REAL TIME EMBEDDED SYSTEM
ECEN 5623
Summer 2017
TIME-LAPSE
Project Report

YASIR ASLAM SHAH
Yasir.shah@colorado.edu

INDEX

INTRODUCTION:

As per Wikipedia [1] *"Time-lapse photography is a technique whereby the frequency at which film frames are captured (the frame rate) is much lower than that used to view the sequence. When played at normal speed, time appears to be moving faster and thus lapsing."*

This project is based on all the concepts of real time scheduling learnt in this course, thus generating an accurate time lapsed video. Time-lapse is the extreme version of the cinematography technique of under cranking. It is also known as fast motion video.

Basically in a time lapsed video, the frames are captured at a very lesser frame rate and the frames are played at a faster rate thereby the time in the video appears to be moving faster. A recorded video of several minutes, or hours or even weeks can be time lapsed to few seconds or even few minutes. The processes that are not noticed by a human eye for example the conversion of day to night, or melting of ice as these processes are in its effect through a large time period hence such large changes can be noted quickly using time lapse. These days, such additional camera functionality is already present in smart devices including other effects like Slow motion, color conversion.

Time lapsing has found worldwide enthusiasts interested majorly in natural processes like formation of clouds, blossoming of flowers, sunrise and sunset.

In this project the frames are captured at a specific frequency of 1Hz and 10Hz.The captured frames are processed with time stamping and naming before saving the frames onto a video format file. Time stamping is also done to keep a track of frames and the time for each frame. This project has been developed on Raspberry Pi, with 1.2GHz processor, 1GB RAM and 18GB on board memory. The Raspberry Pi has USB connections to connect keyboard, mouse and camera. In this project Logitech C200 Camera is used to capture frames. The Raspberry Pi is powered using a 5V adapter with a mini USB connector, the display is connected using HDMi. Pthreads, Semaphore, Sequencer, and Mutex are used in this project for real time system. OpenCV API is used to capture, process and store the frames and generate the desired time lapsed video. Frames are captured at 1Hz and 10Hz for 2000 and 6000 frames respectively and the images are stored as PPM and JPG.

FUNCTIONAL REQUIREMENTS

Project has the following functional Requirements

- Frame Capture: the project focuses on capturing frames from camera using OpenCV libraries at a frequency of 1Hz.The captured frames have a resolution of 640x480 as mentioned in the project requirements. For capturing the frame, a POSIX thread is being implemented that captures frames and the POSIX thread is being controlled using a sequencer.
- Image format: The frames are saved simultaneously on a real time basis as a part of another real time service. The frames are initially saved as .PPM format in a system location defined in the code. The JPG formatted image files are later used to form a video as MPEG4 (avi) format.
- Time stamping frames: The PPM formatted frames are then embedded with frame info as Frame number, Author and Time stamp. The time stamp proves that the images are captured at one frame per second for 1Hz frequency. Uname system command was supposed to be implemented for time stamping however due to issues, I was not able to implement Uname in my approach and I have used cvputText() to provide the required details about each individual frames.
- Compression of size: The PPM formatted image files are then compressed simultaneously into JPG format which reduces the image size by 200%.For example, the size of Image0000.ppm is 900kB whereas the size of its JPG format image as Image000.JPG is only 50kB.Compression of frames is necessary for image storage as well as sending image files over Ethernet.
- Generate a video file from the sequenced .JPG files. For this I am saving all the frames in a location and then using command line with "*ffmpeg –f image2 -1 imagename.jpg video.avi*" to generate a video files of 2000 frames with an .avi format. The video is played back using OMXplayer "*omxplayer video.avi*".
- Storing images: As per the project requirements, I am saving all the frames in a defined location in my system. The images are stored in .JPG and .PPM format.Jpg being the compressed file where as the .ppm files are modified and embedded with timestamp and later formatted into a video format.2000 frames are captured at 1Hz in 33 minutes and 20

seconds however the time lapsed video is of duration of around 100 seconds if played at 20FPS.

- One of the extra requirements mentioned in the project rubric involved sending compressed frames over Ethernet. However due to limited time, I was successful in sending only one file or one compressed folder over Ethernet using client server and socket programming. However for the transfer to be real time service, I couldn't accomplish it in given time
- One of the major requirement of this project involved capturing at a higher rate. In this project, I have tested capturing 6000 frames at 10Hz frequency. The FPS captured at 10Hz is 10 frames per second and the frames are captured as well as saved in .PPM and .JPG format. A total of zero jitter is measured at the end of capturing and saving 6000 frames at 10Hz
- The jitter value and the time stamp as well as the frame number is stored in excel sheet and represented later in the report graphically.

