MCV172, HW#1 – Parts I and II

Oren Freifeld

Part 1 was released on April 3 2017
Part 2 was released on April 8, 2017

Contents

Few Words Before You Start

1. If you are going to use Python+OpenCV, this link may be useful for you: `http://docs.opencv.org/3.0-beta/modules/imgproc/doc/filtering.html`
2. Throughout all computer exercises below, do not implement the convolutions yourself; rather, use standard implementations such as those in scipy.ndimage or OpenCV (*e.g.*, cv2.filter2D). For oft-used filters, there are usually already functions that will spare you the need to pass the filter as an argument (*e.g.*, cv2.GaussianBlur).
3. Get the data: `https://www.cs.bgu.ac.il/~mcv172/wiki.files/hw1_data.tar`
4. Some of the data is saved as mat files. To read these in python, you can use scipy.io.loadmat, which returns a dictionary. For example:

```
Code:
import scipy.io as sio
mdict = sio.loadmat('are_these_separable_filters.mat')
print [k for k in m.keys() if not k.startswith('__')]
Output:
['K3', 'K2', 'K1']
```

so then you can get the data more directly as:

```
K1=mdict['K1']
K2=mdict['K2']
K3=mdict['K3']
```

1 Linear Filtering

1.1 Convolution

The 2D correlation of two 2D digital arrays, I and h is given by

$$\begin{aligned} g &= I \otimes h = h * I \\ g(i,j) &= \sum_{k,l} I(i+k, j+l) h(k,l) \end{aligned} \tag{1}$$

The 2D convolution of two 2D digital arrays, I and h is given by

$$\begin{aligned} g &= I * h = h * I \\ g(i,j) &= \sum_{k,l} I(i-k, j-l) h(k,l) = \sum_{k,l} h(i-k, j-l) I(k,l)\,. \end{aligned} \tag{2}$$

The summation is done over all relevant pixels (*i.e.*, where h is defined).

A slightly different way of writing the same thing is as

$$g(\boldsymbol{x}) = \sum_{\boldsymbol{x}_i} I(\boldsymbol{x} - \boldsymbol{x}_i) h(\boldsymbol{x}_i) = \sum_{\boldsymbol{x}_i} h(\boldsymbol{x} - \boldsymbol{x}_i) I(\boldsymbol{x}_i) \tag{3}$$

For example, if h is Gaussian, then

$$g(\boldsymbol{x}) = \frac{1}{c} \sum_{\boldsymbol{x}_i} \exp\left(-\frac{1}{2} \frac{(\boldsymbol{x} - \boldsymbol{x}_i)^2}{\sigma^2}\right) I(\boldsymbol{x}_i) \tag{4}$$

where c typically is taken as a normalizer: $c = \sum_{\boldsymbol{x}_i} \exp\left(-\frac{1}{2} \frac{(\boldsymbol{x} - \boldsymbol{x}_i)^2}{\sigma^2}\right)$. You should recognize this type of expression from our discussion on, *e.g.*, the Lucas-Kanade method.

Exercise 1 *Show that this operation is linear. In effect, show that if I_1 and I_2 are 2D digital arrays, and $c_1, c_2 \in \mathbb{R}$, then*

$$(c_1 I_1 + c_2 I_2) * h = c_1 (I_1 * h) + c_2 (I_2 * h) \tag{5}$$

◊

Let $I \in \mathbb{R}^{M\times N}$. Let $h \in \mathbb{R}^{m\times n}$. Since $g = I * h$ is linear, and since g is also $M \times N$, we may also write this operation as

$$\underbrace{\boldsymbol{y}}_{(MN)\times 1} = \underbrace{\boldsymbol{H}}_{(MN)\times(MN)} \underbrace{\boldsymbol{x}}_{(MN)\times 1} \tag{6}$$

where $\boldsymbol{x}$ is a vectorized version of I and and $\boldsymbol{y}$ is a vectorized version of g. The values of $\boldsymbol{H}$ are determined by h. If $\boldsymbol{h}$ is small (or large but sparse) then $\boldsymbol{H}$ is sparse (*i.e.*, most of its elements are zero).

Exercise 2 *Assume a zero-boundary condition. In effect, in 1-based indexing, we set* $I(i-k, j-l) = 0$ *whenever at least one of the following is true:* $i-k \le 0$*;* $i-k > M$*;* $j-l \le 0$*;* $j-l > N$*. Assume the vectorization is done by stacking the rows on top of each other, and that* $h \in \mathbb{R}^{3\times 3}$*:*

$$\begin{aligned}
\boldsymbol{x} &= \begin{bmatrix} I(1,1) & \dots & I(1,N) & \dots & I(M,1) & \dots & I(M,N) \end{bmatrix}^T ;\\
\boldsymbol{y} &= \begin{bmatrix} g(1,1) & \dots & g(1,N) & \dots & g(M,1) & \dots & g(M,N) \end{bmatrix}^T ;\\
h &= \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} \\ h_{2,1} & h_{2,2} & h_{2,3} \\ h_{3,1} & h_{3,2} & h_{3,3} \end{bmatrix} \in \mathbb{R}^{3\times 3} .
\end{aligned}$$

Write the expression for the entries of $\boldsymbol{H}$*.* ◊

Computer Exercise 1 *In continue to the previous exercise, let* $M = 12$ *and* $N = 16$*. Implement* $\boldsymbol{H}$ *for the three different filters,*

$$h_1 = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad h_2 = \frac{1}{8}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad h_3 = h_2^T , \tag{7}$$

and display the corresponding results (using imshow; make sure to use titles and colorbar). h_1 *is uniform blur.* h_2 *is the horizontal Sobel filter.* h_3 *is the horizontal Soble filter.*

Remark: if M *and* N *are much larger (e.g.,500), we can still easily implement* $\boldsymbol{H}$ *using a sparse matrix (e.g., using scipy.sparse.lil). But for displaying purposes, we will have to convert it to a dense matrix, and then memory might be an issue.* ◊

```
Some Python commands you may find useful in the exercise above:
a.ravel; a.reshape (when a is a numpy array).
```

Exercise 3 *Given* $g \in \mathbb{R}^{M\times N}$ *and* $\boldsymbol{h} \in \mathbb{R}^{m\times n}$ *but not* I*, and assuming* $g = I*h$*, under what condition on* h *can we invert the convolution and recover* I*?* ◊

