Final Assignment

March 27, 2021

Jaya Parmar, MATLAB filenamesAppendix 1(Main algorithm) - *final.m.* Appendix 2 - *sift.m.* Appendix 3 - *RGBcolorpreprocessing.m.* Appendix 4 - *temperature.m.* Appendix 5 - *entropy.m*

1 Introduction

Our aim is to make an algorithm to recognize one specific object in multiple imagery using techniques learnt in this course.

2 Method

2.1 Read in the infrared (IR) and visual (RGB) images

We use 4 computer screens; 6 cell phone and 5 power outlet images for our work. The project work requests to include 5 circuit board images in the study but those were not found in the image bank hence we restrict out study to the 15 images in each IR and RGB category.

The IR images can be read in as below (replace with the location where you have saved the images)

$$location_{IR} = ('C:\backslash\backslash CIP\&MachineVision\backslash Finalproject\backslash Training\backslash IR') \tag{1}$$

Then make a reference variable ds_{IR} to the location where images are stored.

$$ds_{IR} = imageDatastore(location_{IR}) \tag{2}$$

Similarly save the RGB images in below location

$$location_{RGB} = ('C:\backslash\backslash CIP\&MachineVision\backslash Finalproject\backslash Training\backslash RGB') \tag{3}$$

and use a pointer variable ds_{RGB}

$$ds_{RGB} = imageDatastore(location_{RGB}) \tag{4}$$

Read in all images in an IR and RGB variable as below

$$imgs_{IR} = readall(ds_{IR}) \tag{5}$$

$$imgs_{RGB} = readall(ds_{RGB}) \tag{6}$$

Ask the user to choose an image from the 15 image data with a input prompt as below

$$input = inputdlg('Choose\,image\,number\,between\,1\,to\,15') \tag{7}$$

Save the input in an array format

$$im = cell2mat(input) \tag{8}$$

Since the user input is saved as a character, we convert it to number as below

$$im = str2num(im) \tag{9}$$

2.2 Pre-processing operation

1. **IR image pre-processing**
   Now convert the user selected IR images into double and normalize them.

   $$imIR = double(imgs_{IR}\{im\})/255 \tag{10}$$

   and start pre-processing them by calling the IRpreprocessing function as below

   $$r_{IR} = IRpreprocessing(imIR) \tag{11}$$

   In the IRpreprocessing function, start by seperating the three channels of the IR image

   function$r_{IR} = IRpreprocessing(imIR)$

   $$redIR = imIR(:,:,1) greenIR = imIR(:,:,2) blueIR = imIR(:,:,3) \tag{12}$$

   Plot these channels

   $$figure(1), subplot(2,2,1), imshow(redIR), title('IRRedchannel') \tag{13}$$

   $$subplot(2,2,2), imshow(greenIR), title('IRGreenchannel') \tag{14}$$

   $$subplot(2,2,3), imshow(blueIR), title('IRBluechannel') \tag{15}$$

   $$subplot(2,2,4), imshow(imIR), title('IROriginalImage'), impixelinfo \tag{16}$$

$$sgtitle('IROriginalImageanditschannels') \quad (17)$$

See the results in **Figure 1**. The red channel best captures the object for segmentation purpose. This is because the object to be identified is a heated object which is represented by red as seen 'visually' in an IR image.

One can cross verify this fact from the temperature bar provided in the IR images. Different shades of red represent the highest temperature in most of cell phone and computer screen images.

There are exceptions like image number 10 (computer screen) or the power outlet images in our dataset. We will see these exceptions more clearly in the binarized images where the object to be detected will merge with the background.

We need to binarize the image to start segmentation. We will use graythresh to find the luminance level above which the value is 1 and below it, the value is 0.

$$red_{IR} = graythresh(redIR) green_{IR} = graythresh(greenIR) blue_{IR} = graythresh(blueIR) \quad (18)$$

and binarize the channels

$$r_{IR} = im2bw(redIR, red_{IR}) \quad (19)$$

$$g_{IR} = im2bw(greenIR, green_{IR}) \quad (20)$$

$$b_{IR} = im2bw(blueIR, blue_{IR}) \quad (21)$$

We will also see the combined effect of binarized channels by

$$IR_{combined} = and(and(r_{IR}, g_{IR}), b_{IR}) \quad (22)$$

and plot the results

$$figure(2), subplot(2,2,1), imshow(r_{IR}), title('IRRedchannel') \quad (23)$$

$$subplot(2,2,2), imshow(g_{IR}), title('IRGreenchannel') \quad (24)$$

$$subplot(2,2,3), imshow(b_{IR}), title('IRBluechannel') \quad (25)$$

$$subplot(2,2,4), imshow(IR_{combined}), title('IRCombinedeffect') \quad (26)$$

sgtitle('IR Binarized channels')

Results in **Figure2** confirm that the red channel best represents the object to be segmented.

Now pick the exception image number 10 as user input and see that the object merges with the background in the binarized image. This is because it comes from a different color palette. We will talk about more this exception in the *discussion* section.

We will also discuss temperature values based preprocessing in the *discussion* section.

But for now, we pass on the red channel variable r_{IR} back to the main body of the algorithm.

2. **RGB image pre-processing**
In a similar manner, we will also check if any of the color channels best represents the object in RGB image.

We make RGBpreprocessing function to do this which is called next in our algorithm

As before first convert the RGB images to double and normalize them

$$imRGB = double(imgs_{RGB}\{im\})/255 \tag{27}$$

Now we call our RGB preprocessing function

$$bfill = RGBpreprocessing(imRGB) \tag{28}$$

See **Appendix 3** for the seperate Matlab file RGBpreprocessing.m

function $bfill = RGBpreprocessing(imRGB)$

$$imrgb2gray = rgb2gray(imRGB) \tag{29}$$

$$redRGB = imRGB(:,:,1) greenRGB = imRGB(:,:,2) blueRGB = imRGB(:,:,3) \tag{30}$$

As seen above we include the gray channel to see how it represents our object of interest

We binarize the image channels by finding a threshold level using graythresh

$$red_{RGB} = graythresh(redRGB) green_{RGB} = graythresh(greenRGB) blue_{RGB} = graythresh(blueRGB) \tag{31}$$

$$gray_{RGB} = graythresh(imrgb2gray) \tag{32}$$

binarize

$$r_{RGB} = im2bw(redIR, red_{RGB}) \tag{33}$$

$$g_{RGB} = im2bw(greenIR, green_{RGB}) \tag{34}$$

$$b_{RGB} = im2bw(blueIR, blue_{RGB}) \tag{35}$$

$$gray = im2bw(imrgb2gray, gray_{RGB}) \tag{36}$$

We will also see the combined effect of binarized channels by

$$RGB_{combined} = and(and(r_{RGB}, g_{RGB}), b_{RGB}) \tag{37}$$

and plot the results

$$figure(3), subplot(2,3,1), imshow(r_{RGB}), title(RGBRedchannel') \tag{38}$$

$$subplot(2,3,2), imshow(g_{RGB}), title('RGBGreenchannel'); \tag{39}$$

$$subplot(2,3,3), imshow(b_{RGB}), title('RGBBluechannel') \tag{40}$$

$$subplot(2,3,4), imshow(RGB_{combined}), title('RGBCombinedeffect') \tag{41}$$

$$subplot(2,3,5), imshow(gray), title('RGBGrayimage') \tag{42}$$

$$subplot(2,3,6), imshow(bfill), title('RGBcombinedwithholesfilled'), impixelinfo \tag{43}$$

sgtitle('RGB Binarized channels')

Results in **Figure3** did not give a clear decision making as IR image channels. One could choose between $RGB_{combined}$ and RGB Gray image. We replaced image 1 with a few other RGB images to understand which channel best represents the object but the decision was still unclear.

