

# **IE423 Quality Engineering**

## **Project Part-3 Report**



**Instructor:** Prof. Mustafa Gökçe Baydoğan

### **Group-11 Members:**

Melis Tuncer	2019402156
Batuhan Özkan	2018402033
Kaan Ertan	2019402087

## 1. Introduction

Linen is a textile material produced from the fibers of the flax plant. It is a rather tiresome material to manufacture and weave, but the resulting fiber is highly absorbent. Because of its common source of flax and its usefulness, it has been in human use since the age of the cavemen. Linen garments are today valued for their exceptional freshness, especially in hot conditions. It is known for its durability, breathability, and natural brightness.

In manufacturing, monitoring the processing of linen is important for several reasons. The most important one is quality assurance. Linen is a high-end fabric; therefore, maintaining high-quality standards is crucial. Monitoring the processing helps to ensure that the final product meets the desired quality and performance characteristics. The customers of linen products also expect a certain level of quality, which is above the average expectation for common textile materials. Monitoring the manufacturing process helps prevent defects in the final product, ensuring customer satisfaction.

Another benefit of monitoring the processing of linen is ensuring a higher standard of efficiency. Efficient manufacturing processes contribute to the cost-effectiveness and overall productivity of the process. Monitoring helps identify any inefficiencies or defects that may hinder the production process.

To move on with the motivations regarding the use of images and the identification of defects in linen manufacturing, the following motivations come forward:

Precision: Images provide a detailed representation of the linen material. Analyzing images allows for the identification of subtle defects or irregularities that would have been missed by the eyes of a human controller. By utilizing image pattern recognition, hidden defect patterns can be uncovered and reported to be resolved.

Visual Inspection Efficiency: Automated visual inspection using images is much more efficient and accurate than manual inspection. It can handle an incomparably larger volume of images quickly and consistently, reducing the defects due to human error.

Data-based Decision Making: Image processing techniques allow for automation and also automate decision making when it comes to determining what is defective and what is natural variation. This data can also be used for statistical analysis of defect causes.

Consistency Across Production: Automated image analysis provides a standardized approach to defect detection and correction. Therefore, it ensures consistency across different production parties.

## **2. Background Information: What Has Been Done in the Literature Regarding Process Monitoring on Linen?**

There are various approaches to the issue of linen process monitoring, depending on the specific requirements of the tasks. For instance, the Radio-Frequency Identification (RFID) method is found to be effective in process monitoring, according to research by the UTAR University in Malaysia. In this method, the incoming and outgoing linen (either new or used) to/from a hospital is monitored for its quality and possible new defects. (Xiang, 2023) This is a rather critical case where the linen quality doesn't only affect aesthetics, but also human health. With this approach, the problem can be solved in a reliable and automated manner.

Another approach is the Digital Image Correlation (DIC) method, which can be used to chart and evaluate specification distribution maps of fabrics. An example study was conducted at the Lodz Institute of Technology/Poland. (Baghernezhad et al., 2016) In the study, the researchers utilized DIC method to detect and analyse mass irregularities of yarn on the strain monitoring, during a tensile test. Their test succeeded as the DIC algorithm successfully monitored the process, much more successfully than its human counterparts.

Although these are very valuable approaches, the best approach, in our opinion, is the 2 dimensional autoregressive model for texture analysis and synthesis (Vaishali et al., 2014). With this approach, a very accurate and implementable framework for linen process monitoring is introduced with the use of an automated model. In this model, the linen texture is examined, and an autoregressive model is built accordingly. The independent variables for a single pixel in the image are chosen as the other pixels that surround the pixel to be predicted. With this approach, we were able to find both vertical (column-wise) and horizontal (row-wise) relationships in the texture. Then, any defects in the linen are evaluated and pointed out by the model. We benefitted from this approach greatly in our own study.

## **3. Approach: Explain Your Approaches to this Problem**

We first fitted a distribution to our image's all pixel values using a histogram. We found that the normal distribution reflects the histogram. The crucial assumption here is that the columns and rows are distributed with the same parameters, so that we could flatten the pixel matrix and construct the histogram and a distribution fitting all pixel values. We then calculated the mean and standard deviation of the pixel values. With these information at hand, we detected the pixels that were out of 0.001 probability limits (Figure-1). We saw both the number of pixels out of limits and could see them individually. After this, we replaced these instances with pixels valued 0 (black) and printed the resulting image (Figure-2).

Following these, we applied the same method to 100 windows of size 51x51 of our image. While doing this, we made sure that our windows also had a normal distribution by plotting histograms and checking the distribution. When we compare the output images of the two methods (for the output of the second method, see Figure-3), we see that the number of defects (outlier points) has reduced from 136 to 53, out of 262,144 pixels. When we crosschecked these points, we saw that 37 were shared in both images. This showed us that working locally in smaller windows is a good approach in order to ensure that local variability isn't averaged out when working with such a big data size. Also, having nearly two-thirds of the points in the second image also present in the first one is a sign of the consistency of the procedure. However, it is important not to forget that these two approaches are consistent in their own context, and we need to try other approaches since these may not actually be suitable for our 2D case.

In our second approach, we took each row as a sample with a sample size of 512 to construct an X-bar chart and an S chart (since  $n > 10$ , the S chart is more appropriate than the R chart) for each row, with 3 sigma limits. Again, we found the statistics (mean and variance) and outliers, then plotted the charts (S Chart:Figure-4 ; X-Bar Chart:Figure-5). As a result, the S chart suggested that 116 samples (rows) were out of control, whereas the X-bar chart suggested 156. When we examined the outcomes, we saw that the X-bar and S-charts are not appropriate for defect detection in linen images. If we also were to discard out-of-control points in the X-bar chart after discarding the ones in the S chart, we would lose more than half of all the pixels, which does not make sense for our purpose. In addition, we clearly see a downward shift in the X-bar chart, which indicates that the image is getting darker towards the bottom.

