Automated analysis of dental bur pictures

(Automatyczna analiza zdjęć wierteł dentystycznych)

Igor Tryhub
Numer albumu: 275235

Praca inżynierska

Promotor: dr Paweł Rychlikowski

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

14 września 2018

Abstract

Dental burs are used for cutting hard tissues in most dental operations. Despite their classification has a clear form and structure, it is very difficult to distinguish similar burs from each other. During a medical intervention, an automatic localization and classification of burs could help prevent unsafe situations or even injuries. Therefore, it would be extremely useful to facilitate identification of dental burs by their image. There are very few readily available standardized images of labeled burs to create and train a machine learning classification tool. Therefore, there must precede a bur image detection system as a pre-processing stage. Dental bur detection process presented in this work constitutes a pipeline of applying different computer vision image processing tools and filters. As a result, the implemented system achieved 77.15% of bur detection accuracy rate which could be further improved up to 94.05% providing a better quality of input images.

Keywords: image analysis, image processing, object detection, dental burs, computer vision, OpenCV.

Wiertła dentystyczne są używane do cięcia twardych tkanek w większości operacji stomatologicznych. Mimo że ich klasyfikacja ma wyraźną formę i strukturę, bardzo trudno jest odróżnić podobne wiertła od siebie. Podczas interwencji medycznej automatyczna lokalizacja i klasyfikacja wiertła może pomóc w zapobieganiu niebezpiecznym sytuacjom, a nawet urazom. W związku z tym niezwykle przydatne byłoby ułatwianie identyfikacji wierteł dentystycznych za pomocą ich zdjęcia. Istnieje bardzo niewiele łatwo dostępnych, wystandaryzowanych obrazów oznaczonych wierteł, aby stworzyć i wytrenować narzędzie uczenia maszynowego do ich klasyfikacji. Dlatego też musi istnieć system wykrywania obrazu wierteł jako etap wstępnego przetwarzania. Zaprezentowany w tej pracy proces wykrywania wierteł dentystycznych stanowi potok różnych narzędzi i filtrów wizji komputerowej do przetwarzania obrazu. W rezultacie, wdrożony system uzyskał 77.15% dokładności wykrywania wierteł, która mogłaby być poprawiona do 94.05%, pod warunkiem lepszej jakości obrazów wejściowych.

Słowa kluczowe: analiza obrazów, przetwarzanie obrazów, wykrywanie obiektów, wiertła dentystyczne, wizja komputerowa, OpenCV.

Contents

1 Introduction **4**

2 Methodology **6**

3 Implementation Details **11**

4 User Manual **12**

5 Use Cases and Effects **13**

6 Conclusions and Future Work **16**

7 Appendices **18**

8 References **19**

1 Introduction

Dental burs are used for cutting hard tissues in most dental operations. A similar tool has been used since the very beginning of dentistry, constantly improving year by year. For such a long history of dentistry, a huge number of the most varied burs was created, for the most diverse tasks of a dentist. They are usually made of diamond grit (figure 1) or tungsten carbide (figure 2). These rotational instruments oftentimes have a complex shape of the working part. By correctly picking up all the characteristics of a bur, dentists can achieve the most convenient and effective work. Otherwise, problems of a very different nature may arise in the course of a treatment.

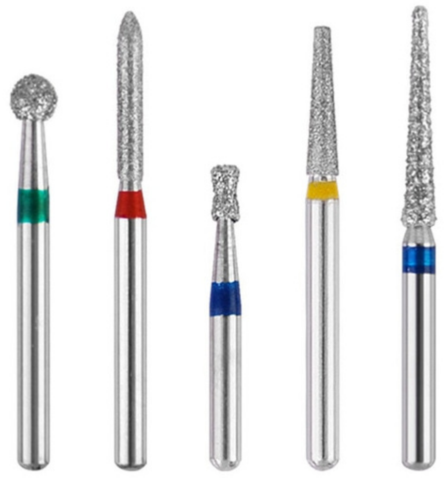


Figure 1: Diamond burs. Source: [1]

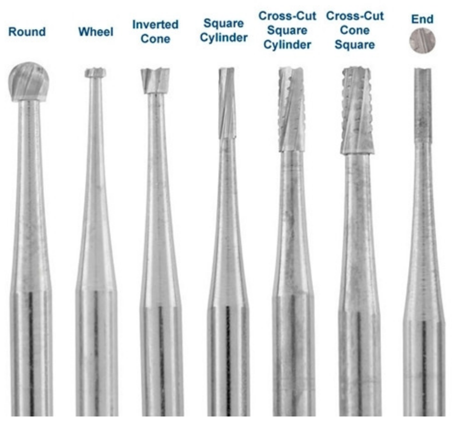


Figure 2: Carbide burs. Source: [2]

