Chronos: Improving Recognizers' Performance by Leveraging Gesture Continuity and Designers' Involvement

Leave Authors Anonymous
for Submission
City, Country
e-mail address

Leave Authors Anonymous
for Submission
City, Country
e-mail address

Leave Authors Anonymous
for Submission
City, Country
e-mail address

ABSTRACT

There is an increasing opportunity for using gestures to implement more intuitive and effective user interfaces. However, the low recognition accuracy in realistic scenarios limits the application scope of gesture interaction paradigm. In this paper, we presents *Chronos*[1], an algorithm framework that improves the performance of gesture recognizers by 1) extracting the continuity information from gesture sequences and, 2) enabling designers to optimize the decision-making rewards. The framework is implemented by integrating dynamic Bayesian network (DBN) with a partially observable Markov decision process (POMDP). Two studies are conducted on a drawing interface to evaluate the framework. Results show *Chronos* improves 3.3% or greater recognition accuracy of four gesture recognizers that commonly used in HCI, and reduces 17.7% or more consuming-time in three designated-drawing tasks and a free-drawing one.

ACM Classification Keywords

H.5.m. Information Interfaces and Presentation (e.g. HCI): Miscellaneous; See http://acm.org/about/class/1998/ for the full list of ACM classifiers. This section is required.

Author Keywords

Gesture recognition, Continuity of gesture, Dynamic Bayesian network, Partially observable Markov decision process

INTRODUCTION

Gestures-based interfaces are changing our daily uses of computing devices. With superior recognition technologies, gestures can provide natural and effective ways to express words, symbols, tables, formulas and drawings [18]. With the development of interfaces operated by pens, fingers, arms, or other instruments, there is an increasing opportunity for using gestures in a wider range of interaction scenarios [10, 21, 30]. The keys to making these systems have a better user experience, are solving four issues including visibility, guidance, memorability and recognition accuracy listed by Long [14]. To address these problems, many gesture recognition techniques based on statistical classification [3, 22], template matching [30] have been proposed trying to improve gesture recognition accuracy in different scenarios. Besides, tools [15] and studios [17] for gesture set design are also been built to facilitate the recognition.

Nevertheless, these methods are still flawed in two aspects. First, the existing recognition algorithms recognize users' gestures with only static similarity measured by geometrical or statistical comparison, but omit useful dynamic information involved in sequential logic. Second, the existing gesture design tools usually focus on enhancing designers' ability on editing gesture templates, the methods that allowing designers to actually manipulate the decision-making strategy of the system has been ignored.

To address these problems, we proposed an algorithm framework *Chronos* that improves the performance of gesture recognition by 1) leveraging the continuity information of gesture sequences and, 2) enabling designers to optimize the decision-making rewards. When a user perform a series of gestures, the order of the gestures and the potential system actions can provide valuable information for enhancing recognition performance. For instance, as shows in Figure 1, let's assume that a user want to execute paste operation by performing gesture "P", but a kernel recognizer (i.e. any recognizer that can provide confidences for each predefined gesture) misrecognizes it as gesture "N". The confidences for each gesture can be adjusted with a transition matrix that expresses how probable the gestures will be performed given the last input confidences. This adjustment enlarges the probability for gesture "P". Further, by searching greatest reward action in current interactive context with a reward function provided by designers, the probability for gesture "P" is raised again, and the final operation action paste is determined.

To implement *Chronos*, we use a dynamic Bayesian network (DBN) [5] integrated with a policy of partially observable Markov decision process (POMDP) [23] to estimate the most possible user intention from the current interaction context. We evaluated the improvements brought by the framework on four recognizers (i.e. DTW [9], Protractor [11] $1 [30], $Q [25], PolyRecGSS [6]) using a practical drawing tool. Results show that *Chronos* enhances the recognition accuracy of

[1] In ancient Greek mythology, one of the titans is called Chronos, which means time.

UIST'19, October 20–23, 2019, New Orleans, USA

ISBN 123-4567-24-567/08/06…$15.00

DOI: http://dx.doi.org/10.475/123_4

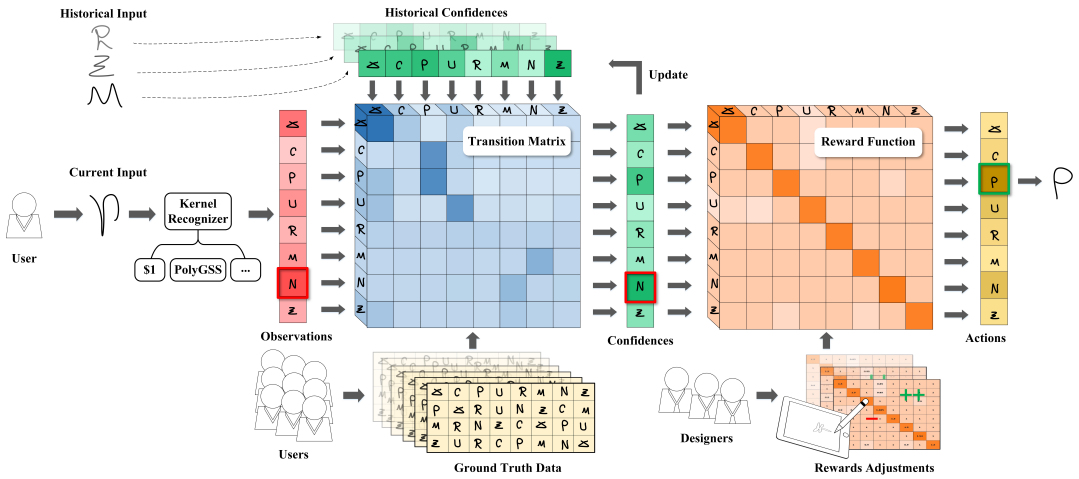

Figure 1. The workflow diagrams of *Chronos*

the four algorithms by 7.67%, 3.62%, 4.60% and 3.63%, respectively; *Chronos* also improves users' drawing experience, reducing consuming-time in three designated-drawing tasks and one free-drawing tasks by 26.28%, 23.59%, 17.79% and 20.95%.

