INM460 / IN3060 Computer Vision

In this lab, we will use Matlab to perform image matching using interest points detection and feature matching. Then, we will also perform object tracking through segmentation. Finally, we will implement a marker-controlled watershed segmentation pipeline.

Resources

The lab will make use of Matlab, and the following data files on the Lab materials 05 folder on Moodle in the Session 5 area:

- GregPen.avi

Task 1: Image matching

In today's lecture, we have seen that image matching (also called image alignment or image registration) can be performed with the following steps:

- Interest point detection
- Feature description
- Feature matching

In this tutorial, we will try to align two images using these steps.

Let's open the original image.

```
original = imread('cameraman.tif');
figure;
imshow(original);
```

We will create ourselves the image to be aligned with the original one (often called "moving image") by rotating and scaling the original one.

```
scale = 1.3;
J = imresize(original,scale);
theta = 31;
distorted = imrotate(J,theta);
figure
imshow(distorted)
```

Now let's perform interest point detection using SURF (we haven't covered this explicitly in the lecture, but we can consider it to be very similar to the one in SIFT). Matlab has a function that does this automatically called `detectSURFFeatures` . Apply it to extract interest points from the original image (call them "ptsOriginal") and from the distorted one (call them ptsDistorted"). Use the `insertMarker` function in Matlab to superimpose the points onto the respective images to check the results.

The next step is feature extraction. Let’s use the dedicated Matlab function `extractFeatures`. Given the fact that “ptsOriginal” and “ptsDistorted” were created using SURF, `extractFeatures` will extract SURF descriptors

```
[featuresOriginal,validPtsOriginal] = ...
            extractFeatures(original,ptsOriginal);
[featuresDistorted,validPtsDistorted] = ...
            extractFeatures(distorted,ptsDistorted);
```

It is possible that not all of the original points were used to extract descriptors. Points might have been rejected if they were too close to the image border. Therefore, the valid points are returned in addition to the feature descriptors.

The patch size used to compute the descriptors is determined during the feature extraction step. The patch size corresponds to the scale at which the feature is detected. Regardless of the patch size, the two feature vectors, `featuresOriginal` and `featuresDistorted`, are computed in such a way that they are of equal length. The descriptors enable you to compare detected features, regardless of their size and rotation.

To obtain candidate matches between the interest points we can compare the relative descriptors using the `matchFeatures` function. Candidate matches imply that the results can contain some invalid matches. Two patches that match can indicate like features but might not be a correct match. A table corner can look like a chair corner, but the two features are obviously not a match.

```
indexPairs = matchFeatures(featuresOriginal,featuresDistorted);
```

Each row of the returned `indexPairs` contains two indices of candidate point matches between the images. Use the indices to collect the actual point locations from both images.

```
matchedOriginal  = validPtsOriginal(indexPairs(:,1));
matchedDistorted = validPtsDistorted(indexPairs(:,2));
```

Let’s display the candidate matches with another built-in function:

```
figure
showMatchedFeatures(original,distorted,matchedOriginal,matchedDistor
ted)
title('Candidate matched points (including outliers)')
```

Inevitably, in general, there will be several mismatches, i.e. points that are wrongly matched and would push the alignment in the wrong direction. If there are enough good ones, the mismatches can be removed by using the RANSAC algorithm.

The `estimateGeometricTransform` function implements M-estimator sample consensus (MSAC), which is a variant of the RANSAC algorithm. MSAC finds a geometric transform and separates the inliers (correct matches) from the outliers (spurious matches).

```
[tform, inlierDistorted,inlierOriginal] = ...
            estimateGeometricTransform(matchedDistorted,...
                matchedOriginal,'similarity');
```

Let’s display the candidate matches after MSAC:

```
figure
showMatchedFeatures(original,distorted,inlierOriginal,inlierDistorte
d)
title('Matching points (inliers only)')
legend('ptsOriginal','ptsDistorted')
```

Finally, we can apply the computed geometric transformation to the distorted image, and see if it aligns to the original one:

```
outputView = imref2d(size(original));
recovered  = imwarp(distorted,tform,'OutputView',outputView);
figure
imshowpair(original,recovered,'montage')
```

Exercise 1: Try other, more dramatic distortions (get creative!). Can you break the image matching algorithm we just implemented?

Exercise 2: Extract the interest points using the Harris detector. How does it compare with SURF?

