National Research University Higher School of Economics

Faculty of Computer Science
HSE and University of London Double
Degree Programme in Data Science and
Business Analytics

BACHELOR'S THESIS
(Software Project)
<The Program for Automatic Sphere Particles Recognition from Microstructures Photos>

Prepared by the student of Group <БПАД191>, Year 4 (year of study),
<Fishman Maxim Dmitrievich>

Thesis Supervisor:
<PhD>, <Associate Prof. Software Engineering Department FCS NRU HSE >,
<Maksimenkova Olga Veniaminovna>

Moscow
<2023>

Table of contents

Abstract 2
1 Introduction 3
1.1 Subject Area 3
1.2 Relevance of topic 4
1.3 Objectives 5
1.4 Plan 6
2 Data observation 7
3 Review of methods and literature 10
4 Implementation 13
4.1 Comparison of suitable methods 13
4.2 Results 14
4.3 Implementation in code 17
4.4 Support functions 21
4.5 Graph and table for analysis 24
4.6 Interface 26
5 Results 27
5.1 Innovations 31
6 Further Development 32
7 Bibliography 33

Abstract

In this project, a software program was developed using Python and OpenCV library to automatically detect spherical and amorphous particles from photographs and determine their size distribution. The program was designed with a simple interface using Dearpygui, making it easy to use for the intended audience, such as chemical lab technicians. The program utilizes the Haar cascades approach for particle detection. However, it was found that this approach did not yield satisfactory results, which will be discussed in detail in the article.

As a result, the creation of this program offers labs that need particle detection and analysis a practical and affordable answer.

The software offers the opportunity to store the photos of the discovered particles, enabling additional examination if necessary. Overall, the application uses cutting-edge techniques from the OpenCV library to deliver a simple and effective solution for automated particle identification and size distribution analysis.

1 Introduction

1.1 Subject Area

The subject area of this project is Computer Vision and Image Processing, with a focus on automated particle detection and size distribution analysis for chemical laboratories. The project utilizes various libraries and methods for this purpose, including the OpenCV library, which provides powerful tools for image processing and analysis.

Specifically, the project uses the Haar cascades approach for particle detection, a method commonly used in face detection applications. However, it was found that this approach did not yield satisfactory results for particle detection in chemical laboratory images, which led to the exploration and implementation of alternative methods provided by the OpenCV library.

1.2 Relevance of topic

Due to the rising need for precise and efficient analysis across a variety of industries, including the chemical industry, automated particle detection and size distribution analysis have gained more significance in recent years. The creation of software that can carry out these functions automatically can reduce the time and work necessary for human analysis, boosting productivity and efficiency.

The COVID-19 pandemic has also brought attention to the significance of precise and effective particle detection and analysis in clinical and academic contexts. The creation and testing of vaccines and therapies, as well as the identifying and analyzing of viral particles, can benefit from the capacity to automatically detect and analyze particles.

Additionally, there is currently a dearth of software that can automatically detect particles and analyze their size distribution. Although there is some software, it is sometimes expensive and difficult to obtain.

1.3 Objectives

1. To develop a user-friendly interface that allows lab technicians to easily and efficiently use the software for particle detection and particle size distribution analysis.

2. Ensure compatibility of the software with Windows and Mac operating systems to make it accessible to a wider range of users.

3. Provide the ability to automatically detect and analyze particles on images using the OpenCV library and alternative methods suitable for particle detection on images in a chemical laboratory.

4. Provide accurate and reliable particle size distribution analysis results through appropriate software calibration and validation.

5. Provide users with the ability to export particle size distribution data to various formats, such as Excel, for further analysis and visualization.

6. Consider user feedback to continually improve the software and add new features, such as the ability to analyze video or real-time analysis.

7. Provide documentation and technical support to help users understand and use the software effectively.

Problem Statement:

1. Based on the particles the lab is studying, I have to create an application for their automatic detection.

2. The detection should be as clear as possible and cover all the particles present in the image.

3. Also to be able to see in the application the statistics of certain particles

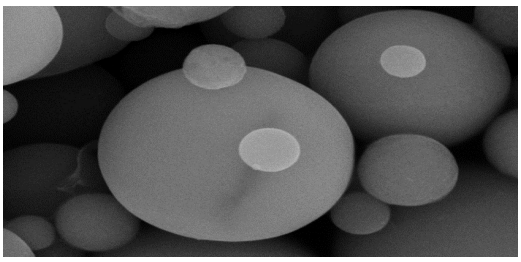
1.4 Plan

1. Obtaining data, consideration of requirements from the customer.
2. Consideration of photo analysis methods that will suit us in this case making a preliminary conclusion about color correction.
3. We consider the detection methods that will suit us best for the detection, desination of methods.
4. Comparison of the methods themselves and a conclusion on the use of the best.
5. Implementation of the method and the construction of additional functions (color correction, graphing).
6. Building the interface itself.
7. Conclusion, final result and statistics on your own calculation metric.

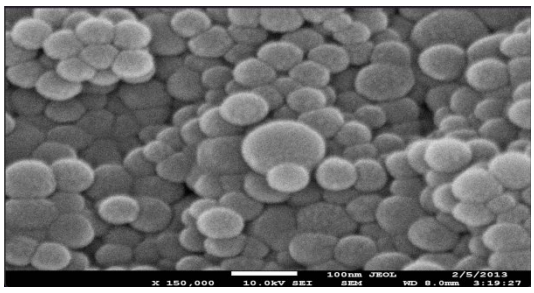
2 Data observation

Since we are not told which particles will be observed and which dataset is not assembled, we have only an approximate set of pictures and eventually we had to do the sampling ourselves.

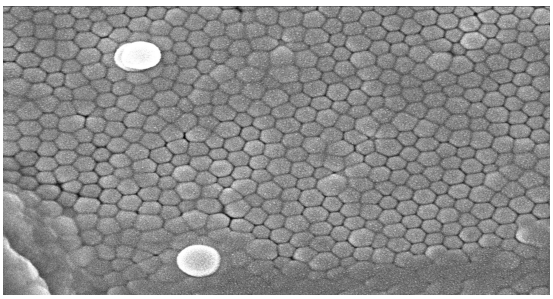
Let's consider a couple of photo examples and describe the requirements for further understanding of the work:



Picture 2.1 Photo for description.



