# Exercise 4 Report/Write-Up

Dhiraj Bennadi and Maitreyee Rao

March 03, 2022

ECEN 5623- Spring 2022

# 1 Problem 1

[10 points] Obtain a Logitech C200 or C270 camera or equivalent and verify that is detected by the DE1-SoC, Raspberry Pi or Jetson Board USB driver. You can check the camera out from the TA's or purchase one of your own, or use another camera that has a compliant UVC driver. Use lsusb, lsmod and dmesg kernel driver configuration tool to make sure your Logitech C2xx USB camera is plugged in and recognized by your DE1-SoC, Raspberry Pi or Jetson (note that on the Jetson, it does not use driver modules, but rather a monolithic kernel image, so lsmod will not look the same as other Linux systems – see what you can find by exploring /cat/proc on you Jetson to find the camera USB device). For the Jetson, do lsusb — grep C200 (or C270) and prove to the TA (and more importantly yourself) with that output (screenshot) that your camera is recognized. For systems other than a Jetson, do lsmod — grep video and verify that the UVC driver is loaded as well (http://www.ideasonboard.org/uvc/ ). To further verify, or debug if you don't see the UVC driver loaded in response to plugging in the USB camera, do dmesg — grep video or just dmesg and scroll through the log messages to see if your USB device was found. Capture all output and annotate what you see with descriptions to the best of your understanding.

## Solution

## USB Identification

```
realtime@RtesSpring2022:~$ lsusb
Bus 002 Device 002: ID 0bda:0411 Realtek Semiconductor Corp.
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 006: ID 046d:082d Logitech, Inc. HD Pro Webcam C920
Bus 001 Device 005: ID 046d:c534 Logitech, Inc. Unifying Receiver
Bus 001 Device 003: ID 0bda:8179 Realtek Semiconductor Corp. RTL8188EUS 802.11n Wireless Network Adapter
Bus 001 Device 002: ID 0bda:5411 Realtek Semiconductor Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
realtime@RtesSpring2022:~$
```

## Driver Verification

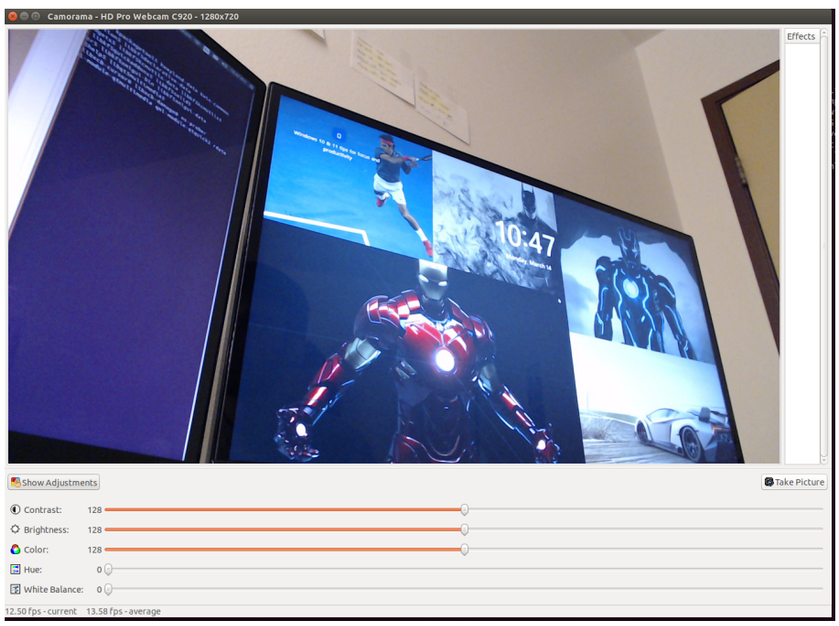
```
realtime@RtesSpring2022:~$ dmesg | grep video
[    0.000000] Kernel command line: tegraid=21.1.2.0.0 ddr_die=4096M@2048M section=512M memtype=0 vpr_resize usb_port_owner_info=0 lane_owner_info=0 emc_max_dvfs=0 touch_id=0@63 video=tegrafb no_console_s
uspend=1 console=ttyS0,115200n8 debug_uartport=lsport,4 earlyprintk=uart8250-32bit,0x70006000 maxcpus=4 usbcore.old_scheme_first=1 lp0_vec=0x1000@0xff780000 core_edp_mv=1075 core_edp_ma=4000 gpt tegra_fbm
em=0x800000@0x92ca0000 ls_hdmi_initialized=1  earlycon=uart8250,mmio32,0x70006000  root=/dev/mmcblk0p1 rw rootwait rootfstype=ext4 console=ttyS0,115200n8 console=tty0 fbcon=map:0 net.ifnames=0 quiet root=
/dev/mmcblk0p1 rw rootwait rootfstype=ext4 console=ttyS0,115200n8 console=tty0 fbcon=map:0 net.ifnames=0
[    0.683297] Linux video capture interface: v2.00
[  853.054240] uvcvideo: Found UVC 1.00 device HD Pro Webcam C920 (046d:082d)
[  853.054929] uvcvideo 1-2.4:1.0: Entity type for entity Processing 3 was not initialized!
[  853.064125] uvcvideo 1-2.4:1.0: Entity type for entity Extension 6 was not initialized!
[  853.072311] uvcvideo 1-2.4:1.0: Entity type for entity Extension 12 was not initialized!
[  853.080493] uvcvideo 1-2.4:1.0: Entity type for entity Camera 1 was not initialized!
[  853.088408] uvcvideo 1-2.4:1.0: Entity type for entity Extension 8 was not initialized!
[  853.096515] uvcvideo 1-2.4:1.0: Entity type for entity Extension 9 was not initialized!
[  853.104795] uvcvideo 1-2.4:1.0: Entity type for entity Extension 10 was not initialized!
[  853.112941] uvcvideo 1-2.4:1.0: Entity type for entity Extension 11 was not initialized!
[  853.121860] usbcore: registered new interface driver uvcvideo
realtime@RtesSpring2022:~$
```

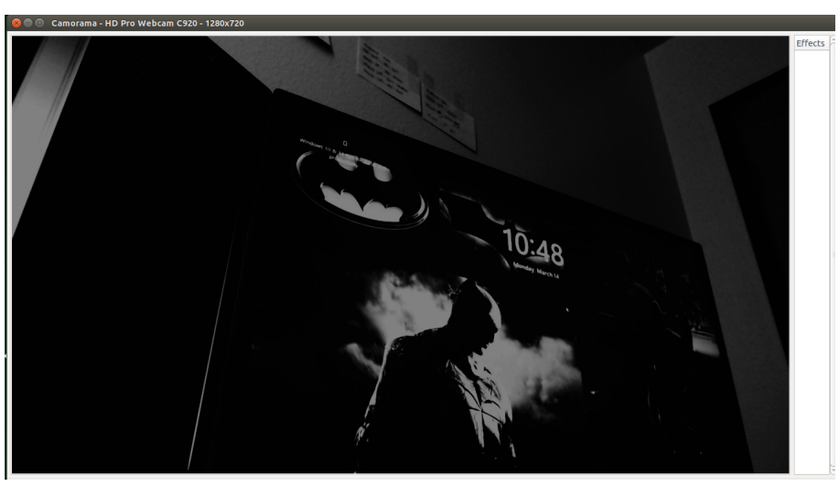
```
realtime@RtesSpring2022:~$ lsmod
Module                  Size  Used by
uvcvideo               90913  0
bnep                   16562  2
fuse                  104554  5
xt_conntrack            3609  1
ipt_MASQUERADE          2346  1
nf_nat_masquerade_ipv4     3111  1 ipt_MASQUERADE
nf_conntrack_netlink    29413  0
nfnetlink               7959  2 nf_conntrack_netlink
xt_addrtype             3670  2
iptable_filter          2481  1
iptable_nat             2882  1
nf_conntrack_ipv4      11992  2
nf_defrag_ipv4          1836  1 nf_conntrack_ipv4
nf_nat_ipv4             6712  1 iptable_nat
nf_nat                 20406  2 nf_nat_masquerade_ipv4,nf_nat_ipv4
nf_conntrack          106659  6 nf_conntrack_ipv4,nf_conntrack_netlink,nf_nat_masquerade_ipv4,xt_conntrack,nf_nat_ipv4,nf_nat
br_netfilter           16216  0
zram                   25920  4
r8188eu               470649  0
cfg80211              591474  1 r8188eu
overlay                48718  0
userspace_alert         5828  0
nvgpu                1589200  56
ip_tables              19441  2 iptable_filter,iptable_nat
x_tables               28951  5 ip_tables,ip6table_filter,ip6t_MASQUERADE,xt_addrtype,xt_conntrack
realtime@RtesSpring2022:~$
```

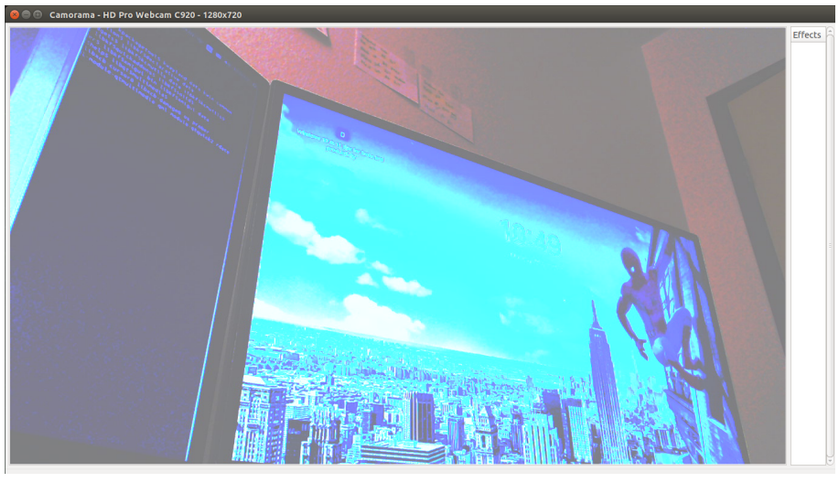
# 2 Problem 2

**option 1: camorama** If you do not have camorama, do apt-get install camorama on your DE1-SoC, Raspberry Pi, or Jetson board [you may need to first do sudo add-apt-repository universe; sudo apt-get update]. This should not only install nice camera capture GUI tools, but also the V4L2 API (described well in this series of Linux articles - http://lwn.net/Articles/203924/ ). Running camorama should provide an interactive camera control session for your Logitech C2xx camera – if you have issues connecting to your camera do a "man camorama" and specify your camera device file entry point (e.g. /dev/video0). Run camorama and play with Hue, Color, Brightness, White Balance and Contrast, take an example image and take a screen shot of the tool and provide both in your report.
**Option 2: Cheese** If you do not have cheese, do sudo apt-get install cheese on your Jetson, Raspberry Pi or DE1-SoC board or other native Linux system. This should not only install nice camera capture GUI tools, but also the V4L2 API (described well in this series of Linux articles - http://lwn.net/Articles/203924/ . Running cheese should provide an interactive camera control session for your Logitech C2xx camera – if you have issues connecting to your camera do a "man cheese" and specify your camera device file entry point (e.g. /dev/video0). Show that you tested your camera with a cheese screen dump and test photo.

