ECPS204 Setup-Part B

CHENG-CHIH, LEE, 29329351
JONATHAN KIM, 42411903

Project Description (2 pts)

- The purpose of this part of the project is to prepare the environment for the canny application with the camera and further modify the canny application to process multiple images in a row. Profiling the canny is also included for evaluating the application performance.
- Finally, we want to encode the captured images into a video.

1. Experimental Setup (4 pts)

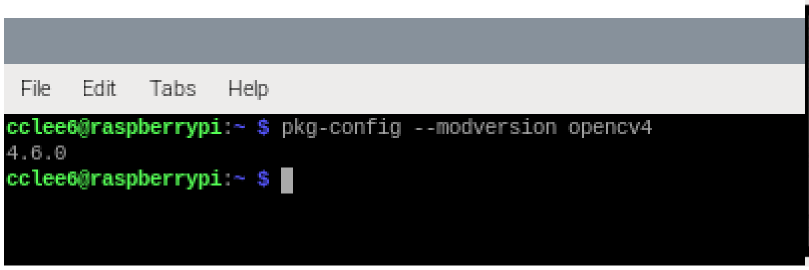
- For this project, we set up a Git repository to exchange files between my RPi and laptop. Following the instructions, we get each step done and recorded.
- We had to use libcamera to do simple camera control since we're using RPi 5 with newer SW.

2. Results (6 pts)

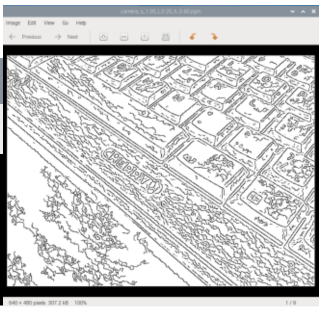
1. **Step 1 & 2:** Install OpenCV and understand the sample code

   -OpenCV4 is successfully installed.

   -Canny application is implemented and we captured one image of my keyboard as shown below.



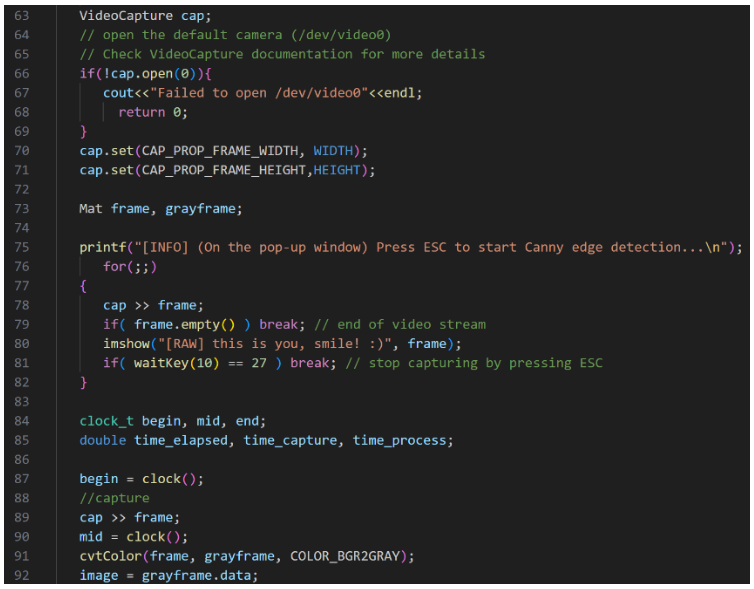
```
cclee6@raspberrypi:~ $ pkg-config --modversion opencv4
4.6.0
cclee6@raspberrypi:~ $
```



   - There are three files included in this project, they are canny_util.c, canny util.h, and camera_canny.cpp.

canny_util.c and canny util.h are the algorithm source code and the header file; camera_canny.cpp is for utilizing the camera to capture images and canny algorithm to process them, including exporting the processed files.

