Ubuntu 环境配置：

卸载 Ubuntu：https://blog.csdn.net/yeqiang19910412/article/details/79121581

删除 EFI 分区：

https://blog.csdn.net/weixin_39750861/article/details/79829532

安装 Ubuntu：https://blog.csdn.net/s717597589/article/details/79117112

阿里云源：

1. 目的是为了防止被墙
2. 建议在网速良好的情况下更新，否则大概率出问题（原因是下载时间过长，部分文件被忽略）。
3. https://blog.csdn.net/ezreal_king/article/details/72790291
4. 一定不要忘了备份原文件！！！

https://blog.csdn.net/areigninhell/article/details/79696751

安装 Terminator

```
sudo apt-get install terminator
```

2 使用

打开 Termintor 按 Ctrl-E(注意是大 E 要按住 Shift)可以垂直分割终端 Ctrl-O 可水平分割终端 按住 Alt 然后按上下左右可以在不同的分割窗中切换 Ctrl-D 可以关闭分割窗

2.1 配置

terminator 配置文件在~/.config/terminator/config 可以通过这个配置文件配置 terminator 的字体和颜色

安装 OpenCV3.3.1：最好在较好的网络条件下安装，否则容易因为文件下载失败导致错误

https://blog.csdn.net/weixin_40494464/article/details/80135983

https://blog.csdn.net/zhangjun62/article/details/80476274

一、安装官方给的 opencv 依赖包

1、sudo apt-get update

2、sudo apt-get install build-essential

3、sudo apt-get install cmake git libgtk2.0-dev pkg-config libavcodec-dev libavformat-dev libswscale-dev

sudo apt-get install libtbb2 libtbb-dev libjpeg-dev libpng-dev libtiff-dev libjasper-dev libdc1394-22-dev

4、sudo apt-get install python-dev python-numpy libtbb2 libtbb-dev libjpeg8-dev libpng12-dev libtiff5-dev libjasper-dev libdc1394-22-dev # 处理图像所需的包

5、sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev liblapacke-dev

6、sudo apt-get install libxvidcore-dev libx264-dev # 处理视频所需的包

7、sudo apt-get install libatlas-base-dev gfortran # 优化 opencv 功能

8、sudo apt-get install ffmpeg

9、sudo apt-get install libgtk-3-dev

10、sudo apt-get install libatlas-base-dev gfortran

11、sudo apt-get install python2.7-dev python3.5-dev

二、下载 OpenCV 源代码

12、wget https://github.com/opencv/opencv/archive/3.3.1.zip

wget https://github.com/opencv/opencv_contrib/archive/3.3.1.zip

三、配置编译 OpenCV

首先在用户目录下创建存放源码的文件夹，将两个源码包解压后放入 opencv 目录下

13、mkdir ~/opencv

14、unzip opencv-3.3.1.zip

unzip opencv-3.3.1.zip.1

15、mv opencv-3.4.1 ~/opencv

mv opencv_contrib-3.4.1 ~/opencv

16、cd opencv 查看一下是否转移成功

17、 cd opencv-3.3.1

18、mkdir build

19、cd build

20、运行以下命令

cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local -D INSTALL_PYTHON_EXAMPLES=ON -D INSTALL_C_EXAMPLES=ON -D OPENCV_EXTRA_MODULES_PATH=../../opencv_contrib-3.3.1/modules -D PYTHON_EXECUTABLE=/usr/bin/python3.5 -D BUILD_EXAMPLES=ON ..

其中 -D 表示定义，其后是命令和参数，第一个是编译类型，源码是是 release（release 是比较稳定的版本，建议用这个版本），所以参数是 RELEASE，第二个是安装路径，第三个和第四个是是否安装 C 与 Python 例子，第五个是扩展模块路径，第六个是 python 可执行程序路径，第七个是是否编译例子，最后那两个点不要去掉，表示上一级目录，而上一级目录是源码，所以一定不要去掉。注意其中的空格

21、sudo make -j8 采用多线程编译，但是这个出错多，如果出错，先运行 make clean，然后运行 sudo make，如果没有 make 命令，先执行 sudo apt-get install make

22、sudo make install

第四步编写一个 test.cpp

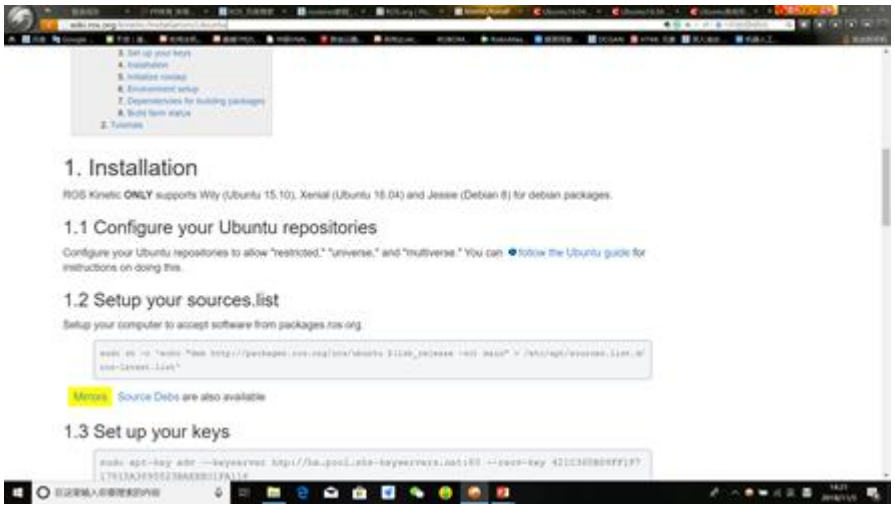
执行 g++: test.cpp -o test `pkg-config --libs--cflags opencv` -ldl 不加后面`pkg-config