Hence we decided to try another approach, edge based segmentation, to directly segment the RGB image instead.

3. **RGB edge based segmentation**
Pass on the imRGB to RGBedgesegmentation function as below

$$bwedge = RGBedgesegmentation(imRGB) \tag{44}$$

function $bfill = RGBedgesegmentation(imRGB)$

$$imrgb2gray = rgb2gray(imRGB) \tag{45}$$

$$bwedge = edge(imrgb2gray,' canny') \tag{46}$$

Plot the edges detected using Canny algorithm in **Figure4**

$$figure(4), imshow(bwedge), t = sgtitle('RGBEdgebasedsegmentation'); t.Color =' w' \tag{47}$$

Our RGB image is now ready for the descriptor creation based on above edge segmentation.

Next step is to segment the pre-processed IR image

2.3 IR Segmentation

With the binary images in hand, we can proceed with the IR segmentation.

Let us start by calling the segmentation function first for the IR image.

$$bwop_{IR} = segmentation(r_{IR}) \tag{48}$$

function $bwop = segmentation(image)$

$$se = strel('rectangle', [10\,10]) \tag{49}$$

The structuring element above will make the smaller objects disappear from the image. Since all objects in our data are rectangles, we use a rectangular structuring element. Our object holds its shape with a 10X10 structuring element. This size can be changed as per user preference and output results.

Let us open the image and return it to main body to see the results

$$bwop = imopen(image, se) \tag{50}$$

$$figure(5), imshow(bwop_{IR}), t = sgtitle('IRsegmentedimage'); t.Color =' w' \tag{51}$$

See **Figure5** in results section show the opened image. See the temperature bar and FLIR logo from Figure 2 have disappeared now due to size of structuring element.

2.4 Descriptors

This is one of the most important part of the algorithm where we 'describe' the features of the image.

At this point, we have edge information extracted from the RGB image and the temperature range (via red channel) of the IR image.

We use common descriptors to identify the object in both RGB and IR images.

Our chosen descriptors are Area and Bounding Box. The first descriptor Area will collect the areas of objects of different shapes and sizes in images. The second descriptor 'Bounding Box' will recognize our object of interest.

We start with the IR image.

$$biggest_{IR} = descriptors(bwop_{IR}); \tag{52}$$

This will call the descriptors function with input parameter as the opened IR image. The output of the function will be $biggest_{IR}$

The purpose of this function is to 'recognize' the object. Object in all images have one thing in common. They are the biggest in area. This defines our first descriptor 'Area'. The project also asks for a way to 'visualize' the identified object. This makes us define second descriptor 'Bounding Box' which will highlight the identified object.

function $biggest_{object} = descriptors(bwop)$

We start by labelling the connected components of the binary opened image

$$[labeled, numobj] = bwlabel(bwop, 8); \tag{53}$$

We feed these labels to the regionprops function and define our descriptors as input parameters to get the regions with these features.

$$regions = regionprops(labeled,' Area',' BoundingBox') \tag{54}$$

Let us seperate out all areas from the regions

$$allAreas = [regions.Area] \tag{55}$$

and sort them in descending order since our goal is to find the biggest area

$$[sortedAreas, sortedIndexes] = sort(allAreas,' descend') \tag{56}$$

$$biggest_{object} = regions(sortedIndexes(1)) \tag{57}$$

$biggest_{object}$ is our object of interest.

Let us go back to the main algorithm and display the bounding box there.

figure(6), imshow(imIR), sgtitle('Recognized IR object')
hold on

$$rectangle('Position', biggest_{IR}.BoundingBox,' EdgeColor',' r',' LineWidth', 3) \tag{58}$$

hold off

See **Figure6** in results section. The identified object visualized with a red outline.

We repeat the process for RGB image by calling the descriptors function again

$$biggest_{RGB} = descriptors(bwop_{RGB}) \tag{59}$$

Plot the results.

figure(7), imshow(imRGB), sgtitle('Recognized RGB object')
hold on

$$rectangle('Position', biggest_RGB.BoundingBox,' EdgeColor',' r',' LineWidth', 3) \tag{60}$$

hold off

See **Figure7** in results section. The identified object visualized with a red outline.

At this point, we can see that we have verified that our descriptors work on the edge information from the RGB image as well as temperature information from the IR image.

2.5 Image Registration

Image Registration is a process of overlaying two or more images of the same scene so that the same scene objects from different images are aligned.

We have our image number 1 as the reference image. We call it base

$$base = imgs_{RGB}\{im\} \tag{61}$$

Note that we are using RGB images for image registration but one can also use IR images.

The goal is to make an algorithm that picks other images within the same class (computer screen in our example) and register them.
This mean first we need to sort our images into their classes namely

$$compscreen = [1, 2, 3, 10]; cs = length(compscreen) \tag{62}$$

where image number 1,2,3 and 10 are the computer screen images in our data bank and cs is the number of computer screen images.

Similarly making the other two classes

$$cellphone = [4, 5, 6, 7, 8, 9]; cl = length(cellphone) \tag{63}$$

$$powerout = [11, 12, 13, 14, 15]; po = length(powerout) \tag{64}$$

Now if the base image is a compscreen class, the floating images should be picked from the same class and the imageregistration function should be called.
See below
if im == 1 OR im == 2 OR im==3 OR im==10
for cidx = 1:cs
C = compscreen(cidx)
$float = imgs_{RGB}\{C\}$
imageregistration(base,float)
end

The function imageregistration will be explained a little later but first lets finish the if condition for the classification

elseif im == 4 OR im == 5 OR im== 6 OR im== 7 OR im== 8 OR im== 9
$base = imgs_{IR}\{im\}$
for lidx = 1:cl
L = cellphone(lidx)
$float = imgs_{RGB}\{L\}$
imageregistration(base,float)
end

See above that we are changing the base image to IR. This is because image registration function could not find matches between the base and float cellphone images.

elseif im == 11 OR im == 12 OR im== OR im==13 OR im==14 OR im== 15
for pidx = 1:po
P = powerout(pidx)
$float = imgs_{RGB}\{P\}$
imageregistration(base,float)
end

else disp('Error') end

function imageregistration(base,float) Image registration process is divided into four steps

(a) Feature detection

(b) Feature matching

(c) Transformation model estimation

(d) Image resampling and transformation

We discuss these steps one by one below.

(a) **Feature detection**
We start by sending the base and float image to the sift algorithm. This algorithm will detect the features in each image by dividing the size of image into K 128-element feature descriptors.
The formation of 128 element descriptor is briefly described below.

