天亮说晚安 计算机网络项目简介 第一周（一）

核心功能

- 读入任意文件，转换为二进制文件。
- 编码为图形码，并生成可播放的视频。
- 信息接收者通过手机录制的视频，重新解码为文件，并显示在屏幕上。

核心技术

- OpenCV——图像生成，生成视频，解析视频，解码。
- Qt——GUI界面，项目构架。

核心代码说明

文件读入模块

```
//@ char* pathC; 保存了文件路径
FILE* f = fopen(pathC, "rb"); //打开文件
char ch;
fseek(f, 0, SEEK_END); //定位到文件末
int nFileLen = ftell(f);//获取文件长度
fseek(f,0,SEEK_SET);//定位到文件头
for(int i=0;i<nFileLen;i++)
{
   ch=fgetc(f);
   //@ std::vector<char> fileReadin;
   fileReadin.push_back(ch); //将字符存入ch末尾
}
fclose(f);//关闭文件
```

为了读取任意类型的文件，我们没有用`feof()`来判断文件尾。在后续项目中，还会出现读取一“帧”数据的情形，使用for循环确定`std::vector<char> fileReadin`的长度是个更好的选择。

编码视频**生成模块**

```
int fourcc = cv::VideoWriter::fourcc('a','v','c','1');//文件编码格式h.264的一种
std::string fileName=outputPath+"/vid_output.mp4";
vid_out=cv::VideoWriter(fileName.toStdString(),fourcc, 24, FRAME, 0);//视频输出流
```

视频的写入用到的是OpenCV的`cv::VideoWriter`，该类是依靠开源库FFMPEG实现的。使用该类时，需要确定写入视频的路径，编码格式，帧率，尺寸和色彩。`fourcc`定义了写入视频的编码为`AVC1`，它是`h.264`编码的一种，并封装为`.mp4`格式，确保所有主流播放器都可以正常的打开和播放。

```
videoFrame=std::vector<cv::Mat>(fileReadin.size());
for(int j=1;j<=fileReadin.size();j++){
   //...增加定位符，写入videoFrame[i]
   ......
   for (int i = 0; i < 8; i++) {
       if (c & 0x01) {
           //...增加图形码，写入videoFrame[i]
       ......
       }
       c >>= 1;
   }
   //写入视频文件
   vid_out<<videoFrame[j];
   for(int d=0;d<WHITEFRAMES;d++)
       //插入空白帧
       vid_out<<videoFrame[0];
   }
   //释放视频文件
   vid_out.release();
```

天亮说晚安 计算机网络项目简介 第一周（二）

视频**解**码**模**块

```
//生成遮罩
std::vector<cv::Mat> mask(8);
for(int i = 0; i < 8; i++) {
   mask[i] = ...
}
```

目前，项目读取编码的手段是将图像根据定位点将图像尺寸还原，用遮罩捕获目标区域，所以需要先初始化`std::vector<mask>`。

```
std::vector<cv::Mat>video_pro;
for (int k = 0; k < frames ; k++) {
   //彩色转灰度
   cv::cvtColor(src, src, cv::COLOR_RGB2GRAY);

   //阈值
   int SRC_THRESH_LOW = this->THRESH;
   cv::threshold(src, src, SRC_THRESH_LOW, 255, cv::THRESH_BINARY);

   //Canny边缘检测
   cv::Mat src_gray = src;
   cv::Mat canny_output;
   cv::Canny(src_gray, canny_output, 200, 255);
```

```
    //轮廓提取
    std::vector<std::vector<cv::Point> > contours;
    std::vector<cv::Vec4i> hierarchy;
    cv::findContours(canny_output, contours, hierarchy, cv::RETR_EXTERNAL,
cv::CHAIN_APPROX_SIMPLE);

    if(contours.size()<1)//若提取失败，跳过此帧
        continue;

    //提取最大矩形，即屏幕现实范围
    cv::Rect poly_rect_max=cv::boundingRect(contours[0]);
    for (int i = 1; i < contours.size(); i++) {
         if(poly_rect_max.area()<cv::boundingRect(contours[i]).area())
              poly_rect_max=cv::boundingRect(contours[i]);
    }

    //ROI坐标生成 tl=topleft,br=bottomright
    int tl_x =  poly_rect_max.tl().x;
    int br_x =  poly_rect_max.br().x;
    int tl_y =  poly_rect_max.tl().y;
    int br_y =  poly_rect_max.br().y;

    //ROI区域生成
    cv::Rect rect(tl_x, tl_y, br_x - tl_x, br_y - tl_y);
    cv::Mat srcRoi=cv::Mat::zeros(src.size(), CV_8UC1);
    srcRoi=src(rect);
    cv::resize(srcRoi, srcRoi, RSFRAME);

    //存入预降噪vector保存
    video_pro.push_back(srcRoi);
}
```

