Matt Shubert
ES 111 Final Project
April 30, 2013

Abstract

The purpose of this Matlab program is to take an image of an airplane in the sky, target it, and then draw a rectangle around the airplane. To do this a couple assumptions must first be made. The first is that the airplane is in the center of the image and not touching the edges. This allows the program to look at the 5 edge pixels and create a three dimensional matrix histogram where the indices represent the RGB values and the value of each indices represents the number of pixels in the background that have those RGB values. The background histogram is then dilated to increase the threshold of the background pixels. Then, each pixel in the image is compared to the background histogram and if the pixels in the image are more or less than the background histogram it is marked as a one. Then the program looks for the largest and smallest x and y pixels that are ones and uses those values to draw a box.

Introduction

The purpose of this Matlab program is to take an image of an airplane and target the airplane by determining where the airplane is in the image and drawing a box around the airplane. While currently the program only works for still images it could be implemented for video imaging. In addition to video tracking, the camera and tracking software could be integrated with a hardware mount allowing the camera to follow the airplane even if it were to leave the current field of view. The current program, which targets an airplane in a still image, works on the principal that while the airplane is in the sky it is photographed against a mostly solid blue background. This blue background is in contrast to the white, black, or colored body of the airplane. The program takes advantage of the difference in background and target and uses that to decide where the airplane is. Because of this, the program can also be used to identify any object that is different from it's background. For this paper, examples of airplanes in flight have been used to showcase the targeting abilities of the program.

Methods

As discussed earlier, the program takes advantage of the fact that airplanes in the sky are photographed against a different, solid background color. Using this idea and the assumption that the target is in the middle of the image and not touching the edges of the image, a three dimensional histogram of the background is created. The program takes the imported image and loops through 15 pixels around each edge of the photo. It then creates a three dimensional histogram matrix where the indices of the matrix are the red, green, and blue values of each pixel, and the value at each index is the number of pixels around the edge with that RGB value. For the sake of conserving memory and speeding up the total program time the original values for the RGB of 255 are converted to 16 so that the background histogram matrix is only a 16x16x16 matrix as opposed to a 255x255x255 matrix. So for example if the pixel in the first row and first column of the image had the RGB values of 12, 34, and 200 respectively each value would be divided by 16 and floored to get 1, 2, and 12. The background histogram function would then take the background histogram matrix and increment the value at (1, 2, 12) by one.

The next step is the histogram dilation function. This is to improve the values of the background histogram which leads to a cleaner image in the final product. The dilation of the background histogram takes the matrix made from the background histogram function and looks at each index and each index that it is touching. If the index is touching another index that has a value then the index being looked at by the function is incremented by one. This means that if an index in the background histogram is zero, but is "touching" (meaning that the index above below, in front of, behind, to the right, or left of) an index that does have a value in the background histogram, the value is increased to one. For the best results this is done a total of 3 times.

After the dilation of the background histogram the program then runs the target function to determine which pixels are target pixels in the image and which are background pixels. This is done by simply looking at the background histogram and setting any indices with values greater than one to zero, and any values of zero to 1 and storing them in a new matrix called the probability histogram. This means that now every RGB pixel value that is a target is 1 and every

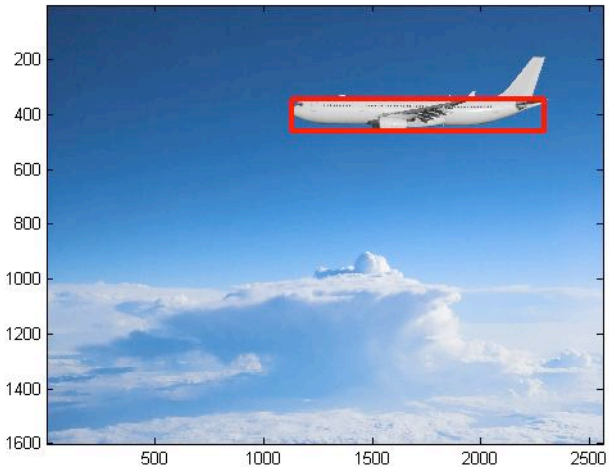
RGB value that is a background pixel is 0. The target function then looks at each pixel in the original function and compares it's RGB value to the probability histogram and creates a new matrix called sil where the pixels are either 1 for target pixels or 0 for background pixels. To help with some noise pixels an erosion function is then performed on the sil matrix. The erosion function is just the opposite of the dilation function performed on the background histogram. This means that if there is an index with a value of 1 that is not touching another pixel with a value of 1 it will be reassigned a value of zero. Once this is done the sil matrix is then dilated to obtain a larger target for the final process of the program.

