Motion Estimation and Segmentation in Images

Siddharth Jain

November 25, 2003

Abstract

Given a pair of images of a scene taken from two different viewpoints by a camera undergoing primarily translational motion with little rotation, we attempt to segment the pixels undergoing coherent motions and estimate their motion by fitting an affine motion model. This is done by starting with an initial guess of motion estimates. The motion estimates are then used to segment the pixels in the images into different layers or clusters. Affine motion models are then fitted to pixels in the individual clusters to obtain new motion estimates and the process is continued iteratively. Applications of this work include for example splitting a scene into different layers such as a foreground layer which is close to the camera and in which pixels are moving rapidly and a background layer which is far away from the camera and in which pixels are undergoing relatively little motion.

1 Introduction

Consider two images of a scene taken by a camera from two different viewpoints e.g. Figure 1(a) and 1(b). The tree is much closer to the camera than the house and therefore between the two images, the tree undergoes a motion greater than the motion of the house. We are interested in grouping pixels undergoing similar motions and thereby segmenting the image into layers undergoing similar motions.

Consider a 3D point $P = (X, Y, Z)$ observed from two viewpoints C_0 and C_1 by a perspective imaging camera as shown in Figure 2. Without loss of generality, C_0 can be taken to be the world origin and the local $\hat{x}$, $\hat{y}$, $\hat{z}$ axes of the camera at position C_0 can be taken to be the world $\hat{x}$, $\hat{y}$, $\hat{z}$ axes. The coordinates of P after perspective projection are then given

(a)  (b) 

Figure 1: A pair of images. We want to group pixels undergoing similar motions.

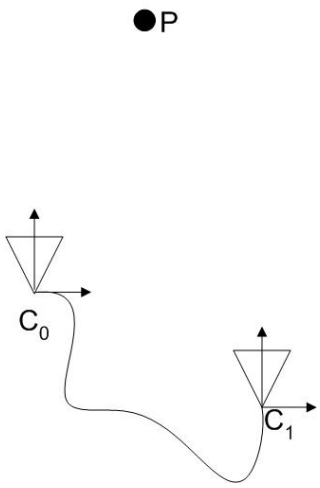


Figure 2: A perspective imaging camera capturing two images of point P from positions C_0 and C_1. The camera does not undergo any rotational motion.

by $Q = (X/Z, Y/Z, 1)$. In order to get the pixel coordinates of P in the camera image, Q has to be transformed by the intrinsic camera calibration matrix [MSKS03]. Letting R denote the pixel coordinates of P in the image taken by the camera,

$$R = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} fs_x & fs_\theta & O_x \\ 0 & fs_y & O_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X/Z \\ Y/Z \\ 1 \end{pmatrix} \tag{1}$$

For most cameras the skew of the pixel s_θ is very small and can be neglected. Then the pixel coords of P in image 1, (u, v) are given by,

$$u = fs_x X/Z + O_x$$
$$v = fs_y Y/Z + O_y$$

In the second view, let the center of projection (COP) of the camera be positioned at $C_1 = (T_x, T_y, T_z)$. If the camera has not undergone any rotational motion so that the local $\hat{x}$, $\hat{y}$, $\hat{z}$ axes at C_1 point in the same direction as the local axes at C_0, then the coords of P relative to the local coordinate system at C_1 are given by $(X - T_x, Y - T_y, Z - T_z)$ and the pixel coords in image 2, (x, y) are given by,

$$x = fs_x \frac{X - T_x}{Z - T_z} + O_x$$
$$y = fs_y \frac{Y - T_y}{Z - T_z} + O_y$$

which after some manipulation can be written as

$$\begin{aligned} x &= \left(\frac{1}{1 - T_z/Z}\right)(fs_x X/Z + O_x) + O_x\left(1 - \frac{1}{1 - T_z/Z}\right) - fs_x \frac{T_x/Z}{1 - T_z/Z} \\ &= F_1 u + F_2 \\ y &= \left(\frac{1}{1 - T_z/Z}\right)(fs_y Y/Z + O_y) + O_y\left(1 - \frac{1}{1 - T_z/Z}\right) - fs_y \frac{T_y/Z}{1 - T_z/Z} \\ &= G_1 v + G_2 \end{aligned}$$

For constant f, s_x, s_y, O_x, O_y, T_x, T_y, T_z; F_1, F_2, G_1, G_2 are functions of Z only. Morever, G_1 is equal to F_1 in above. Thus,

$$\begin{pmatrix} x & y & 1 \end{pmatrix} = \begin{pmatrix} u & v & 1 \end{pmatrix} \begin{pmatrix} F_1(Z) & 0 & 0 \\ 0 & G_1(Z) & 0 \\ F_2(Z) & G_2(Z) & 1 \end{pmatrix} \tag{2}$$

