Notes regarding lab format

We will use Matlab Livescript for this lab. Livescript allows switching between text and code cells.

You will find the entire lab manual in this file. Some exercises require you to write a text answer, others require you to write code. You should not define functions inside this file. Instead save functions to the functions folder and call them from the code cells in this notebook.

Your finished lab report should be a .zip-file containing the data folder, your functions folder and this livescript file. As usual, you should also provide a pdf of the result of running the live script (in the Live Editor, you can **export to pdf** under Save) where all result images should be visible.

In certain sections of this lab, MATLAB might throw warnings about conditioning of matrices. You should turn them off using the command **`warning('off','all')`** and submit the final pdf withput these warnings displayed.

Since we need to access the functions and data folder the first step is to add these two locations MATLAB's path.

```
addpath('./functions');
addpath('./data');
warning('off','all');
```

Important rules: You are only allowed to ask TAs and lectures for help. It is not allowed to copy solutions, not even partially, from other sources (your fellow students other than your group-mate, internet, Large Language Models or whatever).

Lab 3 :Image Registration

An affine transformation is written as:

$$\begin{pmatrix} \tilde{x} \\ \tilde{y} \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} = A \begin{pmatrix} x \\ y \end{pmatrix} + t$$

Apart from rotation, translation and scaling, it also allows stretching the image in an arbitrary dimension.

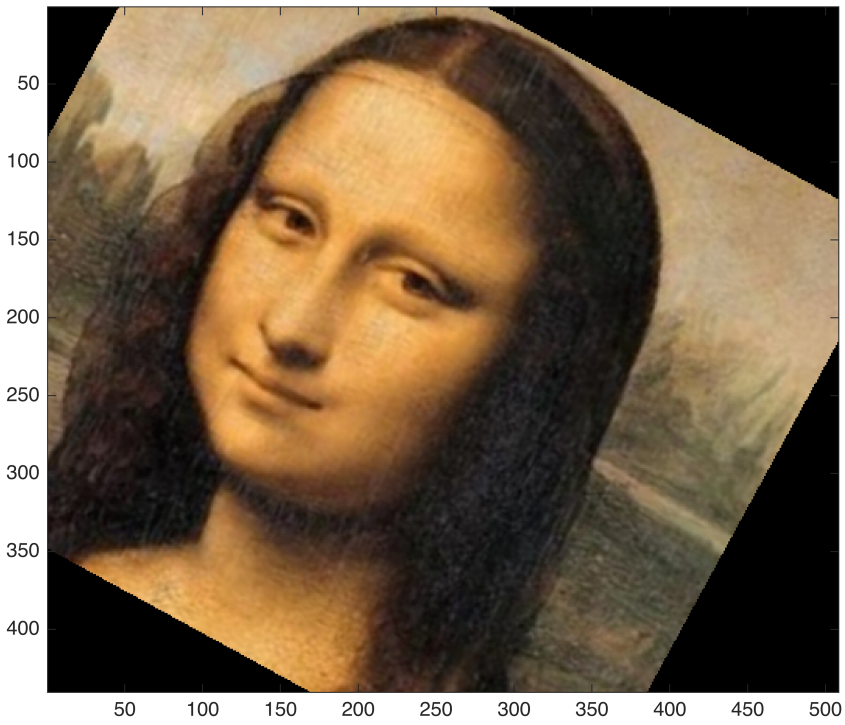
Ex 3.1 Getting started

Once you have found a proper coordinate transformation between two images, you can use the provided function affine_warp to warp the source image and create a warped image. Let's try it.

Load the image mona.png to the variable img. After copying your `read_image` function from previous labs into the *functions* directory, try running the following code snippet. Play around with the values in A, t to see the results and understand the effect of how to rotate, translate and stretch.

```
img = imread('data/mona.png');
A = [0.88 -0.48; 0.48 0.88];
```

```
t = [100;-100];
target_size = size(img);
warped = affine_warp(target_size, img, A, t); imagesc(warped);
axis image;
```



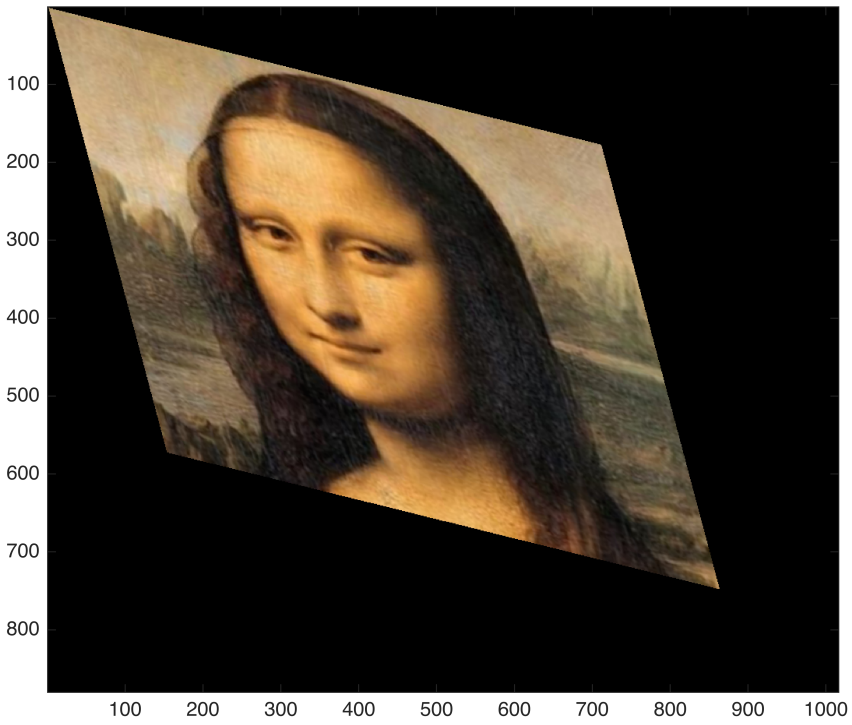
If one would like to stretch the image along the x-axis, one simply applies `A=diag([k,1])` for some stretching factor k. How should one obtain a stretching with a factor k along a diagonal direction with an affine transformation?

HINT : Think about decomposing the A matrix as a rotation and scaling.

