ヒューマンインタフェース 78−10
（1998. 5. 15）

ポインティング精度向上のためのマウスドライバの改良

†齊藤明紀　†西田知博　‡辻野嘉宏　‡都倉信樹
†大阪大学 情報処理教育センター
‡大阪大学 大学院基礎工学研究科 情報数理系専攻

グラフィカルユーザーインターフェース (GUI) システムにおいて, マウス等でのポイントエラーは操作効率の低下や快適性の低下の大きな要因である. ここでは, ボタン操作の副作用として発生するマウスの移動と, ボタン操作を行なう指と移動操作を行なう手首および腕の動きの不一致との二つの原因に注目した.

我々は, マウスが発生する信号の履歴を解析することで利用者が意図したポイント目標座標を推測するアルゴリズムを考案した. またXウィンドウシステムのマウスデバイスドライバー内ににマウスの動きを補正する機能を追加し, 効果を確認した.

Improving Pointing Accuracy by Mouse Device-driver

†Akinori SAITOH　†Tomohiro NISHIDA
‡Yoshihiro TSUJINO　‡Nobuki TOKURA
†Education Center for Information Processing, Osaka University
‡Aftergraduate School of Engineering Science, Osaka University

In Graphical User Interface(GUI) System, a pointing error is one of the primary factor of inefficiency and unpleasantness. We focus on these two types of pointing errors. One is an unintentional mouse motion affected by button operation side effect. The other is a synchronization error between a button operation and a move operation, which come from a time lag between arm–wrist and finger movement.

We propose an algorithm to guess the coordinates that the user want to point analyzing mouse motion and button operation information history. We also added some compensatory functions into the mouse device driver in X window system.

1 はじめに

マウスに代表されるポインティグデバイスがGUIでは広く利用されている. マウスは訓練なしですぐ使えるポインティング装置であるが意図した位置と入力されたポイント位置がずれることがある.

本研究ではマウス装置からの入力の処理方式によって, ポイントエラーを低減することを目指す.

すでに, ジッタの除去処理や, クリックやダブルクリックの際にマウスが数カウント移動してもそれを短距離ドラッグと誤認しないための処理は, 多くのウィンドウシステムで一般的に実装されている.

本研究で解決を目指す問題点は, 以下の二つである.

1. 領域選択のドラッグ終了後直ちにマウスカーソルを他の位置 (特に遠方) に動かすような場合, 選択された領域が意図と異なることがある.
2. マウスボタンを押す反動でマウスが動いてしまうことがある. ピクセル単位の高精度のポインティングの際には問題である.

本研究では,UNIXとXウィンドウシステムを対象として実験を行なった.

図 1: 文字列選択

図 2: 文字列選択の失敗例 1

2 マウスの性質

まず, マウスの構造と通信方式, およびウィンドウシステム側での処理形態について調査した.

現在の主流はシリアルマウスで, 同期あるいは非同期 (調歩同期式) によるビット直列でマウスから計算機に信号が送られる. シリアルマウスの発生するデータは3バイトあるいは4バイトの長さのデータパケットで構成される. 一つのパケットには位置差分情報 (ΔX, ΔY) とボタン状態 (押されている/いない) が含まれる. 位置変化やボタンの状態の変化があるとパケットが発生される. また, 変化がひき続いて起こっている間はパケットを発生しつづける. しかし, その頻度は通信速度の限界の数分の一程度のある一定時間 (サンプリング間隔) ごとである. 位置変化もボタンの状態変化もないときには何もおくらない. よく用いられるサンプリング間隔は, 十数 ms〜数十 ms である.

ボタン操作と移動がサンプリング間隔以下の時間幅内に起きた場合, 一つのデータパケットで両方が報告される. その結果, どちらがさきに起こったのか, あるいは本当に移動しながらマウスボタンを操作したのかの区別ができない.

X でのマウス操作に関するイベントは, 以下の 3 種である.

- Motion Notify(以下 MN)
  マウスの移動を示す.
- Button Press(以下 BP)
  マウスボタンが押されたことを示す.
- Button Release(以下 BR)
  マウスボタンが放されたことを示す.