which we can write concisely as $p_2 = p_1 \cdot \theta$ where $p_2 = (x, y, 1)$, $p_1 = (u, v, 1)$ are row vectors and θ is given by the matrix in above equation. The matrix θ tells how pixels move from image 1 to image 2 and is thus known as the motion matrix. If we have a group of pixels that are images of 3D points with the same Z i.e. having the same depth, these group of pixels have the same motion matrix θ associated with them. It is thus possible to identify pixels in the image that are moving in the same fashion — these pixels have the same motion matrix; and regions in the image that are moving differently will have different motion matrices associated with them. Recall that θ in equation (2) holds when the camera does not undergo any rotational motion between the two capture points C_0 and C_1. In practice little rotational motion of the camera is modeled by a motion matrix of the form

$$\theta = \begin{pmatrix} \theta_{11} & \theta_{12} & 0 \\ \theta_{21} & \theta_{22} & 0 \\ \theta_{31} & \theta_{32} & 1 \end{pmatrix} \tag{3}$$

and this is referred to as an affine motion model. We now have a chicken-and-egg problem in that

- If we know the motion matrices of the different layers in the image, we can segment each pixel based on which motion matrix fits it better.
- Conversely, if we know the motion segmentation and correspondence of the pixels in the two images we can estimate the motion matrix using $p_2 = p_1 \cdot \theta$.

In reality we do not know either. The problem is solved using a K means or EM algorithm in which we start with initial guesses of motion matrices and do the motion segmentation. Then using the motion segmentation we find improved estimates of motion matrices and iterate. Following sections describe the algorithm in more detail.

2 The Algorithm

The algorithm starts by assuming that the image can be segmented into M layers and some initial guesses of the motion matrices $(\theta_1, \theta_2, \ldots, \theta_M)$ for the M layers are given. It then proceeds according to following steps:

2.1 Motion Segmentation

For each pixel the algorithm finds the θ_i that fits it best and declares the pixel as belonging to layer i. This is done in the following way: let p denote

the pixel $(u, v, 1)$ in image 1 which is to be classified. Under motion model θ_i, p maps to $\text{dest}(p|\theta_i) = (x, y, 1) = p \cdot \theta_i$ in image 2. Let $I(p)$ denote the RGB color value at pixel p. Thus $I(p) = (r, g, b)$ where r, g, b are integers lying between 0 and 255 for a true 24 bit RGB color image ($0 \leq r, g, b \leq 255$). Further if $I(p_1) = (r_1, g_1, b_1)$ and $I(p_2) = (r_2, g_2, b_2)$ are the color values at two pixels p_1 and p_2 let $\|I(p_1) - I(p_2)\|$ denote the color difference between pixels p_1 and p_2 which is assumed to lie between 0 and 255 and can be measured for example as $\sqrt{\frac{(r_1-r_2)^2+(g_1-g_2)^2+(b_1-b_2)^2}{3}}$. A simple way of classifying pixel p is to assign it to the layer that gives the minimum value of $\|I(p) - I(\text{dest}(p|\theta_i))\|$ i.e. if the variable $z(p, i)$ denotes the membership of pixel p in image 1 to layer i and

$$\theta_i = \arg\min_{\theta} \|I(p) - I(\text{dest}(p|\theta))\|$$

then the algorithm sets

$$z(p, j) = \begin{cases} 1 & j = i \\ 0 & \text{otherwise} \end{cases}$$

Above approach relies on several assumptions which we make explicit now. If (a) the motion of pixel p is indeed affine and θ_i is known accurately, and (b) the 3D point whose image is p is unoccluded in image 2, then, the corresponding pixel of p in image 2 is given by $\text{dest}(p|\theta_i)$. In addition if (a) a Lambertian reflectance model[1] applies to the 3D point whose image is p, and (b) the lighting conditions do not change from image 1 to image 2, then, color of $\text{dest}(p|\theta_i)$ will be same as color of p. Thus $\|I(p) - I(\text{dest}(p|\theta_i))\|$ will be small only when all of the above conditions hold. For the experiments we demonstrate in this report, the assumptions of Lambertian model and constant lighting hold fairly well but since θ_i is only known approximately and occlusions may occur between images, the algorithm searches for the best match to p in a window centered at $\text{dest}(p|\theta_i)$. The details of the matching process are as follows:

2.1.1 Finding the BestMatch

A rectangular window w in an image is a collection of pixels denoted in following ways:

[1] A Lambertian reflectance model simply means that the reflection from the object is not dependent on the direction from which the object is viewed i.e. the object reflects equally in all directions. A mirror is an example of an object that is not Lambertian as it reflects primarily in one particular direction; such objects are termed as specular.

- $w = \text{win}((x_1, y_1)\text{-}(x_2, y_2))$ denotes the window with top-left corner at pixel (x_1, y_1) and bottom-right corner at pixel (x_2, y_2)
- $w = \text{win}(x, y, wxmin, wxmax, wymin, wymax)$ denotes the window with top-left corner at $(x + wxmin, y + wymin)$ and bottom-right corner at $(x + wxmax, y + wymax)$. Normally in our discussion:
  $wxmin + wxmax = 0$
  $wymin + wymax = 0$
  and so (x, y) is the center pixel in this window. Morever if $wxmax = wymax$ then the window is symmetric in x and y directions and $wxmax$ is referred to as the half-size of the window.
- $w = \text{win}(wxmin, wxmax, wymin, wymax)$ is window of size $(wxmax - wxmin + 1) \times (wymax - wymin + 1)$ pixels whose center pixel is unspecified.