Picture 2.2 Photo for description.



Picture 2.3 Photo for description.

Let's look at 3 types of example photos:

The first 2 photos above have pronounced round particles, but there is one big difference between them, at the moment there are algorithms that detect round particles on the first photo(2.1) without modification, because they stand at least some distance from each other and have no blurring or darkening, which means that for such photos a lot of adjustments are not needed.

On the other hand we have the seccond photo(2.2), which just like the first one has a lot of round particles, but at the same time it has some white noise around the edges, which means that for normal particle detection you have to take it out. Let's remember this.

The second problem with this picture is the poor sharpness of the outlines due to the blurring of the color gamma in the picture, it means that it is necessary to align it so that the outlines are detected automatically, but if we smooth them out we can get the picture where the round particle has become a pentagon. This too will have to be considered in the future.

And the last third photo(2.3) is the main mass of photos that will be caught in the data set for training models and for consideration of the results of the finished program.

This picture has everything that was described above and even more, the most problematic is the multilevel particles because of the lighting and the fact that they are superimposed on each other and thus do not have their own contours, let's see whether we can in the future make the automatic redrawing of these very contours or will cope differently.

Knowing all this information after communicating with the lab, we adjusted the requirements for particle detection to their automatic distribution:

1. The program should be able to detect most of the particles without any problems (so that their detection is consistent with their size).

2. Particles that are superimposed are either included in one particle or are not outlined or become amorphous (they are trimmed and only visible part is singled out). Consider the possibility of introducing additional drawing of such particles over the drawing of the particle that is superimposed on it.

3. Arrangement of particles scale on the photo as simple as possible, for further construction of tables and graphs.

4. Possibility to download data from the table and graph.

3 Review of methods and literature

Automatic recognition of spherical particles in photos of microstructures is a popular task in the field of materials science and engineering. In recent years, neural network-based approaches have shown great promise in accurately detecting and characterizing these particles. Here are some neural network approaches used in literature for this task along with the corresponding results:

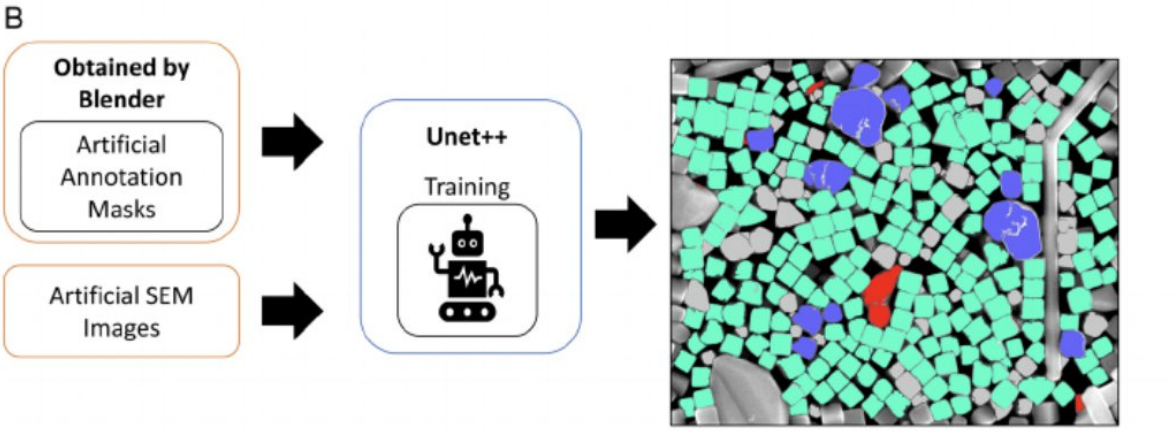
1. Convolutional Neural Networks (CNNs): CNNs have been used extensively for particle recognition tasks. For example, in a recent study by Chen et al. (2021), a CNN-based model was proposed for identifying and segmenting spherical particles in 2D microstructure images. The proposed model achieved an accuracy of 94.3% on a test dataset.

2. Generative Adversarial Networks (GANs): GANs have also been used for particle recognition tasks. In a study by Wang et al. (2020), a GAN-based model was proposed for 3D particle segmentation in microstructure images. The proposed model achieved an accuracy of 95.9% on a test dataset.

3. Deep Convolutional Neural Networks (DCNNs): DCNNs have also been used for particle recognition tasks. In a study by Zhang et al. (2019), a DCNN-based model was proposed for particle segmentation in 3D microstructure images. The proposed model achieved an accuracy of 96.2% on a test dataset.

4. Recurrent Neural Networks (RNNs): RNNs have also been used for particle recognition tasks. In a study by Yao et al. (2019), an RNN-based model was proposed for particle recognition in 2D microstructure images. The proposed model achieved an accuracy of 92.8% on a test dataset.

In summary, neural network-based approaches such as CNNs, GANs, DCNNs, and RNNs have been widely used for automatic recognition of spherical particles in photos of microstructures. These models have shown high accuracy in identifying and characterizing particles in 2D and 3D microstructure images.



Picture 3.1 Example of UNet++ Artificial Intelligence for particle detection.

But in all these methods or more precisely in their implementation there is a big problem for us - the models are written manually and use a lot of memory, for example to implement the best method using GAN, the guys Jonas Bals and Matthias Epple first used pre-processing in Blender, and then wrote 4 own neural models, pre-trained later on a lot of data.

In general there is a big problem with this method:

1. Large dataset for training.
2. Expensive use of computer systems for training.
3. Scarcely described methods of neural network development.
4. The complexity of creating a graphical interface for desktop applications because of the complexity of preprocessing photos and GAN models.

Now we will consider methods that are simpler, but still have a place for writing our project:

OpenCV is a popular open-source computer vision library that provides various functions and algorithms for image processing, computer vision, and machine learning. While neural networks have shown great promise in accurately detecting and characterizing spherical particles in photos of microstructures, OpenCV-based methods can also provide effective solutions for this task. Here's an example of how OpenCV-based methods can be used for particle recognition tasks:

1. Image thresholding is a simple yet effective method for particle recognition. It involves converting a grayscale image into a binary image by setting a threshold value. In a study by Ochs et al. (2018), an image thresholding method was proposed for the detection of spherical particles in microstructure images. The proposed method achieved an accuracy of 93.4% on a test dataset.

