Mohamed Sondo Prof Zhu

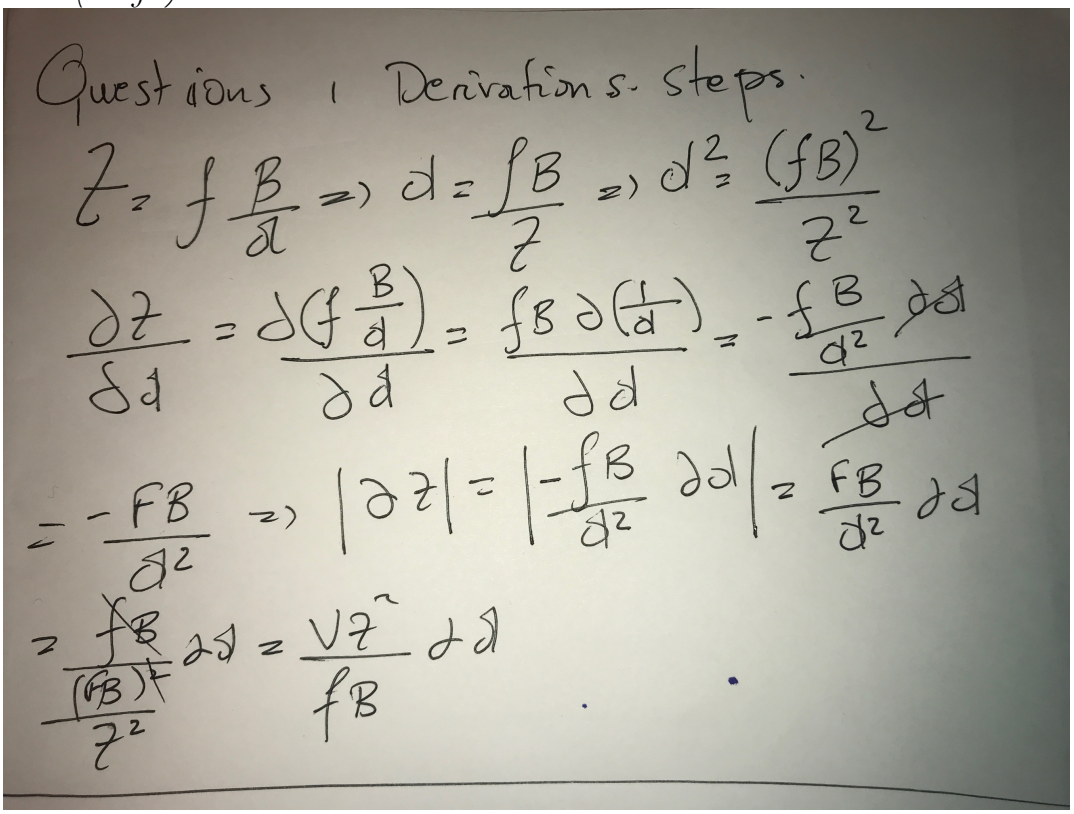
Csc 471 0 11/21/17

Hw 4

1. (Stereo- **30 points**) Estimate the accuracy of the simple stereo system (Z = f B/d, Figure 3 in the lecture notes of stereo vision) assuming that the only source of noise is the localization of corresponding points in the two images, which means the error in estimating d. Please derive (10 points) and discuss (20 points) the dependence of the error in depth estimation of a 3D point as a function of **(1) the baseline width B, (2) the focal length f, (3) stereo matching error, and (4) the depth of the 3D point, Z**.

To determine the position of a point in a disparity image, we can use the equation
$Z = fB/d$
- o Z = the depth, or distance along the camera z-axis
- o f = the focal length
- o B = the baseline width between the two cameras
- o d = the disparity, or distance between two projected points

To determine the depth error, we can calculate the partial derivatives of Z with respect to the variable of uncertainty. Assuming the only uncertainty is in the disparity d, the depth error is equal to
$\partial Z = (Z2/fB)\, \partial d$



If the uncertainty is in the baseline width, the depth error with respect to the baseline is equal to
$\partial Z = (f/d)\,\partial B$
Finally, if the uncertainty is in the focal length, the depth error with respect to the focal length is equal to
$\partial Z = (B/d)\,\partial f$

- Depth error is inversely proportional to the baseline width, meaning a larger baseline width will provide better depth accuracy but a smaller field of view (FOV)
- Depth error is inversely proportional to the focal length, meaning a larger focal length will provide a better depth accuracy but a smaller FOV
- Depth error is proportional to the square of the depth, meaning the nearer a point is, the greater the accuracy (*ie.* the smaller the uncertainty).