Because of these issues, we reevaluated this case. We kept our normality assumption for the rows and found the points (pixels) out of 0.001 probability limits. Then, this time, we discarded (equated to 0) only those points instead of discarding the whole row. After all, we reconstructed the image and counted the number of out-of-control points. Resultingly, we found only 90 defectives, proving that this is a more adequate technique that yields closer results than our initial method.

We moved on with defect detection of the same nature, but column-by-column instead of row-by-row. After the same operations, the result was only 55 defects overall. This approach and the previous row-by-row approach had 19 outliers in common, whereas all of our approaches so far had 12 common outlier points in total. As a result, we concluded that for certain linen patterns that have a strong correlation between the pixels within rows or columns, the last two methods may be preferred since they focus on the variation within the respective rows/columns. They are also conservative when it comes to declaring a point out of control, so if a user seeks a low level-1 error rate, these approaches are suitable.

## **Our Proposal of a Strategy for Defect Detection:**

The approaches that we have used so far have investigated the image in only 1D. On the other hand, linen may, and most probably does, have a 2D pattern that requires a 2D detection strategy. Because of this, we developed approaches that consider 2D. We conducted research for appropriate methods and found a paper that suggests making a 2D autoregression for linen texture check (Vaishali et al., 2014). Getting guidance from the paper, we made predictions for each pixel with the help of the pixels in a certain vicinity (neighborhood). We first tried taking 3x3 windows that contained 9 pixels in total. Then we set the target value as the pixel value that is in the middle of the 3x3 window so that we can predict it (output image: Figure-6). During the operation, we had to discard the first and last rows and columns since they didn't have 8 pixels surrounding them. After this, we checked the constant variance assumption of the residuals (which held) and plotted the 3-sigma control chart for residuals (Figure-7). It turned out that there was an unmanageable amount of outliers, which made us go on with 4-sigma limits instead. The outliers in the control chart for residuals mean there is a big gap between the predicted pixel value and the actual pixel value, which also means there is a problem/defect in that pixel. (Control chart: Figure-8 , Image: Figure-9). This approach yielded 189 o.o.c. points.

Afterward, we decided to also try this method with a 5x5 window size. This size yielded a better R-squared value so we moved on with this size. Again, 3 sigma limits yielded too many o.o.c. points so we utilized 4 sigma limits (Image: Figure-10 , Control chart: Figure-11). This approach yielded 165 o.o.c. points.

The first model which used 3x3 windows, had an R-squared value of 0.762 and found 189 defective pixels. The second model, which uses 5x5 windows, had an R-squared value of 0.805 and found 165 defective pixels. By comparison, increasing the window size yielded a better-fit model for middle pixel prediction. We would also increase the window size more, but that would also increase the computational complexity. Therefore, we propose using the second model to find defective pixels in a linen texture.

## **4. Results: Provide Your Results, Comparison and Discussion**

A general solution to finding defectives may be to apply our first 4 approaches (1:check all pixels at once, 2:check 51x51 windows separately, 3:check row-by-row, 4:check column-by-column) at the same time and find the common out-of-control points by taking the intersection of the out of control points of these 4 approaches. However, depending on the pattern, this approach may miss out-of-control points that are important in any approach. For

example, if there is any considerable defect in one of the 51x51 windows in the picture that could not be found using any other approach, it will not be included in the intersection of o.o.c. points. As mentioned in Part-3: Approach, these methods yielded results of their own and didn't contradict one another. However, even though some methods like checking 51x51 windows yielded very small numbers of outliers, it doesn't mean that they are useful and accurate for our purpose. We discussed that for certain linen patterns which have strong correlation between the pixels within rows or columns, column-by-column or row-by-row methods may be preferred since they focus on the variation within respective rows/columns. However, for a general strategy, these are not useful. Moreover, all these approaches investigate the image in only 1D. On the other hand, linen may, and most probably does, have a 2D pattern that requires a 2D detection strategy. Because of this, we focused on new approaches that consider 2D. We found the aforementioned paper (Vaishali et al., 2014) that focuses on 2D autoregression for textile texture check, and examined it to utilize parts of it for our framework.

We developed a suitable method as discussed above in Our Proposal section, and seeing that it is a useful method, we evaluated it on alternative images (images 48, 65, 30, 16, 98 were selected, the following number of defects were found in these images: 214, 538, 959, 497, 132). This method is a much more realistic and generalizable one since it evaluates the linen texture and defects on a two-dimensional basis and generates its predictions in that sense. Thanks to this logic, the model's control boundaries aren't only bound to variations in a single dimension.

