CV - Assignment 4

Authors: Ronair,srriyer,ssdevadi

Part1

Steps to run the program

- ./render part1/birds/view1.png part1/birds/disp1.png 7.5
- the last argument is optional, it signifies the normalization parameter (having a default value of 10)

Idea:

1. Normalizing the depth map:

First, we calculate the normalized depth map to make each pixel between 0 to ‘2 times normalization_param’. We accept this parameter as the third argument, the default value is 10.

2. Shifting the image:

We shift each pixel to the left. We start shifting from the deepest pixel to the ones that are closest to the camera. The amount of shift is proportional to its depth (in the normalized depth map).

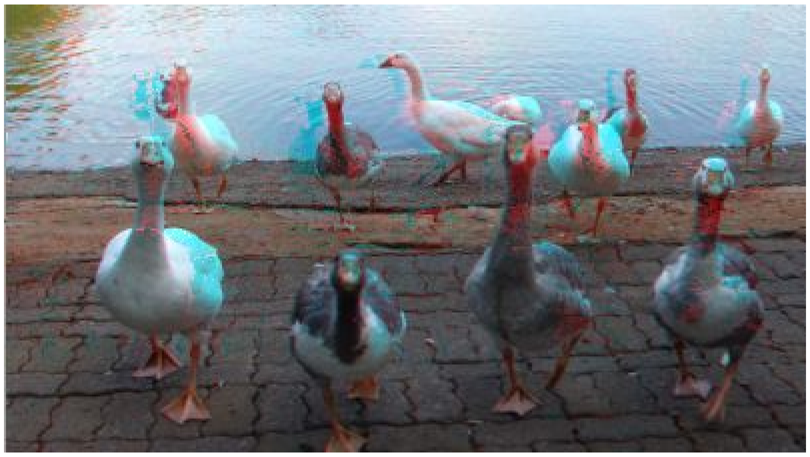
3. Handling blank pixels:

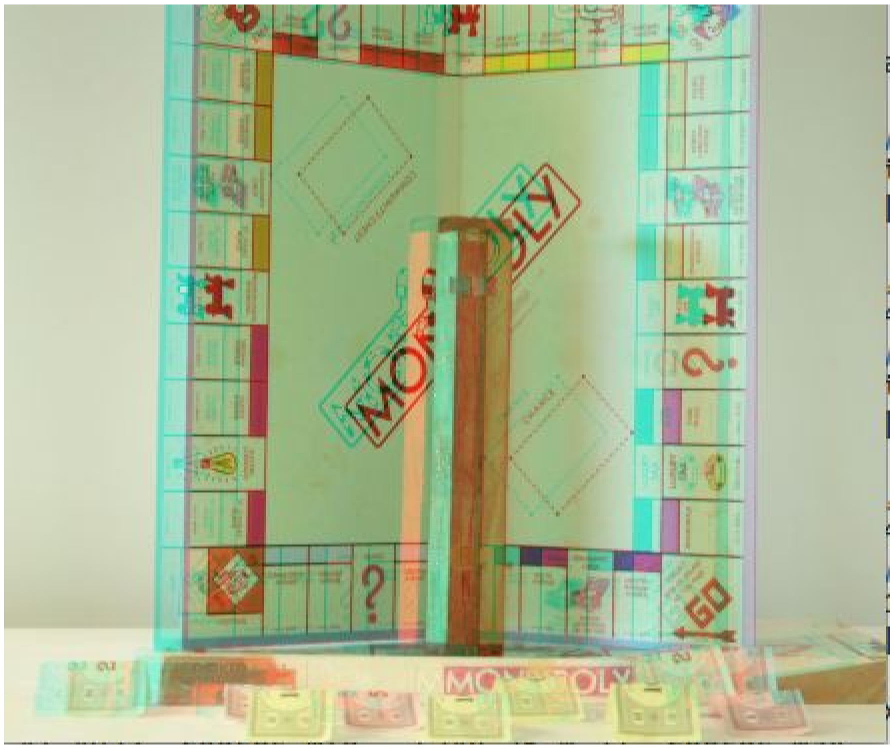
The objects closer to the camera are shifted more than those behind them. This creates a black void in the pixels where the foreground was shifted and background was not. To fix this, we consider the pixels of the object behind the object we’re currently on from the original image. For instance, we fill the void to the immediate right of the bowling pin with the pixels of the bowling ball and so on.