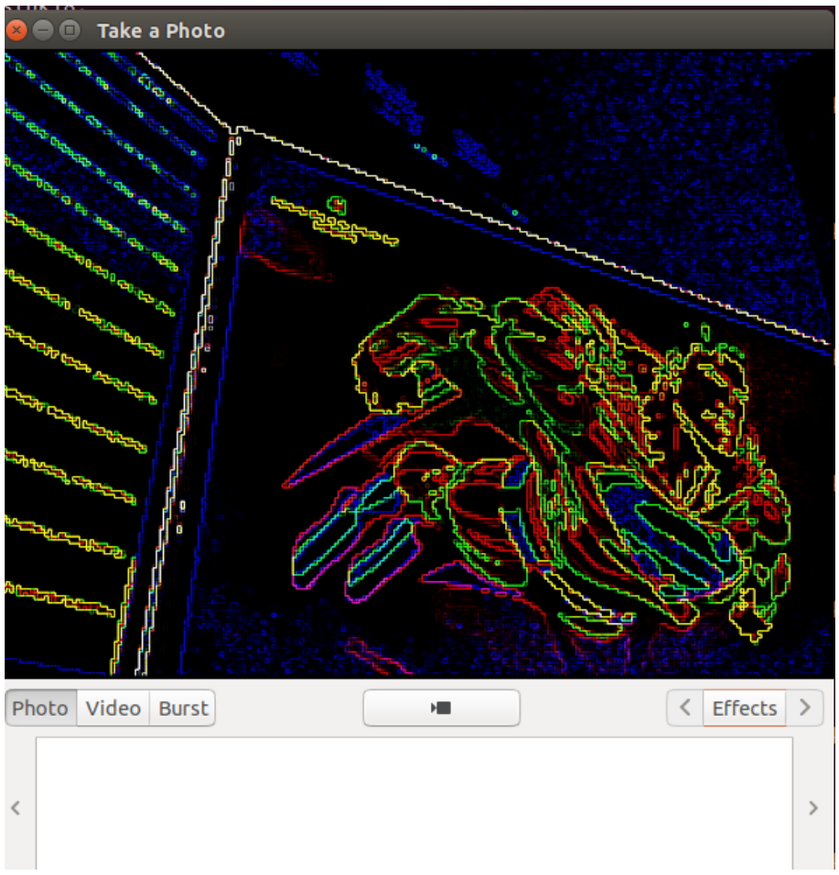
## Camorama

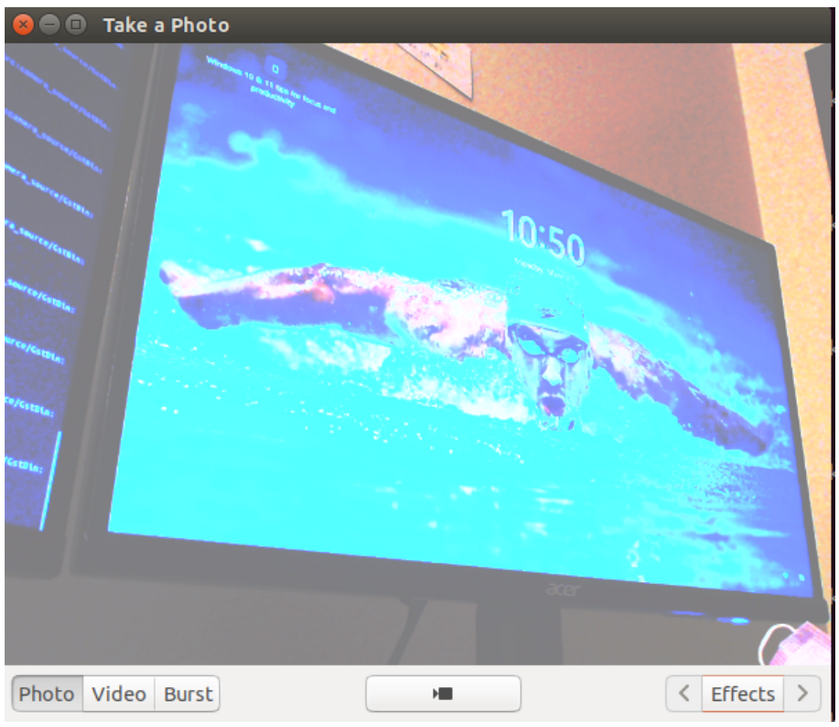



Camorama - HD Pro Webcam C920 - 1280x720
Effects

## Cheese

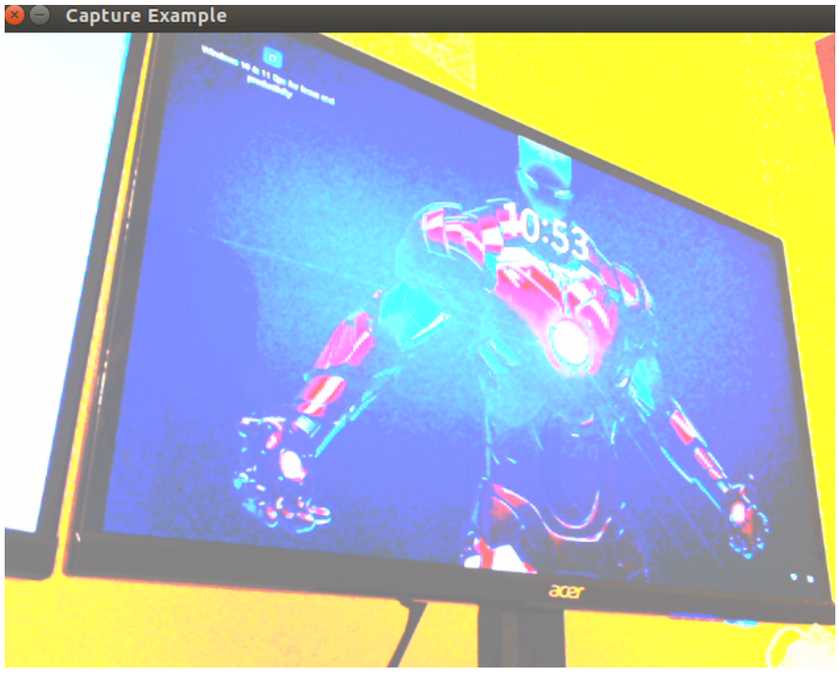

Take a Photo
Photo
Video
Burst
Effects




# 3 Problem 3

[10 points] 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 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.

### Solution




# 4 Problem 4

[20 points] Choose a continuous transformation OpenCV example from computer-vision such as the canny-interactive, hough-interactive, hough-eliptical-interactive, or stereo-transform-impoved or the from the same 4 transforms in computer vision cv3 tested. 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.

### Solution

## 4.1 Canny Transformation

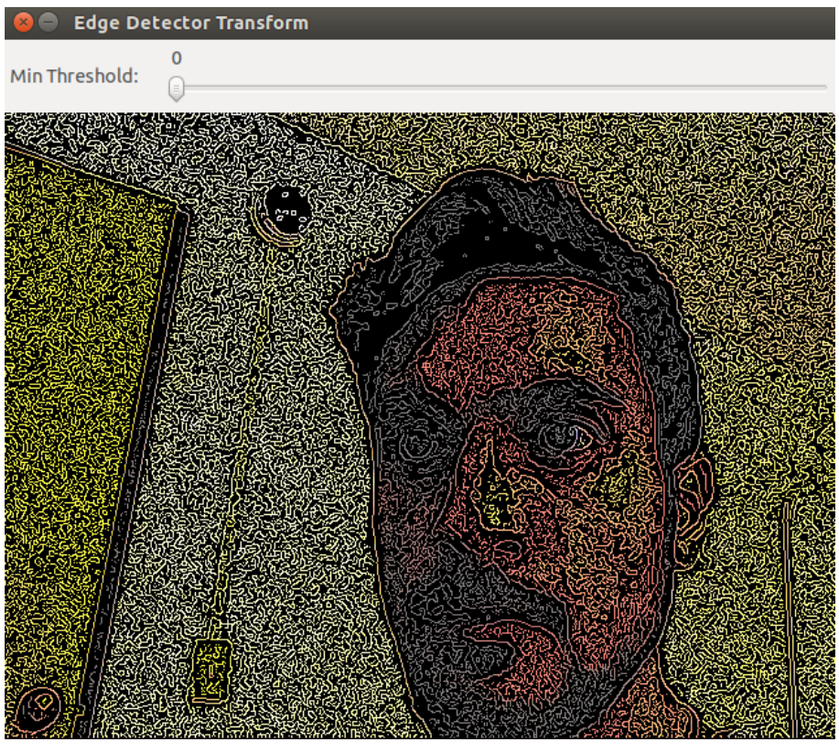
Canny Edge Detection is a popular edge detection algorithm.
It is a multi-stage algorithm with the following stages:

1. Noise Reduction
2. Finding Intensity Gradient of the image
3. Non-maximum suppression
4. Hysteresis Thresholding

## Code Explanation

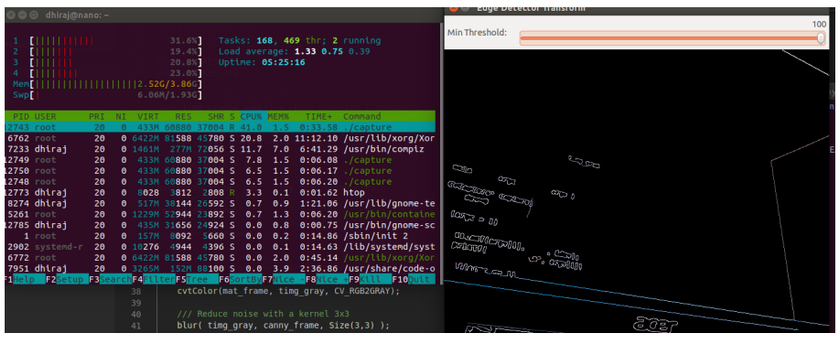
1. The frame from the camera device is captured
2. The captured frame is converted into Matrix datatype for transformation
3. The image is converted to gray scale to make the image single dimensional
4. The image is blurred using a filter to reduce the noise
5. The Canny Detector Algorithm is applied on the blured image
6. The edge detected image is displayed on the image window

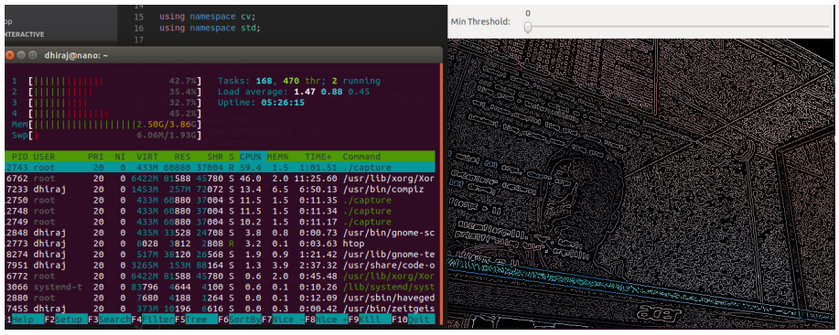
## Code Output




## CPU Load

Images from a camera feed are usually data intensive. The resolution of the image captured, the camera device and the capture algorithms increase or decrease the size of the image. The CPU Load varies between **40% to 60%**. The board used to execute the algorithm is a Jetson Nano. Since Jetson Nano runs an image of Ubuntu, the CPU shares this load across the 4 cores. This is inferred by using the command **htop** and the executable **./capture** can be seen to execute across the cores of the CPU.



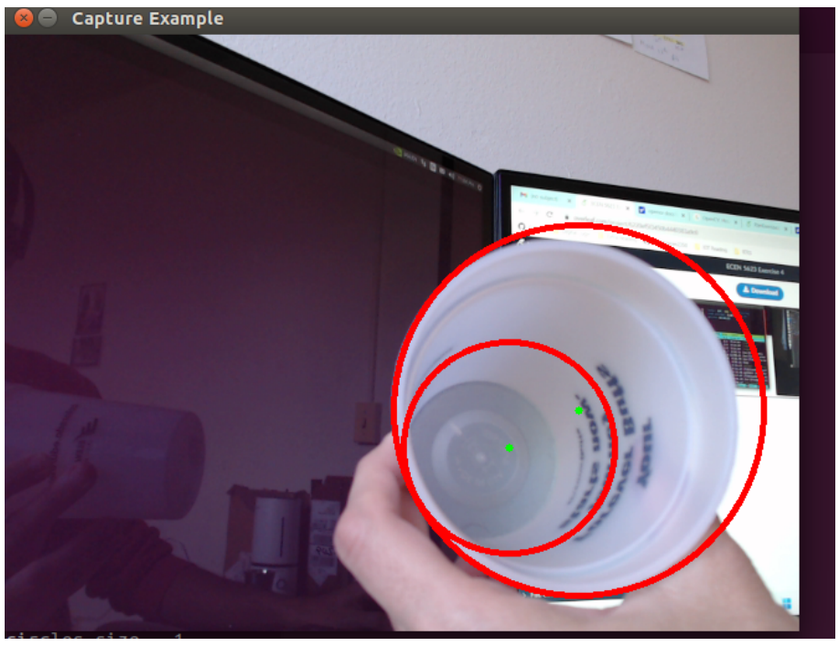

## 4.2 Hough Elliptical Transformation

Hough Elliptical Transformation is used to detect center of circular objects.

