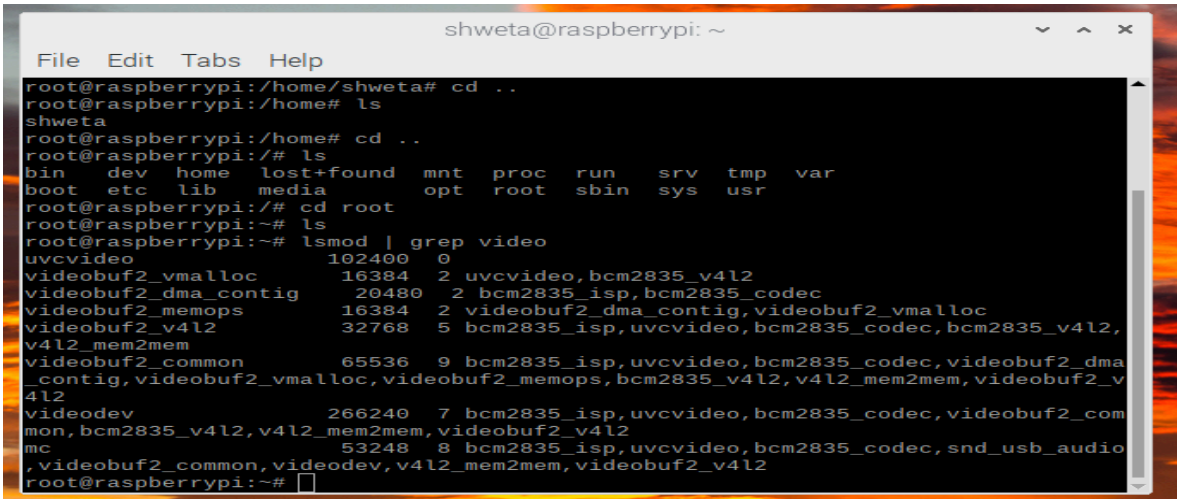
ECEN 5623 - RTES

EXERCISE 4

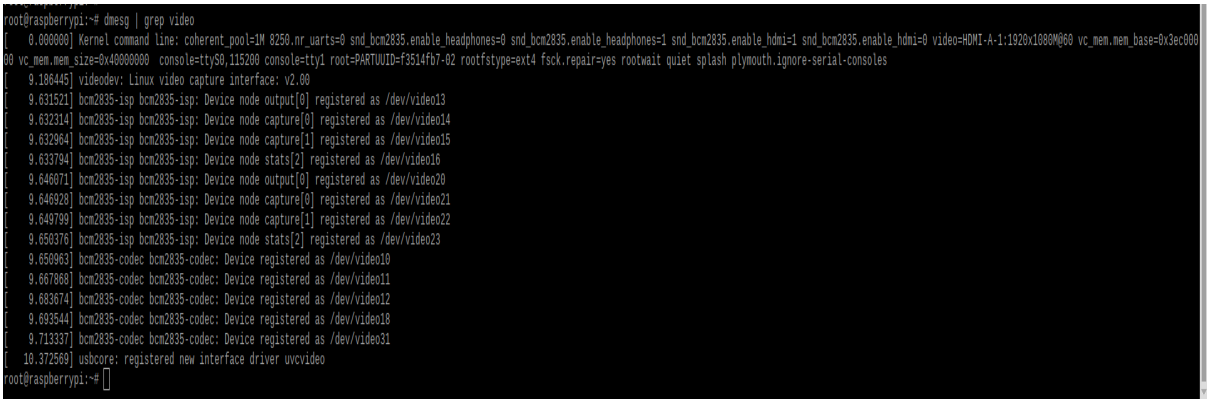
–Shweta Prasad and Shruthi Thallapally

Problem 1:Camera Driver installation on Raspberry Pi for using the Camera.

The output of lsmod | grep video on Raspberry Pi 3B+ with uvc driver installed:

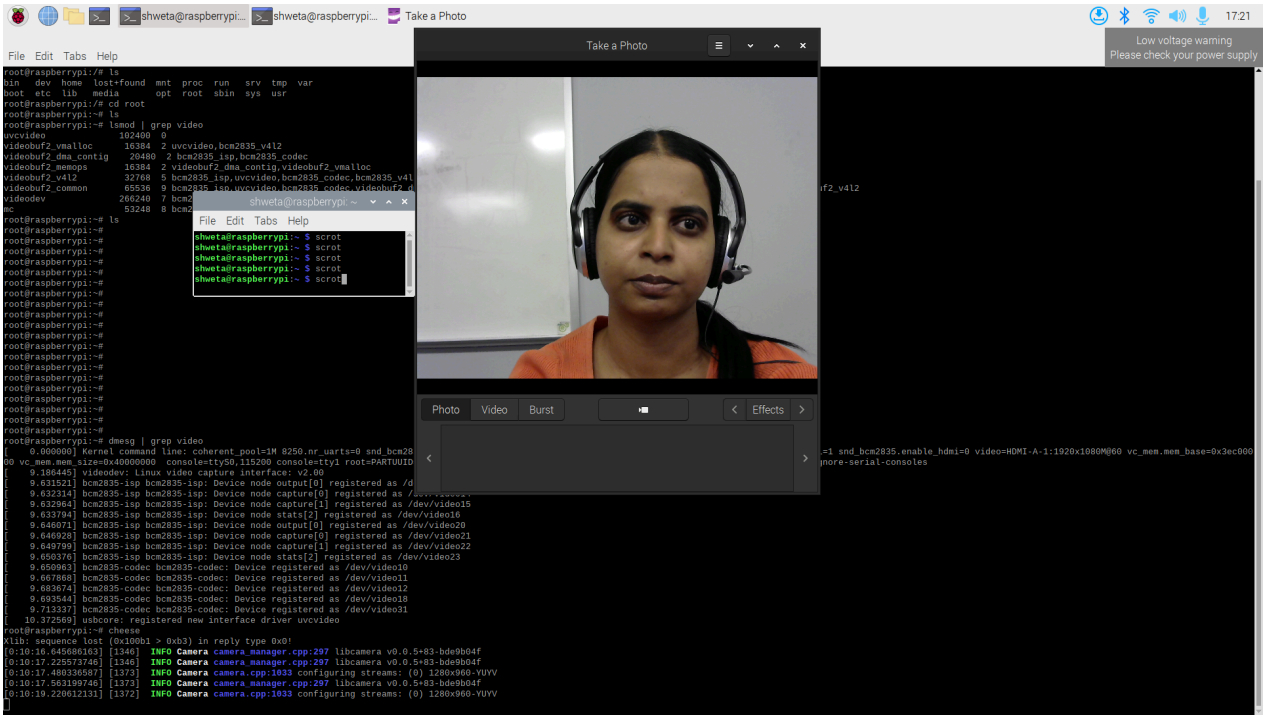


The output of dmesg | grep video showing Kernel command line hdmi camera on Raspberry Pi 3B+:



Problem 2: Option 2: Cheese– Camera Capture:

The following is the screenshot of cheese camera capture application: Evidence of image capture



Problem 3:(a): Computer vision or (b) Capture raw image buffer: Replace read file with camera Canny can and interactive code.

3) Using your verified Logitech C2xx camera on a DE1-SoC, Raspberry Pi or Jetson, verify that it can stream continuously to a raw image buffer for transformation and processing using any one of the below methods, either a) or b):

a) Use example code from the computer-vision or computer_vision_cv3_tested folder such as simple-capture, simpler-capture, or simpler-capture-2. Read the code and modify the device that is opened if necessary to get this to work. Provide a screen shot to prove that you got continuous capture to work. Note that simpler capture requires installation of OpenCV on your DE1-SoC, Raspberry Pi, Jetson, or native Linux system. For the Jetson Nano or TK1 OpenCV will likely already be available on your board, but if not, please

follow simple instructions found here to install openCV [the "Option 2, Building the public OpenCV library from source" is the recommended approach with – DWITH_CUDA=OFF. Don't install CUDA and please leave it off when you build OpenCV.] For the DE1-SoC please find the files soc_system.rbf and SettingUp.pdf on Canvas. They contain instructions for setting up board and installing OpenCV. The TAs have set up using these in the past and were able to complete exercise 4 requirements. The cmake command for opencv installation has been changed so that it works on the board too. Alternatively, you can use OpenCV port found here: http://rocketboards.org/foswiki/view/Projects/OpenCVPort. If you have trouble getting OpenCV to work on your board, try running it on your laptop under Linux first. You can use OpenCV install, OpenCV with Python bindings for 2.x and 3.x for this.