--libs--cflags opencv` -ldl 会报错

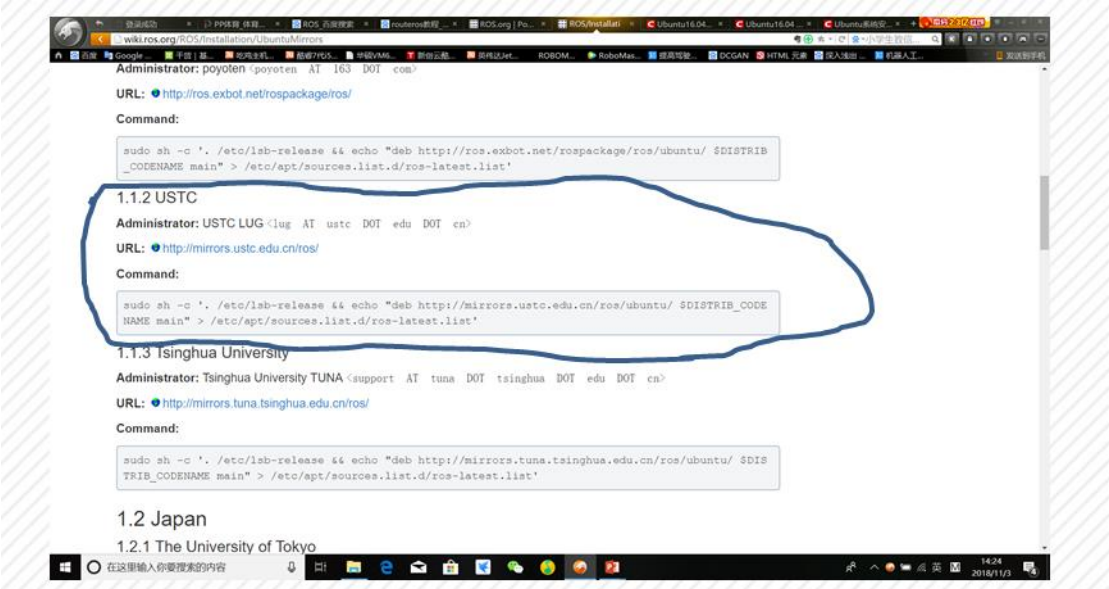
安 装 ROS kinetic ： 看 官 网 教 程

http://wiki.ros.org/kinetic/Installation/Ubuntu

1、选择镜像文件



2、选择源，这里选择中科大的或者清华的



1	sudo sh -c '. /etc/lsb-release && echo "deb http://mirrors.tuna.tsinghua.edu.cn/ros/ubuntu/ sources.list.d/ros-latest.list'

3、设置密码

```
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
```

4、安装

```
1 sudo apt-get update
2 sudo apt-get install ros-kinetic-desktop-full ##全部安装

第二条指令执行出错时，先执行一次 sudo apt-get -f install
```

在ros的使用过程中可能出现“某个依赖库不存在”或者“缺少某个库”的报错，这时只需要执行：

```
sudo apt-get install ros-kinetic-PACKAGE
```

其中PACKAGE用缺少的库文件名代替

5、初始化

```
1 sudo rosdep init
2 rosdep update
```

6、配置环境

```
1 echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
2 source ~/.bashrc
```

7、安装 building package 的依赖（Dependencies for building packages）

```
sudo apt-get install python-rosinstall python-rosinstall-generator python-wstool build-essential
```

8、测试是否安装成功

在三个不同的终端中输入

```
roscore

rosrun turtlesim turtlesim_node


rosrun turtlesim turtle_teleop_key
```


https://blog.csdn.net/lixujie666/article/details/80139112

v4l2 安装及使用

地址：https://github.com/twam/v4l2grab

一、安装

1、安装所需的库（libv4l 和 libjpeg）和自动工具

```
sudo apt-get install libjpeg-dev libv4l-dev autoconf automake libtool
```

2、克隆存储库

```
git clone https://github.com/twam/v4l2grab.git
```

3、转到目录

```
cd v4l2grab
```

4、创建自动工具文件

```
./autogen.sh
```

5、运行配置

```
./configure
```

6、Run make

```
make
```

7、运行 make install

```
sudo make install
```

8、尝试/v4l2grab-h 或仅 v4l2grab，如果您已安装它以获得一些帮助。

9、如果网络摄像机正常工作，请将其添加到“兼容设备”表中。

10、安装 v4l2-ctl 命令程序

sudo apt install v4l-utils

二、使用

1、首先用 v4l2-ctl --list-device 确定 usb 摄像头的 device 编号(一般为 /dev/video0)

2、然后查看该设备可以设置的参数：

v4l2-ctl -d /dev/video0 --list-ctrls

罗技 c930e 摄像头的参数如下：

```
                     brightness (int)    : min=0 max=255 step=1
default=-8193 value=128
                       contrast (int)    : min=0 max=255 step=1
default=57343 value=128
                     saturation (int)    : min=0 max=255 step=1
default=57343 value=128
 white_balance_temperature_auto (bool)   : default=1 value=1
                           gain (int)    : min=0 max=255 step=1
default=57343 value=0
           power_line_frequency (menu)   : min=0 max=2 default=2
value=2
      white_balance_temperature (int)    : min=2000 max=6500 step=1
default=57343 value=4000 flags=inactive
                      sharpness (int)    : min=0 max=255 step=1
default=57343 value=128
         backlight_compensation (int)    : min=0 max=1 step=1
default=57343 value=0
                  exposure_auto (menu)   : min=0 max=3 default=0
value=3
              exposure_absolute (int)    : min=3 max=2047 step=1
default=250 value=250 flags=inactive
         exposure_auto_priority (bool)   : default=0 value=1
                   pan_absolute (int)    : min=-36000 max=36000
step=3600 default=0 value=0
                  tilt_absolute (int)    : min=-36000 max=36000
step=3600 default=0 value=0
                 focus_absolute (int)    : min=0 max=250 step=5
default=8189 value=0 flags=inactive
                     focus_auto (bool)   : default=1 value=1
```

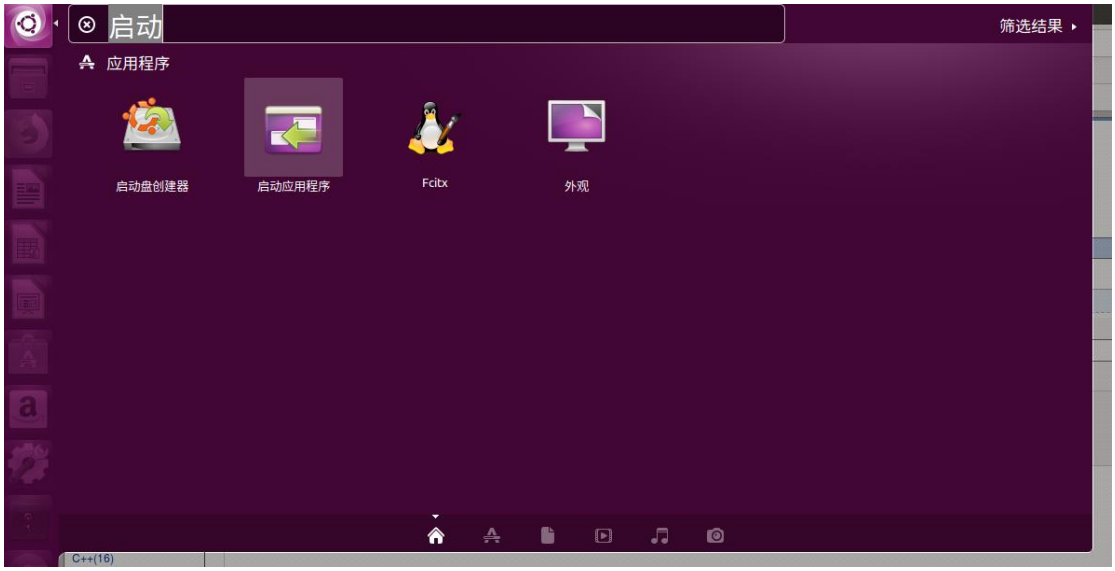
zoom_absolute (int)   : min=100 max=500 step=1 default=57343 value=100

3、最后可以可以设置参数了：

v4l2-ctl -d /dev/video1 -c exposure_absolute=90

###Ubuntu 实现程序开机自启动

1、在程序搜索栏打开启动应用程序



2、添加一项启动程序，其中名称和注释任意，命令栏输：gnome-terminal

3、在~/.bashrc 中添加需要开机自启动的命令

如

```
gnome-terminal -x roscore
sleep 2  #表示休眠 2 秒再启动下一个命令
gnome-terminal -x rosrun test test
```

