

DESY Summer Student Report



CMS Tracker Upgrade: Position detection for PS Module Assembly

Katja Strobel, Furtwangen University, Germany

September 9, 2015

Abstract

For the high luminosity upgrade of the Large Hadron Collider, in 2023, the tracking detector of the CMS experiment, at CERN in Geneva, needs to be updated. DESY, as part of the collaboration, needs to deliver several hundred PS sensor modules for the silicon strip detector. The PS modules consist of various parts, which have to be assembled precisely. In order to ensure precision and repetitive results, an automated assembly system is under development. In this system the upper part is handled by a vacuum gripper, dipped into glue and placed on the bottom part. To ensure the necessary alignment precision a high-resolution camera is part of the system. This report is about creating the code to detect the position of the upper part on the gripper and return its position and angle, to compensate for any movement before connecting the two parts.

Contents

1	Introduction	3
2	Assembly Process	4
3	Scope of Work	4
4	Method	5
	Step 1. OpenCV	5
	Step 4. Detection method for test images	6
	cv::SurfFeatureDetector & cv::FlannBasedMatcher	6
	cv::HoughCircles & cv::cornerHarris	7
	Step 5. Real camera image	8
	Step 6. Test with real camera image	9
	Step 7. Detection method for real image	9
	cv::HoughCircles & cv::HoughLines	9
5	Conclusion	10

1 Introduction

For the high luminosity upgrade of the Large Hadron Collider (HL-LHC), in 2023, the tracking detector of the CMS experiment, at CERN in Geneva, needs to be replaced. The tracking detector itself consists of pixel detectors and strip detectors. The strip detectors consist of several thousand square shaped sensor modules assembled in cylindrical shape, see Figure 1. For the tracking detector upgrade in 2023 this sensor modules has to be replaced by smaller ones with new technology. DESY, as part of the collaboration, needs to deliver several hundred of the PS modules. The PS modules consist of various single parts, which have to be assembled precisely for the detector to reach the required performance. In order to ensure precision and repetitive, reliable results an automated assembly system is currently under development.

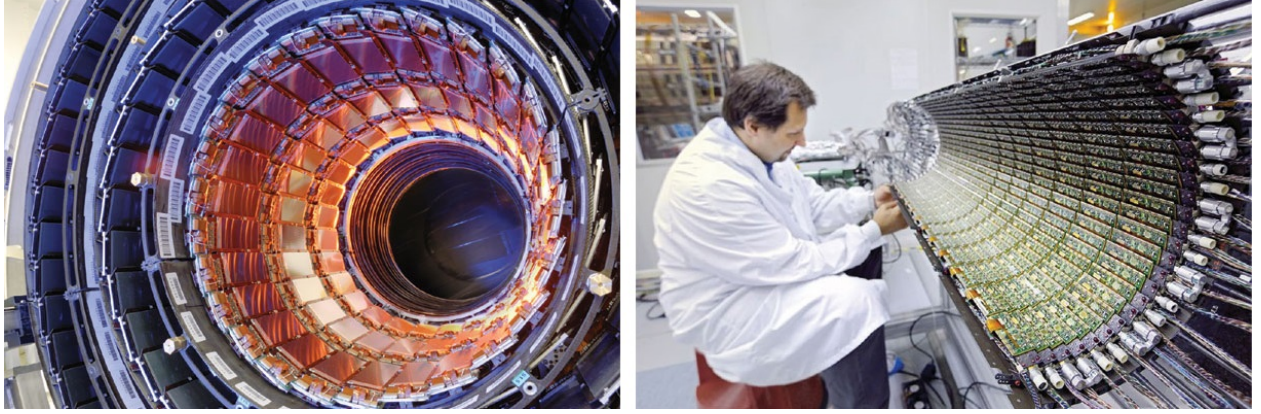


Figure 1: CMS tracking detector

In Figure 2 all components of the PS module are shown. The sensor module production consists of several assembly steps. The focus of this report lies on the gluing process of the the already pre-assembled PS-p silicon sensor (2) with the MPAs (3) to the CFRP base plate (5). The PS-p silicon sensor (2) combined with the MPAs (3) is further referred to as upper part and the carbon fibre reinforced plastic (CFRP) base plate (5) is further referred to as bottom part.