Write your answer with an explanation of the transformation used as a comment here.

```
%%% Your Code here
% Assume stretching along the diagonal direction with the stretching factor
k
k = 1.7;

% Construct the rotation matrix R
height = 508;
width = 440;
theta = atan(width / height); % Calculate the Angle of the diagonal
R = [cos(theta) -sin(theta); sin(theta) cos(theta)];

% Construct the diagonal scaling matrix S
S = diag([k,1]);
% Combining rotation and scaling matrices
```

```
A = R * S * R';
t = [0; 0];

img = imread('data/mona.png');
target_size = size(img);
warped = affine_warp(target_size.*2, img, A, t); imagesc(warped);
axis image;
```




```
%%% Your explanation here
% Calculate the angle theta of the diagonal based on the image's height and
width, then construct the rotation matrix R to rotate along the diagonal
direction.
% Construct the diagonal scaling matrix S where the first diagonal element
is k, controlling the degree of stretching along the diagonal direction.
% Multiply the rotation matrix R with the scaling matrix S to obtain the
affine transformation matrix A that simultaneously rotates and stretches
along the diagonal.
% Finally, use the affine_warp function to apply the affine transformation
matrix A to the original image img, performing transformations.
```

Ex 3.2 Minimum Correspondences

What is the minimal number of point correspondences, K, required in order to estimate an affine transformation between two images?

Write your answer with explanation as a comment here.

```
%%
% Since an affine transformation has 6 degrees of freedom
% and therefore can be represendted by a matrix of 6 parameters.
% Hence, the minimal number of correspondence needed to estimate
% the affine trasformation between two images is 3. So K=3.
% With 3 points, one has 3 equations that relate the 3 pairs
% of corresponding points in the two images. These equations can
% be used to solve for the 6 parameters of the transformation matrix.
% It is worth noting that 3 correspondences are sufficient to
% estimate the transformation matrix but the solution might not
% be unique. It is recommended to use more than 3 points.
```

In general, an estimation problem where the minimal of amount of data is used to estimate the unknown parameters is called **a minimal problem**.

Ex 3.3 Writing test case

For any estimation task it is a good idea to have at least one test case where you know what the answer should be. In this exercise, you should make such a test case for RANSAC. Start by generating random points, store them in a variable named **`pts`**, and a random transformation - **`[A_true, t_true]`**. Then transform **`pts`** to create a variable **`pts_tilde`**. If you want to make it more realistic, add small random noise to **`pts_tilde`**. You now have two sets of points related by a known affine transformation as in *Ex 3.2*. In the following exercises you will try to estimate this transformation. As you know the correct answer it is easy to detect if you make a mistake.

Make a function

```
function [pts, pts_tilde, A_true, t_true] = affine_test_case()
```

that generates a test case for estimating an affine transformation. The transformation should transform **`pts`** to **`pts_tilde.`** Don't add any gross outliers. Outputs **`pts`** and **`pts_tilde`** should be `2 x N-arrays`. Also output the *true* transformation, so you know what to expect from your code.

After you have written the function, test it with a few runs. **Your code here:**

```
%%% Your code here
[pts, pts_tilde, A_true, t_true] = affine_test_case();
size(pts)
```

```
ans = 1×2
     2    63
```

```
size(pts_tilde)
```

```
ans = 1×2
     2    63
```

Ex 3.4 Writing a minimal solver

Make a minimal solver for the case of affine transformation estimation. In other words, **make a function**

```
[A, t] = estimate_affine(pts, pts_tilde)
```

that estimates an affine trasnformation mapping **`pts`** to **`pts_tilde`**, where **`pts`** and **`pts_tilde`** are `2 x K` -`arrays` and K is the number you found in *Ex 3.2*. Try your function on points from the test case in *Ex 3.4*. **Your code here:**

```
%%% Your code here
pts_subset = pts(:, 1:3);
pts_tilde_subset = pts_tilde(:, 1:3);
[A, t] = estimate_affine(pts_subset, pts_tilde_subset)
```

```
A = 2×2
   -0.3922   -0.2617
    1.2409   -0.8871
t = 2×1
    3.6537
 -200.5373
```

```
A_true
```

```
A_true = 2×2
   -0.3922   -0.2617
    1.2409   -0.8871
```

```
t_true
```

```
t_true = 2×1
    3.6537
 -200.5373
```

Ex 3.5 Computing residuals

Make a function

```
function res = residual_lgths(A, t, pts, pts_tilde)
```

that computes the lengths of 2D residual vectors. The function should return an array with N values. *Hint :* Given a `2 x N` matrix, stored in `M`, the column-wise sum of the **squared** elements can be computed as `sum(M.^2, 1)`.

Verify that for no outliers and no noise, you get zero (to machine precision) residual lengths given the true transformation:

```
%%% Your code here
res = residual_lgths(A, t, pts, pts_tilde)
```

```
res = 1×63
10^-26 ×
         0    0.0808         0    0.0808         0    0.3231         0         0 ···
```

Ex 3.6 Test case with outliers

Modify your function **affine_test_case** to create a new function **affine_test_case_outlier** that takes a parameter **outlier_rate** and produces a fraction of outliers among the output points. For example, if **outlier_rate** is 0.2, then 80% of the samples will be related by **pts_tilde = A_true * pts + t_true**. While for the remaining 20%, pts_tilde can be distributed uniformly within some range.

```
[pts, pts_tilde, A_true, t_true] = affine_test_case_outlier(outlier_rate)
```

Test your code here:

```
%%% Your code here
outlier_rate = 0.2;
[pts, pts_tilde, A_true, t_true] = affine_test_case_outlier(outlier_rate);
size(pts)
```

```
ans = 1×2
     2    47
```

```
size(pts_tilde)
```

```
ans = 1×2
     2    47
```

Ex 3.7 Writing a RANSAC based solver

Make a function

```
[A,t] = ransac_fit_affine(pts, pts_tilde, threshold)
```

that uses RANSAC to find an affine transformation between two sets of points. (Like before the transformation should map **pts** to **pts_tilde**.) Test your function on test cases generated with your function **affine_test_case_outlier**.

RANSAC Note : You can use the notes to decide on the number of RANSAC iterations needed given your required confidence and estimated inlier ratio. You can also use fixed iterations but make sure that you have enough iterations to find a reasonable solution.

