Be Boulder.

REAL TIME SIGN LANGUAGE INTERPRETER

ECEN 5623: Real-Time Embedded Systems

PRESENTED BY:

Akshit Shah
Vijoy Sunil
Vikhyat Goyal

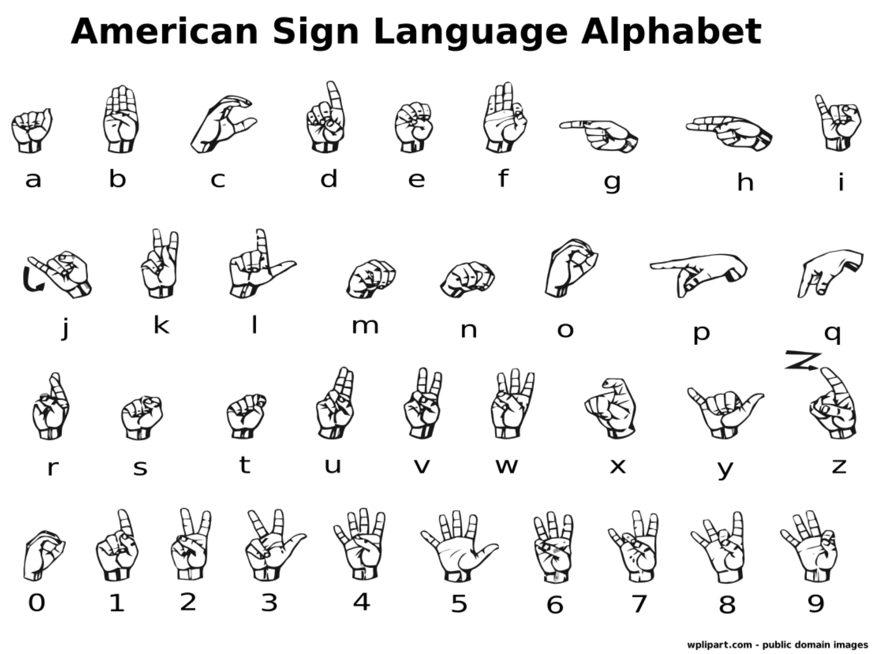
5th May 2017

INDEX

1. Introduction……………………………………………………………………………………………….3
2. Functional Requirements……………………………………………………………………………….4
3. Real time Requirements………………………………………………………………………………….5
4. Functional Design overview and Diagrams……………………………………………………….6
5. Real time analysis and design with Timing Diagrams………………………………………….11
6. Proof of Concept with example outputs and tests completed……………………………….14
7. ASL tests……………………………………………………………………………………………………16
8. Conclusion…………………………………………………………………………………………………18
9. Formal References…………………………………………………………………………………………19
10. Appendix A: ASL test Results…………………………………………………………………………20
11. Appendix B: Code………………………………………………………………………………………25
12. Appendix C: Group Members………………………………………………………………………37

Introduction

Since human existence, many have been physically and mentally challenged with various impediments that hinder their advancement on all economic, political, and social levels. Statistics around the world show a relatively high rate of people with hearing and speaking difficulties. In this report, we present a system that enables hearing impaired and mute people to further connect with the society and aid them in overcoming communication obstacles created by the society's incapability of understanding sign language. The system we propose is based on translating motion into text; it suggests an approach involving human gesture and sign language recognition via a computer processor that assimilates these gestures and signs to produce their equivalent characters. The idea is to convert the method of expression from one form to another in-order to achieve a better understanding by both. The usual way of communication followed by first individual is through signs enacted using hands, this also includes using fingers to spell out the words, each letter at a time. The system achieves real time conversion of these finger signs to text by using image processing and RTOS concepts. We begin by presenting the functional requirements of the system followed by real time requirements. To clearly demonstrate the system and working model we provide functional design overview and diagrams. Later we exhibit Real time analysis and proof of concepts showing are system capable of obtaining desired output. We also added code and reference material for future reference and extension of the system.



Functional (capability) Requirements

Req. ID	Req. Level	Req. Type	Requiement	Test method	Source Link	Design Link
SLI_REQ_1	Capture	Hardware	The Camera should have resolution of atleast 320 x 240.	Review	NA	H1
SLI_REQ_2		Timing	The frames should be captured at a rate of 30fps. And image proessing should be done within 70 ms	System Test	NA	
SLI_REQ_3		Functional	The image should be captured and storred in a sharred memory as soon as possible.	Unit Test	Thread_1	S1
SLI_REQ_4		Safety	No image corruption should happen i.e., A new image should only be captured once the system has processed and extracted the sign from the privious image.	System Test	NA	S1
SLI_REQ_5	Detection	Timing	Hand should be detected , extracted and contoured within 30 ms	System Test	NA	
SLI_REQ_6		Functional	Image should be converted into greyscale image and hand should be detected using contour population and extraction. A configuration stage should be implemented to store the static background without any hand. Once the configuration is achieved, the algorithm should start looking for any new contour and identify it as the hand using a edge detection algorithm.	Unit Test	Thread_2	S2
SLI_REQ_7		Functional	An origin point should be identified at the lower right arear at the base of the palm. All the calculations for each finger position detected are done with reference to this origin point.	Unit Test	Thread_2	S2
SLI_REQ_8		Safety	Hand detection should be protected against random blob detection by always selecting the contour with maximum area in the image frame.	System Test	Thread_2	S2
SLI_REQ_9		Threshold	The image greyscale convertion value is 120	System Test	NA	
SLI_REQ_10	Isolation	Timing	Fingers should be detected, extracted and identified within 50 ms	System Test	NA	
SLI_REQ_11		Functional	All five fingertips should be isolated from the hand. A filter should be defined to extract clearly all five of them with all background noise cancelled. The original image from the catured stage is used to extract the finger position by using specific color filters. The fingers are assosiated with color as given below: 1) Thumb : Black 2) Index : Red 3) Middle : Green 4) Ring : Yellow 5) Little : Blue	Unit Test	Thread_3	S3
SLI_REQ_12		Safety	The threshold for the colors should be set in a way so that no false detection is possible. Clear detection should be possible for all five colors with no overlap in a range of lighting conditions.	System Test	Thread_3	S3
SLI_REQ_13		Threshold	The Threshold for the colors is as below: Black : 0-0-0 Red : 255-0-0 Green : 0-255-0 Yellow : 255-255-0 Blue : 0-0-255	System Test	NA	
SLI_REQ_14	Look-up	Timing	It should be possible to extract the sign from the ROM lookup table within 150 ms	System Test	NA	
SLI_REQ_15		Functional	A shared look-up table should be implemented in the ROM using binary sreach tree. This tree should prestore the Unique Identification Key for each American Sign Language gesture.	Unit Test	Thread_5	S5
SLI_REQ_16		Functional	On receiving the Unique Identification Key , a binary search should be carried on the Look-Up table implemented in the ROM to find the corresponding letter to be displayed. This should be passed over a message queue to the display function.	Unit Test	Thread_5	
SLI_REQ_17	Display	Timing	The display function should be processed within 40 ms.	System Test	NA	
SLI_REQ_18		Functional	The display function should be able to print the letter extracted from the look-up on a external LCD or terminal.	Unit Test	Thread_6	H4,H5,S6
SLI_REQ_19	Training	Safety	There should be no overwriting of a letters reference image with another letter during traing. The images should be stored in a a subfolder for maintainance.	System Test	NA	
SLI_REQ_20		Functional	The system should have a feature to allow new users to train the letters. This is required for a better performance in a varying range of environment and avoid any loss of efficiency due to difference in users.	Unit Test	Main	

Real-Time Requirements

There are five real-time services for this system. These are divided according to the major tasks to be done to achieve a real-time sign language interpretation. The system goal is to achieve a response within 350 milli-second. This implies that the real time deadline of each service can be considered as 330 msec. To account for the safety margin of the system, design goal is to have all the services run once every 350msec, hence we have chosen the real-time deadline of each service to be 330 msec. The deadline has been considered as the period of each service for this system for a predictable and accurate execution.

RT_req_1): The system should be able to decode the sign with maximum accuracy within 350 msec.

RT_req_2): The system should have a safety margin of 10% accounted for overall runtime.

RT_req_3): The following real time service (RT_req_2.1 – RT_req_2.5) timing requirements must be met by the system:

Real-Time Requirement ID	Real-Time Service	Deadline	Period	Semaphore Used
RT_req_2.1	th1_captureimage	330 ms	330 ms	Waits for : SIGNDECODED_SM Releases : NEWIMAGEREADY_SM
RT_req_2.2	th2_extracthand	330 ms	330 ms	Waits for : NEWIMAGEREADY_SM Releases : HANDREADY_SM
RT_req_2.3	th3_extractfeature	330 ms	330 ms	Waits for : HANDREADY_SM Releases : FEATUREREADY_SM
RT_req_2.4	th4_identifyletter	330 ms	330 ms	Waits for : FEATUREREADY_SM Releases : SIGNREADY_SM
RT_req_2.5	th5_displayletter	330 ms	330 ms	Waits for : SIGNREADY_SM Releases : SIGNDECODED_SM

RT_req_4): The system should use SCHED_FIFO policy for achieving a fixed priority scheduling.

RT_req_5): The linux POSIX threads used should be profiled and timestamped for debugging and analysis.

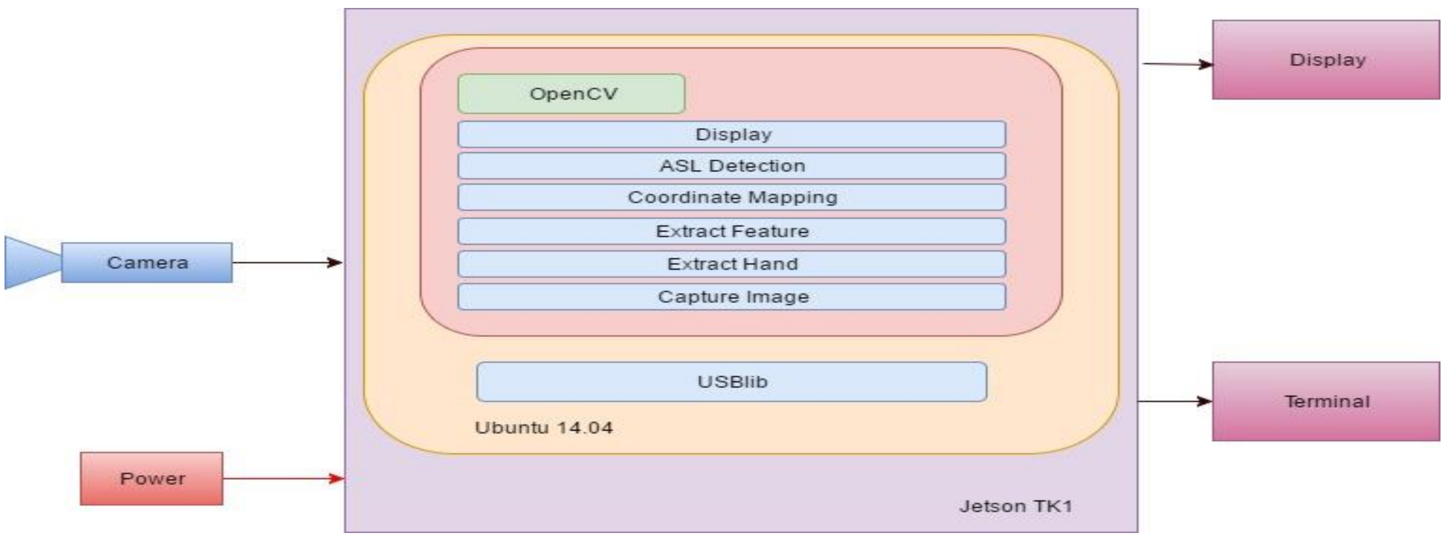
RT_req_6): The service “th1_captureimage” should flush the camera buffer on every call and discard all the frames but the last one to achieve a faster performance and remove the lag created by the ‘vediocapture()’ OpenCV buffers.

RT_req_7): The service “th2_extracthand” should reset the background if too much noise is observed in the frame. This should be done using a global message passed between the services “th3_extractfeature” and “th2_extracthand”. The resetting should not cause the execution time of service to jitter for more than 10%.

