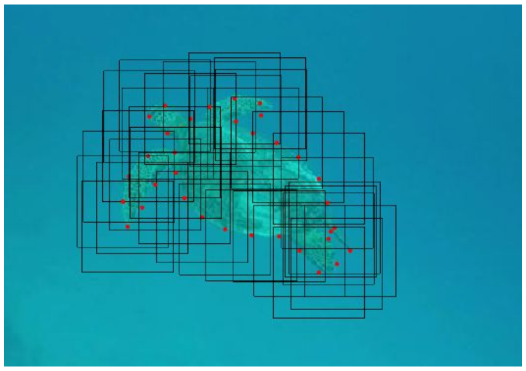
Project 3: Rotobrush

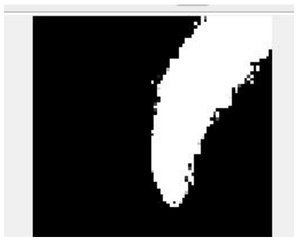
Christopher Yue, Acheev Bhagat & Vishal Hundal

Using 2 Late Days + Extension

Introduction
For this project, we implemented Adobe's SnapCut algorithm that uses local windows to get readings of object shape and color. This was then used to isolate and cut the foreground from the background of a video, based on the frames. To start off our implementation of the project we used roipoly, a built-in MATLAB function, to create a mask around the foreground on the first frame. After this, we would plot multiple windows of each NxN size around the border of the mask. The windows, which overlap, are then used to capture local foregrounds and backgrounds for each frame of the image. This combined with the subsequent steps allowed us to crop out a deformable object from a video. This document below describes our implementation as well as the major problems we faced.

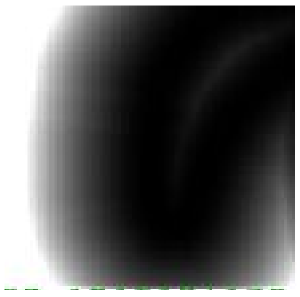


Initialize Color Model
The color model in SnapCut was used to determine which pixels were in the foreground and which were in the background, based on their color readings. For the color model, we used the L*a*b* color space instead of RGB to create the Gaussian Mixture Models because it separates color from lightness, allowing us to minimize the effect of lighting differences for our color model. In this section, most of our difficulties came from the fact that the GMMs wouldn't converge. We found out that we could add a "MaxIter" parameter and regularize the values, so we set it to have a max iteration number of 1,000 and a regularization value of 0.0005.

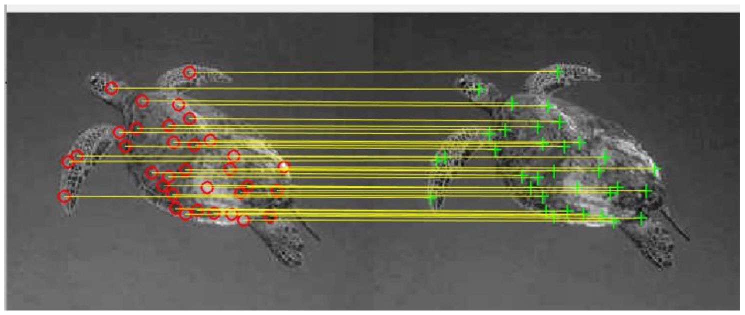


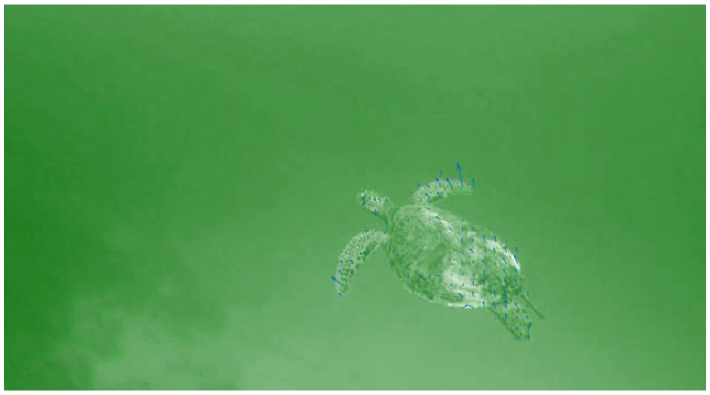
Color Model Confidence
Color model confidence is then used to determine how reliable our foreground and background separation is. In other terms: the confidence represents how separable the local foreground is against the local background using only the color model.

Initialize Shape Model
In this section, we developed code to create shape models for each of the windows in order to model the shape of the boundary rather than the colors around it. Here, our main issue was not knowing what SigmaMin, A, fcutoff, and R were, but we quickly learned what these parameters were for by referring to the research paper.



