Pyone Thant Win, Eric Han
Group 16
CMSC426: Computer Vision
Professor Yannis Aloimonos
November 23, 2021
Extension granted until Nov, 19; Using 4 late days

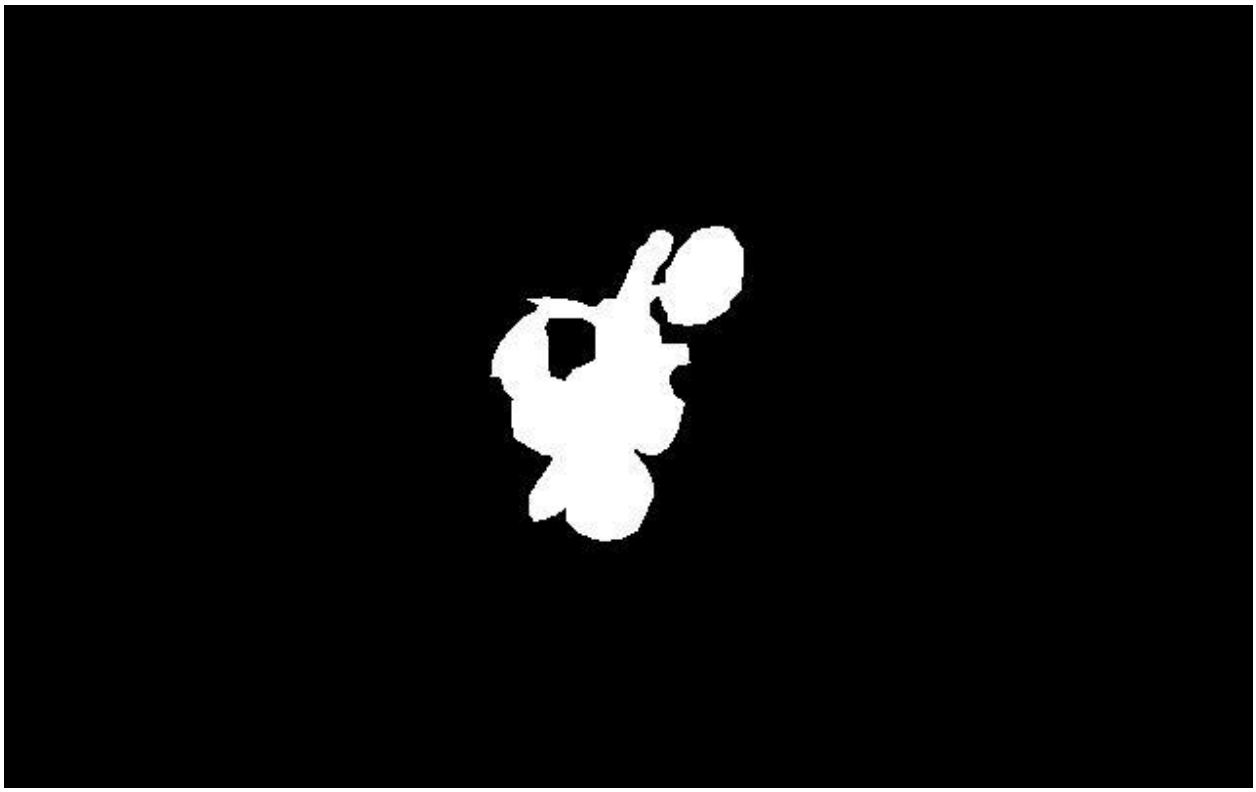
Project 3 Report

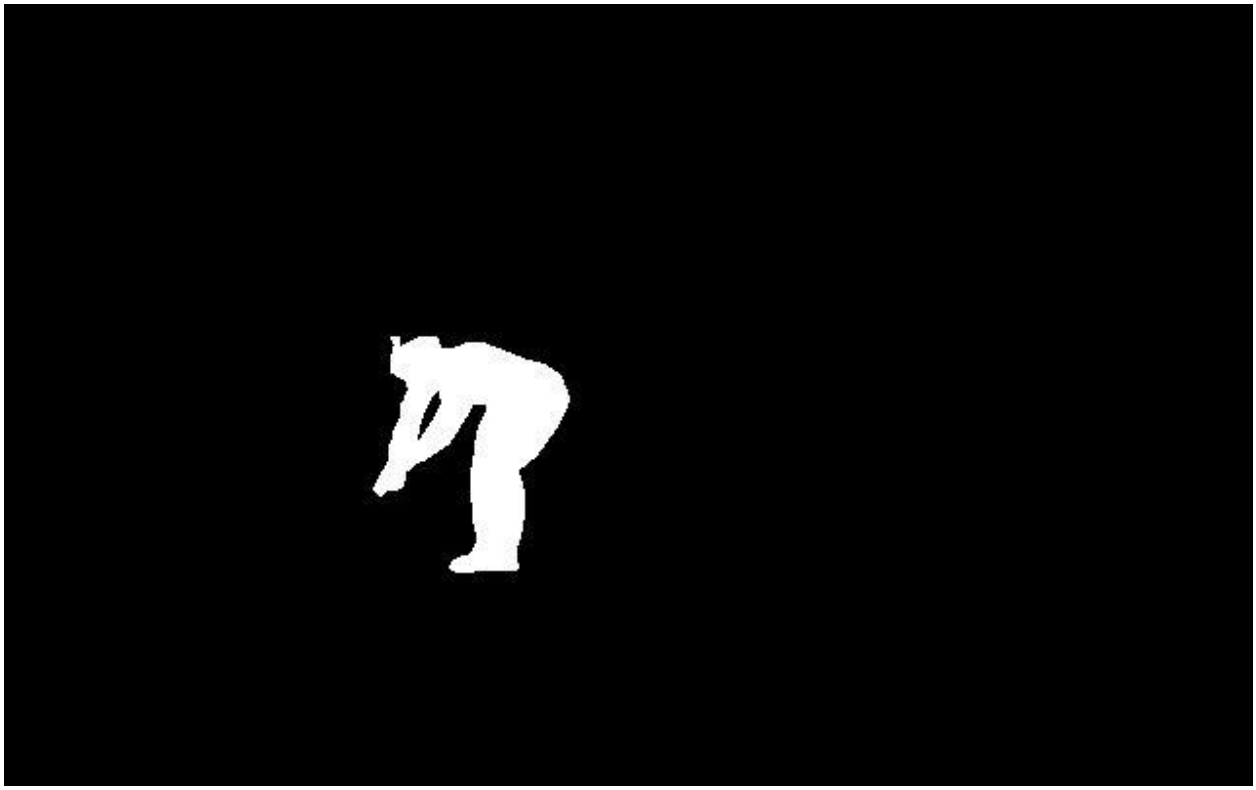
Who Does What

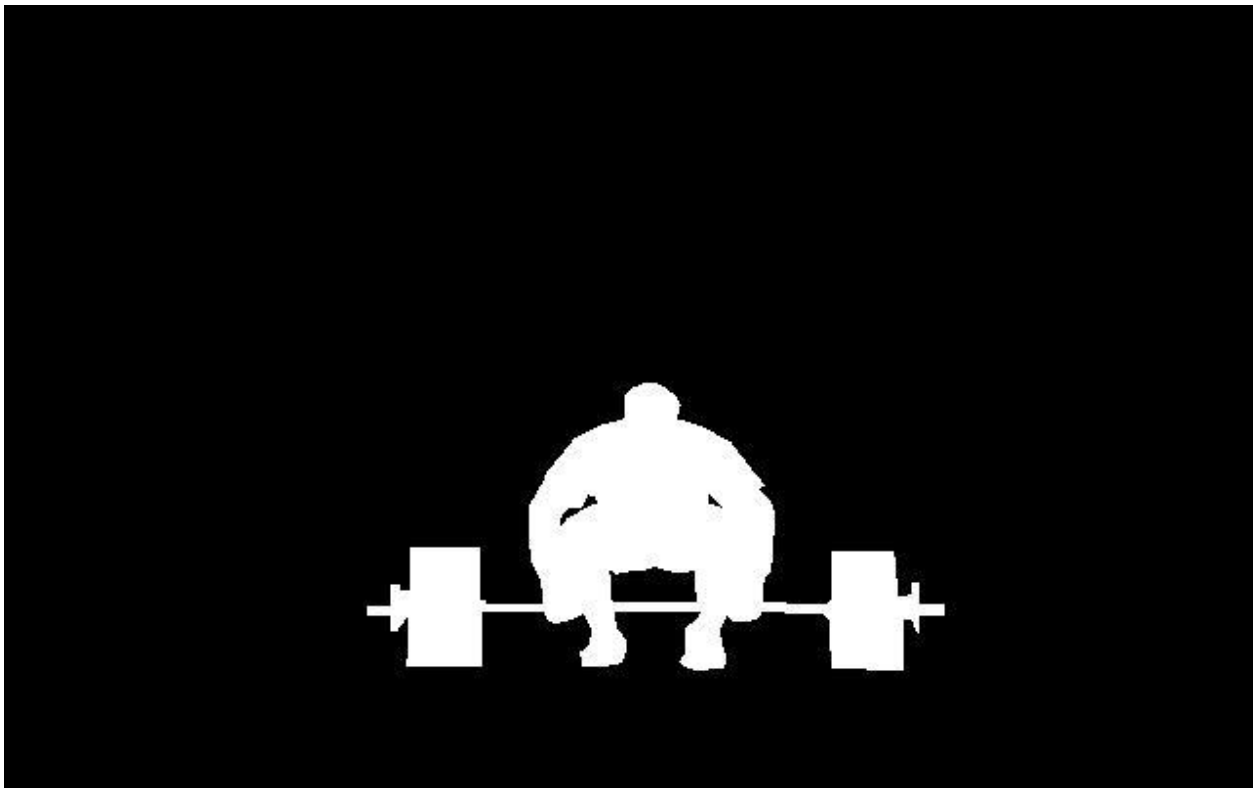
1. Creating masks: Pyone
2. myRotobrush.m: Pyone, Eric
3. initColorModels.m: Pyone
4. initShapeConfidences.m: Pyone
5. localFlowWarp.m: Pyone
6. calculateGlobalAffine.m: Pyone, Eric
7. updateModels.m: Pyone(Update Local Classifier, Updating the Shape and Color Models), Eric(Merging Local Windows)
8. Note: old team member Andrew Sarama worked on initColorModels and initShapeConfidences, but I(Pyone) changed a lot of his code to make it actually work.

