Lennart Alexander Ørnberg
Sign Language Recognition using Leap Motion
B.Sc. Computer Science
18th of March 2016

Declaration

"I certify that the material contained in this dissertation is my own work and does not contain unreferenced or unacknowledged material. I also warrant that the above statement applies to the implementation of the project and all associated documentation. Regarding the electronically submitted version of this submitted work, I consent to this being stored electronically and copied for assessment purposes, including the School's use of plagiarism detection systems in order to check the integrity of assessed work.
I agree to my dissertation being placed in the public domain, with my name explicitly included as the author of the work."

Date:
18th of March 2016

Signed:

Working Documents:
http://www.lancaster.ac.uk/ug/ornberg/

Abstract

Most deaf children are born into hearing families and the majority do not begin their Sign Language learning before age 4-6 in school. As children learning Sign Language from infancy show higher language proficiency than those who do not, it underlines the importance of hearing parents learning how to sign. Despite the increased commercial and academic interest in gesture recognition over the last two decades, there are currently no widespread system to aid Sign Language learners. This report proposes a gesture recognition system for both fixed and gesture based signs, using the Leap Motion sensor, to complement students attending tutored Sign Language classes. Additionally, to support local variations and nuances in signs, the system offers tutors the ability to build unique datasets. The implementation issues and requirements are addressed and its supporting machine learning classifier algorithm is explained in detail. Additionally the importance and implementation of rejection of erroneous gestures for pedagogic purposes, is also outlined. A quantitative study conducted showed the system obtaining an on average lower recognition accuracy, compared to earlier systems only concerned with recognition of fixed signs.

Contents

1 Introduction

There has been increasing commercial and academic interests in gesture recognition in recent years. As technology has become more ubiquitous, the demand for innovative ways of human-computer interaction arise to complement the standard keyboard and mouse, as well as newer form of interaction such as touch screens. This paves the way for solutions that can be used to enable and advance Sign Language speaker's way to control and communicate with digital systems.

Sign Language is the most prominent way of communication between the deaf and hearing impaired and over 125 000 people in the UK have British Sign Language (BSL) as their first or preferred language (AOHL, 2015). Numbers show that most deaf individuals are born into families with hearing parents meaning the number of potential learners are higher (GRI, 2008). For most hearing impaired individuals, acquiring Sign Language occurs in school at age 4-6, with a minority learning the language from infancy from native speakers, either parents or peers (Newport, 1988). Just as with spoken languages, immersion and exposure from an early age results in a noticeable difference in proficiency for Sign Language users as well (Johnson, 1989; Newport, 1988). This gap could be closed by facilitating the means for hearing parents of deaf children to acquire Sign Language communication competence before the child becomes able to learn.

There are numerous of online resources, books and video courses for learning BSL, with the preferred and most efficient option being tutoring one-to-one as instant feedback can be provided. In spite of the fact the recent technological advancement in gesture recognition, there are currently no widespread learning system validating or providing feedback for learners of Sign Languages, that could enhance the at-home learning experience while attending tutored BSL courses.

The purpose of this project is to build and evaluate a Sign Language Interpreter (SLI) using the Leap Motion sensor. This will allow Sign Language learners a platform with instant feedback on signs that are performed, subsequently improving the learning process as SLI's ability to recognize signs will be based on the quality of the input. Just as with spoken languages, there is a wide range of Sign Languages as well, such as Irish, French and American Sign Language (ASL). Additionally, there are multitude of local accents and variation of each of these Sign Languages. Because of this, SLI will be supported by the means for groups of tutors to build and distribute custom datasets without relying on third parties, allowing sign recognition of geographical variations by local sign language students.

This report constitutes of 6 sections. The following section outlines the Leap Motion hardware, its limitations and related works on the subject of Sign Language Interpretation. The third section provides a brief overlook for the system design and the simplistic user interfaces of the accompanying applications. The fourth section discusses issues with two earlier efforts to implement an SLI, explores the final implementation and its components and functionality in detail. In the fifth section, SLIs performance in a quantitative study is in focus and adjustments that were made accordingly based on the study's results are also covered. Lastly, a conclusion is presented in the final section.

2 Background

With a rising interest in dynamic gesture recognition as an alternative to traditional human-computer interaction tools, dynamic hand-shape recognition has been lagging behind (Wachs, 2011) as it requires greater precision. Computer Sign Language recognition used to require ample computing power, as processing input streams of video were one of the common means to provide support for gesture recognition. A number of previous efforts to create sign recognition systems were based on the use of web cameras such as demonstrated by Chen et al (2003). While requiring computing power to process high resolution images, images of less quality were often prone to noise and external light sources. For this reason one of the main challenges on the subject has been hardware. As alternatives to the traditional video cameras, other products have been suggested such as the Cyberglove (Wang et al, 2005), a glove which tracks the users finger positions in 3 dimensional space. While offering more precise tracking, a wearable hardware lacks usability and in the case of Cyberglove, it is not aimed at private use.

Moving away from wearable technology, requirements of background color or other constraints for efficient gesture tracking, a new set of hardware have appeared in recent years. Mass produced products such as Kinect and Leap Motion has been on the market for the last 5 and 3 years respectively, and have made gesture recognition technology public domain. These products are reasonably priced while still able to offer high precision tracking enables developers to produce systems for a range of different applications that can be available to most people.

2.1 The Leap Motion Controller

While Microsoft's Kinect provides in-depth map of its cameras views, and subsequently offering body movement tracking, the more compact Leap Motion focuses exclusively on hand movement in a smaller space. The most vital components of Sign languages are made up of hand shapes, hand motions and positions, which together constitutes a word, and are performed in a limited spaces in front of the speaker. This makes Leap Motion the better choice of the two when disregarding body position and motions, in addition to being more affordable. The Leap sensor also has an advantage that the hand color does not have to be considered.

The core hardware component of the SLI uses this controller, an 8 by 3 centimeter sized IR sensor, composed of two infrared cameras and three IR LEDs that runs on solely the connected USB power. It has a recorded sub-millimeter accuracy down to 0.2 mm for static positions, outclassing the Kinects reported 1.5 cm (Weichert, 2013).

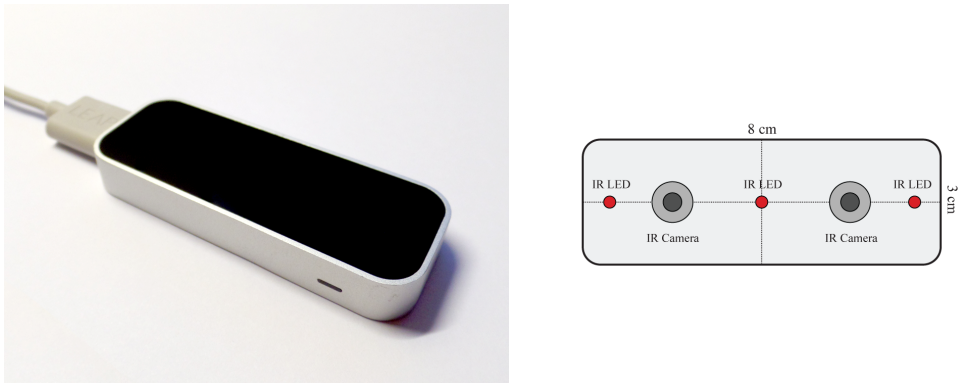


Figure 2.1: The Leap Motion sensor and its schematics

The sensor uses the two IR cameras in a traditional stereo vision manner, triangulating the hands in view and mapping the different hand features points in Cartesian space, with the analysis and feature extraction of the sensor images is hidden from developers. By using the provided API, developers can access the frame per frame based skeleton models, where each model is composed of the directional vectors of the corresponding hand and arm bones in the sensor space.

2.2 Related works

After the Leap Motion sensor was introduced in 2013, there has been published a handful academic works focusing on possible the applications of the sensor and its accuracy. Systems to aid stroke rehabilitation (Khademi et al, 2014), drawing tablet alternatives (Sutton, 2013) and in-air hand written signs recognition (Xu et al, 2014) are a few of the utilizations proposed. A few papers also evaluates the use of Leap Motion as a means for Sign language recognition specifically. Since the Leap Motion API handles the image analysis behind the scenes, it alleviates developers of the cumbersome task of extracting features from raw input data. This allows researchers to focus more on the feature selection, rather than extraction, and the accuracy result of applying machine learning for classification of different signs.

Marin et al (2014) investigates the performance of the Leap Sensor by training a SVM classifier to recognize 10 different static signs with the extensive amount of 1400 samples and was able to achieve an average accuracy of 80%. The paper also focuses on the challenges of which features to include for accurate classification of gesture inputs, and arguing for normalization to accommodate a robust system for users with different hand sizes.

Chuan et al (2014) suggests a similar interpretation system proposed in this report, to teach parents of deaf children ASL, but without the ability of a flexible dataset nor the ability to recognize gestures. The study compares the accuracy of using k-Nearest Neighbor and Support Vector Machine classifiers to exclusively recognize non-moving hand shapes. While a relatively high accuracy of close to 85% was achieved, the authors recognize the sensors hardware limitation. During multiple instances the skeleton model failed to mimic the hand in the sensors view, causing any subsequent classification likely to be mistaken. This highlights one of the reoccurring themes in other papers as well. While there are several hurdles to overcome in developing gesture recognition systems, most of the inaccuracy is attributed to misclassified data produced by the Leap API.

