6.869 Advances in Computer Vision
Andrew Moran
Collaborators: None

Oct 9, 2014 (1:00pm)
Problem Set 5

Problem Set 5

Resources with corresponding images and code are on Stellar under `andrewmo@mit.edu`. The files are in the `pset5.zip` folder. To reproduce the below figures, run `Prob1.m` and `pset5_2main.m`.

Problem 5.1

In this problem, I implemented the belief propagation algorithm for contstructing dense stereo correspondence. This utilized disparities of up to L=10 pixels. Below are the steps taken to achieve the desired results.

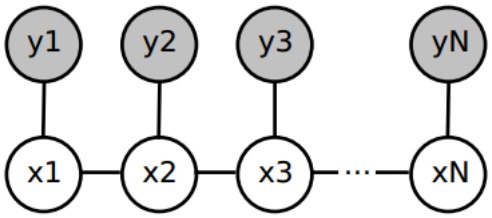
(A) The data term is represented as $\phi(x_i|y_i^l, y_i^r) \propto exp(-\lambda|y_i^l - y_{i-x_i}^r|)$

Code: `D(y,x,l) = exp(-lambda*abs(y_l - y_r));`

(B) The smoothness term is represented as $\psi(x_i, x_{i+1}) \propto exp(-|x_i - x_{i+1}|)$

Code: `P(l1,l2) = exp(-abs(dx1-dx2));`

(C) Below is the hidden Markov model used in this assignment.



The gray nodes are observed pixels, and the white nodes are the disparity that we wish to infer. Messages from node j to node i were computed as follows: $m_{ji}(x_i) = \sum_{x_j} \psi_{ij}(x_i, x_j) \prod_{k \in \eta(j)/i} m_{kj}(x_j)$ It was key to note that the message from the observed state was the data term. As a result, the generic form of calculting messages in code is shown below:

Code: `P * [D(y,x,:) .* msgs_sumprod(y,x,DIRECTION,:)];`

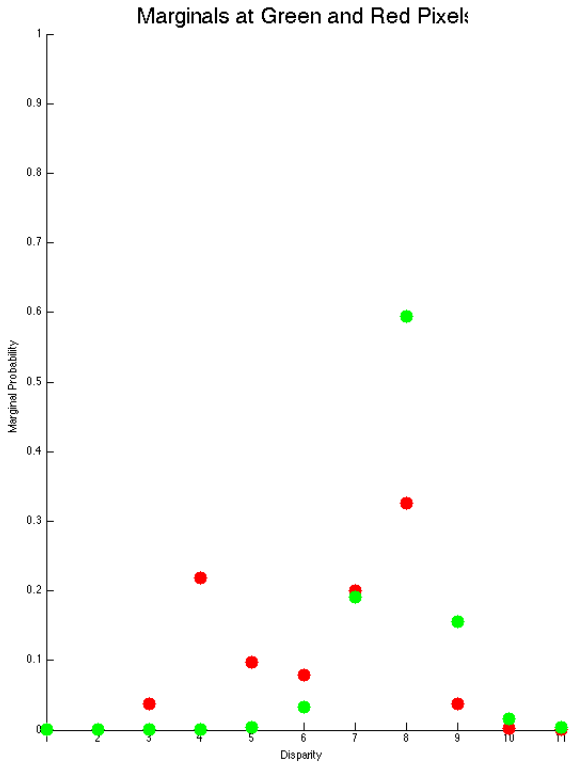
(D) We cane find the marginal probability by using the messages computed earlier as follows:

$$p(x|y^l, y^r) = \frac{1}{Z} \prod_{i=1}^{N} \phi(x_i|y_i^l, y_i^r)\psi(x_i, x_{i+1}) = \prod_{j \in \eta i} m_{ji}(x_i)$$

In summary, each marginal is computed using incoming messages from the top, left, and right neighboring nodes.

