**Human pose Estimation project work**

**Tutorial 1.2 Assignment**

Mukul C Yadav

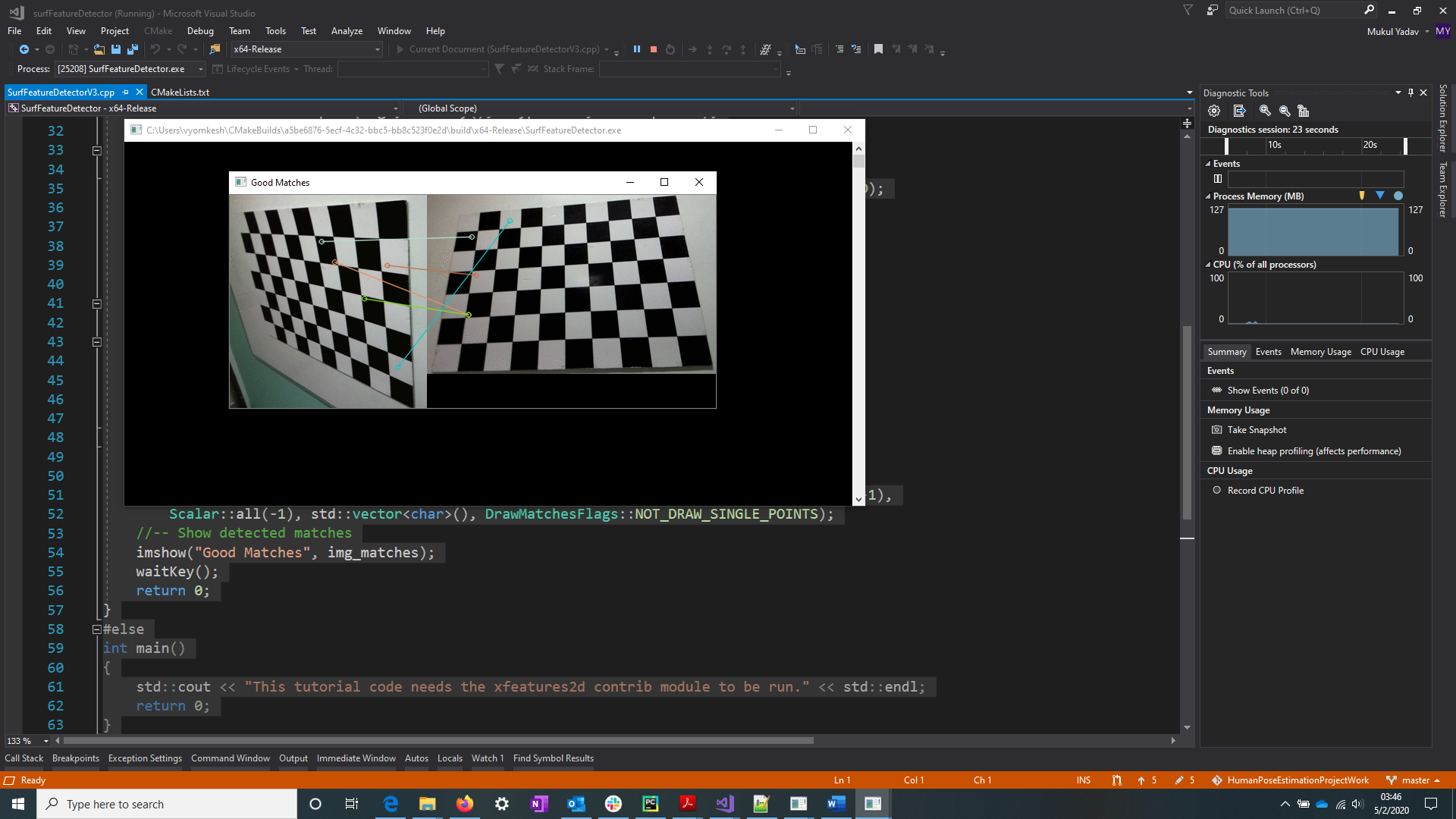
mchand.yadav@ufl.edu

UFID: 7585 9623

**Assignment 1:**

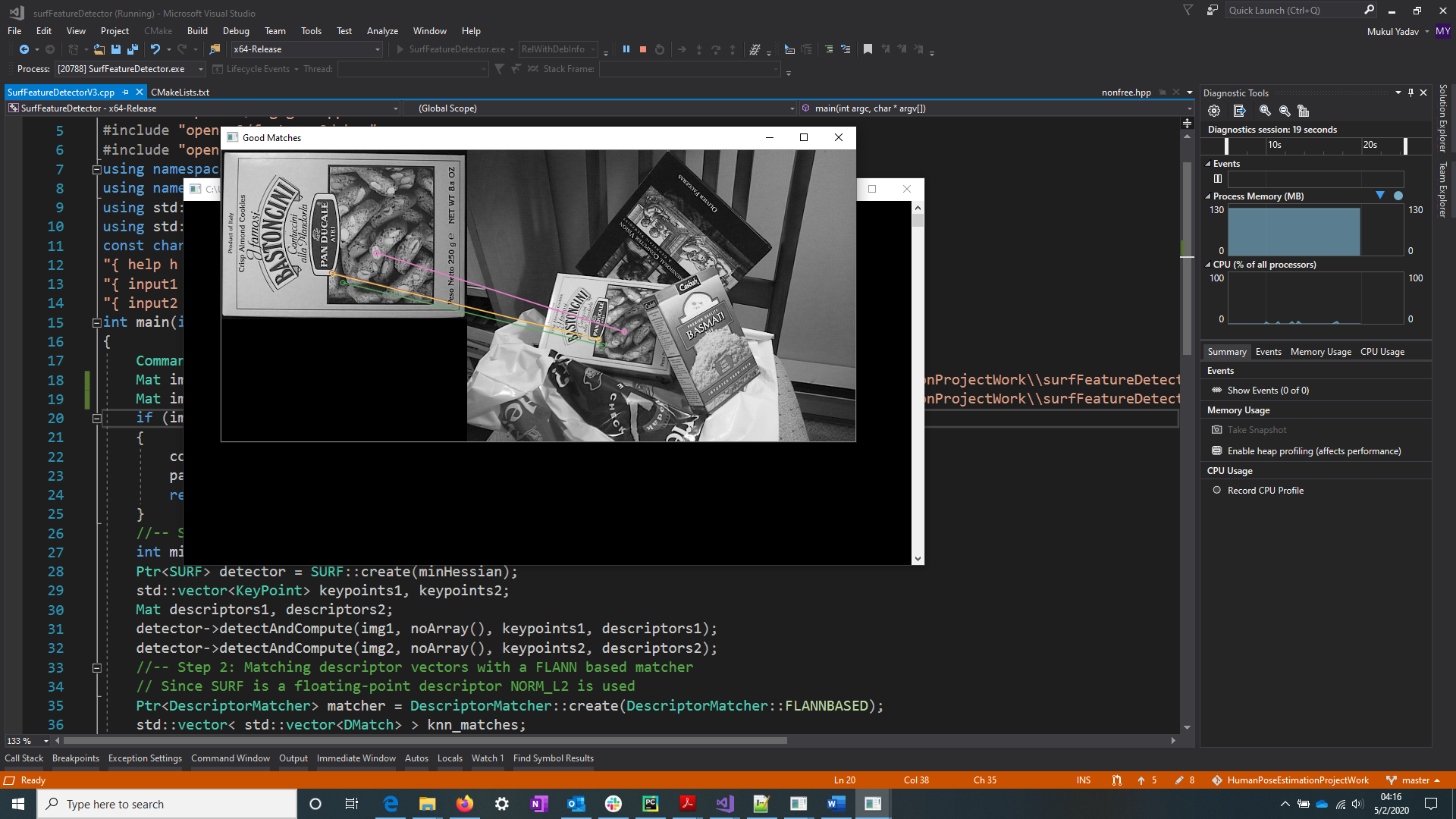
**Feature detection with SURF**

As part of this tutorial assignment, we leveraged SURF (Speeded Up Robust Features) feature detection extractor bundled with OpenCV to detect keypoints in provided image, i.e. chessboard orientation-based snapshots. The keypoints are further sorted and matched based on KNN matching algorithms and then connected using a line. As evident form results of 2 varying instances, the one with homologous formation such as chessboard doesn’t provide a better matching compared to one with a quite asymmetrical image set.