Functional Design Overview and Diagrams

1. HARDWARE BLOCK DIAGRAM

The block diagram clearly indicates the required hardware and flow of the system. It primarily consists of 3 main building blocks. The input camera, the image processing board (NVIDIA Jetson TK1) and output display. In our project, we are using NVIDIA Jetson TK1 development board to perform real-time tasks. We are choosing Jetson because of its power of the GPU for embedded systems applications. We are planning to use its Tegra-accelerated OpenCV for quickly developing and deploying compute-intensive system for our project. To capture the sign language, we are using Logitech C200 camera which we have integrated with the board. The output can be displayed on the terminal or LCD screen indicating the corresponding characters.



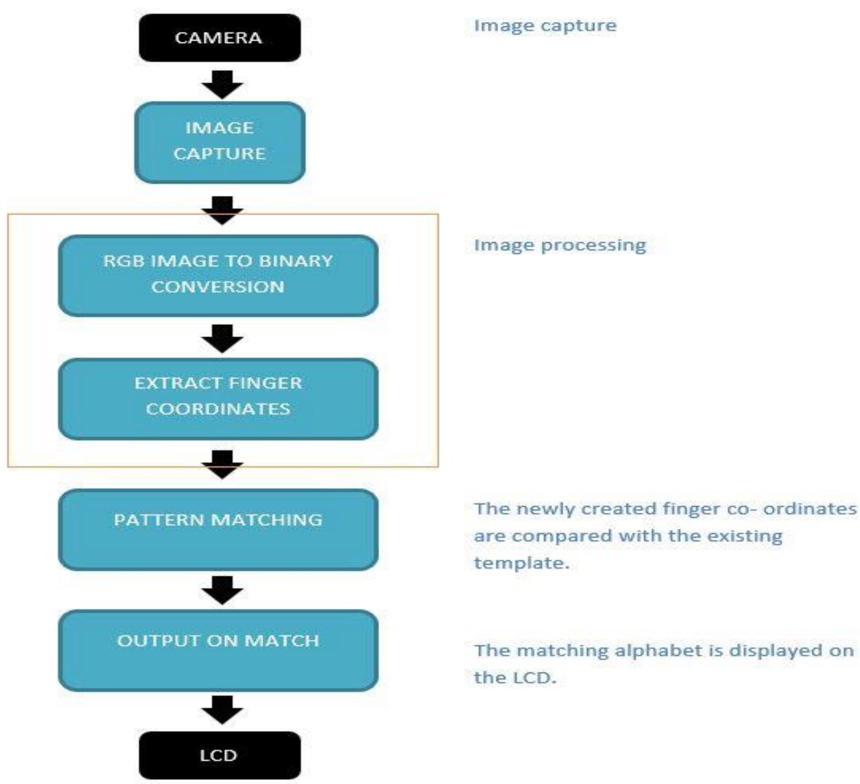
NVIDIA JETSON TK1:

Jetson TK1 is NVIDIA's embedded Linux development platform featuring a Tegra K1 SOC (CPU+GPU+ISP in a single chip). Jetson TK1 comes pre-installed with Linux4Tegra OS (basically Ubuntu 14.04 with pre-configured drivers). There is also some official support for running other distributions using the mainline kernel, discussed further in the Distributions and Mainline kernel sections below. Besides the quad-core 2.3GHz ARM Cortex-A15 CPU and the revolutionary Tegra K1 GPU, the Jetson TK1 board includes similar features as a Raspberry Pi but also some PC-oriented features such as SATA, mini-PCIe and a fan to allow continuous operation under heavy workloads. We are using it for our image processing operations and storing data base.

Logitech C200:

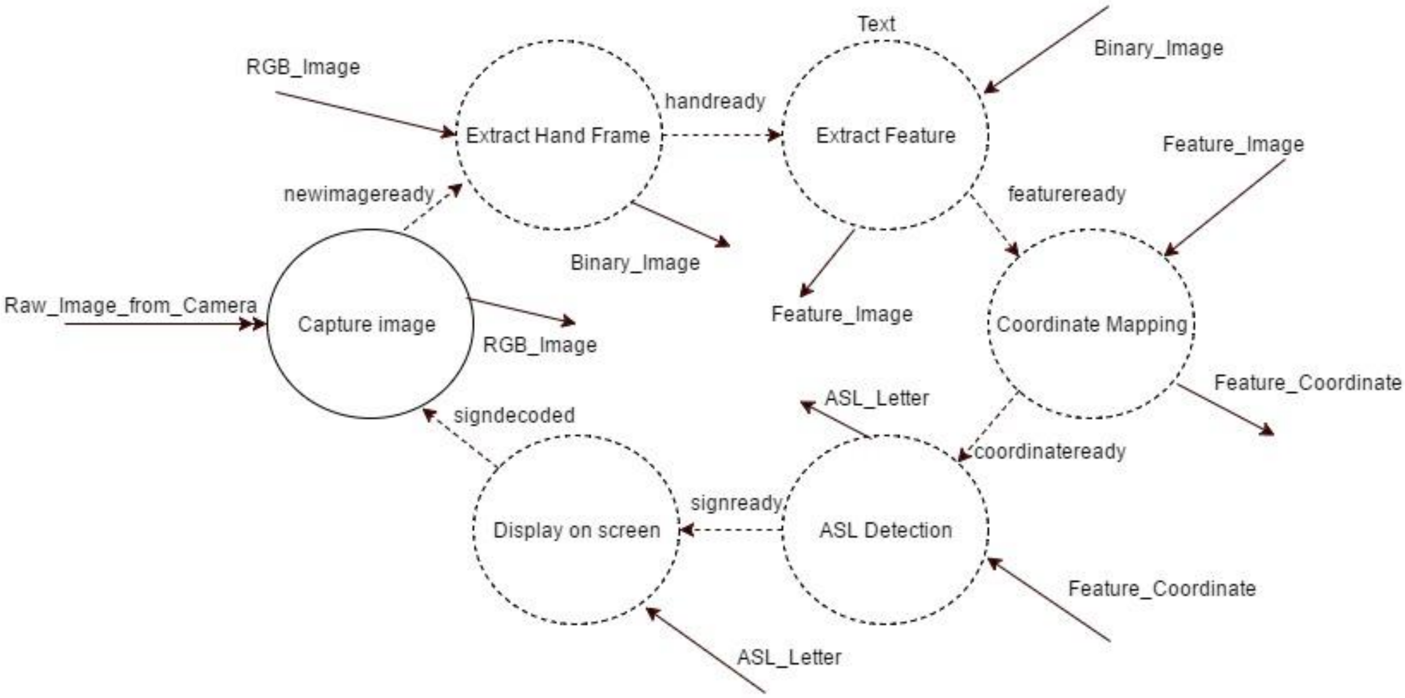
Logitech C200 is VGA camera with frame rate of 30 frames per second. The video capturing capability is 640 x 480 pixels. Accounted with the low cost and efficiency this camera was perfect for our system. The camera was used to continuously stream video to capture hand gesture and give it to Jetson TK1 to process it.

2. SOFTWARE BLOCK DIAGRAM



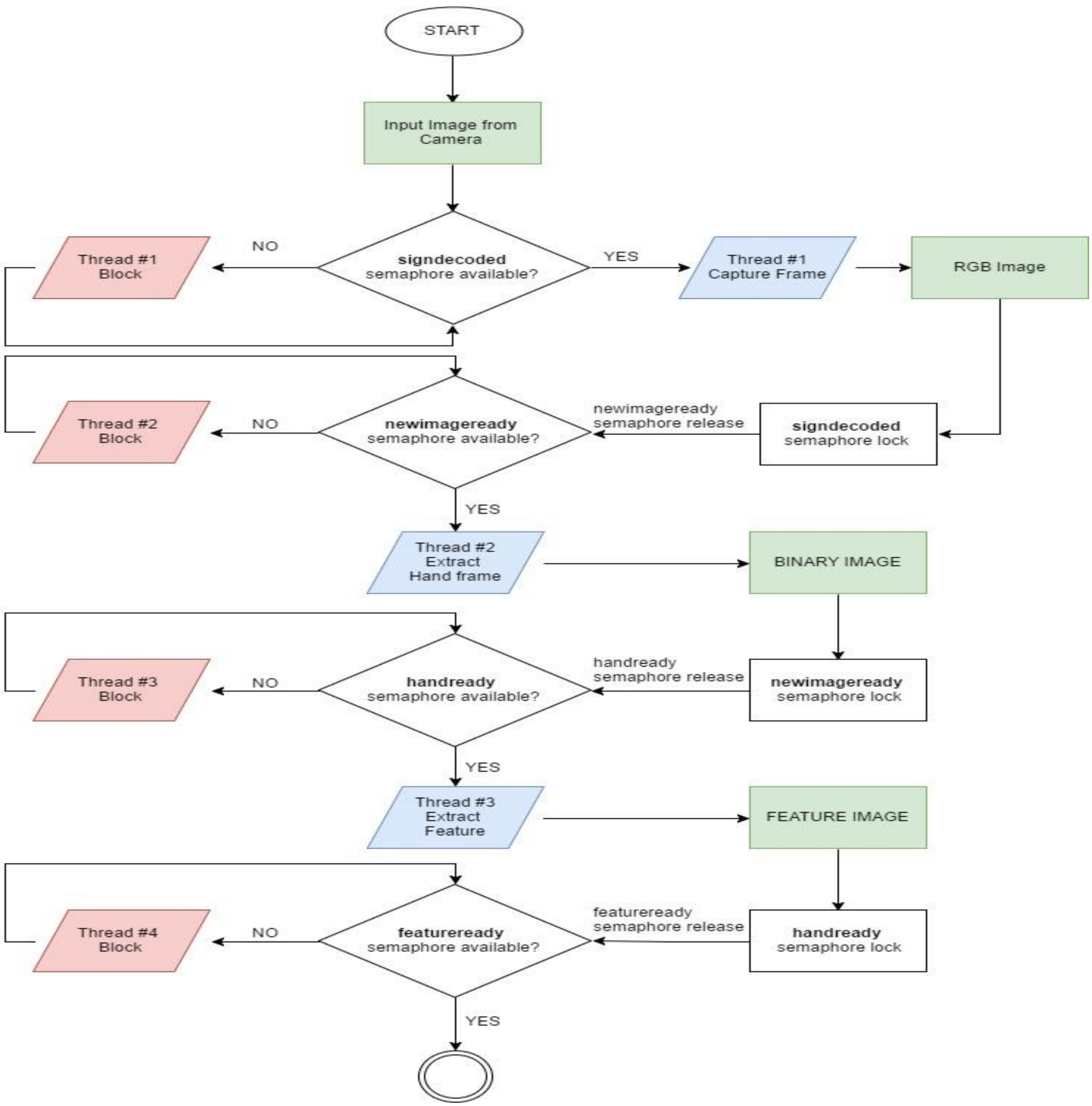
The above diagram shows the software flow of the system. Frames are continuously captured from video camera. Image processing techniques are applied using OpenCV functions to carry out thresholding, contouring and background subtraction to obtain a well-defined hand gesture image. To obtain more precision and quicker response we added an option to train the system. The user can train the system and modify the signs to match the corresponding character. This image is then compared with given data base which consists of images of all the characters. Pattern matching technique is applied to check if the given sign matches with any of the data base letters. If it correctly matches, then the matched image is displayed on the screen along with the corresponding character.

