Flinders University
College of Science & Engineering

ENGR9700
Masters Thesis

Sign-to-Text

Putting the 'language' into Sign Language Recognition

David Mansueto

Supervisors

David Hobbs & Trent Lewis

November 2019

Submitted to the College of Science and Engineering in partial fulfilment of the requirements of the degree of Bachelor of Engineering (Biomedical) (Honours), Master of Engineering (Biomedical) at Flinders University Adelaide Australia

Declaration

I certify that this work does not incorporate without acknowledgement any material previously submitted for a degree or diploma at any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person except where due reference is made in the text.

Signed: *David Mansueto*

Date: *November 2019*

Acknowledgements

My first and foremost acknowledgement must go to my principle supervisor Dr. David Hobbs for his enthusiasm, guidance and steadfast support during this journey and my co-supervisor Dr. Trent Lewis for his grounding cynicism.

Thank you to Jenna Mayne and Debbie Kennewell of Deaf Can:Do for proposing this topic and sponsoring me to attend their Auslan courses "Talking Hands" 1, 2 & 3.

And thank you to A/Prof. Kenneth Pope for the vitalising lunchtime games of ping-pongo.

Last but certainly not least, thank you to Dee for getting me through a thesis that seemed like it would never end.

Abstract

The Sign-to-Text project explores the challenge of sign language recognition (SLR), in the context of a system to recognise Auslan (Australian Sign Language) and translate it into English text on a computer. This project was borne of a request from the Deaf Can:Do group. Workers converse with deaf community members in Auslan, their mutual first language, then must enter case-notes in English a second language with different vocabulary and grammar.

SLR is ostensibly a well-studied field nearly as old as the first linguistic definition of Auslan and has two known commercial solutions, neither of which meet the needs of Deaf Can:Do. A thorough grounding of Auslan linguistics allows defining the components that transform a series of gestures into a rich, expressive language. Auslan is a complex combination of visuo-temporo-spatial cues, including the well-accepted but poorly-clarified phonemes the irreducible, contrastive components of a lexeme and the more elusive contextual elements, such as classifiers, modifiers, mime and the allocation of nouns to spatial locations for deictic reference. A new taxonomic linguistic structure for Auslan that includes all these elements is presented. The intrinsic challenges signed languages present for recognition and translation are defined, including several new challenges.

Recognition begins by observing signing such as via instrumentation of the signer or optical image capture. As non-manual elements are essential for language, an optical input is currently required for true sign *language* recognition, however instrumentation typically provides far higher fidelity suited to the higher complexity of manual elements.

A framework for sign language recognition and translation is proposed. A modular approach encourages multi-modal input. Taking cues from speech recognition, SLR is divided into a visual model that classifies individual phonemes and modifiers and a language model that considers the unfolding sentence as it combines these into lexemes. The integration modifiers, spatial referents and deixis locations are facilitated by additional classifiers and a new memory block within the visual model. The output lexemes including context and modification would be glossed by the language model, completing the recognition stage, leaving a final translation stage into the target language.

Using consumer-grade depth cameras, the framework is implemented up to phonemic recognition (handshape), providing insight into the technical challenges faced by optical sign-language recognition systems. A verification of the system using 5 handshapes and classifying with a neural network achieved 87 % accuracy.

Contents

Figures

Tables

Listings

Glossary

Auslan the sign language (SL) of the Australian deaf community; a contraction of Australian Sign Language [59]. 1 6, 9, 26

Creative BlasterX Senz3D a consumer grade structured-light the combination of colour (specifically, RGB) and depth images (RGB-D) camera that uses an Intel RealSense depth module and is supported by the Intel RealSense software development kit (SDK) 1.0. 39 41, 65

Deaf Can:Do a charitable service provider formed by the joining of The Royal South Australian Deaf Society and the Can:Do group. 1, 2, 4

deixis a contextual extralinguistic reference by means of expression. 9, 10

depth (of a camera) the distance from the camera sensor to the scene, obtained through optical measurement; typically taken as the Z-axis, originating orthogonally to sensor and extending positively towards the scene. xvii, xviii, 27

gloss encoding of a visual gesture into written form. See Appendix A for sylistic conventions. 21

Intel RealSense D435 a consumer grade stereoscopic RGB-D camera. xi, xvii, 34, 39, 41, 42, 45, 50, 51, 54, 56

Intel RealSense D435i a variant of the Intel RealSense D435 that differs only by the addition of an inertial measurement unit (IMU). xi, 41 43, 45, 52, 57, 62, 66

Intel RealSense SDK 2.0 the SDK for the Intel RealSense D400 series depth modules. xv, 43, 45, 47, 48, 65

joint (of a skeleton model) an intersection point between rigid segments where flexion can occur *or* a point of interest (e.g. 'hand' or 'head'), typically given as coordinates in two-dimensional or three-dimensional space. 27

lexeme the minimal units of language, e.g. signs and words, *per se*, divorced from their meaning. Stylised as LEXEME. xviii, 3, 6, 7, 11, 19, 21

lexicon the complete set of lexemes in a language. 2, 6, 23

manual related to the hands. 3, 6, 7

Microsoft Kinect 2.0 also known as 'Microsoft Kinect for Xbox One', a consumer-grade time-of-flight depth camera. xv, 39 41, 66

phoneme base components of a word or sign. Stylised as */phoneme/*. 6, 7, 11, 13, 19, 21

RASR The RWTH Aachen University open source speech recognition system [113]. 30

stereoscopic an optical depth-computation technology that uses trigonometry to estimate depth from the images produced by a (stereo) pair of sensors. 42

structured light an optical depth-computation technology that estimates depth based on distortion of a unique pattern, projected by an infra-red (IR) projectector, reflected of the scene and detected by an IR sensor. 34, 41

time-of-flight an optical depth-computation technology that estimates depth from the time for pulses of IR light to reflect of the scene and return to the camera. 34, 41

Chapter 1

Introduction

SLR is the use of a machine to decipher gestural language. The first documented attempt dates back to 1986 [95] and recent patents [33], [110] and commercial products [32], [111] would seem to suggest that such systems are now technologically viable.

The South Australian community support group Deaf Can:Do have sought such a system to facilitate the production case-notes in their case-worker's native language: Auslan, rather than type them in directly in English. While simply using a keyboard and typing might be the 'obvious solution', there are limitations.

SLs are unique, natural languages, not 'gestural equivalents' of spoken languages, and generally do not have a written form. This creates difficulty interacting with a computer: for example, a hearing-impaired individual (PRO_1[1]) who's first language (L1) is Auslan may correspond with another individual (PRO_2) who is fluent in Auslan using video chat, but if video chat is not an option, PRO_1 must resort to written communication and thus a different language.

Native Auslan speakers are typically hearing impaired and have never heard nor spoken English — the classic counter example, a child of deaf adult (CODA), is not common — so while the hearing impaired individual may learn English, it will most likely be through reading and writing and therefore as a second language (L2). As Auslan and English are dissimilar in lexicon and grammar, a native Auslan speaker typing in English can be likened to a non-hearing impaired individual dictating to a computer in a spoken L2 with different sentence structure to their L1.

Deaf Can:Do have been unable to find a suitable SLR system for Auslan and so suggested this project. The aims of this project are provided in Table 1.1.

[1]Linguistic conventions such as PRO_{3a} and POSS_2 are defined in Appendix A.

Table 1.1: Aims of the Sign-to-Text project.

#	Aim
1.	To define the essential linguistic components of Auslan.
2.	To develop a framework for SLR that incorporate those components.
3.	To implement the framework for automatic recognition of a restricted Auslan vocabulary for Deaf Can:Do case notes.

1.1 Background

The background of this thesis, like much of the thesis itself, is presented in two sections, each considering one of the two sides of SLR: first, linguistics, the study of (sign) language and identification of the essential aspects that can then be targeted by the second side: recognition of signs automatically by a machine.

1.1.1 Signed Languages

SLs are natural languages, they have a defined set of rules: 'grammar' and a defined set of signs: 'lexicon' and, importantly, are *not a gestural version or mime of spoken language* [10], [24], [59], [117]. Further, there is no one 'universal' or international SL, but rather a deaf community in isolation will tend to develop their own, much like spoken language [59], leading to national SLs as well as regional dialects.

SLs have only been recognised as 'true' languages since the 1960's [59], following a disruptive paper on American Sign Language (ASL) by linguist Stokoe [117] that prompted global interest in and "serious" linguistic study of SLs. Since then there have been several efforts [61] to analyse and quantify SLs including through the production of lexicons and dictionaries[2]. Foremost in Australia are the works of Johnston, who coined the term 'Auslan' and developed a lexicon using a hierarchical classification structure, extending it to form an Auslan Dictionary [59] that has been maintained through several forms, migrating to video recordings of signs [63] and eventually into an open-access online format [62] with around 8000 video-based definitions.

Auslan differs from English both in vocabulary: the set of available words and in grammar: the rules of how words are combined to form language. An example of the

[2] A trend in earlier SL lexicographical works is a seeming arbitrariness in the use of the terms 'lexicon' (defined as a complete set of words, without definitions) and 'dictionary' (a complete set of words, with definitions), likely due to differences in opinion – with some pointed references [61]

first are the distinct lexemes in Auslan for three different meanings of the single English lexeme 'party': (birthday) party, (go out and) party and (e.g. a political) party. An example of the latter is the order of words in sentences, fixed in English as: subject noun, verb, object noun; while Auslan has no fixed order but tends to begin with context, such as: actor, verb, undergoer, constituent [24], [64].

This thesis focuses on the engineering aspects of SLR and so avoids the greater grammatical complexities of SLs, delving deep enough only to define the linguistic elements required for unambiguous recognition. These elements are discussed at length in Chapter 2.

1.1.2 Automatic Sign Language Recognition

SLR can be viewed as the natural interaction of a signer with a computer: the computer takes the place of PRO_2 in (unilateral) conversation, much like a speaker may dictate to computer using automatic speech recognition.

There are two main approaches to SLR: instrumented and optical, each with their advantages and disadvantages. Instrumentation, such as the application of sensors to measure joint angles of the hand or tracking the location of the hand in space are highly informative for manual elements but tend to be more difficult to set up, may restrict the signers' ability to sign and, crucially, simply cannot observe non-manual elements (NME). Optical systems tend to be easier to setup, are capable of observing the entire signing space and grant the signer the freedom they are used too, but provide lower quality information thus requiring greater classification effort.

An option for optical systems is to use more cameras, providing more views and so information while potentially reducing obstruction. Hybrid systems that utilise both modalities, such as instrumentation for manual elements and optical systems for NME present a powerful concept, but may still impede the signer.

In the past decade cameras that measure distance from the camera ('*depth*') as well as colour (thus RGB-D) have become available at low cost. The addition of depth directly provides a third dimension of information for optical classification, as well as providing a simple, robust means to isolate, for example, a hand, from the background.

Given the limitations of instrumented approaches, the availability of depth cameras and recent progress in computer vision, a purely optical modality was selected for this study. The inherent challenges of an optical SLR system and the state-of-the-art for computer vision techniques are discussed in Chapter 4.

1.1.3 Gap Analysis & Focus of this Thesis

Developing an automated system to convert sign language into text is a non-trivial task. Efforts in SLR date back to the 1980's and there are now commercially available solutions, yet none fit the needs of Deaf Can:Do. Notably absent is a recognition framework that takes a linguistic perspective to ensure it recognises all aspects that make sign language a language, rather than just implementing sign recognition. As such, this thesis focuses on defining the linguistics of signed languages, exploring existing recognition techniques and producing a holistic sign *language* recognition framework.

The linguistics of signed languages and of Auslan in particular are covered in Chapter 2, providing insight into foundations of SL and the elements required to capture them.

A literature review of the techniques used for the automatic recognition of signed languages, both optical and instrumented, is presented in Chapter 4, along with research into the particular challenges faced by optical systems.

With the requirements and techniques covered, Chapter 3 outlines an inclusive framework for SLR and touches upon the extension of it to translation, which is not a focus of this work.

A rudimentary implementation of the framework is detailed in Chapter 5, taking an optical approach to static handshape recognition.

A validation of the implementation by means of a small trial is presented in Chapter 6.

The learnings from this thesis are summarised in Chapter 7.

Chapter 2

Linguistics

To achieve the goal of SLR it is necessary to first understand the SL in order to recognize it. The contemporary linguistics of signed languages is a relatively young field, with interest revived in the 1960's [117] following a hiatus since the eighteenth century linguists, who's efforts were largely forgotten [64]. Auslan itself is a young language: it is estimated to have originated in the nineteenth century and has only been linguistically defined in the past three decades [59].

There are many SLs, each with their similarities, differences and nuances, but there does not appear to be any meta-study of these variations. The unspoken consensus of SLR researchers seems to be that a system which proves effective for a particular SL could reasonably be adapted for use with other SLs.

2.1 Sign Types

Signed languages contains four 'sign types': lexical signs, classifier signs, sign-mime and mime [59].

Lexical signs are the 'words' of a SL: they have a defined meaning and a prescribed, unambiguous form which is discussed in detail in the following section.

Classifier signs are part of SL context and provide visual adjectives and emphasis. *Descriptive* classifiers are where the hands 'trace over' an imaginary object, describing its size and shape. *Proform* classifiers are where the hands represent an object *per se* and perform its position, orientation and motion.

Sign-mime is an improvised, context-dependent sign used in the place of lexical signs when an individual doesn't know an established sign, or when there is no defined sign, such as for niche, technical and compound terms. Mime is simply 'acting out' a scenario

and discouraged in Auslan except for the use of 'visual quoting'.

2.2 Phonological Structure

A lexicon is the set of all high-level units of a language, the lexemes. Lexemes, in turn, are comprised of 'phonemes': the basic, irreducible, contrastive units of language [10]. Early sign linguistic works espoused Stokoe's term for the sublexical unit: 'chereme', an anagram for and modelled on phoneme, using the Greek base (*χειρ* (*cheír*): "hand")[117]. Recent works take the view that signed language should use generic linguistic terms, rather than special SL-specific terms [10], [24], [125]. There are, however, recent works that buck this trend, for example, using 'viseme' in the place of lexeme [73], presumably a contraction of 'visual' and 'lexeme'.

Phonemes in Auslan include, for example, a */closed-dominant-hand-with-the-index-finger-out-and-hooked/*, a */double-tap/* motion of the dominant hand, a location of the dominant hand relative to the body, for example, */bridge-of-the-nose/*. The Auslan speaker will recognise these phonemes as part of the set that form the lexeme SISTER. If one of these phonemes were to change, for example, the location from */bridge of the nose/* to */chin/*, the lexeme would change; in this case, to DINNER.

These three phonemes fit the phonological categories defined by Stokoe in 1960: configuration, position and motion, but with 'significant combination' names: *designation*, *tabulation* and *signation*, respectively,[117] and since commonly used in abbreviation: 'Dez', 'Sig', 'Tab'.

A fourth 'significant combination' *hand-arrangement* 'Ha' can be added to this list [12], [25] a structural choice Stokoe opposed, viewing arrangement as a subcategory of 'position' (using the term 'attitude') [59], [117]. As an example by juxtaposition, consider MINE and YOURS: both are formed by the */closed/* shape, start in the */neutral/* location and thrust in the direction of the palm, but in MINE the palm faces the signer (PRO_1), while in YOURS the palm faces PRO_2.

In his 1989 thesis, Johnston identified five core elements of a sign: handshape, location, orientation, movement and expression [59]. The first four match those previously described with 'hand-arrangement' becoming *orientation* which is perhaps less illustrative and can be collectively referred to as the manual elements. The fifth category, *expression*, acknowledges that SL is more than hand-waving [59]; a view not commonly held at the time [34] but seemingly taken as 'common knowledge' now [17], [18], [24]

The importance of expression can be demonstrated by */flat hand, digits together/*,

/dominant hand on chest/, */palm facing chest/* and */no movement/*: the addition of an */enthusiastic/* (facial) expression and */nodding/* head creating LIKE, while */stern/* expression and */shaking/* head create DON'T LIKE.

This also shows that expression itself can be considered a list of subcategories, including body posture, head movements, facial expression, gaze direction, eye and eyebrow movements and mouthings [24], [59], [73], collectively referred to non-manual elements (NME): 'everything other than the hands'.

The 'classical' five phonological categories that combine to form a single sign are [10], [24], [59], [125]:

Shape (previously *Designation* 'Dez') shape of the dominant (and subordinate) hand(s).

Location (previously *Tabulation* 'Tab') place or position of articulation of the hand(s) relative to the body.

Orientation (previously *Hand Arrangement* 'Ha') orientation of the dominant (and subordinate) hand(s)

Movement (previously *Signation* 'Sig') the dynamic action or articulation of the hand(s).

Expression or NME, everything other than the hands, including body posture, head gestures, facial expressions and gaze direction.

Each of five phonological categories include many possible phonemes; Johnston created a *detailed decision schema for "Cheremic Order"* [61], shown in Figure 2.1 with the accompanying notes shown in Figure 2.2, to illustrate how the phonological (then, cheremic) categories combine to form lexemes. The first phonological category is the shape of the dominant hand, listed as "Cheremic Order", of which Johnston enumerated 62 phonemes in 1989 and 60 phonemes in 2003 [59], [61].

2.2.1 Locative Terms

An important consideration when discussing locations and orientations in signed languages is both the frame of reference and validity of sublexical combinations, particularly in the case of handedness. It is common for texts to use terms such as 'towards the left', or 'the right hand', both suggesting a requirement for right-handed signing which is not the case, at least for Auslan. If the signs are mirrored appropriately, that is, become 'towards the right' and 'the left hand', respectively, the sign is interpreted the same. An

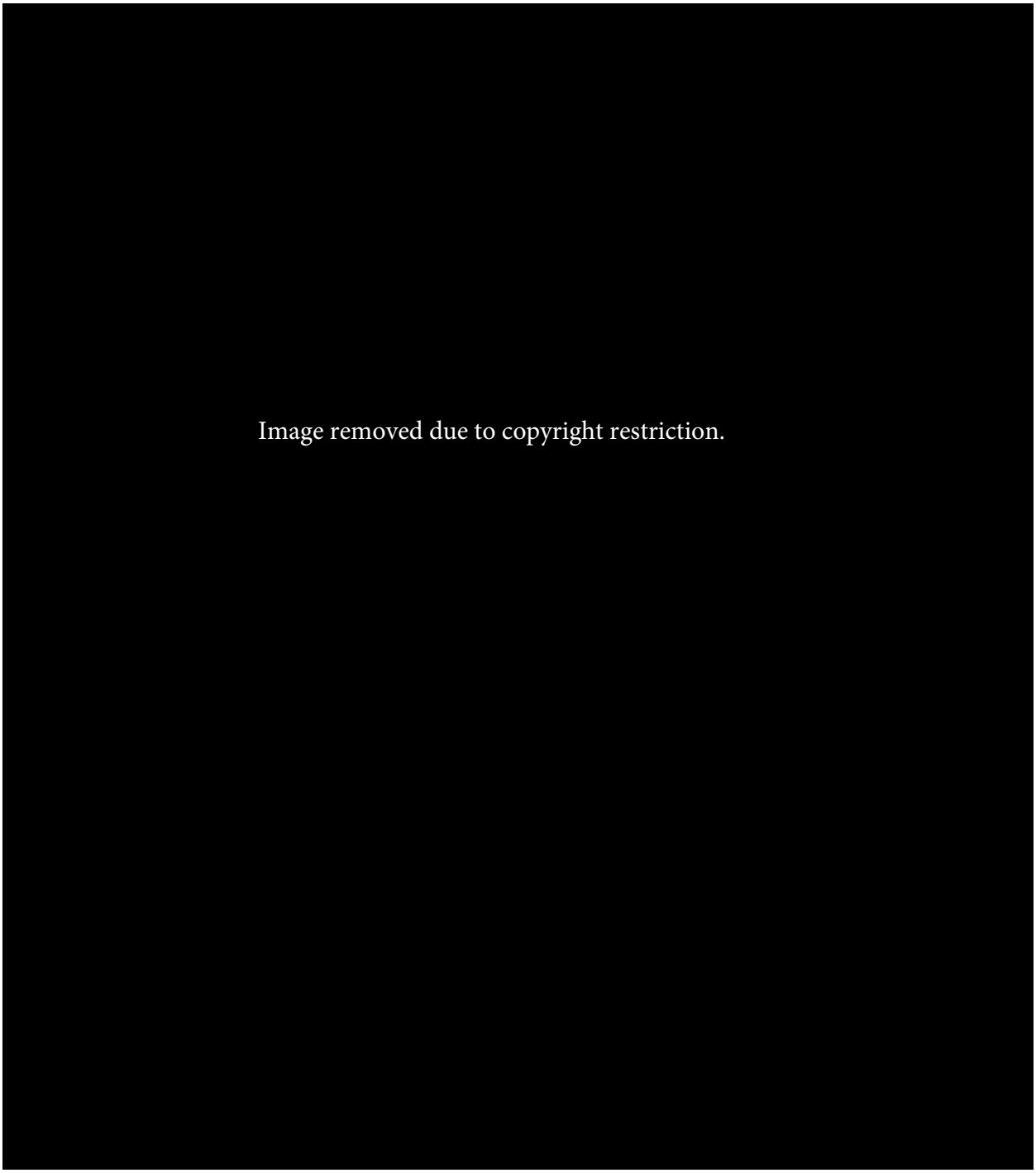


Figure 2.1: Johnston's *detailed decision schema for "Cheremic Order"* [61], showing the cascading identification of phonemes that combine to define a lexeme. The accompanying notes are shown in Figure 2.2

Figure 2.2: Notes that accompanied Johnston's "Cheremic Order", shown in Figure 2.1 [61].

appropriate change in terminology is therefore to use generic terminology, in this case, medial & lateral and dominant & subordinate, respectively.

2.2.2 Contextual Information

Context is the meta information of SL that provides tone and richness to conversation. Emphasis, adjectives and deixis are all conveyed through *how the signs are performed* [59], [61]. Morphology, timing and space all play a part in context.

Classifiers, particularly proform classifiers, are a constant feature of Auslan. For example, WINDY: is 'a breeze' if performed gently while looking nonchalant, 'ceaseless' if repeated many times while looking haggard and wary or 'a gale' if made by large, emphatic movements using the whole upper body with an intense expression.

Signs can be modified by altering their performance temporally or spatially; this almost always seems to be with respect to hand movement and expression. Tempo-spatial inflections include changing: the frequency of repetition, the number of repetitions ('recurrence'), the 'duty cycle' of repetitions ('duration)') and the duration of holds ('permanence') the duration of holds, number of repetitions and changing the path of the motion ('trajectory') [18].

Deixis is the 'pointing' aspect of language, such as 'you' (person), ' there', 'that' (place) and 'yesterday' (time), which are included in Auslan, as well as discourse, empathetic and social deixis which are not included [59], [61], [118].

Figure 2.3: A representation of addressable locations for spatial deixis within a single horizontal layer of three-dimensional signing space, adapted from [59, p. 139].

An example of place deixis is simply pointing, THERE, at the target entity, which is trivial when the entity is present and is necessary when there is no lexeme for the entity, such as for many body parts.

Deixis is appropriate even when the target entity cannot be unambiguously pointed at in the prevailing context. The entity is signed or mimed and allocated a particular location within the signing space, either by simply moving the sign there or by pointing at the space after performing the sign. Pointing at that space is then taken as if pointing directly at the target entity. A diagram of some addressable locations is provided in Figure 2.3.

A more subtle aspect of deixis is the context provided by informative inter-sign movement, termed *contextual deixis*. For example, to sign the sentence 'I got a book from the library' one might sign LIBRARY off to one side, establishing it as a location in space, then, without moving the hands from the finish of LIBRARY, immediately start BOOK at that location, pause, draw the hands to neutral position and then complete the sign. That is, the almost incidental movement from the location where library was signed to the the neutral position provides both 'got a' and 'from the' in the equivalent English sentence.

Deixis means the assignment of entities to locations in space must be remembered and the movement between signs can be communicative: an exception to the general rule.

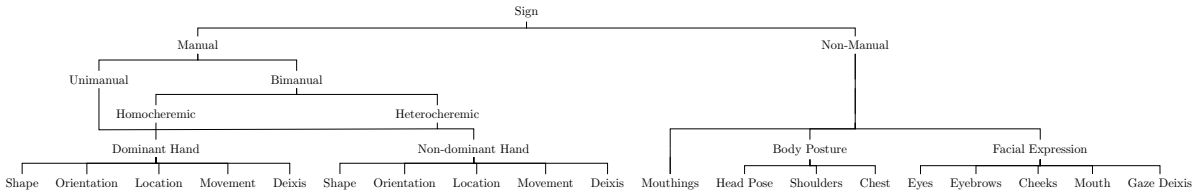


Figure 2.4: A new phonological structure: a taxonomy of sub-*sign* elements, acknowledging that phonemes are not necessarily sub-*lexical* but rather phonemes combine to form all aspects of signed languages.
The bottom level of this taxonomy are the phonemic categories; the numerous possible phonemes for each category are not shown.
Manual elements previously gave no distinction to hand.
(Manual) Orientation and (Manual) Arrangement were previously one phonemic category; arrangement is now expanded as part of the categorical structure, rather than an individual phonemic category.
NME were previously bundled together as a single phonemic category.
Deixis is a new phonemic category that can be articulated by either hand and/or gaze direction.
The two main branches of the taxonomy are shown at full-scale in Figures 2.5 & 2.6.

2.2.3 A New Phonological Taxonomy

If one considers the definition of lexeme as "a unique combination of the smallest contrastive units of language" (i.e. phonemes), then the current *"five major components of sign structure"* [59, p. 46] does form a basis for a lexeme as *expression* can have different 'sub-phonemes'. One must then consider each of the 'sub-phonemes' of expression to be phonemes in their own right.

Then there is the consideration of deixis. Superficially, the distinction between MINE and YOURS is the orientation of the hand and direction of the movement. If one rotates the frame of reference such that it is with respect to the palm, not the signer, then the combination of phonemes is identical. This illustrates the importance of the *spatial referent* and clearly identifies it a a crucial part of the combination that defines the lexeme. As such, 'motion with respect to defined locations within the signing space' must be considered a phoneme.

A hierarchical structure that takes both of these considerations into account yet still supports the notion of 'non-manual elements' can be obtained by dividing the phonological categories into those related to the hands and those not related to the hands, resulting in the *phonological taxonomy* presented in Figure 2.4.

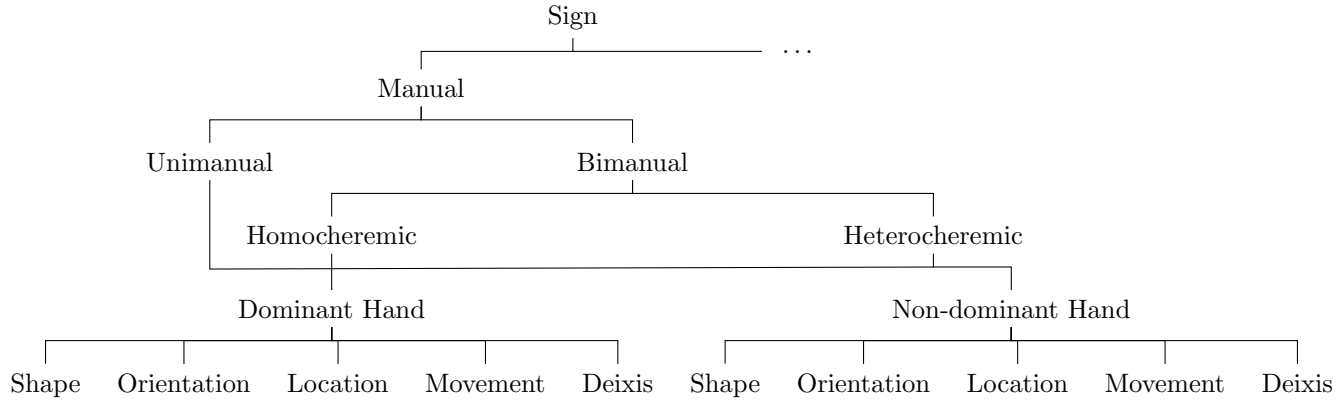


Figure 2.5: Manual elements branch of phonological taxonomy.

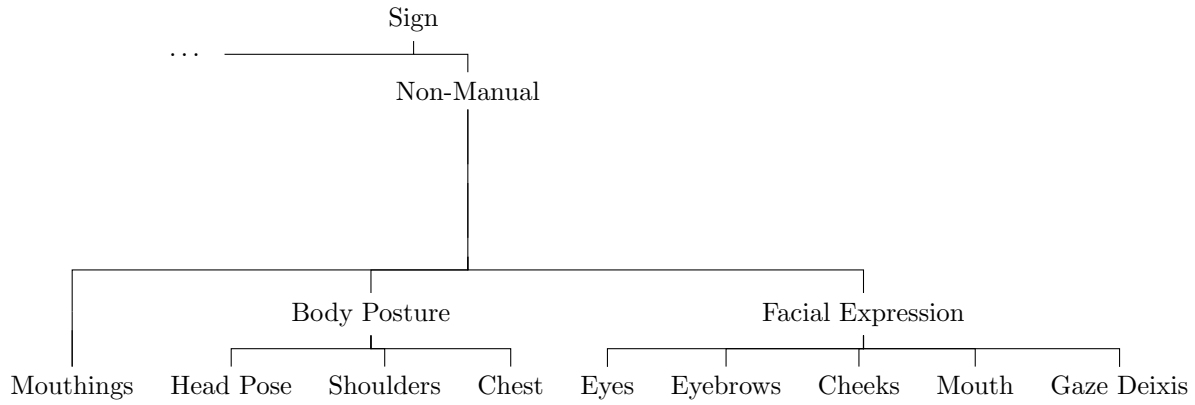


Figure 2.6: Non-manual element branch of phonological taxonomy.

2.3 Challenges

SL specific challenges arise from the diverse, dynamic and variable gestures and nuanced differences between 'critical contextual cue' and 'non-informative connecting movement'. The complexity of understanding SL and so the challenges it poses for automated recognition is apparent in the literature, for example confusion over the distinction of the terms coarticulation and epenthesis in [18] or treating them as the same concept [48].

Indeed, there is seemingly no single comprehensive glossary of challenges. The best lead is a linguistically-motivated group of Dutch researchers lead by ten Holt have published two discussions of SLR challenges [120], [121], Caridakis, Asteriadis and Karpouzis have also contributed.

A clarified, unique list of challenges as present in the literature is presented here:

Intra-signer variation is the slight difference in the way a signer performs a sign. It may be due to "whether the person is agitated or happy" [121, p. 418] or simply the random variation that occurs naturally in human performance, including due to other movement of the signer [18].

Inter-signer variation is the difference between the performances of a sign by different signers [18]; as such, this might be considered an *accent* [121]. The change may be subtle mechanical difference, such as the level of care taken to form the handshape or the gracefulness of the movement. The change may also be a mental difference, where both signers believe they are signing correctly but in truth have different ideas of how the sign should be performed, even down to different perceptions of how a single phoneme is performed.

Inter-signer variation does not extend to *regional dialects*, as here the sign is truly different, being formed by a different combination of phonemes.

Sign-sign interaction is change to the articulation of a given sign that arises due to the ending of the preceding sign and due to the beginning of the succeeding sign [121]. For example, the performance of a sign starting with the hands in neutral position will be different to a performance that begins with the hands above the head.

Movement epenthesis is the additional, non-informative movement that occurs as the hands move from the end location of one sign to the starting location for the subsequent sign [121].

Anticipation is the additional, non-informative movement of the *non-dominant* hand in preparation for a bi-manual shape; thus, anticipation is a subset of movement epenthesis [121].

Coarticulation arises when two discrete signs are performed in parallel; that is, the phonemes for two separate signs are articulated at the same time [18], [121].

Repetition is repeated motion, which can either be phonemic or emphatic [121]. In phonemic repetition, the simple act of repeating forms the entirety of the information: the number of repeats are irrelevant. In emphatic repetition, the frequency, duty cycle and number of cycles are all informative and almost certainly occur in parallel with expressions.

Occlusion refers to the inability to view part of a sign, be it due to other body parts or simply it being on the other side of the scene to the current point of view, such as self-occlusion of the hand [18].

2.3.1 New Challenges

From the linguistic perspective presented in this chapter, it is apparent there are three challenges for sign language recognition not covered in the literature, namely:

Temporo-spatial memory of signs allocated a location within the signing space to enable spatial deixis. For example, one may wish to refer to an individual who is not present, (PRO_3), so allocate them to a location, for example LOC_3. Later the signer could indicate possession of the absent person (POSS_3) by signing OWN towards LOC_3; thus an SLR system must be able to remember the spatial assignment.

Entanglement of 'contextual deixis' with non-informative inter-sign movement (movement epenthesis). Contextual deixis is a new term for additional informative *intra*-sign movement where a sign is started at one spatial deixis location and finished at another. For example, ${}_{3a}\text{BOOK}_1$: signing BOOK, starting at some LOC_{3a} = LIBRARY and ending it at LOC_1: "I got a book from the library", or signing ${}_1\text{WALK}_{3a}$: "I walked to LOC_{3a}", where LOC_{3a} is some defined shop, park or school, etc. Note that the sign in the first example, BOOK, does not normally contain any broad spatial movement (just rotation of the hands about the 'spine' of the book), thus there was a new phonemic element that changed the sense but not the meaning

of the sign; whereas the sign in the second example, WALK, the existing spatial movement during signing was repurposed to provide context.

Temporal boundaries are the points in time where one sign ends and the next begins; even for humans the point where a sign is deemed to begin can vary enormously between signers [5], [6], creating a 'knock-on' challenge for labelling.

2.4 Summary

In summary, signed language can be defined as a highly-context-dependent simultaneous and sequential combinations of established gestural elements on multiple levels that influence and are influenced by their adjacent signs.

Individual signs are defined by manual and non-manual elements that can be phonemic, modifying and deictic. Lexical signs are formed by pseudo-unique combinations of phonemes with broad interpretive range. Verb sense is inferred through motion in relative and absolute directions, provide adjectives and emphasis through dynamics such as speed and repetition. Pronouns are occasionally uniquely defined, achieved through spatial pointing, or else articulated by fingerspelling; spatial allocation allows pointing at entities that could not otherwise be unambiguously pointed at.

Structurally, languages are the combination of a set of defined terms (lexicon) with communicative actions (gestures) according to a set of defined rules (grammar). In signed languages, the lexicon is formed by lexical signs and the gestures include classifier signs, sign-mime, mime and deixis.

A single element of a lexicon is a lexeme, which are formed by unique combinations of sub-lexical units: phonemes. In signed languages, the articulation of phonemes for a single lexeme can be both parallel in space and serial in time.

Phonemes are divided into phonemic categories, which for signed languages are conventionally divided into manual and non-manual subcategories. The phonological taxonomy is shown in Figure 2.4.

The many complexities and subtleties of signed languages present significant challenges for automatic sign language recognition.

Chapter 3

Framework

SLR is a topic of international research effort, with many technical challenges related to observing and classifying a visual/gestural language. There are commercial packages [32], [111] but no solutions which truly encapsulate all elements of signed language, with notable exceptions being contextual signs and modifiers and supporting spatial addressing memory. A subsequent challenge then looms in transcription, due mainly to the absence of a written form of SLs coupled with translation to the target spoken language, to say nothing of the different tolerable variations and contextual-ambiguities of both languages.

Definitions of sign language in literature are often concise but without reference or verbose to the point of obfuscation in authoritative works. The first section of this work strove to define the essential elements that constitute sign language; not many examples were found in SLR literature that considered more than a few elements. Indeed, much work on SLR is more accurately labelled hand gesture recognition (HGR), but does serve to reflect the central challenge: accurate observation and subsequent classification of manual features. Moreover, HGR extends far beyond SLR and really lies in the domain of human-computer interaction (HCI), where hand gestures are hoped to provide means of control without the need for peripherals such as keyboards, mice and joysticks, being particularly pertinent in the growing field of virtual reality (VR).

The question then becomes: what would a framework which covered all essential elements of sign language look like? In it's most simplistic form, an ideal SLR machine would take as input continuous 'sign language' and produce as output an encoded ('written') form (to be used as-is or provided to a translation machine).

For true language recognition an optical system is currently essential, but hybrid systems are also an option, combining the fidelity of instrumentation with the spatial awareness of optical systems. To this end, an idealised framework for SLR should support

and encourage multi-modal observation of the signing scene.

