Automated Visualised Translation from English to British Sign Language

Nicolaos Moscholios

Pembroke College

University of Oxford

Supervised by Dr. Joe Pitt-Francis

A thesis submitted for the degree of

Master of Computer Science

Trinity 2016

Abstract

A large number of people in the world today are born deaf and many rely on sign language, with British Sign Language (BSL) being the most widely used method of signed communication in the UK. BSL is structured in a completely different way to English and, like any language, it has its own grammar. Unlike other languages, it is uncommon for individuals to take interest in it especially when unaffected by deafness. In order to establish a communication bridge between English speakers and deaf individuals there have been attempts at building a formal model of translation. This thesis describes the development and implementation of an interactive online tool to automatically translate from written English to BSL, in order to increase the interest in sign languages even outside the deaf community. Using a rule-based machine translation approach we transform the English sentence to BSL embedded into a 3D animation format, subsequently displayed in the browser in real-time. The system is tested both through an empirical evaluation to assess the translation accuracy, and user testing. Two expert BSL linguists also make an important contribution to the evaluation. The tool is available on any browser and can be used on personal computers, smart-phones and tablets.

Acknowledgements

First of all, I would like to thank my supervisor Dr Joe Pitt-Francis for his valuable guidance throughout the development of the project, especially in writing this report.

My sincere thanks also to Rachel Sutton-Spence, co-author of *British Sign Language: an Introduction* for taking the time to review this project and for her highly reliable feedback, as well as to Adam Schembri, Lecturer in Sociolinguistics at the University of Birmingham for his valuable comments and critique.

Last but not least, my deepest gratitude goes to my family and friends. To my parents for supporting me and trying to help me out regardless of the complexity of the problems I would present them with, both personal and technical. To my friends and colleagues for the memorable times spent in the Computer Laboratory trying to write our thesis, with the occasional laughing sprees and (mostly) deserved coffee breaks. Finally, thank you to my girlfriend for coping with my exasperating self during the last couple of months, motivating me and telling me not to give up.

Notes

The work described in this thesis is available online at `translate.nicmosc.com`. The system is browser based meaning no software installation or technical knowledge is required to access the interface. Examples discussed throughout the report marked by an asterisk ($*$) can be viewed on the website. In order to complement the reading of this work it is highly recommended that the reader try those examples for a more comprehensive understanding of the behaviour of signs in BSL. Please note that if the website has been idle for a certain amount of time, it may take longer to load initially.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

For most people around the world, communication is achieved orally by speaking. Unfortunately some are born deaf or are affected by hearing loss with time. Thus, it is necessary to use another medium of communication. Today there are about 1 million "functionally deaf" UK individuals [2], that is, that require the use of sign language to converse. Additionally there are around 11 million people affected by some degree of hearing loss and while many benefit from hearing aids, sign language bypasses the need to speak, providing a medium for everyone to understand each other.

1.1 Motivation

Communication between members of the Deaf Community in the UK is enabled through British Sign Language, however the ability to converse is eminently reduced when hearing and Deaf[1] individuals need to communicate. Deaf people are taught to read and write English from a young age, but it has been found that they have difficulties understanding text with a reading level above that of primary school students [17]. Thus it is not enough for hearing individuals to rely on written text to interact with their community. Often, those people without hearing deficit learning sign language do so because of a deaf child or relative, meaning only a handful of hearing people have the ability to communicate with the Deaf. Those who wish to learn the language can take lessons from BSL teachers and practice by engaging with the Deaf community. However learning a new language, especially if very different

[1]"Deaf" with a capital D refers to people whom are affected by deafness but who are also active within the Deaf Community and for whom BSL is their primary language. There are also deaf people (with small d) that in contrast prefer not to rely on BSL and use other means such as cochlear implants to communicate.

from our mother tongue, can be time consuming and expensive if one wishes to follow professional courses. Thus, people whom do not require to learn BSL rarely do so because of pure interest, as opposed to learning other spoken languages. Currently there exists no easily accessible tool to get an idea of how signed languages work. Certainly, books are useful to learn the grammar, though it can be problematic to some since images depicting gestures are not very easy to interpret. Other online resources such as UCL's BSL Sign Bank [37] and `SignBSL.com` [32] offer clips of signers showing words and phrases in BSL that can be useful to learn specific signs. The main issue is that they are highly static. Analogously to an English dictionary, these online resources are only useful for looking up individual signs, and do not offer a proper visual explanation of the internal workings of the language. An online *translation* tool from English to BSL may be the answer to increase people's interest in signed languages. Not only could it be used as a complementary learning tool to enhance the understanding of what is read in the books, but also as a way to look up any sentence on the spot and get an idea of what BSL is like.

Automated translation has been an ongoing challenge in the field of computational linguistics, and we are finally seeing very positive results thanks to the recent developments in machine learning. Since written languages are heavily dependent on their syntactic components, they are relatively easy to formalise in such a way that a machine can interpret a sentence and perform operations on it to transition from one language to another. More recently, there have also been attempts at doing so with signed languages such as ASL (American Sign Language) and BSL. Given that there is no formally typed grammar for BSL, it is not easy to find a model that encapsulates all key elements involved, such as facial expressions and eye gaze. In addition, since the language is only signed, the only way to see if an interpretation is correct, is through visual output. While some past implementations have been successful, due to the then high computational power necessary to run these systems of automated translation, most of them are now outdated or have been left at a prototype stage. In 2006, IBM created a fully working system [15] that allowed users to speak and see the translation from English to BSL in real time; however it has been abandoned and is not maintained any more. Moreover, none of the previous approaches was performed with portability in mind. What made systems such as Google Translate so popular, is that they were accessible anywhere, on any platform. Developing an analogous web application for sign language translation would make it easy for anyone to access the necessary information at the appropriate time.

1.2 Objectives

This project will build upon previous work and attempt to solve the interlingual translation from English to British Sign Language by implementing a web application where signs are visualised through a virtual agent. Behaviour will closely follow that of other established translation systems such as Google Translate where users can type a sentence and get a result in real-time. Meanwhile, by overcoming the limitations of pre-synthesised video it should promote the use of a virtual and responsive 3D environment through a lightweight data approach. The rule-based translation methods (discussed in Section 2.2) and 3D animation techniques will allow users to not only find out what a particular sentence corresponds to in BSL, but also practice their skills thanks to the multiple media controls to adapt the playback to their likings and achieve a better understanding. It is imperative to appreciate the fact that the system *cannot* replace a real signer and even less a teacher. Because the translation may be incorrect at times, as with virtually any existing translation system, it is expected to be used as a complement to already established knowledge and for uninformed people to get a general idea of BSL for the first time, and not as the sole learning tool. Being an online web application, it will be the first SL translation system to employ modern cross-browser languages such as JavaScript and Python to achieve portability. Figure 1.1 shows a conceptual overview of the system pipeline; this figure will be referenced multiple times throughout the report. The main objectives of the project are:

- Design and build a machine translation system to transition from written English to British Sign Language through animation. The system should include the analysis of English sentences using techniques learned in the Computational Linguistics course to subsequently transform it into a BSL gloss form[2] and an appropriate data format to generate animations. The system should take into account both manual and non-manual features (see Section 2.1.6). The translation should achieve satisfactory accuracy, enough to translate simple sentences and possibly more complex constructs.

- To visualise the output of the translation, a web application will use web-compatible 3D animation techniques to animate a virtual agent that will perform the gestures. The avatar will be carefully chosen and rigged with a skeleton to ensure all possible movements of the arms and face can be executed. The

[2]Glosses are a representation of the meaning of signs in their English form, for example the sentence *I am eating* is represented as ME EAT, where each of the two glosses describes the meaning of an individual sign. Glosses are further discussed in Chapter 2.

accuracy of the animations is essential to ensure the appropriate usability and understandability of the signs and non-manual features. Animations will be created with the Blender animation software[3], extensively used in the Computer Animation course. Users should be able to control the playback of animations through media controls like pause, play, speed change etc., and also change point of view of the camera.

- Since one of the key features of this project is portability, the website will need to be accessible on multiple platforms including personal computers, tablets and smart-phones. It should work on popular modern browsers such as Chrome, Safari and Firefox. Communication between the server and browser will be achieved through a micro web framework discussed in chapters 3 and 4.

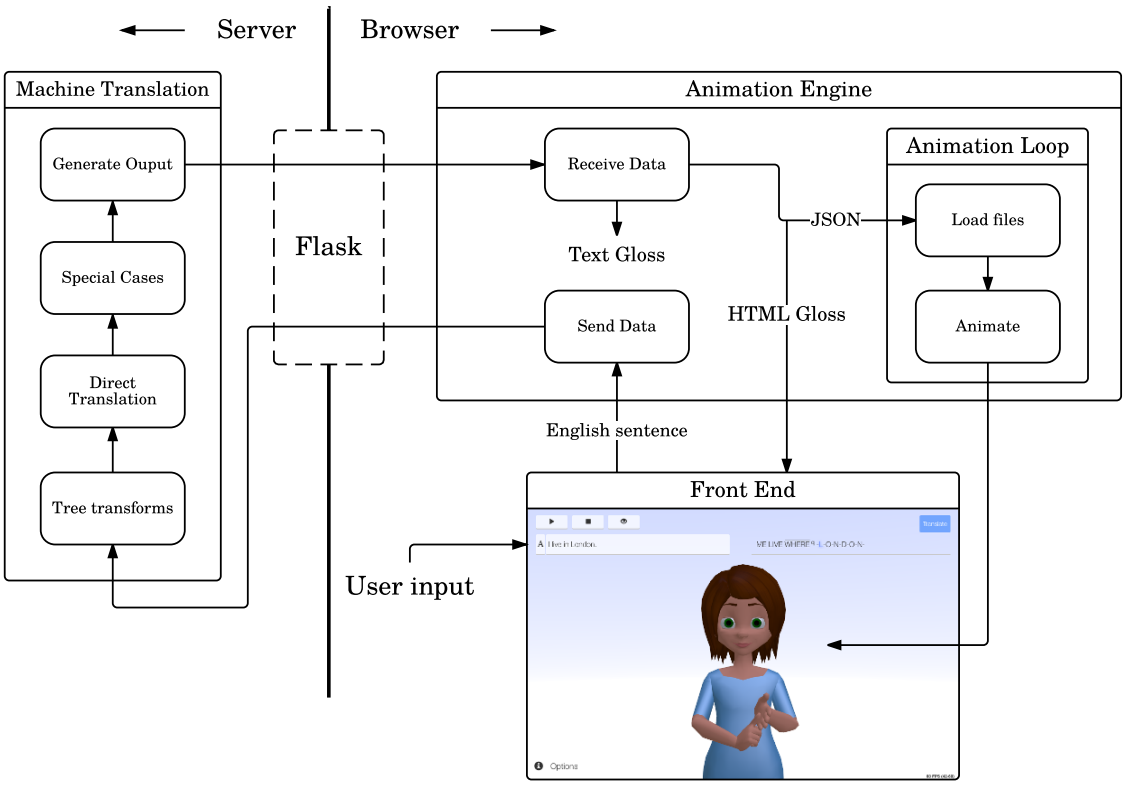


Figure 1.1: Conceptual overview of the system

1.3 Structure

The report will read as follows: in Chapter 2 we give a brief introduction to the linguistics British Sign Language and its differences from written English, as well as

[3] `https://www.blender.org`

existing methods of translation from written to written, and from written to signed languages. Chapter 3 covers the design choices for the system, that is, the development environment and the programming languages, the conceptual structure of the pipeline and the end-user interaction. Following up in Chapter 4 we explain in detail the implementation of both the translation methods and the animation techniques utilised, as well as the communication between the two modules. Finally Chapter 5 discusses the evaluation of the system both in terms of translation accuracy, feedback from the user survey and BSL linguists, while Chapter 6 concludes the report along with project achievements, personal remarks and future work.

Chapter 2

Background

This chapter concerns Machine Translation from written English to British Sign Language. As such it is separated in two main sections: a general overview of BSL grammar and essential linguistics components, and Machine Translation theory and its particular applications in BSL translation.

2.1 British Sign Language: Linguistics Overview

This section will give a short introduction to the linguistics of British Sign Language, including common grammar rules, its morphology and how it differs from written English. Please note that only the most general information is described here in order to follow the concepts discussed in Chapter 4 and other parts of the report, which are based on this theory. However it is not necessary to read this section before the one following as any approach that requires knowledge of BSL linguistics will reference this section for further understanding if necessary. The following examples and explanations are based on the book by Woll and Sutton-Spence, *The Linguistics of British Sign Language: an Introduction*.

British Sign Language, often shortened to BSL, has been an officially recognised language since 2003 [33]. Like English, it has its own grammar [1] and its own lexicon. While not as large as its written counterpart, there are enough signs to convey most common ideas in sign language as many can be combined to create new meanings (*compounds*, see Section 2.1.2). However BSL is not the only signed language; in fact each country has their own variant with different dialects by region. It is then easy to imagine that communication between Deaf individuals coming from different countries can also be difficult. To counteract this problem, there have been attempts at establishing an International Sign Language, albeit without success. Still, signed

languages coming from the same written relative do share features, such as BSL, Australian and NZ Sign Languages all coming from British English. Similarly ASL and ISL (Irish Sign Language) come from LSF (Langue des Signes Française) [8]. In view of the fact that this project is being implemented in the UK, BSL was seen as the most appropriate sign language.

Because BSL can only be transmitted through signing, there is no formal written form. Many different ad hoc notation formats exist such as Stokoe, designed for ASL by William Stokoe only for representing hand movements [34], thus carrying no information about non-manual features and later adapted to BSL by Kyle and Woll [16], HamNoSys (Hamburg Notation System) that can represent any signed language [12], SignWriting which uses visual icons to represent parts of the body [35] and more. Unfortunately all of the above require additional knowledge to be decoded as they employ specialised notations with symbols that cannot be understood by an untrained individual. Thus the most commonly used method to represent sign language without needing to learn the notation is through *gloss*. Gloss uses the most basic representation of the sign in its English written form. For example if one wants to describe a situation where a girl is eating an apple, the gloss would be GIRL EAT APPLE. The advantage of this notation is that we can clearly understand what a sentence means as we associate the sign to its English meaning. The main disadvantage though is that there is no information about the signs whatsoever, we only know that the particular sign for GIRL is signed first, followed by the sign for APPLE and so on. Table 2.1 shows an example sign for every notation type.

Gloss notation also carries some information about non-manual features (see Section 2.1.6) for example the sentence "Where do you live?" would be YOU LIVE $\overset{\text{br}}{\overline{\text{WHERE}}}$, where br represents a brow raise for asking questions. This format is used in the rest of the report and in the application.

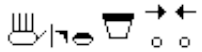
English	Sign	SignWrite	Stokoe	HamNoSys	Gloss
What?					$\overset{\text{br}}{\overline{\text{WHAT}}}$

Table 2.1: Examples of different BSL written forms; from `http://www.signwriting.org`

Let us now discuss the particular features of BSL and how it is structured as a language.

2.1.1 Structure of sentences

In English, sentences follow very similar patterns. Most often we find a noun phrase followed by a verb phrase, each one containing information about the subject and the object (Figure 2.1). BSL, just like English, has its own rules for ordering signs within a sentence. Although sign order differs from ordering of English words, many signers interacting with learners of BSL sometimes employ the spoken English ordering to facilitate understanding. A sentence is divided in two basic parts: **subject**, the theme or topic, most often a noun, a noun phrase or a pronoun, and **predicate**, the remainder of the sentence normally describing the action of the subject. BSL borrows some elements from English, such as the concept of pronouns, adjectives, verbs etc. but also includes *proforms* and *classifier predicates* which do not exist in English.

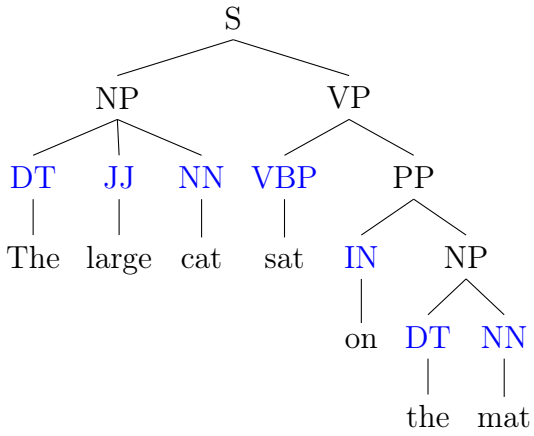


Figure 2.1: English sentence syntax tree. Blue text represents word Part of Speech tags.

Proforms are "anything that refers to, and stands in the place of, something previously identified", that is, one may sign a word such as CAR first, and then when describing an action of the car such as "driving in a zig-zag motion" the dominant hand assumes the proform *shape* associated with the car. Examples of hand shapes include 'B' for vehicles and round/flat objects like PLATE (generally objects having 2 dimensions), 'V', 'V̈' and 'G' (one dimension) for MAN etc. Figure 2.2 shows examples for each[1]. When we use a specific handshape to describe the action or features of a word belonging in a specific group[2] is what is called a classifier predicate. In

[1]There are 22 handshapes in total and interestingly most of the shapes correspond to the ASL signed alphabet.

[2]Groups are arranged by features that can represent physical properties of objects. A human figure performing an action such as walking is symbolised by a reversed 'V' shape to mimic the legs while a static figure would be expressed through a 'G' shape to delineate a whole body.

another example we may want to sign "The car goes under the bridge", thus we would sign CAR and BRIDGE and then use the *classifier* for car VEH-CL (vehicle classifier, represented by the B hand with palm facing down) and the bridge classifier 3D-CL ($\ddot{\text{B}}$ hand) to show the car moving under the bridge object.

B	$\ddot{\text{B}}$	V	$\ddot{\text{V}}$	G
				

Table 2.2: Some BSL proform handshapes; from `http://www.lsfdico-injsmetz.fr`

Pronouns are used similarly to written English but there are some differences. Firstly, there is no distinction between masculine or feminine. Instead a G shape hand is used, normally glossed as Index followed by a number to specify which part of space we are referring to. For instance

John told Mary that he loved her

$=$ -J-O-H-N- Index_1 TELL -M-A-R-Y- Index_2 Index_1 LOVE Index_2 $(*)$

Recall that any of the examples marked by an asterisk $(*)$ can be viewed on the website. Those which are *not* marked as such are either known to behave erroneously because the grammatical features involved are not handled by the system or because the animations for some of the included signs is not available at the time of writing this report. Of course *all* of the examples in this report can be demonstrated, however the outcome will not be as expected. Subject Verb Object (SVO) is the preferred word order in spoken English, while SOV is more common in BSL. Yet, as mentioned previously many learners prefer to use SVO when signing. This works most of the time however some verbs require an SOV construct such as ME PIZZA EAT as the verb EAT is modified by the type of object that is being eaten. In other words the sign for eating a pizza will differ from the one for eating pasta.

2.1.2 Morphology and Morphemes

A morpheme in BSL is considered as the smallest unit of meaning in a word or sign. We can combine them to form signs that have several meaningful parts but are still considered a single sign. Figure 2.2 shows the BSL sign morphology tree. We distinguish between *monomorphemic* signs (cannot be further subdivided) like TRUE,

SAY, MOUSE and *polymorphemic* which are a combination of 2 or more morphemes. For example PROMISE is the combination of the sign for SAY and TRUE whilst CHECK = SEE + MAYBE. We also categorise morphemes as **Free** that can stand alone (i.e. monomorphemes like RED, TRUE, SAY) and **Bound** which have a meaning but *must* be combined with at least one other morpheme, and **Plural** morphemes.

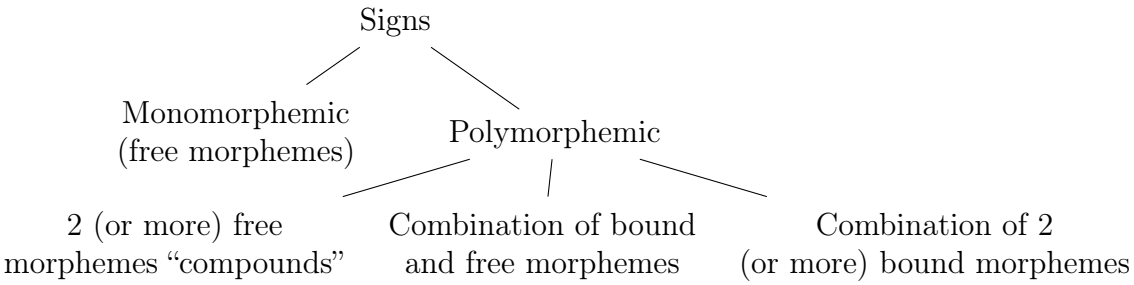


Figure 2.2: BSL sign morphology schema

Free morphemes are often combined to form a **compound**, a sign with a different but related meaning. Some are borrowed from English like BALANCE-SHEET while others aren't: BLOOD = RED + FLOW, TIGER = ZEBRA + ANIMAL. When combining free morphemes, the compound must appear as similar as possible to a single sign, thus the originals are modified by rapid transitioning (both signs are accelerated), the initial hold of the first sign and any repeated movement in the second sign are lost. For example since MOTHER = -M-M- and FATHER = -F-F- then PARENTS = -M-F- and not -M-M-F-F-. Please note that the notation of single letters wrapped by dashes "-" refers to fingerspelling. Fingerspelling is the action of signing letter by letter a word borrowed from English; fingerspelling is discussed in greater detail in Section 2.1.5.

Bound morphemes must be attached to another free morpheme. The sign for DRIVE-CASUALLY combines the free morpheme DRIVE (verb) and bound CASUALLY, since we need to specify what action is being performed in a casual manner. Similarly the agreement verb ASK (see Section 2.1.3.2) requires the subject and object such as YOU and ME (both free morphemes) to form the complete verb YOU-ASK-ME.

Plural morphemes include both free and bound morphemes and can carry information about nouns and verbs. In English this trait corresponds to the terminal *-s*. In "cats" we find cat + s where [s] is the form of a *bound* morpheme: it cannot stand

alone. However in BSL we cannot just add a terminal to the sign. Instead plural morphemes can be attached to a sign, modify the sign itself or appear separately. For example the plural "children" can be signed as CHILD repeated multiple times, or by signing TWO CHILD if we know there are 2 of them, or otherwise using pronouns like CHILD THEM to mean that there are more than one.

Adjectives also carry some morphological information as they convey a particular feature of nouns or pronouns. Adjectives can be **attributive** when they occur in the noun phrase and appear before, after or within a noun e.g. SHIRT WHITE, or **predicative** when they act like a verb e.g. MAN Index_3 TALL where the man is "executing" the action of being tall. However they are not used very often, since signs can be modified directly instead; the sign for BOX involves making a square shape with both hands, however SMALL-BOX and LARGE-BOX will modify the movement for the base sign by leaving less or more space between the hands respectively. Signs that cannot be directly modified make use of the normal adjective signs (there is indeed a sign for both small and large) and can be preceded by a premodifier like VERY, QUITE etc. Adjective signs in comparative (*-er*) and superlative (*-est*) form are modified by making the initial hold very long and tense[3], followed by a rapid release, where the degree of tension depends on the modifier.

2.1.3 Verbs

2.1.3.1 Tense, Aspect and Mood

In English when we want to refer to an action that will happen in another point in time we conjugate the verb ("He will go home") or use words that convey the same idea ("He's going home tomorrow"). In BSL, keywords such as WILL and TOMORROW can be used to show an action happening at a different time. Furthermore, time is always set at the beginning of a sentence and is also emphasized through eye gaze (see Section 2.1.6.3). For instance

$$\text{I went to London yesterday} = \text{YESTERDAY ME GO -L-O-N-D-O-N-} \qquad (*)$$

Aspect is the internal timing of things and describes if something is happening relative to another event, how long it went on for, if it is not finished yet and so on. For example if we are describing the action of someone looking for a long time, the sign

[3]Not to be confused with the *tense* of a verb

LOOK-FOR-A-LONG-TIME is simply the base sign LOOK with the hands kept in position for longer and a facial expression with eyes open wider.

What in English are used as modal auxiliaries like *may, can, shall, must* are performed by signing the corresponding signs either before or after the verb e.g. CAN HAVE, modifying the verb itself by making it stronger and bigger depending on the mood e.g. MUST-ASK is stronger than CAN-ASK, or by using facial expressions.

2.1.3.2 Verb types

In BSL we define two types of signing space: topographic and syntactic. The former describes a real world location according to the *actual* features. For example if standing outside one may directly point at a tree they want to describe. On the other hand, the latter is a space "created within the language" and may not reflect the real world since it is used figuratively. In the sentence

$$\text{I gave my aunt a book} = \text{AUNT Index}_3\ \text{Index}_1\ \text{BOOK}_1\ \text{GIVE-BOOK}_3$$

The real aunt is not where the signer is pointing (Index_3). Furthermore, we distinguish between 3 types of verbs:

- Plain verbs: these signs show little modification and do not move through space. Any information about what and how many entities the verb describes is given through pronouns. E.g. "I like him" is signed ME LIKE HIM i.e. all signs are separated.

- Agreement verbs: can be modified to show additional information and are signed in syntactic space. "I give you ..." is ME-GIVE-YOU, where the hand moves *from* the ME position *to* YOU with the specific handshape (verb stem) for GIVE; pronouns are not signed individually. This movement has 3 steps: subject agreement marker, verb stem and object agreement marker. This means that in YOU-GIVE-ME the movement is essentially reversed.

- Spatial verbs: use topographic space instead and are used to describe a trajectory, speed of movement and location of an action. Examples include RUN-DOWNSTAIRS, DRIVE-TO etc. Most often these use classifier predicates to describe surrounding entities (see Section 2.1.1).

2.1.4 Questions and Negations

When asking a question, the whole sentence is accompanied by an eyebrow raise, a head tilt and opened eyes. The question "Do you like tea?" is written in gloss as $\overline{\text{YOU LIKE TEA}}^{\text{q}}$. Some questions only include a brow raise on parts of the sentence, such as in "You have three children right?" = THREE CHILD HAVE $\overline{\text{RIGHT}}^{\text{q}}$ since we are asking for confirmation.