REAL TIME REQUIREMENTS

The project is divided into three real time services as described below:

- Sequencer: This is the first real time service that enjoys the highest priority in my project. The basic reason to use sequencer is to provide the necessary delay so that my services are fetched within the deadline. The deadline of sequencer is set as per the deadline of constituent services plus a 20% margin for safe schedulibilty and feasibility. The sequencer provides semaphore to the service and the first service is executed once every second and nanosleep is used to provide the necessary delay so that the service is met only once a second for 1Hz case.
- Service 1: This service focuses on capturing frames. OpenCv Libraries and API are implemented in this service. In this project, two cases are implemented with first case the frames captured are at a frequency of 1Hz and in other case, the frames are captured are at a frequency of 10Hz. For capturing frames, Ipllmage is used and declared globally for other threads to access it. The frames are captured and this service further sends control to the other service.

- Service2 Is used to track time in seconds and milliseconds so that the difference between one frame and the next is measured. For debugging purposes, Syslog is used as syslog doesn't add to the delay in the system as compared to printf. This service is also responsible for real time saving and compressing if frames in .ppm and .jpg format. Timestamping as cvPutText is also done to embedded frame details on every individual frame so that the real time response of system can be proved and measured. Similarly for 10Hz frequency, gettimeofday is used to track the timings.

Other Services:

- The real time response is ensured in a way that all the services take 1sec to run. From capturing the frame, till saving the frame in .ppm as well as .jpg format along with time stamped details on .ppm frames, the entire project is real time with the entire process happening in 1 second.
- Generating Video
  Initially for converting all the .jpg Frames into a video formatted file, writer.write was used however due to jitter and delay in the system, I am processing .jpg frames separately into a video format using "*ffmpeg –f image2 -1 imagename.jpg video.avi*"
- The following software tools have made it possible to accurately finish this project with an efficient and deterministic real time system
- Semaphores: Using semaphore has made it possible to reach for the closest accuracy in implementing thread time system design. Sem_wait and Sem_post between all the real time services have made it possible for the system to meet its deadline.
- Sequencer: The use of sequencer has made it possible to develop and test this project for 1Hz and for10Hz with accurate required delays using nanosleep().
- Mutex () : Mutex is used to ensure that a certain segment of a code is never interrupted and the delays provided are accurate.

The final processed frames are written to a video formatted file and a smooth time lapse is available.

FUNCTIONAL DESIGN OVERVIEW AND DIAGRAMS

HARDWARE DESIGN:

The following hardware modules are a part of the project

- Raspberry pi
- USB VGA Logitech C200 Camera
- Linux OS
- Open CV
- Keyboard
- Mouse
- Display HDMi

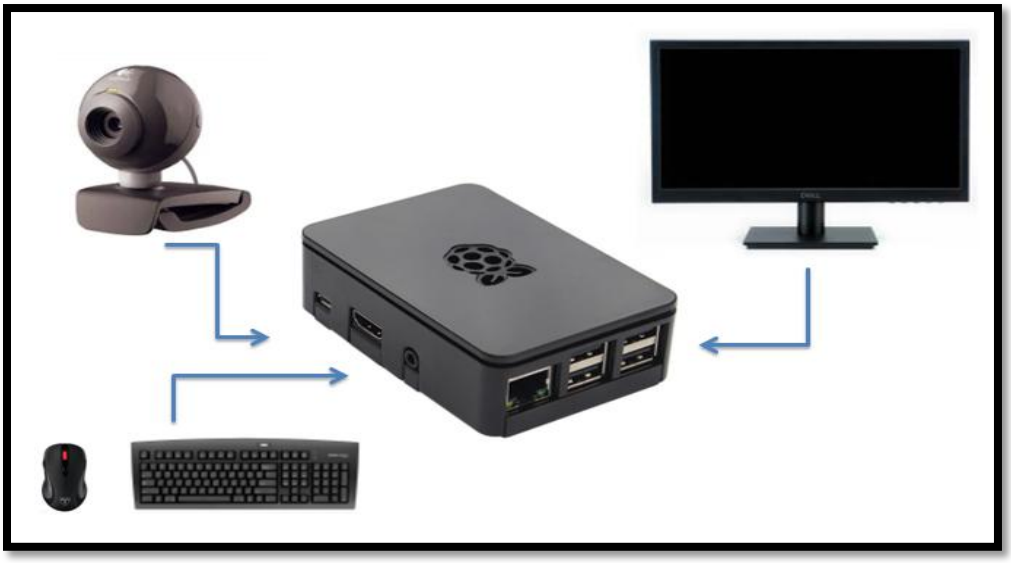
In this project, Embedded Linux OS is used along with OPENCV libraries to complete this project. I have implemented POSIX thread API, Semaphores, Mutex to achieve deterministic flow for an efficient real time system.