Matching on chessboard orientation. (Figure above)

Matching on a book from a pile of stationaries.(Figure below)



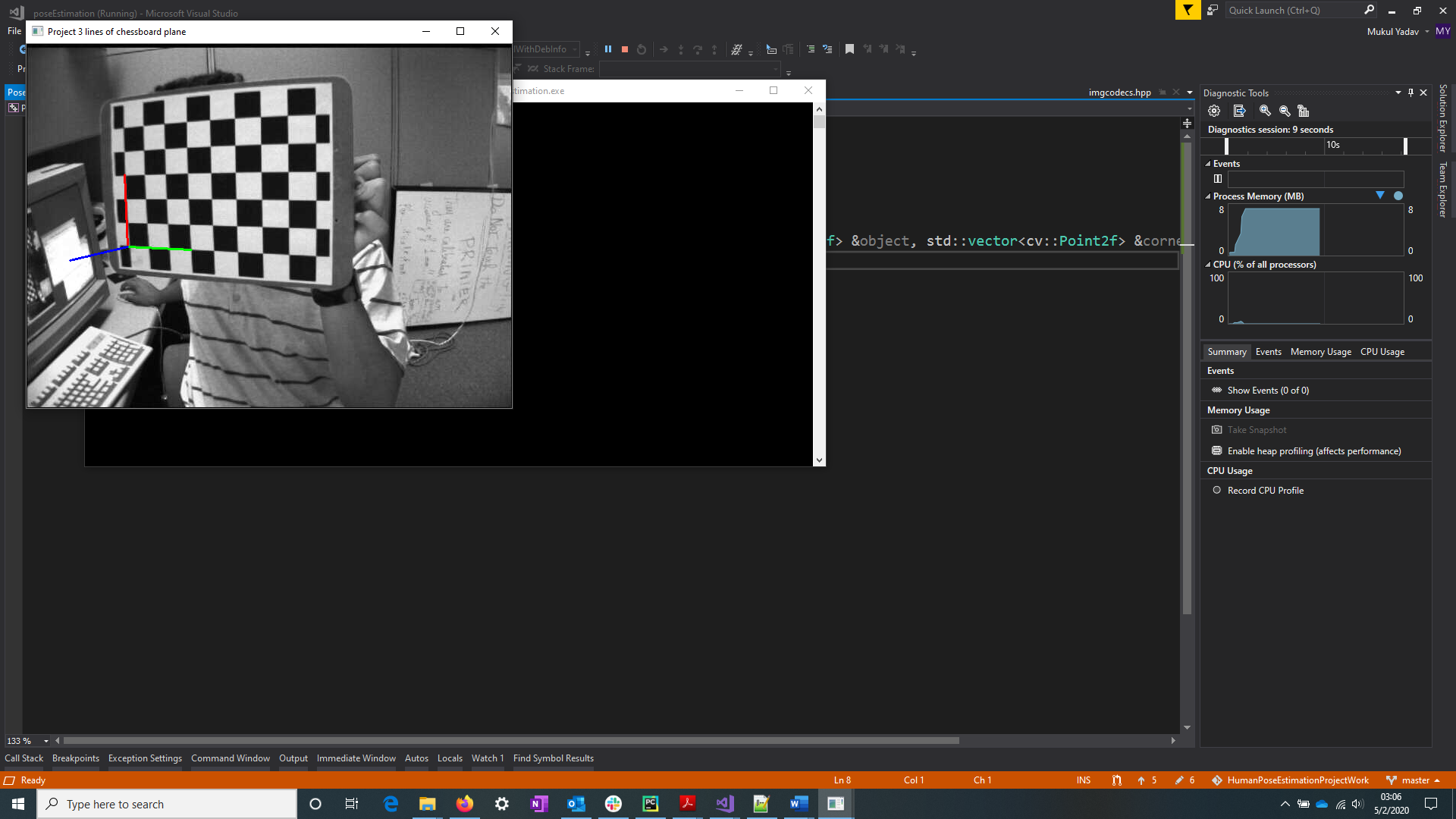
**Source code:**

1. #include <iostream>
2. #include "opencv2/core.hpp"
3. #ifdef HAVE\_OPENCV\_XFEATURES2D
4. #include "opencv2/highgui.hpp"
5. #include "opencv2/features2d.hpp"
6. #include "opencv2/xfeatures2d.hpp"
7. **using** **namespace** cv;
8. **using** **namespace** cv::xfeatures2d;
9. **using** std::cout;
10. **using** std::endl;
11. **const** **char**\* keys =
12. "{ help h |                  | Print help message. }"
13. "{ input1 | box.png          | Path to input image 1. }"
14. "{ input2 | box\_in\_scene.png | Path to input image 2. }";
15. **int** main(**int** argc, **char**\* argv[])
16. {
17. CommandLineParser parser(argc, argv, keys);
18. Mat img1 = imread("C:\\Users\\vyomkesh\\Documents\\UF\\S4\\Individual Study\\HumanPoseEstimationProjectWork\\surfFeatureDetector\\checkerboardOrientation1.png");
19. Mat img2 = imread("C:\\Users\\vyomkesh\\Documents\\UF\\S4\\Individual Study\\HumanPoseEstimationProjectWork\\surfFeatureDetector\\checkerboardOrientation2.png");
20. **if** (img1.empty() || img2.empty())
21. {
22. cout << "Could not open or find the image!\n" << endl;
23. parser.printMessage();
24. **return** -1;
25. }
26. //-- Step 1: Detect the keypoints using SURF Detector, compute the descriptors
27. **int** minHessian = 400;
28. Ptr<SURF> detector = SURF::create(minHessian);
