IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.1511117

AI Based Fabric Defect Detection System

Dr. U. M. Patil[1], Pranil Sanjay Pawar[2], Kanchan Sunil Patil[3],

Vaishnavi Chittaranjan Bhavsar[4], Jayesh Rameshsing Girase[5]

Head of Department, Computer Science and Engineering (Data Science), R. C. Patel Institute of Technology, Shirpur, India[1]

Student, Computer Science and Engineering (Data Science), R. C. Patel Institute of Technology, Shirpur, India[2,3,4,5]

Abstract: Fabric inspection in most mills is still done by people who look at the cloth on a lighted frame and mark the faults they see. After some time they get tired, two people may judge the same fault differently, and nothing is written down. In this paper we describe a fabric defect detection system that works on pixel values, runs in real time, and does not need a dataset of labelled defects or any special camera. Each frame is changed to grey, smoothed, and compared with an estimate of how plain fabric should look at that spot. Pixels that differ by more than a threshold, calculated from the frame itself, are marked, cleaned, joined into regions and sorted into hole, stain, knot, double knot or broken thread by simple shape rules. Every region gets a confidence value, and weak ones are dropped. The code runs on a laptop (FastAPI and OpenCV) and has three modes: image, video and Live Stream. For live use, the user scans a QR code with a phone on the same Wi-Fi and allows the camera. The faults then show on the phone screen while the laptop keeps the log. The length of fabric checked is estimated from feed speed and time, and each fault is saved in a SQLite table with its frame, time, position and size, which can be exported to CSV. In our test runs the fault area was found well in real time, but the class name was sometimes wrong when two faults look alike. We see this as the main weakness and the first thing to fix.

Keywords: Fabric defect detection, Pixel-based image processing, Real-time inspection, Smartphone camera, OpenCV, FastAPI.

I. INTRODUCTION

India has a large textile industry, and a loom delivers fabric in rolls that can be hundreds of metres long. Before a roll is cut, dyed or stitched, someone has to check it for holes, stains, knots, double knots and broken threads. If a faulty roll reaches the buyer, the lot may be rejected and the mill loses money and name [1], [2]. In most units this check is manual. The operator watches the fabric on a lighted frame and marks what he sees. People lose focus after a while, each person has his own standard, small faults get missed, and the exact place of a fault along the roll is usually not noted.



Fig. 1 Manual fabric inspection on a lighted frame

Figure 1 shows the problem we started with. The fabric is stretched on a lit frame and a person checks it by eye. We noticed the following problems with this way of working:

- **Tiredness:** small holes, light stains and thin broken threads are easy to miss after long hours.
- **Different opinions:** two inspectors can grade the same fault differently.

 This work is licensed under a Creative Commons Attribution 4.0 International License



International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026
DOI: 10.17148/IJARCCE.2026.1511117

- **No record:** findings are not stored, and the position of a fault on the roll is not measured.
- **Cost and data:** industrial vision systems are costly, and labelled fault images are not available to us.

Deep learning detectors like YOLO [3] give good accuracy, but they must be trained on many images in which every defect is labelled. A college lab or a small mill rarely has such images, and industrial line-scan cameras are also costly. So we wanted to see whether faults can be found from pixel values alone, in real time, with just a laptop and a phone camera. The idea is simple. Fabric is mostly uniform, so most pixels in a frame look alike, and a pixel that is far from its surroundings in brightness probably belongs to a fault. We use this to find such pixels and then turn them into regions that are classified, measured and saved.

The contributions of this paper are as follows. (1) The pixel-level theory and logic of the detector, which needs no labelled faults. (2) A rule-based classifier with a confidence filter for five fault types. (3) A Live Stream mode where the user only opens the phone camera and sees the faults on the screen. (4) Length estimation and a saved defect log that can be exported as CSV. (5) A frank account of what worked and what did not.

