UNIVERSITY OF COLORADO BOULDER

ECEN-5623

EXTENDED LAB

TIME-LAPSE PROJECT

BY:

TEJAS SHANBHAG

INTRODUCTION:

Time-lapse is a phenomenon where we capture many frames and go through them as quickly as we can. In this project, we capture frames using a USB camera at different rates and viewing it as a video to form a time-lapse.
The project further deals with using different real-time concepts like threading, semaphores, multi-processing, etc to give a deterministic response. Different libraries for these are being used on a LINUX processor.

Processes like frame capture, frame processing, compression, ethernet sharing, jitter analysis is being done in the design. Also, calculation of average execution times, average jitter, total latency, deadlines and missed deadlines are calculated. We are hence able to do all the processing in the required time frames with rates as good as 1 hz to 10 Hz.

FUNCTIONAL (CAPABILITY) REQUIREMENTS

Major functional capability requirements of the real-time software system:

a) Acquiring frames every second using the Logitech C200 camera using the Linux / NVIDIA JETSON TK1 system using the V4L2 library with minimum of 640*480 resolution.
b) Testing the accuracy of the frame capture for 2000 frames (1800) minimum. We will be doing this by capturing the frames of a digital clock where if the first frame has a set time, the 1800th frame must have 30min on the clock with a jitter of +-1 frame over 30 min period. This is measured with the melting of ice or some other phenomenon in the background.
c) Save the images in ppm P6 format while converting the frames and providing color. The ppm headers must include the timestamp, the resolutions and the host name i.e uname -a.
d) Calculate the jitter for each frame and the average jitter with the accumulated latency that builds up.
e) To observe the time-lapse, convert the frames into mpeg and view them as a video using ffmpeg. Observe some outdoor event along with the digital clock event.
f) Compress the ppm frames into jpeg so that less storage is taken up and are easier to send via ethernet. Use OpenCV libraries and functions to achieve this.
g) Run the code at a higher frame rate (preferably 10 Hz) for 9 mins, producing up to 6000 frames and have the jitter and accumulated latency verification for this part.

We use the videoforliunux and opencv libraries for the capture and change. The camera required drivers such as the UVC to be set up to capture frames.

REAL-TIME REQUIREMENTS

The real-time services are listed as follows:

a) Frame capture: The framecapture thread is used to capture the frames of the thread which includes capturing the frame in a particular time in YUV format, converting into ppm and saving it with alterations in the ppm header. The rate of capture can be 1 hz or 10hz.

b) Compression: The second thread is spawned after ppm is achieved. Hence, to save space while transferring, we convert the files into jpeg formats.

c) Jitter analysis: This is a best effort service since we do not need the real-time analysis for it. We store all the timestamps at start and end of both threads in a buffer and calculate the average execution time and jitter for each frame with total signed latency. As a result, this thread is executed at the end of the analysis

For each of the frames I saw the timestamps for each service and the total time it takes for each service to execute for each frame by which we get the average execution time for each service. This gives us the Ci which we take as slightly higher than the average time to get the worst-case execution time.

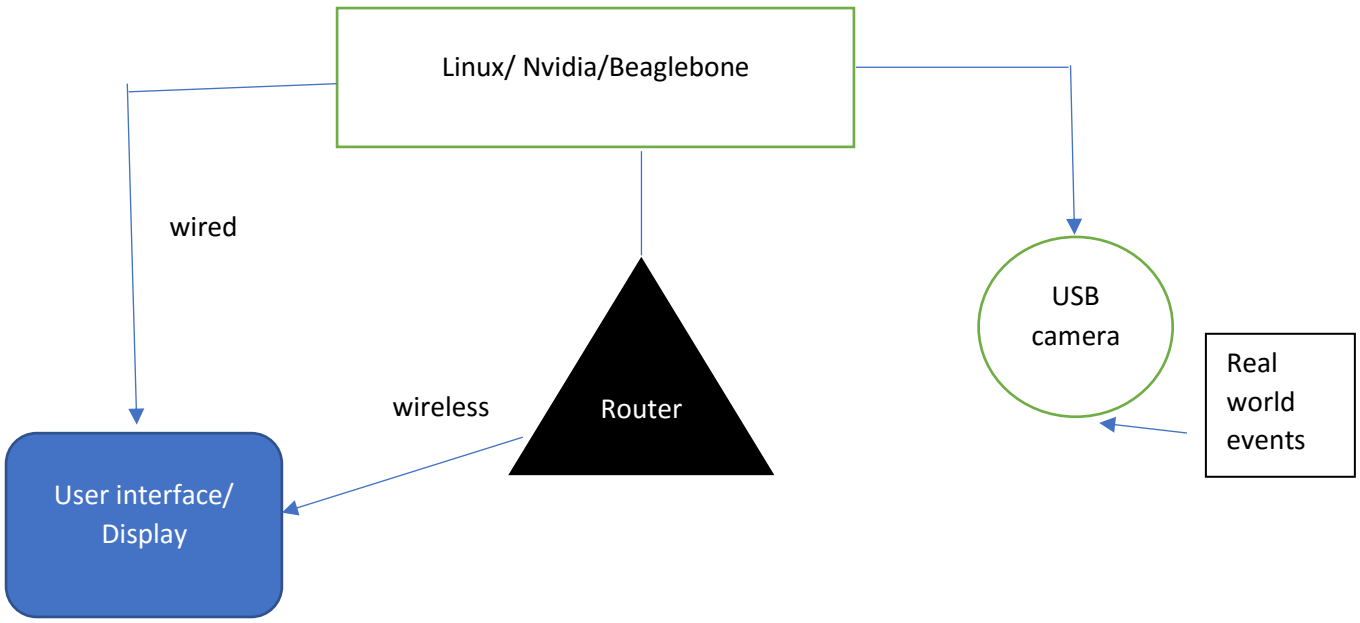
Time period for each thread must be ideally 1s. This makes sense which says that each frame should be captured after a sec. Hence, giving us a confirmation upon the accuracy we wish to achieve.

