Demo: Real-time Aggie Color Correction

Aneesh Rai and Sophie Wang, TAMU
Course Instructor: Professor Deepa Kundur

Introduction

In this lab, we will celebrate school rivalries with a "Color Correction" scheme by converting burnt orange into maroon. First, we will take a look at color spaces and how color works in video:

The color that we see on a screen can be broken down into different components. For example, we can specify the red, green, or blue components. The color can also be represented in terms of its hue, saturation, and brightness. These different components can me mapped on to a graphical system. For example, hue, saturation, and brightness are three different components of a color, so they can be mapped onto a 3-dimensional space. The RGB color space can also be visualized in three dimensions. Below are visualizations of the RGB and HSV color spaces:

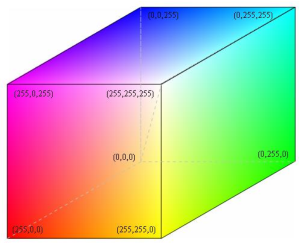


Figure i-1: RGB Color Space – from UC Berkeley

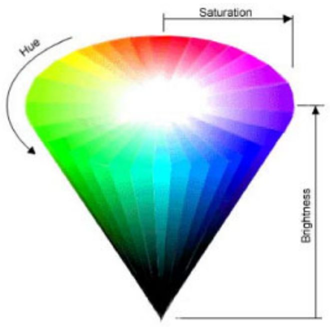


Figure i-2: HSV (Hue, Saturation, and Brightness) Color Space – from Color Tutorial

The output of our camera will be the YCbCr color space. YCbCr has been used to represent images because it produces a high quality image using less data. This is because most of the "information" that humans see in a picture is seen in its brightness. The color information can be at a lower information degrading the quality of the image. This is the idea behind YCbCr – Y represents the brightness of an image while Cb and Cr contain the color information. Since Cb and Cr can be at a lower resolution than Y, they can be sampled at half the rate. This is known as YCbCr 4:2:2 format.

But based on our common RGB understanding of colors, we have to wonder: Where is the green information encoded?! In turns out that green has a strong correlation to brightness – in fact, 60-70% of the brightness we see is due to brightness of the green component of a color. Although this is a slight simplification, since Y stores the brightness information, we can also think of Y as specifying the green component of a color. Below is a visualization YCbCr color space:

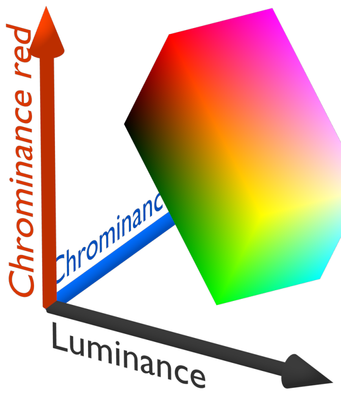


Figure i-3: YCbCr (Luminance = Y, Chrominance blue = Cb, Chrominance red = Cr) Color Space - from Wikipedia

Notice in the above image, there are some components that are in the bounds of the individual Y,Cb, and Cr values that don't correspond to any colors (corresponds to the white space in the visual). This makes the YCbCr color space difficult to visualize. Most of us find it easier to manipulate colors in terms of their RGB components. That's why several office software products (document editing, slideshow creators) allow users to select colors in terms of their RGB values.

Design and Implementation

In this lab we, will convert Y,Cb, & Cr values to RGB values. Below is a visualization of the relationship between the RGB to YCbCr color spaces.

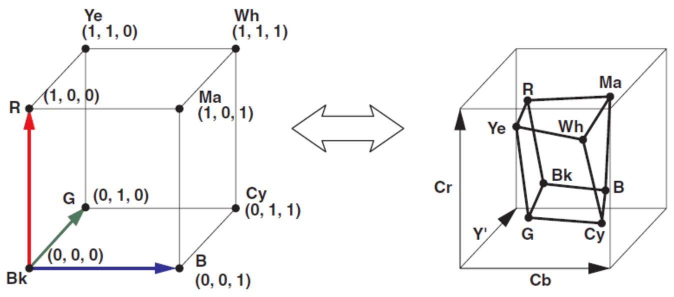


Figure i-4: Relationship between RGB and YCbCr color spaces - from Xilinx

But before we can convert YCbCr to RGB, we have to manipulate the data slightly. Since RGB requires an equal amount of data for each of the three components and YCbCr less data for Cb and Cr, we will use matrix concatenation to make the data matrices the right size for processing. After that, we will convert YCbCr to RGB and then manipulate the RGB values with our own MATLAB function. This function defines a range for orange, compares each pixel in the image to this predefined range, and then converts the pixel to maroon if it is orange.