One assessment of using the Leap Motion for interpreting Australian Sign language (Potter et al, 2013) deemed the sensor as promising towards future sign recognition applications, but at current state too inaccurate, for example for hand shapes perpendicular to the sensor. While most of the signs take place in the Leap sensors view, there are also a collection of signs taking place around the speakers facial area. Signs requiring face or body contact proved to be inaccurate. Additionally, facial expressions help display complex nuances and support the meanings of hand shapes in a sentence. These features are quite clearly out of the scope of detections possible by a system only using the Leap sensor.

While these are the current limitations, there are an ample amount of signs that could be taught and recognized that operate within the dimension of the sensors view. Further development such systems would be beneficial as newer builds and software updates for Leap Motion improve its accuracy, with possible stand-alone or supporting alternatives of hardware able to complement. Mohandes et al (2015) investigates one possible sidestepping of this problem by introducing a joint system combining to Leap Motion sensors perpendicular to each other. The overall accuracy proved to increase by up to 8.5% with a total of 98%. However, requiring two sensors would essentially double the price, and is avoided for the sake of this project. While the dataset in study had a high number of samples per class, they were all performed and recorded by one person. A comparable gain in accuracy instead of the addition of another Leap Motion unit could potentially be achieved by increasing the disparity of the dataset, as seen later in this report.

The common concern for [12], [13] and [14] is the Leap sensor's ability to recognize a finite set of fixed, or non-moving hand signs of different languages. To be able to increase the potential of using Leap sensors as a learning tool , the ability to not only recognize static signs but gestures needs to be included. Therefore this project will propose a system to recognize both fixed and moving signs, with the mentioned additional ability for users to create unique dataset of signs to increase the lexical recognition properties SLI.

2.3 Evaluation criteria

The evaluation of the SLI system will be based on whether it can accurately distinguish a set of fixed and gesture based signs. The accuracy results of **80%** by Marin et al (2014) will be used as the benchmark comparison score. It cannot however be based independently of the quality of the dataset as it is the only means for a system using machine learning algorithms to achieve any accuracy. This means that while a matching accuracy score might occur, the accuracy will vary with the quality of the dataset. However, an accuracy similar to earlier works is expected, regardless of the additional set of gesture based sign classifications that needs to be made, if the dataset contains an apt amount of heterogenous samples.

3 Design

As pointed out previously, earlier efforts to test Leap Motion involved providing a constructed dataset, as they only aim to recognize a finite set of different hand signs. Given SLIs intention to provide the means for tutors to create their own local and course specific datasets the system was split into two major components; a *Builder* and an *Interpreter*.

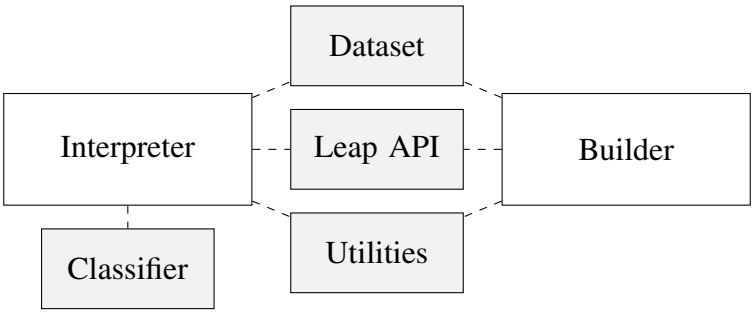


Figure 3.1: SLI file structure

The *Interpreter* is the simple application to be used by the Sign Language learner. It connects to the running Leap Motion service, through its *API* and interprets gestures made in the sensor view. Upon launch the application trains the *Classifier* using the provided *Dataset*. When streams of input is received from the *Leap API*, the sequence of data is extracted for features and compressed by *Utilities* (see Feature Extraction on page 14) before classified by *Classifier*. If the input is recognized as a sign, its word value is printed in the application window.

To enable Sign Language tutors to build and distribute set of signs on their own, without having programming experience nor being reliant on a third party, a *Builder* is provided. This is the instrument for dataset creation. The tutor can input any word as a value for a sign, and record corresponding samples using the *Leap API* and subsequently store them in the *Dataset*. The *Builder* also makes use of *Utilities* to extract features to create vectors used by the *Classifier* later.

3.1 User Interface

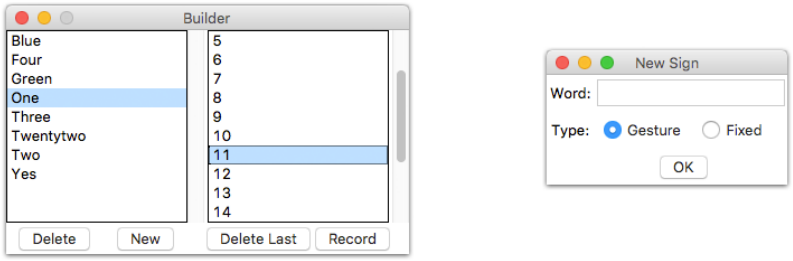


Figure 3.2: *Builder* application windows

Given the limited set of tasks, the visual elements of *Builder* are few. A tutor can select a Sign by its value, delete or create a new one. The list of samples recorded for the corresponding selected Sign are displayed in the list to the right with the options to delete or record a new Sign. Upon creation of a new Sign the user is prompted with the dialog to the right in Figure 3.2, and have to specify whether it is motion based or not, determining if single or sequences of frames are to be stored in the *Dataset* for the corresponding Sign.

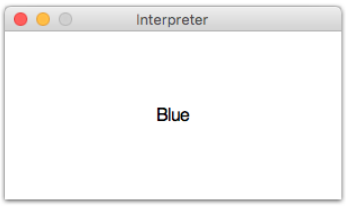


Figure 3.3: *Interpreter* application window

To simplify both applications to accommodate use by non-technical individuals, the amount of interface elements are kept to a bare minimum. The *Interpreter* user interface is therefore even more minimalistic. In a plug-and-play fashion a simple blank screen is shown on startup, that after SLI recognizes input as a sign will print out the word value for the corresponding sign, or a rejection message if the input is deemed too unfamiliar to signs in the dataset.

4 Implementation

This section is concerned with the implementation of SLI in detail. Initially, the two preliminary approaches are outlined to highlight the need for a machine learning based solution. Thereafter, the implementation requirements and the solutions to meet these requirements are presented.

4.1 Preliminary implementations

During development of SLI, two early implementations were discarded in favor of a full machine learning approach. While the final implementation is a technical descendant of the second prior approach, the first approach is not. It bear characteristics from classical machine vision approaches (Davies, 2005) and was ultimately deemed impractical for the use in SLI. However, despite rejecting the implementation on the whole, its shortcomings outlined some of the challenges needed to be solved by the final implementation.

4.1.1 Template matching

The first implementation of SLI took a naive approach to sign recognition, ignoring error tolerance and resulted in inconvenient requirements for dataset creation. Each hand in every frame would have its discrete features extracted, based on fixed thresholds. If a feature, say a finger, had an angle that proved to be amidst a set range, the feature would be given the corresponding discrete value.

Finger	*Pitch*	Up, Down
	Flexure	Extended, Bent

Thumb	*Angle*	Out, Side, Under
	Flexure	Extended, Bent

Hand	*Roll*	Up, Side, Down
	Pitch	Up, Down, Flat
	Direction	Left, Ahead, Right

Table 4.1: Discrete features

The discrete values possible for a feature to inhibit is shown in Table 4.1. The collection of these features for all fingers and the hand were treated as a *hand state*. Every *hand state* were compared against each predefined state in the database, in turn performing a one-to-one feature comparison. If all features matched and the sign belonged to the category of type *fixed* SLI would output the corresponding word. If it was of the type *gesture*, the results would be temporarily stored as signs of type *gestures*, were composed of a sequence of states. If a hand state of this type matched, the system would actively look for the next hand state in the sequence, for a fixed set of time, and if not found would revert back to its original state. For both single and sequence based signs, the match algorithm would return negative for a single feature mismatch.

There are two major limitations regarding this approach. Firstly, this implementation does not have any means of tolerance. While the Leap Motion software provides a millimeter precise skeleton model, errors and noise does occur. If a single feature, such as a finger's flexure does not fall within the fixed thresholds, no match would be produced, despite potentially having an obvious candidate. This becomes increasingly difficult when recognizing a sequence of correct hand states, as one divergent value in one state would abort the whole recognition. With virtually no leeway, for the SLI to function well, one would have to assume a noise-less stream of data input and an error-free gesture *performance* of the user. This would ultimately render the learning experience obnoxious.

Secondly, the precision and capabilities are limited to the number of discrete features that the implementation support. The aforementioned features extracted might not suffice to capture the variations in some of the more similar hand signs yet to be implemented. Complex and specific nuances, of say, rotation in yaw of the palm might prove to be essential to identify specific gestures. Hence, the number of possible features would ultimately govern the usability and flexibility of the system.

4.1.2 A partial Machine Learning approach

In endeavoring to avoid these issues a second system structure was considered, which in essence was a hybrid between the first and the final approach. The main concept of having gestures represented as a sequence of hand states was kept, but the matching technique would now be supported by a machine learning algorithm. Applying a mathematical approach to classification, enables one to keep the numerical data from the Leap skeleton model. This subsequently rids the need of value discretization and its potential problems, in addition to providing a statistical approach to setting a level of error tolerance.

