Exercise 2: Keypoint detection and matching

Keypoints in an image are isolated points that are well-behaved in the sense that they are well locatable and identifiable. Because of these two properties a keypoint in an image can be associated with a keypoint in another image. This forms the basis for many applications, for instance:

- Tracking a specific point in a sequence of images. This is an important clue in image-based motion analysis, and visual navigation.
- Panoramic imaging, i.e., creating a panoramic image by stitching a series of overlapping images.
- Stereo vision. The location in the image of corresponding points in two stereo images provides information about the position of the imaged point in 3D.
- Object recognition.

Several keypoint detection algorithms exist. Each algorithm results in a specific type of keypoint. They are indicated by an acronym, such as SIFT which stands for *Scale invariant Feature Transform*. This is the first keypoint in literature that – as the name suggests – is invariant for a change of scale, and which was accompanied by a method for identification using so-called descriptors.

These descriptors enabled the association of keypoints from different images. The SIFT descriptor is a 128-dimensional vector. Corresponding keypoints from different images should have similar descriptors. A descriptor is a feature of the neighborhood of the keypoint. In the MATLAB documentation, descriptors are called features. This is quite confusing since, in MATLAB, keypoints are also called features.

The seminal paper of SIFT was first published in 1999. After that many other types of keypoints were proposed. In fact, keypoint detection is still an active branch of research in the computer vision community.

The act of associating keypoints is called matching. One of the issues here is that matching is not 100% reliable. It frequently happens that keypoint pairs are wrongly associated. Mismatched keypoint pairs are counteracted by so-called robust estimation.

Keypoints often come with two properties, also called attributes: a scale property and an orientation property. Suppose, for instance, that a keypoint detector would consider a disk-like shape in the image as a usable image structure for keypoint detection, then the center of the disk would be the location of the keypoint, whereas the radius of the disk would be a measure of scale. Since a disk is rotational symmetric, no sensible orientation property would be available. Suppose that this detector is also sensitive to 'half of a disk' shapes, then orientation could be unambiguously defined and the orientation property makes sense. In some cases, a shape is k-fold rotational symmetric. For instance, a cross shape could be 4-fold rotational symmetric. You can rotate it over 90^o without seeing the effect of this rotation. It is 4-fold symmetric since $360^0/4=90^o$. In 4-fold symmetry, the keypoint can be cloned 4 times, each clone having a unique orientation property.

Selection of a suitable type of keypoint detector, and a suitable identification method, for a given application, is often a trial-and-error process. In the current exercise you will get acquainted with different types of keypoint detectors, and we address the problem of robust association of the keypoints. The main question that is addressed in this exercise is as follows:

Given a sequence of images of the bending knee: (a) which type of keypoint, (b) which kind of matching method, (c) and which kind of robust estimation algorithm, are suitable for tracking the upper and lower leg in the two images?

The real challenge is to find a robust algorithm that find keypoints that belong to the upper leg, and matched to keypoints that belong to the lower leg.

Available data and software:

The data and software that will be used for the exercise are:

- Images that were acquired with a handheld camera. The images show the knee that is gradually bended. There are 7 images from a camera: **`ACLtestL01.png`** to **`ACLtestL07.png`**. These are images from the left camera of a stereo camera. Right images are also acquired, but will not be used in this exercise.
- To implement the robustness, the principle of RANSAC will be used. The general framework for RANSAC is given in a function **`ut_ransac`**. To provide you more knowledge about RANSAC, three different MATLAB demos are given that illustrate how **`ut_ransac`** can be used in three different applications.
- Two graphical functions are provided for visualization of intermediate results: **`ut_plotcov3`** and **`ut_plot_axis3D`**.

Assignments

1. Data preparation

Before diving into the topics of keypoint detection, the images need some color preprocessing. During image acquisition, a red pattern was projected to the scene to facilitate stereo matching. MATLAB's functions for keypoint detection use intensity images, i.e., grey scale images, and not RGB images. The function **`rgb2gray`** converts RGB images to intensity images. It does so by $I = 0.30R + 0.59G + 0.11B$. This is not necessarily the best conversion since red is dominate in the projected pattern. To get most of the detected keypoints being concentrated in the regions of the markers, the contrast in these regions must be optimized, and the red pattern must be suppressed.

A linear conversion, in general, is given by $I = aR + bG + cB + d$. To find a good conversion, the coefficients a, b, c, and d must be optimized. We will apply *principal component analysis* (PCA) and will follow the different steps for that. This has to be done only once as we assume that a good conversion for one image is also good for the other images. Since the goal of this exercise is to learn about keypoints, and not about image color processing, the MATLAB code for this part will be given.

1.1. Read the images **`ACLtestL01.png`** and **`ACLtestL02.png`**. Since we need image processing, the first step is to convert the images to type **`double`** using the function **`im2double`**. In this text, the resulting images will be referred by **`im1`** and **`im2`**, respectively.

1.2. To suppress the red pattern as much as possible, we need to characterize the RGB values of the pixels of the red pattern. Therefore, a ROI must be defined which solely consists of the red pattern on the leg, and which does not contain marker regions. Thus, we acquire a dataset of RGB pixels such that this dataset is representative for the red pattern:

You can copy-and-paste the code below to your m-file, but beware that some characters, e.g. the ' (single quote), have different ascii codes in the pdf than in MATLAB's editor. You have to correct that manually.

```
figure; imshow(im1);              % show im1 in a figure window
[imroi,rect] = imcrop;            % select a rectangular roi in the image
                                  % that only contains the red pattern on
                                  % the leg
[nr,nc,~] = size(imroi);          % get the size of the roi
imroi = reshape(imroi,nc*nr,3); % reshape to a (nc*nr) x 3 array
figure;                           % visualize as a cloud in the RGB space
plot3(imroi(:,1),imroi(:,2),imroi(:,3),'.','Markersize',1);
daspect([1 1 1]);                 % equalize the scales of the axes
xlabel('red');
ylabel('green');
zlabel('blue');
box on
```

1.3. The next step is to find a measure of the directivity of the scattering of the obtained cloud in the RGB space. This directivity is provided by estimating the covariance matrix of the cloud. For visualization, we also need the center of gravity, which is obtained by the mean of the dataset:

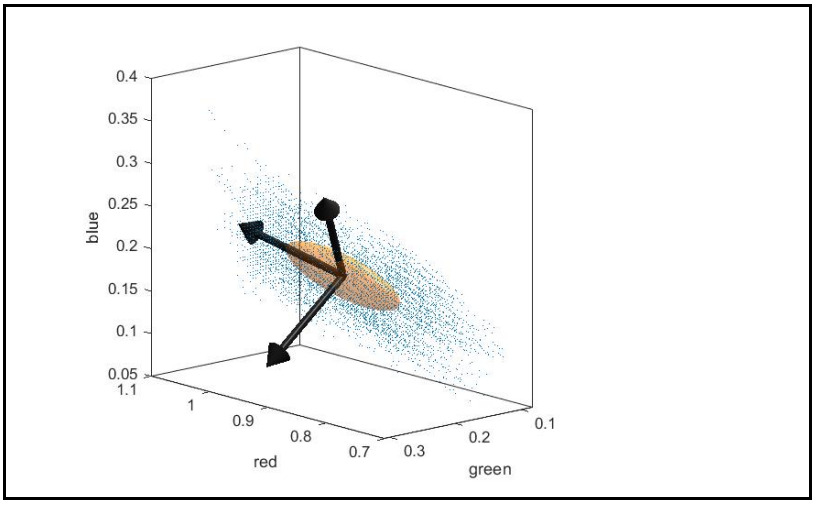
```
Covmat_roi = cov(imroi);          % estimate covariance matrix
mean_roi = mean(imroi);           % estimate the mean
surfp.specularstrength = 1;       % the strength of glossy reflection
surfp.diffusestrength = 0.5;      % the strength of matt reflection
surfp.ambientstrength = 0.7;      % the strength of ambient illumination
surfp.specularexponent = 25;      % the size of glossy spots
surfp.facealpha = 0.3;            % the transparency of a surface
hold on                           % plot ellipsoid representing scattering
ut_plotcov3([],50,50,mean_roi',Covmat_roi,[0.9 0.4 0],surfp);
camlight('left')                  % turn on some spot lights
camlight('right')
```

By manually changing the view point, you can get a visual impression of the shape of the ellipsoid that represents the covariance matrix. Try to find out in which direction the ellipsoid is most elongated, and in which direction it is most flat. These directions will be calculated in question 1.4.

1.4. The directivity of the scattering is quantified by the covariance matrix or ellipsoid. The shape of the ellipsoid is characterized by (a) its 3 principal axes, and (b) the 3 widths of the ellipsoid in the directions of these axes. These shape parameters are easily found: the axes are the eigenvectors **`e(i)`** of the matrix. The widths are given by the square root of their eigenvalues **`L(i)`**. Execute:

```
[E,L] = eig(Covmat_roi);          % get eigenvectors and eigenvalues
T = [E mean_roi'; 0 0 0 1];       % create 4x4 transformation matrix
h = ut_plot_axis3D([],T,'arrowlength',0.15);
view(-50,60)
```

Insert a copy of the figure:



1.5. The function **eig** calculates the eigenvectors and the eigenvalues, but it does not specify the order in which they are given. In order to rectify that, sort the eigenvectors and eigenvalues in ascending order of the eigenvalues:

```
%% sort in ascending order of the eigenvalues
[L,ind] = sort(diag(L));        % sort eigenvalues, small to large
E = E(:,ind);                   % sort corresponding eigenvalue
```

The eigenvectors are stored in the columns of the matrix **E**. I.e., **e(i)=E(:,i)**. Give the results:

L(1)	e(1)	L(2)	e(2)	L(3)	e(3)
3.925e-04	-0.0428 0.8172 -0.5748	0.0011	-0.3680 0.5220 0.7695	0.0064	0.9288 0.2445 0.2784

The largest eigenvalue is **L(3)** and the corresponding eigenvector is **e(3)**. Do you think that the values of **e(3)** make sense.? Explain:

The values of e(3) make sense, as it corresponding to the biggest eigenvalue, with the largest magnitude showing the large variance. Specifically, the value of e(3) in Ox direction (red color) is the biggest value showing the ellipsoid is most elongated in that direction

1.6. The goal is to suppress the red pattern as much as possible. Therefore, a projection of the red, green, and blue component on a direction with least red pattern variations is preferred. This is the direction of the eigenvector with smallest eigenvalue, i.e., **e(1)**. Execute the following code:

```
%% apply the conversion from RGB to intensities
[K,M,~] = size(im1);            % get size of images
im1g = reshape(im1,K*M,3);      % reshape image to vector style
im1g = im1g*E(:,1);             % suppress the red pattern
                                % by projecting on the direction with
                                % minimum variation in the red pattern
```

1.7. The resulting image should fit withing the range of an intensity image, which is **[0,1]**. This can be achieved by first normalize the image to a mean of zero, and a standard deviation of one. After that, the mean can be brought back to 0.5, and a suitable scaling can be applied such that the intensities nicely fit the range **[0,1]**. Apply the following code:

```
%% normalize the grey level range
gmean = mean(im1g);             % get the mean grey level
gstd = std(im1g);               % get the standard deviation
im1g = (im1g-gmean)/gstd;       % normalize mean and std dev
im1g = im1g/25 + 0.5;           % adapt the range so that it fits [0,1];
im1g = reshape(im1g,K,M);       % reshape back to image format
```

1.8. The processing of **im1g** in question 1.6 and 1.7 can be summarized by an image processing operation of the type $I = aR + bG + cB + d$. To prevent that the design of this procedure, as in question 1.2 to 1.5, must be repeated for each new image, the coefficients a , b , c , and d have to be extracted explicitly, so that they can be used for every new image:

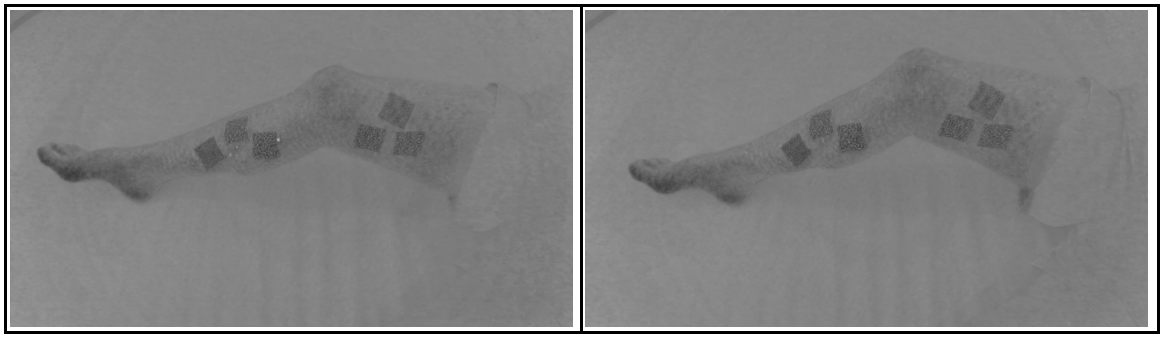
```
%% get conversion parameters for other images
A = E(:,1)/(25*gstd);           % the parameters a,b, and c
d = 0.5 - gmean/(25*gstd);      % the parameter d
```

$a =$ -0.0508 $b =$ 0.9715 $c =$ -0.6834 $d =$ 0.4226

1.9. Instead of applying the procedure in question 1.2 to 1.8, you can now apply the following code to any of the image you want to have the keypoints from:

```
%% transform the images
% a = ...;                          % fill in the numerical value of A(1)
% b = ...;                          % fill in the numerical value of A(2)
% c = ...;                          % fill in the numerical value of A(3)
% d = ...;                          % fill in the numerical value of d
im1g = a*im1(:,:,1) + b*im1(:,:,2) + c*im1(:,:,3) + d;
im2g = a*im2(:,:,1) + b*im2(:,:,2) + c*im2(:,:,3) + d;
```

im1g: **im2g**:



Note: use **imwrite**. Do not copy-and-paste a figure window here.

