An Approach to Sign Language Translation using the Intel RealSense Camera

Jayan Mistry
Department of Computing and Technology
Nottingham Trent University
Clifton Lane, Nottingham NG11 8NS, United Kingdom
mistryjayan@gmail.com

Benjamin Inden
Department of Computing and Technology
Nottingham Trent University
Clifton Lane, Nottingham NG11 8NS, United Kingdom
benjamin.inden@ntu.ac.uk

Published in: Proceedings of the 10th Computer Science and Electronic Engineering Conference. This is the author-archieved version. The final version can be found at http://ieeexplore.ieee.org.

Abstract

An Intel RealSense camera is used for translating static manual American Sign Language gestures into text. The system uses palm orientation and finger joint data as inputs for either a support vector machine or a neural network whose architecture has been optimized by a genetic algorithm. A data set consisting of 100 samples of 26 gestures (the letters of the alphabet) is extracted from 10 participants. When comparing the different learners in combination with different standard preprocessing techniques, the highest accuracy of 95% is achieved by a support vector machine with a scaling method, as well as principal component analysis, used for preprocessing. The highest performing neural network system reaches 92.1% but produces predictions much faster. We also present a simple software solution that uses the trained classifiers to enable user-friendly sign language translation.

Keywords: Sign Language Translation, American Sign Language, Intel RealSense, Machine Learning

1 Introduction

Sign language consists of various gestures that are used as a method of communication. Recent estimates state there are over two hundred sign languages in use around the world today, the exact number is not known with any confidence as new sign languages rapidly emerge through creolization [1]. Recent statistics suggest that around 5% of the world's population is suffering from a type of hearing loss to some degree at a growing rate [2]. Due to Sign Language requiring no artificial hearing instrument and being efficient, a great majority of deaf individuals select sign language as their primary language. It is estimated there are around 70 million deaf people who use sign language as their first language although it is hard to measure the exact number of signers [3]. In addition to this, sign languages are also used by disabled individuals who cannot physically speak.

Although sign languages are widely used there is no common technical interface that can be relied upon to accurately translate sign language into written or spoken text. The task is challenging for various reasons, including context dependency of signs and the variability of signs on spatial and temporal scales. Here we focus on static signs, i.e., those whose meaning is determined by a single hand configuration as opposed to dynamic signs, whose meaning is determined by a pattern of change of hand configuration over time. We also focus on hand gestures, ignoring any cues given by body posture or facial expressions. In American sign language, all letters of the alphabet are of this type, therefore even a system with these constraints can provide a simple solution to spell any word. The recent proliferation of commercial off the shelf consumer devices for depth tracking like the Intel® RealSense™ camera makes this system an affordable option for sign language translation that can also easily extended to more complex gestures, as will be discussed later. Firstly, we review previous systems.

2 Previous Work

Here we review some of the significant prior work related to recognition of sign language gestures. More detailed reviews can be found in [4,5, 6].

One popular approach involves the use of gloves to aid sign language translation by embedding sensors within the fabric of the gloves. One of the first implementations was developed at the University of Illinois, Chicago in 1980 [7]. The gloves had a flexible tube with a light source at one end and a photocell at the other for each finger. The device was quite bulky and did not measure palm orientation.

A glove developed for Bell Labs in 1983 allowed for digital data entry using the 26 manual gestures of the American Sign Language [8]. The glove contained numerous touch, bend, and inertial sensors sewed into the fabric, orientation was tracked via the use of a camera.

The Cyberglove presented in 1990 [9] was a wireless data glove that accurately captured the movement of hand gestures and translated them into text. The system worked by monitoring a set of 22 thin foil strain gauges. This system was an improvement on previous attempts due to the use of strain gauge sensors allowing a greater accuracy, with resolutions within a single degree of flexion. Since then, many variations of sign language translation gloves have been developed based on these ideas, including recent work [10], where the gloves also contain an accelerometer and pressure sensor to distinguish between various signs, and have been constructed with materials costing less than £90 without requiring complex construction methods.

