Computer Vision Lab: Harris Corner Detection

CSCI 380 Computer Vision

1. Create a new matlab script 'harrisCorner.m'
2. Read in the Penguins.jpg image from the Windows sample pictures folder using the imread command (store it in 'myImage')
3. Convert the image to grayscale using the *rgb2gray* command.
4. Cast your variable to a double and reassign it to itself.

   myImage = double(myImage)

5. Create the x and y derivative filters

```
dxFilter = [-1 0 -1; -1 0 1; -1 0 1];
dyFilter = dxFilter';
```

6. Create an X derivative image and a Y derivative image by convolving the appropriate filters with the grayscale penguin image. Use the following names for your new images:

   myImageDerivativeX

   myImageDerivativeY

7. Now that we have the derivatives, we can calculate A, B, C from the book.

   A = myImageDerivativeX .^2
   B = myImageDerivativeY .^2
   C = myImageDerivativeX .* myImageDerivativeY

8. After you calculate the above you want to apply Gaussian smoothing (using a Gaussian Filter) using the conv2 command. Store each of the smoothed derivatives in: smoothedA, smoothedB, smoothedC. You can use the following command to create your Gaussian filter:

```
gaussianFilter = fspecial('gaussian');
```


dbobeldyk@davenport.edu

9. In order to compute the Corner Response Function, we use the following formula (define alpha = 0.04):

   cornerResponseFunction = A .* B – C^2 - alpha * $(A + B)^2$

   **Note: Please be sure to use the smoothed versions of A, B & C for the above formula

10. Next we need to create our isLocalMax function. The Corner Response function usually will have several 'corners' all that appear in the same area. This function will help us to determine which is the best corner for our program to use. Use the following code for 'isLocalMax':

```
function [myBool] = isLocalMax(harrisMatrix, u, v)
    height = size(harrisMatrix, 1);
    width = size(harrisMatrix, 2);

    if(u <= 1 || u >= height || v<=1 || v>=width)
        myBool = false;
    else
        pix = reshape(harrisMatrix, height*width, 1);    %return the image as
a 1 dimensional array (like the book does)
        i0 = (v-1)*height+u;
        i1 = v*height+u;
        i2 = (v+1)*height+u;

        cp = pix(i1);

        myBool = (cp > pix(i0-1) && cp > pix(i0) && cp > pix(i0+1) && cp >
pix(i1-1) && cp > pix(i1+1) && cp > pix(i2-1) && cp > pix(i2) && cp >
pix(i2+1) );
    end
end
```

11. We use this function to help us determine the best corners to use. We do this by stepping through our cornerResponseFunction matrix and check to see if it's value is greater than a certain threshold and it is a local maximum (this is checked by calling the isLocalMax function you just created)
12. Implement the *pseudo code* listed below in order to perform the above task:

    Initialize totalCorners to zero
    For each u:
     For each v:
        If (cornerResponseFunction (u, v) > threshold && islocalMax(cornerResponseFunction, u, v)
            Increment totalCorners by 1.
            Add the corner to our list (each corner contains u, v, q)
           `cornerList(totalCorners).x = u;`

dbobeldyk@davenport.edu

```
cornerList(totalCorners).y = v;
cornerList(totalCorners).q = q;
```

Output the total number of corners that passed the threshold and isLocalMax

*NOTE: Use an initial threshold of 200000 (threshold = 200000)

13. Sort the corners by their respective q value in descending order

14. Create a new variable goodCorners and initialize it to an empty array:

```
goodCorners = [];
```

15. Our corner response function may return a lot of corners that can be quite close to each other. What we'd like to do next is define a minimum distance that we want these features (corners) to be from each other. We'll then want to step through each of our corners and remove the ones that are close too close to each other. We do this by starting with the strongest corner that we found (the first one in the list because it's sorted) and remove other corners that are within the minimum distance to this point.

