筑波大学大学院博士課程

理工情報生命学術院

システム情報工学研究群修士論文

ソフトウェアキーボードの動的スタイル調整手法

LI MIAO

修士（工学）

（情報理工学位プログラム）

指導教員　高橋 伸

2023 年 3 月

概要

本研究は QWERTY 配列キーボードに基づきキーのスタイルを動的に変更するソフトウェアキーボードを提案する．QWERTY 配列キーボードはパソコンなどのデスクトップデバイスで非常に便利で使いやすい．しかし，スマートフォンなどのモバイルデバイスでは必ずしもうまく機能するとは限らない．小さい画面では，キーが密集しているため，指が大きすぎたり，キーのサイズが小さすぎたりすることによる入力効率が低くなる可能性がある．そこで本研究では，よく知られている QWERTY 配列キーボードに基づき，キーのスタイルを動的に調整する新しいソフトウェアキーボードを提案した．ユーザーが入力しようとするキーを予測し，それらのキーのサイズや色などスタイルを変更することで，誤入力を減少し，スマートフォンでのテキスト入力効率を向上させることを目的とする．本研究では，キーのスタイルを動的に調整する方法を提案した．提案手法により QWERTY 配列キーボードの設計と実装を説明した．また，提案したキーボードにおいてキーのスタイルが入力効率に与える影響を調査と考察した．実験の結果としては，キーを適度に大きくすると入力速度が上がる傾向があることが分かった．キーの色を変え, キーが目立ちになりすぎると，ユーザーの入力への集中を妨げ，入力速度に影響を与え，入力速度が下がる傾向が見られた．

目 次

図目次

第1章　はじめに

1.1　背景

日常生活では，スマートフォンは非常に重要なツールである．その中で，キーボードは人間とコンピューターやスマートフォンなどのデバイスとのインタラクションにおいて非常に重要な役割を果たしており，ユーザーにテキストの入力を便利にすることで，デバイスの使用の効率と便利性を向上させる．物理キーボードで入力する場合，ユーザーは長い間タッチタイピングを使用しており，入力時に頻繁にキーを見ずに入力することができる．QWERTY 配列キーボードはデスクトップでは非常に適した構成で，両手で操作するにはとても効率的で操作が行いやすいが，スマートフォンやスマートウォッチなどのモバイルデバイスでは必ずしもうまく機能するとは限らない．このような問題は小型の QWERTY 配列キーボードに関連している．小さなスクリーンでは，キーが密集した QWERTY 配列キーボードは,「ファットフィンガー」[1]（図 1.1）の問題を抱える．ユーザーがスマートフォンでテキスト入力を行う際，画面上のソフトウェアキーボードを指でタップするため，指の大きさにより，隣接する複数のキーを間違って押してしまう現象が起こる．そのため，テキスト入力時にエラーが発生しやすく，入力効率が悪くなることがある．つまり，テキスト入力はキーのサイズが妨げになっているのである．同時に,Rabin らの研究 [2] によれば, ソフトキーボードには触覚フィードバックがないため, ユーザがキーに触れたり，クリックしたり，キーから離れたりしたことを知ることができない．キーの位置を繰り返し確認しなければ, スマートフォンでの入力効率が低くなる可能性がある．

1.2　目的とアプローチ

モバイルデバイスにて QWERTY 配列キーボードを使用するには依然と多くの問題が残っているため，本研究では，入力効率を向上させるソフトウェアキーボードの設計を検討する．標準的な QWERTY 配列キー配置を基に，キーのスタイルを動的に変化させる新しいソフトウェアキーボードを提案する．このキーボードは，QWERTY レイアウトを保ちつつ，ユーザが入力したいキーを予測し，キーのスタイル（色やサイズなど）を動的に変化させる．これにより，ユーザーがキーの位置を確認しやすくなる．また「ファットフィンガー」[1] の問題が軽減することで入力効率が向上すると期待される．

図 1.1: 「ファットフィンガー」のイメージ．小さい画面上のキーに対して指のサイズが大きいため，ユーザーがキーを押し間違えやすい

1.3 本研究の貢献

本研究の貢献は以下の通りである．

- QWERTY 配列キーボードに基づいくキーのスタイルの調整方法を提案した．
- キーのスタイルを動的に調整する方法により QWERTY 配列のキーボードを実装した．
- キーのスタイルが入力効率に与える影響を調査した．

1.4 本論文の構成

本論文の構成は以下の通りである．第 1 章では，本研究の背景，目的とアプローチ，および本研究の貢献を説明する．第 2 章では，QWERTY キーボードを基づき入力方法についての関連研究を述べる．第 3 章では，提案手法の全体像および QWERTY キーボードを用いた提案手法について述べる．第 4 章では，提案手法によるソフトウェアキーボードの設計および実装を述べる．第 5 章では，実験システムおよび実験結果の処理方法を示す．第 6 章では，予備実験および評価実験の内容と結果を説明する. 第 7 章では，提案したキーボードの実験結果について議論を述べる. 第 8 章では，本研究のまとめを述べる．

第2章　関連研究

本章では主に三つの方面から関連研究を述べる．まずは，ソフトウェアキーボードの入力手法についての研究を説明する．次は，QWERTY 配列キーボードに基づいての手法についての研究を紹介する．最後にスマートフォン以外に QWERTY 配列キーボードに基づいての手法を述べる．

2.1　スマートフォンのソフトウェアキーボードの入力手法

スマートフォンで入力効率を向上させるために，さまざまな入力方法が提案されている．SwiftKey キーボード [3] では，SwiftKeyFlow と呼ばれるジェスチャベースの入力方法を使用しており，ユーザーはキーボードで指をスワイプして文字を入力できる．指でキーボード上をスワイプすることで，予測された単語が予測バーに現れ，ユーザーはその単語をクリックして入力することができる．Thumbly キーボード [4] は，片手で使用するために設計されたキーボードであり，特に親指で使用するために設計されている．キーは人間工学を基に設計されており，親指を左右に動かすことで自然にキーボード入力ができる．削除，リターン，大文字などの機能を改善するために，さまざまなスワイプジェスチャが作られた．BlackBerry Z10 のキーボードは [5]，標準の QWERTY 配列キーボードに基づいて開発されている．キーがクリックされると，予測された単語が予測された次の文字の上に表示され，ユーザーはその予測された次の文字を押しながら上にスライドする事で単語を自動的に入力する．TouchPal T+ キーボードは [6]，標準的な QWERTY 配列キーボードの 2 つのアルファベットを 1 つのキーに入れ，14 単位の QWERTY キーボードとして使用している．これにより，指が触れられる領域が拡大され，入力エラー率を低減することで操作しやすくなる．

2.2　QWERTY 配列キーボードに基づくスマートフォンの適応型ソフトウェアキーボード

Sakkos ら [7] は QWERTY 配列キーボードに基づく，モバイルデバイス用の適応型ソフトウェアキーボードを提案した．このキーボードは，ユーザーの辞書や習慣を学習し，次のアルファベットを予測する．予測された文字セットにより，キーの位置を変えずに，予測セットに含まれないアルファベットのキーのサイズが小さくされる．また，キーを小さくすることで，誤入力を防止するための空白を大きくしている．bhatti らの研究 [8] では，キーの色を

変更し，キーの幅を増やすことができる QWERTY 配列ソフトウェアキーボードを提案した．このキーボードでは，ユーザーがキーをクリックするとこのキーボードは，最近押されたアルファベットを基に次に入力する可能性が高いアルファベットを予測し，そのアルファベットに対応するキーの幅を通常のキーの 2 倍に拡大し，とその枠の色を変更する．Rodrigues ら [9] は，2 つの QWERTY 配列キーボードを提案した．一つ目のバリエーション（カラーバリエーション）は，予測アルゴリズムにより，現在入力した単語の次に最も可能性の高いアルファベットを強調表示するものである．2 つ目のバリエーション（単語予測バリエーション）は，ユーザーがテキストを入力する際にキーボードの上に候補単語のリストが表示され，ユーザーは単語リスト内の単語をクリックすることで入力文字数を削減することができる．iPhone のキーボード [10] は，キーを入力するときに各キーのヒットエリアが変化する． ユーザーがアルファベットを入力する時に，iOS が次のアルファベットを予測し，キーごとに重み付けをする．そして，次に入力する可能性の高いキーのヒットエリアが広くなり，誤入力が起きにくくしている．

2.3 スマートフォン以外に QWERTY 配列キーボードに基づいての手法

Dube ら [11] は，仮想現実において Qwerty 配列キーボードにテキスト入力を検討した．この研究 [11] では，Qwerty 配列キーボードに基づいてキーの形とサイズを変更した仮想キーボードを提案し，異なるキーの形やサイズがテキスト入力性能やユーザーエクスペリエンスに与える影響を調査した．その結果，キーの形はテキスト入力速度に影響を与え，サイズは精度に影響を与えることが示した．さらに，両者ともにユーザーエクスペリエンスに影響を与えることが分かった．Rodrigues ら [12] は，タブレット端末上での QWERTY 配列キーボードに基づく入力手法を提案した．提案手法としては，次の 4 つの最も可能性の高いキーの幅や色を変えることである．この手法による入力速度は，標準的な QWERTY キーボードより低かったと報告された．Gunawardana ら [13] は，キーのターゲットに向けてキーのサイズを動的に調整するアルゴリズムを提案し，タブレットに QWERTY 配列キーボードを開発した．このキーボードは，予測方法により，入力確率に応じてキーのターゲットエリアを動的に調整する．

第3章　提案手法

本研究では，ユーザーが入力する内容を基に，ユーザーが次に入力する可能性が高いアルファベットを予測し，それらのアルファベットをより容易に入力できるようにキーのスタイルを動的に変更する手法を提案する．

本章では，まずは提案手法の全体像を紹介する．次に，キーのスタイルを変更する手法を最初に紹介する．そして，単語を予測する方法を紹介する．最後に，単語を予測する方法に基づいて，アルファベットを予測する方法を紹介する．これらの手法を用いることで，入力効率が向上することが期待される．

3.1　提案手法の全体像

本研究で提案するソフトウェアキーボードの全体像（図 3.1）は，次に説明する 3 つのステップで構成される．

- 単語予測ステップ：すでに入力された内容により，ユーザーが入力する可能性が高い単語の集合を予測する．
- アルファベット予測ステップ：単語予測ステップで予測された単語の集合に基づき，次に入力される可能性が高いアルファベットを予測する．
- キーのスタイル調整ステップ：アルファベット予測ステップで計算されたアルファベットに対応するキーのスタイルを変更し，キーボードのレイアウトが崩れないように，変更されたキーのスタイルに従って，他のキーのスタイルと位置を同時に調整する．

図 3.2 は，実際にキーボードを使う際のイメージを表している．例えば，「I will t」が入力されると，次に入力可能な単語「take」，「tell」，「test」，「think」「try」などがある．そして，その次に入力する可能性のあるキー「a」，「e」，「h」，「r」のスタイルが変わることが示されている．

3.2　動的なキーのスタイルの変更手法

本研究では，テキスト入力をよりスムーズに行えるよう，ユーザーが次に入力する可能性が高い文字を予測し，それらの文字をより容易に入力できるようにキーのスタイルを動的に

