An Automated Approach to Mitigate Transcription Errors in Braille Texts for the Portuguese Language

André Roberto Ortoncelli
Federal University of Technology - Paraná - (UTFPR)
Dois Vizinhos, Paraná, Brazil
ortoncelli@utfpr.edu.br

Marlon Marcon
Federal University of Technology - Paraná - (UTFPR)
Dois Vizinhos, Paraná, Brazil
marlonmarcon@utfpr.edu.br

Franciele Beal
Federal University of Technology - Paraná - (UTFPR)
Dois Vizinhos, Paraná, Brazil
fbeal@utfpr.edu.br

ABSTRACT

The quota system in Brazil made it possible to include blind students in higher education. Teachers' lack of knowledge about the braille system can represent a barrier between them and students who use it for writing and reading. Computer-vision-based transcription solutions represent mechanisms for reducing understanding restrictions on this system. However, such tools face nuisances inherent to image processing systems, e.g., illumination, noise, and scale, harming the result. This paper presents an automated approach to mitigate transcription errors in braille texts for the Portuguese language. We propose a selection function, combined with dictionaries, that provides the best correspondence of words based on their braille representation. We validated our proposal on a dataset of synthetic images by submitting them to different noise levels and testing the proposal's robustness. Experimental results confirm the effectiveness of the solution compared to a standard approach. As a contribution of this paper, we expect to provide a method to support robust and adaptable solutions to real use conditions.

KEYWORDS

Computer Vision, Optical Braille Recognition, Spell Checking Methods, Computers in Education

1 INTRODUCTION

The conception of new technologies to assist blind people is a critical challenge of the research community. The World Health Organization (WHO) estimates over 285 million blind and visually impaired people globally, whose 39 million are blind [1]. According to the report "The Conditions of Eye Health in Brazil 2019," prepared by the Brazilian Council of Ophthalmology (CBO), and based on data from WHO reports and indexes from the Brazilian Institute of Geography and Statistics (IBGE), estimates that in Brazil, 1.577,016 individuals are blind, equivalent to 0.75% of the population [2].

The quota system in Brazil allowed blind students to have access to the University. Despite the social advance in recent years, the current teaching model brings many difficulties to realize these students' inclusion. One of these difficulties relies on written communication between teacher and student, typically made using braille, a tactile reading and writing system.

The braille system [3], proposed by Louis Braille in the middle of the 19th century, is a universal writing and reading system. Each character representation uses a rectangular cell of palpable dots, arranged in a 3 x 2 fashion, and presented in Figure 1. To express letters, numbers, punctuation marks, and other symbols are defined as a combination of 6 dots, i.e., 64 combinations. In Figure 2, we depict some examples of the braille alphabet for the Portuguese language.

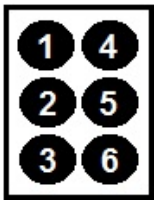

Figure 1: A full braille cell. Each dot on the cell must have a corresponding position on the braille map code, starting from the top-left position following the numerical order.

The vast majority of teachers do not know how to read braille and face comprehension problems, which difficulties evaluating their students' activities. To improve communication and interaction between the teacher and the blind student is a crucial feature of the teacher-student relationship.

Computer-vision-based solutions can help to recognize students' texts reducing such restrictions. In the literature, we find many approaches to deal with the optical braille recognition (OBR) task [4], i.e., transcribe a braille text into an alphanumeric representation [5–7]. These solutions must handle two significant problems on the images: the preprocessing and the segmentation stages. The first stage deals with inherent nuisances on the images, e.g., color, luminance, and contrast variations, as well as with noise, scale, and rotation variations that harm the recognition process. The second stage segments each braille cell. Both are complementary stages, and the first implies strongly on the second.

As an indefectible process is almost inconceivable, some transcription errors eventually occur, especially when dealing with true images. Dictionaries can help with such transcription errors. However, spell-checking solutions use probabilistic aspects on a recommendation, that are strictly related to orthographic issues [8].

This work proposes an automated approach to mitigate transcription errors in braille texts for the Portuguese language. We claim that using a braille-cell-oriented solution can provide a robust tool to OBR systems, principally related to Computer Vision (CV) recognition errors. To the best of our knowledge, this is the first braille-centered approach in the literature.