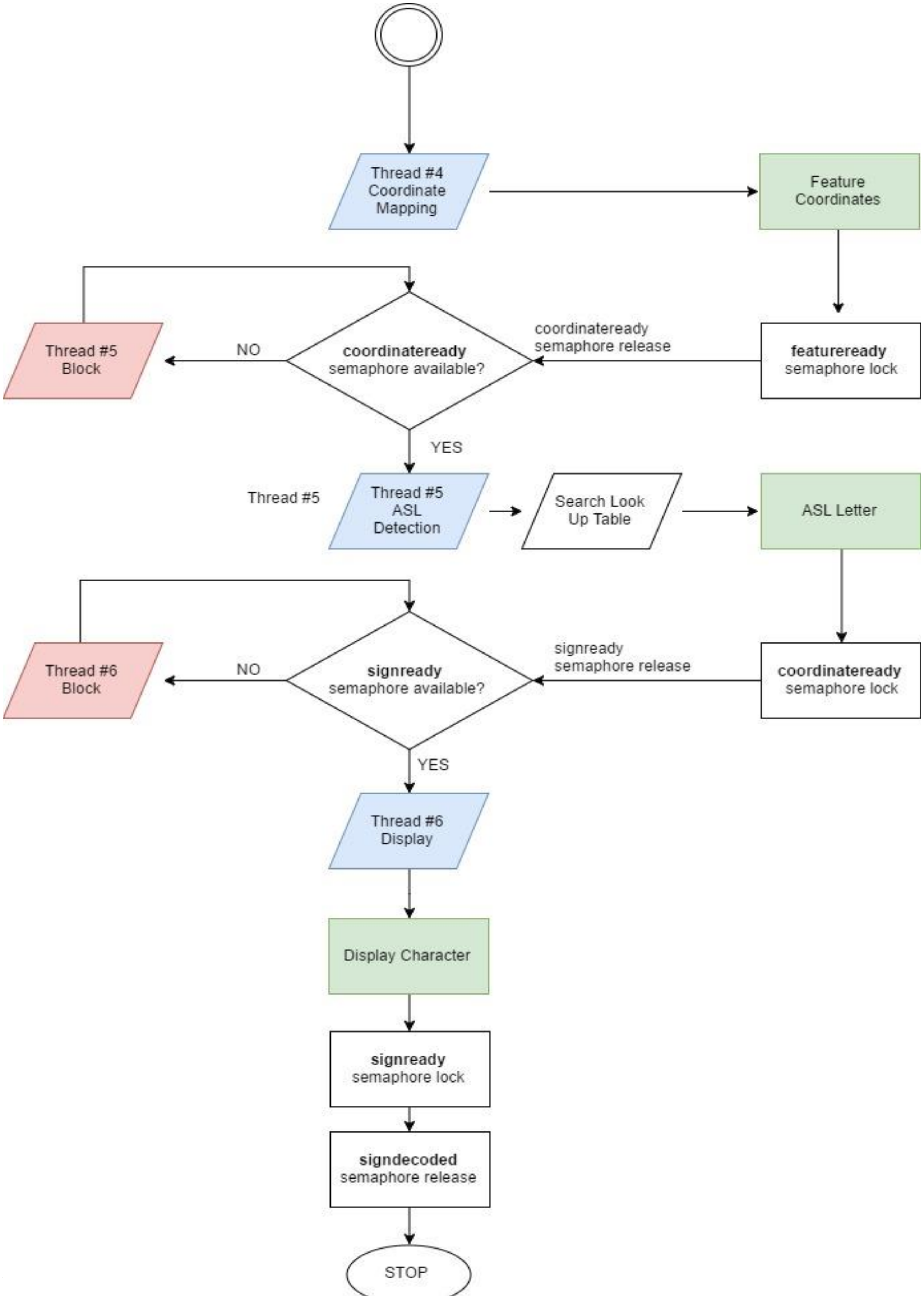
3. CONTEXT FLOW DIAGRAM



The context flow diagram and flow chart generalizes the function of the entire system in relationship to external entities. The entire system "Real time Sign Language Interpreter" is treated as a single process and all its inputs, outputs, sinks and sources are identified and shown. The 1st thread in the system is capturing the hand gesture(sign) which is a raw image and giving out RGB image to the next thread to extract the hand out of the entire image and discarding rest of the image. The hand frame is then converted to greyscale image clearly indicating the area of interest and blacking out rest part. In the next thread the binary image is then fine-tuned by further thresholding to eradicate unwanted noise in the image. Also, we define origin for mapping in this thread. The feature image in the following thread is then mapped per designed algorithm which calculates the finger-tip positions with respect to origin. The mapped data is passed to thread 5 which runs a matching algorithm to match mapped data with the database which w build indicating corresponding character to the gesture. In the final thread the recognized character is displayed on the LCD or host terminal.

4. FLOW CHART


START
Input Image from Camera
signdecoded semaphore available?
NO
Thread #1 Block
YES
Thread #1 Capture Frame
RGB Image
signdecoded semaphore lock
newimageready semaphore release
newimageready semaphore available?
NO
Thread #2 Block
YES
Thread #2 Extract Hand frame
BINARY IMAGE
newimageready semaphore lock
handready semaphore release
handready semaphore available?
NO
Thread #3 Block
YES
Thread #3 Extract Feature
FEATURE IMAGE
handready semaphore lock
featureready semaphore release
featureready semaphore available?
NO
Thread #4 Block
YES


Thread #4
Coordinate
Mapping
Feature
Coordinates
featureready
semaphore lock
coordinateready
semaphore release
coordinateready
semaphore available?
NO
Thread #5
Block
YES
Thread #5
Thread #5
ASL
Detection
Search Look
Up Table
ASL Letter
coordinateready
semaphore lock
signready
semaphore release
signready
semaphore available?
NO
Thread #6
Block
YES
Thread #6
Display
Display Character
signready
semaphore lock
signdecoded
semaphore release
STOP

Real-Time Analysis and Design with Timing Diagrams

The system has the overall requirement to produce the output within a real-time deadline to properly comprehend the actions. We have selected this system deadline as the deadline for all our threads as a complete and error free recognition of the letters will require all threads to complete before this system deadline.
At every instance of system, all the threads should execute to obtain the output. The time period of the system is sum of time period of all services. This implies every service should be executed exactly once within this time period (330 ms). For simplicity of our system we assume Di to be equal to Ti.

Safety margin analysis
The computation time for each service was determined using a profiler. Profiling the services help us to determine the required thread execution time which in turn helps to determine the system time requirements. The worst-case execution time of each thread is calculated by making sure that the thread takes its longest execution path.
The time taken to go through the most computation intensive part, most number output-input accesses, most number of decision and statements is the worst-case execution time and we simulated these conditions for each thread to get their WCET. In addition to this, a 10% margin was added to these measurements to provide safety against unaccounted times

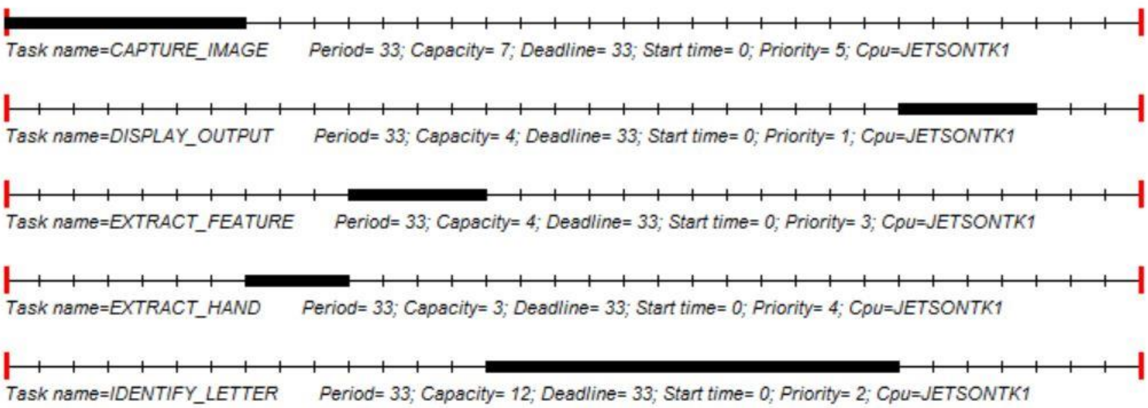
Service	C (ms)	WCET
Capture Image	63.58 65.78 64.14 63.67 62.93	WCET = **66.5 ms** Calculated based on the longest amount of time taken among the first 5 execution instances and adding 10 % safety margin to it.
Extract Hand	24.26 24.40 22.26 8.73 7.03	WCET = **27 ms** The longest path taken by this service is when there is too much noise in the background and the background subtraction function resets the background. A 10% margin is also added to this time to get the worst-case execution time.
Extract Feature	34.71 34.22 35.43 37.87 35.37	WCET = **42 ms** The service creates contours along the background-subtracted image and the longest path taken is when the created contour area is the largest. A 10% safety margin is added to this time to get the worst-case execution time.
Identify Letter	105.32 104.66 101.08 109.89 107.39	WCET = **120 ms** The service scans the list of preloaded images from ‘a’ to ‘z’ and the worst-case execution time is when a match is found at the end of the list – letter ‘z’. A 10% safety margin is added to this time to get the worst-case execution time.
Display output	34.51 33.99 34.58 34.36 34.95	WCET = **38 ms** Calculated based on the longest amount of time taken among the first 5 execution instances and adding 10 % safety margin to it.

Cheddar Worst-Case and Simulation

Service	Priority	WCET(ms)	Ti(ms)	Di(ms)
Capture Image	5	66.5	330	330
Extract Hand	4	27	330	330
Extract Feature	3	42	330	330
Identify Letter	2	120	330	330
Display output	1	38	330	330

Priority 5 > 4 >3 >2 >1

Cheddar Simulation



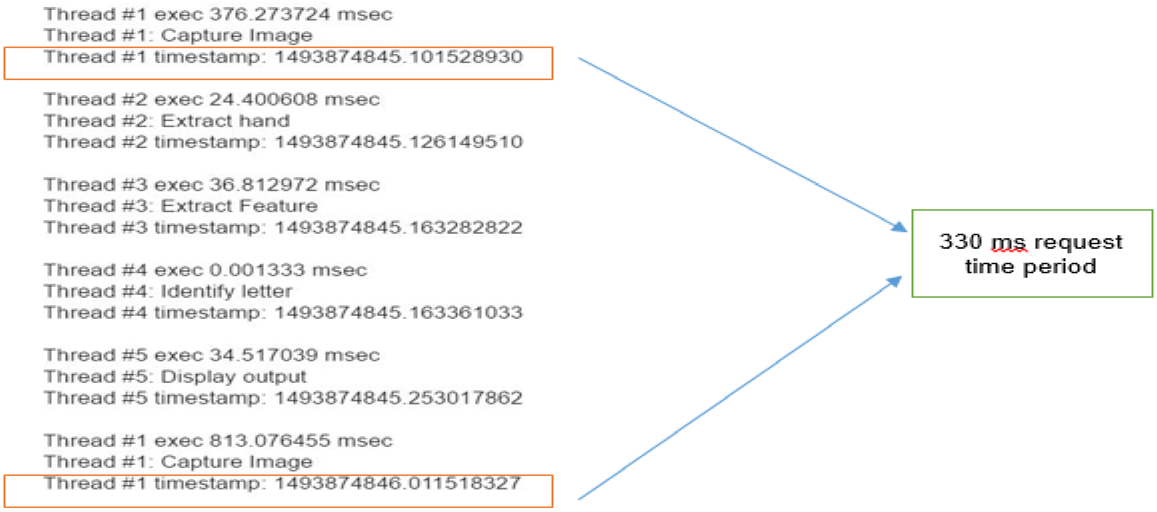
The simulation is run over time period of 330 ms.It can be seen that all the service finish their execution before the deadline.

Cheddar Analysis

```
Scheduling simulation, Processor JETSONTK1 :
- Number of context switches :  4
- Number of preemptions :  0

- Task response time computed from simulation :
    CAPTURE_IMAGE => 7/worst
    DISPLAY_OUTPUT => 30/worst
    EXTRACT_FEATURE => 14/worst
    EXTRACT_HAND => 10/worst
    IDENTIFY_LETTER => 26/worst
- No deadline missed in the computed scheduling : the task set seems to be schedulable.


Scheduling feasibility, Processor JETSONTK1 :
1) Feasibility test based on the processor utilization factor :

- The base period is 33 (see [18], page 5).
- 3 units of time are unused in the base period.
- Processor utilization factor with deadline is 0.90909 (see [1], page 6).
- Processor utilization factor with period is 0.90909 (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).
    DISPLAY_OUTPUT => 30
    IDENTIFY_LETTER => 26
    EXTRACT_FEATURE => 14
    EXTRACT_HAND => 10
    CAPTURE_IMAGE => 7
- All task deadlines will be met : the task set is schedulable.
```

The processor utilization factor for the service set is calculated to be 0.909 which is greater than the Rate Monotonic least upper bound.
Since none of the deadlines are missed, the service set is schedulable.
As per the Cheddar analysis, every thread should have executed at least once with in the least common multiple of the time periods. In this case, since all our time periods are 330 ms and the LCM also being 330 ms, every thread should execute once within this time period.

