On-Pen Handwritten Word Recognition Using Long Short-Term Memory Model

Prepared by Hsun-Fa Cho

(a37805@gmail.com)

Directed by Prof. Pai H. Chou

(phchou@cs.nthu.edu.tw)

Embedded Platform Lab

Dept. of Computer Science

National Tsing Hua University

Hsinchu, Taiwan 30013

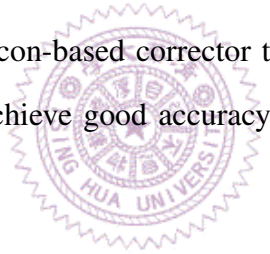
August 28, 2018

摘要

近年來，因穿戴式裝置的流行，基於動作感測的手寫辨識研究越來越多，例如空中手寫、智慧筆等等，但是能達到實用水平的應用卻很少。受限於感測器的限制、連續動作的分割難度這些問題，手寫動作辨識很難做有效率的文字輸入。在這篇論文中，我們提出了一種安裝在筆上的新型硬體裝置，搭配以長短期記憶模型(LSTM)為核心的辨識系統，讓使用者能夠用任何他們想拿來寫字的筆，作為一個有效率的文字輸入介面。

Abstract

This thesis describes a system for text input from handwriting using a conventional pen with a clip-on sensing unit. The clip-on unit is a wireless sensor node that collects data from a triaxial accelerometer and a triaxial gyroscope and transmits it to a conventional personal computer. The host computer then performs segmentation to handle continuous handwriting, followed by LSTM-based classification. Moreover, we use a lexicon-based corrector to increase the accuracy. Experimental results show our proposed system to achieve good accuracy and reasonable latency for interactive use.



Contents

List of Figures

List of Tables

Acknowledgments

I would like to express my sincere gratitude to my thesis advisor Prof. Pai H. Chou for his continuous guidance and encouragement. Besides, I am grateful to Yu-Chieh Teng, Yi-Chang Huang, Yu-Hung Yeh, James Chan, and Zhe-Ting Liu for their advice on deep learning and Tensorflow library. Finally, I also acknowledge the assistance by other members of the Embedded Platform Lab (EPL) at National Tsing Hua University (NTHU) in Taiwan.

Chapter 1

Introduction

1.1 Motivation

In recent decades, handwriting recognition has become mature and widely used. They are now supported as first-class features on modern smartphone and tablet OSs, and they are also used for trackpad-based text input. Although they can be useful, they require the user to be operating a touch-screen or a computer peripheral, which is not the same as writing on paper with a pen.

We believe that there is a need for text input using a natural interface, such as that of a conventional pen on physical paper. Such a natural interface should require minimal setup, and the use of cameras for tracking the written shape would be considered unnatural. Moreover, it should not require the use of a special pen, as the experience of using the user's own pen should be maximally preserved. Moreover, such a system should not require the user to undergo special training to write things in a specific way, such as the use of the Graffiti alphabet; instead, they should be allowed to write words the same way as before, assuming it is intelligible to humans, and there should be no special requirement for pausing.

Such a vision of a new human-computer interface (HCI) device has become feasible in recent years, thanks to the advances in both hardware and software. Motion sensors in the form of 9-DoF inertial sensors (triaxial accelerometer, triaxial gyroscope, and triaxial magnetometer) are now available in a package no larger than 3mm $\times$ 3mm, and microcontroller units (MCU) with both processing and wireless communication capabilities that enable direct connectivity to computers are also available in similar sizes. On the software side, advances in deep learning have enabled accurate recognition from such input. In this thesis, we aim to show the feasibility of a powerful, practical on-pen handwriting

recognition system. Instead of customizing an inertial pen, we make the sensor a clip-on unit so that the users can write text using their favorite pen.

1.2 Contributions

The contributions of our proposed method are twofold. First, we are the first to propose and prototype a miniature embedded system that can perform sensing and communication of the motion data from the pen it is clipped on, so that it can be recognized as text input to a computer. Second, we propose an automatic segmentation algorithm for the writing-motion data so that our system can perform not only character-level but also word-level recognition accurately.

1.3 Thesis Organization

This thesis is organized as follows. Chapter **??** introduces related works on handwriting recognition. Next, Chapter **??** describe the architecture of our system. Third, Chapter **??** discusses the issues of data preprocessing and describe how we process the data. Chapter **??** also illustrates the solutions of data segmentation problem. In Chapter **??**, we present background on recurrent neural network and how LSTM can improve it. To enhance the accuracy of recognition, we use lexicon support to calibrate the output of the LSTM classifier. We also show how a Bayes's corrector works in Chapter **??**. In Chapter **??**, we evaluate our proposed method with three different views: segmentation robustness, the accuracy of selecting different features, and the effects of lexicon support. Finally, Chapter **??** we summarizes our system and lists directions for future research.

Chapter 2

Related Work

This chapter surveys techniques for recognizing hand-written letters in computer vision and from motion. They are further divided into recognizing one isolated letter at a time and recognizing a word composed of multiple letters. Our work falls into the category of the latter, which is also called continuous handwriting recognition.

2.1 Isolated Handwriting Identification

Feature fetching is the primary element on vision-based recognition of handwritten letters. To fetch features from the image, we traditionally use computer vision models, such as PCA and LDA. Gao et al. [**?**] propose an LDA-based compound distance method to recognize isolated Chinese text in images. On the ETL98 and CASIA database, they reduce the error rate of baseline MQDF factors by over 26%. Ebrahimzadeh et al. [**?**] use a HOG-based method to fetch features and a linear SVM classifier to recognize a handwritten digit, and they result in a 97.25% accuracy on the MNIST database. Katiyar et al. [**?**] combine several feature-fetching methods consisting of the box approach and mathematical methods. They use a total of 144 hybrid features as the input of MLPNN (Multilayer Perceptron Neural Network) and achieve an overall recognition rate of 93.23% on the CEDAR CDROM-1 database. The results show that computer vision methods can be effective for well-extracting features from images. However, we usually do not consider cameras as a text-input interface due to privacy concerns.

Wearable devices with built-in sensors have become a viable option for text input in recent years. Motion sensors have the advantages of light weight, low power consumption, and better privacy.

Handwriting recognition from motion data can be divided into trajectory reconstruction and processing raw sensor data. Bahlmann et al. [**?**] introduce a new SVM model that uses a GDTW-kernel implemented by dynamic time warping (DTW). They evaluate the SVM model on the 1c section of the UNIPEN database, which consists of trajectory data, and reach an 11.7% error rate. Sepahvand et al. [**?**] proposes a GPML (Genetic Programming-based Metric Learning) model to recognize the handwriting of isolated Persian/Arabic letters. They first reconstruct the trajectory of an inertial pen, and then feed the geometrical features of the trajectory into the GPML model. Their method achieves 97.3% and 91.2% accuracy for writer-dependent and -independent cases, respectively. Instead of trajectory, Oh et al. [**?**] uses the sensor signals directly, which are collected from an inertial hand-held wand. They use an LDA model to project the triaxial accelerometer and the triaxial gyroscope signals to a class space of ten digits and three gestures. The full 6-axis configuration achieves a 93.23% accuracy.

