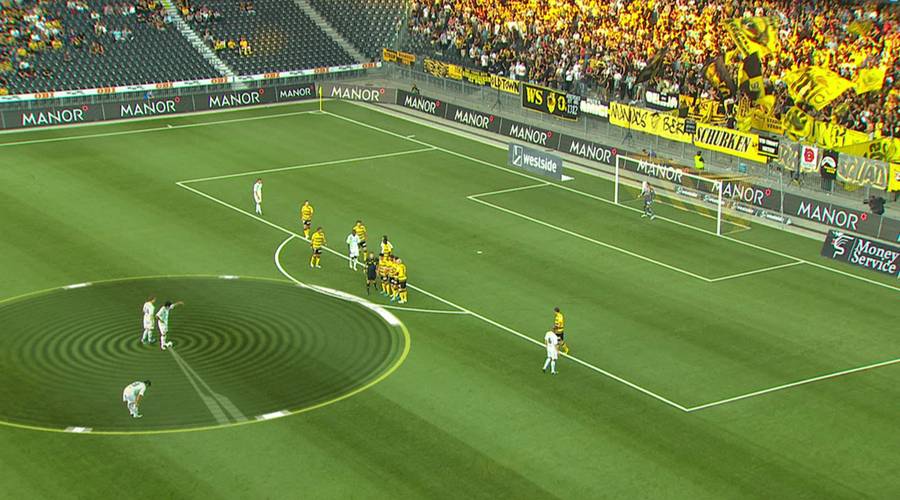
Alexander Buchholz

Intro to Image Processing

Object Detection in Live Video Processing   
Final Report

The goal of the project is to recognize the applications and challenges of live video processing, and to implement live video processing algorithms in MATLAB. As processing power increases, live video is becoming increasingly sophisticated in all manners of applications, such as surveillance, entertainment, and military. While many video processing algorithms are simple in nature, complicated tasks, like object detection, are becoming more common in today’s technology. Figure 1 shows such an example of live video processing in sports, where a player is detected and images are drawn onto the field in real-time.



**Figure 1- Adding Drawings to a Live Sports Video**

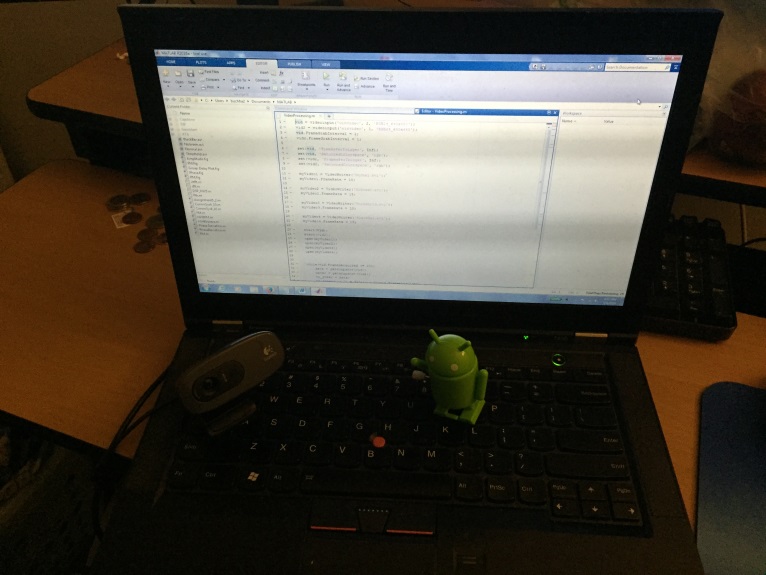
The topic of video processing is very closely related to almost every single topic that was taught in class. Many video processing algorithms, including the algorithms used here, treat a video as a series of images, and process videos one image at a time. Because of this, nearly all of the image processing techniques learned in class can be used for live video processing, so long as the processing time is small. The object detection used in this project is closely related to thresholding and morphological operations. Other object detection applications may include template matching, feature detection, and computer vision algorithms.

This project aims to detect an Android wind-up toy, shown in Figure 2. The toy is a great object for object detection, due to its solid bright green color. As long as the color is unique, and no similar colors exist within the recording environment, color thresholding can be used for image segmentation and object detection.



**Figure 2- Android Wind-up Toy**

Data was collected using a Logitech c270, which supports HD 720p video, 1280 x 720 pixel resolution, and 3.0 megapixel photos. A Lenovo T430 Thinkpad was used to run all MATLAB code. The built in webcam was used as additional video input for screen transitions, but the code contained unresolved bugs, and is not detailed in this report. Figure 3 shows the recording environment and equipment used.



