```
# ============================================================================
# Makefile for Simple Input with OpenCV
# ============================================================================

# ----------------------------------------------------------------------------
# Tools
# ----------------------------------------------------------------------------
COMPILER = g++
#COMPILER = mingw32-g++
LINKER = $(COMPILER)
# Listings with Ascii2Postscript & Postscript2PDF
A2PS = a2ps
PS2PDF = ps2pdf -dPDFX=true -sPAPERSIZE=a4
# Documentation
DOCTOOL = doxygen
# OS specific
OSTYPE = $(shell uname -s)
ifeq ($(findstring NT,$(OSTYPE)),NT)
    # Windows systems
    SFX=.exe
    ARCH = zip
    ARCHOPT =
    ARCHEXT = .zip
else
    # Linux, FreeBSD or Darwin or others
    SFX=
    ARCH = tar
    ARCHOPT = -zcvf
    ARCHEXT = .tgz
endif
# current date (used in archive filename)
DATE = $(shell date +%Y-%m-%d)
# Packages manager
PKGCONFIG = pkg-config
# code static analysis
SPLINT = splint

TOOLS = $(COMPILER) $(A2PS) $(DOCTOOL) $(PKGCONFIG) $(ARCH) $(SPLINT)

# ----------------------------------------------------------------------------
# Flags and packages
# ----------------------------------------------------------------------------

# Compilation flags
CFLAGS = -W -Wall -g
#CFLAGS = -O3
# Debug flag definition : -D DEBUG
# Explicit template generation by protoinstanciation : -fno-implicit-templates
# Automatic template generation : -frepo
# Optimize flags : -O2 or -O3
# No cygwin : -mno-cygwin

# linkage flags
LFLAGS =

# common libraries names (i.e.: m for math, z for zlib, ...)
LIBNAMES =
# common libraries linkage flags (i.e.: -lm -lz -l...)
CLIBS = $(foreach name, $(LIBNAMES),-l$(name))

# Used packages list managed by pkg-config
# (check installed packages with pkg-config --list-all)
PACKAGES = opencv
# Compile and Link flags associated with these packages (if any)
#   - INCLUDES are used during compilation step
#   - LIBS are used during linkage step
ifneq ($(PACKAGES),)
    LIBS = $(shell $(PKGCONFIG) --libs $(PACKAGES))
    INCLUDES = -I. $(shell $(PKGCONFIG) --cflags $(PACKAGES))
else
    LIBS =
    INCLUDES = -I.
endif

# Special linkage flags for OpenCV on Windows with MinGW
ifeq ($(findstring NT,$(OSTYPE)), NT)
    ifeq ($(findstring opencv,$(PACKAGES)), opencv)
        # Enable auto import is needed by openCV with windows in order to
        # import symbols from DLLs
        LFLAGS := -Wl,--enable-auto-import $(LFLAGS)
    endif
endif
```



```

# ----------------------------------------------------------------------------
# Project name and sources (check these variables with "make check")
# ----------------------------------------------------------------------------

# Project name (for listing and archive purpose only)
PROJECT = OpenCV Calibration
# Project nature (c or cpp)
EXT=.cpp
# List of classes or modules (couples of .h/.c[pp]) WITHOUT extensions
MODULES =
# List of programs (.c[pp] files containing main function) WITHOUT extensions
MAINS = calibration imagelist_creator readCalibrationMatrix
# List of c or c++ header files
HEADERS = $(foreach name, $(MODULES), $(name).h)
# List of c or c++ source files
SOURCES = $(foreach name, $(MODULES), $(name)$(EXT)) \
$(foreach name, $(MAINS), $(name)$(EXT))
# List of all sources files (Makefile + .h and .c[pp])
ALLSOURCES = Makefile $(foreach name, $(MODULES),$(name).h $(name)$(EXT)) \
$(foreach name, $(MAINS), $(name)$(EXT))
# List of all source files (with .h replaced by .hpp for correct printing with a2ps)
CPPSOURCES = $(ALLSOURCES:.h=.hpp)
# Additional files (docs, readme, etc.)
ADDITIONAL = Doxyfile pattern.pdf
# Object files to be linked
# Classes or Modules objects
MOBJECTS = $(foreach name, $(MODULES), $(name).o)
# Main programs objects
POBJECTS = $(foreach name, $(MAINS), $(name).o)
# Programs to be generated
PROGRAMS = $(POBJECTS:.o=$(SFX))
# PS and PDF listings
LISTING = $(PROJECT)
# All files for the archive
ALLFILES = $(ALLSOURCES) $(ADDITIONAL)

# Phony targets (don't need file check)
.PHONY : clean realclean doc ps pdf archive edit check checkenv
# suffixes
.SUFFIXES : $(EXT).o

# ----------------------------------------------------------------------------
# targets
# ----------------------------------------------------------------------------
all : .depend $(PROGRAMS)

other : pdf doc archive

# Automatic dependencies generation .o: .h .c[pp]
.depend :
    @echo building dependencies ...
    $(COMPILER) -MM $(SOURCES) $(INCLUDES) > .depend
    @echo done.

# dependencies include
-include .depend

# source files compilation generic target
$(EXT).o:
    @echo compiling $< file ...