II. RELATED WORK

Surveys of fabric inspection are given in [1], [2] and [5]. Older methods use statistics of grey levels, or frequency tools such as Gabor filters and wavelets [6]. Newer ones train classifiers or deep networks [3]. Another line of work trains an autoencoder on normal images only and treats a bad reconstruction as a fault [8], [9]. Table I compares these with our system.

Table I Summary of related work

Approach	Data / Parameters	Algorithms / Techniques	Limitations and Gap
Manual inspection	Human vision, lighting frame	Visual search, manual marking	Fatigue, subjective, no digital record or position
Statistical and spectral	Grey-level statistics, filter responses	GLCM, Gabor filters, wavelets [6]	Tuned per fabric; poor on new weaves; not real time on phones
Supervised deep learning	Large labelled defect dataset	CNN, YOLO, Faster R-CNN [3], [4]	Labelled defects unavailable; fixed class list; heavy hardware
Autoencoder anomaly detection	Defect-free images only	Reconstruction error [8], [9]	Needs training; weak defect typing; no deployment workflow
Proposed pixel-based system	Pixel intensities of live frames	Background deviation, adaptive threshold, morphology, shape rules	No labels or training needed; real time on a phone; logged with position

What we found missing in most papers is the full working setup: a detector that needs no labels, a settings panel that a non-expert can handle, a real position for each fault, a saved log, and a camera that is just the phone already in the operator's pocket. That is what we tried to build.

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.1511117

III. METHODOLOGY

A. System Architecture

The system has a server and clients (Fig. 2). The laptop is the server. It holds the detection code, the database and the web dashboard. The clients are just web browsers: a desktop browser is used to upload images and videos and to read the log, and a phone browser works as the live camera. Nothing has to be installed on the phone.

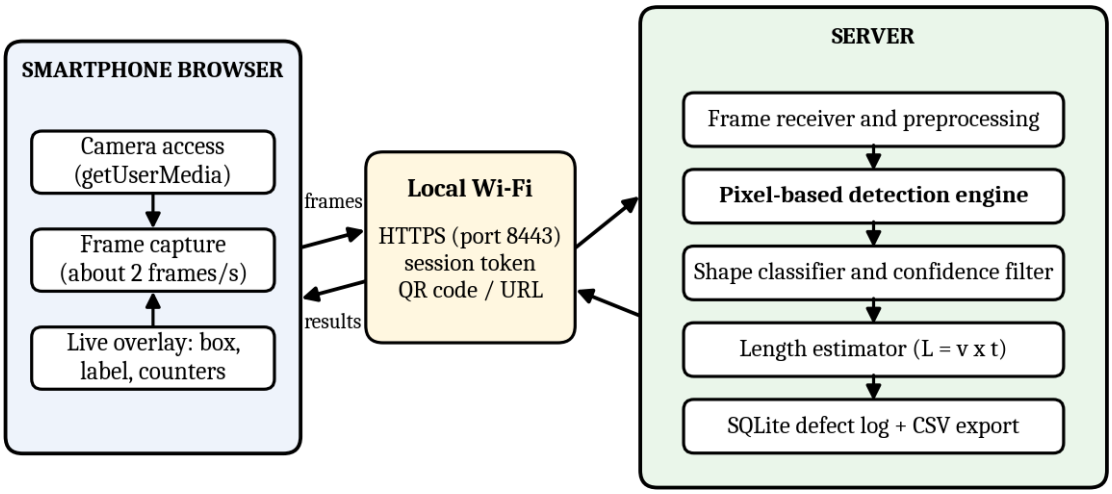


Fig. 2 System architecture

The dashboard has three tabs: **Image Analysis**, **Video Analysis** and **Live Stream**. The top of the page shows the current detection mode and the status. In all our runs the mode was *Heuristic Fallback*, which is the pixel-statistics detector described here. If a trained autoencoder is available, its reconstruction error can be used in place of the difference map of Eq. (4), and the remaining steps stay as they are.