While this remedies the issues of *fixed* signs recognition, *gestures* still remains a complication. The sequence has to be manually constructed, which in itself not only a time consuming process but a challenge, as the best keyframes to represent the gesture can be hard to choose. While this is cumbersome, but feasible for a handful of gestures, the

approach does not take relative positions into account. In many cases, it is the change in position or rotation that defines many gesture based signs.

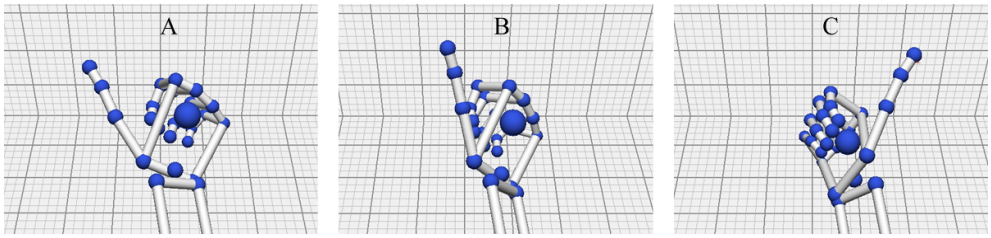


Figure 4.1: Possible rotational start and end positions

A scenario in Figure 4.1 where the classifier is simply trained to recognize state A, and actively tries to classify the following state B, would have to be trained to consider both A and B as possible start positions to be able to consider both A→B and B→C as valid gestures. Consequently, this results in having the classifier train and look for an extra subset of signs for every similar exception. The immediate workaround in this case is to omit some features, such as rotation, in the classifier training data, and store start positions separately. Upon positively classifying the same state A, following a short time interval, one could validate that rotation has occurred and positively output that a specific gesture has been matched.

Adopting the hand state principle from the template matching approach works for fixed signs and gestures composed of a sequence of clearly different hand states. Whenever gestures with the same hand states are to be implemented, as the one described above, special cases has to be implemented as well. The number of exceptions would grow as the number of implemented signs increases, in effect requiring cumbersome manual work for the Sign Language tutors contributing to the database. Additionally, it would challenge the user implementing the signs to make a decision on ambiguous challenges such as “how many keyframes is needed to represent a circle motion?”.

The difficulty of manually crafting representations of hand signs relates to the discretization problem of the template matching approach. Discrete features would need to be implemented as new signs were to be added, and it would be highly impractical to

add exceptions and special cases for every new gesture based hand sign added to the database.
When interpreting motions as fixed snapshots the data on what goes on between these snapshot, is lost. A somewhat shortsighted solution is to deal it them in a manual fashion, as described above, while being highly impractical. The circumvention of this impracticality was what ultimately led to the final approach outlined throughout the remainder of this section.

4.2 Requirements

To alleviate the issues brought to light by the two preliminary approaches, a full-fledged machine implementation was developed. Gestures would be recorded as a continuous stream of inputs, and use a machine learning algorithm to classify the input accordingly, rather than a sequence of separated hand states as initially considered. This would verify, to a certain degree, that the gesture based sign as a whole is performed correctly, since inputs between the snapshots of hand states will not be completely ignored. However, this implementation approach does come with a few requirements, *position relativity*, *duration indifference* and *novelty detection*.

To rid the need of multiple samples to cover possible start and end positions, the hand coordinates can not be relative to the position of the Leap sensor. This requires SLI to matched both fixed and gesture signs regardless of where in the sensors field of view they are observed.

The second requirement is duration indifference. To avoid interpretation cutting off halfway through a motion or including information after the gesture is finished, classifying a set of frames from a fixed time interval is not preferable. Speed and variation of the gesture might vary from the samples in the database, so to achieve higher accuracy SLI is required to be independent of fixed time intervals.

The third requirement is novelty detection. Machine Learning algorithms finds the best fit of an input to a set of classes, regardless of how well the fit actual is. Novelty detection is the ability to recognize inputs that differ from earlier observed cases in great detail, and subsequently reject them. In practical terms, this means that SLI is required to discard clearly erroneous inputs, rather than presenting the best fit as the result.

4.3 Interpreter algorithm

The *Interpreter* application imports and trains a classifier with a provided dataset, recognizes inputs using said classifier and outputs the corresponding class name, being the word value of the sign recognized. As one of the stated requirements to be indifferent to the duration of a gesture performance, SLI should actively detect the beginning and end of a motion, and subsequently being able to differentiate between fixed and gesture based signs.

Hence, as fixed and gesture based signs are distinguished before applied to a machine learning algorithm, the choice was made to use two separate classifiers, one for each said type of sign. While solving the issue of comparing fixed frames and sequences of frames to each other, this also increases the chances of higher classification accuracy, as the number of classes per classifier is reduced.

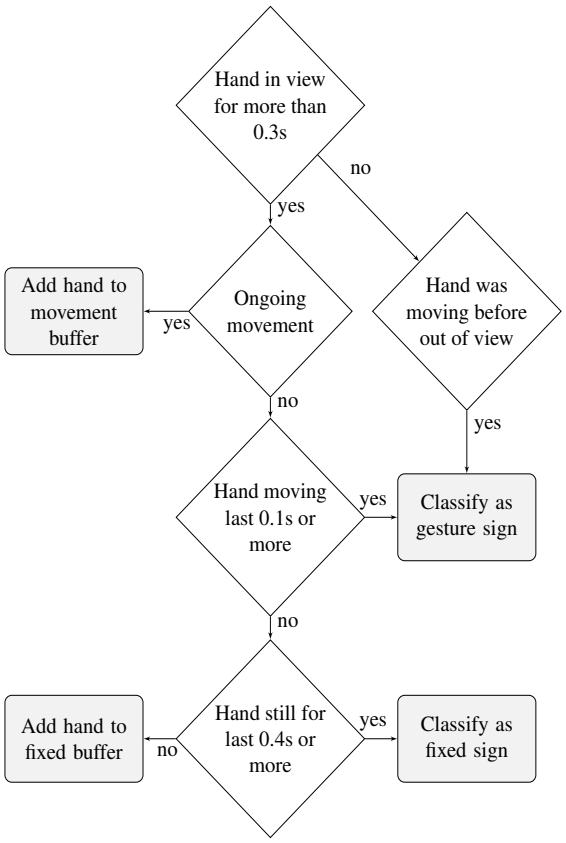


Figure 4.2: Interpreter algorithm flowtable

The Leap API waits 0.3 seconds after a hand has appeared in the view, to make allowance for the hand to enter the view properly before recording movement or static positions. A movement *buffer* ensures some tolerance for muscle tremor or other involuntary movements when fixed signs are performed, and some tolerance of movement discontinuity of gesture based signs. This is done by analyzing the movement between each frame in last 0.3 seconds. The rate of change between these frames has to be above a hardcoded threshold to be classified as a movement, and if the majority of frames in the said interval are determined as such, SLI establishes that a movement have ensued. If so, a second buffer is filled with frames to come until a movement stop is detected, or the hand is moved outside of the sensor view. When no movement is detected, but a hand is present, a third buffer collects all frames were no movement is occurring for a fixed time interval of 0.4 seconds.

When either a movement is finished, or a hand has been still for the last 0.4 seconds, the corresponding buffer of frames are extracted for features and compressed to a fixed length vector. Finally, the vector is fed as input to the corresponding classifier, and the buffers are emptied awaiting the next fixed or gesture based sign.

After the input has been classified as the class it is most likely to belong to, SLI makes the decision to print the class value in the Interpreter window if deemed similar enough (see Novelty Detection, section 4.10 on page 22).

4.4 Feature Extraction

Extracting relevant data is to paramount to solve machine learning problems. Collect an insufficient amount of information and classes might be rendered indistinguishable. Doing the opposite and a learning algorithm may distinguish inputs by entirely different features (Dreyfus, 1992), causing erroneous results.

In earlier efforts to create gesture recognition software, using video analysis, a major challenge was not necessarily which features to extract, but the extraction itself (Chen, 2003). In the case of Leap Motion however, the API provides framework with a skeleton model for each hand in view, and relieves individual developers of the heavy task of analyzing the IR images for relevant information.

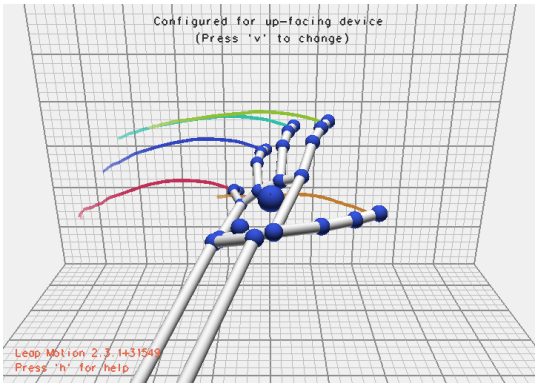


Figure 4.3: Leap Motion skeleton model

Each hand object is constructed of fingers and palm objects, and each finger is constructed by a set of bones. Each object has its corresponding 3 dimensional vector, referencing its direction and position in euclidian space in the sensors view.