Ans:

OpenCV is an open-source computer vision and machine learning software library. It provides functionalities such as object detection and recognition, video processing, camera calibration, etc. Open source Computer Vision Library is developed by Intel and supports multiple programming languages like C++, Python, Java and MATLAB. It ia a powerful resource in various domains including medical imaging, autonomous vehicles, augmented reality, robotics and many more.

Brief description of code and APIs:

Simpler_capture4 code continuously captures frames from the camera connected to the RPi3 B+ and displays the frames in a window named "video_display".

VideoCapture:

VideoCapture class from OpenCV is used to capture video frames from camera or a video file. In simpler_capture4 code, index '0' is passed which means taking input from the first connected camera.

namedWindow():

namedWindow function creates a window with the name specified for displaying the captured frames from images or video.

Mat:

Mat class represents n-dimensional numerical single-channel or multi-channal array. This matrix is used to hold the captured frame from the video.

cam0.isOpened():

The method cam0.isOpened() checks if the video capture device is opened successfully. If it is not opened it exits the program due to error.

cam0.set():

cam0.set() method sets theproperties of the capture device such as dimensions of the frame such as width and height.

cam0.read():

cam0.read() method reads the next vidoe frame from the device capturing video and stores it in the matrix.

imshow():

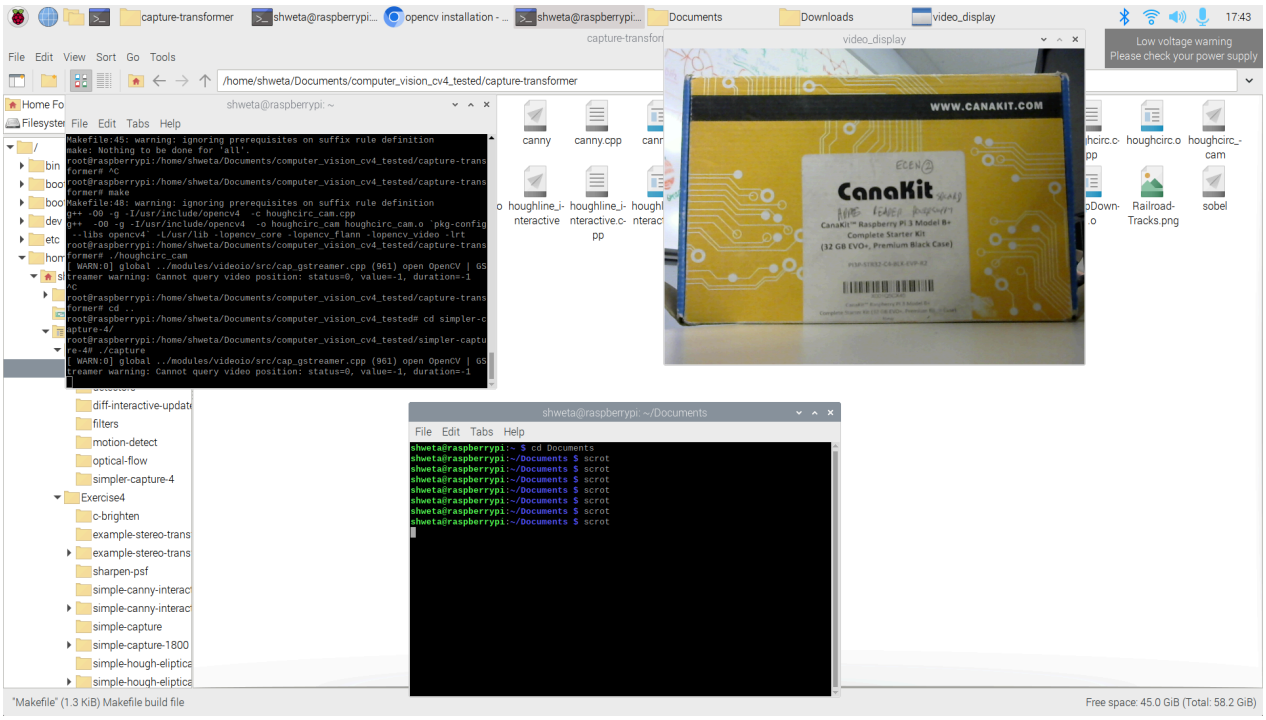
The imshow() function displays the captured frame in the specified window.

waitKey():

The waitKey functions keeps executing the previous function until a keyboard event occurs. It is used to wait for the user input.

destroyWindow():

destroyWindow function destroys the window specified and releases the resources associated with the window.



Problem 4:(a): Continuous transformation OpenCV or (b) Capturing 1800 frames: One of first three. Two transforms. 3.4.1

Question 4: Make them interactive all output. OpenCV vr4 an OpenCV vr3 in the folder

4) a) Choose a continuous transformation OpenCV example from computer vision cv3 tested such as the canny-interactive, hough-interactive, hough-eliptical-interactive, or stereo-transformimpoved . Show a screen shot to prove you built and ran the code. Provide a detailed explanation of the code and research uses for the continuous transformation by looking up API functions in the OpenCV manual (http://docs.opencv.org) and for stereo-transform-improved either implement or explain how you could make this work continuously rather than snapshot only.

Ans:

Canny edge detection algorithm:

The Canny edge detection algorithm effectively detects edges in images while suppressing noise and producing accurate and continuos edge contours. Canny edge detection algorithm involves gaussian smoothing, gradient calculation, non-maximum suppression, double thresholding and edge tracking by hysteresis. It is widely used in applications like object detection,feature extractionand image segmentation in computer visionand image processing.

Brief description and flow of code:

CannyThreshold():

This is a callback function used to perform Canny edge detection algorithm on the captured image form the camera and displays the resulting edge map. First captured input frame is converted into grayscale usig ‘COLOR_BGR2GRAY’. Then the noise in the frame is reduced with gaussian blur ‘blur()’. Canny edge detector ‘Canny()’ is applied to the frame with reduced noise and wrestling edge map us displayed using ‘imshow()’.

Mat:

Mat class represents n-dimensional numerical single-channel or multi-channal array. This matrix is used to hold the captured frame from the video.

blur():

The blur function reduces the noise in the input image by applying gaussian blur.

canny():

The canny function performs canny edge detection algorithm on input image frame.

imshow():

The imshow() function displays the captured frame in the specified window.

main():

The main function initalizes the video capture device using ‘VideoCapture’. A new window is created for displaying the video captured using ‘namedWindow’. After checking if the camera is opened successfully, sets the frame width and height for the captured video. Enters an infinite loop to capture the frames from the camera. The captured frames are displayed in the created window. A trackbar is created using ‘createTrackbar to adjust minimum threshold for the canny edge detection algorithm. ‘CannyThreshold() function is called to perform the detection. It waits for the user input from keyboard to exit the program. When the user input is detected, the display window is destroyed and program ends.

waitKey():

The waitKey functions keeps executing the previous function until a keyboard event occurs. It is used to wait for the user input.

destroyWindow():

destroyWindow function destroys the window specified and releases the resources associated with the window.

VideoCapture:

VideoCapture class from OpenCV is used to capture video frames from camera or a video file. In simpler_capture4 code, index ‘0’ is passed which means taking input from the first connected camera.