2.2 Continuous Handwriting Recognition

Continuous handwriting recognition, i.e., one word or one sentence at a time, is more practical than recognizing isolated letters, which requires pausing or spacing that would otherwise be inserted deliberately by the user. However, continuous handwriting recognition is much more difficult, and the main challenge is with either pre-segmentation or post-decoding. The most common approach is to use a hierarchical architecture. For the image field, Bluche et al. [**?**] combine CNN (Convolutional Neural Network) feature fetching and an HMM (Hidden Markov Model) decoder. The proposed method achieves a Word Error Rate (WER) of 6.9% on RIMES-WR2 database. Kozielski et al. [**?**] applies an LSTM model to topology a 200-dimensional feature from an image. Next, a PCA transformation is used to extract 20 main components as the activation of a Tandem GHMM. They report a 13.7% WER on the RIMES database. Shkarupa et al. [**?**] provides two approaches to recognizing continuous handwriting text. First, they use a CTC (Connectionist Temporal Classification) approach to decode the output of a BLSTM (Bidirectional LSTM) model. The softmax output is interpreted as a probability distribution array and maps it onto each class. The second approach is a Sequence-to-Sequence model, which consists of an LSTM encoder and an LSTM decoder. The raw image is partitioned into several 1-pixel pieces and encoded as features and states. The decoder decodes the output from the encoder as the last label until it encounters an "END" character. They evaluate both

methods on KNMP and Stanford datasets. Eventually, the CTC approach achieves 21.7% WER, and seq2seq-based approach achieves 27.21% WER.

For motion-based recognition of continuous handwriting, Amma et al. [**?**] proposed a statistical-based natural language processing method to recognize airwriting motion. They first detect the Write/No-write state and use the 3-gram processing model, which is a linguistic probability model based on HMM, to identify airwriting word. For the writer independent setup, an 11% WER is achieved, and for the writer-dependent setup, 3% is achieved. Chen et al. [**?**, **?**] reconstructs the trajectory of the airwriting and classifies the path by an HMM classifier. Finally, they observe an overall 9.84% Segment Error Rate (SER) in their experiments.

Chapter 3

System Overview

This chapter describes the architecture for our proposed on-pen handwritten word recognition system. It is organized into the pen side and host side. Our system architecture is shown in Fig. **??**. On the pen side, we need a wireless sensor node that can perform inertial sensing and wireless transmission to the host. The host is a personal computer that receives and processes the sensing data before outputting the recognized words.

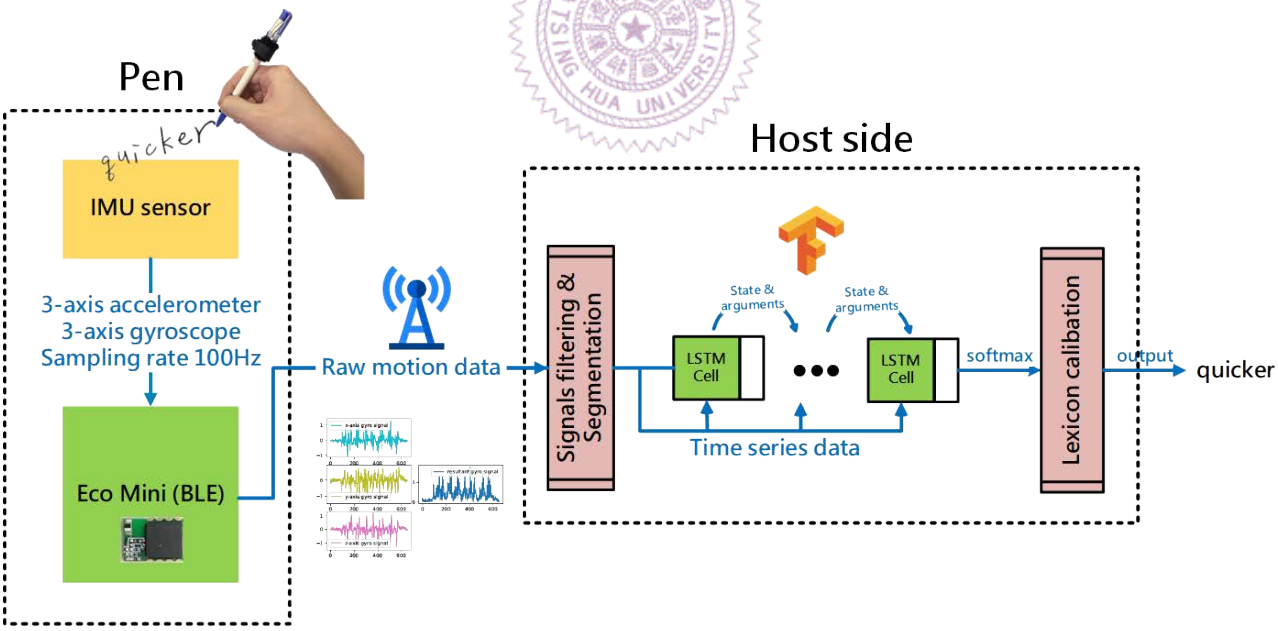


Figure 3.1: The architecture of the on-pen handwritten word recognition system

3.1 Pen Subsystem

Our pen subsystem is a wireless inertial-sensing node that can be clipped onto any pen of the user's choice for text input. This approach represents a much more cost-effective alternative to creating a customized pen that the user may not like. Such features can be realized using a wireless sensor node such as the EcoMini [**?**]. It is based on the CC2541 MCU with integrated Bluetooth Low Energy (BLE) transceiver, and the inertial sensor is the MPU9250 from InvenSense for 9-DoF (i.e., triaxial accelerometer, triaxial gyroscope, and triaxial compass) sensing via I2C protocol. We sample the accelerometer and gyroscope at 100 Hz and transmit the data to the host.

3.2 Host Subsystem

The host side receives the inertial sensor data from the pen. After preprocessing the raw motion data, it performs recognition in two major steps: (1) recognizing each letter in the data by an LSTM classifier, and (2) calibrating the result of LSTM model with lexicon support.

For segmentation, our system segments the continuous data from word-level to letter-level, and the LSTM classifier can recognize each letter signals among raw data. Since the origin classifier still cannot well identify each letter, we use a Bayes' corrector to calibrate the system output. According to the experiment results, we reach our goals and achieve an overall 26.4% WER and 12% CER performance with the system architecture shown in this chapter.

Chapter 4

Data Preprocessing and Segmentation

In this chapter, we show three procedures for data preprocessing and segmentation. First, we remove the contribution of gravity to reduce the signal difference. Next, we filter and resize the data to smooth and normalize the IMU signals. Third, we segment the motion data such that each contains one letter.

4.1 Gravity Elimination

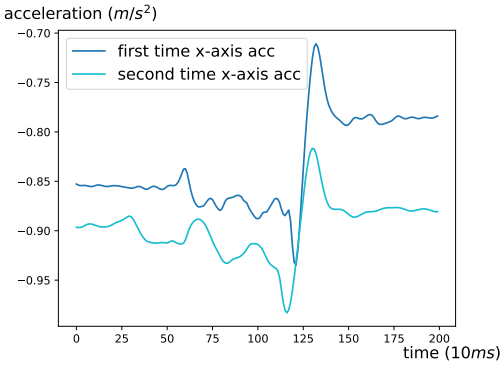
To calculate position from linear acceleration, it is necessary to remove the contribution of gravity from IMU signals. We also need to eliminate the gravity when we directly analyze the pattern of the signals, even though we do not use the pen's position as a feature. For instance, if the user writes the letter 'q' twice, the observed signals of the accelerometer may be distinct. The distinct writing events may impact the results of the recognition model to some extent.

Fig. **??** shows that we compare two writing event with and without gravity elimination. The shapes of curves are similar, but their baselines are different. It may confuse the recognition model. In other words, instead of reconstructing the trajectory, the reason we remove the contribution of gravity is to reduce distinctions of all data in each label.