Extending this concept leads to a fully modular framework that defines the input and expected output of individual sub-unit modules, allowing comparison between different methods and changing of components to suit particular applications (for example where a glove is appropriate versus where one is not) by the same system. For example, the 'handshape' module, required to output estimates of hand-shape, could be selected based upon the input type (optical or instrumented) and internal method depending on the application. This means in the first instance the selection of sub-unit modules is less critical as the cohesion and throughput of the over-arching framework is established.

A framework that takes the traditional structured approach yet includes all of the essential components of SL has been developed, shown as a simplified block diagram in Figure 3.2.

The visual model is responsible for sub-lexical classification; it takes in the stream of observations made about the signing space and slices it up to pass to classifiers for each of the phonemic categories, as well as passing the information to 'temporo-spatial memory'. The temporo-spatial memory block considers how movements change over time and recalls which entities have been allocated to which deictic location; as such it also requires input from the corresponding phonemic classifiers.

In these estimations consideration should be given to 'redundant' fragments, the aspects of an isolated sign that do not provide any discriminative power and could be 'omitted' from the sign classification without any penalty to recognition accuracy [122]. These fragments are not necessarily a particular subunit, but appear to be temporal; multiple studies have found that recognition of a sign by a human signer occurs within the first third of the sign [6], [34], [40], [119]. Realization of a module that takes advantage of this fact could follow different paths; for example, a module could reduce overall compute effort by terminating as soon as estimation reaches a pre-determined threshold, or keep looking for new candidates until the rate of candidate or estimate change falls below a threshold.

The phonemic estimates are passed to a language model that uses statistics, grammar and memory to combine phonemes into lexemes. Having identified the meaning within the grammar of the SL, it is then necessary to record that in a form that can be transferred to a translation stage. The likely intermediate is 'glossing', the codification of lexemes and their constituent elements, such as by Hamburg Notation System [44] (HamNoSys) [46] or SignWriting [74], as shown in Figure 3.1.

It is worth mentioning that some classification models, e.g. HamNoSys, do not use

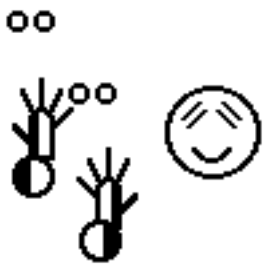


Figure 3.1: AUSLAN encoded using the SignWriting glossing system[1].

linguistic phonemes as the basic sub-units for glossing, but instead use a combination of primitive-phoneme with modifier [46]. For example, HamNoSys produces the many handshape phonemes through the convolution of two smaller sets: 12 basic shapes and 4 modifiers [25].

3.1 Framework Composition

The input *observation stream* can take many forms, such as a singular camera, be it colour or depth, or an instrument such as one that measures joint angles, or any combination of these elements. Key considerations are how much of the signing space can be observed, avoiding occlusion and the spatial discriminative performance of the modality. The stream should be continuous and have sufficient temporal resolution to avoid motion artefact and data loss.

The SLR aspects of the framework are realised by a *visual model* and a *language model*, building on the acoustic and language model framework established by automatic speech recognition [19], [73], [85]. The visual model processes the observations of the signing space to obtain estimations of phonemes while the language model uses statistics to estimate lexemes and construct sentences.

The visual model begins with segmentation, both temporally of continuous stream into individual frames and spatially of individual frames into ROIs. Both the segmented data and the data about segmentation are then passed into individual phoneme sub-stages as well as to a temporo-spatial memory stage. The phoneme sub-stages use the data to extract features that can be used for classification that are then passed to phoneme classification stages, producing estimates for each of the five phonologic categories as well as for contextual cues.

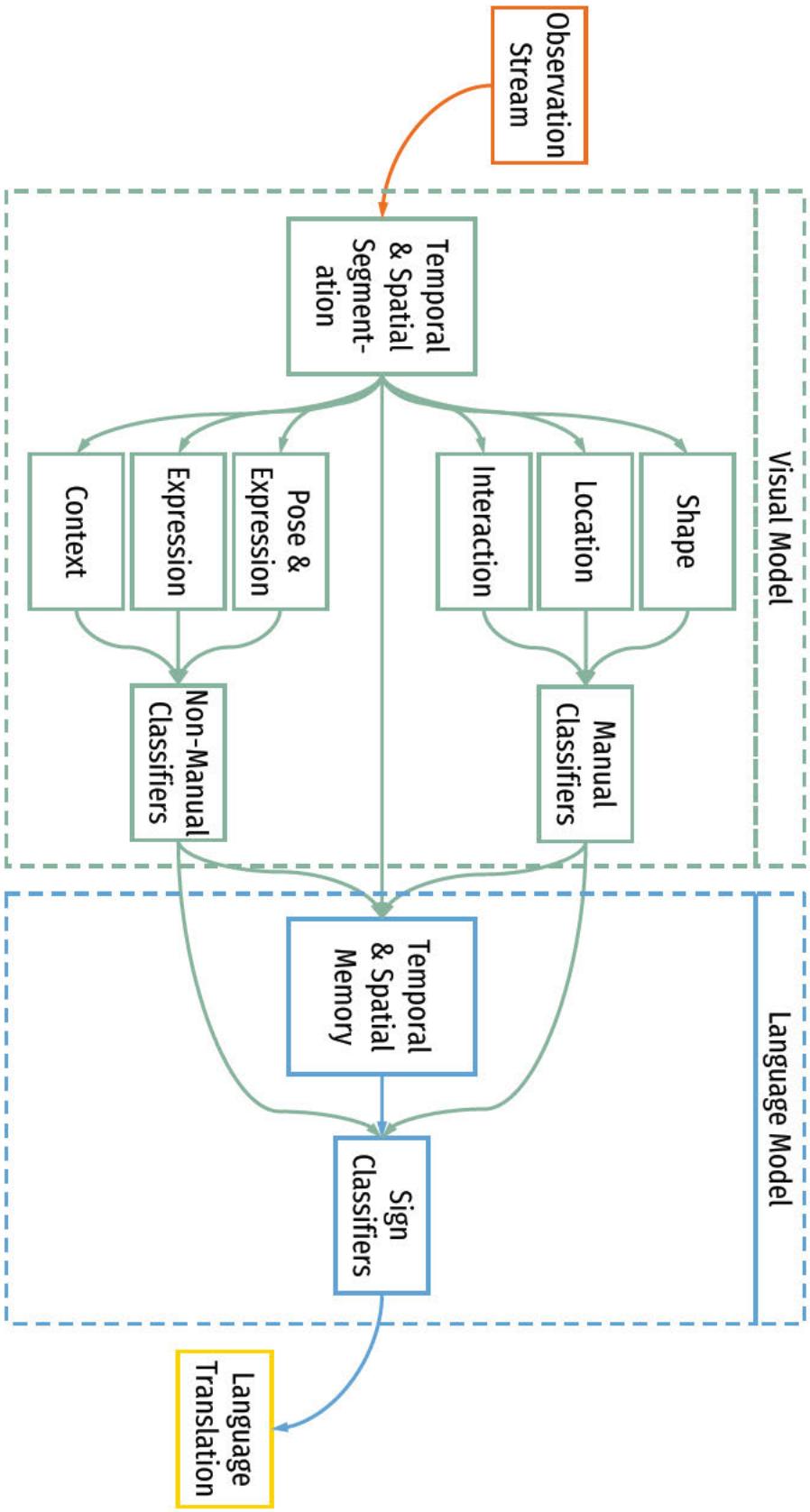


Figure 3.2: Block diagram of framework for automatic sign language recognition accounting for temporo-spatial cues and translation into the target language.

The language model considers phoneme estimates and temporo-spatial memory, using statistical methods to produce the final lexemes estimate and so construct sentences. The unmet challenge faced by this stage is thus the recognition deixis, contextual gesture and classifier signs while ignoring non-informative sign transitions. As SLs have no written form, it will likely be appropriate to gloss the sentences, that is, use a standardised notation system to encode SL, such as HamNoSys [45], [60].

Finally, a *language translation* stage takes the glossed signs and converts the recognised SL into the target output language.

Chapter 4

Recognition

Gesture recognition (GR) has applications for areas such as computer control, robot control, smart infrastructure, computer gaming, smart-home applications, healthcare, vehicle control [22], [31], [47], [57], [58], leading to fragmentation of efforts and nomenclature in the literature. GR is a primary component of natural user interface (NUI) for HCI [21]. The Venn diagram provided in Figure 4.1 shows the overlap between sign language recognition, hand gesture recognition and gesture recognition.

As a sign language lexicon is in essence a well-defined set of gestures, the signs of SL are often used as in GR for consistency and comparability [74]. For some time the most achievable aspect of SLR has been HGR, with much of the literature focused on new devices and methods to improve performance. Combined, these two factors have lead to some ambiguity in the term 'sign language recognition', where much of the purported SLR literature is in reality 'sign recognition', devoid of any linguistic element, or worse, simply 'gesture recognition' [58].

4.1 Observation

In the literature there are two main observation approaches for SLR: instrumented, such as devices which measure joint angles and optical tracking of fiducial markers or different coloured gloves; and *optical* (or visual) techniques that rely entirely upon cameras with nothing attached to the signer.

Instrumented systems have been shown to provide excellent performance for HGR and have been considered for SLR since 1996 [66], but they can be expensive, difficult to use, such as requiring bespoke fitment and laborious positioning each session, as well as an encumbrance to the user [30], [66]. Current implementations also require tethering to

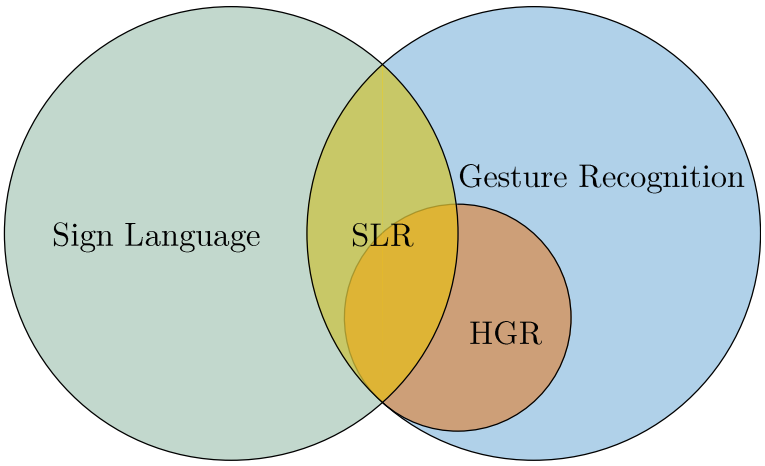


Figure 4.1: Venn diagram of sign language (SL), gesture recognition (GR), hand gesture recognition (HGR) and sign language recognition (SLR), showing that HGR is a subset of GR and that SLR is formed in the overlap between SL and GR which is only partially includes HGR.

a recording device, but this could likely be replaced by wireless data transmission or miniaturisation to create wearable devices. The significant issue of instrumentation for SLR is the complete inability to capture anything other than the shape, orientation, arrangement and location of the hands, leaving them totally ignorant of all NMEs.

Optical is generally seen as the preferable solution by many due to the freedom for natural, unimpeded signing as well as low financial cost, although they perform poorly in terms of computational effort and classification accuracy. The great advantage is the use of the same modality as human sight, meaning that theoretically an optical system can "see" everything we can.

4.2 Literature Review

A literature review was conducted to answer the question: "what are the promising modalities and techniques for SLR?" The results are shown in Tables 4.1 through 4.4.

These tables show that approaches are migrating towards optical modalities, with depth providing an increase in performance, although instrumented approaches are the most accurate. A confound in the reported accuracies are the simplicity of the recognition task; the works on continuous televised broadcasts reporting accuracy as WER (converted to word recognition rate (WRR)) face the greatest challenge: continuous signing. Even here, however, the vocabulary is greatly reduced with no contextual elements,

Table 4.1: Literature review: Survey of sign language and gesture recognition techniques. Goal: gesture recognition (GR), hand gesture recognition (HGR), mouthing recognition (MR), sign language recognition (SLR), sign language recognition and translation (SLRT), sign language translation (SLT). Modality: optical (O), instrumented (I).

Work	Goal	Mode	Device(s)
Camgoz, Hadfield, Koller *et al.* [15]	GR	O	red, green & blue (RGB)
Chen, Deng, Pang *et al.* [22]	HGR	I	wrist-worn camera
Hernández-Vela, Bautista, Perez-Sala *et al.* [49]	HGR	O	Kinect
Keskin, Kıraç, Kara *et al.* [68]	HGR	O	Kinect
Ming and Jianbo [99]	HGR	O	Kinect
Nai, Liu, Rempel *et al.* [101]	HGR	O	Kinect
Marin, Dominio and Zanuttigh [87]	HGR	O	Kinect + Leap Motion
Dominio, Donadeo and Zanuttigh [31]	HGR	O	Kinect 1
Chen, Li, Sun *et al.* [21]	HGR	O	RGB
Haria, Subramanian, Asokkumar *et al.* [47]	HGR	O	RGB
Koller, Ney and Bowden [76]	HGR	O	RGB
Ji, Song, Xiong *et al.* [57]	HGR	O	Worn RGB
Koller, Ney and Bowden [75]	MR	O	RGB
Galka, Masior, Zaborski *et al.* [39]	SLR	H	IMU
Abhishek, Qubeley and Ho [1]	SLR	I	glove with capacitive sensors
Kadous [66]	SLR	I	Nintendo Powerglove
Camgoz, Hadfield, Koller *et al.* [16]	SLR	O	RGB
Kelly, McDonald and Markham [67]	SLR	O	Grayscale
Huang, Zhou, Li *et al.* [52]	SLR	O	Kinect
Inoue, Shiraishi, Yoshioka *et al.* [53]	SLR	O	Kinect
Kumar, Saini, Roy *et al.* [83]	SLR	O	Kinect
Kumar, Gauba, Pratim Roy *et al.* [82]	SLR	O	Kinect + Leap Motion
Agarwal and Thakur [2]	SLR	O	Kinect 1
Conly, Zhang and Athitsos [23]	SLR	O	Kinect 2
Cui, Liu and Zhang [28]	SLR	O	RGB
Forster, Koller, Oberdörfer *et al.* [36]	SLR	O	RGB
Hassan, Assaleh and Shanableh [48]	SLR	O	RGB
Huang and Zhang [51]	SLR	O	RGB
Kishore, Rao, Kumar *et al.* [70]	SLR	O	RGB
Koller, Ney and Bowden [74]	SLR	O	RGB
Koller, Forster and Ney [73]	SLR	O	RGB
Koller, Zargaran, Ney *et al.* [78]	SLR	O	RGB
Koller, Zargaran and Ney [77]	SLR	O	RGB
Koller, Zargaran, Ney *et al.* [79]	SLR	O	RGB
Koller, Camgoz, Ney *et al.* [72]	SLR	O	RGB
Chai, Li, Lin *et al.* [19]	SLRT	O	Kinect 1
Camgoz, Hadfield, Koller *et al.* [17]	SLT	O	RGB

Table 4.2: Literature Review: Sources.
Lexicons: Arabic Sign Language (ArSL), American Sign Language (ASL), Auslan, Chinese Sign Language (CSL), Indian Sign Language (ISL), Japanese Sign Language (JSL), RWTH: RWTH-PHOENIX-Weather datasets.
N/R: not reported.

Work	Source				
	Lexicon	# Lexemes	# Replicates	# Signers	# Samples
[15]	N/R	249	N/R	23	47 933
[22]	ASL: 0-9	10	10	10	1000
[49]	ChaLearn 2011	N/R	N/R	N/R	50 000
[68]	ASL: static A-Z	24	N/R	N/R	65 000
[99]	Hand shapes	5	8	5	200
[101]	ASL: static A-Z	24	N/R	5	120
[87]	Hand shapes	10	10	14	1400
[31]	ASL	12	10	14	1680
[21]	Hand shapes	5	100	1	500
[47]	Hand shapes	6	N/R	N/R	N/R
[76]	RWTH 2014	45	Varies	9	786 750
[57]	Hand shapes	10	5	40	2000
[75]	RWTH 2010	15	Varies	7	1832
[39]	N/R	40	10	5	2000
[1]	ASL: 0-9 *or* A-Z	36	30	1	1080
[66]	Auslan	95	8 to 20	5	6550
[16]	N/R	N/R	N/R	23	1 005 136
[67]	N/R	10	2	24	480
[52]	N/R	25	3	9	675
[53]	JSL: static Hiragana	41	10	8	3280
[83]	ISL	30	9	10	2700
[82]	ISL	50	15	10	7500
[2]	CSL: 0-9	10	N/R	N/R	N/R
[23]	ASL	1113	3	5	450
[28]	RWTH 2014	1081	Varies	9	65 227
[36]	SIGNUM	455	Varies	25	11 109
[48]	ArSL	80	10	N/R	400
[51]	CSL	N/R	N/R	50	17 000
[70]	ISL	200	3	5	3000
[74]	RWTH 2010	421	Varies	7	1832
[73]	RWTH 2014	1081	Varies	9	65 227
[78]	RWTH 2014	1081	Varies	9	65 227
[77]	RWTH 2014	1081	Varies	9	65 227
[79]	RWTH 2014	1081	Varies	9	65 227
[72]	RWTH 2014	1081	Varies	9	65 227
[19]	CSL	239	5	1	239
[17]	RWTH 2014	1066	N/R	9	7096

Table 4.3: Literature review: Segmentation.
Temporal: static = irrelevant, data intrinsically segmented; otherwise continuous: manually segmented or automatically segmented to sentences based on transcript.
Spatial: thresholding: luminance, $\text{chrominance}_{\text{blue}}$ & $\text{chrominance}_{\text{red}}$ (YC_bC_r) colour-space, red, green & blue (RGB) colour-space, depth, joint.
N/R: not reported.

Work	Data Segmentation	
	Temporal	Spatial
[15]	[t-1][t][t+1]	None
[22]	Static	YC_bC_r
[49]	None	None
[68]	Static	Manual
[99]	Static	Joint + depth
[101]	Static	None
[87]	Static	None
[31]		Depth
[21]	Static	GMM
[47]	Static	RGB
[76]	None	None
[57]	None	Assumes pre-segmented
[75]	None	None
[39]	N/R	N/A
[1]	Static	None
[66]	None	N/R
[16]	None	None
[67]	Static	Manual
[52]	None	None
[53]	None	Depth
[83]	Static	None
[82]	Manual	None
[2]	Static	None
[23]	Manual	N/R
[28]	Sentences	None
[36]	None	None
[48]	[t]-[t-1]	None
[51]	None	None
[70]	None	None
[74]	None	None
[73]	None	None
[78]	None	None
[77]	None	None
[79]	None	None
[72]	None	None
[19]	N/R	N/R
[17]	Sentences	None

Table 4.4: Literature review: Recognition.
Level: Phonemic: body movement (BM), body pose (BP), hand movement (HM), hand shape (HS).
Classifiers: common methods: convolutional neural network (CNN), dynamic time warping (DTW), hidden Markov model (HMM), histogram of oriented gradients [29] (HoOG), scalar vector machine (SVM).
Performance measure: mean Jaccard Index (mJI), true positive rate (TPR), word recognition rate (WRR) (entries marked * are converted from word error rate (WER)).

Work	Recognition				
	Level	Features	Classifier	Performance	Measure
[15]	N/R	N/R	3D CNN	31.4 %	mJI
[22]	HS	N/R	Lookup table	99.4 %	'accuracy'
[49]	Lexemes	HoOG, HOF	DTW	67.8 %	tpr
[68]	HS	PCF	RDF	68.0 %	N/R
[99]	HS	SP-EMD	N/R	99.6 %	'accuracy'
[101]	HS	coordinates, angles	RDF	80.7 %	'accuracy'
[87]	HS	coordinates, angles	SVM	91.3 %	'accuracy'
[31]	HS	N/R	SVM	93.8 %	'accuracy'
[21]	HS	N/R	K-SVD	98.7 %	'recognition'
[47]	HS	N/R	Haar	64.0 %	'accuracy'
[76]	HS	Whole frame	EM-CNN-HMM	55.0 %	WRR*
[57]	HS	BEHB*	CNN-SVM	97.7 %	'accuracy'
[75]	Mouthings	N/R	CNN	55.7 %	precision
[39]	HM	N/R	HMM	99.8 %	'accuracy'
[1]	HS	N/R	Lookup table	92.0 %	'accuracy'
[66]	HS	N/R	decision tree	80.0 %	'accuracy'
[16]	HS	N/R	LSTM CNN	80.3 %	'Top-1 accuracy'
[67]	HS	eigenspace	SVM	95.2 %	WRR
[52]	HS & BM	HoOG + skeleton	3D CNN	92.4 %	'accuracy'
[53]	HS	TSC	CNN	84.0 %	'accuracy'
[83]	BP & HS	coordinates, angles	HMM	83.8 %	'accuracy'
[82]	HS & HM	coordinates	HMM	95.6 %	'accuracy'
[2]	FF	N/R	SVM	88.0 %	'accuracy'
[23]	HS & HM	N/R	DTW	62.0 %	'accuracy'
[28]	Lexeme	N/R	CNN + RNN-LSTM	61.3 %	WRR*
[36]	Lexeme	HOG3D	HMM	87.5 %	WRR*
[48]	Lexeme	N/R	HMM	94.5 %	WRR
[51]	Lexeme	N/R	DTW	82.7 %	WRR
[70]	Lexeme	N/R	CNN	89.0 %	WRR
[74]	HM	eigenvectors	RASR	68.5 %	precision
[73]	HM, FE	HOG3D	HMM	47.0 %	WRR*
[78]	Lexeme	Whole frame	CNN-HMM	61.5 %	WRR*
[77]	Lexeme	Whole frame	CNN-BLSTM-HMM	54.9 %	WRR*
[79]	Lexeme	Whole frame	CNN-HMM	58.9 %	WRR*
[72]	MS, HS	Whole frame	CNN-LSTM-HMM	71.7 %	WRR*
[19]	HM	N/R	Nearest match	N/R	N/A
[17]	Lexeme	N/R	CNN & RNN+HMM	N/R	N/A

Table 4.5: Literature review: Signers participating in SLR studies. N/R: not reported.

Work	Participants were signers?
Galka, Masior, Zaborski *et al.* [39]	No
Abhishek, Qubeley and Ho [1]	No
Kadous [66]	Yes
Camgoz, Hadfield, Koller *et al.* [16]	N/R
Kelly, McDonald and Markham [67]	N/R
Huang, Zhou, Li *et al.* [52]	N/R
Inoue, Shiraishi, Yoshioka *et al.* [53]	N/R
Kumar, Saini, Roy *et al.* [83]	N/R
Kumar, Gauba, Pratim Roy *et al.* [82]	Yes
Agarwal and Thakur [2]	N/R
Conly, Zhang and Athitsos [23]	No
Cui, Liu and Zhang [28]	Yes
Forster, Koller, Oberdörfer *et al.* [36]	Yes
Hassan, Assaleh and Shanableh [48]	N/R
Huang and Zhang [51]	N/R
Kishore, Rao, Kumar *et al.* [70]	Yes
Koller, Ney and Bowden [74]	Yes
Koller, Forster and Ney [73]	Yes
Koller, Zargaran, Ney *et al.* [78]	Yes
Koller, Zargaran and Ney [77]	Yes
Koller, Zargaran, Ney *et al.* [79]	Yes
Koller, Camgoz, Ney *et al.* [72]	Yes
Chai, Li, Lin *et al.* [19]	N/R
Camgoz, Hadfield, Koller *et al.* [17]	Yes

though likely considerable adjective modifiers were used.

Interestingly, of the 24 studies on SLR, only 16 reported whether the participants were signers and of those only 13 were signers. Half of the 16 signer-based studies worked on the RWTH-PHOENIX-Weather datasets.

4.2.1 Summary

An interesting progression is that of a group of researchers from RWTH Aachen University, Germany, lead by Kollor, who transitioned from a phonological classifier approach to a whole-scene deep learning approach. The group started using 'subunits' (phonemes), 'trajectories' (hand movements) and speech recognition techniques (RASR) [74] and pioneered the use of 'real signing data' through videos of freely-televised weather broadcasts with (*Deutsche Gebärdensprache*; German Sign Language (DGS)) sign language translation by interpreters[1] complete with transcripts for labelling, which they released as the 'RWTH-PHOENIX-Weather' corpus [74], evolving over time [15], [37] and being augmented at others [16].

The large volume of real data from multiple signers enabled the use of deep learning (convolutional neural network (CNN)) to push the state-of-the-art and classify mouthings [75], multiple phonemes [16] and continuous sign [17], [72], [73], [78], [79].

Kollor's most recent work [72] takes the group full circle, including mouth shape and hand shape phonemic classification in a model that also analyses full frames to achieve a WRR of 71.7 % on a 9-signer dataset.

4.3 Visual Challenges

Visual approaches employ cameras to observe the signer. Challenges in visual gesture recognition (VGR) include:

- Camera-related: image quality, adequate lighting without changes in illumination (creating additional challenge for capture in uncontrolled environments, such as outdoors) [20].
- Accurate capture of three-dimensional (3D) signing space, such as movement towards the capture device [18], [20].

[1] The interpreters are claimed to be "hearing" in [75, p. 479] and "native" in [73, p. 109]; an unlikely combination, but it seems reasonable to assume they are *fluent* signers.

- Appearance of a sign depends on the view-point of the observer [73].
- Spatial segmentation of the region of interest from the background. Multiple independent circumstances in which this arises: isolating the signer from the scene background and isolating e.g. the hands from the signer's chest, or the dominant hand from the subordinate hand.
- Temporal segmentation of signs from a continuous stream of observations is inherently difficult due to the ambiguity of sign boundaries [3], [71], requirement for per-frame labelling for training [51] and a 'knock-on' penalty for classification performance in the event of incorrect segmentation [51], although [77] solve this through iterative realignment.
- Real signing is fast; recording at low frame-rates and at low spatial resolutions leads to motion artefacts [74].
- People come in different shapes and sizes [52].
- Some phonemes are 'loosely constrained' and can vary greatly, creating difficulty in labelling let alone classification [75]

In terms of challenges not noted in the literature, one is skin colour, which no doubt has an effect on both the methods used to create features and then the classification performance of those features. Another challenge is classification of 'sign language' based on the performance of signs by 'non signers'; much of the literature did not specify whether the 'signers' used in their SLR study were, in fact, signers [3], [16], [19], [47], [48], [51] [53]

Camera related constraints

In a visual system, the camera itself presents constraints, including spectral range, image resolution, frame rate, optical characteristics and interface limitations [20].

In terms of spectral range, a trade-off exists between cost and tolerance to changes in illumination infra-red (IR) sensors detecting body heat are highly robust but expensive, while visual range (390 nm to 750 nm) sensors are less robust but far cheaper [20]. Sensitivity to changes in illumination in a holistic sense varies considerably by approach, with some techniques such as full-frame deep-learning algorithms being impressively robust [78], [108] while others leverage additional modalities such as depth data to provide more robust segmentation [35], [65], [83].

The optical setup of a camera dictates how light from the scene reaches the sensor, with two key specifications being the distance from the front of the lens at which the image is focused: 'focal length' f and the diameter of the light entrance hole (or pupil): 'aperture' D; two combine to form a third specification: focal ratio, or 'f-number' N, where $N = \frac{f}{D}$. Focal length is important as it dictates the angle of light that is able to reach the sensor, or field-of-view (FoV), with shorter f giving wider FoV. Aperture dictates both the range of distances from the lens in which the subject remains in focus, or depth-of-field (DoF) and volume of light reaching the sensor. Narrower aperture gives greater DoF (more of the scene in focus), allowing greater tolerance to signer position relative to the camera without loss of information. Wider aperture increases light volume, permitting in an image with greater intensity variation (more information) without requiring compensation by increased sensor gain, which introduces noise. Clearly a compromise must thus be struck between DoF and light volume; in webcams aperture is fixed and selection should consider the tolerance to low illumination[20].

Modern cameras tend to have far greater image resolution than those from a decade ago, such as from 640×480 pixels to 1920×1080 pixels ('1080p'), with a corresponding increase in pixels-per-frame from 0.3 to 2 megapixels [20], [100]. Low-resolution images create several difficulties for computer vision [100]; the most obvious being that the pixel size should be small enough to visually distinguish descriptive features, such as fingers and gaze. Higher resolution means more information available for distinction and classification, but comes at a transfer penalty.

Frame rate also impacts classification ability. As frame rate drops greater periods of time occur in which no image is recorded, increasing the chance that descriptive movement may be missed. Standard video frame rates are around 25 frames per second (fps), or Hertz, with 'high frame rate' cameras popular in action scenarios typically recording at 60 fps. Although there is no known quantification of sign-rate for fluent signing, one paper found a mean rate 2.7 words per second[8], but this was for *finger-spelled* words; if the average word contained 5 letters, this would be around 25 distinct signs formed per second[2]. Koller, Zargaran, Ney *et al.* specify a 'frames per sign' rate of 9.8 for their RWTH-PHOENIX-Weather 2014 corpus, but the the frame rate of the camera and the definition of sign boundaries is unknown [79]; estimating a frame rate in the order of 30 Frames per Second (fps) gives a sign duration of around 0.3 s, or a sign

[2]This estimate of 25 signs per second is questionable as a proxy for sign rate in 'conversational Auslan', as not only does fingerspelling occur rapidly, with little inter-sign movement required, but this paper discussed ASL, which has uni-manual letter signs, thus representing perhaps the smallest possible inter-sign movement.

frequency of 3 Hz, far lower than the 25 Hz estimate above.

Making a somewhat arbitrary compromise and assuming a sign frequency of 15 Hz and applying Nyquist-Shannon sampling theory proscribes a minimum sampling rate of 30 Hz, suggesting standard 24 to 29.97 fps video may be adequate for sign language recognition.

There is also uncertainty in how sampling rate is defined as the exposure (period of time for which time the sensor is receiving light) for each frame is unknown. For example, a camera with a frame rate $f = 10$ fps suggests a period of $T = 1/f = 1/10 = 0.1 = 10$ ms per frame, the actual exposure period might be much less than that, resulting in less motion blur than anticipated but also recording less information.

Temporal resolution is only half of the equation, however; spatial resolution plays an important role in motion blur. If an arm moves one third of the scene between a frame on a low resolution camera where the arm is comprised of relatively few pixels a greater number of pixels will blur than in an image captured by a camera with equivalent optics but a higher sensor resolution.

The cost of higher image resolution and frame rates is the very thing it gains: information. The increased information requires greater interface capability, storage capability and processing time.

With the light intensity value at each pixel stored as a binary value (typically 8 bit) for each colour channel (3 for RGB), the data transfer per second, or 'bitrate' can be calculated: $bitrate = W \times H \times bpp \times f_r$, where W is the frame width, H is the frame height, bpp is the bits per pixel and f_r is the frame rate. An RGB camera recording 640×480 px with 8 bit px^{-1} at a frame rate of 24 Hz thus results in an approx. 177 Mbit s^{-1} transfer rate, while 1080p @ 60 Hz requires nearly 3 Gbit s^{-1}! Add to that multiple infra-red imagers and depth streams from onboard depth module and the transfer rate is quadruple that.

Accommodation of this greatly increased data load is achievable using standard interfaces, with USB 2.0 supporting 480 Mbit s^{-1} being superseded by USB 3.0, which supports a sufficient maximum transfer rate of 4.8 Gbit s^{-1}, in 2011 and subsequent releases of USB 3.1, USB 3.2 and USB 4 supporting 10 Gbit s^{-1}, 20 Gbit s^{-1} and 40 Gbit s^{-1} [114], respectively, providing headroom for greater transfer rate requirements in the future. Additionally, some webcams include internal processors that compress the data prior to transmission, reducing interface requirements but potentially reducing classification power [20].

Capturing 3D Signing Space

Signs are formed in three-dimensional space, with critical information imparted using relative and absolute spatial positioning [59]. Accordingly, a visual recognition system will miss information if it only captures a two-dimensional image. Here again the extent of this effect is dependent upon the techniques employed; state-of-the-art recognition is currently held by a team analysing television-broadcast (2D) weather report signers [72]. There are several approaches to obtaining 3D data, including using singular depth or '3D' cameras or 'visual sensor networks' (VSNs), arrays of 2D (non-depth) cameras, 3D cameras or mixed 2D/3D cameras [20], [127].

The most common 3D cameras couple a typical RGB sensor with an additional IR sensor for time-of-flight depth to gain additional information about the scene without altering that visible to humans [11], [80], [129].

Time-of-flight cameras use frequency-modulated flood illumination. Knowing the speed of light $c = 3 \times 10^8 \,\mathrm{m\,s}^{-1}$ and measuring the time taken for reflected light to reach the sensor (thus the name) allows computation of a depth map of the scene, with the depth D of a given pixel given by: $D = \frac{time\ \ of\ \ flight}{2c}$ [11].

Despite their capability, time-of-flight cameras were expensive and the release of the affordable consumer-grade depth camera Microsoft Kinect for Xbox 360 in 2010 [26] ushered in numerous explorations of the Kinect for gesture recognition, particularly between 2011 and 2013 [2], [9], [19], [43], [65], [109], [129], [130].

Another depth camera technology is structured light. These cameras project a unique infra-red pattern onto the scene and observe the reflected pattern using a sensor. Distortion of the reflected pattern is then computed to determine depth [129].

Although 'standard' CMOS camera sensors are able to observe IR, RBG-D cameras typically employ a discrete sensor for this purpose. The additional sensor often has a lower resolution (e.g. D: 1280×720 versus RGB:1920×1080 in the Intel RealSense D435 [55]) but higher frame rate (e.g. D: 90 fps versus RGB: 30 fps in the Intel D435 [55]) This suggests manufacturers believe there is less need for spatial precision and increased need for temporal precision in depth capture; regardless of intent, this fact may influence approaches to classify motion.

Not only were Kinect cameras available 'off-the-shelf', they also provided much better resolution: 640×480 for the Kinect, versus 160×124 for the SwissRanger SR2, a popular time-of-flight camera of the same era [11], [80], [129]. Although the Kinect was discontinued by Microsoft in 2017 [26], several other consumer devices had become

available, a common choice for gesture recognition being the Intel RealSense camera. A notable development of the RealSense was the introduction of a second IR sensor, providing in-camera stereo 3D vision akin to human sight [55]. Another depth camera that had many investigations for gesture recognition was the Leap Motion sensor; a purely IR depth camera designed to sit on a surface below the hands and fit a skeletal model [20], [82], [87], [88], [106].

Visual sensor networks use multiple cameras simultaneously to obtain a variety of views of the scene. The most basic approach is a pair of standard RGB cameras: 'stereoscopic', which can be extended to an array of multiple RGB cameras, there are also mixed RGB + (RGB-)D VSNs and pure depth sensor VSNs [20], [102]. Although a purely RGB VSN can generate a depth map, the inclusion of depth sensors greatly increases this ability [127] and reduces computational effort [20]. It is worth noting that while many RBG-D cameras are in fact a pairing of a 2D RGB sensor with a 3D depth sensor, they are not considered a VSN as the pairing is not used for additional depth distinction purposes per se; likely the pair are too closely co-located.

Issues with coordinating multiple cameras arise when attempting to match data to produce a singular correspondence, such as difficulty finding unique 'landmarks' for matching, occlusion of landmarks and variations between recorded images due to physical differences of any two cameras and noise [20]; of course the use of multiple cameras proportionally increases data handling requirements, though techniques have been explored to identify and remove the redundant, duplicated data inherent in this approach [127].

4.3.1 Spatial Segmentation

Visual techniques must identify regions of interest in order to achieve an 'apples to apples' comparison. For example, it is common in hand gesture recognition to segment the hand(s), allowing computation of features in that isolated region. Segmentation through pure image analysis remains a challenge in computer vision in general, as pure image analysis approaches rely upon the input data, rendering them limited by changes in illumination and in situations in which the region of interest is visually similar to other regions within the image [20]. Other ways around these issues are to use an additional data source (e.g. depth data) that simplifies segmentation or to simply avoid segmentation entirely, such as in deep learning, as will be discussed later.

4.4 Pattern Recognition Techniques

Automated recognition problems, such as facial recognition, gesture recognition, hand shape recognition and so on are sub-problems of the broader pattern recognition problem that is the essence of computer vision. While each sub-problem has its own region of interest and solution space, the techniques used are common to all. Indeed, sign-language recognition is a clear example of a clustered problem; solving for the overall sign involves solving many smaller problems: the visual phonemes.

Approaches to pattern recognition are constantly evolving, with no one ideal technique for any given problem. Techniques vary in accuracy, tolerance, robustness and efficiency.