Wh- question, which include *what, why, where, when, who, which, how* usually follow the same rule as the above where only the wh- word is accompanied by a question facial expression. For example:

$$\text{Who are your parents? = YOUR -M-F- } \overline{\text{WHO}}^{\text{q}} \quad (*)$$

$$\text{Where did he go? = Index}_1 \text{ GO } \overline{\text{WHERE}}^{\text{q}} \quad (*)$$

However questions can also be used in sentences which are not questions by nature. In fact the following

$$\text{I love John because he's nice = ME LOVE -J-O-H-N- } \overline{\text{WHY}}^{\text{q}} \text{ Index}_1 \text{ NICE} \quad (*)$$

I won't go to the beach if it'll rain tomorrow

$$= \overline{\text{TOMORROW RAIN}}^{\text{q}} \text{ ME } \overline{\text{NOT GO}}^{\text{neg}} \text{ BEACH} \quad (*)$$

essentially turning the sentence into a rhetorical question followed by the answer. The same happens when we try to describe the state of an object or person:

$$\text{The keys are in the kitchen = KEYS } \overline{\text{WHERE}}^{\text{q}} \text{ KITCHEN}$$

$$\text{I live in Essex = ME LIVE } \overline{\text{WHERE}}^{\text{q}} \text{ -E-S-S-E-X-} \quad (*)$$

Sentences that contain negations are signed considering three main elements:
Facial expression: the lips are pushed out and the eyes are narrowed
Head movements: the head turns to the side and is held there (accompanies a specific sign) or alternates between the left and right sides. The latter can negate both whole sentences or single signs e.g.

$$\text{I'm not eating pizza = } \overline{\text{ME PIZZA EAT}}^{\text{neg}}$$

$$\text{ME PIZZA } \overline{\text{EAT}}^{\text{neg}}$$

$$\text{ME PIZZA EAT }^{\text{neg}}$$

Negation signs: these are specific hand gestures to show that a negation is happening, the most common ones are a flat hand, palm down twisting up following a verb or adjective (often used as a suffix with SEE +neg, HAVE +neg for denial of possession or experience) and the NOT sign. Thus the previous example can also be written as

$$\text{I'm not eating pizza} = \text{ME PIZZA } \overset{\text{neg}}{\overline{\text{NOT EAT}}}$$

Additionally, some signs when negated are completely different from the original and are not just preceded by NOT and the *neg* expression. For example the verb "know" is performed by the dominant hand's fist with the thumb up tapping on the forehead, while its negation NOT-KNOW sees the two flat hands moving away from the forehead. This rule also applies to other expressions, such as

$$\text{He didn't show up} = \text{Index}_1 \overset{\text{neg}}{\overline{\text{NOT SHOW-UP}}}$$

2.1.5 Fingerspelling

Fingerspelling is the action of using the signed alphabet to spell out a word, normally borrowed from English. The most general rule for figerspelling is that if the word is less than 3-4 letters long it is spelled in full, otherwise it can be abbreviated. The fingerspelling notation in gloss form is -B-B-C- or -b-b-c- (example for the commonly known broadcasting company). There exist a few abbreviation rules and the most common ones are:

- Using the first syllable e.g. January = -J-A-N-
- When the second letter is 'h' we retain the first 2 e.g. chapter = -C-H-
- We can also use the first and last letter (mainly used for places) e.g. Glasgow = -G-W-

Many names of places also have their own sign and as a result are not fingerspelled. However if an unknown word is introduced in a context for the first time, such as personal name or a place name, then it is first spelled in full, and the following time the abbreviation is used. The name Hannah may be signed as -H-A-N-N-A-H- first and then just as -H-. On the other hand it is not uncommon for signers to use body/character features to refer to someone. For example if I would like to talk about a person who wears glasses, I would first need to sign their full name, followed by the sign for glasses. After setting that context the signing partner would know that every time GLASSES is signed, that particular person is being referred to.

2.1.6 Non-manual features

Non-manual features represent any action carried out by the signer which is not performed by the hands and are used to represent spoken language mouth patterns in combination with signs, enacting actions and setting the context like verb tenses and questions. These are very important in BSL and are often mistakenly neglected by the uninformed individual.

2.1.6.1 Spoken and Oral Components

Spoken components are used to better identify the sign being used. Nouns that are fingerspelled are also mouthed when the last letter is signed; this also applies to abbreviated signs i.e. only one letter is signed but the whole word is mouthed. They are also extremely useful with homonyms[4] since only through mouthing it is possible to distinguish two meanings of a sign that has the same gestures e.g. FINLAND and METAL. The mouthing shape is borrowed from the English spoken version of the words.

Oral components enact real-life actions such as laughing or biting, where the sign is entirely mouthed. The hands are not involved since the act of laughing can be performed by opening the mouth and closing the eyes like one would in a comparable authentic humorous situation. Additionally, mouthing can be used to represent negations as described in Section 2.1.4. On the other hand it is indeed possible to describe the same actions through completely manual signs, albeit these are rarely used.

2.1.6.2 Facial Expressions and Head Movements

Facial expressions are used to mark a question or a negation (as seen above) and to show emotions e.g.

$$\text{I was happy when dad arrived} = \overset{\text{happy}}{\overline{\text{ME HAPPY}}}\ \text{WHEN -D-D- ARRIVE}$$

They are also used in combination with a head nod to mark the topic. The topic can be *temporal* when describing a situation in time such as "When I...", *spatial* and *nominal* when describing a location or an entity. Most often,

$$\text{The dog chased the cat} = \overset{\text{hn}}{\overline{\text{DOG}}}\ \text{CAT CHASE}$$

where hn stands for head nod.

[4]Homonyms in BSL do not refer to simple synonyms. Instead these are signs that, even though share the exact same gestures, carry completely different meanings expressed by non-manual features.

2.1.6.3 Eye Gaze

Eye gaze plays an important role during signing because of multiple reasons:

- Lexical distinction: some signs such as GOD and BOSS are signed in the same way (homonyms) and are distinguished by the eye gaze; when signing GOD the eyes look upwards, similarly to how spoken components are used (see Section 2.1.6.1).
- In conjunction with location and movement: the sign for HE/SHE (glossed as Index) may be in the same location as the sign for YOU but the eyes pointing somewhere else than the listener imply we are discussing another person. In addition the eyes follow the hand during movements e.g. when describing a rolling ball.
- To mark time: in conjunction with a movement of the head, looking on the side can indicate the past, ahead or down indicates present, while looking up indicates the future.

2.2 Machine Translation

Now that the necessary background in BSL linguistics has been laid out, let us discuss machine translation and how it has been previously applied to signed languages. For this section, the book *Speech and Language Processing* by Jurafsky and Martin was used as background reading.

Translation between written languages has been an ongoing research topic in the field of computational linguistics. The basic idea is that we wish to find relations between the syntactic components of the source and target language, and then model them in a way that a machine can understand it. There exist currently two main automated translation approaches: statistical and classical. Statistical Machine Translation (SMT) is more recent than the latter, and has become very popular in recent years thanks to the advancements in machine learning and major increase in processing power of modern computers. It is based on the concept of analysing huge bilingual corpora, which are essentially lists of sentences in the source language, with each having a set of possible reference translations. Classifiers are trained on these corpora and assign a probability of translation for each word in the sentence: this technique is called *word alignment*. In Figure 2.3 each word from the source sentence

is assigned 0 or more words from the target. For instance the Spanish alignment of "did not" is "no" and the verb "slap" in this context set in the past corresponds to "dio una bofetada". If the context of the sentence was set in the future the verb "slap" would translate to "dará una bofetada".

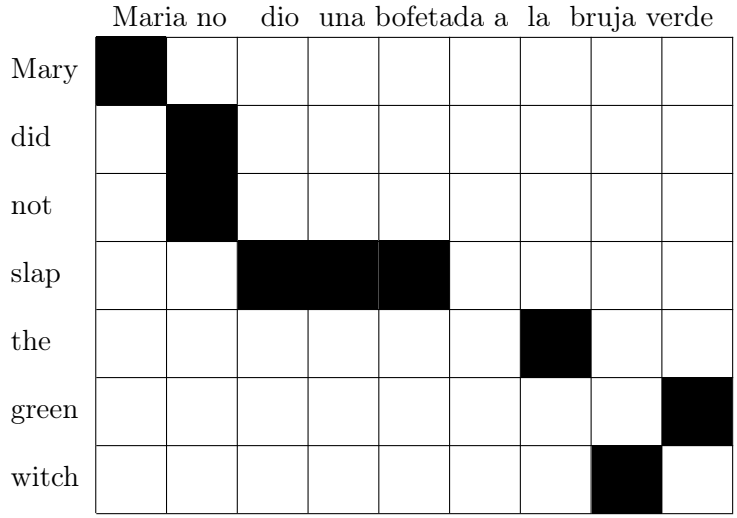


Figure 2.3: Word alignment English-Spanish

However SMT is only possible when large amounts of parallel data are available. In the case they are not, other methods are required. Classical, or Rule-Based, Machine Translation (RBMT) requires deep understanding of both the source and target language in order to work. It involves the creation of "rules" that closely follow the syntactic and semantic constructs of both languages. The Vaquois triangle illustrates the multiple techniques, often combined, in rule-based translation (Figure 2.4).

Direct translation stands at the lowest level and is simply an equivalence relation between the words in the source and target languages. This step is usually performed via a dictionary lookup: word to word translation corpora are much more common than whole sentences. **Transfer** is a process that involves the *transformation* of the source language structure into another that resembles the target. Syntax trees can be generated by analysing both languages and then transforming the source to the specification of the target. From there direct translation can be applied to achieve a complete translation. Finally **interlingua**, while similar to transfer, is the transformation of the source into a universal representation that can be applied to any language. This is most useful when translation needs to occur both from and to the source and target.

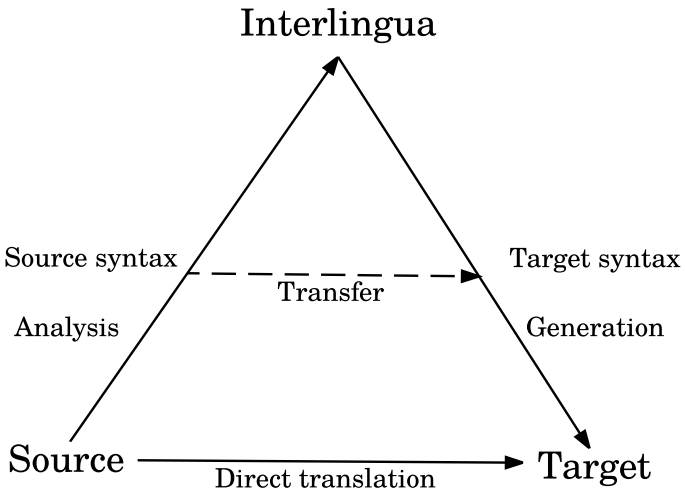


Figure 2.4: Vaquois triangle for Rule-Based MT; illustration inspired by https://en.wikipedia.org/wiki/Machine_translation

While statistical MT approaches usually achieve much higher accuracy than non-statistical methods because of the sheer amount of data used in the process, there have been multiple successful attempts at creating rule-based systems to cover languages with little or no corpora, even though they do prove to be much more time consuming to build and maintain [19]. The most popular open-source system currently using shallow-transfer rules is Apertium[5]. It uses a language-independent specification so that resource-poor language pairs such as Serbo-Croatian/Macedonian can be added gradually [26]. A more recent approach has been that of *hybrid* MT, where both statistical and rule-based methods are combined to achieve more flexibility and accuracy, sometimes using rules to correct the output of the statistical module [31], generate phrases to enrich SMT systems [29] or by using medium sized corpora to induce rules to later be used with unseen sentences[5] [30] [11].

2.2.1 MT with Signed Languages

Sign languages (including BSL and ASL) are visual-spatial by nature, hence there exist no text-based corpora. For this reason, many of the existing approaches have been rule-based rather than statistical, with some exceptions. Here we discuss the most successful approaches at translating into sign languages, their advantages and drawbacks.

[5] https://www.apertium.org

ZARDOZ is a cross-modal translation system based on artificial intelligence using a blackboard control structure, where a common knowledge base is updated by workers as more solutions are found [38]. The solution ultimately being an interlingual representation of the source translation (Figure 2.5). Its main advantage is the ability to translate from English into any sign language, such as Japanese, American and Irish Sign Languages [39]. To animate the output they use a virtual signing doll and calculate signing space to place the hands. However it is missing a sign lexicon i.e. a dictionary of signs, and there is no evaluation of the system in terms of accuracy or quality of the signing.

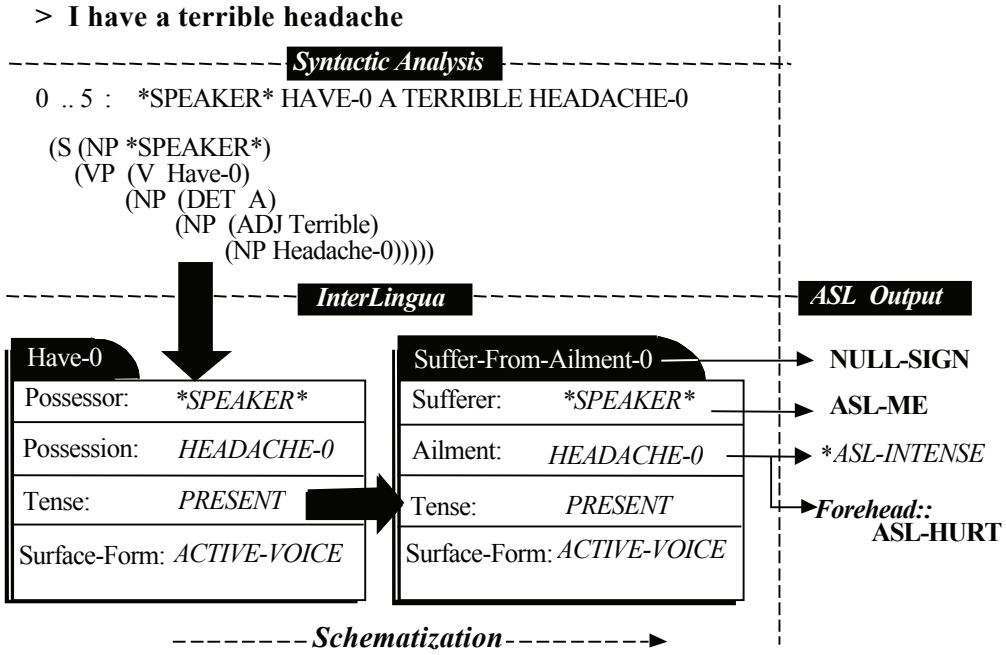


Figure 2.5: Interlingua representation using ZARDOZ; from [38]

ViSiCAST is a transfer based system from English to BSL that uses the HamNoSys phonetic transcription notation [20] [21]; refer back to Table 2.1 for a visual comparison of the different notation schemes. It involves 1) syntactic parsing using the Carnegie Mellon University link grammar parser, 2) semantic transfer i.e. the conversion to the SL structure from written English (here the data resembles an interlingua), and 3) the generation of SiGML[6]. The output is then used to generate the video animation. Each sign stored as SiGML is animated using gesture tracking software for both the face and body movement, making the resulting signing very

[6]Signing Gesture Markup Language, an XML compliant representation of HamNoSys

realistic. While there are no available accuracy results, the system has been tested in face-to-face scenarios in UK post offices with successful outcomes.

TEAM, another rule-based method dating from 2000, is one of the first systems to use an augmented gloss representation of signs to generate its output. Through synchronous tree-adjoining grammars they convert the syntax trees of the source to the target language representation. In addition, embedded parameters help model the non-manual features such as facial expressions and extra-sign modifiers like "Effort" and "Shape", which include speed, weight and spatial information of signs [41].

One of the few purely statistical implementations using solely word alignment techniques is described in [24]. Figure 2.6 shows how word alignment works with a translation from English to SL gloss. The main downside to this approach is that through word alignment there is no transfer of non-manual information, especially since their representation uses basic gloss. In addition the corpus which this system was tested on is restricted and thus does not allow for great flexibility. **MaTrEx** [23] is another data-driven system, used in the development of software to be used in airports. One of the only well documented and properly evaluated implementations of statistical MT with sign languages, it covers multiple language pairs including English to ISL (Irish Sign Language) as well as combinations spoken languages (English, German) to signed languages (German and Irish SL). Its evaluation and performance is reviewed in more detail in Section 5.2 alongside other statistical implementations with insightful results. A particular feature of BSL, and other similarly structure sign languages, is the use classifier predicates. Briefly explained in Section 2.1.1, these help signers construct very complex scenarios and descriptions in sentences and are extremely hard to model in a translation system. The most noteworthy work on the translation of classifiers was carried out by Matt Huenerfauth where interlingual, transfer and direct techniques are combined to create a "multi-path" approach [14][13]. The intermediate representation of a sentence is actually a 3D interpretation of the elements involved in the sentence. For instance the sentence "The car drove down the bumpy road past the cat", involves the initial location of the car and the cat which have to be represented by an appropriate handshape, the articulation of the car movement, as well as the notion that a cat sits on a road path and that the car should move along the same path. While our system is not capable of handling classifier predicates, as will be explained in Chapter 4, it is interesting to mention this work since a simplified version of this approach could be used to implement agreement verbs. This potential approach is further discussed in the future work Chapter 6.3.

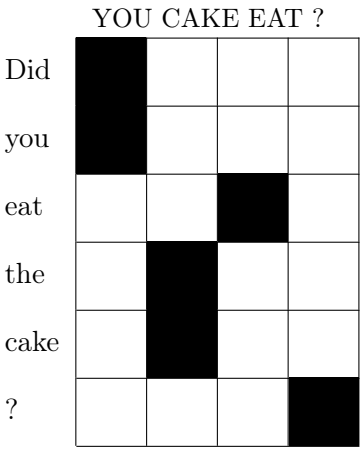


Figure 2.6: Word alignment English-ASL

Ultimately, while all of the above manage to achieve some sort of translation from English to signed languages, none have been developed with portability in mind. The software needs to be either installed on a computer or a server making it only available to a handful of stakeholders. In contrast to the above, an interesting project tackles this by creating a mobile application to translate on the go [4]. Signs are stored in a database and when a request to translate is made, the analysis of the sentence, translation and construction of the video output is done remotely. In addition, a special animation software was made available for users to create their own signs and add them to the dictionary. However this means translation is only available through the app installed on a phone, and the small screen of hand held devices makes it difficult to properly see the animations. Furthermore, only video is returned to the user, meaning there is no freedom of view (such as changing to a first person perspective, like was done in [10]). To our knowledge, there is no cross-platform, online translation tool currently available to translate from English to BSL. Our goal is to create a real-time translation tool available to anyone, anywhere.

2.3 Personal Work

It is necessary to mention that I have already worked on Sign Language in conjunction with Animation. For my undergraduate project I created a system to teach BSL virtually, although it only displayed a set of precomputed animations (animations were simply rendered in real time) and did not include any linguistic analysis nor did it interpolate between signs. The architecture used to build the animation components was not efficient and did not fully exploit the features offered by 3D rendering libraries

such as `OpenGL` (which I will partially use again, through JavaScript). In my previous project, since I wanted to learn more about skeletal animation, I built everything from scratch in Java, something I would not do now considering the online environment and time restrictions imposed by the other aspects of the project. While I have not reused any previous material, I have used the knowledge learnt from it in order to avoid repeating mistakes.

2.4 Summary

In this chapter we presented the basic linguistics concepts of BSL to give an idea of the complex elements to be considered when building a translation system for signed languages. We have also discussed existing approaches to this problem using a range of techniques, their drawbacks and benefits. The most accurate method currently available is Statistical Machine Translation, however the lack of an appropriate corpus requires alternative rationale. The knowledge laid out in this chapter will be used to build an online, cross-platform and portable tool to translate from English to BSL using a corpus-free rule-based method. The design and implementation of this tool and necessary rules to achieve the translation is attentively explained in the following chapters.

Chapter 3

Design

This chapter covers the overall design of the system, from the approach used for the machine translation problem to the interface and interaction with the end user, including the development environment and technology choices for the implementation.

3.1 Language Choices

Nowadays, most web application heavily rely on the use of JavaScript libraries such as `jQuery` which allow for easy interaction with the user interface. We are seeing more and more websites turning into full multi-purpose applications, such as the Google Docs suite which enables complex document editing directly from the cloud. One day we may not even need to install a program or application on our devices, as it will be simply available on a browser. Many of these modern web apps often use 2D and 3D animations for uses ranging from simple aesthetic effects to complex in-browser gaming. Since its release, `WebGL` has been the go-to library for high complexity, cross-browser animation. It is a subset of the high performance graphics library `OpenGL`, used in a very large number of video games as well as applications such as the Adobe suite, multiple computer-aided design software (CAD) and Google Earth. While `WebGL` can be scripted directly to work in browser (`WebGL` is written using the `OpenGL` Shading Language), the best alternative to avoid having to create an animation and rendering engine from scratch is `ThreeJS`[1], a JavaScript API that displays 3D animations. Its skeletal animation module makes it incredibly easy to load in a model with a skeleton of bones and joints, and keyframe animations from popular 3D file formats and render everything into a scene.

[1] `http://threejs.org`

To implement the machine translation module Python was the preferred language choice due to its extensive Natural Language Processing (NLP) library, `nltk`[2] and for its ability to combine OOP and functional programming styles. Furthermore, in the proposed implementation Python works on the server side and performs all machine translation operations to be returned to the browser, thus cutting down the processing workload on the browser. Since data will be travelling across different language modules, it was decided to use the `Flask` framework[3] to enable cross-language communication between the browser and the server. The application is hosted on a free server account of the Heroku cloud platform[4].

3.2 Translation

The translation module in Python was designed following the methods described in the previous work on MT for sign language in Section 2.2.1. Since no corpus was available, it was decided to use a rule-based translation approach using *transfer* and *direct translation* techniques. Figure 3.1 illustrates the translation pipeline working on the server-side.

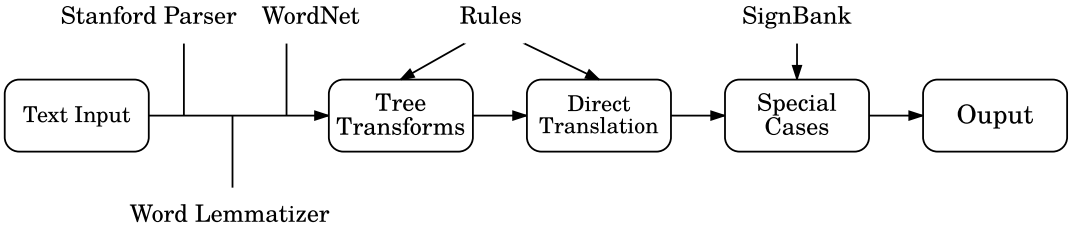


Figure 3.1: Machine Translation module design; the external inputs are obtained either from third party components or preprocessed data from web resources.

The design of the pipeline was heavily inspired by the approach used in [20], while the tree transformation techniques described in [41] is most similar to the one used here. All of the employed methods and algorithms are discussed in detail in Section 4.1. As mentioned in the previous section, Python comes with a large number of available libraries, one of them being `ntlk`. It is used for many tasks of natural language analysis including part of speech (POS) tagging and named entity recognition[5]. In our implementation we use it mainly to:

[2] `http://www.nltk.org`
[3] `http://flask.pocoo.org`
[4] `https://www.heroku.com`
[5] Abbreviated to NER, it is the process of identifying pre-defined named entities in a text such as names of people, places, organisations (e.g. brands), percentages etc.

- Access the WordNet interface for setting the category of each word. For example the noun "man" belongs to the category `noun.person` and the verb "have" to `verb.possession`.
- Employ the provided syntax `Tree` data structure with many useful methods.
- Perform *lemmatisation* over words[6] to retrieve the words roots.

`nltk` also provides a POS tagger, but it is slow and not as accurate when compared to the state-of-the-art Stanford Parser, with approximately 10% difference in accuracy in favour of the latter [36].

Because the system needs to be expandable with time, most of its components, starting with the tree transformation module, are very versatile. Rules are not hard-coded but instead stored in a user-accessible text file that can be progressively modified with time. The way these rules are created and how they affect the translation is explained in Section 4.1.2.

3.3 User Interface

As discussed previously in the Objectives (Section 1.2), the UI should enable users to easily access the controls to insert and translate a sentence of their choosing, whilst allowing continuous visibility of the virtual avatar. The model representing the virtual signer should be made an integrated member of the interface to prevent distractions and potentially degrade usability. The user interface to access the underlying translation system consists of only a single page, with a large view of the virtual agent and floating controls. Figure 3.2 shows a sketch of the interface layout, including desktop and mobile versions since the system is supposed to work on both classes of devices. Figure 3.3 is a screenshot of the final version of the website and when compared to the low fidelity prototype, it is apparent that all elements have been implemented as anticipated.

[6] Lemmatisation is used to "reduce inflectional forms and sometimes derivationally related forms of a word to a common base form" [7]

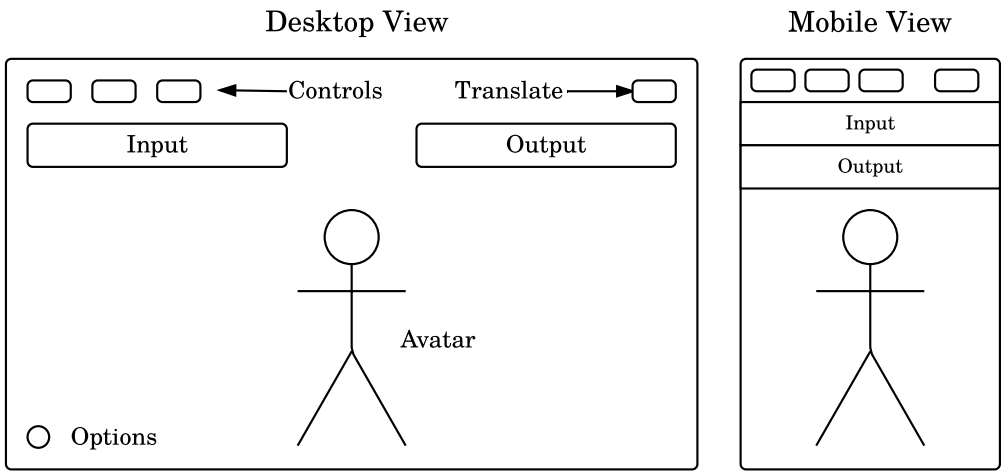


Figure 3.2: UI Low fidelity prototype

Figure 3.3: UI high fidelity prototype

The UI is somewhat similar to the one found on Google Translate, with the input on the left and output on the right, and the familiar "Translate" button coloured in blue (Figure 3.4). Though instead of language choice options we find controls for the camera and playback.

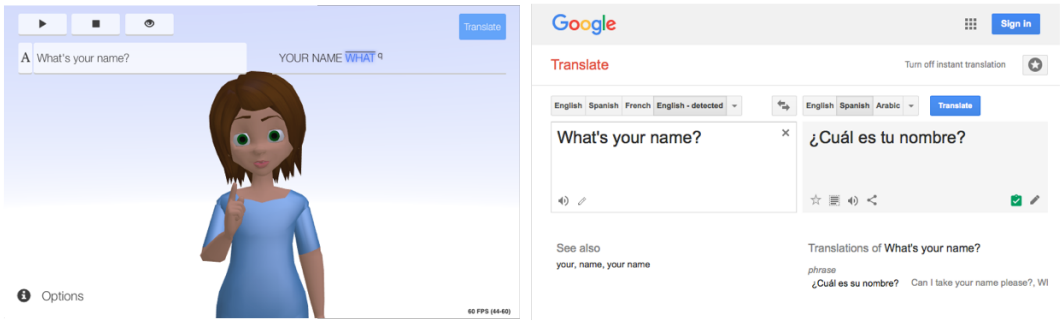


Figure 3.4: UI high fidelity prototype compared to Google Translate

At the bottom of the page are the info and options buttons that create an overlay on top of the current view (Figure 3.5):

- Clicking the info button supplies general information to the user including a disclaimer about translation accuracy as well as currently available signs. In addition, a detailed explanation of the sign symbols and non-manual features displayed during playback is also provided.
- Options include playback speed, automatic camera angle and the ability to hide or show non-manual features.

