Omnidirectional Cameara Calibration {#tutorial_omnidir_calib_main}

This module includes calibration, rectification and stereo reconstruction of omnidirectional camearas. The camera model is described in this paper:

C. Mei and P. Rives, Single view point omnidirectional camera calibration from planar grids, in ICRA 2007.

The model is capable of modeling catadioptric cameras and fisheye cameras, which may both have very large field of view.

The implementation of the calibration part is based on Li's calibration toolbox:

B. Li, L. Heng, K. Kevin and M. Pollefeys, "A Multiple-Camera System Calibration Toolbox Using A Feature Descriptor-Based Calibration Pattern", in IROS 2013.

This tutorial will introduce the following parts of omnidirectional camera calibartion module:

- calibrate a single camera.
- calibrate a stereo pair of cameras.
- rectify images so that large distoration is removed.
- reconstruct 3D from two stereo images, with large filed of view.
- comparison with fisheye model in opencv/calib3d/

Single Camera Calibration

The first step to calibrate camera is to get a calibration pattern and take some photos. Several kinds of patterns are supported by OpenCV, like checkerborad and circle grid. A new pattern named random pattern can also be used, you can refer to opencv_contrib/modules/ccalib for more details.

Next step is to extract corners from calibration pattern. For checkerboard, use OpenCV function `cv::findChessboardCorners`; for circle grid, use `cv::findCirclesGrid`, for random pattern, use the `randomPatternCornerFinder` class in opencv_contrib/modules/ccalib/src/randomPattern.hpp. Save the positions of corners in images in a variable like `imagePoints`. The type of `imagePoints` may be `std::vector<std::vector<cv::Vec2f>>`, the first vector stores corners in each frame, the second vector stores corners in an individual frame. The type can also be `std::vector<cv::Mat>` where the `cv::Mat` is `CV_32FC2`.

Also, the corresponding 3D points in world (pattern) coordinate are required. You can compute they for yourself if you know the physical size of your pattern. Save 3D points in `objectPoints`, similar to `imagePoints`, it can be `std::vector<std::vector<Vec3f>>` or `std::vector<cv::Mat>` where `cv::Mat` is of type `CV_32FC3`. Note the size of `objectPoints` and `imagePoints` must be the same because they are corresponding to each other.

Another thing you should input is the size of images. The file opencv_contrib/modules/ccalib/tutorial/data/omni_calib_data.xml stores an example of objectPoints, imagePoints and imageSize. Use the following code to load them:

```
cv::FileStorage fs("omni_calib_data.xml", cv::FileStorage::READ);
std::vector<cv::Mat> objectPoints, imagePoints;
cv::Size imgSize;
fs["objectPoints"] >> objectPoints;
fs["imagePoints"] >> imagePoints;
fs["imageSize"] >> imgSize;
```

Then define some variables to store the output parameters and run the calibration function like:

```
cv::Mat K, xi, D, idx;
int flags = 0;
cv::TermCriteria critia(cv::TermCriteria::COUNT + cv::TermCriteria::EPS,
200, 0.0001);
std::vector<cv::Mat> rvecs, tvecs;
double rms = cv::omnidir::calibrate(objectPoints, imagePoints, imgSize, K,
xi, D, rvecs, tvecs, flags, critia, idx);
```

`K`, `xi`, `D` are internal parameters and `rvecs`, `tvecs` are external parameters that store the pose of patterns. All of them have depth of `CV_64F`. The `xi` is a single value variable of Mei's model. `idx` is a `CV_32S` Mat that stores indices of images that are really used in calibration. This is due to some images are failed in the initialization step so they are not used in the final optimization. The returned value *rms* is the root mean square of reprojection errors.

The calibration supports some features, *flags* is a enumeration for some features, including:

- cv::omnidir::CALIB_FIX_SKEW
- cv::omnidir::CALIB_FIX_K1
- cv::omnidir::CALIB_FIX_K2
- cv::omnidir::CALIB_FIX_P1
- cv::omnidir::CALIB_FIX_P2
- cv::omnidir::CALIB_FIX_XI
- cv::omnidir::CALIB_FIX_GAMMA
- cv::omnidir::CALIB_FIX_CENTER

Your can specify `flags` to fix parameters during calibration. Use 'plus' operator to set multiple features. For example, `CALIB_FIX_SKEW+CALIB_FIX_K1` means fixing skew and K1.