Current efforts can be reduced to two categories, differentiated at the feature extraction level. In classical manual feature extraction techniques, algorithms are hand-chosen and tailored to isolate particular elements or image *feature descriptors*, while in automated feature extraction an algorithm processes a large corpus to determine its own abstract set of features with high classification power known as 'deep learning'.

The most common pattern recognition technique for HGR is the histogram of oriented gradients [29] (HoOG) algorithm, that divides an image into individual cells. The final output is a set of histograms, one for each cell, produced from direction gradients: the change in intensity value of each pixel compared to it's immediate neighbours in both horizontal and vertical directions (optionally, repeated for each colour) at each pixel in the cell, binned by orientation angle [38], [50].

HoOG is by no means a new method [95]; it has been used for hand-gesture recognition since at least 1995 [38] and remains the most common descriptor used in SVM-based handshape-recognition literature [25], likely due to its aptitude for detecting edges such as those of fingers that are generally challenging to discern in CV.

Although, courtesy of the cell-based approach, HoOG is invariant to scale, the greatest pitfall of HoOG in ASLR may be its dependence on 3D rotation, although this could be mitigated with, for example, stereoscopic cameras to model and correct for rotation.

Chapter 5

Implementation

5.1 Hardware Selection/Capture Modality

As discussed in lit rev, there are two base capture modalities: optical and instrumented, that can be used alone or in combination, e.g. optical and instrumented, optical ×3 and instrumented, etc. Both modalities have their advantages and disadvantages; for Sign-to-Text it was decided that due to their tethering and hampered scalability, instrumented mode would be avoided in preference of a purely optical systems. As a starting point, a single, frontal depth camera provides a single view of 3D space as a pairing of a planar (2D) colour image with 'distance from camera' values for each point: 'depth'.

5.1.1 RGB-D Camera Requirements

Calculation of minimum requirements is based on assumptions regarding recording conditions and the combined premises that: 1. the camera must observe the full span of the signing space and 2. the camera should be as close to the signer as possible to reduce loss of detail.

Scaling the anthropometric data provided in [81, Table 1.3] to suit a 2 m individual (selected somewhat arbitrarily on the basis of a 'realistic large individual'), the signing space occupies a horizontal width of 2.2 m, corresponding to the 2 m individual's arm-span. Subtracting 'downward grip reach' from 'overhead grip reach' gives a vertical signing space span of 2.6 m 0.8 m = 1.8 m.

Colour frame size also known as 'resolution', the size in pixels of a 'frame' (a single still image produced by the camera), usually expressed as width by height (W × H). ASLR requires enough detail to optically recognise handshapes, i.e. individual

fingers must be visually discernible. If a finger is 10 mm wide, being recorded in a frame 2.2 m wide, then 220 fingers could fit in the width of the frame; if it takes 5 px to clearly delineate a finger, then the frame must be 220 fingers × 5 px/finger = 1100 px wide. Similarly for a 1.8 m high signing space, the frame height must exceed 900 px.

Colour frame angle of view also known as field of view (FOV), angle of view (AoV) refers to the angle of the field that is visible to the camera sensor. If the camera-to-signer distance is limited to a maximum of 3 m, then the angle of view must exceed 41° horizontally and 34° vertically, based on geometric calculation:

$$Angle = 2 \times \arctan\left(\frac{\frac{span}{2}}{distance}\right)$$

The greater the AoV, the smaller the camera-to-signer distance can be. There is a third parameter often included in AoV specifications: diagonal angle; but as this is purely a function of width and height it contains no new information and so is not specified here.

Colour frame rate also known as fps, frame rate refers to the number of frames produced by a camera in one second, thus, the appropriate SI unit is Hertz (Hz) but this conveys less information than fps as Hz only counts periodic behaviour and ignores what is being counted. Considerations begin with loss of data: if the frame rate is too low, it may miss information between frames as well as contain motion blur within frames. Further considerations come from the Nyquist-Shannon sampling theorem: the discrete-time signal sample rate (here, frame rate) must be at least twice that of the continuous-time signal it is observing to avoid data loss. Unfortunately no clear data has been obtained on conversational sign frequency so an assumption of less than 10 signs per second was made. Applying Nyquist-Shannon gives a minimum frame rate of 20 Hz.

Depth frame size height and width of the depth image produced by the camera are not as exacting as the colour frame as the depth image is primarily used for automated background removal. If it takes 3 px to delineate a 10 mm finger from the background, then the depth frame must exceed 660 × 540 px (W × H).

Depth frame angle of view are subject to the same requirements as for colour, thus must exceed 41° × 34° (H × V).

Table 5.1: Summary of required camera parameters.

Parameter	Criteria
Colour frame size / resolution	$\geqslant 1100 \times 900$ px (H × V)
Colour frame angle of view	$\geqslant 41° \times 34°$ (W × H)
Colour frame rate	$\geqslant 20$ Hz
Depth frame size / resolution	$\geqslant 660 \times 540$ px (H × V)
Depth frame angle of view	$\geqslant 41° \times 34°$ (W × H)
Depth frame rate	same as colour frame rate
Minimum depth	$\leqslant 1$ m
Maximum depth	$\geqslant 3$ m
Shutter	Global

Depth frame rate for segmentation to be effective it must keep pace with the colour image, thus the frame rate should match.

Minimum depth is the smallest camera-signer distance; a 2 m tall signer standing 3 m from the camera could reach 1 m forward, leaving 3 m − 1 m = 2 m separation between hand and camera. Even if the camera-to-signer distance was 2 m, there would still be a 1 m separation, so this makes a sensible minimum depth.

Maximum depth is the farthest distance the camera needs to observe; as this study has not found an example of a sign that extends behind the signer (an example of a sign that is close to reaching over the shoulder is LAST WEEK[1]), depth need only exceed the camera-to-signer distance: $\geqslant 3$ m.

Shutter refers to the capturing of an image by the sensor. Cameras with a 'rolling' shutter capture a scene row-by-row; thus, time passes between the recording of each row in which motion can occur, causing image defects. 'Global' shutters avoid this issue by capturing the entire scene instantaneously and so are required.

These parameters are summarised in Table 5.1. The relevant specifications of several consumer-grade depth cameras (Creative BlasterX Senz3D [27]m Microsoft Kinect 2.0 [98] (A.K.A. 'Microsoft Kinect for Xbox One') and Intel RealSense D435 [54] are provided in Table 5.2.

[1] http://www.auslan.org.au/dictionary/words/last%20week-2.html

Table 5.2: Comparison of several consumer-grade depth cameras against required parameters.

Parameter	BlasterX Senz3D	Kinect 2.0	RealSense D435
Colour			
Resolution (pixels, W × H)	1920 × 1080	1920 × 1080	1920 × 1080
Angle of view (°, H × V)	67 × 38	No data	69 × 43
Rate (Hz)	30	15 to 30	30
Depth			
Resolution (pixel, W × H)	640 × 480	512 × 424	1280 × 720
Angle of view (°, H × V)	68 × 51	70 × 60	87 × 58
Rate (Hz)	60	30	90
Minimum depth (m)	0.2	0.5	0.28
Maximum depth (m)	1.5	4.5	10
Shutter	Rolling	Rolling	Global

5.1.2 RGB-D Camera Selection

The Creative BlasterX Senz3D is included in the comparison as it was the only camera immediately available at project commencement. While the colour sensor meets requirements, depth falls short. The depth resolution of 640 × 480 px (H × V) is only just short of the required ⩾ 660 × 540 px (H × V); this corresponds to a pixel size of 3.4 × 3.8 mm (H × V) on the target 2.2 × 1.8 m (H × V) scene, which may prove adequate for segmentation at the cost of classification accuracy. The real issue is the limited maximum depth; at a distance of 1.5 m from the camera the signer is in a frame 2.0 m wide but only 1.0 m high; appropriate for isolated hand-shape recognition but unsuitable for sign-language recognition.

Microsoft Kinect 2.0, also known as Microsoft Kinect for Xbox One and Microsoft Kinect for Windows 2.0 is the final version of the Kinect, released in 2013 and discontinued without replacement in 2017 [128]. Kinect 2.0 was designed to observe human activities in close range and so meets many of the requirements, with the exception of low depth resolution (equivalent depth pixel size of 4.3 × 4.2 mm over 2.2 × 1.8 m (H × V) scene 30 % larger than the 3.3 × 3.3 mm maximum pixel size) and a rolling shutter, both of which may prove adequate but would compromise classification accuracy (the latter severely).

The RealSense series of depth cameras are recommended by Microsoft as a replacement

for the discontinued Kinect; the Intel RealSense D435 is the most suitable camera of the RealSense line and meets all requirements. An oddity never fully unravelled are the various specifications for depth resolution: the standard resolution given is 1280×720 px (H × V) [55], however the specification sheet also lists the 'Depth Sensor Active Pixels' as 1280×800 px (H × V) [54, p. 34] and the 'Depth Data Stream' as 848×480 px (H × V) [54, p. 54] this latter is the resolution specified as "optimal" (without rationale) by Intel in a white paper on performance [42]. Taking the highest resolution option, in combination with the wide angle-of-view means that the D435 can capture the entire 2.2×1.8 m (H × V) scene from 1.7 m; however the narrow vertical RGB AoV necessitates a camera-to-signer distance of 2.3 m.

The Creative BlasterX Senz3D must be dismissed due to it's short maximum depth. Although a Microsoft Kinect for Xbox One was available for use and came close to the requirements, having reached end-of-life presents issues for utilisation of this work: any group implementing this work, for example stakeholder Deaf Can:Do, would be required to obtain a Kinect 2.0.

As the only option that exceeds the requirements, an Intel RealSense D435i was purchased for this project (the 'i' designates the inclusion of an inertial measurement unit (IMU) that has no bearing on the current study but may prove beneficial to other projects using the camera in the future; the D435i is for this project's purposes identical to a D435 [54, Table 2-2]).

Depth Technology

It is worth mentioning the underlying mechanism by which a depth camera obtains distance from light, although the extent to which different technologies may impact the results is unknown with no discovered comparison in literature.

The Creative BlasterX Senz3D, the original Microsoft Kinect and the RealSense D435's stable-mate the D415 are *structured light* depth cameras, using a combination of a projector emitting IR and a dedicated sensor for receiving IR[2]. The projector emits a unique pattern (hence the name), reversing the distortion observed by the sensor then allows calculation of distance; this is typically computed in real-time using an internal module[27], [42], [54].

Microsoft Kinect 2.0 is a *time-of-flight* depth camera, which work by projecting pulses of IR and computing distance by the time taken for the reflected pulse to return to the

[2]Camera sensors are typically sensitive to IR so cameras contain software filters to remove it, leaving only 'visible light'; it is likely an IR sensor has, in effect, an inverted filter.

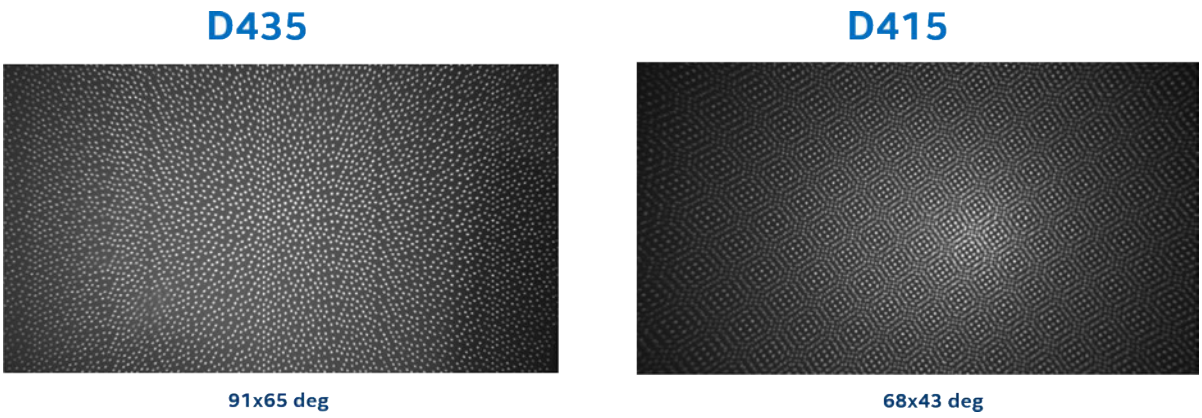


Figure 5.1: Comparison of IR projector patterns: simple 'texture' increasing pattern on left from Intel RealSense D435 where the actual pattern is not important or even necessary, versus 'structured light' pattern on the right where deformation in the pattern observed by the camera are the basis of the depth calculation, such as in the Intel RealSense D415 [42].

camera [11], [98].

The Intel RealSense D435 is a *stereoscopic* depth camera: it uses the differences in the scene observed by two IR sensors separated by a precise distance to compute distance in real-time using an onboard module ("Intel RealSense Vision Processor D4 (DS5 ASIC)"). This camera also contains a projector that emits IR in a pattern, as shown in Figure 5.1, but this is only to increase the 'texture' of the scene, improving depth performance of flat surfaces [54].

5.2 Intel RealSense D435i

The Intel RealSense D435i purchased for this project has serial number 843112070952. It came with a USB cable and miniature tripod.

The USB cable is 1114 mm long with one male 'A' type connector and one male 'C' type connector that matches the socket on the camera[3].

The miniature tripod has a ball-head and extendable legs, but is quite light compared to the D435i making the assembly easy to topple; even the stiffness of coils in the USB cable are enough to cause it to sit unevenly. For this the early parts of this study the

[3]Quality of the cable appears to be important; similar looking cables were not able to provide enough power and/or bandwidth to run all streams at once. Print on the cable claims it meets 3.1 specifications and handles 30 V; perhaps it has higher gauge wire so lower resistance?)

Figure 5.2: Intel RealSense D435i on Manfrotto Pixi Mini tripod.

tripod was replaced by a Manfrotto Pixi Mini tripod as shown in Figure 5.2

5.3 Software

To use a depth camera (or indeed, any non-standard device) a computer requires a *driver*: an interface or set of instructions that define how the device is to be controlled. For the D435i, Intel has produced a SDK they call Intel RealSense SDK 2.0 that includes device drivers, basic utility programs and an application programming interface (API) that allows developers to create their own software.

One of the bundled utilities is the Intel RealSense Viewer, shown in Figure 5.3. This is a convenient way of observing the operation of the camera and it's output as well as discovering the effects of various settings. The three optical sensors of the Intel RealSense D435i, shown in Figure 5.4 can be independently streamed and the depth stream can also be displayed.

Through playing with Intel RealSense D435i in the Viewer it was discovered that the IR projector does not have a significant effect on the depth stream, but ambient lighting conditions do. In particular, light from artificial sources and the reflections of that light seemed to 'blow out' the depth image, with some sources causing interference, perhaps in the IR. Natural light can also cause issues; while Intel specifies that natural light is actually a boon to the depth performance of the D435 [42], the variability was passed

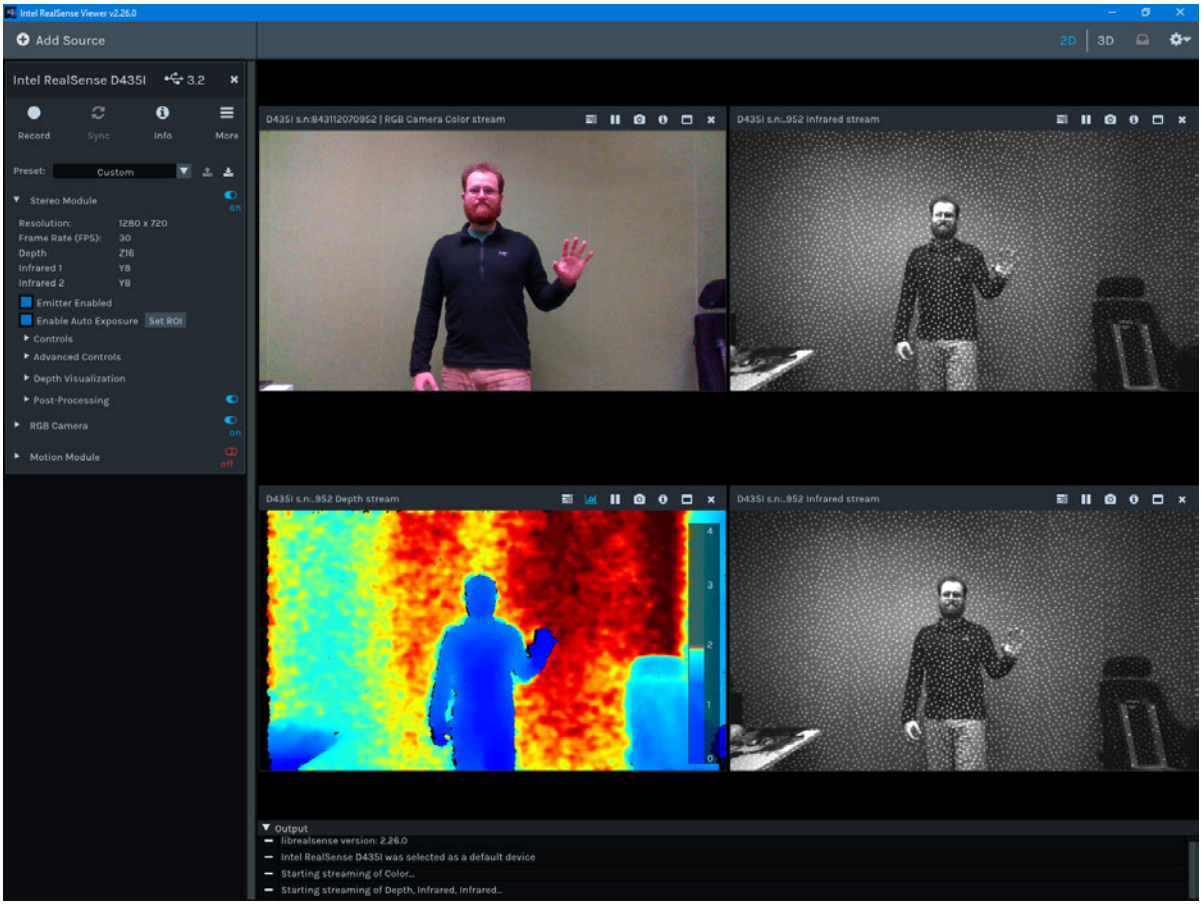


Figure 5.3: The Intel RealSense Viewer allows interaction with RealSense D400 series cameras, viewing and recording individual streams and write-access to parameters. Note the 3 streams corresponding to the 3 optical sensors of the D435: one colour/RGB (top left) sensor and stereo infra-red sensors (top right & bottom right); the 'Depth Stream' is computed in real-time from an onboard module. Also note the 'noise' in the depth data: a slight 'haze' around the silhouette, variability in the background.

Figure 5.4: Sensors of the Intel RealSense D435 [54]. From the perspective of the camera: the circle with an eight-sided gear-like shape on the extreme left is the RGB Imager; the set of concentric circles next to it is the Left (IR) Imager; the identical looking set of concentric cirles on the far right is the Right (IR) Imager and the remaining circle inside a rounded oblong is the IR projector.

onto depth readings. As a work-around, recordings with the Intel RealSense D435i were made in a small room with no windows and fixed intensity linear fluorescent lighting that was found to produced consistent depth readings.

For applications such as an SLR system, the SDK, in particular Intel RealSense SDK 2.0 [56], provides a means of developing bespoke software to regulate control of the camera. Intel RealSense SDK 2.0 is *open source*, meaning that the code from which it is compiled is freely accessible, a feature that has several benefits, foremost of which for this project is access to documentation. The SDK is primarily written for C++ but has 'wrappers' (translator code that encapsulates functions for use in another language) for several other programming languages, including C#, LabVIEW, Python & MATLAB. Due to familiarity, MATLAB was initially selected for this project.

5.3.1 RealSense in MATLAB

The latest version SDK[4] was downloaded and the Windows Installer used to obtain the pre-compiled wrapper. The result is a 'package'[5] `+realsense` that was copied to a MATLAB working directory. Importantly, for MATLAB to recognise the package, it's parent directory must be on the MATLAB *search path*; for example, if the package

[4]At the time, v2.19.2.
[5]MATLAB package folders are denoted by a '+' (plus symbol) [94]

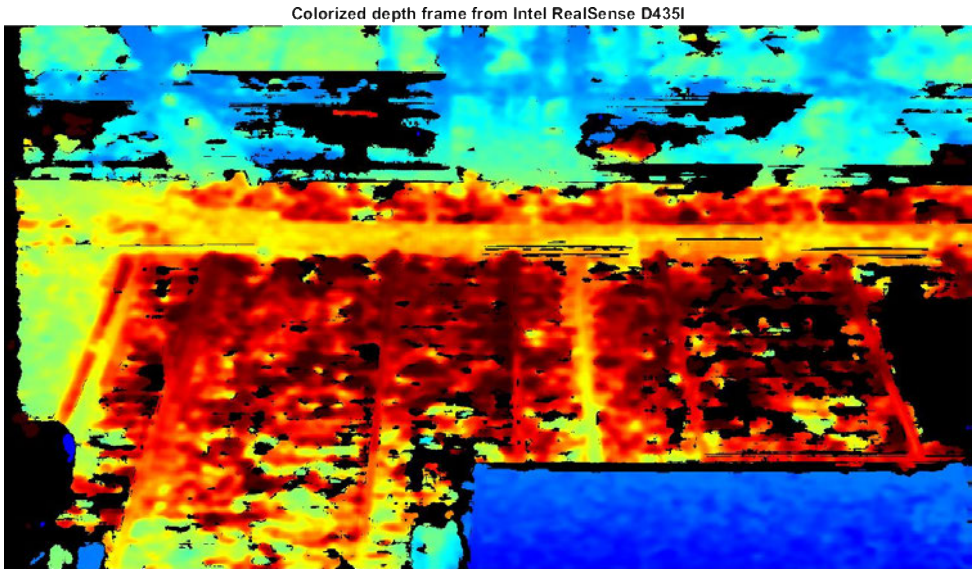


Figure 5.5: A depth frame produced by `depth_example.m` in MATLAB. Note the variable depth of the background (much red and yellow = far, some black = uncertain and bit of blue = close); these are blinds that are blocking natural light and reflecting internal light.

was located at `D:/SignToText/Matlab/+realsense`, the MATLAB command to add the package to the user search path[6] would be: `addpath 'D:/SignToText/Matlab/'`

With the package available on the MATLAB search path the simple example script `depth_example.m` from the SDK[7], shown in Listing 5.1, can be run to validate the setup, producing depth-colourised frames as shown in Figure 5.5.

Although some frames were successfully obtained from the D435i, issues began to appear, seemingly all related to the USB connection and controller. The most reliable way of producing the issues was to run the script a second time: rarely, it would work; most of the time MATLAB would give an error. After investigation, interrogating the camera's connection mode (`usb_type_descriptor`) revealed that sometimes the connection was initially as required: 'USB 3.2', but at other times it reported 'USB 2.1' which was workable, but reduced bandwidth meant the camera provided much smaller frames: 640×480 px, as shown in Figure 5.6; after calling the camera once the camera

[6] A note on directory separators: POSIX specifies '/' (forward slash), as used by UNIX-like systems (e.g. Linux & MacOS), while Windows uses '\' (backslash). Conveniently, however, Windows accepts forward slashes in input, so '/' can be used as the directory separator for most platforms (as is the goal of POSIX).

[7] https://github.com/IntelRealSense/librealsense/blob/master/wrappers/matlab/depth_example.m

```
% Make Pipeline object to manage streaming
pipe = realsense.pipeline();
% Make Colorizer object to prettify depth output
colorizer = realsense.colorizer();

% Start streaming on an arbitrary camera with default settings
profile = pipe.start();
% Get streaming device's name
dev = profile.get_device();
name = dev.get_info(realsense.camera_info.name);

% Get frames. We discard the first couple to allow the camera time to settle
for i = 1:5
    fs = pipe.wait_for_frames();
end

% Stop streaming
pipe.stop();

% Select depth frame
depth = fs.get_depth_frame();
% Colorize depth frame
color = colorizer.colorize(depth);

% Get actual data and convert into a format imshow can use
% (Color data arrives as [R, G, B, R, G, B, ...] vector)
data = color.get_data();
img = permute(reshape(data',[3,color.get_width(),color.get_height()]),[3 2
 ↪  1]);

% Display image
imshow(img);
title(sprintf("Colorized depth frame from %s", name));
```

Listing 5.1: `depth_example.m`: a script included with Intel RealSense SDK 2.0 that demonstrates the basic function of the SDK in MATLAB and thus can be used for validation [56].

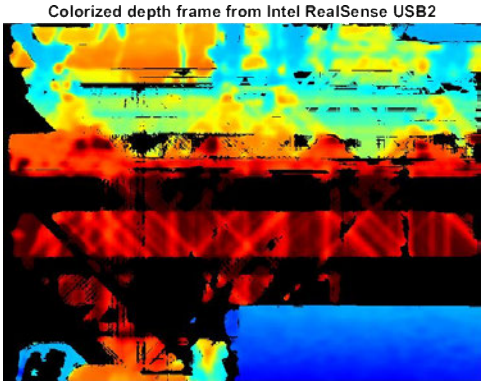


Figure 5.6: A small 640 × 480 px depth frame produced by `depth_example.m` when the D435 is connected in USB 2 mode (compared to 1280×720 px when connected in USB 3 mode), shown at relative scale to Figure 5.5.

simply was not available. To resolve the issue, restarting MATLAB, un-plugging and re-connecting the camera and restarting the computer (and permutations thereof) were tried; restarting proved the most reliable but there were cases where a second restart was required.

With the D435i proving unreliable in MATLAB, other options were considered, such as recording footage by other means (for example, using the Viewer) and using MATLAB to process the files offline. Another option would be to change language: C++ would be a sensible choice as the core language of the SDK; Python is also supported and was ultimately selected as a means of skill development.

5.3.2 RealSense in Python

The Python wrapper for Intel RealSense SDK 2.0: `pyrealsense2`, available as a PyPI distribution[8], can ostensibly be installed via Python's package manager 'Python Installs Packages' (pip) via the shell command: `pip install pyrealsense2` [56]. However, this distribution is pre-compiled using Python 2(.7), which has 'End of Life' set for 2020 [104], so for continuity reasons Python 3 was preferred.

Compiling `pyrealsense2` for Python 3 is complicated and the instructions provided by Intel are in two places: 'Python Wrapper'[9] and 'Windows 8.1 & Windows 10 In-

[8] https://pypi.org/project/pyrealsense2/
[9] https://github.com/IntelRealSense/librealsense/blob/master/wrappers/python/readme.md

stallation'[10] , that overlap and are generally vague, so the steps taken are summarized here.

Compiling Intel RealSense SDK 2.0 for Python 3 under Windows

First, the SDK source (https://github.com/IntelRealSense/librealsense.git) was cloned to a working directory: `C:/librealsense`; it could also have been downloaded as a compressed `.zip` file and unpacked to the same directory. The following programs were installed:

Python 3.6+ Target compiler, https://www.python.org/downloads/windows/

CMake 3.8+ Coordinates build process, https://cmake.org/download/

Visual Studio 2015+ Mandated build compiler, https://visualstudio.microsoft.com/downloads/, including 'Desktop Development with C++' and Windows 10 SDK 10.0.10586+.

The 'Windows 8.1 & Windows 10 Installation' instructions were followed until the step 'Compiling Librealsense with Metadata support', at which point the 'Python Wrapper' instructions were followed, from Building From Source: Windows onwards.

CMake GUI was opened by running the shell command `cmake-gui -DPYTHON_EXECUTABLE=C:/Program Files/Python37/python.exe`, the 'Source' was set to `C:/librealsense` and the 'Build' folder set to `C:/librealsense/build`. During Configuration the 'Generator' was set to 'Visual Studio 16 2019', 'Optional Platform' left at 'x64' along with all other defaults. In the big red panel, the following items were checked: `BUILD_PYTHON_BINDINGS`, `BUILD_PYTHON_DOCS` & `ENFORCE_METADATA`. Running CMake ('Generate') then produced a Visual Studio 'Solution' file: `librealsense2.sln`. The Solution was opened in Visual Studio, the 'Active Path' set to 'Release|x64' and Build initiated.

The produced package, `pyrealsense2.cp37-win_amd64.pyd`, was located in `C:/librealsense/build/Release`, along with `realsense2.dll`. The package was renamed to `pyrealsense2.pyd` and both files were moved to the Python package directory: `C:/Program Files/Python37/Lib/site-packages`.

Finally, build was verified using the interactive mode of the Python Interpreter, as shown in Listing 5.2, where the absence of an error message indicated success.

[10] https://github.com/IntelRealSense/librealsense/blob/master/doc/installation_windows.md

```
> python
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64
↪  bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import pyrealsense2 as rs
>>> pipe = rs.pipeline()
>>> profile = pipe.start()
>>> frames = pipe.wait_for_frames()
>>> pipe.stop()
```

Listing 5.2: Testing `pyrealsense2` in Python 3 by importing the package; the absence of error messages indicates success.

5.4 Accuracy & Resolution

With hardware and software cooperating, the recording environment can be considered. The primary purpose of a depth channel in SLR is for segmentation: using a 'threshold' distance from the camera to include only pixels related to the hands in the colour image, excluding those further from the camera. It is possible, however, that depth itself is valuable as a source of features for classification [86], [101], [105]. In both cases, the *accuracy*: the absolute error between reported distance and ground truth, is not important, but more critical is the *resolution*: the smallest reported change in measurement.

At the 2.3 m camera-to-signer distance determined previously for the Intel RealSense D435 to capture the full arms' reach of a nominal 2 m signer, the accuracy specified by Intel: $\leqslant 2\,\%$ at up to 2 m and 80 % AoV [54, p. 61] translates to an error of ± 46 mm or 92 mm in absolute terms, with the error tending to increase linearly with distance, as shown in Figure 5.7.

To distinguish signs where changes occur at depths corresponding to a single finger, such as shown in Figure 5.8, the depth channel would need resolution smaller than the thickness or width of the finger. Despite robust discussion on accuracy, little can be found on resolution in Intel's documentation; the most useful comment is this:

> "Alternatively, when operating at very close range, the D4xx cameras can inherently deliver *depth resolution well below 1 mm*. To avoid quantization effects, it then becomes necessary to reduce the depth unit to 100 µm, and the max range will be ~ 6.5 m." [42, p. 6]

In summary, the Intel RealSense D435 has accuracy of $\pm 2\,\%$ and a resolution of $\leqslant$

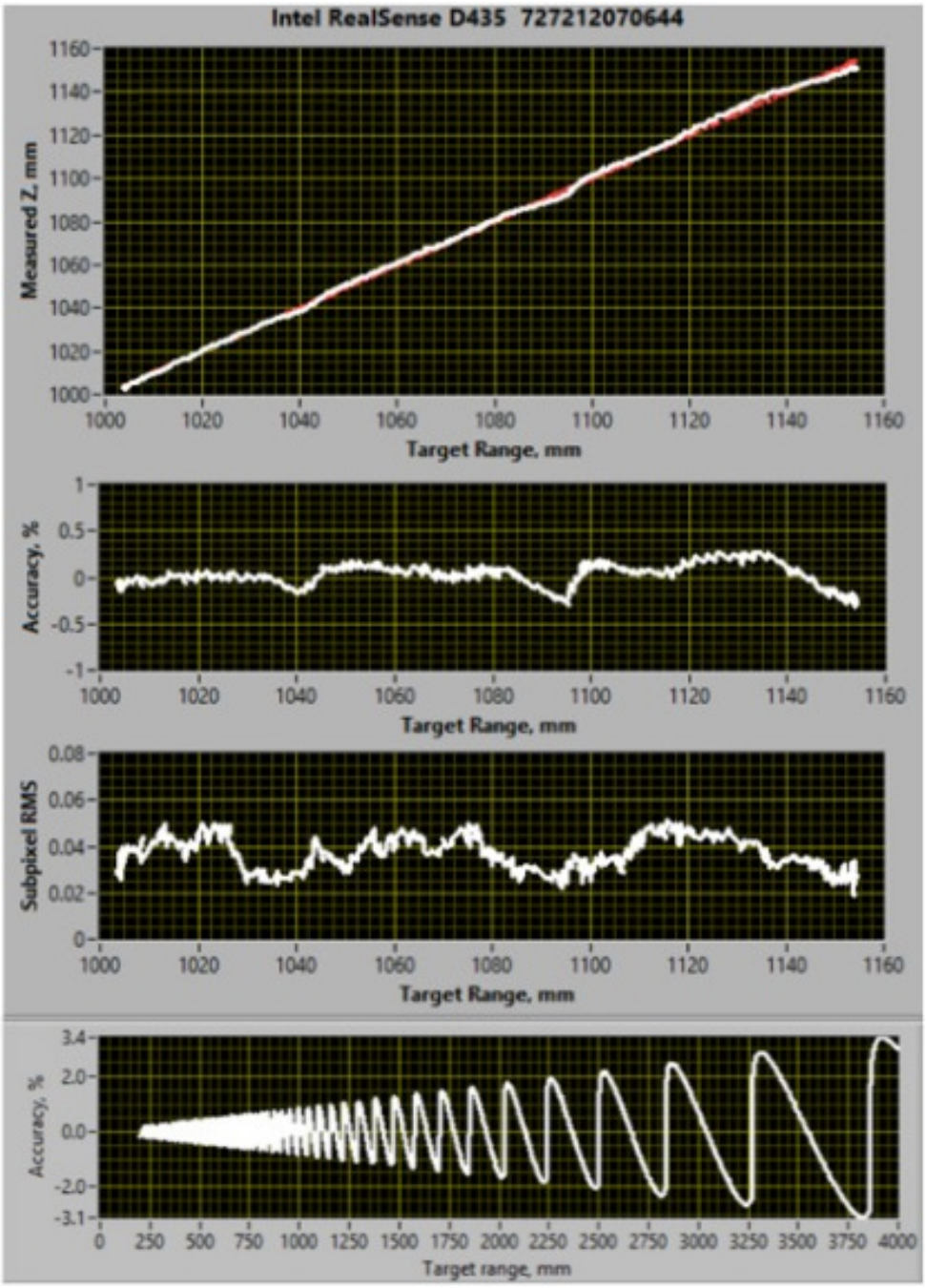


Figure 5.7: Measured depth 'Z' (mm), accuracy (%) and error (subpixel RMS) of a particular Intel RealSense D435 and theoretical error of the Intel RealSense D435 as accuracy (%) (note the change in target range: 0 to 4000 mm for the theoretical plot rather than 1000 to 1160 mm for the top 3 plots); collated from [41].

(a) L (b) N (c) V

Figure 5.8: An example of signs that differ by the use or movement of a single finger: several letters (L, N & V) of the Auslan manual alphabet [62].

1 mm; both specifications are compatible with SLR but have scope for improvement. Tests were conducted to determine if the particular Intel RealSense D435i specimen performs within that specification and potential improvements explored.

5.4.1 Testing Accuracy

As shown in Figure 5.7, the D400 series cameras exhibit linear sensitivity drift: accuracy is proportional to the reading. Simply put, the further from the camera, the less reliable the measurement. A signer standing 3 m from the camera could be measured anywhere between 2.94 m to 3.06 m and be within specification: a tolerable uncertainty of 120 mm. There are two key points of discussion here: first, foreground isolation and second, depth as a classification feature.

Foreground Isolation using Depth

The primary purpose of the depth camera is improving the performance of automated segmentation; there are computer vision techniques for identifying e.g. 'skin colour' and isolating corresponding pixels from a 2D colour image, but performance is low and highly dependent upon uniformity and consistency of lighting as well as whether any of the background contains colour similar to that defined as 'skin colour'. Depth provides 'an extra dimension'; once a feature, say a hand, has been identified, the pixels corresponding to the hand can be segmented from those of it's surroundings by comparing the depth values of those pixels.

In simple terms, the experiment can be defined such that accuracy does not impact segmentation: if the signer is standing at 3 m, the furthest the camera can measure

them is 3.06 m, so if the camera is at least 3.06 m + 2 % × 3.06 m = 3.12 m from the background, or equivalently, if the signer is standing at least 3.12 m 3 m = 120 mm from the background, then depth data can be used to distinguish between them. This distance is easily achievable, but the real problem is not so simple: the hands are likely in front of the signer's body or in contact with it; the separation distance then becomes the thickness of the hand.

