Edge Detection on Images

This example shows how to generate a standalone C library from MATLAB code that implements a simple Sobel filter that performs edge detection on images. The example also shows how to generate and test a MEX function in MATLAB prior to generating C code to verify that the MATLAB code is suitable for code generation.

**Prerequisites**

To run this example, you must install a C compiler and set it up using the 'mex -setup' command. For more information, see [Setting Up Your C Compiler](http://www.mathworks.in/help/coder/examples/rmvd_matlablink__541ac41191176b7cb81acfd3d803d170.html).

**Create a New Folder and Copy Relevant Files**

The following code will create a folder in your current working folder (pwd). The new folder will only contain the files that are relevant for this example. If you do not want to affect the current folder (or if you cannot generate files in this folder), you should change your working folder.

**Run Command: Create a New Folder and Copy Relevant Files**

coderdemo\_setup('coderdemo\_edge\_detection');

**About the 'sobel' Function**

The [sobel.m](http://www.mathworks.in/help/coder/examples/rmvd_matlablink__84a91422eb5fb426a7dc8f7d3043ec42.html" \t "_top) function takes an image (represented as a double matrix) and a threshold value and returns an image with the edges detected (based on the threshold value).

type sobel

% edgeImage = sobel(originalImage, threshold)

% Sobel edge detection. Given a normalized image (with double values)

% return an image where the edges are detected w.r.t. threshold value.

function edgeImage = sobel(originalImage, threshold) %#codegen

assert(all(size(originalImage) <= [1024 1024]));

assert(isa(originalImage, 'double'));

assert(isa(threshold, 'double'));

k = [1 2 1; 0 0 0; -1 -2 -1];

H = conv2(double(originalImage),k, 'same');

V = conv2(double(originalImage),k','same');

E = sqrt(H.\*H + V.\*V);

edgeImage = uint8((E > threshold) \* 255);

**Generate the MEX Function**

Generate a MEX function using the 'codegen' command.

codegen sobel

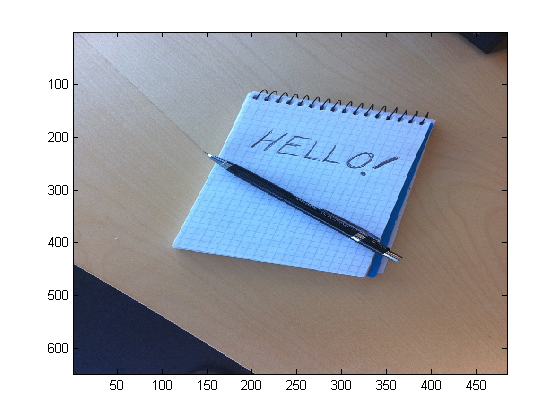
Before generating C code, you should first test the MEX function in MATLAB to ensure that it is functionally equivalent to the original MATLAB code and that no run-time errors occur. By default, 'codegen' generates a MEX function named 'sobel\_mex' in the current folder. This allows you to test the MATLAB code and MEX function and compare the results.

**Read in the Original Image**

Use the standard 'imread' command.

im = imread('hello.jpg');

image(im);



**Convert Image to a Grayscale Version**

Convert the color image (shown above) to an equivalent grayscale image with normalized values (0.0 for black, 1.0 for white).

gray = (0.2989 \* double(im(:,:,1)) + 0.5870 \* double(im(:,:,2)) + 0.1140 \* double(im(:,:,3)))/255;

**Run the MEX Function (The Sobel Filter)**

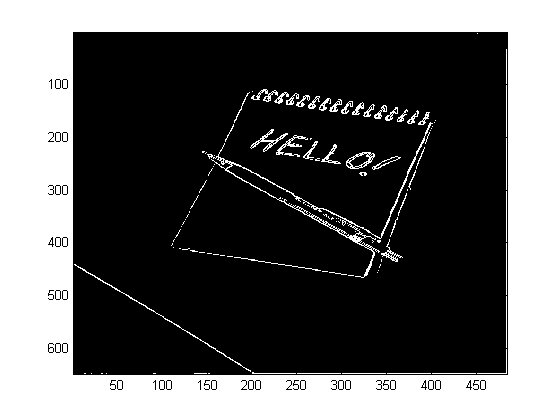
Pass the normalized image and a threshold value.

edgeIm = sobel\_mex(gray, 0.7);

**Display the Result**

im3 = repmat(edgeIm, [1 1 3]);

image(im3);



**Generate Standalone C Code**

codegen -config coder.config('lib') sobel

Using 'codegen' with the '-config coder.config('lib')' option produces a standalone C library. By default, the code generated for the library is in the folder codegen/lib/sobel/

**Inspect the Generated Function**

type codegen/lib/sobel/sobel.c

/\*

\* sobel.c

\*

\* Code generation for function 'sobel'

\*

\* C source code generated on: Fri Aug 09 01:25:43 2013

\*

\*/

/\* Include files \*/

#include "rt\_nonfinite.h"

#include "sobel.h"

#include "sobel\_emxutil.h"

#include "sqrt.h"

#include "conv2.h"

/\* Function Declarations \*/

static double rt\_roundd\_snf(double u);

/\* Function Definitions \*/

static double rt\_roundd\_snf(double u)

{

double y;

if (fabs(u) < 4.503599627370496E+15) {

if (u >= 0.5) {

y = floor(u + 0.5);

} else if (u > -0.5) {

y = u \* 0.0;

} else {

y = ceil(u - 0.5);

}

} else {

y = u;

}

return y;

}

void sobel(const emxArray\_real\_T \*originalImage, double threshold,

emxArray\_uint8\_T \*edgeImage)

{

emxArray\_real\_T \*H;

emxArray\_real\_T \*V;

int b\_H;

int c\_H;

emxInit\_real\_T(&H, 2);

emxInit\_real\_T(&V, 2);

/\* edgeImage = sobel(originalImage, threshold) \*/

/\* Sobel edge detection. Given a normalized image (with double values) \*/

/\* return an image where the edges are detected w.r.t. threshold value. \*/

conv2(originalImage, H);

b\_conv2(originalImage, V);

b\_H = H->size[0] \* H->size[1];

emxEnsureCapacity((emxArray\_\_common \*)H, b\_H, (int)sizeof(double));

b\_H = H->size[0];

c\_H = H->size[1];

c\_H \*= b\_H;

for (b\_H = 0; b\_H < c\_H; b\_H++) {

H->data[b\_H] = H->data[b\_H] \* H->data[b\_H] + V->data[b\_H] \* V->data[b\_H];

}

emxFree\_real\_T(&V);

b\_sqrt(H);

b\_H = edgeImage->size[0] \* edgeImage->size[1];

edgeImage->size[0] = H->size[0];

edgeImage->size[1] = H->size[1];

emxEnsureCapacity((emxArray\_\_common \*)edgeImage, b\_H, (int)sizeof(unsigned

char));

c\_H = H->size[0] \* H->size[1];

for (b\_H = 0; b\_H < c\_H; b\_H++) {

edgeImage->data[b\_H] = (unsigned char)rt\_roundd\_snf((double)(H->data[b\_H] >

threshold) \* 255.0);

}

emxFree\_real\_T(&H);

}

/\* End of code generation (sobel.c) \*/

**Cleanup**

Remove files and return to original folder

**Run Command: Cleanup**