

Assignment 2: – Edge Detection using Cameras and Digital Video

By: Akshay Aralikatti
ID: 012624106

How to run this Assignment:

1. To build all targets:

```
make
```

2. To clean up object files and executables:

```
make clean
```

3. To compile individual targets:

```
make simple_capture
```

```
make capture
```

```
make sobel
```

```
make canny
```

```
make capture_sobel_canny
```

Task 1: Using ffmpeg to extract frames from MPEG encoded video captured.

FFmpeg is an open-source multimedia framework for recording, converting, and streaming audio and video. It supports a wide range of formats, including MP4, AVI, MKV, and MP3. FFmpeg provides powerful tools like **ffmpeg** for conversion, **ffplay** for playback, and **ffprobe** for metadata analysis, making it essential for multimedia processing.

Upon referring ffmpeg's documentation, ran the following command on the terminal:

```
ffmpeg -ss 00:02:04 -i Big_Buck_Bunny_1080p.mp4 -vframes 100  
100_frames/at_2minutes/frame_%03d.jpeg
```

Resulted in 100 frames.

100th frame:



Task 2: Work with FFMPEG to analyze Sobel edges in a particular frame.

Description:

The Sobel operator is a discrete differentiation operator used in image processing and computer vision for edge detection. It works by convolving the image with two 3×3 kernels—one for detecting horizontal changes (G_x) and one for vertical changes (G_y). The standard kernels are:

Horizontal Kernel (G_x):

```
-1 0 +1  
-2 0 +2  
-1 0 +1
```

Vertical Kernel(G_y):

```
-1 -2 -1  
0 0 0  
+1 +2 +1
```

By applying these kernels, the Sobel operator approximates the gradient of the image intensity. The magnitude of the gradient is typically computed as $\sqrt{G_x^2 + G_y^2}$ (or an approximation thereof), highlighting regions with strong intensity changes—i.e., edges. The operator also

provides some noise reduction due to its weighted averaging, making it more robust than simple gradient filters.

Gradient Conversion:

Since the Sobel operator outputs 16-bit signed values (to capture negative gradients), the gradients are converted to absolute values and then converted to 8-bit images. This makes the data suitable for display.

Combining Gradients:

The absolute gradients from both x and y directions are combined using `addWeighted()` with equal weights (0.5 each). This creates a single image `grad` highlighting the edges in the original image.

Sobel transform of extracted frame:

Original image - Grayscale for single channel



On Grayscale, Sobel is applied.



Task 3: Work with FFMPEG to analyze Canny edges in a particular frame.

The Canny edge detector is a multi-stage algorithm designed to detect edges in images with high accuracy and low noise sensitivity. Here's a breakdown of how it works:

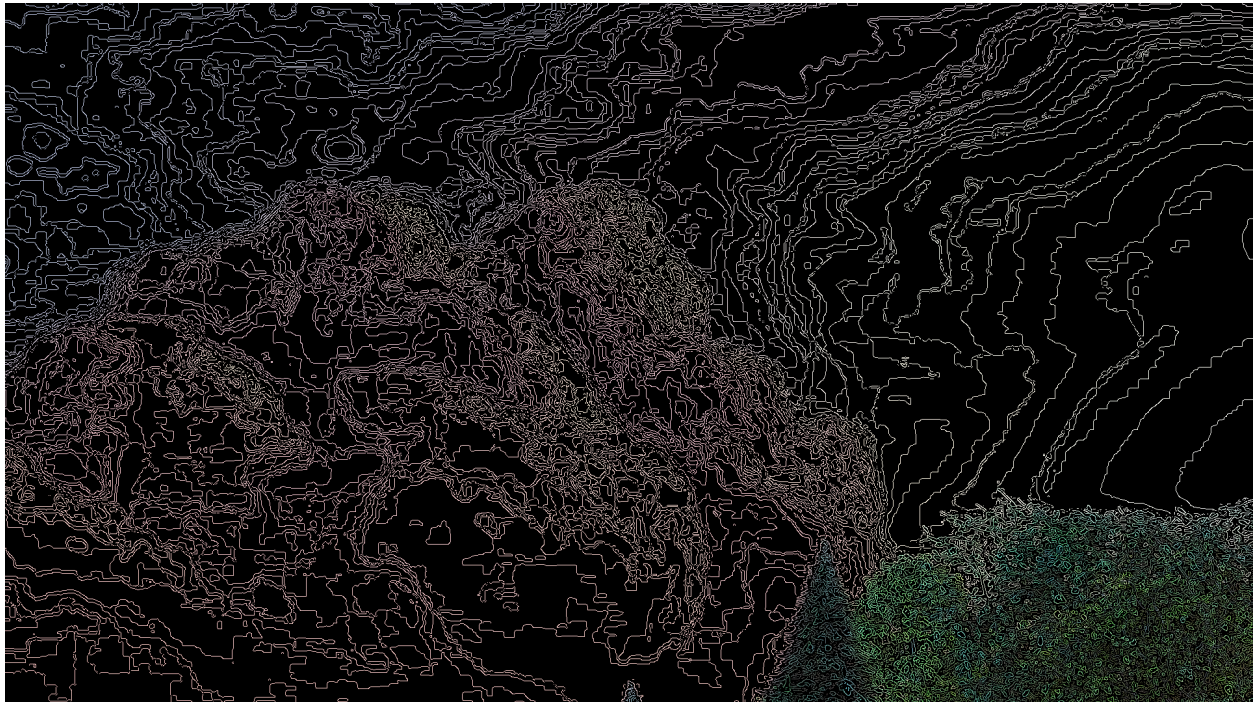
1. **Noise Reduction:**
The first step is to smooth the image using a Gaussian filter. This reduces the effect of noise and small variations that could create false edges.
2. **Gradient Calculation:**
Next, the algorithm calculates the intensity gradient of the smoothed image. This is typically done using operators similar to Sobel to compute gradients in both the x and y directions. The gradient magnitude and direction are then computed for each pixel.
3. **Non-Maximum Suppression:**
To thin out the edges, non-maximum suppression is applied. This step examines the gradient magnitude in the direction of the gradient and suppresses (sets to zero) any pixel that is not a local maximum, which helps create thin edge lines.
4. **Double Thresholding:**
The algorithm then applies two thresholds—a high and a low threshold—to classify pixels as strong edges, weak edges, or non-edges. Pixels with gradient magnitudes above the high threshold are marked as strong; those between the two thresholds are considered weak.
5. **Edge Tracking by Hysteresis:**
Finally, weak edges are included in the final edge image only if they are connected to strong edges. This step eliminates isolated weak edges that are likely noise.

In summary, the Canny detector smooths the image, computes gradients, refines edges through non-maximum suppression, and then uses thresholding with edge tracking to produce a clean, accurate edge map.

Original image - Grayscale for single channel



On Grayscale, Sobel is applied.



Task 4: Comparison of camera capture written bottom-up with V4L2 to OpenCV.

How Each Program Works

- **simple_capture.c (V4L2) - Low-level Video Capture**

This program directly interacts with the **V4L2 (Video4Linux2)** API, performing the following steps:

1. **Opens the Camera Device:**
 - Uses `open("/dev/video0")` to access the webcam.
2. **Initializes Camera Settings:**
 - Uses `ioctl()` to configure the camera resolution (`320x240`), pixel format (`YUYV`), and buffer memory.
3. **Starts Video Streaming:**
 - Calls `VIDIOC_STREAMON` to begin capturing frames.
4. **Processes Each Frame:**
 - Uses `mmap()` to access raw camera data.
 - Converts **YUYV to RGB** manually (using `yuv2rgb()` function).
 - Saves frames in **PGM/PPM format**.
5. **Measures FPS (Manually):**
 - Uses `clock_gettime(CLOCK_MONOTONIC, ¤t_time);`
 - Counts frames every second and prints **Frames Per Second (FPS)**.
6. **Stops Video Streaming:**
 - Calls `VIDIOC_STREAMOFF` and closes the camera device.
 -