For the deadline, we already have the execution times and add them up. If they are well below 1s, we add an extra buffer depending on the jitter we get. On a safe scenario, we add up to 10% of execution time for each thread.

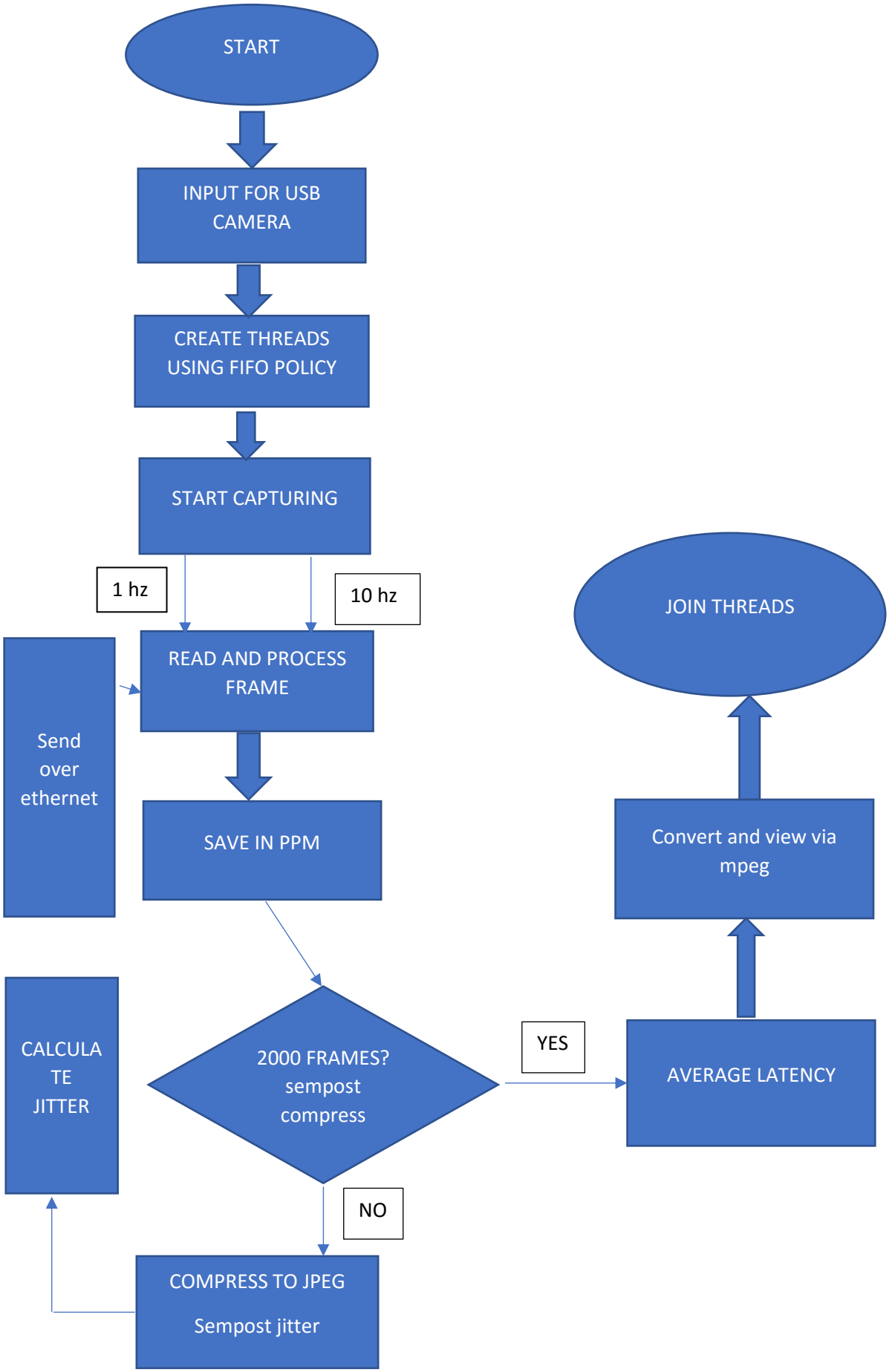
FUNCTIONAL DESIGN OVERVIEW AND DIAGRAMS

For the hardware part, we just have a linux system, Jetson or a Beagle Xm system. We have an USB camera for frame capture. The linux system is connected to a router or a modem which gives it an interface for the user and a medium to send frames over the ethernet to any other system. Further, the camera is used for capturing real world events.

