NAME WAN AHMAD ZAINIE BIN WAN MOHAMAD
ID ME131135

The code is tested and working under Ubuntu 14.04 environment. The code has the following dependencies; 1) OpenCV 2.4, and 2) libjpeg8. The instruction on how to build and execute is available at https://github.com/wanahmadzainie/MET1323.

Source code for SERVER

```
/*
 * MET1323: Broadband Multimedia Networks
 *
 * hw2_server.c
 *
 * Objective:
 * 1. Capture frame from webcam.
 * 2. Compress frame using JPEG.
 * 3. Stream compressed frame.
 *
 * CREDITS:-
 * Adapted from
 * 1) http://nashruddin.com/StrEAMinG_oPENcv_vIdEos_ovER_tHe_nEtWoRk
 * 2) http://coderecipes.blogspot.com/2012/07/how-to-decompress-jpeg-images-using.html
 * 3) https://github.com/alishir/IGT_net/tree/master/igt_server
 *
 */

/**
 * stream_server.c:
 * OpenCV video streaming server
 *
 * Author  Nash <me_at_nashruddin.com>
 *
 * See the tutorial at
 * http://nashruddin.com/StrEAMinG_oPENcv_vIdEos_ovER_tHe_nEtWoRk
 */

#include <netinet/in.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
//#include "cv.h"
//#include "highgui.h"
#include <opencv2/core/core_c.h>
#include <opencv2/highgui/highgui_c.h>
#include <opencv2/imgproc/imgproc_c.h>

#include <jpeglib.h>

#define        MAX_WIDTH             640
#define        MAX_HEIGHT            480
#ifndef MSG_LENGTH
#define        MSG_LENGTH            (500*1024)
#endif
#ifndef        JPEG_QUALITY
#define        JPEG_QUALITY   25
#endif

#define PORT 8888

CvCapture*     capture;
IplImage*      img0;
IplImage*      img1;
int                    is_data_ready = 0;
int                    serversock, clientsock;
```

```
uint8_t               bufRGB[MAX_WIDTH * MAX_HEIGHT *3];
uint8_t               bufJPG[MAX_WIDTH * MAX_HEIGHT *3];
uint64_t      len;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void* streamServer(void* arg);
void  quit(char* msg, int retval);
int encode_frame(int width, int height, int quality, uint8_t *in, uint8_t *out, uint64_t *len);

int decode_frame(uint8_t *in, int len, void *out)
{
        int n_samples;
        struct jpeg_error_mgr err;
        struct jpeg_decompress_struct cinfo = {0};

        /* create decompressor */
        jpeg_create_decompress(&cinfo);
        cinfo.err = jpeg_std_error(&err);
        cinfo.do_fancy_upsampling = FALSE;

        /* set source buffer */
        jpeg_mem_src(&cinfo, in, len);

        /* read jpeg header */
        jpeg_read_header(&cinfo, 1);

        /* decompress */
        jpeg_start_decompress(&cinfo);

        /* read scanlines */
        while (cinfo.output_scanline < cinfo.output_height) {
                n_samples = jpeg_read_scanlines(&cinfo, (JSAMPARRAY) &out, 1);
                out += n_samples * cinfo.image_width * cinfo.num_components;
        }

        /* clean up */
        jpeg_finish_decompress(&cinfo);
        jpeg_destroy_decompress(&cinfo);

        return 0;
}

int main(int argc, char** argv)
{
        pthread_t      thread_s;
        int                    key;

        if (argc == 2) {
                capture = cvCaptureFromFile(argv[1]);
        } else {
                capture = cvCaptureFromCAM(0);
        }

        if (!capture) {
                quit("cvCapture failed", 1);
        }

        img0 = cvQueryFrame(capture);
        img1 = cvCreateImage(cvGetSize(img0), IPL_DEPTH_8U, 3);

        cvZero(img1);
        cvNamedWindow("stream_server", CV_WINDOW_AUTOSIZE);
