Tracking yarns in high resolution fabric images: A real-time approach for online fabric flaw detection

Dorian Schneider
dorian.schneider@ifb.rwth-aachen.de
Institute of Imaging & Computer Vision
RWTH Aachen University, 52056 Aachen, Germany

ABSTRACT
An algorithmic framework for real-time localization of single yarns within industrial fabric images is presented. The information about precise yarn locations forms the foundation for a fabric flaw detection system which is based on individual yarn measurements. Matching a camera frame rate of 15 fps, we define the term “real-time” by the capability of tracking all yarns within a 5 megapixel image in less than 35 ms, leaving a time slot of 31 ms for further image processing and defect detection algorithms. The processing pipeline comprises adaptive histogram equalization, Wiener deconvolution, normalized template matching and a novel feature point sorting scheme. To meet real-time requirements, extensive use of the NVIDIA CUDA framework is made. Implementation details are given and source code for selected algorithms is provided. Evaluation results show that wefts and warps can be tracked reliably and independently of the fabric material or binding. Video and image footage is provided on the project website to expand the paper content.

Keywords: Fabric defect inspection, Textile quality control, On-loom flaw detection, Automated Visual Inspection (AVI), yarn detection, real-time, Wiener deconvolution, CLAHE, ZNCC

1. INTRODUCTION
This paper discusses a framework for tracking single yarns within highly resolved fabric images in real-time. Modern vision based quality control systems for woven fabrics are based on the idea that fabrics appear like near regular, repetitive textures. Detecting irregularities in these pattern seems to be a promising approach to successfully segment potential defective areas. However, most of the proposed approaches operate on low image resolutions (max. 200 ppi, 256x256 resolution) and are limited in terms of precision and processing speed. This work is motivated by the idea of characterizing defects within fabric images by trajectory, position, shape, and texture of isolated yarns rather than treating the whole image as a texture. Detecting and measuring single yarns enables automatic visual inspection systems to perform with a higher degree of precision and reliability, while offering more than just “emergency stop” signals to be fed back to the loom control. This work forms the foundation for a real-world, vision based on-loom fabric defect detection system.1,2 Within this scope, strict algorithmic requirements for processing speed and robustness apply. The fixed camera frame rate of 15 fps, leaves a margin of 66 ms for the system to analyse each image with respect to its yarn structure and extract information of potential defects from it. The input images have a data volume of 5 megapixel each and a spatial resolution of more than 1000 ppi. It is shown that the system works equally well for varying weft densities (pickages), materials as well as fabrics with either plain, twill or satin weaves. This work makes extensive use of the NVIDIA CUDA framework for GPU accelerated computing. Our real-time implementation can track yarn trajectories in less than 35 ms – leaving enough processing time for subsequent flaw detection. On the project website1 source codes and additional video footage are provided.

The paper is structured as follows. Section 2 discusses image enhancement techniques to reduce motion blur, normalize illumination inhomogeneities, and improve contrast of the raw input images by using GPU accelerated Wiener deconvolution and Clip Limited Adaptive Histogram Equalization4 (CLAHE). Section 3 covers real-time feature point extraction using a template matching approach and a zero mean normalized cross correlation7 (ZNCC). In section 4, feature points are mapped into a compact yarn matrix, which acts as a look-up table for yarn positions within the image. The results are presented in section 5 and section 6 concludes the work.
As depicted in Figure 1, the proposed AVI system employs a camera sled which moves at high speed along a fixed traverse mount to visually cover the complete material during production. Due to the fast motion, the input image is degraded by motion blur. Even though the blur effect can be minimized by adjusting the interplay between shutter speed, flash power and camera aperture, software based de-blurring helps to optimize and normalize the raw input images. We benchmarked 13 state-of-the-art algorithms for non-blind deconvolution and provided details on the implementation of Wiener deconvolution for real-time applications using GPU acceleration, source code is provided on the project website. Further, Clip Limited Adaptive Histogram equalization (CLAHE) is applied to enhance contrast and correct illumination inhomogeneities in raw input images (especially towards the borders). The application of CLAHE is an important step to ensure that the subsequent feature extraction process works reliably. In contrast to global histogram equalization, CLAHE is a local operator able to smoothly enhance contrast in the image despite considerable local illumination inhomogeneities (generally caused by local fabric defects and material
variabilities) may be present. To this end, the image is divided into a grid of \( N \times M \) non-overlapping tiles of the same size. For each tile, a local histogram (i.e. a probability density function for the pixel intensities) \( \text{pdf}_{m,n}(x) \) \((\forall m,n \in \mathbb{N}, 0 \leq m \leq M - 1, 0 \leq n \leq N - 1)\) is calculated. The cumulative distribution function for each histogram is then given by