Creating Masks

We used roipoly and manually created masks for four sets of inputs. I first created the mask with some background region we could not reach with 1 roipoly. Then, created another mask to cut out those, as shown in the first, second and third ground truth masks.





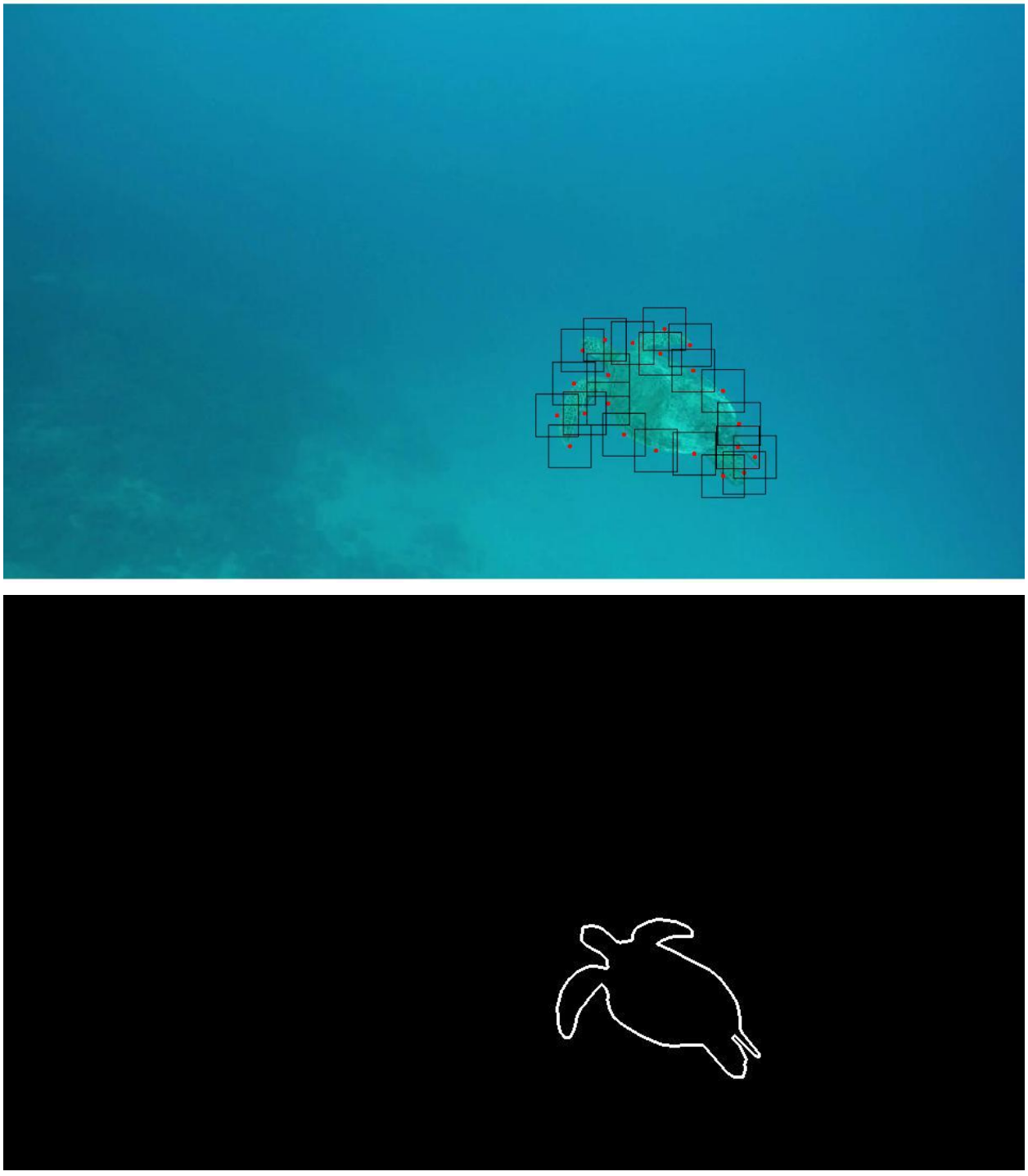


Setup Local Windows

Using the input ground truth mask, first frame of inputs, the number of windows and the width of each window, we generated local window centers. Since the code for generating the local windows was given as the starter code, there was no implementation included in this part. However, we found that 40x40 window size with 23 local windows worked best for the input set
1. For frames 3, we used 40x40 window size and generated 25 windows.
2. For frames 5, we used 40x40 window size and generated 22 windows.

The starter code also creates the mask outline. With this result, we increased the thickness of the mask outline based on the BoundaryWidth variable. We used boundary width of 3 for the turtles input.

Below are images of frame1 window and mask_outline with applied boundary width.



Initialize Color Models

We created a *struct* variable to store information about Color Models. It includes Color Confidences, foreground and background GMMs for each window and foreground probabilities of all pixels in each window. In order to calculate the foreground and background GMMs for each window, we first looped through all the pixels in a window, check it with an input mask. If mask(y,x) is 1, we save the color RGB values in the foreground colors array, and we save those