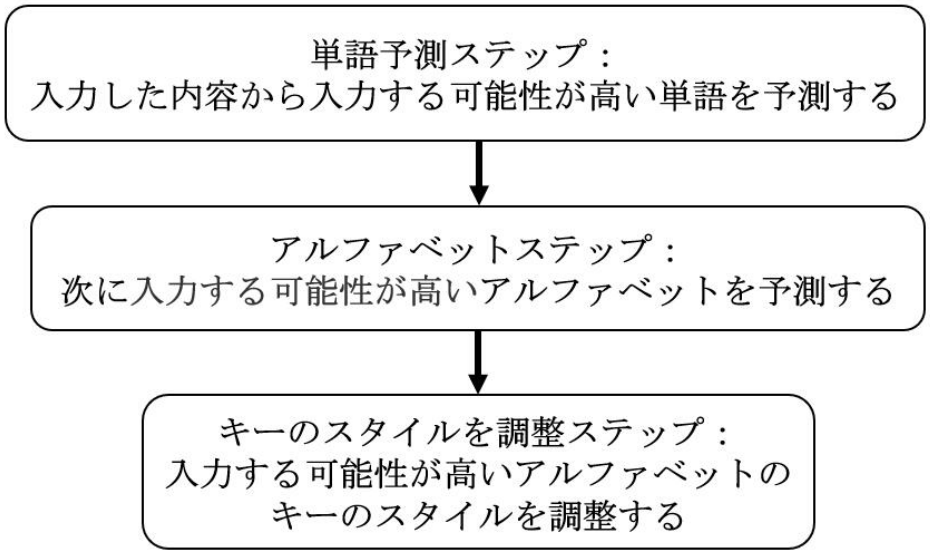


図 3.1: 提案手法の全体像

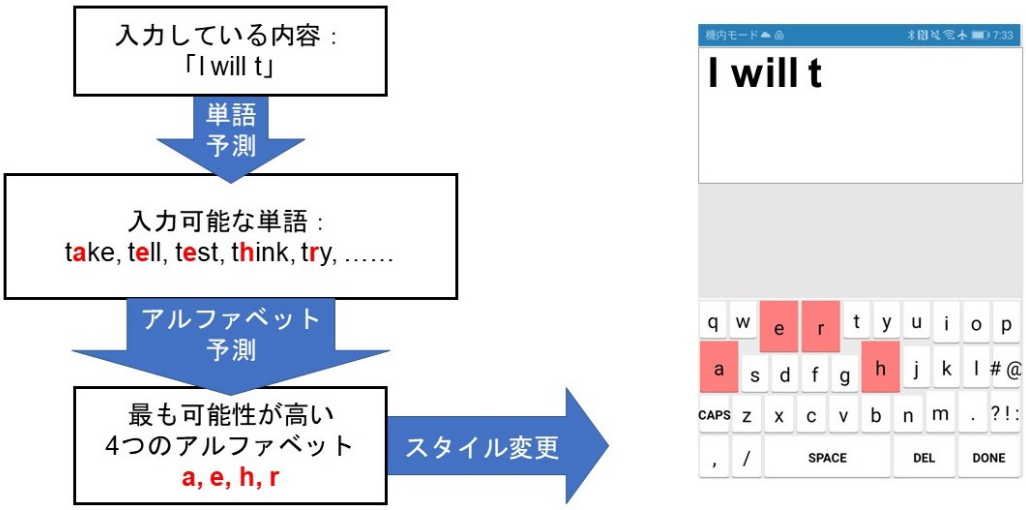


図 3.2: 実際にキーボードを使うイメージ

変更する手法を提案する．この節では，本研究で提案する 3 種類のキーのスタイルの変更手法を説明する．

以下は，キーのスタイルを変更する手法の 3 種類である:

- キーのサイズを大きくする: この手法は，図 3.3 のように，キーのサイズを大きくすることで，キーをより押しやすくすることを目標とする．これは，「ファットフィンガー」[1] の問題を避けることで，入力効率を向上させることを期待する．
- キーの色を変更する: この手法は，図 3.4 のように，キーの色を変えることで，キーをより目立たせ，ユーザーがキーの位置を確認する時間を短縮することを期待する．これにより，入力速度が向上することが期待される．
- キーのサイズを大きくしたと同時に，キーの色を変更する: この手法は，図 3.5 のように，前の二つの手法を組み合わせて，キーをより押しやすくし，またより目立たせることで，入力効率を向上させることを期待する．さらに，ユーザーがキーの位置を確認する時間を短縮することを期待する．この手法を用いることで，さらに入力効率が向上することが期待される．

図 3.3: キーのサイズを大きくする

3.3 単語予測の手法

本研究での提案手法は，ユーザーが次に入力する可能性が高い単語を予測する機能を含む必要がある．これは，現在入力している内容を基に，ユーザーが入力しようとしている可能性のある単語を予測することを指す．単語の予測は本研究の主要な焦点ではないが，提案するキーボードシステムには不可欠の要素である．提案するキーボードシステムは，既に入力された文字から入力する可能性のある単語を予測し，その中から，ユーザーが入力する可能性があるアルファベットを選択する．

図 3.4: キーの色を変更する

図 3.5: キーのサイズを大きくしたと同時に，キーの色を変更する

入力過程での頻繁な単語予測のために，単語予測の処理速度の確保が必要である．そのため，本研究では n-gram 言語モデル [14] を用いた単語予測を採用する．n-gram モデルは，自然言語処理（Natural Language Processing; NLP）においてよく用いられるモデルであり，単語列の次の単語の出現確率を予測することができる．n-gram は，自然言語テキストを n 個の連続単語からなるシーケンスに分割し，テキストの構造，意味，パターンを調査することを意味する．n-gram 言語モデルは，最近の n 単語から任意の単語の出現確率を判定するため，計算量が少なく，計算時間が低いので，単語予測の処理速度の確保することにに役に立つ [14].

n-gram において，n が大きすぎると過学習の問題が発生し，訓練セットに対する予測の精度は高くなるが，実際の使用においての精度が低下する可能性がある．また，n-gram において，n が小さすぎると，n-gram が含む情報量が少なく，モデルの表現力が弱くなる．n が小さい場合，モデルは十分に複雑ではなく，訓練セットを良くフィットできない可能性があり，モデルの精度が低下する．本研究では，3-gram 言語モデルを用いて単語予測を実装することで，十分な予測の精度を確保しつつ，過学習の問題を回避すると考えられる．

3.4 アルファベット予測の手法

本研究では，本研究で使う単語予測を基にした簡単なアルファベット予測方法を提案した．アルファベット予測は，単語予測が終わる時点で, その結果から最も入力される可能性があるアルファベットを特定すること．

具体的には，次のような手順を踏む．今後最も入力される可能性がある n 文字を予測したいとする．まず，単語予測モジュールは (W, p) というペアのリストを返す．ここで，W は単語を表し，p はその単語が入力される確率を表す $(0 < p < 1)$．また，26 個のアルファベットの初期の確率はすべて 0 である．次に，各単語 W の中で，次に入力されるアルファベットの位置にあるアルファベットの確率を，その単語が出る確率 p で増加させる．最後に，確率が最大の n 個のアルファベットを予測結果とする．アルファベット予測の詳細は 4.4 節に説明する．

例えば，最も入力される可能性がある一つのアルファベットを予測する目標として，3 番目のアルファベットが入力されると仮定する. アルファベット予測するまえに，まず単語予測が行い, 4 つの単語 teacher，test，testing，tech が予測され, それぞれの入力確率は 0.05，0.04，0.02，0.01 である．提案したアルファベット予測により，アルファベット a，s，c の確率はそれぞれ 0.05，0.06，0.01 であることが得られる．ここで，確率が最大の一つのアルファベット s が予測の結果となる．

注意すべきは，上記の例では，teacher の確率が最大の 0.05 であるにもかかわらず，teacher の第 3 のアルファベット a の入力確率が最大ではないことである．これは，アルファベット s が 2 つの単語 test，testing に現れるためである．これら 2 つの単語の確率の合計が 0.06 であり，teacher の 0.05 よりも大きいため，s の確率が最大となる．これは，単語予測を単独で使用すると，確率が最大のアルファベットを得ることができない可能性があることを意味する．私たちが提案するアルファベット予測手法は，単語予測を基にして，さらに入力確率が最大

のアルファベットを判断することができる.

第4章　キーボードの実装

本章では，本研究で提案する手法に基づくキーボードの実装の詳細を紹介する．

本章の各節については，まず，システムの実装環境を説明する．次に，キーボードのキースタイルを変更する実装を説明する．次に，単語予測機能を実装する方法を紹介する．最後に，アルファベット予測機能の実装を紹介する．

4.1　システムの実装環境

このシステムの実装において，キーボードプログラムでは Android SDK29 を使用し，Java 言語を用いて開発し，Android Studio 3.2 という IDE を使用した．また，スマートフォンとして Huawei mate10 pro を使用した．単語予測プログラムの実装において，Python 言語を使用し，Pressagio という n-gram を使ったオープンソースツールキットを使用した．また，単語予測プログラムとキーボードプログラムを通信するために，gRPC というフレームワークを使用し，protocol buffer3 というシリアルプロトコルを使用し，Python 言語を使用してプログラムを開発した．

4.2　キーのスタイルが動的に変更できるキーボードの実装

本節では，キーボードのスタイルを動的に変更する方法を紹介する．Android SDK のキーボード API [15] を使用して，スタイルを動的に変更できるキーボードを作成する． Android SDK のキーボード API の機能が限定されており，必要な効果を実現する際に，3 つの課題がある．

- 実行時にキーサイズを動的に調整すること：Android SDK のキーボード API は，レイアウトファイル内で各キーのサイズを指定することができるが，実行時に任意のキーのサイズを直接変更することはできない．また，この API は，キーサイズを調整する方法を提供していない．しかし，Android SDK のキーボード API がオープンソースであるため，直接ソースコードを修正し，キーサイズを変更するコードを追加することができる．
- キーサイズを変更した後に，キーボードのレイアウトが崩れないようにすること：画面の使用可能なスペースが限られているため，いくつかのキーのサイズを変更すると，他のキーのサイズを調整して，サイズを変更していないキーがサイズを変更したキーに

よって覆われないようにし，Qwerty キーボードの完全なレイアウトを保持する必要がある．

- 実行時に動的にキーの色を変更すること：Android SDK のキーボード API は，レイアウトファイル内で各キーのテクスチャ情報を指定することで，キーの色を指定することができるが，実行時に任意のキーの色を調整する方法はない．この問題を解決するには，ソースコードを修正する必要がある．

上述の問題を解決し，提案したキーボードを実装する前に，Android SDK キーボード API の構成を簡単に紹介する．キーボード API は，主に「KeyboardView」と「Keyboard」という 2 つのコンポーネントで構成されている．Keyboard は，Android オペレーティングシステム内で，ソフトキーボードのレイアウトとキー情報を表すクラスである．それには，キーボードのレイアウトやキー情報が含まれており，キーの種類，テキスト内容，アイコン，位置情報などが含まれる．KeyboardView は，キーボードを表示するためのビュー（View）クラスであり，Keyboard のレイアウトとキー情報を画面上に表示することができる．さらに，KeyboardView にリスナーを設定することで，ユーザーがキーを押したときにある動作で反応させることできるようになる（例えば，キーのスタイルの変更はリスナーで行うことができる）．

次の二つの節では，具体的に提案したキーボードを実装する方法を紹介する．

4.2.1 キーのサイズの変更

まず，実行中に動的にキーのサイズを調整するためには，Keyboard クラスのコードを調整する．具体的には，Keyboard クラスにいくつかの関数を追加し，キーボードが実行中にも任意のキーのサイズ情報を調整できるようにした．このようにすることで，特定のキーのサイズを変更する場合には，新しく追加したこの関数を呼び出すだけで済むようになる．次に，KeyboardView クラスのコードも調整し, 変わった Keyboard 情報を再ロードできる関数を追加する．このようにすることで, Keyboard 中のキーのサイズ情報が変更される場合, KeyboardView クラスの再ロード関数を呼び出すことで，更新されたキーボードを再び画面に表示することができる．これで，必要な機能を実現することができる．

そして，キーのサイズが変わってもキーボードのレイアウトが崩れないように，キーのサイズを調整するアルゴリズムを述べる．アルゴリズム 1 はキーを拡大しながら, キーの位置を調整するアルゴリズムとして，レイアウトが崩れないように，指定された集合 K 内のキーを指定された幅 w と高さ h で拡大するものである．アルゴリズム 1 は，K 内のそれぞれのキー k について，次の操作を行う．

まず，k を含む行の他のキーを取得する．これを K' とする．次に，K' 内のそれぞれのキー k' について，$k' \neq k$ の場合，k' の幅を $\frac{w}{|K'|-1}$ で減らす．次に，k の幅を w で増やす．そして，各キーが被らないように，k と K' を移動する (MoveCol 関数を呼び出す).

その後，k を含む列の他のキーを取得する．これを K' とする．次に，K' 内のそれぞれのキー k' について，$k' \neq k$ の場合，k' の高さを $\frac{h}{2}$ で減らす．次に，k の高さを h で増やす．最

Algorithm 1 ScaleKey

Require: K: Set of keys to be enlarged, w: Width to be enlarged, h: Height to be enlarged

function SCALEKEY(K, w, h)
    for each key $k \in K$ **do**
        $K' \leftarrow$ keys in the same row as k
        for each key $k' \in K'$ **do**
            if $k' \neq k$ **then**
                Decrease width of k' by $w/(|K'| - 1)$
        Increase width of k by w
        MOVECOL(k, K')
        $K' \leftarrow$ keys in the same column as k
        for each key $k' \in K'$ **do**
            if $k' \neq k$ **then**
                Decrease height of k' by $h/2$
        Increase height of k by h
        MOVEROW(k, K')
function MOVECOL(k, K)
    for each key $k' \in K$ **do**
        if k' is to the left of k or $k' = k$ **then**
            Move k' to the left until it is adjacent to something
        else
            Move k' to the right until it is adjacent to something
function MOVEROW(k, K)
    for each key $k' \in K$ **do**
        if k' is below k or $k' = k$ **then**
            Move k' up until it is adjacent to something
        else
            Move k' down until it is adjacent to something

後に，各キーが被らないように，k と K' を移動する (MoveRow 関数を呼び出す).