\[
cdf_{m,n}(x) = \sum_{j=0}^{x} \text{pdf}_{m,n}(j).
\] (1)

It forms the basis for the general histogram equalization transformation function given by

\[
\text{hetf}_{m,n}(x) = \text{round}\left(\frac{\text{cdf}_{m,n}(x) - \min(\text{cdf}_{m,n})}{(N \times M - \min(\text{cdf}_{m,n}))} \cdot (B - 1)\right),
\] (2)

where \( B \) denotes the number of bins in the histogram and is generally fixed to 256. To control the effect of noise amplification in homogeneous tiles, CLAHE limits the slope (and hence the contrast enhancement strength) of the histogram equalization transformation function by clipping the histogram values in the corresponding local histograms \( \text{pdf}_{m,n}(x) \) at a certain limit. The clipped parts of each histogram are cumulated and evenly redistributed among all bins after clipping. The clipping value is an adjustable parameter expressed as percentage to the average bin value of the local histogram.

Once \( \text{cdf}_{m,n}(x) \) is determined for each tile, the intensity values of every pixel in a tile are transformed according to (2) using inter-tile interpolation to diminish artefacts along tile borders. For this reason, bilinear interpolation with four transformation functions is used to map the intensity values of pixels located in center tiles of the image. Linear interpolation using two transformation functions is used for pixels of border tiles and no interpolation is used for pixels residing in the four corner tiles. Figure 2 clarifies the interpolation procedure.

### 2.1 CLAHE real-time implementation

To meet real-time requirements, the CLAHE procedure has been entirely implemented using GPU acceleration. The core functionality consists of four index look-up matrices \( I_{1-4} \) and four coefficient look-up matrices \( C_{1-4} \) which have the same size as the input image and are computed off-line when the system is initialized.

While initializing, the image is divided into tiles according to the user provided splitting scheme and input image size. The 2D grid is continuously numbered so that each tile gets a unique index. For each pixel position \((x, y)\) within the image, the corresponding indices of the four nearest tile centers 1-4 are determined and stored each in one of the four index matrices \( I_{1-4} \) at position \((x, y)\). Zeros are stored if no 2nd, 3rd or 4th adjacent tile exists, i.e. for corner or border pixels. Subsequently, the bilinear interpolation coefficients 1-4 are calculated for each image position \((x, y)\) and stored in the coefficient matrices \( C_{1-4} \) at the position \((x, y)\). Again, zeros are stored if no 2nd, 3rd or 4th adjacent quarter tile exist. The eight precomputed look-up matrices are uploaded to the GPU and marked for read access only.

