```cpp
// EEE 598 : DIVP Project 1: Stereo Correspondence using FAST+FREAK, AGAST+LUCID and Sift
// Referernce :https://docs.opencv.org
// Authors : Chandana Srinivasa and Kiran Kumar (Group 1)

#include<stdio.h>
#include<iostream>
#include <vector>
#include"opencv2/core.hpp"
#include"opencv2/features2d.hpp"
#include"opencv2/xfeatures2d.hpp"
#include"opencv2/highgui.hpp"
#include"opencv2/imgproc.hpp"
#include "opencv2/calib3d.hpp"

using namespace cv;
using namespace std;

using namespace cv::xfeatures2d;
/** @function main */


// Function to find out the error value
float errFinder(Mat img_1_Grey, Mat GT_img, vector<KeyPoint> keypoints_1_fast, vector<KeyPoint> keypoints_2_fast, vector< DMatch > matches_Fast_Freak)
{
    // img1: right  img2: left (Reference)

    std::vector<int> pointIndexesLeft;
    std::vector<int> pointIndexesRight;
    for (std::vector<cv::DMatch>::const_iterator it = matches_Fast_Freak.begin(); it != matches_Fast_Freak.end(); ++it)
    {

        // Get the indexes of the selected matched keypoints
        pointIndexesRight.push_back(it->queryIdx);
        pointIndexesLeft.push_back(it->trainIdx);
    }

    // Convert keypoints into Point2f
    std::vector<cv::Point2f> selPointsLeft, selPointsRight;
    cv::KeyPoint::convert(keypoints_1_fast, selPointsRight, pointIndexesRight);
    cv::KeyPoint::convert(keypoints_2_fast, selPointsLeft, pointIndexesLeft);
    int xright = 0, xleft = 0, yright = 0, yleft = 0;

    Mat diff(img_1_Grey.rows, img_1_Grey.cols, CV_8UC1);
    diff = 255; int k = 0; int new_xright = 0; float MSE = 0.0; float add_all = 0.0; float count = 0; float err = 0;
    cout<<"Match count"<<selPointsRight.size() << endl;
    for (int i = 0; i < (int)selPointsRight.size(); i++)
    {
        xright = (int)selPointsRight.at(i).x;
        yright = (int)selPointsRight.at(i).y;

        xleft = (int)selPointsLeft.at(i).x;
        yleft = (int)selPointsLeft.at(i).y;



        k = GT_img.at<uchar>(xleft, yleft);// Disparity from Ground Truth

        new_xright = xleft + k;
        count += 1;
        //root mean square error calculation
        MSE = pow((new_xright - xright), 2) + pow((yleft - yright), 2);
        add_all = add_all + MSE;
        //diff.at<uchar>(yright,xright) = abs(xright - xleft);

    }
    err = sqrt(add_all) / count;
    waitKey();
    return err;
}

int   main(int argc, char** argv)
{
    try
    {
        // ... Contents of your main
    }
    catch (cv::Exception & e)
    {
        cerr << e.what() << endl; // output exception message
    }
    cout << " Output For Without Cross Check and Without Threshold " << endl;


    Mat   img_1 = imread("D:\\ASU_Sem_1\\DIVP_Project\\new_data_set\
      \Flowerpots\\view1.png",CV_BGR2GRAY);// reading input image 1
    Mat   img_2 = imread("D:\\ASU_Sem_1\\DIVP_Project\\new_data_set\
      \Flowerpots\\view5.png",CV_BGR2GRAY);// reading input image 2
    Mat GT_img = imread("D:\\ASU_Sem_1\\DIVP_Project\\new_data_set\\Flowerpots
      \\disp5.png");//reading ground truth disparity values
    //cout << "GT_img"<< GT_img << endl;

    Mat img_1_D, img_2_D, img_1_Grey, img_2_Grey;
    pyrDown(img_1,img_1_D);
    pyrDown(img_2,img_2_D);

    cvtColor(img_1_D, img_1_Grey, CV_BGR2GRAY);
    cvtColor(img_2_D, img_2_Grey, CV_BGR2GRAY);

    // --  Detect the keypoints using a FAST Detector
    Ptr <FastFeatureDetector> detector_fast = FastFeatureDetector::create();
    std::vector<KeyPoint> keypoints_1_fast, keypoints_2_fast;
    detector_fast->detect(img_1_Grey, keypoints_1_fast);
    detector_fast->detect(img_2_Grey, keypoints_2_fast);

    cout << " [INFO] key-point 1 size: Fast " << keypoints_1_fast.size() <<
