Real Time Embedded System

Final Extended Lab

Time Lapse Image Acquisition

Report By

-Vishvesh Raj Singh

-Raghunath Jagnam

INDEX

1 Introduction 3
2 Functional Requirements 3
3 Real-Time Requirements 4
4 Functional Design Overview and Diagrams 5
5 Real-Time Analysis and Design with Timing Diagrams, both measured and expected based upon theory 7
6 Proof-of-Concept with Example Output and Test Completed 9
7 Conclusion 11
8 Formal References 11
9 Appendices with results, code and supporting material 12

1 INTRODUCTION

As a part of the extended lab for RTES 5623, we decided to do Time Lapse Image Acquisition as summarized in the extended lab requirements.

For the completion of the extended lab, the host system would run Linux Ubuntu, and the target system would be NVIDIA Jetson TK1 Embedded development kit. Logitech camera C200 will be used to capture multiple frames at different rates with default resolution 640x480 and later save them at different frequency as instructed in the extended lab guidelines.

The code written for the extended lab has made use of some functions from openCV libraries and RT Linux. The function used from these libraries are explained in the report with references. A detail analysis of the entire extended lab is written along with some of the difficulties faced in the completion of the extended lab.

2 FUNCTIONAL REQUIREMENTS

There are four functional requirements outlined as a part of extended lab requirement.

The first functional requirement includes to obtain a frame with a given resolution of 640x480 as a MINIMUM REQUIREMENT.

- A series a frames needs to be captured from webcam, which is interfaced to the platform by USB. And the frames needs to be captured either at 1 Hz or 24 Hz. The image will be saved in the PPM format with the appropriate header.
- In order to get accuracy in capturing the images, PIT (Programmable Interval Timer) will be used with reference of POSIX clock along with proper sleep mechanism, to obtain the frames at desired frequency. The clock-gettime function will be used for time stamps, which will eventually embedded into the PPM header of the frames as a comment field.
- The frame capture time will be displayed on the terminal, used for confirming the deadlines. The accuracy of the code/program will be determined from these output displayed on the terminal.

The second functionality of the extended lab mainly deals with the verification of the previous functionality on the basis of examples.

- In the second functionality, the accuracy of capturing the frames in the previously functionality is confirmed/verified in this functionality.
- The example video was taken to verify the accuracy, where an ICE cube is taken and it is observed with respect to an external clock, for an approximate amount/period 30 minutes.
- Another example of video of an external event of a sunset with a foreground of a parking lot. We chose this event as it involves both light intensity change and motion.

The third functionality as TARGET GOAL adds further complexity to the project.

- As a part of the third functionality, the frames obtain in the first functionality will be compressed on the target and have less to transfer over the Ethernet.

The fourth functionality as STRECTCH GOALS

- As a part of the stretch requirement, the frames are captured at a much higher rates up to 10 Hz along with continuous download for almost 9 minutes, which will store almost 6000 frames. The jitter and accumulated verification at this higher rate will be repeated.

The fifth functionality as a part of the MINIMUM EXTRA REQUIREMENT

- There will be a mechanism for a motion detection of a particular shape and color, which will trigger the time lapse acquisition followed by all the requirements.

3 REAL-TIME REQUIREMENTS

As a part of the Real Time Requirements, there will be two threads without including the main thread, as per the requirements mentioned.

In the beginning, there is a motion detection function, which basically checks for a round object. If the camera detects a round object, then the process starts, or otherwise it will wait for a round object detection.

The first thread will capture a frame of the given resolution 640x480 live from a webcam. The frames will be saved at a rate of 1 Hz (1 frame per second) using Programmable Interval Counter and POSIX clock for accurate measurement of time.

The first thread runs for almost 33 minutes and 33 seconds, capturing almost 2000 frames. The images were stored in PPM format, and time stamps were embedded into every frame as a comment to clearly identify the system.

High accuracy in capturing the frames at 1 frame per second was achieved by using the POSIX clock, and this was confirmed by observing two phenomenon. First, where an ICE cube was kept with reference to the clock, and second one is the sunset phenomenon.

The second thread performed the compression of more than 2000 frames to store them on the target, and have less data to transfer over Ethernet.

The third thread will capture frame at a higher rate than 1 Hz with continuous download. The third frame was made to run for 9 minutes capturing almost 6000 frames. The jitter and the latency for capturing the frames were verified at this higher rates.

The jitter and accumulated latency will be calculated for the threads in order to determine the deterministic nature of the system.