The parameters that we use to define orange in the matlab function were determined from experimentation.

Buildling the Simulink Model

Picture:

1. Find any image with shades of orange and save it as a JPEG file.
2. To use the image for the software, use the Image to File block. Video and image Processing Blockset → Sources → Image From File. Use the "Browse…" button to find correct JPEG file and set the Sample time to inf. The Image signal should be changed to "Separate color signals" and Output port labels as "R|G|B." It should look like the following:

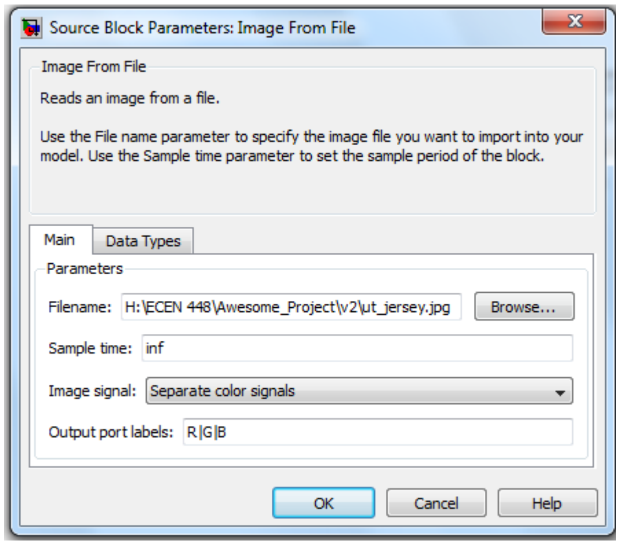

Source Block Parameters: Image From File

Image From File

Reads an image from a file.

Use the File name parameter to specify the image file you want to import into your model. Use the Sample time parameter to set the sample period of the block.

Main | Data Types

Parameters

Filename: H:\ECEN 448\Awesome_Project\v2\ut_jersey.jpg

Sample time: inf

Image signal: Separate color signals

Output port labels: R|G|B

Figure 1: Image to File Parameters

3. The next block to add is the Embedded MATLAB Function block. Add the following code to it:

```
function [red_out,green_out,blue_out]= fcn(red_in,green_in,blue_in)
red_out = red_in;
green_out = green_in;
blue_out = blue_in;
[m,n] = size(red_in);
for i = 1:m
    for j = 1:n
        if red_out(i,j) < 255
            if (green_out(i,j) < red_out(i,j)*0.75)
                red_out(i,j) = 128;
                green_out(i,j) = 0.25*green_out(i,j);
            end
        end
    end
end
```

After the code is put into the function, the block should allow for three inputs and three outputs.

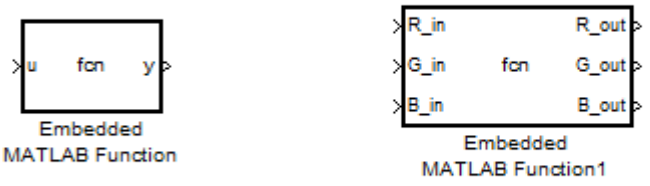
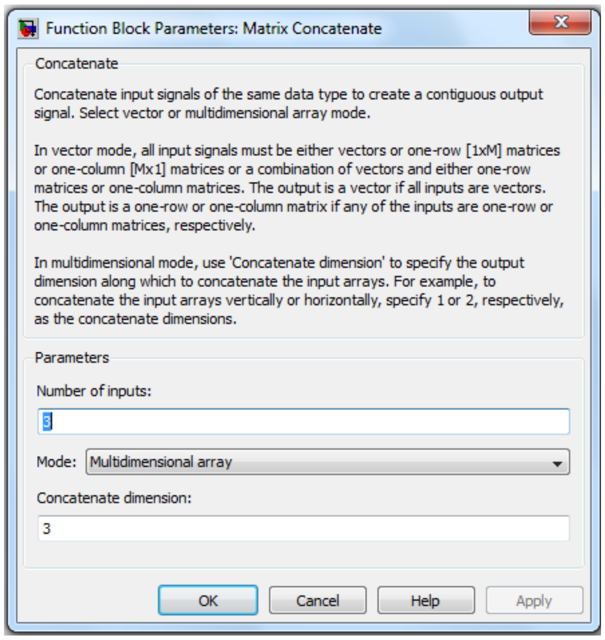


Figure 2: Embedded MATLAB Function block before and after the entry of code, respectively

4. Find the Matrix Concatenate in the Simulink Library set the number of inputs to 3 and check if the parameters look like the screenshot on the following page:


Function Block Parameters: Matrix Concatenate

Concatenate

Concatenate input signals of the same data type to create a contiguous output signal. Select vector or multidimensional array mode.

