**REPORT ON DATA MATRIX DETECTION**

# 1.1 Abstract:

A Data Matrix is a two-dimensional code consisting of black and white cells or dots arranged in either a square or rectangular pattern, also known as a matrix. A 2-D bar code is capable of encoding very large amounts of data within a very small area, it also has a high rate of error correction, and it is very safe. This enables it to be used in a variety of different applications. The report presents an algorithm that identifies a data matrix code based on the “L shaped” finder pattern. In this case, data matrix codes of a size of about ~ (200x200) pixels, are printed in black on a transparent plastic roll. The detected contours coordinates are used to slice the image and it is passed into the open-source python library libdmtx. Finally, a test data containing 800 images is then used to evaluate the suggested method, which yielded impressive results with an accuracy greater than 86%. Further scope for the development of algorithm along with the extreme cases is discussed.

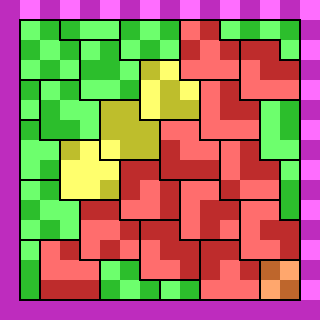
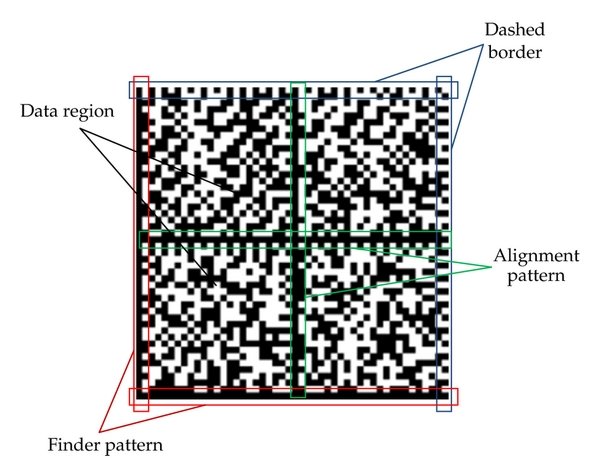
# 1.2 Task Objective:

A plastic roll (web) is fed into a roll-to-roll laser machine, where it may contain defective regions. When there are defective regions on the web, the location information for them will already be provided. The task of manually identifying defective areas is tedious and error prone. Data matrices are printed on the Web at regular intervals and include the information about their location on the web. The primary objective of this task is to scan the data matrix codes and learn about the location so that when the web is fed into the roll-to-roll laser system it is possible to identify the defective regions, and lasering the defective region of web can be prevented. Therefore, additional machine features like monitoring and dynamic processing could be added.

# 1.3 Introduction:

Data Matrix is a two-dimensional code consisting of black and white dots or cells arranged in either a square or rectangular pattern. As shown in figure 1.3.1, it consists of two solid adjacent borders in the shape of an "L" (known as the "Finder Pattern"), and two alternate dark-light borders (known as the "timing patterns"). Finder patterns help locate the data matrix and find its orientation while timing patterns count the number of rows and columns in the matrix. An example of a Data Matrix code, colored to show data (green), padding (yellow), error correction (red), finder and timing (magenta) and unused (orange)