Our proposed pipeline for transcribing braille image texts into plain text, and suggesting likely words to mitigate recognition errors, has three stages: the recognition, the revision, and visualization stages. Given an image of a braille text, our pipeline converts

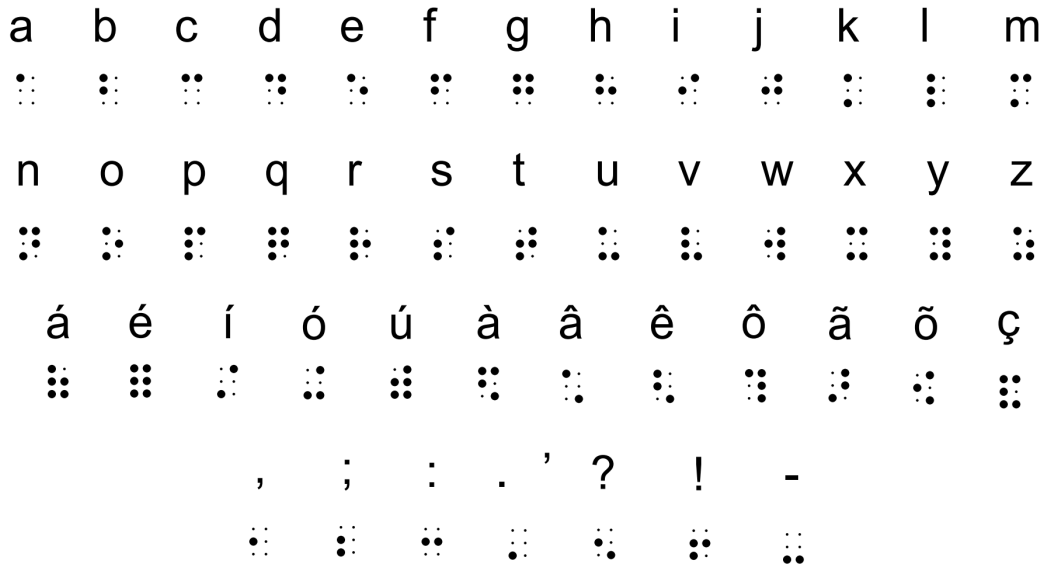

Figure 2: Examples of letters and symbols of the braille alphabet for the Portuguese language.

braille cells into characters by preprocessing the image, then segmenting into lines and letters, and finally converting them into alphanumeric values, following the braille code (e.g., Figure 2). The recognition stage outputs a transcribed text that serves as input to the revision stage, verifying the transcription result in the dictionary and proposing a correction when required. The visualization stage is conceptual, thinking in an application scenario, and in this work, only saves the transcription results in a text file. We depict this process in Figure 3.

2 PROPOSED APPROACH

In this section we detail our proposed apprach in depth. In Section 2.1 reports the computer vision method employed on the recognition stage, and Section 2.2 describer our proposed approach for revising recognition errors in braille texts. The source code of our porposed pipeline is available on https://github.com/ICDI/braille-spellchecker.

2.1 Recognition stage

In this work, the transcribing process of a braille text into the Portuguese language has three steps: preprocessing, lines and columns (characters) segmentation, and conversion of the image representing the braille cell to the corresponding alphanumeric symbol[1].

2.1.1 Preprocessing. this step prepares the input image for the segmentation process. In this work, as we adopted synthetic images, preprocess the images is relatively more straightforward. However, in circumstances that demand the recognition of images captured from printed texts, this step is fundamental for the application's success. It requires more refined and adaptable techniques to the problem. We first remove noise from the images, using the following

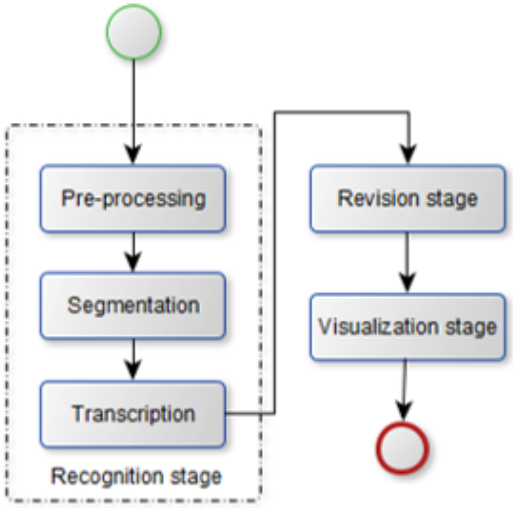

Figure 3: Flowchart of our proposed pipeline. Given a braille document image as input, our proposal preprocesses, segments, and transcribes braille cells into alphanumeric characters. Then we perform a braille-centered revision stage, and finally, the output text can be used for visualization.

image processing techniques: 1) Image thresholding with a value fixed at 120, found empirically; 2) Median filter with 5×5 filter; 3) Morphological opening filter with a squared structuring element of 3×3; 4) Blob removal (area < 10).