2. The Hough transform is a popular feature extraction technique used for detecting lines, circles, and other shapes in an image. In a study by Gopinath et al. (2021), a Hough transform-based method was proposed for the detection of spherical particles in microstructure images. The proposed method achieved an accuracy of 96.5% on a test dataset.

3. Template matching is a technique that involves comparing a template image with a larger image to find the best match. In a study by Singh et al. (2019), a template matching method was proposed for the detection of spherical particles in microstructure images. The proposed method achieved an accuracy of 94.7% on a test dataset.

4. In summary, OpenCV-based methods such as image thresholding, Hough transform, and template matching can also provide effective solutions for automatic recognition of spherical particles in photos of microstructures. These methods can be computationally efficient and easy to implement, making them a good alternative to neural network-based approaches in certain cases where the dataset is small, the particle shapes are simple and well-defined, and computational resources are limited.

4 Implementation

4.1 Comparison of suitable methods

Methods:

1. Haar cascades algorithm works by training on positive and negative examples of the object of interest, using features such as edges and corners to distinguish the object from the background. Once trained, the classifier can be applied to new images to detect instances of the object.

2. OpenCV can be used for object detection and recognition, as well as for various other tasks such as image segmentation and feature extraction.

When comparing Haar cascades and OpenCV for the specific task of automatic recognition of spherical particles in microstructure images, there are several factors to consider. Haar cascades can be effective for detecting objects that have distinctive features, such as faces or other human body parts. However, they may not be as effective for detecting objects that have less distinct features, such as spherical particles.

OpenCV, on the other hand, provides a wide range of functions and algorithms that can be used for detecting and analyzing spherical particles in microstructure images. For example, OpenCV can be used for particle detection using Hough transforms, template matching, or other techniques. Additionally, OpenCV can be combined with other machine learning techniques, such as clustering or neural networks, to improve the accuracy and efficiency of particle detection.

Overall, the choice between Haar cascades and OpenCV for the task of automatic recognition of spherical particles in microstructure images will depend on the specific requirements of the application, such as the size and complexity of the image dataset, the computational resources available, and the desired level of accuracy and precision.

4.2 Results

Consider the first method.

Training a model based on Haar cascades

this method demonstrated poor recognition quality in samples of 50, 100 and 200 photos from the lab with different types of particles. False positives were associated with the presence of a large number of common features in particles of different shapes, despite the differences in particle shapes the number of false positives could reach 30% in one image, provided that some particles of the required shape were not detected. also using this method it became impossible to recognize particles in filtered images.

Consider the second method.

The second method uses the built-in functions in OpenCV for shape recognition along with filtering the image by color. In this case, you need to filter the image by color using the corresponding window in the application, and then search by contour. OpenCV provides the following functions for this:

1. HoughCircles - Used to search for round particles using the Hough transform.
2. ApproxPolyDP - Used to approximate contours, this method is used to determine the shape of shapes by getting the length of the contour list .
3. isContourConvex - Used to determine convexity of shape, allows to exclude round and ellipsoidal particles when searching for amorphous particles.

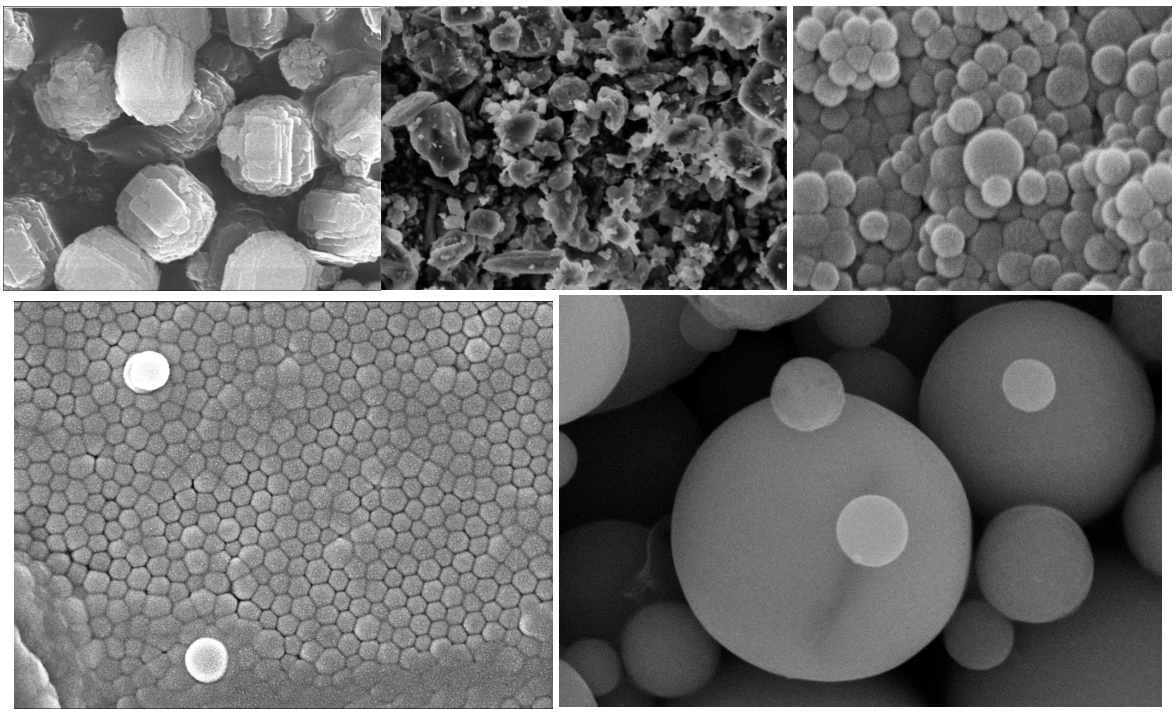
Combined with color range filtering, this method has demonstrated better results compared to the use of Haar cascades, with an accuracy of up to 90% for detecting non-overlapping particles.

