EW309 Computer Vision

Image Processing

Goal

Digital images must be processed to generate actionable information for an algorithm. For this project, we need to identify and track a target with known color and shape in a cluttered environment.

Algorithm Overview

A simplified functional block diagram for the recommended target identification and tracking algorithm is shown in Fig. 1.

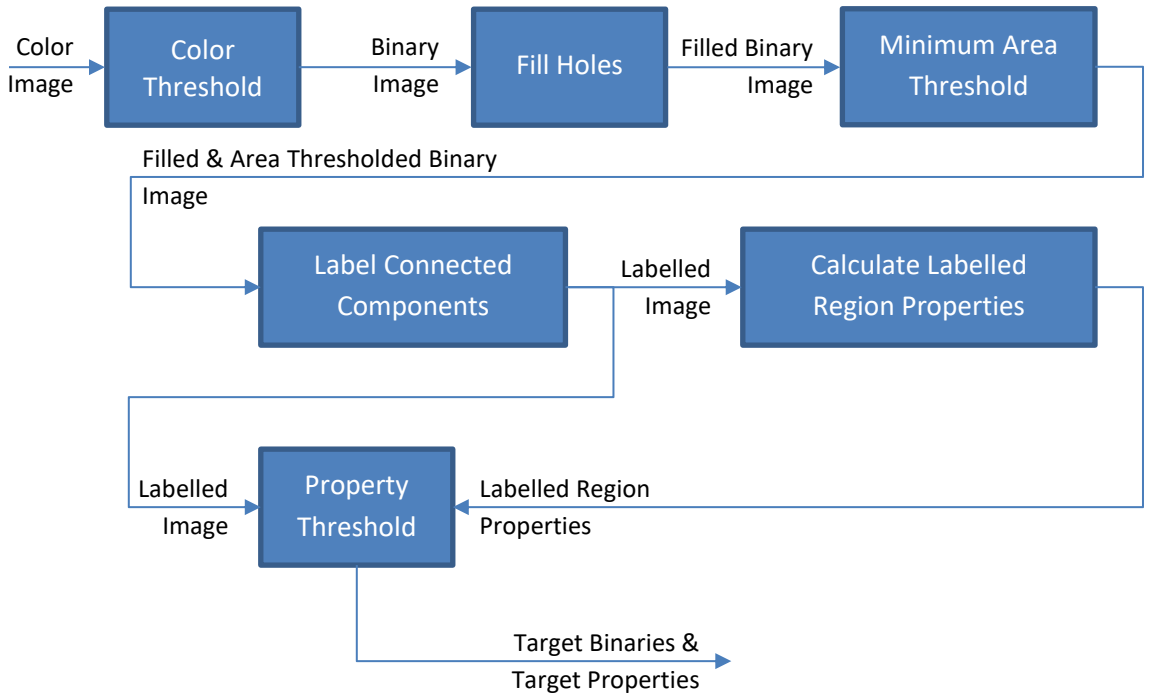

Figure 1: Simplified functional block diagram for the target identification and tracking algorithm.

Color Thresholding

The simplest method to isolate objects in an image is using color thresholding techniques. These techniques define a bounding region in color space and keep pixels whose colors fall in that region. Generally, given an $M \times N \times 3$ color image, the outputs of a color thresholding function will be:

1. An $M \times N$ binary image containing a 1 (i.e. "true" value) in pixel locations where the color falls within the bounding region, and a 0 (i.e. "false" value) everywhere else.
2. An $M \times N \times 3$ masked image containing the original pixel color values in pixel locations where the color falls within the bounding region, and a designated (typically black or $[0,0,0]$) pixel color values everywhere else.

The binary image output will be largely used in this project, while the masked image may be useful for debugging.

Creating a Color Thresholding Function

In the interest of time, we will use the MATLAB Color Thresholder App to define our bounding region and create a thresholding function. Given an image contained in the variable "im," the Color Thresholder App can be called using:

```
colorThresholder(im);
```

This will open a window containing your image and point clouds of four color space options that can be used for thresholding the image (Fig. 2).

