Exploitation de la dynamique du geste en IHM. Application aux fonctions de transfert pour le pointage et l'extraction d'évènements discrets.

Jean-Philippe Deblonde

Exploitation de la dynamique du geste en IHM

Application aux fonctions de transfert pour le pointage et l'extraction d'événements discrets

THÈSE

présentée et soutenue publiquement le 28 septembre 2012

pour l'obtention du

Doctorat de l'Université Lille 1

(spécialité informatique)

par

Jean-Philippe Deblonde

Composition du jury

Président :	Yann Coello, Professeur	Université Lille 3
Rapporteurs :	Pierre De Loor, Professeur	Ecole Nationale d'Ingénieurs de Brest
	Jean-Louis Vercher, Directeur de Recherche	Université Aix-Marseille II
Examinateur :	James Eagan, Maître de Conférences	Télécom ParisTech
Encadrement :	Géry Casiez, Maître de Conférences	Université Lille 1
Directeur :	Laurent Grisoni, Professeur	Université Lille 1

UNIVERSITÉ LILLE 1
Laboratoire d'Informatique Fondamentale de Lille — UMR 8022
Bât. M3 – 59655 VILLENEUVE D'ASCQ CEDEX

Mis en page et composé avec LaTeX et la classe thlifl.

Remerciements

Je remercie ma famille, spécialement mon épouse et ma fille, sans le soutien desquels je n'aurai pu mener à bien cette thèse.

Le temps passé par Laurent Grisoni et Géry Casiez en encadrement, conseils, suivi et relecture a été très précieux.

Les discussions entre amis à la pause café étaient salutaires pour décompresser, trouver des idées et avancer (merci à Jeff, Loïc, Simon, Yoann et les autres du bureau et de l'équipe).

Table des figures

Table des matières

Abstract

Exploitation de la dynamique du geste en IHM.
Application aux fonctions de transfert pour le pointage et l'extraction d'événements discrets.

La dynamique du geste reste peu exploitée dans le domaine de l'interaction homme-machine alors qu'elle permet de définir des critères simples d'analyse de l'intention des utilisateurs. Sa principale utilisation porte sur les fonctions de transfert des périphériques isotoniques. Ces fonctions permettent d'établir une relation entre le geste de l'utilisateur effectué dans l'espace moteur avec le périphérique de pointage (i.e. souris) et le mouvement du pointeur dans l'espace visuel. Les fonctions de transfert utilisées par les systèmes d'exploitation modernes restent cependant fortement méconnues même si des études ont montré qu'elles ont un impact sur les performances des utilisateurs. Pour caractériser ces fonctions, nous avons développé un périphérique électronique qui simule une souris réelle et qui permet d'obtenir les fonctions de transfert de tout système. Nous avons ainsi pu obtenir les fonctions des systèmes Windows, Linux et Mac et nous avons observé des différences notables entre ces fonctions. Nous avons ensuite cherché à optimiser ces fonctions en réduisant au maximum le nombre de leurs degrés de liberté. En faisant varier de manière exhaustive ces paramètres, nous avons ainsi pu observé des différences significatives qui montrent qu'il existe une famille de fonctions pour lesquelles les performances sont optimales. Enfin, nous nous sommes intéressés à l'utilisation de la dynamique du geste dans un autre contexte, celui de la génération d'événements discrets, pertinent dans le cadre des interfaces sans contact de type Kinect. Nous avons ainsi montré qu'il est possible de distinguer la sélection du déplacement dans une tâche d'interaction 3D. La technique développée est ensuite améliorée dans le cas des cibles de petites tailles et son application dans un contexte d'interaction 2D est évaluée.

Exploiting the dynamics of the gesture in HCI.
Applications to the transfer functions for pointing and extracting discrete events.

The dynamic of the gesture is not used a lot in the domain of human-computer interaction, while it can help to define some simple heuristics to analyze the intents of users. Its first use is related to transfer functions for isotonic devices. Those functions establish a relationship between the gesture of the user, executed in motor space with the pointing device (i.e. a mouse), and the movement of the pointer in the visual space. The transfer functions used by modern operating systems are still not well known, even if some studies have shown they have an impact on user's performance. To characterize those functions, we have developed an electronic device which simulates a real mouse, and that

can retrieve the transfer functions of any system. We have then exposed the functions of the Windows, Linux and Mac systems, and we observed some visible differences between those functions. We have then tried to optimize those functions by reducing their number of degrees of freedom. By varying those parameters in an exhaustive way, we have then observed some significant differences that show there is a family of functions for which performances are optimal. Finally, we were interested in the use of the dynamic of the gesture in another context : the generation of discrete events, useful when dealing with contact-less interfaces, like the Kinect. We have shown that it is possible to distinguish between picking and drag'n'drop in a 3D interaction task. The technique is then improved in the case of small size targets, and its application in a 2D interaction context is evaluated.

Introduction

L'utilisateur manipule les périphériques d'entrées afin de manifester une intention auprès du système, qui lui envoie en retour l'information traitée sur un périphérique de sortie (écran, imprimante, haut-parleurs, ou périphériques plus spécialisés encore, destinés aux autres sens humains). L'évolution de l'informatique a apporté de nouveaux périphériques à chaque passage à une nouvelle métaphore d'interaction. Les terminaux de type texte proposaient une interaction en ligne de commande, au clavier. L'utilisation d'interfaces graphiques a introduit la notion de pointage sur des cibles, icônes, boutons et menus. Parmi les différents systèmes de pointages disponibles (joystick, souris, crayon optique ou tablette graphique, par exemple), l'usage de la souris s'est répandu rapidement. Ce périphérique permet de mettre en correspondance le déplacement de la main sur la table, avec le déplacement d'un curseur à l'écran. Cette mise en correspondance peut se faire de plusieurs manières (mise en correspondance constante ou dynamique), et avoir une influence sur les performances de l'utilisateur, en termes de précision, de temps, et de contraintes neuro-musculaires. La première partie de notre travail s'est donc portée sur la définition des problèmes rencontrés avec les techniques de pointage actuelles. Pour ce faire, nous nous sommes intéressés tout d'abord à la dynamique physiologique du geste, puis l'exploration de l'ensemble de techniques existantes de pointage et de facilitation au pointage. La solution utilisée dans le domaine de l'Interaction Homme-Machine pour mettre en correspondance le déplacement du périphérique avec celui d'un curseur à l'écran est celle d'une fonction de transfert.

Cette fonction peut être construite à l'aide de plusieurs profils. Les premiers systèmes à les exploiter proposaient une accélération constante en dessous d'une certaine vitesse, et une autre accélération plus rapide, dès que la vitesse dépassait un seuil. Les systèmes actuels, offrant des surfaces d'affichage beaucoup plus grandes qu'à l'époque, ainsi que des souris et périphériques de pointage beaucoup plus précis. Bien que les systèmes grand public récents proposent une accélération non-linéaire (à faible vitesse, la distance parcourue à l'écran est plus faible que la distance parcourue sur la table, alors qu'à plus grande vitesse, l'inverse se produit), il n'existe pas d'informations disponibles dans la littérature quand à la construction optimale de ces fonctions. De plus, les systèmes d'exploitations commerciaux ne permettent pas d'accéder à leur code source, afin d'en vérifier l'implémentation logicielle.

Notre travail est structuré en plusieurs parties, de détermination de ces fonctions de transfert existantes, suivie de leur comparaison, pour finir sur une méthodologie de construction d'une famille de fonctions optimales pour une série de contraintes données. Nous montrerons aussi comment utiliser les données obtenues dans l'état de l'art et les

informations récupérées des différentes évaluation afin d'augmenter le geste de pointage pour permettre le clic et la sélection de cibles sans avoir accès à un bouton physique.

Notre premier chapitre propose un état de l'art, et une classification des techniques disponibles pour améliorer la tâche de pointage dans une interface graphique.

Le deuxième chapitre présente plus en détails les souris comme périphérique, ainsi que la réalisation d'une "fausse" souris, nous permettant d'accéder aux courbes d'accélérations des systèmes actuels. Cette souris simule l'électronique d'un vrai périphérique, pour pouvoir envoyer au système des informations de déplacement de manière contrôlée. Nous pouvons alors mesurer en retour le déplacement du pointeur à l'écran pour reconstruire les courbes d'accélération du système.

L'expérience menée au troisième chapitre permet de présenter une méthodologie pour comparer entre elles les courbes d'accélération de différents systèmes d'exploitation (Windows, Mac OS, Linux). Cette comparaison nous permet d'établir certaines tendances quand au profil générique d'une courbe d'accélération, ainsi que la plage de valeurs dans laquelle celle-ci peut s'inscrire pour offrir de bonnes performances à l'utilisateur. Le but final étant d'optimiser ces courbes d'accélération. Ces courbes sont-elles compatibles avec l'analyse du geste d'interaction ? Ce chapitre présente également une expérience réalisée afin d'explorer l'éventail des valeurs possibles pour une fonction de transfert, et ainsi de pouvoir construire une famille de courbes offrant de bonnes performances.

Le quatrième et dernier chapitre présente une technique pour sélectionner des cibles en deux dimensions, à l'aide d'un périphérique qui ne dispose pas de boutons, ou lorsque l'utilisateur n'est pas en mesure d'en utiliser, dans le cadre d'un handicap, par exemple. Nous détaillons les informations de l'état de l'art et des expérimentations précédentes qui ont permis la construction de cette technique. Ce chapitre s'articule en deux parties, avec tout d'abord une présentation du travail déjà publié, suivi d'une deuxième partie qui étend se travail à d'autres contextes d'interaction.

L'équipe MINT d'Inria et du LIFL (USTL-CNRS) travaille sur les projets d'Interaction Homme Machine. C'est dans le cadre de la collaboration avec le réseau de clinique Hopale, que nous présentons ce mémoire de thèse.

Chapitre 1

Physiologie du geste et interaction : un état de l'art

Heidegger définit la relation de l'homme à l'objet comme naturelle. L'outil devient un prolongement du corps, une extension du cerveau, utilisé pour accomplir une tâche. L'observation de l'outil et de la matière travaillée n'intervient que lorsqu'une dissonance cognitive s'installe, que l'outil ne se comporte plus comme l'être l'attend, mais se transforme en obstacle pour atteindre le but.

Dotov *et al.* démontrent notamment cela avec une simple expérience (dans [38], il propose une série de tâches motrices de pointage - au milieu de l'expérience, le curseur de la souris se met à "traîner" derrière sa position normale), que lorsque la souris d'un ordinateur ne se comporte pas de manière "normale", les utilisateurs perdent le fil de leur travail, et leur efficacité diminue.

L'acte de pointage sur une cible peut paraître une tâche motrice triviale. L'analyse de la boucle physiologique constituée par l'œil, la main et le système nerveux qui les relie nous montre une complexité plus grande, que les chercheurs ont voulu rationaliser à travers plusieurs lois empiriques. Le temps nécessaire pour réaliser un mouvement de pointage peut être utilisé pour mesurer la capacité de transmission d'information du système moteur humain.

Dans ce chapitre, nous détaillerons dans une première partie l'origine physiologique du geste de pointage, et les lois qui en découlent. Dans une deuxième partie, nous aborderons les techniques d'interaction développées pour exploiter et faciliter ce geste de pointage.

1.1 Boucle physiologique

Le geste de pointage prend place au sein d'une boucle physiologique, qui va de la perception de la cible par le système nerveux, à la réalisation du geste par le système neuromusculaire.

1.1.1 L'origine des mouvements de pointage

La compréhension de la neuromotricité du geste s'est faite en plusieurs étapes, avec les travaux de Woodworth comme point de départ. D'autres modèles plus complexes ont ensuite été construits, et la compréhension des différences entre les tâches de pointage simple, pointage discret, et les tâches de pointage répétitifs s'est affinée. Nous détaillons dans les points qui suivent cette découverte historique progressive de la physiologie et de ces lois du mouvement.

Modèle Phase Balistique/Phase Corrective

Woodworth, dans sa thèse de 1899 sur *La précision des mouvements volontaires*[91], propose un modèle pour le pointage humain, dirigé vers un but. Il construit une expérience durant laquelle des sujets doivent tracer une ligne au crayon entre deux repères éloignés, à travers une fente. Un papier gradué défilant sous l'espace de travail permet d'enregistrer les informations de mouvement. Les sujets devaient effectuer le mouvement d'aller-et-retour en suivant les battements d'un métronome fonctionnant à différents tempos. Dans l'une des expériences, les gestes devaient être réalisés les yeux ouverts, dans l'autre cas, les yeux étaient fermés.

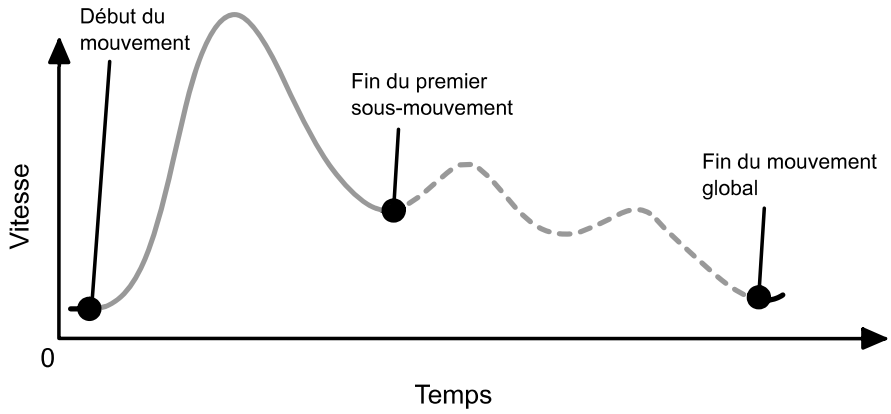

FIGURE 1.1 – Décomposition d'un mouvement de pointage en phase balistique et phase corrective, adapté de [29]).

En analysant les profils de vitesse, il postule qu'un geste dirigé vers une cible est constitué de deux phases. La première phase est dite *balistique*. C'est un mouvement rapide, non contrôlé par le système nerveux au-delà de l'impulsion initiale, et qui couvre la plus grande partie de la distance vers la cible et y amène l'effecteur (ici le bras). La deuxième phase, dite *corrective*, sert à acquérir la cible. Le sujet ralentit et la variabilité du profil de vitesse augmente. Ces oscillations sont alors interprétées comme un ajustement itératif de la trajectoire, et dans cette phase s'effectue alors un contrôle en ligne sensorimoteur pour provoquer ces réajustements (voir Figure 1.1). Les propriétés cinématiques importantes de ce geste sont alors représentées par une courbe en forme de cloche, avec la magnitude et le moment auquel se produit le pic de vélocité déterminé par l'impulsion initiale. Ce pic de

vitesse est un indicateur de l'amplitude finale du mouvement, et se produit généralement avant la moitié du parcours (40%, d'après [66]).

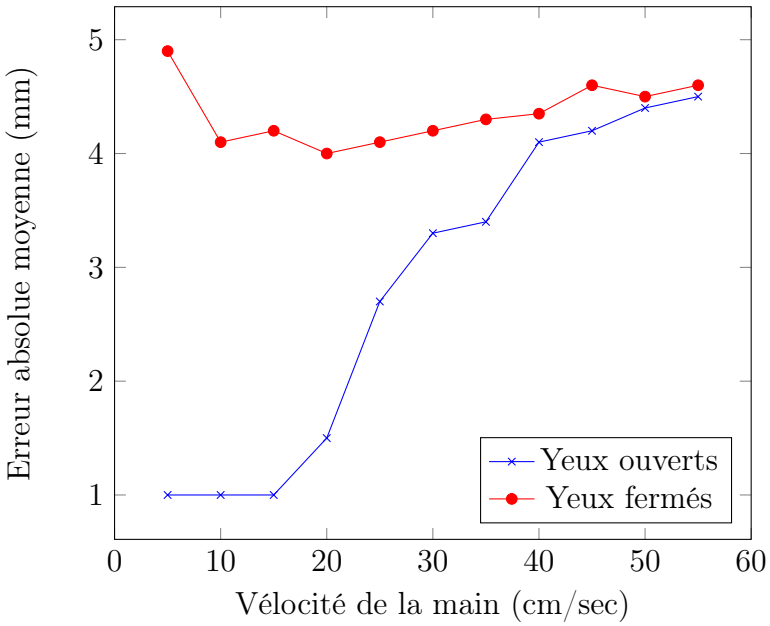

FIGURE 1.2 – Erreur absolue moyenne des mouvements de la main effectués par les sujets avec les yeux ouverts ou fermés. (A partir de Woodworth, 1899[91].)

Lorsque les sujets ont les yeux fermés, l'erreur reste plus ou moins constante quand la vitesse augmente. Quand les sujets ont les ouverts, l'erreur diminue quand la vitesse diminue. L'hypothèse de Woodworth sur ces résultats est celle d'un mouvement entièrement programmé à l'avance dans le cadre d'un mouvement "yeux fermés" (*feedforward*). Lorsque les yeux sont ouverts, un *contrôle en ligne* est possible pour corriger le geste à vitesse réduite (*feedback*). Si la vitesse imposée pour le geste s'accélère, la précision du geste diminue au point de rejoindre celle d'un mouvement réalisé "yeux fermés" (voir Figure 1.2).

La présence de variations dans le profil de vitesse sur la fin du geste n'est pas complètement expliquée dans le modèle de Woodworth. Ce modèle est raffiné avec le modèle des *Corrections Itératives*.

Modèle des *Corrections Itératives*

D'après le modèle des corrections itératives de Crossman *et al.* ([37]), ce mouvement s'articule en deux parties, avec en deuxième partie un mouvement constitué d'une série de sous-mouvements discrets, chacun se rapprochant de 50% de la cible, et amorcé par un feedback indiquant que la cible n'est pas encore atteinte. Le modèle prédit une augmentation linéaire du temps nécessaire pour atteindre la cible lorsque l'Indice de Difficulté augmente (en accord avec la Loi de Fitts, détaillée dans la section 1.2.0.0), mais il faut alors considérer que chaque correction prend un temps constant ([72]). Bien que l'analyse des trajectoires des mouvements effectués lors des expériences confirme cette

théorie, le modèle des corrections itératives ne semble pas correct. En effet, il n'y a pas toujours de sous-mouvements discrets observables, et quand il sont visibles, ils n'ont pas toujours un durée constante ([79],[65]). De plus, ces sous-mouvements ne parcourent pas une proportion constante de la distance restante à la cible ([65]).

Schmidt *et al.*, à l'aide d'une expérience, cherchent à expliquer ces sous-mouvements en s'appuiant sur la variabilité de l'impulsion initiale.

Modèle de la Variabilité de l'Impulsion - *Loi de Schmidt*

Schmidt *et al.* ([98]) proposent une expérience différente de celle de Fitts. Alors que les sujets de l'expérience de Fitts devaient atteindre une cible le plus vite possible, mais dans le temps qu'il choisissaient eux-mêmes, les sujets de l'expérience de Schmidt doivent atteindre leur cible dans un temps prédéterminé, en essayant de minimiser les variations spatiales sur le point d'arrivée. En 200 millisecondes, les sujets doivent atteindre une cible distance de 10 à 30 centimètres, un intervalle de temps trop petit pour permettre un contrôle direct. La déviation standard des points d'arrivées, L_e, augmente avec la distance D à parcourir, et décroît avec le temps T nécessaire au mouvement, et s'exprime dans l'équation 1.1 :

$$L_e = k\left(\frac{D}{T}\right) \tag{1.1}$$

avec k une constante.
D'après Schmidt *et al.*, le bras est "lancé" par une impulsion neuromotrice envoyée au muscle. Cette impulsion fait exercer au muscle une force lors de la première moitié de la trajectoire. Dans la deuxième phase de la trajectoire, le bras "atterrit" sur la cible. Si on utilise plus de *force* pour parcourir plus vite la distance, l'écart du point d'arrivée à la cible sera plus important. Ce modèle s'applique aux mouvements rapides, mais ne donne pas d'explications quand aux corrections basées sur un feedback. De plus, Meyer *et al.* ont trouvé des erreurs dans la méthode utilisée pour dériver la loi de Fitts ([86].

Suite à ces travaux, le modèle *phase balistique/phase corrective* a été affiné par le modèle hybride de *l'Impulsion Initiale Optimisée* de Meyer, que nous décrivons ci-dessous.

Modèle de l'Impulsion Initiale Optimisée - *Loi de Meyer*

Meyer *et al.* proposent le Modèle de l'Impulsion Initiale Optimisée (1988)[85] comme une synthèse des modèles proposés depuis les travaux initiaux de Woodworth. Ce modèle présente le pointage comme un compromis entre la vitesse nécessaire pour atteindre la cible et la déviation du point d'arrivée autour de la cible. Avec un geste rapide, l'utilisateur se rapproche plus vite de la cible, mais avec une déviation plus large, ce qui entraîne une faible probabilité de toucher la cible. Par hypothèse, l'utilisateur réalise un premier mouvement qui le rapproche de la cible. Si le geste arrive sur la cible, la tâche est finie, sinon un deuxième mouvement doit être effectué, et ainsi de suite (voir Figure 1.3).

La relation entre le temps T nécessaire pour atteindre la cible, la distance D à la cible, la largeur L de cette cible et le nombre n de sous-mouvements, est résumée par Meyer avec la formule :

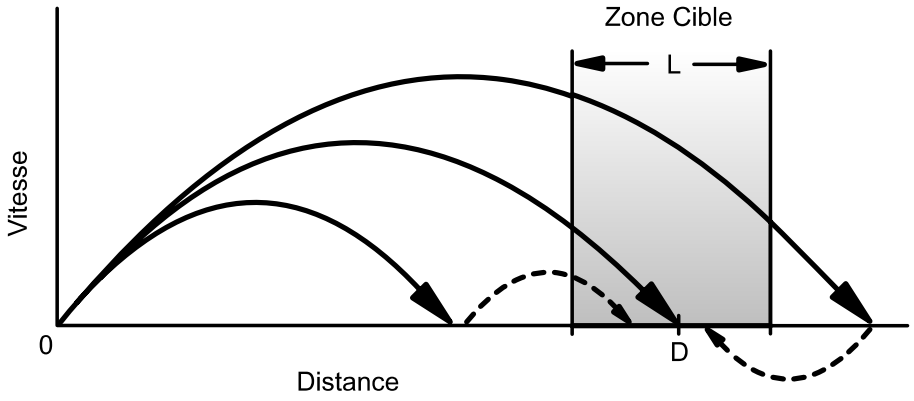

FIGURE 1.3 – Séquence de sous-mouvements vers la cible (modèle de Meyer *et al.*, adapté de [85]).

$$T = a + b \cdot n\left(\frac{D}{L}\right)^{\frac{1}{n}} \tag{1.2}$$

avec a et b des constantes.
Nous verrons dans la section 1.2.0.0 comment cette équation est une généralisation de la loi de Fitts.

Le modèle de Meyer réunit les deux types de mouvements : rapides sans correction (loi de Schmidt) et plus lent, avec plusieurs corrections (Fitts).

Ce modèle est dérivé d'un mouvement rythmique répété, et non d'un geste discret. La distinction entre les deux mouvements est nécessaire, et nous la détaillons ci-après.

Tâche répétitive et tâche discrète

L'expérience originelle de Woodworth est une tâche de pointage rythmique réciproque. Cette répétition est centrale dans le paradigme de Woodworth. En effet, une tâche répétitive ne demande que peu de charge cognitive dans son déroulement, l'attention se porte sur les actions motrices. Dans le fonctionnement journalier, peu de tâches présentent ce type d'action répétitive, et les manipulations d'interfaces graphiques n'en font pas partie. Ces tâches là sont discrètes, et sont souvent précédées et suivies d'actions de types différents.

