

# Gesture Knitter: A Hand Gesture Design Tool for Head-Mounted Mixed Reality Applications

George B. Mo  
Department of Engineering  
University of Cambridge  
Cambridge, United Kingdom  
gm621@cam.ac.uk

John J. Dudley  
Department of Engineering  
University of Cambridge  
Cambridge, United Kingdom  
jjd50@cam.ac.uk

Per Ola Kristensson  
Department of Engineering  
University of Cambridge  
Cambridge, United Kingdom  
pok21@cam.ac.uk

## ABSTRACT

Hand gestures are a natural and expressive input method enabled by modern mixed reality headsets. However, it remains challenging for developers to create custom gestures for their applications. Conventional strategies to bespoke gesture recognition involve either hand-crafting or data-intensive deep-learning. Neither approach is well suited for rapid prototyping of new interactions. This paper introduces a flexible and efficient alternative approach for constructing hand gestures. We present Gesture Knitter: a design tool for creating custom gesture recognizers with minimal training data. Gesture Knitter allows the specification of gesture primitives that can then be combined to create more complex gestures using a visual declarative script. Designers can build custom recognizers by declaring them from scratch or by providing a demonstration that is automatically decoded into its primitive components. Our developer study shows that Gesture Knitter achieves high recognition accuracy despite minimal training data and delivers an expressive and creative design experience.

## CCS CONCEPTS

• Human-centered computing → Gestural input; Mixed / augmented reality; Virtual reality.

## KEYWORDS

gestures, virtual reality, augmented reality

### ACM Reference Format:

George B. Mo, John J. Dudley, and Per Ola Kristensson. 2021. Gesture Knitter: A Hand Gesture Design Tool for Head-Mounted Mixed Reality Applications. In *CHI Conference on Human Factors in Computing Systems (CHI '21)*, May 8–13, 2021, Yokohama, Japan. ACM, New York, NY, USA, 13 pages. <https://doi.org/10.1145/3411764.3445766>

## 1 INTRODUCTION

The emerging potential of head-mounted mixed reality devices highlights the need for expressive gesture-based interaction. Hand gestures are one of the primary ways of interaction in this setting, allowing the user to interact with virtual objects and issue commands.

Although hand gestures have been extensively studied in the context of recognition and robustness in interaction [15, 31, 45, 46], there has been little focus on protocols for expressive design with low data overhead. As a result, many designers limit their efforts to creating simple gestures. This inhibits the development of applications where more expressive hand gestures are desirable, for example in game design [21, 24] or 3D drawing [7].

There have been many tools developed for symbolic gesture design in the context of 2D pen and touch-oriented environments [13, 22, 23, 34]. This paper addresses an analogous problem for a 3D mixed reality environment. Two central principles guide its design.

First, the tool should allow designers to easily implement novel gestures by avoiding the need for a designer to laboriously provide new demonstrations and train new recognizers for each desired gesture. An alternative approach is to synthesize new gestures from previously declared gesture components to build more complex and expressive gestures. This promotes recycling of gesture components and recognizers, aligning with established software engineering principles on reuse and streamlining the design process. Further, instead of having gestures as distinct classes, partitioning a gesture into distinct components, or features, allows more combinations of gesture components to be created, yielding more complex and expressive gestures.

Second, recognition should consider the distinct attributes of hand gestures. For instance, take the gesture of the hand traveling in circles while in a fist. Recognition based on the entire motion explicitly disregards the main gesture features, such as the fist and the repetitive circular motion. As a result, variations such as the number of loops and the speed may confuse the classifier. A better approach is to incorporate the attributes of the gesture directly by defining them directly in terms of simpler gesture components. This allows the recognizer to classify based on attributes, not simply through similarity with previously collected samples.

This paper presents *Gesture Knitter*: a new tool for designers addressing these two limitations and enabling the design of novel hand gestures in a mixed reality setting with high data efficiency. *Gesture Knitter* was conceived to facilitate the process of synthesizing and encoding novel gestures while adhering to software engineering principles of reuse and iterative design. The primary focus of *Gesture Knitter* is in supporting rapid integration of gesture recognition functionality into mixed reality applications. Other aspects, such as the gesture elicitation process and underlying recognition technique, are secondary concerns given this objective and not directly addressed at this stage. In this paper, the tool is demonstrated using the Microsoft HoloLens 2 optical see-through head-mounted display, however, the approach is device agnostic provided that



This work is licensed under a Creative Commons Attribution International 4.0 License.

CHI '21, May 8–13, 2021, Yokohama, Japan

© 2021 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-8096-6/21/05.

<https://doi.org/10.1145/3411764.3445766>

full joint hand tracking is available. Following the lead of 2D pen-based gesture design tools, such as Gesture Script [22], we allow designers to declare new hand gesture classes by declaring scripts and providing demonstrations. However, we also introduce novel capabilities into our application that streamline the design process. Significantly, gesture components are partitioned between gross and fine components. Gross hand components (or primitives) are gestures characterized by the palm's motion, such as moving the hand forwards and backwards. Fine hand primitives are gestures characterized by the motion of the fingers relative to the palm, such as pointing and pinching. Gesture Knitter also provides a visual declarative script interface that allows designers to design expressive complex hand gestures as sequences of user-defined primitives. The script produces new gesture recognizers without requiring explicit samples of the new gesture. Furthermore, the script declaration of a new complex gesture can be inferred from a single user demonstration. Gesture Knitter automatically generates synthetic gesture samples from previous demonstrations, allowing designers to quickly add greater variation to their training examples. Moreover, we provide a tool to provide feedback to the designer on how distinct a newly declared gesture is compared to gestures that have been previously declared.

The key contributions of this work are:

- Gesture Knitter: a design tool for rapid prototyping both one and two-hand gestures for mixed reality applications.
- A user-in-the-loop strategy for efficiently generating supplementary synthetic sample data to improve recognition performance.
- A diagnostic tool for promoting the discernability of gestures.
- A user evaluation of Gesture Knitter yielding an improved understanding of the behaviors and needs related to rapid prototyping of hand gestures for mixed reality.

The remainder of the paper is organized as follows. Section 2 reviews the related work informing the design of Gesture Knitter. The high-level design framework of Gesture Knitter is then presented in Section 3. The lower-level implementation details enabling efficient training and recognition of gestures are described in Section 4. Finally, we evaluate our tool with a designer study and evaluate the performance of Gesture Knitter with empirical data in Section 5. We conclude with limitations and future work and final observations on the outcomes of this work.

## 2 RELATED WORK

There has been much previous work into hand gesture recognition models with different learning based approaches. Visual based approaches include orientation histograms [12], metric based learning [20, 33], neural networks [26], and decision trees [9], while time-series data approaches include Hidden Markov Models [19, 25, 45]. With regards to deconstructing a gesture into separate components, Narayana et al. [28] decomposes the fine and gross hand movement channels to improve gesture recognition with neural networks. An area that applies these hand gesture recognition techniques to a semantically meaningful domain is sign language recognition, which has received extensive attention [29, 38, 39, 46]. In addition, online recognition of hand gestures have been extensively explored

with neural network based approaches [47] and modifications to Hidden Markov Models [8, 14]. Semantic approaches to general hand gesture recognition and design include stochastic context free grammars [5] and the use of motion primitives for human-robot interaction [36]. Although our approach learns its hand gesture recognizer from samples using Hidden Markov Models, we enhance learning and expressiveness by decomposing primitives into fine and gross components in a grammar-based visual declarative script.

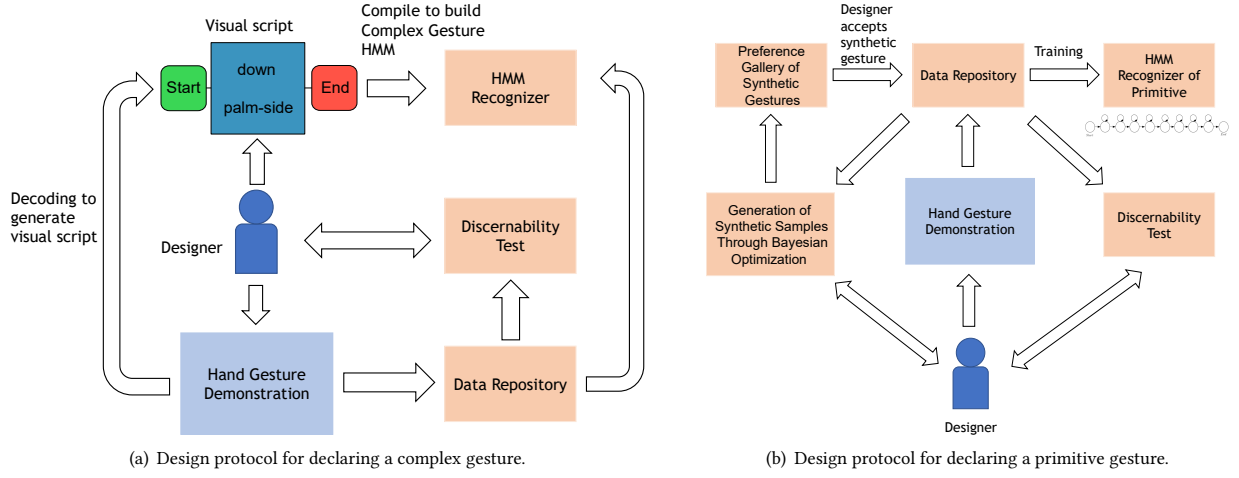
Previous approaches that apply scripts to design and declare gestures are primarily in a 2D sketch or touchscreen environment. For the selection of hand gesture vocabulary, previous approaches have been focused on investigating the universality and intuition of hand gestures [40, 42] and attempts have been made to optimize the vocabulary by performing multiobjective optimization over psychophysiological and technical performance measures [37, 41]. Nacenta et al. [27] explored creating hand gesture sets by evaluating memorability for pre-designed and user-defined gestures. Although there has been some general guidance in the selection of AR/VR hand gestures [30] and attempts to directly translate hand gestures to script language for computer-aided design [15], no prior work has created a comprehensive system to simultaneously aid design and provide recognizers in the creation of novel hand gestures in a head-mounted mixed reality setting.