###Ubuntu usb 设备端口号绑定

1.将串口设备插入 USB 口，通过 lsusb 查看端口信息。例如：

```
king@king:~$ lsusb
Bus 002 Device 003: ID 1770:ff00
Bus 002 Device 002: ID 8087:8000 Intel Corp.
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 004: ID 5986:055c Acer, Inc
Bus 001 Device 003: ID 8087:07dc Intel Corp.
Bus 001 Device 002: ID 8087:8008 Intel Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 004 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 003 Device 003: ID 25a7:2402
Bus 003 Device 005: ID 1a86:7523 QinHeng Electronics HL-340 USB-Serial adapter
Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```


ID 1a86:7523 表示 usb 设备的 ID(这个 ID 由芯片制造商设置，可以唯一表示该设备)

1a86 usb_device_descriptor.idVendor
7523 usb_device_descriptor.idProduct

2.在/etc/udev/rules.d/下创建任意名称的规则配置文件，如：usb.rules。

```
KERNEL=="ttyUSB*", ATTRS{idVendor}=="067b", ATTRS{idProduct}=="2303", MODE:="0777", SYMLINK+="user_uart"
KERNEL=="ttyUSB*", ATTRS{idVendor}=="1a86", ATTRS{idProduct}=="7523", MODE:="0777", SYMLINK+="mcu_uart"
```

意思就是匹配 sys 中内核名为 ttyUSB*的设备，属性匹配依据生产商编号 idVendor 和产品号 idProduct，设定读写权限为 0777，符号链接名为 user_uart-----PL2303 串口转 USB，mcu_uart----CH340 串口转 USB。

idVendor 和 idProduct 由 lsusb -vvv 命令查看。
保存退出后 udev 规则就生效了，重新拔插两个串口设备，就可以看到/dev/user_uart 指向/dev/ttyUSB0，/dev/mcu_uart 指向/dev/ttyUSB1. 这样以来，我只要在程序里打开/dev/user_uart 或/dev/mcu_uart 就可以一直准确的打开指定的串口设备了。

###使用 RoboWave 配置 USB 摄像头的 ROS 下的 OpenCV 运行环境

一.环境准备
Ubuntu16.04
ROS-kinetic
opencv3.3.1
video-stream-opencv（Python）
或者 usb_cam （c++）

一个 USB 摄像头

video-stream-opencv 是 USB 摄像头驱动，关于它的介绍，请看
github:https://github.com/ros-drivers/video_stream_opencv

二. 在 ROS 下创建工作空间

在欢迎界面， 点击“新建工作区”按钮（或选择“文件 - 新建工作区”） ， 选择路径并填写工作区名称， 如“catkin_ws”， 则会创建一个名为“catkin_ws”工作区， 并显示在资源管理
器窗口：

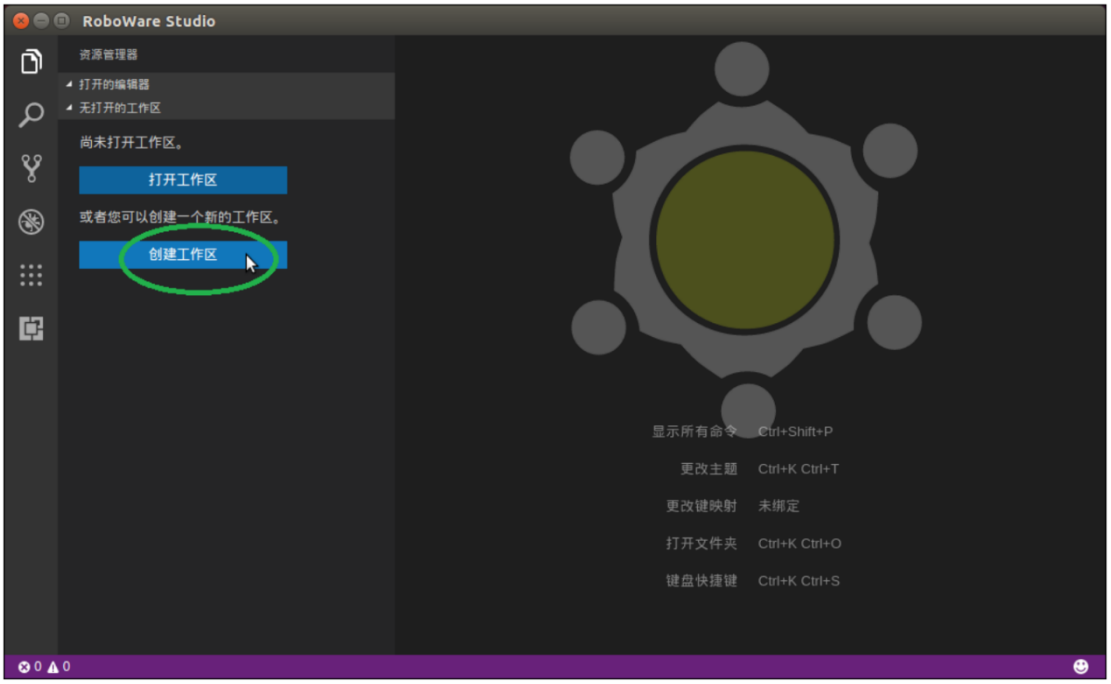


图 3-1 RoboWare Studio 欢迎界面

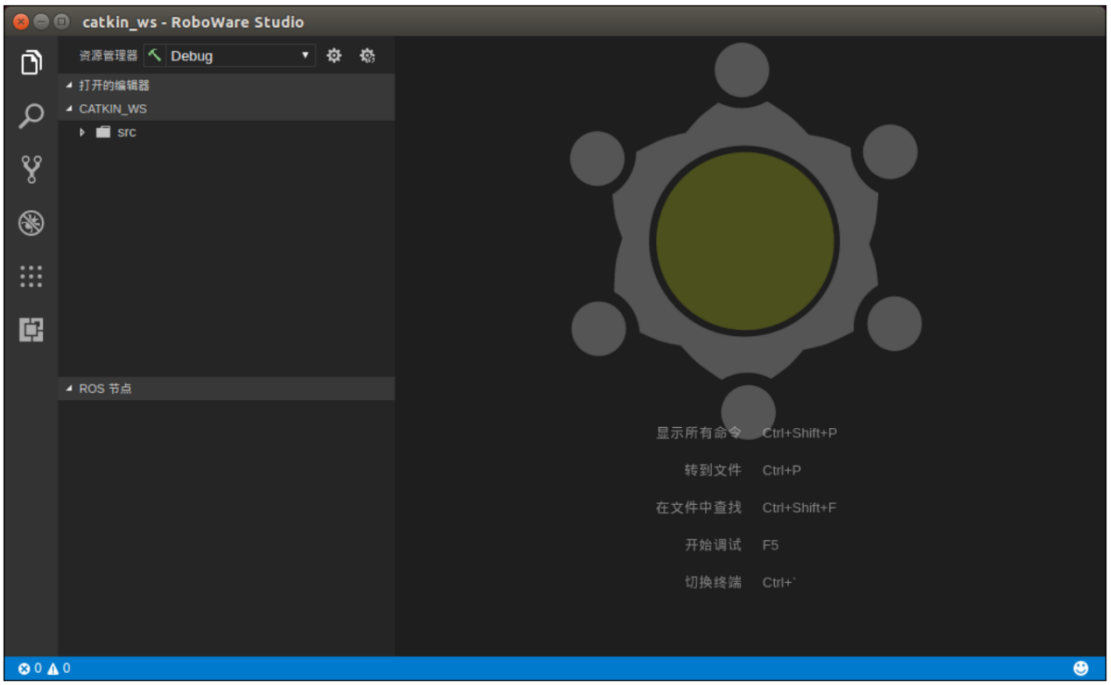


图 3-2 创建完成 catkin ws 工作区

三、在工作空间下创建程序包

1、安装 video-stream-opencv

