**Product Testing Report**

**Team Downey**

**By: Braden Priddy, Malichi Flemming,**

**Saif Wilkes-Davis, Tyler Owens**

**1. Introduction**

Product testing is critical to verify the system's functionality and assess its limitations. Data must be acquired through experimentation, processed for error mitigation, and analyzed for results. For this product, the following parameters were evaluated: consistency and speed of the operating system, camera’s field of view and placement, most efficient processing method, and accurate measurements. The most rigorously tested parameters were the consistency and speed of the operating system because they fulfill the most essential requirement for real-time image processing.

Since real-time systems are defined by hard deadline times, the system would fail if these deadlines weren’t met. Henceforth, most of the testing focused on optimizing the code and the system, as well as testing the limits of the code.

Despite the end product not being functional due to a hardware malfunction of the Speed Goat, the project was a success overall. The code for the system meets the required timing deadline and processes a SLM video on a Speed Goat Real-time Unit. To demonstrate the functionality of the code, a prototype was assembled using a test setup, laptop, and a USB camera. The prototype produced a processed image and a file containing splatter parameter data on a live feed at 30 fps. The last step to completing the product is finding and configuring a Speed Goat compatible camera.

**2. Test Plan**

Due to limited access to the SLM printer laboratory, the construction of a rudimentary model of the SLM printer was required to conduct adequate experimentation. The model was comprised of a cardboard box, the plastic plate where the camera housing is seated, and a Styrofoam ball to imitate the metal splatter. A square hole was cut out on the top of the box which allows the camera housing to fit securely with the plastic plate and for the camera to view inside the box. A circular hole was cut out at the opposite end of the box as an area for the splatter replica to be suspended from. The inside of the box was painted black, and the splatter replica was painted bright green and illuminated with a flashlight to mimic the light contrast.

Test #1

**Testing individual function processing times.**

Throughout the majority of the coding process, only one frame was being processed to see if it met the sponsors’ real-time deadline of 30 milliseconds. One way this can be analyzed is looking at individual functions within the code to see which has the greatest effect on processing time. This gives an idea of what to focus on when optimizing. This test was done multiple times throughout the design process. This test is our final function time analysis test of the completed code.

Chart, bar chart

Description automatically generated

Figure 1: Function time analysis of a single frame

Calculating the sputter area, radius, and centroid take up the greatest percentage of the processing time. These calculations are done using the MATLAB function “regionprops”, very little was done to optimize this as calculating the radius and centroid is difficult. The next most expensive function is the “variable and RGB image initialization”, this cannot be optimized further as the minimum number of variables is already used. The last function that takes a large percentage of the processing time is the “draw contours and calculate the brightness”. This can be optimized by no longer drawing the contour around the image, but this eliminates visual aspect of our display which tells the user that the sputter is being found by the code. The reason this function takes so long is that the code must go through the entire sputter pixel list to calculate the brightness.

Test #2

**Testing frames per second vs the size of melting sputter.**

The code must calculate the brightness and the contours of the main sputter. As the sputter gets bigger the more pixels, we must go through to calculate these parameters. For this test, 5 white squares of increasing size show how each affects the processing time. From this test, the limits of the code were visualized. The code must work at 30 fps and the processing time needs to work under 33 milliseconds.