Task 2: Tracking through segmentation

In the previous lab tutorial, we learned a way to segment starfish out of a static image. However, segmentation can be used to track an object in a video sequence: if instead of being given a static image of starfish, we were given a video, we could apply the segmentation workflow from before to each frame the video, providing a pixel-wise tracking of objects through the video sequence.

Download the file “GregPen.avi” from this week’s Lab materials folder on Moodle. This is a movie Greg made on his webcam, holding (and moving) a pink highlighter in my hand. Matlab supports reading of videos through the VideoReader object. You can load the entire video into memory using

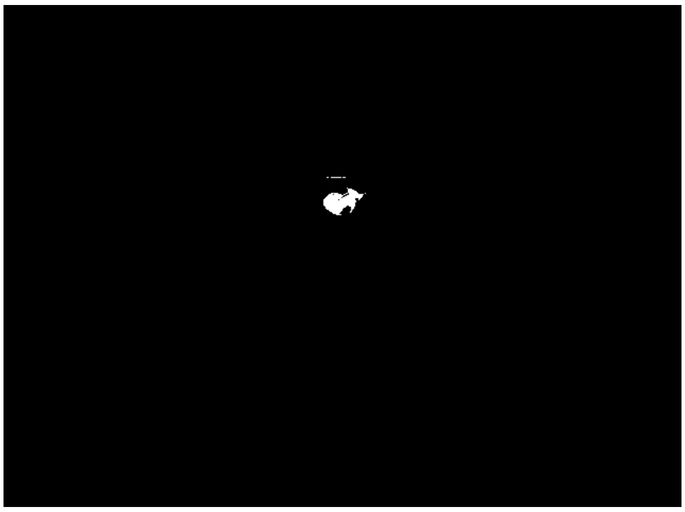
```
    videoReader = VideoReader('GregPen.avi');
    images = read(videoReader);
```

This will store the video frames in a 4D array called images. The ith frame can be read as

```
    I = images(:,:,:,i);
```

Load and display the first frame. Our goal will be to track the pink region through the video by segmenting it on each frame.

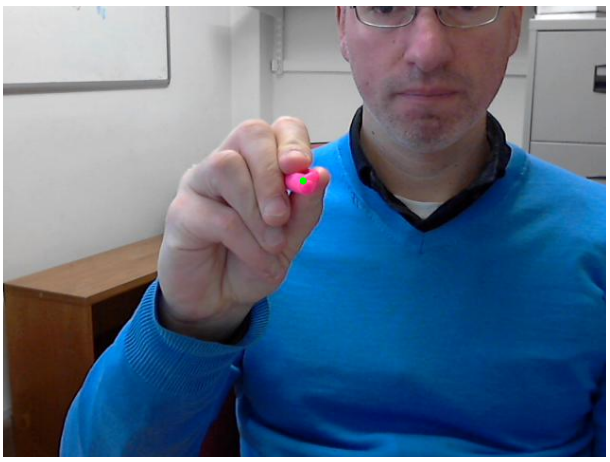
Using the Data Cursor, identify a colour range range [R_low, R_high], [G_low, G_high], [B_low, B_high] that characterises the colours observed for the pink highlighter. Then, perform a colour segmentation using this colour model. My segmentation mask looks like this:



This does a good job of identifying the pink pixels, but there are some false positives. We can retain the largest connect component using

```
K = bwareafilt(logical(J),1);
```

where J is the segmentation mask shown above. `bwareafilt` will filter regions based on area and only keep the largest one (since the second argument to this function is 1). Next, call `regionprops` to get the centroid of the region in K, call `hold on`, and display this centroid using the `scatter` function. My result looks like this, when overlaid on the original image:



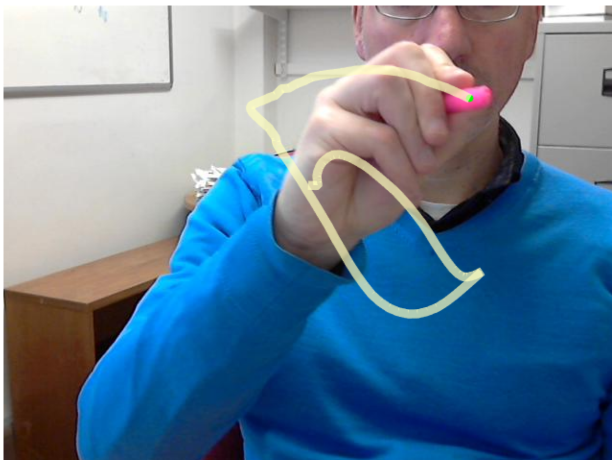
Note I have drawn the centroid as a green filled in circle by passing additional arguments to the `scatter` function.