```
$ cd ~/catkin_ws/src/
$ git clone https://github.com/ros-drivers/video_stream_opencv.git
$ cd ..
$ catkin_make
```

2、编译好之后，添加编译好的文件到环境变量：

```
$ source devel/setup.bash
```

3、测试是否安装成功

在终端输入：

```
$ rosrun video_stream_opencv test_video_resource.py 0  #后面的 0 是摄像头编号
```

https://yq.aliyun.com/ziliao/203675

https://blog.csdn.net/Gpwner/article/details/78961362

1、安装 usb_cam

进入创建好的工作空间：

```
cd ~/catkin_ws/src
git clone https://github.com/bosch-ros-pkg/usb_cam.git
```

然后退回到工作空间，编译代码：

```
cd ~/catkin_ws
catkin_make
```

编译好之后，添加编译好的文件到环境变量：

```
source devel/setup.bash
```

2、然后接下来测试 usb_cam：

先运行 usb_cam 节点：

```
rosrun usb_cam usb_cam_node
```

运行上面命令发现没有显示图像，只看到摄像头打开了。这是因为 ros 发布的 topic 是 /usb_cam/image_raw。新打开一个终端，可以通过如下命令查看：

```
rostopic list
```

所以我们需要运行如下命令才可以看到图像：

```
rosrun image_view image_view image:=/usb_cam/image_raw
```

或者直接写 launch 文件，这样就不用一个终端运行 node，一个终端看图像。新建 usb_cam_test.launch：

```
<launch>
  <node name="usb_cam" pkg="usb_cam" type="usb_cam_node" output="screen" >
    <param name="video_device" value="/dev/video0" />
    <param name="image_width" value="640" />
    <param name="image_height" value="480" />
```

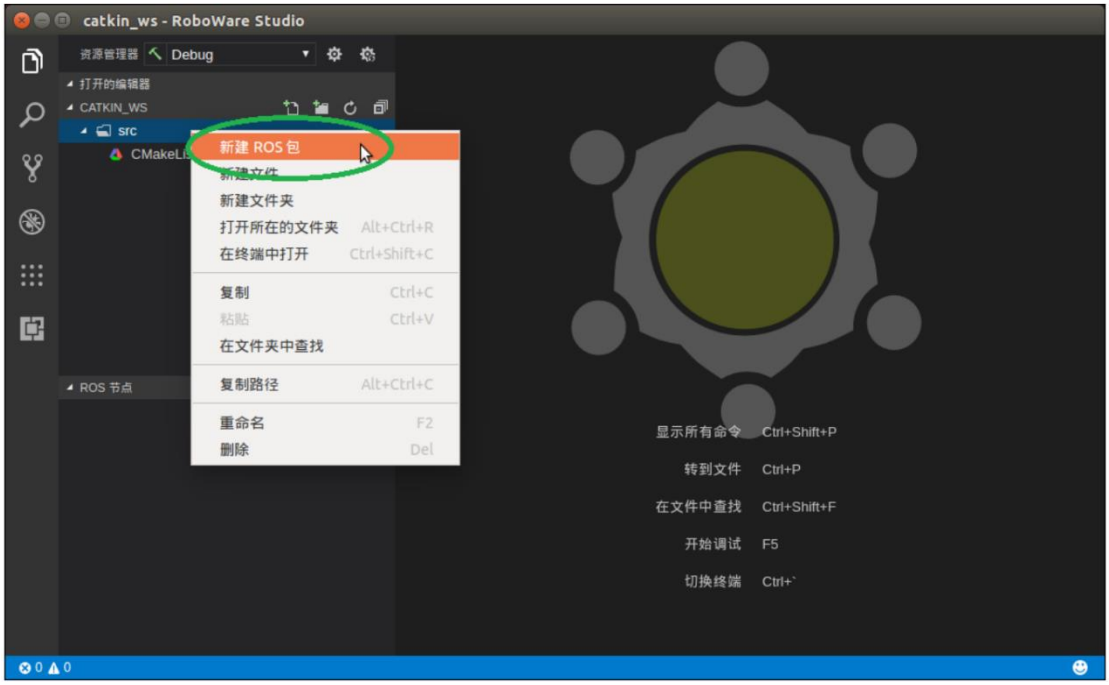
```
    <param name="pixel_format" value="yuyv" />
    <param name="camera_frame_id" value="usb_cam" />
    <param name="io_method" value="mmap"/>
  </node>
  <node name="image_view" pkg="image_view" type="image_view"
respawn="false" output="screen">
    <remap from="image" to="/usb_cam/image_raw"/>
    <param name="autosize" value="true" />
  </node>
</launch>
```

然后终端直接运行：

```
roslaunch usb_cam usb_cam_test.launch
```

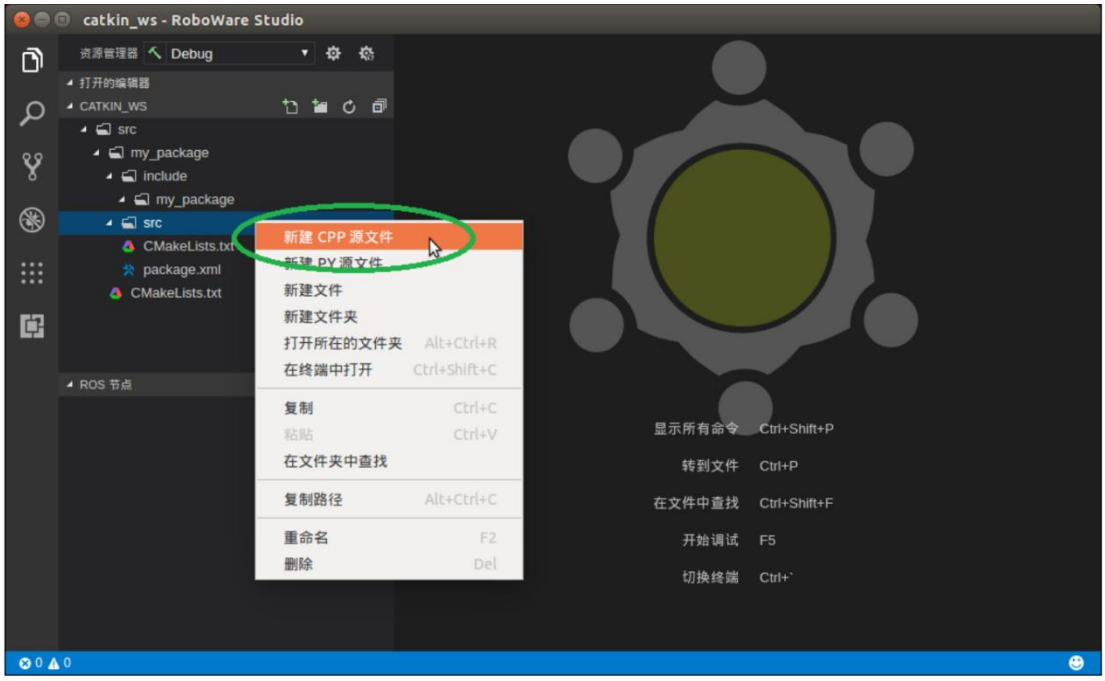
右键点击 ROS 工作区下的“src”， 选择“新建 ROS 包”，输入包名称及其依赖包的名称，如：
“robot_vision roscpp std_msgs cv_bridge image_transport sensor_msgs”

回车后，会创建名为“robot_vision ”、以“roscpp std_msgs cv_bridge image_transport sensor_msgs”为依赖的 ROS 包：



3、创建.cpp 源文件

在创建的程序包的 src 文件中创建一个文本文件，并命名为 opencv_test.cpp,选择添加到可执行文件中。具体代码和注释如下：




