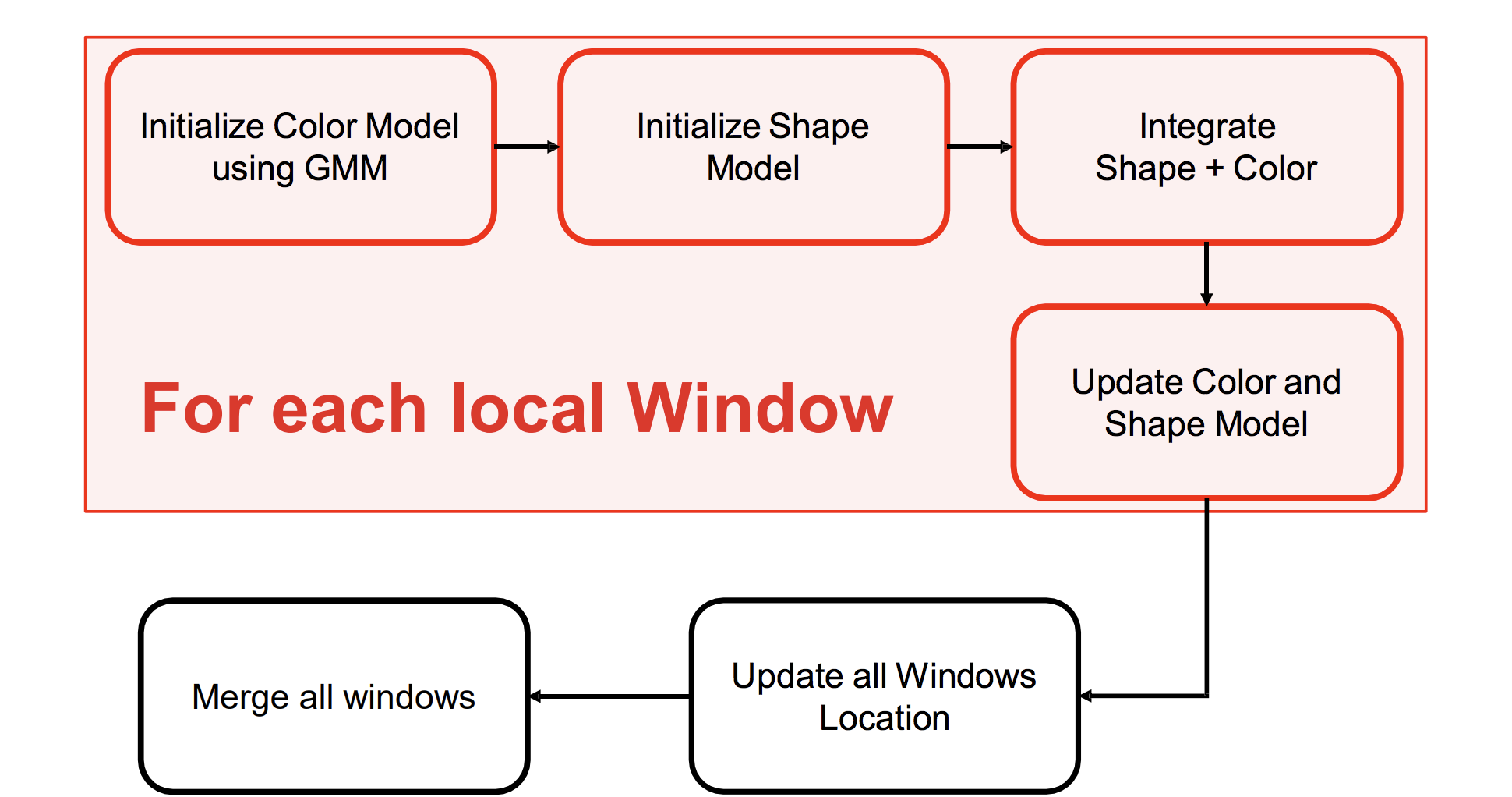
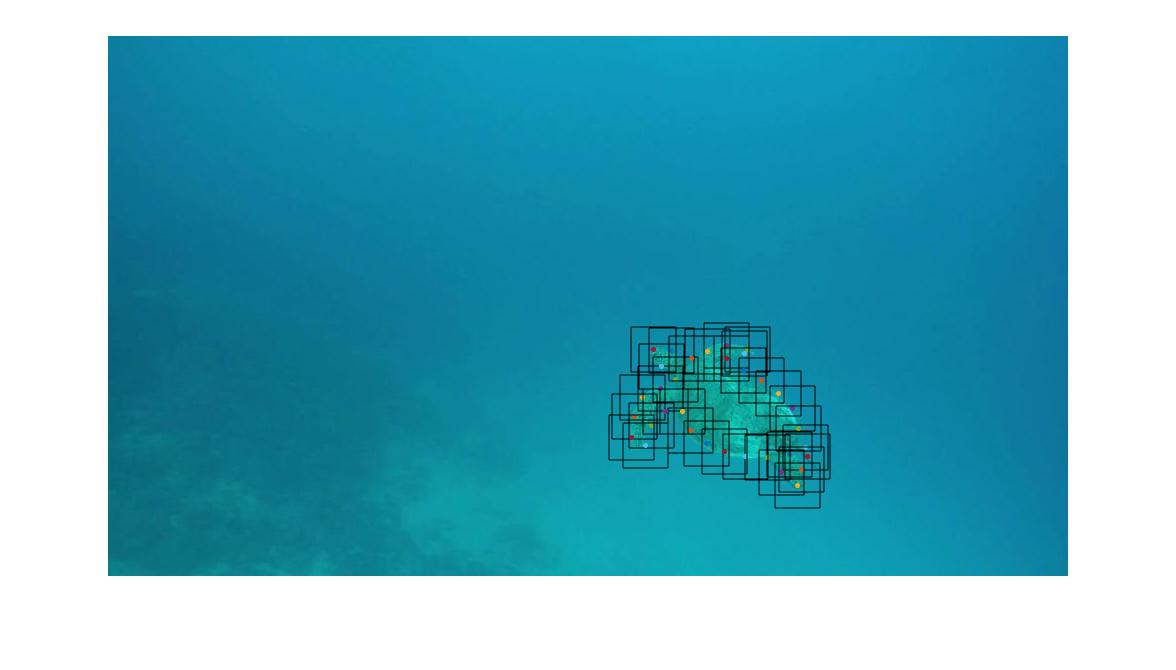
For implementing rotobrush, we followed the systems diagram that was given to us in the slides:



