Using MATLAB to Measure the Diameter of an Object within an Image

Keywords: MATLAB, Diameter, Image, Measure, Image Processing Toolbox

Author: Matthew Wesolowski
Date: November 14th 2014

Executive Summary

Measuring objects within an image or frame can be an important capability for many applications where computer vision is required instead of making physical measurements. This application note will cover a basic step-by-step algorithm for isolating a desired object and measuring its diameter.

Objective

Through this application note you will be able to write a MATLAB script file to import an image, segment the image in order to isolate the desired object from its background and then use the MATLAB functions that come with the Image Processing Toolbox to determine the objects diameter. It is assumed in this Application Note that the reader has a basic knowledge of MATLAB.

Introduction

"MATLAB is a high-level language and interactive environment for computer computation, visualization, and programming. Image Processing Toolbox is an application available for use in MATLAB, which provides a comprehensive set of reference-standard algorithms, functions, and apps for image processing, analysis, visualization, and algorithm development." Using these tools provides a fast and convenient way to process and analyze images without the need for advanced knowledge of a complex coding language.

Hardware

Operating System: Windows XP or better, Mac OS X Lion or better
Processors: Intel or AMD x86 processor
Disk Space: 4 GB or better
RAM: 2048 MB at least recommended
Graphics Card: Hardware accelerated graphics card supporting OpenGL 3.3 with 1 GB GPU memory recommended.

Software

MATLAB 2014a (Covered with this tutorial)
Image Processing Toolbox (Downloadable application)

Process

Import Image

Open the MATLAB software and in the application section; download the Image Processing Tool Box. Create a new MATLAB script file. Refer to *Figure 2* to begin adding code to import the desired image to measure into the MATLAB workspace. The first few lines clear the workspace to remove any previous variables and clear the command window. It is important that the Current Folder that you are working out of be the folder that contains both the script file and image. The command *imread* reads an image and converts it into a "3-dimensional" matrix in the RGB color space. The image used in this tutorial is *ball.jpg (Figure 1),* which is a 534 by 401 pixel image. The *imread* function converts this into a matrix that is 401x534x3 (Rows x Columns x RGB). The final dimension (RGB) corresponds to a red, green and blue intensity level. Use *imshow* to view the produced image in a new window.

Figure 1; Original Image, ball.jpg

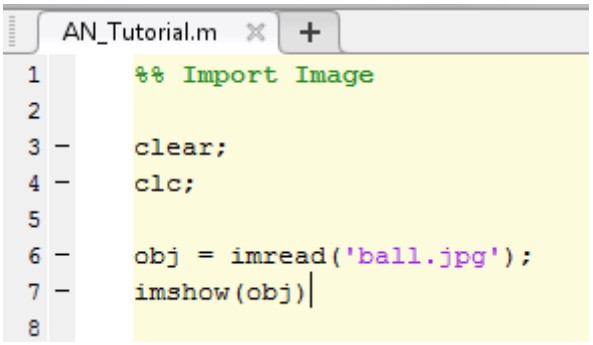

Figure 2; Code to Import an Image

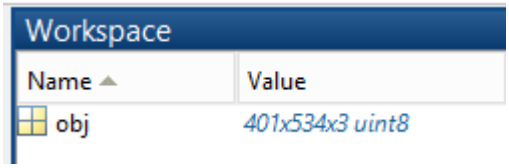

Figure 3; obj variable in workspace

Segment Image

Follow the code in *Figure 4* below to segment the image into a binary image to differentiate the background from the desired objects. The first step taken is to divide the image into three images based on the intensities of each red, green and blue component within the image. This is Color Based Image Segmentation. You can see from *Figure 5* that the blue plane is the best choice to use for Image Thresholding because it provides the most contrast between the desired object (foreground) and the background. Image Thresholding takes an intensity image and converts it into a binary image based on the *level* desired (See line 25). A value between 0 and 1 determines which pixels (based on their value) will be set to a 1 (white) or 0 (black)). To choose the best value suited for your application right-click on the value and at the top of the menu and select "Increment Value and Run Section". Set the increment value to 0.01 and choose the best value at which to threshold. *Figure 5* shows the result of the Image Thresholding at 0.37. You can see that the image (Top-right of *Figure* 6) has been segmented between the object we desire to measure and the background.

```
 9      %% Segment Image
10
11      %Divide image "obj" into its respective RGB intensities
12 -    red = obj(:,:,1);
13 -    green = obj(:,:,2);
14 -    blue = obj(:,:,3);
15
16 -    figure(1)
17 -    subplot(2,2,1); imshow(obj); title('Original Image');
18 -    subplot(2,2,2); imshow(red); title('Red Plane');
19 -    subplot(2,2,3); imshow(green); title('Green Plane');
20 -    subplot(2,2,4); imshow(blue); title('Blue Plane');
21
22      %Threshold the blue plane
23 -    figure(2)
24 -    level = 0.37;
25 -    bw2 = im2bw(blue,level);
26 -    subplot(2,2,1); imshow(bw2); title('Blue plane threholded');
27
```