During on-line processing, the raw image is uploaded to the GPU in the first step. According to the earlier determined grid layout, it is is subsequently split into tiles and for each tile, a 1D histogram is calculated using a CUDA histogram kernel. \(^5\) If clipping is enabled, the clip limit is subtracted from each bin in the histogram (lower limit 0) and the total sum differences is calculated. All histogram values are then clipped to the given parameter and the average of the total difference sum is added to each bin.
To find the cumulative distribution function for each tile according to (1), a 1D scan kernel is applied and the distribution functions are converted to histogram equalization transformation functions according to (2). The transformation functions are finally stored in a $1 \times M \cdot N$ matrix $T_{1-NM}$. The ordering within the matrix must be the same as the indexing of the tiles. Then, straightforward look-up operations can be performed to locally enhance the contrast in the image without visible border artefacts. The final intensity mapping kernel loads 4 bytes at a time from the input image to ensure coalescence and to speed up the process. By applying the proposed scheme, the CLAHE operation can be reduced to roughly 16 look-up operations, 4 multiplications and 4 additions for every pixel in the image. The pseudo kernel shown in Listing 1 facilitates the understanding of the concept.

Listing 1. CUDA kernel for real-time CLAHE calculation

```c
__global__ void claheMapping(int* src, // input image
  int* dst, // destination image
  float** coeff, // look-up matrix $C_{1-4}$
  int** indices, // look-up matrix $I_{1-4}$
  int** histIndices, // look-up array $T_{1-NM}$
  int nPixels // number pixels in a tile
) {

  // 1D index of the current pixel location
  int pixelID = threadIdx.x + blockIdx.x;

  if(pixelID < nPixels){
    int writeBack; // variable for the final output
    int a, b, c, d; // temp variables for the four pixel intensities

    // interpolation values
    float interpValA, interpValB, interpValC, interpValD;
    // read four bytes at a time
    a = ((src[pixelID] >> 0) & 0x000000FF); // intensity of byte 1
    b = ((src[pixelID] >> 8) & 0x000000FF); // intensity of byte 2
    c = ((src[pixelID] >> 16) & 0x000000FF); // intensity of byte 3
    d = ((src[pixelID] >> 24) & 0x000000FF); // intensity of byte 4

    // for each byte, look up four coefficients and four mapping values
    // and calculate the weighted sum as final output
    // Byte 1
    interpValA += coeff[0][byteID + histIndices[3][byteID][a]];
    interpValA += coeff[1][byteID + histIndices[2][byteID][a]];
    interpValA += coeff[2][byteID + histIndices[1][byteID][a]];
    interpValA += coeff[3][byteID + histIndices[0][byteID][a]];

    // Byte 2
    interpValB += coeff[0][byteID + 1 + histIndices[3][byteID + 1][b]];
    interpValB += coeff[1][byteID + 1 + histIndices[2][byteID + 1][b]];
    interpValB += coeff[2][byteID + 1 + histIndices[1][byteID + 1][b]];
    interpValB += coeff[3][byteID + 1 + histIndices[0][byteID + 1][b]];

    // Byte 3
    interpValC += coeff[0][byteID + 2 + histIndices[3][byteID + 2][c]];
    interpValC += coeff[1][byteID + 2 + histIndices[2][byteID + 2][c]];
    interpValC += coeff[2][byteID + 2 + histIndices[1][byteID + 2][c]];
    interpValC += coeff[3][byteID + 2 + histIndices[0][byteID + 2][c]];

    // Byte 4
    interpValD += coeff[0][byteID + 3 + histIndices[3][byteID + 3][d]];
    interpValD += coeff[1][byteID + 3 + histIndices[2][byteID + 3][d]];
    interpValD += coeff[2][byteID + 3 + histIndices[1][byteID + 3][d]];
    interpValD += coeff[3][byteID + 3 + histIndices[0][byteID + 3][d]];

    // pack four bytes and write them back as block
    dst[pixelID] = writeBack;
  }
}

// end claheMapping
```

Table 1 summarizes the time consumption of the proposed CLAHE implementation, source code is available on the project website.