**Figure 3- Recording Setup**

The c270 webcam was pointed at the laptop keyboard, where videos of an Android wind-up toy were recorded and edited in real time. The videos cannot be shown in a word document, but several frames of the raw video are shown in Figures 4, 5 and 6.



**Figures 4, 5, and 6 – Video Screenshots Without Additional Processing**

The project’s initial efforts were focused on collecting video input in a way that individual frames can be processed in real time. To do this, MATLAB’s Image Acquisition Toolbox was used. Videos were set to be recorded with the following specifications [1]:

* 640x480 resolution with 24 bit RGB colors
* 24 bit RGB colors
* Trigger at maximum possible rate
* Collect 1 frame upon every trigger

MATLAB’s VideoWriter function was used to save the videos in ‘avi’ format for documentation purposes. These videos were saved at 15 frames per second using Motion JPEG codec for compression.

The basic process is shown in the flowchart detailed in Figure 7.

Initialize Cameras and Define File Information

Yes

Record 1 Frame, Threshold Green and Red

Apply Median Filter to Green and Red, then Combine

Erode, then Dilate Segmented Areas to Remove Remaining Noise

Count Segmented Areas and Draw Bounding Rectangles

Have 200 Frames Been Collected?

No

Stop Recording, Close Files, and Flush Data

Write Data to Video Files

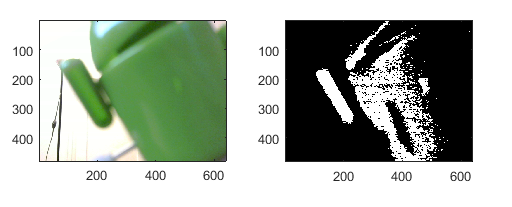
**Figure 7- Flowchart of Algorithm**

Many videos were saved while recording, each detailing important steps in the algorithm’s development. First, global thresholding at 128 out of 255 was tested as a basis for optimizing basic video capturing. This was to ensure that any frame-lag was a result of the object detection algorithm, and not unrelated parts of the code. A global threshold also shows the brightest objects in a room, which was helpful to keep in mind when testing object detection. Images from this thresholding are shown in Figures 8 and 9.

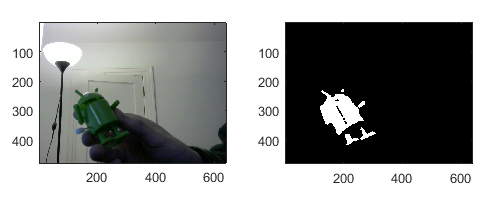


**Figure 8 – Thresholding Live Video Figure 9 - Additional Threshold Image**

Figures 8 and 9 show that the toy robot, when represented as a grayscale image, is darker than the rest of the room. However, other parts of the room were also dark─ it was clear that global thresholding was not a viable approach to detect the robot. More intelligent thresholding was required, so the green color was used for segmentation. To accomplish this, a grayscale image was subtracted from the green components of the image; this resulted in an image containing mostly zeroes, and small integers in locations with higher than average green pixels. The result was thresholded at various values and qualitatively compared, shown in Figures 10, 11, 12 and 13.



**Figures 10 and 11- Poor Green Segmentation at 0.10**



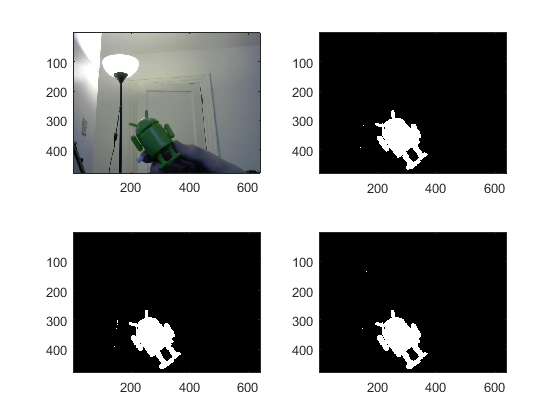
**Figures 12 and 13- Better Green Segmentation at 0.04**

While the best results, shown in Figure 13, were acceptable, segmentation was improved by analyzing RGB values on the robot. Table 1 shows a list of RGB values, taken at various locations of the toy using the impixel command.

