University of Colorado Boulder

ECEN 5623 Real Time Embedded Systems

Course Project Proposal

Dhiraj Bennadi

April 16th, 2022

Guided By: Professor Tim Scherr

1 Proposal

For the ECEN 5623 Course Project, I plan to implement a time-lapse image acquisition system that captures images of a physical process along with an analog clock and displays the images captured at a different rate than the one used for capturing.

Time Lapse Image Acquisition is a technique in which the frequency at which the frames are captured (the frame rate) is much lower than the frequency used to view the sequence. When played at normal speed, time appears to be moving faster and thus lapsing. Similarly the frames captured can also be played at a much lower rate than at which it was captured which slows down an otherwise fast action, creating a slow motion of high-speed photography The applications of time lapse image acquisition and compression vary from film-making, scientific processes to non-commercial uses.

The project intends to implement this Time Lapse Image Acquisition Technique as a real-time embedded system which meets the deadlines of image acquisition, image processing, image time stamping and image encoding on a Nvidia Jetson Nano system on chip board. The projects intends to use the image capturing and processing methods using the OpenCV Library, a popular open source library for real-time image processing. The project also intends to use the basic real time principles and methodologies of handling system resources, memory, synchronization, core affinity, schedulers, thread programming that are acquired as part of learnings from the course.

The system is modelled in a way to use threads, with each thread with a specific task labelled as Services. On a high level, the system intends to use a supervisor thread with its main functionality of synchronization among the various services in the system. Service 1 would be responsible capturing of the images of from the USB Camera. Service 2 would be responsible for conversion of the captured frame into RGB format. Service 3 would be responsible for time stamping of the image and encoding/compression and storing the image at the desired location.

The course project will be implemented individually, and the following high levels tasks would be carried out as part of the project.

High Level Tasks

1. Capturing of Images at VGA Resolution from an USB Camera System
2. Capturing of the Images in PPM P3 or P6 RGB format
3. Capture the Images for a time sequence of 2000 seconds of an analog clock and a physical process
4. Processing of the images using image processing techniques of OpenCV
5. Use of threads for handling of the various tasks in the system
6. Use of thread synchronization mechanisms to achieve coordination in the tasks processing

Expected Output/Testing

The application software is run for a period of 2000 seconds monitoring an analog clock and a physical process. Since the image acquisition to image store sequence is scheduled to be run at the frequency of 1 Hz, after the execution period, 2000 jitter free images with the first frame to record the start time and the last frame to record the end time (start time + 30 mins) would be the main objective. The images would also be manually evaluated to check the quality of the images. To test the feasibility of the systems simulation tools such as Cheddar would be used.

2 Functional Requirements for System Design

1. Capturing of Images at VGA Resolution (640 * 480)
2. Acquisition of Individual frames from the USB Camera using either file operations or image capturing techniques of OpenCV
3. Capturing of Images at the frame rate of 1 Hz
4. Implement a timing mechanism using a PIT Timer or a POSIX clock or use of accurate sleep time duration for accurate time signalling
5. Capture of Images in the PPM P3 or P6 RGB format
6. Capture of Images for 2000 seconds of a physical process and an analog clock
7. Timestamping of the images by embedding the frame number in the PPM header
8. Implement Image Processing features for image enhancement

3 High Level System and Software Design

3.1 High Level Block Diagram

The below figure captures the idea of a High Level Block Diagram. The blocks represent the services that would be implemented in the system. At the time of the course project proposal, the system functionality is divided into 3 services.

1. Image Acquisition
2. Conversion of Image into RGB format
3. Time stamping of the Image + Encoding/Compression and storage of the image

