UNIVERSITY of WASHINGTON

ADAS Vision Tool Check
11/17/15

Table of Contents

List of Figures

A - Activity 1: Color Space Conversions

A.1 - Program Description

This program along with all other activities, processes video frame-by-frame. For both parts of Activity 1, the color space is converted using the built in Matlab function (`rgb2hsv` or `rgb2ycbcr`) then a specific component is weighed as described in **Computer Vision Function Descriptions**. Next, the inverse operation (`hsv2rgb` or `ycbr2rgb`) converts the altered image back to the RGB space. To draw the colored rectangle and text, the `insertText` and `shapeInserter` functions are used, and the locations are defined through simple geometric relationships. The `VideoReader` and `VideoWriter` objects are utilized to read and write the video file, and the `imshow` function used in the while loop displays the frames sequentially in real-time code execution. This general structure for processing video is used in all activities. The steps are summarized in Figure 1 below.

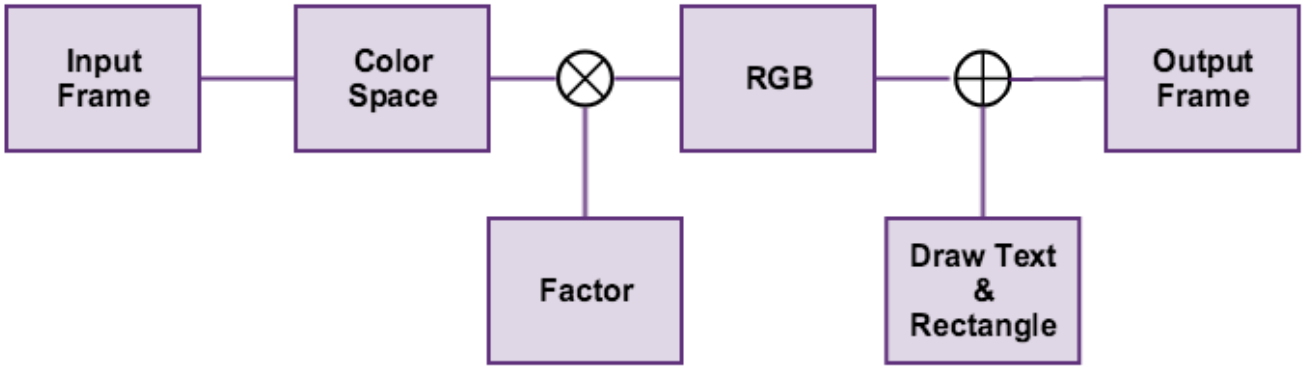

FIGURE 1: ACTIVITY 1 COLOR SPACE TRANSFORMATION DIAGRAM

A.2 - Computer Vision Function Descriptions

Both the YCbCr and HSV models follow the same high-level structure. The input, RGB frame is converted to either the HSV or YCbCr representations, which like RGB is comprised of an [M x N x 3] dimensioned array. The component to be modified (either hue or luma) is extracted resulting in an [M x N] array of pixel data for the given component. Once extracted, the given weight can be applied to the component by simple multiplication. For 20% increase in Y (luma) a factor of 1.2 is used and for 20% decrease in H (hue) the factor 0.8. After this modification, the frames are converted back to RGB, so the text and rectangle can be correctly integrated.

Converting the color space can be useful in computer vison when analysis beyond the RGB color space is required. For example, changing the brightness and contrast levels of an input for thresholding, or increasing an element such as hue to more obviously expose certain color differences.

A.3 - Output Video Results

FIGURE 2: ACTIVITY 1A, SHOWS FRAME AT 2 SECONDS WITH 20% INCREASE IN THE Y COMPONENT IN THE YCBCR COLOR SPACE

FIGURE 3: ACTIVITY 1B, SHOWS FRAME AT 2 SECONDS WITH 20% DECREASE IN THE H COMPONENT IN THE HSV COLOR SPACE

B - Activity 2: Binary Video Vision Processing

B.1 - CV Function Descriptions

The general approach for this activity is first to identify blobs, and second draw a rectangle around blobs meeting certain criteria relating to the dimensions. The flow of this program is shown in Figure 4 below. The input frame is converted to binary format using `im2bw`, and the binary frame is further processed through a series of transforms that attempt to output a white blob around objects that meet the defined criteria. First, a pixel area-based filter is applied using the `bwareaopen` command. This command takes a minimum pixel area input and any cluster of white pixels smaller than this area is mapped to black. The process effectively gets rid of the noisy small white specs. Though an averaging filter could achieve similar results, the area based noise suppression allows for more situational flexibility. A max area could be specified as well, but is unnecessary in this case. With the background completely black, some blobs remain but may not be continuous or connected. A morphological close function, `imclose`, combines erosion and dilation to smooth out pixel differences using a structuring element as an input. A disk of an optimized area is specified as a structuring element and effectively fills in any black holes or discontinuities in the remaining white blobs. The result is a countable quantity of blobs that generally keep the same size through the frames.