`criteria` is the stopping criteria during optimization, set it to be, for example, cv::TermCriteria(cv::TermCriteria::COUNT + cv::TermCriteria::EPS, 200, 0.0001), which means using 200 iterations and stopping when relative change is smaller than 0.0001.

Stereo Calibration

Stereo calibration is to calibrate two cameras together. The output parameters include camera parameters of two cameras and the relative pose of them. To recover the relative pose, two cameras must observe the same pattern at the same time, so the `objectPoints` of two cameras are the same.

Now detect image corners for both cameras as discussed above to get `imagePoints1` and `imagePoints2`. Then compute the shared `objectPoints`.

An example of of stereo calibration data is stored in opencv_contrib/modules/ccalib/tutorial/data/omni_stereocalib_data.xml. Load the data by

```
cv::FileStorage fs("omni_stereocalib_data.xml", cv::FileStorage::READ);
std::vector<cv::Mat> objectPoints, imagePoints1, imagePoints2;
cv::Size imgSize1, imgSize2;
fs["objectPoints"] >> objectPoints;
fs["imagePoints1"] >> imagePoints1;
fs["imagePoints2"] >> imagePoints2;
fs["imageSize1"] >> imgSize1;
fs["imageSize2"] >> imgSize2;
```

Then do stereo calibration by

```
cv::Mat K1, K2, xi1, xi2, D1, D2;
int flags = 0;
cv::TermCriteria critia(cv::TermCriteria::COUNT + cv::TermCriteria::EPS,
200, 0.0001);
std::vector<cv::Mat> rvecsL, tvecsL;
cv::Mat rvec, tvec;
double rms = cv::omnidir::stereoCalibrate(objectPoints, imagePoints1,
imagePoints2, imgSize1, imgSize2, K1, xi1, D1, K2, xi2, D2, rvec, tvec,
rvecsL, tvecsL, flags, critia, idx);
```

Here `rvec` and `tvec` are the transform between the first and the second camera. `rvecsL` and `tvecsL` are the transforms between patterns and the first camera.

Image Rectificaiton

Omnidirectional images have very large distortion, so it is not compatible with human's eye balls. For better view, rectification can be applied if camera parameters are known. Here is an example of omnidirectional image of 360 degrees of horizontal field of view.

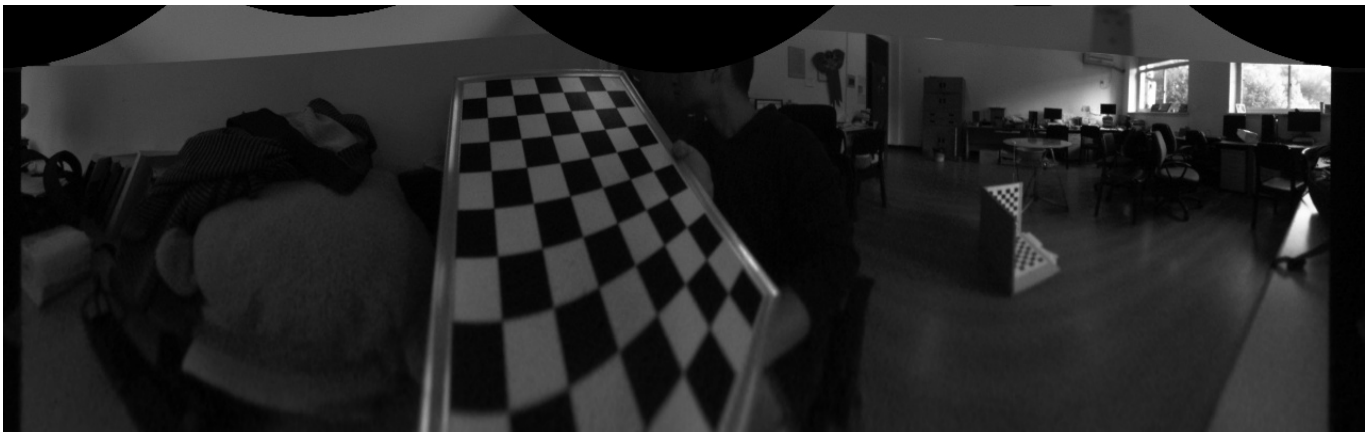
After rectification, a perspective like view is generated. Here is one example to run image rectification in this module:

```
cv::omnidir::undistortImage(distorted, undistorted, K, D, xi, int flags,
Knew, new_size)
```