2.1.2 Segmentation. in this process, we first segment horizontally (rows) and then vertically (columns). Both follow the same principle, and we show the complete procedure in Figure 4. Starting

[1]Our code enhances the code provided in https://github.com/MUSoC/Braille-OCR, involving the adaptation to deal with our dataset's images

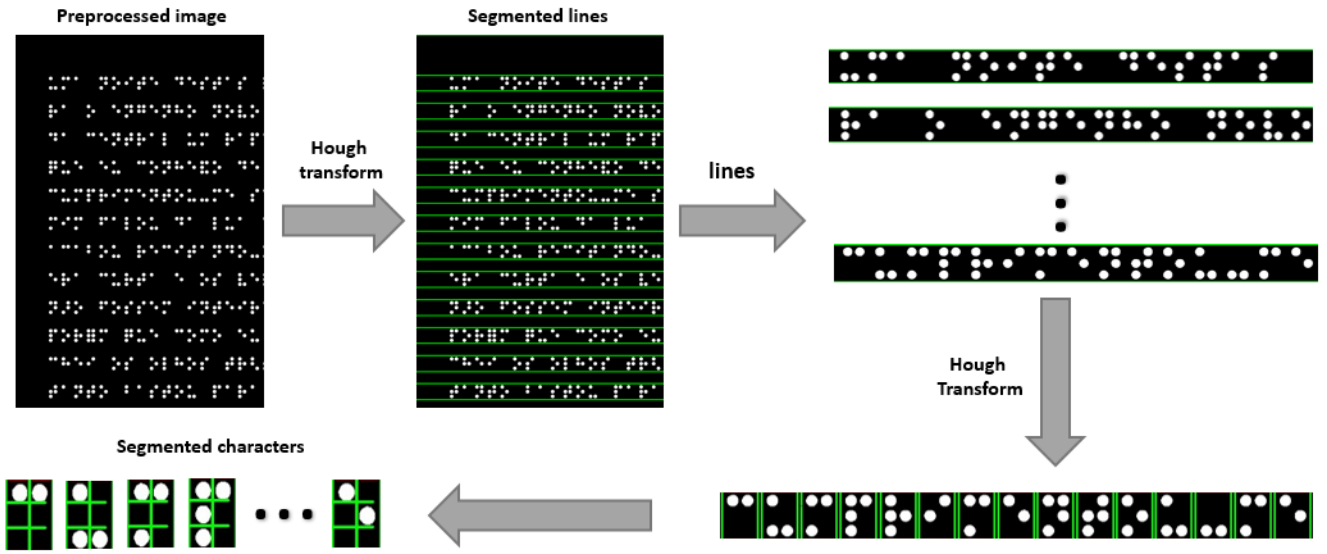

Figure 4: Segmentation pipeline. The algorithm segments page line by starting from a preprocessed image by extracting horizontal Hough segments and cropping the image. After, we perform a similar approach to segment chars by detecting vertical Hough lines. Finally, we split each character segment into points to transcribe the correspondent alphanumeric value.

from the image resulting from the pre-processing, initially, we apply a morphological expansion filter with a structuring element of 10 *timesl*, where l = page width. After that, we perform a Canny filter for edge detection, and finally, we detect the lines that horizontally segment the images, using the Hough line transform. After segmenting rows, we perform a similar process for column segmentation, i.e., the braille characters. Finally, we divide each image portion corresponding to a braille cell into six regions that correspond to the points. We convert each detected dot to a position on the six-position binary vector.

2.2 Revision stage

To understand our proposed revision method is essential to understand some assumptions about the domain of our work. Given the word denoted by a set of letters $P = [l_1, l_2, ..., l_x]$, each letter is part of a pre-defined set of characters. A text is a set of words denoted by $T = [p_1, p_2, ..., p_n]$. Like a text, a dictionary is also a set of words denoted by $D = [p_1, p_2, ..., p_m]$.

In this work context, two aspects separate a text from a dictionary: i) a text can have repeated words. A dictionary has only one instance of each term; ii) in a dictionary, every element is grammatically correct. We use the words in a dictionary as a parameter to correct possible errors in a text.

We can transcribe each letter of a word $l_x \in P$ into the Braille system. The function $\beta(P)$ takes a letter P and returns a vector of such size $|P| * 6$. Each position is a binary value that corresponds to each of the six points of a braille cell. Similarly, the $\psi(\beta(P))$ function receives a braille cells vector and returns the word transcribed in the target language. In Figure 5 we present an example of the transcription pipeline of a set of braille cells into a word.

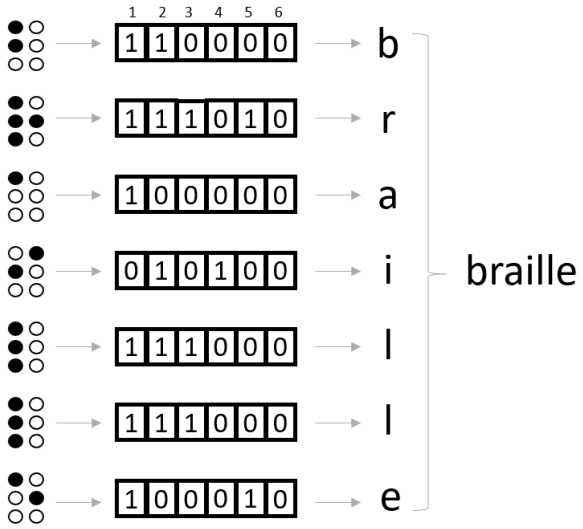

Figure 5: Transcription example of the "braille" word. Each braille cell has six dots that represent a binary code of the same length. We then map each code list into an alphanumeric character.