B. Image as a Pixel Matrix

A frame is a matrix of pixels, $I \in R^{H \times W \times C}$, with height H, width W and C channels (3 for RGB). A fault changes the pixel values locally. A hole is much darker or brighter than the weave around it, a stain changes the brightness over some area, and a broken thread makes a thin line of a different brightness. So the task becomes finding the pixels that do not agree with the normal fabric.

C. Preprocessing

First the frame is resized to a fixed size and changed to grey. This way the dye colour does not matter and the work is less:

$$Y = 0.299\,R + 0.587\,G + 0.114\,B \quad (1)$$

Then a Gaussian filter G_σ removes sensor noise, which would otherwise give false pixels:

$$I_s(x, y) = (G_\sigma * Y)(x, y) \quad (2)$$

D. Background Estimate and Deviation

Fabric looks the same over large areas, so the normal value at a pixel can be estimated by blurring the image with a big kernel. The deviation map then tells how far each pixel is from this estimate:

$$B(x, y) = (G_\Sigma * I_s)(x, y), \quad \Sigma \gg \sigma \quad (3)$$

$$D(x, y) = | I_s(x, y) - B(x, y) | \quad (4)$$

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.1511117

E. Adaptive Statistical Threshold

A fixed threshold does not work when the light changes, so we take the threshold from the statistics of each frame. If μ_D and σ_D are the mean and standard deviation of the deviation map, a pixel is marked when it is more than k standard deviations above the mean:

$$T = \mu_D + k \cdot \sigma_D, \quad M(x, y) = 1 \text{ if } D(x, y) > T, \text{ else } 0 \qquad (5)$$

The value k is the **Sensitivity** slider on the dashboard (2.5 in our runs). A lower k picks up fainter faults but also gives more false alarms. A higher k does the opposite.

F. Morphology, Regions and Features

Opening removes single noise pixels and closing fills small gaps in the mask [11]. The connected regions are found by contour tracing [12], and regions smaller than a minimum area are thrown away. For each region we calculate the bounding box (w × h), the area A, the aspect ratio r = w/h, the circularity $Ci = 4\pi A/P^2$ (P is the perimeter), the solidity $S = A/A_{hull}$, and the average darkness compared with the background.

G. Confidence and Fabric Length

Each region gets a confidence between 0 and 100 %. It becomes higher when the deviation is well above the threshold and when the shape matches the rule of its class. Regions below the **Min Confidence** setting (30 % in our runs) are rejected. The length of fabric checked, and the position of a fault, come from the feed speed v and the time t:

$$L(t) = v \cdot t \qquad (6)$$

IV. ALGORITHM AND LOGIC

Figure 3 shows the whole processing logic, and the steps are listed after it.