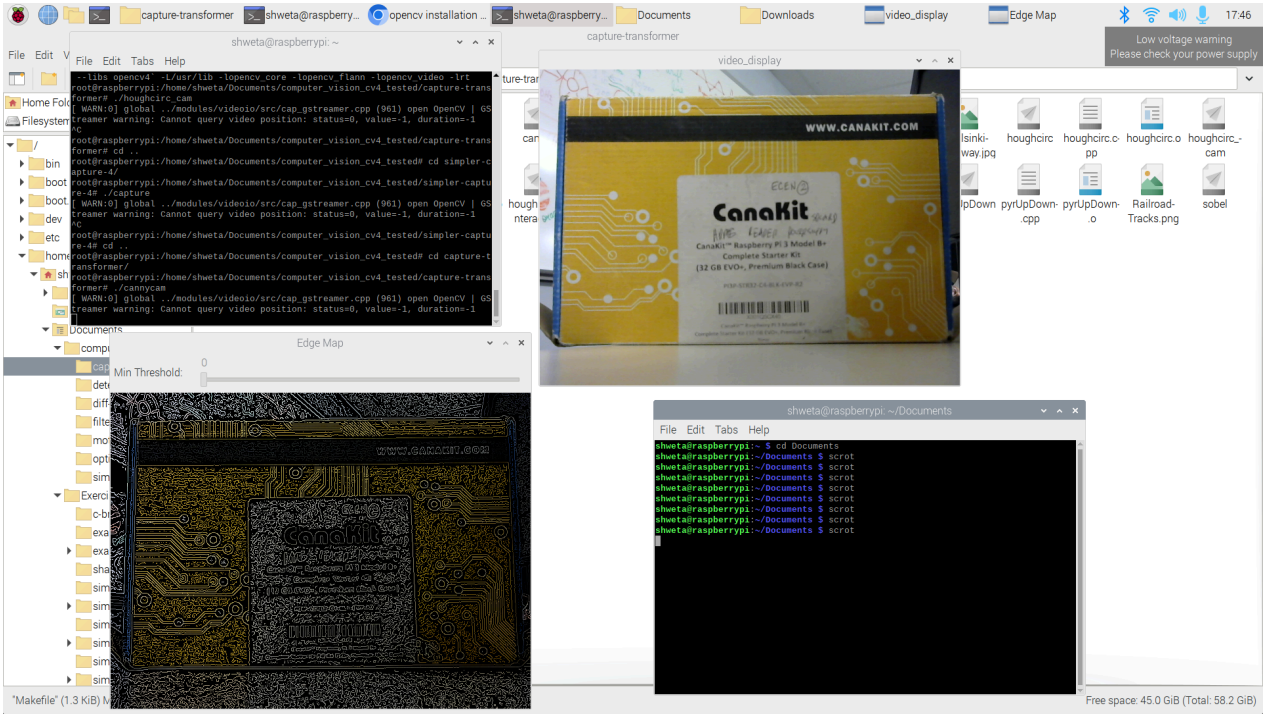
namedWindow():

namedWindow function creates a window with the name specified for displaying the captured frames from images or video.

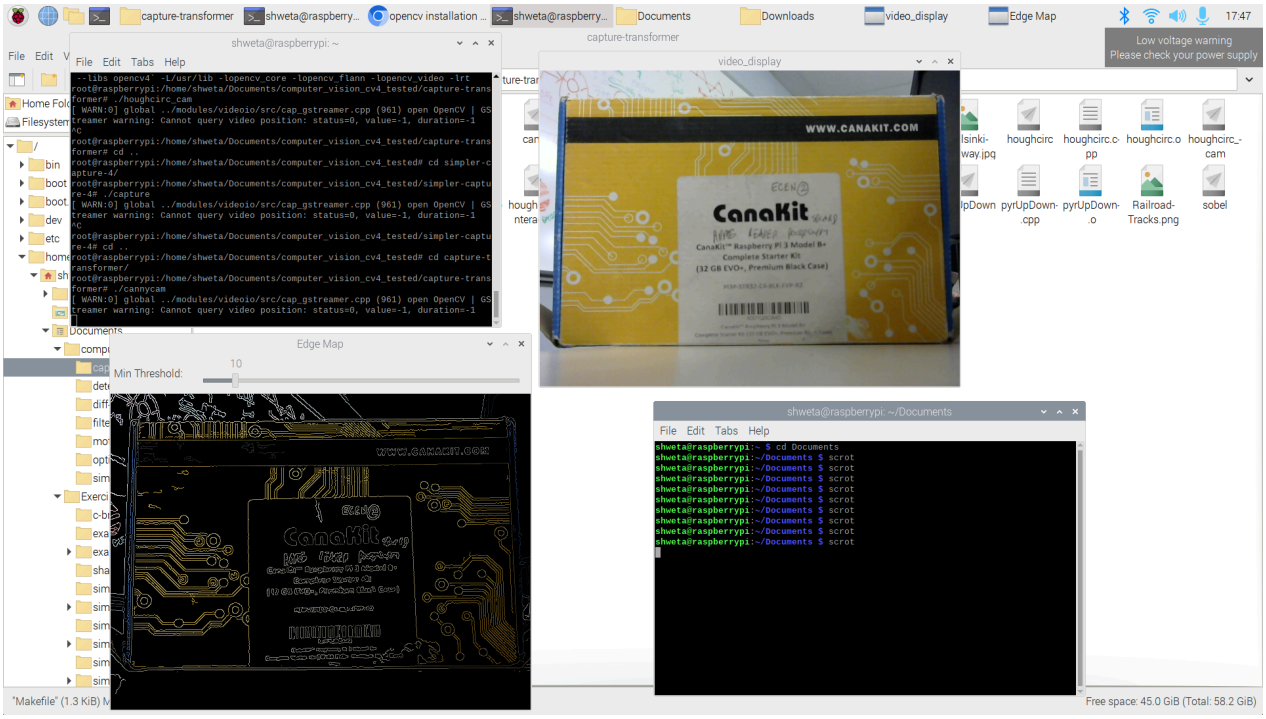
createTrackbar():

A trackbar is created to adjust the minimum threshold value for canny edge detection algorithm. This can be adjusted by the user.

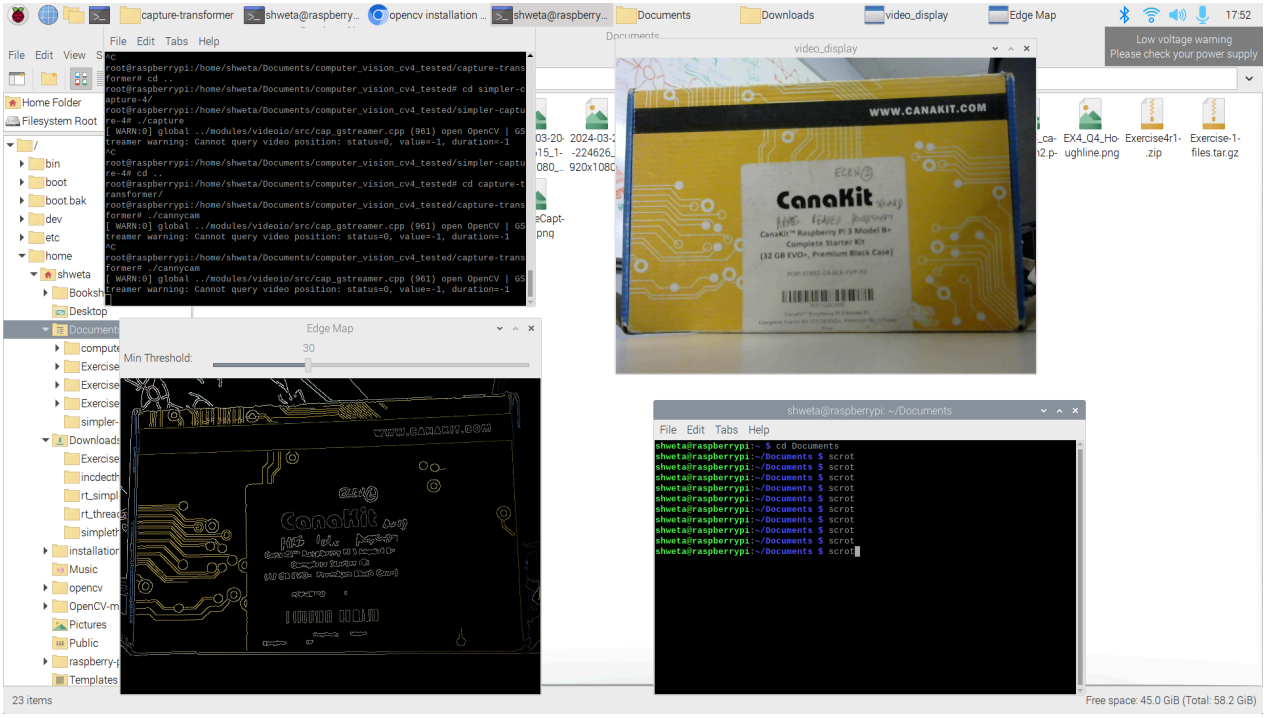
The below image is when the threshold is set to ‘0’.



The below image is when the threshold is set to ‘10’.



The below image is when the threshold is set to ‘30’.



Hough Line Detection Algorithm:

Hough line detection algorithm is a computer vision technique to detect straight lines in an image. It identifies straight lines in an image by representing each edge point as a line in parameter space. The algorithm then accumulates votes in an accumulator array based on the parameters of potential lines passing through each edge point detected using edge detection algorithms like canny edge detection. Peaks in the accumulator array indicate lines with the most votes, which are then extracted as detected lines in the image. This technique is efficient and effective for detecting straight lines which is valuable in applications such as lane detection, shape recognition, and image analysis.

VideoCapture:

VideoCapture class from OpenCV is used to capture video frames from camera or a video file. In simpler_capture4 code, index ‘0’ is passed which means taking input from the first connected camera.

cam0.set():

cam0.set() method sets theproperties of the capture device such as dimensions of the frame such as width and height.

canny():

The canny function performs canny edge detection algorithm on input image frame.

