Mike Facelle
Embedded Linux Project Documentation

OPENCV INSTALLATION

1)
I followed the steps at:
http://tech.enekochan.com/2012/05/21/install-opencv-2-3-1a-in-mac-os-x-10-6/
to install OpenCV.
Because of errors trying to run make, I had to download opencv 2.3.1, not the latest version. This utilized make and cmake. To install, the following commands were run:

```
cd OpenCV-2.3.1
mkdir build
cd build
cmake -G "Unix Makefiles" ..
make sudo
make install
```

This built and installed the libraries needed for OpenCV.

2)
In order to run OpenCV programs with XCode, I followed the steps at:
http://tech.enekochan.com/2012/05/21/use-opencv-in-xcode-4-for-a-mac-os-x-application/
The instructions said to add several folders and flags to specific build parameters in the "*Build Settings*" tab.

Inside "*Search Paths*":
Header Search Paths: **`/usr/local/include`**
Library Search Paths: **`/usr/local/lib`**
Inside "*Linking*":
Other Linker Flags: **`-lopencv_core -lopencv_highgui -lopencv_imgproc`**

3)
I also installed OpenCV on the Raspberry Pi by following steps on their tutorial page:
http://docs.opencv.org/doc/tutorials/tutorials.html
This utilized cmake, as the Mac installation did. The command used were:

```
cd ~/opencv
mkdir release
cd release
cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local ..
```

and after entering the new temporary directory, the last two commands were entered:

```
make
sudo make install
```

This took quite a bit longer than the Mac OS installation, as expected. The openCV libraries were stored in `usr/local/` as specified by the cmake command.

OPENCV PROGRAMMING

4)

I followed tutorials at the OpenCV website:

http://docs.opencv.org/doc/tutorials/tutorials.html

To learn basic functionality, and more advanced techniques like line detection.

Line detection involved first using the Canny Edge method to reduce a black and white image to just lines that mark the borders between shades (when the change in color is sudden). Taken from my main.cpp file, the comments indicate parameters.

```
// source, destinaton, threshold1, threshold2, aperturesize=3
Canny(src, dst, 100, 100, 3);
```

Lines can then be detected using the HoughLines function. Taken from my main.cpp file, the comments indicate parameters.

```
// =============== PROBABILISTIC HOUGH LINE TRANSFORM =================
//      creates line segments
// dst: edge-detector output (should be grayscale)
// lines: vector to store lines found;
// rho: resolution of parameter r in pixels (using 1)
// theta: resolution of parameter theta in radians (using 1 degree)
// threshold: The minimum number of intersections to "detect" a line
// minLinLength: The minimum number of points that can form a line.
//     Lines with less than this number of points are disregarded.
// maxLineGap: The maximum gap between two points to be considered
//     in the same line.
HoughLinesP(dst, lines, 1, CV_PI/180, 25, 50, 20 );
```

5)

I filtered out some of the horizontal lines, since street lanes will almost always be vertical, by determining their slope and removing them if it falls below a certain tolerance value (0.3 was used).

I then wanted to concatenate lines that are adjacent to eachother, or simply the same line, but broken up. This meant a line must meet the following criteria:

1) *Their slopes are equal (within a tolerance, 10% is used)*
2) *Their y-intercepts are equal (within a tolerance, 10px is used)*

3a) *For adjacent lines: the nearest point on one line has a smaller magnitude than the farthest point on the closer line.*

3b) *For separated lines: the nearest point on one line has a larger magnitude than the farthest point on the closer line.*

A line is deemed "closer" if the magnitude of its far point (`l[2]` and `l[3]` for the `Vec4i` format) is larger than the magnitude of another line's far point. This is done by computing the hypotenuse at that point.

When two lines are found to be "adjacent" or "separated" they are fixed by creating a new line.

If separated, then the nearest point of the new line is simply the smallest magnitude point of the two lines, and the farthest point of the new line is simply the largest magnitude point of the two lines.
If adjacent, then both points on the new line are the average of both the nearest and farthest points of the two lines.