To draw rectangles on the remaining blobs, `regionprops` is used on the result of `imclose` with the 'Bounding Box' argument specified. This returns measurements of the best fit dimensions for bounding boxes for each blob. These coordinates locate the text and rectangle objects displayed on the frame. The number of properties returned correspond to the number of boxes to draw (quantity shown in lower left), the coordinates for each box are used to locate the box as well as calculate the information displayed as text below the box. Though the instructions specify for this text to be in the box, it is placed directly below for readability reasons. Basic filtering on the bounding boxes is applied to effectively ignore boxes outside a defined aspect ratio and area. This ultimately reduces false positives in detection of objects of interest. The steps above that create blobs have some limitations with things like fences or large objects on the side of the road. Many of these are filtered out as the area or aspect ratio is not within the expected range.

Finally, using a bitwise OR operation, the binary input is combined with the blob output of the `imclose` step in order to superimpose the blobs on the original frame for clarity. This is converted to RGB and then the text and rectangles objects are drawn on the frame using the properties outlined in the previous paragraph. The resulting frame is RGB with blue rectangles around the accepted blobs, text describing each rectangle, and a counter tracking the number of rectangles per frame.

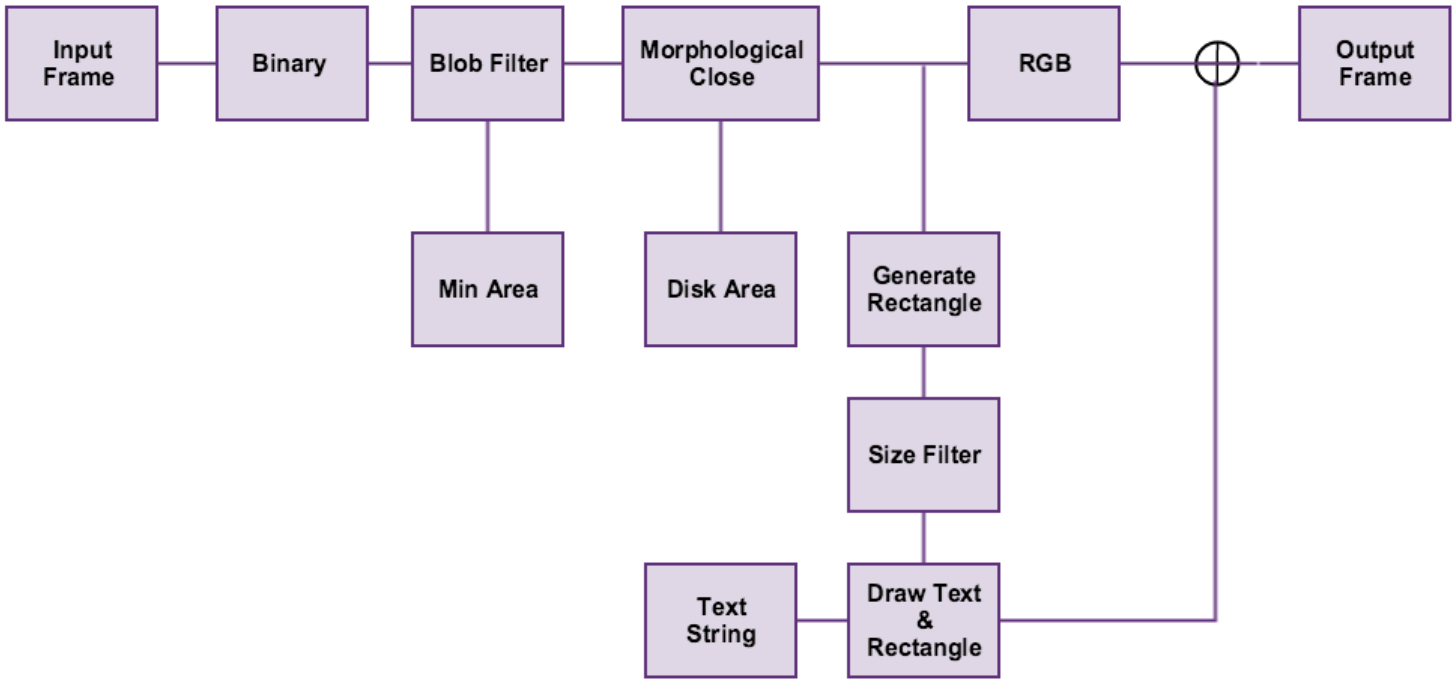

FIGURE 4: ACTIVITY 2 BINARY PROCESSING VIDEO

B.2 - Output Video Results

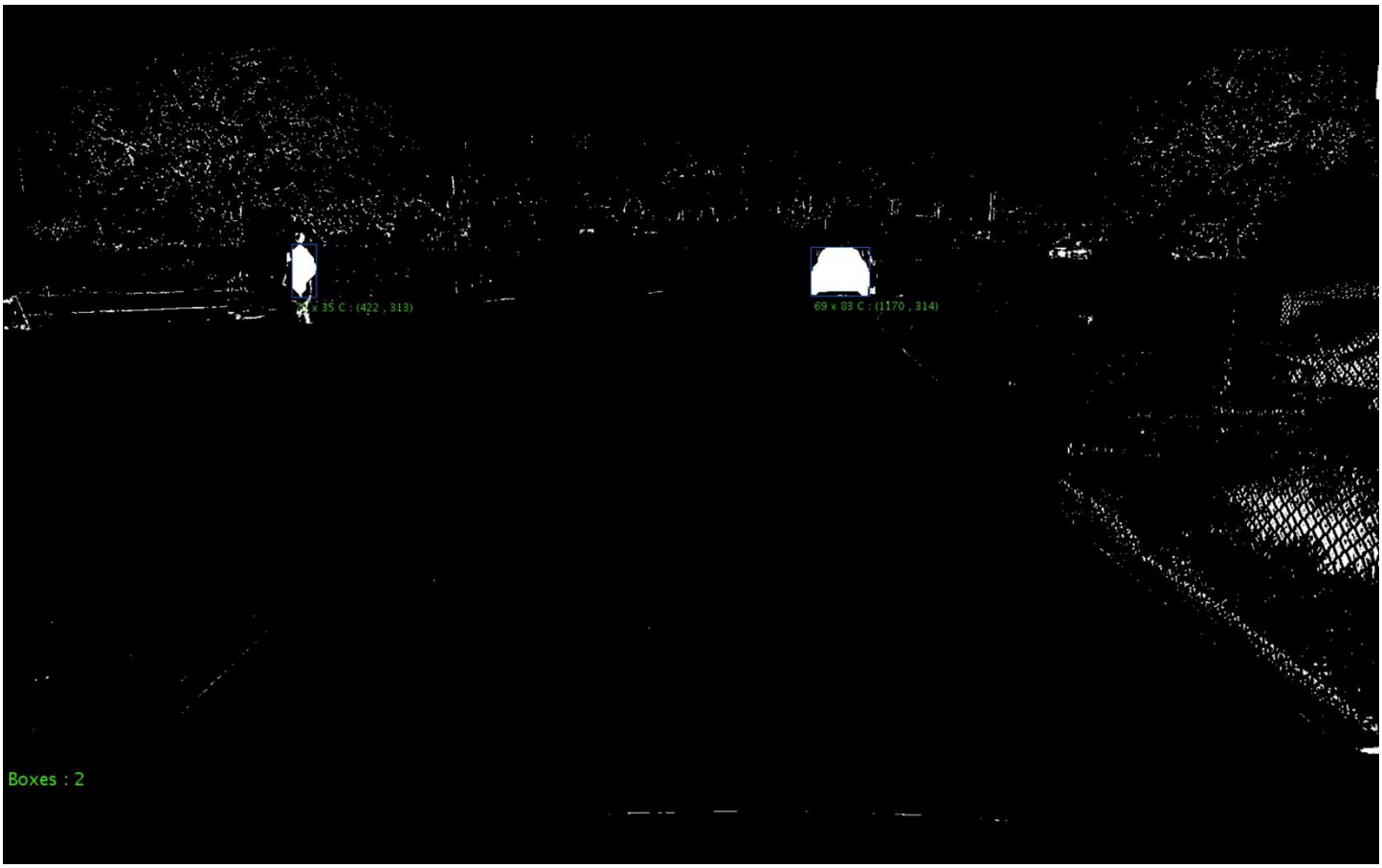

