

Final Project Report: RT part

TITLE: A multi-processor based Vehicle Retrieval System

Course: ECE 7220- Real time embedded computing

Instructor: Dr.DeSouza

Ning, Guanghan pawprint:gnxr9

1.Abstract

This project is interesting in that it renders multi-processor computing for one big task, with reliable communication through the network. It is License Plate Segmentation and Recognition in this project, but it is universal in structure and can be easily reconfigured for other real-life applications.

2. Introduction

The project is about a Vehicle Retrieval System where a camera would capture pictures and localize the license plate, segment the characters and recognize them. Working in an image processing lab and researching in this area, an idea has been there the whole semester to implant the License Plate Detection and Character Recognition program(original work) to Ts-7250 or other hardware with necessary peripherals to simulate the process of vehicle retrieval. After segmentation of the plate into several characters, multiple threads will be created to recognize each character. If one of those threads have met a problem, at least some of the characters will be recognized. This is very important in that if one character is missing, the program would still provide important information for the traffic police.

The License Plate Detection needs the opencv for Linux to be installed. And C make should also be installed to configure opencv for eclipse. The computers in the lab are not allowed to install other software but the project is still managed by putting in header files and C files which specifies and implements certain basic computer vision functions. Honors belong to my advisor because he urged me to use opencv functions and structures as least as possible. IplImage, for example, is replaced by unsigned char* and the width step of the image. Some of the work is done by hand to imitate the process, for example, license plate extraction from the image. Two video clips are attached with the report, one showing the LPD system on a PC, and one showing the imitated LPD system on Ts-7250 boards.

This report focus on the real time embedded computing part, and the computer vision part is explained in the presentation(the **slides** which have details is attached to this report bundle; an **additional report** on this is also attached to this bundle in case curiosity is aroused. It explains the theory behind my approach explicitly).

3. Hardware part

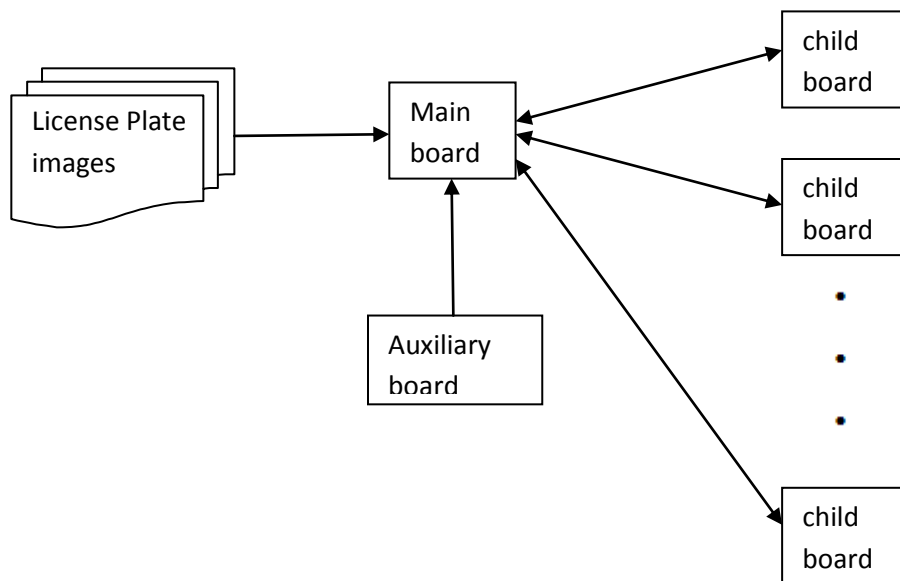
Multiple TS-7250 boards and one auxiliary board are needed.

The "main board" performs segmentation on license plate images. Its auxiliary board provides time to capture/load license plate images.

"Child boards" need to receive segmented characters from the "main board". The characters are transmitted through **ports**.

The "child boards" will recognize the character received and then send the recognized number back to the "main board" through ports.

The "main board" exhibits the recognized numbers.



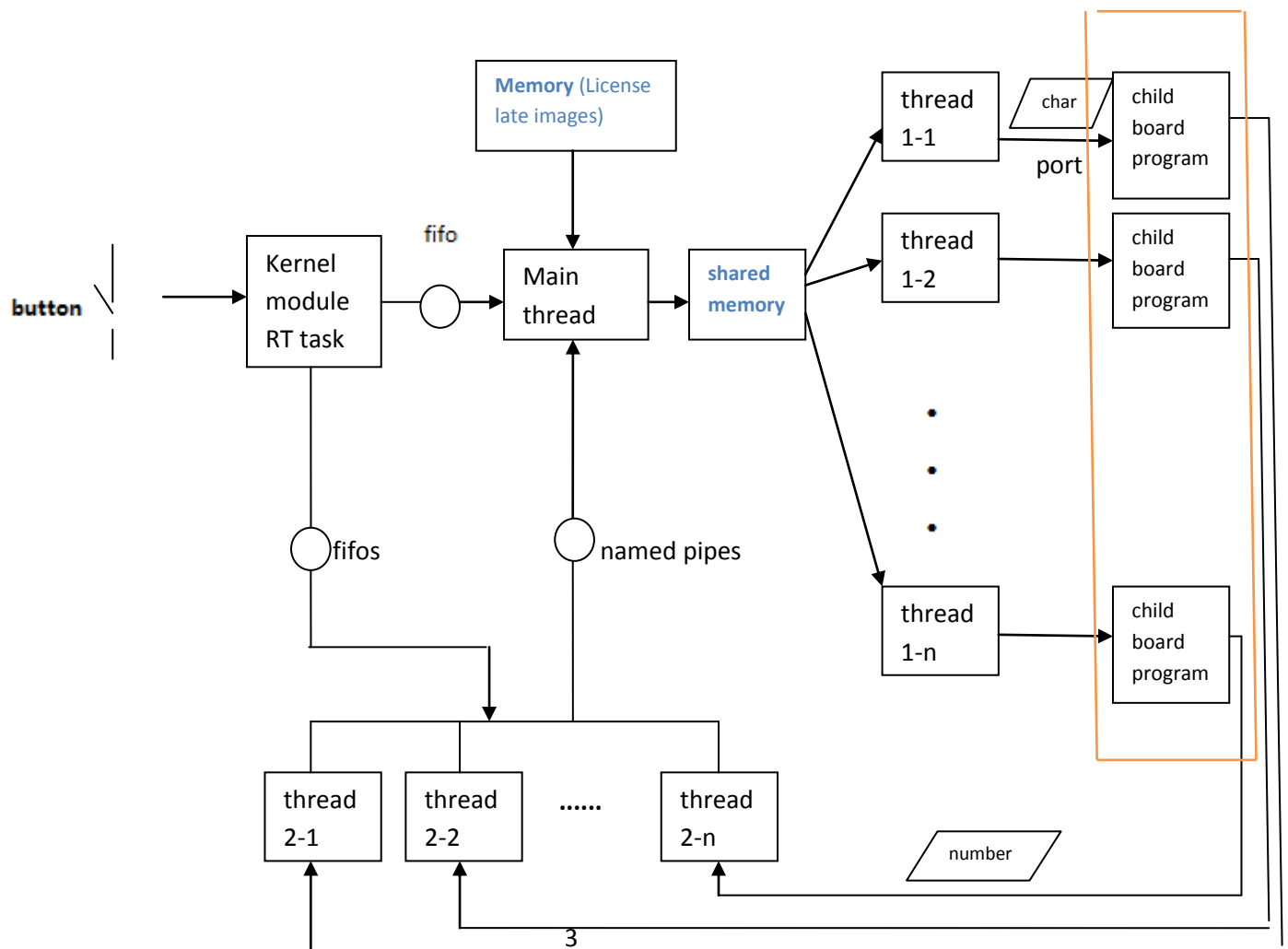
(Diagram1. hardware part)

4. Software part

4.1 The "main board" side

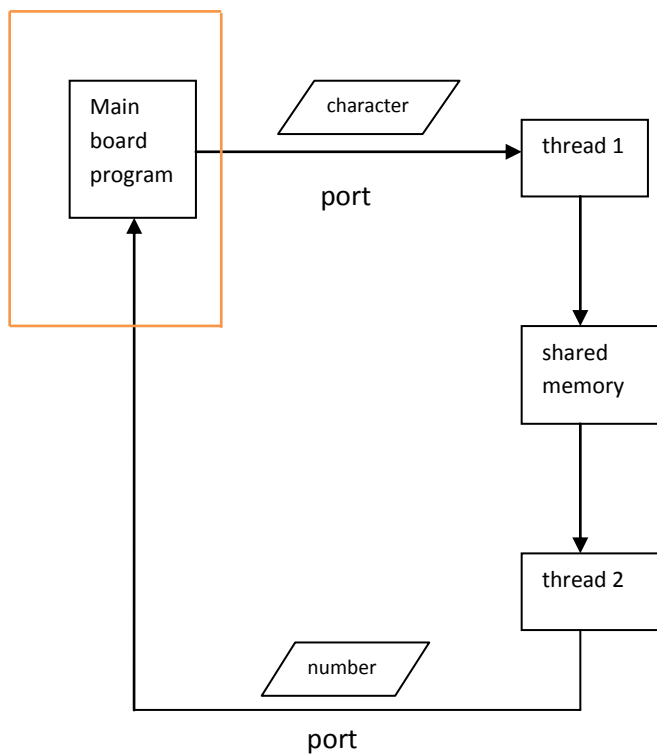
The module for the "main board" will check the B0 button of its auxiliary board.