X では, ボタンとポインタ移動が同時に起こったことに対応するイベントがなく,BR/BP と MN イベントが順に生成される. たとえば XFree86 では MN イベントが先になる.

クリックに付随する意図しないマウス移動は, 短距離のドラッグと紛らわしい. ソフトウェア手法で素早く短いドラッグをクリックとみなすようにすることは既に一般的に行われている. しかし, クリックの際のポイント位置は特に補正されていない. 多くの場合クリックの対象物は十分に大きく, その必要性がないからである.

なお, 本稿ではマウス装置が計算機に送る信号における距離単位を「カウント」と称する. また, ウィンドウシステム側での距離単位は, 画面のピクセル (ドット) 数を用いる.

マウスの空間分解能は通常画面の解像度と一致する (1 カウント=1 ピクセル). しかし,X を含む多くのウィンドウシステムでは非線形 (加速機能つき) のマウス・デバイスドライバが実装されている. この場合, マウスがある一定速度以上で動いている間は, カウントとピクセル比を変え (例えば,1 カウント=3 ピクセル), 狭い作業領域でもマウスが操作できるようにしている.

3 状況調査と実験

1節で述べた問題点がどのようにおきているかを分析するため, 観察と実験を行なった.

3.1 選択エラー

マウスのドラッグによって領域選択を行なうアプリケーションは多い. 図 1はターミナルエミュレタ xterm で文字列の選択を行なった例である. このように, 反転表示によって選択範囲が示される. 図 2は, 選択に失敗した場合の例である. ドラッグ終了地点が 1 行分上にずれている.

このような失敗は単なる利用者の操作ミスの場合もある. しかし, ただしく図 1の状況になっているを目で確認してからマウスボタンを放した (つもり) に

図 3: 文字列選択の失敗例 2

もかかわらず選択エラーが起きることがある. それも, 図 3のように, マウスの手ブレでは説明できないほど大きくずれることがある.

ここで, 利用者がマウスボタンを離してからマウスを他の位置に向けて動かし始めたときに, 実際には移動イベントが先に xterm に与えられているのではないかという仮説を立てた.

xterm や kterm など X window system では, 表示された文字列の copy & paste 行なうためにキーボード操作は必要ない. ドラッグで領域を選択すると自動的にその情報がペーストバッファにおかれる. そのため, 領域選択が完了した時点ですぐさまペースト先へマウスを動かし始めて良い. そのため, 前記の現象がおきやすいと考えられる.

3.1.1 実験

仮説を検証するために実験を行なった (図 4).
実験のタスクは, 以下のようなものである.

- 画面に課題ウィンドウと目的ウィンドウの二つのウィンドウが表示される.
- 課題ウィンドウに表示された文字列をマウスドラッグで選択し, 目的ウィンドウでペースト操作を行なう.

目的ウィンドウは課題ウィンドウを基準にして上下左右の 4 つの方向に配置した場合それぞれについて上記タスクを被験者が行なう.

課題文字列長は 1, 5, 20 文字の 3 種で, 課題ウィンドウ中の 4 隅と中央のいづれかに表示される. 表示位置と文字列長の出現順序はランダムである.

被験者数は 11 で, 一人あたりのタスク回数は 300 である.

この実験では, 文字単位の選択を行なうため, 正しいドラッグ開始位置とドラッグ終了位置は点ではなく使用するフォントでの文字サイズ (8x14 ピクセル) の大きさの矩形領域となる. ペースト操作は, 現実の状況に合わせて目的ウィンドウ内であればどこで行なってもよいものとした.

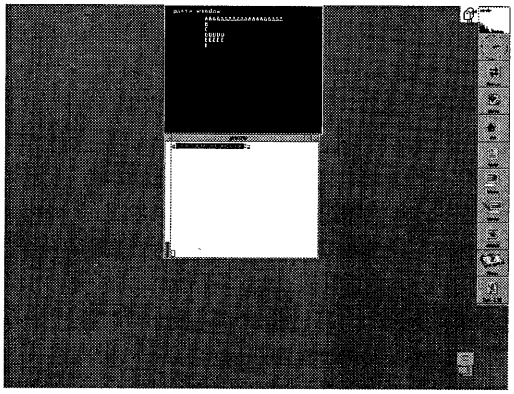

図 4: 実験中の画面

実験は, タスクの成功失敗以外に利用者の操作記録を X window system のイベントレベルですべて記録した. 画面に表示されるウィンドウはイベント記録機能を追加した kterm であり, copy&paste の操作および応答様式は実利用環境と全く同じとなる.

