CMSC 426, Computer Vision
Project 3: Rotobrush!

Prof. Yiannis Aloimonos,
Jack Rasiel and Kaan Elgin

March 15, 2018

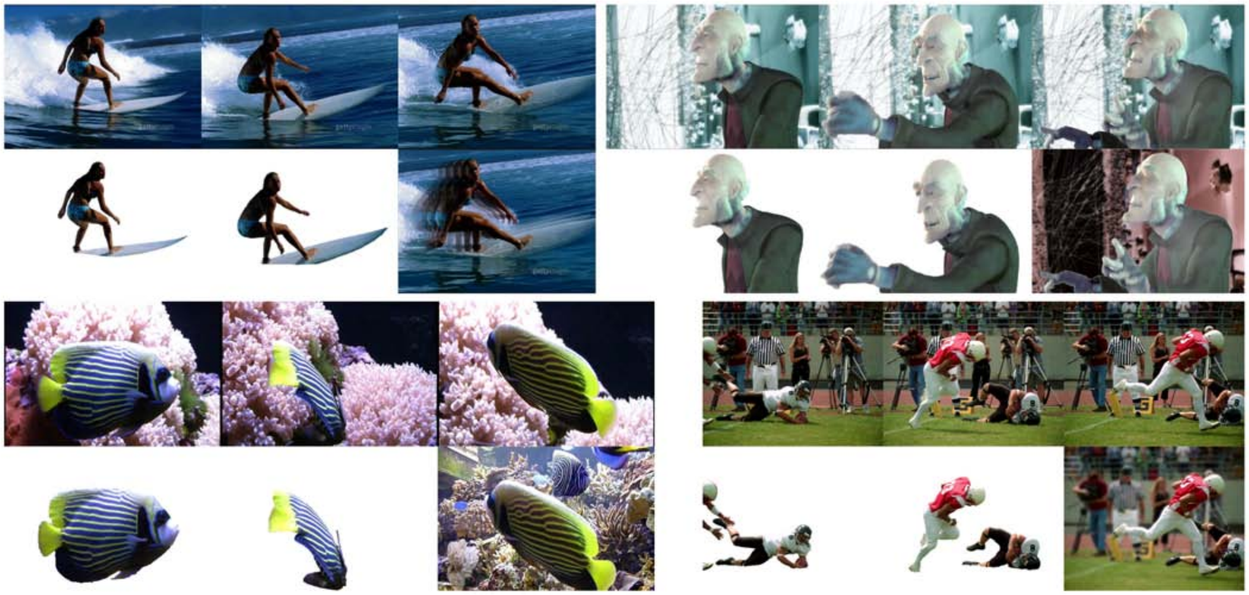


Figure 1: Examples of Video Snapcut. Source: [1]

1 Due Dates

As with per usual, this project is divided into two milestones:

- **MILESTONE 1 DUE DATE:** 11:59:59PM on Friday, **March 16** 2018
  - Initializing classifiers and updating window locations.
- **MILESTONE 2 DUE DATE:** 11:59:59PM on Thursday, **March 29** 2018
  - The rest of the system and results.

The milestones are intended to encourage timely completion of the projects: each milestone should be roughly the same amount of work, based on our estimates. **Reaching both milestones on time earns 10 points of extra credit.** If you don't want the extra credit, you don't need to submit anything for milestone 1. Turn in the full project for milestone 2. You cannot use late days to get credit for milestone 1, and the standard late-day policy applies for milestone 2.

Note that to receive extra credit for milestone 1, you must submit working code (and a report) which performs all functions described for that milestone. I.e., partially complete or broken code is ineligible for extra credit.

Each milestone will be submitted separately on ELMS; for the full submission guidelines **see section 10**.

2 Introduction

In this project, you'll be implementing a pipeline for **video object segmentation**. Given the boundary of an object in the initial frame of a video, your system will track the object's boundary through the subsequent frames. This is a challenging problem, and one with exciting applications across augmented reality, computer graphics, video editing, and more!

