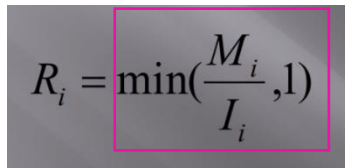


Part I

Explanation of Code:

The Swain-Ballard program I created follows the experiment just as explained in Swain-Ballard paper: Color Indexing. It takes the images and converts them into 3 bit images, just as Swain and Ballard did. By converting the image to a 3 bit image we lessen the amount of colors we have to index. Using 512 bins to sort the pixels into is far more manageable as apposed to an 8 bit image with approximately 16 million bins. I separated the method into two functions. hist_maker creates a 8X8X8 matrix which represents the histogram of the image, and backprojection takes the two histograms and produces the backprojected image from them. The values are first stored 8X8X8 array where each index corresponds with color triplet of the pixel. (MATLAB uses 1 index start so the index values are all increase by 1 to accommodate for this). For example if a Pixel is (0,3,2) (using (r,g,b) color channels) the pixel would be stored in index (1,4,3) of the matrix. Backprojection takes these two histogram matrixes and using the ratio formula (taken from Slide 13 of the Swain Lecture Slides) to find out how much of the object’s colors are in the original image. The line of code that corresponds to the equation, pictured below, has been outlined with the matching color box. After to find the object, I apply a Gaussian blur to the backprojected image to reduce the false positive areas from affecting the tracker. Then I search for the maximum intensity in the blurred image and return these coordinates



$$R_i = \min\left(\frac{M_i}{I_i}, 1\right)$$

```
function mat1 = hist_maker(row, col, data)
    %split into color channels
    red = data(:, :, 1);
    green = data(:, :, 2);
    blue = data(:, :, 3);

    %create a 3D matirix to hold all the for the red and green colors buckets
    mat1 = zeros(8,8,8);

    for i = 1:col
        for j = 1:row
            r = red(j,i)+1;
            g = green(j,i)+1;
            b = blue(j,i)+1;
            mat1(r,g,b) = mat1(r,g,b) +1;
        end
    end
end

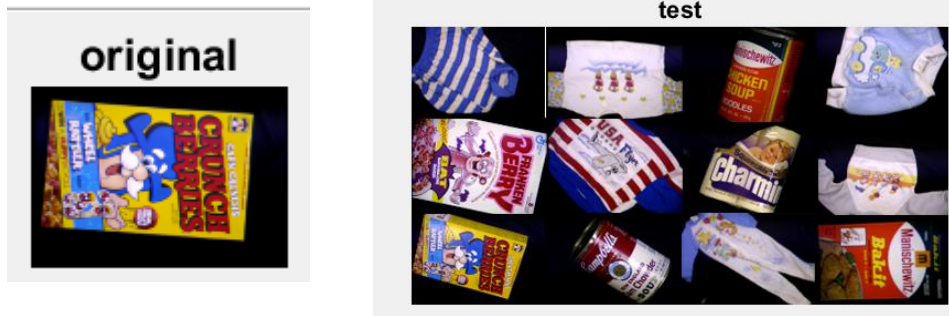
function bpimage = backprojection(hist_M, hist_I, M)

    [row, col, channel] = size(M);
    bpimage = zeros(row, col);
    %split into color channels
    red = M(:, :, 1);
    green = M(:, :, 2);
    blue = M(:, :, 3);

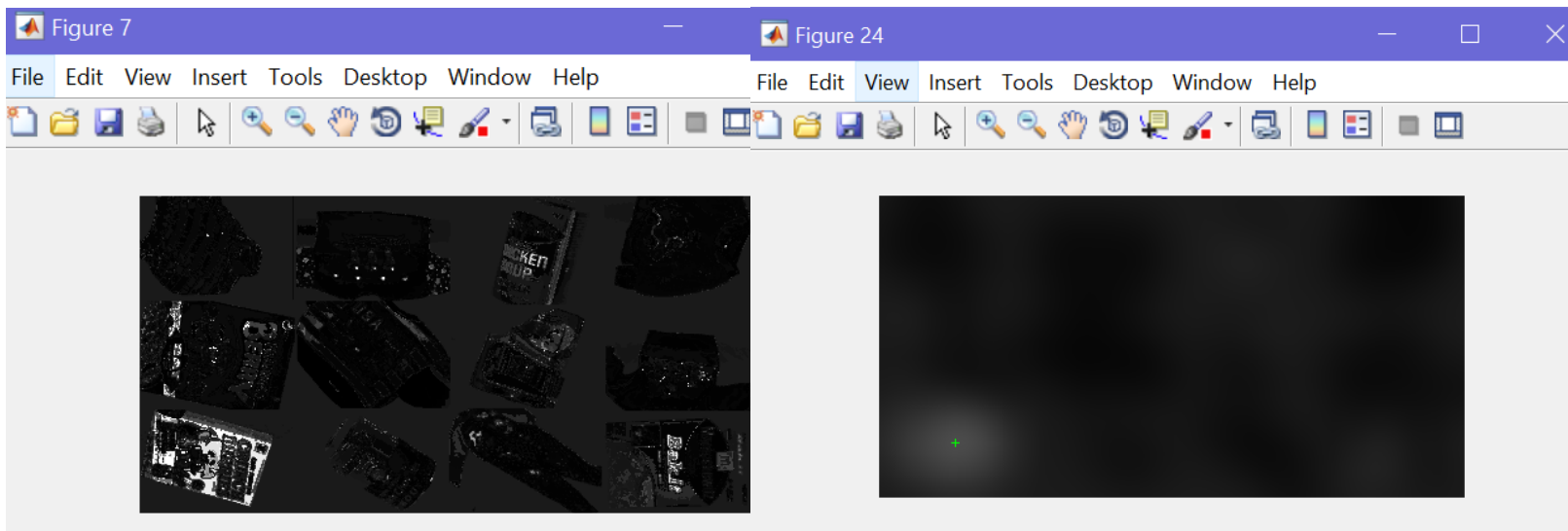
    for i = 1:row
        for j = 1:col
            r = red(i,j)+1;
            g = green(i,j)+1;
            b = blue(i,j)+1;

            bpimage(i,j) = min((hist_M(r,g,b)/hist_I(r,g,b)), 1);
        end
    end
end
```