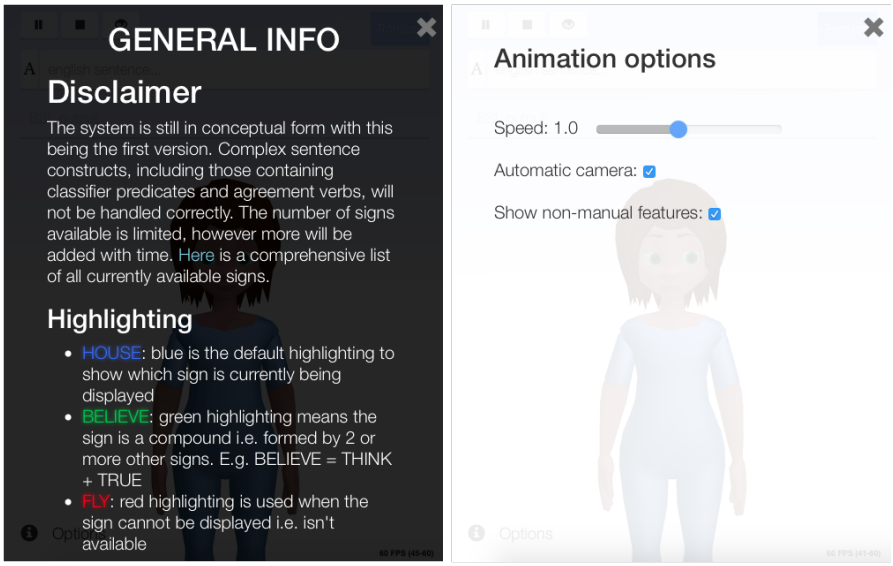


Figure 3.5: Overlay with additional information and options

Playback speed can be decreased by users who would like to follow the gestures more closely, or increased for those who prefer a faster signing style. Non-manual features such as “q” and “neg” are shown in the gloss output on the right hand-side, however by

enabling the "Show non-manual features" in the options, *all* features are shown: this includes facial expression for setting the time, head nods for indicating the subject etc. which are normally not included in the gloss. See Figure C.1 in the appendix for screenshots of the app when displaying non-manual features.

3.3.1 User Interaction

Figure 3.6 illustrates the main actions available to the user and how they affect the application; the "Process Data" state is a separate set of events that happen on the server side. From the controls above the input it is possible to pause the animation currently playing (even when in idle state), play/unpause, and change camera angle. The button press switches between the default free-movement view to first person; the change can be set to *automatic* in the options menu for when fingerspelling[7]. See Figures C.2 and C.3 in the appendix for a comparison between first person and 3rd person free movement view. The translate button is not available to click until a sentence has been entered; this is to prevent potential errors if trying to translate an empty string. After pressing "Translate" a spinning wheel is shown to alert the user that processing is currently happening. Subsequently the output from the machine translation process is returned and displayed as animations. While these are playing the user still has access to the play/pause and view change controls as well as the ability to interrupt the animation and return to idle.

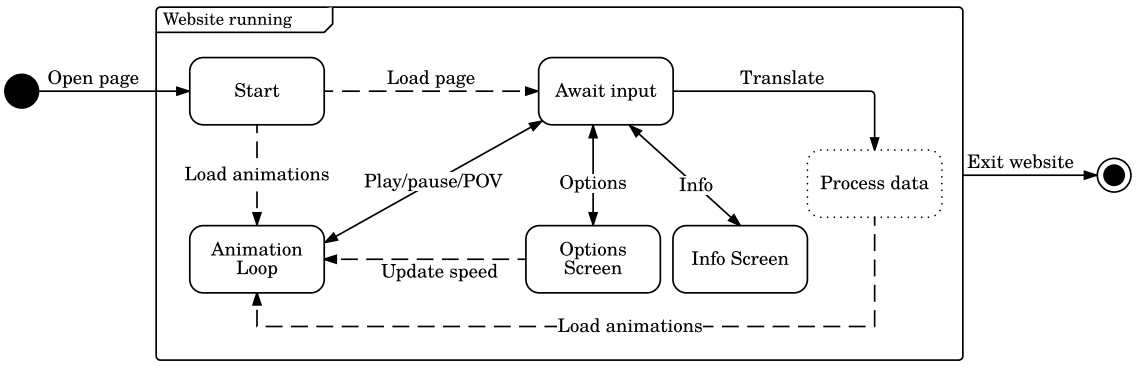


Figure 3.6: User interaction flowchart. Dashed lines show system events and solid lines user actions.

The output is also shown as gloss notation on the right hand side, similarly to what someone would find on the aforementioned Google Translate. To increase the understanding of what is being signed, each gloss word lights up in blue when it

[7]Most often than not the alphabet is illustrated as seen from the signer and not from a frontal view.

is currently being signed by the avatar, similarly to karaoke where the lyrics follow the song with a moving symbol or by changing color. As described in the general information page (Figure 3.5), blue highlighting is the default colour for the current sign, green means the sign is a compound (Section 2.1.2) and red means the sign is not yet available in the dictionary of signs and thus cannot be displayed by the avatar. In addition to the above features, users can also have sentence suggestions to get started with translating by clicking on the "A" to the left of the input box (Figure 3.7).

Figure 3.7: Sentence suggestions auto-fill

3.4 Summary

This chapter described the user interface design for the web application which supports all of the functionalities listed in Section 1.2. The interface was kept simple and similar to existing online translation websites, so as to make user interaction intuitive and fun. Reasons for specific language choices and development environments were made clear, and an overview of the translation pipeline was also given. The specifics of the system implementation will be discussed next and all decisions in the following chapter have been made with the design specifications and previously stated requirements in mind.

Chapter 4

Methods and Implementation

This chapter will detail the development and implementation methods of the aforementioned system, taking into account BSL translation requirements and previous work on this topic. There are two main sections: the translation module in Python where techniques from Section 2.2.1 are employed, and the animation in `TreeJS` which covers skeletal animation in Blender, the JSON file format, 3D scene components, and algorithms to play said files. Finally a short explanation of the communication between the two essential modules is found at the end of this chapter.

4.1 Rule-Based MT in Python

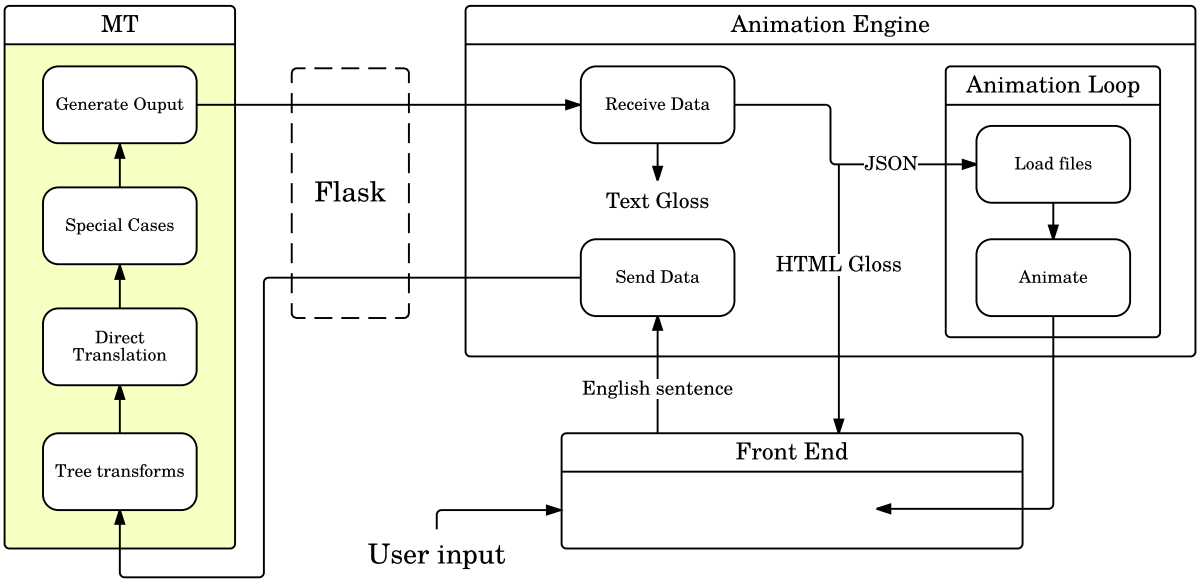


Figure 4.1: System outline: Machine Translation highlighted

Figure 4.1 highlights the components that will be discussed in the following pages.

Before starting a detailed discussion of the implementation of the translation module, let us quickly summarise the translation process from start to end. Keep in mind that the translation begins with a request received on the server side as a result of the user pressing the "Translate" button on the browser, and ends by the server sending the output back to the browser, which is then displayed on screen. At the very beginning, a sentence in English text is sent to the server and information is extracted using multiple resources including the Stanford Parser and WordNet. This analysis step is necessary in all rule-based methods as it is through this information that the system can perform the transformations. The sentence is then set to an intermediate state through transfer. Finally, the data is made into a BSL sentence by embedding non-manual features (discussed in the Background Section 2.1.6). From this representation the output is obtained and sent to the browser.

The Stanford Parser plays a crucial role in the pipeline and it is in fact the initial analysis of the sentence that captures the most important information. While we will not be discussing the internal workings of the parser, it was necessary to develop special code to obtain the resulting data. In our implementation, when the web-app is launched a dummy request is sent to the online version of the parser[1]. If a positive response is received i.e. if the website is online, all following sentences are sent to it directly. Otherwise a local instance of the parser is created and all analysis is performed on the server. This fallback exists because the Stanford website may be offline at times due to multiple reasons, and complete reliance on an external tool such as this one could prove problematic should it be made entirely unavailable. As such, a `Parser` object is created in Python, which depending on the version created (online or local) executes different code to retrieve the data. Details of the data retrieval from both of the parser versions is found in Appendix B.1.

[1]Available at this address `http://nlp.stanford.edu:8080/parser/index.jsp`

4.1.1 Information Extraction

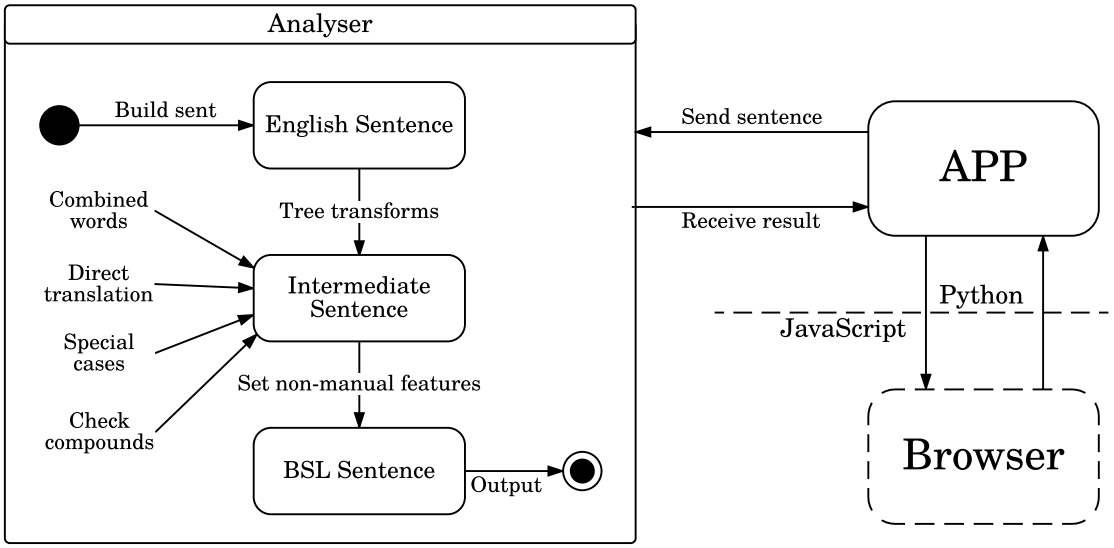


Figure 4.2: Python-side, object oriented overview of the pipeline with transition between sentence representations

Please refer to Figure 4.2 for a detailed representation of the highlighted part of Figure 4.1, an overview of the translation process. The sentence is first passed through the Stanford Parser which returns the following results:

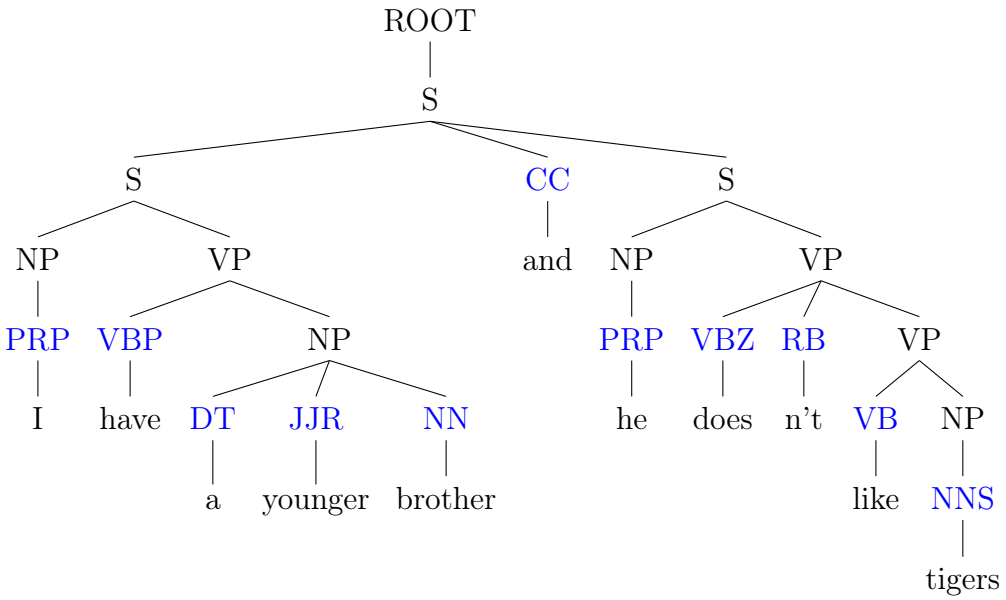
1. The sentence with labelled Part of Speech tags
2. A syntax tree structure
3. The sentence dependencies graph

The sentence entered must be in correct English i.e. all punctuation must be present, in order to be labelled correctly by the parser. Punctuation is then removed from the syntax tree. For instance if we wanted to parse the sentence "I have a younger brother and he doesn't like tigers.", then the three output components of the parser would be

1. Tagging

I/PRP have/VBP a/DT younger/JJR brother/NN and/CC he/PRP does/VBZ n't/RB like/VB tigers/NNS ./.

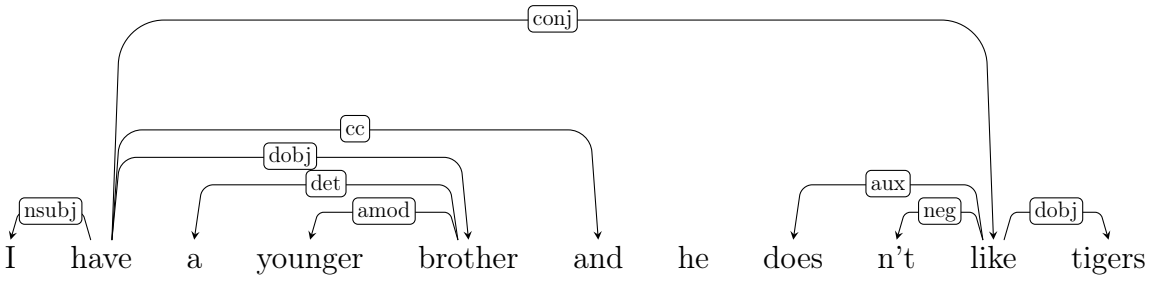
2. Parse



3. Dependencies

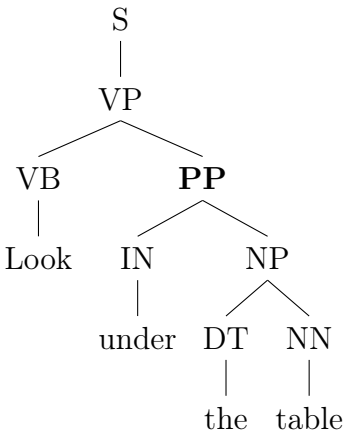
nsubj(have-2, I-1)	*root*(ROOT-0, have-2)
det(brother-5, a-3)	*amod*(brother-5, younger-4)
dobj(have-2, brother-5)	*cc*(have-2, and-6)
aux(like-10, does-8)	*neg*(like-10, n't-9)
conj(have-2, like-10)	*dobj*(like-10, tigers-11)

The above can be graphically represented as

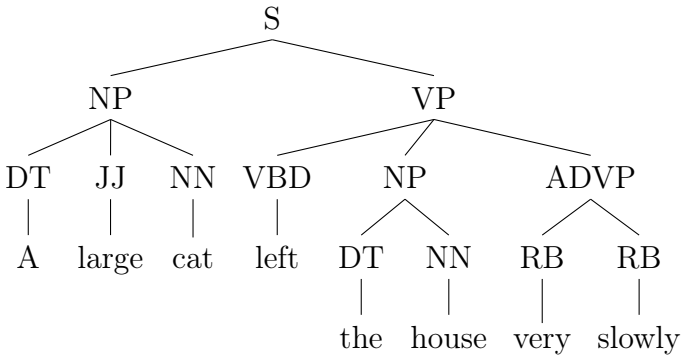


Dependencies are *grammatical relations* between words. For the example above, we see that "I" is the subject of the verb "have", and "his" is a noun possessive modifier of "name". The Stanford dependencies manual [9] details every dependency used in the Stanford Parser, and all tags used in the Penn TreeBank are found at `http://web.mit.edu/6.863/www/PennTreebankTags.html`. There are more than 100 part of speech tags, however to familiarise the reader we list a few of the most common ones:

- DT: determiners such as *a*, *the*, *some* etc.
- NN: the base tag form for **nouns**; this class includes subcategories NNS (plural nouns), NNP (proper nouns) and NNPS (plural proper nouns)
- VB: base tag form for **verbs**; includes VBD (past tense), VBN (past participle), VBZ (3rd person singular present) and more
- JJ: base tag for **adjectives**; includes JJR (comparative) and JJS (superlative)
- PRP$: possessive pronouns, such as *his*, *their* etc.
- NP: noun phrase tag; a noun phrase is usually composed of a noun and a number of other elements but **not** a verb. For example "The small mouse" is a noun phrase containing a determiner (DT) "The", an adjective (JJ) "small" and noun (NN) "mouse"
- VP: the verb phrase; it always contains a verb but may also have other tags as we will show shortly
- PP: prepositional phrase; begins with a preposition (IN tag), for example



- ADJP: adjective phrase; it is formed by an adjective and particles (RB) such as "very"
- ADVP: adverbial phrase; analogously to ADJP it contains an adverb and possibly other tags. The following example shows a sentence consisting of a noun phrase and a verb phrase, with the latter also containing an adverbial phrase



- SQ: inverted yes/no questions e.g. "Do you like pizza?" or the main clause of a *wh*-question e.g. "Which one **do you want?**"
- SBARQ: direct questions, for example "Which one do you want?"

The data obtained from the parser, that is, the three components listed at the beginning of this section (tags, parse and dependencies) are used to create `Word` objects storing the text word, root, POS tag, position in the sentence, role (dependency) as well as what sentence and dependency group the word belongs to. These `Word` objects are our own representation of an English word. The root form of each word is obtained using `nltk`'s word Lemmatizer, which, depending on the POS tag of the word, extracts the basic form of the word. For example the verb "eating" has root "eat" and the noun "children" has root "child". The word category is extracted from WordNet synsets, essentially sets of synonyms where words are interlinked by lexical relations. The lexical category is what our notion of word category corresponds to.

i	Word	POS	S-gp	WN Category	Root	dep	d-gp	target	neg
1	i	PRP	S_1	undefined	i	nsubj		have	False
2	have	VBP	S_1	verb.possession	have	root		tiger	False
3	a	DT	S_1	undefined	a	det		brother	False
4	younger	JJR	S_1	adj.all	young	amod	1	brother	False
5	brother	NN	S_1	noun.person	brother	dobj	1	have	False
6	and	CC	S	undefined	and	cc		have	False
7	he	PRP	S_2	undefined	he	nsubj		like	False
8	does	VBZ	S_2	verb.social	do	aux		like	False
9	n't	RB	S_2	undefined	not	neg		like	True
10	like	VB	S_2	verb.emotion	like	root		tiger	True
11	tigers	NNS	S_2	noun.person	tiger	dobj		like	False

Table 4.1: `Word` objects information

Table 4.1 omits the `num_modified` field that indicates whether a word is modified by a number, such as in "the two *boys*" where "two" modifies "boys". Each `Word` object

is stored in an `EnglishSentence` object in a list. The sentence object also contains information about sentence groups (S-gp in Table 4.1) with their associated tenses like present, past and future. The syntax tree for the sentence is then processed to replace each String representation of the words with actual `Word` objects; this way we can directly alter the structure of the sentence whilst retaining all syntactic information.

4.1.2 Translation Rules

In order to make the system flexible and expandable without having to modify the source code, all parts of the translation module, with the exception of the `Special Cases`, use external user-defined rules to perform the translation. Taking inspiration in the Apertium system (mentioned in Section 2.2), rules can be defined following a standardised format which maps from the source to the target. For the tree transformation rules, we use the Context Free Grammar (CFG) representation of the trees to define source and target mappings. The CFG is a collection of *productions*. The mappings are written following the CFG productions syntax, where the right arrow means the left side symbol *generates* the symbols on the right side. For this reason the symbol used to define the transformation is a vertical line | instead of an arrow →. Listing 4.1 is a snippet of the tree transformation rules used in the application. The full list of rules can be found in Appendix A.

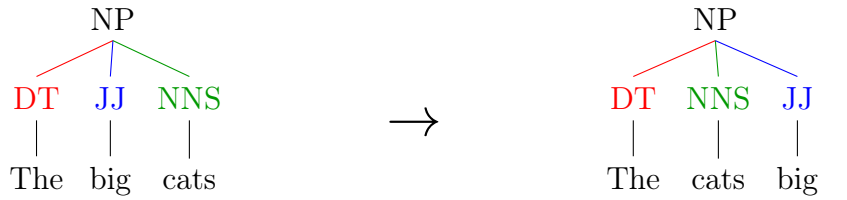
```
NP -> JJ NN~ | NP -> NN~ JJ
NP -> <> JJ NN~ | NP -> <> NN~ JJ
NP -> <> ADJP NN~ | NP -> <> NN~ ADJP

// will cover anything like "has died", "was born" i.e. auxiliaries
VP -> VB~ <> VP | VP -> _ <> VP
// covers "He was sick", "He is tall" etc
VP -> VB~ ADJP <> | VP -> _ ADJP <>

SQ -> VB~ <> | SQ -> _ <>  // removes the "have", "did" etc in questions
SQ -> MD <> | SQ -> _ <>
// handles direct questions introduced by a wh- word or wh-phrase
SBARQ -> WH~ <> | SBARQ -> <> WH~
```

Listing 4.1: Tree Transform rules snippet

In the first three lines are some examples of rules that handle noun phrase constructions. The first two are specific cases of sentences where there an adjective is followed by a noun. Let's take `NP -> <> JJ NN~ | NP -> <> NN~ JJ` as an example. The symbol ~ indicates that any tag prefixed by the preceding text can be matched; with NN this includes NNS, NNP, NNPS as shown in the previous section. A noun phrase such as "The big cats" can be represented by the tree shown in Figure 4.3a. The

diamond ◇ (or <> characters in the ASCII format) means any kind of tag(s) will be matched. In this example, ◇ will be matched with *DT*, a determiner. Figure 4.3b shows the result of applying the given rule to the original tree. We can highlight the tags that match the tree nodes to make the process more obvious:

$$NP \rightarrow \diamond\ JJ\ NN\sim \quad | \quad NP \rightarrow \diamond\ NN\sim\ JJ$$

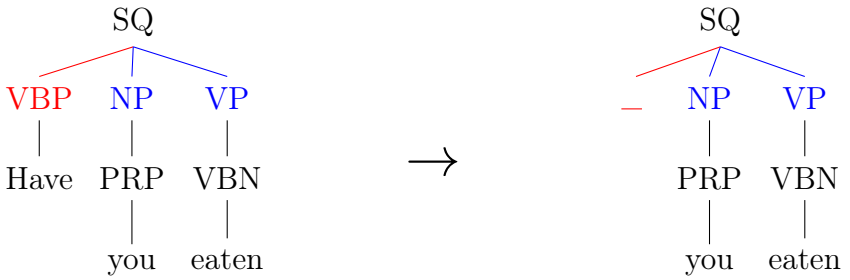


(a) Original syntax tree (b) Transformed tree

Figure 4.3: Tree transformation using rules: noun phrase

The rule on line 3 acts analogously but with an adjective phrase instead of just an adjective. As was explained earlier, an adjective phrase ADJP could be formed by an adverb + adjective such as "very big". Let us illustrate another example from snippet 4.1, namely `SQ -> VB~ <> | SQ -> _ <>` form line 10, which specifically handles the removal of the auxiliary verb at the beginning of a question. The transformation process is schematised in Figure 4.4.

$$SQ \rightarrow VB\sim\ \diamond \quad | \quad SQ \rightarrow \_\ \diamond$$



(a) (b)

Figure 4.4: Tree transformation using rules: question

In the above, the underscore symbol _ means that any tag in the source (the production on the left of |) at the same position as _ in the target (on the right) will be removed. Note that more than 1 rule can be applied to a specific sentence. If we

combine the examples from Figure 4.3 and 4.4 we obtain the sentence "Have the big cats eaten?", which would use both rules to transform the tree as shown in Figure 4.5.

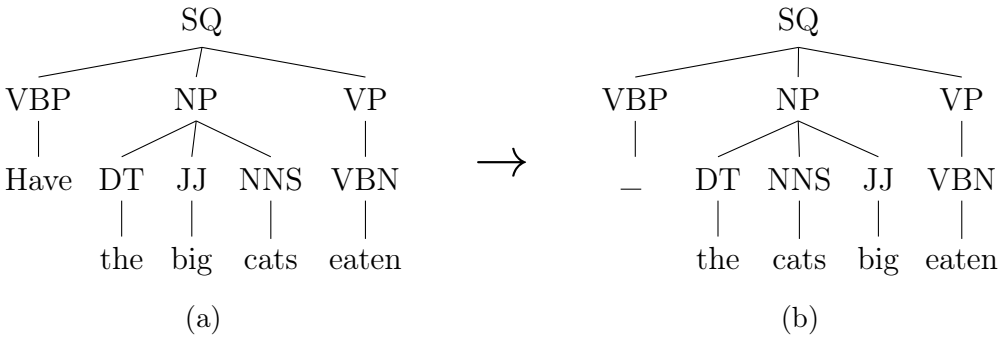


Figure 4.5: Combined example of tree transformation

All of the rules found in the file are created to bring the English sentence syntax tree to a form that most closely resembles that of a BSL sentence. For example knowing that in BSL an **attributive** adjective (see Section 2.2) comes before the noun in order to differentiate it from a **predicative** adjective we may want to specify mapping (1). Recall that the $\sim$ means any subclass of the tag will be matched. Similarly if we know that a verb phrase begins with *any* verb (as implied by $\sim$) and ends with another verb phrase, then the first verb can be discarded with the _ symbol (2). This rule is used for sentence constructs such as "I have eaten", corresponding to ME EAT in BSL. Tree transformation rules are written in a descending order according to their complexity to ensure a correct match going from very specific to more generic cases.

$$NP \rightarrow JJ\ NN\sim \quad | \quad NP \rightarrow NN\sim\ JJ \tag{1}$$

$$VP \rightarrow VB\sim \diamond\ VP \quad | \quad VP \rightarrow \_ \diamond\ VP \tag{2}$$

Rules for direct translation are similar and are divided in 3 subsets:

- Swap: in this category are keywords in a sentence around which words revolve. Strictly speaking, any words that appear *before* the given keyword are placed behind it, and any that appear *after* are placed in front. The keyword itself is then removed from the sentence. For now the only keyword in this category is "if". A simple example would be

$$\text{Tell me if you're okay} = \overline{\text{YOU OKAY}}^{\text{q}}\ \text{TELL ME} \tag{$*$}$$

It is apparent that the words in front of *if* are found at the back of the BSL translation, while the ones following it are at the front and signed with a question facial expression. This is due to rhetoric questions as explained in Section 2.1.4.