4 FUNCTIONAL DESIGN OVERVIEW AND DIAGRAMS

Flowchart

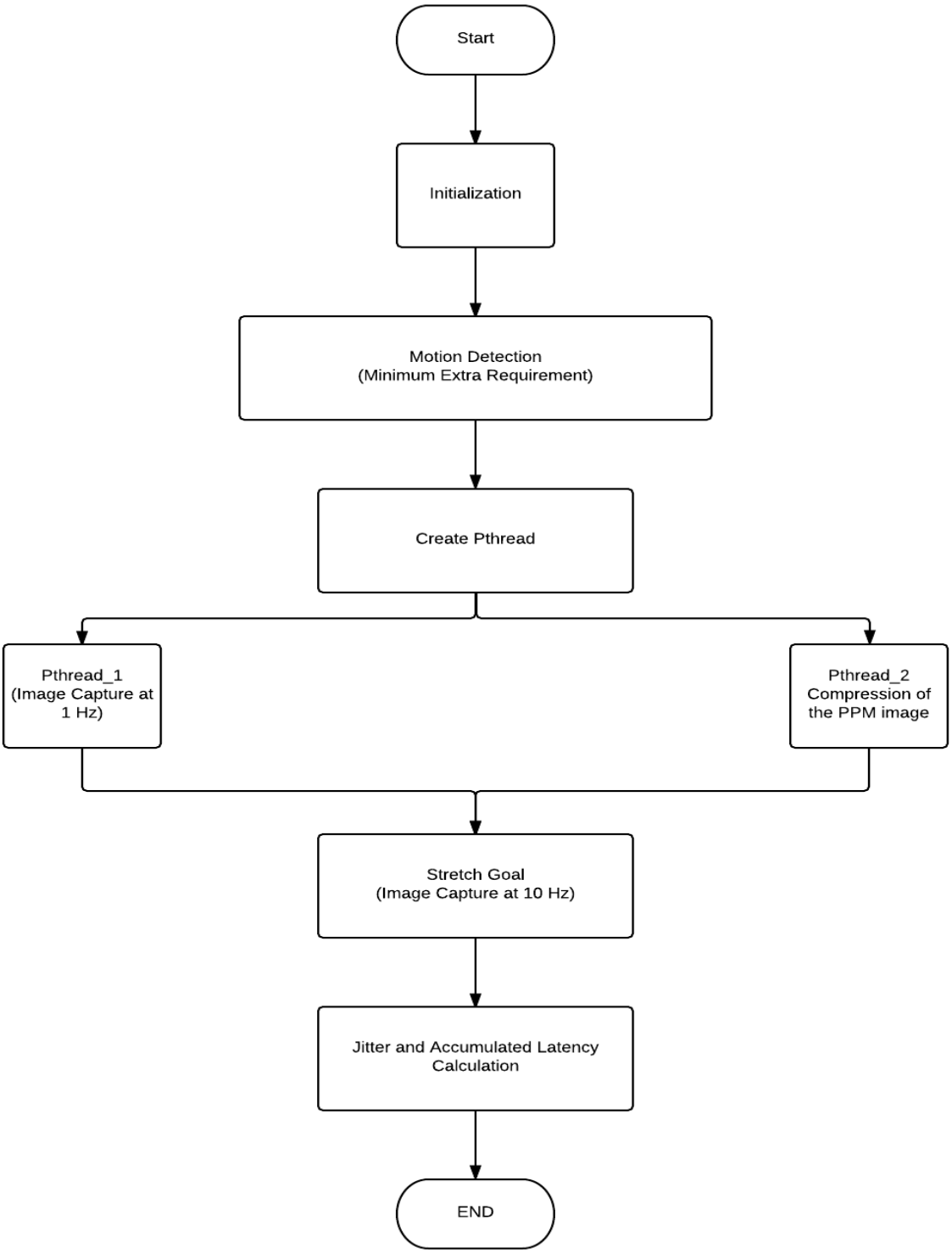


Figure 4.1 Flowchart of the Functional Design of the Extended Lab

As already mentioned in the requirements, the program starts with initializing of the attributes of the pthreads, setting inheritance and setting the scheduling policy to SCHED_FIFO from SCHED_OTHER.

Then the maximum and the minimum priorities of the FIFO are initialized, along with assigning priorities to the threads. There are three threads in the program. First thread performs the frame capture at 1 Hz in the PPM format. Second thread performs the compression of the image from PPM to JPEG. And the third thread performs the frame capture at 10 Hz. The highest priority is assigned to the third thread as we are using RM policy in which most frequently occurring task is given highest priority and the second highest priority is assigned to the first thread as we found after intense testing of the timing that ppm takes less time and thus this is more frequent to occur though their deadlines are same and the third highest priority is assigned to second thread.