Verify your code here. Try different outlier rates. Make sure that you get the right transformation for a reasonable outlier fraction (more than 0.5) as well.

```
%% Your code here
outlier_rate = 0.5;
[pts, pts_tilde, A_true, t_true] = affine_test_case_outlier(outlier_rate);
threshold = 2;
A_true
```

```
A_true = 2×2
   -0.0682    0.2940
    0.3129   -0.6031
```

```
t_true
```

```
t_true = 2×1
  -16.4707
 -121.9758
```

```
[A, t] = ransac_fit_affine(pts, pts_tilde, threshold)
```

```
====================================
RANSAC iterations done!
It took k=10 iterations to terminate...
Residual: 17958.94
Number of inliers: 122
====================================
A = 2×2
   -0.0682    0.2940
    0.3129   -0.6031
t = 2×1
  -16.4707
 -121.9758
```

Ex 3.8 Aligning Images

For this exercise, you should use the function

```
pts = detectSIFTFeature(img); [descs, valid_pts] = extractFeatures(img, pts, 'Method' ,
'SIFT');
```

to extract SIFT features. Note that it only works for grayscale images, so if you have a colour image you need to convert it to grayscale before using these functions. To match features you can use the built-in function matchFeatures. To use the Lowe criterion (with threshold 0.8) you should use the following options:

```
corrs = matchFeatures(descs1', descs2', 'MaxRatio', 0.8, 'MatchThreshold', 100);
```

Write a function

```
warped = align_images(source, target, thresh)
```

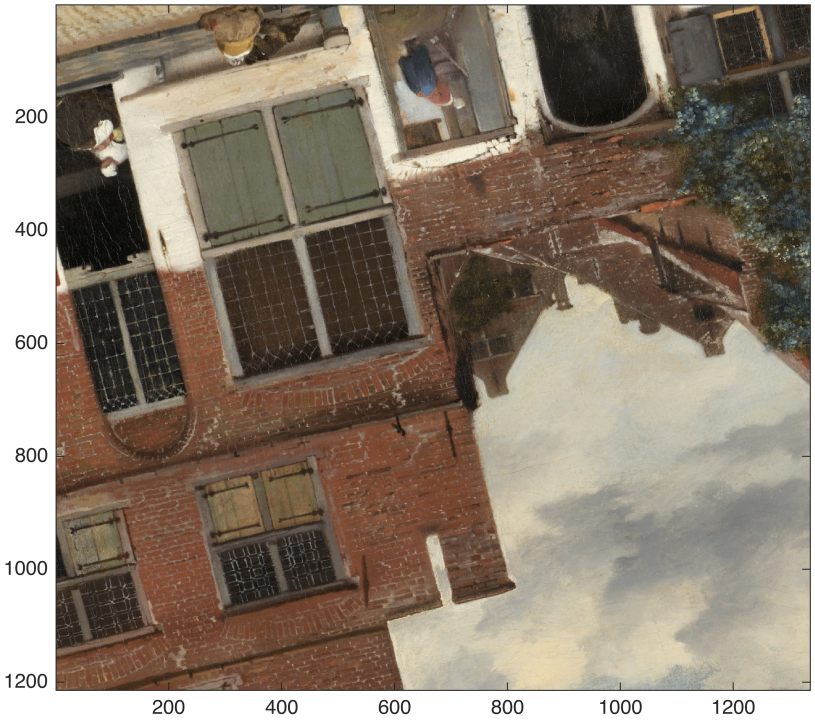
that uses SIFT and RANSAC to align the `source` image to the `target` image with `thresh` as the residual threshold for counting inliers in your RANSAC. To perform the actual warping, use the **`affine_warp`** function within your **`align_images`** function. Be very careful about the order in which you send the points to RANSAC.

Align `vermeer_source.png` to `vermeer_target.png` using your **`align_images`** function. Plot the source, target and warped source images together.

```
%% Your code here
img_src = imread('vermeer_source.png');
source = rgb2gray(img_src);

img_tgt = imread("vermeer_target.png");
target = rgb2gray(img_tgt);

warped = align_images(source, target, 30);
```

```
====================================
RANSAC iterations done!
It took k=142 iterations to terminate...
Residual: 1463611.88
Number of inliers: 366
====================================
```

```
%% You can use the following code snippet to plot images next to each
other. You can also plot them in more appealing ways.
figure
imagesc(img_src);axis image
```



Ex 3.9 Aligning Images with known orientation

Medical images often have less local structure, making SIFT matching more difficult. It often works better if we drop the rotation invariance. The function `extractFeatures` has an option for this.

```
[descs, validPoints] = extractFeatures(img,points,'Upright',true);
```

The above assumes that the image has a default orientation. Modify your `align_images` function to create function `align_images_v2` so that it also takes a boolean argument, `upright` stating whether the images have the same orientation, i.e.

```
warped = align_images_v2(source, target, threshold, upright)
```

Try aligning the images *CT_1.jpg* (as target) and *CT_2.jpg (as source)*. Try with (upright = false) and without (upright = True) rotation invariance at different outlier thresholds. Also plot the warped image and the target image next to each other (or on top of each other with opacity < 1) so one can verify the result.

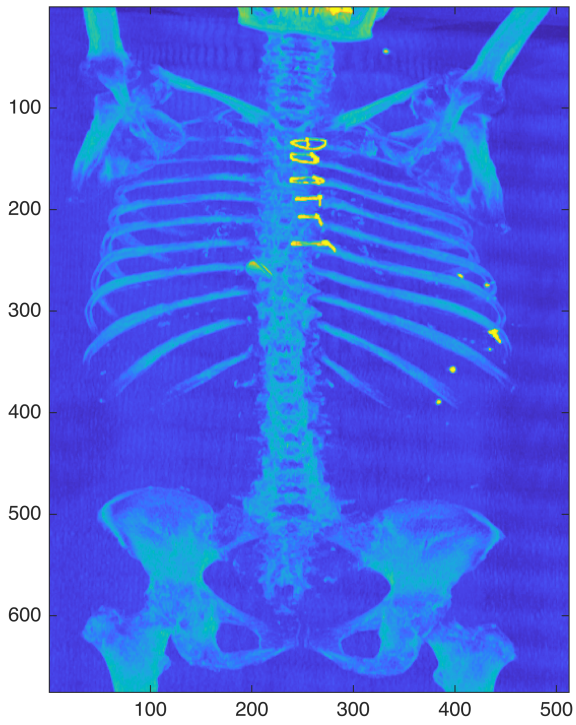
Note: You might have to play around a bit with the threshold value to get a reasonable alignment. If you are unsuccessful, you can also use SURF Features instead of SIFTFeatures.

```
%%% Your code here
```