MoveCol 関数は，指定されたキー k と，そのキーを含む行の他のキー K を受け取り，次の操作を行う．K 内のそれぞれのキー k' について，k' が k の左にある，または $k' = k$ の場合は，k' を左の何かと隣接するまで左に移動する．そうでない場合は，k' を右の何かと隣接するまで右に移動する．

MoveRow 関数は，指定されたキー k と，そのキーを含む行の他のキー K を受け取り，次の操作を行うものである．K 内のそれぞれのキー k' について，k' が k の下にある，または $k' = k$ の場合は，k' を上の何かと隣接するまで上に移動する．そうでない場合は，k' を下の何かと隣接するまで下に移動する．

4.2.2 キーの色の変更

この節では，実行中に動的にキーの色を変更する方法について述べる．キーの背景色情報は，Keyboard に保存されておらず，レイアウトファイルから直接読み取られ，KeyboardView に保存されておらず．KeyboardView によるキーボードのレンダリングをする時には，同じ色を各キーの背景にしてレンダリングをする．つまり，任意のキーの背景の色情報を直接コードで変更することはできない．

この問題を解決するために，まず Keyboard に各キーに対して，そのキーの色を表す変数を追加する．次に，KeyboardView がキーボードをレンダリングをするときのコードを調整する．通常，KeyboardView は，図 4.1(a) に示すように，まず各キーの背景をレンダリングし，その上に対応するアルファベットをレンダリングをする．これを，図 4.1(b) に示すように，まず各キーの背景をレンダリングし，その上に対応するキーに保存された色を持つ矩形パターンを重ねてレンダリングし，最後にその矩形パターンの上にキーのアルファベットをレンダリングをするように調整する．これにより，キーの色を動的に設定することができるようになる．

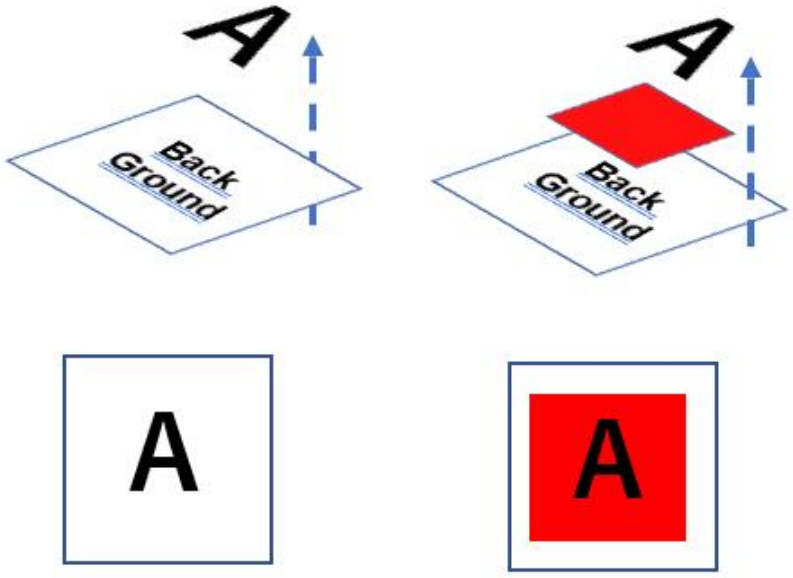
また，色を設定する際には，色だけでなく，不透明度も設定できるため，キーの目立つ程度を調整することができる．図 5.3 に示すように，図 4.2(a) は 100%不透明な赤いキーであり，図 4.2(b) は 30%不透明な赤いキーである．

4.3 単語予測

本節は，キーボードにおける単語予測機能の実装を 2 つの部分に分けて説明する．第一の節では，n-gram 言語モデルを用いた単語予測機能の実装を紹介する．第二の節では，単語予測機能をキーボードに統合する方法を紹介する．

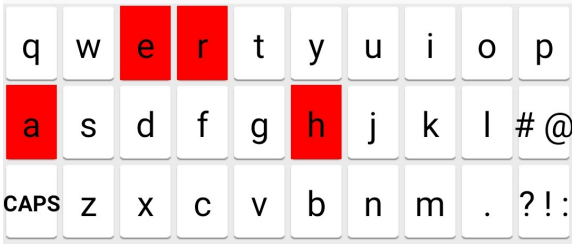
4.3.1 Pressagio に基づく単語予測の実装

単語予測機能は，オープンソースライブラリ「Pressagio」を用いて実装する．Pressagio は，N-gram モデルを使ってテキスト予測のためのオープンソースツールキットとして，ユーザー

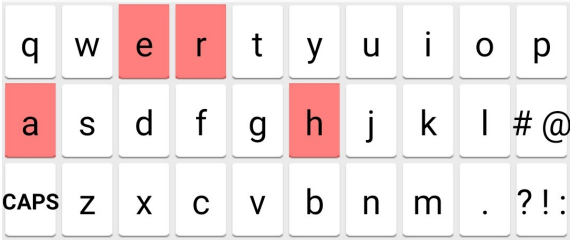


(a) 元のキーのレンダリング　　(b) 色が設定できるキーのレンダリング

図 4.1: キーのレンダリングのイメージ



(a) 100%不透明な赤いキー



(b) 30%不透明な赤いキー

図 4.2: 違う不透明度のキーのイメージ

が次に入力する単語を予測することに使える [16].

論文 [17] で紹介されているデータセットから得られる 1347 個の文を用いて，計 11488 単語のテキストを言語モデルの訓練用コーパスとして，3-gram 言語モデルを訓練する．訓練に使用される文からは，アルファベット以外の記号を全部削除した．Pressagio で訓練された 3-gram モデルは，現在入力された内容に基づいて，単語の予測を行い，(W, p) というペアのリストを返すことができる．W は単語を表し，p はその単語が入力される可能性を表す $(0 < p < 1)$．この結果アルファベットの予測に使用される．

4.3.2 キーボードに単語予測を組み込む実装

Pressagio を基に開発した単語予測プログラムは，Android システム上では直接実行できない．そのため，キーボードと単語予測プログラムの通信を行うサーバプログラムを実装した．

入力を迅速に予測することを確保するためには，通信のパフォーマンスが保障される必要がある．そのため，キーボードと単語予測プログラム間の通信には，RPC 技術で通信プログラムを実装することを選択した．RPC（Remote Procedure Call）とは，異なるコンピュータ間でプログラムをリモートで呼び出すことができるネットワーク通信技術である [18]．この技術は，プログラムがネットワークのことを気にせずに，リモートサービスをローカル関数のように呼び出すことを可能にする．RPC 技術を使用した通信は，RPC がシリアル化のためにバイナリプロトコルを使用するため，転送されるデータのサイズが小さい．そのうえ，RPC フレームワークは通常，性能を向上させるための最適化を備えているため，より高いパフォーマンスを発揮できる．

本研究では, gRPC という RPC フレームワークを使用して通信プログラムを開発する．gRPC は, Google によって開発された高性能でオープンソースの RPC フレームワークであり, Protocol Buffers [19]．をシリアル化メカニズムとして使用する [20].

まず，Protocol Buffers を使用して，ネットワーク通信プロトコルを定義した．このプロトコルには，通信中に使用するメッセージフォーマットとサービス定義が含まれている．

その中で，Predictor というサービスを定義した．このサービスには，RPC 関数 doPredict が含まれており，このメソッドは PredictRequest というリクエストメッセージを受け取り，PredictReply というレスポンスメッセージを返す．

また，リクエストメッセージ PredictRequest とレスポンスメッセージ PredictReply のフォーマットも定義した．PredictRequest には，文字列 input と整数 num が含まれている．ここでの input は，入力された文字を表し，num は予測する最大単語数を表す．一方，PredictReply には複数の suggestion が含まれており，suggestion は文字列 word と浮動小数点数 probability を持つメッセージであり，ここでの word は可能な単語を表し，probability はその可能性を表す．

次に，このプロトコルを使用し，キーボードプログラムと単語予測プログラムの通信を行うことができるように，gRPC フレームワークに基づく Python サーバを開発する．そして，キーボードのプログラム内では，サーバーへの RPC リクエストの送信機能を実装する．サーバ側でキーボードのプログラムからの予測リクエストを処理し，単語予測プログラムを呼び出し，

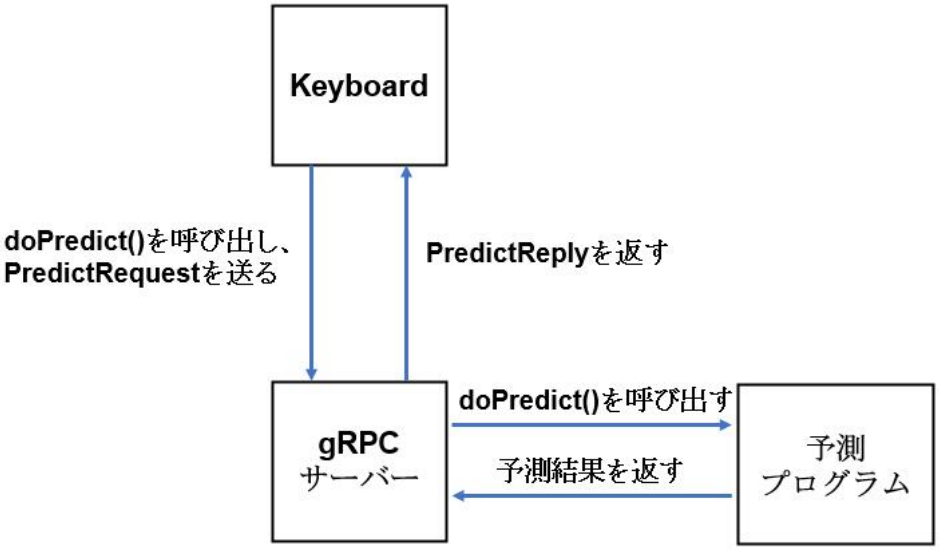


図 4.3: キーボードと単語予測プログラム間の通信のシーケンス

予測結果を含む応答を返すことで，キーボードと単語予測プログラム間の通信を実現する．

図 4.3 は，キーボードと単語予測プログラム間の通信を示すイメージ図である．まず，キーボードは gRPC に提供される関数 doPredict を呼び出し，今の入力内容と予測する最大単語数を PredictRequest にして，gRPC サーバーに送る．gRPC 側はリクエストを引き受けると，予測プログラムの doPredict を呼び出して，予測結果を得る．そしてその結果を PredictReply にして，キーボード側に返す．

4.4 アルファベット予測

前述したように，本研究で提案するアルファベット予測は，単語予測に基づくものである．単語予測機能は，次に入力する可能性のある単語を予測するだけでなく，単語が入力される確率も提供できる．それで，アルファベット予測アルゴリズムは，この確率を利用し，各アルファベットが次に入力される確率を計算し，最も高確率に入力されるアルファベットを見出すことがでる．

次に，提案される次のアルファベットが入力される確率を計算するアルゴリズムを紹介する．アルゴリズム 2 は，2 つのパラメーター P と n を受け取る．P は，単語予測プログラムから返される予測結果になる単語と確率のタプル (W, p) のリストである．W は単語であり，p は単語が現れる確率である．n は，入力中の単語の $n-1$ 文字まで入力が確定し, その単語の n 文字目を予測する．

Algorithm 2 Predict Next Letter

Require: P: The result of word predict program, n: Input is currently fixed up to $n-1$ letters of a word, and predict the nth letter of the word

1: **procedure** PREDICTNEXTLETTER(P, n)
2: $Q \leftarrow$ empty map from letters (a-z) to probabilities
3: **for** ch from 'a' to 'z' **do**
4: $Q[ch] \leftarrow 0$
5: **for** (W, p) in P **do**
6: **if** $n < |W|$ and W_n is a letter **then**
7: $Q[W_n] \leftarrow Q[W_n] + p$
8: return Q