What these systems have in common is that they put the burden on the sign language speakers by requiring them to wear these special gloves. Systems that rely on 2D or 3D computer vision to recognise gestures are much more user friendly. An early approach involved the use of 3 cameras to record gestures of a subject and interpret them in terms of object manipulation tasks [11]. Another paper [12] studied the classification of single-view sign language motion images. A surmise of this paper is that sign language is made of a time sequence of units called cheremes, which are “described by hand shape, movement, and location of the hand, which can be said to express the 3-D features of the sign language”. A dictionary was created containing the details of the 3D features of gestures, which were automatically converted to 2D image features. Another paper classified 31 distinct ASL gestures using a 2D camera video stream [13], but the limitations of 2D images became manifest as difficulties in adapting to different speakers. In contrast, a recent approach that made use of the Microsoft Kinect 3D depth sensing camera achieved an average recognition rate of

92.4% over 55 static and dynamic gestures [14]. The background, lighting conditions, clothing, and skin colour have little impact on the accuracy of the results of this project due to only employing the use of the Kinect's depth sensor. However, as stated in the paper, issues were encountered during testing when users wore bracelets, as the system could not accurately identify hands and hence recognise gestures. Another recent paper used the depth sensing capabilities of the Intel RealSense camera to extract the geometric shape of the hand including the hand skeleton and joints [15]. The method was evaluated on a set of gestures commonly used in human-computer interaction (like grabbing, tapping, or swiping) containing 14 different gestures obtained from 20 participants and achieved an accuracy of 83%. The paper evaluated the efficacy of a depth-based approach against a skeleton-based approach and found the skeleton based approach to be superior in terms of gesture recognition accuracy.

Regarding the recognition of dynamic gestures, an early approach used three sets of features: stop positions; the shape of the trajectory; and the shape of the hand at the stop position [13]. Another method detailed in [14] used a variant of Dynamic Time Warping to achieve gesture recognition. This method finds similarities in temporal sequences despite speed differences, and was used to recognise dynamic gestures with accuracies of up to 90% on a number of related tasks [14, 16, 17].

Another challenge for recognition of sign language gestures is coarticulation, where a hand shape in one sign anticipates the shape or location of the following sign. Previously, Conditional Random Fields have been trained to classify signs in the presence of coarticulation [18]. These models are often used for classification of elements in a sequence as they take the temporal context into account.

3 Methods

Our interest in this paper focuses on using commercial off the shelf 3D vision systems and the following questions:

- Since Python and scikit-learn have become very popular in the machine learning community in recent years, how can one build a sign language gesture recognition system using these tools that has a high accuracy on static gestures (with a view to extending the system to dynamic gestures later)?
- How can the training and its product be embedded into a user friendly application software?
- Where classification mistakes are made, how can we better understand and fix them?

As part of this investigation, we present a new data set that has been collected to address these questions. This contains gestures representing 26 letters from American Sign Language, which is a very widely used sign language around the world.

3.1 System Architecture

The implementation of this project consists of 3 applications which carry out 3 main functions allowing the translation of sign language into text (as shown in Fig. 1):

- The Sign Language Interface for Data Extraction (SLIDE) application extracts hand data from the hands and formats the data appropriately; the data is stored in a format that can be easily used by the other application.
- The Sign Language Interface for Training Application (SLIT) application uses the hand data extracted from the SLIDE application and trains a machine learning model on the hand data.
- The Sign Language Interface for Data Extracting Testing (SLIDET) application extracts hand data from hands displayed to the camera. The application then runs the machine learning model and pipes the hand data to the model to predict the gesture being shown to the camera.

