Real time implementation of multispectral fusion

Long wave Infrared(LWIR) and Visible image fusion

Report for

Real time Embedded Systems (ECEN 5623)

By

Surjith Bhagavath Singh

Table of Contents

Introduction

The primary focus of this project is the fusion of multispectral images which was acquired by two distinctly different cameras (Different Specifications i.e., resolution, Zoom, Grayscale or color). This project was inspired from the SDMSI (Software defined Multi Spectral Imaging for Artic Sensor Networks)[1] paper by Prof. Sam Siewert et al. A real time fusion system has been designed to meet its deadline.

The system was realized in NVIDIA Jetson TK1 embedded Development Kit. Target system is developed to detect unmanned aerial vehicles/flights, threats in coastal borders. During the absence of light, and when the potential threat is camouflaged with the environment; it is very hard to find the threats using Visible cameras. LWIR (Long wave infrared cameras) which is of 8-15 µm wavelength can detect the heat waves emitted by the threat. This character of LWIR can be used with Visible image captured from Visible camera to identify potential threat.

Problem Statement:

A fusion algorithm has to be developed to house two different cameras (different spectrum), which are mounted closely to achieve same POV (Point of View). Multispectral Image fusion is a tough task to implement in real time because of the following reasons

Camera Issues

1. Camera parameters are not the same
2. POV(Point of View) of cameras are different
3. Zoom level and Resolutions are different
4. Visible -RGB , LWIR – Grayscale

Real time Issues

1. Compute intensive algorithm
2. Parameterizing the deadlines takes multiple analysis

Functional Requirements (Capabilities)

Initial idea was to implement the project using an LWIR Camera and a Visible camera. Since the LWIR Camera is so expensive to afford, I decided to use the datasets that are available online[2] (LITIV (Laboratoire d'Interprétation et de Traitement d'Images et Vidéo)). This dataset has been explained in later sections.

- Interface Two cameras (Proof of Concept – Code submitted seperately)

[1] http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=2524381
[2] http://www.polymtl.ca/litiv/en/vid/

- Get the two frames (LWIR and Visible) separately from the dataset
- Parameterize the constants to remap the images to match a POV.
- Fuse the images
- Compile them into a video with timestamp to verify the deadlines

This algorithm will be implemented in Jetson TK1 Development kit for embedded real-time analysis purpose. Dual camera is supported as a proof of concept (Code has been developed). Time taken for fusion will be computed and an appropriate deadline with a margin of 25% will be set for error margin. A reliable system design to perform the requirements will be designed at the end of the course.

Real time Requirements

The initial requirement was to reach deadline with camera frame grabbing of two cameras and processing the image on or before one second. Since the camera is expensive, proof of concept model for fusion has been implemented using the thermal/visible dataset. Real time requirement was to verify the system is processing the image at 1 second timeframe. There are 300 images in each dataset. Each data will be taken at 1 second interval and encoded at 30 fps to give the video of 10 seconds. The computation time will be proved by taking screenshots and the timestamp in videos.

Functional Design Overview:

The functional hardware setup for this project is shown in the diagram below,

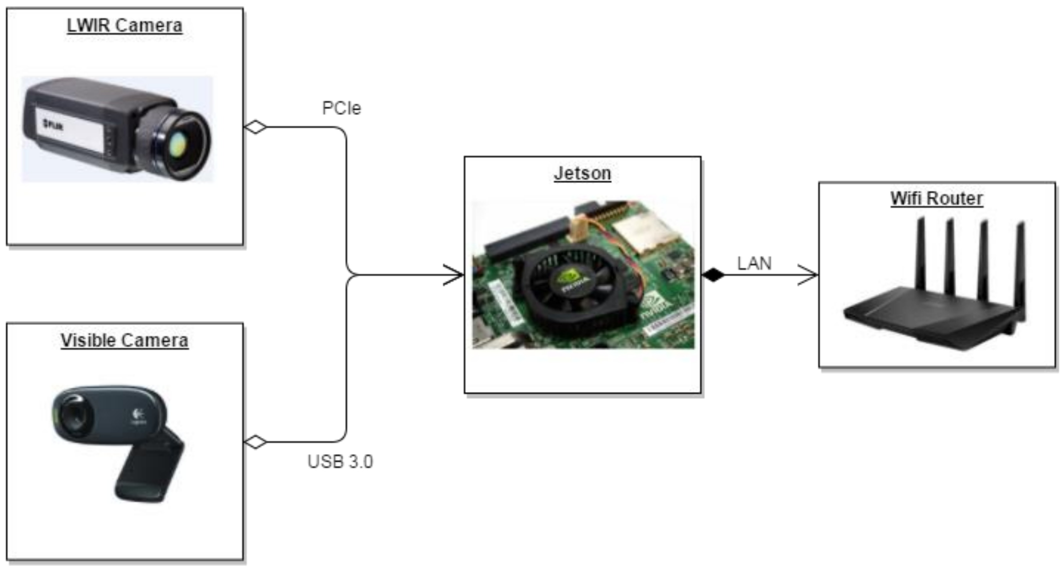


Figure 1: Hardware Setup

This hardware setup is needed for functionality of the algorithm. LWIR camera comes through analog decoder and connected to the Jetson via PCIe Socket. Visible camera is directly connected to the USB 3.0 port in Jetson. Jetson is connected to the WiFi Router , so it can be accessed using SSH.

The functionality of the system is to share the resources using mutex locks in pthreads.

Shared resources for this system is mentioned below,

1. CPU Resource
2. Memory access to write the mat object into a video

For a reliable rate monotonic system, a single cpu core is used for a pthread (By setting affinity for the thread).

Cameras have to acquire the images independently, this will be processed by separate core.

A thread's CPU affinity mask determines the set of CPUs on which it is eligible to run. On a multiprocessor system like jetson, setting the CPU affinity mask is used to obtain performance benefits. For example, by dedicating one CPU to a particular thread (i.e., setting the affinity mask of that thread to specify a single CPU, and setting the affinity mask of all other threads to exclude that CPU), it is possible to ensure maximum execution speed for that thread. Restricting a thread to run on a single CPU also avoids the performance cost caused by the cache invalidation that occurs when a thread ceases to execute on one CPU and then recommences execution on a different CPU.[3]

The software flowchart of the system has been plotted below. There are three main thread functions, Visible image read, LWIR image read and resize and Fusion(time stamps).

The main algorithm is to take two images from two folders and fuse them together and time stamp it and store it back as an image for every 1 seconds. The tricky part is how well the deadline is met for each service and what is the jitter involved with it and analyze it using cheddar. Analyze the real time specification of the task.