```
#include<ros/ros.h> //ros 标准库头文件
#include<image_transport/image_transport.h>/* image_transport 头文件用来在 ROS 系统中的话题上发布和订阅图象消息*/
#include<cv_bridge/cv_bridge.h>/* cv_bridge 中包含 CvBridge 库 */
#include<sensor_msgs/image_encodings.h>/* ROS 图象类型的编码函数*/
#include<iostream> //C++标准输入输出库

//OpenCV3 标准头文件
#include<opencv2/opencv.hpp>


static const std::string OPENCV_WINDOW = "Image window"; //定义输入窗口名称


//定义一个转换的类
class ImageConverter
{
private:
    ros::NodeHandle nh_; //定义 ROS 句柄
    image_transport::ImageTransport it_; //定义一个 image_transport 实例
    image_transport::Subscriber image_sub_; //定义 ROS 图象接收器
    image_transport::Publisher image_pub_; //定义 ROS 图象发布器
public:
    ImageConverter()
```

```
      :it_(nh_) //构造函数
    {    //这里是 usb 摄像头的 topic，官网默认是/camera/image_raw，这里修改为 usb 摄像头/usb_cam/image_raw。
             //定义图象接受器，订阅话题是“/usb_cam/image_raw”
        image_sub_ = it_.subscribe("/usb_cam/image_raw", 1, &ImageConverter::convert_callback, this);
        image_pub_ = it_.advertise("/image_converter/output_video", 1); //定义图象发布器
        //初始化输入输出窗口
        cv::namedWindow(OPENCV_WINDOW);
    }
    ~ImageConverter() //析构函数
    {
         cv::destroyWindow(OPENCV_WINDOW);
    }
    /*这是一个 ROS 和 OpenCV 的格式转换回调函数，将图象格式从 sensor_msgs/Image--->cv::Mat*/
    void convert_callback(const sensor_msgs::ImageConstPtr& msg)
    {
        cv_bridge::CvImagePtr cv_ptr; // 声明一个 CvImage 指针的实例

        try
        {   //将 ROS 消息中的图象信息提取，生成新 cv 类型的图象，复制给 CvImage 指针
            cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::image_encodings::BGR8);
        }
        catch(cv_bridge::Exception& e)  //异常处理
        {
            ROS_ERROR("cv_bridge exception: %s", e.what());
            return;
        }

        image_process(cv_ptr->image); //得到了 cv::Mat 类型的图象，在 CvImage 指针的 image 中，将结果传送给处理函数
         // Output modified video stream
         image_pub_.publish(cv_ptr->toImageMsg());
    }
    /*这是图象处理的主要函数，一般会把图像处理的主要程序写在这个函数中。这里的例子只是一个彩色图象到灰度图象的转化
    到时候要修改就修改这里的函数即可     */
    void image_process(cv::Mat& img)
    {
        // Draw an example circle on the video stream
```

```
        if (img.rows > 60 && img.cols > 60)
            {
                cv::circle(img, cv::Point(50, 50), 10,
CV_RGB(255,0,0));
            }
        // Update GUI Window
         cv::imshow(OPENCV_WINDOW, img);
         cv::waitKey(5);
    }
};

//主函数
int main(int argc, char** argv)
{
    ros::init(argc, argv, "image_converter");
    ImageConverter ic;
    ros::spin();
    return 0;
}
```

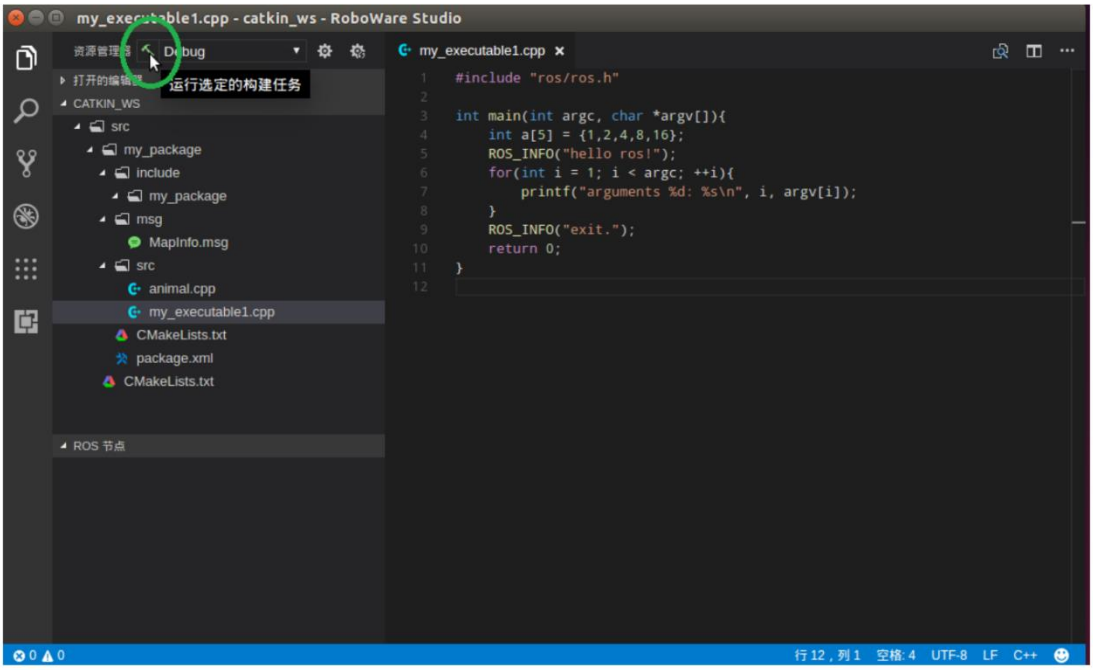
4.CMakeLists.txt

进入包的目录，修改 CmakeList.txt，在文件最后添加：

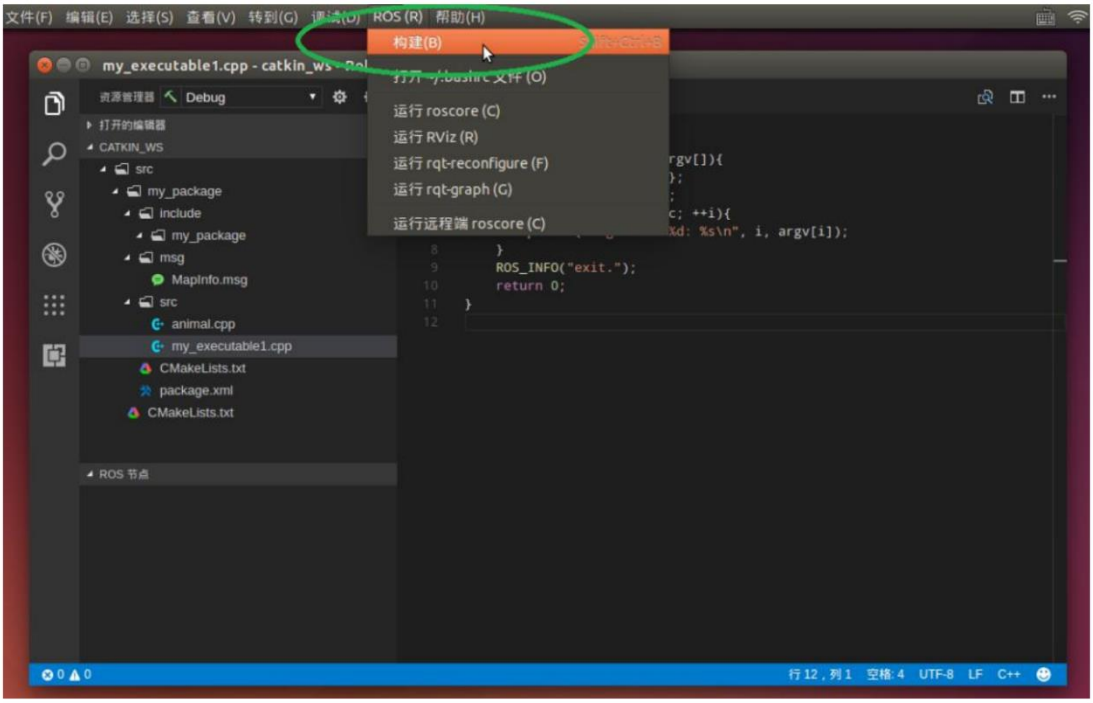
```
find_package(OpenCV REQUIRED)