Our system will be based on the SnapCut algorithm by Bai et. al.[1], which powers the Rotobrush tool in Adobe AfterEffects! To get an intuitive understanding of the system we **highly recommend** watching the author's presentation at SIGGRAPH '09 (the first 14 minutes are relevant to this project).

3 System Overview

Again, we recommend watching the first fifteen minutes of the author's presentation at SIGGRAPH '09. It provides a good overview of the system's design, as well as insights into what specific challenges the authors sought to address with their design choices.

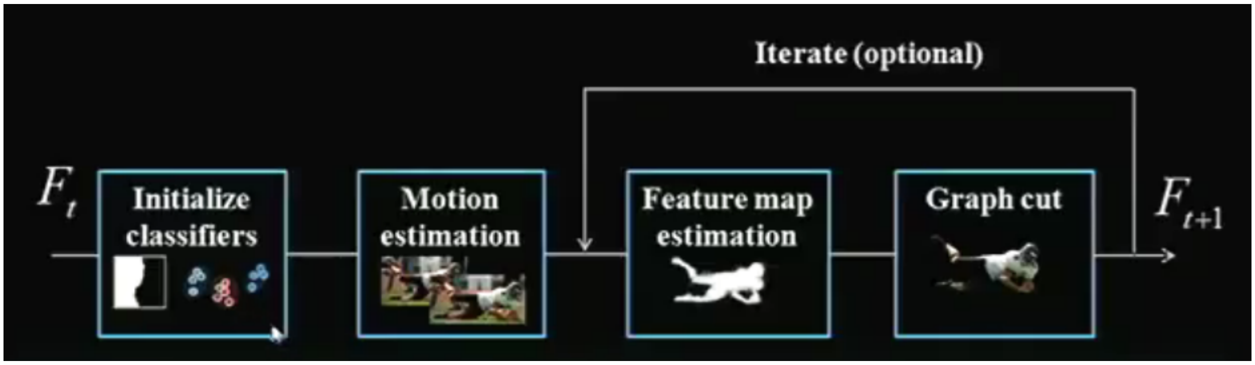


Figure 2: Source: `https://www.youtube.com/watch?v=ZIHrgXY1DNE`

Broadly, the system functions as follows.

Initially, the user provides a mask of the foreground object1. The system samples a series of overlapping image windows along the object's boundary. For each window, we create a "local classifier" which is trained to classify whether a pixel is part of the foreground or background. The classifier considers both the color and shape within its window to classify foreground and background.

For subsequent frames: we will use the local classifiers to estimate an updated foreground mask. First, we estimate the object's movement between frames to re-position the image windows. The goal is to have the windows stay in the same position on the object's boundary, or the most similar place possible. With the window positions updated, we then use the local classifiers to re-estimate the foreground mask for each window. These windows overlap, so we merge their foreground masks together to obtain a final, full-object mask. Optionally, we can re-run the local classifiers and merging steps to refine the mask for this frame.

Before moving on to the next frame, we re-train the local classifiers, so they remain accurate despite changes in background (and possibly foreground) appearance.

3.1 Design Principles

Bai et. al. provide some insight into their approach (emphasis added):

"we have found two important principles that can guide us to develop a better cutout system: (1) Multiple cues should be used for extracting the foreground, such as color, texture, shape, and motion. Among these, shape should play an important role for maintaining a temporally-coherent recognition; and (2) Multiple cues should be evaluated and integrated both locally and globally, rather than just globally, to maximize their discriminant powers." [1]

4 Setting up Local Classifiers

You must provide the system with an initial object mask for the first input frame. We recommend using Matlab's roipoly to create this mask.