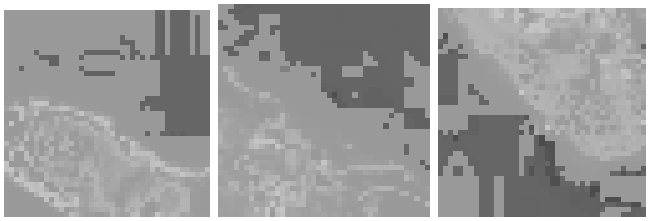
with mask(y,x) is 0 to the background colors array. We change RGB values of the pixels to the LAB color values before adding it to the respective arrays.

Then, we calculated the GMM for foreground and background. We have three ellipsoids for each GMM. For the windows that are unable to converge the GMMs with 3 ellipsoids, we regularized the data to overcome that error. We used a regularization value of 0.1).

Compute Color Model Confidence

Once the foreground and background GMMs are obtained, we calculate the foreground and background probabilities of each pixel, and we incorporate these values along with the distance of the pixel to mask outline to obtain the color confidence of the window.

Color Confidence of frame 1 of the turtle input is as below:



Initialize Shape Model

Next, we calculated the shape confidences of all pixels in all local windows. For Frames1 set of inputs, we used fcutoff = 0.85, SigmaMin = 2, SigmaMax = WindowWidth, and R = 2. A value is calculated using aforementioned values. We obtained these values from the paper.

Compute Shape confidence

Shape confidence is computed by subtracting the exp(-1 * distance of pixel to the mask_outline^(2) / sigma_s^2) from 1. Shape confidences are stored as *struct* and it contains the shape confidence probability of all pixels of all windows. These are the shape confidences of a few windows of the first frame of frames1 set.

