Praktikum 10
Camera Model & Calibration

1. Single Camera Calibration (On-line)

Program berikut ini mengkalibrasi sebuah kamera secara online dengan menggunakan chess-board (papan catur).

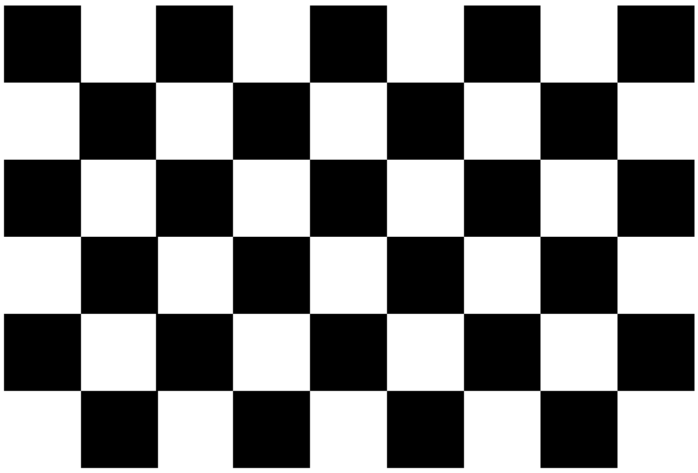
```
#include <cv.h>
#include <cxcore.h>
#include <highgui.h>

int n_boards = 0;
const int board_dt = 20;
int board_w;
int board_h;

int main()
{
        board_w = 5; // Board width in squares
        board_h = 8; // Board height
        n_boards = 8; // Number of boards
        int board_n = board_w * board_h;
        CvSize board_sz = cvSize( board_w, board_h );
        CvCapture* capture = cvCreateCameraCapture( 0 );
        assert( capture );

        cvNamedWindow( "Calibration" );
        // Allocate Sotrage
        CvMat* image_points           = cvCreateMat( n_boards*board_n, 2, CV_32FC1 );
        CvMat* object_points          = cvCreateMat( n_boards*board_n, 3, CV_32FC1 );
        CvMat* point_counts                   = cvCreateMat( n_boards, 1, CV_32SC1 );
        CvMat* intrinsic_matrix               = cvCreateMat( 3, 3, CV_32FC1 );
        CvMat* distortion_coeffs      = cvCreateMat( 5, 1, CV_32FC1 );

        CvPoint2D32f* corners = new CvPoint2D32f[ board_n ];
        int corner_count;
        int successes = 0;
        int step, frame = 0;

        IplImage *image = cvQueryFrame( capture );
        IplImage *gray_image = cvCreateImage( cvGetSize( image ), 8, 1 );

        // Capture Corner views loop until we've got n_boards
        // succesful captures (all corners on the board are found)

        while( successes < n_boards ){
                // Skp every board_dt frames to allow user to move chessboard
                if( frame++ % board_dt == 0 ){
                        // Find chessboard corners:
                        int found = cvFindChessboardCorners( image, board_sz, corners,
                                &corner_count, CV_CALIB_CB_ADAPTIVE_THRESH |
CV_CALIB_CB_FILTER_QUADS );

                        // Get subpixel accuracy on those corners
                        cvCvtColor( image, gray_image, CV_BGR2GRAY );
                        cvFindCornerSubPix( gray_image, corners, corner_count, cvSize(
11, 11 ),
                                cvSize( -1, -1 ), cvTermCriteria(
CV_TERMCRIT_EPS+CV_TERMCRIT_ITER, 30, 0.1 ));

                        // Draw it
                        cvDrawChessboardCorners( image, board_sz, corners, corner_count,
found );
```

```
                        cvShowImage( "Calibration", image );

                        // If we got a good board, add it to our data
                        if( corner_count == board_n ){
                                step = successes*board_n;
                                for( int i=step, j=0; j < board_n; ++i, ++j ){
                                        CV_MAT_ELEM( *image_points, float, i, 0 ) =
corners[j].x;
                                        CV_MAT_ELEM( *image_points, float, i, 1 ) =
corners[j].y;
                                        CV_MAT_ELEM( *object_points, float, i, 0 ) =
j/board_w;
                                        CV_MAT_ELEM( *object_points, float, i, 1 ) =
j%board_w;
                                        CV_MAT_ELEM( *object_points, float, i, 2 ) =
0.0f;
                                }
                                CV_MAT_ELEM( *point_counts, int, successes, 0 ) =
board_n;
                                successes++;
                        }
                }

                // Handle pause/unpause and ESC
                int c = cvWaitKey( 15 );
                if( c == 'p' ){
                        c = 0;
