Quality Inspection and Product Validation
MEC220T | MEC220P

Final Project Report

Study and Analysis by Sujith J (MDM18B052), Sreekiren D S (MDM18B050), Soorya Sriram (MDM18B049), Khirupasagar R (MDM18B063)

Title : Inspection, Validation of parts using Shape Detection by Harris corner detection and Canny Edge Detection

→ Abstract

The below study and analysis is about inspecting the quality of a product, which is being manufactured or already out of the plant,i.e., in the customer's hands by using image processing techniques to measure and inspect the dimensions of an n-sided polygon. We shall start this paper by briefly explaining what the analysis is about and then move on to the present methods as well as our method, taking into account the advantages and disadvantages of both. We shall then study the core theory and look into the algorithm, codes and hence analyse the results. Concluding the paper, we will discuss the applications, feasibility and further extensions of the same. The references are stated in the end.

→ What is the study about?

This study correlates to our course in regard to the same domain of inspection and validation. The scratch of the study is our products in hand, that is they are the data we have, the job lies in inspecting this part, by using our image processing techniques and classify them if they can be accepted or rejected, i.e., in simple if they are within the dimensional and tolerance limits.

Quality is an important factor that has to be given the utmost priority when a product is bought into the market. And when it comes to bolts and nuts, it just looks like a small component, but plays a major role in all the major machinery from a bench vice to an aeroplane. Even small shape changes might cause biggest catastrophes, like we might have seen in some air accidents, when a small rivet is the whole cause. There are many such consequences ranging from the threaded part of a screw struck in a hole tap to collapsing of bridges, a bolt or nut defect can be a reason. Hence, it is very important to inspect the part (bolt and nut) before it is being used in any other application.

→ What are the present methods?

Sometimes the dimensions or tolerances are not even inspected after manufacturing the products nor the customers take effort in checking them. Even if some checking is done, it is by the following methods:

1. **Visual and Manual Inspection:**

    - Visual inspection is also the most widely used method for detecting, examining parts, its shape and corresponding dimensions.
    - Visual inspection is often performed to verify and more closely examine once the part has been manufactured and its dimensions are known.
    - The methods of visual inspection involve a wide variety of equipment, ranging from examination with the naked eye to the use of microscopes and other measurement devices for measuring the dimensions of the parts.

    Advantages:

    - Investment like purchasing devices and equipment and extensive labour need not be put in this mode of inspection and is far more cost efficient.
    - It is widely used for inspecting and determining the surface cracks as there are no requirements of prior knowledge of the measurement devices.

    Disadvantages:

    - Only surface inspection and large flaws can only be identified using visual inspection.
    - Manual inspection, using vernier callipers, micrometer and others lead to less accuracy due to parallax errors and backlash errors, thereby causing errors while validating the parts.

2. **CMMs:**

    - Coordinate measuring machines (CMMs) are used to inspect the dimensions of a finished product.
    - CMMs consist of the machine itself and its probes and moving arms for providing measurement input, a computer for making rapid calculations and comparisons based on the measurement input, and the computer software that controls the entire system.

- Coordinate measuring machines are primarily characterized by their flexibility, being able to make many measurements without adding or changing tools.

Advantages:

- High precision and accuracy
- Requires Less Labor
- Accurate dimensions can be obtained by knowing the coordinates and distance between the two reference points.

Disadvantages:

- The cost of CMMs is so high that small scale industries could not afford them.
- Probe system errors : Every probe system introduces inaccuracies to your data point gathering.These include capture speed, lobe errors, stylus lengths and probe contact pressure. Understanding the errors within the probe system can be very difficult. The lobe errors are mechanical limitations
- Geometric errors : These include linear displacement errors , vertical and horizontal straightness errors, angular errors.
- Environmental errors : The variation in temperature of the environment can lead to error in measurements
- Space appears as a constraint as these machines are large and not compact.

→ Our method : Using Shape Detection by Harris corner detection and Canny Edge Detection

Procedure:

- Image of good clarity is feeded
- Colour image if given as input is converted to a grayscale image.
- The boundary or edges of the image are obtained using canny edge detection.
- Then , the coordinate points of the part are calculated using harris corner detection

- Each side dimension of the part (nut) is found using Coordinate distance formula and the diameter of the part is also found using the same.
- Then, the calculated distance parameters are compared to the size of the respective bolt head or appropriate spanner to the nut.
- Finally, the user is informed if the spanner can be used on the given nut and whether the nut (part) is in accordance with the standard values of the bolt heads or spanner sizes.

About the method :

- Our method is a contact less inspection and measurement method and always accurate when operated within the exact given conditions.
- When properly applied, machine vision can provide accurate and inexpensive inspection of workpieces, thus dramatically increasing product quality.
- Machine vision is also used as an in-process gaging tool for controlling the process and correcting trends that could lead to the production of defective parts.
- This ability to acquire an image, analyze it, and then make an appropriate decision is extremely useful in inspection and quality control applications.

Advantages:

- The presence of a Gaussian filter allows removal of any noise in an image.
- Detects the edges in a noisy state by applying the thresholding method.
- It can be used for a variety of functions, including: identification of shapes, measurement of distances and ranges, gauging of sizes and dimensions, determining orientation of parts and much more.
- Cost and time are highly reduced during this process.

Disadvantages:

- If the amount of smoothing required is important in the spatial domain it may be slow to compute.
- For high accuracy, the quality of images should be very high and images must be clear.
- Harris corner detection gives better results in black and white background and while converting color image to b&w image there would be data point losses.

- On a real-time analysis, the images are to be captured and processed quickly, it is to be made sure that there is no compromise on quality in this process, as the rate of manufacturing is high.

→ Theory

There are two main concepts we have used in studying the idea and formulating the results, they are '**Harris corner detection**' and '**Canny Edge Detection**'. In both of these methods, we start by smoothing the image and thus Gaussian Filters are used for that. Let us first understand these concepts and then go to the algorithm and code in the next section.

