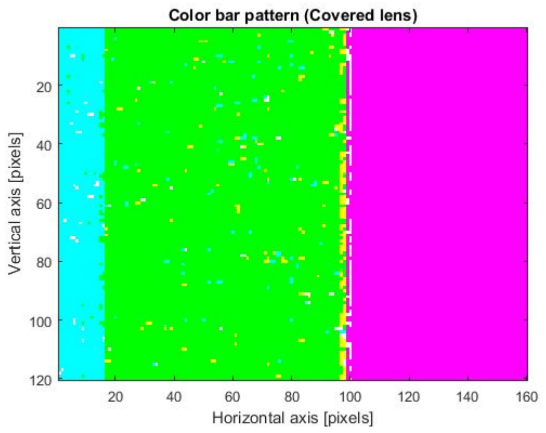
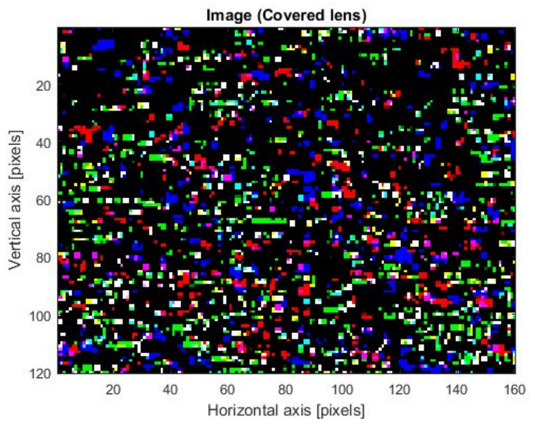
Report on SmartVision findings

I'll summarize my findings here in attempting to get the OV7675 camera working with the STM32F4VG microprocessor. I started off with basically a small section of the color bar test image being displayed and upon trying to acquire at real image, just black with some varied color dots here and there that may be noise (still unclear as to what it actually is).

Original results:





Some setup notes:
In order to run the full setup you'll need Visual Studio 2015 (community version works fine) with Visual GDB. However you need the Custom or Ultimate version of the software if you want the raw terminal which is how I get the image output. This costs over 200 francs. I don't recommend trying to build everything on your own without VisualGDB, it's a massive waste of time. There's a 30 day free trial you can install.

You will also need MatlabR2016 to run the displayrgb script which converts the text output to an RGB image. Note that the text output must be trimmed from the original terminal output which contains register setting statuses and other printf output. The only lines that should remain are just the data itself which lie between the START_PBUFFER_TRANSFER and END_PBUFFER_TRANSFER lines. This will be obvious when you look at the output.

The workflow is the following once you have the build completely set up and the correct com port hooked up in via visual GDB:
a) Look at the raw terminal (should be something like COM# in your debug tabs).
b) Click the save icon and save it to a txt file. I called it "outputTrimmed.txt" since that's what I look for in my matlab script.
c) Open the file, confirm that registers were set correctly, then remove the extraneous print line statements.
d) Run the displayrgb.m script in matlab to view the resulting RGB image (or displayyuv if you've revert back to YUV mode).