LWIR ReSizing

LWIR Image is stored in the folder Thermal and the path of the file is included in the code. This thread reads the image and resizes the image,

How is it resizing the image? This data set has taken two frames consecutively using LWIR camera and Visible Camera. Since both the camera has different parameters(Zoom level and point of view). In order to match both the images common area, it has to be resized. Parameter Calculation has been Explained below.

These parameters are calculated Using GIMP Tool. A screenshot of this process has been attached below. For this example, The matched zoom level is calculated as (1/0.6). The calculation sample is attached below. Measure the matched Images pixel using measure tool in GIMP. For this dataset it is found to be 192x168 Pixels.

Original resolution of the image is 320x280. calculating the scaling factor,

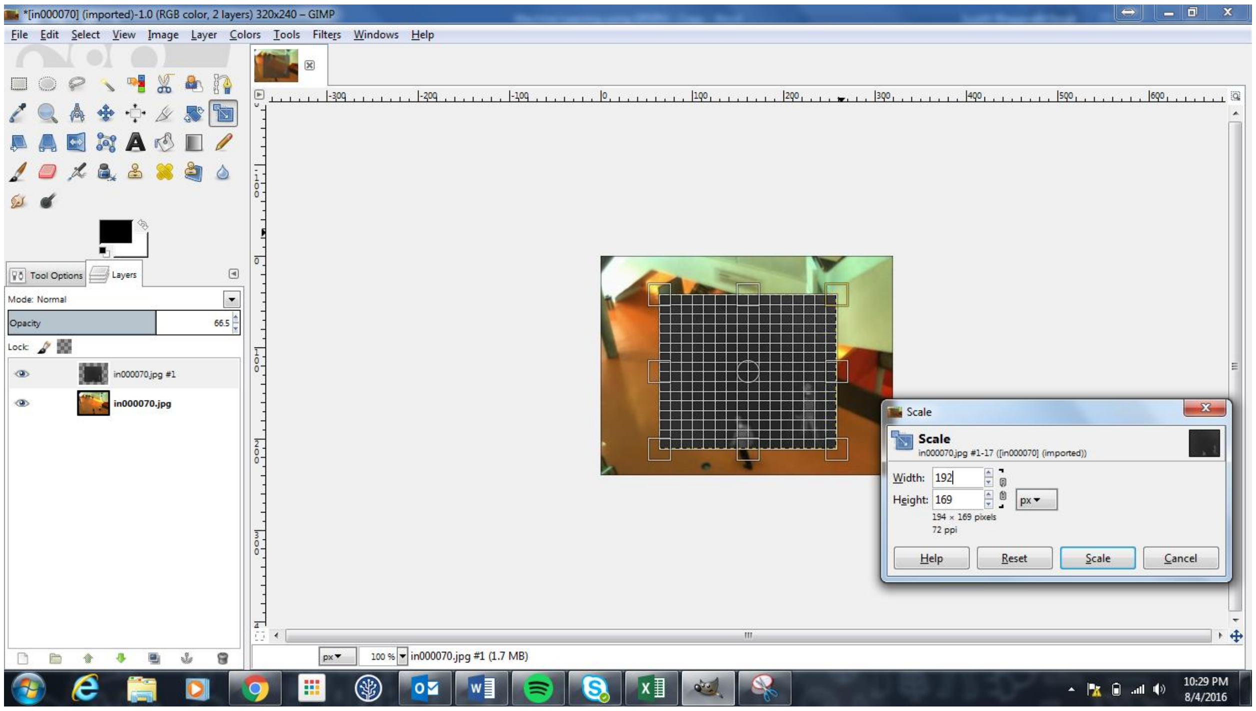
$$Zoom_X = \frac{Original\ Image\ Width}{Scaled\ Image\ width} = \frac{320}{192}$$

$$Zoom_Y = \frac{Original\ Image\ Height}{Scaled\ Image\ Height} = \frac{280}{168}$$

[3] http://man7.org/linux/man-pages/man2/sched_setaffinity.2.html

Adjusting factors Dx and Dy can be calculated from GIMP tool, this is the number of pixels the scaled image has to move in x and y direction. For this dataset it appears to be Dx=62 and Dy=125 Pixels.

For Every Camera pair this will be a constant. Once the cameras are mounted on a fixed mount this parameter doesn't change.


*[in000070] (imported)-1.0 (RGB color, 2 layers) 320x240 – GIMP

Mode: Normal

Opacity 66.5

Scale

in000070.jpg #1-17 ([in000070] (imported))

Width: 192

Height: 169 px

194 × 169 pixels

72 ppi

in000070.jpg #1 (1.7 MB)

Figure 2: GIMP Tool Screenshot

This parameter will be fed into the remapping function by openCV. Then this remapped image is fused with Visible image and written back into the file.

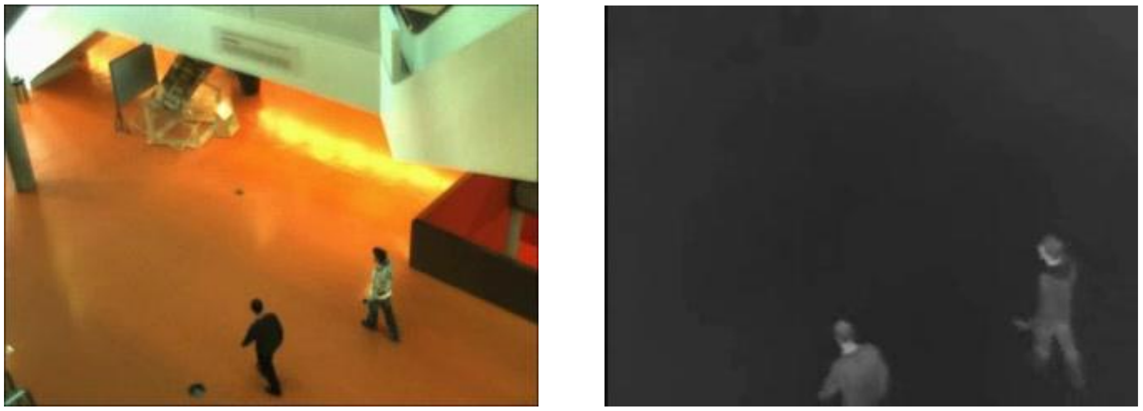


Figure 3: Visible and LWIR Images

Fusion[4]

In computer vision, Multisensor Image fusion is the process of combining relevant information from two or more images into a single image. The resulting image will be more informative than any of the input images.