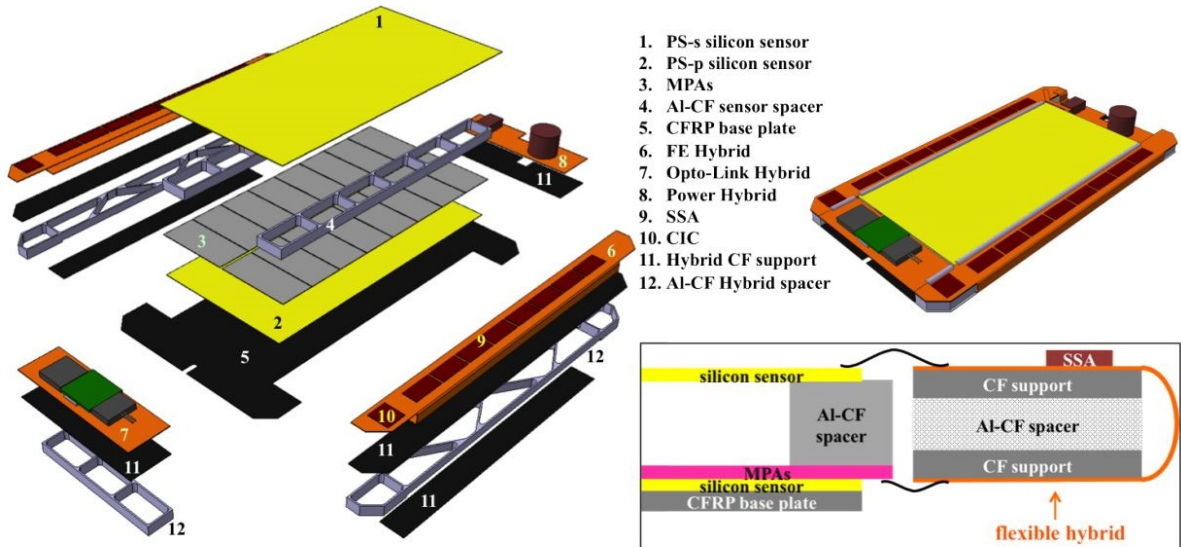


Figure 2: Exploded view of PS module components (left), assembled module (right)

2 Assembly Process

The automated assembly station for this gluing process is shown in Figure 3. The general process is as follows: the upper part is collected by a vacuum gripper, moved to a receptacle with glue, dipped into glue and placed on the bottom part. The gripper is attached to an arm that can move in X, Y and Z direction. The bottom part is fixed manually, in the correct position, to a round plate. The plate can do a precise rotational movement to compensate for possible rotational misalignment.