**Setup Local Windows**

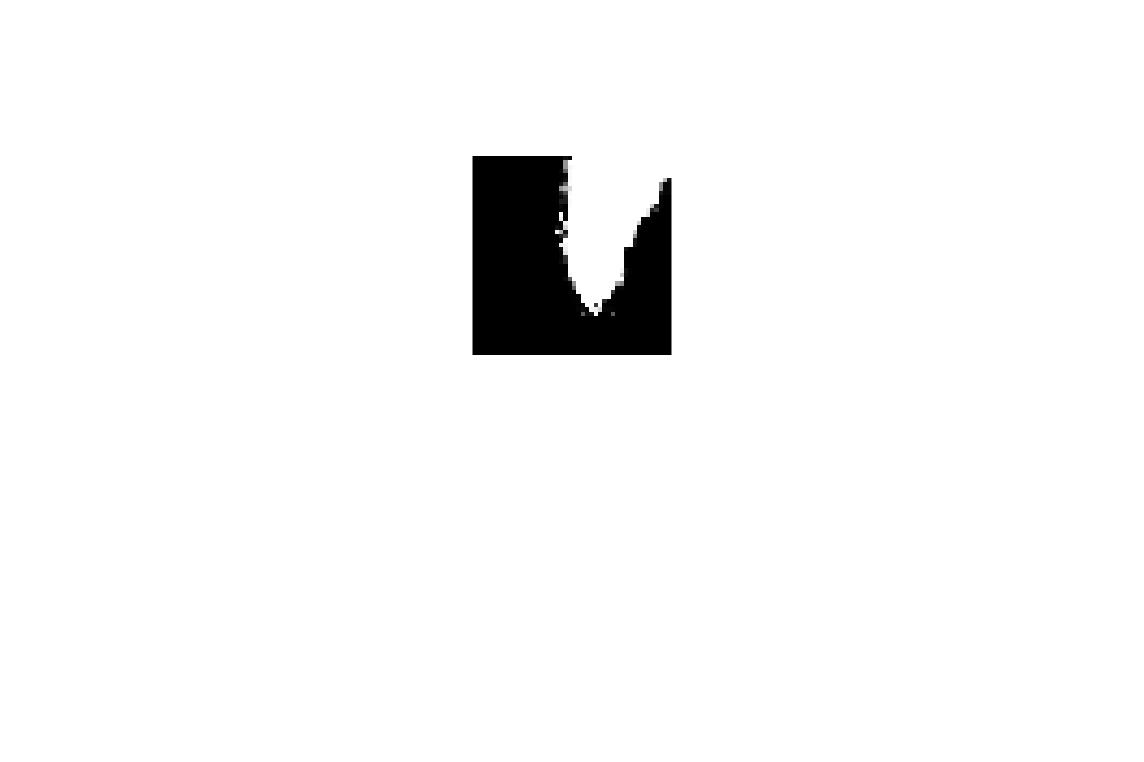
To initialize the shape/color models, we needed to create local windows along the outline of each figure. Luckily, we were given initLocalWindows.m so all we had to do was use roipoly to make the mask. We chose to have 35 windows with a width of 45 pixels because we learned that we wanted to have as many points as possible without slowing the runtime. Below is an image with the local windows we had for the turtle. We initially had a width of 30 pixels because we wanted about 1/3 of the window