# Measuring the position of the scale line on the camera image

Mariusz Wiśniewski (KD-712)
Główny Urząd Miar
ul. Elektoralna 2 00-139 Warszawa
mariusz.wisniewski@gum.gov.pl
Wydział Geodezji i Kartografii PW
plac Politechniki 1, 00-661 Warszawa
mariusz.wisniewski4.dokt@pw.edu.pl

## Abstract

*The report presents an attempt to create a method that detects scale lines on rulers which could be used for 50 meter interferometric comparator located in the laboratory of the Central Office of Measures. I analyzed three approaches: contrast enhancement with image operations and analytical line position detection; edge detection with Hough line fitting; line detection using machine learning. The detected position of the line, was compared with values obtained during manual human inspection. First have the smallest uncertainty of determining the location of the center of the line with the standard deviation 2.56 pix. Second was very unreliable. The last one has the best efficiency in correctly indicating the lines in the image reaching 95%. The methods needs further work to improve the measurement uncertainty*.

## 1. Introduction - Problem

In the process of calibration, it is necessary to determine the accuracy of the scale of measuring instruments. Visual inspection is used in traditional methods. Automated measurements would speed up the process and potentially enable greater accuracy. Difficulties are inhomogeneities in the execution of measurement lines.

### 1.1. Measuring setup

Time and Length Division of the Central Office of Measures has got the only one automatic 50 m interferometric comparator in Poland[1,2,3]. The 50 m bench is used for calibration of tapes of a nominal length (0-50) m, rigid, semi-rigid length measures, semi-rigid to measure the diameter and circumference of a nominal length (0-5) m, laser EDM's in the measuring range (0.5-50) m and determination of laser interferometers. The errors of indicated displacement are determined by comparison with reference laser interferometer.

The bench is situated in the corridor laboratory located in the basement. It has its own air conditioning system. Environmental conditions during calibration is stabilized and set to required air temperature 20 ± 1 °C and relative humidity 50 ± 10 % RH.

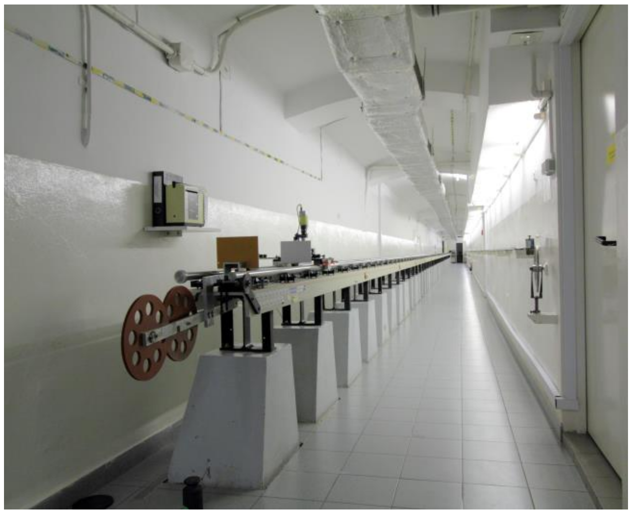
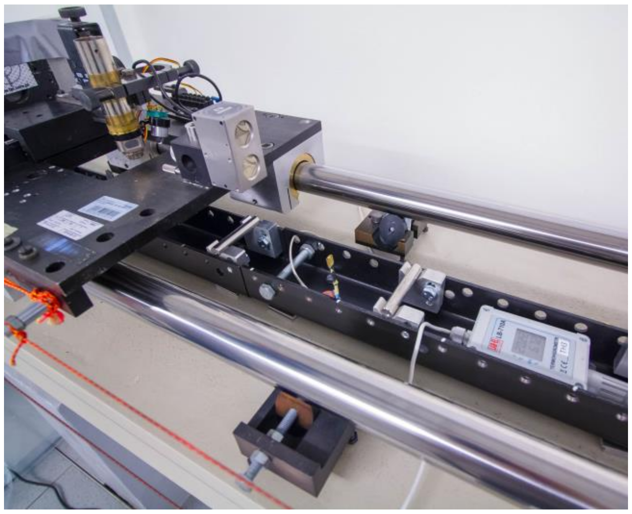



Figure 1: 50 meter interference comparator located in the laboratory of the Central Office of Measures.

The measuring carriage is carrying microscope with CCD camera and the optics associated with the interferometer (Figure 1). The carriage is driven using belt-transmission and electrical engine with remote – controlled variable speed drive. Image from the CCD camera directed

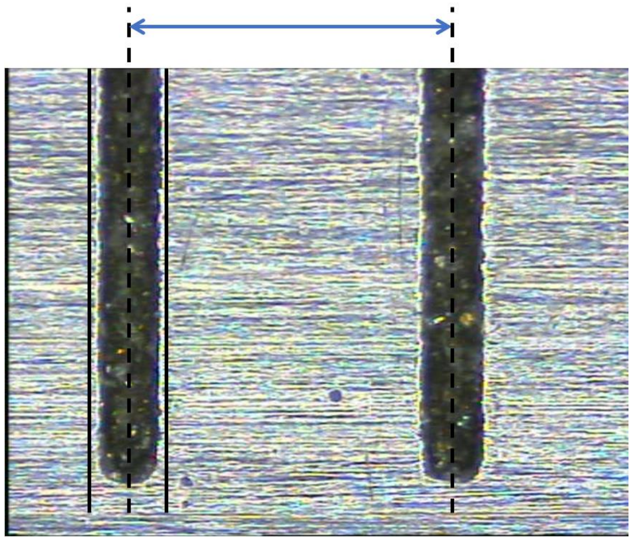


Figure 2: Measurand for the scale lines calibration is the distance between the center of the lines.

at the measured instrument are used to adjust the position of measuring carriage. Scale marks should be placed symmetrically between lines of bisector visible on the monitor. The He-Ne laser interferometer HP 5529A with long-range option laser head HP 5519A is used for measuring positions of scale marks.

## 1.2. Problem

The defined measurand in the case of scale calibration is the distance between the center of the line, which is shown in Figure 2. Scale lines are applied to the surface by various techniques such as incisions, etching or printing. Often the lines are significantly different from the expected even, clearly distinguishable band on the screen. Lines can change their thickness and brightness, contain gaps or holes. In addition, they bear traces of use in the form of additional scratches and stains of any inclination or shape. Sometimes calibrated instruments are crooked, which causes that the distance from the camera changes and the image under the microscope loses focus. The drawn lines can also be lighter, darker than the background or even have a similar brightness but differ in texture.

The task of the person performing the measurements is to interpret the image and indicate the lines in the image. I would like to prepare software that would support or minimize the need for human measurements. I suppose that complete automation will be impossible to achieve because there are cases when even an experienced person has a problem with clearly indicating the position of the line, but with the vast majority of calibration lines the lines are clear and easy to indicate. Reducing the need for manual calibration to only specific cases will significantly affect the efficiency of work in the laboratory. The possibility of using the method in everyday measurements is determined primarily by the uncertainty with which the algorithm will indicate the location of the centers of the lines. It should be not worse than human capabilities, because the goal of the Central Office of Measures is to provide calibration certificates with the best possible uncertainty, regardless of the time necessary to carry out the calibration.

## 1.3. State of the art

As this is a side project, I was unable to devote much time to an in-depth review of the literature. Detecting lines in an image seems to be a common task. Despite this, I have not been able to find a description of the solution to such a problem used in laboratories performing calibration of rulers and scale lines.

