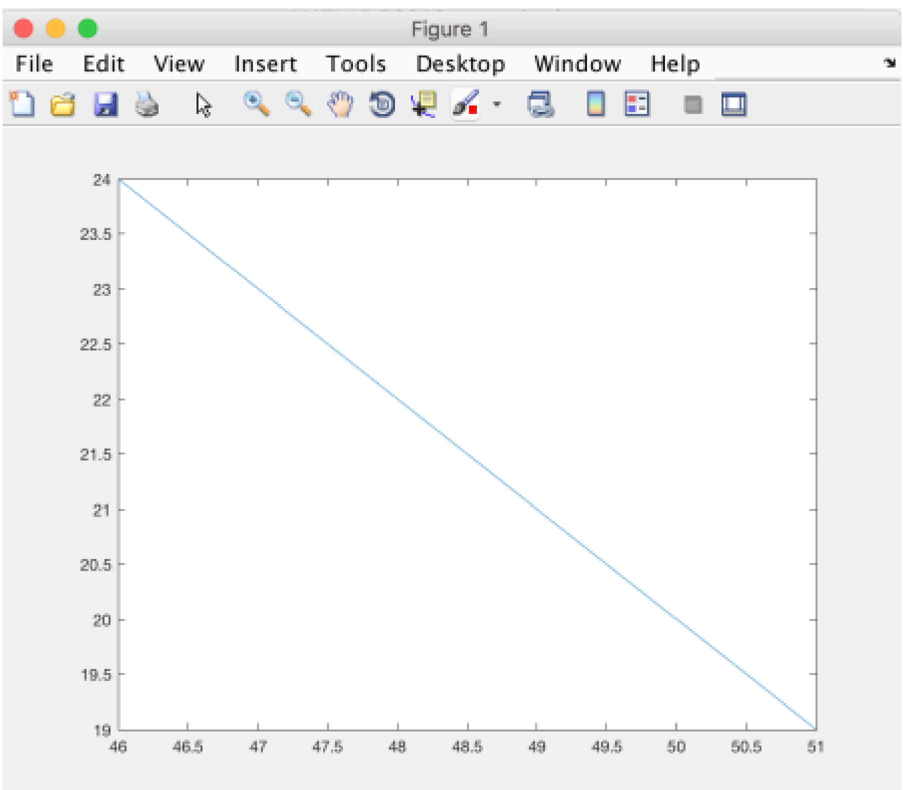
Plot for Threshold vs False Positive



Plot for Threshold vs True Positive



Plot for True Positive vs False Positive



The best threshold to use for SIFT was 0.8. The above three plots show that the different values for the threshold were chosen and the output. The plot for was between false positive and true positive. Threshold was implemented in SIFT feature description. The best threshold used was determined to be 0.8 mainly because from the plot it was determined that the value spiked and gave better true positives and less false positives.

The MOPS threshold was much better a descriptor SIFT descriptor because it detects large temporal values as well as different view points, scales and orientations. MOPS was able to return the same exact image as the original image.

The Source Code was Computationally expensive, thus had to resort to comparing these images for a single pair of images to detect the Confusion Matrix. The Source Code was able to detect the features for SIFT and for MOPS but was not able to compute and plot the features.