- Multiple words: these are mappings for words that appear together. For example the sequence "...25 years old" matches the rule *year old* → _ *age*. Again, the _ symbol implies deletion.

- The rest of the direct mappings are categorised by POS tag. Examples of personal pronouns rules are *us* → *we* and *he* → *ix* where *ix* stands for the Index notation as explained in Section 2.1.1. Here, nouns are also mapped for fingerspelling such as *parents* → *-m-m-*.

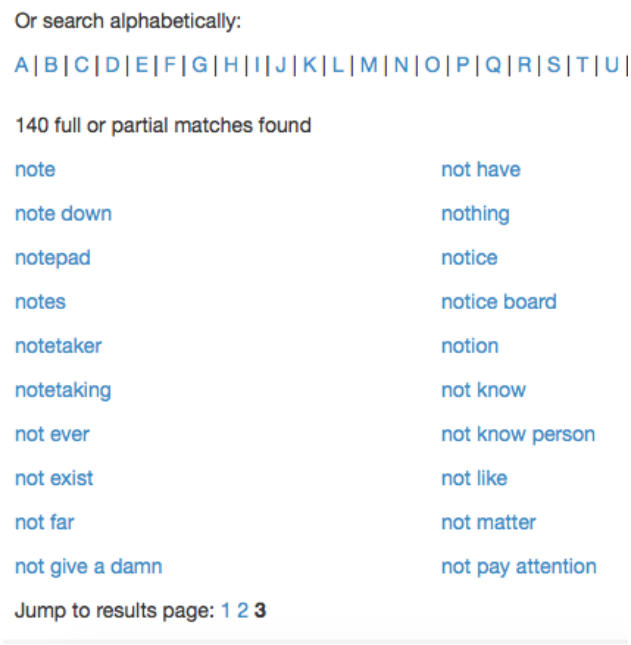
Listing 4.2 shows a snippet of the direct translation rules file; again, the full file is found in Appendix A. The syntax used here is different: the right arrow → defines the mapping i.e. anything on the left maps to the elements on the right, however deletion symbol _ keeps the same function.

```
SWAP
if -> if // moves the sentence after the if before the cause
WORDS
there be -> _ _          // removes the existential there + is
if it -> _ _             // removes the it (SBAR)
have you -> _ you        // remove any 'have you' questions
...
PRP
i->me
he->ix                   // converts personal pronouns
him->ix                  // to index pointers
she->ix
her->ix
it->_
us->we
they->them
MD
will->_
NNS
parent->m-f-             // converts word to fingerspelling
NN
mom->m-m-
...
```

Listing 4.2: Direct translation rules snippet

In addition to the above there are also two files that complement the translation rules by carrying extra information. The first contains data obtained from the SignBank

website[2] using a simple HTML web scraper. Figure 4.6a shows the website data and 4.6b the formatted result. The Python code for the scraper can be found in Appendix B.2. Since each word and word sequence on the website corresponds to a single sign in BSL, we use this data to separate word sequences and match them in the input text. Any sentence that includes a sequence of words found in the `signbank_multi.txt` file will see those words combined in a single sign. In 4.6b, the word "notice board" would be signed as one unit and thus become NOTICE-BOARD in BSL.



(a)

```
about time
above all
...
note down
not ever
not exist
not far
not give a damn
not have
notice board
not know
not know person
not like
not matter
not pay attention
...
worn out
would not
year before
```

(b)

Figure 4.6: SignBank website data and formatted result

The second file contains *compound* morphemes. Recall from Section 2.1.2 that a compound morpheme is a sign made up of 2 or more other free morpheme signs. There are only a couple of these available at the time of writing this report and Listing 4.3 shows the contents of this file. The exact usage of this data is explained in more detail in Section 4.1.3.3.

```
blood->red flow
believe->think true
check->see maybe
promise->say true
tiger->zebra animal
```

Listing 4.3: `compound_morphemes.txt` file

[2] `http://bslsignbank.ucl.ac.uk`

4.1.3 System Components

4.1.3.1 Syntax Tree Transforms

As was discussed in the Background Section 2.2.1, the TEAM project uses adjoining tree grammars to convert the English sentences into an intermediate representation from which the ASL translation is obtained. We use a similar approach but instead of an interlingua, we keep the sentence words objects as they are, and after changing the tree structure we obtain our definition of intermediate representation.

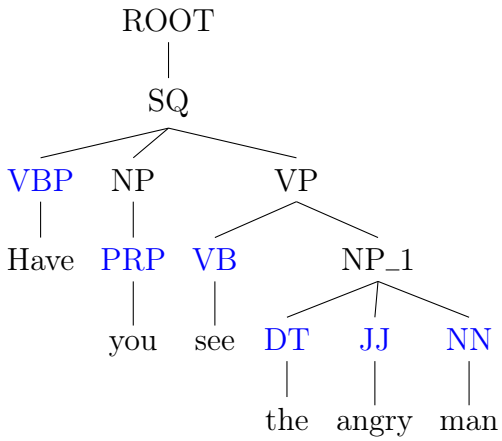
Algorithm 4.1 Applying tree transformation rules to the source sentence

Require: transform_rules is a list of Mapping objects with source and target

```
1:  function ApplyTreeTransforms(sentence)
2:      new_productions ← [ ]
3:      productions ← sentence.getProductions
4:
5:      for each prod  in productions
6:          current_prod ← prod
7:
8:          for each mapping  in transform_rules
9:              source ← mapping.Rebuild
10:             match ← Match(source, current_prod)
11:
12:             if match exists then
13:                 source_i ← makeSourceGroupIndexes
14:                 target_i ← linkSourceTarget(source, target, source_i)
15:                 new_target ← constructTarget(target, match, target_i)
16:                 modified ← true
17:                 current_prod ← new_target              ▷ Update the production
18:         end
19:         if modified is true then
20:             prod_obj ← constructProduction(prod_string)
21:             new_productions ← new_productions +prod_obj
22:         else
23:             new_productions ← new_productions + prod
24:     end
25:     grammar ← CFG(new_productions)
26:     return generate(grammar)
```

Let us go through a step-by-step run of this fundamental algorithm in the pipeline; line references are from Algorithm 4.1. Assume we would like to transform the following sentence: "Have you seen the angry man?", corresponding to $\overline{\text{YOU SEE MAN ANGRY}}^{\text{Q}}$.

Before the function is called, all POS tags in the tree are made unique by adding indices where necessary as this allows us to regenerate the new syntax tree without ambiguities, that is, the CFG will only have one solution. Thus, the syntax tree structure with root word representations would look like this



We obtain the productions from the current syntax tree (line 3), and for the above example we get the following CFG productions composed of Terminals (words) and Nonterminals (tags):

$$
\begin{aligned}
&ROOT \rightarrow SQ && SQ \rightarrow VBP \quad NP \quad VP \\
&VBP \rightarrow have && NP \rightarrow PRP \\
&PRP \rightarrow you && VP \rightarrow VBN \quad NP\_1 \\
&VBN \rightarrow see && NP\_1 \rightarrow DT \quad JJ \quad NN \\
&DT \rightarrow the && JJ \rightarrow angry \\
&NN \rightarrow man
\end{aligned}
$$

While iterating through each of the above productions we look for a rule in the list of mappings (transform_rules) on line 8. For demonstrative purposes assume production (3) and mapping rule (4) exist.

$$\text{(production)} \quad VP \rightarrow VB \quad VP\_2 \quad ADJP \quad NNS\_3 \tag{3}$$

$$\text{(source)} \quad VP \rightarrow VB \diamond NN\sim \quad | \quad VP \rightarrow \_ \quad NN\sim \diamond \quad \text{(target)} \tag{4}$$

Rule (2) essentially removes the verb and pushes the ending noun to the front of the verb phrase keeping everything else in between. We would then expect the production to become $VP \rightarrow NNS\_3 \ VP\_2 \ ADJP$. Since we need to perform a string match

between the existing source and source stored in the rules, we rebuild the given source with `regex` syntax (line 9). Thus the above source becomes

$$VP(\_?\backslash d?) \rightarrow VB(\_?\backslash d?)(.*)NN([A-Z]*)(\_?\backslash d?)\$$$

The above symbols replace the simple text characters used when writing the rules. Each character has a specific meaning in `regex`. To begin with, $VP(\_?\backslash d?)$ will match any string that begins with VP **possibly** followed by an underscore _ and a digit d. This in fact corresponds to the desired effect of having a tag potentially followed by a unique ID, as is the case for the *angry man* example. Then $\diamond$ is replaced by $(.*)$ which will match **any** string. Finally, the sequence $NN([A-Z]*)(\_?\backslash d?)\$$ means that any string beginning with NN and possibly followed by any number of *letters* and/or followed by an underscore plus a digit will be matched. Any of the following examples will produce a match for the `regex` rule just described: NN, NN_1, $NNPS$, $NNPS\_2$ etc. The match is then obtained and if successful, generates a group of matches between the two sources. For the production (3), the groups would be (‘ ’,‘ ’,‘ $VP\_2 \quad ADJP$ ’, ‘ S ’, ‘ $\_3$ ’) meaning that VP and VB yielded an exact match (empty group), the $\diamond$ tag matched to $VP\_2 \quad ADJP$ and NN matched to the tuple (S, $\_3$). Afterwards, *Source Match Indexes* (SMI) are created. SMI can be defined as follows: after a match, each element in the source yields match groups specifying the “differences” between the original source element and given production. Any group created is assigned an index starting from 1. In addition, any element from the source that is marked with a $\sim$ is linked to a pair of indexes instead of just one since the match can happen for both the subcategory and the unique tag ID. For the current example we find SMI $= [1, 2, 3, (4, 5)]$. Figure 4.7 schematises the above explanation.

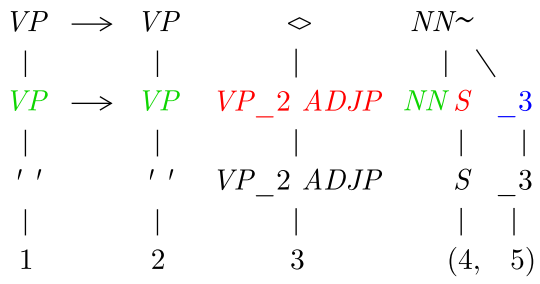


Figure 4.7: Source Match Indexes derived from the match groups

Source and target elements are linked by rearranging the indexes according to the target (line 14). Table 4.2 shows a run of the linking algorithm for the current example. The index is increased by 1 at each iteration and actually represents the

Target	Source	Index	Backtrace	TMI	Source Chunks
VP	*VP*	0	0	[ ]	[*VP*, →, *VB*,◇, *NN*∼]
_	*VB*	1	0	[1]	[→, *VB*,◇, *NN*∼]
NN∼	◇	2	0	[1, 2]	[◇,*NN*∼]
NN∼	*NN*∼	3	1	[1, 2]	[◇, *NN*∼]
_	_	2	0	[1, 2, (4, 5)]	[◇]
				[1, 2, (4, 5), 3]	[ ]

Table 4.2: Run of the LinkSourceTarget algorithm. TMI stands for Target Match Indexes.

index of the element in the SMI. On the first step *VP* matches *VP*, so we fetch element at position 0 from SMI, 1. Then _ matches *VB* (we assume that when _ is found, the first element from the source is used) so we fetch element at position 1 from SMI, 2. Then *NN*∼ does not match ◇, thus we set the backtrace. Backtrace is used to reset the index once the element from the target is found in the source. In fact *NN*∼ is found further on, so in the step following we match *NN*∼ and fetch element at position 3 from SMI, $(4, 5)$. After resetting the index (index - backtrace - 1) we match ◇ to ◇ and fetch 3 from SMI at position 2, yielding TMI $= [1, 2, (4, 5), 3]$. We then construct the new target by combining the original target (read from the file) to the match groups using the TMI positions. For each tag in the target we find the group of characters at the position from the TMI and append it to said tag (line 15). Recall that the target for this example is *VP* → _ *NN*∼ ◇.

Production	Target	Value in TMI	Match Group				
			1	2	3	4	5
[ ]	*VP*	1	[‘ ’	‘ ’	‘ *VP*_2 *ADJP* ’	‘ *S* ’	‘ _2 ’]
[*VP*]	_	2	[‘ ’	‘ ’	‘ *VP*_2 *ADJP* ’	‘ *S* ’	‘ _2 ’]
[*VP*, →]	*NN*∼	(4,5)	[‘ ’	‘ ’	‘ *VP*_2 *ADJP* ’	‘ *S* ’	‘ _2 ’]
[*VP*, →, *NNS*_3]	◇	3	[‘ ’	‘ ’	‘ *VP*_2 *ADJP* ’	‘ *S* ’	‘ _2 ’]

[*VP*, →, *NNS*_3, *VP*_2 *ADJP*]

Table 4.3: Reconstruction of the target. Yellow highlights the current selection and green those already selected.

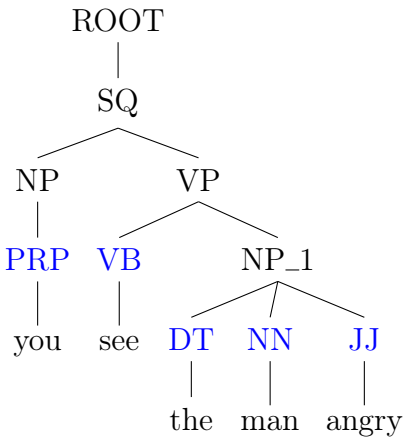
From the run in Table 4.3 we find that *VP* is linked to character group 1, however the group is empty, so nothing is appended. Then _ is matched to nothing, so we don't insert it in the new target: in fact _ means we do not want the Nonterminal to be included in the new production. Then *NN*∼ is linked to the tuple (4,5) so we fetch elements from the groups *S* and _3, which are both appended to *NN* yielding

*NNS*_3. Finally $\diamond$ is linked to group 3 containing *VP*_2 *ADJP*, which replaces it. The final production is *VP* → *NNS*_3 *VP*_2 *ADJP*, as was expected.

Once all productions have been exhausted and updated we generate a new CFG to build the new transformed syntax tree (line 26). Matching rules for "Have you seen the angry man?" are

$$SQ \rightarrow VBP\ NP\ VP \quad \text{matches} \quad SQ \rightarrow VB\sim\ \diamond \quad | \quad SQ \rightarrow \_\ \diamond$$
$$NP\_1 \rightarrow DT\ JJ\ NN \quad \text{matches} \quad NP \rightarrow DT\ \diamond\ NN \quad | \quad NP \rightarrow\ DT\ NN \diamond$$

And the new syntax tree for the intermediate representation is



4.1.3.2 Direct Translation

Before applying the direct translation rules as described in Section 4.1.2, we use SignBank's post-processed data to identify words that may be combined to form a single sign. Note that these are **not** *compound morphemes* but simply expressions that use multiple words in English, however only require 1 sign in BSL. For example the sentence

$$\text{He passed out} = \text{Index}_1\ \textsc{pass-out}$$

While there are signs for both PASS and OUT , they do not simply combine, instead a completely different sign is used (see Section 2.1.4). Refer back to Figure 4.6 for a visualisation of the web data before and after being processed. Words are replaced one by one until exhaustion, the tree representation is dropped and the list of words as a sequence is used instead. At this point we are using an `IntermediateSentence` object, which contains said list of words and multiple methods to further modify the data representation. It is important to note that any modification to the word is recorded by its root; the original word representation is only kept as a reference.

4.1.3.3 Special Cases

Special cases handle transformations that cannot be applied by handwritten external rules. Though it is possible to model them similarly to the tree and direct transformation rules, due to time limitations these are performed through hard-coded rules. The sentence is essentially iterated word by word and if a rule matches the condition it is modified. Rules include and are not limited to:

- The replacement of the "in" and "at" prepositions with "where" to change the sentence into a rhetoric question e.g. "I live in Spain" = ME LIVE $\overline{\text{WHERE}}^{\text{q}}$ -S-P-A-I-N-
- The deletion of the word "that" if found in a clause introduced by a subordinating conjunction (SBAR sentence group) e.g. "You think that I'm sad?" = $\overline{\text{YOU THINK ME SAD}}^{\text{q}}$
- The creation of a personal pronoun Index whenever a proper noun is the subject of the sentence e.g. "Mary is my sister" = -M-A-R-Y- Index_1 MY SISTER
- The shifting of time-related words from the `noun.time` WordNet category towards the front of the sentence to mark the temporal topic (see Section 2.1.3.1)

After applying the special cases there are a few more operations required before generating the output. Firstly, whenever an Index or possessive pronoun is inserted as `IX` and `POSS` respectively, in order to differentiate between multiple actors in the sentence, each is given a unique identifier; in previous examples this was represented as Index, followed by a subscript digit e.g. Index_2. This problem falls into a separate linguistics field called *coreference resolution* and due to the time restrictions a more primitive method was used to solve it. Essentially, because `IX` is inserted when a proper noun is found, through the gender of the name and the dependency of the word we can assign the correct ID to each actor. For example

Jane loved the letter Mike gave her =

-J-A-N-E- IX_1 LOVE LETTER -M-I-K-E- IX_2 GIVE IX_1 $(*)$

The analysis starts with 3 "empty" indices IX and using the information from the name Jane and the preposition "her" we can link them together, assigning the same index to each: IX_1. The remaining actor in the sentence is Mike which, being a different gender, is assigned another index, IX_2.

Secondly we check for compounds using a separate file where each sign (in BSL form) maps to two or more other BSL signs to form compounds. This is done separately from direct translation because the gloss for compound morphemes still represents the original idea: if PROMISE is a compound formed by the signs SAY and TRUE we still gloss PROMISE but sign the other two (see Section 2.1.2 for details). Any `Word` object that is a compound is assigned the free morphemes that form it to mark it as such. Taking the example from Table 4.1, at this point in the processing timeline the `Word` objects stored in the `IntermediateSentence` would look like what is depicted in Table 4.4. Recall that we are interested in the root of each word and not the word values themselves.

i	Word	POS	S-group	**Root**	dependency	target	neg	compound
0	i	PRP	S_1	me	nsubj	have	False	
1	have	VBP	S_1	have	root	tiger	False	
2	brother	NN	S_1	brother	dobj	have	False	
3	younger	JJR	S_1	young	amod	brother	False	
4	he	PRP	S_2	ix_1	nsubj	like	False	
5	n't	RB	S_2	not-like	neg	like	True	
6	tigers	NNS	S_2	tiger	dobj	like	False	zebra animal

Table 4.4: `Word` objects after transformation

4.1.3.4 The Container object

A very important characteristic of BSL is the parallel signing of manual and non-manual features. Not only can facial expressions and eye gaze change the meaning of a word (Section 2.1.6.3), but they can also be used to set the time (Section 2.1.3.1) and differentiate elements in the sentence (Section 2.1.6.2). At any given time during signing, a manual sign may be accompanied by 0 or more non-manual features. The following sentence exemplifies this concept:

I'm not going to the beach today because it was raining (a)

$= \text{TODAY ME } \overset{\text{neg}}{\overline{\text{NOT GO}}} \text{ BEACH } \overset{\text{q}}{\overline{\text{WHY}}} \text{ RAIN}$ (∗)

We can see that a question is being asked with WHY and a negation on the action NOT GO. However, there are more elements that the gloss cannot represent. Let us imagine each non-manual feature as a container of one or more words. Each container may also be within another container. The previous glossed example (a) would be

represented as (1) below, however some additional features that are extra information to the signer have to also be stored, thus yielding (2)

```
today me (not go)[neg] beach (why)[q] rain                    (1)
today (me)[hn] (not go)[neg] (beach)[1] ((why)[q] rain)[past]  (2)
```

In (2), where the extra features are included, `hn` stands for head nod and `past` means the action is signed with a past tense-associated expression (explained in Section 2.1.6). The `1` tag also defines a *group* and in the animation module, this will be interpreted as a pause between the signs for BEACH and WHY. These `Container` objects work similarly to a Rose-Tree data structure, where each node (the container) may have 1 or more children that can also be containers themselves. Building of this container data structure is done in the setNonManualFeatures method in the `IntermediateSentence` class (see Figure 4.2).

4.1.3.5 Output

The final step of the Machine Translation module is to generate the BSL output. At this point all the data is set to a `BSLSentence` object that contains the methods to create the outputs. Given the example (a) above, the output would consists of 3 separate parts:

1. A textual representation: as shown above
2. A HTML Gloss representation:

```
<span id="0">TODAY</span>
<span id="1">ME</span>
<over>
  <span id="2">NOT</span>
  <span id="3">GO</span>
</over>
<sup> neg</sup>
<span id="4">BEACH</span>
<over>
  <span id="5">WHY</span>
</over>
<sup> q</sup>
<span id="6">RAIN</span>
```

Listing 4.4: Gloss HTML format

3. A JSON format:

```
(   // manual features and file paths
  [ { "index": 0, "name": "today", "path": "words/t" },
    { "index": 1, "name": "me", "path": "words/m" },
    { "index": 2, "name": "not", "path": "words/n" },
    { "index": 3, "name": "go", "path": "words/g/verbs" },
    { "index": 4, "name": "beach", "path": "words/b" },
    { "index": 5, "name": "why", "path": "words/w" },
    { "index": 6, "name": "rain", "path": "words/r/verbs" }
  ],
  {   // non-manual feature animation files
    "anims": ["hn", "neg", "past", "q"]
  },  // concurrent animation sequence
  [ { "start": [], "end": [] },
    { "start": ["hn"], "end": ["hn"] },
    { "start": ["neg"], "end": [] },
    { "start": [], "end": ["neg"] },
    { "start": [], "end": [] },
    { "start": ["past", "q"], "end": ["q"] },
    { "start": [], "end": ["past"] }
  ],
  [ // modifier activation
    { "modifiers": [] },
    { "modifiers": [] },
    { "modifiers": [] },
    { "modifiers": [] },
    { "modifiers": ["pause"] },
    { "modifiers": [] },
    { "modifiers": [] }
  ]
)
```

Listing 4.5: JSON format

The HTML gloss is obtained simply by replacing any occurrence of `(` with an opening `over` tag and any occurrence of `[` with the `sup` tag, and analogously for the closing tags. The `over` tag will set an *overline* style on any string within it, while the `sup` tag sets its content to a *superscript*; this is all done through CSS styles. Individual signs are also wrapped in a `span` tag with unique IDs corresponding to their position. This is later used by JavaScript to highlight the words in turn as they are signed (see Figure 3.2 in Chapter 3).

The JSON format contains much more information and it is what is most important for the animation. It is composed of a tuple of 4 objects:

1. A list of objects containing an index, the value and path of a word sign. The index is necessary since fingerspelled words are loaded as separate files, though the index of the word is the same for all letters in said word as exemplified below for the sentence ME THINK -B-O-B- IX_1 NICE

```
{ "index": 0, "name": "me", "path": "words/m" },
{ "index": 1, "name": "think", "path": "words/t/verbs" },
{ "index": 2, "name": "B", "path": "alphabet" },
{ "index": 2, "name": "O", "path": "alphabet" },
{ "index": 2, "name": "B", "path": "alphabet" },
{ "index": 3, "name": "ix_1", "path": "words/i" },
{ "index": 4, "name": "nice", "path": "words/n" }

```

2. A list of animation files to be loaded for non-manual features: we first load all necessary files to be played in parallel with signs and then arrange them according to the 3rd object.

3. A list of objects where every object position represents a word in the first list. At any index position, 0 or more non-manual signs may start or stop playing. This parameter is explained more in depth in the upcoming Section 4.2.3.2.

4. A similar list to the above, but only for modifier features. These include pauses between groups of signs, superlative and comparative adjective modifiers. These are in a separate list because instead of playing different signs concurrently, they modify the sign animation at the index they appear in. The details of sign modifiers are explained in Section 4.2.3.3.

4.2 Animation in ThreeJS

The following section will discuss the techniques used to animate the virtual agent displaying the signed result. It will cover the creation of a rigged model in Blender, the animation and exporting process into a `JSON` format and the implementation of the animation engine to display it. Figure 4.8 highlights the components of the system covered here.

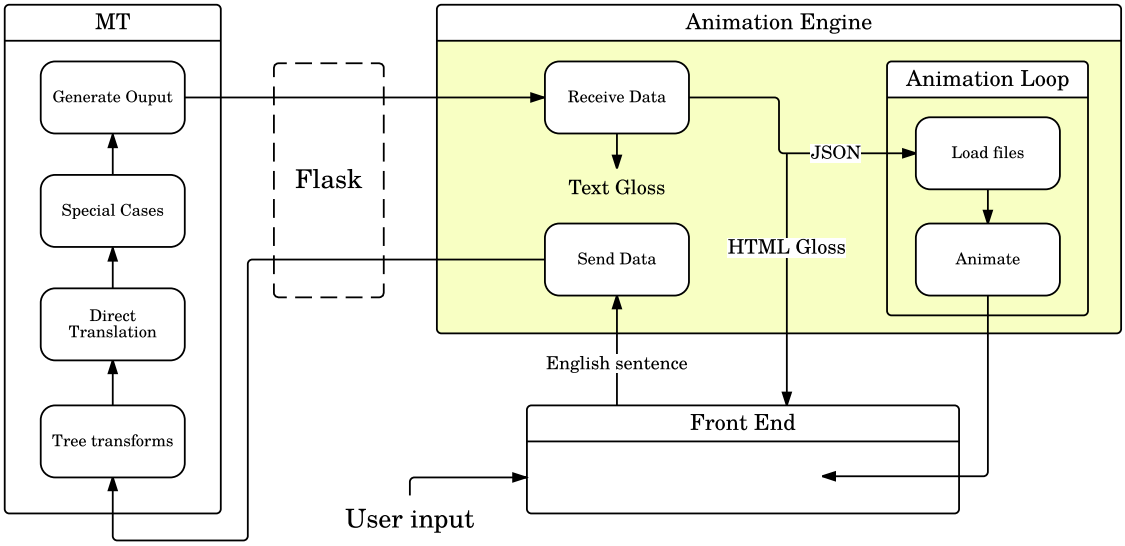


Figure 4.8: System outline: Animation Engine highlighted

4.2.1 Blender

Blender is a general purpose CAD application to create 3D renderings and animations. It was extensively used in the Computer Animation course in Michaelmas Term to create complex skeletal animations for the practicals and final exam. In addition, its compatibility with `ThreeJS` through an exporting plug-in made it the best choice of animation software for this project. Blender was used to rig and animate the avatar model that would display the sign animations.