In the literature, there are many ways to detect lines, edges and characteristic points that can be used in solving this problem [4]. They are successfully used to analyze images from simple programs that scan completed exam forms to autonomous vehicles and photogrammetric systems. They are also used in professional measuring machines with optical heads[5].

The rapid development of machine learning techniques and object recognition in images may make the creation of complex algorithms unnecessary.[6]

In our laboratory, a few years ago, an attempt was made to automatically determine the position of the line by applying the Gauss function parameters to the cross-section of the line. Due to the previously described line inhomogeneities and disturbances in the vicinity, or due to improper selection of the function, the matched line position was not reliable and the method was never implemented.

At the beginning of the report, I will present data preparation and camera calibration. Next, I will discuss each of the three methods in detail. In the third part, I will present the results obtained by each method. At the end, I will compare the methods and give conclusions.

## 2. Proposed solutions

It is proposed to use automatic detection of measurement lines using three algorithms: with contrast enhancement and analytical line position detection; edge detection with Hough line fitting; line detection using machine learning.

## 2.1. Preparation of test data

Preparing for the task, during each measurement for the last few months, I recorded images from the camera and places indicated manually where, in my opinion, the measurement line is located. The program used on a daily basis at the measuring station can only display the image from the camera and another program puts two vertical lines on its image. It was necessary to prepare a new program for capturing the image from the camera and applying virtual bisector lines to the image, which is capable of saving the image and information about the indicated position of the line. In this way, a database of 1276 images was gathered, which I could use to test analysis methods or to train artificial intelligence.

## 2.2. Camera calibration

The calibration procedure currently used assumes positioning the center of the measured line exactly in the same horizontal position in the image. Calibrating the camera is not necessary in this case because the measurement is always made in the same place, without examining the relative position in the image. For each measurement, the value of the carriage shift relative to the first measured line is read from the interferometer.

Calibration of the camera will allow us to measure the position for a wide range of the field of view, provided that the entire line is in the frame and its center can be determined. It will also allow us to determine the size of the error of the method scaled in the length measurement unit.

A series of measurements of the position of the same line was made for various shifts in the full range of the horizontal axis. For each line position, the value from the interferometer was read and the position of the line on the screen was determined by shifting the virtual bisector by the best-matching number of pixels.

The interferometer software does not provide the possibility of digital transfer of the result to external software. What's more, it must work in a virtual environment because it cannot be run on modern operating systems. It was necessary to prepare a program that would capture the image from the virtual system window and then perform OCR of the values displayed by the program.

The collected values made it possible to calculate the horizontal scale of the image by fitting a linear function to them.

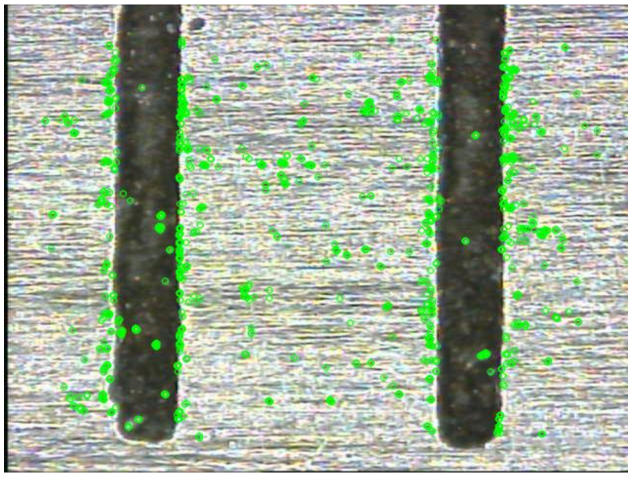


Figure 3: Detection of characteristic points for the method of automatic camera calibration.

Using the collected calibration images and the corresponding positions from the interferometer, I performed an alternative image scaling procedure that did not require to point line position manually. The series of images contains the same part of the scale with the lines only slightly shifted. I used the procedure `detectAndCompute()` to detect characteristic points in the image [7] (Figure 3) and `cv2.BFMatcher()` to match points on images [8]. The `cv2.findHomography()` function allowed to automatically find the offset between the examined image and the first image considered as a reference [9]. Indicating the fixed first image as the reference one limits the possibility of shifting because the images must overlap at least partially for common points to be identified. However, this limits the error that could occur when calculating the offset only between successive images. The function successfully found a large number of characteristic points in the entire range of manual shifts. A linear function was fitted to the determined shifts, as in the case of manual measurements.

I obtained a standard deviation of 0.37 pix and 0.35 pix for manual and automatic measurements, respectively.

The values are so close to each other that it can be said that the automatic algorithm turned out to be as accurate as the manual measurement.

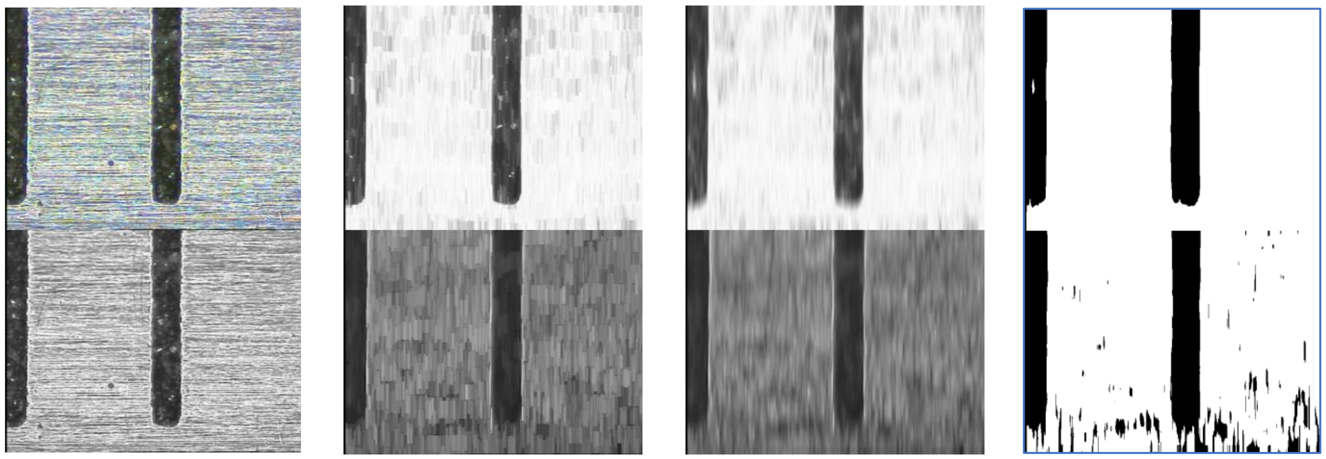
## 2.3. Algorithmic approach

In the first approach, the method of algorithmic image processing and basic signal thresholding will be used to determine the position of the lines in the images.

