Elec4622 Laboratory Project 3, 2019 T2

David Taubman

August 2, 2019

1 Introduction

This is the third project, to be demonstrated and assessed within your regular scheduled laboratory session in Week 10. The project is worth a nominal 10 marks, but the maximum possible mark is 16 – everything above 10 counts as a bonus mark, although tasks are not specifically designated as bonus or otherwise. Some tasks can be undertaken independently of others, so you do not necessarily need to start from Task 1, or even do Task 1 at all, in order to get marked.

This project aims to help familiarize you with gradient operators, concepts from keypoint detection, and global motion all at once. As preparation, you should review the lecture notes on image analysis and motion estimation. You may also find it useful download and study the materials entitled "Lab 5" on the class web-site, which provide you with an introduction to block-based motion estimation. However, this project itself involves only global motion estimation.

2 Global Motion Estimation

In this project, you will be estimating a global translational motion vector $\mathbf{v}$ from local motion estimates $\mathbf{v_k}$ at a collection of keypoints $\mathbf{k}$. The estimation process involves two images (video frames) denoted here as $x[\mathbf{n}]$ and $y[\mathbf{n}]$, where $x[\mathbf{n}]$ is a source image and $y[\mathbf{n}]$ is the target image that will ultimately be approximated by motion compensation based on $x[\mathbf{n}]$. Keypoints $\mathbf{k}$ are found within the target frame $y[\mathbf{n}]$, and for each $\mathbf{k}$ a motion vector $\mathbf{v_k}$ is estimated from

$$\mathbf{v_k} = \underset{\mathbf{u}}{\operatorname{argmin}}\, J_{\mathbf{k}}(\mathbf{u})$$

where

$$J_{\mathbf{k}}(\mathbf{u}) = \sum_{\mathbf{n}\in\mathcal{B}_{\mathbf{k}}} |y[\mathbf{n}] - x[\mathbf{n}-\mathbf{u}]|^p$$

where $p = 1$ for the SAD metric and $p = 2$ for the MSE metric[1], while $\mathcal{B}_{\mathbf{k}}$ is a local region (a "block") of image samples, around the keypoint location $\mathbf{k}$.

In this project, you will work with blocks of $(2H+1) \times (2H+1)$ samples that are centered at $\mathbf{k}$, and you will avoid using any keypoints that are within H rows or columns of the boundary of the target image $y[\mathbf{n}]$. The value of H will be an argument to your programs so that you can explore the impact of block size upon the estimated results.

[1] Strictly speaking, with $p = 2$, the metric is a "sum of squared errors" rather than "mean squared error," but one could always normalize the sum of squared errors without changing the matching result.

Based on the estimated motion vectors $\mathbf{v_k}$, a global motion vector $\mathbf{v}$ can be found using a least squares fitting procedure, as follows:

$$\mathbf{v} = \underset{\mathbf{u}}{\operatorname{argmin}} J_{\text{fit}}(\mathbf{u})$$

where

$$J_{\text{fit}}(\mathbf{u}) = \sum_{\mathbf{k}} \|\mathbf{u} - \mathbf{v_k}\|^2 = \sum_{\mathbf{k}} (u_x - v_{\mathbf{k},x})^2 + \sum_{\mathbf{k}} (u_y - v_{\mathbf{k},y})^2$$

Here, the subscripts x and y denote the horizontal and vertical components of a motion vector. The solution to this minimization problem is easily found (e.g., by setting derivatives with respect to u_x and u_y to 0) to be

$$\mathbf{v} = \frac{\sum_{\mathbf{k}} \mathbf{v_k}}{\sum_{\mathbf{k}} 1} \tag{1}$$

That is, one only needs to take the average of the individual motion vectors found at each keypoint.

Global motion is only applicable to certain types of video content. In general, one should use a richer global motion model than pure translation – common models include the affine or perspective models, as described in your lecture notes. To simplify things for this project, we limit our attention to global translational motion, as described above, and test content suitable for exploring the solution will be uploaded to the class web-site for you to use in this project.

3 Keypoint Detection

In this project, you will find keypoints $\mathbf{k}$ using a method that is inspired by the Harris corner detector. Specifically, you will evaluate the following figure of merit at each location $\mathbf{n}$:

$$K[\mathbf{n}] = \frac{\det\left(\bar{\Gamma}_\sigma[\mathbf{n}]\right)}{\operatorname{trace}\left(\bar{\Gamma}_\sigma[\mathbf{n}]\right)}$$

Here, $\bar{\Gamma}_\sigma[\mathbf{n}]$ is a 2×2 matrix formed by aggregating local matrices $\Gamma_\sigma[\mathbf{n}]$ over the same region ("block") that is used to compute the motion itself. Specifically,

$$\bar{\Gamma}_\sigma[\mathbf{n}] = \sum_{\mathbf{j} \in \mathcal{B}_{\mathbf{n}}} \Gamma_\sigma[\mathbf{j}]$$

and

$$\Gamma_\sigma[\mathbf{n}] = \nabla_\sigma[\mathbf{n}] \cdot \nabla_\sigma^t[\mathbf{n}]$$