Our work addresses this gap as Gesture Knitter interactively allows the designer to declare new gestures with a visual script, interactively train the recognizer, and generate new synthetic samples for 3D hand gestures. Declarative scripts have been employed in methods such as LADDER [13] and predefined primitives from perceptual components have been used in sketch recognition systems [34]. While our method does use declarative scripts to define new complex hand gestures, the primitives are partitioned between gross and fine components in order to give the designer more flexibility in design. Example-based script approaches have been used for 2D touch-screen multi-touch gestures in Gesture Coder [23], and rendering scripts to aid the recognition of 2D sketches were explored in Gesture Script [22]. Our method attempts to encapsulate the learning of a user-defined complex gesture by Viterbi decoding to generate a visual declarative script from which the user can give direct feedback to recognize the newly declared gestures.

## 3 GESTURE KNITTER'S DESIGN FRAMEWORK

In this section, we outline the key components of Gesture Knitter. These are: 1) a component allowing the declaration of new fine and gross primitives; 2) a component providing the option to generate synthetic primitive trajectories and to test gesture discernment; and 3) a component providing complex gesture declaration and recognizer synthesis using the visual declarative script. The design pipeline for declaring new primitives and new complex gestures is shown in Figure 1.

To illustrate the practical usage of Gesture Knitter we follow a hypothetical user, Alice, as she designs three expressive gestures for her gaming application, *Adventures in Wonderland*. The three gestures that she wants to design are *Shrink*, *MadRiddles* and *Grow*. In *Shrink*, the hand travels down while the palm is facing to the side. In *MadRiddles*, the hand first travels forward with the index finger



**Figure 1:** (a) After giving a hand demonstration for a complex gesture, the designer can choose to use the decoding option to automatically generate the visual script, or declare it from scratch. (b) After giving some demonstrations of the primitive, the designer can choose to either generate synthetic samples or provide more demonstrations to train the primitive recognizer.

extended, then the hand moves to the right with the palm facing downwards, and finally the hand retracts back to the body with a thumbs up. In *Grow*, the hand first moves in counterclockwise circles while holding a clenched fist for two to three loops, and then the hand is moved forward with the palm facing downwards. These three gestures are visualized in Table 1.

### 3.1 Declaring Primitives


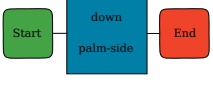
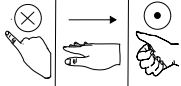
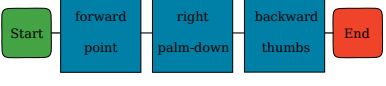
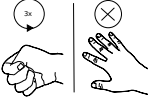
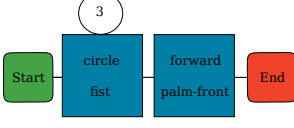
Alice sees that the predeclared primitives in the library do not sufficiently express the complex gestures that she had in mind. To rectify this, Gesture Knitter supports interactive training of a recognizer from user demonstrations. Alice first records a few examples of the hand trajectories corresponding to the new fine or gross primitive hand gesture defined. We refer to gross hand gestures as the gestures characterized only by the trajectory of the center of mass of the hand as it moves through space. We refer to fine hand gestures as the gestures characterized by the trajectory of the fingers relative to the center of mass of the hand. As with previous 2D gesture design toolkits such as Gesture Script [22], our system preserves the interaction of quickly training a recognizer from demonstrations.

To build the *Shrink* gesture, Alice must declare the gross primitive of *down* and fine primitive of *palm-side*. For *Grow*, the gross primitives would be *circle* and *forward*, while the fine primitives would be *fist* and *palm-front*. For *MadRiddles*, the gross primitives are *forward*, *right*, *backward*; the fine primitives are *point*, *palm-down*, and *thumbs*. Note that once one primitive is declared, declaring another complex gesture does not need re-declaration of the primitive: that same primitive can just be used directly.

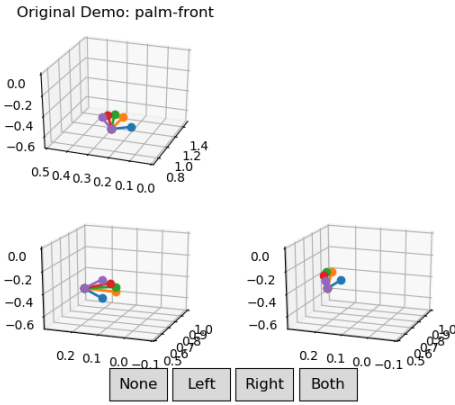
### 3.2 Data Synthesis

Gesture Knitter also provides an auxiliary tool to generate new synthetic samples by adding variation to given hand demonstrations. After asking the system to generate new synthetic samples,

**Table 1: Visual script declarations of the three complex gestures *Shrink*, *MadRiddles* and *Grow*. The hand actions show how the gesture is performed. Note that the visual script nodes are in chronological order from left to right.**

Gesture	Visual Script Declaration
 <i>Shrink</i>	
 <i>MadRiddles</i>	
 <i>Grow</i>	

animation frames presenting the dynamic hand gesture primitives are shown to Alice, as shown in Figure 2. The preference gallery presents two synthesized demonstrations, as well as the original demonstration for comparison. Alice then indicates which of the animations from the preference gallery, if any, are suitable demonstrations of the desired primitive gesture. The recognizer for this new primitive gesture is trained using these synthesized gestures, hence increasing the recognizer’s robustness to different variations.



**Figure 2: Preference gallery interface for generating synthetic samples. The user selects the synthetic samples to include in the dataset by comparing them to the original demonstration for the purpose of training the primitive recognizers.**

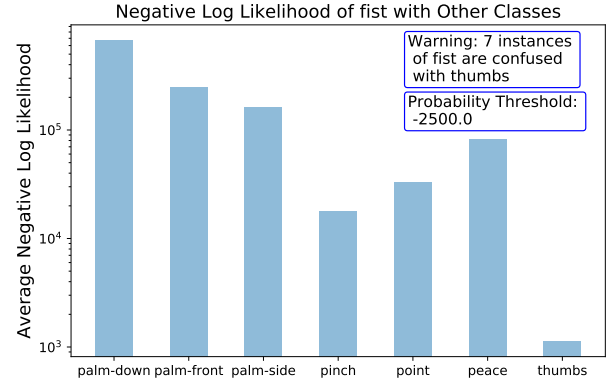
Separate sample generators are created and trained for fine and gross primitive gesture sets.

### 3.3 Discernability of Gestures

A fundamental problem for the designer is creating gesture primitives and complex gestures that are easily distinguished by a recognizer. To aid in this task, Alice can test a newly declared gesture with the discernability tool that detects the similarity of this gesture in relation to previously declared gestures of the same category. For instance, if Alice declares the fine primitive of *fist*, she can use this tool to compare this new gesture's demonstrations to previously declared fine primitives. If there is a high degree of similarity between this new gesture and previously declared gestures, the tool notifies the designer of the possibility of discernment issues between them. For instance, comparing her demonstrations of *fist* with previously declared fine primitives show that multiple instances of her demonstrations are possibly confused with *thumbs*, as shown in Figure 3. When faced with this issue, Alice can choose to either alter the design of the novel gesture or provide additional samples to the two similar gestures to improve their discernability during recognition.

### 3.4 Declaring Gestures with Visual Scripts

Gesture Knitter supports the declaration of complex gestures with a visual scripting interface by concatenating primitive gross and fine hand gestures as shown in Figure 4. We assume for now that Alice is happy with the library of fine and gross primitive hand gestures that are already declared. Alice starts by selecting *New Start Node* by declaring the start node of the gesture. A complex gesture is composed of a sequence of nodes, and each node has fields denoting the gross component and the fine component. She can declare a node with the fields by clicking on *New Hand Node*, and she can edit that node by selecting the option *Edit Node Properties* or delete it by selecting *Delete Node*. There are looping options for each node, as seen in the declaration for *Grow* in Table 1, which



**Figure 3: Output of the discernability tool when tested with the *fist* fine primitive. It produces a chart giving the average negative log likelihoods of the *fist*'s demonstrations given the recognizers for the other fine primitives.**

can be assigned by clicking *Add Loop*. Alice can connect two nodes together by selecting both nodes and then choosing *Add Connection*. The chronological ordering of the nodes follows sequentially from the node arrangement with respect to the start node. Alice ends the gesture by declaring the end node by clicking on *New End Node*. During the declaration, Alice can click the **Validate** button to check if the script obeys the syntax. After the declarative script is built, Alice clicks the **Compile** button and an automatic recognizer is built for the new complex gesture composed of the trained recognizers for the primitive hand gestures. Alice can now use these compiled recognizers in her desired application.

