Color Blob Detection on Raspberry Pi at Video Rates.

Daniel Clouse

1 December, 2017

INTRODUCTION

This document describes 3 programs which together allow color blob detection to be performed on a Raspberry Pi single board computer, and the results to be sent both to a Roborio robot controller computer and to a Windows laptop for real-time viewing. The three programs are:

- input_raspicam_696.c (a plugin to mjpeg_streamer meant to run on the Raspberry Pi),
- test_udp_client.java (a test program meant to run on the Roborio), and
- dash_696.py (a tkinter GUI meant to run on the laptop).

All three programs are available at https://github.com/team696-public

Figure 1 shows the dash_696 GUI displaying the video stream with yellow color blobs outlined with red bounding boxes. The range of matching Y, U, and V color values are user-selectable, and may be specified either on the command line, or by filling out a form in the GUI, as shown.

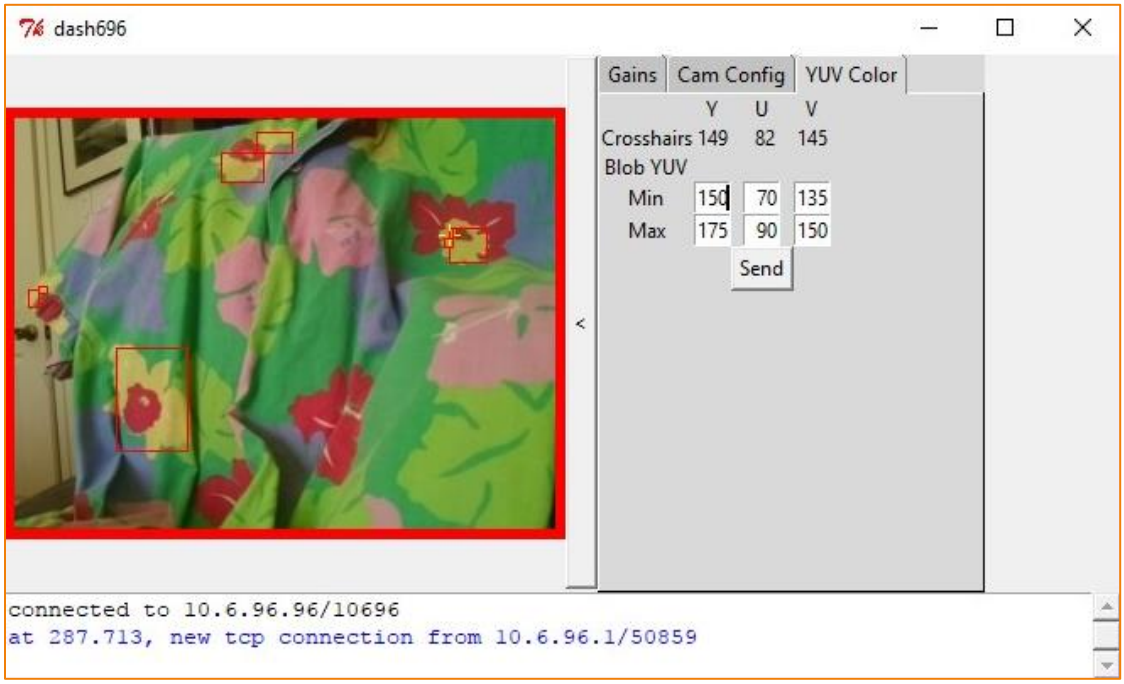

Figure 1: Connected to Raspberry PI

INPUT_RASPICAM_696.C

Input_raspicam_696.c is an mjpeg_streamer plugin. Source code for both mjpeg_streamer and for the input_raspicam_696.c plugin is available at https://github.com/team696-public/mjpeg-streamer. Instructions for compiling are included in the README file at the root of the repository. The bash script at mjpeg-streamer-experimental/start_696.sh will start the program using command line options that are known to work.