As a part of the Minimum Extra Requirement, a circle detection technique is used to start the image capture. The idea here is that whenever there is a motion detected by camera, all the three pthread will be created and the image capturing starts. In the motion detection we used Hough circle transform to achieve it. Initially a frame is grabbed from the camera and later it is converted into gray scale to transformations in a much simpler way. Later we used a Gaussian blur to remove any false detections that can occur. Later actual operation of finding circles using Hough circles is done with Upper threshold of 200 for the internal Canny edge detector and 100 for Threshold for center detection. Finally when a circle is detected a blue boundary with green center is drawn on the original frame.

The first thread, which captures 1 frame per second, will capture 2000 frames in a row. These frames captured will be saved in the PPM format along with the proper header. The header in the PPM image is further edited, and the time stamps are included in the header along with other details. The time stamps show the value of the time when it was captured obtain from Linux's clock_gettime function. Every PPM image will have their header updated with the proper timing information.

The second thread, which also captured the 1 frame per second, stores the frame in JPEG format rather than PPM. The frames saved in both the first and second thread are the same, with the only difference in the format.

As there is a capturing of same frame in both the threads, thus there is a mutex in the program which protects the sharing of the shared resource, and so the frame stored does not get corrupted.

The third threads store frames at a higher rate of almost 10 Hz. The frames stored in the JPEG format. The thread runs for 9 minutes, storing almost 6000 frames in total.

One of the important key feature of the program is the algorithm, which has been used to store the frames. The algorithm does not make use of the frame rates or other factor while storing the frames, it makes use of the time obtain from Linux's clock_gettime function, and thus assures that frames are captured at the right period. Other factors like frame rates etc. have a tendency to produce additional latency to the system due to many factors, but by using time as reference, the chances of such latency and jitter reduces to a great extent. Thus, this key feature makes the system more reliable and predictable as compared to others.

For the in depth analysis, the jitter calculation for every frame, and a final accumulated latency verification in the code is done, which will serve as a criterion in defining the deterministic behavior of the system.

As a part of the extended lab we had to hit many road blocks in the way. We can list of few of them in this section.

1. MIT magic cookie
   We ran into this problem when we tried to run our programs as sudo as this would be a requirement as scheduling policy would not change without sudo .Later changing/removing Xauthority files we were able solve that.
2. openCV – CvframeQuery();
   This particular function of frame query is taking a lot of time around 100ms seconds which is most exhausting task done by NVIDIA jetson, to rectify this we used CVgrabframe instead , it was not a 100% perfect solution but it was still a solution.
3. Memory resources
   After completing around 25 minutes of the video we ran to errors related to memory resources, we later realized it was because of few variables which we changed into double caused.

5 Real-Time Analysis and Design with Timing Diagrams, both measured and expected based upon theory

As a part of the Real Time Analysis and Design, several techniques were used to verify the system predictability and deterministic nature.

Cheddar Analysis

The entire extended lab is divided into three parts

1.Motion/circle detection – No pthreads this is program is just a trigger for the main functionality.

2. We used three pthreads for the time lapse part of the code

i. In the first pthread we use to save the ppm images at 1HZ

ii. We save the frames captured at 1 HZ.

iii. In this part we used a single thread to capture camera frames at 10 Hz

Time Lapse at 1 Hz

In this we have used the three real time events

1. To save the ppm image at 1Hz

2. To save jpg image at 1 Hz.

3. To save jpg image at 10Hz

We plan to use RM policy for scheduling the three tasks. After a thorough analysis of the time taken to store jpg and ppm we realized the time to encode a jpg image takes more time than a general ppm image. So we have given 1 Hz ppm more priority than 1Hz jpg and according to RM policy 10Hz jpg image saving task has the highest priority.

