Thông tin đào tạo

Thông báo
Lịch khai giảng
Đăng ký học

Nội dung mới cập nhật

Thông báo tạm dừng các khóa đào tạo lập trình nhúng

USB Device Drivers, Phần 1. USB driver trên Linux (1)
USB Device Drivers, Phần 2. USB driver trên Linux (2)
USB Device Drivers, Phần 3. Trao đổi dữ liệu với thiết bị USB

Device Drivers, Phần 1. Cơ bản về driver trên Linux
Device Drivers, Phần 2. Viết driver đầu tiên trên Linux
Device Drivers, Phần 3. Lập trình C ở tầng nhân
Device Drivers, Phần 4. Linux Character Drivers
Device Drivers, Phần 5. Character Device Files
Device Drivers, Phần 6. Các thao tác đối với file thiết bị

Lập trình mạng > Xử lý ảnh OpenCV >

Thu nhận ảnh từ camera sử dụng OpenCV

Thu nhận dữ liệu từ camera là thao tác cơ bản đầu tiên để cung cấp dữ liệu ảnh đầu vào cho các ứng dụng xử lý ảnh (hoặc các ứng dụng khác) muốn làm việc với luồng dữ liệu video từ thiết bị camera. Bài viết này minh họa cách thức sử dụng một số hàm cơ bản cung cấp bởi thư viện OpenCV để lấy (capture) dữ liệu từ camera (usb camera, usb webcam).

Ý tưởng chính là:

- Sử dụng hàm cvCaptureFromCAM() để mở camera

- Khởi động một bộ định thời (timer) với khoảng thời gian định thời (interval) thích hợp (với số frame/s muốn lấy) để tiến hành lấy từng khung hình (frame) dữ liệu từ thiết bị camera. Mỗi sự kiện định thời xảy ra sẽ thực hiện truy vấn một khung hình sử dụng hàm cvQueryFrame()

- Frame lấy được sẽ được convert sang dạng biểu diễn phù hợp (trong tình huống này là QImage trong ứng dụng Qt) và hiển thị lên điều khiển phù hợp (QLabel, QWidgets, …)

Để tiện lợi cho việc phát triển các ứng dụng xử lý ảnh tiếp theo sau khi thu nhận, chúng ta sẽ viết một lớp chuyên biệt để làm việc với thiết bị camera.

Đoạn mã sau triển khai 1 lớp ImageCapture

File ImageCapture.h

```
#ifndef IMAGECAPTURE_H
```

Trang web được xây dựng bởi

Embedded247
Bộ môn Kỹ thuật máy tính,
Viện Công nghệ Thông tin và Truyền thông,
Đại học Bách Khoa Hà Nội
Phòng 502, B1, ĐH BKHN
Tel: 04-38696125


```
#define IMAGECAPTURE_H

#include <QObject>

#include <QImage>

#include <QTimer>

#include <cv.h>

#include <highgui.h>


class ImageCapture : public QObject

{

    Q_OBJECT

public:

    explicit ImageCapture(QObject *parent = 0);

    ~ImageCapture();

 signals:

 public slots:

    //!< attempts to start camera, camera number is given in cameraIndex (see cvCaptureFromCAM docs)

    void captureFromCamera( int cameraIndex = -1 );     //index = -1 to capture form any camera

    void stopCapture();          //!< stops timer and release OpenCV capture structure

protected:

    QImage convert(IplImage * image);     //!< create QImage from OpenCV image data

    QImage IplImage2QImage(const IplImage *iplImage);

protected slots:

    void timer_stick(); //!< grab and send frame on each timer tick

signals:
```

```
        void imageCaptured( const QImage& image );

        void error( const QString& text );

protected:

        QTimer * _timer;          //!< timeout signal is used to capture frames at regular intervals (or whenever possible in case of camera)

        CvCapture * _cvCap;  //!< used to grab image data from video or camera

        IplImage* imagerd;

};

#endif // IMAGECAPTURE_H
```