3. FEATURE POINT DETECTION

In order to retrieve the lattice information of the fabric image, the proposed framework applies a template matching algorithm to detect characteristic feature points. As depicted in Figure 3, woven fabrics are produced by interweaving horizontally passing yarns (wefts) and vertically passing yarns (warp) according to a predefined weaving pattern (binding). We denote all weft-warp intersections within the image where warps lie above wefts as node points (NP) – these locations form the feature points of interest. As for the pre-processing step, the feature point extractor is preceded by an off-line setup where the machine operator calibrates the system by manually selecting an NP-template from a defect free reference image (cf. Figure 3).

We tested and evaluated several similarity based correlation measures like SAD, SHD, SSD and NCC in terms of NP-detection precision and real-time application. Only the zero-mean-normalized-cross-correlation (ZNCC) measure proved to be robust enough for reliable feature point detection. It should be stated that none of the investigated measures detected reliably NPs in unprocessed raw images, making pre-processing indispensable.

Lewis published an elegant way to efficiently calculate the ZNCC using the FFT in combination with summed area tables (i.e. integral images). The derivation is in the following further expanded for an optimal GPU implementation. Given the general formula for the ZNCC by:

\[
f(u, v) = \frac{n(u, v)}{d(h, v)} = \frac{\sum_{x,y} [f(x, y) - \bar{f}_{u,v}] [t(x-u, y-v) - \bar{t}]}{\sqrt{\sum_{x,y} [f(x, y) - \bar{f}_{u,v}]^2 \sum_{x,y} [t(x-u, y-v) - \bar{t}]^2}}.
\]

Here, \(t(x, y)\) is the template image with width \(Y\) and height \(X\), \(\bar{t}\) is the template mean, \(f(x, y)\) is the pre-processed input image with width \(U\) and height \(V\), \(\bar{f}_{u,v}\) is the mean of the input image region under the template and \(f(u, v)\) is the correlation result with the same size as the input image. With \(t'(x-u, y-v) = |t(x-u, y-v) - \bar{t}|\), the numerator can be reformulated according to Lewis:

\[
n(u, v) = \sum_{x,y} [f(x, y) - \bar{f}_{u,v}] t'(x-u, y-v)
= \sum_{x,y} f(x, y) t'(x-u, y-v) - \sum_{x,y} f_{u,v} t'(x-u, y-v)
= \sum_{x,y} f(x, y) t'(x-u, y-v) - \bar{f}_{u,v} \sum_{x,y} t'(x-u, y-v)
\]

and since \(t'\) has no mean, \(\sum_{x,y} t'(x-u, y-v) = 0\), leaving

\[
n(u, v) = \sum_{x,y} f(x, y) t'(x-u, y-v) = \mathcal{F}^{-1} [\mathcal{F}(f(x, y)) \cdot \mathcal{F}^*(t'(x, y))].
\]

More, the denominator can be rearranged to

\[
d(u, v) = \sqrt{\sum_{x,y} [f(x, y) - \bar{f}_{u,v}]^2 \sum_{x,y} [t(x-u, y-v) - \bar{t}]^2}
\]

\[
\sum_{x,y} [f(x, y) - \bar{f}_{u,v}]^2 = \sum_{x,y} [f^2(x, y) - 2\bar{f}_{u,v} f(x, y) + \bar{f}_{u,v}^2]
= \sum_{x,y} f^2(x, y) - 2\bar{f}_{u,v} \sum_{x,y} f(x, y) + \bar{f}_{u,v}^2
= \sum_{x,y} 2\bar{f}_{u,v} f(x, y) = 2\bar{f}_{u,v} \sum_{x,y} f(x, y) = 2A\bar{f}_{u,v}^2
\]

\[\text{(since} \sum_{x,y} 2\bar{f}_{u,v} f(x, y) = 2\bar{f}_{u,v} \sum_{x,y} f(x, y) = 2A\bar{f}_{u,v}^2)\]
\[
\sum_{x,y} f^2(x, y) - 2A \bar{f}^2_{u,v} + A \cdot \bar{f}^2_{u,v} = \sum_{x,y} f^2(x, y) - A \cdot \bar{f}^2_{u,v}
\]

