Alexandria University
Faculty of Engineering
Computer and Systems Engineering
Department

CS 4E3 Computer Vision
Assigned: Saturday, March 6, 2020
Due: Thursday, March 26, 2020

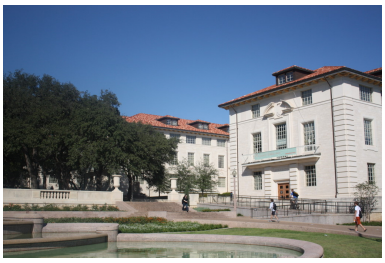
Assignment 2

1 Part1: Image Mosaics

In this part of the assignment, you will implement an image stitcher that uses image warping and homographies to automatically create an image mosaic. We will focus on the case where we have two input images that should form the mosaic, where we warp one image into the plane of the second image and display the combined views. This problem will give some practice manipulating homogeneous coordinates, computing homography matrices, and performing image warps. For simplicity, we will specify corresponding pairs of points manually using mouse clicks.

Figure 1: Image Mosaics

(a) First View



(b) Second View

(c) Final Output

1.1 Getting Correspondences

Plot the input images and collect mouse click positions. The results will be sensitive to the accuracy of the corresponding points, when providing clicks, choose distinctive points in the image that appear in both views.

1.2 Compute the Homography Parameters

Write a function that takes a set of corresponding image points and computes the associated 3×3 homography matrix H. This matrix transforms any point p in one view to its corresponding homogeneous coordinates in the second view, p', such that $p' = Hp$. Note that p and p' are both $3D$ points in homogeneous coordinates. The function should take a list of $n \geq 4$ pairs of corresponding points from the two views, where each point is specified with its $2D$ image coordinates.

We can set up a solution using a system of linear equations $Ax = b$, where the 8 unknowns of H are stacked into an 8-vector x , the $2n$-vector b contains image points from one view, and

the $2n \times 8$ matrix A is filled appropriately so that the full system gives us $\lambda p' = Hp$. There are only 8 unknowns in H because we set $H_{3,3} = 1$. Solve for the unknown homography matrix parameters.

Verify that the homography matrix your function computes is correct by mapping the clicked image points from one view to the other, and displaying them on top of each respective image. Be sure to handle homogenous and non-homogenous coordinates correctly.

1.3 Warping Between Image Planes

Write a function that can take the recovered homography matrix and an image, and return a new image that is the warp of the input image using H . Since the transformed coordinates will typically be sub-pixel values, you will need to sample the pixel values from nearby pixels. For color images, warp each RGB channel separately and then stack together to form the output.

To avoid holes in the output, use an inverse warp. Warp the points from the source image into the reference frame of the destination, and compute the bounding box in that new reference frame. Then sample all points in that destination bounding box from the proper coordinates in the source image (linear interpolation). Note that transforming all the points will generate an image of a different shape/dimensions than the original input.

1.4 Create the output mosaic

Once we have the source image warped into the destination images frame of reference, we can create a merged image showing the mosaic. Create a new image large enough to hold both (registered) views; overlay one view onto the other, simply leaving it black wherever no data is available. Do not worry about artifacts that result at the boundaries.

2 Part2: Stereo Vision

In this part of the assignment you will implement and test some simple stereo algorithms. In each case you will take two images I_l and I_r (a left and a right image) and compute the horizontal disparity (ie., shift) of pixels along each scanline. This is the so-called baseline stereo case, where the images are taken with a forward-facing camera, and the translation between cameras is along the horizontal axis.

2.1 Block Matching

To get the disparity value at each point in the left image, you will search over a range disparities, and compare the windows using two different metrics: Sum of Absolute Differences (SAD) and

Sum of Squared Differences. Do this for windows of size w where $w = 1$, 5 and 9. The disparity, d.

2.2 Dynamic programming

Consider two scanlines $I_l(i)$ and $I_r(j)$. Pixels in each scanline may be matched, or skipped (considered to be occluded in either the left or right image). Let d_{ij} be the cost associated with matching pixel $I_l(i)$ with pixel $I_r(j)$. Here we consider a squared error measure between pixels given by:

$$d_{ij} = \frac{(I_l(i) - I_r(j))^2}{\sigma^2}$$

where σ is some measure of pixel noise. The cost of skipping a pixel (in either scanline) is given by a constant c_0. For the experiments here we will use $\sigma = 2$ and $c_0 = 1$. Given these costs, we can compute the optimal (minimal cost) alignment of two scanlines recursively as follows:

1. $D(1,1) = d_{11}$
2. $D(i,j) = min(D(i1, j1) + d_{ij}, D(i1, j) + c_0, D(i, j1) + c_0)$

The intermediate values are stored in an N-by-N matrix, D. The total cost of matching two scanlines is $D(N,N)$. Note that this assumes the lines are matched at both ends (and hence have zero disparity there). This is a reasonable approximation provided the images are large relative to the disparity shift. Given D we find the optimal alignment by backtracking. In particular, starting at $(i,j) = (N,N)$, we choose the minimum value of D from $(i-1, j-1), (i-1, j), (i, j-1)$. Selecting $(i-1, j)$ corresponds to skipping a pixel in I_l (a unit increase in disparity), while selecting $(i, j-1)$ corresponds to skipping a pixel in I_r (a unit decrease in disparity). Selecting $(i-1, j-1)$ matches pixels (i,j), and therefore leaves disparity unchanged. Beginning with zero disparity, we can work backwards from (N,N), tallying the disparity until we reach $(1,1)$.

A good way to interpret your solution is to plot the alignment found for single scan line. To display the alignment plot a graph of I_l (horizontal) vs I_r (vertical). Begin at $D(N,N)$ and work backwards to find the best path. If a pixel in I_l is skipped, draw a horizontal line. If a pixel in I_r is skipped, draw a vertical line. Otherwise, the pixels are matched, and you draw a diagonal line. The plot should end at $(1,1)$.

2.3 Disparity Computation

Repeat the process explained above fore each row of the image and compute the disparity maps for image pairs.

Good Luck