File ImageCapture.cpp

```
#include "imagecapture.h"


ImageCapture::ImageCapture(QObject *parent) :
    QObject(parent)
{
    _timer = NULL;
    _cvCap = NULL;
    imagerd = NULL;
    this->_timer = new QTimer(this);
    connect(_timer, SIGNAL(timeout()), this, SLOT(timer_stick()));
}
ImageCapture::~ImageCapture()
{
    this->stopCapture();
```

```
}
void ImageCapture::captureFromCamera(int cameraIndex)
{
    this->stopCapture(); //Stop if camera opened
    _cvCap = cvCaptureFromCAM(cameraIndex);
    if(_cvCap)
    {
        this->_timer->setInterval(10);
        this->_timer->start();
    }
    else   //error if this function returns NULL
    {
        emit error(tr("Invalid camera index: %1").arg(cameraIndex));
    }
}
void ImageCapture::stopCapture()
{
    this->_timer->stop();
    cvReleaseCapture(&_cvCap);
}
void ImageCapture::timer_stick()
{
    IplImage *image = cvQueryFrame(this->_cvCap);
    if(image)
```

```
    {
        //emit imageCaptured(convert(image));
        emit imageCaptured(IplImage2QImage(image));
    }
}


//
// here we will convert OpenCV image to QImage, copying the pixels row-by-row
// OpenCV stores the pixel data in row-major order, in BGR format
// QImage expects RGB, so we need to swap pixel valules
//
// for the sake of simplicity, only BGR24 images are handled (3 channels, 8-bit each)
//
// easier way is to pass image->imageData buffer directly to QImage constructor
// and then use QImage::rgbSwapped(), but its good to know how to manipulate pixel
// values directly
//
QImage ImageCapture::convert(IplImage *image)
{
    QImage result;
    if(image)
    {
        uchar *buffer = NULL;
        QImage::Format format = QImage::Format_Invalid;
```

```
        if(image->depth==IPL_DEPTH_8U && image->nChannels==3)
        {
            //prepare image data buffer
            //we need enough space for WxH pixels, each pixel have 3 'uchar' values (R, G, B)
            buffer = (uchar*)malloc(image->width*image->height*3*sizeof(uchar));
            format = QImage::Format_RGB888;
            if(!buffer)
            {
                emit error(tr("Insufficient memory for image buffer!"));
                return result;
            }
            // copy image data row by row
            // image data layout is like this (w=image width, b(i) - blue component of i-th pixel in row, g - green, r -
red):
            // row 0: [ b(0), g(0), r(0), b(1), g(1), r(1), ..., b(w-1), g(w-1), r(w-1) ]
            // row 1: [ b(w), g(w), r(w), b(w+1), g(w+1), r(w+2), ... ]
            // row 2: [ b(2*w), ... ]
            // then each row is placed one after another (h=image height):
            // image->imageData = [ row0, row1, row2, ..., row(h-1) ]
            for( int y = 0; y < image->height; ++y ){
                    // each row pixels data starts at row*width position (if flipped, we count rows from h-1 to 0)
                    const int srcRow = 1 ?  y*image->width : (image->height-y-1)*image->width;
                    const int dstRow = y*image->width;
                    for( int x = 0; x < image->width; ++x ){
                            // multiply by 3, because each pixel occupies 3 entries in array
```

```
                    const int srcPixel = (x+srcRow)*3;
                    const int dstPixel = (x+dstRow)*3;
                    // copy pixel value, swap r<->b channels
                    buffer[ dstPixel + 0 ] = image->imageData[ srcPixel + 2 ];  // red
                    buffer[ dstPixel + 1 ] = image->imageData[ srcPixel + 1 ];  // green
                    buffer[ dstPixel + 2 ] = image->imageData[ srcPixel + 0 ];  // blue
                }
            }
        }
        else{
            emit error( tr("Format not supported: depth=%1, channels=%2").arg(image->depth).arg(image->nChannels));
            return result;
        }
        if( buffer ){
            // when creating QImage from buffer data, we need to make sure that
            // buffer remains valid until image is deleted, so we make explicit copy
            // and free the buffer immediately
            QImage * tmp = new QImage(buffer, image->width, image->height, format);
            result = tmp->copy();
            delete tmp;  free(buffer);
        }
    }
    else{
        emit error(tr("Image pointer is NULL"));
```

```
    }
    return result;
}
QImage ImageCapture::IplImage2QImage(const IplImage *iplImage)
{
        int height = iplImage->height;
        int width = iplImage->width;

        const uchar *qImageBuffer =(const uchar*)iplImage->imageData;
        QImage img(qImageBuffer, width, height, QImage::Format_RGB888);
        return img.rgbSwapped();
}
```