RELATED WORK

We summarized our survey of the existing gesture recognition methods into two categories including statistical classification based methods and template matching based methods.

Statistical methods are effective to describe various types of gestures and infer from small samples how the gestures will be performed in most cases. The statistical classification based methods include Rubine's popular classifier [20] that captures 13 key properties of a gesture that matter for recognition, Hidden Markov Models (HMMs) [3, 22] that describes gesture sequence as unobservable states, and infers the gestures sequence by observing coherent user behavior, Nearest-Neighbor (NN) [1, 6] that classifies gestures as category of the nearest gesture sample in high dimensional space and, deep neural network [28, 29] which extracts more complex hierarchical features from raw images or sketch trajectories. This type of methods is accurate, robustness, and supported by profound mathematical theory. However, some of them, like HMMs and deep learning approaches, have to be trained with large-scale data set, while others require high programming skills to train, build and debug, making designers hard to understand the very meanings of gestures they designed.

Template matching methods usually measure a similarity, such as Euclidean distance, between the input gestures and templates in a library, and determine the one with maximum similarity as the intended input. Comparing to statistical methods, the template matching methods are more intuitive, more interpretative and faster, making them more popular with lightweight gesture user interfaces such as web pages and mobile applications. DTW [9] is one of the most classical template matching recognizers, which uses dynamic programming to solve the matching problem between two gesture samples. Although it supports single-sample learning, it is computationally expensive and sometimes too flexible in matching. $1 [30] recognizers aims to provide a cheap, easy-implemented and intuitive method to improve the design and prototyping for gesture user interfaces. The recognizer leverages a golden searching algorithm to find the minimum trajectory distance between input gesture and templates, which is proved to have high accuracy and low cost simultaneously. After years of improvement, it has been extended in supporting multiple-stroke gesture [2, 26, 27], enhancing computing speed [11] and reducing memory cost [25]. Nevertheless, these methods classify gestures with only static similarity measured by geometrical comparison, but ignore useful dynamic information involved in sequential user behaviors, which we emphasized in this paper.

Human behaviors usually contain rich temporal information that can be used to enhance the perception of user intentions [4]. For instance, Liu et al. analyzed user moving behaviors in maps and predicted next destination by leveraging the temporal contexts with Spatial Temporal Recurrent Neural Networks [19]. A recent work [12] models users' temporal information with temporal neural network LSTM [7] to infer their intentions, and further accelerate the selection speed in menu selection tasks. More relevant to gesture recognition, Yang Li et al. [13] demonstrated how we can decompose touch screen gestures into continuously temporal fragments, and infer gesture input with a dynamic Bayesian network [5]. These works indicate that dynamic gestures are not only related to geometric characteristics but also the kinetic ones. They greatly inspire us for using the temporal information to improve recognizer's accuracy and robustness.

In addition to focusing on the recognition algorithms themselves, tools and studios for designing gesture set and gesture interaction process were also presented. They include the famous SHARK2 [10], Gesture Coder [16], Gesture Studio [17], and more recently Gesture Script [15] and GestureWiz [24]. They provide interactive declarative guidance and visualization preview for designers to more intuitively edit gesture

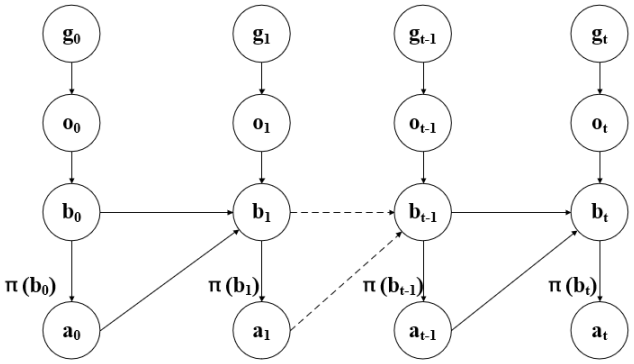

Figure 2. The whole structure of *Chronos*

templates, enhancing the on-line recognition and accelerating off-line gesture design. However, these gesture design tools usually focus on enhancing designers' ability on editing gesture templates, but ignores methods that allowing designers to actually manipulate the decision-making strategy of the system.

ALGORITHM FRAMEWORK

Inspired the work of cognitive user interfaces [31], we proposed an algorithm framework named *Chronos* that infers user's intention by considering three parts of information including input gestures, prior user intentions and rewards of possible system actions, to improve the interaction efficiency in gesture user interfaces. Comparing with above works, *Chronos*'s allow designers to take part in design of *Chronos*'s decision-making strategy which contributes to enhancing the performance of the system.

