Test Plan For NASA Image Droplet Analysis

Dream Team 2015

Client: Benjamin D. Shaw

RAMYA BHASKAR RBHASKAR@UCDAVIS.EDU
AMANDA HO AYJHO@UCDAVIS.EDU
WILLIE HUEY WHUEY@UCDAVIS.EDU
RYLAN SCHAEFFER RYSCHAEFFER@UCDAVIS.EDU

(Dated: April 1, 2015)

`Summary of Product to be Tested`

The product will be used to identify and calculate the change in fuel droplet diameter as a function of time. Droplet images are in the form of TIFF image sequences. The sequences of interest are provided by NASA through Professor Benjamin Shaw.

Each sequence contains approximately one thousand individual images, each exhibiting minimal to heavy sooting. Sooting is a phenomenon in which the fuel droplet becomes obscured because of the ongoing combustion reaction.

Not all images will be grayscale—some are taken with other types of cameras and other methods will need to be implemented to detect the droplets captured with different cameras.

I. RESOURCES REQUIRED FOR TESTING

MATLAB with image processing toolbox http://www.mathworks.com/products/matlab/
NASA image sequences must be acquired from Benjamin Shaw or NASA
Note: The computers we tested our program on had 8GB of RAM. Using a computer with less RAM may impact time to complete.
OpenCV http://opencv.org/

II. ESTIMATED PERSON-HOURS FOR TESTING

Since our project is based on comparing the circle of best fit to the droplet, the testing process is not independent of the development process. As a consequence, we expect to spend over 30 hours of testing, as a rough estimate.

III. PACKAGING, BUILDING, CONFIGURATION AND OPTIONS

A. Github Repo

https://github.com/BamyaRhaskar/-15SD

The public availability of the code has yet to be determined. Professor Shaw will let us know whether the code should be publicly released.

B. Checkout, Build on the Major Elements

Matlab: The repository for the code needs to be downloaded. After they have been downloaded, the folders containing the image sequences must be placed in the Matlab folder (For Windows, this is located in the Documents folder). Run the visualize_circle.m script and and enter the images you want to analyze.

OpenCV: The repository for the code first needs to be downloaded. Once that is done, we can focus on the process of installing OpenCV (for Mac's only at the moment).

Note, that this will require installation of several other helper applications.

1. First, make sure that the Mac OS X operating system is up to date, with Yosemite installed.
2. Install XCode if it hasnt been installed already.
3. Make sure that Command Line Tools has also been installed (can be completed from within the XCode Application).

Once the above is done, the application CMake will have to be installed. In order to do this, simply visit website below and follow the sites instructions.

http://www.cmake.org/download/

With CMake installed, we can then proceed to install OpenCV.

Download OpenCV somewhere on computer. Create two new folders in the OpenCV directory, called SharedLibs and StaticLibs. Open CMake application, click browse source and go to the OpenCV folder. Click browse build and go to StaticLibs folder. Click configure button, click unix makefile (assuming working with a Unix environment). After tests are performed, uncheck the following boxes in red.

1. BUILD_SHARED_LIBS
2. BUILD_TESTS
3. WITH_1394
4. WITH_FFMPEG
5. Add an SDK path to CMAKE_OSX_SYSROOT : /Applications/Xcode.app/Contents/Developer/ Platforms/MacOSX.platform/Developer /SDKs/MacOSX10.9.sdk
6. Add x86_64 to CMAKE_OSX_ARCHITECTURES

Click Configure again, then Click Generate. Now, open Terminal and type:

1. cd <path/to/your/opencv/staticlibs/folder/>
2. make
3. sudo make install

Type in password and StaticLibs libraries should then be installed. Repeat same above process for SharedLibs, except when you first click browse build, go to SharedLib folder instead, and in the CMake phase, recheck BUILD_SHARED_LIBS.

If you are still having trouble with installation please refer here for additional support:

http://blogs.wcode.org/2014/10/howto-install-build-and-use-opencv-macosx-10-10/

With OpenCV now successfully installed, the code can be executed by the following steps:

1. navigate to directory with appropriate code
2. g++ hough.cpp
3. then you get an executable called hough
4. and then you get an executable called: ./hough
5. and you run as: ./hough *file name that we want to analyze*

IV. HOW TO INSTALL AND CONFIGURE THE ELEMENTS

Matlab: Currently, the path names are hardcoded to make testing easier. The final program should allow the user to input the directory they want to work on. Directory should be included in the Matlab directory for ease.

OpenCV: Covered in above section.

V. SANITY CHECK BASIC FUNCTION TO ENSURE ITS BUILT & INSTALLED PROPERLY

The code is well documented, with a number of user prompts. Code should execute without error if built properly.

A. Process for defect reporting & repair

Matlab Script Failure: To ensure that the program runs smoothly make sure there are no errors when running the script. If there are errors, ensure that the directory with the image sequences are within the Matlab path, if not, add the directory to the Matlab path. When running the program, ensure at the beginning that you input image numbers that are within the bounds of the images in the directory.

OpenCV: If there are issues configuring OpenCV please go through the instructions again to make sure you haven't missed anything. If you have further issues please contact Ramya Bhaskar.

B. Quality Control

There are two types of bugs: false positives and failures. False positives are difficult to detect because the algorithm reports success (a circle of best fit has been found) and can only be caught by manually inspecting the image or checking the position of the circle and its radius. If a false positive is reported, that specific image sequence will need to be examined to understand why the algorithm produced a false positive.

Failures are also difficult to decide what to do with. In some cases, the image signal isnt present and nothing can be said about the placement of the circle of best fit. But in other cases, the algorithm might report that a circle of best fit does not exist, despite the droplet being clearly identifiable. In the latter case, we will approach the error similarly to false positives.

C. Developer Repair

When a bug report is fixed, and it passes regression tests, we will be using Github to commit and push the updates to the testing team and leave a comment on facebook to ensure the developers see the changes.

VI. FUNCTIONAL TESTING PLAN

We will be testing our code by checking our results by eye. In most cases, we will be able to confirm if the area of interest is correctly identified. Furthermore, we will manually run the imellipse function in Matlab on an

image series and compare the results to the corresponding automated results from our code. By checking the coordinates and measurements of the result, we will be able to determine the accuracy of our solution.

Our client will also provide us with a series of simulated images for evaluating the accuracy of our code in the presence of extreme sooting. We will consider the results of both Matlab and OpenCV approaches when deciding whether we will continue with one or both solutions.