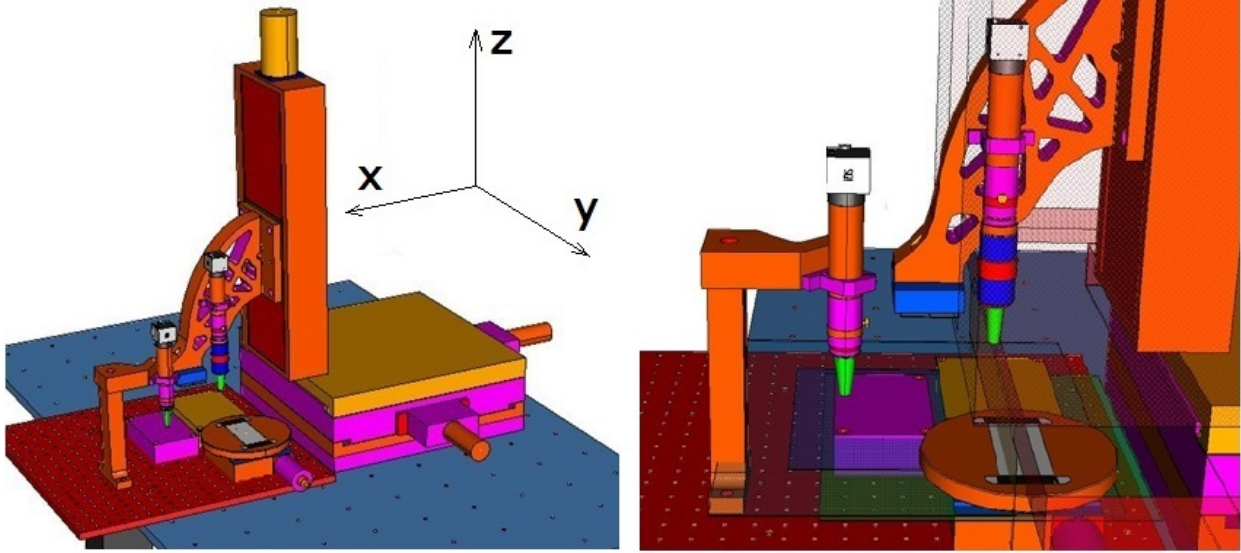


Figure 3: Assembly station PS modules

3 Scope of Work

The upper part when collected by the vacuum gripper might be displaced. To ensure the necessary alignment precision of $\pm 10\mu m$, the actual position of the upper part on the gripper needs to be detected, so that the displacement can be compensated for before the two pieces are combined. Therefore the assembly station is equipped with a high resolution camera, to provide images of the actual position of the sensor module on the gripper. On the PS modules on each corner is a L-shaped alignment mark $720 \times 700\mu m$ big. The current plan to detect the accurate position of the upper part on the gripper is to move the upper part under the camera, detect the shape of the alignment mark, return the values of X and Y axis displacement and displaced angle around Z axis.

The task was to write a program that detects the alignment marks and returns the values. The following requirements were given:

- Programming language C++
- Library or program for image detection has to be open source
- Robust algorithm (not influenced by dirty surfaces, scratches or light conditions)
- Algorithm should be able to detect 360° rotation of alignment mark

4 Method

In this chapter first the undertaken steps to complete the summer student project are stated as an overview. Afterwards the most important steps are explained in more detail.

1. Research on image detection programs (2 days)
2. Install C++ and include OpenCV Library (0.5 days)
3. Learn how to program in C++ and to use OpenCV (1-2 weeks)
4. Program algorithms to detect contours in test images
5. Get real camera image
6. Test algorithm with real image
7. Adjust algorithm that it works for real image and returns displacement in X,Y axis and angle around Z axis
8. Test program with different images and adjust parameter of algorithms to make code robust
9. Comment code to make it coherent to other people

As the assembly station was not finished at the time the summer student program started, high contrast images of the alignment mark were provided for the first programming tests. The simplified alignment mark images are shown in Figure 4. Image number 1 should serve as reference image, the ideal position, whereas number 2 is a displaced image and image number 3 is displaced and rotated.

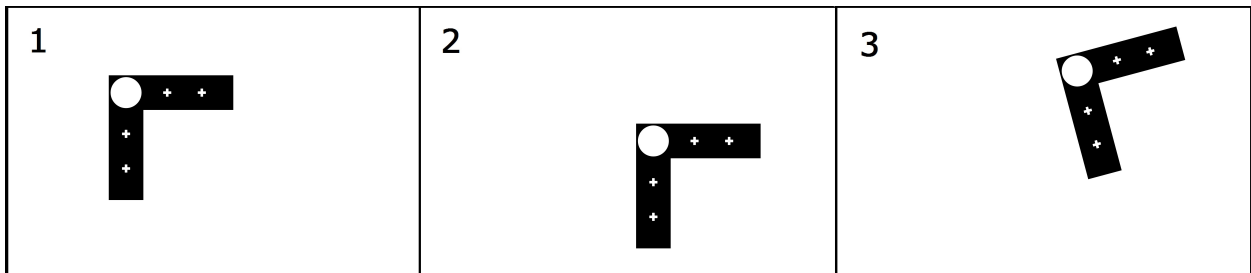


Figure 4: High contrast alignment marks

Step 1. OpenCV

Open Source Computer Vision was selected as image detection program. It is a class library mainly aimed at real-time computer vision. It can be included into C++ and Python Programs.