In vector mode, all input signals must be either vectors or one-row [1xM] matrices or one-column [Mx1] matrices or a combination of vectors and either one-row matrices or one-column matrices. The output is a vector if all inputs are vectors. The output is a one-row or one-column matrix if any of the inputs are one-row or one-column matrices, respectively.

In multidimensional mode, use 'Concatenate dimension' to specify the output dimension along which to concatenate the input arrays. For example, to concatenate the input arrays vertically or horizontally, specify 1 or 2, respectively, as the concatenate dimensions.

Parameters

Number of inputs: 3

Mode: Multidimensional array

Concatenate dimension: 3

Figure 3: Matrix Concatenate Parameters

5. The last block to get is the Video Viewer and connect all the blocks like below:

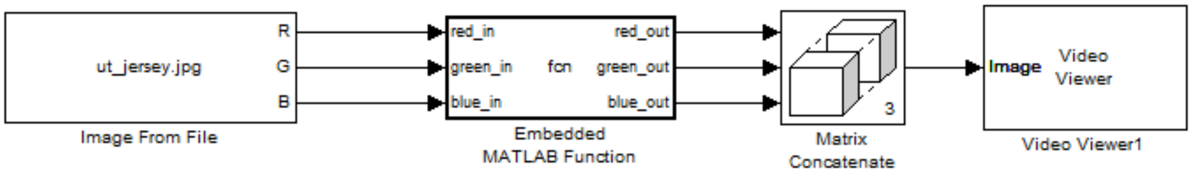


Figure 4: Simulink Model for Picture to be implemented

6. Now add another Matrix Concatenate and Video Viewer to the Image From File to output the original file. The model should look like the following:

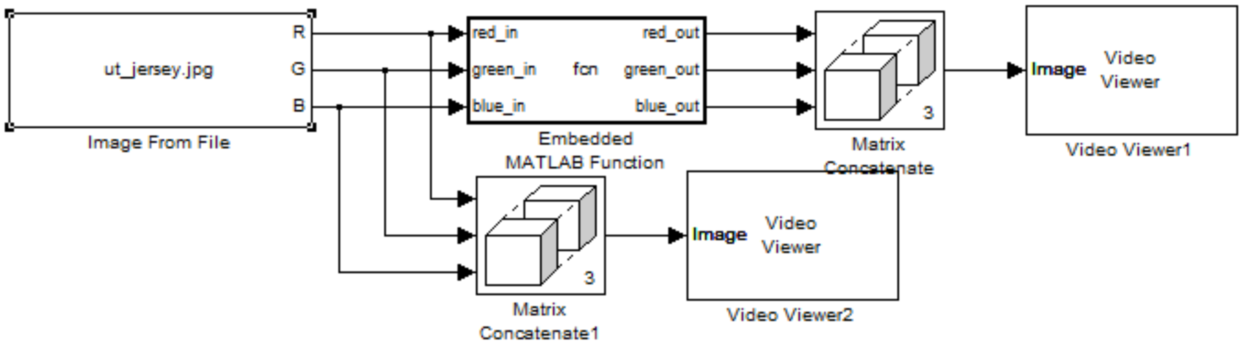


Figure 5: Final Simulink Model with two outputs

7. Run the model and see how your original image has changed.

Video:

8. Take the same software model for picture and replace the Image from File with From Multimedia File (Video and image Processing Blockset).
9. Click on the Embedded MATLAB Function and change the code to the following:

```
function [R_out,G_out,B_out]= fcn(R_in,G_in,B_in)

[m,n] = size(R_in);

R_out = R_in;
G_out = G_in;
B_out = B_in;

for i = 1:m
    for j = 1:n
        if (R_out(i,j) > 0.1) && (R_out(i,j) < 1)
            if (G_out(i,j) > 0.3) && (G_out(i,j) < 0.9)
                if (B_out(i,j) > 0) && (B_out(i,j) < 0.3)
                    R_out(i,j) = 0.3;
                    G_out(i,j) = 0;
                    B_out(i,j) = 0;
                end
            end
        end
    end
end
```

10. Add another Matrix Concatenate and Video Viewer set into the RGB output from the From Multimedia File. Make sure the model looks like the following:

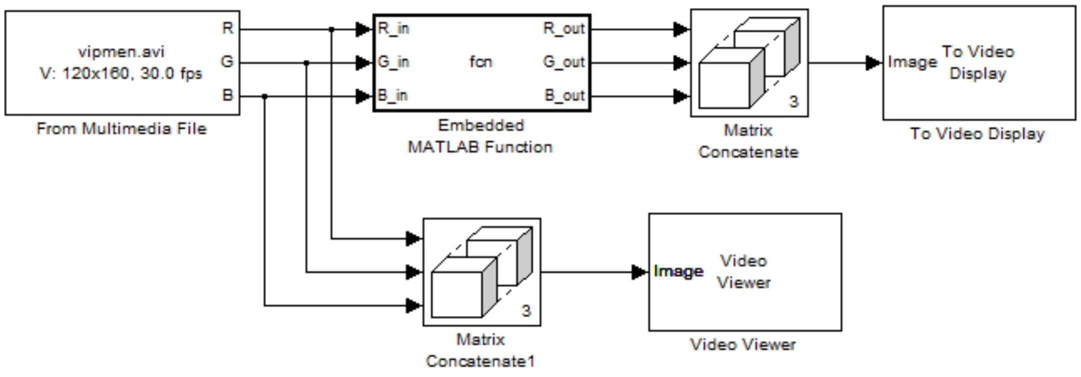


Figure 6: Simulink Model for Video to be implemented

Digital Video Hardware Implementation

1. First find the Video Capture block in Simulink: Target Support Package → Supported Processors → Board Support → DM6437 EVM → Video Capture
2. In the same DM6437 EVM folder, also get the Deinterleave block, Interleave block, and Video Display block for the model.
3. Connect the Deinterleave block after the Video Capture block and two sets of the Interleave and Video Display blocks together like the following:

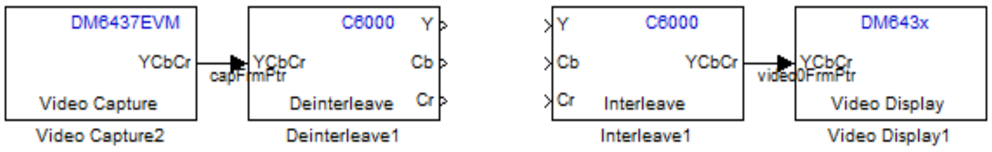


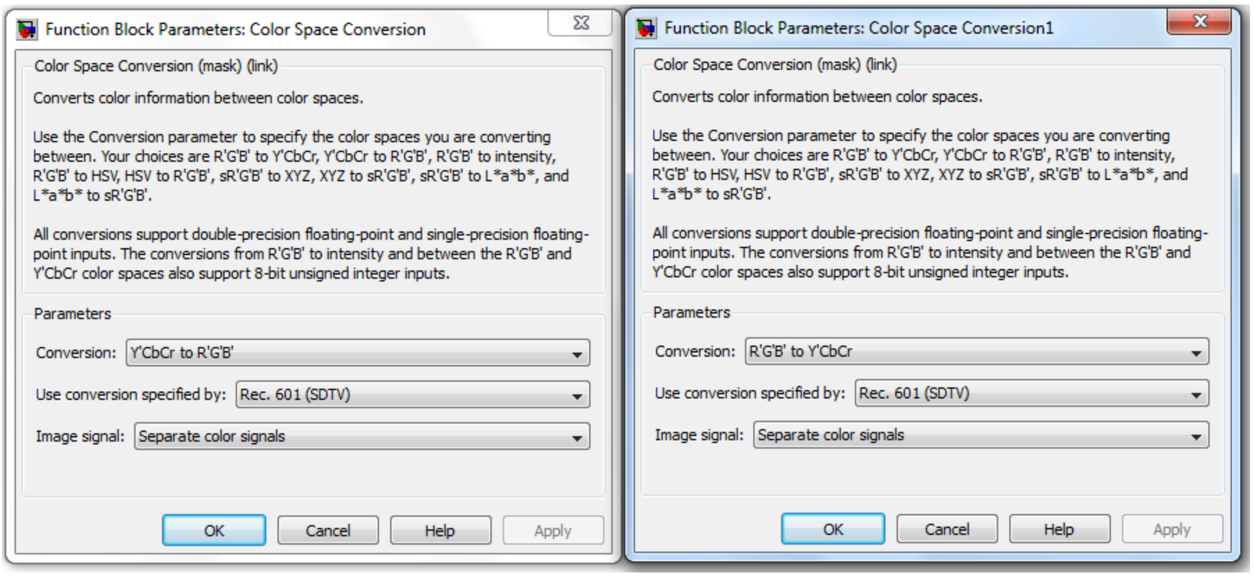
Figure 7: Part of the Simulink model

4. In User-Defined Functions, find the Embedded MATLAB Function block and type the following function in:

```
function [R_out,G_out,B_out]= fcn(R_in,G_in,B_in)
%Change these parameters to change the range identified as orange--------
Red_upper = 255;
Red_lower = 150;
Blue_upper = 200;
Blue_lower = 50;
Green_upper = 200;
Green_lower = 50;
%------------------------------------------------------------------------
R_out = R_in;
G_out = G_in;
B_out = B_in;

for i = 1:720
    for j = 1:480

        if (R_in(i,j) > Red_lower) && (R_in(i,j) < Red_upper)&& (G_in(i,j) > Green_lower) &&
(G_in(i,j) < Green_upper)&& (B_in(i,j) > Blue_lower) && (B_in(i,j) < Blue_upper)
                          R_out(i,j) = 50;
                          G_out(i,j) = 0;
                          B_out(i,j) = 0;

        end
    end
end
```

This should create three inputs and outputs onto the MATLAB block.

5. Find the Color Space Conversion from the Video and Image Processing Blockset and put two into the model. Set the first block's conversion parameter to Y'CbCr to R'G'B' and the Image signal to Separate color signals. The second block will have the same Image signal parameter of Sparate color signals but with the opposite conversion parameter (R'G'B' to Y'CbCr)


Function Block Parameters: Color Space Conversion

Color Space Conversion (mask) (link)

Converts color information between color spaces.

Use the Conversion parameter to specify the color spaces you are converting between. Your choices are R'G'B' to Y'CbCr, Y'CbCr to R'G'B', R'G'B' to intensity, R'G'B' to HSV, HSV to R'G'B', sR'G'B' to XYZ, XYZ to sR'G'B', sR'G'B' to L*a*b*, and L*a*b* to sR'G'B'.

All conversions support double-precision floating-point and single-precision floating-point inputs. The conversions from R'G'B' to intensity and between the R'G'B' and Y'CbCr color spaces also support 8-bit unsigned integer inputs.

Parameters

Conversion: Y'CbCr to R'G'B'

Use conversion specified by: Rec. 601 (SDTV)

Image signal: Separate color signals

Function Block Parameters: Color Space Conversion1

Color Space Conversion (mask) (link)

Converts color information between color spaces.

Use the Conversion parameter to specify the color spaces you are converting between. Your choices are R'G'B' to Y'CbCr, Y'CbCr to R'G'B', R'G'B' to intensity, R'G'B' to HSV, HSV to R'G'B', sR'G'B' to XYZ, XYZ to sR'G'B', sR'G'B' to L*a*b*, and L*a*b* to sR'G'B'.

All conversions support double-precision floating-point and single-precision floating-point inputs. The conversions from R'G'B' to intensity and between the R'G'B' and Y'CbCr color spaces also support 8-bit unsigned integer inputs.

Parameters

Conversion: R'G'B' to Y'CbCr

Use conversion specified by: Rec. 601 (SDTV)

Image signal: Separate color signals

Figure 8: Parameters of the first and second Color Space Conversion respectively

6. Connect the Embedded MATLAB Function block in between the two Color Space Conversion blocks in the following order:

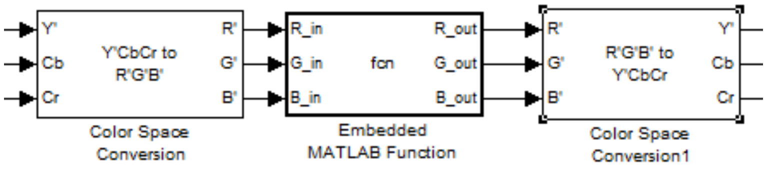


Figure 9: Color Space and MATLAB connection

7. Under Math Operations, get two Matrix Concatenate blocks. For the first Matrix Concatenate, connect the output of Cb of Deinterleave to both of the inputs. Do the same with the Cr of Deinterleave.

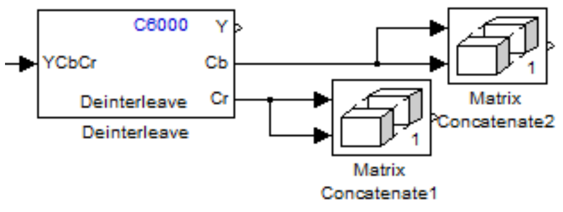


Figure 10: Connecting Matrix Concatenate

8. Now connect the output of Y to the Y' of the 'Y'CbCr to R'G'B'' Color Space Conversion block. The output of the Matrix Concatenate of Cb should go to Cb of the Conversion block and the same thing for Cr. Now the left side of the model should look like this.