3.2 実験結果

選択操作の誤り率は, 約 13%であった. これは, ドラッグ開始位置が文字列先頭から外れていたものも含んだ誤り率である. 誤り事例のうちドラッグ終了位置のずれによるものは約半数 (5.5%) であった. さらにそのうち半数強 (3.9%) は, 一旦正しい位置でマウスカーソルが静止した後でマウスが動きはじめ, 正しい目的領域を外れてから BR イベントが起きているものである.

仮説で考えた種類のポインティングエラーの発生率は 3.9% であるが, その中には動き始めの MN と BR が同じタイムスタンプ値を持つものと, そうでないものがある. 後者では静止後 1〜3 回程度の MN イベントが発生した後で BR が発生している.

前者は, 2節で考察したように, ボタンを離す操作とその後のマウス移動がマウスの時間軸分解能以下の間隔で起こった可能性が疑われる.

図 5は後者の状況を表した図である. 横軸が時間,

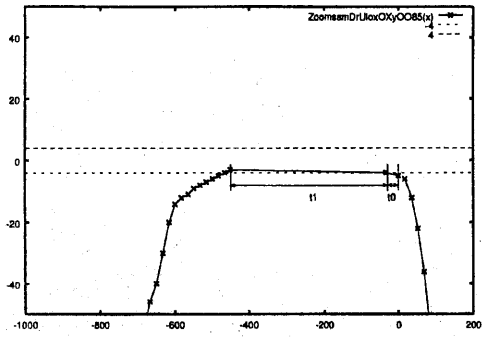

図 5: エラーの状況例

イベント種別	総数	手ぶれ発生回数	平均移動距離
Button Press	5208	584	2.5
Button Relase	2760	332	3.3

図 6: 手ぶれの分析結果

縦軸がその時点でのマウスの x 座標として, 各 MN, BR イベントの起こった時刻とその時の x 座標値を×でプロットしている. 分かりやすさのため, 横 (時間) 軸ではBRが起こった時刻を 0 としている. 縦軸は文字の中心を 0 としている. 実験で使用したフォントの横幅は 8 ピクセルであるから, x 軸の値が ± 4 以下 (点線の間) ならば文字列の最後の文字がポイントされていることになる.

この例では, ドラッグは左から右方向に行なわれ, 正しい位置にマウスが移動したあと, 約 400ms のあいだ (t1) マウスが静止している, その後マウスは左に移動し始め急激に加速して左に動いている.

BR は動き始めの MN の後 t0(約 50ms) 後に発生しており, この時マウスカーソルは目標とする文字の左隣の文字の上にある.

このことから, 利用者がマウスボタンを離してからマウスを他の位置に向けて動かし始めているつもりだが, 実際には指の応答が遅れ, マウスボタンを放し終らないうちに腕が動き始めていることが推測できる.

3.3 手ぶれ

マウスボタンのプレスおよびリリース時のマウスのぶれについて調査を行なった.

時刻	マウスからのデータ		
	ΔX	ΔY	ボタン
0	10	-5	ON
	↕ 150ms の静止		
150	1	-1	ON
180	0	-2	ON
210	0	0	OFF

図 7: マウスからの信号例

演習室での学生の X の操作記録 [1] を分析したところ, ボタン操作にともなうポインタのぶれは少なくとも 5〜10%程度の割合で発生していた.

分析は以下の要領で行なった.

目的が補正アルゴリズムの開発であるので, ボタン操作起因のぶれであかどうかが疑わしいものは排除し確実性の高いものを抽出することを目的とした.

すなわち, マウスが静止した状態でボタン操作を行なった場合のマウス位置の変動のみを調べ, 完全に静止していない状態でのマウスボタン操作データは省くことにした.