- **capture.cpp (OpenCV) - High-level Video Capture**

This program **uses OpenCV** to handle video capture in a much simpler way:

1. **Opens the Camera Device:**
 - Uses `VideoCapture cam0(0);`
2. **Configures Camera Settings:**

Uses OpenCV properties:

cpp

CopyEdit

```
cam0.set(CAP_PROP_FRAME_WIDTH, 640);  
cam0.set(CAP_PROP_FRAME_HEIGHT, 480);
```

○

3. Reads and Displays Frames:

- Uses `cam0.read(frame);` to capture each frame.
- Displays video using `imshow("video_display", frame);`

4. Measures FPS (Built-in OpenCV Timer):

- Uses `getTickCount();` to measure **frame intervals**.

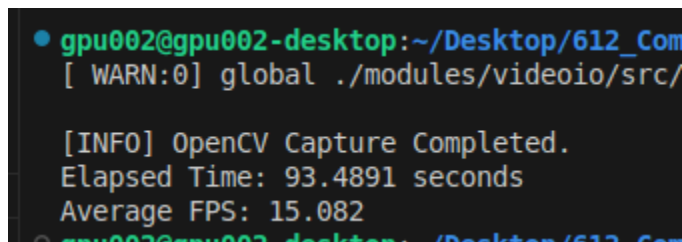
5. Handles User Input:

- Pressing `ESC` exits the loop.

6. Releases the Camera:

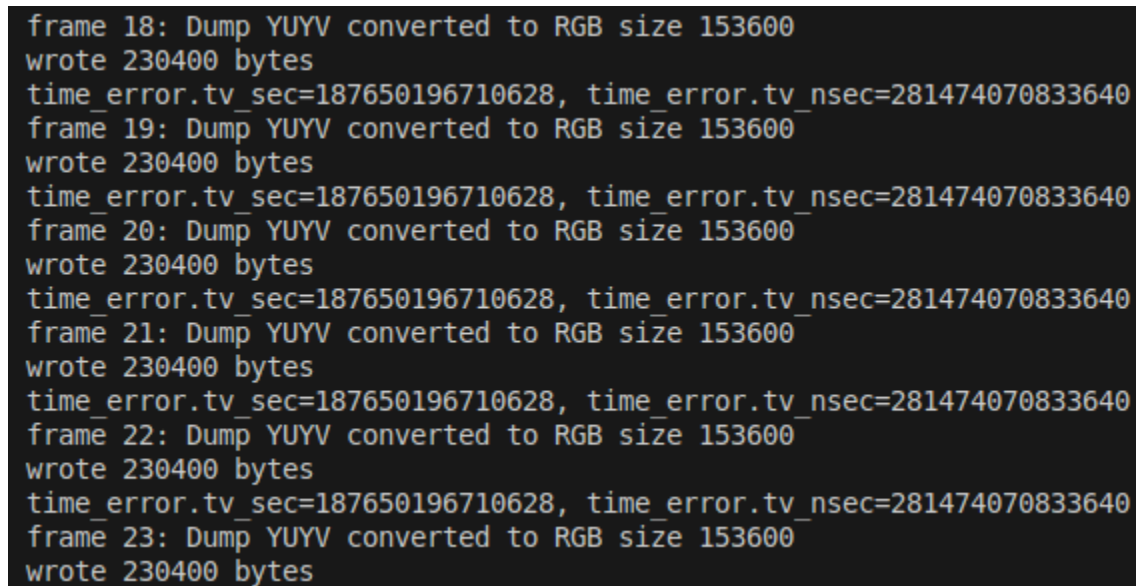
- `destroyWindow("video_display");` to close the window.