NOTE:

However, with long hours of working on Raspberry Pi, I personally found the hardware heating ups and hanging up numerous times thereby increasing inconvenience to the user.

Hardware Flow Daigram

The hardware flow diagram for this project follows as below:

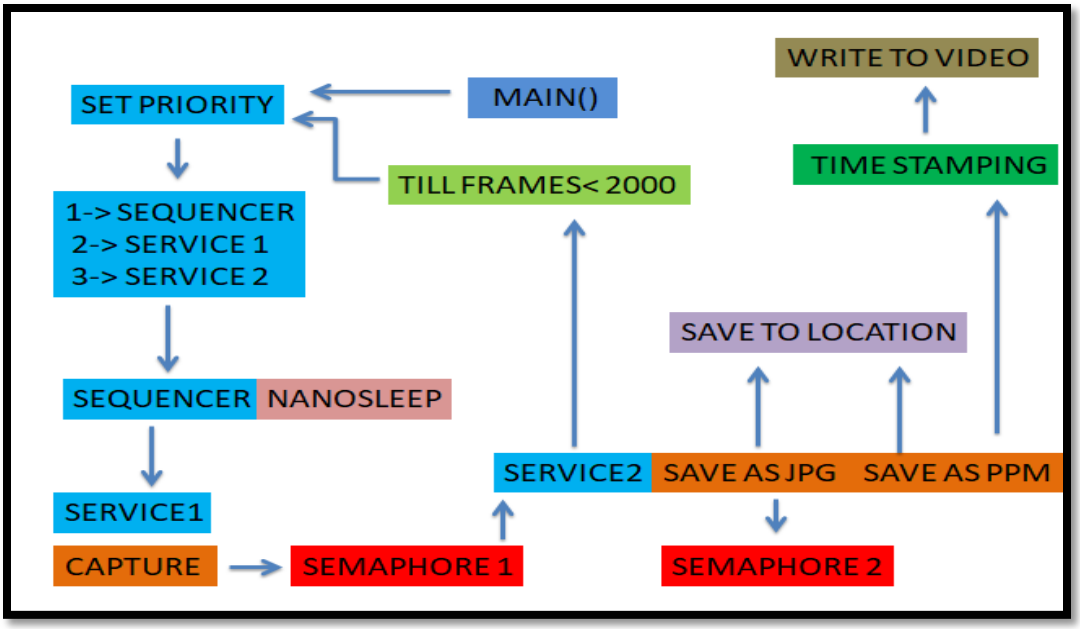


<- Figure 1

- **Hardware Design Block containing Raspberry Pi as the brains for the system along with USB connected Logitech C200 Camera, Mouse and Keyboard.**
- **The Monitor is**

SOFTWARE ARCHITECTURE DIAGRAM

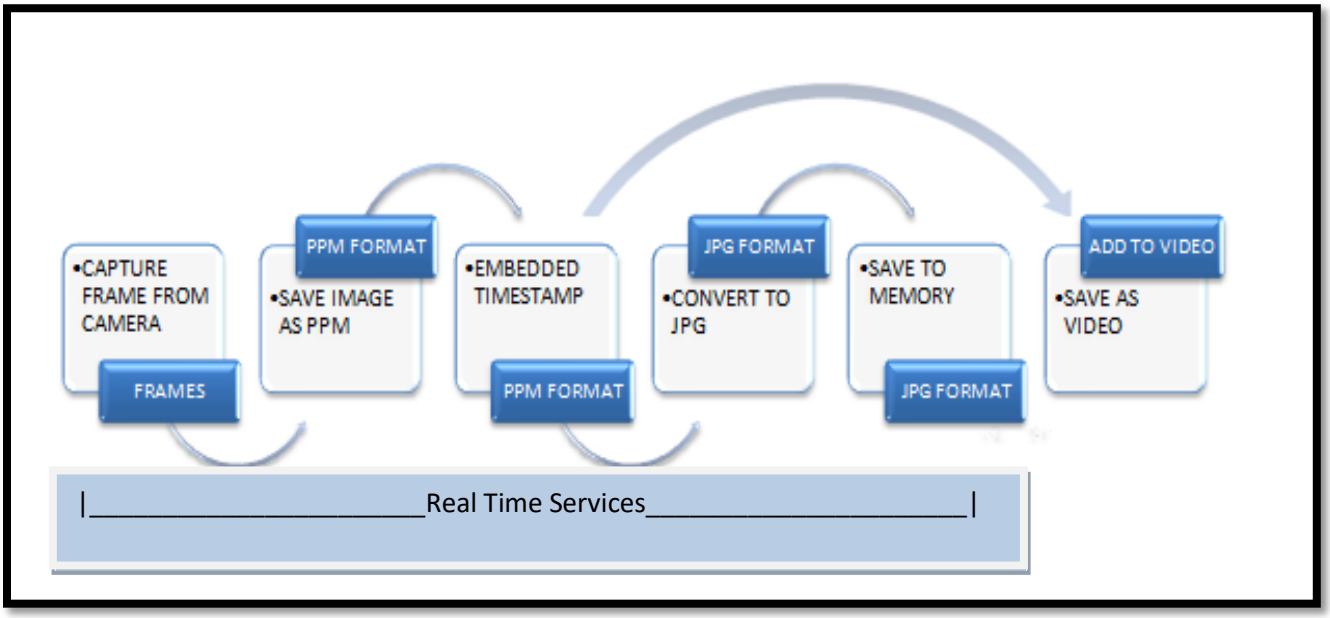
The Software flow diagram is expressed as below:



<- Figure

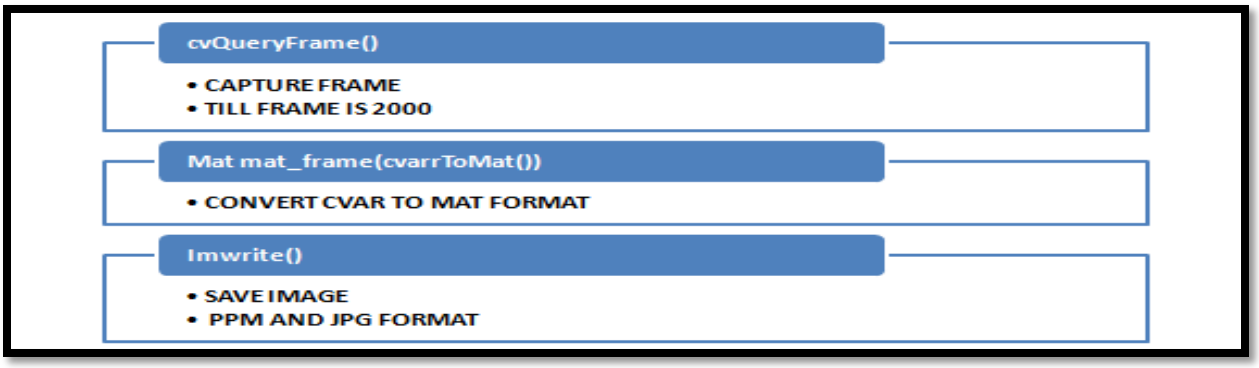
- **Software flow diagram**
- **All the blocks except for WRITE TO VIDEO is non Real, whereas all the blocks are real time based on a hard deadline**

SOFTWARE FLOW DAIGRAM



- **Software Block diagram with various functional modules**
- **The frames are captured, and processed with different file formatting and frames are added to a final video file**
- **The capturing, saving as well as time stamping frames basically include the real time system**

FUNCTIONAL BLOCK DIAGRAM:



- **Functional Blocks using OPENCV**
- **From capturing frames till saving the frames as .ppm and .jpg**

TIME STAMP:

Getctimeofday is used to get timings of the execution and various other processes like the duration between one image capture to another or the time duration between one image capture till it is completely processed and saved. Syslog has been implemented to display the time stamp in the entire function through system log messages as syslog takes less space than printf and doesn't harms the program with delay and jitter as compared to printf. Timing values in seconds and milliseconds are implemented to ensure the accuracy of system

JITTER:

Jitter is mainly calculated as a service misses its expected deadline. Each complete frame is supposed to get complete in 1second or less than that, however several times the service missed its deadline and is delayed by few milliseconds. A plot of frame number vs jitter at that particular frame is plotted in excel sheet and is graphically represented in this report.

IMAGE AND VIDEO CONFIGURATION:

1. Images are captured as
    - 640x480 Resolutions
    - 1 FPS for 1Hz
    - 10 FPZ for 10Hz
    - JPG and PPM format

2. Time lapsed video

    - 640x480 resolution
    - 20FPS
    - 100 seconds time lapsed video for 33min and 20 seconds long video (2000 Frames)
    - 300 seconds time lapsed video for 3 min and 2o seconds one video (6000 Frames)