        cvMoveWindow("stream_server", 0, 0);

        /* print the width and height of the frame, needed by the client */
        fprintf(stdout, "width:  %d\nheight: %d\n\n", img0->width, img0->height);
        fprintf(stdout, "Press 'q' to quit.\n\n");

        /* run the streaming server as a separate thread */
        if (pthread_create(&thread_s, NULL, streamServer, NULL)) {
                quit("pthread_create failed.", 1);
```

```
	}

	while(key != 'q') {
		/* get a frame from camera */
		img0 = cvQueryFrame(capture);
		if (!img0) break;

		img0->origin = 0;

		/* compress frame, thread safe */
		pthread_mutex_lock(&mutex);

		len = MAX_WIDTH * MAX_HEIGHT * 3;
		encode_frame(img0->width, img0->height, JPEG_QUALITY, img0->imageData, bufJPG,
&len);

		/* decompress for testing */
		//decode_frame(bufJPG, len, bufRGB);
		//cvSetData(img1, bufRGB, img0->width * 3);//img1 = img0;
		//cvShowImage("stream_server1", img1);

		is_data_ready = 1;
		pthread_mutex_unlock(&mutex);

		/* also display the video here on server */
		cvShowImage("stream_server", img0);
		key = cvWaitKey(30);
	}

	/* user has pressed 'q', terminate the streaming server */
	if (pthread_cancel(thread_s)) {
		quit("pthread_cancel failed.", 1);
	}

	/* free memory */
	cvDestroyWindow("stream_server");
	//cvDestroyWindow("stream_server1");
	quit(NULL, 0);
}

/**
 * This is the streaming server, run as a separate thread
 * This function waits for a client to connect, and send the grayscaled images
 */
void* streamServer(void* arg)
{
	struct sockaddr_in server;

	/* make this thread cancellable using pthread_cancel() */
	pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL);
	pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);

	/* open socket */
	if ((serversock = socket(PF_INET, SOCK_STREAM, 0)) == -1) {
		quit("socket() failed", 1);
	}

	/* setup server's IP and port */
	memset(&server, 0, sizeof(server));
	server.sin_family = AF_INET;
	server.sin_port = htons(PORT);
	server.sin_addr.s_addr = INADDR_ANY;

	/* bind the socket */
	if (bind(serversock, (const void*)&server, sizeof(server)) == -1) {
		quit("bind() failed", 1);
	}

	/* wait for connection */
	if (listen(serversock, 10) == -1) {
		quit("listen() failed.", 1);
	}
```

```
	/* accept a client */
	if ((clientsock = accept(serversock, NULL, NULL)) == -1) {
		quit("accept() failed", 1);
	}

	/* the size of the data to be sent */
	int imgsize = MSG_LENGTH;//img1->imageSize;
	int bytes, i;

	/* start sending images */
	while(1)
	{
		/* send the compressed frame, thread safe */
		pthread_mutex_lock(&mutex);
		if (is_data_ready) {
			//bytes = send(clientsock, img1->imageData, imgsize, 0);
			bytes = send(clientsock, bufJPG, imgsize, 0);
			is_data_ready = 0;
		}
		pthread_mutex_unlock(&mutex);

		/* if something went wrong, restart the connection */
		if (bytes != imgsize) {
			fprintf(stderr, "Connection closed.\n");
			close(clientsock);

			if ((clientsock = accept(serversock, NULL, NULL)) == -1) {
				quit("accept() failed", 1);
			}
		}

		/* have we terminated yet? */
		pthread_testcancel();

		/* no, take a rest for a while */
		usleep(1000);
	}
}

/**
 * this function provides a way to exit nicely from the system
 */
void quit(char* msg, int retval)
{
	if (retval == 0) {
		fprintf(stdout, "%s", (msg == NULL ? "" : msg));
		fprintf(stdout, "\n");
	} else {
		fprintf(stderr, "%s", (msg == NULL ? "" : msg));
		fprintf(stderr, "\n");