Exercise 4 *Let*

$$h = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} . \tag{8}$$

What is the effect of convolving an image with this $\boldsymbol{h}$*? (optional: try it yourself on a computer to verify your answer)* ◊

Computer Exercise 2 *The image used here is **mandrill.png**. Display it and the results for convolving it with each one of the following filters: an isotropic Gaussian with $\sigma = 3$; an isotropic Gaussian with $\sigma = 11$; a uniform 21×21 blur kernel (*e.g.*, in Python: h=np.ones((3,3))/21).* ◊

```
Some Python commands you may find useful for the exercise above:
scipy.ndimage.gaussian_filter; scipy.ndimage.uniform_filter;
cv2.blur,cv2.GaussianBlur.
```

Computer Exercise 3 *Get K_1, K_2 and K_3 from* `are_these_separable_filters.mat`*. Using svd, determine, based on the singular values of these filters, which one is separable and which is not. Remark: due to numerical errors, you are not going to get perfect zeros even if the filter is separable. You will have to settle for treating very small numbers (*e.g.*, 10^{-12}) as zero.* ◊

```
Some Python commands you may find useful for the exercise above:
scipy.linalg.svd, np.diag, np.allclose.
```

1.2 More General Filters

The convolution operator (similarly to the correlation operator) is not just a linear operation, it is also space invariant (or also called location invariant).

More generally, we can define the following operation: I and h is given by

$$g(i,j) = \sum_{k,l} I(i-k, j-l)h_{i,j}(k,l)\,. \tag{9}$$

The difference from convolution is that here the filter depends on the location, (i, j). For example, consider a situation where we want to blur the image but also have reasons to expect more noise on the right part of the image. In which case, we may want to use a Gaussian filter such that its standard deviation increases with x. While space-dependent, this kind of filter is still not a function of I.

An even more general approach is to adapt the filter to the image itself; *i.e.*, make it a function of I. For example, suppose we want to blur an image but do it in a way that preserves edges. There are several ways to go about it. One of them is what is called bilateral filtering:

$$g(\boldsymbol{x}) = \frac{1}{c}\sum_{\boldsymbol{x}_i} f^r(|I(\boldsymbol{x}) - I(\boldsymbol{x}_i)|)f^d(|\boldsymbol{x}) - \boldsymbol{x}_i|)I(\boldsymbol{x}_i) \tag{10}$$

$$c = \sum_{\boldsymbol{x}_i} f^r(|I(\boldsymbol{x}) - I(\boldsymbol{x}_i)|)f^d(|\boldsymbol{x}) - \boldsymbol{x}_i|) \tag{11}$$

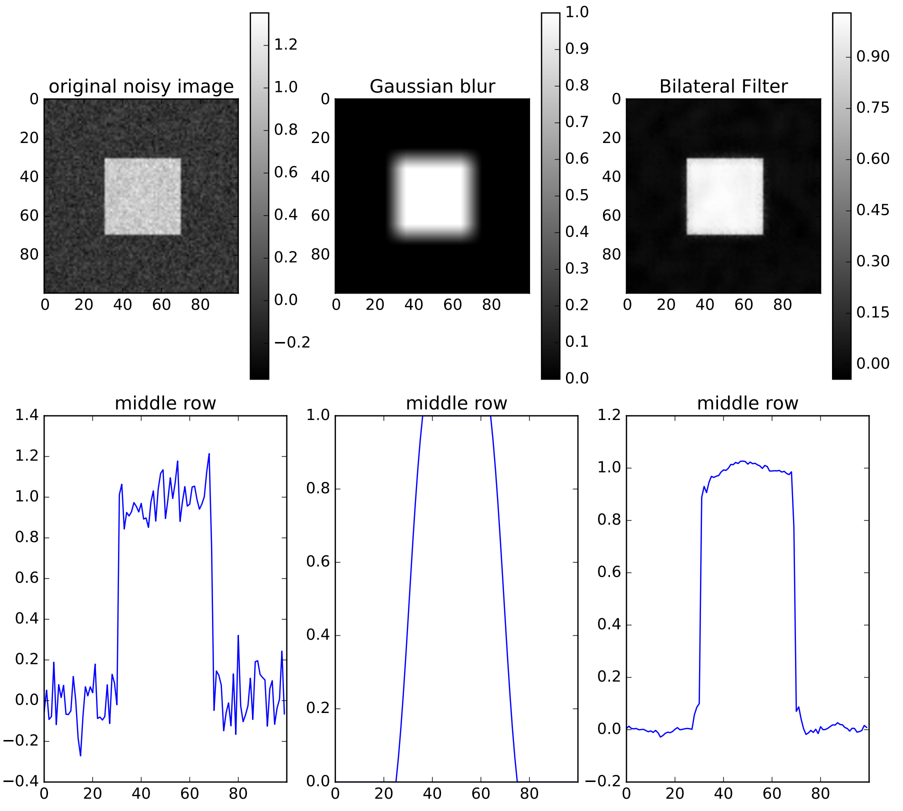
where f_r controls the weight as a function of the distance in the range (*i.e.*, intensity difference) and f_d controls the weight as as function of the distance in the domain (*i.e.*, spatial distance) For example we can take both these to be Gaussians, of different variances:

$$g(\boldsymbol{x}) = \frac{1}{c}\sum_{\boldsymbol{x}_i} \exp\left(-\frac{1}{2}\frac{I(\boldsymbol{x}) - I(\boldsymbol{x}_i))^2}{\sigma_r^2}\right)\exp\left(-\frac{1}{2}\frac{(\boldsymbol{x} - \boldsymbol{x}_i)^2}{\sigma_s^2}\right)I(\boldsymbol{x}_i) \tag{12}$$

$$c = \sum_{\boldsymbol{x}_i} \exp\left(-\frac{1}{2}\frac{I(\boldsymbol{x}) - I(\boldsymbol{x}_i))^2}{\sigma_r^2}\right)\exp\left(-\frac{1}{2}\frac{(\boldsymbol{x} - \boldsymbol{x}_i)^2}{\sigma_s^2}\right)\,. \tag{13}$$