to overlap with another, but that caused the output mask to have a hole in it so we increased it to 45. We also initially had 40 windows, but that took too long to run.



**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 because we realized that we still needed those. For the math, we followed what was in the slides to calculate foreground probability and confidence.

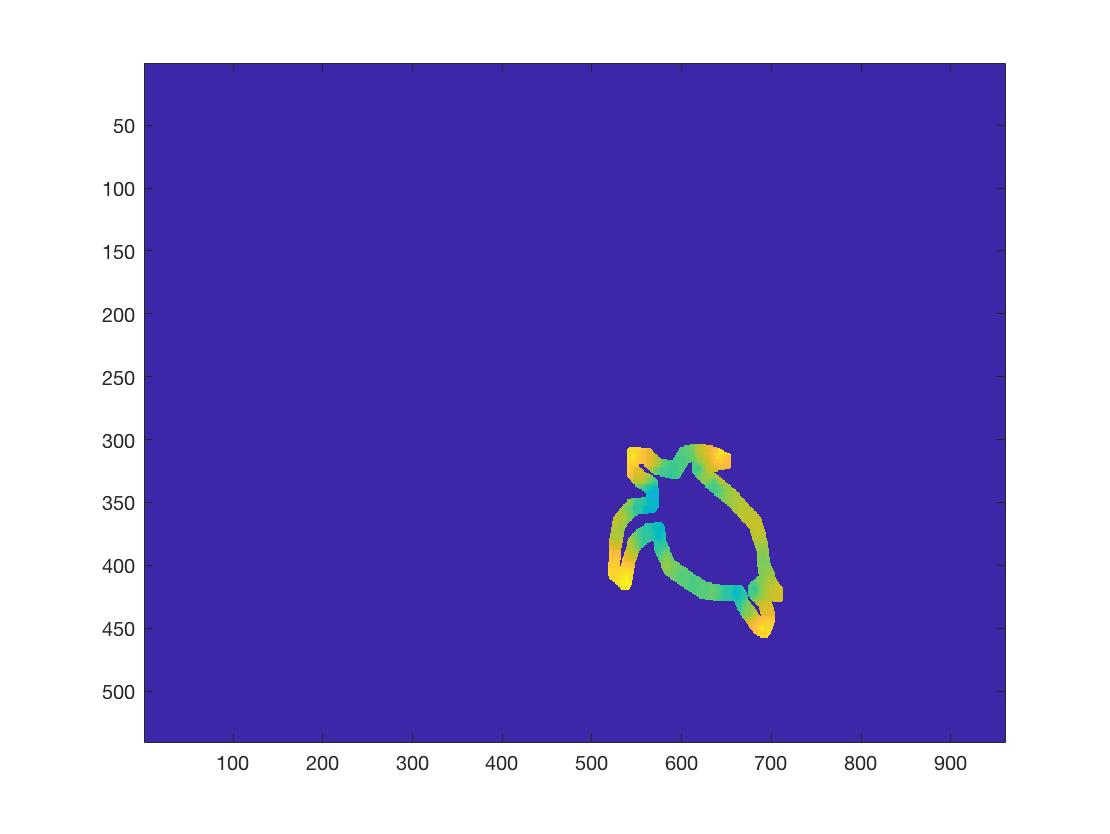
Here is an image of a window on the left and what we calculated as the foreground probability on the right for a local window of the turtle:



It’s a little off, but still retains the main shape.

**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 well with the results of the color model. Below is an image of the color confidence.



**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 0.85 for , 2 for , 2 for R, and about 1700 for A. We got these numbers from the paper and a little bit of fine turning.

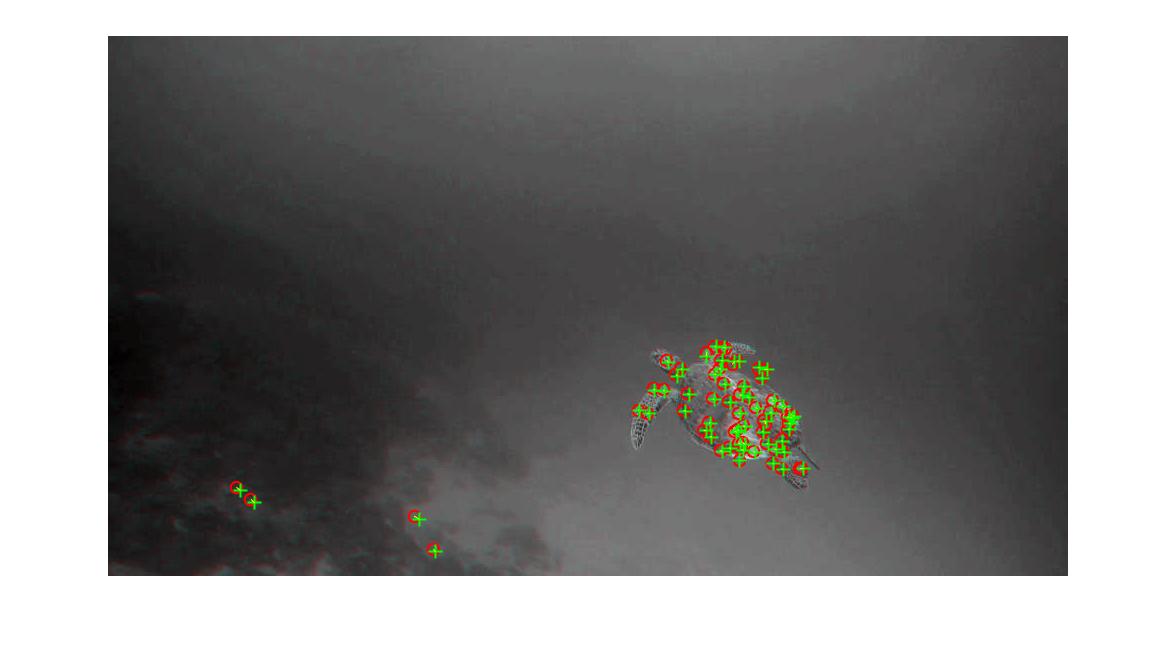
Here is an example of a shape confidence we calculated:

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

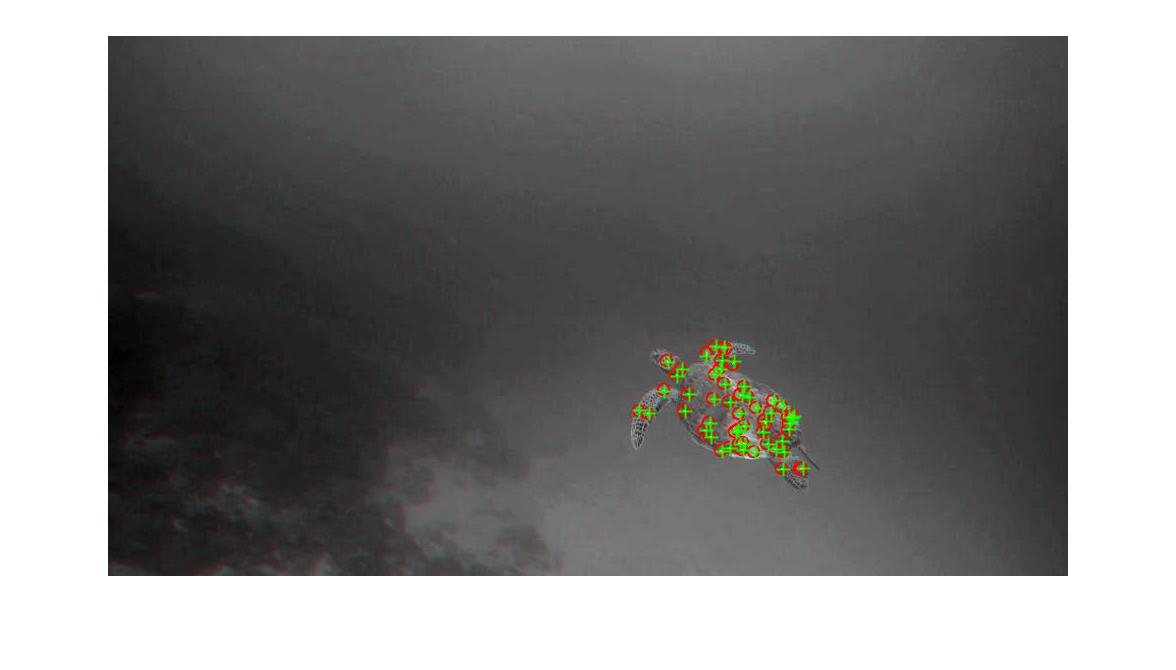


**Estimate Entire Object Motion**

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.



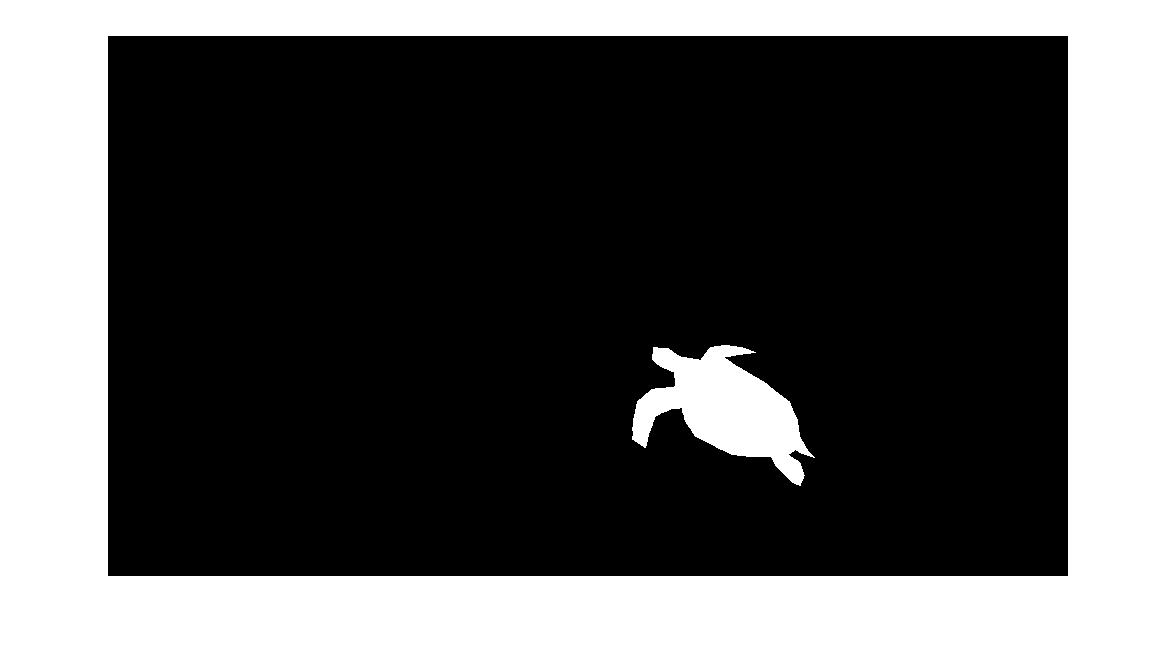
As you can see, there were some features that shouldn’t have been there on the left. We then filtered those out by checking if it was in the mask. The image below is what we got after this filtering.



