C-Libras: A Gesture Recognition App for the Brazilian Sign Language

Tiago Trotta[1], Leonardo Rocha[1], Telma Rosa de Andrade[1], Marcelo de Paiva Guimarães[2], and Diego Roberto Colombo Dias[1(✉)]

[1] Universidade Federal de São João del-Rei, São João del-Rei, MG, Brazil
{trotta,lcrocha,telmadelac,diegodias}@ufsj.edu.br
[2] Universidade Federal de São Paulo, Osasco, SP, Brazil
marcelo.paiva@unifesp.br

Abstract. Sign languages are visual representations used by hearing or speech impaired people to communicate between themselves and with other people. There are over 140 sign languages globally, and they can be developed by deaf communities or derived from other existing sign languages. The signs made in this context are not considered gestures but words articulated primarily by the hands, while possibly involving facial expressions and trunk movements, making it far from trivial to understand them. Thus, automatic sign language recognition supported by machine learning has shown significant advancement in recent years. This paper presents a mobile application capable of recognizing gestures representing the letters of the Brazilian Sign Language's (Libras) alphabet. Our methodology has three steps: the construction of the gesture dataset containing all the letters of the Libras alphabet; the training of the machine learning model used in the gesture classification; and the development of the desktop/mobile application used to capture the gestures to be classified.

Keywords: Sign language recognition · Machine learning · Augmented reality

1 Introduction

Sign languages are visual representations based on hand gestures and facial and body expressions. They are distinct from conventional languages, such as English and Portuguese, because they are not auditory and do not require vocal expression. Because they develop within deaf communities, they are considered a natural language, provided with grammar, phonology, morphology, syntax, and semantics. Thus, sign languages are more complex than just a set of hand gestures. Therefore, due to their complexity, recognizing their gestures can be classified as a challenge to be beaten by machine learning techniques.

Supported by CNPq, FAPEMIG and Huawei.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2022
O. Gervasi et al. (Eds.): ICCSA 2022, LNCS 13375, pp. 603–618, 2022.
https://doi.org/10.1007/978-3-031-10522-7_41

There is no universal sign language, and they can vary worldwide. In Brazil, there is the Brazilian Sign Language (Libras). According to research conducted by the Brazilian Institute of Geography and Statistics (IBGE) in 2019, in Brazil, 2.3 million people two years of age or older (1.1% of the population) suffer from hearing disabilities, out of which 22.4% of the people between 5 and 40 years of age know Libras [8].

In 2002 President Luís Inácio "Lula" da Silva sanctioned Law No. 10.436, which regulated Libras and recognized it as a legal and genuine means of communication. Through this law, Libras allows the deaf to use gestures instead of oral communication legitimately. In 2005, federal decree No. 5626 was published, establishing the mandatory teaching of Libras by universities in their undergraduate, pedagogical, and speech therapy courses. Thus, the use of sign language has gained space in Brazil, which has favored the development of deaf and hard-of-hearing children from an early age [9].

Among the many fields of study around sign languages, Sign Language Recognition (SLR) aims to develop computational approaches capable of automatically identifying a sequence of signs and correctly understand their meaning [5]. Many SLR approaches treat the problem simply as gesture recognition, but sign languages are far more complex and nuanced than just collections of specific gestures. A more manageable challenge within SLR would be to correctly classify the alphabet of a sign language, since these gestures are usually much simpler and less context-sensitive.

One of the techniques frequently used in SLR is Machine learning (ML), a subfield of artificial intelligence (AI) that automates the building of analytical models, allowing machines to adapt to various scenarios independently. ML has been applied to solve various problems of our daily lives related to image interpretation, voice command recognition, and even disease identification, such as COVID-19 [12]. In the area of SLR, it has not been different. Several works in the literature address the topic [3,4,7]. However, it is still an open area that demands research both in the preparation of datasets and the training of intelligent models.

Thus, in this paper, we proposed and developed a mobile and desktop application capable of capturing hand gestures and classifying them like letters of the Libras's alphabet using a Support Vector Machine (SVM). We tested the application with different SVM models, trained using different sets of letters and different numbers of example gestures per letter, and discovered that the number of examples per letter affects the model's accuracy. Our tests also show that the number and choice of letters recognized by the model affect accuracy as well.

In Sect. 2 we present and discuss the main sign language recognition (SLR) approaches, in Sect. 3 we explain the development of our application, Sect. 5 contains the results of our tests, and in Sect. 6 we present our conclusions and talk about future work.