HoughLinesP():

HoughLinesP function performs the probabilistic Hough Line Transform on the output of canny() function.

line():

The Line function is used to draw the detected lines on the original captured image frame.

imshow():

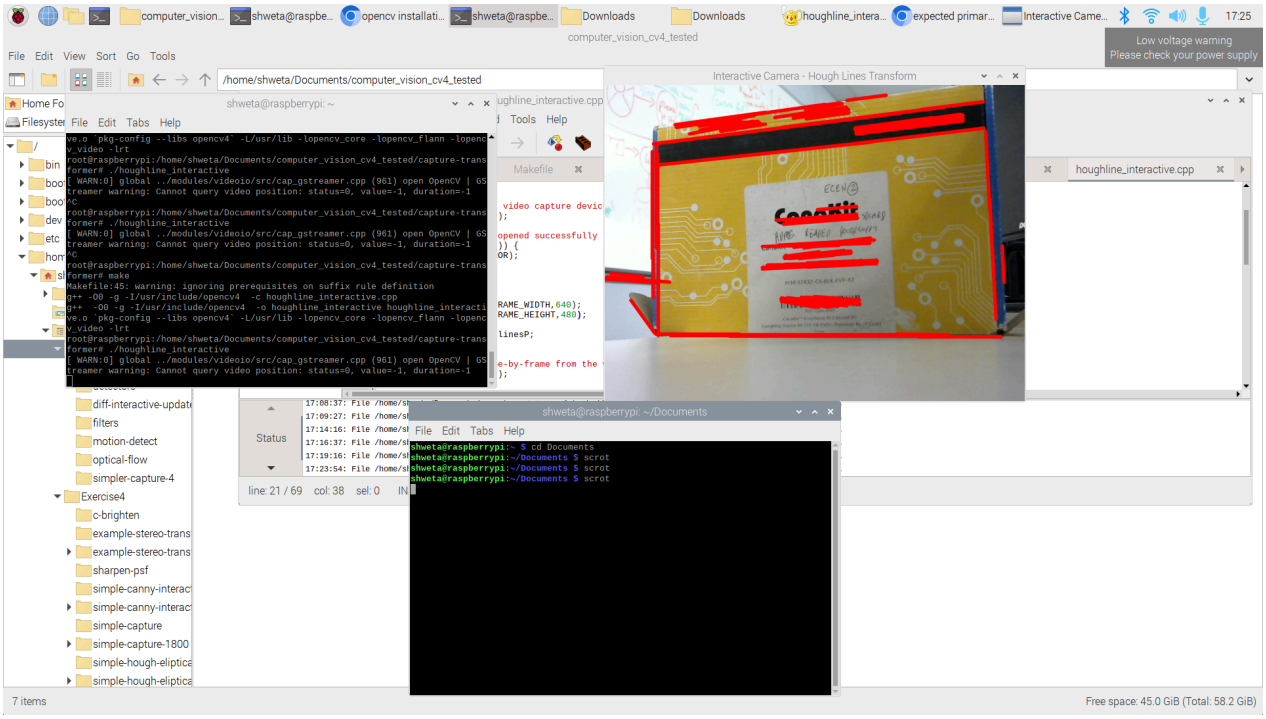
The imshow() function displays the captured frame in the specified window.

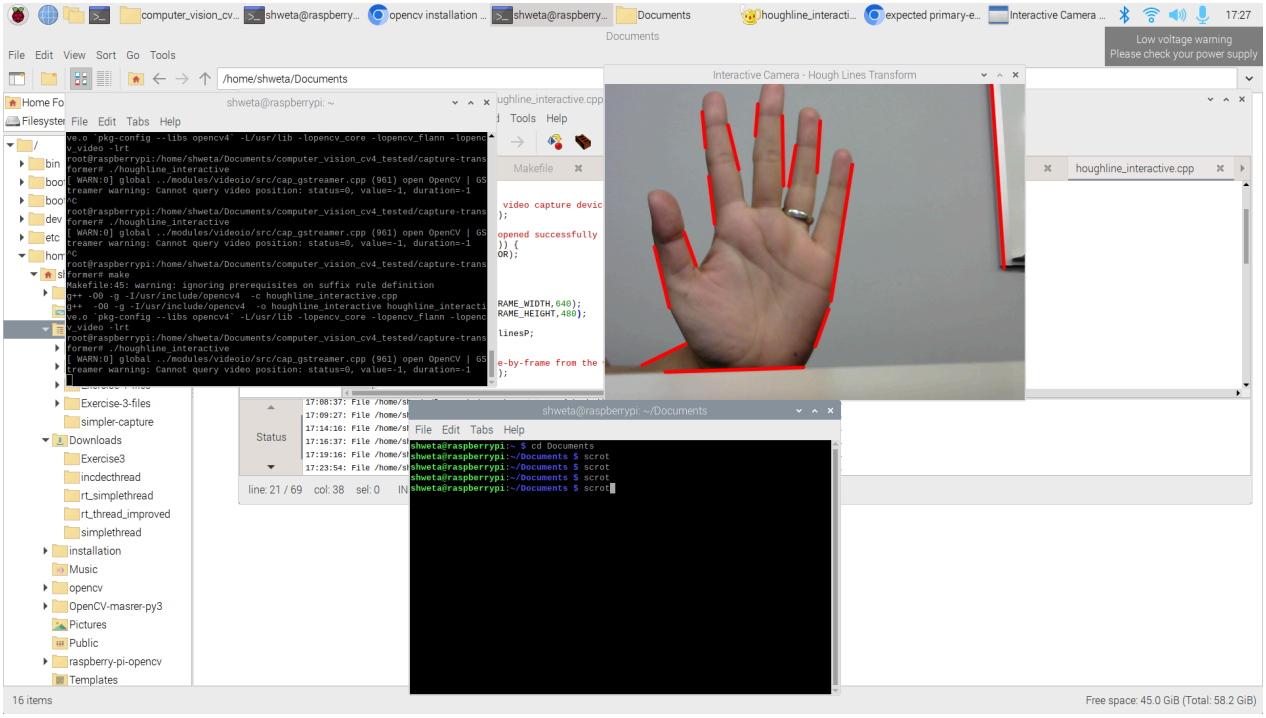
waitKey():

The waitKey functions keeps executing the previous function until a keyboard event occurs. It is used to wait for the user input.

destroyWindow():

destroyWindow function destroys the window specified and releases the resources associated with the window.





Hough Circle Detection Algorithm:

It is a technique used to detect circles in images in computer vision. The algorithm works by transforming the image space to a parameter space where each pixel in the image corresponds to a circle in the parameter space. It accumulates votes in the parameter space for possible circles by considering the edge points detected in the image. Peaks in the accumulator array represent the potential centers of circles, and their radii can be determined based on the radius range provided to the algorithm. By selecting the peaks that exceed a certain threshold, the algorithm identifies the circles present in the image. Hough Circle Detection is robust to noise and can accurately detect circles even in the presence of occlusions or partial visibility. It finds applications in various fields such as object recognition, medical imaging, and industrial quality control.

VideoCapture:

VideoCapture class from OpenCV is used to capture video frames from camera or a video file. In simpler_capture4 code, index ‘0’ is passed which means taking input from the first connected camera.

cam0.set():

cam0.set() method sets theproperties of the capture device such as dimensions of the frame such as width and height.

cam0.read():

cam0.read() method reads the next vidoe frame from the device capturing video and stores it in the matrix.

medianBlur():

meadianBlr function is used to reduce noise in the grayscale image by applying median blur.