Exercise 5 *Show that bilateral filtering is nonlinear.* ◊

The main purpose of such a filter is to preserve edges while blurring, as shown in the figure below:



Computer Exercise 4 *Get the (noisy) image from* `bilateral.mat` *(shown in the left column in the figure above), and, using bilateral filtering, get a result similar to the one in the rightmost figure above.* ◊

A Python command you may find useful for the exercise above: `cv2.bilateralFilter`

1.3 Laplacian and Laplacian of Gaussian

Exercise 6 *Calculate (by taking analytical derivatives) the Laplacian of a Gaussian;* i.e.,

$$\nabla^2 G(x,y,\sigma) \triangleq \frac{\partial^2 G(x,y,\sigma)}{\partial x^2} + \frac{\partial^2 G(x,y,\sigma)}{\partial y^2} = \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) G(x,y,\sigma)\,,$$

where

$$G(x,y,\sigma) = \frac{1}{2\pi\sigma^2}\exp(-\frac{x^2+y^2}{2\sigma^2})\,,$$

and show it is indeed given by

$$\nabla^2 G(x,y,\sigma) = \left(\frac{x^2+y^2}{\sigma^4} - \frac{2}{\sigma^2}\right) G(x,y,\sigma)\,.$$

Since $G(x,y,\sigma) = G(x^2+y^2,\sigma)$ (a sloppy notation indicating that $G(x,y,\sigma)$ depends on x and y only through $r^2 \triangleq x^2+y^2$) and since $\left(\frac{x^2+y^2}{\sigma^4} - \frac{2}{\sigma^2}\right)$ can be written as another function of x^2+y^2 (and σ), it is obvious that, at a given point (x,y), $\nabla^2 G(x,y,\sigma)$is invariant w.r.t. rotations of I about this point. It is tempting to think that this is solely because G is rotational invariant. However, it turns out that the Laplacian of I itself is also rotationally invariant:

Exercise 7 *Let*

$$\begin{aligned} u &= x\cos\theta - y\sin\theta \\ v &= x\sin\theta + y\cos\theta \end{aligned} \qquad \diamond$$

and show that, independently of the value of θ, it always holds (assuming I is twice differentiable of course) that

$$\nabla^2 I(x,y) = \nabla^2 I(u(x,y), v(x,y))\,;$$

i.e.*, that*

$$\frac{\partial^2 I(x,y)}{\partial x^2} + \frac{\partial^2 I(x,y)}{\partial y^2} = \frac{\partial^2 I(u(x,y),v(x,y))}{\partial u^2} + \frac{\partial^2 I(u(x,y),v(x,y))}{\partial v^2}\,.$$

Two 3×3 simple discrete approximations of the Laplacian operator are

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}\,. \tag{14}$$

Consistently with the previous exercise, we can see some (approximated) radial symmetry.

1.4 Directional Filters

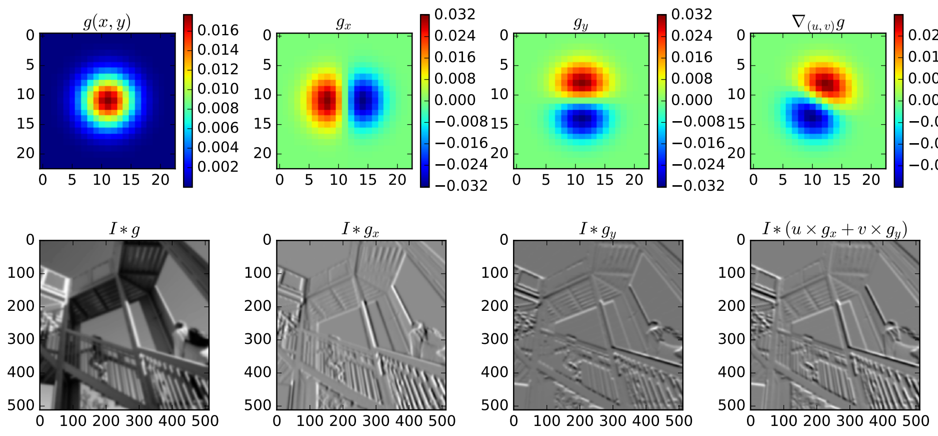
Let

$$g(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\frac{x^2+y^2}{\sigma^2}\right)\,.$$

Note that:

$$\frac{\partial}{\partial x} g(x,y) = -\frac{x}{\sigma^2} g(x,y) \qquad \frac{\partial}{\partial y} g(x,y) = -\frac{y}{\sigma^2} g(x,y) \tag{15}$$

The top row of the figure below shows g, $\frac{\partial}{\partial x}g$, $\frac{\partial}{\partial x}g$ and $u * \frac{\partial}{\partial x}g + v * \frac{\partial}{\partial y}g$ where $u = \cos\theta$ and $v = \sin\theta$ with $\theta = 115°$. In other words, the last filter provides an approximated directional derivative in the (u,v) direction. The second row shows the results of convolving an image (from `ascent.jpg`) with these filters.



Computer Exercise 5 *The image used here is* `ascent.jpg`*. Create 5 filters that correspond to the directional derivatives in 5 different angles. Show the filters and the results of applying them to the image.* ♢

```
Some Python commands you may find useful in the exercise above:
np.mgrid, np.exp,np.cos,np.sin, a.sum() (where a is a numpy array),
ndimage.convolve, plt.subplots_adjust
```

2 Optical Flow

2.1 Affine Functions – You don't have to submit your solutions to the exercises or computer exercises in this section – but you will find it useful to read it anyway for the reminder of this HW

Any *linear* map, $f \in \mathcal{F}(\mathbb{R}^n, \mathbb{R}^m)$, can be written as $f : \boldsymbol{x} \mapsto \boldsymbol{Ax}$ for some $\boldsymbol{A} \in \mathbb{R}^{m \times n}$; *i.e.*, $\boldsymbol{A}$ is an m-by-n matrix.