2 Sign Language Recognition Approaches

Among the several approaches currently applied to SLR, we can highlight the use of inertial sensors and computer vision. In this section, we talk about the two technologies, technical development solutions, and some work that has set out to create tracking devices.

2.1 Sensor Based Approaches

Sensors have been widely employed in motion detection, so they are helpful in body tracking. In the specific case of sign languages, most gestures are concentrated on the upper limbs, specifically the hands.

The advance in virtual reality has enabled the proposal of several unconventional interaction devices. Among them, we can highlight the gloves (Data Gloves and Power Gloves), widely used in the context of SLR [1,11,15].

Despite gloves having been widely used to track hand gestures, other kinds of sensors have been employed, such as inertial sensors, made up of accelerometers, gyroscopes and compasses [19,20].

One of the great benefits of employing sensors in body tracking is the accuracy achieved. Since the sensors are attached to the user's body, they do not suffer from occlusion issues, something common in image capture-based solutions that employ computer vision.

2.2 Computer Vision-Based Approaches

Computer vision-based tracking techniques allow body tracking without any device or sensor attaching to the user's body, enabling mobility and naturality in movement execution.

Usually, vision-based tracking is performed by cameras, either RGB or infrared, such as the Kinect, which has both kinds of cameras. Among the solutions that only use infrared cameras, we can highlight the Leap Motion Controller [16]. On the side of the spectrum, some solutions only use RGB cameras, of which we can highlight MediaPipe, a framework used to build multimedia applications that require body tracking, allowing the detection of objects, faces, hands, hair segmentation among other features [10]. MediaPipe uses intelligent models developed by Google, using their machine learning framework - TensorFlow and TensorFlow Lite. In this work, we employ MediaPipe to capture gestures, both from webcam feed and from previously recorded video.

Although gloves are predominantly sensor-based, some approaches have explored the use of computer vision, such as colored gloves or colored markers positioned on the wrist or fingertips [6,18]. In 2017, Sadek et al. proposed and developed a low-cost smart glove capable of analyzing the anatomical shape of the hands [14]. However, despite the low cost, the use of the device required computer vision techniques since it used colored markers. The device proposed in 2015 by Rossol et al. also uses computer vision as tracking technology [13]. However, it employs the Leap Motion device as a tracking medium instead of

RGB cameras. The Leap Motion is a device specially designed to perform hand and finger tracking, allowing the identification of hands, fingers, orientation, and palm location.

3 Methodology

This work was developed in three steps, illustrated in Fig. 1. The first step was the construction of a gesture dataset containing examples of the Libras' alphabet gestures to be used as training data for an ML model. This step can be divided into three sub-steps: (1) The capturing of the gestures from videos downloaded from YouTube or recorded by RGB cameras; (2) The processing of the captured footage to extract the hands' joints, using the MediaPipe [10] framework; and (3) The storing of the gestures in the ReBase [2] database.



Fig. 1. Illustration of the methodology.

The second step was training an ML model, more specifically an SVM, to classify new gestures. Finally, the third and last step was the development of a desktop and mobile application, using the Unity game engine [17], to capture gestures from webcams and use the trained SVM to classify them.

3.1 Gesture Dataset Construction

The training dataset is a collection of several gesture examples for each of the letters in the Libras' alphabet, taken from video files recorded by regular RGB cameras. Some videos were recorded by members of the research team, and others were downloaded from various YouTube channels via a Python script using the *pytube* library.

Once downloaded, all the videos were processed by another Python script, using the *MediaPipe* and *OpenCV* libraries, in order to extract the hands' joints, as illustrated by Fig. 2. After extracting the joints' coordinates, the gestures received labels to be used as classes by the SVM. The labels were automatically chosen based on the original video's title, using various regular expressions - for example, a video named "Libras Alphabet - Letter A" would be labeled "libras_a". Finally, the gestures' coordinates and labels were written into JSON files, so they could be processed and inserted into the ReBase database.

ReBase is a database dedicated to storing body movements, specifically neuromotor rehabilitation movements. All the movements are stored in the form of



Fig. 2. MediaPipe processing to extract the hand's joints.

the rotations of the body's joints over time, which take up much less storage space than video files. In addition, working with joints and rotations means the movements can be recorded by any kind of body tracking device, such as the Kinect or even RGB cameras using MediaPipe, in addition to preserving the anonymity of the data, since no images of the subjects are stored. Even though ReBase was designed with a different context in mind - body movements instead of hand gestures - its data format is flexible enough to be adapted to other use cases.