HoughCircles():

HoughCircles function performs circle detection.

circle():

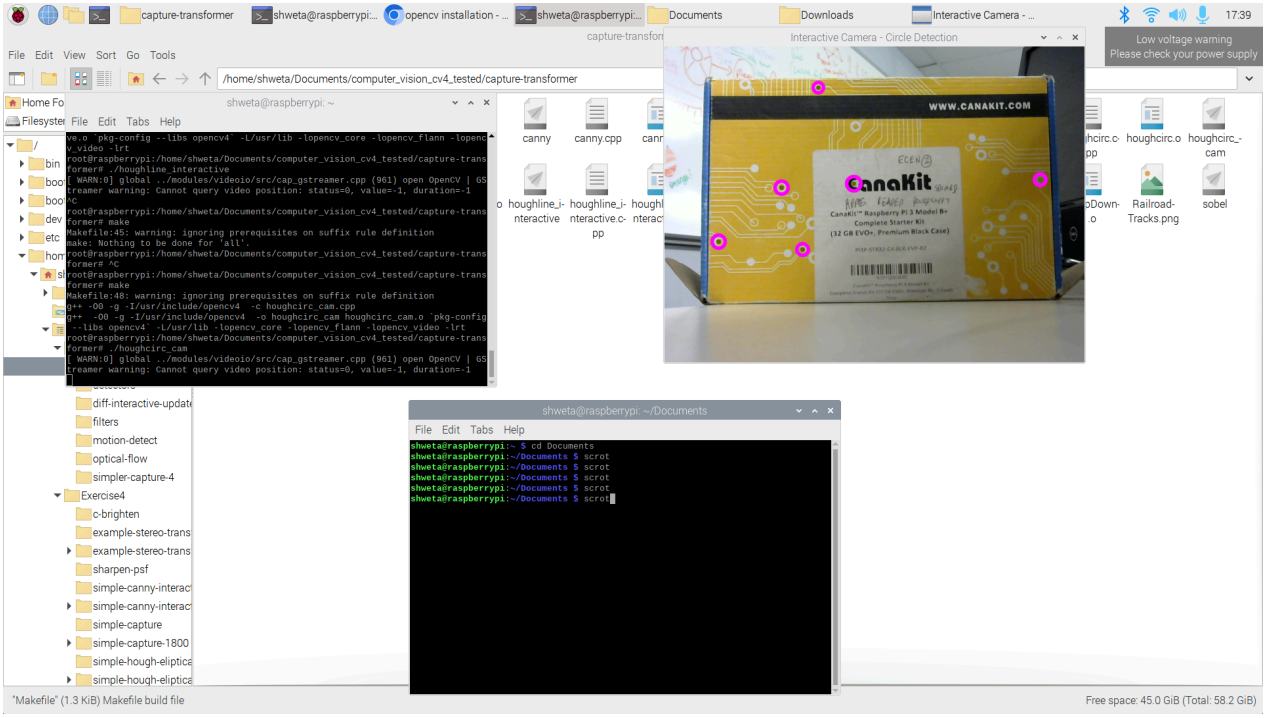
Circle function is used to draw detected circles on the captured input image frame.

waitKey():

The waitKey functions keeps executing the previous function until a keyboard event occurs. It is used to wait for the user input.

destroyWindow():

destroyWindow function destroys the window specified and releases the resources associated with the window.



Problem 5:(a) Three real-time interactive transformations

Design concepts (Flow Chart or Diagram of your solution)

We used the simple-canny, simple-hough and hough-elliptical transforms. We modified the example-stereo-transform-improved program from Prof. Sam Siewerts example codes to include these 3 transforms and convert into Q5_jitter.c with SCHED_FIFO scheduling algorithm and with the default scheduling algorithm with three resolutions having 18 data sets.

1. Input Resolution Selection:

- The user selects from a list of resolutions supported by the Logitech C200 camera, such 640x480, 320x240, 160x120.

2. Transformation Selection:

- The user chooses one of the real-time interactive transformations to compare: simple-canny, simple-hough, or hough-elliptical and also the scheduling algorithm and resolution
- This selection is asked after running the executable program (./Q5_jitter)

3. Capture and Processing Loop:

- The program captures frames from the camera continuously.
- For each frame, the selected transformation is applied, and the processing time is recorded using time-stamps.
- The transformed frames are displayed in real-time for visual inspection.

4. Average Frame Rate Calculation:
- The program runs the selected transformation for each resolution multiple times to gather data.
- It calculates the average frame rate by analyzing the delays between successive captures.
- The average frame rate is calculated for each resolution and transformation**.**

5. Soft Real-time Deadline Assignment:
- Based on the average frame rates obtained, soft real-time deadlines are assigned.
- The deadlines are set to be approximately 20 milliseconds greater than the corresponding frame rates to provide some margin.

6. SCHED_FIFO Conversion and Deadline Evaluation:
- The program converts the processing of the selected transformation to use the SCHED_FIFO scheduling policy.
- Each frame processing task is scheduled with a priority corresponding to the chosen transformation.
- The program measures the actual execution time of each task and evaluates whether deadlines are met.
- Jitter in the frame rate relative to the deadline is measured to assess the predictability of the system.

Algorithm analysis (Description of your solution and each transform used)

Initialization:
- The program initializes by including the necessary header files, defining constants for deadlines, and declaring global variables for file logging and storing execution times**.**

Canny Thresholding Transformation:
- The `CannyThreshold` function applies the Canny edge detection algorithm to the input frame.
- It uses a mutex to ensure thread safety when accessing the frame buffer and calculates execution time using clock timestamps.
- After applying the transformation, the transformed frame is displayed in real-time, and frame number, jitter, and execution time are logged to a CSV file.

Hough Line Transformation:
- The `imageProcessingThread` function handles the Hough line transformation.
- It captures frames from the camera and applies the Canny edge detection algorithm.
- Then, it detects lines using the Hough transform and draws them on the frame.
- Execution time is measured similarly to the Canny transformation, and relevant data is logged to the CSV file.

Hough Elliptical Transformation:
- Similar to the Hough line transformation, the `imageProcessingThread` function captures frames and applies Gaussian blur.
- It then detects circles using the Hough circle transform and draws them on the frame.

- Execution time is measured, and data is logged to the CSV file.

Main Function:
- In the main function, the user selects the resolution and scheduling policy.
- Based on the chosen transformation type, a corresponding thread is created to handle the processing.
- The program continuously captures frames, signals the processing thread, and displays the original frames.
- Upon quitting, the log file is closed, and the processing thread is cancelled and joined.

Logging and Analysis:
- The program logs frame **number, jitter, and execution time** for each frame processed.
- These data are used to analyze the performance of each transformation in terms of meeting soft real-time deadlines and jitter.
- The analysis can help determine the feasibility of using different transformations under specific conditions and resolutions.

By individually running the program for each transformation for 3 different resolution we get Average Frame Rate in ms:

Resolution	Canny	Hough	Hough Elliptical
640 x 480	30	70	35
320 x 240	9	35	20.5
160 x 120	2.4	11	19.3

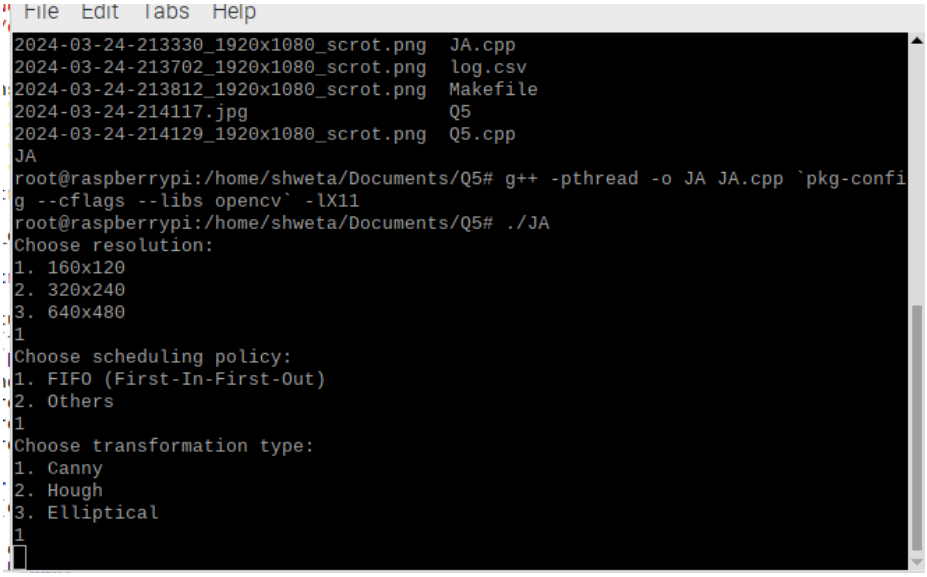
we decide the soft-real time deadline using the Frame rates found above. Once we get the time between each capture and release of the camera resource, we calculate the jitter by subtracting the delay with the deadlines.