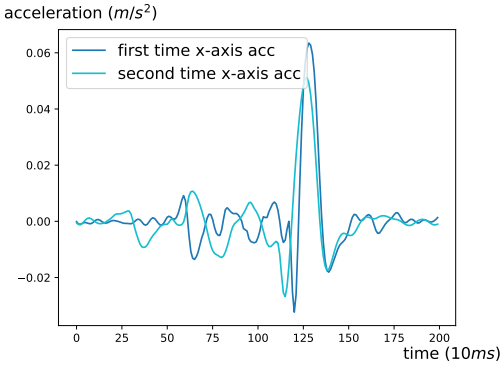
Generally, the force of gravity always influences the acceleration value observed:

$$a_o = -g - \sum \frac{F_s}{mass} \tag{4.1}$$

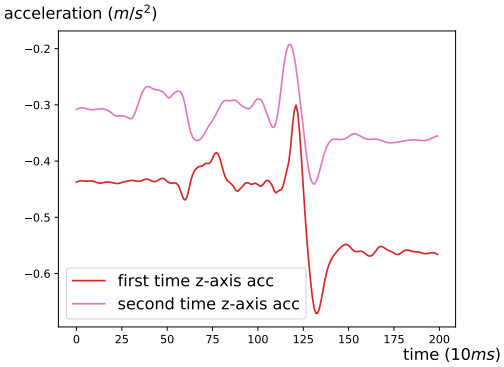
According to the equation, we can remove the contribution of gravity by applying either a high-pass or a low-pass filter. In our work, we use a low-pass filter, and the implementation is shown in



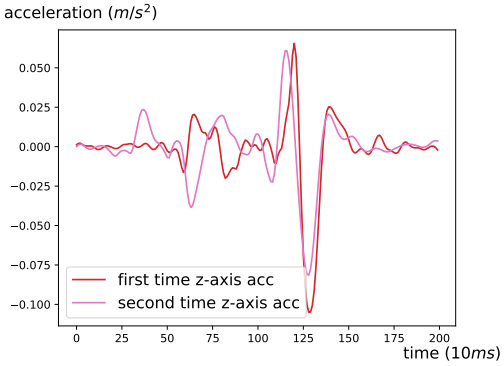
(a) x-axis accelerometer signal without gravity elimination



(b) z-axis accelerometer signal without gravity elimination



(c) z-axis accelerometer signal without gravity elimination



(d) z-axis accelerometer signal with gravity elimination

Figure 4.1: Writing the letter 'q' twice and comparing the difference between removing and not removing the influence of gravity from accelerometer data. The inertial data shown above have already been processed by a Butterworth low-pass filter and have been normorlized.

Algorithm **??**. The signals after eliminating gravity are shown in the right part of Fig. **??**.

4.2 Signal Processing

The IMU data contains not only the contribution of gravity but also high-frequency noises. Triaxial acceleration signals are widely used for classifying different gestures in motion-recognition works. The motions to recognize often include swinging their arms, legs, or body. Therefore, when processing these gestures' signals, they only need to trivially get rid of the signals whose frequency is higher than a specified threshold. However, it is difficult to recognize some small but fast motions such as writing with a pen. The frequency of the pen's movement is between 1 and 10 Hz, depending on the writer. Additionally, the magnitude of the triaxial acceleration may be in the noise level, and filters could easily filter out true but small signals as if they were noise. Consequently, we cannot apply the filter indiscriminately as in Algorithm **??**. Besides the filter rate α, a low-pass filter also contains other

parameters, such as sampling rate ω, order n, and cutoff frequency ω_c. We implement a Butterworth low-pass filter whose mathematical model can be represented by Eq. (**??**). The higher the order, the faster the slope rolls off. Unlike other filters, the Butterworth filter does not cause non-monotonic ripples in the cutoff frequency, regardless of the order.

After filtering, we find that the magnitudes and durations of the signals are related to how fast the user writes. The faster the writing speed, the larger the fluctuation in the acceleration; the slower the writing speed, the longer the event window. As a result, normalization is needed for the recognition. We first apply feature scaling to scale the range of magnitudes, which is shown in Eq. (**??**). Feature scaling is a general method to rescale the range of value while still keeping the signal's features. Last, the lengths of different letters' windows are usually different. Hence, linear interpolation as in Eq. (**??**) is used to resize the original signal:

$$G_n(\omega) = 1/\sqrt{1+(\omega/\omega_c)^{2n}}, \qquad \text{for } n = 1, \omega = 100 \text{ Hz}, \omega_c = 2.2 \text{ Hz} \tag{4.2}$$

$$x_i' = \frac{x_i - \min(X)}{\max(X) - \min(X)}, \qquad X = \{x_1, x_2, \cdots, x_n\} \tag{4.3}$$

$$f(x) = \frac{f(x_{\text{prev}})(x_{\text{next}} - x) + f(x_{\text{next}})(x - x_{\text{prev}})}{x_{\text{next}} - x_{\text{prev}}} \tag{4.4}$$

4.3 Data segmentation

Data segmentation is the most important part of all preprocessing procedures in continuous recognition of handwritten words. Extensive research has been done on how to separate each letter from one writing event.

4.3.1 Related Work

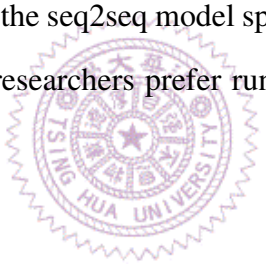
HMM is one traditional way to segment sequence data. Bluche et al. [**?**] apply grapheme segmentation [**?**] and ConvNN to the recognition of continuous handwritten words. Grapheme segmentation is implemented by an over-segmenting approach. They separate the handwriting into numerous graphemes and then reconstruct them as individual letters by an HMM model. After the segmenting, they identify each segment by a convolutional neural network.

Another traditional way to segment continuous data is DTW algorithm, which was originally used

for calculating the similarity between two streams of time-series data. Osman et al. [**?**] implement an online signature verification system based on DTW segmentation.

As a modern approach, Seq2seq model is designed to transform one sequence into another sequence with a different length. Sueiras et al. [**?**] use a Seq2seq model to recognize continuously handwritten words in images. Moreover, they improve the segmentation robustness of Seq2seq model with a convolutional reader and attention mechanism.

We evaluated the approaches surveyed above to determine the most suitable one for our system. First, we applied an HMM to handle the segmentation. However, because of the problems we describe in Section **??** and Section **??**, it was not effective. Next, we also tried DTW-based segmentation. Unfortunately, due to the time complexity of the DTW algorithm, the overhead of matching 26 patterns is too high. Furthermore, if we segment data by DTW comparison, we have to determine how frequently a match occurs. This is another issue that should be discussed. A seq2seq model not only segments but also recognizes continuous handwritten words at the same time. It seems that seq2seq model is a good choice for our work. Nevertheless, the seq2seq model spends a long time on encoding, attention mechanism, and decoding. Therefore, researchers prefer running it offline rather than online in an application.



4.3.2 Our Approach

Our approach to segmentation works as follows. First, instead of segmenting on accelerometer signals, we segment on gyroscope data. A gyroscope outputs the angular velocity about each of the three axes of rotation. It is generally used for calculating the angle of rotation, but may gradually drift after a while. As a result, the signals of the gyroscope need to be calibrated either with magnetometer signals or by a high-pass filter. The reason for using gyroscope data is that it has a higher signal-to-noise ratio (SNR) than an accelerometer does for small movement. Therefore, we implement a threshold-based segmentation as shown in Algorithm **??**.