The structure of the *Chronos* is shown in Figure 2. For initialization (time 0), we set the recognition result of the kernel recognizer as confidence of the user's intention b_0 as well as the corresponding system action a_0. For the following input gestures, *Chronos* determines the final system action through three steps. Firstly, we use the kernel recognizer to provide the probabilities g_t by comparing the input gesture with the predefined gestures to obtain an observation o_t by normalization for the following steps. Secondly, we use the DBN to adjust this probability to the condition probability $b(s_t)$ of the user's intention s_t when observing the last condition probability b_{t-1}, last system action a_{t-1} and current observation o_t. Finally, we determine the final action a_t of the system by maximizing reward value that considers all possible system responds with a reward strategy $\pi(b_t)$. The last step of the framework is known as the POMDP strategy.

Kernel Recognizer

For each input gesture in time t, *Chronos* requires a initial statistical observation o_t for the input gesture. This statistical observation inputs with raw gesture data and outputs with normalized probabilities of how likely the input gesture is one in the predefined gestures. Technically, any gesture recognizer that only relies on the information of current input can be set as the kernel recognizer in *Chronos*. We chose the following four well-known gesture recognizers as the kernel recognizers:

DTW Recognizer

DTW recognizer is a simple and effective method commonly used in speech and gesture recognition. The algorithm uses dynamic programming technique to solve the matching problem between two samples with different lengths [9]. We select DTW as it compares gesture templates by building an adjacency matrix and looking for the shortest path.

$1 Recognizer Implemented in Protractor

$1 recognizer is a geometric template matcher, it compares the input stroke to gesture templates with the closest gesture distances in a 2-D euclidean space [30], which fits *Chronos*. Protractor can improves $1's speed, which it employs a novel method to measure the similarity between gestures, by calculating a minimum angular distance between them with a closed-form solution [11].

$Q Recognizer

$Q is a super-quick,articulation-invariant point-cloud sketch gesture recognizer for mobile, wearable, and embedded devices with low computing resources, it is an improvement of $P algorithm and supports single stroke and multi-stroke gestures [25]. We want to learn about recognition performance of the multi-stroke recognizer for unistroke gestures by using *Chronos*.

PolyRecGSS Recognizer

PolyRecGSS uses a nearest neighbor approach, and requires a small number of training templates for each class [6]. The similarities between gestures are calculated through a three steps procedure with Golden Section Search algorithm [8], which can be considered as observations of *Chronos*. PolyRecGSS recognizer is novel recognition algorithm and exclusive $family recognizer, so we adopt it in our experiment.

Dynamic Bayesian Confidence Inference

The *Chronos* provides a stronger prediction for the current user intention by considering users' last behavior. DBN extends Bayesian network by relating states to each other over adjacent time steps. The value of a state at time t can be calculated from the internal regressors and the immediate prior state (time t-1) [5]. We use DBN to update the user's intention s_t at time t which depends on prior user intention s_{t-1}, the prior system action a_{t-1} and current observation o_t. Since the user intention is an abstract concept, we use the Equation1 to calculate the confidence of the user intention $b(s_t)$ at time t. Besides, $p(o_t|s_t)$ can be considered as statistical observation o_t, and $p(s_t|s_{t-1},a_{t-1})$ is a transfer matrix collected from user data. The process of updating user intention confidences is shown in Figure 2.

$$\begin{aligned} b(s_t) &= p(s_t|o_t,a_{t-1},b_{t-1}) \\ &= p(o_t|s_t)\sum_{s_{t-1}} p(s_t|s_{t-1},a_{t-1})b(s_{t-1}) \end{aligned} \quad (1)$$

Partially Observable Markov Decision Process

Chronos should have the ability of choosing a most profitable system respond among all possibilities. POMDP is a generalization of a Markov decision process (MDP). It models the relationship between an agent and its environment. Formally, a POMDP is a 7-tuple (S,A,T,R,Ω ,O,γ) [23], where S is a set of states, A is a set of actions, T is a set of conditional transition probabilities between states, R: S $\times A \rightarrow \mathbb{R}$ is the reward function, Ω is a set of observations, O is a set of conditional observation probabilities, and $\gamma \in [0,1]$ is the discount factor. It is a decision process considering both confidences of user intention and rewards of interactive context. S, T, Ω and O are given by other components of *Chronos*, the system will make the most profitable operation action through the reward function combined with updated user intention confidences.

The solution of above is a DP problem with time complexity exponentially depending on the size of the state spaces, action spaces and observation spaces. We adopt an approximate solution for efficiency consideration, and assign the discount factor γ to 0 leading the algorithm to only regard the immediate maximum reward.

The role of the reward function is to guide the algorithm to find the most profitable system action heuristically by Equation2. In addition, $r(s,a_t)$ is reward function whose essence is a matrix about user intention and system actions, s is user intention, and a_t is system action at t time. Each unit in the matrix represents the reward value. In this paper, we invited 4 designers to design the reward function for *Chronos*. Figure 2 shows the process of mapping user intentions to system actions.

$$\begin{aligned} \pi(b_s) &= \arg\max_{a_t}\{r(b_t,a_t)\} \\ &= \arg\max_{a_t}\sum_{s_t} b_t(s)r(s,a_t) \end{aligned} \tag{2}$$

STUDY 1: IMPROVING RECOGNIZERS' PERFORMANCE

To verify whether *Chronos* can improve the performance of the recognizers' in practical interactive tasks, we built a drawing tool that uses sketch gestures to active system commands such as "copy" and "paste", and then, based on the drawing tool, we conducted a study to train and test the framework. The study contained four steps: 1) Collecting ground truth data with the drawing tool, 2) training the transition matrix of the framework, 3) tuning the reward functions of the framework and, 4) comparing the accuracy of recognition algorithms that with or without the framework using the ground truth data collected in step 1.