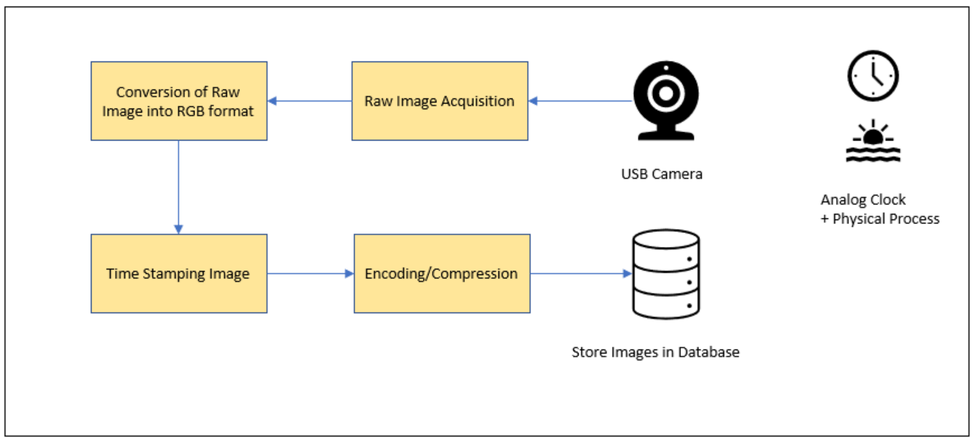

Figure 1: High Level Block Diagram

3.2 Hardware Block Diagram

The below figure captures represents the Hardware involved in the system. The Nvidia Jetson Nano Board would be the Processing System involved in the image acquisition, image processing and storage of the image. The Hardware used to capture the image is an USB Camera (Logitech C920s HD Webcam). The USB camera and the Jetson Nano board are interfaced via the USB Communication Protocol.

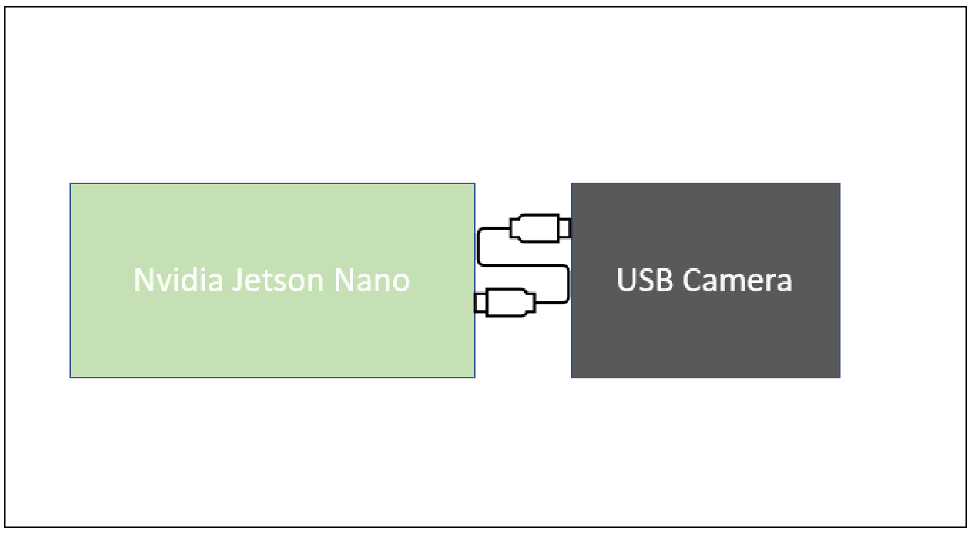

Figure 2: Hardware Block Diagram

3.3 Software Architecture Diagram

The below figure represents the software architecture of the system. The application code invokes API calls from a range of standard libraries to custom libraries used particular for this application.