**Table 1- Sample Pixels from Toy Robot in Figure 2**

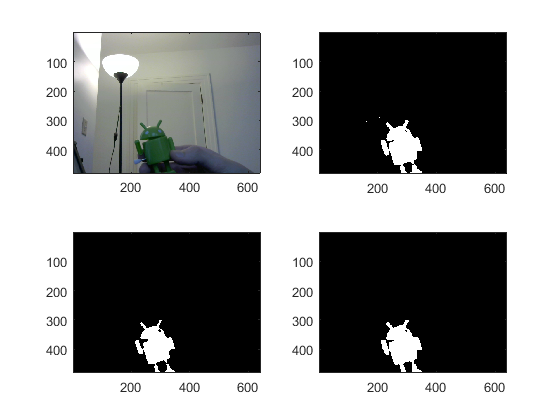
|  |  |  |  |
| --- | --- | --- | --- |
| **Pixel** | **Red** | **Green** | **Blue** |
| **1** | 53 | 60 | 16 |
| **2** | 108 | 123 | 20 |
| **3** | 109 | 124 | 23 |
| **4** | 54 | 61 | 17 |
| **5** | 136 | 155 | 30 |
| **6** | 223 | 215 | 72 |

Table 1 shows that the robot, whether in bright or dark environments, contains a large amount of red in addition to green. By exploiting this, segmentation was improved by lowering the threshold even further, applying a [3 3] median filter, then combining the red and green segments together. Figures 14, 15, 16 and 17 show the results.

****

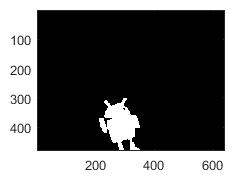
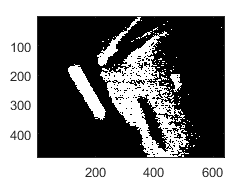
**Figure 14, 15, 16 and 17- Red (Top-Right) and Green (Bottom-Left) Used for Segmentation at 0.02 (Bottom-Right)**

Although the segmentation was improved, a few unwanted pixels were introduced. These pixels passed the threshold test, despite being separate from the toy robot. Since the pixels were small and few, a morphological structuring element was used to remove the pixels. The segments were eroded by a disc of size 5, and then dilated by 5 pixels. This removed the unwanted pixels at the cost of some minor segment distortion, detailed in Figures 18, 19, 20 and 21.

****

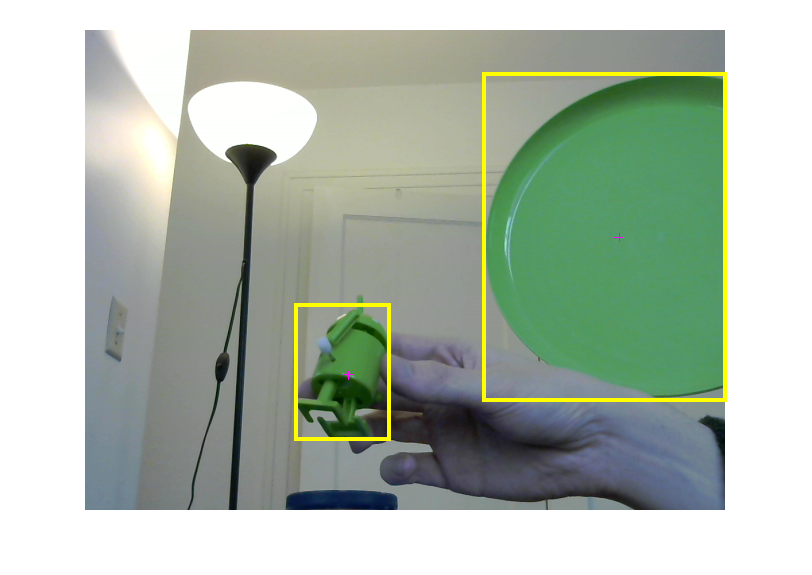
**Figures 18, 19, 20 and 21- Black/White Image (Top-Right) Eroded (Bottom-Left) then Dilated (Bottom-Right)**

The final segment in Figure 21 shows a very clear image of the robot. Compared to the original segment, repeated in Figure 22, and final segment shown in Figure 23 is much clearer, and without significant noise.