With the below output from time stamping the threads, it can be seen that there is a difference of 330 ms between the first release and the next of Thread #1 – Capture Image.

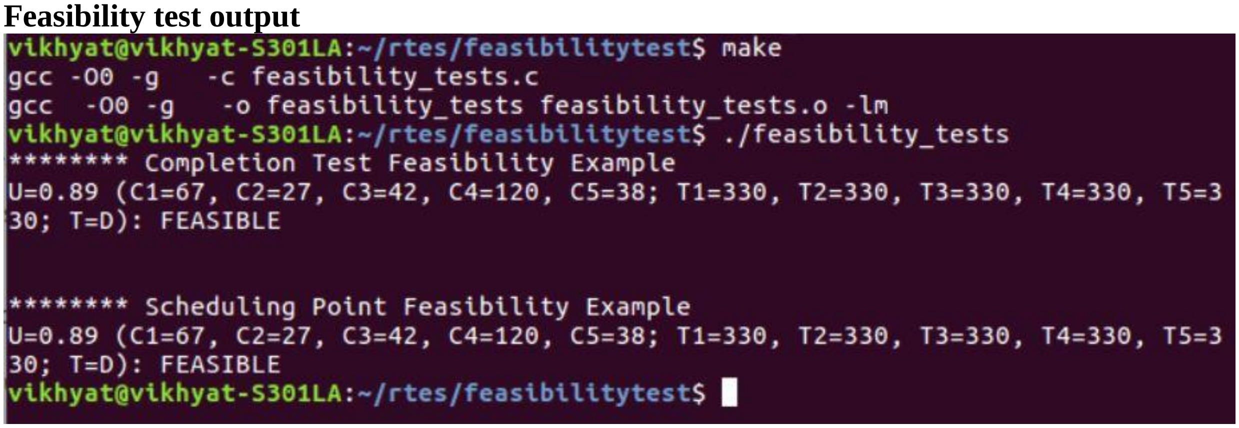


```
Thread #1 exec 376.273724 msec
Thread #1: Capture Image
Thread #1 timestamp: 1493874845.101528930

Thread #2 exec 24.400608 msec
Thread #2: Extract hand
Thread #2 timestamp: 1493874845.126149510

Thread #3 exec 36.812972 msec
Thread #3: Extract Feature
Thread #3 timestamp: 1493874845.163282822

Thread #4 exec 0.001333 msec
Thread #4: Identify letter
Thread #4 timestamp: 1493874845.163361033

Thread #5 exec 34.517039 msec
Thread #5: Display output
Thread #5 timestamp: 1493874845.253017862

Thread #1 exec 813.076455 msec
Thread #1: Capture Image
Thread #1 timestamp: 1493874846.011518327
```


Scheduling Point / Completion tests

The feasibility test code tests the feasibility for RM policy using completion test and schedule point test. The execution times and the time period of the services are fed in and it is found that the service set for this system are feasible and the worst-case execution times computed through our measurements satisfy both the tests.

Feasibility test output



```
vikhyat@vikhyat-S301LA:~/rtes/feasibilitytest$ make
gcc -O0 -g   -c feasibility_tests.c
gcc  -O0 -g   -o feasibility_tests feasibility_tests.o -lm
vikhyat@vikhyat-S301LA:~/rtes/feasibilitytest$ ./feasibility_tests
******** Completion Test Feasibility Example
U=0.89 (C1=67, C2=27, C3=42, C4=120, C5=38; T1=330, T2=330, T3=330, T4=330, T5=3
30; T=D): FEASIBLE


******** Scheduling Point Feasibility Example
U=0.89 (C1=67, C2=27, C3=42, C4=120, C5=38; T1=330, T2=330, T3=330, T4=330, T5=3
30; T=D): FEASIBLE
vikhyat@vikhyat-S301LA:~/rtes/feasibilitytest$
```

The Cheddar Output for the service set also **agrees with the feasibility test** outputs.
The service set is thus both safe and feasible.

Proof-of-Concept with Example Output and Tests Completed

Proof-of-Concept:

Two experiments were designed for checking the possibility of extracting hand gesture from a given frame and using houghdroff distance formula to match two sets of image. This along with the thread schedule feasibility experiment and cheddar tool simulation were used as proof-of-concept.

1) Hand extraction using MOG2 background subtraction:

The first experiment was to call first two services “th1_captureimage” and “th2_extracthand” in a SCHED_FIFO scheme and check for the ability to extract the background from a given image.

Experiment Set-Up:

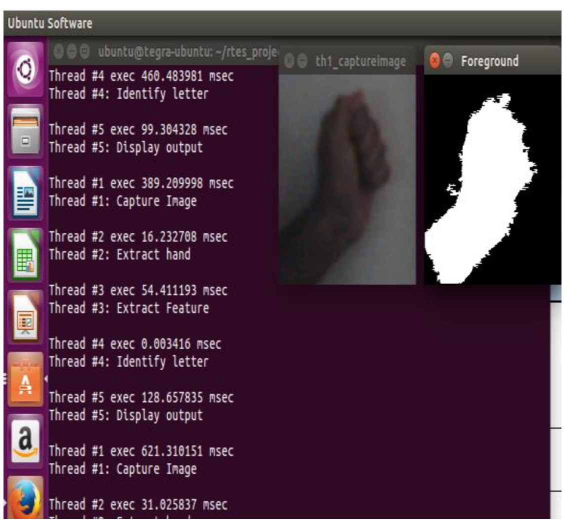
- The experiment setup involved Ubuntu based laptop with the in-built webcam.
- The code resets the background every 10 frames and checks for any new object in the frame subsequently.
- Any additional information in the frame is filled with white colour and the background is kept as black.
- The same code is then run with Jetson TK1 and Logitech camera as an addition check to prove the design.

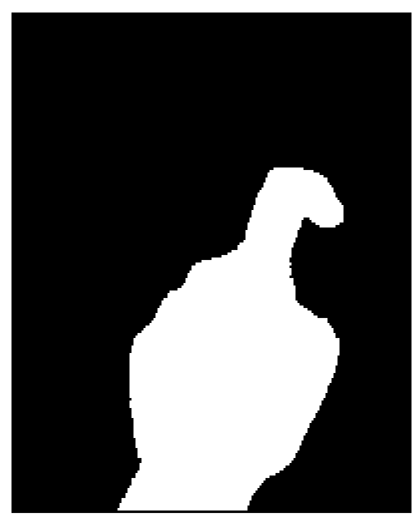
Experiment Results:

It was observed that the system could extract the new object with perfection and a reset after 10 frames was a fast and good solution to shield against changing backgrounds.

The frame rate was found to be a little slower with Jetson and hence the system was updated to perform a flush of the preloaded frames from the buffer to achieve a more predictable and real-time behaviour.

The below screenshots show the effect of background substraction using MOG2 method:



2) Sign identification using Housdroff distance:

The second experiment was to call the service "th4_identifyletter" , feed it with an array of images to be matched using housdroff mathematical formula to find the similarity between two given vector arrays. The setup was done to have a SCHED_FIFO. A comparison was done with the histogram matching method to benchmark the performance of housdroff matching and to give a go-nogo to the algorithm for our system.

Experiment Set-Up:

- The experiment setup involved Ubuntu based laptop with the in-built webcam.
- The code takes two sample set of images as an input and maps the vector distance between the sample set and a preloaded test image.
- The same set of images were then fed into the histogram matching algorithm and match accuracy was recorded.
- The code for housdroff is then run with Jetson TK1 and Logitech camera as an addition check to prove the design.

Experiment Result:

It was observed that housdroff mathematic formula worked as a good matching algorithm with a high accuracy and hence could be used along with background subtraction for ASL translation.

A comparison with the histogram matching algorithm showed that the system works better with housdroff algorithm and hence, this was chosen as the matching algorithm for our system.

The screenshot below shows Housdroff distance calculation for an array of multiple images with varying similarity with a test image:

```
ubuntu@tegra-ubuntu:~/ASL-tegra$ sudo I
[sudo] password for ubuntu:
HIGHGUI ERROR: V4L/V4L2: VIDIOC_S_CROP
 | diff: 1357.07
 | diff: 1977.55
 | diff: 1807.26
 | diff: 1712.53
 | diff: 1681.16
 | diff: 912.759
 | diff: 1755.19
 | diff: 1150.66
 | diff: 1627.21
 | diff: 1247.86
 | diff: 1073.11
 | diff: 1294.79
 | diff: 1266.77
 | diff: 1257.41
 | diff: 1589.99
 | diff: 1666.72
 | diff: 1289.94
 | diff: 503.089
 | diff: 770.314
 | diff: 1821.04
 | diff: 1628.29
 | diff: 1566.99
 | diff: 1458.86
```

ASL Tests:

The system was tested with a two stage testing plan:

Unit Test: Every individual service was tested with the below test specification table and the results were documented:

ASL Unit Test Specifications						
Test Case	**Test Code**	**Test Aim**	**Test Steps**	**Test Setup**	**Test expected result**	**Test Result**
UTC_1	Image Capture	Test access to camera	1) Call th1 only using a one thread system setup 2) Input the camera used as 0 3) Check result 4) Input the ceamra used as 1 5) Check result	Set TEST_MODE = TEST_1_ON	1) At 3 : Camera should be accessible and image frame should be captured. 2) At 5: Camera should not be accessible and error message should be seen on terminal.	Passed
UTC_2		Test frame capture rate	1) Call th1 only using a one thread system setup 2) Profile the time taken between two frame captures.	Set TEST_MODE = TEST_1_ON	The frame should not take more than 70 msec to be captured	Passed
UTC_3		Test Flush of buffers	1) Call th1 only using one thread system setup 2) change the frames to flush to 1 3) check camera delay 4) change the frames to flush to 4 5) check camera delay	Set TEST_MODE = TEST_1_ON	1) At 3: Camera delay should be observed 2) At 5: No camera delay should be observed	Passed
UTC_4	Hand extraction	Test background subtraction	1) Call th2 only using one thread system setup 2) Check the image	Set TEST_MODE = TEST_2_ON	The background image should not be visible	Passed
UTC_5		Test Reset of Background	1) Call th2 only using one thread system setup 2) Set reset as 10 3) Check the image	Set TEST_MODE = TEST_2_ON	The background should be reset to the new image	Passed
UTC_6	Feature Extraction	Contour detection	1) Call th3 only using one thread system setup 2) Bring a closed contour object (hand/ball) in the frame 3) Check that the object is detected	Set TEST_MODE = TEST_3_ON	The object should be extracted from the image	Passed
UTC_7		Contour Threshold check	1) Call th3 only using one thread system setup 2) Set the threshold for contour detection as 255 3) Bring a closed contour object (hand/ball) in the frame 4) Check that the object is detected	Set TEST_MODE = TEST_3_ON	No object should be detected	Passed
UTC_8	Letter Identification	Check for Look-up of letter	1) Call th4 only using one thread system setup 2) Prefeed with a array of dummy image vectors 3) identify if the match algorithm works 4) check the output differnce on console	Set TEST_MODE = TEST_4_ON	The differnece should be minimum for identical images and	Passed
UTC_9	Display	Check that the letter is augmented properly	1) Call th5 only using one thread system setup 2) Stub the "Asl-LETTER" value as 'b' 3) Check the image	Set TEST_MODE = TEST_5_ON	The image should be diaplayed with letter 'b' printed.	Passed
UTC_10	Training	Creating a new reference image	1) Set the system for traing mode 2) Capture a new image 3) Check if the image can be written into a new file	Set TEST_MODE = TEST_Training_ON	The image should be stored in images folder with a name as given during test.	Passed

Note: Results proofs of each test attached in Appendix "A"

System Test: The overall system test involved following the below test specification on laptop and the Jetson TK1 system:

ASL System Test Specifications						
Test Case	**System Requirment**	**Test Aim**	**Test Steps**	**Test Setup**	**Test expected result**	**Test Result**
STC_1	Training	Creating reference image for letters	1) Run the system 2) Selected "y" to start training mode 3) Press 'a' 4) Check the terminal 5) Check images folder	Jetson tk1 with ASL code and logitect connected. SSH through Laptop	1) At 4: Terminal showed "a: written" 2) At 5 : a new image a.png was created in folder images.	Passed
STC_2	Training	Overwriting pre-existing files	1) Run the system 2) Selected "y" to start training mode 3) Press 'a' 4) Check the terminal 5) Check images folder 6) Press "a" 7) Check Terminal 8) Check images folder	Jetson tk1 with ASL code and logitect connected. SSH through Laptop	1) At 4: Terminal showed "a: written" 2) At 5 : a new image a.png was created in folder images. 3) At 7 : Terminal showed "a: written" 4) At 8 : in folder images, a.png was overwritten by the new image.	Passed
STC_3	Training	ESC key ends the testing mode	1) Run the system 2) Selected "y" to start training mode 3) Press 'ESC' key 4) Check terminal	Jetson tk1 with ASL code and logitect connected. SSH through Laptop	1) At 4: The training mode should be let and system should go in full running mode.	Passed
STC_4	ASL Main Mode	Check if letters 'A', 'B', 'C; could be detected	1) Run the system 2) Selected "n" to start training mode 3) Bring the hand in position of 'A' in the frame. 4) Check Letter window 5) Bring the hand in position of 'B' in the frame. 6) Check Letter window 7) Bring the hand in position of 'C' in the frame. 8) Check Letter window 9) Check terminal	Jetson tk1 with ASL code and logitect connected. SSH through Laptop	1) At 4: Letter 'a' should be visible in the window. 2) At 6: Letter 'b' should be visible in the window. 3) At 8: Letter 'c' should be visible in the window.	Passed
STC_5	ASL Main Mode	Thread request frequency analysis	1) Run the system with all services active 2) Run the system for 1000 cycles 3) analyse the result using excel	Jetson tk1 with ASL code and logitect connected. SSH through Laptop	The request frequency calculated should match the request frequency as cacluted using cheddar.	Passed

Note: Results proof of each test attached in Appendix "A"

CONCLUSION

We presented in this report the implementation of a system that aims to translate American Sign Language gestures into text. The system is based on an image processing approach where a video camera is used to capture hand movements. After the hand is detected and tracked using Hausdorff algorithm employing C++'s OpenCV library functions, frames are periodically captured and processed. Background subtraction is first applied, then thresholding and contouring. After contouring, the image is sent to be compared to a set of images in a reference database. The comparison is based on image difference. A threshold level is experimentally determined to decide between a match or mismatch. In case of a match, the meaning of the matched image is displayed onto the screen along with the character. We have used Embedded Linux with POSIX extensions to make it real-time. During the implementation of the system we have considered real-time problems and ensured that shared data is safe and threads are run according to scheduled algorithm. Even though we have successfully demonstrated our project with required test cases and outputs, there is still scope of improvement with better OS and quicker algorithm which are listed below

FUTURE SCOPE

Future enhancement of the system is to make it universal. Since different parts of the world uses different sign language we can train our system and create a new database for different sign and make it usable for any sign language. Also, the system can be enhanced to interpret words and store it which later can be read to create a sentence. The system can also be integrated with other devices which converts text to speech to increase its scope. Embedded Linux involves lot of overhead but we can replace it with FreeRTOS which can improve efficiency and give us quick response.

Formal References (and Attributions to Anyone who helped not on the team)

1. Intelligent Sign Language Recognition Using Image Processing- Sawant Pramada, Deshpande Saylee, Nale Pranita, Nerkar Samiksha, Mrs. Archana Vidya

http://www.iosrjen.org/Papers/vol3_issue2%20(part-2)/H03224551.pdf

2. Real Time Sign Language Conversion project

http://www.ieee.org/education_careers/education/preuniversity/real_time_sign_language.html

3. Vision-based sign language translation device

https://www.researchgate.net/publication/261460857_Vision based_sign_language_translation_device

4. A real-time continuous gesture recognition system for sign language

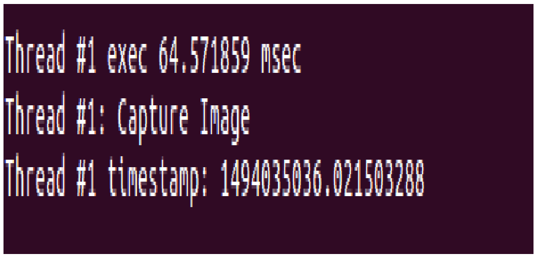
http://ieeexplore.ieee.org/abstract/document/671007/

Appendix A: ASL Test Results

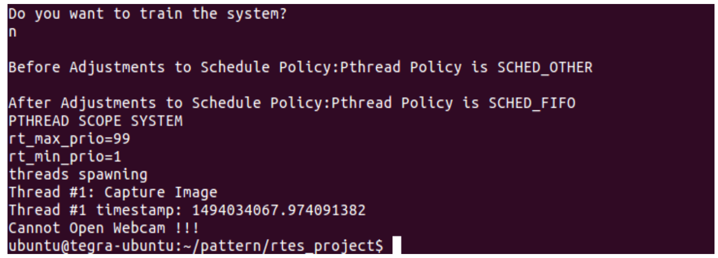
1) Unit Test Results:

UTC_1:

Check with camera available-

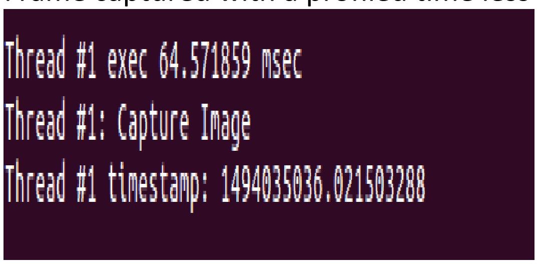


Check with camera not available-



UTC_2:

Frame captured with a profiled time less than 70 msec:

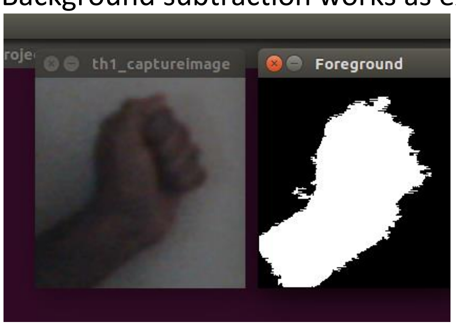


UTC_3:

No camera delay was seen with the correct number of flushed buffers.

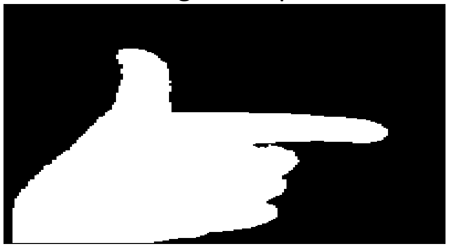
UTC_4:

Background subtraction works as expected:



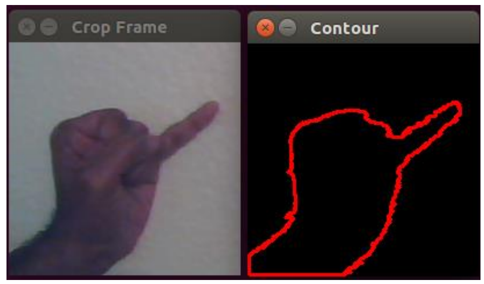
UTC_5:

Reset of background produces better results:



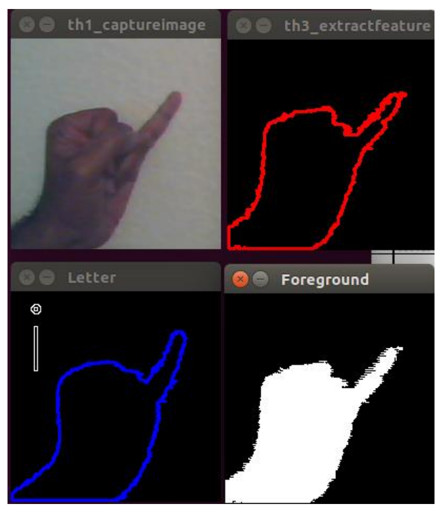
UTC_6,7:

Contour detection works as expected:



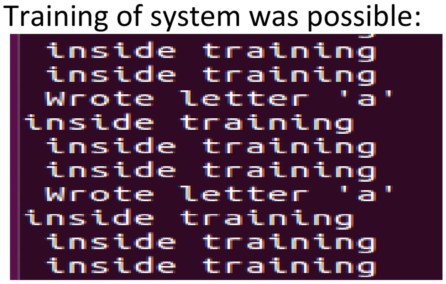
UTC_8,9:

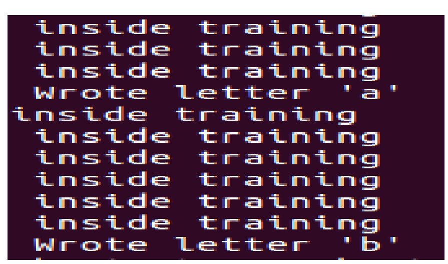
Letter identification works as expected:



UTC_10:

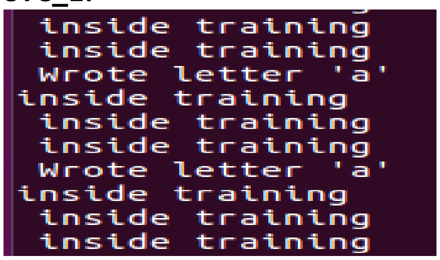
Training of system was possible:



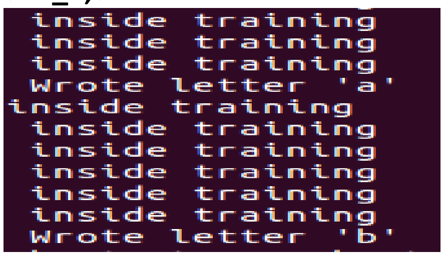


2) System Test Results:

STC_1:

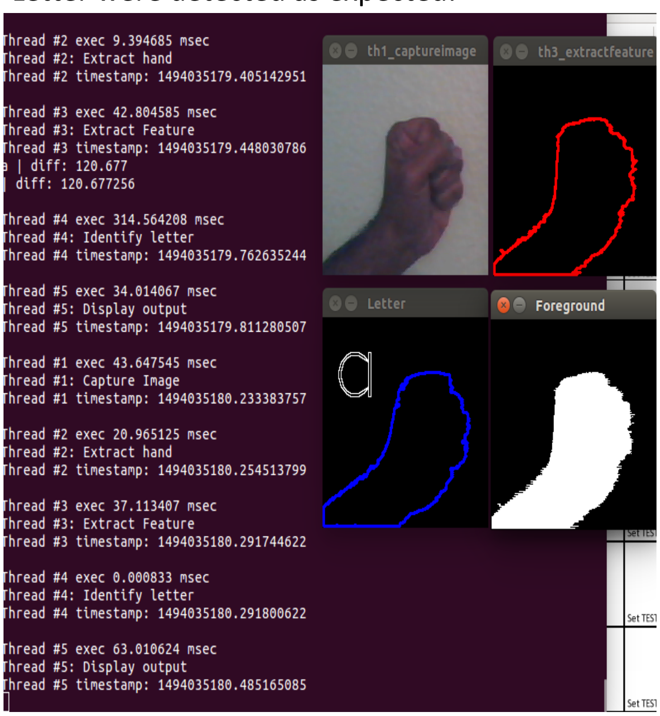


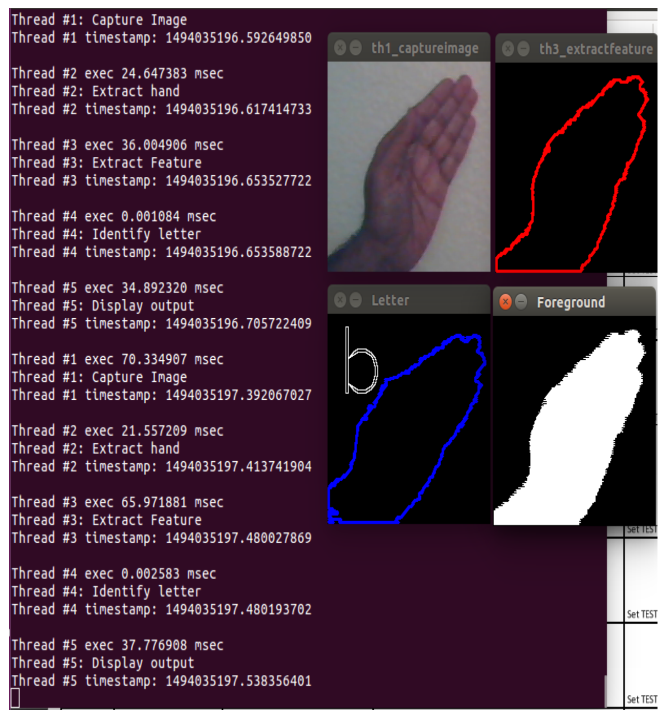
STC_2,3:



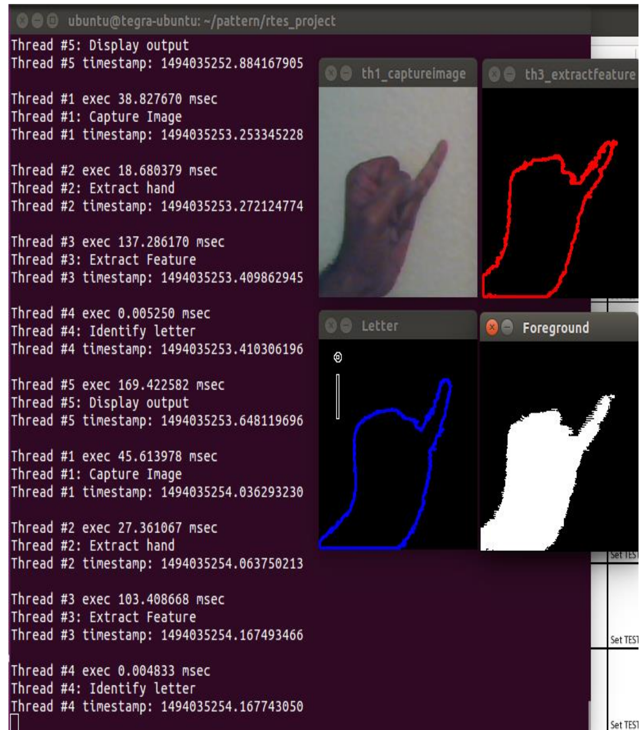
STC_4:

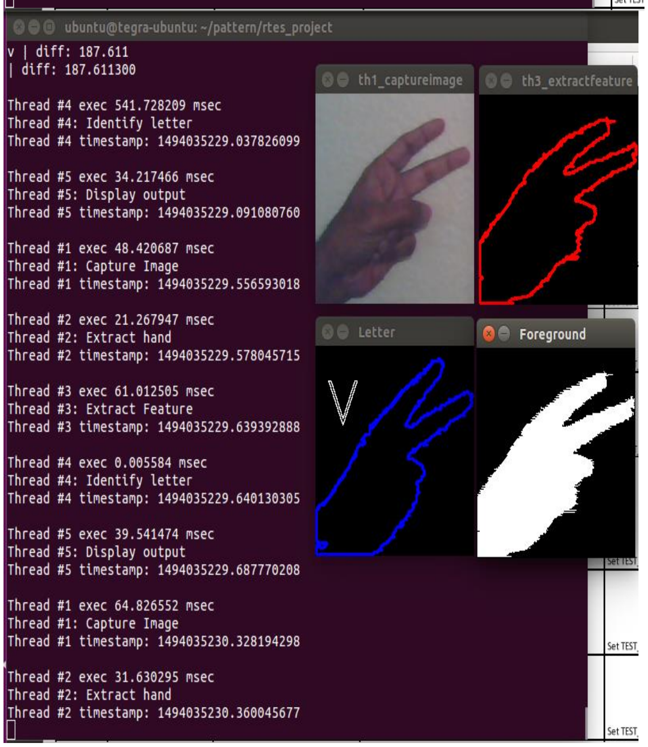
Letter were detected as expected:

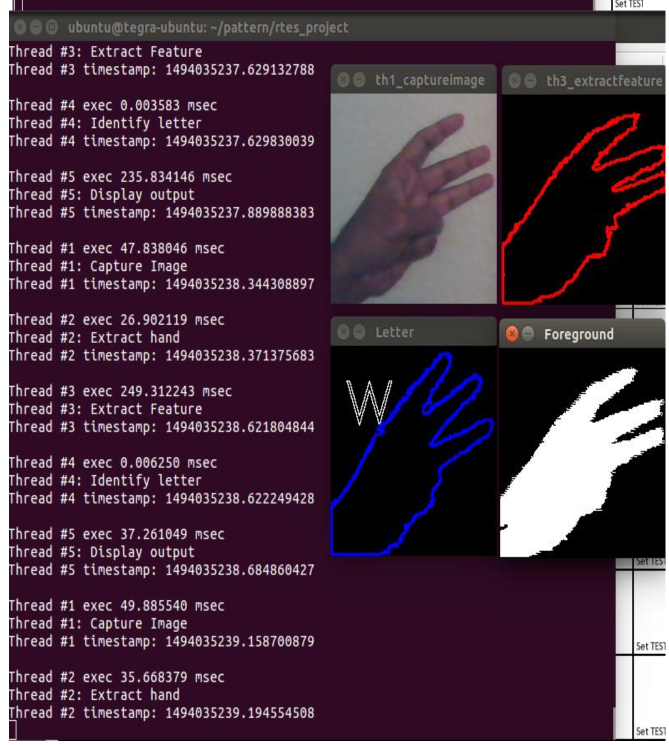





ubuntu@tegra-ubuntu: ~/pattern/rtes_project
Thread #3: Extract Feature
Thread #3 timestamp: 1494035215.332787711
Thread #4 exec 0.001500 msec
Thread #4: Identify letter
Thread #4 timestamp: 1494035215.332931795
Thread #5 exec 36.706228 msec
Thread #5: Display output
Thread #5 timestamp: 1494035215.388507054
Thread #1 exec 77.457480 msec
Thread #1: Capture Image
Thread #1 timestamp: 1494035216.144655986
Thread #2 exec 29.347215 msec
Thread #2: Extract hand
Thread #2 timestamp: 1494035216.174326535
Thread #3 exec 59.897182 msec
Thread #3: Extract Feature
Thread #3 timestamp: 1494035216.234514718
Thread #4 exec 0.003166 msec
Thread #4: Identify letter
Thread #4 timestamp: 1494035216.234724552
Thread #5 exec 74.505706 msec
Thread #5: Display output
Thread #5 timestamp: 1494035216.321290695
Thread #1 exec 59.543299 msec
Thread #1: Capture Image
Thread #1 timestamp: 1494035216.905131747
Thread #2 exec 42.920737 msec
Thread #2: Extract hand
Thread #2 timestamp: 1494035216.948230402
th1_captureimage
th3_extractfeature
Letter
C
Foreground
Set TEST
Set TEST


ubuntu@tegra-ubuntu: ~/pattern/rtes_project
Thread #5: Display output
Thread #5 timestamp: 1494035252.884167905
Thread #1 exec 38.827670 msec
Thread #1: Capture Image
Thread #1 timestamp: 1494035253.253345228
Thread #2 exec 18.680379 msec
Thread #2: Extract hand
Thread #2 timestamp: 1494035253.272124774
Thread #3 exec 137.286170 msec
Thread #3: Extract Feature
Thread #3 timestamp: 1494035253.409862945
Thread #4 exec 0.005250 msec
Thread #4: Identify letter
Thread #4 timestamp: 1494035253.410306196
Thread #5 exec 169.422582 msec
Thread #5: Display output
Thread #5 timestamp: 1494035253.648119696
Thread #1 exec 45.613978 msec
Thread #1: Capture Image
Thread #1 timestamp: 1494035254.036293230
Thread #2 exec 27.361067 msec
Thread #2: Extract hand
Thread #2 timestamp: 1494035254.063750213
Thread #3 exec 103.408668 msec
Thread #3: Extract Feature
Thread #3 timestamp: 1494035254.167493466
Thread #4 exec 0.004833 msec
Thread #4: Identify letter
Thread #4 timestamp: 1494035254.167743050
th1_captureimage
th3_extractfeature
Letter
I
Foreground


ubuntu@tegra-ubuntu: ~/pattern/rtes_project
v | diff: 187.611
| diff: 187.611300
Thread #4 exec 541.728209 msec
Thread #4: Identify letter
Thread #4 timestamp: 1494035229.037826099
Thread #5 exec 34.217466 msec
Thread #5: Display output
Thread #5 timestamp: 1494035229.091080760
Thread #1 exec 48.420687 msec
Thread #1: Capture Image
Thread #1 timestamp: 1494035229.556593018
Thread #2 exec 21.267947 msec
Thread #2: Extract hand
Thread #2 timestamp: 1494035229.578045715
Thread #3 exec 61.012505 msec
Thread #3: Extract Feature
Thread #3 timestamp: 1494035229.639392888
Thread #4 exec 0.005584 msec
Thread #4: Identify letter
Thread #4 timestamp: 1494035229.640130305
Thread #5 exec 39.541474 msec
Thread #5: Display output
Thread #5 timestamp: 1494035229.687770208
Thread #1 exec 64.826552 msec
Thread #1: Capture Image
Thread #1 timestamp: 1494035230.328194298
Thread #2 exec 31.630295 msec
Thread #2: Extract hand
Thread #2 timestamp: 1494035230.360045677
th1_captureimage
th3_extractfeature
Letter
V
Foreground
Set TEST
Set TEST


ubuntu@tegra-ubuntu: ~/pattern/rtes_project
Thread #3: Extract Feature
Thread #3 timestamp: 1494035237.629132788
Thread #4 exec 0.003583 msec
Thread #4: Identify letter
Thread #4 timestamp: 1494035237.629830039
Thread #5 exec 235.834146 msec
Thread #5: Display output
Thread #5 timestamp: 1494035237.889888383
Thread #1 exec 47.838046 msec
Thread #1: Capture Image
Thread #1 timestamp: 1494035238.344308897
Thread #2 exec 26.902119 msec
Thread #2: Extract hand
Thread #2 timestamp: 1494035238.371375683
Thread #3 exec 249.312243 msec
Thread #3: Extract Feature
Thread #3 timestamp: 1494035238.621804844
Thread #4 exec 0.006250 msec
Thread #4: Identify letter
Thread #4 timestamp: 1494035238.622249428
Thread #5 exec 37.261049 msec
Thread #5: Display output
Thread #5 timestamp: 1494035238.684860427
Thread #1 exec 49.885540 msec
Thread #1: Capture Image
Thread #1 timestamp: 1494035239.158700879
Thread #2 exec 35.668379 msec
Thread #2: Extract hand
Thread #2 timestamp: 1494035239.194554508
th1_captureimage
th3_extractfeature
Letter
W
Foreground

Appendix B: Code

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <pthread.h>
#include <sched.h>
#include <semaphore.h>
#include <syslog.h>
#include <sstream>
#include <cmath>

#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/video/background_segm.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include "opencv2/calib3d/calib3d.hpp"
#include "opencv2/legacy/legacy.hpp"
#include "opencv2/highgui/highgui.hpp"

using namespace cv;
using namespace std;

#define TEST_OFF 1
#define TEST_1_ON 2 //test thread 1
#define TEST_2_ON 3 //test thread 2
#define TEST_3_ON 4 //test thread 3
#define TEST_4_ON 5 //test thread 4
#define TEST_5_ON 6 //test thread 5
#define TEST_MODE TEST_OFF


#define HRES (320)
#define VRES (240)
#define NSEC (1000000000)
#define MSEC (1000000)
#define NSEC_PER_MICROSEC (1000)
#define NUM_CPUS (1)
#define KEY_ESC (27)

#if(TEST_MODE == TEST_OFF)
#define NUM_THREADS (5)
#else
#define NUM_THREADS (1)
#endif

#define MAX_WORDS 26                    // Number of letters
#define NUM_LAST_LETTERS 3              // Number of letters to store
#define MIN_FREQ 2                      // Minimum frequency of last letters
#define THRESH 200
#define SAMPLE_RATE 1
#define RESET_THRESH  25000000
#define DIFF_THRESH 270
```