Once B0 button on the auxiliary board of the TS-7250 "main board" is pressed, which simulates the process of this Vehicle retrieval system capturing the image of vehicles, the module will send the message through a **fifo** to the main thread, who will then read license plates from a pre-defined directory. The main thread will thereafter perform segmentation of the plate.



port

(**Diagram2.** Software part- "main board" side. The part within orange rectangle belongs to "child board program")



(**Diagram3.** Software part- "child board" side. The part within orange rectangle belongs to the "main board")

5. Implementation

5.1 Kernel Module

The kernel module is implemented only for the trigger of reading in an image. Once the button on the auxiliary board is pressed, the kernel module sends a flag into the user space program of the main board through FIFO to acknowledge that the camera is triggered, so an image should be read. In real life

experience, this trigger should come from a pair of sensors on the high-speed road that supervise vehicles. Here it is done by hand to decide when to read an image. The images are license plates images, which are already extracted from frames of vehicles on the road.

The kernel module is implemented by constantly checking the port B of the auxiliary board. If any of the button is pressed, the value of the corresponding register will change and this will trigger the kernel module sending the flag.

5.2 Main board

The Main board reads the image files in "bmp" format and store them in structure CImage. The implementation of this is from open source, contributed by Chen, Lee. The image is stored in shared memory.

Multiple threads are created to perform segmentation of the image into several segments. The segmentation work is original, which is explained in additional report on Computer Vision part.

Another set of multiple threads are created to listen for the transmitting of recognized numbers from child boards. Flags are used for the synchronization. Each digit corresponds to a flag, which indicates that the digit is received.

When all the flags are indicating positive, the main thread of the board will display the recognized plate number. And it clears all these flags afterwards. Unknown digits will be displayed as "*".

5.3 Socket and TCP

It is UDP that broadcasts in Lab5 but in this project TCP is chosen as a reliable way to transmit images. Besides, the ports are pre-allocated to boards in the lab by hand. Multiple sockets will be created and they use different ports. The main board creates multiple sockets and is the server of them to send segments to child boards. At the same time, the main board is also the client of each child board server, which sends recognized digits to the main board. Likewise, each child board plays both roles of server and client, but in different sockets.

In the project, the main board program creates sockets as a server first. When child board programs are executed, the child board programs will connect to the main board, and then creates their own sockets as a server. When the button of the auxiliary board is pressed for the first time, threads will be created for the main board to join and connect to child board programs as a client.

Again, the synchronization is done with flags.

5.4 Child board

The child board program has two threads, one creating a socket and be the server, and one connecting to the main board and be the client.

The server thread sends the recognized number, while the client thread receives segments and perform the character recognition, whose theories are explained in the additional report.

6. Results and future work:

The framework of this project is steady, with the communication being reliable and stable. It is also flexible to reconfigure into other real-life applications. And the recognition rate is 99% based on a 400 training characters/100 testing images trial. The segmentation, however, needs future work to reinforce in the embedded system. Without OpenCV, some elegance in the code is gone. Segmentation is to be re-written to deal with the raw data more properly to avoid loss of information.

Future work will be based on the framework. Since the license plate detection, practically speaking, does not necessarily need multi-processor to run and send results back through network, it is the framework that counts. Other cases, for example, tasks like data training of deep learning that takes a long time to finish, would require multiple processors to work at the same time. Another scenario will be, when a task is on computational intelligence, for example, running EDA algorithm or just genetic algorithm to decide, for instance, a deformable vehicle model. Then the results for each run could be different, and the final result will be based on a statistical evaluation. The multi-processor frame will be useful in that it can be multiple processors running the same program on the same data, and they may render different results. They can send their own results back to the core program to decide the final result based on statistical evaluation. Other examples will be without number, which on the other hand proves this course to be important and useful.

7. Appendix

NOTE: Not all the code included here but only the important ones.

All the code not included here can be found in the attachment of the report bundle.

```
/*
=====
Name      : UserSpace Main Board.c
Author    : Ning, Guanghan
Version   :
Copyright : University of Missouri
Description : Hello World in C, Ansi-style
=====
*/
//For TCP image transmit
#include <sys/types.h>
#include <sys/socket.h>
```

```

#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <time.h>
//=====

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h> //header for creating threads
#include <sys/time.h> //header for time
#include <unistd.h>
#include <fcntl.h>
#include "serial_ece4220.h" //header for Reading from Serial Program

#include <semaphore.h> //header for semaphores
#include "rtai.h" //headers for rtai
#include "rtai_lxrt.h"
#include "rtai_sched.h"
#include "sys/time.h" //header for time

//#include "stdafx.h"
#include "chenLeeCV.h"
#include "segment.h"

#define MAXSIZE 14

int kernel_pressed= 0;
int segment_finished= 0;
int flag= -1, flag1= -1; flag2= -1;
//int flag_num_received= 0;
int flag_num1_received= 0;
int flag_num2_received= 0;
int flag_num4_received= 0;
int first_time= 1;
int N_img= 1;
CImage* segmentImg;

unsigned char rect1[MAXSIZE];
unsigned char rect2[MAXSIZE];
unsigned char rect3[MAXSIZE];
unsigned char rect4[MAXSIZE];

int digit1, digit2, digit4;
typedef struct Plate
{
    unsigned char segment1[MAXSIZE];
    unsigned char segment2[MAXSIZE];
    unsigned char segment3[MAXSIZE];
    unsigned char segment4[MAXSIZE];
}plate;

//=====
//Computer Vision Functions

```

```

void Read_image()
{
    free(segmentImg);

    char *imgName; //int save_success;
    sprintf(imgName, "%d%s", N_img, ".bmp");
    printf("...\n");
    segmentImg = clLoadImage(imgName);
    printf("An image has been successfully read!\n");
    printf("...\n");
    clSaveImage("result2.bmp", segmentImg);
    printf("...\n");

    //if(save_success)

    N_img++;
    if(N_img==6) N_img= 1; //there are only 6 plate images
    kernel_pressed= 0;
}

void Segmentation(unsigned char* s1, unsigned char* s2, unsigned char* s3, unsigned char*s4)
{

    KMeanType * KM;
    KM = (KMeanType *)malloc(sizeof(KMeanType));
    //KM->nKMeanClusters = 2;
    //allocate memmory for it
    kMeanInit(KM);

    int widthStep;
    widthStep= (segmentImg->channels)*(segmentImg->width);

    int xxx= 0, sss= 0;
    int *xx, *ss;
    xx= &xxx; ss= &sss;
    segments( (unsigned char *)segmentImg->imageData, widthStep, 0, 0, 0, 0, KM, xx,ss);
    free(KM);

    //resize each segment patch and recognize
    double q= 2.5; //a character is q times the length of a "."
    int s, x; //the scale of "." : the number of pixels a "." occupies
    int space= 2;
    s= (*ss);
    x= (*xx);

    //standard segment
    CImage* standards;
    //Get character ROIs
    CVRect rect_for_character[7]; //express with xx and ss
    int result; int k; int more= 0;
    for (k=0; k<7; k++) //from k = 2, to ignore the chinese characters before the numbers
    {
        //printf("This is segmentation of the %d rect of character\n", k);
        rect_for_character[k].x = x+ k* ((q*s) )+ more;
        rect_for_character[k].y = 0;
        if(k>1) rect_for_character[k].x += s;
        rect_for_character[k].height= segmentImg->height;
        rect_for_character[k].width= q*s;

        standards = (CImage*)malloc(sizeof(CImage));
        standards->width= q*s;
        standards->height= segmentImg->height;
        int widStep= (standards->channels)*(standards->width);
        int wid= standards->width;
        int ht= standards->height; //printf("WidStep, wid, ht is: %d, %d, %d\n", widStep, wid, ht);
    }
}

```



```

        int i,j;
        for (i=0; i<ht; i++)
            for (j=0; j<wid; j++)
            {
                standards->imageData[i*widStep+ j*3+ k] = segmentImg->imageData[(i+
rect_for_character[k].y)* widthStep+ (j+ rect_for_character[k].x)*3+ k];
            }
            if(k== 0) s1= standards;
            if(k== 1) s2= standards;
            if(k== 3) s4= standards;
        }
    printf("Segmentation done!\n");
}
//=====
//helpful small functions
int receive(char* msg, char* buffer, int q)
{
    int same;
    same= strcmp(msg, buffer, q); //note that the length should be the same!!!
    if(same== 0) //means (msg == buffer)
    {
        printf("%s received!\n", msg);
        return 1;
    }
    else
    {
        //printf("%s not received!\n", msg);
        return 0;
    }
}
//=====
//Functions in threads
void SendSegment4(unsigned char* s)
{
    //send recognized number to the main board(as server)
    int sockfd, new_fd;
    struct sockaddr_in my_addr;
    struct sockaddr_in their_addr;
    unsigned int sin_size, myport, listnum;
    myport = 2012;
    listnum = 10; //maximum listening number > number of characters in a plate

    if((sockfd = socket(PF_INET, SOCK_STREAM, 0)) == -1 )
    {
        perror("socket is error\n");
        exit(1);
    }
    my_addr.sin_family = PF_INET;
    my_addr.sin_port = htons(myport);
    my_addr.sin_addr.s_addr = INADDR_ANY; //my ip address
    bzero(&(my_addr.sin_zero),0);

    //=====
    their_addr.sin_family = PF_INET;
    their_addr.sin_port = htons(myport);
    their_addr.sin_addr.s_addr = inet_addr("10.3.52.15"); //all ip address can be clients
    bzero(&(their_addr.sin_zero),0);
    //=====

    if(bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr)) == -1)
    {
        perror("bind is error\n");
        exit(1);
    }

    if(listen(sockfd, listnum) == -1)