这一步是降噪得以实现的核心，通过电脑屏幕底色的白色，将目标区域初步的提取出来，降低外部的噪声。`cv::cvtColor()`、`cv::threshold()`实现初步的噪声去除，`cv::Canny()`、`cv::findContours`提取出ROI区域和相应的轮廓，将轮廓外的区域全部消除。此步本应该为可选项，但是该段逻辑在`Release 1.0`的版本本是被强制执行的，会出现用户很精准的没有录入噪声，却读取失败的情况，`Beta 2.0`版本中拟予以优化。

```
std::vector<cv::Mat> video_diff;
for(int i=1;i<video_pro.size();i++){
    //差值
    cv::Mat st=video_pro[i]-video_pro[i-1];

    //形态学开
    cv::Mat kernelsrc = cv::getStructuringElement(cv::MORPH_RECT,
cv::Size(23, 23), cv::Point(-1, -1));
    cv::morphologyEx(st, st, cv::MORPH_OPEN, kernelsrc);

  //存入差值vector
```

```
    video_diff.push_back(st);
}
```

使用帧差法有两个目的，第一个是进一步去除噪声。第二目的是捕获关键帧。这里的关键帧是指清晰成像的，带有信息的图像。实际帧之间做差时，有三种情况：

- [i-1]帧为空白帧，[i]帧为关键帧（无论清晰与否，下同）——这时，由于减去白色的缘故，[i]帧信息全部丢失。
- [i-1]帧为关键帧，[i]也为关键帧——做差后，[i]会删去大部分信息，留下一些无用的噪点。
- [i-1]帧为关键帧，[i]空白帧——这时，空白帧[i]会因为剪掉了信息区域外的黑色而保留信息区域。

这个时候，理论上，第一种情况产生黑色空白帧直接被忽略，第二种情况在形态学开操作cv::morphologyEx()的处理下，退化为黑色空白帧，只有第三种情况产生的帧会被捕获。但在实际解算时，由于灰阶响应的缘故，阈值后的信息会出现不完整的情况。这种情况会直接导致程序不正常退出或者OpenCV断言错误，实际UI显示为卡66%的进度。 Relese 1.0版本中这个现象尤为严重，因此为Relese 2.0版本发布前的重中之重。

```
for(int i=0;i<video_diff.size();i++){
    //过滤低信息图片
    if(cv::countNonZero(video_diff[i])==0)
        continue;
    cv::Mat read=video_diff[i];
    //Canny边缘查找
    cv::Mat src_gray = read;
    cv::Mat canny_output;
    cv::Canny(src_gray, canny_output, 200, 255);

    //轮廓查找
    std::vector<std::vector<cv::Point> > contours;
    std::vector<cv::Vec4i> hierarchy;
    cv::findContours(canny_output, contours, hierarchy, cv::RETR_EXTERNAL,
cv::CHAIN_APPROX_SIMPLE);
    if(contours.size()<=2)
        continue;
    //从轮廓生成最小矩形
    std::vector<cv::Rect>poly_rect(contours.size());
    for (int i = 0; i < contours.size(); i++) {
       poly_rect[i] = cv::boundingRect(contours[i]);
    }
    //矩形轮廓x坐标整理
    for (int i = poly_rect.size(); i > 0; i--) {
        for (int j = 0; j < i - 1; j++) {
            if (poly_rect[j].tl().x > poly_rect[j + 1].tl().x)
                swap(poly_rect[j], poly_rect[j + 1]);
        }
    }
    //矩形轮廓y坐标整理
    for (int i = 0; i + 1 < poly_rect.size(); i += 2) {
        if (poly_rect[i].tl().y > poly_rect[i + 1].tl().y)
             swap(poly_rect[i], poly_rect[i + 1]);
```

```
    }

    //ROI坐标生成 tl=topleft,br=bottomright
    int tl_x = poly_rect[0].tl().x;
    int br_x = poly_rect[poly_rect.size() - 1].br().x;
    int tl_y = poly_rect[0].tl().y;
    int br_y = poly_rect[poly_rect.size() - 1].br().y;
    if(tl_x&&br_x&&tl_y&&br_y==0)
        continue;

    //矩形轮廓重建
    cv::Mat rscrc = cv::Mat::zeros(canny_output.size(), CV_8UC1);
    for (int i = 0; i < poly_rect.size(); i++) {
        cv::rectangle(rscrc, poly_rect[i], WHITE, cv::FILLED);
    }

    //ROI区域生成
    cv::Rect rect(tl_x, tl_y, br_x - tl_x, br_y - tl_y);
    cv::Mat srcRoi=cv::Mat::zeros(rscrc.size(), CV_8UC1);
    srcRoi=rscrc(rect);
    cv::resize(srcRoi, srcRoi, RSFRAME);

    //解码
    char c=0;
    for (int i = 7; i >= 0; i--) {
        //缓存帧
        cv::Mat p = cv::Mat::zeros(RSFRAME, CV_8UC1);

        //添加遮罩
        p = srcRoi-mask[i];

        //译码
        if (cv::countNonZero(p) >500)
            c |= 0x01;
        if (i != 0)
            c <<= 1;
    }
}
```

最后解码阶段，相当于第一步的复现，没有新的内容，最后的循环实现了解码，相当于编码的逆过程。