For two windows of the same size $w_1 = \text{win}(x_1, y_1, wxmin, wxmax, wymin, wymax)$ and $w_2 = \text{win}(x_2, y_2, wxmin, wxmax, wymin, wymax)$ a pixel (x, y) in w_1 has a corresponding pixel in w_2 given by $(x_2 + (x - x_1), y_2 + (y - y_1))$[2]. For two windows of the same size we can measure the difference between them as the rms (root mean square) difference in color values of corresponding pixels:

$$\text{RMS}(w_1, w_2) = \sqrt{\frac{\sum_{y=wymin}^{wymax} \sum_{x=wxmin}^{wxmax} \delta(x_1 + x, y_1 + y)\delta(x_2 + x, y_2 + y) d^2(I(x_1 + x, y_1 + y), I(x_2 + x, y_2 + y))}{\text{N}(w_1, w_2)}} \tag{4}$$

where d denotes the difference between two colors (which can be measured for example as root mean square of difference of RGB values).

$$\delta(x, y) = \begin{cases} 1 & \text{if } I(x, y)\text{is known} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{N}(w_1, w_2) = \sum_{y=wymin}^{wymax} \sum_{x=wxmin}^{wxmax} \delta(x_1 + x, y_1 + y)\delta(x_2 + x, y_2 + y)$$

$\text{N}(w_1, w_2)$ is the number of nonzero terms in the numerator of (4). The δ's have been introduced in equation (4) to take care of pixels at the border of the image. $\text{RMS}(w_1, w_2)$ thus lies between 0 and 255.

In order to find the best matching pixel to p under motion given by θ_i a correlation window w_p is taken centered at p and whose half-size is set

[2]Note that this use of the word correspondence must not be confused with the other context in which we use the word. Consider a 3D point P that is unoccluded in both image 1 and image 2. Let the image of P in image 1 be given by pixel p. When we speak of the corresponding point of p in image 2, we mean the pixel in image 2 that is the image of the same 3D point P.

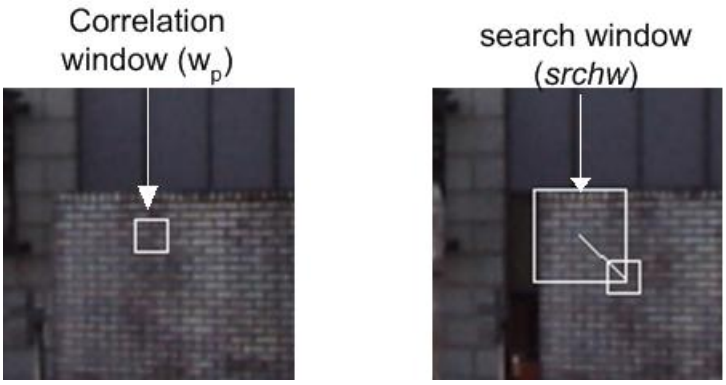


Figure 3: Finding the best match. The center of the best matching window lies in the search window and its RMS difference from w_p is minimum.

to 4 in all of our experiments. The algorithm then finds the best matching window to w_p in image 2 denoted by best$(w_p|\theta_i)$ as illustrated in Figure 3. best$(w_p|\theta_i)$ is the window that minimizes the difference as measured by the RMS metric in equation (4) and whose center is constrained to lie in a search window $srchw_i$ centered at dest$(p|\theta_i)$. The best matching pixel to p is then given by the center pixel of best$(w_p|\theta_i)$ and denoted by best$(p|\theta_i)$. $srchw_i$ is initialized to 32 and modified adaptively for different layers as explained later in Section 2.2. Few points worth mentioning about the search window are:

- If p belongs to layer i and θ_i is known accurately then best$(p|\theta_i)$ would lie close to dest$(p|\theta_i)$ and the search window would be small. Since θ_i is not known accurately, the algorithm takes into account the uncertainty or error in estimation of θ_i by searching in a window centered at dest$(p|\theta_i)$. At initialization the uncertainty is high and so we set half-size of the search window to a high value of 32 pixels.
- Making the search window too large is also problematic because:
  - The time taken by the algorithm would increase in proportion to the size of the search window. Finding the best match is probably the most expensive component of the algorithm.
  - More importantly, if p does not belong to θ_i then the algorithm should not be able to find any good match to p by searching in a window around dest$(p|\theta_i)$. Therefore size of the search window should not be made too large.
  - Another reason is that if p does belong to θ_i, we reduce the chances of getting spurious matches by searching in a window centered at

dest$(p|\theta_i)$ instead of searching for the best match in the whole image.