### *2.3.1 Digital image processing filtering operations*

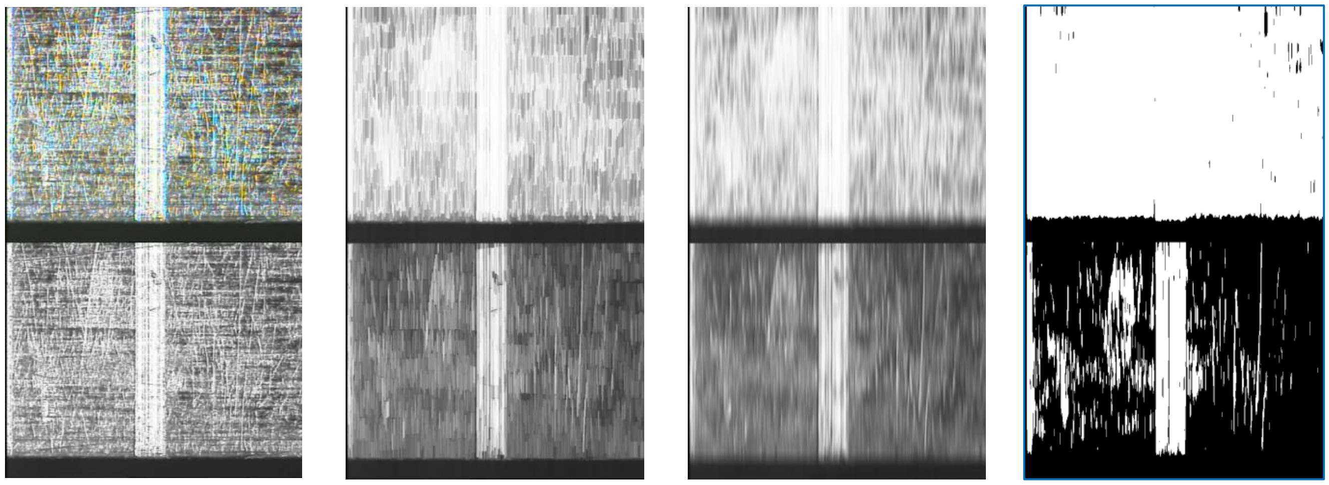
Ideally, we should see the area of the scale line clearly distinguishable from the background brightness level. Applying a threshold to such images would be very simple. Usually, both the background and the lines are covered with various textures with brightness contained in the full gray scale. Some operations are necessary to make the difference between the two areas determinable.

Morphological transformations are simple operations based on brightness and shape on the image [10]. It use structuring element or kernel which decides the nature of operation. The kernel slides through the image. All the pixels near boundaries between areas with different brightness will be affected depending upon the size of kernel. Two basic morphological operators are erosion and dilation. Erosion makes darker areas lighter, and dilation does the opposite. Opening is just another name of erosion followed by dilation. Closing is reverse of Opening, Dilation followed by Erosion. Opening is useful in removing white noise, and closing helps remove dark noise. It is normally performed on binary images but can also be applied to a wider range of levels. The OpenCV



grayscale | closing/opening | average blur | binary threshold (OTSU).

Figure 4: An example of operations performed on an image for the case of dark lines on a light background



grayscale | closing/opening | average blur | binary threshold (OTSU).

Figure 5: An example of operations performed on the image for the case of light lines on a dark background with a large number of random lines making detection difficult.

function `cv2.morphologyEx()` can perform advanced morphological transformations using these basic operations.

The scale lines may be lighter or darker than the background. Therefore, it was necessary to perform different actions on the image for each of the two cases. For the case of dark lines on a light background, the closing filter was applied. For light lines on a dark background, the opening filter worked better. At this point, the program does not yet know what case it is dealing with, therefore both ways of image analysis are performed in parallel.

I used an unusual kernel shape for the above operations. It is only one pixel wide and 29 pixels high. Therefore, it only affects the values in the vertical axis of the image. This is to preserve as much information as possible about the position of the edge of the line. Measured lines are always positioned parallel to the vertical axis of the image. The value of 29 pixels was selected experimentally, increasing it until satisfactory filtering results were achieved and has no theoretical justification.

The resulting images were then blurred by averaging [11] using the same narrow kernel using `cv2.filter2D()`. As a result of the applied operations, we obtain images with a greater contrast of line areas in relation to the background.

### *2.3.2 Image binarization*

In Simple Thresholding, the global value of threshold was used which remained constant throughout. Fixed threshold is not possible because the images are very different from each other. In Adaptive thresholding, the threshold value is calculated for smaller regions with different threshold values for different regions with respect to the change in lighting. In Otsu Thresholding, a value of

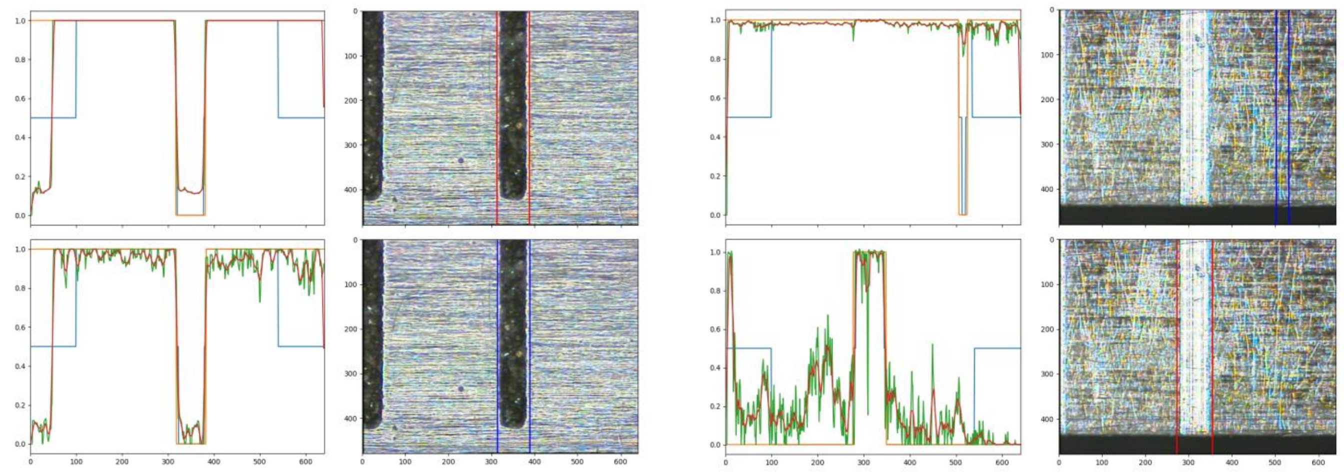

signal analysis detected position of a line signal analysis detected position of a line.

Figure 6: An example of signal analysis and visualization of the position of the detected line edges in the image. The final result is marked in red. A blue result means rejection of the solution.

the threshold isn't chosen but is determined automatically. This threshold is determined by minimizing intra-class intensity variance, or equivalently, by maximizing inter-class variance [12]. Tests have shown that the Otsu method performs better for test data.

The Otsu binarization was applied independently to each of the light and dark line case analysis pipelines. The `cv2.threshold()` function was used, where `cv2.THRESH_OTSU` was passed as an extra flag. Figures 4 and 5 show the effects of subsequent stages of image processing for both cases. One can notice the difference in the response to the filters and the final binarized version of the image..

### 2.3.3 *Indicating the location of the line edges*

The next steps are to indicate the location of the line edges we are looking for. The two-dimensional image was flattened to the horizontal axis by adding up the values in the vertical columns.

It is proposed to use automatic detection of measurement lines using algorithms with conditioning described directly or/and by teaching the neural network to recognize lines.

Noise was smoothed using a 9 pixel wide moving average. The width of the averaging window was selected experimentally. I tried to limit the width so as not to blur the information about the position of the edge of the line.

