ECPS204 Project1-Setup

CHENG-CHIH, LEE, 29329351
JONATHAN KIM, 42411903

Project Description (2 pts)

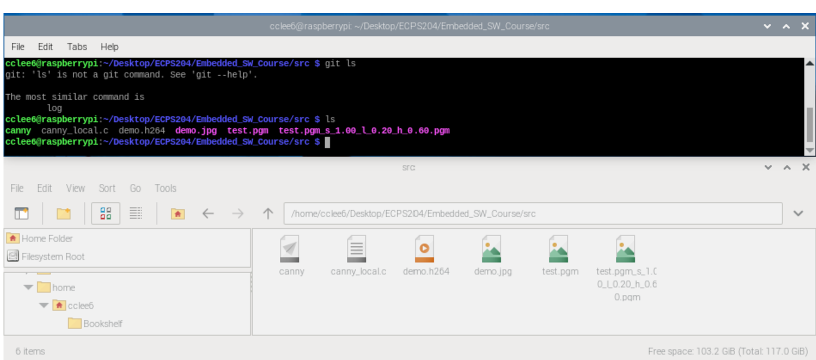
● PartA

- The purpose of this project is to set up Raspberry Pi with the camera and canny.c compilation environment. Furthermore, we want to test different results of canny.exe with at least 4 sets of parameters input.
- We expect to see the directory of the files used and captured images and videos. Furthermore, we expect to see the difference between each trail with comments prepared.

1. Experimental Setup (4 pts)

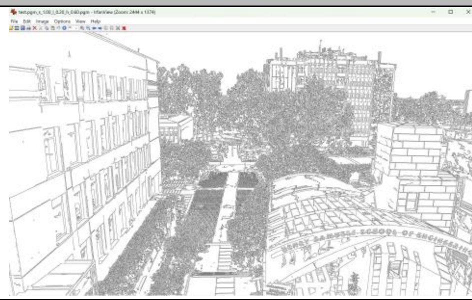
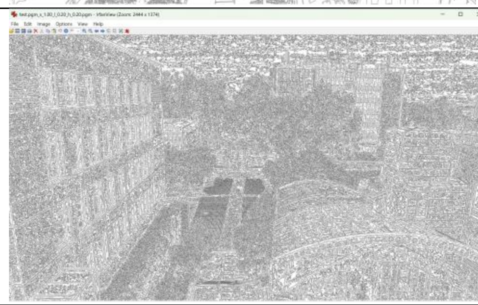
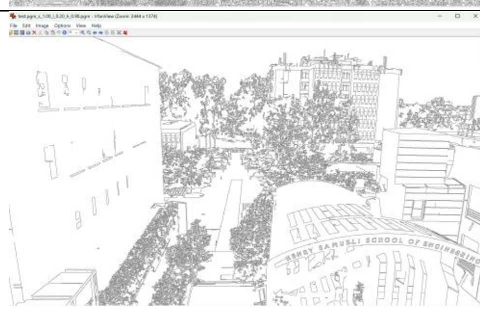
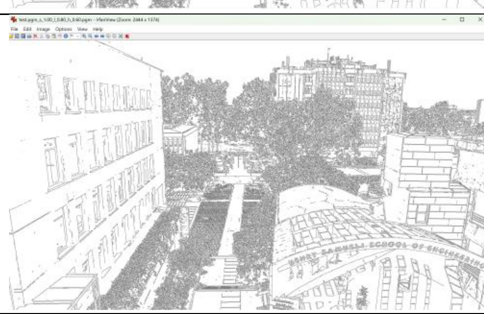
- For this project, we set up a Git repository to exchange files between my RPi and laptop. Following the instructions, I get each step done and recorded.
- We had to use libcamera to do simple camera control since we're using RPi 5 with newer SW.
- In the canny parameters' comparison part, we choose to change one parameter at a time to control the changes.

2. Results (6 pts)



This is the directory and files we used for this project. On the right-hand side is the captured image.

Canny application with different parameters and comparison.

Parameter Sets <sigma>/<tlow>/<thigh>	**Results**	**Comment and observation**
1.0/0.2/0.6		Example Command
1.0/0.2/0.2		By tuning **<thigh>** down to 0.2, there are too much noise kept.
1.0/0.2/0.9		By tuning **<thigh>** up to 0.9, there are way less noise kept.
1.0/0.8/0.6		By tuning **<tlow>** up to 0.8, this set is probably not making sense. Since the lower threshold is now higher than higher threshold.
2.0/0.2/0.6		By tuning **<sigma>** up to 2, edges become clearer and simpler (strong edges).

