Project Summary

Ben Picker

1. Introduction

This project seeks to segment an image into background and foreground using loopy belief propagation. This will be performed using semi-supervised foreground/background segmentation based on hand-labeled examples from each class.

Masking

We wish to permit labeling of foreground and background without having to label the entire image. Instead, we want to permit only a small portion of the image to be marked and be able to extrapolate the rest of the labels from the markings (as seen below). This will reduce the difficulty of labeling considerably as the label line need only be within the confines of the foreground areas.

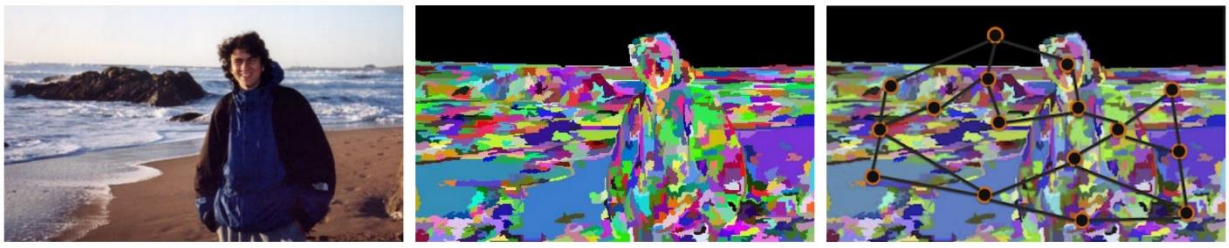
(a) Original Image (b) Labeled Image

Figure 1: Interactive foreground/background segmentation of input image.

As you can see, the entire image hasn't been labeled, just two lines that are supposed to indicate the respective areas. In this particular case, green = foreground and blue = background. Thus, this task will be to perform semi-supervised foreground/background segmentation based on hand-labeled examples from each class. The mask can be found in `scribble-mask.mat`.

Super pixels

We will also be introducing the notion of a super pixel for our analysis. A super pixel is basically a dictionary. For a given super pixel index i, there will be a set of indices $a, b, c, d, \ldots$, each of which are pixels associated with it.



(a) Original Image (b) Superpixels (c) Neighborhood Graph

Figure 2: Super pixels superimposed on original image and example neighborhood graph

This is useful because the number of pixels can become too great and increase the complexity of the analysis excessively. Good results can be achieved through dimensionality reduction by grouping the pixels together via similarity.

In this problem, the super pixel labels will be given. However, it is possible to generate these using K-means clustering using standard metrics (e.g. Euclidean distance) to measure the distance between pixels of the image. The super pixel labels can be found in `super_pixel_map.mat`.

2. Idealized Markov Random Field to represent latent foreground-background states

We will first introduce the idealized version of the image segmentation task without super pixels. It will provide a useful background for understanding the super pixel task.

We will have some feature of the image we are trying to capture which will be represented by Y_i. For instance, if 1D it could be brightness, if 3D it could be color (e.g. RGB or CIELUV). Furthermore, each pixel will have a corresponding random variable $X_i \in \{0,1\}$, where $0 =$ background and $1 =$ foreground.

Importantly, we will assume the following structure to the graph.

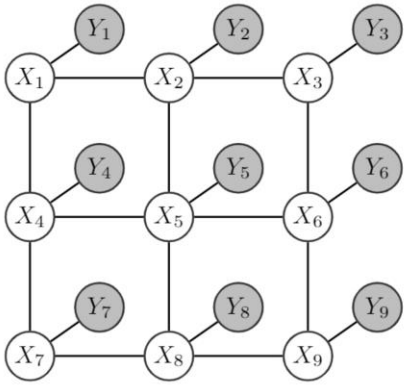


Figure 3: An example Markov random field for image segmentation.

An image is $H \times W$ grid where there are 3 classes of pixels: middle, non-corner border pixels, and corner pixels.