In remote sensing applications,the increasing availability of space borne sensors gives a motivation for different image fusion algorithms. Several situations in image processing require high spatial and high spectral resolution in a single image. Most of the available equipment is not capable of providing such data convincingly. Image fusion techniques allow the integration of different information sources. The fused image can have complementary spatial and spectral resolution characteristics. However, the standard image fusion techniques can distort the spectral information of the multispectral data while merging.

In satellite imaging, two types of images are available. The panchromatic image acquired by satellites is transmitted with the maximum resolution available and the multispectral data are transmitted with coarser resolution. This will usually be two or four times lower. At the receiver station, the panchromatic image is merged with the multispectral data to convey more information.

Many methods exist to perform image fusion. The very basic one is the high pass filtering technique. Later techniques are based on Discrete Wavelet Transform, uniform rational filter bank, and Laplacian pyramid. In this system Averaging fusion technique has been used.

This fused image is then time stamped by getting the raw system wide clock and printing it in the image. This loops through 300 frames in each thermal and visible directory. The software stateflow has been attached below.

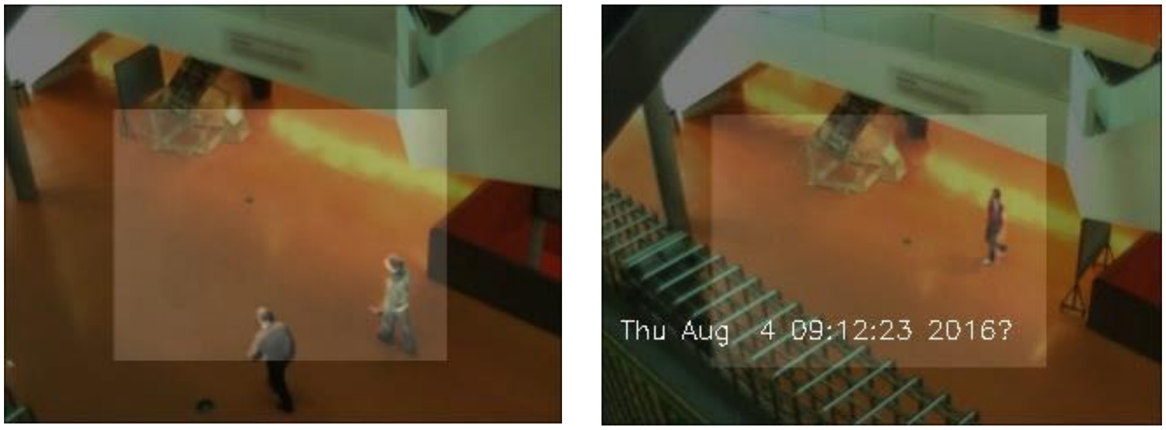


Figure 4: Fused Images before and after time stamping

Software Real-time Design Overview

The software flow has a main function, which creates one main thread and joins it.

- Main function spawns mytransform thread, this thread has the highest priority of all.

[4] https://en.wikipedia.org/wiki/Image_fusion

- Mytransform thread spawns three other threads(LWIRthread,Visiblethread,Fusionthread) with affinity set for each thread creation. This function loops through frame numbers.
- LWIR Thread reads the LWIR image from the directory and remaps it according to the parameters that we discussed in the functional requirements – This is the longest process in the system(Remap compute intensive).
- Visible Thread reads visible camera image from the directory and stores it as a Mat object. – Fastest process in the system
- Fusion Thread fuses both IR and Visible Mat objects using averaging fusion and timestamps it using puttext function by openCV.
- Timed Unlock is the important function of the entire system, this handles how to unlock the lock according to the deadlines, this computes the time remaining to meet the deadline and sets the system to sleep for that time. This has been well explained Fig : Timed Unlock

The output snapshots and video has been attached with the submissions.