Figure 1.3.1 Data Matrix



# 1.4 Literature Review

Qiang Huang et al. in [1] presented the Finder Pattern detection method that is mainly based on the line segment detection (‘L’) when it is distinguishable and continuous. However, it is challenging to find this approach helpful to detect matrix codes that do not have a continuous line on its finder pattern. Chutatape, O. et al. in [2] discussed a method using Hough lines to detect the Horizontal and vertical lines in an image and thereby finding the matrix code within the image. However, it is computationally expensive to rely on Hough lines alone to detect the compact-size data matrix codes. Dai, Y.G., Liu, et al. in [3] discusses the detection of highly compact-sized data matrix code on a metal surface using a digital image processing technique where binarization is achieved by improving the traditional Otsu algorithm. They used the Hough transformation to detect the four vertices of the Data Matrix code, which makes it long to recognize the matrix code and the matrix codes are well defined. D.K. Hansen et al. [4] demonstrated a method to detect the data matrix by training a deep learning neural network. They described a way to adopt the state-of-art deep learning-based detector YOLO (You Only Look Once) to detect barcodes in a fast and reliable way. However, training a deep learning neural network would require a lot of data. That is a fascinating perspective to investigate, however, the accuracy provided in the paper is not so convincing to choose this method and it requires huge amounts of data to train. Ladislav Karrach et al. in [5] compared the impact of various cameras on the detection and decoding of data matrix codes in real scene images by locating the finder pattern using adaptive thresholding techniques and connecting neighboring points to continuous regions. They proceeded onto finding the right angles isosceles triangle (that has the ‘finder patter’ as equal sides) which is not ideal in this case as the finder pattern sometimes is distorted or mis printed. Ion-Cosmin Dita et al. [6] provided a method to localize the modules of Industrial Data Matrix Code (IDMC) marked on curved surfaces, where an imaginary grid that has the same orientation of the IMDC is projected onto the image and accordingly the grid modules of the code are scanned. If the surface is not a sphere the results are not accurate with this method.

# 1.5 Code Development

## 1.5.1 Image Preprocessing

The code development is focused on detecting the "L"-shaped solid adjacent borders (finder pattern). This is accomplished using different image processing steps. The first step is to read the image.



The image is then converted to grayscale to reduce the number of channels. A 3x3 kernel (as shown in the following code) is applied to the gray image, giving us gradX (figure:1.1.2) where only horizontal lines are detected. The transpose of this kernel is passed again over the grayscale image. This gives us gradY (figure:1.1.4), where only vertical lines are seen. Gradient image (Fig:1.1.4) is obtained as we subtract gradY from gradX. This helps us in obtaining the objects that contain both horizontal and vertical lines in them and get rid of the rest of the objects that are circular, rhomboid, etc.