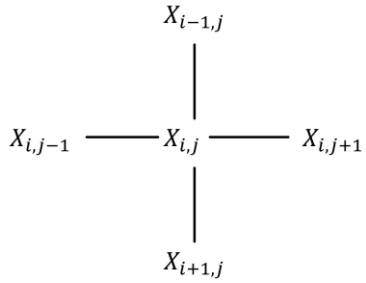
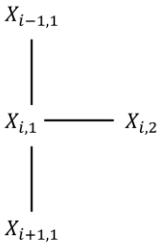
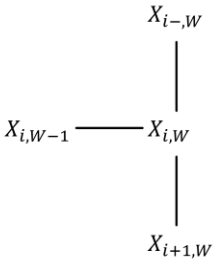
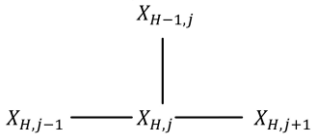
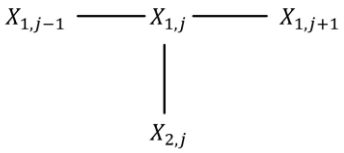
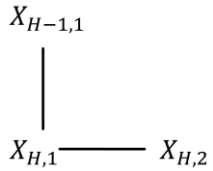
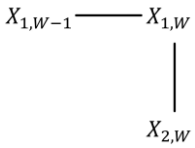
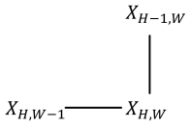
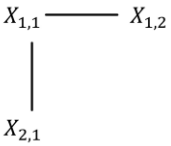
MIDDLE PIXEL	NON-CORNER BORDER PIXEL	CORNER BORDER PIXEL
	*Vertical, Left:*  *Vertical, Right:*  *Horizontal, bottom:*  *Horizontal, top:* 	*Lower left:*  *Upper right:*  *Lower right:*  *Upper Left:* 

Table 1: Graph edge relationships for different parts of the grid

This model will make the following assumptions:

- *Pairwise Markov Property:* Any two non-adjacent variables are conditionally independent given all other variables.[1]

$$X_u \perp X_v | X_{V \setminus \{u,v\}}$$

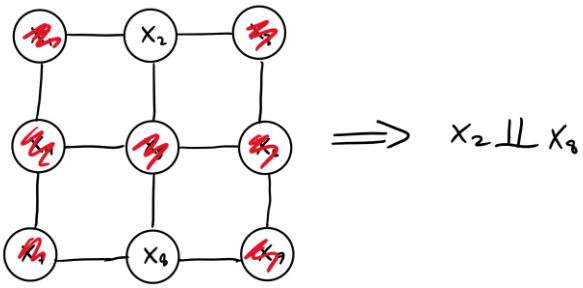


Figure 4: Example PGM with Pairwise Markov Property

- *Local Markov Property:* A variable is conditionally independent of all other variables given its neighbors:[2]

$$X_v \perp X_{V \setminus N[v]} | X_{N[v]}$$

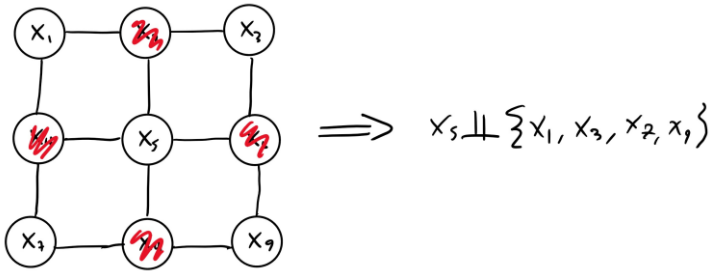


Figure 5: Example PGM with Local Markov Property

- *Global Markov Property*: Any two subsets of variables are conditionally independent given a separating subset.[3]

$$X_A \perp X_B | X_{\text{sep}}$$

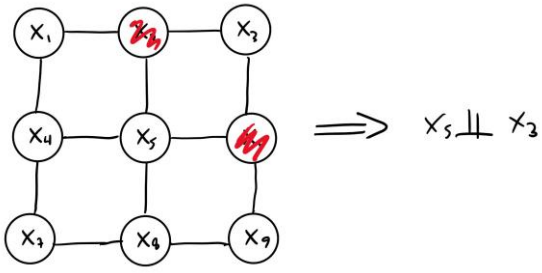


Figure 6: Example PGM with Global Markov Property

[1] Source: https://en.wikipedia.org/wiki/Markov_random_field
[2] Source: https://en.wikipedia.org/wiki/Markov_random_field
[3] Source: https://en.wikipedia.org/wiki/Markov_random_field