Once the best match to pixel p under motion matrix θ_i has been found a difference measure between p and best$(p|\theta_i)$ is defined as the product of two quantities: 1. Euclidian distance of best matching pixel from the predicted destination pixel, and, 2. color difference between best matching window and correlation window at p.

$$\mathcal{D}(p,\theta_i) \stackrel{def}{=} \|\text{best}(p|\theta_i) - \text{dest}(p|\theta_i)\| * \text{RMS}(w_p, \text{best}(w_p|\theta_i)) \tag{5}$$

For a K means algorithm pixel p is classified as belonging to the layer that gives minimum value of $\mathcal{D}(p,\theta)$. Thus letting

$$\theta_i = \arg\min_{\theta} \mathcal{D}(p,\theta) \tag{6}$$

$$z(p,j) = \begin{cases} 1 & j = i \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

The reason of including the Euclidian distance in the definition of $\mathcal{D}$ is that if multiple layers have the same value of best(p,θ_i) which can occur for example if the search window is too large, the algorithm will pick out the layer for which best(p,θ_i) lies closest to the predicted destination. For EM a soft classification needs to be done

$$z(p,i) = \frac{1/\mathcal{D}(p,\theta_i)}{\sum_{j=1}^{j=M} 1/\mathcal{D}(p,\theta_j)} \tag{8}$$

This completes the motion segmentation.

2.2 Compute Sigma

Associated with each layer i a quantity σ_i is defined given by

$$\sigma_i \stackrel{\text{def}}{=} \sqrt{\frac{\sum_p z(p,i)\|I(p) - I(\text{dest}(p|\theta_i))\|^2}{\sum_p z(p,i)}} \tag{9}$$

σ_i in a sense tells how well θ_i describes the motion of layer i. This suggests that we can use σ_i to adaptively update the size of the search window for layer i. The algorithm updates the half-size of the search window for each layer as

$$srchw_i = \min(32, \max(5, \sigma_i/2)) \tag{10}$$

2.3 Motion Estimation

After motion segmentation the algorithm finds improved estimates of θ_i^{t+1}. We first discuss motion estimation for K means and come to EM later. Under the assumed affine motion model if p belongs to layer i and best$(p|\theta_i^t)$ is best match to p under θ_i^t found in the previous section on motion segmentation then

$$p \cdot \theta_i^{t+1} = \text{best}(p|\theta_i^t) \tag{11}$$

The superscript t denotes the time step. Thus we can form a linear system of equations given by

$$\begin{pmatrix} u_1 & v_1 & 1 \\ u_2 & v_2 & 1 \\ \vdots & \vdots & \vdots \\ u_N & v_N & 1 \end{pmatrix} \theta_i^{t+1} = \begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots \\ x_N & y_N & 1 \end{pmatrix} \tag{12}$$

or

$$A \cdot \theta_i^{t+1} = B \tag{13}$$

where $(u_m, v_m, 1)$ is pixel in image 1 classified as belonging to layer i and $(x_m, y_m, 1)$ is the best match to $(u_m, v_m, 1)$ found in the motion segmentation step. Thus the algorithm can find improved motion estimates for each layer. For each matching pair $(p, \text{best}(p|\theta_i)$ the algorithm assigns a confidence value or weight given by

$$\eta^t(p,i) \stackrel{\text{def}}{=} 1 - \frac{\text{RMS}(w_p, \text{best}(w_p|\theta_i^t))}{K} \tag{14}$$

where K is a normalizing constant to ensure that η lies between 0 and 1. Since RMS lies between 0 and 255, K would be set to 255. However, as explained later in Section 3 we improve the algorithm further and use pixels for which RMS is small. In that case we set $K = 60$ in all our experiments. The updated motion matrix using the weights is then given by

$$\theta_i^{t+1} = \arg\min_{\theta} \sum_p \eta^2(p,i) \|p \cdot \theta - \text{best}(p|\theta_i^t)\|^2 \tag{15}$$

The summation is taken over all pixels in image 1 that have been classified as belonging to layer i. It is easily shown that the solution of (15) is obtained by solving (13) where each row of A and B is multiplied by the appropriate weight $\eta(p,i)$. The extension to EM is straightforward:

$$\theta_i^{t+1} = \arg\min_{\theta} \sum_p z^2(p,i)\eta^2(p,i) \|p \cdot \theta - \text{best}(p|\theta_i^t)\|^2 \tag{16}$$

The summation is taken over all pixels in the image in this case. The new motion estimates found in this way are then used to refine the motion segmentation and the algorithm iterates.

The algorithm can be very briefly summarized as:
$\theta(1) \leftarrow$ initial motion estimates
for iter = 1:T
E step: z(iter) $\leftarrow$ motion segmentation based on θ(iter) and the data available as described in Section 2.1;
srchw $\leftarrow$ update as in Section 2.2;
M step: θ(iter+1) $\leftarrow$ new motion estimates based on z(iter) and data available as described in Section 2.3;
end for

In practice performing motion segmentation for each pixel takes up inordinately huge amounts of time. Also note that in principle 3 perfect correspondences is all that is required to find θ. Thus to find θ the algorithm needs a few but good corresponding pairs. In the next section we mention a number of modifications to the basic algorithm described in this section to get faster and more accurate results.