-*Quick overview of the **sample code**:*



```
VideoCapture cap;
// open the default camera (/dev/video0)
// Check VideoCapture documentation for more details
if(!cap.open(0)){
    cout<<"Failed to open /dev/video0"<<endl;
      return 0;
}
cap.set(CAP_PROP_FRAME_WIDTH, WIDTH);
cap.set(CAP_PROP_FRAME_HEIGHT,HEIGHT);

Mat frame, grayframe;

printf("[INFO] (On the pop-up window) Press ESC to start Canny edge detection...\n");
    for(;;)
{
    cap >> frame;
    if( frame.empty() ) break; // end of video stream
    imshow("[RAW] this is you, smile! :)", frame);
    if( waitKey(10) == 27 ) break; // stop capturing by pressing ESC
}

clock_t begin, mid, end;
double time_elapsed, time_capture, time_process;

begin = clock();
//capture
cap >> frame;
mid = clock();
cvtColor(frame, grayframe, COLOR_BGR2GRAY);
image = grayframe.data;
```

a.Declare VideoCapture cap

b.Open cap and check status

c.Setting width and height of frames

d.Declaring frame and grayframe

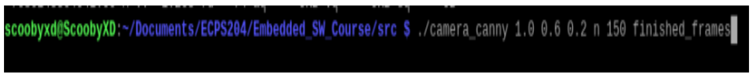
e.for loop to show camera images until “Enter” key is pressed

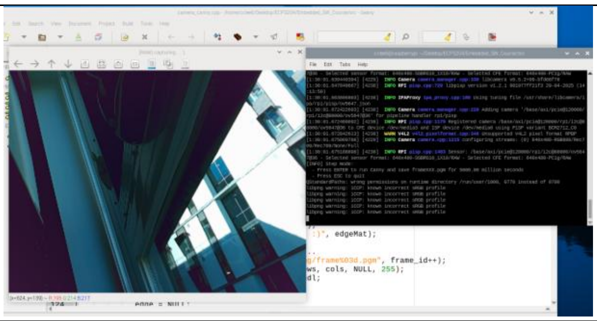
f.Convert captured frame to gray scale and push to canny for processing.

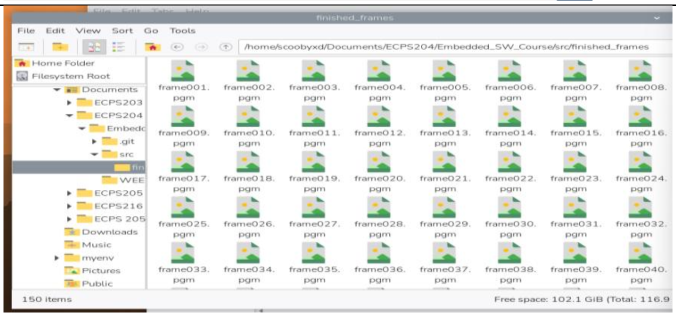
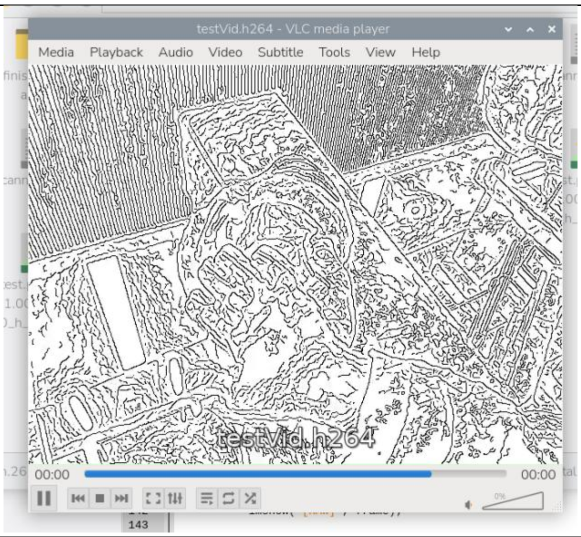
2. **Step 3:** Implement your code for processing multiple images and save in numerical order

   We modify the code to capture images in a roll for canny application. You can config to capture by number of frames or time in seconds.