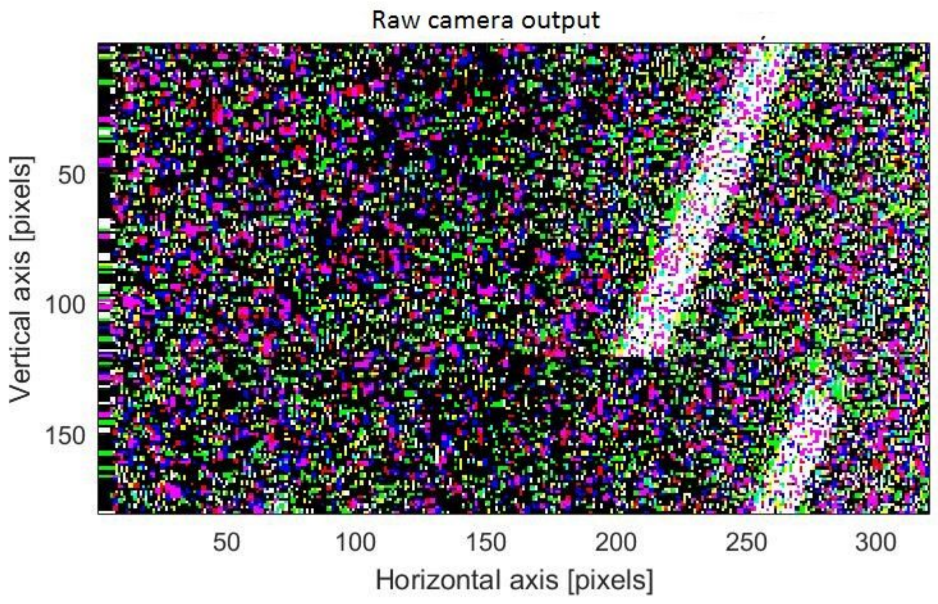
Windowing
The first thing I did was to try to fix the windowing to get a better view of the whole image.

Before we were using QQVGA which is basically a windowed version of VGA. I tried out QVGA mode which is a subsampled version of VGA. The native resolution of this is 320 x 240 and gives you the full subsampled image. Indeed windowing registers are disabled in QVGA mode and have no effect whatsoever.

Unfortunately due to SRAM limitations we can still only send a limited number of these pixels over. Because windowing is disabled we are stuck with the image starting at the beginning of the buffer. The resulting image is 320 x 180, the ⅔ of the original image.

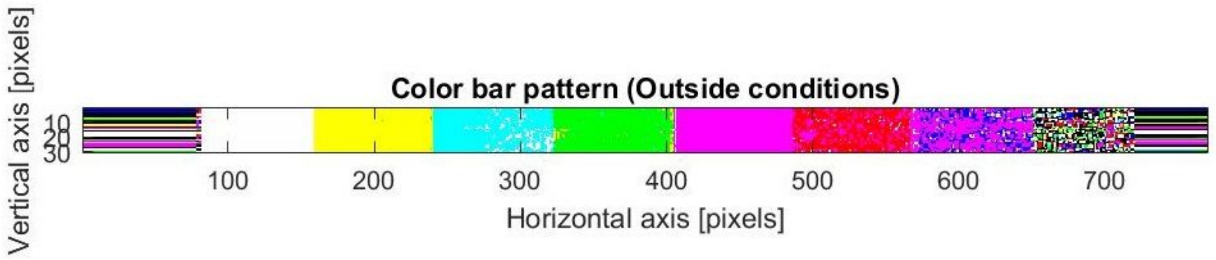
Note that I will not say it's the "top" ⅔ of the image despite starting at the beginning of the buffer because the image seems to have some odd wrapping issues where the beginning of the buffer actually starts somewhere in the middle of the image after which the "top" of the image starts. To debug this further it would be helpful to actually have a full image.

There is indeed a strange issue with one of the windowing registers (VSTOP). That register for some reason is unchangeable and while it seems that windowing has no effect in QVGA mode, the fact that that particular register is experiencing such a strange issue could be indicative of some other underlying issue.



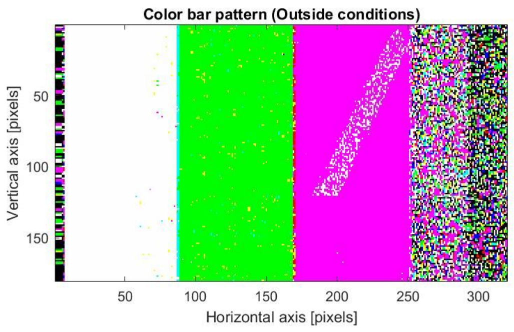
Color bar results
Using QVGA we can get the whole color bar. Important things to note:
-There are dummy pixels to the left and right of said color bar, they look like junk. The border pixels are also apparent in the real camera image as well.
-The color bar has some odd noise issues that worsens approaching the right side of the image.
-The color bar is somehow affected by the actual image that the camera would have taken while not in color bar mode. Explained in the next section.