Step 4. Detection method for test images

After several algorithms were read into and compared, for the following algorithms test programs were created to find a solution for the image detection.

`cv::SurfFeatureDetector` & `cv::FlannBasedMatcher`

The "SurfFeaturDetector" detects several keypoints on the images. An algorithm selects than the good matches and the "FlannBaseMatcher" matches similar keypoints on two images together and draws a line between the two matched points. The result in comparing the reference image 1 with the displaced image 2, can be seen in Figure 5. It can be seen, that

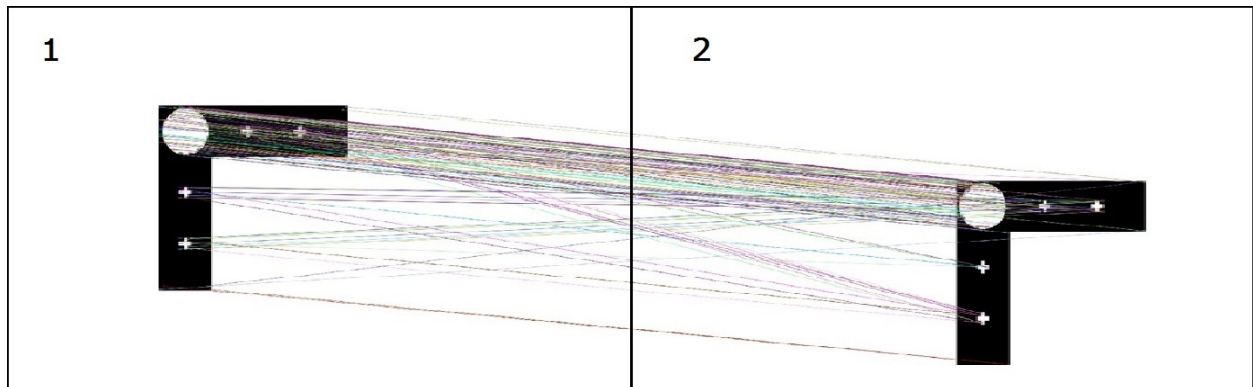


Figure 5: SurfFeatureDetector

some point were good matches, e.g. at the circle, and others, especially at the four white crosses, are bad matches. Bad matches are the lines that go to the wrong point on the second image. In order to find out what causes this wrong matches the detected keypoints from SurfFeatureDetector are investigated further. It turns out the keypoints consist not only of x and y coordinate but of 8 parameters, e.g. size - keypoint diameter, angle - keypoint orientation, etc. After this discovery the assumption is made, that the angle or keypoint orientation is responsible for the bad matches. By changing a parameter from DEFAULT to DRAW RICH KEYPOINTS the keypoints are drawn not only as points, but with size and orientation, which is indicated as a line from the center to the radius. How the keypoints actually look like can be seen in Figure 6.

