K-MEANS CLUSTERING-BASED POROSITY ASSESSMENT

K-MEANS CLUSTERING-BASIERTE POROSITÄTSANALYSE

Vova Lisin

Institute of Construction Materials (IWB), University of Stuttgart

SUMMARY

In analysing concrete samples to determine the composition in general and the porosity in particular, several different methods are used. One of these is scanning electron microscopy (SEM). This article presents a more cost-effective alternative that allows a rudimentary composition analysis of clay samples to be carried out quickly and without expensive equipment by simply using a mobile phone. By employing the so-called k-means clustering algorithm, a cropped photograph of a concrete sample is subjected to a specialized segmentation process, allowing for the detection and categorization of different components within the sample. First, the mathematical basis is briefly presented, followed by the implementation in a program code. Finally, the new application is tested on a specific example, and the analysis results are discussed.

ZUSAMMENFASSUNG

Bei der Analyse von Betonproben zur Bestimmung der Zusammensetzung im Allgemeinen und der Porosität im Speziellen kommen unterschiedliche Verfahren zur Anwendung. Darunter zu nennen ist etwa die sog. Rasterelektronenmikroskopie (REM). Hier vorgestellt wird eine kostengünstigere Alternative, die es erlaubt eine rudimentäre Zusammensetzungsanalyse von Betonproben mithilfe eines Mobiltelephons zügig und ohne teure Gerätschaften zu realisieren. Durch Verwendung des sog. k-means-Clustering-Algorithmus wird ein Photozuschnitt einer Betonprobe einer speziellen Segmentierung zugeführt, wodurch unterschiedlich geartete Bestandteile einer Probe detektiert und nach Kategorie gruppiert werden können. Kurz erörtert wird zunächst der mathematische Unterbau und anschließend die Umsetzung in einem Programmcode. Die neue Anwendung wird schließlich an einem konkreten Beispiel erprobt.

1. PROBLEM DISCUSSION

1.1 Task definition

The figure below shows a picture of a concrete sample prepared with epoxy resin.

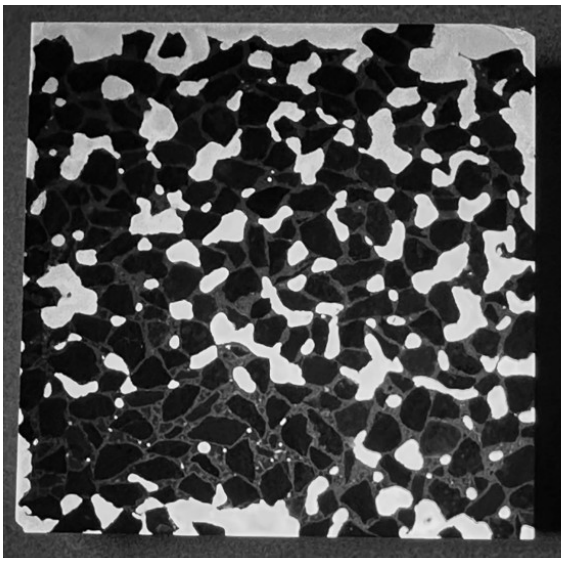


Fig. 1: Concrete sample prepared with epoxy resin (black & white representation)

The task now is to construct a suitable software application for an image-based “express analysis” to determine the structural composition of a concrete sample.

1.2 Theoretical basis

The basis of the implemented analysis environment is the basic algorithm of the so-called k-means clustering method. Clustering is a fundamental principle of machine learning, which aims to divide data points into groups (clusters) without prior labelling. The idea behind it is that data points that are in the same group are in some way ‘similar’, while data points from different groups are ‘dissimilar’. These similarities or dissimilarities can be quantified through a distance function.

$$d(x_i, x_j) \ = \ \sqrt{(L_i - L_j)^2 + (a_i - a_j)^2 + (b_i - b_j)^2} \tag{1}$$

Accordingly, (1) defines a Euclidean distance metric that has to satisfy a minimisation problem for any two points $x_i = x(L_i, a_i, b_i)$ and $x_j = x(L_j, a_j, b_j)$ with:

$$\min_{C} \sum_{i=1}^{n} \sum_{j=1}^{k} \mathbf{1}_{\{x_i \in C_j\}} \left[(L_i - L(c_j))^2 + (a_i - a(c_j))^2 + (b_i - b(c_j))^2 \right] \tag{2}$$

The optimal requirement in (2) equates to an optimal corresponding assignment of n data points (pixels) $x_i = x(L_i, a_i, b_i),\ i \in [1, n] \cap \mathbb{N}$ with respect to the k clusters $C_j = (L_i, a_i, b_i),\ j \in [1, k] \cap \mathbb{N}$, within the so-called CIELAB colour space, where L_i und L_j represent the brightness of the i-th pixel and the j-th cluster center, respectively. The colour channels a_i and a_j, as well as $b_i$und b_j, represent the colour channels along the green-red axis and the blue-yellow axis, respectively. A j -th cluster center c_j, also known as a centroid, represents the average colour of all $|C_j|$ pixels x_i that are assigned to the cluster C_j and is expressed by:

$$c_j = \left(\frac{1}{|C_j|}\sum_{x_i \in C_j} L_i,\ \ \frac{1}{|C_j|}\sum_{x_i \in C_j} a_i,\ \ \frac{1}{|C_j|}\sum_{x_i \in C_j} b_i\right) \tag{3}$$