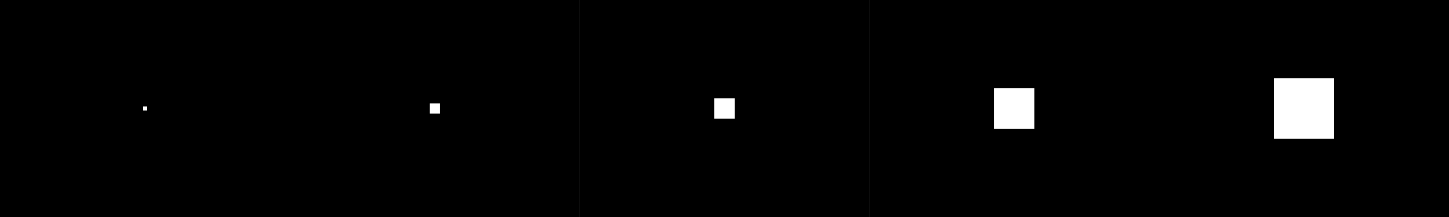


Figure 2: Five 1440x1080 images of squares

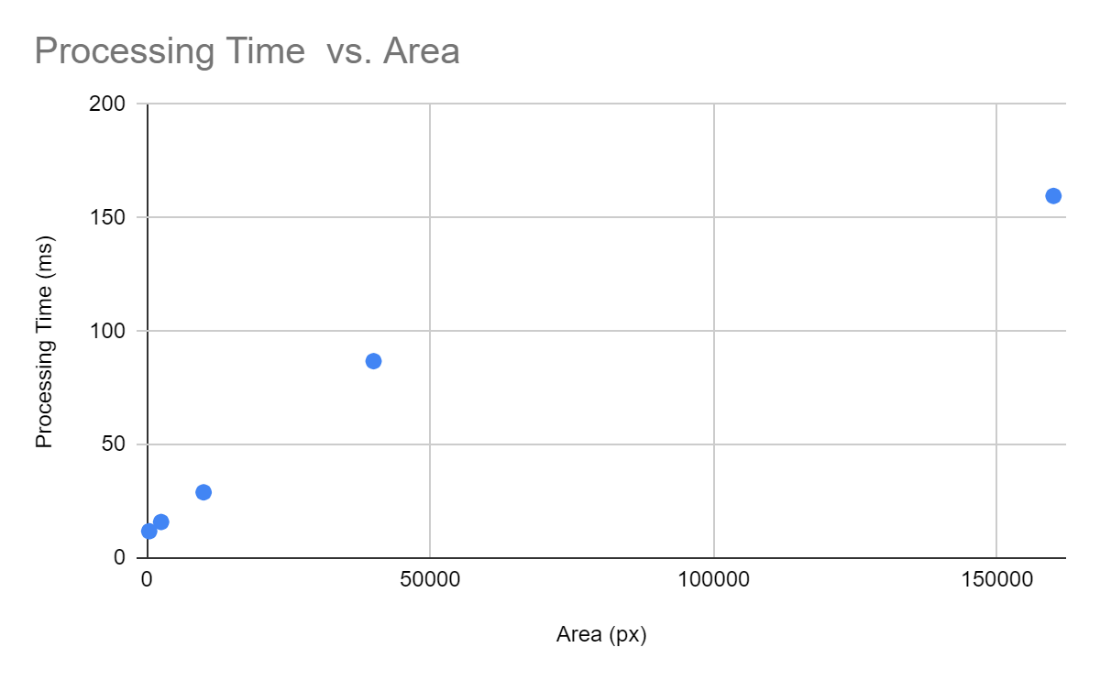


Figure 3: Processing time vs. Area

Table 1: Processing time vs. Area of squares

|  |  |  |
| --- | --- | --- |
|  | Area | Processing Time |
| Square 1 | 400 px | 11.81 ms |
| Square 2 | 2500 px | 15.88 ms |
| Square 3 | 10000 px | 28.97 ms |
| Square 4 | 40000 px | 86.78 ms |
| Square 5 | 90000 px | 159.61 ms |

Test #3

**Testing different image processing methods to optimize speed and the melting pool’s true area.**

There are plenty of different image filters provided by OpenCV and MATLAB. Different filters are tried and tested for the various arguments for each function. After the filters are applied, the area of the melting pool and the processing time are calculated. For this test, one base image was chosen as the independent variable to keep the results consistent. The true area of the base image is calculated and compared to the areas of the filtered images. Each test is run 5 times to acquire an average time.

A couple of important things to note about this test is that the threshold limit for each test is 215. Meaning that any pixel value less than 215 is turned black (0) and all other pixels become white (1). Thresholding is an essential filter to calculate the area and other parameters of the sputter. One other important note is that only the processing time of the filters is taken, not the time it takes to calculate the area of the sputter. This is done to isolate the impact the filters have on the processing time because if one calculated area is bigger than another area that would affect the processing time.

Filters used

* Gaussian Blur: reduces the image noise and the detail of an image.
* Dilate: adds pixels to the boundaries of a binary image.
* Erode: subtracts pixels from the boundaries of a binary image.
* Threshold: simplest way to create a binary image.

