These are basic notes of how the pi got setup. I have rearranged them to be in a better order, not necessarily the order I did them, so this exact sequence of commands wasn’t done.

The open source vision processing library (“OpenCV” for Open computer Vision) takes a lot of space, so we have to start with an 8gb SDHC micro, and install [2013-09-25-wheezy-raspbian.img (available as a torrent)](http://www.raspberrypi.org/downloads). At boot I could only get video by plugging into my HDMI TV.

run ‘**sudo nano //boot/config.txt**’, and change lines to read:

**disable\_overscan=1**

**hdmi\_force\_hotplug=1**

**hdmi\_group=2**

**hdmi\_mode=16**

yielding a more comfortable 1024x768 @ 60Hz. Supported modes can [be found here.](http://elinux.org/RPi_config.txt#Video_mode_options)

**sudo rpi-update** ## updates the pi firmware

**sudo apt-get update** ## updates the list of servers and packages

**sudo apt-get upgrade** ## 57 items updated

**sudo raspi-config** ## Enable the camera option, expand the root file

## system (make use of all 8GB), change the

## timezone and locale

**sudo apt-get install cmake libgtk2.0-dev python-dev libavformat-dev libswscale-dev** ## 89 items installed for development

If raspi-config doesn’t fix things up:

* US keyboard: **nano /etc/default/keyboard** layout to 'us'
* Time Zone with '**sudo dpkg-reconfigure tzdata**'

connecting a RJ45 gives access to the web. If you need a static address, [instructions can be found here.](http://elinux.org/RPi_Setting_up_a_static_IP_in_Debian)

Got the newest OpenCV (2.4.7 from http://opencv.org/) and extracted it to opencv-2.4.7. In a terminal from the home (~) folder:

**mkdir Paladins**

**cd Paladins**

**mkdir OpenCV**

**cd OpenCV**

**cmake ../../opencv-2.4.7** ## sets up the project for ‘make’

**make**  ## This took about **9 hours** to run!!

**sudo make install**

Set a static IP address - but only if a DHCP address isn’t given right away. This allows it to be plugged into an Ethernet cable at power-up and it will get on the internet, but if it is turned on at the robot, 10 seconds later it is registered as 10.36.18.99:

**sudo nano /etc/dhcp/dhclient.conf**

Add the following to the end:

**timeout 10;**

**lease {**

**interface "eth0";**

**fixed-address 10.36.18.99;**

**option subnet-mask 255.255.255.0;**

**renew 2 2022/1/1 00:00:01;**

**rebind 2 2022/1/1 00:00:01;**

**expire 2 2022/1/1 0:00:01;**

**}**

Then, following along with [Josh Larson's post on Chief Delphi](http://www.chiefdelphi.com/forums/showthread.php?t=118981) to [pull his code from github](https://github.com/Josh-Larson/CameraBoardAPI). Josh (with team 2502 “Talon Robotics”) out of Eden Prairie, MN spent time over the summer of 2013 to increase the speed of camera frame grabs.

In ~/Paladins:

**git clone git://github.com/Josh-Larson/CameraBoardAPI**

builds a sub-folder CameraBoardAPI

Can start the video daemon running with:

**sudo uv4l --sched-rr --driver raspicam --auto-video\_nr --width 640 --height 480 --encoding yuv420 --nopreview --imgfx blur --awb off --framerate 30**

which makes the camera available at /dev/video0

Josh has an example program (FindContours.cpp) in CameraBoardAPI/examples. By trial and error, I figured out the C++ compiler command line to make this program:

**g++ -L/usr/lib/uv4l/uv4lext/armv6l/ -I ../src/ -L ./ -lraspicam -luv4lext -Wl,-rpath,'/usr/lib/uv4l/uv4lext/armv6l',-rpath,'/usr/local/lib' `pkg-config --cflags opencv` `pkg-config --libs opencv` -o FindContours FindContours.cpp**

Notice we have to put the path in for the library (-L) as well as the linker (-rpath) to the installed video and opencv packages. With this knowledge, you should now be able to make changes within FindContours.cpp, save, compile and test (by running it with the command line ‘**LD\_PRELOAD="./libraspicam.so.0.0" ./FindContours**’).

My FindContours.cpp and supporting makefile has been uploaded to Google Drive

Remote viewing of what the pi can seen is installed, an application called [“BerryCam”](http://www.fotosyn.com/berrycam-support/).

To run the server,“sudo python /home/berryCam.py &” in a terminal.

Then download and run the app for you Droid or iPhone, connecting to the pi’s IP address over WiFi.

It can also be reached via standard web page using the URL: <http://10.36.18.99:8000/berrycam>

Yet another source of Vision processing is Team 341, who made it to Worlds in 2012 and posted their off-board **Java** code here:

<http://www.chiefdelphi.com/media/papers/2676>

Paging through that, it appears very well put together and something that can be followed. Good explanation of their variables:

<http://www.chiefdelphi.com/forums/showpost.php?p=1288982&postcount=5>

A set of good thoughts from a team last year [posted on Delphi](http://www.chiefdelphi.com/forums/showthread.php?threadid=123917). The takeaway there is configuration values stored in a file (no code changes during competition) and storing diagnostic images for post-game review. There is also reply RE a mechanism for graceful shutdown of the linux box (don’t corrupt the file system where the images are stored).

As far as learning about the format of an actual frame of data from the pi camera, I found [this wiki](http://en.wikipedia.org/wiki/YUV) to be very informative, and enabled me to pull the two chrominance (UV) components to yield a full color image.

The way FRC suggests to transfer data between computers uses WPILib’s NetworkTables. I found a project on github from FRC team 602 [“Loudoun County” out of Leesburg, VA] (<https://github.com/anidev/networktables-pc>) that is a Linux compileable version of last years WPILib NetworkTables. I had to change the CMakeLists.txt file’s CMAKE\_CXX\_FLAGS variable to be “-g -Wall -Wno-error -pthread -lrt”in order to get it to compile and link. The test\_server and test\_client programs then worked, and should be able to be used by FindContours.cpp.

Finding a way to put data to the SmartDashboard as a camera:

Installed mjpg Streamer as per <http://blog.miguelgrinberg.com/post/how-to-build-and-run-mjpg-streamer-on-the-raspberry-pi>

Am able to see changing images from any web browser -- still not knowing how to imbed in SmartDashboard.

Also, if our program outputs to /tmp/stream/pic.jpg the ‘masked’ or marked-up image, that should be the one we get remotely.