```

```

    {
        perror("listen is error\n");
        exit(1);
    }
    //printf("HELLO?!\n");
    printf("start to listen for child Board <4>\n");

    while(1)
    {
        sin_size = sizeof(struct sockaddr_in);

        if((new_fd = accept(sockfd,(struct sockaddr *)&their_addr,&sin_size)) == -1)
        {
            perror("accept is error\n");
            continue;
        }

        printf("Main board server to send segment<4>:\n got connection from child
board %s\n",inet_ntoa(their_addr.sin_addr));
        int connected = 1;

        char *p;
        char sock_buf[1024];
        bzero(sock_buf, 1024);
        p = sock_buf;

        time_t t; int rand_n;

        while(connected)
        {
            if(flag>=0)
            {
                flag1= flag; //copy flag for another sendsegments
                flag2= flag; //copy flag for another sendsegments
                usleep(1000000); //wait for 1s

                printf("send 4th segment!\n");
                if(send(new_fd, &s, MAXSIZE, 0) == -1) //send the segment to
corresponding child board
                {
                    perror("send is error\n");
                    close(new_fd);
                    exit(0);
                }
                flag= -1;
            }
        }
    }
}

void SendSegment1(unsigned char* s)
{
    //send recognized number to the main board(as server)
    int sockfd, new_fd;
    struct sockaddr_in my_addr;
    struct sockaddr_in their_addr;
    unsigned int sin_size,myport,listnum;
    myport = 2013;
    listnum = 10; //maximum listening number > number of characters in a plate

    if((sockfd = socket(PF_INET,SOCK_STREAM, 0)) == -1)
    {
        perror("socket is error/n;");
        exit(1);
    }

```

```

}
my_addr.sin_family = PF_INET;
my_addr.sin_port = htons(myport);
my_addr.sin_addr.s_addr = INADDR_ANY; //my ip address
bzero(&(my_addr.sin_zero),0);

//=====
their_addr.sin_family = PF_INET;
their_addr.sin_port = htons(myport);
their_addr.sin_addr.s_addr = inet_addr("10.3.52.14"); //all ip address can be clients
bzero(&(their_addr.sin_zero),0);
//=====

if(bind(sockfd,(struct sockaddr *)&my_addr,sizeof(struct sockaddr)) == -1)
{
    perror("bind is error\n");
    exit(1);
}

if(listen(sockfd,listnum) == -1)
{
    perror("listen is error\n");
    exit(1);
}
//printf("HELLO?!\n");
printf("start to listen for child Board <1>\n");

while(1)
{
    sin_size = sizeof(struct sockaddr_in);

    if((new_fd = accept(sockfd,(struct sockaddr *)&their_addr,&sin_size)) == -1)
    {
        perror("accept is error\n");
        continue;
    }
    printf("Main board server to send segment<1>:\n got connection from child
board %s\n",inet_ntoa(their_addr.sin_addr));
    int connected = 1;

    char *p;
    char sock_buf[1024];
    bzero(sock_buf, 1024);
    p = sock_buf;

    time_t t; int rand_n;

    while(connected)
    {
        if(flag1>=0)
        {
            usleep(1000000); //wait for 1s
            printf("send 1st segment!\n");
            if(send(new_fd, &s,MAXSIZE, 0) == -1) //send the segment to
corresponding child board
            {
                perror("send is error\n");
                close(new_fd);
                exit(0);
            }
            flag1=-1;
            /*Only one sendsegment thread clears the flag*/
        }
    }
}

```

```

    }
}
void SendSegment2(unsigned char* s)
{
    //send recognized number to the main board(as server)
    int sockfd, new_fd;
    struct sockaddr_in my_addr;
    struct sockaddr_in their_addr;
    unsigned int sin_size, myport, listnum;
    myport = 2014;
    listnum = 10; //maximum listening number > number of characters in a plate

    if((sockfd = socket(PF_INET, SOCK_STREAM, 0)) == -1)
    {
        perror("socket is error\n");
        exit(1);
    }
    my_addr.sin_family = PF_INET;
    my_addr.sin_port = htons(myport);
    my_addr.sin_addr.s_addr = INADDR_ANY; //my ip address
    bzero(&(my_addr.sin_zero), 0);

    //=====
    their_addr.sin_family = PF_INET;
    their_addr.sin_port = htons(myport);
    their_addr.sin_addr.s_addr = inet_addr("10.3.52.13"); //all ip address can be clients
    bzero(&(their_addr.sin_zero), 0);
    //=====

    if(bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr)) == -1)
    {
        perror("bind is error\n");
        exit(1);
    }

    if(listen(sockfd, listnum) == -1)
    {
        perror("listen is error\n");
        exit(1);
    }
    printf("start to listen for child Board <2>\n");

    while(1)
    {
        sin_size = sizeof(struct sockaddr_in);

        if((new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size)) == -1)
        {
            perror("accept is error\n");
            continue;
        }

        printf("Main board server to send segment<2>:\n got connection from child
board %s\n", inet_ntoa(their_addr.sin_addr));
        int connected = 1;

        char *p;
        char sock_buf[1024];
        bzero(sock_buf, 1024);
        p = sock_buf;

        time_t t; int rand_n;

```

```

        while (connected)
        {
            if (flag2 >= 0)
            {
                usleep(1000000); //wait for 1s
                printf("send 2nd segment!\n");
                if (send(new_fd, &s, MAXSIZE, 0) == -1) //send the segment to
                    corresponding child board
                {
                    perror("send is error\n");
                    close(new_fd);
                    exit(0);
                }
                flag2 = -2;
                /*Only one sendsegment thread clears the flag*/
            }
        }
    }
}

void ReceiveNumber1(int* p_num)
{
    //Client of receiving numbers
    //Receive numbers from child boards: TCP protocol
    int sockfd;
    char buffer[1024];
    struct sockaddr_in server_addr;
    struct hostent *host;
    int portnumber, nbytes;

    if ((host = gethostbyname("10.3.52.14")) == NULL)
    {
        printf("Gethostname error\n");
        exit(1);
    }
    if ((portnumber = atoi("2001")) < 0)
    {
        printf("Port number error\n");
        exit(1);
    }
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
    {
        printf("Socket Error:%s\n", strerror(errno));
        exit(1);
    }
    bzero(&server_addr, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(portnumber);
    server_addr.sin_addr = *((struct in_addr *) host->h_addr);
    if (connect(sockfd, (struct sockaddr *) &server_addr,
        sizeof(struct sockaddr)) == -1)
    {
        printf("Connect Error:%s\n", strerror(errno));
        exit(1);
    }

    while (1)
    {
        if ((nbytes = read(sockfd, buffer, 1024)) == -1)
        {
            printf("Read Error:%s\n", strerror(errno));
            exit(1);
        }
        buffer[nbytes] = '\0';
        printf("Received number from child board(1): %s\n", buffer);

        //Send received numbers to main thread by named pipes
    }
}

```

```

        /*p_num= buffer[nbytes-1]-'0';
        *p_num= buffer[0]-'0';
        //printf("p_num= %d\n", *p_num);
        //Send received numbers to main thread by named pipes
        // int pipe1;
        // pipe1= open("Number1",O_WRONLY);
        // write(pipe1, p_num, sizeof(int));
        //num1= *p_num;
        digit1= *p_num;
        flag_num1_received= 1;

        usleep(2000000); //wait for 0.1s
    }
    close(sockfd);
}

void ReceiveNumber2(int* p_num)
{
    //Receive numbers from child boards: TCP protocol
    int sockfd;

    char buffer[1024];
    struct sockaddr_in server_addr;
    struct hostent *host;
    int portnumber,nbytes;

    if((host=gethostbyname("10.3.52.13"))==NULL)
    {
        printf("Gethostname error\n");
        //exit(1);
    }

    if((portnumber=atoi("2002"))<0)
    {
        printf("Port number error\n");
        //exit(1);
    }
    if((sockfd=socket(AF_INET,SOCK_STREAM,0))==1)
    {
        printf("Socket Error:%s\n",strerror(errno));
        //exit(1);
    }
    bzero(&server_addr,sizeof(server_addr));
    server_addr.sin_family=AF_INET;
    server_addr.sin_port=htons(portnumber);
    server_addr.sin_addr=*((struct in_addr *)host->h_addr);
    if(connect(sockfd,(struct sockaddr *)&server_addr,
    sizeof(struct sockaddr))==1)
    {
        printf("Connect Error:%s\n",strerror(errno));
        //exit(1);
    }

    while(1)
    {
        if((nbytes=read(sockfd,buffer,1024))==1)
        {
            printf("Read Error:%s\n",strerror(errno));
            exit(1);
        }
        buffer[nbytes]='\0';
        printf("Received number from child board(2): %s\n",buffer);

        //Send received numbers to main thread by named pipe;
        *p_num= buffer[0]-'0';
    }
}

```

```

        digit2= *p_num;
        flag_num2_received= 1;

        usleep(2000000); //wait for 0.1s
    }
    close(sockfd);
}

void ReceiveNumber4(int* p_num)
{
    //Receive numbers from child boards: TCP protocol as client
    int sockfd;
    char buffer[1024];
    struct sockaddr_in server_addr;
    struct hostent *host;
    int portnumber,nbytes;

    if((host=gethostbyname("10.3.52.15"))==NULL)
    {
        printf("Gethostname error\n");
        //exit(1);
    }

    //segmentation fault here

    if((portnumber=atoi("2004"))<0)
    {
        printf("Port number error\n");
        //exit(1);
    }
    if((sockfd=socket(AF_INET,SOCK_STREAM,0))==-1)
    {
        printf("Socket Error:%s\n",strerror(errno));
        //exit(1);
    }
    bzero(&server_addr,sizeof(server_addr));
    server_addr.sin_family=AF_INET;
    server_addr.sin_port=htons(portnumber);
    server_addr.sin_addr=*((struct in_addr *)host->h_addr);
    if(connect(sockfd,(struct sockaddr *)&server_addr,
    sizeof(struct sockaddr))==-1)
    {
        printf("Connect Error:%s\n",strerror(errno));
        //exit(1);
    }

    while(1)
    {
        if((nbytes=read(sockfd,buffer,1024))!=-1)
        {
            printf("Read Error:%s\n",strerror(errno));
            //exit(1);
        }
        buffer[nbytes]='\0';
        printf("Received number from child board(4): %s\n",buffer);

        //Send received numbers to main thread by named pipes
        // *p_num= buffer[nbytes-1]-'0';
        *p_num= buffer[0]-'0';
        //printf(" *p_num= %d\n", *p_num);
        // int pipe4;
        // pipe4= open("Number4",O_WRONLY);
        // write(pipe4, p_num, sizeof(int));
        //num4= *p_num;
    }
}