Drawing Tool

The interface of the drawing tool consisted by three parts (Fig 3a): 1) Drawing zone where users draw and performed gestures; 2) Info zone where we hinted the users important information such as the current task and gesture templates; 3) Labeling zone where users labeled their ground true intentions (Fig 3a) and designers adjusted the reward functions (Fig 3b).

When users draw a picture, they used a digital pen for drawing (Fig 3c) and used their finger for performing gestures (Fig 3d).

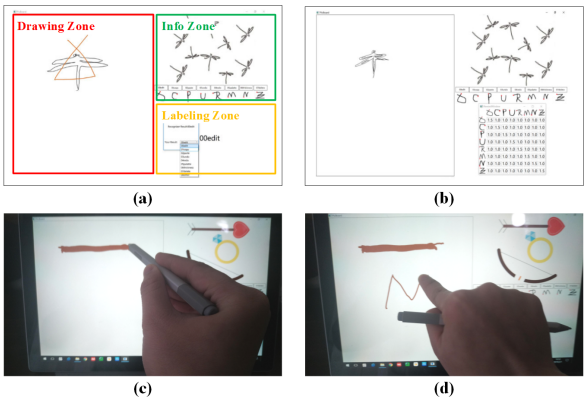

Figure 3. Interface of the drawing tool

The design purpose of this multi-channel interaction pattern is to eliminate the ambiguity between drawing and performing gestures. The drawing tool supported 8 system commands: switch to edit mode, copy, paste, undo, redo, palette mode, sketch thickness adjustment mode and delete mode. Edit mode provided translation, zooming, and rotation functions for sketches. Besides, palette mode and sketch thickness adjustment mode worked on next sketches. Finally, delete mode allows users delete ink sketches by points or by stroke.

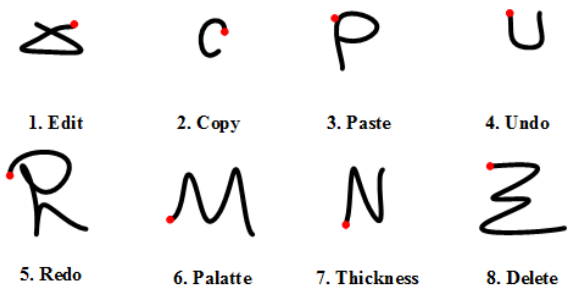

Figure 4. The gestures and their associated commands

Considering the practicability of the drawing tool and user memory burden, we selected eight sketch gestures matching 8 system commands in tool, as shown in Figure 4. The red points shown in Figure 4 are the starting points of each uni-stroke sketch gesture.

Drawing Tasks

In this study, participants were required to complete three drawing tasks in order to collect the ground true data and adjust the reward function. The tasks included two designated drawing tasks and one free drawing task as follow:

Task 1: Participants were asked to copy a picture with rich color (Figure 5a). This task required the participants to use palette mode and sketch thickness adjustment mode frequently.

Task 2: Participants were asked to copy a picture with many repetitive objects (Figure 5b). This task required

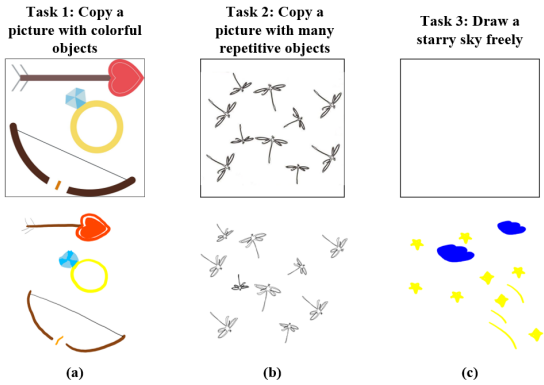

Figure 5. The drawing tasks (top) and the a typical users' work (bottom) for each task

the participants to use edit mode, copy command and paste command frequently.

Task 3: Participants were asked to draw a picture freely on a theme "starry sky" (Figure 5c). Participants could use any command they like.

Participants and Apparatus

We recruited two groups of participants, a user group and a designer group. The user group consisted of 12 participants (6 females and 6 males, with an average age of 25.0). All of them were right-handed and daily users of gesture based interfaces. The designer group consisted of 4 designers (2 females and 2 males, with an average age of 27.25). Two of them were our research team members, and the other two were front-end and interaction designers in the laboratory.

A pen-based device Surface Pro5 and original electronic pen of the device was used in experiment. The drawing tool was implemented by WPF and could run on Windows 10 operating system.

In the following sections, we introduce how we conducted the four steps of the study.

Collecting the Ground Truth Data

In this step, the participants in the user group were required to finish the three drawing tasks using gestures. The order of the tasks was counterbalanced across participants. As there is no gesture recognizer in the system yet, the participants must select their actual intention every time after they performed a gesture. Participants used a drop-down list in the labeling zone to select their actual intention and the ground truth data of performed gesture was recorded in this way (Figure 3a). Each sample of the ground true data consisted of ground truth user intention and trajectory of the input gesture. We also collected the continuity information of gestures for the training of the transition matrix. If a participant performed a series of gestures, the samples of these gestures will be marked as a gesture sequence, and, in the opposite, if a non-gesture input such as drawing a red line occur between two gestures, then they would not be treated as a gesture sequence.