3 Some modifications to the basic algorithm

At each iteration the algorithm randomly picks a pixel in image 1 and motion segments it. If the pixel lies in a featureless region[3] it is difficult to find accurately the corresponding pixel in the other image because multiple pixels will seem to match and there are not enough features in the correlation window to distinguish the multiple matches e.g. try finding the corresponding pixels of the pixels marked red in Figure 4. Thus if the algorithm gets a pixel p that lies in a featureless region, it discards the pixel and does not use it for motion estimation. Specifically, if the standard deviation of color of pixels in the correlation window centered at p is less than K_2 ($K_2 = 15$ in all our experiments), the pixel is motion segmented but not used for motion estimation.

$$\text{Feature test: Is StdDev}(\text{win}(p, -4, 4, -4, 4)) > K_2? \tag{17}$$

Another thing to note is that if RMS(w_p, best($w_p|\theta$)) is large, this suggests that the best match is not good enough and maybe the algorithm is better

[3]By featureless region we mean a region which mostly has a constant color i.e. has low color variance.

(a)  (b) 

Figure 4: Try finding corresponding pixels in (b) of the pixels marked red in (a). It is difficult to match featureless regions as they do not have any distinguishing characteristics to tell them apart from spurious matches.

off not using this pair for motion estimation. Remember theoretically the algorithm just requires a few good matches to estimate θ and the process of finding the best match is guaranteed to give a best match — the algorithm should further check if the best match is really a good match or not. Thus if RMS(w_p, best($w_p|\theta$)) $> K_3$ the pixel is not used for motion estimation. We set $K_3 = 30$ in our experiments.

$$\text{Good Match test: Is RMS}(w_p, \text{best}(w_p|\theta)) < K_3? \tag{18}$$

The algorithm keeps choosing pixels randomly making sure that no pixel is chosen multiple times, motion segments them and uses them for motion estimation if they pass the enough features and good match tests in equations (17, 18). The algorithm chooses as many pixels as necessary so that it has at least $K_4 = 10$ pairs for motion estimation of each layer. The 'sparse processing' speeds up the algorithm by orders of magnitude. A final tweak that can be employed is to note that in equation (3) θ_{11}, θ_{22} are close to unity whereas θ_{12}, θ_{21} are close to zero. We impose following constraints on the optimization problem in equations (15, 16):

$$0.8 < \theta_{11}, \theta_{22} < 1.2 \tag{19}$$

$$-0.2 < \theta_{12}, \theta_{21} < 0.2 \tag{20}$$

With these constraints we now have a constrained least squares problem which can be solved using standard methods [Pan98].

We quickly recap the salient features of the algorithm below:

- Randomization
- Feature test equation (17)

- Good Match test equation (18)
- Adaptive search window size Section 2.2
- Weighted objective function equation (16)
- Constrained Optimization equations (19, 20)

A detailed pseudocode is given in Section 6 which also takes into account pathological inputs in which all the pixels have been motion segmented in random order but the algorithm is not able to obtain at least $K_4 = 10$ matches, that pass the feature and good match tests, for motion estimation of each layer.

4 Results

In this section we show results on 4 test cases in Figures 5-8. We used two machines for computation — one running at 2783 Mhz and another running at 699 Mhz. Both machines use Intel Pentium processors and run on Windows OS. The time taken mentioned in this section is w.r.t. a 2783 Mhz processor where the time taken by the processor running at 699 Mhz was multiplied by $699/2783 = 0.25$ to translate to the time scale on the 2783 Mhz machine. The code is written in MATLAB. We implemented the K means version of the algorithm and did 10 iterations on all 4 test cases. The number of layers M was set to 2 and the motion matrices were initialized to $\theta_1 = \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 35.0 & 0.0 & 1.0 \end{pmatrix}$ and $\theta_2 = \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ -10.0 & 5.0 & 1.0 \end{pmatrix}$ for the two layers. The flower garden images (Figure 5) are 200×137 pixels and all other images are 200×150 pixels. All images are 24 bit true color images. Based on the randomization and sparse processing mentioned in Section 3, the running time of the algorithm is independent of the image size. Generating a dense motion segmentation as in Figures 5-8 (c), (d), (g), (h) however requires segmenting each pixel and the time taken for this obviously increases in proportion to the image size. In Figures 5-8 (a), (b) show the input images; (c), (d) show motion segmentation of (a) at the end of 10 iterations; (e), (f) show motion estimates for the two layers in (c), (d) at the end of 10 iterations; (g), (h) show motion segmentation for (a) at some iteration when σ is small and is mentioned for the different cases respectively later in the discussion; (i), (j) show motion estimates for the layers in (g) and (h); (k) shows the variation of σ vs. the iterations for the two layers. We describe the results in more detail below:

Flower Garden: This is shown in Figure 5. Figure 5 (k) shows a general trend of decrease in σ as the iterations progress although it exhibits a little oscillatory behavior. Figures 5 (c), (d) show the motion segmentation of Figure 5(a) into two layers after 10 iterations. The motion estimates for the two layers in (c), (d) after 10 iterations are given in Figures 5 (e) and (f). The motion segmentation in Figures 5 (c) and (d) does not seem to be perfect as there are some parts of the tree in Figure 5 (c) and parts of the house in Figure 5 (d). In Figures 5 (g), (h) we show motion segmentation resulting from the motion estimates at the beginning of iteration 7. These motion estimates, which are listed in (i) and (j), are seen to give low σ values in Figure 5 (k). The motion segmentation also appears better compared to Figures (c) and (d). There is less of the tree in (g) and the house is almost all in (g). The time taken was 7.5 hours for unoptimized MATLAB code. Executable code should run much faster (10 times faster?).