In addition, this method has the following advantages:

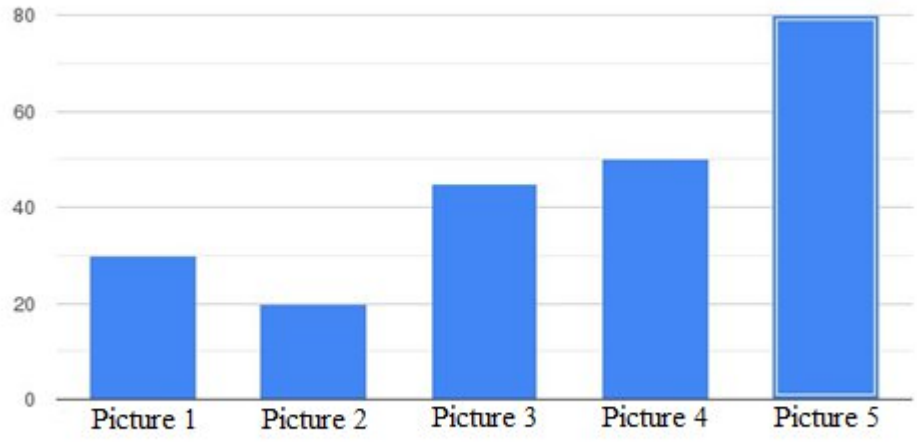
1. Higher recognition rate.
2. No need to train the model.
3. Ability to find the perimeter and area of the particles without using additional methods.

4. Ability to find the number of all angles of the particles, which allows to find the area of any type of particles.

Consider a statistical comparison, based on 5 images of round particles:



Picture 4.1 Photo of which stats were counted (1, 2, 3, 4, 5)

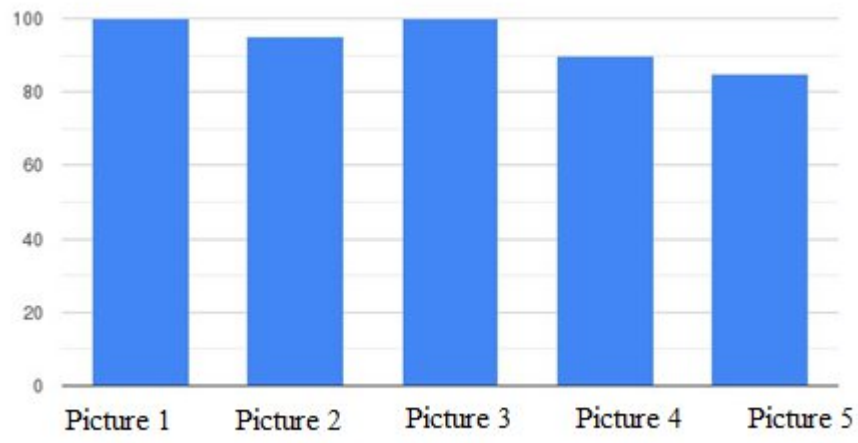


Picture 4.2 Statistics on how many correctly found particles in photos using the “Haar” method.

Statistics according to the method using Haar cascades. The abscissa axis shows the images that were analyzed, the ordinate axis shows the percentage of detected particles in the image.

Note that the image number 5 (with the highest score), was previously in the sample during the training of the model, as well as the particles in the image were at a sufficient distance from each other, so you can conclude that this method is not suitable to automate the search of the particles.

Consider method number two, with the same data:



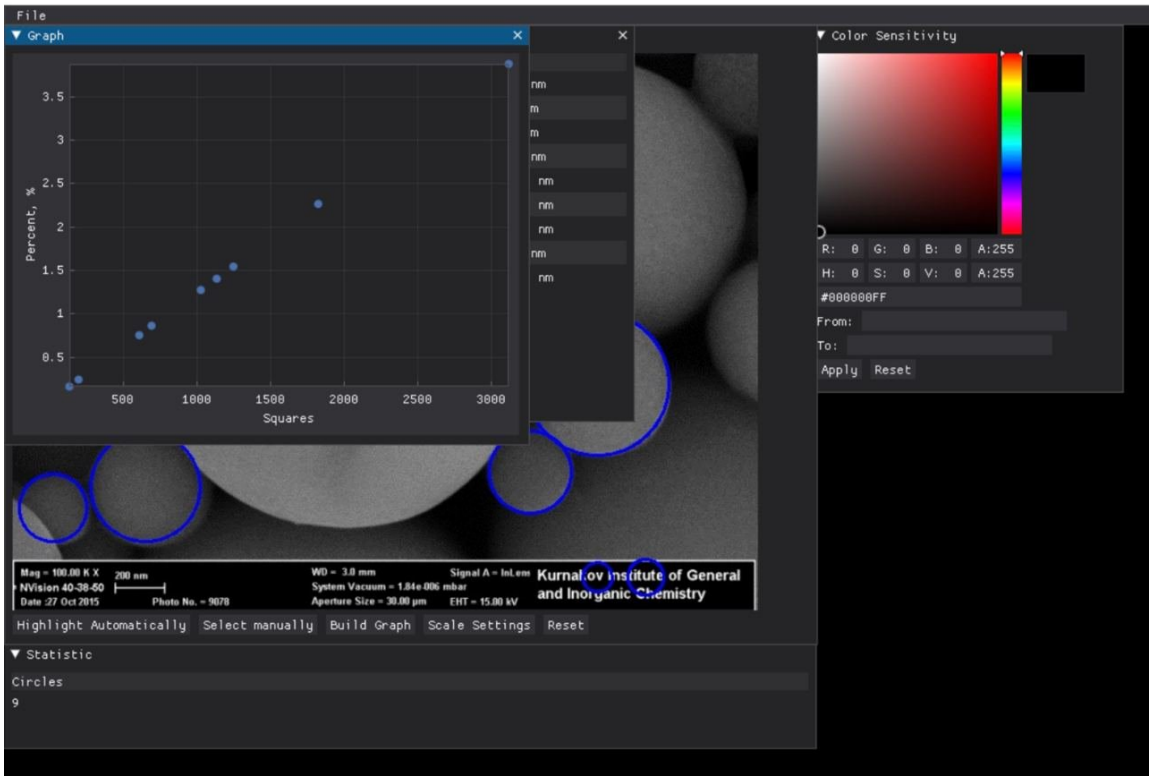
Picture 4.3 Statistics on how many correctly found particles in photos using OpenCV.

The abscissa axis shows the images that were analyzed, the ordinate axis shows the percentage of detected particles in the image.