On line 3, we derive the resultant angular velocity from the triaxial gyroscope signals by a cosine formula. Next, on lines 4 and 5, we filter the raw resultant angular velocity signal with a Butterworth low-pass filter and find all of its local minima. Two local minima can be paired up as one window. Afterward, line 7 through line 25 iterate over each window to determine if it contains the entire motion for the letter. If not, then the current window is deprecated or concatenated with the next; otherwise,

the head and tail points are added to W_s list, and the window is reset. Finally, the W_s contains the list of valid segments. The results of each step in all segmentation procedures are shown in Fig. **??**.

Algorithm 1: Low-pass filter implementation

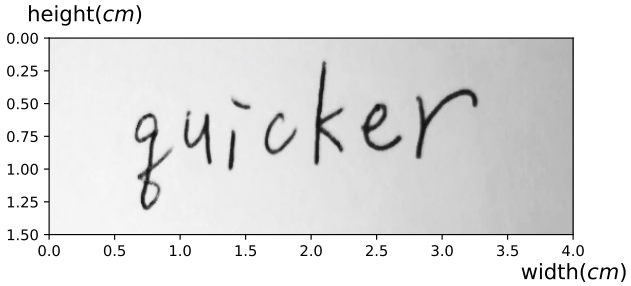
input: a_o : acceleration observed
input: G : current gravity
output: a_r : acceleration after removing gravity

```
constant α = 0.8;
initialize G ← 0;
while input new a_o do
    G ← α × G + (1 − α) × a_o;
    output a_r ← a_o − G;
end
```

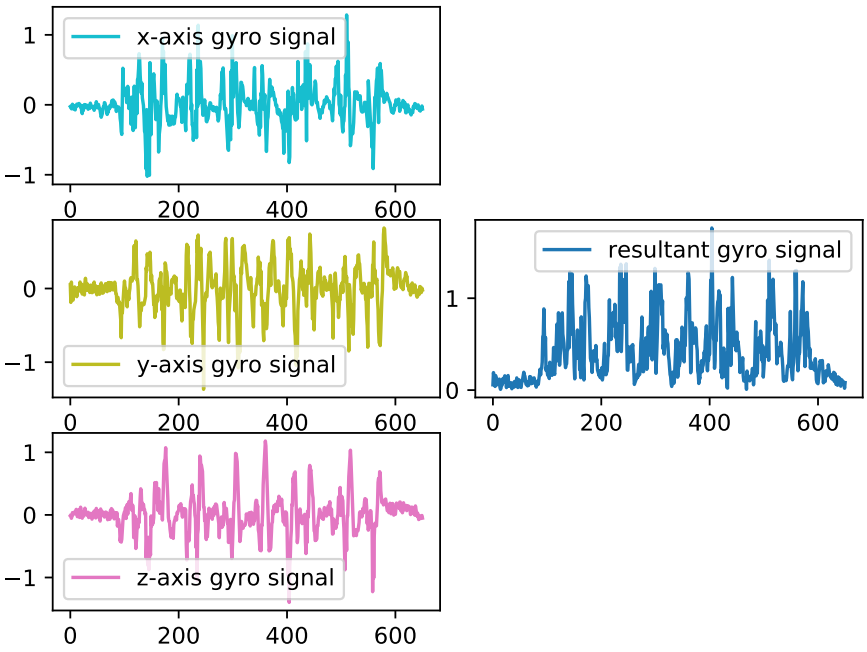
Algorithm 2: Data segmentation algorithm

input: $\omega_x, \omega_y, \omega_z$: triaxial gyroscope signals
output: W_s : list of windows

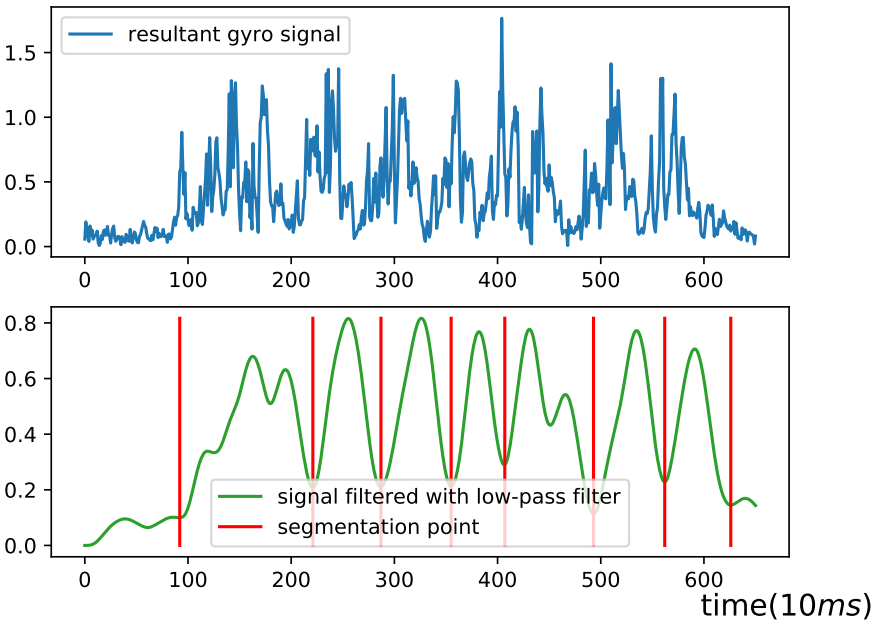
constant $\beta = 1/3.5$;
initialize $W_s \leftarrow \{\}$;
$\omega_i \leftarrow \left\{ \sqrt{\omega_{xi}^2 + \omega_{yi}^2 + \omega_{zi}^2} | 0 \le i \le n-1 \right\}$;
$\omega \leftarrow$ Butterworth low-pass filter($\omega_{0\cdots n-1}$);
$P_{0\cdots m-1} \leftarrow m$ of all local minima from ω;
$height \leftarrow \max_{i=0}^{n-1}(\omega_i) - \min_{i=0}^{n-1}(\omega_i)$;
$head, tail \leftarrow 0$;
window $w \leftarrow []$;
for $i \leftarrow 0 \cdots m-2$ **do**
    if *w is empty* **then**
        $w \leftarrow [\omega_j | j \in [P_i \cdots P_{i+1}]]$;
        $head \leftarrow P_i$;
    end
    $tail \leftarrow P_{i+1}$;
    $rise \leftarrow \max(w) - \omega_{head}$;
    $fall \leftarrow \max(w) - \omega_{tail}$;
    $range \leftarrow \max(w) - \min(w)$;
    if $range < \beta \cdot height$ **then**
        $w \leftarrow []$;
        continue;
    end
    if $rise < \beta \cdot height$ ***or*** $fall < \beta \cdot height$ **then**
        w.append($[\omega_k | k \in [P_{i+1} \cdots P_{i+2}]]$);
        continue;
    end
    $W_s \leftarrow W_s \cup \{(head, tail)\}$;
    $w \leftarrow []$;
end
return W_s;



(a) Sample handwriting of the word “quicker”



(b) Transforming triaxial gyroscope signal to the resultant signal by the cosine formula



(c) Filtering the resultant gyroscopic signal to find the segmentation points

Figure 4.2: Three steps of segmentation: (1) calculating the resultant signal, (2) filtering the resultant signal, (3) finding local minimum and searching for the usable window from all local minimum points.

Chapter 5

Recognition Method

This chapter describes our methods for analyzing the on-pen motion sensor data. We first give a review of Recurrent Neural Network (RNN). Next, we show how a long short-term memory (LSTM) model can improve the basic RNN model and the reasons we adopt LSTM in our system. Third, we explain the use of lexicon support for enhancing the performance of our recognition system.