One-dimensional signal binarization was carried out using hysteresis thresholding [13]. In this method, we define two thresholds. They determine the value $s_x$ from which we consider the state as high and low as certain. The area in between can be defined as a state of uncertainty. We treat an uncertain value as belonging to the surrounding certain state. Analyzing the data from the camera, I set the lower threshold at 30% and the upper threshold at 70% of the total signal amplitude.

$$a = \max(s) - \min(s)$$

$$\begin{cases} s_x > 70\%\, a & \text{high state} \\ 30\%\, a < s_x < 70\%\, a & \text{switching state} \\ s_x < 30\%\, a & \text{low state} \end{cases}$$

Until now, the program does not know whether it is dealing with dark lines on a light background or light lines on a dark background. The lines should be thinner than the surrounding background so that even a fragment of the scale can be seen where the value should be read. The decision about the case can be made by adding up the occurrences of high and low states in the signal after binarization. If the number of highs is higher than lows, these are most likely dark lines on a light background. If the number of highs is less than the number of lows, these are most likely bright lines on a dark background.

The method coped very well with most of the observed noise in the signal, isolating the background and line intervals from each other. In special cases, when the line was barely distinguishable from the background, or was characterized by a vertical structure, it happened that both edges were detected as two or many lines, separated by areas detected as background. To combine these areas into one, I introduced the gluing distance parameter. It determines how closely adjacent areas previously considered as a line can be connected to each other. Unfortunately, this parameter is not universal for each instrument subjected to calibration. Scales with lines spaced 1 mm apart are usually as thick as 0.2 mm. For dense scales, with lines every 0.1 mm, the lines are much thinner. In addition, it is possible to change the magnification of the microscope. The gluing parameter must therefore be set manually on a case-by-case basis or with some other condition. This issue will not be extended or analyzed in this report. The successive stages of signal processing and the recognized position of the line can be traced in Figure 6.

In the initial assumptions for the implementation of the task, I wanted to apply the matching of the trapezoidal function to the received data, but I observed a high instability of the results and abandoned this idea.

*2.3.4 Decide on the best detection result*

Recognizing whether we are dealing with dark lines on a light background or light on a dark background also indicates which of the two lines of image analysis is taken into account as the final result. However, in case both the opening and closing methods lead to the same conclusion about the line type, I leave the last door open for changing the selection. The last value that determines the choice is the signal amplitude. If the amplitude for the opening method is higher, it becomes the final result. If the amplitude for the closing method is lower, it becomes the result. This condition has been experimentally confirmed to be effective on the test sample.

## 2.4. Edge detection with Hough line fitting

In the second algorithm, I tried to detect only the edges of lines instead of extracting whole lines from the background. The raw images were pre-blurred with a median filter with a kernel size of 11 pix. Then I blurred an image of a Gaussian filter with a kernel size of 5x199 pix In this approach, I allowed the lines to be slightly blurred on the horizontal axis. I used a sobel operator working horizontally with a kernel size of 9 pix to detect the edge [14]. In the Figure 7, you can see the positive and negative part of result after using the `cv2.sobel()` function. The brightness maxima correspond to the occurrence of the edge between the light and dark areas, independently for the rising and falling edges. Finally, Otsu binarization was performed on both results.

Straight lines were matched to each image using The Hough line transform [15]. The fitting parameters have been selected so that only vertical solutions are searched, which we expect to describe the scale lines well. This gave me a list of potential horizontal line starts and ends. Sometimes too many edges were found that were close to each other or within a line. The algorithm first found the nearest line starts and ends, and then checked whether it could extend the line with adjacent detections, similarly to the previous approach. Line widening ended when the maximum expected line size was reached. The center of the line was calculated on the basis of the detected extreme edges of the line.

## 2.5. Line detection using machine learning

Once again, a database of sample images was used. For each image, using the `labelimg` program [16], the areas where the lines to be detected are marked. I have divided the objects into two classes: "line" and "edge". I ignored the fact that the lines can be lighter, darker and sometimes barely distinguishable from the background. I wanted to see if this would make a difference when training the model.

I used TensorFlow Lite to train my own detection model [17]. This is my first approach to machine learning, so I used a sample script that guided me through the process of setting up and training the model in a simple way [18]. The

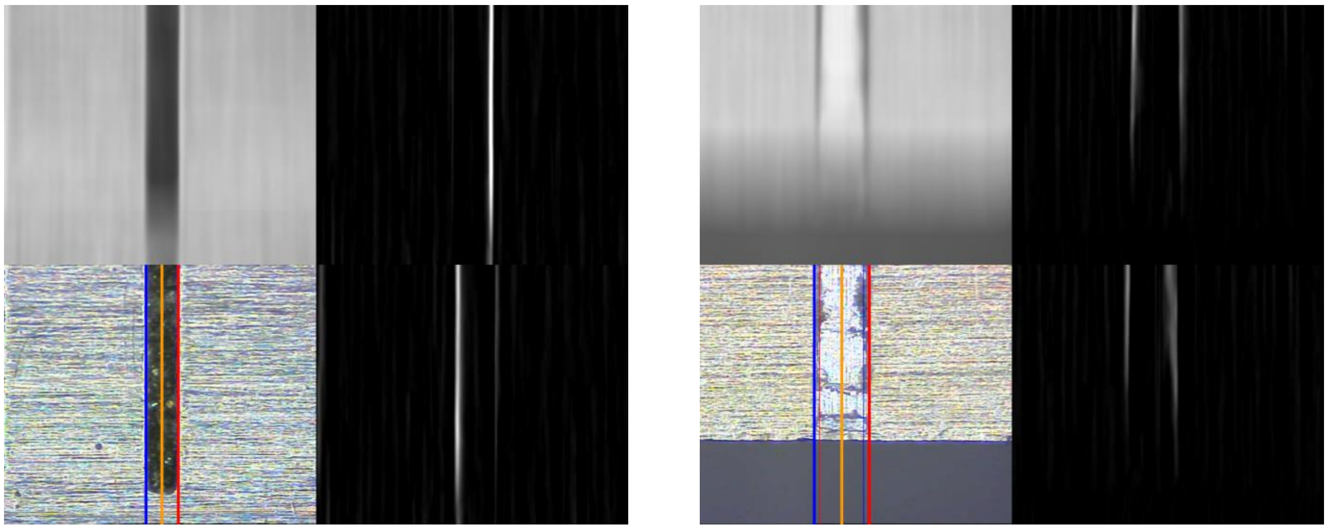


Figure 7: Sample images for the edge detection with Hough line fitting. The top left corner shows the blurred image. The right panels show the results of the right and left edge detection. The lower left panel shows the result of the detection. The edge with decreasing brightness is marked in blue and the edge with increasing brightness is marked in red. The middle of the line is marked in orange.

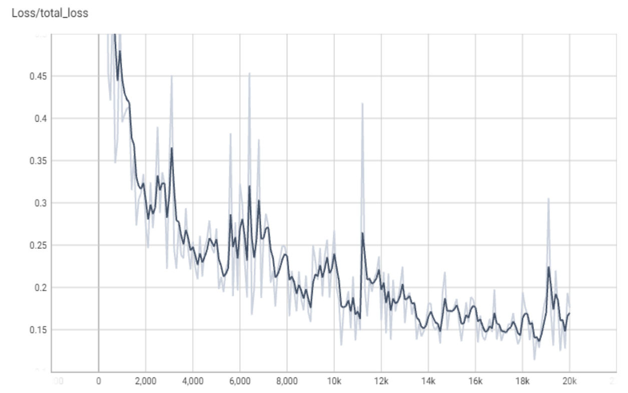


Figure 8: The progress of the learning process along with subsequent iterations.

