Robot Vision Systems

Lecture 8: Python wrappers in OpenCV

Michael Felsberg

michael.felsberg@liu.se

Why Python Wrappers

- Assume a small library based on OpenCV
- Python interface for
  - Testing
  - Distribution
  - Prototyping
- Similar to OpenCV's Python interface

Workflow

- Write a Python module that provides an interface to your C++ library in Python
- This module is written in C/C++
- Python connection is obtained by **#include "Python.h”**
- Most important: provides **PyObject**

Possible Techniques

- Two options to integrate your OpenCV-based code

1. Use OpenCV's style
   - See modules/python/src2
   - Mostly automatic
   - (Now) documented
2. Use Cython
   - Well documented
   - Semi-automatic

OpenCV Python bindings

Python scripts in modules/python/src2

1. modules/python/CMakeFiles.txt checks the modules to be extended to Python and grabs their header files
2. header files passed to generator script modules/python/src2/gen2.py, which calls modules/python/src2/hdr_parser.py (header parser script)
   - header file -> small Python lists
   - all details about function, class etc.
   - only functions specified with certain macros

OpenCV Python bindings

- gen2.py generates wrapper functions in build/modules/python/pyopencv_generated_*.h
- Some basic OpenCV types and complex classes functions need manual wrappers in modules/python/src2/cv2.cpp
  - e.g. Mat becomes Numpy array, Size two integers
- This is compiled then to build the cv2 module
- Mostly C++ code (thus almost C++ speed)

Wrapper Macros

- CV_EXPORTS_W void integral(InputArray src, OutputArray sum, int sdepth = -1);
- CV_EXPORTS_AS(integral2) void integral(InputArray src, OutputArray sum, OutputArray sqsum, int sdepth = -1, int sqdepth = -1);
- CV_EXPORTS_W void minEnclosingCircle(InputArray points, CV_OUT Point2f& center, CV_OUT float& radius);
- class CV_EXPORTS_W_SIMPLE Dmatch (by v)
- class CV_EXPORTS_W_MAP Moments (native)
- CV_WRAP DMatch();
- CV_PROP / CV_PROP_RW float distance;

Cython

- C-Extensions for Python, cython.org
- Installation as Python package or from homepage
- Workflow:
  - Generate Cython file (Py[classname].pyx)
  - Generate setup Python file (setup.py)
  - Change possibly ARCHFLAGS + dynamic lib paths
  - **python setup.py build_ext –inplace**
    generates Py[classname].cpp and compiles it as .so

Setup file example

- from distutils.core import setup
- from distutils.extension import Extension
- from Cython.Distutils import build_ext
- from Cython.Build import cythonize
- module1 = Extension(*"ChannelBasis"*,
  [*"PyChannelBasis.pyx"*],
  include_dirs=[*"/usr/local/include/","…"*],
  libraries=[*"Channelbasis"*],
  library_dirs=[*"Release"*],
  language=*"c++")*
- setup(cmdclass = {*"build_ext": build_ext*},
  ext_modules = [module1])

The pyx file

- Consists of three parts
  - Header
  - Extern definitions of existing C++ classes
  - Definition of the Python wrapper classes
- Mixture of C/C++ and Python code
- Mainly Python-style (indent etc)
- Many Python keywords exist in two versions (def/cdef, import/cimport, etc)
- Some limitations of, e.g., inheritance

Header example

- # distutils: language = c++
- from libcpp.vector cimport vector
- cimport numpy as np
- from cpython cimport PyObject
- from cv2 import CV_32F
- import numpy as np
- np.import_array()

Definition of C++ classes

- cdef extern from *"opencv2/opencv.hpp" namespace "cv":*
- cdef cppclass Mat:
- Mat()
- Mat(int,int,int,void*)
- int rows
- int cols
- float* data
- int channels()

Extern definition (cont.)

- cdef extern from *"ChannelBasis.h" namespace "cvl":*
- …
- cdef cppclass CombinedChannelBasis:
- CombinedChannelBasis()
- void setParameters(vector[ChannelBasis*])
- cdef cppclass ChannelVector:
- ChannelVector(ChannelBasis*)
- …
- void addSample(Mat)
- ...

Wrapper class definitions

- cdef class **PyChannelBasis:**
- cdef ChannelBasis *thisptr
- def **__cinit__(*self):***
- pass
- …
- cdef class **PyCombinedChannelBasis(PyChannelBasis):**
- def **__cinit__(*self):***
- *self.thisptr = <ChannelBasis*> new CombinedChannelBasis()*
- def **__dealloc__(*self):***
- del *self.thisptr*
- def **setParameters(*self, PychBasisVector):***
- cdef vector[ChannelBasis*] chBasisVector
- cdef ChannelBasis* pdummy
- for x in PychBasisVector:
- pdummy = (<PyChannelBasis>x).thisptr
- chBasisVector.push_back(pdummy)
- (<CombinedChannelBasis*>*self.thisptr).setParameters(chBasisVector)*

Wrapper classes (cont)

- cdef class **PyChannelVector:**
- …
- def **asarray(*self):***
- return np.PyArray_SimpleNewFromData(2,[*self.thisptr.rows,self.thisptr.cols],np.NPY_FLOAT32,self.thisptr.data)*
- def **addSample(*self, vals):***
- if vals.ndim == 2:
- vals = np.array([vals.T]).T
- cdef Mat cvVals
- cdef np.ndarray[np.float32_t, ndim = 3, mode = *'c'] valsa = np.ascontiguousarray(vals,dtype = np.float32)*
- cdef int CVtype
- CVtype = ((CV_32F&7) + ((np.size(vals,2)-1) << 3))
- cvVals = Mat(np.size(vals,0),np.size(vals,1),CVtype,valsa.data)
- *self.thisptr.addSample(cvVals)*

Test Function

- General a good idea: replicate C++-library test function as Python script
- Include your new wrapper
  - import ChannelBasis as cb
- And other required modules
  - import numpy as np
  - import cv2

Example code

- image = cv2.imread(*"house_orig.png"*)
- image = cv2.cvtColor(image,cv2.cv.CV_RGB2GRAY)
- CCBB = cb.PyCos2ChannelBasis()
- CCBB.setParameters(10, 0, 255)
- cCVB = cb.PyChannelVector(CCBB)
- cCVB.addSample(image)
- cCVB.channelImage()[:] =
  cv2.GaussianBlur(cCVB.channelImage(),
  ksize = (7, 7), sigmaX = 1.5, sigmaY = 1.5)
- fImage2 = cCVB.decode(2)
- cv2.imshow(*"Mode0 Image",*
  cv2.convertScaleAbs(fImage2[:,:,0].squeeze()))
- cv2.waitKey(0)