Refer the hardware block diagram below

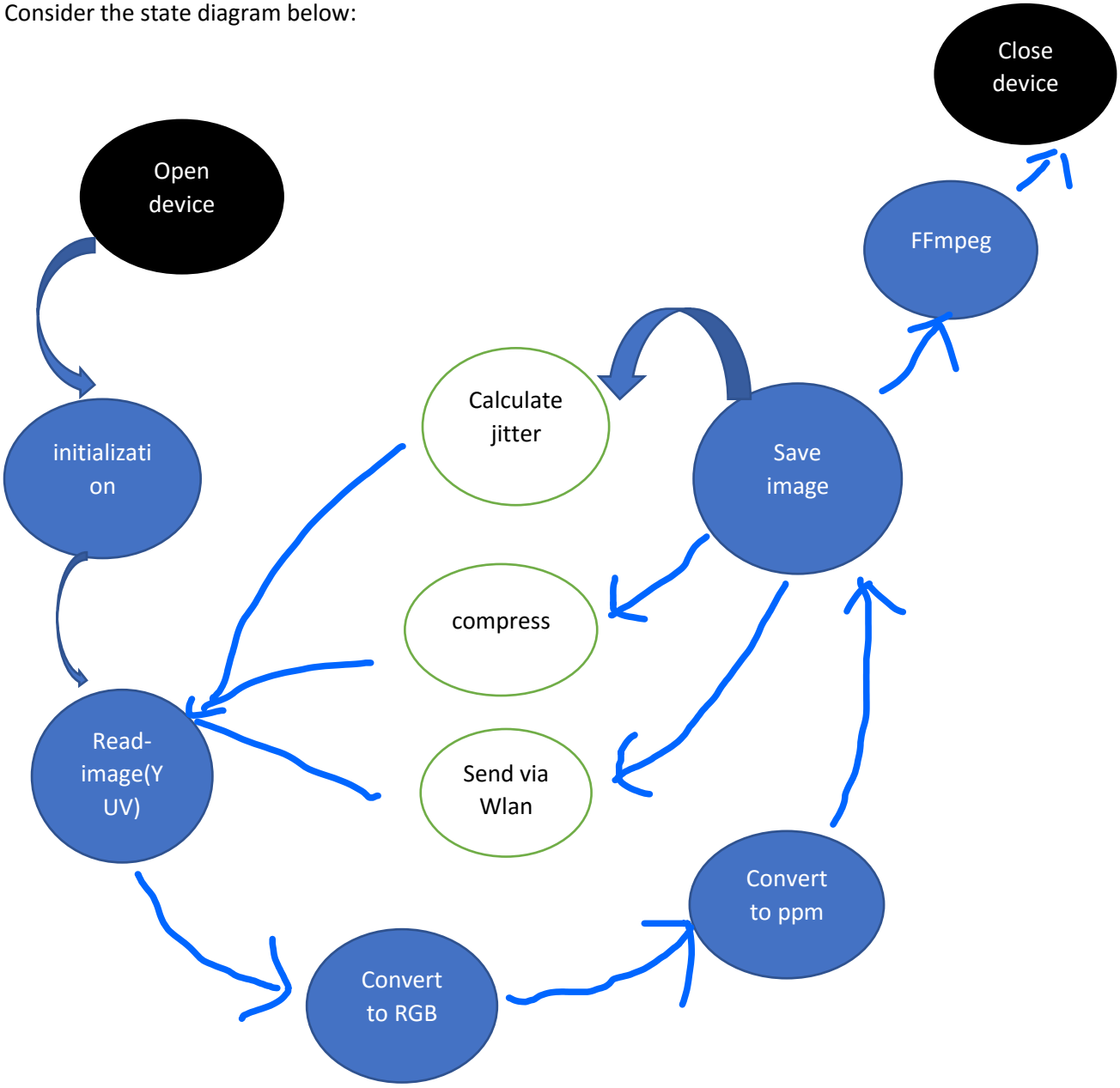


Consider the software organizational flow chart as shown below



- To describe the software organizational chart described above, we start with the main thread where we decide which camera device to use based on the argument given.
- Then we create threads and give them attributes to change to scheduling policy to First -in First- Out.
- After which, we have semaphores defined for the compression thread, jitter analysis thread and ethernet transfer.
- As a result, we then go into the highest priority thread which is for frame capture.
- At the start of the thread, we open the device for capture, do all the initializations for read, mmap and user pointer after which we have a function for starting capture.
- As a result, then we go into the mainloop which has a for loop for the number of iterations as the number of frames to be taken.
- In this loop, we first capture and read the image this then goes into processing. The camera captures the images in YUV format which is then successfully converted to RGB format since ppm takes input in the RGB format.
- As a result, we store the images in ppm format and modify the header to display the timestamp, the host processor “uname -a” and the resolution of the frame.
- Once the ppm frame is saved, we need to convert it to jpeg and to transfer it via the ethernet and store flash space.
- The semaphore for compression is posted. As a result, we use opencv libraries and load the ppm image and save the jpeg form using imread and imwrite respectively.
- Now the semaphore for jitter is posted and jitter calculates the time it took for each frame capture with the delay and compare it to 1s or with the average frame rate.
- Hence, the jitter is calculated and the semaphore for the ethernet transfer is posted after which the frames are transferred and downloaded continuously over the ethernet.
- This goes back for another frame capture in the capture thread and executes the same sequence till 2000 frames are received after which we calculate the total time and total latency.
- Then, the frames are converted and viewed via ffmpeg to see a timelapse video.
  In the end, the device is closed and threads are destroyed.