SOURCE CODE :

```
%% Creating Database of Image, Checkerboard and 5 New Images.
%imageDB{1}=imread('Image.bmp');
I=imread('checkerboard.jpg');
I=rgb2gray(I);
n=0;
for i = 0:5:60
    n=n+1;
   imageDB{n} = imrotate(I,i);
   I=double(I);
end
for i=1:13
   for j=0:10:30
       n=n+1;
       I = I - min(I(:));
   I = I / max(I(:));
    %/ Add noise to image
   v = var(I(:))/(10^(j/10));
   imageDB{n}  =imnoise(imageDB{i},'gaussian',0,v);

    end
end
imageDB{66}=im2uint8(imread('image1.jpg'));
imageDB{67}=im2uint8(imread('image2.jpg'));
imageDB{68}=im2uint8(imread('image3.jpg'));
imageDB{69}=im2uint8(imread('image4.jpg'));
imageDB{70}=im2uint8(imread('image5.jpg'));
imageDB{66}=rgb2gray(imageDB{66});
imageDB{67}=rgb2gray(imageDB{67});
imageDB{68}=rgb2gray(imageDB{68});
imageDB{69}=rgb2gray(imageDB{69});
imageDB{70}=rgb2gray(imageDB{70});
n=70;
% for image1.jpg
for i = 0:5:60
    n=n+1;
   imageDB{n} = imrotate(imageDB{66},i);
   imageDB{66}=double(imageDB{66});
end
for i=71:83
   for j=0:10:30
       n=n+1;
       imageDB{66}  = imageDB{66} - min(imageDB{66}(:));
   imageDB{66}  = imageDB{66} / max(imageDB{66}(:));
    %/ Add noise to image
   v = var(imageDB{66}(:))/(10^(j/10));
   imageDB{n}  =imnoise(imageDB{i},'gaussian',0,v);

    end
end
% For image2.jpg
for i = 0:5:60
    n=n+1;
   imageDB{n} = imrotate(imageDB{67},i);
   imageDB{67}=double(imageDB{67});
end
for i=136:148
```

```
    for j=0:10:30
        n=n+1;
        imageDB{67} = imageDB{67} - min(imageDB{67}(:));
    imageDB{67} = imageDB{67} / max(imageDB{67}(:));
     %/ Add noise to image
    v = var(imageDB{67}(:))/(10^(j/10));
    imageDB{n} = imnoise(imageDB{i},'gaussian',0,v);

    end
end
% For image3.jpg
for i= 0:5:60
    n=n+1;
    imageDB{n}=imrotate(imageDB{68},i);
    imageDB{68}=double(imageDB{68});
end
for i=201:213
    for j=0:10:30
        n=n+1;
        imageDB{68} = imageDB{68} - min(imageDB{68}(:));
    imageDB{68} = imageDB{68} / max(imageDB{68}(:));
     %/ Add noise to image
    v = var(imageDB{68}(:))/(10^(j/10));
    imageDB{n} = imnoise(imageDB{i},'gaussian',0,v);
    end
end
% For image4.jpg
for i= 0:5:60
    n=n+1;
    imageDB{n}=imrotate(imageDB{69},i);
    imageDB{69}=double(imageDB{69});
end
for i=266:278
    for j=0:10:30
        n=n+1;
        imageDB{69} = imageDB{69} - min(imageDB{69}(:));
    imageDB{69} = imageDB{69} / max(imageDB{69}(:));
     %/ Add noise to image
    v = var(imageDB{69}(:))/(10^(j/10));
    imageDB{n} = imnoise(imageDB{i},'gaussian',0,v);
    end
end
% For image5.jpg
for i= 0:5:60
    n=n+1;
    imageDB{n}=imrotate(imageDB{70},i);
    imageDB{70}=double(imageDB{70});
end
for i=331:343
    for j=0:10:30
        n=n+1;
        imageDB{70} = imageDB{70} - min(imageDB{70}(:));
    imageDB{70} = imageDB{70} / max(imageDB{70}(:));
     %/ Add noise to image
    v = var(imageDB{70}(:))/(10^(j/10));
    imageDB{n} = imnoise(imageDB{i},'gaussian',0,v);

    end
end

%% Problem 2 (Harriss Detector, Fostner Harriss and Adaptive Non Maximal Suppression.)