Calculating values of T, C for storing the ppm image

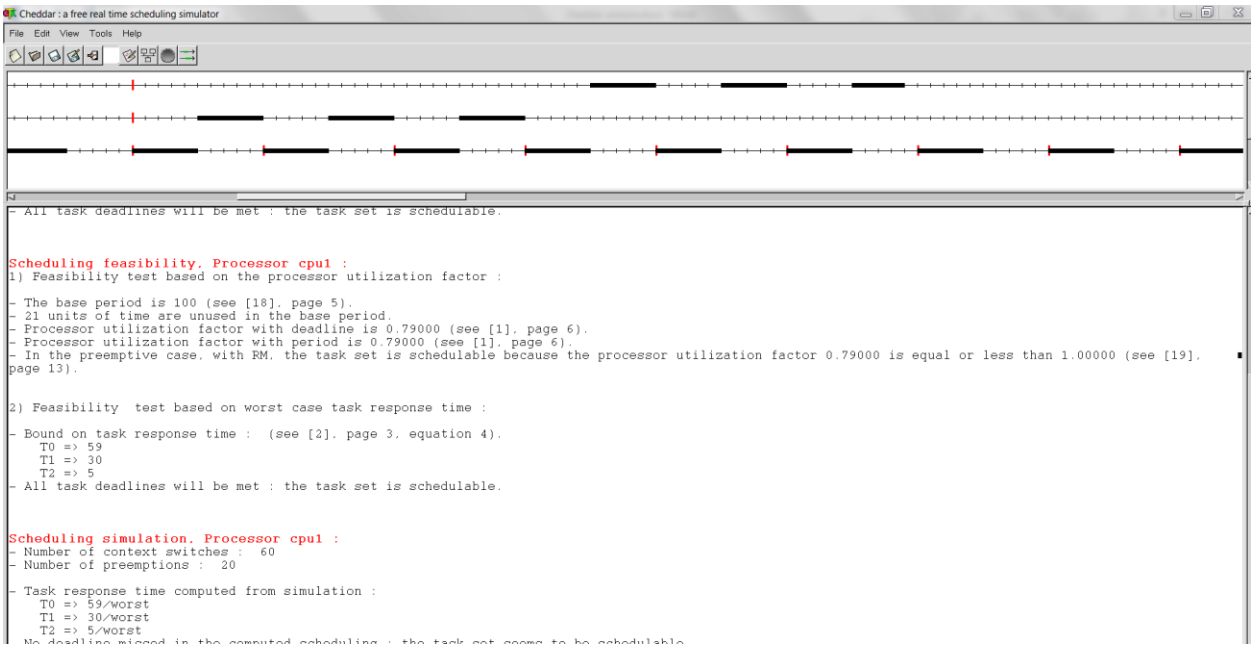
The best and closest approximation we can make was by using a single pthread and try to obtain the maximum time taken to complete through 100 of same repetitive tasks and also find the average time.

Using the maximum and average time we estimate the T,C values and we have repeated the process for all the three threads

Task1: T—1sec; C—140ms;

Task2: T—1sec; C—130ms;

Task3: T—100msec; C—60ms;



6 PROOF-OF-CONCEPT WITH EXAMPLE OUTPUT AND TEST COMPLETED

Test to show all cores are enable

```
Thread idx=1, sum[0...200]=19900
Thread idx=1 ran on core=0, affinity contained: CPU-0

Thread idx=1 ran 0 sec, 71 msec (71592 microsec)

Thread idx=2, sum[0...300]=44850
Thread idx=2 ran on core=0, affinity contained: CPU-0

Thread idx=2 ran 0 sec, 71 msec (71552 microsec)

Thread idx=3, sum[0...400]=79800
Thread idx=3 ran on core=0, affinity contained: CPU-0

Thread idx=3 ran 0 sec, 71 msec (71582 microsec)

TEST COMPLETE
root@tegra-ubuntu:/home/ubuntu/affinity_test# ./pthread
This system has 4 processors configured and 1 processors available.
number of CPU cores=4
Using sysconf number of CPUS=4, count in set=4
Pthread Policy is SCHED_OTHER
Pthread Policy is SCHED_FIFO
PTHREAD SCOPE SYSTEM
rt_max_prio=99
rt_min_prio=1
Setting thread 0 to core 0
 CPU-0
Launching thread 0
Setting thread 1 to core 1
 CPU-1
Launching thread 1
Setting thread 2 to core 2
 CPU-2
Launching thread 2
Setting thread 3 to core 3
 CPU-3
Launching thread 3

Thread idx=0, sum[0...100]=4950
Thread idx=0 ran on core=0, affinity contained: CPU-0

Thread idx=0 ran 0 sec, 71 msec (71692 microsec)

Thread idx=1, sum[0...200]=19900
Thread idx=1 ran on core=0, affinity contained: CPU-0

Thread idx=1 ran 0 sec, 71 msec (71628 microsec)
```