Estimate Entire-Object Motion

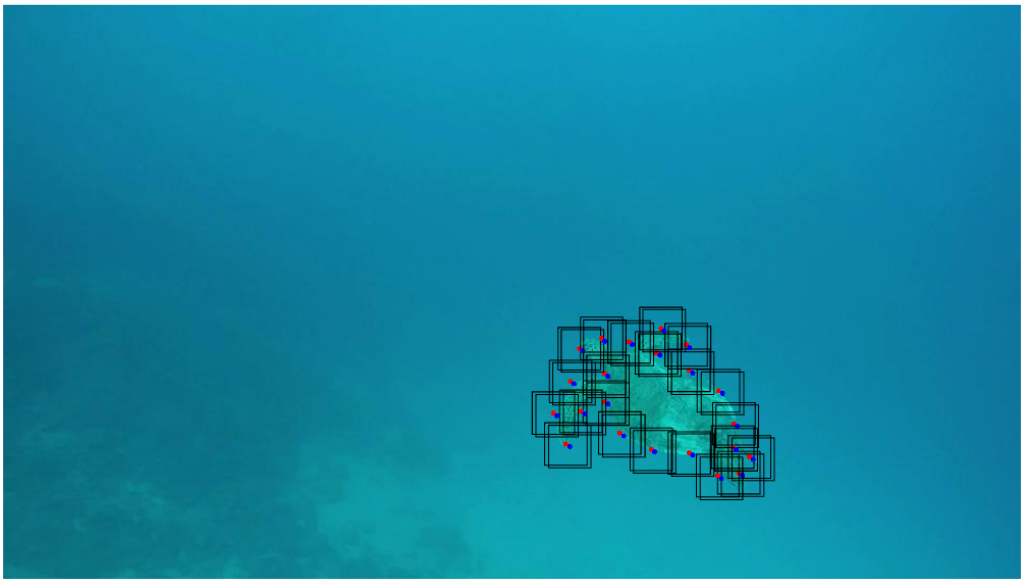
This involves taking a before and after frame, and then calculating the affine transformation between the two.

To do this, we first identified feature pairs in both frames and found the affine transformation matrix through them. We then apply this transformation to the current mask, frame, and local window center pixels to obtain the warped versions of these.

Warped Frame 1:

Estimate Local Boundary Deformation

As the author mentioned, using warped version of previous frame is not a good estimation. To get a more accurate displacement of the region of interest between previous frame and current frame, we use optical flow. We used opticalFlowFarneback since it provides bigger values. We calculated the average value of Vx and Vy of pixels in the foreground with information from the mask. Then, we added it to the old local window center coordinates to get new ones. When the average value of Vx and Vy less than 0.1



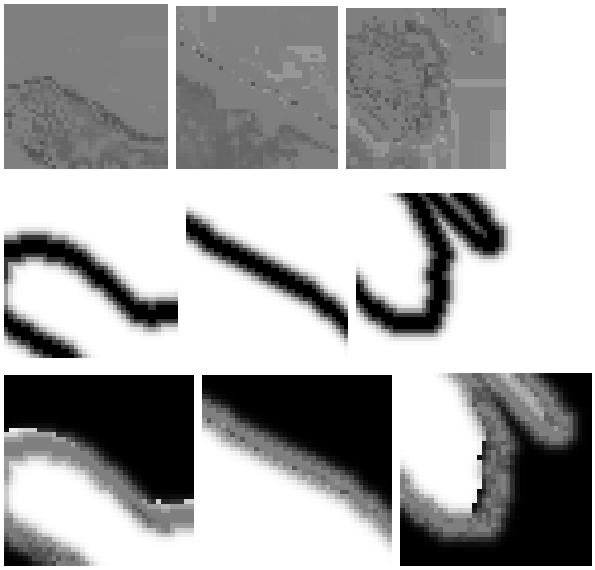
Update Color Model (and color confidence)

Using old shapeConfidence values, we checked the color distributions (foreground and background) of new windows. We used the threshold values mentioned in the paper (greater than 0.75 for foreground and greater than 0.25 for background). To ensure that newly calculated

GMMs are not drastically different, or that new GMMs do not perform worse, from old ones, we compare the number of foreground pixels in the new window. If the total number of foreground pixels in the new window is greater than the last frame's window, it means the new GMMs are not optimized. Therefore, we use the old GMMs for that window. Otherwise, we use the new GMMs. If we use new GMMs, we recalculate the shape confidences of that window.