Finally, $\mathbf{1}_{\{x_i \in C_j\}}$ operates like an affiliation test and is called an indicator function. Now, if an i-th data point x_i with $x_i \in C_j$ belongs to the j-th Cluster C_j, then $\mathbf{1}_{\{x_i \in C_j\}} \equiv 1$ applies. However, if the respective pixel with $x_i \notin C_j$ is not a cluster element of C_j, then correspondingly $\mathbf{1}_{\{x_i \in C_j\}} \equiv 0$ will be assigned. In other words, this operator ensures that only the distances between the points and the centres of the respective clusters to which they belong are calculated and minimised by (2).

On principle, the k-means algorithm is a heuristic search method that implies iterative optimisation. The algorithm is a type of greedy algorithm that makes the best local decision in each iteration (namely assigning the points to the nearest cluster centre), but there is no guarantee a global best solution will be achieved.

The heuristic steps:

1. Initialisation: The algorithm starts with a random selection of cluster centers. This starting point strongly influences the result, as the algorithm converges to a local minimum from this point.

2. Assignment: Each point is assigned to the cluster with the nearest center of the centroid.

3. Recalculation: The cluster centers are updated as the mean values of the points in the cluster.

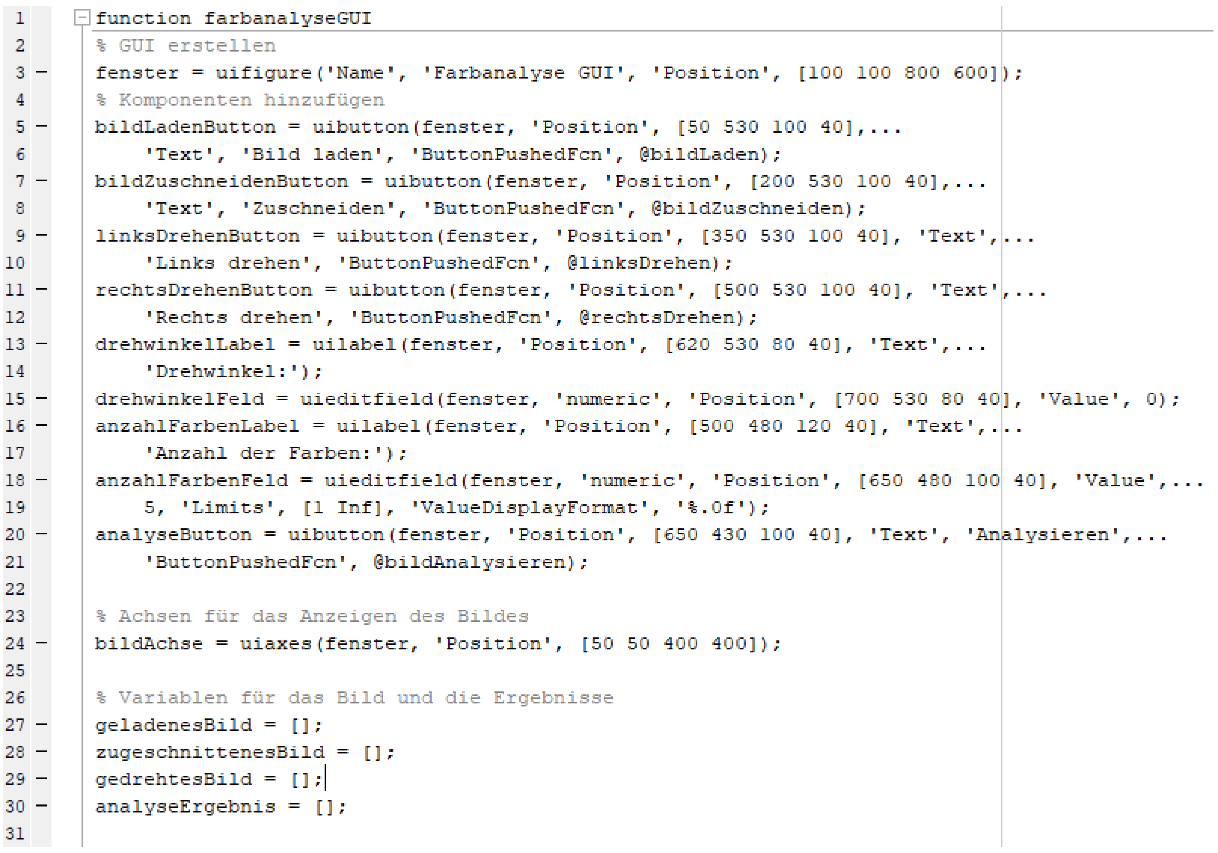
4. Repetition: The process is repeated until the assignments no longer change or a cancellation criterion (e.g. a maximum number of iterations) is reached.