MediaPipe's hand tracking module defines 21 joints for the hand, while ReBase defines 20 joints for key parts of the body. For this application, 20 of the hand joints defined by MediaPipe were mapped to the joints defined by ReBase, as shown in Fig. 3, thus making it possible to store hand gestures in the database. The joints 4, 8, 12, 16, and 20 were purposely ignored because they represent the tips of the fingers and cannot be rotated. The movements were then inserted into ReBase as JSON files, following ReBase's format, via the Unity ReBase API. For this purpose, we developed a Unity application to utilize the abstractions provided by the API. The application was also responsible for converting the joints' coordinates into rotations.

Pre-processing. Before being submitted to the ML model, the gestures were pre-processed. First, all the rotations were normalized between 0° and 306°, as shown by Eq. 2, to avoid any negative rotations that could result from the conversion process. In addition, the gestures had their durations normalized to *30 frames*, the length of the shortest example. Every gesture over 30 frames had frames removed from different points along its duration, in regular intervals, until their length was equal to 30, as shown in Listing 1.1. The gestures were then converted into an array containing all the rotations in the x axis, followed by all the rotations in the y axis, followed by all the rotations in the z axis, as defined by Expression 1, where n represents the gesture's number of frames.

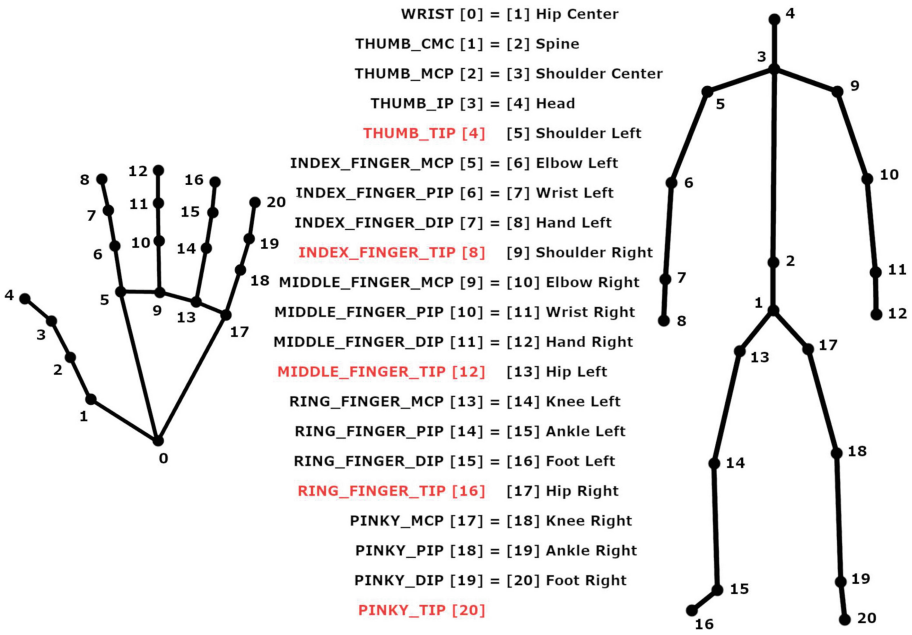

Fig. 3. MediaPipe's hand joints (left) mapped to ReBase's joints (right). The joints in red were ignored. (Color figure online)

```
def reduce_frames(gesture, n_frames):
    length = n_frames
    # While the gesture's length is larger than 30 frames
    while length > TARGET_LENGTH:
        count = 0
        # Remove frames in regular intervals
        step = math.ceil(length / TARGET_LENGTH)
        for i in range(0, length, step):
            gesture[i] = None
            count += 1
            length -= 1
            if length == TARGET_LENGTH:
                break
        gesture = [x for x in gesture if x != None]
    return gesture

```

Listing 1.1. Gesture length normalization

$$[x_1, x_2, ..., x_n, y_1, y_2, ..., y_n, z_1, z_2, ..., z_n] \tag{1}$$

$$NormalizeRotation(x) : \begin{cases} x - 360 & \text{if } x > 360 \\ 0 & \text{if } -0.1 < x < 0 \\ x & \text{otherwise} \end{cases} \quad (2)$$

3.2 Gesture Classification

Once the training dataset had been constructed, processed, and stored in ReBase, we used it to train SVMs to classify new gestures. The SVMs were constructed in Python, using the *scikit-learn* library. The Radial Basis Function (RBF) was the chosen kernel function, and all of SVMs' parameters were set to scikit-learn's default values. For this application, each of the 27 letters of the Libras' alphabet represent a class.