Fig: 1.5.2 gradX

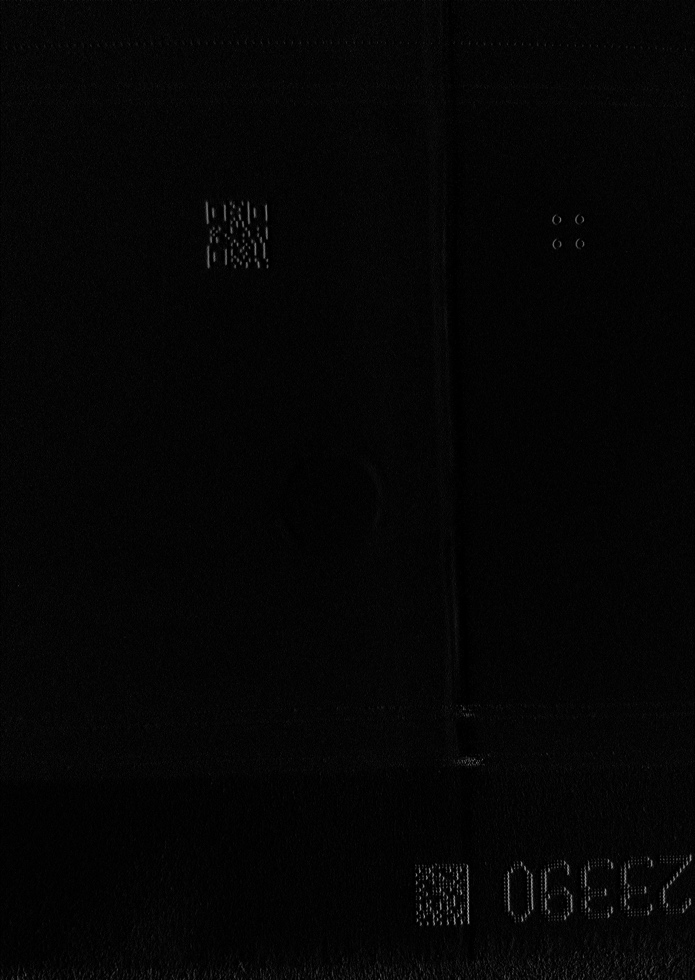
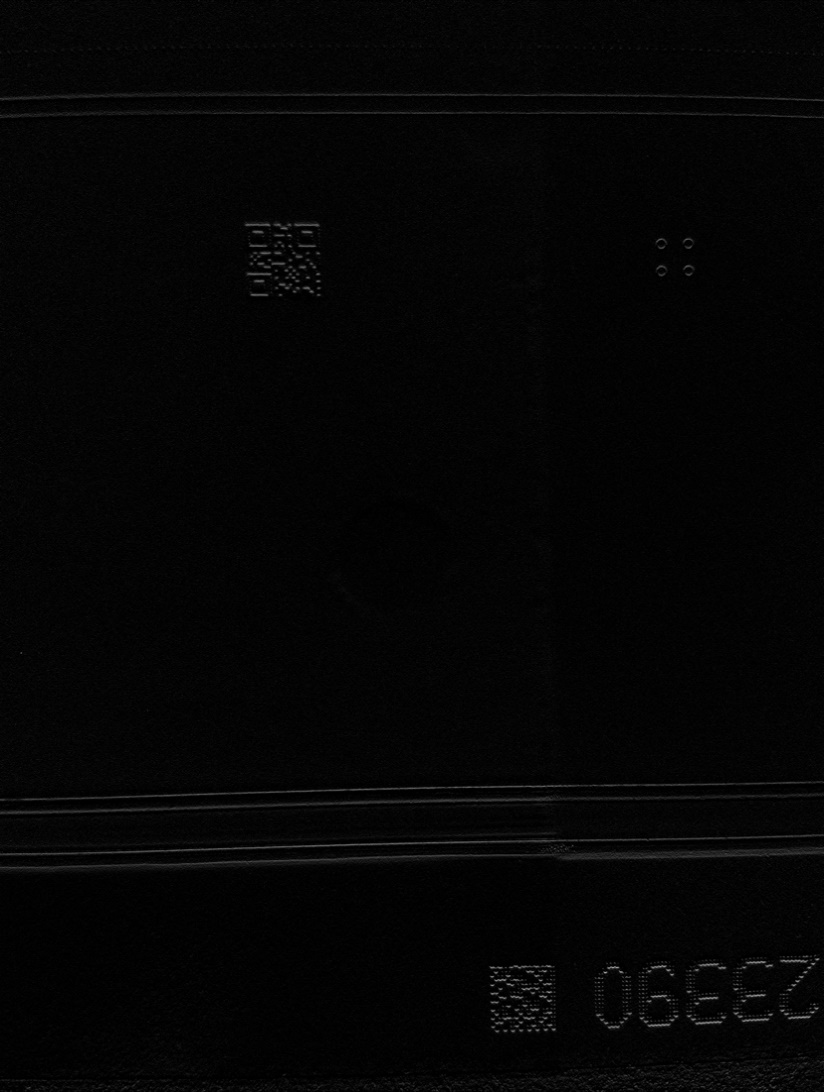


Fig: 1.5.3 gradY

Fig: 1.5.1 original Image

## 1.5.2 Closing operation

The gradient image is then smoothened by using gaussian blur with a filter kernel size of 7x11. The smoothened image is then converted into black and white by using Otsu Thresholding technique. The images are often captured in well exposed condition and hence global thresholding technique like OTSU is helpful in removing additional noise. Then, morphological closing operation is performed on the thresholded image to cover the small gaps if they present in the detected contours. Later, the closed image is eroded and dilated in order to further reduce the gaps between within the matrix code and connect different broken pieces of the contour. This process can be performed more than once to produce better results, but the results would not vary by much. To keep the model cheap, we perform it only once. As seen in the figure 1.1.6 the data matrix contour is prominently visible, and easier to be detected than any of the previous steps.