2-(Motion- **40 points**) Could you obtain 3D information of a scene by viewing the scene by a camera rotating around its optical center (10 points)? Discuss why or why not(10 points). What about translating the camera along the direction of its optical axis (10 points)? Explain. (10 points)

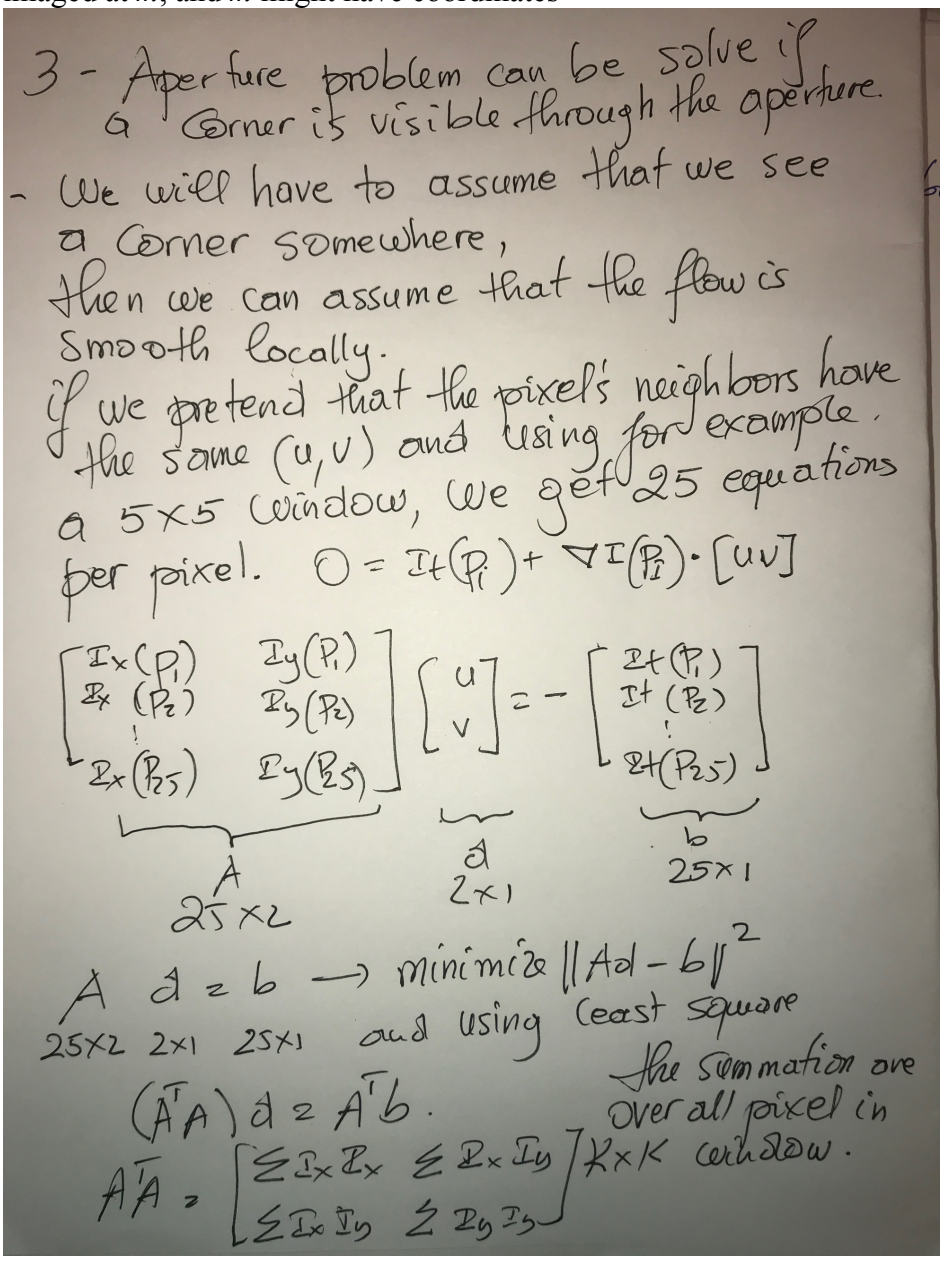
$$\begin{pmatrix} v_x \\ v_y \end{pmatrix} = \frac{1}{f}\begin{pmatrix} xy & -(x^2+f^2) & fy \\ y^2+f^2 & -xy & -fx \end{pmatrix}\begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix}$$