Results – Searching For Captain Crunch in the Collage



The back-projected image produced and the object located applying a gaussian blur of 20 and searching for the maximum value pixel in the image (the green marker).



Part II

Explanation of Code:

The Swain-Ballard tracker I created utilizes the histogram function and backprojection function I created in Part I. I combined the histogram function and the backprojection function into one function called “tracker”. The method behind this program simply takes each frame and produce a backprojected image using the Swain-Ballard Method. The backprojected image clearly shows us where the cup is located, in most intensity. To help the program find the cup I apply a Gaussian blur to the backprojected image to reduce the false positive areas from affecting the tracker. Then I search for the maximum intensity in the blurred image and return these coordinates (which should be where the cup is located). The colored boxes around the code snippets match up with the highlighted lines in the explanation above:

```
function [x , y] = tracker(frame1, video, cup)

    [row, col, color, frame] = size(video); %get rows, columns, color channels, and frame number
    of video
    frame1 = floor((double(frame1) * 8)/265); %get 3bit image of the frame

    [rowc, colc, colors] = size(cup); %get rows, columns, color channels, of cup

    cuphist = hist_maker(rowc, colc, cup); %get the histogram for the cup
    framehist = hist_maker(row, col, frame1); %get the histogram for the frame

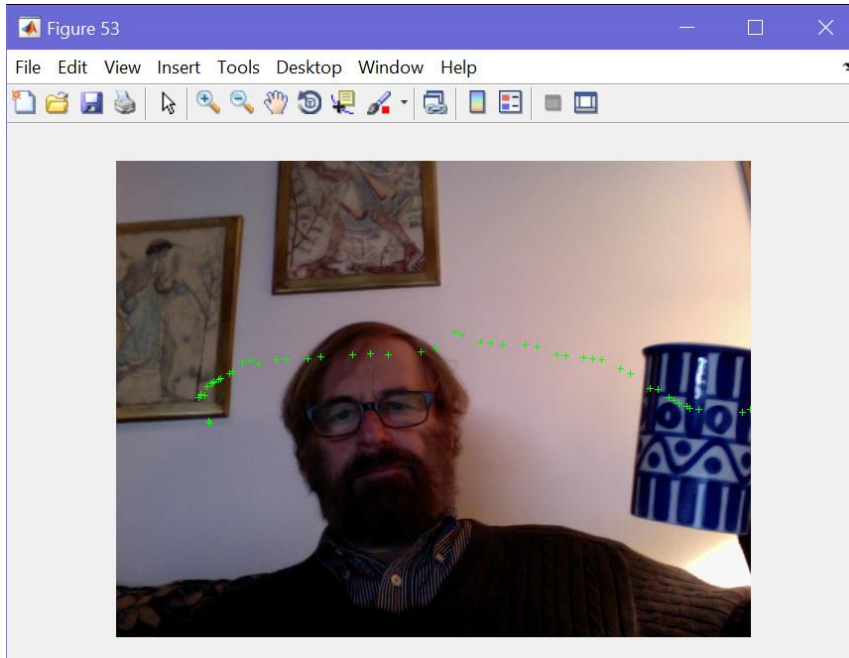
    backimage = backprojection(cuphist, framehist, frame1); %get the backprojection image of the
    frame
```

```
filter = imgaussfilt(backimage, 30); %apply a Gaussian Filter of 30 on the frame to reduce  
the chance of false positives
```

```
[maxpoint, coord] = max(filter(:)); %find the maximum intensity pixel in the picture  
[x , y] = ind2sub(size(filter),coord); %return the coordinates of brightest pixel is which  
should be where the object is
```