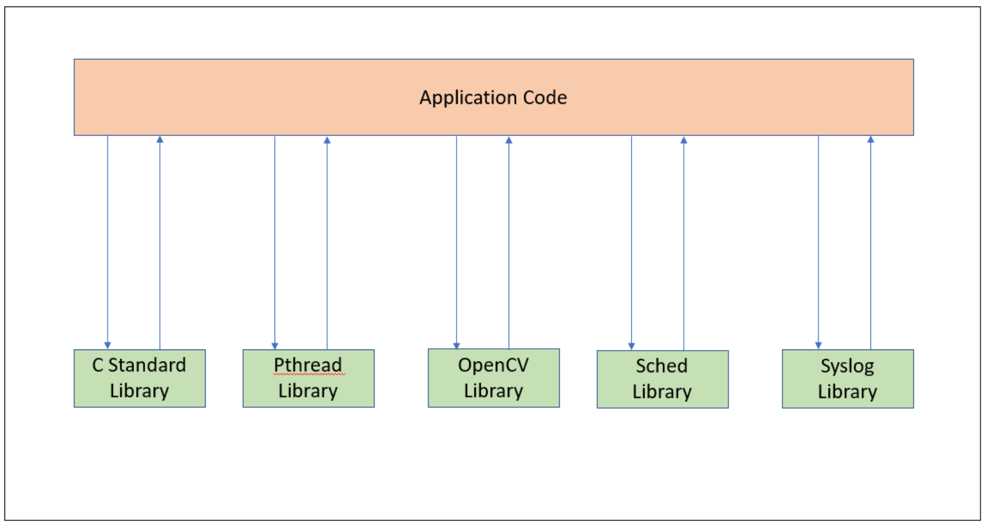

Figure 3: Software Architecture Diagram

Libraries to be used

1. stdio.h, stdlib.h , unistd.h are used for type definitions, standard system calls.
2. Pthread libraries are used for creation of threads, joining of threads for execution
3. OpenCV Library is used for image capture, image processing
4. Sched Library is used for assigning scheduler properties to the threads and also set the scheduler policy
5. Syslog library is used for logging of the timestamps

3.4 Flowchart

The system execution begins with the main function which handles the initializes of the scheduler policy, attaching thread affinity to the cores in the CPU, initialization of the USB device (Camera), and creation of threads with appropriate priority. The execution of threads is also triggered by the main function.
The control and synchronization is mainly with the Sequencer thread. The synchronization is mainly handled by the usage of semaphores. Based upon the frequency of the thread execution, the Sequencer allows the execution of the thread functionality by releasing/posting the semaphore upon which the thread is waiting for execution. Each of the services in the system have a similar structure of waiting on a semaphore and once the Sequencer posts the semaphore, the execution is carried out. Appropriate timestamps are recorded on each execution of the thread and are logged in the syslog file.

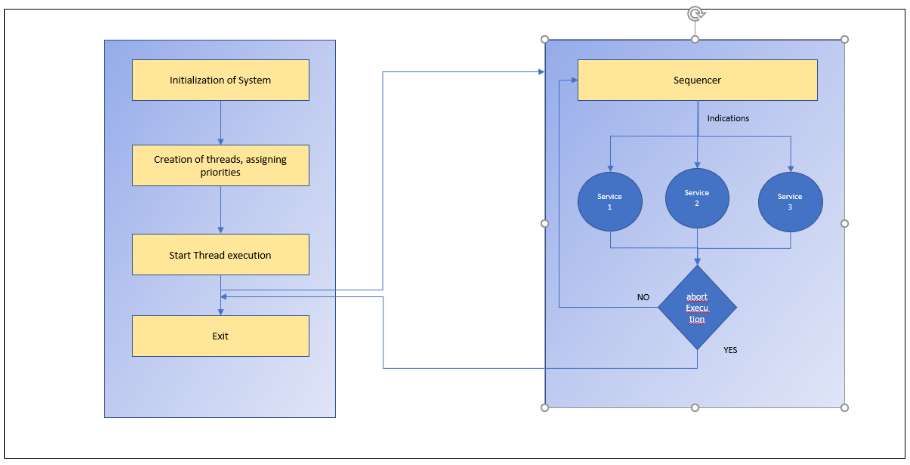

Figure 4: Software Flowchart

4 Real Time Services and Requirements

The system executes 3 services to achieve the time-lapse image acquisition. The system for all the 3 services to complete the cycle from capture to storage of the image has a deadline of 1 sec. As per the RM policy the deadline is equal to the time period and hence the time period for the whole sequence of events is 1 sec. At the time of proposal all the execution times are considered to be the Worst Case Execution Time. Further optimization for reducing the execution times will be attempted

Parameter	Value
Time Period (Ti)	1000 msec
Deadline (Di)	1000 msec
Execution Time (Ci)	200 msec

Table 1: Timing Parameters.

Service 1

This service is responsible for capturing of the images from the USB Camera.

Parameter	Value
Time Period (Ti)	1000 msec
Deadline (Di)	1000 msec
Execution Time (Ci)	50 msec

Table 2: Timing Parameters Service 1

Service 2

This service is responsible for conversion of the raw image captured into RGB format.

Parameter	Value
Time Period (Ti)	1000 msec
Deadline (Di)	1000 msec
Execution Time (Ci)	100 msec

Table 3: Timing Parameters Service 2

Service 3

This service is responsible for time stamping the image and storage of the image at the desired destination

Parameter	Value
Time Period (Ti)	1000 msec
Deadline (Di)	1000 msec
Execution Time (Ci)	50 msec

Table 4: Timing Parameters Service 3

References

[1] Real-Time Embedded Components and Systems with Linux and RTOS - Dr Sam Siewert and John Pratt

[2] Code Reference

[3] Open CV APIs