Questions may continue on the back. Please write clearly. What I cannot read, I will not grade.

Consider a camera that moves in front of a static scene for which the Brightness Change Constraint Equation (BCCE)

\[ \frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v + \frac{\partial I}{\partial t} = 0 \]

is known to hold. The three derivatives and the two unknown optical flow components \(u\) and \(v\) in this equation are all functions of the coordinates \(x, y\) of the pixels in the image. The \(x\) axis is horizontal in the image and points to the right. The \(y\) axis is vertical in the image and points up. The components \(u\) and \(v\) of the optical flow are along the \(x\) and \(y\) axes, respectively.

We say that the camera moves sideways if the vector that describes the motion of the camera in the world is parallel to the rows (scan lines) of the camera’s image sensor. For a camera that moves sideways, all image points move parallel to the \(x\) axis, so that \(v(x, y) = 0\) for all \(x, y\) and the BCCE consequently reduces to the following:

\[ \frac{\partial I}{\partial x} u + \frac{\partial I}{\partial t} = 0 . \]

(a) In one sentence, why is solving for optical flow fundamentally easier if the camera is known to be moving sideways in a static world?

(b) Can the aperture problem arise at all for a camera moving sideways in a static world?

(c) The Horn-Schunck energy function for optical flow for a camera moving sideways in a static world reduces to the following:

\[ E = \sum_x \sum_y \left( \frac{\partial I}{\partial x} u + \frac{\partial I}{\partial t} \right)^2 + \lambda \sum_x \sum_y \left\{ [u(x+1, y) - u(x, y)]^2 + [u(x, y+1) - u(x, y)]^2 \right\} . \]

Use \( \lambda = \text{lambda0} = 150 \) in all your work that requires a scalar regularization constant, both in this question and elsewhere. Modify the code provided for this assignment on the homework web page so that a call of the following form

\[ \text{flow} = \text{estimate}\_\text{flow}\_hs(i0, i1, \text{'lambda', lambda0}, \text{'horizontal', true}); \]

will minimize the energy as defined above. The call

\[ \text{flow} = \text{estimate}\_\text{flow}\_hs(i0, i1, \text{'lambda', lambda0}); \]

should still minimize the full energy

\[ E = \sum_x \sum_y \left( \frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v + \frac{\partial I}{\partial t} \right)^2 + \lambda \sum_x \sum_y \left\{ [u(x+1, y) - u(x, y)]^2 + [u(x, y+1) - u(x, y)]^2 + [v(x+1, y) - v(x, y)]^2 + [v(x, y+1) - v(x, y)]^2 \right\} . \]

If you closely follow the hints given in the Programming Notes later on, you can answer this question without understanding much about the details of the code provided. Explain your rationale, and hand in only the MATLAB code you add or change.

(d) Run both versions (that is, with parameter horizontal set to false and then to true) of your code (both times with regularization constant lambda0 = 150), and use the provided function displayFlows to display computed and true flow. Make sure that the first argument to estimate_flow_hs is from the provided image i0.png, and the second from i1.png.

The true flow can be read from the provided image flow01.png. Both input images and true flow are adapted from images in the BOWLING 2 dataset at http://vision.middlebury.edu/stereo/data/scenes2006/. True flow was computed by projecting a highly structured pattern of light onto the scene, and using that information to reconstruct the three-dimensional geometry of the scene. Since the sideways displacement between camera positions was measured as well, this knowledge allows determining where every point in the world ends up in each of the two images and, as a consequence, the flow for every pixel. Note that flow is given as an integer pixel map (so it can be stored in a regular image file). Because of this, the true flow is only known up to pixel resolution.\(^1\)

Call the provided function displayFlows with calls of this form:

\(^1\)In the original dataset, resolution is higher than in this assignment. Images were made smaller so you can run the code in less time and on less memory.
where the variable names have obvious meanings. The last argument numbers the group of figures displayed. Make sure you do not confuse the group numbers.

The function `displayFlows` makes PDF files for all the pictures for you. If you do not like PDF, just modify the `print` command in `displayFlows` to output what you want. The display function does not print out the title you see at the top of the figure windows. Make sure you label figures in your work, and report the corresponding RMSE values. You need not hand in the mesh plots (files with the word 'Mesh' in them). These are made for your perusal. If you drag the mouse over the mesh results, the plots rotate.

You should not expect large improvements in going from 2D to 1D flow, because the results produced by Horn-Schunck are already reasonably good, and room for improvement is tight. An improvement of 2 percent in RMSE is good. If your RMSE for the 1D version is worse than for the 2D version, then there is a problem somewhere.