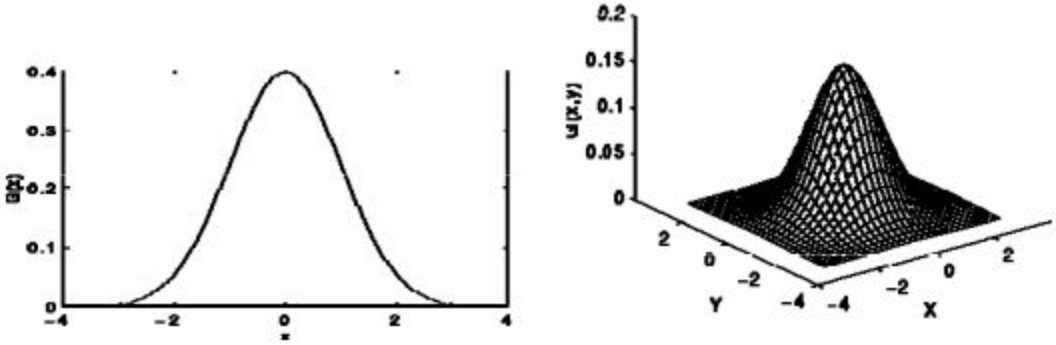
1. Gaussian Filtering

Gaussian filtering, also called the gaussian blur is a method used in image processing to remove noise by blurring the images to some extent, based on the parameters set by the user. The Gaussian function, also called a smoothing operator is basically a probability distribution for noise or data and given by :

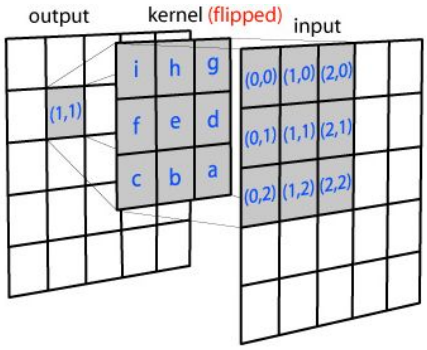
$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \quad G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

$$H_{ij} = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(i-(k+1))^2 + (j-(k+1))^2}{2\sigma^2}\right); 1 \le i, j \le (2k+1)$$

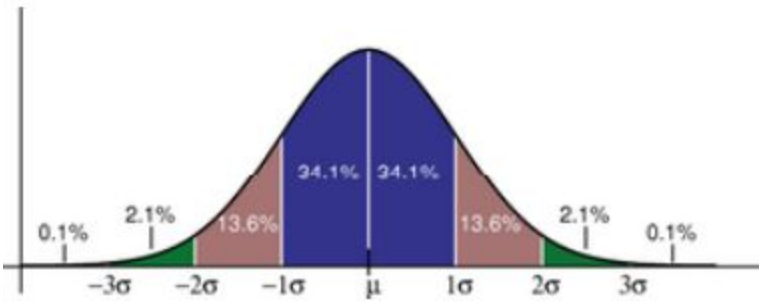
The distribution, as shown in the figure is assumed to have a mean of 0. Sigma(σ), refers to the standard deviation and plays an important role in determining the Gaussian Kernel. A kernel is nothing but a square or array of pixels, whose elements are determined by the function given above. Important parameters that influence the array elements are variance, radius and number of pixels.



In the process of blurring the image, each pixel of the image gets multiplied with the corresponding elements of the kernel matrix. For example, the value at (1,0) in the input array by the value at (h) in the kernel array, and so on. Hence, the final image resulting after this product appears to be our blurred image.



The Gaussian kernel can be 1D as well as 2D (we deal with 2D kernels) and we can intuitively say that the larger the kernel matrix then more expensive is the operation. But an advantage we have here is that the kernel matrix for the gaussian blur is always symmetrical, thus you can also multiply each axis (x and y) independently, which will decrease the total number of multiplications.This is because the Gaussian function is symmetrical. The values located between +/- σ account for 68% of the set, while two standard deviations from the mean (blue and brown) account for 95%, and three standard deviations (blue, brown and green) account for 99.7%. account for 99.7%. An example of a 5*5 kernel is given below, keeping the mean as 0 and standard deviation as 1.



$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Some Important points to be kept in mind, when using the Gaussian Filtering is that:

The Gaussian filter works by using the 2D distribution as a point-spread function, this is achieved by convolving the 2D Gaussian distribution function with the image. We need to produce a discrete approximation to the Gaussian function. This theoretically requires an infinitely large convolution kernel, as the Gaussian distribution is non-zero everywhere. Fortunately the distribution has approached very close to zero at about three standard deviations from the mean. 99% of the distribution falls within 3 standard deviations which means we can normally limit the kernel size to contain only values within three standard deviations of the mean.

The Gaussian filter is a non-uniform low pass filter and hence the kernel coefficients diminish with increasing distance from the kernel's centre. Central pixels have a higher weighting than those on the periphery. Larger values of σ produce a wider peak (greater blurring), thus, Kernel size must increase with increasing σ to maintain the Gaussian nature of the filter. Gaussian kernel coefficients depend on the value of σ. At the edge of the mask, coefficients must be close to 0. The kernel is rotationally symmetric with no directional bias and the Gaussian kernel is separable, which allows fast computation. Gaussian filters might not preserve image brightness.

2. Harris Corner Detection

The Harris point detection method is a standard technique for identifying and locating interest points in an image. The main idea is based on Moravec's detector, that relies on the autocorrelation function of the image for measuring the intensity differences between a patch and windows shifted in several

directions. The success of the Harris detector resides in its simplicity and efficiency.