アルゴリズムには，文字 a から z が現れる確率を保存するマッピングテーブル Q がある．まず，すべての文字の発生確率を 0 に初期化する．その後，P のすべてのタプル (W, p) を走査し，W の n 番目のアルファベット W_n の入力確率を W の確率 p と仮定して，Q[W_n] に足し合わせる．最後に，マッピングテーブル Q をアルゴリズムの結果として返す．

このアルゴリズムを通じて，単語予測の結果に基づいて，最も入力される可能性のある文字を確定することができる．この機能をキースタイルを動的に変更する機能と統合することで，キーボード内でユーザーが次に入力する可能性がある文字のスタイルを動的に変更する機能を実現することができる．

第5章　実験システムと実験結果の処理

提案したキーボードが入力効率に与える影響を調べるため，実験システムを設計および実装した．この実験システムにより，実験参加者により良いユーザーエクスペリエンスを提供したいと考えている．本章では，まず実験システムの概要を説明し，次に実験結果の処理について述べる．

5.1　実験システム

本実験システムでは，さまざまな種類のキーボードが設定できる．セットアップ画面（図5.1）では，キーのサイズや色や実験参加者の名前などを設定することができる．キーボードが設定された後，メイン画面（図5.2）に移る．この画面の上部にある選択ボックスで，入力テキストを選択できる（現在の選択テキスト：0SHORT_BENCH1.TXT）．実験参加者が入力中に頻繁にキーを切り替えないようにするために，入力テキストには大文字，句読点やその他の記号は含まれない．画面の上部には，現在入力するテキストの提示があり，実験参加者が1つのアルファベットを入力するたびに，赤いカーソルが現在の入力位置を示す．実験参加者が1つの文章を入力してから，DONE キーを押すと自動的に次の文章に移る．

5.2　実験結果の処理

実験システムを使用した実験において生じる記録を保存するためのデータ構造である InputLog を設計した．これを使用して，実験中にユーザーが行ったすべてのキーボード入力操作を記録する．毎回入力操作により新しい InputLog が生成されることで，それぞれの実験における全ての入力操作の情報を一連の InputLog で保存することができる．

InputLog は，入力されたキー情報（文字，スペースまたは改行），入力された時間，予測結果，正しい文字として期待される入力，および予測に要した時間を記録することができる．これらの情報は，ユーザーの入力速度（WPM），予測の精度，および入力のエラー率（ErrorRates）を計算するのに使われ，これらのデータを分析することで，より正確に実験の結果を理解することに役に立つ．

以下の節では，InputLog を用いて WPM と ErrorRates を計算する方法を詳しく説明する．

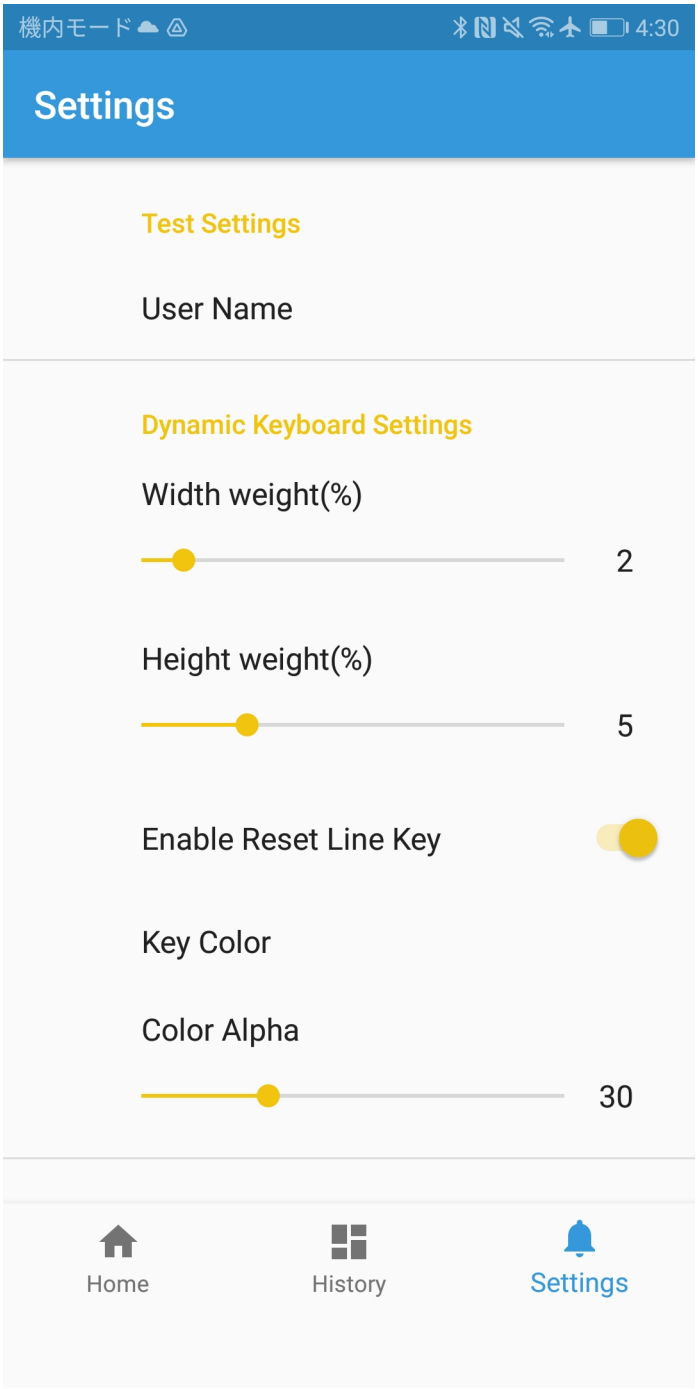


図 5.1: 実験システムのセットアップ画面

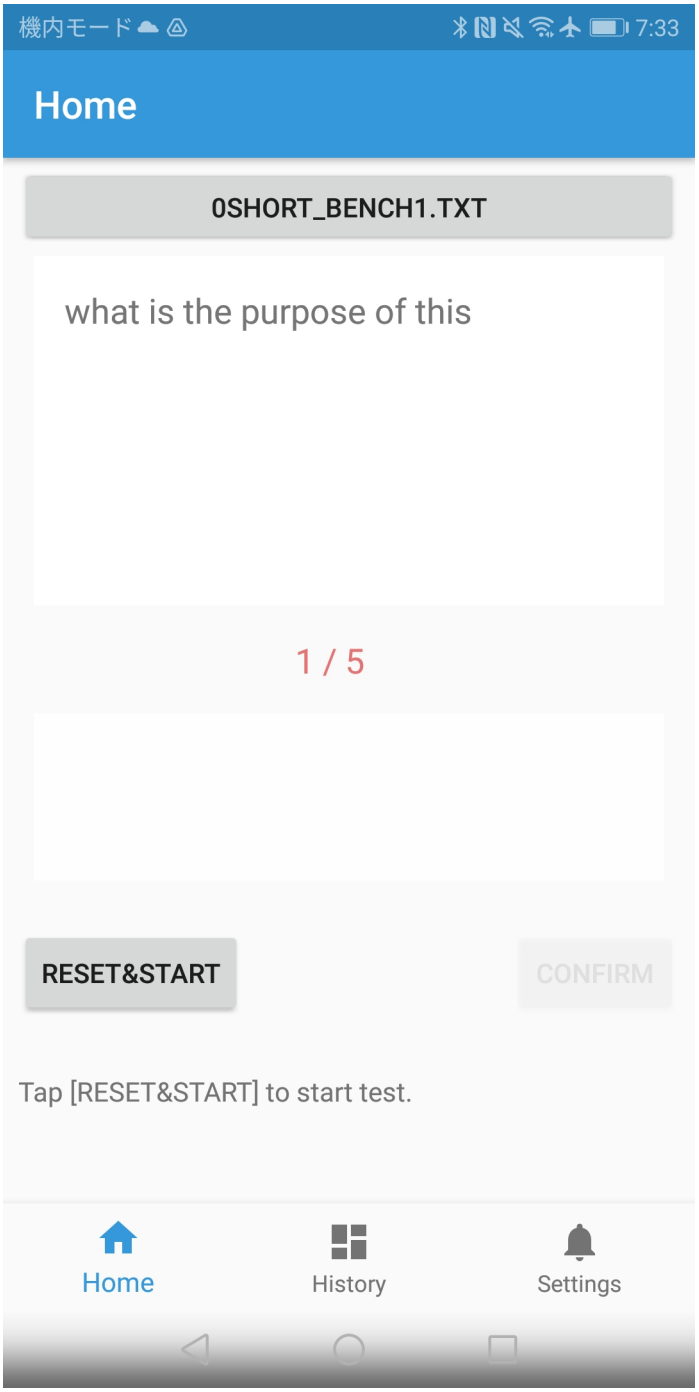
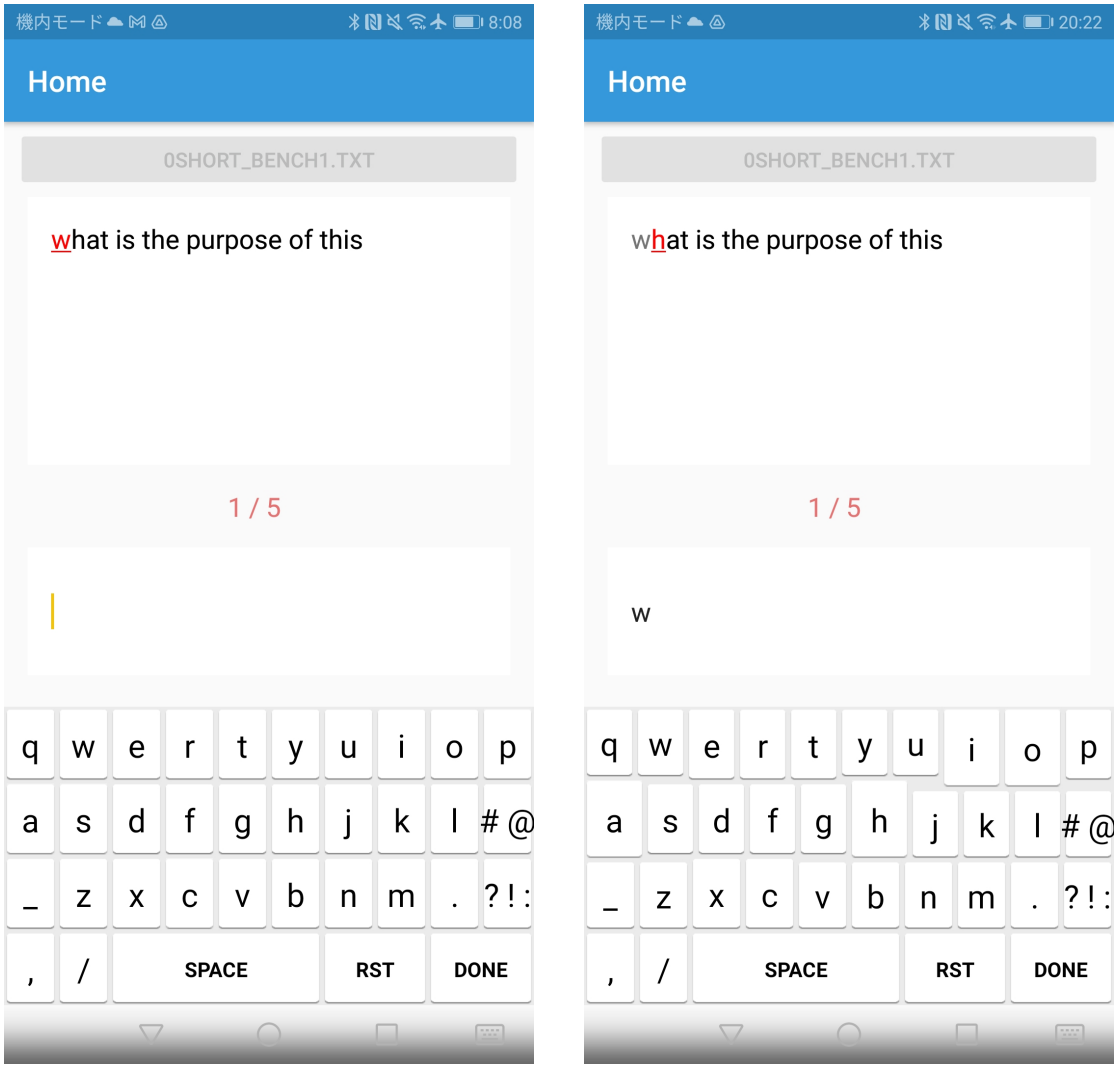


図 5.2: 実験システムのメイン画面



(a) (b)