Producers carry out the classification according to the general commodity international system of standards – ISO. According to the ISO system, a group membership of the instrument is determined by the following characteristics:

type of material covering the working part of the tool, length and type of the shank, shape of the working part, the abrasiveness of the working part, and the largest diameter of the working part of the tool. For instance, the ISO-code 806.314.001.524.018 carries the following information:

- 806 – diamond coating with galvanic bond;
- 314 – shank for a turbine handpiece;
- 001 – spherical shape of the working part;
- 524 – medium coarse abrasiveness;
- 018 – the largest diameter of the working part is 1.8 mm.

Despite this classification has a clear form and structure, it is extremely difficult to distinguish similar burs from each other. Oftentimes, burs are handled without any package and the engraved coding marks are hard to read without special optical equipment. During a medical intervention, an automatic localization and classification of burs could help prevent unsafe situations or even injuries. We approached few experts in the industry who handle dental burs on a daily basis in large amounts, and they confirmed the need of building a solution to facilitate identification of burs by their image.

The initial idea of this thesis was to create a first-of-a-kind machine learning tool, and to train it on numerous visual data coming either from self-made pictures or the Internet. But the problem was that very few standardized images of labeled burs are readily available. In order to efficiently train a neural network classifier, the input labeled images need to be consistent with each other, i.e. to have similar orientation, size and illumination. Therefore, there should inevitably arise a bur image detection system as a pre-processing stage which would enable their further recognition. That is why in this work the author focuses on localization of bur images, leaving out their classification for future studies.

As of the moment of this work publishing, there is no solution known to the Author which would enable extraction of burs from the background. Consequently, it is difficult to evaluate the effectiveness of the implemented system. However, in 2016 there was published a similar study by French researches aiming to localize and identify laparascopic instruments from a surgery video stream [3]. Even though, general conditions and limitations of these two studies are different, a good future reference value was established – out of 623 labeled images, their system achieved 71.63% total accuracy rate in image classification for 7 kinds of instruments.

2 Methodology

The image processing methods can be roughly divided in two main groups: computer vision techniques [4] and machine learning techniques [5] . Machine learning approach is more contemporary, but also more demanding in computations. It uses statistical methods to infer new knowledge from a lot of tagged data, without being explicitly programmed. Currently, the best algorithms for object detection tasks are based on convolutional neural networks [6] (CNN,), even though they still struggle with small objects or distorted images. They use some features to imitate the visual cortex: so-called simple cells react to straight lines from different angles, whereas complex cells' reaction is associated with the activation of a certain set of simple cells. The idea of CNNs is to alternate convolution layers which aggregate the product results for each fragment from the previous layer, and pooling layers which are a non-linear compaction of a characteristic map. On the other hand, computer vision relates to how computers can be made to obtain a high-level understanding of digital images. It is a more traditional approach originating from the second half of the 20th century which, from the technical point of view, aims to automate tasks that the human eye can do. Computer vision is usually based on relatively simple and fast computations.

Taking into consideration all the advantages and disadvantages of the two groups, it was decided to use a combined approach for the needs of this project. For the initial bur detection stage, it was resolved to apply computer vision tools for the sake of both implementation simplicity and runtime efficiency. The idea was to create a processing pipeline of computer vision methods in order to find smaller regions of interesting image data which can be further classified by more sophisticated computational methods, such as machine learning.

The major complexity when first tackling the problem was the author's lack of experience in computer vision techniques. He had neither previously taken any computer-graphics-related courses, nor had he worked with any computer vision libraries. First steps in approaching the solution were rather uncertain and required a lot of trials and errors before the proper and efficient way of building an processing architecture was invented. At the initial stage, the implemented program was only tested on a very limited number of bur pictures. But with the level of development of the software solution, the number of test pictures grew, which allowed to catch more and more sophisticated subtleties encountered in new images, until the system achieved acceptable level of effectiveness.

Dental bur detection process implemented in this work constitutes a pipeline of applying different image processing tools and filters (figure 3). The initial stage of the pipeline consists of reading an original image from a specified source folder and converting it from RGB into a grayscale color space. Grayscale conversion is a kind of image simplification which facilitates its binarization and at the same time ensures minimal loss of information encoded in RGB color space. There were also attempts to use RGB to HSV conversion as an intermediate step, but several simulations proved it to be ineffective as the most recognition-relevant information was contained in the V-channel, which turned out to be identical to the information contained into a grayscale representation.