\[
\sum_{x,y} f^2(x, y) - \frac{1}{A} \left[ \sum_{x,y} f(x, y) \right]^2
\]

\[
\Rightarrow d(u, v) = \sqrt{\left[ \sum_{x,y} f^2(x, y) - \frac{1}{A} \left[ \sum_{x,y} f(x, y) \right]^2 \right]^2 \sum_{x,y} t'^2}
\]

\[
= \sqrt{\sum_{x,y} f^2(x, y) - \frac{1}{A} \left[ \sum_{x,y} f(x, y) \right]^2} \cdot \sqrt{A \cdot \sigma_t}, \tag{5}
\]

where \( \sigma_t \) is the standard deviation of the template and \( A = (X \cdot Y) \), the number of pixels in the template. The numerator (4) and denominator (5) are combined to give the final correlation equation

\[
f_{xcorr}(u, v) = \frac{\mathcal{F}^{-1} [\mathcal{F}(f(x, y)) \cdot \mathcal{F}^*(t'(x, y))]}{\sqrt{\sum_{x,y} f^2(x, y) - \frac{1}{A} \left[ \sum_{x,y} f(x, y) \right]^2} \cdot \sqrt{A \cdot \sigma_t}} \tag{6}
\]

The correlated output image \( f_{xcorr} \) is further filtered by a non-maximum filter operation according to:

\[
f_{dilated}(u, v) = f_{xcorr}(u, v) \oplus \text{str}_{\text{rect}} \tag{7}
\]

\[
f_{\text{nodes}}(u, v) = \begin{cases} 1 & \text{if } f_{\text{dilated}}(u, v) = f_{\text{corr}}(u, v) \\ 0 & \text{otherwise.} \end{cases} \tag{8}
\]

Here, the operator \( \oplus \) denotes a gray value dilation and \( \text{str}_{\text{rect}} \) is a rectangular structure element with fixed edge length. The xy-coordinates in \( f_{\text{nodes}} \) which are unequal zero, mark the node point features in the image and are saved in a list \( L_{\text{NP}} \) for the subsequent yarn mapping.

### 3.1 ZNCC real-time implementation

The following steps are conducted for an efficient real-time implementation of ZNCC:

1. Offline: Calculation of the term \( \sqrt{A} \cdot \sigma_t \) in (6).
2. Offline: Calculation of the conjugate Fourier transformation \( \mathcal{F}^*(t'(x, y)) \) of the zero mean template using the cuFFT library.\(^9\)
3. Online: Calculation of the squared summed area table \( I^2_f \) and the summed area table \( I_f \) of the squared input image using NVIDIA’s NPP library.\(^10\)
4. Limiting all summed area table entries to template size by following:

\[
I^2_{f_1}(x, y) = + I_{f_1}(x - 1, y - 1) \\
+ I_{f_1}(x + X - 1, y + Y - 1) \\
- I_{f_1}(x + X - 1, y - 1) \\
- I_{f_1}(x - 1, y + Y - 1)
\]

\[
I^2_{f_2}(x, y) = + I^2_f(x - 1, y - 1) \\
+ I^2_f(x + X - 1, y + Y - 1) \\
- I^2_f(x + X - 1, y - 1) \\
- I^2_f(x - 1, y + Y - 1)
\]

where \( X \) and \( Y \) are the width and height of the template, respectively.
Figure 4. Correlation heat map and feature detection of a fabric sample with plain weave. a) correlated image result found by (6). b) Detected NPs within the image.

5. Transforming the input image into frequency domain using $DFT$, followed by a pointwise multiplication with the template spectrum and inverse Fourier transformation.

6. Pointwise calculation of the denominator according to (5) by

$$d(x, y) = \sqrt{I_f^2(x, y) - I_{f2}^2(x, y)} \cdot \sqrt{A \cdot t_\sigma}$$

7. Calculation of the final correlation result by pointwise division

$$f_{xcorr}(x, y) = \frac{n(x, y)}{d(x, y)}$$

8. Performing Non-Maximum-Suppression (NMS) on the correlation result according to (7, 8)

9. Storing the coordinates of non-zero positions in the NMS filtered image for further processing in a node point list $L_{NP}$.