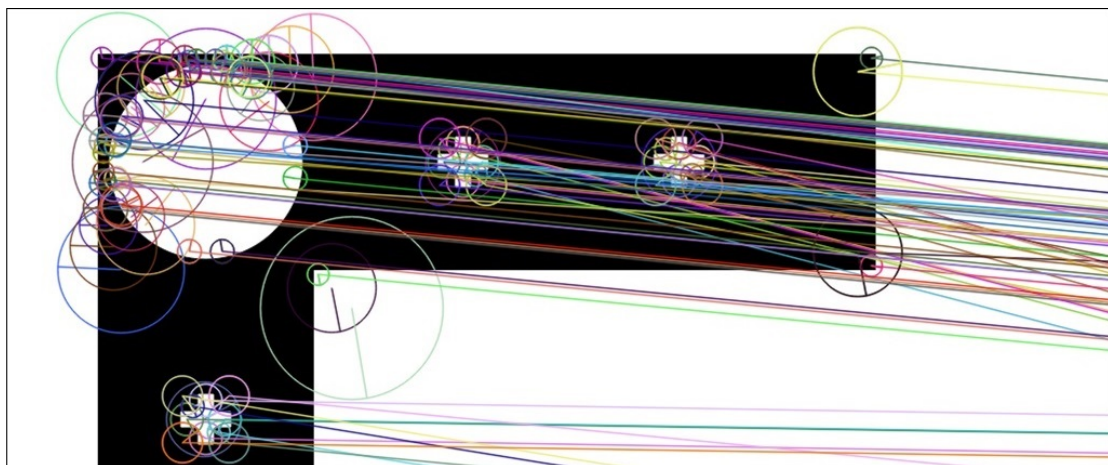


Figure 6: Keypoints with orientation

As seen from the image the keypoint orientation is random and has no effect on the matching to the other image as seen. However an example code with this method on the OpenCV library info page [3] showed in despite of some bad matches the contour are successfully detected. To find out if the contour detection for our example works even with the bad matches, the code is modified to draw a green line around the detected contour. In Figure 7 the results are shown on the left side for the displaced image and on the right side for a displaced and rotated image, both compared with the same reference image that is not shown here.

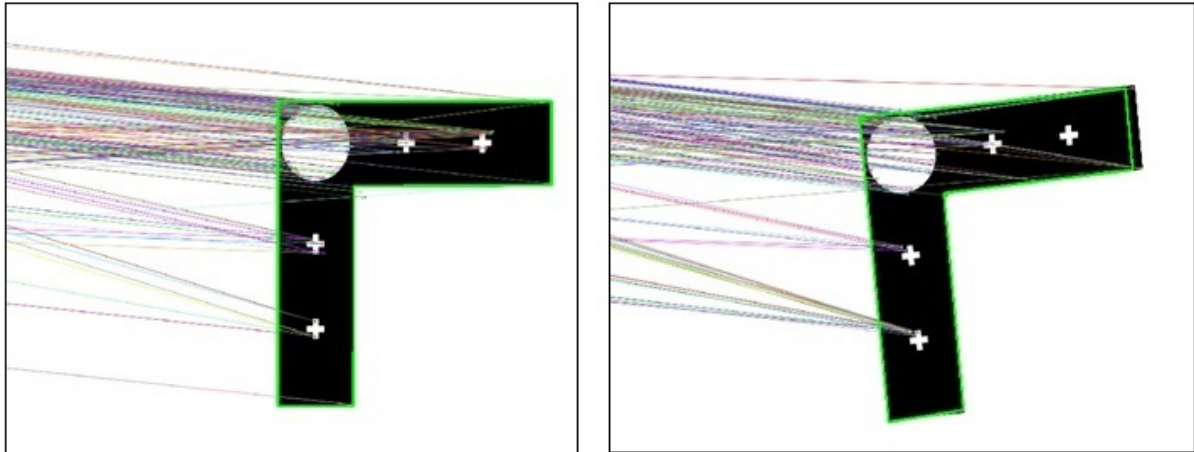


Figure 7: Detected contours

It can be seen that the displaced image works well but already a slightly rotated image shows significant inaccuracy. And with over 10° rotation the detection does not work anymore. This shows that this method is not suitable for the purpose.

cv::HoughCircles & cv::cornerHarris

As the circle detection seem to work best the HoughCircles algorithm is used to detect the circle. Now only one more reliable point in each image is needed to determine the orientation in the camera frame. To get the remaining point first of all the cornerHarris algorithm is used to detect all the corners on the image. Then the distance from the center to all the points is calculated and saved in a matrix. Now the smallest distance that is bigger than zero is selected. The point that belongs to this distance has to be the inner corner of the alignment mark. For better visualization a green line is drawn from the detected circle to the selected corner. The result can be seen in Figure 8. These method works well for the high resolution test images, now the program has to be tested at the real image.