The plugin does the following:

1. captures video frames from the Raspberry Pi camera,
2. converts each frame to YUV color space,
3. detects all pixels that match specified ranges in Y, U, and V,
4. combines adjacent matched pixels into blobs,
5. calculates the bounding box, and weighted centroid positions of each blob,
6. downsamples each video frame to reduce video stream bandwidth,
7. converts each video frame to mjpeg for streaming to the laptop,
8. inserts a description of each blob in the frame header for inclusion in the mjpeg video stream
9. synchronizes the Roborio clock to the Raspberry Pi clock,
10. for each frame (timestamped with Roborio clock), sends a description of each blob to the Roborio via UDP stream,
11. accepts camera parameter settings via TCP from the dash696 GUI running on the laptop.

Using the plugin command line settings contained in start_696.sh, the plugin is capable of capturing 640x480 pixel frames at 30 frames per second and doing all the required image processing in real-time with minimal lag. To save network bandwidth, the frames are down-sampled to 320x240 before streaming to the GUI. Figure 2 shows an experiment to measure video lag. It is an image of an iphone stop watch app showing the time measured in one hundredths of a second, and the slightly delayed video frame of the stop watch displayed on the laptop. The lag between capture and display on the GUI is less than 250 msecs. This test was done using a wired ethernet connection from the Raspberry Pi to the laptop. We expect additional lag over wifi.

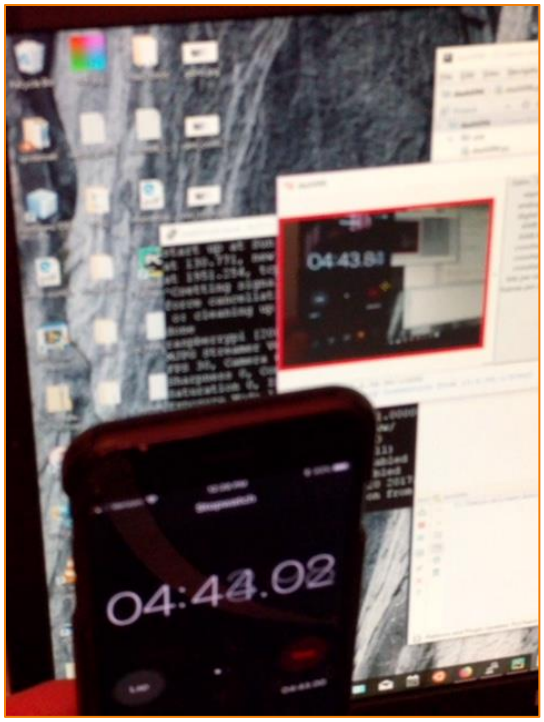

Figure 2: Video lag is less than 250 msecs.

Much of the work performed by input_raspicam_696 is done on the GPU. This includes steps 1, 2, 6, and 7, listed above. The remaining processing is done on the CPU.

We have not directly measured time required by the CPU for image processing, and in fact this time varies with the number of matching bounding boxes produced. However, for the kinds of images we expect to see in FRC play, in which we are picking out only a few matching targets, the input_raspicam_696 program runs on a Raspberry Pi Zero at about 25% CPU utilization. Slight lag of the bounding boxes relative to the rest of the image can be observed when the camera is moved quickly. This lag looks to be about 1 video frame.

Source code for input_raspicam_696 is found in the directory mjpg-streamer-experimental/plugins/input_raspicam_696. Much of the code in this directory is specific to either the mjpeg-streamer application, or to communications protocols with the dash_696 GUI (tcp_comms.c) or the Roborio application (udp_comms.c). However, the blob detection algorithm in detect_color_blobs.c was written to be more independent of the application. It should not be difficult to port this code for use with a different camera or on a different processor. Descriptions of all functions are included in the comments in detect_color_blobs.h.

