CS143: Introduction to Computer Vision
Homework Assignment 3
Affine motion and Image Registration

Due: November 3 at 10:59am - problems 1 and 2
Due: November 9 at 10:59am - all problems

The assignment is worth 15% of your total grade.
It is graded out of a total of 100 (plus 20 possible extra points).

Goals.

This assignment introduces you to affine motion estimation. Once again it builds upon what you have already done; in particular, you will use image pyramids and derivative filters.

You will use an affine motion estimation algorithm to solve a real problem: aligning multiple low-quality views of a license plate to reconstruct a single higher quality view of that same license plate.

Summary

Files you need.

For problems 2 and 3 you will be asked to use the following two images:

`/course/cs143/asgn/asgn3/boat.tif`

`/course/cs143/asgn/asgn3/warped_boat.tif`

As well as a set of affine transformations generated by running:

`/course/cs143/asgn/asgn3/generate_test_affine_transforms.m`

For problem 4 you will use the (pseudo)surveillance sequence found at:

`/course/cs143/data/LicensePlates/edited`

And for the extra credit problem, the data is located at:

`/course/cs143/data/Police/Vehicle1`

Additionally, we provided a number of Matlab functions and function headers in the directory `/course/cs143/asgn/asgn3/`, which you should start from. Their main purpose is to make it easier for us to grade your homework. You are therefore asked to copy those files and modify them, so that they perform the appropriate task and display your results. You are of course free to write your own scripts and/or functions and only add calls to these to the scripts we provided.

What to hand in.

Please hand in enough information for us to understand what you did, what things you tried, and how it worked!

You will hand in your Matlab code, which is supposed to show all your results, and any text or explanations that are required. Please hand in the modified `problem[2-4].m` scripts, the modified functions (see above), as well as any other Matlab scripts or functions that your solution requires. Make sure that all your results are displayed properly!

You will also hand in any text or explanations electronically. They should be in a separate file than the source code, and either in plain text format, in Postscript, or in PDF.

To handin the assignment run the commands below from the folder that contains your solutions. The entire contents of that folder will be zipped and sent to us.

- Problems 1 and 2:

  ```
  /course/cs143/bin/cs143_handin asgn3_p1_p2
  ```

- All problems

  ```
  /course/cs143/bin/cs143_handin asgn3all
  ```

Do all the following problems:

Problem 1 : The affine transformation

(10 points)

As discussed in class, the motion of planes in the world can often be approximated by an *affine transformation*. This is nothing more that a translation specified by parameters (a_3, a_6) applied after a linear transformation specified by parameters (a_1, a_2, a_4, a_5). So an affine transformation A is completely specified by its six parameters: $(a_1, a_2, a_3, a_4, a_5, a_6)$.

Let's first see how an affine transformation $A = (a_1, a_2, a_3, a_4, a_5, a_6)$ acts on some image location (a vector in the plane) $\begin{bmatrix} x & y \end{bmatrix}^T$. After the transformation is applied we obtain a new image location $\begin{bmatrix} x' & y' \end{bmatrix}^T$ given by:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_1 & a_2 \\ a_4 & a_5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a_3 \\ a_6 \end{bmatrix} \tag{1}$$

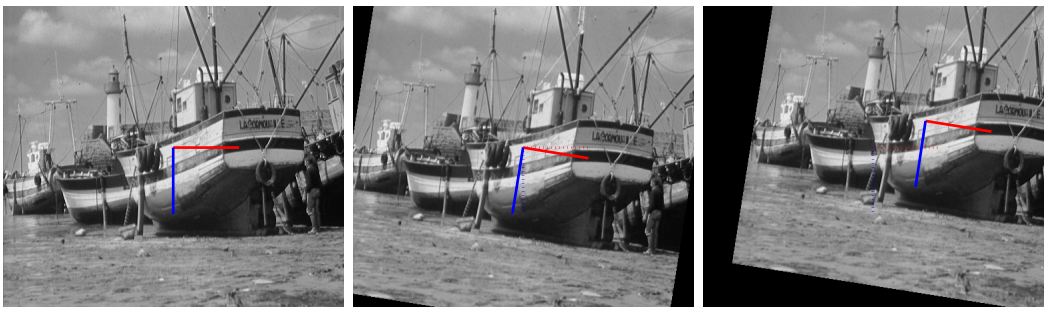
The expression above is equivalent to the more useful form:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2}$$