FIGURE 5: ACTIVITY 2 IMAGE SEGMANTATION IDENTIFYING PEDESTRIAN AND VEHICLE AS BLOBS AT 2 SECONDS

C - Activity 3: Thresholding Fundamentals

C.1 - CV Function Descriptions

In order to effectively segment out the lanes, region of interest (ROI) based processing is used to focus the majority of processing on a predefined region of pixels. In practice, this ROI likely needs to be adaptive or tuned, specifically on a sharp turn or incline, but in the input video, the camera is fixed and the road region stays fairly constant relative to the vehicle as well. Thus, the area outside the ROI can be ignored.

A simple pixel mask is generated with the ROI colored white and everything else black. The ROI coordinates are generated to find a box that best fits the road region. To process the frames, the input is converted to HSV using `rgb2hsv` and the V component representing the lightness is isolated and converted to binary using `im2bw`. A threshold argument is applied in the binary conversion to set pixels greater than the threshold to white and everything else black.

To accommodate for increasing brightness of the video, feedback is used to scale the threshold for passing high-contrast colors out of the ROI. The threshold is lower-bounded and decreased as the video brightens, which is a situational solution.

The last step is combining the mask created based off the ROI with the binary lane detection frame using bitwise ADD. The resulting combination only contains white pixels where both the mask and the video frame have white pixels and is black elsewhere. Thus the output frame is only white where the lanes within the ROI exist. This result is converted to RGB and exported as a final frame. The process is outlined in Figure 6 below.

