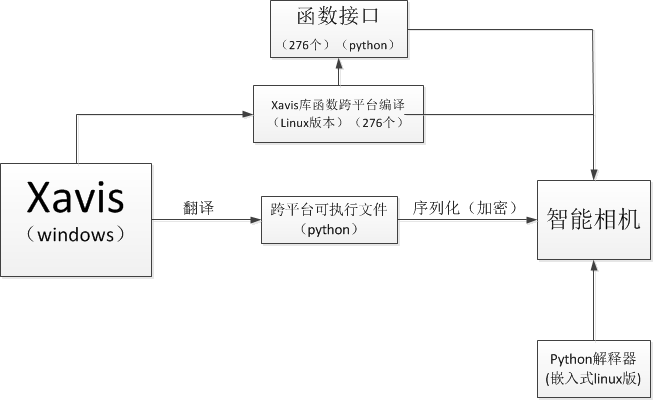
**智能相机组态编程软件开发项目书**

1. **智能相机组态编程软件开发思想**



1. **具体工作内容**
2. 给智能相机安装嵌入式Linux版本Python解释器（已完成）
3. 编写Xaivs脚本翻译程序（进行中）
4. 编写脚本文件（可执行文件）PC-Windows系统内加密操作（已完成）
5. 编写脚本文件（可执行文件）嵌入式Linux系统内解密操作（已完成）
6. 库函数封装（函数文件+函数接口）（进行中）
7. **函数接口示例**

//定义接口

# -\*- coding: utf-8 -\*-

import os

**def process\_image(imagename):**

**""" process an image and save the results in a file"""**

**path = os.path.abspath(os.path.join(os.path.dirname("\_\_file\_\_"),os.path.pardir))**

**path1 = path+"/python/ConsoleCV"**

**path2= path+"/python/"+imagename**

**if imagename[-3:] == 'jpg':**

**cmmd = str(path1+" "+path2)**

**os.system(cmmd)**

**print 'processed', imagename**

**elif imagename[-3:] == 'bmp':**

**cmmd = str(path1+" "+path2)**

**os.system(cmmd)**

**print 'processed', imagename**

**else:**

**print 'Cannot processed', imagename**

imagename="beach.jpg"

path = os.path.abspath(os.path.join(os.path.dirname("\_\_file\_\_"),os.path.pardir))

print path

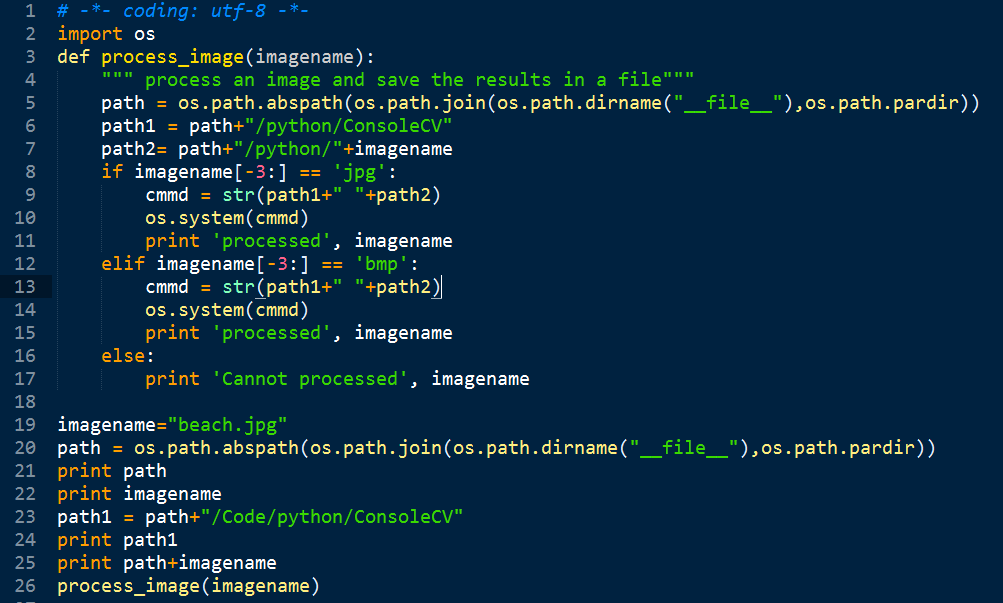
print imagename

path1 = path+"/Code/python/ConsoleCV"

print path1

print path+imagename

process\_image(imagename)