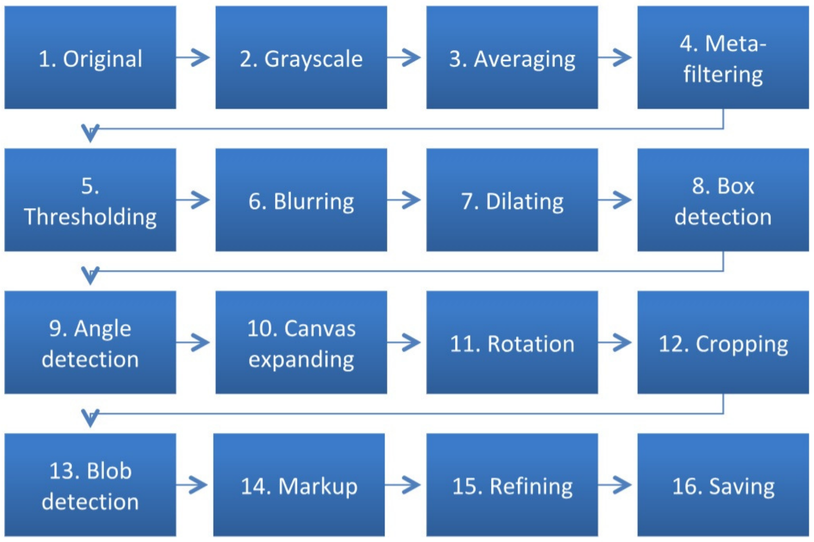


Figure 3: Image processing pipeline

Some of the source images were made in conditions of imperfect illumination. In some of them, large and dark shadows surrounded the instruments being detected. Some other contained strong light glisters on the burs. After the conversion to a grayscale, the saturation of shadows was very close to the saturation of the instruments and the saturation of glisters was close to the saturation of the background. In order to diminish these two negative effects, it was decided to apply a two-dimensional convolution averaging filter with a kernel of size 15 (figure 4), which replaces each central pixel with a new average value of its neighborhood. As a result, it would reduce the noise and normalize the image for subsequent handling. We also tried to introduce a contrast stretching function into the processing pipeline. However, the empirical comparison of its effects proved to bring unsubstantial improvement and was eventually excluded for the sake of simplicity.

$$K = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Figure 4: An example of a 5x5 averaging kernel

Once illumination-related noise is substantially diminished, the image is ready for extracting the background. Thresholding segmentation is the method aiming to separate darker objects in the foreground from a significantly lighter background. There are different classes of built-in methods to perform thresholding in OpenCV (Open Source Computer Vision Library) [7]: simple thresholding and adaptive thresholding. Simple thresholding is more straightforward and simply divides the image pixels into two classes based on the thresholding saturation value. Whereas, adaptive thresholding takes into consideration lighting conditions in different regions of an image (figure 5).

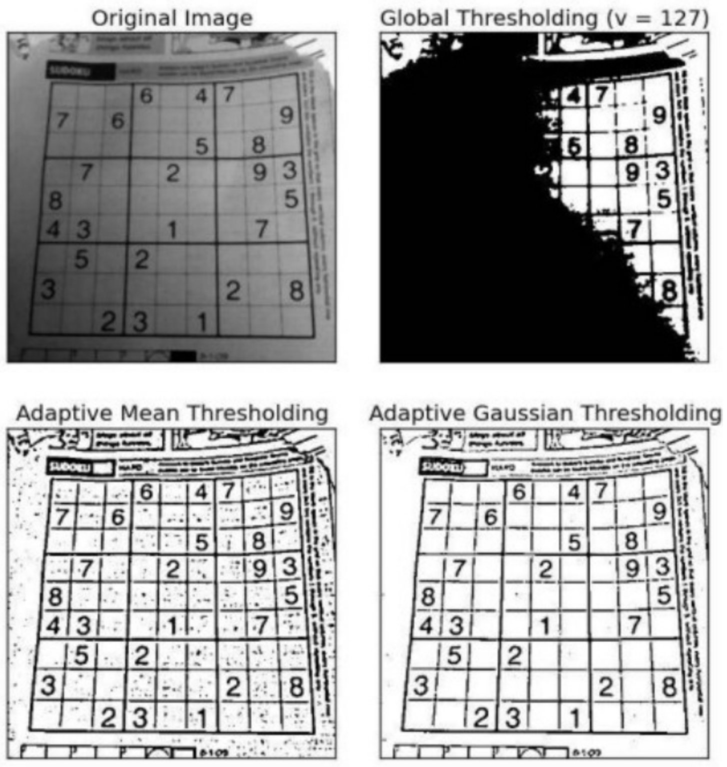

Original Image
Global Thresholding (v = 127)
Adaptive Mean Thresholding
Adaptive Gaussian Thresholding