The proposed method takes a text (T) and dictionary (D) and uses D as a reference to correct possible grammatical errors in the words of T.

The algorithm uses the function $compare(p_t, p_d)$ to compare a word $p_t \in T$ with a word $p_d \in D$ (where $|p_d| = |p_t|$). Equation 1 defines the value returned by the *compare* function.

$$compare(p_t, p_d) = \sum_{i=1}^{|p_t|*6} |\beta(p_t)[i] - \beta(p_d)[i]| \quad (1)$$

To minimize the Equation 1, the algorithm calculates the best match for each word $p_t \in T$. Then we check for each word $p_t \in T$, the value corresponding to all words $p_d \in D$, with the same lenght. If p_d minimizes this equation and is equals p_t, then we consider p_t grammatically true, otherwise we replace the referred word. Algorithm 1 details the calculus of the $compare(p_t, p_d)$ function, and Figure 6 presents a graphical example of using the algorithm with | T | = 1 and | D | = 2.

Algorithm 1 Revision on the trascription result.

```
Require: T, D
for i = 0 to T.size() do
    min = ∞
    aux = 0
    for j = 0 to D.size() do
        if (T[i].size() == D[j].size() then
            diff = compare (T[i],D[j])
            if (diff < min) then
                min = diff
                aux = j
            end if
        end if
    end forT[i] = D[j]
end for
return T
```

3 EXPERIMENTAL EVALUATION

We evaluate our revision proposed method's robustness in two sets of experiments. In both, we imposed errors and submitted them to our proposal. In experiment A, we randomly added different percentages of error in each word, working directly with the text, so we do not recognize the braille character in images. In experiment B, we executed the full pipeline on synthetic images of braille texts, i.e., as described in Section 2. We adopted synthetic images in detriment to real ones to give more control to the recognition process and consequently have more reliable results on the revision proposal.

We compared our results with those obtained by the pySpellChecker library [8]. We contrasted our results with this library because it uses the Levenshtein distance, a metric commonly used by spellcheckers [9]. Besides using the Levenshtein Distance, the pySpellChecker method relies on a word frequency list (a dictionary with word frequencies). Higher frequency values represent the most frequently used words, so they are more likely to be the correct results. Another feature of pySpellChecker is that if a searched word is not in the dictionary, it returns a suggestion, if and only if a cost function is lower than a threshold. Otherwise, this library does not suggest any word as a correction.

We detail the experimental database in Subsection 3.1. In Section 3.2, we present the evaluation metrics we use on the experiments. Sections 3.3 and 3.4 present the experimental setup as well as the results and discussions concerning the experiments A and B.

3.1 Dataset

We carried out experiments based on ten texts and one dictionary. We got these texts from classic literature books of public domain[2]. Table 1 shows details of the books, with name and author. We select only the first chapter (poem, scene, or act) of each book in the experiments.

Table 1: Selected books and respective authors, employed to build our experimental dataset.

Book	Author
Dom Casmuro	Machado de Assis
Iracema	José de Alencar
Poemas de Álvaro de Campos	Fernando Pessoa
Quincas Borba	Machado de Assis
A Morte do Lidador	Alexandre Herculado
Macbeth	William Shakespeare
A Dívida	Artur Azevedo
Alma Inquieta	Olavao Bilac
A Dama do Pé-De-Cabra	Alexandre Herculano
A Igreja do Diabo	Machado de Assis

As for the dictionary, as we compare our method with the pySpellchecker library, we select the dictionary recommended on that library website, which has a link to the repository of the Word-Frequency project[3]. This repository has dictionaries in different languages. We selected the dictionary in the Brazilian Portuguese language, which has 848,043 words accompanied by the frequency list.

We preprocessed the dictionary by removing words containing special characters not belonging to the Portuguese language, i.e., those different from accented letters (á, à, â, ã, é, ê, í, ó, ô, õ, ú, and ç), and the punctuation characters hyphen (-) and apostrophe (‘ or ’). This processing step removed 12,400 words from the dictionary. We kept these characters because they are common in the writing words in the Portuguese language. This chosen dictionary is the reference for the pySpellChecker library, and besides the words, it also presents their frequency list. In our trials, we consider the full version (i.e., words and frequencies) for pySpellChecker, and only the word list for our approach. The dataset we use in this paper is available on http://link_ommited_due_revision_purposes.

3.2 Evaluation Metrics

To compare the results of the experimental instances, we adopted two metrics: the Levenshtein distance and the percentage of correctly transcribed words (or hit rate).

The Levenshtein distance (or edition distance) calculates the minimum number of operations (inserting, deleting, or replacing a character) required to transform one string into another. This is a commonly used metric in spellcheckers, as it is useful in determining how similar two strings are [9].