Thus, assume we have a graph $G = (V, E)$ which represent the X_is and Y_is. Then, if the X_is have the above properties, we call this a Markov Random Field, which we will denote $\mathcal{U}$. Note, we assume the Y_is are given since we will have the image already and just want the latent states representing the background and foreground.

We can then define node potentials and edge potentials. There will be one node potential ϕ_i for each pixel, which will be indexed $i \in \{1, \dots, H \times W\}$. Similarly, we will also have edges e_{ij} where pixels follow the edge rules above. We will denote these edges via $E_{\mathcal{U}}$. The edge potential will be denoted $\phi_{i,j}$. This yields the joint distribution

$$P(\boldsymbol{X}, \boldsymbol{Y}) = \frac{1}{Z} \prod_{i=1}^{H \times W} \phi_i(X_i, Y_i) \prod_{(i-j) \in E_{\mathcal{U}}} \phi_{i,j}\left(X_i, X_j\right) \tag{1}$$

where $\phi_i(X_i, Y_i)$ relates a pixel's class (i.e. background or foreground) to its feature representation, Y_i and $\phi_{i,j}\left(X_i, X_j\right)$ expresses the influence among neighboring pixels.

3. Super Pixel Markov Random Field to represent latent foreground-background states

The above model suffers from problems with complexity because, for large H and W, the number of edges and nodes increase the cost of computation excessively. Consequently, we will try to reduce the scope of the computation using super pixels, which are just clusters of pixels.

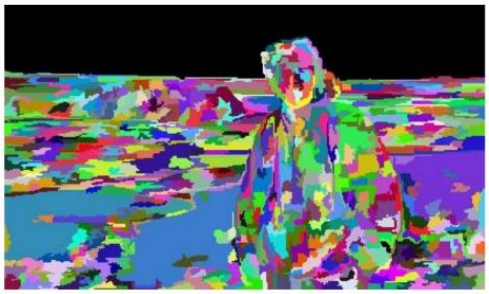


Figure 7: Example grouping of the image by super pixels

Consequently, instead of a $H \times W$ grid, we will have a network of super pixels.

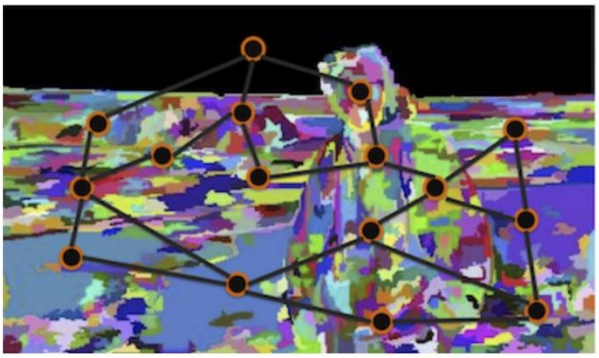


Figure 8: Example neighborhood graph on super pixels

After clustering the pixels, we need to create a graph to represent them. The graph we will work with will not be between pixels, but instead between super pixels and we will want an adjacency matrix to describe which super pixels are connected. To create the adjacency matrix between super pixels, it will be necessary therefore to first identify all the adjacent super pixels. But keep in mind, a graph doesn't have any spatial organization necessarily. Although the grid in Figure 2 is spatially organized, the super pixels need not be. We get the adjacency matrix using `get_supix_adj_matrix`. The logic can be read from the code. The adjacency matrix can be seen visualized below:

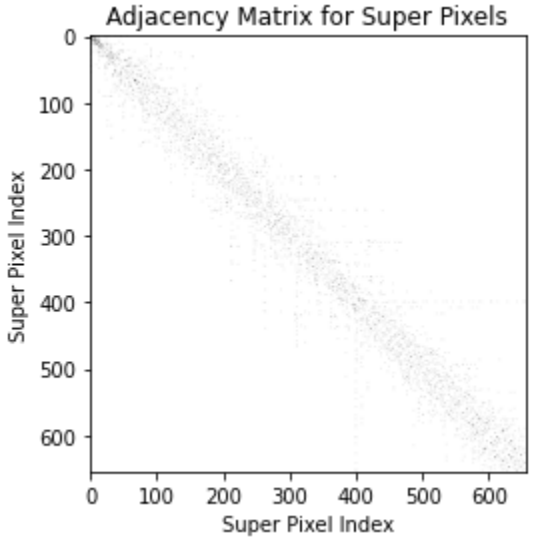


Figure 9: Adjacency matrix for this problem

From the adjacency matrix, we can then check the number of adjacent cells by summing the rows, which yields the following histogram.

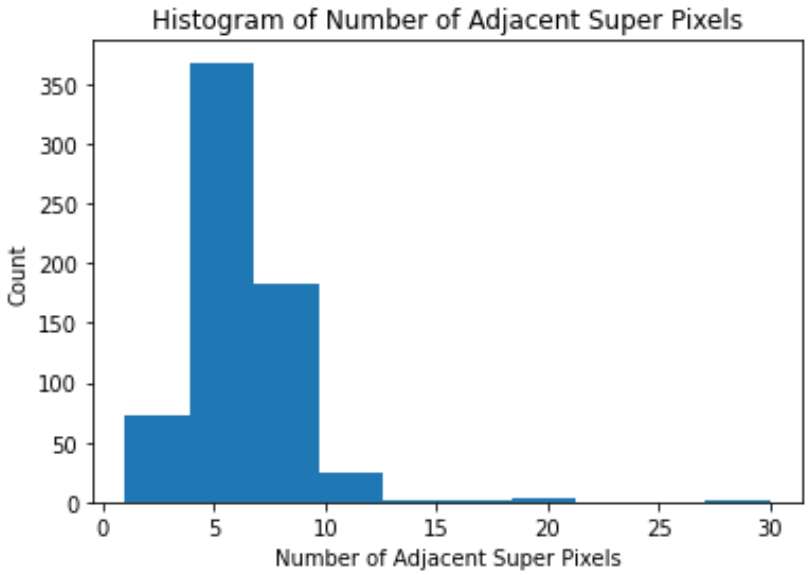


Figure 10: Histogram of number of adjacent super pixels

Notice how, unlike the uniformity of the edge rules for the idealized MRF model, the nodes in the super pixel model vary considerably in the number of neighbors. Despite these limitations, we are going to make the same assumptions that the graph is an MRF.

4. Super Pixel Implementation of Uncalibrated Beliefs

Let's first introduce a few notations. Let Y_i denote the ith super pixel, whose value will be the average color associated with all the pixels assigned to that super pixel. This can be obtained using `get_super_image`, which returns the average color in each CIELUV dimension associated with super pixel's pixels. Let X_i be binary which denotes whether that super pixel belongs to the foreground or background. We will assume that, if a super pixel belongs to the background, then all of its associated pixels belong to the background (and vice versa).