The motion field equation under rotation around the camera's optical center can be written as Since Z is not included in the equation, no 3D information is carried. The disparity between any two images under this rotation is 0. In order to be able to extract any 3D information, some translational operation must be applied instead. If we move the camera along its optical axis however, 3D information can be obtained because translational images can be captured at different time frames

There would be no depth information because we do not have 2 reference points to compare. For translation along the optical axis, there is no depth information. This is because you are simply changing the focal length if you move the camera forward or backwards. You need to have some distance in the X or Y direction in order to capture the 3D information

3. (Motion- 10 points) Explain that the aperture problem can be solved if a corner is visible through the aperture.

Points, as entities in their own right, will be denoted in italics. When such points are expressed in Euclidean coordinates, we will use bold notation, and when they are expressed in projective coordinates, they will be bold with a tilde. Thus a point *M* in three space might be imaged at *m*, and *m* might have coordinates



The corner could help us see the decoction of whole image It might appear to be going in a certain direction but we cannot be sure. For example, an edge may appear to be moving diagonally, but it will be moving up and down.

4. (Stereo Programming - **20 points + 20 bonus points**) Use the image pair (Image 1, Image 2) for the following exercises.

For the fundamental matrix, I generated the A matrix and used SVD of A. Then I created the

Fundamental Matrix using the A matrix. The epipolar line graph is the one shown below

Algorithm Summary

```
read in L and R images and if necessary, convert both to greyscale
  loop for each pixel in the left image
      generate matrix of left scan window for the current pixel
      calculate boundaries for the large right support window
      generate matrix of large right support window
      begin loop for each pixel in the large right support window
          get matrix of scan window size for the current pixel in support window
          calculate disparity by performing sum of squared differences
          if the disparity for the current right scan window is closer to 0 then
          note disparity and x value within the right support window
      end loop
      calculate the vector from the current row point to closest disparity match
      map the value between 0-255 for display purposes
      add value to array of disparity map array
  end loop
  write image out
end
```

Algorithm details

Let SampleImage contain the image we are sampling from and let Image be the mostly empty image that we want to fill in (if synthesizing from scratch, it should contain a 3-by-3 seed in the center randomly taken from SampleImage, for constrained synthesis it should contain all the known pixels). WindowSize, the size of the neighborhood window, is the only user-settable parameter. The main portion of the algorithm is presented below.