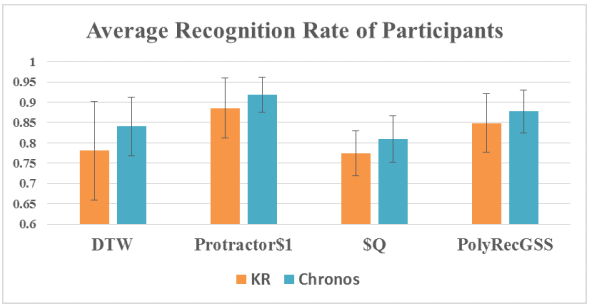

Figure 6. Average recognition accuracy of participants. Error bars represent standard deviations.

Training the Transition Matrix

In our experiment, the transition matrix omitted system action to avoid sparse data, so the transition probability between two gestures changed from $p(s_t|s_{t-1},a_{t-1})$ to $p(s_t|s_{t-1})$. After data collection, we calculated the transition probability between two gestures by Equation3. C is the count of s_{t-1} tuning to s_t calculated by the tuning count of ground truths between two consecutive user data whose ground truths are s_{t-1} and s_t. T is the sum of all gesture transition counts, K is calculated as k/n^2 (k is constant, k=6, n is number of gesture category), and S is the sum of each row of the transition matrix for row normalization. Equation3 used smoothing technique to avoid conditional transition probabilities 0.

$$p(s_t|s_{t-1}) = \frac{(K*T+C)}{S} \tag{3}$$

Tuning the Reward Functions

In the second phase, the participants in the designer group were required to finish the three drawing tasks first, and tune reward function by reward function window (Figure 3b). It consists of a matrix of 8 rows and 8 columns, which represent for user intention and system actions respectively. We required each designers to perform 3 drawing tasks by using 4 kernel recognizers combined with *Chronos*, and design one reward function for each recognizer. The order of drawing tasks and recognizers were both counterbalanced across designers. When designers changed a value of reward function window, *Chronos's* reward function was modified in time.

When we initialized reward function, the reward value of the current gesture converted to the corresponding system action was 1.5, while the reward value of the current gesture converted to the corresponding system actions of other gestures was 1.0. During the drawing tasks, designers adjusted reward function according practical continuity of gesture sequences. For instance, other gestures executed by designers were misrecognized as gesture undo when last system action was edit operation. That's because, the two gestures were often invoked continuously in tasks, the probability between two gestures was high in transition matrix, so designers needed turn down the reward value when some gestures like copy were mapping to system action undo. During tuning process, D1 thought:

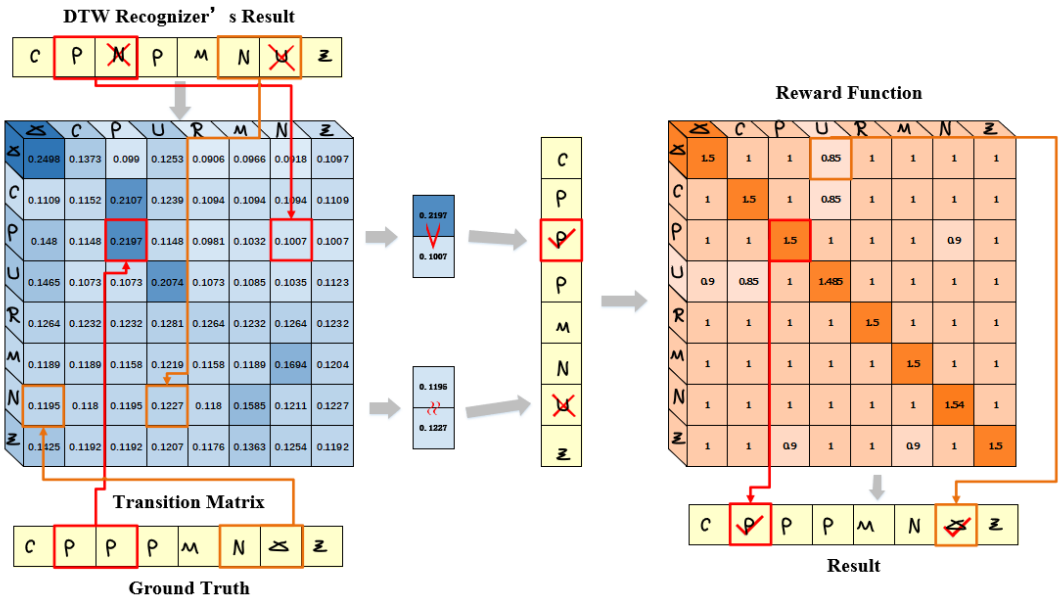

Figure 7. Influence of gesture continuity and designers's involvement on accuracy

"Some gestures are always continuously invoked in the drawing process, if the reward value was too high, it would affect the accuracy of other gestures." Besides, D3 found: *"In the drawing process, some gestures have shown strong continuity. In this case, there is no need to set the reward value too high."* Finally, the reward function of each recognizer was the average of 4 reward functions designed by 4 designers.

Comparing the Accuracy of Recognition Algorithms

We compared recognition results of the four kernel recognizers in 8 conditions. Kernel recognizer (**KR**): the results produced by the kernel recognizer (i.e. DTW, Protractor $1, $Q, PolyRecGSS). *Chronos*: the results produced by recognizer enhanced by *Chronos*. Each category of gesture had only one template.