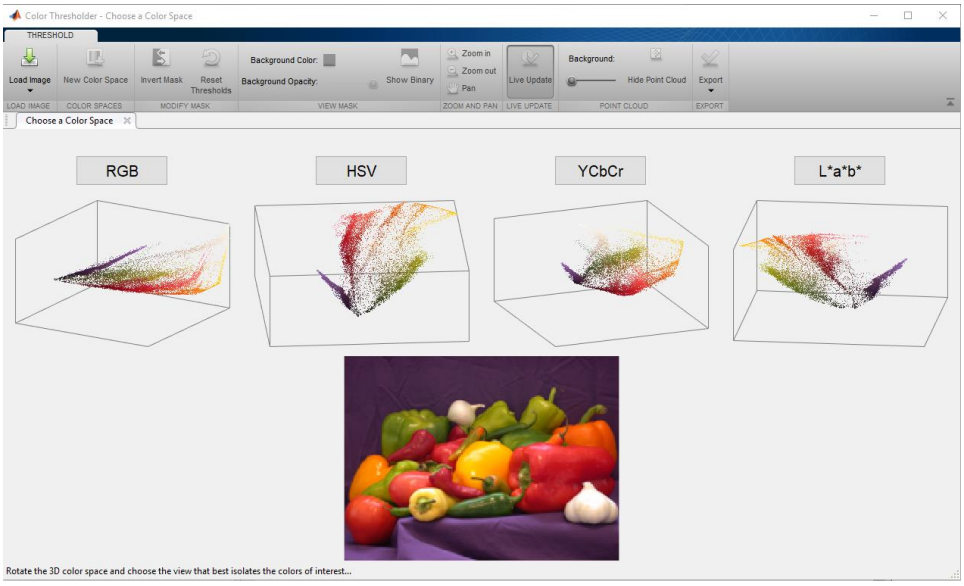

Figure 2: Color space selection prompt associated with the MATLAB Color Thresholder App.

Per the prompt in the window, each point cloud in 3D color space can be rotated to find the option that best isolates whatever color(s) you want to threshold. Once a viable option is identified, select the color space using the button above the 3D plot. Doing so will update the window based on your selection (Fig. 3).

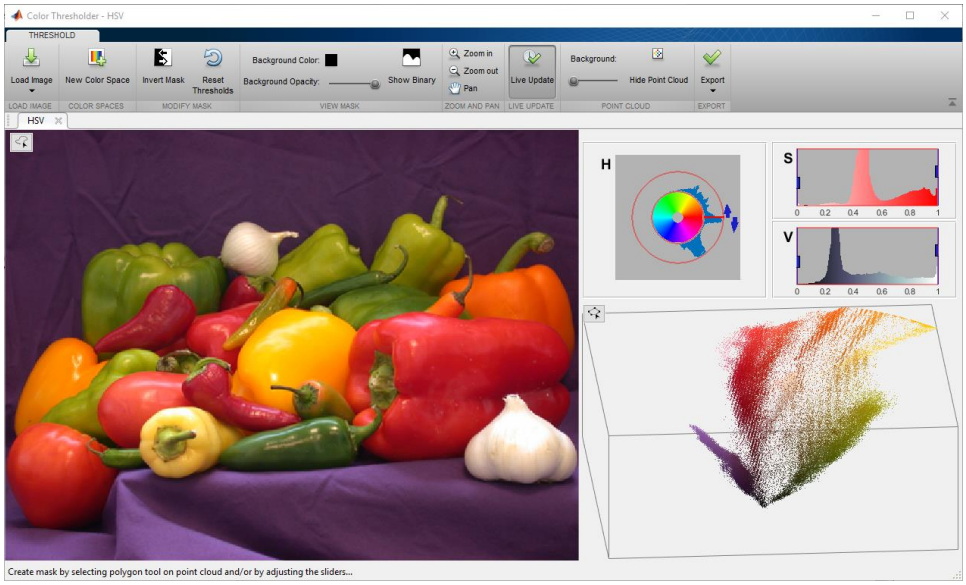

Figure 3: Threshold region definition prompt associated with the MATLAB Color Thresholder App.

The threshold region definition window provides three options to specify a threshold region:

1. Sketching a region in the image to try and segment (upper left),
2. Tuning threshold values using sliders for each color channel (upper right), and
3. Defining a polygon region in rotated 3D color space (lower right).

While each individual method or combination of methods can be effective, we will focus this effort on method (3). As an example, to threshold the green peppers in the image shown in Fig. 3, we can rotate the color space point cloud to better isolate the green pixels, then draw one or more polygon(s) around those colors in the point cloud (Fig. 4).