include_directories(${OpenCV_INCLUDE_DIRS})

add_executable(opencv_test src/opencv_test.cpp) #根据自己的文件名修改
里面 opencv_test 的内容

target_link_libraries(opencv_test                ${catkin_LIBRARIES}
${OpenCV_LIBRARIES})
```

其中， Debug 和 Release 选项分别表示构建调试版和发布版， 默认构建方式为本地构建 。

有两种构建方法：



点击构建按钮构建



菜单选择 “ROS - 构建”

图 3-12 构建 ROS 包

编译完成后，执行：

```
source devel/setup.bash
```

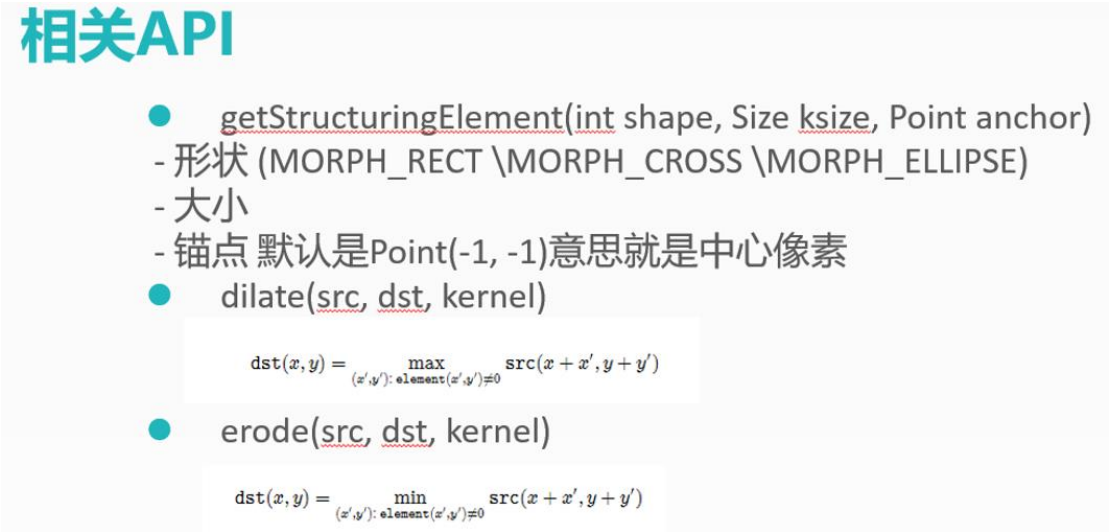
先打开一个终端运行 roscore，用以节点之间的通信交互。
再打开一个终端运行 rosrun usbcam usbcam_node
再打开一个终端运行 rosrun opencvtest opencv_testcam
之后即可看到 opencv 处理后摄像头的图像。

如果找不到包，执行命令

```
rospack profile
source ~/catkin_ws/devel/setup.bash
```

OpenCV——图像处理入门：膨胀与腐蚀、图像模糊、边缘检测

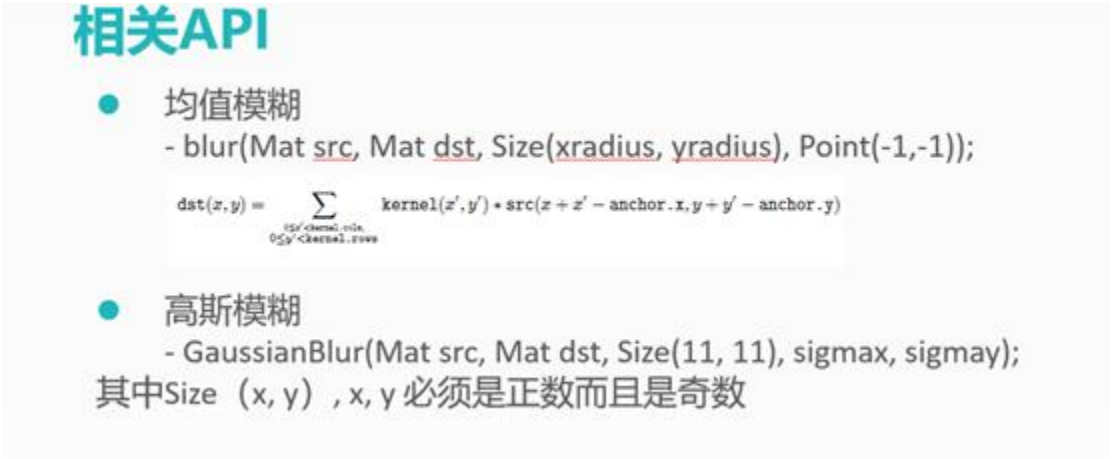
膨胀和腐蚀：