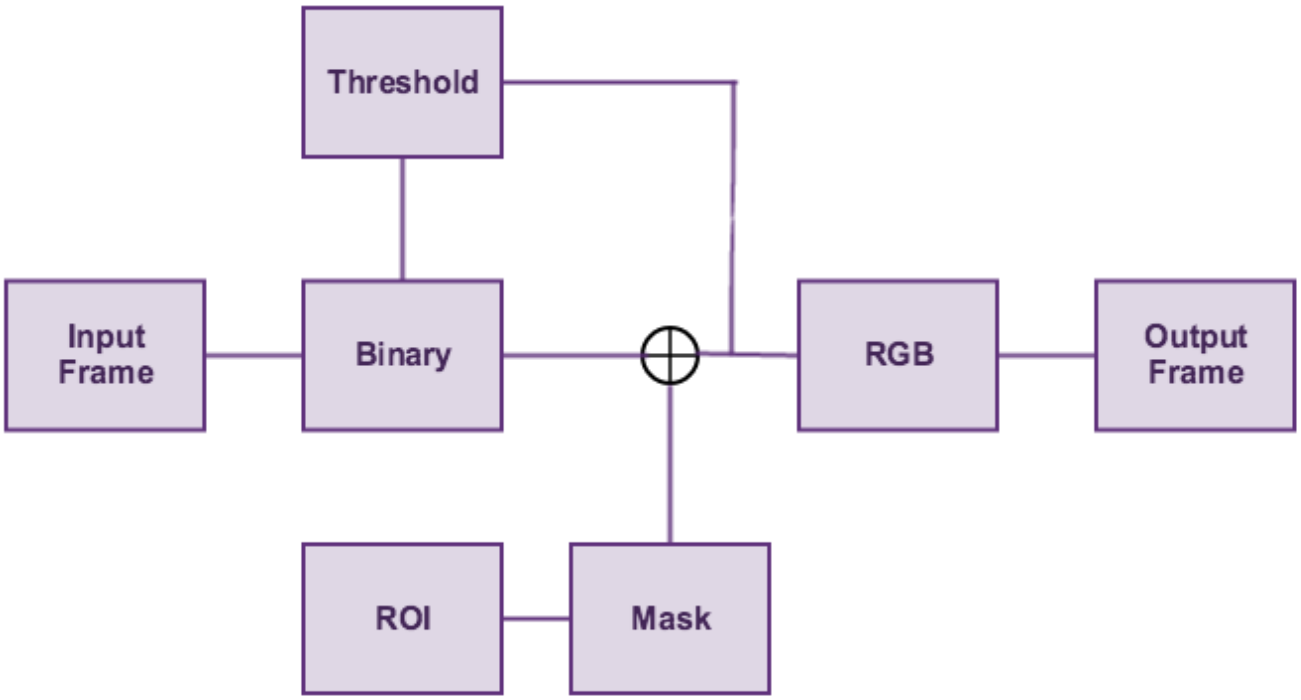

FIGURE 6: ACTIVITY 3 LANE SEGMENTATION VIA BINARY THRESHOLDING

C.2 - Output Video Results