Consider the state diagram below:

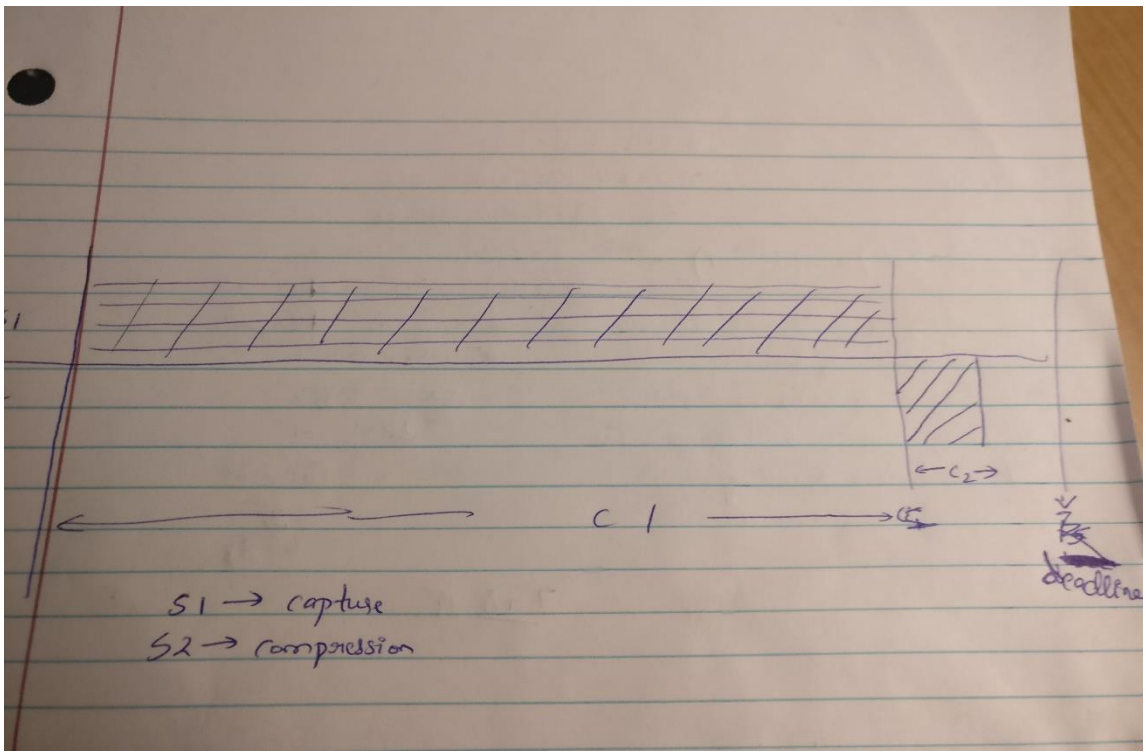


- Now to explain, the state diagram, it's similar to the software flow diagram wherein we show a transition of states of the processor in this diagram.
- Firstly, we open the device and start capturing.
- As explained above, we can read the image in YUV format which we need to convert into PPM P6 format.
- So, we have a conversion function to convert YUV to RGB.
- As a result, the frames are saved in ppm and then we go to the compression, jitter and ethernet download thread in succession for each frame until all the number of frames are captured.

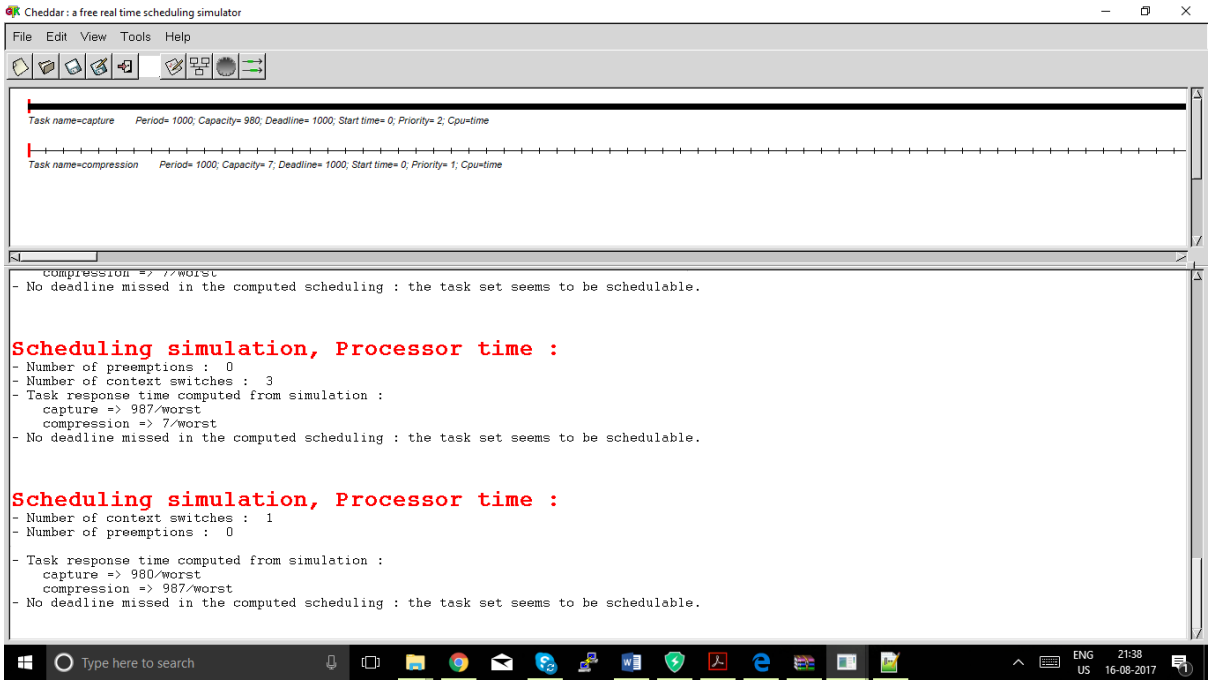
- After which we do the analysis with the best effort threads to convert into mpeg and to calculate the total latency.