Wells Fargo: This is shown in Figure 6. Fig. (k) again shows a general trend of decrease in σ with the iterations although the oscillatory behavior is much greater. Motion segmentation in (c) and (d) does not seem to be very promising. Fig. (g) and (h) show motion segmentation using motion matrices (i) and (j). These motion matrices are obtained at the beginning of iteration 10. The motion segmentation is not very different from (c) and (d) but note the low σ values in Fig. (k) that the motion matrices in (i) and (j) give. The algorithm seems to have got stuck in a local minima. The time taken was 6.2 hours.

Ross: This is shown in Figure 7. Fig. (k) shows oscillatory decrease in σ with the iterations. Fig. (c), (d) show motion segmentation after 10 iterations and (g), (h) show motion segmentation at iteration 6. Results appear satisfactory — the building façade forms one layer and the tree forms the other layer. This is a hard test case because the tree is thin and occupies little portion of the image. Time taken was 7.6 hours.

Fourth: This is shown in Figure 8. Once again Fig. (k) shows decrease in sigma with the iterations. The oscillatory behavior is also present. Fig. (c), (d) show motion segmentation after 10 iterations and (g), (h) show motion segmentation at iteration 8. The results seem fair enough — the wooden post has been marked out fairly well in both iteration 8 and at the final step. Time taken was 4.84 hours.

The test cases give an idea of the performance of the algorithm. In practice the algorithm has to face many challenges such as:

- Some pixels are present in image 1 but not in image 2 and vice versa. This messes up correspondences and happens because of two reasons: 1. occlusions — an object that is present in an image may be occluded

by some other object in the other image and hence not appear in the other image. 2. Field of View effects — pixels in non-overlapping portion of the two images are present only in one image. Note this for example in Figure 7 (d) and (h) in which one can see a vertical stripe at the right edge of the image. Pixels in this stripe do not appear in Figure 7 (b).

- Finding the correspondences, i.e. the best match, accurately is tricky. Recall that here we have assumed a Lambertian reflectance model with constant lighting conditions between the two images.
- The affine motion model is only an approximation to the real motion of the pixels.
- Setting $M = 2$ may not be quite true for all the test cases. For example in Figure 6 maybe we should use more layers. Similarly in Figure 8 maybe the wooden post, house, and the tree form 3 layers (there could be an additional fourth layer comprising the distant background consisting of the sky etc.).

5 Conclusions

We presented an iterative K means or EM algorithm for motion estimation and segmentation in images. There is ample scope for further work:

- Perhaps the most important addition would be to determine the number of layers M automatically.
- Extension of this work to a video sequence consisting of more than just two images. Note that in equation (2), F_1, F_2, G_1, G_2 now become functions of T_x, T_y, T_z in addition to Z since T_x, T_y, T_z cannot be taken to be constants now. This means the motion matrices change with time i.e. we should not expect a single motion matrix to describe the motion of a layer in the whole sequence. Rather we should process consecutive images in pairs and thus we get motion estimates evolving in time.
- Finally in this work we were more concerned with segmenting *pixels* undergoing similar motions. If the task is to segment the image into coherent *objects* that are moving similarly then we should utilize spatial coherence of the image in motion segmentation and use morphological operations as necessary to remove outliers etc.

(a)  (b) 

(c)  (d) 

(e) $\theta_1 = \begin{pmatrix} 0.833 & 0.051 & 0 \\ -0.2 & 1.064 & 0 \\ 31.226 & -12.757 & 1 \end{pmatrix}$ (f) $\theta_2 = \begin{pmatrix} 0.8 & 0.042 & 0 \\ -0.073 & 0.980 & 0 \\ -4.919 & 1.261 & 1 \end{pmatrix}$

(g)  (h) 

(i) $\theta_1 = \begin{pmatrix} 0.856 & -0.009 & 0 \\ -0.116 & 0.994 & 0 \\ 21.176 & 2.876 & 1 \end{pmatrix}$ (j) $\theta_2 = \begin{pmatrix} 0.8 & 0.038 & 0 \\ 0.031 & 0.959 & 0 \\ -12.548 & 3.702 & 1 \end{pmatrix}$