分析したものは,X window system のサーバーがクライアントアプリケーション (ウィンドウマネージャ) に対して送ったイベントである.

どのイベントも, マウスカーソルの座標とタイムスタンプ (ms 単位) を持つ. 分析したデータにはこれ以外にマウスカーソルとウィンドウ境界の横断やキーボード操作などすべての種類の X window system のイベントが含まれる.

調査および実験の結果, マウスが移動している間は,MN イベントは 10 数 ms からせいぜい 30〜40ms 以下の間隔で発生し続けることがわかっている. この最小間隔はマウスのサンプリング間隔であり, ウィンドウシステムとマウスの機種の組合せで一定の値に定まる. 今回分析したデータを取得した環境 (SONY NEWS/NEWS OS4.2.1a+) では, 30ms であった. そこで, 移動イベントが 100ms 以上発生しないということはマウスが一旦静止したとみなして良いと考えた.

また, マウスを移動させた後それに対するマウスカーソルの動いた先を目で確認し, さらに指を動かしてマウスボタンを押す (あるいは放す) には, おおむね 300ms 程度を要すると考えられる.

そこで 3 倍の安全率を見込んで, 静止した状態から

時刻	マウスからのデータ		
	ΔX	ΔY	ボタン
0	10	-5	ON
210	0	0	OFF
210	1	-1	OFF
210	0	-2	OFF

図 8: 手法 1 による補正例

100ms 以内に BP あるいは BR イベントが発生した場合を, マウスが静止した状態でボタン操作を行なったと判断することにした. このとき, 静止した時の座標とボタンイベントの発生した座標が異なるものを手ブレによるポイントエラーとみなした.

分析対象はのべ 179 名の学生がログインしてからログアウトするまでの操作記録であり, イベント総数は 4464863 である. 分析結果は図 6 の通りである.

この基準で抽出した手ぶれと思われる記録を調べたところ, イベント列としての発生形態は以下の 2 種であることが分かった.

- MN の 100ms 以上後で BP あるいは BR が起き, その座標が MN の座標とは異なる.
- MN の 100ms 以上後で MN が 1〜3 回起こり,BP あるいは BR が発生する.

後者の場合,MN イベントで報告されるマウスの移動量 ($\sqrt{\Delta x^2 + \Delta y^2}$) はほとんどの場合 (87%) は 1 である.

4 補正手法

前節で示したエラーの補正のために必要なことは以下の 3 点である.

- ボタン操作イベントごとに, 補正を行なうべきかそうでないかを識別する手法を考える.
- 利用者が意図したポイント位置を推定する.
- ポイント点補正の実装手法を考案する.

X window system では, マウスを動かしながらボタン操作を行なうことを要求するアプリケーションはほとんどない. よって, 移動中のボタン操作は補正の候補である. ただし, ウィンドウのフォーカス操作

時刻	マウスからのデータ			
	ΔX	ΔY	ボタン	
0	10	-5	ON	
150	1	-1	ON	
180	0	-2	ON	
210	-1	3	ON	*1
210	0	0	OFF	
210	1	-3	OFF	*2

図 9: 手法 2 による補正例

やウィンドウ移動など許されるポイント地点の範囲が大きな時にはマウスを動かしたままでボタン操作を行なうことはあるので, すべてが補正の対象ではない.

前節で述べた調査・実験の結果より, 停止していたマウスが動き始めた直後にあるいは動き始めと同時に行なわれたボタン操作は, 利用者は静止中に行なうことを意図したものだと考えられる.

よって, 静止していたマウスが動き始めたあと数 10〜300ms 程度以内に起こったボタン操作イベントを補正の対象とする.

一旦マウスカーソルを静止させた後で動かし始め, 動きはじめた直後 (数 10ms 後) にマウスボタンを放す (あるいは押す) という操作は X では要求されないし, 人間が意図的に特定の点をポイントするために行なうこともできない. そこで, 利用者が意図したポイント位置は, ボタン操作に先立つ静止位置である.

ポイント点の補正手法は, 二種類を実装し, 比較した.