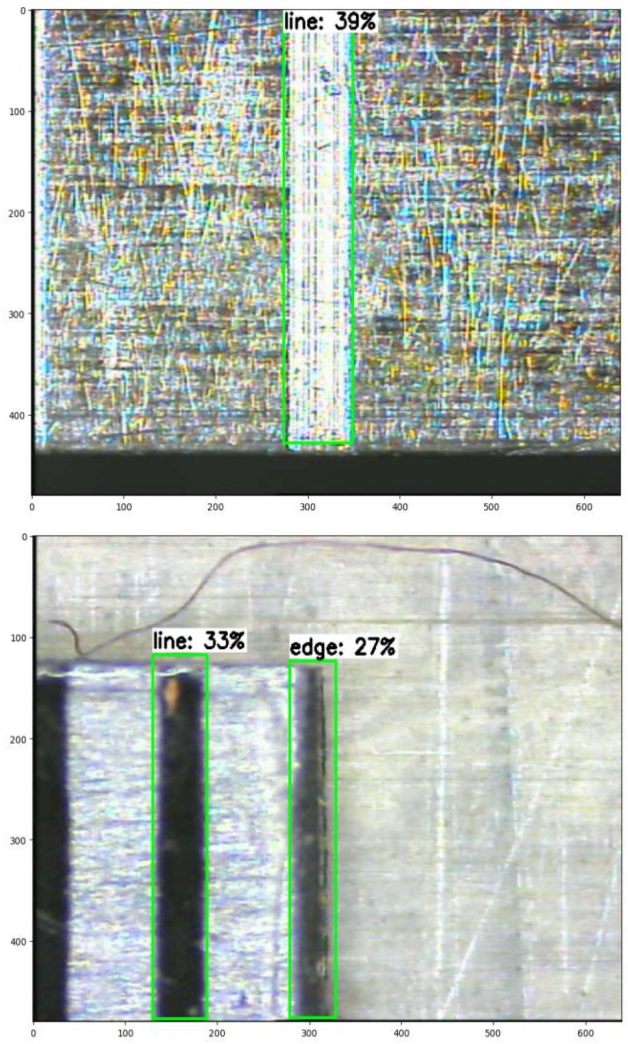


Figure 9: Sample line detections made by the trained model SSD-MobileNet-v2-FPNLite-320x320. Green rectangles represent image areas recognized as lines or edges.

calculations were made in the free Google Colab environment [19]. I used the SSD-MobileNet-v2-FPNLite-320x320 (floating point and quantized) model, which, according to tests, was characterized by high accuracy [20].

Due to the limited free GPU time in the Colab environment, I only used 145 images to teach model. The images were uploaded to the server and divided into parts for training, testing and verification. The training was set for 20,000 repetitions. The model training time was approximately 2.5 hours. The progress of the learning process along with subsequent iterations is shown on Figure 8.

The resulting model was run on the local computer. For the analyzed all images, I received rectangles with a potential place where the scale lines are located. Examples of detection results are shown in Figure 9. Sometimes there were multiple lines in the field of view that were recognized by the model. For comparison with manual measurements, only the line closest to the center of the field of view was analyzed further. For each such line, the position of half its width was calculated.

## 3. Results

The use of the algorithm is intended to replace or reduce the need to perform manual measurements with the necessary human presence. Due to the nature of the task, a significant deterioration in the uncertainty of the results is unacceptable. Therefore, it is crucial to compare the results obtained by the algorithms with the results obtained by an experienced metrologist. A database of 1,276 measurements was used as reference to compare the performance and accuracy of the algorithms.

### 3.1. Algorithmic approach

Sample results for cases detected correctly and those for which the detection was erroneous are presented in Appendix A. The program shows very good efficiency in recognizing clear lines, contrasting with the background. Most calibrated scales have such lines, which means that at least in this case automatic detection can be applied. Also, accidental dirt, scratches, stains or printing defects have little impact on the correct determination of line edges.

The program encounters the greatest problems when the calibrated device fills only a part of the frame and its brightness differs significantly from the surroundings. In this case, the Otse method chooses a threshold not between the brightness of the line and the scale background, but between the brightness of the scale background and the environment. The algorithm then goes blind and doesn't see the line in most cases.

The second most common failure is a very small difference in brightness between the line and the background. This was to be expected for a method based on the contrast between the line and the background. The

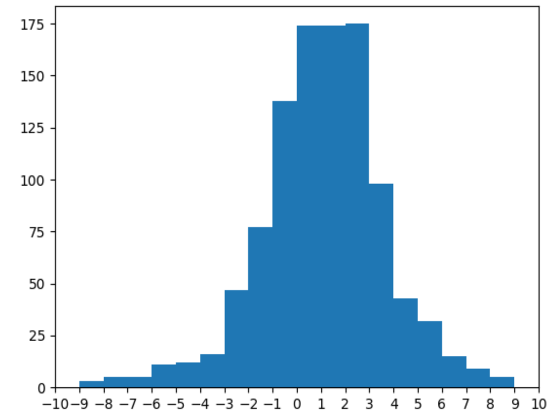


Figure 10: Histogram for the analytical method with the distribution of the difference from the position indicated by a human.

only solution to such cases could be more advanced texture analysis, but this is too broad a topic to cover in this report.

The third common case is a fluctuating line brightness detected as many lines or a fragment of a line. Despite the use of hysteresis and the gluing function, the part of the line whose total brightness turns out to be insufficient to exceed the threshold is often lost.

The analytical algorithm, which can be described as a brute force algorithm, turned out to be surprisingly effective in detecting lines in the image. For 1041 images (81.5%), the center of the line was determined at a distance of no more than 10 pixels from the position indicated by the human. This means that it was unable to detect lines in only 18.5% of cases. The histogram with the distribution of the difference relative to the position indicated by a human is shown in Figure 10. The histogram shows values only in the range from -10 to 10 ignoring outliers. the standard deviation of the difference for the tested sample is 2.58 pix. The standard deviation tested for repeated human measurements of the same single line was 0.95 pix. This means that the uncertainty of the measurement made by the current version of the algorithm is greater than human capabilities. Still, it should be noted that this is a very good result for such a primitive algorithm in the initial testing phase.

Using the information obtained from the calibration of the camera, the size in pixels can be converted to millimeters. The obtained standard deviation is 7.7 µm.

An alarming value is the median difference of 1.0 pix. Such an accurate value suggests that at some stage of the transformation there is a positional error of one pixel. Despite repeated checks of the program, it has not yet been possible to determine which function may be responsible for this.

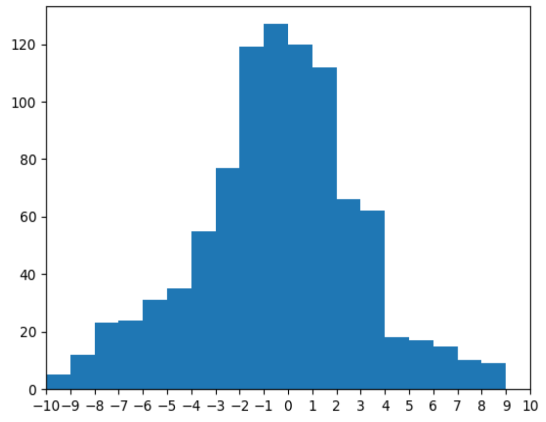


Figure 11: Histogram for the edge detection method with the distribution of the difference from the position indicated by a human.

## 3.2. Edge detection with Hough line fitting

Sample results for cases detected correctly and those for which the detection was erroneous are presented in Appendix B. The program shows good efficiency in recognizing clear lines, contrasting with the background. The presence of additional vertical artifacts very often led to the detection of a significant number of false edge detections. The algorithm of combining adjacent detections did not always lead to the determination of the correct position of the line.