4.2.1.1 Model

In the Computer Animation course we used an open-source avatar model called Ludwig (Figure 4.9a). While it would have been convenient to use it for this project having a pre-set skeleton structure, it was not compatible with the Blender plug-in to export to `ThreeJS`. Consequently, since it would have been necessary to re-rig the model, the opportunity was taken to find a more "appealing" avatar. Elena (displayed in 4.9b) is one of the most visually pleasing free models available online

(from `http://www.blendswap.com/blends/view/69967`). Unfortunately, being also pre-rigged it was also not compatible with `ThreeJS`'s export plug-in. Thus it was necessary to remove the skeletal structure and manually rig the model.

(a) (b)

Figure 4.9: Potential 3D avatars - Ludwig and Elena

4.2.1.2 Skeletal Animation

Skeletal animation for human figures is the process of constructing a hierarchical bone structure (armature) which is afterwards attached to the overlaying skin, and subsequently animated. *Rigging* describes the action of building a skeleton structure made of virtual bones on a previously modelled object. These bones are not rendered, in the sense that they are not visible in the output, instead they move parts of the object they are assigned to. The skin, technically called *mesh*, of the model object is made of many interconnected vertices that form faces. Groups of faces are assigned a colour or material to modify the appearance of the surface. When rigging a model, each bone is assigned 0 or more vertices and its motion affects the vertex positions according to a specific weight. In Figure 4.10 are depicted two virtual bones i and j. The square box represents the mesh, each bold dot designates a vertex and the extremes of each dotted triangle represent a joint. The weight that a bone exerts on a vertex decreases as the distance between it and the surrounding vertices increases. This even distribution of weights produces a smooth curve between joint rotations.

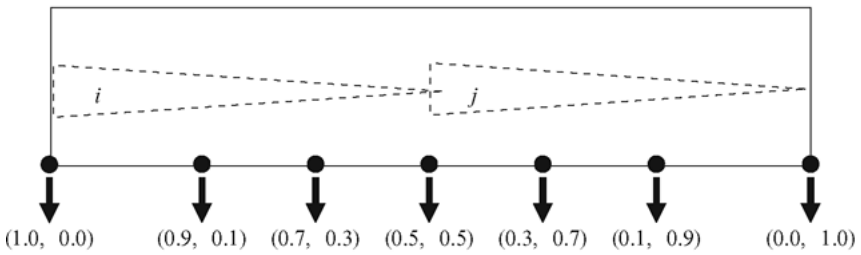


Figure 4.10: A visualisation of the weight falloff; from `http://what-when-how.com`

The weight distribution is normally performed automatically in Blender once the skeleton structure is completed, though some manual tweaking is required, especially with small or detailed parts of the body such as the hands and the face. When performing the rigging, the model is first imported and is set to a pose-position i.e. with the arms pointing sideways away from the body. The skeleton is then constructed bone by bone until a satisfactory freedom of movement is achieved. Given that it must be a hierarchical structure, each non-terminal bone has at least 1 child, except for Inverse Kinematics[3] (IK) bones which indirectly modify the bones they are assigned to. These IK bones include both elbow and hand pairs `elbow_IK.R(.L)` and `arm_IK.R(.L)` as well as those bones to arc the fingers. Figure 4.12a shows the partial bone structure for the Elena model with an x-ray view, and Figure 4.12b shows a more exhaustive textual representation. There are additional bones that are not shown in the representation because of redundancy. Nonetheless, to make it clear, each finger bone such as the thumb or index also has 2 more children to simulate the phalanges, and while each finger has 3 specialised bones to move its phalanges, it is also given 1 IK bone to automatically arc the fingers. Figure 4.11 shows a close-up screenshot of the hand bones as seen from the Blender viewport with the IK fingers highlighted. Rotating any of those bones closer or further away from the palm of the hand will extend or retract the finger bones.

[3]The inverse of Forward Kinematics: given a point or trajectory by the IK bones, the joint angles required for the end effector (the affected bone) to reach a target are automatically calculated.

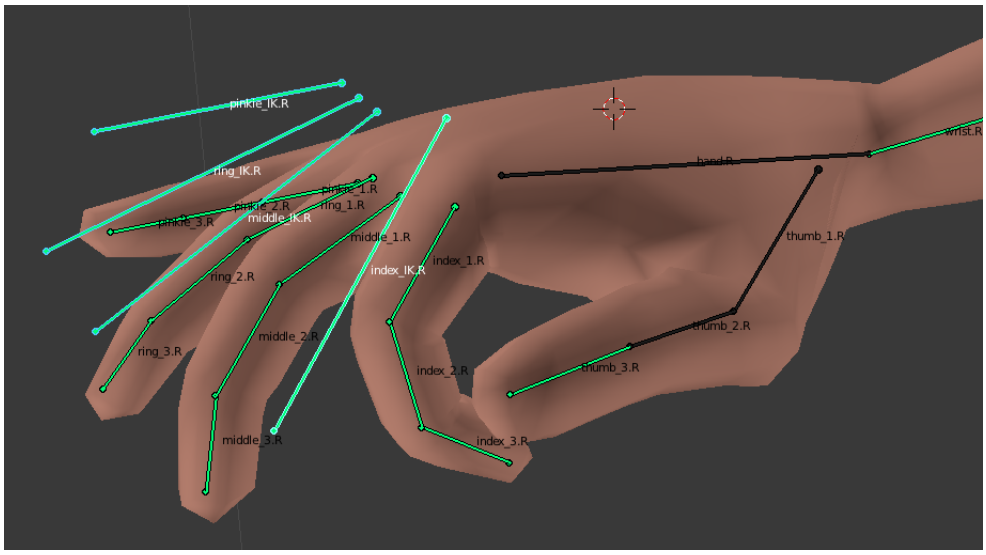
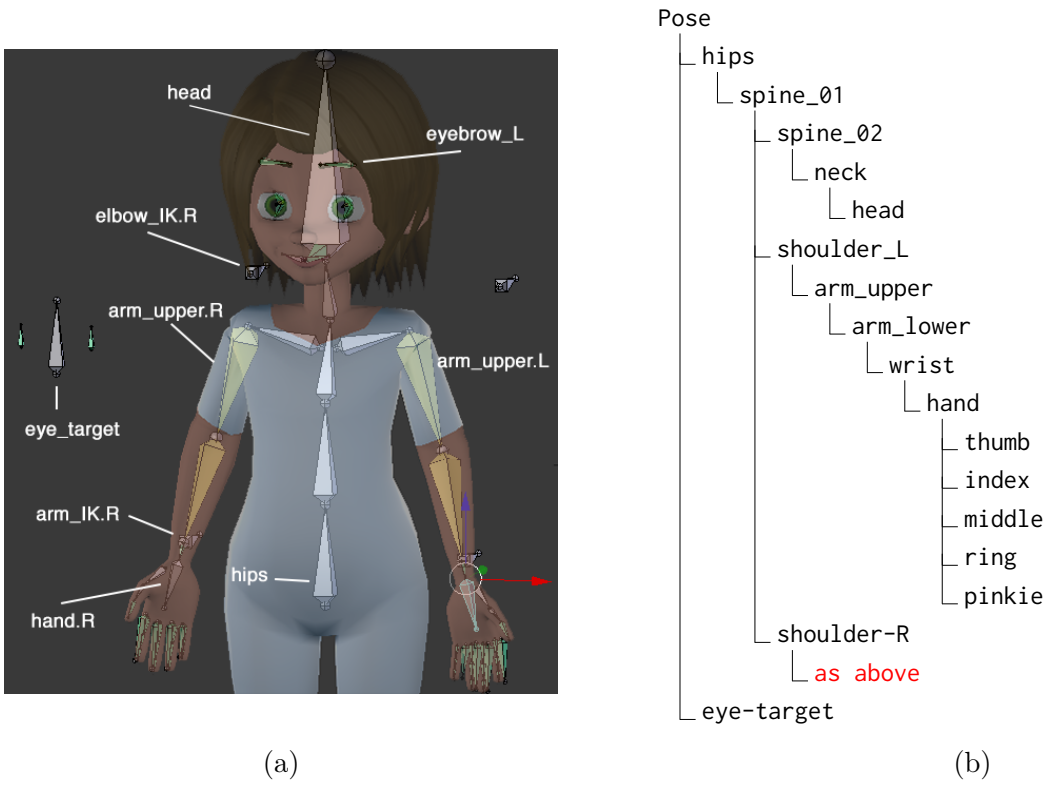


Figure 4.11: Close up of the avatar hand and underlying bones. In light-blue are IK finger bones.



(a) (b)

Figure 4.12: 3D model bone structure

In Figure 4.12a the bones affected by an IK bone are highlighted in yellow. One may move the `arm_IK` bone to lift the hand directly and making the upper and lower arms follow, without having to animate those manually. In other words, IK bones

are not attached to any part of the mesh, instead they affect other bones which, in turn, do modify the mesh. This is solely done to assist during the animation process as the `ThreeJS` plug-in exports animations for all bones regardless of their type. In this particular case the model is missing all leg bones as they are unnecessary within the scope of the project, and their absence does remove some processing load. To animate a bone, we first select it and add a keyframe at a given point in time. Usually a starting keyframe is added for any bone which needs to move from the initial position (both arms down against the sides) to the next. We then select a new position for the bone and set a new keyframe at another time step. The default interpolation value between keyframes is Bezier and was kept that way, since it makes gestures look reasonably more authentic to real-life movements than Linear (Figure 4.13). Every sign is animated individually and then set to textual format through the JSON exporter plug-in.

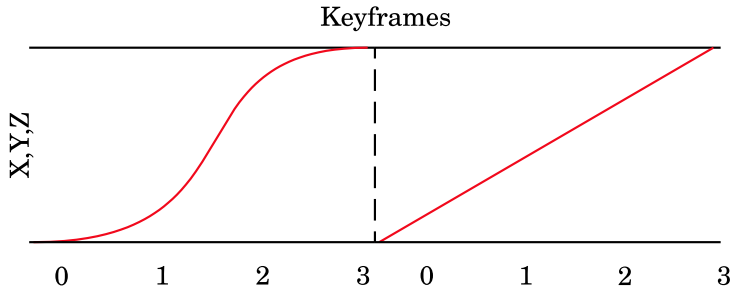


Figure 4.13: Bezier vs. Linear interpolation. The XYZ axis represents the values of each axis in 3D space.

4.2.2 JSON Format and JS Formatter

When exporting with `ThreeJS`, one must select all objects that should be included in the `.json` file, in our case being the mesh and the armature. What results from the export is a large file ($\approx$ 1Mb) containing all the information about the selected objects. The following code snippet shows an overview of the type of data contained in a `.json` file from the exporter

```
{
  "textures":[...],
  "images":[...],
  "object":{
    "matrix":[1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1],
    "children":[{
      "name":"body",
      "matrix":[...],
      "type":"Mesh",
      "material":"B29BD348-16B8-30D6-A840-62213DC577EE",
```

```
11        "geometry":"9142F0FA-03F6-3018-AC00-EF02F6717A93"
12      }],
13      "type":"Scene"
14    },
15    "materials":[...],
16    "animations":[...],
17    "geometries":[{
18      "data":{
19        "normals":[...],
20        "name":"body_plane.001Geometry.1",
21        "vertices":[...],
22        "influencesPerVertex":2,
23        "bones":[...],
24        "animations":[...],
25        "faces":[...],
26        "skinWeights":[...],
27        "skinIndices":[...]
28      },
29      "materials":[...],
30      "type":"Geometry"
31    }]
32  }
```

Listing 4.6: JSON file for ThreeJS

The first level of attributes holds information about the scene, the objects it contains and the global variables such as textures, materials and geometries. The `object` on line 4 defines the base for the mesh of the avatar, the identity matrix and holds the IDs of the materials and geometries that belong to it. `geometries` on line 17 holds all 3-dimensional objects of the scene; in this case there is only 1. Here are defined all geometrical features like faces and their normals, the vertices that make them up as well as the weights for the bones that affect them: by default only 2 bones can affect a specific vertex (line 22). The other very important information is stored in the `bones` and `animations` attributes. We will keep these two aside for now.

As will be explained in more detail in Section 4.2.3, since we wish to keep model and animations separated to be able to import the animations as and when needed, the above snippet with the model data does not contain any animations. All bones are present but do not move. Any new animation file should thus contain information about *which* bones move, and *when* and *where* they move. Here is where the `JS Formatter` comes into play. As every export of a sign from Blender takes up a lot of space due to the large number of vertices and faces stored in it from the model, it is essential to remove this unnecessary and redundant information and only keep what is truly required. As such, a JSON to JS data compression program was written in `Java` and consists of a simple file chooser application where one can pick a `.json` file,

choose a destination directory, purge it of the extra data and convert that `.json` file into a `.js` file[4]. The `ThreeJS` exporter for Blender does not have any sort of options to ignore immobile bones, so the program was written from scratch. This additional step is required to substantially decrease the size of files, essentially compressing them to improve loading times and reduce the network traffic when dealing with animations. The CONVERT found in the formatter function is described in Algorithm 4.2 and is what deals with the removal of unnecessary data.

It is interesting that a plug-in with similar functionality does not exist yet, as by removing unnecessary bones we enable the animation of multiple bones in the same skeleton but from different clips. Imagine that two clips, A and B exist, and both contain animations that apply to all bones in the skeleton. Suppose that A contains animations to move the left arm, and B the right arm. However in both cases *all* bones are assigned keyframes because of the way Blender exports JSON files, thus in A the right arm bones are "animated" as still i.e. they do not move, and analogously in B for the left arm. If both clips were to be played simultaneously, the animations from each would be assigned a weight of 0.5[5]. Hence what would result is an inaccurate animation as all the joint would only complete half of the expected rotations. Thus by removing all the keyframes that do not affect the bones' movements we achieve parallel clip animations with 100% weights.

```
"bones":[{
    "name":"arm_IK.L",
    "parent":-1,
    "scl":[1,1,1]                       // scale
    "rotq":[0,0,-0,1],                  // rotation quaternion
    "pos":[0.625183,2.37131,0.041744]   // position
  },{ ... },{                           // all other bones
    "name":"eye.L",
    "parent":8,
    "scl":[1,1,1],
    "rotq":[0.000658,0.711393,-0.702794,-0.000655],
    "pos":[0.167223,0.180856,-0.319214]
}],
"animations":[{
  "hierarchy":[{
    "parent":-1,        // one for each bone, following bones order
    "keys":[{           // one key for each time step
      "pos":[0.625183,2.37131,0.041744],
      "scl":[1,1,1],
      "rot":[0,0,-0,1],
```

[4]Strangely, `ThreeJS` reads `.js` instead of `.json`. The file structure remains a `JSON` attribute tree, it is only a matter of changing the extension.

[5]The weight is evenly distributed by the number of animations that apply to a bone. If there were 3 simultaneous clips the weight would be $0.\overline{3}$.

```
21        "time":0          // time step and above attributes for bone
22      },{ ... },{         // all other keys
23        "pos":[0.625183,2.37131,0.041744],
24        "scl":[1,1,1],
25        "rot":[0,0,-0,1],
26        "time":1.23333      // last time step
27      }]}, {
28      "parent":-1,
29      "keys":[...]},
30      ...                   // all other anim data
31    ],
32    "name":"love",          // name of particular clip
33    "fps": 60,              // frames per second
34    "length":1.23333        // all time steps end on this time
35  }]
```

Listing 4.7: `animation` and `bones` attributes for the sign `love`

Listing 4.7 details the contents of the `bones` and `animations` attributes. We can see on line 21 and 26 that for each time step there are attribute values for each bone: depending on the export options, timesteps can go frame by frames or skip some to reduce file size. Given 60fps and a time length of 1.23333 seconds, the total number of frames is 74. With 11 keys per bone, the information is being updated approximately every 7 frames. From any newly exported file containing the information shown in Listing 4.6 we get the bones and animations. For each bone and for each keyframe we check if either the location or rotation changes. If it does we keep the bone and animation associated with it, otherwise both are removed. This decreases the file size from 1Mb to approximately 30kb depending on the number of bones moved and duration of the whole clip. The output of Convert is similar to what is shown in Listing 4.7 above, in that it contains the `bones` and `animation` attributes but includes only the bones that move, and a slightly longer duration (line 8 in Algorithm 4.2). The file that results from the conversion is what will be read by the animation engine to display the signs.

Algorithm 4.2 Removing unnecessary data from animation file

```
1: function CONVERT(file)
2:     object ← READFILE(file)
3:
4:     bones ← object.geometries.data.bones
5:     animations ← object.geometries.data.animations
6:     keyframes ← animations.hierarchy
7:
8:     animations.length ← animations.length + 0.3
9:     for each i in bones
10:        still ← true
11:        location ← keyframes[i].keys[0].loc                ▷ First timestep
12:        rotation ← keyframes[i].keys[0].rotq
13:
14:        for each key in keyframes[i].keys
15:            if location ≠ key.loc or rotation ≠ key.loc then
16:                still ← false
17:                break
18:        end
19:        if still is true then
20:            bones.REMOVE(i)
21:            animations.REMOVE(i)
22:            i ← i−1
23:    end
24:    return object(animations, bones)
```

4.2.3 Animation Engine

This section details the JavaScript implementation of the animation engine, including all setup required to build a 3D scene, the animation loop and algorithms to play multiple clips in parallel.

The `ThreeJS` website contains a large amount of fully working examples to get started with creating simple 3D scenes. It also includes help on how to import models with animations from external files. The library supports multiple types of file formats, such as Object (`.obj`), Filmbox (`.fbx`), Collada (`.dae`) and more. To begin with, Collada was the chosen data format to store the model and animations, primarily because it is highly flexible whilst keeping a reasonable file size. In fact, Collada uses XML and when storing animations, instead of defining the attributes for each bone at every n timesteps, it only keeps the data for the starting and ending

positions at every *keyframe* instead. Unfortunately, whilst developing the engine, it was found that the Collada support for `ThreeJS` was not compatible with the included `AnimationMixer` module used to play clips. Thus it was dropped for the alternative `JSON` format instead. The Collada testing was performed very early in the development of the project. In fact, many of the 3D techniques described in the next pages had been approved *before* designing the interdependent translation modules. It was deemed advantageous to ensure the correct behaviour of essential elements such as the 3D animation before attempting to create the underlying algorithms.

In WebGL, and consequently in `ThreeJS`, the following elements are required to create and render a scene:

- The scene itself: containing *all* 3D objects to be rendered
- One or more virtual cameras: these define the viewports of the scene and are what displays the results on the screen
- One or more lights
- A renderer: this is what does most of the processing. The WebGL rendering pipeline is schematised in Figure 4.14, although it is virtually the same for all renderers.

In our own implementation of a scene, we find the following objects:

- Two directional lights and an ambient light to achieve proper illumination of the avatar
- A perspective camera set to view the model from the front by default
- A camera target used to point the camera towards the avatar when not in "fingerspelling" first person view
- The model loaded from an external file

An external OpenGL shading language script is included in the HTML file to get a "skydome" effect with a white ground and blue sky gradient; this is only done for aesthetic purposes.

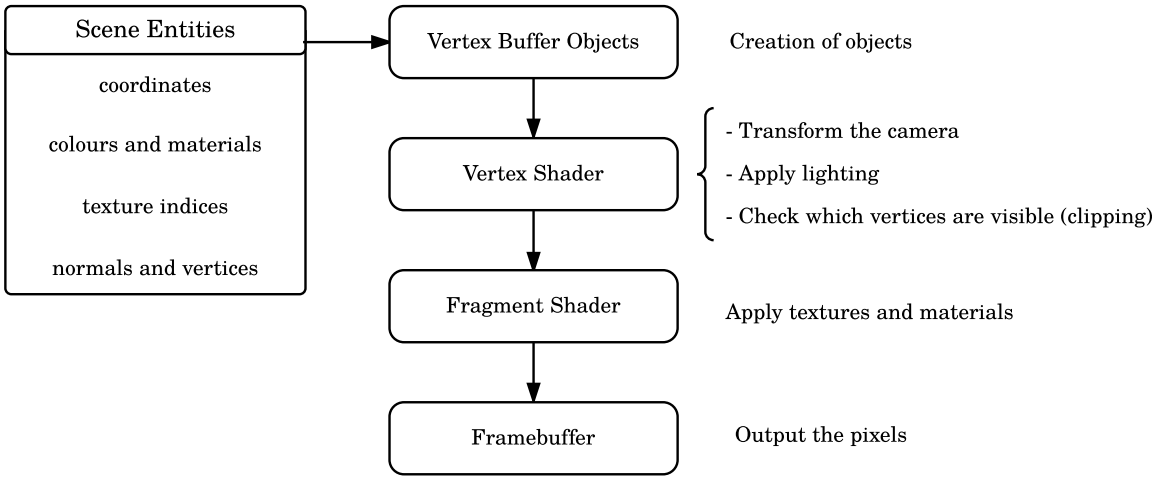


Figure 4.14: WebGL rendering pipeline

We will now discuss the steps taken by the JavaScript side of the application from initialisation to loading new animations after sending a translation request.

4.2.3.1 Scene Initialisation

Whenever the webpage is first opened, the JavaScript file is called and the scene is built. To begin with, all of the aforementioned elements are created, then the model and the two initial animations, `idle.js` and `blinking.js` are loaded. Loading the model requires two steps: first the file is read and geometry data is separated from the materials data and both are stored in different variables. Next, using the native `ThreeJS ObjectLoader` method both are combined to create a `SkinnedMesh`, the technical term used in `ThreeJS` to define a rigged model. The sequence of methods used to load the model is only called once, however any code used to load the initial animations is recalled each time a new translation response is received. The animation loading sequence works as follows:

1. Given a url of animation files in the same format as described in Listing 4.5, the data is converted into JSON using `jQuery`'s `$.getJSON` method. The result of this operation is stored in an array of promises[6].

2. Once all promises have been loaded, we iterate over them and three things may happen:

   (a) If the promise returns a 200 code, then the data was loaded correctly and an `AnimationClip` object is created and added to the list of clips.

[6]Defined as "a single asynchronous operation that hasn't completed yet, but is expected in the future".

(b) If a 404 code is returned, then the previous method is called again but with the `/verbs` path removed. This is done because sign files are stored alphabetically and some signs are identical for both the verb and the noun e.g. LOVE. Recall from Listing 4.5 that any sign animation is given as an object with multiple attributes. Using the `love` example, the animation object for the verb would be `{ "index": 2, "name": "love", "path": "words/l/verbs" }`. However since the sign for LOVE is the same for both the verb and noun, we only store one version in the `words/l` path. See Figure C.4 in the Appendix for a complete structure of the file storage.

(c) If a 404 code is returned twice, then a default `unknown` sign clip is created. This simply loads an animation displaying the model with a confused facial expression and with both hands and shoulders raised (mimicking a "I don't know" look).

3. If the clips are loaded in the initialisation stage, then both are started by calling the `play()` methods. Otherwise additional methods are called to set up the non-manual features clips.

4.2.3.2 Handling non-manual features

As was discussed in Section 4.1.3.5, the server side returns 3 distinct results: a textual gloss, HTML gloss and JSON representation. In the latter we find information about the sign files and non-manual features that need to be loaded, as well as when they have to be played. Using the second argument of the JSON object, we get the names of all non-manual features required for a specific sentence. The following example

I will not go to the beach today because it was raining

$$= \text{TODAY ME } \overline{\text{NOT GO}}^{\text{neg}} \text{ BEACH } \overline{\text{WHY}}^{\text{q}} \text{ RAIN} \qquad (*)$$

yields the following arguments

```
{ \\ second arg
  "anims": [ "future", "hn", "neg", "past", "q"]
},
[ \\ third arg
  { "start": ["future"], "end": [] },
  { "start": ["hn"], "end": ["hn"] },
  { "start": ["neg"], "end": [] },
  { "start": [], "end": ["neg"] },
  { "start": [], "end": ["future"] },
  { "start": ["past", "q"], "end": ["q"] },
  { "start": [], "end": ["past"] }
]
```

The files for `future`, `hn`, `neg`, `past` and `q` are obtained and loaded in a similar fashion to what was just discussed. Then using the third argument of the JSON object we replace each newly created `AnimationClip` object in the data structure so that it can be accessed directly. This operation is performed with a simple loop and an equality check for the clip name. Let us also take the opportunity to better explain this third argument in detail.

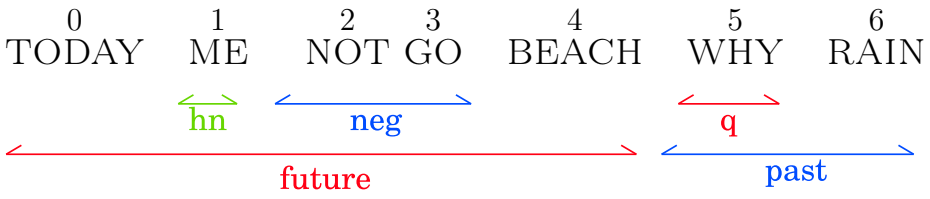


Figure 4.15: A BSL sentence and visualised non-manual sequence

Given a sentence with its container objects set, the concurrent events of non-manual features can be displayed as shown in Figure 4.15. Because the animation engine will play one sign at a time in order of appearance, it was necessary to find a data structure that could easily represent the concurrency of this sequence. The following code shows the above representation in a JSON-compatible format. Each pair in the list contains two inner lists: the first list determines the clips that should start playing and the second those which should stop.

```
   0          1          2          3
[ ( [future], [ ] ),( [hn], [hn] ),( [neg ], [ ] ),( [ ], [neg] ),
   4          5          6
( [ ], [future] ),( [q, past], [q] ),( [ ], [past] ) ]
```

4.2.3.3 Animation Loop

After loading all files, the app will be running in a loop, where the same initial animations (`idle`/`blink`) are repeated until the user requests to make a translation. The `animate()` method represents this loop, where booleans are constantly checked to ensure the correct methods are called. There are 4 important tests happening here:

- If `display_translation` is true, then the boolean `first_step` is set to true to activate the clips that were previously loaded. The boolean to display the translation is always set after the translation response has been received and all clips have been loaded.

- If any of the step variables is set to true, these being `first_step`, `continuous_step`, `final_step` we check if the clips need to be paused, if the automatic camera setting is enabled and call the methods to play the non-manual features if necessary.
- If the cancel button is pressed then all manual and non-manual clip variables are reset.
- If we have reached the end of an animation sequence for a particular sentence, the non-manual variables and the interface settings are reset.

In addition at the end of this loop there is the call to the `render()` method, which takes care of updating the camera, the renderer itself, the statistical data such as framerate and the `Tween` library[7] used for altering modifier clips (see the following Section 4.2.3.4). Whenever a new response from the server arrives, the files are loaded and the sign clips are played in sequence. The full pseudocode for this sequence can be found in Appendix B.3. Because the rendering implementation makes constant calls to the `animate()` method, each step in the display loop must be entered and exited through booleans otherwise the clips would start playing every time a render call is made, around every 10-15ms [27].