The first features to be extracted were the 3D coordinates of the center position of each finger's *distal phalange*, the utmost bone, to represent the tip of each finger. As described in the earlier approaches of SLI, only finger flexure and pitch were initially collected, rendering the system indifferent to all signs with varying interstitial space, which is circumvented by collecting all three coordinates per finger. However, since all coordinates are relative the sensor, its positions are dependent on where in the sensor view the hand is placed. As the first implementation requirement stated, SLI needs to be indifferent to sensor relative position. Therefore, to train a classifier to recognize signs independent of its position in the sensor space, each finger coordinate were normalized by subtracting the coordinates of the palms center as origo.

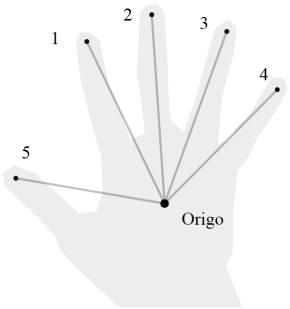


Figure 4.4: Normalization

The same principle was applied to the palms center coordinates as well, but rather than comparing it to another point, it was normalized relative to itself over time. With the first position being origo, each palm center in the subsequent frame would be relative to the center in the previous frame. This allows a gesture to have any start position and produce relating values to the the same gesture with a different start position elsewhere in the sensor view.

Palm pitch and roll were the only features dependent on their position relative to the sensor. Given that SLI for now only assumes single handed signs and since the palm cannot rotate independently of the arm, the only other directional vector to compare it to is the *XZ-plane* of the sensor view. This means that the start angle of any sample has an absolute value. Regarding pitch, it too was left dependent on the sensors *XZ-plane*. It was assumed that the angle between the palm and said plane was a more versatile factor than the angle between the palm and the arm. This is illustrated in Figure 4.5, where both movements A and B have variation in the angle between palm and the *XZ-plane*, while the angle between the palm and arm bone is left unaltered for movement B.

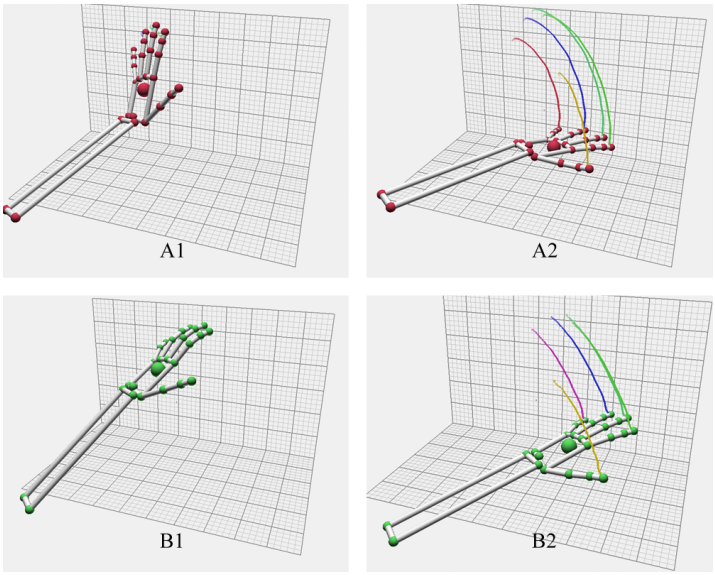


Figure 4.5: Pitch between palm/XZ-plane and palm/arm

As stated earlier, one of the requirements for SLI were to be independent of time. Not requiring signs to be performed within a fixed time interval increases the accuracy of the classifier as the capturing of a motion will not be at risk of stopping prematurely. Accordingly, any recorded sequence of frames might vary in length, depending on the

speed of the performed gesture. Therefore, before being passed to a classifier, the frame sequence of length *n* were compressed to *m* number of keyframes. All features in the frames from 0 to *n/m* was averaged to a single frame, and repeatedly done so for frames between *n/m* and *(n/m)*2* and so on. To find an optimal *m*, that is how many key frames SLI should consider, the implemented classifier described later in this report were used for testing. The average accuracy of a 1000 tests for each keyframe length of 5 to 20 with a randomized dataset of 4 classes produced the following results:

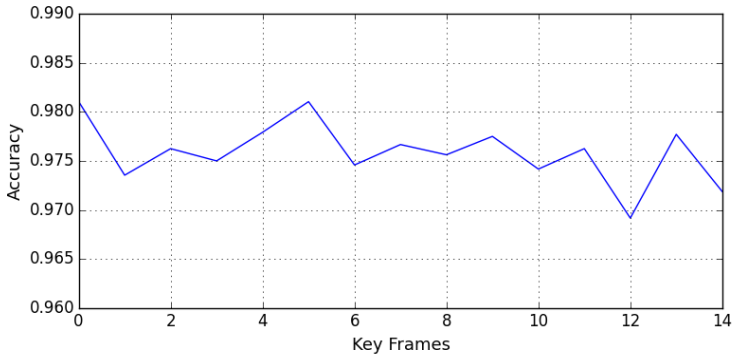


Figure 4.6: Accuracy for *n* keyframes

A cursory look at the accuracies show that for all test cases they were reasonably high. However, quite interestingly there were a slight decrease as the number of keyframes incremented. A mere speculation on why this might be, is that potential jitter in the movement and erroneous data were smoothened out upon averaging for a lower number of keyframes, having less of an effect on the classifiers decision. Regardless, the final keyframe value was set as 10, having the same accuracy as for a keyframe length of 5 but reasoned to better at representing longer, more complex gesture signs that was not part of the dataset used for testing.

The final feature vector is a 200 long array of floating point numbers (20 features by 10 frames) representing any motion over an arbitrary length of time.

4.5 Choosing a classifier

While many machine learning algorithms can be applied to solve a given problem with high accuracy, some algorithms naturally tend to yield better results depending on the attributes of the data.

The role of the SLI is to recognize inputs and classify them given a set of defined signs, whether that being British Sign Language or American Sign Language. The output made by the SLI should be the word value, meaning that all classes in the dataset have to be labelled; hence this is a supervised learning problem. There are a range of different classifier approaches within this category, each with it its own set of characteristics, such as k-Nearest Neighbor, Naive Bayes, Artificial Neural Networks and Decision Trees.

Upon choosing a classifier, there were several requirements to consider. One being that the SLIs inputs would be ad hoc, requiring a quick response from the classifier. Lazy learners, such as the k-Nearest Neighbor algorithm, would not be ideal, with a run time complexity of $O(n)$ (John, 1995), where n is the total number of data samples in the dataset, regardless of class. As classification accuracy would increase with the number of samples, so would response time.

Another requirement were novelty detection. The performed sign, given as inputs from the Leap Motion sensor are, not given to be correct even for human standards, it might not exist in the dataset or even be an official sign. For teaching use this would be particularly confusing as a classifier lacking novelty detection would pick the best option available, regardless of how well it actually matches the input.

Already before testing, the Naive Bayes classifier were a promising option as it inhibits features for both of the aforementioned requirements;

- Several components, such as prior probability can be precomputed, resulting in a computational complexity of $O(c)$ where c is the number of classes.
- As the probability for each class is the final product of a classification operation, novelty detection could be achieved by use of a probability based hardcoded threshold.

The classifier does however (naively) assumes that each feature is conditionally independent. This does not hold true for the features that can be extracted from the Leap Motion skeleton model. The x, y and z values of each finger are not independent from each other as say, the positions of two fingers predetermines possible positions for a third. Despite this assumption, the classifier tends to do well in areas where features are not conditionally independent, such as text classification and speech recognition.

4.6 Classifier Evaluation

In a preliminary dataset, 7 different fixed and gesture signs, were tested against 6 different classifiers from *Scikit-learn* for Python to determine their accuracy performance. The dataset was randomized, and split a training set and a testings set, with the former and latter respectively iteratively increasing and decreasing in size for each round. To ensure that a particular randomization did not result in particularly good or bad training set, every round with a n sample size were repeated 1000 times.

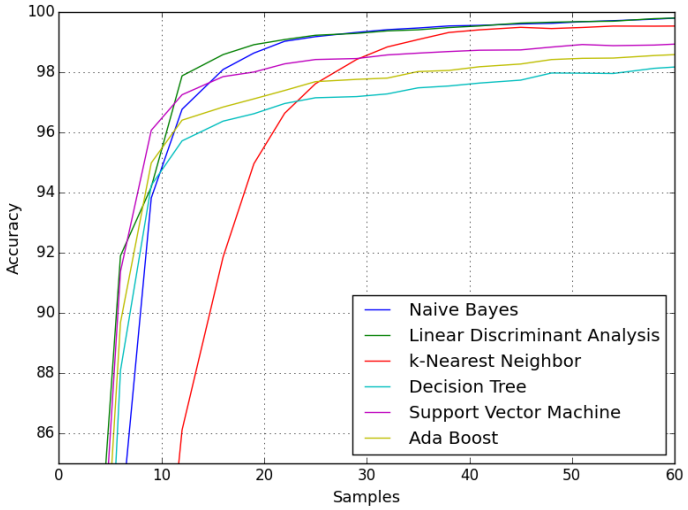


Figure 4.7: Classifier accuracies

As presumed, Naive Bayes proved to do well, along side Linear Discriminant Analysis (LDA). LDA demonstrated a marginally less score in accuracy but with a less marginally shorter execution time upon further comparison.