```

```

        digit4= *p_num;
        flag_num4_received= 1;
        usleep(2000000); //wait for 0.1s
    }
    close(sockfd);
}

void ReadImage(plate *detected_plate)
{
    int check;
    pthread_t num1_thread;
    pthread_t num2_thread;
    pthread_t num3_thread;
    pthread_t num4_thread;
    int num1, num2, num3, num4;

    int fd_fifo_in;
    fd_fifo_in= open("/dev/rtf/1", O_RDWR); //1 is fifo_write from kernel. then it is fifo_in for user space

    //int pipe1;
    // pipe1= open("prints",O_WRONLY);
    first_time= 1;
    while(1)
    {
        usleep(250000);
        //read from fifo to see if button is pressed
        flag = read(fd_fifo_in, &kernel_pressed, sizeof(kernel_pressed));
        printf("\n Button is pressed!\n");
        if ((flag>=0)&&(first_time== 1))
        {
            printf("Begin to receive number from child boards!\n");

            check = pthread_create(&num1_thread, NULL, (void *)&ReceiveNumber1, (void *)&(num1));
            if (check!=0) printf("num1_thread error\n");
            check = pthread_create(&num2_thread, NULL, (void *)&ReceiveNumber2, (void *)&(num2));
            if (check!=0) printf("num2_thread error\n");
            //check = pthread_create(&num3_thread, NULL, (void *)&ReceiveNumber3, (void *)&(num3));
            // if (check!=0) printf("num3_thread error\n");*/
            check = pthread_create(&num4_thread, NULL, (void *)&ReceiveNumber4, (void *)&(num4));
            if (check!=0) printf("num4_thread error\n");

            first_time= 0;
            usleep(250000);
            flag= -1;
        }
        if(flag>= 0)
        {
            segment_finished= 0;
            Read_image();
            Segmentation((*detected_plate).segment1,(*detected_plate).segment2, (*detected_plate).segment3,
            (*detected_plate).segment4 );
            segment_finished= 1; //flag to indicate this for the other 2 set of threads
            usleep(250000);
            flag= -1;
        } else;
    }
}

//=====

int main(void)
{
    //-----//
    printf("-----\n");
    printf("-----Plate segmentation and recognition-----\n");
}

```



```

printf("-----Main Board Program-----\n");
printf("-----\n");
printf("\n\n\n\n\n\n\n\n");

//-----Declaration of variables-----//
int check;
int pipe1, pipe2, pipe3, pipe4;
pthread_t ReadImage_thread;

pthread_t seg1_thread;
pthread_t seg2_thread;
pthread_t seg3_thread;
pthread_t seg4_thread;

pthread_t num1_thread;
pthread_t num2_thread;
pthread_t num3_thread;
pthread_t num4_thread;

plate *detected_plate= (plate*)malloc(sizeof(plate));

int fd_fifo_in, fd_fifo_out;
fd_fifo_in= open("/dev/rtf/1", O_RDWR); //1 is fifo_write from kernel. then it is fifo_in for user space
fd_fifo_out= open("/dev/rtf/o", O_RDWR);

system("mkfifo Number1 >& /dev/null"); //create a pipe file
system("mkfifo Number2 >& /dev/null"); //create another pipe file
system("mkfifo Number3 >& /dev/null"); //create a pipe file
system("mkfifo Number4 >& /dev/null"); //create another pipe file

//Create the thread to read images from memory and do extraction and segmentation
check = pthread_create(&ReadImage_thread, NULL, (void *)&ReadImage, (void *)(&detected_plate));
if (check!=0) printf("ReadImage_thread error\n");
//wait until this is finished
//pthread_join(ReadImage_thread, NULL);

//Create the thread to read images from memory
check = pthread_create(&seg1_thread, NULL, (void *)&SendSegment1, (void *)(&detected_plate).segment1));
if (check!=0) printf("seg1_thread error\n");
check = pthread_create(&seg2_thread, NULL, (void *)&SendSegment2, (void *)(&detected_plate).segment2));
if (check!=0) printf("seg2_thread error\n");
// check = pthread_create(&seg3_thread, NULL, (void *)&SendSegment, (void *)(&detected_plate).segment3));
// if (check!=0) printf("seg3_thread error\n");
check = pthread_create(&seg4_thread, NULL, (void *)&SendSegment4, (void *)(&detected_plate).segment4));
if (check!=0) printf("seg4_thread error\n");

while(1)
{
    //exhibit the recognized numbers when button is pressed
    if((flag_num1_received)&&(flag_num2_received)&&(flag_num4_received))
    {
        printf("The plate number is: ** %d*%d*d\n", digit4, digit2, digit1);
        flag_num1_received= 0;
        flag_num2_received= 0;
        flag_num4_received= 0;
    }
}

pthread_join(ReadImage_thread, NULL);

return EXIT_SUCCESS;
}

```

```

//=====
/* User space Child Board program */
//Each child board has a program
//They are the same except the index and the port number
//Here is the code for child board <1>
//=====
//For TCP image transmit
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#define MAXSIZE 1000

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <time.h>

#include "errno.h"
#include "sys/types.h"

#include "netinet/in.h"
#include "sys/wait.h"

#include <pthread.h> //header for creating threads
#include <sys/time.h> //header for time
#include <fcntl.h>

#include <semaphore.h> //header for semaphores
#include "rtai.h" //headers for rtai
#include "rtai_lxrt.h"
#include "rtai_sched.h"
#include "sys/time.h" //header for time

#include <recognition.h> //header for character recognition
#include <chenLeeCV.h> //for CImage structure
#define STANDARD_WIDTH 32

//shared memory
int num1, num2, num3, num4; int copy;
int flag_received_segment= 0;

//Computer Vision functions
int SVM_Recognition(unsigned char* segment)
{
    int result;
        CImage Segment;
        Segment->imageData= segment;
        Segment->width= STANDARD_WIDTH;
        Segment->channels= 8;

        configureSystem(&gst);
        initSystem(&gst,svm);
        result= svmTestCharacter(&gst,svm, Segment, k);

    return result;
}

```

```

void SendNumber1(int num)
{
    //send recognized number to the main board(as server)
    int sockfd, new_fd;
    struct sockaddr_in my_addr;
    struct sockaddr_in their_addr;
    unsigned int sin_size, myport, listnum;
    myport = 2001;
    listnum = 10; //maximum listening number > number of characters in a plate

    if((sockfd = socket(PF_INET, SOCK_STREAM, 0)) == -1)
    {
        perror("socket is error\n");
        exit(1);
    }
    my_addr.sin_family = PF_INET;
    my_addr.sin_port = htons(myport);
    my_addr.sin_addr.s_addr = INADDR_ANY; //my ip address
    bzero(&(my_addr.sin_zero), 0);

    //=====
    their_addr.sin_family = PF_INET;
    their_addr.sin_port = htons(myport);
    their_addr.sin_addr.s_addr = inet_addr("10.3.52.16"); //all ip address can be clients
    bzero(&(their_addr.sin_zero), 0);
    //=====

    if(bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr)) == -1)
    {
        perror("bind is error\n");
        exit(1);
    }

    if(listen(sockfd, listnum) == -1)
    {
        perror("listen is error\n");
        exit(1);
    }
    printf("start to listen for the main board\n");

    while(1)
    {
        sin_size = sizeof(struct sockaddr_in);

        if((new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size)) == -1)
        {
            perror("accept is error\n");
            continue;
        }

        printf("Child Board server to send recognized number: \n got connection from Main Board %s\n", inet_ntoa(their_addr.sin_addr));
        int connected = 1;

        char *p;
        char sock_buf[1024];
        bzero(sock_buf, 1024);
        p = sock_buf;

        char recognized_number[2];
        while(connected)
        {
            if(flag_received_segment)
            {

```

```

        recognized_number[o]= 'o'+ num1;
        if(send(new_fd, &recognized_number[o],14, 0) == -1)
        {
            perror("send is error\n");
            close(new_fd);
            exit(0);
        }
        flag_received_segment= 0;
        printf("sent number from child board is:%d\n", num1);
    }
    usleep(100000); //wait for 0.1s
}

}

}

void ReceiveSegment(unsigned char* s)
{
    //Receive numbers from child boards: TCP protocol as client
    int sockfd;
    char buffer[1024];
    struct sockaddr_in server_addr;
    struct hostent *host;
    int portnumber,nbytes;
    int result;
    char result_char;

    if((host=gethostbyname("10.3.52.16"))==NULL)
    {
        printf("Gethostname error\n");
        //exit(1);
    }

    if((portnumber=atoi("2013"))<0)
    {
        printf("Port number error\n");
        //exit(1);
    }
    if((sockfd=socket(AF_INET,SOCK_STREAM,0))== -1)
    {
        printf("Socket Error:%s\n",strerror(errno));
        //exit(1);
    }
    bzero(&server_addr,sizeof(server_addr));
    server_addr.sin_family=AF_INET;
    server_addr.sin_port=htons(portnumber);
    server_addr.sin_addr=*((struct in_addr *)host->h_addr);
    if(connect(sockfd,(struct sockaddr *)&server_addr,
    sizeof(struct sockaddr))== -1)
    {
        printf("Connect Error:%s\n",strerror(errno));
        exit(1);
    }

    while(1)
    {
        if((nbytes=read(sockfd,buffer,MAXSIZE))== -1)
        {
            printf("Read Error:%s\n",strerror(errno));
            exit(1);
        }
        buffer[nbytes]='\0';

        /* recognition of the segment using SVM */
        //extract features from standards
        result= SVM_Recognition(buffer);
        result_char = result + 'o';
    }
}