    $(COMPILER) $(CFLAGS) -c $< $(INCLUDES)
    @echo done.

# programs link generic target
%$(SFX) : %.o $(MOBJECTS)
    @echo Linking Executable $@ ...
    $(LINKER) $(LFLAGS) -o $@ $^ $(LIBS) $(CLIBS)
    @echo done

# run the program(s)
run : $(PROGRAMS)
    @$(foreach prgm, $(PROGRAMS), echo executing $(prgm); $(prgm);)

# cleaning object files, listings, documentation and programs
clean : unlinks
    @echo cleaning obj, listing and doc files ...
#   @echo cleaning *.o *~ .depend $(PROGRAMS) $(LISTING).ps $(LISTING).pdf doc
    rm -rf *.o *~ .depend $(PROGRAMS) $(LISTING).ps $(LISTING).pdf doc
    @echo done.

# cleaning also generated archives
```

```
165 realclean : clean
166     @echo cleaning archives files ...
167 #   @echo cleaning $(PROJECT)-*$(ARCHEXT)
168     rm -f $(PROJECT)-*$(ARCHEXT)
169     @echo done.
170
171 # Creating links for C++ header files
172 links : unlinks
173     @echo creating symbolic links for C++ header files
174     $(foreach header, $(HEADERS), ln -s $(header) $(header)pp;)
175     @echo done.
176
177 # Destroying links for C++ header files
178 unlinks :
179     @echo cleaning symbolic links for C++ header files
180     rm -f *.hpp
181     @echo done.
182
183 # Postscript listing
184 ps : $(ALLSOURCES) links
185     @echo generating Postscript listing ...
186     $(A2PS) -2 --file-align=fill --line-numbers=1 --font-size=10 \
187     --chars-per-line=90 --tabsize=4 --pretty-print --highlight-level=heavy \
188     --prologue="gray" \
189     -o$(LISTING).ps $(CPPSOURCES);
190     @echo done.
191
192 # PDF listing
193 pdf : ps
194     @echo generating PDF listing ...
195     $(PS2PDF) $(LISTING).ps $(LISTING).pdf
196     @echo done.
197
198 # editing files
199 edit :
200     @echo editing files ...
201     @echo editor is : $(EDITOR)
202     "$(EDITOR)" $(ALLFILES) &
203     @echo done.
204
205 # generating documentation in doc folder
206 doc : Doxyfile $(ALLSOURCES)
207     @echo generating documentation ...
208     $(DOCTOOL) Doxyfile
209 #   make -C doc/latex
210     @echo done.
211
212 # backup important files in dated archive
213 archive : pdf $(ALLFILES)
214     @echo generating archive ...
215     $(ARCH) $(ARCHOPT) $(PROJECT)-$(DATE)$(ARCHEXT) $(ALLFILES) $(LISTING).pdf
216     @echo done.
217
218 # Check project variables
219 check :
220     @echo Checking project variables ...
221     @echo project name: $(PROJECT)
222     @echo modules: $(MODULES)
223     @echo main programs: $(MAINS)
224     @echo header files: $(HEADERS)
225     @echo source files: $(SOURCES)
226     @echo All source files: $(ALLSOURCES)
227     @echo All CPP source files: $(CPPSOURCES)
228     @echo Classes and Modules objects: $(MOBJECTS)
229     @echo Main programs objects: $(POBJECTS)
230     @echo Main programs targets: $(PROGRAMS)
231     @echo compile flags: $(CFLAGS) \<file\> $(INCLUDES)
232     @echo link flags: $(LFLAGS) \<output\> $(LIBS) $(CLIBS)
233     @echo done.
234
235 # Check tools versions and locations, and also required packages versions
236 checkenv :
237     @echo required tools : $(TOOLS)
238     @$(foreach tool, $(TOOLS),\
239     echo ------------------------------------------------------------------------;\
240     echo $(tool) version    : `$(tool) --version`;\
241     echo $(tool) location   : `which $(tool)`;)
242     @echo ------------------------------------------------------------------------;
243     @echo required packages : $(PACKAGES)
244     @$(foreach package, $(PACKAGES), echo "$(package) version : " \
245     `$(PKGCONFIG) --modversion $(package)`;)
```



```
1  #include <stdio.h>
2  #include <time.h>
3
4  #include <string>
5
6  #include "opencv2/calib3d/calib3d.hpp"
7  #include "opencv2/core/core.hpp"
8  #include "opencv2/highgui/highgui.hpp"
9  #include "opencv2/imgproc/imgproc.hpp"
10
11 using namespace cv;
12 using namespace std;
13
14 /**
15  * usage sentence
16  */
17 const char * usage =
18       " \nexample command line for calibration from a live feed.\n"
19       "   calibration  -w 4 -h 5 -s 0.025 -o camera.yml -op -oe\n"
20       " \n"
21       " example command line for calibration from a list of stored images:\n"
22       "   imagelist_creator image_list.xml *.png\n"
23       "   calibration -w 4 -h 5 -s 0.025 -o camera.yml -op -oe image_list.xml\n"
24       " where image_list.xml is the standard OpenCV XML/YAML\n"
25       " use imagelist_creator to create the xml or yaml list\n"
26       " file consisting of the list of strings, e.g.:\n"
27       " \n"
28       "<?xml version=\"1.0\"?>\n"
29       "<opencv_storage>\n"
30       "<images>\n"
31       "view000.png\n"
32       "view001.png\n"
33       "<!-- view002.png -->\n"
34       "view003.png\n"
35       "view010.png\n"
36       "one_extra_view.jpg\n"
37       "</images>\n"
38       "</opencv_storage>\n";
39
40 /**
41  * Help displayed at program launch when interactive calibration is on
42  */
43 const char * liveCaptureHelp =
44       "When the live video from camera is used as input, the following hot-keys "
45       "may be used:\n"
46       "  <ESC>, 'q' - quit the program\n"
47       "  'g' - start capturing images\n"
48       "  'u' - switch undistortion on/off\n";
49
50 /**
51  * Help function.