Figure 5: Different thresholding methods' effects. Source: [8]

One of the options is to refer to the metadata of an image, inferring the median, mean and mode of its pixels' saturation. Those pixels which are more saturated (i.e. lighter) than the value computed from the metadata do surely belong to the background, whereas those less saturated should be further examined with more advanced binarization techniques, such as adaptive thresholding. Since it is more effective to compute the metadata aggregations on images of smaller resolution, the thresholding value is calculated on a thumbnail of an averaged image by the following empirically obtained by us formula:

$$\frac{3 \times (mean + median) + mode}{7}$$

While it was obvious that adaptive thresholding would be necessary to approach the background extraction problem, it wasn't clear which of the thresholding values to choose: the mean of neighborhood area (ADAPTIVE_THRESH_MEAN_C) or the weighted sum of neighborhood values where weights are a gaussian window (ADAPTIVE_THRESH_GAUSSIAN_C). Empirical studies has shown a slight advantage of the weighted sum value method in filtering out background noise, but it was about twice as consuming in terms of computational time as the mean value method. Therefore, it was decided to utilize the latter one for the sake of time efficiency. The combination of both metadata and adaptive mean thresholding methods enables, on the one hand, to filter out large defects of the background (e.g. large shadows or noisy background) and, on the second hand, to adapt to the lighting conditions of smaller regions of an image.

In effect of thresholding, a binary image is obtained. However, due to lighting imperfections, it is still far from resembling ideally bur-shaped blobs. The edges of the blobs are sharp and might even decompose a bur into two parts due to light color-coding rings or transversal glisters. In order to avert these unwanted artifacts, we would like to smooth the edges and close the breaks. This is effectively done by two-stage median blurring interchanged with dilating. As a result, not only we obtain nicely smoothed and closed blobs, but also widen their area, which will be extremely useful feature against loss of effective pixels further on during cropping the identified blobs.

But before we can crop the identified blobs, we need to identify rectangular regions of the image which best encloses them. Since blobs can hypothetically be of different shape and rotation, this task is not as straightforward as it might seem. Luckily enough, OpenCV Jeżeli to pierwsze wystąpienie OpenCV to proszę dać odnośnik do bibliografii provides us with a built-in function named 'findContours', which returns rectangular-shaped contours. Those contours are calculated to fit a blob with a rectangle of minimal area, thus optimal. It is worth noting that even though the function works for images with black blobs and white background, it is meant to work on inverse images (i.e. white blobs on black background). One of the side effects of incorrect input is yielding the whole image perimeter as a contour which shapes its content.

Having some expert knowledge about the characteristics of the original images which enter the pipeline, we can substantially speed up the processing by discarding the notoriously unsuitable contours in terms of size. Therefore, there were created three different filters which check whether a contour has achieved a minimum pixel number and whether it takes a proportion of an image which lies between specified limits. If a contour satisfies all the three conditions, it is chosen for cropping and further processing. Otherwise, it is just discarded as improper.

If either OpenCV or NumPy (a package for scientific computing with Python) [9] provided us a functionality for cropping tilted images, we would simply need to find the coordinates of its contours' corners and supply them to the necessary library function. But since such functionality is not yet know to the author, the pipeline had to be extended with an image rotation step which would rotate the image around the center of a contour so that it becomes positioned vertically. For that purpose, the contour's center coordinates and its' rotation angle needed to be found applying rather simple trigonometric formulas on finding the angle between a cathetus and a hypotenuse (by denoting the longer side being the hypotenuse). It should be noted that pure image rotation results in loosing of effective protruding pixels in the corners of the input image due to artifactual cropping. Therefore, a rotated image should be added margins which would enable all the effective pixels to always remain unaffected by rotational corner cutting. The latter is achieved by overlaying the original image at the center of a previously calculated greater canvas.

Finally, now we are able to crop the contoured regions of an original image. Having calculated the coordinates of contours' corners in a rotated overlaid image, we simply use NumPy array slicing and save a sliced image in a corresponding folder of cropped contours for the given original image. It is important to observe that single original image may eventually lead to emergence of a number of cropped regions. That is why the latter ones have to be properly indexed in the destination directory.