There are no official Python APIs for ReBase, but it is possible to retrieve the data from the ReBase REST Server via HTTP requests. Thus, we created a Python script to download the training dataset, process it, and feed it to the SVMs, using the *requests* library. After training, the SVMs were dumped into files using the *Joblib* library so that they could be used later in the C-Libras application.

4 The C-Libras Application

The C-Libras application was developed for PC and Android smartphones and allows users to have their gestures classified in real-time by the trained SVMs. C-Libras follows the client-server architecture, as illustrated by Fig. 4. The clients are responsible for capturing the users' gestures, processing them and sending them to the server, which submits them to the SVM for classification and sends back the results. Each client communicates with the server via a WebSocket.

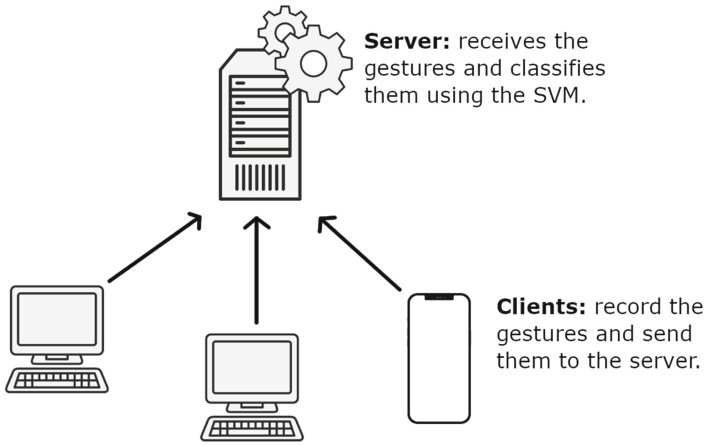

Fig. 4. C-Libras' client-server architecture.

The C-Libras Client. The C-Libras client was developed in Unity, using the *MediaPipe Unity Plugin* to process the gestures and extract the joints and the *WebSocketSharp* library to establish a WebSocket connection to the server.

Once the users start the application, they are presented with the C-Libras' main screen (Fig. 5), where they can perform the gestures to be classified. After each gesture is classified, the alphabet letter corresponding to the classification is displayed on the top section of the screen. The text display can be cleared by clicking the "Limpar" ("Clear") button. From this screen, the user can also access the camera configurations window (Fig. 6) by clicking the leftmost button. The configurations include selecting the video source from a list of cameras connected to the device, changing the camera's resolution, etc.

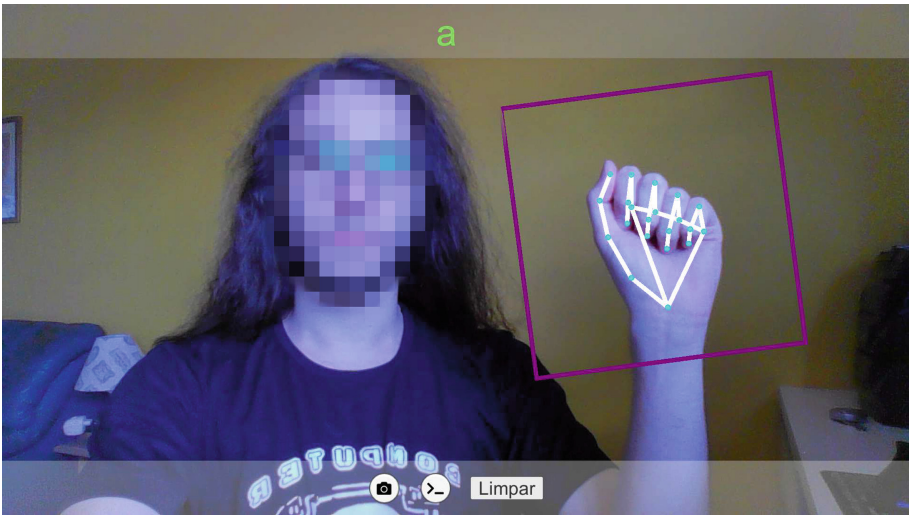

Fig. 5. C-Libras' main screen.