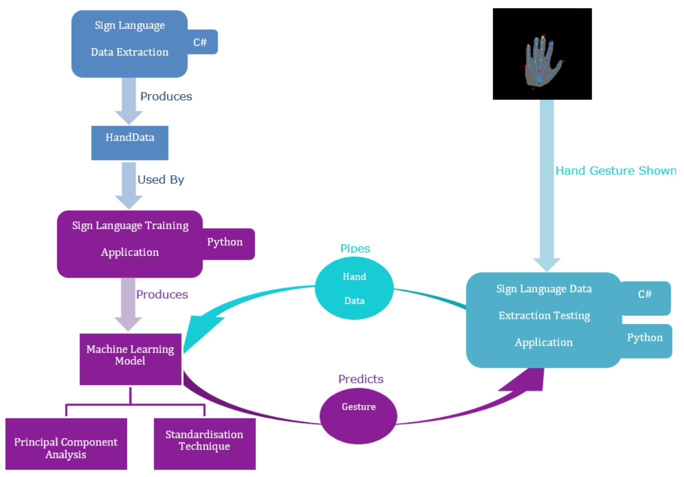

Sign Language Data Extraction
C#
Produces
HandData
Used By
Sign Language Training Application
Python
Produces
Machine Learning Model
Principal Component Analysis
Standardisation Technique
Hand Gesture Shown
Pipes
Hand Data
Sign Language Data Extraction Testing Application
C#
Python
Predicts
Gesture

Fig. 1. How the applications interact

The user interface of the SLIDET application has been designed with Jacob Nielsen's ten heuristics for the evaluation of user interfaces in mind [19] (Fig. 2). For example, visibility of System Status is ensured by constant feedback of the hand position and skeleton being tracked on the hand. There is also clear text displayed telling the user the application is ready to receive a sign. Every time a letter is added to the text box, the letter is spoken. A flashing cursor is shown in the text box that shows where the letter signed is going to be added. User control and freedom have been provided by making it possible to use the application without keyboard or mouse (or glove). The users can move their hand to the left or right border which allows for them to add a space in the text box. The users can also move their hand down to delete the last letter from the text box in case they make a mistake. Once the application has started the user can create sentences and have them spoken out loud. After the application has started, a visual message appears notifying the user to wave to start.

3.2 ASL Hand Gestures Data Set

A number of data sets were reviewed for this project (e.g., [20, 21]), but were found to be not perfectly suitable for this project as they were either 2D datasets, heavily pixelated or contained gestures that are not specific to ASL. Additionally, the sample size of these data sets was typically not large and those datasets that did have a large sample size per gesture were restricted to a small number of gestures.

Our new data set (donated to the UCI Machine Learning Repository, https://archive.ics.uci.edu/ml) contains 100 signs each from 10 signers for each of the 26 letters, so 26,000 instances from 26 classes overall. For each sign, it contains 90 features:

- A rotation quaternion for each of 20 joints on the hand (the RealSense camera provides 22, but data for the wrist and palm joints was removed as it was constant 0)
- The degree of flexion of each finger
- The degree of openness of the hand
- A palm orientation quaternion

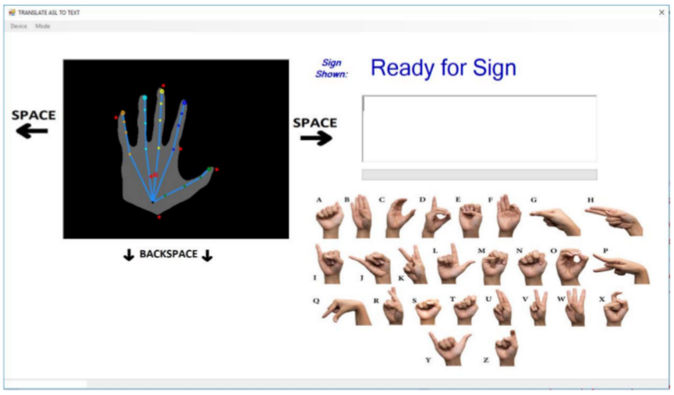


Fig. 2. The user interface of the SLIDET application

This data was gathered using an Intel RealSense F200 camera and the RealSense API. According to Intel, to produce a depth frame, the RealSense's infrared projector illuminates the scene with coded infrared vertical bar patterns that are predefined and increase in spatial frequency. The patterns created interact with the scene and the reflections are captured by the Infrared camera. The captured pixel values are then processed by the imaging ASIC to generate a depth frame. Further depth streams that are captured create a video stream that is transmitted to the client system [22].