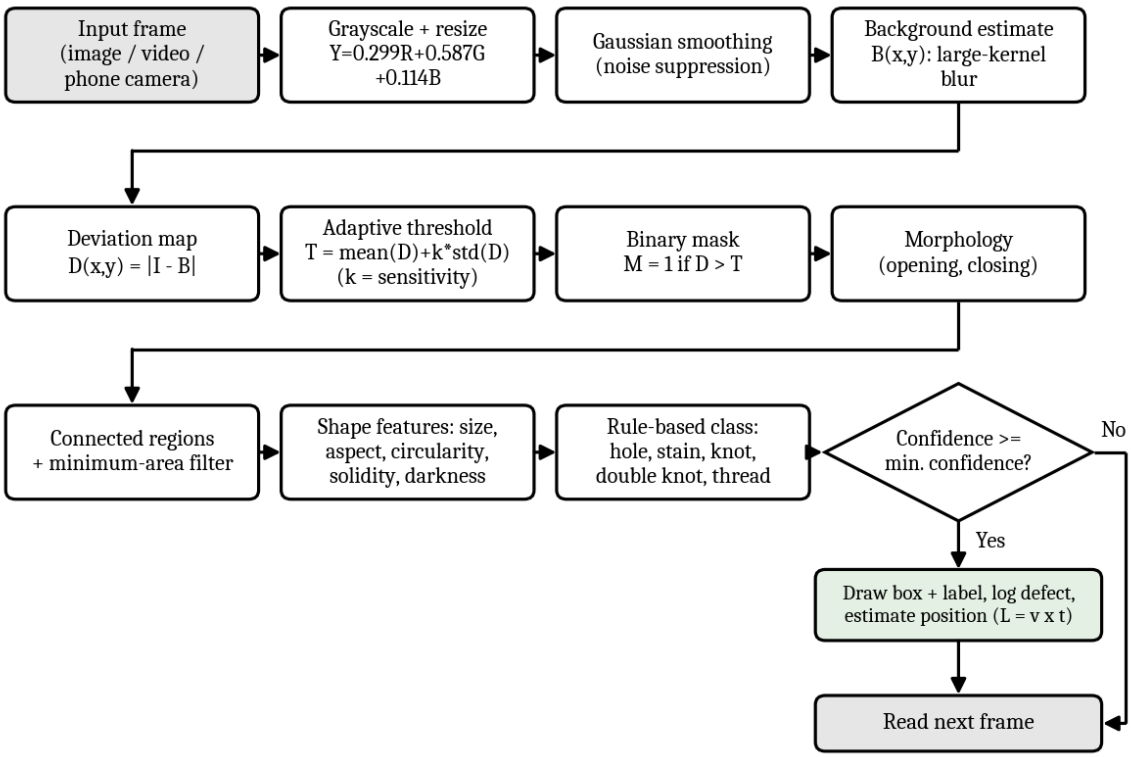


Fig. 3 Flowchart of the pixel-based detection logic

1. Take a frame from an uploaded image, an uploaded video or the live phone camera.
2. Change it to grey, resize and smooth it (Eqs. 1–2), and estimate the background (Eq. 3).
3. Find the deviation map and the adaptive threshold, and build the binary mask (Eqs. 4–5).
4. Clean the mask by morphology, extract the regions and remove the small ones.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.1511117

5. Calculate the shape features, give a class using the rules of Table II, and find the confidence.
6. If the confidence is at least Min Confidence, draw the box and label, estimate the position (Eq. 6) and write the entry in the log. Otherwise drop the region. Then read the next frame.

Table II gives the rules we use for classification. They are kept simple so that a teacher or supervisor can read them and change them.

Table II Rules for defect classification

Defect	Typical pixel appearance	Dominant cues
Hole	Compact, very dark (or very bright) region with sharp deviation	High darkness, high circularity
Stain	Irregular patch of moderate deviation over a larger area	Large area, low solidity, soft contrast
Knot	Small compact blob slightly different from weave	Small area, aspect ratio near 1
Double knot	Wider blob or pair of blobs	Area above knot range, aspect ratio about 2
Broken thread	Thin elongated line	Aspect ratio far from 1, low fill of the box

Table III lists the settings the operator can change on the dashboard.

Table III Detection settings

Setting	Value used	Effect
Sensitivity (k)	2.5	Number of standard deviations in the threshold T (Eq. 5)
Feed speed	10 m/min	Fabric speed v used for length and position (Eq. 6)
Min Confidence	30 %	Detections below this confidence are rejected

V. REAL-TIME LIVE STREAM MODE

The Live Stream mode is made for a person who has only a phone. The dashboard shows a QR code and the address of the laptop with a session token (Fig. 5a), and the status says *Waiting for Phone* until the phone connects. Then the operator:

1. checks that the phone is on the same Wi-Fi network as the laptop;
2. scans the QR code or opens the shown address, and accepts the browser certificate warning (the page is served over HTTPS because browsers allow camera access only on secure pages);
3. allows camera access, points the camera at the fabric and taps *Start Camera*.

After this the phone captures frames and sends them to the laptop over Wi-Fi. The laptop runs the logic of Fig. 3 and sends the result back, and the phone draws a coloured box with the class name and confidence on the live view (Fig. 5b). The phone screen also shows the frames processed, defects found, estimated length and frame rate (about 2 frames per second in our test), along with *Stop* and *Flip* (change camera) buttons. The same detections appear in the log on the laptop, and the session is closed with *End Session*. While adding this mode we did not change the existing image and video functions.