After all, the aim of the k-means algorithm is to compute an optimal segmetisation mapping to the partition $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$ of the pixels of a given set of n data points $\mathcal{X} = \{x_1, x_2, \dots, x_n\}$ that minimises the sum of the squared distances from each point to the centroid of its associated cluster, according to the following rule:

$$\arg \min_{\{C_1, C_2, \dots, C_k\}} \sum_{i=1}^{k} \sum_{x \in C_i} ||\mathrm{x} - c_i||^2 \qquad (4)$$

2. PROGRAM CODE IMPLEMENTATION EXAMPLE

The implementation in a program CODE and the development of an application is simple and can be realised in any language at high interpreter level without major problems. The following MATLAB CODE illustrates the basic procedure.

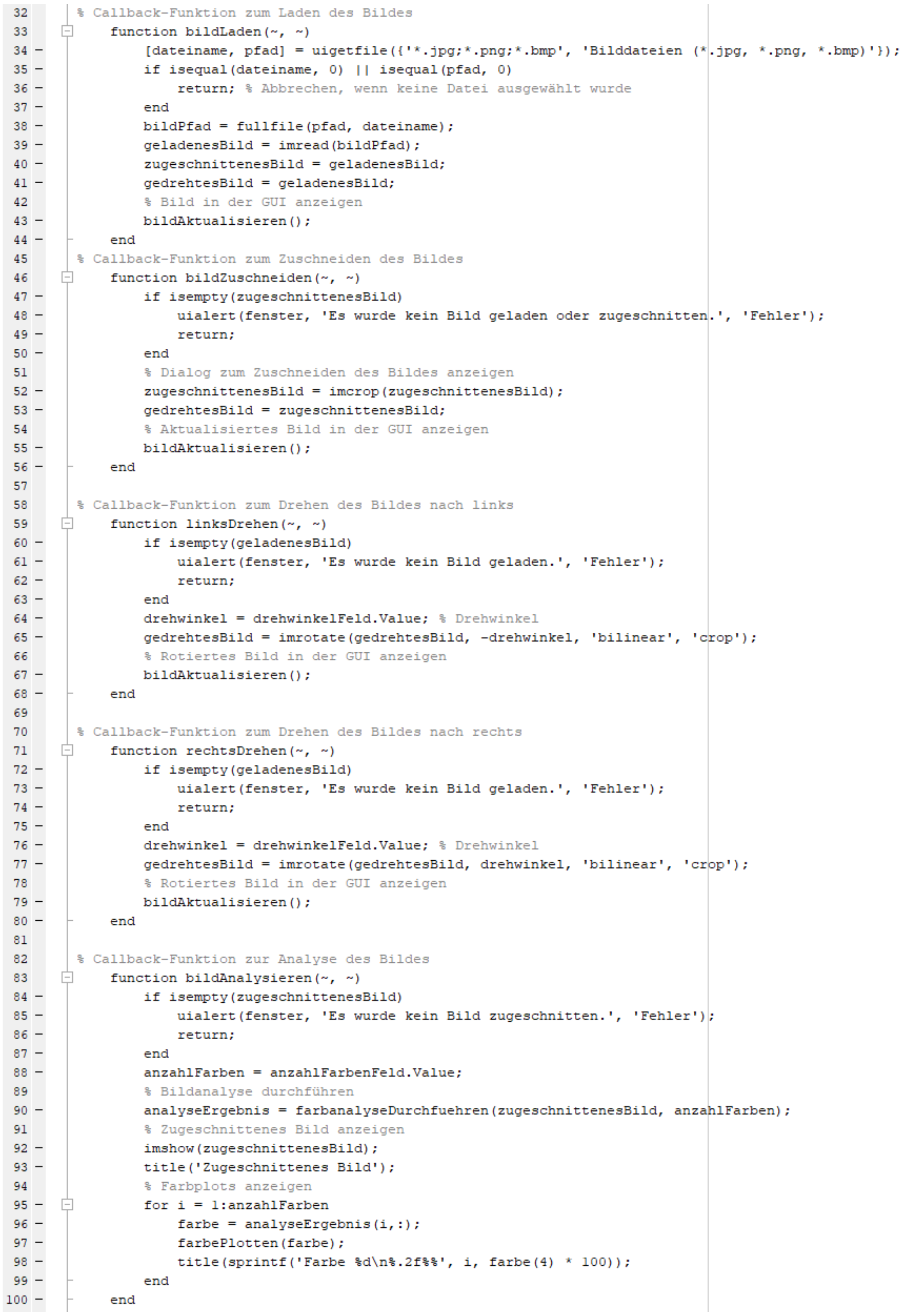


```matlab
function farbanalyseGUI
% GUI erstellen
fenster = uifigure('Name', 'Farbanalyse GUI', 'Position', [100 100 800 600]);
% Komponenten hinzufügen
bildLadenButton = uibutton(fenster, 'Position', [50 530 100 40],...
    'Text', 'Bild laden', 'ButtonPushedFcn', @bildLaden);
bildZuschneidenButton = uibutton(fenster, 'Position', [200 530 100 40],...
    'Text', 'Zuschneiden', 'ButtonPushedFcn', @bildZuschneiden);
linksDrehenButton = uibutton(fenster, 'Position', [350 530 100 40], 'Text',...
    'Links drehen', 'ButtonPushedFcn', @linksDrehen);
rechtsDrehenButton = uibutton(fenster, 'Position', [500 530 100 40], 'Text',...
    'Rechts drehen', 'ButtonPushedFcn', @rechtsDrehen);
drehwinkelLabel = uilabel(fenster, 'Position', [620 530 80 40], 'Text',...
    'Drehwinkel:');
drehwinkelFeld = uieditfield(fenster, 'numeric', 'Position', [700 530 80 40], 'Value', 0);
anzahlFarbenLabel = uilabel(fenster, 'Position', [500 480 120 40], 'Text',...
    'Anzahl der Farben:');
anzahlFarbenFeld = uieditfield(fenster, 'numeric', 'Position', [650 480 100 40], 'Value',...
    5, 'Limits', [1 Inf], 'ValueDisplayFormat', '%.0f');
analyseButton = uibutton(fenster, 'Position', [650 430 100 40], 'Text', 'Analysieren',...
    'ButtonPushedFcn', @bildAnalysieren);

% Achsen für das Anzeigen des Bildes
bildAchse = uiaxes(fenster, 'Position', [50 50 400 400]);

% Variablen für das Bild und die Ergebnisse
geladenesBild = [];
zugeschnittenesBild = [];
gedrehtesBild = [];
analyseErgebnis = [];