The final process in the targeting program is to draw the box around the target. This is done in the draw target function by importing the sil matrix after the erosion and dilation. The function then sums the values in each row and column going from top to bottom, bottom to top, right to left, and left to right. The first summed row going from top to bottom that does not equal 0, meaning that there is one index with a value of 1, or a target pixel, becomes the maximum y value. The bottom to top loop gives the minimum y value, the right to left gives the maximum x value and the left to right loop gives the minimum x value. These four values are then used in the built in Matlab function rectangle to draw the box around the image.

Results

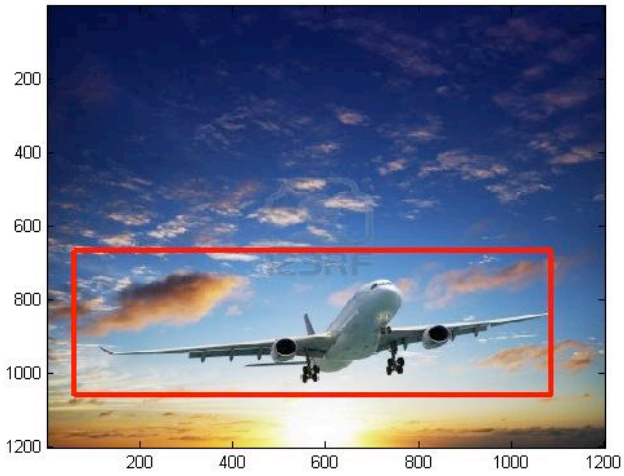
The program was tested on 7 different, randomly selected, images from google of airplanes flying. The program worked for 6 of the 7 selected images. Below are selection of three of the six tested images that were successfully targeted with the program.

Original Photo



After Targeting Program

Original Photo



After Targeting Program

Original Photo

After Targeting Program

While the program worked for the majority of images tested one of the randomly selected 7 images was not able to be targeted. During the target draw function the program outputted an error saying that it was trying to draw something out of the bound of the image. This meant that the program had assigned target pixels some where along the edges of the picture while the airplane was not touching the edge. Below is the image that could not be targeted.

The only one of seven selected images that was not successfully targeted

It can be seen, however, that this is not a typical picture of an airplane in the sky and the issues probably came from the reflection of the sun running from the middle of the image down to the bottom.

Conclusion

I believe that the program worked successfully for the goal it was meant to accomplish given the amount of pictures it was successfully able to target and the complicatedness of some of the photos it could successfully target. Out of the seven randomly selected photos it was able to target 6 of them with very good accuracy. There was one photo of those six where the accuracy was a little off given the effects of the sun in the picture, but the airplane still ended up in the targeting region. The only one of the selected test photos it could not target was very busy with unusual visual elements not common to a standard picture of an airplane in the sky.

Aircraft_Targeting.m

```
%clear all variable
tic
close all
clear all

%Import Image
targetImageSingle = imread('airplane2.jpg');
%Convert Image Matrix to double
targetImage = double(targetImageSingle);

%get red, green, and blue background values using background function
[bH] = backgroundHistogram(targetImage);

% %dilate the background histogram
[bHDilation] = histogramDilation(bH);
for ctr = 1:2
    [bHDilation] = histogramDilation(bHDilation);
end
%get silhouette of target
[sil] = target(targetImage, bHDilation);

%Clean up noise pixels in sil using erosion function
[silErosion] = erosion(sil);


%make target larger
[silDilation] = dilation(silErosion);

drawTarget(silDilation, targetImageSingle)

toc
```