We want the probability of a given average color vector to be conditioned on the foreground and background. Thus, we will have two separate GMMs:

$$P_{\text{fg}}(Y) = \sum_{k=1}^{K} \pi_k^{(\text{fg})} f\left(Y; \mu_k^{(\text{fg})}, \Sigma_k^{(\text{fg})}\right) \tag{2}$$

$$P_{\text{bg}}(Y) = \sum_{k=1}^{K} \pi_k^{(\text{bg})} f\left(Y; \mu_k^{(\text{bg})}, \Sigma_k^{(\text{bg})}\right) \tag{3}$$

The number of classes K for the GMMs are parameters of the model. Given Equation 3, this means we will need to estimate, for K classes, we would have

$$\left\{\pi_1^{(\text{fg})}, \cdots, \pi_K^{(\text{fg})}, \mu_1^{(\text{fg})}, \cdots, \mu_K^{(\text{fg})}, \Sigma_1^{(\text{fg})}, \cdots, \Sigma_K^{(\text{fg})}, \pi_1^{(\text{bg})}, \cdots, \pi_K^{(\text{bg})}, \mu_1^{(\text{bg})}, \cdots, \mu_K^{(\text{bg})}, \Sigma_1^{(\text{bg})}, \cdots, \Sigma_K^{(\text{bg})}, \right\}$$

These will then be the inputs such that

$$\phi_i(X_i = 1, Y_i) = P_{\text{fg}}(Y_i) \tag{4}$$

$$\phi_i(X_i = 0, Y_i) = P_{\text{bg}}(Y_i) \tag{5}$$

That gives us the node potentials. Meanwhile, the edge potential will be defined via

$$\phi_{i,j}(X_i, X_j) = \exp\left(-\rho \cdot \mathbb{I}(X_i \neq X_j)\right) \tag{6}$$

where ρ is a parameter of the model and $\mathbb{I}$ is the indicator function. Intuitively, the edge potentials are 1 if $X_i = X_j$ and $e^{-\rho}$ if $X_i \neq X_j$. Thus, the edge potentials are uniformly influenced by our choice of ρ and so the variations we see in the edge potential impact will come from variation in the node potentials.

Note: all of this should be done using log probabilities to avoid numerical instabilities.

To train the GMM, we take the points labeled by the scribbles and train the GMM on these. Thus, `data_fg` corresponds to the points labeled by the foreground scribble and `data_bg` corresponds to the points labeled by the background scribble. With these, we train the GMM.

We then initialize our model using

Procedure: Initialize_CGraph$(Y, P_{fg}(\cdot), P_{\text{bg}}(\cdot))$**:**
  for each super pixel $i \in \{0, \cdots, n_{\text{supix}}\}$:
    $\log\left(\beta_i^{(\text{fg})}\right) = \log\left(P_{\text{fg}}(Y_i)\right)$
    $\log\left(\beta_i^{(\text{bg})}\right) = \log\left(P_{\text{bg}}(Y_i)\right)$
  for each edge $(i - j) \in E_{\mathcal{U}}$:
    $\log(\delta_{ij}) = 0$
    $\log(\delta_{jj}) = 0$
  return $\log(\boldsymbol{\delta}^{(\text{fg})}), \log(\boldsymbol{\delta}^{(\text{bg})}), \log(\boldsymbol{\beta}^{(\text{fg})}), \log(\boldsymbol{\beta}^{(\text{bg})})$

Figure 11: Cluster graph initialization algorithm

Importantly, we call $\log\left(P_{\text{fg}}(Y_i)\right)$ and $\log\left(P_{\text{bg}}(Y_i)\right)$ the ith log-beliefs. These initializations will be what we use for belief propagation.

6. Belief Propagation

The goal of belief propagation is to essentially update the probabilities of a given super pixel belonging to the background or foreground. If we can update these probabilities, then for any given super pixel we can do

$$\underset{\{\text{fg,bg}\}}{\text{argmax}}\{P_{\text{fg}}(Y_i), P_{\text{bg}}(Y_i)\} \tag{7}$$

which should give us a rule as to whether the pixel belongs to the foreground or background. The question is, how do we hone these probabilities?