2. Keypoint detection and visualization

Keypoints can be roughly divided into corner-like features, and blob-like feature. In the first category, MATLAB's computer vision toolbox supports four different algorithms to find corners: HARRIS, MIN_EIGENVALUE, FAST, and ORB. It also supports four different algorithms to find blobs: BRISK, SURF, MSER, and KAZE. Because of a patent, SIFT keypoints are not in the toolbox. The eight available MATLAB functions with their properties[1] are:

MATLAB function	returned object	metric property	scale property	orientation property
detectHarrisFeatures	cornerPoints	x		
detectMinEigenFeatures	cornerPoints	x		
detectFASTFeatures	cornerPoints	x		
detectORBFeatures	ORBPoints	x	x	x
detectBRISKFeatures	BRISKPoints	x	x	x
detectSURFFeatures	SURFPoints	x	x	x
detectKAZEFeatures	KAZEPoints	x	x	x
detectMSERFeatures	MSERRegions		ellipse axes	x

INTERMEZZO

The computer vision toolbox uses classes. A class supersedes the possibilities of structs. It encapsulates a collection of related data, like a struct does. In a class, the fields are called properties. You can (often; not always) access these properties in the same way as you accessed the fields of a struct. In addition to this, a class also encapsulates functions, called methods or object functions, to manipulate the data.

The points returned by, for instance, detectFASTFeatures is not just a 2D array with coordinates. It is an instance of the class cornerPoints. Such an instance of a class is called an object. See the documentation of cornerPoints. For instance, it has a method 'plot'. If pts is an object of the class cornerPoints, then pts.plot or just plot(pts) will activate this method.

[1] The **metric property** is a measure of how reliable a keypoint is, i.e. its strength.

To see which of these 8 feature detectors are most suited for the current application, apply the 8 functions to the image **`im1g`**. Call the resulting objects **P11**, **P12**, **P13**, **P14**, **P15**, **P16**, **P17**, and **P18**. Next, visualize the found keypoints by showing the image in a figure window, and then overlaying this image with the keypoints. Note that all returned objects have built in functions, *methods*, for plotting. See their documentation, and see the Intermezzo. One of the object functions is **`selectStrongest`**. Use this function to limit the plot to at most, say, 500 keypoints. This is to prevent cluttering of the plot.

Most functions have additional input arguments, 'Name, Value' pairs, which influence the behavior of the function. Read the documentation, try to understand which effect they might have. Play around with these parameters, and select the one that you think are best. Motivate your choice.

When you have selected the parameters, store the figure windows to files and insert them on the next page.

3. Repeat question 2 with the next image in the sequence, **`im2g`**. Use the same parameters as selected in question 2. Call the resulting objects **P21**, **P22**, **P23**, **P24**, **P25**, **P26**, **P27**, and **P28**. You may want to visualize the results for yourself, but there is no need to report them in this document.

   Based on the results of question 2 and 3, select the keypoint type which you believe performs best for the current application. Hereafter, use this keypoint type. Apply the selected procedure to both images. We will refer to the selected objects by **P1** and **P2**

   Selected keypoint type, and the motivation for this:

```
I selected Min-Eigenvalue method as it well detected
points from all six markers on the legs. The number of
the detected points are sufficient and less outliers,
so I believe it is the suitable method
```

4. Keypoint matching and visualization

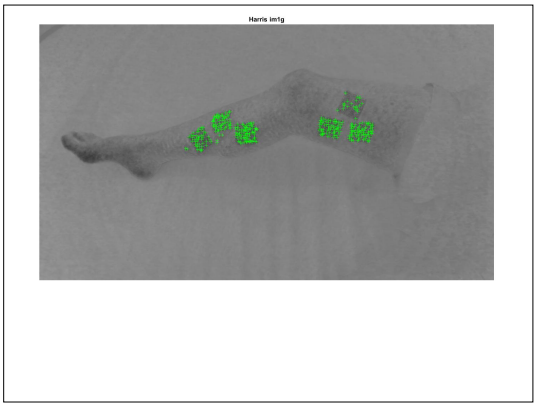
For identification of keypoints, the computer vision toolbox contains the function **`extractFeatures`**. The input of this function consists of an image, and an object with keypoints that were detected in the image. It returns keypoint descriptors. Not for all keypoints a descriptor can be calculated. Therefore, the output consists also of an object with valid keypoints, i.e., a subset of the input keypoints. **`extractFeatures`** implements 6 different types of descriptors: 'BRISK', 'FREAK', 'SURF', 'ORB', 'KAZE', and 'Block'.

Calculate the descriptors of **P1** and **P2** using the SURF descriptors, and return the resulting set of keypoints and their descriptors in **P11v**, **D11**, **P21v**, and **D21**.

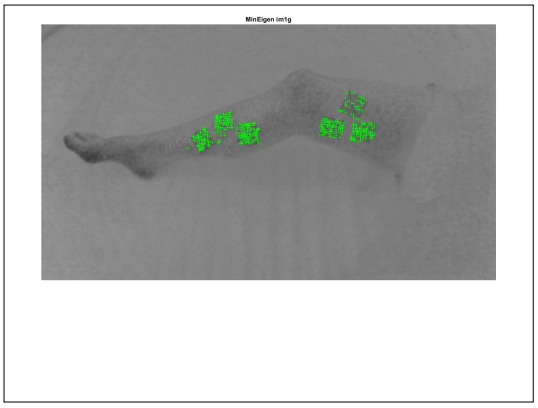
Matching is performed with MATLAB's function **`matchFeatures`**. This function takes as an input two descriptor arrays, e.g., **D11** and **D21**, and returns a table, e.g., **M1**, with two columns. Each row entry of the table represents a matching pair of keypoints. For instance, the row **`M1(4,:)`** represents a match between keypoint **`P11v(M1(4,1))`** and keypoint **`P21v(M1(4,2))`**.