For 938 images (73.5%), the center of the line was determined at a distance of no more than 10 pixels from the position indicated by the human. This means that it was unable to detect lines in 26.5% of cases. The histogram

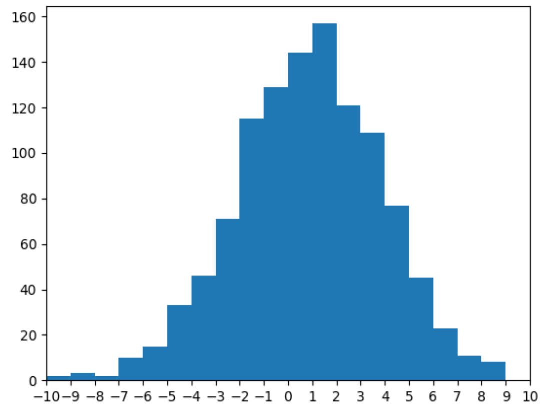


Figure 12: Histogram for the machine learning method with the distribution of the difference from the position indicated by a human.

with the distribution of the difference relative to the position indicated by a human is shown in Figure 11. The standard deviation of the difference for the tested sample is 3.4 pix. This means that the uncertainty of the measurement made by the Hough version of the algorithm is greater than previous method. Due to poor detection efficiency and large errors in position assessment, it is not suitable for use in everyday measurements.

Using the information obtained from the calibration of the camera, the size in pixels can be converted to millimeters. The obtained standard deviation is 10.7 μm.

## 3.3. Line detection using machine learning

Sample results are presented in Appendix C. Line detection with the trained model did an excellent job of pinpointing 1122 lines (95.1%) of the test images. Despite such high efficiency, it did not detect many clear lines in a few special cases. There were probably too few such examples among the training images.

High efficiency did not go with very good accuracy. The standard deviation of the difference for the tested sample is 3.0 pix. Using the information obtained from the calibration of the camera, the standard deviation is 9.1 μm. The histogram with the distribution of the difference relative to the position indicated by a human is shown in Figure 12.

It is difficult to say what criteria the model used to select the area, but often the position of the start and end was randomly shifted, mainly in right direction. It is worth noting that the model never confused the line with the background, whether it was lighter, darker or barely distinguishable. In addition, the model was the only one able to correctly name and indicate the lines located on the ending edge of the scale.

## 4. Summary

Three approaches to automatic detection of scale lines observed by the camera are presented.

The smallest uncertainty of determining the location of the center of the line was obtained thanks to the analytical method. The method provides full control of each image processing step and the conditions determining the location of the line. The method fails in the case of lines barely distinguishable from the background, and when the scale occupies only a small part in the field of view.

The best efficiency in correctly indicating the lines in the image was achieved by the method using machine learning. The method allows us to set the confidence threshold, but it is not known what criteria are used to select the line.

The method based on edge detection turned out to be very unreliable and only in the case of clear lines gave results similar to the analytical method. The filtering parameters were very difficult to set.

Measurement uncertainty with each of the three methods at the present stage is 2-3 times worse than human capabilities. Nevertheless, the results are promising. Image processing for the analytical method can probably be improved by selecting the right areas and better segmentation. The model can be trained on a much larger sample and with more repetitions. It is also possible to combine methods, for example, using a high-performance machine learning model to indicate a line and then an analytical method to analyze only a fragment of the image.

The algorithm needs further work to improve the measurement uncertainty. More analysis and further testing is needed to see if this level of uncertainty could be accepted as a standard method of automatic measurement. Perhaps the automation obtained thanks to this algorithm will allow to increase the number of repetitions of measurements to reduce the average error, but if the program detects lines in the same way each time, the repetitions will not lead to a better result.

The software has been made available on github: `http://github.com/MarandW/CV_lines_detector`

## References

[1] Czułek, D. Wzorcowanie Materialnych Miar Długości – Przymiarów Wstęgowych Na Zautomatyzowanym 50 m Komparatorze Interferencyjnym, *XXXVII Międzyuczelniana Konferencja Metrologów i Konferencja Grantowa,* , nr 5, 93-100, 2005.

[2] Wiśniewski, M. 50 m komparator interferencyjny, IX Szkoła-Konferencja *„Metrologia Wspomagana Komputerowo", MWK-2011*, 99-100, 2011.

[3] Wiśniewski, M. Monitoring warunków środowiskowych i ich wpływ na wyniki pomiarów uzyskiwanych 50 m komparatorem interferencyjnym, *Podstawowe Problemy Metrologii, PPM 2012*, 123-126, 2012

[4] Rui Sun et al., Survey of Image Edge Detection, *Frontiers in Signal Processing*, Volume 2, 2022. doi 10.3389/frsip.2022.826967

[5] Catalucci, S., Thompson, A., Piano, S. et al. Optical metrology for digital manufacturing: a review. *Int J Adv Manuf Technol* 120, 4271–4290 (2022). https://doi.org/10.1007/s00170-022-09084-5

[6] Z. -Q. Zhao, P. Zheng, S. -T. Xu and X. Wu, "Object Detection With Deep Learning: A Review," in *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3212-3232, Nov. 2019, doi: 10.1109/TNNLS.2018.2876865

[7] Introduction to SIFT `https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html`

[8] Feature Matching `https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html`

[9] Morphological Transformations https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html

[10] Smoothing Images
`https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html`

[11] Image Thresholding
`https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html`

[12] Canny Edge Detection
`https://docs.opencv.org/4.x/d7/de1/tutorial_js_canny.html`

[13] Sobel Derivatives
`https://docs.opencv.org/4.x/d2/d2c/tutorial_sobel_derivatives.html`

[14] Hough Line Transform
`https://docs.opencv.org/4.x/d9/db0/tutorial_hough_lines.html`

[15] LabelImg
`https://github.com/heartexlabs/labelImg`

[16] TensorFlow
`https://www.tensorflow.org/`

[17] Evan Juras, TensorFlow Lite Object Detection API in Colab
`https://colab.research.google.com/github/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi/blob/master/Train_TFLite2_Object_Detction_Model.ipynb`

[18] Google Colab
`https://colab.research.google.com/`

[19] Evan Juras, TensorFlow Lite Object Detection Model Performance Comparison
`https://www.ejtech.io/learn/tflite-object-detection-model-comparison`