```

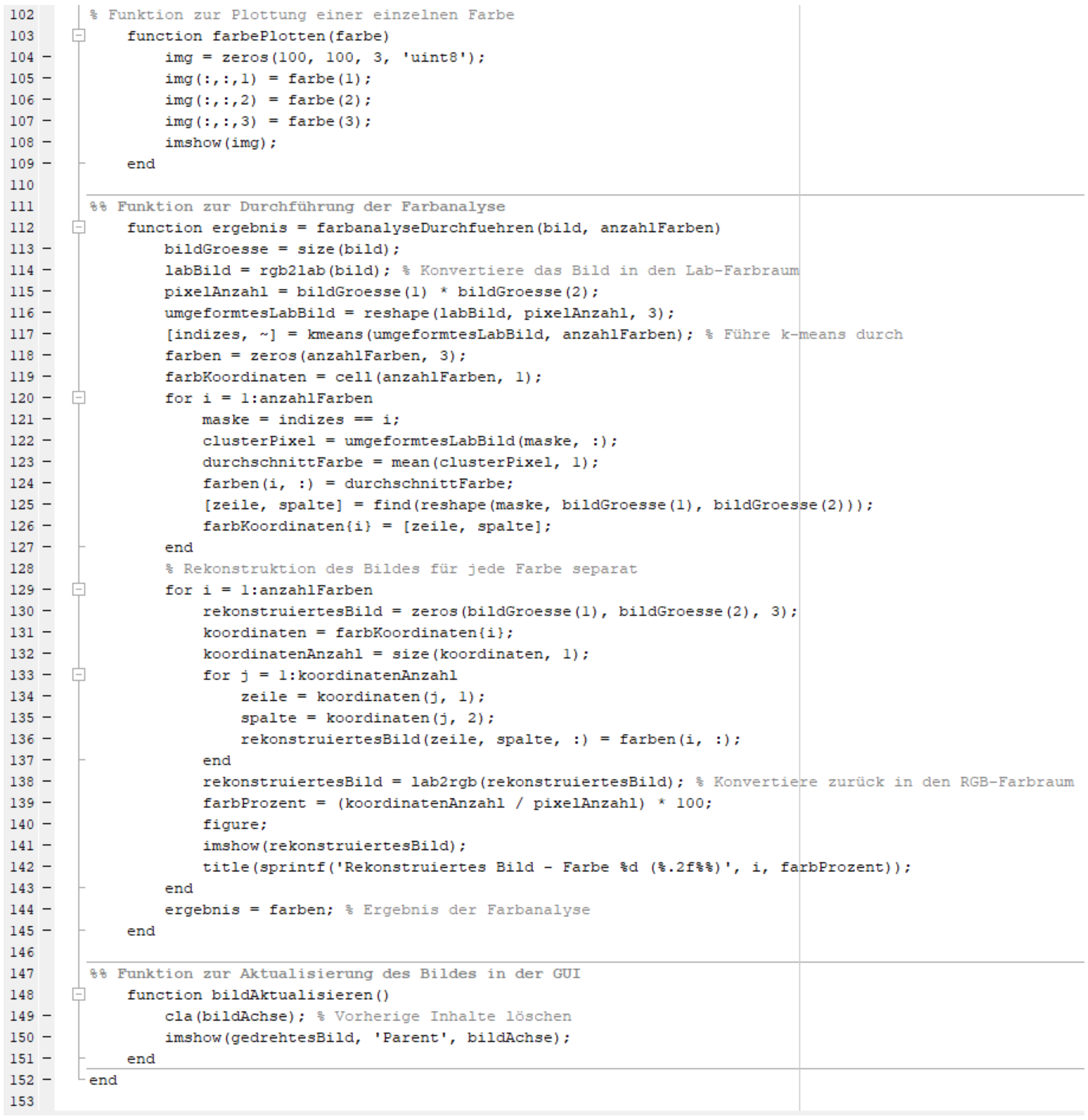


```
% Callback-Funktion zum Laden des Bildes
    function bildLaden(~, ~)
        [dateiname, pfad] = uigetfile({'*.jpg;*.png;*.bmp', 'Bilddateien (*.jpg, *.png, *.bmp)'});
        if isequal(dateiname, 0) || isequal(pfad, 0)
            return; % Abbrechen, wenn keine Datei ausgewählt wurde
        end
        bildPfad = fullfile(pfad, dateiname);
        geladenesBild = imread(bildPfad);
        zugeschnittenesBild = geladenesBild;
        gedrehtesBild = geladenesBild;
        % Bild in der GUI anzeigen
        bildAktualisieren();
    end
% Callback-Funktion zum Zuschneiden des Bildes
    function bildZuschneiden(~, ~)
        if isempty(zugeschnittenesBild)
            uialert(fenster, 'Es wurde kein Bild geladen oder zugeschnitten.', 'Fehler');
            return;
        end
        % Dialog zum Zuschneiden des Bildes anzeigen
        zugeschnittenesBild = imcrop(zugeschnittenesBild);
        gedrehtesBild = zugeschnittenesBild;
        % Aktualisiertes Bild in der GUI anzeigen
        bildAktualisieren();
    end

% Callback-Funktion zum Drehen des Bildes nach links
    function linksDrehen(~, ~)
        if isempty(geladenesBild)
            uialert(fenster, 'Es wurde kein Bild geladen.', 'Fehler');
            return;
        end
        drehwinkel = drehwinkelFeld.Value; % Drehwinkel
        gedrehtesBild = imrotate(gedrehtesBild, -drehwinkel, 'bilinear', 'crop');
        % Rotiertes Bild in der GUI anzeigen
        bildAktualisieren();
    end

% Callback-Funktion zum Drehen des Bildes nach rechts
    function rechtsDrehen(~, ~)
        if isempty(geladenesBild)
            uialert(fenster, 'Es wurde kein Bild geladen.', 'Fehler');
            return;
        end
        drehwinkel = drehwinkelFeld.Value; % Drehwinkel
        gedrehtesBild = imrotate(gedrehtesBild, drehwinkel, 'bilinear', 'crop');
        % Rotiertes Bild in der GUI anzeigen
        bildAktualisieren();
    end

% Callback-Funktion zur Analyse des Bildes
    function bildAnalysieren(~, ~)
        if isempty(zugeschnittenesBild)
            uialert(fenster, 'Es wurde kein Bild zugeschnitten.', 'Fehler');
            return;
        end
        anzahlFarben = anzahlFarbenFeld.Value;
        % Bildanalyse durchführen
        analyseErgebnis = farbanalyseDurchfuehren(zugeschnittenesBild, anzahlFarben);
        % Zugeschnittenes Bild anzeigen
        imshow(zugeschnittenesBild);
        title('Zugeschnittenes Bild');
        % Farbplots anzeigen
        for i = 1:anzahlFarben
            farbe = analyseErgebnis(i,:);
            farbePlotten(farbe);
            title(sprintf('Farbe %d\n%.2f%%', i, farbe(4) * 100));
        end
    end
```