In effect, we achieve a bunch of cropped small margined image regions which potentially enclose the objects of our interest – dental burs. However, oftentimes they contain some other darker objects from original image which are parts of background of a suitable proportion. In order to truly detect our rotary instruments, we should utilize the expert knowledge about their shape. As we intuitively know, all of them are prolonged (of a low inertia ratio), highly convex and slightly circular (figure A.1). Feeding this information as parameters of an OpenCV's built-in 'SimpleBlobDetector_create' function, we are able to differentiate between proper bur-shaped objects and other noise objects. It is also necessary to keep in mind that previously cropped images, apart from carrying improper objects, may contain several "glued together" burs at once, or might be wrongly rotated due to strong shade and excessive dilation effects. Therefore, those images need to be further refined by repeating all the pipeline stages – from grayscale conversion down to image cropping.

Simultaneously with contour finding and blob detection, the cropped images are marked up for demonstration and testing purposes. Their contours are denoted with red rectangles, whereas the centers of successfully detected blobs are marked with red circles. Those marked-up images are subsequently saved in a separate catalog. The contours inside of which there was detected a properly-shaped blob are chosen to be rotated, cropped and saved to a folder with refined images. The refining stage runs much faster than for original image as its size is usually several hundreds times smaller. Moreover, it is also more accurate as it disregards the peripheral features of an original image. After the refining stage, images are expected to contain single vertically oriented burs, which would constitute a normalized input for further machine-learning-based classification tool which is left out as a future work.

3 Implementation Details

For the needs of this project, a general-purpose programming language Python of release 2.7.12 was used [10] . It was chosen due to its simplicity, rich functionality, a large community of users, numerous forums and good documentation, which are the key to creating a fully functional working prototype within a limited time frame. Above Python, there was used a package for scientific computing NumPy of version 1.15.1 and open-source computer vision library OpenCV, version 3.4.2. The packages were installed utilizing Python package manager PIP, version 18.0.

The project source code consists out of numerous functions which are contained in four modules:

- 'start.py' – the main module which contains the path variables, launches the whole program and tracks its execution time;
- 'img_processing.py' – constitutes of basic functions for initial image processing, such as color inversion, grayscale conversion, and background extraction;
- 'blob_cropping.py' – the heart of the program, takes care of finding and rotating contours, as well as of initial image cropping;
- 'bur_detection.py' – contains the functions needed for the refining stage of image processing: blob detection, making sure the detected blob is of a proper characteristics and lies within a contour.

At the program runtime, there is verbose notification of the processing state. It puts to the standard output logs regarding which image is being currently processed, at which processing stage it is, how long each stage takes and to which directory the resulting images are saved. By default, the output images are placed into three separate folders with a tree-like structure, retaining the hierarchy along with the file and sub-directory naming from the source directory:

- 'cropped' – an auxiliary catalog that includes images after the initial stage of their processing and cropping, serving not only as an input for the the markup and refining stage, but also for demonstration purposes;
- 'markups' – an auxiliary catalog that includes marked-up images, with red rectangles representing the contours and red dots representing the centers of detected blobs obtained at the refining stage, which serve exclusively for demonstration purposes;
- 'refined' – the catalog containing the final version of resulting images after their detection. Files from this catalog were used to determine effectiveness and accuracy of the solution.

4 User Manual

The project was designed to be platform independent. The development and testing were performed in 64-bit Ubuntu 16.04 LTS OS environment. Before running the program, it is necessary to unpack the source Python files into a desired folder and specify the path variables (either absolute, or relative to a current folder) in the 'start.py' file:

- source_dir – the path to a folder with images to be detected by the program;
- cropped_dir – the path to a folder to save cropped images based on preliminary blob analysis;
- markup_dir – the path to an auxiliary folder to save marked-up images from the 'cropped_dir';
- refined_dir – the path to a folder to save refined images based on blob detection analysis.

Once all the packages are installed and path variables are set, it suffices to run the 'start.py' file with Python interpreter, for example by executing 'python start.py' command in bash. The 'source_dir' folder will be recursively searched for '.jpg' files, which will be put into a processing pipeline one at a time. Corresponding sub-folders for cropped, marked-up and refined images would be automatically created and filled with resulting images.

5 Use Cases and Effects

Even though it was initially assumed that the original pictures would be of a good resolution and illumination, made against a uniform white background, we decided to put the bur detector into a real-life test, and therefore prepared a versatile self-made testing dataset of 639 images. Most of the pictures were shot in deliberately noised conditions with a Samsung Galaxy J5 (2016) phone 13 Mp camera. Nevertheless, after we had manually examined 779 resulting images, 77.15% of them turned to be correctly detected, well positioned and cropped bur images. At the same time, as much as 94.05% of original burs were successfully found, with only less than 6% being either lost or significantly distorted during processing (figure 6). On average, it took about 2.71 sec for an image to get through the initial cropping stage and another 0.16 sec at the markup and refining stages, which sums up to 2.87 sec per each 13 Mp image to get fully processed. The time might variate depending on the number of burs in a picture, background noise and illumination conditions.