Found keypoints in question 2:

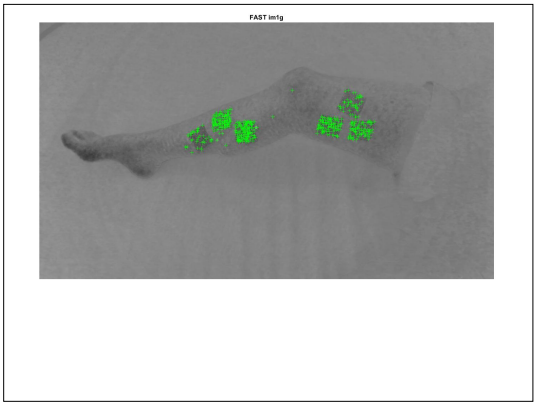
Harris:



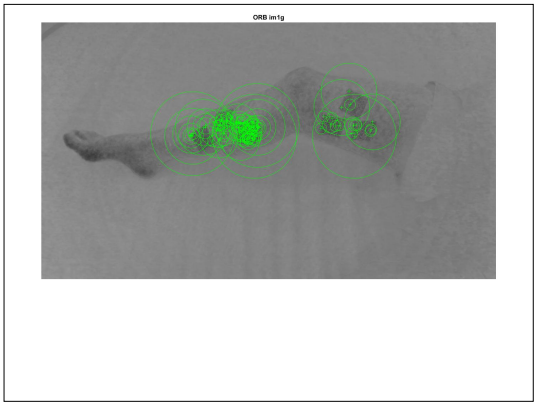
MinEigen:



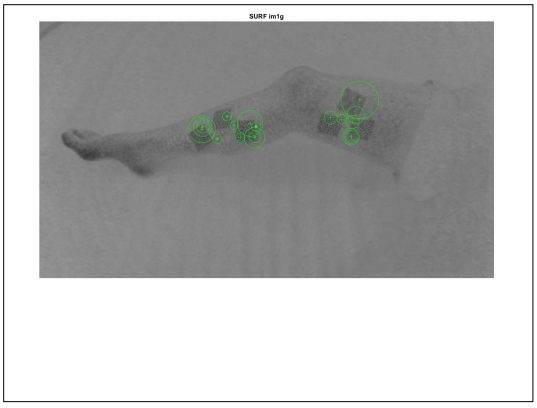
FAST:



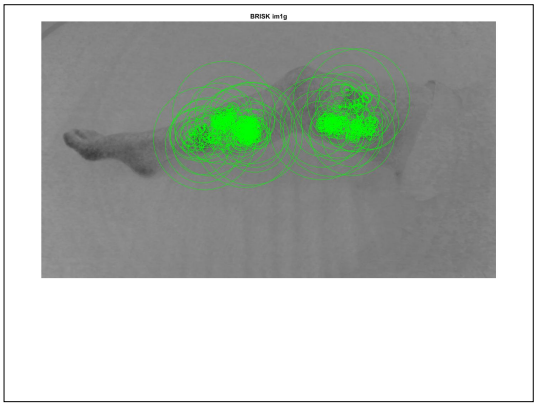
ORB:



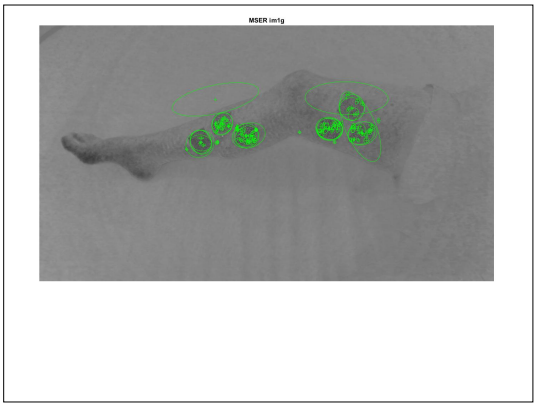
SURF:



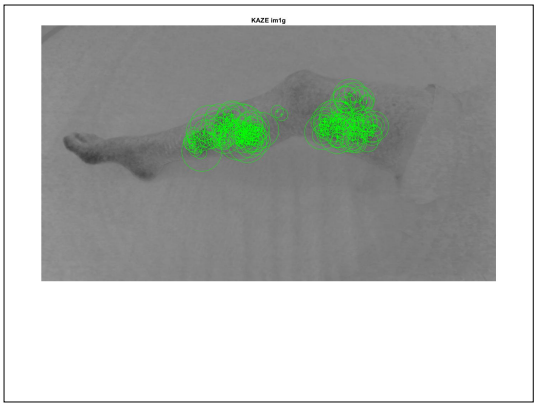
BRISK:



MSER:



KAZE:



Visualization of the matches is performed with the function **`showMatchedFeatures`** with the option **`'montage'`**.

Apply **`matchFeatures`** with the data **`D11`** and **`D12`**. Result in **`M1`**. Visualize with the **`'montage'`** option. **`matchFeatures`** has an option **`'MaxRatio'`**. Play around with this option to get results that you prefer. Repeat this procedure for the other 5 description methods[2]. Write the figure windows to file and insert them on the next pages.

Based on the results, shown in the figures above, select the descriptor type and matching procedure which you believe performs best for the current application. Hereafter, use these descriptors. Apply the selected procedure to both images. We will refer to the selected objects by **`P1v`** and **`P2v`**.

Selected descriptor type and motivation for this choice:

The descriptor type that I selected is Block techniques. This descriptor returns sufficient large number of matching points in both lower and upper legs. The matching is also quite correct between the two images

5. Inlier detection and cluster identification

We need robustness: no outliers. Apart from that, we need keypoints at two different locations: a cluster of keypoints at the markers on the lower leg, and a cluster of keypoints at the upper leg. Keypoints at other positions, for instance, at the foot, are not wanted. To achieve this, RANSAC will be used.

RANSAC is an algorithm that estimates the parameters describing a dataset. Point of departure is that we have a parametric model that characterizes the dataset. This dataset is supposed to have some outliers: data items that do not comply with the model. These outliers would compromise the estimation of the parameters. Therefore, they should be detected first.

As an input for RANSAC, we need:

- Data.
- A model that can explain the data, and which depends on parameters.
- Based on this model: an estimator that is able to estimate the parameters from given data.
- An error criterion that tells us how well (or how bad) a data item is explained by the model.
- A method for consistency checking. This check tells us whether we are content with a given consensus set, i.e., with a set of presumed inliers.

Data: depending on the chosen keypoint type, we have per matching keypoint pair the locations:

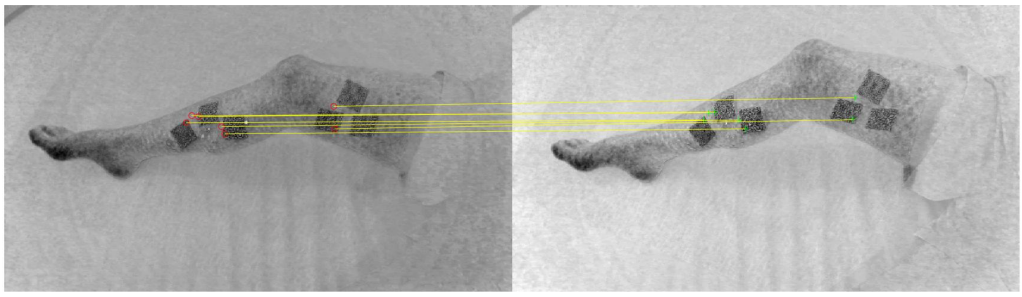
- $\mathbf{y} = (x_1, y_1)^{\mathrm{T}}$: location of a keypoint in image 1
- $\mathbf{z} = (x_2, y_2)^{\mathrm{T}}$: location of the matching keypoint in image 2