Steps 1-9 can directly be computed on the GPU which allow a very efficient implementation for the feature point detection. Experiments showed that the NP detection still works with sufficient precision when executed images scaled to as low as 60% of the original image size (combined with upward interpolation of the feature point coordinates). This characteristic has been employed in the system design. Table 1 summarizes the time consumption of the discussed feature point detection framework. Source code for the GPU implementation is available on the project website.

4. YARN MAPPING

Given the locations of node points within the image in $L_{NP}$, the next step aims to organize them in a compact matrix form in which each row (column) represents a single weft (warp) in the fabric. Each row (column) of the matrix then contains the $L_{NP}$ indices of all NPs that belong to a specific weft (warp). Figure 5.c illustrates the concept.
Figure 5. a) Illustration of distributed node points in a fabric image. b) Corresponding intermediate yarn matrix $Y_i$. c) Final yarn matrix $Y$.

The process is based on the input of two vector pairs: Two yarn vectors $\{v_{\text{weft}}, v_{\text{warp}}\}$ indicate the approximate spacing between adjacent wefts and warps within the image. The information can be computed automatically by frequency space analysis of defect free reference images. Two additional neighbourhood vectors $\{n_1, n_2\}$ indicate the averaged displacement from one node point to its two closest, adjacent node points (cf. Figure 5). The displacement information is closely linked to the pickage and the binding of the investigated material. The vector pair can be assigned manually by the operator or better automatically from defect free reference images.\(^{11}\)

The node point list $L_{NP}$ and the neighbourhood vector pair $\{n_1, n_2\}$ are used to create a yarn matrix $Y$ containing information about node point neighbourhood relationships. The algorithm is based on a region growing process that links positions in the near regular fabric lattice to node points in $L_{NP}$.

At first, several initialization points (seeds, i.e. NPs where the subsequent mapping process may start its spreading from) are determined and marked in $L_{NP}$. Typically, an initialization grid of $4 \times 4$ is chosen.

The following steps take place during the first iteration:

1. A search Matrix $M$, an intermediate yarn matrix $Y_i$ and a query point queue $QP$ are initialized.
2. The NP-coordinates from $L_{NP}$ are mapped into a kd-Tree structure.\(^ {12} \)
3. A first seed point $s$ with image coordinates $s_{x,y}$ is chosen from $L_{NP}$.
4. The center position of matrix $M$ with coordinates $m_{0x,y}$ is marked as *found*.
5. The corresponding position in $Y_i$ is filled with the list index of $s$ in $L_{NP}$.
6. The seed point $s$ is unmarked as seed in $L_{NP}$.
7. In matrix $M$, the four vertically and horizontally adjacent positions to $m_{0x,y}$ (namely $m_{0x+1,y}$, $m_{0x,y+1}$, $m_{0x-1,y}$, $m_{0x,y-1}$) are marked as *listed*.
8. Four corresponding, potential NP-coordinates are derived from $s$ by adding the vectors $\{n_1, n_2, -n_1, -n_2\}$ to $s_{x,y}$ respectively.
9. The new potential NP-coordinates are appended to the query point queue $QP$ together with their corresponding coordinates in matrix $M$.

When the first iteration finishes, the following steps take place for every subsequent iteration:

10. The first potential NP-location is removed from the queue $QP$ and is used as seek point $a$.
11. Using the already initialized kd-Tree,\(^ {12} \) the closest NP $f$ to the seek point $a$ is efficiently found.
12. If \( f \) resides within certain radius around \( a \), its corresponding position in \( M \) is marked as *found* and its index in \( L_{NP} \) is saved in \( Y_i \).