Detect_color_blobs.c is fast because the first step of processing throws out all pixels that do not match the specified YUV color ranges. The pixels in a line that do match are represented using a run length encoding representation that greatly reduces the amount of memory needed to represent the relevant parts of the image. Runs that touch in adjacent lines are combined using a tree-based union operation

with path compression. To the extent possible, memory accesses are performed sequentially to maximize cache hits. An algorithm that uses similar methods is described here (Balch & Veloso, 2000).

TEST_UDP_CLIENT.JAVA

The test_udp_client.java program may be found in the directory mjpeg-streamer-experimental/plugins/input_raspicam_696/test. This program was written to demonstrate the Roborio end of the UDP comms interface used to communicate bounding box information to the Roborio from the Raspberry Pi. We haven't run this code yet on a Roborio, and some work will be needed to integrate this code into a robot program.

The code is a simple message handling loop. Each time through the loop, it waits for a message to arrive on the udp socket. Only two types of message are expected. If the message is of type Request_Time, it sends back a message of type Echo_Time. This is used both to support clock synchronization, and to allow the Raspberry Pi to check to see that the Roborio code is still running. If the message is of type Udp_Blob_List, then this test program simply prints the set of blobs that were sent. In real robot code, this behavior will need to be replaced by code to do something more meaningful with the information – e.g. aim a turret, or drive the robot towards one of the bounding boxes.

```
Class Blob_Stats {
   public short min_x;
   public short max_x;
   public short min_y;
   public short max_y;
   public int sum_x;
   public int sum_y;
   public in count;
}
```

Figure 3: Blob_Stats declaration in Java.

The Raspberry Pi time-stamps each Udp_Blob_List message with its best estimate of the Roborio clock time at which the video frame was captured. This message also includes an array of Blob_Stats, one for each blob. Figure 3 shows the declaration of Blob_Stats. It contains 7 fields. Four of the fields specify the bounding box that contains the blob. (Min_x, min_y) specifies the left, upper corner of the bounding box in pixel coordinates. (Max_x, max_y) specifies the right, lower corner. The convention used to represent pixel coordinates is that the (0, 0) pixel is the leftmost, uppermost corner of the image with the x-axis pointing to the right, and the y-axis pointing down the image. The count field specifies the number of pixels in the blob. The sum_x field specifies the sum of all the x-coordinates of all blob pixels; the sum_y field specifies the sum of all y-coordinate. From these three fields the centroid of the blob can be calculated as (sum_x / count, sum_y / count).

DASH_696.PY

Dash_696 is a tkinter GUI that runs on a laptop to communicate with the Raspberry Pi. Source code for this GUI may be found at https://github.com/team696-public/dash696.

Figure 4 shows what the GUI looks like if you start it up and the mjpeg-streamer program is not running of the Raspberry Pi. In this case, it keeps trying to connect, and every few seconds writes out an error message to let you know that it is still trying. It writes out the IP address / TCP port number it is trying to connect to. To change these values, search in dash696.py for the variables SERVER_IP_ADDR, and TCP_PORT. SERVER_IP_ADDR must match the IP address of the Raspberry Pi. TCP_PORT must match DEFAULT_TCP_COMMS_PORT specified in input_raspicam_696.c.