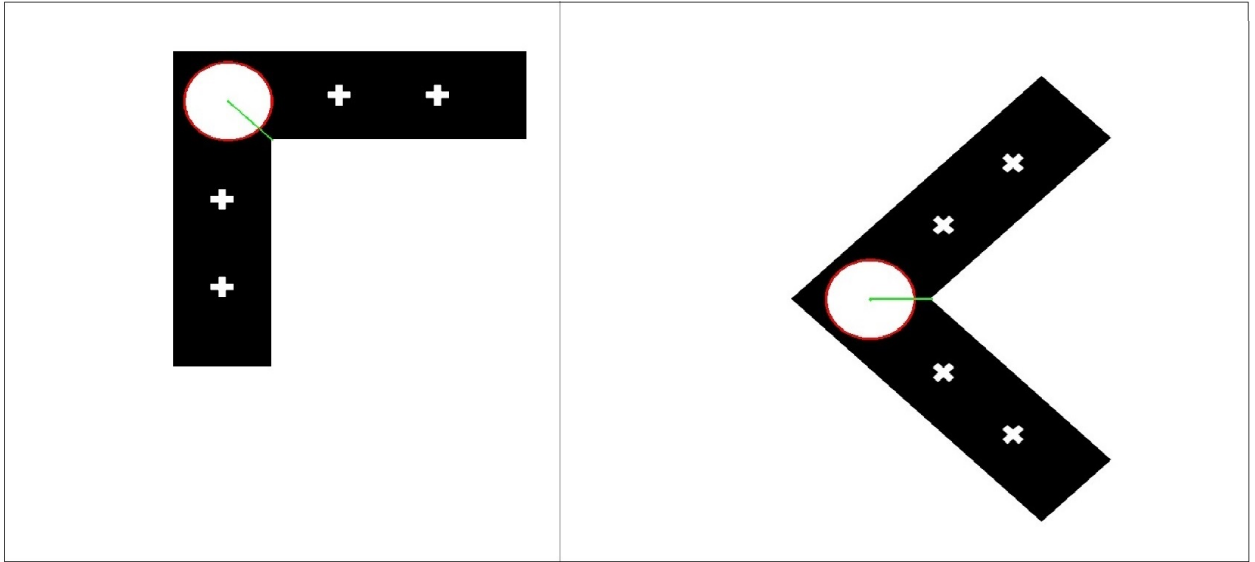


Figure 8: Result of HoughCircles and CornerHarris

Step 5. Real camera image

As the assembly station had a technical issue the real camera image from the assembly station was not available. In order to be able to test the program for the real image anyways a camera holder was designed and built. As safety instructions were necessary to operate the machines in the mechanical tool shop help was provided from a colleague of the collaboration. The camera holder with camera and the PS module is shown in Figure 9 and the real image taken is shown in Figure 10.



Figure 9: Camera holder with camera and PS module

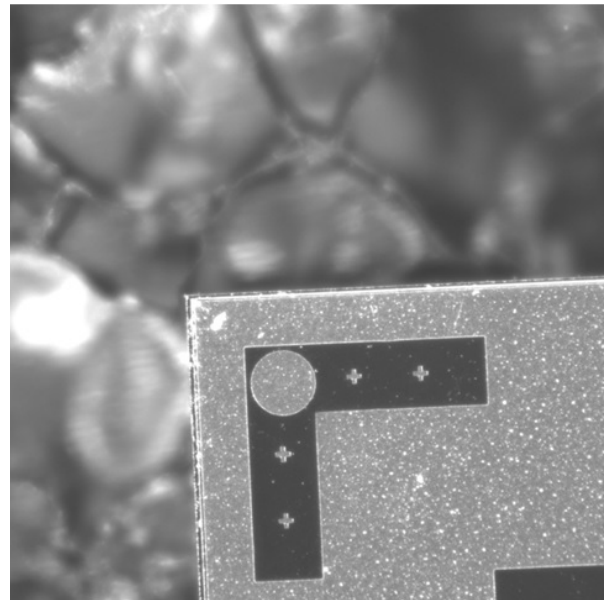


Figure 10: Real camera image

Step 6. Test with real camera image

The HoughCircles & cornerHarris algorithms from step 4 does not work for the real image. The circle is detected but the cornerHarris detection failed as several points on the image are detected even though at this positions there are no actual corners. It was tried make the corner detectable by adapt the image by adding color filters but without success. Possible reasons could be that the result depends on the light conditions or that the surface has dirt on it. However another method to find the inner corner is needed.