図 5.3: 実験システムの入力画面

5.2.1 WPM の計算

本節では，InputLog を用いてユーザーが入力した際の 1 分あたりの入力単語数（WPM）の計算方法を説明する．WPM は，ユーザーの入力速度を測定するためによく使われる指標であり，ユーザーの入力効率を反映できる．WPM は，ユーザーが 1 分間に入力できる単語数を表し，計算式は次のとおりである．

$$\textbf{WPM} = \frac{n \times 60 \times 1000}{5 \times t}$$

ここで，n は，時間 t 内にユーザーが入力した文字数を表し，単位は個である．t は，ユーザーが入力するのにかかる時間を表し，単位はミリ秒である．WPM は通常，入力速度を測定するために使われるため，WPM を計算する際には，通常文字数 n を 5 で割ることになる．これは，英語では単語の平均文字数は 5 なので，WPM の結果がユーザーの入力速度をより良く反映するようにするために，文字数 n を 5 で割ることが通常である．

```
1:  function GETWPMS(logs)
2:      n ← 0
3:      t1 ← null
4:      t2 ← null
5:      lastLog ← null
6:      ret ← empty list
7:      for each log ∈ logs do
8:          if n == 0 then
9:              if log.type is ALPHABET then
10:                 t2 ← null
11:                 t1 ← log.time
12:         else if t2 == null then
13:             if log.type is NEXTLINE or log is the last one in logs then
14:                 n ← 0
15:                 t2 ← lastLog.time
16:                 wpm ← (n×60×1000) / (5×(t2−t1))
17:                 add wpm to ret
18:         if log.type is ALPHAPET then
19:             n ← n + 1
20:         lastLog ← log
21:     return ret
```

GETWPMS は，入力中のユーザーの毎分キャラクター数（WPM）を計算する機能である．これは，入力記録（InputLog のリスト $logs$）をトラバーサルすることで実現される．トラバー

サル中に，新しい入力行（すなわち，ユーザーが「改行」キーを押すか，すべての入力記録を巡回した場合）が見つかる度に，その行の WPM が計算され，結果セットに加えられる．最後に，プログラムは，すべての入力行の WPM を含む結果セットを返す．

具体的には，プログラムはまず，いくつかの変数を初期化する．n は，現在の入力行の入力文字数を記録するために使用され，t_1 および t_2 は，それぞれ，現在の入力行の開始時間と終了時間を記録するために使用される．$lastLog$ は，前回の入力記録を記録するために使用される．ret は，結果を保存するために使用される．次に，プログラムは入力記録をトラバーサルする．トラバーサル中に，現在の入力記録がアルファベットである場合，n を 1 加える．現在の入力行の終わり（すなわち，ユーザーが「改行」キーを押すか，すべての入力記録をトラバーサルした場合）が見つかると，その行の WPM が計算され，結果セットに加えられ，n がクリアされる．最後に，プログラムは，すべての入力行の WPM を含む結果のリスト ret を返す．

5.2.2 ErrorRates の計算

本節では，入力時のユーザーのエラー率（ErrorRates）を計算する方法を説明する．計算された ErrorRates は，入力テキストとソーステキストの間の差異の程度を反映する．この差異の程度を表すために，入力テキストとソーステキストの編集距離を，ソーステキストの百分率で表す．計算方法は次のとおりである．

$$\textbf{ErrorRates} = \frac{d \times 100}{length(src)}$$

ここで，d は入力テキストとソーステキストの編集距離を表し，$length(src)$ はソーステキストのテキスト長を表す．編集距離は，1 つの文字列を別の文字列にするために必要な最小の編集操作数を表す．これには，挿入，削除，および置換が含まれる [21]．入力テキストとソーステキストの間の編集距離が小さいほど，その間の差異が小さくなる．編集距離が，ソーステキストの百分率で小さいほど，差異の程度が低くなり，つまり ErrorRates が低くなる．

```
1: function GETERRORRATES(logs, src)
2:     S ← GETSUBSRCTEXTS(src)
3:     I ← GETSUBINPUTTEDTEXTS(logs)
4:     ret ← empty list
5:     for i ← 0 to length of S do
6:         d ← EDITDISTANCE(S_i, I_i)
7:         e ← (d×100) / length(src)
8:         add e to ret
9:     return ret
```

GETERRORRATES は，入力された文書を行単位に分割したもの I と，それに対する原文 S

の行単位の分割の文書間の間のそれぞれの ErrorRates$e$ を計算するものである．ここで，d は S_i と I_i の編集距離を表す．最初に，GETSUBSRCTEXTS を呼び出して，原文を行単位で分割したもの S を取得する．次に，GETSUBINPUTTEDTEXTS を呼び出して，入力された文章を行単位で分割したもの I を取得する．次に，空リスト ret を初期化し，これを最終的な ErrorRates を保存するために使用する．S にある各行について，EDITDISTANCE を呼び出して，S_i と I_i の編集距離 d を計算する．その後，d に 100 をかけたものを，文章 src の長さで割り，その行の ErrorRates e として求める．これは，原文と入力文の間の差異が，原文の何パーセントを占めるかを表すものである．最後に，各行の ErrorRates e を ret に追加する．最終的に，ret を返す．

擬似コードで使われるいくつかの関数の説明はつのとおりである．

- GETSUBSRCTEXTS: この関数は，文字列 src を入力として受け取り，src を改行で区切り複数の行に分け，それらを配列の要素として返す関数である．
- GETSUBINPUTTEDTEXTS: この関数は，ログを記録する配列 $logs$ を入力として受け取り，$logs$ の配列を反復処理し，各行の開始と終了（改行の InputLog で区切ることで検出）を探し，各行に入力する内容を配列の要素として返す関数である．
- EDITDISTANCE: この関数は，2 つの文字列 $s1$ および $s2$ を入力として受け取り，Wagner–Fischer アルゴリズム [22] を使用し，$s1$ と $s2$ の間の編集距離を計算し，その結果を整数として返す関数である．

第6章　実験

本研究では, キーのスタイルが入力効率に与える影響を調査し, 実験を行った. 本章では，予備実験 1，予備実験 2 および評価実験から構成されている．まずは, 予備実験 1 でキーのサイズが入力効率に与える影響を調査する. 次に, 予備実験 2 でキーのサイズと色両方が入力効率に与える影響を調査する. 最後に, 本論文で提案した予測方法を基にキーのサイズと色両方が入力効率に与える影響を調査する.

6.1　予備実験 1：サイズの入力に対する影響の調査

6.1.1　実験参加者

実験参加者は大学院生 9 人（24-27 歳，女性 3 名，男性 6 名）を対象とした．そのうち，6 人がスマートフォンでの QWERTY 配列キーボードの使用経験があり，3 人が QWERTY 配列キーボードの使用経験がなかった.

6.1.2　実験環境

本実験では，Android スマートフォン (HUAWEI Mate 10 Pro) を使用する．この Android スマートフォンには，本論文で設計・開発した QWERTY 配列のソフトウェアキーボードが搭載されている．ソフトウェアキーボードのキーは，N,S,M,L の４つのキーのサイズ（図 6.1）を拡大する方法を用意する：

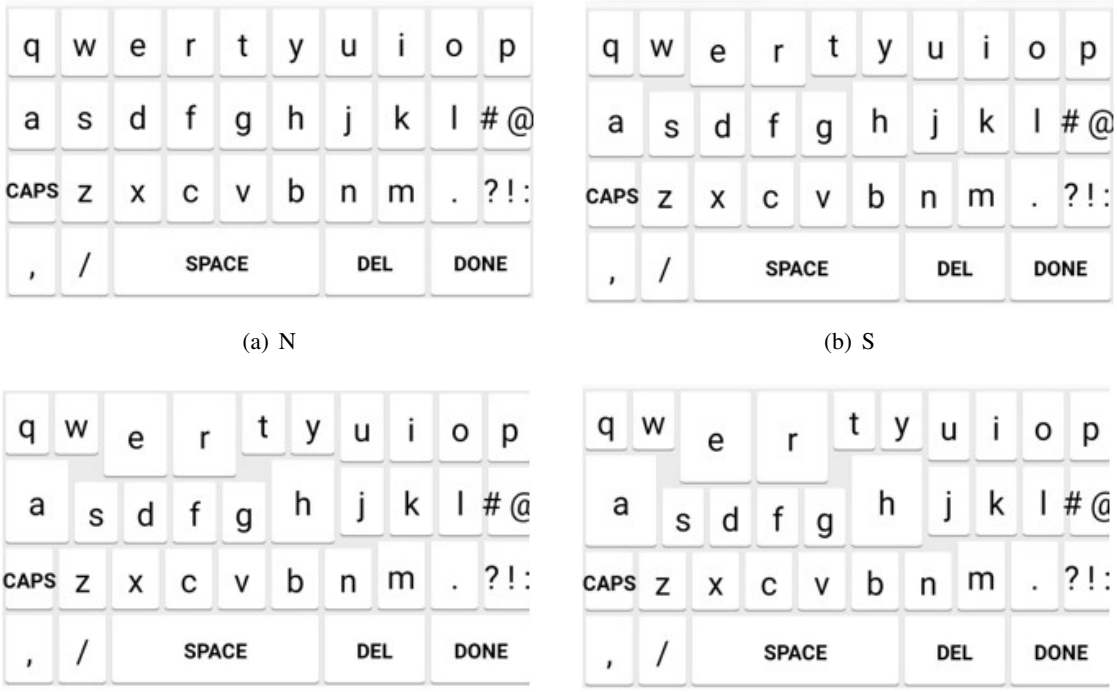
N: 幅と高さを 0%伸ばす

S: 幅がおよそ 18%伸ばし, 高さはおよそ 10%伸ばす

M: 幅がおよそ 36%伸ばし, 高さはおよそ 20%伸ばす

L: 幅がおよそ 54%伸ばし, 高さはおよそ 30%伸ばす

予備実験で，予測の精度と予測にかかる時間の影響を考えずに，キーのスタイルが動的に変更することによる影響を考察する目的であるため，予想精度が 100%となる実験用のアルファベット予測プログラムを使った．この予測プログラムは，1 つの正しいアルファベットと 3 つのその他のアルファベットを返す．正しいアルファベットはテストテキストから得るも



(a) N (b) S

(c) M (d) L

図 6.1: 予備実験 1 における使ったキーのサイズ

のである．他の 3 つのアルファベットについは, 約 8 万個の単語からプレフィックスが入力している内容と同じである単語を探し，探した単語の中次に入力する位置にあるアルファベットが出た回数を計り，回数が最も多い 3 つのアルファベットである．ここの 8 万個の単語は 12dicts という公開の英語辞書 [23] から得た文字しか含まない単語である．

実験に入力された文章については，Vertanen ら [17] のスマートフォンのメールでよく使われるフレーズ集から抽出した 20 個の文章を実験用のベンチマークとして，実験参加者はこの実験用のベンチマークで実験を行った．

6.1.3 実験内容

本実験の目的は，キーのサイズが入力効率に与える影響を調査することである．

本実験のタスクは，スマートフォン画面上部に提示された文章を入力することである. タスクが始まる際には，実験参加者に START ボタンを押してもらってから，入力をしてもらう．タスクを終える際には，実験参加者にキーボードにある DONE ボタンを押してもらう．5 文章の入力を 1 セッションとし，計 4 セッションを行った．入力各セッションの最後には，実験参加者に約 3 分間の休憩を取ってもらう．実験参加者は合計 4 つの種類のキーボード（図 6.1）を使い，入力セッションのタスクを行った．合計 720 回（=5 タスク ×4 セッション ×4 手法 ×9 人）の測定が行われた．

本実験の手順は以下の通りである. まず，実験参加者に実験用ソフトウェアキーボードの使い方を説明した．そして，実験参加者に実験前のアンケートに回答してもらった．実験前のアンケートに回答してから，練習セクションを始めた．実験参加者に 4 種類のソフトウェアキーボード（N，S，M，L）をそれぞれ使って練習してもらった． 練習終了後，5 分間の休憩を取ってもらう．まずはNサイズのソフトウェアキーボードを使用し実験を行った．1 種類のソフトウェアキーボードでの実験を終了後 10 分間の休憩を取ってもらった．休憩した後に次の種類のソフトウェアキーボードでの実験を始めた．すべての種類のソフトウェアキーボードを実験した後で実験参加者にアンケートに回答してもらい，実験を終了した．