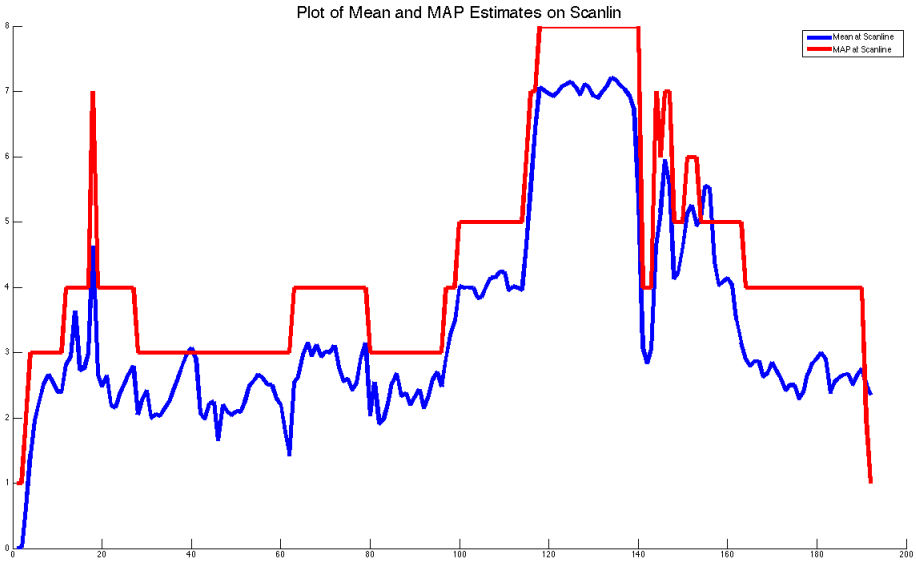
(E) Computing the mean was finding the expectation for each pixel using both the disparities and their marginal probabilities. This was calculated as $\sum_l lP(x_i = l)$.

(F) Below are the plots for the marginals of the green and red pixel.

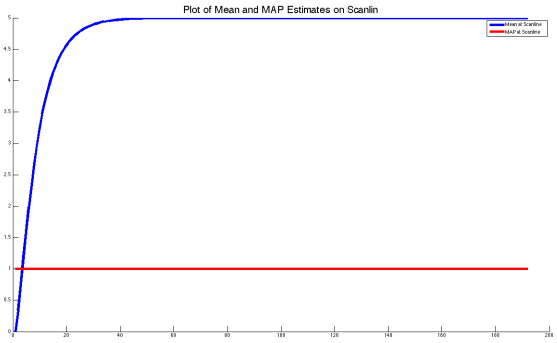


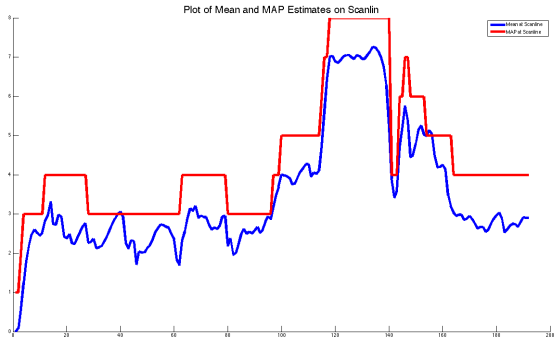
The marginals at an edge pixel has very high probability since that's where the largest change in data occurs. The green pixel is further from an edge, put when the disparity is large enough that it reaches an edge, the probability marginal has a significant spike at that point. On the otherhand, the red pixel is on an edge. Therefore small disparities either to the right or left of the pixel will show noticeable spikes in its left and right marginals (however, not as large).

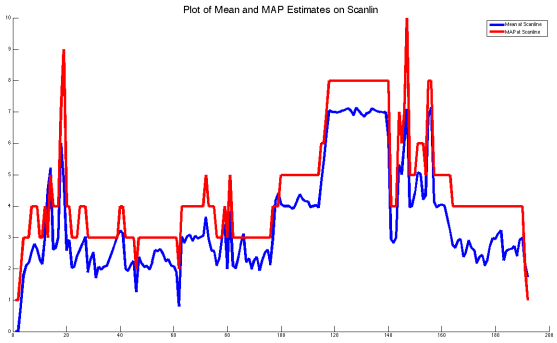
(G) Mean and MAP estimates of the disparity for $\lambda = .2$ for scanline 61 (MAP explained in (j)):

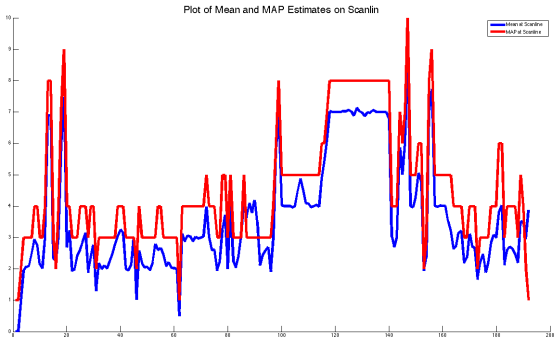


(H) Mean and MAP estimates of the disparity for $\lambda = 0, .1, .5, 1$ respectively:





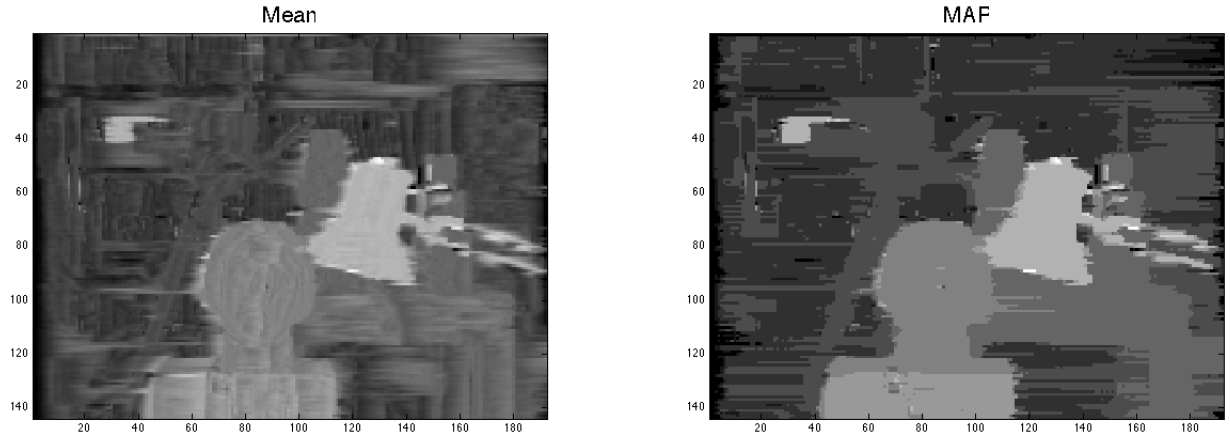




The top two plots show that decreasing λ generates a more uniform mean estimate to the dis-

parity, where as the bottom two plots show that increasing λ generates a mean estimate that has more spikes and less uniform. This is because λ determines the weight we place on the difference in disparities. The higher λ, the stronger the weights are applied, and therefore the more likley the results vary.

(I) Mean and MAP disparity as an image:

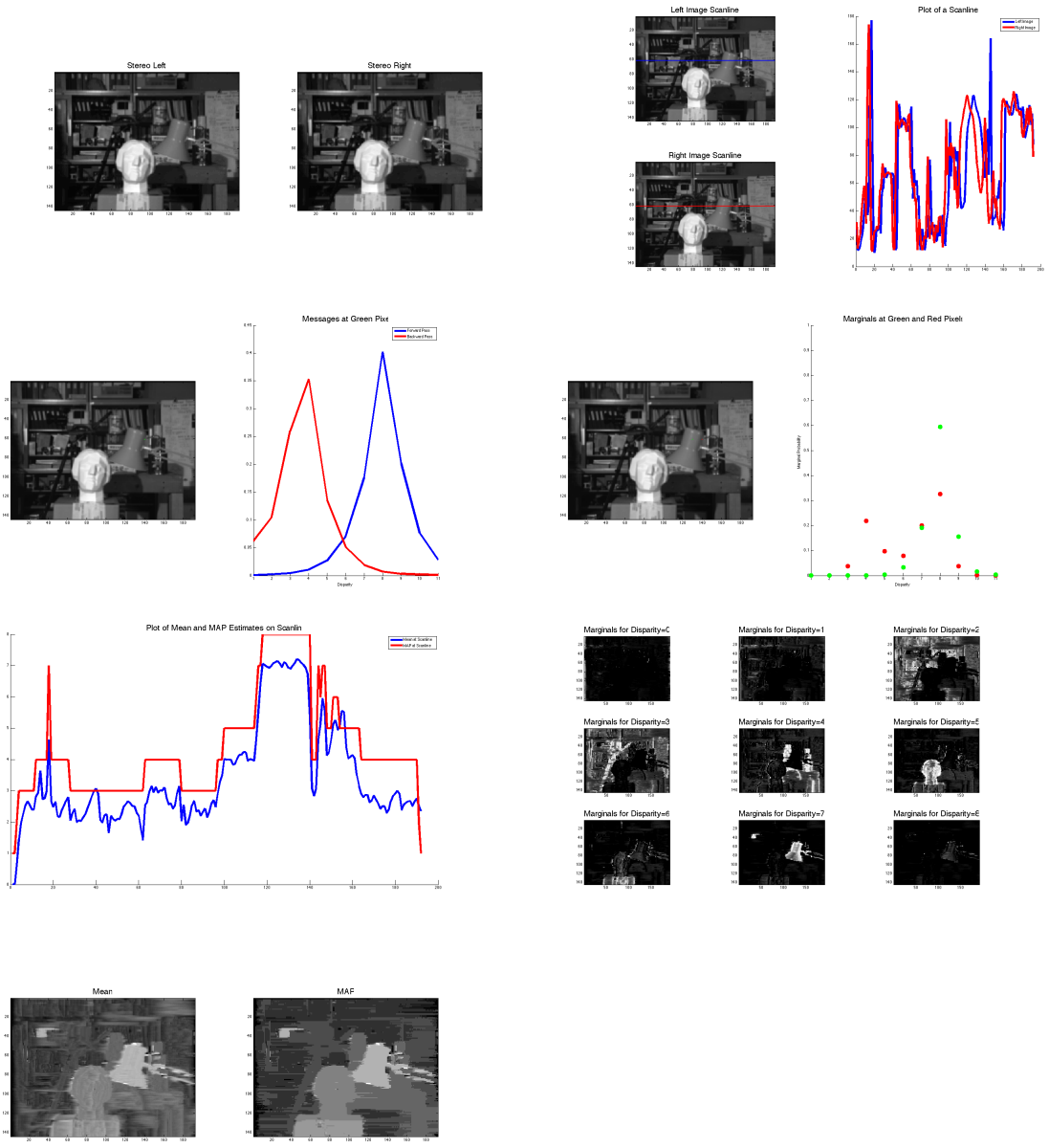


(J) Calculating the MAP estimate required an update to the message update equation. Instead of doing a sum, do a max operation:

$$m_{p\to q}(x_q) = max_{x_p}\phi(x_p)\psi(x_p, x_q) \prod_{s\in \eta p/q} m_{s\to p}(x)$$

Then computing the MAP estimate is the `arg_max` index out of the largest marginal probabilities, given you the disparity that produces the max marginal. From part G, we see that there is a difference bewteen the Mean and MAP estimates. The MAP estimate tends to have a higher average and more smooth compared to the Mean estimate. Also, the MAP is more step-like where as the Mean is more sporadic.

Visualization: For convenience, this is a complete small version of the ouput for `Prob1.m`



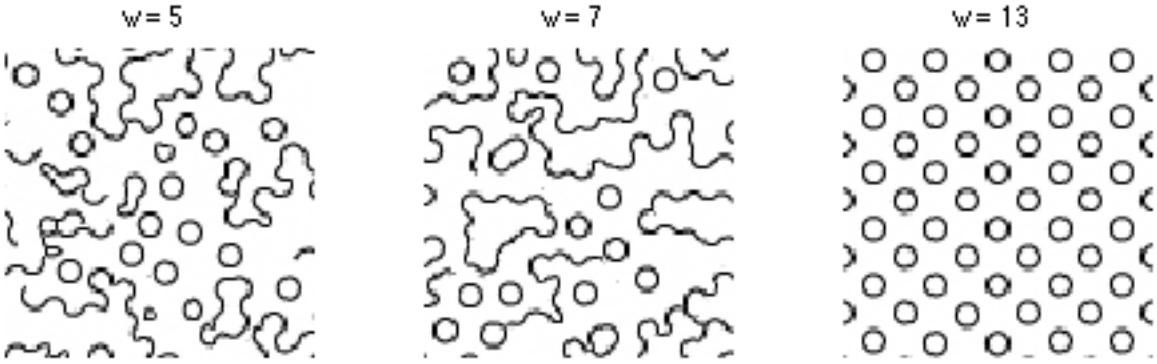
Problem 5.2

In this problem, I implemented the Efros and Leung algorithm for texture synthesis. I followed pseudocode provided at `http://graphics.cs.cmu.edu/people/efros/research/NPS/alg.htr` It is repeated below for convenience.