Par exemple, Fitts montre en 1964 ([48]) que les gestes discrets ont un plus grand *Indice de Performance*. Cela peut s'expliquer notamment par des décisions nécessaires, qui sont connexes au geste, comme la recherche visuelle de la cible. Ces différences entre une tâche de pointage discret et une tâche de pointage réciproque doivent être prise en compte, notamment lors de l'interprétation des expériences, en ne considérant pas les tâches répétitives comme la simple concaténation de tâches discrètes.

Cinématique d'un pointage discret et cinématique d'un pointage réciproque

Un geste de pointage discret correspond à un seul mouvement de pointage, partant d'un point de départ et arrivant sur une cible. Ce mouvement commence et termine le pointage avec la main du sujet à l'arrêt. Dès lors, la vitesse et l'accélération sont nulles au début et à la fin du geste.

Le profil de vitesse du geste est symétrique, "en cloche". Le profil d'accélération comporte deux phases opposées. La première phase, d'accélération, se finit sur une accélération nulle quand le pic de vitesse est maximal. La deuxième phase, de décélération, dissipe l'énergie du mouvement pour ralentir celui-ci. La position du membre effecteur atteint son maximum lorsque la vitesse et l'accélération sont nulles (voir Figure 1.4).

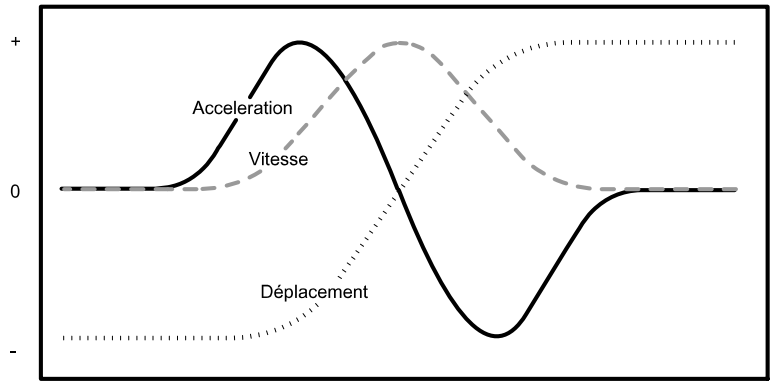

FIGURE 1.4 – Représentation des profils de vitesse, de déplacement et d'accélération d'un mouvement de pointage discret, adapté de [55]).

Une succession de mouvements de pointage discrets peut se voir comme une concaténation de ces profils, retournés symétriquements, avec la même organisation cinématique. Dès lors, la phase d'accélération est bien distincte de la phase de décélération (Guiard, [55]).

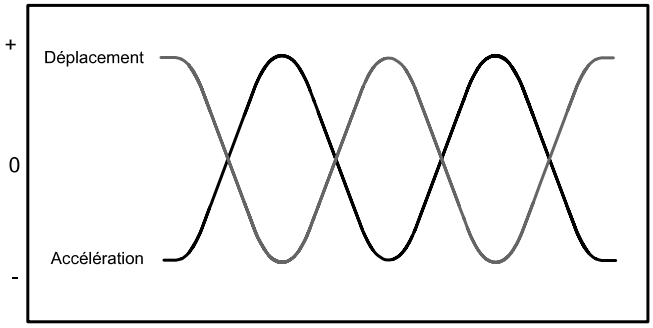

FIGURE 1.5 – Représentation des profils de vitesse et de déplacement d'un mouvement de pointage réciproque, avec un faible Indice de Difficulté (adapté de [44]).

Lors d'un mouvement de pointage réciproque, les deux phases de décélération et d'ac-

célération au point de revirement sont fusionnées : lorsque le membre effecteur arrive à l'arrêt, l'accélération est maximale. Dès lors, le mouvement observé dans une tâche de pointage réciproque est un mouvement continu (pour un Indice de Difficulté faible).

Guiard ([55]) conclut que les mouvements réciproques ne peuvent être assimilés à la concaténation de mouvements discrets. En effet, le cerveau anticipe le prochain déplacement de la main lors d'un mouvement répétitif, les profils d'accélération et de vitesse sont donc, en partie, différents des profils présents lors d'une tâche de pointage simple.

Nous pouvons maintenant chercher à comprendre comment le cerveau anticipe et planifie ces gestes de pointage.

Planification du déplacement de la main

Certaines informations connexes à la stricte cinématique du geste humain sont nécessaires pour caractériser ce geste, que ce soit au niveau de l'organisation neurologique, ou au niveau perceptif.

Une série d'études s'est intéressée à l'organisation articulaire des gestes de pointage sans contraintes de précision. Ainsi, sans ces contraintes de précision, mais en appliquant une contrainte d'amplitude, le mouvement se caractérise par un profil de vitesse symétrique invariant quelle que soit cette amplitude (Morasso notamment, [87]). Dès lors, lorsque nous ne demandons pas d'être précis quand à l'arrivée sur la cible, mais en forçant l'amplitude du geste à être constante, le profil de vitesse devient symétrique (il n'y a plus de mouvements correcteurs en fin de geste).

Morasso, en enregistrant le déplacement de la main lorsque des sujets pointent des cibles en deux dimensions, remarque que la main a tendance à se déplacer en ligne droite (ou à concaténer des segments de droite). Il en conclut donc que le système nerveux planifie les mouvements de la main en terme de position de cette main, avant l'initiation du geste (des coordonnées extrinsèques et non des coordonnées intrinsèques, qui utiliseraient les relations entre les muscles et les articulations, en termes d'angles et de moments). Il semble donc que le cerveau met en place le mouvement du bras et de la main en calculant une série de positions dans l'environnement qu'il perçoit et qui l'entoure, plutôt que d'utiliser la perception interne des mouvements relatifs des muscles et segments osseux entre eux.

Keele et Posner (1968,[71]) concluent de leurs expériences qu'il faut au moins 200 millisecondes pour utiliser la vue afin de corriger un geste de pointage. Cette valeur limite est confirmée dans les travaux qui suivent (Carlton, 1981[23] ; Zelaznik, Hawkins & Hisselburgh, 1983[108]).

Afin de déterminer quel sens, de la vision ou du toucher est plus important dans la construction des gestes, différentes expériences sont menées. Les expériences de Gibson en 1933 [51] et de Rock & Harris en 1967 [92] montrent la prépondérance de la vision sur le toucher dans la proprioception et dans la perception extérieure des objets et distances. Müsseler *et al.*([88]) montrent que les participants devant réaliser une trajectoire sur un écran en ne voyant pas leur main, alors qu'un facteur de gain modifiait fortement les gestes, étaient incertains quand aux trajectoires réellement effectuées par ledit membre. Le système kinesthésique ne semble pas apporter suffisamment d'informations par rapport au système visuel, ce qui peut corroborer les observations de Morasso sur l'utilisation par le système nerveux d'un positionnement extrinsèque. Dans ce cas, il est normal que

la vue, qui permet d'évaluer l'environnement, prenne plus d'importance que le système kinesthésique.

Tout en ayant connaissance de la prépondérance de la vue sur le toucher pour organiser le geste, il faut pouvoir décomposer ce dernier en ses phases constituantes. Bohan *et al.*([15]) établissent alors un ensemble d'heuristiques pour séparer chaque phase du geste. La fin d'un geste de pointage est défini au moment où la vitesse enregistrée atteint 2% du pic de vélocité global. Le début du geste de pointage est défini de la même manière, mais à 5% cette fois ci, afin d'éliminer une perturbation présente sur de nombreux essais, et attribuée à la friction avec la surface de la table. Les sous-mouvements sont décomptés quand (a) la différence entre le pic de vélocité local au sous-mouvement et les minimas qui l'entourent dépassent 15% du pic de vitesse global (b) les minimas entourant ce sous-mouvement sont en-dessous de 50% du maximum local et sous 15% du pic de vélocité global.

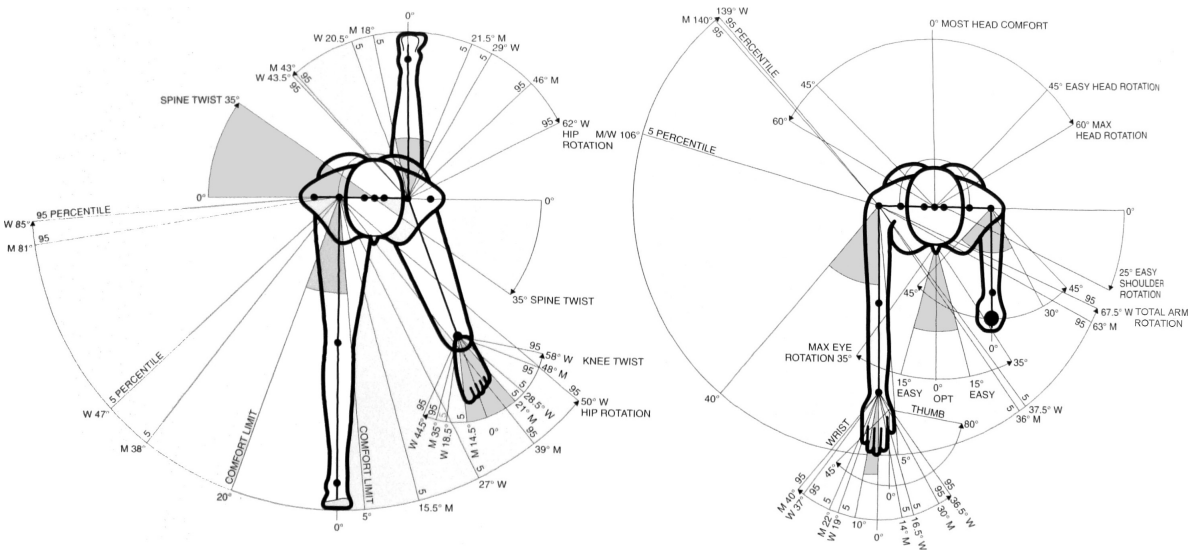

FIGURE 1.6 – Les zones de confort pour les mouvements articulaires sont représentées en grisé. (Lors du design d'une interaction, il est conseillé de rester dans l'angle minimum des 5 percentiles. D'après [5].)

Après avoir regardé la planification au niveau du cerveau, il faut également s'attarder sur les membres supérieurs du corps (bras et main), supports du geste.

1.1.2 Physiologie du geste

Le geste de pointage à l'aide d'une souris, va entrainer l'utilisation de groupes musculaires différents suivant la distance à parcourir sur la table. De grandes différences d'amplitude de mouvements entraînent des changement biomécaniques importants [54, 77].

Les mouvements de grande amplitude sont majoritairement réalisés à l'aide des articulations de l'épaule et du coude, alors que les mouvements de petite amplitude sont réalisés à l'aide des articulations des poignets et des doigts. Les études réalisées par Balakrishnan

et al. [9], Langolf & Chaffin [79], ou Zhai *et al.* [109] montrent de larges variations dans la relation Indice de Difficulté/Temps de déplacement suivant les groupes articulaires utilisés pour réaliser une tâche de pointage. Bohan *et al.*([15]) montrent que les mouvements impliquant les doigts, pour réaliser les petits mouvements à proximité des cibles, sont le facteur limitant dans l'acquisition des cibles. En effet, les doigts sont naturellement conçus pour la réalisation de tâches motrices fines, alors que la souris n'autorise qu'une prise "en mâchoire", qui limite l'utilisation des doigts. Guiard *et al.*[54] montrent d'ailleurs que l'utilisation d'un pointeur, dont la prise en main mobilise plus les articulations des doigts, entraîne de meilleures performances sur les mouvements à petite échelle en amplitude.

La durée d'un sous-mouvement est définie par le temps de réaction du système physiologique [1]. Card *et al.*([22]) définissent le modèle de traitement humain dans l'équation 1.3, avec le temps nécessaire pour un sous-mouvement T_{subM} :

$$T_{subM} = T_p + T_c + T_m \tag{1.3}$$

T_p, T_c et T_m représentent respectivement le temps nécessaire aux processus perceptuels, cognitifs, et moteurs.
L'équation donne une approximation du temps nécessaire à l'utilisateur pour observer le progrès du déplacement T_p, décider d'une correction T_c, et effectuer cette correction T_m. Pour un utilisateur de référence ([22]), cela donne un temps de 240 millisecondes pour réaliser cette observation et effectuer la correction en conséquence.

Après avoir regardé le fonctionnement du corps, permettant les gestes de pointage, nous pouvons regarder les lois mathématiques et physiques qui formalisent ce fonctionnement.

1.2 Lois motrices invariantes

Afin de modéliser le geste de pointage humain, plusieurs lois motrices nous sont utiles. Ces lois vont nous permettre de déterminer à l'avance, en connaissant les distances et tailles des cibles présentent dans une tâche de pointage, quels vont être les temps nécessaires pour réaliser cette tâche, ainsi que les taux d'erreurs que nous pouvons obtenir en retour. Ces lois sont regroupées ici, car elles sont toutes conséquences de l'observation du geste de pointage, même si elles ont pu être exploitées dans d'autres configurations gestuelles.

Loi de Hick

Les travaux de Shannon et Weaver en 1949 ([99]) sur la théorie de l'information servent de fondement à la théorie de la communication. En prenant analogie avec le téléphone, Shannon et Weaver définissent la quantité d'information passant sur un canal de communication. Dans ce modèle, l'information permet de réduire l'incertitude afin de prévoir quel événement va se produire. Dès lors, l'information correspond à la quantité d'incertitude levée par la réalisation d'un événement (dans ce modèle, nous ne prenons pas en compte

1. Un sous-mouvement, comme expliqué en partie1.1.1 est une des sous partie du mouvement, soit la phase ballistique ou une des phases correctives

la valeur et la signification dudit événement). L'information apportée par l'événement est d'autant plus grande que sa probabilité d'apparition est faible mais non nulle. Shannon et Weaver formalisent ce modèle dans l'équation suivante :

$$H = \log_2 \left(\frac{1}{p} \right) \tag{1.4}$$

avec H représentant l'information, et p la probabilité d'apparition de l'événement. Le *bit* (***bi****nary digi****t***) est l'unité d'information la plus petite.
Un canal de transmission réel est forcément bruité. Dès lors, la capacité de transmission d'information d'un canal se mesure en bit/sec, correspondant aux nombre de bits par unité de temps, moins le bruit.

Hick et Hyman (1952 [57] et 1953 [62]) appliquent la théorie de l'information dans le cadre de la prise de décision. La loi de Hick mesure le temps qu'il faut à un sujet pour prendre une décision quand plusieurs possibilités de choix lui sont offertes. Ce temps de réaction augmente de manière logarithmique avec le nombre d'alternatives possibles. Cette loi s'exprime ainsi :

$$T = b \cdot \log_2(n) \tag{1.5}$$

avec T le temps de réaction, b une constante mesurée empiriquement, et n le nombre de choix possibles.
Pour n événements équiprobables, la probabilité de chaque événement est :

$$p = \frac{1}{n} \tag{1.6}$$

Nous pouvons alors écrire l'équation 1.5 comme ceci :

$$T = b \cdot \log_2 \left(\frac{1}{p} \right) \tag{1.7}$$

qui est similaire à l'équation 1.4.

De même, la loi de Hick est proche, dans sa forme, à la loi de Fitts. Une manière de comprendre la forme logarithmique de la *Loi de Hick* est de voir que les gens divisent l'ensemble des choix possibles en sous-catégories, supprimant la moitié des choix à chaque étape, plutôt que de considérer chacun des choix un par un, ce qui demanderait un temps linéaire.

Loi de Fitts

Sur les travaux de Woodworth détaillés précédemment, Fitts a construit une expérience en 1954 pour mettre en relation le temps nécessaire pour atteindre une cible, ainsi que la distance à celle-ci et sa largeur, pour un mouvement à une dimension. Il demande aux sujets de déplacer un stylet alternativement entre deux cibles aussi vite que possible, alors que la distance entre les cibles et largeur de celles-ci varient. Fitts note que le temps augmente lorsque la distance entre les cibles augmente, et également lorsque la largeur de la cible diminue. Par rapport à Woodworth qui ne proposait qu'une contrainte

en amplitude, Fitts ajoute une contrainte de précision. La relation est résumée dans l'équation 1.8, dite formulation de *Shannon-MacKenzie* :

$$T = a + b \cdot \log_2\left(\frac{D}{L} + 1\right) \tag{1.8}$$

avec T le temps moyen pour effectuer le mouvement, a et b des constantes empiriques venant d'une mesure des données, D la distance au centre de la cible et L la largeur de la cible mesurée selon l'axe de mouvement.
La formulation de MacKenzie permet d'éviter les *Indices de Difficultés* négatifs.

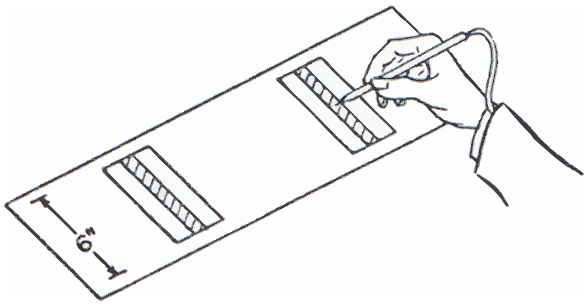

FIGURE 1.7 – Dispositif de l'expérience de Fitts en pointage réciproque

L'*Indice de Performance* (*IP*), exprimé en *bits/seconde*, caractérise la vitesse avec laquelle le pointage peut être réalisé, de manière indépendante des cibles en présence. *IP* peut s'exprimer de deux manières différentes : $IP = 1/b$, qui a le désavantage d'ignorer l'effet de a pour la première forme, et $IP = ID_{moyen}/MT_{moyen}$ pour la deuxième forme, avec le défaut d'utiliser une définition arbitraire de « moyen ». Le terme $\log_2\left(\frac{D}{L} + 1\right)$ est appelé l'*Indice de Difficulté* (*ID*) de la tâche, et a une unité en *bits*. Cet *ID* correspond à la quantité d'information à traiter dans le cadre de cette tâche de pointage. Le temps T augmente linéairement avec l'Indice de Difficulté. Nous avons dès lors un compromis *vitesse-précision*.

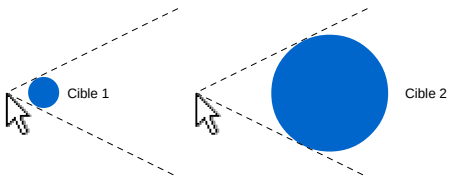

FIGURE 1.8 – Pour conserver le même indice de difficulté (*ID*), alors que la distance à la cible augmente, le système doit augmenter la taille de cette cible.

La loi de Fitts (Equation1.8) apparaît ici comme un cas particulier de la loi de Meyer (1.2) : lorsque n, le nombre de sous-mouvements, décroît, T augmente fortement. Lorsque n approche l'infini, nous pouvons réécrire la formule 1.2 comme ceci :

$$T = a + b \cdot \log_e \left(\frac{D}{L} \right) \tag{1.9}$$

Cette équation rappelle évidemment la loi de Fitts (l'utilisation de logarithme en base e plutôt qu'en base 2 n'a pas de conséquence ici). Bien que les sujets n'effectuent pas une infinité de sous-mouvements, la loi de Fitts, lorsqu'elle représente une condition limite, donne une méthode précise pour adapter les résultats de nombreuses expériences au compromis *vitesse-précision*.

L'équation (1.8) s'avère valable dans un grand nombre de cas d'utilisations : des mouvements de pointages discrets ([48]), le transfert de chevilles à insérer dans des anneaux ([3]), actionner un joystick ou tourner une poignée pour déplacer un curseur sur un écran ([65],[86]), lancer des fléchettes sur une cible ([73]), réaliser des tâches de pointage sous l'eau ([74]) ou manipuler des objets sous un microscope ([79]).

Dans le cadre informatique, la loi de Fitts peut être utilisée de manière *prédictive* ou *comparative*. La méthode prédictive permet de connaître à l'avance le temps que mettra l'utilisateur pour atteindre tel ou tel élément de l'interface ou si l'Indice de Difficulté de la tâche n'est pas trop élevé. La méthode comparative permet de classer entre eux des techniques d'interaction ou des périphériques de pointage pour une même tâche à réaliser. C'est de cette manière que Card, en 1978 ([100]), utilise l'*Indice de Performance* pour comparer différents périphériques de pointage et voit la souris arriver en tête.

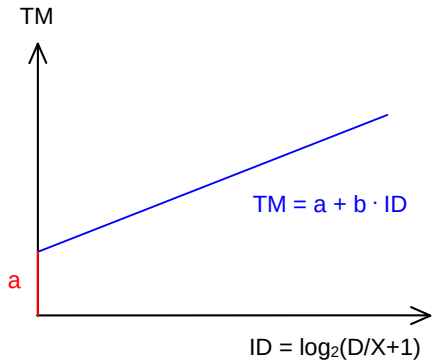

FIGURE 1.9 – Expression graphique de la loi de Fitts, avec visualisation des variables a et b.

Les différents versions de la loi de Fitts nous donnent :

Version logarithmique

— Fitts (1954)

$$T = a + b \cdot log_2 \left(2\frac{D}{W} \right) \tag{1.10}$$

— MacKenzie (1992)

$$T = a + b \cdot log_2 \left(\frac{D}{W} + 1 \right) \tag{1.11}$$

Version linéaire

— Schmidt *et al.*(1979)

$$T = a\frac{D}{W} \tag{1.12}$$

Version puissance

— Meyer *et al.*(1988)

$$T = a\left((\frac{D}{W})^{\frac{1}{2}}\right) \tag{1.13}$$

Ces différentes versions, qui sont utilisées dans la littérature, sont équivalentes entre elles, par l'utilisation des facteurs a et b dépendants de la physiologie et du type de tâche.

2/3 *Power Law*

Lacquaniti *et al.*, au début des années quatre-vingt ([78]), formule la corrélation entre la cinématique de l'écriture et les mouvements d'une trajectoire dans une formule, connue sous le nom de *2/3 Power Law*. Cette loi lie la courbure C d'une trajectoire à la vélocité angulaire Ω du stylet qui la génère, par une puissance de 2/3.

$$\Omega(t) = k\, C^{2/3}(t) \tag{1.14}$$

avec k une constante appelée *facteur de gain*.

Cette loi a été montrée comme valide dans de nombreux classes de mouvements (dessins simples, trajectoires locomotrices le long de courbes prédéfinies - du pied dans la marche humaine ou de l'oeil dans les poursuites occulaires[102]).

Principe d'isochronie

Freeman, en 1914 ([49]), établit la tendance spontanée à accroître la vitesse du mouvement en fonction de la distance à parcourir (sans contraintes imposées sur la vitesse du geste). Cette observation exprime l'invariance du temps d'exécution du mouvement en relation avec l'amplitude de ce dernier.

Lorsque l'utilisateur n'a pas de contraintes quand à la précision de la tâche à effectuer, la vitesse du geste va augmenter naturellement. Ainsi, il peut conserver un temps d'exécution constant lorsque la distance à la cible augmente.