Exercise 8 *A linear map from* $\mathbb{R}^n$ *to* $\mathbb{R}^n$, $f : \boldsymbol{x} \mapsto \boldsymbol{Ax}$, *is invertible if and only if* $\boldsymbol{A}$ *satisfies a condition. What is this condition?* ♢

Exercise 9 *Let* $f : \mathbb{R} \to \mathbb{R}$ *be defined by* $f : x \mapsto ax + b$ *for some* $a, b \in \mathbb{R}$. *In other words,* f *describes a straight line. Show that if* $b \neq 0$ *then* f *is a nonlinear function. Remark: people often refer to such functions as linear, and occasionally we may also do it ourselves, but, strictly speaking, this is incorrect.* ♢

Definition 1 (Affine map) *A map,* $f : \mathbb{R}^n \to \mathbb{R}^m$, *is called affine if it can be written as* $f : \boldsymbol{x} \mapsto \boldsymbol{Ax} + \boldsymbol{b}$ *for some* $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ *and* $\boldsymbol{b} \in \mathbb{R}^m$. ♢

Thus, $f : x \mapsto ax + b$ from the exercise above is an affine function (and linear $\iff b = 0$). Note the affine maps generalize linear maps (*i.e.*, every linear map is affine, the converse is false).

Exercise 10 *Under what condition(s) an affine map, $f : \boldsymbol{x} \mapsto \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$, is invertible?* $\diamondsuit$

Exercise 11 (Composition of affine maps) *Let $h = g \circ f$ where $f : \mathbb{R}^n \to \mathbb{R}^m$ and $g : \mathbb{R}^m \to \mathbb{R}^k$ are two affine maps:*

$$\begin{aligned} f &: \boldsymbol{x} \mapsto \boldsymbol{A}_1\boldsymbol{x} + \boldsymbol{b}_1 && \boldsymbol{A}_1 \in \mathbb{R}^{m\times n} && \boldsymbol{b}_1 \in \mathbb{R}^m\,; \\ g &: \boldsymbol{x} \mapsto \boldsymbol{A}_2\boldsymbol{x} + \boldsymbol{b}_2 && \boldsymbol{A}_2 \in \mathbb{R}^{k\times m} && \boldsymbol{b}_2 \in \mathbb{R}^k\,. \end{aligned} \tag{16}$$

Show that $h : \mathbb{R}^n \to \mathbb{R}^k$ is affine. $\diamondsuit$

Exercise 12 (Composition of invertible affine maps) *Let $h = g \circ f$ where $f : \mathbb{R}^n \to \mathbb{R}^n$ and $g : \mathbb{R}^n \to \mathbb{R}^n$ are two invertible affine maps. Show that h is also an invertible affine map.* $\diamondsuit$

Exercise 13 (inverse of an invertible affine map) . *Let $f : \mathbb{R}^n \to \mathbb{R}^n$, $\boldsymbol{x} \mapsto \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$ be an invertible affine map. Find f^{-1} and show it is affine too.* $\diamondsuit$

2.2 Image Warping – You don't have to submit your solutions to the exercises or computer exercises in this section – but you will find it useful to read it anyway for the reminder of this HW

Suppose we have a nominal optical flow, $\boldsymbol{u}(\cdot) = (u(\cdot), v(\cdot))$; *i.e.* at every pixel $\boldsymbol{x} = (x, y)$, suppose we know the value of $\boldsymbol{u}(\boldsymbol{x}) \in \mathbb{R}^2$. Given an image I, we would like create (on a computer) the so-called warped image,

$$I_{\text{warpped}}(x', y') = (I(x + u(\boldsymbol{x}), y + v(\boldsymbol{x})) \qquad x' = x + u(\boldsymbol{x}) \qquad y' = y + v(\boldsymbol{x})\,. \tag{17}$$

The equation above assumes a continuous setting; we, however, care about digital images. Usually, even if x and y are integers, $x+u(x, y)$ and/or $y+v(x, y)$ are not. On a computer, when we create the image that contains the values of I_{warpped}, we need to define the values of that image in integral locations. The solution involves an interpolation, *e.g.*, bilinear.

Example 1 (bilinear interpolation) *Let f be a function from $\mathbb{R}^2$ to $\mathbb{R}$. Suppose we know the values of f at integral locations, and want to know find $f(x, y)$ where $x \notin \mathbb{Z}$ and/or $y \notin \mathbb{Z}$. Bilinear interpolation approximates $f(x, y)$ from the values of f at the corners of a square containing (x, y). If we choose a coordinate system in which the four points where f is known are (0, 0), (0, 1), (1, 0), and (1, 1), then*

$$f(x, y) \approx f(0, 0)(1 - x)(1 - y) + f(1, 0)x(1 - y) + f(0, 1)(1 - x)y + f(1, 1)xy\,. \tag{18}$$

$\diamondsuit$

However, in image warping, the setting is somewhat reversed. Consider images with N_{rows} rows and N_{cols} columns. The values we know,

$$I(x + u(\boldsymbol{x}), y + v(\boldsymbol{x})) \qquad \forall x \in \{1, \ldots, N_{\text{rows}}\} \text{ and } \forall y \in \{1, \ldots, N_{\text{cols}}\}\,, \tag{19}$$

usually fall in non-integral locations, while the values we want to find,

$$I_{\text{warpped}}(\tilde{x},\tilde{y}) \qquad \forall \tilde{x} \in \{1,\dots,N_{\text{rows}}\} \text{ and } \forall \tilde{y} \in \{1,\dots,N_{\text{cols}}\}\,, \tag{20}$$

are placed in integral locations.

The easiest and standard way to do warping is, for every pixel $(\tilde{x},\tilde{y})$ in I_{warpped}, to see "where it came from" and then interpolate between the values of the pixes in the original image that are around that location. In other words, let $T : \mathbb{R}^2 \to \mathbb{R}^2$ be defined by $T : (x,y) \mapsto (x+u(x,y), y+v(x,y)$. Thus, $I_{\text{warpped}} = I \circ T$. We define the value of $I_{\text{warpped}}(\tilde{x},\tilde{y})$ as

$$(I \circ T^{-1})(\tilde{x},\tilde{y}) \triangleq I(T^{-1}(\tilde{\boldsymbol{x}}))\,,$$

where $\tilde{\boldsymbol{x}} = (\tilde{x},\tilde{y})$. While $(T^{-1}(\tilde{\boldsymbol{x}}))$ is usually not an integral location, we are now back in the setting of standard interpolation so we can interpolate between the values of I in the integral locations around $T^{-1}(\tilde{x},\tilde{y})$. One catch is that T may not be invertible. E.g., the flow may send both pixels to the same location. One "solution" is to ignore this problem and, using the negative flow, just set $T^{-1}(\tilde{x},\tilde{y}) \approx (\tilde{x} - u(\tilde{x},\tilde{y}), \tilde{y} - v(\tilde{x},\tilde{y}))$.