We therefore have a 4D vector $\mathbf{x}^{\mathrm{T}} = \begin{bmatrix} \mathbf{y}^{\mathrm{T}} & \mathbf{z}^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}}$ per pair. If we enumerate the matching pairs by m, the m-th pair gives a data item $\mathbf{x}_m$ We have M data items: $\mathbf{x}_1, \cdots, \mathbf{x}_M$ each being an 4D vector. The data can be represented in a $4 \times M$ dimensional array, which we represent here by

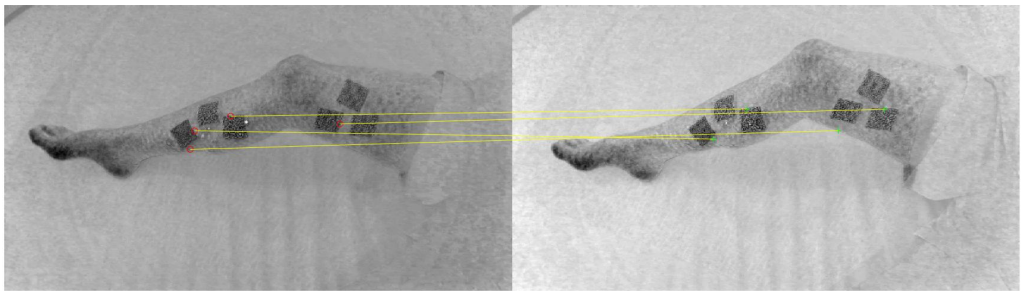
[2] Maybe not all descriptor methods can be combined with all keypoint types. If you meet this limitation, then choose another keypoint type.

Found matches in question 4:

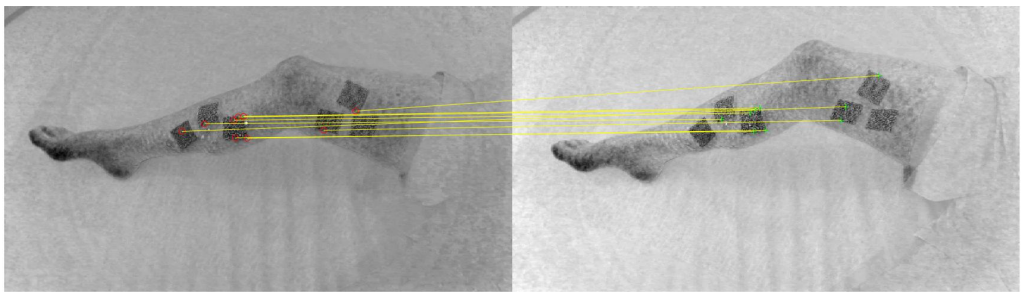
BRISK:



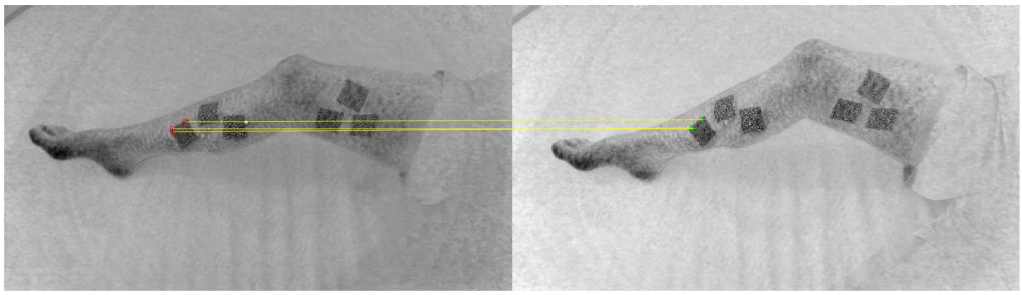
FREAK:



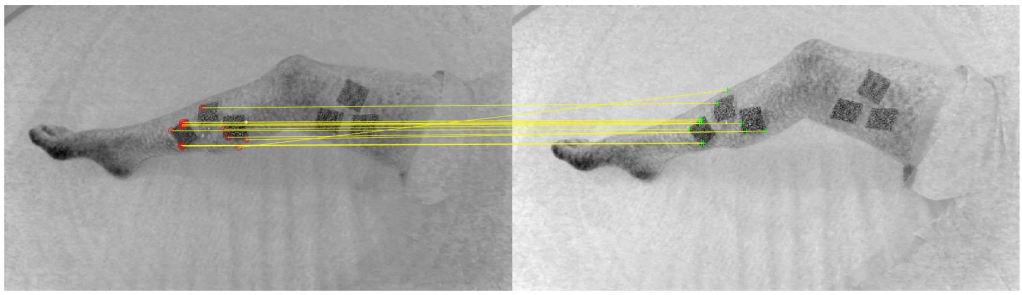
SURF:



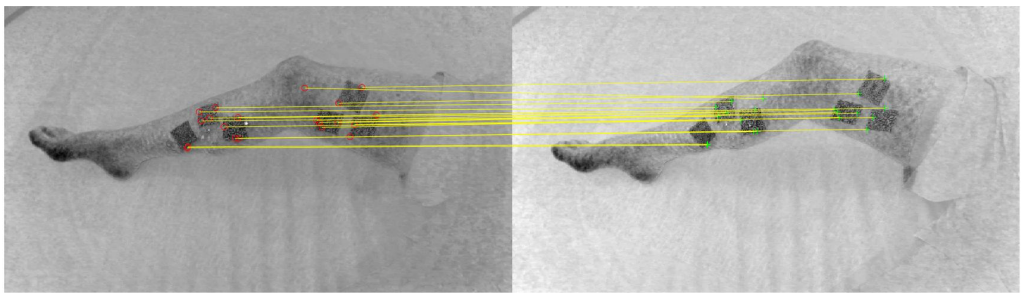
ORB:



KAZE:



Block:



the variable $\mathbf{X}$. We can also opt to use the scale and orientation attributes, s_1, s_2, φ_1, and φ_2, in which case the dimension of the data becomes $8 \times M$.

The model: the model is a mathematical characterization of the data. There are two clusters. The strategy is to apply RANSAC twice. One of the clusters is identified in the first call of RANSAC. Then after removing the items from this cluster from the dataset, the second cluster is identified by a second call.

To identify a cluster, one has to find the characteristics which are common in the data items in a cluster. One can think of the following:

A. The locations of the keypoints in a cluster all originate from a relatively small marker area. If the centroid of the cluster in the first image is denoted by $\mathbf{p}$, then the model describing this concentration of keypoints is:

$$\|\mathbf{y}_m - \mathbf{p}\| < R \quad \text{for all keypoints in the first image and all belonging to the same cluster} \tag{1}$$

Here, R is the radius of a circle in which all locations of in the cluster must lying. R is a design parameter, which must be selected beforehand by checking the size of the marker area. $\mathbf{p}$ is the position of the cluster. It is a parameter that RANSAC has to estimate.

B. Within a cluster, the displacements of the keypoints, from image 1 tot image 2, cannot change much if the time between acquisition of image 1 and image 2 is small:

$$\|\mathbf{z}_m - \mathbf{y}_m\| < D \tag{2}$$

D is tolerance. This is a design parameter, which must be selected beforehand. It must be selected such that the variations of displacements within one cluster is tolerated.