Minimum jerk law

En dérivant la position, nous pouvons obtenir la vitesse d'un membre effecteur, et en dérivant une seconde fois, son accélération. Flash et Hogan, en 1985, en dérivant l'accélération pour obtenir le *jerk* d'un mouvement, établissent qu'un mouvement biologique est optimal lorsqu'il est le plus régulier possible, c'est à dire lorsqu'il minimise ce *jerk* (i.e. la variation d'accélération - [58]). Parmi l'ensemble des trajectoires possibles pour réaliser un geste, le système nerveux choisit la solution qui minimise la fonction de coût C :

$$C = \frac{1}{2}\int_{t_1}^{t^2}\left[\left(\frac{d^3x}{dt^3}\right)^2 + \left(\frac{d^3y}{dt^3}\right)^2\right]\,\mathrm{d}t \tag{1.15}$$

Steering Law

Faire naviguer un curseur dans un tunnel étroit, comme lors de l'utilisation d'un menu déroulant, n'est pas une tâche de Fitts, car cela demande une contrainte de précision constante. En effet, le curseur ne doit pas sortir du tunnel. Pour un tunnel en ligne droite (1.10) de largeur L et de longueur D, la loi de navigation (*steering law*) prédit que le mouvement prendra un temps qui est fonction linéairement de L et de D :

$$T = a + b\frac{D}{L} \tag{1.16}$$

Cette loi peut également modéliser des trajectoires courbes arbitraires, ainsi que la vélocité instantanée ([1]).

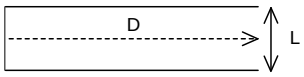

FIGURE 1.10 – Déplacement dans un tunnel droit de longueur A et de largeur L. L'utilisateur doit suivre la ligne pointillée sans sortir des bordures.

1.2.1 Influence de la cible et de la distance sur la trajectoire

Ruiz *et al.* montrent ([95]) que la vitesse instantanée, l'accélération et le *jerk* sont affectés par les contraintes de la cible. Ces contraintes, (la taille et la direction de la cible), sont celles qui induisent le plus les différences entre les gestes de pointage. De plus, les contraintes de cible impactent le geste dans les 70% du début de la trajectoire, et ce dès les 20% de cette trajectoire, juste après le début du geste. Nous voyons dès lors que l'impact de ces contraintes sur l'organisation du geste survient très tôt dans le geste, ce qui peut être consistant avec la théorie de la planification extrinsèque des coordonnées du geste par le système nerveux présenté en 1.1.1.0. En effet, la construction d'un geste dans ces conditions demande très tôt l'utilisation des propriétés de la cible dans l'environnement.

1.2.2 Limites de Fitts - petites cibles

Chapuis et Dragicevic établissent ([31]) que la loi de Fitts n'est plus valable pour les très petites cibles. La petite taille dans l'espace visuel entraîne une faible lisibilité, faisant diminuer les performances. La précision du système moteur est également un facteur limitant, alors que les problèmes de quantification du dispositif de pointage n'interviennent que très faiblement. Ces limites du système physiologique (les problèmes apparaissent à partir de cibles demandant 4 dots de déplacement avec la souris - soit 0,254 mm pour une souris à 400 DPI) concordent avec les propositions de Casiez ([29]). La limite intervient lorsque le plus petit mouvement de déplacement de la souris à effectuer est inférieur au plus petit déplacement discret que les muscles de la main et du poignet peuvent réaliser en partant d'une main à l'arrêt.

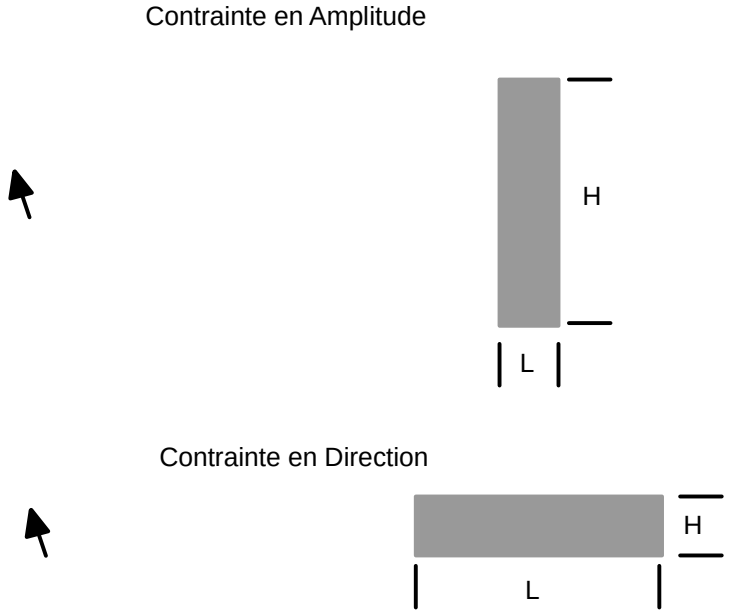

FIGURE 1.11 – Contraintes sur la tâche de pointage. En haut, une contrainte en amplitude, ou contrainte d'arrêt. En bas, une contrainte en direction, ou *steering constraint*.

1.2.3 Extension à deux dimensions

L'expérience de Fitts se déroule dans une seule dimension, alors que la plupart des interactions avec les interfaces graphiques modernes sont bidimensionnelles. Certains chercheurs (Card *et al.*[100], Jagacinski et Monk ([64]) ont appliqué le paradigme de Fitts à un pointage en deux dimensions, mais en ramenant néanmoins ce pointage à un pointage à une seule dimension, c'est à dire dans la direction du geste. Ils considèrent la distance jusqu'au centre de la cible, avec la largeur de la cible définie comme la partie de la cible intersectant cette trajectoire. Cette approche simple ne prend pas en compte deux facteurs : l'angle d'approche, et la largeur de la cible lorsqu'elle est asymétrique. En effet, le corps humain n'a pas les mêmes facilités de déplacement dans toutes les directions. Card *et al.*([100]) ont malgré tout montré que l'angle d'approche avait peu d'influence sur le résultat final (pas de différence significative de cet angle sur le temps total en utilisant une souris, et seulement 3% en utilisant un joystick).

Le problème lié à la forme de la cible apparaît lorsque la cible n'est pas ronde. Que devons nous considérer comme largeur de cible dans le cas d'une cible asymétrique (par exemple pour un mot dans un traitement de texte, qui sera plus large que haut, ou une icône non-rectangulaire) ? Ce problème de détermination de la largeur effective d'une cible est illustré dans la figure 1.12. Lors de l'acquisition d'une cible en deux dimensions, plusieurs termes peuvent potentiellement utiliser la hauteur H, la largeur L, la diagonale L'

MacKenzie ([82]) évalue différentes heuristiques pour apprécier la largeur effective d'une cible :

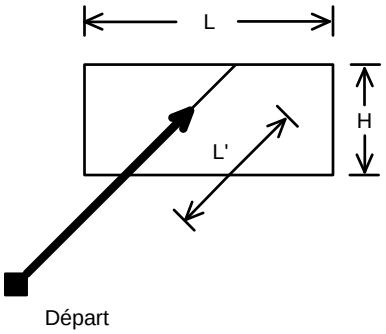

FIGURE 1.12 – Détermination de la largeur effective d'une cible.

— L : largeur de la cible
— L + H : largeur plus hauteur de la cible
— L × H : largeur × hauteur de la cible
— MDLH : Minimum De la Largeur et de la Hauteur
— L' : largeur de la cible intersectée par le vecteur d'approche

En conclusion de son expérience, L + H et L × H ne sont pas meilleurs que l'actuel L. MDLH et L' sont meilleurs que L, mais non significativement différents l'un de l'autre. Dès lors, MDLH est le modèle préféré, car moins coûteux en calculs que L'. Le modèle L' reste supérieur dans le cas de cibles non-rectangulaires, et conserve l'unidimensionnalité inhérente au modèle de Fitts.

Les lois physiologiques étudiées, et leur fonctionnement en deux dimensions validées, nous pouvons maintenant regarder le corpus de techniques d'interactions disponibles. En effet, les lois précédemment listées servent à construire de meilleures techniques d'interaction. La suite de notre chapitre va dès lors détailler les techniques d'interactions existantes.

1.3 Techniques d'interaction

Avant de présenter les techniques d'interactions en elles-mêmes, il nous a semblé important de préciser le vocabulaire qui pourra être utilisé par la suite, relatif au curseur de la souris, ainsi qu'à son gain.

1.3.1 *Control-Display Gain*

La méthode la plus directe pour interagir avec un système est de toucher directement la cible, ce qui est par exemple possible sur les écrans tactiles (disponibles sur les systèmes informatiques depuis 1974 avec l'écran d'*Elographics* disponible à cette époque). L'utilisation de systèmes plus contraints avec un périphérique comme une souris impose une mise en correspondance entre le déplacement du périphérique et sa représentation à l'écran. Le gain, ou *Control-Display Gain* - abrégé en *CD-gain* - dans la littérature, exprime la relation entre le degré de contrôle demandé par l'utilisateur et la réponse du système en

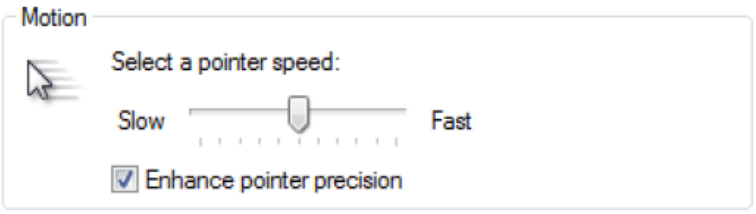

FIGURE 1.13 – Exemple de l'ajustement de la *vitesse* de la souris sous Windows XP (en ajustant cette vitesse, l'utilisateur change en pratique le gain).

conséquence du mouvement.

Nous parlons d'*espace physique réel* pour décrire le support sur lequel se déplace la souris (généralement un plan de travail), et d'*espace virtuel* pour l'écran sur lequel le curseur s'affiche. Le gain se définit alors comme le rapport entre la vitesse du pointeur ($V_{pointeur}$) dans l'espace virtuel et celle de la main (V_{main}) dans l'espace réel, exprimées dans les mêmes unités [29] (équation 1.17).

$$gain = \frac{V_{pointeur}}{V_{main}} \tag{1.17}$$

Ce gain peut être constant, avec le curseur qui se déplace de la même distance dans l'espace virtuel que la distance parcourue par la souris dans l'espace réel, multipliée par le gain. Il peut aussi varier en fonction de la vitesse de la main. Il y a dès lors une *accélération* du pointeur, avec la fonction donnant la valeur de gain en fonction de la vitesse de la main. Cette fonction est appelée *fonction de gain*. Lorsque le gain reste constant, quelle que soit la vitesse de la main, nous parlons de *fonction de gain constante* ; lorsqu'il varie, nous parlons de *fonction de gain variable* ou *dynamique*.

Cette fonction porte également le nom dans la littérature de *fonction de transfert*.

Fonction de transfert

Une courbe de transfert est une transformation mathématique qui s'applique aux données venant d'un périphérique d'entrée. Cette fonction doit améliorer le contrôle du périphérique en le rendant plus stable, mais une mauvaise fonction de transfert peut limiter les performances. Pour les périphériques mesurant une force, la fonction de transfert transforme la force en vitesse (par exemple le joystick isométrique Trackpoint sur les ordinateurs portables IBM contrôlant la vitesse à laquelle le curseur se déplace). D'autres transformations sont possibles, comme position-vers-position ou vitesse-vers-vitesse (comme pour les tablettes ou les souris).

En 1949, Jenkins et Conner ([68]) publient une expérience qui demande aux sujets de réaliser une trajectoire constituée de deux phases, avec une première partie "grossière", et une deuxième phase utilisant un contrôle plus fin. Les performances sur ces deux phases sont optimales lorsque le gain est proche de 1 (gain de 1 :18). Ils concluent que ce ratio combine les avantages de la vitesse pour le déplacement avec la finesse de l'ajustement final. La figure 1.14 (adaptée de [68]) présente le ratio de *Control-Display gain* associé

avec les temps de mouvements minimums pour la phase ballistique entraînent les temps les plus longs en phase corrective, et vice versa. Le ratio CD optimal pour cette tâche est à l'intersection des deux.

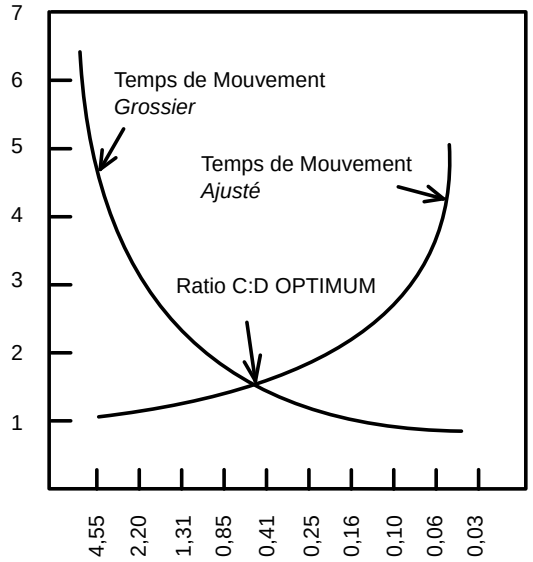

FIGURE 1.14 – ratio CD optimal pour les phases ballistiques et correctives

D'autres chercheurs ont remarqué cette courbe en forme de U avec des performances optimales lorsque le gain est modéré [50, 109]. Certains ont conclu que les fonctions de gain dynamique n'ont que peu d'effet [67], ou qu'elles diminuent les performances [52, 101]. Ces études ont porté sur des fonctions de gains dynamiques qui étaient non continues, ou avec un gain maximum peu élevé, alors que les courbes effectivement utilisées par les systèmes d'exploitation modernes sont au contraire continues, avec des valeurs de gains importantes pour des vitesses élevées. Casiez *et al.* ont étudié l'influence des fonctions de gains constantes et dynamiques dans les tâches de pointage [29]. Pour les fonctions de gains constantes, ils ont utilisé des valeurs de gain égales à 1, 2, 4, 6, 8 et 12. Pour les fonctions de gain dynamiques, ils ont utilisé les fonctions de gains dynamiques du système d'exploitation Microsoft Windows XP, en extrayant les valeurs de la base de registre. Les résultats montrent que les fonctions de gains dynamiques testées améliorent significativement, de 3,3% en moyenne, le temps de pointage par rapport aux fonctions de gains constantes, avec un avantage marqué pour les petites cibles (5,6%). Il a également été montré que pour les fonctions de gains constantes ainsi que pour les différentes fonctions de gains dynamiques proposées par Windows, il existe un ensemble de fonctions pour lesquelles les performances sont optimales. En effet, les courbes Windows correspondant à des facteurs d'échelle de 0,1 ; 0,25 et 1,25 ont montré une diminution des performances.

Le vocabulaire relatif au curseur de la souris et à son gain étant défini, nous pouvons passer aux techniques d'interactions proprement dites.

1.3.2 Les différentes techniques d'interaction

Afin de faciliter le pointage dans les interfaces graphiques, quatre approches sont possibles :

1. Rapprocher la cible (réduire D)
2. Agrandir la cible (augmenter W)
3. Moduler le gain de la souris
4. Sauter les espaces vides

Dans le cadre d'une interaction indirecte avec interface graphique et une souris, deux approches sont possibles. Le pointage dans le monde physique est gouverné par la loi de Fitts (voir 1.2.0.0), sans modification possible des contraintes physiques du pointage (que ce soit la distance à la cible ou sa taille, par exemple), alors que le pointage dans un monde virtuel peut modifier ces contraintes lorsqu'elles relèvent de l'implémentation logicielle. Pour cela, afin d'être "meilleur" que cette loi, nous pouvons influencer artificiellement les deux facteurs que sont la distance à la cible (c'est à dire la réduire), ou augmenter la taille de cette dernière, ce qui permet dans les deux cas de réduire l'Indice de Difficulté de la tâche.

Le système d'exploitation, pour faciliter la tâche de pointage, peut agir sur le fonctionnement du pointeur de souris lui-même (sa taille ou sa vitesse, par exemple), ou manipuler le fonctionnement de la cible (en diminuant sa distance au pointeur, ou en changeant sa taille de manière interactive). Ces deux approches donnent un résultat final identique (faciliter l'acquisition d'une cible), mais leur implémentation est différente.

Certaines caractéristiques physique de l'écran nous donnent déjà des zones facilement atteignables vis-à-vis de la loi de Fitts : les 4 coins de l'écran, et le pixel *sous* le curseur. En effet, en "lançant" la souris dans la direction d'un des 4 coins de l'écran, l'utilisateur est certain que le curseur va s'arrêter, bloqué par les bords de l'écran. Ainsi, les cibles placées à cet endroit-là auront alors une largeur "infinie". Appert *et al.* quantifient en 2008 ([4]) cet effet, dont l'intuition était présente dans la communauté depuis longtemps : pointer une cible présente sur le bord de l'écran peut être jusqu'à 44% plus rapide que pour la même cible non-adjacente au bord. Potentiellement, l'utilisateur peut rester pour toute la trajectoire en phase balistique, le curseur arrivera quand même sur la cible. Comme illustré dans la figure 1.15, la présence d'une cible contre un bord permet de profiter d'une largeur de cible "infinie", et donc de réaliser l'acquisition de cette cible plus rapidement, car sans geste de correction de la trajectoire.

Ils montrent également que la direction du pointage a une influence sur les performances. De même, Phillips et Triggs ([89]) montrent que se déplacer vers la droite est plus rapide que de se déplacer dans les autres directions, pour un droitier (le pointage *distal* - qui s'éloigne du corps avec le membre effecteur, est plus facile physiologiquement à réaliser que le pointage *proximal*, qui se rapproche du corps avec le membre effecteur). Dès lors, la performance du pointage est affectée par la direction du déplacement.

De même, toute icône ou menu placés sous le curseur de la souris sera atteint immédiatement par l'utilisateur, sans devoir bouger la souris. C'est pourquoi les menus contextuels (généralement accessibles à l'aide d'un autre bouton sur la souris que celui utilisé pour réaliser le clic principal lors de la tâche de pointage) apparaissent à cet endroit.

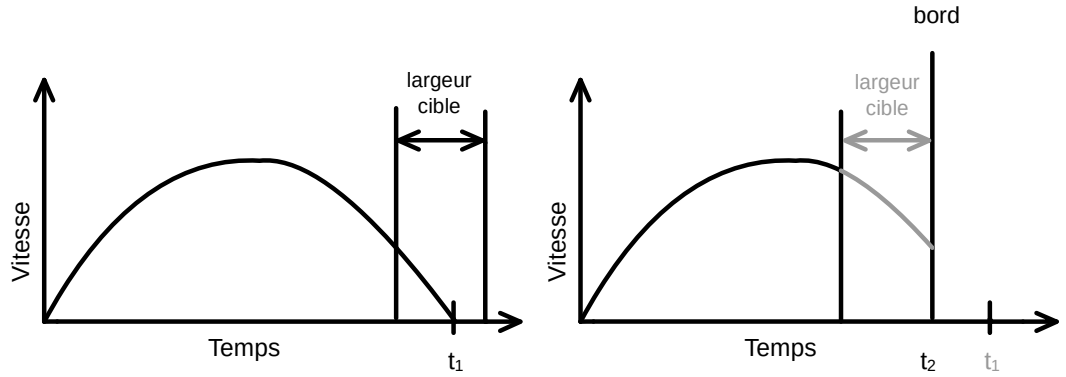

FIGURE 1.15 – La présence d'une cible contre un bord permet de profiter d'une largeur de cible "infinie".

Ces informations devront rester à l'esprit lors de la conception d'expérience de pointage, afin de ne pas, par inadvertance, modifier le geste de pointage ou le faciliter en proposant à l'utilisateur de s'appuyer dessus.

Différentes catégories d'interactions sont possibles avec un pointeur de souris, notamment le pointage et l'acquisition de cible, ainsi que le *glisser-déposer*. Dans la littérature, l'accent a été mis sur la première catégorie d'interactions.

Manipulation de la cible

1. *Drag'n'Pop*

   Baudisch *et al.* ont développé la technique du *drag'n'pop* (par opposition au drag'n'drop) pour rapprocher les icônes distantes du pointeur [10]. Avec cette technique, le système réagit aux mouvements du curseur initiés dans une direction, en approchant du curseur une ou plusieurs icônes virtuelles, copies des icônes potentiellement visées par le geste d'origine. L'interface ne change pas, mais l'utilisateur peut interagir avec ces icônes plus faciles à atteindre, pour un geste de *drag'n'drop*, par exemple. Baudisch *et al.* ont mesuré des temps 3,7 fois plus rapides pour de grandes distances, mais avec un taux d'erreur un peu plus élevé. Malgré ces améliorations des performances, la technique pose problème lors de la détermination des cibles. Il peut être compliqué de déterminer si l'utilisateur va atteindre des cibles proches ou des cibles lointaines, ou plusieurs cibles dans le même axe. De plus l'éventuelle activation de ces cibles virtuelles, dans le cas d'un "faux positif", peut interférer avec le geste de l'utilisateur, diminuent les performances et gênent l'utilisateur. Cette solution est plus opérante lorsque l'écran de l'utilisateur est peu encombré de cibles.

   La figure 1.16 illustre le principe de la technique. Une cible virtuelle à partir des icônes sur la gauche est fournie par le système à proximité du curseur de souris. Ces cibles virtuelles ne sont présentes que pour la durée du déplacement du curseur et n'affectent donc pas l'ensemble de l'interface.

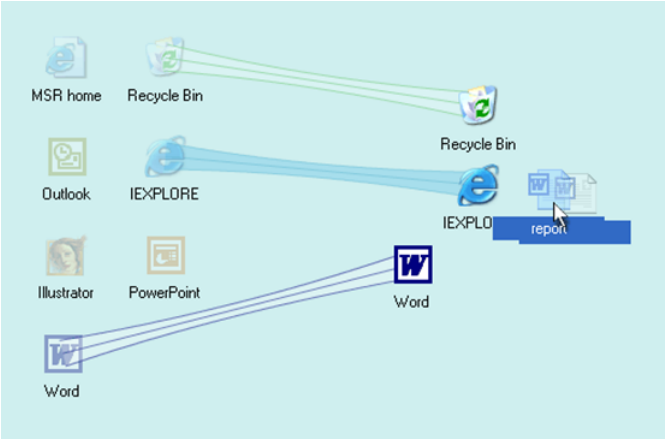

FIGURE 1.16 – Drag'and'Pop - Baudisch *et al.*. (*Image d'après [10]*)

Une des solutions pour exploiter cette technique malgré tout (alors que plusieurs cibles potientielles sont au voisinage de la cible) est proposée par Balakrishnan ([8]) : utiliser un mode pour activer la technique (on perd alors la fluidité de la technique).

Un mode est une interaction qui se comporte différemment suivant le contexte dans lequel elle se déroule. Par exemple, la touche majuscule d'un clavier est un mode : suivant son état, une même touche envoie une lettre majuscule ou minuscule.

2. *Area Cursors - Curseurs de Zone*

   Kabbash et Buxton proposent d'augmenter artificiellement la largeur de la cible dans l'espace moteur, en utilisant une *zone* plutôt qu'un curseur pour cliquer sur ladite cible [70]. Ils montrent ainsi que les cibles ayant une taille plus petite que la zone à sélectionner acquièrent finalement une taille égale à cette zone. Même si cette technique permet l'acquisition de petites cibles, et facilite l'action à plus grande distance, en réduisant leur Indice de Difficulté, la tâche se trouve compliquée lorsque plusieurs cibles entrent simultanément dans la zone de sélection.