```matlab
% Funktion zur Plottung einer einzelnen Farbe
    function farbePlotten(farbe)
        img = zeros(100, 100, 3, 'uint8');
        img(:,:,1) = farbe(1);
        img(:,:,2) = farbe(2);
        img(:,:,3) = farbe(3);
        imshow(img);
    end

%% Funktion zur Durchführung der Farbanalyse
    function ergebnis = farbanalyseDurchfuehren(bild, anzahlFarben)
        bildGroesse = size(bild);
        labBild = rgb2lab(bild); % Konvertiere das Bild in den Lab-Farbraum
        pixelAnzahl = bildGroesse(1) * bildGroesse(2);
        umgeformtesLabBild = reshape(labBild, pixelAnzahl, 3);
        [indizes, ~] = kmeans(umgeformtesLabBild, anzahlFarben); % Führe k-means durch
        farben = zeros(anzahlFarben, 3);
        farbKoordinaten = cell(anzahlFarben, 1);
        for i = 1:anzahlFarben
            maske = indizes == i;
            clusterPixel = umgeformtesLabBild(maske, :);
            durchschnittFarbe = mean(clusterPixel, 1);
            farben(i, :) = durchschnittFarbe;
            [zeile, spalte] = find(reshape(maske, bildGroesse(1), bildGroesse(2)));
            farbKoordinaten{i} = [zeile, spalte];
        end
        % Rekonstruktion des Bildes für jede Farbe separat
        for i = 1:anzahlFarben
            rekonstruiertesBild = zeros(bildGroesse(1), bildGroesse(2), 3);
            koordinaten = farbKoordinaten{i};
            koordinatenAnzahl = size(koordinaten, 1);
            for j = 1:koordinatenAnzahl
                zeile = koordinaten(j, 1);
                spalte = koordinaten(j, 2);
                rekonstruiertesBild(zeile, spalte, :) = farben(i, :);
            end
            rekonstruiertesBild = lab2rgb(rekonstruiertesBild); % Konvertiere zurück in den RGB-Farbraum
            farbProzent = (koordinatenAnzahl / pixelAnzahl) * 100;
            figure;
            imshow(rekonstruiertesBild);
            title(sprintf('Rekonstruiertes Bild - Farbe %d (%.2f%%)', i, farbProzent));
        end
        ergebnis = farben; % Ergebnis der Farbanalyse
    end

%% Funktion zur Aktualisierung des Bildes in der GUI
    function bildAktualisieren()
        cla(bildAchse); % Vorherige Inhalte löschen
        imshow(gedrehtesBild, 'Parent', bildAchse);
    end
end

```

This attached script is structured in an object-oriented manner and generated a very rudimentary designed application. It allows the loading and cropping an image file to be analysed. As soon as the cutting is complete, the user can define the desired number of clusters and in this way specify the degree of depth of the intended segmentation. In this instance, it is recommendable to set up the app several times in order to determine the number of clusters that are required by testing.