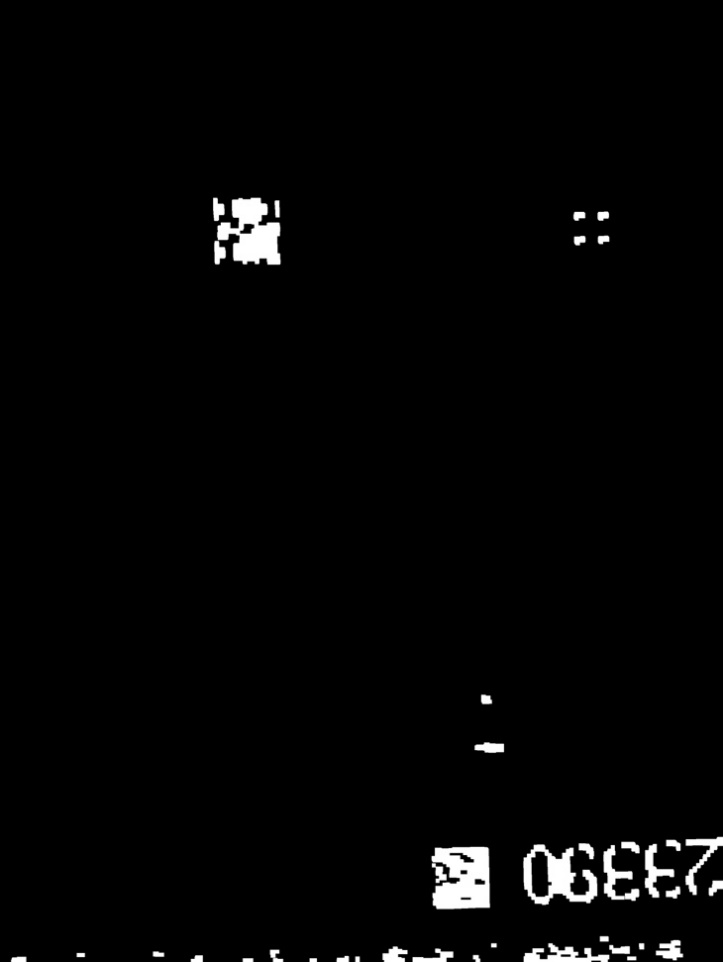
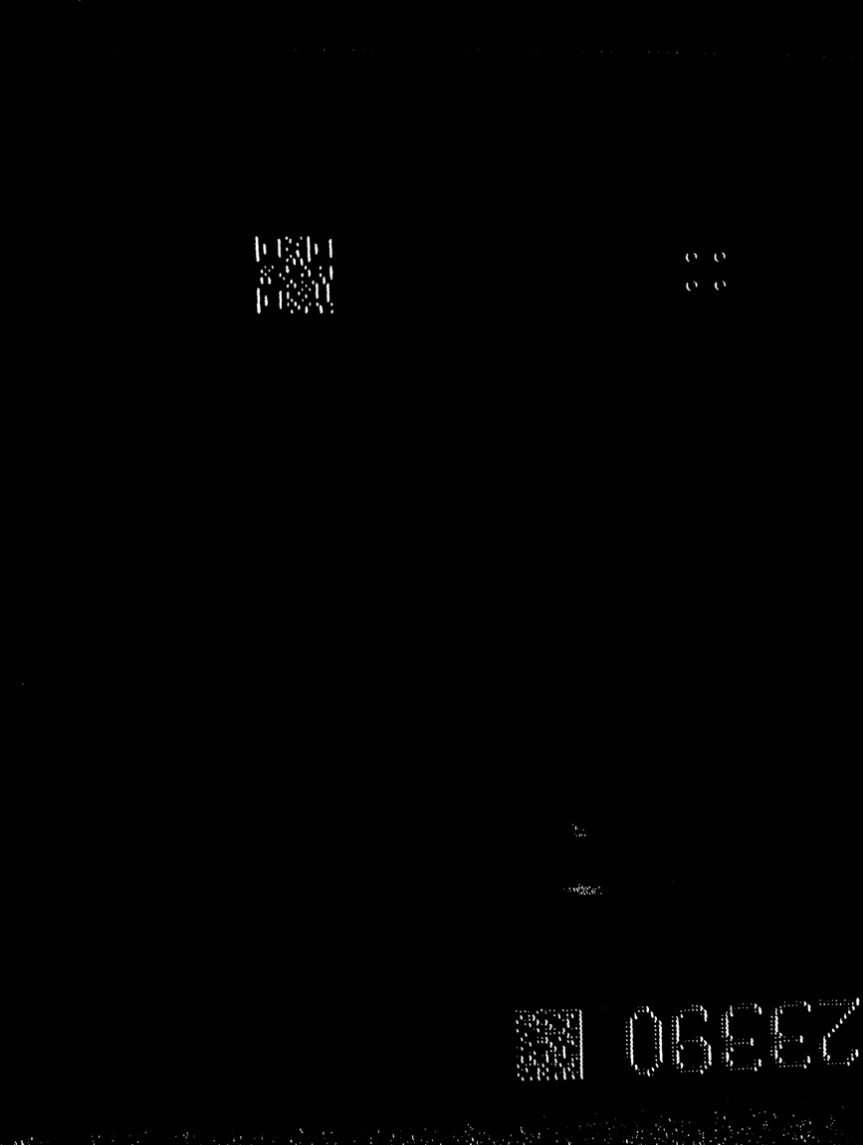