   Afin de remédier à ce problème, Worden *et al.* ([107]) proposent d'améliorer les *areas cursors* en les séparant en deux zones : la zone autour du curseur, plus large, et un point, discret, au centre du curseur lui-même. Lorsque les cibles sont éloignées, le curseur se comporte comme dans l'expérience de Kabbash et Buxton. Lorsque plusieurs cibles sont à l'intérieur de la zone d'activation du curseur, le point distinct au centre du curseur sert à faire la discrimination entre les cibles. Leur expérience montre que leur curseur se comporte dans le pire des cas avec les mêmes performances que celui de Kabbash et Buxton, mais réalise de meilleures performances quand les cibles sont proches. Le curseur de zone est donc plus rapide que le curseur classique, surtout quand les cibles sont petites, mais la taille de la zone d'activation reste fixée à l'avance, réduisant son efficacité.

3. *Bubble Cursor*

   Grossman et Balakrishnan proposent, avec le *Bubble Cursor*[53], un curseur dont la zone d'activation circulaire grandit jusqu'à inclure la cible la plus proche de ce

curseur. L'utilisateur ne doit alors plus déplacer la souris pour amener le curseur plus proche de cette cible. La zone d'activation ne contient alors qu'un seul objet à la fois. L'efficacité de la technique diminue lorsque plusieurs cibles potentielles sont accessibles de manière équidistante.

Grossman et Balakrishnan notent que la technique correspond de manière formelle à un agrandissement de chacun des objets jusqu'à ce que leur taille soit équivalente à une région correspondante dans un diagramme de Voronoï. Dans ce type de diagramme, chaque cible est au centre d'une cellule qui maximise sa surface comparé aux autres cellules.

De manière graphique, le curseur bulle est dessiné comme un cercle transparent, dont la zone d'activation intersecte en surimpression la cible la plus proche. Afin de ne pas intersecter une seconde cible, la zone d'activation peut ne pas entourer complètement l'objet le plus proche. Dans ce cas, une deuxième bulle est accolée à la première, qui s'étend à partir de l'intersection et englobe l'objet le plus proche.

La figure 1.17 présente le raisonnement permettant de construire un Bubble Cursor à partir d'un *area cursor*, en 4 étapes. a) Un curseur de zone facilite l'acquisition d'une cible en utilisant une surface plus grande qu'un curseur discret. b) Il est difficile d'isoler la cible désirée lorsque le curseur de zone couvre plusieurs cibles potentielles. c) Le *Bubble Cursor* corrige le problème en b) en utilisant une zone active dynamique de manière à ce que seule la cible la plus proche soit sélectionnée. d) Le *Bubble Cursor* se déforme pour inclure une cible lorsque la forme circulaire du curseur ne permet pas de le faire sans sélectionner une autre cible voisine.

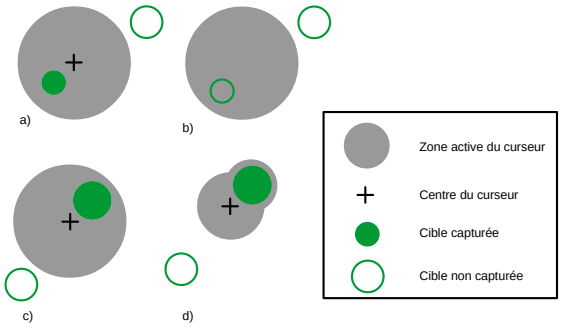

FIGURE 1.17 – *Bubble Cursor*

L'expérience de Grossman et Balakrishnan montre que le curseur bulle est plus rapide, que ce soit pour le curseur classique ou pour l'*Object Pointing*, et qu'il a un taux d'erreur moindre, même dans des conditions incluant jusqu'à 200 cibles (donc 200 "distracteurs potentiels").

4. *Expanding Targets*

   Pour améliorer le pointage, nous pouvons utiliser différentes techniques pour augmenter la taille de la cible. La première, présentée par McGuffin et Balakrishnan ([84]), est d'augmenter la taille des cibles proches du curseur. Ils montrent que les utilisateurs bénéficient de la technique, même lorsqu'elle n'apparaît alors que le

curseur a déjà parcouru 90% de la distance à la cible. De plus, la loi de Fitts reste correcte sur ce type de cible, lorsque le geste est évalué sur une dimension. De même, l'expérience confirme la théorie des sous-mouvements correctifs. En effet, les utilisateurs, même tard dans le geste, s'aperçoivent du changement de taille de la cible dans une des phases correctives, et peuvent atteindre la cible à l'intérieur de ses nouvelles limites, plutôt que les limites correspondantes à la taille initiale. Zhai, Conversy, Baudoin-Lafont et Guiard ([112]) étendent cette découverte aux situations dans lesquelles l'utilisateur ne sait pas à l'avance que les cibles vont s'agrandir. Cockburn et Firth ([35]) montrent que les cibles qui s'agrandissent sont sélectionnées plus rapidement que les cibles classiques, mais que le *goal crossing* ([2]) est plus rapide encore. Ce type de *widgets* qui ajustent leur taille de manière dynamique se retrouve par exemple avec le *dock* de MacOSX. Les icônes y sont affichées à une taille réduite, et augmentent de taille dynamiquement lorsque le curseur se rapproche d'elles. Dans le cas du dock, si l'utilisateur n'arrive pas sur la cible perpendiculairement à celle-ci, les icônes vont se déplacer dans l'espace visuel, impliquant une discontinuité dans l'espace moteur également. Le problème disparaît si les icônes se superposent les unes aux autres ([84]).

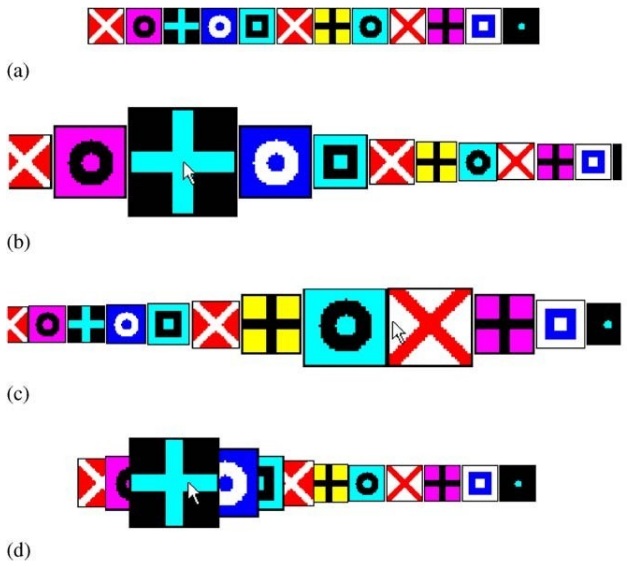

FIGURE 1.18 – *Expanding Targets*

Différents conceptions sont possibles pour les *Expanding Targets* et sont présentées sur la figure 1.18. a) Les cibles sont normales lorsque le curseur est éloigné. b) Conception qui fait grossir complètement la cible sous le curseur, et partiellement les cibles à proximité en repoussant les cibles adjacentes. c) Un utilisateur se trouvant dans la configuration *b)* et désirant atteindre la cible adjacente verra les cibles se décaler, empêchant la concordance entre la cible se trouvant à l'emplacement cible à l'initiation du geste et la cible se trouvant sur cet emplacement cible à la fin du geste. d) Une conception permettant aux cibles de se superposer conserve une concordance dans l'espace moteur, même si la concordance dans l'espace visuel

n'est pas conservée.

5. *Goal Crossing* ([35])

   La technique du *Goal Crossing* permet à l'utilisateur de sélectionner une cible en passant à travers la zone correspondante, tout en gardant une touche enfoncée, pour activer la sélection. Cela permet de ne pas décélérer la souris lorsque le pointeur passe au dessus de la cible [1, 2]. Accot et Zhai ont travaillé avec un stylet plutôt qu'une souris, mais montrent que la tâche de passage de cible suit la loi de Fitts. Dans la configuration présentée lors de l'expérience, la largeur de la cible est infinie dans l'espace moteur. En effet, la tâche étant validée lorsque l'utilisateur traverse la cible, et non plus lorsqu'il s'arrête dedans, il n'y a aucune contrainte en largeur sur la cible. Un des problèmes de la technique apparaît lorsque plusieurs cibles potentielles sont alignées sur le même axe. L'utilisateur doit alors changer son axe d'approche, afin d'éviter de sélectionner une cible non désirée.

   MacKenzie propose en 1992 la technique du *Stroke-Through*([83]). Similaire dans la principe au *goal crossing*, elle demande à l'utilisateur d'enfoncer un bouton de la souris avant de passer sur l'icône cible. Une fois la cible dépassée, l'utilisateur relâche le bouton de la souris. Toutefois, le geste de pointage est différent lorsqu'un bouton est enfoncé de lorsqu'il est effectué librement. MacKenzie ([82]) montre que les temps de pointage sont plus rapides que les temps de *glisser-lacher*. Cette différence s'explique par les interférences (cognitives et physiologiques) causées par la nécessité de garder un bouton enfoncé. Malgré cela, l'expérience démontre une amélioration de 40% en temps sur la sélection classique.

6. *Starbust*

   L'expansion des cibles dans l'espace qui les environne amène à la segmentation de l'espace sous la forme d'un diagramme de Voronoï. Pour les zones dans lesquelles ces cibles sont denses, l'expansion est minimale. La technique *Starbust* proposée par Baudisch *et al.*([11]) connecte chaque cible à un morceau d'espace périphérique, hors de toutes les autres cibles, afin de créer des zones de tailles importantes.

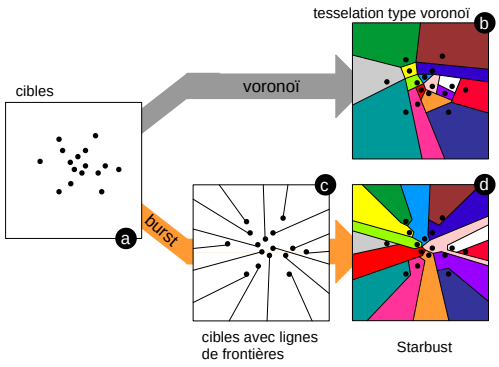

FIGURE 1.19 – *Starbust*

Sur la figure 1.19, nous pouvons voir une illustration de la différence de partition d'un plan couvert de cibles soit par la technique d'un diagramme de Voronoï clas-

sique, soit par la technique Starbust. (a→b) Les techniques traditionnelles d'expansion des cibles le font dans *l'espace immédiatement adjacent*. Pour les cibles présentes au sein d'un groupe cette expansion est minimale. (a→b→d) L'algorithme Starbust connecte les cibles avec l'espace périphérique pour produire des zones de taille raisonnable pour toutes les cibles (y compris celles présentes au sein d'un groupe).

Un des défauts de l'algorithme utilisé est la création de zones longues et peu larges, qui sont des cibles plus difficiles à acquérir que des cibles rondes, plus étalées.

7. *DynaSpot*

   Afin de ne pas subir les problèmes liés à la présence de cibles éventuelles sur la trajectoire du curseur, Chapuis *et al.*([32]) proposent le *DynaSpot*, qui fait dépendre la zone d'activation d'un curseur de sa vitesse. Le DynaSpot se comporte comme un pointeur discret à faible vitesse ou à l'arrêt, et grandit juqu'à une taille maximale à grande vitesse, taille prédéfinie pour ne pas surcharger le canal visuel.

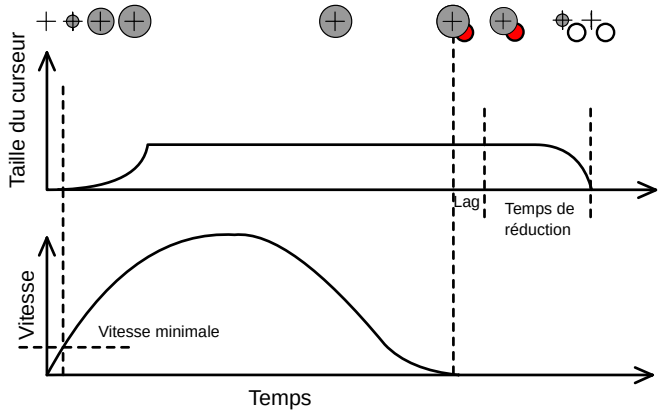

FIGURE 1.20 – *DynaSpot*

Lorsque plusieurs cibles peuvent être potentiellement sélectionnées par le curseur, la proximité du centre du curseur détermine la cible acquise. La technique est 18% plus rapide qu'une technique de pointage classique, et ses performances sont équivalentes au *Bubble Cursor*. La technique est légèrement plus efficace pour des densités faibles de cibles et légèrement moins efficaces pour des densités importantes.

Après avoir manipulé le curseur, d'autres techniques sont possibles, notamment manipuler la surface sur laquelle se déplace ce curseur. Nous présentons ici quelques techniques de marquage de cette surface (ou marquage de l'écran).

1. Poussière magnétique

   Hurst *et al.* ([61]) proposent une technique *pseudo-haptique*, indépendante du système d'exploitation, de l'application (et donc de la sémantique des cibles) et de l'action (simple clic ou *glisser-lâcher*), la *poussière magnétique*. Le système agrège, application par application, l'ensemble des clics et *glisser-lâcher* de la souris, de manière similaire à une patine qui reflète l'endroit où les utilisateurs cliquent le plus

souvent. En associant une petite force attractive avec chaque évènement du pointeur, et en les additionnant, le curseur sera naturellement attiré vers les endroits qui ont été utilisés le plus souvent (de manière similaire à une carte de chaleur). L'utilisation de force discrètes permet d'éviter qu'un clic à un endroit aléatoire ne soit problématique (en effet, un clic placé à un endroit qui n'est pas utilisé d'habitude pourrait ajouter un gradient de forces de déplacement, alors que la zone n'est jamais utilisée). De plus, les patines sont différentes pour chaque application, et peuvent être partagées entre différents utilisateurs, afin d'accélérer la construction de ces cartes. L'avantage de la technique est d'assister la tâche de pointage, mais également le parcours de trajectoires, et de supporter dans le temps l'apparition soudaine de nouvelles cibles ou leur disparition.

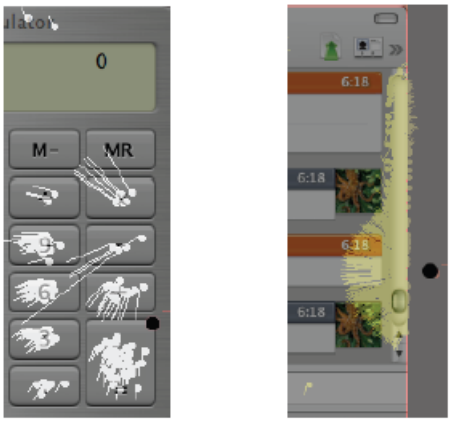

FIGURE 1.21 – Poussière magnétique

La figure 1.21 nous illustre la technique de la poussière magnétique. (**A gauche**) Une illustration de l'accumulation des clics (points blancs) sur les boutons les plus utilisés d'une calculatrice, qui créent une force cumulée (lignes blanches) sur le pointeur (point noir). (**A droite**) Les forces accumulées (lignes jaunes) lors d'un glissé du pointeur lorsqu'il quitte l'ascenseur vertical, sur la fenêtre d'un client de messagerie instantanée.

2. *Kinematic templates*

   Fung *et al.* définissent les *Kinematic Templates* comme un outil explicitant des manipulations dans un espace moteur spécifiques à un contenu donné, pour l'édition visuelle de contenu. Ainsi, l'utilisateur final peut définir, dans un outil de CAO, des zones dans lesquelles le comportement du curseur est modifié, afin de faciliter certains tracés.

   Ces *templates* peuvent être *actifs*, ils altèrent alors le mouvement indépendamment des changements sur les coordonnées x et y, ou *passifs*, ils altèrent alors le gain. (Un exemple de *template* actif serait magnétique : ce *template* attire de manière active le curseur vers un point prédéfini. Un exemple de *template* passif serait un template permettant de dessiner un cercle ou une forme géométrique, quel que soit le point de départ de cette forme à l'écran). Ces *Kinematics templates* ne fonctionnent, et

n'ont été étudiés que lorsque le bouton de la souris est *enfoncé*, situation dès lors similaire à un *glisser-lâcher*, et non à une tâche de pointage.

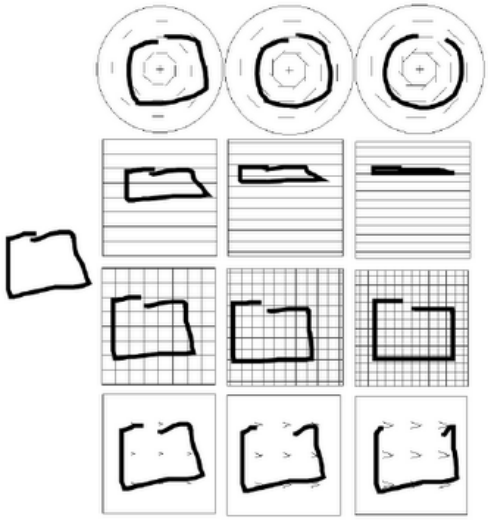

FIGURE 1.22 – Exemple de *Kinematic templates*

Nous pouvons voir un exemple avec la figure (1.22) : à gauche le tracé réalisé dans l'espace moteur, à droite le rendu à l'écran de ce tracé, en utilisant différents *templates* (rond, grille plus ou moins serrée, déplacement suivant une dimension...).

3. UIMarks

   Roussel et Chapuis ([33]) proposent la création des *UIMarks*, afin de sélectionner certaines zones de l'écran, et de leur appliquer une sémantique propre à l'aide d'actions possibles sur ces zones. Ces marques peuvent être créées par l'utilisateur, en anticipation de leur utilisation, ou par le logiciel, en réponse à des actions de l'utilisateur ou en réaction à l'activation d'une autre marque. Par exemple, l'utilisateur peut déterminer une zone qui lancera l'action, à chaque fois qu'il la traverse, de sauvegarder le document sur lequel il travaille.

   L'utilisation d'une variante du curseur à bulle permet une sélection rapide d'une des marques, et accélère donc de manière significative l'acquisition de la cible sous-jacente.

Manipulation du curseur

1. *Object Pointing*

   L'interface graphique présentée à l'écran est constituée, comme toute image numérique, d'une matrice de pixels contigus. Bien que contenant les éléments d'interfaces nécessaire à sa manipulation (curseur, icônes, menus...), la majorité des pixels servent à illustrer cette interface ([56]). Par exemple, 50 éléments sélectionnables à l'écran, mesurant en moyenne $20 \times 20 = 400$ pixels sur un écran de résolution 1600×1200 pixels, seul $50 \times 400 = 20000$ pixels seront sélectionnables parmi

1600×1200=1920000 pixels. Dès lors, 98,9% des pixels disponibles ne fournissent aucune information utile pour la tâche de pointage (d'après [8]). La proposition radicale de Guiard *et al.* ([56]) est de supprimer les pixels inutiles de l'interface, et de placer les éléments cibles de l'interface de manière contiguë. Dès lors, la distance D entre les cibles est réduite de manière significative. Afin de ne pas modifier l'interface dans l'espace visuel, Guiard *et al.* proposent que la technique de l'*Object Pointing* s'effectue dans l'espace moteur. En utilisant cette technique, le curseur "saute" de cible en cible, sans parcourir l'intégralité de l'espace vide qui les sépare. Lorsque le curseur bouge, et que sa vitesse dépasse un certain seuil, le curseur saute sur la cible suivante, dans l'axe du mouvement initié. Leur expérience montre que l'*Object Pointing* est 74% plus rapide que le pointage classique dans le cadre d'un pointage réciproque. En situation réelle, l'efficacité de la technique dépend de la densité des cibles à l'écran. Lorsque celles-ci sont peut nombreuses et éloignées, la technique donne de meilleurs résultats que lorsqu'elles sont plus denses.

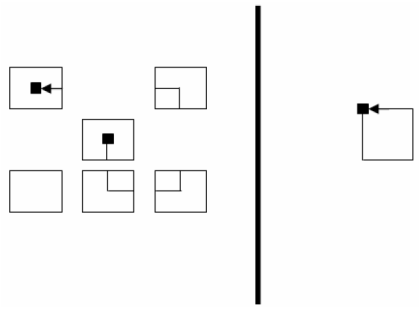

FIGURE 1.23 – *Object Pointing*

Sur la figure 1.23, nous pouvons voir que l'*Object Pointing* diminue l'amplitude du pointage au "sautant" au dessus des espaces vides. A gauche du diagramme est présenté le mouvement dans l'espace moteur, et à droite le mouvement du curseur équivalent, dans l'espace visuel.

Bien qu'étant la technique donnant le plus de résultats significatifs parmi les techniques de marquage de la surface sur laquelle se déplace le curseur, l'implémentation de l'*Object Pointing* est gênée par des problèmes pratiques. L'utilisateur peut vouloir sélectionner des pixels individuels entre les cibles (par exemple entre les lettres dans un traitement de texte, ou les pixels d'une image). L'espace "vide" peut aussi être porteur d'une interaction potentiellement, comme lorsque l'utilisateur doit sélectionner plusieurs icônes. Dans ce dernier cas, il doit cliquer en dehors du groupe d'icônes, et tracer une boîte englobante qui contient l'ensemble des cibles désirées. Fournir un mode pour passer du pointage classique à l'*Object Pointing* diminue la fluidité de l'interaction. De plus, comme constaté par Guiard *et al.* eux-mêmes, une interface classique présente déjà les éléments de manière contiguë (barres de menus et d'icônes - sans espaces entre eux), ramenant la technique à une tache de pointage classique dans ce cas.

2. *PointAssist*

Les jeunes enfants (4 ans) montrent des difficultés à utiliser la souris pour cliquer sur des cibles à l'écran, conséquence de leur développement psychomoteur inachevé. Hourcade *et al.*([60]) amènent les performances de ces enfants à celles de jeunes de 18 à 20 ans en implémentant la technique *PointAssist* d'assistance au pointage, qui s'applique lorsqu'elle détecte des difficultés de pointage. Lorsque le système détecte un sous-mouvement (voir 1.1.1) dont la direction change deux fois successivement, la technique diminue le gain du curseur afin de le ralentir, et passe ainsi en *mode de précision*. Si le sous-mouvements dépasse un seuil préétabli, le système quitte ce mode de précision. Avec cette technique, la précision des enfants passe de 79% à 92%. La technique fonctionne également sur les personnes âgées, bien que l'étude de Hourcade de 2010 ([59]) conserve le modèle des sous-mouvements développé pour les enfants, qui n'a pas été modélisé de nouveau pour ces personnes âgées.

3. MAGIC - Gaze pointing

   Les dispositifs de suivi du regard sont souvent utilisés pour permettre aux personnes handicapées de manipuler les interfaces normalement utilisées par les personnes valides. Afin de ne pas réécrire spécifiquement les interfaces, le déplacement du pointeur est affecté au suivi du point de regard du sujet. Ce dispositif est limité par la nature physiologique du regard humain, constamment en mouvement. Ainsi toute zone de l'écran devient potentiellement une cible. De plus, il n'est pas possible d'utiliser les clins d'oeil pour cliquer sans disconfort à long terme. Zhai *et al.*([110]) proposent la technique *MAGIC*, en repositionnant grossièrement le curseur de souris là où l'utilisateur regarde, puis de le laisser achever le mouvement à l'aide de la souris. Le curseur saute ainsi de zone cible en zone cible lorsque le nouveau point de regard est à au moins 200 pixels de distance de l'ancien.