```
// POSIX thread declarations and scheduling attributes
pthread_t threads[NUM_THREADS];
pthread_attr_t rt_sched_attr[NUM_THREADS];

pthread_attr_t main_attr;
int rt_max_prio, rt_min_prio;

struct sched_param rt_param[NUM_THREADS];
struct sched_param main_param;
pid_t mainpid;

// initialize start time to 0sec and 0 nanosec

struct timespec start_time = {0, 0};

// Thread declarations

void *th1_captureimage(void*);
void *th2_extracthand(void*);
void *th3_extractfeature(void*);
void *th4_identifyletter(void*);
void *th5_displayletter(void*);

void aslt_init();
//void doSystemCalls(char c);

// Global data

Mat rgb_image;                              // output th1--> rgb_imge
Mat binary_image;
Mat drawing;                          // output th2--> binary_image
vector<vector<Point> > feature_image;                   // output th3--> feature_image
char asl_letter;                           // output th4--> asl_letter
VideoCapture capture;
Ptr<BackgroundSubtractor> pMOG2;
vector<Point> letters[MAX_WORDS];
vector<Vec4i> hierarchy;
vector<vector<Point> > contours;
int frames = 0;
int maxIndex = 0;
int reset = 0;

sem_t     SIGNDECODED_SM,
    NEWIMAGEREADY_SM,
    HANDREADY_SM,
    FEATUREREADY_SM,
    SIGNREADY_SM;


void print_scheduler(void)
{
   int schedType;

   schedType = sched_getscheduler(getpid());

   switch(schedType)
   {
     case SCHED_FIFO:
```

```
        printf("Pthread Policy is SCHED_FIFO\n");
          break;
    case SCHED_OTHER:
          printf("Pthread Policy is SCHED_OTHER\n");
      break;
    case SCHED_RR:
          printf("Pthread Policy is SCHED_OTHER\n");
          break;
    default:
      printf("Pthread Policy is UNKNOWN\n");
      break;
  }

}

int delta_t(struct timespec *stop, struct timespec *start, struct timespec *delta_t)
{
  int dt_sec=stop->tv_sec - start->tv_sec;
  int dt_nsec=stop->tv_nsec - start->tv_nsec;

  if(dt_sec >= 0)
  {
    if(dt_nsec >= 0)
    {
      delta_t->tv_sec=dt_sec;
      delta_t->tv_nsec=dt_nsec;
    }
    else
    {
      delta_t->tv_sec=dt_sec-1;
      delta_t->tv_nsec=NSEC+dt_nsec;
    }
  }
  else
  {
    if(dt_nsec >= 0)
    {
      delta_t->tv_sec=dt_sec;
      delta_t->tv_nsec=dt_nsec;
    }
    else
    {
      delta_t->tv_sec=dt_sec-1;
      delta_t->tv_nsec=NSEC+dt_nsec;
    }
  }

  return(1);
}

int distance_2(vector<Point> a, vector<Point> b) {
    int maxDistAB = 0;
    for (size_t i = 0; i < a.size(); i++) {
        int minB = 1000000;
        for (size_t j = 0; j < b.size(); j++) {
```

```
            int dy = (a[i].y - b[j].y);
            int tmpDist = dx*dx + dy*dy;

            if (tmpDist < minB) {
                minB = tmpDist;
            }
            if (tmpDist == 0) {
                break; // can't get better than equal.
            }
        }
        maxDistAB += minB;
    }
    return maxDistAB;
}
double distance_hausdorff(vector<Point> a, vector<Point> b) {
    int maxDistAB = distance_2(a, b);
    int maxDistBA = distance_2(b, a);
    int maxDist = max(maxDistAB,maxDistBA);

    return sqrt((double)maxDist);
}

void aslt_init(void)
{
    int numframe = 0;
    Mat frame;
    sem_init(&SIGNDECODED_SM, 0, 1);
    sem_init(&NEWIMAGEREADY_SM, 0, 0);
    sem_init(&HANDREADY_SM, 0, 0);
    sem_init(&FEATUREREADY_SM, 0, 0);
    sem_init(&SIGNREADY_SM, 0, 0);

    //************Preload letter images starts*********//
    for (int i = 0; i < MAX_WORDS; i++) {
        char buf[13 * sizeof(char)];
        sprintf(buf, "images/%c.png", (char)('a' + i));
        Mat im = imread(buf, 1);
        if (im.data) {
            Mat bwim;
            cvtColor(im, bwim, CV_RGB2GRAY);
            Mat threshold_output;
                        // Detect edges using Threshold
            threshold( bwim, threshold_output, THRESH, 255, THRESH_BINARY );
            findContours(threshold_output, contours, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, Point(0, 0) );
            letters[i] = contours[0];
        }
    }
    //************Preload letter images ends*********//


    //************learn starts*********************//

    pMOG2 = new BackgroundSubtractorMOG2();
    //************learn ends  *********************//

}
```

```
void flush(VideoCapture& camera)
{
    int delay = 0;
    //QElapsedTimer timer;

    int framesWithDelayCount = 0;

    while (framesWithDelayCount <= 4)
    {
        camera.grab();
        framesWithDelayCount++;
    }
}

void flush1(VideoCapture& camera)
{
    int delay = 0;
    int framesWithDelayCount = 0;

    while (framesWithDelayCount <= 1)
    {
        camera.grab();
        framesWithDelayCount++;
    }
}

void *th1_captureimage(void*)
{
    Mat frame;

    capture = VideoCapture(0);
    capture.set(CV_CAP_PROP_BUFFERSIZE, 3);


    while(1)
    {
        printf("Thread #1: Capture Image\n\r");
    struct timespec timestamp_th1 = {0, 0};
            struct timespec finish_time_th1 = {0, 0};
            struct timespec thread_dt_th1 = {0, 0};
    struct timespec start_time_th1 = {0, 0};
    clock_gettime(CLOCK_REALTIME, &timestamp_th1);
    printf("Thread #1 timestamp: %lld.%.9ld \n\r", (long long)timestamp_th1.tv_sec, timestamp_th1.tv_nsec);


        sem_wait(&SIGNDECODED_SM);                    // SIGNDECODED_SM take

    clock_gettime(CLOCK_REALTIME, &start_time_th1);

                // Create the capture object
        if (!capture.isOpened()) {
            cerr << "Cannot Open Webcam !!!" << endl;        // Error in opening the video input
            exit(EXIT_FAILURE);
        }
    flush(capture);
                                    // Read the current frame
```

```
        if (!capture.read(frame)) {
            cerr << "Unable to read next frame." << endl;
            cerr << "Exiting..." << endl;
            exit(EXIT_FAILURE);
        }
        cv::Rect myROI(50, 150, 200, 200);              // Crop Frame to smaller region : output --> rgb_image
        rgb_image = frame(myROI);
        imshow("th1_captureimage", rgb_image);              // output th1--> rgb_imge
        char q = cvWaitKey(33);

        clock_gettime(CLOCK_REALTIME, &finish_time_th1);
        delta_t(&finish_time_th1, &start_time_th1, &thread_dt_th1);    //compute the time of thread execution from the start and
end times

        printf("\nThread #1 exec %lf msec \n\r", (double)((double)thread_dt_th1.tv_nsec / (MSEC*10)));

        sem_post(&NEWIMAGEREADY_SM);                   // NEWIMAGEREADY_SM give

    }
    capture.release();

}

void *th2_extracthand(void*)
{
    while(1)
    {
        printf("Thread #2: Extract hand\n\r");
        struct timespec start_time_th2 = {0, 0};
            struct timespec finish_time_th2 = {0, 0};
            struct timespec thread_dt_th2 = {0, 0};
    struct timespec timestamp_th2 = {0, 0};
    clock_gettime(CLOCK_REALTIME, &timestamp_th2);
    printf("Thread #2 timestamp: %lld.%.9ld \n\r", (long long)timestamp_th2.tv_sec, timestamp_th2.tv_nsec);

        sem_wait(&NEWIMAGEREADY_SM);                   // NEWIMAGEREADY_SM take
        clock_gettime(CLOCK_REALTIME, &start_time_th2);

        if(reset <= 10){
        reset++;
        pMOG2 = new BackgroundSubtractorMOG2();
        }


        pMOG2->operator()(rgb_image, binary_image);
        //imshow("raw", binary_image);

        clock_gettime(CLOCK_REALTIME, &finish_time_th2);
        delta_t(&finish_time_th2, &start_time_th2, &thread_dt_th2);//compute the time of thread execution from the start and
end times

        printf("\nThread #2 exec %lf msec \n\r", (double)((double)thread_dt_th2.tv_nsec / MSEC));
        sem_post(&HANDREADY_SM);                   // HANDREADY_SM give
    }
```

```
}
void *th3_extractfeature(void*)
{
   while(1)
   {

      Mat threshold_output;                        // Generate Convex Hull
         printf("Thread #3: Extract Feature\n\r");
      struct timespec start_time_th3 = {0, 0};
         struct timespec finish_time_th3 = {0, 0};
         struct timespec thread_dt_th3 = {0, 0};
   struct timespec timestamp_th3 = {0, 0};
   clock_gettime(CLOCK_REALTIME, &timestamp_th3);
   printf("Thread #3 timestamp: %lld.%.9ld \n\r", (long long)timestamp_th3.tv_sec, timestamp_th3.tv_nsec);

      sem_wait(&HANDREADY_SM);                 // HANDREADY_SM take
      clock_gettime(CLOCK_REALTIME, &start_time_th3);

      threshold( binary_image, threshold_output, THRESH, 255, THRESH_BINARY );          // Detect edges using
Threshold

      findContours( threshold_output, feature_image, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, Point(0,
0) );    // Find contours
      //imshow("feature" , feature_image);
      drawing = Mat::zeros(rgb_image.size(), CV_8UC3);                        // Find largest contour
      double largest_area = 0;
      for (int j = 0; j < feature_image.size(); j++)
      {
         double area = contourArea(feature_image[j], false);    // Find the area of contour
         if (area > largest_area) {
            largest_area = area;
            maxIndex = j;                    // Store the index of largest contour
         }
      }

      //printf("%d", maxIndex);                        // Draw Largest Contours
      Scalar color = Scalar(0, 0, 255);
      drawContours(drawing, feature_image, maxIndex, Scalar(255, 255, 255), CV_FILLED); // fill white

                              // Draw Contours
      Mat contourImg = Mat::zeros(rgb_image.size(), CV_8UC3);
      drawContours( contourImg, feature_image, maxIndex, Scalar(0, 0, 255), 2, 8, hierarchy, 0, Point(0, 0) );

      // Reset if too much noise
      Scalar sums = sum(drawing);
       int s = sums[0] + sums[1] + sums[2] + sums[3];
      if (s >= RESET_THRESH) {
        reset = 10;
      }

      imshow("Foreground", drawing);
      if (contourImg.rows > 0)
         imshow("th3_extractfeature", contourImg);
      char q = cvWaitKey(33);

      clock_gettime(CLOCK_REALTIME, &finish_time_th3);
```

```
        delta_t(&finish_time_th3, &start_time_th3, &thread_dt_th3);    // compute the time of thread execution from the start and
end times

        printf("\nThread #3 exec %lf msec \n\r", (double)((double)thread_dt_th3.tv_nsec / MSEC));
        sem_post(&FEATUREREADY_SM);                    // FEATUREREADY_SM give
    }

}
void *th4_identifyletter(void*)
{
    while(1)
    {
        printf("Thread #4: Identify letter\n\r");
        struct timespec start_time_th4 = {0, 0};
            struct timespec finish_time_th4 = {0, 0};
            struct timespec thread_dt_th4 = {0, 0};
    struct timespec timestamp_th4 = {0, 0};
    clock_gettime(CLOCK_REALTIME, &timestamp_th4);
    printf("Thread #4 timestamp: %lld.%.9ld \n\r", (long long)timestamp_th4.tv_sec, timestamp_th4.tv_nsec);


        sem_wait(&FEATUREREADY_SM);                    // FEATUREREADY_SM take
        clock_gettime(CLOCK_REALTIME, &start_time_th4);