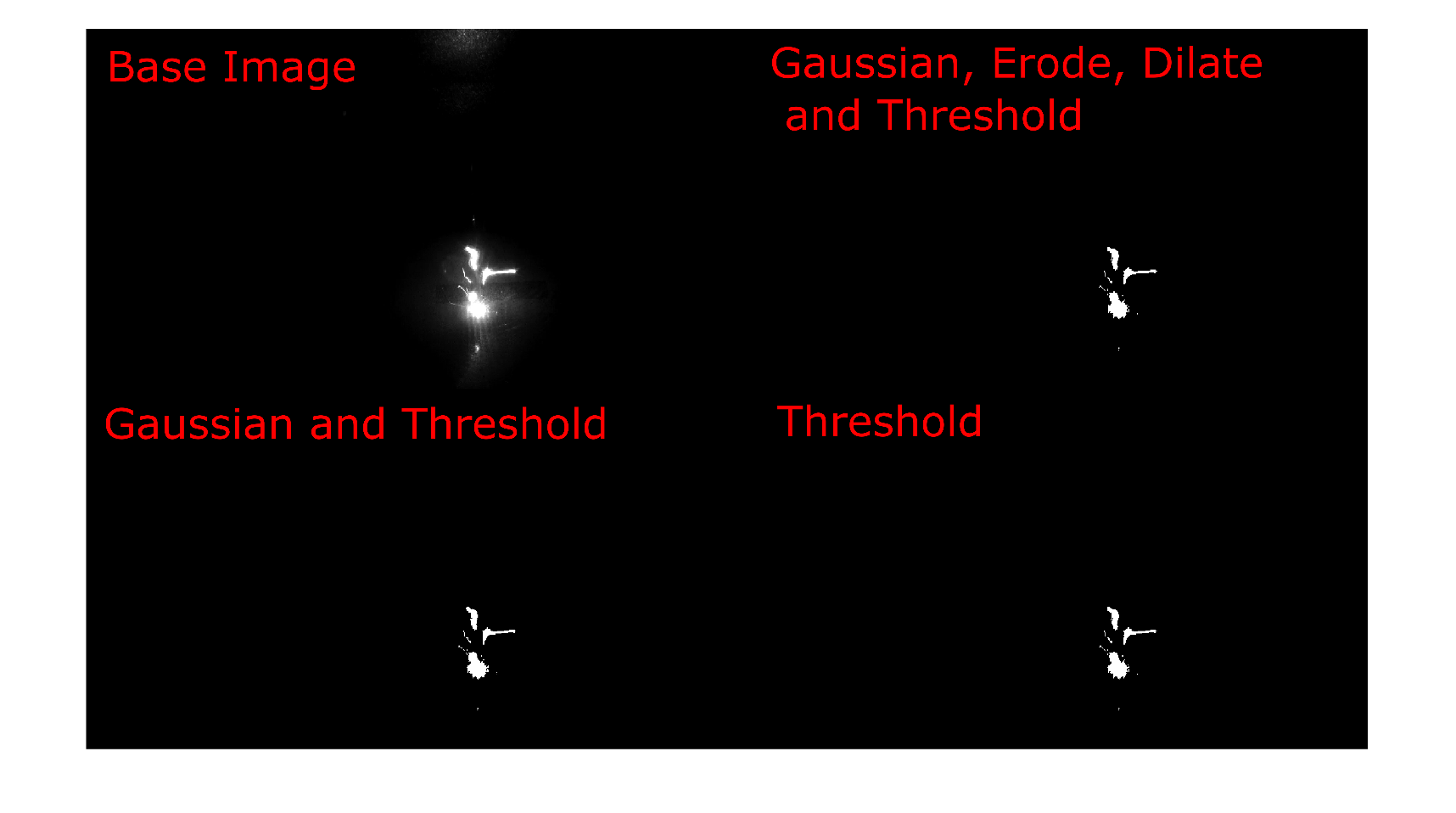


Figure 4: Montage of filtered images

Table 2: Filtered images processing time and area

|  |  |  |
| --- | --- | --- |
|  | Area | Processing Time |
| Base Image | 3237 px | 0 ms |
| Gaussian Blur, Threshold, Dilate, and Erode | 3236 px | 11.67 ms |
| Gaussian Blur and Threshold | 3239 px | 7.08 ms |
| Threshold | 3238 px | 2.22 ms |

It is seen from the results that filters have a negligible effect on the area of the sputter. There is no loss in any important information from using Gaussian Blur, Dilate, and Erode. For the final model, it was chosen to stay with thresholding.