   La technique rencontre cependant un problème lié à l'utilisation séquentielle de la souris et de la vue, les deux pouvant être en mouvement lors de l'initiation d'un nouveau geste. Le curseur saute alors au delà de la cible, vers une position difficile à anticiper par l'utilisateur.

4. Ninja Cursor

   Kobayashi et Igarashi ([75]) utilisent les *Ninja Cursors* pour tenter d'accélérer l'acquisition de cible sur des distances importantes. Ces curseurs sont présentés en grille à l'écran, avec un intervalle constant entre chacun d'eux, et l'ensemble de cette grille bouge de concert lorsque l'utilisateur déplace la souris. Cette technique accélère la tâche de pointage, car la cible à atteindre est, dans le pire des cas, à proximité d'un curseur dont la distance correspond à l'intervalle standard sur la grille de curseur. Dans la majorité des cas, ce curseur sera plus proche de la cible que le dernier curseur utilisé par le sujet. Un des problèmes de la technique provient de l'ambiguïté du choix du curseur quand certains d'entre eux survolent différentes cibles potentielles en même temps.

   La technique des Ninja Cursor, visuellement originale, est présentée sur la figure 1.24. **(En haut) :** Tous les curseurs suivent le déplacement physique de la souris. **(En bas) :** Les flèches indiquent la distance entre chaque cible rectangulaire et le curseur le plus proche. Avec un curseur unique, les cibles sont éloignées du curseur (*à gauche*). En utilisant plus de curseurs, en moyenne, ces cibles sont plus d'au

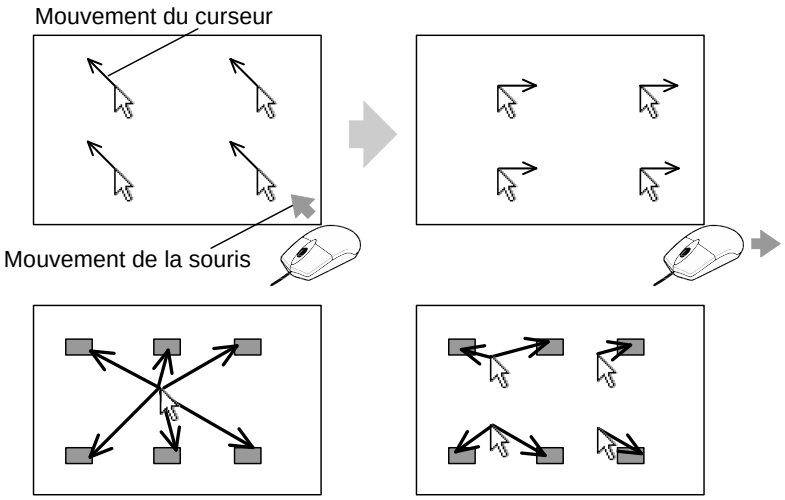

FIGURE 1.24 – *Ninja cursors*

moins un des curseurs (*à droite*).

5. Rake Cursor
   Afin de remédier à l'inconvénient des *Ninja Cursors*, Blanch et Ortega ([14]) utilisent simultanément un dispositif de suivi du regard. En fournissant ainsi un canal d'entrée supplémentaire, ils évitent l'utilisation d'une interaction supplémentaire pour déterminer le curseur choisi par l'utilisateur. Une zone en surbrillance à l'écran indique à l'utilisateur où se porte son regard, et active le curseur le plus proche de cette zone.

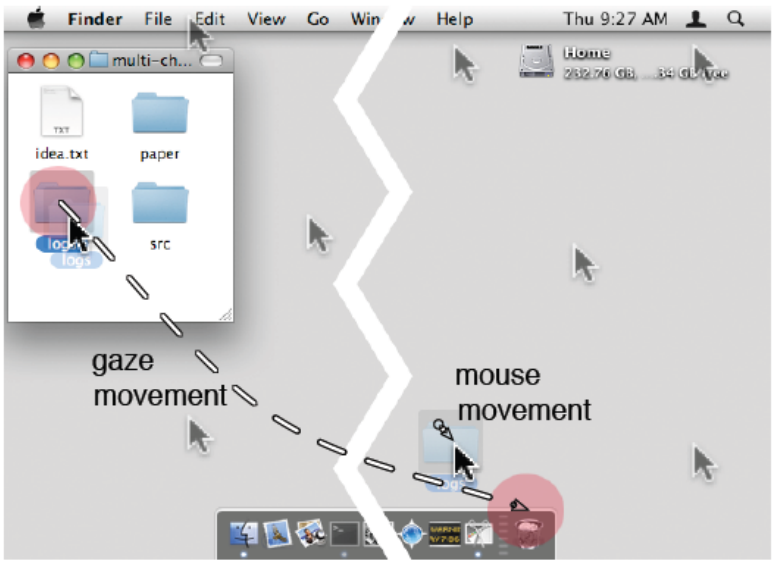

FIGURE 1.25 – Rake Cursor

Les multiples mouvement de curseur pour des *tirer*-(à gauche) -*lâcher*(à droite) sont

visibles sur la figure 1.25. Le regard sélectionne le curseur actif (le disque rouge indique la position du regard, ajoutée à l'image, les curseurs ont été agrandis).
La technique est plus performante que le *MAGIC pointing*, tout en ne surchargeant pas le canal visuel avec des informations de contrôle moteur.

6. Cibles Collantes
   Worden *et al.* proposent l'utilisation de *"cibles collantes"*([107]) afin d'aider les personnes plus âgées à acquérir les cibles de manière plus aisée. Lorsque le pointeur se déplace à proximité de la cible, le gain du pointeur est manipulé afin de rendre la cible "collante". De petits mouvements ne feront pas sortir le curseur de la cible, mais de plus grands gestes décolleront le curseur de la cible.
   Cockburn [36], afin d'attirer et de retenir le curseur plus longtemps sur la cible, manipule le gain du curseur à faible vitesse (condition généralement atteinte lorsque l'utilisateur est sur la cible).
7. Pointage Sémantique
   Blanch, Guiard, et Beaudoin-Lafon ([13]) appelent *Pointage Sémantique* la technique consistant à manipuler le gain du pointeur pour le faire "coller" aux objets. Ce Pointage Sémantique augmente la taille des cibles dans l'espace moteur, sans changer leur aspect dans l'espace visuel. Le gain est augmenté lorsque l'utilisateur se déplace vers la cible, et diminué lorsqu'il est à l'intérieur de la cible. Ils démontrent que le pointage sémantique est plus rapide et plus précis que les opérations effectuées à gain constant. Cette technique de pointage peut diminuer en efficacité lorsque plusieurs cibles affectées d'un poids sémantique (donc modifiant le gain du pointeur) se trouvent dans la trajectoire du curseur vers sa cible. Ce curseur va alors être ralenti, alors que sa cible effective est plus loin, diminuant dès lors les performances [26].

FIGURE 1.26 – (**Nouvelle conception pour les barres de défilement**) : (a) version originale. (b) nouvelle version : espace visuel et (c) espace moteur.

   Blanch fait remarquer dans ([12]) qu'un prenant en compte le profil d'accélération de la souris, le système peut ignorer l'information sémantique d'une cible lorsqu'il est manifeste que cette cible se trouve dans la trajectoire du curseur, mais que l'utilisateur ne désire pas s'y arrêter. Cette dernière modification apportée au *Pointage Sémantique* permet de rendre la technique plus robuste lorsque des cibles se trouvent sur la trajectoire du pointeur.
8. Predictive Pointing
   Lank *et al.*([80]) modélisent le point d'arrivée du curseur, à partir des informations de trajectoire, à l'aide du *Predictive Pointing*. La technique peut identifier une cible spécifique 42% du temps, et est adjacente à la cible 39% du temps, même

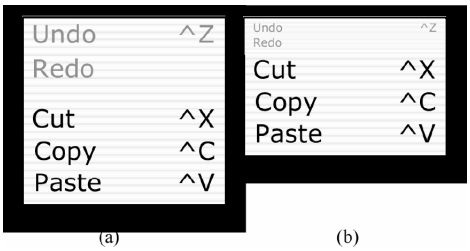

FIGURE 1.27 – (**Nouvelle conception pour les menus**) : (a) version visuelle inchangée. (b) version dans l'espace moteur.

FIGURE 1.28 – (**Nouvelle conception pour les boutons**) : (a) version visuelle inchangée. (b) version dans l'espace moteur.

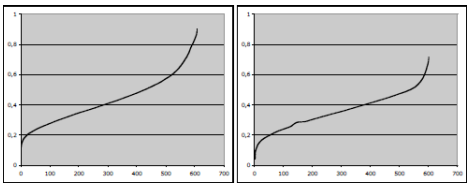

FIGURE 1.29 – Trajectoires typiques d'un pointage (abscisse : position du curseur, ordonnée : temps). A gauche, sans le pointage sémantique et à droite avec un pointage sémantique ayant une échelle de 4.

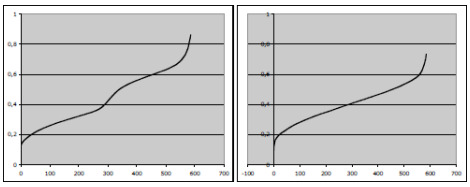

FIGURE 1.30 – Pointage sémantique avec un distracteur en 300. A gauche, le pointage sémantique classique et à droite en tenant compte de la vitesse et de l'accélération.

sur des cibles très petites (15 pixels de large). Cette technique est active dans la première partie du geste, avant le premier sous-mouvement. Le *Predictive Pointing* donne alors une efficacité maximum dans la prédiction à 80% de la longueur de la trajectoire.

La technique a été développée sur Tablet PC, avec un stylet. Ruiz et Lank ([94]) l'étendent à la souris, avec une efficacité maximum dans la prédiction à 90% de la trajectoire. La technique identifie alors une cible spécifique 35% du temps, et est adjacente à la cible 25% du temps (i.e. la technique peut identifier un sous-ensemble de cibles 60%) du temps.

9. *Delphian Desktop*

   Takeshi *et al.*([6]) utilisent la prédiction du pointage pour sélectionner une cible à l'aide du *Delphian Desktop*, en faisant "sauter" le curseur vers la cible prédite. Le modèle utilisé pour la prédiction est beaucoup plus simple que celui de Lank lors du *Predictive Pointing*. La technique est efficace quand la distance à la cible choisie est supérieure à 800 pixels, les prédictions étant fausses lorsque les trajectoires sont plus courtes. Afin de ne pas perdre l'utilisateur, le système indique avec un signal sonore et une animation visuelle le "saut" du curseur. Takeshi *et al.*proposent de combiner la technique avec les différentes techniques d'expansion de cibles, ou le *Pointage Semantique*, en prédisant le point d'arrivée du curseur lorsque le curseur atteint le pic de vitesse par exemple. Tout comme le *Pointage Semantique*, l'efficacité de la technique diminue avec la concentration des cibles autour de la cible à déterminer.

10. *Angle Mouse*

    L'*Angle Mouse* est une technique proposée par Wobbrock[106], qui diminue le gain de la souris lorsque le curseur s'écarte de l'axe principal initié au début du mouvement. Bien que la technique n'apporte pas de gain significatif pour les personnes valides, elle est 10,3% plus efficace que la technique Windows par défaut et 11% plus efficace que les cibles collantes pour les personnes handicapées.

11. Souris à pression proportionnelle

    Sato et al.[96] proposent une souris dont le nombre de dots envoyés au système pour un déplacement de la souris est proportionnel à la force appliquée par l'utilisateur sur le périphérique.

En conclusion de cet état de l'art sur les techniques d'interaction, nous pouvons nous rendre compte du problème récurrent à ces techniques : l'implémentation logicielle de la technique. En effet, les systèmes d'exploitations actuels, ainsi que les plateformes logicielles associées aux divers langages d'exploitation fournissent aux programmeurs des bibliothèques de *widgets* et de *techniques*. Ainsi, une technique qui n'est pas fournie en standard ne sera pas implémentée pour l'utilisateur final de manière répandue. De plus, il n'est pas possible d'accéder facilement aux éléments de l'interface afin de les augmenter avec des informations extérieures, limitant les expériences de type pointage sémantique ou définition par avance des cibles à choisir. Enfin, Balakrishnan, dans son étude sur les techniques de pointage ([8]) note que le critère final de choix d'une technique est son acceptation par les utilisateurs. Une technique provoquant des distractions visuelles ou motrices trop importantes sera rejetée par les utilisateurs.

1.4 Conclusion

Depuis 1963 et l'introduction de la souris par Engelbart, les interfaces graphiques ont pris la place de la ligne de commande comme métaphore d'interaction principale avec les systèmes d'exploitation. L'interaction à l'aide d'un curseur de souris y est omniprésente : Johnson en 1993 - [69] - montre que l'utilisateur passe près de 65% du temps à déplacer la souris lorsqu'il manipule l'ordinateur.

Les utilisateurs manipulant la souris ne font plus la distinction entre le fonctionnement du curseur et celui de la souris. Ils transfèrent les propriétés du curseur (vitesse du déplacement) à celui du *périphérique souris*. L'outil qui sert de périphérique devient un prolongement naturel du fonctionnement du bras et de la main, reproduisant et étendant le geste naturel du pointage. Dotov *et al.* montrent que lorsque le fonctionnement du périphérique devient transparent et s'efface pour l'utilisateur, les mouvements de l'utilisateur suivent une loi qui se répète de sujet en sujet ([38]). Lorsque l'outil est défectueux, cette loi n'est plus observable, et nous savons alors que les performances de l'utilisateur vont se dégrader.

Dès lors, l'optimisation de la tâche de pointage, même minime, peut amener des gains substantiels pour l'utilisateur, que ce soit en temps, en précision ou en diminution de la fatigue d'utilisation. Cette constatation assure la légitimité de notre démarche présentée ci-après.

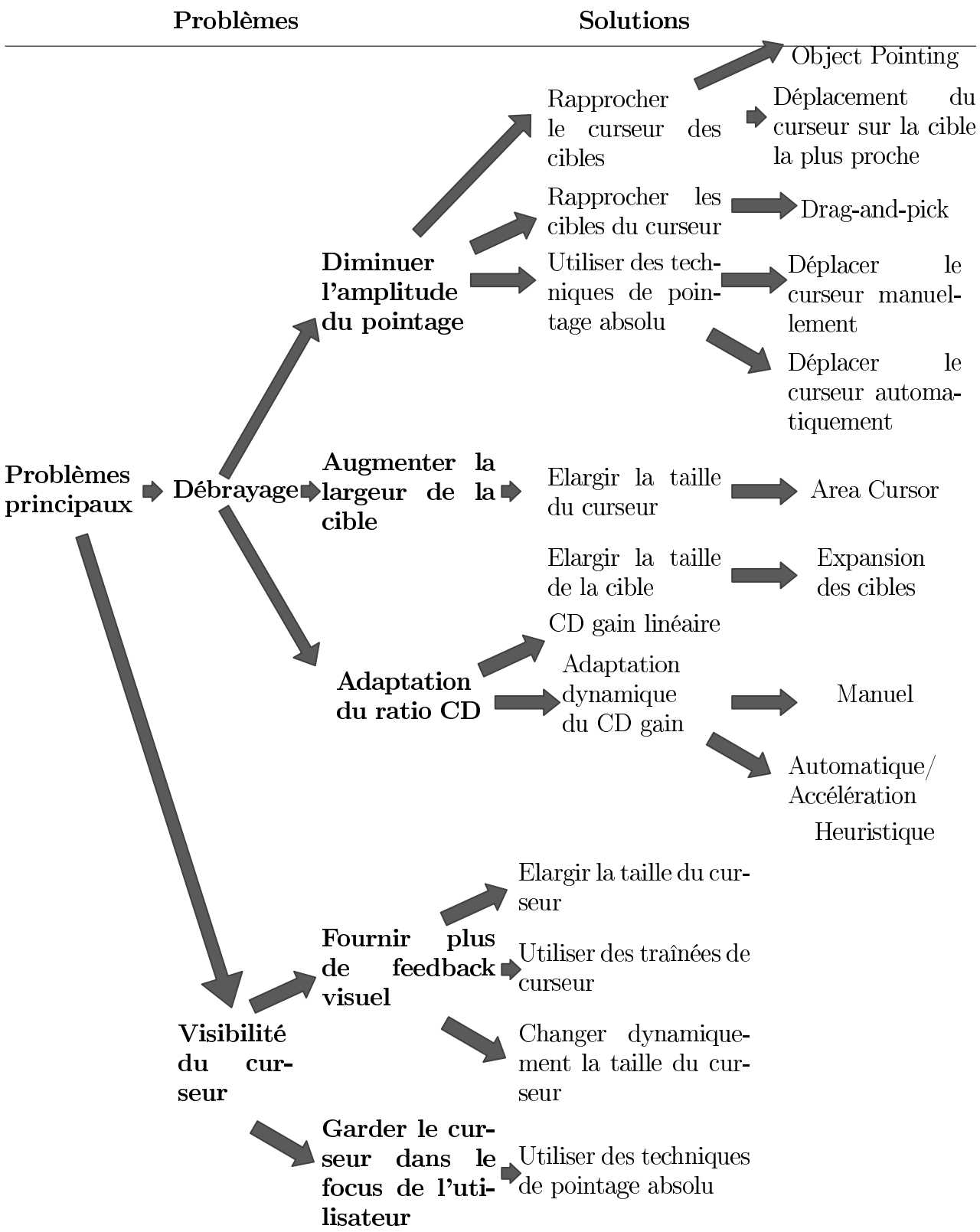

FIGURE 1.31 – Espace des solutions possibles pour améliorer les techniques de pointage.

Chapitre 2

Périphériques et mesure objective de leur courbe de transfert

Dans ce chapitre, nous présentons les différents périphériques physiques disponibles pour interagir avec un système informatique. Parmi ces périphériques, nous nous attachons ensuite à l'étude de la souris plus particulièrement, et de sa relation avec le système d'exploitation. Les courbes d'accélérations des différents systèmes d'exploitations, dont le principe de fonctionnement a été introduit dans le chapitre précédent, ne nous sont pas connues, et ne sont pas référencées dans la littérature. Les connaître est malgré tout nécessaire, ne serait-ce que pour en créer de nouvelles et se comparer à l'existant.

Dès lors, nous montrons également dans ce chapitre comment concevoir un périphérique qui se fait passer pour une souris face à un système d'exploitation. Cela permet de récupérer les informations relatives à l'accélération du geste de pointage pour ce système d'exploitation, sans connaissance à priori dudit système.

2.1 Périphériques d'entrée

Afin d'interagir avec un système informatique, l'utilisateur a besoin d'un périphérique d'entrée. Face à la diversité des périphériques disponibles, nous présentons une taxonomie de ceux-ci, afin de distinguer plus tard ceux qui bénéficieront des techniques présentées dans ce document.

2.1.1 Taxonomie des périphériques

Buxton ([18]) classe les périphériques manipulés par la main sur un graphe à deux dimensions, en fonction des propriétés physiques mesurées, et du nombre de degrés de liberté, ainsi que du type de contact avec l'utilisateur [2]. La classification se fait selon 6 critères : continu/discret (un périphérique continu peut parcourir une plage de valeurs, alors qu'un périphérique discret ne peut prendre qu'un nombre fini d'états), le moyen d'action (main, pied, voix), la grandeur physique mesurée (position, angle, pression), le

2. Un degré de liberté définit la possibilité de mouvement dans l'espace suivant un axe, en rotation ou en translation. Les degrés de liberté sont au nombre de six : trois en rotation et trois en translation.

nombre de degrés de liberté (1, 2 ou 3), et enfin le type d'interaction (qui dépend de la méthode de saisie du périphérique). Card *et al.* étendent cette classification [21] en la généralisant aux périphériques continus et discrets, et en séparant les notions de rotation et de translation. La perception cognitive de ces taxonomies est ajoutée par Jacob et Sibert [63], afin de faire la distinction entre les périphériques intégrables (les degrés de liberté sont considérés comme un seul bloc), et les périphériques séparables (les degrés de liberté sont traités séparéments).

		degré de liberté	
		linéaire	rotatif
position	absolue	position P	angle R
	relative	mouvement dP P	delta angle dR
force	absolue	force F	couple T
	relative	delta force dF	delta couple dT

FIGURE 2.1 – Classement des différentes interactions possibles et des degrés de libertés pouvant être associés.

Ces interfaces présentent des degrés de résistances différents à l'utilisateur. Ces résistances peuvent être isotoniques, isométriques ou élastiques.

— **Interfaces isotoniques**

Un périphérique isotonique a une résistance nulle. Cela comprend donc les périphériques mesurant une position ou une variation de position, telles les souris.
Entre ces périphériques, certains sont relatifs, et peuvent donc être débrayés. Par exemple, en soulevant la souris, l'utilisateur peut la positionner ailleurs dans l'espace moteur, tout en laissant l'origine de cette souris identique dans l'espace de travail. Une souris volante, telle que la Flying Mouse (de Logitech) est absolue, puisque la troisième dimension est également utilisée, et n'est pas disponible pour un débrayage (ce débrayage est alors modal, en appuyant sur un bouton). Parmi les interfaces isotoniques, une séparation peut être établie entre les périphériques isotoniques relatifs et les périphériques isotoniques absolus. La résistance d'un périphérique isotonique est constante (elle ne varie pas suivant la force ou la distance parcourue).

interfaces isotoniques relatives

Lorsqu'un périphérique dispose de deux degrés de libertés (sans chaîne cinématique), il est relatif. Le troisième degré de liberté est utilisé pour débrayer le périphérique. L'utilisateur peut alors déplacer le périphérique dans l'espace sans déplacer le curseur à l'écran, pour recentrer la souris sur sa table, par exemple.

interfaces isotoniques absolues

Lorsque la position absolue d'un périphérique est mesurée dans le plan, il n'est plus possible de débrayer en utilisant un troisième degré de liberté. Un bouton est alors utilisé sur le périphérique pour indiquer ce débrayage lorsqu'il est pressé. Les périphériques à trois degrés de libertés sont quand à eux toujours absolus, car l'utilisateur ne peut pas débrayer dans une quatrième dimension.

— **Interfaces isométriques**
Un périphérique isométrique mesure des forces ou des couples, mais reste fixe. Nous trouvons dans cette catégorie, par exemple la Spaceball, ou le Trackpoint sur les portables d'IBM.

— **Interfaces élastiques**
Les joysticks offrent également une résistance, comme les interfaces isométriques, mais elle est variable. Les périphériques élastiques offrent un retour kinesthésique absent des périphériques isométriques, facilitant leur apprentissage. Le périphérique revient en position neutre une fois la poussée effectuée par l'utilisateur. La Space Mouse est un exemple de périphérique pour les environnements 3D.

Au vu de cette classification, la souris est alors un périphérique continu, actionné à la main, disposant de 2 degrés de liberté en translations, intégrables, et isotonique relatif.

Ces périphériques présentent différents modes de contrôle, afin de choisir quelle caractéristique du périphérique va être utilisée afin d'interagir avec le système.

2.1.2 Modes de contrôle

Afin de pouvoir interagir avec le système, un périphérique doit mettre en relation une de ses caractéristiques, une grandeur, avec une caractéristique du système informatique manipulé.

Fonction de transfert

Une fonction de transfert met en relation la force que l'utilisateur exerce sur le périphérique (pour un périphérique isométrique ou élastique) ou le changement de position (pour un périphérique isotonique). Ces fonctions de transferts peuvent permettre d'effectuer deux types de contrôle.