The client application operates in a simple workflow, described by Fig. 7. Once every frame, the application uses the MediaPipe Unity Plugin to process the latest frame from the live feed - i.e., the webcam - and extract the hands' joints. Then, it calculates the joints' rotations and stores them in an array, as defined by Expression 1, in order to translate the data to the same format used to train the SVMs. During this process, the rotations are also normalized as described by Eq. 2. After the frames' joints are extracted and converted, the resulting array is stored in a buffer, and the process is repeated for the next frame. Once the buffer is filled with 30 arrays - each representing a frame - the application sends them to the server via the WebSocket, clears the buffer, and immediately restarts to fill it while waiting for the classification. When the client receives a response from the server, the corresponding letter of the alphabet is displayed.

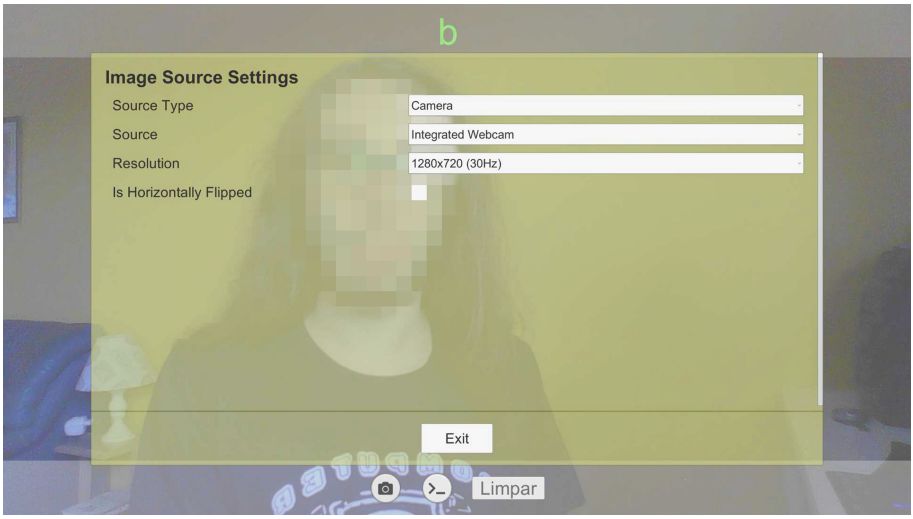

Fig. 6. C-Libras' camera configuration window.

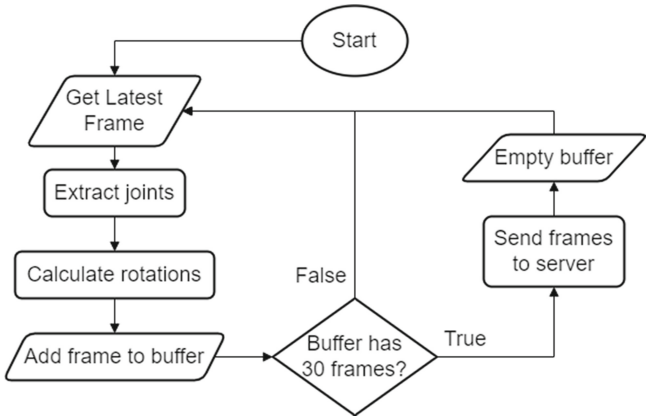

Fig. 7. C-Libras client workflow.

The C-Libras Server. The C-Libras server was developed in Python, using the *Django* web framework. Similar to the client, the server also has a simple workflow (Fig. 8). On startup, the server loads an SVM model from a file, using the Joblib library, and begins listening for connections. Once a WebSocket connection is established, the server waits for a message containing the 30 frames for classification. When the frames are received, the server only has to submit them to the SVM, since the client has already done all the pre-processing. Lastly, the results are sent back to the client via the WebSocket, and the server waits for the following message. A sequence diagram is presented in Fig. 9, exemplifying the interactions between the server and the client.

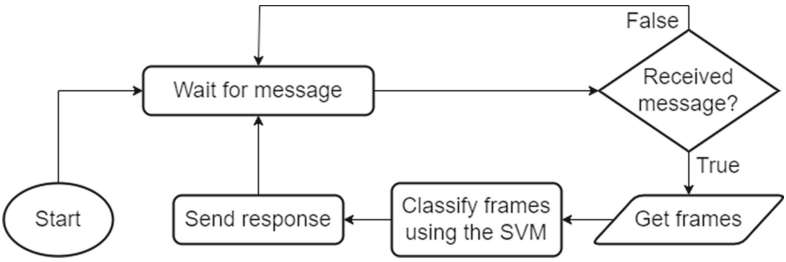

Fig. 8. C-Libras server workflow.