29. std::vector<KeyPoint> keypoints1, keypoints2;
30. Mat descriptors1, descriptors2;
31. detector->detectAndCompute(img1, noArray(), keypoints1, descriptors1);
32. detector->detectAndCompute(img2, noArray(), keypoints2, descriptors2);
33. //-- Step 2: Matching descriptor vectors with a FLANN based matcher
34. // Since SURF is a floating-point descriptor NORM\_L2 is used
35. Ptr<DescriptorMatcher> matcher = DescriptorMatcher::create(DescriptorMatcher::FLANNBASED);
36. std::vector< std::vector<DMatch> > knn\_matches;
37. matcher->knnMatch(descriptors1, descriptors2, knn\_matches, 2);
38. //-- Filter matches using the Lowe's ratio test
39. **const** **float** ratio\_thresh = 0.7f;
40. std::vector<DMatch> good\_matches;
41. **for** (**size\_t** i = 0, j=0; i < knn\_matches.size() && j<5; i++)
42. {
43. **if** (knn\_matches[i][0].distance < ratio\_thresh \* knn\_matches[i][1].distance)
44. {
45. good\_matches.push\_back(knn\_matches[i][0]);
46. ++j;
47. }
48. }
49. //-- Draw matches
50. Mat img\_matches;
51. drawMatches(img1, keypoints1, img2, keypoints2, good\_matches, img\_matches, Scalar::all(-1),
52. Scalar::all(-1), std::vector<**char**>(), DrawMatchesFlags::NOT\_DRAW\_SINGLE\_POINTS);
53. //-- Show detected matches
54. imshow("Good Matches", img\_matches);
55. waitKey();
56. **return** 0;
57. }
58. #else
59. **int** main()
60. {
61. std::cout << "This tutorial code needs the xfeatures2d contrib module to be run." << std::endl;
62. **return** 0;
63. }
64. #endif