```

```

        printf("I have received segment, recognized as:%s\n", result_char);
        flag_received_segment= 1;
        //Send recognized number to main thread by named pipes
        num1= result;
    }
    close(sockfd);
}

int main(void)
{
    //-----//
    printf("-----\n");
    printf("-----Plate segmentation and recognition-----\n");
    printf("-----Child Board 1 Program-----\n");
    printf("-----\n");
    printf("\n\n\n\n\n\n\n\n");
    //-----Declaration of variables-----//
    int check;//, flag= -1;
    //seg threads receive plate segments from the main board(as client)
    pthread_t  seg1_thread;
    pthread_t  seg2_thread;
    pthread_t  seg3_thread;
    pthread_t  seg4_thread;
    //num threads send recognized numbers to the main board(as server)
    pthread_t  num1_thread;
    pthread_t  num2_thread;
    pthread_t  num3_thread;
    pthread_t  num4_thread;

    num1=1; num2=2; num3=3; num4= 4;
    unsigned char* segment1;
    unsigned char* segment2;
    unsigned char* segment3;
    unsigned char* segment4;

    //Create the thread to read images from memory
    check = pthread_create(&seg1_thread, NULL, (void *)&ReceiveSegment, (void *)(segment1));
    if (check!=0) printf("seg1_thread error\n");
    /* check = pthread_create(&seg2_thread, NULL, (void *)&ReceiveSegment, (void *)(segment2));
    if (check!=0) printf("seg2_thread error\n");
    check = pthread_create(&seg3_thread, NULL, (void *)&ReceiveSegment, (void *)(segment3));
    if (check!=0) printf("seg3_thread error\n");

    check = pthread_create(&seg4_thread, NULL, (void *)&ReceiveSegment, (void *)(segment4));
    if (check!=0) printf("seg4_thread error\n");*/

    check = pthread_create(&num1_thread, NULL, (void *)&SendNumber1, (void *)&num1);
    if (check!=0) printf("num1_thread error\n");
    /*check = pthread_create(&num2_thread, NULL, (void *)&SendNumber2, (void *)&num2);
    if (check!=0) printf("num2_thread error\n");
    check = pthread_create(&num3_thread, NULL, (void *)&SendNumber3, (void *)&num3);
    if (check!=0) printf("num3_thread error\n");
    check = pthread_create(&num4_thread, NULL, (void *)&SendNumber4, (void *)&num4);
    if (check!=0) printf("num4_thread error\n");*/

    pthread_join(seg1_thread, NULL);
    pthread_join(num1_thread, NULL);
    /* pthread_join(num2_thread, NULL);
    pthread_join(num3_thread, NULL);
    pthread_join(num4_thread, NULL);*/

    return EXIT_SUCCESS;
}

/*

```

```
=====
Name : kernel module for main board.c
```

```
Author : Ning, Guanghan
```

```
Version :
```

```
Copyright : University of Missouri
```

```
Description : Hello World in C, Ansi-style
=====
```

```
*/
```

```
#define MODULE
```

```
#define __KERNEL__
```

```
#include <linux/module.h>
```

```
#include <linux/kernel.h>
```

```
#include <linux/time.h>
```

```
#include <asm/io.h>
```

```
#include <rtai.h>
```

```
#include <rtai_sched.h>
```

```
#include <rtai_fifos.h>
```

```
#include <unistd.h>
```

```
#include <stdio.h>
```

```
//#include <semaphore.h>
```

```
MODULE_LICENSE("GPL");
```

```
RTIME period; //check the button every 100ms.
```

```
static RT_TASK rtask; // rtask to check the button and derive GPSdata
```

```
typedef struct GPSDATA
```

```
{
```

```
    int pressed;
```

```
}GPSdata;
```

```
char msg[1024];
```

```
static void RT_task_func(int t)
```

```
{
```

```
    int n;
```

```
    char buffer[1024];
```

```
    int fifo_read, fifo_write;
```

```
    GPSdata data_from_kernel;
```

```
    //-----
```

```
    //Configure Bo button
```

```
    unsigned long *ptr;
```

```
    unsigned long *pdata;
```

```
    unsigned long *pddr;
```

```
    ptr = (unsigned long*)__ioremap(0x80840000,4096,0); //mapping
```

```
    pddr = ptr + 5;
```

```
data direction register
```

```
    pdata = ptr + 1;
```

```
register
```

```
    fifo_read= 0;
```

```
    fifo_write= 1;
```

```
    printk("RT task created. \n");
```

```
    while(1)
```

```
    {
```

```
        rt_task_wait_period(); //wait a period before executing next code
```

```
    //-----check button-----
```

```
    n= *pdata- 126 +96;// -128;
```

```
        if( n!= 0) //waiting for the button to be pressed
```

```
        {
```

```
            printk("n = %d\n", n);
```

```
            data_from_kernel.pressed= 1;
```

```
// point to
```

```
// point to data
```

```

        rtf_put(fifo_write, &data_from_kernel.pressed, sizeof(data_from_kernel.pressed));
        n= 0;    //stop putting flag to user space through fifo
    }
}

int init_module(void)
{
    GPSdata data_from_kernel;
    int fifo_read, fifo_write;
    fifo_read= 0;    //0 symbolizes fifo_in
    fifo_write= 1;    //1 symbolizes fifo_out
    rt_set_periodic_mode();    //make it periodic mode

    printk("Initiate module!\n");
    //Button configuration
    //Configure Bo button
    unsigned long *ptr;
    unsigned long *pdata;
    unsigned long *pddr;
    ptr = (unsigned long*)__ioremap(0x80840000,4096,0); //mapping
    pddr =ptr + 5;    // point to
data direction register
    pdata = ptr + 1;    // point to data
register
    *pddr |= 0x01;    // set Bo
as input
    *pdata &= 0x00;    //set initial value 1
    //Create FIFO
    rtf_create(fifo_read, 10*sizeof(msg)); //fifo 0 read
    rtf_create(fifo_write, 10*sizeof(data_from_kernel)); //fifo 1 write
    period = start_rt_timer(nano2count(1000000)); //100* 1ms =100ms

    //Create Tasks
    rt_task_init(&rttask, RT_task_func, 0, 256, 0, 0, 0);
    rt_task_make_periodic(&rttask, rt_get_time() + 100*period, 100*period);

    return 0;
}

void cleanup_module(void)
{
    printk("Clean up module!!!!\n");
    stop_rt_timer();
    rtf_destroy(0);    //destroy fifo_in
    rtf_destroy(1);    //destroy fifo_out
    rt_task_delete(&rttask);
}

```

```

//=====
//Name      : segment.h : The implementation of plate segmentation
//=====
#include <stdio.h>
#include <highgui.h>

#define MAX_NUM_CLUSTERS 6
#define MAX_VECTOR_SIZE 10
#define AVG_ERROR_THRESHOLD 1
#define MAX_KMEAN_DATA_SIZE 8000
#define MAX_COORDINATE_SIZE 8000
#define SHOWSEGMENT

typedef struct KMeanTag
{
    int ** kMeanClusterCenters;
    int ** kMeanClusterCentersBuf;
    int * kMeanClusterRatios;
    int ** kMeanDataBuf;
    unsigned char * dataLabel;

    int nKMeanClusters;
    //.....
    int **Coordinates;
    //.....
}KMeanType;
void kMeanInit(KMeanType * KM)
{
    int n, nKMeanClusters;

    KM->kMeanDataBuf = (int **)malloc(MAX_KMEAN_DATA_SIZE * sizeof(int *));
    for(n=0;n<MAX_KMEAN_DATA_SIZE; n++)
        KM->kMeanDataBuf[n] = (int *)malloc(MAX_VECTOR_SIZE * sizeof(int));

    KM->kMeanClusterCenters = (int **)malloc(MAX_NUM_CLUSTERS * sizeof(int *));
    for(n=0; n<MAX_NUM_CLUSTERS; n++)
        KM->kMeanClusterCenters[n] = (int *)malloc(MAX_VECTOR_SIZE * sizeof(int));

    KM->kMeanClusterCentersBuf = (int **)malloc(MAX_NUM_CLUSTERS * sizeof(int *));
    for(n=0; n<MAX_NUM_CLUSTERS; n++)
        KM->kMeanClusterCentersBuf[n] = (int *)malloc(MAX_VECTOR_SIZE * sizeof(int));

    KM->kMeanClusterRatios = (int *)malloc(MAX_NUM_CLUSTERS * sizeof(int));

    KM->dataLabel = (unsigned char *)malloc(MAX_KMEAN_DATA_SIZE);

    //.....
    KM->Coordinates = (int **)malloc(MAX_COORDINATE_SIZE * sizeof(int *));
    for(n=0;n<MAX_COORDINATE_SIZE; n++)
        KM->Coordinates[n] = (int *)malloc(MAX_VECTOR_SIZE * sizeof(int));
    //.....

    //KM->nKMeanClusters = 2;
    //if(nKMeanClusters > MAX_NUM_CLUSTERS) nKMeanClusters = MAX_NUM_CLUSTERS;
}
int kMeanClustering(int ** data, int nPoints, int nDim, int nClusters, KMeanType * KM)
{
    int stepLen, threshold;
    int n, No, N1, k, i, nIterations;
    int avg, tmp, diff, minDist, minPtr, ct, dist, contFlag;
    int ** tmpBuf;
    int flag;
    int ** kMeanClusterCenters;
    int ** kMeanClusterCentersBuf;