For shorthand we will write $\mathbf{r}' = A \cdot \mathbf{r}$ to mean "the vector $\mathbf{r}' = \begin{bmatrix} x' & y' \end{bmatrix}^T$ is obtained by applying the affine transformation A to the vector $\mathbf{r} = \begin{bmatrix} x & y \end{bmatrix}^T$"

a. (5 points) What is the inverse of an affine transformation? In other words, if $\mathbf{r}' = A \cdot \mathbf{r}$ find another affine transformation A^{-1} such that $A^{-1} \cdot \mathbf{r}' = \mathbf{r}$. **Hint:** Use equation 2.

b. (5 points) What is the composition of two affine transformations? In other words, if $\mathbf{r}' = A \cdot \mathbf{r}$ and $\mathbf{r}'' = B \cdot \mathbf{r}'$ find another affine transformation $C = B \cdot A$ such that $\mathbf{r}'' = C \cdot \mathbf{r}$. Is the composition of affine transformations commutative? In other words, is it always the case that $A \cdot B = B \cdot A$? **Hint:** As before, use equation 2.



(a) Original image (b) A linear transformation... (c) ... followed by a translation

Figure 1: Affine image warping

Problem 2 : Affine image warping

(20 points)

We are now ready to define affine transformations on images. Let J be the image obtained by applying the affine transformation A to some image I (for short, the image obtained by *warping* I). Viewing both I and J as functions, the resulting image, J, is defined as $J(\mathbf{r}) = I(A^{-1} \cdot \mathbf{r})$ for all vectors in the plane $\mathbf{r}$.

This definition makes sense only if there is a coordinate system associated with each image. For the purpose of this assignment, we will choose the coordinate system such that $(0, 0)$ corresponds to the center of the image. Also, we assume that a warped image will have the same size as the initial image.

To help you get some intuition on affine transformations, we will first provide you with out Matlab implementation of affine warping and ask you to perform a number of affine transformations using it. We will, then, ask you to implement your own version of this function to make sure you understand how it works.

```
function J = affine_warp(I,A)

T = maketform('affine',A');

xd = [1,size(I,2)] - size(I,2)/2;
yd = [1,size(I,1)] - size(I,1)/2;

ud = [1,size(I,2)] - size(I,2)/2;
vd = [1,size(I,1)] - size(I,1)/2;
```

```
J =imtransform(I,T,'FillValues',nan,...
    'XData',xd,'YData',yd,...
    'UData',ud,'VData',vd);
```

Notice that in the code for `affine_warp()` we are using `nan`'s (not-a-number) to represent missing data that lies outside the boundaries of the initial image but ends up lying within the boundaries of the warped image. However, we can also use `nan`'s to mask out flawed data or data we're not interested in. For the rest of the assignment we will assume that images may contain `nan`'s to represent missing data.

(5 points) Part I

Apply the following affine transformations to `boat.tif` using the `affine_warp()` function above. Display the result.

a. A translation

$$A_t = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 30 \\ 0 & 1 & 60 \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

b. A clockwise rotation by $\theta = \pi/4$ about the center of the image

$$A_r = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

c. A scaling by half in the horizontal direction and one third in the vertical direction

$$A_s = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1/2 & 0 & 0 \\ 0 & 1/3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

d. Apply $(A_t \cdot A_r)$ and then $(A_r \cdot A_t)$ to `boat.tif`. Does A_t commute with A_r? Now apply $(A_r \cdot A_s)$ and then $(A_s \cdot A_r)$ to `boat.tif`. Does A_r commute with A_s?

e. The image `warped_boat.tif` was obtained by first applying A_1 and then A_2 to `boat.tif`. In other words: $J = A_2 \cdot A_1 \cdot I$. If A_2 is the affine transformation given below, find and display $A_1 \cdot I$.

$$A_2 = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.9 & 0.1 & 10 \\ 0.1 & 0.9 & 20 \\ 0 & 0 & 1 \end{bmatrix} \tag{6}$$

(15 points) Part II

Implement your own version of `affine_warp()` by using the algorithm seen in class. Your implementation should behave like ours but use lower level functions such as `interp2()` and `meshgrid()` instead of the higher level function `imtransform()`.