The variable *distorted* and *undistorted* are the origional image and rectified image perspectively. *K*, *D*, *xi* are camera parameters. *KNew* and *new_size* are the camera matrix and image size for rectified image. *flags* is the rectification type, it can be:

- RECTIFY_PERSPECTIVE: rectify to perspective images, which will lose some filed of view.
- RECTIFY_CYLINDRICAL: rectify to cylindrical images that preserve all view.
- RECTIFY_STEREOGRAPHIC: rectify to stereographic images that may lose a little view.
- RECTIFY_LONGLATI: rectify to longitude-latitude map like a world map of the earth. This rectification can be used to stereo reconstruction but may not be friendly for view. This map is described in paper: *Li S. Binocular spherical stereo[J]. Intelligent Transportation Systems, IEEE Transactions on, 2008, 9(4): 589-600.*

The following four images are four types of rectified images discribed above:



It can be observed that perspective rectified image perserves only a little field of view and is not goodlooking. Cylindrical rectification preserves all field of view and scene is unnatural only in the middle of bottom. The distortion of stereographic in the middle of bottom is smaller than cylindrical but the distortion of other places are larger, and it can not preserve all field of view. For images with very large distortion, the longitude-latitude rectification does not give a good result, but it is available to make epipolar constraint in a line so that stereo matching can be applied in omnidirectional images.

Note: To have a better result, you should carefully choose `Knew` and it is related to your camera. In general, a smaller focal length leads to a smaller field of view and vice versa. Here are recommonded settings.

For RECTIFY_PERSPECTIVE

```
Knew = Matx33f(new_size.width/4, 0, new_size.width/2,
               0, new_size.height/4, new_size.height/2,
               0, 0, 1);
```

For RECTIFY_CYLINDRICAL, RECTIFY_STEREOGRAPHIC, RECTIFY_LONGLATI

```
Knew = Matx33f(new_size.width/3.1415, 0, 0,
               0, new_size.height/3.1415, 0,
               0, 0, 1);
```

Maybe you need to change `(u0, v0)` to get a better view.

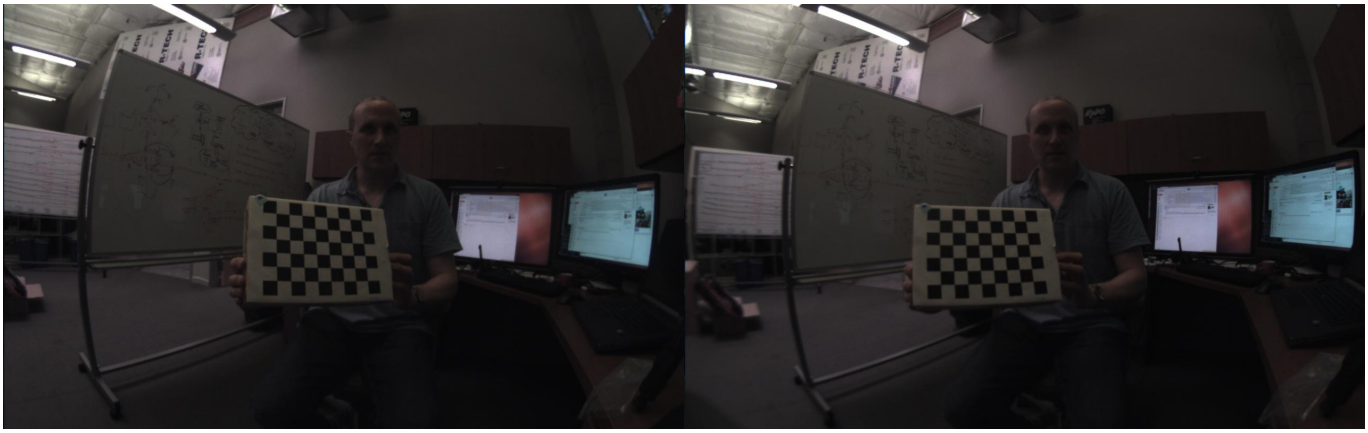
Stereo Reconstruction

Stereo reconstruction is to reconstruct 3D points from a calibrated stereo camera pair. It is a basic problem of computer vison. However, for omnidirectional camera, it is not very popular because of the large distortion make it a little difficult. Conventional methods rectify images to perspective ones and do stereo reconstruction in perspective images. However, the last section shows that recifying to perspective images lose too much field of view, which waste the advantage of omnidirectional camera, i.e. large field of view.