Once you have a reasonable detection for the first frame, put your code in a loop and see if it tracks the pink region reliably through the video. Note: this video has 300 frames. If Matlab is not displaying your images in the loop, call `drawnow` at the bottom of your loop after you've drawn all your graphics. The code structure may look like this:

```
close all;
videoReader = VideoReader('gregPen.avi');
images = read(videoReader);
for i = 1:300
    I = images(:,:,:,i);
    hold off;
    imshow(I);

    % Perform segmentation here

    % Keep largest component

    % Compute centroid c

    hold on;
    scatter(c.Centroid(1), c.Centroid(2), 'g', 'filled');
    drawnow;

end
```

You can imagine how this example could be extended -- essentially we have made a human computer interface. Some examples:

- You could create a drawing program where the user selects colours and draws on the screen (basic idea shown below) using their webcam as input.
- You could create a computer game where the user's movement controls the gameplay.



Exercise 3: Test other segmentation algorithms from the ones presented in the previous lecture to see if you can replace the threshold-based one.

Task 3: Marker-controlled watershed segmentation pipeline

This example shows how to use watershed segmentation to separate touching objects in an image. The watershed transform is often applied to this problem. The watershed transform finds "catchment basins" and "watershed ridge lines" in an image by treating it as a surface where light pixels are high and dark pixels are low.

Segmentation using the watershed transform works better if you can identify, or "mark," foreground objects and background locations. Marker-controlled watershed segmentation follows this basic procedure:

1. Compute a segmentation function. This is an image whose dark regions are the objects you are trying to segment.

2. Compute foreground markers. These are connected blobs of pixels within each of the objects.

3. Compute background markers. These are pixels that are not part of any object.

4. Modify the segmentation function so that it only has minima at the foreground and background marker locations.

5. Compute the watershed transform of the modified segmentation function.

This example highlights many different Image Processing Toolbox™ functions, including `imgradient`, `watershed`, `label2rgb`, `labeloverlay`, `imopen`, `imclose`, `imreconstruct`, `imcomplement`, `imregionalmax`, `bwareaopen`, `graythresh`, and `imimposemin`.

Step 1: Read in the Color Image and Convert it to Grayscale

```
rgb = imread('pears.png');

I = rgb2gray(rgb);

imshow(I)
```

```
text(732,501,'Image courtesy of Corel(R)',...
     'FontSize',7,'HorizontalAlignment','right')
```

Step 2: Use the Gradient Magnitude as the Segmentation Function

Compute the gradient magnitude. The gradient is high at the borders of the objects and low (mostly) inside the objects.

```
gmag = imgradient(I);

imshow(gmag,[])

title('Gradient Magnitude')
```

Can you segment the image by using the watershed transform directly on the gradient magnitude?

```
L = watershed(gmag);

Lrgb = label2rgb(L);

imshow(Lrgb)

title('Watershed Transform of Gradient Magnitude')
```

No. Without additional preprocessing such as the marker computations below, using the watershed transform directly often results in "oversegmentation."

Step 3: Mark the Foreground Objects

A variety of procedures could be applied here to find the foreground markers, which must be connected blobs of pixels inside each of the foreground objects. In this example you'll use morphological techniques called "opening-by-reconstruction" and "closing-by-reconstruction" to "clean" up the image. These operations will create flat maxima inside each object that can be located using `imregionalmax`.

Opening is an erosion followed by a dilation, while opening-by-reconstruction is an erosion followed by a morphological reconstruction. Let's compare the two. First, compute the opening using `imopen`.

```
se = strel('disk',20);

Io = imopen(I,se);

imshow(Io)

title('Opening')
```

Next compute the opening-by-reconstruction using `imerode` and `imreconstruct`.

```
Ie = imerode(I,se);

Iobr = imreconstruct(Ie,I);

imshow(Iobr)

title('Opening-by-Reconstruction')
```

Following the opening with a closing can remove the dark spots and stem marks. Compare a regular morphological closing with a closing-by-reconstruction. First try `imclose`:

```
Ioc = imclose(Io,se);

imshow(Ioc)

title('Opening-Closing')
```