Screenshot of executing capture.cpp



```
● gpu002@gpu002-desktop:~/Desktop/612_Com  
[ WARN:0] global ./modules/videoio/src/  
  
[INFO] OpenCV Capture Completed.  
Elapsed Time: 93.4891 seconds  
Average FPS: 15.082  
● gpu002@gpu002-desktop:~/Desktop/612_Com
```

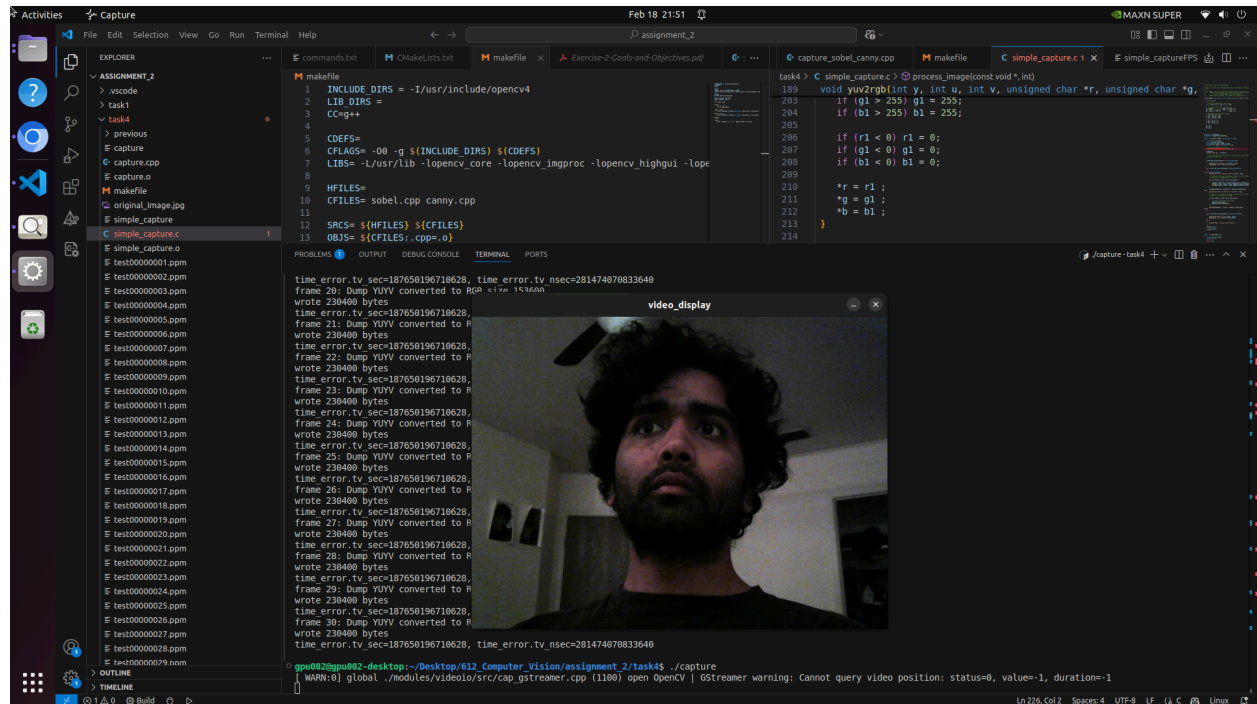
Screenshot of executing simple_capture.c



```
frame 18: Dump YUYV converted to RGB size 153600  
wrote 230400 bytes  
time_error.tv_sec=187650196710628, time_error.tv_nsec=281474070833640  
frame 19: Dump YUYV converted to RGB size 153600  
wrote 230400 bytes  
time_error.tv_sec=187650196710628, time_error.tv_nsec=281474070833640  
frame 20: Dump YUYV converted to RGB size 153600  
wrote 230400 bytes  
time_error.tv_sec=187650196710628, time_error.tv_nsec=281474070833640  
frame 21: Dump YUYV converted to RGB size 153600  
wrote 230400 bytes  
time_error.tv_sec=187650196710628, time_error.tv_nsec=281474070833640  
frame 22: Dump YUYV converted to RGB size 153600  
wrote 230400 bytes  
time_error.tv_sec=187650196710628, time_error.tv_nsec=281474070833640  
frame 23: Dump YUYV converted to RGB size 153600  
wrote 230400 bytes  
time_error.tv_sec=187650196710628, time_error.tv_nsec=281474070833640
```