Figure 4; Code to Segment the Image

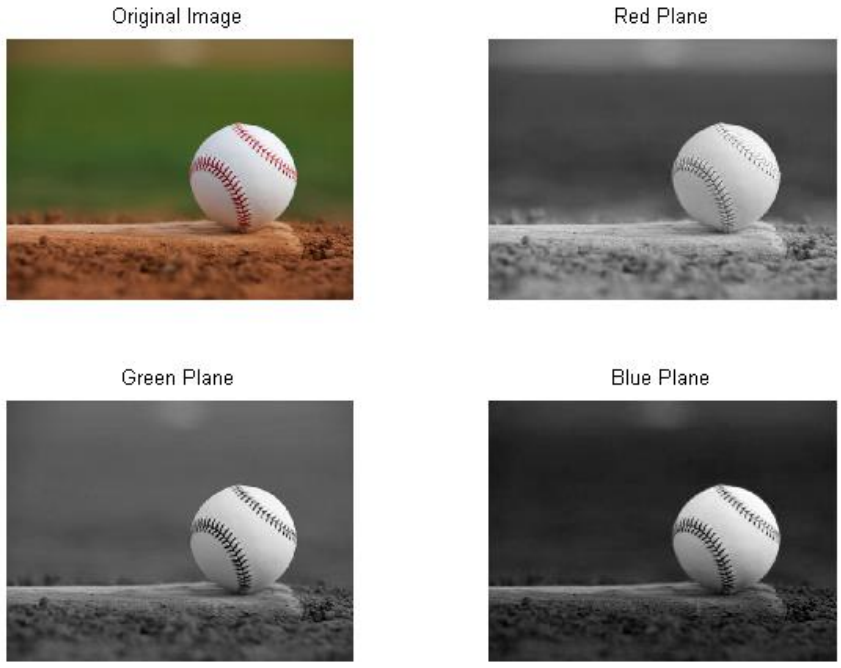

Figure 5; Color Thresholding

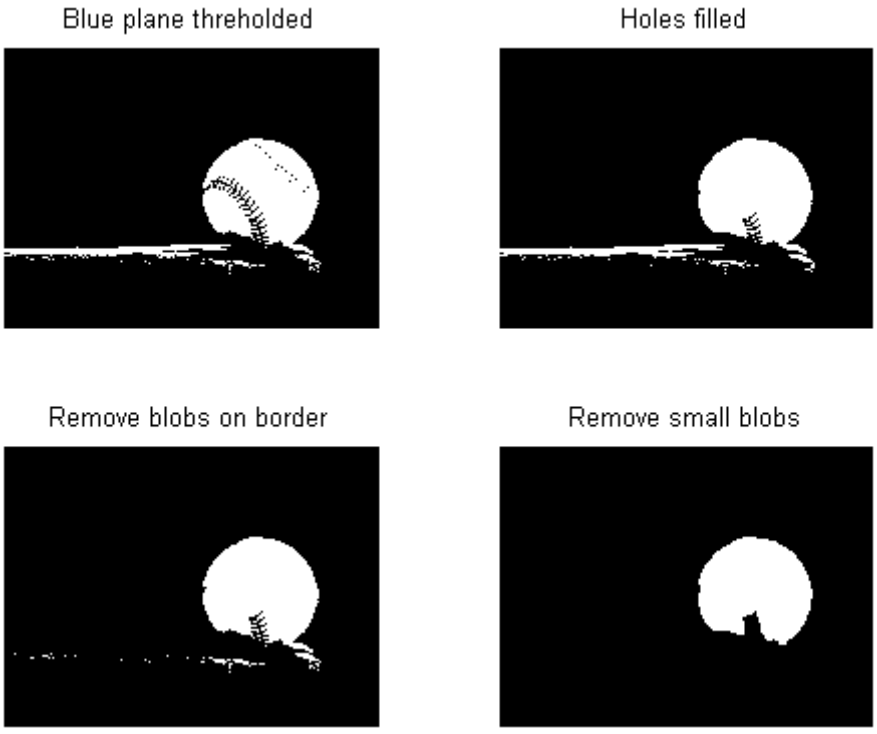

Figure 6; Complete Segmentation and Cleanup Image

Segmentation Continued (Remove Noise)

As you can see from the top-left image in *Figure 6* there is quite a bit of "noise" and we need to clean the image up significantly to improve the accuracy of our diameter measurement. Refer to *Figures 6 & 7* on the procedures taken to clean up the image and provide a more uniform blob to analyze. Blobs in this document are any collection of white pixels that touch to create a cohesive and distinct object.

```
28      %% Remove Noise
29
30      %Fill any holes
31 -    fill = imfill(bw2,'holes');
32 -    subplot(2,2,2); imshow(fill); title('Holes filled');
33
34      %Remove any blobs on the border of the image
35 -    clear = imclearborder(fill);
36 -    subplot(2,2,3); imshow(clear); title('Remove blobs on border');
37
38      %Remove blobs that are smaller than 7 pixels across
39 -    se = strel('disk',7);
40 -    open = imopen(fill,se);
41 -    subplot(2,2,4); imshow(open); title('Remove small blobs');
42
```

Figure 7; Code to Cleanup Image/Remove Noise

Measuring Image

The image in the bottom-right corner of *Figure 6* is the result of all image segmentation and cleanup procedures to provide one distinct and cohesive blob, which represents the ball in the original image. Having the original image in a binary form such as this will make it easy for other functions built into MATLAB to quickly analyze the region and a host of different information. The *regionprops* function is the tool that will provide the *MajorAxisLength* of the blob in the image. As you can see, by not suppressing line 45 *(Figure 8)* with a semi-colon, the diameter is displayed in the Command Window *(Figure 10)*.