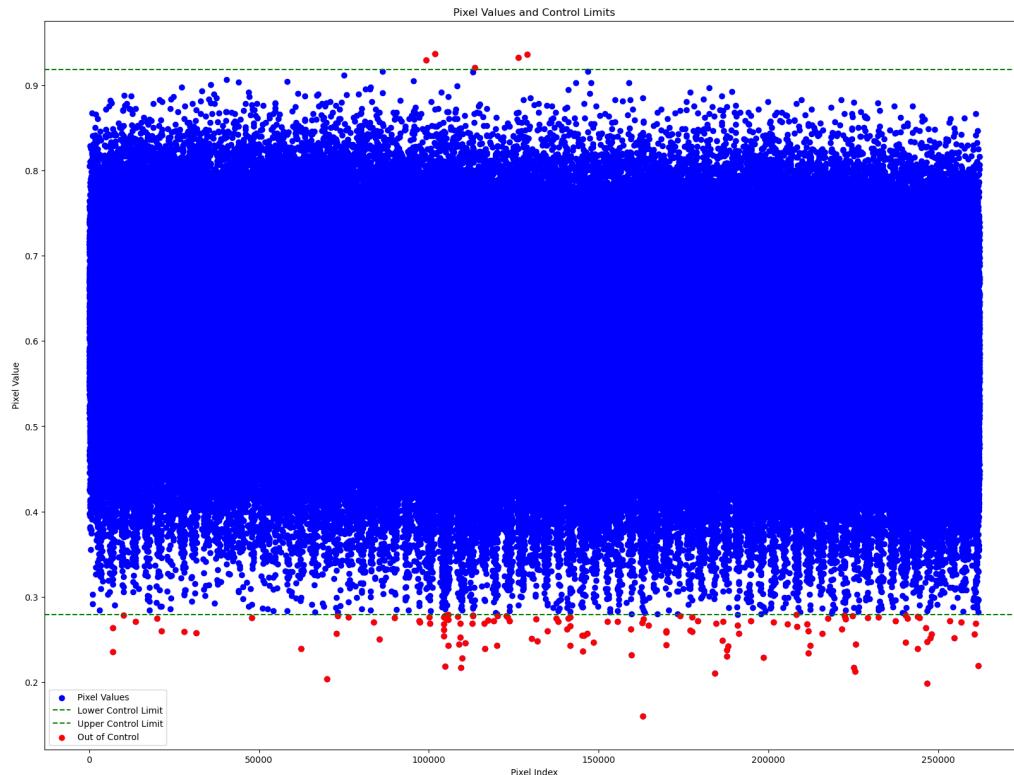
## **5. Conclusions and Future Work: Summarize Your Findings and Comments Regarding Your Approach. What Are Possible Extensions to Have a Better Approach?**

Our sample linen had a somewhat diagonal pattern, but it wasn't a perfect 45 degrees diagonal. When we tried to construct control charts for each individual row or column, this approach didn't make good sense since it simply found the average color value and pointed out extreme color-valued pixels. However, the main issue for our defect detection is also about the relative location of our pixels. For instance, a pixel may even possess the mean color value but might be located in a dark region and therefore need to be classified as an outlier, whereas the mentioned model would never classify it as such. Therefore, we argue that if the initial models were to be used, the only acceptable way to do so would be a model where the exact diagonal angle of the examined linen's pattern is found and new row and column matrices are produced according to this angle (or a new image that is aligned to this angle is taken and the default x and y axes are used). Thereby, each row will (and should) have extremely similar colored pixels, and an outlier will be found based on this. In this approach, all the pixels in a row or column will (in the ideal case) indeed need to possess the same color. Since we had a limited-sized image for our problem,

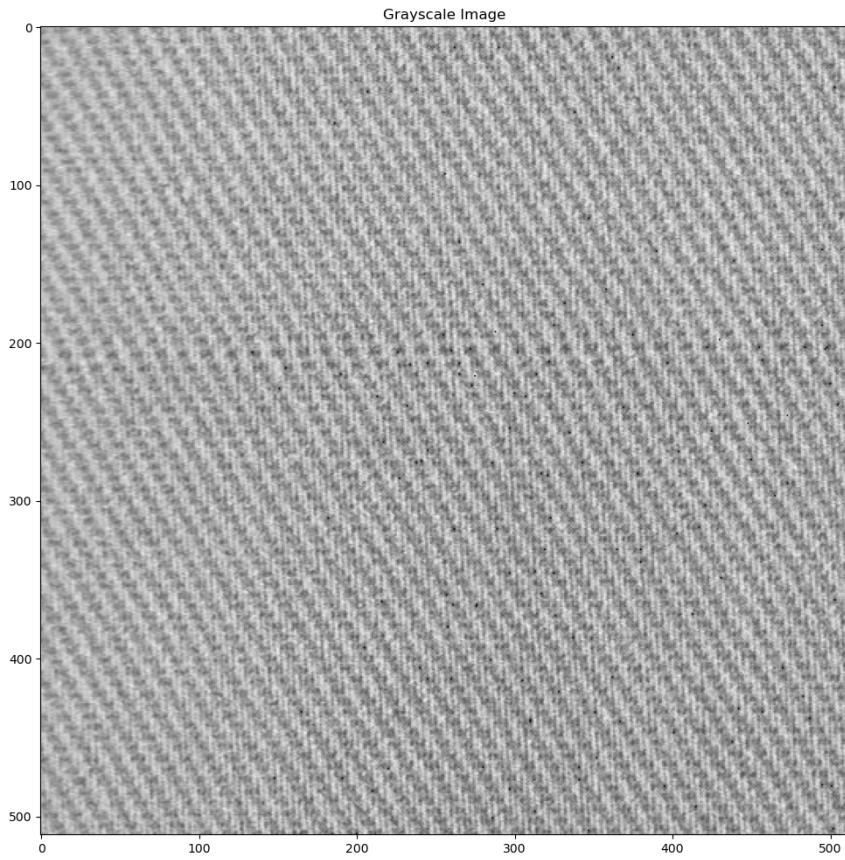
if we rotated the image so that the pattern was aligned with the x-axis, our rows would have different lengths, and it would be impossible to conduct a statistically significant study (since we would have very little sample with various row sizes). Therefore, we were motivated to come up with a better and more generalizable strategy that we developed.

Our proposal for a better future approach is to first use machine learning methods to identify the intended (ideal) linen pattern. From this ideal/aimed pattern, control charts with the target mean should be constructed (not the data entries but only the control limits). Then, our strategy should be implemented in a machine learning framework according to the features of the linen pattern in order to yield the most accurate defect detection result.