The signs were recorded from users sitting in front of the camera, the distance between hands and camera was approximately 20 - 25 cm, while the angle was approximately 90°. The data extracted by the camera is invariant under changes of distance and relative orientation of the hand within reasonable boundaries as specified by the manufacturer as long as there is not too much occlusion. Of course, differences in lighting and shadows are also not a problem for this type of camera.

75% of the instances were used as a training set in all experiments, while the other 25% were used as a test set. Some experiments also use validation sets.

3.3 Pre-processing

A range of pre-processing techniques available in the scikit-learn library [23] were applied:

A *StandardScaler* rescales each feature independently such that they have a mean of zero and a standard deviation of one. Like for the other pre-processing methods, this is based on the training set only but will be applied to the test set as well. A *MinMaxScaler* transforms features by scaling each feature to a given range, here between zero and one. A *MaxAbsScaler* divides each feature by its maximum absolute value. This method thus scales each feature such that the maximal absolute value of each feature in the training set will be 1.0. It does not centre/shift the data, so no sparsity is destroyed. A *RobustScaler* sets the median to zero and scales the data according to the interquartile range, it is therefore more robust to outliers than a StandardScaler.

A further preprocessing technique is *Normalization*. Unlike the other techniques, this does not treat features individually, but scales individual samples to have unit norm. It is normally used when the similarity between samples needs to be calculated. Also, *Principal Component Analysis (PCA)* is a well known technique for dimensionality reduction.

3.4 Machine Learning Models

The performance of a support vector machine (SVM) and a multilayer perceptron (MLP) are compared. Both are implemented in scikit-learn [23]; the MLP uses the ReLU transfer function and the Adam solver [24] as training algorithm by default. Scikit-learn's default parameters for Adam are used (batch size 200, 200 iterations, learning rate 0.001, numerical stability parameter 10^{-8}).

A genetic algorithm [25] was created to find the optimum number of neurons and layers for the multilayer perceptron. A population of 500 random tuples with a minimum of 1 and a maximum of 5 random integer values was generated. Each integer value denoted the number of neurons within a layer (between 1 and 40). Evaluation of the individuals was by training the MLP and measuring its fitness on a separate evaluation set. The top 20% MLP as well as a random number of weaker MLPs were selected as parents. One point crossover was then performed as well as mutation (setting the number of neurons in a single layer, with a maximum of 100) with a probability of 20%. The genetic algorithm was run for 200 generations, convergence was reached during that time. An MLP with three hidden layers and layer sizes (21, 21, 25) emerged as champion and was used for all further experiments.

4 Results and Discussion

The highest performance reached is 95.0%, this is achieved with an SVM and the MaxAbsScaler (Table 1). The MLP comes close with 92.1%. All scaling techniques increase the performance of both MLP and SVM except the RobustScaler, which leads to a drastic performance decrease. Using PCA also increases the performance although only slightly.

Two further sets of experiments (Table 2, 3) confirm that using the Adam solver and the ReLU is indeed the best choice among several standard alternatives.

The confusion matrix for the trained SVM without pre-processing (Fig. 3) shows that the classification mistakes are not evenly distributed, but occur between specific pairs of signs. (A similar picture emerges when looking at confusion matrices for configurations with pre-processing.) Table 4 shows the most common confusions both for SVM and MLP. It can be seen that some confused signs are very similar overall (like R/U), some differ mostly in the configuration of one particular joint (like O/N), while some seem to differ by large but similar changes in several joints (like M/C). As exemplified by a plot of the first two principal components of the S and T signs (Fig. 4), some signs are indeed hard to classify based on the provided data alone.

