Peter Jochem
Computer Vision
MP7

Algorithm:

This algorithm is used to track the head of a person through a series of 500 images. We define where the girls head is at the start of the image set, and then search through the next image at nearby locations for regions that are most similar. The algorithm uses an ellipse with a fixed aspect ratio to define where the girl's head is in the image. To find the ellipse's location in the next image, we use an exhaustive local search over the prior center's neighboring pixels. At each neighbor's location, we center an ellipse and compute a metric of how similar this ellipse's pixels are to the prior image's ellipse. The ellipse with the best score is then chosen as the next image's ellipse. The best score will depend on which metric we are using. I implemented three different metrics. My first metric of similarity was the sum of square errors. So, at each location in the ellipse, I computed the square of the difference between each ellipse's intensity. The results are described below. My next metric was cross correlation. At each equivalent point in the ellipse, I computed the product of each locations intensity. The chosen ellipse is the one which maximizes this sum of cross correlations. The final, and most effective metric was the normalized cross correlation. This is cross correlation of the two ellipses but normalized. This gave me the best results. The normalized cross correlation results are really amazing. It's quite a simple algorithm but works quite well.

Results Analysis

The most effective metric was the normalized cross correlation. The sum of square errors and the cross correlation metrics gave me quite poor results.
I included the output of each algorithm as a video. output_normalized_cc.avi is the image set processed by the normalized cross correlation algorithm. output_cc.avi is the image set processed by the initial cross correlation algorithm. output_sum_square.avi is the image set processed by the sum of square errors algorithm.

Overall, I got pretty good results for the normalized cross correlation algorithm. One downside to this algorithm is that as mistakes are made, there is no real way for them to be corrected. Over time, the ellipse drifts from the real/ideal ellipse since the algorithm only checks for similarity across the prior ellipse. So, if the prior ellipse has drifted, the next ellipse will be impacted by this error. In summary, the errors compound without any mechanism to correct for it. You can see this drifting in the algorithm in the video output_normalized_cc.avi.
This compounding of error can also be seen in the other videos. In those, it is far more extreme. The ellipse drifts quite quickly and then has no way to correct itself.
If any part of the frame leaves the image, then I choose to simply not display the frame. So, for the sum of square errors and the cross correlation videos, the frame will be shown at the start then eventually drift to a point where part of it is out of the image so I choose not to display the bounding box anymore.

The image processing is quite slow. If you would like to run my code to generate the results given above, then do the following:
To process the image set with sum of square errors, run python3 cbirch.py sse. The resulting output is in a file called output_sum_square.avi. If any part of the frame leaves the image, then I choose to simply not display the frame.
To process the image set with cross correlation, run python3 cbirch.py cc. The output is in a file called output_cc.avi. If any part of the frame leaves the image, then I choose to simply not display the bounding box.

To process the image set with normalized cross correlation run python3 cbirch.py ncc. The output is in a file called output_normalized_cc.avi