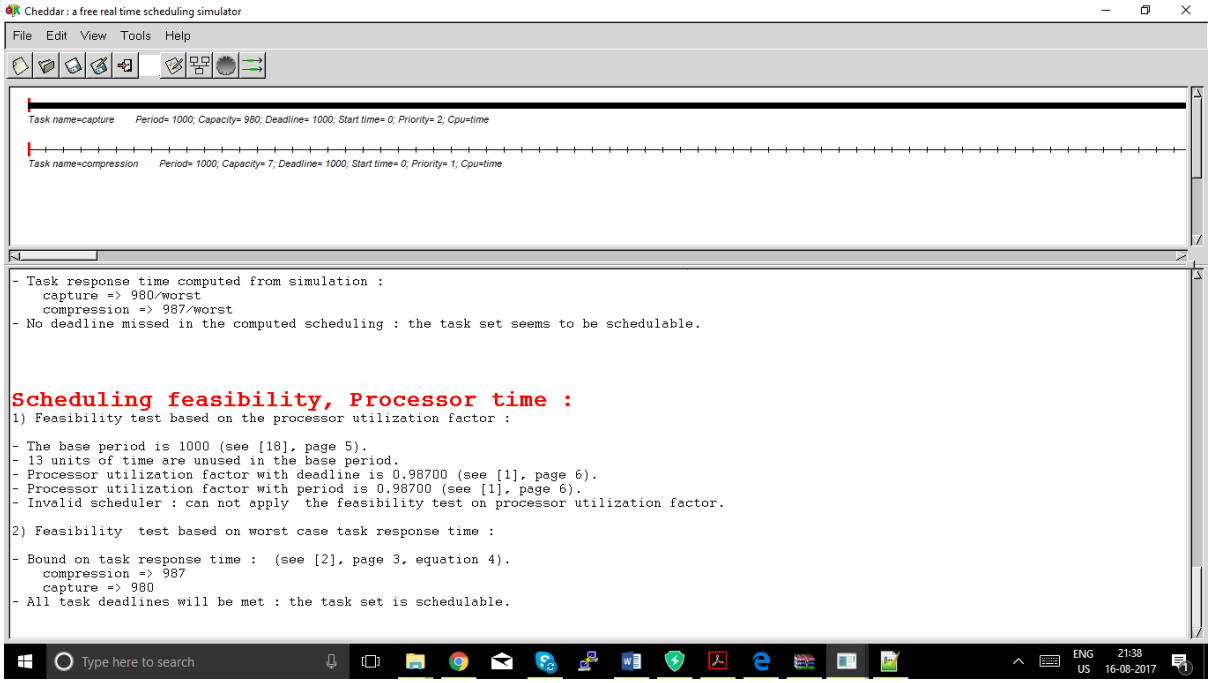
REAL-TIME ANALYSIS AND DESIGN WITH TIMING DIAGRAMS, BOTH MEASURED AND EXPECTED BASED UPON THEORY