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

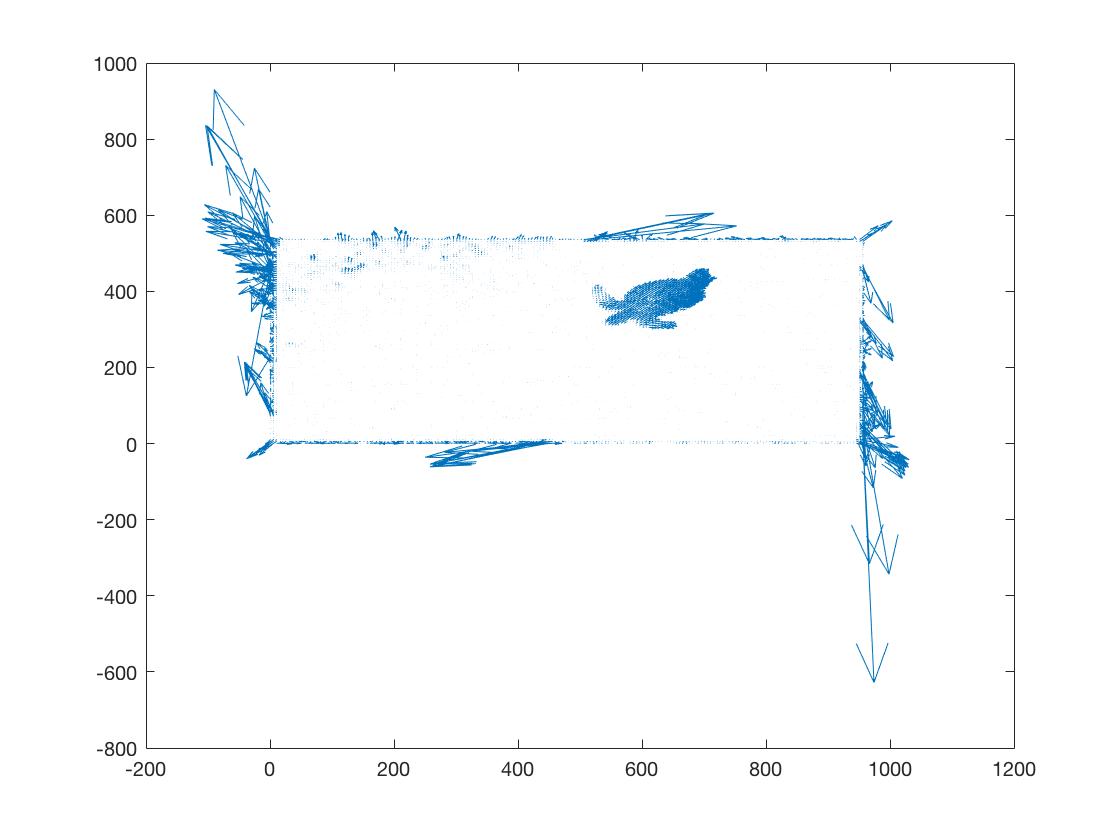
Below are images of the warped frame and warped mask, respectively.