5.1 Background: RNN

Neural networks can be divided into feedforward and recurrent. A feedforward neural network is also called MLP (Multi-Layer Perceptrons), where the input goes through all its hidden layers like a function. A recurrent neural network, which is inspired by the recurrent biological neural system, contains one or more cycles in the network topology. It can be applied to many usages: signal simulation, pattern modeling, classification, and so on. Depending on STM (Short-Term Memory) mechanism, a recurrent neural network (RNN) can remember the current states of the network then immediately inform other nodes. Therefore, RNN is suitable for identifying patterns of time series.

We usually implement the STM by a time delay training [**?**]. Fig. **??** shows a simple recurrent neural network. We denote the input units by $\vec{u}(n)$, and the internal units by $\vec{x}(n)$. According to the denotations, we can present the simplified RNN without output units' feedback by the following equations:

$$\vec{u}(n) = (u_1(n), \cdots, u_K(n)) \tag{5.1}$$

$$\vec{x}(n) = (x_1(n), \cdots, x_N(n)) \tag{5.2}$$

$$W^{\text{in}} = (w^{\text{in}}_{ij}), \text{for } i \in [1,K], j \in [1,N] \tag{5.3}$$

$$W = (w_{ij}), \text{for } i, j \in [1,N] \tag{5.4}$$

$$\vec{x}(n+1) = f(W^{\text{in}}\vec{u}(n) + W\vec{x}(n)) \tag{5.5}$$

In the above equations, n is the current time step, K and N denote the number of input and internal units, and W^{in}, W mean trainable weights. We can implement the W^{in} and W in the network as $K \times N, N \times N$ weight matrices, and a weight of 0 denotes no connection between two nodes. As using zero-delay training, the information produced by internal units instantly flows through each node in the network. When the delay value is d, the current output signal will become one part of the input signal after d time steps. Therefore, we can describe the model with the following formula, where $\vec{y}(n)$ denotes the output signal of a time step n:

$$\vec{x}(n+1) = f(W^{\text{in}}\vec{u}(n) + W\vec{x}(n) + W^{\text{delay}}\vec{y}(n-d)) \tag{5.6}$$

Depending on the implementation, the recurrent neural network model can tune the current state by the short-term memory. However, STM mechanism has several weak points. Jaeger's experiments [**?**] show that the longer the delay, the poor the training performance. Besides, the recurrent neural network still has the problems of gradient vanishing or exploding. These situations result from the propagation of trainable weights between the internal nodes. If the weight is less than 1, the final value approaches 0 after numerous multiplications. On the other hand, if the weight is greater than 1, then the final value approaches infinity. When the number of steps in training delay is greater than 10, the recurrent neural network may lose the meaning of memory because of the gradient problems. As a result, it is unclear whether the RNN can actually keep important information in memory.

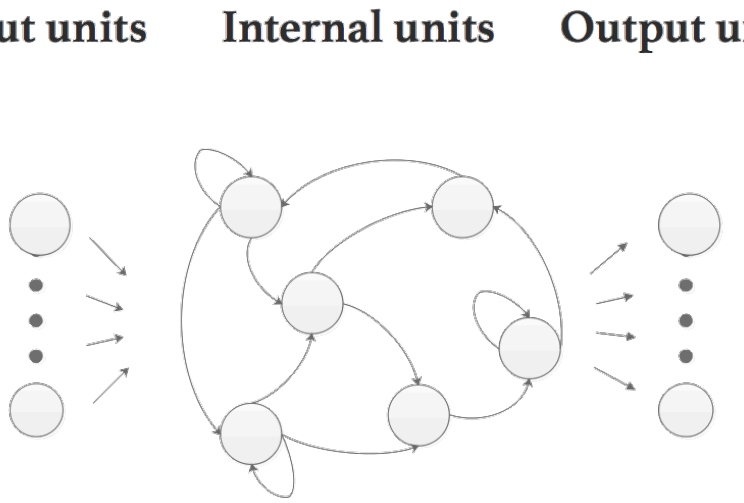


Figure 5.1: Simplified recurrent neural network

5.2 Long Short-Term Memory

There exist several solutions to solve the gradient vanishing and gradient exploding problem. One of them is called LSTM (Long Short-Term Memory). It improves on the basic RNN by the CEC (Constant Error Coursel), which is commonly called the cell *state*. The CEC protects LSTM network with the input gate and the output gate. First, we need to squash all inputs of the cell just like normalization. To limit the value of each weight, the gates are realized by a sigmoid function, which outputs a value in $[0, 1]$. As the CEC input gate output a 0 value, the input is not allowed to enter the LSTM cell. On the contrary, the cell output will not perturb the other cells in the network when the CEC output gate closes.

Relying on those gates, LSTM can store memory for a long time. However, the advantages of CEC mechanism are also its weak points. The long-term memory has a chance of leading to bad performance when LSTM analyzes a new sequence. One solution is to reset the cell state when encountering a new time series. To reset the cell state, we need to either notify the cell that the new data is arriving or train another cell to detect it. Either way of reset causes external overhead, and we need to spend more time on designing the network structure. Many solutions are listed in Gers et al. [**?**] to forget the out-of-date and needless memory. The most well-known and effective method among them is *forget gating*. It means that a forget gate is embedded in the LSTM cell for "learning to forget." Gers et al. also describe the implementation and modularization of three kinds of gates in details. For more natural understanding, we can define the LSTM cell with the following simplified mathematical model:

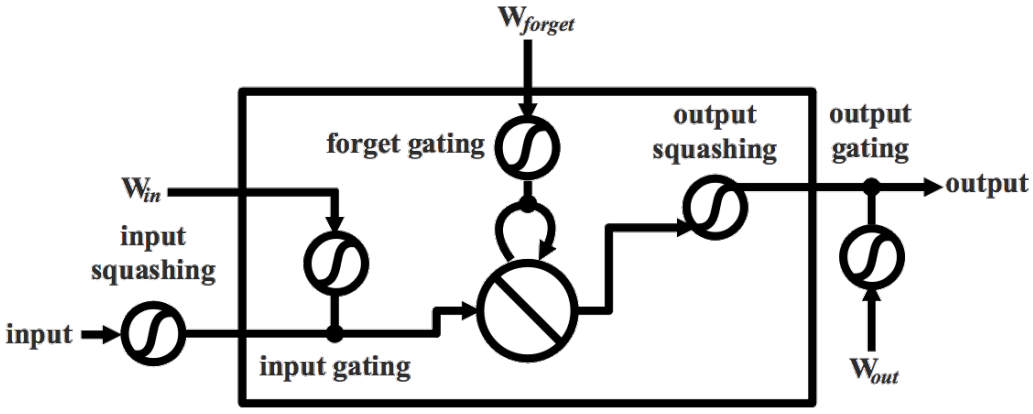


Figure 5.2: LSTM cell structure based on Equations (**??**), (**??**), and (**??**).

$$f(t) = \frac{1}{1+e^{-t}}, \qquad g(t) = \frac{4}{1+e^{-t}} - 2, \qquad h(t) = \frac{2}{1+e^{-t}} - 1 \tag{5.7}$$

$$cell_{\text{in}}(n) = g(w_c^{\text{in}} \vec{y}(n-1)), \qquad \vec{u}(n) = f(cell_{\text{in}}(n)), \qquad \vec{x}(n) = f(w_c^{\text{internal}} \vec{u}(n)) \tag{5.8}$$

$$cell_{\text{out}}(n) = h(w_c^{\text{out}} \vec{x}(n)), \qquad \vec{y}(n) = f(cell_{\text{out}}(n)) \tag{5.9}$$

Fig. **??** shows the figure of LSTM based on the above equations, where $f(t)$ denotes the gating function, $g(t), h(t)$ denote the squashing function, and $cell_{\text{in}}, cell_{\text{out}}$ denote the input and output of the cell, and w_cs denotes the trainable weights. At last, the LSTM not only keeps important information in memory but also forgets worthless factors. Due to segmentation challenges, few researchers have investigated the problem of on-pen handwritten word recognition. We employ a basic LSTM model to recognize our on-pen handwriting pattern.