REAL-TIME ANALYSIS AND DESIGN WITH TIMING DIAGRAMS

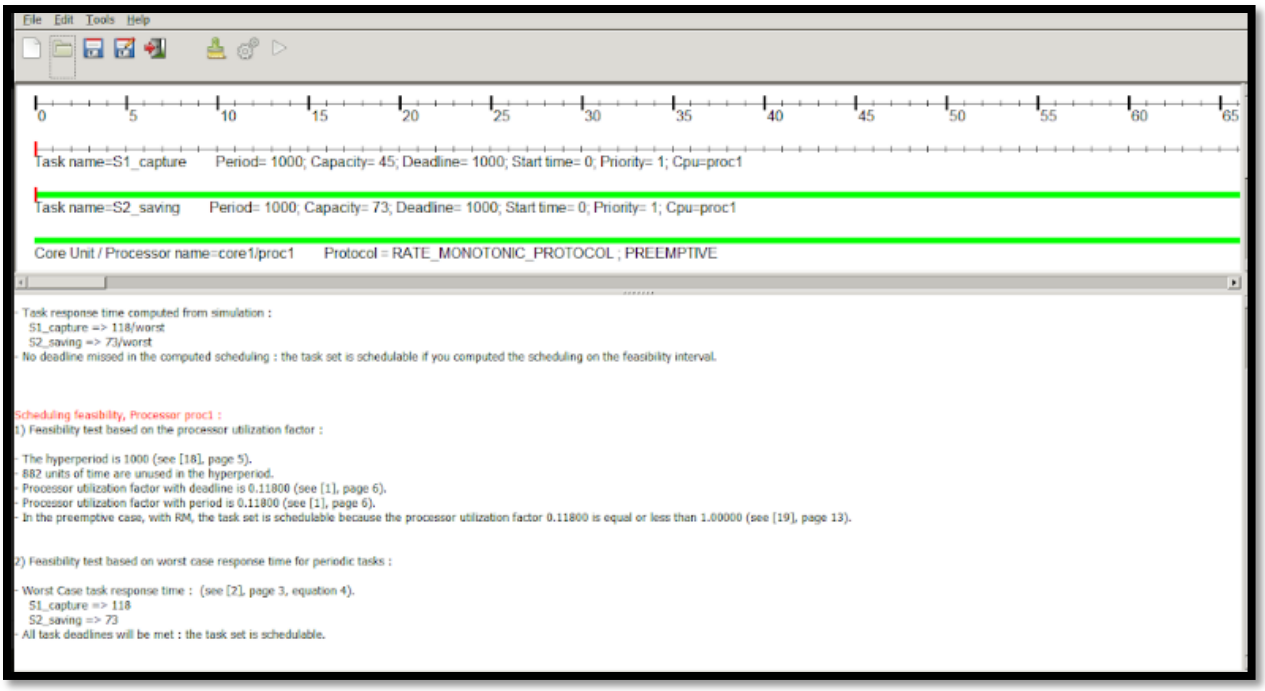
FIFO is used as Scheduling policy for this project as thread priority can be set as per the incoming threads. Rate Monotonic Policy is used as least execution time task is given highest priority.

Cheddar Analysis is done for both cases and are presented in this report

Cheddar Analysis:

Theoretical Analysis for 1Hz -Cheddar Analysis:

Theoretical Analysis for 1Hz -Cheddar Analysis:

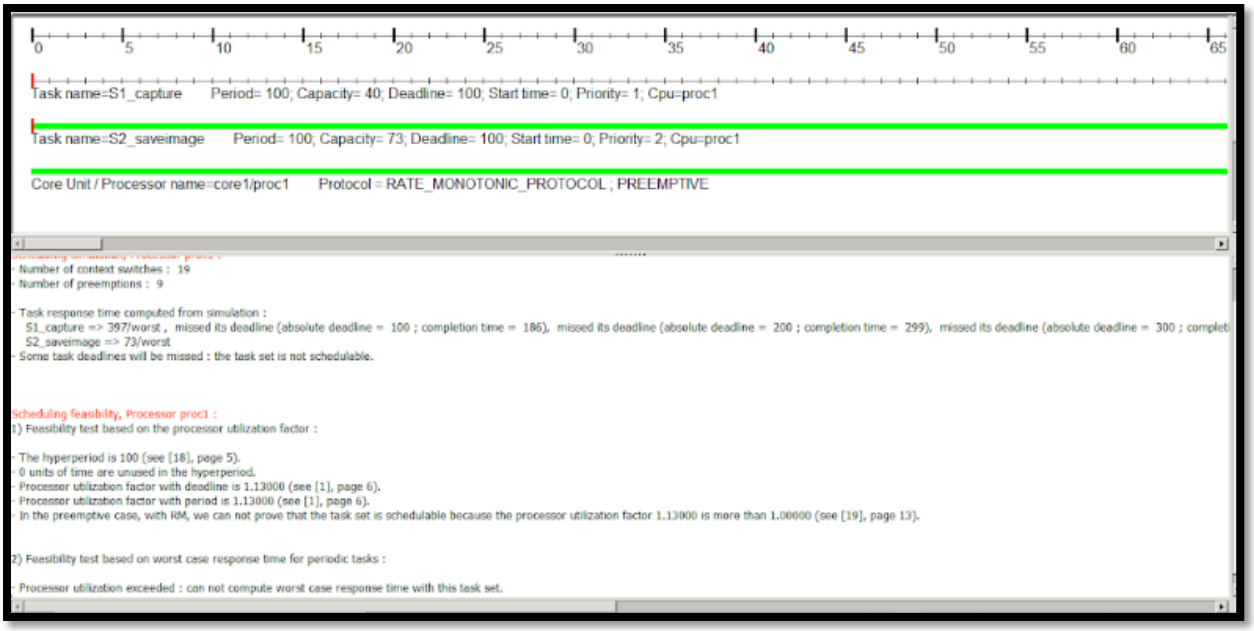


In this Cheddar analysis, the two worst case considered gives us a Capacity of 45 for Task1 (Capturing) and 73 for Task2 (Saving).The deadline considered for both the cases is hard deadline as 1second or 1000milliseconds.

- The Utility factor for Task 1 with C1=45 and T1 as 1000ms is U1=0.045
- The Utility factor for Task 2 with C2=73 and T2 as 1000ms is U2=0.073
- Total Utilization factor is 0.045 + 0.073= 0.118
- Utilization is 11.8%

Considering the worst case analysis for the case operating at 1Hz, the cheddar simulations point that the task is both feasible and schedulable. No deadlines are missed in this case

Theoretical Analysis for 10Hz -Cheddar Analysis:



In this Cheddar analysis, the two worst case considered gives us a Capacity of 40 for Task1 (Capturing) and 73 for Task2 (Saving).The deadline considered for both the cases is hard deadline as 0.1second or 100milliseconds.

- The Utility factor for Task 1 with C1=40 and T1 as 1000ms is U1=0.40
- The Utility factor for Task 2 with C2=73 and T2 as 1000ms is U2=0. 73
- Total Utilization factor is 0. 40 + 0. 73= 1.13
- Utilization is 113%
- With RM, the task predictability is unknown as the Utilization factor is greater than 1.00 or 100%
- Deadlines are missing and the task is not schedulable and not feasible

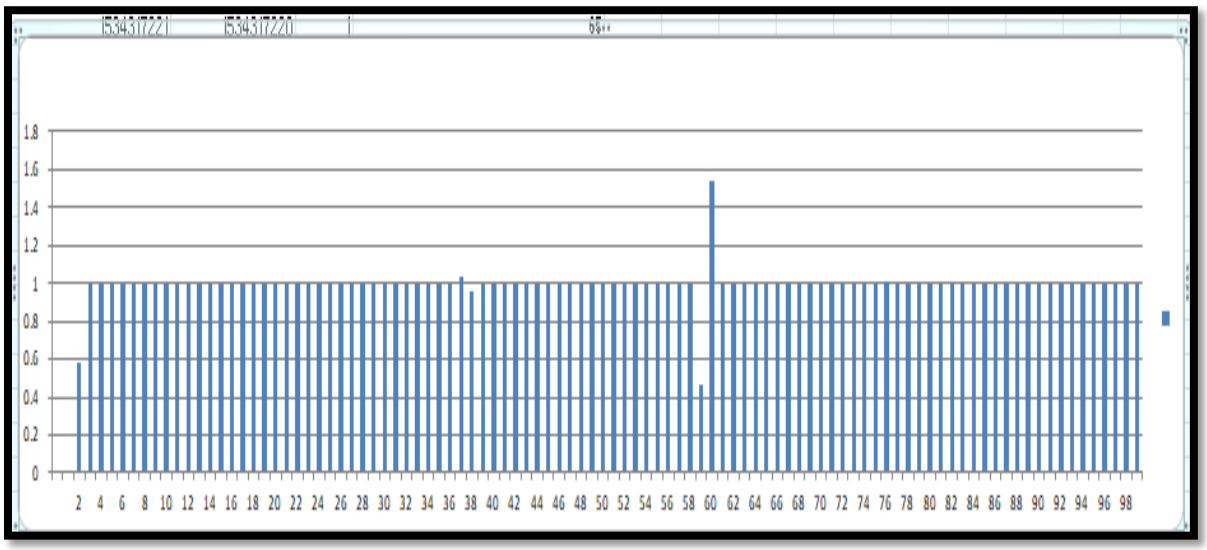
JITTER ANALYSIS

Jitter Analysis - For 1Hz

The jitter calculated is plotted against number of frames and the observations are such that the negative and positive jitter keeps the frame capture frequency at 1Hz only

- Total Time :1800s
- Calculated Time :1999.688s
- Jitter :312ms
- The Average jitter per frame is :156uS

JITTER PLOT VS FRAME



The plot shows the duration between two frames.Two frames have a difference of a 1second for 1Hz frequeny with some positive and negative jitter in microseconds per frame.