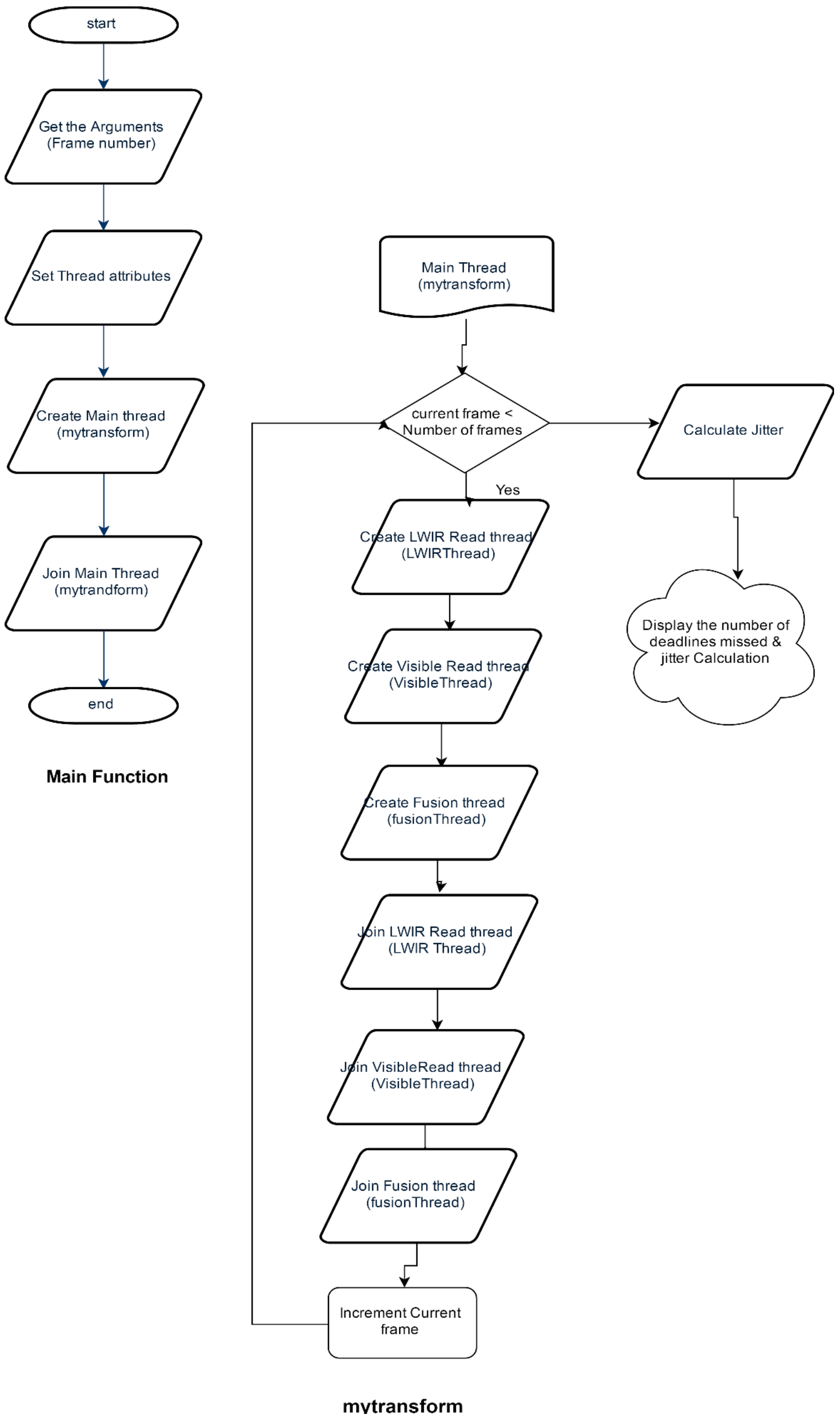

start
Get the Arguments (Frame number)
Set Thread attributes
Create Main thread (mytransform)
Join Main Thread (mytrandform)
end
Main Function
Main Thread (mytransform)
current frame < Number of frames
Yes
Calculate Jitter
Display the number of deadlines missed & jitter Calculation
Create LWIR Read thread (LWIRThread)
Create Visible Read thread (VisibleThread)
Create Fusion thread (fusionThread)
Join LWIR Read thread (LWIR Thread)
Join VisibleRead thread (VisibleThread)
Join Fusion thread (fusionThread)
Increment Current frame
mytransform

Figure 5: Main Function and mytransform thread

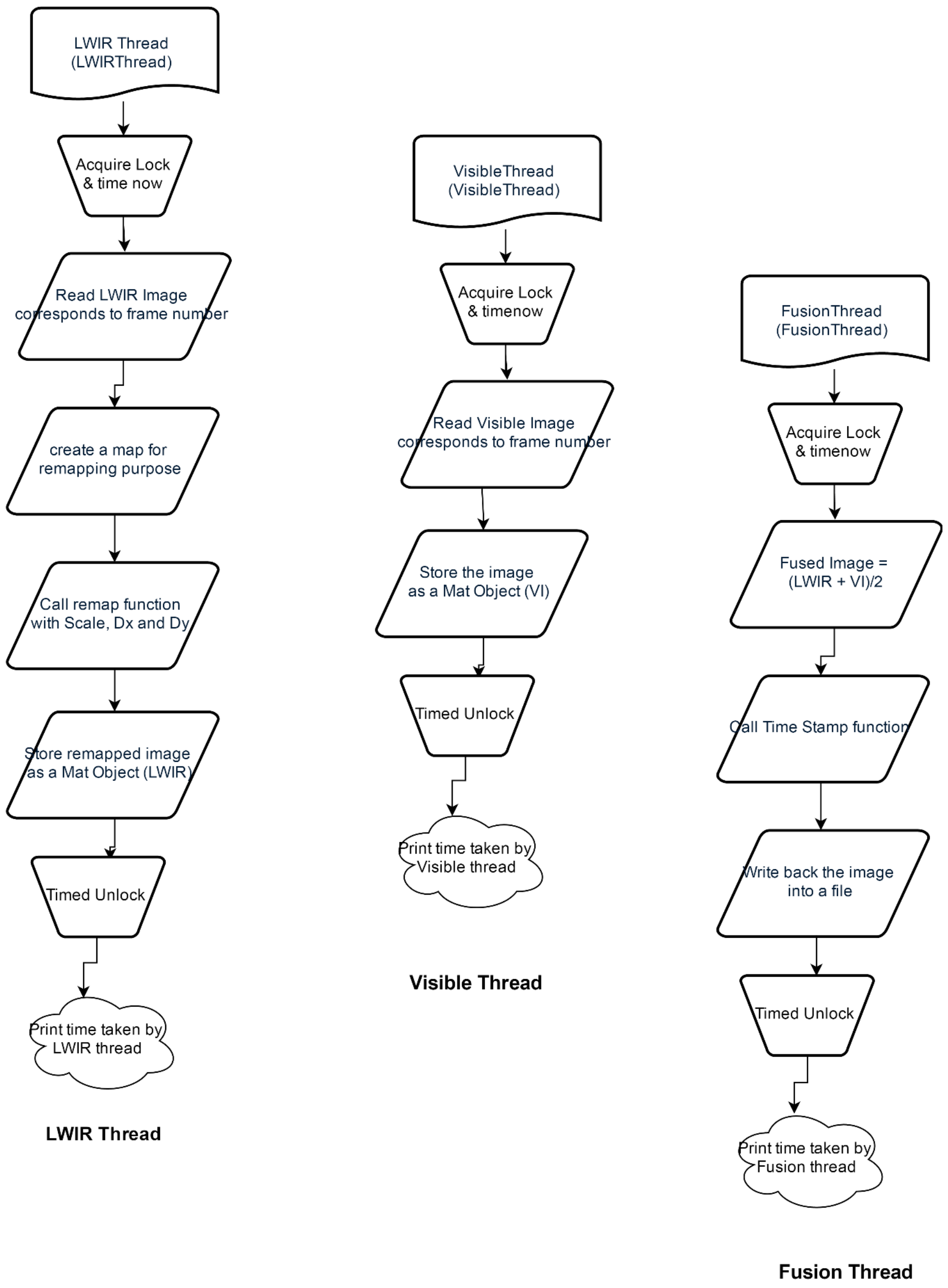

LWIR Thread (LWIRThread)
Acquire Lock & time now
Read LWIR Image corresponds to frame number
create a map for remapping purpose
Call remap function with Scale, Dx and Dy
Store remapped image as a Mat Object (LWIR)
Timed Unlock
Print time taken by LWIR thread
LWIR Thread
VisibleThread (VisibleThread)
Acquire Lock & timenow
Read Visible Image corresponds to frame number
Store the image as a Mat Object (VI)
Timed Unlock
Print time taken by Visible thread
Visible Thread
FusionThread (FusionThread)
Acquire Lock & timenow
Fused Image = (LWIR + VI)/2
Call Time Stamp function
Write back the image into a file
Timed Unlock
Print time taken by Fusion thread
Fusion Thread

