William Schulmeister
Sarah Bang
Yashashri Pendse
Brian Chin

Project 3 - Rotobrush

*USED **5** LATE DAYS*

Processes

Incremental output is shown in the next section where we comment on its faultiness.

Parameters

1. **Initial Mask**

The initialized mask was formed with a call to roipoly with an included ROIPOLY flag to use the stored mask or reform. As always, the user-dependant mask is a potential weakness of the algorithm and provides a level of variability.

2. **Global Params**

The global variables were tuned throughout the process to review various results and outputs. As described in the later section, we saw limited results so the variables were tuned rather arbitrarily past constraints for the object coverage and speed. In general, no changes were made to the file structure from the initial download.

For most development, these params were selected as satisfactory place holders: REG = 00.1, NUM_GAUSS = 3, WindowWidth = 30, Windows = 5, BoundaryWidth = 5 and ProbMaskThreshold = 85. Our call to the setup for LocalWindows was also the same as the initial file structure barring the error mentioned in Piazza.

Init Color Models

For each window and each pixel in the window's space, we distinguish the foreground from the background using the user-drawn ROI. Using this segmentation, we compute the GMM for both foreground/background using the RGB values of the image in the LAB color space. We store prob, gmm_f, gmm_b, dist (d_x), and the found pixels in the ColorModel struct. Finally, we use the probabilities we gathered from the previous steps to calculate the confidence for each window, stored as the Confidences component of the ColorModel struct.

Worth noting is the fact that the entire algorithm is heavily reliant on the parameters and models developed at this step. The user-defined nature is then a risk for the overall quality.

Init Shape Confidence

For each window, we fetch the dist and confidence parameters from the ColorModel struct. Using the confidence of color model, distance, and sigma (SigmaMin arg), we follow the provided formula to calculate the shape confidence. Like the previous step, the necessity of user-provided fcutoff, A, R and SigmaMin parameters provides even more variability. In a more complete system, these variables would be tuned based on the input and not trial and error.

Affine Transformation

The algorithm then finds the matched Harris features between the previous grayscale frame and the next. The estimated geometric transform is then applied appropriately to the image, frame and mask. This step is largely by the book, but the calculation of features across the whole image and not just about the masked area could create problems in busy frames with unforeseen edge cases.

Flow Warp

Before identifying optical flow, we define a new ROI which is simply the mask and any pixel which is less than 30 away from its border. This optimization is to prevent unforeseen flow from foreign objects. Although 30 is arbitrary, we found that it was sufficient given the small differences that were possible between frames.

Additionally, we used the method of applying the mean flow to the windows to find their new location. Since we were confident that the flow was limited to mostly the object, this was the approach we chose. A more complete system may look to take out outliers or, at the very least, use a more robust calculation. If speed is entirely not a concern, changing window location may be done on an individual basis where the closest vectors are taken into account.

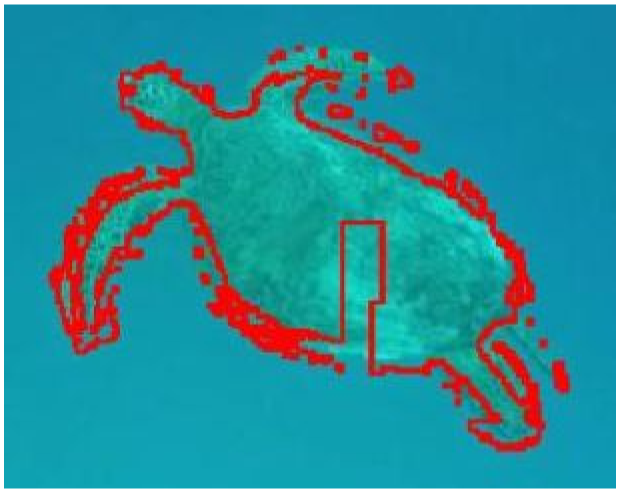
Updating

The first several sections of the updating procedure were rehashes of the processes in initShapeModels and initShapeConfidences, following the updating procedures outlined in the documentation. Worth noting is the fact that we chose to use a threshold of .6 to to form the final mask. This decision was made because of the poor results from lazysnapping using the parameters we tried. This was likely due to insufficient space coverage with regards to window width and window number, but we were limited to our laptops for running the programs.