4.1 Local Windows (5 pts)

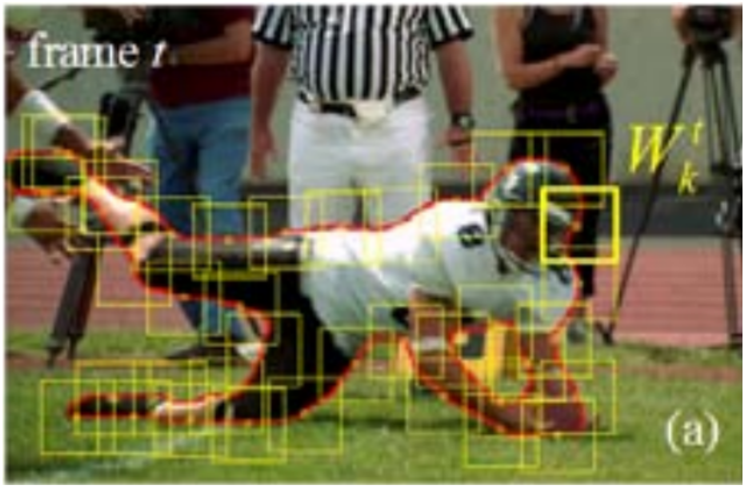


Figure 3: Example of overlapping local windows placed along an object's border.

Given the initial mask, sample local windows as shown in 3. The size and density of the windows is up to you. Note Bai et. al.: "[...] the distances between updated neighboring windows may slightly vary [between frames]. This is one of the main reasons to use overlapping windows [...], so that after propagation, the foreground boundary is still fully covered". (More hints abound in the paper!)

The window-level "local classifiers" are composed of a color model (a Gaussian Mixture Model to be precise), a shape model (the foreground mask and a shape confidence mask). Confidence metrics are calculated for the color and shape models: for the color model this is a single value, and for the shape model it is a mask. When the color and shape models are integrated into a single mask, the confidence values are used to assign more weight to the more confident model.

We'll detail these below:

4.2 Initializing the Color Model (10 pts)

The purpose of the color model is to classify pixels as foreground or background based on their color. The assumption is that foreground and background pixels will generally differ in color. The color model is based on Gaussian Mixture Models; you can use Matlab's `fitgmdist` and `gmdistribution` classes, but not the vision-specific functions/tools based on GMMs.

To create the color model, you must train two different GMMs: one to give the probability of a pixel being in the foreground, and another for the background. The foreground GMM is trained with pixels from the foreground, and the background GMM with pixels from the background. To help avoid sampling the wrong region, Bai et. al. recommend only choosing pixels more than a certain distance from the foreground/background boundary.

The output of the two models are combined as specified in equation 1 of the Snapcut paper:

$$p_c(x) = p_c(x|\mathcal{F}) / \left(p_c(x|\mathcal{F}) + p_c(x|\mathcal{B})\right),$$

where $p_c(x|F)$ and $p_x(x|B)$ are the probabilities given by the two GMMs.

See section 2.1 of [1] for more details on implementing the color model.

4.3 Color Model Confidence (5 pts)

Bai et. al. define a confidence metric for the color model. This is a single value computed for the entire window, intended to ”describe how separable the local foreground is against the local background” in the window.

To calculate the confidence value, see **equation 2** of the Snapcut paper, in section 2.1. Note that all variable definitions etc. needed to understand the equation are given in the sentence preceding the equation and the paragraph following it. Note also that the use of integral notation in this equation means ”sum over all pixels in the window”.

4.4 Shape Model (and Shape Confidence) (15 pts)

The local shape model is a combination of two masks: the foreground mask, and a shape confidence mask. The foreground mask is given by the user at start (using ROIpoly or a similar tool), as mentioned previously.