For a hand of a thickness of, for example, 12 mm, at 2 % accuracy, the error would be ±12 mm × 0.5 = ±6 mm, so the camera would need to be no further than 6 mm ÷ 2 % = 300 mm from the hand for accurate segmentation; clearly this is not feasible. Another option for improving hand segmentation performance is improving the accuracy of the camera.

Manufacturer's Best-Known-Methods for Depth Performance

As a means of improving performance, Intel recommends: setting the camera to the 'optimal' depth resolution and using low gain and checking for good exposure, manually adjusting if necessary [42].

Intel states the optimal depth resolution for the D435 as 848 × 480 px, yet as justification states *"The higher the input resolution, …, the better the depth precision"* [42, p. 1]. The D435 has an 'active sensor pixel' resolution of 1280 × 800 px [42, p. 9] and supports a resolution of 1280 × 720 px, so the stated 'optimal' resolution is not the highest input resolution[11].

Appropriate exposure means that the light received by the camera sensor corresponding to the object of interest uses as much of the light intensity range of the sensor as possible, rather than 'saturating' at full or no illumination, losing all distinction. To this end, recordings were conducted in a well illuminated room but relied upon automatic exposure compensation.

Should camera performance not meet the specified accuracy, Intel recommends calibration. An individual named Calvert has developed their own calibration tool, claiming "accuracy can be improved by an order of magnitude at 2.5 metres and becomes almost linear in the depth" [13, p. 1].

[11] It was realised while writing this that the 848 × 480 px likely corresponds to the output of the depth module, so presumably there is some scaling involved to match the resolution to that of the source IR sensors; this would mean that 848 × 480 px is the 'raw' depth resolution.

Image removed due to copyright restriction.

Available to view online:
https://www.calvert.ch/maurice/improving-the-depth-map-accuracy-of-realsense-cameras-by-an-order-of-magnitude/

Figure 5.9: Plot of average Z (depth) error (mm) against distance for Intel RealSense D435: specification (grey, middle line), measured/stock (red, top line) and after Calvert calibration (green, bottom line) [13].

Calvert's Calibration Tool

The Calvert RealSense Calibrator [14] uses the depth stream for calibrating depth rather than the colour sensor, as per the Intel Calibration Tool [55] and averages multiple measurements to obtain nearly-linear accuracy, as shown in Figure 5.9.

Calibration requires a target be produced precisely for accurate calibration, with the instructions suggesting printing five copies of the target on a sheet of paper and fixing them to a flat board. For greater precision than manually-positioned and squared targets, the entire target was drawn in computer software in vector format, as shown in Figure 5.10. The target was printed on an A1 sheet of Tyvek (flash-spun non-woven high-density polyethylene, providing high dimensional stability and durability) at high precision by a commercial printer. Finally, the target was fixed to a board and clamped to a stand, as shown in Figure 5.11, then, using a bubble level, the target was levelled horizontally and vertically.

The camera was installed on a floor-standing tripod with three-axis angle adjustment

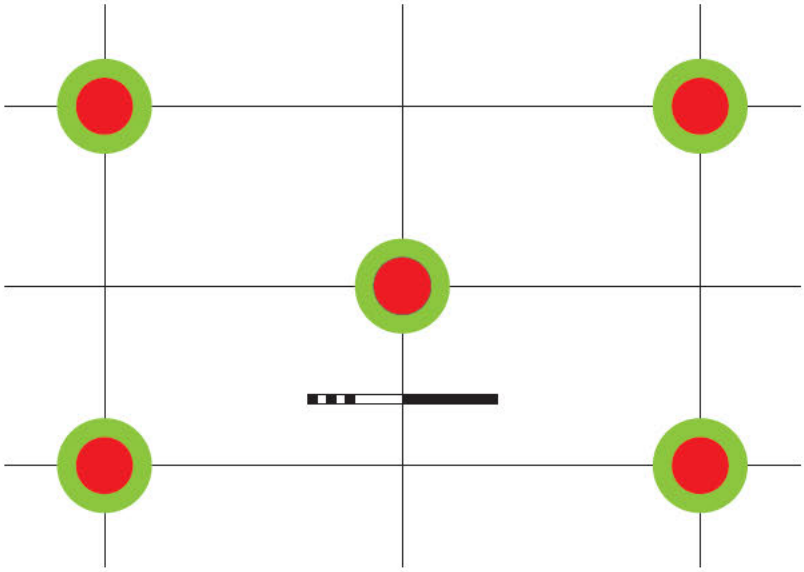


Figure 5.10: Scalar vector graphic of calibration target drawn as per [14].

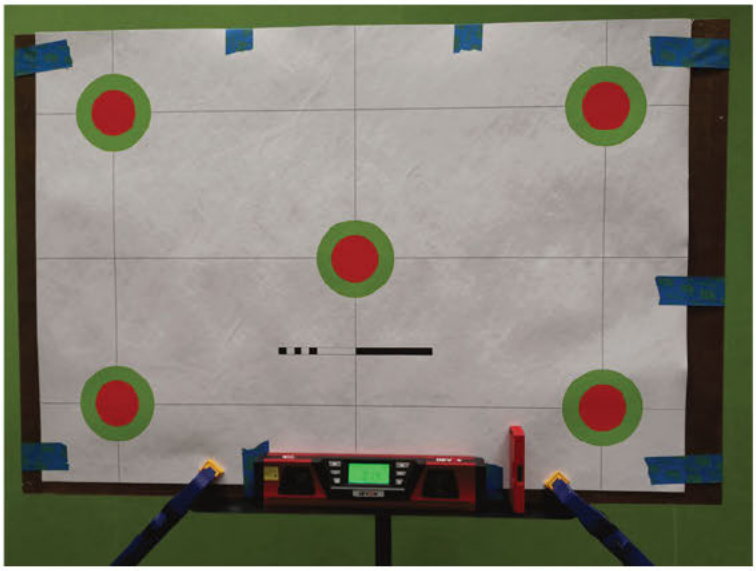


Figure 5.11: Calvert calibration target fixed to a board, levelled vertically and horizontally on a stand. The slight shadows visible across the sheet show it is not perfectly flat; this may have caused issues had the calibration tool worked.

Figure 5.12: Diagrammatic derivation ground truth depth Z: the distance between the scene and the depth sensor of the Intel RealSense D435. Note that the sensor is displaced behind the front glass a distance $Z\prime = 4.2\,\text{mm}$ [54].

and levelled horizontally and vertically. Using a tape measure, the camera was positioned such that the ground truth depth Z was $2000\,\text{mm} \pm 1\,\text{mm}$, accounting for the 4.2 mm distance between the depth start point and the front glass [54] shown in Figure 5.12. Using real-time visual feedback, the camera was positioned orthogonally to the centre of the target; this proved challenging and the results were likely out by some few millimetres and a degree or so both vertically and horizontally.

At this point the software [14] was run, gave an error and crashed. The issue appeared to be related to building from source, potentially resolvable but likely requiring considerable learning. With limited time left for the project and stock accuracy adequate for at least background segmentation, Calvert calibration was abandoned.

5.4.2 Testing Resolution

With the settings adjusted appropriately: `depth_unit=100`, the D435 should offer ample depth resolution of around 100 µm: more than enough to distinguish between finger widths an order of magnitude larger; indeed, even with `depth_unit` at it's default setting the depth resolution is more than adequate. Unfortunately, the depth resolution is still bounded by individual pixels, with a lateral resolution of around 3 mm that will limit distinction.

To test if actual resolution matched promised resolution, a resolution test board was

conceived that used nominal 8 × 19 mm dress-all-round timber that produces depth changes of 3, 8, 11, 16 & 19 mm across lateral spacings of 8 & 19 mm; the technical drawing of the board is shown in Figure 5.13 and a photograph of the produced board in Figure 5.14.

The board was positioned on a stand with the strips running horizontally and, using bubble levels, adjusted until it was level horizontally and vertically. The Intel RealSense D435i was positioned on a tripod, set 1000 mm from the basal platen of the board to the glass of the camera and, using bubble levels, set level horizontally and vertically. Using real-time visual feedback via the RGB sensor, the camera was translated relative to the width of the board and rotated until they appeared to be coplanar.

With the Intel RealSense Viewer version 2.25.0 set as shown in Table 5.3, a brief recording was made. From the recording, a single depth frame was taken after allowing a few seconds for any automatic settings to stabilise (e.g. auto exposure), manually cropped to a region-of-interest (ROI) within the board and then depths for an arbitrarily-selected single column of pixels (intersecting all wooden strips) were extracted. The column was then down-sampled by retaining every 12 value such that the remaining number of values corresponded with the height of the board in mm (i.e. one value for each mm).

The column of depths was then converted from floats in metres to truncated integers in millimetres and the step-change between pixels was then calculated.

A plot of the raw data showing measured distance (in RealSense 'depth units') against vertical displacement (mm), overlaid with the profile of the resolution test board is shown in Figure 5.15; it shows that there is some general agreement between where transitions occur and changes in depth, but there appears to be no correlation between the magnitudes of the change (real-to-measured) and there are changes in depth where there is no change in the board.

The distance (float, m and integer, mm) and measured step (mm) values for the first 15 pixels are shown in Table 5.4, along with the full sequence of expected step-changes from the physical board. A stem-plot of the measured step-changes is provided in Figure 5.16. Interestingly, there are many small steps, rather than the theoretical result: infrequent large steps separated by many zero steps. As each pixel represents a lateral translation of around 3 mm, these results indicate that changes are not 'abrupt'; the D435i is not producing high-frequency changes in depth, but rather smoothing the depth changes out.

From these findings it is clear that accuracy is low, with considerable fluctuation the depth results that could cause issues in segmentation and that although depth resolution

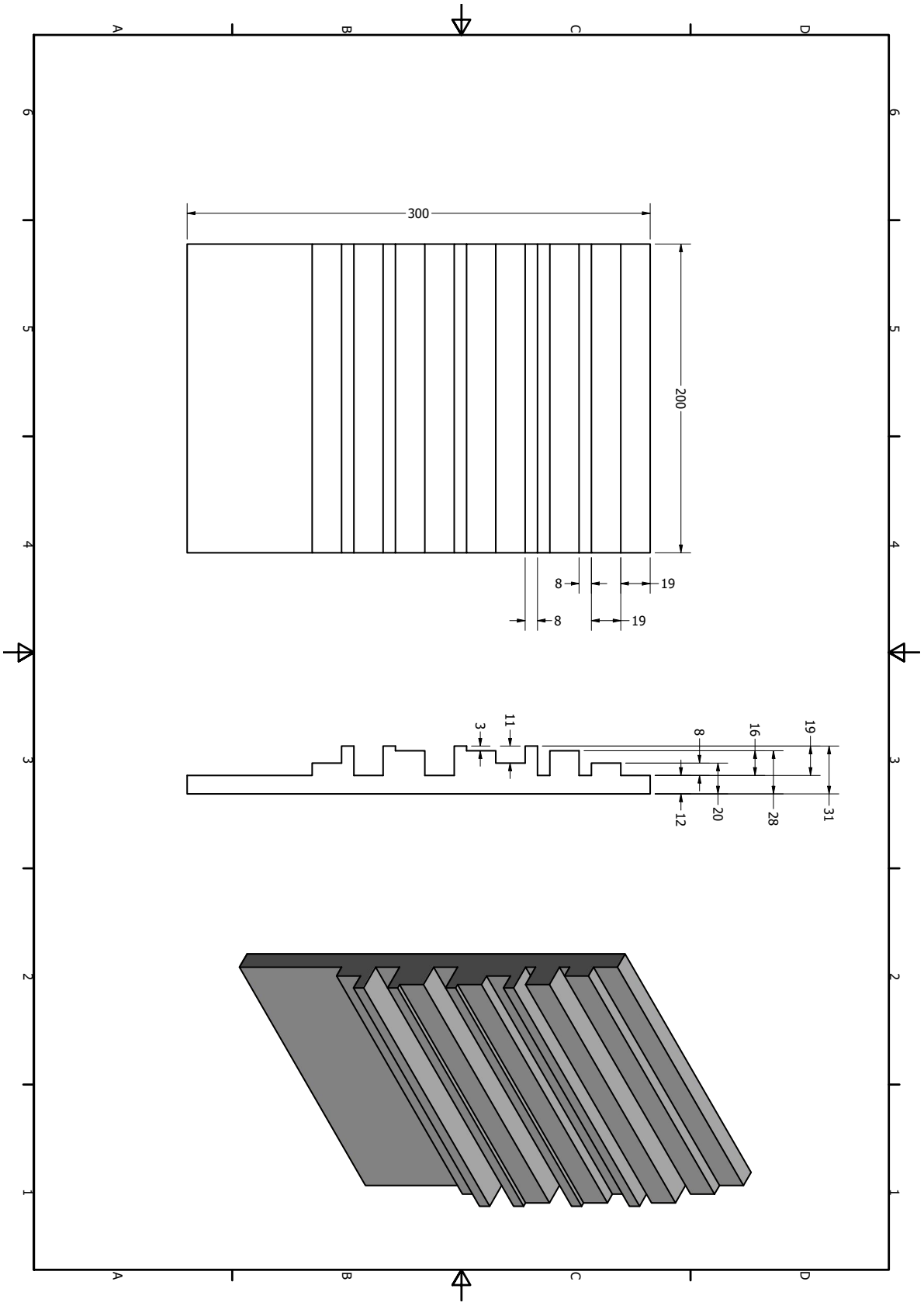


Figure 5.13: Drawing of resolution test board.

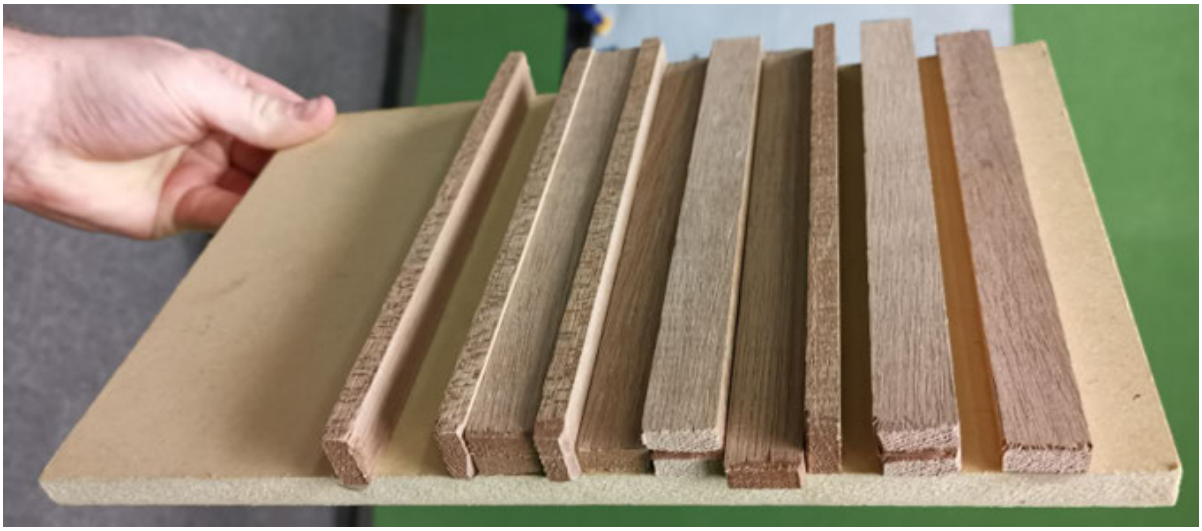


Figure 5.14: Photograph of actual resolution test board.

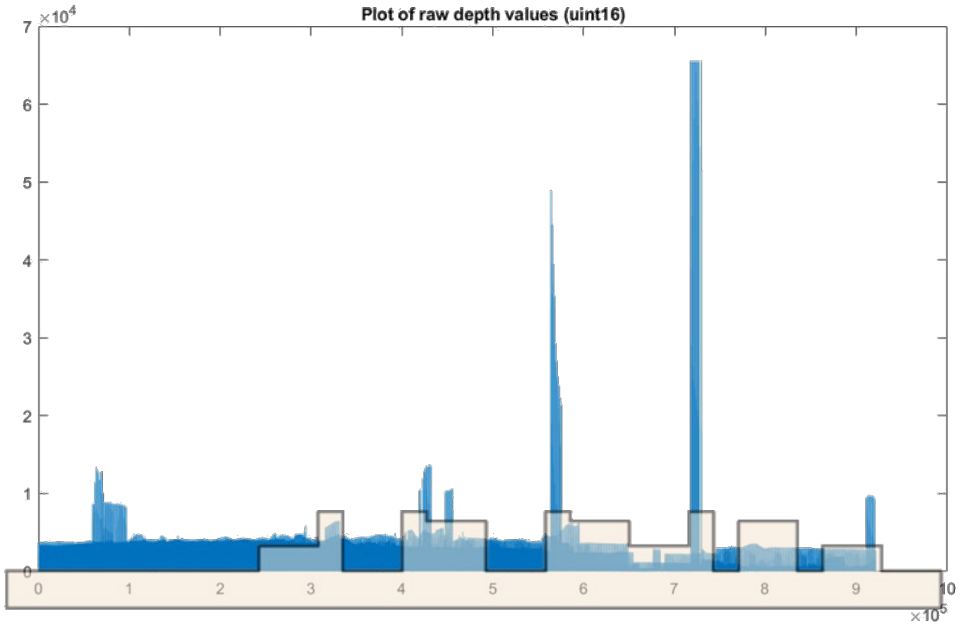


Figure 5.15: Stem plot of raw depth (RealSense 'depth units') against vertical displacement (mm) overlaid with profile of resolution test board.

Table 5.3: Intel RealSense Viewer settings during resolution test.

Setting	Value
Stereo Module	
Resolution	1280 × 720
Frame Rate	6
Depth	Z16
Infrared 1	Y8
Infrared 2	Y8
Emitter	Enabled
Auto Exposure	Enabled
AE ROI	Manually selected ROI within board
Controls	
Depth Unit	0.0010
all others	as per defaults
Advanced Controls	
all	as per defaults
Depth Visualisation	
Visual Preset	Fixed
Color Scheme	Jet
Histogram Equalization	Disabled
Min Distance	0.95 m
Max Distance	1.05 m
Post Processing	
all	Disabled
RGB Camera	
all	Disabled
Motion Module	
all	Disabled

Table 5.4: Depth values from resolution test: first 15 values, top-to-bottom of distance (float, m), truncated distance (integer, mm) and measured step change (difference between this pixel and one below it, mm) compared against expected step change of board profile.

Float (m)	Integer (mm)	Measured Step (mm)	Expected Step (mm)
1.01800005	1018	-2	8
1.01600005	1016	-3	-8
1.01300005	1013	-3	16
1.01000005	1010	-2	-16
1.00800005	1008	-3	19
1.00500005	1005	-2	-11
1.00300005	1003	-1	8
1.00200005	1002	-2	3
1.00000005	1000	-1	-19
0.99900005	999	-1	16
0.99800005	998	-1	3
0.99700005	997	-1	-19
0.99600005	996	-1	19
0.99500005	995	0	-11
0.99500005	995	-1	-8

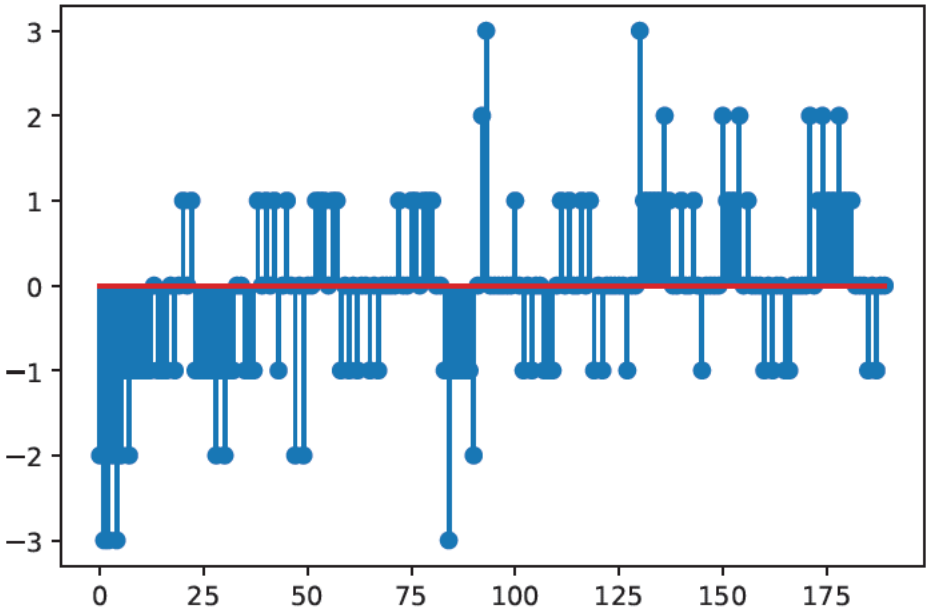


Figure 5.16: Stem plot of the step-change in measured depth for points within the target; horizontal axis is distance down the board; vertical axis is step-change of depth (mm) from one pixel to the one below it.

per se is adequate at 1 mm, depth is smoothed, perhaps by the low lateral resolution, such that there are no large step changes; accordingly, depth data as it stands may not prove particularly powerful for classification.

5.5 Recording

An offline trial was targeted in order to make a fair, scientific comparison of methods from pre-processing, through feature extraction and to classification. To do so the capture must occur separately and be recorded, ideally in a manner that can be re-loaded as if it was streaming live from a camera. During the difficulties of communicating with the Intel RealSense D435i in MATLAB, other options were considered, such as recording separately; the Viewer included in the Intel RealSense Viewer also functions as a recorder, so was used as a simple means of achieving the recording goal with little fuss...

5.5.1 Recording via RealSense Viewer

The Intel RealSense Viewer provides means for adjusting camera settings, saving/loading those settings to file for repeatability and recording to file, thus, Viewer met all requirements. A trial recording saved silently (without prompt) to `%HOMEPATH%/Documents/20190824_114547.bag`; the save directory can be changed in Viewer settings.

The `.bag` extension denotes a 'bag' file: a container-type developed as part of the Robot Operating System [103]. The reason for Intel's adoption of the bag file here can only be speculated upon, but the bag file does have the highly desirable property of supporting play black [103]. The RealSense SDK provides an example[12] for reloading a recording session from a bag file in Python; the key element being the function `pyrealsense2.config.enable_device_from_file( pyrealsense2.config(), 'bag_path_as_string')`, from which point the code is identical to streaming directly from a camera.

In summary, the Viewer utility included in the Intel RealSense SDK can be used to record footage streaming from the D435i depth camera and save the data to a 'bag' file, which can then be used to 'play back' the stream, facilitating comparative offline studies.

5.6 Prompting

The role of the prompter is to provide the participant with a cue of which sign to perform. A lexicon must be provided, along with number of replicates per lexeme, with classification robustness increasing with number of replicates. To reduce confounds in recordings it is appropriate to present cues in a random order, however the resulting sequence of labels should be recorded. Finally, the prompter must consider the timing of the cues. A script, `prompter.py`, shown in Listing B.1, was written in Python 3.

The script specifies a `lLexicon`, a bespoke `Lexicon` class object containing a list of lexeme labels and paths to corresponding images (in hindsight, the class was redundant; a simple Python Dictionary data structure would suffice.) Using the labels of `lLexicon`, along with the number of replicates per lexeme `nReplicates`, a list of strings `lsTrials` is created that contains each lexeme label repeated the appropriate number of times in random order.

[12] https://github.com/IntelRealSense/librealsense/blob/master/wrappers/python/examples/read_bag_example.py

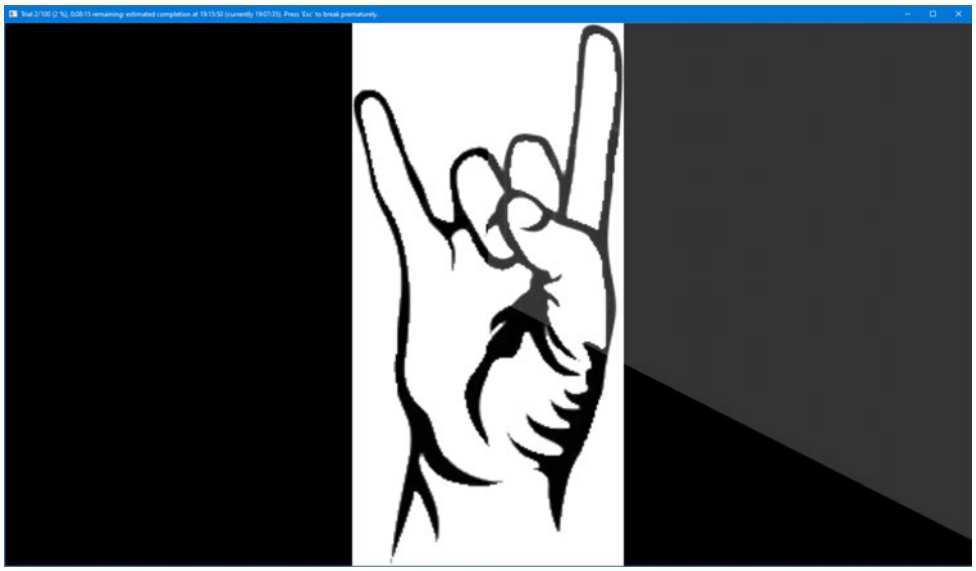


Figure 5.17: Example teleprompting produced by `prompter.py`, showing visual cue (in this case, the */animal/* handshape) and the opaque countdown animation (currently around the '20 seconds' mark).

The script uses the OpenCV package to create a window, shown in Figure 5.17, in which the lexeme images are displayed and handle interrupts. It also allows for specification of which monitor the prompter is to be displayed on, hence, if the computer system has multiple monitors, the prompter and recorder can run on separate monitors.

The prompter shows a 'placeholder', shown in Figure 5.18, while it waits for the user to start the session; this allows, for example, the user to initiate the recorder and the participant to get into position. The session is started by pressing Enter; facilitated in this study by a foot switch such that the user can control and participate at the same time. The session can be terminated early by pressing Escape.

Once the session begins, the prompter shows the first lexeme image and begins a countdown, displaying a partially translucent overlay that sweeps like the hand of a clock, from the 12 O'Clock position in a clockwise manner until it covers the lexeme. This overlay provides the user with an indication of time remaining at that lexeme while inducing minimal cognitive load. Once the countdown is complete the screen is blanked, allowing time for the participant to lower their hands to a neutral position, reducing sign-sign interaction and possibly encouraging more repeatable performances.

Timing is controlled by `nSecondsPerCountdown`, the amount of time for which the image for that lexeme is displayed and `nSecondsPerTrial`, the amount of time from the

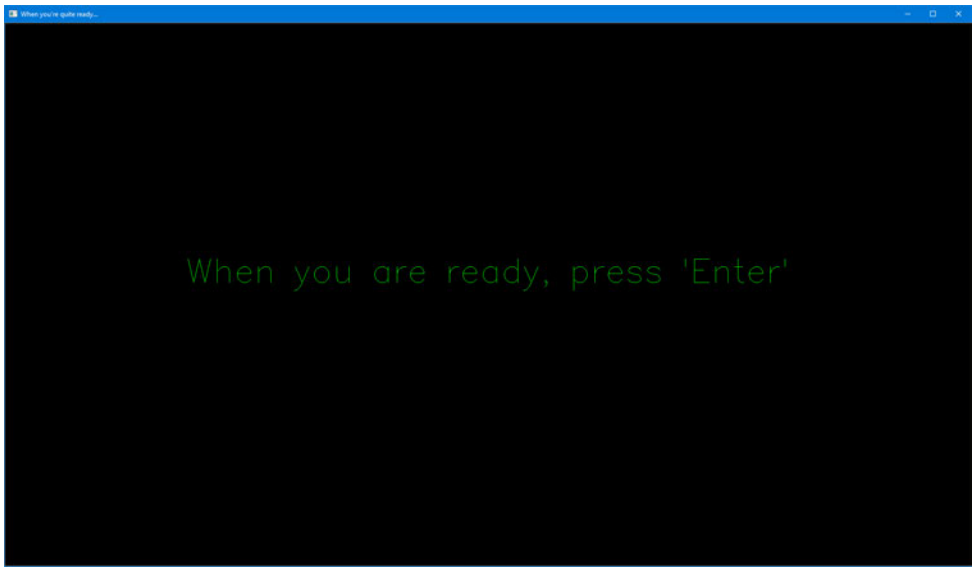


Figure 5.18: Placeholder produced by `prompter.py`.

start of one lexeme image until the display of the next. The screen blank time is the intervening time (`nSecondsPerTrial` - `nSecondsPerCountdown`). The overlay increases by 1° every 60th of `nSecondsPerCountdown`.

For largely unexplored reasons, observed countdown timing exceeds that specified by a large zero offset (around 1 s) and a small linear offset; this was managed by setting `nSecondsPerCountdown` lower than the calculated time by, for example, 1 s.

The prompter saves the session details including a list of lexemes as they were presented to a log file. In anticipation of the eventual development of an automatic temporal segmentation routine, such as using the first derivative of optical flow, the log also provides detailed information regarding timing.

5.7 Skeleton

One of the major advances of consumer grade depth cameras are skeleton models [116] that provide 3D coordinates of 'joints'. These joints can be used for both for pose estimation (one of the target phonemes) as well as to locate regions of interest, such as the hands. While the RealSense SDK and Creative BlasterX Senz3D used at the start of this study provided a skeleton model, Intel RealSense SDK 2.0 *does not*.

Intel suggested users acquire a skeleton model from 'middleware', recommending NuiTrack (a contraction of 'natural user interface' and 'tracker'), proprietary software that

offers a free trial. The trial limits recording duration to three minutes, but that was deemed adequate (and could certainly be worked around). NuiTrack runs in either of the game engines Unreal or Unity; both are proprietary but free for students. The prior is scripted in C++ and the latter in C#. Unity was selected somewhat arbitrarily, largely based on chance to learn C#.

Both NuiTrack and Unity were installed as per manufacturer recommendations; however, despite overcoming several roadblock errors, NuiTrack could not be made to work. In the interest of time, NuiTrack and, hence, the Intel RealSense D435i, were abandoned.

5.8 Microsoft Kinect

Although Microsoft Kinect 2.0 has a far smaller depth sensor than the Intel RealSense D435: 512×424 px versus 1280×720 px, only 23.5 % of the area and a rolling shutter, the Kinect for Windows SDK 2.0 provides a 25-joint skeletal model, as enumerated in the SDK in Listing 5.3 includes hand 'joints' and, critically, was available for use immediately.

It was discovered that for Microsoft Kinect 2.0 to work with Microsoft Windows 10, the Registry must be modified, as shown in Listing 5.4.

The Kinect for Windows SDK 2.0 is natively scripted in C++ so for familiarity, a series of tutorials were followed and extended to overlay the RGB frame with lines connecting arm joints and triangles connecting hand 'joints', as shown in Figure 5.19.

5.8.1 Recording Kinect

Like the Intel RealSense SDK 2.0, Kinect for Windows SDK 2.0 includes a viewer application, Kinect Studio, shown in Figure 5.20 that has the ability to record directly, saving the inevitable API issues[13]. After fastidiously setting up the recording space a trial recording was performed and some peculiarities were noticed of the output file.

By default, the file saves silently to `%userprofile%/Documents/<date>.xef`. The location was changed in the settings but there is still no success dialogue. The `xef` file extension indicates a proprietary Microsoft file format that can only be read back inside Kinect Studio. Several attempts were made using various applications and instructions from online forums to access the recording from inside Kinect for Windows SDK 2.0 in C++ but without success.

[13]Technically, the Intel SDK Viewer is 'like' the Microsoft SDK viewer which is around 4 years senior.

```
typedef enum _JointType
{
    JointType_SpineBase = 0,
    JointType_SpineMid = 1,
    JointType_Neck = 2,
    JointType_Head = 3,
    JointType_ShoulderLeft = 4,
    JointType_ElbowLeft = 5,
    JointType_WristLeft = 6,
    JointType_HandLeft = 7,
    JointType_ShoulderRight = 8,
    JointType_ElbowRight = 9,
    JointType_WristRight = 10,
    JointType_HandRight = 11,
    JointType_HipLeft = 12,
    JointType_KneeLeft = 13,
    JointType_AnkleLeft = 14,
    JointType_FootLeft = 15,
    JointType_HipRight = 16,
    JointType_KneeRight = 17,
    JointType_AnkleRight = 18,
    JointType_FootRight = 19,
    JointType_SpineShoulder = 20,
    JointType_HandTipLeft = 21,
    JointType_ThumbLeft = 22,
    JointType_HandTipRight = 23,
    JointType_ThumbRight = 24,
    JointType_Count = (JointType_ThumbRight+1)
}
```

Listing 5.3: Kinect for Windows SDK 2.0 joint type enumeration definition [97]. There are several 'joints' per hand: wrist, hand, tip and thumb for each left and right hand, which, while clearly not joints in the true sense of the word (or even as defined in the API: "Connects two bones of a skeleton" [96]), these coordinates are very useful for automatic segmentation.

```
Windows Registry Editor Version 5.00

; Fixes Kinect infinite disconnect/reconnect issue on Windows 10
; https://support.microsoft.com/en-us/help/4032123/kinect-sensor-is-not-recognized-on-a-surface-book

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Class\{36fc9e60-c465-11cf-8056-444553540000}]
"LowerFilters"=""
```

Listing 5.4: Windows 10 Registry modification to enable USB power support for Microsoft Kinect 2.0.

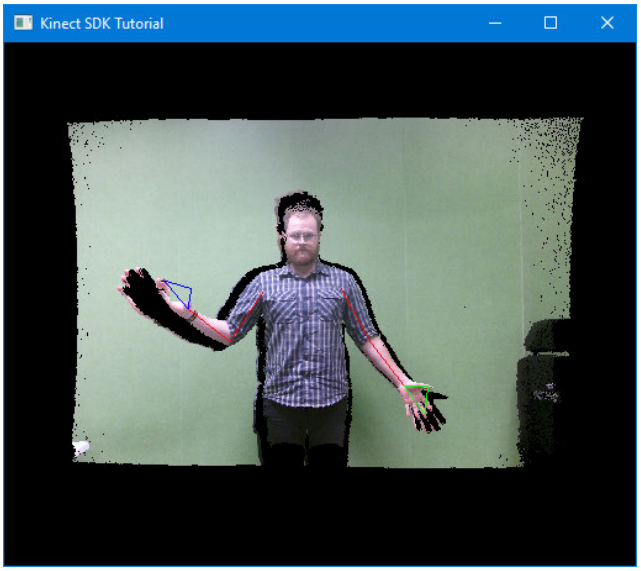


Figure 5.19: Kinect skeleton joint overlay on 'coloured pointcloud' (RGB pixel values mapped to 3D coordinates in synthetic 'camera' space); triangle vertices correspond to `Wrist<Side>`, `HandTip<Side>` & `Thumb<Side>`, coloured green for the left side and blue on the right-hand-side. Note low-quality output with substantial shadow and artefact, particularly the right hand and estimated joint in space.

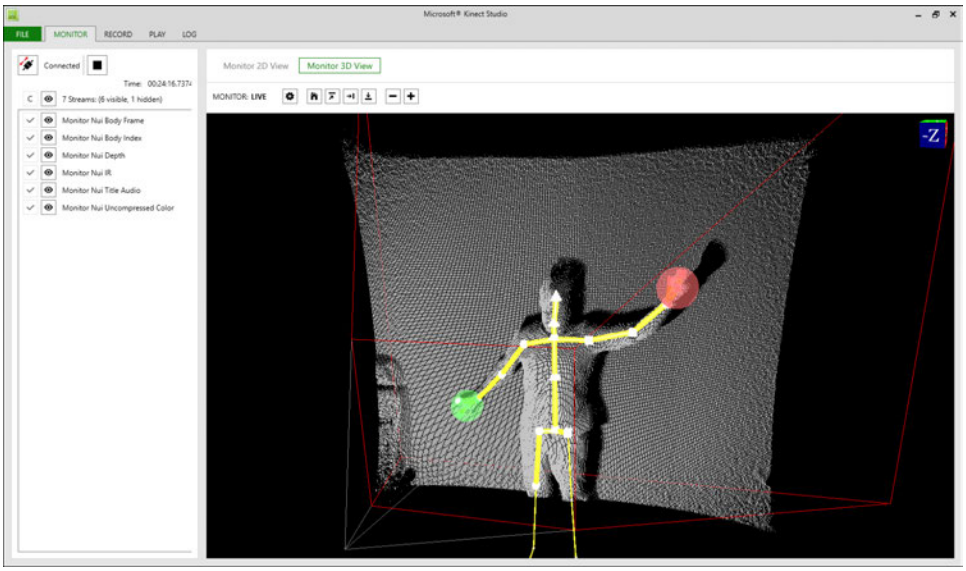


Figure 5.20: Kinect for Windows SDK 2.0 viewer: Kinect Studio, showing pointcloud view with connected-joint 'skeleton' (yellow) and hand tracking (red and green circles).