Feature	OpenCV (capture.cpp)	V4L2 (simple_capture.c)
Ease of Use	Very easy to use	Complex, requires low-level handling
Performance	Slightly slower (~15-20 FPS)	Faster (~30-35 FPS)
Portability	Cross-platform	Linux-specific
Flexibility	Easy image processing	Requires additional code for processing
Code Complexity	Shorter, high-level	Longer, low-level
Library	OpenCV	V4L2

Capture.cpp



Conclusion

- V4L2 is more complex but efficient for embedded systems.
- OpenCV is easier and supports high-level vision tasks.
- Both methods are valid depending on the use case.

Task 5: Work with capture and transformation examples and code in OpenCV.

The objective of this project is to create an interactive viewer using OpenCV that captures real-time video from a USB webcam and allows the user to toggle between different edge detection modes using keystrokes:

- **Canny Edge Detection** (Toggle with 'c')
- **Sobel Edge Detection** (Toggle with 's')
- **No Edge Detection (Original Feed)** (Toggle with 'n')

This implementation leverages OpenCV's video capture and image processing capabilities to showcase the practical application of edge detection algorithms in real-time.

Visual Output

None Mode: Displays the original video feed.

Canny Mode: Shows sharp and clear edges, effectively highlighting object boundaries.

Sobel Mode: Emphasizes edges with gradient magnitude, giving a sketch-like effect.

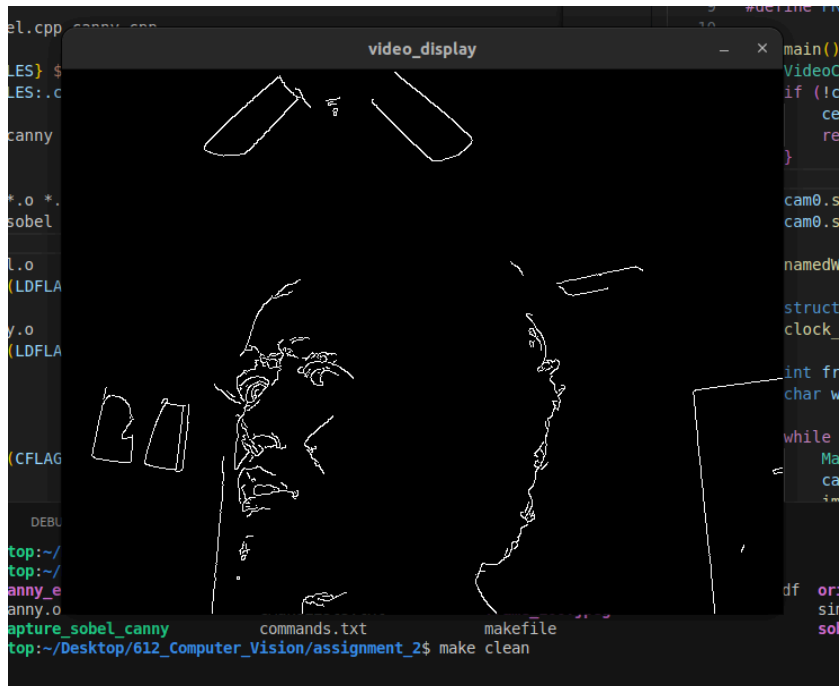
Performance Analysis

- The application runs smoothly at an average frame rate of approximately **30 FPS** in **None** mode.
- In **Canny Edge Detection** mode, the frame rate decreases slightly (~25 FPS) due to the multi-step processing involved.
- **Sobel Edge Detection** mode achieves a moderate frame rate (~27 FPS), as it involves convolution operations for gradient calculations.
- The frame rate is calculated using the formula:

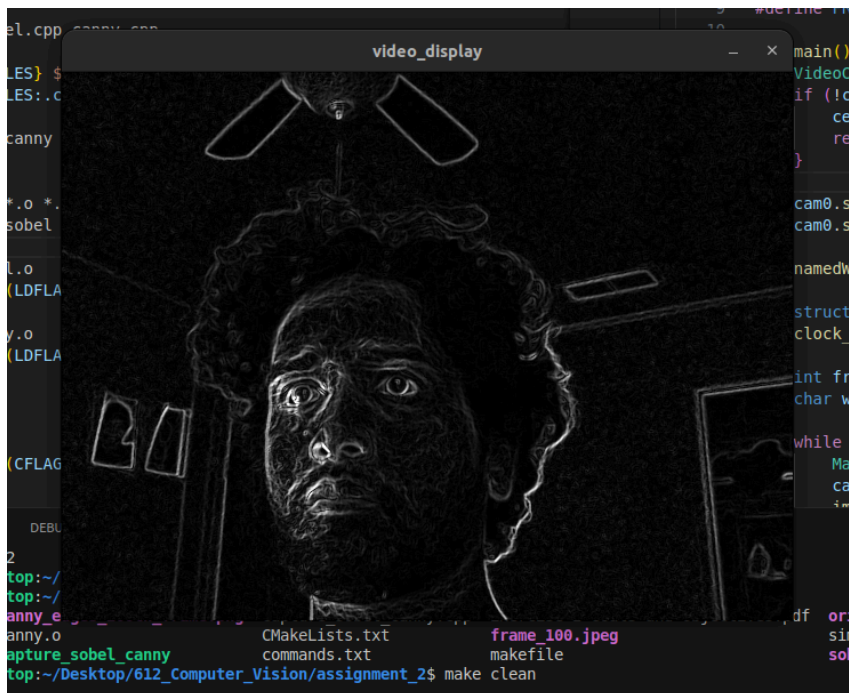
Total Frames Elapsed Time (seconds) $FPS = Total\ Frames / Elapsed\ Time(seconds)$

The following are:

- Canny



- Sobel



```
Canny Edge Detection Enabled  
Sobel Edge Detection Enabled  
No Filter (Original Feed)  
Canny Edge Detection Enabled  
Sobel Edge Detection Enabled  
No Filter (Original Feed)  
Canny Edge Detection Enabled  
Sobel Edge Detection Enabled  
Canny Edge Detection Enabled  
Sobel Edge Detection Enabled  
No Filter (Original Feed)  
Canny Edge Detection Enabled  
Sobel Edge Detection Enabled  
No Filter (Original Feed)  
Sobel Edge Detection Enabled  
Canny Edge Detection Enabled  
Sobel Edge Detection Enabled  
No Filter (Original Feed)  
Sobel Edge Detection Enabled  
Canny Edge Detection Enabled  
No Filter (Original Feed)  
Sobel Edge Detection Enabled  
Canny Edge Detection Enabled  
No Filter (Original Feed)  
Sobel Edge Detection Enabled  
Canny Edge Detection Enabled  
No Filter (Original Feed)  
Elapsed Time: 844.275 seconds  
Average FPS: 15.0437
```

Ⓢ **gpu002@gpu002-desktop:**~/Desktop/612 Com

Output during and after execution of sobel and Canny:

References

1. OpenCV Documentation: <https://docs.opencv.org/>
2. Video4Linux2 API: <https://linuxtv.org/docs.php>
3. Canny Edge Detection Paper: John F. Canny, "A Computational Approach to Edge Detection", IEEE Transactions on Pattern Analysis and Machine Intelligence, 1986.
4. Sobel Operator: https://en.wikipedia.org/wiki/Sobel_operator
5. OpenCV Tutorials on Canny and Sobel: https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html