Remark 1 *Alternatively, instead of warping I_1 toward I_2, we can warp I_2 toward I_1; in other words, use the known values of T as the values of $(T^{-1})^{-1}$ which are required in order to create $I_2 \circ T^{-1}$.* ♢

Before we continue, familiarize yourself first with numpy's `mgrid` (the Matlab equivalent is meshgrid); *e.g.*:

```
In [1]: from numpy import mgrid
In [2]: yy,xx=mgrid[0:3,0:4]
In [3]: xx
Out[3]:
array([[0, 1, 2, 3],
       [0, 1, 2, 3],
       [0, 1, 2, 3]])

In [4]: yy
Out[4]: array([[0, 0, 0, 0],
       [1, 1, 1, 1],
       [2, 2, 2, 2]])
```

The following is from OpenCV Doc for cv2.remap: (with some mild edits and omissions:

```
cv2.remap(src, map$_1$, map$_2$, interpolation,dst,...)
src: Source image (e.g., $I_1$)
dst: Destination image (e.g., $I_1 \circ T$)
map$_1$: $x$ values (e.g., $xx - u$)
map$_2$: $y$ values (e.g., $yy - v$)
interpolation: Interpolation method
```

There are many standard implementations for image warping. For example, the function cv2.remap transforms the source image using the specified map: $\texttt{dst}[y,x] = \texttt{dst}(x,y) = \texttt{src}(\texttt{map}_1(x,y), \texttt{map}_2(x,y))$ where values of pixels with non-integer coordinates are computed using one of the available interpolation methods. The convention used above is that $I(x, y)$ (round parentheses where x is the first coordinate) is used for "math" and $I[y, x]$ (square brackets where y is the first coordinate) for "code".

Computer Exercise 6 *Write your own function for bilinear interpolation.* $\diamondsuit$

Computer Exercise 7 *Get img1, u and v from* `imgs_for_optical_flow.mat`*. Use your bilinear interpolation function to warp img1 using the* (u, v) *flow. Show your result along with the result obtained by using a standard implementation of image warping;* e.g.*, OpenCV's remap function.* $\diamondsuit$

Of course, if the flow is invertible and the inverse is known or has a closed-form parametric form, we had better use it explicitly. For example, this is the case for affine flow, provided it is also invertible.

Computer Exercise 8 *Get img1 from* `imgs_for_optical_flow.mat`*. Use your bilinear interpolation function to warp img1 using the (invertible) affine flow:*

$$\boldsymbol{u}(\boldsymbol{x}) = \begin{bmatrix} 0.5 & 0.2 & 0 \\ 0 & 0.5 & 8 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad \diamondsuit$$

Show your result along with the result obtained by using a standard implementation of affine image warping; e.g.*, OpenCV's cv2.warpAffine function (note that this function takes the inverse map as its input).*

From now on, you are free to use standard implementations of image warping (which are probably faster then yours, and usually also offer additional and better methods for interpolation; *e.g.*, cubic splines). So you do not have to use your own implementation in the following exercises/assignments.

2.3 Computing the Spatial Derivatives Required for Optical-Flow Estimation

Computer Exercise 9 *Get img1 from* `imgs_for_optical_flow.mat`*. Using derivative filters, compute, for I=img1, the following images:*

$$I_x = \frac{\partial}{\partial x} I = I * h_x, \quad I_y = \frac{\partial}{\partial y} I = I * h_y,$$
$$I_{xx} = \frac{\partial^2}{\partial x^2} I = I * h_{xx}, \quad I_{yy} = \frac{\partial^2}{\partial y^2} I = I * h_{yy} \, .$$

where h_x is a filter which approximates derivative in the x direction, h_y is a filter which approximates derivative in the y direction, h_{xx} is a filter which approximates second derivative in the x direction, and h_{yy} is a filter which approximates second derivative in the y direction. You may want to blur I a little by convolving it with g_0, a Gaussian of small standard deviation, σ_0, before taking these derivatives, or, equivalently, convolve I with the derivatives of g_0. $\diamondsuit$

```
A Python command you may find useful for the exercise above:
cv2.getDerivKernels
```

2.4 Optical Flow

The gradient-constraint equation for gray-scale 2D images is

$$\boxed{I_x(\boldsymbol{x},t)u(\boldsymbol{x}) + I_y(\boldsymbol{x},t)v(\boldsymbol{x}) + I_t(\boldsymbol{x},t) = 0}\,. \tag{21}$$

Exercise 14 (optical flow for RGB images) *Consider the RGB case;* i.e., $I(\boldsymbol{x},t) = \begin{bmatrix} R(\boldsymbol{x},t) & G(\boldsymbol{x},t) & B(\boldsymbol{x},t) \end{bmatrix}^T$. *Suggest an analogous gradient-constraint equation for optical flow between RGB images. Explain under what conditions the new equation will have: 1) infinitely-many solutions (as we had in the 2D case – recall,* e.g., *our discussion on normal flow); 2) a unique solution; 3) no solution.* $\diamondsuit$

In the 2D case, for gray-scale images, a simple per-pixel cost function that penalizes deviations from the gradient-constraint equation is

$$\varepsilon^2(u(\boldsymbol{x}), v(\boldsymbol{x})) = (I_x(\boldsymbol{x},t)u(\boldsymbol{x}) + I_y(\boldsymbol{x},t)v(\boldsymbol{x}) + I_t(\boldsymbol{x},t))^2\,. \tag{22}$$

Exercise 15 *Suggest an analogous cost function for the RGB case. Is your suggested function still convex? Explain.* $\diamondsuit$

Exercise 16 (3D optical flow) *Derive the gradient-constraint equation for optical flow between volumetric images (*e.g., *CT images);* i.e., *the image "sequence" is now $I(x,y,z,t)$.* $\diamondsuit$

Fact 1 *Let X be a random variable (Gaussian or not) with a p.d.f. $p_X(x)$. Let a and b be two real scalars such that $a \neq 0$. Let $Y = aX + b$. Then:*

(i) If $E(X) = \mu$, then $E(Y) = a\mu + b$. If $\mathrm{VAR}(X) = \sigma^2$, then $\mathrm{VAR}(Y) = a^2\sigma^2$.