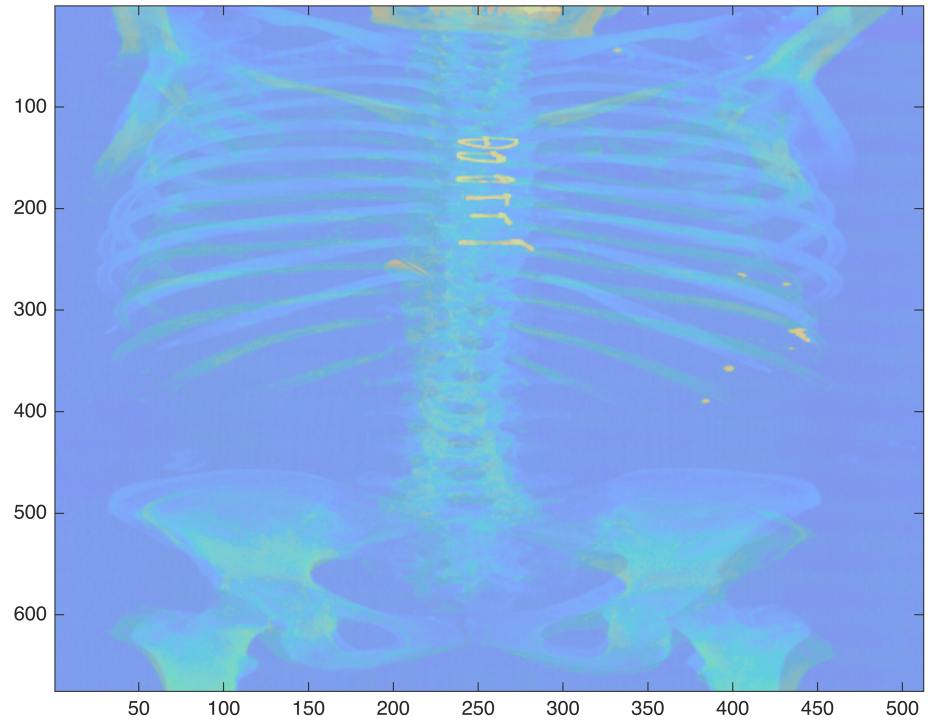
```
source = imread('CT_1.jpg');
target = imread('CT_2.jpg');

threshold = 25;
upright = true;

warped = align_images_v2(source, target, threshold, upright);
```

```
====================================
RANSAC iterations done!
It took k=3986 iterations to terminate...
Residual: 131037.25
Number of inliers: 7
====================================
```

```
%% Example simple code for plotting images on top of each other
figure
imagesc(warped);axis image
```




```
imagesc(target, "AlphaData", 0.4); hold on;
imagesc(warped, "AlphaData", 0.4);
```



Explain your observations of using and not using rotation invariance at differnt thresholds here.

```
%%
% When we don't use rotational invariance the image is kind of aligned, but
% if we use rotational invariance the results seem to have a heavy
% distortion and sometimes are unwatchable.
```

Ex 3.10 Fluorescent Image example

In this exercise, you should use the two images *'tissue_fluorescent.tif'* and *'tissue_brightfield.tif'*. Fluorescent images are inverted versions of regular images. MATLAB's SIFT implementation is unable to find good matches between the two images, so we provide you the matches extracted using a SIFT library called vl_sift.

The matched points between *tissue_fluorescent* and *tissue_brightfield* are provided to you in the file **matches_ex10.mat**. Use your function *ransac_fit_affine* to align the two images and then warp the source to target.

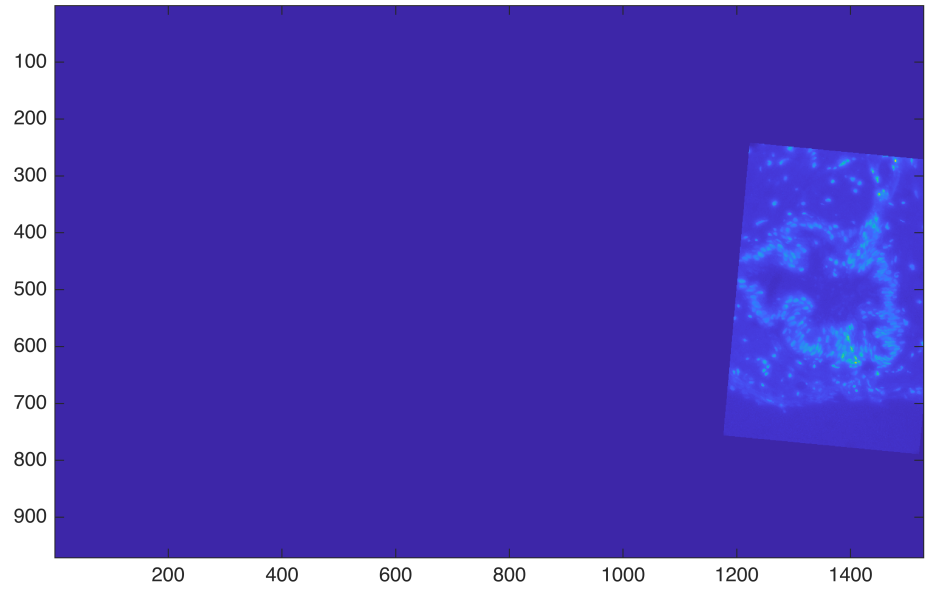
Plot the warped image and the target image side by side (or on top of each other with opacity < 1) so one can verify the result.

```
%%% Your code here
% Load image and match point data
source = imread('tissue_fluorescent.tif');
target = imread('tissue_brightfield.tif');
load('matches_ex10.mat');
```

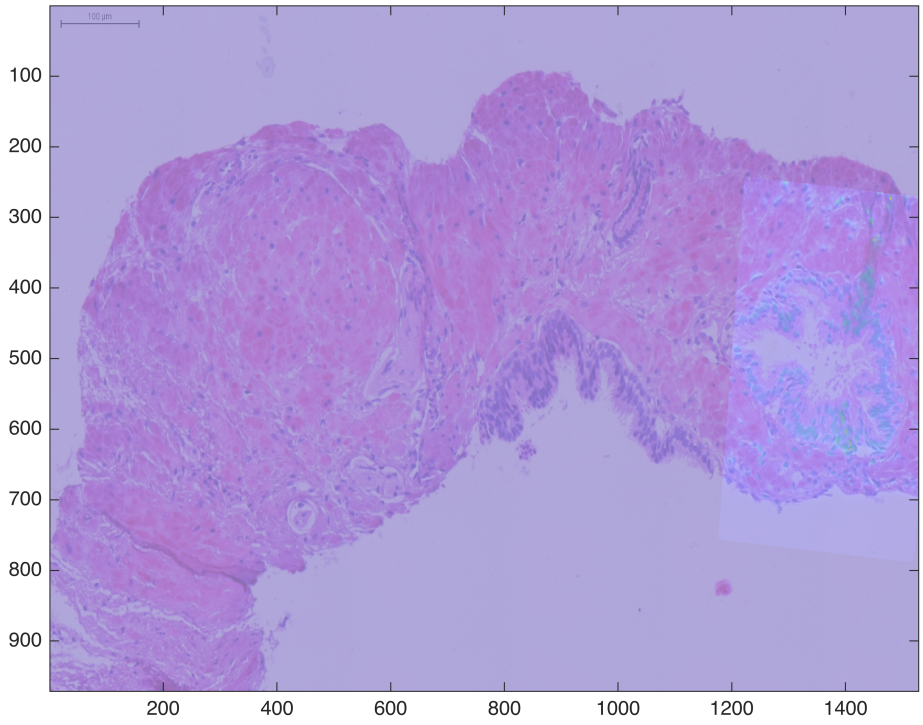
```
threshold = 10;
[A, t] = ransac_fit_affine(matches_ex10.pts_fluorescent,
matches_ex10.pts_brightfield, threshold);
```

```
====================================
RANSAC iterations done!
It took k=77 iterations to terminate...
Residual: 205994.55
Number of inliers: 41
====================================
```

```
warped = affine_warp(size(target), source, A, t);

%% Example simple code for plotting images on top of each other
figure
imagesc(warped); axis image
```




```
imagesc(target, "AlphaData", 0.4); hold on;
imagesc(warped, "AlphaData", 0.4);
```



Warping

Ex 3.11 Get Pixel value

So far you have used Matlab's function for warping. The reason is that it is difficult to write a Matlab function for warping that is not painfully slow. Now you will get to write one anyway, but we will only use it for very small images.