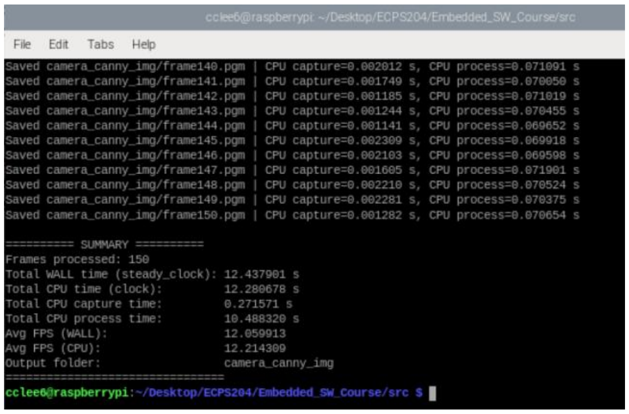
   Parameters are now: ./<source file><sigma><tlow><thigh><length/seconds><output folder>

Steps	Results	Comment
Parameter input ./<source file><sigma><tlow><thigh><length/seconds><output folder>	 `scoobyxd@ScoobyXD:~/Documents/ECPS204/Embedded_SW_Course/src $ ./camera_canny 1.0 0.6 0.2 n 150 finished_frames`	Can set n for frames or s for seconds

Application running	
	`:)", edgeMat);`
	`g/frame%03d.pgm", frame_id++);`
	`ws, cols, NULL, 255);`
	`dl;`
		Application initialized and showing camera views

Files exported to intended directory and files renamed to: frames###.pgm	 /home/scoobyxd/Documents/ECPS204/Embedded_SW_Course/src/finished_frames frame001.pgm frame002.pgm frame003.pgm frame004.pgm frame005.pgm frame006.pgm frame007.pgm frame008.pgm frame009.pgm frame010.pgm frame011.pgm frame012.pgm frame013.pgm frame014.pgm frame015.pgm frame016.pgm frame017.pgm frame018.pgm frame019.pgm frame020.pgm frame021.pgm frame022.pgm frame023.pgm frame024.pgm frame025.pgm frame026.pgm frame027.pgm frame028.pgm frame029.pgm frame030.pgm frame031.pgm frame032.pgm frame033.pgm frame034.pgm frame035.pgm frame036.pgm frame037.pgm frame038.pgm frame039.pgm frame040.pgm 150 items Free space: 102.1 GiB (Total: 116.9	All the files are now saved in the intended folder and renamed
Turn frames into video in .h264 format	 testVid.h264 - VLC media player	New video made

3. Step 4: Profiling the performance



```
cclee6@raspberrypi: ~/Desktop/ECPS204/Embedded_SW_Course/src
File  Edit  Tabs  Help
Saved camera_canny_img/frame140.pgm | CPU capture=0.002012 s, CPU process=0.071091 s
Saved camera_canny_img/frame141.pgm | CPU capture=0.001749 s, CPU process=0.070050 s
Saved camera_canny_img/frame142.pgm | CPU capture=0.001185 s, CPU process=0.071019 s
Saved camera_canny_img/frame143.pgm | CPU capture=0.001244 s, CPU process=0.070455 s
Saved camera_canny_img/frame144.pgm | CPU capture=0.001141 s, CPU process=0.069652 s
Saved camera_canny_img/frame145.pgm | CPU capture=0.002309 s, CPU process=0.069918 s
Saved camera_canny_img/frame146.pgm | CPU capture=0.002103 s, CPU process=0.069598 s
Saved camera_canny_img/frame147.pgm | CPU capture=0.001605 s, CPU process=0.071901 s
Saved camera_canny_img/frame148.pgm | CPU capture=0.002210 s, CPU process=0.070524 s
Saved camera_canny_img/frame149.pgm | CPU capture=0.002281 s, CPU process=0.070375 s
Saved camera_canny_img/frame150.pgm | CPU capture=0.001282 s, CPU process=0.070654 s

========== SUMMARY ==========
Frames processed: 150
Total WALL time (steady_clock): 12.437901 s
Total CPU time (clock):         12.280678 s
Total CPU capture time:         0.271571 s
Total CPU process time:         10.488320 s
Avg FPS (WALL):                 12.059913
Avg FPS (CPU):                  12.214309
Output folder:                  camera_canny_img
================================
cclee6@raspberrypi:~/Desktop/ECPS204/Embedded_SW_Course/src $ 
```

- ========== SUMMARY ==========
- Frames processed: 150
- Total WALL time (steady_clock): 12.437901 s
- Total CPU time (clock):   12.280678 s
- Total CPU capture time: 0.271571 s
- Total CPU process time: 10.488320 s
- Avg FPS (WALL):   12.059913
- Avg FPS (CPU):   12.214309
- Output folder: camera_canny_img
- ================================

4.1/2

We used <chrono> library to measure the Wall time.