Figure 6: LWIR,Visible and Fusion Thread

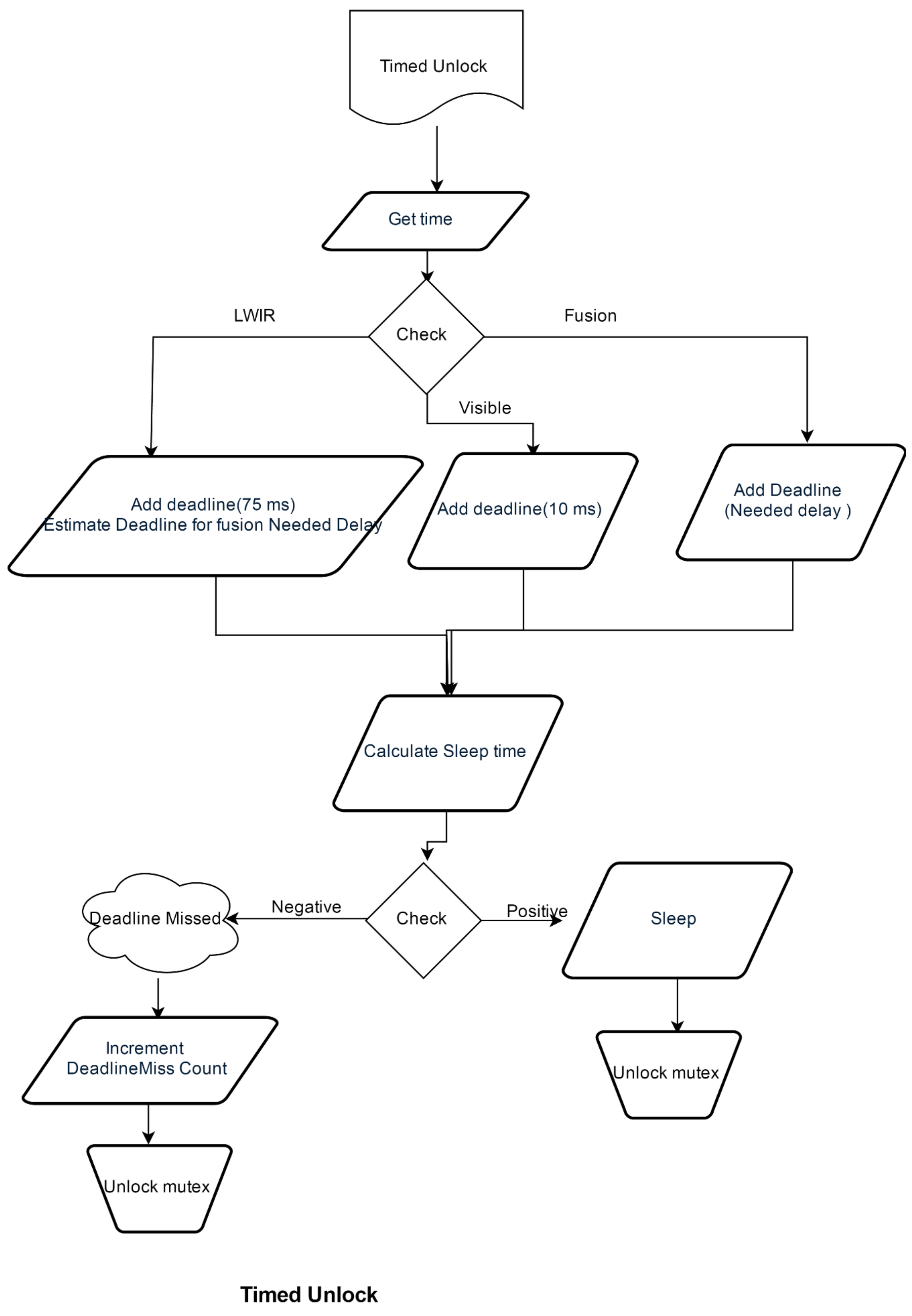


Figure 7: Timed Unlock

Cheddar Analysis

Real time analysis for the tasks that has been mentioned in the previous section is analyzed using cheddar tool, Screenshot has been attached below.

Rate monotonic analysis has been done on the task sets I have mentioned in last section.

Table 1: Deadline scheduled

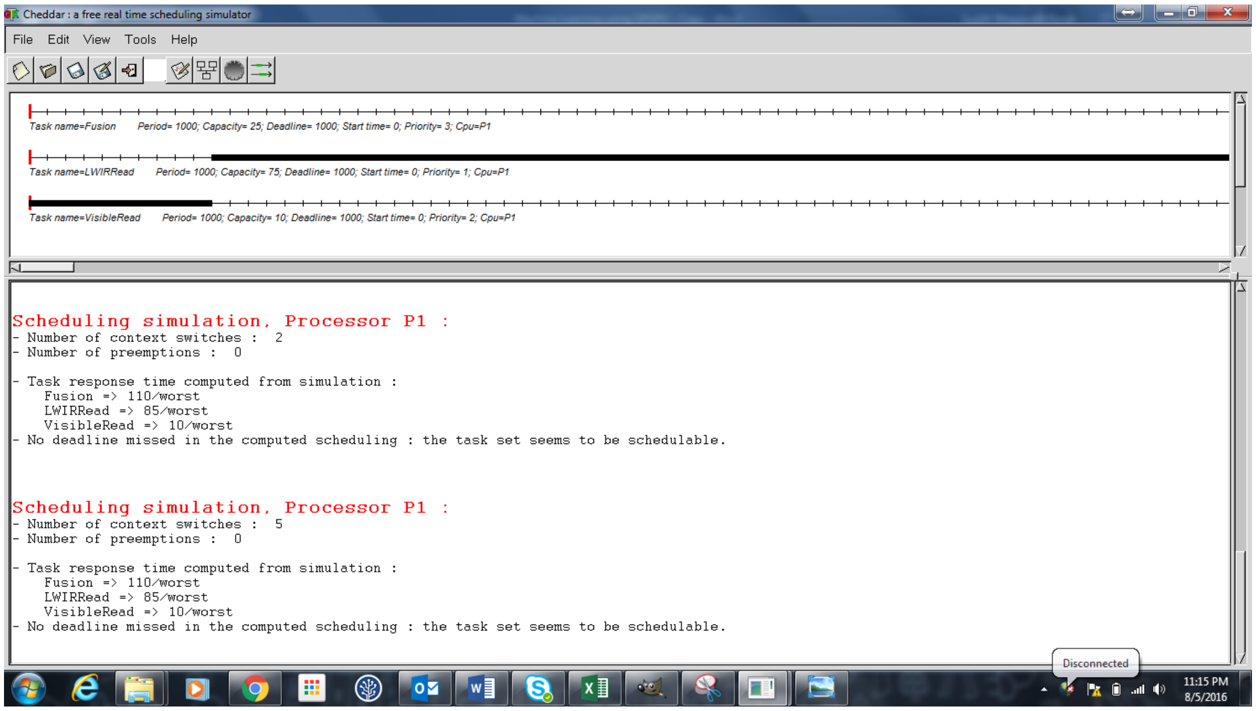
Task	Priority	Deadline	Period
VisibleRead	Medium	10ms	1000ms
LWIRRead	High	75ms	1000ms
Fusion	Low	25ms	1000ms

This is feasible according to cheddar analysis.