## Code Explanation

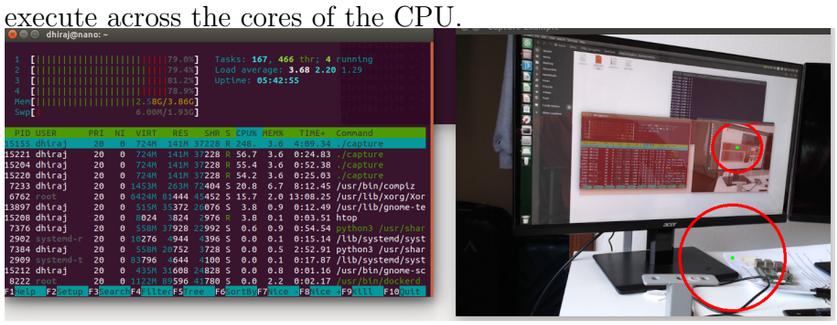
1. The frame from the camera device is captured
2. The captured frame is converted into Matrix datatype for transformation
3. The image is converted to gray scale to make the image single dimensional
4. The Hough Circles transformation is applied on the blurred imaged and the circles are calculated in the image
5. The circles identified in the image are displayed on the image using drawing tool APIs

## Code Output




## CPU Load

Images from a camera feed are usually data intensive. The resolution of the image captured, the camera device and the capture algorithms increase or decrease the size of the image. The CPU Load is around **240%**. The board used to execute the algorithm is a Jetson Nano. Since Jetson Nano runs an image of Ubuntu, the CPU shares this load across the 4 cores. This is inferred by using the command **htop** and the executable **./capture** can be seen to execute across the cores of the CPU.



## 4.3 Hough Line Transformation

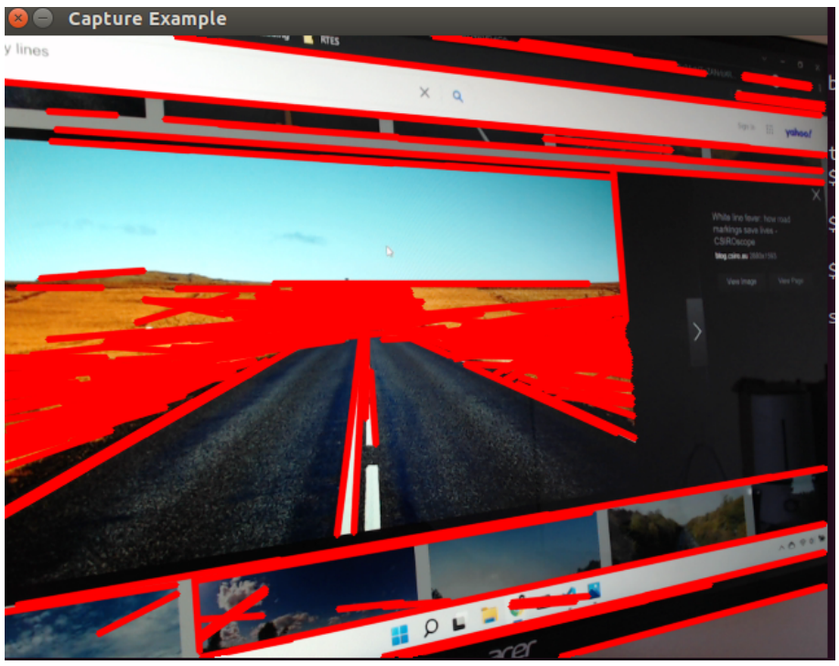
Hough Line Transformation is used to detect lines in the image. The initial stage of the Hough transformation is edge detection.

## Code Explanation

1. The frame from the camera device is captured
2. The captured frame is converted into Matrix datatype for transformation

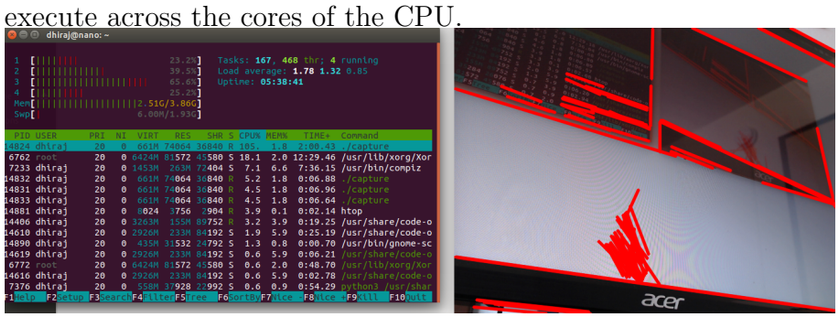
3. The Canny transformation is applied on the image for edge detected
4. The Hough Lines Transformation is applied on the edge detected image
5. The lines identified in the image are displayed on the image using the drawing tool APIs

## Code Output




## CPU Load

Images from a camera feed are usually data intensive. The resolution of the image captured, the camera device and the capture algorithms increase or decrease the size of the image. The CPU Load is around **100%**. The board used to execute the algorithm is a Jetson Nano. Since Jetson Nano runs an image of Ubuntu, the CPU shares this load across the 4 cores. This is inferred by using the command **htop** and the executable **./capture** can be seen to execute across the cores of the CPU.



# 5 Problem 5

[40 points] Using a Logitech C2xx, choose 3 real-time interactive transformations to compare in terms of average frame rate at a given resolution for a range of at least 3 resolutions in a common aspect ratio (e.g. 4:3 for 1280x960, 640x480, 320x240, 160x120, 80x60) by adding time-stamps (there should be a logging thread) to analyze the potential throughput. You should then get at least 9 separate datasets running 1 transformation at a time. Based on average analysis, pick a reasonable soft real-time deadline (e.g. if average frame rate is 12 Hz, choose a deadline of 100 milliseconds to provide some margin) and convert the processing to SCHED_FIFO and determine if you can meet deadlines with predictability and measure statistical jitter in the frame rate relative to your deadline for both the default scheduler and SCHED_FIFO in each case.

For this problem we decided to go with three real time interactive transformations namely Simple Canny, Hough Elliptical and Hough Line.To accomplish this task we used two threads one for capture and transform and one for logging. We use a mutex lock to lock the shared resources.In order to conduct the analysis we took the time required for every frame, we took 150 total frames for every interactive transformation.To find the jitter,depending on the average of all frames we decided a soft deadline and based on whether each frame meets the deadline we removed the jitter based on how far was the frame from meeting the deadline. As seen in the code 6.1 We are protecting the shared variable of elapsed time with a mutex

## 5.1 Solution Approach

The Image transformations executed in this problem are:

1. Canny Transformation
2. Hough Elliptical
3. Hough Lines

Camera feeds are usually data intensive that depend on the resolution of the camera, frame capture algorithms, processing power. The approach we have implemented to handle large stream of data is to implement the capturing and processing in one thread and logging the timestamps in another thread. We have used the pthread library for the implementation of threads. For logging purpose the library chrono is used with the system clock as the source. The program also makes use of mutexes to protect the critical shared resource in our case the timing variable and a sync variable to help with the synchronization of the logging after the frame capture and transformation.
To make the program modular, the program accepts arguments of the camera device number and the image transformation algorithm as parameters and the arguments are passed to the capture thread to invoke the corresponding transformation algorithm.
We have analysed the image transformation algorithms for 3 different resolutions with the aspect ration of 4:3 namely:

1. 80*60

2. 320*240
3. 1280*960

We have used 2 Scheduling policies, **SCHED_OTHER** and **SCHED_FIFO** to analyse the timing and jitter of the images being capture.

## 5.2 Code Output and Analysis

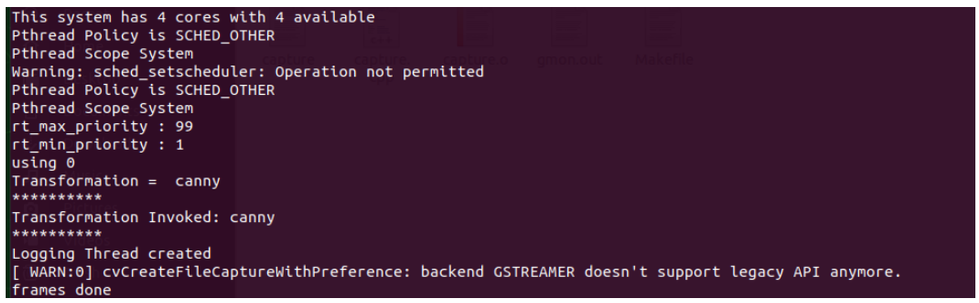



Figure 1: Sched Other

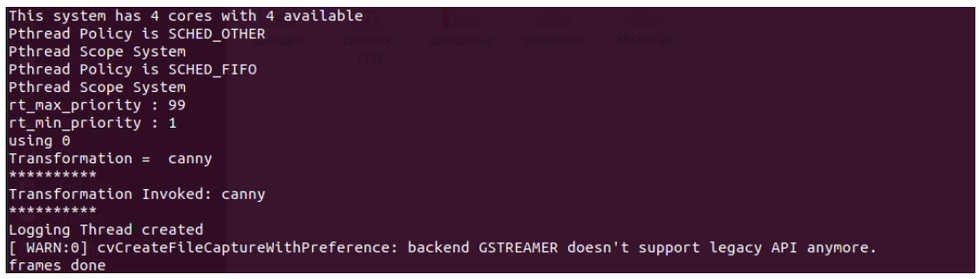



Figure 2: Sched Fifo

For the timing and jitter analysis we take a dataset of 150 frames, using syslog to print the timestamps and have used data analysis tools to analyse and plot the graphs.

```
void *captureThread(void *args)
{
    cout << "**********" << endl;
    cout << "Transformation Invoked: " << (char*)args << endl;
    cout << "**********" << endl;

    char t1[] = "canny";
    char t2[] = "houghelliptical";
    char t3[] = "houghlines";

    CvCapture* capture;
    namedWindow( timg_window_name, CV_WINDOW_AUTOSIZE );
    // Create a Trackbar for user to enter threshold
    createTrackbar( "Min Threshold:", timg_window_name, &lowThreshold, max_lowThreshold,
```

```
    capture = (CvCapture *)cvCreateCameraCapture(dev);
    cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_WIDTH, HRES);
    cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_HEIGHT, VRES);

    while(1)
    {
        auto startTime = std::chrono::system_clock::now();
        frame=cvQueryFrame(capture);
        if(!frame) break;

        if(!memcmp((string*)args, t1, sizeof(t1)))
        {
            CannyThreshold(0, 0);
        }
        else if(!memcmp((string*)args, t2, sizeof(t2)))
        {
            simpleHoughElliptical();
        }
        else if(!memcmp((string*)args, t3, sizeof(t3)))
        {
            simpleHough();
        }
        else
        {
            threadExit = true;
            break;
        }

        pthread_mutex_lock(&mtx);
        auto endTime = std::chrono::system_clock::now();
        elapsed_seconds = endTime - startTime;
        syncVariable = true;
        frames++;
        pthread_mutex_unlock(&mtx);
        if(frames == 150){
            printf("frames done\n");
            cvReleaseCapture(&capture);
            threadExit=true;
            break;
        }

    }

    pthread_exit(NULL);
}
```

During our analysis for different resolutions and different transforms we realised every resolutions has different completion time, different average and different jitter, below is our

observation for jitter analysis for Simple canny transformation:-

### 5.2.1 Canny Transform with FIFO schedule :

Resolution : 80 X 60: As seen in Figure 3 : The frequency is 30 frames per second i.e 30 HZ so we know that it take 33ms for each frame, which would be our soft deadline.