52  * Display :
53  *  - complete arguments description
54  *  - usage sentence
55  *  - help sentence
56  * @see usage
57  * @see liveCaptureHelp
58  */
59 void help()
60 {
61     printf(
62           "This is a camera calibration sample.\n"
63           "Usage: calibration\n"
64           "    -w <board_width>        # the number of inner corners per one of board dimension\n"
65           "    -h <board_height>       # the number of inner corners per another board dimension\n"
66           "    [-n <number_of_frames>]  # the number of frames to use for calibration\n"
67           "                             # (if not specified, it will be set to the number\n"
68           "                             #  of board views actually available)\n"
69           "    [-d <delay>]             # a minimum delay in ms between subsequent attempts to capture a next view\n"
70           "                             # (used only for video capturing)\n"
71           "    [-s <squareSize>]        # square size in some user-defined units (1 by default)\n"
72           "    [-o <out_camera_params>] # the output filename for intrinsic [and extrinsic] parameters\n"
73           "    [-op]                    # write detected feature points\n"
74           "    [-oe]                    # write extrinsic parameters\n"
75           "    [-zt]                    # assume zero tangential distortion\n"
76           "    [-a <aspectRatio>]       # fix aspect ratio (fx/fy)\n"
77           "    [-p]                     # fix the principal point at the center\n"
78           "    [-v]                     # flip the captured images around the horizontal axis\n"
79           "    [-V]                     # use a video file, and not an image list, uses\n"
80           "                             # [input_data] string for the video file name\n"
81           "    [-su]                    # show undistorted images after calibration\n"
82           "    [input_data]             # input data, one of the following:\n"
```

```
            "                         #  - text file with a list of the images of the board\n"
            "                         #    the text file can be generated with imagelist_creator\n"
            "                         #  - name of video file with a video of the board\n"
            "                         # if input_data not specified, a live view from the camera is used\n"
            "    [--device [0|1]]        # internal or external camera device\n"
            "    [--reduce <reduce factor>] # image reduce factor\n"
            "    [-m] || [--manual]      # trigger captures manualy with 'c' key\n"
            "\n");
    printf("\n%s", usage);
    printf("\n%s", liveCaptureHelp);
}

typedef enum { DETECTION = 0, CAPTURING = 1, CALIBRATED } CalibState;

/**
 * Compute reprojection errors from calibrated camera by comparing reprojected
 * object points to image extracted points
 * @param objectPoints 3D object points
 * @param imagePoints 2D image points
 * @param rvecs rotation vectors
 * @param tvecs translation vectors
 * @param cameraMatrix calibrated camera matrix
 * @param distCoeffs distorsion coefficients
 * @param perViewErrors Per View errors ?
 * @return
 */
static double computeReprojectionErrors(
    const vector<vector<Point3f> > & objectPoints,
    const vector<vector<Point2f> > & imagePoints,
    const vector<Mat> & rvecs,
    const vector<Mat> & tvecs,
    const Mat & cameraMatrix,
    const Mat & distCoeffs,
    vector<float> & perViewErrors)
{
    vector<Point2f> imagePoints2;
    int i, totalPoints = 0;
    double totalErr = 0, err;
    perViewErrors.resize(objectPoints.size());

    for (i = 0; i < (int) objectPoints.size(); i++)
    {
        projectPoints(Mat(objectPoints[i]),
                      rvecs[i],
                      tvecs[i],
                      cameraMatrix,
                      distCoeffs,
                      imagePoints2);
        err = norm(Mat(imagePoints[i]), Mat(imagePoints2), CV_L2);
        int n = (int) objectPoints[i].size();
        perViewErrors[i] = (float) std::sqrt(err * err / n);
        totalErr += err * err;
        totalPoints += n;
    }

    return std::sqrt(totalErr / totalPoints);
}

/**
 * Compute chessboard corners from bordSize and squareSize
 * @param boardSize board size (i.e. [6,8])
 * @param squareSize square size on the board (i.e. 30 mm)
 * @param corners inner corner points on the chessboard
 */
static void calcChessboardCorners(Size boardSize,
                                  float squareSize,
                                  vector<Point3f> & corners)
{
    corners.resize(0);

    for (int i = 0; i < boardSize.height; i++)
    {
        for (int j = 0; j < boardSize.width; j++)
        {
            corners.push_back(
                Point3f(float(j * squareSize), float(i * squareSize), 0));
        }
    }
}

/**
 * Run Calibration procedure
```



```
 * @param imagePoints chessboard image points on all views
 * @param imageSize image size
 * @param boardSize board size
 * @param squareSize square size on the chessboard
 * @param aspectRatio image aspect ratio
 * @param flags OpenCV calibration flags.