Perform a few image warps using both functions to show us that that they behave the same.

For the rest of the assignment, you may use either version of affine image warping.

Problem 3 : Affine motion estimation

(50 points total)

The general problem we will address here is that of recovering the affine motion parameters given an initial image and its warped version. In other words, given two images I and J we want to recover the affine parameters A such that `J = affine_warp(I,A)`.

This problem becomes harder as the displacement between the two images increases. Thus, we need different algorithms for different amounts of motion. You will be asked to successively improve your algorithm to deal with "tiny", "small" and "moderate" displacements.

a. (30 points) "Tiny" displacements

Given two images I and J we set up following cost function:

$$E(a_1, a_2, ..., a_6) = \sum_{\mathbf{r} \in \mathcal{V}} (\texttt{affine\_warp}(I(\mathbf{r}), A) - J(r))^2 \tag{7}$$

$$E(a_1, a_2, ..., a_6) = \sum_{\mathbf{r} \in \mathcal{V}} (I(A^{-1}\mathbf{r}) - J(r))^2 \tag{8}$$

For simplicity, we will temporarily define: $B = A^{-1}$. The above expression becomes:

$$E(b_1, b_2, ..., b_6) = \sum_{\mathbf{r} \in \mathcal{V}} (I(B\mathbf{r}) - J(r))^2 \tag{9}$$

$$E(b_1, b_2, ..., b_6) = \sum_{\mathbf{r} \in \mathcal{V}} (I(xb_1 + yb_2 + b_3, xb_4 + yb_5 + b_6) - J(x, y))^2 \tag{10}$$

It's difficult to find the minimum of this cost function because it is not linear in the affine parameters. Consequently, we will approximate it using a Taylor series expansion about the point (x, y). We get the following approximate expression (as seen in class):

$$E_{approx}(b_i) = \sum_{\mathbf{r} \in \mathcal{V}} (I_x \cdot ((b_1 - 1)x + b_2 y + b_3) + I_y \cdot (b_4 x + (b_5 - 1)y + b_6) + I_t)^2 \tag{11}$$

Here $I_t = (I - J)$ and $\mathcal{V}$ is the set of all pixel location where the relevant quantities (I_x,I_y,I_t) are defined (i.e not `nan`'s).

Now finding the minimum of the cost function is manageable. We will impose that all the partial derivatives $\partial E_{approx}/\partial b_1$, $\partial E_{approx}/\partial b_2$, ... , $\partial E_{approx}/\partial b_6$ be equal to zero. Once you have the partial derivatives, you have many equations with same six unknowns. You can now rearrange the terms to write these equations in matrix form (this is a very common and useful thing to do!). By doing this we reduce our problem to solving a linear equation of the form $M \cdot X = Y$.

$$\begin{bmatrix} \sum I_x^2 x^2 & \sum I_x^2 xy & \sum I_x^2 x & \sum I_x I_y x^2 & \sum I_x I_y xy & \sum I_x I_y x \\ \sum I_x^2 xy & \sum I_x^2 y^2 & \sum I_x^2 y & \sum I_x I_y xy & \sum I_x I_y y^2 & \sum I_x I_y y \\ & & ... & ... & & \\ & & ... & ... & & \end{bmatrix} \cdot \begin{bmatrix} b_1 - 1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 - 1 \\ b_6 \end{bmatrix} = \begin{bmatrix} -\sum I_t I_x x \\ -\sum I_t I_x y \\ -\sum I_t I_x \\ -\sum I_t I_y x \\ -\sum I_t I_y y \\ -\sum I_t I_y \end{bmatrix}$$

Carefully compute the missing elements of the six by six matrix. As you know from previous assignments, if you get the math wrong, this will not work.

Also, make sure that the derivative filters you are using are properly normalized (Look at the matrix equation and convince yourself that multiplying the image derivatives by some constant influences your result). Here is a simple derivative filter that will work reasonably well: `[-1,0,1]/2`. Of course, you are

free to use any derivative filter you want. Your choice will influence the accuracy of the algorithm.