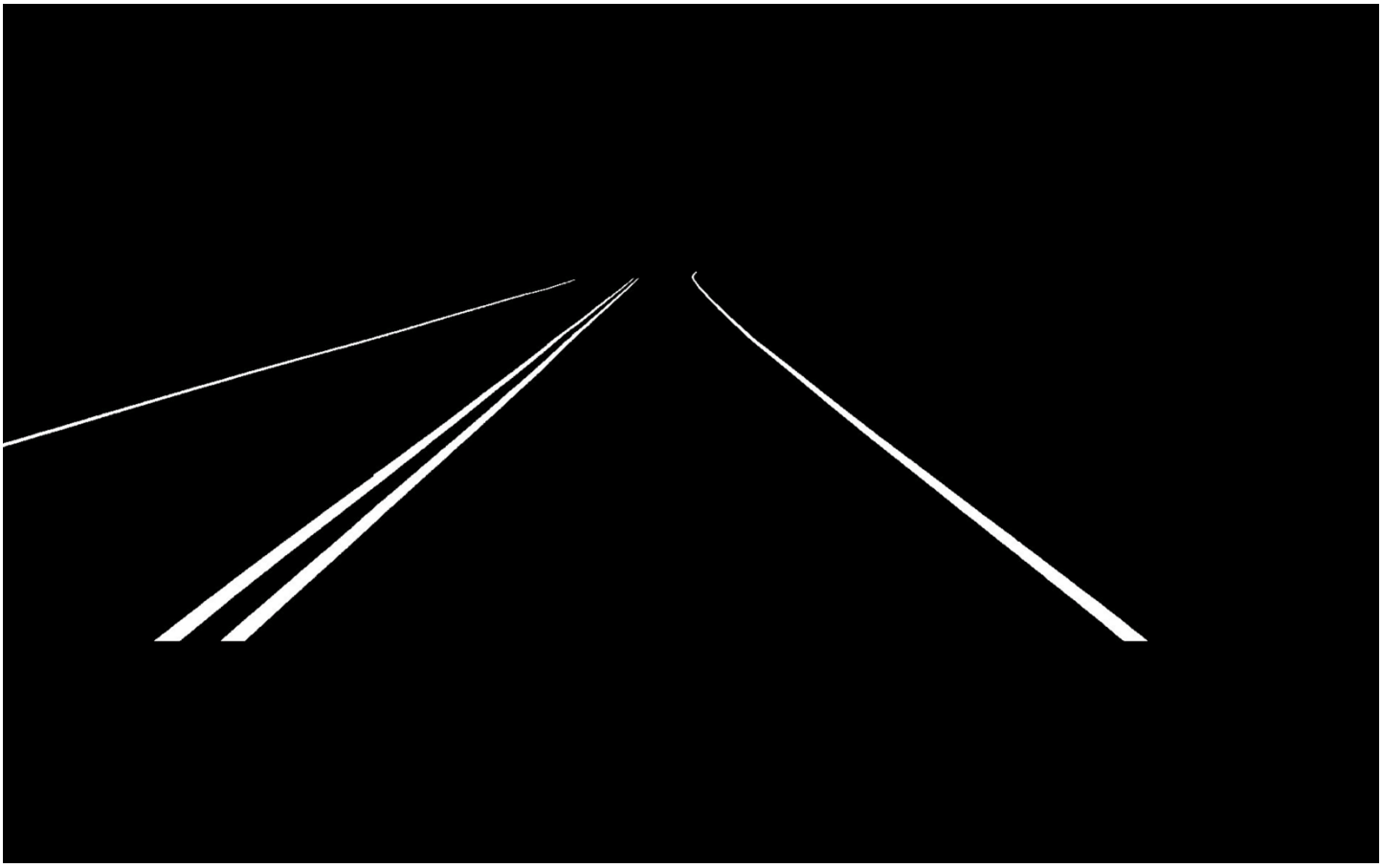

FIGURE 7: ACTIVITY 3 ROI BINARY THRESHHOLD LANE SEGMENTATION AT 2 SECONDS