	}

	if (clientsock) close(clientsock);
	if (serversock) close(serversock);
	if (capture) cvReleaseCapture(&capture);
	if (img1) cvReleaseImage(&img1);

	pthread_mutex_destroy(&mutex);

	exit(retval);
}

/* JPEG compression, using libjpeg */
int encode_frame(int width, int height, int quality, uint8_t *in, uint8_t *out, uint64_t *len) {
	JSAMPROW row_pointer[1];

	struct jpeg_compress_struct cinfo;
	struct jpeg_error_mgr jerr;

	cinfo.err = jpeg_std_error(&jerr);
	jpeg_create_compress(&cinfo);
```

```
	jpeg_mem_dest(&cinfo, &out, len);

	/* Setting the parameters of the output file. */
	cinfo.image_width = width;
	cinfo.image_height = height;
	cinfo.input_components = 3;
	cinfo.in_color_space = JCS_RGB;
	/* default compression parameters */
	jpeg_set_defaults(&cinfo);

	/* do the compression */
	jpeg_set_quality(&cinfo, quality, TRUE);
	jpeg_start_compress(&cinfo, TRUE);

	while (cinfo.next_scanline < cinfo.image_height) {
		row_pointer[0] = &in[cinfo.next_scanline * cinfo.image_width * 3];
		jpeg_write_scanlines(&cinfo, row_pointer, 1);
	}

	/* clean up */
	jpeg_finish_compress(&cinfo);
	jpeg_destroy_compress(&cinfo);

	return 0;
}
```

Source code for CLIENT

```
/*
 * MET1323: Broadband Multimedia Networks
 *
 * hw2_client.c
 *
 * Objective:
 * 1. Receive stream of JPEG images.
 * 2. Decompress frame.
 * 3. Display raw image.
 *
 * CREDITS:-
 * Adapted from
 * 1) http://nashruddin.com/StrEAMinG_oPENcv_vIdEos_ovER_tHe_nEtWoRk
 * 2) http://coderecipes.blogspot.com/2012/07/how-to-decompress-jpeg-images-using.html
 * 3) https://github.com/alishir/IGT_net/tree/master/igt_server
 *
 */

/**
* stream_client.c:
* OpenCV video streaming client
*
* Author  Nash <me_at_nashruddin.com>
*
* See the tutorial at
* http://nashruddin.com/StrEAMinG_oPENcv_vIdEos_ovER_tHe_nEtWoRk
*/

#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
//#include "cv.h"
//#include "highgui.h"
#include <opencv2/core/core_c.h>
#include <opencv2/highgui/highgui_c.h>
#include <opencv2/imgproc/imgproc_c.h>
```

```
#include <jpeglib.h>

#define       MAX_WIDTH             640
#define       MAX_HEIGHT            480
#ifndef MSG_LENGTH
#define       MSG_LENGTH            (500*1024)
#endif
uint8_t       bufRGB[MAX_WIDTH * MAX_HEIGHT *3];
uint8_t       bufJPG[MAX_WIDTH * MAX_HEIGHT *3];

IplImage* img;
int       is_data_ready = 0;
int       sock;
char*     server_ip;
int       server_port;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void* streamClient(void* arg);
void  quit(char* msg, int retval);
int decode_frame(uint8_t *in, int len, void *out);

int main(int argc, char** argv)
{
       pthread_t thread_c;
       int width, height, key;
       if (argc != 5) {
              quit("Usage: stream_client <server_ip> <server_port> <width> <height>", 0);
       }
       /* get the parameters */
       server_ip   = argv[1];
       server_port = atoi(argv[2]);
       width       = atoi(argv[3]);
       height      = atoi(argv[4]);
       /* create image */
       img = cvCreateImage(cvSize(width, height), IPL_DEPTH_8U, 3);
       cvZero(img);
       /* run the streaming client as a separate thread */
       if (pthread_create(&thread_c, NULL, streamClient, NULL)) {
              quit("pthread_create failed.", 1);
       }
       fprintf(stdout, "Press 'q' to quit.\n\n");
       cvNamedWindow("stream_client", CV_WINDOW_AUTOSIZE);
       cvMoveWindow("stream_client", 680, 0);