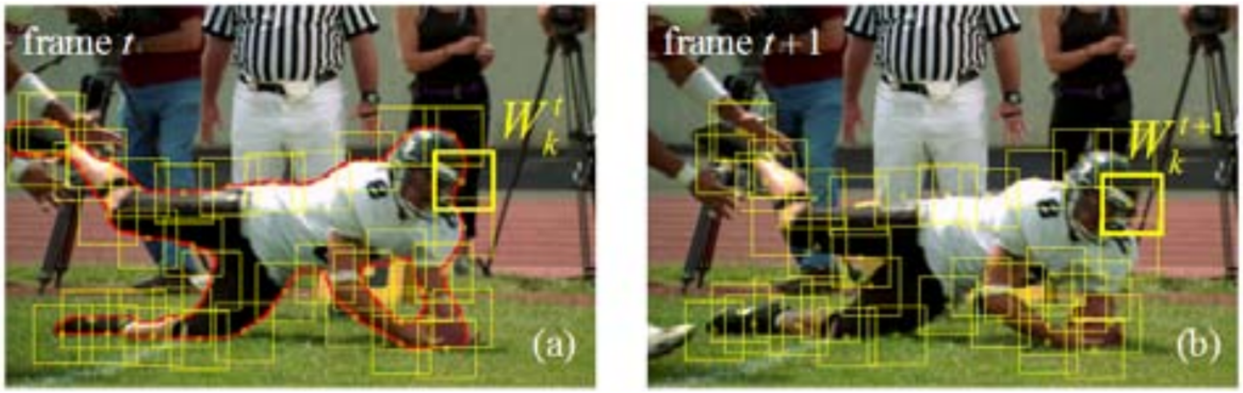
The shape confidence mask is defined by **equations 3 and 4** of the Snapcut paper (in sections 2.1 and 2.4, respectively). To give a general intuition: the shape confidence decreases as color confidence increases, and also increases with distance from the object's boundary. Fig. 5 in the paper provides an illustration of these properties. To save you some time searching through the paper: the paragraphs immediately before and after equations 3 and 4 give all the variable definitions and other context needed to understand them.

5 Updating Window Locations

In the previous section, we set up local classifiers for identifying foreground and background pixels within their windows. As we move on to the next video frame, the object and background may move and/or change. Our local classifier windows must move to stay centered on approximately the same place on the object's boundary.

We accomplish this by tracking two kinds of motion: the rigid motion of the whole object, and then the smaller local deformations of the object's boundary. For example, in 5 the football player is overall falling downwards (motion affecting the entire object), as well as bending his arms, turning his head, bending his legs, etc. (changing specific regions of the object's boundary).

These methods correspond to section 2.2 in [1].



5.1 Estimate Whole-Object Motion (5 pts)

To estimate the overall motion of the object, find matching feature points on the object in the two frames, and use them to find an affine transform between the images. Use this to align the object in frame 1 to frame 2. ”This initial shape alignment usually captures and compensates for large rigid motions of the foreground object.” You are allowed to use Matlab’s `estimateGeometricTransform` as well as its feature detection and matching functions.

5.2 Estimate Local Boundary Deformation (10 pts)

After applying the affine transform from the previous step, we want to track the boundary movement/deformation for the local windows. Optical flow will give us an estimate of each pixel’s motion between frames. However, as Bai et. al. note, optical flow ”is unreliable, especially near boundaries where occlusions occur”. Luckily, we know exactly where the object’s boundary is! Thus, to get a more accurate estimate, find the average of the flow vectors inside the object’s bounds. Use this average vector to estimate how to re-center the local window in the new frame.

While this method for window re-positioning is not perfect, errors are accommodated by the significant overlap between neighboring windows: they should still overlap enough that the object’s entire boundary is covered.

Note that, for this project, you are allowed to use Matlab’s optical flow functionality, such as the `opticalFlowHS` class.

6 Update Local Classifiers

Now that the local windows have been properly re-centered, we can update the local classifiers for the new frame.

6.1 Updating the Shape Model

The shape model is composed of the foreground mask and the shape confidence map. These are both carried over from the previous frame.