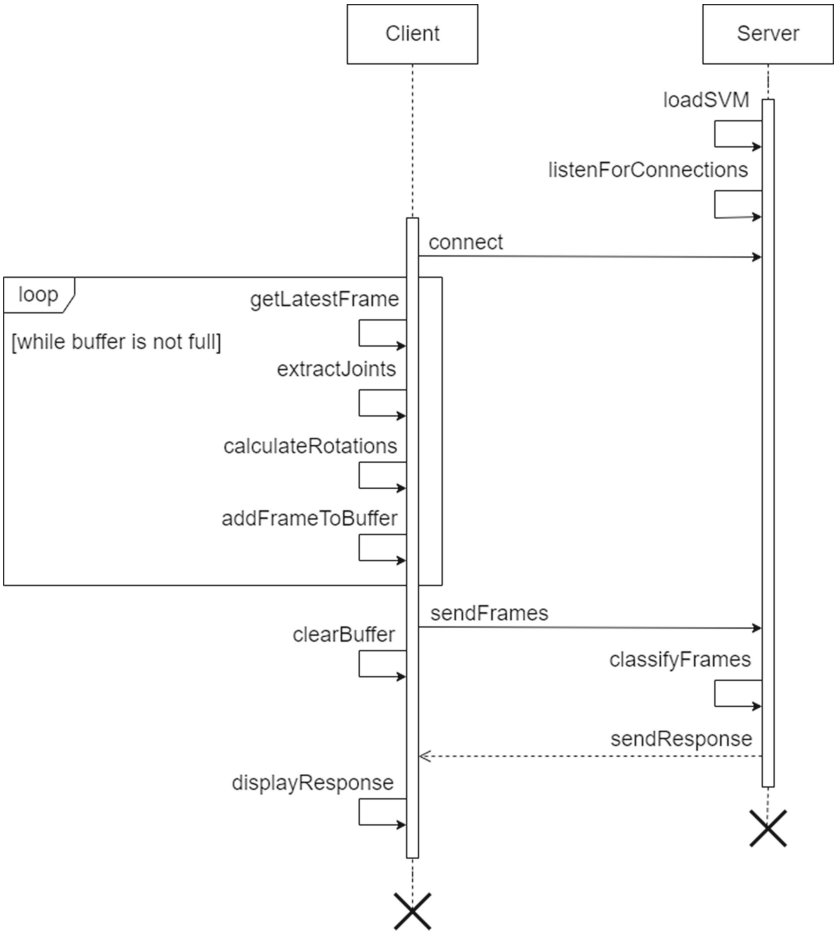

Fig. 9. Interaction between the C-Libras client and server.

5 Results

We performed three experiments to measure the accuracy of our application. In the first experiment, we trained multiple SVM models using seven examples per letter, varied the letters for each, and measured their accuracy. We chose to use 7 gestures for each letter for this example because this was the number of gestures available for the letters we had the least gestures.

In the second experiment, we trained multiple models using only the letters 'A', 'B', and 'C', varied the number of examples per letter, and measured their accuracy. We chose these letters for this experiment because they had the most examples available. The third experiment was the same as the second, but we used the letters 'A', 'E', and 'S'.

To measure the application's accuracy with a given SVM model, we calculated the accuracy for all letters and divided it by the number of letters, as represented by Eq. 3, where L is the set of letters recognized by the model. The accuracy of each letter was calculated by executing the gesture ten times and dividing by ten the number of times the SVM model classified it correctly, as represented by Eq. 4, where c is the number of correct classifications.

$$TotalAccuracy = \frac{\sum_{i \in L} Accuracy(i)}{|L|} \tag{3}$$

$$Accuracy(i) = \frac{c}{10} \tag{4}$$

When training an ML model, it is essential to use a large and diverse dataset. As this application is still in its initial stages, the training dataset is not as large as preferable, but already provides an excellent foundation to expand upon. In total, the training dataset contains 339 examples of gestures. There are 7 examples for each of the 27 letters of the Libras' alphabet, including the letter 'Ç' (c cedilla), out of which 3 were recorded by 3 different volunteers and 4 were taken from YouTube educational videos. There are also 50 additional examples for the letters 'A', 'B', and 'C', recorded by a member of the research team.

Table 1 presents the results for Experiment 1, ordered by the number of letters recognized by the model, which we will call the "size" of the model, and by accuracy. We can see that the SVM trained to recognize the full Libras' alphabet obtained an extremely low accuracy. There were too few examples for each gesture and too many classes, which resulted in an SVM model of poor quality. Usually, when working with ML classifications, especially when there are many classes, the number of examples for each class can easily reach the thousands.