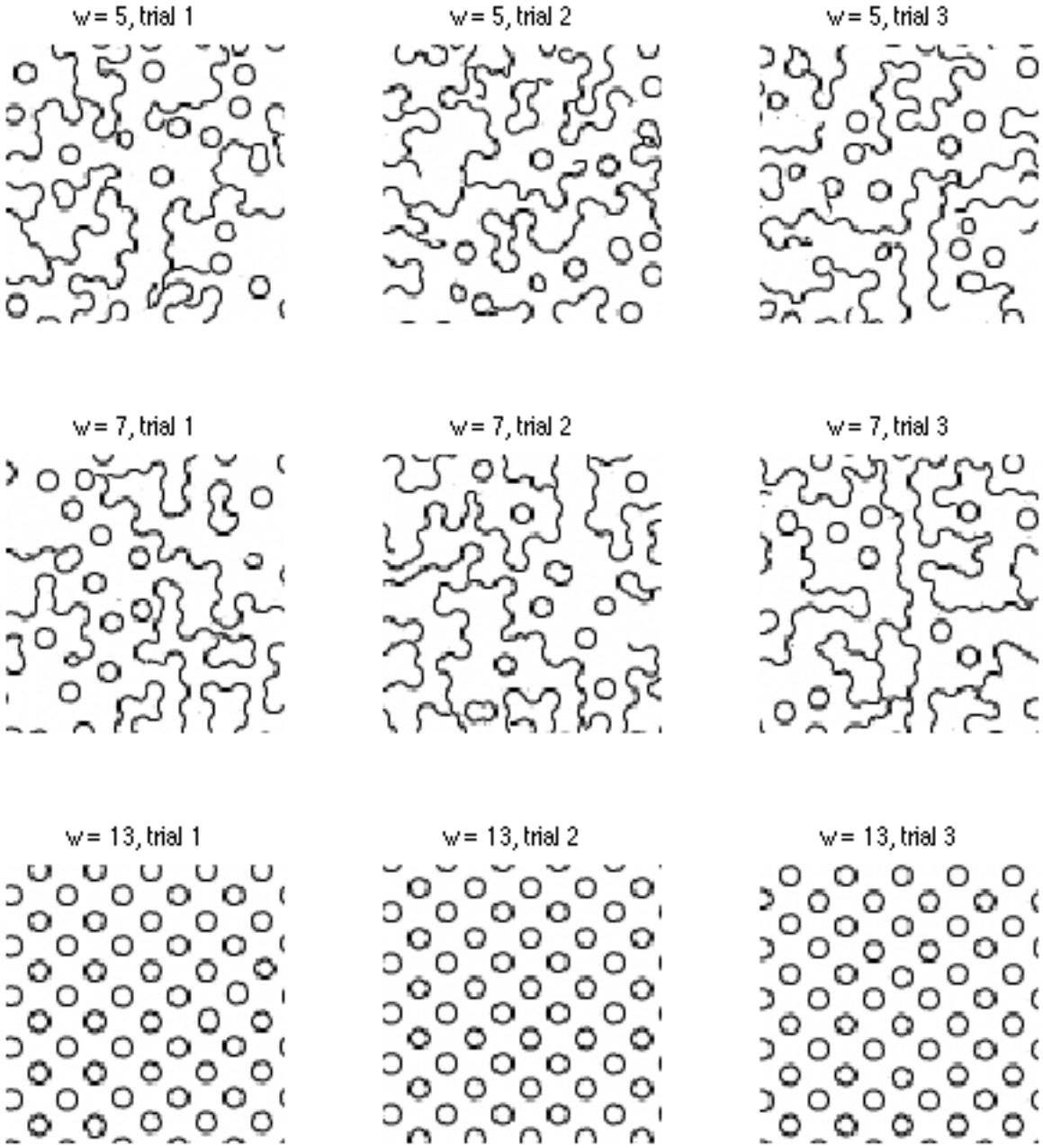
```
function GrowImage(SampleImage,Image,WindowSize)
 while Image not filled do
   progress = 0
   PixelList = GetUnfilledNeighbors(Image)
   foreach Pixel in PixelList do
     Template = GetNeighborhoodWindow(Pixel)
     BestMatches = FindMatches(Template, SampleImage)
     BestMatch = RandomPick(BestMatches)
     if (BestMatch.error < MaxErrThreshold) then
       Pixel.value = BestMatch.value
       progress = 1
     end
   end
   if progress == 0
     then MaxErrThreshold = MaxErrThreshold * 1.1
 end
 return Image
end
```

Although I have a slightly different implementation, the two main functions that was needed to be constructed was `SynthTexture` and a subroutine called `FindMatches`.

Below are the results of running the texture `rings.jpg` (shown above) using window widths $w = 5, 7, 13, s = [100, 100]$ and an initial starting seed of $(x, y) = (4, 32)$.



Here we see that the performance increases as the window size gets larger. When $w = 13$, the texture looks like the original. For smaller window sizes, only few of the dots are formed. Below shows the results iterating over a particular window size with the same starting seed.



We see for the smaller window sizes, running the algorithm multiple times with the same

starting seeds had different results, this is because as the texture is getting generated, the source texture is contstantly estimating each pixel based on the prior estimated pixel. Therefore, as the algorithm is incrementally adding pixels, it has to decide whether to complete an edge or not. Since the window size is smaller, the adjusted rings texture has edges on the borders. On the otherhand, the results for $w = 13$ look similar. Perhaps this is because the window is large enough that it can be repeated based on the borders of the original source texture and not estimating whether to continue/end on edge of a ring.