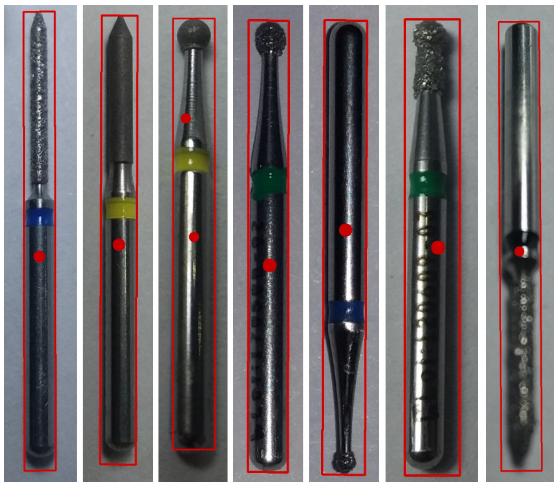


Figure 6: Marked-up examples of properly detected burs at close to normal shooting conditions

However, some of the testing images had highly non-uniform background which was eventually falsely detected as good blobs (figure 7b, 7c), and few were even recognized as bur-shaped blobs (figure 7a). As it can be seen from the figure 7, the implemented system of proportion and “bur-shapeness” filters attributed to diminishing the fraction of background parts falsely detected as burs. As a matter of fact, only 18.59% of the resulting detected images were background noise which could be avoided providing satisfactory background conditions.

Illumination conditions were the second most problematic issue for the system accuracy. Some pictures were taken with a bright source of light from a side, which casted strong background shadows beside instruments (figure 8). Some other were taken using an in-built flash light of the camera, which generated strong glisters on the smooth shank surface of dental burs. These two factors brought about a challenge with correct delimitation of the blobs – shadows were

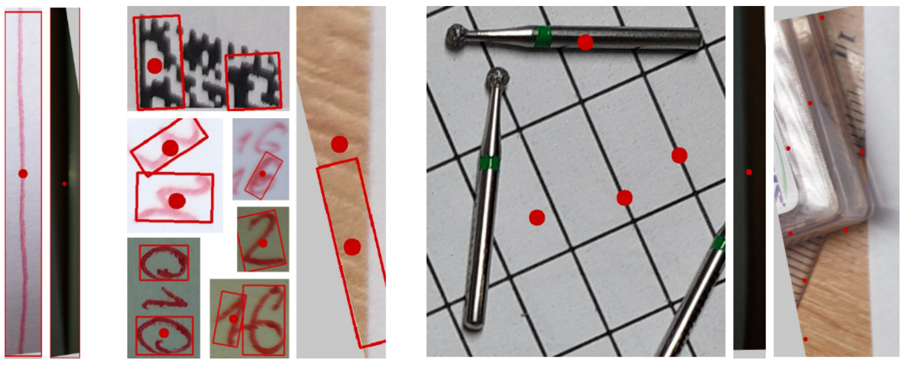

(a) (b) (c)

Figure 7: Marked-up examples of improperly detected blobs due to background noise: a) false positives, b) successfully filtered out by proportion filters, c) parts of background were detected as bur-shaped, but not recognized as separate blobs

perceived as continuation of burs, whereas transversal glisters on the narrow parts of the instruments were recognized as a separation between blobs and cut images in two parts (figure 9a). Part of the testing bur images, despite normal lighting, were also mistakenly taken as two burs due to their light color coding bands. Most prominent problem was noticed on yellow bands which after thresholding processing step tended to merge with background (figure 9b). Altogether, 5.16% of the testing images were detected with some defects: either fractured or with significant deviation from the vertical axis.

