Embedded Systems Modeling and Design
ECPS 203
Fall 2018

Assignment 4

Posted: October 24, 2018
Due: October 31, 2018 at 6pm

Topic: From single image to video stream processing

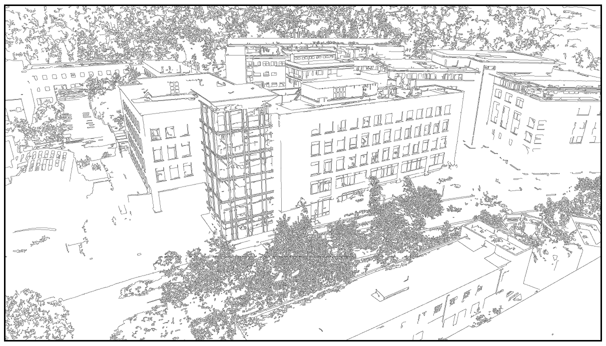
1. Setup:

With this assignment, we resume the modeling of our application example, the Canny Edge Detector. This week, we will convert the application from single image to video stream processing. More specifically, we will process a sequence of images extracted from a stream of video frames.

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

```
mkdir hw4
cd hw4
```

Instead of the previous golf cart image, we will from now on use a video that we have captured in 2017 by flying a drone around the Engineering Hall building at UCI. As before, we want to convert the original video frames into edge images using the Canny algorithm.



Our captured video is available in a shared directory on the server. To save disk space, please do not copy the data into your account, but create a symbolic link to it, as follows:

```
ln -s ~ecps203/public/DroneFootage DroneFootage
```

Note that we do not have a working video player available on the EECS Linux servers (for copyright and network bandwidth reasons). However, you can easily download the **`DroneFootage/DJI_0003.MOV`** movie file to your local laptop and play it there.

2. Prepare a set of video frames suitable for use in a test bench

In order to use the captured video stream in a test bench of a SystemC model, we need to extract a sample set of frames and convert those to grey-scale images that our Canny application can process.

Step 1: Extract individual video frames from the movie file

In order to store image frames, let's first create a directory **`video`** for storing these frames:

```
mkdir video
cd video
ln -s ../DroneFootage/DJI_0003.MOV
```

Our captured video is a few minutes long, but for our design and development in SystemC, we only need a few representative frames for processing in our test bench. To keep later timing calculations simple, we will extract 30 frames from the stream, representing 1 second of real-time video at 30 frames per second (FPS).

We have installed the **`ffmpeg`** software package on the Linux servers which is a powerful video processing tool. To get an overview, browse the **`ffmpeg`** documentation which is available online here:

```
https://www.ffmpeg.org/ffmpeg.html
```

A typical call to extract video frames looks like this:

```
/opt/pkg/ffmpeg/bin/ffmpeg -ss starttime -t length -i video.mov
    -r ratio frame%03d.png
```

Select suitable parameters for ***`starttime`***, ***`length`***, and ***`ratio`*** in order to extract 30 "pretty" video frames of your choice from our drone movie.

As a result, you should store 30 image files named "**`Engineering001.png`**" through "**`Engineering030.png`**" in your **`video`** directory. You can inspect these images then with the Linux **`eog`** tool.

Step 2: Convert the color frames to grey-scale images in PGM format

Since our Canny application can only read PGM input images, we need to convert the PNG frames appropriately. Here, you can use the **`pngtopnm`** and **`ppmtopgm`** Linux tools. You can run these either separately or simply in a shell pipeline fashion. Please refer to their manual pages for detailed instructions.

As a result, you should create 30 additional image files named "**`Engineering001.pgm`**" through "**`Engineering030.pgm`**" in your **`video`** directory. Again, you can view these images easily with the Linux **`eog`** tool.

3. Convert your Canny application to process a stream of video frames

With the video frames prepared, we can now remodel the Canny application from Assignment 2 in order to process these images and convert them into corresponding edge images.

Step 1: Recode your Canny C++ model to process a sequence of video frames

As starting point, you can use your own C++ model which you have created in the previous Assignment 2. Alternatively, you may start from the provided solution for Assignment 2 which you can copy as follows:

```
cd hw4
cp ~ecps203/public/CannyA2_ref.cpp Canny.cpp
```

To process the 30 frames, put a loop into the main function around the load, canny, and save function calls and adjust the code so that the names used match the frame names prepared in your **`video`** directory. For example, the first image read should be "**`video/Engineering001.pgm`**" and the corresponding edge image should be stored as file "**`video/Engineering001_edges.pgm`**".

Note that you will also need to change the image size from formerly 320x240 pixels to the new higher resolution of 2704x1520 pixels produced by the drone camera.

The higher image resolution naturally leads to higher memory usage of our application. Specifically, the array variables holding the image data (which we introduced in Assignment 2) grow large. Note that many of those variables are local variables which get allocated on the stack. At the same time, the stack size of a process in the Linux environment is typically limited. If so, your application will "crash" with a segmentation fault or similar memory error.

To avoid stack overflow, you can adjust the stack space allocation in your Linux shell. This configuration depends on the shell you are using, which you can identify with the following command:

```
echo $SHELL
```

If you use the **`csh`** or **`tcsh`** shell, then adjust your stack size as follows:

```
limit stacksize 128 megabytes
```

On the other hand, if you use the **`sh`** or **`bash`** shell, then set your stack size like this:

```
ulimit -s 128000
```

With this large stack allocation in place, your application model should run without memory problems. Run the Canny application on the prepared input images and validate the generated edge images in the **`video`** directory with the **`eog`** tool.

Step 2: Calibrate the Canny parameters to optimize the output images

As discussed in Assignment 2, the Canny algorithm uses a few configuration parameters that can be adjusted to generate "better" output images. Since we are no longer using the golf cart image, the settings for these parameters may change as well.

In order to optimize the **`sigma`**, **`tlow`** and **`thigh`** constants, we have prepared an interactive Canny calibration tool based on OpenCV examples. You can run this tool on any image as follows:

```
~ecps203/bin/CannyCalibration ImageFileName
```

The **`CannyCalibration`** tool will open a graphical user interface (GUI) with several windows showing the **`ImageFileName`** in original, black-and-white blurred, and edge format. In the calibration window, three sliders allow to adjust the minimum and maximum threshold and the kernel size used in the Canny algorithm. Any modification of these values will immediately show effect in the output image.

Find the "best looking" values for your chosen video frames and adjust your model source code to match these settings.

4. Submission:

For this assignment, submit the following deliverables:

```
Canny.cpp
Canny.txt
```

Again, the text file should briefly mention whether or not your efforts were successful and what (if any) problems you encountered. Please be brief!

Note that you don't need to submit your image files. We only need your application source file and corresponding text file for grading. To submit these files, change into the parent directory of your **`hw4`** directory and run the **`~ecps203/bin/turnin.sh`** script.

As before, remember that you can use the turnin-script to submit your work at any time before the deadline, *but not after!* We recommend to submit early and even incomplete work, in order to avoid missing the hard deadline.

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

For any technical questions, please use the course message board.

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