## Appendix:



*Figure-1*



*Figure-2*

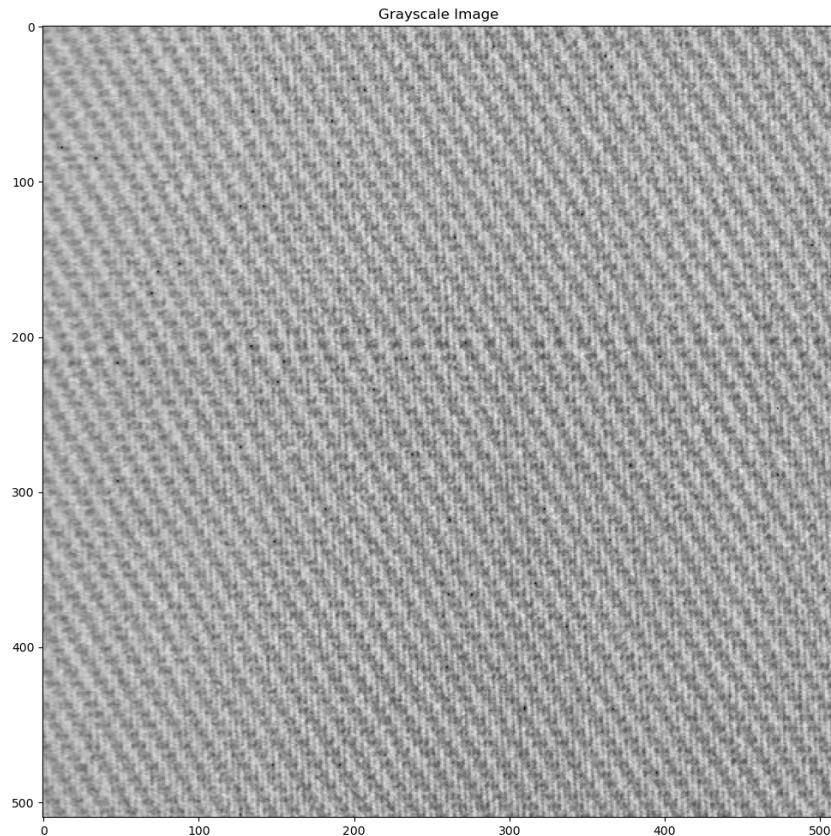


Figure-3