 *  - CV_CALIB_USE_INTRINSIC_GUESS  1
 *  - CV_CALIB_FIX_ASPECT_RATIO     2
 *  - CV_CALIB_FIX_PRINCIPAL_POINT  4
 *  - CV_CALIB_ZERO_TANGENT_DIST    8
 *  - CV_CALIB_FIX_FOCAL_LENGTH 16
 *  - CV_CALIB_FIX_K1  32
 *  - CV_CALIB_FIX_K2  64
 *  - CV_CALIB_FIX_K3  128
 *  - CV_CALIB_FIX_K4  2048
 *  - CV_CALIB_FIX_K5  4096
 *  - CV_CALIB_FIX_K6  8192
 *  - CV_CALIB_RATIONAL_MODEL 16384
 * @param cameraMatrix 3x3 camera matrix.
 * \f[
 * A = \left(
 *  \begin{array}{ccc}
 *      f_x & 0 & c_x \\
 *      0 & f_y & c_y \\
 *      0 & 0 & 1
 *  \end{array}
 * \right)
 * \f]
 * @param distCoeffs 1x8 distorsion coefficients vector.
 * Such as if
 * \f[
 * \left(
 *  \begin{array}{c}
 *      x \\
 *      y \\
 *      z
 *  \end{array}
 * \right) = R
 * \left(
 *  \begin{array}{c}
 *      X \\
 *      Y \\
 *      Z
 *  \end{array}
 * \right) + t
 * \f]
 * \f$x' = \frac{x}{z}\f$
 *
 * \f$y' = \frac{y}{z}\f$
 *
 * \f$x'' = x' \frac{1+k_1r^2+k_2r^4+k_3r^6}{1+k_4r^2+k_5r^4+k_6r^6}
 * + 2p_1x'y' + p_2(r^2 + 2x'^2)\f$
 *
 * \f$ y'' = y' \frac{1+k_1r^2+k_2r^4+k_3r^6}{1+k_4r^2+k_5r^4+k_6r^6}
 * + 2p_2x'y' + p_1(r^2 + 2y'^2) \f$ where
 * \f$ r^2 = x'^2 + y'^2\f$
 *
 * \f$u = f_x \cdot x'' + c_x\f$
 *
 * \f$v = f_y \cdot y'' + c_y\f$
 * @param rvecs Rotation vectors for each view
 * @param tvecs TRanslation vector for each view
 * @param reprojErrs Points reprojection errors
 * @param totalAvgErr total average error
 * @return true if calibration went right
 */
static bool runCalibration(vector<vector<Point2f> > imagePoints,
                           Size imageSize,
                           Size boardSize,
                           float squareSize,
                           float aspectRatio,
                           int flags,
                           Mat & cameraMatrix,
                           Mat & distCoeffs,
                           vector<Mat> & rvecs,
                           vector<Mat> & tvecs,
                           vector<float> & reprojErrs,
                           double & totalAvgErr)
{
    cameraMatrix = Mat::eye(3, 3, CV_64F);
    if (flags & CV_CALIB_FIX_ASPECT_RATIO)
    {
```

```
247         cameraMatrix.at<double>(0, 0) = aspectRatio;
248     }
249
250     distCoeffs = Mat::zeros(8, 1, CV_64F);
251
252     vector<vector<Point3f> > objectPoints(1);
253     calcChessboardCorners(boardSize, squareSize, objectPoints[0]);
254
255     objectPoints.resize(imagePoints.size(), objectPoints[0]);
256
257     double rms = calibrateCamera(objectPoints,
258                                  imagePoints,
259                                  imageSize,
260                                  cameraMatrix,
261                                  distCoeffs,
262                                  rvecs,
263                                  tvecs,
264                                  flags | CV_CALIB_FIX_K4 | CV_CALIB_FIX_K5);
265     ///*|CV_CALIB_FIX_K3*/|CV_CALIB_FIX_K4|CV_CALIB_FIX_K5);
266     printf("RMS error reported by calibrateCamera: %g\n", rms);
267
268     bool ok = checkRange(cameraMatrix) ∧ checkRange(distCoeffs);
269
270     totalAvgErr = computeReprojectionErrors(objectPoints,
271                                             imagePoints,
272                                             rvecs,
273                                             tvecs,
274                                             cameraMatrix,
275                                             distCoeffs,
276                                             reprojErrs);
277
278     return ok;
279 }
280
281 /**
282  * Save camera calibration matrix to file
283  * @param filename file name to save data
284  * @param imageSize image size
285  * @param boardSize board size
286  * @param squareSize board squares size
287  * @param aspectRatio aspect ratio
288  * @param flags CV calibration flags
289  * @param cameraMatrix camera matrix
290  * @param distCoeffs distorsion coefficients
291  * @param rvecs rotation vectors
292  * @param tvecs translation vectors
293  * @param reprojErrs reprojection errors
294  * @param imagePoints image points
295  * @param totalAvgErr tota average error
296  */
297 void saveCameraParams(const string & filename,
298                       Size imageSize,
299                       Size boardSize,
300                       float squareSize,
301                       float aspectRatio,
302                       int flags,
303                       const Mat & cameraMatrix,
304                       const Mat & distCoeffs,
305                       const vector<Mat> & rvecs,
306                       const vector<Mat> & tvecs,
307                       const vector<float> & reprojErrs,
308                       const vector<vector<Point2f> > & imagePoints,
309                       double totalAvgErr)
310 {
311     FileStorage fs(filename, FileStorage::WRITE);
312
313     time_t t;
314     time(&t);
315     struct tm * t2 = localtime(&t);
316     char buf[1024];
317     strftime(buf, sizeof(buf) - 1, "%c", t2);