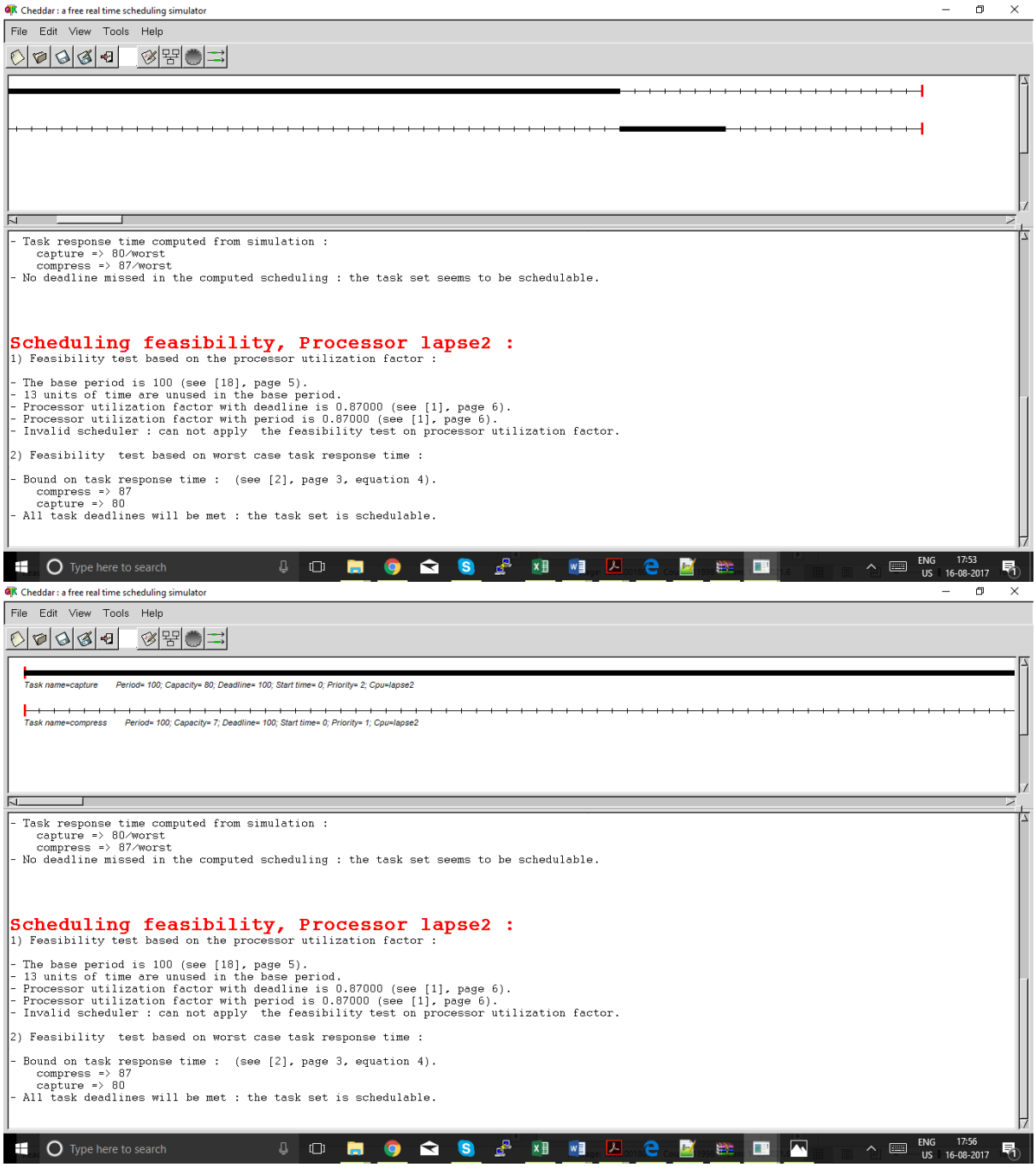
For this question, the real-time analysis using cheddar is implemented which is what we must be getting and hand-drawn which is expected. The hand-drawn diagram is shown below.



Now, we look at the cheddar analysis using rate-monotonic scheduler and FIFO to see the scheduling of the services.
As a result, we see that the first service is executed first and then the second and no deadline is missed from the simulations shown below and the real-time services are feasible.


Cheddar : a free real time scheduling simulator
File Edit View Tools Help
Task name=capture Period= 1000; Capacity= 980; Deadline= 1000; Start time= 0; Priority= 2; Cpu=time
Task name=compression Period= 1000; Capacity= 7; Deadline= 1000; Start time= 0; Priority= 1; Cpu=time
compression => 7/worst
- No deadline missed in the computed scheduling : the task set seems to be schedulable.
Scheduling simulation, Processor time :
- Number of preemptions : 0
- Number of context switches : 3
- Task response time computed from simulation :
capture => 987/worst
compression => 7/worst
- No deadline missed in the computed scheduling : the task set seems to be schedulable.
Scheduling simulation, Processor time :
- Number of context switches : 1
- Number of preemptions : 0
- Task response time computed from simulation :
capture => 980/worst
compression => 987/worst
- No deadline missed in the computed scheduling : the task set seems to be schedulable.
Type here to search
ENG US 21:38 16-08-2017


Cheddar : a free real time scheduling simulator
File Edit View Tools Help
Task name=capture Period= 1000; Capacity= 980; Deadline= 1000; Start time= 0; Priority= 2; Cpu=time
Task name=compression Period= 1000; Capacity= 7; Deadline= 1000; Start time= 0; Priority= 1; Cpu=time
- Task response time computed from simulation :
capture => 980/worst
compression => 987/worst
- No deadline missed in the computed scheduling : the task set seems to be schedulable.
Scheduling feasibility, Processor time :
1) Feasibility test based on the processor utilization factor :
- The base period is 1000 (see [18], page 5).
- 13 units of time are unused in the base period.
- Processor utilization factor with deadline is 0.98700 (see [1], page 6).
- Processor utilization factor with period is 0.98700 (see [1], page 6).
- Invalid scheduler : can not apply the feasibility test on processor utilization factor.
2) Feasibility test based on worst case task response time :
- Bound on task response time : (see [2], page 3, equation 4).
compression => 987
capture => 980
- All task deadlines will be met : the task set is schedulable.
Type here to search
ENG US 21:38 16-08-2017



Lapse 2 runs at 10hz

The computed time for a single frame are

Services	Time to execute(ms) (Ci)	Deadline(ms) (Di)
Frame capture with conversion 1hz	980.1	1000
10hz	80	100
Compression 1hz	6	8
Total 10hz	7	8

The table above shows the calculated execution times and the periods and deadlines assumed. The execution times are added with the margin of error to get the deadline.