### 3.5 Automatically Generating Declarative Scripts

Gesture Knitter supports the generation of a declarative script directly from a complex gesture demonstration. For instance in *Grow*, Alice records a demonstration on her mixed reality headset. After importing the demonstration into Gesture Knitter, she presses the **Decode** button in the toolbar and the system decodes the primitive components of the hand gesture by outputting an inferred visual declarative script of the new complex gesture. The system essentially infers the new gesture's syntactic representation using its priors for the primitive gestures. If there are errors with the script generated, another hand demonstration can be provided, or the visual script can be edited directly to correct the errors. The edited script is then verified and compiled into a recognizer as in the previous section.

For the specific case of designing two-handed complex gestures, the same process is followed but now the trajectories are recorded with both hands, and each of the primitive component nodes in the visual declarative script consists of five fields: the fine and gross primitive components of each of the left and right hands and the number of loops, as shown in the Figure 5. The primitives that the designer can declare now not only consist of primitives specifically

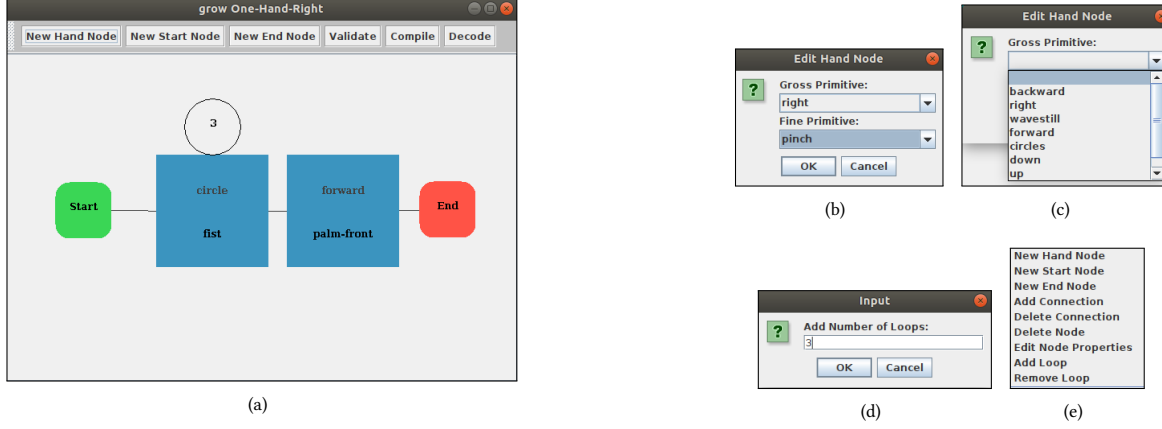


Figure 4: (a) Full visual script declaration for *Grow*. Note that the green start node and red end node indicate the start and end of the declaration. Edges between the nodes indicate the sequential order of the nodes. Each node has two fields indicating the fine and gross components. (b) Right clicking or declaring a new node reveals this menu from which the designer can select the fine and gross components of the node. (c) The pull-down menu showing the primitive gesture options to select from. (d) The menu for declaring the number of loops on a specific node. (e) The context menu shown when right clicking on a node.

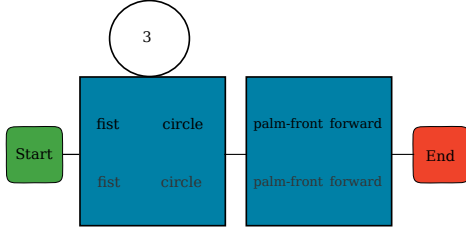


Figure 5: A visual declarative script for *Grow* for a two-handed scenario. Note that the top of each node represents the left gesture component and the bottom represents the right gesture component.

for the right hand, but also gross and fine primitives particular to the left hand.

## 4 EFFICIENTLY TRAINING AND RECOGNIZING GESTURES

We now present the algorithms that Gesture Knitter uses in order to learn recognizers, decode demonstrations, and generate synthetic samples to improve recognition rates.

### 4.1 Data Preprocessing

Hand gestures are represented as time-series trajectories  $\{\mathbf{x}_i\}_{i=1}^n$  recorded by the hand tracker on the HoloLens 2. For one-handed gestures, we take each  $\mathbf{x}_i$  to be 18 dimensional, where the first three entries represent the  $x, y, z$  components of the palm, and the remaining 15 represent the  $x, y, z$  of the thumb, index, middle, ring, and pinky fingers. For two-handed gestures, we take each  $\mathbf{x}_i$  to be 36 dimensional, each of the two 18 dimensional halves corresponding to the right and left hands. For the design protocol, we set  $n=300$ , corresponding to a five second recording for the

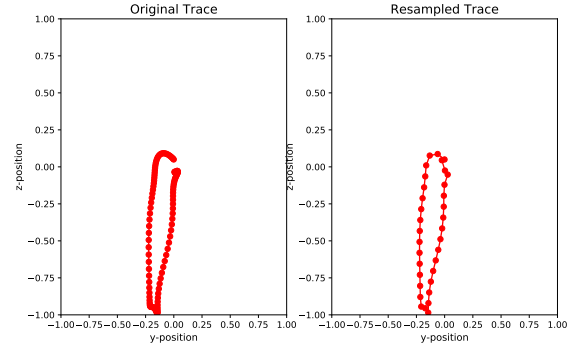


Figure 6: (left) The original smoothed time-series velocity trace of a downward motion projected on a  $yz$ -plane. (right) An illustration of equidistant resampling of the trace.

primitives and complex gestures. We first transform the trajectory to a body-centered frame at the sternum to establish a stationary frame of reference.

For the gross component of the recorded gesture, we only take into account the time-series given by the palm coordinates. We first smooth the palm coordinates using a Gaussian filter and we take the first differences of the smoothed coordinates to yield the velocity of the trajectory. As the defining characteristic of the gross gesture is the shape of the trajectory, we resample the trajectory points so they are evenly spaced on the curve, where the number of points is proportional to the curve length (see Figure 6). For the fine component, we take the position of the fingers relative to the palm for the whole trajectory.

## 4.2 Learning Gesture Recognizers

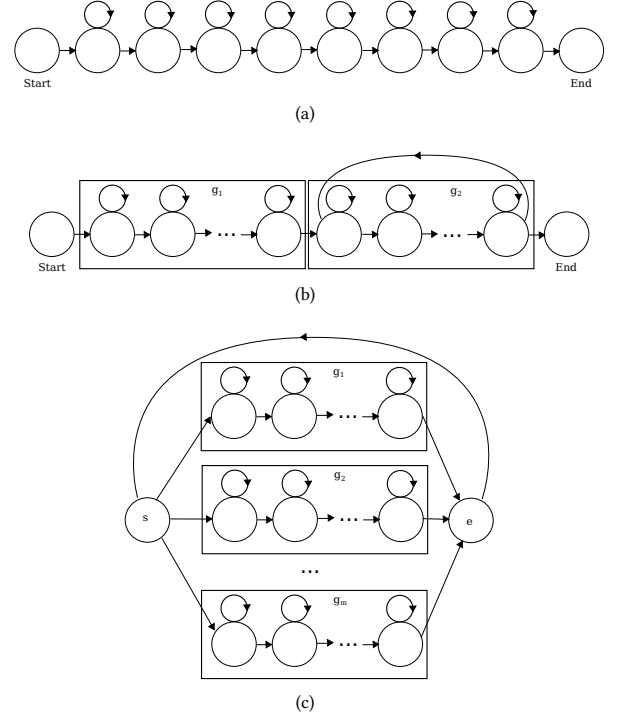
A Hidden Markov Model (HMM) is a probabilistic model that represents probability distributions over sequences of observations, and is especially useful for time-series data. It assumes that the observations, which in our case are the hand trajectories, are dependent on a hidden process, which is assumed to be Markov. Specifically, we leverage a Multivariable Gaussian HMM as this model allows joining recognizers that infer time-series data composed of known sub-components. Our choice is motivated by the fundamental goal of Gesture Knitter, which is to synthesize novel hand gestures by arranging primitives by concatenation or adding loops. Many other parametric probabilistic methods have been applied to related problems [2, 3, 10, 11, 18, 44] but are not well suited to our task.

The structure for the HMMs we use for primitive gestures is a Multivariable Gaussian HMM with loop transitions for each of the hidden states. We found that 12 hidden states works best for gross primitives and eight hidden states works best for fine primitives. The fine primitive HMM is shown in Figure 7(a). We train the HMMs given the samples for the primitive trajectories using the Baum-Welch algorithm, where estimates of the transition probabilities are iteratively improved during learning. The HMMs are trained for the gross and fine primitives only using the samples from demonstrations or synthetically generated samples (covered later in Section 4.5).

## 4.3 Recognizer Synthesis and Classification

We focus on the recognizer synthesis and classification process for one-hand complex gestures. When the declarative script for the complex gesture is compiled, we obtain a list of aligned sequential components for the gross primitives, fine primitives, and loops that correspond to the complete gesture nodes. For each one-hand complex gesture, we build two HMMs corresponding to the gross component, a concatenation of gross primitives, and the fine component, a concatenation of the fine primitives, separately. Here we describe the process for the fine component as the gross component is constructed in the same way. For each fine gesture primitive, we retrieve the pre-trained HMM corresponding to that primitive. For one fine primitive component that sequentially follows another fine primitive component, we concatenate the two corresponding pre-trained HMMs by adding a transition between the end state of the first fine primitive's HMM to the start state of the second fine primitive's HMM as shown in Figure 7(b). If the corresponding loop value to that fine primitive is greater than one, we add a transition between the end state of that particular primitive's HMM to its start state. For two-handed complex gestures, we have four HMMs corresponding to the right-gross, right-fine, left-gross, and left-fine components.