The reason for 1000 ms deadline is that, to process every frame for second and time stamp it. These frames are encoded into a video using avconv in Jetson.

```
avconv -f image2 -I in%06d.jpg video.mpg
```

This video is attached with the submission files.



```
Task name=Fusion       Period= 1000; Capacity= 25; Deadline= 1000; Start time= 0; Priority= 3; Cpu=P1
Task name=LWIRRead     Period= 1000; Capacity= 75; Deadline= 1000; Start time= 0; Priority= 1; Cpu=P1
Task name=VisibleRead  Period= 1000; Capacity= 10; Deadline= 1000; Start time= 0; Priority= 2; Cpu=P1
```

```
Scheduling simulation, Processor P1 :
- Number of context switches :  2
- Number of preemptions :  0

- Task response time computed from simulation :
    Fusion => 110/worst
    LWIRRead => 85/worst
    VisibleRead => 10/worst
- No deadline missed in the computed scheduling : the task set seems to be schedulable.


Scheduling simulation, Processor P1 :
- Number of context switches :  5
- Number of preemptions :  0

- Task response time computed from simulation :
    Fusion => 110/worst
    LWIRRead => 85/worst
    VisibleRead => 10/worst
- No deadline missed in the computed scheduling : the task set seems to be schedulable.
```

Figure 8: Cheddar Analysis

Results:

The system has a jitter which is manageable. The jitter is caused by the usleep function.

Jitter is calculated from estimated deadline and Average time taken for it to process every frame. As a system over all jitter is calculated and tabulated below. This jitter will cause a change in the system for every **3030** frames one frame will missout. Average jitter is 0.33 ms which makes a 1 frame miss(1 second jitter delay) at 3030.30th frame.

Table 2: Results from code

Task	Deadline	Time taken (Average)	Jitter – Calculated from code (Average)	
LWIRRead	75	75.203	0.203	
Visible	10	9.997	-0.03	
Fusion	25	25.157	0.157	

Conclusion

The system was designed and implemented from the skills that I have gained throughout the course. Real time analysis tools like Cheddar is really helpful in analyzing complex set of tasks. In this case this is just 3 tasks, So it was easy to analyze it. The system is inspired from SDMSI(Software Defined Multispectral imaging project) and implemented fusion functionality. Real time analysis help understand the system reliability overall.

References

I want to thank Professor Dr.Sam Siewert, TAs for the course and the research group that I was a part of which includes Shivasankar, Ram, Akshay , Ryan, Demi , Dr.Siewert for the help and clarification of the implemented algorithm.