16. First thing we will want to do is initialize our minimum distance:

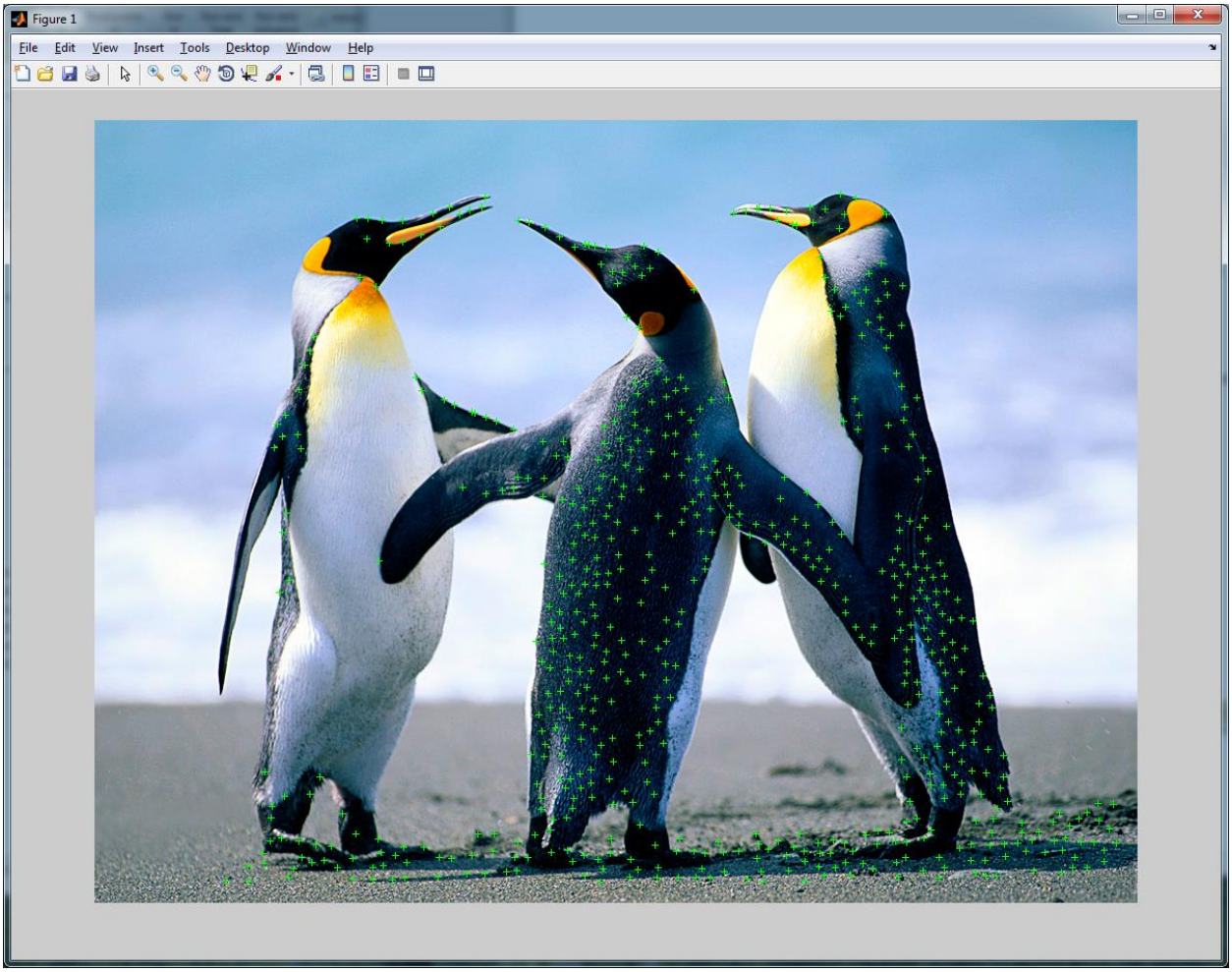
```
minDistance = 10;
```

17. Copy and paste the following code into your program and **complete the code**

```
while(size(cornerList,2) > 0)
    c1 = cornerList(1);
    cornerList = cornerList(2:end);  %remove the first one

    goodCorners = [goodCorners c1];

    cornersToRemove = [];
    for i=2:size(cornerList,2)
        if( <<<complete code>>>>)
            cornersToRemove = [cornersToRemove i];
        end
    end

    cornerList(cornersToRemove) = [];  %remove the corners that we found
that are too close

end
```


dbobeldyk@davenport.edu

18. Step through the goodCorners list and mark each of the corners in the image with a green ‘cross’ or x that is 3 pixels in width and height.
19. Display the image to the user, it should look similar to:



If you complete early you may:

1. Play with modifying the ‘minDistance’ , threshold, and k values. Observe the changes made to the output
2. Implement the above lab using the ‘corners’ command.
3. Instead of representing the corners in the image with a cross or x, numerically represent the corner in the image. (ie the best corner would have a ‘1’ , the second best would have a ‘2’).

dbobeldyk@davenport.edu