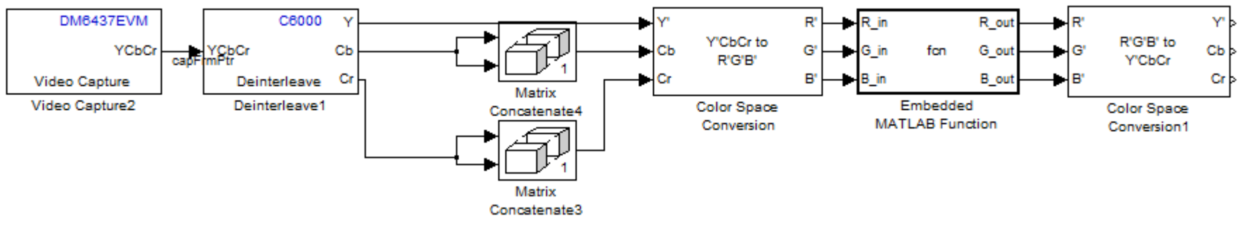


Figure 11: adding the matrices

9. Look for the Submatrix blocks in the Simulink Library. Signal Processing Blockset → Math Functions → Matrices and Linear Algebra → Matrix Operations. Get two copies of it and place them next to the 'R'G'B' to Y'CbCr' Color Conversion block. Connect one to the output of Cb and one to the output of Cr.
10. Also connect the output of Deinterleave block to the Interleave and Video Display set created earlier.

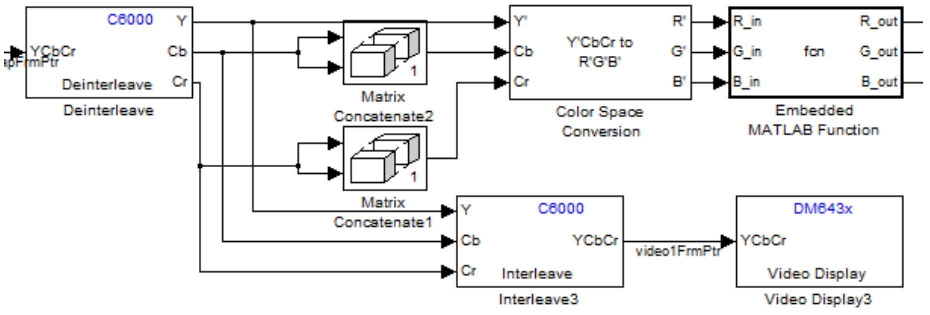


Figure 12: Intermediate step

11. Now find the set of Interleave and Video Display blocks from earlier and connect the circuit in the following fashion.

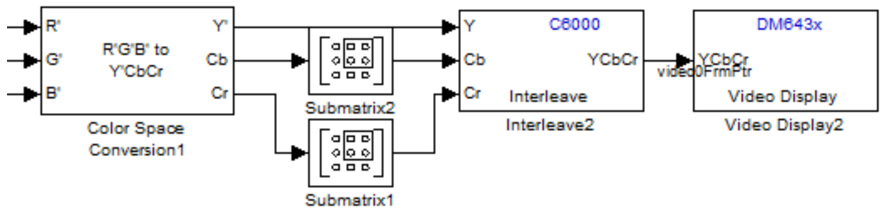


Figure 13: connecting the end

12. Lastly, find the DM6437EVM block and place it in the model. Texas Target Support Package → Supported Processors → Texas instruments C6000 → Target Preferences. This is what the completed model should look like.

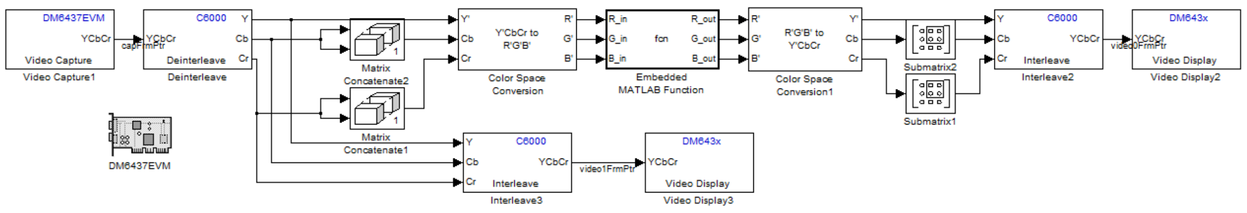


Figure 14: Put a caption here to explain what your demo implementation is about.