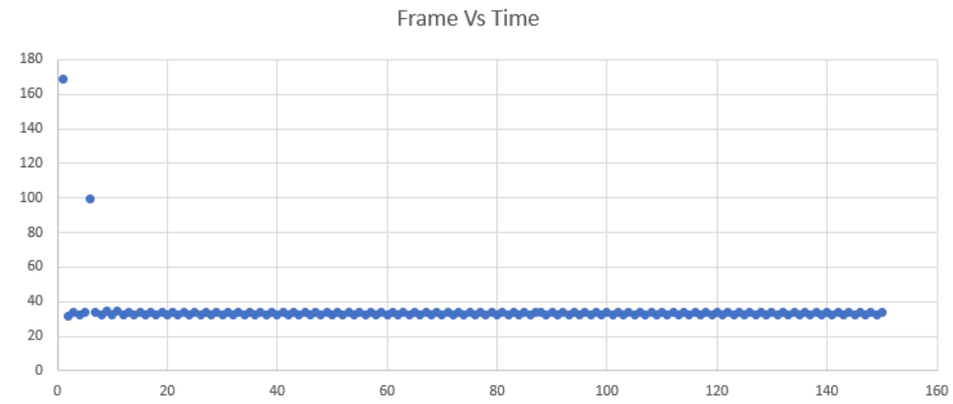



Figure 3: Graph for frames Vs Time

Jitter is how much deviation each frame has from the deadline we can have a tolerance of +-10% so we can see in figure 10 that there our jitter is well inside 10% allowed deadline so all our frames would meet soft deadline.

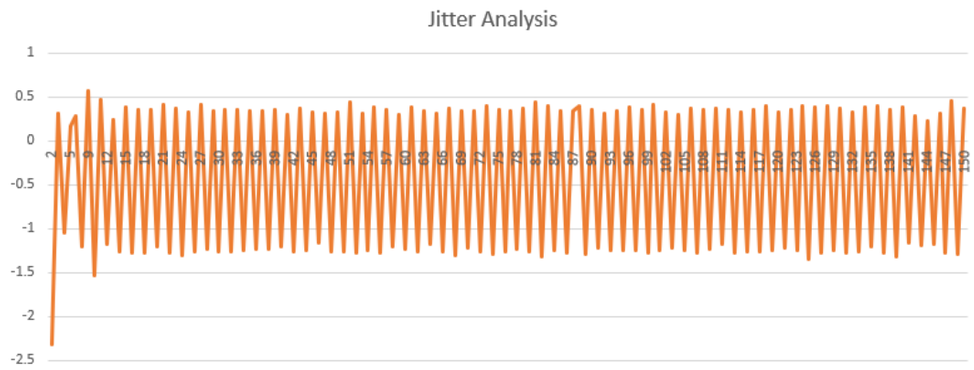



Figure 4: Jitter Analysis for Simple Canny

Resolution: 320 x 240 As seen in Figure 5 : The frequency is 30 frames per second i.e 30 HZ so we know that it take 33ms for each frame, which would be our soft deadline.

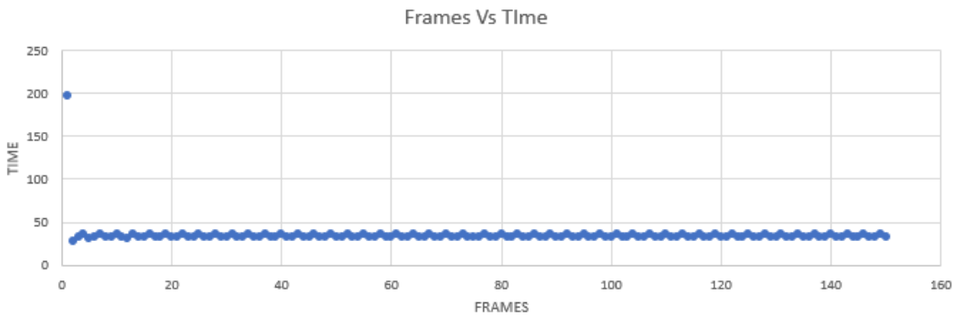



Figure 5: Graph for frames Vs Time

We can see in figure 12 that there our jitter is well inside 10% allowed deadline so all our frames would meet soft deadline. But here you can observe there are few frames that may miss the deadline, as we increase the resolution it takes time to process.

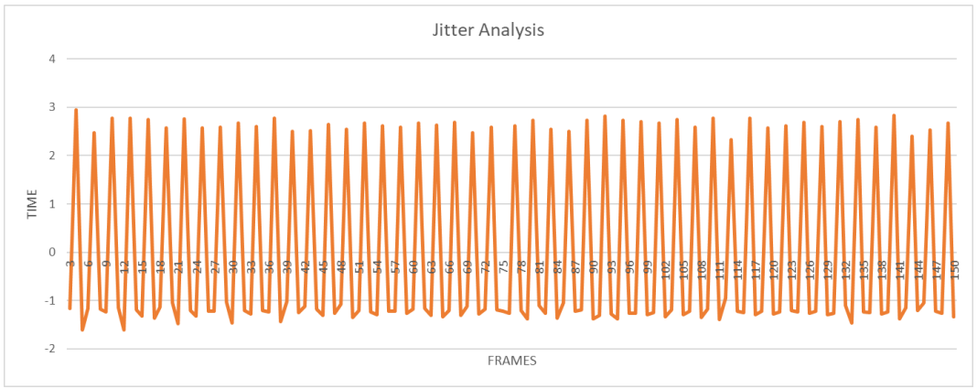



Figure 6: Jitter Analysis for Simple Canny 320x240

Resolution: 1280 x 960 :

As seen in Figure 7 : The frequency is 10 frames per second i.e 10 HZ so we know that it take 100ms for each frame, which would be our soft deadline.

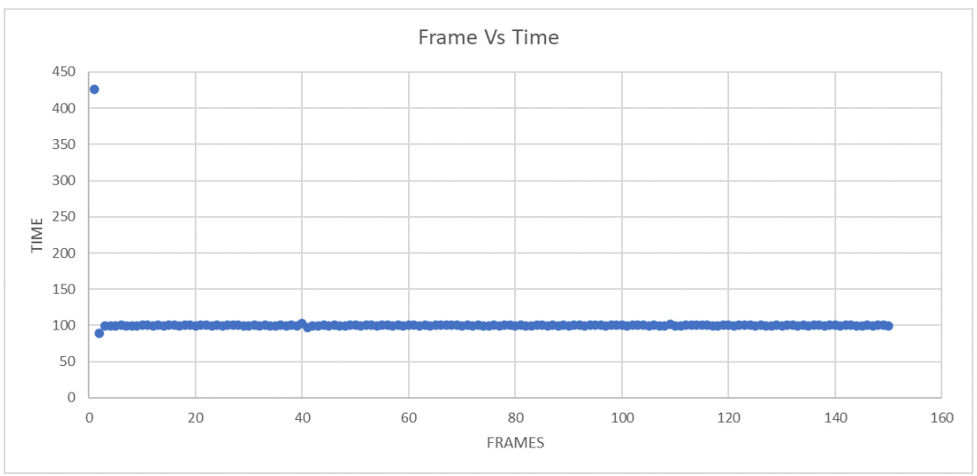



Figure 7: Graph for frames Vs Time

We can see in figure 8 that there our jitter is well inside 10% allowed deadline so all our frames would meet soft deadline. But here you can observe there are few frames that may miss the deadline, as we increase the resolution it takes time to process.

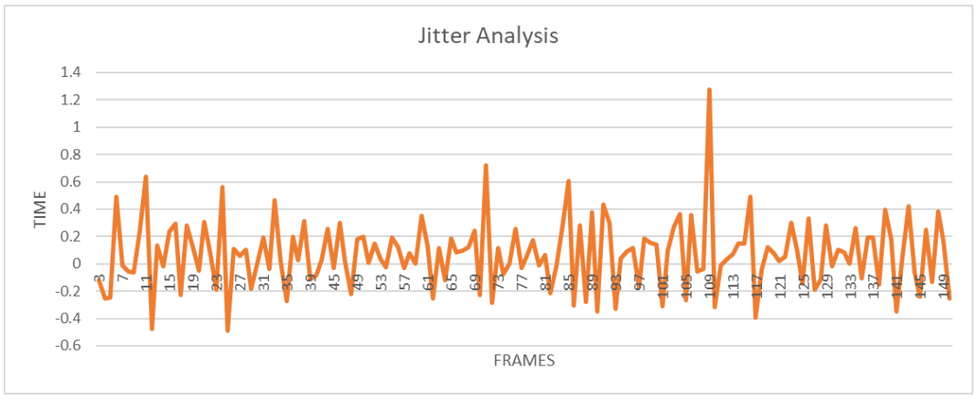



Figure 8: Jitter Analysis for Simple Canny 1280x960

### 5.2.2 Hough Elliptical Transform with FIFO schedule :

Resolution : 80 X 60: As seen in Figure 9 : The frequency is 30 frames per second i.e 30 HZ so we know that it take 33ms for each frame, which would be our soft deadline.

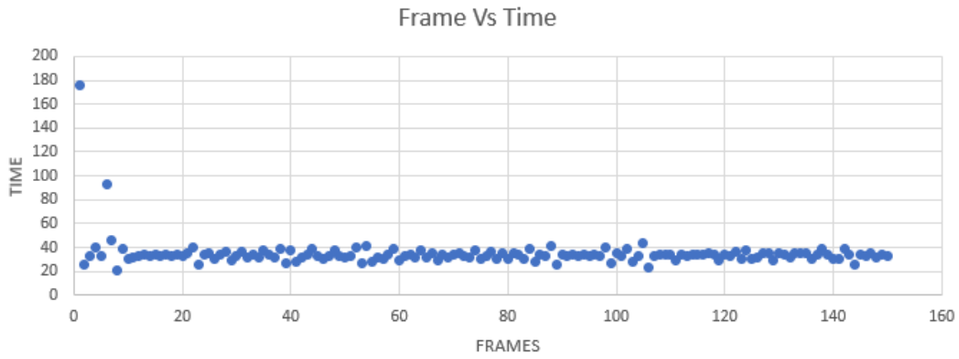



Figure 9: Graph for frames Vs Time

Jitter is how much deviation each frame has from the deadline we can have a tolerance of +-10% so we can see in figure 10 that there our jitter is well inside 10% allowed deadline so all our frames would meet soft deadline.

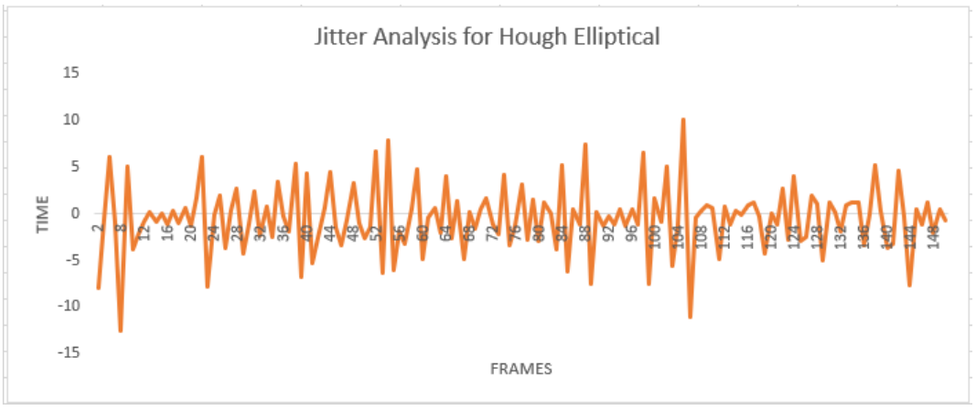



Figure 10: Jitter Analysis for Hough Elliptical with resolution 80x60

Resolution: 320 x 240

As seen in Figure 29 : The frequency is 30 frames per second i.e 30 HZ so we know that it take 33ms for each frame, which would be our soft deadline.

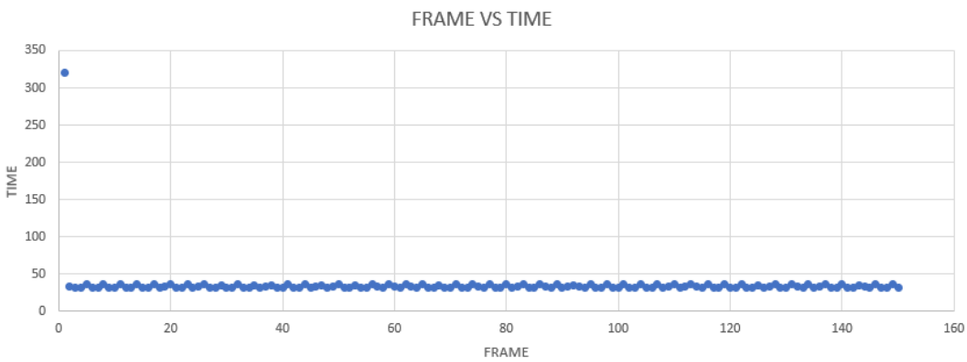



Figure 11: Graph for frames Vs Time

We can see in figure 12 that there our jitter is well inside 10% allowed deadline so all our frames would meet soft deadline. But here you can observe there are few frames that may miss the deadline, as we increase the resolution it takes time to process.