The idea can be expressed through the autocorrelation function as follows

$$E(h) = \sum w(x) \left(I(x+h) - I(x)\right)^2$$

The function w(x) allows selecting the support region (window like areas in the figure below) that is typically defined as a rectangular or Gaussian function. Taylor series expansion can be used to arrive at

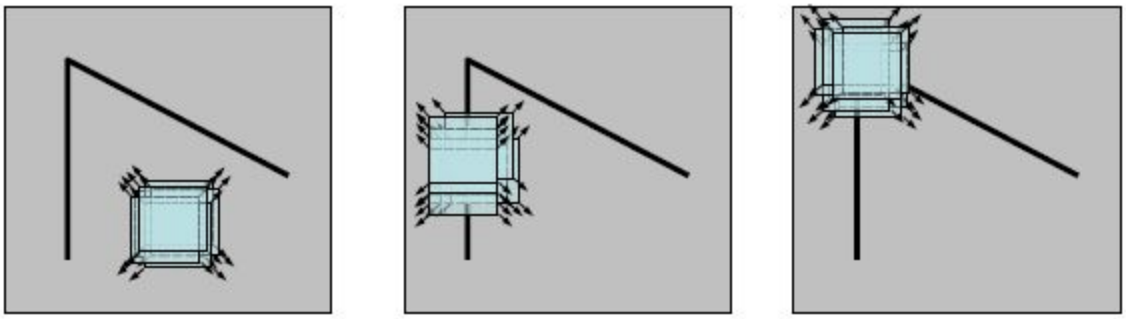
$$E(h) \simeq \sum w(x) \left(\nabla I(x) h\right)^2 dx = \sum w(x) \left(h^T \nabla I(x) \nabla I(x)^T h\right)$$

This last expression depends on the gradient of the image through the autocorrelation matrix, or structure tensor, which is given by

$$M = \sum w(x) \left(\nabla I(x) \nabla I(x)^T\right) = \begin{pmatrix} \sum w(x) I_x^2 & \sum w(x) I_x I_y \\ \sum w(x) I_x I_y & \sum w(x) I_y^2 \end{pmatrix}$$

Analyzing their values, we may find three possible situations:

1. Both eigenvalues are small, $\lambda 1 \approx \lambda 2 \approx 0$, then the region is likely to be a homogeneous region with intensity variations due to the presence of noise.
2. One of the eigenvalues is much larger than the other one, $\lambda 1 \gg \lambda 2 \approx 0$, then the region is likely to belong to an edge, with the largest eigenvalue corresponding to the edge orthogonal direction.
3. Both eigenvalues are large, $\lambda 1 > \lambda 2 \gg 0$, then the region is likely to contain large intensity variations in the two orthogonal directions, therefore corresponding to a corner-like structure.



"flat" region: no change in all directions

"edge": no change along the edge direction

"corner": significant change in all directions

In order to improve the stability of the response, we propose a simple scale space approach by checking that the corner is still present after a zoom out.

At each scale, we first zoom out the image by a factor of two and compute the Harris' corners. This is done in a recursive way, as many times as specified by the number of scales chosen (NScales). Then, we compute the corners at the current scale and check that they are present in both scales. The algorithm stops at the coarsest scale (NScales = 1).Note that the value of σi is also reduced by a factor of two at the coarse scales. This allows us to preserve the same area of integration. Also, we select the points at the finer scale for which there exists a corner at a distance less than σi. The corner position is divided by two inside the distance function.

Let's go into the details of various stages involved in this algorithm, they are:

1. **Smoothing the image**

   This stage is done by convolution with a Gaussian function and multiplying with the kernel matrix. This can be of two types, 'Stack integral images' (SII) method and 'Discrete Gaussian' method. We usually adopt the former method as it is faster, but though it is less accurate than the latter it is not of much importance in our case since we are dealing with basic shapes and not complex images.

   It is also important to ensure that the detector has a high repeatability rate, i.e., it must be rotationally invariant in detecting the interest points as the image is rotated from 0 to 180 degree, then we say that the repeatability rate is high. Experimentally, we can see that the precision is far better at 0,90, and 180 degrees. The repeatability rate also depends on the complexity of the image. For our image, a repeatability rate of more than 1.5 can be observed. Once the image is smoothened, we can move on to calculate the gradient of the image.

2. **Computing the gradient of the image**

   There are again two types of gradient masks we can adopt in order to compute the gradient. One, 'Central difference approach' and 'Sobel

operator'.

$\frac{1}{2}\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$ $\frac{1}{2}\begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$

(a) Central differences

$\frac{1}{8}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ $\frac{1}{8}\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

(b) Sobel operator

The Harris method commonly uses the former method and we can adopt the latter too in case of complex images. What the Sobel operator does is that it does some extra smoothening work, and thus more accurate. But in our case, we can simply proceed with the Central Difference approach since our images are simple and there is no much difference when the Sobel operator is used instead.

3. **Computing autocorrelation matrix**

This is the main step of the entire algorithm and all the interest points are determined keeping this matrix as the base and initial data. The products of the derivatives are calculated at each position and the coefficients of the matrix are assigned values accordingly.

$$E(u,v) \cong [u,v] \quad M \quad \begin{bmatrix} u \\ v \end{bmatrix} \qquad M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

4. **Computing corner strength**

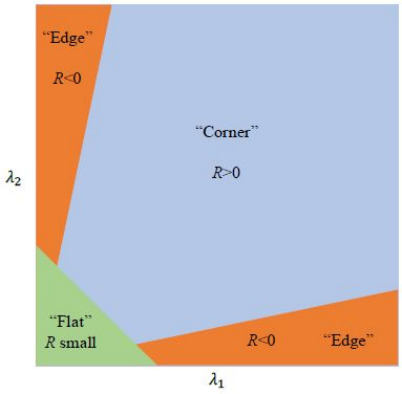
The autocorrelation matrix is symmetric and positive semidefinite, yielding two real non-negative eigenvalues. Analyzing these eigenvalues, we may define corner response functions that are invariant to in-plane rotations. We implement the following any of the following standard functions.

Harris and Stephens [9]	$R_H = \lambda_1\lambda_2 - \kappa \cdot (\lambda_1 + \lambda_2)^2$
Shi and Tomasi [21]	$R_{ST} = \begin{cases} \lambda_{min} & \text{if } \lambda_{min} > \tau \\ 0 & \text{Otherwise} \end{cases}$
Harmonic mean [3]	$R_{HM} = \frac{2\lambda_1\lambda_2}{\lambda_1+\lambda_2}$

We shall use the Harris Stephens method

$$R_H = \lambda_1\lambda_2 - \kappa \cdot (\lambda_1 + \lambda_2)^2 = det(M) - \kappa \cdot trace(M)^2$$

where κ is a value typically between 0.04 or 0.06. This function not only allows to calculate interest points but also to detect edges. Figure below depicts the regions of this detector in the λ1 – λ2 plane. When the value of R is negative, it means that one of the eigenvalues is much larger than the other one and the pixel is likely to belong to an edge. When R is positive and large, both eigenvalues are large, then it is likely to be a corner. If both eigenvalues are small, R is small and it is part of a flat region.