Code

```
/*
Author: Surjith Bhagvath Singh
Functionality: This code performs real time multispectral fusion and analyses
the deadlines

References: Dr.Sam Siewert 's code examples
*/

#include <pthread.h>
#include <stdio.h>
#include <sched.h>
#include <time.h>
#include <stdlib.h>
#include <unistd.h>
#include <iostream>
#include <errno.h>

#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>


using namespace cv;
using namespace std;

#define HRES 320
#define VRES 480
#define NUM_THREADS           4
#define START_SERVICE         0
#define HIGH_PRIO_SERVICE     1
#define MID_PRIO_SERVICE      2
#define LOW_PRIO_SERVICE      3
#define NUM_MSGS        3

//Camera Constants
float ZOOM = 0.6;
float DX = 62.0, DY = 125.0;

//Initializations of global variables
int global_id = 0;
int fps = 30;
char filename_VI[150],filename_IR[150],filename_fused[150];
Mat map_x,map_y,IR,VI,remapped_IR;
struct timespec timenow,timesoon,neededsleep;
int num_frames = 300;
VideoWriter output_fused;
double total_visible=0,total_fusion=0,total_lwir=0;
double sleep_lwir=0,sleep_visible=0,sleep_fusion=0;
int deadlinemiss;


pthread_t threads[NUM_THREADS];
pthread_attr_t rt_sched_attr[NUM_THREADS];
int rt_max_prio, rt_min_prio;
struct sched_param rt_param[NUM_THREADS];
struct sched_param nrt_param;

pthread_mutex_t msgSem;
pthread_mutexattr_t rt_safe;

int rt_protocol;


//Write video functionality
 void write_vid(Mat image)
 {
  output_fused << image;
 }

//timespec difference function
void timespec_diff(struct timespec *start, struct timespec *stop,
struct timespec *result, bool check_neg)
{
  result->tv_sec = stop->tv_sec - start->tv_sec;
  result->tv_nsec = stop->tv_nsec - start->tv_nsec;

  if ( check_neg && result->tv_nsec < 0) {
    result->tv_sec = result->tv_sec - 1;
    result->tv_nsec = result->tv_nsec + 1000000000;
  }

  //printf("\ndiff time %lu sec, %lu nsec\n",result->tv_sec,result->tv_nsec);

}

//Timed Unlock
int timed_unlock(int argument)
{
    struct timespec absolute_time,sleep_time;
    int error;
    double microsleep;

    clock_gettime(CLOCK_MONOTONIC, &absolute_time);

    long long int converted_time = ((timenow.tv_sec*1000000000) +
timenow.tv_nsec );
    if(argument == 1)
    {

      timesoon.tv_sec  = timenow.tv_sec ;
      timesoon.tv_nsec = timenow.tv_nsec+75000000;
      neededsleep.tv_sec = timenow.tv_sec+1;
      neededsleep.tv_nsec= timenow.tv_nsec;

      timespec_diff(&absolute_time,&timesoon,&sleep_time,true);


    }
    else if(argument == 2)
    {
      timesoon.tv_sec  = timenow.tv_sec ;
      timesoon.tv_nsec = timenow.tv_nsec+10000000;
      timespec_diff(&absolute_time,&timesoon,&sleep_time,true);

    }
    else if(argument == 3)
    {
      timesoon.tv_sec  = timenow.tv_sec ;
      timesoon.tv_nsec = timenow.tv_nsec+25000000;

      timespec_diff(&absolute_time,&neededsleep,&sleep_time,true);
    }



    microsleep = ((sleep_time.tv_sec * 1000000000.0f) +
sleep_time.tv_nsec)/1000.0f;

    if(microsleep>=0)
    {
    printf("sleep_time  = %f milliseconds\n",microsleep/1000.0 );
    usleep(microsleep);
      if(argument == 1)
    {
      sleep_lwir = sleep_lwir+microsleep;
    }
    else if(argument == 2)
    {
      sleep_visible = sleep_visible+microsleep;
    }
    else if(argument == 3)
    {
     sleep_fusion = sleep_fusion+microsleep;
    }

    pthread_mutex_unlock(&msgSem);

    }
    else if  (microsleep<0)
    {
      pthread_mutex_unlock(&msgSem);
      printf("Deadline Missed = %llf\n",microsleep);
      deadlinemiss++;
    }

}


// Map updation for remap functionality
void update_map()
{
    for( int j = 0; j < IR.rows; j++ )
    {
        for( int i = 0; i < IR.cols; i++ )
       {
                  map_x.at<float>(j,i) = (1/ZOOM)*(i - IR.cols*(DX/HRES)) ;
                  map_y.at<float>(j,i) = (1/ZOOM)*(j - IR.rows*(DY/VRES)) ;
       }
    }
}


//Time stamp function
Mat time_stamp(Mat image)
{

        time_t rawTime;
        struct tm * timeinfo;
        time (&rawTime);
        timeinfo = localtime(&rawTime);
        printf("timeinfo  %s",asctime(timeinfo));
        putText(image,asctime(timeinfo),Point2f(10,190),FONT_HERSHEY_SIMPLEX,
0.5, Scalar(255,255,255,255));
        return image;
}

void LWIRRead(int, void*)
{
     snprintf(filename_IR,150,
"/home/ubuntu/project/lab6/litiv2012_dataset/SEQUENCE2/THERMAL/input/in%06d.j
pg",global_id);

     IR = imread(filename_IR,CV_LOAD_IMAGE_COLOR);

     map_x.create(IR.size(),CV_32FC1);

     map_y.create(IR.size(),CV_32FC1);

     update_map();



remap(IR,remapped_IR,map_x,map_y,INTER_LINEAR,BORDER_CONSTANT,Scalar(0.6,0.6,
0));



}


//Visible read function
void VISIBLERead(int,void*)
{
   snprintf(filename_VI,150,
"/home/ubuntu/project/lab6/litiv2012_dataset/SEQUENCE2/VISIBLE/input/in%06d.j
pg",global_id);
   VI = imread(filename_VI,CV_LOAD_IMAGE_COLOR);

}

//Fusion
void Fusion(int,void*)
{
      snprintf(filename_fused,150,
"/home/ubuntu/project/lab6/fusion/in%06d.jpg",global_id);

  Mat fused_image = (VI*0.5 + remapped_IR*0.6);

  fused_image = time_stamp(fused_image);

  imwrite(filename_fused,fused_image);



}



//Visible thread
void *VISIBLEThread(void *threadid)
{
  struct timespec start,end,diff,lock_start,lock_end,lock_diff;

  clock_gettime(CLOCK_MONOTONIC,&lock_start);
  pthread_mutex_lock(&msgSem);
  clock_gettime(CLOCK_MONOTONIC,&timenow);
  clock_gettime(CLOCK_MONOTONIC,&start);

  VISIBLERead(0,0);
  clock_gettime(CLOCK_MONOTONIC,&end);


  timespec_diff(&start,&end,&diff,true);

  double time_elapsed = (diff.tv_sec * 1000000000.0f) + diff.tv_nsec ;

  printf("Elapsed time for VisibleThread inside lock is %lu Seconds %llf
milliseconds\n",diff.tv_sec,diff.tv_nsec/1000000.0);
  clock_gettime(CLOCK_MONOTONIC,&timenow);
  timed_unlock(2);
  clock_gettime(CLOCK_MONOTONIC,&lock_end);

  timespec_diff(&lock_start,&lock_end,&lock_diff,true);

  printf("Entire Lock time for VisibleThread is %lu Seconds %llf
milliseconds\n\n",lock_diff.tv_sec,lock_diff.tv_nsec/1000000.0);


  double visible_time = (lock_diff.tv_sec * 1000000000.0f) +
lock_diff.tv_nsec ;
  total_visible = total_visible + visible_time;

}

//FusionThread
void *FusionThread(void *threadid)
{
  struct timespec start,end,diff,lock_start,lock_end,lock_diff;
  clock_gettime(CLOCK_MONOTONIC,&lock_start);
  pthread_mutex_lock(&msgSem);