VI. IMPLEMENTATION

The back end is written in Python with FastAPI [14]. OpenCV [13] does the pixel operations, SQLite stores the detections, and the front end is an HTML, CSS and JavaScript dashboard. The phone camera is opened through the browser media capture interface [15]. A log entry holds the class, confidence, frame number, time, estimated position and box size. The dashboard also has a **Jump** button that moves the video to the frame of a fault, a **Show Debug Info** panel, **Export CSV** and **Clear Log**. Because every inspection is saved in this way, the log works as an audit trail.

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.1511117

VII. RESULTS AND DISCUSSION

We tested the system on a ten-second video of fabric moving on an inspection line (3.44 MB) and then live from a phone. Figure 4 shows the video analysis output and Fig. 5 shows the live session.

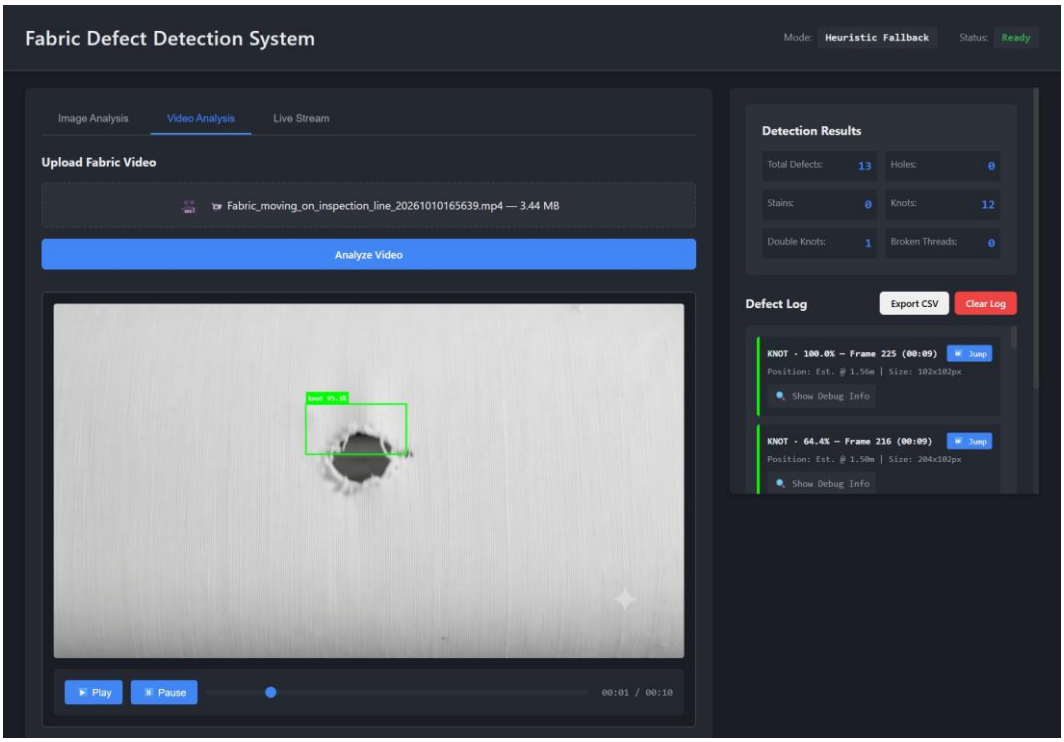


Fig. 4 Video analysis output with detected defect and defect log

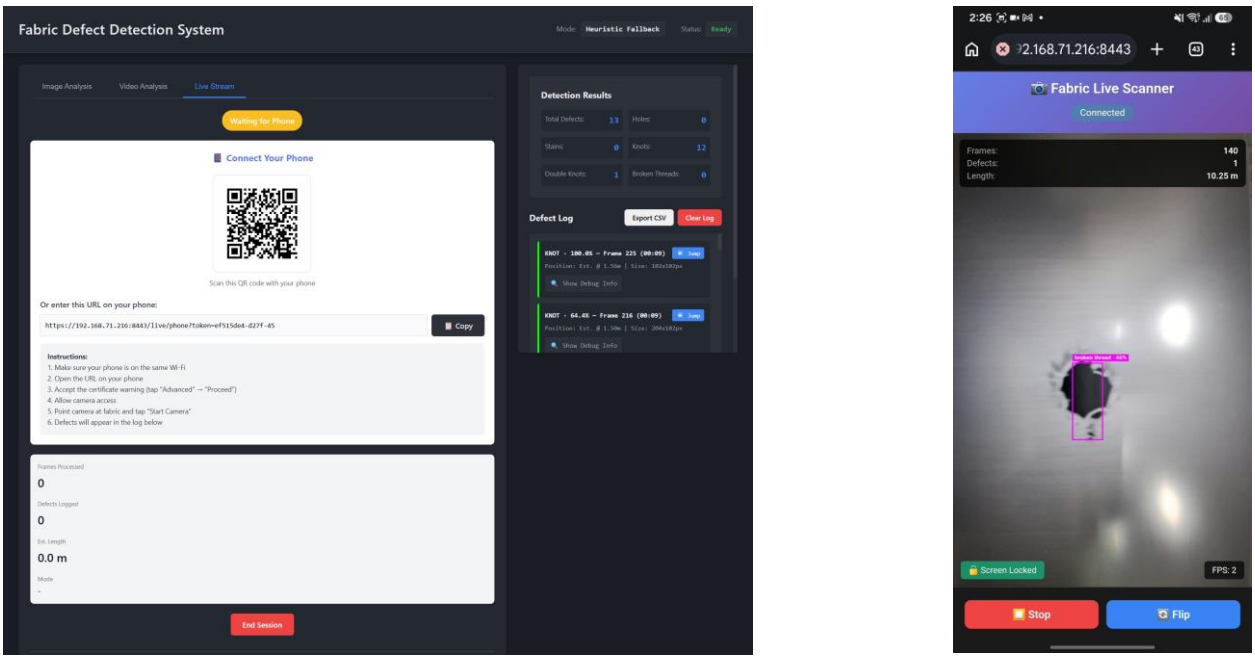


Fig. 5 Live Stream mode: (a) dashboard with QR code, (b) phone camera detection

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.1511117

Table IV lists the values we read from the two runs.

Table IV Observed outputs of the test runs

Item	Video analysis	Live phone stream
Input	10 s video, 3.44 MB	Phone camera over Wi-Fi
Settings	k = 2.5, 10 m/min, 30 %	k = 2.5, 10 m/min, 30 %
Total detections	13 (12 knot, 1 double knot)	1 defect logged