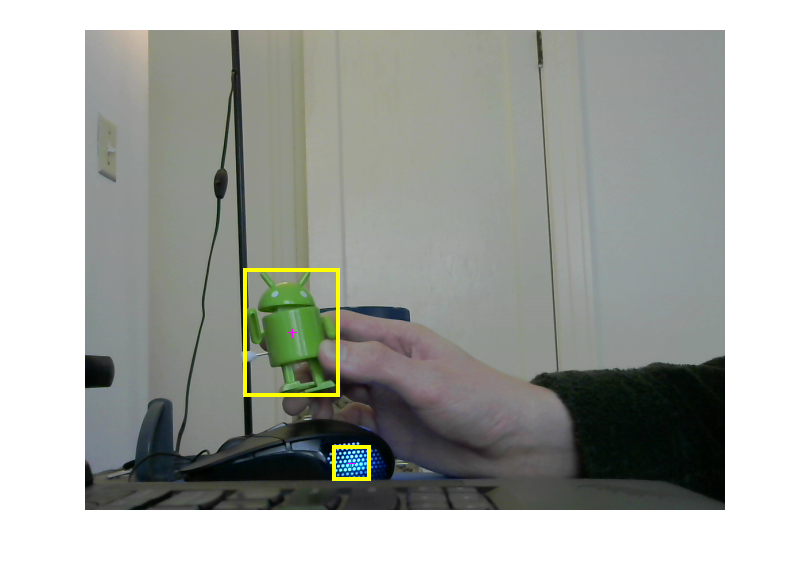
**Figures 22 and 23- Comparison of Initial and Final Segmentation**

From this point, the algorithm labels every unconnected segment on each frame, and applies a bounding box to each. Unfortunately, since these bounding boxes are drawn on top of the data, and are not part of the data themselves, they do not appear in the videos. However, an image of successful object detection is shown in Figure 24. The robot, and anything colored similarly to the robot, are picked up by the algorithm and labeled.

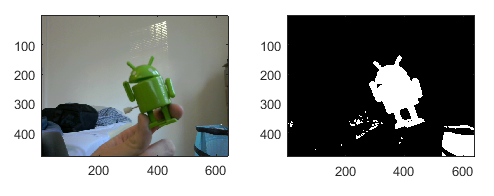
****

**Figure 24- Successful Object Detection on Bright Green Objects**

While the algorithm works has zero errors in some environments, like the environment shown in Figure 24, significant flaws were noticed when the robot was placed in different environments. Figures 25, 26 and 27 detail some errors in other environments.

****

**Figure 25- Unexpected Error Caused by Bright Lights**



**Figure 26 and 27- Segmentation Errors in Colorful Environments**

These errors occur because the algorithm relies on comparing Red and Green values on an RGB scale. Many objects have Red and Green values compared to the toy robot, but do not appear that way to the human eye. This is because the RGB scheme is not the best representation for how the human eye perceives color. Table 2 shows some pixel values collected from the false object in Figure 25.

**Table 2- Pixels Collected from Light in Figure 25**

|  |  |  |  |
| --- | --- | --- | --- |
| **Pixel** | **Red** | **Green** | **Blue** |
| **1** | 152 | 255 | 255 |
| **2** | 131 | 255 | 255 |
| **3** | 141 | 255 | 255 |

Table 2 shows that, if the pixels are bright enough, they might get picked up by the thresholding algorithm incorrectly.

This algorithm could be improved by changing the way that thresholding occurs. First, instead of comparing RGB values, colors should be compared on a L\*a\*b\* colorspace. In L\*a\*b\* space, L\* represents the “Lightness” of an image, while a\* and b\* are color coordinates. From here, multiple approaches can be taken. One approach is to create a color distance function:

With a distance function, the values *RobotColor* and *Threshold* can be chosen so that *d(x,y)*  always lands within the colors listed in Table 1. This was attempted briefly in RGB colorspace, but processing time increased to unacceptable levels due to the immense number of operations required.

A more realistic approach is to use Otsu’s method for color segmentation in L\*a\*b\* space. A research team at Bharathiar University, India, showed that Otsu’s method can be used in L\*a\*b\* space with low computational cost, and at speeds feasible for real-time image processing [2].

Object Detection could be improved even further by applying other detection algorithms learned in class, besides color thresholding. The robot has a unique shape, so template matching seems like a reliable option for object detection. Some of the robot’s features are unique too, such as the “antennae” on its head, so feature detection seems like a reliable option too. While both of these would reduce any errors in the algorithm, they would also increase processing time─ further testing is needed to see if these methods are viable detection options that do not ruin the algorithm’s processing time.

Videos of the algorithm in action are listed on the GoogleDrive, emailed to you. Other machines might not be able to run the MATLAB code easily, due to hardware/software differences.

# References

|  |  |
| --- | --- |
| [1] | "HD WEBCAM C270," Logitech, [Online]. Available: http://www.logitech.com/en-us/product/hd-webcam-c270. [Accessed 9 May 2016]. |
| [2] | D. Napoleon, A. Shameena and R. Santhoshi, "Color Segmentation Using Otsu Method and Color Space," in *International Conference on Innovation in Communication, Information and Computing*, Coimbatore, 2013. |