```
#include <opencv2/opencv.hpp>
#include <iostream>
//#include <opencv2/highgui/highgui.hpp>
//#include <imgproc/imgproc.hpp>
using namespace cv;

int main(int argc, char** argv)
{
    Mat srcImage = imread("test.jpg");
    imshow("[原图]", srcImage);

```

```
12      //进行膨胀操作
13      Mat element1 = getStructuringElement(MORPH_RECT, Size(5, 5));
14      Mat dstImage1;
15      dilate(srcImage, dstImage1, element1);
16      imshow("[膨胀效果图]", dstImage1);
17
18      //进行腐蚀操作
19      Mat element2 = getStructuringElement(MORPH_RECT, Size(5, 5));
20      Mat dstImage2;
21      erode(srcImage, dstImage2, element2);
22      imshow("[腐蚀效果图]", dstImage2);
23
24      waitKey(0);
25
26      return 0;
27  }
```

图像模糊：



```cpp
#include <opencv2/opencv.hpp>
#include <iostream>
//#include <opencv2/highgui/highgui.hpp>
//#include <imgproc/imgproc.hpp>
using namespace cv;

int main(int argc, char** argv)
{
    Mat srcImage = imread("test.jpg");
    imshow("[原图]", srcImage);

    //进行均值滤波操作
    Mat dstImage1;
    blur(srcImage, dstImage1, Size(7,7));
    imshow("[效果图]", dstImage1);

    waitKey(0);

    return 0;
}
```

Canny 边缘检测：


```cpp
#include <opencv2/opencv.hpp>
#include <iostream>
//#include <opencv2/highgui/highgui.hpp>
//#include <imgproc/imgproc.hpp>
using namespace cv;

//Canny 边缘检测
```

```
int main(int argc, char** argv)
{
    Mat srcImage = imread("test.jpg");
    imshow("[原图]", srcImage);

    Mat edge, gray;
    //将原图转换成灰度图像
    cvtColor(srcImage, gray, COLOR_BGR2GRAY);
    //均值滤波降噪
    blur(gray, edge, Size(3, 3));
    //运行 Canny 算子
    Canny(edge, edge, 3, 9, 3);


    imshow("[效果图]", edge);

    waitKey(0);

    return 0;
}
```

OpenCV——视频操作基础

读入视频：

```
VideoCapture 类
//方法一
VideoCapture capture;
capture.open("test.avi");

//方法二
VideoCapture capture("test.avi");
```

```
#include <opencv2/opencv.hpp>
#include <iostream>
//#include <opencv2/highgui/highgui.hpp>
//#include <imgproc/imgproc.hpp>
using namespace cv;
```

```
6
7 int main(int argc, char** argv)
8 {
9     //读入视频
10     VideoCapture capture1("test.mp4");
11     //读入摄像头
12     VideoCapture capture2(0);
13
14     while (1)
15     {
16         Mat frame1, frame2;//定义一个 Mat 类变量，用于存储每一帧的
图像
17
18         //capture.read(frame);两种方法读取当前帧
19         capture1 >> frame1;//两种方法读取当前帧
20
21         imshow("读取视频",frame1);
22
23         capture2 >> frame2;
24         imshow("读取摄像头", frame2);
25         waitKey(30);//延时 30ms
26     }
27     return 0;
28 }
```

OpenCV——基本图形绘制（椭圆、圆、多边形、直线、矩形）

```
1 //绘制椭圆
2 void DrawEllipse(Mat img, double angle)
3 {
4     int thickhness = 2;
5     int lineType = 8;
6
7     ellipse(img,
8         Point(WINDOW_WIDTH / 2, WINDOW_WIDTH / 2),
9         Size(WINDOW_WIDTH / 4, WINDOW_WIDTH / 16),
10         angle,
11         0, 360,
12         Scalar(255, 129, 0),
13         thickhness,
```

```
14          lineType);
15
16 }
```

函数 DrawEllipse 调用了 OpenCV 中的 ellipse 函数，将椭圆画到图像 img 上，椭圆中心为点(WINDOW_WIDTH/2.0, WINDOW_WIDTH/2.0)，并且大小位于矩形(WINDOW_WIDTH/4.0, WINDOW_WIDTH/16.0)内。椭圆旋转角度为 angle，扩展的弧度从 0 度到 360 度。图形颜色为 Scalar(255,129,0)代表的蓝色，线宽(thickness)为 2，线型(lineType)为 8（8 联通线型）。

```
1 //绘制实心圆
2 void DrawFilledCircle(Mat img, Point center)
3 {
4     int thickness = -1;
5     int lineType = 8;
6
7     circle(img,
8         center,
9         WINDOW_WIDTH / 32,
10        Scalar(0, 0, 255),
11        thickness,
12        lineType);
13 }
```

函数 DrawFilledCircle()调用了 OpenCV 中的 circle 函数，将圆画到图像 img 上，圆心由点 center 定义，圆的半径为 WINDOW_WIDTH/32，圆的颜色为 Scalar(0,0,255)，按 BGR 的格式为红色，线粗定义为 thickness = -1，因此绘制的圆是实心的。

当 thickness 为其他>0 的值时为正常的空心圆

```
1 void DrawPolygon(Mat img)
2 {
3     int lineType = 8;
4
5     Point rookPoints[1][20];
6     rookPoints[0][0] = Point(WINDOW_WIDTH/4,7* WINDOW_WIDTH/8);
```