Now, given the different complex hand gesture classes  $\{c_i\}_{i=1}^m$  and their corresponding HMM recognizers, the gross and fine components, we can perform classification. For a given demonstration  $\mathcal{D}$ , we run the forward algorithm on each of the complex gesture HMMs corresponding to each of the complex gesture classes in order to find  $\log p(\mathcal{D}|c_i)$  for  $i = 1, \dots, m$ . Essentially, the forward algorithm for each complex hand gesture class  $c_i$ , is applied to  $\mathcal{D}$  on the gross and fine HMMs of the complex gestures to yield  $\log p_{g,c_i}$  and  $\log p_{f,c_i}$  corresponding to the log probabilities for the



**Figure 7: (a) An eight-state HMM for a fine primitive. (b) Fine component of complex gesture, consisting of gesture  $g_1$  followed by gesture  $g_2$  looped twice. Here, the corresponding HMM of  $g_2$  is concatenated to  $g_1$  and looped. (c) Fine decoding HMM by parallel HMMs corresponding to fine primitives  $g_1, \dots, g_m$  and a transition from the end  $e$  to start  $s$ .**

class's gross and fine components respectively. Then we have that  $\log p(\mathcal{D}|c_i) = \log p_{g,c_i} + w \log p_{f,c_i}$  for the mixture model.  $w$  is an empirically determined parameter. We found that  $w = 0.1$  works the best in our experiments. Therefore, we classify the gesture into the class  $C$  where  $C = \arg\max_{c_i} \log p(\mathcal{D}|c_i)$ , that is, the complex gesture class with the highest log probability of generating the data sequence. For two-hand gestures, the recognizer would have the log probability of the demonstration given the class as a sum of four terms: the log probabilities of the gross and fine components for both the right and left hands. We use the same weighting for the mixture model between gross and fine components for two-handed gestures.

## 4.4 Decoding Demonstrations

The main algorithm that we use to translate a demonstration  $\mathcal{D}$  to a declarative script is the Viterbi algorithm, which computes the most likely sequence of hidden states given the demonstration. Again, we focus on one-hand gestures as two-hand gestures can be easily extended by applying the same method for the other hand. First, we describe the decoding HMMs corresponding to the gross and fine components on which we run the Viterbi algorithm. Consider the fine component of the decoding HMM, and call its start and end states  $s$  and  $e$  respectively. For every fine primitive,



we connect its corresponding primitive HMM to the fine decoding HMM by connecting  $s$  to the starting state of the fine primitive HMM and the end state of that fine primitive HMM to  $e$ , and subsequently normalizing the transition probabilities. Then, to simulate the sequence of different fine gestures, we add a transition from  $e$  to  $s$ , essentially looping the decoding HMM. The whole decoding HMM is shown in Figure 7(c). Hence, after running Viterbi with  $\mathcal{D}$ , we obtain a sequence of hidden states that correspond to the sequence of fine primitive gestures. We build a similar decoding HMM for gross primitives. For two-hand gestures, we construct the two decoding HMMs for each of the left and right hands. Given the hidden states, we normalize the series with the smallest length of the number of states traversed for one particular gesture primitive, and subsequently align the fine and gross primitive states for the script nodes.

#### 4.5 Preference Gaussian Processes and Sample Generation

We apply Bayesian optimization to our preference gallery tool to generate synthetic samples from which the user can select. Bayesian optimization is a frequently employed algorithm in interactive machine learning for environments such as procedural animation design [1], visual design optimization [17], and panel placement in a crowd-sourced AR setting [6]. Most of these methods optimize over a set of design parameters in order to generate new samples to enhance a concrete design goal, such as visibility and material appearance. Moreover, previous works have primarily used interactive machine learning in an AR/VR setting to recognize hand gestures for an online supervised classification task [16, 32]. Attempts have been made to create a generative model for hand gestures, such as using generative adversarial networks, but that approach requires large amounts of data [43]. Using template models to infer variation parameters of the gestures [4] is another approach. However, such methods can only be effectively applied to lower dimensional data. Specifically, the fine gestures are 15 dimensional for one hand, and due to the curse of dimensionality, this makes distance metric approaches for comparing gestures ineffective. Instead, we use Bayesian optimization to optimize the variation we can apply to previously collected demonstrations to increase the recognition accuracy of the classifier. Our novel approach allows designers to automatically generate new samples without the repetitive labor of providing explicit demonstrations.

In our particular scenario, the objective function value is hidden because when training, we are only given data about the relative preference of one generated sample to another sample in representing the original sample of the primitive gesture. To solve this, we follow the framework as outlined in Brochu et al. [1], which we refer the reader to for more details on the approach. The essential idea is that from the preference gallery user choice, we use Gaussian process (GP) regression to model the latent valuation function given the parameters used to generate the synthetic demonstrations and their preference rankings. From the GPs, we use the Bayesian optimization approach to decide where to sample next while trading off exploitation and exploration with the expected improvement acquisition function.

We implement two Bayesian optimizers to generate synthetic samples for the fine primitives and gross primitives separately. For the fine primitives, the parameters are the noise variances  $\{\sigma_i\}_{i=1}^3$  from which we sample  $\{\theta_i\}_{i=1}^3$ , the roll, yaw, and pitch deviation, from the normal distribution  $\theta_i \sim \mathcal{N}(0, \sigma_i^2)$ . From  $\{\theta_i\}_{i=1}^3$ , we construct the rotation matrix with which we multiply the coordinates of each of the fingers relative to the palm position. For the gross primitives, the parameters are the noise variances  $\{\omega_i\}_{i=1}^3$  from which we sample the dilation factors  $\{\lambda_i\}_{i=1}^3$  for the  $x, y, z$  directions from the normal distribution  $\lambda_i \sim \mathcal{N}(1.0, \omega_i^2)$ . From  $\{\lambda_i\}_{i=1}^3$ , we construct the 3×3 diagonal matrix with which we dilate the palm positions along the entire trajectory relative to the starting position of the palm. Note that for the same parameter set  $\theta_i$  or  $\lambda_i$ , we get a different synthetic trajectory each time we are sampling from the normal distribution set by these parameters. This approach serves to introduce variation in terms of the shape, scale, and speed of the gross primitives and variability in the orientation of the fine primitive motions. We add 30 training samples to the two Bayesian optimizers before we use the preference galleries to generate synthetic data.

In our approach, we add a heuristic to randomize our decision each iteration to either choose to sample two new parameter sets to obtain a new sample or to pick two of the previously sampled parameters to generate trajectories from. This is to query the user about the ranks between two previous parameter sets that have not yet been directly compared for a better estimate of the latent valuation function.

#### 4.6 Discernment of Gestures

We run the demonstrations of the newly declared gesture on all the other previously declared gestures' HMMs in the same type of general class, that is, gross primitives, fine primitives, one-handed complex gestures. These will provide a log probability for each HMM that demonstrates the likelihood that any of the other classes is likely to be similar to the new gesture. If any of these log probabilities is above an empirically determined threshold then we give a warning to the designer that, given these demonstrations, there is a possibility that the new gesture is similar to a gesture that is already declared.

#### 4.7 Online Recognition

Gesture Knitter supports online recognition applications where the system is required to detect gesture activity within a stream of possible non-gesture data. In order to recognize multiple gestures sequentially, we construct a movement HMM representing the gesture class trained from the gross motion of the discrete complex gesture samples we have obtained. Note that we detect movement only through the palm trajectory or the gross motion. Hence, for a continuous stream of frames, we first apply the preprocessing step of detecting if a gesture lies within a window of 150 frames, stepping by 30 frames each iteration. For the window, we run the trajectory on the movement HMM, and if the probability of movement is greater than a log probability threshold, we run the window data on the recognizer, indicating what is the most likely gesture to be in this window. We accept the classification if the log probability of the class selected is greater than -30,000.0 for one-handed complex

gestures and -5,000.0 for two-handed complex gestures. For both types of complex gestures, we set the movement threshold to -50.0. All of the above parameters were selected based on those yielding the best performance given collected user data. We accumulate all the recognition results for the most recent set of windows where we expect there to be a gesture and make a decision based on a majority vote.

#### 4.8 Implementation Details

The visual declarative script is implemented using Java. The HMM recognition system is implemented using Pomegranate [35] in Python3, and the Bayesian optimization and hand animations are implemented with Numpy and Matplotlib in Python3 respectively. The hand tracking component is implemented in C# and Unity using the Microsoft HoloLens 2 headset.