Output and Notes on Completeness

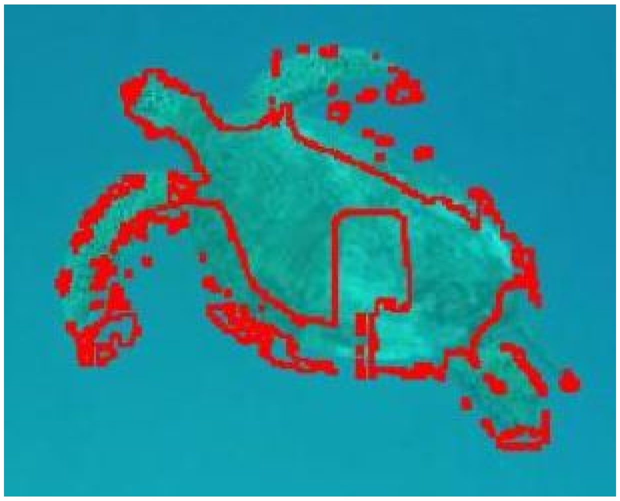
The output below isn't complete, but we believe it demonstrates understanding and a closeness to the accepted solution. We could not run all the frames because the runtime took too long and was often abruptly stopped by an error in the provided getFrames procedure.

Turtle and Comments

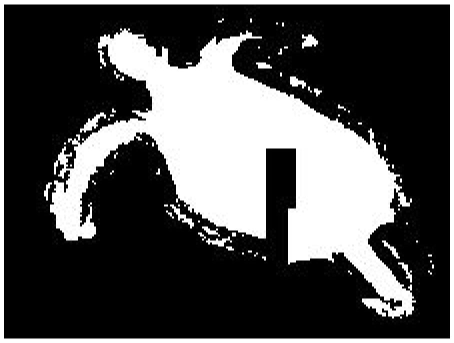


The output of the second frame of the first iteration of the main loop is shown to the left, done with parameters: 5 GMMs, 7 windows and 100 pixels window width. The sizes are not big enough to mask the entire turtle in this specific case, but these images are just to demonstrate the problems.

As in similar cases with smaller parameters, our algorithm shows a cursory ability to segment the turtle's movement and draw a rather shaky outline. The small errors are exacerbated with every subsequent frame, as seen below on the very next frame, so a complete video is impossible given our current implementation. Specific to the turtle, the algorithm may struggle with the similarity of the foreground and background. Plainly, there is one major flaw with the output as a whole:

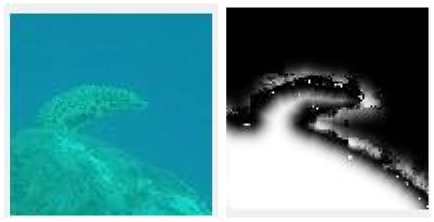


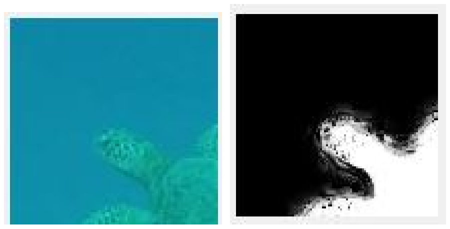
Inconsistent Border from the Integration Step



As can be seen on the right, the algorithm could not develop strong decisions along the outer border of the turtle, even if its location was accurate. While it is possible that this inconsistency was due to poor parameter choice (a likely part of the problem), it is more accurately due to flaws in our update algorithm, which needed to consolidate color and shape confidence into a strong outline.

As can be seen below, the algorithm actually did manage fair results for some calculations of shape confidence, but these results were not consistent enough for a completely functional segmentation. Both pairs of results were taken after the integration step of updating from frame 1-2.