alpha = 0.04;  %%constant for forstner harris metric
%%harris detector derivative kernels
harx = [-2,-1,0,1,2];
hary = [-2;-1;0;1;2];
%%3X3 sobel kernels
xKern = [-1,0,1;-2,0,2;-1,0,1];
yKern = [-1,-2,-1;0,0,0;1,2,1];
for i=1:395
    n=n+1;
    xi=zeros(size(imageDB{i}));
    yi=zeros(size(imageDB{i}));
    xi2=zeros(size(imageDB{i}));
    xy=zeros(size(imageDB{i}));
    yi2=zeros(size(imageDB{i}));
    imageDB{i}=padarray(imageDB{i},5);
    for(j=3:1:size(imageDB{i},1)-2)
```

```
        for(k=3:1:size(imageDB{i},2)-2)
        imageDB{n}(j-2,k-2)=sum(sum([double(imageDB{i}(j,k-2:k+2))].*harx));
        end
    end
    for(j=3:1:size(imageDB{i},1)-2)
        for(k=3:1:size(imageDB{i},2)-2)
            imageDB{n+1}(j-2,k-2)=sum(sum([double(imageDB{i}(j-2:j+2,j))].*hary));
        end
    end
    ix2=imageDB{n}.^2;
    iy2=imageDB{n+1}.^2;
    ixiy=xi.*yi;
    %ix2=gauss(ix2);
    %iy2=gauss(iy2);
    %ixiy=gauss(ixiy);
    imageDB{n+2}=zeros(size(xi));

    for(j=1:1:size(xi,1))
        for(k=1:1:size(xi,2))
            AC=[xi2(j,k),ixiy(j,k);ixiy(j,k),yi2(j,k)];
%               AC=gauss(AC);  %not going to apply a gaussian filter to a 4X4
            imageDB{n+2}(j,k)=det(AC)-alpha*trace(AC)^2;
        end
    end

%fhmetric(fhmetric~=0)=255;  %threshold so it is easier to view

%figure();
%imshow(uint8(imageDB{n+2}))
%title('Forstner-Harris Metric')
amaxes=zeros(size(imageDB{n+2}));
origmetric=imageDB{n+2};
imageDB{n+2}=padarray(imageDB{n+2},5);
for(j=3:1:size(imageDB{n+2},1)-2)
    for(q=3:1:size(imageDB{n+2},2)-2)
         neighborhood=imageDB{n+2}(j-1:j+1,q-1:q+1);
         neighborhood=reshape(neighborhood,1,[]);
         neighborhood=sort(neighborhood,'descend');
        if (neighborhood(1)==imageDB{n+2}(i,j))
            amaxes(j-2,q-2)=neighborhood(1)-neighborhood(2);
%               amaxes(i-2,j-2)=255;
        else
            amaxes(j-2,q-2)=0;
        end
    end
end

locmaxes=amaxes;
midpoint=sum(amaxes(:))/nnz(amaxes);
midpoint=midpoint^0.4;
amaxes(amaxes<=midpoint)=0;
amaxes(amaxes>midpoint)=255;

%figure();
%imshow(uint8(amaxes))
%title('Local maxes with threshold')

%% adaptive non maximal suppression %%
origmetric=padarray(origmetric,5);
q=.01;
for j=101:1:size(origmetric,1)-100
    for k=101:1:size(origmetric,2)-100
        if locmaxes(j-100,k-100)~=0
            for s=2:1:100
                if checkRadius(q*origmetric(j,k),j,k,s)
                    imageDB{n+3}(j-100,k-100)=s;
                    break
                else
                    imageDB{n+3}(j-100,k-100)=101;
                end
            end
        else
            imageDB{n+3}(j-100,k-100)=0;
        end
    end
end
totpix=nnz(imageDB{n+3});
```

```
testit=101;
while(nnz(imageDB{n+3})>.9*totpix)
   imageDB{n+3}(imageDB{n+3}>testit)=0;
    for j=1:1:size(imageDB{n+3},1)
        for k=1:1:size(imageDB{3},2)
               if imageDB{n+3}(j,k)>testit
                imageDB{n+3}(j,k)=0;
               end
        end
     end
testit=testit-1;
end
imageDB{n+3}(imageDB{n+3}~=0)=255;
end
```