---

backgroundHistogram.m

```
function[bH] = backgroundHistogram(x)
%Takes a three dimensional matrix of red, green, and blue values from an
%image and out puts a Histogram Matrix where the indices are the RGB values
%and the value in the indicies is the number of pixels which have those
%corresponding RGB values

%Gets the size of the image matrix
[rows,columns,~] = size(x);

%Creates the empty Background Histogram Matrix
bH = zeros(16,16,16);

%Looks though the 15 pixels on the edge of the image and converts the
%256 values to 16.  Then takes the red, green, and blue value of the
%pixel in the image matrix and increments the value of the Histogram
%matrix with those indices by one.
```

```
for ctr = 1:columns
    for ctr2 = 1:15
        rindex = floor((x(ctr2,ctr,1))/16)+1;
        gindex = floor((x(ctr2,ctr,2))/16)+1;
        bindex = floor((x(ctr2,ctr,3))/16)+1;
        bH(rindex,gindex,bindex) = bH(rindex,gindex,bindex)+1;

        rindex = floor((x(end-ctr2+1,ctr,1))/16)+1;
        gindex = floor((x(end-ctr2+1,ctr,2))/16)+1;
        bindex = floor((x(end-ctr2+1,ctr,3))/16)+1;
        bH(rindex,gindex,bindex) = bH(rindex,gindex,bindex)+1;
    end
end
for ctr = 1:rows
    for ctr2 = 1:15
        rindex = floor((x(ctr,ctr2,1))/16)+1;
        gindex = floor((x(ctr,ctr2,2))/16)+1;
        bindex = floor((x(ctr,ctr2,3))/16)+1;
        bH(rindex,gindex,bindex) = bH(rindex,gindex,bindex)+1;

        rindex = floor((x(ctr,end-ctr2+1,1))/16)+1;
        gindex = floor((x(ctr,end-ctr2+1,2))/16)+1;
        bindex = floor((x(ctr,end-ctr2+1,3))/16)+1;
        bH(rindex,gindex,bindex) = bH(rindex,gindex,bindex)+1;
    end
end
```

---

histogramDilation.m

```
function [bHDilation] = histogramDilation(bH)
    %inputs the background histogram and dilates outputting a bHDilation

    [rows,columns,sheets] = size(bH);
    bHDilation = bH;

    %looks at each index above, below, to the right and left and in front of
    %and behind an index.  Sums the value of all 6 and if it's greater than
    %or equal to one, ie one or more of the six indices had a value of 1,
    %the index being tested is changed to have a value of 1
    for sctr = 1:sheets
        for rctr = 1:rows
            for cctr = 1:columns
                total = (bH(min(rctr+1,16),cctr,sctr))+
(bH(max(rctr-1,1),cctr,sctr))+(bH(rctr,min(cctr+1,16),sctr))+
(bH(rctr,max(cctr-1,1),sctr))+(bH(rctr,cctr,min(sctr+1,16)))+
(bH(rctr,cctr,max(sctr-1,1)));
                if (total >= 1)
                    bHDilation(rctr,cctr,sctr) = 1;
                end
            end
        end
    end
```

---

target.m

```
function [sil] = target(x, bH)
    %takes inputs (image, backgroundHistogram) and out puts a matrix of 1s
    %and zeros where 1 is a target pixel and 0 is a background pixel

    %creates a matrix with the RGB values of the image matrix coverted from
    %255 values to 16
    x16 = floor(x./16)+1;

    %gets the values for the rows and columns of the image
    [rows,columns,~] = size(x);

    %logically, if the value of an index in the background histogram is 0,
    %meaning its a target pixel, the value in the probabilityHistogram is
    %changed to 1.  If the value of the index in the background histogram
    %is anything other than 0, ie a background pixel, it is set to zero
    probablilityHistogram = bH < 1;

    %creat a matrix fro the siloute of the image
    sil = zeros(rows, columns);

    %creates a matrix with the same dimensions as the image.  Looks at each
    %RGB value of the image and if that index in the probability histogram
    %is 1, the coresponding pixel in the siloute matrix is set to 1.
    for rctr = 1:rows
        for cctr = 1:columns
            sil(rctr, cctr) = probablilityHistogram(x16(rctr,cctr,1),
x16(rctr,cctr,2), x16(rctr,cctr,3));
        end
    end
```