[2]Books collected from the website: http://www.dominiopublico.gov.br/
[3]https://github.com/hermitdave/FrequencyWords

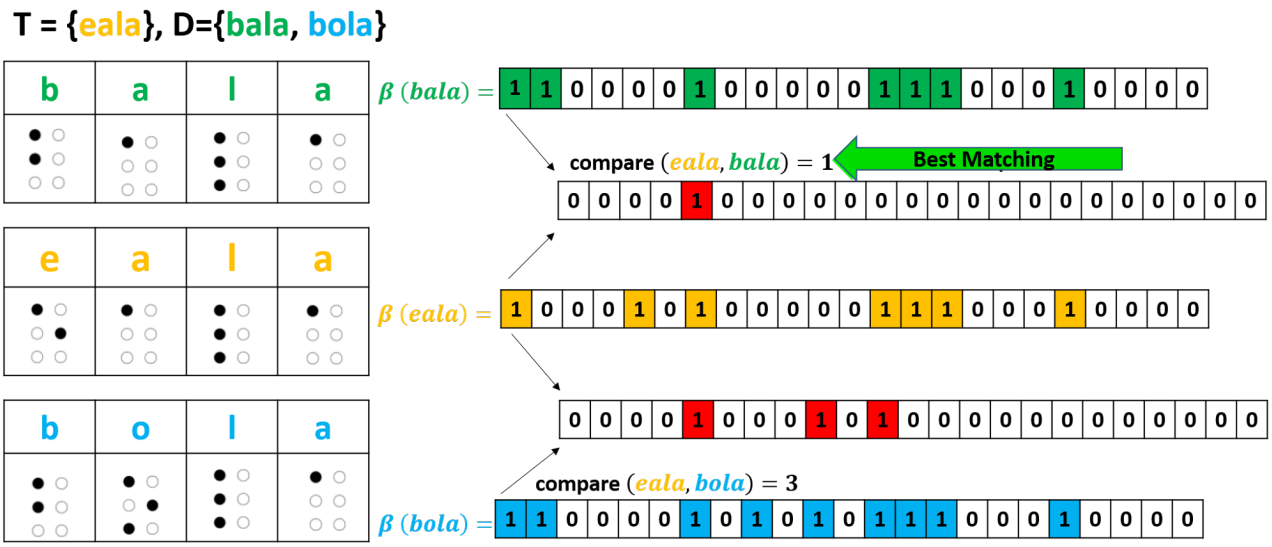

Figure 6: Example of the proposed algorithm's execution, with |T| = 1 and |D| = 2.

The hit rate is a straightforward metric that computes the number of words correctly transcribed/corrected concerning the ground-truth. How higher this value, the better the experimental results. With the Levenshtein's distance, lower values are better. In unlikely to get all the words right, so lower values for the Levenshtein distance show that the words incorrectly transcribed are more similar to the reference set.

In addition, we employed three measures to understand the behavior of the proposed method in the experiments: char recognition error, word recognition error, and words found in the dictionary. The char recognition error relates to the recognition algorithm. By the same logic, the word recognition error shows the words correctly detected. These metrics allow analyzing quantitatively the impact of each type of noise used in experiment B. The percent of words in the dictionary represents the number of recognizable words by the dictionary. This metric plays important information by denoting the upper-bound values for each experiment.

3.3 Experiment A: random error

For each text on the dataset, we create a set T with all its words. Then we convert each word $p_x \in T$ to a six-bin-code representation, using the β function. Later, we impose an error on this code vector, flipping the values of $\beta(p_x)$ randomly. For instance, considering word "computação" (with length 10), and $|\beta(\text{"computação"})| = 60$, if we add 10% error, it means that the values of 6 positions (10% of 60) will be changed.

After adding the error in each word $p_x \in T$, we convert the *corrupted* vector back to a word using the ψ function. Finally, for each word, we executed a correction proposal algorithm on the induced grammatical error.

We test tweve imposed error instances (from 2.5 to 30% with a step of 2.5%) in every text on the dataset. With these values, it was possible to assess the proposed method with higher error rates, which were more significant than the errors caused by the noise added in experiment B (Subsection 3.4) that simulate real braille transcription problems. Table 2 present the results of experiment A. The first column shows the percentage of errors added in each word. The second group of columns shows the average Levenshtein distance. And the last group of columns presents the hit rate.