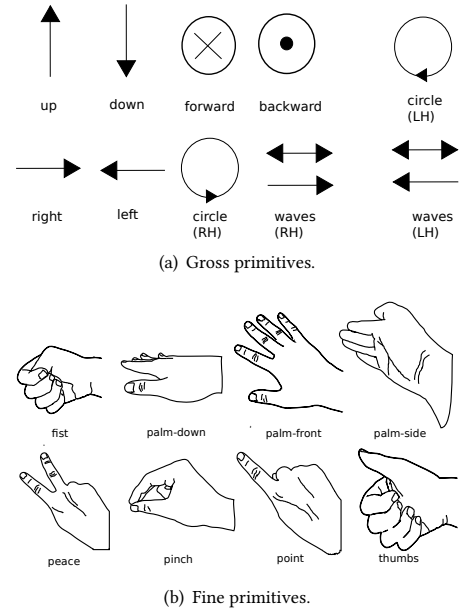
### 5 EXPERIMENTAL EVALUATION

We evaluate Gesture Knitter in two separate experiments to gain insight into the design process and the recognition capabilities of the system. The purpose of the first study, conducted with eight individuals (all right handed, three female), was to collect representative gesture data to evaluate recognition and decoder performance. In Study 1, performance is evaluated with reference to several tasks: 1) recognition rate of the complex gestures; 2) decoding performance when declaring complex gestures; 3) the aid of synthetic samples in increasing recognition rate; and 4) the potential of recognition performance in an online recognition setting. The purpose of the second study, conducted with five of the same participants from Study 1 (one female), was to observe the participants' behaviors and use of Gesture Knitter during a design task of a custom complex gesture with a newly declared primitive. The first study took two hours and the design study took one hour including breaks, and all eight were compensated £40 for their time.

#### 5.1 Study 1: Data Collection

The data collection for evaluating Gesture Knitter involved collecting repeated gesture samples for a defined set of gross, fine and complex combined gestures. For gross primitives, we collected five samples from each participant for each of the eight gross primitive classes shown in Figure 8(a) for the right hand. We logged two more gross primitive classes for the left hand corresponding to circles and waves, five samples per gesture from each participant. Note that we used the data from other six right hand gross primitives for the left hand as well. For fine primitives, we collected five samples from each participant for each of the eight fine primitive classes shown in Figure 8(b) for both the right and left hands due to motor differences between the dominant and non-dominant hands. Five further samples from each participant were collected for each of the ten complex one-handed gestures in Table 2 and for each of the ten complex two-handed gestures in Table 3. When executing all gestures, participants were required to start from a nominal rest position: the hand with the palm facing down. The gross and fine primitives were selected by following the gestures suggested by Piumsomboon et al. [30]. The complex gestures were selected to showcase a variety of distinct complex gestures that can be synthesized using Gesture Knitter's visual declarative script.

To demonstrate the potential of applying Gesture Knitter to online recognition, we also collected five samples of three complex gestures performed in sequence within a 20 second reel. Each individual was asked to perform one of each of the five sequences of three gestures: (1) *Shrink, Execution, PeaceOut*; (2) *TickTock, MadRiddles, Flamingo*; (3) *Grow, Caterpillar, CheshireDance*; (4) *Push, CheshireDance, PeaceOut*; (5) *Execution, Shrink, Flamingo* (see Table 2 for gesture syntax). We did this for both one-handed complex gestures and two-handed complex gestures. Participants were requested to keep their hands relatively still between two complex gestures for five seconds during the continuous reel.



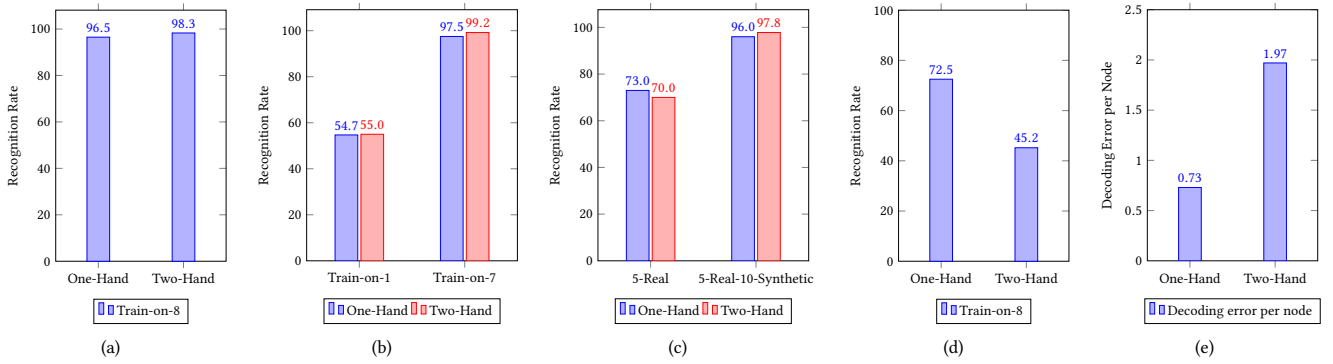
**Figure 8: (a) All gross primitives. Note that for the right hand circle, the direction is counterclockwise while for the left hand circle, the direction is clockwise. For the right hand waves, the gesture starts by going right while for the left hand waves, it starts by going left. (b) All fine primitives for the right hand, which are mirrored for the left hand.**

#### 5.2 Recognition and Decoding Evaluation

We first tested the rate of recognition of all the eight participants' one-hand and two-hand complex gestures when trained with all the primitives collected (that is, 8 participants  $\times$  5 repetitions = 40 training examples per primitive gesture). Note that none of the recognizers are trained with any of the complex gesture samples.

The recognition accuracy for the ten one-hand and two-hand gestures is shown in Figure 9(a). Gesture Knitter achieves an average accuracy of 96.5% for one-handed complex gestures and 98.3% for two-handed complex gestures. To examine the impact of training data diversity, we also conducted two cross-validation experiments with the full dataset: *Train-on-1* involved training on primitive data from one person and testing on the complex gestures from the





**Figure 9:** (a) The overall recognition rate on all complex gesture data when all the participants' primitive data is used for training. (b) The cross-validation recognition rate when trained with one participant's primitive data versus if trained with seven participants' primitive data. This shows that with more variety in user training data, the recognizers are more robust against new users' gestures. (c) The recognition rate when trained with five samples randomly selected for each primitive versus training with those five and ten synthetically generated from those five. (d) The online recognition rate when the primitives are trained with data from all eight users. (e) The number of errors per node for decoding complex gestures. Note that one-handed complex gestures have three fields per node, while two-handed have five fields per node.

**Table 2:** The ten one-handed complex gestures collected. Gesture nodes are listed in sequential order with each being denoted as *gross-primitive* / *fine-primitive* / (*repeats*), where *repeats* are blank if 1.

Gesture	Declaration
Shrink	1. down / palm-side
Push	1. forward / palm-front
TickTick	1. waves / palm-front
PeaceOut	1. left / peace
Execution	1. up / fist
Caterpillar	1. right / pinch
Flamingo	1. up / pinch 2. left / peace
CheshireDance	1. backward / thumbs 2. circle / point / 3
Grow	1. circle / fist / 3 2. forward / palm-front
MadRiddles	1. forward / point 2. right / palm-down 3. backward / thumbs

other seven participants; and *Train-on-7* instead trains on primitive data from seven participants and tests on the complex gesture from the other single participant. These results are shown in Figure 9(b). For the *Train-on-1* condition, the recognition rate is 54.7% and 55.0% for one-handed and two-handed complex gestures respectively. The cross-validation accuracy for *Train-on-7* is 97.5% for one-handed and 99.2% for two-handed complex gestures, illustrating that increased variation in the primitive data introduced by more individuals creates increased robustness in recognition of new users' gestures. In addition, the average time for classifying one-handed gestures was 470.2 ms and for two-handed gestures was 1095.6 ms on a Dell Latitude E5440 PC with an Intel Core i5.

**Table 3:** List of the 10 two-handed complex gestures collected. The gesture nodes are listed in sequential order with each node being denoted as *right-gross-primitive* / *right-fine-primitive* / *left-gross-primitive* / *left-fine-primitive* / (*repeats*), where *repeats* are blank if it is 1.

Gesture	Visual Script Declaration
Shrink	1. down / palm-side / down / palm-side
Push	1. forward / palm-front / backward / palm-front 2. backward / palm-front / forward / palm-front
TickTick	1. waves / palm-front / waves / palm-front
PeaceOut	1. left / peace / left / peace
Execution	1. up / fist / up / fist
Caterpillar	1. right / pinch / left / pinch
Flamingo	1. up / pinch / up / pinch 2. right / peace / left / peace
CheshireDance	1. backward / thumbs / backward / thumbs 2. circle / point / circle / point / 3
Grow	1. circle / fist / circle / fist / 3 2. forward / palm-front / forward / palm-front
MadRiddles	1. forward / point / forward / point 2. right / palm-down / left / palm-down 3. backward / thumbs / backward / thumbs

We then assessed the efficacy of generating synthetic samples from our preference gallery-based Bayesian optimization. Note that synthetic samples were generated by the first author using the preference gallery. We first selected five examples for each gross and fine primitive from a randomly selected participant as training data, and then we synthetically generate ten new samples for each primitive from these five samples using our preference gallery. Figure 9(c) compares the recognition rate on one-handed and two-handed complex gestures from all participants when trained with five real samples to five real plus 10 synthetic samples for each fine

and gross primitive. For both types of complex gestures, we observe a significant increase in recognition rate to a comparable rate to training with all the primitive samples when trained with the 10 synthetic samples, specifically from 73.0% to 96.0% for one-handed complex gestures and from 70.0% to 97.8% for two-handed complex gestures. This suggests that the generated synthetic gestures introduce sufficient variability into the primitives that translates to substantial improvements in recognition accuracy for different users while simultaneously alleviating data overhead.