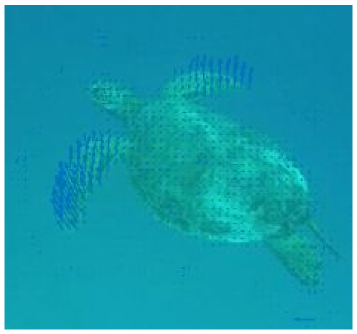
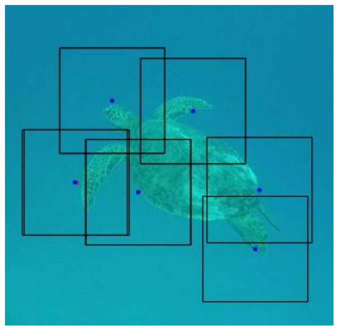
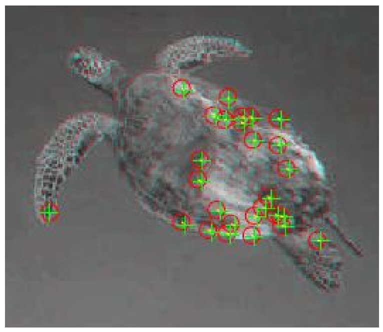


The pair on the right shows relatively strong integration. The shape of the turtle is clear and consistent with the colors and clarity expected. Given results only of this nature, we would’ve been able to find the best parameters to maximize the strength of our foreground mask. Ultimately, it was matches like the one on the left that demonstrated the problem. Unlike the successful segmentation, the left pair was mostly segmented incorrectly along the edges, with apparent disagreement between the confidences resulting in a strange white-black alternation in the output mask. Although this specific issue is remedied in the extracted mask with a call to imfill in this specific case, problems demonstrated in this step are repeated in future instances, forming less salvageable masks.

Notes on Flow and the Affine Transformation

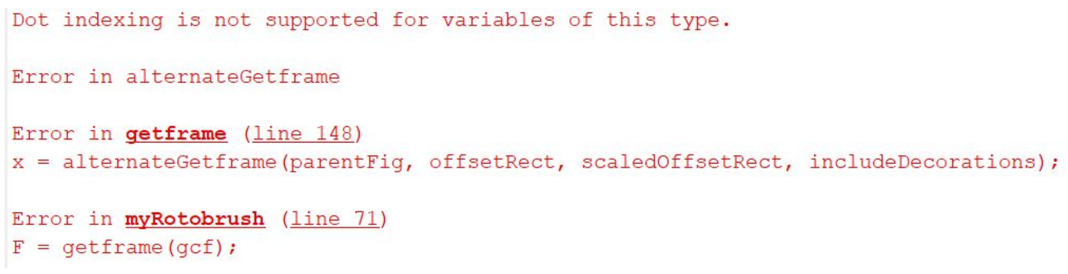
The output for flow and updates to windows seem to be realistic and consistent with the expected results. As seen below, the changes to frames are relatively identical between both methods, a sign of a healthy algorithm.

Note: We are unsure as to why a window gets dropped in our runs when outputted. We do not, however, expect this bug to affect the comments made here and elsewhere or the accuracy of the totality of the algorithm.

GetFrame Errors

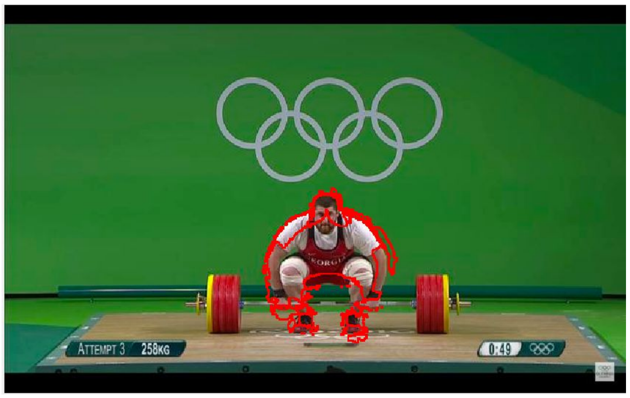
Frequently, we would get the error below. Its appearance was random, but we speculate it had to do with the parameters chosen. Ultimately, it affected our ability to run long processes as it would inevitably show up without explanation.



```
Dot indexing is not supported for variables of this type.

Error in alternateGetframe

Error in getframe (line 148)
x = alternateGetframe(parentFig, offsetRect, scaledOffsetRect, includeDecorations);

Error in myRotobrush (line 71)
F = getframe(gcf);
```