4. Constructing the 3D image:

In the get_anaglyph function, we simply create an image by taking the blue and green channels from the original image and the red channel from the shifted image.

Sample output:





Part 2

Section 1 (Naive segmentation)

Steps to run:

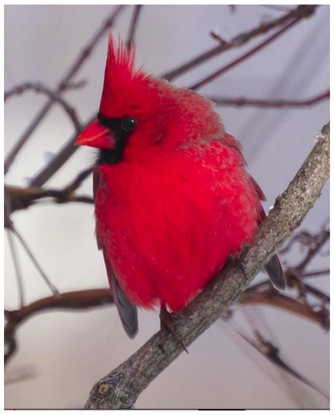
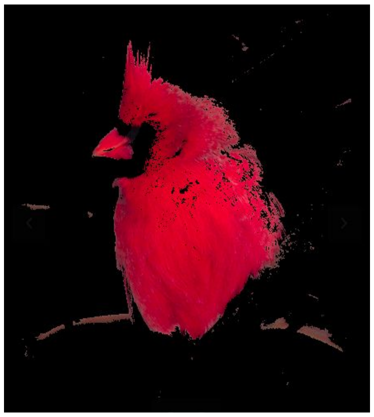
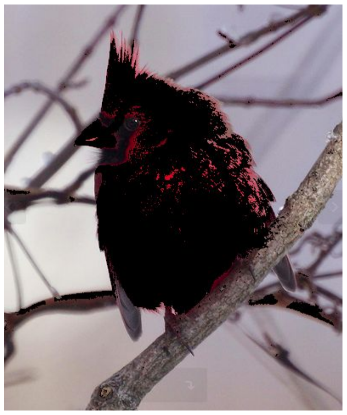
```
./segment part2/cardinal.png part2/cardinal-seeds.png
```

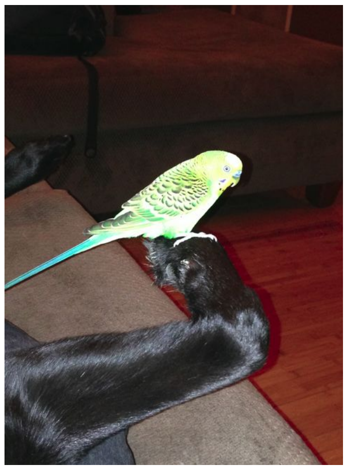
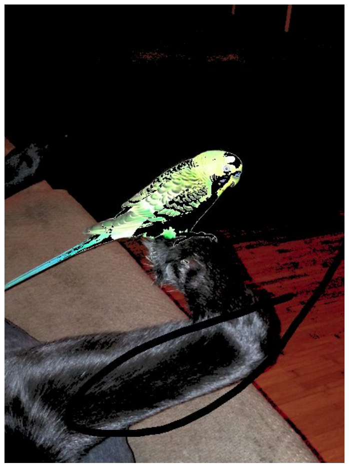
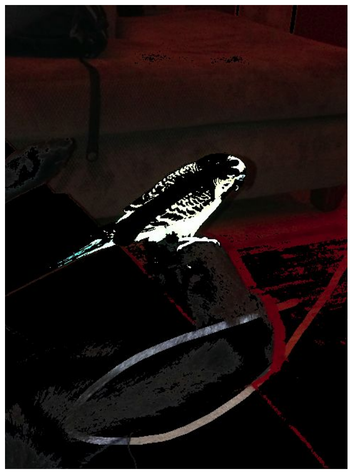
The 2 threshold parameters for Naive segmenter and MRF can be tweaked in config.txt

Brief idea Naive Segmenter:

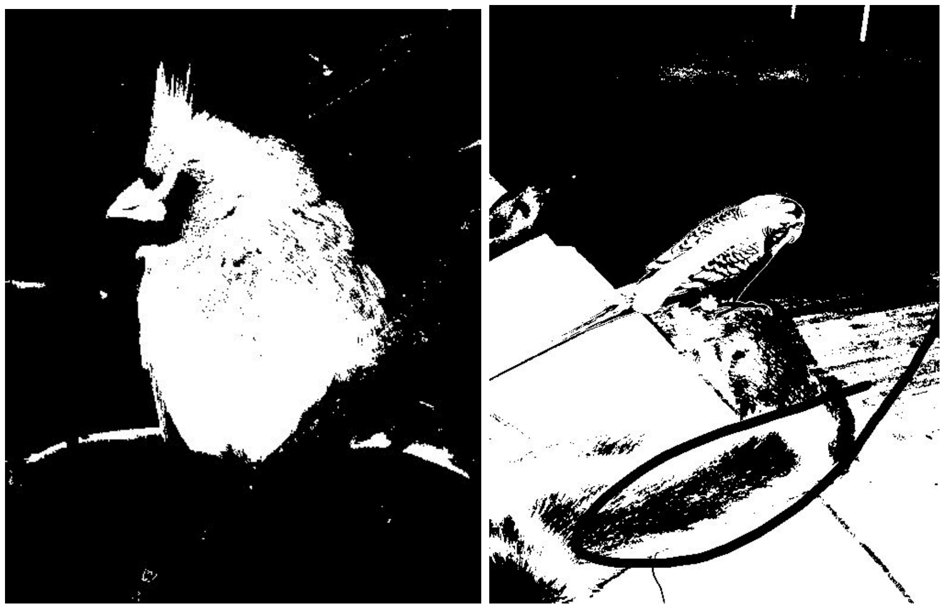
- We first calculate the mean and variance of foreground pixels to get an idea of what what foreground looks like.
- We scan every pixel and use a multivariate normal distribution using the calculated means and diagonal covariance matrix for every pixel in the RGB space. We then use a constant factor Beta. While Beta represents background the normal distribution represents foreground. Whichever factor produces lower count results in the corresponding labeling.
- We tried different values for the threshold and found that values between 13 and 18 give good results.

Sample output

Original Image	Foreground	Background
cardinal.png --> parameter = 15		
		
parakeet.png --> parameter = 18		

Disparity Map for cardinal.png and Parakeet.png with above K values.



Comment:

- MRF actually improves upon the naive result. This can be seen in places where it removes stray black or white pixels taking the neighborhood into consideration.

Section 2 (MRF segmentation)

Steps to run:

```
./segment naive part2/cardinal.png part2/cardinal-seeds.png
./segment mrf part2/cardinal.png part2/cardinal-seeds.png
```