6.1.4 実験結果

各々のキーボードの平均 WPM について, N の平均 WPM は 23.61（SD=5.80）であり，S の平均 WPM は 24.65（SD=6.10）であり，M の平均 WPM は 23.91（SD=7.18）であり，L の平均 WPM は 23.46（SD=6.91）である．まず，Shapiro-Wilk 検定を使用して，各々のキーボードのデータの正規性を確認し，すべてのデータの正規性を確認できたため ($p_N = 0.23, p_S = 0.14, p_M = 0.06, p_L = 0.11$)，一元配置の分散分析を使用して，各々のキーボードの平均 WPM の間に有意差があることを確認できなかった. ($p = 0.958$) そして, 図 6.2 の箱ひげ図に示しているように, N, M, L のデータは似ている範囲内（20〜25 前後）に分布されているが, S のデータがより狭い範囲内（22.5〜25 前後）に分布されている．

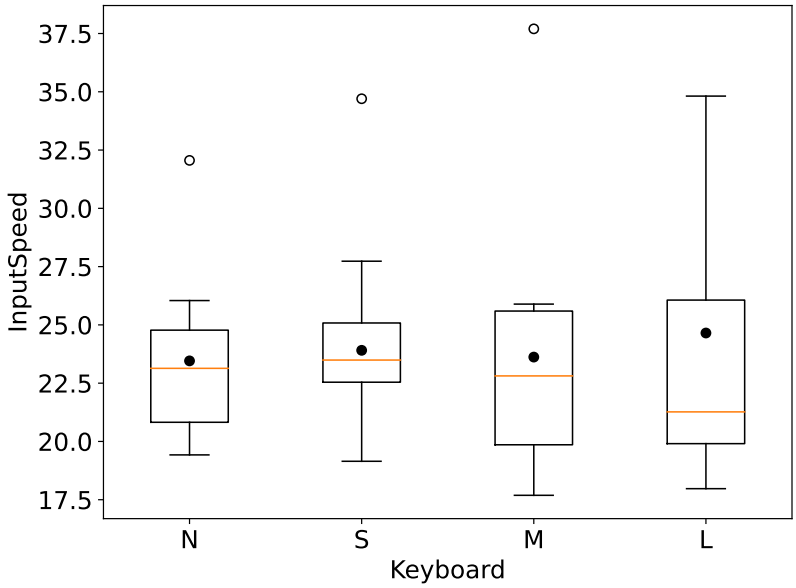


図 6.2: 予備実験 1 におけるキーボードごと毎の入力速度（WPM）

各々のキーボードの平均 ErrorRates について, N の平均 ErrorRates は 11.37（SD=14.61）であり，S の平均 ErrorRates は 8.11（SD=11.92）であり，M の平均 ErrorRates は 8.1（SD=9.95）であり，L の平均 ErrorRates は 8.81（SD=10.92）である．まず，Shapiro-Wilk 検定を使用して，各々のキーボードのデータの正規性を確認し，すべてのデータの正規性を確認できなかったため ($p_N = 0.01, p_S = 0.03, p_M = 0.48, p_L = 0.1$)，Kruskal-Wallis 検定を使用して，各々のキーボードの平均 WPM の間に有意差があることを確認できなかった ($p = 0.75$)．そして, 図 6.3 の箱ひげ図に示しているように, 4 つのキーボードのデーター分布は似ているように見える．

6.2 予備実験 2：色とサイズの入力に対する影響の調査

6.2.1 実験参加者

実験参加者は大学院生 9 人（24-29 歳，女性 5 名，男性 4 名）を対象とした．全員がスマートフォンでの QWERTY 配列キーボードの使用経験がある．そのうち，6 人の使用程度（1〜5：1 が QWERTY 配列キーボードを一度も使わなかったこと，5 が QWERTY 配列キーボードを頻繁に使用すること）は 5 であり，3 人の使用程度は 2 である．

6.2.2 実験環境

本実験では，Android スマートフォン (HUAWEI Mate 10 Pro) を使用する．この Android スマートフォンには，本論文で設計・開発した QWERTY 配列のソフトウェアキーボードが搭載

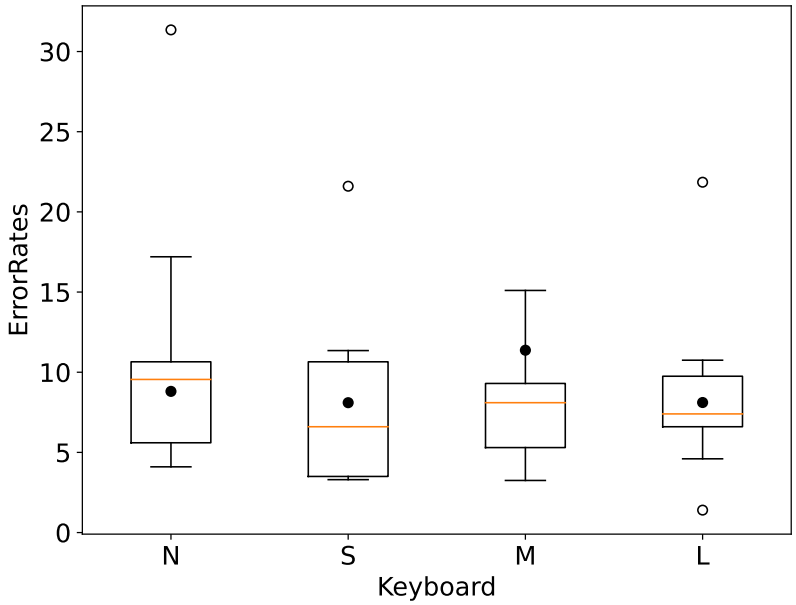


図 6.3: 予備実験 1 におけるキーボードごと毎の平均エラー率（%）

されている．ソフトウェアキーボードのキーは，N,S,rN,rS の 4 つのキーのスタイル（図 6.7）を拡大する方法を用意する:

N: 幅と高さを 0%伸ばす

S: 幅がおよそ 18%伸ばし, 高さはおよそ 10%伸ばす

rN: 幅と高さを 0%伸ばす；30%不透明な赤い色にする

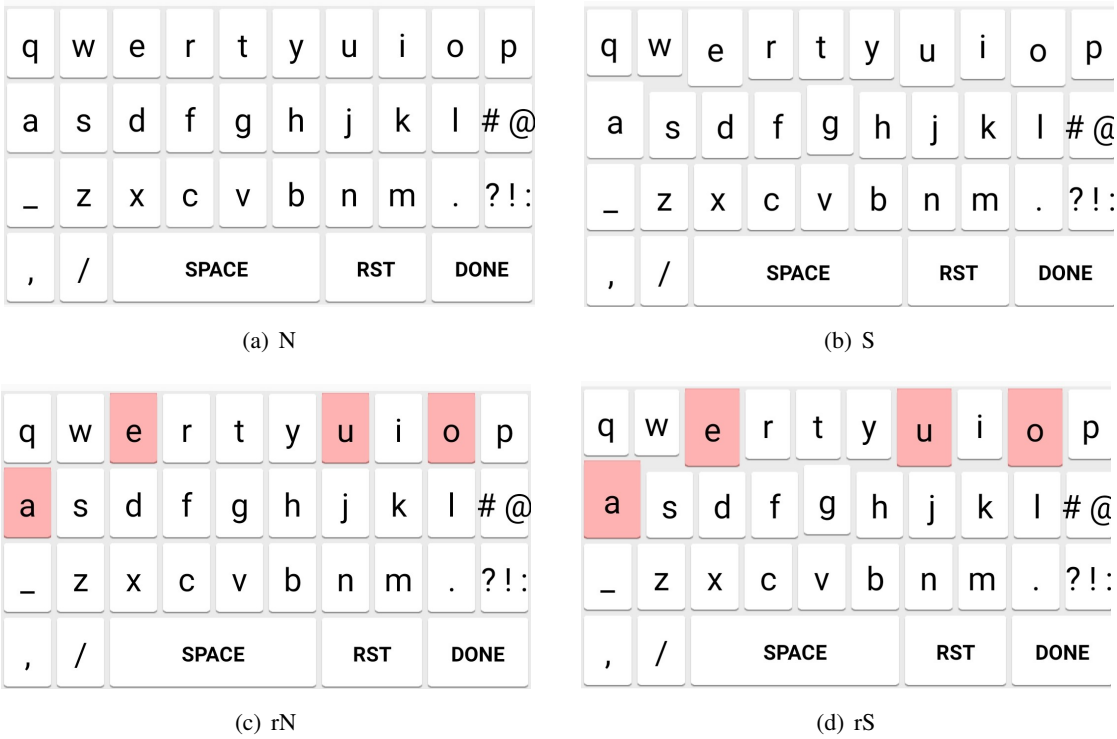
rS: 幅がおよそ 18%伸ばし, 高さはおよそ 10%伸ばす；30%不透明な赤い色にする

予備実験 2 では使った予測方法は予備実験 1 と同じである．実験に入力された文章については，Vertanen ら [17] のスマートフォンのメールでよく使われるフレーズ集から抽出した 20 個の文章を実験用のベンチマークとして，実験参加者はこの実験用のベンチマークで実験を行った．

6.2.3 実験内容

本実験の目的は，キーのサイズと色両方が入力効率に与える影響を調査することである．

本実験のタスクは，スマートフォン画面上部に提示された文章を入力することである. タスクが始まる際には，実験参加者に START ボタンを押してもらってから，入力をしてもらう．タスクを終える際には，実験参加者にキーボードにある DONE ボタンを押してもらう．5 文



(a) N

(b) S

(c) rN

(d) rS

図 6.4: 予備実験 2 における使ったキーのスタイル

章の入力を 1 セッションとし，計 4 セッションを行った．入力各セッションの最後には，実験参加者に約 3 分間の休憩を取ってもらう．実験参加者は合計 4 つの種類のキーボードを使い，入力セッションのタスクを行った．合計 720 回（=5 タスク ×4 セッション ×4 手法 ×9 人）の測定が行われた．

本実験の手順は以下の通りである. まず，実験参加者に実験用ソフトウェアキーボードの使い方を説明した．そして，実験参加者に実験前のアンケートに回答してもらった．実験前のアンケートに回答してから，練習セクションを始めた．実験参加者に 4 種類のソフトウェアキーボード（図 6.7）をそれぞれ使って練習してもらった． 練習終了後，5 分間の休憩を取ってもらう．まずはNサイズのソフトウェアキーボードを使用し実験を行った．1 種類のソフトウェアキーボードでの実験を終了後 10 分間の休憩を取ってもらった．休憩した後に次の種類のソフトウェアキーボードでの実験を始めた．すべての種類のソフトウェアキーボードを実験した後で実験参加者にアンケートに回答してもらい，実験を終了した．

6.2.4 実験結果

各々のキーボードの平均 WPM について, N の平均 WPM は 26.71（SD=6.19）であり，S の平均 WPM は 28.28（SD=6.92）であり，rN の平均 WPM は 25.66（SD=6.09）であり，rS の平均 WPM は 27.01（SD=5.51）である．まず，Shapiro-Wilk 検定を使用して，各々のキーボードのデータの正規性を確認し，すべてのデータの正規性を確認できたため ($p_N = 0.17, p_S = 0.07 p_{rN} = 0.89, p_{rS} = 0.35$)，一元配置の分散分析を使用して，各々のキーボードの平均 WPM の間に有意差があることを確認できなかった ($p = 0.80$)．そして, 図 6.5 の箱ひげ図に示しているように, N, S, rS のデータは似ている範囲内（22.5〜32.5 前後）に分布されているが, rN のデータが他の範囲内（21.5〜28.5 前後）に分布されている．

各々のキーボードの平均 ErrorRates について, N の平均 ErrorRates は 2.86（SD=4.11）であり，S の平均 ErrorRates は 1.81（SD=4.12）であり，rN の平均 ErrorRates は 1.73（SD=2.93）であり，rS の平均 ErrorRates は 1.62（SD=3.16）である．まず，Shapiro-Wilk 検定を使用して，各々のキーボードのデータの正規性を確認し，すべてのデータの正規性を確認できたため ($p_N = 0.23, p_S = 0.96, p_{rN} = 0.39, p_{rS} = 0.30$)，一元配置の分散分析を使用して，各々のキーボードの平均 WPM の間に有意差があることを確認できなかった ($p = 0.11$)．そして, 図 6.6 の箱ひげ図に示しているように, rN と rS のデータの分布は似ているような狭い範囲内（1.5〜2 前後）に分布されている上で, N と比べて小さいところに分布されているということを確認できる．