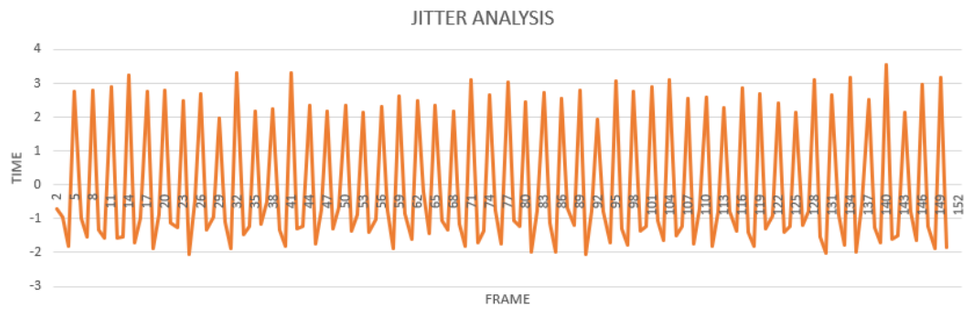



Figure 12: Jitter Analysis for Hough elliptical 320x240

Resolution: 1280 x 960 :

As seen in Figure 37 : The frequency is 10 frames per second i.e 10 HZ so we know that it take 100ms for each frame, which would be our soft deadline.

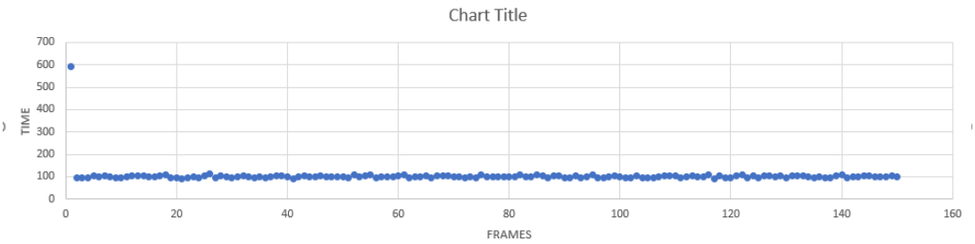



Figure 13: Graph for frames Vs Time

We can see in figure 32 that there our jitter is well inside 10% allowed deadline so all our frames would meet soft deadline. But here you can observe there are few frames that may miss the deadline,a spike shows it would definetly miss our soft deadline, as we increase the resolution it takes time to process.BUt if we see the percentage of frames missing the deadline is really less.

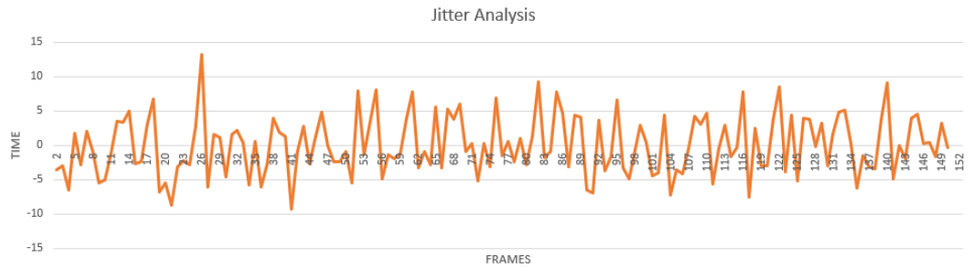



Figure 14: Jitter Analysis for Hough Elliptical 1280x960

### 5.2.3 Hough Line Transform with FIFO schedule :

Resolution : 80 X 60: As seen in Figure 15 : The frequency is 30 frames per second i.e 30 HZ so we know that it take 33ms for each frame, which would be our soft deadline.

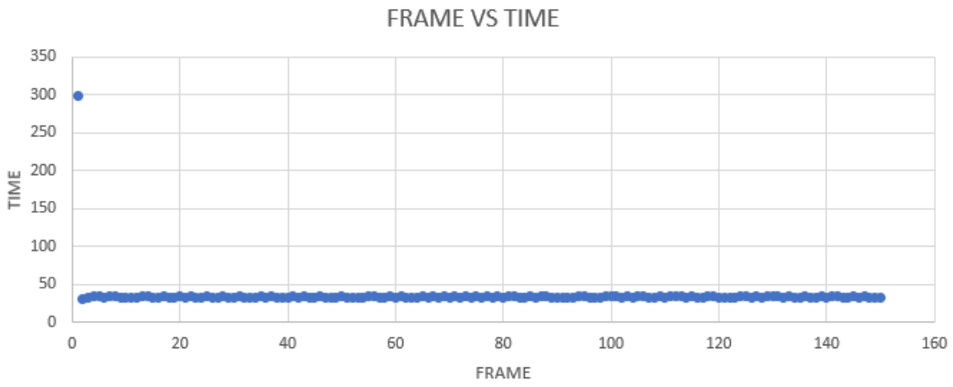



Figure 15: Graph for frames Vs Time

Jitter is how much deviation each frame has from the deadline we can have a tolerance of +-10% so we can see in figure 16 that there our jitter is well inside 10% allowed deadline so all our frames would meet soft deadline.

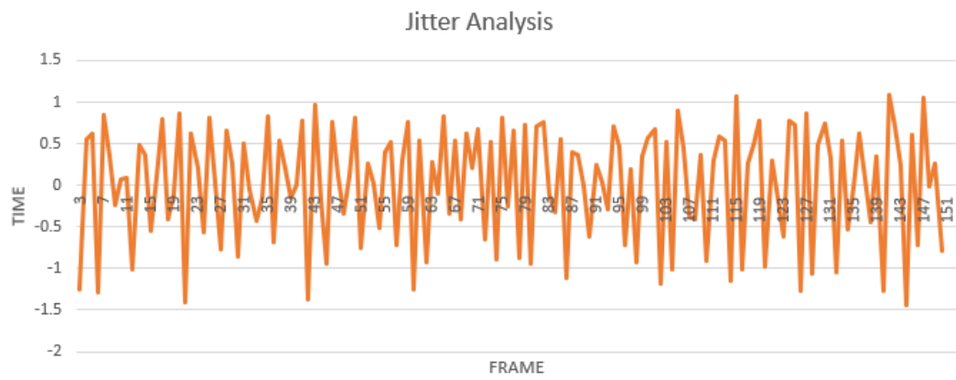



Figure 16: Jitter Analysis for Hough line transform with resolution 80x60

Resolution: 320 x 240

As seen in Figure 35 : The frequency is 30 frames per second i.e 30 HZ so we know that it take 33ms for each frame, which would be our soft deadline.

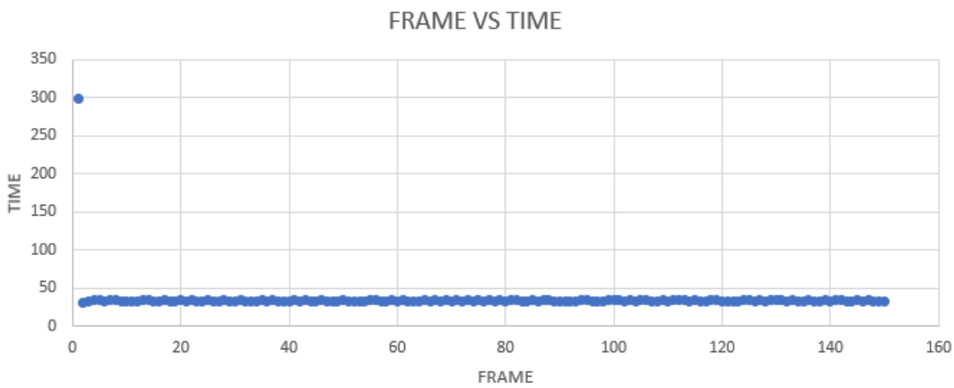



Figure 17: Graph for frames Vs Time

We can see in figure 36 that there our jitter is well inside 10% allowed deadline so all our frames would meet soft deadline. But here you can observe there are few frames that may miss the deadline, and few that do miss the deadline, as we increase the resolution it takes time to process.

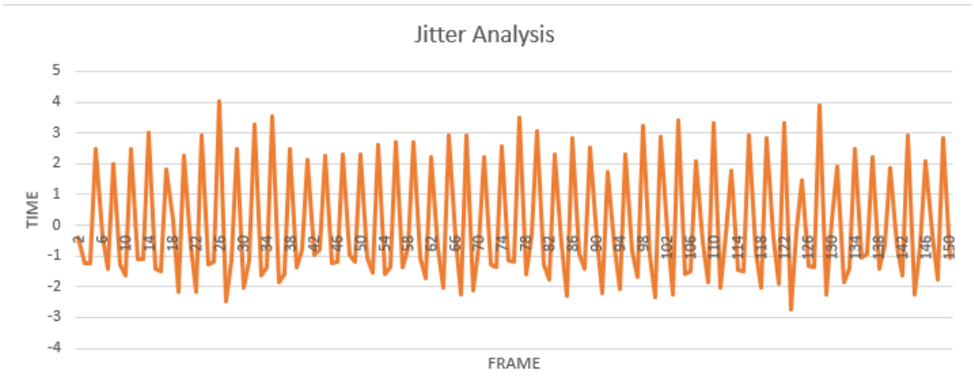



Figure 18: Jitter Analysis for Hough line 320x240

Resolution: 1280 x 960 :

As seen in Figure 19 : The frequency is 9 frames per second i.e 9 HZ so we know that it take 111ms for each frame, which would be our soft deadline.

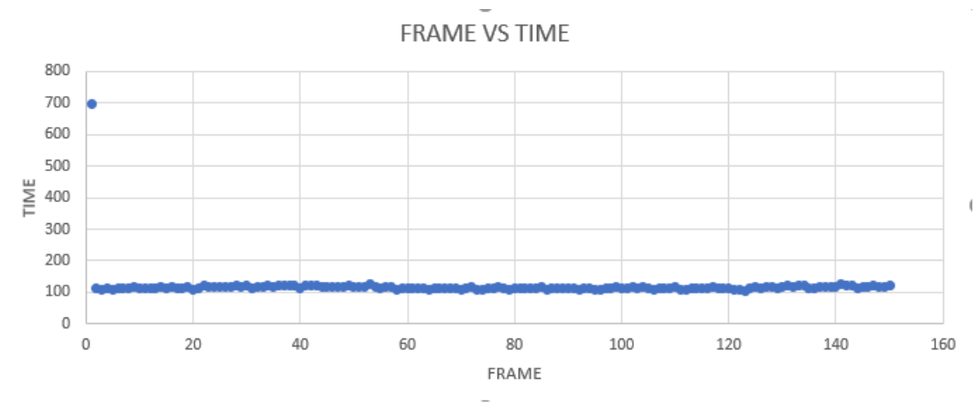



Figure 19: Graph for frames Vs Time

We can see in figure 38 that there our jitter is well inside 10% allowed deadline so all our frames would meet soft deadline.But here you can observe there are few frames that may miss the deadline, as we increase the resolution it takes time to process.BUt if we see the percentage of frames missing the deadline is really less.

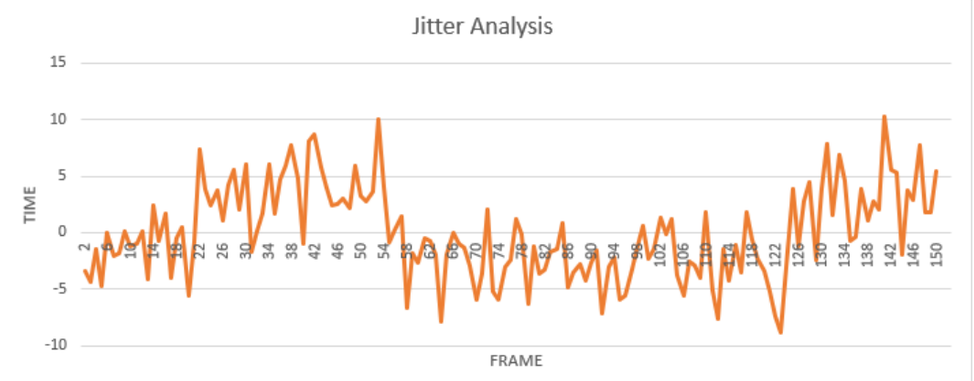



Figure 20: Jitter Analysis for Hough line 1280x960

### 5.2.4 Canny Transform with SCHED_OTHER Default schedule :

Resolution : 80 X 60: As seen in Figure **??** : The frequency is 30 frames per second i.e 30 HZ so we know that it take 33ms for each frame, which would be our soft deadline.