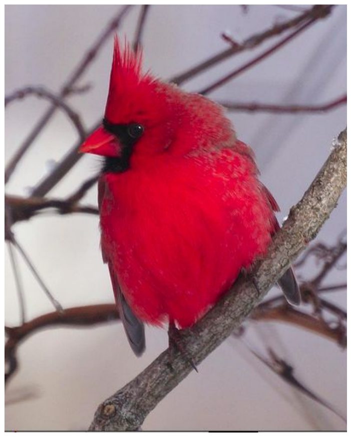
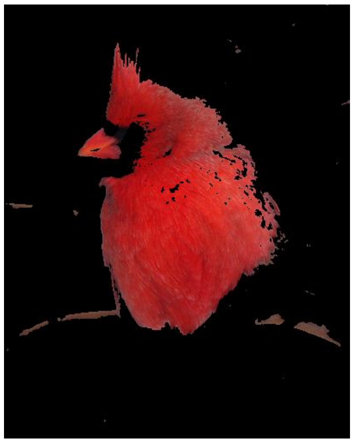
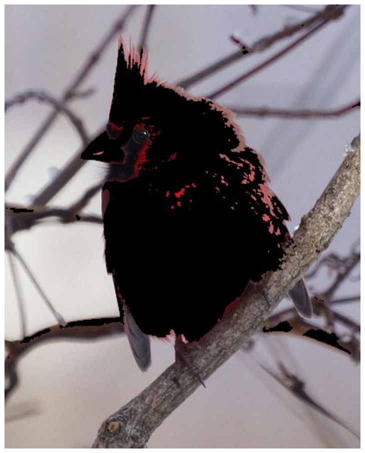
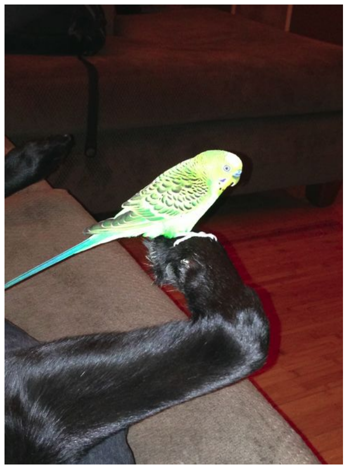
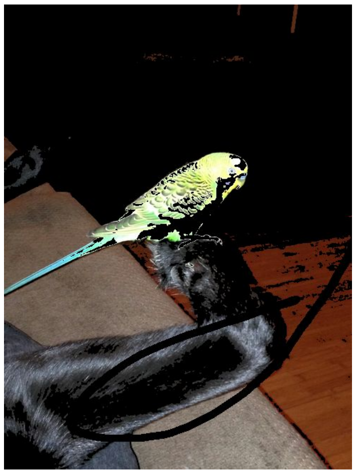
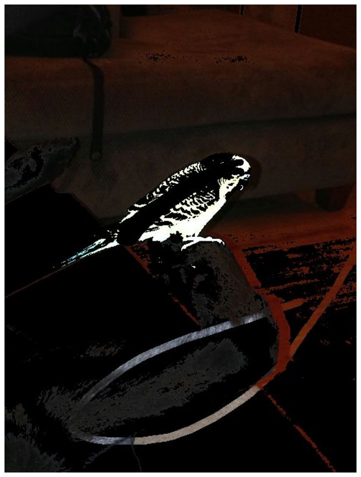
Brief Idea

- Run the below algorithm with the disparity image and emission probability (cost) created by Naive segmentation. The transition probability (cost) will simply be 1 if the states are not equal, or 0 if the states are equal.
- We have implemented a synchronous version of Loopy BP. We currently have a 2 4 dimensional matrix. We just used CImg class for it allows for 4 dimensional matrices (3 dimensional object with n channels). There are two such object: cur_msg, and prev_msg. In every iteration cur_msg is populated using the values from the prev_msg. In the initialisation phase, both these messages are zeroed out.
- In every iteration, the message that one node, transmits to **one** of the neighbors for **one** of the state s1 is a cost value. This cost value is calculated as minimum of sum of below variables evaluated for every state s2:
    - Cost of current node at state s2 (retrieve from emission cost of naive segmentation)
    - Cost of transition from current node at state s2 to the neighbor at state s1. (This is simply as a C expression:s1 == s2? 0: 1
- We've run MRF for 2 iterations, since it's taking too long to run. And it is giving us a decent output. Results slightly improve after every iteration.
    - We can improve the efficiency by storing the received messages from all other nodes for all of states the current node can take in an array. This avoids several repeat computations.

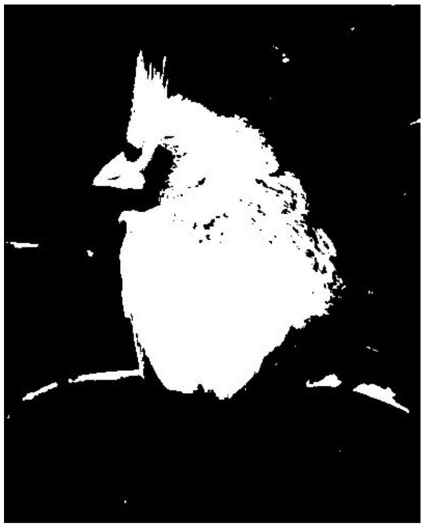
MRF Run time 2 iterations approx: 5-7 mins. Please increase number of iterations for better results.