Jitter Analysis - For 10Hz

The jitter calculated is plotted against number of frames and the observations are such that the negative and positive jitter keeps the frame capture frequency at 10Hz only. There are timing fluctuations for each frame but the overall jitter is compensated for an average jitter per frame as

- Total time for last 1000 frames :100s
- Calculated Time for last 1000 frames :100.255s
- The total jitter for 6000 frames is :255ms
- The Average jitter per frame is :255us

ANALYSIS OF JITTER PLOT

The jitter plot gives an insight on how negative and positive jitter compensates task to meet deadline of 1Hz and 10Hz.
The jitter values for 10Hz were calculated based on the syslog for the last 1000 frames whereas the timestamp printed using syslog was used to generate data for jitter calculations for 1Hz

OVERALL ANALYSIS:

In the overall analysis, this project is an application of all the real time concepts taught in the course. This project taught me the how a deadline missing can affect the performance of any system since when testing 1Hz and 10Hz throughout the project build up, the missing deadlines are vivid with missing frames and rough time lapse.

With very less jitter, the time lapse video is very smooth while it is being run with proper timestamp changing each second for every frame.

PROOF OF CONCEPT:

1Hz:

- Proof of concept is provided as Images are saved as PPM and JPG with series from Image0000 till Image1999 for 1Hz
- Images are saved as IMAGE000.JPG and IMAGE000.PPM with size as 40kB and 1000kB respectively, Image compression is done by saving the frames as .JPG format
- All the four thousand files are saved and a video is generated as .MP4

Figure showing proof of images saved as ppm and jpg

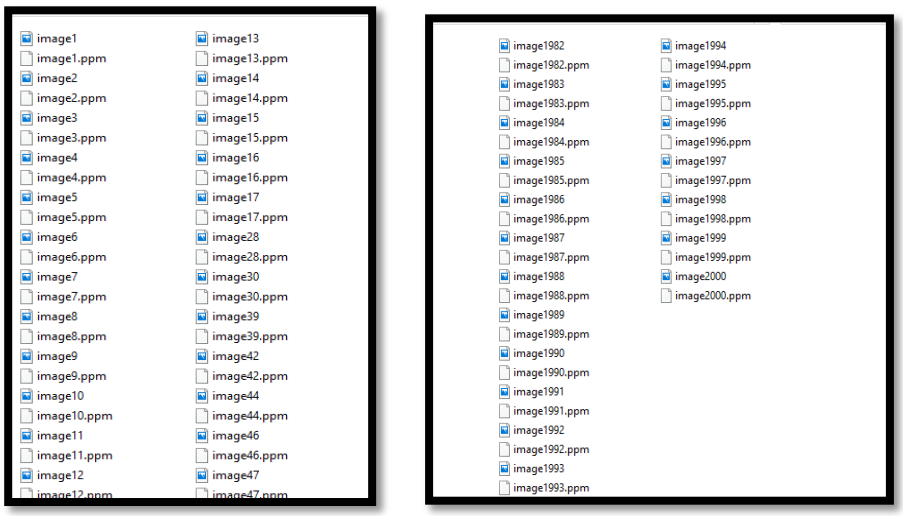
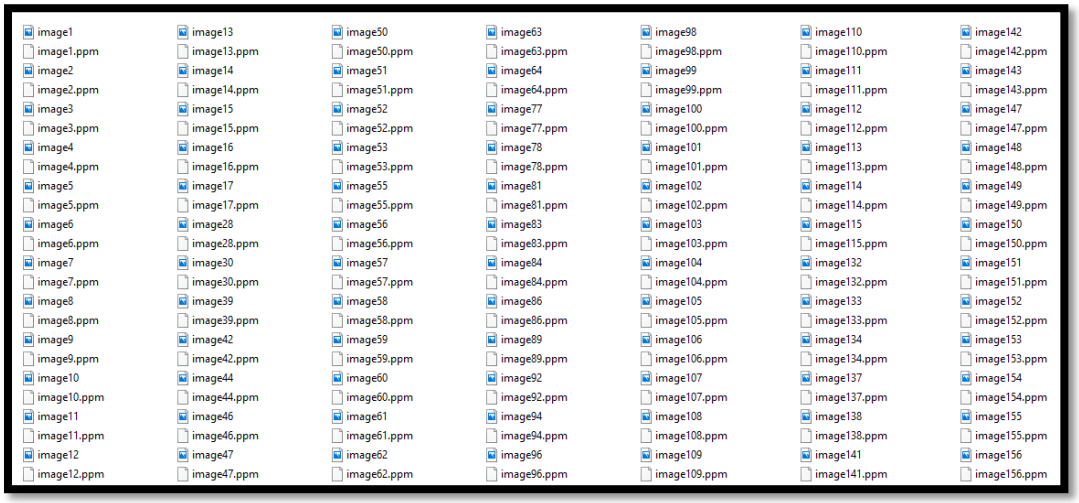
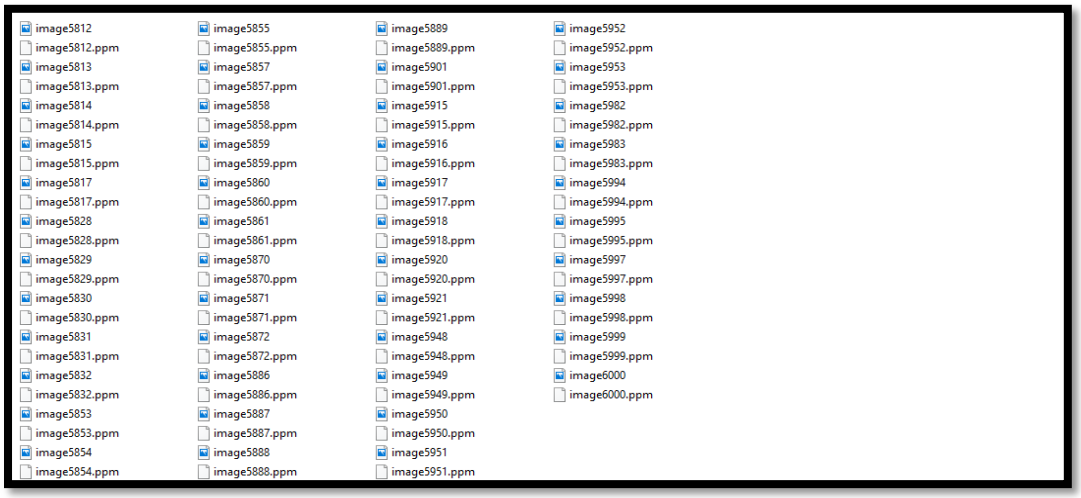