Now use `imdilate` followed by `imreconstruct`. Notice you must complement the image inputs and output of `imreconstruct`.

```
Iobrd = imdilate(Iobr,se);

Iobrcbr = imreconstruct(imcomplement(Iobrd),imcomplement(Iobr));

Iobrcbr = imcomplement(Iobrcbr);

imshow(Iobrcbr)

title('Opening-Closing by Reconstruction')
```

As you can see by comparing `Iobrcbr` with `Ioc`, reconstruction-based opening and closing are more effective than standard opening and closing at removing small blemishes without affecting the overall shapes of the objects. Calculate the regional maxima of `Iobrcbr` to obtain good foreground markers.

```
fgm = imregionalmax(Iobrcbr);

imshow(fgm)

title('Regional Maxima of Opening-Closing by Reconstruction')
```

To help interpret the result, superimpose the foreground marker image on the original image.

```
I2 = labeloverlay(I,fgm);

imshow(I2)

title('Regional Maxima Superimposed on Original Image')
```

Notice that some of the mostly-occluded and shadowed objects are not marked, which means that these objects will not be segmented properly in the end result. Also, the foreground markers in some objects go right up to the objects' edge. That means you should clean the edges of the marker blobs and then shrink them a bit. You can do this by a closing followed by an erosion.

```
se2 = strel(ones(5,5));

fgm2 = imclose(fgm,se2);

fgm3 = imerode(fgm2,se2);
```

This procedure tends to leave some stray isolated pixels that must be removed. You can do this using `bwareaopen`, which removes all blobs that have fewer than a certain number of pixels.

```
fgm4 = bwareaopen(fgm3,20);

I3 = labeloverlay(I,fgm4);

imshow(I3)

title('Modified Regional Maxima Superimposed on Original Image')
```

Step 4: Compute Background Markers

Now you need to mark the background. In the cleaned-up image, `Iobrcbr`, the dark pixels belong to the background, so you could start with a thresholding operation.

```
bw = imbinarize(Iobrcbr);

imshow(bw)

title('Thresholded Opening-Closing by Reconstruction')
```

The background pixels are in black, but ideally we don't want the background markers to be too close to the edges of the objects we are trying to segment. We'll "thin" the background by computing the "skeleton by influence zones", or SKIZ, of the foreground of bw. This can be done by computing the watershed transform of the distance transform of bw, and then looking for the watershed ridge lines (`DL == 0`) of the result.

```
D = bwdist(bw);

DL = watershed(D);

bgm = DL == 0;

imshow(bgm)

title('Watershed Ridge Lines)')
```

Step 5: Compute the Watershed Transform of the Segmentation Function

The function `imimposemin` can be used to modify an image so that it has regional minima only in certain desired locations. Here you can use `imimposemin` to modify the gradient magnitude image so that its only regional minima occur at foreground and background marker pixels.

```
gmag2 = imimposemin(gmag, bgm | fgm4);
```

Finally we are ready to compute the watershed-based segmentation.

```
L = watershed(gmag2);
```

Step 6: Visualize the Result

One visualization technique is to superimpose the foreground markers, background markers, and segmented object boundaries on the original image. You can use dilation as needed to make certain aspects, such as the object boundaries, more visible. Object boundaries are located where `L == 0`. The binary foreground and background markers are scaled to different integer values so that they are assigned different labels.

```
labels = imdilate(L==0,ones(3,3)) + 2*bgm + 3*fgm4;

I4 = labeloverlay(I,labels);

imshow(I4)

title('Markers and Object Boundaries Superimposed on Original
Image')
```

This visualization illustrates how the locations of the foreground and background markers affect the result. In a couple of locations, partially occluded darker objects were merged

with their brighter neighbor objects because the occluded objects did not have foreground markers.
Another useful visualization technique is to display the label matrix as a color image. Label matrices, such as those produced by `watershed` and `bwlabel`, can be converted to truecolor images for visualization purposes by using `label2rgb`.

```
Lrgb = label2rgb(L,'jet','w','shuffle');

imshow(Lrgb)

title('Colored Watershed Label Matrix')
```

You can use transparency to superimpose this pseudo-color label matrix on top of the original intensity image.

```
figure

imshow(I)

hold on

himage = imshow(Lrgb);

himage.AlphaData = 0.3;

title('Colored Labels Superimposed Transparently on Original
Image')
```

Copyright 1993-2018 The MathWorks, Inc.