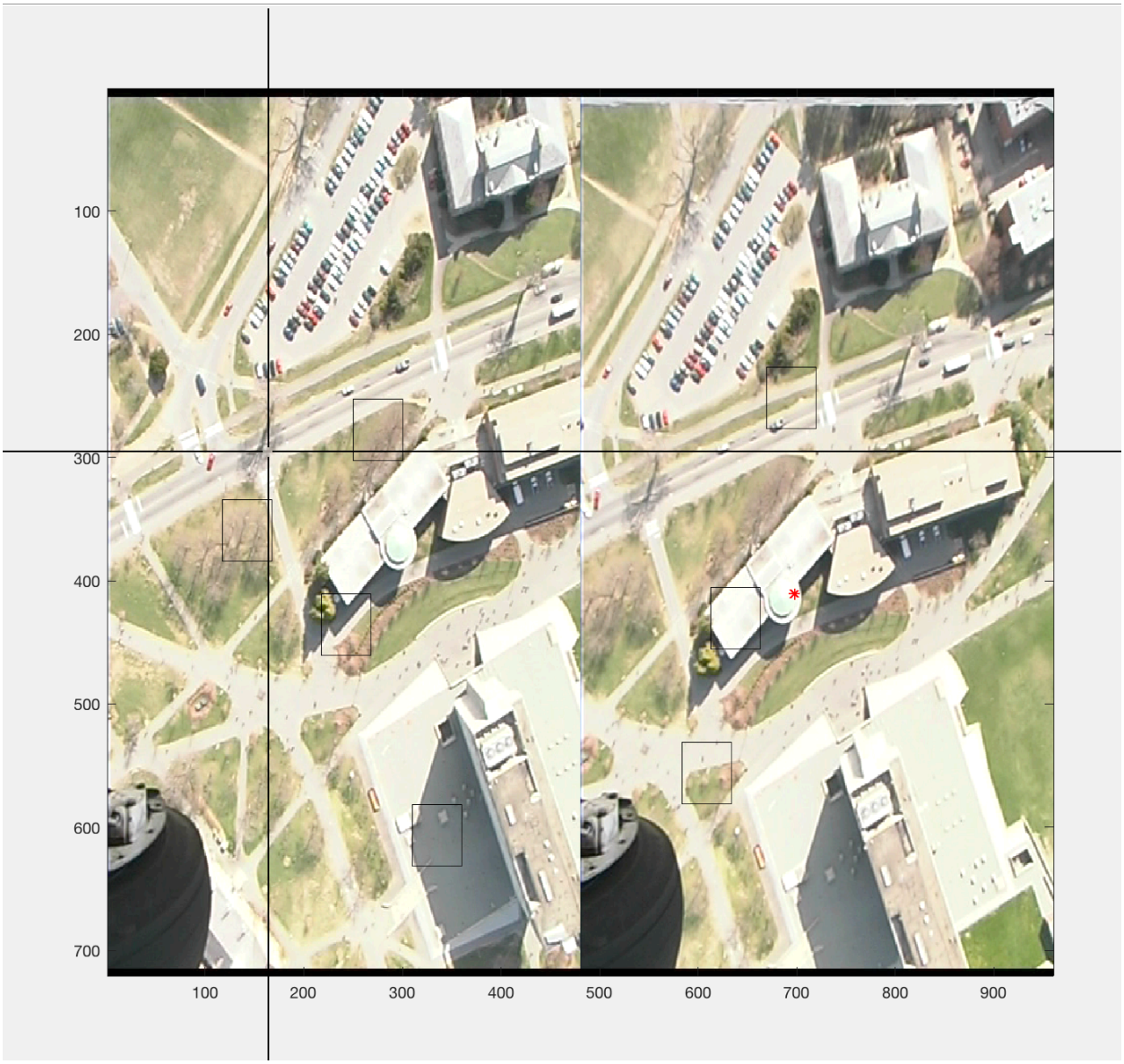
```
function GrowImage(SampleImage,Image,WindowSize)
while Image not filled do
progress = 0
PixelList = GetUnfilledNeighbors(Image)

foreach Pixel in PixelList do
Template = GetNeighborhoodWindow(Pixel)

BestMatches = FindMatches(Template, SampleImage) BestMatch = RandomPick(BestMatches)
Pixel.value = BestMatch.value
end

return Image
end
```