       while(key != 'q') {
              /**
               * Display the received image, make it thread safe
               * by enclosing it using pthread_mutex_lock
               */
              pthread_mutex_lock(&mutex);
              if (is_data_ready) {
                     cvShowImage("stream_client", img);
                     is_data_ready = 0;
              }
              pthread_mutex_unlock(&mutex);
              key = cvWaitKey(10);
       }
       /* user has pressed 'q', terminate the streaming client */
       if (pthread_cancel(thread_c)) {
              quit("pthread_cancel failed.", 1);
       }
       /* free memory */
       cvDestroyWindow("stream_client");
       quit(NULL, 0);
}

/**
* This is the streaming client, run as separate thread
*/
void* streamClient(void* arg)
{
       struct  sockaddr_in server;
       /* make this thread cancellable using pthread_cancel() */
```

```
	pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL);
	pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);
	/* create socket */
	if ((sock = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
		quit("socket() failed.", 1);
	}
	/* setup server parameters */
	memset(&server, 0, sizeof(server));
	server.sin_family = AF_INET;
	server.sin_addr.s_addr = inet_addr(server_ip);
	server.sin_port = htons(server_port);
	/* connect to server */
	if (connect(sock, (struct sockaddr*)&server, sizeof(server)) < 0) {
		quit("connect() failed.", 1);
	}
	int  imgsize = MSG_LENGTH;//img->imageSize;
	char sockdata[imgsize];
	int  i, j, k, bytes;
	/* start receiving images */
	while(1) {
		/* get raw data */
		for (i = 0; i < imgsize; i += bytes) {
			if ((bytes = recv(sock, sockdata + i, imgsize - i, 0)) == -1) {
				quit("recv failed", 1);
			}
		}

		/* decompress received frame, convert to IplImage format, thread safe */
		pthread_mutex_lock(&mutex);

		decode_frame(sockdata, MAX_WIDTH * MAX_HEIGHT * 3, bufRGB);
		cvSetData(img, bufRGB, img->width * 3);

		is_data_ready = 1;
		pthread_mutex_unlock(&mutex);
		/* have we terminated yet? */
		pthread_testcancel();
		/* no, take a rest for a while */
		usleep(1000);
	}
}

/**
* This function provides a way to exit nicely from the system
*/
void quit(char* msg, int retval)
{
	if (retval == 0) {
		fprintf(stdout, "%s", (msg == NULL ? "" : msg));
		fprintf(stdout, "\n");
	} else {
		fprintf(stderr, "%s", (msg == NULL ? "" : msg));
		fprintf(stderr, "\n");
	}
	if (sock) close(sock);
	if (img) cvReleaseImage(&img);
	pthread_mutex_destroy(&mutex);
	exit(retval);
}

int decode_frame(uint8_t *in, int len, void *out)
{
	int n_samples;
	struct jpeg_error_mgr err;
	struct jpeg_decompress_struct cinfo = {0};

	/* create decompressor */
	jpeg_create_decompress(&cinfo);
	cinfo.err = jpeg_std_error(&err);
	cinfo.do_fancy_upsampling = FALSE;

	/* set source buffer */
```

```
        jpeg_mem_src(&cinfo, in, len);

        /* read jpeg header */
        jpeg_read_header(&cinfo, 1);

        /* decompress */
        jpeg_start_decompress(&cinfo);

        /* read scanlines */
        while (cinfo.output_scanline < cinfo.output_height) {
                n_samples = jpeg_read_scanlines(&cinfo, (JSAMPARRAY) &out, 1);
                out += n_samples * cinfo.image_width * cinfo.num_components;
        }

        /* clean up */
        jpeg_finish_decompress(&cinfo);
        jpeg_destroy_decompress(&cinfo);

        return 0;
}
```

Result