Combine Shape and Color Models

In this step, we merged the foreground maps by using the newly produced color and shape models. Essentially, we are putting more weight on the model that is more confident. After this step, we obtain foreground probabilities of pixels in all windows.
Here are new color and shape confidences, and foreground probabilities (after *frame 2*):



Merge Local Windows

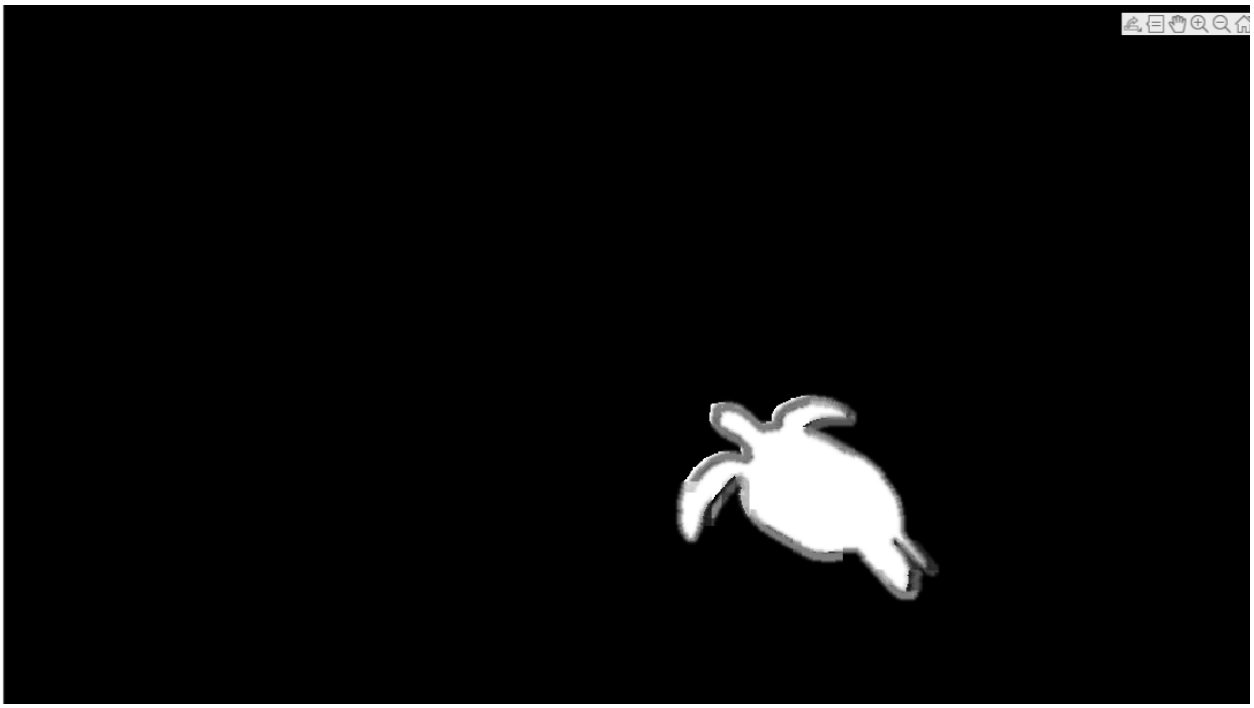
We now create a real foreground probability mask of the . We loop through every single pixel of a mask we created and see if that pixel is inside any of the foreground probability masks of the new local windows. If they are, we loop through all new local windows and do a summation using their foreground probabilities along with the distance from the current pixel to the corresponding local window centers. We made a distance method to compute this that takes in the x and y coordinates of both pixels.

If the pixel was not in any local window, we thought it would automatically become 0. That was not the case, as there are pixels inside the foreground that also are not in any local windows. Therefore, we used the value of the pixel in the warped mask to get the proper value.

Something we could have done differently was the method of seeing if the pixel is in any local windows. We decided to loop through all local windows, but since there are no cases where 3 or more windows will overlap in the same area, the only local windows that could overlap with each other are the next local window and it's mirrored local window because our local windows will be

created in a clockwise manner. Therefore we could only check those windows instead of looping through all local windows to check for overlap.

Ex: Frame2 of turtle



Extract final foreground mask (and Lazy Snapping)

Our methodology was to loop through every pixel in the real-valued probability map created after merging local windows, and seeing if that value is greater than the foreground probability threshold. If the value is greater than the threshold, then that pixel is part of the foreground. If less than, then that pixel is part of the background.