5.3 Lexicon Support

In the literature on continuous handwritten word recognition, an OOV (out-of-vocabulary) detection is often used to calibrate the output of recognition models. Briefly, when the output word does not exist in the source dictionary, we may abandon the output or choose the most similar word from the dictionary. In this section, we demonstrate how we correct the misspelling of LSTM.

5.3.1 Bayes' Corrector

According to Bayes' Theorem, we can assume that each misclassification of LSTM model as an independent event. Therefore, we can describe the misspelling probability of the LSTM by the following equation:

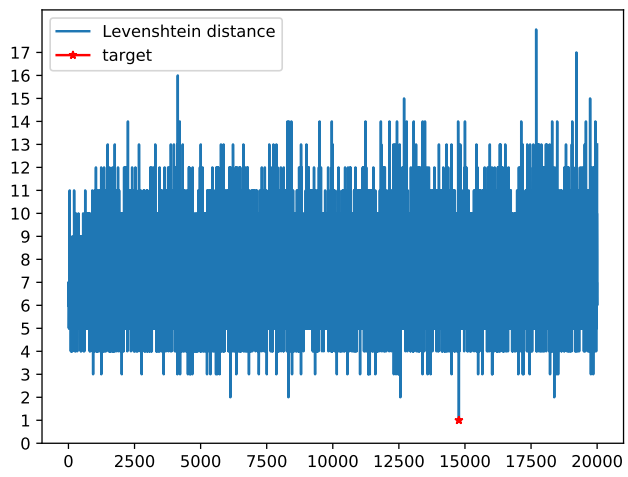
$$P(W) = \phi P(c_0)P(c_1)\cdots P(c_n), \quad W = c_0c_1\cdots c_n \tag{5.10}$$

P denotes the misspelling probability, W denotes the output of recognition model that consists of characters $\{c0, c1, \cdots, c_n\}$, and ϕ denotes the corrector's weighting value. Based on the assumption, we can calculate the probability distribution of misclassification of all characters. Our experiment shows that LSTM sometimes misidentifies characters $\{b, c, f, h, u, v\}$ as $\{p, e, l, n, a, r\}$, respectively, because of their similar motion (see in Fig. **??**). To address this problem, we specify the misspelling probability of those confusing characters and apply them on the corrector.

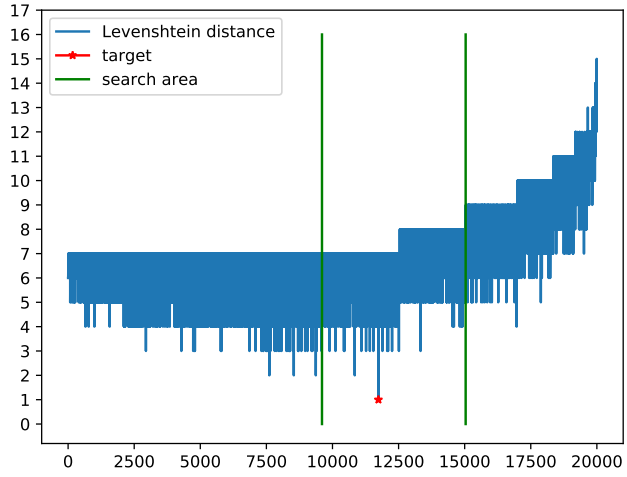
At the beginning of one correction procedure, the Bayes' corrector generates some possible strings that have an edit distance of less than 3 and removes those strings that do not exist in the dictionary. Additionally, the corrector only generates a substitution from $\{p, e, l, n, a, r\}$ to $\{b, c, f, h, u, v\}$ for accuracy and efficiency. In the next step, the corrector calculates the probability of each candidate and choose the one with the maximum probability as the final word. Because the Bayes' corrector does not generate all possible strings, it can calibrate LSTM outputs quickly. Unfortunately, the Bayes' corrector is not robust enough, because the LSTM classifier may produce outputs that the corrector has never generated at the beginning. For example, an output string has more than three wrong letters. In these conditions, we have to calibrate the output of LSTM further.

5.3.2 Weighted Levenshtein distance

When the Bayes' corrector fails to find the suitable candidate in the dictionary, we should try to find the most similar one in the whole dictionary. However, the "similarity" between two strings is difficult to define, and searching for the most similar word in the whole dictionary incurs much more computation cost than the Bayes' correction does. Even though Levenshtein distance, also called edit distance, is commonly used for calculating the similarity between two strings, it is not suitable for our application. For instance, the distance between "bat" and "pat" should be shorter than between "bat" and "cat," because for the LSTM classifier, the letters 'b' is more easily confused with 'p'



(a) search in a lexicographical ordered dictionary with 20k vocabulary size



(b) search in a length ordered dictionary with 20k vocabulary size

Figure 5.3: The Levenshtein distance distribution of the string "qaicker" in a 20k vocabulary size dictionary.

than with another letter. Therefore, the corresponding substitution weights of those confusing letters should be less than others. According to the experiment described in the Section **??**, we tune the best substitution weights of the letters $\{b,c,f,h,u,v\}$ to $\{p,e,l,n,a,r\}$. Besides, we also tune the other substitution weight, insertion weight, and deletion weight.

Levenshtein distance can be computed by dynamic programming with a time complexity of $O(mn)$, where m and n are the length of the predicted string and of the correct string, respectively. That is, the average time complexity of finding the most similar word in a lexicographically ordered dictionary is $O(N \times mn)$, where N denotes the vocabulary size. Among the distance distribution shown in Fig. **??**, we find that it is harder to search in a lexicographically-ordered dictionary. It is much easier to find the target in a length-ordered dictionary, where the words are reordered by their number of letters. We set the searching area between the length range of m to $m+1$ because the segmentation of our system sometimes fails to segment the motion of two consecutive letters. Depending on the mechanism, we can achieve an average 7.5 times speedup compared to basic searching.

Chapter 6

Evaluation

This chapter first describes our experimental setup and method for data collection. Second, we evaluate the experimental results in terms of segmentation robustness, accuracy of using different features, and improvement after lexicon support. Finally, we summarize the overall performance of our system.

6.1 Experimental Setup

6.1.1 Data Acquisition

As we illustrate in the beginning of this thesis, we want to develop a real-time handwritten word recognition system without creating a custom inertial pen. For the purpose of feasibility study in this thesis, we collected our dataset from a single writer rather than multiple users. One major assumption with our recognition algorithm for now is that the writer must hold the pen such that the sensor is kept in a fixed orientation. Specifically, we mount the sensor on the left side of the pen as shown in Fig. **??**. This is because our system does not transform the sensor's coordinate from local to global, but a future implementation may lift this assumption by using the magnetometer.