Table I. Performance of different learners with pre-processing techniques

Pre-Processing Technique	**Accuracy of the Multilayer Perceptron**		**Accuracy of Support Vector Machine**	
	Without PCA	***With PCA***	***Without PCA***	***With PCA***
No pre-processing	78.5%	81.5%	85.7%	86.1%
StandardScaler	82.7%	83.4%	87.9%	89.6%
MinMaxScaler	82.4%	84.8%	87.4%	88.2%
MaxAbsScaler	91.5%	92.1%	92.6%	95.0%
RobustScaler	35.2%	45.0%	37.0%	47.0%
Normalisation	79.6%	82.1%	88.3%	87.4%

Table II. Performance of MLP with different training techniques

Solver	**Multi Layer Perceptron Accuracy**
lbfgs	60.2%
sgd	65%
Adam	78.5%

Table III. Performance of MLP with different transfer functions

Activation Function	**Accuracy of Multilayer Perceptron**
identity	71.7%
logistic	55.3%
tanh	60.4%
ReLU	78.5%

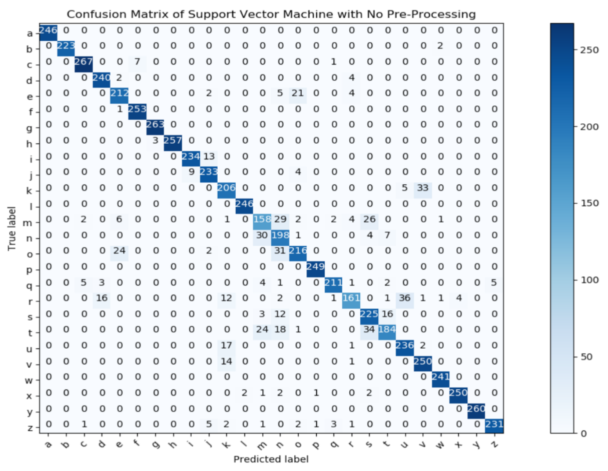


Fig. 3. Confusion Matrix of SVM without Pre-Processing

On a system with a 2.5 GHz Intel Core i7-6500U processor, 16 GB RAM, and Windows 10, the training process took 9.6 s for the MLP, and 7.8 s for the SVM. The trained MLP took between 166 µs and 401 µs (mean 215µs over 1000 signs) to output a prediction, while the trained SVM took between 1.45 ms and 2.21 ms (mean 1.64 ms). A test of the GUI showed that there were 1.05 s on avarage between the completion of a sign and the output of the corresponding letter on screen and as sound. This time could be decreased if the audio output was removed, allowing for a fluent interaction with skilled signers as during the processing time the GUI is already receptive for the next gesture.

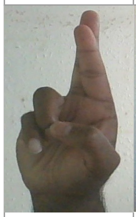
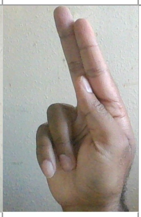
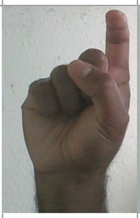
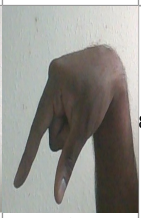
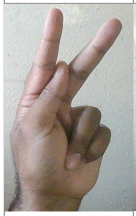
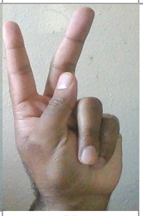
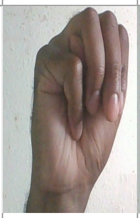
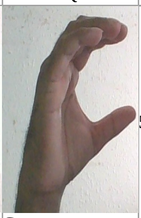
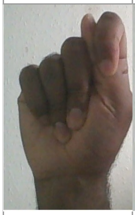
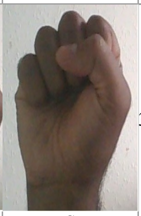
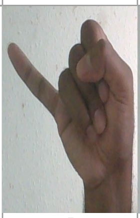
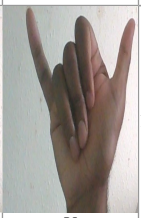
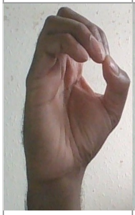
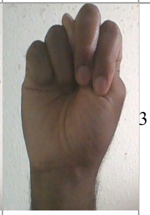
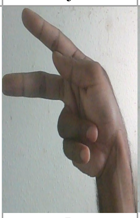
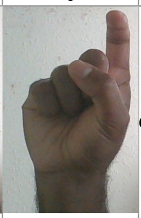
5 Conclusions