(ii) If, in addition, X is a Gaussian random variable then Y is also a Gaussian random variable. Thus, $X \sim \mathcal{N}(\mu, \sigma^2)$ and $Y \sim \mathcal{N}(a\mu + b, a^2\sigma^2)$. $\diamondsuit$

Fact 2 *Let X be a random variable (Gaussian or not) with a p.d.f. $p_X(x)$. Let a and b be two real scalars such that $a \neq 0$. Let $Y = aX + b$. Then, $p_Y(y)$, the p.d.f. of y, is given by*

$$p_Y(y) = \left|\frac{d}{dy}\frac{y-b}{a}\right| p_X\left(\frac{y-b}{a}\right) = \left|\frac{1}{a}\right| p_X\left(\frac{y-b}{a}\right)\,. \tag{23}$$

$\diamondsuit$

Exercise 17 (optical flow and a Gaussian noise) *(i) Let $X \sim \mathcal{N}(\mu, \sigma^2)$ and let $Y = aX + b$ where $a > 0$ and $b \in \mathbb{R}$. Show that $p_Y(y) = \left|\frac{1}{a}\right| p_X\left(\frac{y-b}{a}\right)$ is consistent with the fact that $Y \sim \mathcal{N}(a\mu + b, a^2\sigma^2)$.*

*(ii) Let Y be a random variable such that $y = \mu + \varepsilon$ where μ is constant and ε is a realization (*i.e., *a sample) from a zero-mean Gaussian noise with variance σ^2. How is Y distributed? Write down the expression for $p_Y(y)$.*

(iii) Let us write the gradient-constraint equation as

$$\mu \triangleq I_x(\boldsymbol{x},t)u_x + I_y(\boldsymbol{x},t)u_y = -I_t(\boldsymbol{x},t)\,.$$

Consider the LHS as a known deterministic scalar. For the sake of the current discussion, it does not matter that this scalar, μ, depends on the two deterministic (but unknown) parameters, $u(\boldsymbol{x},t)$ and $v(\boldsymbol{x},t)$. Since we do not expect the equation to hold perfectly, let us treat the RHS, $-I_t(\boldsymbol{x},t)$, as a noisy observation of μ, under the assumption of an additive Gaussian noise: $-I_t(\boldsymbol{x},t) = I_x(\boldsymbol{x},t)u_x + I_y(\boldsymbol{x},t)u_y + \varepsilon = \mu + \varepsilon$ where

$$p(\varepsilon;\sigma^2) = \mathcal{N}(\varepsilon;0,\sigma^2) \triangleq \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{1}{2}\frac{\varepsilon^2}{\sigma^2}\right) \qquad \sigma > 0\,.$$

Under these assumptions, how is $-I_t(\boldsymbol{x},t)$ distributed? $\diamondsuit$

For the following computer exercises, get img1, img2, img3, img4, img5, and img6 from `imgs_for_optical_flow.mat`. We will consider five pairs of frames:

pair #1: I_1=img1 and I_2=img2;

pair #2: I_1=img1 and I_3=img3;

pair #3: I_1=img1 and I_4=img4;

pair #4: I_1=img1 and I_5=img5;

pair #5: I_1=img1 and I_6=img6.

You already computed the spatial derivatives of I_1.

Computer Exercise 10 *For each one of the pairs, compute and display the (approximated) temporal derivatives by simply subtracting I_1 from each one of the other images.* $\diamondsuit$

In what follows we assume images of N_{rows} rows and N_{cols} columns.

2.5 Horn-and-Schunck Optical Flow

The Horn-and-Schunck cost function (in its discrete from – which is the only form of it we discussed) is:

$$\begin{aligned}
E(u,v,I_1,I_2) &= E_{\text{data}}(u,v,I_1,I_2) + \lambda E_{\text{smoothness}}(u,v)\\
&= \left(\sum_{i,j}(I_x(i,j,t)u_{i,j} + I_y(i,j,t)v_{i,j} + I_t(i,j,t))^2\right)\\
&+ \lambda\sum_{i,j}\left[(u_{i,j}-u_{i+1,j})^2 + (u_{i,j}-u_{i,j+1})^2 + (v_{i,j}-v_{i+1,j})^2 + (v_{i,j}-v_{i,j+1})^2\right]\\
&= \sum_{i,j}(I_x(i,j,t)u_{i,j} + I_y(i,j,t)v_{i,j} + I_t(i,j,t))^2\\
&+ \lambda\left[(u_{i,j}-u_{i+1,j})^2 + (u_{i,j}-u_{i,j+1})^2 + (v_{i,j}-v_{i+1,j})^2 + (v_{i,j}-v_{i,j+1})^2\right]
\end{aligned} \tag{24}$$

where $\lambda > 0$ controls the tradeoff, and we use (i,j) to denote pixel locations in order to emphasize the discrete nature of the domain. For computing the gradient, the derivatives are (for a pixel not on the border of the image):

$$\frac{\partial E_{\text{data}}}{\partial u_{i,j}} = 2(I_x(i,j,t)u_{i,j} + I_y(i,j,t)v_{i,j} + I_t(i,j,t))I_x(i,j,t) \tag{25}$$

$$\frac{\partial E_{\text{data}}}{\partial v_{i,j}} = 2(I_x(i,j,t)u_{i,j} + I_y(i,j,t)v_{i,j} + I_t(i,j,t))I_y(i,j,t) \tag{26}$$

$$\frac{\partial E_{\text{smoothness}}}{\partial u_{i,j}} = 2\lambda(4u_{i,j} - (u_{i+1,j} + u_{i,j+1} + u_{i-1,j} + u_{i,j-1})) \tag{27}$$

$$\frac{\partial E_{\text{smoothness}}}{\partial v_{i,j}} = 2\lambda(4v_{i,j} - (v_{i+1,j} + v_{i,j+1} + v_{i-1,j} + v_{i,j-1})) \tag{28}$$