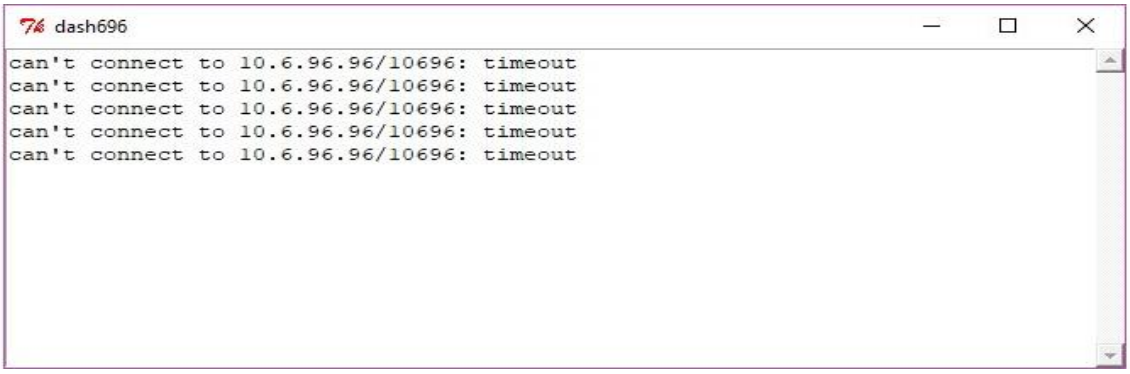

Figure 4: Dash696 GUI before connection to Raspberry Pi

Figure 5 shows the what the GUI looks like once the connection the Raspberry Pi is made. It shows the video feed on the left, a set of tabs on the right, and the text box from Figure 4 below. Notice that there are no red bounding boxes in this image. That is because there are no pixels in the frame that match the current set of YUV color ranges.

If you click on the tall vertical bar to the right of the video image that is marked "<" the tabs, and the text box will disappear leaving only the video image, as shown in Figure 6. Clicking again on that bar will restore the missing widgets (Figure 5).

The video image in Figure 5 and Figure 6 is surrounded by a red border. This indicate that there is no connection to the Raspberry Pi from the Roborio. When that connection is made, the red border turns green, as shown in Figure 7. If the Roborio disconnects, the border turns yellow.

Figure 5: Gains Tab

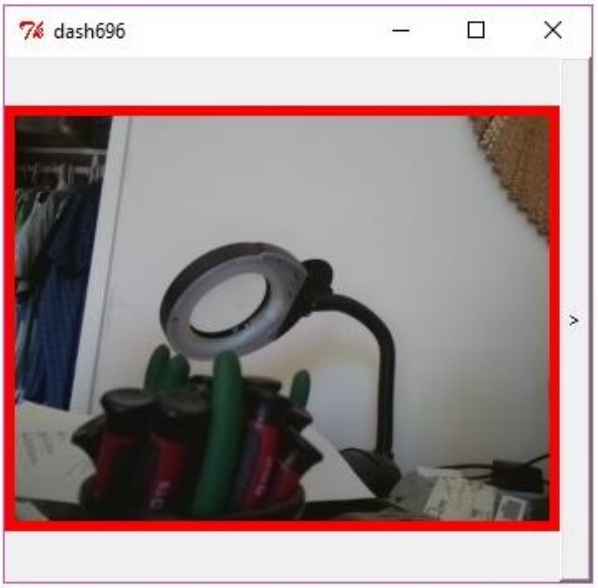

Figure 6: After clicking on "<" bar.