The difference between the profiling is that the Wall time (real-world elapsed time) includes the process idle time, time waiting for I/O …etc. The clock() time is for only the process time itself. **So, the Wall time is expected to be a bit longer than the process time.**

4.3

Difference between WALL time and CPU time: 0.1573 sec

Difference between WALL FPS and CPU FPS: 0.1544 fps

Sometimes, the Wall time gets a bit shorter than the clock() cpu time. Our idea is that the OpenCV does multithreading, leading to the adding-up on the cpu time but the overall (Wall) time is shorter.

4.4

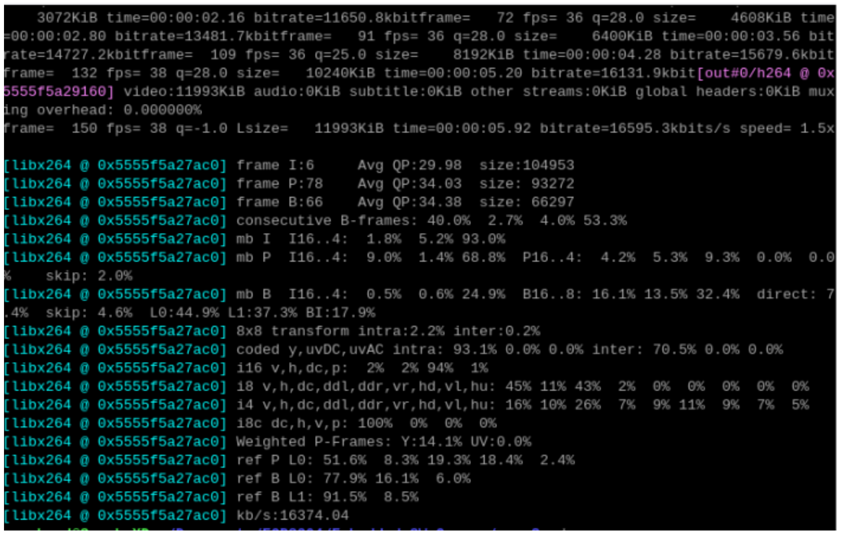
After Adding the "time" command infront of the execution, we have the following markings showing. "time" is a shell tool to do the profiling; real stands for the WALL time, user stand for the CPU time and sys stand for the kernel time.

- **real 0m13.038s**
- **user 0m12.337s**
- **sys   0m0.594s**

They are not matching our measurement, we'll discuss on it in the discussion part.

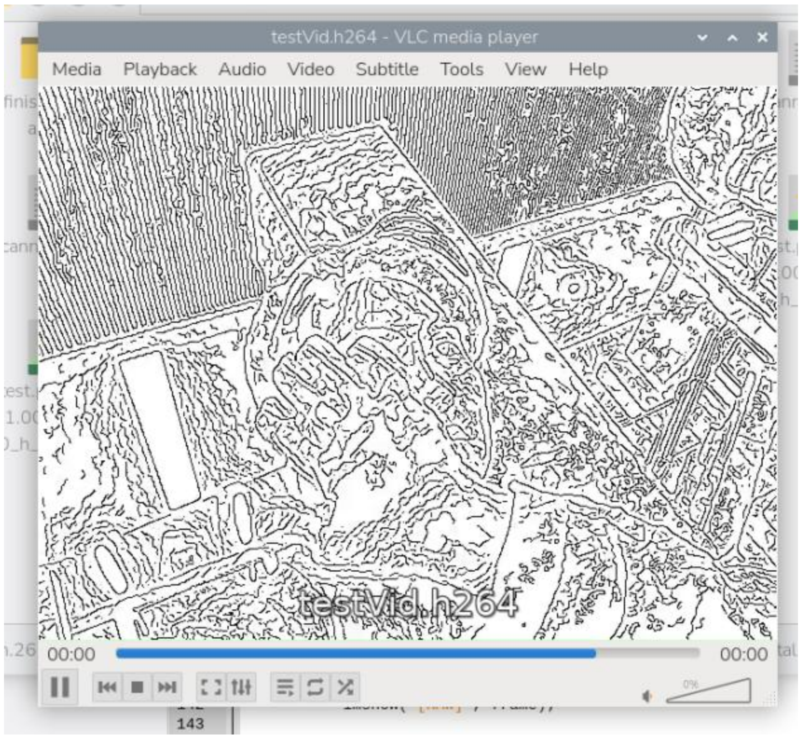
4. Step 5: Encode your frames into a video