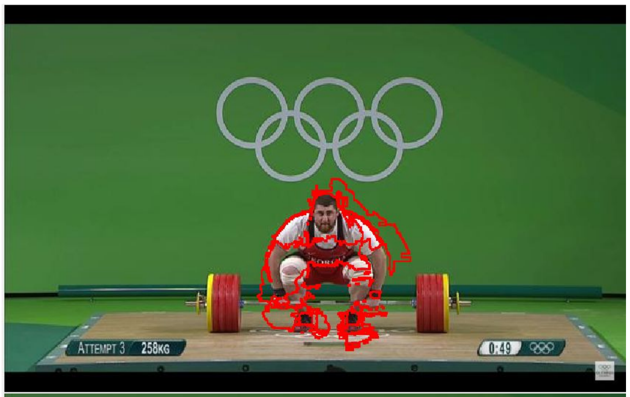
More output is shown below.

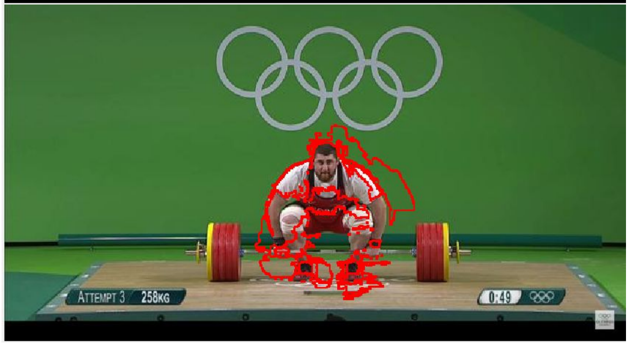
Powerlifter and Comments



The output of the third frame is shown to the left, done with the parameters: 5 GMMs, 20 windows and 50 pixels window width. We determined that these numbers to be the best way to generate local windows around the powerlifter.

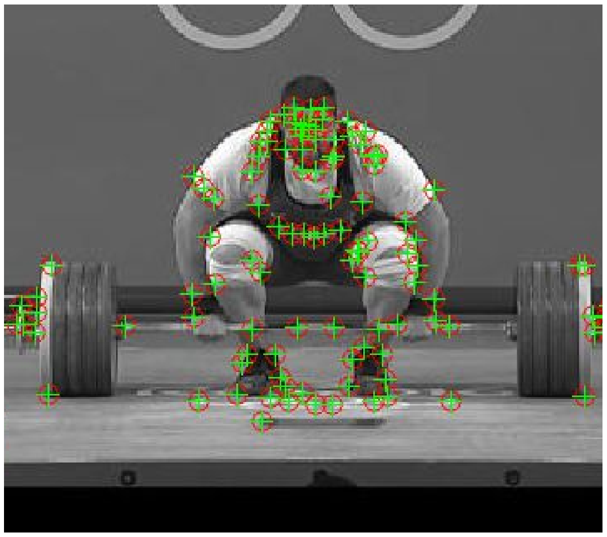
As similar to our other cases, the program is able to find the boundaries but as it progresses, there are issues with locating them on future frames. The more specific issue of this image is the different objects that we want to remove from the image (the barbell) cutting through the picture, as well as the holes in between the person's arms and legs that show the background that result in bounding errors.