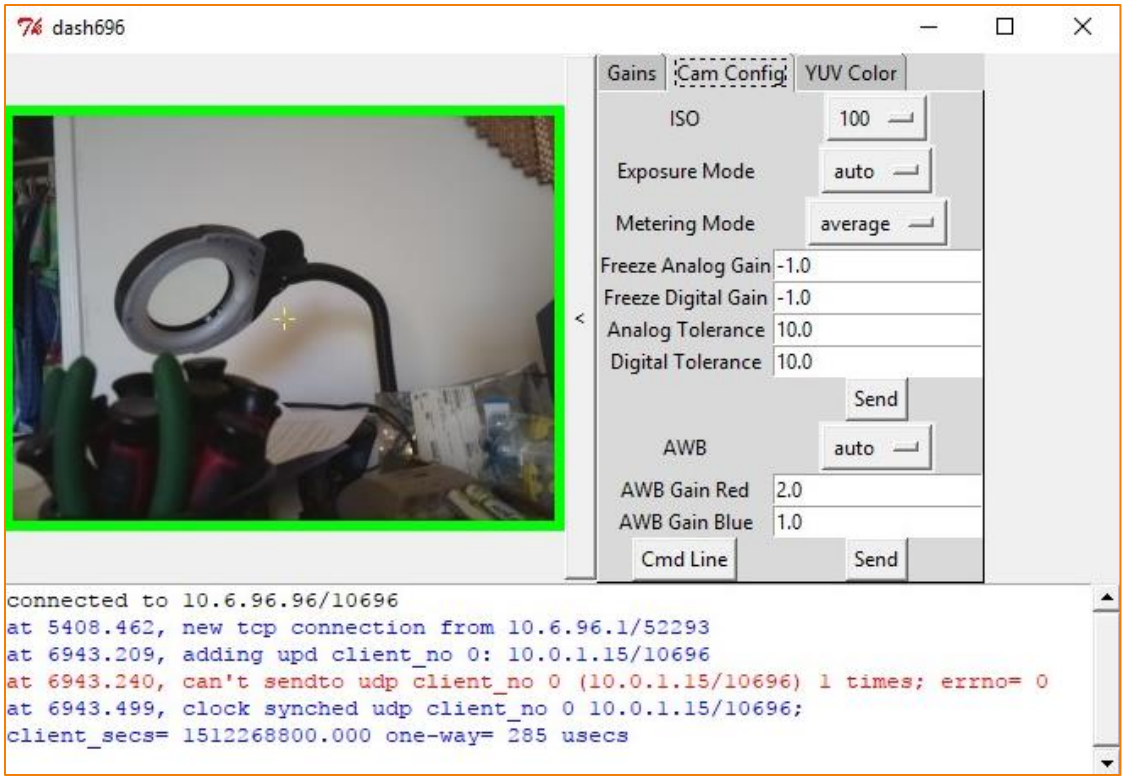

Figure 7: Connected to Roborio

In Figure 5, the tab labelled "Gains" shows the camera settings that apply to the current video frame. Figure 8 shows the meaning of each field listed on the Gains Tab. Note that the "crosshairs Y/U/V" values specify the YUV color value of the pixel at the center of the yellow crosshairs in the video frame. In Figure 5, the crosshairs are in the center of the image. You can move the crosshairs by right clicking anywhere within the image.

Name	Meaning
exposure	Exposure time of this frame
analog	Analog gain
digital	Digital gain
AWB red/blue	Auto white balance for red and blue
crosshairs Y/U/V	YUV color of the pixel in the crosshairs in the video frame
bits per second	Current video bandwidth
frames per second	Current frame rate

Figure 8: Gains Tab Fields

Figure 9 shows what the GUI looks like when you click on the "YUV Color" tab. Note that the Crosshairs Y, U, and V values from Figure 5 are duplicated here. Also, fields are provided that allow you to select the range of Y, U, and V color values that the Raspberry Pi will use to define the pixels to include in a color blob. Type in the values you want. When you click the "Send" button, the Raspberry Pi will start using the values you provided.

Figure 10 shows the effect of changing Y, U, V color settings. Notice how the red bounding boxes in the images change to select different color regions based on the Y, U, V color settings. In each of the four

cases, we moved the crosshairs to a region of interest by right clicking, then we set the Y, U, and V ranges on the YUV Color Tab to cover the color of interest, then clicked “Send.”