4.2.3.4 Non-manual features and modifiers

The loop described above also takes care of calling the methods for playing non-manual features and altering sign animations whenever a modifier is involved. Following the structure described in Section 4.2.3.2, we access the two lists at the index given by the main loop. Then if a clip is found in the `start` list, it is played. Alternatively, if a clip is found in the `end` list, it is ended by simply calling the `fadeOut()` method. The loop for modifiers is separate and analogously to the above it accesses modifier commands from the given index position. The 3 types of modifiers and their effects on animations are shown in Table 4.5. These effects are applied using the `Tween` library, which allows us to interpolate between any value of any object in JavaScript. To begin with, whenever a modifier is found, the clip assigned to it is paused, and then its time-scale is changed according to the parameter of the modifier. Finally the time-scale is interpolated following a specific delay and easing, though these do not apply to the *pause* modifier.

[7] `http://www.createjs.com/tweenjs`

Modifier type	Duration (δ)	Start timescale	End timescale	Delay	Easing
Comparative	$\frac{(dur - 0.3) \times 1000}{\alpha}$	$\alpha \times 0.2$	$\alpha \times 1.5$	$\frac{\delta}{3}$	Quadratic
Superlative	$\frac{(dur - 0.3) \times 1000}{\alpha}$	$\alpha \times 0.1$	$\alpha \times 1.7$	$\frac{\delta}{1.5}$	Quartic
Pause	$dur+ = \frac{0.2}{\alpha}$	n/a	n/a	n/a	n/a

Table 4.5: Modifier parameters. *dur* stands for the clip's original duration and α for the global animation speed (range between 0.2-2.0)

4.3 Cross-language Communication with Flask

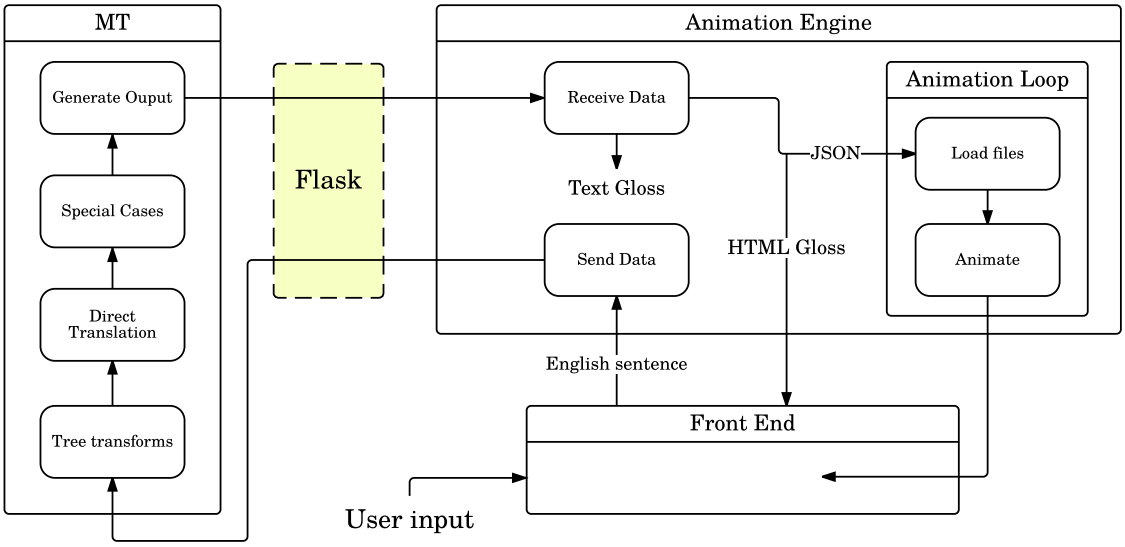


Figure 4.16: System outline: Flask Framework highlighted

Here we very shortly touch on the communication between JavaScript and Flask. Flask is a microframework for Python that allows bidirectional communication between the server and browser as depicted in Figure 4.16. The web application is essentially driven by Python: when the website is uploaded to the Heroku server, the Flask app is created. From here the app *route* is defined. The route that points to the root (line 7 in Listing 4.8) contains the method that is called when the root address of the website is entered, which returns the main HTML page where all of the JavaScript imports are made. Then any route created by Flask will point to where JavaScript will send the request from. In this case on line 12 in Listing 4.8 the route is `/_process_text`, thus JavaScript sends the request to the same address as can be seen on line 5 in Listing 4.9. When Python is done processing the data, it is returned and can be further exploited by JavaScript.

```
1  # Initialize the Flask application
2  app = Flask(__name__)
3  # Initalise the translating analyser
4  analyser = Analyser(app=True)
5
6  # This route will show a form to perform an AJAX request
7  @app.route('/')
8  def index():
9      return render_template('indexPage.html')
10
11 # Route that will process the AJAX request, result as a proper JSON
       response (Content-Type, etc.)
```

```python
@app.route('/_process_text')
def process_text():
    text = request.args.get('input_text', '', type = str)

    sys.stdout = open(devnull, 'w') # suppress printing
    data = analyser.process(text)
    sys.stdout = sys.__stdout__     # reset printing

    # data[0] is gloss, data[1] is html, data[2] is json
    return jsonify(result=(data[1],data[2]))

if __name__ == '__main__':
  app.run()
```

Listing 4.8: Python and JavaScript communication, Python code

```javascript
function beginTranslate(){
    var text = $('input[name="input_text"]').val();
    if (Interface.current_text != text) {
        // get the text from the textbox and send it to python
        $.getJSON('/_process_text', {
            input_text: text
        }, function (data) {  // on finish request receive data
          // print the result on screen (gloss)
            Interface.setGloss('bsl', data.result[0]);
            ... // read clips and other data...
        }
    }
}
```

Listing 4.9: Python and JavaScript communication, JavaScript code

4.4 Summary

In this chapter we discussed the implementation of the Machine Translation module with Python, the different components that make it up and showed how an English sentence is transformed into BSL. The biggest challenge was finding a method that could properly capture all syntactic and semantic elements of BSL. In fact, while the signs themselves are crucial, all non-manual features also play a meaningful role when signing. Furthermore, we described the virtual avatar and how it is animated in Blender, the exporting process and the file format read by JavaScript. Substantial amount of work went into creating complementary software to assist the translation and animation process, such as the HTML scraper for obtaining additional translation data and the `JS Formatter` to remove unnecessary objects from the Blender animation exports. It was also shown how the animation engine implemented in `ThreeJS` reads the animation files and the way they are displayed in a concurrent sequence when

a translation request is made. Finally we touched on the communication between the two modules using the Flask microframework. All of the objectives mentioned in Section 1.2 have been met, however to assess the effectiveness of the implemented system, a formal evaluation is necessary and is discussed in the following chapter.

Chapter 5

Evaluation

In this chapter, the evaluation for the system has been divided into two main parts: a linguistics perspective to determine the accuracy of the translation according to a small dataset of reference translations, and a user testing feedback obtained through a survey. In addition, it was possible to get two domains experts: Rachel Sutton-Spence, co-author of the book *The linguistics of British Sign Language: an Introduction* and Adam Schembri, Lecturer in Sociolinguistics at the University of Birmingham to review the system and provide constructive feedback.

5.1 Translation

Before discussing the results for translation accuracy, it is necessary to understand that the current evaluation methods are not entirely fitting when compared to the evaluation of written-written translation. While glosses allow for an easy evaluation approach using string similarity metrics, this representation does not properly encapsulate all the linguistic information that is normally carried through signing [23]. In fact, the accuracy evaluation only takes into account the exact gloss representation, and ignores all visual-spatial knowledge. In other words, it is possible to inspect the correct use of signs in a translation, including the parallel non-manual features, however it is difficult to automatically evaluate how "good" an animation for a particular sign is.

Section 5.2 gives a comparison of the system's performance and compares those described in the Background Section 2.2.1. However, due to the little amount of formal evaluation results provided, more related work is presented here which, in turn, includes evidence of empirical results. These other projects have not been discussed in the background because their implementation closely follows that of previously

mentioned methods and focuses on the translation of sign languages not derived from English.

5.1.1 Evaluation Metrics

BLEU (BiLingual Evaluation Understudy) is a language-independent precision metric for translation that compares a candidate translation (the hypothesis) with one or more reference sentences [25]. It is currently the most popular evaluation metric for translation in the field of Computational Linguistics. Based on a modified n-gram precision metric p, it counts the maximum number of times a word appears in any of the reference translations. The count of each candidate word is then reduced by the maximum count and a penalty is added for brevity of sentences i.e. the shorter the sentence the more impact any incorrect word placement will have. It is typically applied over a whole corpus and not between single sentences, although a Smoothed BLEU method exists for this task [6]. For this metric, the higher the score the better.

$$\text{BLEU} = \left(\prod_{n=1}^{N} p_n \right)^{\frac{1}{N}} \times \text{BP}$$

$$\text{where } p = \frac{\%\text{ of } n\text{-grams from hypothesis in reference}}{\%\text{ of } n\text{-grams in reference}},$$

$$N = n\text{-gram size of 1 to 4}$$

$$\text{and BP} = \min\left(1, \frac{\text{hypothesis length}}{\text{reference length}}\right)$$

WER (Word Error Rate) works on a word-level (Levenshtein) distance comparison between words in a reference translation and a hypothesis. The score is calculated as the minimum number of editing steps[1] required to go from the reference to the hypothesis. Essentially, any word that is not found in the same place in both sentences adds a penalty to the overall score. While more primitive to BLEU, many translation systems also include this score in their evaluation. For this metric, the lower the score the better.

$$\text{WER} = \frac{n.\text{ of substitutions} + n.\text{ of insertions} + n.\text{ of deletions}}{\text{reference length}}$$

[1] By editing steps, we mean any word level transformation from the reference to the hypothesis. For instance, if a word found in the reference is missing in the hypothesis, then that is considered an editing step; steps include substitutions, insertions and deletion of words.

5.1.2 Short evaluation of previous projects

Given the very little amount of results provided by previous systems, it is extremely hard to get an accurate idea of how each system compares with one another. None of the rule-based systems included statistical results; nearly all the information was in the form of "performs reasonably well" or "not quite there yet" which are highly broad and cannot be considered as empirical comparisons. Let us quickly go through a short review of each of the works included in the results below.

MaTrEx 1 Airport Announcement Accessibility [22]. This SMT implementation focuses on the airport information domain, where security and general announcements are automatically translated into ISL (Irish Sign Language). An English-ISL bilingual dictionary was created from the data obtained from the ATIS (Airline Travel Information System) dataset. In this paper they provide results for both WER and PER (Position-Independent Error Rate). PER, in contrast to WER, also takes word ordering into account. Their testing reports a minimum WER score of 41.68%.

MaTrEx 2 This work is developed by the same authors from the above. Here they use a similar SMT approach but with an improved corpus and more insightful evaluation metrics [23]. In a similar fashion to the corpus used in our evaluation, they remove sentences that include classifiers from the overall corpus. However they also omit all non-manual features from the evaluation, only keeping them in the animation output. This choice fundamentally affects the accuracy results obtained in the evaluation, and the authors note that it does not allow for a fully accurate representation of sign language and machine translation evaluation. In addition to WER and PER they also provide BLEU. They achieve a maximum BLEU score of 45.64% and a minimum WER score of 54.56%.

LSE MT Spanish Sign Language [28]. This system attempts translation from Spanish to LSE (Lengua de Signos Española) and includes both rule and statistical based approaches with empirical results for each. Their domain is limited to Driver's Licences and ID renewals, with a corpus made of around 4000 sentences annotated in glosses and SiGML. Here only a BLEU score is provided for each approach using SiGML, with the best being 68.23%.

TSL MT Chinese to Taiwanese SL [40]. One of the few hybrid systems to include statistical evaluation, it utilises a language specific notation to TSL in a bilingual Chinese-TLS corpus containing around 36.000 sentences. Like the above, it provides a BLEU score in addition to the IBM Model 3 score, a metric which is not considered further in this evaluation being a purely SMT focussed metric. They achieve a score of 86% for BLEU.

5.1.3 Data and Approach

This system was created using a manually expandable rule-based method since no real BSL corpus exists to this day. All of the previous works have either created their own corpus thanks to very patient expert translators and annotators, or used corpora from very specific domains which would be unsuited for the purpose of this project. However, in order to properly observe the translation accuracy it was necessary to find data to use as reference. The website HandSpeak[2] is an American Sign Language learning resource that provides lessons as well as many example sentences. These sentences are formed by a gloss text and an English equivalent text annotation; the text is also accompanied by video. Because the predominant difference between ASL and BSL are the signs themselves, and the structure of sentences stays relatively similar, it was possible to use those sentence pairs as evaluation data. The glossing format used on the website does not quite follow the same one used in our system. For instance all of the sentences marked by a question do not specify exactly which signs are signed with a "q" expression and likewise for "neg". Some glossed sentences also keep punctuation which is not necessary, and some of the target glosses can be applied to more than one English version. For the example $\overset{\text{q}}{\overline{\text{YOU FEEL COLD}}}$ we find two translations: "Are you cold?" and "Are you feeling cold?".

One of the major advantages of using an online corpus through an HTML scraper is the ability to filter unwanted or corrupted data. Some examples include mathematical symbols such as "Is $24 \div 4 = 6$?" which our system cannot handle. The use of classifier predicates (see Section 2.1.2) in reference sentences also required further manual filtering. These constructs use a very particular notation:

The cat looked at the mouse and walked away. (1)
= MOUSE [left] CAT [right] LOOK-AT WALK-AWAY [loc]

A pen on the paper. = PAPER CL-PAPER PEN CL-PEN [loc] (2)

[2] http://www.handspeak.com

In example (1) the left and right notation specifies the location of the entities in the syntactic space (see Section 2.1.3.2). In (2) the CL- means that the hand assumes the correct handshape for that particular entity. In this case the classifier for paper would be a B hand with the palm facing down to imitate the top of a table, and the pen classifier would be represented by a G hand for long and thin objects. For this reason the sentence pairs extracted from the website have also been split into those *with* classifier predicates and those *without* as it was deemed beneficial to perform accuracy results on respective and combined datasets.

Therefore both the source and target were formatted to comply with our evaluation method. Manual formatting included the removal of inconsistent brackets, insertion of dashes for fingerspelled words e.g. "Joe" = -J-O-E- and the alteration of the Index notation from `IX-me` to `IX_1` which is the notation employed by this system. Moreover, to ensure that word orderings of the glosses would abide by BSL grammar rules, reference translations in this reduced corpus were adjusted accordingly. Table 5.1 shows general information about the corpus used in our evaluation.

Dataset	Stats	English	BSL
Basic	N. of sentences	185	
	Avg. sentence length	6.76	4.95
	N. of words	1251	916
Classifiers	N. of sentences	49	
	Avg. sentence length	8.24	6.94
	N. of words	404	340
Total	N. of sentences	234	
	Avg. sentence length	7.07	5.36

Table 5.1: Test corpus statistical information

The following section discusses the accuracy of our system using this corpus. To assure the reader, it is important to point out that the dataset considered for the evaluation was never employed as a *training* set. In other words, the rules written to model the transformations from English to BSL were based on the knowledge previously acquired and not from the sentences in this corpus. After formatting the sentence pairs obtained with the web scraper, they were kept as a "black box", in the sense that their grammatical structure was never analysed. Whenever a new rule is added to files read by the translation module, the system accuracy is tested to assess if the rule improves or worsens the overall performance. Should the accuracy be reduced, we can use the aforementioned Smoothed BLEU [6] metric to compare

sentences that may have been affected. Table 5.2 shows a possible output of the testing using the Smoothed BLEU for individual sentence pairs.

English	BSL Reference	BSL Hypothesis	S-BLEU Score
Do you need help?	`(YOU NEED HELP)[q]`	`(YOU NEED HELP)[q]`	1.0
Have you got any story?	`(HAVE ANY STORY)[q]`	`(YOU GET STORY ANY)[q]`	0.502
I like you a lot.	`ME LIKE YOU LOT`	`ME LIKE YOU LOT`	1.0
How do you feel?	`YOU FEEL (HOW)[q]`	`YOU FEEL (HOW)[q]`	1.0
When did you move in?	`YOU MOVE-IN (WHEN)[q]`	`YOU MOVE IN (WHEN)[q]`	0.865
Keep in touch.	`CONTINUE TO-CONTACT`	`KEEP IN TOUCH`	0.217

Table 5.2: Output for pair-wise sentence evaluation

The BLEU score for the sentence "When did you move in?" can be easily improved by adding a new word sequence `move in` in the SignBank set to combine them into a single sign. On the other hand we can see from the last sentence "Keep in touch" that the reference translation is completely different from the system output as the concept of "maintaining communication" uses signs for concrete action of touching to convey an abstract meaning. To achieve the correct translation, this particular sentence would require a targeted direct translation rule.

5.2 Accuracy Results

System	Approach	Data/Format	WER (%)	BLEU (%)
LSE MT	Rule-Based	SiGML	n/a	64.33
	Statistical	SiGML	n/a	68.23
TSL MT	Hybrid	TSL gloss	n/a	86
MaTrEx 1	Statistical	Gloss	41.68	n/a
MaTrEx 2	Statistical	Gloss	80.37	31.84
		SiGML	54.46	45.64
This system	Rule-Based	Gloss (w/o classifiers)	54.88	64.26
		Gloss (only classifiers)	86.34	40.69
		Combined	61.47	57.40

Table 5.3: Accuracy results for SL translation. For WER lower means better, for BLEU higher means better. For clarity, red highlighting represent a poor score and green a favourable score.

Table 5.3 shows this system's accuracy compared to those described in Section 5.1.2. The results fully confirm the claim made in Section 2.2, stating that *hybrid* approaches usually deliver the highest accuracy. We can also note that the rule-based approach in

LSE generates a lower BLEU score than the statistical method on the same data, again confirming what was previously expressed. Our system achieves a decent accuracy on the dataset without classifiers but also a rather large error rate. It is therefore safe to assume that many of the words from the reference translation actually appear in the wrong position, or even do not appear at all, in the hypothesis. We can show this in the following example taken from the corpus:

English	I haven't eaten yet
Reference	ME EAT $\overline{\text{NOT-YET}}^{\text{neg}}$
Hypothesis	ME $\overline{\text{NOT EAT YET}}^{\text{neg}}$

Yielding a Smoothed BLEU score of 76.96% and WER of 66.6%. We can visualise the workings of WER in Figure 5.1. Following the given formula for WER we find a value of $\frac{2}{3}$ where the numerator is given by 1 substitution and 1 insertion, and the denominator 3 being the length of the reference. Because we are splitting the two sentences through single spaces, we also keep all information about the position of non-manual features. The (NOT chunk in the hypothesis shows that a non-manual feature begins at that point. Had the reference translation been NOT at that position, it would have counted as incorrect; this is indeed the desired behaviour.

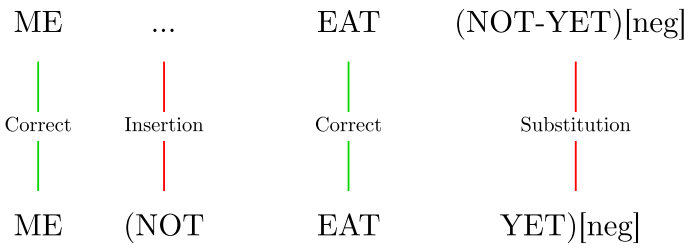


Figure 5.1: WER operations; green lines show correct alignments, and red lines incorrect alignments.

Let us look at another example but with the classifiers dataset (the workings for this example are shown in Figure. 5.2):

English	The police stormed the building
Reference	BUILDING CL-BUILDING POLICE STORM-IN
Hypothesis	POLICE STORM BUILDING

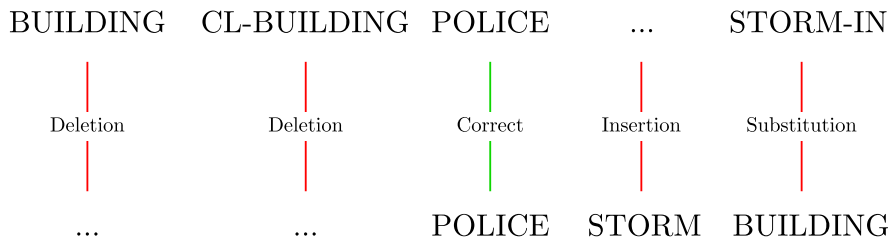


Figure 5.2: WER operations on a classifier predicate sentence

Here we obtain a Smoothed BLEU score of 42.89% and WER of 100%. The WER numerator value 4 is given by 2 deletions, 1 substitution and 1 insertion. Since there can be any number of insertions, the overall score may go over 1. It is then clamped back to the max value of 1, or 100%, although it is not the case here. Note that, from the outset, classifiers were never intended to work with the system, and by looking at the WER score on the classifier-only dataset this becomes apparent. A 86.34% error rate means that nearly *all* of the words in the reference translation differ from the actual output hypothesis. This is understandable since the translation module is not designed to output gloss with classifier predicates, nor is it capable of resolving classifier predicates from an English sentence.

Even though the combined score is accounted for, it would be sensible to consider the classifier-free subset as the most significant measure. Once again, it is crucial to remind ourselves that these evaluation scores are highly discrepant as a result of the greatly varying corpora employed by each approach and the divergent decisions made by the authors to omit or incorporate non-manual features.

5.3 Survey

To collect information on the general usability of the application a survey was made public for people to test the website. The survey covered questions on bugs, ease of use of the website, the appeal of the avatar and understandability of the sign animations. All questions from the survey can be found in Appendix D. There were 43 responses in total with an audience ranging from students to professionals from different backgrounds. All responses were anonymous.

To begin with, a general question about each respondent's fluency in BSL or other equivalent sign language was asked. This was done to get a better insight of how different levels of knowledge judge an online tool. The intention of this question was to distinguish praising comments from someone with a high level of BSL, from someone with no or little knowledge, since the opinion of a skilled user would be more

valuable in comparison. 36 participants had no experience in BSL whatsoever, 5 knew a couple of signs, and 2 had enough skill to communicate with the Deaf in every day situations (Level 1-2). Unfortunately no one had a BSL level higher than 3, however that is why the feedback of linguists was specially sought after and their remarks are discussed in Section 5.4. Around 75% of the participants with no experience also expressed some sort of interest in BSL, be it just curiosity about how Deaf people communicate or eventually wanting to learn the language.

5.3.1 Usability

Users were asked to specify the browser and device employed to access the website in order to accurately identify bugs and browser specific issues. Figure 5.3 shows the given responses; please note that an additional "Internet Explorer" field was available, though it was never selected. Following that, they were asked to try the website a try following some simple guidelines on BSL translation and a short explanation of the interface. 11 out of the 43 participants mentioned experiencing some sort of bugs. Most were related to the laggy animations, probably due to optimisation issues on the users' devices. When comparing the bugs and the devices they appear on, no correlation was found. It is safe to assume that the bugs are not related to implementation issues, but rather appear to be specific to the users' browser or device version. Some users actually misinterpreted the unknown words alert (highlighted in red and accompanied by the shoulder shrug) as bugs. This suggests that some other method of specifying unavailable signs might need to be considered.

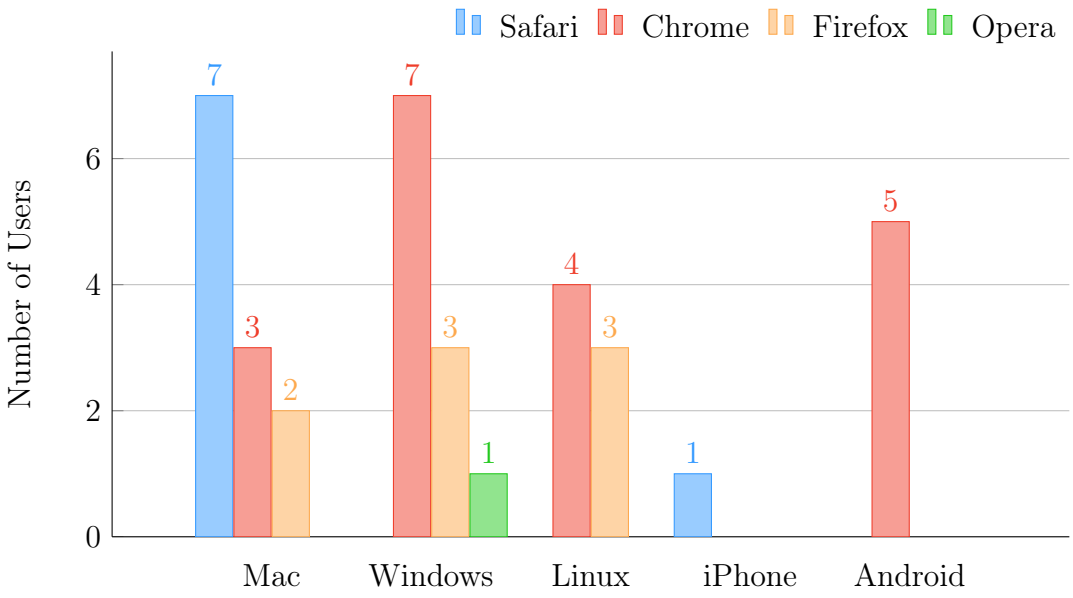
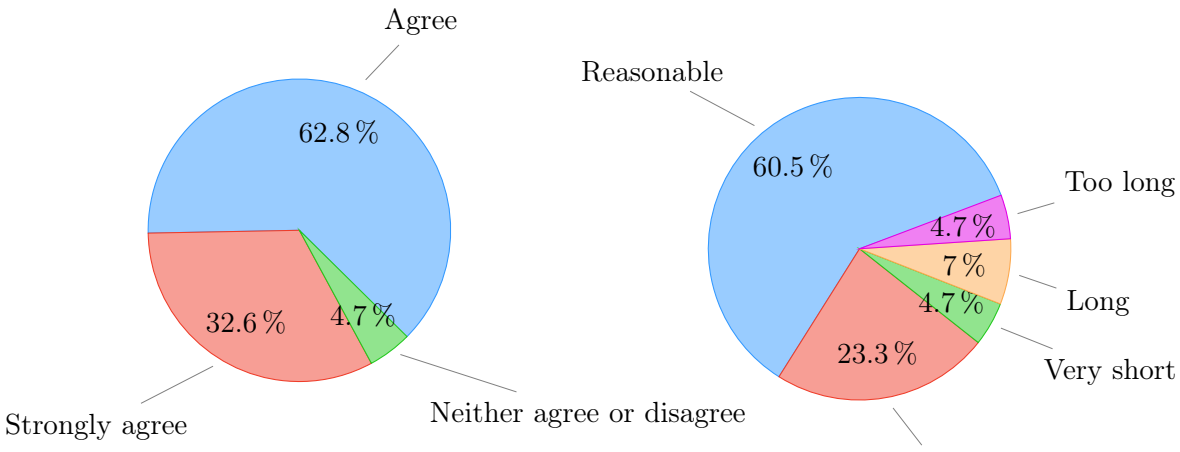


Figure 5.3: Devices and browsers running the website

There was a strong agreement towards the understandability of the interface elements though some people did not grasp the meaning of the "A" button for suggestions, even though it was clearly stated in the questionnaire (Figure 5.4a). When asked about loading times the majority rated them as "Reasonable" or "Short" (Figure 5.4b). However there were cases where loading was too long, and this might be due to 2 particular reasons:

1. A reduced internet connection will certainly increase loading times as all the scene variables have to be created and the model must be loaded; recall that the avatar file exceeds 1Mb.