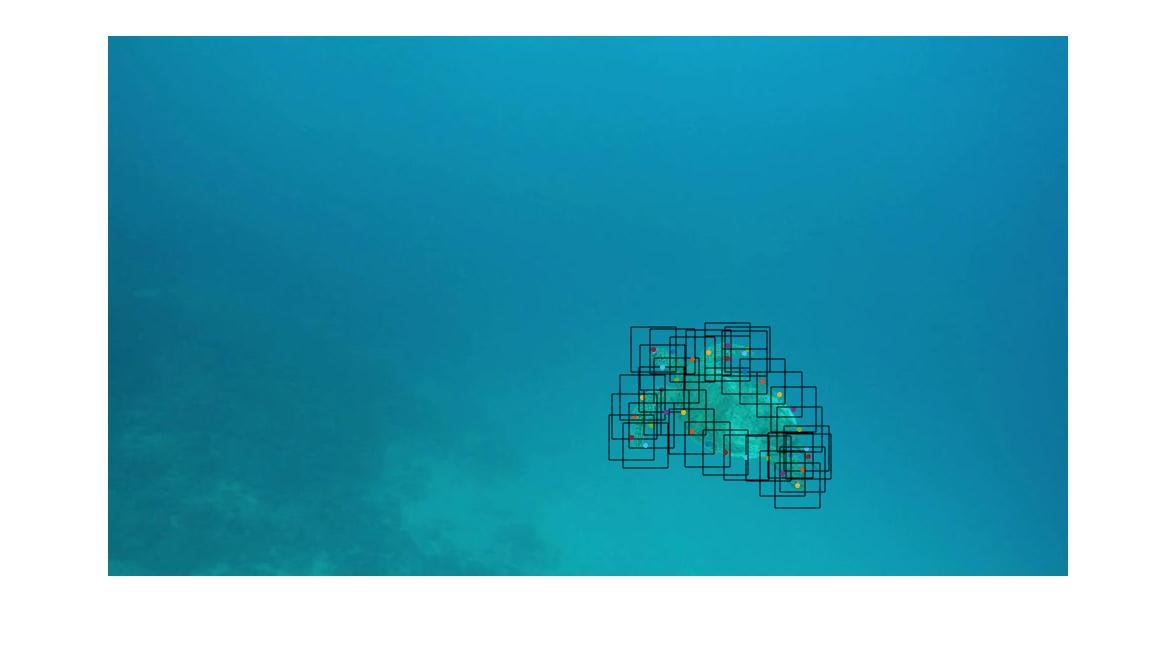


**Estimate Local Boundary Deformation**

For localFlowWarp, we used estimateFlow and opticalFlowFarneback. Below is an image of the optical flow object.



Then, we used Vx and Vy to get the new local windows from the old ones. We didn’t have any problems with this. Below is an image of the new windows.



**Update Color Model (and color confidence)**

For updating the color model and color confidence, it was more difficult because there was less information in the slides. We went through the paper and attempted to follow their algorithm. For the color models, we ran the initialized GMM on each local window. Then, we got the foreground probability on each pixel and chose pixels that were 75% likely to be in the foreground (that’s the number they used in the paper). We added those new points to the points that were used in the GMM and then fit a new GMM. Then, we ran GMM again with those new points and got new foreground probabilities for the windows.

We also calculated a new distance based on the warped mask outline.

Then, we had to decide if we wanted the new GMM or the previous GMM. To do this, we just checked how many more points we characterized as foreground points. If it was reasonably bigger, we took the new GMM and calculated new confidences.

**Combine Shape and Color Models**

For combining the shape and color models, we initialized a new shape confidence with the new color model. Then, to combine the models, we looked at each window, and used the warped mask, shape confidence, and color model probability. We used from the paper for this.

**Merge Local Windows**

To merge local windows, we went through each window and used from the paper.

**Extract Final Foreground Mask**

For extracting the final foreground mask, we checked pf from the merging and saw if it was over our threshold of 0.5. We then used imfill for each window because we had a lot of holes initially, and then again for the entire mask.

Results

Change around parameters and test file each individually

Finish report and make sure it has everything - Talk more about problems and successes

While the foreground consistency assumption could be violated by luminance variance (this can be addressed by illumination-invariant features instead of Lab), foreground self-occlusion, or object rotation, we found this assumption to hold true for most common scenarios in natural videos, and it helps to significantly improve the