Calculation of Jitter and deadline.

```
**********************************Summary************************
Resolution 640*480
Average time  3.938070 ms
Frame Rate  253.931519 Hz
Total Frames : 162
Deadline Missed : 1
Deadline Not Missed: 161
```

d

Removing Jetson from IDLE mode

```
root@tegra-ubuntu:~# cat /sys/devices/system/cpu/cpuquiet/tegra_cpuquiet/enab
0
```

Using all the Jetson's four cores.

```
root@tegra-ubuntu:~# echo 1 > /sys/devices/system/cpu/cpu0/online
-bash: echo: write error: Invalid argument
root@tegra-ubuntu:~# echo 1 > /sys/devices/system/cpu/cpu1/online
-bash: echo: write error: Invalid argument
root@tegra-ubuntu:~# echo 1 > /sys/devices/system/cpu/cpu2/online
-bash: echo: write error: Invalid argument
root@tegra-ubuntu:~# echo 1 > /sys/devices/system/cpu/cpu3/online
-bash: echo: write error: Invalid argument
```

Capturing 5400 frames in exactly 9 minutes.

```
Frame 1470622126 sec, 544507723 nsec, dt= 0.00 msec, avedt= 0.00 msec, rate=  inf fps, frame count 12589
Frame 1470622126 sec, 584814015 nsec, dt= 0.00 msec, avedt= 0.00 msec, rate=  inf fps, frame count 12590
Frame 1470622126 sec, 623097491 nsec, dt= 0.00 msec, avedt= 0.00 msec, rate=  inf fps, frame count 12591
Frame 1470622126 sec, 661947101 nsec, dt= 0.00 msec, avedt= 0.00 msec, rate=  inf fps, frame count 12592
Frame 1470622126 sec, 701049606 nsec, dt= 0.00 msec, avedt= 0.00 msec, rate=  inf fps, frame count 12593
Frame 1470622126 sec, 739293901 nsec, dt= 0.00 msec, avedt= 0.00 msec, rate=  inf fps, frame count 12594
Frame 1470622126 sec, 777983015 nsec, dt= 0.00 msec, avedt= 0.00 msec, rate=  inf fps, frame count 12595
Frame 1470622126 sec, 816355958 nsec, dt= 0.00 msec, avedt= 0.00 msec, rate=  inf fps, frame count 12596
Frame 1470622126 sec, 855449384 nsec, dt= 0.00 msec, avedt= 0.00 msec, rate=  inf fps, frame count 12597
Frame 1470622126 sec, 894816658 nsec, dt= 0.00 msec, avedt= 0.00 msec, rate=  inf fps, frame count 12598
Frame 1470622126 sec, 934962664 nsec, dt= 0.00 msec, avedt= 0.00 msec, rate=  inf fps, frame count 12599
Frame 1470622126 sec, 975357025 nsec, dt= 0.00 msec, avedt= 0.00 msec, rate=  inf fps, frame count 12600
Frame 1470622127 sec, 17413052 nsec, dt= 0.00 msec, avedt= 0.00 msec, rate=  inf fps, frame count 12601
Taking snapshot number 5391
Frame 1470622127 sec, 59624057 nsec, dt= 0.00 msec, avedt= 0.00 msec, rate=  inf fps, frame count 12602
Taking snapshot number 5392
Frame 1470622127 sec, 102216186 nsec, dt= 0.00 msec, avedt= 0.00 msec, rate=  inf fps, frame count 12603
Taking snapshot number 5393
Frame 1470622127 sec, 145299301 nsec, dt= 0.00 msec, avedt= 0.00 msec, rate=  inf fps, frame count 12604
Taking snapshot number 5394
Frame 1470622127 sec, 190135573 nsec, dt= 0.00 msec, avedt= 0.00 msec, rate=  inf fps, frame count 12605
Taking snapshot number 5395
Frame 1470622127 sec, 232089632 nsec, dt= 0.00 msec, avedt= 0.00 msec, rate=  inf fps, frame count 12606
Taking snapshot number 5396
Frame 1470622127 sec, 273436978 nsec, dt= 0.00 msec, avedt= 0.00 msec, rate=  inf fps, frame count 12607
Taking snapshot number 5397
Frame 1470622127 sec, 318934509 nsec, dt= 0.00 msec, avedt= 0.00 msec, rate=  inf fps, frame count 12608
Taking snapshot number 5398
Frame 1470622127 sec, 367688672 nsec, dt= 0.00 msec, avedt= 0.00 msec, rate=  inf fps, frame count 12609
Taking snapshot number 5399
Frame 1470622127 sec, 411245730 nsec, dt= 0.00 msec, avedt= 0.00 msec, rate=  inf fps, frame count 12610
Taking snapshot number 5400
Frame 1470622127 sec, 456384206 nsec, dt= 0.00 msec, avedt= 0.00 msec, rate=  inf fps, frame count 12611
Frame 1470622127 sec, 495307359 nsec, dt= 0.00 msec, avedt= 0.00 msec, rate=  inf fps, frame count 12612
Frame 1470622127 sec, 538367984 nsec, dt= 0.00 msec, avedt= 0.00 msec, rate=  inf fps, frame count 12613
Frame 1470622127 sec, 579636480 nsec, dt= 0.00 msec, avedt= 0.00 msec, rate=  inf fps, frame count 12614
Frame 1470622127 sec, 619268453 nsec, dt= 0.00 msec, avedt= 0.00 msec, rate=  inf fps, frame count 12615
Exiting ...


Number of pictures taken 5400
root@vishvesh-Inspiron-1545:/home/vishvesh/Documents/RTES Final Project/backup stereo code/Good/GOOD_2_3_aug/Good_10MHz/simpler-capture# ffmpeg
```