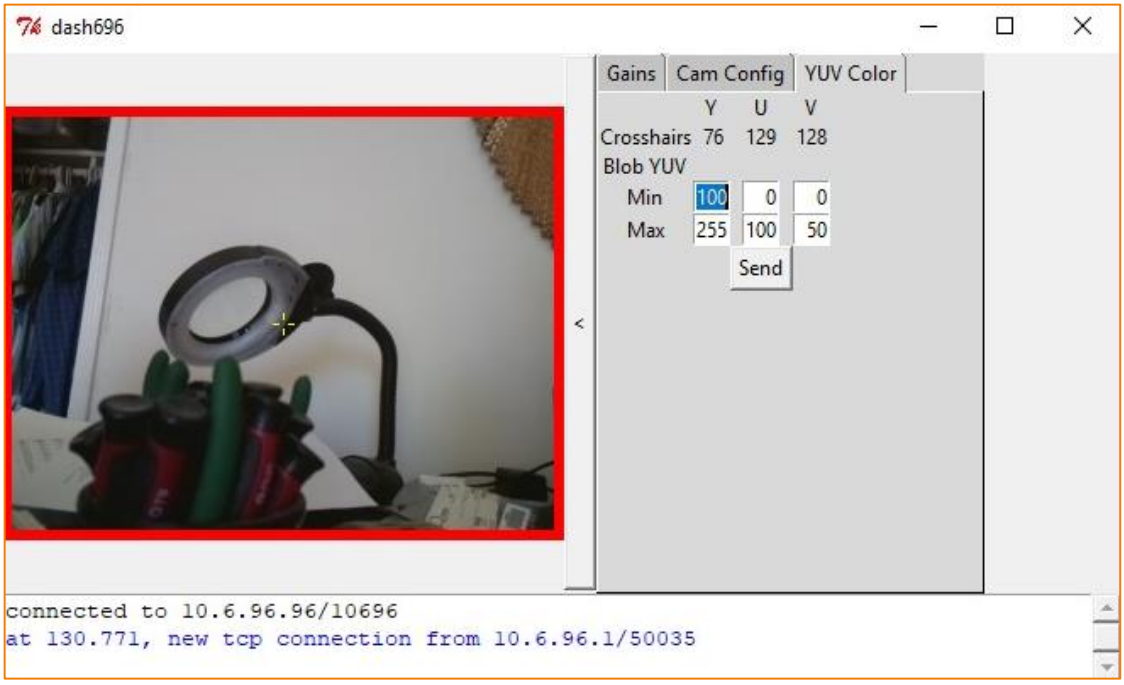

Figure 9: YUV Color Tab

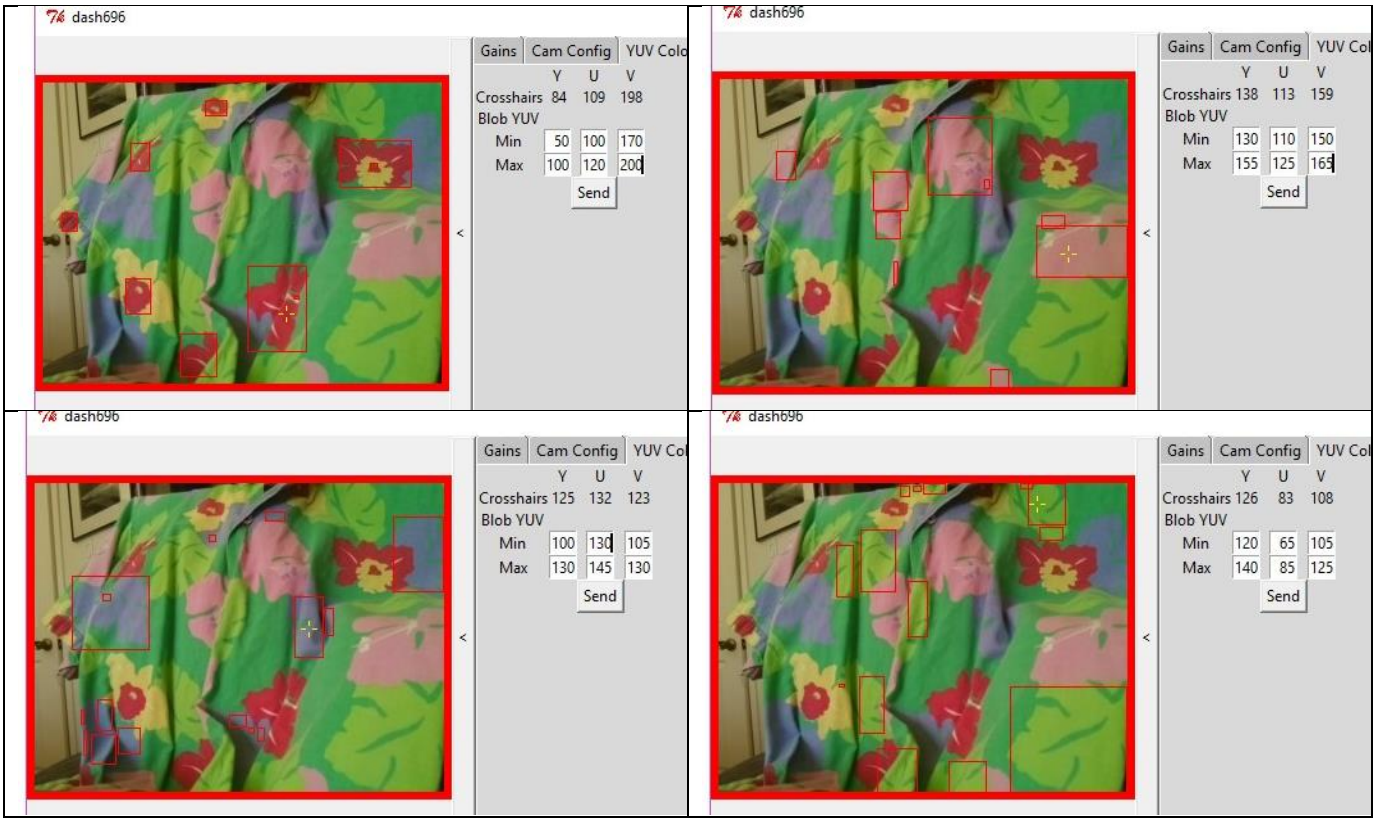

Figure 10: Changing YUV Color Settings

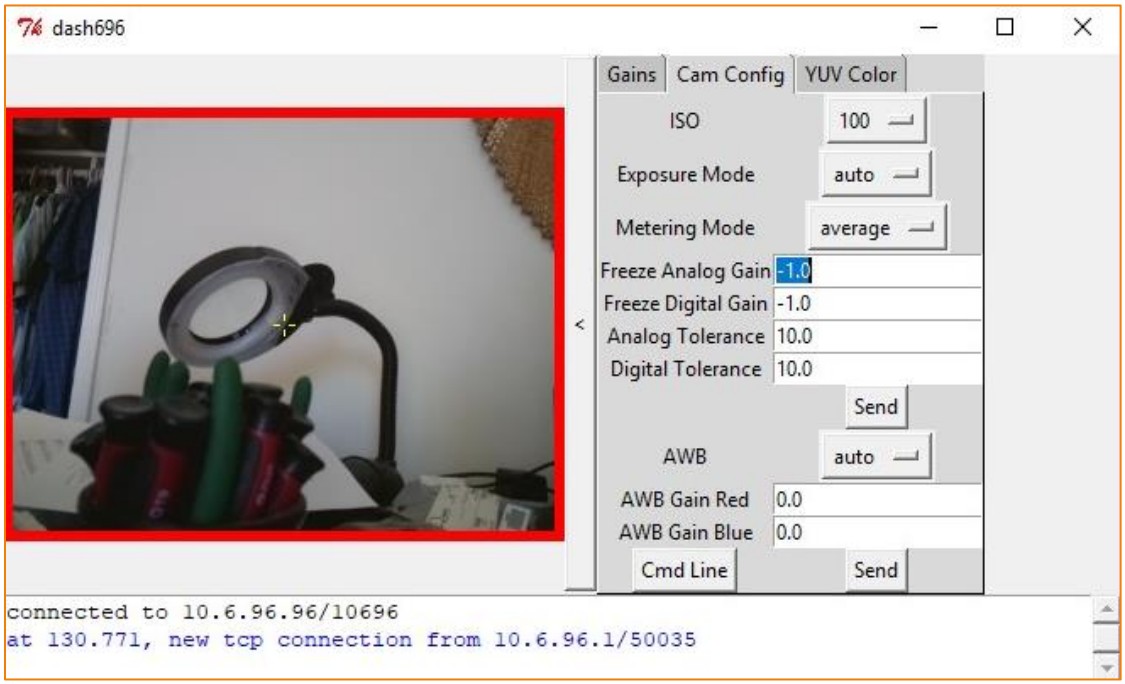

Figure 11: Cam Config Tab

Figure 11 show the Cam Config Tab. This tab allows you to set camera parameters. The Raspberry Pi camera has many more parameter settings than are shown here. We selected what we felt were the

most important settings for controlling the brightness and color balance of the image, since these are likely to be most important for making the color blob detection work in a consistent and predictable fashion. Chapter 6 of the Picamera User Manual (https://picamera.readthedocs.io/en/release-1.13/fov.html) provides the best description we have found of the camera hardware, and how it is affected by camera parameter settings.

The ISO setting affects the brightness of the image. You can click on the button that says "100" in the figure to select from the legal set of ISO values. The new value is applied as soon as you let go of the menu.

Similarly, Exposure Mode and Metering Mode allow you to select from legal values for these settings. Metering Mode sets the method used to determine the current brightness of the image for use in the automatic Exposure Mode algorithms. Exposure Mode selects from among the set of algorithms to use for setting the exposure of each frame. The exposure is controlled by adjusting the exposure time, and the analog and digital gains (mentioned in the discussion of the Gains Tab). Of most interest to us are the "auto" and "off" Exposure Mode settings.

