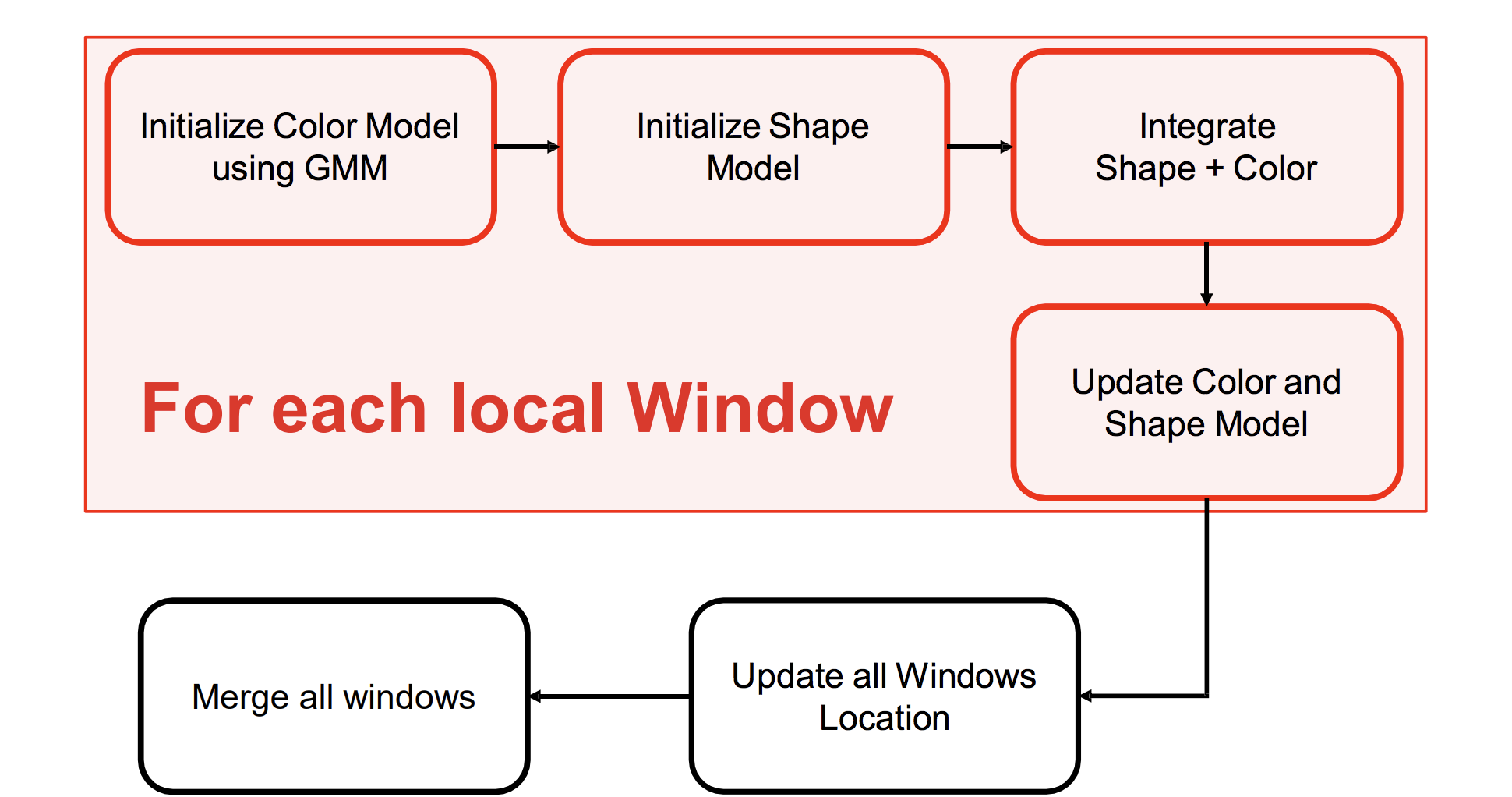
For implementing rotobrush, we followed the systems diagram that was given to us in the slides:



**Setup Local Windows**

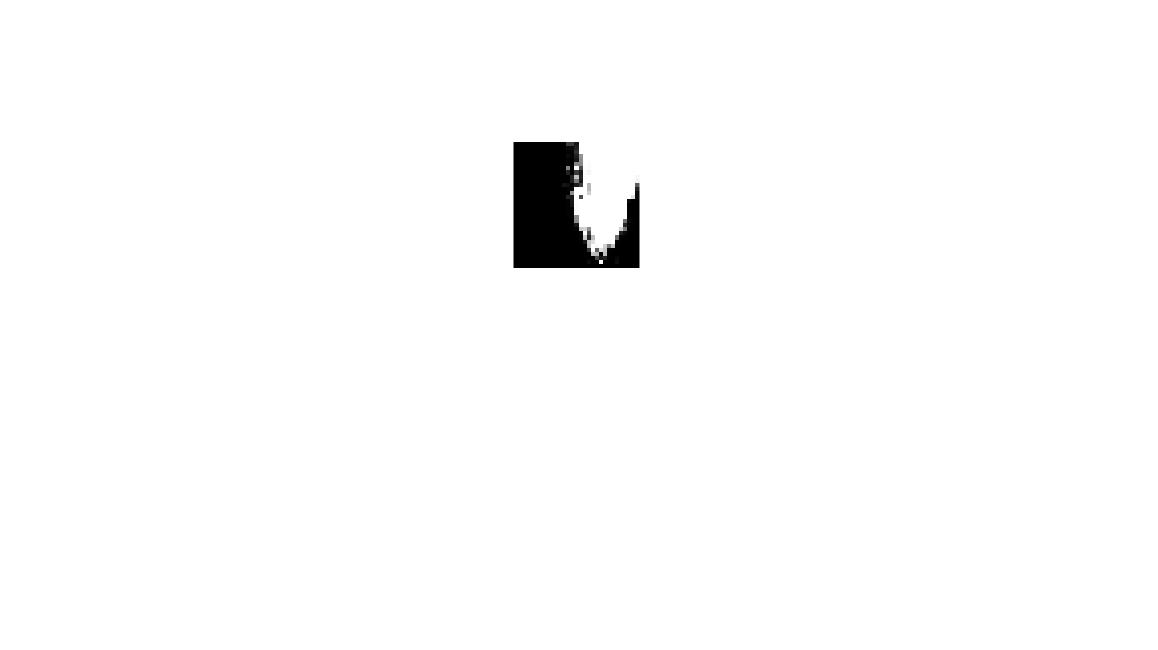
To initialize the shape/color models, we needed to create local windows along the outline of each figure. Luckily, we were given initLocalWindows.m so all we had to do was use roipoly to make the mask. We chose to have 35 windows with a width of 30 pixels because we learned that we wanted to have as many points as possible without slowing the runtime, and we wanted about 1/3 of the window to overlap with another. Below is an image with the local windows we had for the turtle.



**Initialize Color Models**

Creating the color models wasn’t too difficult in terms of the math, but figuring out what we wanted to store in color models was difficult. We decided to store for each window in a cell: color model confidence, GMM for background and foreground, local windows, the points in the foreground and background, distance from points to the mask, and foreground probability for each point. We initially only had confidence, GMM, distance, and probability. We, then later added local windows, and points when we wrote the updateModels file. For the math, we followed what was in the slides to calculate foreground probability and confidence.

Here is an image of a window on the left and what we calculated as the foreground probability on the right:



**Compute Color Model Confidence**

To calculate color model confidence, we used . We used bwdist to calculate distance for ω. To find , we used and the GMM and pdf to find the foreground probability. As you can see from the above images, we thought we did pretty well with the results of the color model.

**Initialize Shape Model**

Creating the shape models also wasn’t that difficult. The only thing we decided to store in it was the shape confidences.