                        while( c != 'p' && c != 27 ){
                                c = cvWaitKey( 250 );
                        }
                }
                if( c == 27 )
                        return 0;
                image = cvQueryFrame( capture ); // Get next image
        } // End collection while loop

        // Allocate matrices according to how many chessboards found
        CvMat* object_points2 = cvCreateMat( successes*board_n, 3, CV_32FC1 );
        CvMat* image_points2 = cvCreateMat( successes*board_n, 2, CV_32FC1 );
        CvMat* point_counts2 = cvCreateMat( successes, 1, CV_32SC1 );

        // Transfer the points into the correct size matrices
        for( int i = 0; i < successes*board_n; ++i ){
                CV_MAT_ELEM( *image_points2, float, i, 0) = CV_MAT_ELEM( *image_points,
float, i, 0 );
                CV_MAT_ELEM( *image_points2, float, i, 1) = CV_MAT_ELEM( *image_points,
float, i, 1 );
                CV_MAT_ELEM( *object_points2, float, i, 0) = CV_MAT_ELEM(
*object_points, float, i, 0 );
                CV_MAT_ELEM( *object_points2, float, i, 1) = CV_MAT_ELEM(
*object_points, float, i, 1 );
                CV_MAT_ELEM( *object_points2, float, i, 2) = CV_MAT_ELEM(
*object_points, float, i, 2 );
        }

        for( int i=0; i < successes; ++i ){
                CV_MAT_ELEM( *point_counts2, int, i, 0 ) = CV_MAT_ELEM( *point_counts,
int, i, 0 );
        }
        cvReleaseMat( &object_points );
        cvReleaseMat( &image_points );
        cvReleaseMat( &point_counts );

        // At this point we have all the chessboard corners we need
        // Initiliazie the intrinsic matrix such that the two focal lengths
        // have a ratio of 1.0

        CV_MAT_ELEM( *intrinsic_matrix, float, 0, 0 ) = 1.0;
        CV_MAT_ELEM( *intrinsic_matrix, float, 1, 1 ) = 1.0;

        // Calibrate the camera
        cvCalibrateCamera2( object_points2, image_points2, point_counts2, cvGetSize(
image ),
                intrinsic_matrix, distortion_coeffs, NULL, NULL,
CV_CALIB_FIX_ASPECT_RATIO );
```

```
        // Save the intrinsics and distortions
        cvSave( "Intrinsics.xml", intrinsic_matrix );
        cvSave( "Distortion.xml", distortion_coeffs );

        // Example of loading these matrices back in
        CvMat *intrinsic = (CvMat*)cvLoad( "Intrinsics.xml" );
        CvMat *distortion = (CvMat*)cvLoad( "Distortion.xml" );

        // Build the undistort map that we will use for all subsequent frames
        IplImage* mapx = cvCreateImage( cvGetSize( image ), IPL_DEPTH_32F, 1 );
        IplImage* mapy = cvCreateImage( cvGetSize( image ), IPL_DEPTH_32F, 1 );
        cvInitUndistortMap( intrinsic, distortion, mapx, mapy );

        // Run the camera to the screen, now showing the raw and undistorted image
        cvNamedWindow( "Undistort" );

        while( image ){
                IplImage *t = cvCloneImage( image );
                cvShowImage( "Calibration", image ); // Show raw image
                cvRemap( t, image, mapx, mapy ); // undistort image
                cvReleaseImage( &t );
                cvShowImage( "Undistort", image ); // Show corrected image

                // Handle pause/unpause and esc
                int c = cvWaitKey( 15 );
                if( c == 'p' ){
                        c = 0;
                        while( c != 'p' && c != 27 ){
                                c = cvWaitKey( 250 );
                        }
                }
                if( c == 27 )
                        break;
                image = cvQueryFrame( capture );
        }

        return 0;
}
```