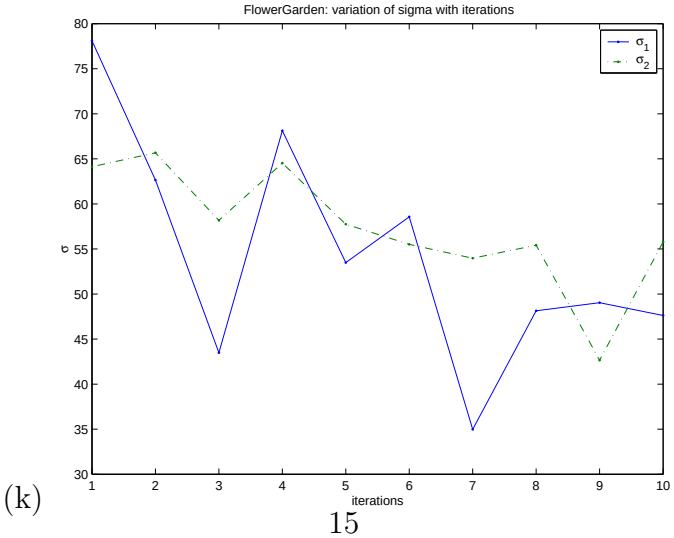
(k) 

Figure 5: Flower Garden (a), (b) Input images (c), (d) Motion segmentation obtained after 10 iterations (e), (f) Motion estimates after 10 iterations (g), (h) Motion segmentation at 7th iteration (i), (j) Motion estimates at beginning of 7th iteration (k) Plot of sigma vs. iterations.

(a)  (b) 

(c)  (d) 

(e) $\theta_1 = \begin{pmatrix} 1.082 & -0.024 & 0 \\ 0.165 & 0.966 & 0 \\ -6.413 & 3.323 & 1 \end{pmatrix}$ (f) $\theta_2 = \begin{pmatrix} 1.002 & 0 & 0 \\ -0.168 & 1.021 & 0 \\ 21.754 & -1.614 & 1 \end{pmatrix}$

(g)  (h) 

(i) $\theta_1 = \begin{pmatrix} 1.131 & -0.012 & 0 \\ 0.148 & 0.981 & 0 \\ -8.603 & 2.098 & 1 \end{pmatrix}$ (j) $\theta_2 = \begin{pmatrix} 1.03 & -0.04 & 0 \\ -0.2 & 1.009 & 0 \\ 19.427 & 7.194 & 1 \end{pmatrix}$

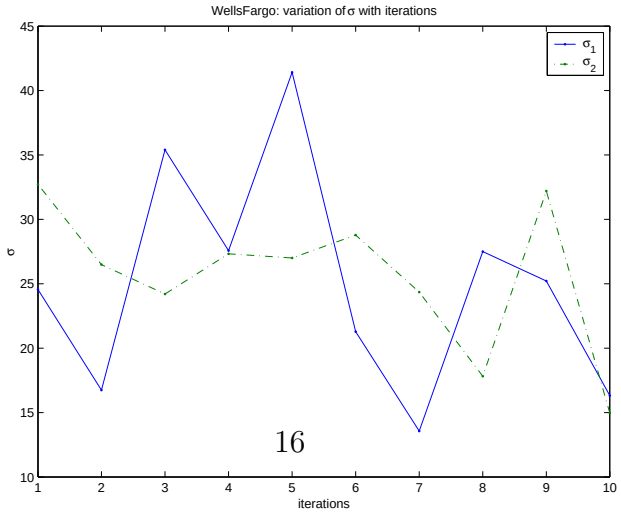
(k) 

Figure 6: Wells Fargo (a), (b) Input images (c), (d) Motion segmentation obtained after 10 iterations (e), (f) Motion estimates after 10 iterations (g), (h) Motion segmentation at 10th iteration (i), (j) Motion estimates at beginning of 10th iteration (k) Plot of sigma vs. iterations.

(a)  (b) 

(c)  (d) 

(e) $\theta_1 = \begin{pmatrix} 0.991 & 0.001 & 0 \\ 0.028 & 0.964 & 0 \\ 20.015 & 4.551 & 1 \end{pmatrix}$ (f) $\theta_2 = \begin{pmatrix} 0.977 & -0.023 & 0 \\ 0.147 & 1.005 & 0 \\ -3.619 & 2.918 & 1 \end{pmatrix}$

(g)  (h) 

(i) $\theta_1 = \begin{pmatrix} 0.988 & 0.006 & 0 \\ 0.016 & 1.007 & 0 \\ 22.794 & -0.745 & 1 \end{pmatrix}$ (j) $\theta_2 = \begin{pmatrix} 1.168 & -0.012 & 0 \\ 0.169 & 1.034 & 0 \\ -35.394 & 1.007 & 1 \end{pmatrix}$

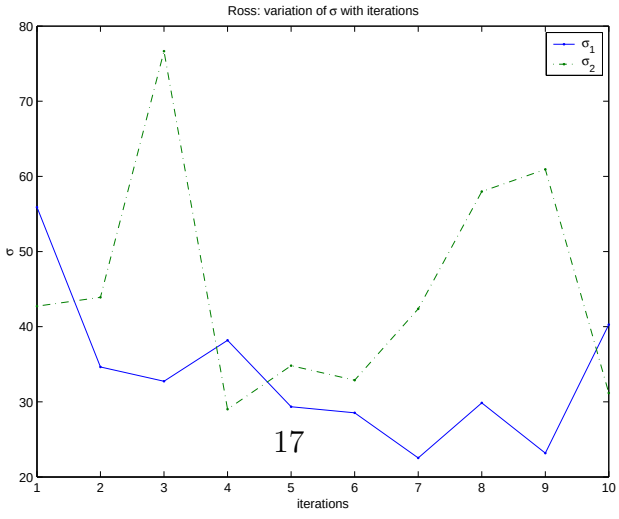
(k) 

Figure 7: Ross (a), (b) Input images (c), (d) Motion segmentation obtained after 10 iterations (e), (f) Motion estimates after 10 iterations (g), (h) Motion segmentation at 6th iteration (i), (j) Motion estimates at beginning of 6th iteration (k) Plot of sigma vs. iterations.