This code was able to create a data base of the 65 images of the various rotated versions and the images with the noise and also images with noise+database.

The code then created 395 images after adding the 5 images and adding noise, and rotating for all the versions.

This made the program initially computationally expensive.

Once this was implemented, the Feature Detection was implemented.

Confusion matrix Code:

If eucledian distance< threshold
True positive value was returned

Else if eucledian distance? Threshold
False positive value was returned.

SIFT

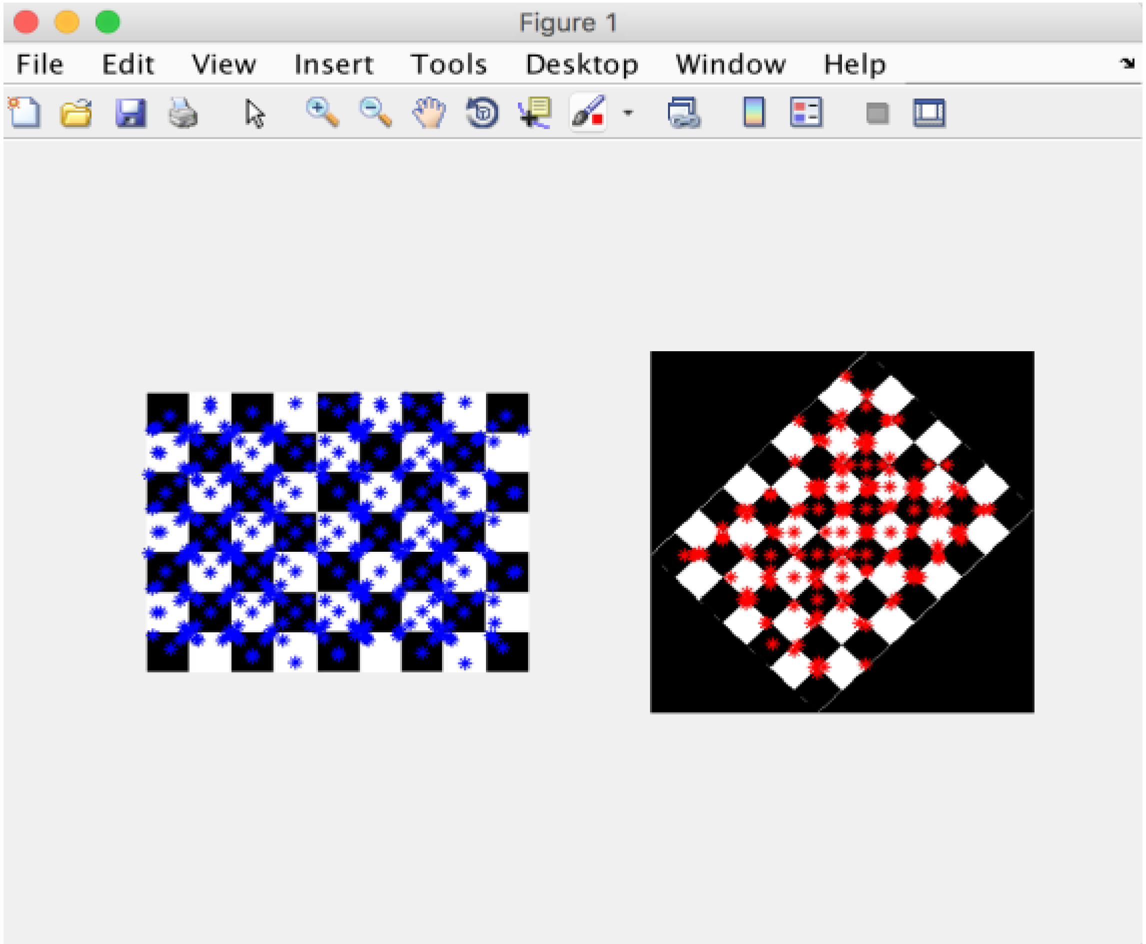


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help

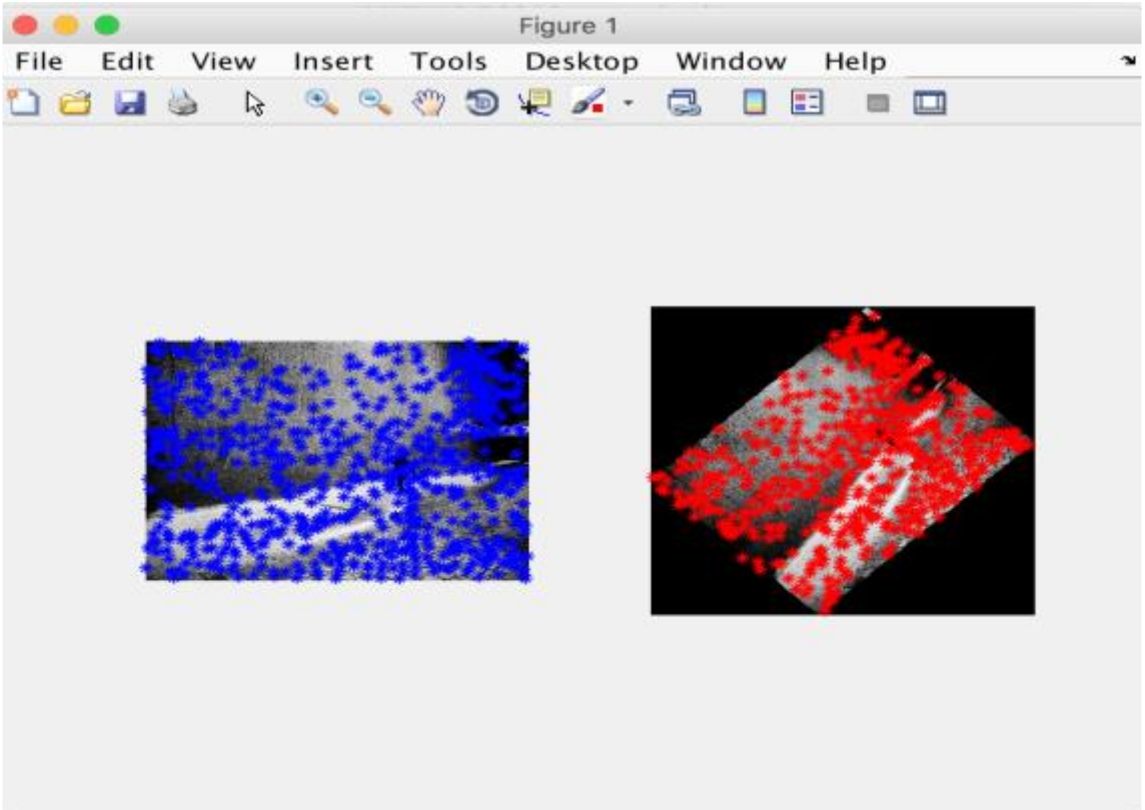


Figure 1
File Edit View Insert Tools Desktop Window Help

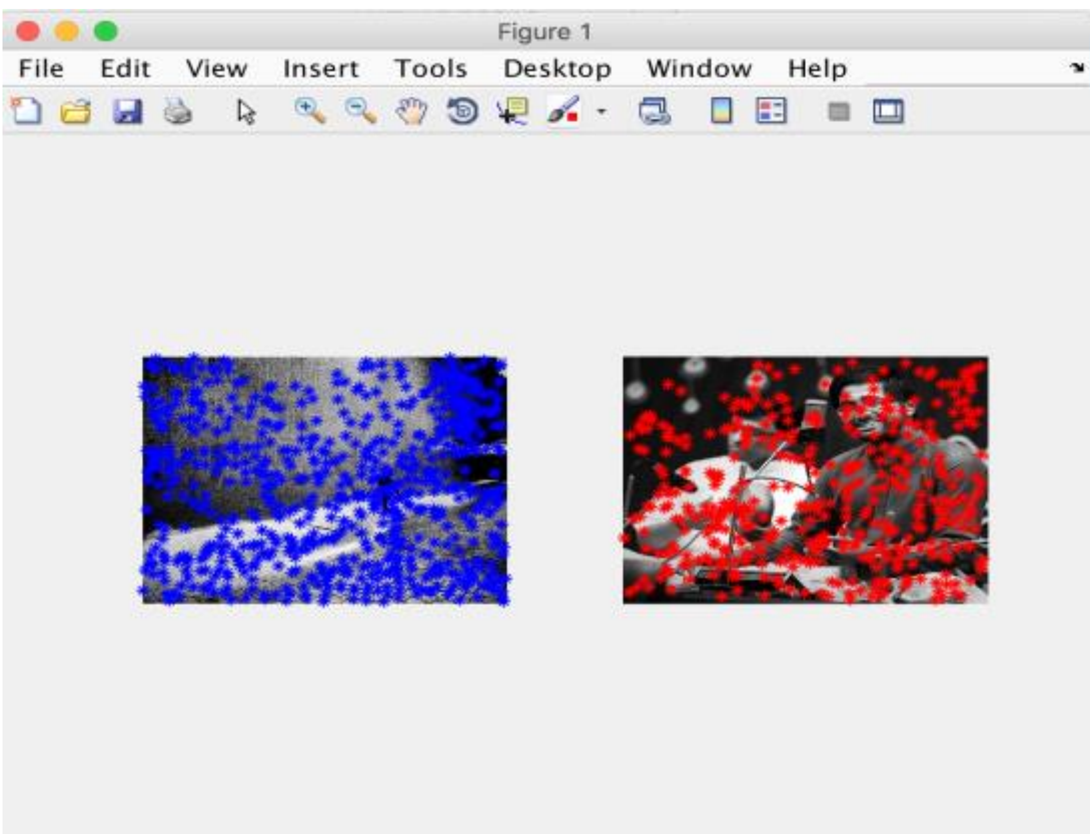


Figure 1
File Edit View Insert Tools Desktop Window Help

The above images was obtained on running the SIFT algorithm.

SIFT Algorithm:

```
I = imread('image1.jpg');
J = imread('image3.jpg');

I = single(rgb2gray(I)); % Conversion to single is recommended