Write a function

```
value = sample_image_at(img, position)
```

that gives you the pixel value at `position`. If the elements of `position` are not integers, select the value at the closest pixel. If it is outside the image, return 1 (=white). Try your function on a simple image to make sure it works.

```
%% Your code here
img = rgb2gray(imread('mona.png'));
sample_image_at(img, [10,10])
```

```
ans = uint8

    154
```

Ex 3.12 Warp

Now, you will do a warping function that warps a 16 x 16 image according to the coordinate transformation provided in `transform_coordinates.m`.

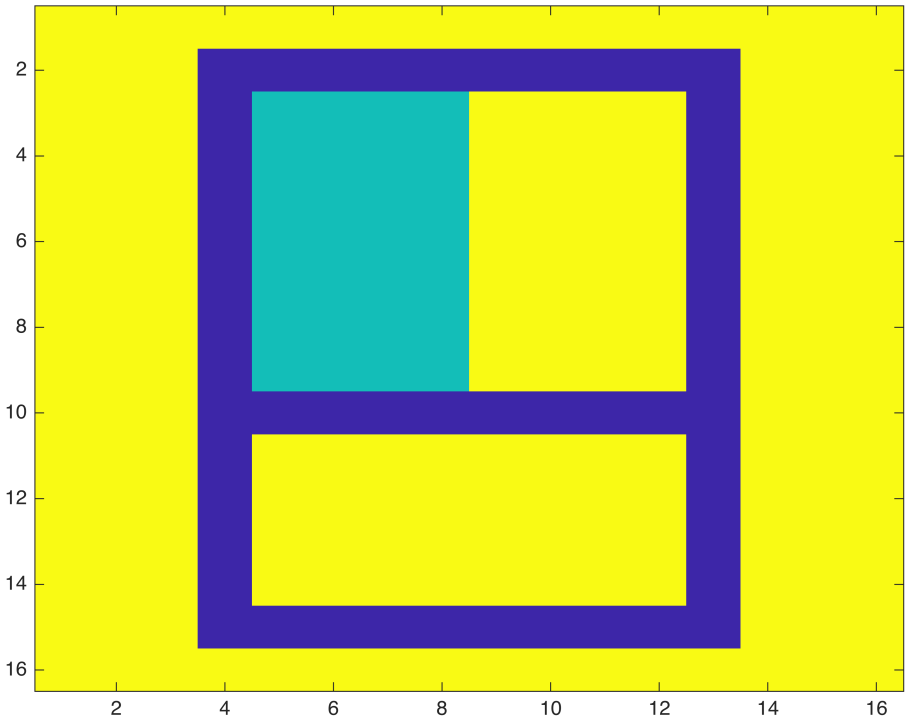
Write a function

```
warped = warp_16x16(source)
```

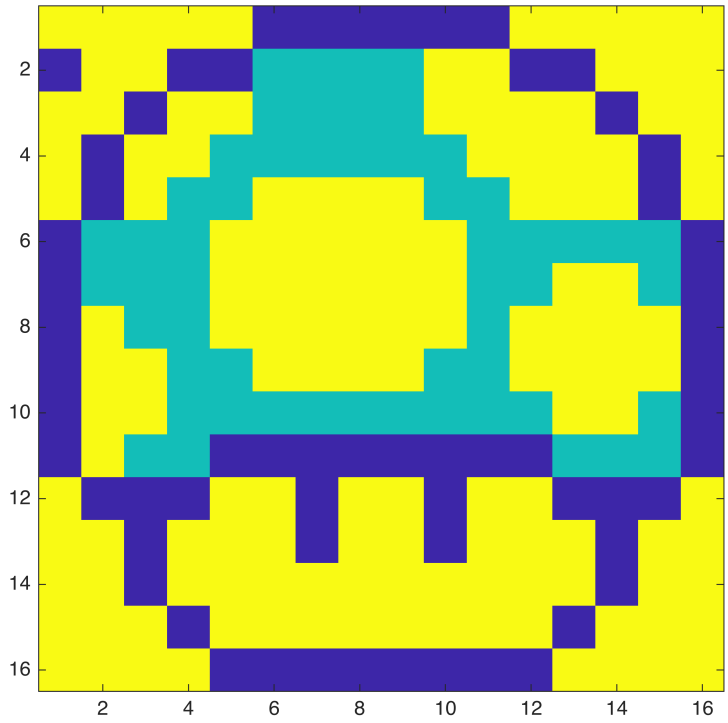
that warps **source** according to **`transform_coordinates`** and forms an output 16 X 16 image warped. More specifically, the value at pixel (i,j) of the warped image should be the value at pixel (i_t, j_t) of source, where (i_t, j_t) is obtained by applying transform_coordinates on (i,j). Use your function `sample_image_at` to extract pixel values. Try the function on `source_16x16.tif` and plot the answer using imagesc. You will get to see a meaningful image if you get it right.