318
319     fs << "calibration_time" << buf;
320
321     if (¬rvecs.empty() ∨ ¬reprojErrs.empty())
322     {
323         fs << "nframes" << (int) std::max(rvecs.size(), reprojErrs.size());
324     }
325     fs << "image_width" << imageSize.width;
326     fs << "image_height" << imageSize.height;
327     fs << "board_width" << boardSize.width;
328     fs << "board_height" << boardSize.height;
```



```
329     fs << "square_size" << squareSize;
330
331     if (flags & CV_CALIB_FIX_ASPECT_RATIO)
332         fs << "aspectRatio" << aspectRatio;
333
334     if (flags ≠ 0)
335     {
336         sprintf(buf,
337                 "flags: %s%s%s%s",
338                 flags & CV_CALIB_USE_INTRINSIC_GUESS ? "+use_intrinsic_guess" : "",
339                 flags & CV_CALIB_FIX_ASPECT_RATIO ? "+fix_aspectRatio" : "",
340                 flags & CV_CALIB_FIX_PRINCIPAL_POINT ? "+fix_principal_point" : "",
341                 flags & CV_CALIB_ZERO_TANGENT_DIST ? "+zero_tangent_dist" : "");
342         cvWriteComment(*fs, buf, 0);
343     }
344
345     fs << "flags" << flags;
346
347     fs << "camera_matrix" << cameraMatrix;
348     fs << "distortion_coefficients" << distCoeffs;
349
350     fs << "avg_reprojection_error" << totalAvgErr;
351     if (¬reprojErrs.empty())
352     {
353         fs << "per_view_reprojection_errors" << Mat(reprojErrs);
354     }
355
356     if (¬rvecs.empty() ∧ ¬tvecs.empty())
357     {
358         Mat bigmat((int) rvecs.size(), 6, CV_32F);
359         for (int i = 0; i < (int) rvecs.size(); i++)
360         {
361             Mat r = bigmat(Range(i, i + 1), Range(0, 3));
362             Mat t = bigmat(Range(i, i + 1), Range(3, 6));
363             rvecs[i].copyTo(r);
364             tvecs[i].copyTo(t);
365         }
366         cvWriteComment(*fs,
367                        "a set of 6-tuples (rotation vector + translation "
368                        "vector) for each view",
369                        0);
370         fs << "extrinsic_parameters" << bigmat;
371     }
372
373     if (¬imagePoints.empty())
374     {
375         Mat imagePtMat(
376             (int) imagePoints.size(), imagePoints[0].size(), CV_32FC2);
377         for (int i = 0; i < (int) imagePoints.size(); i++)
378         {
379             Mat r = imagePtMat.row(i).reshape(2, imagePtMat.cols);
380             Mat imgpti(imagePoints[i]);
381             imgpti.copyTo(r);
382         }
383         fs << "image_points" << imagePtMat;
384     }
385 }
386
387 /**
388  * Read string list from FileStorage
389  * @param filename the file name to read
390  * @param l string vector
391  * @return true if everything went right, false otherwise
392  */
393 static bool readStringList(const string & filename, vector<string> & l)
394 {
395     l.resize(0);
396     FileStorage fs(filename, FileStorage::READ);
397     if (¬fs.isOpened())
398     {
399         return false;
400     }
401     FileNode n = fs.getFirstTopLevelNode();
402     if (n.type() ≠ FileNode::SEQ)
403     {
404         return false;
405     }
406     FileNodeIterator it = n.begin(), it_end = n.end();
407     for (; it ≠ it_end; ++it)
408     {
409         l.push_back((string) *it);
410     }
```

```
411     return true;
412 }
413
414 /**
415  * Run Calibration and save results to file
416  * @param outputFilename output file name
417  * @param imagePoints image points for each view
418  * @param imageSize image size
419  * @param boardSize board size (in inner corner numbers)
420  * @param squareSize board square size
421  * @param aspectRatio aspect ratio
422  * @param flags CV calibration flags
423  * @param cameraMatrix camera calibration matrix
424  * @param distCoeffs distorsion coefficients
425  * @param writeExtrinsics Also write extrinsic parameters to file
426  * @param writePoints Also write points to file
427  * @return true if calibration have been performed and results saved to file,
428  * false otherwise
429  */
430 bool runAndSave(const string & outputFilename,
431                 const vector<vector<Point2f> > & imagePoints,
432                 Size imageSize,
433                 Size boardSize,
434                 float squareSize,
435                 float aspectRatio,
436                 int flags,
437                 Mat & cameraMatrix,
438                 Mat & distCoeffs,
439                 bool writeExtrinsics,
440                 bool writePoints)
441 {
442     vector<Mat> rvecs, tvecs;
443     vector<float> reprojErrs;
444     double totalAvgErr = 0;
445
446     bool ok = runCalibration(imagePoints,
447                              imageSize,
448                              boardSize,
449                              squareSize,
450                              aspectRatio,
451                              flags,
452                              cameraMatrix,
453                              distCoeffs,
454                              rvecs,
455                              tvecs,
456                              reprojErrs,