- First, an orientation histogram of 8 bins is created on 4X4 subregion. This histogram is computed from the magnitude and orientation values of the sample subregion.

- 4X4 =16 such histograms of 8 bins each are made tol cover the entire 16X16 region.

- The descriptor is a vector of all values of these histograms. i.e. a vector of 16 histograms of 8 bins = 128 elements.

- In order to achieve orientation invariance, coordinates of this descriptor and gradient orientations are rotated relative to keypoint orientation.

- The feature descriptor vector is then normalized to unit length to achieve invariance to affine changes in the illumination.

- The final result is feature descriptors which are robust and distinctive.

The sift.m in Appendix 2 returns three output parameters -

- image - which is the image in double format.

- descriptor - a KX128 matrix where each row of the 128 element descriptor gives an invariant descriptor for one of the K keypoints. The 128 element descriptor is normalized to unit length.
- locs - a KX4 matrix, in which each row has the 4 values for a keypoint location (row, column, scale, orientation). The orientation is in the range [-PI, PI] radians.

We three returned parameters for the base image are baseImg, bdesc and bpos as below
$[baseImg, bdesc, bpos] = sift(base)$

Similarly the returned parameters for the float image are floatImg, fdesc and fpos
$[floatImg, fdesc, fpos] = sift(float)$

(b) **Feature matching**
Here we try to match the detected features between base and float images by defining a match threshold called 'distRatio'.
distRatio calculates the proximity of features in the first sample with respect to the second sample.
If the 128 features in the first sample are in 80% proximity of the second sample features than there is a match. We take the coordinates/indices of the first sample number and store in a new row vector match.
On the contrary if the distRatio criteria $vals(1) < distRatio * vals(2)$ is not satisfied then drop the indices of that sample number.

The code for this operation is below
distRatio = 0.8
match=zeros(1,size(bdesc,1))
for i = 1 : size(bdesc,1)
dotprods = bdesc(i,:) * fdesc'
$[vals, indx] = sort(acos(dotprods))$

$if(vals(1) < distRatio * vals(2))$
match(i) = indx(1)

```
else
match(i) = 0
end
end
```

Then we append the returned base and float double images
temp = appendimages(baseImg,floatImg)

and plot Figure 9 with the accepted matches between the baseImg and floatImg

```
figure('Position', [100 100 size(temp,2) size(temp,1)])
colormap('gray')
imagesc(temp)
hold on
cols1 = size(baseImg,2)
matchedDescs=zeros(numel(find(match)),4)
```

See **Figure9** in results section which shows accepted matches between image number 1 and itself.
Similarly **Figure11** shows the accepted matches between image number 1 and 2.
Figure13 shows the accepted matches between image 1 and image 3 and **Figure15** between image 1 and image 10.

We display the matched descriptorsby running loop of size bdesc is run with a condition that if there are non zero matches then make a four column matchedDescs variable where the first two columns are the positions of bpos and last two columns are the positions of fpos where a match was found. See below

```
j=1
for i = 1: size(bdesc,1)
if(match(i) > 0)
line([bpos(i,2) fpos(match(i),2)+cols1], [bpos(i,1) fpos(match(i),1)],
'Color', 'c')
matchedDescs(j,:)=[bpos(i,2) bpos(i,1) fpos(match(i),2) fpos(match(i),1)]
j=j+1
end
end
hold off
num = sum(match > 0)
fprintf('Found %d matches.n', num)
```

(c) **Transformation model estimation**
With the matched points from matched bpos and fpos, we can create

matching point pairs as below

matchedPtsDistorted = [matchedDescs(:,3) matchedDescs(:,4)]
matchedPtsOriginal = [matchedDescs(:,1) matchedDescs(:,2)]

We can estimate a geomatric transformation model which will return a 2-D geomatric transform object tform.
This transformation is based on RANSAC algorithm.
This transform object maps the inliers in matchedPtsDistorted to matchedPtsOriginal.
We ask the function to return these inlierpoints as inlierpoints1 and inlierpoints2.
Lastly, we also ask for a status return as the function returns an error for conditions that cannot produce the results otherwise

[tform,inlierpoints1,inlierpoints2,status] =
estimateGeometricTransform(matchedPtsDistorted,matchedPtsOriginal,'projective')

We have used transformation type 'projective' for greater accuracy.
One could also use 'similarity' transformation type and see the difference inlier keypoints.

We concatenate the inlierpoints 1 and 2 before plotting them

bestMatches=cat(2,inlierpoints1,inlierpoints2)

Code below will plot these bestMatches which are the inliers returned by our ransac algorithm

figure('Position', [100 100 size(temp,2) size(temp,1)])
colormap('gray')
imagesc(temp)
hold on
cols1 = size(baseImg,2)
for i = 1: size(bestMatches,1)
line([bestMatches(i,1) bestMatches(i,3)+cols1], [bestMatches(i,2) best-
Matches(i,4)], 'Color', 'c')
end
hold off
fprintf('Found %d best matches.n', size(bestMatches,1))

Figure10 shows the best matches between image number 1 inlierpoints with its ownself.
Figure12 shows the best matches between image number1 inlierpoints with image number 2 inlierpoints.

Figure14 shows the best matches between image number1 inlierpoints with image number 3 inlierpoints.
Figure16 shows the best matches between image number1 inlierpoints with image number 10 inlierpoints.

(d) **Image resampling and transformation**

Now is the time to display the base and float images. First convert the base image to world coordinates via imref2d

imref=imref2d([size(baseImg,1) size(baseImg,2)])

Later apply geomatric transformation to the image using imwarp. Imwarp transforms float image according to the displacement field of base image.

warpedImg=imwarp(float,tform,'OutputView',imref)

Show the registration result using imtool.

imtool(imfuse(warpedImg,base,'blend'))

Figureimtool1 shows reference (or base) image registered with itself.

Figureimtool2 shows reference image registered with second computer screen RGB image.

Figureimtool3 shows reference image registered with third computer screen RGB image and

Figureimtool4 shows reference image registered with fourth computer screen RGB image.