$\{Y_i\}_{i=1}^{n_{\text{supix}}}$ \\ the "super image," i.e. average color for
\\ each super pixel
$P_{\text{fg}}(\cdot)$ \\ the initial GMM pdf for getting a specific Y_i
if it belongs to the foreground
$P_{\text{bg}}(\cdot)$ \\ the initial GMM pdf for getting a specific Y_i
if it belongs to the background

Procedure: CGraph-SP-Calibrate$\left(\{Y_i\}_{i=1}^{n_{\text{supix}}}, P_{\text{fg}}(\cdot)\ , P_{\text{bg}}(\cdot)\ \right)$

$\log(\boldsymbol{\delta}^{(\text{fg})})\ , \log\left(\boldsymbol{\delta}^{(\text{bg})}\right), \log\left(\boldsymbol{\beta}^{(\text{fg})}\right), \log\left(\boldsymbol{\beta}^{(\text{bg})}\right)$
$=$Initialize-CGraph$\left(\{Y_i\}_{i=1}^{n_{\text{supix}}}, P_{\text{fg}}(\cdot)\ , P_{\text{bg}}(\cdot)\ \right)$
\\ this step is completely unnecessary, but I'm including it
\\ to match Koller's notation
$\log\left(\boldsymbol{\psi}^{(\text{fg})}\right), \log\left(\boldsymbol{\psi}^{(\text{bg})}\right) = \log\left(\boldsymbol{\beta}^{(\text{fg})}\right), \log\left(\boldsymbol{\beta}^{(\text{bg})}\right)$
\\ while… graph not calibrated
while $\exists i, j$ s.t. $\left|\log\left(\delta_{ij}\right) - \log\left(\delta'_{ij}\right)\right| > \epsilon$:
Select $(i - j) \in E_{\mathcal{U}}$
$\log\left(\delta_{ij}^{(\text{fg})}\right), \log\left(\delta_{ij}^{(\text{bg})}\right)$
$= \text{SP-Message}\left(\boldsymbol{\psi}^{(\text{fg})}, \boldsymbol{\psi}^{(\text{bg})}, \boldsymbol{\delta}^{(\text{fg})}, \boldsymbol{\delta}^{(\text{bg})}\right)$
\\ after the messages stop changing, then we update the
\\ probability functions so we can compute our argmax
\\ tilde denotes updated versions of beliefs
for each super pixel i:

$$\log\left(\tilde{\beta}_i^{(\text{fg})}\right) = \log\left(\psi_i^{(\text{fg})}\right) + \sum_{k \in \text{Nb}_i} \log\left(\delta_{ki}^{(\text{fg})}\right)$$

$$\log\left(\tilde{\beta}_i^{(\text{bg})}\right) = \log\left(\psi_i^{(\text{bg})}\right) + \sum_{k \in \text{Nb}_i} \log\left(\delta_{ki}^{(\text{bg})}\right)$$

\\ the updated probabilities that we can use for the argmax
return $\left\{\log\left(\tilde{\beta}_i^{(\text{fg})}\right)\right\}_{i=1}^{n_{\text{supix}}}, \left\{\log\left(\tilde{\beta}_i^{(\text{bg})}\right)\right\}_{i=1}^{n_{\text{supix}}}$

Figure 12: Cluster graph sum-product calibration algorithm for loopy belief propogation

Let's break this down. We start off with these initial log-probabilities and we have these things called "log-messages" which we set to 0. We want to change the log-messages until they stop changing. So for each edge, we update the log-messages. Once we get the (hopefully) converged log-messages, then we can update the probabilities, so called "calibration."

We also need then SP-Message, which is defined below:

Procedure: SP-Message$\left(\boldsymbol{\psi}^{(\mathrm{fg})}, \boldsymbol{\psi}^{(\mathrm{bg})}, \boldsymbol{\delta}^{(\mathrm{fg})}, \boldsymbol{\delta}^{(\mathrm{bg})}\right)$

\\ Koller step 1

$$\log\left(\psi^{(\mathrm{fg})}(\boldsymbol{C}_i)\right) = \log\left(\psi_i^{(\mathrm{fg})}\right) + \sum_{k\in(\mathrm{Nb}_i-\{j\})} \log\left(\delta_{ki}^{(\mathrm{fg})}\right)$$

$$\log\left(\psi^{(\mathrm{bg})}(\boldsymbol{C}_i)\right) = \log\left(\psi_i^{(\mathrm{bg})}\right) + \sum_{k\in(\mathrm{Nb}_i-\{j\})} \log\left(\delta_{ki}^{(\mathrm{fg})}\right)$$