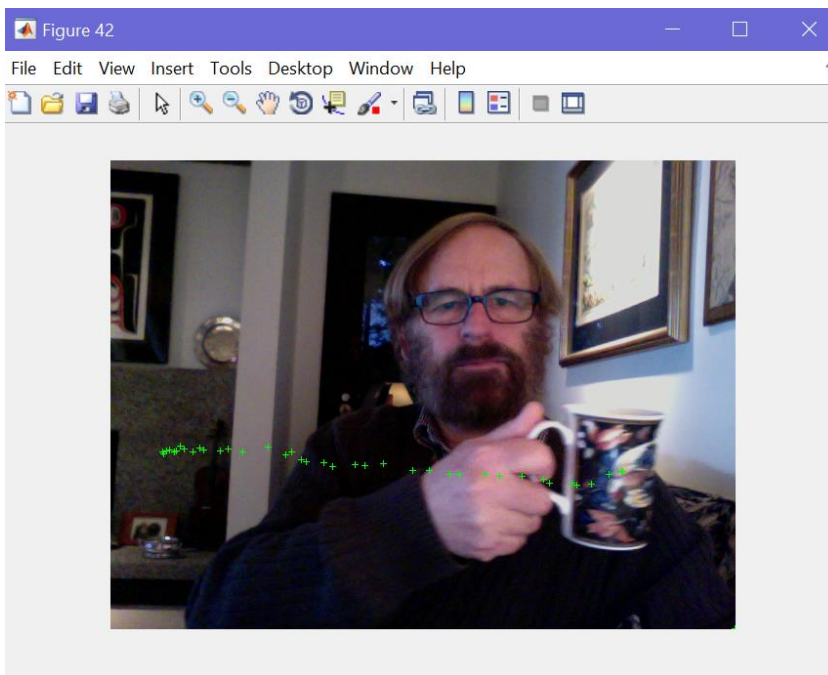
```
end
```

Results:



As we can see in the results of both the black cup and the blue cup, the Swain-Ballard Method of tracking an object produces reliable results. Because it is tracking the brightest pixel in the frame it potentially could provide unreliable results if there are other points of intensity in the back projected image. To maximize the best chance of finding the cup, I used a gaussian blur with an very high intensity. I used 30 chosen through experimentation different values. In addition I

found with this method, I found it was very easy to create unreliable tracking if the picture of the cup used to compare to the current frame had a window that was too big or too small. So the user must be aware of that when using ginput to grab the object window.



