Embedded Systems Modeling and Design
ECPS 203
Fall 2021

Assignment 2

Posted: October 4, 2021
Due: October 13, 2021 at 6pm

Topic: Clean C++ model of the Canny Edge Decoder with static memory allocation

1. Setup:

This assignment continues the refinement of the *Canny Edge Detector* application source code introduced in Assignment 1. We will take the first steps to prepare the code for an implementation as an embedded system. Specifically, we will perform some more code cleaning and remove the unwanted dynamic memory allocation.

We will use the same Linux account and the same remote server as for the previous assignment. Start by creating a new working directory, so that you can properly submit your deliverables in the end.

```
mkdir hw2
cd hw2
```

2. Creating a clean C++ model with static memory allocation

As starting point, we will use the single-file ANSI-C source code of the Canny Edge Detector which you have prepared in Assignment 1.

Step 1: Bug fix in **`non_max_supp`** function

Unfortunately, the orginal Canny implementation by Mike Heath contains a bug that we should fix from the beginning. Specifically, the suppression of non-maximum points in the algorithm incorrectly omits a column of pixels at the right and a row of pixels at the bottom of the image. Although it is a classic off-by-one bug in loop iterations, this bug is difficult to find without clear understanding of the internals of the algorithm.

So we will provide the bug fix here. You may copy the patched source code file from this directory on the server:

```
cp ~ecps203/public/canny.c ./
```

Please compare the provided bug-fix file to your own file from Assignment 1. You will find the bug fix applied in the **`non_max_supp`** function.

Step 2: Clean-up the source code so that there are no compilation warnings

Since we will utilize the IEEE SystemC language for simulation and exploration in later assignments, and SystemC is based on the C++ language, we will use C++ as modeling language from now on. To reflect this, rename the **`canny.c`** file into the initial C++ file **`canny.cpp`**.

Try compiling the new file with the GNU C++ compiler **`g++`**, while enabling all warnings the compiler has to offer. Specifically, use the options **`–Wall –pedantic –O2`** with **`g++`**.

You will need to apply a few more patches to the source code so that there are no errors and no warnings during the compilation. This probably will require a few iterations of source code adjustment, but this investment will pay off many times in the end. Remember, there is nothing worse than chasing a nasty bug later, when the compiler already reports a bad line in the source code at the beginning!

While it is not a requirement (not a deliverable) for this assignment, it is highly recommended that you create a suitable **`Makefile`** for building your application model. This will greatly simplify your compilation and testing iterations.

You are done with this step when your source code compiles fine without errors or warnings and the executable properly creates the output image with the correct edges. Please compare the output image against the one produced in Assignment 1. Although hardly noticeable, the only difference should be the pixels at the very right and bottom of the image, which should look better now, due to the bug fix.

Step 3: Fix the user-adjustable configuration parameters for embedded system design

In order to implement your application model later into an actual embedded system (or hardware chip), you need to decide on the configuration parameters which were kept flexible in the initial software code, but must become fixed constants for a System-on-Chip (SoC) implementation.

In your **`canny.cpp`** model, refine the source code such that the following configuration parameters become hard-coded constants:

```
rows = 240
cols = 320
sigma = 0.6
tlow  = 0.3
thigh = 0.8
```

Any other command-line parameters, such as the image file name, can only be passed to an embedded system test bench, not to the actual SoC anymore. For simplicity, we here opt for hard-coding all prior command-line arguments in the source code, so that there are no arguments to pass to the executable any more. Specifically for the file name, hard-code it also, as follows:

```
infilename = “golfcart.pgm”
```

Step 4: Remove or replace all dynamic memory allocation

Dynamic memory allocation (i.e. the use of functions **`malloc()`**, **`calloc()`**, and **`free()`**) is clearly not feasible in a hardware implementation, because the desired SoC cannot instantiate a new memory chip at runtime! Thus, we will use static arrays with fixed sizes at compile time.

As a consequence, we need to remove all dynamic memory allocation from the application source code. We suggest to start with replacing the **`malloc()`** and corresponding **`free()`** calls (and ignore **`calloc()`** for the beginning). You will notice that there are only four **`malloc()`** calls in the entire source code. Three of those are actually never used, so you can easily remove them. Moreover, you can safely remove all functions from the code that are not used (Hint: our image is a grey-scale image!).

The one remaining **`malloc()`** and the corresponding **`free()`** call should be replaced with the use of an array with fixed size. Double-check your model so that it still simulates correctly after removing the **`malloc()`** and **`free()`** calls. You may also want to create a backup file, before you apply the source code modifications in the next step.

Finally, remove or replace the function calls to **`calloc()`** and the corresponding **`free()`** calls from the source code. Again, we want to use arrays with static sizes instead.

Hint 1: In function **`make_gaussion_kernel`**, an array **`kernel`** is filled with parameters. The size of this array (variable **`window_size`**) generally depends on the configuration parameter **`sigma`**. However, since we set **`sigma`** to a constant value in the previous step, **`window_size`** also becomes a fixed value. Specifically, **`sigma=0.6`** leads to **`window_size=5`**.

This will work fine as long as you don't change these values independently. However, if we find later, for whatever reason, that **`sigma`** needs adjustment for a better output picture, we will also need to adjust **`window_size`** (otherwise, for a too-small value, we will be running into an array-index-out-of-range problem). We can avoid this problem by setting **`window_size`** larger, say **`window_size=21`** which allows **`sigma`** to increase up to 4.0. The maximum value 4.0 is chosen here based on the following comment from the original code: *"A broad range of parameters to use as a starting point are: sigma 0.60-2.40, [...]"*. So 4.0 is a reasonable upper bound, leaving some safety margin from the indicated boundaries.

Hint 2: The two functions **`radian_direction`** and **`angle_radians`** in the original Canny implementation are useful to demonstrate the working of the algorithm (the resulting gradient direction image can be output to a file and then be viewed). However, this functionality serves no purpose in our embedded system model where we are only interested in the final edge image. Thus, you can safely remove both functions (and the included dynamic memory allocation) from the source code of your model.

3. Submission:

For this assignment, submit the following deliverables:

```
canny.cpp
canny.txt
```

The text file should briefly describe whether or not your efforts were successful and what (if any) problems you encountered. We will take this input into account when grading your submission. Please be brief!

To submit your deliverables, change into the parent directory of your **hw2** directory and run the **~ecps203/bin/turnin.sh** script. As before, this command will locate the current assignment files and allow you to submit them.

Finally, remember that you can use the turnin-script to submit your work at any time before the deadline, *but not after!* Since you can submit as many times as you want (newer submissions will overwrite older ones), it is highly recommended to submit early and even incomplete work, in order to avoid missing the hard deadline.

Late submissions will not be considered!

To double-check that your submitted files have been received, you can run the **~ecps203/bin/listfiles.py** script.

For any technical questions, please consult with the TA in the lab or use the course message board.

--

Rainer Doemer (EH3217, x4-9007, doemer@uci.edu)