We also noted that the size of the SVM affects the accuracy of the model, as evidenced by the overall tendency of smaller models to have a higher accuracy score - except for the model trained with the letters 'A', 'B', 'M', 'P', and 'W', which scored lower than the larger "A to F" model. Furthermore, the selection of the letters also influenced the accuracy score. That is because, with so few examples for each letter, there is a very high chance that the SVM will confuse

Table 1. Results for the first experiment, with varying letters and fixed number of gestures per letter.

Letters	Accuracy
Full alphabet	14.4%
A to M	30.8%
A to F	61.7%
A, B, M, P, W	42%
A, C, T, X	67.5%
A, C, N, T	95%

letters with similar signs, such as 'F' and 'T', 'A' and 'S', or even 'A' and 'X' (Figs. 10a, b and c). This concept is evidenced by the higher accuracy score of the last SVM compared to the penultimate one: both models had the same size, but the last one had a set of letters more distinct from each other.

Table 2 presents the results for Experiment 2, ordered by accuracy and by the number of examples for each letter. The total accuracy for the SVMs trained to recognize only the letters 'A', 'B', and 'C' is much higher than the accuracy of the more general models used in Experiment 1. That is because it is easier for the model to distinguish between fewer classes, as evidenced in Experiment 1. Moreover, the letters used in this experiment are distinct enough from each other (Fig. 10d) that it allowed the model to accurately distinguish between them with as few as seven examples per letter.

Table 2. Results for the second experiment, with fixed letters and varying number of gestures per letter.

Size	Acc. for 'A'	Acc. for 'B'	Acc. for 'C'	Total Acc.
1	100%	0%	100%	66.7%
5	70%	100%	90%	86.7%
7	100%	100%	100%	100%
10	100%	100%	100%	100%

The results of Experiment 3, presented in Table 3, establish the importance of the letters set. This experiment used an SVM of the same size as the one used in Experiment 2, but the letters used are much more similar to each other (Fig. 10e). As a result, the total accuracies of the models in Experiment 3 were much lower than those of the ones used in experiment 2, even though they had the same size. It is also important to note that, in both Experiment 2 and 3, while an increase in the model's size translated to an increase in total accuracy,



(a) ‘F’ and ‘T’.



(b) ‘A’ and ‘S’.



(c) ‘A’ and ‘X’.



(d) ‘A’, ‘B’ and ‘C’.



(e) ‘A’, ‘E’ and ‘S’.

Fig. 10. Comparisons of similar letters of the Libras’ alphabet.

Table 3. Results for the third experiment, with fixed letters and varying number of gestures per letter.

Size	Acc. for 'A'	Acc. for 'E'	Acc. for 'S'	Total Acc.
1	50%	0%	30%	26.7%
5	10%	20%	0%	40%
7	90%	100%	0%	63.3%

it did not translate directly to an increase in accuracy for all letters, because the size was not yet large enough to eliminate confusion between letters.

6 Conclusion

This work presents a mobile and desktop application, C-Libras, capable of capturing hand gestures and classifying them as letters of the Libras's alphabet. C-Libras follows a server-client architecture in which the client captures the gestures and sends them to the server, which classifies them by using an SVM. The training dataset was constructed by capturing the gestures with regular RGB cameras, processing them with Google's MediaPipe to extract the hands' joints, calculating their rotations, and storing them in the ReBase database.

We tested many SVMs with different sets of letters and different numbers of examples per letter. We concluded that, besides the number of examples per letter, the size of the letter set affects the model's accuracy. Our SVM trained with seven examples per letter to recognize the entire Libras' alphabet obtained an accuracy of 14.4%, while our models with fewer letters obtained much higher scores, such as the model for the letters 'A', 'B' and 'C', which obtained 100% accuracy with the same number of examples per letter. We also concluded that the letters included in the set affect accuracy as well: our model for the letters 'A', 'E' and 'S' obtained only 63.3% accuracy with the same number of letters and examples per letter as the 'A', 'B' and 'C' model.

In the future, we plan to expand our training dataset by recording more examples for each letter of the Libras' alphabet, executed by different voluntaries. We also plan to fine-tune our SVMs, experimenting with different kernel functions and different parameters. This way, we expect to be able to train an SVM to classify the entire Libras's alphabet accurately. Furthermore, to measure usability and reliability, we plan to test and validate our application in real scenarios with real users, including members of the deaf community. Finally, we plan on implementing a reverse approach, allowing the application to receive text input and display the corresponding alphabet signs.