	Accuracy	*Time*
Naive Bayes	99.888%	193.6ms
LDA	99.883%	8.7ms

Table 4.2: NB and LDA test results

Note that, live test cases would differ, and as the quality of the samples may also vary, it would render the results different. With two almost equally well performing classifiers, and the same or similar training data, there would presumably not be any practical difference of between a SLI using LDA or Naive Bayes.

Due to these trifle difference in results, other factors were considered. Given the the straightforward mathematics and what was deemed to be simpler to actualize, the Naive Bayes classifier were chosen for implementation.

4.7 Inherent classifier rationale

There are several well functioning Machine Learning frameworks for most programming languages, with Python being a popular choice for scientific computing. The mentioned *Scikit-learn* is the most predominant one, with the highest amount of contributors and a wide range of functionalities. However, there were a couple arguments to implement an inherent classifier. Firstly, it would offer a greater understanding during development of SLI, to abate further understanding of SLI performance and evaluation. Secondly, the final code and system as a whole would have no dependencies other than the required Leap Motion libraries. Thirdly, novelty detection is the third stated requirement and non-existing feature in both the LDA and Naive Bayes classifiers in *Scikit-learn*. In order to reject erroneous input, this functionality needed to be implemented, and ultimately rendered a inherent classifier implementation the preferable solution.

4.8 Classifier implementation

The Naive Bayes classifier makes use of Bayes theorem:

$$P(c|x) = \frac{P(c)P(x|c)}{P(x)} \tag{1}$$

The combined probability of a feature vector belonging to a class is made up of the prior probability *P(c)*, and the product of the probability that a feature vector belongs to a class. *P(x)*, the denominator in the original formula, is the probability of the feature vector alone, which is the same regardless of the class to have its probability calculated for. Given that *P(x)* is a constant, it can be ignored, since it is not going to affect any class probabilities relative to each other.

The original Bayes Theorem with works discrete values, and it is challenging to determine how many values for each different features to create. As discussed earlier, the issue of discretization of features was one of the arguments to implement a machine learning solution. To make the Naive Bayes classifier work with discrete values, one can assume a Gaussian distribution for each feature over all samples in a feature vector (Mitchell, 1997). The probability of feature *x* belonging to class is therefore:

$$P(x|c) = \frac{1}{\sigma_c\sqrt{2\pi}} e^{-\frac{(x-\mu_c)^2}{2\sigma_c^2}} \tag{2}$$

Calculating the probability often produces small numbers, and multiplying them together by the lengths of 200 puts the classifier in risk of losing precision due to arithmetic underflow. To sidestep this problem, the logarithm of all three composing parts of the equation were taken, resulting in

$$\boxed{\log(P(c)) + \log(\frac{1}{\sigma\sqrt{2\pi}})} - \frac{(x-\mu)^2}{2\sigma^2} \tag{3}$$

The boxed section of equation 3, represents the precomputed value, which is calculated and stored upon training the classifier to avoid unnecessary computation when classification occurs. The latter part is what needs to be calculated for every feature x of the input vector. Ultimately, some addition, subtraction and multiplication is what is needed to estimate the probability of a vector belonging to a class, resulting in a relatively incomplex and resource cheap operation.

$$\max_i(P(c_i)\prod_j^k P(x_j|c_i)) \tag{4}$$

Finally, the classification of a vector simply made based on the highest probability of all class probabilities.

4.9 Quality of classifier implementation

To test the quality of the implemented classifier it was compared to the aforementioned Gaussian Naive Bayes classifier from *Sci-kit* for Python. Using the same testing code as earlier, but maintaining a 80/20 training/test ratio was kept throughout a 1000 iterations.

	Accuracy	*Time*
Sci-Kit	98.07%	0.03ms
Implemented	98.07%	0.82ms

Table 4.3: Sci-Kit and Implemented classifier test results

The identical accuracy scores indicates that the implemented version performs just as well as Sci-kit's Naive Bayes. Interestingly, it is a whole 27 times slower, presumably due to its lack of use of external array and data handling libraries, such as Numpy. Despite the order of magnitude in computational time, the higher average execution for the implemented version should be insignificant in the sense of user experience, given that it is reasonably fast and still independent of the number of samples per class.

4.10 Novelty detection

Machine learning algorithms try to find the best class fit for a given set of conditions. While the template matching approach discussed earlier were too strict, using a plain machine learning algorithm will attribute a feature vector to any class, regardless of how well it fits, as long as it is the best of the ones available. Any input is always bound to have a higher chance of belonging to one class than the other, so in order to reject error cases novelty detection has to be implemented.

In general novelty detection for most machine learning problems is hard. As mentioned earlier, for teaching purposes a too lenient classifier is not desirable as some level of correctness is required. On the flip side, too strict of a classifier would be aggravating for both teaching and other types use as the Leap Motion output is never guaranteed to be error free and would ultimately decrease the accuracy by ignoring recognition as a result of being too rigid.

The main issue of implementing novelty detection for SLI, was to find or generate data that could be used for testing. The threshold for what makes constitutes a valid gesture might vary from sign to sign, or even from person to person. There are a variety of situations that can arise where signs dubious to the system is rejected, but considered valid by human standards, or vice versa. Hence, what constitutes an outlier in terms of the classes in the dataset is difficult to determine. For this reason two types of errors were assumed; First being legitimate attempts at signs that might exists in a dataset. To test this condition, one class was left out of the training data, and exclusively used for testing, hence every results would by human standards be considered wrong.

The second type of errors assumed was purposely disparate ones. A separate set of inputs were produced, consisting of hand signs with static finger positions and movement in the opposite directions compared to the signs in the dataset to maximize difference.

The two types of error cases above were tested, alongside a normal dataset, where no classes were omitted and the dataset was randomized and split to a training and testing set before measuring the results.

One straightforward approach to handle novelty detection is by a fixed probability threshold. With a given dataset, the classifier could set a minimum value for a probability, based on prior testing with said dataset and subsequently reject all other cases falling below the threshold.

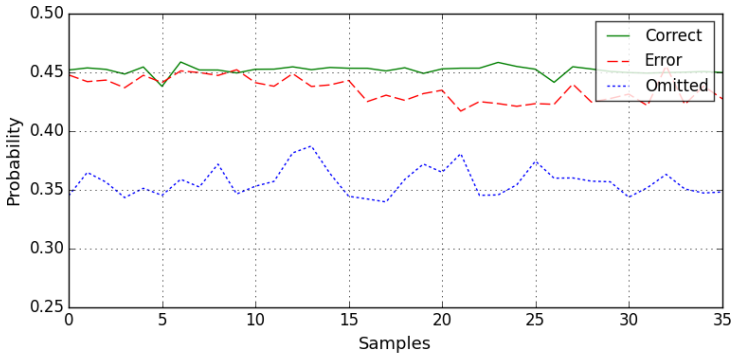


Figure 4.8: Probabilities

The results indicate a significant difference between *Correct* and *Omitted*, while showing only a slight difference between *Correct* and *Error*. Setting a fixed threshold around 0.40, would in this case most likely do well separating the former and the latter category, however SLI would still accept purposely erroneous inputs as valid signs. Another note to make is that this type of novelty detection would greatly depend on the signs (classes) in the dataset. When an increasing number of signs with similar features are added to the training data, the likelihood of each is bound to decrease. If there are several similar classes to choose from and since the sum of all probabilities is always one, the distribution of probabilities will decline, subsequently decreasing the margin for novelty detection.

As an example, one could consider a SLI trained for ASL numbers from *20*, *21* and *22*, where the first two are similar in finger movement, with the latter differing in both finger movement, palm position and palm movement.

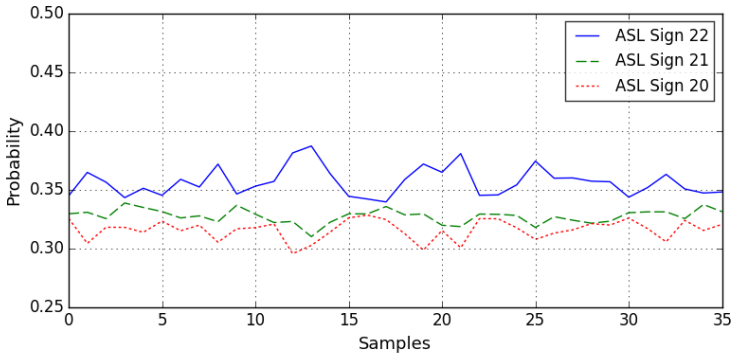


Figure 4.9: Probabilities for *omitted* test

The number of standard deviations from the mean (z-score) for *22* is likely to be high, as it bears little resemblance to the other two. However, the z-score for *21* is inclined to be less, given that *20* will be a significantly stronger candidate than *22* as outlined in the above figure. This would cause the rigorousness of th novelty detection to vary greatly beteween different signs.

As the two outlined novelty detection approaches is governed by the dataset of signs, a third option avoids the influence of other existing classes all together. Upon training the classifier, one can calculate the furthest euclidean distance in *nth* dimensional space from a sample to the average of the class. The distance from an input feature vector to this average center, is a means of comparison without being affected by the properties of other classes.