2. If the user was the first to access the website after an idle period, they would have experienced longer loading times. The application is hosted on a free account of the Heroku platform and their terms and conditions state that any free website is given 550 hours of activity every month. All websites are managed by `dynos`, and any `dyno` will be set to idle after 30 minutes of inactivity. Thus, if a user accesses the website while the `dyno` is sleeping, their *initial* loading time will be longer than usual. On the other hand all other delays should not be longer than a couple of seconds.



(a) The UI elements are clear and easy to understand

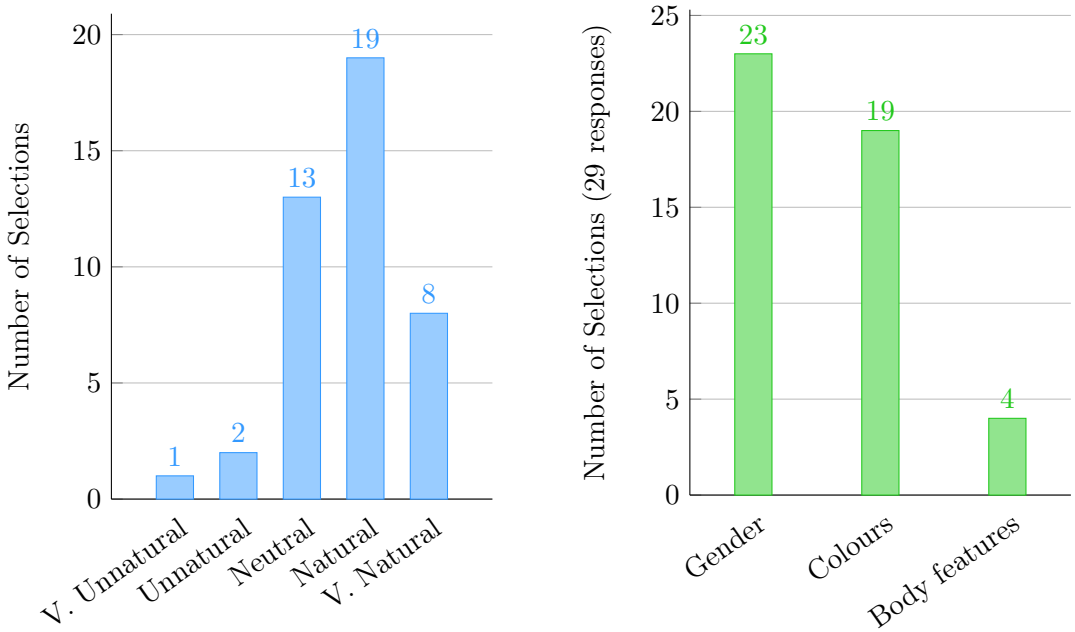
(b) Loading times

Figure 5.4: Visualised data for responses on clarity of the UI and loading times (43 responses)

5.3.2 Effectiveness of the Application

Aside from the technical evaluation of the website, users were also presented with a section investigating the avatar itself. This included questions such as "Does the avatar come across as friendly?" and "How would you rate the movements of the avatar?". The vast majority of users (86%) found the virtual agent appealing and rated its movements as quite natural; the choice was between 1 for completely unnatural and 5 for very natural (Figure 5.5a). Interestingly, those people whom did not like the look of the avatar also found its movements unnatural. It was shown in [3] that "individuals are more influenced by agents who are similar to themselves with respect to appearance-related characteristics". This might explain why the participants in our survey highly requested the customisation options to achieve their preferred avatar look. Figure 5.5b shows the different elements users would like to be able to edit, with gender being the most requested. Interestingly, for my previous undergraduate project the use of a "male"[3] virtual avatar sparked some controversy as many did not appreciate the inability to choose a female equivalent. This particular preference is also discussed in [3], where they suggest that younger individuals, especially students, may find female agents more "powerful role/social models", which explains the request for a female avatar. Surprisingly, even when presented with a female agent, gender swap was still the most requested customisation feature.

[3]Quotes are used in this context because the model did not quite resemble a man, but had more of a humanoid look. Though the overall appearance bore more resemblance to a male human and somewhat comparable to the Ludwig model previously shown in Figure 4.9a.



(a) Rating of avatar movements

(b) Additional customisations

Figure 5.5: Movement ratings and avatar customisation including skin and hair colour, avatar height etc.

Participants were also given the task of setting the "Show non-manual features" in the options and attempt to follow the animation at the same time as the notifications (see Figure C.1 for reference). Again, the vast majority was very well able to do so (86%), with some mentioning the speed change feature to be very useful in that regard.

Of the two users with enough experience that completed the survey, one did not have the possibility to try the website because of a missing graphics driver that prevented the avatar from appearing. To the question about translation accuracy, the other replied with "There is a fair amount of accuracy", not being very precise. However they did point out that the flat hand for possessive pronouns is incorrect (ASL uses the flat hand, BSL uses a closed fist) and the fact that London should not be fingerspelled as it has its own sign. These two observations are also discussed by the experts in the next section[4].

As was previously mentioned at the beginning of the report, the application was built to make British Sign Language translation available online to those currently learning that would like to practice, and for people who are simply curious about the language. However it is not meant to be used solely as a learning tool. Users

[4]Please note that the two experienced users that participated to the survey may not necessarily be Rachel and Adam, the two consulted linguists.

were asked if they would recommend the website to someone curious about BSL and to someone learning it, with most people praising the idea of such a tool. To be exact, 90% would recommend it to someone *curious* about BSL, while 81% would recommend it to someone *learning* the language. This slight division of opinion might be a result of potential incorrect translations: it would not be wise to use an inaccurate teaching tool, however the occasional mistake can be overlooked if individuals solely use it to get an idea of the language. Looking at the comments linked with the same question, some specifically commended the ease of use and simplicity of the website. The ability to change point of view (from 3rd to 1st person) was well received. Others also liked the fact that the translation is explained, mainly in the additional info, but also thanks to the non-manual features notifications. The experienced user pointed out the range of situations where this tool could come useful

> "I would recommend this mainly because I think it may have more potential than is first thought. I can think of different applications going forward in the field of deaf education but also for people who are fluent in BSL but who find written text difficult to understand. Of course there would need to be a lot more words etc inputted for this latter use. Then there is also the interest factor. The syntactic changes between English and BSL are interesting and well demonstrated here. It could form one useful way of practising the language if enough words were inputted. The learner could see the sentence first, try to practise it in BSL and then see the Avatar do it as a means of checking progress."

Nonetheless, there were also some negative comments, mainly linked to the lack of vocabulary. This is understandable, as there are many important signs describing emotions and basic actions such as "drinking" which are not available. Most signs are simply missing because of time limitations, as each sign animation takes around 5-15 minutes to create depending on the complexity of the hand gestures. On the other hand, many verbs cannot be added yet because they require syntactic agreement with the object. As was shortly described in Section 2.1.1, a verb sign may be modified by the object it describes. For instance the action of drinking wine is signed differently from the one drinking coffee because the information about *what* is being consumed is incorporated directly into the handshape of the verb. Thus, it would be wrong to add a sign for "drinking" since there is no universal sign for that action: it requires a noun modifier.

Less significant criticism was aimed at the interface. Some users did not particularly understand the placement of the "Translate" button on the far right, and would have instead preferred it near the input box. In addition, the "A" button for suggestions was not clear enough and as a result people tried to insert sentences that did not quite work; particularly, names were entered without being capitalised, leading to the system not recognising them as such and thus were not fingerspelled.

Despite the complaints, the application was overall well received by the general public and notably endorsed for its simplicity and innovative application of BSL translation over the browser, making it easily accessible to anyone on a multitude of devices without the need to install it. While there are some changes that could be made to improve its usability, the lack of signs does impose a problem as not many BSL sentences can be generated presently.

5.4 Feedback from BSL Linguists

During the time the survey was available, I got in touch with Rachel Sutton-Spence, author of the book employed to learn the linguistics of BSL. Adam Schembri, lecturer in Sociolinguistics at the University of Birmingham was also approached. The website and survey were made available to them to have a first-hand experience and to provide general thoughts about the translation and avatar signer.

Similarly to the general opinion of the public, the aesthetics of the website were well received, although Rachel expressed that the idea of a virtual signing avatar required some getting used to. This isn't surprising as, in fact, virtual agents used in teaching or training environments are still not universally accepted as comparable to humans [18]. However one of the main advantages of using a virtual agent is the ability to make it infinitely repeat an exercise or action for the benefit of the user. It is important to understand that depending on the implementation of the avatar and the techniques employed to animate it, a potential lack of realism and restriction on the type of interactions can lead to a degraded user experience. Manually animating humanoid figures requires skills and is often time consuming. That is why many prefer to use motion capture instead; this approach is discussed in more detail in the future work Section 6.3. Adam on the other hand, specifically mentioned that the avatar was understandable and that the animations were very clear.

The rest of the comments were targeted at the translations. Firstly, because the system uses a generic rule for fingerspelling, any proper noun is automatically fingerspelled regardless of the type (person, place, brand etc.). For this reason any sentence

that includes a city or country name would actually show that particular word as being fingerspelled, such as Italy = -I-T-A-L-Y-. From the email correspondence with Rachel she expressed the following concern

> "A blanket rule that all Proper nouns are fingerspelled is not really reliable, though, because there are many proper nouns that do have signs, especially place names. An automated translation system that doesn't know signs for major place names (e.g. common European countries, or major UK cities) would be odd."

WordNet synsets include multiple subcategories for nouns such as `noun.animal`. `noun.person`, `noun.plant` and many more, including `noun.location`. With a simple modification to the `Word` object class it would be possible to perform a boolean check for place names. Naturally, place names would have to be animated as well to be displayed. An additional resource could also come useful since there are some exceptions, such as Essex which is indeed fingerspelled.

Secondly, an issue solely related to the sign representation and that was previously spotted by a participant in the survey, was the incorrect handshape for possessive pronouns such as *yours* or *his/hers*. The BSL handshape for personal pronouns is a pointed closed fist, however the current animation displays a flat hand, used in ASL. This was ultimately a personal mistake and through more judicious analysis it could have easily been avoided. Nevertheless, since it is only an issue linked to a sign's representation it can easily be mended by changing the animations for those pronouns that use the `POSS` sign.

Another issue pinpointed by Rachel had to do with numbers and quantifier signs. For example, any sign modified by a number such as "three pounds" where pounds is the currency £, should be combined with the number itself to form £-3. Similarly, when describing the age of a person we should expect AGE-22. Currently, our system displays AGE 22 and POUND 33, meaning each sign is performed individually. Unfortunately this problem falls within the same category as classifiers and agreement verbs, where the sign animations must be combined in more complex manners than just playing them concurrently. Since it is not possible, and would be foolish, to animate *all* sign combinations, the system requires additional work to merge complex sign modifiers such as £ and age with quantifiers and number signs.

Moreover, some signs that utilise the data from SignBank are found to be mishandled. As mentioned by Adam, the sentence "How are you?" should result in a single sign $\overset{\text{q}}{\overline{\text{HOW-ARE-YOU}}}$. However because the SignBank step is performed *after* the tree

transformations, some rules may apply to the sentence structure and move words to a position where the SignBank composed words do not match any more. Specifically, "How are you?" currently yields the BSL translation YOU $\overline{\text{HOW}}^{\text{q}}$. The rule that applies here is one covering *Wh*-adverb phrases, where any question starting with a *wh-* word pushes the word at the end of the sentence and marks it with a question tag. While this rule correctly applies to most cases, with HOW-ARE-YOU the SignBank lookup would have to be performed before the tree transformations. Another cause of this inaccurate behaviour is the use of the root replacing the word value i.e. if "How are you" is the source text in the SignBank data, but the root values of each of those words corresponds to "How is you", obviously the match will not happen. But as a result of the flexibility and modularity of the system, this can be corrected easily by moving the SignBank step before the tree transformations, although it would require the system to correctly assign the new single tag to each new aggregated word. Figure 5.6 schematises this problem: since 3 words with different POS tags are combined into one, which POS tag should be assigned to the new word?

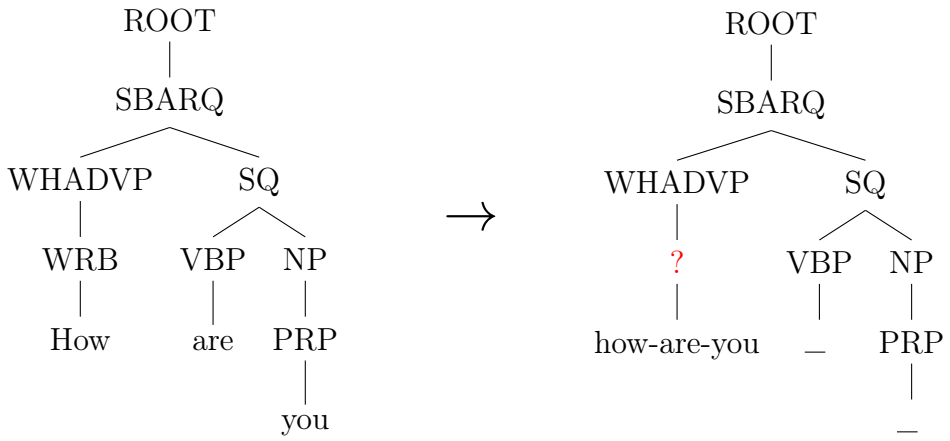


Figure 5.6: Issue with using SignBank before tree transformations

Finally, Adam mentioned the incorrect sign placement of locations in rhetorical sentence constructs such as

I lived in Italy = ME LIVE $\overline{\text{WHERE}}^{\text{q}}$ ITALY (1)

I arrived in London yesterday = YESTERDAY ME ARRIVE $\overline{\text{WHERE}}^{\text{q}}$ LONDON (2)

where the supposedly correct translation should be ME LIVE ITALY for the example (1). In contrast, Rachel argued that the translation with the $\overline{\text{WHERE}}^{\text{q}}$ pattern (the current system behaviour) is indeed correct. It can be assumed that this is because of the differing modes of use of such sentence constructs between BSL signers.

Overall, the two linguists consulted believe the application is an interesting approach to the translation and transmission of BSL through a virtual agent, though it does not properly represent how some Deaf people sign. This is something that can be improved with the inclusion of more rules and further fine-grained analysis to cover specific cases such as the aforementioned fingerspelling for place names. In addition, more work needs to be done to handle classifier predicates and agreement verbs, though the potential approach for this is reviewed in the following chapter.

5.5 Summary

Through an empirical evaluation we compared this system's translation accuracy to that of previous implementations, attentively discussing differences in approaches, preferred metrics and selected corpora. Because no accurate scheme to assess the correctness of animations exists, more subjective evaluations were also employed. The survey feedback was highly assistive in discovering flaws in the user interaction and web interface in addition to understanding missing features or incorrect behaviour of the system. The device/browser specific question gave a good insight in the type of technology people used to access the tool and results demonstrated its high portability and stable performance. The general feedback also helped to get an idea of the naturalness and intelligibility of animations. The more targeted expert feedback disclosed some overlooked imperfections, such as the incorrect use of the possessive signs and fingerspelling for place names. Some of the suggested improvements can be easily achieved since they require only a few modifications to the overall system. However building a large library of signs will take time and additional changes to accommodate more complex sentences as remarked by the linguists will have to be carefully designed and implemented in the future.

Chapter 6

Conclusion

6.1 Discussion

The work described in this report explored the development of a portable web based translation system from written English to British Sign Language. We presented existing techniques in well established topics such as machine translation, 3D animation and modern web programming and attempted to combine them into an innovative approach to sign language translation. We presented an overview of BSL to give an idea of the differences between written and signed English, as well as to support the decisions made when designing the system. A rule-based method allows us to create grammatical transformations to achieve translation between English and BSL, and while more advanced methods exist such as statistical and hybrid approaches, the final implementation demonstrates potential in solving the interlingual problem of translation. With 3D animation techniques a virtual agent displays the animations of the resulting sign language translation in real time. The available media commands to interact with the avatar make it easy to repeat and carefully understand signs, and the additional settings can be used to adjust the playback to the individual's needs. The resulting system was thus developed successfully, incorporating all the necessary features listed in Section 1.2. Through an empirical evaluation we assessed the performance of the translation in terms of accuracy, and with the help of both experts and untrained individuals we asserted the correctness of signing from the virtual avatar and the general usability of the website. While there is vast room for improvement and potential expansion, the system is highly regarded as innovative and useful, especially because of its ease of use and attractiveness which could intrigue individuals into learning more about BSL and therefore increase their understanding of the Deaf community. Being a web application, the information generated is available on any modern device and browser, making it remarkably easy to access and applicable to a

multitude of situations, including learning and instant translations. The adoption of a user-accessible file management approach makes the system highly extensible. Translation rules can be adjusted to improve the translation while sign animations can continuously be added to expand the available library of signs, without the necessity to modify any of the code.

6.2 Difficulties and Achievements

Some obstacles were encountered throughout the duration of this work, ranging from design decisions of the core system, to technical difficulties with the adopted technologies. These challenges forced me to think differently to what I first expected, and I believe the work produced for this thesis would never have been possible had I not planned in advance exactly what needed to be achieved. One of the most challenging aspects was the extensive reading of BSL linguistics, a crucial step required to correctly develop grammar rules. Had the approach been statistical rather than rule-based, it would still have been necessary to understand the underlying mechanics of sign languages, especially since it would have involved the construction of a large corpus as was done in other works. In retrospect, a statistical or hybrid approach would have probably led to better accuracy and covered more possible translations, however would have required much more preparation, potentially undermining the completion of the equally important animation module. Periodically testing methods and ideas in small scale before implementing them in the final product prevented mistakes that could have caused unexpected outcomes. Namely, the use of the JSON format instead of Collada led to smoother animations without the need to build an interpolation engine from scratch. Had Collada not been tested with `ThreeJS` beforehand, more time would have been consumed modifying the code to migrate to JSON.

The skills learnt in the Computer Animation and Computational Linguistics courses were an immense asset that gave me a useful insight when designing the system. This was also my first large scale programming project using Python, something I had only used minimally in the past. It was also my first web application using JavaScript and Flask. It is satisfying to observe the different modules functioning smoothly after completion, especially considering the high complexity of such a system. The extensive knowledge gained by working on this project made me even more interested in web development and I hope to create other equivalently complex and useful applications in the future, as well as continuing to improve this one.

6.3 Future Work

The project completed for this thesis lays the foundations for possible future work to either improve the current implementation or build upon it to extend it with additional features.

6.3.1 Improvements

As was mentioned in the previous section, there is definitely room for improvement. On the linguistics side, many rules are still required to get higher accuracy and cover more sentence constructs. Luckily, the current approach is highly extensible thanks to the standardised annotation format, meaning that rules can be added by simply editing the rule files without needing to modify any code. Additional sign animations to display more gestures can also be added independently.

Currently, the `Special Cases` module still utilises hardcoded rules. To further improve the flexibility of the system, a handwritten rule approach could also be created for it so that procedures that do not fall within the tree transformations or direct translation can be added gradually. Furthermore, many features of BSL still remain untouched. For example **agreement verbs** are not handled by the translation nor the animation modules. One way of solving this would be using a 3D representation of subject and object whenever an agreement verb is identified, similarly to what was done by Huenerfauth's work on classifier predicates [14]. For instance the sentence "I asked you a question" translates to ME-ASK-YOU QUESTION, where ME and YOU are incorporated in the sign. To achieve this, one could create a basic shape for the ASK sign and after determining the entities involved, the direction of the arm would follow that of the subjects. In this example the hand would assume the shape of the verb "ask" and move from the signer ME to the position of the partner YOU. Naturally things become more complicated when covering **classifier predicates**. Following Huenerfauth's approach, any entity that is associated with a specific *group* (see Section 2.1.1 for details on classifiers) can be processed separately from the rest of the signs. As was shown in Figure 5.2, any object involved in a classifier predicate must be represented with a handshape. In this example, the "building" entity is signed with the BUILDING sign and referred to with the (CL-BUILDING) classifier, which is then utilised as default instead of the actual sign for the building entity. Of course, this would involve the development of an entirely separate module to cover these particular constructs, though its inclusion would certainly increase the translation accuracy and variety of sentences the system can handle.

On the other hand, the system could be redesigned with a hybrid approach in mind, using statistical techniques to infer tree transformation and direct translation rules. As a fun but not necessarily significant interaction improvement, a speech recognition module (perhaps using Google's Cloud Speech API[1]) could be set at the beginning of the pipeline to recognise spoken text and automatically translate any recognised sentence. This would be more of an additional feature that users can enable or disable in the settings. Another possible enhancement could be the inclusion of a "storyboard"-style visual feedback of the resulting translation, where each sign, in addition to being displayed as gloss and animation, is also given as an image depicting the sign in the way it is usually shown in sign language textbooks. It would consist of a still image of the avatar with arrows showing the movements of the hands and face. This could be deemed beneficial as some people may be used to this kind of sign visualisation, and its inclusion would reinforce their understanding of the translation. Being a static representation, it can be "read" at the user's own pace rather than at a fixed playback speed. This conceptual visualisation is shown in Figure 6.1a and compared to a sign illustration from `british-sign.co.uk`[2] in Figure 6.1b.

(a) (b)

Figure 6.1: Static illustrations of the sign ANIMAL

6.3.2 Extensions

Given the number of subfields of Computer Science touched by this work, there are quite a few ideas that come to mind for potential related future projects. Some of

[1] `https://cloud.google.com/speech/`
[2] `http://www.british-sign.co.uk/british-sign-language/dictionary/`

these include:

- The creation of a motion capture to JSON file system. Essentially, one could exploit modern animation techniques for recording body and facial movements to generate sign animations. This method was already employed with ViSiCAST [21]. Because motion capture often records movement for all visible joints, the main challenge in this project would be the separation between bones that should be considered *essential* to the sign and those that can be discarded to prevent the incorrect weight distribution with playback of simultaneous animations. The output from this system could then be directly used in our web application.

- A project aimed more towards image processing would involve the automated generation of animation data from pre-synthesised video. An ambitious, yet very interesting concept, it could automatically collect online data from videos of signers (these can be found on websites such as SignBank, SignBSL.com and YouTube) to quickly build a virtual dictionary of signs that can be used in our system.

- Finally, the web application itself could eventually be made into a browser add-on. In the introduction of this report, it was stated that reading text is comparatively more challenging for Deaf than for hearing individuals. Hence, given some text on a random website, one could open the add-on and with a window-in-window technique, the translating avatar would appear as a preview and translate the text being displayed to assist native BSL signers. A rendered conceptual visualisation of this idea is shown in Figure 6.2.

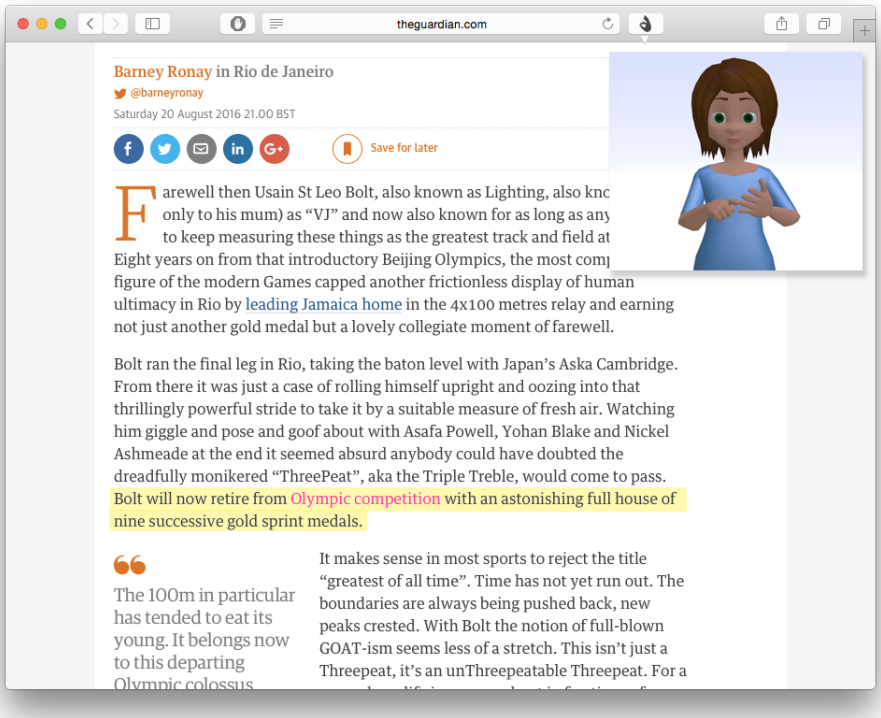


Figure 6.2: Possible browser add-on for the current system. The current sentence is highlighted in yellow and the words being signed (approximated) are coloured in pink.

References

[1] Action on Hearing Loss. Deafness, 2016. [Online and accessed 14-February-2016]. URL: http://www.actiononhearingloss.org.uk/your-hearing/about-deafness-and-hearing-loss/deafness/myths-about-deafness.aspx.

[2] Action on Hearing Loss. Statistics, 2016. [Online and accessed 20-April-2016]. URL: https://www.actiononhearingloss.org.uk/your-hearing/about-deafness-and-hearing-loss/statistics.aspx.

[3] Amy L Baylor. Promoting motivation with virtual agents and avatars: role of visual presence and appearance. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 364(1535):3559–3565, 2009.

[4] Mehrez Boulares and Mohamed Jemni. Mobile sign language translation system for deaf community. Technical report, Research Laboratory of Technologies of Information and Communication Electrical Engineering, University of Tunis, 2012.

[5] Michael Carl. Inducing translation templates for example-based machine translation. Technical report, Institut för Angewandte Informationsforschung, 1999.

[6] Boxing Chen and Colin Cherry. A systematic comparison of smoothing techniques for sentence-level BLEU. *ACL 2014*, page 362, 2014.

[7] Hinrich Schütze Christopher D. Manning, Prabhakar Raghavan. *Introduction to Information Retrieval*, chapter Boolean Retrieval. Cambridge University Press, 2008.

[8] DCAL. Is sign language the same the world over?, 2016. [Online and accessed 14-August-2016]. URL: http://www.ucl.ac.uk/dcal/faqs/questions/bsl/question6.

[9] Marie-Catherine de Marneffe and Christopher D. Manning. *Stanford typed dependencies manual*, 2008. Revised for the Stanford Parser v.3.5.2 in April 2015.

[10] Wanessa Machado do Amaral, Jose Mario De Martino, and Leandro Martin Guertzenstein Angare. Sign language 3D virtual agent. Technical report, Department of Computer Engineering and Industrial Automation, FEEC, University of Campinas, 2010.

[11] Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. What's in a translation rule? Technical report, Computer Science Department,Columbia University, 2004.

[12] Thomas Hanke. HamNoSys - representing sign language data in language resources and language processing contexts. In *Representation and processing of sign languages*, pages 1–6, 2004.

[13] Matt Huenerfauth. A multi-path architecture for machine translation of English text into American Sign Language animation. In *Proceedings of the Student Research Workshop at HLT-NAACL*, pages 25–30, 2004.

[14] Matt Huenerfauth. *Generating American Sign Language Classifier Predicates for English-to-ASL Machine Translation.* PhD thesis, University of Pennsylvania, 2006.