References

1. Assaleh, K., Shanableh, T., Zourob, M.: Low complexity classification system for glove-based Arabic sign language recognition. In: Huang, T., Zeng, Z., Li, C., Leung, C.S. (eds.) ICONIP 2012. LNCS, vol. 7665, pp. 262–268. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34487-9_32
2. Barbosa, T., et al.: Rebase: data acquisition and management system for neuromotor rehabilitation supported by virtual and augmented reality. In: Anais do XXIII Simpósio de Realidade Virtual e Aumentada, pp. 181–185. SBC, Porto Alegre, RS, Brasil (2021). https://sol.sbc.org.br/index.php/svr/article/view/17535
3. Cheok, M.J., Omar, Z., Jaward, M.H.: A review of hand gesture and sign language recognition techniques. Int. J. Mach. Learn. Cybern. **10**(1), 131–153 (2017). https://doi.org/10.1007/s13042-017-0705-5
4. Chong, T.W., Lee, B.G.: American sign language recognition using leap motion controller with machine learning approach. Sensors **18**(10), 3554 (2018)
5. Cooper, H., Holt, B., Bowden, R.: Sign Language Recognition, pp. 539–562. Springer, London (2011). https://doi.org/10.1007/978-0-85729-997-0_27
6. El-Bendary, N., Zawbaa, H.M., Daoud, M.S., Hassanien, A.E., Nakamatsu, K.: ArSLAT: Arabic sign language alphabets translator. In: 2010 International Conference on Computer Information Systems and Industrial Management Applications (CISIM), pp. 590–595. IEEE (2010)
7. Elakkiya, R.: Machine learning based sign language recognition: a review and its research frontier. J. Ambient. Intell. Humaniz. Comput. **12**(7), 7205–7224 (2021)
8. de Geografia e Estatistica (IBGE), I.B.: PNS 2019: país tem 17,3 milhões de pessoas com algum tipo de deficiência. Agência de Notícias - IBGE. https://agenciadenoticias.ibge.gov.br/en/agencia-press-room/2185-news-agency/releases-en/31465-pns-2019-brazil-has-17-3-million-persons-with-some-type-of-disability. Accessed May 2022
9. Fitzpatrick, E.M., et al.: Sign language and spoken language for children with hearing loss: asystemantic review. Pediatrics **137**(1) (2016)
10. Google: MediaPipe. https://google.github.io/mediapipe/. Accessed Mar 2021
11. Mohandes, M., A-Buraiky, S., Halawani, T., Al-Baiyat, S.: Automation of the arabic sign language recognition. In: Proceedings. 2004 International Conference on Information and Communication Technologies: From Theory to Applications, pp. 479–480. IEEE (2004)
12. Roberts, M., et al.: Common pitfalls and recommendations for using machine learning to detect and prognosticate for Covid-19 using chest radiographs and CT scans. Nat. Mach. Intell. **3**(3), 199–217 (2021)
13. Rossol, N., Cheng, I., Basu, A.: A multisensor technique for gesture recognition through intelligent skeletal pose analysis. IEEE Trans. Hum.-Mach. Syst. **46**(3), 350–359 (2015)
14. Sadek, M.I., Mikhael, M.N., Mansour, H.A.: A new approach for designing a smart glove for Arabic sign language recognition system based on the statistical analysis of the sign language. In: 2017 34th National Radio Science Conference (NRSC), pp. 380–388. IEEE (2017)
15. Shukor, A.Z., et al.: A new data glove approach for Malaysian sign language detection. Procedia Comput. Sci. **76**, 60–67 (2015)
16. Ultraleap: Leap Motion Controller. https://www.ultraleap.com/product/leap-motion-controller/. Accessed Mar 2021
17. Unity: Unity - Game Engine. https://unity.com/pt. Accessed Mar 2021

18. Wang, R.Y., Popović, J.: Real-time hand-tracking with a color glove. ACM Trans. Graph. (TOG) **28**(3), 1–8 (2009)
19. Zhang, H., Wang, Y., Deng, C.: Application of gesture recognition based on simulated annealing BP neural network. In: Proceedings of 2011 International Conference on Electronic and Mechanical Engineering and Information Technology, vol. 1, pp. 178–181. IEEE (2011)
20. Zhang, X., Chen, X., Li, Y., Lantz, V., Wang, K., Yang, J.: A framework for hand gesture recognition based on accelerometer and EMG sensors. IEEE Trans. Syst. Man Cybern. Part A Syst. Hum. **41**(6), 1064–1076 (2011)