Part III

Explanation of Code:

For part III I utilized the simple mean shift method (formula taken from slide 16 of the Mean-Shift PowerPoint) and implemented it in my MATLAB function called `tracker_mean`. The line of code that correspond to the equation, pictured on the right, have been outlined with the matching color box. The kernel used was the Uniform Kernel.

$$\mathbf{m}(\mathbf{x}) = \frac{\sum_{i=1}^n \mathbf{x}_i g\left(\frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{h}\right)}{\sum_{i=1}^n g\left(\frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{h}\right)} - \mathbf{x}$$

```
function Mx = tracker_mean(backproj, x , y)
    %backproj is the backprojected image given by the swain and Ballard method in part two
    %y and x are the coordinates of the center of the cup

    PointX = backproj(x,y); %get the pixel value at the center of the cup

    [row, col, bpColor] = size(backproj); %get the number of rows and columns and color channels

    SumTop = [0,0]; %variable to hold the top part of the mean equation
    SumBot = 0; %variable to hold the bottom of the mean equation

    for i = max(x-80,1):min(x+80,row) %using two for loops go through every pixel in the window
        for j = max(y-45,1):min(y+60,col)

            PointXi = backproj(i,j); %get the value of the pixel at pixel(i,
            top = (-1*(((PointX-PointXi).^2)/10)); %compute the top part of the mean function
            (||x - xi||^2)/h % where h is an arbitrary constant

            top = top * [i,j]; %multiply by Xi

            SumTop = SumTop + top; %sum all the numerator values together

            SumBot = SumBot + (-1*(((PointX-PointXi).^2)/10)); %perform the bottom part of the
            mean equation % (||x-xi||^2)/h where h is arbitrary constant

        end
    end
    result = SumTop/SumBot; %divide the Sum of numerator by Sum of denominator

    Mx = result - [x,y]; %perform last step of mean shift function and subtract x from the result
    to get the mean shift vector
    return
end
```

The main part of this program looks at each frame in the video used, then using the backprojection method from part II produces a back projected image called `track_im` using the function `back_hist` which I wrote in part II.

The mean shift function `tracker_mean` is called in a small for loop 5 times to get the best approximation. `Mean_tracker` produces a vector that is added to the original x, y coordinates to shift the window to the area where the object has moved. (or at least to where mean shift has approximated it to have moved)