Ex: We changed the threshold from 0.75 to 0.60 and the right foot of the turtle got a bit more pixels, but was then cut off in later frames.

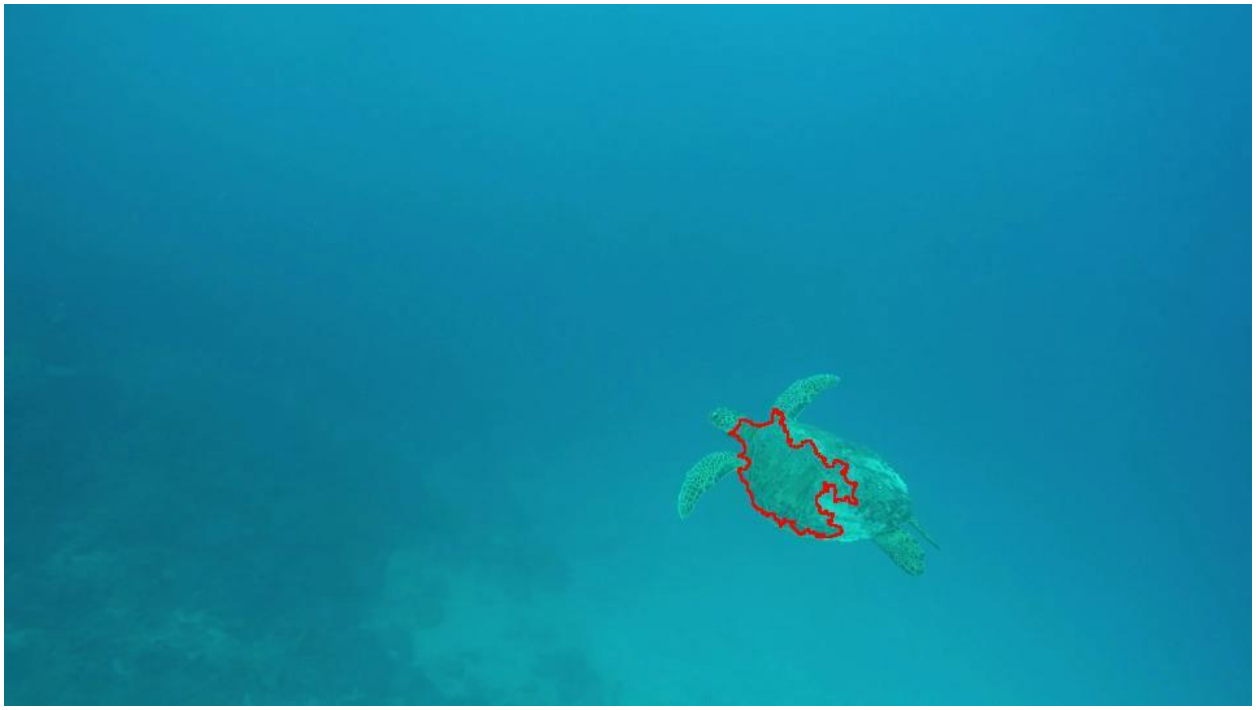
Our end result masks have some small bits of the foreground cut off, such as the right foot of the turtle after the second frame. By decreasing the threshold, more pixels will come into the foreground, but only a bit more. We had to be careful adjusting the threshold though, as the threshold should be the same number for calculating the color confidence and shape models. The mask also had some issues, with some foreground pixels being located slightly outside the foreground. We attempted to increase the threshold to get rid of some of those pixels, but it didn't work entirely.

Another issue with the mask was that it kept shrinking and cutting off parts as the frames progressed. At a certain frame, GMM would not work anymore. The increasing of the threshold also may have contributed to the mask shrinking.

Although we implemented lazy snapping, we did not use it. The reason is that since superpixels creates random boundaries, it does not perfectly cover all regions in the foreground. We experimented using 4000 superpixels, but masks would disappear after 9th or 10th frame.

The result, without using lazy snapping, gave us a slightly better mask, but it still shrunk as time went on. GMM would eventually still fail. However, it is important to note that our new local window centers are still maintained around the boundary of the region of interest.

Here are a few output frames that appear in the output video (with and without lazy snapping):
With lazy snapping (overlay results of frames 7 to 10):



Without lazysnapping (overlay and mask results of frames 7 to 10):