— ***Contrôle en position***
Lorsque la position ou la force exercée par l'utilisateur sur le périphérique est utilisée directement pour se positionner dans l'environnement du système informatique, nous parlons d'un *contrôle en position*. Une souris qui n'utilise pas de courbe d'accélération, par exemple, va déplacer le curseur à l'écran d'un multiple de la distance parcourue sur la table.

— ***Contrôle en vitesse***
Lorsque c'est la *vitesse* (et non plus la position) dans l'environnement informatique qui dépend de la position ou de la force exercée sur le périphérique, nous parlons de *contrôle en vitesse*.

2.1.3 Périphérique direct ou indirect

Cette relation entre caractéristique du périphérique et caractéristique du système manipulé peut-être exprimée sous forme de relation directe ou indirecte. Lorsque le périphérique utilisé permet d'indiquer physiquement sur l'écran la position du pointeur, comme

un écran tactile, le périphérique est direct. Lorsqu'un périphérique introduit une distance entre la main de l'utilisateur et l'écran, il est indirect.

La relation entre le déplacement du curseur à l'écran et le déplacement du périphérique permet d'introduire la notion de *control display gain*.

2.1.4 *Control-Display gain*

Le *control display gain* a déjà été présenté avant (voir 1.3.1), nous ne présentons ici donc qu'un rapide résumé.

Le *CD gain* permet d'exprimer le rapport de déplacement du curseur à l'écran en fonction de la force ou de la vitesse appliquée par l'utilisateur sur le périphérique. Par exemple, dans le cas d'une souris, ce gain met en relation la distance parcourue sur l'espace de travail par la souris et l'espace parcouru à l'écran par le curseur. Ce gain peut-être constant, impliquant que les distances sont multiples l'une de l'autre, avec un facteur constant. La relation peut aussi être non linéaire, pour prendre non plus la distance parcourue par la souris, mais sa vitesse.

Différentes techniques d'interactions peuvent s'appliquer à ce *control display gain*, ou se combiner avec lui. Pour présenter ces techniques, nous allons nous restreindre à un périphérique.

2.1.5 La souris comme périphérique d'interaction

Dans la suite de notre travail, nous porterons notre attention plus spécifiquement sur les *souris* comme périphérique de pointage. Pour cela, nous présentons d'abord l'historique de sa conception, puis les techniques d'interactions qui lui sont attachées.

La souris

Douglas Englebart et ses collègues inventent la souris en 1963 au Standford Research Institute, obtenant un brevet en 1970 ([42]), pour remplacer le crayon optique. L'ingénieur Bill English fabrique une souris dans un petit casier en bois fermé au dessus, avec deux roues verticales touchant la table, placées à la perpendiculaire sous le casier. La souris ne dispose alors que d'un seul bouton, par manque de place (comme le rapporte English lui-même). English, Englebart & Berman ([43]) comparent la souris à d'autres périphériques de pointage (crayon optique, joystick, tablette graphique), et montrent qu'elle est la plus rapide pour réaliser une tâche de pointage.

Différentes améliorations seront ajoutées à la souris, afin d'aider aux tâches annexes au pointage, comme une molette pour faire défiler horizontalement ou verticalement du texte, un suivi sans fil, ou des moteurs pour un retour vibro-tactile. Finalement, les roues et la boule nécessaire pour mesurer le déplacement de la souris seront remplacées par un capteur optique, qui fournit les mêmes informations de déplacement de la souris.

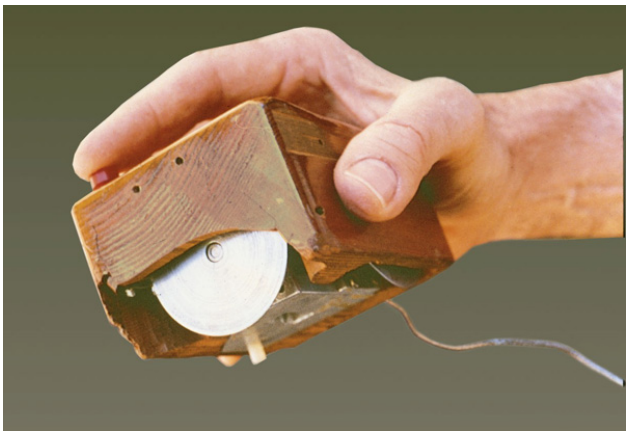

FIGURE 2.2 – Prototype de la première souris, proposée par Engelbart.

Balakrishnan *et al.* [7] énumèrent les propriétés de la souris qui la font toujours utiliser massivement comme périphérique d'interaction :

— ***Facteur de forme :***
Le facteur de forme de la souris, associé à son déplacement sur une surface plane permet de choisir la prise en main que l'utilisateur préfère. De plus, le bras peut se reposer sur la table. Ce facteur de forme correspond à un profil, une forme, adaptée à la préhension humaine naturelle.

— ***Stabilité :***
La souris est assez lourde, avec une surface de contact relativement large, amortissant les tremblements naturels de la main. De plus, la souris reste dans l'état dans lequel l'utilisateur la place, alors qu'un joystick, un stylet ou un périphérique doit d'abord être "pertubé" pour être repris en main une fois lâché.

— ***Mode relatif et absolu :***
Les souris sont assez flexibles pour fonctionner en mode relatif ou absolu, et le mécanisme de débrayage implicite (soulever la souris) est suffisament naturel pour être appris rapidement.

— ***Ordre du contrôle :***
Zhai ([111]) montre que pour des tâches 3D usuelles comme la manipulation d'objets, les périphériphériques d'entrée en contrôle de position sont meilleurs que les périphériques en contrôle en vitesse (*"rate control"*). Une souris peut-être utilisée en contrôle en vitesse en utilisant une courbe de transfert appropriée . L'inverse n'est pas vrai : les périphériques à capture de force ne peuvent fonctionner en mode de contrôle de position.

— ***Mise en correspondance avec le curseur :***
La mise en correspondance du déplacement du curseur avec le déplacement de la souris est naturel et intuitif (par exemple, déplacer la souris vers la gauche déplacera le curseur vers la gauche, mais l'utilisateur peut choisir d'inverser ce sens de déplacement). D'autres périphériques n'ont pas une mise en correspondance aussi immédiate (par exemple les joysticks, qui ne se meuvent pas dans le même plan

que celui du curseur). De même, lorsque l'utilisateur lâche la souris et la reprend plus tard, le curseur reste immobile, facilitant sa réacquisition par l'utilisateur.

— ***Position des boutons :***
Les boutons sur la souris sont manipulés dans l'axe perpendiculaire au mouvement de celle-ci. L'utilisateur peut alors facilement les utiliser sans déplacer la souris par inadvertance.

— ***Familiarité :***
Plusieurs générations d'utilisateurs informatiques ont maintenant grandi en manipulant la souris, accentuant ainsi la familiarité avec le périphérique et de ce fait améliorant les performances à l'utilisation. L'omniprésence de la souris lorsqu'il s'agit d'utiliser un périphérique de pointage et la familiarité du grand public avec son utilisation demandent que les innovations sur le facteur de forme ou l'utilisation ne soient qu'incrémentales, afin de ne pas perdre le bénéfice de son utilisation ubiquitaire.

Card, English & Burr montrent ([100]) qu'un utilisateur peut pointer avec la souris presque aussi bien qu'avec la main (ou qu'avec un stylet), et qu'une souris est systématiquement meilleure qu'un joystick ou que les touches d'un clavier pour sélectionner une cible. Cela se montre vrai pour toutes les combinaisons de taille et de distance dans leur expérience. A l'époque de l'expérience, Card note que la familiarité avec le périphérique importe beaucoup. Par exemple, les débutants ne savaient pas comment réagir avec leur souris lorsque le pointeur atteignait le bord de l'écran. Le taux d'erreurs, pour toutes les tailles de cibles, reste inférieur à 5% (les cibles de 1 caractère de large montent à 8%, mais malgré cela, les performances restent meilleure que le deuxième meilleur périphérique pour cette tâche, les touches du clavier, qui montent à 10%). Pour le positionnement, la souris reste systématiquement à moins de 1,5 seconde pour atteindre la cible, quelle que soit sa taille, alors que le joystick, second meilleur périphérique dans ce cas, met 200 millisecondes de plus).

Hybrid Finger Mouse : une variante pour le périphérique de pointage

Cao *et al.*([19]) proposent de combiner une souris, avec une prise en main classique, à un petit module, déplacé uniquement par l'index, en conjonction malgré cela avec le reste de la souris. Ce genre de dispositif permet de combiner les performances du poignet et de la motricité fine du doigt.

Pour une des techniques testées, la plus efficace, la position du curseur de la souris est calculée comme la *somme* du déplacement de la souris et du déplacement de la petite partie mobile (en effet, il n'est pas possible en pratique de déplacer la main sans déplacer le doigt également - nous pouvons voir sur la figure 2.3 l'utilisation de la partie "souris" pour déplacer le poignet, et la partie, beaucoup plus petite, qui y est attachée, pour le doigt). Dès lors, le fait de n'utiliser que le doigt déplace le curseur avec un gain divisé par deux par rapport au gain de l'ensemble du périphérique. Ainsi, l'utilisateur peut utiliser la main pour les gestes grossiers, et le doigt pour les geste précis. Le système hybride améliore les performances sur le temps et le taux d'erreurs par rapport à un système classique (par exemple, le taux d'erreur passe de 4,7 % pour la souris hybride à 7,8 % pour une souris normale).

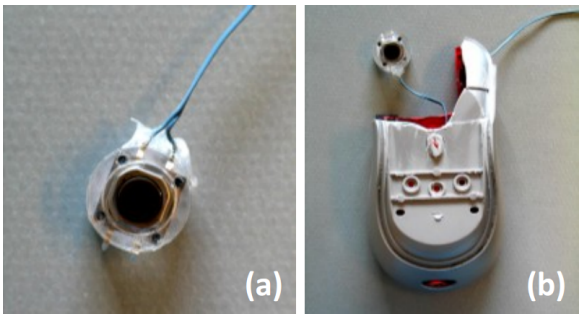

FIGURE 2.3 – A) Finger Mouse et B) Hybrid Finger Mouse

Nous avons présenté les différents périphériques d'interactions et leur caractéristiques disponibles pour l'accès aux systèmes informatiques. Dans la suite du travail présenté ici, nous nous intéressons plus particulièrement à la souris comme périphérique d'interaction, et aux courbes d'accélération comme technique logicielle.

2.2 Mesure objective de la courbe de transfert d'un système

Les systèmes d'exploitation actuels ne fournissent pas la courbe de transfert qu'ils appliquent pour accélérer le déplacement du curseur à l'écran. La suite du présent chapitre propose une solution technique pour récupérer de manière objective la courbe d'un transfert d'un tel système.

Dans ce travail, nous nous restreignons à l'interaction avec une souris, et nous présentons la solution technique développée pour déterminer cette courbe d'accélération (ou la famille de courbes d'accélération) utilisée par un système, sans connaissance au préalable du système, et sans avoir accès à son code source.

Afin d'interagir avec un système graphique moderne, plusieurs solutions sont aujourd'hui disponibles. Pour l'interaction directe, l'écran tactile permet à l'utilisateur de toucher directement la cible. Pour une interaction indirecte, un joystick, une souris ou un touchpad sont les dispositifs les plus courants. Ces périphériques déplacent un curseur à l'écran, transposition du doigt de l'utilisateur, afin de réaliser les tâches de pointage.

Les systèmes de pointage indirect mettent en correspondance le déplacement du périphérique d'interaction avec le déplacement d'un curseur à l'écran. Nous utilisons le terme de courbe de transfert, ou de courbe d'accélération pour exprimer cette mise en correspondance. Cette courbe d'accélération peut être constante, le déplacement du périphérique sur la surface de travail déplacera alors le curseur de la même distance à l'écran.

Les systèmes modernes proposent une implémentation non linéaire : lorsque la main de l'utilisateur se déplace à vitesse relativement faible, le déplacement à l'écran est inférieur au déplacement réel, et lorsque la main se déplace rapidement, le curseur se déplace d'une distance supérieure à la distance couverte par le périphérique. Cette méthode permet d'optimiser la tâche de pointage, en permettant un pointage de précision à faible vitesse,

supérieur aux contraintes physiologiques du bras et de la main, et en économisant de l'énergie à l'utilisateur lors des déplacements longs à travers l'écran.

Il n'existe pas de méthode connue pour construire une courbe de pointage optimale prenant en compte le périphérique utilisé et les préférences de l'utilisateurs. De plus, les systèmes actuels ne fournissent pas l'implémentation de l'accélération qu'ils appliquent au pointage.

Afin de pouvoir comparer les différents systèmes d'exploitation quand à l'efficacité de l'accélération employée sur le pointage, nous devons extraire du système ladite courbe d'accélération.

2.2.1 La relation souris système d'exploitation

La souris

Une souris est un périphérique qui mesure sa position de manière relative. Elle envoie au système la variation de sa position par rapport à la mesure précédente. La position est mesurée avec une certaine résolution, généralement exprimée en DPI (*dots per inch* ou points par pouce). Les informations numérisées (exprimées en nombre de points) sont stockées sur 8 bits pour chaque axe et envoyées, dans le cas d'une souris USB, à 125 Hz sur le bus (par défaut dans la norme USB-HID). La résolution par défaut d'une souris est de 400 DPI, sachant que sont commercialisées des souris jusqu'à 5600 DPI. A partir de la résolution de la souris exprimée en DPI (Res), du nombre de bits utilisé pour stocker les informations ($Taille$) et de la fréquence d'envoi des données ($Freq$), il est possible de calculer la vitesse maximale mesurable ($VitesseMax$) exprimée en $m.s^{-1}$ selon l'équation 2.1.

$$VitesseMax = Freq \times 2^{Taille-1} \times \frac{0,0254}{Res} \tag{2.1}$$

Ainsi la vitesse maximale mesurable pour une souris 400 DPI branchée sur un port USB à 125 Hz est de 1,016 $m.s^{-1}$. Il est intéressant de voir que cette vitesse chute à 0,127 $m.s^{-1}$ dans le cas d'une souris à 3200 DPI. Pour cette raison les constructeurs de souris ont généralement prévu la possibilité d'envoyer les informations à 1000 Hz. Dans ce cas, on retrouve une vitesse maximale mesurable de 1,016 $m.s^{-1}$.

Le système d'exploitation lit les informations brutes reçues (en nombre de points) et en déduit une vitesse du périphérique en supposant que sa résolution est de 400 DPI [3]. Que ce soit sur Windows, Mac ou Linux, le changement notable de comportement du pointeur quand on change de résolution de souris montre que *ces systèmes n'ont aucune connaissance de cette information.* Le système prend également en compte la résolution de l'écran et sa fréquence de rafraîchissement pour obtenir la vitesse du pointeur désirée en fonction du gain utilisé. Les systèmes d'exploitation peuvent également appliquer un mécanisme de sous-pixelisation qui consiste à enregistrer les déplacement de la souris sans forcément bouger le pointeur à l'écran[1].

3. http ://www.microsoft.com/whdc/archive/pointer-bal.mspx

2.2.2 Les courbes existantes

Hormis les systèmes basés sur Unix, les systèmes d'exploitations grand public ne fournissent pas leur code source. Ainsi, lorsque l'utilisateur final veut modifier le comportement du système, il est contraint par les outils mis à disposition par la société proposant le système d'exploitation. De ce point de vue, Windows et Mac OS X fonctionnent comme une boîte noire. En effet, les différentes informations disponibles dans un éventuel panneau de configuration ne permettent pas de connaître les traitements internes réalisés par le système sur les données qu'il reçoit en entrée.

Lors de la mise en service commerciale de Windows XP, Microsoft a mis en ligne quelques informations sur le stockage par le système des courbes d'accélération de la souris, disponible à l'adresse *http ://www.microsoft.com/whdc/archive/pointer-bal.mspx* . Ces données, bien qu'utiles pour obtenir une vision succincte du système, ne sont plus actualisées. De plus, il n'y a aucune assurance que l'implémentation logicielle au sein du système respecte cette description. En effet, le fonctionnement du système du point de vue de l'utilisateur dépend autant de la couche d'exploitation que des pilotes fournis par le constructeur du périphérique.

Apple ne communique pas d'informations quand aux techniques utilisées avec Mac OS X, mais nous pouvons accéder à ces informations d'une manière détournée. En effet, Mac OS est basé sur le noyau Mach, *opensource*, et Apple met en ligne les sources du noyau de son système. Dès lors, Casiez a pu analyser les fichiers sources de la couche de gestion des périphériques (voir [29]), afin de reconstruire les courbes d'accélérations utilisées par le système. Malgré cela, et comme pour Windows, aucune garantie n'est apportée quand au parcours logique utilisé par le système d'exploitation au sein des différentes couches logicielles. De plus, l'analyse de la seule couche de gestion des périphériques ne nous garanti pas que d'autres modifications ne soient effectuées ailleurs par le système.

Balistique pour Windows XP

La *balistique* est le terme utilisé dans les documents techniques chez Microsoft pour parler des techniques relatives à l'accélération du curseur. Le terme est à comprendre dans le sens de la description d'un geste balistique (voir le chapitre précédent, dans l'état de l'art).

Jusqu'à Windows XP, l'accélération du curseur était appliquée séparément sur les axes X et Y. Dès lors, l'utilisateur qui déplaçait la souris en un cercle parfait sur la table obtenait plutôt un carré aux coins arrondis à l'écran.

Pour Windows XP, le système calcule la vitesse de la souris, qui lui envoie unités physique de déplacement (*mickeys*) selon l'équation suivante :

$$V_{souris} = mickey \times FrquenceduBus \times \frac{0,0254}{RsolutionSouris} \tag{2.2}$$

Avec la vitesse maximale enregistrable par la souris exprimée par l'équation :

$$V_{max} = 2^{Taille-1} \times FrquenceduBus \times \frac{0,0254}{RsolutionSouris} \tag{2.3}$$

Par défaut, la fréquence du bus USB, définie dans la norme USB-HID [4], est de 125 Hertz. Cette même norme définit la résolution par défaut d'une souris à 400 *DPI* (*dots per inch* ou *points par pouce*). De même, les *mickey* (la plus petite unité de déplacement) sont envoyés sur un octet, avec le premier bit comme bit de signe. Cette valeur appartient alors à [0..127].

Ainsi la vitesse maximale mesurable par une souris 400 DPI branchée sur un port USB à 125 Hz est de 1,016 $m.s^{-1}$. Certains constructeurs de périphériques proposent aujourd'hui des souris avec une résolution plus élevée, à 3200 DPI voire 5600 DPI maintenant. Il est intéressant de noter que la vitesse maximale enregistrable par la souris chute à 0,127 $m.s^{-1}$ dans le cas d'une souris avec une résolution de 3200 DPI. Pour cette raison les constructeurs de souris ont généralement prévu la possibilité d'envoyer les informations à 1000 Hz à l'aide de pilotes propriétaires. Dans ce cas, on retrouve une vitesse maximale mesurable de 1,016 $m.s^{-1}$. Malgré cela, il est difficile d'atteindre cette valeur en pratique, l'électronique de la carte-mère limitant les performances. Ainsi, une souris avec une résolution élevée installée sur un système ne disposant pas des pilotes appropriés sera paradoxalement moins performante qu'une souris avec une résolution plus faible (ce qui sera systématiquement le cas pour une souris à 5600 DPI, *même disposant des pilotes constructeurs*, puisqu'elle va mesurer des vitesses inférieures à une souris à 3200 DPI). *De même, un logiciel implémentant sa propre méthode pour accéder aux périphériques d'interaction contournera le pilote du constructeur, et limitera à nouveau le périphérique.*

Lorsque le système reçoit l'information de vitesse de la souris, il peut alors la mettre en correspondance avec la vitesse du curseur à l'écran. La vitesse du curseur se définit à l'aide de l'équation 2.4

$$V_{pointeur} = mickey \times Frquencedel'cran \times \frac{0,0254}{Rsolutioncran} \tag{2.4}$$

Pour Windows XP, l'équipe de développement a réalisé une étude d'utilisabilité, et en a déduit une courbe de transfert *parente*, qui sert à construire l'ensemble des courbes. Cette courbe est constituée de 5 couples de points mettant en relation la vitesse de la souris et la vitesse du curseur. La courbe est interpolée linéairement entre chacun de ces points.

A partir de cette fonction parente, une famille de courbes est extrapolée, chacune de ces courbes correspondant à une valeur sur le panneau de sélection d'accélération de la souris.

Pour cette famille de courbes, quelle que soit la valeur de la courbe choisie, le début de la courbe donne un gain inférieur à 1, afin de s'assurer que tous les pixels de l'écran soient accessibles. De plus, les mouvements entraînant des déplacements inférieurs à 1 pixel sont accumulés en mémoire, jusqu'à devenir effectivement visibles à l'écran (nous obtenons alors une précision *inférieure* au pixel). Avant Windows XP, le système calculait l'accélération indépendamment sur X et sur Y. Dès lors le mouvement du pointeur était biaisé vers l'axe de plus grand déplacement (ce qui donnait des carrés arrondis plutôt que des cercles parfaits lorsqu'ils étaient tracés à la souris). Pour résoudre ce problème, le système calcule maintenant le vecteur de déplacement, sur lequel est alors appliqué

4. *http ://www.usb.org/developers/hidpage/*

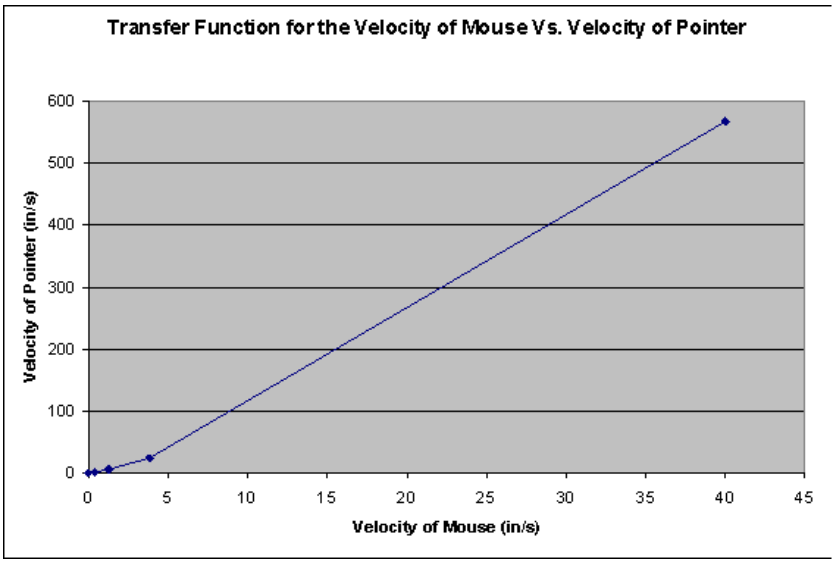

FIGURE 2.4 – Fonction de transfert exprimant la vitesse du curseur à l'écran sur la vitesse de la souris sur la table

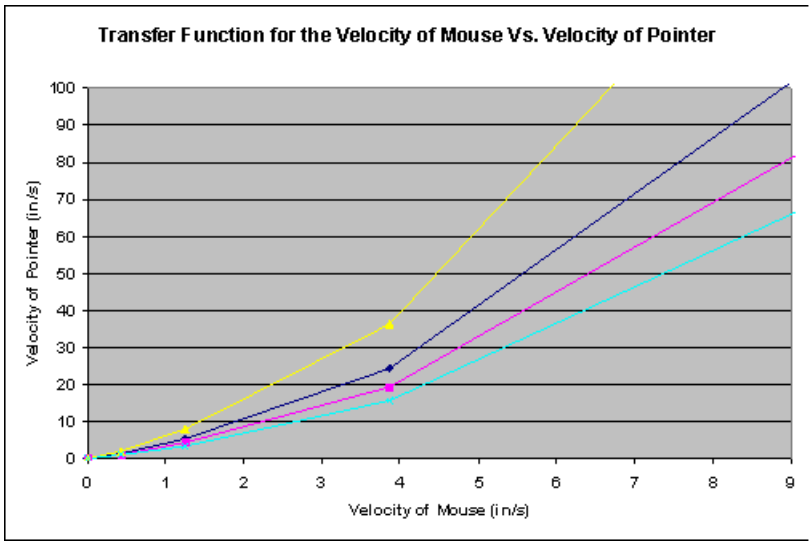

FIGURE 2.5 – Famille de Fonctions de transfert extrapolée à partir de la fonction d'origine

l'accélération.