The video is encoded as below.



```
    3072KiB time=00:00:02.16 bitrate=11650.8kbitframe=   72 fps= 36 q=28.0 size=    4608KiB time
=00:00:02.80 bitrate=13481.7kbitframe=   91 fps= 36 q=28.0 size=    6400KiB time=00:00:03.56 bit
rate=14727.2kbitframe=  109 fps= 36 q=25.0 size=    8192KiB time=00:00:04.28 bitrate=15679.6kbit
frame=  132 fps= 38 q=28.0 size=   10240KiB time=00:00:05.20 bitrate=16131.9kbit[out#0/h264 @ 0x
5555f5a29160] video:11993KiB audio:0KiB subtitle:0KiB other streams:0KiB global headers:0KiB mux
ing overhead: 0.000000%
frame=  150 fps= 38 q=-1.0 Lsize=   11993KiB time=00:00:05.92 bitrate=16595.3kbits/s speed= 1.5x

[libx264 @ 0x5555f5a27ac0] frame I:6     Avg QP:29.98  size:104953
[libx264 @ 0x5555f5a27ac0] frame P:78    Avg QP:34.03  size: 93272
[libx264 @ 0x5555f5a27ac0] frame B:66    Avg QP:34.38  size: 66297
[libx264 @ 0x5555f5a27ac0] consecutive B-frames: 40.0%  2.7%  4.0% 53.3%
[libx264 @ 0x5555f5a27ac0] mb I  I16..4:  1.8%  5.2% 93.0%
[libx264 @ 0x5555f5a27ac0] mb P  I16..4:  9.0%  1.4% 68.8%  P16..4:  4.2%  5.3%  9.3%  0.0%  0.0
%    skip: 2.0%
[libx264 @ 0x5555f5a27ac0] mb B  I16..4:  0.5%  0.6% 24.9%  B16..8: 16.1% 13.5% 32.4%  direct: 7
.4%  skip: 4.6%  L0:44.9% L1:37.3% BI:17.9%
[libx264 @ 0x5555f5a27ac0] 8x8 transform intra:2.2% inter:0.2%
[libx264 @ 0x5555f5a27ac0] coded y,uvDC,uvAC intra: 93.1% 0.0% 0.0% inter: 70.5% 0.0% 0.0%
[libx264 @ 0x5555f5a27ac0] i16 v,h,dc,p:  2%  2% 94%  1%
[libx264 @ 0x5555f5a27ac0] i8 v,h,dc,ddl,ddr,vr,hd,vl,hu: 45% 11% 43%  2%  0%  0%  0%  0%  0%
[libx264 @ 0x5555f5a27ac0] i4 v,h,dc,ddl,ddr,vr,hd,vl,hu: 16% 10% 26%  7%  9% 11%  9%  7%  5%
[libx264 @ 0x5555f5a27ac0] i8c dc,h,v,p: 100%  0%  0%  0%
[libx264 @ 0x5555f5a27ac0] Weighted P-Frames: Y:14.1% UV:0.0%
[libx264 @ 0x5555f5a27ac0] ref P L0: 51.6%  8.3% 19.3% 18.4%  2.4%
[libx264 @ 0x5555f5a27ac0] ref B L0: 77.9% 16.1%  6.0%
[libx264 @ 0x5555f5a27ac0] ref B L1: 91.5%  8.5%
[libx264 @ 0x5555f5a27ac0] kb/s:16374.04
```


testVid.h264 - VLC media player
Media Playback Audio Video Subtitle Tools View Help
testVid.h264
00:00
00:00

3. Problems and Discussion (6 pts)

- Difference between measured ALL time and CPU time. We think it’s because OpenCV is using multithreading, leading to the result.
- The shell time tool result being different from our measured WALL and CPU time. We think it’s because our implementation is ignoring the top and bottom parts of the program, including the initialization and profiling parsing time.

4. Conclusion (2 pts)

- In conclusion we set up the Raspberry Pi, OpenCV, made sure it ran canny, and that the camera works as intended with it. We used frames from the camera, split it, measured timing and renamed the frames, and then pieced it back together into a video.