Here, $\nabla_\sigma[\mathbf{n}]$ is a scale dependent gradient vector that could be obtained by applying derivative of Gaussian (DOG) filters to the target image $y[\mathbf{n}]$. However, for simplicity, in this project you will use a difference of Gaussian approach, setting

$$\nabla_\sigma[n_1, n_2] = \frac{1}{2} \begin{pmatrix} y_\sigma[n_1 + 1, n_2] - y_\sigma[n_1 - 1, n_2] \\ y_\sigma[n_1, n_2 + 1] - y_\sigma[n_1, n_2 - 1] \end{pmatrix}$$

where $y_\sigma[\mathbf{n}]$ is obtained by subjecting $y[\mathbf{n}]$ to a Gaussian low-pass filter with scale-parameter (standard deviation) σ.

There is quite a lot to take in here. To help clarify things, here is a summary of the steps involved:

1. Apply a Gaussian filter to the target fame $y[\mathbf{n}]$ to get $y_\sigma[\mathbf{n}]$.
2. Evaluate the gradient vector $\nabla_\sigma[\mathbf{n}]$ at each pixel location $\mathbf{n}$, using $y[\mathbf{n}]$.
3. Evaluate the so-called "structure tensor" $\Gamma_\sigma[\mathbf{n}]$ at each pixel location $\mathbf{n}$, using $\nabla_\sigma[\mathbf{n}]$.
4. Evaluate $\bar{\Gamma}_\sigma[\mathbf{n}]$ from $\Gamma_\sigma[\mathbf{n}]$. Note that this is a moving sum operation with window size $(2H+1) \times (2H+1)$, which can be solved efficiently.
5. Evaluate $K[\mathbf{n}]$ from $\bar{\Gamma}_\sigma[\mathbf{n}]$, being careful to avoid division by 0.

The keypoints $\mathbf{k}$ that you finally select should be local maxima of the figure of merit $K[\mathbf{n}]$, such that

$$K[\mathbf{k}] > \tau$$

where τ is a threshold. By "local maximum" we mean that $K[\mathbf{k}]$ must satisfy

$$K[\mathbf{k}] \geq K[\mathbf{k}+\mathbf{j}], \text{ for } \mathbf{j} \in \left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \end{pmatrix} \right\}$$

Evidently, the number of keypoints that satisfy these two conditions (local maximum and $K[\mathbf{k}] > \tau$) is a monotonically decreasing function of τ. In this project, you will adjust τ to the largest threshold that yields at least N_K keypoints, where N_K is an argument to your program. The simplest way to do this is with a bisection search:

- Starting with an initial guess τ_0, you work out the number of keypoints that satisfy the criteria, call it N_{τ_0}.
- If $N_{\tau_0} > N_K$, double the threshold setting $\tau_1 = 2\tau_0$; if less, halve the threshold, setting $\tau_1 = \tau_{0/2}$. Continue in this way until you get two thresholds τ_a (above) and τ_b (below) such that $\tau_a > \tau_b$ and $N_{\tau_a} \leq N_K \leq N_{\tau_b}$.
- From there on, each subsequent threshold that you test bisects the interval $[\tau_b, \tau_a]$. That is, you try $\tau_c = (\tau_a + \tau_b)/2$, finding N_{τ_c} which must lie in the range $N_{\tau_a} \leq N_{\tau_c} \leq N_{\tau_b}$. If $N_{\tau_c} < N_K$, τ_c is too large, so replace τ_a with τ_c and continue. Otherwise, replace τ_b with τ_c before continuing.
- The algorithm finishes once you $N_{\tau_a} = N_{\tau_b}$, at which point the solution is $\tau = \tau_b$.

4 Tasks

Task 1: (4 marks) Building upon "Lab 5" (see the course web-site) if you like, or else starting from scratch, develop a global translational motion estimator that uses a fixed set of keypoints. For this exercise, there is no automated discovery of keypoints, and there is no need to implement the Harris-like keypoint detector described in the notes above, but it would be sensible for you to implement your algorithm in such a way that the keypoints that are used are placed in array, whose contents can be changed to implement later tasks.

Your program should accept two BMP images from the test sets on the course web-site, plus three arguments H, S and Δ, as follows:

- H is the block half-size from the introductory notes in Section 2 above.
- S is the search range, such that the motion estimation procedure explores motion vectors $\mathbf{v_k}$ in the range $\pm S$ pixels in each direction, for a total of $(2S+1)^2$ candidate motion vectors.
- Δ is the separation between keypoints, such that $\mathbf{k}$ is of the form

$$\begin{pmatrix} k_1 \\ k_2 \end{pmatrix} = H + S + \Delta \begin{pmatrix} n_1 \\ n_2 \end{pmatrix}$$

  where n_1 and n_2 are integers in the range

$$0 \leq n_1 < \frac{\text{width} - 2(H+S)}{\Delta},\ 0 \leq n_2 < \frac{\text{height} - 2(H+S)}{\Delta}$$

  Note that these keypoints have the property that you should not need to rely upon boundary extension during the search process, since no keypoints exist close to the image boundaries.