To gauge the potential of these recognizers for applications in a more challenging online recognition use-case, we evaluated online recognition performance using the 20 second trace in which participants sequentially performed three complex gestures. This dataset consists of a total of 120 one-handed and 120 two-handed complex gestures performed sequentially by the eight participants. The results are shown in Figure 9(d). The recognition rate is calculated as follows: an incorrectly recognized gesture, or no output when there is a gesture, results in an error; if a gesture is recognized when there was none we introduce an error and add one to the total number of gestures. The recognition rate for one-handed was 72.5% and for two-handed was 45.2%. The lower accuracy, notably for two hands, is due to firstly the high frequency of false activations where a hand gesture class was detected when there was none, and secondly from when there were no gestures detected when there indeed was one. These points suggest that better online gesture recognition delimitation techniques are likely to further improve the recognition rate, but this is beyond the scope of the current paper. In many prototyping development exercises, we anticipate that there will in fact be some form of explicit gesture start delimitation in which case the effective recognition accuracy is as per Figure 9(a) (i.e., in excess of 95%).

Finally, we assess our decoding mechanism as an aid for designers from our entire collection of one-handed and two-handed complex gestures. We assess the efficacy of decoding by using a variant of the edit distance. We define the edit distance between two scripts as the number of edits on the nodes' fields for them to be the same. Note that for one-handed complex gestures, the number of fields per node is three (fine, gross, and number of loops) and for two-handed complex gestures, the number of fields per node is five (right-fine, right-gross, left-fine, left-gross, and number of loops). We then divide the total edit distance by the number of nodes of that particular complex gestures' script declaration to obtain the decoding error rate per node. Intuitively, this value corresponds to the expected number of fields requiring a change for each node to form the right declaration after giving a demonstration. For one-handed gestures, the average error rate was 0.73 edits per node while for two-handed gestures, the average error rate was 1.97 edits per node as shown in Figure 9(e). Gestures such as *TickTock* may generate large errors when the gross primitives decoded are repetitions of *right* and *left*, which can be thought of as an alternative formulation of the gesture. Despite the decoder sometimes omitting or adding extraneous nodes, this performance demonstrates that the decoding mechanism is a helpful tool for designers to edit the visual declarative script generated from providing a demonstration instead of starting from scratch.

### 5.3 Study 2: Design Study

We conducted a design study with five designers recruited from within our institution (four male and one female, average age 21.6 years) to analyze the usability of Gesture Knitter's features and design protocol. All five participants had previously taken part in Study 1. This was to ensure participants were already familiar with the HoloLens 2 hand tracking functionality. Three participants had prior experience using machine learning, one had programmed gesture recognition before, and one had experience with AR/VR applications.

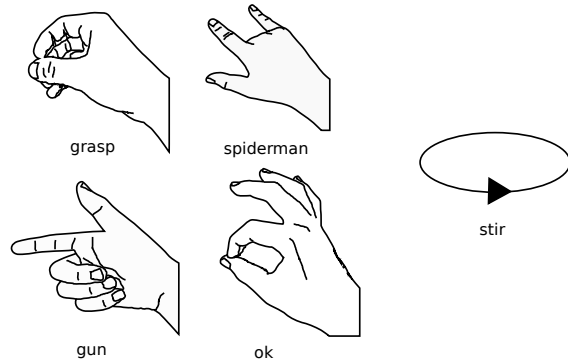
We asked each participant to declare one new fine or gross primitive different from the sixteen evaluated in Study 1 and to use that new primitive to design a custom one-handed complex gesture. They were free to use any of the predeclared fine and gross primitives. Prior to the design task, participants were introduced to the idea of decomposing a hand gesture into gross and fine components and synthesizing complex gestures. Participants were familiar with the distinction between fine, gross, and complex gestures, having already been exposed to this concept in Study 1. Study 2 was scheduled to ensure that there was no more than a one day break after Study 1. Participants were asked to prepare and notify the study organizers of their new primitive and complex gesture concept prior to attending the study in order to avoid the same new primitive being chosen by multiple individuals. This happened when two individuals wanted to declare the same fine primitive *Spiderman*.

We first provided the participants with a tutorial on Gesture Knitter and walked them through the process of using the visual declarative script to declare the *Shrink* and *Flamingo* gestures from scratch and by demonstration. Specifically, we stated to them that they must input their novel primitive gesture first by providing 10 demonstrations and then declaring their novel complex gesture. Next, participants completed a task to declare a declarative script for *Grow* and *Push*.

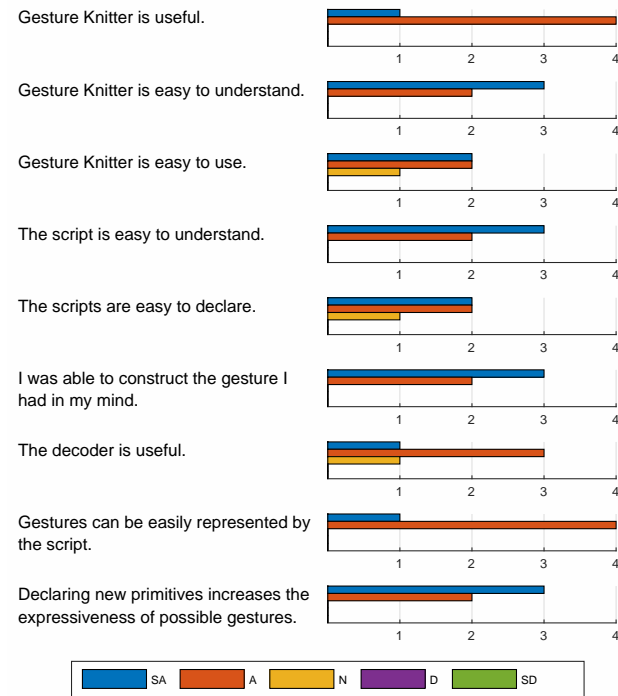
We then asked participants to start declaring their novel primitive and designing their novel complex gesture. During the design process, we followed a think-aloud protocol in which participants were asked to verbalize their reasoning and decision making at each step of implementing the novel complex gesture. Participants were instructed to provide ten demonstrations of the novel primitive gesture as part of the design process, as well as ten demonstrations of their novel complex gesture. If they decided to declare their gesture from scratch, we asked them to try out the decoding mechanism afterwards. After the design process, participants completed a post-study questionnaire. The study was conducted on a laptop with an external mouse and all hand gestures were recorded using the HoloLens 2.

#### 5.3.1 Results.

All five participants completed the design study by declaring either a gross or fine primitive and using that to construct a novel one-handed complex gesture. Four novel fine primitives and one novel gross primitive were declared, as shown in Figure 10. The five novel complex gestures are presented in Figure 4. During the initial synthesis of the complex gesture, four participants opted to declare the script from scratch and one participant decided to provide a demonstration (for the *Dispose* gesture).



**Figure 10: The five novel primitives that the designers came up with. Stir is a gross primitive which is one counterclockwise circle in the horizontal plane.**



**Figure 11: The five-point Likert responses to the post-design questionnaire. Here, SA is strongly agree, A is agree, N is neutral, D is disagree, and SD is strongly disagree.**

The post-design questionnaire captured participants' responses to the statements summarized in Figure 11 on a five point Likert scale. Notably, all participants agreed that Gesture Knitter was useful, easy to understand, and expressive.

The following observations are taken from the think-aloud component of the study. When designers opted to declare the script from scratch, they inserted the gesture nodes in chronological order. This suggests that the visual declarative script corresponds directly to the performance of the complex gesture. Typically, they first

**Table 4: List of the five one-handed complex gestures designed by designers during the design study. The gesture nodes are listed in sequential order with each being denoted as *gross-primitive* / *fine-primitive* / (*repeats*), where *repeats* are blank if it is 1.**

Complex Gesture	Visual Script Declaration
Spiderweb	1. forward / spiderman 2. backward / palm-front
Alright	1. circle / ok / 2 2. up / ok
Dispose	1. left / grasp 2. forward / pinch
Cauldron	1. stir / fist / 3
Shotgun	1. right / gun 2. left / fist

declare all the gesture nodes, including the start and end nodes, and then connect the nodes together with the edges. During the declaration, the designers would also perform the complex gesture to guide themselves through the design process. This behavior implies that the physical declaration of the visual declarative script from scratch intuitively maps to performing the complex gesture. In addition, when performing the decoding by demonstration, the designer would still need to traverse from start to end to verify the decoding and make any of possibly a few modifications accordingly. This observed workflow is likely to have been influenced by the nature of the task and the introductory instructions given. We conjecture that increased exposure and trust in the system would lead to participants leveraging the declaration by demonstration functionality more extensively given its efficiency. It is also plausible that the decoding by demonstration functionality may be useful to users with no prior exposure to the system, as it may provide a helpful introduction on how to construct and edit nodes to represent gestures a novice designer would have in mind.

The visual declarative script therefore appears to directly and intuitively correspond to a syntactic representation that represents the physical performance of the gesture. The fact that most participants had no prior experience in programming gesture recognition, and were nevertheless able to build a custom gesture recognizer, demonstrates that Gesture Knitter is easy to use. The recognition rates for the novel complex gestures vary between different classes. For *Alright*, *Shotgun*, and *Cauldron*, all of the 10 designer provided samples for each class were correctly classified. By contrast, *Dispose* had a recognition rate of 10% and none of the designer samples of *Spiderweb* were correctly classified. *Dispose* was most frequently confused with *Caterpillar* while *Spiderweb* was confused with *Push*. The discernability tool identifies these potential misclassifications, highlighting its merit as a design aid to create distinct gestures.