```
 7      rookPoints[0][1] = Point(3* WINDOW_WIDTH/4,7* WINDOW_WIDTH/8);
 8      rookPoints[0][2] = Point(3* WINDOW_WIDTH/4,13*
WINDOW_WIDTH/16);
 9      rookPoints[0][3] = Point(11* WINDOW_WIDTH/16,13*
WINDOW_WIDTH/16);
10      rookPoints[0][4] = Point(19* WINDOW_WIDTH/32,3*
WINDOW_WIDTH/8);
11      rookPoints[0][5] = Point(3* WINDOW_WIDTH/4,3* WINDOW_WIDTH/8);
12      rookPoints[0][6] = Point(3*WINDOW_WIDTH/4, WINDOW_WIDTH/8);
13      rookPoints[0][7] = Point(26* WINDOW_WIDTH/40, WINDOW_WIDTH/8);
14      rookPoints[0][8] = Point(26* WINDOW_WIDTH/40, WINDOW_WIDTH/4);
15      rookPoints[0][9] = Point(22* WINDOW_WIDTH/40, WINDOW_WIDTH/4);
16      rookPoints[0][10] = Point(22* WINDOW_WIDTH/40,
WINDOW_WIDTH/8);
17      rookPoints[0][11] = Point(18* WINDOW_WIDTH/40,
WINDOW_WIDTH/8);
18      rookPoints[0][12] = Point(18* WINDOW_WIDTH/40,
WINDOW_WIDTH/4);
19      rookPoints[0][13] = Point(14* WINDOW_WIDTH/40,
WINDOW_WIDTH/4);
20      rookPoints[0][14] = Point(14* WINDOW_WIDTH/40,
WINDOW_WIDTH/8);
21      rookPoints[0][15] = Point(WINDOW_WIDTH/4, WINDOW_WIDTH/8);
22      rookPoints[0][16] = Point(WINDOW_WIDTH/4,3* WINDOW_WIDTH/8);
23      rookPoints[0][17] = Point(13* WINDOW_WIDTH/32,3*
WINDOW_WIDTH/8);
24      rookPoints[0][18] = Point(5* WINDOW_WIDTH/16,13*
WINDOW_WIDTH/16);
25      rookPoints[0][19] = Point(WINDOW_WIDTH/4,13* WINDOW_WIDTH/16);
26
27      const Point* ppt[1] = { rookPoints[0] };
28      int npt[] = { 20 };
29
30      fillPoly(img,
31          ppt,
32          npt,
33          1,//好像只能为1，填其他数程序出错
34          Scalar(255, 255, 255),
35          lineType);
36
37  }
```

1、cvPolyLine 绘制简单或多样的多边形。

void cvPolyLine(CvArr* img, CvPoint** pts, int* npts, int contours, int is_closed,
CvScalar color, int thickness=1, int line_type=8, int
shift=0);
img 图像。
pts 折线的顶点指针数组。
npts 折线的定点个数数组。也可以认为是 pts 指针数组的大小
contours 折线的线段数量。
is_closed 指出多边形是否封闭。如果封闭，函数将起始点和结束点连线。
color 折线的颜色。
thickness 线条的粗细程度。
line_type 线段的类型。参见 cvLine。
shift 顶点的小数点位数。

2、cvFillPoly 用于一个单独被多边形轮廓所限定的区域内进行填充。

函数可以填充复杂的区域,例如，有漏洞的区域和有交叉点的区域等等。
void cvFillPoly(CvArr* img, CvPoint** pts, int* npts, int contours,CvScalar color,
int line_type=8, int shift=0);
img
图像。
pts 指向多边形的数组指针。
npts 多边形的顶点个数的数组。
contours 组成填充区域的线段的数量。
color 多边形的颜色。
line_type 组成多边形的线条的类型。
shift 顶点坐标的小数点位数

绘制直线

```
1 int thickness = 2;
2     int lineType = 8;
3     line(rookImage,//要绘制的图
4         Point(0, 0),//起始点
5         Point(WINDOW_WIDTH, WINDOW_WIDTH),//终点
6         Scalar(255, 255, 255),
7         thickness,
```

```
8          lineType);
```

绘制矩形

```
1 rectangle (image3,  rec1,Scalar(0, 0, 255), -1, 8, 0)
```

rectangle(CvArr* img, CvPoint pt1, CvPoint pt2, CvScalar color,

int thickness=1, int line_type=8, int shift=0);

img

图像.

pt1

矩形的一个顶点。

pt2

矩形对角线上的另一个顶点

color

线条颜色 (RGB) 或亮度（灰度图像 ）(grayscale image）。

thickness

组成矩形的线条的粗细程度。取负值时（如 CV_FILLED）函数绘制填充了色彩的矩形。

line_type

线条的类型。见 cvLine 的描述

shift

坐标点的小数点位数。

OpenCV——计时函数和访问像素的三种方法对比

5.1.4 计时函数

另外有个问题是如何计时。可以利用这两个简便的计时函数——getTickCount()和 getTickFrequency()。

- getTickCount()函数返回 CPU 自某个事件（如启动电脑）以来走过的时钟周期数
- getTickFrequency()函数返回 CPU 一秒钟所走的时钟周期数。这样，我们就能轻松地以秒为单位对某运算计时。

这两个函数组合起来使用的示例如下。

```
1 double time0 = static_cast<double>(getTickCount());//记录起始时间
2
3 time0 = ((double)getTickCount() - time0) / getTickFrequency();
4 cout << "运行时间为 "<< time0 << "秒"<<endl;//输出运行时间
```

访问像素的三种方法

【方法 1】指针访问，c 语言操作符[ ]（速度最快，但有越界的风险）

```
1     int rowNumber = dst.rows;//行数
2     int colNumber = dst.cols*dst.channels();//列数*通道数=每一行的元素个数
3
4     //循环遍历每个元素
5     for (int i = 0; i < rowNumber; i++)//行循环
6     {
7         uchar* data = dst.ptr<uchar>(i);//获取第 i 行首地址
8         for (int j=0; j < colNumber; j++)//列循环
9             data[j] = data[j] / div* div+div/2;//颜色缩减操作
       //也可以写成   *data++=*data/div*div+div/2;

10    }
```

【方法 2】迭代器 iterator（绝对安全，不会越界）

第二种方法为用迭代器操作像素，这种方法与 STL 库的用法类似。

在迭代法中，我们所需要做的仅仅是获得图像矩阵的 begin 和 end，然后增加迭代直至从 begin 到 end。将*操作符添加在迭代指针前，即可访问当前指向的内容。

```
1     Mat_<Vec3b>::iterator it = dst.begin<Vec3b>();//初始位置的迭代器
2     Mat_<Vec3b>::iterator itend = dst.end<Vec3b>();//终止位置的迭代器
3     //彩色图像每个像素有 3 个通道
4     for (; it != itend; ++it)
5     {
6         (*it)[0] = (*it)[0] / div * div + div / 2;
7         (*it)[1] = (*it)[1] / div * div + div / 2;
8         (*it)[2] = (*it)[2] / div * div + div / 2;
9     }
```

【方法 1】动态地址计算（最慢）

```
 1     int rowNumber = dst.rows;//行数
 2     int colNumber = dst.cols;//列数
 3     for (int i = 0; i < rowNumber; i++)
 4     {
 5         for (int j = 0; j < colNumber; j++)
 6         {
 7             //彩色图像每个像素有 3 个通道
 8             dst.at<Vec3b>(i, j)[0] = dst.at<Vec3b>(i, j)[0] / div * div + div / 2;
 9             dst.at<Vec3b>(i, j)[1] = dst.at<Vec3b>(i, j)[1] / div * div + div / 2;
10             dst.at<Vec3b>(i, j)[2] = dst.at<Vec3b>(i, j)[2] / div * div + div / 2;
11
12         }
13     }
```

让我们讲解一下上述的代码。

Mat 类中的 cols 和 rows 给出了图像的宽和高。而成员函数 at（int y，int x）可以用来存取图像元素，但是必须在编译期知道图像的数据类型。需要注意的是，我们一定要确保指定的数据类型要和矩阵中的数据类型相符合，因为 at 方法本身不会对任何数据类型进行转换。

对于彩色图像，每个像素由三个部分构成：蓝色通道、绿色通道和红色通道（BGR）。因此，对于一个包含彩色图像的 Mat，会返回一个由三个 8 位数组成的向量。OpenCV 将此类型的向量定义为 Vec3b，即由三个 unsigned char 组成的向量。这也解释了为什么存取彩色图像像素的代码可以写出如下形式：

```
image.at<Vec3b>(j,i)[channel]=value;
```

其中，索引值 channel 标明了颜色通道号。

另外需要再次提醒大家的是，OpenCV 中的彩色图像不是以 RGB 的顺序存放的，而是 BGR，所以程序中的 outputImage.at<Vec3b>(i,j)[0]代表的是该点的 B 分量。同理还有(*it)[0]。