Frames / rate	Frames 216 and 225 shown in log	140 frames at about 2 FPS
Position	1.50 m and 1.56 m at about 00:09	Est. length 10.25 m
Example label	Knot 95.3 % (box on hole)	Broken thread 46 % (box on hole)

From these outputs we can say the following.

- **Finding the fault works.** In both runs the box sits on the dark irregular region that clearly differs from the weave. So the pixel deviation step finds the fault in real time.
- **Position matches the formula.** At 10 m/min a fault at 9.0 s should be at 1.50 m, and one at 9.375 s at 1.5625 m. The log shows 1.50 m (frame 216) and 1.56 m (frame 225), which matches Eq. (6) for a video of 24 frames per second.
- **Classification is the weakest part.** The fault in Figs. 4 and 5b looks like a hole, but the video run called it a knot (95.3 %) and the phone run called it a broken thread (46 %). The shape rules depend a lot on how the region is cut out. Here the frayed edge of the hole made the box larger and changed its aspect ratio.
- **Repeated counting.** The 13 detections in a ten-second video look like the same fault seen again in nearby frames (frames 216 and 225 are only nine frames apart). So the total is higher than the real number of faults, and detections must be joined by position and time.
- **Lighting.** The phone frame in Fig. 5b has bright reflections. Uneven light is the main cause of false deviation for any pixel-based method, and the Sensitivity setting is the practical way to handle it.