457                              totalAvgErr);
458     printf("%s. avg reprojection error = %.2f\n",
459            ok ? "Calibration succeeded" : "Calibration failed",
460            totalAvgErr);
461
462     if (ok)
463     {
464         saveCameraParams(outputFilename,
465                          imageSize,
466                          boardSize,
467                          squareSize,
468                          aspectRatio,
469                          flags,
470                          cameraMatrix,
471                          distCoeffs,
472                          writeExtrinsics ? rvecs : vector<Mat>(),
473                          writeExtrinsics ? tvecs : vector<Mat>(),
474                          writeExtrinsics ? reprojErrs : vector<float>(),
475                          writePoints ? imagePoints : vector<vector<Point2f> >(),
476                          totalAvgErr);
477     }
478     return ok;
479 }
480
481 /**
482  * Calibration Main program
483  * @param argc argument count
484  * @param argv arguments values
485  * @return 0 if arguments are missing or if everything went right. return -1
486  * in all error cases
487  */
488 int main(int argc, char ** argv)
489 {
490     Size boardSize, imageSize;
491     float squareSize = 1.f, aspectRatio = 1.f;
492     Mat cameraMatrix, distCoeffs;
```



```
493     const char * outputFilename = "out_camera_data.yml";
494     const char * inputFilename = 0;
495
496     int i, nframes = 10;
497     bool writeExtrinsics = false, writePoints = false;
498     bool undistortImage = false;
499     int flags = 0;
500     VideoCapture capture;
501     bool flipVertical = false;
502     bool showUndistorted = false;
503     bool videofile = false;
504     int delay = 1000;
505     clock_t prevTimestamp = 0;
506     CalibState mode = DETECTION;
507     int cameraId = 0;
508     int reduceFactor = 1;
509     vector<vector<Point2f> > imagePoints;
510     vector<string> imageList;
511     bool manualTrigger = false;
512     int key;
513
514     if (argc < 2)
515     {
516         help();
517         return 0;
518     }
519
520     for (i = 1; i < argc; i++)
521     {
522         const char * s = argv[i];
523         if (strcmp(s, "-w") == 0)
524         {
525             if (sscanf(argv[++i], "%u", &boardSize.width) != 1 ||
526                 boardSize.width <= 0)
527             {
528                 return fprintf(stderr, "Invalid board width\n"), -1;
529             }
530         }
531         else if (strcmp(s, "-h") == 0)
532         {
533             if (sscanf(argv[++i], "%u", &boardSize.height) != 1 ||
534                 boardSize.height <= 0)
535             {
536                 return fprintf(stderr, "Invalid board height\n"), -1;
537             }
538         }
539         else if (strcmp(s, "-s") == 0)
540         {
541             if (sscanf(argv[++i], "%f", &squareSize) != 1 || squareSize <= 0)
542             {
543                 return fprintf(stderr, "Invalid board square width\n"), -1;
544             }
545         }
546         else if (strcmp(s, "-n") == 0)
547         {
548             if (sscanf(argv[++i], "%u", &nframes) != 1 || nframes <= 3)
549             {
550                 return printf("Invalid number of images\n"), -1;
551             }
552         }
553         else if (strcmp(s, "-a") == 0)
554         {
555             if (sscanf(argv[++i], "%f", &aspectRatio) != 1 || aspectRatio <= 0)
556             {
557                 return printf("Invalid aspect ratio\n"), -1;
558             }
559             flags |= CV_CALIB_FIX_ASPECT_RATIO;
560         }
561         else if (strcmp(s, "-d") == 0)
562         {
563             if (sscanf(argv[++i], "%u", &delay) != 1 || delay <= 0)
564             {
565                 return printf("Invalid delay\n"), -1;
566             }
567         }
568         else if (strcmp(s, "-op") == 0)
569         {
570             writePoints = true;
571         }
572         else if (strcmp(s, "-oe") == 0)
573         {
574             writeExtrinsics = true;
```

```
575         }
576         else if (strcmp(s, "-zt") == 0)
577         {
578             flags |= CV_CALIB_ZERO_TANGENT_DIST;
579         }
580         else if (strcmp(s, "-p") == 0)
581         {
582             flags |= CV_CALIB_FIX_PRINCIPAL_POINT;
583         }
584         else if (strcmp(s, "-v") == 0)
585         {
586             flipVertical = true;
587         }
588         else if (strcmp(s, "-V") == 0)
589         {
590             videofile = true;
591         }
592         else if (strcmp(s, "-o") == 0)
593         {
594             outputFilename = argv[++i];
595         }
596         else if (strcmp(s, "-su") == 0)
597         {
598             showUndistorted = true;
599         }
600         else if (strcmp(s, "--device") == 0)
601         {
602             sscanf(argv[++i], "%d", &cameraId);
603             printf("Scanned camera Id = %d\n", cameraId);
604             if (cameraId < 0)
605             {