Figure showing proof of images with a timestamp of difference as 2000s

10Hz

- Proof of concept is provided as Images are saved as PPM and JPG with series from Image0000 till Image5999 for 10Hz
- Images are saved as IMAGE000.JPG and IMAGE000.PPM with size as 40kB and 1000kB respectively, Image compression is done by saving the frames as .JPG format
- All the twelve thousand files are saved only and no video is generated for this case.
- Figure showing proof of images saved as ppm and jpg





- Figure showing difference between first and last image is 600 seconds

VIDEO

Video with Analogue Clock:

A .avi file is generated from all the files for 1Hz with a timestamp. The frames were captured for 33 minutes and 20 seconds
Below are the screenshots from the frames

Difference in time stamp = 2000 seconds for 2000 frames

Video with Ice Melting:

A similar .avi file is generated from all the files for 1Hz with ice melting with timestamp for each frame.The frames were captured for 33 minutes and 20 seconds

The video generated with .avi format are converted into .MP4 format as per requisites for the project

ZIP FILE CONTENTS:

Zip file uploaded over D2L contains the following:

- Code and Makefile for 1Hz
- Code and Makefile for 10Hz
- 1Hz Video of analog watch
- 1Hz Video of Melting ice
- Report

Note: Images as ppm and jpg for 1Hz and 10 Hz are stored but couldn't upload because of the limitation of D2L for files to be less than 1GB

FUTURE WORK

In future, adding to this project, I would like to implement a real time service that provides image transformation to my existing project. Reversing the video and implementing a slow motion video can also be implemented using the knowledge learnt from this project.

Problems faced:

- Due to insufficient number of VGA –HDMi converter, I faced a lot of issues throughout y entire semester
- Overheating issues of RPi and RPi getting stuck numerous times
- Light issues affecting the jitter

CONCLUSION

This project beautifully strengthened all my real time concepts and enhanced my knowledge with Opencv libraries and function. In the end I feel complacent with the course and its teaching and my performance so far. The learning curve will go a long way in terms of the scope of opencv and real time systems.

ACKNOWLEDGMENT:

I would like to thanks Professor Sam Siewart and my TA for all their help and guidance in this course.

REFERENCE:

1. https://en.wikipedia.org/wiki/Time-lapse_photography
2. Professor Sam Siewart (RTES 5263)Website http://ecee.colorado.edu/~ecen5623/ecen/
3. Real Time Embedded System, Author Sam Siewart and John Pratt