A successful attempt involved using a third-party utility to convert the Kinect Studio data file (`.xef`) into a Matlab data file (`.mat`) [115]. While the utility did (eventually) work, the resulting file only contained an array of depth frames, an array of infra-red frames and a vector of timestamps: *no colour frames or joint data.*

5.8.2 Kinect in MATLAB

The final approach to obtaining data from Kinect was to record in MATLAB directly, using Kin2 [124]. Although it can be downloaded precompiled from GitHub[14] the package does not work 'out of the box'. After investigation, the culprit was found to be the MEX file. The MEX file was eventually rebuilt following the author's instructions [123], that *absolutely require* use of the Visual C++ compiler from Microsoft Visual Studio 2012 onwards (MinGW-64 cannot be made to work, despite assurances to the contrary [69].

A MATLAB script, shown in Listing B.2, captures the Microsoft Kinect 2.0 data to a as a vector of `struct` that is finally saved to disk.

When run, the script attempts to connect to a Microsoft Kinect 2.0, creating a `Kin2` object `oKin2` that streams colour frames, depth frames and body data. Two figure windows are created, one showing the RGB stream overlaid with joints, the second the

[14] https://github.com/jrterven/Kin2

depth stream. This allows the user to ensure the participant is being correctly recognised and is well within the frame. Once the session is ready to begin, the participant initiates recording by forming a fist with their right hand (ideally, this would be programatically set to the non-dominant hand, rather than hardcoded). The session can be terminated early by pressing Escape.

The recording session duration `dSession` = `nSecondsPerTrial` × `nTrials` + `dBreathingRoom`, where `nTrials` = `nSigns` × *nReplicates*. `nSigns`, `nReplicates` (per sign) and `nSecondsPerTrial` are manually set in accordance with the Prompter. `dBreathingRoom` is a temporal buffer to account for peace of mind, arbitrarily set to 5 s (this has not proved necessary and may be omitted).

The capture rate is set at `nSamplesPerSecond = 2`, selected somewhat arbitrarily as having sufficient temporal resolution to ensure there are a few stable frames in a sign dwell time of around 3 s (set in the Prompter). Samples are taken using a `for` loop, using MATLAB's `cputime` and `pause` to control timing while accounting for time taken in sample acquisition and containing a 'timeout' condition.

For each sample, `oKin2` returns: the colour frame, as a 1920 × 1080 × 3 array of 8-bit unsigned integers, the depth frame, as a 512 × 424 array of 16-bit unsigned integers and body data as a `struct`. These, along with a millisecond-resolution datestamp, are saved (out) to a sample-indexed row (r) vector of `struct` (u): `ruOut`.

Kinect Registration in MATLAB

One of the glaring omissions of the Kin2 wrapper is an offline method for *registration*: the mapping between colour and depth frames such that the depth (distance from camera) can be defined for any given pixel on the colour frame; as such, registration is essential for depth-based spatial segmentation of the colour image. There are online methods, such as `Kin2.mapDepthPoints2Color`, however these require a Kinect to be connected and take about 20 s to align all pixels in the (far faster) depth to colour direction.

As the RGB & D sensors are unable to move relatively and the light received by the sensors has no bearing upon their spatial relationship, it was deemed reasonable to assume that depth-pixel-to-colour-pixel mapping should be constant. A map was made 5 times, with the output being identical between two pairs of maps and generally being close between all five maps, although no real statistical analysis was conducted. A concession to this variability was to incorporate the map generation into the recording script, such that each recording session would have a map created for that camera and

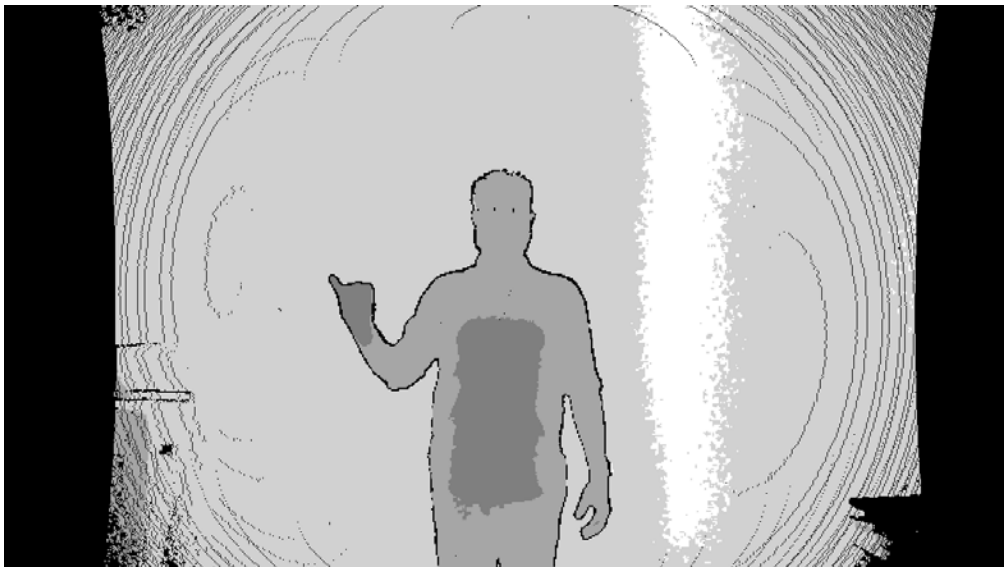


Figure 5.21: Image produced by mapping the 512 × 424 depth frame's intensity values onto the 1920 × 1080 pixel colour frame, histogram equalized for clarity. Note the limitations to usable space due to the curious banding (perhaps an artefact of different lens curvature?) and discrepancy between sensor aspect ratios.

that setup.

The code shown on lines 187 to 193 of Listing B.2, produces a depth-frame-sized, two-dimensional array of integer coordinates to the corresponding pixel in the colour frame. Code on lines 195 to 209 then invert the map, providing a quick way to access the depth corresponding to a given colour pixel. Aligning the depth frame intensity values onto the colour frame provides the image in Figure 5.21. Both maps are saved to disk alongside the recording data.

5.9 Temporal Segmentation

The ultimate realisation of SLR includes temporal segmentation where the system observes both dynamic changes of motion in time and from that determines when to capture static snapshots. This implementation of SLR simply uses manual selection of a single static representation of a sign. This temporal segmentation is achieved by a bespoke MATLAB 'AppDesigner' GUI `SelectFramesFromRecordings.mlapp`, shown in Listing B.3. The initial screen of the GUI is shown in Figure 5.22.

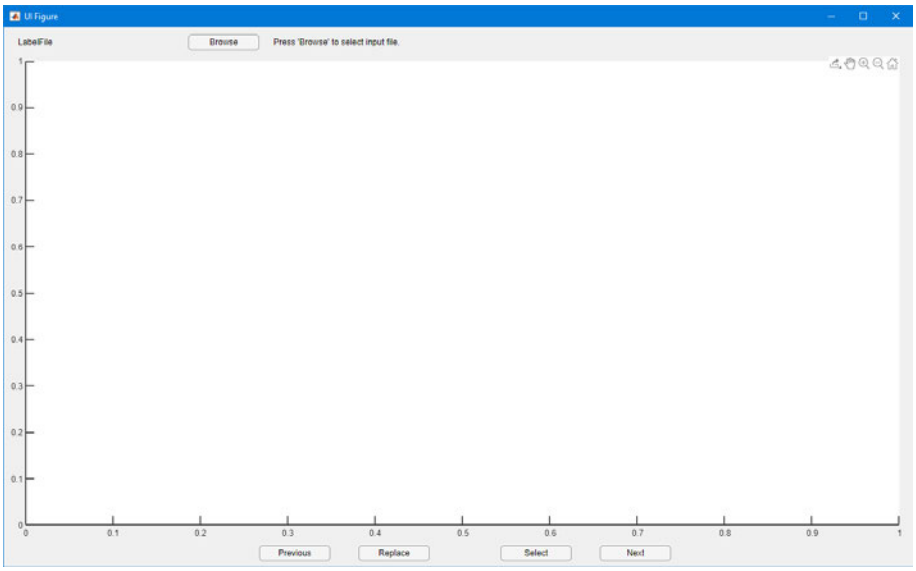


Figure 5.22: The initial screen of `SelectFramesFromRecordings.mlapp`

The GUI provides a graphical file browser via Browse to select a target output `mat` file from `KinectRecorder.m` that is then loaded and stored as a global variable: `app.ruRec`; the selected path is displayed in the status text and the first colour frame is then displayed, as shown in Figure 5.23.

The user is provided controls to move forwards Next or backwards Previous through frames; the current frame number is displayed in the status text. When they are satisfied the current frame is a good instance of the sign (e.g. little or no motion blur, well framed), they may Select it, appending the corresponding `struct` from `ruRec` to the selections `app.ruSln`.

If the user makes an error in selection it may be reversed by navigating to a preferable frame and pressing Replace. While this allows recovering from a mis-click by replacing it with the next correct frame it relies upon user memory to press the correct button and will not work for the final sign. An appropriate extension to the app would be the inclusion of a 'filmstrip' of selected frames that could be navigated by and deleted.

When the app is closed, the output file is saved to the source directory the user browsed to initially. As MATLAB is unable to save files $\geqslant 2$ GB using the default MAT-file version (7) and the files produced may be tens of GB so the MAT-file version is set to 7.3 using the tag (`'-7.3'`) [92].

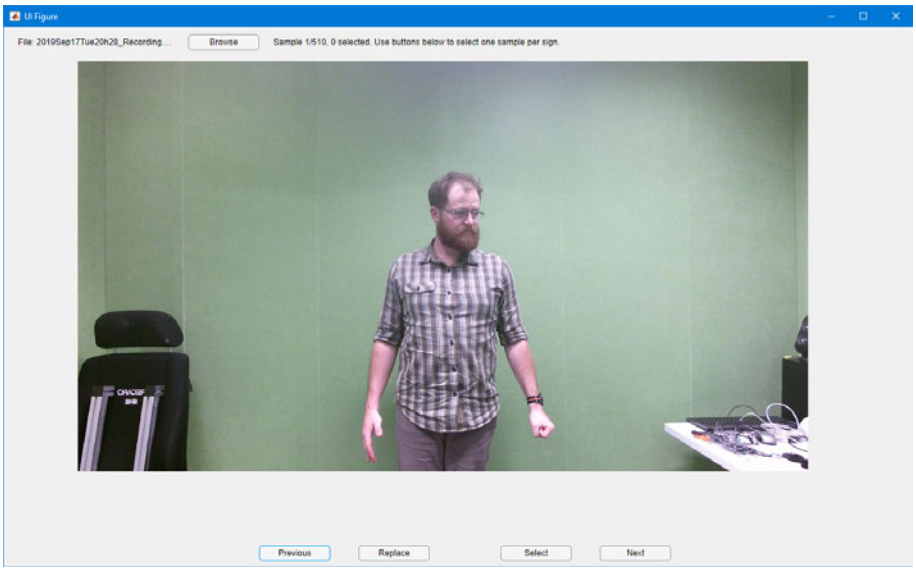
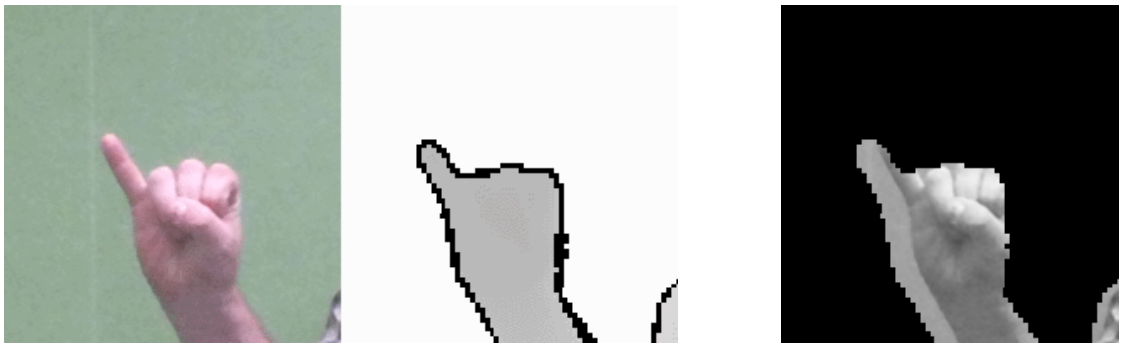


Figure 5.23: The GUI `SelectFramesFromRecordings.mlapp` showing the first colour frame of a MATLAB recording file (produced by `KinectRecorder.m`) that has been loaded. Note the right-hand is formed into a fist; this is the cue that started the recorder.

5.9.1 Label-Frame Alignment Verification

A vital stage is verification of alignment between frames selected by `SelectFramesFromRecordings.mlapp` and expected labels generated by `prompter.py`. For this study they have been manually validated via the MATLAB function `VerifySelectionAndLabelling.m`, shown in Listing B.6, which displays the images one-by-one in a Figure window, the title of which is the expected label. If the label matches, the user presses [c], if there is a mismatch they press [m] and they may quit by pressing [q]. At the end of the session the function prints a list of mismatch indices and labels to the command window.

While on one hand it is easy to idealise integration this process into the frame selection GUI, upon greater inspection integration does not present a long-term solution: once an automatic temporal segmentation script has been developed, for example by taking the median of the first derivative of optical flow of the sampled frames, the verification would again need to be a post-segmentation process. So despite the current implementation being very basic and not particularly user friendly, it was deemed suitable for purpose.



(a) Cropped ROI of the hand: colour on the left and depth on the right.

(b) Red channel of colour ROI with distant pixels masked out.

Figure 5.24: Automatic segmentation by determining depth at the `Hand` point and excluding more distant pixels. This */bad/* example demonstrates a weakness of this naïve method: failure to exclude parts of the forearm and shoulder. The misalignment of registration can be just discerned in (a) and is all-too-evident in (b).

5.10 Spatial Segmentation

Spatial segmentation refers to the isolation of the pixels corresponding to the target from the rest of the image. In this case, the target is the dominant hand. This was preformed inside a MATLAB script `ProcessRecordings.m`, shown in Listing B.4, that automates all steps from loading the selected frames to producing the numerical features for classification.

There are several 'joints' that could be of use for segmentation, particularly the `Hand`, `Wrist` and `HandTip` for manual segmentation. The entire skeleton is useful for body pose estimation and the 'head joint' provides a starting point for facial segmentation. This study has only implemented unimanual segmentation.

The initial concept was to use depth data to segment colour pixels corresponding to the hand from the background. This is achieved by finding the depth at, for example the `Hand` joint and excluding points with greater depth. Unfortunately, there was a misalignment between depth and colour frames, as shown in Figure 5.24.

The misalignment is likely an artefact of registration and attempts were made at discovering the source and at applying correction algorithms, both without success. At first glance the misalignment in Figure 5.24b appears to be translation to (frame) left as the medial edge of the hand is obscured; however, the separation between the mask and shoulder is much less, suggesting a non-linear relationship.

(a) (b)

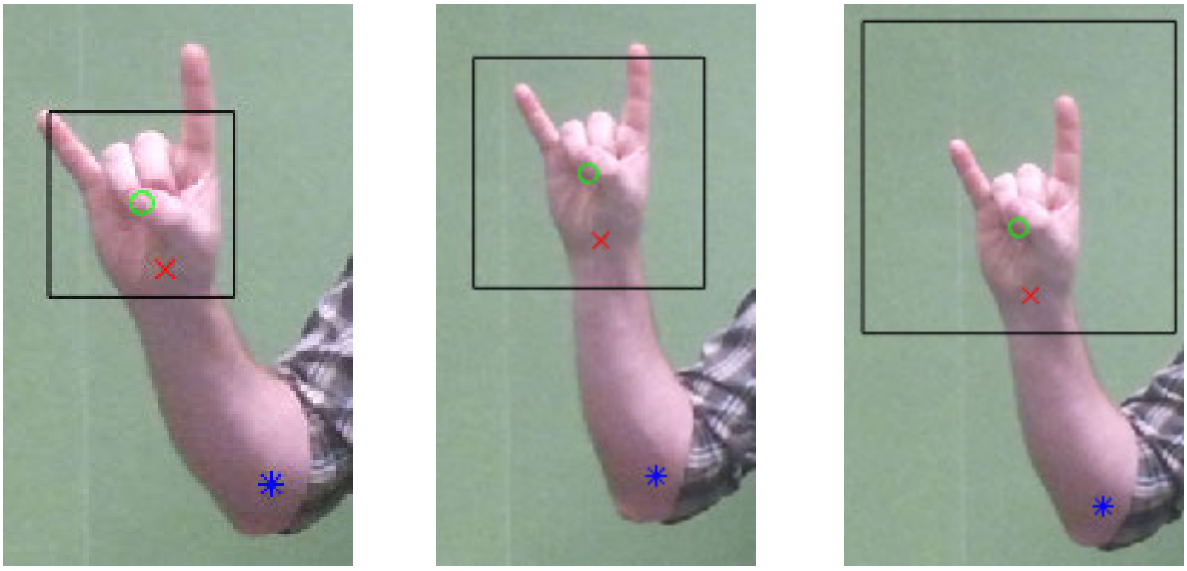
Figure 5.25: Two different instances of */good/* marked with joints: `HandLeft` (◯), `WristLeft` (×) and `ElbowLeft` (∗); the points are both inaccurate and imprecise (as this is a 2D view of 3D space, it is possible there is a 'parallax' viewing error, but it was judged that any such error could not explain the variation observed).

An alternative method was conceived, using the `Hand` point to define the centre of the ROI and use `HandTip` or a ratio of the distance between the `Wrist` and `Elbow` points to define the distal extent. For the proximal extent of the ROI a line through the `Wrist` perpendicular to the `Elbow` was envisaged. For the width, a ratio of lengths or the `Thumb` joint were considered.

A brief exploration of points, shown in Figure 5.25, shows how wildly they vary, precluding this approach.

Finally an automatic segmentation method was arrived at that defined a square ROI around the `HandLeft` joint, as shown in Figure 5.26. Initially attempts were made to dynamically scale the size of the square based on the distance between the wrist and the elbow, but due to variability this was not effective. In the interest of time, a static definition of 200 px was provided, being wide enough to encompass the outstretched hand of the participant standing 2 m from the Microsoft Kinect 2.0. For height, the square was translated up 20 % of the diameter to better include the fingers and reduce inclusion of the forearm that would likely otherwise negatively impact classification accuracy.

Once the ROI has been defined on the colour frame it is 'cropped' (by means of



(a) 0.5 % (b) 0.75 % (c) 200 px

Figure 5.26: Different approaches to define the diameter of square ROI around the `Hand` joint (◯). (a) and (b) are dynamic, defining the diameter as a proportion of the Euclidean distance between the `Wrist` (×) and `Elbow` (✳) in three-dimensional space. (c) applies a fixed diameter (in this case, 200 px) and is translated vertically up 20 % of the diameter.

array slicing) to produce the final RGB ROI. Using the depth map created during the recording, the ROI is mapped onto the depth frame which is likewise cropped.

5.10.1 Binary Image of Hand

The binary image, or 'mask', is the realisation of the depth segmentation discussed in the preceding section; points in the depth ROI more distant than the hand are set to 0 ('off') and points no more distant than the hand are set to 1 ('on'), hence, binary, or 'logical' in MATLAB datatype vernacular. To account for parts of the hand slightly more distant than the distance of the pixel at `HandLeft`, a tolerance is added by means of `iMaxHandDepthMm`.

Although the binary image is not used for segmentation via masking, it may hold information that empowers classification so is output to disk along with the colour and depth ROIs.

5.10.2 Implementation

As mentioned, the automatic implementation of spatial segmentation through to feature extraction is performed by the MATLAB script `ProcessRecordings.m`, shown in Listing B.4.

Automatic Spatial Segmentation

`ProcessRecordings.m` calls the `RoiImagesFromSelectedFrames.m` function, shown in Listing B.5, which *requires a physically connected Microsoft Kinect 2.0*, passing the selected-frame-indexed row vector of `struct` containing Kinect data, map from colour to depth pixels, setting the diameter of the square ROI to 200 px and setting the maximum distance further than the hand joint to include to 200 mm[15]. The function returns three selected-frame-indexed row vectors of `cell`, one containing an `iRoiDiamPx` $\times$ `iRoiDiamPx` $\times\ n_{channels}$ (where $n_{channels} = 3 \therefore$ RGB) array of 8 bit unsigned integers for the colour ROI image, one containing an `iRoiDiamPx` $\times$ `iRoiDiamPx` array of 16 bit unsigned integers for the depth ROI image and one containing an `iRoiDiamPx` $\times$ `iRoiDiamPx` array of `logical` values for the binary hand ROI image.

The selected frames MAT-file is loaded using `ValidatedLoad.m`, shown in Listing B.11, in order to making the loading of files clearer, as it was discovered MATLAB 'wraps'

[15] A distance of 200 mm was selected as being approximately the distance from the fingertips to wrist when forming 'deep' signs such as */animal/*.

loaded files inside a `struct`; this function simply uses a provided expected variable name to unwrap the file and validate that it is the correctly-named variable.

Feature Extraction

Features are extracted using the MATLAB function `ExtractFeaturesFromRoiImages.m`, shown in Listing B.7, which takes as input the serial number of the current sign and the cropped ROIs for colour, depth and the binary image and collates them to a single `struct`.

From the binary image, geometric properties are extracted using region properties from the MATLAB Image Processing Toolbox [93]: area per convex area (the area of the convex hull), area per filled area, perimeter per area, perimeter per convex area and perimeter per filled area. Each of these features are a single normalised scalar.

The MATLAB Computer Vision Toolbox [91] was used to extract more complex features from the depth and colour images. As the colour image comprises three channels, most of the functions cannot be applied directly, with the exception of HoOG, so that is performed separately. HoOG, speeded-up robust features [7] (SURF), maximally stable extremal regions [89] (MSER), KAZE (not actually an abbreviation, but a stylized form of a Japanese word that means 'wind' [4]), binary robust invariant scalable keypoints [84] (BRISK) and oriented FAST and rotated BRIEF [112] (ORB) were applied to each of: each channel of the RGB image, the MATLAB-calculated greyscale version of the RGB image, the depth image and a histogram-equalized version of the depth image. Each of these functions return a variable number of scalars that are reshaped into a column.

The not particularly readable code on lines 46 through 55 are creating the labels, slicing and collating the individual images and creating anonymous functions to perform the extraction. The loops in lines 57 through 63 is where the extraction actually occurs. This separation allows for dynamic naming of the structure's fields and is perhaps a little more readable than the alternative.

`ProcessRecordings.m` collates the individual feature `struct`s into a row vector and saves them to the source directory as the original filename suffixed by `"_extractedFeatures.mat"`.

5.11 Feature Selection

To reduce the enormous number of features produced by `ProcessRecordings.m` (> 600 000) the features are pruned by `FeatureStructToArrayWithPca.m`, shown in Listing B.8. PCA is used to reduce the number of features down to the number of principal components specified by `nPcaComponents`, ranked by explanation of the data.

First, any empty fields are removed, along with fields that are thought to correlate strongly with one another (that is, ones that are linear combinations).

For each of the remaining features, the feature values for each lexeme are trimmed to the height of the shortest column, then vertically concatenated into one column; the columns for all the lexemes are then horizontally concatenated, forming a two-dimensional array.

Once transposed, this array forms a "design matrix", $n \times p$ where n (rows) is the number of 'observations', or *instances* (in this case, lexemes) and p (columns) is the number of 'variables', or *features*. The design matrix is saved to disk as `anIxFCollated.mat` along with the labels of the surviving features.

5.11.1 Multi-layer PCA Selection

To clarify the multi-layer PCA selection strategy: the first layer is performed inside `FeatureStructToArrayWithPca.m`, where PCA can be used to select `FeatureStructToArrayWithPca.nPcaComponents` principal components at the function value column stage; that is, selecting the number of components per feature (where, for example, the BRISK featue set may have ≈ 90 'subfeature' columns). The second layer occurs within `ClassifyMl.m` where PCA is used to select `ClassifyMl.nComponents` principal components; classification is then performed on these components, rather than individual features. This layered approach arose organically as a way of reducing linear combinations and may be redundant; a single, more considered approach could likely provide the same result and be easier to interoperate but unfortunately has not been explored due to time constraints.

5.12 Classification

Classification is performed in MATLAB using `newff` from the R2010a version of the Neural Network Toolbox [90] using the script `ClassifyMl.m` shown in Listing B.9. The

design matrix `anIxFCollated.mat` is loaded, constant rows are removed and the mean of each row is mapped to 0 while the standard deviation of each row is mapped to 1. PCA is performed to select the `nComponents` principal components ranked by explanation of variance of the data.

10-fold cross-validation is used to perform supervised learning. For each fold, a neural network with a single hidden layer of 10 neurons is created. Early stopping is achieved by validation and by limiting the maximum number of training iterations. The output transfer function is log-sigmoid, mapping values to between 0 and 1. The training algorithm is Levenberg-Marquardt back-propagation.

Finally, Rand accuracy and Bookmaker Informedness are calculated and printed to the command window.

5.12.1 Measuring Performance

The traditional measure of machine learning performance, accuracy, is generally provided by 'precision', a measure that only accounts for 'true positive accuracy' `tpa` = ratio of times a class was correctly predicted (or 'true positive' `tp`) to the total number of times the class was predicted (or 'predicted positive' `pp`, which in turn is the sum of `tp` and 'false positives' `fp`). The accuracy of results are presented using two measures that take 'true negatives' `tn` into account: Rand Index, a weighted mean of correct predictions to the total number of cases and Bookmaker Informedness, which specifies probability versus chance; both measures are calculated from values in the Contingency matrix (Table 5.5) as follows [107]:

$$\begin{aligned}
\text{Rand Index} &= \texttt{rp} \times \texttt{tpr} + \texttt{rn} \times \texttt{tnr} \\
&= \texttt{rp} \times \frac{\texttt{pp}}{\texttt{rp}} + \texttt{rp} \times \frac{\texttt{pn}}{\texttt{rn}} \\
\text{Informedness} &= \text{Recall} + \text{Inverse Recall} \quad 1 \\
&= \texttt{tpr} + \texttt{tnr} \quad 1 \\
&= \frac{\texttt{tp}}{\texttt{pp}} + \frac{\texttt{tn}}{\texttt{pn}} \quad 1
\end{aligned}$$

Table 5.5: The binary Contingency table [107]. Inside the matrix, the prefix `t` denotes 'true' while the prefix `f` denotes 'false'; the suffix `p` denotes 'positive' and the suffix `n` denotes 'negative'. The rows and columns are summated: labels with prefix `p` denote 'predicted' and prefix `r` denotes 'real'; the suffixes are as before ('positive' and 'negative'). For example, `tp` is 'true positive', `pp` is 'predicted positive' and `rn` is 'real negative'.

`tp`	`fp`	`pp`
`fn`	`tn`	`pn`
`rp`	`rn`	`1`

Chapter 6

Validation

6.1 Introduction

The purpose of the validation study is to test the developed method for static handshape recognition. A set of signs, or lexicon, was selected that provided achievable challenge, as per the literature [47], [101], [126].

6.2 Method

The validation study follows that detailed in Chapter 5; only a summary of method and the rational for decisions are provided here.

6.2.1 Study Parameters

Lexicon

Several possible sets of signs, or lexicons, were considered for the validation study. The Auslan alphabet is a convenient self-contained set of which only two letters include motion H and J but being predominantly bimanual means it is more difficult to automatically segment images to a regular region of interest, increasing classification challenge. One could select just the static unimanual signs, or reduce that further to similar signs, such as the pairs L & R, N & V and D & P, which would be challenging for any system to distinguish between. Auslan numbers 0 to 9 are unimanual and almost static they are performed with a slight forward jerk, but this could be omitted but the silhouettes are quite similar. For simplicity, a lexicon of five basic handshapes (thus, phonemes), all with distinct silhouettes, was selected for the validation: */five/*, */closed/*,

(a) */five/* (b) */closed/* (c) */animal/* (d) */good/* (e) */bad/*

Figure 6.1: Images of the lexicon of five basic handshape phonemes used for the validation study of the Sign-to-Text system.

/animal/, */good/* and */bad/*, as shown in Figure 6.1.

Performance Guidelines

To improve consistency in the validation it was decided to perform the handshapes laterally and anteriorly at approximately chest height that is, by a natural rotation of the elbow such that the forearm moves 'upwards' and 'outwards' while the elbow hung naturally from the shoulder without consciously restricting elbow translation. The hand was to return to a 'hanging' neutral position between performances. The participant was to stand in the same spot and reduce other movements for the duration.

Replicates and Timing

For this first validation study the number of replicates per lexeme `nReplicates` was set to 10, being adequate for basic classification without being too time consuming during the initial vetting of the system.

The time between lexemes, `nSecondsPerTrial` was set to 5 s and the dwell time per lexeme, `nSecondsPerCountdown` in `propmter.py` was set to 2 s, knowing that on the hardware employed 2 s resolved to an empirically-measured actual dwell time of (2.92 ± 0.04) s.

The total time t_{total}:

$$\begin{aligned} t_{total} &= \texttt{nTrials} \times \texttt{nReplicates} \times \texttt{nSecondsPerTrial} \\ &= 5 \times 10 \times 5\,\text{s} \\ &= 250\,\text{s} \end{aligned}$$

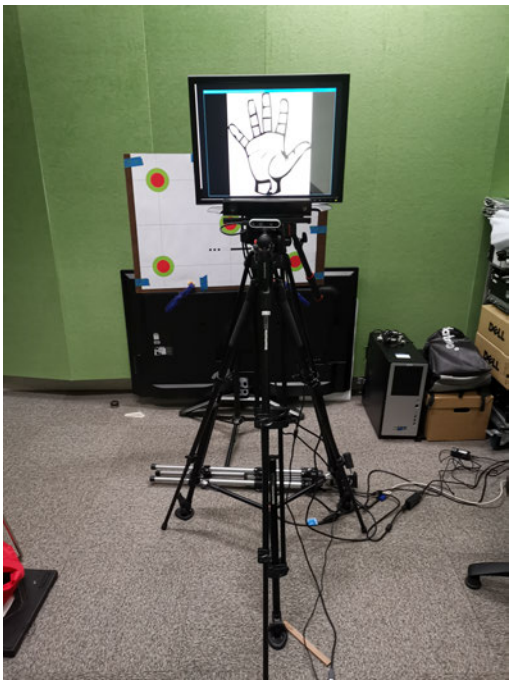


Figure 6.2: Recording setup with Microsoft Kinect 2.0 and a computer monitor set up on tripods. A Intel RealSense D435i is also mounted on the tripod in this image: this was purely for interest and served no functional purpose.

6.2.2 Setup

Recording was performed in a room with no natural light to reduce illumination variability and linear fluorescent artificial lighting that had been confirmed not to interfere with the depth measurement. The setup is mostly[1] shown in Figures 6.2 and 6.3.

The Microsoft Kinect 2.0 was set up on a floor-standing tripod with three-axis adjustability and levelled horizontally and vertically. As the wall was not flat, the recording setup was aligned by means of carpet squares. The position where the participant would stand was set at an intersection between carpet tile seams approximately 0.5 m from the wall. The camera was positioned atop the seam that ran perpendicular to the wall, such that it was 2.00 m from the signer position. The Calvert Calibration Target was set up carefully aligned atop the intersection of the carpet tile seams; using real-time visual feedback via the RealSense Viewer the camera & tripod were carefully positioned such that the camera was inline with and perpendicular to the central vertical line of the target. The camera was levelled, alignment reviewed and the target removed.

[1] The setup as shown was staged hastily for the capture of these photographs and is not actually aligned as specified.

Figure 6.3: Example of participant signing in front of camera.

A computer monitor was clamped to a second floor-standing tripod, positioned just behind the first tripod and adjusted such that the display was just higher than the top of the Microsoft Kinect 2.0 and hence, not obscured.

The FootSwitch3-F3.4 was placed on the floor just in front (towards the camera) of the target carpet seam. The left key (1) was set to Left Mouse, the central key (2) to Escape and the right key (3) to Enter.

The Microsoft Kinect 2.0, mounted monitor and foot-switch were connected to a (non-University) desktop computer on an adjacent desk; the Microsoft Kinect 2.0 was also connected to power. As the desktop computer had two monitors already and had inadequate graphical ability to run all three monitors, one of the regular monitors was disabled and the mounted monitor enabled in software.

6.2.3 Session Flow

Due to the disjointed nature of the software used to prompt and record, care was taken in sequencing events. First, the Python prompter script `prompter.py` (Listing B.1) was run. During initialisation the display window was adjusted to appear on the mounted monitor and Enter pressed, taking the script into the placeholder screen. Second, the

MATLAB recorder script `KinectRecorder.m` (Listing B.2) was initialised and the figure windows moved to be visible on the desktop's monitor. Third, the Prompter window was selected, completing initialisation.

When the participant was in position and confirmed to be fully within the RGB frame with the skeleton overlay functioning correctly, the recording session was begun by forming a fist with the non-dominant (right) hand and the prompting session by pressing Enter key 3 on the foot-switch.

At the end of the prompting session the script closed silently; the recording script took a few seconds longer than expected to complete, then left a success message in the Command Window.

The 'recording' output file from `KinectRecorder.m` was saved to `sDirOut` (default `"D:/KinectRecordings/"`), as a timestamped `mat` file, in this case: `2019Sep17Tue20h28_Recording.mat`.

The Recorder script also created a map of pixel coordinates from the depth frame to the colour frame, saved to the same directory as `2019Sep17Tue20h28_aiD2C.mat` and a reverse map (colour to depth) as `2019Sep17Tue20h28_aiD2C.mat`.

6.2.4 Segmentation

Temporal segmentation, the selection of one frame per lexeme, was performed manually in the MATLAB AppDesigner GUI `SelectFramesFromRecordings.mlapp` (Listing B.3). The MATLAB recording was loaded from disk, eventually[2] resulting in the screen shown in Figure 5.23. While making frame selections, care was taken to select frames without perceptible motion artefact, as shown in Figure 6.4. On closing the GUI the selections file was saved to the source directory as `2019Sep17Tue20h28_Recording.mat_selectedFrames.mat`.

Frame-to-label verification was performed manually in `VerifySelectionAndLabelling.m` (Listing B.6), visually comparing the selected frames to expected labels as extracted from the `prompter.py` log file `Prompter_2019Sep17Tue20h22.log`, the first 15 lines of which are shown in Listing 6.1; all frames were correctly labelled.

[2]Loading a large recording file (> 2 GB) does take considerable time during which the GUI does not appear to be doing anything…

```
1  INFO:root:This is:
   ↪ d:\Dropbox\Uni\2019S1\ENGR9700\signtotext\Python\prompter.py
2  INFO:root:It is: 2019-09-17 20:22:08.429736
3  INFO:root:Timing: trials: 5 s, countdown: 2 s
4  INFO:root:Lexicon: ['animal', 'bad', 'closed', 'five', 'good'] (5 items.)
5  INFO:root:Repeats per sign: 10; 50 total trials, taking 250 s.
6  INFO:root:Trial sequence: ['bad', 'closed', 'good', 'animal', 'bad', 'five',
   ↪ 'five', 'bad', 'animal', 'five', 'animal', 'good', 'good', 'closed',
   ↪ 'five', 'five', 'animal', 'good', 'closed', 'good', 'good', 'animal',
   ↪ 'closed', 'bad', 'five', 'animal', 'bad', 'animal', 'bad', 'animal',
   ↪ 'good', 'good', 'bad', 'bad', 'closed', 'five', 'bad', 'animal', 'closed',
   ↪ 'bad', 'closed', 'animal', 'five', 'closed', 'five', 'good', 'good',
   ↪ 'five', 'closed', 'closed']
7  INFO:root:Starting trial 1/50: 'bad' at 20:23:25
8  INFO:root:Countdown completed in 0:00:02.831060, next trial starts in
   ↪ 0:00:02.168940
9  INFO:root:Starting trial 2/50: 'closed' at 20:23:30
10 INFO:root:Countdown completed in 0:00:02.791286, next trial starts in
   ↪ 0:00:02.208714
11 INFO:root:Starting trial 3/50: 'good' at 20:23:34
12 INFO:root:Countdown completed in 0:00:02.771153, next trial starts in
   ↪ 0:00:02.228847
13 INFO:root:Starting trial 4/50: 'animal' at 20:23:39
14 INFO:root:Countdown completed in 0:00:02.759979, next trial starts in
   ↪ 0:00:02.240021
15 INFO:root:Starting trial 5/50: 'bad' at 20:23:44
16 ...
```

Listing 6.1: First 15 lines of `Prompter_2019Sep17Tue20h22.log`, the logfile produced by `prompter.py` for the validation study.