Now, fill in the missing parts of the code below. You will find the following Matlab functions extremely useful: `nansum()`, `meshgrid()`. very useful.

```
function A = tiny_affine_flow(I,J)

Ix = ?
Iy = ?
It = I - J;

M = ?        % compute the six by six M matrix
Y = ?
X = ?        % solve M*X=Y

B = ?        % recover B = A^-1 from X
A = ?        % recover A from B = A^-1
```

To test this function we ask you to apply made up affine transformations to `boat.tif` to generate pairs of images. Then try to recover the known transformations using `tiny_affine_flow()`.

To show us that your code is working properly, do this for all the transformations produced by `generate_test_affine_transforms.m` and report the results.

What is the largest displacement that can still be considered "tiny"? (give an approximate answer in pixels).

b. (10 points) "Small" displacements

As you know, Taylor's approximation only works when the displacements are sufficiently small. To deal with larger displacements we have to use the exact affine transformation function to some extent. The following algorithm will do this. Your job is to understand how it works and fill in the missing parts.

```
function A = small_affine_flow(I,J,A)
             % A is the initial guess
             % for the affine parameters

while ?      % Halt when convergence is detected
```

```
            % make sure you don't iterate more
            % than 20 times.
dA = tiny_affine_flow(affine_warp(I,A),J);

A = compose(A,dA)
            % Remember that composition
            % is not commutative.
            % Argue if this should be A*dA or dA*A
end;
```

To test this function we ask you to apply made up affine transformations to `boat.tif` to generate pairs of images. Then try to recover the known transformations using `small_affine_flow()`.

To show us that your code is working properly, do this for all the transformations produced by `generate_test_affine_transforms.m` and report the results.

What is the largest displacement that can still be considered "small"? (give an approximate answer in pixels).

c. (10 points) "Moderate" displacements

We can solve for even larger displacements if we observe that displacements become smaller as we reduce the scale of images. Even though the motion at the largest scale is not "small" enough for `small_affine_flow()` it is almost always the case that the motion at a coarser scale is "small" enough. So we can solve for the motion at the coarsest scale first and then use the result to get a good starting guess for the next finest scale. We repeat this until we reach the top (largest) scale.

Your job is to implement this coarse-to-fine algorithm as:

```
function A = moderate_affine_flow(I,J,A)
```

Having a nice representation of Gaussian pyramids will help you here. Also, keep in mind that the magnitude of the flow at each level is half of that of the finer lever in the pyramid.

To test this function we ask you to apply made up affine transformations to `boat.tif` to generate pairs of images. Then try to recover the known transformations using `moderate_affine_flow()`.

To show us that your code is working properly, do this for all the transformations produced by `generate_test_affine_transforms.m` and report the results.

What is the largest displacement that can still be considered "moderate"? (give an approximate answer in pixels).

Problem 4 : Image registration

(20 points)

Now that we have the tools to recover affine flow parameters, we can use them to solve a real-life problem.

At `/course/cs143/data/LicensePlates/edited` you will find a number of consecutive frames from a (pseudo)surveillance video that shows a truck driving by. We would like to identify the vehicle, but, due to the low quality of the images, the characters on the license plate are illegible in all frames.

To get a better image, we assume that the motion of the license plate is affine and solve for the affine transformations between every pair of frames in the surveillance sequence. Then we warp each frame so that the license plate appears at the same location in all of them. This is called image registration. Once we have registered the frames we can average them to get a better quality image of the license plate (mind the `nan`'s. Use `nanmean()`). Intuitively, we are "combining" information from all the low quality frames into a single high quality frame.

Your job is to implement this algorithm, show us the mean image of the license plate and attempt to read it. Here are a few hints:

- Use the masks provided to set all pixels that are not part of the truck's front grille to `nan`'s. If they are anything else (such as 0), your registration will not work.

- Unless given a good initial guess, the affine flow algorithm will only produce reliable results for consecutive frames. To find displacements between non-consecutive frames you will have to compose displacements between consecutive frames to get a good starting guess for the affine flow algorithm.

- Although you need all frames for registration, you don't have to use all of them to compute the mean image. Ignoring frames that are too blurry will most likely improve your results.

Graduate students are required to do problem 5.

Undergraduates can choose to do problem 5 for 10 points extra credit.

Problem 5 : Real crime video

At `/course/cs143/data/Police/Vehicle1` you can find a real surveillance video. The police would like to know what the license plate of the red car is and you job is to provide them an image that makes it readable.

Modify your affine motion estimation algorithm to use all three color channels of the sequence effectively. Then register the frames and compute a mean image.