Following Lab 5, the metric employed here for the motion search should be SAD.

Your program should work out the global motion vector $\mathbf{v}$, derived from equation (1), printing this as a result that will be assessed.

Task 2: (4 marks) Extend the work in Task 1 to produce a motion compensated image $y'[\mathbf{n}] = x_{\mathbf{v}}[\mathbf{n}]$. That is, you need to translate image $x[\mathbf{n}]$ by the global motion $\mathbf{v}$, so that it aligns (approximately) with image $y[\mathbf{n}]$. Note that $\mathbf{v}$ is not generally integer valued, even though the $\mathbf{v_k}$ values here are integer-valued. Thus, you will need to use interpolation to perform the shifting operation. For this purpose you should simply use **bilinear interpolation**. You can, if you like, use the approach in "Lab 5," where individual blocks are separately motion compensated, except that you must use the global motion vector in every block, and use bi-linear interpolation. Equivalently, you can simply shift the original image by $\mathbf{v}$, which may be simpler.

- Boundary extension is very important here. The code from "Lab 5" does not select motion vectors that will map a block beyond the boundaries of the source frame, but since a single global vector is being selected for all blocks, this constraint would force $\mathbf{v}$ to be $\mathbf{0}$, which is not interesting. It follows that your modified code will need to extend the original source image by at most S pixels in any direction.
- Use the **point-symmetric boundary extension** method covered in lectures, rather than the symmetric extension method.
- Your program should write out the motion compensated image $y'[\mathbf{n}]$ in BMP format
- Your program should also compute the MSE between $y'[\mathbf{n}]$ and $y[\mathbf{n}]$, printing this value along with the global motion vector $\mathbf{v}$.
- You will need to explore the use of different H, S and Δ values and explain their significance. In particular, you need to explain what happens as H is changed.

Task 3: (1 mark) Modify the search criterion used in Task 2 to find motion vectors, so that the best vector is considered to be that which minimizes MSE over the block, rather than SAD. What impact does this have upon the MSE value computed in Task 2? Can you find a good explanation for your observations?

Task 4: (4 marks) Write a program to find a set of N_K keypoints, following the method described in Section 3. You can do this completely independently from the preceding tasks. Your program should accept two arguments N_K and H, in addition to a single BMP input image, and it should write out the original image overlaid with 3×3 blocks of red pixels at each keypoint location. For each of interpretation, you should halve the intensity of all original input samples in the output image, so that the 3×3 red blocks are easier to see. You should restrict the keypoint algorithm to searching for keypoints that lie at least H pixels in from the image boundaries, so that you don't need to rely upon boundary extension.

- If your implementation does not involve any Gaussian filtering, the maximum number of marks you can receive here is limited to 2.
- If your program does not allow σ to be supplied on the command-line (at least covering the range $1 \leq \sigma \leq 4$), the maximum number of marks you can receive is limited to 3.

Task 5: (3 marks) Combine Task 2 with Task 4, so that the keypoints found in Task 4 are used for Task 2. Here, you should restrict the algorithm to search for keypoints that lie at least $H + S$ pixels away from the image boundaries, so that you don't need to rely upon boundary extension for either the keypoint discovery process or the motion estimation process. Boundary replication is important only for generating the motion compensated output image, as discussed in Task 2.

- Explore the improvements obtained in global motion compensated MSE when using this scheme, in comparison to the uniform set of keypoints in Task 2, for various block sizes and various values of σ.

5 Assessment

You should not rely upon implementing this project within the scheduled laboratory sessions. There will be some opportunity to do this within your Week 9 laboratory and at the start of your Week 10 laboratory, but you must be prepared to demonstrate your work and be assessed individually starting from the second hour of your Week 10 lab session.

In order to obtain the full marks for any given task, you must have a working program to demonstrate and you must be able to explain how it works and **answer questions very quickly**!! We will not be able to wait around while you fiddle with running your program from the debugger or viewing images with the Windows viewer, so have things set up for easy validation by the course staff and **use "mi_viewer"** to show your images directly from the command-line, allowing multiple images to be viewed simultaneously and individual pixel values to be read-off by the marker with simple mouse clicks.

You may feel free to re-use code from previous laboratory sessions, so long as you understand it. You may also discuss the project with other students in the class, but **your programs should be your own original work**. As with assessment tasks in other subjects, UNSW regards **plagiarism as a serious offense**; for a first offense, the penalty would be loss of all marks for the project.

You will also be asked to submit your code via an Assignment item on the course's Moodle page, **within 2 hours of the end of your Week 10 lab session**.