3 Results

Figure 1

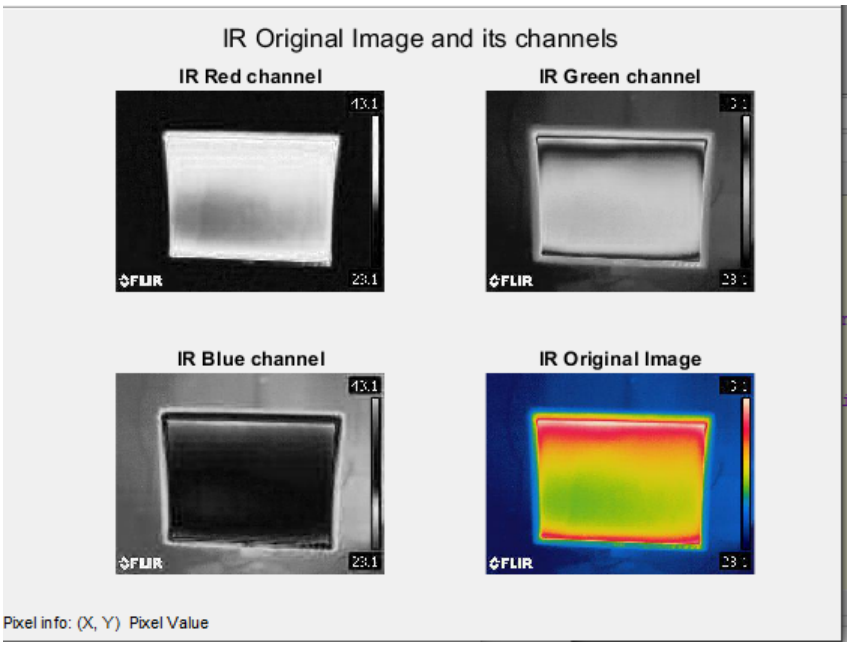


Figure 2

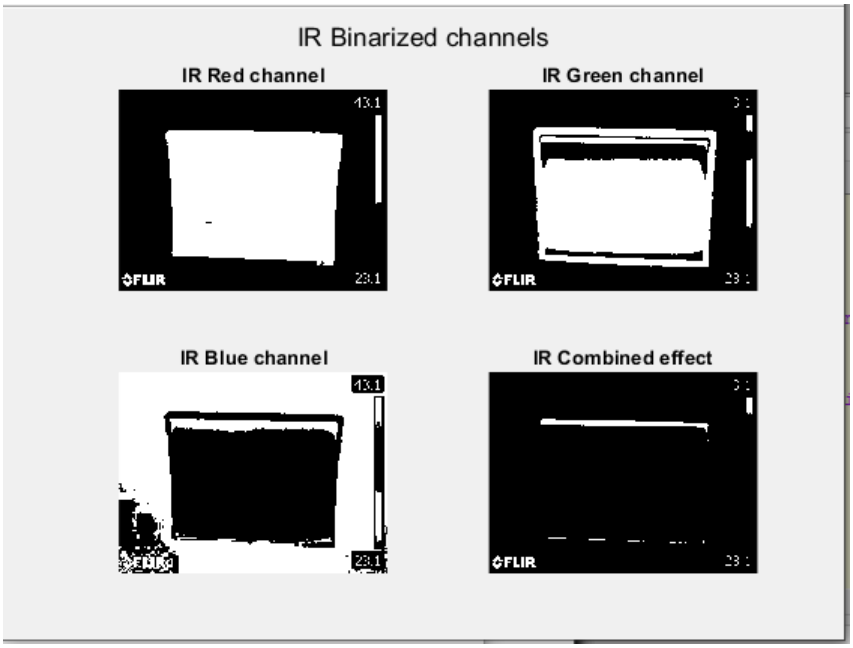


Figure 3

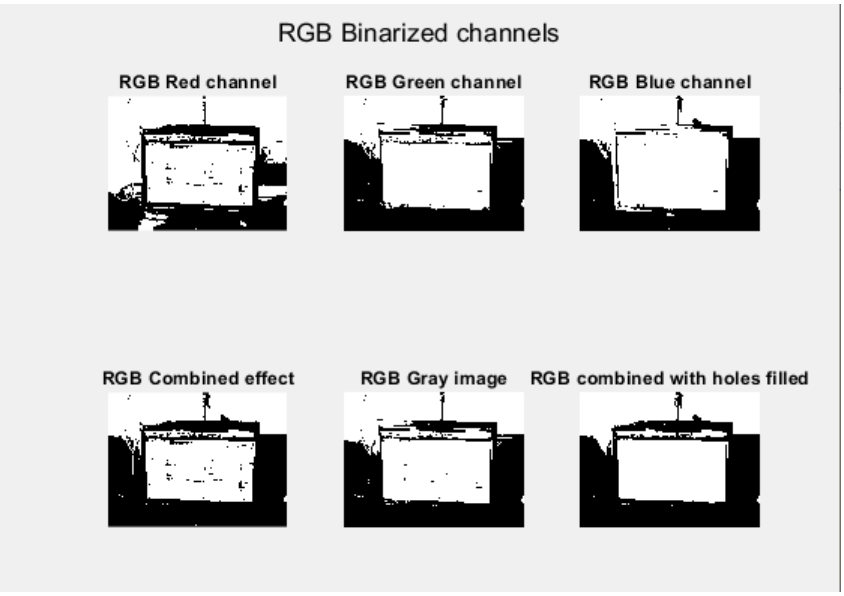


Figure 4

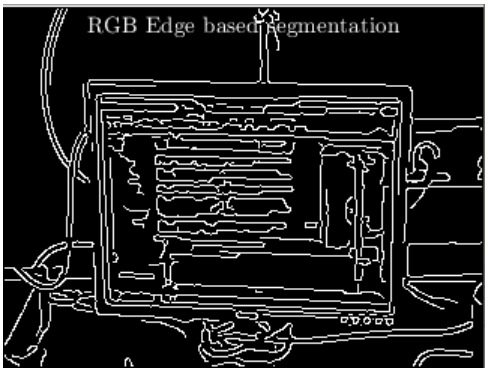


Figure 5

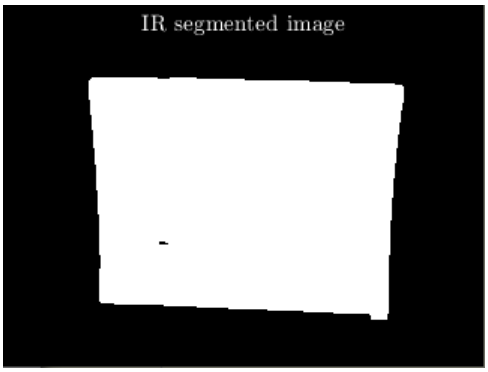


Figure 6

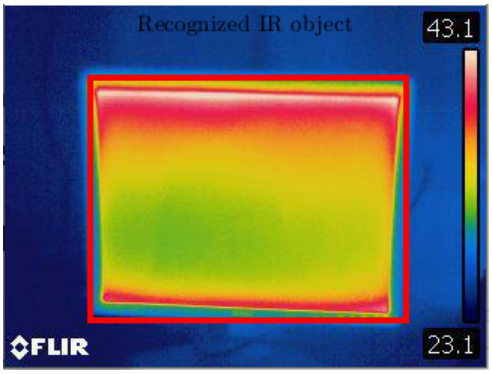


Figure 7

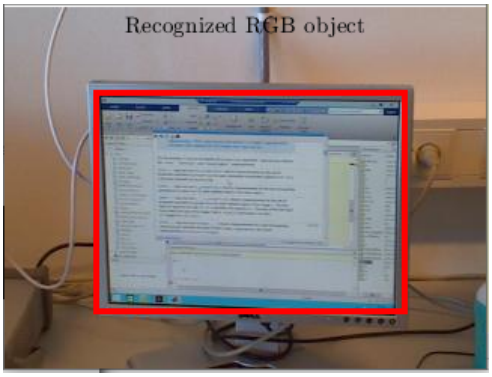


Figure 8 - None
Figure 9 - Lines joining the **accepted** matches between image and itself