(a) Motion artefact

(b) Steady

Figure 6.4: Example of motion artefact in the recording: in (a) movement of the arm was fast enough that it changed location while the shutter was open, so the image captured is an averaging of that motion over that time. In (b) the arm is fairly steady: any motion artefact is too small to perceive.

Region-of-Interest Images

Region of interest images were produced by `RoiImagesFromSelectedFrames.m`, shown in Listing B.5, passing arguments `iRoiDiamPx = 200` and `iMaxHandDepthMm = 120`. The position of the ROI was established by the coordinates of the `HandLeft` joint in the colour frame, which was then cropped to produce the 200×200 px RGB ROI image. The depth ROI image was then produced by mapping the colour ROI into the depth frame. Finally, the depth value at `HandLeft` plus `iMaxHandDepthMm` was used as a threshold to binarize the depth ROI, forming the binary ROI image.

6.2.5 Feature Extraction and Selection

`ProcessRecordings.m` then called `ExtractFeaturesFromRoiImages.m`, shown in Listing B.7, to perform feature selection from the ROI images. The feature value `struct`s for each lexeme were horizontally concatenated. The row of `struct` was saved to disk, completing the `ProcessRecordings.m` script.

Feature selection was performed by `SelectFeaturesFromStructWithPca.m` twice, once with `nComponents = 2` and once at three principal components. The design matrix `anIxFCollated.mat` was saved to disk.

6.2.6 Classification

Classification was performed by a neural network in MATLAB using the script `ClassifyMl.m`, shown in Listing B.9. `anIxFCollated.mat` was loaded from disk, constant rows were removed and the mean value of each row mapped to 0 and deviation of each row mapped to 1.

The neural network was supervised, using 10-fold cross-validation. A single hidden layer with 10 neurons was created for each fold. Early stopping was triggered using validation by 20 % of within-fold data or reaching a limit of 500 epochs. A log-sigmoid output transfer function mapped the results to between 0 and 1, matching the mapping of the design matrix.

Classification was repeated a number of times, once using the 2 PCA set and six times using the 3 PCA set. For the last five, an additional layer of selection by PCA was performed within `ClassifyMl.m`, choosing the 15, 5, 4, 3 and then 2 most explanatory principal components. The accuracy as reported by `ClassifyMl.m` was recorded.

6.3 Results

The results of classification are presented in Table 6.1, the composition of the three principal components used in Set 6. are shown in Figure 6.5.

6.4 Discussion

In terms of feature selection, the plot of contribution to classified components (Figure 6.5) shows that Component 1 largely dominated by geometric properties of the binary image, with the 27 % of the variance explained by: the ratio of perimeter to area of the binary image, the ratio of perimeter to filled area of the binary image and the area per convex hull area of the binary image. 21 % of the variance was explained by HoOG features: the first principal component for each of the colour ROI, depth ROI and histogram-equalized-depth ROI (these last two then are likely linear combinations).

Component 2 was dominated by the second principal component of the HoOG for the depth ROI, histogram-equalized-depth ROI and colour ROI, explaining 33 % of the variance. The perimeter per convex area of the binary image accounted for a further 10 % of the variance.

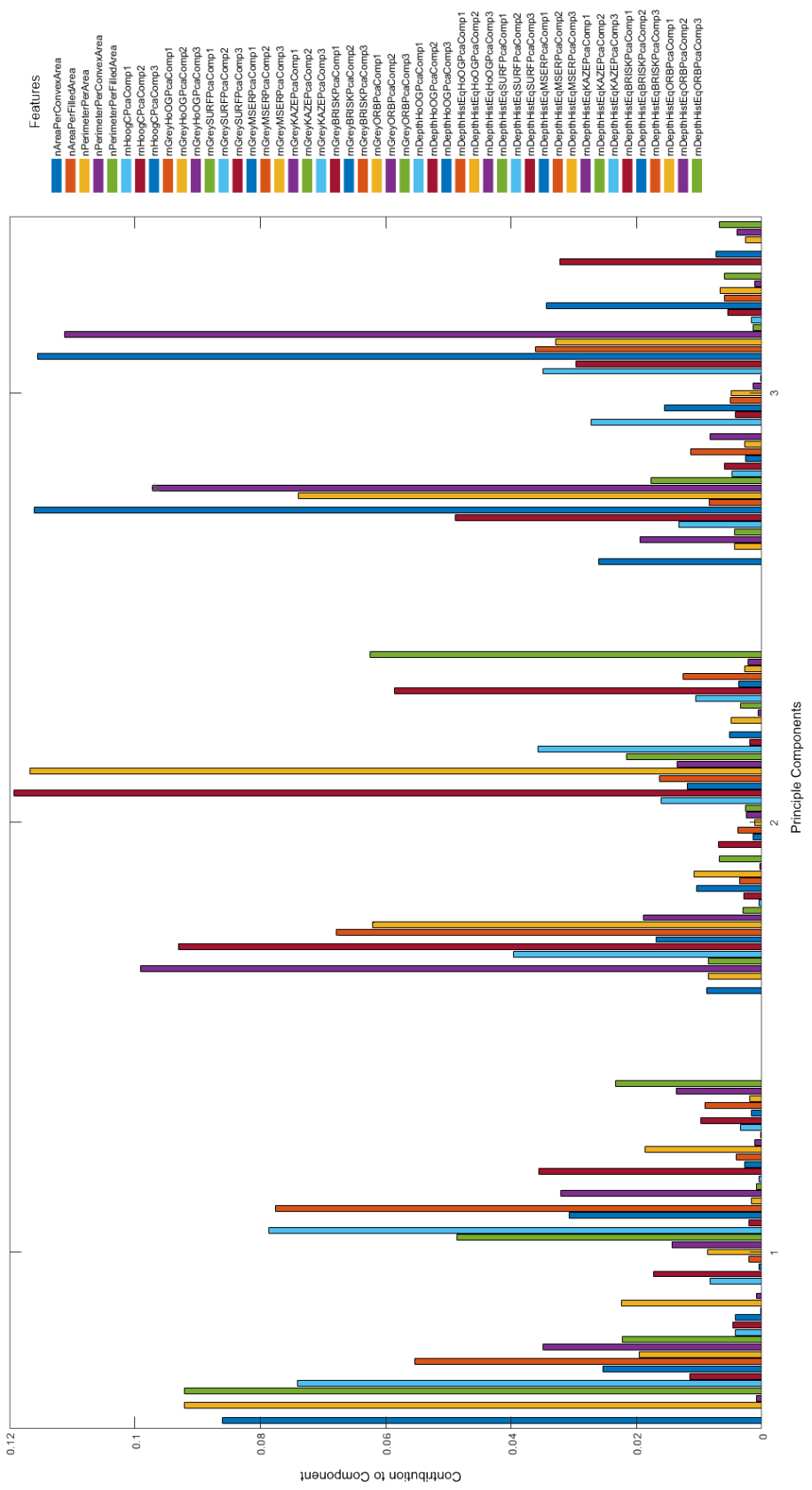


Figure 6.5: Plot of feature's contributions to the three principal components that formed the set with greatest accuracy (6.).

Table 6.1: Results of classification for the verification study of 5 handshapes and 10 replicates (50 observations/instances). Feature extraction was the same for all result sets. The first layer of feature selection (L1) uses PCA to select which subfeatures of a feature set are include by ranking contribution to explanation of variance. Layer two (L2) uses PCA again produce principal components containing these features.

Set	# Components		Rand Index		Bookmaker Informedness	
	L1	L2	Mean	SD	Mean	SD
1.	2	49	0.43	0.06	0.33	0.07
2.	3	47	0.55	0.06	0.48	0.09
3.	3	15	0.76	0.05	0.71	0.07
4.	3	5	0.79	0.04	0.78	0.05
5.	3	4	0.86	0.02	0.86	0.03
6.	3	3	0.86	0.03	0.87	0.03
7.	3	2	0.78	0.03	0.77	0.04

The third principal component of HoOG for each of the: colour, depth, histogram-equalized depth and grayscale ROIs explained 43 % of the variance for Component 3. The second principal component of HoOG for both grey and colour accounted for a further 13 % of the variance.

The geometric properties of the binary image and HoOG explained most of the variance across the components; the other Computer Vision Toolbox features contributed little, with the exception of the third component of ORB and the second component of BRISK, both of the histogram-equalised depth ROI, which both accounted for 6 % of the variance in Component 2. This result is inline with much of the literature, where geometric properties can clearly be seen to be highly informative for these diverse silhouettes and HoOG is prevalent.

A mean Bookmaker Informedness of 87 % gives high accuracy compared to chance and outperforms a similar study [47] but not as well as [126] who achieved 99 % 'accuracy' (definition unknown) using super-pixel earth mover's distance on distinct silhouettes or [31] who achieved 93 % on a set of 1680 samples.

6.5 Conclusion

This validation study aimed to assess the groundwork implementation of the proposed framework via static handshape recognition. The development of the capture system and supporting software was non-trivial, but has been shown to perform at a level comparable to the literature for a small basic lexicon and a single signer.

Chapter 7

Conclusion

This project investigated the enduring challenge of automatically recognising sign language, interpreting it and translating it to a second language. Although attempts have been made to address this challenge for over 30 years, SL remains a challenge for both linguistic and technical reasons.

The linguistics of signed languages were explored thoroughly, providing insight into the elements that define sign language and synthesising the various components and scholarly views to provide a new taxonomy of linguistic structure. Untangling the phonemic categories and setting them all at the same basal level, rather than having reducible irreducible phonemes provides a more complete and rigorous representation.

Challenges intrinsic to signed languages were also researched, finding valuable information in the linguistic literature but some confused interpretations within the recognition literature. The existing challenges are laid out in a single, detailed list with examples for the more commonly misconstrued labels. Through an understanding of both sides of the SLR coin, three additional challenges are proposed, including the requirement for remembering the assignment of entities to particular locations within the signing space, the entanglement of spatial referents with movement epenthesis and the well-known but not previously listed challenge of identifying temporal edges between signs.

A framework that incorporates all the elements of signed language is presented. The 'acoustic model, linguistic model' approach from speech recognition is adapted to form linguistic hierarchy of 'visual model' then 'linguistic model'. A modular approach supports and hopefully encourages multiple input modalities. The addition of temporo-spatial memory block is essential to support spatial deictic signs and contextual deictic modifiers, but the actual implementation and requirements have not been considered.

The recognition of sign languages is an evolving field where new technologies such as depth cameras and new techniques such as deep learning are slowly helping researchers

overcome long-standing barriers. The use of multi-modal sign observation helps reduce occlusion and making use of existing large labelled datasets such as televised weather forecasts provide the means to drastically improve inter-signer accuracy.

A rudimentary implementation of the framework was undertaken as a means of vetting the systems and enhancing concepts, providing an excellent exercise in technical hurdle-jumping. Despite issues with both hardware and software, a system was eventually devised to recognise static handshapes. The implementation was validated by means of a trial of 5 handshapes, achieving a Bookmaker Informedness accuracy of 87 %.

7.1 Future Work

Future work should extend this implementation to include greater phonemic recognition; first adding to handshape recognition then adding additional categories using established pattern recognition techniques and possibly instrumentation. Once several classifiers are producing phonemes, the 'puzzle pieces' of temporo-spatial memory and contextual deictic modifiers can be explored. Development of a language model will require a statistical approach from a linguistic perspective, before glossing and language translation.

Appendices

Appendix A

Linguistic Conventions

The stylistic conventions for annotating the linguistics of signed language used in this thesis are taken from the works of Johnston, albeit some of them with definition interpreted through context as their explicit definition could not be discovered.

PRO_{3a} STAND_{3a} WHEN $_{3a}\text{ASK}_{3b}$ WHY PRO_{3b} ANGRY

He_a was standing just there when he_a asked him_b why he_b was angry.

YESTERDAY, $_1\text{MEET}_{3a}$ POSS_2 BROTHER.

I met your brother yesterday.

PRO_{3a} = TELL_1 PRO_2 ENGAGED

They told me you were engaged.

Appendix B

Recognition Code

B.1 Teleprompter script

```
import cv2
import numpy as np
import logging
import DUtils
from random import shuffle
from screeninfo import get_monitors
from datetime import datetime, timedelta


class Lexicon:
  def __init__( self, lsSigns, pPath, sPrefix, sSuffix, lsImages=None):
    self.lsSigns = lsSigns
    self.pPath = pPath
    self.sPrefix = sPrefix
    self.sSuffix = sSuffix
    self.lsImages = lsImages if type( lsImages) is list and len( lsImages) > 0
    ↪  else [str( pPath / ( sPrefix + sSign + sSuffix)) for sSign in lsSigns]
    self.dSignImages = dict( zip( lsSigns, self.lsImages))

  def __repr__( self):
    return self.lsSigns

def centre_window_on_monitor( sWinName = None):
  # Use global version if no argument provided
  # (sometimes overloading would really be sensible...)
  if sWinName == None:
    sWinName = sWindowName
  iWinX, iWinY, iWinW, iWinH = cv2.getWindowImageRect( sWinName)
  mMonitor = get_monitors()[ iMonitor]
  iDelX = mMonitor.x + int( ( mMonitor.width - iWinW) * 0.5)
  iDelY = mMonitor.y + int( ( mMonitor.height - iWinH) * 0.5)
  cv2.moveWindow( sWinName, iDelX, iDelY)


def setup_checker( lLexicon):
```

```
35
36    global iMonitor
37
38    nSetupTimeoutSeconds = 10
39    nMilliSecondsPerSecond = 1000
40    nSetupTimeoutMilliSconds = nSetupTimeoutSeconds * nMilliSecondsPerSecond
41
42    print( f'Attempting to load image with path: { lLexicon.lsImages[0]}')
43
44    nPressedKey = None
45    lmMonitors = get_monitors()
46    sMonitors = DUtils.list_to_pretty_string( list( range( 1, len( lmMonitors) +
      ↪  1)))
47
48    print( 'You should see a window showing the image.')
49    print( "If the image does not work, press 'Esc'")
50    print( f'Regardless of where the image is shown, press the key of the number
      ↪  of the monitor it should be on: {sMonitors}')
51    print( "If the image works and is on the correct screen, press 'Enter'")
52
53    while not DUtils.keypress_matches( nPressedKey, 'enter'):
54      # print( f'Current monitor: {lmMonitors[0]}')
55      nPressedKey = show_image( cv2.imread( lLexicon.lsImages[0]),
        ↪  nSetupTimeoutMilliSconds)
56
57      if len( lmMonitors) > 1:
58        bKeyMatched = False
59        for i in range( 1, len(lmMonitors) + 1):
60          if DUtils.keypress_matches( nPressedKey, str( i)):
61            logging.debug( f'i: { i}, iM: { iMonitor}, iMNew: { i - 1}')
62            bKeyMatched = True
63            iMonitor = i - 1 # convert to list index
64            centre_window_on_monitor()
65            break
66        if bKeyMatched:
67          continue
68
```

```
69      if DUtils.keypress_matches( nPressedKey, 'enter'):
70        cv2.destroyAllWindows()
71        return True
72      if DUtils.keypress_matches( nPressedKey, 'esc'):
73        return False
74      else:
75        raise Exception( 'Invalid keypress or timeout')
76
77
78  def combine_images( anSector, anImage):
79    nAlpha = 0.4
80    nBeta = 1
81    nGamma = 0
82    return cv2.addWeighted( anSector, nAlpha, anImage, nBeta, nGamma)
83
84
85  def show_image( anImage, nDelayMilliSeconds):
86    cv2.imshow( sWindowName, anImage)
87    return cv2.waitKey( nDelayMilliSeconds)
88
89
90  def resize_image_maintain_aspect( anImage, tnResolution):
91
92    # Resize image, maintaining aspect ratio
93
94    nTgtW, nTgtH = tnResolution
95    nOldH, nOldW, _ = anImage.shape
96
97    nScaleW = nTgtW / nOldW
98    nScaleH = nTgtH / nOldH
99
100   nScale = min( nScaleW, nScaleH)
101
102   nNewW = int( nOldW * nScale)
103   nNewH = int( nOldH * nScale)
104
105   anImage = cv2.resize( anImage, ( nNewW, nNewH))
```

```
106
107   # Pad resized image to window size
108
109   nPadW = nTgtW - nNewW
110   nPadH = nTgtH - nNewH
111
112   nPadT = nPadB = nPadL = nPadR = 0
113
114   if nPadW:
115     nPadL = nPadW // 2
116     nPadR = nPadW - nPadL
117   if nPadH:
118     nPadT = nPadH // 2
119     nPadB = nPadH - nPadT
120
121   return cv2.copyMakeBorder( anImage, nPadT, nPadB, nPadL, nPadR,
      ↪  cv2.BORDER_CONSTANT, value=[ 0, 0, 0])
122
123
124 def clock_sector( nEndAngle, tnResolution):
125   nWidth, nHeight = tnResolution
126
127   anSector = np.zeros( ( nHeight, nWidth, 3), np.uint8) # reset
128
129   nAxisScale = 0.71 # slightly larger than 1/sqrt(2) so radius greater than
      ↪  diagonal
130
131   tnCenter = ( int( nWidth * 0.5), int( nHeight * 0.5))
132   tnAxes = ( int( nHeight * nAxisScale), int( nWidth * nAxisScale))
133   nRotation = -90 # rotation of major axis CW from x+ in degrees
134   nStartAngle = 0 # rotation of start CW from major in degrees
135   # nEndAngle = 45 # rotation of end CW from major in degrees
136   tnColour = (128, 128, 128) # BGR in [0,255]
137   nThickness = -1 # guessing -1 means 'solid'?
138
139   cv2.ellipse( anSector, tnCenter, tnAxes, nRotation, nStartAngle, nEndAngle,
      ↪  tnColour, nThickness)
```

```
140
141   return anSector
142
143 def show_placeholder_until_ready():
144   sWinName = "When you're quite ready..."
145   anImg = np.zeros((nWindowHeight,nWindowWidth,3), np.uint8)
146   sText = "When you are ready, press 'Enter'"
147   fFontFace = cv2.FONT_HERSHEY_SIMPLEX
148   nFontScale = 2
149   tnFontColor = ( 0, 255, 0)
150   nLineThickness = 1
151   nTextWidth, nTextHeight = cv2.getTextSize( sText, fFontFace, nFontScale,
      ↪  nLineThickness)[0]
152   tnOrigin = (
153     int( ( nWindowWidth - nTextWidth) * 0.5),
154     int( ( nWindowHeight - nTextHeight) * 0.5)) # bottom, left corner of text
155
156   cv2.putText( anImg, sText, tnOrigin, fFontFace, nFontScale, tnFontColor,
      ↪  nLineThickness)
157
158   cv2.imshow( sWinName, anImg)
159   centre_window_on_monitor( sWinName)
160   nPressedKey = cv2.waitKey(0)
161   logging.debug( f'Placehold exit key: { DUtils.keypress_string(
      ↪  nPressedKey)}')
162   cv2.destroyAllWindows()
163   if DUtils.keypress_matches( nPressedKey, 'enter'):
164     return True
165   else:
166     raise Exception( 'Invalid keypress during placeholder')
167
168
169 def initialise_logging( sLogFile):
170   logging.basicConfig( filename =s LogFile, level = logging.DEBUG)
171   logging.info( f'This is: { __file__}')
172   logging.info( f'It is: { datetime.now()}')
173   print( f'Logging to { sLogFile}.')
```

```
174
175 # *** BEGIN ACTUAL SCRIPT ***
176
177 # Begin logging
178 sRunId = datetime.now().strftime( '%Y%b%d%a%Hh%M')
179 initialise_logging( 'PrompterLogs/Prompter_' + sRunId + '.log')
180
181 # Global variables
182 sWindowName = "Press 'Esc' to exit"
183 iMonitor = 0
184
185 # Timing
186 # * NOTE: countdown time is considerably more than specified (likely due to
    ↪ mechanism of CV2 drawing); e.g. a specified 2 second countdown might take
    ↪ around 6 seconds in reality. Accordingly, make sure trial period time is
    ↪ much larger than countdown time.
187 # Examples:
188 # @ nStepsPerCircle = 60:
189 #   Trials: 5 s, Countdown: 3 s, Actual 3.75 ± 0.15 s - i3-3220
190 #   Trials: 5 s, Countdown: 3 s, Actual 4.1 s         - i3-3220, Kv2 recording
191 #   Trials: 5 s, Countdown: 2 s, Actual 2.92 ± 0.04 s - i3-3220, Kv2 recording
192 nSecondsPerTrial = 3 #! USER CONFIG
193 nSecondsPerCountdown = 1 #! USER CONFIG
194 nMilliSecondsPerSecond = 1000
195 nMilliSecondsPerCountdown = nSecondsPerCountdown * nMilliSecondsPerSecond
196 nDegreesPerCircle = 360
197 nStepsPerCircle = 60 #! USER CONFIG
198 nDegreesPerStep = nDegreesPerCircle / nStepsPerCircle
199 nMilliSecondsPerStep = int( nMilliSecondsPerCountdown / nStepsPerCircle)
200 logging.info( f'Timing: trials: { nSecondsPerTrial} s, countdown: {
    ↪ nSecondsPerCountdown} s')
201
202 # Signs & Trials
203 pRoot = DUtils.get_abs_path_to_named_parent( 'signtotext')
204 pRel = pRoot / 'Images' / 'Auslan'
205
206 lAlphabetDistinct = Lexicon( [ 'a', 'b', 's', 'w', 'z'], pRel, '', '.jpg')
```

```
207 lAlphabetSimilarPairs = Lexicon( [ 'l', 'r', 'n', 'v', 'd', 'p'], pRel, '',
    ↪ '.jpg')
208 lAlphabetStatic = Lexicon( [ 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'i', 'k', 'l',
    ↪ 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z'],
    ↪ pRel, '', '.jpg') # omits dynamic signs: H, J
209 lShapesSimple = Lexicon( ['animal', 'bad', 'closed', 'five', 'good'], pRel,
    ↪ '', '.png')
210
211 lLexicon = lShapesSimple #! USER CONFIG
212 nReplicates = 2 #! USER CONFIG
213 lsTrials = lLexicon.lsSigns * nReplicates
214 shuffle( lsTrials) # works on argument without return
215 lsTrialPaths = [ lLexicon.dSignImages[ sSign] for sSign in lsTrials]
216 logging.info( f'Lexicon: { lLexicon.lsSigns} ({ len( lLexicon.lsSigns)}
    ↪ items.)')
217 logging.info( f'Repeats per sign: { nReplicates}; { len( lsTrials)} total
    ↪ trials, taking {len(lsTrials)*nSecondsPerTrial} s.')
218 logging.info( f'Trial sequence: { lsTrials}')
219
220
221 # Check images are loading and we've got correct monitor
222 iMonitor = 0
223 if not setup_checker( lLexicon):
224   raise Exception( 'Setup not correct')
225
226
227 # Define size of window on target monitor
228 nWindowScale = 0.9 #! USER CONFIG
229 nWindowWidth = int( get_monitors()[ iMonitor].width * nWindowScale)
230 nWindowHeight = int( get_monitors()[ iMonitor].height * nWindowScale)
231 anBlackRect = np.zeros( ( nWindowHeight, nWindowWidth, 3), np.uint8)
232
233 # Wait until ready to begin...
234 show_placeholder_until_ready()
235
236
237 # Start the trials!
```

```
238 dtSessionStart = datetime.now()
239 bEarlyExitByKeypress = False
240 for iTrial, sSign in enumerate( lsTrials):
241   dtTrialStart = datetime.now()
242   logging.info( f"Starting trial { iTrial + 1}/{ len( lsTrials)}: '{ sSign}'
      ↪  at { dtTrialStart.strftime('%H:%M:%S')}")
243
244   anImage = cv2.imread( lLexicon.dSignImages[ sSign])
245   anImage = resize_image_maintain_aspect( anImage, ( nWindowWidth,
      ↪  nWindowHeight))
246
247   # Establish window on correct monitor once per image
248   tdUntilEstimatedCompletion = timedelta( seconds=( nSecondsPerTrial * ( len(
      ↪  lsTrials) - iTrial)))
249   dtEstimatedCompletion = dtTrialStart + tdUntilEstimatedCompletion
250   sWindowName = f"Trial { iTrial + 1}/{ len( lsTrials)} ({ int( ( iTrial +1 )
      ↪  / len( lsTrials) * 100)} %), {tdUntilEstimatedCompletion} remaining:
      ↪  estimated completion at {dtEstimatedCompletion.strftime('%H:%M:%S')}
      ↪  (currently {datetime.now().strftime('%H:%M:%S')}). Press 'Esc' to break
      ↪  prematurely."
251   cv2.destroyAllWindows() # in case they linger
252   cv2.imshow( sWindowName, anImage)
253   centre_window_on_monitor()
254
255   # Countdown overlay for this trial
256   for iStep in range( nStepsPerCircle + 1):
257     anSector = clock_sector( iStep * nDegreesPerStep, ( nWindowWidth,
        ↪  nWindowHeight))
258     nPressedKey = show_image( combine_images( anSector, anImage),
        ↪  nMilliSecondsPerStep)
259     if DUtils.keypress_matches( nPressedKey, 'escape'):
260       bEarlyExitByKeypress = True
261       break
262   if bEarlyExitByKeypress:
263     break
264
265   dtCountdownEnd =  datetime.now()
```

```
266   tdCountdownTook = dtCountdownEnd - dtTrialStart
267   tdNextTrialStart = dtTrialStart + timedelta( seconds = nSecondsPerTrial) -
      ↪ dtCountdownEnd
268   logging.info(f'Countdown completed in { tdCountdownTook}, next trial starts
      ↪ in { NextTrialStart}')
269   # logging.debug(f'Start: {fTrialStartTime}')
270   # logging.debug(f'Exit: {fTrialEndTime}')
271   # logging.debug(f'Cycle: {nTrialPeriodSeconds}')
272   # logging.debug(f'Actual: {fTrialEndTime - fTrialStartTime}')
273   # logging.debug(f'Next: {fTrialStartTime + nTrialPeriodSeconds}')
274   # logging.debug(f'Remaining: {fTimeUntilNextTrial}')
275
276   if tdNextTrialStart.total_seconds() < 0:
277     raise Exception( 'Countdown time exceeded trial time!')
278
279   nPressedKey = show_image( anBlackRect, int(
      ↪ tdNextTrialStart.total_seconds()) * nMilliSecondsPerSecond)
280   if DUtils.keypress_matches( nPressedKey, 'escape'):
281     bEarlyExitByKeypress = True
282     break
283
284   # Destroy this trial's window (not sure if necessary!)
285   cv2.destroyAllWindows
286
287 dtSessionEnd = datetime.now()
288
289 if bEarlyExitByKeypress:
290   logging.info( f'Exited prematurely on trial { iTrial + 1}/{ len( lsTrials)}
      ↪ at { dtSessionEnd}.')
291 else:
292   logging.info( f'Session complete at { dtSessionEnd}.')
293   logging.info( f'Session took { dtSessionEnd - dtSessionStart} at { (
      ↪ dtSessionEnd - dtSessionStart).total_seconds / len( lsTrials)} seconds
      ↪ per trial.')
294   print( lsTrials)
295
296 cv2.destroyAllWindows()
```

Listing B.1: `prompter.py`: Python 3 script written to provide randomised visual que from pool of lexemes at regular time intervals, recording timing and label information to a log file.

B.2 Kinect Recording script

```
%% Record Kinect Colour, Depth and Skeleton data


%% Session Parameters

sDirOut = "D:/KinectRecordings/";
nSamplesPerSecond = 2;
nSigns = 5;
nReplicates = 20;
nTrials = nSigns * nReplicates;
nSecondsPerTrial = seconds( 5);
dBreathingRoom = seconds( 5);
dSession = nSecondsPerTrial * nTrials + dBreathingRoom;

%% Preparation

% Create and init Kinect 2 object
if exist('oKin2', 'var'), oKin2.delete; end
oKin2 = Kin2('color','depth','body');

% User control keys
cKeyEsc = char(27);
cExitKey = cKeyEsc;
cKeyEnter = char(13);
cAcknKey = cKeyEnter;

sInputPrompt = "When you are ready, start recording by making a " + ...
    "fist with your right hand, or press 'Exit' to cancel";
disp( sInputPrompt)
cPressedKey = '';

%% Create figures
% [...if only MATLAB had classes...]

% Depth
```

```
36 uD.sName = "Depth Source";
37 uD.nWidth = oKin2.cDepthWidth;
38 uD.nHeight = oKin2.cDepthHeight;
39 uD.nUpperLimit = 4000;
40 uD.nLowerLimit = 255 / uD.nUpperLimit;
41 uD.sBitDepth = "uint16";
42 uD.anImage = zeros( uD.nHeight, uD.nWidth, uD.sBitDepth);
43 uD.oFigure = figure( 'Name', uD.sName + ": " + sInputPrompt, ...
44     'NumberTitle', 'off');
45 uD.axes = axes;
46 uD.show = imshow( uD.anImage);
47 % Listen to keypress in this figure
48 set( gcf, 'keypress', 'cPressedKey = get( gcf, ''currentchar'');');
49
50 % Colour
51 uC.sName = "Colour Source";
52 uC.nWidth = oKin2.cColorWidth;
53 uC.nHeight = oKin2.cColorHeight;
54 uC.nChannels = 3; % RGB
55 uC.sBitDepth = "uint8";
56 uC.anImage = zeros( uC.nHeight, uC.nWidth, uC.nChannels, uC.sBitDepth);
57 uC.oFigure = figure( 'Name', uC.sName + ": " + sInputPrompt, ...
58     'NumberTitle', 'off');
59 uC.axes = axes;
60 uC.show = imshow( uC.anImage);
61 % Listen to keypress in this figure
62 set( gcf, 'keypress', 'cPressedKey = get( gcf, ''currentchar'');');
63
64 %% Get system warm and wait for user...
65
66 % Calculate session parameters and pre-allocate output struct array
67 dSample = seconds(1 / nSamplesPerSecond);
68 nSamples = round( nSamplesPerSecond * seconds( dSession));
69 ruOut = struct( ...
70     'nTimestamp', cell( 1, nSamples), ...
71     'dCpuTime', cell( 1, nSamples), ...
72     'anD', cell( 1, nSamples), ...
```

```
73      'anC', cell( 1, nSamples), ...
74      'uBody', cell( 1, nSamples));
75
76  lUserWantsToCancel = false;
77  i = 0;
78  while true
79      i = i + 1;
80
81      % Check for input
82      if ~isempty( cPressedKey)
83          disp( "Pressed: " + cPressedKey)
84          if cPressedKey == cExitKey
85              disp( "Exiting")
86              lUserWantsToCancel = true;
87              break;
88          elseif cPressedKey == cAcknKey
89              disp( "Moving on")
90              break
91          else
92              cPressedKey = ''; % reset mis-press
93          end
94      end
95
96      % Get sensor data
97      nValidData = oKin2.updateData;
98      if not( nValidData)
99          pause(0.02) % essential!
100         continue
101     end
102
103     % Update images
104     imshow( repmat( uint8( oKin2.getDepth * uD.nLowerLimit), [ 1 1 3]), ...
105         'Parent', uD.axes);
106     imshow( oKin2.getColor, 'Parent', uC.axes);
107
108     % Get & draw skeleton
109     [ruBodies, ~, ~] = oKin2.getBodies( 'Quat');
```

```
110     if isempty( ruBodies), continue; end
111     oKin2.drawBodies( uD.axes, ruBodies, 'depth', 5, 3, 15);
112     oKin2.drawBodies( uC.axes, ruBodies, 'color', 10, 6, 30);
113 
114     % Continue by gesture
115     if ruBodies(1).RightHandState == 3
116         disp( "Right-hand fist detected, moving on")
117         break
118     end
119 
120 end
121 close all
122 if lUserWantsToCancel
123     disp("Session cancelled")
124     oKin2.delete
125     return
126 end
127 
128 %% Run recording
129 cPressedKey = '';
130 tSessionStart = tic;
131 for iSample = 1 : round( nSamples)
132 
133     dSampleStart = seconds( cputime);
134 
135     % Allow user to break out early
136     if ~isempty( cPressedKey)
137         disp( "Pressed: " + cPressedKey)
138         if strcmp( cPressedKey, cExitKey)
139             break;
140         else
141             cPressedKey = ''; % reset mis-press
142         end
143     end
144 
145     % I will have data!
146     nValidData = oKin2.updateData;
```

```
147     [ cuBodies, ~, nTimestamp] = oKin2.getBodies( 'Quat');
148     dSampleTime = seconds( cputime);
149     lTimedOut = false;
150     iAttempt = 1;
151     while( not( nValidData) || isempty( cuBodies))
152         iAttempt = iAttempt + 1;
153         pause( 0.02) % essential!
154         nValidData = oKin2.updateData;
155         [ cuBodies, ~, nTimestamp] = oKin2.getBodies( 'Quat');
156         dSampleTime = seconds( cputime);
157         if dSampleTime - dSampleStart >= dSample
158             lTimedOut = true;
159             break
160         end
161     end
162     if lTimedOut, continue; end
163 
164     ruOut( iSample).dCpuTime = dSampleTime; % duration
165     ruOut( iSample).nTimestamp = nTimestamp; % seconds * 1E7
166     ruOut( iSample).anD = oKin2.getDepth; % uint16
167     ruOut( iSample).anC = oKin2.getColor; % uint8
168     ruOut( iSample).uBody = cuBodies( 1); % body struct
169 
170     pause( seconds(dSample + dSampleStart) - cputime)
171 
172 end
173 disp( "Session complete.")
174 toc( tSessionStart)
175 
176 sTimeStamp = strcat( datestr( now, "yyyymmmdd"), datestr( now, "dddHH"), ...
177     'h' , datestr( now, "MM"));
178 
179 sOutputFile = fullfile( sDirOut, strcat( sTimeStamp, "_Recording.mat"));
180 save( sOutputFile, 'ruOut', '-v7.3')
181 disp( "Saved to " + sOutputFile)
182 
183 %% Get map for current session (should be the same every time, but hey)
```

```
184
185 % Map from pixel ( x, y) in depth frame to pixel ( x, y) in colour frame
186 aiD2C = zeros( oKin2.cDepthWidth, oKin2.cDepthHeight, 2);
187 for iX = 1 : oKin2.cDepthWidth
188     for iY = 1 : oKin2.cDepthHeight
189         aiD2C( iX, iY, :) = oKin2.mapDepthPoints2Color( [ iX iY]);
190     end
191 end
192
193 % Map from pixel ( x, y) in colour frame to pixel ( x, y) in depth frame
194 aiC2D = nan( oKin2.cColorWidth, oKin2.cColorHeight, 2);
195 for iXD = 1 : oKin2.cDepthWidth
196     for iYD = 1 : oKin2.cDepthHeight
197         riC = reshape( aiD2C( iXD, iYD, :), 1, 2);
198         iXC = riC( 1);
199         iYC = riC( 2);
200         if iXC == 0 || iYC == 0 || ...
201                 iXC > oKin2.cColorWidth || iYC > oKin2.cColorHeight
202             continue
203         end
204         aiD = cat( 3, repmat( iXD, 3), repmat( iYD, 3));
205         aiC2D( iXC : iXC + 2, iYC : iYC + 2, :) = aiD;
206     end
207 end
208
209 save( fullfile( sDirOut, strcat( sTimeStamp, "_aiC2D.mat")), 'aiC2D')
210 save( fullfile( sDirOut, strcat( sTimeStamp, "_aiD2C.mat")), 'aiD2C')
211
212 %% Wrap up
213
214 oKin2.delete;
215 close all
```

Listing B.2: `KinectRecorder.m`: MATLAB script written to record depth frames, colour frames and body data from Microsoft Kinect 2.0.