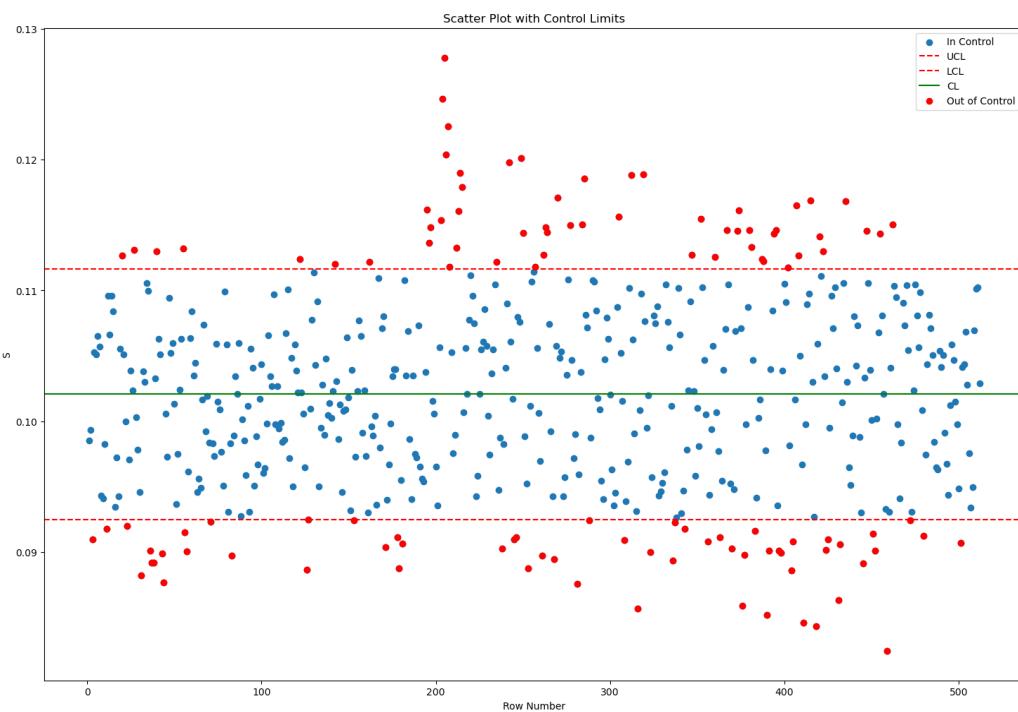
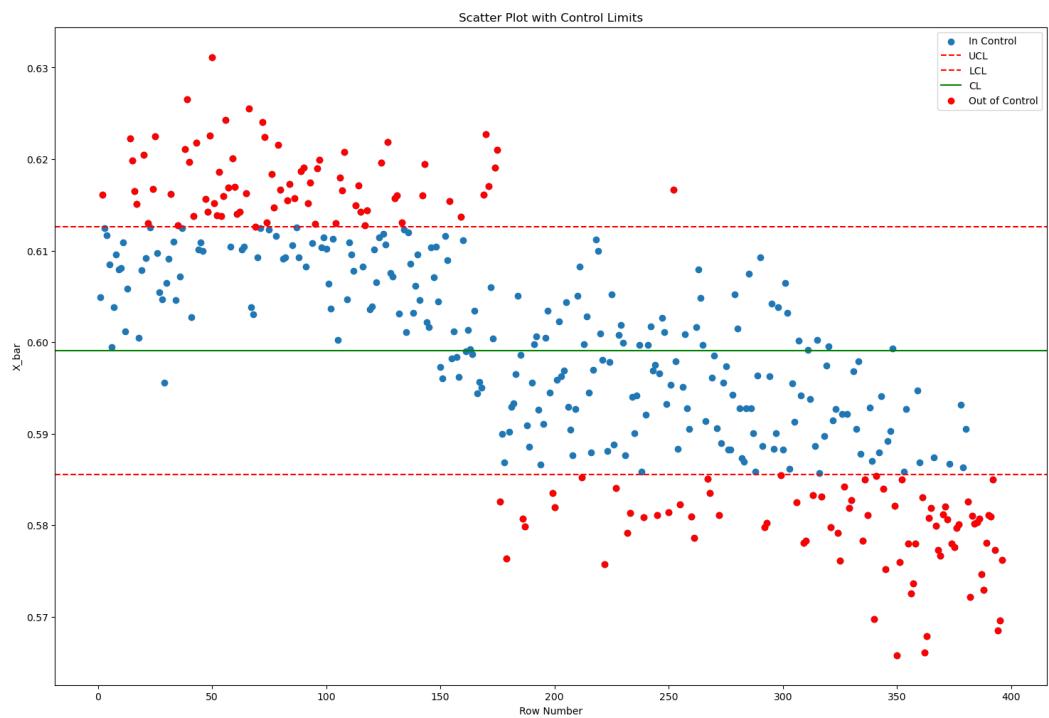
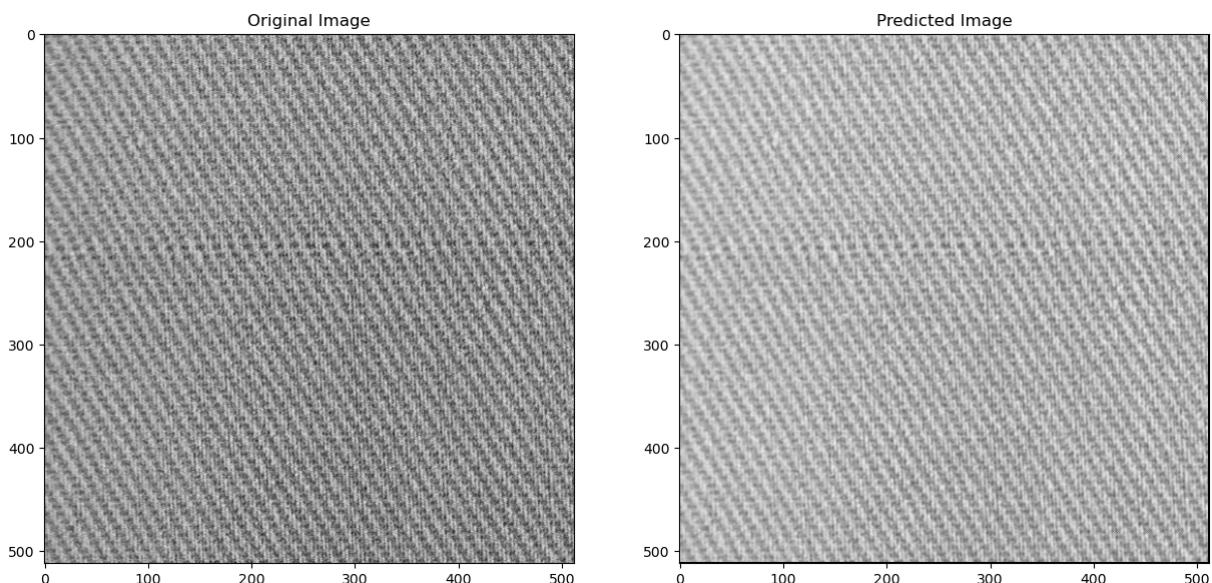


