

16-720 Assignment 5

Edward Hsiao (edhsiao@cmu.edu)

Due: Tuesday, November 17, 2009 at 11:59pm

1 Introduction

Tracking moving objects and estimating dominant camera motion are central to a vast number of computer vision tasks. In this assignment, you will design a simple system to detect moving objects from a short sequence¹ taken by an Unmanned Aerial Vehicle (UAV).

The short sequence you will be working with contains 50 frames and is included in the homework. You will notice that in this sequence both the camera and the people on the ground are moving, making object detection and tracking very difficult. Your goal throughout this assignment will be to detect the people on the ground for each frame of the video and track their movements.

2 Video Stabilization (55 points)

For a stationary camera, a common method for moving object detection is background subtraction in which each frame of the video is subtracted from a learned background model. When the camera is moving as well, things become much more difficult. Thus an initial step that most systems take is to stabilize the video (i.e. warp all frames to the viewpoint of the first frame), in essence, making the camera stationary.

2.1 Dominant Motion Estimation (40 points)

In this section, you will approximate the warping between two successive frames as an affine transformation. This assumption is reasonable if the majority of the pixels corresponding to stationary objects have small depth variation relative to their distance from the camera. Begin by implementing a tracker for affine motion using the equations for affine flow described in class.

To estimate dominant motion, the entire image $I(t+1)$ will serve as the “template” to be tracked for $I(t)$. Using the affine motion equations, you can recover the 6-vector of affine flow parameters iteratively and reconstruct the equivalent affine transformation matrix, M . The matrix M should map the homogeneous image coordinates of $I(t+1)$ to $I(t)$ according to $X_t = MX_{t+1}$ where $X = [x, y, 1]^T$. Each update of the affine parameters, Δp is computed via the least squares method using the pseudoinverse.

Note that unlike other template tracking problems in which the template to be tracked is small in comparison with the size of the image, the warped image of $I(t+1)$ will almost always not be contained fully in $I(t)$. Hence the matrix of image derivatives, A , and the temporal derivatives, I_t , must be computed only on the pixels in the region common to $I(t)$ and the warped version of $I(t+1)$ to be meaningful.

¹UCF-Lockheed UAV dataset: <http://server.cs.ucf.edu/~vision/aerial/index.html>

Implement the following function:

```
M = estimateDominantMotion(img, baseimg)
```

The function returns M , the 3×3 affine transformation matrix that maps `img` to `baseimg` for actual pixel coordinates, not normalized ones. Here you can think of `img` as the template which you wish to warp to the view of `baseimg` (i.e. `img` is $I(t+1)$ and `baseimg` is $I(t)$ in our above description). You must use affine optical flow for this section.

Implementation hints

- Convert your images to `double` and grayscale.
- Use a Gaussian derivative filter instead of pixel differences when computing the spatial derivatives. We have provided you with a function called `gaussianDerivative` to generate a derivative filter. You should only use the valid regions when calculating the optical flow.
- Scaling image coordinates will be crucial for this type of problem. You should scale coordinates to be between 0 and 1 when computing the A matrix. Be sure to undo the scaling whenever you need to use coordinates to actually reference pixels in the image. Also note that I_x and I_y will vary depending on your scaling, so be sure to compensate for that.
- Note that applying an affine transformation M_1 followed by an affine transformation M_2 is equivalent to a transformation by $M_{12} = M_2M_1$. It may be easier to keep track of the total accumulated warp (i.e. M_{12} here) as you iterate towards convergence and warp the original image using that cumulative warp at each iteration.
- The spatial derivatives remain the same between iterations when converging to the correct affine parameters, p . The only thing that changes will be I_t as you warp $I(t+1)$ more and more to better match $I(t)$.
- The matrix A contains one row for every pixel in the image. As discussed above, the temporal derivative, I_t should only be computed on those pixels of the warped $I(t+1)$ that overlap with $I(t)$. Thus, at a given iteration, you will need to ignore some rows of A corresponding to the pixels that do not overlap. Note that this does not require completely rebuilding A . Instead, you only need to keep track of which rows of A are currently active and index those rows when computing Δp .
- You will need to “unscale” M so that it is the affine transformation for the actual pixel coordinates and not the normalized ones.

2.2 Stabilization (15 points)

Now that you are able to compute the affine transformation between consecutive images, you can compute the affine transformation from each image to the first frame. However, if you implement the stabilization naively in a sequential manner, you will notice that your stabilization drifts as you go further in the sequence due to cumulative error in your affine registration. To address this, you will need to design a method to compensate for the drift.