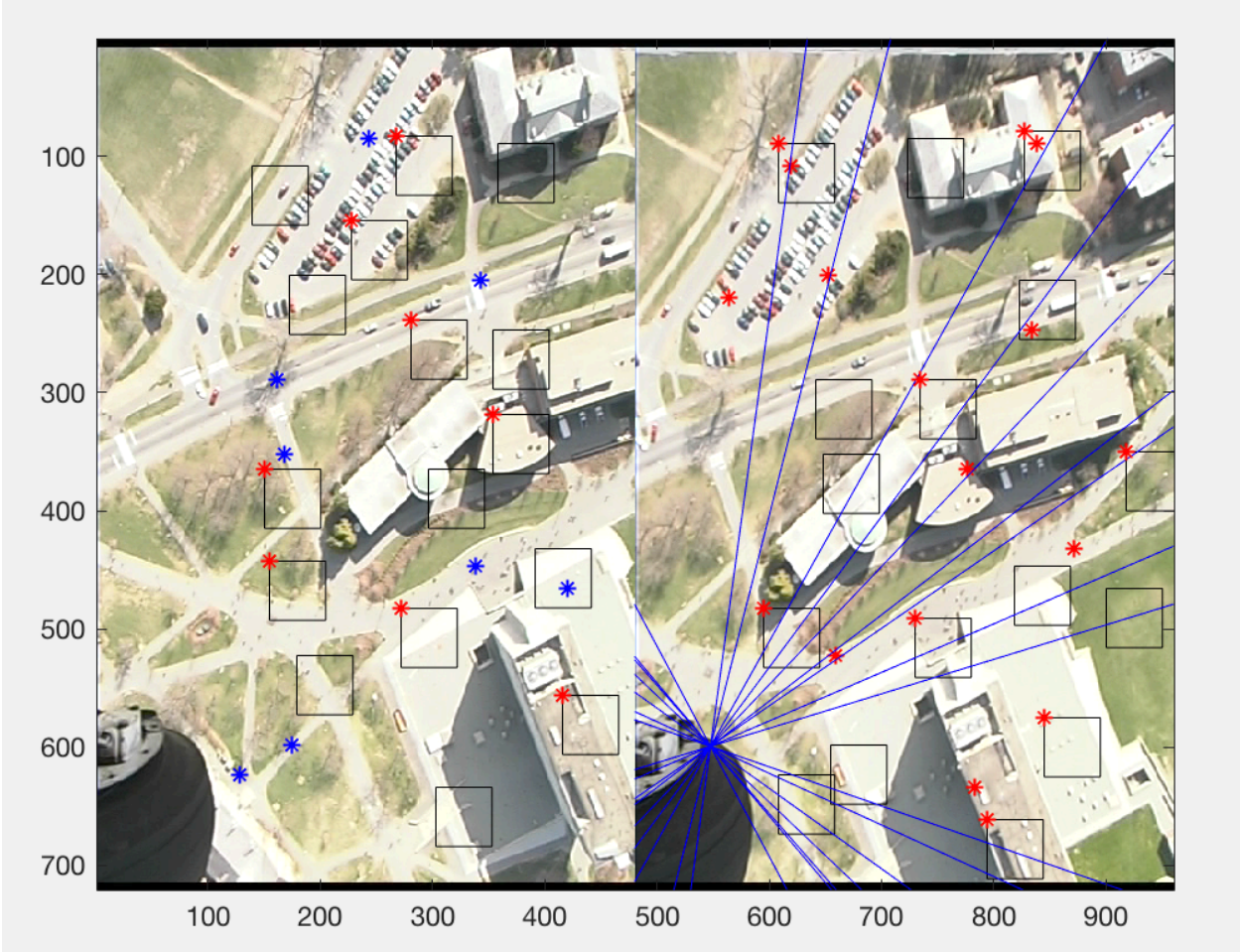
Function GetUnfilledNeighbors() returns a list of all unfilled pixels that have filled pixels as their neighbors (the image is subtracted from its morphological dilation). The list is randomly permuted and then sorted by decreasing number of filled neighbor pixels. GetNeigborhoodWindow() returns a window of size WindowSize around a given pixel.

RandomPick() picks an element randomly from the list. FindMatches() is as follows:

```
function FindMatches(Template,SampleImage)
ValidMask = 1s where Template is filled, 0s otherwise
TotWeight = sum i,j ValidMask(i,j)
for i,j do
for ii,jj do
dist = (Template(ii,jj)-SampleImage(i-ii,j-jj))^2
SSD(i,j) = SSD(i,j) + dist*ValidMask(ii,jj)
end
SSD(i,j) = SSD(i,j) / TotWeight
end
PixelList = all pixels (i,j) where SSD(i,j) <= min(SSD)*(1+ErrThreshold)
return PixelList
end
```

In our implementation the constant were set as follows: ErrThreshold = 0.1. Pixel values are in the range of 0 to 1.


100
200
300
400
500
600
700
100
200
300
400
500
600
700
800
900


100
200
300
400
500
600
700
100
200
300
400
500
600
700
800
900