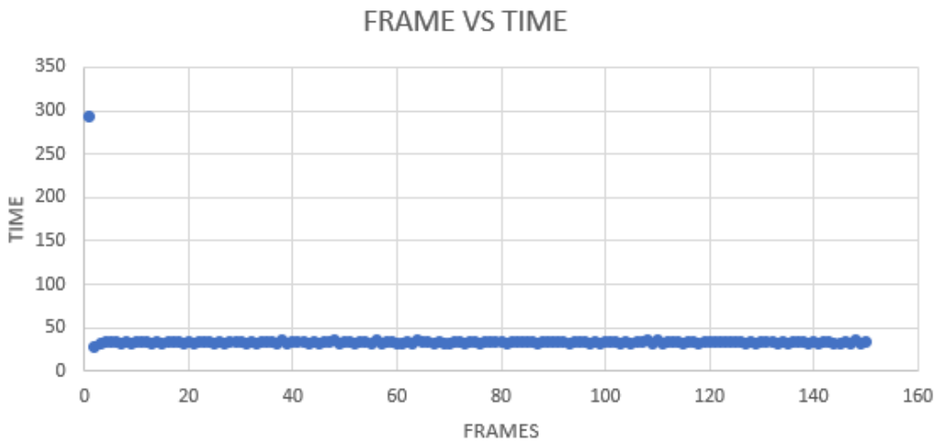



Figure 21: Graph for frames Vs Time

Jitter is how much deviation each frame has from the deadline we can have a tolerance of +-10% so we can see in figure 22 that there our jitter is well inside 10% allowed deadline so all our frames would meet soft deadline.

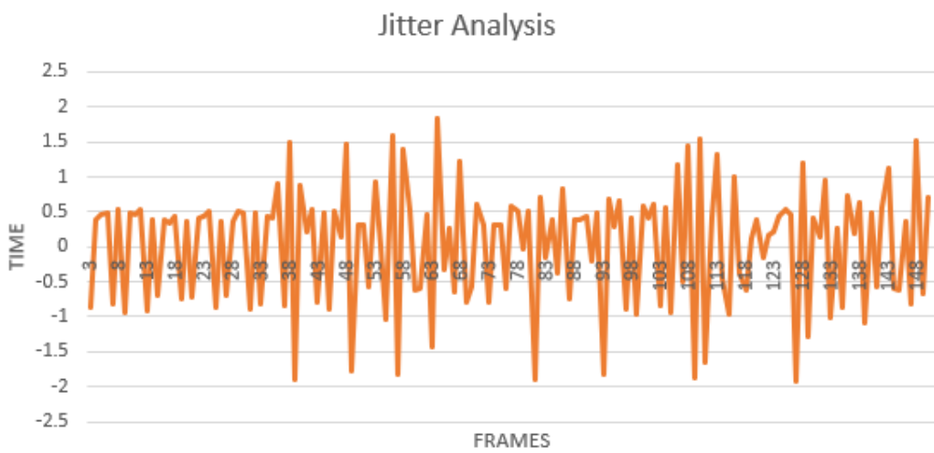



Figure 22: Jitter Analysis for Simple Canny

Resolution: 320 x 240 As seen in Figure 23 : The frequency is 30 frames per second i.e 30 HZ so we know that it take 33ms for each frame, which would be our soft deadline.

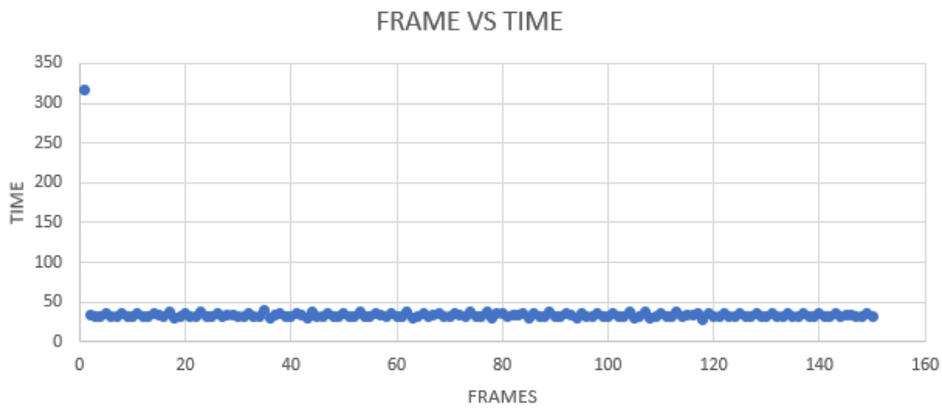



Figure 23: Graph for frames Vs Time

We can see in figure 24 that there our jitter is well inside 10% allowed deadline so all our frames would meet soft deadline. But here you can observe there are few frames that may miss the deadline, as we increase the resolution it takes time to process.

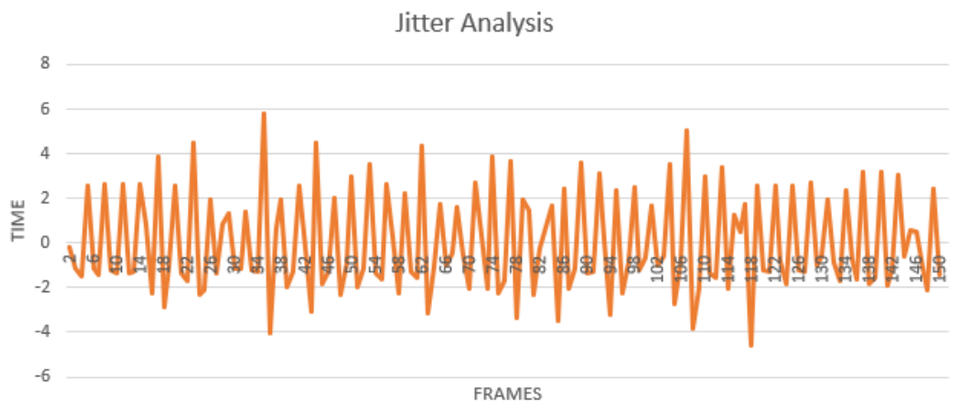



Figure 24: Jitter Analysis for Simple Canny 320x240

Resolution: 1280 x 960 :

As seen in Figure 25 : The frequency is 10 frames per second i.e 10 HZ so we know that it take 100ms for each frame, which would be our soft deadline.

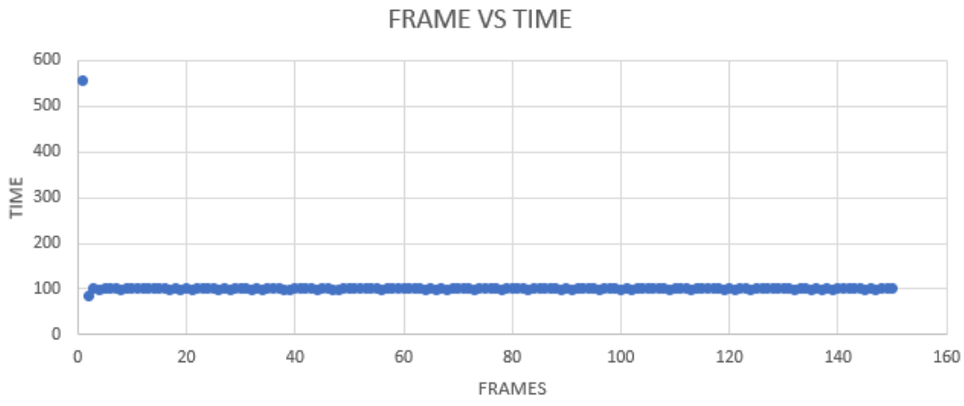



Figure 25: Graph for frames Vs Time

We can see in figure 26 that there our jitter is well inside 10% allowed deadline so all our

frames would meet soft deadline. But here you can observe there are few frames that may miss the deadline, as we increase the resolution it takes time to process.

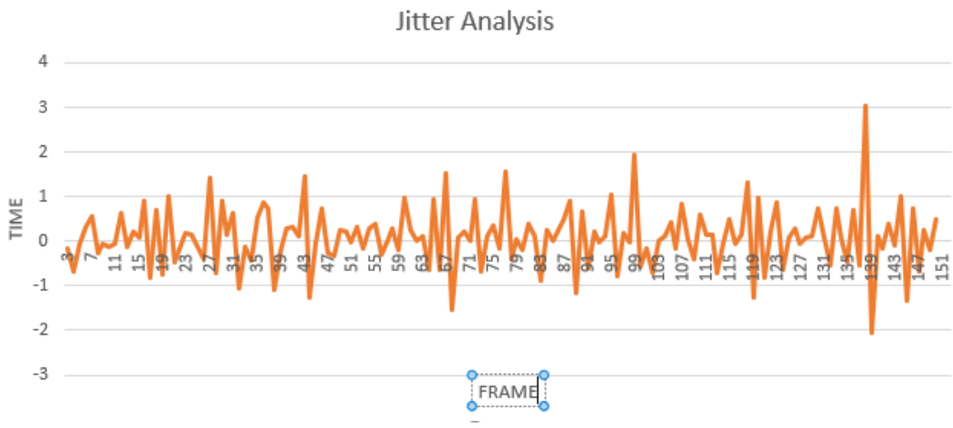



Figure 26: Jitter Analysis for Simple Canny 1280x960

### 5.2.5 Hough Elliptical Transform with SCHED_OTHER Default schedule :

Resolution 80x60: As seen in Figure 27 : The frequency is 30 frames per second i.e 30 HZ so we know that it take 33ms for each frame, which would be our soft deadline.

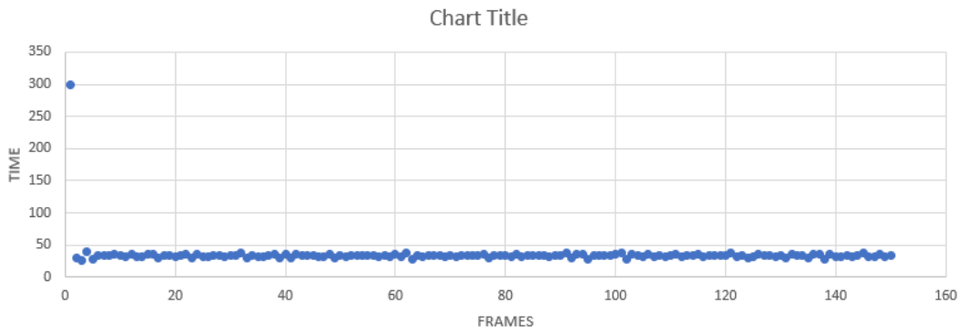



Figure 27: Graph for frames Vs Time

Jitter is how much deviation each frame has from the deadline we can have a tolerance of +-10% so we can see in figure 34 that there our jitter is well inside 10% allowed deadline but some of them do overshoot but overall our frames would meet soft deadline.

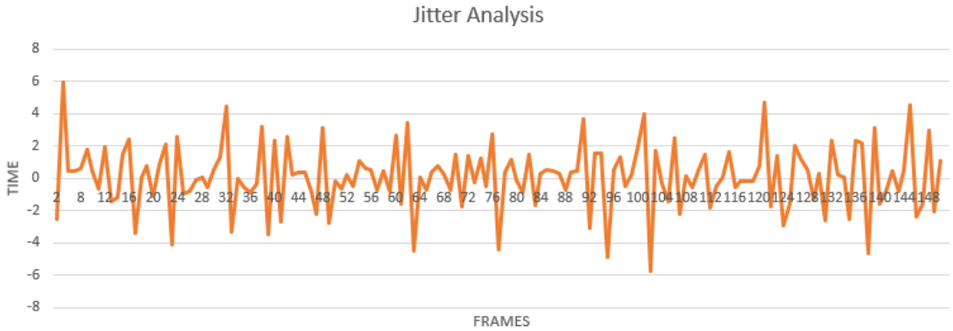



Figure 28: Jitter Analysis for Hough Elliptical

Resolution: 320 x 240 As seen in Figure 29 : The frequency is 30 frames per second i.e 30 HZ so we know that it take 33ms for each frame, which would be our soft deadline.