  clock_gettime(CLOCK_MONOTONIC,&timenow);
  clock_gettime(CLOCK_MONOTONIC,&start);

  Fusion(0,0);
  clock_gettime(CLOCK_MONOTONIC,&end);


  timespec_diff(&start,&end,&diff,true);

  double time_elapsed = (diff.tv_sec * 1000000000.0f) + diff.tv_nsec ;

  printf("Elapsed time for FusionThread inside lock is %lu Seconds %llf 
milliseconds\n",diff.tv_sec,diff.tv_nsec/1000000.0);
  clock_gettime(CLOCK_MONOTONIC,&timenow);
  timed_unlock(3);
  clock_gettime(CLOCK_MONOTONIC,&lock_end);

  timespec_diff(&lock_start,&lock_end,&lock_diff,true);

  printf("Entire Lock time for FusionThread is %lu Seconds %llf 
milliseconds\n\n",lock_diff.tv_sec,lock_diff.tv_nsec/1000000.0);


  double fusion_time = (lock_diff.tv_sec * 1000000000.0f) + lock_diff.tv_nsec 
;
  total_fusion = total_fusion + fusion_time;

}


//LWIR Thread
void *LWIRThread(void *threadid)
{
  struct timespec start,end,diff,lock_start,lock_end,lock_diff;

  clock_gettime(CLOCK_MONOTONIC,&lock_start);
  pthread_mutex_lock(&msgSem);
  clock_gettime(CLOCK_MONOTONIC,&timenow);
  clock_gettime(CLOCK_MONOTONIC,&start);

  LWIRRead(0,0);
  clock_gettime(CLOCK_MONOTONIC,&end);


  timespec_diff(&start,&end,&diff,true);

  double time_elapsed = (diff.tv_sec * 1000000000.0f) + diff.tv_nsec ;

  printf("Elapsed time for LWIRThread inside lock is %lu Seconds %llf 
milliseconds\n",diff.tv_sec,diff.tv_nsec/1000000.0);

  timed_unlock(1);
  clock_gettime(CLOCK_MONOTONIC,&lock_end);

  timespec_diff(&lock_start,&lock_end,&lock_diff,true);
  printf("Entire Lock time for LWIRThread is %lu Seconds %llf
milliseconds\n\n",lock_diff.tv_sec,lock_diff.tv_nsec/1000000.0);

  double lwir_time = (lock_diff.tv_sec * 1000000000.0f) + lock_diff.tv_nsec ;
  total_lwir = total_lwir + lwir_time;

}


//Main transform
void *mytransform(void *threadid)
{
  printf ("\nInside transform function ");
      pthread_mutex_lock(&msgSem);
  pthread_mutex_unlock(&msgSem);
  cpu_set_t cpus;


for(global_id = 1;global_id <= num_frames; global_id++)
{
for (int i = 0; i < 3; i++)
{
    CPU_ZERO(&cpus);
    CPU_SET(i, &cpus);

    int rc;

    if(i==0)
    {
      pthread_attr_setaffinity_np(&rt_sched_attr[HIGH_PRIO_SERVICE],
sizeof(cpu_set_t), &cpus);
      rc = pthread_create(&threads[HIGH_PRIO_SERVICE],
&rt_sched_attr[HIGH_PRIO_SERVICE], LWIRThread, (void *)HIGH_PRIO_SERVICE);

      if (rc)
      {
       printf("ERROR; pthread_create() rc is %d\n", rc);
       perror(NULL);
       exit(-1);
      }
    }

    if(i==1)
    {
      pthread_attr_setaffinity_np(&rt_sched_attr[MID_PRIO_SERVICE],
sizeof(cpu_set_t), &cpus);

      rc = pthread_create(&threads[MID_PRIO_SERVICE],
&rt_sched_attr[HIGH_PRIO_SERVICE], VISIBLEThread, (void *)MID_PRIO_SERVICE);

      if (rc)
      {
       printf("ERROR; pthread_create() rc is %d\n", rc);
       perror(NULL);
       exit(-1);
      }
    }

     if(i==2)
      {
        pthread_attr_setaffinity_np(&rt_sched_attr[LOW_PRIO_SERVICE],
sizeof(cpu_set_t), &cpus);

        rc = pthread_create(&threads[LOW_PRIO_SERVICE],
&rt_sched_attr[HIGH_PRIO_SERVICE], FusionThread, (void *)LOW_PRIO_SERVICE);

        if (rc)
        {
          printf("ERROR; pthread_create() rc is %d\n", rc);
          perror(NULL);
          exit(-1);
        }
      }


  pthread_join(threads[HIGH_PRIO_SERVICE], NULL);
  pthread_join(threads[MID_PRIO_SERVICE], NULL);
  pthread_join(threads[LOW_PRIO_SERVICE], NULL);

}


}

double avg_lwir = total_lwir/num_frames;
double avg_visible = total_visible/num_frames;
double avg_fusion = total_fusion/num_frames;

double avg_sleep_lwir = sleep_lwir/num_frames;
double avg_sleep_visible = sleep_visible/num_frames;
double avg_sleep_fusion = sleep_fusion/num_frames;

printf("Average time for LWIR \t= %llf \nAverage time for Visible \t= %llf
\nAverage time for Fusion \t= %llf \nDeadlines Missed
\t=%d",avg_lwir,avg_visible,avg_fusion,deadlinemiss);

printf("\n\nAverage Sleep time for LWIR \t= %llf \nAverage Sleep time for
Visible \t= %llf \nAverage Sleep time for Fusion \t= %llf
\n",avg_sleep_lwir,avg_sleep_visible,avg_sleep_fusion);


}




//Scheduler print function
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");
   }
}

//Main function
int main ()
{
   int rc, invSafe=0, i, scope;
   struct timespec sleepTime, dTime;

   print_scheduler();

   pthread_attr_init(&rt_sched_attr[START_SERVICE]);
   pthread_attr_setinheritsched(&rt_sched_attr[START_SERVICE],
PTHREAD_EXPLICIT_SCHED);
   pthread_attr_setschedpolicy(&rt_sched_attr[START_SERVICE], SCHED_FIFO);

   pthread_attr_init(&rt_sched_attr[HIGH_PRIO_SERVICE]);
   pthread_attr_setinheritsched(&rt_sched_attr[HIGH_PRIO_SERVICE],
PTHREAD_EXPLICIT_SCHED);
   pthread_attr_setschedpolicy(&rt_sched_attr[HIGH_PRIO_SERVICE],
SCHED_FIFO);

   pthread_attr_init(&rt_sched_attr[LOW_PRIO_SERVICE]);
   pthread_attr_setinheritsched(&rt_sched_attr[LOW_PRIO_SERVICE],
PTHREAD_EXPLICIT_SCHED);
   pthread_attr_setschedpolicy(&rt_sched_attr[LOW_PRIO_SERVICE], SCHED_FIFO);

   rt_max_prio = sched_get_priority_max(SCHED_FIFO);
   rt_min_prio = sched_get_priority_min(SCHED_FIFO);

   rc=sched_getparam(getpid(), &nrt_param);

   if (rc)
   {
       printf("ERROR; sched_setscheduler rc is %d\n", rc);
       perror(NULL);
       exit(-1);
   }

   print_scheduler();

   printf("min prio = %d, max prio = %d\n", rt_min_prio, rt_max_prio);
   pthread_attr_getscope(&rt_sched_attr[START_SERVICE], &scope);
   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");

   pthread_mutex_init(&msgSem, NULL);

   rt_param[START_SERVICE].sched_priority = rt_max_prio;
   pthread_attr_setschedparam(&rt_sched_attr[START_SERVICE],
&rt_param[START_SERVICE]);

   printf("Creating thread %d\n", START_SERVICE);
   rc = pthread_create(&threads[START_SERVICE],
&rt_sched_attr[START_SERVICE], mytransform, (void *)START_SERVICE);

   if (rc)
   {
       printf("ERROR; pthread_create() rc is %d\n", rc);
       perror(NULL);
       exit(-1);
   }
   printf("Start services thread spawned\n");


   printf("will join service threads\n");

   if(pthread_join(threads[START_SERVICE], NULL) == 0)
     printf("START SERVICE done\n");
   else
     perror("START SERVICE");


   rc=sched_setscheduler(getpid(), SCHED_OTHER, &nrt_param);

   if(pthread_mutex_destroy(&msgSem) != 0)
     perror("mutex destroy");

   printf("All done\n");

   exit(0);
}
```