**//调用函数**

# -\*- coding: utf-8 -\*-

from ctypes import \*

def func\_imgprocess(path,str):

#f = '/home/forrest/Code/python/libcv.so'

cdll.LoadLibrary(path)

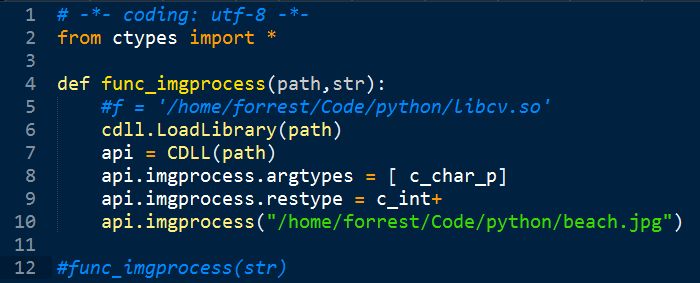
api = CDLL(path)

api.imgprocess.argtypes = [ c\_char\_p]

api.imgprocess.restype = c\_int+

api.imgprocess("/home/forrest/Code/python/beach.jpg")

#func\_imgprocess(str)



1. **封装库函数前期准备工作**
2. 安装PC-Linux系统
3. 安装Linux下对应版本的OpenCV（OpenCV2.3.1）
4. 安装gcc-arm交叉编译器
5. **可能用到的Linux命令**

1.安装python版本的opencv

sudo apt-get install python-opencv //sudo是取得权限

2.编译c程序：

gcc -c hello.c

gcc -o hello hello.o //-o后面制定目标文件名

3.编译python库函数

g++ -fPIC wrapper.cpp -o example.so -shared -I/usr/include/python2.7

（g++ -fPIC -o example.so wrapper.cpp -shared 也可实现）

4.编译生成静态库.a

静态库的原则是“以空间换时间”，增加程序体积，减少运行时间;在编译时候，先生成目标文件.o，然后用ar文件对目标文件归档，生成静态库文件。

如：ar -rc libtest.a myalib.o （注意：ar -rc 目标 .o文件名），目标一定要以lib开头。

使用时候，在链接时候，加上选项 -l 后接库文件名，注意：必须是文件名去掉后缀和lib，

如：gcc -o main main.o -ltest而且-ltest必须放在main.o的后面，（规则是，越底层的库越要放在后面）。

5.编译生成动态库.so

（1）生成：在链接时，用如下选项：-shared -fpic 如： gcc -fpic -shared -o libtest.so myalib.c

（2）使用：有隐式使用和显示使用。

隐式使用就是共享方式，程序一开始运行就调进去。在链接时候用如下：

gcc -o main main.o -L./lib -ltest(像静态库的一样)

显示使用就是在程序中用语句把动态库调进来，用系统调用：dlopen、dlsym、dlerror、dlclose函数，那样在编译链接时候，不用加上：-L./lib -ltest了。不过要使用dl\*系列函数在编译链接时要加上 –ldl

6、如果同一目录下，既有静态库也有动态库，比如libtest.a libtest.so都存在，那么dl程序（等一下介绍）就把动态库调进去，没有动态的，就找静态的。再没有，就报错。

7、动态库的搜索路径

dl对动态库的搜索路径如下（按顺序如下）

a.编译目标代码时指定的动态库搜索路径；（如果要指定程序行时在./lib目录下找库文件libtest.so，

命令如下：gcc -o main main.c -L./lib -ltest -Wl,-rpath ./lib) ，其中，-Wl的意思是，后面的选项直接交给ld程序处理,-rpath选项是说更改搜索路径为后面的参数./lib