Soft real-time deadline computed(ms):

Resolution	Canny	Hough	Hough Elliptical
640 x 480	50	90	55
320 x 240	30	55	40
160 x 120	20	30	40

Running Code:

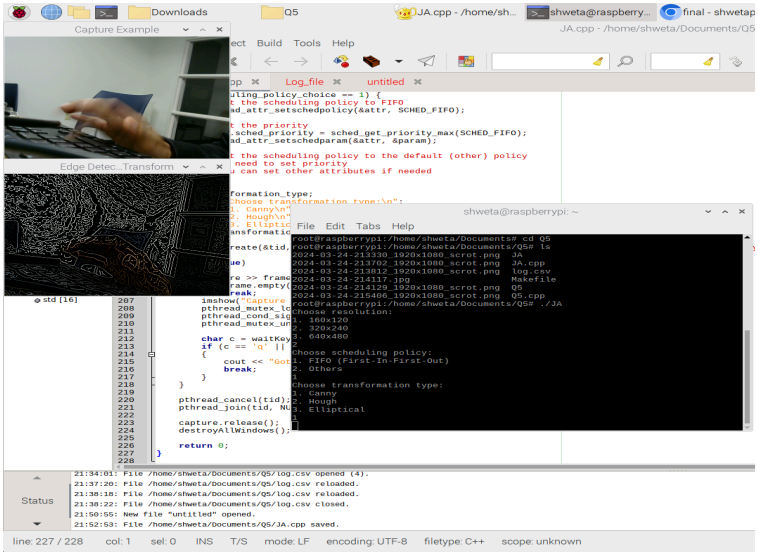
- Execute the program along with the transform to run. Ex. ./Q5_jitter
- The selection of resolution is prompted.
- After selection of resolution the scheduling policy selection is prompted.
- After selection of scheduling policy selection, the transformation selection is prompted.
- Quitting is done by pressing ‘q’.



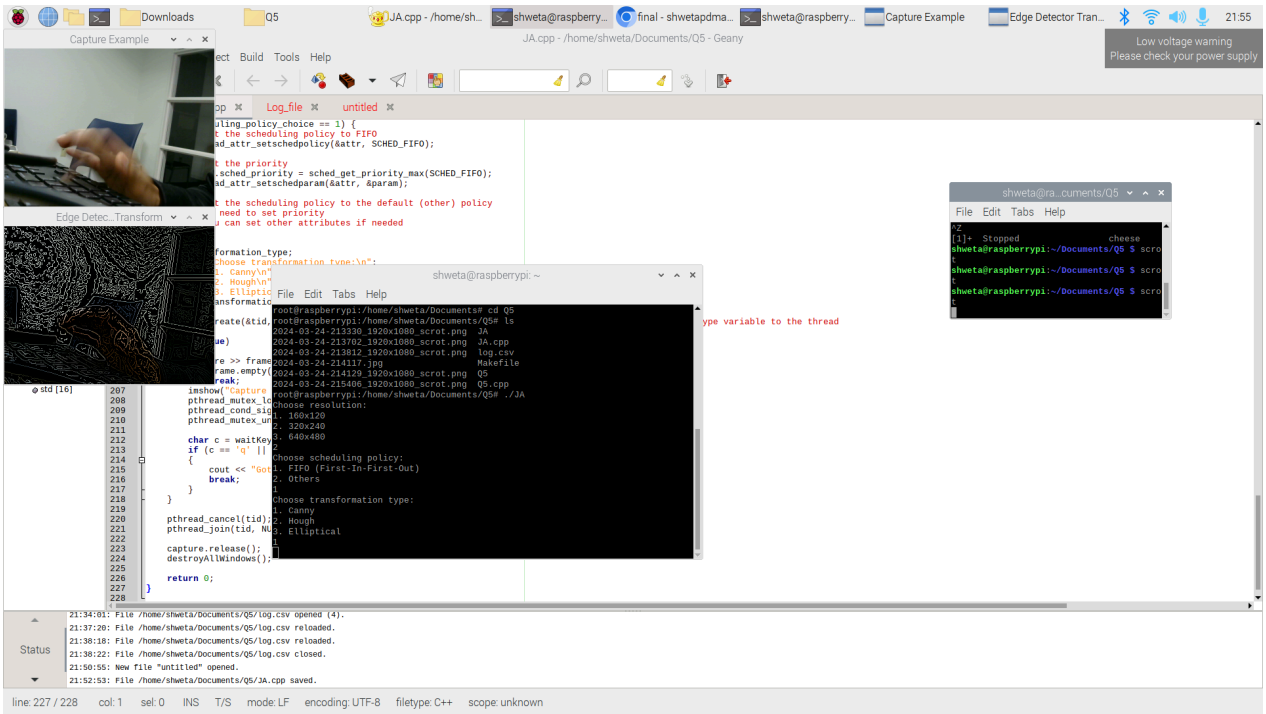
Prototype analysis including output screenshots:

(1)The screenshot for Canny Transformation:

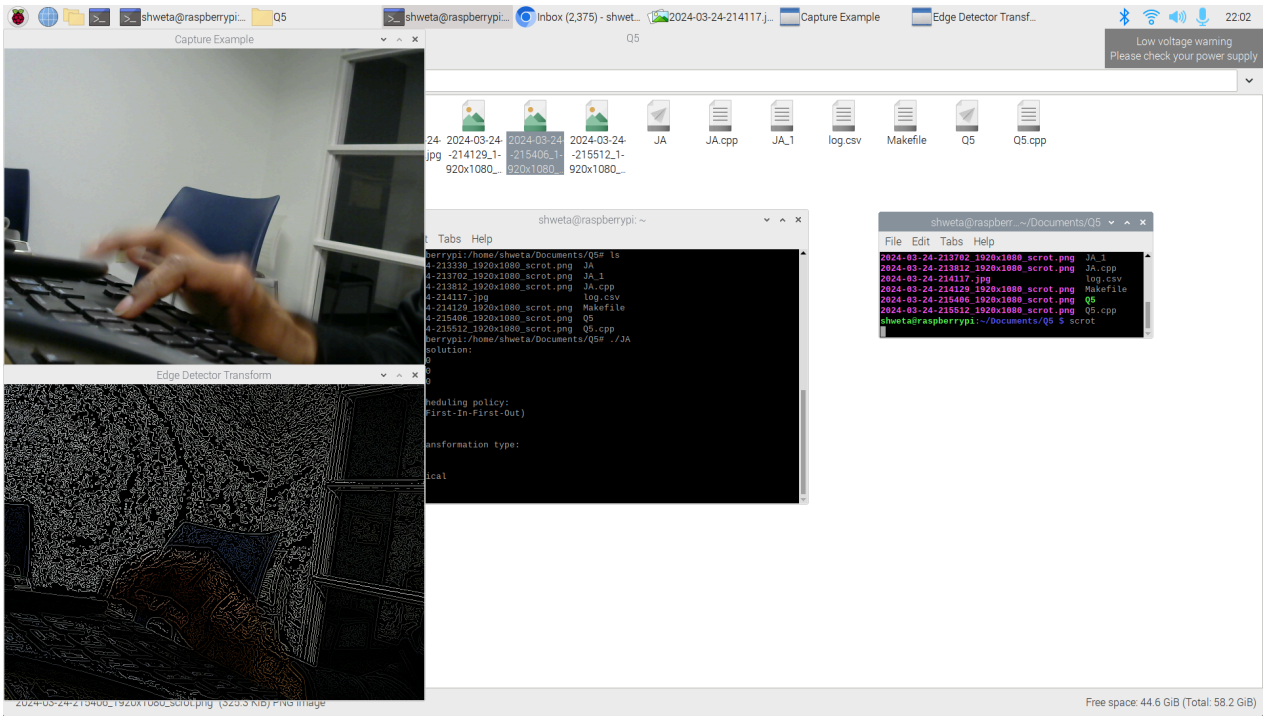
160 x 120:



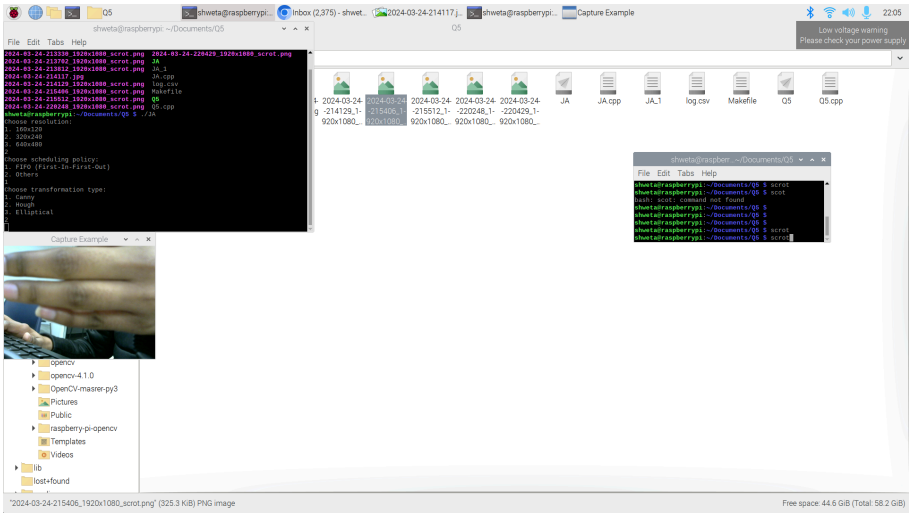
320 x 240:

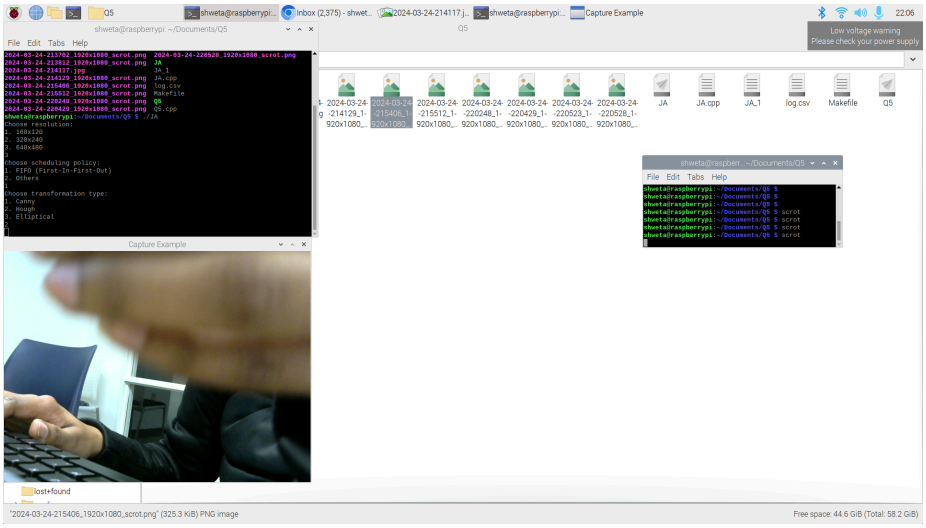


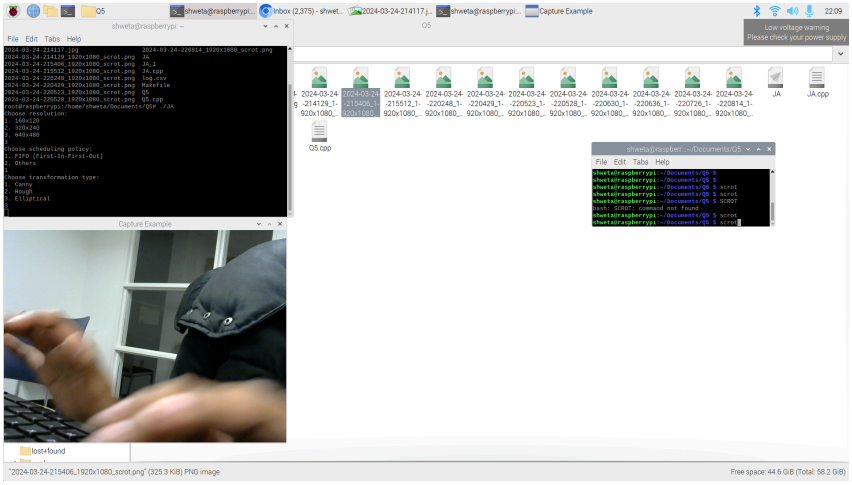
640 x 480


Capture Example
Edge Detector Transform
Low voltage warning
Please check your power supply
JA
JA.cpp
JA_1
log.csv
Makefile
Q5
Q5.cpp
Free space: 44.6 GiB (Total: 58.2 GiB)

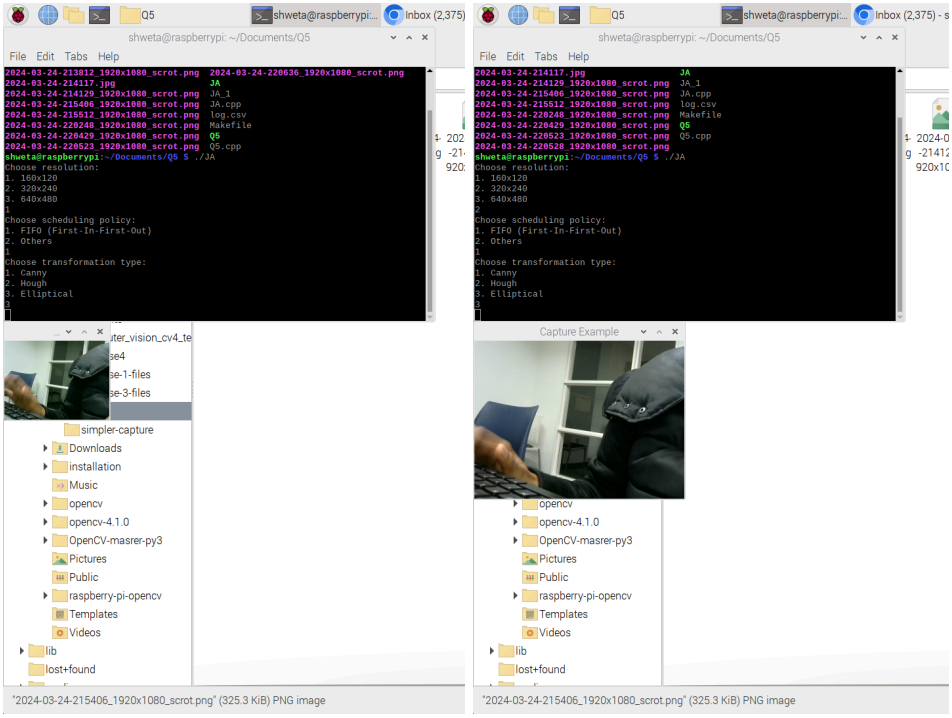
(2)The screenshot for Hough Transformation:

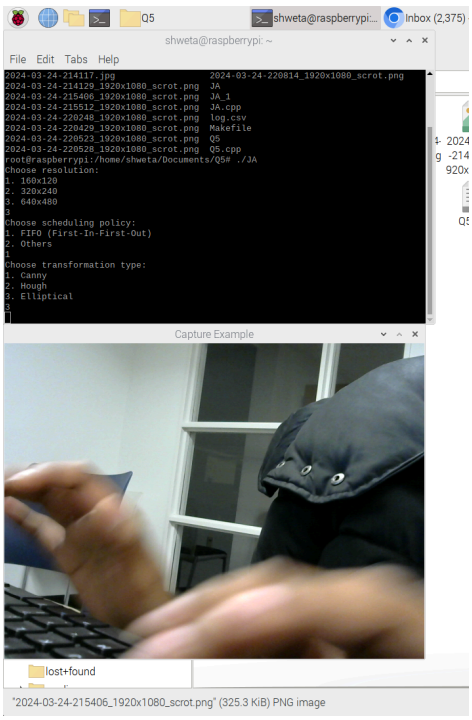






(3)The screenshot for Hough Elliptical:





Final predictable response jitter analysis

Average jitter(ms) in SCHED_FIFO for 100 frames:

Resolution	Canny	Hough	Hough Elliptical
640 x 480	-20	-150	-35
320 x 240	-15	-20	8
160 x 120	-17	-14	3

Jitter is calculated by taking the difference between the deadline and the frame rate. Positive values mean that the deadline was met and negative values mean that it was missed.