Petunjuk praktikum:

- Siapkan *chessboard* (papan catur) seperti pada gambar dibawah ini



- Kemudian arahkan kamera ke *chessboard*, untuk diambil gambarnya beberapa kali. Tunggu sampai muncul window baru yang merupakan hasil kalibrasi.
- Amati dan bandingkan hasil kalibrasi dengan gambar sebelumnya, sebuatkan perbedaannya.

2. Stereo Calibration (Off-line)

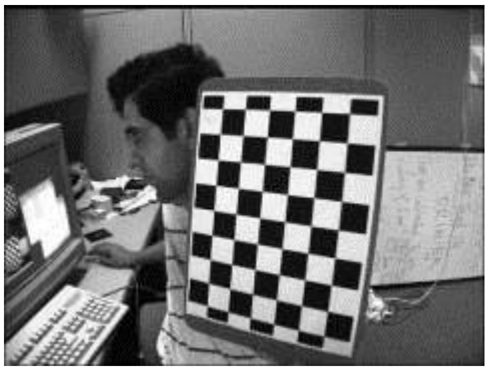
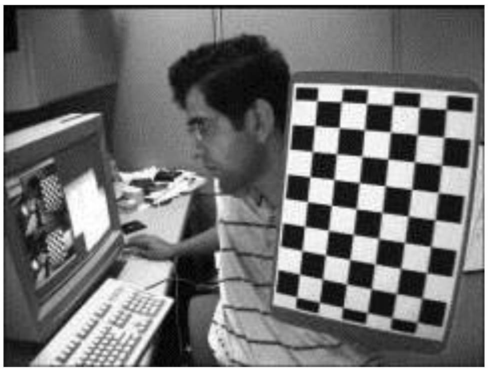
Program berikut ini mengkalibrasi dua buah kamera secara offline dengan menggunakan chess-board (papan catur).

```cpp
#include "cv.h"
#include "cxmisc.h"
#include "highgui.h"
#include <vector>
#include <string>
#include <algorithm>
#include <stdio.h>
#include <ctype.h>

using namespace std;

//
// Given a list of chessboard images, the number of corners (nx, ny)
// on the chessboards, and a flag: useCalibrated for calibrated (0) or
// uncalibrated (1: use cvStereoCalibrate(), 2: compute fundamental
// matrix separately) stereo. Calibrate the cameras and display the
// rectified results along with the computed disparity images.
//
static void
StereoCalib(const char* imageList, int nx, int ny, int useUncalibrated)
{
    int displayCorners = 0;
    int showUndistorted = 1;
    bool isVerticalStereo = false;//OpenCV can handle left-right
                                  //or up-down camera arrangements
    const int maxScale = 1;
    const float squareSize = 1.f; //Set this to your actual square size
    FILE* f = fopen(imageList, "rt");
    int i, j, lr, nframes, n = nx*ny, N = 0;
    vector<string> imageNames[2];
    vector<CvPoint3D32f> objectPoints;
    vector<CvPoint2D32f> points[2];
    vector<int> npoints;
    vector<uchar> active[2];
    vector<CvPoint2D32f> temp(n);
    CvSize imageSize = {0,0};

    // ARRAY AND VECTOR STORAGE:

    double M1[3][3], M2[3][3], D1[5], D2[5];
    double R[3][3], T[3], E[3][3], F[3][3];
    CvMat _M1 = cvMat(3, 3, CV_64F, M1 );
    CvMat _M2 = cvMat(3, 3, CV_64F, M2 );
    CvMat _D1 = cvMat(1, 5, CV_64F, D1 );
    CvMat _D2 = cvMat(1, 5, CV_64F, D2 );
    CvMat _R = cvMat(3, 3, CV_64F, R );
    CvMat _T = cvMat(3, 1, CV_64F, T );
    CvMat _E = cvMat(3, 3, CV_64F, E );
    CvMat _F = cvMat(3, 3, CV_64F, F );
    if( displayCorners )
        cvNamedWindow( "corners", 1 );

// READ IN THE LIST OF CHESSBOARDS:
    if( !f )
    {
        fprintf(stderr, "can not open file %s\n", imageList );
        return;
    }
    for(i=0;;i++)
    {
        char buf[1024];
        int count = 0, result=0;
        lr = i % 2;
        vector<CvPoint2D32f>& pts = points[lr];
        if( !fgets( buf, sizeof(buf)-3, f ))
            break;
        size_t len = strlen(buf);
        while( len > 0 && isspace(buf[len-1]))
            buf[--len] = '\0';
        if( buf[0] == '#')
            continue;
        IplImage* img = cvLoadImage( buf, 0 );
        if( !img )
            break;
        imageSize = cvGetSize(img);
        imageNames[lr].push_back(buf);

    //FIND CHESSBOARDS AND CORNERS THEREIN:
        for( int s = 1; s <= maxScale; s++ )
        {
            IplImage* timg = img;
            if( s > 1 )
            {
                timg = cvCreateImage(cvSize(img->width*s,img->height*s),
                    img->depth, img->nChannels );
                cvResize( img, timg, CV_INTER_CUBIC );
            }
            result = cvFindChessboardCorners( timg, cvSize(nx, ny),
                &temp[0], &count,
                CV_CALIB_CB_ADAPTIVE_THRESH |
                CV_CALIB_CB_NORMALIZE_IMAGE);
            if( timg != img )