3. Problems and Discussion (6 pts)

- First, we were a little stuck by the camera library but that was an easy fix by switching to libcamera.
- Setting up the repository took us a little bit of time since we would need the passkey to operate git access on the RPi.
- Other parts are fine and clear.

4. Conclusion (2 pts)

- After setting up the environment and RPi, the camera functioned as expected.
- After testing canny with different parameters, we came up with the conclusion that the <sigma> defines the magnitude of smoothing; <tlow> defines the lower threshold to remove an edge; <thigh> defines the higher threshold to keep an edge.

● PartB

- 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)

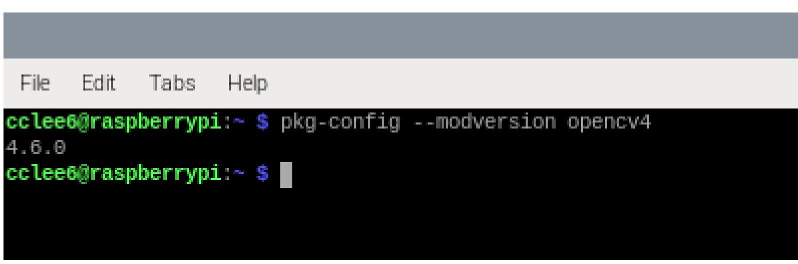
- For this project, we set up a Git repository to exchange files between my RPi and laptop. Following the instructions, we get each step done and recorded.
- We had to use libcamera to do simple camera control since we're using RPi 5 with newer SW.
- Everything is based on PartA

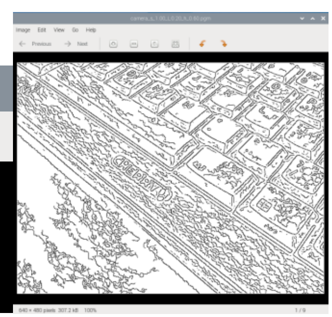
2. Results (6 pts)

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.





   - 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

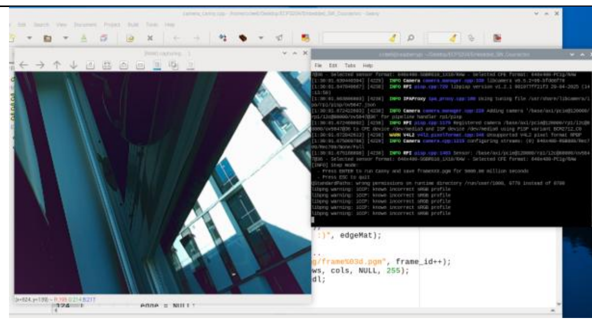
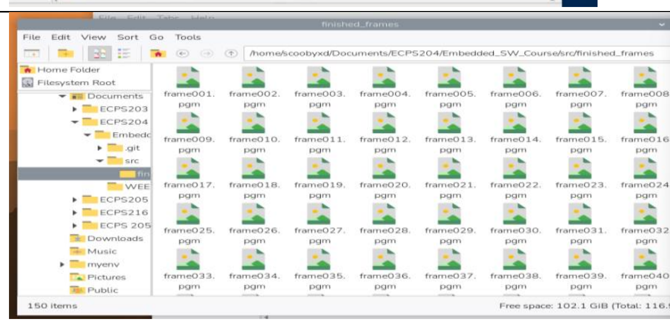
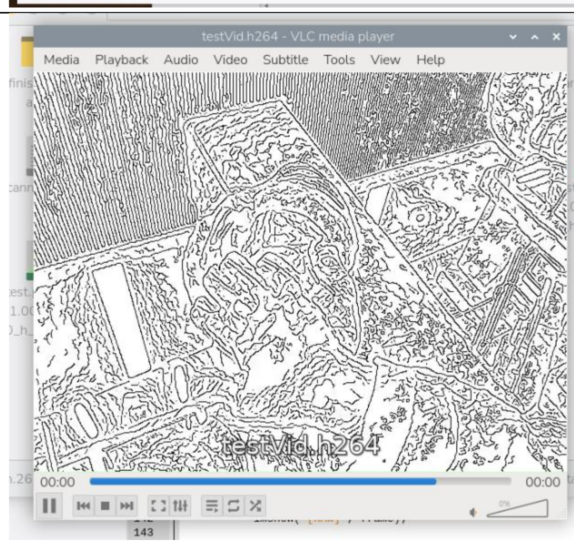
e.for loop to show camera images until “Enter” key is pressed