7 CONCLUSION

In conclusion, the system accuracy and predictability were tested, and the results were compared. The results shown above sections verifies that we were able to complete the time lapse videos with a very a good precision and less jitter.

Better deterministic system was achieved due to algorithm which was used in the program. The algorithm made use of the time rather than other factor in order to capture the frame. That was probably one of the reason of getting high accuracy and low jitter during program execution

8 FORMAL REFERENCES

There were many references from different codes from the link given below

http://mercury.pr.erau.edu/~siewerts/cec450/code/

http://mercury.pr.erau.edu/~siewerts/cs415/code/computer-vision/

The OpenCV functions references

http://docs.opencv.org/2.4/modules/highgui/doc/reading_and_writing_images_and_video.html

http://docs.opencv.org/master/d8/dfe/classcv_1_1VideoCapture.html#gsc.tab=0

http://www.rubydoc.info/github/gonzedge/ruby-opencv/OpenCV/CvCapture

http://www.emgu.com/wiki/files/1.3.0.0/html/6c5248ae-fe37-c418-9613-72688f24ae70.htm

http://www.emgu.com/wiki/files/1.3.0.0/html/9e24d43f-647a-acd5-6be3-ccd02a6e0674.htm

http://docs.opencv.org/2.4/modules/core/doc/basic_structures.html#Mat::Mat%28%29

http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_circle/hough_circle.html

Image Capture references

https://codeplasma.com/2012/12/03/getting-webcam-images-with-python-and-opencv-2-for-real-this-time/

https://jayrambhia.wordpress.com/2012/05/10/capture-images-and-video-from-camera-in-opencv-2-3-1/

http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_gui/py_video_display/py_video_display.html

http://www.theimagingsourceforums.com/showthread.php?324116-How-to-get-accurate-time-stamps-for-each-frame

http://docs.opencv.org/2.4/doc/tutorials/introduction/display_image/display_image.html

http://docs.opencv.org/2.4/modules/highgui/doc/reading_and_writing_images_and_video.html

The RT Linux function references

http://linux.die.net/man/3/clock_gettime

http://pubs.opengroup.org/onlinepubs/007908775/xsh/pthread_mutex_lock.html

http://www.tutorialspoint.com/c_standard_library/c_function_sprintf.htm

The pthread references

https://computing.llnl.gov/tutorials/pthreads/

http://www.cs.fsu.edu/~baker/realtime/restricted/notes/pthreads.html

http://man7.org/linux/man-pages/man2/sched_setscheduler.2.html

The POSIX references

http://www2.fisica.unlp.edu.ar/electro/temas/p7/POSIX4.pdf

http://ecee.colorado.edu/~ecen5623/ecen/ex/Linux/POSIX/

9 APPENDICES WITH RESULTS, CODE AND SUPPORTING MATERIAL

As a part of the final submission, there were two videos which were submitted. First video is of sunset taken from the window, which has the real time motion of the trees and people moving. The second video is of ICE melting with respect to a Google Digital Clock. There is also a motion detection video, which demonstrate how the real time processing starts when a round object is detected.

There are also two folder containing 2000 frames in ppm format along with the header and other in jpg format. There are then three folders which contains in total 5400 frames, as the size of all the frames were very large.

The Final Extended Lab code is also submitted in the dropbox.