5. Non-maximum suppression

The purpose of the non-maximum suppression step is to find the best interest point in each local neighborhood. In several works, the maxima are obtained from the 3 × 3 neighbors around the feature. This means that two interest points can be separated by one pixel only. This can be an acceptable distance for low resolution images, however, this is not convenient for high resolution ones.

For this reason, we chose to implement a generic algorithm that extracts points using a radius 'r'. Intuitively, this radius must be related to the size of the integration kernel, σi, used in Step 3. A reasonable choice is to calculate the radius as r = 2σi. In this way, the features will be separated by the area of influence of the Gaussian function. Computing the maximum of an array only requires one comparison per pixel, but finding the local maxima is much more cumbersome. A brute force algorithm has a computational cost of O((2r + 1)2N), with N the number of pixels. Current methods require around two comparisons per pixel on average.

At the beginning, we take into account a threshold to skip low values in the corner strength function. This threshold depends on the level of noise in the image and on the function selected. The Harris measure and the

harmonic mean are similar but the threshold must be higher in the first function to approximately select the same number of points, as depicted in Figure 6. The threshold for the Shi-Tomasi function should be in general smaller than the previous ones. For each line, we first reject border points. The next peak is found and compared with the pixels on the right and then on the left. We use an array, skip, that allows us to jump positions that have been already visited and are smaller than a neighbor. It is interesting to prioritize the non-visited neighbors first so that the number of comparisons are reduced and new positions are marked as skippable. If the selected point is the maximum in its line, it is then compared with the 2D regions, of size (2r + 1) × r, below and above the current line.

6. **Selecting output corners**

Now as we have suppressed all the non-maximum corners, it is important to select the corners, or more precisely interest points we want, we can do this by either of the methods:

1. The simplest one is to select all the corners detected. In this case, the sorting is given by the non-maximum suppression algorithm, i.e. sorted by rows and then by columns.
2. All the corners are sorted by their corner strength values in descending order. This is interesting for applications that need to process the more discriminant features first.
3. Another alternative is to select a subset of the corners detected. The user specifies a number of corners to be found and the application returns the set with the highest discriminant values.
4. We may also select a set of corners equally distributed on the image, which is interesting for several applications. In that case, the user specifies a number of cells and the total number of points to be detected. The algorithm tries to find the same amount of points in each cell. It is possible that no distinctive points are detected in some cells, so, in general, the number of features will be smaller than the target number of points specified by the user.

7. **Calculating subpixel accuracy**

This is the final step in the process, let us see the images below

You can see that the corner does not lie on a single pixel. The corner is "spread out" (In fact, in real life situations, it's almost impossible to get corners to lie on exact pixels). So, with a corner detection algorithm like the Shi-Tomasi corner detector or the Harris corner detector, you will end up with a corner like (56, 120). But, scientists and other people want a corner like (56.768, 120.1432). This is subpixel accuracy. Hence, it becomes necessary to refine features with subpixel accuracy.

For this, we need to estimate an approximate function in a local neighborhood around each feature and then find its maximum. We can implement two interpolation methods based on quadratic and quartic polynomials, respectively. We observe that the results of the quartic interpolation are better for both repeatability rate values, but the difference is not meaningful. The quadratic approach provides a closed-form solution, is simpler to implement and faster. The runtime of the quadratic approximation is between two and three times faster than the quartic interpolation on average. Note, however, that the time spent in this step is very small in comparison with other steps in the algorithm.

3. Canny Edge Detection

Canny Edge detection, a popular edge detection algorithm developed by John F. Canny in 1986. It is a multi-stage algorithm and we will go through each stage. Before we start proceeding towards the steps, it's important to first convert the image into grayscale as this detector is based on the grayscale images.

1. Noise reduction

We get rid of the noise in the image by using a Gaussian filter/Gaussian Blur to smooth it. The extent of blur required can be adopted by choosing the kernel size as per our choice. Commonly, we use the 5*5 kernel.

Once the kernel matrix is generated, and multiplied with the image matrix, we get the blurred image, this is then passed onto the next step.

2. **Intensity Gradient Calculation**

In this step, we calculate the edge intensity and direction by calculating the gradient of the image using edge detection operators. After the image is smoothed, the gradient (I) w.r.t 'x' and 'y' are calculated and can be convolved with Sobel kernels (K) along respective gradients ('x' and 'y') (used as a mask for edge detection). Examples of the Sobel kernels are of the form:

$$K_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, K_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}.$$

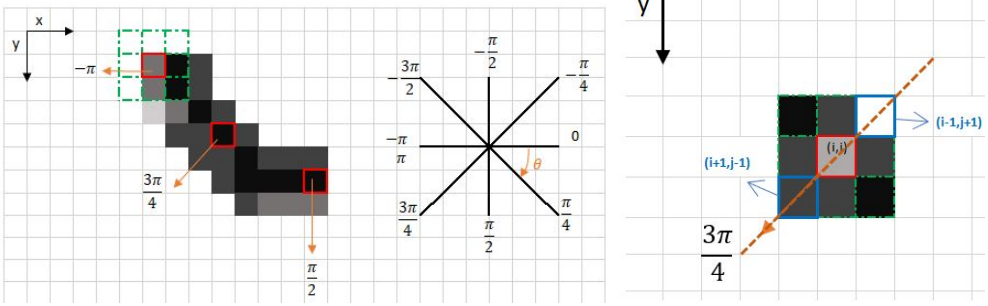
Thus, now the magnitude and slope of the gradient can be calculated as:

$$|G| = \sqrt{I_x^2 + I_y^2},$$
$$\theta(x, y) = arctan\left(\frac{I_y}{I_x}\right)$$

By the end of this step, we can see that some of the edges are thick and others are thin. Non-Max Suppression steps will help us mitigate the thick ones.