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

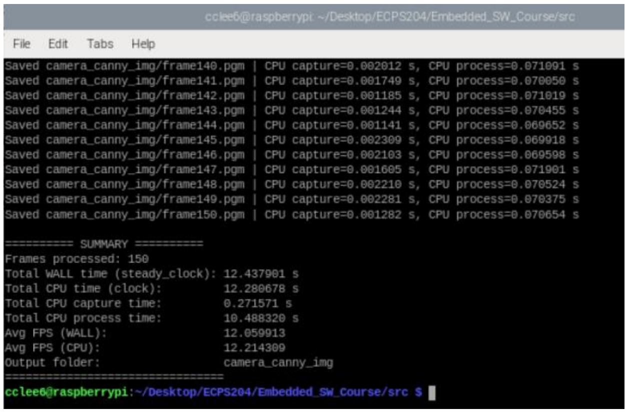
2. **Step 3:** Implement your code for processing multiple images and save in numerical order

   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>		Can set n for frames or s for seconds
Application running		Application initialized and showing camera views
Files exported to intended directory and files renamed to: frames###.pgm		All the files are now saved in the intended folder and renamed
Turn frames into video in .h264 format		New video made

3. Step 4: Profiling the performance



- ========== 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

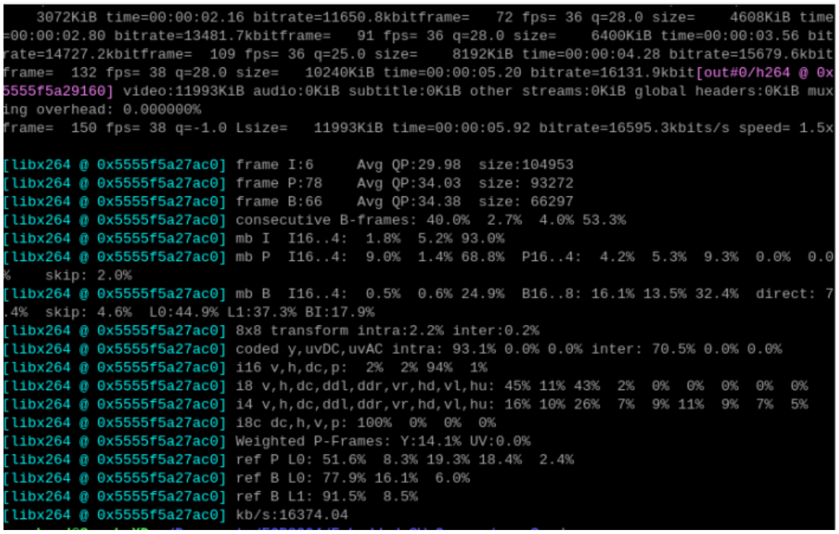
After Adding the "time" command infront of the execution, we have the following markings showing. "time" is a shell tool to do the profiling; real stands for the WALL time, user stand for the CPU time and sys stand for the kernel time.

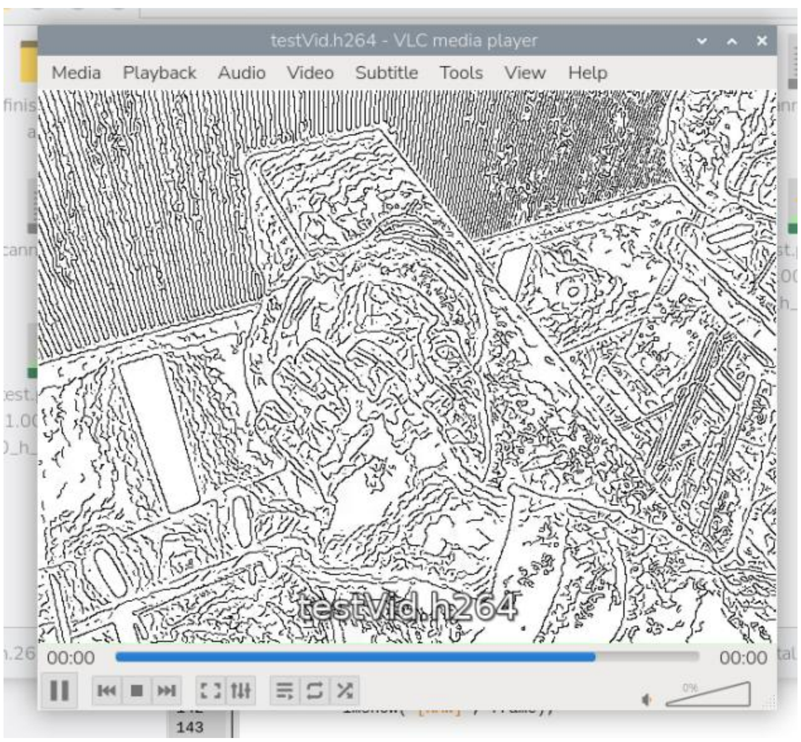
- **real 0m13.038s**
- **user 0m12.337s**
- **sys 0m0.594s**

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 below.





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.