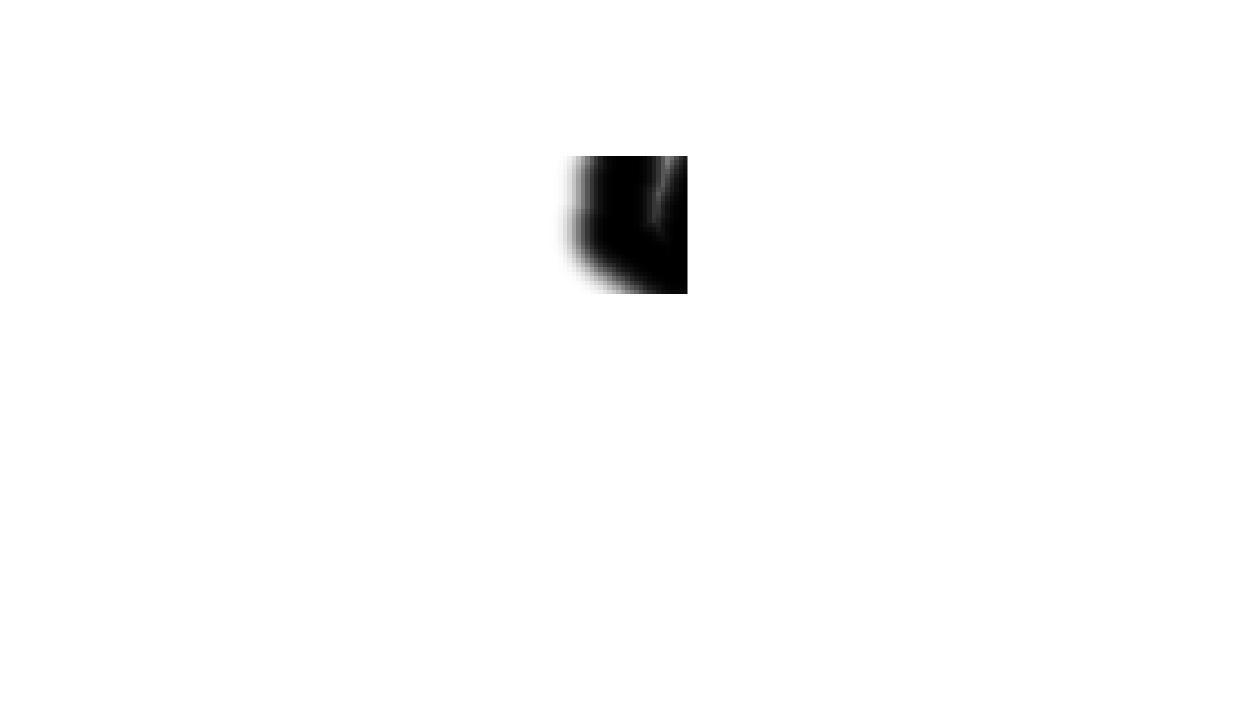
**Compute Shape Confidence**

To calculate confidence, we used the formula in the slides. . For , we used the formula in the slides. If was between and 1, then . Otherwise, if it was between 0 and , then .

For the parameters, we used 0.6 for , 100 for , 2 for R, and about 100 for A.

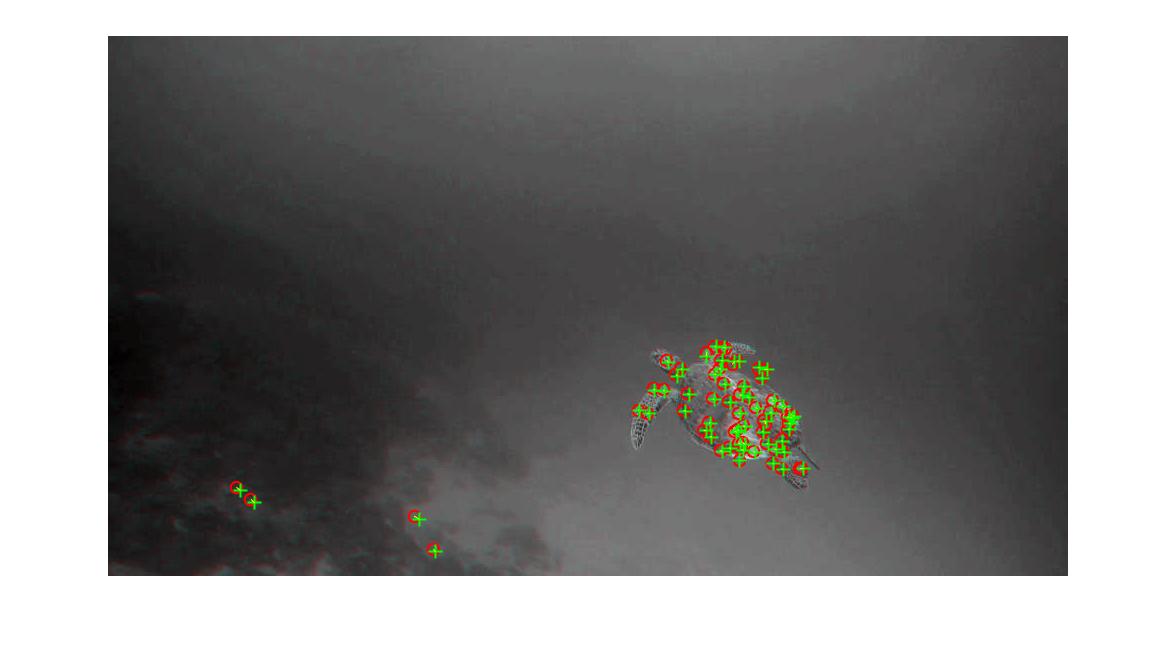
Here is an example of a shape confidence we calculated:

It looks a little off, but that may be because the parameters could’ve been adjusted a little better.