2.2.3 Un dispositif d'acquisition objectif innovant

Le seul moyen d'obtenir de manière objective les informations d'accélération est de fournir des données connues au préalable en entrée, et d'observer en sortie les données du système en retour, c'est le principe de l'*ingénierie inverse*.

La souris d'un ordinateur servant à déplacer le curseur à l'écran, nous proposons ici un

système contrôlé par l'utilisateur qui se fait reconnaître par le système comme une souris. Il est ensuite aisé de mesurer la position du curseur à l'écran par voie logicielle.

En utilisant un micro-contrôleur connecté à une interface USB, et en se conformant aux spécifications HID, le périphérique est considéré comme une souris par le système d'exploitation (ce qui nous assure d'un traitement identique à celui d'une vraie souris). Ce périphérique envoie un nombre contrôlable de *dots* au système, de 1 à 127, et nous mesurons le déplacement du curseur à l'écran en pixels, sans modifier aucune des structures de données internes au système d'exploitation.

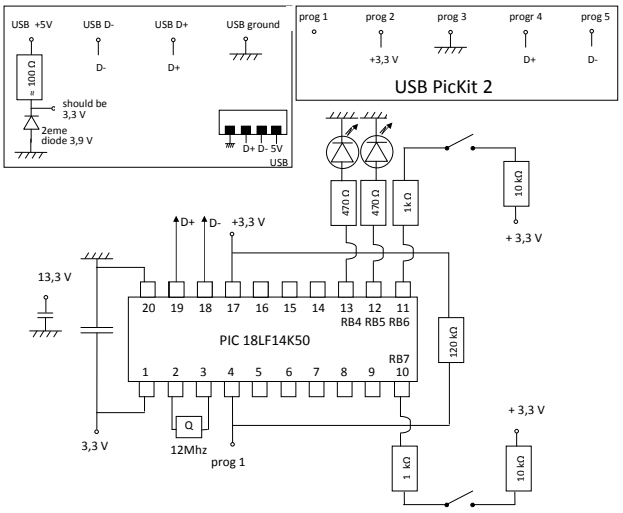

FIGURE 2.6 – Schéma électrique du dispositif d'acquisition.

Suivant les réglages du système, un faible nombre de *dots* envoyé au système peut ne pas être suffisant pour déplacer de manière effective le curseur à l'écran, compte tenu de la sous-pixelisation utilisée par certains systèmes. Ce problème peut être résolu en envoyant un nombre important de *dots* consécutifs, et de diviser ensuite la valeur en pixels mesurée à l'écran par ce nombre de paquets. Ainsi, si le système déplace le curseur de 100 pixels pour 10 paquets de 1 *dot*, il déplacera le curseur de 10 pixels pour 1 *dot*.

Nous avons relié le micro-contrôleur à un interrupteur pour pouvoir commander celui-ci. Lorsque l'utilisateur appuie sur l'interrupteur, le nombre de *dots* envoyés par le périphérique est incrémenté : il envoie d'abord 10 paquet de 1 *dot*, puis 10 paquets de 2 *dots* au clic suivant, et ainsi de suite jusqu'à 10 paquets de 127 *dots*, ceci à 125 Hz.

Sur la machine cible, un programme mesure la position du curseur en pixels après l'envoi des 10 paquets de *dots*. Il ramène ensuite le curseur à gauche de l'écran lorsque l'utilisateur appuie sur une touche du clavier. A chaque touche du clavier pressée, le programme enregistre le couple (*dots,pixels*) dans un fichier. A aucun moment l'utilisateur ne déplace le curseur à l'écran par un autre système que notre dispositif, pour ne pas ajouter d'information parasite, dont nous ne pourrions tracer la provenance.

Une première série d'essais a montré que les paquets contenant à la fois des informations de déplacement et d'appui sur les boutons pouvaient être interprétés comme des paquets ne contenant que des informations de pression sur les boutons. En effet, les

systèmes d'exploitation semblent affecter une priorité différente pour les actions de déplacement et les actions de clics. (*Cela peut avoir un sens du point de vue de l'utilisateur : ce dernier ne va pas remarquer un décalage de quelques pixels à l'écran, mais ressentira comme un ralentissement du système le fait que ses clics ne soient pas effectués immédiatement.*) Cette prioritisation apparente des données illustre le point que nous avons soulevé précédemment : nous ne connaissons pas à priori les opérations qu'effectue le système hors de la boucle [données du périphérique entrées-déplacement à l'écran], avec les données que nous lui fournissons.

Afin de s'assurer que la distance mesurée à l'écran en pixels correspond bien au nombre de *dots* effectivement envoyés par le micro-contrôleur, ce dernier n'envoie que des informations de déplacement. Le système d'exploitation combine les informations sur les axes X et Y d'une manière qui ne nous est pas accessible, nous n'envoyons les paquets de mouvement que sur l'axe des X. En effet, nous ne pourrions mesurer convenablement ce qui relève de la courbe d'accélération uniquement sur les axes X et Y, et ce qui relève de la formule combinant ces deux axes. De plus, la formule de combinaison de ces informations n'a aucune raison à priori d'être triviale, car plusieurs facteurs sont à prendre en compte sur les systèmes actuels : respecter la géométrie des gestes de l'utilisateur, tirer partie de la géométrie des écrans (normalement plus larges que hauts), s'accommoder des différences de performances physiologiques entre des mouvements sur l'axe parallèle au corps ou sur l'axe s'en éloignant.

Il est possible d'atteindre le bord de l'écran avec le curseur avant d'avoir atteint les 127 *dots* envoyés à l'ordinateur. Il est dans ce cas nécessaire de réaliser la même procédure que précédemment, mais en envoyant un nombre consécutifs de paquets moins important (par exemple 5 paquets de X *dots*, au lieu de 10 paquets). Pour garder une bonne précision, les informations récupérées avec 10 paquets de X dots peuvent servir à définir le début de la courbe, à faible vitesse, et les informations envoyées avec 5 paquets de X dots peuvent servir à définir la fin de la courbe (en pondérant de manière appropriée chaque ensemble de paquets).

L'ensemble de la méthode doit être répétée pour chacun des niveaux d'accélérations offerts par le système d'exploitation (11 sous Windows XP, 10 sous Mac OS).

Les figures 2.7 et 2.7 présentent, respectivement, les 10 courbes d'accélération pour Mac OS et les 11 courbes pour Windows XP, en exprimant dans les deux cas le nombre de pixels parcourus à l'écran en ordonnée, pour le nombre de dots envoyés en entrée en abscisse)

Notation Gain-Vitesse

Nous avons présenté la définition du gain dans la partie 1.3.1. Il nous semble essentiel pour la suite de notre travail de changer de mode de représentation des familles de courbes, et d'utiliser ce gain. En effet, les représentations usuelles (vitesse de la main contre vitesse à l'écran), tout comme dans les représentations données plus haut (points sur la table contre pixels à l'écran), ne nous permettent pas de s'abstraire de certains problèmes. Il est plus facile de comparer de manière intuitive les profils des courbes entre elles lorsque nous les exprimons dans l'espace vitesse de la main contre gain à l'écran. Ainsi, la courbe de gain constant sera une droite alignée sur l'ordonnée 1 (en bleu sur le graphique 2.11).

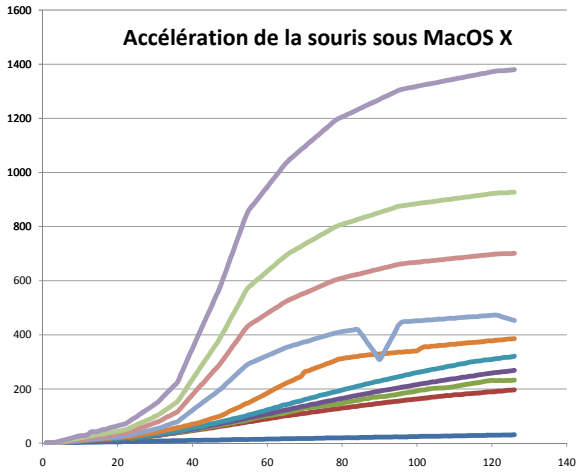

FIGURE 2.7 – Courbes d'accélération Mac OS X

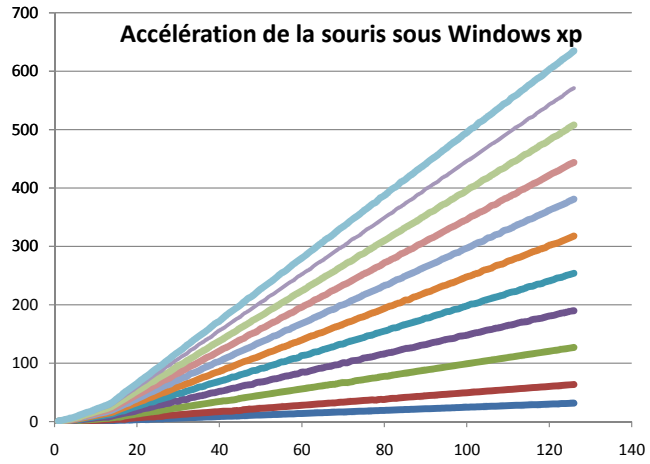

FIGURE 2.8 – Les 11 courbes d'accélération de la souris sous Windows XP

De plus, les courbes d'accélération exprimées en points contre pixels ne sont pas indépendantes des caractéristiques matérielles du système, et rendent donc difficile la comparaison à travers différents systèmes.

De même,une courbe de gain 4 : 1 sera plus haute sur le graphique que la courbe de gain 1 : 1.

Il est également aisé de voir qu'une courbe avec un gain plus petit que 1 à faible vitesse entraînera un déplacement à l'écran plus petit que celui de la main sur la table.

Nous pouvons alors exprimer les courbes que nous avons obtenues à l'aide de notre dispositif dans l'espace vitesse/gain.

Les artéfacts présents sur les courbes proviennent directement des données relevées sur les différents systèmes d'exploitation, et nous pouvons supposer qu'ils sont également présent lors d'une utilisation classique du système, hors de notre expérience. Ce point

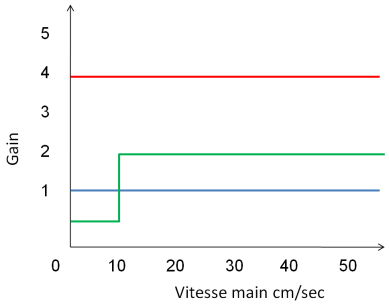

FIGURE 2.9 – Représentation des courbes d'accélération en utilisant la vitesse de la main sur la table en abscisse et le gain du curseur à l'écran en ordonnée.

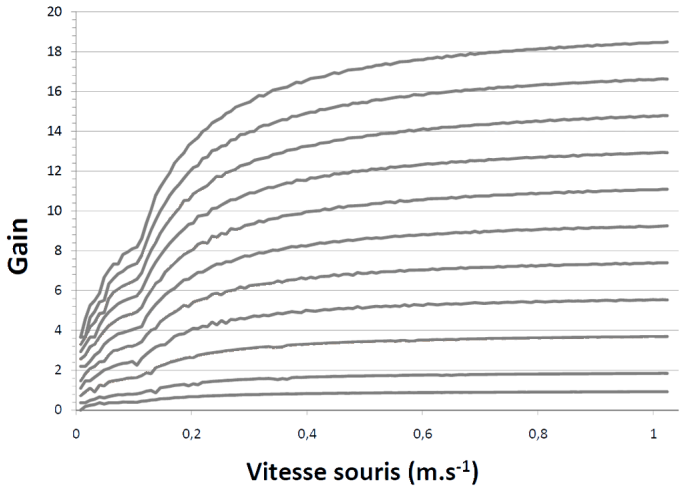

FIGURE 2.10 – Courbes d'accélération Windows XP dans l'espace vitesse/gain.

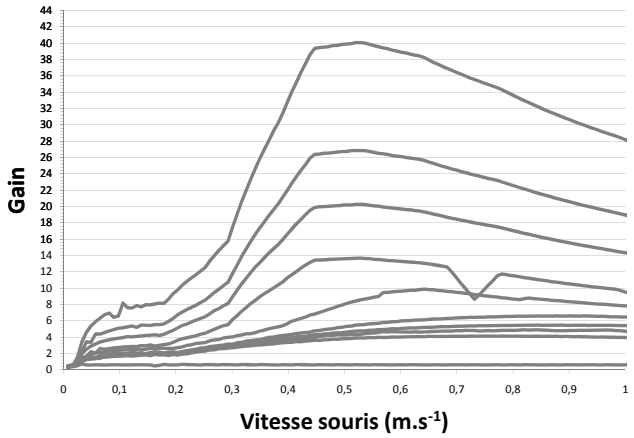

FIGURE 2.11 – Courbes d'accélération Mac OS X dans l'espace vitesse/gain.

illustre notre remarque sur l'implémentation logicielle des techniques présentées dans la littérature : l'utilisateur n'a aucune garantie que la transposition de l'un à l'autre n'a pas induit des erreurs de programmation.

Comparaison avec les courbes connues

Nous pouvons maintenant comparer ces données avec les données déjà connues sur les courbes d'accélérations des différents systèmes.

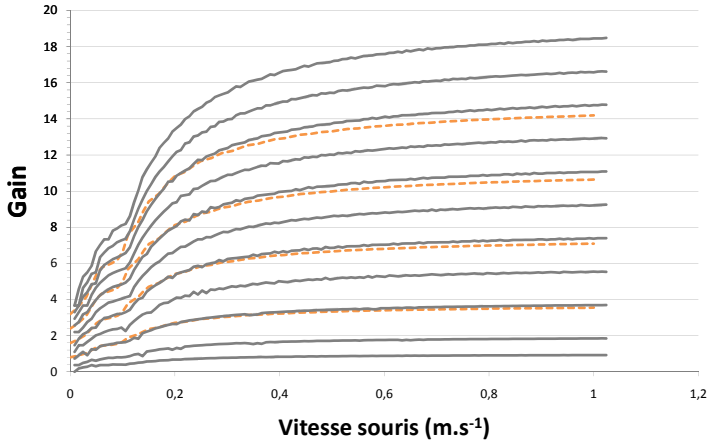

FIGURE 2.12 – Courbes d'accélération Windows XP (en gris) comparées aux courbes trouvées par Casiez ([29] - en orange pointillé)

La figure 2.12 présente les courbes identifiées par notre méthode, comparées aux courbes présentées par Casiez ([29] - ces courbes avaient été reconstruites à l'aide des informations disponibles dans la base de registre). Les courbes sont sensiblement identiques.

Les courbes Mac OS sont comparées dans la figure 2.13. La différence avec les courbes présentées dans le document ([29]) justifie à nouveau la méthode objective que nous avons présentée plus haut. Le peu de code source disponible d'un système d'exploitation (quand il est disponible) ne permet pas de déterminer les différents facteurs modificatifs appliqués hors de la boucle d'interaction par le système, ce qui peut expliquer la différence entre les courbes.

La visualisation des courbes MacOS dans l'espace vitesse/gain permet de voir un profil décroissant pour ces courbes au-delà d'une certaine vitesse. Ainsi, l'utilisateur pourrait avoir l'impression d'une accélération moins rapide de son curseur de souris au-delà d'une certaine vitesse (bien que l'on puisse douter, aux vitesses de la main et du curseur mises en jeu que l'utilisateur puisse réellement faire ce distinguo).

Une analyse des vitesses utilisées lors des pré-expériences a montré que les utilisateurs des systèmes concernés (Windows comme MacOS) restent 97% du temps inférieur à une vitesse de 40 cm/sec.

Cette dernière information précise la zone de travail pour les éventuelles améliorations à apporter à la technique. En effet, il est inutile de concevoir une technique d'interaction

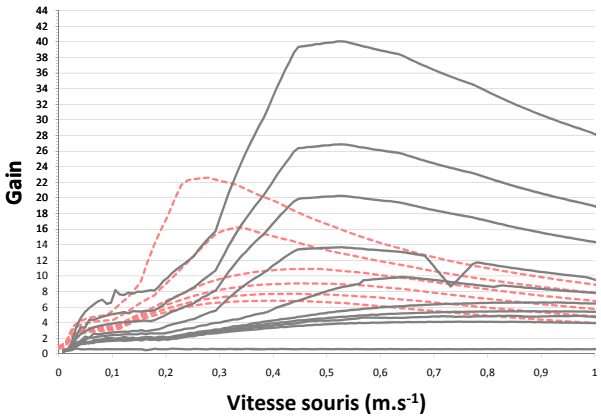

FIGURE 2.13 – Courbes d'accélération MacOS X (en gris) comparées aux courbes trouvées par Casiez ([29] - en rouge pointillé)

dont les améliorations doivent être reportées sur l'intégralité de la courbe d'accélération, quand les utilisateurs n'utiliseront pas effectivement l'intégralité du travail. A l'avenir, la construction d'une technique d'interaction doit voir réaliser au préalable une étude des plages de vitesses utilisées effectivement dans le contexte de l'interaction, avec un périphérique donné.

2.2.4 Discussion sur les résultats du périphérique

Nous avons présenté dans ce chapitre un dispositif innovant pour obtenir les courbes d'accélérations du curseur de souris, et cela quel que soit le système, sans accès au code source du système, et sans modification logicielle de ce dernier.

La version actuelle du système fonctionne actuellement dans une seule dimension. Il serait intéressant dans une version ultérieure de pouvoir fonctionner avec deux dimensions, afin de comprendre comment le système d'exploitation combine les informations de mouvement en X et en Y.

Les courbes obtenues sont différentes, visuellement et dans leur ressenti lors de l'utilisation. Nous pouvons en déduire que des choix différents ont été pris lors de la construction de ces courbes, choix qui vont alors déterminer les performances de l'utilisateur lorsqu'il utilisera ces courbes.

La mise à disposition des courbes d'accélération des systèmes connus va permettre à l'avenir de comparer de nouvelles techniques d'interaction aux techniques existantes, en permettant de s'abstraire du système sur lequel se déroule l'expérience. De plus, afin que cette comparaison soit pertinente entre machines et dispositifs différents, nous avons montré qu'il est nécessaire de s'abstraire du matériel pour raisonner en terme de vitesse de la main face à un gain (ou une vitesse à l'écran), et non plus en pixels, unité dépendante intrinsèquement du système sur lequel la technique a été conçue.

Lorsqu'un périphérique installe ses propres pilotes, nécessaires à son utilisation, le fonctionnement du système d'exploitation peut en être modifié. Ainsi, une souris de marque Logitech (par exemple) va remplacer les courbes du systèmes par ses propres courbes de

transfert. Il est alors nécessaire pour l'opérateur, lorsqu'il utilise la technique présentée dans ce chapitre, de vérifier qu'il mesure bien la courbe de transfert d'origine du système et non celle apportée par un périphérique dédié.

Enfin, l'étude a montré que l'utilisateur n'utilise qu'une sous-partie de l'accélération qui lui est proposée presque l'intégralité du temps. Les améliorations ultérieures pour ces techniques doivent donc se concentrer sur cette sous-partie, au risque de ne pas voir apparaître de gain réel à utiliser une nouvelle technique.

Une fois les courbes d'accélération des systèmes Windows et Mac OS obtenues, il nous faut maintenant les comparer afin de déterminer éventuellement laquelle est la plus "efficace" au regard des performances en temps et en précision. Cela nous permettra également d'avoir une référence pour d'autres comparaison avec de nouvelles courbes à l'avenir. La suite de notre chapitre se porte donc sur l'étude comparée des courbes Windows et Mac OS.

2.3 Conclusion

Nous avons présenté une méthode pour extraire une courbe de gain d'un système d'exploitation, sans connaissance précise sur ce dernier. Nous avons également présenté les diverses courbes de gain présentes dans les systèmes d'exploitation grand public courants (Windows et Mac OS).

En observant ces courbes, le lecteur peut se rendre compte de la différence entre l'implémentation théorique d'une courbe d'accélération du pointeur souris, et le comportement réel de son implémentation dans la machine.

Dans un avenir proche, nous aimerions pousser plus avant l'étude comparée des courbes de gains. La question de l'adaptation de la courbe de gain au comportement d'un utilisateur est également une question importante, qui mériterait d'être étudiée.

Pouvons nous aller au-delà de l'utilisation actuelle des fonctions d'accélération ? En effet, les systèmes actuellement utilisés utilisent des courbes "universelles" pour tous les utilisateurs. Est-il nécessaire de créer des courbes adaptées à chaque utilisateur ?

Pour répondre à ces question, il nous faut connaître les critères permettant de construire des courbes d'accélération optimales, et nous chercherons à proposer une solution dans le chapitre suivant, en partant d'une comparaison de performances sur les courbes Windows et Mac OS.

Chapitre 3

Quelques résultats expérimentaux sur les fonctions de gains

Suite au travail sur l'obtention des courbes d'accélération d'un système au chapitre précédent, nous présentons une expérience rapide qui compare les performances d'utilisation des courbes Windows et Mac OS, obtenues par notre dispositif.

Malgré un "ressenti" différent lors de l'utilisation de systèmes d'exploitation différent, il n'y a pas de comparaisons de performances disponibles entre les courbes d'accélération du curseur pour les systèmes Windows et MacOS. La première partie de ce chapitre présente une telle comparaison de performance, et détaille une procédure pour tester une courbe d'accélération sur un système différent de son système d'origine.

Par la suite, en deuxième partie de chapitre, et une fois la comparaison Windows/Mac effectuée, nous chercherons à construire une famille de "bonnes courbes" d'accélération, par comparaison. En effet, comme nous l'avons vu dans les chapitres précédents, le profil d'une courbe d'accélération impacte les performances. La construction d'une famille de courbes "optimales" permet donc d'offrir à l'utilisateur des performances améliorées en temps et/ou nombre d'erreurs.

3.1 Etude comparée des courbes Windows et Mac OS

Nous rappelons ici qu'un système d'exploitation, pour déplacer le curseur à l'écran matérialisant la souris peut utiliser différentes mises en correspondance. Le déplacement peut, par exemple, être constant (le curseur parcourt la même distance à l'écran que la souris sur la table). Il est également possible de proposer une relation non-linéaire entre l'espace physique et l'espace virtuel à l'écran. Un déplacement à faible vitesse sur le plan de travail va faire se déplacer le curseur d'une distance inférieure à la distance parcourue par la main. Un déplacement beaucoup plus rapide va permettre de couvrir la plus grande partie de l'écran avec un geste plus restreint que la distance qu'il faudrait réellement parcourir.

Cette mise en correspondance stockée sous forme de fonction de transfert par le système, ou *courbe d'accélération* par raccourci de langage.