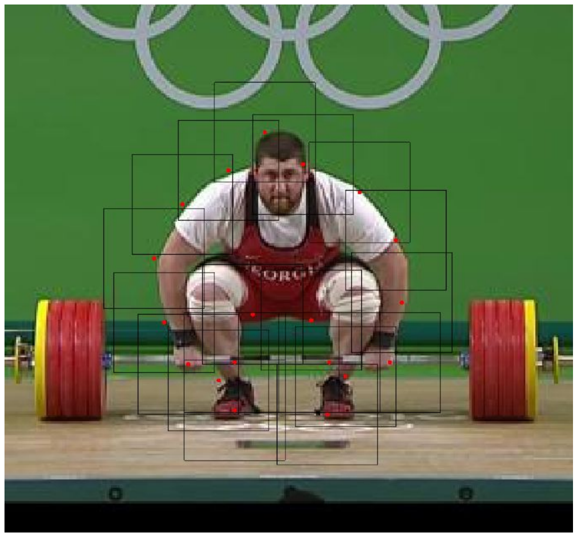




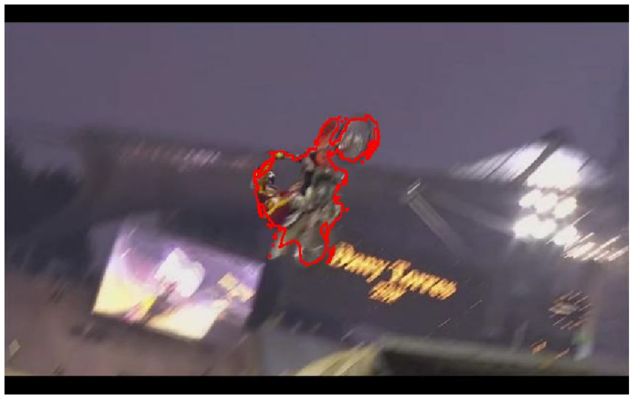
The Image Update and Local Windows

As we can see below, the output for the local windows and the image update, seem to be relatively good, with good overlap and a good distribution of the windows themselves. This also shows that our algorithm is able to accurately pinpoint the edges.





Biker and Comments



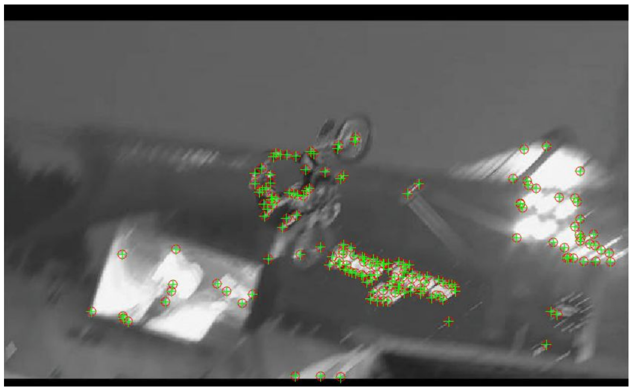
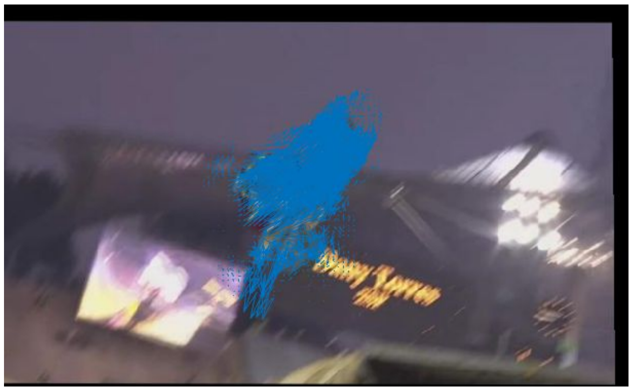
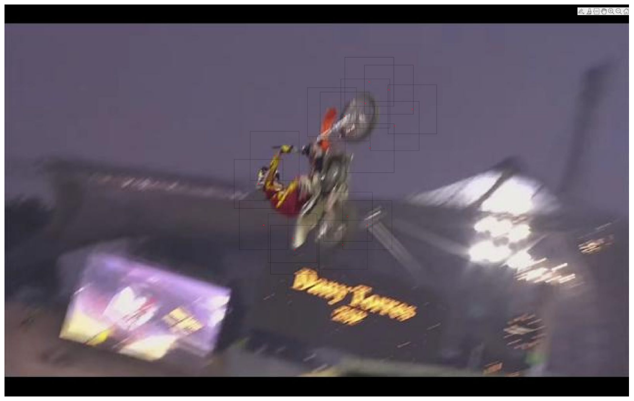
With the biker example, we choose the parameters of: 5 GMMs, 17 windows and 50 pixels window width. In this particular example, the main issue was that a lot of movement can be seen throughout the frame of the image as well as significant motion blur from the camera tracking the biker.