Table 2: Results of experiment inducing random error to the texts. We compare our method (ours) with pySpellChecker [8] library on the Levenshtein distance and hit words percentage metrics. Best values on each metric in bold

% of error added	Levenshtein dist.		% of hit	
	ours	pySpell	ours	pySpell
2.5	**0.10**	0.12	**96.3**	89.9
5.0	**0.21**	0.77	**86.3**	52.0
7.5	**0.44**	1.45	**70.5**	30.1
10.0	**0.87**	1.98	**45.9**	13.9
12.5	**1.25**	2.39	**34.0**	12.3
15.0	**1.62**	2.63	**26.9**	12.2
17.5	**2.16**	3.10	**10.0**	0.6
20.0	**2.43**	3.25	**6.4**	0.7
22.5	**2.73**	3.44	**4.0**	0.2
25.0	**2.96**	3.60	**2.6**	0.4
27.5	**3.14**	3.70	**2.1**	0.3
30.0	**3.35**	3.81	**1.4**	0.2

Figures 7 and 8 presents the results of Table 2 as graphs. The x-axis of the graphs refers to the percentage of error added in each experimental instance. The y-axis shows the results of each of the metrics. Figure 7 represents the average Levenshtein distance for each word of the texts, and Figure 8 represents the percentage of the hit.

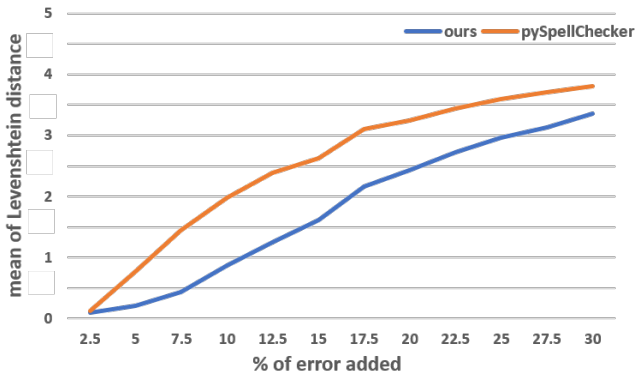

Figure 7: Results of experiment inducing random error to the texts. We compare our method (ours) with pySpellChecker [8] library on the Levenshtein distance. Higher values are better.Our proposal outperforming the other competitor in every tested situation.

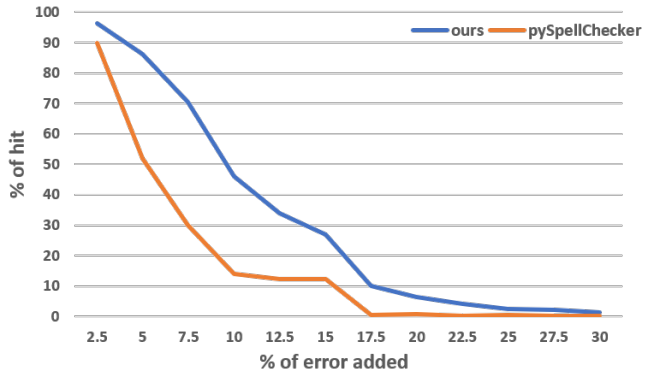

Figure 8: Results of experiment inducing random error to the texts. We compare our method (ours) with pySpellChecker [8] library on the hit words percentage. Lower values are better. Our proposal outperforming the other competitor in every tested situation.

3.4 Experiment B: noise in synthetic image

To synthetically produce braille documents from the original texts, we first performed a preprocessing step similar to the previous experiment. Besides the char removal, we also got rid of digits and uppercase letters. This last preprocessing step guarantees non-extra-braille characters addition because, in the braille system, number and capital letters representation demand a previous extra char.

We converted the preprocessed texts to braille with the Braille Fácil software [10]. We converted the output braille texts to plain white images with black dots. We also imposed some transformations on the pictures to simulate some variability on the data and test our recognition algorithm. We carried out six image transformations on the Gimp software [11]: in the first two, we applied Gaussian blur filters with sigma values of 3.0 and 5.0; in the other two, we randomly spread the braille dots with a variation value of 10 and 20 pixels on both vertical and horizontal direction; the last two correspond to the combination of several processing techniques, producing the artistic filters named photocopy and canvas. In Figure 9, we present the transformation we made on the images applied to a braille document segment.

The error rates in the transcription process of experiment B are in Table 3 - each line refers to a type of noise added to the images. The last two columns represent the percentage of error in chars and word recognition, respectively.

Table 3: Recognition of braille characters/words on the proposed dataset. Column chars refers to the recognition rate of our recognition stage. Columns words refer to the word ratio returned as true on the dictionary

	Recognition error	
	chars	**words**
Plain	100.0	97.4
Gaussian blur ($\sigma = 3.0$)	96.6	92.6
Gaussian blur ($\sigma = 5.0$)	95.7	91.7
Spread noise (10)	96.3	92.1
Spread noise (20)	89.5	83.8
Photocopy filter	90.7	90.5
Canvas filter	97.4	87.6

Table 4 presents the results of experiment B. The first column describes the type of noise used. The second column group shows the Levenshtein Distance for each analyzed method. Finally, the last group of columns shows the hit percentage for each technique. In the last line of this table, we present the average of the results.