```



```

int * kMeanClusterRatios;
unsigned char * dataLabel;

if (nPoints < (4 * nClusters)) return -1;

kMeanClusterCenters = KM->kMeanClusterCenters;
kMeanClusterCentersBuf = KM->kMeanClusterCentersBuf;
kMeanClusterRatios = KM->kMeanClusterRatios;
dataLabel = KM->dataLabel;

//initial clusters;
stepLen = nPoints / nClusters;

for(n=0; n<nClusters; n++)
{
    No = n * stepLen;
    N1 = (n+1) * stepLen;

    if(n == (nClusters-1)) N1 = nPoints;

    if((N1 - No) < 1) N1 = No + 1;

    for(i=0; i<nDim; i++)
    {
        avg = 0;
        for(k=No; k<N1; k++)
            avg = avg + data[k][i];

        avg = avg / (N1 - No);
        kMeanClusterCenters[n][i] = avg;
    }
}

contFlag = 1;
nIterations = 0;

while(contFlag == 1)
{
    for(k=0; k<nClusters; k++)
    for(i=0; i<nDim; i++)
        kMeanClusterCentersBuf[k][i] = 0;

    for(k=0; k<nClusters; k++)
        kMeanClusterRatios[k] = 0;

    for(n=0; n<nPoints; n++)
    {
        minDist = 1000000000;
        for(k=0; k<nClusters; k++)
        {
            //find the distance to each cluster center
            tmp = 0;
            for(i=0; i<nDim; i++)
            {
                diff = data[n][i] - kMeanClusterCenters[k][i];
                tmp = tmp + diff * diff;
            }

            if(tmp < minDist)
            {
                minDist = tmp;
                minPtr = k;
            }
        }
        //assign the label
        dataLabel[n] = minPtr;
    }
}

```

```

        //assign this point to the minPtr cluster
        for(i=0;i<nDim; i++)
            kMeanClusterCentersBuf[minPtr][i] = kMeanClusterCentersBuf[minPtr][i] + data[n][i];
        kMeanClusterRatios[minPtr] = kMeanClusterRatios[minPtr] + 1;
    }

    //now update the new cluster center
    dist = 0;

    for(k=0; k<nClusters; k++)
    {
        ct = kMeanClusterRatios[k];

        if(ct > 2)
        {
            for(i=0;i<nDim; i++)
                kMeanClusterCentersBuf[k][i] = kMeanClusterCentersBuf[k][i] / ct;
        }
        else
        {
            //use the old one
            for(i=0;i<nDim; i++)
                kMeanClusterCentersBuf[k][i] = kMeanClusterCenters[k][i];
        }

        for(i=0; i<nDim; i++)
        {
            diff = kMeanClusterCenters[k][i] - kMeanClusterCentersBuf[k][i];
            if(diff < 0) diff = -diff;

            dist = dist + diff;
        }
    }

    //decide to loop again or not
    threshold = nClusters * nDim * AVG_ERROR_THRESHOLD;
    if((nIterations > 20) || (dist < threshold))
        contFlag = 0;

    //switch the center buffers
    tmpBuf = kMeanClusterCenters;
    kMeanClusterCenters = kMeanClusterCentersBuf;
    kMeanClusterCentersBuf = tmpBuf;

    nIterations++;
}

//output the percentage
for(k=0; k<nClusters; k++)
    kMeanClusterRatios[k] = (100 * kMeanClusterRatios[k])/nPoints;

return 1;
}

//END: K-Mean Clustering-----
void segments(unsigned char * frame, int widthStep, int Xo, int Yo, int X1, int Y1, KMeanType * KM, int *xx, int *ss) //Good One
{
    //Variables for part1
    int r, c, step, ct, n, thr;
    unsigned char * ptrFrame;
    int ** kMeanClusterCenters;
    int * kMeanClusterRatios;
    int ** kMeanDataBuf;
    int ** Coordinates;
    unsigned char * dataLabel;
    int nPoints;

```

```

    int num_zeros;

    //variables for part2
    int x, y, i, j, avg;
    int * histo = (int *)malloc((X1- X0) * sizeof(int));
    int max, final_x, final_s;
    long total;

    double q= 2.5; //a character is q times the length of a "."
    int s; //the scale of "." : the number of pixels a "." occupies
    int so= 0;
    int more= 0; //more pixels beyond the plate's edge on both sides
    int space= 2; //length of space
    int w_char; //width of a chracter: q*s
    int score[7];
    //-----
    //PART ONE: Make the plate binary. Foreground 1, background 0
    kMeanClusterCenters = KM->kMeanClusterCenters;
    kMeanClusterRatios = KM->kMeanClusterRatios;
    kMeanDataBuf = KM->kMeanDataBuf;
    Coordinates = KM->Coordinates;
    dataLabel = KM->dataLabel;

    thr = MAX_KMEAN_DATA_SIZE - 2;
    n = 0;
    for(r=Y0; r<Y1; r++)
    for(c=X0; c<X1; c++)
    {
        //sample the pixel
        ptrFrame = frame + r * widthStep + c * 3;
        for(i= 0; i<3; i++)
        {
            kMeanDataBuf[n][i] = ptrFrame[i];
        }
        Coordinates[n][0] = c;

        Coordinates[n][1] = r;

        n++;

        if(n > thr)
        {
            printf("Overflow\n");
            n = thr; //now overflow
        }
    }

    //do k-means to sepearate foreground from background
    nPoints= (X1-X0)*(Y1-Y0);
    kMeanClustering(kMeanDataBuf , nPoints , 3, 2, KM);

    //Separate the foreground from the background
    num_zeros= 0;
    for (n= 0; n< nPoints; n++)
    {
        if( dataLabel[n]== 0) num_zeros++; //number of foreground pixels
    }
    //if foreground is the plate , background is the plate characters, switch their clusters
    if (num_zeros > nPoints- num_zeros) //white character pixels should be smaller than blue plate pixels, if not, switch
    {
        for (n= 0; n< nPoints; n++)
        {
            dataLabel[n]= 1- dataLabel[n]; //0 becomes 1 and 1 becomes 0.
        }
    }
    //paint black on foreground and white on background
    for (n= 0; n< nPoints; n++)
    {

```

```

        ptrFrame= frame + Coordinates[n][1] * widthStep + Coordinates[n][0] * 3;
    for( i= 0; i<3; i++)
    {
        ptrFrame[i]= ( dataLabel[n]== 0 ? 0 : 255); //if a pixel belongs to foreground, then it's 0, painted black
    }
}

//Make the edges clear
for (x= X0, y= Y0; x< X1, y<Y1; x++, y++)
{
    //UP and bottom
    if ( (y< Y0+2)|| (y> Y1-2) )
    {
        ptrFrame= frame + y * widthStep + x * 3; //saves computation if put inside "if"
        ptrFrame[0]= ptrFrame[1]= ptrFrame[2]= 255;
    }
    //left and right
    else if ( (x< X0+2)|| (x> X1-2) )
    {
        ptrFrame= frame + y * widthStep + x * 3;
        ptrFrame[0]= ptrFrame[1]= ptrFrame[2]= 255;
    }
}
//-----//
//PART TWO: Segment based on the binaried image
//histo was all 0s at the beginning
for( x= 0; x< X1- X0; x++) histo[x]= 0;

for (n= 0; n< nPoints; n++)
{
    ptrFrame= frame + Coordinates[n][1] * widthStep + Coordinates[n][0] * 3;
    x= Coordinates[n][0]- X0;
    if (ptrFrame[0]== 0) histo[x]++; //accumulate character(black) pixels
}
//-----
//Don't do binarization here!!
//-----
max= 10;
final_x= X0;
final_s= 1;
for(i=0; i<7; i++) score[i]= 0;
total= 0;
//find optimum x and s
for ( x= X0- more; x<= X0+ int(q*(X1-X0+ more)/(7*q+1)); x++)
    for(s= 1; s<= int((X1- x + more)/(7*q+1)); s++)
    {
        s0= 0;
        for (i= 0; i<7; i++)
        {
            if(i==2) s0= s;
            for( n= x+ i*q*s + s0; n< x+ (i+1)*q*s + s0; n++)
            {
                if( (n< X1)&&(n>=X0) ) score[i]+= ( n< x+ (i+1)*q*s + s0 - space)? histo[n- X0]:((-1)* histo[n- X0]);
            }
        }
        total= score[0]+ score[6]+ (score[1]+score[2]+score[3]+score[4]+score[5]);
        if (total > max)
        {
            max= total;
            final_x= x;
            final_s= s;
        }
        total= 0;
        for(i=0; i< 7; i++) score[i]= 0;
    }
}
*xx= final_x;