The words that we write in the experiments are generated from a 20000-word dictionary. Totally 3761 seconds of raw motion data is added to our dataset for 749 words. Besides, 538 among them belong to the training set, 86 the validation set, and 125 the test set. Each handwritten sequence is segmented from word-level to letter-level, and we normalize them to a length of 100 by linear interpolation. In other words, the data for each written letter is encoded as an $N \times 100$ matrix, where N is the number of features. Section **??** shows a comparison of different feature selections. Furthermore,

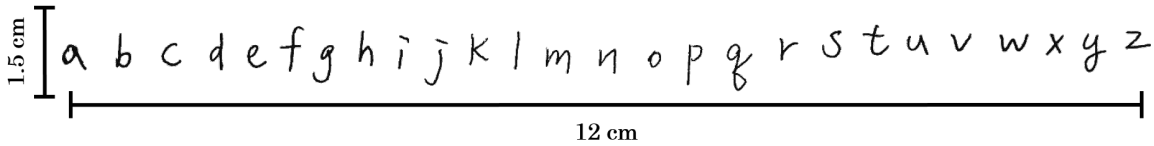


Figure 6.1: All characters handwriting template

we collected the validation data and test data one week and one month, respectively, after having collected the training data. Therefore, the LSTM model can not only validate itself without future knowledge but also adapt itself to changes in the user's new writing habits. After all, we divide the 26 letters of the English alphabet into 26 classes. The handwriting template of each alphabet, which roughly in size of 5mm×5mm, is shown in Fig. **??**.

6.1.2 System environment

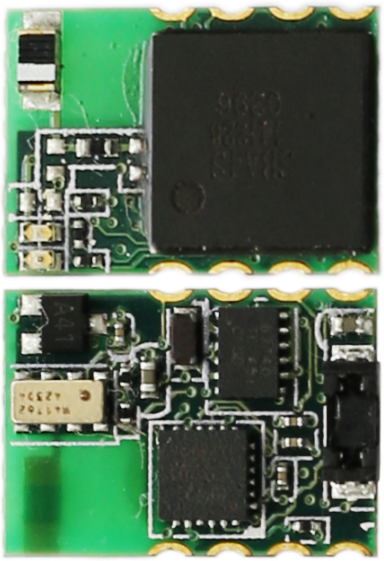
We run our training environment on a computer with an AMD R51600 6-core 16-thread processor and 16 GB memory. We train the basic LSTM model using the TensorFlow Library with support for an NVidia GTX1080 Ti GPU with 11 GB memory. For practical considerations, we build the on-pen handwritten word recognition system in an Ubuntu 16.04 environment with two cores and 2 GB memory. The LSTM model employed in the system is frozen [**?**] for size reduction and the speed of recognition. Additionally, the parameters of both Bayes' corrector and weighted Levenshtein distance searching are tuned in the testing environment by random seeding. We select the best sets of parameters after seeding 100 times.

6.2 Experimental Results

6.2.1 Segmentation Robustness

Even though our proposed method can calibrate the result of classifier output, the corrector cannot recover from misspelling as the output has too many errors. Therefore, when the segmentation fails, the LSTM classifier also loses effectiveness. Because of the sensors' limitations, signals observed from on-pen device become unsegmentable if the writing speed is too fast. Therefore, currently we do not support recognition of cursive letters. Under those constraints, we evaluate our segmentation method with Word Segmentation Rate (WSR) and Segmentation Accuracy (SA), which defined by

(a) The way an user hold the pen and fix the orientation of MPU9250 motion sensor equiped on the pen



(b) EcoMini

Figure 6.2: Hardware setup

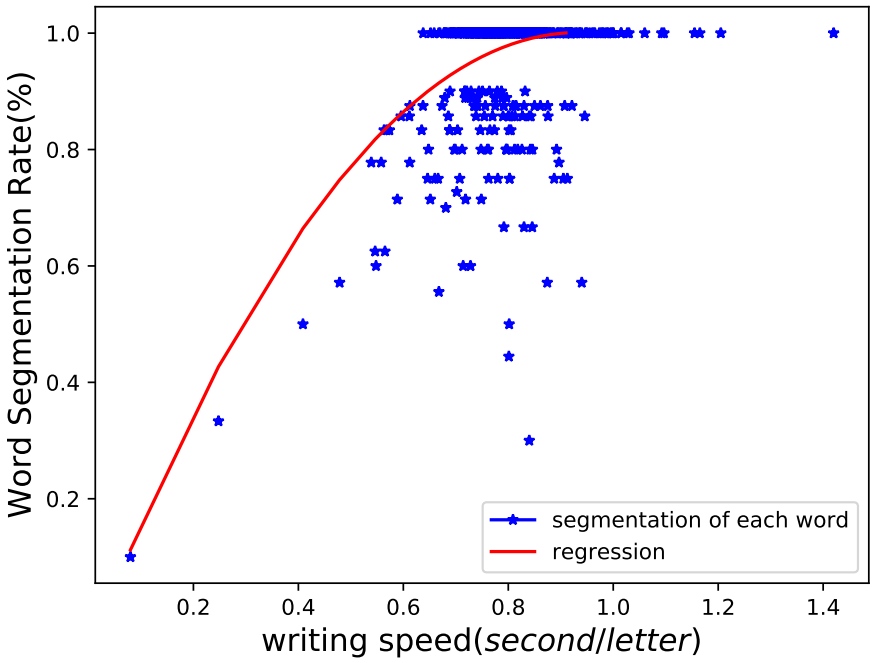


Figure 6.3: The word segmentation rate distribution with different writing speed

the following equations:

$$WSR = 1 - \frac{|L - L_{\text{pred}}|}{L} \tag{6.1}$$

$$SA = \frac{N_{\text{correct}}}{N_{\text{dataset}}} \tag{6.2}$$

We denote the length of ground truth string by L, the length of the predicted string by L_{pred}, the number of words in the dataset by N_{dataset}, and the amount of correct segmentation word by N_{correct}. A correct segmentation means that the number of output segments is equal to the length of the ground-truth string. We analyze the WSR performance on the data with different writing speeds in the range of 0.1 to 1.4 seconds/word. Fig. **??** shows our segmentation approaches a 0.832 SA in our dataset. Even if it cannot segment the word correctly, the method usually misses only one letter, and the word can almost always be recovered by the corrector.

6.2.2 Accuarcy of Using Different Features

We also want to know what the best combination of the motion data features is. First, for the LSTM model in our system, the learning rate is set to 0.001, and number of neurons is set to 64. We evaluate the accuracy and error rate of each experiment after training the model for 2500 epochs, which takes roughly 20 to 30 minutes. All experiments are judged on the test set, which is collected during the one-month training period. Next, we generate 14 features from the writing data, which are shown in Table **??**. Additionally, for easier reading, the expression $\{A, G\}$ denotes using all features from the accelerometer and gyroscope.

Among all configurations we have evaluated, we find that the full setup is not the best, and in fact it is even worse than several configurations. The best setting is using all features, except for z-axis accelerometer signals. We apply it to our system and reach a 76.3% accuracy for character recognition.