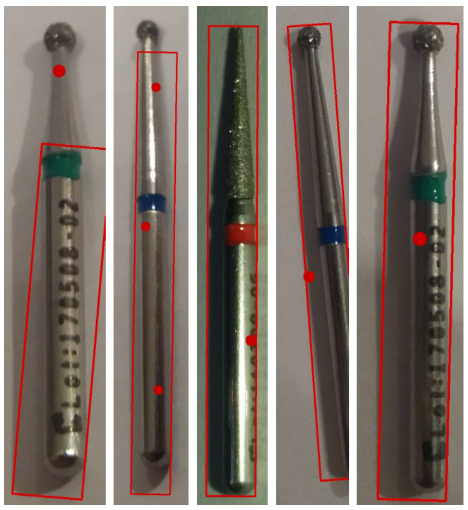
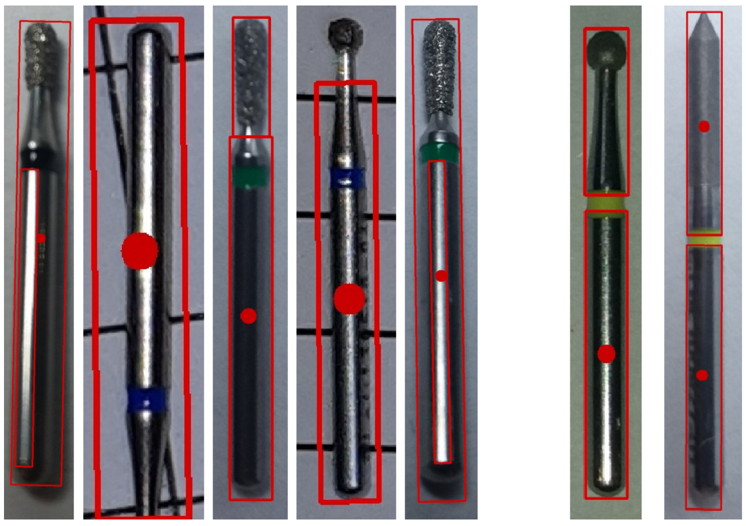


Figure 8: Marked-up examples of improper detections caused by shadow bias

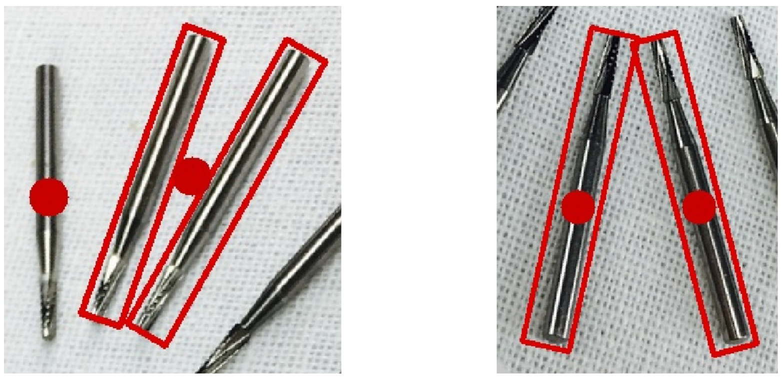
As part of the system efficiency testing, there were attempts to recognize multiple burs situated close together to one another. Even though there were some complications related to sticking the dilated blobs together (figure 10a), the subsequent refining stage proved to be very efficient in separating such clusters



(a) (b)

Figure 9: Marked-up examples of improper detections caused by:
a) flash lighting bias, b) failure to recognize color band as part of a bur

(figure 10b). Moreover, in numerous cases the refining stage managed to correct minor misconceptions of the previous cropping stage and locate instruments more precisely (figure A.2).



(a) (b)

Figure 10: Marked-up examples of images with multiple burs close together:
a) burs stick together in effect of a dilation procedure, b) ability of refining stage to separate glued burs coming from cropping stage

Only 0.78% resulting detections appeared to be false negatives, i.e. burs were not recognized despite being in a picture. On the other hand, more than 94% of instrument images were localized and cropped properly. Insignificant portion of those properly detected were directed upside-down due to lack of a working part detection mechanism.

6 Conclusions and Future Work

The work carried out in this thesis has indicated decent results in detecting dental burs in unprofessional medium-quality images. The proposed implementation of the pipeline of computer vision methods appeared to detect more than 94% of original pictures, and to provide the detection accuracy rate of more than 77%. Most failures and inaccuracies were attributed to the non-uniform background and improper illumination of the instruments. These factors could be further researched in order to mitigate their effect on the detection quality. Some of the solutions to overcome the weaknesses of the current system are:

- converting the oriented image to HSV and using saturation filters in order to detect and classify color bands;
- researching the feasibility of applying a histogram equalization in order to mitigate illumination defects;
- performing Hough transform for proper line detection in order to more precisely detect a bur shank;
- utilizing Haar cascade classifier in order to detect burs based on their overall structure instead of pure blob shapes analysis;
- refactoring the Python source code into C++ programming language code in order to speed up the computations in production stage;
- preparing a set of training data in order to obtain more objective effectiveness and accuracy results.