Figure 1.5.6 Closed Image

Figure 1.5.5 Thresholded gradient

Figure 1.5.4 Gradient Image

## 1.5.3 Finding Contours and selecting Region of Interest

Contours are found in the closed image. The height and width of the matrix code is already a known variable which is about 200 pixels. Therefore, the detected contours which obey this size+/- 50 pixels are selected. After selecting the required contour, a bounding rectangle for the contour is considered. To prevent any loss of code, an additional border of about 50 pixels is added on each edge of the rectangle, which gives us a new bounding box.



Using the new bounding box coordinates we slice the original image to obtain the region of interest (ROI) as shown in fig: 1.1.8. ROI is a sliced original 3-dimensional image. It is then converted into grayscale, then it is thresholded using the OTSU thresholding technique, it is then eroded and dilated to connect the separate dots in the threshold into continuous lines and we obtain closed ROI as shown in figure 1.1.9.



Finally, we pass the closed ROI into the pylibdmtx decode function where we obtain the data scanned.

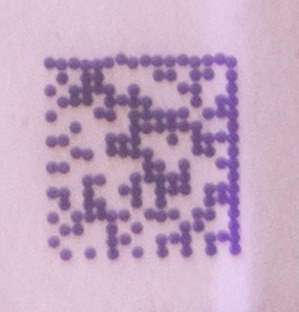
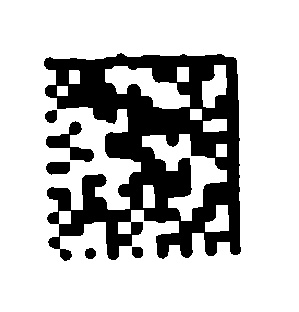


Figure 1.1.8 ROI

Figure 1.1.7 Detected Contour

Figure 1.1.9 Closed ROI

Upon successful detection, the code is returned and any other case the model continuous to search for the data matrix in the image and feed it to the decode function. The code is then sent to HTTP API through a network call.

# 1.6. Metrics

To evaluate the performance of the model, a sample test data that contains 803 images are taken.

The model is run on the test data, for which the following results are observed.

Total number of sample images: 803

Number of files where data matrix is detected and decoded successfully: 734

The number of files where data matrix is not detected is: 65

Number of files where data is not present: 4

True positives = 734 (85.95%)

False positives = 0 (0,00%)

True negatives = 51 (5.97%)

False negatives = 69 (8.08%)

Chart, bar chart

Description automatically generated

Figure 1.2.2 Heat Map for confusion matrix

Chart, treemap chart

Description automatically generated

Figure 1.2.1 Bar Plot

From the Confusion matrix,

* Out of 803 images, the (734) or ~86% of the instances represent True positives, the successful detection, and the decoding of the data matrix codes.
* Out of the remaining 120 instances, 51 represents inaccurate detections like QR-codes or other contours that don’t contain Matrix codes.
* 69 instances are matrix code detections, but they could not be decoded for various reasons like matrix code distortion or noise over or around the matrix code.