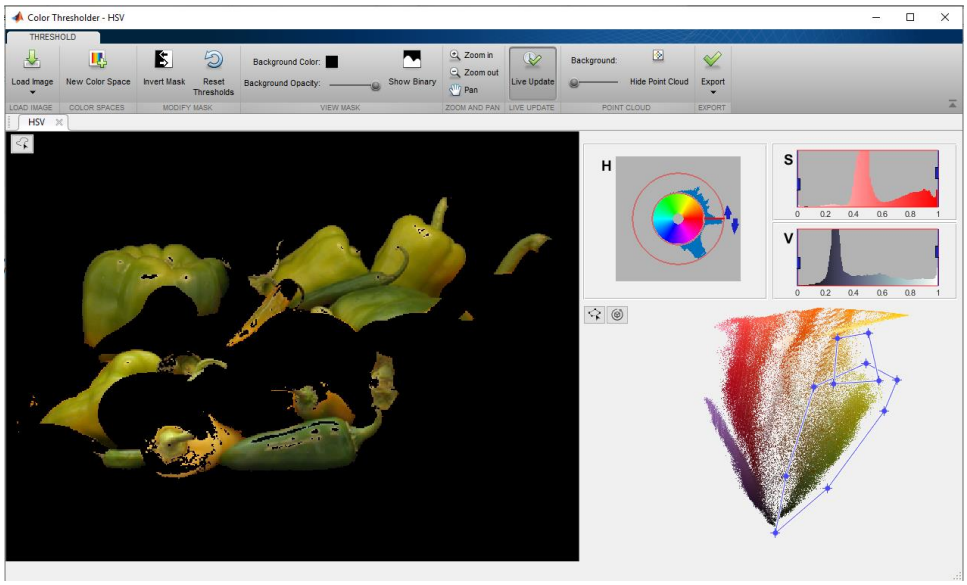

Figure 4: Example of segmenting green peppers by creating two polygon regions in rotated color space using the MATLAB Color Thresholder App.

As you create closed polygon regions, you will see a masked image form where your original image used to be. The points of each polygon can be adjusted, and polygons can be added/removed until a satisfactory result is achieved.

Once you are satisfied with your result, you can create your thresholding function by selecting the "Export" dropdown and choosing "Export Function." This will automically generate a MATLAB function with the default name "createMask." You can choose to adjust this name as needed in the function declaration line of the code, and save the function to your current directory. Be sure the filename matches the function name used in the declaration line of the code. Once saved, you can create a binary image from a color image using:

```
bin = createMask(im);
```

This assumes that your color image is contained in "im" and you kept the default thresholding function name "createMask." In this example, output binary image will be placed in the variable "bin."

Filling Holes

Holes in binary images generated using color thresholding are common. The causes are usually associated with variable lighting and/or reflective surfaces. Holes are defined as black (i.e. 0 or false) regions that are fully surrounded by white (i.e. 1 or true) pixels. Assuming a binary image "bin," holes can be removed using the "imfill" function in MATLAB with the following syntax:

```
binFill = imfill(bin,'holes');
```

This creates a new binary contained in the variable "binFill" with all holes removed.

Minimum Area Thresholding

Much like holes, small regions of noise in binary images generated using color thresholding are common. These can be removed by setting a minimum connected component area threshold. Connected components are defined as continuous regions of white pixels, and area is defined by the sum of all connected pixels in a given connected region. Assuming a binary image "bin" and a minimum area value of "minArea," minimum area thresholding can be performed using the "bwareaopen" function in MATLAB with the following syntax:

```
binMinArea = bwareaopen(bin,minArea);
```

This creates a new binary "binMinArea" where regions with areas below the specified "minArea" value are removed.

Labelling Connected Components

Continuous regions of white pixels (i.e. 1 or true) in a binary image can be separated into individual connected components (i.e. continuous, connected regions of pixels). This is an important tool as individual targets, when properly thresholded, should yield a single connected region of white pixels.

Assuming a binary image "bin," the connected components can be labelled and the number of connected components can be counted using the "bwlabel" function in MATLAB with the following syntax:

```
[lbl, n] = bwlabel(bin);
```

This creates a labelled image "lbl" and provides the total number of individual connected components "n." Labelled images contain integer values ranging from 0 to the value contained in "n." The result is an image whose first connected component is filled with 1s, second connected component is filled with 2s, etc. until the nth connected component is reached. As with binary images, 0s represent unthresholded regions of the image.