                cvReleaseImage( &timg );
            if( result || s == maxScale )
                for( j = 0; j < count; j++ )
            {
                temp[j].x /= s;
                temp[j].y /= s;
            }
            if( result )
                break;
        }
        if( displayCorners )
        {
            printf("%s\n", buf);
            IplImage* cimg = cvCreateImage( imageSize, 8, 3 );
            cvCvtColor( img, cimg, CV_GRAY2BGR );
            cvDrawChessboardCorners( cimg, cvSize(nx, ny), &temp[0],
                count, result );
            cvShowImage( "corners", cimg );
            cvReleaseImage( &cimg );
            int c = cvWaitKey(1000);
            if( c == 27 || c == 'q' || c == 'Q' ) //Allow ESC to quit
                exit(-1);
        }
        else
            putchar('.');
        N = pts.size();
        pts.resize(N + n, cvPoint2D32f(0,0));
        active[lr].push_back((uchar)result);
    //assert( result != 0 );
        if( result )
        {
         //Calibration will suffer without subpixel interpolation
            cvFindCornerSubPix( img, &temp[0], count,
                cvSize(11, 11), cvSize(-1,-1),
                cvTermCriteria(CV_TERMCRIT_ITER+CV_TERMCRIT_EPS,
                30, 0.01) );
            copy( temp.begin(), temp.end(), pts.begin() + N );
        }
        cvReleaseImage( &img );
    }
    fclose(f);
    printf("\n");

// HARVEST CHESSBOARD 3D OBJECT POINT LIST:
    nframes = active[0].size();//Number of good chessboads found
    objectPoints.resize(nframes*n);
    for( i = 0; i < ny; i++ )
        for( j = 0; j < nx; j++ )
        objectPoints[i*nx + j] =
        cvPoint3D32f(i*squareSize, j*squareSize, 0);
    for( i = 1; i < nframes; i++ )
```

```
        copy( objectPoints.begin(), objectPoints.begin() + n,
        objectPoints.begin() + i*n );
    npoints.resize(nframes,n);
    N = nframes*n;
    CvMat _objectPoints = cvMat(1, N, CV_32FC3, &objectPoints[0] );
    CvMat _imagePoints1 = cvMat(1, N, CV_32FC2, &points[0][0] );
    CvMat _imagePoints2 = cvMat(1, N, CV_32FC2, &points[1][0] );
    CvMat _npoints = cvMat(1, npoints.size(), CV_32S, &npoints[0] );
    cvSetIdentity(&_M1);
    cvSetIdentity(&_M2);
    cvZero(&_D1);
    cvZero(&_D2);

// CALIBRATE THE STEREO CAMERAS
    printf("Running stereo calibration ...");
    fflush(stdout);
    cvStereoCalibrate( &_objectPoints, &_imagePoints1,
        &_imagePoints2, &_npoints,
        &_M1, &_D1, &_M2, &_D2,
        imageSize, &_R, &_T, &_E, &_F,
        cvTermCriteria(CV_TERMCRIT_ITER+
        CV_TERMCRIT_EPS, 100, 1e-5),
        CV_CALIB_FIX_ASPECT_RATIO +
        CV_CALIB_ZERO_TANGENT_DIST +
        CV_CALIB_SAME_FOCAL_LENGTH );
    printf(" done\n");

// CALIBRATION QUALITY CHECK
// because the output fundamental matrix implicitly
// includes all the output information,
// we can check the quality of calibration using the
// epipolar geometry constraint: m2^t*F*m1=0
    vector<CvPoint3D32f> lines[2];
    points[0].resize(N);
    points[1].resize(N);
    _imagePoints1 = cvMat(1, N, CV_32FC2, &points[0][0] );
    _imagePoints2 = cvMat(1, N, CV_32FC2, &points[1][0] );
    lines[0].resize(N);
    lines[1].resize(N);
    CvMat _L1 = cvMat(1, N, CV_32FC3, &lines[0][0]);
    CvMat _L2 = cvMat(1, N, CV_32FC3, &lines[1][0]);

//Always work in undistorted space
    cvUndistortPoints( &_imagePoints1, &_imagePoints1,
        &_M1, &_D1, 0, &_M1 );
    cvUndistortPoints( &_imagePoints2, &_imagePoints2,
        &_M2, &_D2, 0, &_M2 );
    cvComputeCorrespondEpilines( &_imagePoints1, 1, &_F, &_L1 );