        //************

        // Compare to reference images
        if (feature_image.size() > 0 && frames++ > SAMPLE_RATE && feature_image[maxIndex].size() >= 5) {
            RotatedRect testRect = fitEllipse(feature_image[maxIndex]);
            frames = 0;
            double lowestDiff = HUGE_VAL;
            for (int i = 0; i < MAX_WORDS; i++) {
                if (letters[i].size() == 0) continue;

                double diff = distance_hausdorff(letters[i], feature_image[maxIndex]);

                if (diff < lowestDiff) {
                    lowestDiff = diff;
                    asl_letter = 'a' + i;
                }
            }
            if (lowestDiff > DIFF_THRESH) { // Dust
                asl_letter = 0;
            }
            cout << asl_letter << " | diff: " << lowestDiff << endl;
            printf("| diff: %f \n\r",lowestDiff );
        }
        //************

        clock_gettime(CLOCK_REALTIME, &finish_time_th4);
        delta_t(&finish_time_th4, &start_time_th4, &thread_dt_th4);    // compute the time of thread execution from the start and
end times

        printf("\nThread #4 exec %lf msec \n\r", (double)((double)thread_dt_th4.tv_nsec / MSEC));
        sem_post(&SIGNREADY_SM);                    // SIGNREADY_SM give
```

```
}
void *th5_displayletter(void*)
{
    int letterCount = 0;                    // number of letters captured since last display
    char lastLetters[NUM_LAST_LETTERS] = {0};
    Mat letter_image = Mat::zeros(200, 200, CV_8UC3);
    char lastExecLetter = 0;                // last letter sent to doSystemCalls()



    while(1)
    {
        printf("Thread #5: Display output\n\r");
        struct timespec start_time_th5 = {0, 0};
            struct timespec finish_time_th5 = {0, 0};
            struct timespec thread_dt_th5 = {0, 0};
    struct timespec timestamp_th5 = {0, 0};
    clock_gettime(CLOCK_REALTIME, &timestamp_th5);
    printf("Thread #5 timestamp: %lld.%.9ld \n\r", (long long)timestamp_th5.tv_sec, timestamp_th5.tv_nsec);

        sem_wait(&SIGNREADY_SM);            // SIGNREADY_SM take
        clock_gettime(CLOCK_REALTIME, &start_time_th5);

        letterCount %= NUM_LAST_LETTERS;            // Show majority of last letters captured
        lastLetters[letterCount++] = asl_letter;    // input from th4
        letter_image = Mat::zeros(200, 200, CV_8UC3);

        int counts[MAX_WORDS+1] = {0};
        for (int i = 0; i < NUM_LAST_LETTERS; i++)
            counts[lastLetters[i] + 1 - 'a']++;

        int maxCount = 0;
        char maxChar = 0;
        for (int i = 0; i < MAX_WORDS+1; i++) {
            if (counts[i] > maxCount) {
                maxCount = counts[i];
                maxChar = i;
            }
        }

        if (maxChar && maxCount >= MIN_FREQ)
        {
            maxChar = maxChar - 1 + 'a';
        char buf[2 * sizeof(char)];
            sprintf(buf, "%c", maxChar);
        putText(letter_image, buf, Point(10, 75), CV_FONT_NORMAL, 3, Scalar(255, 255, 255), 1, 1);
        vector<vector<Point> > dummy;
        dummy.push_back(letters[maxChar-'a']);
    drawContours( letter_image, dummy, 0, Scalar(255, 0, 0), 2, 8, hierarchy, 0, Point(0, 0) );

        }
            if (maxChar != lastExecLetter) {
                lastExecLetter = maxChar;
                //doSystemCalls(maxChar);
            }
```

```
        imshow("Letter", letter_image);                    // output th5--> letter_image
        char q = cvWaitKey(33);

        clock_gettime(CLOCK_REALTIME, &finish_time_th5);
        delta_t(&finish_time_th5, &start_time_th5, &thread_dt_th5);    //compute the time of thread execution from the start and
end times

        printf("\nThread #5 exec %lf msec \n\r", (double)((double)thread_dt_th5.tv_nsec / MSEC));
        sem_post(&SIGNDECODED_SM);                    // SIGNDECODED_SM give
    }

}

int main( int argc, char** argv )
{
    int rc,scope,i;
    cpu_set_t cpuset;
    CPU_ZERO(&cpuset);                            //setting the CPU cores of all cores to 0.
    for(i=0; i < NUM_CPUS; i++)
    CPU_SET(i, &cpuset);
    char keyboard = 0;    // last key pressed
    int training_mode = 0; // 0 = no training; 1 = training
    printf("Do you want to train the system? \n\r");
          cin >> keyboard;
          if(keyboard == 'y')
        {
        training_mode = 1;
        }


    if(training_mode)
    {
    capture = VideoCapture(0);
    capture.set(CV_CAP_PROP_BUFFERSIZE, 3);
    pMOG2 = new BackgroundSubtractorMOG2();
    while (keyboard != KEY_ESC) {
        printf("inside training \n\r ");
        if (!capture.isOpened()) {
        // Error in opening the video input
        cerr << "Cannot Open Webcam... " << endl;
        exit(EXIT_FAILURE);
        }

            Mat frame;          // current frame
            Mat fgMaskMOG2;     // fg mask fg mask generated by MOG2 method
        flush1(capture);
        if (!capture.read(frame)) {
            cerr << "Unable to read next frame." << endl;
            cerr << "Exiting..." << endl;
            exit(EXIT_FAILURE);
        }
```

```
// Crop Frame to smaller region
    cv::Rect myROI(50, 150, 200, 200);
    Mat cropFrame = frame(myROI);

    // Update the background model
    pMOG2->operator()(cropFrame, fgMaskMOG2);

    // Generate Convex Hull
    Mat threshold_output;
    vector<vector<Point> > contours;

    // Detect edges using Threshold
    threshold(fgMaskMOG2 , threshold_output, THRESH, 255, THRESH_BINARY );

    // Find contours
    findContours( threshold_output, contours, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, Point(0, 0) );

    // Find largest contour
    Mat drawing1 = Mat::zeros(cropFrame.size(), CV_8UC3);
    double largest_area = 0;
    int maxIndex = 0;
    for (int j = 0; j < contours.size(); j++) {
      double area = contourArea(contours[j], false);    // Find the area of contour
      if (area > largest_area) {
        largest_area = area;
        maxIndex = j;  // Store the index of largest contour
      }
    }

  // Draw Largest Contours
    Scalar color = Scalar(0, 0, 255);
    drawContours(drawing1, contours, maxIndex, Scalar(255, 255, 255), CV_FILLED); // fill white
    // Draw Contours
    Mat contourImg = Mat::zeros(cropFrame.size(), CV_8UC3);
    drawContours( contourImg, contours, maxIndex, Scalar(0, 0, 255), 2, 8, hierarchy, 0, Point(0, 0) );

    // Reset if too much noise
    Scalar sums = sum(drawing1);
    int s = sums[0] + sums[1] + sums[2] + sums[3];
    if (s >= RESET_THRESH) {
      pMOG2 = new BackgroundSubtractorMOG2();
      continue;
    }

    // Show the current frame and the fg masks
    imshow("Crop Frame", cropFrame);
    imshow("Foreground", drawing1);
    if (contourImg.rows > 0)
      imshow("Contour", contourImg);

    keyboard = waitKey(10);

    if (keyboard >= 'a' && keyboard <= 'z') {
      cout << "Wrote letter '" << (char)keyboard << '\'' << endl;

      // save in memory
```

```
            letters[keyboard - 'a'] = contours[maxIndex];

            // write to file
            char buf[13 * sizeof(char)];
            sprintf(buf, "images/%c.png", (char)keyboard);
            imwrite(buf, drawing1);
        }

        // Manual reset
        if (keyboard == ' ')
            pMOG2 = new BackgroundSubtractorMOG2();

    // Delete capture object

    }
    destroyAllWindows();
    capture.release();
}


    mainpid=getpid();                    //get the thread id of the calling thread.

    rt_max_prio = sched_get_priority_max(SCHED_FIFO);          //max priority of the SCHED_FIFO
    rt_min_prio = sched_get_priority_min(SCHED_FIFO);          //min priority of the SCHED_FIFO

    printf("\nBefore Adjustments to Schedule Policy:");
    print_scheduler();                    //print scheduler before assigning SCHED_FIFO

    rc=sched_getparam(mainpid, &main_param);              //get the scheduling parameters of the thread and transferring it to
main_param
    main_param.sched_priority=rt_max_prio;                 //setting the max priority of the calling thread to 99

    rc=sched_setscheduler(getpid(), SCHED_FIFO, &main_param);       //set the SCHED_FIFO scheduler of the
main_param.
    if(rc < 0) perror("main_param");

    printf("\nAfter Adjustments to Schedule Policy:");
    print_scheduler();                      //print scheduler after assigning SCHED_FIFO

    pthread_attr_getscope(&main_attr, &scope);              //obtain the scope of the main_attr and print it
    if(scope == PTHREAD_SCOPE_SYSTEM)
        printf("PTHREAD SCOPE SYSTEM\n");
    else if (scope == PTHREAD_SCOPE_PROCESS)
         printf("PTHREAD SCOPE PROCESS\n");
    else
         printf("PTHREAD SCOPE UNKNOWN\n");


                            //Attribute settings for the Threads
       for(i=0; i <NUM_THREADS; i++)
       {
        rc=pthread_attr_init(&rt_sched_attr[i]);                  // intializing the pthread attributes for the five threads.
        rc=pthread_attr_setinheritsched(&rt_sched_attr[i], PTHREAD_EXPLICIT_SCHED);    // set to explicit schedule policy
and later to SCHED_FIFO
        rc=pthread_attr_setschedpolicy(&rt_sched_attr[i], SCHED_FIFO);          //set the schedule policy of the five threads to
SCHED_FIFO
```

```
        rc=pthread_attr_setaffinity_np(&rt_sched_attr[i], sizeof(cpu_set_t), &cpuset);    //set the affinity of the CPU cores to
zero

        rt_param[i].sched_priority=rt_max_prio-i;                    //set the priorities of the hreads as 98,97,96,95 and 94
respectively.
        pthread_attr_setschedparam(&rt_sched_attr[i], &rt_param[i]);            //set the scheduling parameters of the five
pthreads.
       }

    printf("rt_max_prio=%d\n", rt_max_prio);
    printf("rt_min_prio=%d\n", rt_min_prio);

    aslt_init();

    printf("threads spawning\r\n");

#if((TEST_MODE == TEST_OFF) || (TEST_MODE == TEST_1_ON))
    pthread_create(&threads[0],                   // pointer to thread descriptor
                    &rt_sched_attr[0],       // use set attributes
                    th1_captureimage,      // thread function entry point
                    (void *)(NULL)         // parameters to pass in
                    );
#endif

#if((TEST_MODE == TEST_OFF) || (TEST_MODE == TEST_2_ON))
   pthread_create(&threads[1],
                    &rt_sched_attr[1],
                    th2_extracthand,
                    (void *)(NULL)
                    );
#endif

#if((TEST_MODE == TEST_OFF) || (TEST_MODE == TEST_3_ON))
    pthread_create(&threads[2],
                    &rt_sched_attr[2],
                    th3_extractfeature,
                    (void *)(NULL)
                    );
#endif

#if((TEST_MODE == TEST_OFF) || (TEST_MODE == TEST_4_ON))
    pthread_create(&threads[3],
                    &rt_sched_attr[3],
                    th4_identifyletter,
                    (void *)(NULL)
                    );
#endif

#if((TEST_MODE == TEST_OFF) || (TEST_MODE == TEST_5_ON))
    pthread_create(&threads[4],
                    &rt_sched_attr[4],
                    th5_displayletter,
                    (void *)(NULL)
                    );
#endif
```

```
    for(i=0; i < NUM_THREADS; i++)
       pthread_join(threads[i], NULL);            //join the pthread wiith the existing processes

    pthread_exit(NULL);
    capture.release();                    // Delete capture object

}
```

Appendix C: Group Members

Vijoy Sunil Kumar
Graduate Student [ECEE]
University of Colorado, Boulder
Visu3975@colorado.edu
+1-720-755-7273

Vikhyat Goyal
Graduate Student [ECEE]
University of Colorado, Boulder
vikhyat.goyal@colorado.edu
+1-720-519-8976

Akshit Shah
Graduate Student [ECEE]
University of Colorado, Boulder
aksh9247@colorado.edu
+1-303-217-6650