Exercise 18 *Complete the details: assuming one-based indexing, write the gradient terms for a pixel on the boundary;* i.e., *where $i = 1$ or $j = 1$ or $i = N_{rows}$ or $j = N_{cols}$; do not forget to include corner cases: $(i = 1, j = 1)$, $(i = 1, j = N_{cols})$, $(i = N_{rows}, j = 1)$, and $(i = N_{rows}, j = N_{cols})$.*

Exercise 19 *(i) What $(u(\cdot), v(\cdot))$ will minimize E_{data}? Note that at the minimizer, $\frac{\partial E_{data}}{\partial u_{i,j}}$ and $\frac{\partial E_{data}}{\partial v_{i,j}}$ are zero.*

(ii) What $(u(\cdot), v(\cdot))$ will minimize $E_{smoothness}$? Again, note that at a minimizer, $\frac{\partial E_{smoothness}}{\partial u_{i,j}}$ and $\frac{\partial E_{smoothness}}{\partial v_{i,j}}$ are zero. Show that the minimizer is not unique. ◊

To make the notation more compact, and also in preparation for our discussion on Markov Random Fields later on, let a single index, s (short for site), denote a generic (i, j) pair, and, for $s = (i, j)$, let $s \sim s'$ denote the fact that s' is a neighbor of s according to 4-connectivity; *i.e.*,

$$s' \in \{(i+1, j), (i-1, j), (i, j+1), (i, j-1)\}\,. \tag{29}$$

We now rewrite the cost function, while also dropping the dependency on t:

$$E(u, v, I_1, I_2) = \sum_s \left[(I_x(s)u(s) + I_y(s)v(s) + I_t(s))^2 + \lambda \sum_{s':s\sim s'} \left[(u(s) - u(s'))^2 + (v(s) - v(s'))^2 \right] \right] \tag{30}$$

where the summation $\sum_{s':s'\sim s}$ is done over the (usually 4) neighbors of s, and the double counting in that summation is accounted for by adjusting λ by a factor of 2.

At every pixel s, not on the border of the image:

$$\frac{\partial}{\partial u(s)} E = 2(I_x^2(s)u(s) + I_x(s)I_y(s)v(s) + I_x(s)I_t(s)) + 2\lambda \left(2 \sum_{s':s\sim s'} (u(s) - u(s')) \right) \tag{31}$$

$$\frac{\partial}{\partial v(s)} E = 2(I_y(s)I_x(s)u(s) + I_y^2(s)v(s) + I_y(s)I_t(s)) + 2\lambda \left(2 \sum_{s':s\sim s'} (v(s) - v(s')) \right) \tag{32}$$

which is of course similar to what we had before.

To get critical points, we set the gradient to zero (so can ignore the 2) and obtain the **normal equations**:

$$I_x^2(s)u(s) + I_x(s)I_y(s)v(s) + I_x(s)I_t(s) + 2\lambda \sum_{s':s\sim s'} (u(s) - u(s')) = 0 \tag{33}$$

$$I_y(s)I_x(s)u(s) + I_y^2(s)v(s) + I_y(s)I_t(s) + 2\lambda \sum_{s':s\sim s'} (v(s) - v(s')) = 0\,. \tag{34}$$

Exercise 20 *Complete the details: assuming one-based indexing, write the gradient terms for a pixel s on the boundary.* ♢

This leads to a very large, but also very sparse, linear system of the form $\boldsymbol{A\xi} = \boldsymbol{b}$. If $N = N_{\text{cols}} \times N_{\text{rows}}$ is the number of pixels, then $\boldsymbol{A}$ is $2N \times 2N$, while $\boldsymbol{\xi}$ and $\boldsymbol{b}$ are $2N \times 1$. For example, suppose we use the following ordering:

$$\boldsymbol{\xi} = \begin{bmatrix} u(1,1) & v(1,1) & u(1,2) & v(1,2) & \dots & u(2,1) & v(2,1) & u(2,2) & v(2,2) & \dots & u(N_{\text{rows}},1) & v(N_{\text{rows}},1) & u(N_{\text{rows}},2) & v(N_{\text{rows}},2) & \dots \end{bmatrix}^T . \tag{35}$$

Then (using 1-based indexing), for a pixel $s = (i, j)$ not on the border of the image, with $i_s = (i-1) \times N_{\text{cols}} + j$ denoting the linear index of s, the row of $\boldsymbol{A}$ that corresponds to $u(s)$, *i.e.*, row #$(2i_s - 1)$, is

$$\begin{bmatrix} \dots 0 & \underbrace{-2\lambda}_{2i_s-2N_{\text{cols}}-1} & \underbrace{0}_{2i_s-2N_{\text{cols}}} & \dots & \underbrace{-2\lambda}_{2i_s-3} & \underbrace{0}_{2i_s-2} & \underbrace{(I_x^2(s)+8\lambda)}_{2i_s-1} & \underbrace{I_x(s)I_y(s)}_{2i_s} & \underbrace{-2\lambda}_{2i_s+1} & \underbrace{0}_{2i_s+2} & \dots & \underbrace{-2\lambda}_{2i_s+2N_{\text{cols}}-1} & \underbrace{0}_{2i_s+2N_{\text{cols}}} & \dots \end{bmatrix} \tag{36}$$

while the row of $\boldsymbol{A}$ that corresponds to $v(s)$, *i.e.*, row #$(2i_s)$, is

$$\begin{bmatrix} \dots 0 & \underbrace{0}_{2i_s-2N_{\text{cols}}-1} & \underbrace{-2\lambda}_{2i_s-2N_{\text{cols}}} & \dots & \underbrace{0}_{2i_s-3} & \underbrace{-2\lambda}_{2i_s-2} & \underbrace{I_y(s)I_y(s)}_{2i_s-1} & \underbrace{(I_y^2(s)+8\lambda)}_{2i_s} & \underbrace{0}_{2i_s+1} & \underbrace{-2\lambda}_{2i_s+2} & \dots & \underbrace{0}_{2i_s+2N_{\text{cols}}-1} & \underbrace{-2\lambda}_{2i_s+2N_{\text{cols}}} & \dots \end{bmatrix} \tag{37}$$

where all the other entries of these two rows equal to zero.

