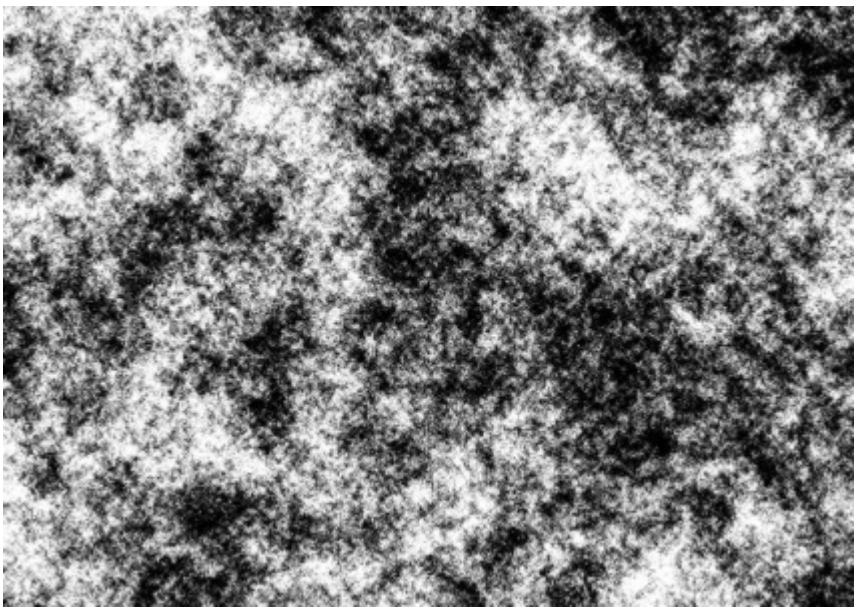
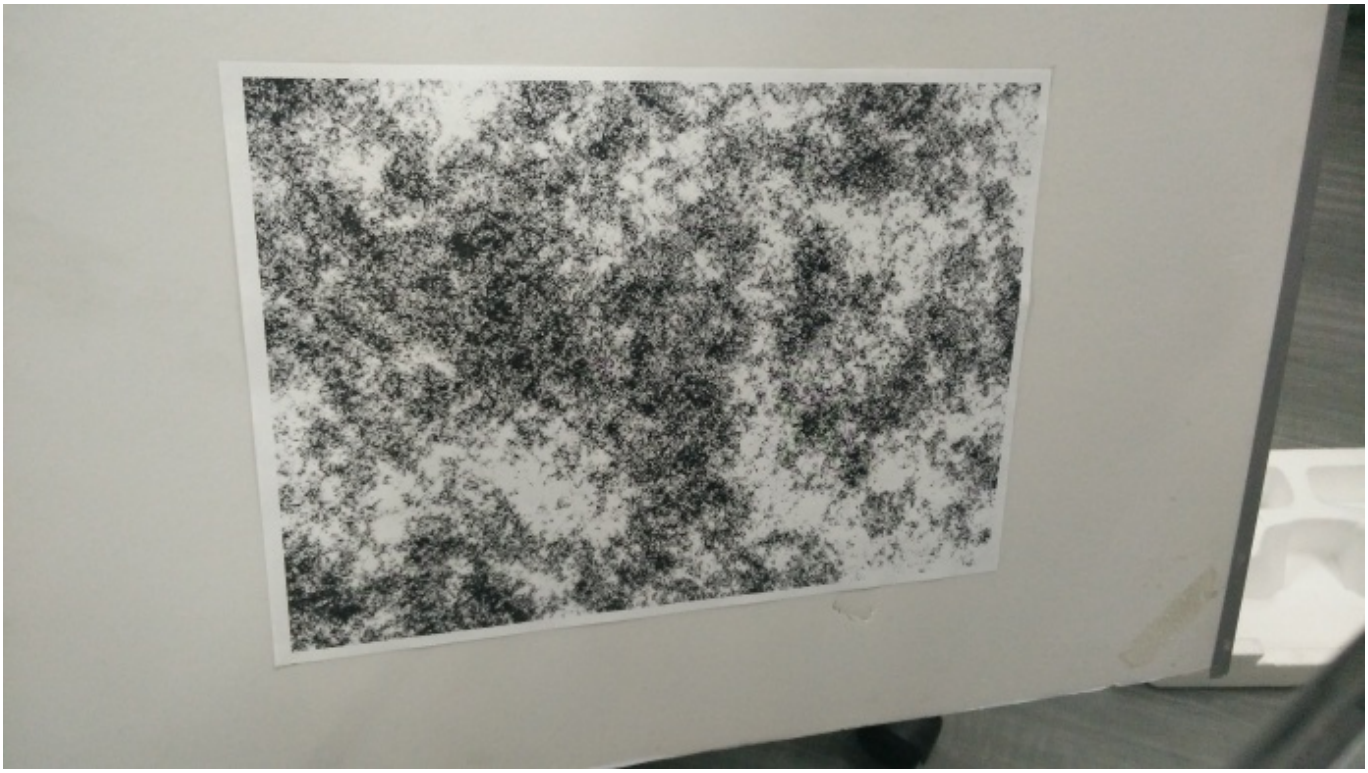
Multi-camera Calibration {#tutorial_multi_camera_main}