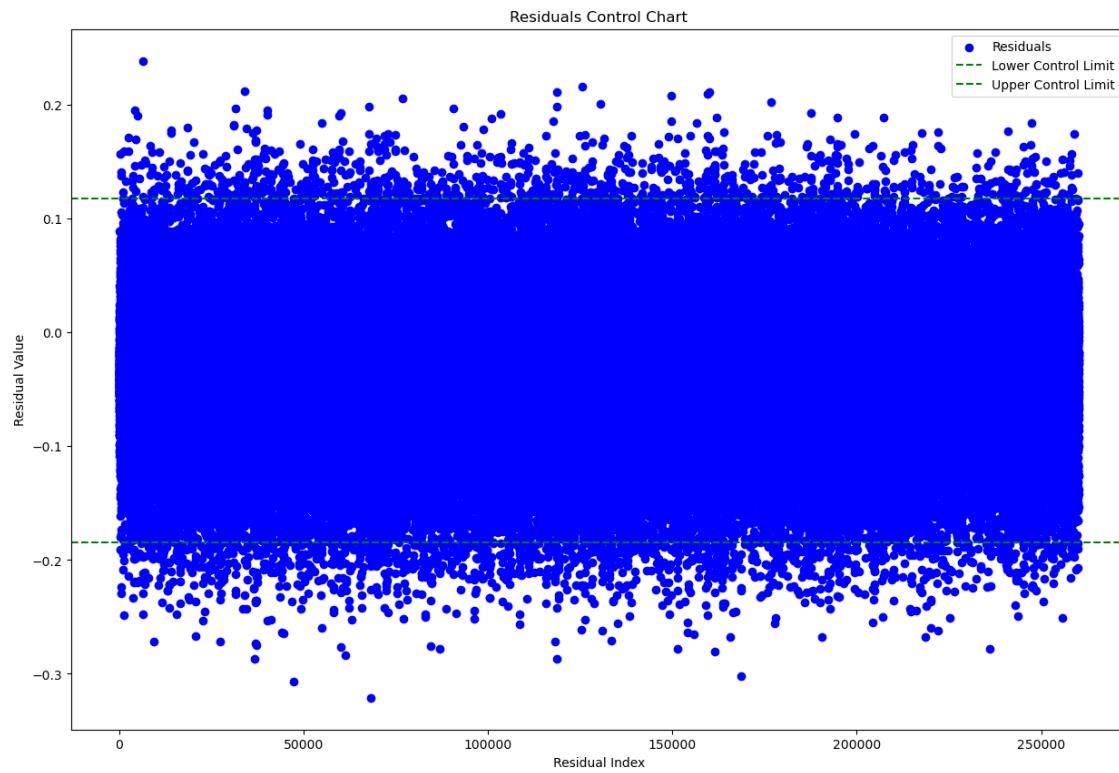
Figure-4



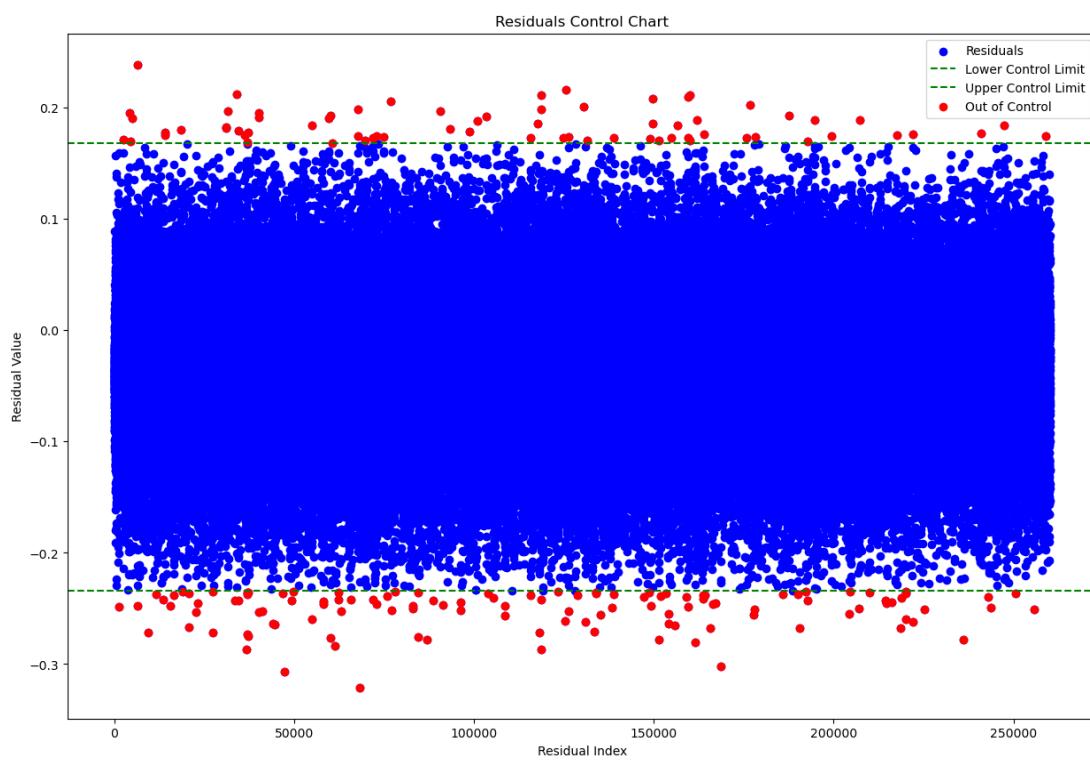
*Figure-5*



*Figure-6 (Original image , Reconstructed 3x3 Vicinity Image)*



*Figure-7*



*Figure-8*

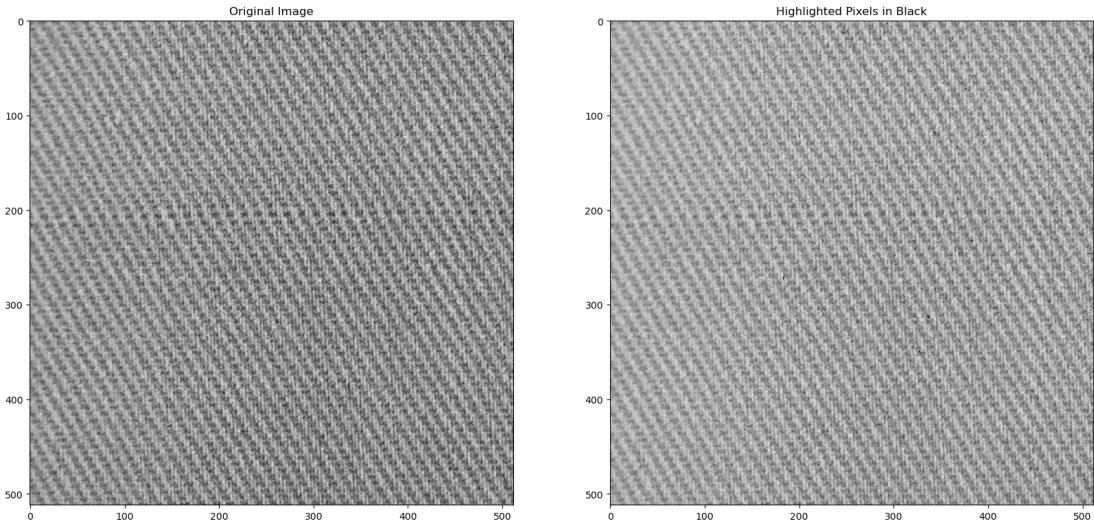


Figure-9

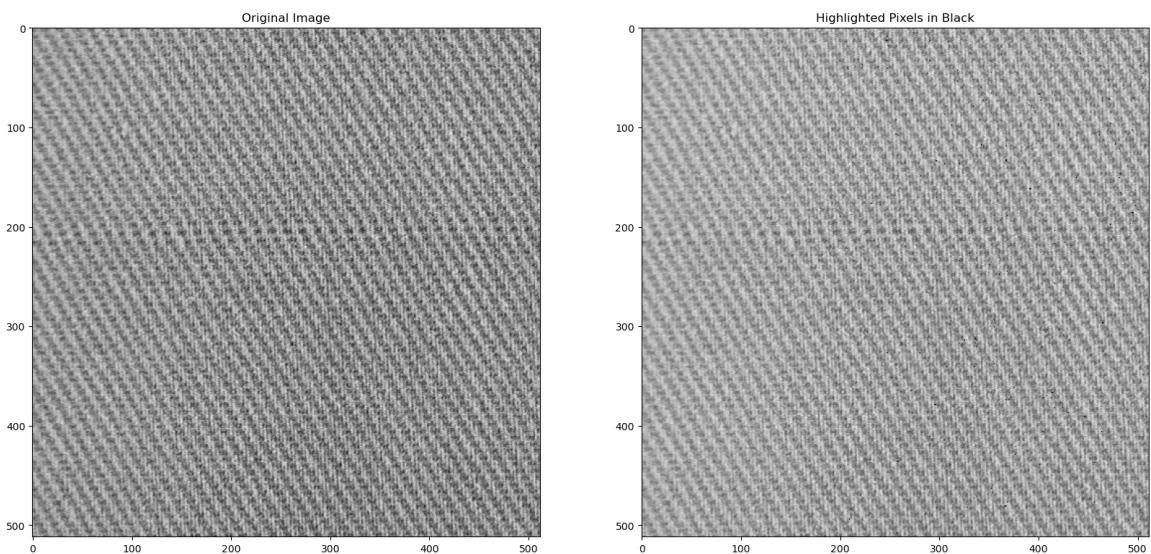
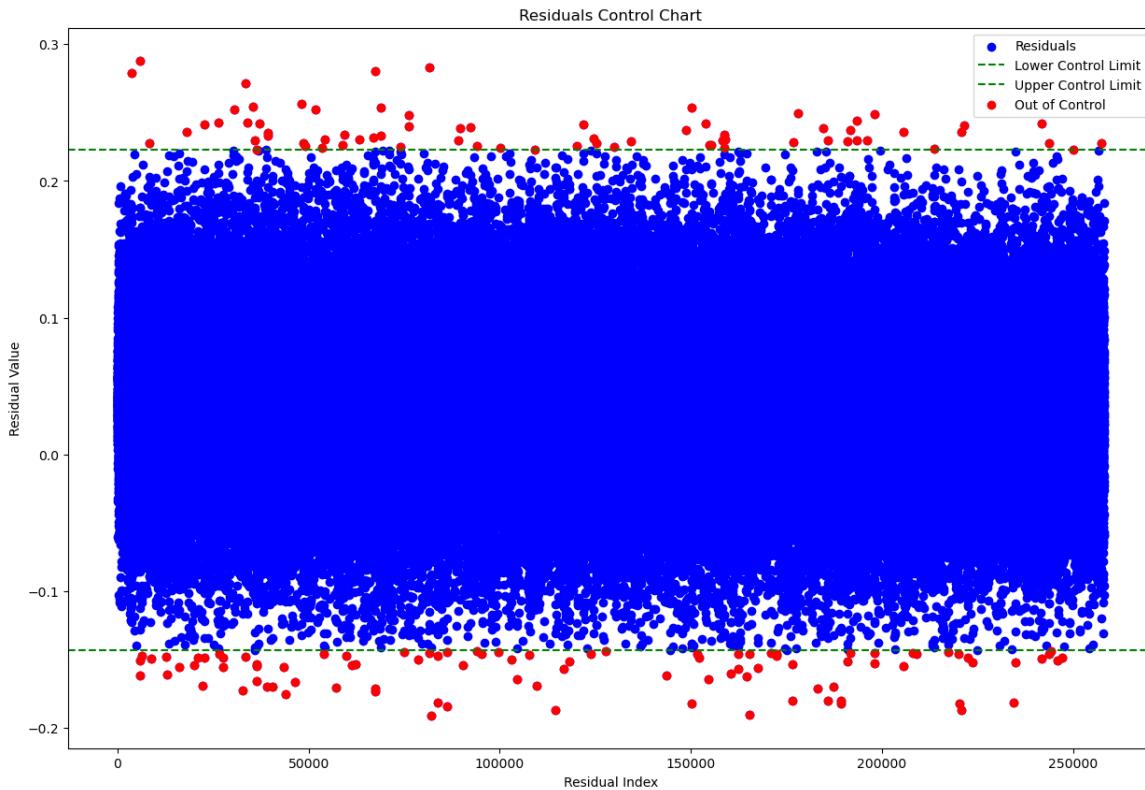


Figure-10



*Figure-11*

## References:

- Lim Fo Xiang (2023). Development of UTAR Hospital RFID Linen Monitoring through Gantry using JAVA. Universiti Tunku Abdul Rahman / Malaysia. Retrieved from: <http://eprints.utar.edu.my/5501/>
- Baghernezhad S. , Ghane M. , Moezzi M. (2016). Strain Monitoring in Woven Fabrics with Locally Induced Mass Irregularities Using an Image Based Method. Lodz Institute of Technology. Retrieved from: <https://yadda.icm.edu.pl/baztech/element/bwmeta1.element.baztech-556e8f82-00bf-43ce-9172-5dc8174d1da2>
- Vaishali D. , Ramesh R. , Christaline J. A. (2014). 2D Autoregressive Model for Texture Analysis and Synthesis. IEEE. Retrieved from: <https://ieeexplore.ieee.org/document/6950027/authors#authors>

- OpenAI. ( 2023). ChatGPT (Sep 25 version) [Large language model].  
<https://chat.openai.com/chat>.