The images also contain similar sized QR-codes along with the data matrices, and these come in the way of detection as QR-codes also contain the ‘L’ shape within them. QR-code does not return anything when passed into the decoding function Pylibdmtx. Therefore, as far as data matrix code is detected and decoded in an image QR-code does not impose any threat. However, the major issue of concern is when the data matrix is detected but not decoded.

# 1.7. Evaluation

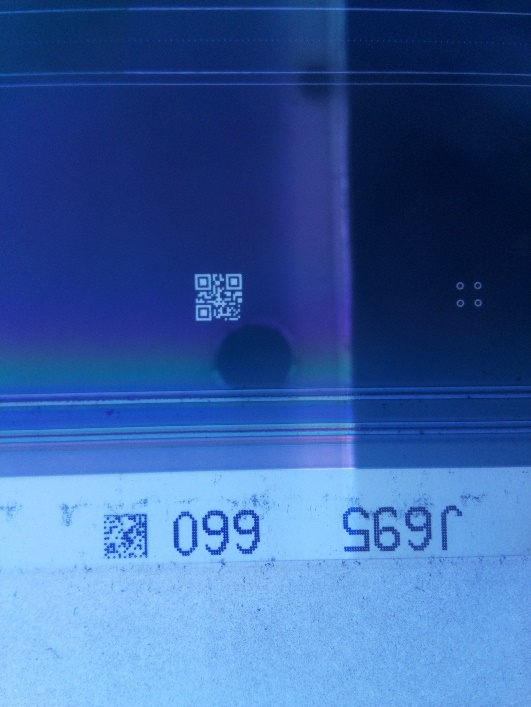
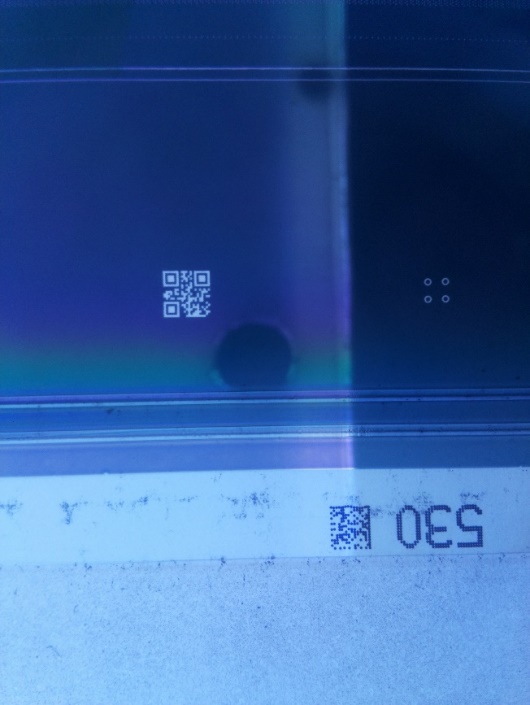
Successful detection and decoding of the matric code could be observed when the code is visible, or have little to no noise, distortion or the code is not destroyed. The following images from figure 1.3.1 are some examples where there is a successful extraction of the code.

Figure 1.3.1 Successfully detected images



The case of interest is where the matrix codes are detected but not decoded. The following images show the instances where the code is not successfully extracted.