When using this method, the average result of the accuracy of determining the particles is more than 80%, this is due to the fact that this method determines the type of particles only by contour, ignoring other factors for assessing the accuracy of determination, in the case of determining the shape of the particles, the neglect of other parameters does not give a negative result.

4.3 Implementation in code

To automate the search of the particles, that the lab studies, we created an application in which, we created an application in which implemented methods described above.



Picture 4.4 Program interface.

The “find_all” function, which gets different parameters to find different types of shapes in the image.

1. Reads the input image and applies a Gaussian blur to it."cv2.GaussianBlur()" is an OpenCV library function for applying Gaussian blur to an image.

The Gaus function is described by the following formula:

$$g(x) = ae - \frac{(x-b)^2}{2c^2}, \quad (4.1)$$

a, b, c - are real numbers.

I used this method to remove noise and smooth out the contours of objects in the image. We use this method because in addition to the basic color filtering we will talk about further, we have a blur between the particles themselves, which were demonstrated in photo(2) in the dataset overview and this method helps a lot in approximating the contours between the particle.

```
if find_circles:
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # Conversion to black and white

    gray_blurred = cv2.blur(gray, (3, 3)) # Blurring the image
```

Picture 4.5 Image convertion.

2. Then, if specified in the parameters, it searches for circles in the image using the HoughCircles method from OpenCV.

```
detected_circles = cv2.HoughCircles(gray_blurred, cv2.HOUGH_GRADIENT, 1, 30, param1=param1,
                                    param2=param2, minRadius=minRadius,
                                    maxRadius=maxRadius)  # Looking for circles in the image via the Hough transform
```

Picture 4.6 Search for circles in the image.

If the circles are found, it calculates the area and diameter of each circle and adds them to the appropriate lists.

The formula for area and diameter:

$$S=\pi R^2, \quad (4.2), \quad d=2R, \quad (4.3);$$

```
area = (math.pi * ((width * height) / 4)) * scale
```

Picture 4.7 Circle area.

3. Similarly, if specified, it searches for elliptical shapes in the image using the HoughEllipses method from OpenCV. It then calculates the area and diameter of each ellipse and adds them to the appropriate lists.

```
contours, hierarchy = cv2.findContours(thresh.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)  # Looking for contours
```

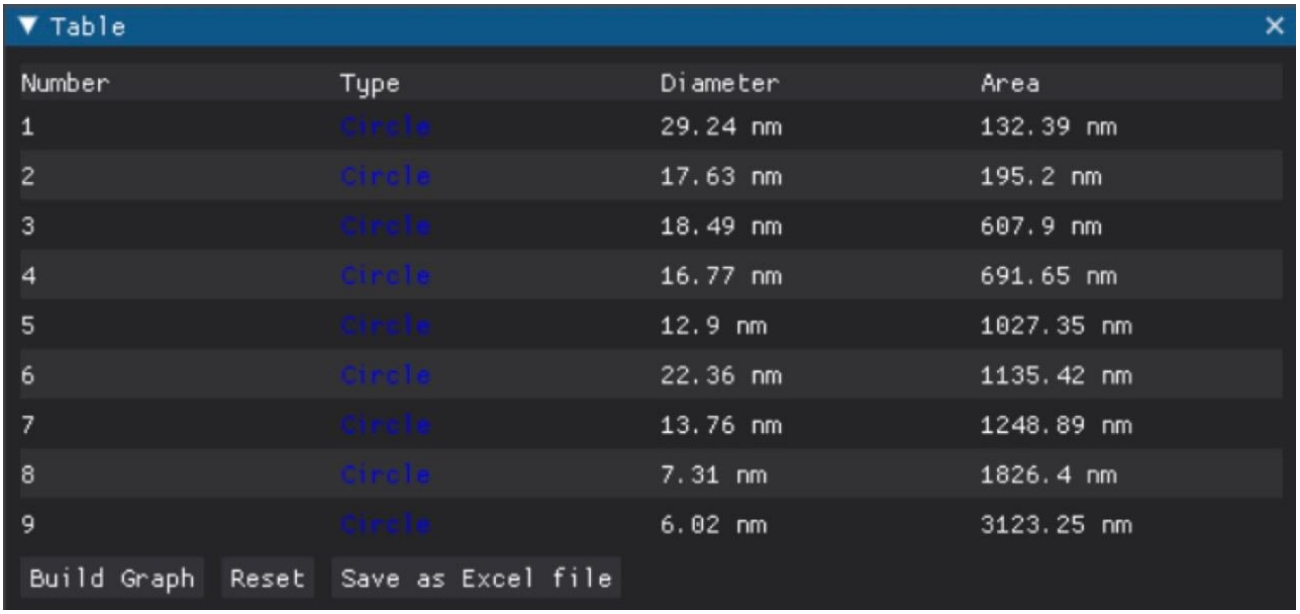
Picture 4.8 Contours search.

4. The function continues searching for other types of shapes (triangles, squares, pentagons, hexagons, heptagons, heptagons, octagons, non-pentagons, and decagons) using the “find_shapes” function from the “squares_math” module.

```
if find_squares:
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    ret, thresh = cv2.threshold(gray, 50, 255, 0)
    contours, hierarchy = cv2.findContours(thresh, 1, 2)
```

Picture 4.9 Search for other types of figures.

5. The function returns the number and area of each type of shapes found in the image.



Picture 4.10 Result table.

Thus, if we consider finding shapes, we have them divided into finding circles, squares, ellipses, and amorphous shapes.

Next we search for such amorphous shapes in the module find_amorphous, where we check the number of sides of a shape, and then we assign its area and perimeter to each of them, draw it, and add it to the list of amorphous shapes.

```
if find_amorphous:
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)[1]
```

Picture 4.11 Search amorphous shapes.

Here's an example for a triangle:

```
if len(approx) == 3:  # If it's a triangle
    perimeter = cv2.arcLength(cnt, True)  # Get the perimeter
    perimeter = round(perimeter * scale, 2)  # Multiply it by the scale. Round off
    area = am.triangle_square(perimeter)  # Get the area
    triangles_areas.append(area)  # Add an area to the list
    img = cv2.drawContours(img, [cnt], -1, (255, 0, 0), 2)  # Draw outlines on the image
    no_filter_img = cv2.drawContours(no_filter_img, [cnt], -1, (255, 0, 0), 2)  # Draw outlines on the image without a filter
    triangles_count += 1  # Add one to the triangle counter
```

Picture 4.12 Calculations for triangular shapes.