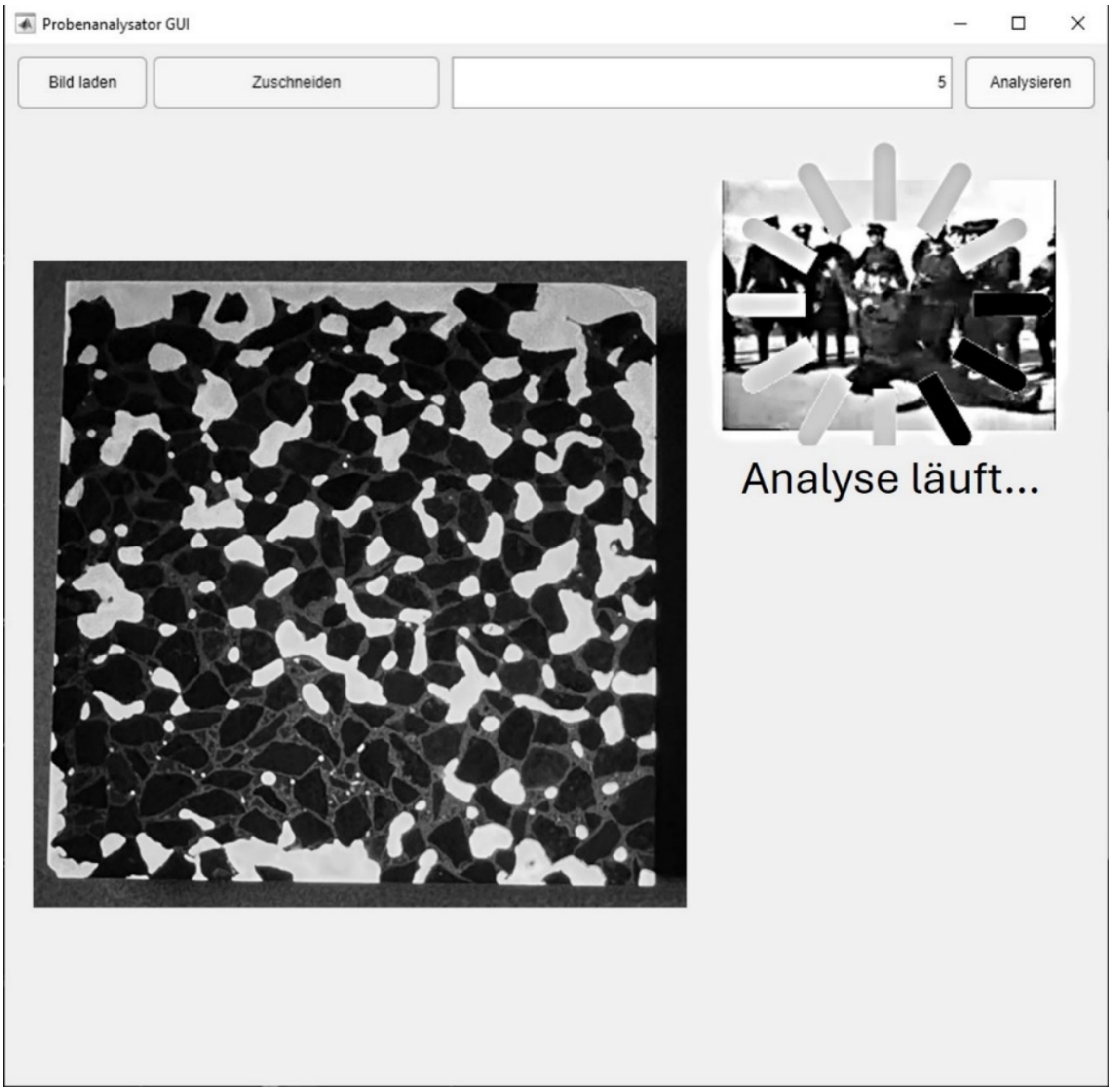


Fig. 2: Layout of the new application

3. RESULTS AND DISCUSSION

3.1 Case distinction

3.1.1 $k = 2$ - Porosity Assessment

If the number of clusters is set to two, the porosity of the concrete sample is determined directly, as the k-means algorithm in this case divides the pixels into two main categories: the pores filled with epoxy resin and the actual concrete material. By clearly segmenting into these two clusters, the number of pixels representing the epoxy area relative to the total number of pixels in the sample can be used as a measure of porosity. The percentage coverage of the cluster comprising the epoxy-filled pores thus corresponds directly to the porous portion of the sample.