```

```

        *ss= final_s;

        //Draw the segmenting lines
#ifdef SHOWSEGMENT
        for (y= Yo; y<Y1; y++)
        {
            for ( i= 0; i<8; i++)
            {
                x= final_x+ (i*q)*final_s;
                ptrFrame= frame + y * widthStep + x * 3;
                //if (i==1) //printf("ptrFrame is: (%d)\n", ptrFrame);
                if(i> 1)

                {
                    ptrFrame+= 3*final_s;
                }

                ptrFrame[0]= 255; ptrFrame[1]= ptrFrame[2]= 0;
                ptrFrame -= 3*space;
                ptrFrame[0]= 255; ptrFrame[1]= ptrFrame[2]= 0;
            }
            ptrFrame= frame + y * widthStep + int(final_x+ 2*q*final_s- space)* 3;
            ptrFrame[0]= 255; ptrFrame[1]= ptrFrame[2]= 0;
        }
#endif

    }
    void show_segment(unsigned char * frame, int widthStep, int Yo, int Y1, int *xx, int *ss )
    {
        int i, x, y, q=2.5, space= 2;
        int final_x= *xx;
        int final_s= *ss;
        printf("frame is: (%d)\n", frame);
        printf("(final_x, final_s) is: (%d, %d)\n", final_x, final_s);
        printf("(Yo, Y1) is: (%d, %d)\n", Yo, Y1);
        unsigned char * ptrFrame;
        //Draw the segmenting lines
        for (y= Yo; y<Y1; y++)
        {
            for ( i= 0; i<8; i++)
            {
                x= final_x+ (i*q)*final_s;
                ptrFrame= frame + y * widthStep + x * 3;
                if (i==1) printf("ptrFrame is: (%d)\n", ptrFrame);
                if(i> 1)

                {
                    ptrFrame+= 3*final_s;
                }

                ptrFrame[0]= 0; ptrFrame[1]= ptrFrame[2]= 255;
                ptrFrame -= 3*space;
                ptrFrame[0]= 0; ptrFrame[1]= ptrFrame[2]= 255;
            }
            ptrFrame= frame + y * widthStep + int(final_x+ 2*q*final_s- space)* 3;
            ptrFrame[0]= 255; ptrFrame[1]= ptrFrame[2]= 0;
        }
    }
}

```

```

/* Implementation of character recognition */

#include "features.h"
#include "svm_classifier_clean.h"
#include "chenLeeCV.h"

SVM_GST gst;
svm_classifier_clean<int,double> svm[NUMBER_OF_MODULES];

void configureSystem(SVM_GST * gst)
{
    gst->path = "../Release/";
    gst->imageListFileName = "testImages.txt";
}

void initSystem(SVM_GST * gst,svm_classifier_clean<int,double>*svm)
{
    int k;
    char modelFilePath[256];

    gst->sampleLable = (int *)malloc(MAX_SAMPLE_SIZE * sizeof(int));
    gst->classLable = (int *)malloc(MAX_SAMPLE_SIZE * sizeof(int));

    gst->feature = (double *)malloc(MAX_SAMPLE_SIZE * sizeof(double));

    gst->nClasses = 10; //7;

    /*Load Module*/
    for (k=0;k< NUMBER_OF_MODULES;k++)
    {
        sprintf(modelFilePath,"%smodel/00%d.mod",gst->path,k);
        svm[k].svm_init_clean(modelFilePath);
    }
}

void svmTest(double * feature, int featureSize, int n, float * scores, svm_classifier_clean<int,double> *svm)
{
    int temp = 6;
    svm[n].svm_classifier_clean(&temp,feature,scores,featureSize,1);
}

int svmTestCharacter(SVM_GST * gst,svm_classifier_clean<int,double> *svm, CImage *src, int k)
{
    char * path;
    char * tmpStr;
    int ct, lable, i, len, n;
    int * sampleLable, * classLable;
    double * feature;
    float score;
    int classFlag, classResult;
    int widthStep;
    widStep= (src->width)*(src->channels);

    sampleLable = gst->sampleLable;
    classLable = gst->classLable;
    path = gst->path;
    gst->featureSize = 96;

    feature = gst->feature;
    //-----//
    Combine_Features(feature, (unsigned char *)src->imageData, widStep, 0, 0, src->width, src->height);
    //-----//

    classResult = -1;
    svmTest(feature, gst->featureSize, 0, &score, svm);
    if(score > 0)
        classResult = 0;
    else
    {

```



```

//feature.c
/* features for character recognition */
#include <stdlib.h>
#include "Global.h"
#include "Define.h"
#include "features.h"

double* Feature_1(double *FH1, unsigned char * frame, int widthStep, int CXo, int CYo, int CX1, int CY1)
{
    int x, y, i;
    int x_step;
    int MID= 175;
    unsigned char *ptrFrame, *pre_pix;
    //double *FH1=(int *)malloc(16 * sizeof(double));

    for ( i= 0; i< 16; i++) FH1[i] = 0;
    x_step= (CX1- CXo)/16; //here add 1 so x_step is not 0
    for ( i= 0; i< 16; i++)
    {
        x= CXo + i*x_step;
        for ( y= CYo; y< CY1; y++)
        {
            ptrFrame = frame + y * widthStep + x * 3;
            pre_pix= ptrFrame- widthStep;
            if ((ptrFrame[0]< MID)&& (pre_pix[0]> MID)) {FH1[i]++;}
        }
    }
    return FH1;
}

double* Feature_2(double *FH2, unsigned char * frame, int widthStep, int CXo, int CYo, int CX1, int CY1)
{
    int x, y, i;
    int y_step;
    int MID= 175;
    unsigned char *ptrFrame, *pre_pix;

    for ( i= 0; i< 16; i++) FH2[i] = 0;
    y_step= (CY1- CYo)/16;
    for ( i= 0; i< 16; i++)
    {
        y= CYo + i*y_step;
        for ( x= CXo; x< CX1; x++)
        {
            ptrFrame = frame + y * widthStep + x * 3;
            pre_pix= ptrFrame- 1;
            if ((ptrFrame[0]< MID)&& (pre_pix[0]> MID)) FH2[i]++;
        }
    }
    return FH2;
}

double* Feature_3(double *FH3, unsigned char * frame, int widthStep, int CXo, int CYo, int CX1, int CY1)
{
    int x, y, i, ct1[16], ct2[16];
    int x_step;
    int MID= 175;
    unsigned char *ptrFrame;

    for ( i= 0; i< 16; i++)
    {
        ct1[i] = 0;
        ct2[i] = 0;
        FH3[i] = 0;
    }
    x_step= (CX1- CXo)/16 ;
    for ( i= 0; i< 16; i++)

```



```

    {
        x= CX0 + i*x_step;
        for ( y= CY0; y< CY1; y++)
        {
            ptrFrame = frame + y * widthStep + x * 3;
            if (ptrFrame[0]< MID) ct1[i]++;
            if (ptrFrame[0]> MID) ct2[i]++;
        }
        FH3[i]= 10*ct1[i]/(ct1[i]+ ct2[i]+0.00001);
    }
    return FH3;
}
double* Feature_4(double *FH4, unsigned char * frame, int widthStep, int CX0, int CY0, int CX1, int CY1)
{
    int x, y, i, ct1[16], ct2[16];
    int y_step;
    int MID= 175;
    unsigned char *ptrFrame, *pre_pix;

    for ( i= 0; i< 16; i++)
    {
        ct1[i] = 0;
        ct2[i] = 0;
        FH4[i] = 0;
    }
    y_step= (CY1- CY0)/16;

    for ( i= 0; i< 16; i++)
    {
        y= CY0 + i*y_step;
        for ( x= CX0; x< CX1; x++)
        {
            ptrFrame = frame + y * widthStep + x * 3;
            if (ptrFrame[0]<MID )
            {
                ct1[i]++;
            }
            if (ptrFrame[0]>MID)
            {
                ct2[i]++;
            }
        }
        FH4[i]= 10* ct1[i]/(ct1[i]+ ct2[i]+ 0.00001);
    }
    return FH4;
}
double* Feature_raw(double *FH_raw, unsigned char * frame, int widthStep, int CX0, int CY0, int CX1, int CY1)
{
    int x, y, i, j;
    int x_step, y_step;
    unsigned char *ptrFrame, *ptrFrame2, *ptrFrame3, *ptrFrame4;

    for ( i= 0; i< 32; i++) FH_raw[i] = 0;
    x_step= y_step= 2;

    for ( i= 0; i< 4; i++)
    for ( j= 0; j< 8; j++)
    {
        x= CX0 + i*x_step; // +2i
        y= CY0 + j*y_step; // +2j

        ptrFrame = frame + y * widthStep + x * 3;
        ptrFrame2 = frame + y * widthStep + (x+1) * 3;
        ptrFrame3 = frame + (y+1) * widthStep + x * 3;
        ptrFrame4 = frame + (y+1) * widthStep + (x+1) * 3;
    }
}

```

```

        //FH_raw[i+j*4]= (ptrFrame[0]+ ptrFrame2[0]+ ptrFrame3[0]+ ptrFrame4[0])/4;
        FH_raw[i+j*4]= (ptrFrame[0]+ ptrFrame2[0]+ ptrFrame3[0]+ ptrFrame4[0]+ptrFrame[1]+ ptrFrame2[1]+
ptrFrame3[1]+ ptrFrame4[1]+ptrFrame[2]+ ptrFrame2[2]+ ptrFrame3[2]+ ptrFrame4[2])/12;
        //Weight of this feature
        FH_raw[i+j*4]/= 200; //94%
    }
    return FH_raw;
}