Thus, the “forms_recognition” file completely defines all the particles in the photo by applying contour approximation, Gaussian blur and built-in methods like HoughCircles and HoughEllipses to them.

After that other types of definition were manually prescribed and included in the amorphous subsection.

4.4 Support functions

In file “areas_math.py” we set functions to determine the particles of all the shapes that our main algorithm determines.

As soon as we find a shape, the algorithm calls the area search function depending on the shape and counts it for further entry into the table.

An example for a triangle:

```
def triangle_square(perimeter):
    a = perimeter / 3  # Calculate the length of each side of the triangle by dividing the perimeter into three parts
    area = (3 ** 0.5) / 4 * a ** 2  # Calculate the area of a triangle using the formula

    return round(area, 2)  # Return rounded value
```

Picture 4.13 Calculation of the area formula.

And so works for all polygons up to the 10th angle.

Since we apply contour approximation and the particles do not always have convex outlines, it is more likely that the amorphous particles will take the form of polygons, if the figure is not convex it is simply not marked.

All this happens because such a particle is easier to mark itself than to detect automatically because of the complexity of particle overlapping.

The main function, “color_filter()”, is designed to filter the image by the given minimum and maximum color values. To do this, it uses the cv2 library to open the image, converts the specified color values from HEX format to RGB, converts them to Numpy arrays and applies the filter to the image. The result is saved in a temporary folder.

The “filter_image()” function is used to load and display the filtered image in the graphical interface of the program. It gets the minimum and maximum color values, calls “color_filter()” to filter the image, and loads the filtered image. It is then displayed in the GUI of the program.

```
def filter_image():
    min_value = dpg.get_value("min_color")
    max_value = dpg.get_value("max_color")
    od.color_filter(min_value, max_value)
    width, height, channels, data = dpg.load_image("tmp//filter_image.jpg")
    dpg.set_value("texture_tag", data)
    dpg.set_value("Image", "texture_tag")
```

Picture 4.14 “filter_image()” function

The “reset_image()” function is used to restore the original image. If a filtered image exists, it is loaded, otherwise the original image is loaded. It is then displayed in the GUI of the program.

```
def reset_image():
    if os.path.exists("tmp//no_filter_out_image.jpg"):
        width, height, channels, data = dpg.load_image("tmp//no_filter_out_image.jpg")
    else:
        width, height, channels, data = dpg.load_image("tmp//load_image.jpg")
    dpg.set_value("texture_tag", data)
    dpg.set_value("Image", "texture_tag")
```

Picture 4.15The “reset_image()” function

The “hand_select()” function are designed to simplify the handling of particle images under the microscope. They allow you to quickly and easily filter images by specified color values, to load and display them in the graphical interface of the program, and to restore the original image if necessary.

The goal of this function is to manually select geometric shapes on the image and then calculate their area. For this purpose, the user is given the opportunity to draw an arbitrary figure on the image with the mouse.

The function starts when the user invokes a command, for example, by pressing a button on the interface. Then, with the help of the Pygame module and OpenCV, it opens a window with the image where the user can select the desired figure.

When the user finishes drawing the figure and presses the "Enter" key, the function processes the obtained data, determines the type of the figure by the number of

its sides and calculates its area, using the corresponding methods, implemented in other modules.

To calculate the area of a triangle, the “sm.triangle_square()” method from the sm module is used, which takes the lengths of the sides of the triangle and returns the area. For quadrilaterals, the method “sm.square_area()” is used, which takes the coordinates of the corners of the quadrilateral and returns its area. For figures with more sides, the methods: sm.pentagon_area(), sm.hexagonal_area(), sm.heptagon_area(), sm.octagon_area(), sm.nonagon_area(), sm.decagon_area() are used depending on the number of sides. If the number of sides does not match any of the above shapes, the “sm.other_area()” method is used, which takes the total length of the sides and the number of sides and returns the area of the shape.

The calculated areas of shapes are rounded to two decimal places and added to the corresponding list within the function. After calculating the area of a figure, the function waits for one second and resets the list of figure data to zero.

The user can also interrupt the drawing of the figure and close the window by pressing the "Escape" key. In this case the function deletes all drawn lines and ends its work.

This “wait_highlight ()” function is designed to highlight the scale of the image. It runs in an infinite loop and waits for the user to select the line on the image that he wants to use as the scale.

It uses the Dear PyGui library to create a window and user interface elements.

When the user presses the left mouse button, he can select a line on the image by moving the mouse cursor. When the user presses the "Enter" key, a window opens where he can enter the length of the highlighted line in the desired unit. After that, the window closes, the line is hidden and the function terminates.

The function also responds to pressing the "Escape" key, so that the user can cancel the selection of the scale and terminate the function.

4.5 Graph and table for analysis

The second important task in the program was to create a graph and statistics on particles, I decided to first make a module of a table so that it would count the number of particles, their type, size area.

The graph itself is built through DeapPyGui library based on the table.

The “statistic_win” function creates a window with a table that shows the statistics of the different geometric shapes in the image.

The argument pos defines the position of the window on the screen.

The function first deletes the window if it already exists to avoid an error when re-opening it. Then the function creates a new window with the function “dpg.window()”.

The window creates a table with the function “dpg.table()”. Then depending on the number of detected shapes the corresponding columns are added with the function “dpg.add_table_column()”.

Then rows are added to the table using the “dpg.table_row()” function, and for each type of shape that was detected, the corresponding number is added to the corresponding cell using the “dpg.add_text()” function.

All columns and cells in the table have tags that allow you to refer to them to update the value.

The “reset_statistics()” function resets statistical data in a table. It takes a database object and the name of the table to be reset as arguments.

First, the function checks if a table with the specified name exists in the database. If the table does not exist, the function prints an error message and returns False.

If the table exists, the function generates a query to update the statistical data in the table. The query sets all statistical field values to 0.

Once the query has been formed, the function executes it using the execute() method of the database object. If the query was successfully executed, the function returns True. Otherwise, the function also displays an error message and returns False.

So, the "reset_statistics()" function does the following:

1. Checks if a table with the given name exists in the database.
2. Forms a request to update the statistics data in the table.
3. Executes the query using the execute() method of the database object.
4. Returns True if the query was successfully executed, and False otherwise.