13. Now go the top toolbar and under Simulation, click on Configuration Parameters, and check these points:
    a. Under solver
        i. Start time: 0
        ii. Stop time: inf
        iii. Type: Fixed-step
        iv. Solver: discrete (no continuous states)
    b. Real-Time Workshop → Embedded IDE Link, make sure the System stack size (MAUs) is 8192
14. For the DSP board, make sure the power source is plugged in as well as the connection to the computer is sound. Connect the video camera to the to the rightmost input of the board and connect the tv connection into the 2nd input.
15. Tools → Real-Time Workshop → Build model, or click Ctrl-B to start the build.
16. Now test with various colors to make sure the code works and edit the Matlab code to better comprehend the parameters.

Results

Picture Software:

We started off with applying our algorithm to the software for pictures. We were able to change all the orange into maroon for the picture very easily and below is our results:

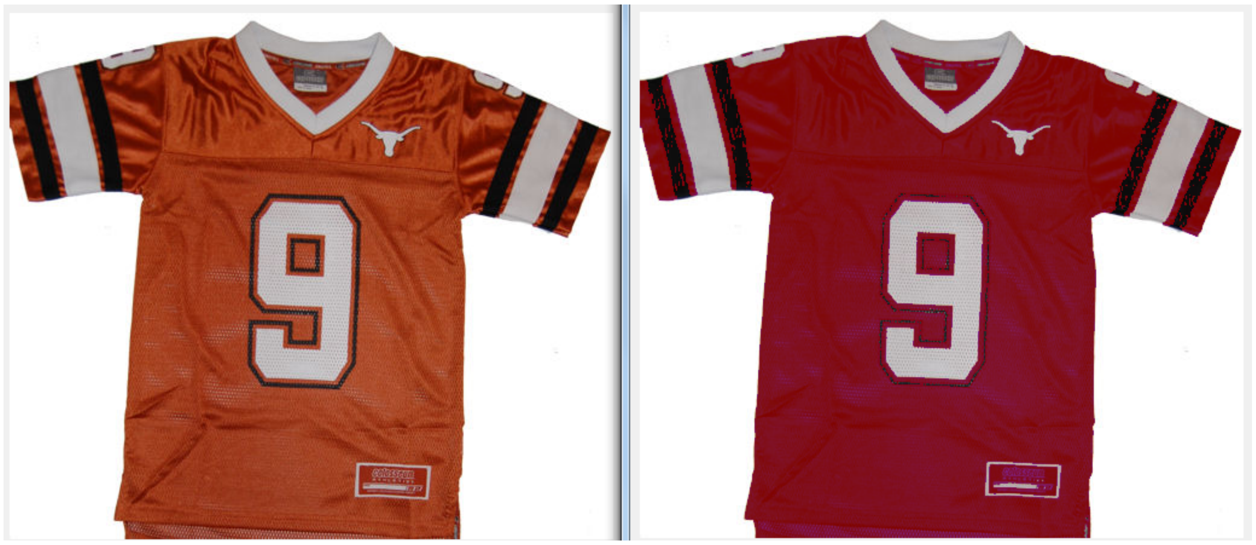


Figure 15: the Before and After output of the software

One of the things that could be worked on in the future is to better match the shade of maroon for a smoother transition.

Video Software:

The next step was to trying color processing on a video with orange material, the results are shown below:

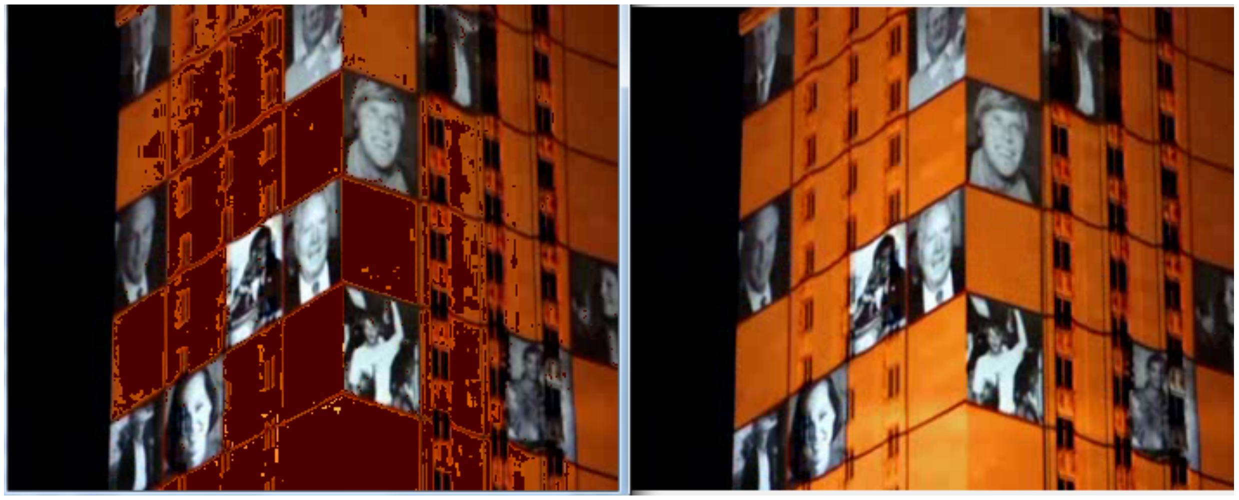


Figure 16: The Before and After output of the video software

This was harder to implement because of the continuous motion of the video and we were pretty successful in getting the orange out of the video.

Hardware Implementation:

The following screenshots are of the various orange objects turned into maroon. The bottom screen is the unfiltered image and the top will be after filtering.

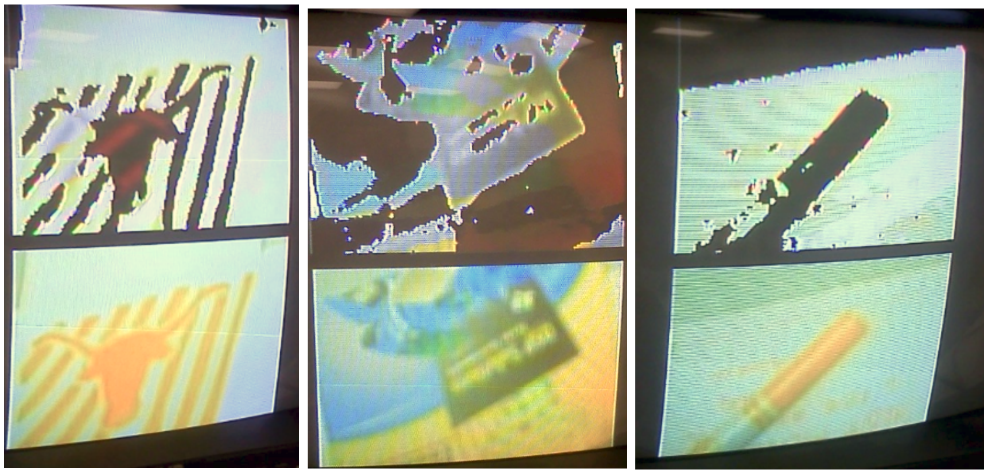


Figure 14-16: orange to maroon real time conversion

The hardware was much tougher to implement due to different sizes in the matrices. Through a great deal of experimenting with colors, we were able to find suitable number ranges for orange and block it out in real-time.

References

Kundur, Deepa, ECEN 448: Real-Time DSP Course Notes, http://www.ece.tamu.edu/~deepa/ecen448/handouts.html

Hawkes, Gregg. "Digital Component Video Conversion 4:2:2 to 4:4:4." *Introductory Digital Systems Laboratory*. Xilinx, 19 Dec. 2001. Web. 13 Dec. 2011. <http://www-mtl.mit.edu/Courses/6.111/labkit/appnotes/xapp294_04.pdf>.

Fig. i-2:

Jewett, Tom. "Color Tutorial - HSB." *TomJewett.com - Home Page*. Color Tutorial. Web. 13 Dec. 2011. <http://www.tomjewett.com/colors/hsb.html>.

Nave,R."Color Space,"*Hyperphysics*.Web.13Dec.2011.<http://hyperphysics.phy-astr.gsu.edu/hbase/vision/colspa.html>.

Fig. i-4:

Payette, Benoit. "Color Space Converter: R'G'B' to Y'CbCr." *Product Support and Documentation*. Xlinx,12 Sept. 2002.Web13Dec.2011. <http://www.xilinx.com/support/documentation/application_notes/xapp637.pdf>.

Fig. i-1:

Sequin, Carlo. "CS 184 Global_Illumination." *Foundations of Computer Graphics*. UC Berkeley. Web. 13 Dec. 2011. <http://www.cs.berkeley.edu/~sequin/CS184/TOPICS/ColorSpaces/Color_0.html>.

Fig. i-3:

"Talk:YCbCr."*Wikipedia*. Web. 13 Dec. 2011. <http://en.wikipedia.org/wiki/Talk:YCbCr>.