Active Contours Graduate Project

ECE6310: ACTIVE CONTOURS

December 10, 2020

Bradley Selee
Clemson University
Department of Electrical and Computer Engineering
bselee@clemson.edu

1 Purpose

This project was an expansion upon the active contour algorithm we learned and implemented in lab 5. The goal of this project was to segment complex images with active contours and wrap the algorithm inside a graphical user interface (GUI). The project should allow the user to draw contour lines around or inside objects within an image. These contours will "actively" shrink or grow around the desired object, representing the rubber-band and balloon model. The user should be able to drag contour points away from the line, and have the contour adjust to this removed point. The GUI should be able to load color PNM images, draw contour lines, and clear images. Parameters such as number of iterations, energy weights and contour sizes do not need to be controlled by the GUI. Instead, these parameters are constants at the beginning of the project code. Each time these are changed, the user will restart the program.

2 Methods and Materials

For this project, my implementation was created solely using Python. The only libraries used were Tkinter, for the GUI, and Python's Image Library (PIL), to get pixel data. Inside the project folder there are two main components: the back-end algorithm, and the front end GUI to control the program. The images to be segmented are also inside this folder, but are not required to be in the folder.

2.1 Code Design and Implementation

The main idea behind the active contours algorithm is to minimize energy around a contour line. Typically, there exists external energies and internal energies. The external energy is completely dependent upon the image. This energy is usually some variation of edge detection and/or Gaussian smoothing. As for the internal energy, this is dependent upon constraints around the shape of the contour line, such as the curvature of the line. Once these energies are defined, a window is created around each contour point, and all energies are summed at each of inside the window. Finally, the minimum value is found inside this window and the contour point moves to this minimum pixel location. This process is repeated for as many iterations as the user defines.

Based on this algorithm, my program was designed to store the contour lines and internal/external energies, while still being flexible for the user to create and modify. To begin, the user will load in a PNM image. During this loading, about 4 convolutions will be performed to create the external energies. Only a few of these are actually used, and the rest showcase attempts that I tried to improve the algorithm. Once the images are loaded, the user can draw draw several contour lines around objects in the image. Right click and drag to initialize a rubber-band contour, and left click to create a circle contour with using the balloon model. Once the contours are created, the user can shift+left click

to remove a point from the contour and drag it off into space. Once the desired contours are created, the active contour algorithm can be ran from the menu bar in the 'Contours' tab. When run, the internal energies are created for each contour, with a window around each point. All energies are summed, the minimum point is found, and the initialized contour attempts to fit the desired shape.

2.2 Energies

There were many external energies I tried to improve this algorithm. Originally, I tried using only the inverted, magnitude of the sobel edge detection. This energy alone seemed to work well. I tried to improve it by adding another energy which would compare color intensities, and the point would move closer to pixels with similar colors. This was accomplished by extracting the R, G, and B channels of the image, then taking the Euler distance between the current point and the point in each window. This worked well for some objects, but made other objects worse. Ultimately, I did not use either of these energies for the final implementation.

The final external energy I used was an attempt at making a gradient vector flow/field (GVF). I am not sure I implemented it well but it did improve my results when the algorithm was ran. The external energy follows the equation

$$E_{ext} = \frac{|\nabla I(x) * G_\sigma(x)|}{max_x(|\nabla I(x) * G_\sigma(x)|)}$$

The internal energies worked relatively well and did not need adjusting. There were two used and can be interpreted as contour elasticity and curvature

$$E_{curvature} = (X_{i+1} - 2X_i + X_{i-1})^2 + (Y_{i+1} - 2Y_i + Y_{i-1})^2$$
$$E_{elasticity} = (average\_distance - \sqrt{(X_{i+1} - X_i)^2 + (Y_{i+1} - Y_i)^2})$$

3 Results

There were five given PNM images for this project. Below shows the best results obtained from each image using the rubber-band and balloon model.