Table 4: Results of experiment inducing noise to the document images. We compare our method (ours) with pySpellChecker [8] library on the Levenshtein distance and hit words percentage metrics. Best values on each metric in bold

	Levenshtein dis.		% of hit	
noise	**ours**	**pySpell**	**ours**	**pySpell**
Gaussian blur ($\sigma = 3.0$)	0.12	**0.08**	**94.8**	94.1
Gaussian blur ($\sigma = 5.0$)	0.13	**0.08**	**94.3**	93.8
Spread noise (10)	0.17	**0.13**	**93.9**	93.6
Spread noise (20)	0.24	**0.19**	**87.8**	86.8
Photocopy filter	0.19	**0.12**	91.2	**91.5**
Canvas filter	**0.37**	0.39	**89.0**	88.6
Average results	0.20	**0.17**	**91.8**	91.4

Figures 11 and 10 present the results of the Table 4 graphically. In Figure 11 we compare the Levenshtein Distance, and in Figure 10 we compare the percentage of hit.

3.5 Discussion

In experiment A, the proposed algorithm outperformed the pySpellCheker library for every tested instances, considering the two metrics. These results show our approach's robustness when we found

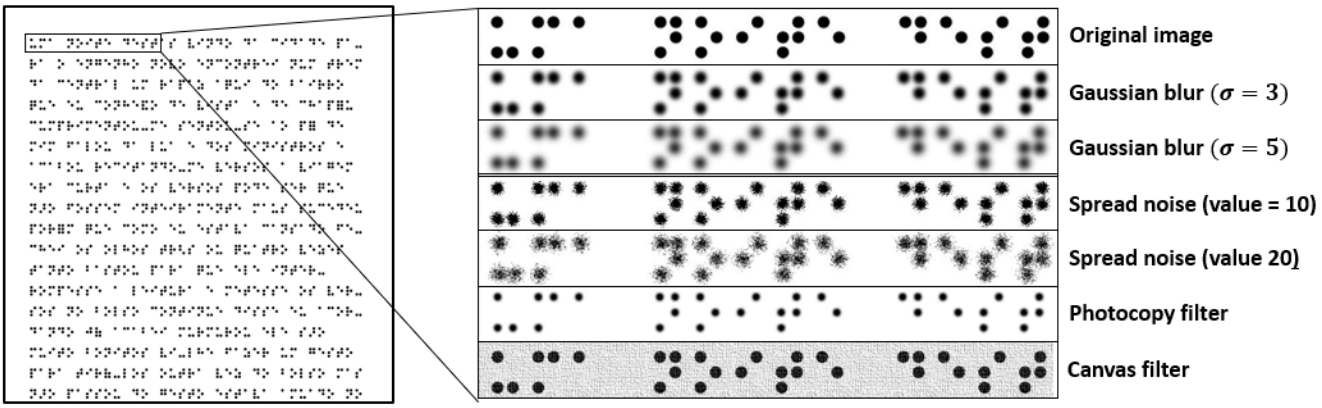

Figure 9: Experimental dataset. We converted the original braille texts into the image representation. Then we applied some image processing filters to induce errors in the recognition algorithm. We present each of these filters with the respective name/parameters.

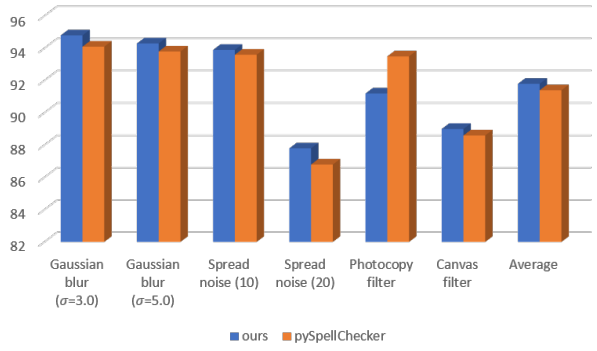

Figure 10: Results of experiment inducing noise to the document images. We compare our method (ours) with pySpellChecker [8] library on the hit words percentage. Higher is better.

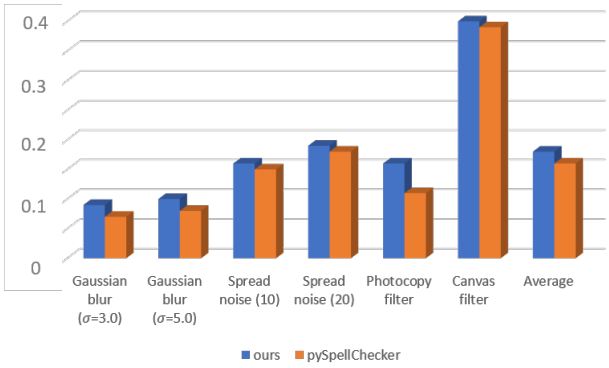

Figure 11: Results of experiment inducing noise to the document images. We compare our method (ours) with pySpellChecker [8] library on Levenshtein distance. Lower is better.

a higher transcription error level, i.e., the higher the percentage of induced error, the results from both methods get worse. Still, proportionally our approach shows far better results than pySpellChecker.