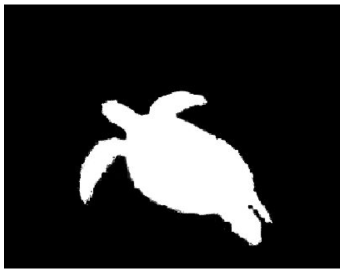
Motion Estimation
We then estimate the motion of the object by estimating the affine transformation from one frame to the next, in order to account for large, rigid changes between the foregrounds of two frames. Initially, there were too many matches between the two frames, which ruined the estimated transformation. We solved this issue by limiting the matches to only ones within the mask. This then allows us to find the optical flow of the object in the foreground. For each window, we calculate the average of the optic flow vectors and use this average to shift the center of the window to account for fine movements of the foreground, such as when a person raises his or her hand. This is the section of the project where we came across the most errors. One problem we had was with the global affine. For one the affine never operated past the first set of windows. Only the first affine was calculated and it was never changed. This error was caused because the set of warped windows was never updated. We fixed this by saving the warped windows as a local set of new windows and warping those windows on the next iteration. Another issue we have is with our flow calculation. We tried to alleviate this at office hours, though it was to no avail. The problem with our flow calculation is that the calculated flow between the frames is too small to affect the motion estimation. Effectively, our motion estimation only works for rigid motion. Furthermore, we ran into numerous errors such as non-finite values and index out of bounds. We were able to solve the non-finite values error by changing all NaN and Inf values to 0, but we were not able to find a way to solve the out of bounds errors (even with padding) given the time constraints of the project and the fact that all of the sets of frames took a very long time to render.





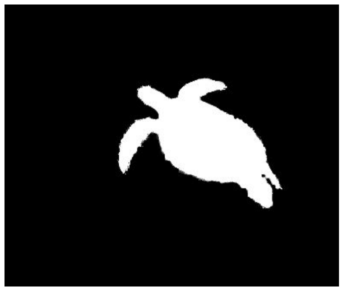
Updating Color and Shape Models
Given that the next frame has new local windows, we have to update the shape and color models to account for this. The affine transformation can be carried over from the last frame, but for the color model, we create two new GMM models for the foreground and the background for the last and current frame. We only sample certain pixels above a certain user-set threshold number. While updating the color model, we must also update the color confidence values as well.
In this section, we had issues with index out of bounds errors, but we fixed this by making sure to pad everything.

Combined Probability (Looks almost the same as the final mask for this set of frames)



Extracting the Final Mask
We now have a probability map for each window that tells us where the foreground is most likely to be found. We combine all of these maps together to get a global foreground map. This gives us a decimal-based probability mask. To convert this to a binary mask, we used lazysnapping. We did this by cropping the current frame and mask, and then passing that into MATLAB's built-in lazy snapping function, along with the foreground and background mask. We decided to crop the frame to reduce the runtime and increase the efficiency of this part of the implementation. This allowed us to create a binary mask, dependent upon if the probability was above a certain threshold value. This was then used to cut out the foreground of the image from the background in a live moving video, completing the Rotobrush algorithm.

Final Mask



Extra Credit: Stabilizing the color model
We implemented this section by altering updateModels. Instead of considering the previous frame's GMM along with the current frame's GMM, we keep propagating the first frame's GMM until there is a significant difference between the colors in the first frame and the colors in the current frame. This difference is determined by getting the probability mask of the current frame using both the first frame's GMM and the current frame's GMM and seeing if the number of foreground pixels from the current frame's GMM is greater than that of the initial frame's GMM by a user-set threshold. If it is over the threshold, the user is prompted to use roipoly to re-mask the image, thus creating a brand new mask and color model. Now, instead of propagating the initial frame's GMM, this new GMM is propagated, and the process cycles through the remainder of the frames.

Drawbacks of our implementation:
The largest drawback of our implementation of SnapCut/Rotobrush is that it is very dependent on user-set thresholds, meaning that the algorithm has the possibility of either working very well or very poorly with some given thresholds. With incorrect thresholds, our implementation returns results that are very poor. To ensure quality results, the parameters and thresholds must be very finely tuned for each set of frames. The scope of this issue can be highlighted as thresholds/parameters had to be set for initializing the color model, motion estimation, updating the shape and color models, extracting the final mask and to stabilize the color model. The extra credit portion, stabilizing the color model, that we implemented, may also be a drawback of our implementation, given that when the difference between the current and original color models is too high, roipoly is run, requiring user input. While this may give better results, it reduces the robustness as user input is required to recalibrate the algorithm. Also, using our algorithm on

objects that are not uniformly colored spinning on the central axis would fail, given the colors in the previous frame may be drastically different from those in the current one. Another drawback of our implementation would be cutting out objects that seem to move in a seemingly random manner, as they would skew our flow and transformation. If we had the ability to fix the flow warp and improve upon the color model to make it more robust, then our results would be better. Given that our implementation is quite basic compared to Adobe's, our algorithm is not as robust. Additionally, we ran into other problems such as having an almost singular mask passed into lazysnapping for certain sets of frames and having inaccurate foreground masks. Our only solution to the former problem was to remove lazysnapping for these certain sets, but we had no solution to the latter. The latter problem was most likely a result of the window centers not updating properly. Having inaccurate windows leads to having inaccurate color and shape models, and thus an inaccurate final mask.

Unfortunately, we were only able to render videos for 3 out of the 5 sets of frames, as we received errors that we couldn't seem to fix for the other 2 even after spending multiple days trying to.