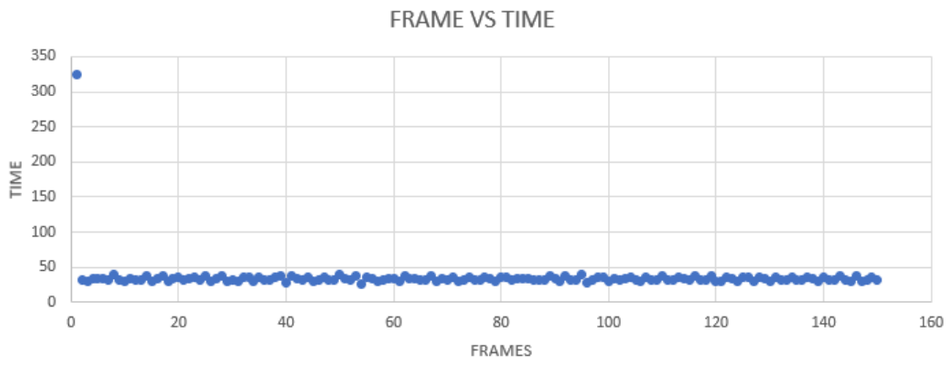



Figure 29: Graph for frames Vs Time

We can see in figure 30 that our jitter is well inside 10% allowed deadline so all our frames would meet soft deadline. But here you can observe there are few frames that may miss the deadline, as we increase the resolution it takes time to process.

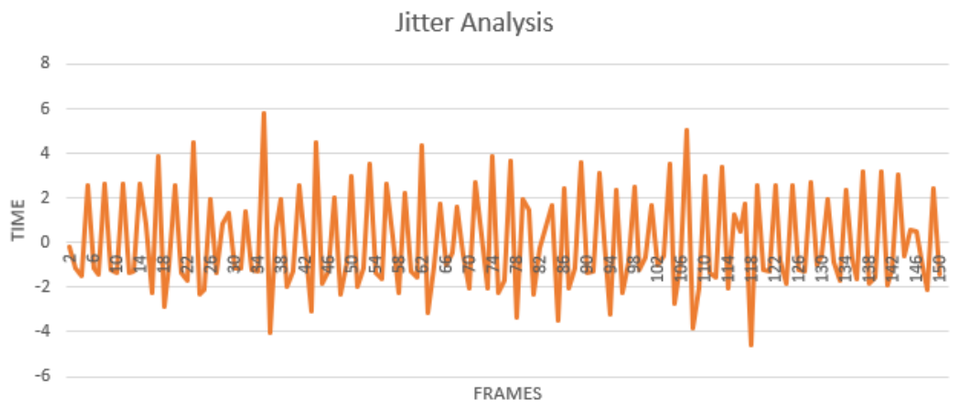



Figure 30: Jitter Analysis for Hough Elliptical 320x240

Resolution: 1280 x 960 :

As seen in Figure **??** : The frequency is 10 frames per second i.e 10 HZ so we know that it take 100ms for each frame, which would be our soft deadline.

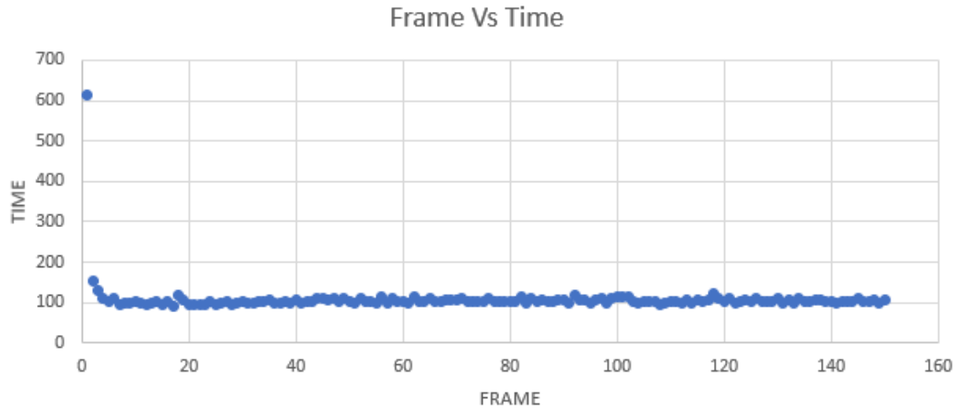



Figure 31: Graph for frames Vs Time

We can see in figure 32 that there our jitter is well inside 10% allowed deadline so all our frames would meet soft deadline. But here you can observe there are few frames that may

miss the deadline, as we increase the resolution it takes time to process.

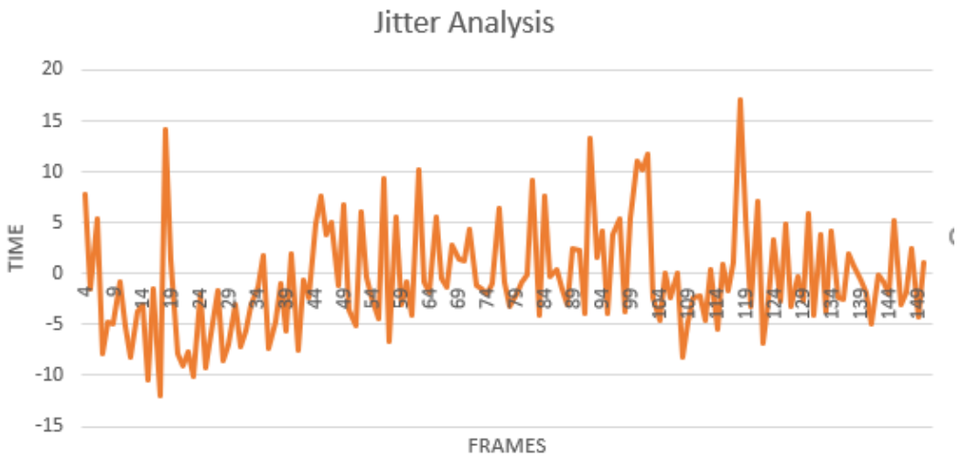



Figure 32: Jitter Analysis for Hough Elliptical 1280x960

### 5.2.6 Hough Line with SCHED_OTHER Default schedule :

Resolution 80x60: As seen in Figure 33 : The frequency is 30 frames per second i.e 30 HZ so we know that it take 33ms for each frame, which would be our soft deadline.

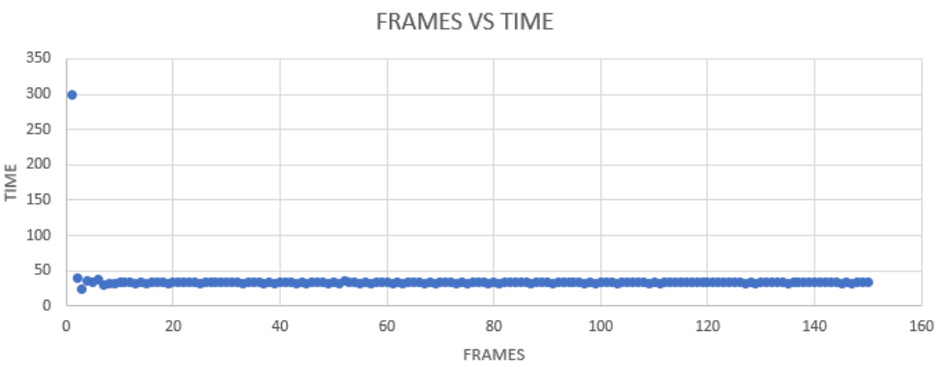



Figure 33: Graph for frames Vs Time

Jitter is how much deviation each frame has from the deadline we can have a tolerance of +-10% so we can see in figure 16 that there our jitter is well inside 10% allowed deadline but some of them do overshoot but overall our frames would meet soft deadline.

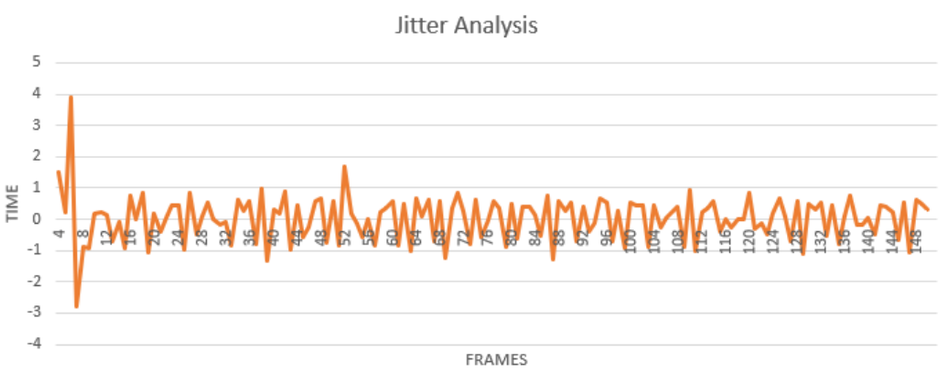



Figure 34: Jitter Analysis for Hough line

Resolution: 320 x 240 As seen in Figure 35 : The frequency is 30 frames per second i.e 30 HZ so we know that it take 33ms for each frame, which would be our soft deadline.

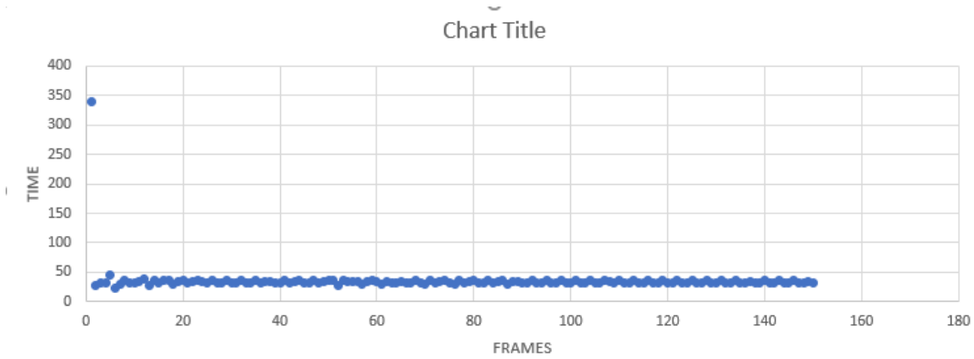



Figure 35: Graph for frames Vs Time

We can see in figure 36 that our jitter is well inside 10% allowed deadline so all our frames would meet soft deadline. But here you can observe there are few frames that may miss the deadline, as we increase the resolution it takes time to process.

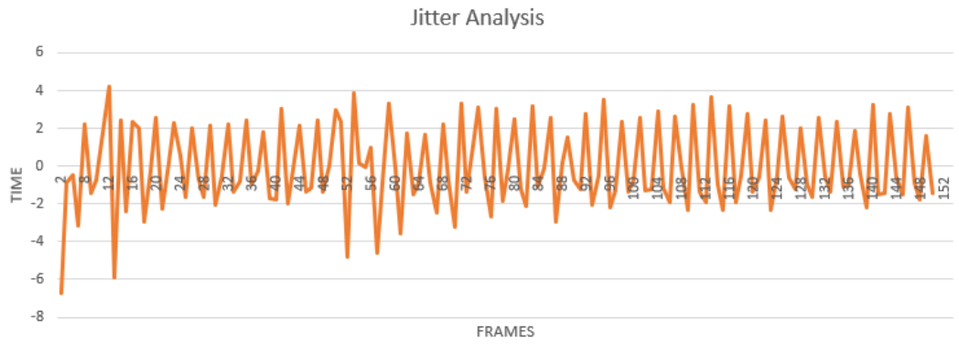



Figure 36: Jitter Analysis for Hough line 320x240

Resolution: 1280 x 960 :

As seen in Figure **??** : The frequency is 10 frames per second i.e 8 HZ so we know that it take 125ms for each frame, which would be our soft deadline.