As seen in the above figure, for most test cases, the two categories *Error* and *Omitted* are closer to each other than they are to *Correct*, indicating that the Euclidean distance is a good measure for novelty detection. The average for each category are shown in Table 4.4.

	Correct	*Errors*	*Omitted*
Distance	0.655	1.082	0.986

Table 4.4: Euclidean distances for test sets

The mean value of the two error categories and *Correct* resulted in 0.85 which was finally set as the the fixed threshold for maximum distance. While data points between 0.85 and 1 are rejected and technically closer to the class average than the worst sample in the dataset, the ability for SLI to reject unknown signs was valued to be a more important feature. Additionally, it should be noted that while this threshold is independent from characteristics of other classes in a dataset, the samples of the best matching class governs the result. A class with tightly clustered samples and little variation will have a shorter maximum Euclidean distance, and effectively a small margin for what is tolerated as a valid sample of said class. This means that it is up to the creator of the database to ensure that some variance in the samples for each class is included.

5 Quantitative study

In order to evaluate SLI overall accuracy performance, a quantitative study was conducted. The aim of the study was to measure the systems difference in performance for different users, and type of errors that could occur, with potential improvements that could be done to rid some of these errors.

5.1 Study Conduct

Prior to executing the experiment, the participants were allowed to get familiar with the Leap Motion sensor, with the API skeleton model displayed on screen. Then the 8 participants were asked first asked to build a dataset of a set of 6 signs, with the exact number of 20 samples, with random variation at their own desire after being informed of the benefits of a dataset with sample variation. Half of the specified signs were of a fixed type; ASL for *One*, *Two* and *Three* and the rest of gesture type; ASL for *Green*, *Blue* and *Yes* (see Appendix A on page 37). These signs were chosen as they are single handed signs, and did not include overlapping of fingers, given Leap Motions mentioned limitations.

Upon completion of creating the dataset, the participants were asked to perform the signs they had recently recorded. The created dataset was imported to the Interpreter application, and a randomized list of 60 instructions was generated, consisting of 10 duplicates of each of the 6 class names. The word value of sign asked to be performed was displayed at the top of the screen with the interpretation result displayed below. For each trial, the application would wait until an input was recorded and classified before requesting the performance of the next sign with a 1 second delay. After classification, the results form the classifier, the correct answer, the instruction and the feature vector was stored. There was no time limit between each trial as the Interpreter do not record if no hand is in view. Additionally, the participants were allowed to consult instructions on how the signs were performed between each sign. During the entire study, participants were shown the skeleton model output from the Leap API to understand how movements were tracked, and so that erroneous interpretation by the Leap sensor could be noted. None of the participants had prior experiences with sign languages, which gave a good indication on how novel Sign Language learners would use the system. For this reason the signs picked for this study were with the requirement of not being overly complex.

After all trials were completed, all datasets were combined to one dataset containing 6 classes with effectively 160 samples per class, and subsequently used to train a separate classifier. The feature vectors from all trials were re-classified with this classifier and the results were then compared to the respective trial's correct answer to calculate a new accuracy score.

5.2 Accuracy and Errors

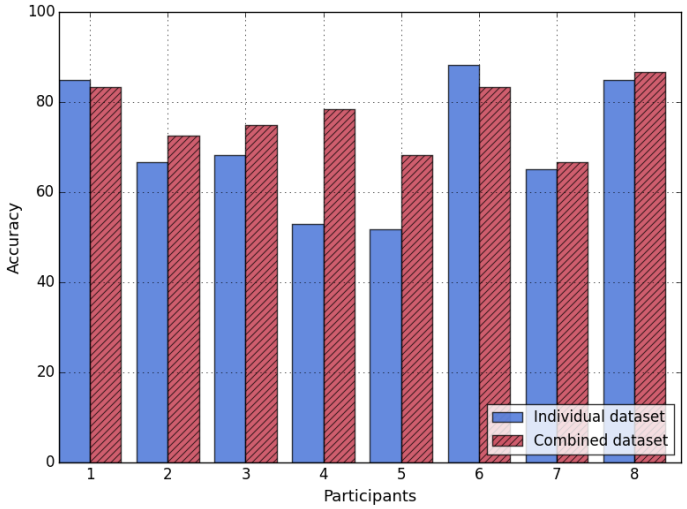


Figure 5.1: Participant accuracy

Using each participant's respective dataset as training data, just above **70%** of all inputs were classified correctly. Re-classifying all feature vectors from every trial with the combined dataset however, gave an average increase in accuracy of 6.4, a total of **76.4%**, showing that an eightfold increase in sample size had a positive effect on the results.
For the average inaccuracy of **23.6%**, the following four types of errors were observed:

5.2.1 Faulty user input

Simply incorrectly produced input. Either in the form of the completely wrong sign or the correct one, but lacking one or more important features, such as a missing extended finger. While this can be remedied by intentionally including alternate or possible versions for signs in the dataset, it would be reactionary as they might overlap with classes yet to be implemented and essentially rendering SLI less accurate based on human errors.

5.2.2 Misleading datasets

Some classes might be affected by a dataset with corrupted samples, causing the center of the cluster to shift in a disadvantageous direction, subsequently rejecting input that would otherwise be classified correctly. The accuracy increase of using the collective datasets indicates that the samples created did not provide ample variation to effectively

cover the 10 inputs per class in the trials. This is supported by the results of cross tested the datasets and trial inputs against each other; 8 classifiers trained with each participants dataset and tested with input from each participant individually (see Appendix B on page 39). With an average accuracy of **58.9%**, the results imply that the accuracy is greatly dependent on a large and heterogeneous database, as with any other machine learning case.

However, it seems that a small sample size can produce acceptable results as well. A classifier was trained with a randomly selected 10% portion of the combined dataset (an approximate 16 samples per class) and tested against the remaining 90%. The accuracy results were **91.8%** and **94.2%** for fixed and gesture signs respectively. As the participants were able to make sure that the skeleton model corresponded with the hand shape before recording, all database samples can be considered produced correctly. This surmises that the dataset is not the major cause responsible for the classification errors, as the accuracy for the trials resulted in a **17%** lower score.

5.2.3 Interpreterer algorithm error

The distinguishing between fixed and gesture signs, as outlined earlier, is a straightforward algorithmic solution with a hardcoded threshold for maximum movement. Despite performing a fixed sign if the movement threshold is surpassed, the input is classified as a gesture regardless. **25%** of all trial errors were fixed signs mistakenly categorized as gestures, with only **6%** of all errors attributed to gestures being falsely categorized as fixed signs.

5.2.4 Sensor error

As Leap Motion has known issues and occasionally produces erroneous data, both participants and the on-screen skeleton model were observed during the trials and cases of clear mismatches were noted. The main issue appeared to occur when the hand entered the sensor view, already in sign shape, especially for signs requiring non-extended fingers. In some cases however, the classifier was able to classify correctly despite erroneous sensor interpretation. For instance, when *Two* was performed, and the skeleton model would occasionally produce a *Two* with the ring finger mistakenly extended, but deemed closer to *Two* than *Three* by the classifier, despite the number of extended fingers. This goes to show that the number of actual inaccurate Leap API outputs might not be represented solely by the overall inaccuracy rate of **23.6%**, as false positives did occur.

5.3 Study Results

Given the results of the study by Marin et al (2014) for only fixed signs with an accuracy rate of **80.86%**, SLI performed reasonably well in comparison, supporting both fixed and gestures. There is however, room for improvement.

The results of **93%** when training and testing with the dataset only, indicates that the improvement issue lies within real time interpretation. With a standard deviation of **13%** for the prior accuracy test, and three participants with a score close to **90%**, there is a noticeable difference in participant performance. Results also depends on users getting accustomed to the system before testing. If participants were allowed more time with the system prior to the trials, the overall results are likely to increase. However, completing the study with participants with no prior SLI nor Leap Motion experience highlighted the limitations of SLI.

Apart from increased user experience to enhance accuracy, there are two major elements of improvements to consider. First simply being to encourage a large sample size per class in order to get datasets with adequate variation, as shown with the **7.6%** increase increasing the sample size 8 times the original size. Tutors creating the dataset for taching use, should be urged to create around 100 samples per sign, using different participants to make sure for some sample variation.

While there is little that can be done to remedy the erroneous output of the Leap Motion sensor, most of the remaining errors were attributed to the *Interpreter* algorithm, which is the second area of improvement. As mentioned, **25%** of errors were due fixed signs being recognized as gestures, which infers that the hardcoded minimum movement threshold is perchance too strict. Additionally, the 0.3 interval allowing the hand to enter the frame might be too short. If a hand performing a fixed sign moves enters the frame too slowly, while still exceeding the movement threshold, it is recognized as a gesture. On the contrary, **6%** of all errors were gestures recognized as fixed signs. If the gesture sign is performed immediately when entering the frame, and the motion finish before the waiting interval is over, the *Interpreter* only register the lack of movement as categorizes it as fixed. The upshot is that the fixed threshold inaccurately favors movements, and needs to be increased, while the waiting interval seems to cause problems for both types of signs.

5.4 Adjustments

A smaller test was additionally conducted to review the issues concerning the fixed/gesture threshold related errors. The combined movement value were recorded for three different signs; A gesture with the whole hand moving and an almost static hand movement with an added slight drift. The third movement consisted of static hand with an exaggerated slowly oscillating finger, to account for gestures with less movement.