```
43      %% Measure Object Diameter
44
45 -    diameter = regionprops(open,'MajorAxisLength')
46
47      %Show result
48 -    figure(3)
49 -    imshow(obj)
50 -    d = imdistline; %Include a line to physically measure the ball
51
```

Figure 8; Code to Measure the Object Diameter

Results

The diameter is now displayed in the Command Window to be approx. 170 pixels across. This was verified in *Figure 9* by using the *imdistline* function in line 50 *(Figure 8).* As you can see between the two figures, the value calculated by the code was very close to the manual measurement in *Figure 9.*

Figure 9; Original Image Manually Measure

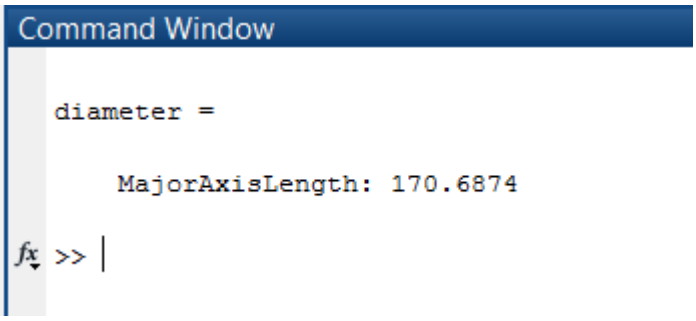

Figure 10; Diameter Output in the Command Window

Recommendations

There are a multitude of options within the *regionprops* function that can output other measurements besides the *MajorAxisLength*. Type *Help regionprops* in the Command Window to get more information about the function. *Help* can be used with any function to obtain more information. Provide comments for each section of code as demonstrated. Comments provide a quick explanation of each section of code that helps other users debug and understand the code.

References

1. *Image Processing Made Easy*. Perf. Andy Thé. *Mathworks*. N.p., 15 Oct. 2014. Web. 07 Nov. 2014. <http://www.mathworks.com/videos/image-processing-made-easy-81718.html>.