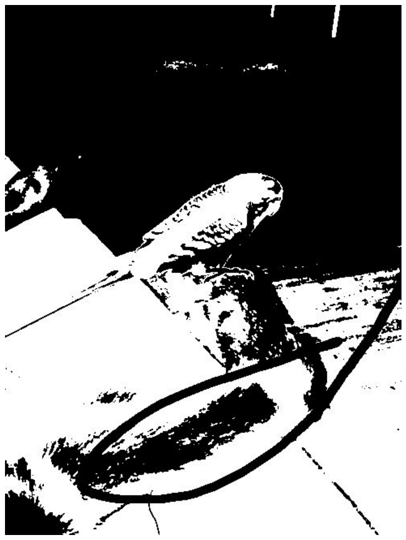
- We have committed using number of iterations in MRF to be 2. This is a configuration item present in config.txt. Please increase this number to get better results.
- Also, MRF produces images such as temp_000000.png, temp_000001.png, .. for every iteration. Take a look at it see how it improves at every iteration.

Sample Output

Original Image	Foreground	Background
cardinal.png --> parameter = 15		
		
parakeet.png --> parameter = 18		
		

Disparity Map for cardinal.png and Parakeet.png with above K values

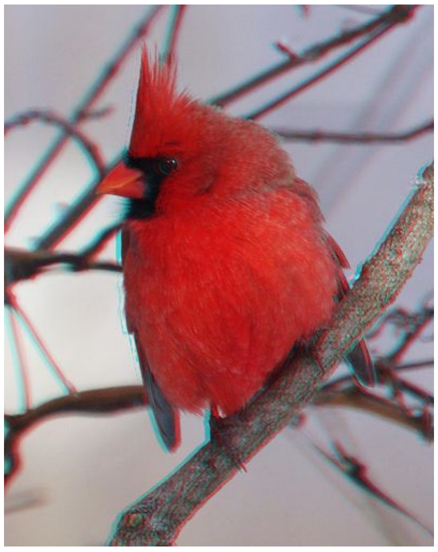


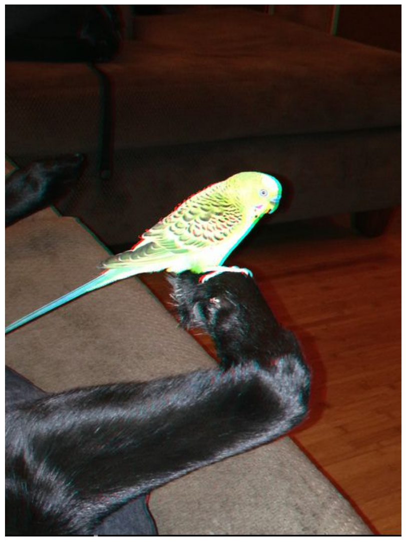


Part 2 Section 3
(Running Part 2 generated disparity maps using Part 1)

Stereo Output

Output for cardinal.png and parakeet.png (using MRF segmentation)





How well it works?

When viewed through the 3D glasses, the cardinal image looks actually quite well. For the parakeet image, we just need to focus a bit more to get the 3D effect, otherwise it works well too!