When Exposure Mode is set to "auto" the analog and digital gains are allowed to adjust automatically, along with the exposure time. When Exposure mode is set to "off" the two gains are frozen at whatever the current settings are, and only exposure time is changed automatically.

The only method the camera provides for freezing the analog and digital gains is to set exposure mode to "off" at which point the gains are frozen at the current setting. We would like a way to freeze the analog and digital gains at values that we specify. An approximation to this desired behavior is supplied by the four parameter values: Freeze Analog Gain, Freeze Digital Gain, Analog Tolerance, and Digital Tolerance. Freeze Analog Gain and Freeze Digital Gain specify the values at which we would like to Freeze the two gain values. Analog Tolerance, and Digital Tolerance specify how close to our desired frozen values we will accept. If Exposure Mode is set to "auto", and we set these fields to meaningful values (and click "Send"), the two gains will continue to float until the point in time they happen to match our desired values. When that happens, the gains will be frozen. The Exposure Mode setting will continue to display "auto," but the "analog" and "digital" fields on the Gains Tab will change to "analog (frozen)" and "digital (frozen)," and you will notice that these field no longer will change value. To make them float again, you must change one the Freeze Analog Gain or Freeze Digital Gain values and click "Send."

OK, there are only a couple of settings left to discuss on the Cam Config Tab: the auto white balance settings and the Cmd Line button.

White balance determines the color balance between blue and red. The "auto" setting will set this balance automatically depending on the current color balance of the image. Several automatic color balance algorithms are available on the AWB pull down menu. If AWB is set to "off" then the values supplied in the AWB Red and AWB Blue fields are used. The values currently in use may be seen in the AWB red and AWB blue fields of the Gains Tab.

If you push the Cmd Line button, some text will be written to the text window. This text represents the command line to run on the Raspberry Pi to apply the current set of camera parameters from the Cam Config and YUV Color Tabs. You can play around with camera parameters in the GUI, then capture this

command line. If the next time you start up the mjpg_streamer program on the Raspberry Pi, you use this command line, instead of the one in the start_696.sh script, the program will start up using the modified set of parameters you came up with.