606                 fprintf(stderr, "wrong camera Id : %d\n", cameraId);
607                 cameraId = 0;
608             }
609         }
610         else if (strcmp(s, "--reduce") == 0)
611         {
612             sscanf(argv[++i], "%d", &reduceFactor);
613             if (reduceFactor ≤ 0)
614             {
615                 fprintf(stderr, "wrong reduce factor 1/%d\n", reduceFactor);
616                 reduceFactor = 1;
617             }
618         }
619         // Scan all arguments not starting with -
620         else if (s[0] ≠ '-')
621         {
622             if (isdigit(s[0]))
623             {
624                 sscanf(s, "%d", &cameraId);
625             }
626             else
627             {
628                 inputFilename = s;
629             }
630         }
631         else if ((strcmp(s, "-m") == 0) ∨ (strcmp(s, "--manual") == 0))
632         {
633             manualTrigger = true;
634         }
635         else
636         {
637             return fprintf(stderr, "Unknown option %s", s), -1;
638         }
639     }
640
641     printf("Required camera Id is %d\n", cameraId);
642
643     if (inputFilename)
644     {
645         if (¬videofile ∧ readStringList(inputFilename, imageList))
646         {
647             mode = CAPTURING;
648         }
649         else
650         {
651             capture.open(inputFilename);
652         }
653     }
654     else
655     {
656         capture.open(cameraId);
```



```
657     }
658
659     if (¬capture.isOpened() ∧ imageList.empty())
660     {
661         return fprintf(stderr, "Could not initialize video capture\n"), -2;
662     }
663
664     if (¬imageList.empty())
665     {
666         nframes = (int) imageList.size();
667     }
668
669     if (capture.isOpened())
670     {
671         printf("%s", liveCaptureHelp);
672     }
673
674     namedWindow("Image View", CV_WINDOW_AUTOSIZE | CV_GUI_NORMAL);
675
676     for (i = 0;; i++)
677     {
678         Mat view, viewGray;
679         bool blink = false;
680
681         if (capture.isOpened())
682         {
683             Mat view0;
684             capture >> view0;
685             if (reduceFactor ≠ 1)
686             {
687                 resize(
688                     view0,
689                     view,
690                     Size(view0.cols / reduceFactor, view0.rows / reduceFactor),
691                     0,
692                     0,
693                     INTER_AREA);
694             }
695             else
696             {
697                 view0.copyTo(view);
698             }
699         }
700         else if (i < (int) imageList.size())
701         {
702             view = imread(imageList[i], 1);
703         }
704
705         if (¬view.data)
706         {
707             if (imagePoints.size() > 0)
708             {
709                 runAndSave(outputFilename,
710                            imagePoints,
711                            imageSize,
712                            boardSize,
713                            squareSize,
714                            aspectRatio,
715                            flags,
716                            cameraMatrix,
717                            distCoeffs,
718                            writeExtrinsics,
719                            writePoints);
720             }
721             break;
722         }
723
724         imageSize = view.size();
725
726         if (flipVertical)
727         {
728             flip(view, view, 0);
729         }
730
731         vector<Point2f> pointbuf;
732         cvtColor(view, viewGray, CV_BGR2GRAY);
733
734         bool found = findChessboardCorners(view,
735                                            boardSize,
736                                            pointbuf,
737                                            CV_CALIB_CB_ADAPTIVE_THRESH &
738                                            CV_CALIB_CB_FAST_CHECK &
```

```
                                        CV_CALIB_CB_NORMALIZE_IMAGE);

        // improve the found corners' coordinate accuracy
        if (found)
        {
            cornerSubPix(viewGray,
                         pointbuf,
                         Size(11, 11),
                         Size(-1, -1),
                         TermCriteria(CV_TERMCRIT_EPS + CV_TERMCRIT_ITER,
                                      30,
                                      0.1));
        }

        bool trigger;
        if (manualTrigger)
        {
            trigger = (key == 'c');
        }
        else
        {
            trigger = clock() - prevTimestamp > delay * 1e-3 * CLOCKS_PER_SEC;
        }

        if (mode == CAPTURING &&
            found &&
            (¬capture.isOpened() ∨
             trigger))
        {
            imagePoints.push_back(pointbuf);
            prevTimestamp = clock();
            blink = capture.isOpened();
        }

        if (found)
        {
            drawChessboardCorners(view, boardSize, Mat(pointbuf), found);
        }

        string msg = mode == CAPTURING ?