Appendix:

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <pthread.h>
#include <sched.h>
#include <X11/Xlib.h> // Include Xlib header

#include "opencv2/core.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/calib3d.hpp"
#include "opencv2/core/utility.hpp"
#include "opencv2/core/types.hpp"

#include <fstream> // Include for file operations
#include <iomanip> // Include for formatting output
#include <vector> // Include for storing execution times

using namespace cv;
using namespace std;

#define canny_deadline 20
#define hough_deadline 30
#define hough_elip_deadline 40
#define MSEC 1000000

ofstream logfile("log.csv"); // Declaration moved here
vector<long> execution_times; // Declaration moved here

// Global variables for logging
int frame_number = 0;
long total_execution_time = 0;
long jitter = 0;

char snapshotname[80] = "snapshot_xxx.jpg";

/* Canny transform parameters */
int lowThreshold = 0;
int const max_lowThreshold = 100;
int kernel_size = 3;
int canny_ratio = 3;
Mat canny_frame, timg_gray, timg_grad;
```

```
// Transform display window
char timg_window_name[] = "Edge Detector Transform";

Mat frame;

pthread_mutex_t frame_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t frame_cond = PTHREAD_COND_INITIALIZER;

void CannyThreshold(int, void*)
{
    Mat mat_frame;
    pthread_mutex_lock(&frame_mutex);
    while (frame.empty()) {
        pthread_cond_wait(&frame_cond, &frame_mutex); // Wait for a new frame
    }
    mat_frame = frame.clone(); // Cloning the frame to prevent data race
    pthread_mutex_unlock(&frame_mutex);

    cvtColor(mat_frame, timg_gray, COLOR_BGR2GRAY);
    blur(timg_gray, canny_frame, Size(3, 3));
    Canny(canny_frame, canny_frame, lowThreshold, lowThreshold * canny_ratio, kernel_size);
    timg_grad = Scalar::all(0);
    mat_frame.copyTo(timg_grad, canny_frame);
    imshow(timg_window_name, timg_grad);
}

void *imageProcessingThread(void *arg)
{
    double total_execution_time_canny = 0.0;
    double total_execution_time_hough = 0.0;
    double total_execution_time_elliptical = 0.0;
    /* Common parameters */
    VideoCapture capture;

    Mat disp, gray;
    int dev = 0;

    /* For Hough Interactive */
    vector<Vec4i> lines;

    /* For Hough Elliptical Interactive */
    vector<Vec3f> circles;

    timespec timestamp;
    int transformation_type = *((int *)arg);
    while (true)
    {
```

```
pthread_mutex_lock(&frame_mutex);
if (!frame.empty())
{
    pthread_mutex_unlock(&frame_mutex);
    if (transformation_type == 1)
    { // Canny
        // Measure execution time
        clock_gettime(CLOCK_REALTIME, &timestamp);
        long start_sec = timestamp.tv_sec;
        long start_nsec = timestamp.tv_nsec;

        CannyThreshold(0, 0);

        clock_gettime(CLOCK_REALTIME, &timestamp);
        long stop_sec = timestamp.tv_sec;
        long stop_nsec = timestamp.tv_nsec;
        long delta_time_sec = stop_sec - start_sec;
        long delta_time_nsec = stop_nsec - start_nsec;
        long execution_time = (delta_time_sec * MSEC + delta_time_nsec / MSEC);
        jitter = canny_deadline - execution_time;

        execution_times.push_back(execution_time);

                    // Log frame number, jitter, and execution time
        logfile << frame_number << "," << jitter << "," << execution_time << endl;

        // Increment frame number
        frame_number++;

    }
    else if (transformation_type == 2)
    { // Hough
        // Measure execution time
        clock_gettime(CLOCK_REALTIME, &timestamp);
        long start_sec = timestamp.tv_sec;
        long start_nsec = timestamp.tv_nsec;

        Mat mat_frame(frame); // Removed 'frame'

        Canny(mat_frame, canny_frame, 50, 200, 3);

        HoughLinesP(canny_frame, lines, 1, CV_PI / 180, 50, 50, 10);

        for (size_t i = 0; i < lines.size(); i++)
        {
            Vec4i l = lines[i];
            line(mat_frame, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(0, 0, 255), 3, LINE_AA);
        }
```

```
    clock_gettime(CLOCK_REALTIME, &timestamp);
    long stop_sec = timestamp.tv_sec;
    long stop_nsec = timestamp.tv_nsec;
    long delta_time_sec = stop_sec - start_sec;
    long delta_time_nsec = stop_nsec - start_nsec;
    long execution_time = (delta_time_sec * MSEC + delta_time_nsec / MSEC);
    jitter = hough_deadline - execution_time;

    execution_times.push_back(execution_time);

    logfile << frame_number << "," << jitter << "," << execution_time << endl;

    // Increment frame number
    frame_number++;
}
else if (transformation_type == 3)
{ // Hough Elliptical
    // Measure execution time
    clock_gettime(CLOCK_REALTIME, &timestamp);
    long start_sec = timestamp.tv_sec;
    long start_nsec = timestamp.tv_nsec;

    capture >> frame;
    if (frame.empty())
        break;

    Mat mat_frame(frame);

    cvtColor(mat_frame, gray, COLOR_BGR2GRAY);

    GaussianBlur(gray, gray, Size(9, 9), 2, 2);

    HoughCircles(gray, circles, HOUGH_GRADIENT, 1, gray.rows / 8, 100, 50, 0, 0);

    printf("Circles.size = %d\n", circles.size());

    for (size_t i = 0; i < circles.size(); i++)
    {
        Point center(cvRound(circles[i][0]), cvRound(circles[i][1]));
        int radius = cvRound(circles[i][2]);
        // circle center
        circle(mat_frame, center, 3, Scalar(0, 255, 0), -1, 8, 0);
        // circle outline
        circle(mat_frame, center, radius, Scalar(0, 0, 255), 3, 8, 0);
    }

    imshow("Capture Example", mat_frame);
```

```
            clock_gettime(CLOCK_REALTIME, &timestamp);
            long stop_sec = timestamp.tv_sec;
            long stop_nsec = timestamp.tv_nsec;
            long delta_time_sec = stop_sec - start_sec;
            long delta_time_nsec = stop_nsec - start_nsec;
            long execution_time = (delta_time_sec * MSEC + delta_time_nsec / MSEC);
            jitter = hough_elip_deadline - execution_time;


            execution_times.push_back(execution_time);

                        // Log frame number, jitter, and execution time
            logfile << frame_number << "," << jitter << "," << execution_time << endl;

            // Increment frame number
            frame_number++;
         }
      }
      else
      {
         pthread_cond_wait(&frame_cond, &frame_mutex); // Wait for a new frame
         pthread_mutex_unlock(&frame_mutex);
      }
   }
   pthread_exit(NULL);
}

int main(int argc, char **argv)
{
   XInitThreads(); // Initialize Xlib for multi-threaded operation

   VideoCapture capture;

   string resolution_choice;
   cout << "Choose resolution:\n";
   cout << "1. 160x120\n";
   cout << "2. 320x240\n";
   cout << "3. 640x480\n";
   getline(cin, resolution_choice);

   if (resolution_choice == "1") {
      capture.open(0);
      capture.set(CAP_PROP_FRAME_WIDTH, 160);
      capture.set(CAP_PROP_FRAME_HEIGHT, 120);
   } else if (resolution_choice == "2") {
      capture.open(0);
      capture.set(CAP_PROP_FRAME_WIDTH, 320);
```

```
    capture.set(CAP_PROP_FRAME_HEIGHT, 240);
} else if    (resolution_choice == "3") {
    capture.open(0);
    capture.set(CAP_PROP_FRAME_WIDTH, 640);
    capture.set(CAP_PROP_FRAME_HEIGHT, 480);
} else {
    cerr << "Invalid choice\n";
    return EXIT_FAILURE;
}

if (!capture.isOpened())
{
    cerr << "Error: Unable to open camera" << endl;
    return EXIT_FAILURE;
}

namedWindow("Capture Example", WINDOW_AUTOSIZE);

pthread_t tid;
pthread_attr_t attr;
struct sched_param param;

pthread_attr_init(&attr);

int scheduling_policy_choice;
cout << "Choose scheduling policy:\n";
cout << "1. FIFO (First-In-First-Out)\n";
cout << "2. Others\n";
cin >> scheduling_policy_choice;

if (scheduling_policy_choice == 1) {
    // Set the scheduling policy to FIFO
    pthread_attr_setschedpolicy(&attr, SCHED_FIFO);

    // Set the priority
    param.sched_priority = sched_get_priority_max(SCHED_FIFO);
    pthread_attr_setschedparam(&attr, &param);
} else {
    // Set the scheduling policy to the default (other) policy
    // No need to set priority
}

int transformation_type;
cout << "Choose transformation type:\n";
cout << "1. Canny\n";
cout << "2. Hough\n";
cout << "3. Elliptical\n";
cin >> transformation_type;
```

```
    pthread_create(&tid, &attr, imageProcessingThread, &transformation_type); // Pass the
transformation_type variable to the thread

    while (true)
    {
        capture >> frame;
        if (frame.empty())
            break;
        imshow("Capture Example", frame);
        pthread_mutex_lock(&frame_mutex);
        pthread_cond_signal(&frame_cond); // Signal the processing thread
        pthread_mutex_unlock(&frame_mutex);

        char c = waitKey(10);
        if (c == 'q' || c == 'Q') // Check for 'q' or 'Q' key press to quit
        {
            cout << "Got Quit" << endl;
            break;
        }
    }

    // Close log file
    logfile.close();
    pthread_cancel(tid); // Cancel the image processing thread
    pthread_join(tid, NULL); // Wait for the thread to finish

    capture.release();
    destroyAllWindows();

    return 0;
}
```