4.1 手法 1

マウスが停止状態から動き始めた時にマウス移動データを蓄積して, X アプリケーションに渡さない. ある時間幅 t_{wait} 以内にボタン操作が行なわれたら, 蓄積した移動情報をボタン操作の次に出力する.

t_{wait} 以内にボタン操作が行なわれなかった場合には, 蓄積した情報を出力する.

この手法では, 例えば図 7のような信号をマウスが発生した場合には, それを図 8の用に変形してから X サーバーのイベント発生部に渡す.

この手法では時間幅 t_{wait} のタイムアウト処理のために, タイマーを使用する必要がある. 調査した

		補正なし	補正あり
全体		13.2%	10.6%
	ボタンを放す操作の誤り	6.9%	3.9%
	それ以外	6.3%	6.7%

図 10: 補正前後のエラー率

限りでは UNIX 用の X サーバーでは setitimer や ualarm などのタイマーサービスは使っておらず, 本手法は問題なく実装することができる.

4.2 手法 2

図 9に例を示す.

移動データはそのまま上位層に渡すが, 同時にタイムスタンプとともに記録する. ボタン操作データをマウスから受信したら, 推測した正しいポイント位置にマウスカーソルを動かす移動情報 *1 を生成してからボタンイベントを生成する. その後に,*1 をうち消す移動 *2 を生成する.

ただし, ボタン操作がマウスが動き出してから t_{wait} 以上後になって起こった時には, 意図的にマウスを動かしながらボタンを操作したものとみなして補正は行なわない.

4.3 実装と検証

上で提案した手法はどちらも C 言語で 100 から 200 行程度の処理を X サーバーに追加するだけでよい. 例えば手法 2 を実装した PC-UNIX 用 X サーバー (XFree86) では, 修正したファイルは 4 つであった.

どちらの手法でも, アプリケーションが受けとる BR および BP イベントが持つ座標値は同じとなる.

3.1.1節と同じ実験を手法 1 を組み込んだ X サーバーのもとで行なった. t_{wait}を 100ms として実験した結果, エラー率は 10.6% であり, ボタンを放す操作を原因とするエラー率は 6.9%から 3.9%へとほぼ半減した.

4.4 その他の実験結果

手法 1 に関して, t_{wait}を 100ms, 200ms, 300ms と変化させて操作感について被験者の意見を求めたところ, 200ms や 300ms では マウスカーソルの追従性に違和感があり, 特に 300ms では円滑な操作が行なえないとの感想が得られた. 100ms の場合は多くの被験者は無修正のものと同じ使用感であると答えたが, 一部にマウスカーソルの応答性の低下に気づいたものもいた.

手法 2 に関しては, マウスカーソルの応答速度の問題はないが, 一旦目標を外れたマウスカーソルがボタン操作時に戻るため, 図 3のように誤った領域を選択した状態が一瞬表示されてしまう.

手法 1,2 とも, これによる X サーバーの負荷の増大はごく少量と思われ検出限界以下であった.

4.5 考察

今回は停止した場所を利用者が意図するポイント位置と推測したが, マウスを短時間静止させたつもりでも実際は数カウントの移動が発生し続けている場合がある. このような場合には今回の手法は適用できない.

単純にマウスの静止を検出するのではなく, 静止とほぼ等しい低速度の移動や, マウスの移動方向の変化点あるいは, 移動速度が極小となる時点を検出するなどの手法を追加すれば, さらに補正範囲を広げられる可能性がある.

5 まとめ

マウスにおけるポインティング操作の際におこるエラーをとりあげ, 指と腕の動作の同期ずれによるエラーが起きていることを示した. また, ボタンに掛ける力がマウスボディに伝わったために起こるぶれについても調べた.

さらに, これらによるエラーを打ち消す手法を考案し, X サーバーに組み込むことでポインティング操作のエラー率が低下することを示した.

本稿では特にマウスを取り上げたが, 同様な特性を持つ他のポインティングデバイスにも本手法は適用可能と考えられる.

参考文献

[1] 西田 他: "GUI における実操作履歴の取得とその意図分析 ", 情報処理学会ヒューマンインターフェース研究会報告,66-2, pp.7-14(1996). 訳, アスキー出版局,1990.