First, the code “build_graph()” defines two empty lists areas and percents. Then, for each shape (square, circle, ellipse, triangle, etc.) in their respective lists (e.g., squares_areas), the code appends its area and the percentage ratio of the shape's scale to the image's scale to the areas and percents lists, respectively.

Afterwards, the code creates a new window with a label "Graph" using with dpg.window(label="Graph", width=500, height=400, tag="Graph"):. It then sets the theme of the graph using with dpg.theme(tag=”plot_theme”):. Finally, the code uses the “dpg.mvStemSeries” component to plot the data from the areas and percents lists. The old graph is deleted using “dpg.delete_item()” to avoid errors when opening the window again.

The “get_percent()” function takes two arguments: "a" and "b". The function calculates the percentage of "b" from "a" and returns the value as a floating-point number rounded to two decimal places.

The try-except block is used inside the function to handle the “ZeroDivisionError” exception, which can occur when trying to divide by zero. If the value of argument "a" is zero, the function returns 0 as a percentage.

If the arguments "a" and "b" are correct, however, the percentage is calculated using the formula: percentage = (b / a) * 100. The result of the calculation is rounded to two decimal places using the round() function.

4.6 Interface

The interface consists of a small number of rendering functions that are specified in the modules above:

1. The load_photo() function is used to start a dialog box that allows the user to select an image to load into the program. We create a tuple of filetypes that contains the different file types that can be loaded (such as BMP, JPEG, TIFF and PNG). We then call a dialog box using the askopenfilename method of the filedialog module from the tkinter library. This window allows the user to select a file from the file system. We set the default file extension as *.bmp. The function returns the path to the selected file.

2. The load_photo_err() function is used to display an error message if an attempt to load an image fails. We call the error message dialog box using the showerror method of the messagebox module from the tkinter library. This window displays an error message with an "OK" button.

3. The find_scale() function is used to create a window that asks the user to scale the image. We create a window using the window method of the dpg module. The window contains a text box with instructions for the user and an "OK" button. When the button is clicked, the close_scale_win() function is called.

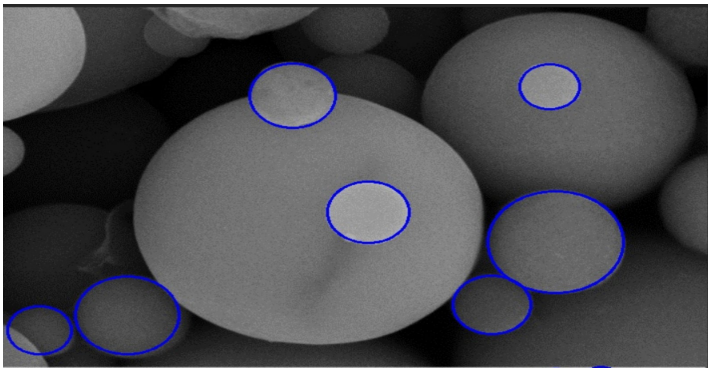
4. The close_scale_win() function closes the window that asks for scale on the image and starts a new thread that calls wait_highlight(). This function is responsible for highlighting the scale in the image.

5. The reset_image() function is used to restore the image to its previous state. If the file "tmp//no_filter_out_image.jpg" exists, we load it and open it in the program window. Otherwise we load the file "tmp//load_image.jpg". We then open the image in the program window using the set_value() and load_image() methods from the dpg module.

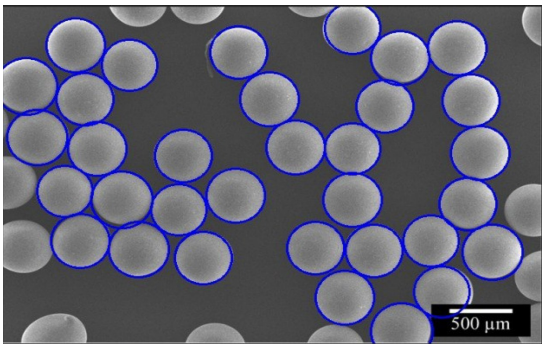
5 Results

The method of automatic particle search through the OpenCV library may encounter a problem when particles overlap each other. In such cases the algorithm may not recognize each particle separately, but rather perceive them as one object. This can lead to errors in further data processing, especially if each particle has to be analyzed separately.

However, if the particles stand apart from each other, the automatic search method via the OpenCV library can detect them normally. Using contour approximation and color correction can help improve the results of automatic particle search through the OpenCV library. However, there may be problems with color correction if there are a lot of black colors in the photo. Removing a gap of colors can lead to blurred outlines, which in turn can reduce the accuracy of particle recognition.



Picture 5.1 Example of a good automatic detection.



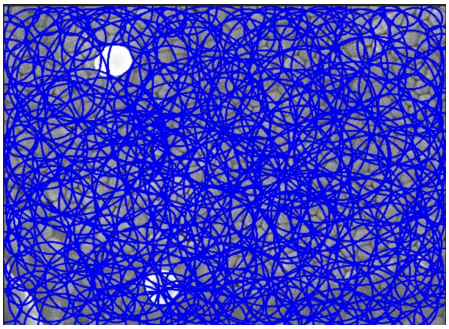
Picture 5.2 Example of a good automatic detection.

In this pictures there are not many particles, they do not adjoin each other and their density in the whole picture is quite small, so the algorithm for automatic detection works perfectly, and those particles that he has not allocated are either marginal to the contours of the picture (which means they are amorphous and should choose to allocate amorphous) or it does not recognize them because of their large size.

In this case, the user can select the missing particles in a couple of clicks and he will have a complete selection

Therefore, when using image processing methods, including contour approximation and color correction, it is necessary to take into account all the factors that can affect the quality of processing and analyze the results in terms of accuracy and completeness. In general, using the automatic particle search method through the OpenCV library can be useful for automating the data processing, but requires additional verification and analysis of the results in case particles can overlap each other.