Finally participants were given an opportunity to provide comments on the features of Gesture Knitter. When asked about what they liked best about the system, two users mentioned Gesture Knitter's decoding mechanism. One commented that "*decoding is significantly less mentally strenuous than connecting blocks together.*"

The three other designers all mentioned the simple and straightforward design interface. One stated that the “*visual declarative script is user-friendly and one can see how complex movements can be broken down node by node.*” These comments suggest that Gesture Knitter is easy to learn and supports rapid development of novel complex hand gestures.

In terms of possible features that the designers wanted to see implemented in the future, there were suggestions on improving the script to make the script even more powerful. One wanted the system to “*learn to recognize the borders between successive gestures*”, hinting that the transitions between gesture components can also be user-defined. Another wanted the incorporation of the speed of the gesture as a semantic feature in determining the gesture class. This aligns well with our vision for future work.

## 6 DISCUSSION AND FUTURE WORK

Although Gesture Knitter allows users to expressively create hand gestures with low data overhead, several enhancements can be added to improve the system’s capabilities. The hand gestures performed in this paper have a specific way in which they are performed, such as the circle primitive being counterclockwise for the right hand. Although HMMs are able to recognize different demonstrations of the same class given sufficient sample variation and number, the visual declarative script would need to support multiple ways of performing a gesture. One possible route for future work would be allowing the designer to provide multiple script candidates for a particular complex gesture.

We have illustrated the feasibility of Gesture Knitter as a tool that can be used to create novel gestures for head-mounted mixed reality applications. Crucially, it demands very little data to be collected by the designer. While this paper has explored one particular online continuous recognition setting, further investigations can be conducted to improve online recognition as well as recognition depending on user context. More delimitation and windowing techniques can be explored with respect to Gesture Knitter to further optimize the system for online recognition. Another fruitful avenue of future work is to explore user interaction with these declared gestures for various applications in mixed reality. For example, one compelling direction would be to probe user experiences when using Gesture Knitter for a gaming application.

With regards to synthetic data generation, this process could be conceivably extended to generate a sufficiently large sample of gestures to train a deep neural network architecture, such as a recurrent neural network or a generative adversarial network. However, this averts the primary goal of having low data overhead. One natural direction to explore would be to find other low dimensional generative processes to generate effective synthetic hand gesture samples.

## 7 CONCLUSIONS

We have introduced Gesture Knitter, a tool for creating hand gestures for head-mounted mixed reality applications. It augments the ability for designers to create new gestures from primitives through visual declarative scripts with low data overhead, while allowing for creativity and expressiveness during the design process. Gesture Knitter features additional tools, such as decoding, synthetic

sample generation, and discernment which serve as useful aids to the designer. In addition, Gesture Knitter supports both one-handed and two-handed gestures. Moreover, we have demonstrated the recognition capabilities of Gesture Knitter by assessing gesture variability across different users and different levels of data sparsity, and the prospect of deploying the generated recognizers to online recognition settings. In summary, Gesture Knitter is shown to be a compelling tool for designers to rapidly prototype custom gesture recognizers for a wide variety of mixed reality applications.

## ACKNOWLEDGMENTS

This work was supported by EPSRC (grants EP/R004471/1 and EP/S027432/1). G. B. Mo was supported by a Trinity College Summer Studentship fund. Supporting data for this publication is available at <https://doi.org/10.17863/CAM.62935>.

## REFERENCES

- [1] Eric Brochu, Tyson Brochu, and Nando De Freitas. 2010. A Bayesian Interactive Optimization Approach to Procedural Animation Design. *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation* (2010), 10 pages. <https://doi.org/10.2312/SCA/SCA10/103-112> Artwork Size: 10 pages ISBN: 9783905674279 Publisher: The Eurographics Association.
- [2] Sylvain Calinon. 2016. A tutorial on task-parameterized movement learning and retrieval. *Intelligent Service Robotics* 9, 1 (Jan. 2016), 1–29. <https://doi.org/10.1007/s11370-015-0187-9>
- [3] Sylvain Calinon, Florent Guenter, and Aude Billard. 2007. On Learning, Representing, and Generalizing a Task in a Humanoid Robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 37, 2 (April 2007), 286–298. <https://doi.org/10.1109/TSMCB.2006.886952> Conference Name: IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics).
- [4] Baptiste Caramiaux, Nicola Montecchio, Atsu Tanaka, and Frédéric Bevilacqua. 2014. Adaptive Gesture Recognition with Variation Estimation for Interactive Systems. *ACM Transactions on Interactive Intelligent Systems* 4, 4 (Dec. 2014), 18:1–18:34. <https://doi.org/10.1145/2643204>
- [5] Qing Chen, Nicolas D. Georganas, and Emil M. Petriu. 2008. Hand Gesture Recognition Using Haar-Like Features and a Stochastic Context-Free Grammar. *IEEE Transactions on Instrumentation and Measurement* 57, 8 (Aug. 2008), 1562–1571. <https://doi.org/10.1109/TIM.2008.922070> Conference Name: IEEE Transactions on Instrumentation and Measurement.
- [6] John J. Dudley, Jason T. Jacques, and Per Ola Kristensson. 2019. Crowdsourcing Interface Feature Design with Bayesian Optimization. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. Association for Computing Machinery, Glasgow, UK, 1–12. <https://doi.org/10.1145/3290605.3300482>
- [7] John J. Dudley, Hendrik Schuff, and Per Ola Kristensson. 2018. Bare-Handed 3D Drawing in Augmented Reality. In *Proceedings of the 2018 Designing Interactive Systems Conference (DIS '18)*. Association for Computing Machinery, New York, NY, USA, 241–252. <https://doi.org/10.1145/3196709.3196737>
- [8] Stefan Eickeler, Andreas Kosmala, and Gerhard Rigoll. 1998. Hidden Markov model based continuous online gesture recognition. In *Proceedings. Fourteenth International Conference on Pattern Recognition (Cat. No.98EX170)*, Vol. 2. 1206–1208 vol.2. <https://doi.org/10.1109/ICPR.1998.711914> ISSN: 1051-4651.
- [9] Eng-Jon Ong and R. Bowden. 2004. A boosted classifier tree for hand shape detection. In *Sixth IEEE International Conference on Automatic Face and Gesture Recognition, 2004. Proceedings. IEEE*, Seoul, Korea, 889–894. <https://doi.org/10.1109/AFGR.2004.1301646>
- [10] Shai Fine, Yoram Singer, and Naftali Tishby. 1998. The Hierarchical Hidden Markov Model: Analysis and Applications. *Machine Learning* 32, 1 (July 1998), 41–62. <https://doi.org/10.1023/A:1007469218079>
- [11] Jules Françoise and Frédéric Bevilacqua. 2018. Motion-Sound Mapping through Interaction: An Approach to User-Centered Design of Auditory Feedback Using Machine Learning. *ACM Transactions on Interactive Intelligent Systems* 8, 2 (June 2018), 16:1–16:30. <https://doi.org/10.1145/3211826>
- [12] William T. Freeman and Michal Roth. 1994. Orientation Histograms for Hand Gesture Recognition. In *International Workshop on Automatic Face and Gesture Recognition*. Zurich, Switzerland, 296–301.
- [13] Tracy Hammond and Randall Davis. 2005. LADDER, a sketching language for user interface developers. *Computers & Graphics* 29, 4 (Aug. 2005), 518–532. <https://doi.org/10.1016/j.cag.2005.05.005>
- [14] Sanna Kallio, Juha Kela, and Jani Mantyjarvi. 2003. Online gesture recognition system for mobile interaction. In *SMC'03 Conference Proceedings. 2003 IEEE*