For DTW (Figure 6), *Chronos* got the better performance (84.07%), followed by **KR** (78.08%). There were significant differences between KR and *Chronos* ($F_{1,11} = 5.907, p = 0.033$); For Protractor $1 (Figure 6), *Chronos* got the better performance (91.79%), followed by **KR** (88.58%). There were significant differences between KR and *Chronos* ($F_{1,11} = 4.952, p = 0.048$); For $Q (Figure 6), *Chronos* got the better performance (80.98%), followed by **KR** (77.42%). There were significant differences between KR and *Chronos* ($F_{1,11} = 8.273, p = 0.015$); For PolyRecGSS (Figure 6), *Chronos* got the better performance (87.94%), followed by **KR** (84.86%). There were significant differences between KR and *Chronos* ($F_{1,11} = 5.053, p = 0.046$). The results of experiment showed that the recognition rate wasn't high when there was only one training template for each gesture category, and accuracy of the best recognizer was lower than 90%. After the enhancement of *Chronos*, the accuracy of all the recognizers had been improved, which were 7.67%, 3.62%, 4.60% and 3.63% respectively.

In order to clarify the gesture continuity and designers' involvement on recognition accuracy, we gave an example shown in Figure 7 to explain reason. First of all, from ground truth and DTW recognizer's result, we could find two sketch gestures were misrecognized. In the next, we leveraged the transition probability between two continuous gestures to improve recognizer's performance. In transition matrix, the probability of gesture paste converting into itself was high, confidence of gesture paste was raised, and the misrecognized was corrected. Nevertheless, the probability between gesture thickness and gesture edit was low, thus it failed to correct the misrecognition. Then we utilized *Chronos*'s reward function designed by designers to further enhance the recognizer. Gesture edit was easier to be mapped to system action undo when last user intent was gesture thickness, so the reward value that gesture edit mapping to system action undo was turned down by our designers in DTW's reward function. Since the confidence was punished by the reward function, the ground truth was recognized out.

STUDY 2: TESTING THE FRAMEWORK IN THE FIELD

In this study, we tested if our framework improved the interaction efficiency in drawing tasks in the field. We compared time consumption between the Protractor $1 (**KR**), which was the best kennel recognition from results of study 1, and *Chronos* with Protractor $1 built-in. Subjective feedback was also recorded.

Design

We selected four drawing tasks including three designated tasks and one free drawing task in this study:

Task 1 and *Task 2*: The same as the first two tasks in our study 1.

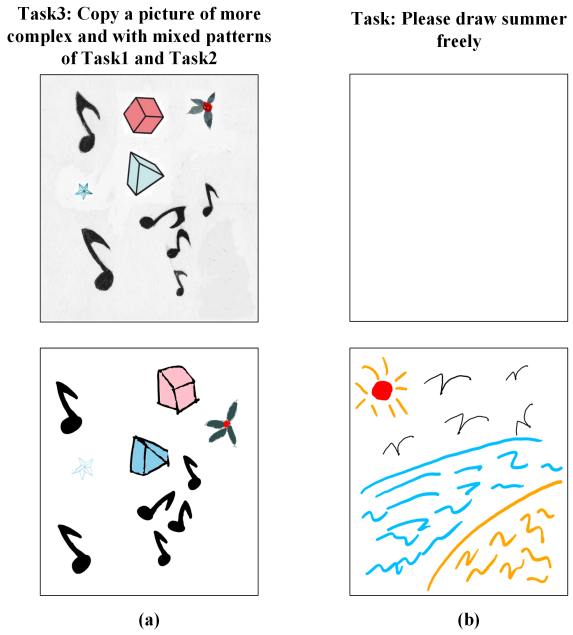

Figure 8. The drawing tasks (top) and the a typical users' work (bottom) for added tasks

Task 3: Participants were asked to copy a picture that is more complex and with mixed patterns of task 1 and 2 (i.e. colorful, having repetitive objects with different scales), as showed in Figure 8a. This task required the participants to use more commands and their orders are more unpredictable.

Task 4: Participants were asked to do a new free drawing on a new theme "summer" (Figure 8b).

Participants and Procedure

The 12 participants of study 1 were invited to take part in this study. They were asked to draw the four tasks with one of the algorithms (i.e. **KR** and *Chronos*) first and then the other. The order of tasks and algorithms were both counterbalanced across participants. We got 4 tasks $\times$ 2 algorithms $\times$ 12 participants = 96 drawing in total.

Results

We compared the task completion time of the four tasks in two conditions. Kernel recognizer (**KR**): the results produced by the kernel recognizer (Protractor $1). *Chronos*: the results produced by recognizer enhanced by *Chronos*.