More involved displacement models can be considered. For instance, the displacements of the keypoints come from a rotating and translating 3D rigid body. Using stereo vision, as in exercise 2, the 3D positions of the keypoints are known. This can be exploited in a more accurate motion model of the keypoints. However, the techniques become soon quite complex.

The estimator: if the models that are expressed in eq (1) and (2) are adopted, the only parameter to estimate is $\mathbf{p}$, the center of the cluster. Given a subset of N samples $\mathbf{x}_n$ that all belong to the same cluster, an obvious estimator is the mean:

$$\hat{\mathbf{p}} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{y}_n \tag{3}$$

Error criterions: the error criterion is used to decide whether a data item is belonging to the cluster of which the center has been estimated, or not. One can use the model expressed in eq (1):

$$\varepsilon_m = \frac{1}{R}\|\mathbf{y}_m - \hat{\mathbf{p}}\| \tag{4}$$

The factor $1/R$ is added to normalize the criterion. The advantage is that when the criterion is to be compared with a threshold, this threshold becomes independent of the choice of R.

A combination of eq (1) and (2) is as follows:

$$\varepsilon_m = \frac{1}{R}\|\mathbf{y}_m - \hat{\mathbf{p}}\| + \frac{1}{D}\|\mathbf{z}_m - \mathbf{y}_m\| \tag{5}$$

Other combinations are also thinkable:

$$\varepsilon_m = \sqrt[q]{\left(\frac{1}{R}\left\|\mathbf{y}_m - \hat{\mathbf{p}}\right\|\right)^q + \left(\frac{1}{D}\left\|\mathbf{z}_m - \mathbf{y}_m\right\|\right)^q} \tag{6}$$

q is a real number. If $q = 1$, the expression reduces to eq (5). If $q = 2$, the Euclidian norm is applied. If $q \to \infty$, the criterion approaches:

$$\varepsilon_m = \max\left(\frac{1}{R}\left\|\mathbf{y}_m - \hat{\mathbf{p}}\right\|, \frac{1}{D}\left\|\mathbf{z}_m - \mathbf{y}_m\right\|\right) \tag{7}$$

One of these error criterions must be calculated for each data item. It is convenient to implement this in just one MATLAB function. The output of this function is the sequence ε_m which can conveniently be represented in an M -dimensional array $\boldsymbol{\varepsilon}$.

Detection of inliers: the error criterions $\boldsymbol{\varepsilon}$ are compared with a threshold. This threshold determines the error tolerance that we apply to decide whether a data item is an inlier or not. The set of presumed inliers is called the *consensus set*.

Consistency checking: we have to decide whether we are satisfied with this set, so that RANSAC can terminate. For this purpose, we have a consistency check function:

$$flag = C\left(\hat{\mathbf{p}}, \mathbf{X}_{con}, M\right) \tag{8}$$

where $\mathbf{X}_{con}$ represents the consensus set, i.e., all data items that are considered as inliers. M is the full number of data items. Several consistency checks can be envisaged. The simplest one is, that the number of inliers is large enough compared with M . The output of the function is a flag, which takes the value 1 for 'consistent', or 0 for 'not consistent'.

5.1. **Preparation.** Before continuing with the next questions, you are advised to carefully study one of the RANSAC examples that were shown in the lecture or mentioned in the syllabus:

- Download the file ransac_demo.zip. Extract the files.
- There are 3 demos. Choose one, e.g., **`demo_1_ransac_estimation_of_mean.m`**
- Set a breakpoint at the line that contains **`ut_ransac`**
- Run the code.
- When it stops, step in the function **`ut_ransac`**.
- Next single step through the function (use: step and not step in)
- Try to interpret what happens by inspecting the output variables of functions.
- Compare the encountered MATLAB functions with the one described in the text above. Can you relate the MATLAB functions with the mathematical functions in this text?

It does not make sense to continue if you don't understand the denotation of these functions.

5.2. The model. Describe in words shortly the model that you choose. You can use the model that is outlined above. If you have better ideas, you can use that:

We use the model described above for this exercise. First, we find center of randomly selected point. Utilizing the setup values of radius R, we can separate the two group of points (cluster 1/ 2 or lower/upper leg) base on the distance of them to this center. Within one cluster, the value of disparity D between 2 pictures helps us to check which points are most likely outliners. Apply the model of selection twice for the two groups.

5.3. **The estimator.** Create a MATLAB function, the header of which is:

```
function [parm_est, not_ok] = est_parm(Xsub)
```

Here, **Xsub** is a set of N data items. It is a subset of the full dataset. The first 2 rows of **Xsub** contain locations of all keypoints in image 1. The last 2 rows contain the locations from image 2. Thus, each column consists of the 2D vector $\mathbf{y}_m$, stacked with the 2D vector $\mathbf{z}_m$. Stacked together they form a 4D vector. The function returns an estimate of the parameter vector that you have defined in the model. The flag **not_ok** must be set to 0. Only if the estimator encountered a calamitous error, such that the parameter cannot be estimated, the flag must be set to 1.

MATLAB listing of **est_parm**:

```
function [parm_est, not_ok] = est_parm(Xsub)
% Input:
% Xsub 1st and 2nd row is x,y location of points in pic 1
% Xsub 3rd and 4th row is x,y location of points in pic 2
% Output:
% Location center of the cloud in pic 1 and pic 2 (4x1 - colume vector)
% not ok: checking if the calculation is possible
parm_est = mean(Xsub')';
not_ok = 0;    % flag to see whether the estimation was successful
end
```

5.4. **The error criterion evaluator.** Describe in a few words the error criterion that you are going to use.

I use the Euclidian norm for error criterion evaluator, or q=2 as indicating in equation (6). The error for each criteria (radius and disparity) will be normed sum with square root of 2. I prefer to use this method because Euclidian norm is widely used in finding error and is a popular method in engineering field. We use R=150 and D=100 in this part

Create a MATLAB function, the header of which is:

```
function e = err_eval(parm,X)
```

Here, **X** is a set of data items arranged in the column. **parm** is an estimated parameters vector. The function returns an array of error measures, one for each data item.

MATLAB **listing** of **err_eval**:

```
function e = err_eval(parm,X)
% Input:
% Parameters (4x1 column vector) and X(Nx4 column data set)
% Output:
% error criterion using Euclidian norm
global R D
ym=X(1:2,:);% Mx2 column vectors
zm=X(3:4,:);% Mx2 column vectors
e1=(1/R)*vecnorm(ym-parm(1:2)); % error 1 of all the point - row vector
e2=(1/D)*vecnorm(zm-ym); % error 2 of all the point - row vector
e=sqrt(e1.^2+e2.^2); % Euclidian norm for row vector
end
```

5.5. **The consistency check.** Describe in a few words the method for a consistency check. This check expresses whether a consensus set is considered to be acceptable, so that RANSAC can terminate.

The consistency is checked by considering if the amount of consensus set satisfying over the threshold number of point. The threshold is chosen to be over 10% of the original number of points.