PROOF-OF-CONCEPT WITH EXAMPLE OUTPUT AND TESTS COMPLETED

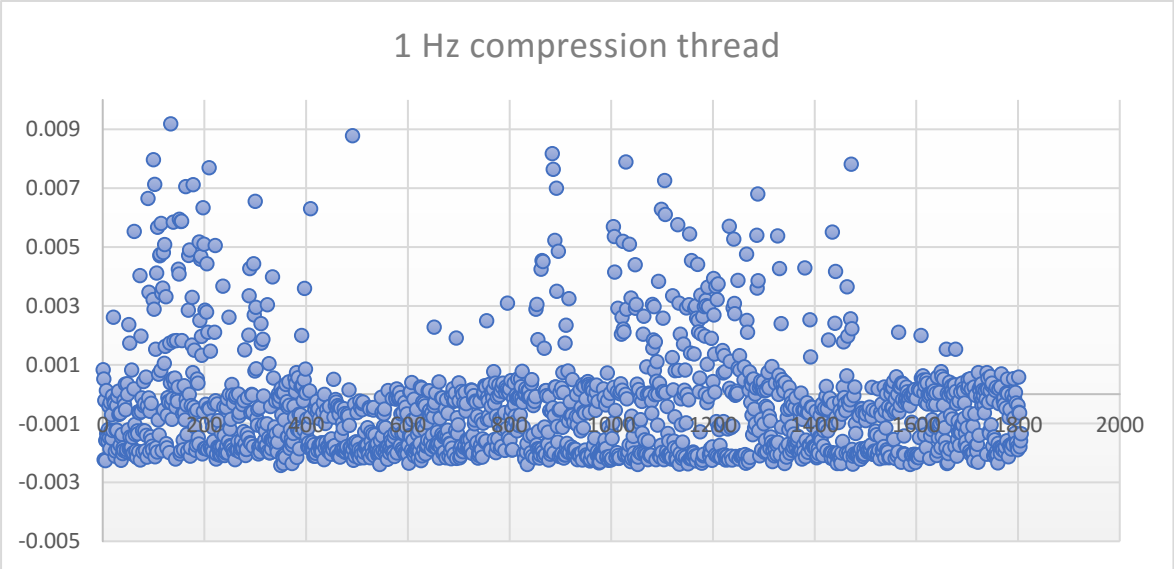
Now after capturing 1800 frames at 1hz, we get a latency which is less than 2s and gives us a maximum of 2 frames lost in 30 mins is 1800 secs giving us an average jitter of 0.6ms per frame.
Similarly, running at a higher frequency gives us a higher jitter of around 3ms per frame. This sounds low but accumulates to more than 18 seconds of delay in 6000 frames. This gives us an accuracy of around 96.11%.

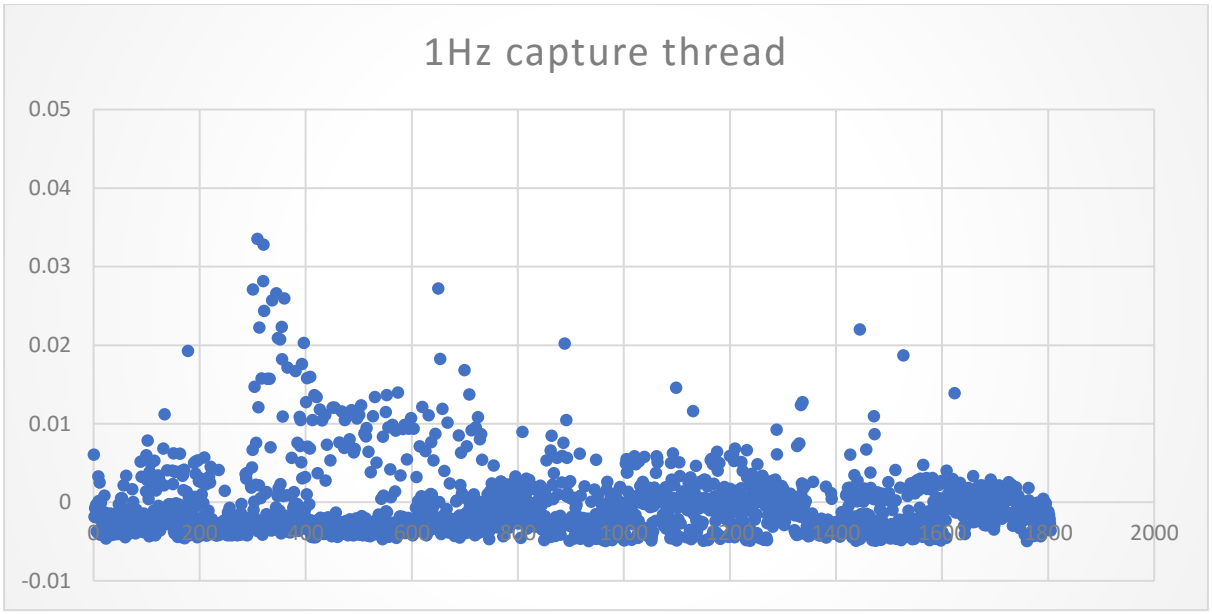
The log files for both the frequencies are attached.
From the values we note down the table for average jitter and total latency.