Calculating Labelled Region Properties

Region properties are measurable attributes associated with a binary region or connected component in a labelled image. While there are numerous properties that can be calculated for a binary or labelled region, we will consider the following subset for this initial investigation:

- Area – the sum or total number of pixels contained in a given binary or labelled region.
- Centroid – the "center of mass" associated with a given binary or labelled region. For a target that is symmetric and a single, connected component; we expect the centroid to lie in the middle of the segmented target.
- Eccentricity – is a measure that quantifies the elongation of a binary or labelled region. Eccentricity values range from 0 (for a perfectly circular region) and 1 (for a region that is a line segment).

Given a labelled image “lbl, the region properties of area, centroid, and eccentricity can be calculated for a binary or labelled imaged using the “regionprops” command in MATLAB with the following syntax:

```
stats = regionprops(lbl,'area','centroid','eccentricity');
```

This creates an n-element structured array “stats” (assuming n labelled connected components contained in “lbl”) that contains the fields “Area,” “Centroid,” and “Eccentricity.” As an example, the centroid for the 2nd connected component can be recovered using the following syntax:

```
cent = stats(2).Centroid;
```

Region Property Thresholding

Using the properties calculated for the labelled connected component regions of our image, we can eliminate any regions whose properties do not match the properties of known targets. To do this, we can set minimum and maximum threshold values for area and eccentricity. Given:

- maximum area values for a viable target defined “maxArea” and
- minimum and maximum eccentricity values for a viable target defined “minEcc” and “maxEcc”

An array of index values “idx” of labelled regions whose area and eccentricity value falls within these bounds can be found using the “find” command in MATLAB with the following syntax:

```
idx = find(...
      [stats.Area] <= maxArea & ...
      [stats.Eccentricity] >= minEcc & ...
      [stats.Eccentricity] <= maxEcc);
```

Note that the “...” allows us to separate a single line of code into multiple lines for commenting and better readability. Additionally, minimum area threshold values were imposed previously using “bwareaopen.”

Given the index values “idx” for regions associated with viable targets, the properties of these regions can be recovered using using the following syntax:

```
statsTargets = stats(idx);
```

Where “statsTargets” is a structured array with fields matching “stats.” To recover individual binary images of a target, we can use the following:

```
for i = idx
      binTargets{i} = lbl == i;
end
```

Where “binTargets” will be a cell array whose elements are individual binary images.

Packaging Your Results

Combine your target thresholding and properties code a single MATLAB function whose input is a color image and whose outputs are properties for viable targets and binaries for each viable target. For example, using the variable names presented in this document:

```
[statsTargets,binTargets] = targetThreshold(im);
```

This function should accomplish the following:

1. Clearly define property threshold values (e.g. minArea, maxArea, minEcc, and maxEcc),
2. Perform color thresholding (using your "createMask" or alternately named function),
3. Fill any holes in the resultant binary
4. Impose a minimum area threshold
5. Label connected components
6. Calculate region properties of connected components
7. Impose remaining property thresholding
8. Output target properties and target binaries

Documenting Your Function

Take the time to write help documentation for your function. This will be a long semester, and it is easy to forget how to use even code that you have written. The commonly used format for MATLAB help documentation is as follows:

```
function [out1, out2, …] = funcName(in1, in2, …)
%FUNCNAME A brief description of what the function does
%    [out1, …] = FUNCNAME(in1, …) an in depth description of how the
%    function works given the specified input/output combination.
%    Any additional notes/comments/details about the aforementioned function
%    call.
%
%    [out1, …] = FUNCNAME(in1, in2, …) an in depth description of how the
%    function works given the specified input/output combination.
%    …
%
%    See also RELATEDFUNC1, FELATEDFUNC2, …
%
%    Contributor(s) names & date initially created
```

Bringing It All Together

Take the time to combine everything you have learned into a single MATLAB script to accomplish the following:

1. Initialize your camera and acquire an image
2. Create a figure named “EW309 – Target Threshold Test” (use the “Name” property)
3. Create an axes in your figure and adjust the “NextPlot” property
4. Add your image to the axes
5. Add a line object that we can use to display centroids (created using the line or plot functions). For example, given an axes object handle “axs,” to create a large magenta asterisk:

```
cnt = plot(0,0,'*m','Parent',axs,'MarkerSize',10);
```

6. Create an indefinite while loop (using `while true`)
    a. Get an updated image
    b. Update your image object color data with the new image (use the “CData” property)
    c. Perform your target thresholding
    d. Update your line object with the centroid of the first viable target (use the “XData” and “YData” property)
    e. Allow your plot to update using the `drawnow` command