Blurry Picture

The images are very blurry and results in a very hard time finding boundaries:

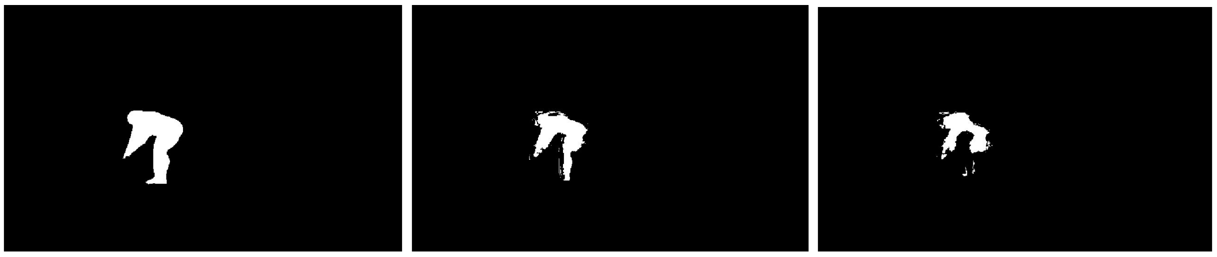
The Image Update and Local Windows

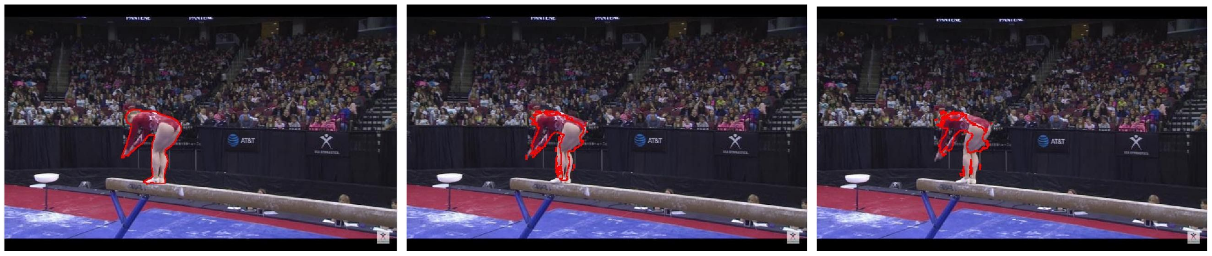
We are able to produce a decent flow analysis even though the blurriness of the image, as well as create a good boundary around the curvature and enough overlapping local windows.