Figure 1.3.2 Unsuccessful detection



## 1.7.1 Reasons for unsuccessful detection

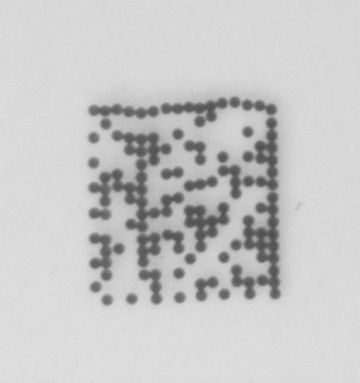
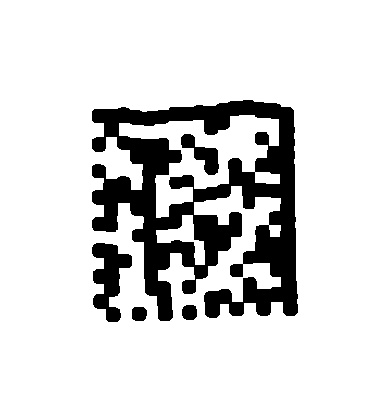
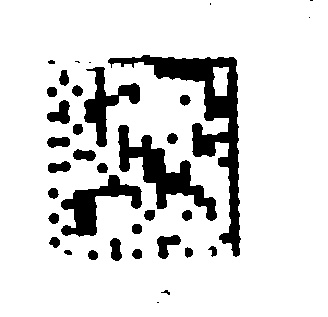
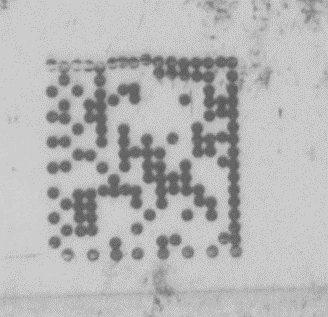
Among several reasons for a matrix code to not be detected, the major problems are

1. Presence of noise around or over the matrix code like the tape, additional gray pixels
2. Not properly printed or a broken matrix code.
3. Distortion in the matrix code
4. Varied Lighting conditions

The following images (figure 1.3.3 and figure 1.3.4) demonstrate the cases where the model finds it difficult to decode the matrix code.

Figure 1.3.4 improperly printed matrix code

Figure 1.3.3 Distorted matrix code



It can be clearly seen that in the presence of noise the global thresholding seems not very effective. The model needs a clever thresholding technique to achieve better results. Local thresholding techniques haven’t provided

**Further improvements:**

A different approach for preprocessing the ROI made the instances where the code extraction was unsuccessful proved to be successful. However, it does not work very well with other images where the code was successfully detected.



There is still a large scope of improvement.

1. One of the major things is improving the thresholding techniques to extract the code removing any extra noise to be able to use in any lighting conditions.
2. Improving the data matrix detection by refining the image preprocessing techniques.
3. As the web is a transparent sheet, there is a need to use a white background it is difficult to obtain images every time without any distortion and additional noise as there is a need to have the web roll over an additional white background before capturing it.

# 1.8. References

1. Huang, Q.; Chen, W.S.; Huang, X.Y.; Zhu, Y.Y. Data matrix code location based on finder pattern detection and bar code border fitting. Math. Probl. Eng. 2012, 2012, 515296, doi:10.1155/2012/515296.
2. Chutatape, O., Guo, L.F.: A modified Hough transform for line detection and its performance. J. Pattern Recognit.32, 181–192 (1999).

<https://www.researchgate.net/deref/https%3A%2F%2Fdoi.org%2F10.1016%2FS0031-3203(98)00140-X>

1. Dai, Y.G., Liu, L.Z., Song, W., Chao, D., Zhao, X.L.: The realization of identification method for Data Matrix code. In: 2017 International Conference on Progress in Informatics and Computing, pp.410–414. IEEEPress, Nanjing(2017). <https://www.researchgate.net/deref/https%3A%2F%2Fdoi.org%2F10.1109%2FPIC.2017.8359582>
2. D.K. Hansen and K. Nasrollahi. “Real-Time Barcode Detection and Classification Using Deep Learning”, in Proc. of 9th International Joint Conference on Computational Intelligence, vol.1,2017, pp.321-327

<https://vbn.aau.dk/en/publications/real-time-barcode-detection-and-classification-using-deep-learning>

1. Ladislav Karachi, Elena Pivarciova, Y. Nikitin “Comparing the impact of different cameras and image resolution to recognize the data matrix codes.” <https://www.researchgate.net/publication/327863572_Comparing_the_impact_of_different_cameras_and_image_resolution_to_recognize_the_data_matrix_codes>
2. Ion-Cosmin Dita, Marius Otesteanu, Franz Quint, Nagaraju Annamdevula “A scanning method for industrial data matrix codes marked on spherical surfaces” <https://www.researchgate.net/publication/261860699_A_scanning_method_for_industrial_data_matrix_codes_marked_on_spherical_surfaces>