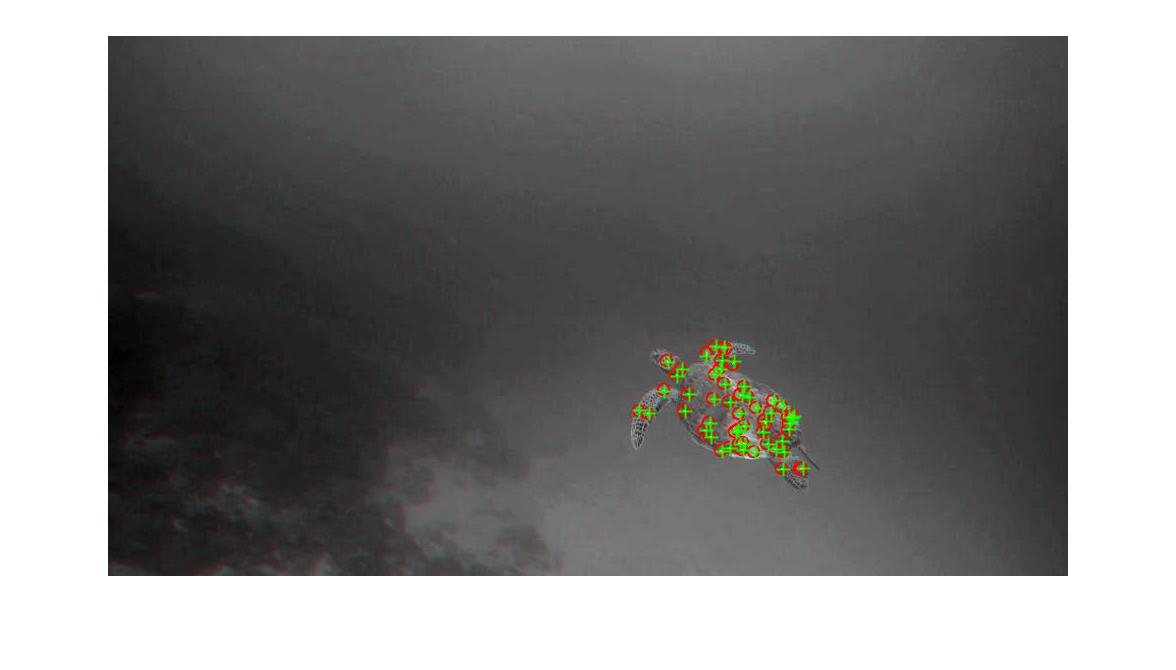


**Estimate Entire Object Motion**

For calculateGlobalAffine, we initially were using detectHarrisFeatures to get the features, but we weren’t satisfied with those results, so we switched to detectSURFFeatures. Below is a picture of the matched features on the first frame with the second frame after we extracted the features.