Your code here

```
%% Your code here
source = imread('source_16x16.tif');
figure
imagesc(source)
```




```
warped = warp_16x16(source);
figure
imagesc(warped), axis image
```



Least Squares

Ex 3.13 Least squares

Write a function

```
[A, t] = least_squares_affine(pts, pts_tilde)
```

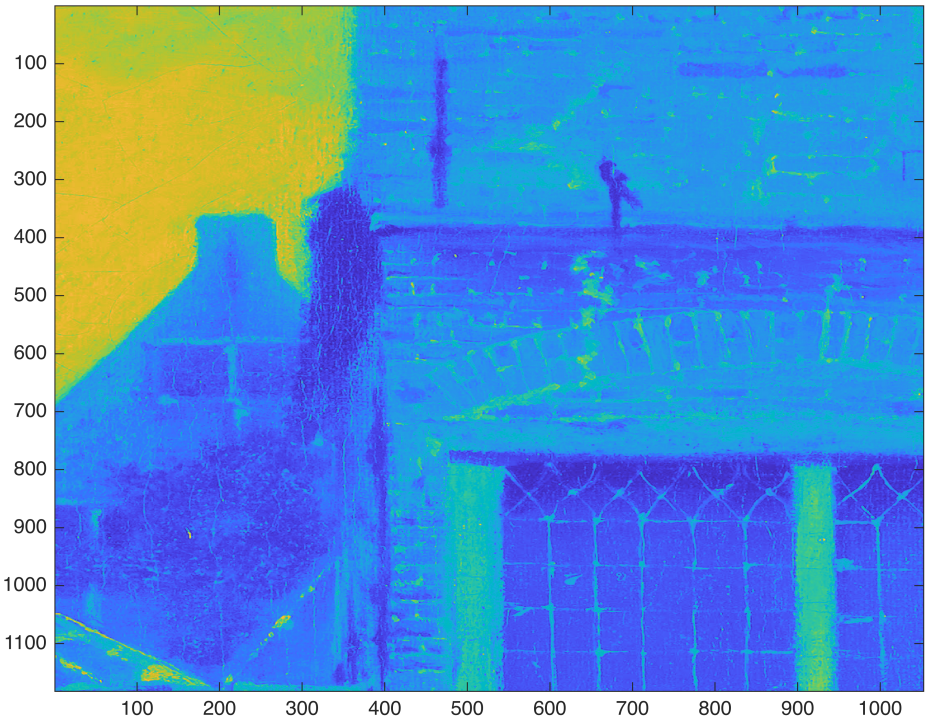
hat estimates an affine transformation mapping `pts` to `pts_tilde` in least squares sense, i.e., all points in `pts` and `pts_tilde` are used to compute the transformation. (Depending on how you wrote you estimate_affine.m, this might be very easy.) Use it on the inliers from RANSAC to refine the estimate. Add this to `align_images` to form a new function **`align_images_inlier_ls`** and test it on the `Vermeer` images for different outlier thresholds. Instead of extracting the SIFT features, use the noisy points and matches provided to you in **`data_ex13.mat`**. Plot the final resulting images next to each other. Also, check the mean/ median value of the residuals over all inlier points in the two cases.

Hint: Start with a low inlier ratio (e.g. 0.05) estimate to make sure your RANSAC runs for sufficient number of iterations and to be able to see a difference.

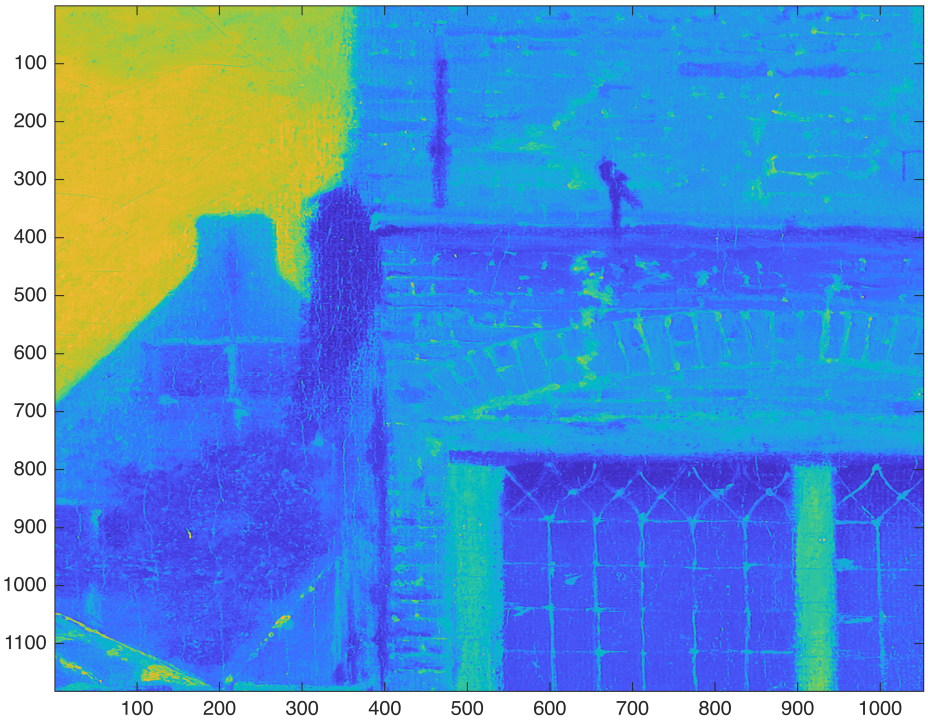
```
%% Your code here
source = rgb2gray(imread('vermeer_source.png'));
target = rgb2gray(imread('vermeer_target.png'));

for threshold = [30,50,70,100]
    fprintf('\n\n\nAligning images with threshold=%.0f...',threshold)
```

```
    warped = align_images_v2(source, target, threshold, false);
    figure, imagesc(warped),imagesc(source),imagesc(target);

    warped_ls = align_images_inlier_ls(source, target, threshold, false);
    figure, imagesc(warped_ls),imagesc(source),imagesc(target);

end
```

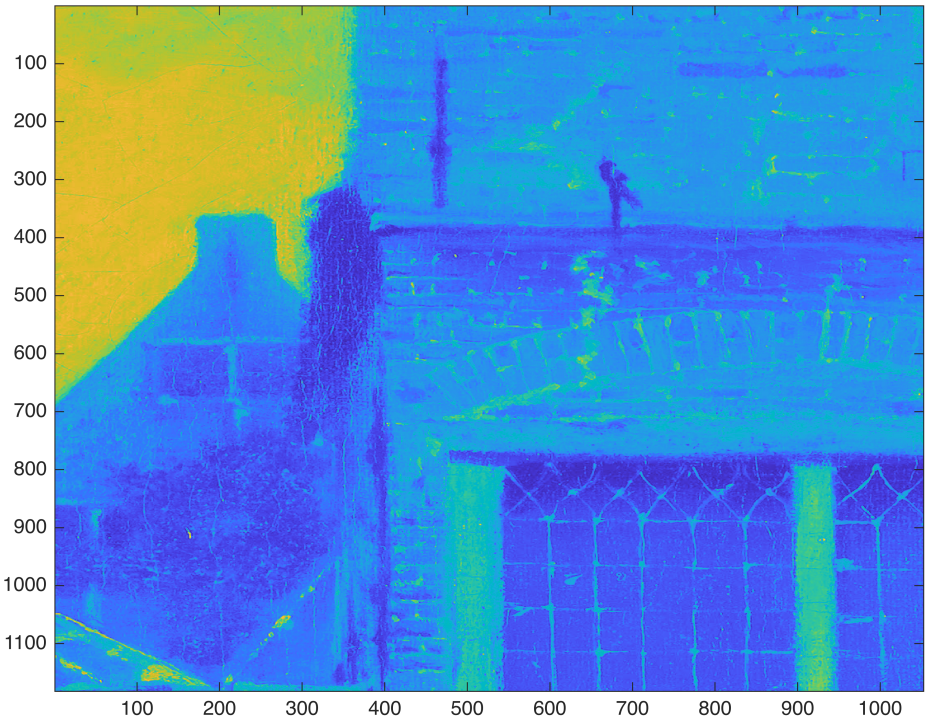
```
Aligning images with threshold=30...
=====================================
RANSAC iterations done!
It took k=15929 iterations to terminate...
Residual: 1881966.28
Number of inliers: 22
=====================================
```