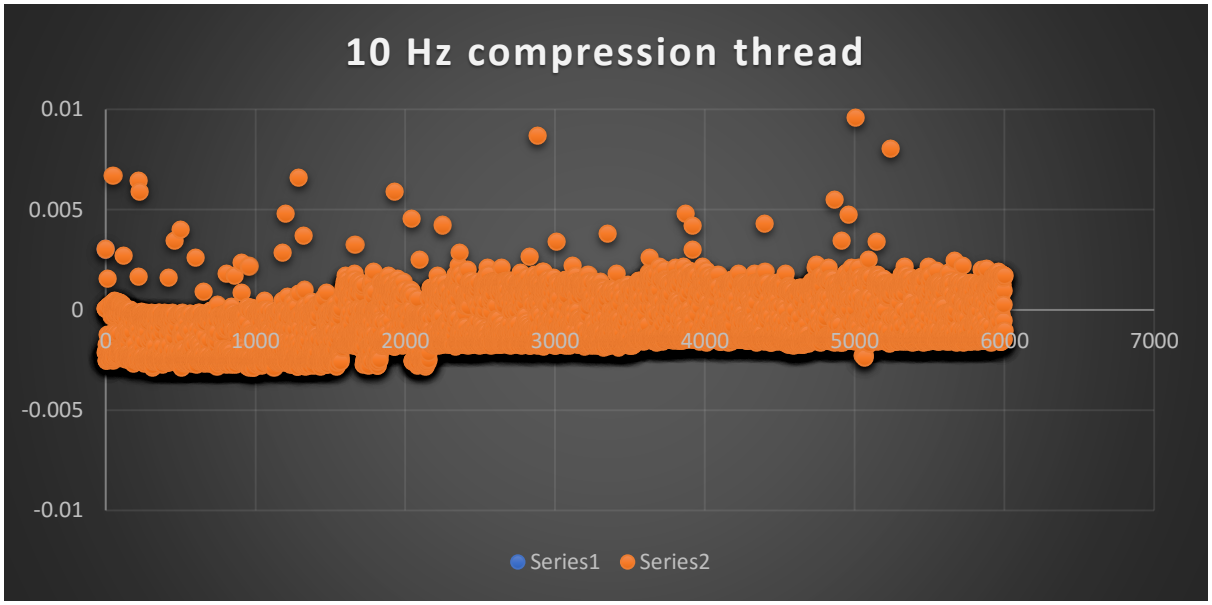
Services	Average_jitter (s)	Total latency (s)
Frame capture with conversion 1hz	-0.00060	-1.09176
10hz	-0.64114	21.60303
Compression 1hz	-0.00036	-0.64114
Total 10hz	-0.00043	-2.58140

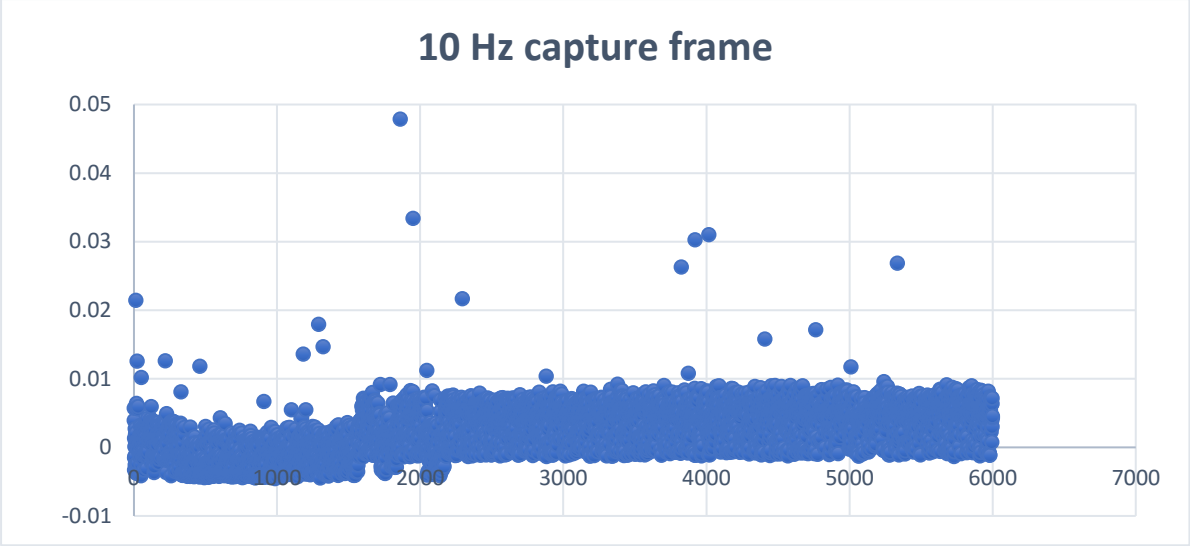
The first ppm file shows the time of 12:45:34 while the 1800th frame is at 1:15:32 showing a difference of 2 s. The video captured using ffmpeg at the rate of 30fps is attached.

To make a graph of jitter, it can be shown below for both threads with both frequencies.




1Hz capture thread
0.05
0.04
0.03
0.02
0.01
0
-0.01
2000


10 Hz compression thread
0.01
0.005
0
-0.005
-0.01
1000
2000
3000
4000
5000
6000
7000
Series1
Series2



CONCLUSION:

Hence, after calculating the jitter and accuracy of the camera and the system to get calculated values at certain frequencies, we get different accuracies with the 1hz capture being more accurate. We are able to reach the accuracy desired as well as print the ppm frame with resolution, time stamp and user name.

Since the real-time threads involve writing in ppm and jpeg the pc takes some time to access and write the disc. This as a result, adds to the latency desired for higher frame-rate. This project can have a more accurate and deterministic response by using a hardware timer instead of a software timer and avoiding the use of writing into the disc in real-time for higher frame rate.

Hence, the minimum, target and stretch goals are targeted.

REFRENCES:

- Simple capture code provided by Prof. Sam Siewart in this course have been the reference for my code.
- Linux manual pages
- OpenCv manual