```

```

void Combine_Features( double* Feature, unsigned char * frame, int widthStep, int CXo, int CYo, int CX1, int CY1)
{
    int n, i;
    n= 0;

    double *FH1, *FH2, *FH3, *FH4, *FH_raw;

    FH1= (double *)malloc( 16 * sizeof(double));
    FH2= (double *)malloc( 16 * sizeof(double));
    FH3= (double *)malloc( 16 * sizeof(double));
    FH4= (double *)malloc( 16 * sizeof(double));
    FH_raw= (double *)malloc( 32 * sizeof(double));

    FH1= Feature_1(FH1, frame, widthStep, CXo, CYo, CX1, CY1);
    FH2= Feature_2(FH2, frame, widthStep, CXo, CYo, CX1, CY1);
    FH3= Feature_3(FH3, frame, widthStep, CXo, CYo, CX1, CY1);
    FH4= Feature_4(FH4, frame, widthStep, CXo, CYo, CX1, CY1);
    FH_raw= Feature_raw(FH_raw, frame, widthStep, CXo, CYo, CX1, CY1);

    for (i= 0; i< 16; i++)
    {
        Feature[n*16+ i]= FH1[i];
    }
    n++;
    for (i= 0; i< 16; i++)
    {
        Feature[n*16+ i]= FH2[i];
    }
    n++;
    for (i= 0; i< 16; i++)
    {
        Feature[n*16+ i]= FH3[i];
    }
    n++;
    for (i= 0; i< 16; i++)
    {
        Feature[n*16+ i]= FH4[i];
    }
    n++;
    for (i= 0; i< 32; i++)
    {
        Feature[n*16+ i]= FH_raw[i];
    }
    return;
}

```

```

//chenLeeCV.h
/* Open source code */
/*not original work*/
/*implements some basic computer vision functions for interface*/

#ifndef CHENLEECV_H
#define CHENLEECV_H

typedef struct CvRect
{
    int x;
    int y;
    int width;
    int height;
}

typedef struct
{
    //unsigned short  bfType;
    unsigned long   bfSize;
    unsigned short  bfReserved1;
    unsigned short  bfReserved2;
    unsigned long   bfOffBits;
} CBitmapFileHeader;

typedef struct
{
    unsigned long  biSize;
    long  biWidth;
    long  biHeight;
    unsigned short  biPlanes;
    unsigned short  biBitCount;
    unsigned long  biCompression;
    unsigned long  biSizeImage;
    long  biXPelsPerMeter;
    long  biYPelsPerMeter;
    unsigned long  biClrUsed;
    unsigned long  biClrImportant;
} CBitmapInfoHeader;

typedef struct
{
    unsigned char rgbBlue;
    unsigned char rgbGreen;
    unsigned char rgbRed;
    unsigned char rgbReserved;
} CRGBQuad;

typedef struct
{
    int width;
    int height;
    int channels;
    unsigned char* imageData;
} CImage;

CImage* cLoadImage(char* path);
bool cSaveImage(char* path, CImage* bmpImg);

#endif

```



```

//printf("This file contains palette, it's not real colormap\n\n");
channels = 1;
width = bmpInfoHeader.biWidth;
height = bmpInfoHeader.biHeight;
offset = (channels*width)%4;
if (offset != 0)
{
    offset = 4 - offset;
}
//bmpImg->mat = kzCreateMat(height, width, 1, 0);
bmpImg->width = width;
bmpImg->height = height;
bmpImg->channels = 1;
bmpImg->imageData = (unsigned char*)malloc(sizeof(unsigned char)*width*height);
step = channels*width;

quad = (CIRgbQuad*)malloc(sizeof(CIRgbQuad)*256);
fread(quad, sizeof(CIRgbQuad), 256, pFile);
free(quad);

for (i=0; i<height; i++)
{
    for (j=0; j<width; j++)
    {
        fread(&pixVal, sizeof(unsigned char), 1, pFile);
        bmpImg->imageData[(height-1-i)*step+j] = pixVal;
    }
    if (offset != 0)
    {
        for (j=0; j<offset; j++)
        {
            fread(&pixVal, sizeof(unsigned char), 1, pFile);
        }
    }
}
}
else if (bmpInfoHeader.biBitCount == 24)
{
    //printf("Real colormap\n\n");
    channels = 3;
    width = bmpInfoHeader.biWidth;
    height = bmpInfoHeader.biHeight;

    bmpImg->width = width;
    bmpImg->height = height;
    bmpImg->channels = 3;
    bmpImg->imageData = (unsigned char*)malloc(sizeof(unsigned char)*width*3*height);
    step = channels*width;

    offset = (channels*width)%4;
    if (offset != 0)
    {
        offset = 4 - offset;
    }

    for (i=0; i<height; i++)
    {
        for (j=0; j<width; j++)
        {
            for (k=0; k<3; k++)
            {
                fread(&pixVal, sizeof(unsigned char), 1, pFile);
                bmpImg->imageData[(height-1-i)*step+j*3+k] = pixVal; //bmp file starts from low-left, now matrix becomes starting from
high-left;)
            }
            //kzSetMat(bmpImg->mat, height-1-i, j, kzScalar(pixVal[0], pixVal[1], pixVal[2]));
        }
    }
}

```

```

    }
    if (offset != 0)
    {
        for (j=0; j<offset; j++)
        {
            fread(&pixVal, sizeof(unsigned char), 1, pFile);
        }
    }
}
}
}

return bmpImg;
}

int clSaveImage(char* path, CImage* bmpImg)
{
    FILE *pFile;
    unsigned short fileType;
    CBitMapFileHeader bmpFileHeader;
    CBitMapInfoHeader bmpInfoHeader;
    int step;
    int offset;
    unsigned char pixVal = '\0';
    int i, j;
    CArgbQuad* quad;

    pFile = fopen(path, "wb");
    if (!pFile)
    {
        return 0;
    }

    fileType = 0x4D42;
    fwrite(&fileType, sizeof(unsigned short), 1, pFile);

    if (bmpImg->channels == 3)//24bits, 3 channels, colormap
    {
        step = bmpImg->channels*bmpImg->width;
        offset = step%4;
        if (offset != 4)
        {
            step += 4-offset;
        }

        bmpFileHeader.bfSize = bmpImg->height*step + 54;
        bmpFileHeader.bfReserved1 = 0;
        bmpFileHeader.bfReserved2 = 0;
        bmpFileHeader.bfOffBits = 54;
        fwrite(&bmpFileHeader, sizeof(CBitMapFileHeader), 1, pFile);

        bmpInfoHeader.biSize = 40;
        bmpInfoHeader.biWidth = bmpImg->width;
        bmpInfoHeader.biHeight = bmpImg->height;
        bmpInfoHeader.biPlanes = 1;
        bmpInfoHeader.biBitCount = 24;
        bmpInfoHeader.biCompression = 0;
        bmpInfoHeader.biSizeImage = bmpImg->height*step;
        bmpInfoHeader.biXPelsPerMeter = 0;
        bmpInfoHeader.biYPelsPerMeter = 0;
        bmpInfoHeader.biClrUsed = 0;
        bmpInfoHeader.biClrImportant = 0;
        fwrite(&bmpInfoHeader, sizeof(CBitMapInfoHeader), 1, pFile);

        for (i=bmpImg->height-1; i>=0; i--)

```

```

{
    for (j=0; j<bmplmg->width; j++)
    {
        pixVal = bmplmg->imageData[i*bmplmg->width*3+j*3];
        fwrite(&pixVal, sizeof(unsigned char), 1, pFile);
        pixVal = bmplmg->imageData[i*bmplmg->width*3+j*3+1];
        fwrite(&pixVal, sizeof(unsigned char), 1, pFile);
        pixVal = bmplmg->imageData[i*bmplmg->width*3+j*3+2];
        fwrite(&pixVal, sizeof(unsigned char), 1, pFile);
    }
    if (offset!=0)
    {
        for (j=0; j<offset; j++)
        {
            pixVal = 0;
            fwrite(&pixVal, sizeof(unsigned char), 1, pFile);
        }
    }
}
}
else if (bmplmg->channels == 1)//8 bit, single channel, greyscale
{
    step = bmplmg->width;
    offset = step%4;
    if (offset != 4)
    {
        step += 4 - offset;
    }

    bmpFileHeader.bfSize = 54 + 256*4 + bmplmg->width;
    bmpFileHeader.bfReserved1 = 0;
    bmpFileHeader.bfReserved2 = 0;
    bmpFileHeader.bfOffBits = 54 + 256*4;
    fwrite(&bmpFileHeader, sizeof(ClBitMapFileHeader), 1, pFile);

    bmpInfoHeader.biSize = 40;
    bmpInfoHeader.biWidth = bmplmg->width;
    bmpInfoHeader.biHeight = bmplmg->height;
    bmpInfoHeader.biPlanes = 1;
    bmpInfoHeader.biBitCount = 8;
    bmpInfoHeader.biCompression = 0;
    bmpInfoHeader.biSizeImage = bmplmg->height*step;
    bmpInfoHeader.biXPelsPerMeter = 0;
    bmpInfoHeader.biYPelsPerMeter = 0;
    bmpInfoHeader.biClrUsed = 256;
    bmpInfoHeader.biClrImportant = 256;
    fwrite(&bmpInfoHeader, sizeof(ClBitMapInfoHeader), 1, pFile);

    quad = (CIRgbQuad*)malloc(sizeof(CIRgbQuad)*256);
    for (i=0; i<256; i++)
    {
        quad[i].rgbBlue = i;
        quad[i].rgbGreen = i;
        quad[i].rgbRed = i;
        quad[i].rgbReserved = 0;
    }
    fwrite(quad, sizeof(CIRgbQuad), 256, pFile);
    free(quad);

    for (i=bmplmg->height-1; i>=0; i--)
    {
        for (j=0; j<bmplmg->width; j++)
        {
            pixVal = bmplmg->imageData[i*bmplmg->width+j];
            fwrite(&pixVal, sizeof(unsigned char), 1, pFile);
        }
    }
}

```

```
    if (offset!=0)
    {
        for (j=0; j<offset; j++)
        {
            pixVal = 0;
            fwrite(&pixVal, sizeof(unsigned char), 1, pFile);
        }
    }
}
fclose(pFile);

return 1;
}
```