(e) Why should you not expect improvements to below about 0.5 pixels RMSE?

(f) Is there some way in which the flow values computed by Horn-Schunck (either version) are better than those in the "true flow" values in `flow01.png`? Can you see this in some of your results? If so, where? Explain carefully. You may want to look at the mesh plots.

(g) Providing true flow in file `flow01.png` required a very significant amount of work by the Middlebury people. Also, if true flow is known, then there is no point in computing it from the images again – other than when testing an algorithm. So in general situations it is useful to have a method for checking how consistent the flow \( u \) computed by Horn and Schunck is with the data. It would be tempting to compute an image showing the absolute value of the data term,

\[
D = \frac{\partial I}{\partial x} u + \frac{\partial I}{\partial t}.
\]

Why is this not a good idea? Explain carefully.

(h) Write a function

```matlab
function D = residual(I, J, u)

    % takes as inputs the two images I and J that are given to a flow computation, and the output flow u. The function residual
    % warps the second image J by the computed flow to make a new image K that is more similar to the first image I. That is,
    %
    % \( K(x) = J(x + u) \).

    since \( u \) is not integer-valued, use bilinear interpolation to compute K. The MATLAB function `interp2` with option 'linear' does bilinear interpolation. See the MATLAB product help for how to use `interp2`.

    The function `residual` then outputs an image \( D \) with the absolute value of \( K - I \).

    Hand in your code for residual and a printout of the residual image for the results obtained with the one-dimensional (that is, with horizontal set to true) version of Horn-Schunck. To save ink, if `res` is the output from `residual`, hand in a print of the image `data.jpg` produced as follows:

    ```matlab
    mx = max(res(:));
    res = uint8(round((mx - res) / mx * 255));
    imwrite(res, 'data.jpg');
    ```

(i) In your previous answer, you computed the data residual for the output from Horn-Schunk. The function `grad` provided with this assignment computes the gradient of the image passed as argument. Use that function to write an approximate estimate of the smoothness residual image

\[
\sqrt{[u(x + 1,y) - u(x,y)]^2 + [u(x,y + 1) - u(x,y)]^2}.
\]

Hand in your code and the resulting image, printed with the same technique used for the data residual.
(j) How could you use the smoothness residual you computed in the previous question to improve the results of Horn-Schunck along occluding boundaries, using also the fact that Horn-Schunk, as modified for this assignment, can take a space-varying regularization term $\lambda(x, y)$? See the programming notes for an explanation of the latter statement. Explain your idea carefully, but without implementing it, and say why it should work. There are many possible valid answers to this question. If you implement your solution (next question), then it is OK to explain details as you describe the implementation. If not, provide detail here. Specifically, how would you compute $\lambda(x, y)$, exactly? If you do not answer the next question, make sure that you pay close attention to your answer to this one. Implementing, while optional, helps being precise and correct.

(k) (10 points extra credit) Implement your suggestion, hand in your code, show the results, and analyze tradeoffs. You should not expect improvements by more than 2-4 percent in RMSE relative to the one-dimensional version of Horn-Schunck.

Programming Notes

The code for optical flow provided for this assignment is a modified version of the code written by Deqing Sun and available at http://www.cs.brown.edu/˜dqsun/research/software.html. That code allows specifying various optimization parameters with syntax of the following form:

```matlab
flow = estimate_flow_hs(i0, i1, ParamName1, ParamValue1, ...);
```

For instance:

```matlab
flow = estimate_flow_hs(i0, i1, 'lambda', lambda);
```

Multiple name/value parameter pairs are allowed, in any order.

The three main modifications made to Sun’s code for this assignment are as follows:

- The flow computed by the Horn-Schunck algorithm is median-filtered. This is because some of the modifications you will study in this assignment cause isolated outliers, which median filtering effectively eliminates.

- The parameter lambda, corresponding to the regularization parameter $\lambda$ in the expressions for the energy $E$, can be either a scalar (this was the only option in the original code) or a two-dimensional array of the same size as either input image. [Sun’s code works for both color and gray images, but we only consider gray images in this assignment.] When lambda is an array, the regularization parameter $\lambda$ is not a constant but a function of pixel position, $\lambda(x, y)$. All the equations in Sun’s code have been modified in order to handle this space-varying version of regularization.

- The modified code can handle a parameter name/value pair with name horizontal and value either true or false. The default value is false, so if you do not specify this parameter you get the full, two-dimensional optical flow. When the parameter is either unspecified or set to false, the variable this.horizontal in the function flow_operator defined in file @hs_optical_flow/flow_operator.m will have the value false, and the else clause of the conditional starting at “if this.horizontal” in that function will be executed. The code in the else clause is the original code by Sun. The code in the “if” clause currently has placeholders for four variable assignments, and it is your job to define those four values so that the code minimizes the “sideways” version of $E$ instead of the full, two-dimensional one, as explained next.

To write the code in the “if” clause (question (c) in this assignment), all you need to understand is that the Horn-Schunck code, in either version, repeatedly solves an equation of the form

$$Aw = b \quad \text{where} \quad w = \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} u_1 \\ \vdots \\ u_n \\ v_1 \\ \vdots \\ v_n \end{bmatrix}.$$ 

Here, $n$ is the number of pixels in each of the two input images to `estimate_flow_hs`, the $n$-dimensional vector $u$ collects all the horizontal components of flow, and the $n$-dimensional vector $v$ collects all the vertical components.

If you look just after the comment “% Compute the operator,” you see how the $2n \times 2n$ matrix $A$ and the $2n \times 1$ vector $b$ are built. Specifically, $A$ has the following form:

$$A = \frac{1}{\sigma_D^2} \begin{bmatrix} duu & duv \\ duv & dvv \end{bmatrix} - \frac{1}{\sigma_S^2} M \quad \text{where} \quad M = \begin{bmatrix} F & 0 \\ 0 & F \end{bmatrix}$$
and the two constants $\sigma_D^2$ and $\sigma_S^2$ for the data and smoothness term are irrelevant to our discussion. Each of the four blocks in each of these matrices – including the two zero blocks in $M$ – has size $n \times n$. Similarly,

$$b = \frac{1}{\sigma_S^2} M \, uv(:) - \frac{1}{\sigma_D^2} \begin{bmatrix} \text{Itx}(:) \\ \text{Ity}(:) \end{bmatrix} \quad \text{where} \quad uv(:) = \begin{bmatrix} u \\ v \end{bmatrix}$$

is a $2n \times 1$ vector. While it would be instructive for you to understand what all the pieces mean, this is not necessary to answer question (c). All you need to understand is the following: If you are given a nonsingular linear system of the form

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{12} & A_{22} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix},$$

how can you redefine the blocks $A_{12}, A_{22},$ and $b_2$ so that the following requirements are satisfied?

- The modified system is still nonsingular (that is, the determinant of the modified matrix on the left-hand side is still nonzero), and
- The system is guaranteed to have a solution with $v = 0$, and
- Given that $v = 0$, the modified system and the original system yield the same solution for $u$.

Once you know what the blocks $A_{12}, A_{22},$ and $b_2$ need to look like, it should be straightforward to define $M, \text{Ity}, dvv,$ and $duv$ in the “if” clause to achieve this. **Important: make sure that all your matrices are sparse, or else you will run out of memory. Full vectors are OK.** See the MATLAB help for sparse for building sparse empty matrices, sparse identity matrices, or any other sparse matrix you may need.

### Sample Exam Questions

Answers to the questions below are not due as part of this homework assignment. They will not be graded, and no sample answers will be provided (you will find answers by reading the textbook and notes). They are here only as a way for you to test your own knowledge in computer vision, and to give you an idea of the type of questions that may come up in the final exam. Questions below pertain to sections 4.1.1-4.1.4, 8.1, 8.4-8.5, to the reading notes on mathematical preliminaries, and to this homework assignment.

- Give a formula for the Sum of Squared Differences $SSD(x_0, d)$ between images $I(x)$ and $J(x)$ in a window $W(x_0)$ centered around pixel position $x_0$ in image $I(x)$, and for displacement $d$. You may apply the displacement to either $I$ or $J$.
- What does the Lucas-Kanade algorithm compute, in terms of $SSD(x_0, d)$?
- In what way does the Brightness Change Constraint Equation exhibit the aperture problem?
- What happens to the output of the Horn-Schunck optical flow algorithm if the regularization parameter is set to zero?
- Why does the square power in the regularization term worsen the performance of Horn-Schunk relative to a smaller power?
- Why is a smaller power problematic?
- How are large displacements handled in optical flow algorithms that are based on the Brightness Change Constraint Equation, which assumes small displacements between frames?
- What is the magnitude of the projection of vector $a = (0, 2)$ onto the vector $b = (3, 4)$?
- Would the answer to the previous question change if $a$ and $b$ were switched?
- Write the cross product of vectors $a = (2, 1, 3)$ and $b = (0, 1, 4)$.
  - Is
    $$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
    a rotation matrix? Why or why not?
- Write the inverse of the following transformation from $p$ to $q$:
  $$q = Rp + t.$$