6.2 Updating the Color Model (and Color Confidence value) (15 pts)

The distribution of colors in the foreground and background may change from one frame to the next, as different parts of the scene move independently. We want to update the color model to reflect these changes. Simply replacing the existing color model with a new pair of GMMs every frame could pose problems. For one, if there's a sudden change in color in one frame, which quickly disappears in the next, our color model will be completely de-railed. Moreover, the new GMMs may have degraded performance because of improper labling of the pixels used to train them. This is because we label "foreground" and "background" pixels based on the foreground mask, and that may be less accurate after updating window locations in the previous step.

So what can we do? Bai. et. al. propose to compare two color models: the existing one from the previous frame and a combination of the previous and new frame's GMMs. They first observe that the colors in the foreground region don't change much between frames, while the background region can change significantly. Therefore we'd expect the number of pixels classifier by the model as foreground to be relatively consistent between frames. If the number of foreground pixels increases under the new color model, then we should stick with the old one.

If we choose the new color model, we must also re-compute the color confidence value, as was done in section 4.3.

7 Combining Shape and Color Models (5 pts)

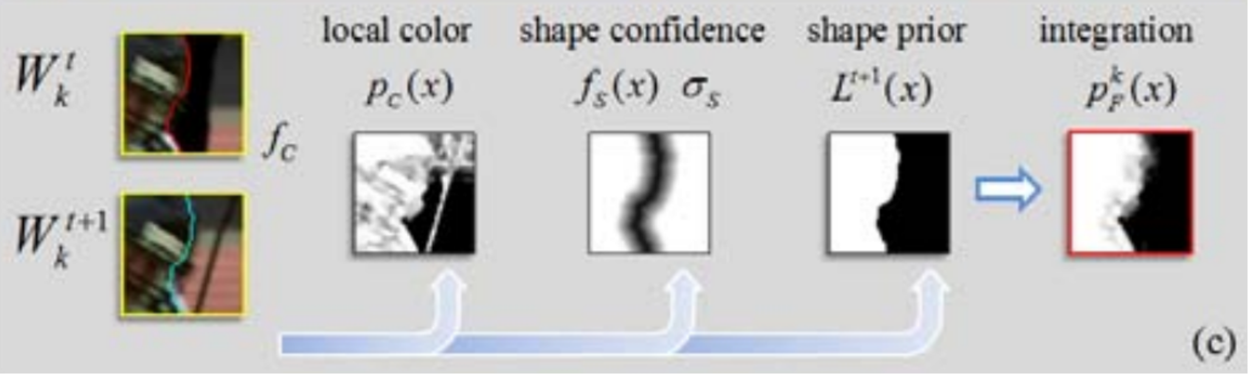


Figure 4: Merging local classifiers.

For each window, we now merge the foreground maps produced by the shape and color models, weighting them based on the shape confidence map. This is done as per equation 5 in Bai et. al.:

$$p_{\mathcal{F}}^{k}(x) = f_s(x)L^{t+1}(x) + (1 - f_s(x))p_c(x).$$

Where $p_f^k(x)$ is the foreground map for the kth window, $f_s(x)$ is the shape confidence map, $L^{t+1}(x)$ is the shape model's foreground mask, and $p_c(x)$ is the color model's foreground mask.

8 Merging Local Windows and Extracting the Final Foreground Mask (30 pts)

After the previous step, we have a foreground probability mask for each local window. We now merge the overlapping local windows into a global foreground mask. This is done via a weighted sum (equation 6 in [1]):

$$p_{\mathcal{F}}(x) = \frac{\sum_k p_{\mathcal{F}}^k(x)(|x - c_k| + \epsilon)^{-1}}{\sum_k (|x - c_k| + \epsilon)^{-1}},$$

"where k is the index of local windows (the sum ranges over all the k-s such that the updated window W_k^{t+1} covers the pixel), ϵ is a small constant (0.1 in the system), and c_k is the center of the window ($|x - c_k|$ is the distance from the pixel x to the center)." [1]

This gives a real-valued probability map for the foreground mask. We want a binary mask. The simplest solution would be to threshold the values of the probability map. This may produce a somewhat rough result. Bai et. al. use Graph Cut segmentation to obtain a better final result: you are encouraged (but not required) to use Matlab's `lazysnapping` tool to implement this.