Part 3 (Stereo)

Steps to run

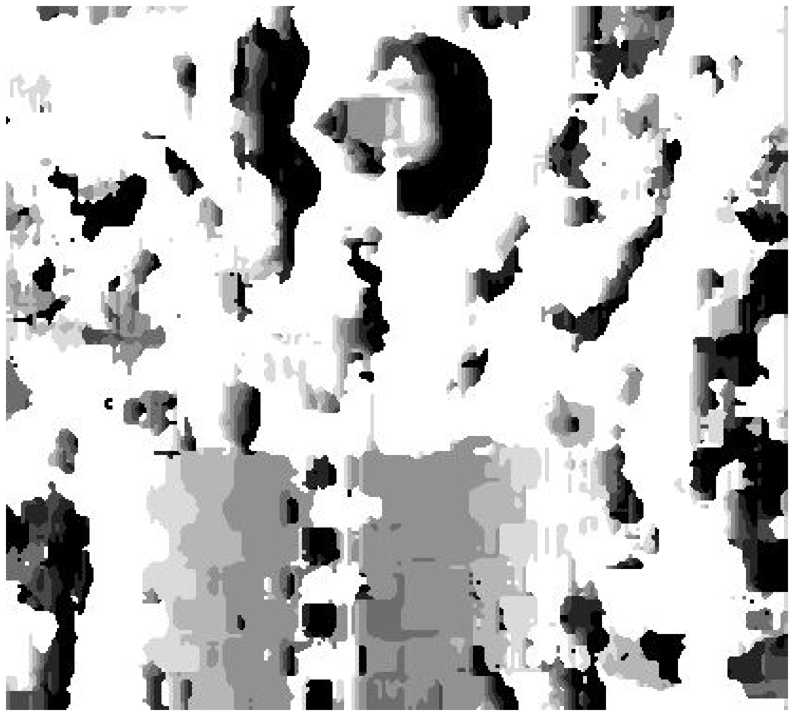
```
./stereo left_image.png right_image.png disp.png
```

Brief idea

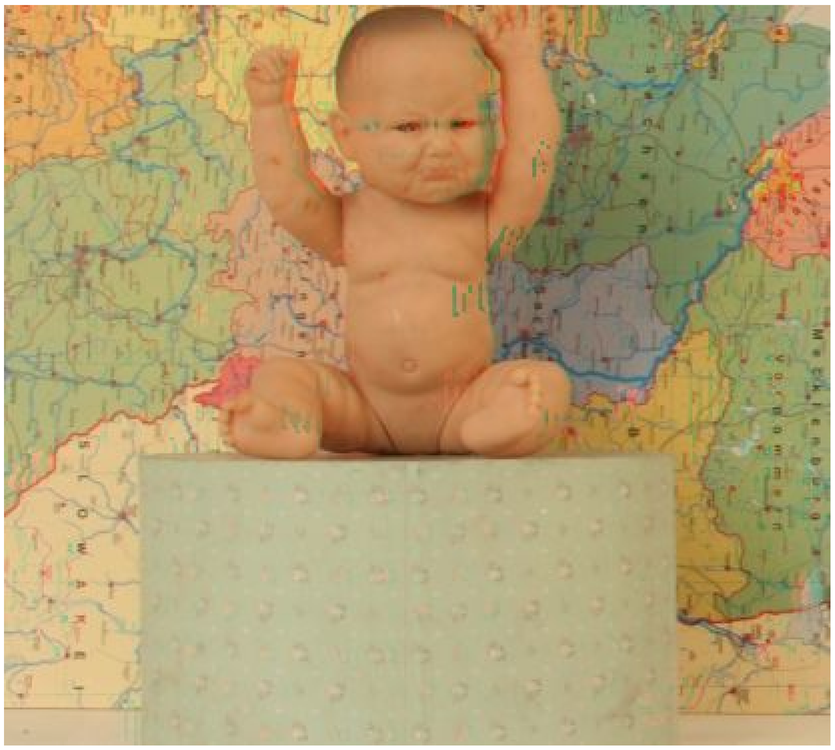
- We can almost use the 80% of the Naive stereo code for this part. In the naive stereo, we choose the depth that pays minimum cost. In MRF, we retain the cost calculated at every depth, and use that as cost function. The pairwise function remains the same as in part2.
- To improve the speed we have limited ourselves to only 8 depths. In actuality, we should be running with depth = image_width

Sample Output

Disparity Map for Baby1 picture



Output for Stereo (with MRF)



Output for Stereo (with Naive)