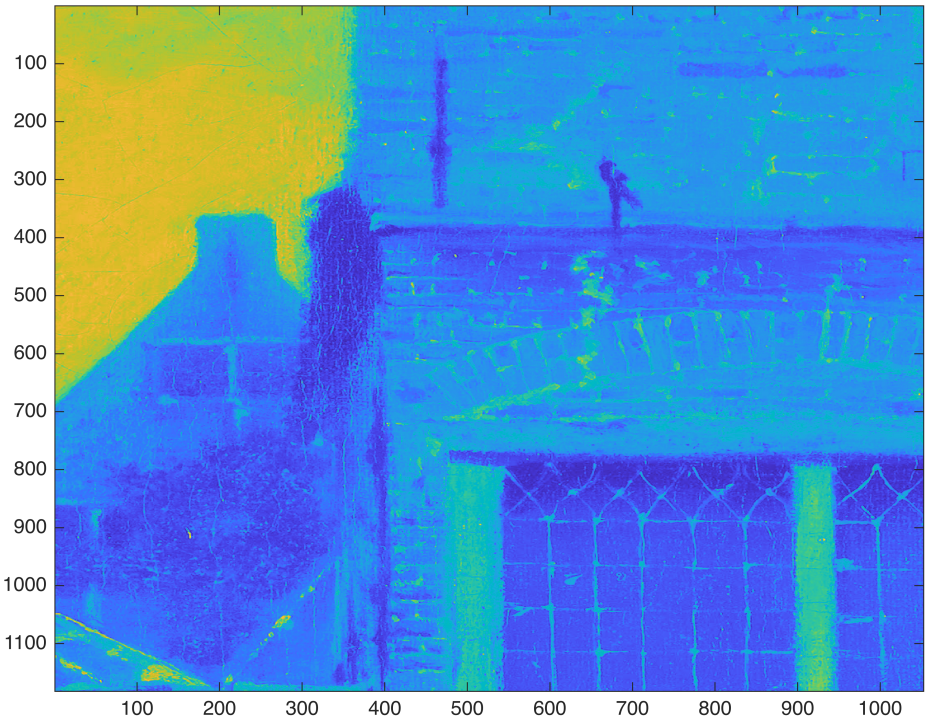

```
=====================================
RANSAC with LS iterations done!
It took k=140 iterations to terminate...
Residual: 1468595.21
Number of inliers: 367
=====================================
```



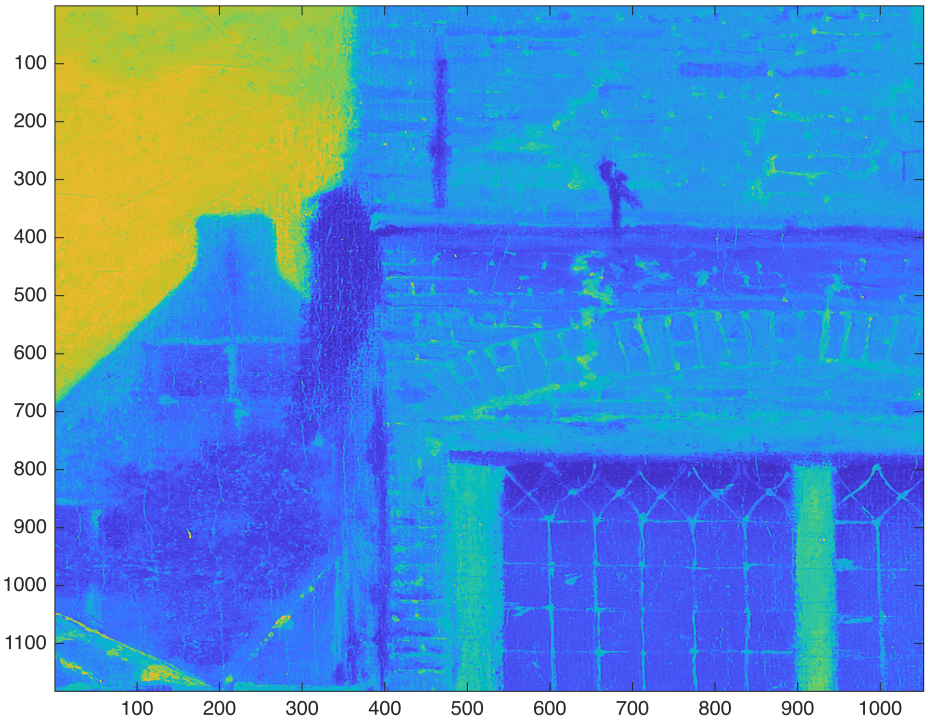

```
Aligning images with threshold=50...
=====================================
RANSAC iterations done!
It took k=8616 iterations to terminate...
Residual: 1875352.19
Number of inliers: 27
=====================================
```