Exercise 21 *Complete the details: assuming one-based indexing, write the rows of $\boldsymbol{A}$ that correspond to pixels on the border of the image;* i.e., *where $i = 1$ or $j = 1$ or $i = N_{rows}$ or $j = N_{cols}$; do not forget to include corner cases: ($i = 1, j = 1$), ($i = 1, j = N_{cols}$), ($i = N_{rows}, j = 1$), and ($i = N_{rows}, j = N_{cols}$).* ♢

Finally:

$$\boldsymbol{b} = \begin{bmatrix} \dots & \underbrace{-I_x(s)I_t(s)}_{2i_s-1} & \underbrace{-I_y(s)I_t(s)}_{2i_s} & \dots \end{bmatrix}^T . \tag{38}$$

Computer Exercise 11 *For each one of the five frame pairs, construct $\boldsymbol{A}$ and $\boldsymbol{b}$ (use a sparse matrix for $\boldsymbol{A}$) and, using a sparse linear solver, solve for $\boldsymbol{\xi}$. Display the estimated flows, and the results of warping I_1 using these flows. Repeat this with several different values of λ and visually explore the effect of λ on the resulting flows.* ***Optional:*** *do the robust version using Geman-McClure function and Iterative Reweighted Least Squares.* ♢

```
Python commands you may find useful for the exercise above:
scipy.linalg.lstsq; scipy.sparse.linalg.lsqr.
```

Computer Exercise 12 *Repeat the exercise above, but do this in a coarse-to-fine manner. After estimating the flow in each level, and before propagating it to the next finer level, you may want to apply median filtering to the estimated flow.* ***Optional:*** *do the robust version using Geman-McClure function and Iterative Reweighted Least Squares.* ◊

2.6 Lucas-and-Kanade Optical Flow

Let $\boldsymbol{x}_i = (x_i, y_i)$ be in some neighborhood of $\boldsymbol{x} = (x, y)$. An affine flow model has the following form:

$$\begin{aligned}\begin{bmatrix} u(\boldsymbol{x}_i) \\ v(\boldsymbol{x}_i) \end{bmatrix} &= \begin{bmatrix} \theta_1 & \theta_2 & \theta_3 \\ \theta_4 & \theta_5 & \theta_6 \end{bmatrix} \begin{bmatrix} x_i - x \\ y_i - y \\ 1 \end{bmatrix} \\ &= \underbrace{\begin{bmatrix} x_i - x & y_i - y & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_i - x & y_i - y & 1 \end{bmatrix}}_{A(\boldsymbol{x},\boldsymbol{x}_i)\triangleq} \underbrace{\begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \\ \theta_6 \end{bmatrix}}_{\boldsymbol{\theta}\triangleq}\end{aligned} \tag{39}$$

Together with the gradient-constraint equation, we get:

$$\nabla_{\boldsymbol{x}} I(\boldsymbol{x}_i, t)\boldsymbol{A}(\boldsymbol{x}, \boldsymbol{x}_i)\boldsymbol{\theta} + I_t(\boldsymbol{x}_i, t) = 0 \tag{40}$$

The Lucas-Kanade solution for affine flow is then:

$$\hat{\boldsymbol{\theta}}_{\text{LK}}(\boldsymbol{x}) = \underbrace{\boldsymbol{M}^{-1}(\boldsymbol{x})}_{6\times 6}\underbrace{\boldsymbol{b}(\boldsymbol{x})}_{6\times 1} \tag{41}$$

where

$$\boldsymbol{M}(\boldsymbol{x}) = \sum_i g(\boldsymbol{x}, \boldsymbol{x}_i)\boldsymbol{A}(\boldsymbol{x}, \boldsymbol{x}_i)^T \nabla_{\boldsymbol{x}} I(\boldsymbol{x}_i, t)^T \nabla_{\boldsymbol{x}} I(\boldsymbol{x}_i, t)\boldsymbol{A}(\boldsymbol{x}, \boldsymbol{x}_i) \tag{42}$$

$$\boldsymbol{b}(\boldsymbol{x}) = -\sum_i g(\boldsymbol{x}, \boldsymbol{x}_i)\boldsymbol{A}(\boldsymbol{x}, \boldsymbol{x}_i)^T \nabla_{\boldsymbol{x}} I(\boldsymbol{x}_i, t)^T I_t(\boldsymbol{x}_i, t) \tag{43}$$

and g is a weight function.

We can use this kind of method in two different ways (yielding different results).

1. Solve this system just once, for the entire image, using a weight function whose support is the entire image (where typically the weights are all 1). In which case, we may as well take $\boldsymbol{x}$ to be the center of the image. This results in a single affine flow model for the entire image (since this is really a global model, I prefer referring to the overall LK approach as patch-based – where in this particular case the patch is the entire image – rather than referring to it as a local approach).

2. Solve such a system for each pixel $\boldsymbol{x}$, where typically the support of the weight function is limited to a small neighborhood (*e.g.*, 3×3, 5×5, etc.). In which case, we typically let $g(\boldsymbol{x}, \boldsymbol{x}_i)$ decay with the distance between $\boldsymbol{x}$ and $\boldsymbol{x}_i$; *e.g.*, in a Gaussian way.

We we will refer to the first option as globally-affine flow and to the second as locally-affine flow.

Computer Exercise 13 *Implement the iterative LK algorithm for a locally-affine flow, and apply it to each one of the five frame pairs. After estimating the flow in each level, and before propagating it to the next finer level, you may want to apply median filtering to the estimated flow. Display the estimated flows, and the results of warping I_1 using these flows.* ***Optional:*** *do the iterative coarse-to-fine version.* ***Optional:*** *do the robust version using the Geman-McClure function and Iterative Reweighted Least Squares.* $\diamondsuit$

Computer Exercise 14 *Implement the iterative LK algorithm for a globally-affine flow. After estimating the flow in each level, and before propagating it to the next finer level, you may want to apply median filtering to the estimated flow. Display the estimated flows, and the results of warping I_1 using that flow.* ***Optional:*** *do the iterative coarse-to-fine version.* ***Optional:*** *do the robust version using the Geman-McClure function and Iterative Reweighted Least Squares.* $\diamondsuit$