9 Extra Credit

- Implement the iterative refinement process described at the end of section 2.5 in [1]. **(5Pts)**
- Augment the local classifiers with additional image features beyond shape and color (Example: image texture – textons!). Be sure to describe your approach and results in your report! **(20Pts)**

References

[1] X. Bai, J. Wang, D. Simons, and G. Sapiro, "Video snapcut," *ACM SIGGRAPH 2009 papers on - SIGGRAPH 09*, 2009.

10 Submission Guidelines

If your submission does not comply with the following guidelines, you'll be given ZERO credit:

10.1 File tree and naming

Your submission on Canvas must be a zip file, following the naming convention **"YourDirectoryID_proj3_milestone#.zip"**. For example, jrasiel_proj3_milestone1.zip (for milestone 1). The file **must have the following directory structure**, based on the starter files:

```
YourDirectoryID_proj3_milestone#.zip
├── Code/
│   ├── MyRotobrush.m
│   └── (Any dependencies of MyRotobrush.m)
├── Input/ - MyRotobrush should look for input frames here.  Leave it empty.
├── Output/ - MyRotobrush should return processed frames here.
├── results1.mp4
├── results2.mp4
├── results3.mp4
├── results4.mp4
├── results5.mp4
└── report.pdf - Your report.
```

10.2 Running your code

When run, **MyRotobrush.m** must load a set of frames from the folder `Frames/`. Assume that the frames will be jpeg images of the form 1.jpg, 2.jpg, etc. Your program must then prompt the user to specify a region of interest in the first frame (for example using the ROIPoly tool), and then track the specified object through the rest of the frames. For each frame, draw the tracked boundary in red and save the result with the same filename in `Output/`.

10.3 Output videos

You have been provided the frames from five video clips. Run your code on each set of frames and create videos, named as indicated above. For each video track the following:

- `Frames1`: Track the turtle. *Source: Nitin Sanket's friend.*
- `Frames2`: Track the motorcycle and rider. *Source:* https://www.youtube.com/watch?v=jhxCJLm4C-0
- `Frames3`: Track the gymnast. *Source:* https://www.youtube.com/watch?v=gMNCwgXtQIw
- `Frames4`: Track the powerlifter, **without the weights**. *Source:* https://www.youtube.com/watch?v=EiqEFUFM-KI

- `Frames5`: Track the lizard (including tail). *Source:* *https://www.youtube.com/watch?v=Mp1R4Lxoj5c*

10.4 Report

You will be graded primarily based on your report and the quality of your result videos. We want you to demonstrate an understanding of the concepts involved in the project, and to show the output produced by your code.

Include visualizations of the output of your system. For at least one video clip, you must:

- show the locations of the local windows on your object (like fig. TODO),
- track the movement of one of the local windows over at least four frames,
- show how the color, shape, and shape confidence maps of one window change over at least four consecutive frames,
- show how the color and shape models are merged for those frames,
- draw the final object boundary onto those frames.

As usual, also describe what you did for each step, including any problems you encountered and/or interesting solutions you implemented. Discuss what sort of input images cause your system to behave poorly. Show examples. Your report must be full English sentences, **not** commented code.

A reminder: Every student should present AT LEAST once in the class (you can volunteer to present for more than one project) and can choose to do for any of the projects.

11 Collaboration Policy

You are restricted to discuss the ideas with at most two other people. But the code you turn-in should be your own, and should be the result of you exercising your own understanding of it. If you reference anyone else's code in writing your project, you must properly cite it in your code (in comments) and your writeup. For the full honor code refer to the CMSC426 Spring 2018 website.

DON'T FORGET TO HAVE FUN AND PLAY AROUND WITH IMAGES!