B.3 Temporal Segmentation

```
classdef SelectFramesFromRecordings_mlapp < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure        matlab.ui.Figure
        BrowseButton    matlab.ui.control.Button
        UIAxes          matlab.ui.control.UIAxes
        SelectButton    matlab.ui.control.Button
        NextButton      matlab.ui.control.Button
        PreviousButton  matlab.ui.control.Button
        ReplaceButton   matlab.ui.control.Button
        LabelStatus     matlab.ui.control.Label
        LabelFile       matlab.ui.control.Label
    end


    properties (Access = private)
        sFile % Input file
        sPath % File path
        ruRec % Loaded recording
        ruSln = nan; % Selected recordings, nan because isempty(struct()) = 0
%         oKin2 = Kin2();
        iSample = 0; % Sample iterator
        anImage % Buffered image
        uBody % Buffered body
    end

    methods (Access = private)

        function LoadAndDisplay(app)
            LoadFile(app)
            DisplayNextFrame(app)
        end

        function LoadFile(app)
```

```
36              uLoaded = load( fullfile(app.sPath, app.sFile)); % always gives a
    ↪  1x1 struct of var...
37              app.ruRec = uLoaded.cuOut; % so peel off outer layer to target row
                ↪  of structs.
38              UpdateFileText( app)
39          end
40
41          function UpdateFileText( app)
42              SetFileText( app, "File: " + app.sFile);
43          end
44
45          function SetFileText( app, sFile)
46              app.LabelFile.Text = sFile;
47          end
48
49          function BufferNextFrame(app)
50              if app.iSample >= numel( app.ruRec), return; end
51              app.iSample = app.iSample + 1;
52              BufferFrame(app);
53          end
54
55          function BufferLastFrame(app)
56              if app.iSample <= 1, return; end
57              app.iSample = app.iSample - 1;
58              BufferFrame(app);
59          end
60
61          function BufferFrame(app)
62              app.anImage = app.ruRec( app.iSample).anC;
63              app.uBody = app.ruRec( app.iSample).uBody;
64          end
65
66          function DisplayBufferedFrame(app)
67              imshow( app.anImage, 'Parent', app.UIAxes);
68  %             app.oKin2.drawBodies( app.UIAxes, app.uBody, 'color', 10, 6,
    ↪  30);
69              UpdateStatus( app)
```

```
70          end
71
72          function DisplayNextFrame(app)
73              BufferNextFrame(app)
74              DisplayBufferedFrame(app)
75          end
76
77          function DisplayLastFrame(app)
78              BufferLastFrame(app)
79              DisplayBufferedFrame(app)
80          end
81
82          function SelectCurrentFrame(app)
83              SelectFrame( app)
84          end
85
86          function ReplacePreviousFrame(app)
87              if isstruct( app.ruSln)
88                  if numel( app.ruSln) > 1
89                      app.ruSln = app.ruSln(1 : end - 1);
90                  else
91                      app.ruSln = nan;
92                  end
93              end
94              SelectFrame( app)
95          end
96
97          function SelectFrame( app)
98              if ~isstruct( app.ruSln)
99                  app.ruSln = app.ruRec( app.iSample);
100             else
101                 app.ruSln( end + 1) = app.ruRec( app.iSample);
102             end
103             UpdateStatus(app)
104         end
105
106         function SetStatusText(app, sStatus)
```

```
107             app.LabelStatus.Text = sStatus;
108         end
109 
110         function UpdateStatus(app)
111             if ~isstruct( app.ruSln)
112                 nSelected = 0;
113             else
114                 nSelected = numel( app.ruSln);
115             end
116 
117             SetStatusText( app, "Sample " + app.iSample + '/' + numel(
                ↪  app.ruRec) + ", " + nSelected + " selected. Use buttons below
                ↪  to select one sample per sign.");
118         end
119     end
120 
121 
122     % Callbacks that handle component events
123     methods (Access = private)
124 
125         % Code that executes after component creation
126         function startupFcn(app)
127             SetStatusText( app, "Press 'Browse' to select input file.");
128         end
129 
130         % Button pushed function: NextButton
131         function NextButtonPushed(app, event)
132             DisplayNextFrame(app)
133         end
134 
135         % Button pushed function: BrowseButton
136         function BrowseButtonPushed(app, event)
137             [ app.sFile, app.sPath] = uigetfile("*.mat");
138             LoadAndDisplay(app);
139         end
140 
141         % Button pushed function: PreviousButton
```

```
142         function PreviousButtonPushed(app, event)
143             DisplayLastFrame(app)
144         end
145 
146         % Button pushed function: SelectButton
147         function SelectButtonPushed(app, event)
148             SelectCurrentFrame(app)
149         end
150 
151         % Button pushed function: ReplaceButton
152         function ReplaceButtonPushed(app, event)
153             ReplacePreviousFrame(app)
154         end
155 
156         % Close request function: UIFigure
157         function UIFigureCloseRequest(app, event)
158             if isstruct( app.ruSln)
159                 ruSln = app.ruSln; %#ok<ADPROPLC>
160                 save( fullfile( app.sPath, app.sFile + "_selectedFrames.mat"),
                    ↪  ...
161                     "ruSln", "-mat");
162             end
163             delete(app)
164         end
165     end
166 
167     % Component initialization
168     methods (Access = private)
169 
170         % Create UIFigure and components
171         function createComponents(app)
172 
173             % Create UIFigure and hide until all components are created
174             app.UIFigure = uifigure('Visible', 'off');
175             app.UIFigure.Position = [100 100 1280 760];
176             app.UIFigure.Name = 'UI Figure';
```

```
177             app.UIFigure.CloseRequestFcn = createCallbackFcn(app,
                ↪  @UIFigureCloseRequest, true);
178
179             % Create BrowseButton
180             app.BrowseButton = uibutton(app.UIFigure, 'push');
181             app.BrowseButton.ButtonPushedFcn = createCallbackFcn(app,
                ↪  @BrowseButtonPushed, true);
182             app.BrowseButton.Position = [261 729 100 22];
183             app.BrowseButton.Text = 'Browse';
184
185             % Create UIAxes
186             app.UIAxes = uiaxes(app.UIFigure);
187             title(app.UIAxes, '')
188             xlabel(app.UIAxes, '')
189             ylabel(app.UIAxes, '')
190             app.UIAxes.Position = [11 41 1260 680];
191
192             % Create SelectButton
193             app.SelectButton = uibutton(app.UIFigure, 'push');
194             app.SelectButton.ButtonPushedFcn = createCallbackFcn(app,
                ↪  @SelectButtonPushed, true);
195             app.SelectButton.Position = [701 9 100 22];
196             app.SelectButton.Text = 'Select';
197
198             % Create NextButton
199             app.NextButton = uibutton(app.UIFigure, 'push');
200             app.NextButton.ButtonPushedFcn = createCallbackFcn(app,
                ↪  @NextButtonPushed, true);
201             app.NextButton.Position = [841 9 100 22];
202             app.NextButton.Text = 'Next';
203
204             % Create PreviousButton
205             app.PreviousButton = uibutton(app.UIFigure, 'push');
206             app.PreviousButton.ButtonPushedFcn = createCallbackFcn(app,
                ↪  @PreviousButtonPushed, true);
207             app.PreviousButton.Position = [361 9 100 22];
208             app.PreviousButton.Text = 'Previous';
```

```
209
210             % Create ReplaceButton
211             app.ReplaceButton = uibutton(app.UIFigure, 'push');
212             app.ReplaceButton.ButtonPushedFcn = createCallbackFcn(app,
                ↪  @ReplaceButtonPushed, true);
213             app.ReplaceButton.Position = [501 9 100 22];
214             app.ReplaceButton.Text = 'Replace';
215
216             % Create LabelStatus
217             app.LabelStatus = uilabel(app.UIFigure);
218             app.LabelStatus.Position = [381 729 890 22];
219             app.LabelStatus.Text = 'LabelStatus';
220
221             % Create LabelFile
222             app.LabelFile = uilabel(app.UIFigure);
223             app.LabelFile.Position = [21 729 220 22];
224             app.LabelFile.Text = 'LabelFile';
225
226             % Show the figure after all components are created
227             app.UIFigure.Visible = 'on';
228         end
229     end
230
231     % App creation and deletion
232     methods (Access = public)
233
234         % Construct app
235         function app = ExtractFrames_mlapp
236
237             % Create UIFigure and components
238             createComponents(app)
239
240             % Register the app with App Designer
241             registerApp(app, app.UIFigure)
242
243             % Execute the startup function
244             runStartupFcn(app, @startupFcn)
```

```
245
246             if nargout == 0
247                 clear app
248             end
249         end
250
251         % Code that executes before app deletion
252         function delete(app)
253
254             % Delete UIFigure when app is deleted
255             delete(app.UIFigure)
256         end
257     end
258 end
```

Listing B.3: `SelectFramesFromRecordings.mlapp`: MATLAB 'App' GUI to manually select one representative frame per sign.

B.4 Spatial Segmentation and Feature Extraction

```
%% Specify recording set

% Validation 1: 5 signs 10 replicates
sFileDir = "D:/KinectRecordings/";
sInputFileName = "2019Sep17Tue20h28_Recording.mat_selectedFrames.mat";
rsLabels = [ "bad", "closed", "good", "animal", "bad", "five", "five", ...
    "bad", "animal", "five", "animal", "good", "good", "closed", ...
    "five", "five", "animal", "good", "closed", "good", "good", ...
    "animal", "closed", "bad", "five", "animal", "bad", "animal", ...
    "bad", "animal", "good", "good", "bad", "bad", "closed", "five", ...
    "bad", "animal", "closed", "bad", "closed", "animal", "five", ...
    "closed", "five", "good", "good", "five", "closed", "closed"];
sMapFileName = "2019Sep17Tue20h28_aiC2D.mat";

% % Validation 2: 5 signs 20 replicates
% sFileDir = "D:/KinectRecordings/";
% sInputFileName = "2019Sep21Sat16h07_Recording.mat_selectedFrames.mat";
% rsLabels = ["closed", "animal", "five", "bad", "animal", "good", ...
%     "closed", "five", "five", "five", "animal", "five", "good", "bad", ...
%     "good", "closed", "bad", "bad", "five", "good", "five", "good", ...
%     "five", "bad", "animal", "good", "bad", "closed", "five", "closed", ...
%     "good", "five", "good", "five", "bad", "bad", "five", "animal", ...
%     "good", "five", "animal", "closed", "good", "good", "good", "bad", ...
%     "closed", "animal", "closed", "closed", "bad", "closed", "bad", ...
%     "good", "closed", "bad", "animal", "good", "good", "closed", ...
%     "animal", "closed", "bad", "closed", "closed", "five", "animal", ...
%     "closed", "good", "closed", "bad", "bad", "good", "closed", "bad", ...
%     "animal", "closed", "animal", "animal", "five", "animal", "good", ...
%     "animal", "five", "good", "five", "closed", "animal", "animal", ...
%     "bad", "animal", "bad", "bad", "good", "five", "animal", "animal", ...
%     "five", "five", "bad"];
% rsLabels = [ rsLabels( 2 : end) rsLabels( 1)];
% sMapFileName = "2019Sep21Sat16h07_aiC2D.mat";

%% Groundwork
```

```
36
37 iRoiDiamPx = 200;
38 iMaxHandDepthMm = 120;
39 nSamples = numel( rsLabels);
40 U16toU8 = @( aiImg16) uint8( 255 * double( aiImg16) / 65535);
41
42 %% Region-of-Interest Images
43 % Load selected-frame data file from disk and
44 % get region-of-interest images for colour, depth and the binary image
45
46 [ rcanCRoi, rcanDRoi, rcalBRoi] = RoiImagesromSelection( ...
47     LoadDataFromDiskAndVerify( ...
48     fullfile( sFileDir, sInputFileName), nSamples), ...
49     ValidatedLoad( fullfile( sFileDir, sMapFileName), 'aiC2D', 'strict'), ...
50     iRoiDiamPx, iMaxHandDepthMm);
51
52 %% Verify Selection and Labelling
53 % Basic imshow/Figure based GUI for manual verification
54
55 VerifySelectionAndLabelling( rcanCRoi, rsLabels);
56
57 %% Extract features!
58
59 ruFeats = arrayfun( @( iSample) ExtractFeaturesFromRoiImages( iSample, ...
60     rcanCRoi{ iSample}, U16toU8( rcanDRoi{ iSample}), rcalBRoi{ iSample}), ...
61     1 : nSamples);
62
63 %% Save features to disk
64
65 save( fullfile( sFileDir, sInputFileName + "_extractedFeatures.mat"), ...
66     'ruFeats');
```

Listing B.4: `ProcessRecordings.m`: MATLAB script to automate processing of selected frames of Microsoft Kinect 2.0 data.

```
function [ rcanCRoi, rcanDRoi, rcalBRoi] = RoiImagesFromSelectedFrames( ...
    ruData, aiC2D, iRoiDiamPx, iMaxHandDepthMm)
% Spatially segments square about non-dominant hand from Kinect data files.
% *** Requires physically-connected Kinect! ***
% Arguments:
% ruData :  lexeme-indexed row of struct, each of which contains:
%             * a colour frame
%             * a depth frame
%             * body data
% aiC2D :   colour-frame-width x colour-frame-height x coordinate-pair
%           array of integers that map pixels locations in colour frame to
%           the location of the corresponding pixel in depth frame.
% iRoiDiamPx :  diameter of the square region-of-interest surrounding
%           the hand. Units = px.
%           [default = 200]
% iMaxHandDepthMm : maximum distance farther than the distance of the hand
%           joint to include in the binary image. Units = mm.
%           [default = 200]
%
% Returns:
% rcanCRoi : lexeme-indexed row of cells containing colour ROI as
%           iRoiDiamPx-by-iRoiDiamPx-by-3 channel array of uint8.
% rcanDRoi : lexeme-indexed row of cells containing depth ROI as
%           iRoiDiamPx-by-iRoiDiamPx array of uint16.
% rcalBRoi : lexeme-indexed row of cells containing binary image (mask) as
%           iRoiDiamPx-by-iRoiDiamPx array of logicals.
%
% Note: the ROI is centred about the hand joint horizontally but
% translated vertically up from the hand joint by 20 % of iRoiDiamPx.

% Parse optional arguments & provide default values as required
if nargin > 4
    error( "GetRoiAndMaskFromRecording:TooManyInputs", ...
        "Takes at most 3 arguments.");
end
```

```
36 if nargin < 4, iMaxHandDepthMm = 200; end
37 if nargin < 3, iRoiDiamPx = 200; end
38 if nargin < 2, load aiC2D.mat aiC2D; end
39
40 % Ensure we have the correct data type
41 iMaxHandDepthMm = cast( iMaxHandDepthMm, 'uint16');
42 iRoiDiamPx = cast( iRoiDiamPx, 'uint16');
43 aiC2D = cast( aiC2D, 'uint16');
44
45 % Calculate region of interest (ROI) offests (left & down from hand pixel)
46 iRoiXOffset = idivide( iRoiDiamPx, 2);
47 iRoiYOffset = 2 * idivide( iRoiDiamPx, 3); % only down 1/3
48
49 % Initialise Kinect and ensure it's ready
50 oKin2 = Kin2('color','depth','body');   % theoretically just need methods,
51 while true                              % but need to be sure it works...
52     % Get sensor data
53     nValidData = oKin2.updateData;
54     if ( nValidData)
55         break
56     end
57     pause( 0.02) % essential
58 end
59
60 %% Segment!
61 nSamples = numel( ruData);
62 rcanCRoi = cell( 1, nSamples);
63 rcanDRoi = cell( 1, nSamples);
64 rcalBRoi = cell( 1, nSamples);
65 for iSample = 1 : nSamples
66
67     % Localise data from input struct
68     anC = ruData( iSample).anC;
69     anD = ruData( iSample).anD;
70     uBody = ruData( iSample).uBody;
71
72     % Find hand, use to define region of interest (ROI)
```

```
73      % * Requires real connected Kinect!
74      anPos3D = uBody.Position';
75      aiPosC = oKin2.mapCameraPoints2Color( anPos3D);
76      riHandC = aiPosC( oKin2.JointType_HandLeft, :);
77      assert( min( riHandC) > 0, "Kinect not properly initialised.")
78      iXHandC = riHandC( 1);
79      iYHandC = riHandC( 2);
80      try
81          iRoiW = iRoiDiamPx;
82          iRoiH = iRoiW;
83          iRoiX = iXHandC - iRoiXOffset;
84          iRoiY = iYHandC - iRoiYOffset;
85      catch
86          disp( "i=" + iSample + " XHC=" + iXHandC + ...
87              " YHC=" + iYHandC)
88      end
89      % Crop colour image to ROI
90      rcanCRoi{ iSample} = imcrop( anC, [ iRoiX iRoiY iRoiW iRoiH]);
91
92      % Use map to register colour ROI with depth ROI
93      aiDRoi = aiC2D( iRoiX : iRoiX + iRoiW, iRoiY : iRoiY + iRoiH, :);
94      % Get depth value at those depth pixels
95      anDRoi = zeros( 201, 201, 'uint16');
96      for iX = 1 : 201
97          for iY = 1 : 201
98              riM = reshape( aiDRoi( iX, iY, :), 1, 2);
99              iXHandD = riM( 1);
100             iYHandD = riM( 2);
101             if iXHandD <= 0 || iXHandD > oKin2.cDepthWidth || ...
102                     iYHandD <= 0 || iYHandD > oKin2.cDepthHeight
103                 continue
104             end
105             anDRoi( iY, iX) = anD( iYHandD, iXHandD);
106         end
107     end
108     rcanDRoi{ iSample} = anDRoi;
109
```

```
110     % Create binary image using depth value at hand 'joint'
111     riHandD = aiC2D( iXHandC, iYHandC, :);
112     iXHandD = riHandD( 1);
113     iYHandD = riHandD( 2);
114 
115     try
116         nDepthAtHand = anD( iYHandD, iXHandD);
117     catch
118         error( "i=" + iSample + ...
119             " XHC=" + iXHandC + " YHC=" + iYHandC + ...
120             " XHD=" + iXHandD + " YHD=" + iYHandD);
121     end
122 
123     alBRoi = anDRoi <= nDepthAtHand + iMaxHandDepthMm & ...
124         anDRoi ~= 0;
125     alBRoi = imfill( alBRoi, 'holes');
126     [ naMaskWithLabelledRegions, nRegions] = bwlabel( alBRoi);
127     if nRegions > 1 % hope like crazy first region is hand...
128         alBRoi = naMaskWithLabelledRegions == 1;
129     end
130     [~, nRegions] = bwlabel( alBRoi);
131     if nRegions < 1
132         warning( "No regions in mask for sample %d", iSample);
133     end
134     rcalBRoi{ iSample} = alBRoi;
135 
136 end
137 
138 oKin2.delete
139 
140 end
```

Listing B.5: `RoiImagesFromSelectedFrames.m`: MATLAB function to spatially segment frames into ROI for colour and depth and produce a binary image (or 'mask') of the hand.

```
function VerifySelectionAndLabelling( rcanImages, rsLabels)
% Function for manual validation that selected frames match the labels.
%
% Arguments:
% rcanImages : row of cell containing images.
% rsLabels : row of strings containing labels.

if numel( rcanImages) ~= numel( rsLabels)
    error( "Number of elements do not match!")
end

riMismatches = [];
for i = 1 : numel( rcanImages)
    oFig = figure( ...
    'Name', "Press 'c' for match, 'm' for mismatch, or 'q' to quit", ...
    'NumberTitle', 'off');
    imshow( rcanImages{ i});
    title( strcat( num2str( i), ": ", rsLabels( i)));
    waitfor( oFig, 'CurrentCharacter')
    if strcmp( oFig.CurrentCharacter, 'q')
        break
    elseif strcmp( oFig.CurrentCharacter, 'm')
        riMismatches = [ riMismatches i]; %#ok<AGROW>
    end
%     close gcf
end
end
```

Listing B.6: `VerifySelectionAndLabelling.m`: MATLAB function for manual visual verification that selected images and labels match.

```
function uFeats = ExtractFeaturesFromRoiImages( ...
    iSample, anCRoi, anDRoi, alBRoi)
% Extracts numerical features for classification.
%
% Arguments:
% iSample : integer, serial number of the current sample; only used
%           for diagnostic purposes.
% anCRoi : square array by 3-channel (RGB) of uint8: the colour
%           region-of-interest.
% anDRoi : square array of uint16: the depth region-of-interest.
% alBRoi : square array of logical values: mask of the hand within the
%           region-of-interest.
%
% Returns:
% uFeats : struct containing one field per feature;
%             * feature values are single or columns

%% Geometric Features
% Using the binary mask of the hand.

uMaskProps = regionprops(alBRoi, ...
    { 'Area', 'ConvexArea', 'FilledArea', 'Perimeter'});
% We may get multiple prop structs (if there are multiple regions in image)
% so check and only take first row -- hope it's the hand!
if numel( uMaskProps) > 1
    warning('ExtractFeaturesFromHandRoi:multipleRegions', ...
        '%s detected multiple regions (%i) in sample %i.', ...
        mfilename, numel( uMaskProps), iSample)
    uMaskProps = uMaskProps( 1);
end

uFeats.nAreaPerConvexArea = uMaskProps.Area / uMaskProps.ConvexArea;
uFeats.nAreaPerFilledArea = uMaskProps.Area / uMaskProps.FilledArea;
uFeats.nPerimeterPerArea = uMaskProps.Perimeter / uMaskProps.Area;
uFeats.nPerimeterPerConvexArea = uMaskProps.Perimeter / uMaskProps.ConvexArea;
```

```
36 uFeats.nPerimeterPerFilledArea = uMaskProps.Perimeter / uMaskProps.FilledArea;
37
38 %% CV Features
39 % Functions from MATLAB Computer Vision Toolbox
40
41 % HoOG works on RGB (3D array)
42 uFeats.rnHoogC = extractHOGFeatures(anCRoi, 'CellSize', [4 4]);
43
44 % Others only work on single-channel (2D) input
45 % First prepare and label the images and the functions
46 rsImages = [ "Red", "Green", "Blue", "Grey", "Depth", "DepthHistEq"];
47 rcanImages = { anCRoi( :, :, 1), anCRoi( :, :, 2), anCRoi( :, :, 3), ...
48     rgb2gray( anCRoi), anDRoi, histeq( anDRoi)};
49 rsFunctions = [ "HoOG", "SURF", "MSER", "KAZE", "BRISK", "ORB"];
50 rcFunctions = { @( naImg) extractHOGFeatures( naImg, 'CellSize', [4 4]), ...
51         @( naImg) reshape( extractFeatures( naImg, detectSURFFeatures(
          ↪  naImg)), 1, []), ...
52         @( naImg) reshape( extractFeatures( naImg, detectMSERFeatures(
          ↪  naImg)), 1, []), ...
53         @( naImg) reshape( extractFeatures( naImg, detectKAZEFeatures(
          ↪  naImg)), 1, []), ...
54         @( naImg) reshape( getfield( extractFeatures( naImg,
          ↪  detectBRISKFeatures( naImg)), 'Features'), 1, []), ...
55         @( naImg) reshape( getfield( extractFeatures( naImg,
          ↪  detectORBFeatures( naImg, 'NumLevels', 5)), 'Features'), 1, [])};
56 % Then actually extract the features, stored in dynamically-named fields
57 for iImage = 1 : numel( rsImages)
58     naImage = rcanImages{ iImage};
59     for iFunction = 1 : numel( rsFunctions)
60         uFeats.( "rn" + rsImages{ iImage} + rsFunctions{ iFunction}) = ...
61             rcFunctions{ iFunction}( naImage);
62     end
63 end
64
65 end
```

Listing B.7: `ExtractFeaturesFromRoiImages.m`: MATLAB function that extracts numerical features for classification.

B.5 Feature Selection and Classification

```
%% Convert struct of features to 2D array, using PCA to select the components

% [ coeff, score, latent] = pca( X):
% X     : 'design matrix', n-by-p.
%       : n (rows) = 'observations', 'number of samples observed' ==
 ↪  "instances"
%       : p (cols) = 'variables', 'number of variables (features) measured' ==
 ↪  "features"
% coeff : principal component coefficients
%       : features-by-components
% score : instances-by-components

nPcaComponents = 3;

% load variables
load ruFeats;
load riLabels;

% Remove empty fields
ccacFieldNames = fieldnames( ruFeats);
rlEmptyFields = any( cell2mat( arrayfun( @( iFeat) cellfun( @( sField)
 ↪  isempty( ruFeats( iFeat).( sField)), ccacFieldNames), 1 : numel( ruFeats),
 ↪  'UniformOutput', false))', 1);
ruFeats = cell2mat( arrayfun( @( iFeat) rmfield( ruFeats( iFeat),
 ↪  ccacFieldNames( rlEmptyFields)), 1 : numel( ruFeats), 'UniformOutput',
 ↪  false));
rsFieldNames = string( fieldnames( ruFeats)');

% Remove some fields
rsTermsToRemove = [ "Red", "Green", "Blue"];
rlFieldsToRemove = any( cell2mat( arrayfun( @( sTerm) contains( rsFieldNames,
 ↪  sTerm), rsTermsToRemove, 'UniformOutput', false)'), 1);
ruFeats = cell2mat( arrayfun( @( iFeat) rmfield( ruFeats( iFeat),
 ↪  rsFieldNames( rlFieldsToRemove)), 1 : numel( ruFeats), 'UniformOutput',
 ↪  false));
```

```
27 rsFieldNames = string( fieldnames( ruFeats)');
28
29 nFields = numel( rsFieldNames);
30 nInstances = numel( riLabels);
31
32 %% Collate features
33 rcrnMaximums = arrayfun( @( uFeat) structfun( @( Field) max( size( Field)),
   ↪  uFeat), ruFeats, 'UniformOutput', false);
34 anFxIMaximums = cell2mat(rcrnMaximums);
35 cnFMinimums = min( anFxIMaximums, [], 2);
36 nFeatures = sum( cnFMinimums == 1) + sum( cnFMinimums > 1) * nPcaComponents;
37 anFxICollated = zeros( nInstances, nFeatures);
38 rsFeatures = string( zeros( 1, nFeatures));
39 fSliceField = @( Field, nMinimum) Field( 1 : nMinimum);
40
41 iColStart = 1;
42 for iField = 1 : nFields
43     if cnFMinimums( iField) > 1
44         anFxI = cell2mat( arrayfun( @( iFeat) fSliceField( ruFeats( iFeat).(
           ↪  rsFieldNames{ iField}), cnFMinimums( iField)), 1 : numel(
           ↪  ruFeats), 'UniformOutput', false)');
45         [ ~, anFxI, ~] = pca( double( anFxI), 'NumComponents',
           ↪  nPcaComponents);
46         iColEnd = iColStart + nPcaComponents - 1;
47         rsF = strcat( repmat( rsFieldNames( iField) + "PcaComp", [ 1
           ↪  nPcaComponents]) + ( 1 : nPcaComponents));
48     else
49         anFxI = vertcat( ruFeats( :).( rsFieldNames{ iField}));
50         iColEnd = iColStart;
51         rsF = rsFieldNames{ iField};
52     end
53     % Append new features on the right
54     anFxICollated( :, iColStart : iColEnd) = anFxI;
55     rsFeatures( 1, iColStart : iColEnd) = rsF;
56     iColStart = iColEnd + 1;
57 end
58
```

```
59 % transpose so the matrix conforms to 'design matrix' convention
60 %   n (rows) are "observations" = instances = lexemes
61 %   p (cols) are "variables" = features
62 anIxFCollated = anFxICollated';
63
64 %% save!
65 save rsFeatures
66 save anIxFCollated
```

Listing B.8: `FeatureStructToArrayWithPca.m`: MATLAB script that selects features from the lexeme-indexed row of `struct` containing numerical features, using PCA to reduce covariance.

```
%% load the data
clear
load riLabels
load anIxFCollated

P = anIxFCollated;
T = full( ind2vec( riLabels));

%% preprocess the data

[ mP, mSettings] = mapstd( removeconstantrows( P));

%% Principle Component Analysis

nComponents = 3;
[ coeff, score, latent, tsquared, explained, mu] = pca( ...
    mP', 'NumComponents', nComponents);

%% choose a P

ptrain = score';

%% learn!
% out loop
nouter = 10;
for noi = 1:nouter % 10x

    % get the partitions for 10-fold cross validation
    CVO = cvpartition(vec2ind(T),'KFold',10);

    fprintf('outer %d...\n', noi);

    clear acc
    for cvi = 1:CVO.NumTestSets

```

```
36 % create a new neural network without test data
37 % single hidden layer with 10 neurons
38 net = newff(ptrain(:,CVO.training(cvi)),T(:,CVO.training(cvi)),10);
39
40 % early stopping
41 % by validation (using 20 % of the not-held-out data)
42 net.divideParam.trainRatio = 0.8; % with 1.0 use everything for training
43 net.divideParam.valRatio = 0.2;
44 net.divideParam.testRatio = 0.0; % since doing kfold CV anyway...
45 % or by maximum training iterations
46 net.trainParam.epochs = 500; % when to stop
47 net.trainParam.showWindow = true;
48
49 % output transfer function
50 % Log-sigmoid maps all values to between 0 and 1
51 net.layers{2}.transferFcn = 'logsig';
52
53 % train the network with data from this fold
54 % uses default trainlm algorithm ("Levenberg-Marquardt backpropagation")
55 [net,tr{noi}(cvi)] =
   ↪ train(net,ptrain(:,CVO.training(cvi)),T(:,CVO.training(cvi)));
56
57         % test network with the held out fold data
58         out = net(ptrain(:,CVO.test(cvi)));
59         cm{noi}(:,:,cvi) = compet(out) * T(:,CVO.test(cvi))';
60
61         acc(cvi) = sum(diag(cm{noi}(:,:,cvi))/sum(sum(cm{noi}(:,:,cvi))));
62
63         fprintf('\tcv %d, acc = %0.2f\n', cvi, acc(cvi));
64
65     end
66
67     % get the total of the confusion matrix
68     tcm(:,:,noi) = sum(cm{noi},3);
69
70     [bm(noi).res,bm(noi).vec] = bookmaker(tcm(:,:,noi));
71     bmval(noi) = bm(noi).res.bookmaker;
```

```
72      accval(noi) = bm(noi).res.randAverage;
73      fprintf('\tbm %.2f\n', bm(noi).res.bookmaker);
74  end
75
76  % print results!
77  fprintf(1, 'bookmaker informedness, mean: %.2f, sd: %.2f\n', mean(bmval),
    ↪  std(bmval,0));
78  fprintf(1, 'accuracy, mean: %.2f, sd: %.2f\n', mean(accval), std(accval,0));
```

Listing B.9: `ClassifyMl.m`: MATLAB script that uses machine learning (neural network) to performed supervised learning classification.

B.6 Support Scripts

```
from pathlib import Path
import cv2
import numpy as np
import ctypes
from screeninfo import get_monitors


def list_enumeration_to_pretty_string( lItems):

  sOut = "'1'"

  if type( lItems) == type( None):
    return None
  if type( lItems) is not list:
    return sOut

  nItems = len( lItems)

  for i in range( 2, nItems + 1):
    sSep = "," if i < nItems else " or"
    sOut += f"{ sSep} '{ i}'"
  return sOut


def list_to_pretty_string( lsItems):

  if type( lsItems) == type( None) or type( lsItems) is not list:
    return None

  for i, s in enumerate( lsItems):
    if i == 0:
      sOut = f"'{ s}'"
      continue
    sSep = "," if i < len( lsItems) - 1 else " or"
    sOut += f"{ sSep} '{ s}'"
```

```
36
37    return sOut
38
39
40  def coords_of_new_monitor( iOldMonitor, iNewMonitor):
41    lmMonitors = get_monitors()
42    return ( lmMonitors[ iNewMonitor].x - lmMonitors[ iOldMonitor].x,
43      lmMonitors[ iNewMonitor].y - lmMonitors[ iOldMonitor].y)
44
45
46  def get_monitor_resolution():
47    """
48    Returns resolution of ?current monitor as (width, height).
49    Requires `ctypes` module; only works on Microsoft Windows.
50    """
51    user32 = ctypes.windll.user32
52    user32.SetProcessDPIAware()
53    return [ user32.GetSystemMetrics( 0), user32.GetSystemMetrics( 1)]
54
55
56  def get_abs_path_to_named_parent( sNamedParent):
57    p = Path.cwd()
58    # until the immediate parent is a match
59    while p.parts[ -1].lower() != sNamedParent.lower():
60
61      if len( p.parents) <= 0:
62        raise Exception( f'Reached root directory before finding target
          ↪  directory: { sNamedParent} from: { Path.cwd()}')
63
64      p = p.parents[0] # 'go up one level'
65
66    return p
67
68  def keypress_string( nPressedKey):
69    return '%d (0x%x), 2LSB: %d (%s)' % (
70        nPressedKey,
71        nPressedKey,
```

```
72        nPressedKey % 2 ** 16,
73        repr( chr( nPressedKey % 256)) if nPressedKey % 256 < 128 else '?'
74      )
75
76
77  def explore_keypresses():
78    nPressedKey = None
79    nKeyEsc = 27
80    while nPressedKey != nKeyEsc:
81      cv2.imshow( 'Press any key; Esc breaks out', np.zeros( ( 1, 1, 3),
        ↪  np.uint8))
82      nPressedKey = cv2.waitKey( 0)
83      print( keypress_string( nPressedKey))
84    cv2.destroyAllWindows()
85
86  def keypress_matches( nPressedKey, *lsKeyNames):
87    # ASCII Table
88    dAscii = {
89      'nul': 0, 'null':0,
90      'bs': 8, 'backspace': 0,
91      'tab': 9,
92      'lf':10, 'nl':10, 'new line':10, 'line feed':10, # e.g. CTRL + Enter
93      'cr':13, 'enter':13, 'return':13,
94      'esc':27, 'escape':27,
95      ' ':32, 'space':32,
96      '!':33, 'exclamation mark':33,
97      '"':34, 'double quote':34,
98      '#':35, 'number':35, 'pound':35, 'hash':35,
99      '$':36, 'dollar':36,
100     '%':37, 'percent':37,
101     '&':38, 'ampersand':38, 'and':38,
102     "'":39, 'single quote':39,
103     '(':40, 'left parenthesis':40, 'open parenthesis':40,
104     '(':41, 'right parenthesis':41, 'close parenthesis':41,
105     '*':42, 'asterisk':42,
106     '+':43, 'plus':43,
107     ',':44, 'comma':44,
```

```
    '-':45, 'hyphen':45, 'dash':45,
    '.':46, 'period':46, 'dot':46, 'full stop':46,
    '/':47, 'slash':47, 'forward slash':47, 'divide':47,
    '0':48, 'zero':48,
    '1':49, 'one':49,
    '2':50, 'two':50,
    '3':51, 'three':51,
    '4':52, 'four':52,
    '5':53, 'five':53,
    '6':54, 'six':54,
    '7':55, 'seven':55,
    '8':56, 'eight':56,
    '9':57, 'nine':57,
    ':':58, 'colon':58,
    ';':59, 'semicolon':59,
    '<':60, 'less than':60,
    '=':61, 'equals':61,
    '>':62, 'greater than':61,
    '?':63, 'question mark':63,
    '@':64, 'at':64,
    'A':65,
    'B':66,
    'C':67,
    'D':68,
    'E':69,
    'F':70,
    'G':71,
    'H':72,
    'I':73,
    'J':74,
    'K':75,
    'L':76,
    'M':77,
    'N':78,
    'O':79,
    'P':80,
    'Q':81,
```

```
145     'R':82,
146     'S':83,
147     'T':84,
148     'U':85,
149     'V':86,
150     'W':87,
151     'X':88,
152     'Y':89,
153     'Z':90,
154     '[':91, 'left bracket':91, 'open bracket':91,
155     '\\':92, 'backslash':92, 'back slash':92,
156     ']':93, 'right bracket':93, 'close bracket':93,
157     '^':94, 'circumflex':94, 'caret':94,
158     '_':95, 'underscore':95,
159     '`':65, 'grave':96,
160     'a':97,
161     'b':98,
162     'c':99,
163     'd':100,
164     'e':101,
165     'f':102,
166     'g':103,
167     'h':104,
168     'i':105,
169     'j':106,
170     'k':107,
171     'l':108,
172     'm':109,
173     'n':110,
174     'o':111,
175     'p':112,
176     'q':113,
177     'r':114,
178     's':115,
179     't':116,
180     'u':117,
181     'v':118,
```

```
182     'w':119,
183     'x':120,
184     'y':121,
185     'z':122,
186     '{':123, 'left brace':123, 'open brace':123,
187     '|':124, 'bar':124, 'vertical bar':124, 'pipe':124,
188     '}':125, 'right brace':125, 'close brace':125,
189     '~':126, 'tilde':126,
190     'del':127, 'delete':127
191   }
192
193   for sKeyName in lsKeyNames:
194     if nPressedKey == dAscii.get( sKeyName.lower(), -1):
195       return True
196   return False
```

Listing B.10: `DUtils.py`: Python 3 package containing generic support functions used by `prompter.py`.

```
function varOut = ValidatedLoad( sFileFull, sVarName, varargin)
% Loads the specified file and, if necessary, unwraps the loaded
% file to get to the target variable name.
%
% Arguments:
% sFileFull : string, absolute path and filename of target file.
% sVarName : string, name of the target variable.
% Optional : By default, ValidatedLoad will return what it loaded
%            but display a warning in the Command Window;
%            passing the tag 'strict' will prevent return if the
%            loaded variable name does not match sVarName, throwing
%            an error instead.
%
% Returns:
% varOut :  the loaded variable.

if ismember( varargin, 'strict')
    lStrict = true;
else
    lStrict = false;
end

uLoaded = load( sFileFull);

try
    varOut = uLoaded.( sVarName);
catch
    if lStrict
        error( 'ValidatedFile:strictInvalidLoad', ...
            "%s could not find '%s' in %s.", ...
            mfilename, sVarName, sFileFull)
    else
        rcFieldNames = fieldnames( uLoaded);
        varOut = uLoaded.( rcFieldNames{ 1});
        warning( 'ValidatedFile:relaxedInvalidLoad', ...
```

```
36              "%s could not find '%s' in %s; loading '%s' instead.", ...
37              mfilename, sVarName, sFileFull, rcFieldNames{ 1})
38      end
39  end
40  end
```

Listing B.11: `ValidatedLoad.m`: MATLAB function to automatically 'unwrap' loaded MAT-files and validate the variable name.