    cvComputeCorrespondEpilines( &_imagePoints2, 2, &_F, &_L2 );
    double avgErr = 0;
    for( i = 0; i < N; i++ )
    {
        double err = fabs(points[0][i].x*lines[1][i].x +
            points[0][i].y*lines[1][i].y + lines[1][i].z)
            + fabs(points[1][i].x*lines[0][i].x +
            points[1][i].y*lines[0][i].y + lines[0][i].z);
        avgErr += err;
    }
    printf( "avg err = %g\n", avgErr/(nframes*n) );

//COMPUTE AND DISPLAY RECTIFICATION
    if( showUndistorted )
    {
        CvMat* mx1 = cvCreateMat( imageSize.height,
            imageSize.width, CV_32F );
        CvMat* my1 = cvCreateMat( imageSize.height,
            imageSize.width, CV_32F );
        CvMat* mx2 = cvCreateMat( imageSize.height,

            imageSize.width, CV_32F );
        CvMat* my2 = cvCreateMat( imageSize.height,
            imageSize.width, CV_32F );
        CvMat* img1r = cvCreateMat( imageSize.height,
            imageSize.width, CV_8U );
        CvMat* img2r = cvCreateMat( imageSize.height,
            imageSize.width, CV_8U );
        CvMat* disp = cvCreateMat( imageSize.height,
            imageSize.width, CV_16S );
        CvMat* vdisp = cvCreateMat( imageSize.height,
            imageSize.width, CV_8U );
        CvMat* pair;
        double R1[3][3], R2[3][3], P1[3][4], P2[3][4];
        CvMat _R1 = cvMat(3, 3, CV_64F, R1);
        CvMat _R2 = cvMat(3, 3, CV_64F, R2);

// IF BY CALIBRATED (BOUGUET'S METHOD)
        if( useUncalibrated == 0 )
        {
            CvMat _P1 = cvMat(3, 4, CV_64F, P1);
            CvMat _P2 = cvMat(3, 4, CV_64F, P2);
            cvStereoRectify( &_M1, &_M2, &_D1, &_D2, imageSize,
                &_R, &_T,
                &_R1, &_R2, &_P1, &_P2, 0,
                0/*CV_CALIB_ZERO_DISPARITY*/ );
            isVerticalStereo = fabs(P2[1][3]) > fabs(P2[0][3]);
    //Precompute maps for cvRemap()
            cvInitUndistortRectifyMap(&_M1,&_D1,&_R1,&_P1,mx1,my1);
            cvInitUndistortRectifyMap(&_M2,&_D2,&_R2,&_P2,mx2,my2);
        }

//OR ELSE HARTLEY'S METHOD
        else if( useUncalibrated == 1 || useUncalibrated == 2 )
     // use intrinsic parameters of each camera, but
     // compute the rectification transformation directly
     // from the fundamental matrix
        {
            double H1[3][3], H2[3][3], iM[3][3];
            CvMat _H1 = cvMat(3, 3, CV_64F, H1);
            CvMat _H2 = cvMat(3, 3, CV_64F, H2);
            CvMat _iM = cvMat(3, 3, CV_64F, iM);
    //Just to show you could have independently used F
            if( useUncalibrated == 2 )
                cvFindFundamentalMat( &_imagePoints1,
                &_imagePoints2, &_F);
            cvStereoRectifyUncalibrated( &_imagePoints1,
                &_imagePoints2, &_F,
                imageSize,
                &_H1, &_H2, 3);
            cvInvert(&_M1, &_iM);
            cvMatMul(&_H1, &_M1, &_R1);
            cvMatMul(&_iM, &_R1, &_R1);
            cvInvert(&_M2, &_iM);
            cvMatMul(&_H2, &_M2, &_R2);
            cvMatMul(&_iM, &_R2, &_R2);
    //Precompute map for cvRemap()
            cvInitUndistortRectifyMap(&_M1,&_D1,&_R1,&_M1,mx1,my1);

            cvInitUndistortRectifyMap(&_M2,&_D1,&_R2,&_M2,mx2,my2);
        }
        else
            assert(0);
        cvNamedWindow( "rectified", 1 );

// RECTIFY THE IMAGES AND FIND DISPARITY MAPS
        if( !isVerticalStereo )
            pair = cvCreateMat( imageSize.height, imageSize.width*2,
            CV_8UC3 );
        else
            pair = cvCreateMat( imageSize.height*2, imageSize.width,
            CV_8UC3 );

//Setup for finding stereo corrrespondences