## Appendix A

Examples of detection of scale lines using the algorithmic method

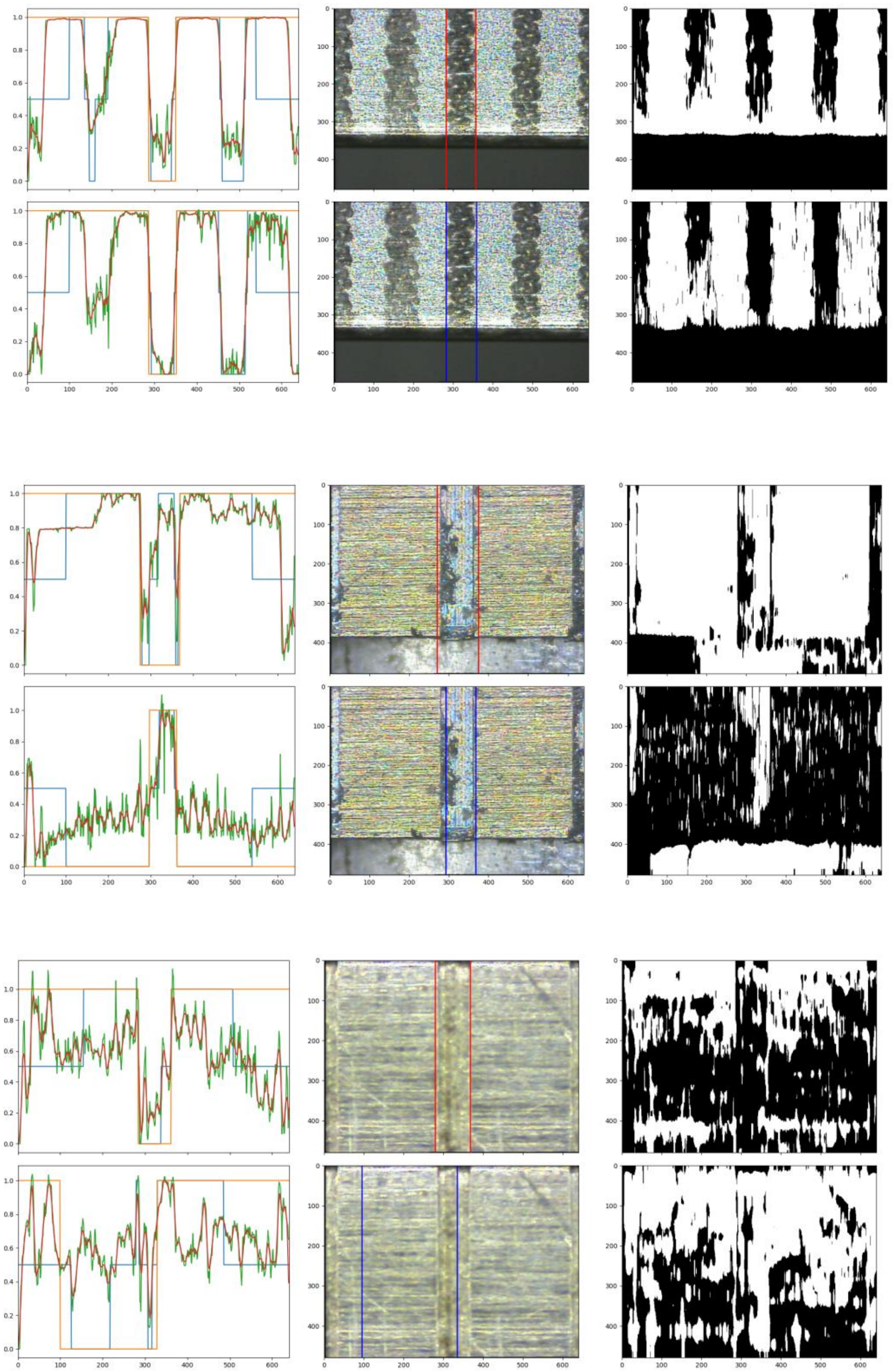


Figure A1: An example of signal analysis and visualization of the position of the detected line edges in the image. The final result is marked in red. A blue result means rejection of the solution.

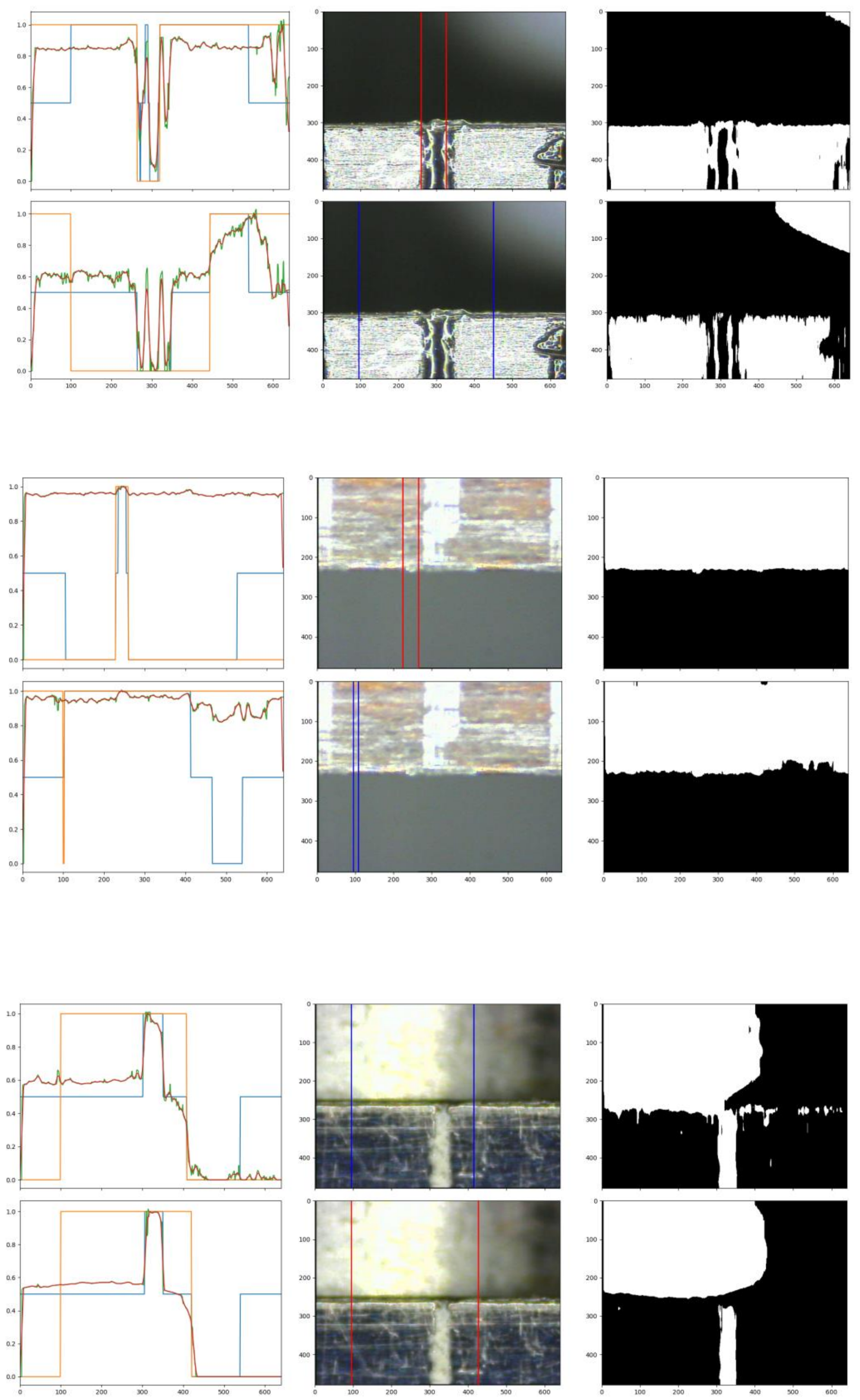


Figure A2: An example of signal analysis and visualization of the position of the detected line edges in the image. The final result is marked in red. A blue result means rejection of the solution.