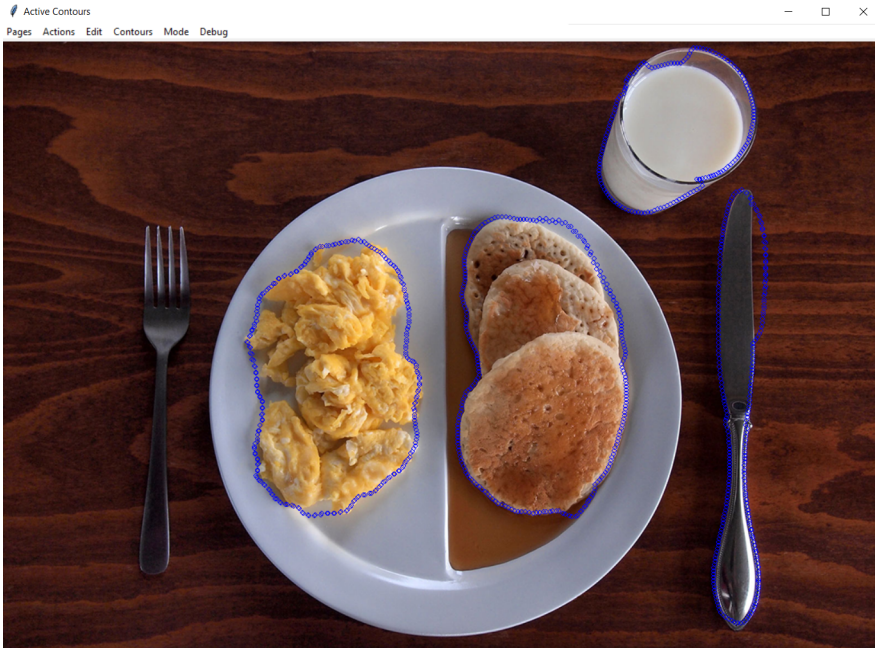

Figure 3.1: Eggs, pancakes, milk final rubber-band model

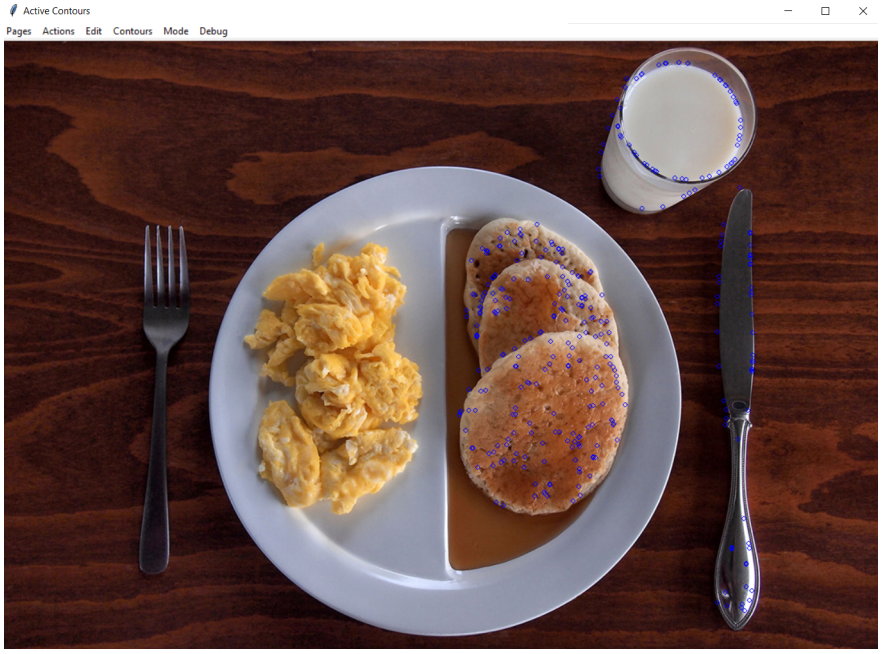

Figure 3.2: Eggs, pancakes, milk final balloon model

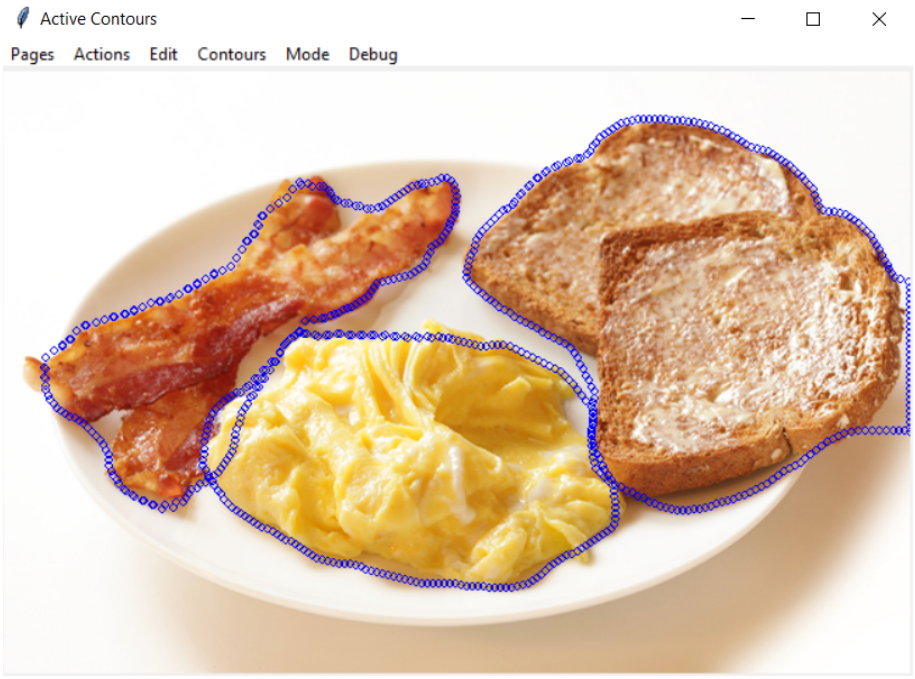

Figure 3.3: Eggs, toast, bacon final rubber-band model

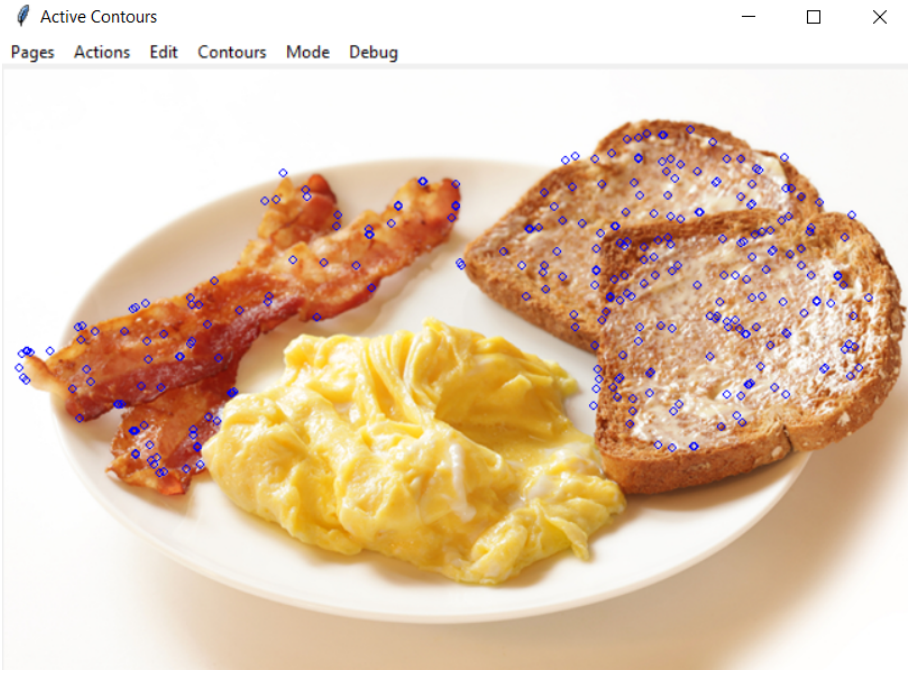

Figure 3.4: Eggs, toast, bacon final balloon model

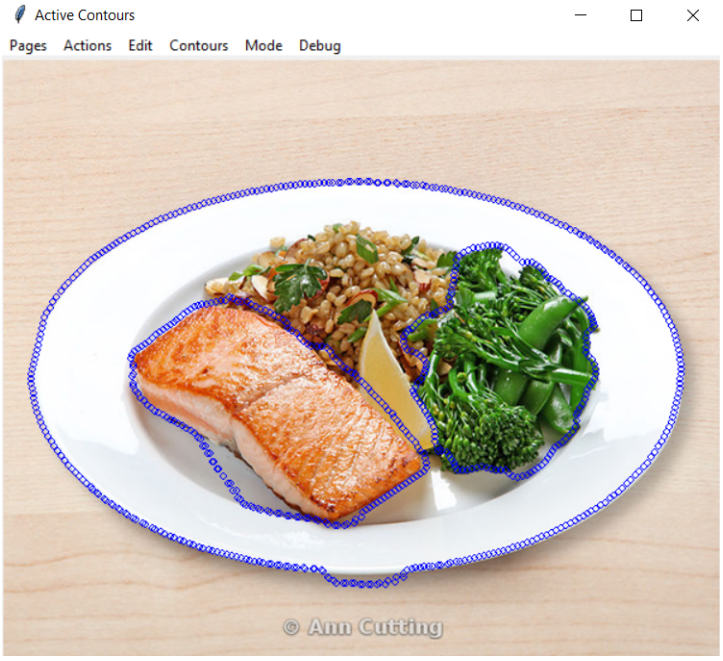

Figure 3.5: Salmon, rice, broccoli final rubber-band model

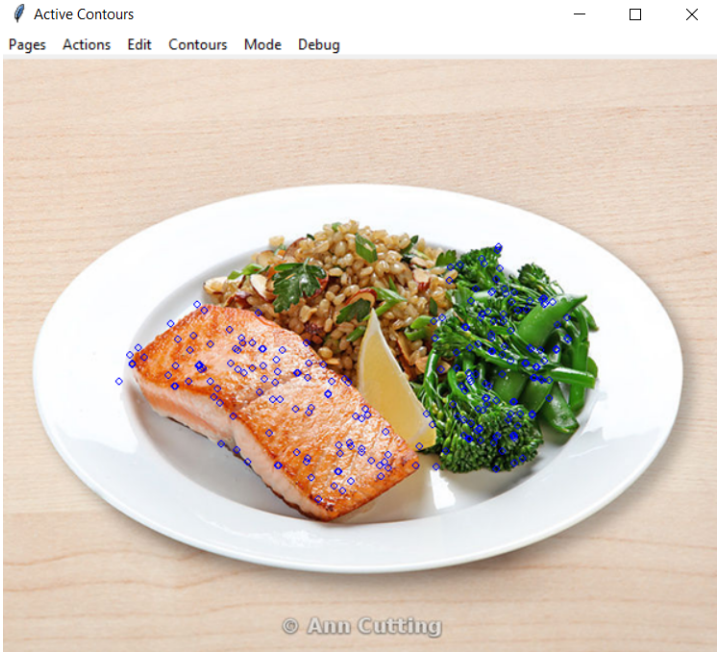