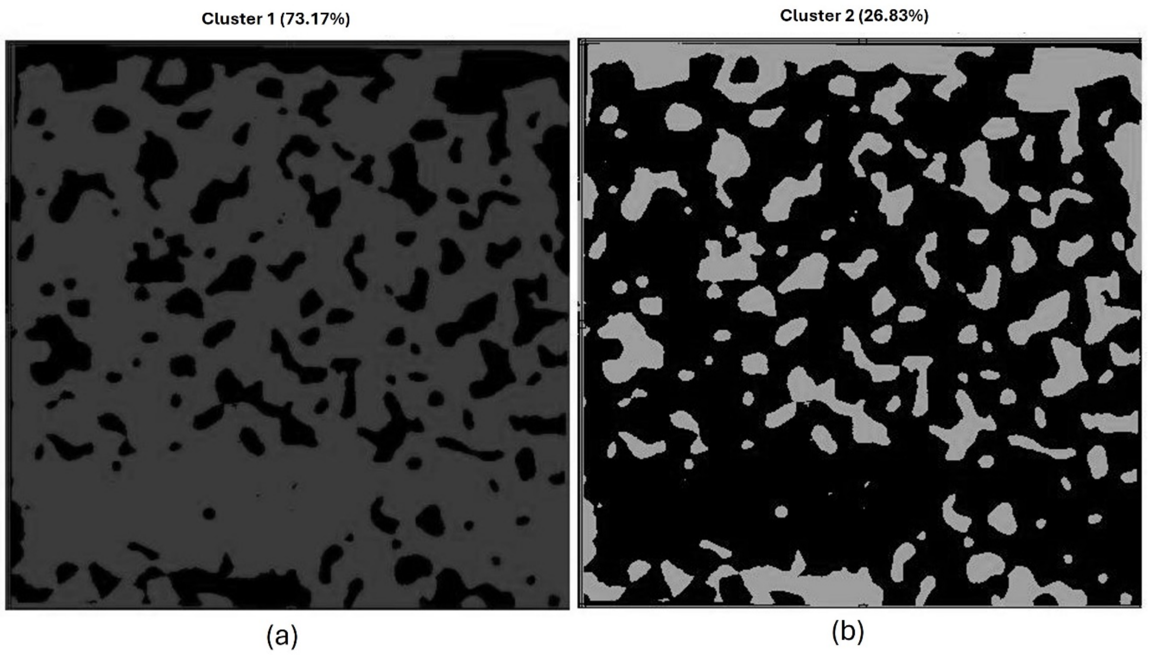


Fig. 3: (a) Concrete - 73.17 %, (b) Epoxy-filled voids (detected pores) - 28.83 %

3.1.2 $k > 2$ - Refined segmentation

In the case that the number of clusters is set to more than two, a refined segmentation is performed. In this case, the k-means algorithm not only distinguishes between the pores filled with epoxy resin and the concrete matrix, but also identifies additional substructures within the sample. For instance, different aggregates, cement phases or microcracks within the concrete can be subdivided into different clusters. This finer segmentation enables a more detailed analysis of the material composition and the distribution of various elements within the sample.