Write the following function:

```
Ms = stabilize(sequence)
```

This function should take in a cell array `sequence` of full path filenames and return a $N \times 9$ matrix, Ms , where each row contains the affine transformation from that frame to the first frame. Since our sequence has 50 frames, $N = 50$ in our case. Each row will be of the form $[a, b, c, d, e, f, g, h, i]$ corresponding to the affine transformation M below.

$$M = \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix} \quad (1)$$

For example, the first row of your `Ms` should be `[1, 0, 0, 0, 1, 0, 0, 0, 1]` corresponding to the identity matrix.

Save the affine transformations `Ms` in `Ms.mat`. Also apply the affine transformations to your video sequence and save the results in a directory called `stabilized`. These images should look similar to Figure 1 where only regions common to frame 1 are shown. The better stabilization you have here, the easier and more accurate your object detection will be later. We have provided you with the TA's `Ms` for comparison in a file called `Ms_TA.mat`. You should only use the `Ms` in this file to complete the rest of the homework as a last resort.



Figure 1: Video stabilization: (left) Frame 1 (center) Frame 25 stabilized (right) Frame 50 stabilized

2.3 Extra Credit - Mosaic (+5 points)

Since you have calculated the affine transformation from every frame to the first frame in the previous section, you can create a mosaic with all the images in the sequence similar to Homework 1. Save your mosaic as `mosaic.jpg`.

3 Moving Object Detection (30 points)

Now that we have a stabilized video sequence, we can apply background subtraction to detect objects. The idea is that if we have a model of what the background looks like (i.e. without any moving objects), any significant difference between a video frame and the background can be considered a moving object.

3.1 Background Model (10 points)

In this section, you are free to use any method to learn a model of the background. Some popular approaches such as the mean, median, and mixture models were described in class. An example of a learned background is shown in Figure 2. The more accurate the background is, the easier it is to detect moving objects. If you are not getting satisfactory results, consider improving your stabilization.

Write the following function:

```
background = learnBackground(sequence, Ms)
```

The variable `background` can be anything you want from a 2D matrix specifying a static background, a struct if you have additional parameters, to a 3D matrix specifying a background for each frame...etc.

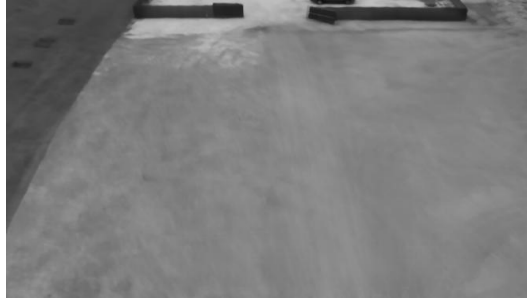


Figure 2: Example learned background

3.2 Object Detection (10 points)

Use the background you learned in the previous section and apply background subtraction on each frame of the video. You may use any morphology functions in Matlab such as `imdilate` and `imerode` to clean up your images. We have also provided a hysteresis thresholding function, `hysthresh` which may work better than straight thresholding of your difference image. The goal for this section is to have one detection for each object in the scene.

Write the following function:

```
detections = detectMovingObjects(sequence, Ms, background)
```

The function returns `detections`, an $N \times 1$ cell array, where each cell i corresponds to frame i of the sequence. Each cell contains a $K_i \times 2$ matrix of (x, y) coordinates for the K_i moving objects detected in that frame. The (x, y) coordinates should be the objects location on the *stabilized* video sequence. Save `detections` in `detections.mat`.

You will notice that the estimates of the moving pixels are a bit noisy. There are several reasons for this, ranging from poor image quality, changes in lighting with viewpoint, and compression related artifacts, to inaccurate affine registration and the choice of thresholds. We do not expect perfect results, after all, this is real world data!

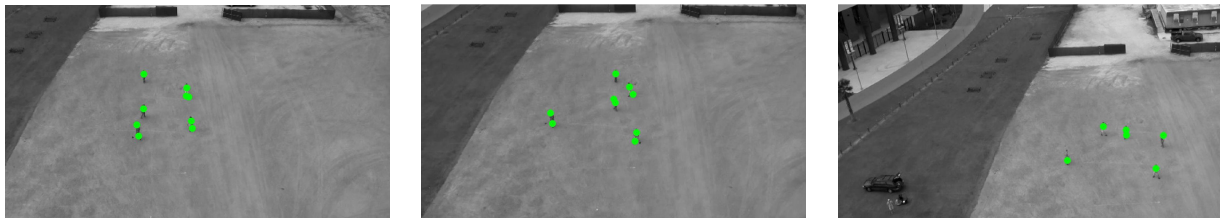


Figure 3: Object detections: (left) Frame 1 (center) Frame 25 (right) Frame 50

Your results should be plotted on the *original* video sequence and should look similar to those in Figure 3. Save your results in a directory called `detections`.