References

[1] K. S. Abhishek, L. C. K. Qubeley and D. Ho, 'Glove-based hand gesture recognition sign language translator using capacitive touch sensor', in *2016 IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC)*, Hong Kong: IEEE, 2016, pp. 334–337. DOI: 10.1109/EDSSC.2016.7785276.

[2] A. Agarwal and M. K. Thakur, 'Sign language recognition using Microsoft Kinect', in *2013 Sixth International Conference on Contemporary Computing (IC3)*, Noida, India: IEEE, Aug. 2013, pp. 181–185. DOI: 10.1109/IC3.2013.6612186.

[3] S. C. Agrawal, A. S. Jalal and R. K. Tripathi, 'A survey on manual and non-manual sign language recognition for isolated and continuous sign', *International Journal of Applied Pattern Recognition*, vol. 3, no. 2, pp. 99–134, 1 Jan. 2016. DOI: 10.1504/IJAPR.2016.079048.

[4] P. F. Alcantarilla, A. Bartoli and A. J. Davison, 'KAZE Features', in *Computer Vision – ECCV 2012*, A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato and C. Schmid, Eds., red. by D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi and G. Weikum, vol. 7577, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 214–227. DOI: 10.1007/978-3-642-33783-3_16.

[5] J. Arendsen, A. J. van Doorn and H. de Ridder, 'When and how well do people see the onset of gestures?', *Gesture*, vol. 7, no. 3, pp. 305–342, 2007. DOI: 10.1075/gest.7.3.03are.

[6] ——, 'When do people start to recognize signs?', *Gesture*, vol. 9, no. 2, pp. 207–236, 2009. DOI: 10.1075/gest.9.2.03are.

[7] H. Bay, A. Ess, T. Tuytelaars and L. Van Gool, 'Speeded-up robust features (SURF)', *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, Jun. 2008. DOI: 10.1016/j.cviu.2007.09.014.

[8] U. Bellugi and S. Fischer, 'A comparison of sign language and spoken language', *Cognition*, vol. 1, no. 2-3, pp. 173–200, Jan. 1972. DOI: 10.1016/0010-0277(72)90018-2.

[9] K. K. Biswas and S. K. Basu, 'Gesture recognition using Microsoft Kinect', in *The 5th International Conference on Automation, Robotics and Applications*, Wellington, New Zealand: IEEE, Dec. 2011, pp. 100–103. DOI: 10.1109/ICARA.2011.6144864.

[10] D. Brentari, J. Fenlon and K. Cormier, *Sign Language Phonology*. Oxford University Press, 30 Jul. 2018, vol. 1. DOI: 10.1093/acrefore/9780199384655.013.117.

[11] P. Breuer, C. Eckes and S. Müller, 'Hand gesture recognition with a novel ir time-of-flight range camera–a pilot study', in *MIRAGE 2007: Computer Vision/Computer Graphics Collaboration Techniques*, vol. 4418, 28 Mar. 2007, pp. 247–260. DOI: 10.1007/978-3-540-71457-6_23.

[12] British Deaf Association, *Dictionary of British Sign Language/English*. London: Faber and Faber Ltd, 1992.

[13] M. Calvert. (3 Nov. 2018). Improving the depth map accuracy of RealSense cameras – by an order of magnitude, [Online]. Available: http://www.calvert.ch/maurice/improving-the-depth-map-accuracy-of-realsense-cameras-by-an-order-of-magnitude/ (visited on 14/10/2019).

[14] ——, *RealSense-Calibrator*, 20 Sep. 2019. [Online]. Available: https://github.com/smirkingman/RealSense-Calibrator (visited on 14/10/2019).

[15] N. C. Camgoz, S. Hadfield, O. Koller and R. Bowden, 'Using convolutional 3d neural networks for user-independent continuous gesture recognition', in *2016 23rd International Conference on Pattern Recognition (ICPR)*, Cancun: IEEE, Dec. 2016, pp. 49–54. DOI: 10.1109/ICPR.2016.7899606.

[16] ——, 'SubUNets: End-to-end hand shape and continuous sign language recognition', in *2017 IEEE International Conference on Computer Vision (ICCV)*, Venice: IEEE, Oct. 2017, pp. 3075–3084. DOI: 10.1109/ICCV.2017.332.

[17] N. C. Camgoz, S. Hadfield, O. Koller, H. Ney and R. Bowden, 'Neural sign language translation', in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT: IEEE, Jun. 2018, pp. 7784–7793. DOI: 10.1109/CVPR.2018.00812.

[18] G. Caridakis, S. Asteriadis and K. Karpouzis, 'Non-manual cues in automatic sign language recognition', *Personal and Ubiquitous Computing*, vol. 18, no. 1, pp. 37–46, Jan. 2014. DOI: 10.1007/s00779-012-0615-1.

[19] X. Chai, G. Li, Y. Lin, Z. Xu, Y. Tang, X. Chen and M. Zhou, 'Sign language recognition and translation with Kinect', p. 2, 2013.

[20] B. K. Chakraborty, D. Sarma, M. Bhuyan and K. F. MacDorman, 'Review of constraints on vision-based gesture recognition for human–computer interaction', *IET Computer Vision*, vol. 12, no. 1, pp. 3–15, 1 Feb. 2018. DOI: 10.1049/iet-cvi.2017.0052.

[21] D. Chen, G. Li, Y. Sun, J. Kong, G. Jiang, H. Tang, Z. Ju, H. Yu and H. Liu, 'An interactive image segmentation method in hand gesture recognition', *Sensors*, vol. 17, no. 2, p. 253, 27 Jan. 2017. DOI: 10.3390/s17020253.

[22] F. Chen, J. Deng, Z. Pang, M. Baghaei Nejad, H. Yang and G. Yang, 'Finger angle-based hand gesture recognition for smart infrastructure using wearable wrist-worn camera', *Applied Sciences*, vol. 8, no. 3, p. 369, 3 Mar. 2018. DOI: 10.3390/app8030369.

[23] C. Conly, Z. Zhang and V. Athitsos, 'An integrated RGB-D system for looking up the meaning of signs', in *Proceedings of the 8th ACM International Conference on PErvasive Technologies Related to Assistive Environments - PETRA '15*, Corfu, Greece: ACM Press, 2015, pp. 1–8. DOI: 10.1145/2769493.2769534.

[24] H. Cooper, B. Holt and R. Bowden, 'Sign language recognition', in *Visual Analysis of Humans*, T. B. Moeslund, A. Hilton, V. Krüger and L. Sigal, Eds., London: Springer London, 2011, pp. 539–562. DOI: 10.1007/978-0-85729-997-0_27.

[25] H. Cooper, E.-J. Ong, N. Pugeault and R. Bowden, 'Sign language recognition using sub-units', in *Gesture Recognition*, S. Escalera, I. Guyon and V. Athitsos, Eds., Springer International Publishing, 2017, pp. 89–118. DOI: 10.1007/978-3-319-57021-1_3.

[26] A. D. C. A. Coroiu and A. Coroiu, 'Interchangeability of Kinect and Orbbec sensors for gesture recognition', in *2018 IEEE 14th International Conference on Intelligent Computer Communication and Processing (ICCP)*, Sep. 2018, pp. 309–315. DOI: 10.1109/ICCP.2018.8516586.

[27] Creative. (2013). BlasterX Senz3D, [Online]. Available: https://us.creative.com/p/peripherals/blasterx-senz3d (visited on 11/10/2019).

[28] R. Cui, H. Liu and C. Zhang, 'Recurrent convolutional neural networks for continuous sign language recognition by staged optimization', 1 Jul. 2017, pp. 1610–1618. DOI: 10.1109/CVPR.2017.175.

[29] N. Dalal and B. Triggs, 'Histograms of oriented gradients for human detection', in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, San Diego, CA, USA: IEEE, 2005, pp. 886–893. DOI: 10.1109/CVPR.2005.177.

[30] M. Dilsizian, P. Yanovich, S. Wang, C. Neidle and D. Metaxas, 'A new framework for sign language recognition based on 3D handshape identification and linguistic modeling', in *Proceedings of the 9th International Conference on Language Resources and Evaluation, LREC 2014*, European Language Resources Association (ELRA), 1 Jan. 2014, pp. 1924–1929. [Online]. Available: https://www.researchwithrutgers.com/en/publications/a-new-framework-for-sign-language-recognition-based-on-3d-handsha (visited on 07/11/2019).

[31] F. Dominio, M. Donadeo and P. Zanuttigh, 'Combining multiple depth-based descriptors for hand gesture recognition', *Pattern Recognition Letters*, vol. 50, pp. 101–111, Dec. 2014. DOI: 10.1016/j.patrec.2013.10.010.

[32] M. Elwazer and C. Bentley. (2017). Home | Kintrans, [Online]. Available: https://www.kintrans.com/ (visited on 08/10/2019).

[33] M. Elwazer, C. Bentley and H. S. A. Mohammed, 'Automatic body movement recognition and association system', U.S. Patent 20170351910A1, 7 Dec. 2017. [Online]. Available: https://patents.google.com/patent/US20170351910A1/en (visited on 08/10/2019).

[34] K. Emmorey and D. Corina, 'Lexical recognition in sign language: Effects of phonetic structure and morphology', *Perceptual and Motor Skills*, vol. 71, pp. 1227–1252, 3_suppl Dec. 1990. DOI: 10.2466/pms.1990.71.3f.1227.

[35] S. Escalera, V. Athitsos and I. Guyon, 'Challenges in multi-modal gesture recognition', in *Gesture Recognition*, S. Escalera, I. Guyon and V. Athitsos, Eds., Cham: Springer International Publishing, 2017, pp. 1–60. DOI: 10.1007/978-3-319-57021-1_1.

[36] J. Forster, O. Koller, C. Oberdörfer, Y. Gweth and H. Ney, 'Improving Continuous Sign Language Recognition: Speech Recognition Techniques and System Design', in *Proceedings of the Fourth Workshop on Speech and Language Processing for Assistive Technologies*, Grenoble, France: Association for Computational Linguistics, Aug. 2013, pp. 41–46. [Online]. Available: https://www.aclweb.org/anthology/W13-3908 (visited on 07/11/2019).

[37] J. Forster, C. Schmidt, O. Koller, M. Bellgardt and H. Ney, 'Extensions of the sign language recognition and translation corpus RWTH-PHOENIX-Weather', vol. 1, 1 May 2014.

[38] W. T. Freeman and M. Roth, 'Orientation histograms for hand gesture recognition', *International workshop on automatic face and gesture recognition*, vol. 12, pp. 269–301, 1995.

[39] J. Galka, M. Masior, M. Zaborski and K. Barczewska, 'Inertial motion sensing glove for sign language gesture acquisition and recognition', *IEEE Sensors Journal*, vol. 16, no. 16, pp. 6310–6316, Aug. 2016. DOI: 10.1109/JSEN.2016.2583542.

[40] F. Grosjean, 'Sign & word recognition: A first comparison', *Sign Language Studies*, vol. 32, pp. 195–220, 1981. DOI: 10.1353/sls.1982.0003.

[41] A. Grunnet-Jepsen, J. N. Sweetser, T. Khuong, D. Tong and J. Woodfill, *Subpixel linearity improvement for Intel® RealSense™ depth camera D400 series*, 5 Jul. 2019. [Online]. Available: https://www.intelrealsense.com/wp-content/uploads/2019/07/White_Paper_on_Subpixel_Linearity_Improvement-1.pdf.

[42] A. Grunnet-Jepsen, J. N. Sweetser and J. Woodfill, *Best-known-methods for tuning Intel® RealSense™ D400 depth cameras for best performance*, 2018. [Online]. Available: https://www.intel.com/content/dam/support/us/en/documents/emerging-technologies/intel-realsense-technology/BKMs_Tuning_RealSense_D4xx_Cam.pdf.

[43] I. Guyon, V. Athitsos, P. Jangyodsuk, B. Hamner and H. J. Escalante, 'ChaLearn gesture challenge: Design and first results', 21 Jun. 2012, pp. 1–6. DOI: 10.1109/CVPRW.2012.6239178.

[44] T. Hanke, 'HamNoSys - Representing sign language data in language resources and language processing contexts', presented at the Workshop on the Representation and Processing of Sign Languages at the Languages and Resources Evaluation Conference (LREC 04), 2004, pp. 1–6.

[45] ——, 'HamNoSys – Hamburg Notation System for Sign Languages', 14 Nov. 2007.

[46] T. Hanke and Schmaling, *Sign language notation system*, 2004. [Online]. Available: http://www.sign-lang.unihamburg.de/projects/hamnosys.html.

[47] A. Haria, A. Subramanian, N. Asokkumar, S. Poddar and J. S. Nayak, 'Hand gesture recognition for human computer interaction', *Procedia Computer Science*, vol. 115, pp. 367–374, 2017. DOI: 10.1016/j.procs.2017.09.092.

[48] M. Hassan, K. Assaleh and T. Shanableh, 'Multiple proposals for continuous arabic sign language recognition', *Sensing and Imaging*, vol. 20, no. 1, Dec. 2019. DOI: 10.1007/s11220-019-0225-3.

[49] A. Hernández-Vela, M. Á. Bautista, X. Perez-Sala, V. Ponce-López, S. Escalera, X. Baró, O. Pujol and C. Angulo, 'Probability-based dynamic time warping and bag-of-visual-and-depth-words for human gesture recognition in RGB-D', *Pattern Recognition Letters*, vol. 50, pp. 112–121, Dec. 2014. DOI: 10.1016/j.patrec.2013.09.009.

[50] C. Huang and J. Huang, 'A fast HOG descriptor using lookup table and integral image', 18 Mar. 2017. arXiv: 1703.06256 [cs]. [Online]. Available: http://arxiv.org/abs/1703.06256 (visited on 11/07/2019).

[51] J. Huang and Q. Zhang, 'Video-based sign language recognition without temporal segmentation', presented at the 32nd AAAI Conference on Artificial Intelligence, 2018. arXiv: 1801.10111. [Online]. Available: https://www.academia.edu/35798439/Video-based_Sign_Language_Recognition_without_Temporal_Segmentation (visited on 16/04/2019).

[52] J. Huang, W. Zhou, H. Li and W. Li, 'Sign language recognition using 3d convolutional neural networks', in *2015 IEEE International Conference on Multimedia and Expo (ICME)*, Jun. 2015, pp. 1–6. DOI: 10.1109/ICME.2015.7177428.

[53] K. Inoue, T. Shiraishi, M. Yoshioka and H. Yanagimoto, 'Depth sensor based automatic hand region extraction by using time-series curve and its application to Japanese fingerspelled sign language recognition', *Procedia Computer Science*, vol. 60, pp. 371–380, 2015. DOI: 10.1016/j.procs.2015.08.145.

[54] Intel, *Intel® RealSense™ D400 series product family datasheet, revision 006*, Jun. 2019. [Online]. Available: https://www.intelrealsense.com/wp-content/uploads/2019/07/Intel-RealSense-D400-Series-Datasheet-Jun-2019.pdf.

[55] ——, *Intel® RealSense™ depth module D400 series custom calibration, revision 1.5.0*, Jan. 2019. [Online]. Available: https://www.intel.com/content/dam/support/us/en/documents/emerging-technologies/intel-realsense-technology/RealSense_D400%20_Custom_Calib_Paper.pdf (visited on 14/08/2019).

[56] ——, *IntelRealSense/librealsense*, Intel® RealSense™, 13 Oct. 2019. [Online]. Available: https://github.com/IntelRealSense/librealsense (visited on 13/10/2019).

[57] P. Ji, A. Song, P. Xiong, P. Yi, X. Xu and H. Li, 'Egocentric-vision based hand posture control system for reconnaissance robots', *Journal of Intelligent & Robotic Systems*, vol. 87, no. 3-4, pp. 583–599, Sep. 2017. DOI: 10.1007/s10846-016-0440-2.

[58] F. Jiang, S. Zhang, S. Wu, Y. Gao and D. Zhao, 'Multi-layered gesture recognition with Kinect', in *Gesture Recognition*, S. Escalera, I. Guyon and V. Athitsos, Eds., Cham: Springer International Publishing, 2017, pp. 387–416. DOI: 10.1007/978-3-319-57021-1_13.

[59] T. Johnston, 'Auslan: The sign language of the australian deaf community', Ph.D. University of Sydney, 1989, 267 pp.

[60] ——, 'Transcription and glossing of sign language texts: Examples from Auslan (Australian Sign Language)', *International journal of sign linguistics*, vol. 2, no. 1, pp. 3–28, 1991. [Online]. Available: https://www.academia.edu/792079/_1991_Transcription_and_glossing_of_sign_language_texts_Examples_from_Auslan_Australian_Sign_Language_ (visited on 13/03/2019).

[61] ——, 'Language standardization and signed language dictionaries', *Sign Language Studies*, vol. 3, no. 4, pp. 431–468, 2003. DOI: 10.1353/sls.2003.0012.

[62] ——, (2014). Signbank, [Online]. Available: http://www.auslan.org.au/dictionary/ (visited on 20/03/2019).

[63] T. A. Johnston, *Signs of Australia on CD-ROM: A dictionary of Auslan (Australian Sign Language)*, 1998.

[64] T. A. Johnston and A. Schembri, *Australian Sign Language (Auslan): An Introduction to Sign Language Linguistics*. Cambridge: Cambridge University Press, 2007, xiv+323.

[65] Jungong Han, Ling Shao, Dong Xu and J. Shotton, 'Enhanced computer vision with Microsoft Kinect sensor: A review', *IEEE Transactions on Cybernetics*, vol. 43, no. 5, pp. 1318–1334, Oct. 2013. DOI: 10.1109/TCYB.2013.2265378.

[66] M. W. Kadous, 'Machine recognition of Auslan signs using PowerGloves: Towards large-lexicon recognition of sign language', p. 10, 1996.

[67] D. Kelly, J. McDonald and C. Markham, 'A person independent system for recognition of hand postures used in sign language', *Pattern Recognition Letters*, vol. 31, no. 11, pp. 1359–1368, Aug. 2010. DOI: 10.1016/j.patrec.2010.02.004.

[68] C. Keskin, F. Kıraç, Y. E. Kara and L. Akarun, 'Hand pose estimation and hand shape classification using multi-layered randomized decision forests', in *Computer Vision – ECCV 2012*, A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato and C. Schmid, Eds., red. by D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi and G. Weikum, vol. 7577, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 852–863. DOI: 10.1007/978-3-642-33783-3_61.

[69] S. Khailaie. (1 Jan. 2018). Manual setup of MinGW compilers for building Matlab MEX files in Windows, [Online]. Available: http://khailaie.com/notes/MEX/MEX-MinGW-setup-Windows.html (visited on 31/10/2019).

[70] P. V. V. Kishore, G. A. Rao, E. K. Kumar, M. T. K. Kumar and D. A. Kumar, 'Selfie sign language recognition with convolutional neural networks', *International Journal of Intelligent Systems and Applications*, vol. 10, no. 10, pp. 63–71, 8 Oct. 2018. DOI: 10.5815/ijisa.2018.10.07.

[71] S. Kita, I. van Gijn and H. van der Hulst, 'Movement phases in signs and co-speech gestures, and their transcription by human coders', in *Gesture and Sign Language in Human-Computer Interaction*, I. Wachsmuth and M. Fröhlich, Eds., red. by J. G. Carbonell, J. Siekmann, G. Goos, J. Hartmanis and J. van Leeuwen, vol. 1371, Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 23–35. DOI: 10.1007/BFb0052986.

[72] O. Koller, C. Camgoz, H. Ney and R. Bowden, 'Weakly supervised learning with multi-stream CNN-LSTM-HMMs to discover sequential parallelism in sign language videos', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2019. DOI: 10.1109/TPAMI.2019.2911077.

[73] O. Koller, J. Forster and H. Ney, 'Continuous sign language recognition: Towards large vocabulary statistical recognition systems handling multiple signers', *Computer Vision and Image Understanding*, vol. 141, pp. 108–125, Dec. 2015. DOI: 10.1016/j.cviu.2015.09.013.

[74] O. Koller, H. Ney and R. Bowden, 'May the force be with you: Force-aligned signwriting for automatic subunit annotation of corpora', in *2013 10th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG)*, Shanghai, China: IEEE, Apr. 2013, pp. 1–6. DOI: 10.1109/FG.2013.6553777.

[75] ——, 'Deep learning of mouth shapes for sign language', in *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*, Dec. 2015, pp. 477–483. DOI: 10.1109/ICCVW.2015.69.

[76] ——, 'Deep Hand: How to train a CNN on 1 million hand images when your data is continuous and weakly labelled', in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, pp. 3793–3802. DOI: 10.1109/CVPR.2016.412.

[77] O. Koller, S. Zargaran and H. Ney, 'Re-Sign: Re-aligned end-to-end sequence modelling with deep recurrent CNN-HMMs', in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017, pp. 3416–3424. DOI: 10.1109/CVPR.2017.364.

[78] O. Koller, S. Zargaran, H. Ney and R. Bowden, 'Deep Sign: Hybrid CNN-HMM for continuous sign language recognition', in *Procedings of the British Machine Vision Conference 2016*, York, UK: British Machine Vision Association, 2016, pp. 136.1–136.12. DOI: 10.5244/C.30.136.

[79] ——, 'Deep sign: Enabling robust statistical continuous sign language recognition via hybrid cnn-hmms', *International Journal of Computer Vision*, vol. 126, no. 12, pp. 1311–1325, Dec. 2018. DOI: 10.1007/s11263-018-1121-3.

[80] E. Kollorz, J. Penne, J. Hornegger and A. Barke, 'Gesture recognition with a time-of-flight camera', *International Journal of Intelligent Systems Technologies and Applications*, vol. 5, no. 3/4, pp. 334–343, 1 Jan. 2008. DOI: 10.1504/IJISTA.2008.021296.

[81] K. E. Kroemer Elbert, H. B. Kroemer and A. D. Kroemer Hoffman, 'Chapter 1 - Size and mobility of the human body', in *Ergonomics (Third Edition)*, K. E. Kroemer Elbert, H. B. Kroemer and A. D. Kroemer Hoffman, Eds., Academic Press, 1 Jan. 2018, pp. 3–44. DOI: 10.1016/B978-0-12-813296-8.00001-3.

[82] P. Kumar, H. Gauba, P. Pratim Roy and D. Prosad Dogra, 'A multimodal framework for sensor based sign language recognition', *Neurocomputing*, vol. 259, pp. 21–38, Oct. 2017. DOI: 10.1016/j.neucom.2016.08.132.

[83] P. Kumar, R. Saini, P. P. Roy and D. P. Dogra, 'A position and rotation invariant framework for sign language recognition (SLR) using Kinect', *Multimedia Tools and Applications*, vol. 77, no. 7, pp. 8823–8846, Apr. 2018. DOI: 10.1007/s11042-017-4776-9.

[84] S. Leutenegger, M. Chli and R. Y. Siegwart, 'BRISK: Binary robust invariant scalable keypoints', in *2011 International Conference on Computer Vision*, Barcelona, Spain: IEEE, Nov. 2011, pp. 2548–2555. DOI: 10.1109/ICCV.2011.6126542.

[85] T. Lewis, 'Noise-robust audio-visual phoneme recognition', Ph.D. Flinders University, South Australia, 2005, 282 pp.

[86] J. Lin, X. Ruan, N. Yu and J. Cai, 'Multi-cue based moving hand segmentation for gesture recognition', *Automatic Control and Computer Sciences*, vol. 51, no. 3, pp. 193–203, May 2017. DOI: 10.3103/S0146411617030063.

[87] G. Marin, F. Dominio and P. Zanuttigh, 'Hand gesture recognition with leap motion and kinect devices', in *2014 IEEE International Conference on Image Processing (ICIP)*, Paris, France: IEEE, Oct. 2014, pp. 1565–1569. DOI: 10.1109/ICIP.2014.7025313.

[88] ——, 'Hand gesture recognition with jointly calibrated Leap Motion and depth sensor', *Multimedia Tools and Applications*, vol. 75, no. 22, pp. 14 991–15 015, Nov. 2016. DOI: 10.1007/s11042-015-2451-6.

[89] J. Matas, O. Chum, M. Urba and T. Pajdla, 'Robust wide baseline stereo from maximally stable extremal regions', in *Proceedings of British Machine Vision Conference*, 2002, pp. 387–396. DOI: 10.5244/C.16.36.

[90] MathWorks. (2010). Newff :: Functions (Neural Network Toolbox™), [Online]. Available: https://au.mathworks.com/help/releases/R2010a/toolbox/nnet/newff.html (visited on 02/11/2019).

[91] ——, (2019). Feature detection and extraction, [Online]. Available: https://au.mathworks.com/help/vision/feature-detection-and-extraction.html (visited on 02/11/2019).

[92] ——, (2019). MAT-file versions, [Online]. Available: https://au.mathworks.com/help/matlab/import_export/mat-file-versions.html (visited on 02/11/2019).

[93] ——, (2019). Measure properties of image regions - MATLAB regionprops, [Online]. Available: https://au.mathworks.com/help/images/ref/regionprops.html (visited on 02/11/2019).

[94] ——, (2019). Packages Create Namespaces, [Online]. Available: https://au.mathworks.com/help/matlab/matlab_oop/scoping-classes-with-packages.html (visited on 30/10/2019).

[95] R. K. McConnell, 'Method of and apparatus for pattern recognition', U.S. Patent 4567610A, 1986.

[96] Microsoft. (21 Oct. 2014). Joint Structure, [Online]. Available: https://docs.microsoft.com/en-us/previous-versions/windows/kinect/dn758664(v%3dieb.10) (visited on 31/10/2019).

[97] ——, (21 Oct. 2014). JointType Enumeration, [Online]. Available: https://docs.microsoft.com/en-us/previous-versions/windows/kinect/dn758663(v%3dieb.10) (visited on 31/10/2019).

[98] ——, (19 Dec. 2016). Kinect hardware, [Online]. Available: https://web.archive.org/web/20161219032759/https://developer.microsoft.com/en-us/windows/kinect/hardware (visited on 11/10/2019).

[99] C. Ming and X. Jianbo, 'Fast gesture recognition algorithm based on superpixel distribution and EMD metric', in *Recent Developments in Intelligent Systems and Interactive Applications*, F. Xhafa, S. Patnaik and Z. Yu, Eds., vol. 541, Cham: Springer International Publishing, 2017, pp. 267–274. DOI: 10.1007/978-3-319-49568-2_38.

[100] Z. Mo and U. Neumann, 'Real-time hand pose recognition using low-resolution depth images', in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2, Jun. 2006, pp. 1499–1505. DOI: 10/bpft92.

[101] W. Nai, Y. Liu, D. Rempel and Y. Wang, 'Fast hand posture classification using depth features extracted from random line segments', *Pattern Recognition*, vol. 65, pp. 1–10, May 2017. DOI: 10.1016/j.patcog.2016.11.022.

[102] K. Ogawara, J. Takamatsu, K. Hashimoto and K. Ikeuchi, 'Grasp recognition using a 3D articulated model and infrared images', in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, vol. 2, Oct. 2003, 1590–1595 vol.2. DOI: 10.1109/IROS.2003.1248871.

[103] Open Source Robotics Foundation and I. Saito. (2 May 2015). Bags - ROS Wiki, [Online]. Available: http://wiki.ros.org/Bags (visited on 15/10/2019).

[104] B. Peterson. (3 Nov. 2008). PEP 373 – Python 2.7 release schedule, [Online]. Available: https://www.python.org/dev/peps/pep-0373/ (visited on 13/10/2019).

[105] V. Pitsikalis, A. Katsamanis, S. Theodorakis and P. Maragos, 'Multimodal gesture recognition via multiple hypotheses rescoring', in *Gesture Recognition*, S. Escalera, I. Guyon and V. Athitsos, Eds., Cham: Springer International Publishing, 2017, pp. 467–496. DOI: 10.1007/978-3-319-57021-1_16.

[106] L. E. Potter, J. Araullo and L. Carter, 'The Leap Motion controller: A view on sign language', in *Proceedings of the 25th Australian Computer-Human Interaction Conference on Augmentation, Application, Innovation, Collaboration - OzCHI '13*, Adelaide, Australia: ACM Press, 2013, pp. 175–178. DOI: 10.1145/2541016.2541072.

[107] D. M. W. Powers, *Evaluation: From precision, recall and F-factor to ROC, informedness, markedness & correlation*, Dec. 2007. [Online]. Available: http://david.wardpowers.info/BM/Evaluation_SIETR.pdf.

[108] G. A. Rao, K. Syamala, P. V. V. Kishore and A. S. C. S. Sastry, 'Deep convolutional neural networks for sign language recognition', in *2018 Conference on Signal Processing And Communication Engineering Systems (SPACES)*, Vijayawada: IEEE, Jan. 2018, pp. 194–197. DOI: 10.1109/SPACES.2018.8316344.

[109] Z. Ren, J. Yuan and Z. Zhang, 'Robust hand gesture recognition based on finger-earth mover's distance with a commodity depth camera', in *Proceedings of the 19th ACM International Conference on Multimedia - MM '11*, Scottsdale, Arizona, USA: ACM Press, 2011, p. 1093. DOI: 10.1145/2072298.2071946.

[110] D. Retek, D. Palhazi, M. Kajtar, A. Alvarez, P. Pocsi, A. Nemeth, M. Trosztel, Z. Robotka and J. Rovnyai, 'Computer vision based sign language interpreter', pat. WO2019094618A1, 16 May 2019. [Online]. Available: https://patents.google.com/patent/WO2019094618A1/en?inventor=Zsolt+Robotka&oq=Zsolt+Robotka (visited on 08/10/2019).

[111] Z. Robotka. (2018). Home - SignAll, [Online]. Available: https://www.signall.us/ (visited on 08/10/2019).

[112] E. Rublee, V. Rabaud, K. Konolige and G. Bradski, 'ORB: An efficient alternative to SIFT or SURF', in *2011 International Conference on Computer Vision*, Barcelona, Spain: IEEE, Nov. 2011, pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544.

[113] D. Rybach, C. Gollan, G. Heigold, B. Hoffmeister, J. Lööf, R. Schlüter and H. Ney, 'The RWTH Aachen University open source speech recognition system', presented at the 10th Annual Conference of the International Speech Communication Association, 2009, pp. 2111–2114.

[114] B. Saunders and J. Balich, *USB Promoter Group announces USB4 specification*, 4 Mar. 2019. [Online]. Available: https://usb.org/sites/default/files/2019-03/USB_PG_USB4_DevUpdate_Announcement_FINAL_20190226.pdf (visited on 16/03/2019).

[115] SergentMT. (16 Jul. 2014). Kinect Version 2 Depth Frame to .mat File Exporter Tool, [Online]. Available: https://www.codeproject.com/Tips/819613/Kinect-Version-Depth-Frame-to-mat-File-Exporter (visited on 30/10/2019).

[116] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook and R. Moore, 'Real-time human pose recognition in parts from single depth images', *Communications of the ACM*, vol. 56, no. 1, p. 116, 1 Jan. 2013. DOI: 10.1145/2398356.2398381.

[117] W. C. Stokoe, 'Sign language structure: An outline of the visual communication systems of the American deaf (2005 reprint)', *The Journal of Deaf Studies and Deaf Education*, vol. 10, no. 1, pp. 3–37, 1960. DOI: 10.1093/deafed/eni001.

[118] Summer Institiute of Lingistics, Inc. (SIL). (3 Dec. 2015). Deixis, [Online]. Available: https://glossary.sil.org/term/deixis (visited on 19/03/2019).

[119] G. A. ten Holt, A. J. V. Doorn, M. J. Reinders, E. A. Hendriks and H. D. Ridder, 'Human-inspired search for redundancy in automatic sign language recognition', *ACM Transactions on Applied Perception (TAP)*, vol. 8, no. 2, p. 15, 2011. DOI: 10.1145/1870076.1870083.

[120] G. A. ten Holt, P. Hendriks and T. Andringa, 'The eye of the beholder: Automatic recognition of Dutch sign language', Masters, University of Groningen, The Netherlands, 2004, 93 pp.

[121] ——, 'Why don't you see what I mean? Prospects and limitations of current automatic sign recognition research', *Sign Language Studies*, vol. 6, no. 4, pp. 416–437, 2006. DOI: 10.1353/sls.2006.0024.

[122] G. A. ten Holt, M. J. Reinders, E. A. Hendriks, H. de Ridder and A. J. van Doorn, 'Influence of handshape information on automatic sign language recognition', in *International Gesture Workshop*, Springer, 2009, pp. 301–312. DOI: 10.1007/978-3-642-12553-9_27.

[123] J. R. Terven. (6 Jul. 2017). Kinect 2 Interface for Matlab, [Online]. Available: https://au.mathworks.com/matlabcentral/fileexchange/53439 (visited on 30/10/2019).

[124] J. R. Terven and D. M. Córdova-Esparza, 'Kin2. A Kinect 2 toolbox for MATLAB', *Science of Computer Programming*, vol. 130, pp. 97–106, 15 Nov. 2016. DOI: 10.1016/j.scico.2016.05.009.

[125] C. Vogler and D. Metaxas, 'Handshapes and movements: Multiple-channel american sign language recognition', in *Gesture-Based Communication in Human-Computer Interaction*, A. Camurri and G. Volpe, Eds., red. by G. Goos, J. Hartmanis and J. van Leeuwen, vol. 2915, Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 247–258. DOI: 10.1007/978-3-540-24598-8_23.

[126] C. Wang, Z. Liu and S.-C. Chan, 'Superpixel-based hand gesture recognition with Kinect depth camera', *IEEE Transactions on Multimedia*, vol. 17, no. 1, pp. 29–39, Jan. 2015. DOI: 10.1109/TMM.2014.2374357.

[127] X. Wang, Y. Şekercioğlu, T. Drummond, V. Frémont, E. Natalizio and I. Fantoni, 'Relative pose based redundancy removal: Collaborative RGB-D data transmission in mobile visual sensor networks', *Sensors*, vol. 18, no. 8, p. 2430, 26 Jul. 2018. DOI: 10.3390/s18082430.

[128] M. Wilson. (25 Oct. 2017). Exclusive: Microsoft has stopped manufacturing the Kinect, [Online]. Available: https://www.fastcompany.com/90147868/exclusive-microsoft-has-stopped-manufacturing-the-kinect (visited on 11/10/2019).

[129] Z. Zafrulla, H. Brashear, T. Starner, H. Hamilton and P. Presti, 'American sign language recognition with the Kinect', in *Proceedings of the 13th International Conference on Multimodal Interfaces - ICMI '11*, Alicante, Spain: ACM Press, 2011, p. 279. DOI: 10.1145/2070481.2070532.

[130] Z. Zhang, 'Microsoft Kinect sensor and Its effect', *IEEE Multimedia*, vol. 19, no. 2, pp. 4–10, Feb. 2012. DOI: 10.1109/MMUL.2012.24.