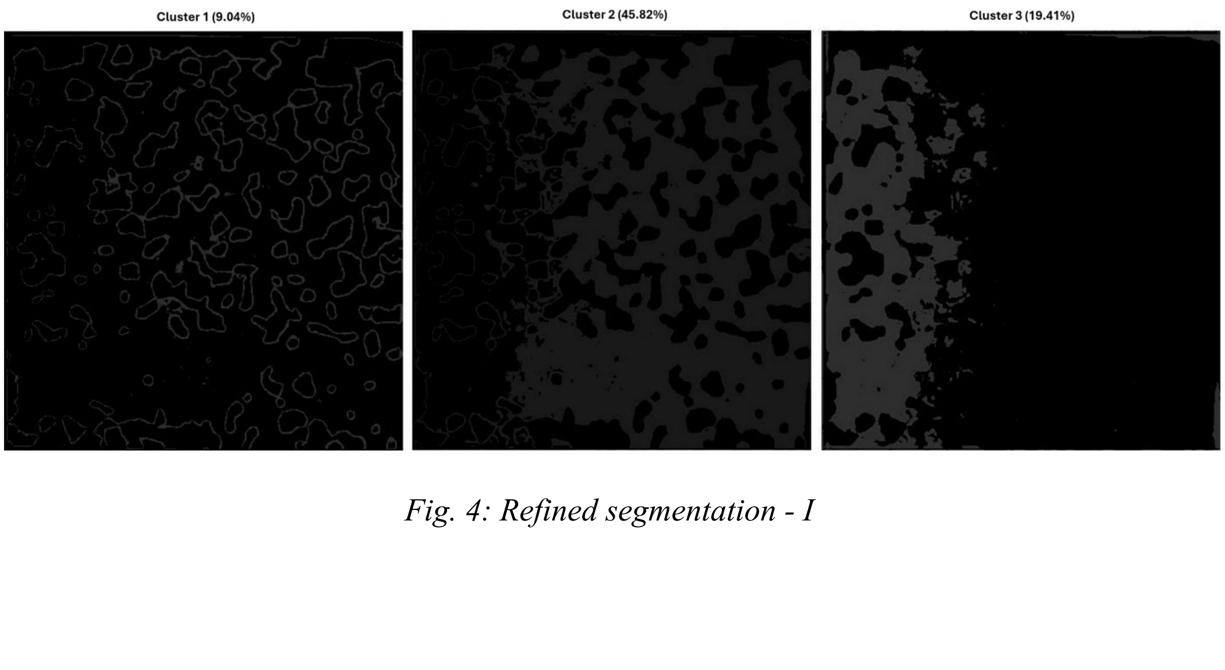


Fig. 4: Refined segmentation - I

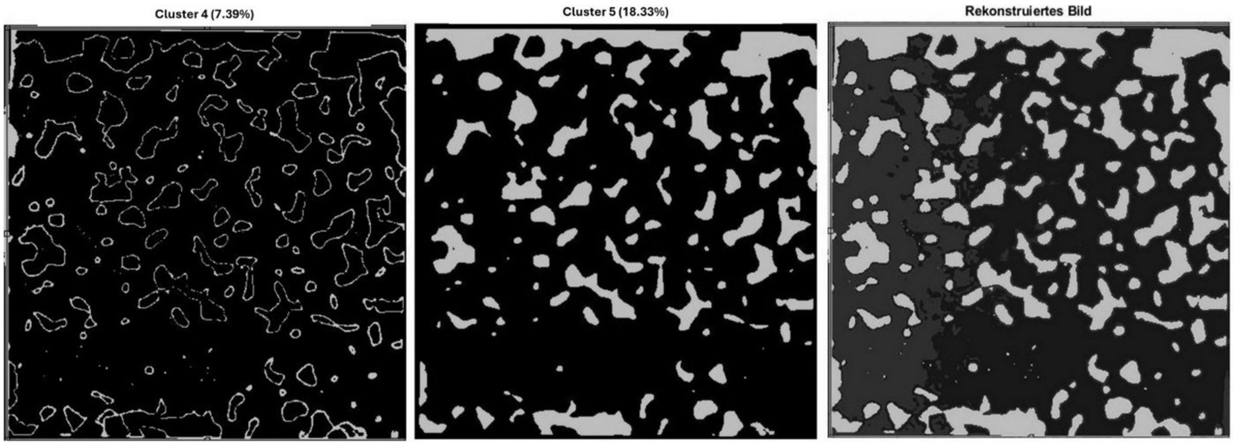


Fig. 5: Refined segmentation - II

4. CONCLUSION AND OUTLOOK

Based on the present review, the results demonstrate the practical utility of the k-means cluster algorithm in assessing the porosity of concrete samples made with epoxy resin. By segmenting the sample into clusters representing the concrete matrix and the epoxy-filled voids, the algorithm provides a direct and efficient method for calculating porosity. This approach offers a more accessible and cost-effective alternative to conventional methods such as scanning electron microscopy (SEM) or X-ray fluorescence spectroscopy, which often require complex and expensive equipment. Two clusters (k = 2) are particularly effective in determining porosity. The clear distinction between the concrete material and the pores filled with epoxy resin allows an accurate assessment of the porous areas within the sample. The direct relationship between the pixel count in the epoxy cluster and the total pixel count allows a straightforward calculation of porosity and makes the method a practical tool for fast, non-invasive analysis. By increasing the number of clusters (k > 2), a finer segmentation can be achieved. This enables the identification of additional substructures within the concrete, such as different aggregate types, cement phases and potential microcracks. These capabilities allow for a more detailed analysis of the composition of concrete samples, providing valuable insights into the distribution and interaction of different materials within the concrete matrix. The ability to perform such detailed analyses without specialised, costly equipment underlines the practical advantages of the approach.

While the current study relies on the use of epoxy resin to fill the pores, future work could investigate the possibility of extending this image-based segmentation technique to analyse concrete samples without the use of epoxy resin. By refining the clustering algorithm or incorporating additional machine learning techniques, it might be possible to recognise and classify voids and microstructures in the concrete matrix directly based on their optical features. In this respect, there are several approaches that could be pursued further.

1. **Improved image pre-processing**: future studies could explore advanced image pre-processing techniques to increase the contrast between different regions of the sample, enabling the differentiation between voids, aggregates and other structures without the need for epoxy filling. Techniques such as contrast enhancement, edge detection or the application of different illumination conditions during image acquisition may be beneficial as well.
2. **Algorithm Refinement**: A further improvement of the k-means algorithm or the application of other clustering methods such as hierarchical clustering or Gaussian mixture models (GMM) could improve the accuracy of segmentation in samples without epoxy. Such methods may contribute to distinguish between microstructural features based on their optical characteristics alone, such as differences in colour, the texture or their brightness
3. **Machine learning and deep learning approaches:** Deep learning machine models, and particularly those using convolutional neural networks (CNNs), have been shown to be promising in segmenting and analysing complex images. By training such models on a large dataset of known composition concrete samples, automatic detection and classification of pores, aggregates and other structural features could be possible without the need for epoxy resin. These methods could potentially offer even greater accuracy and more operational agility than conventional clustering techniques.

Apart from that, this brief excursus is an illustration of how problems can be handled with elementary means and common engineer's sense. Work smart - not hard!