(a)  (b) 

(c)  (d) 

(e) $\theta_1 = \begin{pmatrix} 0.8 & 0.2 & 0 \\ -0.03 & 0.912 & 0 \\ 53.388 & -15.925 & 1 \end{pmatrix}$ (f) $\theta_2 = \begin{pmatrix} 1.12 & -0.043 & 0 \\ 0.2 & 1.061 & 0 \\ -17.957 & 0.572 & 1 \end{pmatrix}$

(g)  (h) 

(i) $\theta_1 = \begin{pmatrix} 0.8 & 0.176 & 0 \\ -0.028 & 0.878 & 0 \\ 53.207 & -12.584 & 1 \end{pmatrix}$ (j) $\theta_2 = \begin{pmatrix} 1.104 & 0.007 & 0 \\ 0.08 & 1.054 & 0 \\ -4.473 & -3.368 & 1 \end{pmatrix}$

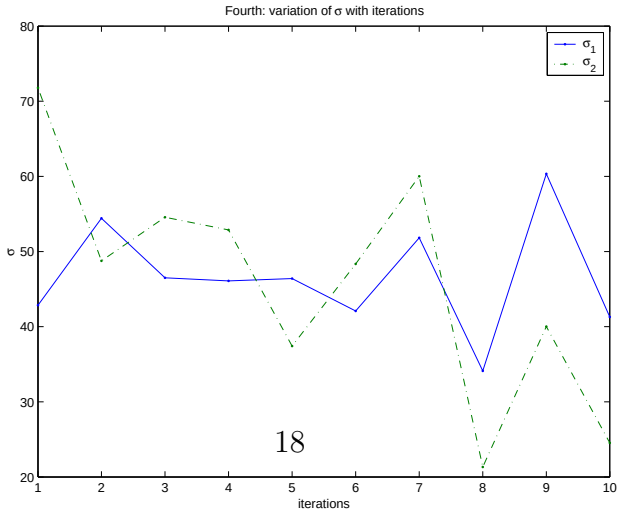
(k) 

Figure 8: Fourth (a), (b) Input images (c), (d) Motion segmentation obtained after 10 iterations (e), (f) Motion estimates after 10 iterations (g), (h) Motion segmentation at 7th iteration (i), (j) Motion estimates at beginning of 7th iteration (k) Plot of sigma vs. iterations.

6 Pseudocode

inputs: I_1, image 1 and I_2, image 2
  M, the number of layers
  θ_{init}, initial value of θ
  T, the number of iterations to perform

$K = 60; K_2 = 15; K_3 = 30; K_4 = 10;$
$\theta(1) \leftarrow \theta_{init}$;
halfsize($srchw$) $\leftarrow$ 32 for each layer;
for t = 1:T

  while (not all pixels in I_1 are processed) and
  (number of pixels for motion estimation in each layer $< K_4$)
    $p \leftarrow$ choose random pixel in I_1 taking care that the same pixel is not
      chosen multiple times;
    $z(p,t) \leftarrow$ motion segment the pixel as described in Section 2.1;
    if pixel passes feature and good match tests in equations (17, 18)
      add it to list of pixels available for motion estimation;
      compute weight factors in equation (14);
    endif
  end while

  if (all pixels in I_1 have been processed) and
    (number of pairs for motion estimation in each layer $< K_4$)
      increase K_3 to $K_3 + 10$;
      try to find more matches now;
      continue the process of trying to find more matches and iteratively
        increasing K_3 to compensate for ill conditioning of
        the problem until enough pairs in each layer are obtained
        or K_3 becomes equal to K;
  endif

  Compute $\sigma(t)$ and update search window size as in Section 2.2;
  $\theta(t+1) \leftarrow$ solve equation (16) subject to (19, 20);

end for

$z(T+1) \leftarrow$ dense motion segmentation based on $\theta(T+1)$;
output: $z(T+1)$ motion segmentation
    $\theta(T+1)$ motion estimation

References

[MSKS03] Y. Ma, S. Soatto, J. Kosecka, and S. S. Sastry. *An Invitation to 3-D Vision: From Images to Geometric Models*, chapter 3 equation 3.14. Springer-Verlag, 2003.

[Pan98] J. C. Pant. *Introduction to Optimization: Operations Research*, chapter 4. Jain Brothers, New Delhi, 1998.

Further Reading

- J. Y. A. Wang, and E. H. Adelson. *Representing Moving Images with Layers.* IEEE Transactions on Image Processing, Vol. 3, No. 5, September 1994, pp. 625–638.