13. As above, steps 7-9 apply. For steps 7 and 8, positions already marked as *listed* are ignored.

If \( f \) doesn’t reside inside the predefined radius, its position in \( M \) is marked as *gap*, the current iteration aborts and a new iteration starts by removing another seek point from the query queue. As soon as the query queue is empty, the next non-processed seed point is found in \( L_{NP} \), and the procedure restarts from the very beginning at step 1. If all seed points are unmarked and \( QP \) is empty, the system finishes. An example of a resulting yarn matrix \( Y_i \) is illustrated in Figure 5.b.

The intermediate yarn matrix \( Y_i \) is now transformed into the final yarn matrix. For this purpose, a transformation matrix \( T \) is computed. The coordinates \( \{x, y\}, \{x_n, y_n\}, \{x_v, y_v\} \) of a NP within the image-, yarn-vector- and neighbourhood-coordinate system can be expressed as

\[
\begin{bmatrix}
  x \\
  y
\end{bmatrix} =
\begin{bmatrix}
  n_1 & n_2
\end{bmatrix} \cdot
\begin{bmatrix}
  x_n \\
  y_n
\end{bmatrix} =
\begin{bmatrix}
  v_{weft} & v_{warp}
\end{bmatrix} \cdot
\begin{bmatrix}
  x_v \\
  y_v
\end{bmatrix}.
\]

(8)

Rearranging yields the transformation matrix \( T \)

\[
\begin{bmatrix}
  x_v \\
  y_v
\end{bmatrix} = T \cdot
\begin{bmatrix}
  x_n \\
  y_n
\end{bmatrix},
\]

with

\[
\begin{bmatrix}
  v_{weft} & v_{warp}
\end{bmatrix}^{-1} \cdot
\begin{bmatrix}
  n_1 & n_2
\end{bmatrix}.
\]

The final yarn map \( Y \) is found by mapping each entry \((x, y) \ (\forall x \in \{1, ..., O\}), \forall y \in \{1, ..., P\}\), \( O, P \) matrix width and height) in \( Y_i \) onto \( Y \) using \( T \), according to

\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix} = round(T \cdot
\begin{bmatrix}
  x \\
  y
\end{bmatrix})
\]

(9)

\[
Y(x', y') = Y_i(x, y).
\]

(10)

### 4.1 Yarn trajectory interpolation

Each row in the yarn matrix is a list of NPs that reside on a single weft. The same way, each column contains NPs for a specific warp. To get the trajectory of each yarn (i.e. the coordinates of all center pixels that reside on that yarn) linear interpolation between adjacent column/row node points is executed. By calculating the slope of the line between two NPs and interpolating all pixels that reside on that line, the yarn trajectories can be found. All calculations of the yarn mapping block are performed on the CPU by distributing calculation tasks among the available cores. Table 1 lists the timing results.

![Figure 6. Illustration of the yarn trajectory interpolation concept.](image)

Figure 6. Illustration of the yarn trajectory interpolation concept. a) Only node points (green) are known with certitude. Between adjacent node points, yarn center position are interpolated using the slope of the line that links adjacent NPs (red lines). b) Results of detected yarn center pixels.

It should be stated that the resulting trajectories are estimations, since only the positions of node points are known with certitude. The interpolation is based on the assumption that yarn directions won’t abruptly change between two adjacent NPs. The premise holds since the spacial distance between two adjacent yarns amounts to a few millimeters only. Experiments proved the robustness and accuracy of the proposed framework as presented in the next section.
5. RESULTS

Within the scope of a large scale industrial research project for on-line fabric quality control, the proposed algorithmic framework has been integrated into a Gamma 8R rapier weaving machine from Picanol NV. The image processing hardware consists of a 17-950 Intel Processor with 4 cores and hyper threading, 8 GB RAM and a NVIDIA GTX 570 graphic card. A JAI BM-500GE monochrome camera feeds 5 megapixel (2456x2058) images to the system at a frame rate of 15 fps. The images are taken with a flashing back-light illumination that is installed on a rail below the material and is synchronized to camera sled and trigger.