        CvStereoBMState *BMState = cvCreateStereoBMState();
        assert(BMState != 0);
        BMState->preFilterSize=41;
        BMState->preFilterCap=31;
        BMState->SADWindowSize=41;
        BMState->minDisparity=-64;
        BMState->numberOfDisparities=128;
        BMState->textureThreshold=10;
        BMState->uniquenessRatio=15;
        for( i = 0; i < nframes; i++ )
        {
            IplImage* img1=cvLoadImage(imageNames[0][i].c_str(),0);
            IplImage* img2=cvLoadImage(imageNames[1][i].c_str(),0);
            if( img1 && img2 )
            {
                CvMat part;
                cvRemap( img1, img1r, mx1, my1 );
                cvRemap( img2, img2r, mx2, my2 );
                if( !isVerticalStereo || useUncalibrated != 0 )
                {
              // When the stereo camera is oriented vertically,
              // useUncalibrated==0 does not transpose the
              // image, so the epipolar lines in the rectified
              // images are vertical. Stereo correspondence
              // function does not support such a case.
                    cvFindStereoCorrespondenceBM( img1r, img2r, disp,
                        BMState);
                    cvNormalize( disp, vdisp, 0, 256, CV_MINMAX );
                    cvNamedWindow( "disparity" );
                    cvShowImage( "disparity", vdisp );
                }
                if( !isVerticalStereo )
                {
                    cvGetCols( pair, &part, 0, imageSize.width );
                    cvCvtColor( img1r, &part, CV_GRAY2BGR );
                    cvGetCols( pair, &part, imageSize.width,
                        imageSize.width*2 );
                    cvCvtColor( img2r, &part, CV_GRAY2BGR );
                    for( j = 0; j < imageSize.height; j += 16 )
                        cvLine( pair, cvPoint(0,j),
                        cvPoint(imageSize.width*2,j),
                        CV_RGB(0,255,0));
                }
                else
                {
                    cvGetRows( pair, &part, 0, imageSize.height );
                    cvCvtColor( img1r, &part, CV_GRAY2BGR );
                    cvGetRows( pair, &part, imageSize.height,
                        imageSize.height*2 );
                    cvCvtColor( img2r, &part, CV_GRAY2BGR );
                    for( j = 0; j < imageSize.width; j += 16 )
                        cvLine( pair, cvPoint(j,0),
                        cvPoint(j,imageSize.height*2),
                        CV_RGB(0,255,0));
                }
                cvShowImage( "rectified", pair );
                if( cvWaitKey() == 27 )
                    break;
            }
            cvReleaseImage( &img1 );
            cvReleaseImage( &img2 );
        }
        cvReleaseStereoBMState(&BMState);
        cvReleaseMat( &mx1 );
        cvReleaseMat( &my1 );
        cvReleaseMat( &mx2 );
        cvReleaseMat( &my2 );
        cvReleaseMat( &img1r );
        cvReleaseMat( &img2r );
        cvReleaseMat( &disp );
    }
}

int main(void)
{
    StereoCalib("stereo_calib.txt", 9, 6, 1);
    return 0;
}
```

Petunjuk praktikum:

- Siapkan *chessboard* (papan catur) seperti pada percobaan diatas.
- Ambil beberapa gambar menggunakan dua buah kamera (stereo). Kamera kanan dan kamera kiri masing-masing sebanyak 14 kali. Seperti contoh pada gambar di bawah ini.

- Kemudian jalankan program, tunggu sampai muncul window baru yang merupakan hasil kalibrasi yang dapat menghasilkan kedalaman.
- Bandingkan tingkat kedalaman hasil kalibrasi dengan kedalaman sebenarnya.