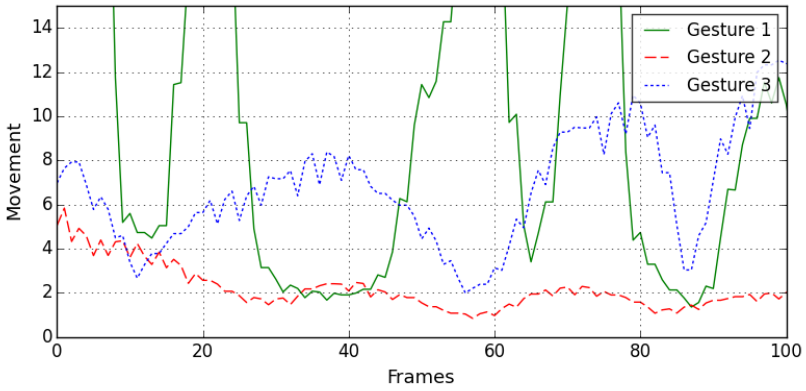


Figure 5.2: Recorded movement

Upon entering the sensor view, all movements have a high, but decreasing rate of movement as the hand is moving to position before initiating the gesture. The previously set waiting interval at 0.3 seconds is in this case unnecessary as the two gestures seem to have started at around 10 frames in, indicating 0.1 seconds as a more fitting fixed time interval. The prior threshold set to 1 is in most cases surpassed by the slight drifting movement, and would have subsequently been categorized as a fixed sign by SLI.

	Gesture 1	*Gesture 2*	*Gesture 3*
Average	7.84	5.92	2.48

Table 5.1: Average movement values for Gesture 1-3

The average value between movements displays a distinguished difference between the fixed sign and gestures movements. With a difference of 3.43 between the lowest and middle averages, the center value of 4.2 and was updated as SLIs new fixed threshold. However, while both gesture movements dips below this threshold for several frames, the movement buffer outlined in section 4.3 on page 13, is still present. In a sliding

window fashion, less than 15 of the last continuous 30 frames have to be below the set threshold to classify as non-movement, assuring some tolerance of start and stops for gestures.

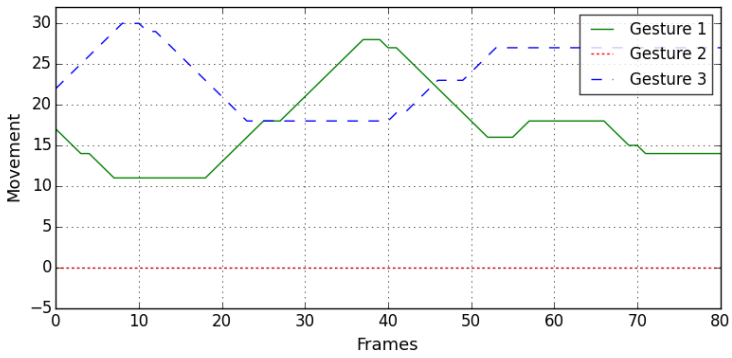


Figure 5.3: Movement buffer

As seen in Figure 5.3, the updated thresholds does a decent job of distinguishing the non-moving Gesture 2 for the rest. In spite of the fact of Gesture 1 inhibiting a value below 15 for the last 30 frames for a period of 10 frames, the fixed frame buffer still have to be filled with a continuous sequence of 40 frames in order for classification to initiate. For Gesture 2, a classification as a fixed sign would being at frame mark 40 as the majority of the frames are under the set movement threshold of value 4.2. With the updated threshold and decreased waiting interval, most of the fixed/gesture errors from the trials should be avoided in similar future trials.

6 Conclusion

This project aimed to develop a functional system to recognize both fixed and gesture based hand signs, using the Leap Motion sensor. Based on previous works and relatively straightforward approaches to recognize static hand shapes, a machine learning approach were considered and implemented to also accommodate for motion recognition.

The criteria of achieving an overall accuracy of 80% (in comparison with [12]) was not met by and approximately 3.6%. However, given the increase in accuracy of using the combined dataset the results are likely to differ greatly over different datasets. Additionally, with the parameter adjustments based on the study results, future quantitative studies will hopefully indicate an increased system accuracy. The non-mathematical approach to distinguish motion and non-motions proved to be a potential pitfall as it does not accommodate for user assumptions, on which the accuracy is greatly dependent. Possible solutions to sidestepping these challenges in future versions of SLI, if they prove to persist even with the new thresholds, might include a machine learning approach to differentiate between fixed and gesture based signs. The type of classification would then be used to determine wether to use the classifier for fixed or gesture based signs. The total sample size of 960 and the average accuracy of 76.4% indicates that SLI would offer Sign Language tutors the means to train fairly usable systems for student use, despite having a lower number of samples in comparison with Marin et al (2014). The results of the quantitative study showed a strong indication that the variation in samples were the dominant factor increasing accuracy, meaning that SLIs database should preferably be trained by a group of tutors.

As outlined, parts of the lack in accuracy was attributed to the interpretation algorithm, some to user mistakes but the majority of errors were caused by faulty sensor output from the Leap system. This is the one reoccurring theme in most papers considering Sign Language recognition using the Leap sensor, and the conclusions of mentioned studies is that the greater room for improvement lies within the accuracy of the Leap API alone. These shortcomings could either be remedied by additional hardware (Kinect or another Leap sensor), or by future upgrades to the Leap systems ability to increase the accuracy of mimic hands in the sensor view. The system in its present state could likely be applied for teaching use of novel learners, but with certain limits. For pedagogic purposes, the skeleton model should preferably be displayed along side the running Interpreter for Sign Language learners to be able to detect cause of mismatch between the SLI input and output. However, the most apparent lack of functionality is the support of two handed signs, which would have to be explored in future versions of SLI.

To improve the learning experience, a great part of the classifier implementation re-

volved around novelty detection. A system being too strict, having little or no tolerance, would be highly impractical. Likewise, from a learning perspective, a too lenient system would be impractical. The investigations to a find a means to assess the quality of classification, deemed the Euclidean distance the most practical approach, as it is less prone to changes in the number of classes of a dataset.

The lack of machine learning experience lead to the consideration of the described template matching approach for a greater part of the allocated project time span. While offering few practical solutions, it was more helpful in terms of highlighting issues that would need to be solved, as well as motivation to find a more adequate solution. The gained understanding of machine learning concepts, not only by applying existing implementations, but developing a classifier from scratch turned out to be an interesting experience. Increasing the knowledge of mathematical estimations to solve problems are skills that will likely prove to be useful in future projects.
While the research and implementation of the classifier were more of a mathematical endeavor, the interpretation algorithm proved to be more of a straightforward programming task. The "accidental" inclusion of both Computer Science fields, offering some variation in types of problem solving was appreciated as the skill set in both improved.

The end result is a Sign Language Interpreter, with the ability to recognizing both fixed and gesture based signs with decent, but limited proficiency. While currently unable to recognize the full scope of signs due to the Leap API and implementation shortcomings, a continuous effort to improve learning aids has the potential to significantly improve Sign Language education.

6.1 Future work

SLI recognizes hand signs including fixed and motions based ones, the next step would be to extend the system to recognize signs both including fixed and gesture signs for one and two hands.

Further development of the Interpreter application could also be considered. A scoring system, incorporating interactive learning sessions would enhance the learning experience of SLI. Supplementary to this functionality, a spaced repetition system could ensure that the learner practices more on poorly learned signs.

Additionally, context in Sign Languages is just as important as in spoken language. Further accuracy could be achieved by extending SLI to take previous recognized signs in consideration. If new input is likely to be part of a sentence, then the corresponding probabilities using techniques from text classification used in machine learning could be used to increase the decisiveness of the classification.

References

[1] AOHL (2015), *Action On Hearing Loss Brochure*

[2] Gallaudet Research Institute (2008), *Regional and National Summary Report of Data from the 2007-08 Annual Survey of Deaf and Hard of Hearing Children and Youth*, Gallaudet University, Washington, DC: GRI, USA

[3] E.L. Newport (1988), Language Sciences, Volume 10, Number 1, *Constraints on Learning and Their Role in Language Acquisition: Studies of the Acquisition of American Sign Language*, University of Illinois, USA, 147-172

[4] J. S. Johson, E. L. Newport (1989), Cognitive Psychology 21, *Critical Period Effects in Second Language Learning: The Influence of Maturational State on the Acquisition of English as a Second Language*, University of Illinois, USA, 60-99

[5] J. Wachs, M. Kölsch, H. Stern, Y. Edan (2011), Communications of the ACM Vol 54, No.2, *Vision-Based Hand-Gesture Applications*, 62-71

[6] F. Chen, C. Fu, C. Huang (2003), Image and Vision Computer 21, *Hand gesture recognition using a real-time tracking method and hidden Markov models*, Elsevier, 745-758

[7] H. Wang, M. Leu, C. Oz (2005), *American Sign Language Recognition Using Multi-Dimensional Hidden Markov Models*, Journal of Information Science and Engineering 22, 1109-1123

[8] F. Weichert, D. Bachmann, B. Rudak and D. Fisseler (2013), Sensors 13, *Analysis of the Accuracy and Robustness of the Leap Motion Controller*, Technical University Dortmund, Germany, May 14, 6380-6393