b.环境变量LD\_LIBRARY\_PATH指定的动态库搜索路径；

c.配置文件/etc/ld.so.conf中指定的动态库搜索路径；（修改完文件后，用ldconfig更新）

d.默认的动态库搜索路径/lib和/usr/lib；

* 编译C程序：

gcc -c func1.c func2.c

* 编译生成静态库：libfunc.a

ar -rv libfunc.a func1.o func2.o

* gcc -o test test.c -L/home/zsx/Xavis库函数 -lfunc
* 隐式连接opencv库和libCImage1.a/libCImage1.so库（库位置：/home/zsx/Xavis库函数）

gcc -o test zsx.cpp -L/home/zsx/Xavis库函数 -lCImage1 `pkg-config --cflags --libs opencv` -lstdc++

g++ -o test zsx.cpp -L/home/zsx/Xavis库函数 -lCImage1 `pkg-config --cflags --libs opencv`

* 拷贝libfunc.so到/usr/lib

sudo cp -r /home/zsx/Xavis库函数/zsx/libfunc.so /usr/lib

* 删除文件libfunc.so:

sudo rm -i /usr/lib/libfunc.so

1. **需改进封装基础类**
2. **XBase：**class CIMAGEDLL\_IMPEXPSL XBase:public CObject
3. **XInt：**class CIMAGEDLL\_IMPEXPSL XInt : public XBase
4. **XInts**：class CIMAGEDLL\_IMPEXPSL XInts : public XBase
5. **XDouble**：class CIMAGEDLL\_IMPEXPSL XDouble : public XBase
6. **XDoubles：**class CIMAGEDLL\_IMPEXPSL XDoubles : public XBase
7. **XString：**class CIMAGEDLL\_IMPEXPSL XString : public XBase
8. **XStrings：**class CIMAGEDLL\_IMPEXPSL XStrings : public XBase
9. **XRect：**class CIMAGEDLL\_IMPEXPSL XRect : public XBase（矩阵部分的方法定义需要进行修改。原来Xavis是采取鼠标画框形式，智能相机需改为输入两点（左上角&右下角）坐标形式）
10. **XRects**：class CIMAGEDLL\_IMPEXPSL XRects : public XBase
11. **X2DImage：**class CIMAGEDLL\_IMPEXPSL X2DImage : public XBase
12. 舍弃：X2DImages，X3DImage，X3DImages
13. **无需封装库函数**
14. 基础函数无需封装，如图片读取readimage、变量转换inttodouble等。
15. 程序控制函数无需封装，如if、while等
16. 外设操作函数无需封装，如openframe
17. **待封装函数分类**

**（注：函数查找Xavis未出版书籍：《机器视觉组态软件Xavis》）**

1. **标示输出函数（6个）：**需对函数内容进行更改，难度较大。
2. **图像处理函数（122个）**
3. 图像处理函数（14个）
4. 图像变换函数（40个）
5. 图像融合函数（5个）
6. 阈值分割函数（14个）
7. 直方图处理函数（6个）
8. 连通域处理函数（6个）
9. 形态学处理函数（13个）
10. 仿射变换函数（9个）
11. 参数计算函数（15个）
12. **对象测量与目标检测函数（40个）**
13. 对象测量函数（10个）
14. 边缘检测函数（15个）
15. 特诊检测函数（15个）
16. **模式识别函数（50个）**
17. 目标匹配函数（14个）
18. 目标识别函数（26个）
19. 目标跟踪函数（10个）
20. **相关分类**

**函数分类**

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **函数类名** | **待封装个数** | **难度** |
| 1 | 标示输出函数 | 6 | ★★★★★☆ |
| 2 | 图像处理函数 | 122 | ★★☆ |
| 3 | 对象测量与目标检测函数 | 40 | ★★★☆ |
| 4 | 模式识别函数 | 50 | ★★★☆ |

**建议分工**

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **函数类名** | **待封装个数** | **难度** |
| 1 | 标示输出 | 6 | ★★★★ |
| 2 | 图像处理+图像变换+阈值分割 | 68 | ★★☆ |
| 3 | 连通域处理+形态学处理+仿射变换+参数计算+对象测量 | 53 | ★★★ |
| 4 | 直方图处理+边缘检测+特诊检测 | 36 | ★★★ |
| 5 | 目标匹配+目标跟踪+图像融合 | 29 | ★★★☆ |
| 6 | 目标识别 | 26 | ★★★☆ |