3. **Non-maximum suppression**

After getting gradient magnitude and direction, a full scan of the image is done to remove any unwanted pixels which may not constitute the edge. For this, at every pixel, the pixel is checked if it is a local maximum in its neighborhood in the direction of gradient. The purpose of the algorithm is to check the neighbourhood (in the same direction) of the pixel that is being processed. If any of the pixels around it appears to be of a higher intensity, than the one being processed or surrounding it, then this pixel is kept and the other one's intensity is accounted to 0. If there are no pixels in the edge direction having more intense values, then the value of the current pixel is kept.



In the figure given, the red checks indicate the pixel being processed, the orange line indicates the direction of gradient, and the dotted green checks depict the surrounding pixels. For example, In the second figure the direction is the orange dotted diagonal line. Therefore, the most intense pixel in this direction is the pixel ***(i-1, j+1).***

The result of this image is that we see thinner edges compared to the previous step's result

4. Double Threshold

This step aims at identifying strong, weak and non-relevant pixels in the image. The strong pixels alone contribute for the final edge while the other two don't. Thus, for this we define two thresholds, high threshold to identify the strong pixels and low threshold to identify some of the non-relevant pixels.

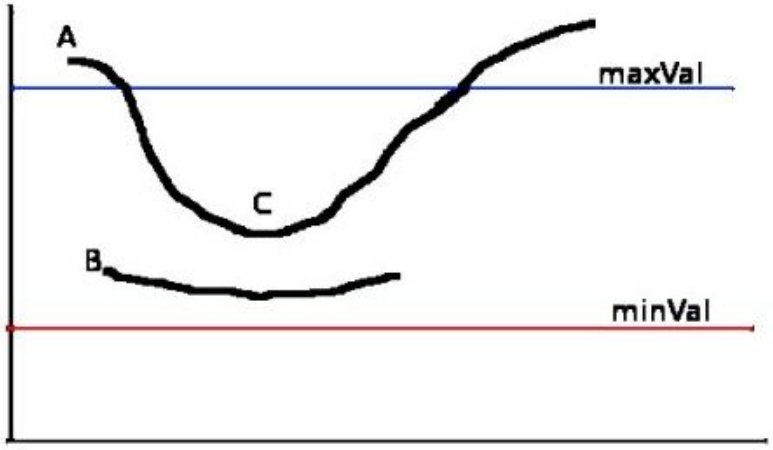
All the pixels within these thresholds are classified as weak pixels and the hysteresis mechanism (next step) will help us identify the strong and non-relevant pixels that lie above the high threshold.

The end result of this step is that the image now just consists of two types of pixels, i.e., weak and strong pixels.

5. Edge Tracking by Hysteresis

Based on the threshold results, the hysteresis consists of transforming weak pixels into strong ones, if and only if at least one of the pixels around the one being processed is a strong one.

The strong pixels are white in color while the weak ones are gray in color. Thus, when we are talking in terms of edges, it is the same algorithm applied on a large scale for many pixels. In the figure given below, The edge A is above the high threshold, so considered as "sure-edge". Although edge C is below a high threshold, it is connected to edge A, so that is also considered as a valid edge and we get that full curve. But edge B, although it is above minVal and is in the same region as that of edge C, it is not connected to any "sure-edge", so that is discarded. So it is very important that we have to select minVal and maxVal accordingly to get the correct result.



→ Code and logic/algorithm

We deal primarily with two codes here, the first one deals with detecting the corners of the hexagon of the bolt head/nuts, calculating the side lengths and inspecting them. But some bolt heads are circular in shape, hence we cannot use a corner detection here, hence an edge detection algorithm is necessary, and our second code caters this need. From that the diameter can be calculated and inspected.

CODE 1:

```
import cv2
import numpy as np

img = cv2.imread('hex.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gray = np.float32(gray)
dst = cv2.cornerHarris(gray,5,3,0.04)
ret, dst = cv2.threshold(dst,0.1*dst.max(),255,0)
dst = np.uint8(dst)
ret, labels, stats, centroids = cv2.connectedComponentsWithStats(dst)
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100,
0.001)
corners =
cv2.cornerSubPix(gray,np.float32(centroids),(5,5),(-1,-1),criteria)
for i in range(1, len(corners)):
    print(corners[i])
img[dst>0.1*dst.max()]=[0,0,255]
cv2.imshow('image', img)
cv2.waitKey(0)
cv2.destroyAllWindows
```

CODE 2 (FOR CIRCULAR NUTS):

The “pixels per metric” ratio:

In order to determine the size of an object in an image, we first need to perform a “calibration” (not to be confused with intrinsic/extrinsic calibration) using a reference object. Our reference object should have two important properties:

Property #1: We should know the dimensions of this object (in terms of width or height) in a measurable unit (such as millimeters, inches, etc.)

Property #2: We should be able to easily find this reference object in an image, either based on the placement of the object (such as the reference object always being placed in the top-left corner of an image) or via appearances (like being a distinctive color or shape, unique and different from all other objects in the image). In either case, our reference should should be uniquely identifiable in some manner

In this example, we'll be using the large washer as our reference object and, ensure it is always the left-most object in our image:

By guaranteeing the large washer is the left-most object, we can sort our object contours from left-to-right, grab the quarter (which will always be the first contour in the sorted list), and use it to define our pixels_per_metric, which we define as:
pixels_per_metric = object_width / know_width

```
# USAGE
# python object_size.py --image images/example_01.png --width 0.955
# python object_size.py --image images/example_02.png --width 0.955
# python object_size.py --image images/example_03.png --width 3.5

# import the necessary packages
from scipy.spatial import distance as dist
from imutils import perspective
from imutils import contours
import numpy as np
import argparse
import imutils
import cv2

def midpoint(ptA, ptB):
```

```
	return ((ptA[0] + ptB[0]) * 0.5, (ptA[1] + ptB[1]) * 0.5)

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True,
	help="path to the input image")
ap.add_argument("-w", "--width", type=float, required=True,
	help="width of the left-most object in the image (in inches)")
args = vars(ap.parse_args())


# load the image, convert it to grayscale, and blur it slightly
image = cv2.imread(args["image"])
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
gray = cv2.GaussianBlur(gray, (7, 7), 0)

# perform edge detection, then perform a dilation + erosion to
# close gaps in between object edges
edged = cv2.Canny(gray, 50, 100)
edged = cv2.dilate(edged, None, iterations=1)
edged = cv2.erode(edged, None, iterations=1)

# find contours in the edge map
cnts = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL,
	cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)

# sort the contours from left-to-right and initialize the
# 'pixels per metric' calibration variable
(cnts, _) = contours.sort_contours(cnts)
pixelsPerMetric = None

# loop over the contours individually
for c in cnts:
```

```
	# if the contour is not sufficiently large, ignore it
	if cv2.contourArea(c) < 100:
		continue

	# compute the rotated bounding box of the contour
	orig = image.copy()
	box = cv2.minAreaRect(c)
	box = cv2.cv.BoxPoints(box) if imutils.is_cv2() else
cv2.boxPoints(box)
	box = np.array(box, dtype="int")

	# order the points in the contour such that they appear
	# in top-left, top-right, bottom-right, and bottom-left
	# order, then draw the outline of the rotated bounding
	# box
	box = perspective.order_points(box)
	cv2.drawContours(orig, [box.astype("int")], -1, (0, 255, 0), 2)

	# loop over the original points and draw them
	for (x, y) in box:
		cv2.circle(orig, (int(x), int(y)), 5, (0, 0, 255), -1)

	# unpack the ordered bounding box, then compute the midpoint
	# between the top-left and top-right coordinates, followed by
	# the midpoint between bottom-left and bottom-right coordinates
	(tl, tr, br, bl) = box
	(tltrX, tltrY) = midpoint(tl, tr)
	(blbrX, blbrY) = midpoint(bl, br)

	# compute the midpoint between the top-left and top-right
points,
	# followed by the midpoint between the top-righ and bottom-right
	(tlblX, tlblY) = midpoint(tl, bl)
	(trbrX, trbrY) = midpoint(tr, br)
```

```
	# draw the midpoints on the image
	cv2.circle(orig, (int(tltrX), int(tltrY)), 5, (255, 0, 0), -1)
	cv2.circle(orig, (int(blbrX), int(blbrY)), 5, (255, 0, 0), -1)
	cv2.circle(orig, (int(tlblX), int(tlblY)), 5, (255, 0, 0), -1)
	cv2.circle(orig, (int(trbrX), int(trbrY)), 5, (255, 0, 0), -1)

	# draw lines between the midpoints
	cv2.line(orig, (int(tltrX), int(tltrY)), (int(blbrX),
int(blbrY)),
		(255, 0, 255), 2)
	cv2.line(orig, (int(tlblX), int(tlblY)), (int(trbrX),
int(trbrY)),
		(255, 0, 255), 2)

	# compute the Euclidean distance between the midpoints
	dA = dist.euclidean((tltrX, tltrY), (blbrX, blbrY))
	dB = dist.euclidean((tlblX, tlblY), (trbrX, trbrY))

	# if the pixels per metric has not been initialized, then
	# compute it as the ratio of pixels to supplied metric
	# (in this case, inches)
	if pixelsPerMetric is None:
		pixelsPerMetric = dB / args["width"]

	# compute the size of the object
	dimA = dA / pixelsPerMetric
	dimB = dB / pixelsPerMetric

	# draw the object sizes on the image
	cv2.putText(orig, "{:.1f}in".format(dimA),
		(int(tltrX - 15), int(tltrY - 10)),
cv2.FONT_HERSHEY_SIMPLEX,
		0.65, (255, 255, 255), 2)
```

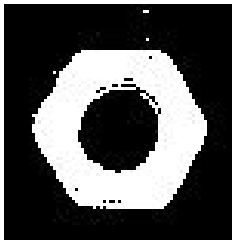
```
    cv2.putText(orig, "{:.1f}in".format(dimB),
        (int(trbrX + 10), int(trbrY)), cv2.FONT_HERSHEY_SIMPLEX,
        0.65, (255, 255, 255), 2)

    # show the output image
    cv2.imshow("Image", orig)
    cv2.waitKey(0)
```

→ Analysing the results:

Requirement:

Perfect lighting so that there is no noise in image further not affecting the accuracy of harris point detection.

INPUT



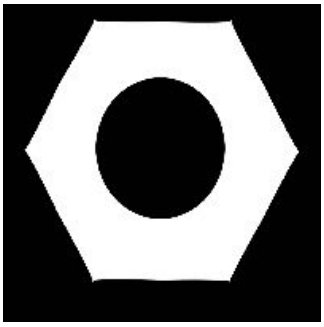
OUTPUT

Images like mentioned above will give poor results leading to lots of error.

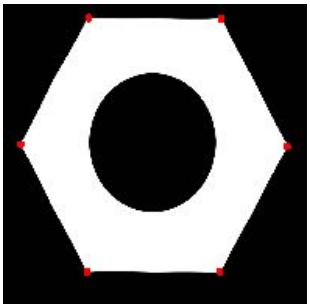
Code 1 results:

Nut 1:

Input:



Output:



Coordinates:

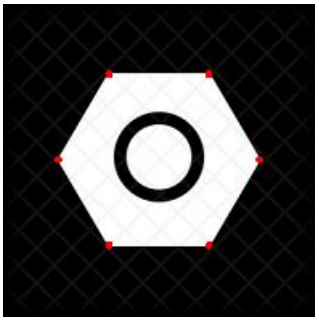
```
[57.209637 10.003809]
[142.9227   10.39489]
[13.293825 91.38222 ]
[185.64276  92.36891]
[ 56.216854 173.70592 ]
[141.89447 173.52042]
```

Nut 2:

Input:

Output:

Coordinates:



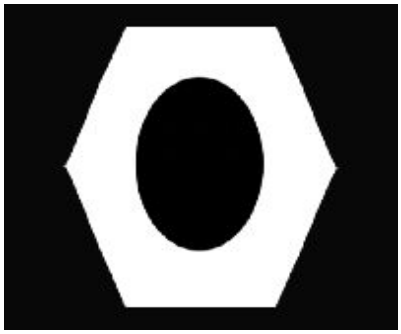
```
[68.201004 45.535267]
[130.87268   45.525833]
[36.27281 99.52803]
[162.71236  99.48125]
[ 68.20725 153.46288]
[130.91612 153.47049]
```

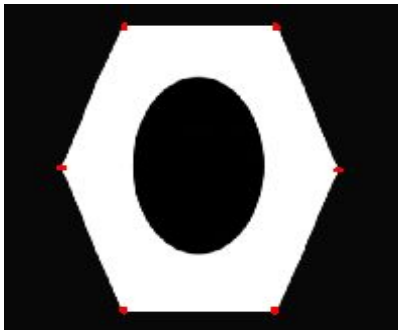
Nut 3:

Input:

Output:

Coordinates:





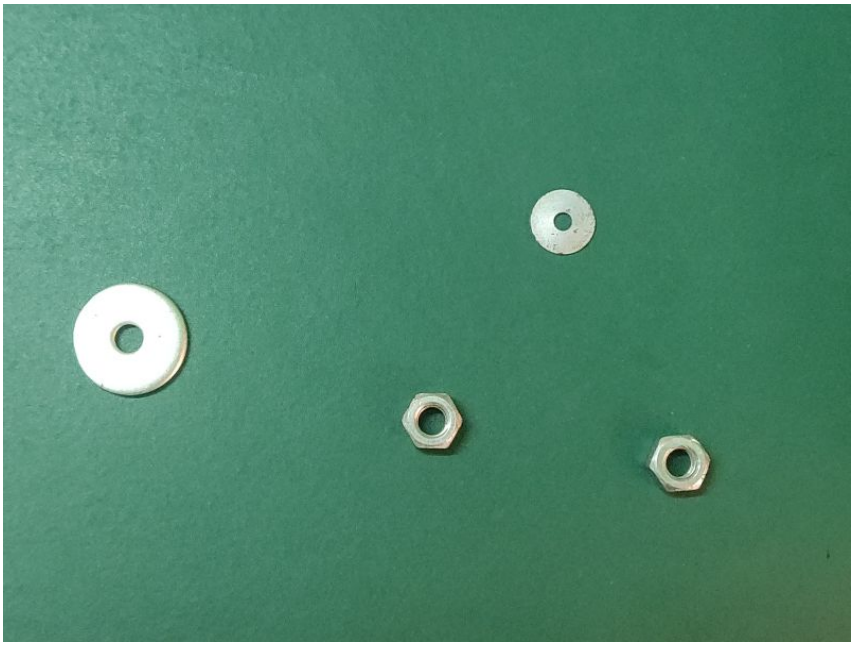
```
[61.641335 13.547419]
[137.53452  13.56637]
[30.310785 98.86359 ]
[168.53157 100.34501]
[ 61.3131  185.43108]
[137.19398 185.45607]
```

The Above is an analysis of a batch of bolts, and is approved or rejected based on comparison with the average allowed size taking tolerance into account.

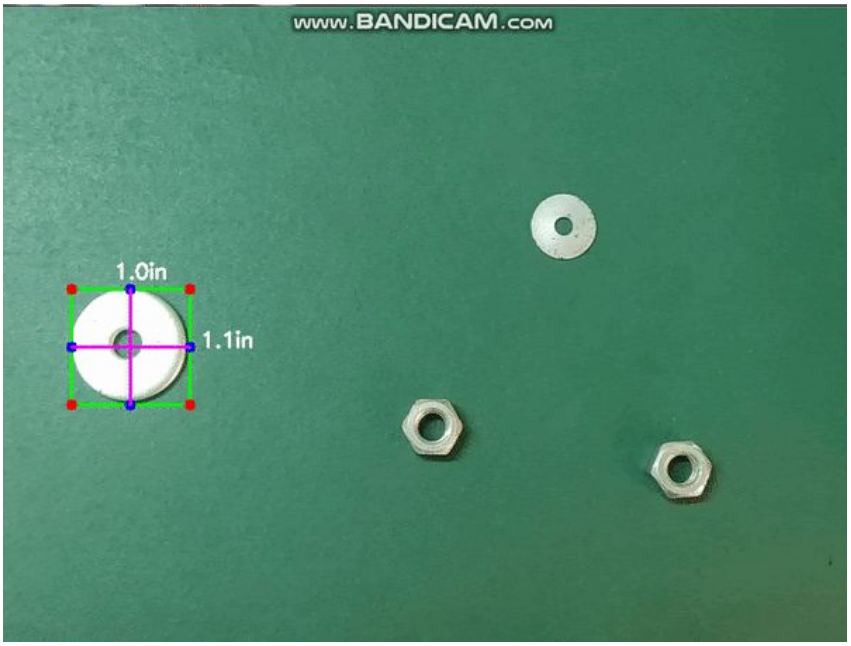
A sample of 6 are taken from a batch of a type of bolt heads and each of their lengths is computed. The ones that are within the approved range are in green while the ones that aren't are in red (in column 4). In some cases, it is not practical to measure every nut or bolt manufactured in a batch. Hence, we take the average of each of these sample batches and compare it with the required range of values keeping the tolerance in mind. If this sample average is well within the range, the batch is approved, while the ones not are rejected.

