EECS 222: Embedded System Modeling
Winter 2020

Assignment 1

Posted:	January 7, 2020
Due:	January 15, 2020 at 6pm
Topic:	Setup and Introduction to Application Example

1. Setup:

A Linux account has been created for you on the EECS department servers. The login and password credentials are the same as for your UCInetID.

Any of the following hosts can be used as computing server:

```
crystalcove.eecs.uci.edu
zuma.eecs.uci.edu
bondi.eecs.uci.edu
laguna.eecs.uci.edu
```

In order to remotely connect to the server, you will need to use the secure shell protocol (SSH) via a client software terminal. Note that we will mostly use the command line interface, so a simple terminal program (e.g. **`Putty`** for Windows) will be sufficient. Occasionally, however, we will also need graphical tools with GUI for which you need X client software (e.g. **`Xming`**). See the course web site resource page for pointers to suitable SSH clients.

Use your SSH terminal to connect to one of the servers and make yourself familiar with the available commands in the Linux environment. You may also configure your shell setup to have a convenient working and C/C++ programming environment.

For this course, you will need to be comfortable with editing text and source code files in a text editor (e.g. **`vi`**, **`vim`**, **`pico`**, **`emacs`**, or **`gedit`**). You will also need to be familiar with the GNU C/C++ compiler chain for building executable programs. If you are not familiar with these tools, study available online tutorials and other resources and practice C/C++ programming in Linux.

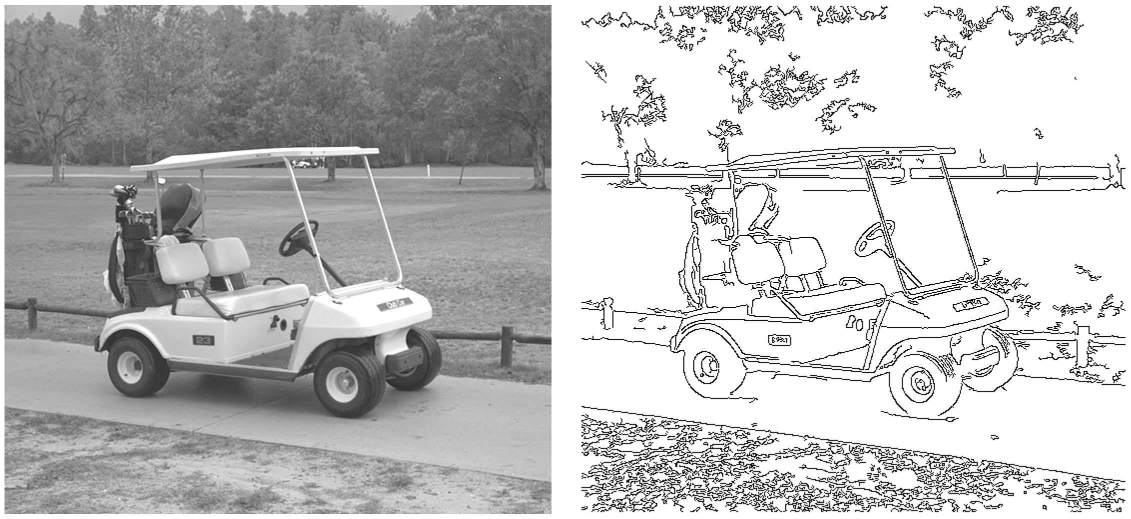
Since our modeling example is an image processing application, you will also need tools for handling digital images. There are many command-line tools available in Linux that allow you to convert and manipulate images. Most start with **`pnm`**, **`pbm`**, or **`pgm`**, depending on the file format they process. To enumerate them, you can type their prefix into your shell followed by a **`TAB`**. Documentation is available via the corresponding **`man`** pages.

In addition, graphical tools are available as well (if you have an X client running), which may be more convenient to use. To view images, you can use **`eog`** which supports most image types (including our pgm format). Finally, in order to manipulate images (e.g. if you want to use your own photos as a test case for our application), you can use **`gimp`**, a very powerful image editor in Linux.

2. Application Example

For the embedded system modeling project in this course, we will use a specific image processing application, namely an implementation of the *Canny Edge Detector* algorithm. The overall project goal is to design a suitable embedded system model of this application and describe it in a System-Level Description Language (SLDL). This embedded specification model will then not only be simulated for functional and timing validation, but also be refined for synthesis and implementation as an embedded System-on-Chip (SoC) suitable for real-time use in a digital camera.

The Canny Edge Detector algorithm takes an input image, e.g. a digital photo, and calculates an output image that shows only the edges of the objects in the photo, as illustrated in the figure below. We will assume that this image processing is to be performed in real-time in a digital camera by a SoC that we model and design.



Please refer to the following sources for more information on the Canny application:

John Canny, *"A Computational Approach to Edge Detection"*, IEEE TPAMI, 1986.

http://en.wikipedia.org/wiki/Canny_edge_detector

ftp://figment.csee.usf.edu/pub/Edge_Comparison/source_code/canny.src

You may refer to other web sites as well to study the Canny approach for edge detection in digital images.