**Assignment 2:**

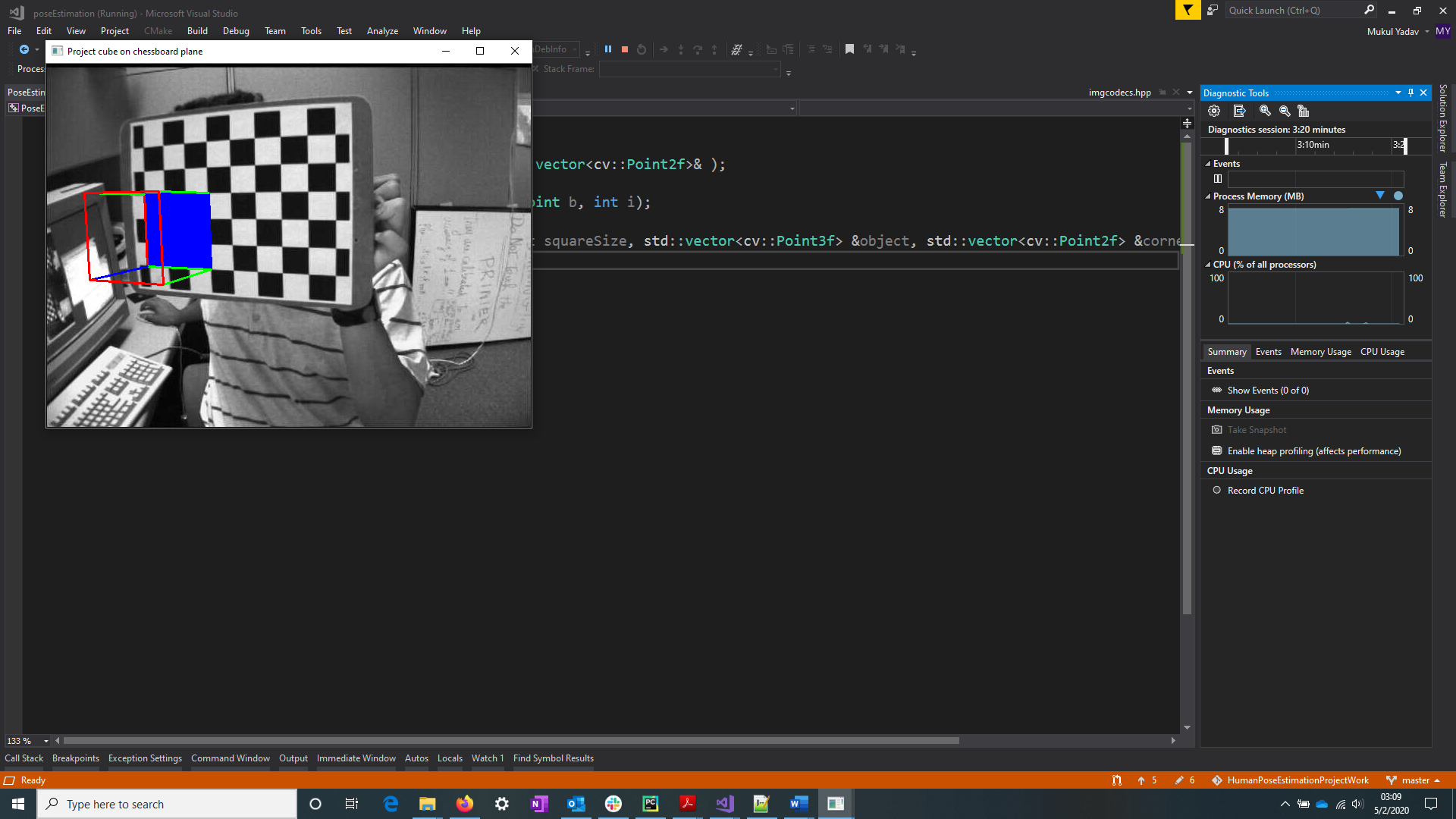
**Pose Estimation**

This tutorial assignment involves finding chess board corners followed by camera calibration and estimating chessboard plane rotation and translation vectors by employing OpenCV bundled solvePnP routine. SolvePnP helps resolve pose and perspective of image plane with respect to camera plane.