Code 2 results:

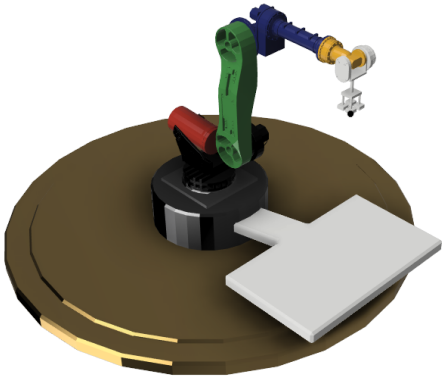
Input:



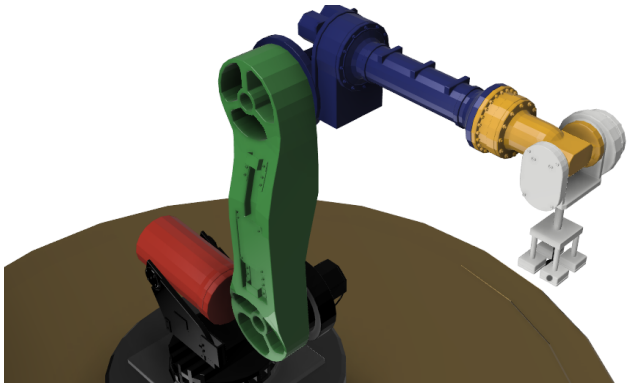
Output: Watch the gif



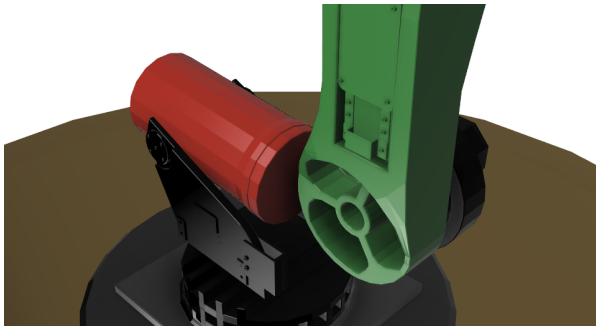
→ Our prototype



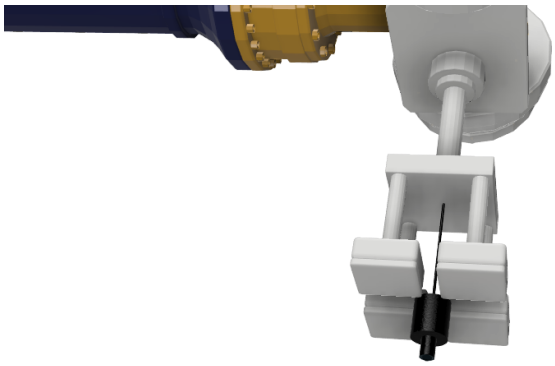
Perspective View



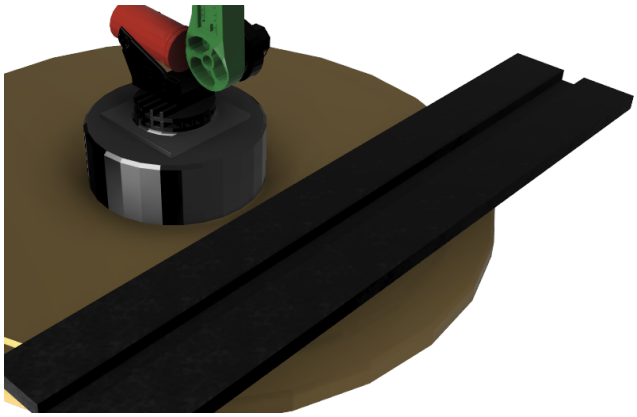
Movable arms at multiple locations

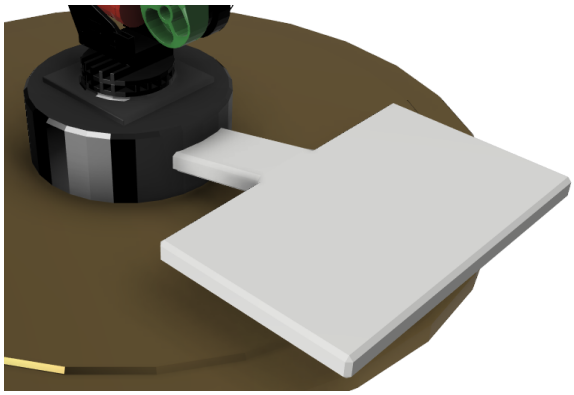


Counter balancing mass for precision



Stiff Camera holders





The Part can be inspected when moving in a belt (left image) as well as individually using a portable platform (right image).

→ Applications and feasibility

The applications in this area are varied as mentioned in some of the previous sections and let us precisely deal with some of the applications we can use the same for and account on its feasibility too.

1. Measure and compare the manufactured size of a bolt head with that of the actual size.
2. Inspection of parts whose shape is the same of any n-sided polygon.
3. Measurement of tiles of any fixtures of respective shape like squares and rectangles.
4. A slight modification in the code can be used for facial recognition
5. Any object/contour,in the image, can be found and the appropriate dimensions can be found.

Accounting for the feasibility of the above suggested method, it can be used directly in manufacturing plants. That is, when a part passes through a belt, a camera and a interface programmed with the same can be embedded along the belt at some point so that parts within the specified dimensions are taken to the next step and the rest can be rejected either through hand-picking or a seperate belt or any other mechanism.

→ Concluding remarks and Further extensions

This is just the base and pillar of inspecting the product and the process can be embedded into a mobile application such that the customers can use the same to inspect such similar products at home. Some further developments in the program can be such that non-regular shapes or custom shapes can be programmed into the code. This might help us in areas like inspecting parameters of gears (the tooth size, diameter of gear etc), finding the circularity and cylindricity of an object and many more. Hence, we can say that there exists a scope to expand this method into various domains on further work and developments.

We have seen the pros and cons of the existing methods as well as our methods. From some of the previous methods we see that there are problems with accuracy in one and space, cost etc in the other. So in simple words, we would expect a solution that is economical, uncompromising on quality, compact, quick and easy to operate (i.e., user-friendly). We can clearly conclude that our method satisfies all the above expectations, also yielding very accurate results when respective parameters are set according to the domain of application and the part being inspected.

→ References

1. An Analysis and Implementation of the Harris Corner Detector by Javier S´anchez, Nelson Monz´on, Agust´ın Salgado,
2. Rick Szeliski's lecture notes, CS courses, Paul G Allen School of Computer science and Engineering, Washington.
3. Patrices Lectures (Gaussian filters), School of Computer Science, The university of Auckland, NZ.
4. Stackexchange (Computer Graphics)
5. Stackexchange (DSP)
6. Towards Data Science
7. Open-CV python tutorials
8. Computer vision courses