[15] IBM. Say it Sign it, 2007. [Online and accessed 14-February-2016]. URL: `https://www-03.ibm.com/press/us/en/pressrelease/22316.wss`.

[16] Jim G. Kyle and Bencie Woll. *Sign Language: The Study of Deaf People and Their Language.* Cambridge University Press, Cambridge, UK, 1985.

[17] Sue Lewis. Supporting reading within an auditory oral approach. In *International Conference on Deaf Education.* University of Edimburgh, 1997.

[18] Michael W Link, Polly P Armsby, Robert C Hubal, and Curry I Guinn. Accessibility and acceptance of responsive virtual human technology as a survey interviewer training tool. *Computers in Human Behavior*, 22(3):412–426, 2006.

[19] Ariadna Font Llitjoós, Jaime G Carbonell, and Alon Lavie. A framework for interactive and automatic refinement of transfer-based machine translation. In *Proceedings of the Tenth Workshop of the European Association for Machine Translation.* Language Technologies Institute, Carnegie Mellon University, 2005.

[20] Ian Marshall and Eva Safar. Sign language translation via DRT and HPSG. In *Third International Conference, CICLing*, pages 58–68, 2002.

[21] Ian Marshall and Eva Safar. A prototype text to British Sign Language (BSL) translation system. Technical report, School of Information Systems, University of East Anglia, 2003.

[22] Sara Morrissey and Andy Way. Joining hands: Developing a sign language machine translation system with and for the deaf community. National Centre for Language Technology, 2007.

[23] Sara Morrissey and Andy Way. Manual labour: tackling machine translation for sign languages. In *Machine Translation*, volume 27, pages 25–64. Springer, 2013.

[24] Achraf Othman and Mohamed Jemni. Statistical sign language machine translation: from English written text to American Sign Language gloss. In *IJCSI International Journal of Computer Science Issues*, pages 65–73, 2011.

[25] Kishore Papineni, Roukos Salim, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[26] Hrvoje Peradin and Francis Tyers. A rule-based machine translation system from Serbo-Croatian to Macedonian. Technical report, University of Zagreb and Universitat d'Alacant, 2012.

[27] Chandler Prall. requestAnimationFrame is not your logic's friend, 2016. [Onilne and Accessed on 11-August-2016]. URL: `http://www.chandlerprall.com/2012/06/requestanimationframe-is-not-your-logics-friend/`.

[28] Rubén San-Segundo, Verónica López, Raquel Martín, David Sánchez, and Adolfo García. Language resources for Spanish - Spanish Sign Language (LSE) translation. In *Proceedings of the 4th workshop on the representation and processing of sign languages*, pages 208–211. LREC, 2010.

[29] V. M. Sanchez-Cartagena, F. Sanchez-Martinez, and J. A. Perez-Ortiz. An open-source toolkit for integrating shallow-transfer rules into phrase- based statistical machine translation. Technical report, Universitat d'Alacant, 2012.

[30] Felipe Sanchez-Martinez and Mikel L. Forcada. Inferring shallow-transfer machine translation rules from small parallel corpora. In *Journal of Artificial Intelligence Research 34*, pages 605–635, 2009.

[31] Satoshi Shirai, Francis Bond, and Yamato Takahashi. A hybrid rule and example-based method for machine translation. In *Natural Language Processing Pacific Rim Symposium*, pages 49–54, 1997.

[32] Sign BSL. BSL sign language dictionary, 2016. [Online and accessed 30-July-2016]. URL: `http://www.signbsl.com`.

[33] Signature. British Sign Language, 2016. [Online and accessed 30-July-2016]. URL: `http://www.signature.org.uk/british-sign-language`.

[34] William Stokoe. The sign structure: an outline communication systems of the American Deaf. Technical report, University of Buffalo, 1960.

[35] Stuart M. Thiessen. A grammar of SignWriting. Master's thesis, University of North Dakota, Grand Forks, North Dakota, 2011.

[36] Yuan Tian and David Lo. A comparative study on the effectiveness of part-of-speech tagging techniques on bug reports. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 570–574. IEEE, 2015.

[37] UCL. BSL sign bank, 2016. [Online and accessed 30-July-2016]. URL: `http://bslsignbank.ucl.ac.uk/`.

[38] Tony Veale and Alan Conway. Crosss modal comprehension in ZARDOZ, an English to sign-language translation system. In *7th International Generation Workshop*, pages 249–252, 1994.

[39] Tony Veale, Alan Conway, and Brona Collins. The challenges of cross-modal translation: English to sign language translation in the ZARDOZ system. In *Machine Translation*, volume 13, pages 81–106. Kluwer Academic Publishers, 1998.

[40] Chung-Hsien Wu, Hung-Yu Su, Yu-Hsien Chiu, and Chia-Hung Lin. Transfer-based statistical translation of Taiwanese Sign Language using PCFG. 6(1), 2007.

[41] Liwei Zhao, Karin Kipper, William Schuler, Christian Vogler, and Martha Palmer. A machine translation system from English to American Sign Language. Technical report, University of Pennsylvania, 2000.

Background Reading

Deaf Solutions 3. *BSL 320. Linguistics Exam.* Jan 2016.

CG Cookie. Blender: Introduction to Character Rigging, 2016. [Online; accessed 20-April-2016]. URL: `https://vimeo.com/33551536`.

Daniel Jurafsky and James H. Martin. *Speech and Language Processing.* Pearson Education Ltd, Essex, UK, 2014.

Tony Parisi. Programming 3D Applications with HTML5 and WebGL, 2014.

Signature. *Teacher Notes: Level 3 Certificate in British/Irish Sign language.* Sept 2013 - Aug 2014.

Rachel Sutton-Spence and Bencie Woll. *The Linguistics of British Sign Language: An Introduction.* Cambridge University Press, Cambridge, UK, 1999.

Appendix A

Translation rules

Tree transformations

```
NP -> PDT <> | NP -> _ <>
// very specific case
NP -> <> JJ NN~ CC <> NN~ | NP -> NN~ <> JJ CC NN~ <>
NP -> <> JJ JJ NN~ | NP -> <> NN~ JJ JJ
NP -> DT <> NN~ | NP -> NN~ DT <>
NP -> JJ NN~ | NP -> NN~ JJ

NP -> DT RB JJ NN~ | NP -> NN~ DT RB JJ
NP -> RB JJ NN~ | NP -> NN~ RB JJ
NP -> <> JJ NN~ | NP -> <> NN~ JJ
NP -> <> ADJP NN~ | NP -> <> NN~ ADJP

// will cover anything like "has died", "was born" i.e. auxiliaries
VP -> VB~ <> VP | VP -> _ <> VP
// covers "He was sick", "He is tall" etc
VP -> VB~ ADJP <> | VP -> _ ADJP <>
VP -> TO <> | VP -> _ <>

PP -> TO NP NP | PP -> _ NP NP
PP -> TO NP | PP -> _ NP

S -> NP ADVP VP | S -> NP VP ADVP

SQ -> VB~ <> | SQ -> _ <>  // removes the "have", "did" etc in questions
SQ -> MD <> | SQ -> _ <>

// handles direct questions introduced by a wh- word or wh-phrase
SBARQ -> WH~ <> | SBARQ -> <> WH~

SINV -> MD <> | SINV -> _ <>

WHNP -> WDT NN~ | WHNP -> NN~ WDT   // handles "which book..."
WHNP -> WHNP <> | WHNP -> <> WHNP
```

Listing A.1: tree_transforms.txt

Direct translation

```
SWAP
if -> if // moves the sentence after the if before the cause
WORDS
there be -> _ _           // removes the existential there + is
if it -> _ _            // removes the it in SBAR
have you -> _ you         // remove any 'have you' questions
time last -> past time
year old -> _ age
time what -> what time
happy year new -> happy new year
year next -> next-year _
week next -> next-week _
day next -> next-day _
year last -> last-year _
week last -> last-week _
as much as -> same _ _
DT
a->_
an->_
some->_
this->ix
those->ix
that->ix
these->ix
the->_
PRP
i->me
he->ix
him->ix
she->ix
her->ix
it->_
us->we
they->them
POS
's->_
'->_
IN
if->_
because->why
as->_
MD
will->_
NNS
parent->m-f-
NN
mom->m-m-
mum->m-m-
mother->m-m-
dad->d-d-
father->d-d-
CC
```

```
and->_        // remove and conjunction
->dollar
RB
o'clock->time
VB
be->_
VBD
be->_
VBZ
be->_
VBG
be->_
VBP
be->_
```

Listing A.2: direct_translation.txt

SignBank multi-words

```
about time
above all
add up
adhesive tape
adultery adulterous
air force
alarm bell
alcoholic drink
all day
// ... 680 more lines
will do
wind someone up
wipe out
work hard
work out
world record
worn out
would not
year before
```

Listing A.3: signbank_multi.txt

Appendix B

Additional algorithms and Code

B.1 Data Extraction from the Stanford Parser

Upon launching the web-app, a `Parser` object is created in Python and depending on the availability of the online interface of the Stanford Parser, the object will either act as an HTML scraper on the interface's source, or create a local version of the parser (Listing B.1).

```
from nltk.parse.stanford import StanfordParser, StanfordDependencyParser

class Parser:
    def __init__(self, app=False):
        # first check accessibility of stanford website
        status_code = 0
        try:
            initRequest = requests.post(
              "http://nlp.stanford.edu:8080/parser/index.jsp",timeout=5)
            # test that the parser still available
            status_code = initRequest.status_code

        except requests.ConnectionError:
            print "Not online, using offline parser"
        except requests.exceptions.Timeout:
            print "Response too slow, using offline parser"

        if (status_code == 200): # online
            self.online = True
        else:
            self.online = False

            # setup necessary variables to create local stanford parser
                os.environ['STANFORD_PARSER'] = APP_PARSER
                ...
            # setup tree parser
            self.parser = StanfordParser()
            # setup dependency parser
            self.dependency_parser = StanfordDependencyParser()
```

Listing B.1: Parser object creation from the Stanford implementation

Because each version yields the same data but in a different format, the `parse` method executes the correct code for each situation. As can be seen on line 7 in Listing B.3, when the Stanford Parser is online we use the `BeautifulSoup` library to scrape the HTML page where the result of the request is displayed. From there we obtain the 3 essential objects: the tagged sentence (line 10), the parse tree (line 12) and the dependencies (line 14). On the other hand, if we are using a local version then the methods differ as the parser will return objects instead of raw text. These objects have to comply to the same format, thus they are altered accordingly.

```
1  def parse(self, sentence):
2      if(self.online):
3          r =requests.post("http://nlp.stanford.edu:8080/parser/index.jsp",
4              data={'parse': 'Parse',
5              'parserSelect': 'English',
6              'query': sentence})
7          soup = BeautifulSoup(r.text, "html.parser")
8          tagging = ' '.join(soup.find("div",
9                      {"class": "parserOutputMonospace"}).text.split())
10         parseTree = ' '.join(soup.find_all("pre",
11                     {"class": "spacingFree"})[0].text.split())
12         parseTree = nltk.Tree.fromstring(parseTree)
13
14         dependencies = soup.find_all("pre",
15                     {"class": "spacingFree"})[1].text.split('\n')
16         return [tagging, parseTree, dependencies]  # complete result
17     else:
18         parseTree = list(self.parser.raw_parse(sentence))[0]
19
20         dependencies_res = self.dependency_parser.raw_parse(sentence)
21         dependencies_res = dependencies_res.next()
22         dependencies = list(triples_alt(dependencies_res))
23
24         leaves = list(parseTree.subtrees(lambda t: t.height() == 2))
25         tagging = ''
26         for pair in leaves:
27             # remove brackets then split tag/word
28             txt = str(pair).strip("()").split()
29                 tagging += txt[1]+'/'+txt[0] + ' '
30             tagging = tagging[:-1]
31             return [tagging, parseTree, dependencies]
```

Listing B.2: Obtaining the data from the remote or local parser

B.2 HTML scraper for the SignBank data retrieval

```
url = 'http://bslsignbank.ucl.ac.uk/dictionary/search/?'
query = 'query='
page = '&page='
file = open('signbank_multi.txt', 'w')
for letter in ascii_uppercase:
    index = 1
    prevWords = ''
    while(True): # until we haven't found all the words for this letter
        html = requests.get(url+query+letter+page+str(index))

        soup = BeautifulSoup(html.text, "html.parser")
        table = soup.find('div', {'id':'searchresults'})

        soup = BeautifulSoup(table.text, 'html.parser')
        words = filter(lambda x: len(x) > 0,soup.get_text().split('\n'))

        if words == prevWords:
            break
        print words
        prevWords = words

        for w in words:
            # we only save those words that make up 1 sign
            if len(w.split(' ')) > 1:
                file.write(w+'\n')
        index += 1
file.close()
```

Listing B.3: Python code for parsing the SignBank website

B.3 Main animation sequence loop

Algorithm B.1 Clips animation sequence

Require: fade_counter is a variable to access individual clip objects in the clip list
Require: mixer is an object that directly modifies the clips for fading, playing etc.
Require: URL.manual contains the data to load clips (index, name, path)

```
 1:
 2: function playSequence
 3:     clip ← clips[fade_counter]
 4:     if first_step is true then
 5:         mixer.Set(clip)                                  ▷ Loop, reset and play clip
 6:         modifierLoop(fade_counter)
 7:         fade to next clip
 8:
 9:         fade_counter ← fade_counter + 1
10:         first_step ← false
11:         continuous_step ← true
12:
13:     if continuous_step is true then
14:         if URL.manual length > 1 then
15:             if clip.time > clip.duration then
16:                 pause clip
17:
18:                 if clip.name = next clip.name then
19:                     change next clip.name
20:                 mixer.Set(clip)
21:                 modifierLoop
22:                 checkCompounds
23:                 fade to next clip
24:
25:                 if fade_counter = URL.manual length then          ▷ Sequence end
26:                     continuous_step ← false
27:                     final_step ← true
28:                 fade_counter ← fade_counter + 1
29:         else
30:             continuous_step ← false
31:             final_step ← true
32:     if final_step is true then
33:         if clip.time > clip.duration then
34:             pause clip
35:             mixer.Reset
36:             fade to init clips                                    ▷ Idle clip
37:
38:             reset HTML gloss
39:             final_step ← false
```

Appendix C

Screenshots and Project Structure

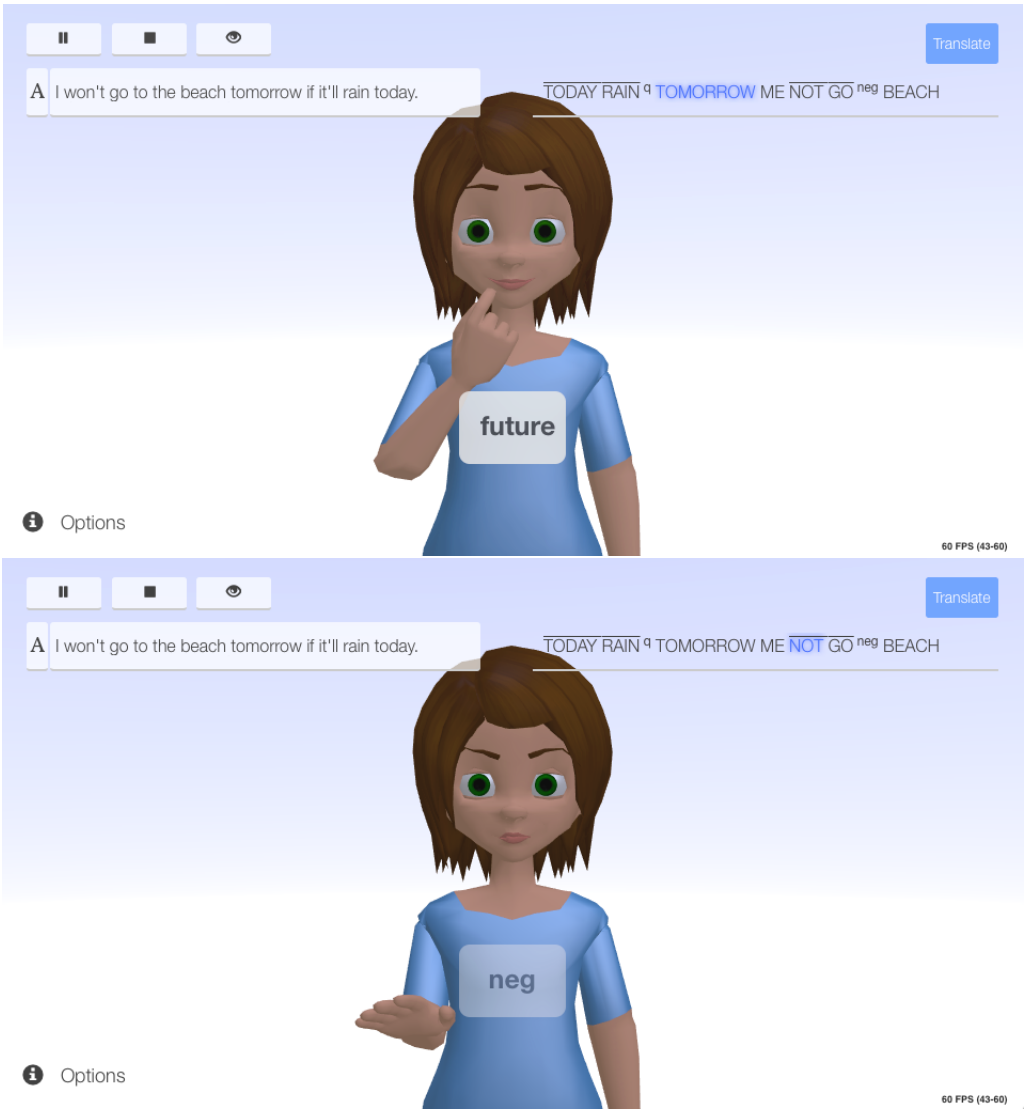


Figure C.1: "Show non-manual features" enabled playback

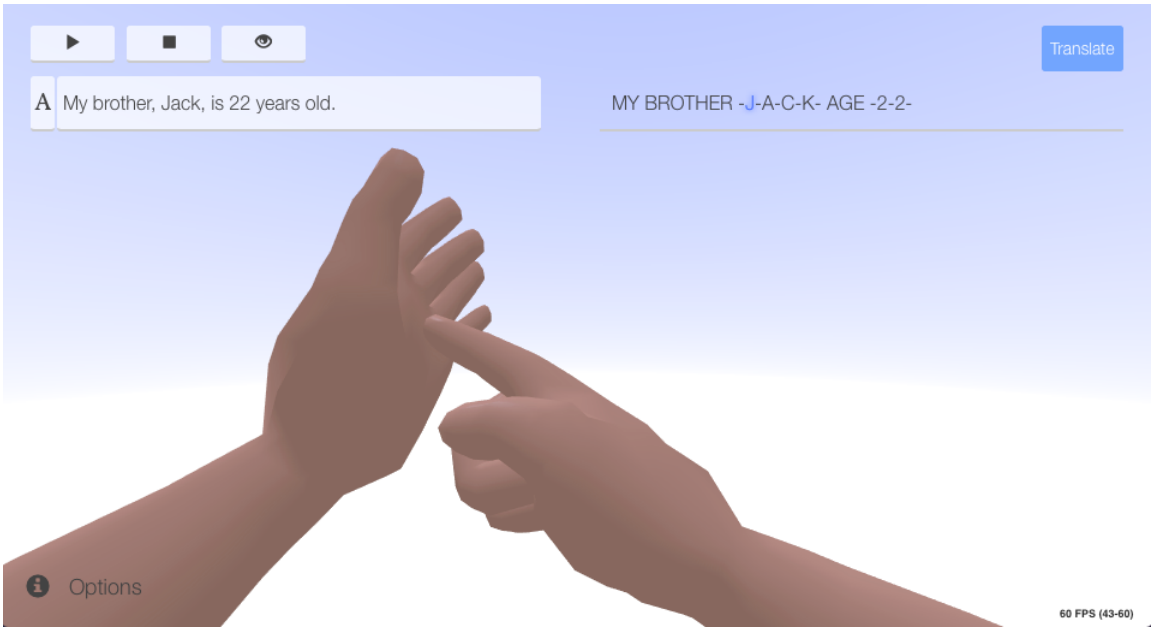


Figure C.2: First person view of the avatar when fingerspelling the letter "J"

Figure C.3: 3rd person view at the same time as what is displayed in Figure C.2

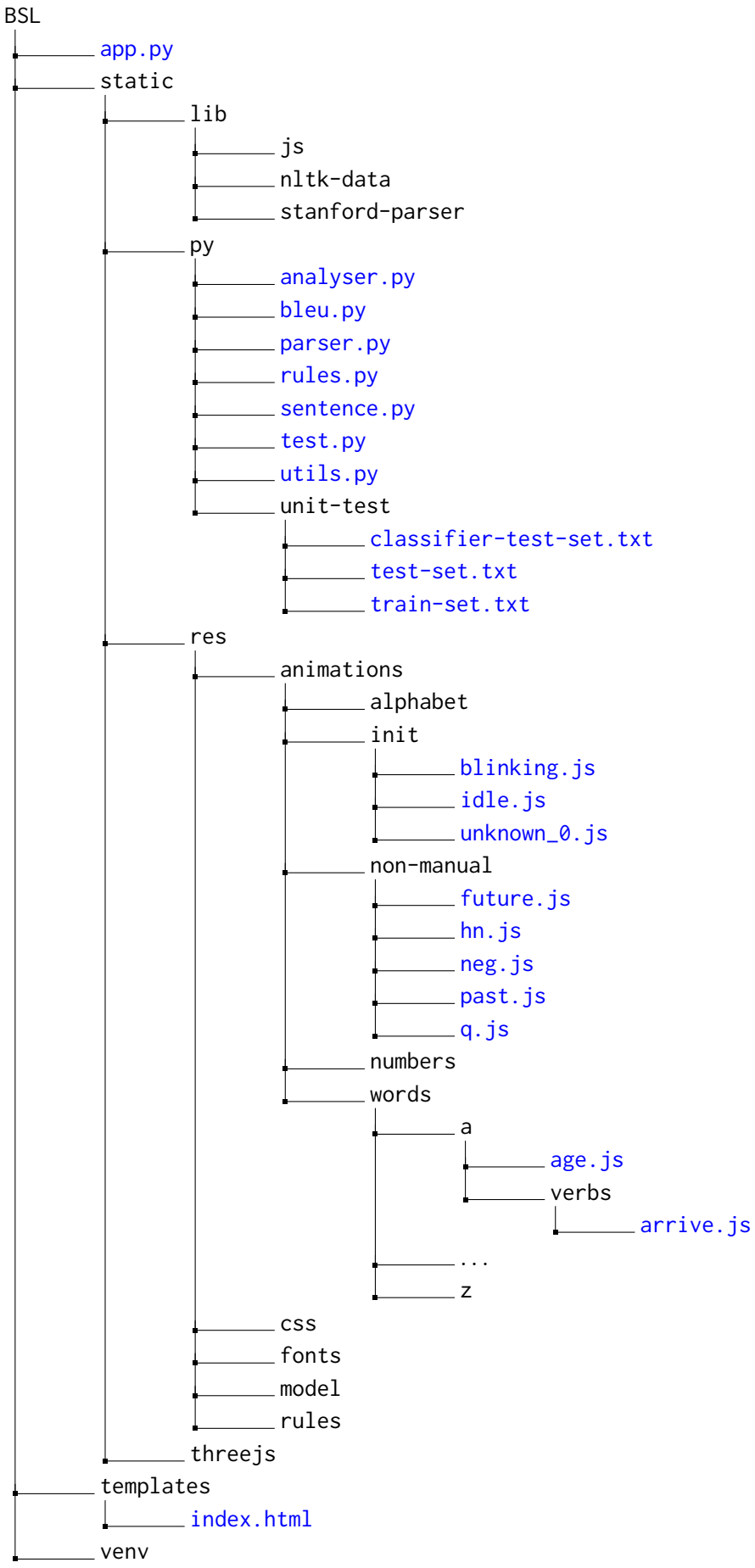


Figure C.4: File structure of the project; files are highlighted in blue

Appendix D

Survey Questions

1. **What is your level of knowledge of sign language (BSL, ASL or equivalent)? Choose one.**

   ◯ No experience at all

   ◯ Limited knowledge (Know a few signs and/or the alphabet)

   ◯ Level 1-2 (Enough skills to communicate with Deaf people in a range of everyday situations)

   ◯ Level 3-4 (Enough skill to interact on a regular basis with Deaf people in a work environment)

   ◯ Level 6 (Enough skill to work professionally with Deaf people e.g. teachers in BSL, translators etc.)

   If you answered No experience, have you ever been curious about BSL or Sign languages in general?

   ◯ Yes

   ◯ No

At this point the users are directed to the website `translate.nicmosc.com` where the application is hosted. They are given some guidelines on what is available and a short introduction to the application in general.

2. **Did you notice any bugs? If no skip this question. Otherwise choose any of the following.**

   ☐ Website did not load

☐ The spinning wheel does not stop and no animation is shown

☐ The animations are laggy (frames per seconds are low)

☐ The website crashed

☐ Other: .....

3. **Please specify the device and browser you are currently using. If you found bugs and switched browsers/device, select the former. Please click only 1 of the following.**

	Safari	Chrome	Firefox	Opera	Internet Explorer
Mac computer	◯	◯	◯	◯	◯
Windows computer	◯	◯	◯	◯	◯
Linux computer	◯	◯	◯	◯	◯
iPhone	◯	◯	◯	◯	◯
Android phone	◯	◯	◯	◯	◯

4. **The interface buttons are clear and it is easy to understand what they do. To what extent do you agree with this statement?**

◯ Strongly agree

◯ Agree

◯ Neither agree or disagree

◯ Disagree

◯ Strongly disagree

5. **Does the avatar come across as friendly?**

◯ Yes

◯ No

◯ Kind of

6. **If you answered "No" or "Kind of" above, why do you think so?**

.....

7. **How would you rate the movements of the avatar?**

	1	2	3	4	5	
Very unnatural	◯	◯	◯	◯	◯	Very natural

8. **Do you think the avatar should be customisable? If so select which options you would like to see. Otherwise skip this question.**

   ☐ Gender

   ☐ Colours (skin, clothes, hair etc.)

   ☐ Body features (height, weight, arms length etc.)

   ☐ Other .....

9. **How did you feel about loading times?**

   Please note that depending on your internet connection and other factors (such as the machine you are using) may affect this. Also the first translation will always take longer than the following ones.

   ◯ Very short

   ◯ Short

   ◯ Reasonable

   ◯ Long

   ◯ Too long

10. **You were able to follow the animation along with the text output and notifications. To what extent do you agree with this statement?**

    ◯ Agree

    ◯ Disagree

    ◯ Other: .....

11. **If you feel that you have enough experience to comment on the accuracy of the translation (in terms of adequacy and fluency), please do so below.**

    Adequacy is a rating of how much information is transferred between the original and the translation, and fluency is a rating of how good the translation is.

    .....

12. **Would you recommend this website to someone curious about sign language?**

    ◯ Yes

    ◯ No

    ◯ Don't care

13. **Why?**

    .....

14. **Would you recommend this website to someone who is learning BSL?**

    ◯ Yes

    ◯ No

    ◯ Don't care

15. **Why?**

    .....

16. **Finally, what did you like about the application?**

    .....

17. **What did you dislike?**

    .....