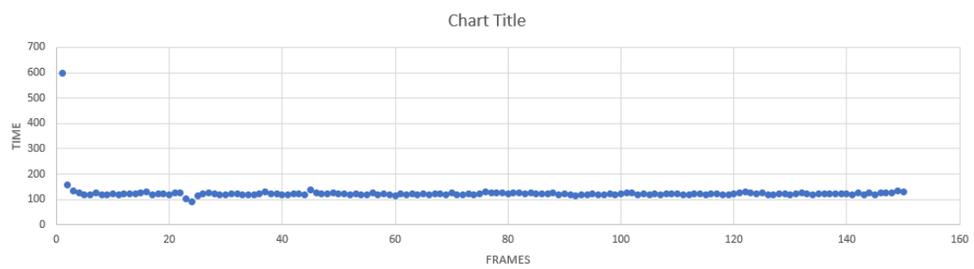



Figure 37: Graph for frames Vs Time

We can see in figure 38 that there our jitter is well inside 10% allowed deadline so all our frames would meet soft deadline. But here you can observe there are few frames that may miss the deadline, as we increase the resolution it takes time to process.

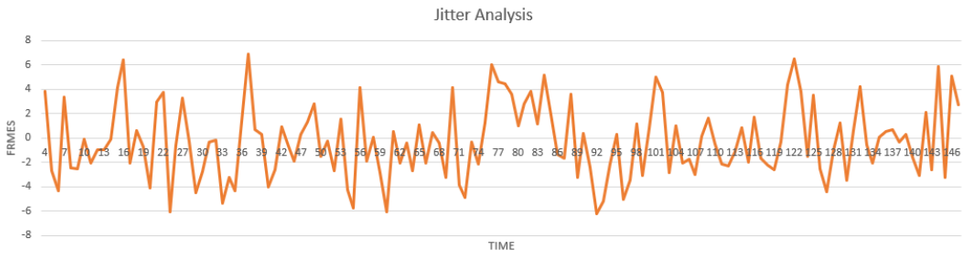



Figure 38: Jitter Analysis for Hough line 1280x960

# 6 Appendix- Code

## 6.1 Problem 5

### 6.1.1 Main Function

```
/*
 *
 *  Example by Sam Siewert
 *
 */
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <sched.h>
#include <sys/sysinfo.h>
#include <sys/types.h>
#include <syslog.h>


#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>

#include <chrono>
#include <pthread.h>

using namespace cv;
using namespace std;

#define HRES 80
#define VRES 60
#define SCHED (SCHED_FIFO)
int dev=0;
char argumentValue[20];
```

```
#define NUM_OF_THREADS 2

pthread_attr_t rt_sched_attr[NUM_OF_THREADS];
int rt_max_priority;
int rt_min_priority;
struct sched_param rt_param[NUM_OF_THREADS];
struct sched_param main_param;
pthread_attr_t main_attr;
pid_t mainpid;


// Transform display window
char timg_window_name[] = "Edge Detector Transform";
int lowThreshold=0;
int const max_lowThreshold = 100;
int kernel_size = 3;
int edgeThresh = 1;
int ratioForImage = 3;
Mat canny_frame, cdst, timg_gray, timg_grad;

void CannyThreshold(int, void*);
void simpleHoughElliptical(void);
void simpleHough(void);

IplImage* frame;
int frames;
std::chrono::duration<double, std::milli> elapsed_seconds;

/*mutex*/
pthread_mutex_t mtx;

bool syncVariable = false;
bool threadExit = false;

void *loggingThread(void *args)
{
    cout << "Logging Thread created" << endl;
    while(1)
    {
        if(threadExit == true)
        {
            break;
        }
        if(syncVariable == true)
        {
            syncVariable = false;
            pthread_mutex_lock(&mtx);
            //cout << "End - Start " << elapsed_seconds.count() * 1000 << " ms" << end
```

```
            //cout << "End - Start " << elapsed_seconds.count() << " ms" << endl;
            syslog(LOG_NOTICE, "Frame %d complete in %lf ms\n", frames, elapsed_seconds)

            pthread_mutex_unlock(&mtx);
        }
        else
        {
            ;
        }
    }

    pthread_exit(NULL);

}

void *captureThread(void *args)
{
    cout << "**********" << endl;
    cout << "Transformation Invoked: " << (char*)args << endl;
    cout << "**********" << endl;

    char t1[] = "canny";
    char t2[] = "houghelliptical";
    char t3[] = "houghlines";

    CvCapture* capture;
    namedWindow( timg_window_name, CV_WINDOW_AUTOSIZE );
    // Create a Trackbar for user to enter threshold
    createTrackbar( "Min Threshold:", timg_window_name, &lowThreshold, max_lowThreshold,

    capture = (CvCapture *)cvCreateCameraCapture(dev);
    cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_WIDTH, HRES);
    cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_HEIGHT, VRES);

    while(1)
    {
        auto startTime = std::chrono::system_clock::now();
        frame=cvQueryFrame(capture);
        if(!frame) break;

        if(!memcmp((string*)args, t1, sizeof(t1)))
        {
            CannyThreshold(0, 0);
        }
        else if(!memcmp((string*)args, t2, sizeof(t2)))
        {
            simpleHoughElliptical();
        }
```

```
        else if(!memcmp((string*)args, t3, sizeof(t3)))
        {
            simpleHough();
        }
        else
        {
            threadExit = true;
            break;
        }

        pthread_mutex_lock(&mtx);
        auto endTime = std::chrono::system_clock::now();
        elapsed_seconds = endTime - startTime;
        syncVariable = true;
        frames++;
        pthread_mutex_unlock(&mtx);
        if(frames == 150){
            printf("frames done\n");
            cvReleaseCapture(&capture);
            threadExit=true;
            break;
        }

    }

    pthread_exit(NULL);
}



void CannyThreshold(int, void*)
{

    //Mat mat_frame(frame);
    Mat mat_frame = cv::cvarrToMat(frame);

    cvtColor(mat_frame, timg_gray, CV_RGB2GRAY);

    /// Reduce noise with a kernel 3x3
    blur( timg_gray, canny_frame, Size(3,3) );

    /// Canny detector
    Canny( canny_frame, canny_frame, lowThreshold, lowThreshold*ratioForImage, kernel_si

    /// Using Canny's output as a mask, we display our result
    timg_grad = Scalar::all(0);

    //cout << "Size = " << sizeof(frame) << endl;
```

```
    mat_frame.copyTo( timg_grad, canny_frame);

    imshow( timg_window_name, timg_grad );


}


/*Printing Scheduler Details*/
void print_scheduler(void)
{
    int schedType;
    int scope;

    schedType = sched_getscheduler(getpid());

    switch(schedType)
    {
        case SCHED_FIFO:
            cout << "Pthread Policy is SCHED_FIFO" << endl;
        break;
        case SCHED_OTHER:
            cout << "Pthread Policy is SCHED_OTHER" << endl;
        break;
        case SCHED_RR:
            cout << "Pthread Policy is SCHED_RR" << endl;
        break;

        default:
            cout << "Pthread Policy is unknown" << endl;
        break;

    }

    pthread_attr_getscope(&main_attr, &scope);

    if(scope == PTHREAD_SCOPE_SYSTEM)
    {
        cout << "Pthread Scope System" << endl;
    }
    else if(scope == PTHREAD_SCOPE_PROCESS)
    {
        cout << "Pthread Scope Process" << endl;
    }
    else
    {
        cout << "Pthread Scope Unknown" << endl;
```

```
    }
}
```

```
int main( int argc, char** argv )
{
    int rc;

    openlog("ECEN REAL TIME:", 0, 0);
    syslog(LOG_NOTICE, "\nTEST START\n");
    cout << "This system has " << get_nprocs_conf() << " cores with " << get_nprocs() <<

    /*Obtain the process ID*/
    mainpid = getpid();

    rt_max_priority = sched_get_priority_max(SCHED);
    rt_min_priority = sched_get_priority_min(SCHED);

    print_scheduler();

    rc = sched_getparam(mainpid, &main_param);

    main_param.sched_priority = rt_max_priority;

    if(rc = sched_setscheduler(getpid(), SCHED_FIFO , &main_param) < 0)
    {
        perror("Warning: sched_setscheduler");
    }
    print_scheduler();

    cout << "rt_max_priority : " << rt_max_priority << endl;
    cout << "rt_min_priority : " << rt_min_priority << endl;



    if(argc > 1)
    {
        sscanf(argv[1], "%d", &dev);
        printf("using %s\n", argv[1]);
        sscanf(argv[2], "%s", argumentValue);
        //strncpy(argumentValue , argv[2], s)
        printf("Transformation =  %s\n", argumentValue);
    }
```

```
else if(argc == 1)
    printf("using default\n");

else
{
    printf("usage: capture [dev]\n");
    exit(-1);
}


rc = pthread_attr_init(&rt_sched_attr[0]);
rc = pthread_attr_setinheritsched(&rt_sched_attr[0], PTHREAD_EXPLICIT_SCHED);
rc = pthread_attr_setschedpolicy(&rt_sched_attr[0] , SCHED_FIFO);

rt_param[0].sched_priority = rt_max_priority - 1;
pthread_attr_setschedparam(&rt_sched_attr[0] , &rt_param[0]);


pthread_t capThread;
pthread_create(&capThread, NULL , captureThread, (void*)argumentValue);


rc = pthread_attr_init(&rt_sched_attr[1]);
rc = pthread_attr_setinheritsched(&rt_sched_attr[1], PTHREAD_EXPLICIT_SCHED);
rc = pthread_attr_setschedpolicy(&rt_sched_attr[1] , SCHED_FIFO);

rt_param[1].sched_priority = rt_max_priority - 1;
pthread_attr_setschedparam(&rt_sched_attr[1] , &rt_param[1]);
pthread_t logThread;
pthread_create(&logThread, NULL , loggingThread, NULL);


pthread_join(capThread, NULL);
pthread_join(logThread, NULL);

while(1)
{
    if(threadExit)
    {
        break;
    }
}


cvDestroyWindow(timg_window_name);
syslog(LOG_NOTICE, "\nThread Exit\n");
```

```
};



/*Transformations*/

void simpleHoughElliptical(void)
{
    Mat gray;
    vector<Vec3f> circles;
    Mat mat_frame = cv::cvarrToMat(frame);
    cvtColor(mat_frame, gray, CV_BGR2GRAY);
    GaussianBlur(gray, gray, Size(9,9), 2, 2);

    HoughCircles(gray, circles, CV_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 );
    }



    if(!frame)
    {
        return;
    }

    //cvShowImage(timg_window_name, frame);
    imshow("Capture Example", mat_frame);

}

void simpleHough(void)
{
    vector<Vec4i> lines;
    Mat canny_frame;
    Mat mat_frame = cv::cvarrToMat(frame);
    Canny(mat_frame, canny_frame, 50, 200, 3);

    //cvtColor(canny_frame, cdst, CV_GRAY2BGR);
    //cvtColor(mat_frame, gray, CV_BGR2GRAY);
```

```
    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, CV_AA)
    }


    if(!frame)
    {
        return;
    }
    imshow("Capture Example", mat_frame);
    //cvShowImage(timg_window_name, frame);
}
```

### 6.1.2 Makefile

```
INCLUDE_DIRS =
LIB_DIRS =
CC=g++

CDEFS=
CFLAGS= -O0 -pg -g $(INCLUDE_DIRS) $(CDEFS)
LIBS= -lrt
CPPLIBS= -L/usr/lib -lopencv_core -lopencv_flann -lopencv_video -lpthread

HFILES=
CFILES=
CPPFILES= capture.cpp

SRCS= ${HFILES} ${CFILES}
CPPOBJS= ${CPPFILES:.cpp=.o}

all:        capture

clean:
        -rm -f *.o *.d
        -rm -f capture

distclean:
        -rm -f *.o *.d

capture: capture.o
        $(CC) $(LDFLAGS) $(CFLAGS) -o $@ $@.o `pkg-config --libs opencv` $(CPPLIBS)
```

```
depend:

.c.o:
        $(CC) $(CFLAGS) -c $<

.cpp.o:
        $(CC) $(CFLAGS) -c $<
```

# 7 Appendix- Image

# Appendix- References

1. Exercise document
2. Open CV Documentation
3. Code Source