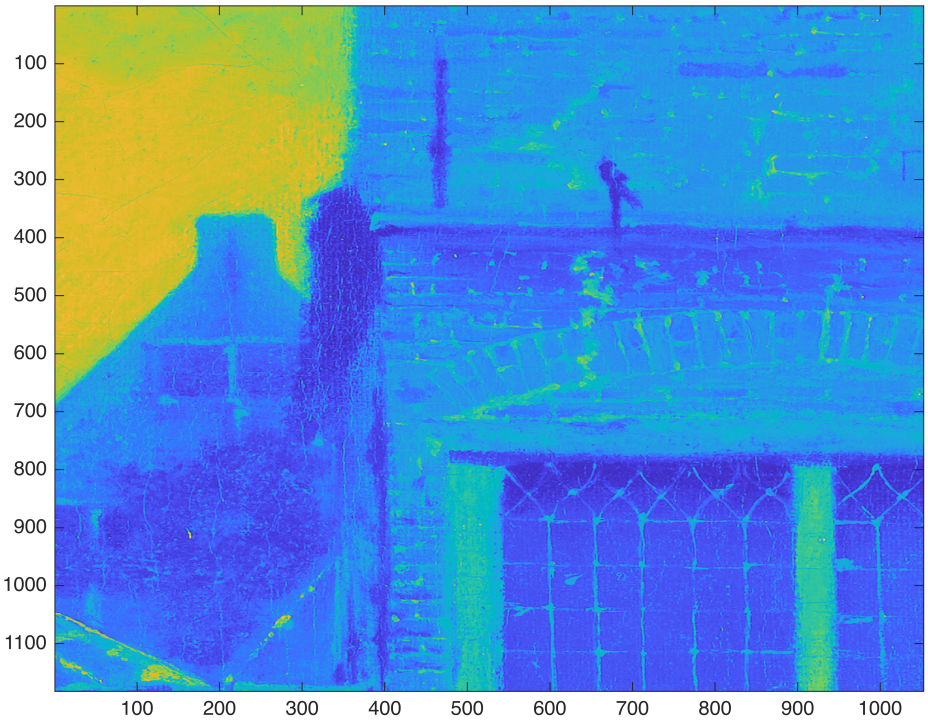

```
=====================================
RANSAC with LS iterations done!
It took k=139 iterations to terminate...
Residual: 1466907.99
Number of inliers: 368
=====================================
```



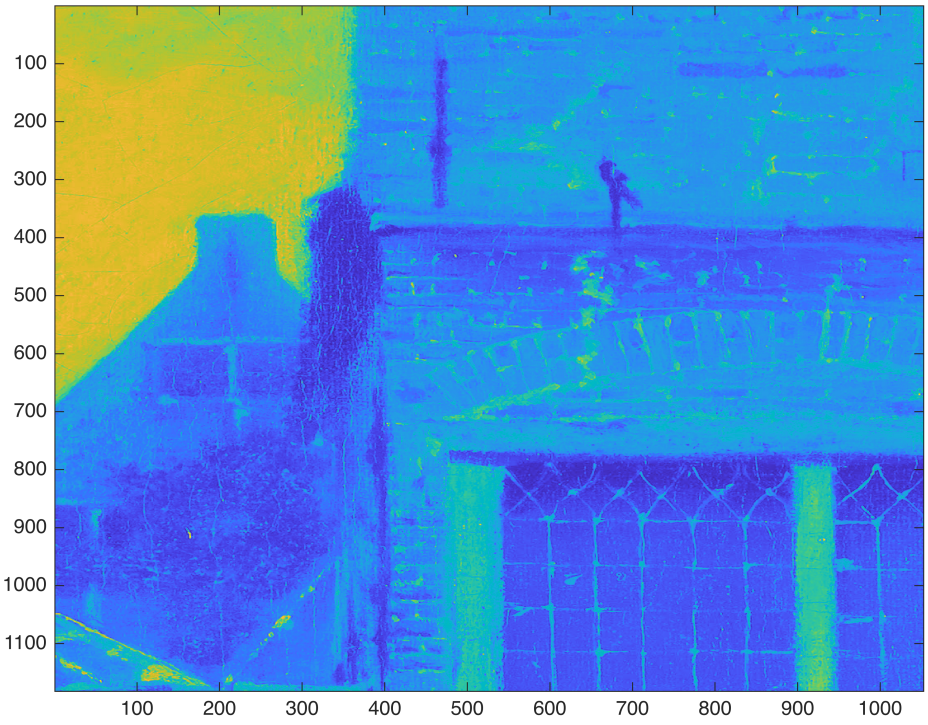

```
Aligning images with threshold=70...
=====================================
RANSAC iterations done!
It took k=10941 iterations to terminate...
Residual: 1913197.31
Number of inliers: 26
=====================================
```



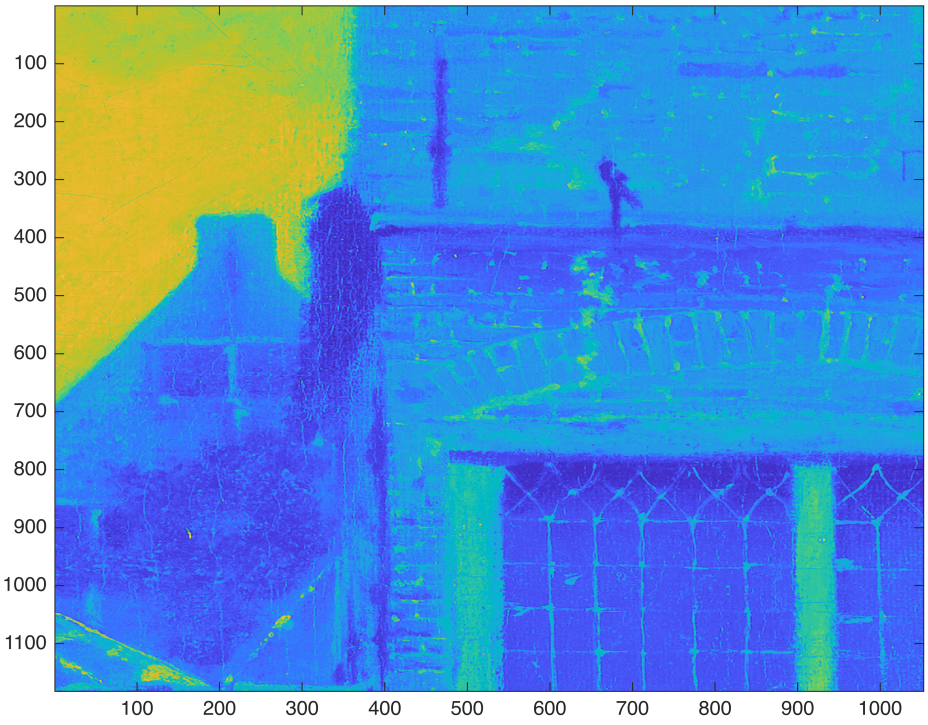

```
=====================================
RANSAC with LS iterations done!
It took k=139 iterations to terminate...
Residual: 1466809.43
Number of inliers: 368
=====================================
```




```
Aligning images with threshold=100...
=====================================
RANSAC iterations done!
It took k=5692 iterations to terminate...
Residual: 311884.67
Number of inliers: 31
=====================================
```




```
=====================================
RANSAC with LS iterations done!
It took k=137 iterations to terminate...
Residual: 1466315.50
Number of inliers: 370
=====================================
```



Do you notice an improvement? Observe visually and also in terms of the residuals. **Explain your observations and the plausible reasons behind them as comment here. (It is okay to not see an improvement, but try to explain the behaviour.)**

```
%%
%We don't notice any particular difference. After some tries we noticed
%that when the threshold is lower we find more inliears wrt the previous
%implementations. This may be due to the fact that  least-squares are less
%supscetible to outliers.
```