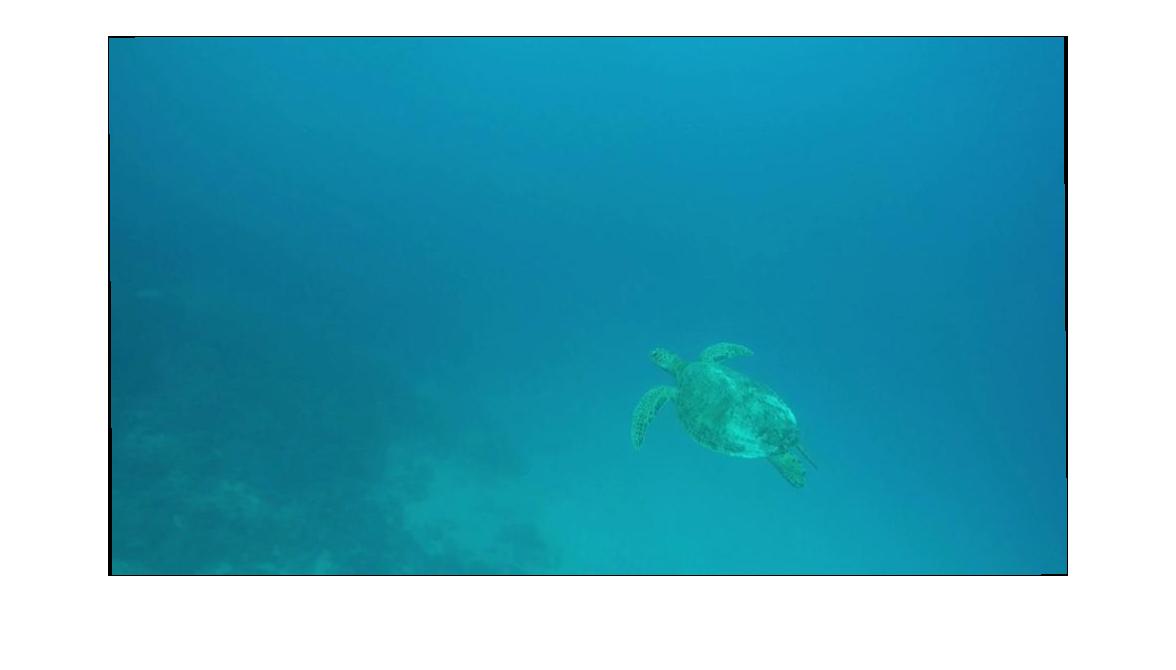


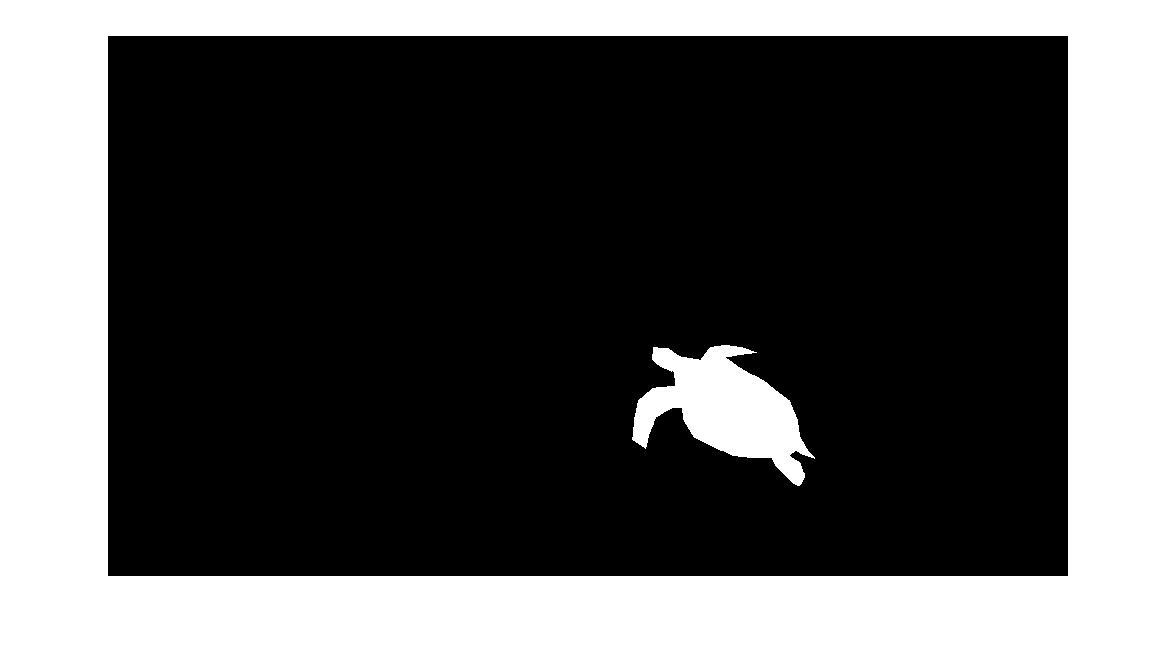
As you can see, there were some features that shouldn’t have been there on the left. We then filtered those out by checking if it was in the mask. The image below is what we got after this filtering.



We then used estimateGeometricTransform, imwarp, transformPointsForward to get the warped mask, warped mask outline, warped frame, and new local windows.

Below are images of the warped frame and warped mask.





**Estimate Local Boundary Deformation**

For localFlowWarp, we used estimateFlow and opticalFlowFarneback