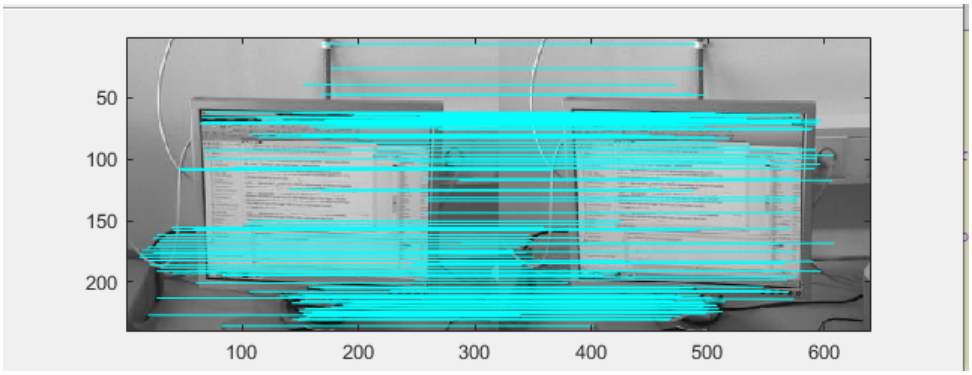


Figure 10 - Lines joining **best** matches between image and itself

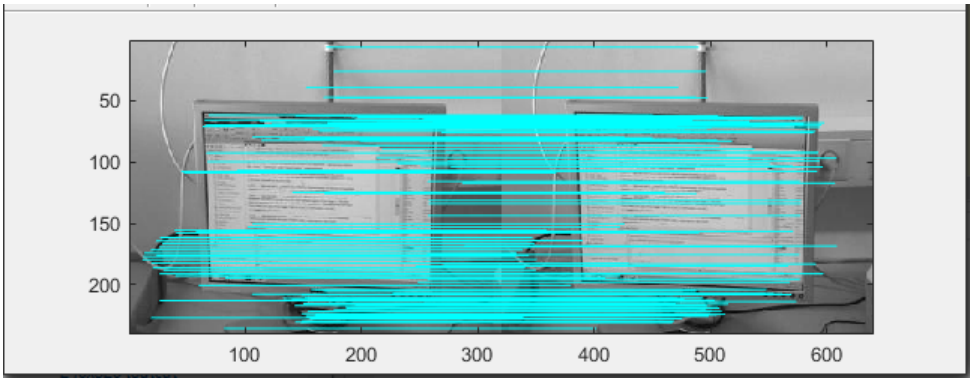


Figure imtool1 - Reference image registered with **itself**

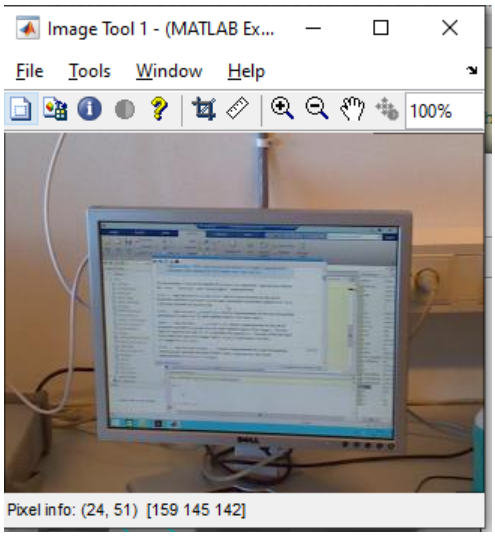


Figure11 - Lines joining the accepted matches between image and **second image**

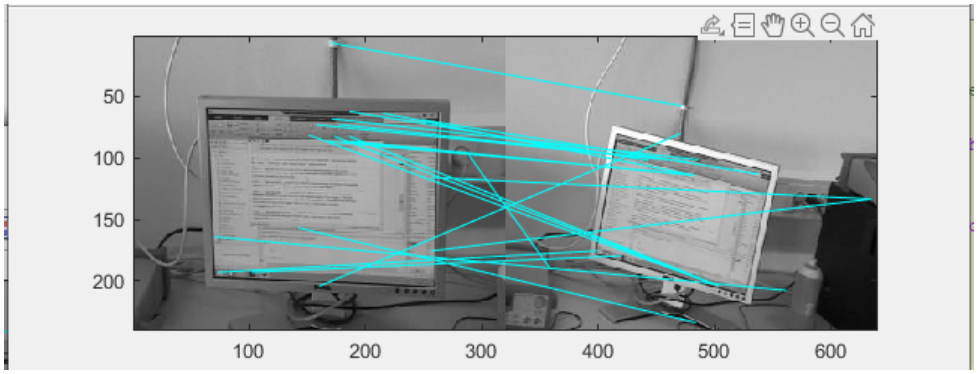


Figure12 - Lines joining best matches between image and **second image**

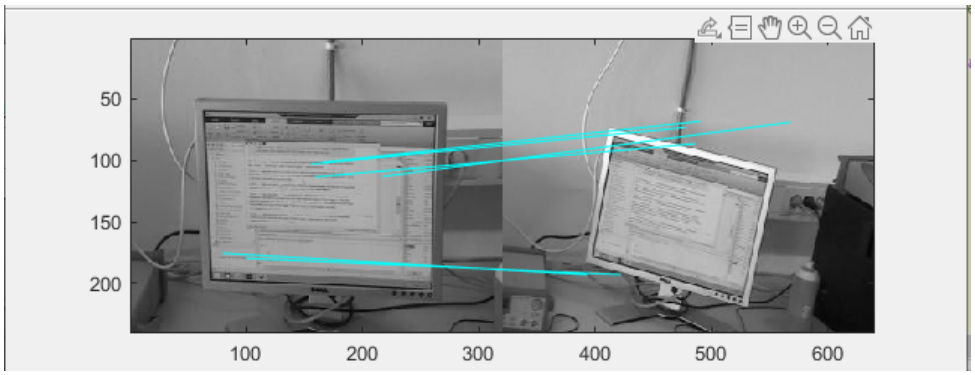


Figure imtool2 - Reference image registered with **second** computer screen RGB image