Figure 3.6: Salmon, rice, broccoli final balloon model

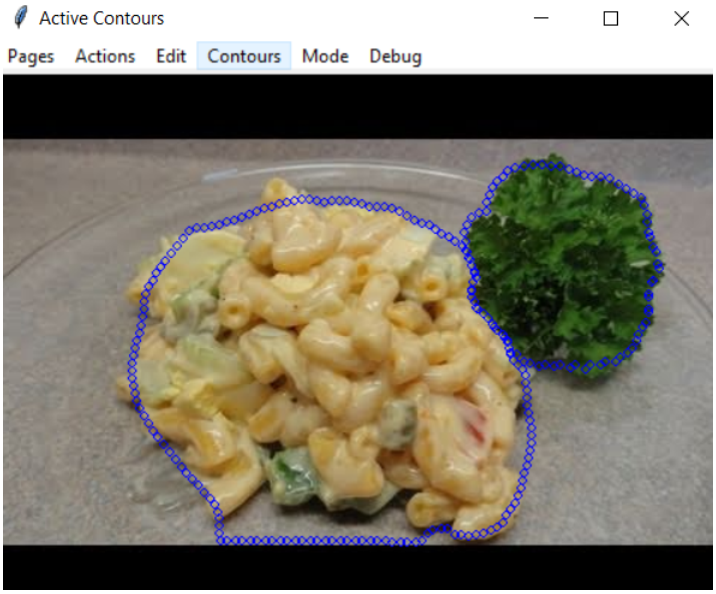

Figure 3.7: Macaroni, kale final rubber-band model

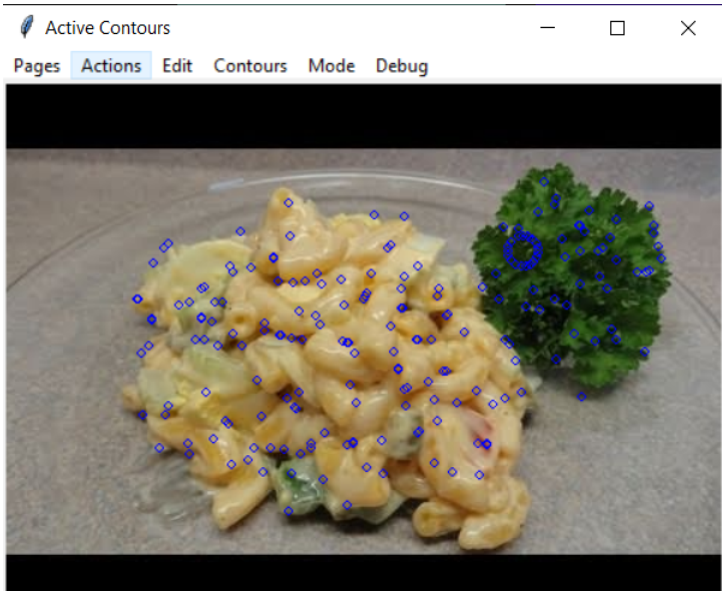

Figure 3.8: Macaroni, kale final balloon model

Figure 3.9: Hush puppies and biscuits final rubber-band model

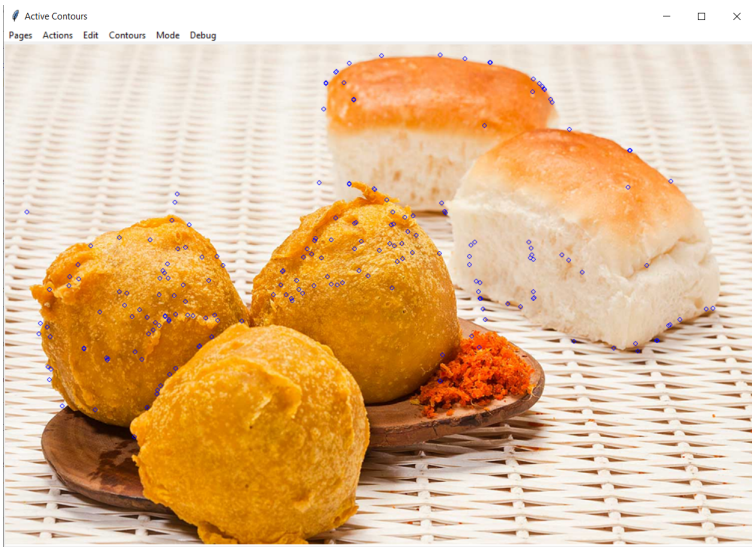

Figure 3.9b: Hush puppies and biscuits final balloon model

4 Conclusion

Overall, I think this project was successful. My implementation shows the use of the active contour algorithm for both the rubber-band model and the balloon model. The rubber-band model was more successful and worked on complex images, compared to the balloon model not working as well on complicated images. The GUI implements the

required features and gives the user a clean interface to work with. If I were to redo this project, I would have allocated more time for the balloon model implementation. I originally thought that this would be quick, but the correct energies were more difficult to find than I originally thought. I also would have cleaned up the code more to remove unnecessary functions and provide better comments. To improve the overall project, I would look more into gradient vector fields. I believe my implementation of the GVF was not as efficient as it should have been. Finally, I would look into compiling certain function in C code, then calling these through Python to improve speed. Due to the slowness of Python, some convolutions took a while to complete.

References

No further references were needed for this project.