Step 7. Detection method for real image

cv::HoughCircles & cv::HoughLines

As the HoughCircles detection works for the real image, the idea is to use HoughLines detection to detect all lines on the image. At the HoughLines algorithm the minimum length of the lines can be specified. In order to not detect the lines from the crosses, a minimum line length is selected. Then all intersections of the detected lines are computed. And as a last step the nearest intersection to the circle center that is bigger than the radius of the circle is selected. This intersection has to be the inner corner. In Figure 11 the final result can be seen.

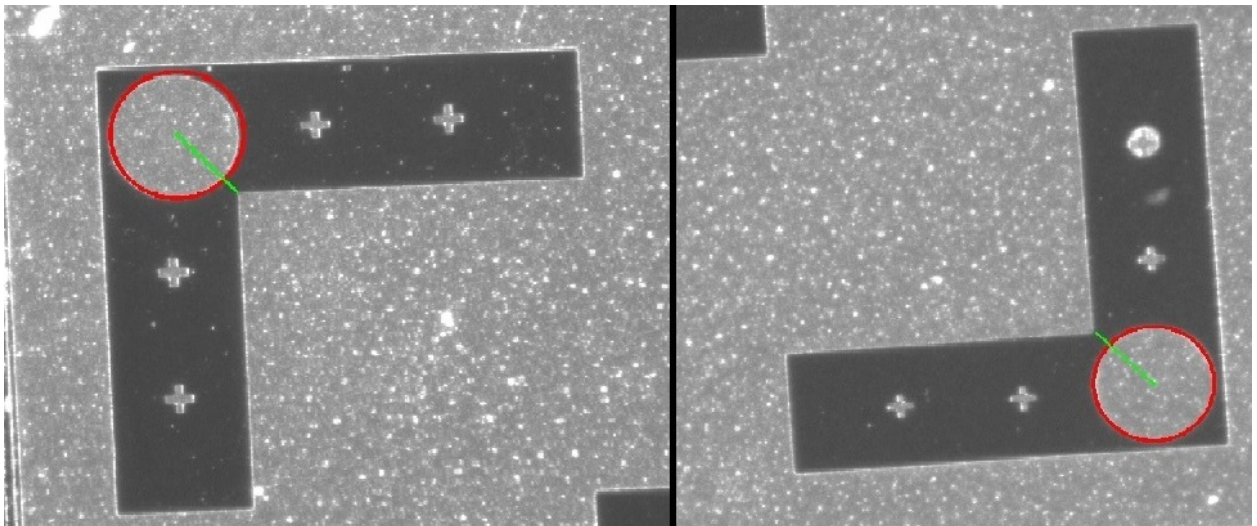


Figure 11: Final result

5 Conclusion

In my point of view the project was complied successfully. The result looks very promising and as the code works for all 4 real camera images that were made it is assumed that it is robust to dirt and scratches on the surface. When the assembly station is working the code might need some adjustments at some parameters, due to different light conditions and the different camera settings, but in general the code is working. The parameters that might need adjustment are commented in the code, therefore it should be easy for a third person to find them and adjust them. Also the overall commenting of the code in English language should make it easy for everyone with programming knowledge to understand what the code is doing.

References

- [1] DESY paper *Phase II Technical proposal - Tracker chapter*, July 29, 2014
- [2] The CMS Collaboration, *CMS Technical Design Report for the Pixel Detector Upgrade*, CERN-LHCC-2012-016.
- [3] OpenCV Library Infopage <http://docs.opencv.org/index.html>