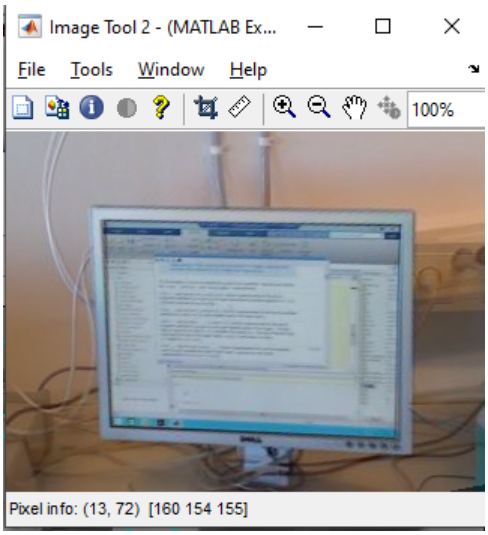


Figure13 - Lines joining the accepted matches between image and **third image**

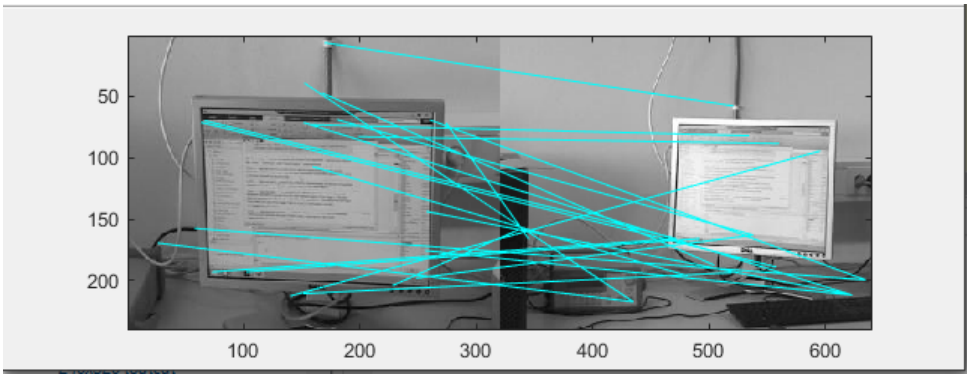


Figure14 - Lines joining best matches between image and **third image**

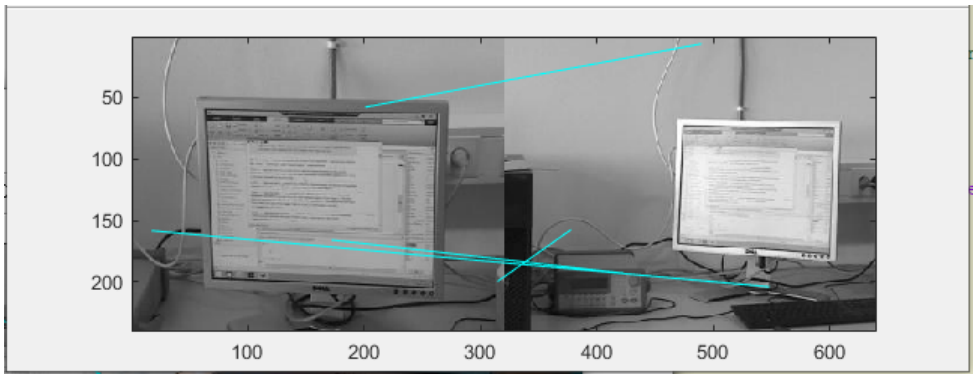


Figure imtool3 - Reference image registered with **third** computer screen RGB image

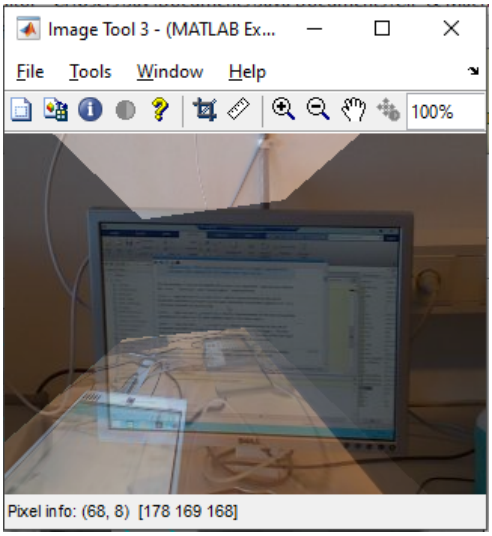


Figure15 - Lines joining the accepted matches between image and **fourth image**

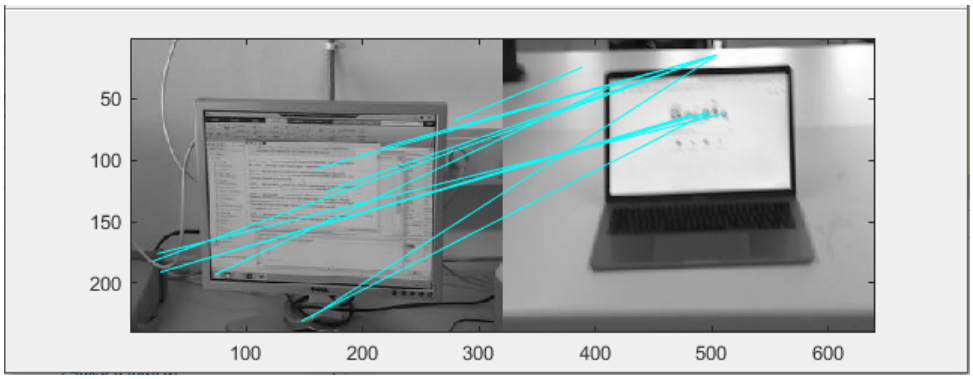


Figure16 - Lines joining best matches between image and **fourth image**

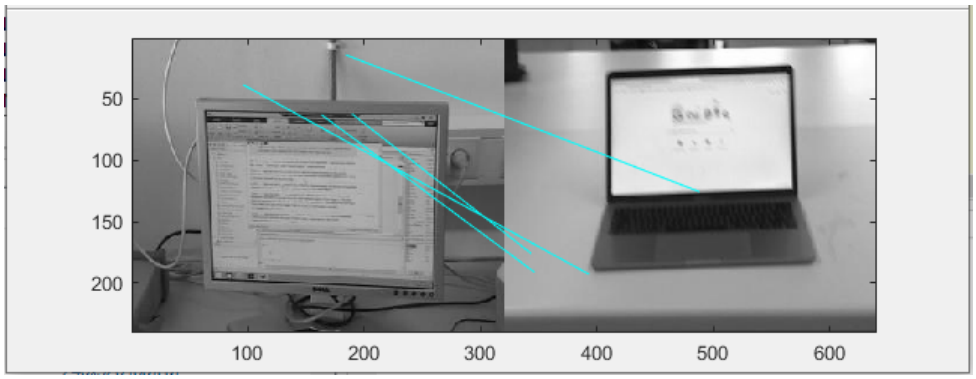
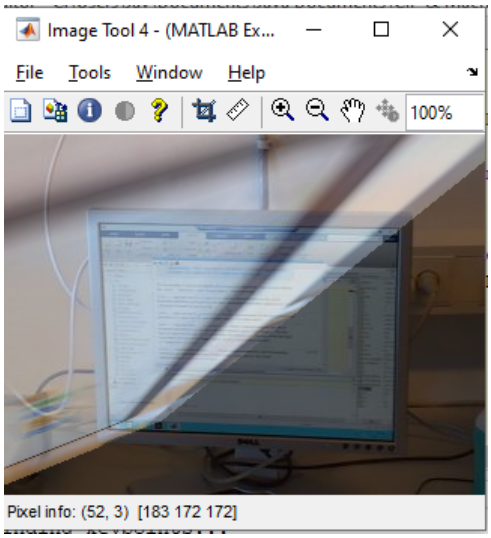


Figure imtool4 - Reference image registered with **fourth** computer screen RGB image



4 Discussion

1. Image numbers 10 -15 (computer screen FLIR0813 and power outlet images) in our IR image dataset cannot be properly segmented. See Figure 5 for their IR segmentation results. This is due to the fact that IR images are not from the same color palette of FLIR program. These images are from iron color palette where yellow is chosen to represent hottest temperature. See an image, the temperature scale has a shorter range plus there is a 'min' prefix before the temperature icon on top. Our segmentation algorithm works on the ranibow color palette where red is hottest temperature. One needs to use RGB images segmentation for these images.