\\ Koller step 2

$$\tau^{(\mathrm{fg})}(\boldsymbol{S}_{ij}) = \sum_{\boldsymbol{C}_{ij}-\boldsymbol{S}_{ij}} \exp\left(\log\left(\psi^{(\mathrm{fg})}(\boldsymbol{C}_i)\right)\right)$$

$$\tau^{(\mathrm{bg})}(\boldsymbol{S}_{ij}) = \sum_{\boldsymbol{C}_{ij}-\boldsymbol{S}_{ij}} \exp\left(\log\left(\psi^{(\mathrm{bg})}(\boldsymbol{C}_i)\right)\right)$$

\\ then we rescale them to get Gibbs distributions

$$Z = \tau^{(\mathrm{fg})}(\boldsymbol{S}_{ij}) + \tau^{(\mathrm{bg})}(\boldsymbol{S}_{ij})$$

$$\log\left(\delta_{ij}^{(\mathrm{fg})}\right) = \log\left(\frac{1}{Z}\,\tau^{(\mathrm{fg})}(\boldsymbol{S}_{ij})\right)$$

$$\log\left(\delta_{ij}^{(\mathrm{bg})}\right) = \log\left(\frac{1}{Z}\,\tau^{(\mathrm{bg})}(\boldsymbol{S}_{ij})\right)$$

\\ the updated probabilities that we can use for the argmax

return $\log\left(\delta_{ij}^{(\mathrm{fg})}\right), \log\left(\delta_{ij}^{(\mathrm{bg})}\right)$

Figure 13: Sum-product message update algorithm

This is the logged version of Koller's version of Judea Peral's algorithm, which can be seen here:

Algorithm 11.1 Calibration using sum-product belief propagation in a cluster graph

Procedure CGraph-SP-Calibrate (
  Φ, // Set of factors
  $\mathcal{U}$ // Generalized cluster graph Φ
)
1 Initialize-CGraph
2 **while** graph is not calibrated
3 Select $(i\text{–}j) \in \mathcal{E}_{\mathcal{U}}$
4 $\delta_{i \to j}(\boldsymbol{S}_{i,j}) \leftarrow$ SP-Message(i, j)
5 **for** each clique i
6 $\beta_i \leftarrow \psi_i \cdot \prod_{k \in \mathrm{Nb}_i} \delta_{k \to i}$
7 **return** $\{\beta_i\}$

Procedure Initialize-CGraph (
  $\mathcal{U}$
)
1 **for** each cluster $\boldsymbol{C}_i$
2 $\beta_i \leftarrow \prod_{\phi \,:\, \alpha(\phi)=i} \phi$
3 **for** each edge $(i\text{–}j) \in \mathcal{E}_{\mathcal{U}}$
4 $\delta_{i \to j} \leftarrow 1$
5 $\delta_{j \to i} \leftarrow 1$
6

Procedure SP-Message (
  i, // sending clique
  j // receiving clique
)
1 $\psi(\boldsymbol{C}_i) \leftarrow \psi_i \cdot \prod_{k \in (\mathrm{Nb}_i - \{j\})} \delta_{k \to i}$
2 $\tau(\boldsymbol{S}_{i,j}) \leftarrow \sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \psi(\boldsymbol{C}_i)$
3 **return** $\tau(\boldsymbol{S}_{i,j})$

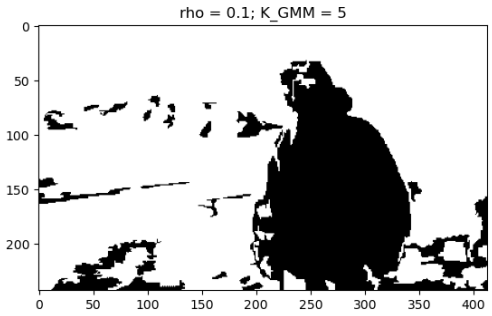
Figure 14: CGraph-SP-Calibrate Algorithm from Koller

Due to time constraints, I was not able to elaborate further on the details of the algorithm. But the results were acceptable, so I will have to pursue that at a later time.