This tutorial will show how to use the multiple camera calibration toolbox. This toolbox is based on the usage of "random" pattern calibration object, so the tutorial is mainly two parts: an introduction to "random" pattern and multiple camera calibration.

Random Pattern Calibration Object

The random pattern is an image that is randomly generated. It is "random" so that it has many feature points. After generating it, one print it out and use it as a calibration object. The following two images are random pattern and a photo taken for it.





To generate a random pattern, use the class `cv::randpattern::RandomPatternGenerator` in `ccalib` module. Run it as

```
cv::randpattern::RandomPatternGenerator generator(width, height);
generator.generatePattern();
pattern = generator.getPattern();
```

Here `width` and `height` are width and height of pattern image. After getting the pattern, print it out and take some photos of it.

Now we can use these images to calibrate camera. First, `objectPoints` and `imagePoints` need to be detected. Use class `cv::randpattern::RandomPatternCornerFinder` to detect them. A sample code can be

```
cv::randpattern::RandomPatternCornerFinder finder(patternWidth,
patternHeight, nMiniMatches);
finder.loadPattern(pattern);
finder.computeObjectImagePoints(vecImg);
vector<Mat> objectPoints = finder.getObjectPoints();
vector<Mat> imagePoints = finder.getImagePoints();
```

Here variable `patternWidth` and `patternHeight` are physical pattern width and height with some user defined unit. `vecImg` is a vector of images that stores calibration images.

Second, use calibration functions like `cv::calibrateCamera` or `cv::omnidir::calibrate` to calibrate camera.

Multiple Cameras Calibration

Now we move to multiple camera calibration, so far this toolbox must use random pattern object.

To calibrate multiple cameras, we first need to take some photos of random pattern. Of cause, to calibrate the extrinsic parameters, one pattern need to be viewed by multiple cameras (at least two) at the same time. Another thing is that to help the program know which camera and which pattern the photo is taken, the image file should be named as "cameraIdx-timestamp.*". Photos with same timestamp means that they are the same object taken by several cameras. In addition, cameraIdx should start from 0. Some examples of files names are "0-129.png", "0-187.png", "1-187", "2-129".

Then, we can run multiple cameras calibration as

```
cv::multicalib::MultiCameraCalibration multiCalib(cameraType, nCamera,
inputFilename,patternWidth, patternHeight, showFeatureExtraction,
nMiniMatches);
multiCalib.run();
multiCalib.writeParameters(outputFilename);
```

Here `cameraType` indicates the camera type, `multicalib::MultiCameraCalibration::PINHOLE` and `multicalib::MultiCameraCalibration::OMNIDIRECTIONAL` are supported. For omnidirectional camera, you can refer to `cv::omnidir` module for detail. `nCamera` is the number of camers. `inputFilename` is the name of a file generated by `imagelist_creator` from `opencv/sample`. It stores names of random pattern and calibration images, the first file name is the name of random pattern. `patternWidth` and `patternHeight` are physical width and height of pattern. `showFeatureExtraction` is a flags to indicate whether show feature extraction process. `nMiniMatches` is a minimal points that should be detected in each frame, otherwise this frame will be abandoned. `outputFilename` is a xml file name to store parameters.