      endl;

```

```cpp
    cout << " [INFO] key-point 2 size: Fast" << keypoints_2_fast.size() << endl;

    // --  Draw keypoints
    Mat   img_keypoints_1_fast;
    Mat   img_keypoints_2_fast;
    drawKeypoints(img_1_Grey, keypoints_1_fast, img_keypoints_1_fast, Scalar::all(-1), DrawMatchesFlags::DEFAULT);
    drawKeypoints(img_2_Grey, keypoints_2_fast, img_keypoints_2_fast, Scalar::all(-1), DrawMatchesFlags::DEFAULT);

    // --  Show detected (drawn) keypoints
    //imshow("Keypoints 1", img_keypoints_1_fast);
    imwrite("D:\\ASU_Sem_1\\DIVP_Project\\Output_images\\img_keypoints_1_fast.png", img_keypoints_1_fast);

    //imshow("Keypoints 2", img_keypoints_2_fast);
    imwrite("D:\\ASU_Sem_1\\DIVP_Project\\Output_images\\img_keypoints_2_fast.png", img_keypoints_2_fast);

    Mat desc1_fast, desc2_fast;

    //FREAK Descriptor
    Ptr<FREAK> freak = FREAK::create(true,true,22.0f,4);
    freak->compute(img_1_Grey, keypoints_1_fast, desc1_fast);
    freak->compute(img_2_Grey, keypoints_2_fast, desc2_fast);


    // finding the matched points using BFMatcher
    BFMatcher matcher(NORM_L2);
    //BFMatcher matcher(NORM_L2,true);  //For crosscheck
    std::vector< DMatch > matches_Fast_Freak;
    std::vector< DMatch > matches_fast_freak;
    matcher.match(desc1_fast,desc2_fast, matches_fast_freak);
    Mat img_matches_fast;
    // -------- For Thresholding ------

    //DMatch f; float mini_fast = 200;
    //for (int k = 0; k < matches_fast_freak.size(); k++)
    //{
        //f = matches_fast_freak[k];
        //if (f.distance < 3 * mini_fast)
            //matches_Fast_Freak.push_back(matches_fast_freak[k]);
    //}
    matches_Fast_Freak = matches_fast_freak;  // Remove this line for thresholding

    //draw the best matches
    drawMatches(img_1_D, keypoints_1_fast, img_2_D, keypoints_2_fast, matches_Fast_Freak, img_matches_fast);

    //-- Show detected matches
    //imshow("FAST+FREAK", img_matches_fast);
    imwrite("D:\\ASU_Sem_1\\DIVP_Project\\Output_images\\img_matches_fast_wo_cross_wo_thresh.png", img_matches_fast);


    // Sift keypoints detection
    Ptr  <SIFT> detector = SIFT::create();
    std::vector<KeyPoint> keypoints_1_Sift, keypoints_2_Sift;


    detector->detect(img_1_Grey, keypoints_1_Sift);
    detector->detect(img_2_Grey, keypoints_2_Sift);
    // --  Draw keypoints
    Mat   img_keypoints_1_Sift;
    Mat   img_keypoints_2_Sift;

    cout << " [INFO] key-point 1 size: Sift " << keypoints_1_Sift.size() << endl;
    //KeyPointsFilter::retainBest(keypoints_1_fast, 500);

    cout << " [INFO] key-point 2 size: Sift" << keypoints_2_Sift.size() << endl;


    Mat desc1_Sift, desc2_Sift;
    drawKeypoints(img_1_D, keypoints_1_Sift, img_keypoints_1_Sift, Scalar::all(-1), DrawMatchesFlags::DEFAULT);
    drawKeypoints(img_2_D, keypoints_2_Sift, img_keypoints_2_Sift, Scalar::all(-1), DrawMatchesFlags::DEFAULT);
    // --  Show detected (drawn) keypoints
    //imshow("Keypoints 1 SIFT", img_keypoints_1_Sift);
    imwrite("D:\\ASU_Sem_1\\DIVP_Project\\Output_images\\img_keypoints_1_Sift.png", img_keypoints_1_Sift);

    //imshow("Keypoints 2 SIFT", img_keypoints_2_Sift);
    imwrite("D:\\ASU_Sem_1\\DIVP_Project\\Output_images\\img_keypoints_2_Sift.png", img_keypoints_2_Sift);

    //Sift descriptor
    detector->compute(img_1_D, keypoints_1_Sift,desc1_Sift);
    detector->compute(img_2_D, keypoints_2_Sift,desc2_Sift);


    // Matches
    std::vector< DMatch > matches_sift;
    std::vector< DMatch > matches_Sift;
    //DMatch A; float mini = 30;
    matcher.match(desc1_Sift, desc2_Sift, matches_sift);
    // -------- For Thresholding ------

    //for (int k = 0; k < matches_sift.size(); k++)