                     "100/100" :
                     mode == CALIBRATED ? "Calibrated" : "Press 'g' to start";
        int baseLine = 0;
        Size textSize = getTextSize(msg, 1, 1, 1, &baseLine);
        Point textOrigin(view.cols - 2 * textSize.width - 10,
                         view.rows - 2 * baseLine - 10);

        if (mode == CAPTURING)
        {
            if (undistortImage)
            {
                msg = format("%d/%d Undist", (int) imagePoints.size(), nframes);
            }
            else
            {
                msg = format("%d/%d", (int) imagePoints.size(), nframes);
            }
        }

        putText(view,
                msg,
                textOrigin,
                1,
                1,
                mode ≠ CALIBRATED ? Scalar(0, 0, 255) : Scalar(0, 255, 0));

        if (blink)
        {
            bitwise_not(view, view);
        }

        if (mode == CALIBRATED ∧ undistortImage)
        {
            Mat temp = view.clone();
            undistort(temp, view, cameraMatrix, distCoeffs);
        }

        imshow("Image View", view);
        key = 0xff & waitKey(capture.isOpened() ? 50 : 500);

        if ((key & 255) == 27)
        {
```



```
            break;
        }

        if (key == 'u' ∧ mode == CALIBRATED)
        {
            undistortImage = ¬undistortImage;
        }

        if (capture.isOpened() ∧ key == 'g')
        {
            mode = CAPTURING;
            imagePoints.clear();
        }

        if (mode == CAPTURING ∧ imagePoints.size() ≥ (unsigned) nframes)
        {
            if (runAndSave(outputFilename,
                           imagePoints,
                           imageSize,
                           boardSize,
                           squareSize,
                           aspectRatio,
                           flags,
                           cameraMatrix,
                           distCoeffs,
                           writeExtrinsics,
                           writePoints))
            {
                mode = CALIBRATED;
            }
            else
            {
                mode = DETECTION;
            }
            if (¬capture.isOpened())
            {
                break;
            }
        }
    }

    if (¬capture.isOpened() ∧ showUndistorted)
    {
        Mat view, rview, map1, map2;
        initUndistortRectifyMap(cameraMatrix,
                                distCoeffs,
                                Mat(),
                                getOptimalNewCameraMatrix(cameraMatrix,
                                                          distCoeffs,
                                                          imageSize,
                                                          1,
                                                          imageSize,
                                                          0),
                                imageSize,
                                CV_16SC2,
                                map1,
                                map2);

        for (i = 0; i < (int) imageList.size(); i++)
        {
            view = imread(imageList[i], 1);
            if (¬view.data)
            {
                continue;
            }
            // undistort( view, rview, cameraMatrix, distCoeffs, cameraMatrix );
            remap(view, rview, map1, map2, INTER_LINEAR);
            imshow("Image View", rview);
            int c = 0xff & waitKey();
            if ((c & 255) == 27 ∨ c == 'q' ∨ c == 'Q')
            {
                break;
            }
        }
    }

    return 0;
}
```

mar 09, 16 16:13 **imagelist_creator.cpp** Page 1/1

```cpp
/*this creates a yaml or xml list of files from the command line args
 */

#include <string>
#include <iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>

using std::string;
using std::cout;
using std::endl;

using namespace cv;

void help(char** av)
{
    cout
            << "\nThis creates a yaml or xml list of files from the command line args\n"
             "usage:\n./" << av[0] << " imagelist.yaml *.png\n"
            << "Try using different extensions.(e.g. yaml yml xml xml.gz etc...)\n"
            << "This will serialize this list of images or whatever with opencv's FileStorage framework"
            << endl;
}

int main(int ac, char** av)
{
    if (ac < 3)
    {
        help(av);
        return 1;
    }

    string outputname = av[1];

    Mat m = imread(outputname); //check if the output is an image - prevent overwrites!
    if (¬m.empty())
    {
        std::cerr
                << "fail! Please specify an output file, don't want to overwrite your images!"
                << endl;
        help(av);
        return 1;
    }

    FileStorage fs(outputname, FileStorage::WRITE);
    fs << "images" << "[";
    for (int i = 2; i < ac; i++)
    {
        fs << string(av[i]);
    }
    fs << "]";
    return 0;
}
```

mar 09, 16 16:11 **readCalibrationMatrix.cpp** Page 1/1

```cpp
/*
 * readCalibrationMatrix.cpp
 *
 *  Created on: 2 avr. 2011
 *      Author: davidroussel
 */

#include <iostream>
#include <opencv2/core/core.hpp>

using namespace std;
using namespace cv;

ostream & usage (ostream & os, char * name)
{
    os << "usage : " << name << " <calib_camera_data_file.yaml>" << endl;
    return os;
}

int main (int argc, char ** argv)
{
    string filename;
    FileStorage * fs = NULL;

    // ------------------------------------------------------------------
    // parse arguments
    // ------------------------------------------------------------------
    if (argc < 2)
    {
        cerr << usage(cerr, argv[0]);
    }
    else
    {
        filename = argv[1];
    }
    // ------------------------------------------------------------------
    // search for calibration matrix in file
    // ------------------------------------------------------------------
    fs = new FileStorage(filename, FileStorage::READ);
    if (¬fs→isOpened())
    {
        cerr << "Failed to open FileStorage : " << filename << endl;
        return EXIT_FAILURE;
    }

    Mat cameraMatrix;
    (*fs)["camera_matrix"] >> cameraMatrix;

    cout << "matrix size = ["<< cameraMatrix.rows << "x" << cameraMatrix.cols
         << "]" << endl;
    cout << "matrix element size = " << cameraMatrix.elemSize() << endl;
    cout << "Camera matrix = " << cameraMatrix << endl;

    // ------------------------------------------------------------------
    // Explain calibration matrix parameters
    // ------------------------------------------------------------------
    delete fs;
    return EXIT_SUCCESS;
}
```