The following snippet of code from my program shows that main part where the actual mean shift method is being applied: (I have highlighted the line of code where `tracker_mean` is called)

```

for i = 1: frame_num

    %current_frame = blackcup(:,:,i); %uncomment this line if using backcup
    current_frame = bluecup(:,:,i); %uncomment this line if using bluecup

    %track_im = back_hist(current_frame, blackcup, cup);
    track_im = back_hist(current_frame, bluecup, cup);

    for r = 1:5 %run mean shift a few times to get a better approximation of denisty
        mx = tracker_mean(track_im, point(1), point(2));
        point = round(point + (mx)*0.2);

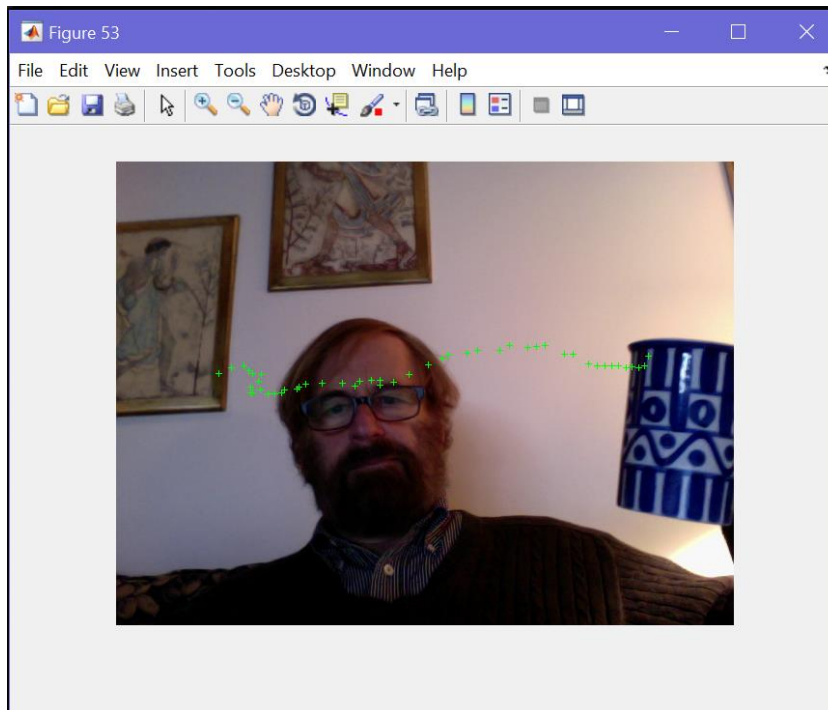
    end

    current_frame = insertMarker(current_frame,[point(2) point(1)]);
    figure;
    imshow(current_frame);
    lastframe = insertMarker(lastframe, [point(2),point(1)]);

end

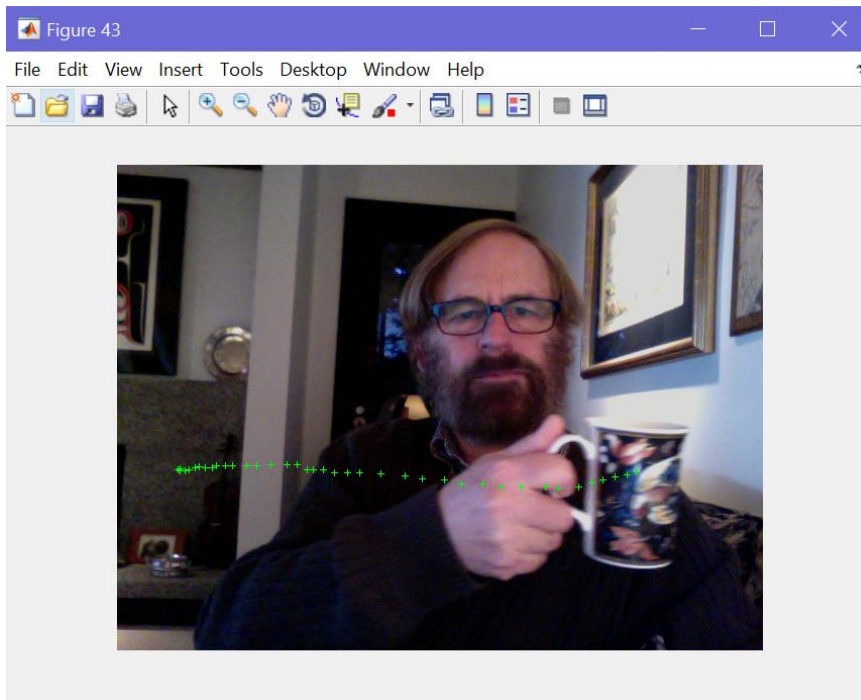
```

Results:



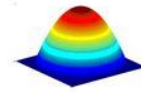
As we can see in both images on the left, the results of mean shift are quite positive. I did notice that the blue cup tracking was much more shaky than the black cup tracking (as you can see). I believe that this had to do with the lighting and how the blue cup seems slightly more reflective than that black cup. Thus the blue cup's backprojected image may have been changing more than in the backup video where the lighting was brighter and the cup was not as reflective.

In my original method I attempted to use a density kernel, specifically the Epanechnikov Kernel (formula shown below – taken from mean shift notes, slide 47)



implementation and use the uniform kernel formula (shown right – taken from mean shift notes, slide 47).

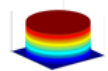
Epanechnikov kernel:



$$k(x) = \begin{cases} 1 - x & \text{if } \|x\| \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

kernel to get solid results. I'm not sure if it was my implementation that was incorrect or perhaps the kernel itself was not suited for this video but it left me with inaccurate and unreliable tracking. So I decided to keep that out of my

Uniform kernel:



$$g(x) = -k(x) = \begin{cases} 1 & \text{if } \|x\| \leq 1 \\ 0 & \text{otherwise} \end{cases}$$