6.2.3 Improvement After Lexicon Support

We test four kinds of configuration of lexicon support: without corrector, with basic Bayes' corrector, with only Weighted Levenshtein Distance Searching (WLDS), and with Bayes' corrector support by WLDS, in this experiment. WER and CER are the main factors by which we evaluate the system

Table 6.1: Analysis of feature selection

(a) 14 features we generate and their symbols

feature	symbol
x-axis accelerometer	a_x
y-axis accelerometer	a_y
z-axis accelerometer	a_z
x-axis gyroscope (angular velocity)	g_x
y-axis gyroscope (angular velocity)	g_y
z-axis gyroscope (angular velocity)	g_z
filterd x-axis accelerometer	a_x^f
filterd y-axis accelerometer	a_y^f
filterd z-axis accelerometer	a_z^f
filterd x-axis gyroscope (angular velocity)	g_x^f
filterd y-axis gyroscope (angular velocity)	g_y^f
filterd z-axis gyroscope (angular velocity)	g_z^f
resultant gyroscope	g_r
filterd resultant gyroscope	g_r^f

(b) feature selection performance

feature selection	accuracy
full configuration	0.688
$a_x, a_y, a_x^f, a_y^f, G, G^f, g_r, g_r^f$	**0.763**
$a_x, a_z, a_x^f, a_z^f, G, G^f, g_r, g_r^f$	0.708
$a_y, a_z, a_y^f, a_z^f, G, G^f, g_r, g_r^f$	0.692
$a_x, a_x^f, G, G^f, g_r, g_r^f$	0.609
$A, A^f, g_x, g_y, g_x^f, g_y^f, g_r, g_r^f$	0.720
$A, A^f, g_y, g_z, g_y^f, g_z^f, g_r, g_r^f$	0.615
$A, A^f, g_x, g_z, g_x^f, g_z^f, g_r, g_r^f$	0.663
$A, A^f, g_x, g_x^f, g_r, g_r^f$	0.471

Table 6.2: Corrector performance comparison

Corrector	WER	CER	Correction time (second/word)
without corrector	0.776	0.237	None
with basic Bayes' corrector	0.504	0.452	**0.032**
with only WLDS	0.328	0.144	0.484
with Bayes' corrector support by WLDS	**0.264**	**0.120**	0.232

performance and are defined by the following equations:

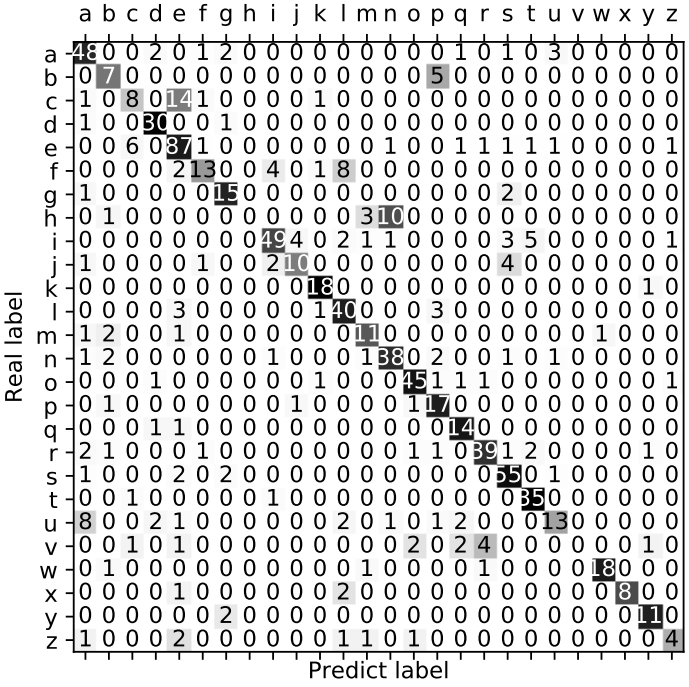
$$WER = \frac{N_{\text{err}}^w}{N^w} \tag{6.3}$$

$$CER = \frac{\sum^{N^w} d_{\text{pred, real}}}{N^l} \tag{6.4}$$

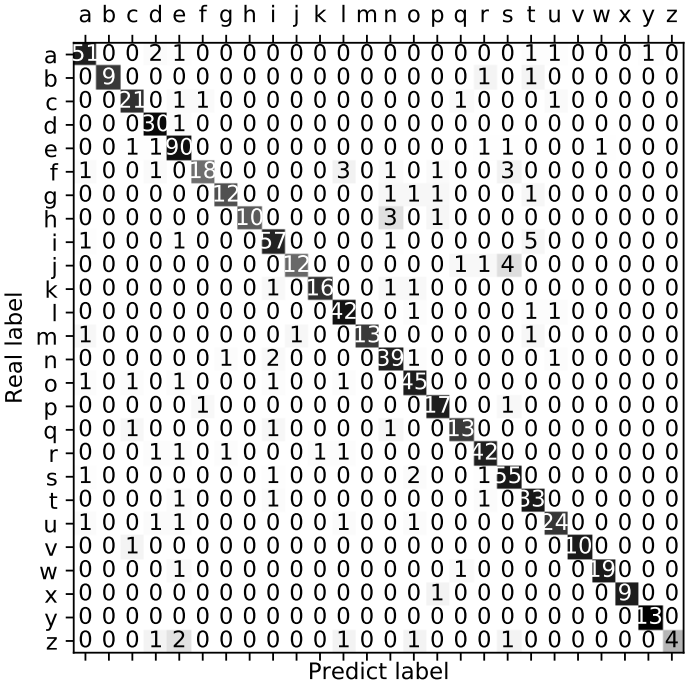
We denote the number of words and letters by N^w and N^l, the number of false predicted words by N_{err}^w, and the standard Levenshtein distance between prediction and ground truth by $d_{\text{pred,real}}$. Besides WER and CER, we also validate the correction time per word because spending a long time on calibration leads to a negative user experience. In Table **??**, we can see that the Bayes' corrector support by WLDS can improve the result of LSTM classifier from 0.776 WER to 0.264 WER and reduce the average correction time to half of the corrector with only WLDS.

6.3 Summary

We have evaluated the writing speed and accuracy of our continuous handwriting recognition system. With the experiment on segmentation robustness (Section **??**), we show that our threshold-based segmentation can handle a writing speed up to 0.7 second/letter. In most case, the corrector can revise the output of LSTM. In the experiment on feature selection (Section **??**), we find that feeding more features to the machine-learning classifier does not always lead to better accuracy. The features we should choose depend on the mounting position of the IMU. According to the comparison ofTable **??** (b), it also indicates that gyroscope is more suitable than accelerometer for recognizing small but fast motions. In the experiment on lexicon support (Section **??**), we measure how our Bayes' corrector supported by WLDS improve the result of recognition. Without using any corrector, the difficulty of continuous identification brings about a high error rate of LSTM. The problem results from the sensor limitation and the variation of writing habits. However, after calibration, we decrease the WER from 0.776 to 0.264.



(a) The result of LSTM classifier



(b) The result calibrated by lexicon support

Figure 6.4: Confusion matrix on future test set

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this thesis, we propose a sensing system that can be clipped onto any pen and transmits the motion data to a computer to recognize continuous handwriting. For flexibility, our hardware allows writers to use any pen as a text-input interface without requiring an extra writing pad. Our recognition technique first performs segmentation followed by a word-level correction that such that it can recognize the on-pen handwriting from letter-level to word-level.

The experimental results show that the new system including both the hardware and the recognition algorithm is efficient and reliable. The LSTM parameterization is determined by the size of features and hidden layers. Therefore, increasing the number of classes to recognize does not impact the cost of training and identifying. For reliability, first, our segmentation method can handle a writing speed up to 0.7 second/letter. Second, with the lexicon support, our LSTM recognizer achieve a 26.4% WER and 12% CER.

Our system currently uses only 6-DoF of the 9-DoF inertial sensor and are limited to local coordinates. If we transform the orientation from local to global, we can train a user-independent model so that the applications will become more general. For scalability, we likely need to identify more labels, such as uppercase letters.

7.2 Future Work

Currently, our system only supports user-dependent case. To be more practical, a global coordinate transformation of the motion sensor is needed. Next, the current segmentation algorithm cannot handle a writing speed faster than 2 words/second and is a bottleneck of recognition accuracy. To solve the problem, we have two solutions to enhance the system performance. The first is to apply the seq2seq model on our system. It can automatically segment continuous data with the attention mechanism. The second is to use a more complex model for classification, such as Bidirectional LSTM or multi-level LSTM. Through a more powerful classifier, the system has a better chance of achieving higher accuracy.