3. Instructions

The purpose of this first assignment is for you to become familiar with the Canny algorithm and its application source code.

Step 1: Download the application source code

You can download the Canny application in C source code from the web site at ftp://figment.csee.usf.edu/pub/Edge_Comparison/source_code/canny.src.

Alternatively, you can copy the same source file and a suitable input image from our dedicated instructor account for this course:

```
mkdir eecs222
mkdir eecs222/hw1
cd eecs222/hw1
cp ~eecs222/public/canny.src .
cp ~eecs222/public/golfcart.pgm .
```

We will use this C reference implementation of the Canny edge detection algorithm as the starting point for our embedded design model. The golf cart image will serve as an initial input image.

We provide one more tool, named **`ImageDiff`**, which you can access via a symbolic link:

```
ln -s ~eecs222/public/ImageDiff ImageDiff
```

You can use this command line tool for comparing PGM images pixel by pixel (instead of the Linux **`diff`** tool). The **`ImageDiff`** tool is actually built from Canny source code functions and compares the individual pixels of two input images (first and second argument) and generates an output image (third argument) which shows the differences. It also reports the number of mismatching pixels found. For example, use **`ImageDiff`** as follows:

```
./ImageDiff Image1.pgm Image2.pgm diff.pgm
```

The mismatching pixels found are then indicated in the generated **`diff.pgm`** file.

Step 2: Test the given C code

Convert the downloaded **`canny.src`** file into a *single* ANSI-C file **`canny.c`**. Few adjustments will be necessary in order to compile the application on our Linux infrastructure with a modern compiler chain. Then you can compile and test the application, similar to the following:

```
vi canny.c
gcc canny.c -lm -o canny
./canny golfcart.pgm 0.6 0.3 0.8
eog golfcart.pgm_s_0.60_l_0.30_h_0.80.pgm
```

The generated output image should look similar to the one shown above.

Step 3: Study the application

Before we go and write a system-level model of this application, we need to study and understand it well. Examine the source code and its execution!

You should start thinking about questions like the following:

What are the main functions of the algorithm? Which functions are used for data input and for data output? Where does the actual computation occur? Which of the functions is the one with the most complexity? Can we expect the application to run in real-time as required by our overall goals? How much memory is needed when the algorithm runs? Are there any obvious candidates for hardware acceleration? What should better be performed in software? …

Some of these questions are easy, some are harder. Don’t worry, you don’t need to have all the answers at this time, but you may want to start thinking about these issues so that we can discuss them in class throughout this course.

Step 4: Bug fix in **`non_max_supp`** function

Unfortunately, the original 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. This case is an example of a classic off-by-one bug in loop iterations.

Locate the **`non_max_supp`** function in the source code and take a close look at the two loops iterating over the inner pixels of the image. You will find that a column of pixels at the right and a row of pixels at the bottom are not included in the computation.

Fix this bug and compare the new output image against the image produced by the buggy implementation. The **`ImageDiff`** tool should be very helpful here! The only difference should be the pixels at the very right and bottom of the image, which should look better now.

Step 5: Clean-up the source code so that there are no compilation warnings

When compiling your C code with the chosen compiler, you will want to enable all warnings the compiler has to offer. For the GNU compiler **`gcc`** in particular, use the option **`–Wall`** for the compilation. This will result in a number of warnings displayed which should be addressed by correcting the source code appropriately.

You will need to apply a few patches to the source code so that there are *no warnings* and *no errors* 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 points out a bad line of code at the beginning!

You are done with this step when your source code compiles fine without errors or warnings and the executable properly creates the expected output image with the correct edges.

3. Submission:

For this assignment, turn in the following deliverables:

```
canny.c
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 these files, change into the parent directory of your **`hw1`** directory and run the **`~eecs222/bin/turnin.sh`** script. This command will locate the current assignment deliverables and allow you to submit them, as follows:

```
doemer@bondi.eecs.uci.edu: ~eecs222/bin/turnin.sh
=====================================================
EECS222 Winter 2020:
Assignment "hw1" submission for doemer
Due date: Wed Jan 15 18:00:00 2020
** Looking for files:
**   canny.c
**   canny.txt
=====================================================
* Please confirm the following:                     *
* "Following the Academic Honesty Policy at UCI,    *
* I submit my own original work."                   *
* Please type YES to confirm. YES
```

```
====================================================
Submit canny.c [yes, no]? y
  File canny.c has been submitted
Submit canny.txt [yes, no]? y
  File canny.txt has been submitted
====================================================
  Summary:
====================================================
Submitted on Mon Jan  6 18:01:26 2020
You just submitted file(s):
  canny.c
  canny.txt
```

Note that you can use this **turnin.sh** 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 simply overwrite older ones), it is highly recommended to submit early and even incomplete work, in order to avoid missing the deadline.

The deadline is hard. Late submissions will not be considered!

To double-check that your submitted files have been received, you can run the following script:

```
doemer@bondi.eecs.uci.edu: ~eecs222/bin/listfiles.py
=====================================================
EECS 222 Winter 2020: "hw1" listing for doemer
=====================================================
Files submitted for assignment "hw1":
canny.c
canny.txt
```

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

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