VIII. LIMITATIONS AND FUTURE WORK

Our evaluation is only observational. We have no ground-truth labels, so there are no precision or recall figures, and the test video is a demonstration clip and not footage from a production line. The rule-based classifier mixes up faults with similar outlines, the length estimate assumes a steady feed speed, and a phone on Wi-Fi gives only a few frames per second. In future we plan to (i) test on real fabric rolls with faults marked by an expert, (ii) merge detections of the same fault across frames, (iii) replace or support the rules with a classifier trained on the confirmed entries of the log, (iv) try structural similarity or autoencoder reconstruction error as a better deviation measure [8], [9], and (v) use a roller encoder for exact length.

IX. CONCLUSION

We built a real-time fabric defect detector that works on pixel values and needs no labelled faults. It estimates the normal background, marks the pixels that differ by more than an adaptive threshold, cleans the mask and classifies the regions with simple shape rules. In Live Stream mode the user opens the phone camera, the faults appear on the screen, and the laptop logs them with their position. Our test runs show that faults are located well and the length estimate is consistent, but the class labels are the weakest part and will be improved first. We think this is an affordable and easy to explain way to start digital fabric inspection.

ACKNOWLEDGMENT

The authors thank their guide, Dr. U. M. Patil, Head of the Department of Computer Science and Engineering (Data Science), R. C. Patel Institute of Technology, Shirpur, for the guidance and the facilities.

REFERENCES

[1]. A. Kumar, “Computer-vision-based fabric defect detection: A survey”, *IEEE Transactions on Industrial Electronics*, vol. 55, no. 1, pp. 348-363, 2008.



International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026
DOI: 10.17148/IJARCCE.2026.1511117

[2]. H. Y. T. Ngan, G. K. H. Pang and N. H. C. Yung, "Automated fabric defect detection—A review", *Image and Vision Computing*, vol. 29, no. 7, pp. 442-458, 2011.
[3]. J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You only look once: Unified, real-time object detection", in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pp. 779-788, 2016.
[4]. S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks", in *Advances in Neural Information Processing Systems 28 (NIPS)*, 2015.
[5]. K. Hanbay, M. F. Talu and Ö. F. Özgüven, "Fabric defect detection systems and methods—A systematic literature review", *Optik*, vol. 127, no. 24, pp. 11960-11973, 2016.
[6]. A. Kumar and G. K. H. Pang, "Defect detection in textured materials using Gabor filters", *IEEE Transactions on Industry Applications*, vol. 38, no. 2, pp. 425-440, 2002.
[7]. N. Otsu, "A threshold selection method from gray-level histograms", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62-66, 1979.
[8]. P. Bergmann, M. Fauser, D. Sattlegger and C. Steger, "MVTec AD—A comprehensive real-world dataset for unsupervised anomaly detection", in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR)*, pp. 9592-9600, 2019.
[9]. G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks", *Science*, vol. 313, no. 5786, pp. 504-507, 2006.
[10]. R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed. New York, NY, USA: Pearson, 2018.
[11]. R. M. Haralick, S. R. Sternberg and X. Zhuang, "Image analysis using mathematical morphology", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 9, no. 4, pp. 532-550, 1987.
[12]. S. Suzuki and K. Abe, "Topological structural analysis of digitized binary images by border following", *Computer Vision, Graphics, and Image Processing*, vol. 30, no. 1, pp. 32-46, 1985.
[13]. G. Bradski, "The OpenCV Library", *Dr. Dobb's Journal of Software Tools*, 2000.
[14]. S. Ramírez, "FastAPI documentation". [Online]. Available: https://fastapi.tiangolo.com
[15]. W3C, "Media Capture and Streams". [Online]. Available: https://www.w3.org/TR/mediacapture-streams/

 This work is licensed under a Creative Commons Attribution 4.0 International License