    //{
        //A = matches_sift[k];
        //if (A.distance < 3 * mini)
        //matches_Sift.push_back(matches_sift[k]);
    //}
    matches_Sift = matches_sift; // Remove this line for thresholding

    Mat img_matches_Sift;
    drawMatches(img_1_D, keypoints_1_Sift, img_2_D, keypoints_2_Sift,
        matches_Sift, img_matches_Sift);

    //-- Show detected matches
    //imshow("SIFT", img_matches_Sift);
    imwrite("D:\\ASU_Sem_1\\DIVP_Project\\Output_images\\img_matches_Sift_wo_cross_wo_thresh.png", img_matches_Sift);

    //AGAST Keypoints
    Ptr  <AgastFeatureDetector> detector_agast = AgastFeatureDetector::create();
    std::vector<KeyPoint> keypoints_1_Agast, keypoints_2_Agast;
    detector_agast->detect(img_1_Grey, keypoints_1_Agast);
    detector_agast->detect(img_2_Grey, keypoints_2_Agast);

    cout << " [INFO] key-point 1 size: Agast " << keypoints_1_Agast.size() << endl;
    //KeyPointsFilter::retainBest(keypoints_1_fast, 500);

    cout << " [INFO] key-point 2 size: Agast" << keypoints_2_Agast.size() << endl;


    // --  Draw keypoints
    Mat   img_keypoints_1_Agast;
    Mat   img_keypoints_2_Agast;
    Mat desc1_Lucid, desc2_Lucid;
    drawKeypoints(img_1_D, keypoints_1_Agast, img_keypoints_1_Agast, Scalar::all(-1), DrawMatchesFlags::DEFAULT);
    drawKeypoints(img_2_D, keypoints_2_Agast, img_keypoints_2_Agast, Scalar::all(-1), DrawMatchesFlags::DEFAULT);

    // --  Show detected (drawn) keypoints
    //imshow("Keypoints 1 AGAST", img_keypoints_1_Agast);
    imwrite("D:\\ASU_Sem_1\\DIVP_Project\\Output_images\\img_keypoints_1_Agast.png", img_keypoints_1_Agast);
    //imshow("Keypoints 2 AGAST", img_keypoints_2_Agast);
    imwrite("D:\\ASU_Sem_1\\DIVP_Project\\Output_images\\img_keypoints_2_Agast.png", img_keypoints_2_Agast);

    //Detectors using LUCID
    Ptr<LUCID> lucid = LUCID::create();
    lucid->compute(img_1_D, keypoints_1_Agast, desc1_Lucid);
    lucid->compute(img_2_D, keypoints_2_Agast, desc2_Lucid);

    // Matches
    std::vector< DMatch > matches_Agast_Lucid;
    std::vector< DMatch > matches_agast_lucid;
    DMatch a; float mini_lucid = 10;
    matcher.match(desc1_Lucid, desc2_Lucid, matches_agast_lucid);

    // -------- For Thresholding ------
    //for (int k = 0; k < matches_sift.size(); k++)
    //{
        //a = matches_agast_lucid[k];
        //if (a.distance < 3 * mini_lucid)
        //matches_Agast_Lucid.push_back(matches_agast_lucid[k]);
    //}
    matches_Agast_Lucid = matches_agast_lucid; // Remove this line for thresholding


    Mat img_matches_Agast_Lucid;
    drawMatches(img_1_Grey, keypoints_1_Agast, img_2_Grey, keypoints_2_Agast, matches_Agast_Lucid, img_matches_Agast_Lucid);
    //-- Show detected matches
    //imshow("AGAST+LUCID", img_matches_Agast_Lucid);
    imwrite("D:\\ASU_Sem_1\\DIVP_Project\\Output_images\\img_matches_Agast_Lucid_wo_cross_wo_thresh.png", img_matches_Agast_Lucid);


    // Finding the error : Calling the errFinder function

    float err_fast_freak, err_Sift, err_Agast_Lucid;
    err_fast_freak = errFinder(img_1_Grey, GT_img, keypoints_1_fast, keypoints_2_fast, matches_Fast_Freak);
    cout << "err value fast freak" << err_fast_freak << endl;
    err_Sift = errFinder(img_1_Grey, GT_img, keypoints_1_Sift, keypoints_2_Sift, matches_Sift);
    cout << "err value Sift" << err_Sift << endl;
    err_Agast_Lucid = errFinder(img_1_Grey, GT_img, keypoints_1_Agast, keypoints_2_Agast, matches_Agast_Lucid);
    cout << "err value Agast Lucid" << err_Agast_Lucid << endl;

    waitKey(0);
    waitKey();
    return 0;
}



    //Mat Disparity(994, 1440, CV_8U, diff);
    //imshow("Disparity", diff);


```