J = single(rgb2gray(J)); %in the documentation

[F1 D1] = vl_sift(I);
[F2 D2] = vl_sift(J);

% Where 1.5 = ratio between euclidean distance of NN2/NN1
[matches score] = vl_ubcmatch(D1, D2, 0.5);

subplot(1,2,1);
imshow(uint8(I));
hold on;
plot(F1(1,matches(1,:)),F1(2,matches(1,:)),'b*');

subplot(1,2,2);
imshow(uint8(J));
hold on;
plot(F2(1,matches(2,:)),F2(2,matches(2,:)),'r*');
```

MOPS Algorithm:

Normalized 8*8 intensity patches were used to detect feature points. The fostner harris metric was applied to this 8*8 matrix which was positioned on every feature that was detected. The kernel used was 1/16[1 4 6 4 1]. The output was more appealing using MOPS due to its orientational individuality. The orientational patches were spaced at a 5 pixel distance by using image pyramids. That avoided aliasing.

```
b = imgaussfilt(A);
b = imgaussfilt(A);
b = imgaussfilt(A);
b = imgaussfilt(A);
%% constants %%
alpha = 0.04; %%constant for forstner harris metric
%%harris detector derivative kernels
hxKern = [-2,-1,0,1,2];
hyKern = [-2;-1;0;1;2];
%%3X3 sobel kernels
xKern = [-1,0,1;-2,0,2;-1,0,1];
yKern = [-1,-2,-1;0,0,0;1,2,1];

%% blur my image as a preprocessing tool to make the derivatives nicer %%



%% pad then take the x and y derivatives respectively %%
ix=zeros(size(b));
iy=zeros(size(b));
ix2=zeros(size(b));
ixiy=zeros(size(b));
iy2=zeros(size(b));

b=pad(b,'m');
for(i=3:1:size(b,1)-2)
    for(j=3:1:size(b,2)-2)
        ix(i-2,j-2)=sum(sum([double(b(i,j-2:j+2))].*hxKern));
    end
end
for(i=3:1:size(b,1)-2)
    for(j=3:1:size(b,2)-2)
        iy(i-2,j-2)=sum(sum([double(b(i-2:i+2,j))].*hyKern));
```

```
    end
end

figure();
imshow(ix);
title('x-derivative')
figure();
imshow(iy);
title('y-derivative')
ix2=ix.^2;
iy2=iy.^2;
ixiy=ix.*iy;
ix2=gauss(ix2);
iy2=gauss(iy2);
ixiy=gauss(ixiy);
fhmetric=zeros(size(ix));

for(i=1:1:size(ix,1))
    for(j=1:1:size(ix,2))
        AC=[ix2(i,j),ixiy(i,j);ixiy(i,j),iy2(i,j)];
%           AC=gauss(AC);  %not going to apply a gaussian filter to a 4X4
        fhmetric(i,j)=det(AC)-alpha*trace(AC)^2;
    end
end
```