We have shown that classification of American Sign Language sign with the Intel RealSense camera is feasible with high accuracy and speed. While we have used static gestures only, previous work by others [14] shows that it is possible to also recognise dynamic gestures in principle. We found that a support vector machine used together with pre-processing by a MaxAbsScaler and PCA yielded the best results although some other combinations of methods were close. In particular, a multilayer perceptron with the same kind of preprocessing is almost as accurate but predictions are much faster. The mutual similarity of skeleton data for some signs makes higher accuracies hard to achieve but we believe that an improvement in accuracy could be achieved if further features were added based on the optical 2D image of the sign.

The recognition of non-manual signs could be implemented by tracking the full skeleton of the user making the gestures. Although the Intel RealSense camera does not support skeleton tracking, this could be implemented with another depth sensing camera that supports skeletal tracking such as the Microsoft Kinect [26].

All machine learning components were realised using the widely used scikit-learn library. A user-friendly application was build that uses the trained model to allow spelling of words and sentences using American Sign Language, and could be scaled for the inclusion of further signs.

Table IV. Most commonly confused signs

Support Vector Machine			**Multi-Layer Perceptron**		
Actual Sign	**Predicted Sign**	**Instan-ces**	**Actual Sign**	**Predicted Sign**	**Instan-ces**
 R	 U	36	 Z	 Q	89
 K	 V	33	 M	 C	58
 T	 S	34	 J	 Y	105
 O	 N	31	 P	 Z	62

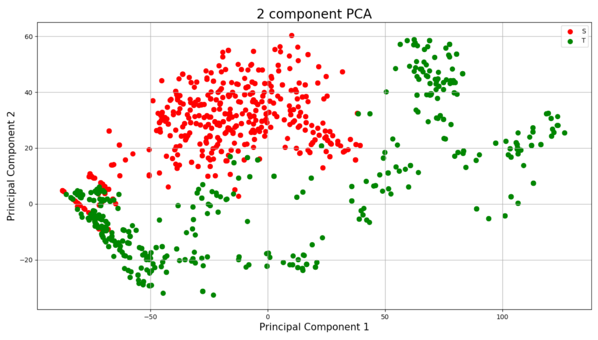


Fig. 4. A plot of the S versus T sign principal components

REFERENCES

[1] S. Chalk. Do we know how many sign languages there are in the world? - Clarion UK. [online] *Clarion UK*. Available at: http://www.clarion-uk.com/know-many-sign-languages-world/ [Accessed 20 Oct. 2017].

[2] N. Oishi and J. Schacht. Emerging treatments for noise-induced hearing loss. *Expert opinion on emerging drugs*, 16(2):235–245, 2011.

[3] Anon.. Sign Language. *World Federation of the Deaf*. [online] Available at: https://wfdeaf.org/human-rights/crpd/sign-language/ [Accessed 25 Feb. 2018].

[4] H. Cheng, L. Yang, and Z. Liu. Survey on 3D hand gesture recognition. *IEEE Transactions on Circuits and Systems for Video Technology*, 26(9), pp.1659-1673, 2016.

[5] S. Mitra and T. Acharya, "Gesture recognition: A survey," *IEEE Transactions on Systems, Man, and Cybernetics C*, 37:3, pp. 311–324, 2007.