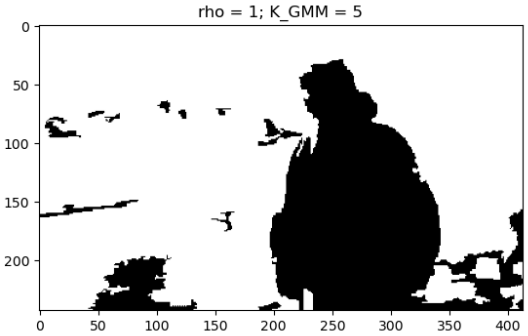
7. Results

The model takes ρ as a parameter for the influence of the edge potentials on the updates. I tested this by varying it and the results can be seen below. The optimal value by visual inspection appears to be 7.75.

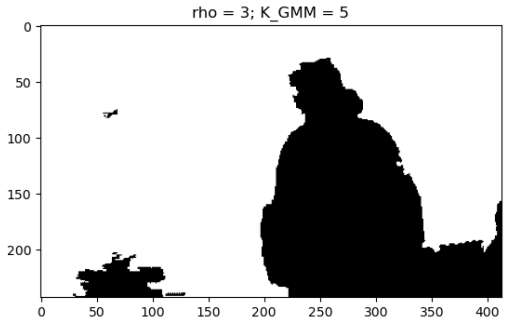
(a)



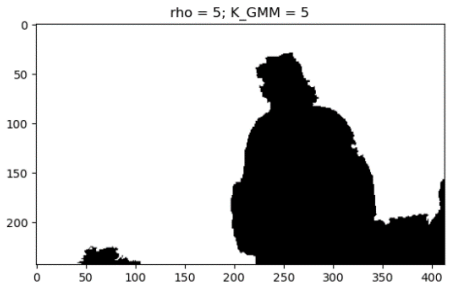
(b)



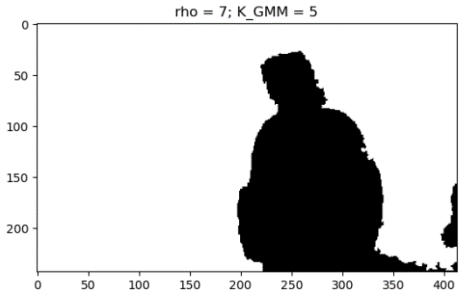
(c)



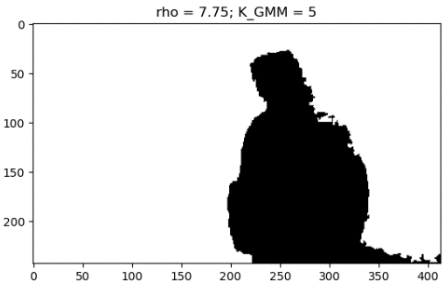
(d)



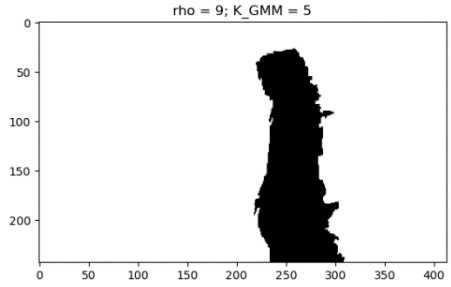
(e)



(f)


rho = 7.75; K_GMM = 5
0
50
100
150
200
0
50
100
150
200
250
300
350
400

(g)


rho = 9; K_GMM = 5
0
50
100
150
200
0
50
100
150
200
250
300
350
400

(h)