Then, we choose a point at random and project lines parallel to 3-axes incident on chessboard plane.(As seen in screenshot above)

This exercise is further extended by drawing a cubical polygon on the identified square of chessboard plane. (See below)



**Source code:**

1. #include <opencv2/opencv.hpp>
2. #include <opencv2/calib3d/calib3d.hpp>
3. #include <opencv2/highgui/highgui.hpp>
4. #include <opencv2/imgproc/imgproc.hpp>
5. #include <stdio.h>
6. #include <iostream>
7. #include <opencv2/core/utility.hpp>
8. #include "PoseEstimation.h"
10. **using** **namespace** cv;
11. **using** **namespace** std;
13. **int** main()
14. {
15. Mat img = imread("C:\\Users\\vyomkesh\\Documents\\UF\\S4\\Individual Study\\HumanPoseEstimationProjectWork\\poseEstimation\\resources\\images\\right01.jpg", IMREAD\_COLOR);
16. Mat outputMat, rvec, tvec;
17. cv::Size boardSize(6,9);
18. **const** **float** squaresize = 3.0;
19. cv::OutputArray ptVec = outputMat;
21. String filename("C:\\Users\\vyomkesh\\Documents\\UF\\S4\\Individual Study\\HumanPoseEstimationProjectWork\\poseEstimation\\resources\\univ\_intrinsics.yml");
22. FileStorage fs(filename, FileStorage::READ);
23. Mat intrinsics, distortion;
24. fs["camera\_matrix"] >> intrinsics;
25. fs["dist\_coefs"] >> distortion;
27. std::vector<Point3f> boardPoints; // fill the array as described at 3. ...
28. boardPoints.push\_back(Point3f(3, 0, 0));
29. boardPoints.push\_back(Point3f(0, 3, 0));
30. boardPoints.push\_back(Point3f(0, 0, 3));
32. vector<Point3f> objectpoints, testobjectpoints;
34. testobjectpoints.push\_back(Point3f(9.0, 0, 0));
35. testobjectpoints.push\_back(Point3f(0, 9.0, 0));
36. testobjectpoints.push\_back(Point3f(0, 0, -9.0));
37. testobjectpoints.push\_back(Point3f(0, 0, 0));
39. **bool** found = findChessboardCorners(img, boardSize, ptVec, CALIB\_CB\_ADAPTIVE\_THRESH);
41. vector<Point2f> point2fVecFromMat, resultCorners;
42. Mat\_to\_vector\_Point2f(ptVec, point2fVecFromMat);
44. calcChessboardCorners(boardSize, squaresize, objectpoints, point2fVecFromMat);
46. Point p0, p1, p2, p3;
47. **if** (found)
48. {
49. solvePnP(Mat(objectpoints), point2fVecFromMat, intrinsics, distortion, rvec, tvec, **false**);
50. projectPoints(Mat(testobjectpoints), rvec, tvec, intrinsics, distortion, resultCorners);
51. std::cout << resultCorners;
53. p1.x = resultCorners[0].x;
54. p1.y = resultCorners[0].y;
55. p2.x = resultCorners[1].x;
56. p2.y = resultCorners[1].y;
57. p3.x = resultCorners[2].x;
58. p3.y = resultCorners[2].y;
59. p0.x = resultCorners[3].x;
60. p0.y = resultCorners[3].y;
62. Createline(img, p0, p1, 1);
63. Createline(img, p0, p2, 2);
64. Createline(img, p0, p3, 3);
66. namedWindow("Project 3 lines of chessboard plane");
67. imshow("Project 3 lines of chessboard plane", img);
68. waitKey(0);
69. }
71. testobjectpoints.push\_back(Point3f(9, 9, -9.0));
72. testobjectpoints.push\_back(Point3f(9, 9, 0));
73. testobjectpoints.push\_back(Point3f(0, 9, -9.0));
74. testobjectpoints.push\_back(Point3f(9, 0, -9.0));
76. projectPoints(Mat(testobjectpoints), rvec, tvec, intrinsics, distortion, resultCorners);
77. std::cout << resultCorners;
78. Point p4, p5, p6, p7;
79. p1.x = resultCorners[0].x; p1.y = resultCorners[0].y;
80. p2.x = resultCorners[1].x; p2.y = resultCorners[1].y;
81. p3.x = resultCorners[2].x; p3.y = resultCorners[2].y;
82. p0.x = resultCorners[3].x; p0.y = resultCorners[3].y;
83. p4.x = resultCorners[4].x; p4.y = resultCorners[4].y;
84. p5.x = resultCorners[5].x; p5.y = resultCorners[5].y;
85. p6.x = resultCorners[6].x; p6.y = resultCorners[6].y;
86. p7.x = resultCorners[7].x; p7.y = resultCorners[7].y;