erosion.m

```
function [silErosion] = erosion(sil)
    %Inputing the sil matirx and eroding noise pizels

    %Gets the size of the matrix and creats a new matrix with the same
    %dimensions
    [rows,columns] = size(sil);
    silErosion = sil;

    %looks at each index in the matrix and if it's not touching another
    %index with a value of one, it is reasigned a value of 0
    for rctr = 2:(rows-1)
        for cctr = 2:(columns-1)
            total = (sil(rctr+1,cctr))+(sil(rctr-1,cctr))+(sil(rctr,cctr+1))+
(sil(rctr,cctr-1));
            if (total < 4)
                silErosion(rctr,cctr) = 0;
            end
        end
    end
```

---

dilation.m

```
function [silDilation] = dilation(silErosion)
    %takes imput silErosion which has cleaned up noisy pixels and re
    %exapnds to get a better image of the target

    %sets the variable rows and columns to the number of rows and columns in
the matrix silErosion
    [rows,columns] = size(silErosion);

    %creates a new matrix the same size as silErosion
    silDilation = silErosion;

    %looks at each index above, below, to the right and left and infront of
    %and behind an index.  Sums the value of all 6 and if it's greater than
    %or equal to one, ie one or more of the six indicies had a value of 1,
    %the index being tested is changed to have a value of 1
    for rctr = 2:(rows-1)
        for cctr = 2:(columns-1)
            total = (silErosion(rctr+1,cctr))+(silErosion(rctr-1,cctr))+
(silErosion(rctr,cctr+1))+(silErosion(rctr,cctr-1));
            if (total >= 1)
                silDilation(rctr,cctr) = 1;
            end
        end
    end
```

---

drawTarget.m

```
function drawTarget(silDilation, targetImageSingle)
    %Takes the silDilation matrix of target pixels and background pixels
    %and finds the left, right, top, and bottom most pixels of the target

    %Get the size of the matrix
    [rows, columns] = size(silDilation);

    %pre allocation
    theSum = 0;
    yMax = 0;
    yMin = 0;
    xMax = 0;
    xMin = 0;

    %the first row, looping from top to bottom, that adds up to something
    %other than 0, meaning that there is a target pixel in the row, becomes
    %the max y value for the target rectangle
    for ctr = 1:rows
        theSum = sum(silDilation(ctr,:));
        if (theSum ~= 0)
            yMax = rows-ctr;
            break
        end
    end
```

```
    %the first row, looping from bottom to top, that adds up to something
    %other than 0, meaning that there is a target pixel in the row, becomes
    %the min y value for the target rectangle
    for ctr2 = 1:rows
        theSum = sum(silDilation(end-ctr2,:));
        if (theSum ~= 0)
            yMin = ctr2;
            break
        end
    end


    %the first column, looping from right to left, that adds up to something
    %other than 0, meaning that there is a target pixel in the column,
    %becomes the min x value for the target rectangle
    for ctr3 = 1:columns
        theSum = sum(silDilation(:, ctr3));
        if (theSum ~= 0)
            xMin = ctr3;
            break
        end
    end


    %the first column, looping from left to right, that adds up to something
    %other than 0, meaning that there is a target pixel in the column,
    %becomes the max x value for the target rectangle
    for ctr4 = 1:columns
        theSum = sum(silDilation(:,end-ctr4));
        if (theSum ~= 0)
            xMax = columns-ctr4;
            break
        end
    end

    %compute the height and width of the rectangle to be used in the
    %rectangle function
    height = yMax - yMin;
    width = xMax - xMin;

    %image the target image and draw the rectangle over it
    image(targetImageSingle)
    rectangle('Position', [xMin,rows-yMax,width,height],'LineWidth',3,
'EdgeColor', 'r')
```

Bibliography

Matlab help site. http://www.mathworks.com/help/matlab/