M You

how can I get the pixel values of the greyscale image

ChatGPT

To get the pixel values of the grayscale image, you can access the elements of the NumPy array representing the image. The pixel values are the intensity values of the individual pixels. Here's an example:

```
from skimage import color, io
import numpy as np

# Read the image and convert it to grayscale
img = color.rgb2gray(io.imread('0055.jpg'))

# Get the pixel values as a NumPy array
pixel_values = np.array(img)

# Print the pixel values
print(pixel_values)
```

M You

I have 512x512 matrix that contains grayscale pixels of my image. I want to look at each 3x3 sub-matrix in it, and predict the pixel in the right middle of that 3x3 matrix as a weighted sum of the 8 pixels surrounding it. I need a 2D autoregressive model that would give the weights of these surrounding 8 pixels. Can you write the code for the model?

ChatGPT

Certainly! For a 2D autoregressive model, you can use a linear regression to predict the pixel in the right middle of each 3x3 sub-matrix based on the surrounding 8 pixels. The weights of these pixels will be the coefficients of the linear regression model.

Here's a sample code using the scikit-learn library for linear regression:

```

import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Assuming img_matrix is your 512x512 grayscale image matrix
# Replace this with your actual image data

# Function to extract 3x3 sub-matrices and their corresponding
target pixels
def extract_submatrices_and_targets(image_matrix):
    submatrices = []
    targets = []

    for i in range(1, image_matrix.shape[0] - 1):
        for j in range(1, image_matrix.shape[1] - 1):
            submatrix = image_matrix[i-1:i+2, j-1:j+2].flatten()
            target_pixel = image_matrix[i, j]
            submatrices.append(submatrix)
            targets.append(target_pixel)

    return np.array(submatrices), np.array(targets)

# Extract sub-matrices and targets
submatrices, targets = extract_submatrices_and_targets(img_matrix)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    submatrices, targets, test_size=0.2, random_state=42)

# Create and train a linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
predictions = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, predictions)
print(f"Mean Squared Error: {mse}")

# Display the coefficients of the model (weights)
weights = model.coef_
print("Model Coefficients (Weights):")
print(weights)

```

 **You**

Now I have my weights. I need to reconstruct my image by predicting each pixel by using these weights

 **ChatGPT**

To reconstruct your image using the weights obtained from the linear regression model, you can apply the same approach. Iterate through each  $3 \times 3$  sub-matrix in your original image, flatten it, and use the learned weights to predict the value of the central pixel. You can create a new image matrix based on these predicted values.

Here's an example of how you can do this:

```
# Assuming img_matrix is your 512x512 grayscale image matrix
# weights is the array of learned coefficients from the
linear regression model

# Function to reconstruct the image using learned weights
def reconstruct_image(image_matrix, learned_weights):
    reconstructed_image = np.zeros_like(image_matrix)

    for i in range(1, image_matrix.shape[0] - 1):
        for j in range(1, image_matrix.shape[1] - 1):
            submatrix = image_matrix[i-1:i+2, j-1:j+2].flatten()
            predicted_pixel = np.dot(submatrix, learned_weights)
            reconstructed_image[i, j] = predicted_pixel

    return reconstructed_image

# Reconstruct the image using learned weights
reconstructed_img = reconstruct_image(img_matrix, weights)

# Display the original and reconstructed images side by side
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(img_matrix, cmap='gray')
plt.title('Original Image')

plt.subplot(1, 2, 2)
plt.imshow(reconstructed_img, cmap='gray')
plt.title('Reconstructed Image')

plt.show()
```

 You

how can I come back from an array of length 260100 to 510x510 matrix



 ChatGPT

If you have a 1D array with a length of 260100 and you want to reshape it into a 510x510 matrix, you can use the `'reshape'` method of NumPy. Here's an example:

```
import numpy as np

# Assuming your 1D array is named 'flattened_array'
flattened_array = np.array([your_data_here]) # Replace with your actual data

# Reshape the array into a 510x510 matrix
reshaped_matrix = flattened_array.reshape(510, 510)

# Display the resulting matrix
print(reshaped_matrix)
```

 You

how can I delete each 5th element of an 510x510 array

 ChatGPT

If you want to delete every 5th element along a specific axis of a NumPy array, you can use array slicing and indexing. Here's an example of deleting every 5th element along the first axis (axis=0) of a 510x510 array:

```
import numpy as np

# Create a sample 510x510 array
my_array = np.random.random((510, 510))

# Delete every 5th element along the first axis
updated_array = np.delete(my_array, np.arange(4, my_array.shape[0], 5), axis=0)

print(updated_array.shape)
```

 You

does `my_matrix[a:b, c:d]` include row at index b

 ChatGPT

In Python, when using slicing notation like `'my_matrix[a:b, c:d]'`, the slice includes the starting index '`a`' but excludes the ending index '`b`'. Therefore, the row at index '`b`' is not included in the resulting slice.

For example, if '`a = 1`' and '`b = 4`', the slice '`'my_matrix[1:4, c:d]'`' will include rows at indices 1, 2, and 3, but not the row at index 4.