Installing on Raspberry Pi

This software was written in Visual Studio 2017 and its setup as a visual studio project. If you want to do any heavy source code editing, I would recommend it be done in visual studio. The ide on raspberry pi works fine for some small changes but compiling and testing take a lot longer. If you don’t have visual studio installed, you can download the free edition from here.

<https://visualstudio.microsoft.com/vs/community/>

The repos have 3 top level folders. DebugMedia contains videos used in testing. OpenCV contains binaries and libraries specifically for Windows. UniversalMarkerDetection contains all the C++ source and header files and some visual studio project files. The C++ source files are broken up in the following way.

1. Debug.cpp: contains functions that assist in troubleshooting and tweaking some of the setting for the code. For example, placing points and drawing lines on the screen.
2. ImageProcessing.cpp: This is where most of the heavy lifting is done. In here you define colors and have functions to find patterns.
3. MathUtility.cpp: has some basic math helper functions
4. ScreenCapture.cpp: This is just used in testing on windows machine. Originally, I used this to capture images from the screen before I implemented reading from files. This is kind of legacy code and will not be used on Raspberry Pi but I’ll keep it in the project as it may be helpful again at some point.
5. Main.cpp: Start of the program.

And each source file has a header file except for main.cpp.

The first thing that needs done is to install OpenCV on the Raspberry Pi. If you don’t already have it installed this is the tutorial I used. I should note I did this right after a fresh install of Raspbian 4.1.4 (Nov 2018).

<http://pklab.net/?id=392&lang=EN>

Once you have OpenCV installed you can download and place the source files on a folder in the raspberry pi. You will not need and should not include screenCapture.cpp or screenCapture.h. The source code is still pointed to use the video files in the DebugMedia or pull from the camera.

Depending on the type of camera you are using you may need to run the following command.

sudo modprobe bcm2835-v4l2

I had to do this when I used the Pi cam but not for any usb cameras.

To test out the program you can compile it by running this command within the folder of the source files.

g++ $(pkg-config --libs --cflags opencv) -o SimpleGrab \*.cpp

The g++ is the command for the compiler. The following part tells the compiler to find the OpenCV library that we installed and link it to the project.

$(pkg-config --libs --cflags opencv)

The -o SimpleGrab options tells the compiler to name the output file “SimpleGrab”

\*.cpp tells the compiler to compile all files with the cpp extension in the directory into the executable.

Note\* If you receive an error about not being able to find an include file. You may have to change the way the file name is enclosed. Instead of using brackets use quotation marks.

#include “debug.h” vs. #include <debug.h>

After you have it running you will need to adjust variable within the source code for the specific color and number of colors you want to match.

In the file imageProcessing.h there is a macro called NUM\_MATCH. Set it to the number of colors required to match before a notification is made.

Additionally, right above the NUM\_MATCH there is another macro called MARKER\_SIZE. This sets the minimum size in pixels an area of color must be for it to be considered a match. This is where the altimeter will come into play. When this code is used with an altimeter this should be setup as a variable and a calculation to find the correct value for size of an object at specific heights.

Now lets go ahead and add a specific color to the source code to match. First in imageProcessing.h at the bottom you will see a few declarations like

extern color blue;

At the very end add a new line that says

extern color [the name of your color variable];

Then in imageProcessing.cpp you will define the range for the color in HSV values. The line looks like the following.

Color blue = { “Blue”, fullHSV(180, 80, 90), fullHSV(190, 100, 100) };

You will add a new line like that with your new color variable name. The first value “Blue” is the name of the color. The second value fullHSV(180, 80, 90) is the lower limit of the range and the third value fullHSV(190, 100, 100) is the upper value in the range.

We will be working in the HSV color space. The function fullHSV takes 3 parameters.

1. 0 – 360 degrees of hue
2. 0 – 100% of saturation
3. 0 – 100% of value

In order to help find HSV values I downloaded a free tool titled Just Color Picker. There are a bunch of them out there that is just the one I chose.

<https://download.cnet.com/Just-Color-Picker/3000-2191_4-10428271.html>

From an image you can pick out the values for the lower and upper bounds using that tool. Give yourself enough range for each value but not too much that you get hits outside of the color your looking for. Lighting plays a huge role in determining HSV value. The same color blue will not have the same HSV value in low light conditions as it does high lighting. Be aware of this when selecting the range. This is something that will need to be played around with to get the best possible outcome.

That should be all that is needed to get things working. One other note of a feature that is partially implemented but not used as of now is the ability to calculate the physical width of the image. In mathUtility.cpp there is a function called getDimensions. When passed as parameters the altitude and viewing angle of the lens being used it will return the distance across the image in the same units as the altitude.