For the first task (Figure 9), *Chronos* got the better performance (224.22s), followed by **KR** (304.16s). There were significant differences between KR and *Chronos* ($F_{1,11} = 5.950, p = 0.033$); For the second task (Figure 9), *Chronos* got the better performance (243.32s), followed by **KR** (318.42s). There were significant differences between KR and *Chronos* ($F_{1,11} = 11.229, p = 0.006$); For the third task (Figure 9), *Chronos* got the better performance (369.87s), followed by **KR** (449.90s). There were significant differences between KR and *Chronos* ($F_{1,11} = 23.39, p = 0.005$); For the fourth task (Figure 9), *Chronos* got the better performance (96.93s),

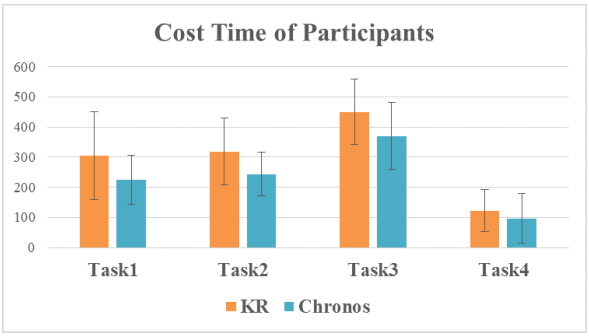

Figure 9. Average cost time of participants. Error bars represent standard deviations.

followed by **KR** (122.62s). There were significant differences between KR and *Chronos* ($F_{1,11} = 5.13, p = 0.045$). In a few cases, *Chronos* cost more time or the two conditions cost roughly the same time, the reason was that some participants used KR in the latter order and they were already familiar with the task pictures. In most cases, *Chronos* profoundly improved the fluency of interaction and reduced the time-consuming of 26.28%, 23.59%, 17.79% and 20.95% respectively.

Subjective Feedback

We conducted a post-test survey with a 5-points Likert scale for rating the 2 conditions according to preference, accuracy, and fluency, as showed in Figure 10. For preference (Figure 10), *Chronos* got the better performance (4.33), followed by **KR** (2.75). There were significant differences between KR and *Chronos* ($F_{1,11} = 67.305, p < 0.001$); For accuracy (Figure 10), *Chronos* got the better performance (4.25), followed by **KR** (2.33). There were significant differences between KR and *Chronos* ($F_{1,11} = 70.108, p < 0.001$); For fluency (Figure 10), *Chronos* got the better performance (4.17), followed by **KR** (2.33). There were significant differences between KR and *Chronos* ($F_{1,11} = 45.897, p < 0.001$).

We received positive responses about *Chronos*'s practicality and good user experience:

> *P2:"Chronos brought me a more fluid user experience, and I draw much quicker".*

> *P5:"Since kernel recognizer was enhanced by Chronos, I became more willing to paint by using the drawing tool".*

Participants were able to feel about the enhancement of recognition in some gestures, and were amazed about the way *Chronos* made this enhancement by extracting gesture continuity information:

> *P1:"When using other recognizers, the gesture of "paste" was sometimes misrecognized as gesture of "thickness mode". But this kind of mistake seldom happened to Chronos."*

> *P8:"I feel Chronos never make mistake when I perform the "edit" gesture, it seems the Chronos already know what I plan to do in some situations."*

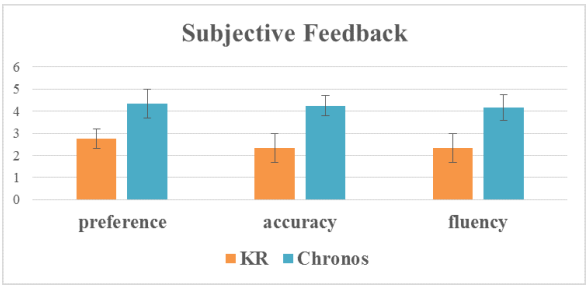

Figure 10. Experiment experience of participants. Error bars represent standard deviations.

Participants thought that the idea of inviting designers to design reward function was interested, novel and helpful:

> *P7:"When the designers did the reward function tuning in the realistic scenarios, misrecognitions of gesture was reduced a lot".*

Participants enjoyed the drawing by using the algorithm framework and created some nice works shown in Fig 5b and Fig 8b:

> *P3:"Fast operation switching and high precision gesture recognition made my drawing process more smooth and enabled me to concentrate more on the painting."*

> *P9:"When I drew stars continuously, the fluid user experience brought me unprecedented pleasure, which was very helpful for me to complete tasks."*

CONCLUSION

This paper presents an algorithm framework *Chronos* that improves the performance of gesture recognition by gesture continuity and designers' involvement. The results indicate that the previous user intention and optimal planning contribute to enhancing recognition accuracy and user experience. We received feedback on improving fluency and accuracy for of using *Chronos*. We shows that *Chronos* robustly improves the performance of gesture recognition of different gesture recognizers in different drawing tasks. However our framework is limited by the greedy searching algorithm for optimal system action. In the future, we will improve the searching algorithm to consider global optimum or approximate global optimum strategy. We are also interested in testing the feasibility and usability of *Chronos* in a wider range of application scenarios.