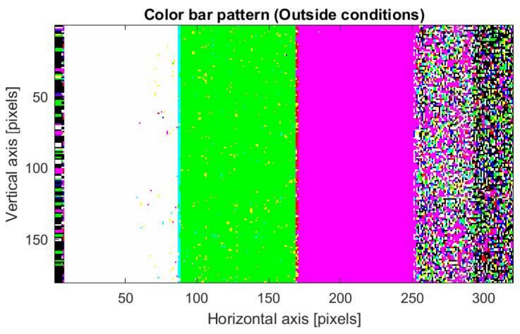
Below is a sample of windowed VGA color bar to show the presense of the dummy pixels. There are much fewer dummy pixels in QVGA mode (see previous figure) but they are still present.



Real image results
We still get just black with noise. The noise worsens going to the right of the image. We can however see certain lights in the scene and white areas. In particular it seems only sensitive to fluorescent lights. The room where I was working as rectangular fluorescent lights on the ceiling which the camera was able to pick up, but was unable to pick up something such as my phone flash light. I surmise this has something to do with the flicker rate of fluorescent lights (I believe 50 Hz in europe). Some brief research on photography issues shows that if the exposure time is not set correctly you can end up with overly dark images in which you can only see flourescent lights. This suggests that the issue here may have something to do with the exposure time settings.

Another interesting note is that a faint white silhouette appears in the color bar image where the aforementioned fluorescent lights would have appeared in the normal image, implying that somehow the actual camera image is affecting the test image. The silhouette disappears if I cover up the lens. (Uncovered on left, covered on right)





RGB vs YUV

To confirm that the data itself wasn't being corrupted, I converted the data format from YUV to RGB565. This takes the same amount of data. One pixel is still 2 bytes. I get qualitatively the same amount kind of image from YUV as I do RGB which makes me believe that the problem lies with the camera register settings themselves somehow.

Another strange thing is that when recomposing the image from the datastream in matlab, I needed to make the image width 1 pixel wider because there was an extra dummy pixel that wasn't there in YUV mode. So in RGB mode the actual resulting image size is 321 x 179. This is a minor issue however, there are certainly bigger fish that need frying.

Failed register setting attempts:
Thus I next tried to play with various settings such as exposure time, white balance, banding filters and color format settings to try to get a decent picture. My attempts however were in vain as the image never changed at all from these experiments. I will briefly describe what registers I fiddled with here but I will not bother describing the results because as I just mentioned, there was no change to report. I will leave these register setting attempts commented out in the code.

<u>AEC/AGC/AWB</u>
This stands for automatic exposure control, automatic gain control and automatic white balance control. These are all on by default and there are a few parameters that can modify their behavior.

<u>Exposure time</u>
Takes effect only when AEC is turned off. I set it to the maximum value and the minimum value and some values randomly in between.

<u>White balance</u>
The default manual white balance appears to be 2x for each of the red/green/blue channels. I tried lowering that to 1x.

Banding filter

This is the only register that mentions flicker rates, so I decided I may as well play with it.
It is off by default. I set it to on and set some registers to make the exposure times valid (since the filter requires specific exposure times).

Conclusions

Had there been some amount of incremental progress here I would have at least had some direction of how to approach the problem. However, this was unfortunately not the case. I still think however that the exposure settings warrant further investigation just based on the fact that the camera does capture those fluorescent lights.

Perhaps you can find someone who knows more generally about camera firmware registers to fiddle around here since I've pretty much been flying blind here. Altnernatively you can really just pester Omnivision until they actually respond.

Perhaps yet another option is to take the optical sensors from mice and use those instead. If you're trying to simulate bug vision I think that each camera itself doesn't really need to be particularly high resolution, so maybe an optical mouse sensor would be sufficient. Also probably pretty cheap and light weight.