89. std::vector<cv::Point>drawpic;
90. drawpic.push\_back(p0);
91. drawpic.push\_back(p2);
92. drawpic.push\_back(p5);
93. drawpic.push\_back(p1);
94. std::vector<std::vector<cv::Point>>contours;
95. contours.push\_back(drawpic);
96. fillPoly(img, contours, Scalar(255, 0, 0));
97. Createline(img, p0, p1, 3);
98. Createline(img, p0, p2, 3);

101. Scalar color;
102. Createline(img, p0, p1, 1);
103. Createline(img, p0, p2, 2);
104. Createline(img, p0, p3, 3);
105. //Createline(img, p0, p3, 2);
106. Createline(img, p2, p5, 3);
107. Createline(img, p1, p5, 3);
108. Createline(img, p2, p6, 2);
109. Createline(img, p5, p4, 2);
110. Createline(img, p1, p7, 2);
111. Createline(img, p6, p4, 1);
112. Createline(img, p4, p7, 1);
113. Createline(img, p7, p3, 1);
114. Createline(img, p6, p3, 1);


118. namedWindow("Project cube on chessboard plane");
119. imshow("Project cube on chessboard plane", img);
120. waitKey(0);
122. **return** 0;
123. }

126. //vector\_Point2f
127. **void** Mat\_to\_vector\_Point2f(OutputArray& oa, std::vector<Point2f>& v\_point)
128. {
129. v\_point.clear();
131. // Should Work on any OpenCV version
132. cv::Mat mat = oa.getMat();
133. Point2f \*data = (Point2f \*)mat.data;
134. **int** length = mat.total();
135. std::vector<Point2f> input;
136. v\_point.assign(data, data + length);
138. }
140. **void** calcChessboardCorners(Size boardSize, **float** squareSize, vector<Point3f> &object, vector<Point2f> &corners){
141. **for** (**int** i = 0; i < boardSize.height; i++)
142. **for** (**int** j = 0; j < boardSize.width; j++)
143. object.push\_back(Point3f(**float**(j\*squareSize), **float**(i\*squareSize), 0));
144. }
146. **void** Createline(Mat inputma, Point a, Point b, **int** i)
147. {
148. Scalar colors = Scalar(255, 0, 0);
149. **if** (i == 1){
150. colors = Scalar(0, 0, 255);
151. }
152. **else** **if**(i == 2) {
153. colors = Scalar(0, 255, 0);
154. }
155. line(inputma, a, b, colors, 2, LINE\_8);
156. }

**Resources used:**

* [**http://www.planetb.ca/syntax-highlight-word**](http://www.planetb.ca/syntax-highlight-word) **(Copy & paste from IE for usage in word)**