## Appendix B

Examples of detection of scale lines using edge detection with Hough line fitting

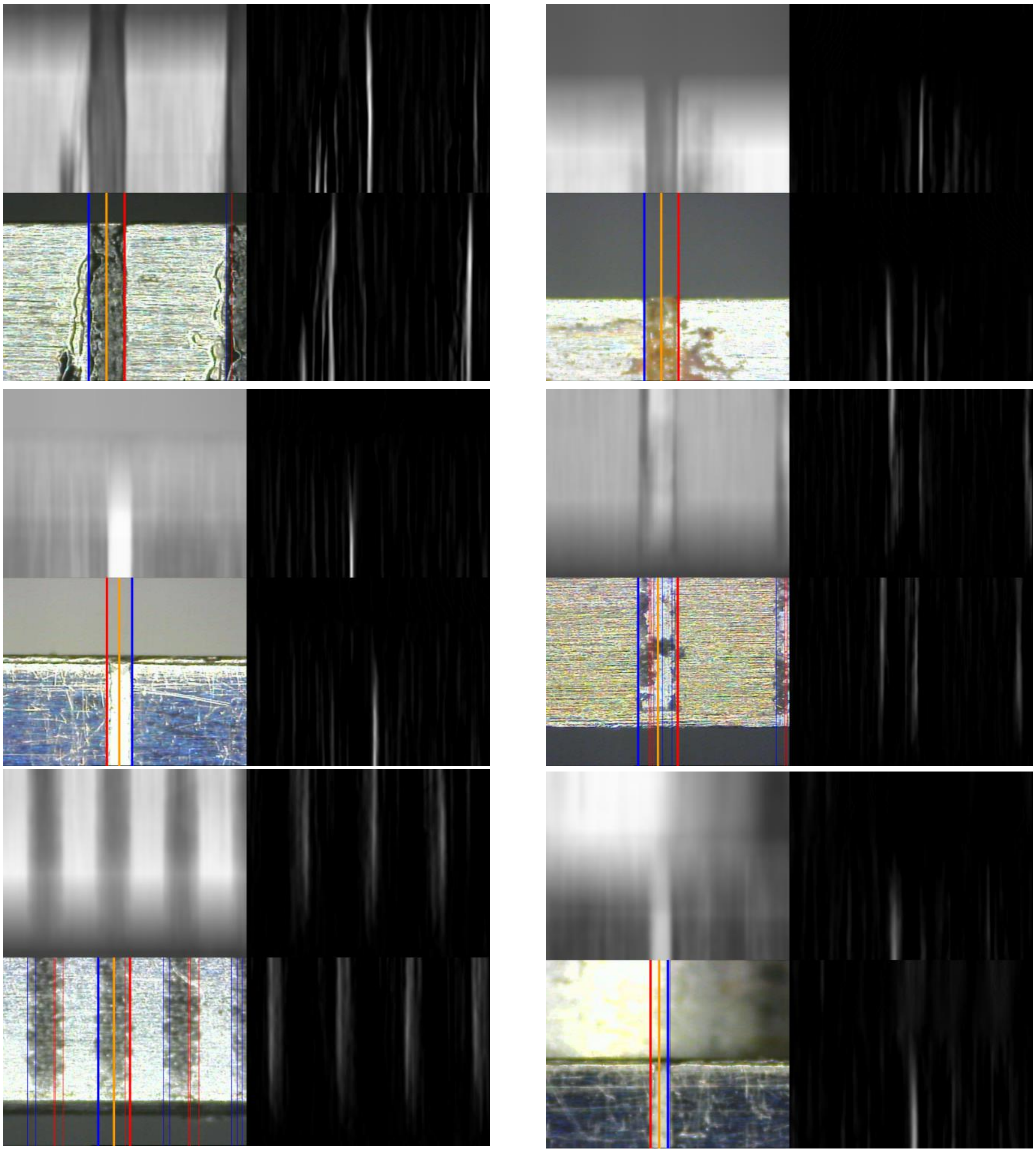


Figure B1: Sample images for the edge detection with Hough line fitting. The top left corner shows the blurred image. The right panels show the results of the right and left edge detection. The lower left panel shows the result of the detection. The edge with decreasing brightness is marked in blue and the edge with increasing brightness is marked in red. The middle of the line is marked in orange.

## Appendix C

Examples of detection of scale lines using machine learning

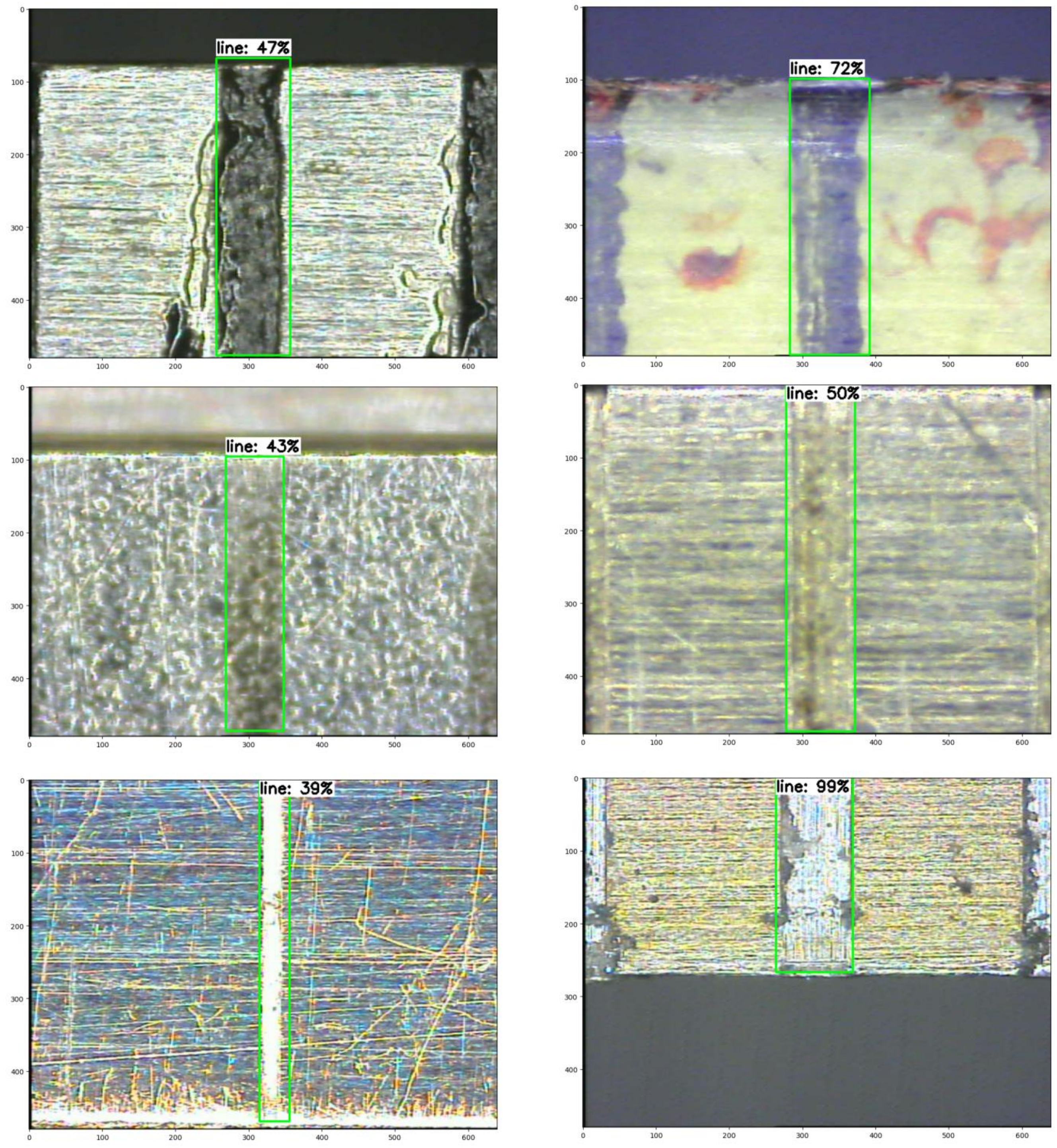


Figure C1: Sample line detections made by the trained model SSD-MobileNet-v2-FPNLite-320x320. Green rectangles represent image areas recognized as lines or edges.