- International Conference on Systems, Man and Cybernetics. Conference Theme - System Security and Assurance (Cat. No.03CH37483)*, Vol. 3. 2070–2076 vol.3. <https://doi.org/10.1109/ICSMC.2003.1244189> ISSN: 1062-922X.
- [15] Jinsheng Kang, Kang Zhong, Shengfeng Qin, Hongan Wang, and David Wright. 2013. Instant 3D design concept generation and visualization by real-time hand gesture recognition. *Computers in Industry* 64, 7 (Sept. 2013), 785–797. <https://doi.org/10.1016/j.compind.2013.04.012>
  - [16] Simon Katan, Mick Grierson, and Rebecca Fiebrink. 2015. Using Interactive Machine Learning to Support Interface Development Through Workshops with Disabled People. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems - CHI '15*. ACM Press, Seoul, Republic of Korea, 251–254. <https://doi.org/10.1145/2702123.2702474>
  - [17] Yuki Koyama, Issei Sato, Daisuke Sakamoto, and Takeo Igarashi. 2017. Sequential line search for efficient visual design optimization by crowds. *ACM Transactions on Graphics* 36, 4 (July 2017), 1–11. <https://doi.org/10.1145/3072959.3073598>
  - [18] Dana Kulić, Christian Ott, Dongheui Lee, Junichi Ishikawa, and Yoshihiko Nakamura. 2012. Incremental learning of full body motion primitives and their sequencing through human motion observation. *The International Journal of Robotics Research* 31, 3 (March 2012), 330–345. <https://doi.org/10.1177/0278364911426178> Publisher: SAGE Publications Ltd STM.
  - [19] Hyeon-Kyu Lee and J.H. Kim. 1999. An HMM-based threshold model approach for gesture recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 21, 10 (Oct. 1999), 961–973. <https://doi.org/10.1109/34.799904> Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.
  - [20] Yi Li. 2012. Hand gesture recognition using Kinect. In *2012 IEEE International Conference on Computer Science and Automation Engineering*. 196–199. <https://doi.org/10.1109/ICSESS.2012.6269439> ISSN: 2327-0594.
  - [21] Zhihan Lv, Alaa Halawani, Shengzhong Feng, Shafiq ur Rehman, and Haibo Li. 2015. Touch-less interactive augmented reality game on vision-based wearable device. *Personal and Ubiquitous Computing* 19, 3-4 (July 2015), 551–567. <https://doi.org/10.1007/s00779-015-0844-1>
  - [22] Hao Lü, James A. Fogarty, and Yang Li. 2014. Gesture script: recognizing gestures and their structure using rendering scripts and interactively trained parts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. Association for Computing Machinery, Toronto, Ontario, Canada, 1685–1694. <https://doi.org/10.1145/2556288.2557263>
  - [23] Hao Lü and Yang Li. 2012. Gesture coder: a tool for programming multi-touch gestures by demonstration. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems - CHI '12*. ACM Press, Austin, Texas, USA, 2875. <https://doi.org/10.1145/2207676.2208693>
  - [24] Mounib Mazouzi, Paula Alavesa, Timo Koskela, and Timo Ojala. 2016. Ghost hunters: ambient light and hand gesture utilization in mobile augmented reality games. In *Proceedings of the 15th International Conference on Mobile and Ubiquitous Multimedia - MUM '16*. ACM Press, Rovaniemi, Finland, 365–367. <https://doi.org/10.1145/3012709.3017602>
  - [25] Byung-Woo Min, Ho-Sub Yoon, Jung Soh, Yun-Mo Yang, and Toshiaki Ejima. 1997. Hand gesture recognition using hidden Markov models. In *Computational Cybernetics and Simulation 1997 IEEE International Conference on Systems, Man, and Cybernetics*, Vol. 5. 4232–4235 vol.5. <https://doi.org/10.1109/ICSMC.1997.637364> ISSN: 1062-922X.
  - [26] Pavlo Molchanov, Shalini Gupta, Kihwan Kim, and Jan Kautz. 2015. Hand gesture recognition with 3D convolutional neural networks. In *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, Boston, MA, USA, 1–7. <https://doi.org/10.1109/CVPRW.2015.7301342>
  - [27] Miguel A. Nacenta, Yemliha Kamber, Yizhou Qiang, and Per Ola Kristensson. 2013. Memorability of Pre-Designed and User-Defined Gesture Sets. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. Association for Computing Machinery, New York, NY, USA, 1099–1108. <https://doi.org/10.1145/2470654.2466142>
  - [28] Pradyumna Narayana, J. Ross Beveridge, and Bruce A. Draper. 2018. Gesture Recognition: Focus on the Hands. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5235–5244. <https://doi.org/10.1109/CVPR.2018.00549> ISSN: 2575-7075.
  - [29] Lionel Pigou, Sander Dieleman, Pieter-Jan Kindermans, and Benjamin Schrauwen. 2015. Sign Language Recognition Using Convolutional Neural Networks. In *Computer Vision - ECCV 2014 Workshops*, Lourdes Agapito, Michael M. Bronstein, and Carsten Rother (Eds.). Vol. 8925. Springer International Publishing, Cham, 572–578. [https://doi.org/10.1007/978-3-319-16178-5\\_40](https://doi.org/10.1007/978-3-319-16178-5_40) Series Title: Lecture Notes in Computer Science.
  - [30] Thammathip Piumsomboon, Adrian Clark, Mark Billingham, and Andy Cockburn. 2013. User-defined gestures for augmented reality. *CHI '13 Extended Abstracts on Human Factors in Computing Systems* (2013), 955–960. <https://doi.org/10.1145/2468356.2468527>
  - [31] Rafael Radkowski and Christian Stritzke. 2012. Interactive hand gesture-based assembly for augmented reality applications. In *Proceedings of the 2012 International Conference on Advances in Computer-Human Interactions*. Citeseer, 303–308.
  - [32] Stefan Reifinger, Frank Wallhoff, Markus Ablassmeier, Tony Poitschke, and Gerhard Rigoll. 2007. Static and Dynamic Hand-Gesture Recognition for Augmented Reality Applications. In *Human-Computer Interaction. HCI Intelligent Multimodal Interaction Environments*, David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, and Julie A. Jacko (Eds.). Vol. 4552. Springer Berlin Heidelberg, Berlin, Heidelberg, 728–737. [https://doi.org/10.1007/978-3-540-73110-8\\_79](https://doi.org/10.1007/978-3-540-73110-8_79) Series Title: Lecture Notes in Computer Science.
  - [33] Zhou Ren, Junsong Yuan, Jingjing Meng, and Zhengyou Zhang. 2013. Robust Part-Based Hand Gesture Recognition Using Kinect Sensor. *IEEE Transactions on Multimedia* 15, 5 (Aug. 2013), 1110–1120. <https://doi.org/10.1109/TMM.2013.2246148>
  - [34] Eric Saund, David Fleet, Daniel Lerner, and James Mahoney. 2003. Perceptually-supported image editing of text and graphics. In *Proceedings of the 16th annual ACM symposium on User interface software and technology (UIST '03)*. Association for Computing Machinery, Vancouver, Canada, 183–192. <https://doi.org/10.1145/964696.964717>
  - [35] Jacob Schreiber. 2018. Pomegranate: Fast and Flexible Probabilistic Modeling in Python. *Journal of Machine Learning Research* 18, 164 (2018), 1–6.
  - [36] Suwon Shon, Jounghoon Beh, Cheoljong Yang, David K. Han, and Hanseok Ko. 2011. Motion primitives for designing flexible gesture set in Human-Robot Interface. In *2011 11th International Conference on Control, Automation and Systems*. 1501–1504. ISSN: 2093-7121.
  - [37] Helman I. Stern, Juan P. Wachs, and Yael Edan. 2004. Hand gesture vocabulary design: a multicriteria optimization. In *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583)*, Vol. 1. 19–23 vol.1. <https://doi.org/10.1109/ICSMC.2004.1398266> ISSN: 1062-922X.
  - [38] Thad Starner and Alex Pentland. 1995. Real-time American Sign Language recognition from video using hidden Markov models. In *Proceedings of International Symposium on Computer Vision - ISCV*. 265–270. <https://doi.org/10.1109/ISCV.1995.477012>
  - [39] Thad Starner, J. Weaver, and Alex Pentland. 1998. Real-time American sign language recognition using desk and wearable computer based video. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20, 12 (Dec. 1998), 1371–1375. <https://doi.org/10.1109/34.735811> Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.
  - [40] Helman I. Stern, Juan P. Wachs, and Yael Edan. 2006. Human Factors for Design of Hand Gesture Human - Machine Interaction. In *2006 IEEE International Conference on Systems, Man and Cybernetics*, Vol. 5. 4052–4056. <https://doi.org/10.1109/ICSMC.2006.384767> ISSN: 1062-922X.
  - [41] Helman I. Stern, Juan P. Wachs, and Yael Edan. 2006. Optimal Hand Gesture Vocabulary Design Using Psycho-Physiological and Technical Factors. In *7th International Conference on Automatic Face and Gesture Recognition (FGR06)*. 257–262. <https://doi.org/10.1109/FGR.2006.84>
  - [42] Helman I. Stern, Juan P. Wachs, and Yael Edan. 2008. Optimal Consensus Intuitive Hand Gesture Vocabulary Design. In *2008 IEEE International Conference on Semantic Computing*. 96–103. <https://doi.org/10.1109/ICSC.2008.29>
  - [43] Hao Tang, Wei Wang, Dan Xu, Yan Yan, and Nicu Sebe. 2019. GestureGAN for Hand Gesture-to-Gesture Translation in the Wild. *arXiv:1808.04859 [cs]* (July 2019). [arXiv: 1808.04859](https://arxiv.org/abs/1808.04859).
  - [44] Joëlle Tilmann, Alexis Moinet, and Thierry Dutoit. 2012. Stylistic gait synthesis based on hidden Markov models. *EURASIP Journal on Advances in Signal Processing* 2012, 1 (March 2012), 72. <https://doi.org/10.1186/1687-6180-2012-72>
  - [45] Youwen Wang, Cheng Yang, Xiaoyu Wu, Shengmiao Xu, and Hui Li. 2012. Kinect Based Dynamic Hand Gesture Recognition Algorithm Research. In *2012 4th International Conference on Intelligent Human-Machine Systems and Cybernetics*, Vol. 1. 274–279. <https://doi.org/10.1109/IHMSC.2012.76>
  - [46] Zahoor Zafrulla, Helene Brashear, Thad Starner, Harley Hamilton, and Peter Presti. 2011. American sign language recognition with the kinect. In *Proceedings of the 13th international conference on multimodal interfaces - ICMI '11*. ACM Press, Alicante, Spain, 279. <https://doi.org/10.1145/2070481.2070532>
  - [47] Chun Zhu and Weihua Sheng. 2009. Online hand gesture recognition using neural network based segmentation. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2415–2420. <https://doi.org/10.1109/IROS.2009.5354657> ISSN: 2153-0866.