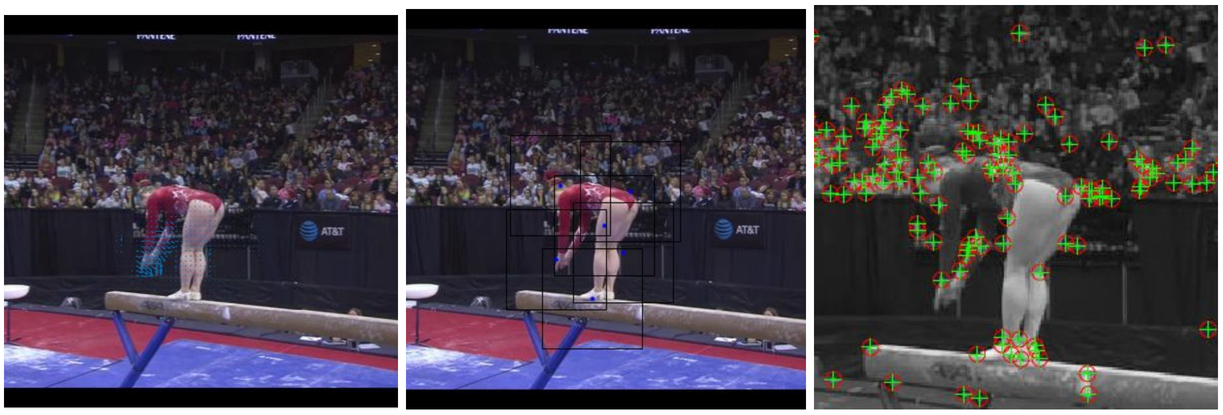
Gymnast and Comments

The parameters we used were: 5 GMMs, 7 windows and 100 pixels window width.



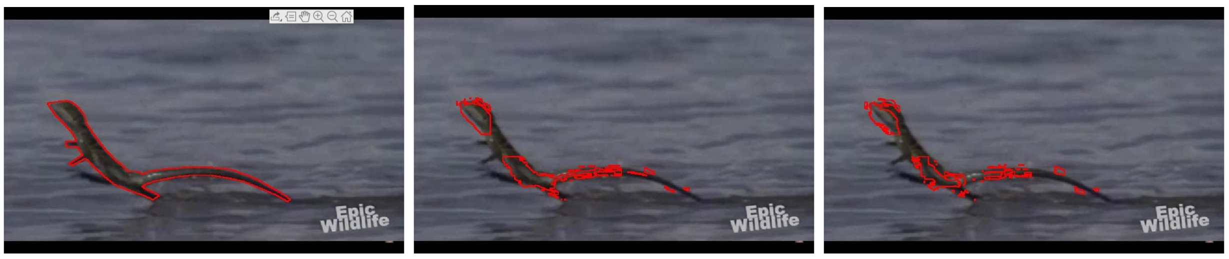


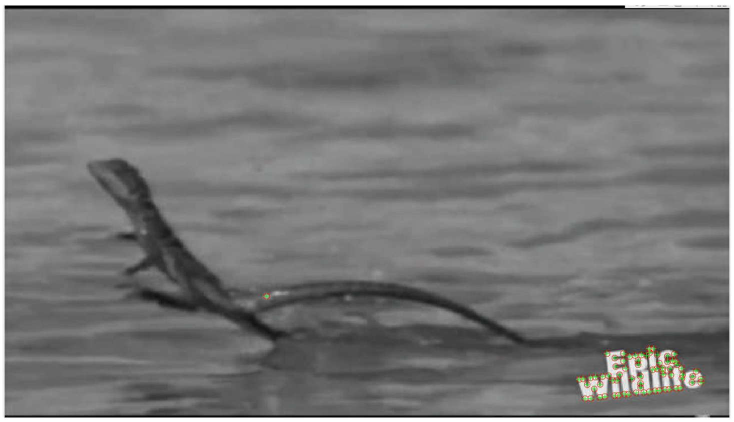
After the initial mask, in the process of updating and moving on to the next frame, our foreground mask quickly deteriorates. We go from having a decent mask in the 2nd frame, to losing half her body in the 3rd frame. This mostly occurred in the legs of the gymnast in the photos, and her back which was overlapping with the audience. The legs of the gymnast were similar in color to the beam she was standing on, and the multitude of colors in the audience also blended into the gymnasts' backs..



Lizard

The parameters we used were: 5 GMMs, 5 windows and 100 pixels window width. We ran into errors with calculating gmm in our initial Color Model when we used more windows.
Similarly to the other examples, our mask quickly deteriorated in accuracy. The problem was the very thin edges of the mask, it was difficult to even create the initial mask because it was hard to tell where the border of the lizard was, and what was just a shadow or the water.





Edges were centered around the logo rather than the subject of the photo because the boundaries for the lizard were so thin/blurred.