[6] J. Suarez and R. R. Murphy, "Hand gesture recognition with depth images: A review," in *Proceedings of the IEEE International Symposium on Robot Human Interaction and Communication*, Paris, France, Sep. 2012, pp. 411–417.

[7] Erard, M. (2018). Why Sign-Language Gloves Don't Help Deaf People. [online] *The Atlantic*. Available at: https://www.theatlantic.com/technology/archive/2017/11/why-sign-language-gloves-dont-help-deaf-people/545441/ [Accessed 2 Mar. 2018].

[8] T.A. DeFanti and D.J. Sandin. "Final Report to the National Endowment of the Arts." US NEA R60- 34-163, University of Illinois at Chicago, Chicago , Ill., 1977.

[9] J. Kramer and L. Liefer. "The Talking Glove: An Expressive and Receptive "Verbal" Communication Aid for the Deaf. Deaf-Blind, and Nonvocal." Technical Report, Stanford University. Department of Electrical Engineering. Stanford, Calif. 1989.

[10] Anon. Low-Cost Smart Glove Translates American Sign Language Alphabet and Controls Virtual Objects. 2017-07-12. *Targeted News Service*. [Accessed 02 Feb. 2018].

[11] A.G. Hauptmann, P. McAvinney, and S.R. Shepard. Gesture analysis for graphic manipulation. Technical Report CMU-CS-88-198, Carnegie Mellon University, 1988.

[12] S. Tamura and S. Kawasaki. Recognition of sign language motion images. *Pattern Recognition*, 21(4), pp.343-353, 1988.

[13] C. Charayaphan and A.E. Marble. Image processing system for interpreting motion in American Sign Language. *Journal of Biomedical Engineering*, 14(5), pp.419-425, 1992.

[14] G. Plouffe and A.M. Cretu. Static and dynamic hand gesture recognition in depth data using dynamic time warping. *IEEE transactions on Instrumentation and Measurement*, 65(2), pp.305-316, 2016.

[15] Q. De Smedt, H. Wannous, and J.P. Vandeborre. Skeleton-based dynamic hand gesture recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1206-1214, 2016.

[16] P. Chanda, S. Auephanwiriyakul and N. Theera-Umpon. Thai sign language translation system using upright speed-up robust feature and dynamic time warping. In *2012 IEEE International Conference on Computer Science and Automation Engineering (CSAE)*, pp. 70-74, 2012.

[17] W. Ahmed, K. Chanda and S. Mitra. Vision based Hand Gesture Recognition using Dynamic Time Warping for Indian Sign Language,. In *2016 International Conference on Information Science (ICIS)*, pp. 120 -125, 2016.

[18] R. Yang and S. Sarkar. Detecting coarticulation in sign language using conditional random fields. In *18th International Conference on Pattern Recognition*, 2:108-112, IEEE 2006.

[19] J. Nielsen and R. Molich. Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems,* pp. 249-256, ACM 1990.

[20] Facundoq.github.io. Sign Language Recognition Datasets –. [online] Available at: http://facundoq.github.io/unlp/sign_language_datasets/index.html [Accessed 17 Apr. 2018].

[21] Kaggle.com. Sign Language MNIST | Kaggle. [online] Available at: https://www.kaggle.com/datamunge/sign-language-mnist [Accessed 17 Apr. 2018].

[22] Anon. Intel® RealSense™Camera SR300 Product Data Sheet. https://software.intel.com/sites/default/files/managed/0c/ec/ realsense-sr300-product-datasheet-rev-1-0.pdf [Accessed 21 June 2018].

[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, and J. Vanderplas. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825-2830, 2011.

[24] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization, In *3rd International Conference for Learning Representations*, San Diego, 2015.

[25] A.E. Eiben and J.E. Smith. *Introduction to evolutionary computing* (Vol. 53). Springer-Verlag 2015.

[26] Anon. (n.d.). Kinect Sensor. [online] Available at: https://msdn.microsoft.com/en-gb/library/hh438998.aspx [Accessed 30 Mar. 2018].