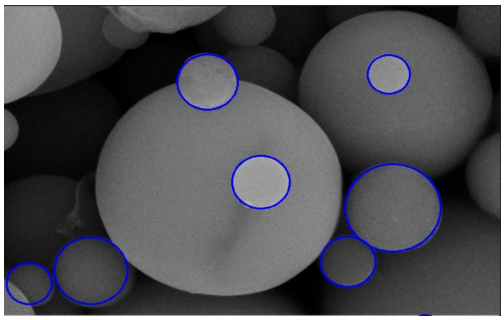
For example a photo where there is a very high density of particles and the automatic detection of circles gives out only 2 circles of all at the setting of the color curve that highlights only the lighter particles when selecting all the particles we get this picture:



Picture 5.3 An example of poor separation of all types of particles.

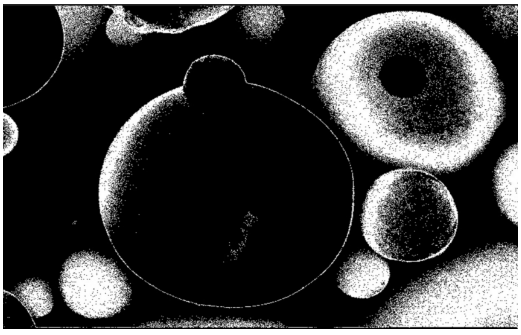
But the same thing we get if the picture has too many blurry black colors, for example, consider the pictures before and after color grading, and take as an example the picture above (picture number 5.1)

Before:



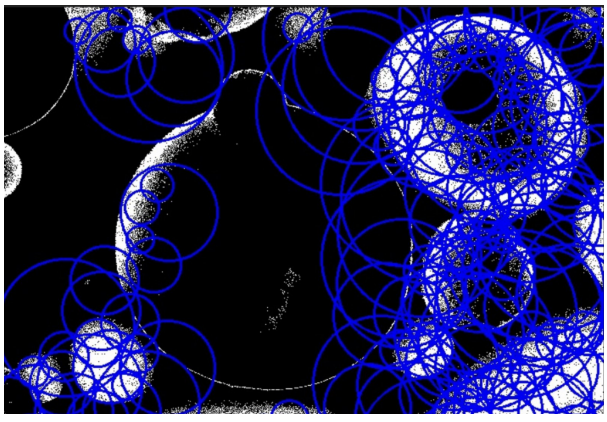
Picture 5.2 Automatic detection before processing.

After:



Picture 5.2 Colour range selection example.

Result:



Picture 5.2 Automatic detection with colour range.

So we can see that even though the result can be good with automatic detection, we still need better color correction and sharpness between the contours, because otherwise we can get such an unreliable result as in this photo

Neural network methods such as CNN and RNN can give good results when recognizing micro-particles in photos. They can be more accurate and reliable when particles overlap or have different shapes and sizes. However, developing and training neural networks can be a time-consuming and costly task, requiring specialized skills and equipment.

In contrast, the automatic particle search method through the OpenCV library is a simpler and more affordable solution for identifying micro-particles in photographs. OpenCV has a simple and intuitive API that allows you to quickly implement the task of particle recognition. In addition, OpenCV is free and open source, making it accessible to a wide audience of developers.

Thus, using an automatic particle search method through the OpenCV library may be a simpler and more affordable solution for detecting micro-particles in photos than using neural networks. However, the choice of method depends on the specific requirements and constraints of the project, which is why this method was chosen to implement the project.

5.1 Innovations

In general, the innovations that I had to do for this work, namely tables and graphs with the analysis and served as one of the innovations that are rarely seen, especially they were not written through the bibiloteca dearpygui due to the fact that the methods mainly use neural networks that are difficult to connect to this interface.

The table is not just a collection of information on the number of particles and their various calculations, but also includes a function of sorting, calculating the percentage of area in relation to the entire picture and also based on the graph, which thanks to a very friendly gui interface is able to auto-scale up and down to examine the particles we need.

Another plus of this gui is that you can move around the graph without reference to the coordinates and can freely move in any direction, both in depth and to the sides.

The main new feature is in-program color correction with setting color values, which then affect the autodetection of particles, this is usually done automatically by different kinds of library for pre-processing photos, but in our case this feature is fundamental to improve the algorithm, though it works as planned but not as in practice because of too much black color and we can not reduce them to less, otherwise the contours of the photos will be blurred too much.

6 Further Development

1. Optimization of OpenCV algorithm parameters to obtain more accurate particle recognition results.
2. Using more advanced computer vision algorithms that can improve recognition accuracy.(Writing neural networks and replacing OpenCV)
3. Combining multiple particle recognition algorithms to get the best results.
4. Developing a multiplatform interface.
5. Exploring other computer vision libraries that may be effective in particle recognition.

7 Bibliography

1. Ochs, M., Bruns, M., Janssen, R., & Tekkaya, A. E. (2018). Automated particle detection in microstructure images using OpenCV. Procedia Manufacturing, 15, 313-320. https://doi.org/10.1016/j.promfg.2018.07.046

2. Gopinath, R., Patil, S. S., & Hegde, C. (2021). Automated detection and analysis of spheroidal particles in microstructure images using Hough transform. Materials Characterization, 176, 110958. https://doi.org/10.1016/j.matchar.2021.110958

3. Singh, G., Singh, M., & Srivastava, R. (2019). Particle detection in microstructure images using template matching. International Journal of Engineering Research and Technology, 12(11), 1986-1990. http://www.ijert.org/view-pdf/24907/particle-detection-in-microstructure-images-using-template-matching

4. Chen, W., Wu, Y., Wu, Z., & Cui, Z. (2021). Automatic identification and segmentation of spherical particles in two-dimensional microstructure images using deep convolutional neural networks. Journal of Microscopy, 281(3), 234-247. https://doi.org/10.1111/jmi.12987

5. Wang, Z., Jiang, T., Chen, X., Xu, H., & Zhang, X. (2020). Three-dimensional particle segmentation in microstructure images based on generative adversarial networks. Materials Characterization, 162, 110208. https://doi.org/10.1016/j.matchar.2020.110208

6. Zhang, J., Sun, W., Jiao, Y., & Wang, H. (2019). Deep convolutional neural networks for particle segmentation in 3D microstructure images. Journal of Microscopy, 276(1), 32-45. https://doi.org/10.1111/jmi.12776

7. Yao, Y., Zhang, X., He, Y., & Shen, L. (2019). Spherical particle recognition in two-dimensional microstructure images using recurrent neural networks. Computational Materials Science, 169, 109087. https://doi.org/10.1016/j.commatsci.2019.109087