Concerning experiment B, we also outperform pySpellChecker considering the % hit metric (except for the Photocopy Filter noise). Regarding Levenshtein's distance, our method presents higher values. The pySpellChecker algorithm minimizes the Levenshtein's distance, and we expected lower values on such metrics. This behavior did not occur in experiment A because our method's accuracy rate was proportionally higher, affecting the Levenshtein distance results.

The data presented in Table 3 help to understand the behavior of the method in the experiments. This table shows the percent correctly recognized chars and the percentage of words identified with each type of induced noise related to the ground truth that also exists in the dictionary. This percentage of words represents the maximum accuracy that our method can achieve in experiment B. It is impossible to accurately suggest a word if it does not exist in the dictionary. In this context, we found that the more representative the dictionary is about the transcribed text, the better our method's results should be.

The information regarding the character recognition error (shown in Table 3 is essential for understanding the experimental results. The more unsuccessfully recognized characters we have, the more challenging the text review process is.

We should note that the method of reviewing transcribed braille texts proposed in this work compares only words of the same size. Given an entry of size n, our approach only compares terms of length n. This is a limitation of classical methods based on Levenshtein's distance, which we plan to improve in future works.

Despite this limitation, our results outperformed by far on most experimental situations imposed, which confirms that our method is effective and can offer a reliable tool for reviewing texts transcribed from Braille to Portuguese.

4 CONCLUSIONS AND FUTURE WORKS

In this paper, we presented a full pipeline to transcribe and revise braille texts in images. To the best of our knowledge, our proposal is the first braille-cell-oriented approach to correct misrecognized words in texts. Results show that our method outperforms standard dictionary-based approaches. This paper represents a study regarding the revision process, and in further studies, we aim to test our proposal in real-braille-images. However, these initial results show a remarkable potentiality of our approach as a tool for image-based-braille-transcription systems.

Future directions of this seminal work include to test our proposal on real braille-printed images, single and double-sided. We also aim to implement a functional mobile application and evaluate its human-interaction aspects following works like [12]. Another essential trend nowadays in computer vision is deep learning, and proposals like U-Net [13] and BraUNet [7] must be considered to improve the recognition stage of our pipeline. We also aim to work with other dictionaries and extend our proposal to deal with different lengths of resulting words.

We focus our revision method on correct mis-recognized words that are expected to be orthographically correct on the original document. However, when dealing with real human-produced texts, some typos and grammatical errors may occur. Our work can also act in this domain, being an orthographic and grammatical revisor. Finally, we aim to apply natural language processing algorithms to, besides an orthographic suggestion, also consider semantic features to improve our revision stage.

REFERENCES

[1] World Health Organization. Blindness and vision impairment, 2020. URL https://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment.
[2] Mariana Almeida. As Condições da Saúde Ocular no Brasil. *Revista Universo Visual*, pages 6–8, sep 2019. Available: https://universovisual.com.br/secao/edicoes/pdfs/UV113.pdf. Accessed in: 16/12/2020.
[3] American Foundation for the Blind. What Is Braille?, 2020. URL https://www.afb.org/blindness-and-low-vision/braille/what-braille.
[4] Samer Isayed and Radwan Tahboub. A review of optical braille recognition. In *2015 2nd World Symposium on Web Applications and Networking (WSWAN)*, pages 1–6. IEEE, 2015.
[5] Apostolos Antonacopoulos and David Bridson. A robust braille recognition system. In *International Workshop on Document Analysis Systems*, pages 533–545. Springer, 2004.
[6] Filippo Stanco, Matteo Buffa, and Giovanni Maria Farinella. Automatic braille to black conversion. In *Congress of the Italian Association for Artificial Intelligence*, pages 517–526. Springer, 2013.
[7] Renqiang Li, Hong Liu, Xiangdong Wang, Jianxing Xu, and Yueliang Qian. Optical braille recognition based on semantic segmentation network with auxiliary learning strategy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 554–555, 2020.
[8] Tyler Barrus. Pyspellchecker - project description, 2020. URL https://pypi.org/project/pyspellchecker/.
[9] Frederic P Miller, Agnes F Vandome, and John McBrewster. *Levenshtein distance: Information theory, computer science, string (computer science), string metric, damerau? Levenshtein distance, spell checker, hamming distance.* Alpha Press, 2009.
[10] Instituto Benjamin constant. Braille fácil, 2009. URL http://intervox.nce.ufrj.br/brfacil/.
[11] The GIMP Development Team. Gimp. URL https://www.gimp.org.
[12] Giovanni Maria Farinella, Paolo Leonardi, and Filippo Stanco. A mobile application for braille to black conversion. In *International Conference on Advanced Concepts for Intelligent Vision Systems*, pages 741–751. Springer, 2015.
[13] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.