The system has been evaluated on-line with several different materials, bindings and pickages. For all investigated bindings (plain, twill and satin weave) and pickages (up to 30 yarns/cm), the system proved to be very robust and precise. Yarns could be tracked in defect free and defective images in real-time. It is difficult to provide a quantitative benchmark for the algorithm performance, since the process of labeling reference images is very time consuming regarding the average amount of yarns within a typical fabric image (≥250). Instead, on the project website, video footage and result images are provided to demonstrate the working system. It should be highlighted that the introduced feature point/interpolation concept enables the proposed framework to reliably track wefts and warps, even though large parts of the yarns are not visible on the surface (due to the characteristics of the binding). At the moment the process is limited to non patterned fabrics, Jacquard typed materials are not supported. Table 1 summarizes the computation times for single algorithmic blocks and different pickages. The computation time raises with higher pickages which is caused by the higher amount of NP features that need to be processed. On average, the framework takes 41 ms with deconvolution to track all yarns within the image, already leaving enough time for subsequent sophisticated defect detection algorithms. However, the vision system is not optimal in terms of uniformity and flashing power. By optimizing these hardware parameters, the time consumption for the pre-processing block can be drastically reduced. In fact, it has been demonstrated that by over powering the back light flash for a short period of time, the shutter speed of the camera can be reduced to an amount that removes the need for deconvolution. Moreover, yarn mapping and interpolation units are CPU based at the moment. A future transfer to the GPU will further speed up the system.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>10 yarns/cm</th>
<th>15 yarns/cm</th>
<th>20 yarns/cm</th>
<th>25 yarns/cm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deconvolution</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>CLAHE, 5x5 grid</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Feature Extraction (50% down scale)</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>Yarn Mapping</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Yarn Interpolation</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Σ with deconvolution</td>
<td>40</td>
<td>40</td>
<td>41</td>
<td>43</td>
</tr>
<tr>
<td>Σ without deconvolution</td>
<td>28</td>
<td>28</td>
<td>29</td>
<td>31</td>
</tr>
</tbody>
</table>

Table 1. Timing table for all algorithmic blocks with different weft densities (pickages) using the same material and binding. All units in [ms].

6. CONCLUSION

We introduced a vision based framework for tracking yarns in woven fabrics. A new concept based on high resolution images, feature point extraction, and interpolation allows reliable real-time tracking of single wefts and warps within an fabric image, even though yarns are not completely visible for the camera system. The system is able to process plain, twill and satin weaves and is limited to non patterned materials. All process steps have been described in detail and additional GPU accelerated source code for selected algorithms is provided through the project website. The framework has been validated on a prototype defect detection system mounted to an industrial loom. The results were satisfying and proved to be robust for a variety of tested materials, bindings and pickages (cf. Figure 7-10).
Subsequent framework blocks already analyse provided yarn information to detect defects within the images with high accuracy and low false alarm rate. Future work will expand the proposed ideas to develop a system able to detect the binding of an unknown fabric image fully automatically.

Figure 7. Tracked yarns in a cotton plain weave sample with a loop defect.

Figure 8. Tracked yarns in a cotton satin weave sample with a thick yarn defect

Figure 9. Tracked yarns in a polyester twill weave sample with local lattice defects.
7. ACKNOWLEDGEMENTS

The project "Integratives Messsystem zur Fehlererkennung während der textilen Flächenherstellung", AiF-no. 355 ZN/1, initiated by the research association Textil e. V., Reinhardtstr. 12-14, 10117 Berlin and the FQS-Forschungsgemeinschaft Qualität e. V., August-Schanz-Straße 21A, 60433 Frankfurt am Main, has been promoted by the Federal Ministry of Economics and Technology via the AiF as part of a programme to support the joint industrial research and development.

REFERENCES