Test #4

**Showing parameters vs not showing parameters**

The main focus of the project is meeting the real-time hard deadline required to process frames at 30 fps. If the system is unable to meet this deadline, then our system is useless to the sponsors. There are various ways in which the code can be optimized without affecting the functionality of the end product. One way in which this is done is by eliminating the visual aspects of the display. This means the code would still save the parameters of the main sputter (X/Y cords, Area, Radius, and Brightness) to a file, but it would not display these parameters in real-time. To test this, there will be three different versions of the code. One in which all the parameters and contours are displayed, a version where only the contoured image is displayed, and a final version where only the preprocessed image is displayed.

For this test the test model will be utilized to process a live video feed. This live video feed will be able to spit a file with the parameter, with a display window, and an extra Simulink block that will take the fps of the processed video feed. An average fps will be taken from the live feed and will be compared to each test.



Figure 5: Test mock setup



Figure 6: Live process on the test setup

The first image [Figure 6] is from the base video without any image processing and calculations of the parameters. Due to hardware limitations of the ThinkPad and the UVC camera only a max of 28.28 frames per second can be achieved. The second test returns a contoured live feed and a “.mat file” at the of the test. There is only a roughly 3 frame drop between the first and second tests. This is not the case when showing the parameters to the user. For the third test both the contour and parameters are shown to the user, but it greatly affects the fps of the live feed. The parameters are already saved to a “.mat file” at the end of the test, so it does not make sense to visually show it to the user if affects the processing time drastically. When the code is implemented onto the real-time unit it can run 60+ fps making it impossible for any user to read the parameters in the live feed.

Test #5

**Testing the accuracy of radius measurements to verify that the code’s outputs are valid.**

By printing out a copy of the splatter image to scale, the radius can be manually measured. A ruler will be used to make the measurement. The manually measured radius data should coincide with the data output from the system. This is needed to determine whether the code is working properly, and that the user will receive useful data. Five tests would have been performed to verify the accuracy of the measurements. Again, because of the hardware malfunction, this test could not be carried out.

Test #6

**Testing the portability of the system.**

Each component of the system will be weighed using a digital bathroom scale. Additionally, the operator of the system will document how each part is picked up and if they meet the requirements in the chart. If the object is less than 51 lbs. and all answers are yes/true, then the object qualifies as an OSHA safe lift. [Table 3] The OSHA requirements are met completely.

Table 3: OSHA test requirements

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Object | Weight (lbs) | Able to keep 7 inches away from front of body | No twisting involved in the lifting process | Object located at waist level and directly in front of the person | The object has a handle | The load inside does not shift once lifted. |
|
| ThinkPad laptop | 2.5 | yes | yes | yes | yes | yes |
| Speed Goat controller | 1.8 | yes | yes | yes | yes | yes |
| Camera housing | .1 | yes | yes | yes | yes | yes |
| Camera w/lens | 1.5 | yes | yes | yes | yes | yes |

Test #7

**Testing the consistency of the camera placement**

To check the constancy of the placement of the camera, fiduciary markers would be placed into the mock setup. The operator will rotate the camera until the markers are aligned with the expected orientation of the system and take a screenshot of the video feed. The operator will then remove the camera and redo the process. Software, WinMerge, will be used to compare the pixels of the two screenshots and consistency will be noted by counting the number of differences calculated by the software. The mock setup was not built with the exact dimensions of the SLM printer and therefore rendered this test invalid.

**3. Analysis**

During this testing process, there was no raw data evaluated. All test results are shown above in the previous section. Because of the malfunctioning hardware and limited access to the SLM printer, there were no measurements made.

**4. Conclusions/Recommendations**:

Five tests were performed on this system to check for processing time, frames per second, speed optimization, consistency of parameters, and OSHA compliancy. The results for these tests were satisfactory and met the target values. However, there were two tests that were not performed because of the hardware malfunction prior to beginning the tests and the lack of access to the SLM printer. These would have evaluated the accuracy of measurements being made by the system along with the consistency of the camera’s field of view. For future testing, having a fully functional control system and full access to the SLM printer is highly recommended.