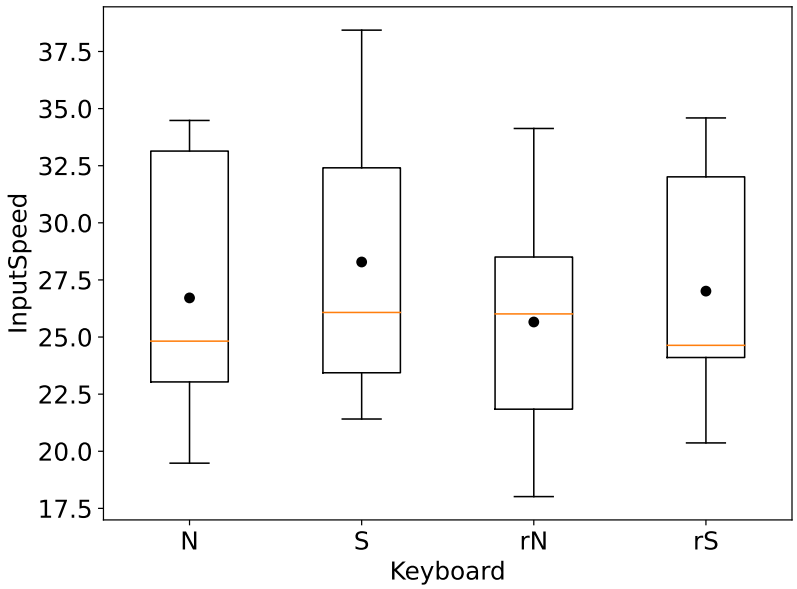


図 6.5: 予備実験 2 におけるキーボードごと毎の入力速度（WPM）

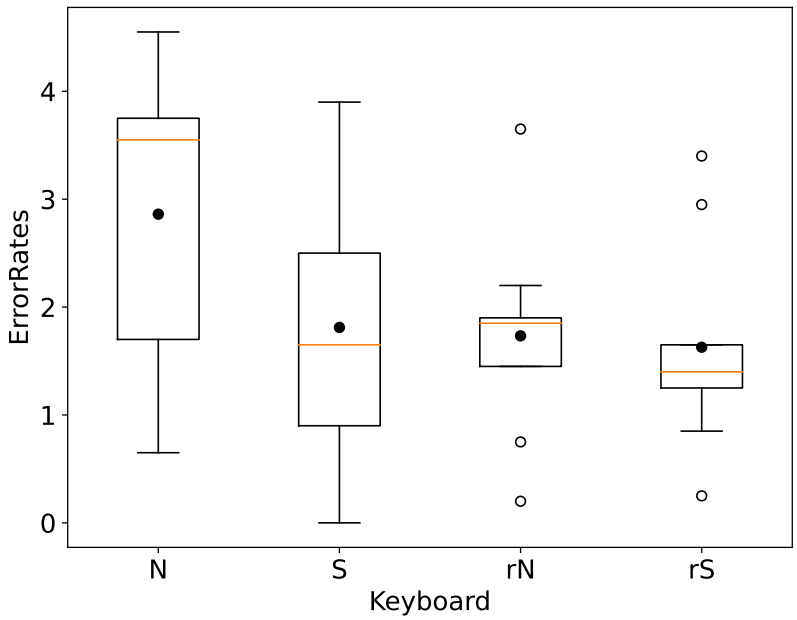


図 6.6: 予備実験 2 におけるキーボードごと毎のエラー率（%）

6.3 実験：提案したキーボードの評価実験

6.3.1 実験目的

本論文で実装した単語予測方法とアルファベット予測方法に基づき，キーのスタイルを調整することで入力効率に与える影響を調査する．

6.3.2 実験参加者

実験参加者は大学院生 5 人（24-29 歳，女性 3 名，男性 2 名）を対象とした．全員がスマートフォンでの QWERTY 配列キーボードの使用経験がある．そのうち，2 人の使用程度（1〜5：1 が QWERTY 配列キーボードを一度も使わなかったこと，5 が QWERTY 配列キーボードを頻繁に使用すること）は 5 であり，3 人の使用程度は 2 である．

6.3.3 実験環境

本実験では，Android スマートフォン (HUAWEI Mate 10 Pro) を使用する．この Android スマートフォンには，本論文で設計・開発した QWERTY 配列のソフトウェアキーボードが搭載されている．アルファベットと単語予測は, 予備実験と違い, 第 4 章の説明に従い実装された予測モジュールで行う. ソフトウェアキーボードのキーは，N,S,rN,rS の 4 つのキーのスタイル（図 6.7）を拡大する方法を用意する：

N: 幅と高さを 0%伸ばす

S: 幅がおよそ 18%伸ばし, 高さはおよそ 10%伸ばす

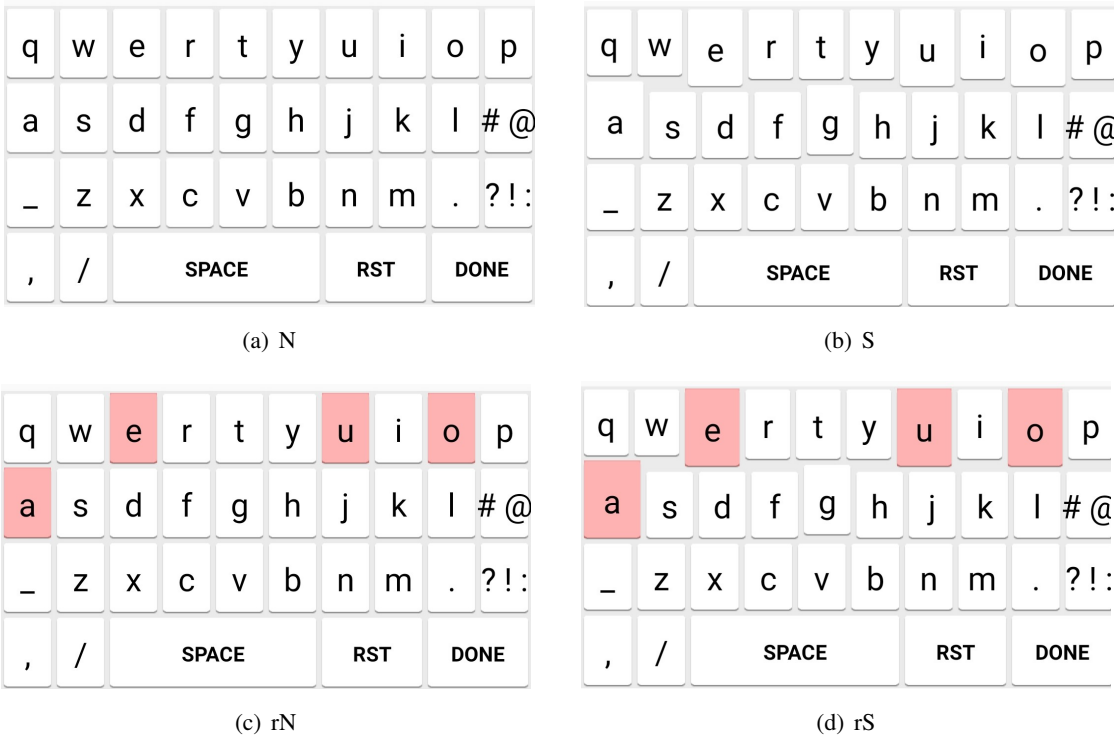
rN: 幅と高さを 0%伸ばす；30%不透明な赤い色にする

rS: 幅がおよそ 18%伸ばし, 高さはおよそ 10%伸ばす；30%不透明な赤い色にする

実験に入力された文章については，Vertanen ら [17] のスマートフォンのメールでよく使われるフレーズ集から抽出した 20 個の文章を実験用のベンチマークとして，実験参加者はこの実験用のベンチマークで実験を行った．

6.3.4 実験内容

本実験のタスクは，スマートフォン画面上部に提示された文章を入力することである. タスクが始まる際には，実験参加者に START ボタンを押してもらってから，入力をしてもらう．タスクを終える際には，実験参加者にキーボードにある DONE ボタンを押してもらう．5 文章の入力を 1 セッションとし，計 4 セッションを行った．入力各セッションの最後には，実験参加者に約 3 分間の休憩を取ってもらう．実験参加者は合計 4 つの種類のキーボードを使



(a) N (b) S

(c) rN (d) rS

図 6.7: 評価実験における使ったキーのスタイル

い，入力セッションのタスクを行った．合計 400 回（=5 タスク ×4 セッション ×4 手法 ×5 人）の測定が行われた．

本実験の手順は以下の通りである．まず，実験参加者に実験用ソフトウェアキーボードの使い方を説明した．そして，実験参加者に実験前のアンケートに回答してもらった．実験前のアンケートに回答してから，練習セクションを始めた．実験参加者に 4 種類のソフトウェアキーボード（図 6.7）をそれぞれ使って練習してもらった． 練習終了後，5 分間の休憩を取ってもらう．まずは N サイズのソフトウェアキーボードを使用し実験を行った．1 種類のソフトウェアキーボードでの実験を終了後 10 分間の休憩を取ってもらった．休憩した後に次の種類のソフトウェアキーボードでの実験を始めた．すべての種類のソフトウェアキーボードを実験した後で実験参加者にアンケートに回答してもらい，実験を終了した．

6.3.5 実験結果

実験に毎回予測にかかる時間は平均的に 49.09ms である．予測の正確率は 90.35%である．平均予測時間は各 InputLog に記録されている予測にかかる時間の平均値である．予測の正確率は予測結果に正しい文字を含む InputLog の数と全ての InputLog の数の比率である．

各々のキーボードの平均 WPM について, N の平均 WPM は 26.67（SD=5.61）であり，S の平均 WPM は 27.77（SD=6.86）であり，rN の平均 WPM は 24.44（SD=5.26）であり，rS の平均 WPM は 26.28（SD=4.47）である．まず，Shapiro-Wilk 検定を使用して，各々のキーボードのデータの正規性を確認し，すべてのデータの正規性を確認できたため ($p_N = 0.49, p_S = 0.36, p_{rN} = 0.56, p_{rS} = 0.43$), 一元配置の分散分析を使用して, 各々のキーボードの平均 WPM の間に有意差があることを確認できなかった ($p = 0.80$)．そして, 図 6.8 の箱ひげ図に示しているように, N, rN のデータは似ている範囲内（23〜29 前後）に分布されていて, S, rS のデータが似ている範囲内（24〜27 前後）に分布されている．

各々のキーボードの平均 ErrorRates について, N の平均 ErrorRates は 1.76（SD=3.04）であり，S の平均 ErrorRates は 1.9（SD=2.96）であり，rN の平均 ErrorRates は 1.78（SD=3.53）であり，rS の平均 ErrorRates は 1.31（SD=2.43）である．まず，Shapiro-Wilk 検定を使用して，各々のキーボードのデータの正規性を確認し，すべてのデータの正規性を確認できたため ($p_N = 0.97, p_S = 0.22, p_{rN} = 0.28, p_{rS} = 0.46$)，一元配置の分散分析を使用して，各々のキーボードの平均 WPM の間に有意差があることを確認できなかった ($p = 0.85$)．そして, 図 6.9 の箱ひげ図に示しているように, N, rN のデータは似ている範囲内（1.5〜2.5 前後）に分布されていて, S, rS のデータが似ている範囲内（1.3〜1.7 前後）に分布されている．