[9] M. Khademi, H. M. Hondori, A. McKenzie, L. Dodakian, C. V. Lopes, S. C. Cramer (2014), CHI '14 Extended Abstracts on Human Factors in Computing Systems, *Free-hand interaction with leap motion controller for stroke rehabilitation*, University of California, USA, 1663-1668

[10] J.Sutton (2013), ACM SIGGRAPH 2013 Studio Talks, no. 21, *Airpainting with Corel Painter Freestyle and the Leap Motion Controller: A Revolutionary New Way to Paint*

[11] N. Xu, W. Wang, X. Qu (2015), Image and Graphics: 8th International Conference, *Recognition of In-Air Handwritten Chinese Character Based on Leap Motion Controller*, Tianjin, China, August 13-16 160-168

[12] G. Marin, F. Dominio, P. Zanuttigh (2014), Image Processing (ICIP), 2014 IEEE International Conference on, *Hand Gesture Recognition with Leap Motion and Kinect Devices*, University of Padova, Italy, 1565-1569

[13] C. Chuan, E. Regina, C. Guardino (2014), 13th International Conference on Machine Learning and Applications, *American Sign Language Recognition Using Leap Motion Sensor*, University of North Florida, USA, 541-544

[14] L.E. Potter, J. Araullo, L. Carter (2013), Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration, *The Leap Motion controller: A view on sign language*, Griffith University, Brisbane, Australia, November 25 - 29, 175-178

[15] M. Mohandes, S. Aliyu, M. Deriche (2015), International Multi-Conference on Systems, Signals & Devices, *Prototype Arabic Sign Language Recognition using Multi-Sensor Data Fusion of Two Leap Motion Controllers*, King Fahd University, Saudi Arabia

[16] E. R. Davies (2005), *Computer Vision*. Elsevier, London

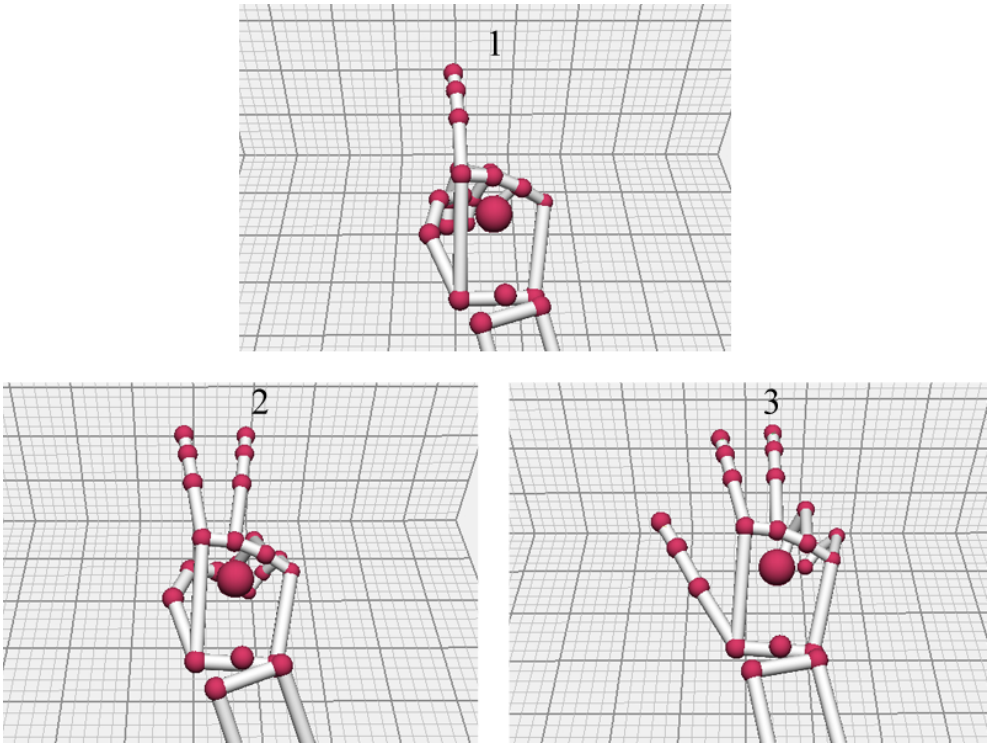
[17] H. Dreyfus, S. Dreyfus (1992), AI & Society, *What Artificial Experts Can and Cannot Do*, University of California, USA, 18-26

[18] G. John, P. Langley (1995), UAI'95 Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence, *Estimating Continuous Distributions in Bayesion Classifiers*, Morgan Kaufmann Publishers, San Mateo, USA, 338-345

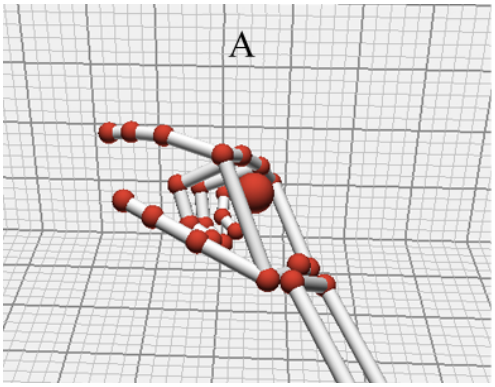
[19] T. M. Mitchell (1997), *Machine Learning*, McGraw-Hill, 177-198

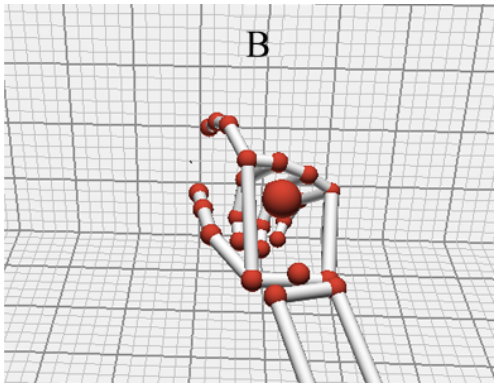
Appendices

Appendix A

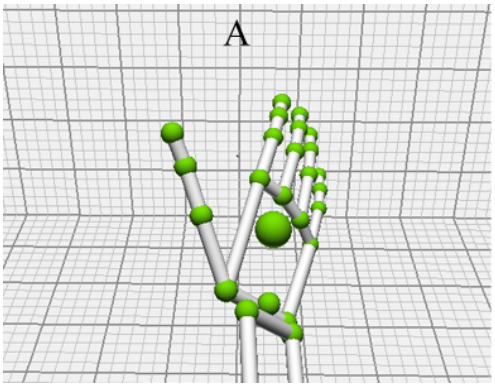


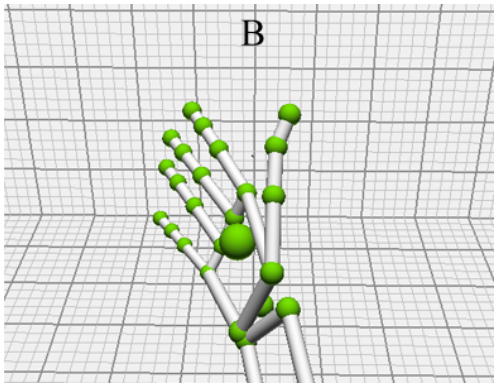
ASL for the numbers 1, 2 and 3



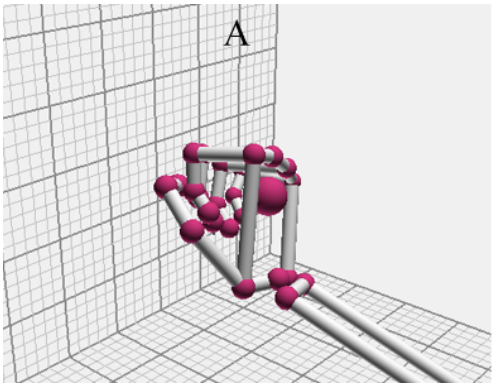


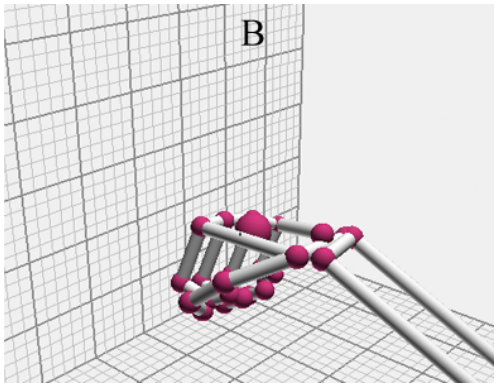
ASL for ’Green’





ASL for ’Blue’





ASL for ’Yes’

Appendix B

		Participant trials							
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Participant datasets	**1**	85	50	57.9	86.4	56.2	73	52.7	100
	2	67.6	66.7	65.8	63.6	46.8	69.2	52.7	86.2
	3	79.4	66.6	68.3	54.5	56.2	92.3	69.4	96.5
	4	44.1	41.6	34.2	52.9	43.7	50	30.5	62
	5	50	52.7	42.1	36.4	51.6	80.7	47.2	68.9
	6	52.9	47.2	47.4	36.4	37.5	88.3	52.7	79.3
	7	61.7	41.6	57.9	22.7	25	84.6	65	65
	8	64.7	41.6	36.8	59	50	88.46	50	85

Average	***59.26***

Cross test of participants trial data and datasets