Sử dụng lớp trên trong lớp giao diện chính MainWindow của ứng dụng

File MainWindow.h

```
#include <QMainWindow>
#include <imagecapture.h>
namespace Ui {
    class MainWindow;
}
class MainWindow : public QMainWindow
{
    Q_OBJECT
```

```
public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
private:
    Ui::MainWindow *ui;
public slots:
    void on_actionOpen_Camera_triggered();
    void displayImage(const QImage &image);
protected slots:
    void captureError(const QString &text);
public:
    ImageCapture *_capture;
};
#endif // MAINWINDOW_H
```

File MainWindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QMessageBox>
#include "stdio.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
```

```
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    this->_capture = new ImageCapture(this);
    connect(ui->menuFile, SIGNAL(triggered(QAction*)), this, SLOT(on_actionOpen_Camera_triggered()));
    connect(_capture, SIGNAL(imageCaptured(const QImage&)), this, SLOT(displayImage(const QImage&)));
    connect(_capture, SIGNAL(error(const QString&)), this, SLOT(captureError(const QString&)));
}

MainWindow::~MainWindow()
{
    delete _capture;
    delete ui;
}
void MainWindow::on_actionOpen_Camera_triggered()
{
    QMessageBox msg;
    //msg.setText("Hello");
    //msg.exec();

    this->_capture->captureFromCamera(-1);
    if(!this->_capture)
        printf("No camera detected");
```

```
}
void MainWindow::displayImage(const QImage &image)
{
    const QPixmap pix = QPixmap::fromImage(image);
    const QSize size = ui->labelDisplay->size();
    this->ui->labelDisplay->setPixmap(pix.scaled(size,Qt::KeepAspectRatio));
}
void MainWindow::captureError(const QString &text)
{
    QMessageBox::critical(this,windowTitle(), text);
}
```

Một số lưu ý:

- Trong file .pro đặt cấu hình đường dẫn đến thư viện OpenCV biên dịch cho ARM để có thể biên dịch ứng dụng chạy trên KIT ARM

- Hàm cvCaptureFromCAM(index) có tham số index là chỉ số của camera muốn mở.

Cần xác định chỉ số này (giá trị 0,1,2,…) tương ứng với camera muốn mở. index = -1 cho phép mở camera bất kỳ mà hệ thống tìm được.

Khi chạy trên nền tảng ARM, hàm này có thể thực hiện không chính xác, không mở được với một số loại camera (hàm trả về NULL) (nguyên nhân nằm trong thư viện OpenCV biên dịch cho ARM). Để tránh tình huống này, khi đó có thể sử dụng phương pháp khác để làm việc với thiết bị camera, đó là dùng API của hệ điều hành Linux (V4L2 - Video for Linux 2). (Chi tiết xem bài viết ở đây)

- Trong ứng dụng minh họa trên có sử dụng hàm để biến đổi từ ảnh IplImage (quản lý bởi thư viện OpenCV) sang ảnh Qimage (quản lý bởi Qt) để hiển thị trên giao diện đồ họa của Qt. Hàm được sử dụng là:

```
QImage convert(IplImage * image);

Hoặc:    QImage IplImage2QImage(const IplImage *iplImage);
```

- Sau khi có một khung hình được chuyển thành dạng biểu diễn QImage, có thể dễ dàng hiển thị khung hình này lên một điều khiển (ví dụ QLabel) bằng cách sau

```
void MainWindow::displayImage(const QImage &image)
{
    const QPixmap pix = QPixmap::fromImage(image);
    const QSize size = ui->labelDisplay->size();
    this->ui->labelDisplay->setPixmap(pix.scaled(size,Qt::KeepAspectRatio));
}
```

- Các khung hình được lấy và hiển thị theo thời gian định thời đã thiết lập, kết quả hiển thị sẽ là luồng video được thu nhận từ thiết bị camera.

(Mã nguồn minh họa tham khảo trong file đính kèm)

QtCvCapture.rar (37k) DCE- SOICT, 01:12 27-12-2013 v.2

Comments

You do not have permission to add comments.