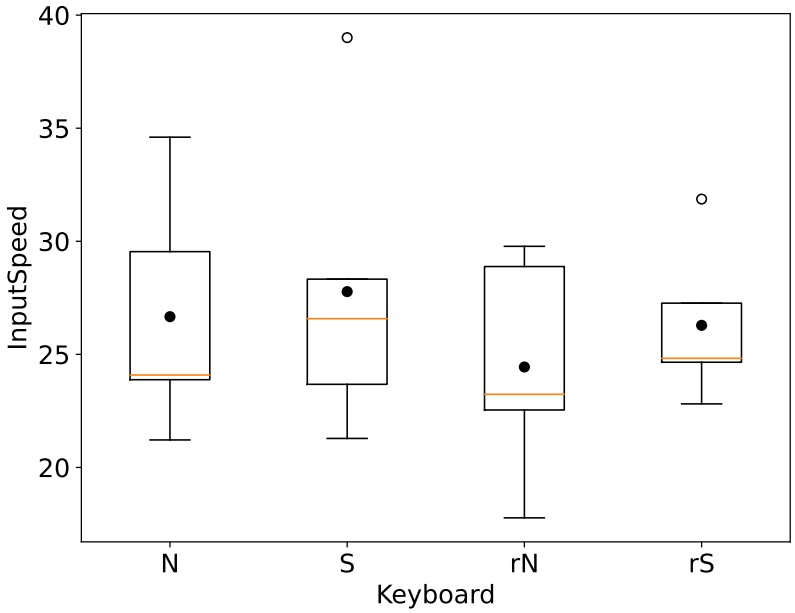


図 6.8: 実験におけるキーボードごと毎の入力速度（WPM）

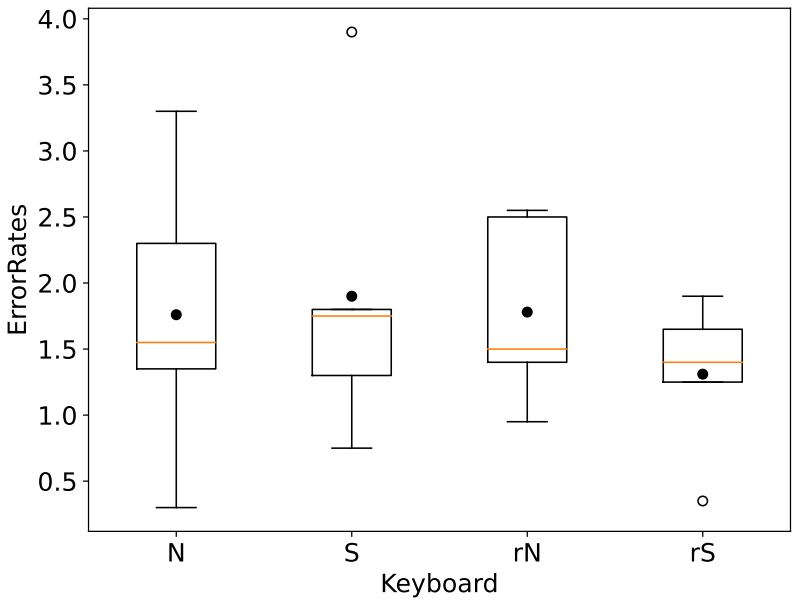


図 6.9: 実験におけるキーボードごと毎のエラー率（%）

第7章　議論

7.1　入力速度に関する考察

予備実験 1 では, 提案した 4 つのキーボードの中で S の平均入力速度が最も速いことが分かった. つまり, 適切にキーを大きくすることで入力速度が上がるという傾向が見られた. さらに, データの分布では, S が他の 3 つのキーボードよりも下限が高くなっており, キーのサイズを適切に大きくすることで, 入力速度の遅い人に対して入力速度を上げる傾向があることを示している.

予備実験 2 では, 提案した 4 つのキーボードの中で, N の平均入力速度は rN の平均入力速度より速く, S の平均入力速度は rS の平均入力速度より速いことが分かった. つまり, 色が変わると入力速度が下がる傾向が見られた. また, データの分布で, rN は他のキーボードに比べて著しく低い分布をしたことが示した. キーの色を変えるだけで入力速度にマイナスの影響がある. これは, キーの色を変えると入力に集中できなくなり, 入力速度が落ちるからだと推測される.

評価実験では, 予備実験 2 と似ている結果が出た. 提案したキーボードの中で N の平均速度は rN の平均速度より速く, S の平均速度は rS の平均速度より速いことが分かった. これは, 予測精度が 90%程度の場合, キーの色を変更すると入力速度が下がる傾向が見られた. 同時に, S と rS の平均速度はそれぞれ N と rN よりも速く, 予測精度が 90%程度の場合, キーのサイズを適切に大きくすると, 入力速度が上がるという傾向が見られた.

予備実験と評価実験で比較したキーボードには有意差はなかったが, データの分布と平均入力速度から, キーのサイズを適切に大きくすると入力速度が上がる傾向が見られた. ただし, キーの色のみを変えると速度が下がる傾向がある. そして, 予測精度を 100%から 90%に落としても同様の傾向がある. 有意差がなかった理由として考えられるのは, 以下の通りである.

- キーのスタイルが変更すると, ユーザーによって影響が異なる. 例えば, 一部分のユーザーには, キーを拡大することが積極的な影響を与え, 他のユーザーには消極的な影響を与えることがある. 特に, Qwerty キーボードの入力に非常に慣れているユーザーにとって, キーの変更は消極的な影響をもたらす可能性が高い.
- 実験参加者の人数が少なく, 結果の分析に利用できるデータが少なすぎた.
- 短時間の実験中に, 実験参加者が動的にキーを変更できるキーボードに適応していなかったため, キースタイルの変更が入力速度に大きな影響を与えることはなかったと思われる.

7.2 エラー率に関する考察

予備実験 1 では, 提案した 4 つのキーボードでエラー率の分布がほぼ同じであった．これはキーのサイズの変更がエラー率に明確な影響を与えないことを示している. 予備実験 2 では, N の平均エラーレートが他の 3 つのキーボードよりも高く, キースタイルの変更によりエラー率が下がる傾向を示している．また, rN と rS のデータ分布も同様に, N と S に比べてはるかにコンパクトであり, キーの色を変更するとエラー率が他のキーボードよりも安定していることが示唆された. 評価実験で提案したキーボードによる平均エラー率はほぼ同じであり，予測精度が 100%に達していない場合，キーのスタイルの変更がエラー率に明確な影響を与えないことを示している．また，N および rN のデータ分布はほぼ同じであり，S および rS のデータ分布はほぼ同じであり，やはり予測精度が 100%に達していない場合，キーの色を変更してもエラー率に大きな影響を与えないことが分かった．評価実験と予備実験の結果の差が大きいのは，予測精度が 100%に達していないため，ユーザーが間違った予測により間違ったキーを入力し，キースタイルを変更した際のエラー率が高くなったためと考えられる．評価実験と予備実験の結果の差が大きいのは，予測精度が 100%に達していないため，ユーザーがキーボードを使用する際に，間違った予測の結果によって入力ミスが発生する可能性がある．つまり，キーのスタイルを変更するとエラー率が高くなった可能性があると考えられる．エラー率に対して有意差がなかった理由として考えられるのは, 以下の通りである.

- エラー率に影響を与える要因が多すぎるから. 例えば, 実験参加者が英語に把握する程度，Qwerty 配列キーボードに把握する程度，実験にかかる集中力などのエラー率に影響を与える要因と比較し，キースタイルの変更は明らかな影響を与えることはできない可能性がある.
- 実験の人数が少なすぎるため, 結果分析に使用できるデータが少なすぎる.

第8章　まとめ

QWERTY 配列のキーボードはスマートフォンなどのモバイルデバイスではキーが密集しているため誤入力しやすくなる. 本研究では, スマートフォンなどのモバイルデバイスで QWERTY 配列キーボードを基に入力効率を向上させるため, キーのスタイルを調整することで新しいソフトウェアキーボードを提案した. 提案したキーボードにより, キーのスタイル（キーのサイズおよび色）に入力効率に与える影響を調査した. 実験で本研究で提案する動的にキーのスタイルを変更する手法が入力効率に与える影響を調査し, 結果としては, 予測する 4 つのアルファベットに 1 つのが正しい確率が 90%以上の場合, キーを適切に大きくすると入力速度が上がる傾向が見られ, キーの色を変えたり，大きくして目立ちすぎると，ユーザーの入力への集中を妨げ，入力速度が下がる傾向が見られた．また, 予測する 4 つのアルファベットに 1 つのが正しい確率が 100%を確保できない場合, キーのスタイルの変更はエラー率への影響が見られなかった.

謝辞

本研究を進めるにあたり，ご指導いただいた，高橋伸准教授，志築文太郎教授，川口一画助教に心から感謝いたします．特に高橋伸准教授には，研究の方向性や内容や論文の書き方などについて多くの助言と指導をいただいただけでなく，日常生活でも大きな支えとなったことに感謝いたします．重ねて感謝申し上げます．

インタラクティブプログラミング研究室の皆様には，私の研究にあらゆる面で助けていただき感謝いたします．留学生の私としては，言語の問題や情報不足に直面することが多かったのですが，研究室の皆様は私の質問にお力を貸していただきました．特に UBIQUITOUS チームの皆様には，チームゼミ，チーム研究，論文の書き方などで助けていただき感謝いたします．最後に，留学生活を支えていただき家族や友人の助けにも感謝いたします．

参考文献

[1] Katie A Siek, Yvonne Rogers, and Kay H Connelly. Fat finger worries: how older and younger users physically interact with pdas. In *IFIP Conference on Human-Computer Interaction*, pp. 267–280. Springer, 2005.

[2] Ely Rabin and Andrew M Gordon. Tactile feedback contributes to consistency of finger movements during typing. *Experimental Brain Research*, Vol. 155, No. 3, pp. 362–369, 2004.

[3] Swiftkey keyboard: The best keyboard for android & ios. Accessed: 2021-09-09.

[4] Thumbly keyboard. Accessed: 2021-09-09.

[5] Blackberry 10 keyboard setup. Accessed: 2021-09-09.

[6] Justin Cuaresma and I Scott MacKenzie. A study of variations of qwerty soft keyboards for mobile phones. In *Proceedings of the International Conference on Multimedia and Human-Computer Interaction-MHCI*, pp. 126–1. Citeseer, 2013.

[7] Panos Sakkos, Dimitrios Kotsakos, Ioannis Katakis, and Dimitrios Gunopulos. Anima: Adaptive personalized software keyboard. *arXiv preprint arXiv:1501.05696*, 2015.

[8] Muhammad Shahid Bhatti, Muhammad Arslan Ahmed, Suzu Saddia Kajiya, Abdul Qayum, Imran Latif, Abdul Karim Shahid, Rizwan Qureshi, and Sajid Ibrahim Hashmi. Mistype resistant keyboard (nexkey). In *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*, pp. 1–7. IEEE, 2018.

[9] Élvio Rodrigues, Micael Carreira, and Daniel Gonçalves. Improving text-entry experience for older adults on tablets. In *International Conference on Universal Access in Human-Computer Interaction*, pp. 167–178. Springer, 2014.

[10] Method, device, and graphical user interface providing word recommendations for text input.

[11] Tafadzwa Joseph Dube and Ahmed Sabbir Arif. Impact of key shape and dimension on text entry in virtual reality. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*, pp. 1–10, 2020.

[12] Khaldoun Al Faraj, Mustapha Mojahid, and Nadine Vigouroux. Bigkey: A virtual keyboard for mobile devices. In *International Conference on Human-Computer Interaction*, pp. 3–10. Springer, 2009.

[13] Asela Gunawardana, Tim Paek, and Christopher Meek. Usability guided key-target resizing for soft keyboards. In *Proceedings of the 15th international conference on Intelligent user interfaces*, pp. 111–118, 2010.

[14] Peter F Brown, Vincent J Della Pietra, Peter V Desouza, Jennifer C Lai, and Robert L Mercer. Class-based n-gram models of natural language. *Computational linguistics*, Vol. 18, No. 4, pp. 467–480, 1992.

[15] Android software development kit. `https://developer.android.com/sdk`. Accessed: 2023-01-09.

[16] Pressagio. `https://github.com/Poio-NLP/pressagio`. Accessed: 2023-01-09.

[17] Keith Vertanen and Per Ola Kristensson. A versatile dataset for text entry evaluations based on genuine mobile emails. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*, pp. 295–298, 2011.

[18] John Bloomer. *Power programming with RPC*. " O'Reilly Media, Inc.", 1992.

[19] Protocol buffers. `https://developers.google.com/protocol-buffers/`. Accessed: 2023-01-09.

[20] grpc: A high performance, open-source universal rpc framework. `https://grpc.io`. Accessed: 2023-01-09.

[21] Vladimir I Levenshtein, et al. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, Vol. 10, pp. 707–710. Soviet Union, 1966.

[22] Robert A Wagner and Michael J Fischer. The string-to-string correction problem. *Journal of the ACM (JACM)*, Vol. 21, No. 1, pp. 168–173, 1974.

[23] Alan Beale. The 12dicts word lists. Retrieved from http://www.wyrdplay.org/12dicts.html, 2016. June 20.

著者論文リスト

本論文に関する論文および発表

- 査読なし国内会議論文

1. LI MIAO，高橋伸．ソフトウェアキーボードの動的レイアウト調整手法．情報処理学会第 84 回全国大会，情報処理学会，2022 年 3 月，2 pages.
2. LI MIAO, 高橋伸．DStyleKeyboard: キーが動的に変化するソフトウェアキーボードの開発．第 30 回インタラクティブシステムとソフトウェアに関するワークショップ（WISS2022），日本ソフトウェア科学会，2022 年 12 月，3 pages.