Not only has this work proved its aptitude to detect dental burs and produce standardized input for a supervised machine learning classifier, but also revealed many promising ideas on how to further develop the project. The following are the major suggestions for future work to make the project a finished product:

- manual labeling the refined images from the initial testing set, based on the direction of the working part;
- utilizing the labeled images from the previous point as a training data input to either a SVM classifier or a CNN-like classifier to recognize whether the working part is pointed upwards;
- rotating 180 degrees the burs pointed downwards to enable them being a training input to the main classifiers;
- designing a hue-based filter for detecting the color of the color bands;
- designing and training of a shape-oriented CNN in order to classify bur shapes coming from a limited training range (i.e. ball, peak, cylinder, etc.);
- designing and training of a size-oriented CNN to classify the diameters of the working parts of dental burs;

- combining the classifiers and integrating them into one pipeline, testing their overall classification accuracy;
- in case the accuracy is below the acceptance rate, scrapping the Internet in order to obtain more training samples, and labeling them;
- creating a user-friendly interface (perhaps a mobile application), which would collect new samples of burs and their pre-confirmed labels, provided by users; the application would work in a cloud, receiving a user-made picture and returning a prediction for ISO-code to be further displayed next to the bur image in the user interface.

7 Appendices

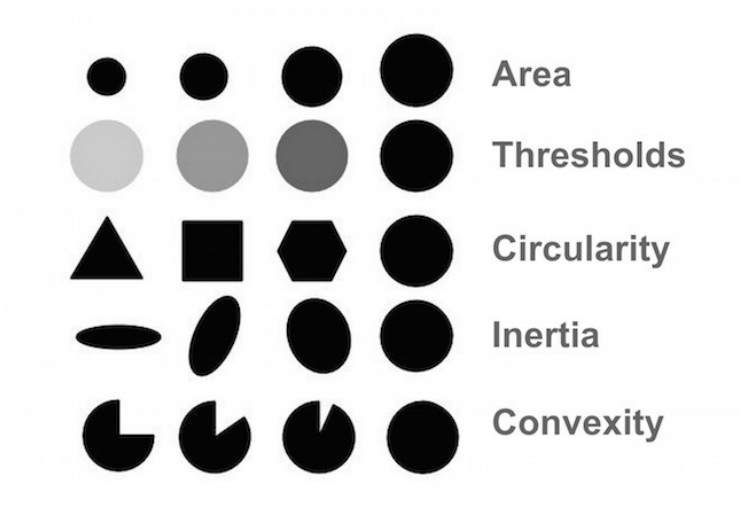


Figure A.1: Demo of input parameters' meaning for OpenCV 'SimpleBlobDetector_create' function. Source: [11]

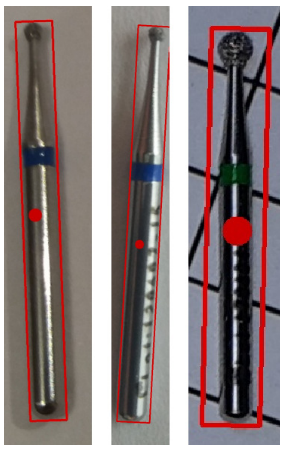


Figure A.2: Marked-up examples of the effectiveness of the refining stage in improving the misconceptions of the cropping stage

8 References

[1] https://www.aliexpress.com/item/100-PCS-Dental-Diamond-Bur-for-High-Speed-Handpiece-1-6MM-Size/32537269249.html

[2] https://www.gessweincanada.com/product-p/128-0010m.htm

[3] Letouzey A., Decrouez M., Agustinos A., Voros S. (2016). *Instruments Localisation and Identification for Laparoscopic Surgeries.* Grenoble: Univ. Grenoble Alpes, SurgiQual Institute, INSERM.

[4] Szeliski, R. (2010). *Computer Vision: Algorithms and Applications.* London: Springer.

[5] Beyeler M. (2017). *Machine Learning for OpenCV: Intelligent image processing with Python.* Birmingham: Packt Publishing.

[6] Nikolenko, S. et al. (2018). *Glubokoe obuchieniye. Pogruzheniye v mir nieyronnych sietiey.* Saint Petersburg: Pitier.

[7] https://opencv.org/

[8] https://docs.opencv.org/3.4.0/d7/d4d/tutorial_py_thresholding.html

[9] http://www.numpy.org/

[10] Matthes, E. (2015). *Python Crash Course: A Hands-On, Project-Based Introduction to Programming.* San Francisco: No Starch Press.

[11] https://www.learnopencv.com/blob-detection-using-opencv-python-c/