2. An way to handle this shortcoming is to pre-process the IR images based on their temperature values. See Appendix 4 for a script segmenting IR images based on temperature values.

3. Image registration works well with the IR and RGB images of all classes. However, in three out of six cellphone images, the SIFT algorithm could not find any matches. Hence we changed the cellphone RGB images to IR images for registration with SIFT algorithm.

4. Cell phone RGB images can be pre-processed using their textures. See Appendix 5 for a entropy based pre-processed image which can be binarized and used for creating descriptors.

5. One could also try enhancing the contrast of cellphone RGB images with histogram techniques and re-checking the cellphone image registration with RGB images.

6. Image registration is used to overlay two or more images of the same scene so that the same scene objects from different images are aligned. In our example image, the computer screen FLIR0813 does not belong to the same scene. This results into improper results. See imtool4 figure. This is a constraint from our our image dataset and not from the SIFT algorithm.

5 Appendix

1. **Appendix 1 - final.m**

```
set(0, 'defaulttextinterpreter','Latex');
clc;
close all;

location_IR = ('C:\Users\Jay\Documents\Jaya
   Documents\CIP & Machine Vision\Final project\
   Training\IR');
ds_IR = imageDatastore(location_IR);

location_RGB = ('C:\Users\Jay\Documents\Jaya
   Documents\CIP & Machine Vision\Final project\
   Training\RGB');
ds_RGB = imageDatastore(location_RGB);
%imshow(preview(ds_RGB));

imgs_IR = readall(ds_IR);
imgs_RGB = readall(ds_RGB);

input = inputdlg('Choose image number between 1 to
    15');
im = cell2mat(input);
im = str2num(im);

imIR = double(imgs_IR{im})/255;
r_IR = IRpreprocessing(imIR);

imRGB = double(imgs_RGB{im})/255;
bfill = RGBcolorpreprocessing(imRGB);
bwedge = RGBedgesegmentation(imRGB);
```

```
bwop_IR = IRsegmentation(r_IR);
figure(5),imshow(bwop_IR),t = sgtitle('IR
    segmented image');t.Color = 'w';

biggest_IR = descriptors(bwop_IR);
figure(6), imshow(imIR),sgtitle('Recognized IR
    object');
hold on;
rectangle('Position',biggest_IR.BoundingBox,'
    EdgeColor','r','LineWidth',3);
hold off;

biggest_RGB = descriptors(bwedge);
figure(7), imshow(imRGB),sgtitle('Recognized RGB
    object');
hold on;
rectangle('Position',biggest_RGB.BoundingBox,'
    EdgeColor','r','LineWidth',3);
hold off;

%% Image registration
base = imgs_RGB{im};
compscreen = [1,2,3,10]; cs = length(compscreen);
cellphone = [4,5,6,7,8,9]; cl = length(cellphone);
powerout = [11,12,13,14,15]; po = length(powerout)
    ;

if im == 1|| im == 2|| im==3|| im==10  %Computer
    screen
     for cidx = 1:cs
         C = compscreen(cidx);
         float = imgs_RGB{C};
         imageregistration(base,float);
     end

elseif im == 4|| im==5|| im==6|| im==7|| im==8||
    im==9  %Cell phone
     base = imgs_IR{im};    %RGB cellphone images
         show no matches hence changing to IR images
         .
     for lidx = 1:cl
         L = cellphone(lidx);
         float = imgs_IR{L};
         imageregistration(base,float);
     end
```

```
elseif im == 11|| im==12|| im==13|| im==14|| im
    ==15  % Power outlet
     for pidx = 1:po
         P = powerout(pidx);
         float = imgs_RGB{P};
         imageregistration(base,float);
     end

else
     disp('Error')
end

%% IR color based image preprocessing
function r_IR = IRpreprocessing(imIR);
imrgb2gray = rgb2gray(imIR);
redIR = imIR(:,:,1);
greenIR = imIR(:,:,2);
blueIR = imIR(:,:,3);

figure(1),
subplot(2,2,1), imshow(redIR), title('IR Red
    channel');
subplot(2,2,2), imshow(greenIR), title('IR Green
    channel');
subplot(2,2,3), imshow(blueIR), title('IR Blue
    channel');
subplot(2,2,4), imshow(imIR), title('IR Original
    Image'),impixelinfo;
sgtitle('IR Original Image and its channels');

red_IR = graythresh(redIR);
green_IR = graythresh(greenIR);
blue_IR = graythresh(blueIR);

r_IR=im2bw(redIR,red_IR);
g_IR=im2bw(greenIR,green_IR);
b_IR=im2bw(blueIR,blue_IR);
IR_combined = and(and(r_IR,g_IR),b_IR);

figure(2),
subplot(2,2,1), imshow(r_IR), title('IR Red
    channel');
subplot(2,2,2), imshow(g_IR), title('IR Green
    channel');
```

```
subplot(2,2,3), imshow(b_IR), title('IR Blue
    channel');
subplot(2,2,4), imshow(IR_combined), title('IR
    Combined effect');
sgtitle('IR Binarized channels');
end

%% RGB Edge based segmentation
function bwedge = RGBedgesegmentation(imRGB)
imrgb2gray = rgb2gray(imRGB);
bwedge = edge(imrgb2gray,'canny');
figure(4),imshow(bwedge),t = sgtitle('RGB Edge
    based segmentation');t.Color = 'w';
end

%% IR segmentation
function bwop = IRsegmentation(image)
se = strel('rectangle',[10 10]);
bwop = imopen(image,se);
end

%% Common Descriptor function for IR and RGB
    images
function biggest_object = descriptors(bwop)
[labeled,numobj] = bwlabel(bwop,8);
regions=regionprops(labeled,'Area','BoundingBox');
allAreas = [regions.Area];
[sortedAreas, sortedIndexes] = sort(allAreas, '
    descend');
biggest_object = regions(sortedIndexes(1));
end

function imageregistration(base,float)
%%Feature detection
[baseImg, bdesc, bpos] = sift(base);
[floatImg, fdesc, fpos] = sift(float);

%% Feature matching
distRatio = 0.8;
match=zeros(1,size(bdesc,1));
for i = 1 : size(bdesc,1)
   dotprods = bdesc(i,:) * fdesc';
   [vals,indx] = sort(acos(dotprods));

   if (vals(1) < distRatio * vals(2))
      match(i) = indx(1);
```

```
    else
       match(i) = 0;
    end
end

temp = appendimages(baseImg,floatImg);

figure('Position', [100 100 size(temp,2) size(temp
    ,1)]);
colormap('gray');
imagesc(temp);
hold on;
cols1 = size(baseImg,2);
matchedDescs=zeros(numel(find(match)),4);

j=1;
for i = 1: size(bdesc,1)
  if (match(i) > 0)
    line([bpos(i,2) fpos(match(i),2)+cols1], ...
         [bpos(i,1) fpos(match(i),1)], 'Color', 'c
            ');
     matchedDescs(j,:)=[bpos(i,2) bpos(i,1) fpos(
        match(i),2) fpos(match(i),1)];
     j=j+1;
  end
end
hold off;
num = sum(match > 0);
fprintf('Found %d matches.\n', num);

%% Transformation model estimation
matchedPtsDistorted = [matchedDescs(:,3)
    matchedDescs(:,4)];
matchedPtsOriginal = [matchedDescs(:,1)
    matchedDescs(:,2)];
[tform,inlierpoints1,inlierpoints2,status] =
    estimateGeometricTransform(matchedPtsDistorted,
    matchedPtsOriginal,'projective');

bestMatches=cat(2,inlierpoints1,inlierpoints2);

temp = appendimages(baseImg,floatImg);

figure('Position', [100 100 size(temp,2) size(temp
    ,1)]);
colormap('gray');
```

```
imagesc(temp);
hold on;
cols1 = size(baseImg,2);
for i = 1: size(bestMatches,1)
    line([bestMatches(i,1) bestMatches(i,3)+cols1
       ], ...
     [bestMatches(i,2) bestMatches(i,4)], 'Color',
         'c');
end
hold off;
fprintf('Found %d best matches.\n', size(
   bestMatches,1));

%% image resampling and transformation
imref=imref2d([size(baseImg,1) size(baseImg,2)]);
warpedImg=imwarp(float,tform,'OutputView',imref);
imtool(imfuse(warpedImg,base,'blend'));
end
```