The first step of stereo reconstruction is stereo rectification so that epipolar lines are horizontal lines. Here, we use longitude-latitude rectification to preserve all filed of view, or perspective rectification which is available but is not recommended. The second step is stereo matching to get a disparity map. At last, 3D points can be generated from disparity map.

The API of stereo reconstruction for omnidrectional camera is `omnidir::stereoReconstruct`. Here we use an example to show how it works.

First, calibrate a stereo pair of cameras as described above and get parameters like `K1`, `D1`, `xi1`, `K2`, `D2`, `xi2`, `rvec`, `tvec`. Then read two images from the first and second camera respectively, for instance, `image1` and `image2`, which are shown below.

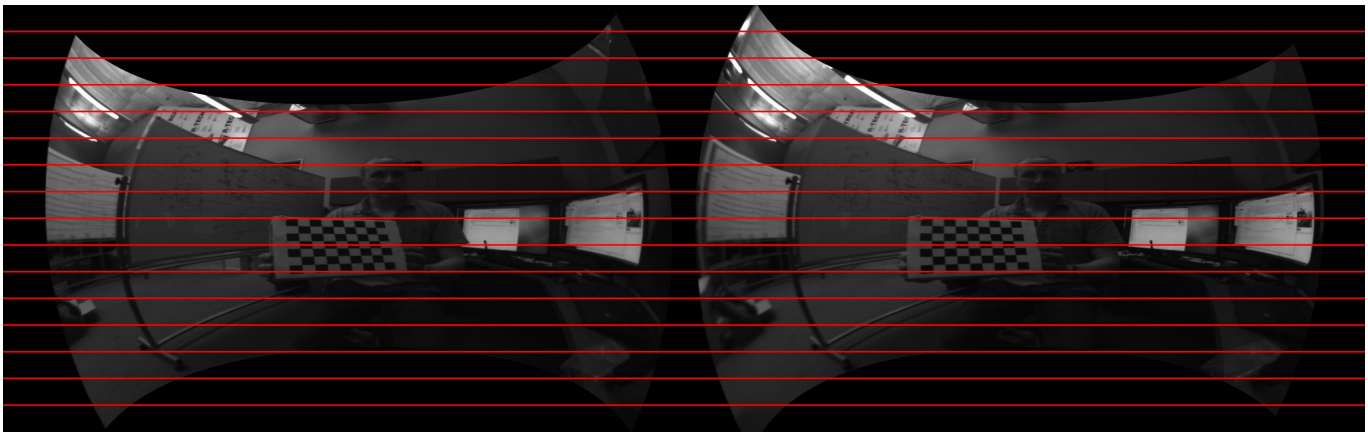


Second, run `omnidir::stereoReconstruct` like:

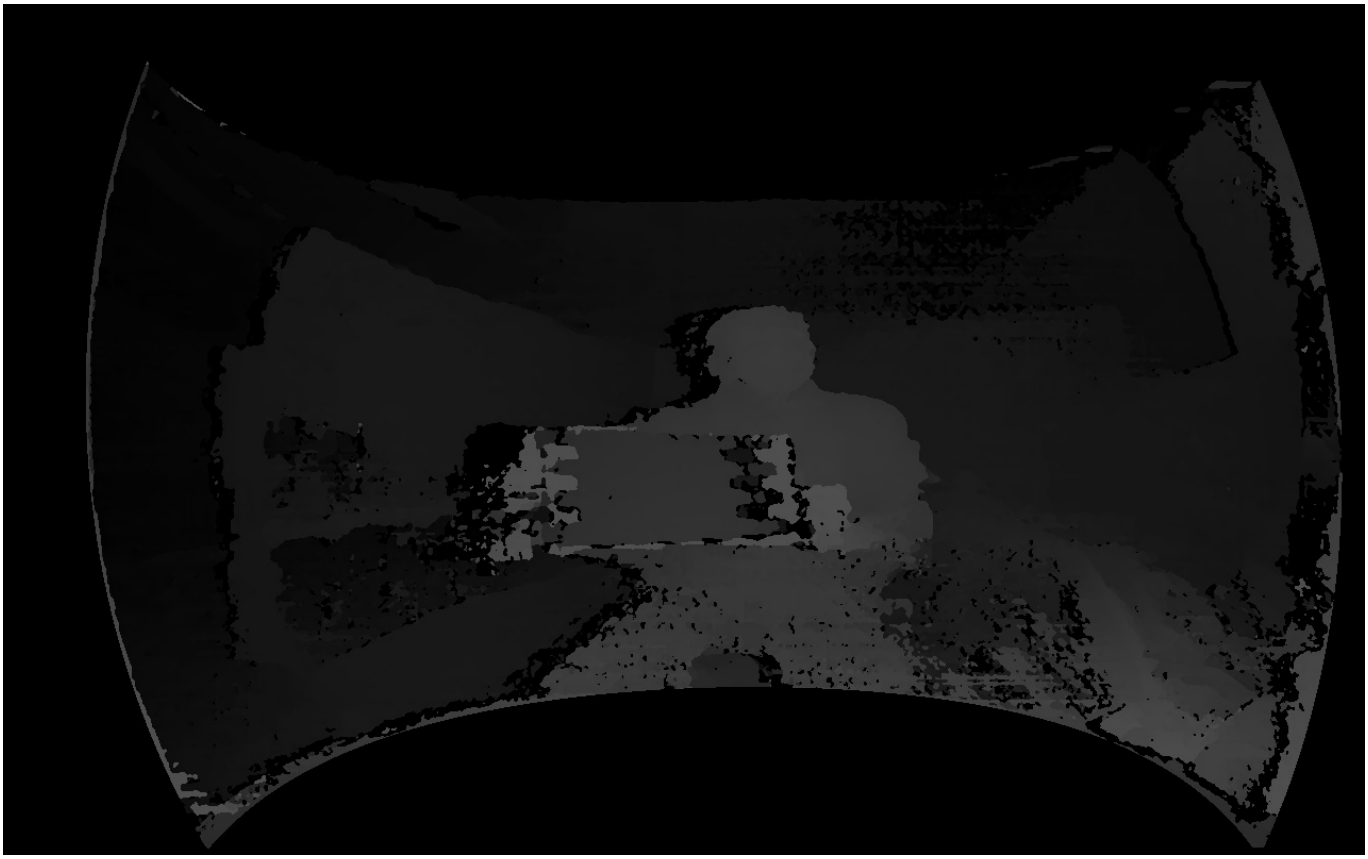
```
cv::Size imgSize = img1.size();
int numDisparities = 16*5;
int SADWindowSize = 5;
cv::Mat disMap;
int flag = cv::omnidir::RECTIFY_LONGLATI;
int pointType = omnidir::XYZRGB;
// the range of theta is (0, pi) and the range of phi is (0, pi)
cv::Matx33d KNew(imgSize.width / 3.1415, 0, 0, 0, imgSize.height / 3.1415,
0, 0, 0, 1);
Mat imageRec1, imageRec2, pointCloud;

cv::omnidir::stereoReconstruct(img1, img2, K1, D1, xi1, K2, D2, xi2, R, T,
flag, numDisparities, SADWindowSize, disMap, imageRec1, imageRec2, imgSize,
KNew, pointCloud);
```

Here variable `flag` indicates the recectify type, only `RECTIFY_LONGLATI`(recommend) and `RECTIFY_PERSPECTIVE` make sense. `numDisparities` is the max disparity value and `SADWindowSize` is the window size of `cv::StereoSGBM`. `pointType` is a flag to define the type of point cloud, `omnidir::XYZRGB` each point is a 6-dimensional vector, the first three elements are xyz coordinate, the last three elements are rgb color information. Another type `omnidir::XYZ` means each point is 3-dimensional and has only xyz coordiante.

Moreover, `imageRec1` and `imagerec2` are rectified versions of the first and second images. The epipolar lines of them have the same y-coordinate so that stereo matching becomes easy. Here are an example of them:



It can be observed that they are well aligned. The variable `disMap` is the disparity map computed by `cv::StereoSGBM` from `imageRec1` and `imageRec2`. The disparity map of the above two images is:



After we have disparity, we can compute 3D location for each pixel. The point cloud is stored in `pointCloud`, which is a 3-channel or 6-channel `cv::Mat`. We show the point cloud in the following image.