REFERENCES

1. Lisa Anthony and Jacob O. Wobbrock. 2010. A Lightweight Multistroke Recognizer for User Interface Prototypes. In *Proceedings of Graphics Interface 2010 (GI '10)*. Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 245–252. http://dl.acm.org/citation.cfm?id=1839214.1839258
2. Lisa Anthony and Jacob O. Wobbrock. 2012. $N-protractor: A Fast and Accurate Multistroke Recognizer. In *Proceedings of Graphics Interface 2012 (GI '12)*. Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 117–120. http://dl.acm.org/citation.cfm?id=2305276.2305296
3. Tim Dittmar, Claudia Krull, and Graham Horton. 2015. A new approach for touch gesture recognition: Conversive Hidden non-Markovian Models. *Journal of Computational Science* 10 (2015), 66 – 76. DOI: http://dx.doi.org/https://doi.org/10.1016/j.jocs.2015.03.002
4. Paul Fraisse. 1984. Perception and Estimation of Time. *Annual Review of Psychology* 35, 1 (1984), 1–36.
5. Nir Friedman, Kevin P Murphy, and Stuart J Russell. 1998. Learning the structure of dynamic probabilistic networks. *uncertainty in artificial intelligence* (1998), 139–147.
6. Vittorio Fuccella and Gennaro Costagliola. 2015. Unistroke Gesture Recognition Through Polyline Approximation and Alignment. (2015), 3351–3354.
7. Sepp Hochreiter and Jurgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9, 8 (1997), 1735–1780.
8. Kiefer J. 1953. Sequential Minimax Search for a Maximum. *Proc. Amer. Math. Soc.* 4, 3 (1953), 502–506.
9. Eamonn J Keogh and Michael J Pazzani. 2001. Derivative Dynamic Time Warping. (2001), 1–11.
10. Per Ola Kristensson and Shumin Zhai. 2004. SHARK 2 : a large vocabulary shorthand writing system for pen-based computers. (2004), 43–52.
11. Yang Li. 2010. Protractor: a fast and accurate gesture recognizer. (2010), 2169–2172.
12. Yang Li, Samy Bengio, and Gilles Bailly. 2018. Predicting Human Performance in Vertical Menu Selection Using Deep Learning. *human factors in computing systems* (2018), 29.
13. Yang Li, Hao Lu, and Haimo Zhang. 2014. Optimistic Programming of Touch Interaction. *ACM Transactions on Computer-Human Interaction* 21, 4 (2014), 24.
14. Allan Christian Long. 1998. Improving gestures and interaction techniques for pen-based user interfaces. (1998), 58–59.
15. Hao Lu, James Fogarty, and Yang Li. 2014. Gesture script: recognizing gestures and their structure using rendering scripts and interactively trained parts. (2014), 1685–1694.
16. Hao Lü and Yang Li. 2012. Gesture Coder: A Tool for Programming Multi-touch Gestures by Demonstration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, New York, NY, USA, 2875–2884. DOI: http://dx.doi.org/10.1145/2207676.2208693
17. Hao Lü and Yang Li. 2013. Gesture Studio: Authoring Multi-touch Interactions Through Demonstration and Declaration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 257–266. DOI: http://dx.doi.org/10.1145/2470654.2470690

18. Andre Meyer. 1995. Pen computing: a technology overview and a vision. *ACM Sigchi Bulletin* 27, 3 (1995), 46–90.

19. Liu Q, Wang S, Wang L, and et al. 2016. Predicting the Next Location: A Recurrent Model with Spatial and Temporal Contexts. *Thirtieth Aaai Conference on Artificial Intelligence* (2016).

20. Dean Rubine. 1991. Specifying gestures by example. *international conference on computer graphics and interactive techniques* 25, 4 (1991), 329–337.

21. Frode Eika Sandnes, Tek Beng Tan, Anders Johansen, Edvin Sulic, Eirik Vesterhus, and Eirik Rud Iversen. 2012. Making touch-based kiosks accessible to blind users through simple gestures. *Universal Access in The Information Society* 11, 4 (2012), 421–431.

22. Tevfik Metin Sezgin and Randall Davis. 2005. HMM-based Efficient Sketch Recognition. In *Proceedings of the 10th International Conference on Intelligent User Interfaces (IUI '05)*. ACM, New York, NY, USA, 281–283. DOI: http://dx.doi.org/10.1145/1040830.1040899

23. Edward J Sondik. 1978. The Optimal Control of Partially Observable Markov Processes over the Infinite Horizon: Discounted Costs. *Operations Research* 26, 2 (1978), 282–304.

24. Maximilian Speicher and Michael Nebeling. 2018. GestureWiz: A Human-Powered Gesture Design Environment for User Interface Prototypes. (2018), 107.

25. Anthony L. Vatavu, R.-D. and J.O Wobbrock. 2018. $Q: A Super-Quick, Articulation-Invariant Stroke-Gesture Recognizer for Low-Resource Devices. In *Proceedings of 20th International Conference on Human-Computer Interaction with Mobile Devices and Services. New York: ACM Press (MobileHCI '18)*. https://doi.org/10.1145/3229434.3229465

26. Radudaniel Vatavu. 2017. Improving Gesture Recognition Accuracy on Touch Screens for Users with Low Vision. (2017), 4667–4679.

27. Radudaniel Vatavu, Lisa Anthony, and Jacob O Wobbrock. 2012. Gestures as point clouds: a $P recognizer for user interface prototypes. (2012), 273–280.

28. Saiwen Wang, Jie Song, Jaime Lien, Ivan Poupyrev, and Otmar Hilliges. 2016b. Interacting with Soli: Exploring Fine-Grained Dynamic Gesture Recognition in the Radio-Frequency Spectrum. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST '16)*. ACM, New York, NY, USA, 851–860. DOI: http://dx.doi.org/10.1145/2984511.2984565

29. Xinggang Wang, Xiong Duan, and Xiang Bai. 2016a. Deep Sketch Feature for Cross-domain Image Retrieval. *Neurocomputing* 207 (2016), 387–397.

30. Jacob O Wobbrock, Andrew D Wilson, and Yang Li. 2007. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. (2007), 159–168.

31. Steve J Young. 2010. Cognitive User Interfaces. *IEEE Signal Processing Magazine* 27, 3 (2010), 128–140.