2. Appendix 2 - sift.m

```
% Credits: Thanks for initial version of this
   program to D. Alvaro and
%          J.J. Guerrero, Universidad de Zaragoza
   (modified by D. Lowe)

function [image, descriptors, locs] = sift(
   imageFile)

image=imageFile;
image = rgb2gray(image);
[rows, cols] = size(image);
% Convert into PGM imagefile, readable by "
   keypoints" executable
f = fopen('tmp.pgm', 'w');
if f == -1
    error('Could not create file tmp.pgm.');
end
fprintf(f, 'P5\n%d\n%d\n255\n', cols, rows);
fwrite(f, image', 'uint8');
fclose(f);

% Call keypoints executable
if isunix
    command = '!./sift ';
```

```
else
    command = '!siftWin32 ';
end
command = [command ' <tmp.pgm >tmp.key'];
eval(command);

% Open tmp.key and check its header
g = fopen('tmp.key', 'r');
if g == -1
    error('Could not open file tmp.key.');
end
[header, count] = fscanf(g, '%d %d', [1 2]);
if count ~= 2
    error('Invalid keypoint file beginning.');
end
num = header(1);
len = header(2);
if len ~= 128
    error('Keypoint descriptor length invalid (
       should be 128).');
end

% Creates the two output matrices (use known size
   for efficiency)
locs = double(zeros(num, 4));
descriptors = double(zeros(num, 128));

% Parse tmp.key
for i = 1:num
    [vector, count] = fscanf(g, '%f %f %f %f', [1
       4]); %row col scale ori
    if count ~= 4
        error('Invalid keypoint file format');
    end
    locs(i, :) = vector(1, :);

    [descrip, count] = fscanf(g, '%d', [1 len]);
    if (count ~= 128)
        error('Invalid keypoint file value.');
    end
    % Normalize each input vector to unit length
    descrip = descrip / sqrt(sum(descrip.^2));
    descriptors(i, :) = descrip(1, :);
end
fclose(g);
```

3. Appendix 3 - RGBcolorpreprocessing.m

```
%Trying Color based preprocessing of RGB image
function bfill = RGBcolorpreprocessing(imRGB)

imrgb2gray = rgb2gray(imRGB);
redRGB = imRGB(:,:,1);
greenRGB = imRGB(:,:,2);
blueRGB = imRGB(:,:,3);

% figure,
% subplot(2,3,1), imshow(redRGB), title('RGB Red
    channel');
% subplot(2,3,2), imshow(greenRGB), title('RGB
    Green channel');
% subplot(2,3,3), imshow(blueRGB), title('RGB Blue
     channel');
% subplot(2,3,4), imshow(imRGB), title('RGB
    Original Image');
% subplot(2,3,5), imshow(imrgb2gray), title('RGB
    Gray channel'),impixelinfo;
% sgtitle('RGB Original Image and its channels');

red_RGB = graythresh(redRGB);
green_RGB = graythresh(greenRGB);
blue_RGB = graythresh(blueRGB);
gray_RGB = graythresh(imrgb2gray);

r_RGB=im2bw(redRGB,red_RGB);
g_RGB=im2bw(greenRGB,green_RGB);
b_RGB=im2bw(blueRGB,blue_RGB);
RGB_combined = and(and(r_RGB,g_RGB),b_RGB);
gray = im2bw(imrgb2gray, gray_RGB);

bfill = imfill(RGB_combined,'holes');

figure(3),
subplot(2,3,1), imshow(r_RGB), title('RGB Red
    channel');
subplot(2,3,2), imshow(g_RGB), title('RGB Green
    channel');
subplot(2,3,3), imshow(b_RGB), title('RGB Blue
    channel');
subplot(2,3,4), imshow(RGB_combined),title('RGB
    Combined effect');
```

```
subplot(2,3,5), imshow(gray), title('RGB Gray
    image');
subplot(2,3,6), imshow(bfill), title('RGB combined
     with holes filled'),impixelinfo
sgtitle('RGB Binarized channels');
end
```

4. **Appendix 4 - temperature.m**

```
function temperature(IR);

filepath = ('C:\Users\Jay\Documents\Jaya Documents
    \CIP & Machine Vision\Final project\Training\IR
    \csv');
ds_temp = tabularTextDatastore(filepath,'
    FileExtensions','.csv');

T = readall(ds_temp);
A = table2array(T);
raw = (A -min(A(:)))/(max(A(:))-min(A(:)));

bins=hist(raw(:),100);
centre=find(bins==max(bins))/100;
low=centre-0.1;
if low<0
     low=0;
end
high=centre+0.1;
if high>1
     high=1;
end

raw=mat2gray(raw,[low high]);
figure(20),imshow(raw),sgtitle('Temperature based
    segmentation')
```

5. **Appendix 5 - entropy.m**

```
%% Try on cell phone RGB images. Entropy spread
    across image - Eim or Sim can be used for
    segmentation instead.
original = imread('FLIR0163- photo.jpg');
original = rgb2gray(original);
ent = entropyfilt(original);
Eim = rescale(ent);
```

```
S = stdfilt(original,ones(9));
Sim = rescale(S);
R = rangefilt(original,ones(9));
figure(4),subplot(2,2,1),imshow(original),title('
    original');
subplot(2,2,2),imshow(Eim),title('local entropy');
subplot(2,2,3),imshow(Sim),title('local standard
    deviation');
subplot(2,2,4),imshow(R),title('local range');
sgtitle('texture image vs original');
```