Create a MATLAB function, the header of which is:

```
function ok = consistencycheck(parm,Xc,M)
```

parm is the estimated parameter vector, **Xc** is a consensus set, which is a subset of the full set of data items. **M** is the number of data items in the full set. The function returns a flag indicating whether the set is considered as consistent or not.

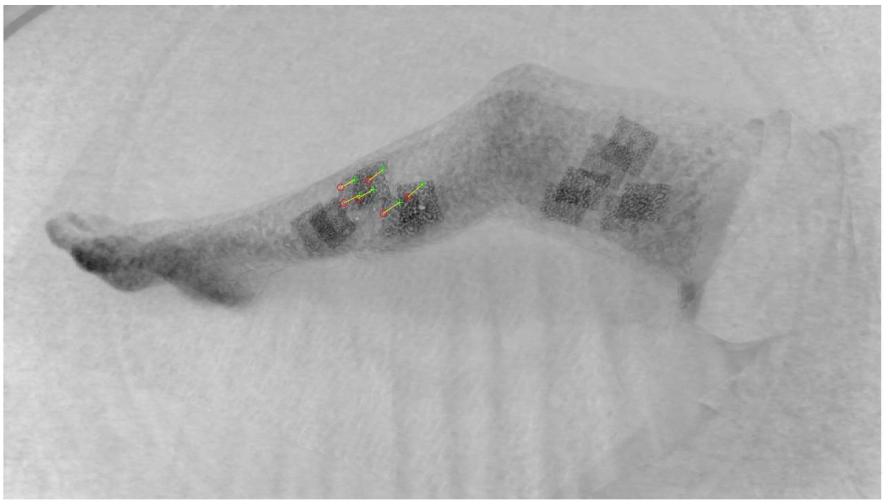
MATLAB listing of **consistencycheck**:

```
function ok = consistencycheck(parm,Xc,M,percentage)
global percentage
nmin = percentage*M;            % size should of consensus set should at least has
ok = (length(Xc)>=nmin);        %            a size larger than 50% of the data set
end
```

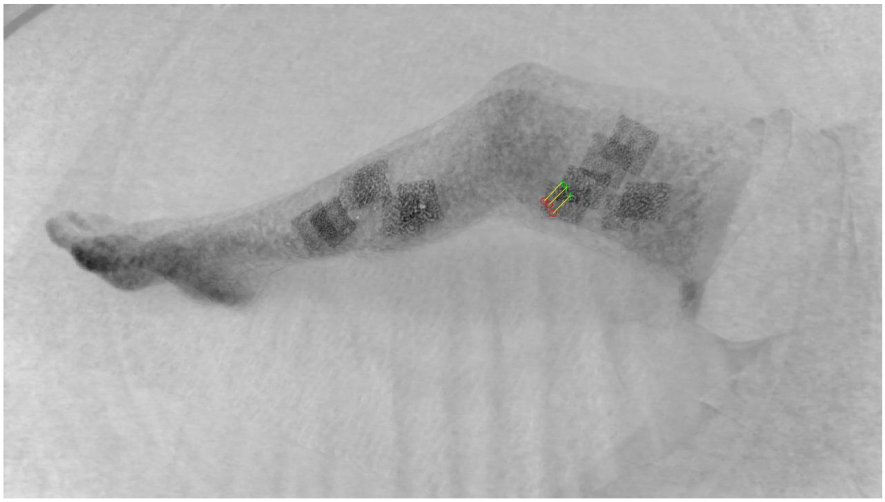
5.6. Application of RANSAC

Apply **ut_ransac** to the set of matching pairs found in question 4, and visualize the results by showing the two clusters of keypoints in two separate visualizations. Use **showMatchedFeatures** with the option **'blend'**.

Cluster 1:



Cluster 2:

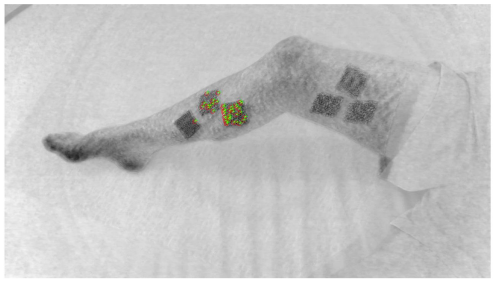


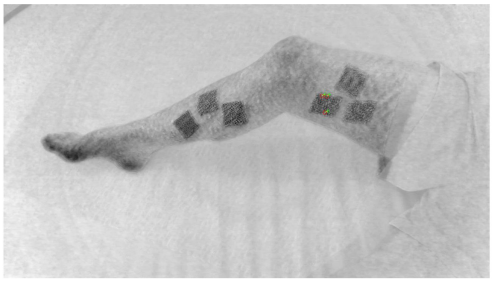
6. Assessment of the Results

6.1. To test the robustness of the method, developed in the previous questions, apply and visualize this method to the image pairs, mentioned below. Do this without altering any design parameter.

`ACLtestL02.png-ACLtestL03.png:`

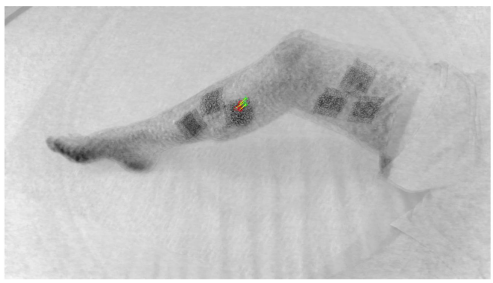
Cluster 1: Cluster 2:

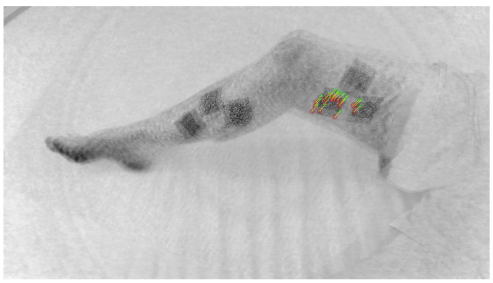




`ACLtestL03.png-ACLtestL04.png:`

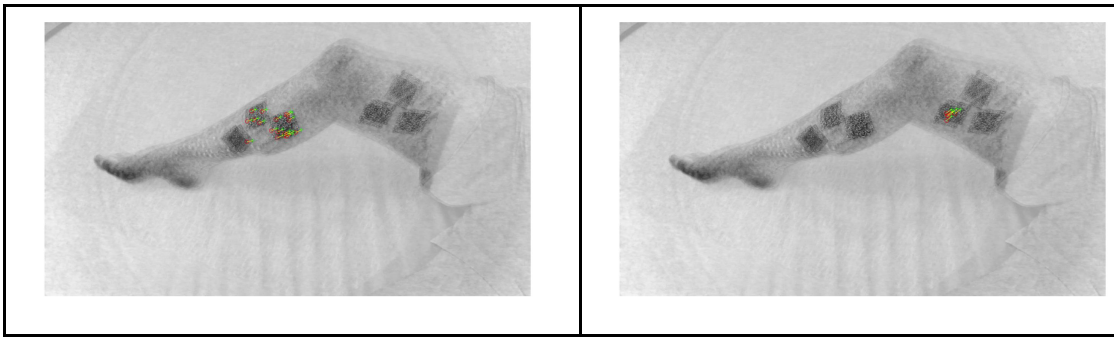
Cluster 1: Cluster 2:





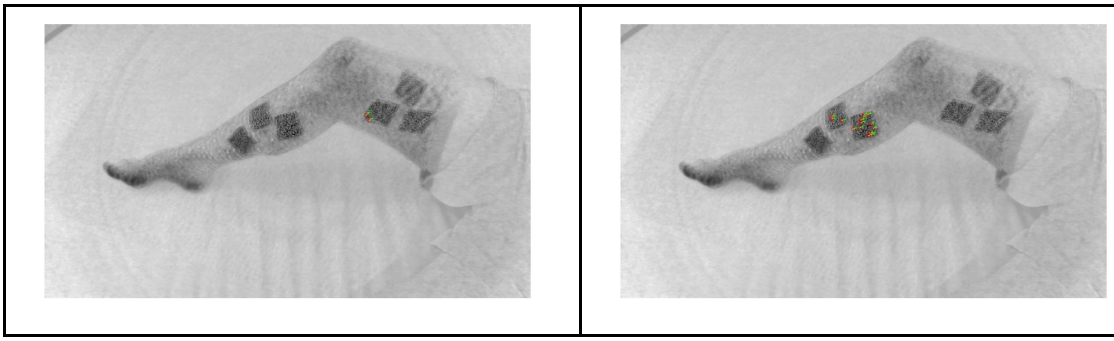
ACLtestL04.png-ACLtestL05.png:

Cluster 1: Cluster 2:



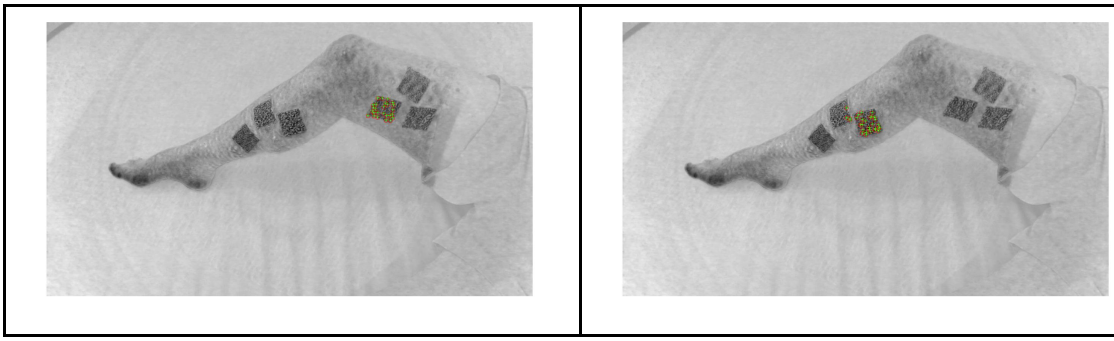
ACLtestL05.png-ACLtestL06.png:

Cluster 1: Cluster 2:



ACLtestL06.png-ACLtestL07.png:

Cluster 1: Cluster 2:



6.2. How do you assess the usefulness of the result in relation to the application that we have in mind? What works well? Are there any(possible) limitations?

Base on R and D, we are most certainly able to separate the clusters and the outliers in all the pictures. This shows the robustness of the method. However, as we can see that, with the fix values of R and D, we do not have the same quality of results in all the matching pairs. This is understandable as the leg movement velocity is not the same, make the disparity of points in each pair of picture are different. With slower movement, more points are satisfied with the error criteria as the disparity does not change that much. With faster velocity, the threshold for disparity or radius or tolerance might be needed to reconsider. Thus for the application of motion tracking, save-velocity movement might be more preferable if we are using this matching/detecting method.

7. Take home messages = preparation for the oral exam

Prepare yourself for the oral exam by making a list of new concepts that you have learned in this exercise. Make sure that you know these concepts, and that you understand them. Examples of questions that may be asked during the oral are:

		level
1.	What are the attributes of a keypoint? What is the descriptor?	knowledge
2.	Can you mention a few applications of keypoints?	knowledge
3.	Can explain how these keypoints are used in these applications?	understanding
4.	Can you mention the steps in the RANSAC algorithm?	knowledge
5.	Can you explain these steps?	understanding

Matlab code of your script (not the functions that are already listed above):

```
%%%%% 3DCV Exercise 3: Key points detection and matching %%%%%%%%%%%%%

%% Summary
% Keypoint: Isolated points that locatable and indentifable
% SIFT - Scale invariant feature transform => Identifying keypoints descriptors
% Keypoints has scale + orientation property: k-fold rotation symmetry
% Trial and error process for keypoints finding method
% Purpose of exercise : Given sequence of bending knee
% a/ What are keypoints b/ which matching method c/ robust estimation
% algorithm for tracking leg ?
% Idea: robust alg finding keypoints of upper leg => Match to key points
% belong to lower leg

% Available data
% - 7 Images hand held camera of bending leg
% - RANSAC for robustness
% - Visualization file ut_plotcov3 and ut_plot_axis3D

clear all; close all

%% Part 1: Data Preparation
% Color preprocessing - rgb to grey image with optimize conversion
% (suppress red color)
% I=aR+bG+cB+d with optimized a,b,c,d using PCA
% 1.1: Read the images
I1=imread('ACLtestL06.png');
I2=imread('ACLtestL07.png');
im1=im2double(I1); % change to double data
im2=im2double(I2);

% 1.2: To suppress red pattern, need to know the pixels RGB values of the
% red pattern. This part we choose ROI of this red pattern (no marker)
load 'imroi.mat'
%figure; imshow(im1); % show im1 in a figure window
%[imroi,rect] = imcrop; % select a rectangular roi in the image
                        % that only contains the red pattern on
                        % the leg
%[nr,nc,~] = size(imroi); % get the size of the roi
%imroi = reshape(imroi,nc*nr,3); % reshape to a (nc*nr) x 3 array
figure(1); % visualize as a cloud in the RGB space
plot3(imroi(:,1),imroi(:,2),imroi(:,3),'.','Markersize',1);
daspect([1 1 1]); % equalize the scales of the axes
xlabel('red');
ylabel('green');
zlabel('blue');
box on
%save imroi imroi

% 1.3: Measuring directivity (covariance) of obtained cloud in RGB space
% and center of gravity (mean) of data set
Covmat_roi = cov(imroi); % estimate covariance matrix
mean_roi = mean(imroi); % estimate the mean
surfp.specularstrength = 1; % the strength of glossy reflection
surfp.diffusestrength = 0.5; % the strength of matt reflection
surfp.ambientstrength = 0.7; % the strength of ambient illumination
surfp.specularexponent = 25; % the size of glossy spots
surfp.facealpha = 0.3; % the transparency of a surface
hold on % plot ellipsoid representing scattering
```