3.3 Tracking (10 points)

With your raw detections from the previous section, generate tracks for each of the people in the sequence (hint: look for simpler ways to use the detections to track the people instead of using a Lucas Kanade tracker). You will likely have multiple tracks for the same person, since your detections are not perfect.

Thus, you will need to aggregate those tracks together for your final result. The goal for this section is to have one track per person and should look similar to Figure 4. You will find it very difficult to complete this section if your images are not stabilized correctly or your detections are too noisy. Thus you will most likely need to fine tune your parameters from the previous sections.

Write the following function:

```
tracks = generateTracks(detections)
```

This function should return `tracks`, an $M \times 1$ cell array, where each cell corresponds to a particular person. Each cell contains an $N \times 2$ matrix corresponding to the (x, y) coordinates of the N frames in the sequence. Save `tracks` in `tracks.mat`. Also, on the first frame of the sequence, plot the track of each person in a different color, similar to Figure 4 and save it as `tracks.jpg`.

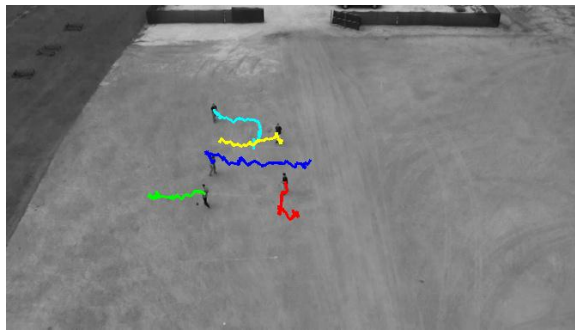


Figure 4: Example trajectories for the people in the sequence

4 Dichromatic Reflectance Theory (15 points)

Most computer vision algorithms today assume that all surfaces in the world are *Lambertian*, meaning that regardless of the viewing angle, the surfaces appear to be the same color. However, in the real world, many objects have a *specular* component. This means that under certain viewing and lighting angles, the surface will appear much brighter. Intuitively, if something is shiny, like a mirror, it has a strong specular component. For this homework, you will have to read Sections 2.1 and 3 of the following paper and answer some short questions:

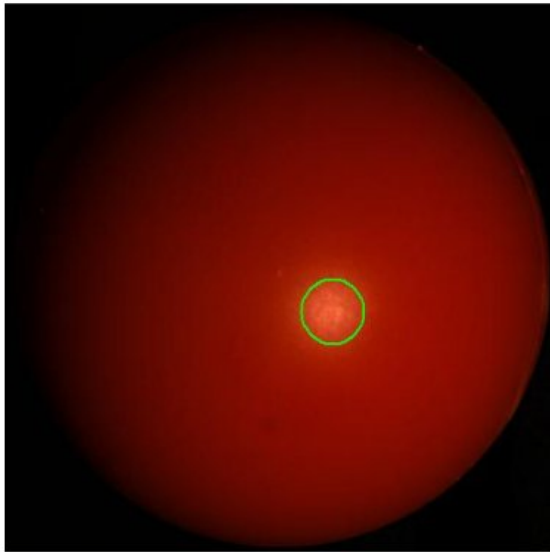
- [1] S. P. Mallick, T. E. Zickler, D. J. Kriegman, and P. N. Belhumeur. *Beyond Lambert: Reconstructing Specular Surfaces Using Color*. CVPR, 2005.

Given an input image, one can rotate the color space to recover the diffuse component of the image. Here, you should think of the RGB colors as vectors in a three dimensional space. For example if the pixel has values $R = 50$, $G = 100$ and $B = 150$, then this would correspond to the vector $(50, 100, 150)$. The main intuition behind this paper is that for real world objects in RGB space, the diffuse components often form a surface that is perpendicular to the specular component. An example with just red and green channels is shown in Figure 5. At first, it may seem odd that the ratio between red and green is so different between the diffuse and specular components.

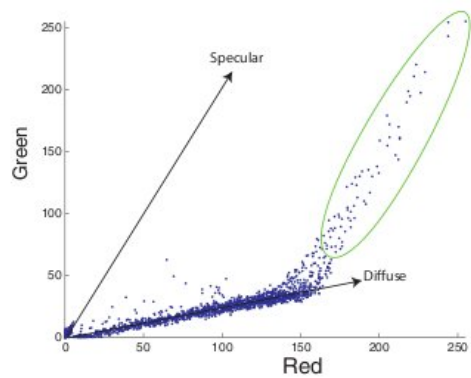
We can begin to understand this by looking at Equation 1 in the paper:

$$f(\lambda, \theta) = g_d(\lambda)f_d(\theta) + f_s(\theta) \quad (2)$$

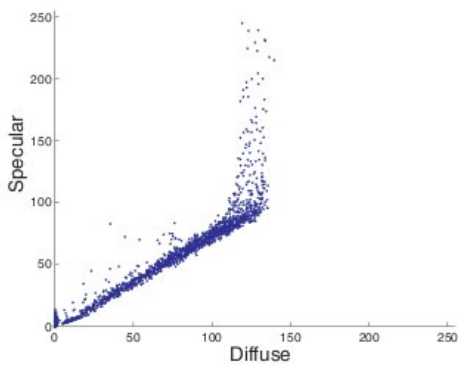
In this equation, the amount of light reflected toward the viewer is dependent on the wavelength of the light λ . In the paper's notation, θ is a vector consisting of four angles: two for the angle between the light source and the surface normal, and two for the angle between the viewing direction and the surface normal. The total brightness is the sum of the diffuse component $g_d(\lambda)f_d(\theta)$ and the specular component $f_s(\theta)$. According to this model, the specular component will reflect all wavelengths equally, while the diffuse component will reflect different percentages of different wavelengths of light. For example, if a red sphere is illuminated with white light, the diffuse component will reflect much more red light than green, so g_d would be much larger for red than green. On the other hand, the specular component will reflect all the wavelengths equally, so the specularity will be white.



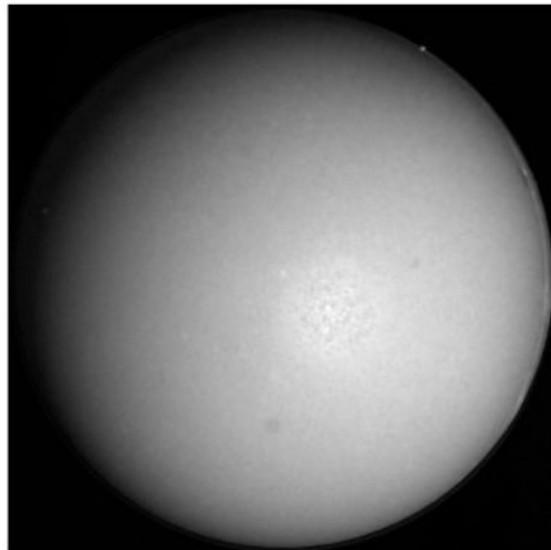
(a) An image of a shiny red sphere



(b) The red and green values for each pixel



(c) The color space has been rotated to align the specular component with the y-axis



(d) The diffuse component of the sphere, after the color space rotation

Figure 5: In (a) and (b), the green circle is drawn to show corresponding pixels. Not only are the specular pixels brighter, but they have a different proportion of red versus green. When we rotate the colorspace in (c), the pixel values in the x-dimension are not related to the specularity. We can then throw out the specular information, and get just the diffuse component in (d).

Q1 (5 points): For the red sphere example, how can you determine the color of the input light? (You do not need to write a mathematical formula.)

Q2 (5 points): For the red sphere example, the diffuse and specular components are not completely orthogonal. Why is this not important?

Q3 (5 points): What effect does the dot product between the surface normal \hat{n} and the angle of the incident light \hat{l} have on Equation 2 in the paper? Why is this important? ($D_k f_d(\theta)$ and $S_k f_s(\theta)$ are just a more thorough way of writing the light intensity due to the diffuse and specular components).

5 Submission

The following should be submitted electronically:

- Code for all the functions in the assignment. You should also submit any auxiliary code written to support these functions and other supporting `.mat` files. To help us grade, you must use the function names/arguments and the image filenames we have described. We will deduct points or return your work if you fail to adhere to our naming conventions. Note that helper functions are fine, but the main function calls should be identical to the prototypes we provided.
- `Ms.mat` - matrix of affine transformations from each frame to the first frame
- `stabilized` - directory containing the stabilized images for each frame in the sequence similar to Figure 1
- `background.jpg` - image of your learned background similar to Figure 2
- `detections.mat` - cell array of objects detected in the sequence. These (x, y) coordinates should be on the stabilized images.
- `detections` - directory containing the detections for each frame in the sequence similar to Figure 3. These should be on the original video sequence.
- `tracks.mat` - cell array of tracks for each person in the sequence
- `tracks.jpg` - image of your tracking results with each person's track plotted in a different color. This should look similar to Figure 4.
- Write up - A pdf or doc file describing the methods you used for video stabilization, background modeling, object detection and tracking. You should describe how you accounted for drift in the stabilization section and how you aggregated the tracks in the tracking section.
- Short answers to Q1-Q3. A few sentences each will suffice.
- (extra credit) - Code and mosaic of the video sequence `mosaic.jpg`.