Les utilisateurs qui changent pour la première fois de système d'exploitation ressentent

un fonctionnement différent de la souris sur leur nouveau système, par rapport à l'ancien système d'exploitation qu'ils utilisaient. Le plus souvent, les reproches sont fait à MacOS, avec une impression de souris plus "molle" que sous Windows (les utilisateurs réalisent une synesthésie, et transfèrent le déplacement visible du curseur à celui du comportement de la souris. La souris devient une extension de leur bras et de leur main, plus qu'outil comme décrit dans l'état de l'art).

3.1.1 Expérience

Nous voulons présenter une expérience qui compare différentes courbes d'accélération de la souris, provenant des 3 systèmes d'exploitation Windows, MacOSX et Linux. Les recherches précédentes dans la littérature ont comparé des courbes d'accélérations constantes, ainsi qu'un travail sur la courbe de Windows, non linéaire [29].

En proposant une tâche de pointage à une dimension, nous pouvons déterminer quelle courbe (ou famille de courbes) est la meilleure au regard de certains critères. Cet optimum se définit par rapport au temps nécessaire pour effectuer l'action de cliquer sur une cible (le *Temps*). Le taux d'erreurs (lorsque l'utilisateur clique hors de la cible) doit également être pris en compte. En effet, l'utilisateur qui ne resterait qu'en phase balistique du geste de pointage (geste très rapide effectué en boucle ouverte) pourrait effectivement parcourir l'intégralité de l'écran en une fraction de seconde, mais la dispersion du pointeur autour du point d'arrivée serait telle que le mouvement serait inefficace.

Les courbes de gains choisies correspondent aux courbes par défaut des systèmes Windows et MacOS, c'est-à-dire le réglage utilisé par la majorité des utilisateurs (qui ne changent pas les paramètres de leur ordinateur). Nous y avons ajouté les courbes obtenues pour chacun des deux systèmes poussant l'accélération au maximum dans le panneau de configuration. Nous y ajoutons également la courbe Constante de gain 4 (qui a obtenu de bonnes performances dans [29]), et la courbe Linux par défaut.

Hypothèse sur les performances suivant les courbes

En observant que les courbes sont différentes entre elles, nous prenons l'hypothèse qu'en retour les performances des utilisateurs seront différentes suivants les différentes courbes. Ces différences vont se manifester à travers les différentes largeurs et distances à la cible. Les courbes avec un gain élevé à grande vitesse doivent être plus rapides, car elles permettent de prendre moins de temps pour couvrir les grandes distances, et les courbes avec un gain faible à vitesse réduite doivent aider l'utilisateur à réduire son taux d'erreur quand il clique sur de petites cibles, en amortissant le geste.

Utilisateurs

12 personnes ont pris part à l'expérience, 7 hommes et 5 femmes, avec un âge moyen de 24.8 ans (SD = 2.0). Un des utilisateurs était gaucher, mais utilisait la souris de la main droite. Tous les sujets avaient une vue normale ou corrigée pour être normale, et utilisaient WindowsXP (ou Vista) et Linux et/ou MacOSX comme second système d'exploitation.

Tous travaillaient dans le domaine de l'informatique, et utilisaient quotidiennement un ordinateur.

Deux sujets utilisaient des souris dédiées aux jeux vidéos, mais personne ne jouait sur PC (ceux qui jouaient sur consoles utilisaient une manette, sans conséquence donc sur un éventuel apprentissage de la souris). Cette information est importante, car les joueurs de jeux de tirs sur PC ont tendance à désactiver l'accélération de la souris en jouant. Cela leur permet de conserver un même déplacement à l'écran lorsque les déplacements sur la table sont identiques en distance parcourue. Ainsi, la mémoire musculaire du corps peut entrer en compte et permettre à ces joueurs d'augmenter leurs performances en profitant de leurs réflexes. Il suffit d'observer sur internet les nombreuses réclamations de joueurs sur les forums des éditeurs de jeux lorsqu'un jeu de ce genre ne propose pas d'option pour désactiver l'accélération de la souris.

Au travail, un seul sujet changeait l'accélération de la souris sur son système, les autres la laissaient au réglage par défaut.

Matériel utilisé pour l'expérience

L'expérience s'est déroulée sur un PC Windows XP SP3. La souris optique est une Logitech G9, utilisée à 400 DPI. Les utilisateurs travaillent sur un écran LCD 22 pouces, avec une résolution de 1680 × 1050 à 96 DPI (1 pixel = 0.282 mm). L'application, programmée en C++/OpenGL, contourne le pilote de la souris. Elle récupère directement les informations de la souris via DirectInput, pour obtenir des données brutes, non prétraitées par le système. L'écran est rafraichi à 60 fps, et les données récupérées à 512 Hz sont sauvegardées à la fin de chaque bloc pour éviter des ralentissements dus aux accès disques. Pour mesurer le débrayage (lorsque l'utilisateur soulève la souris de la table), nous avons installé un interrupteur ultra-léger sous la souris. Cet interrupteur s'ouvre lorsque le périphérique ne touche plus la table. Le système enregistre alors l'information de débrayage.

Les utilisateurs sont installés sur un bureau dégagé, à 50 cm de l'écran. Les courbes sont stockées sous forme de table dans des fichiers XML (valeur de déplacement relatif de la souris en *dots* vs. déplacement du pointeur en pixels).

Tâche à effectuer lors de l'expérience

L'utilisateur doit accomplir une tâche de pointage réciproque, en utilisant le pointeur de la souris pour cliquer alternativement sur deux cibles verticales à l'écran. Lorsqu'une cible est effectivement cliquée, elle devient grise pour indiquer que l'autre cible doit être cliquée à son tour.

Les cibles prennent toute la hauteur de l'écran, le pointeur lui-même étant une ligne verticale de un pixel de large, ramenant la tâche à un pointage à une dimension.

Les cibles sont de trois largeurs différentes, et sont placées à trois distances différentes, de manière symétrique par rapport au centre de l'écran.

L'utilisateur doit cliquer sur une cible avant de pouvoir passer à la suivante, un bip retentit si le clic se fait hors de la cible. Les erreurs impactent dès lors le temps nécessaire pour réaliser la tache de pointage sur une cible donnée.

Ce design permet au participant de réaliser l'expérience à "vitesse normale", sans se dépêcher pour finir le plus vite possible (une situation qui entraînerait plus d'erreurs qu'une utilisation normale).

Tous les 10 clics, un taux d'erreur cumulatif est affiché, avec un message encourageant les participants à se conformer à un taux d'erreur de 4%, en ralentissant ou accélérant si nécessaire.

Le curseur de la souris peut sortir des limites de l'écran, ce qui évite d'utiliser les bords de la fenêtre pour faciliter l'acquisition de la cible. En effet, lors de l'utilisation classique d'une interface graphique, le curseur de la souris s'arrête au bords de l'écran. Ainsi, les utilisateurs expérimentés peuvent "lancer" un mouvement en restant en phase balistique, et voir le curseur arrêté par le bord de l'écran. Il leur suffit ensuite de reprendre la souris pour finir le geste plu précisément. Les cibles qui sont placées près des coins et des bords de l'écran profitent alors de ce gain de vitesse, et deviennent accessibles plus rapidement que si le geste était réalisé de manière classique (d'après la formulation de la loi de Fitts - voire 1.2.0.0, ces cibles profitent d'une largeur "infinie").

L'utilisateur doit réaliser dix répétitions pour une distance et une largeur de cible donnée, et le répéter cinq fois de suite (blocs). Cette séquence est ensuite répétée pour chacune des trois distances et des trois largeurs.

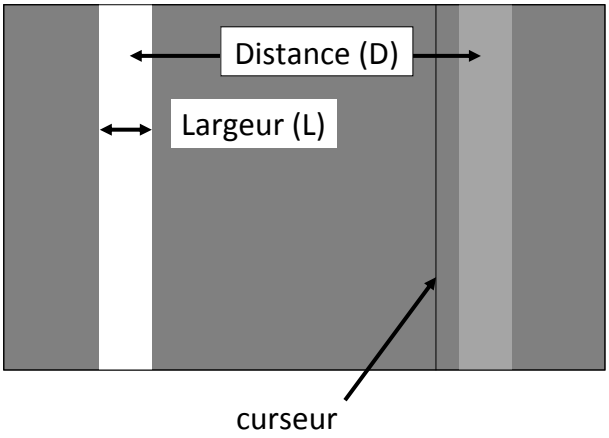

FIGURE 3.1 – Affichage de l'expérience.

La figure (3.9) présente l'affichage de l'expérience. Les cibles sont affichées comme des barres verticales pleines, équidistantes du centre de l'écran, opposée l'une à l'autre, le long de l'axe horizontal. La cible à sélectionner est en blanc (a), et la dernière cible, qui est le point du départ du mouvement, en gris clair (b). Le curseur est représenté par une ligne verticale noire d'un pixel de large (c).

Organisation de l'expérience

Les COURBES Windows et MacOS de type DÉFAUT correspondent à l'accélération d'origine du système, et les courbes MAXIMUM correspondent à l'accélération poussée au maximum sur chacun des deux systèmes dans leur panneaux de configurations respectifs.

Le design de l'expérience est intra-sujets. Les variables indépendantes sont les COURBES avec six niveaux : WINDOWS-DÉFAUT, WINDOWS-MAXIMUM, MACOS-DÉFAUT, MACOS-MAXIMUM, LINUX et CONSTANTE, la distance à la cible D (D_P = 6.25 cm, D_M = 12.5 cm, D_G = 25 cm - ou (157,315,629) en pixels sur cette configuration particulière) et la largeur L entre les deux cibles (L_P = 0.08 cm, L_M = 0.16 cm, L_G = 0.32 cm) ou (2, 4, 8) en pixels). Les combinaisons D-L donnaient neufs tâches d'indices de difficultés présentés dans le tableau suivant :

$D \setminus L$	0.08	0.16	0.32
6.25	6.30	5.33	4.14
12.5	7.29	5.54	4.19
25.0	8.29	5.72	4.23

FIGURE 3.2 – table des Indices de Difficultés - D et L sont en cm

Nous avons utilisé un carré latin pour contrebalancer l'expérience, en alternant 2 courbes Windows, 2 courbes MacOSX, 1 courbe Linux et une courbe d'accélération constante de ratio 4 :1.

Pour Windows et Mac OS X, nous avons choisi les courbes d'accélération proposées par défaut par ces 2 systèmes, ainsi que celles proposant l'accélération maximum. La courbe Linux est seule proposée par X.org sous Ubuntu.

A la fin de chaque courbe, il était demandé à l'utilisateur de noter ses préférences pour la courbe, sur une échelle de 1 (très désagréable) à 7 (très agréable). Les participants recevaient l'instruction de prendre des pauses quand ils le désiraient, pendant qu'un des messages de l'expérience était affiché à l'écran. L'expérience a été finie en 120 minutes en moyenne, en une seule session, en incluant les pauses.

Notre design expérimental est donc :

12 sujets ×
6 Courbes ×
9 combinaisons D-L ×
5 Blocs ×
10 essais
= 32 400 essais au total

Résultats de l'expérience

Si cela n'est pas spécifié autrement, nous avons utilisé Bonferroni comme correction dans la suite de l'analyse pour les ANOVA.

Taux d'Erreur

Le taux d'erreur moyen, pour tous les participants était de 4.27% (SD = 1.44%). Les analyses de mesure répétées de la variance montrent un effet significatif de la LARGEUR ($F_{2,20}$=17.445, p<0.0001) sur le Taux d'Erreur. Une comparaison paire à paire montre

des différences significatives sur toutes les LARGEURS ($p<0.05$) : 52% des ERREURS se produisant à 2 pixels et 42% à 4 pixels.

Nous observons également un effet significatif de la DISTANCE ($F_{2,20}=217.652$, $p<0.0001$), la comparaison paire à paire montre que toutes les DISTANCES sont significatives ($p<0.015$). Aucune erreur ne se produit à 6.25 cm, et 65% des erreurs à 12.5 cm. Une différence significative se produit sur BLOC×LARGEUR ($F_{8,80}=2.339$, $p<0.026$). A partir du BLOC 2, nous avons une différence entre la LARGEUR 8 pixels et les deux autres ($p<0.026$).

Pour LARGEUR×DISTANCE ($F_{4,40}=8.485$, $p<0.0001$) la DISTANCE 25.0 cm est significativement différente des deux autres sur toutes les LARGEURS ($p<0.010$).

Temps pour le Mouvement

Le TEMPS est défini comme l'intervalle nécessaire pour effectuer un geste complet pour cliquer à l'intérieur d'une cible, en partant de la cible précédente. Pour la suite de l'analyse sur le TEMPS, les essais pour lesquels l'utilisateur a cliqué hors de la cible sont supprimés.

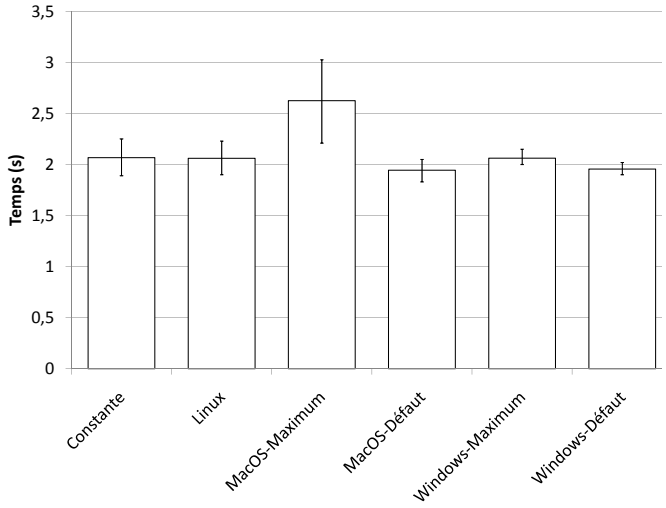

FIGURE 3.3 – Temps moyen en secondes pour sélectionner une cible suivant le type de courbe utilisée. Les moustaches représentent les intervalles de confiance à 95%.

En supprimant les *outliers* sur le TEMPS, nous avons plusieurs différences significatives. Les analyses de mesures répétées de la variance montrent un effet significatif des BLOCS ($F_{4,40}=8.511$, $p<0.0001$) sur le TEMPS : puisque les interactions significatives se déroulent entre les BLOCS 1 et 2,3,4,5 ($p<0.024$), nous avons un effet d'apprentissage. Cet effet significatif disparaît d'ailleurs lorsque l'ANOVA n'est effectuée que sur les 4 derniers BLOCS. Nous supprimons donc le premier BLOC dans la suite de l'analyse.

DISTANCE montre un effet significatif ($F_{2,20}=226.718$, $p< 0.0001$), ainsi que LARGEUR ($F_{2,20}=260.494$, $p< 0.0001$). L'analyse paire à paire montre une différence significative entre toutes les DISTANCE ($p<0.0001$). Le TEMPS passe de 1.76 s pour 6.25 cm, à 1.90 s pour 12.5 cm et à 2.2 s pour 25.0 cm.

De même une différence significative apparaît entre toutes les LARGEURS ($p<0.0001$). Le TEMPS passe de 2.36 s pour 2 pixels, à 1.93 s pour 4 pixels, et à 1.58 s pour 8 pixels.

Pour l'effet significatif de COURBES ($F_{5,50}=13.673$, $p<0.0001$), MACOS-MAXIMUM est différente de toutes les autres ($p<0.042$) (avec un temps moyen de 2.29s pour réaliser une action de pointage, alors que la moyenne de l'ensemble des autres COURBES est à 1.89s (SD = 0.06).

Nous avons également une interaction entre LARGEUR×DISTANCE ($F_{4,40}=2.994$, $p< 0.031$), avec pour chaque Largeur et Distance une différence significative avec toutes les autres ($p<0.015$).

Pour DISTANCE×COURBE ($F_{10,100}=6.013$, $p<0.0001$), nous avons une différence significative pour MACOS-MAXIMUM sur toutes les DISTANCES ($p<0.008$) : MACOS-MAXIMUM est significativement différente de toutes les autres COURBES sur les DISTANCES 6.25 cm, 12.5 cm et 25.0 cm. WINDOWS-DÉFAUT et WINDOWS-MAXIMUM sont également différentes ($p<0.048$) sur les DISTANCES 6.25 cm et 12.5 cm. De plus WINDOWS-MAXIMUM est différente d'avec LINUX ($p<0.041$). Sur l'interaction LARGEUR×COURBE ($F_{10,100}=6.169$, $p<0.0001$), MACOS-MAXIMUM est différente de toutes les autres COURBES ($p<0.039$) sur les LARGEUR 2 pixels, 4 pixels et 8 pixels. De plus, sur la LARGEUR 2 pixels, MACOS-DÉFAUT est différente de WINDOWS-DÉFAUT ($p<0.035$), alors que WINDOWS-DÉFAUT est différente de WINDOWS-MAXIMUM ($p<0.028$).Nous avons également noté une influence de LARGEUR× COURBE×DISTANCE ($F_{20,200}=3.735$, $p<0.0001$) sur le TEMPS.

Dépassement de la cible - *Overshooting*

Pendant l'expérience, nous mesurons l'OVERSHOOTING à partir du moment où le curseur dépasse la cible et sort de celle-ci. L'OVERSHOOT se mesure en pixels.

Nous avons supprimé les séries contenant des clics hors de la cible, ainsi que les outliers sur l'OVERSHOOT. La LARGEUR a une influence sur l'OVERSHOOT ($F_{2,20} = 5.283$, $p< 0.015$), entre 4 pixels et 8 pixels (45 pixels d'OVERSHOOT sur 4 pixels et 54 pixels sur 8 pixels).

Pour la DISTANCE ($F_{2,20} = 65.902$, $p< 0.0001$) toutes les DISTANCES ont une influence ($p<0.006$ - entre 25.0 cm, 90 pixels d'OVERSHOOT et 12.5 cm (57 pixels), alors qu'il n'y a aucun OVERSHOOT sur 6.25 cm). Les COURBES ($F_{5,50} = 52.014$, $p < 0.0001$) montrent également un effet significatif.

La comparaison paire à paire montre des différences significatives sur COURBES : entre CONSTANTE et toutes les autres ($p<0.004$), de même pour WINDOWS-DÉFAUT ($p<0.0119$). LINUX est différente de toutes les autres, à l'exception de MACOS-DÉFAUT ($p<0.004$). MACOS-MAXIMUM est différente de toutes les autres, à l'exception de WINDOWS-MAXIMUM ($p<0.006$). Enfin, MACOS-DÉFAUT est significativement différente de toutes à l'exception de LINUX ($p<0.019$) (voir 3.4).

Pour BLOC×WIDTH ($F_{8,80} = 3.169$, $p<0.004$), et BLOC×DISTANCE ($F_{8,80} = 4.171$, $p<0.0001$) nous avons des différences significatives. Pour la LARGEUR 2 pixels, nous avons des différences entre le BLOC 1 et les autres ($p<0.032$), ainsi qu'entre les BLOCS 1 avec 3 et 4 ($p<0.037$), et le BLOC 3 avec 5 ($p<0.041$). De même, pour la DISTANCE 25.0 cm, la différence significative est entre les BLOCS 1 et 5 ainsi que 3 et 5 ($p<0.018$).

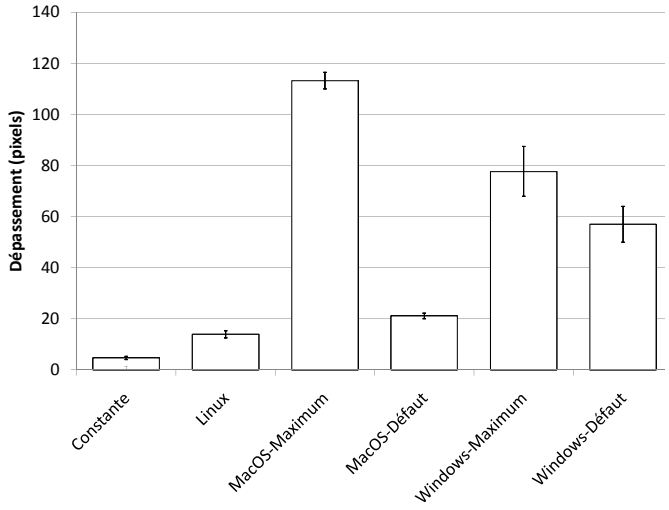

FIGURE 3.4 – Overshoot moyen en pixels pour un clic par courbe

Pour LARGEUR×COURBE ($F_{10,100} = 3.814$, $p<0.0001$), nous avons un effet significatif. De plus, l'analyse paire à paire nous montre que nous avons une différence significative : sur les 3 DISTANCES, CONSTANTE est différente de toutes les autres ($p<0.019$). LINUX est différente de toutes les autres à part MACOS-DÉFAUT sur 2 et 4 pixels, ainsi que de MACOS-DÉFAUT et WINDOWS-DÉFAUT sur 8 pixels ($p<0.019$). MACOS-MAXIMUM est différente de toutes les autres à part WINDOWS-MAXIMUM sur les DISTANCES 2 pixels et 8 pixels, ainsi que de WINDOWS-DÉFAUT et WINDOWS-MAXIMUM sur 4 pixels ($p<0.005$). MACOS-DÉFAUT est différente de toutes les autres à part LINUX sur toutes les DISTANCES, ainsi que de WINDOWS-DÉFAUT sur 8 pixels ($p<0.038$). WINDOWS-DÉFAUT est différente de toutes les autres à part MACOS-MAXIMUM sur 2 pixels ($p<0.004$), ainsi que de MACOS-DÉFAUT sur 8 pixels ($p<0.038$). Enfin WINDOWS-DÉFAUT est différente de MACOS-MAXIMUM ainsi que de WINDOWS-MAXIMUM sur 4 pixels ($p<0.019$).

Sur DISTANCE×COURBE ($F_{10,100} = 28.404$, $p<0.0001$), nous avons une différence entre toutes les COURBES pour la DISTANCE 12.5 cm ($p<0.037$). Sur la DISTANCE 25.0 cm, CONSTANTE et WINDOWS-DÉFAUT sont différentes de toutes les autres ($p<0.044$). LINUX est différente de toutes les autres à l'exception de MACOS-DÉFAUT ($p<0.024$). MACOS-MAXIMUM est différente de toutes les autres à l'exception de MACOS-DÉFAUT ($p<0.044$). MACOS-DÉFAUT est différente de toutes les autres à l'exception de LINUX ($p<0.030$). Enfin, WINDOWS-MAXIMUM est différente de toutes les autres à l'exception de MACOS-MAXIMUM ($p<0.015$). Nous avons également une différence significative entre LARGEUR×DISTANCE ($F_{40,40} = 6.853$, $p<0.0001$) BLOCK×DISTANCE×LARGEUR($F_{16,160} = 7.720$, $p<0.0001$), BLOCK×COURBE×LARGEUR ($F_{16,160} = 1.751$, $p<0.005$), DISTANCE×COURBE×LARGEUR ($F_{20,200} = 6.889$, $p< .0001$) ainsi que BLOCK×COURBE×LARGEUR×DISTANCE ($F_{80,800} = 2.285$, $p< 0.0001$).

Préférences Utilisateurs

Les préférences des utilisateurs ne concordent pas toutes avec les courbes donnant les meilleurs temps pour réaliser l'expérience. En effet, la plus mauvaise courbe (MacOSX-maximum) sur le critère du Temps a une préférence de 5.084, alors qu'une bonne courbe (MacOSX-défaut) est à 5.158 (différence de 1.45% seulement).

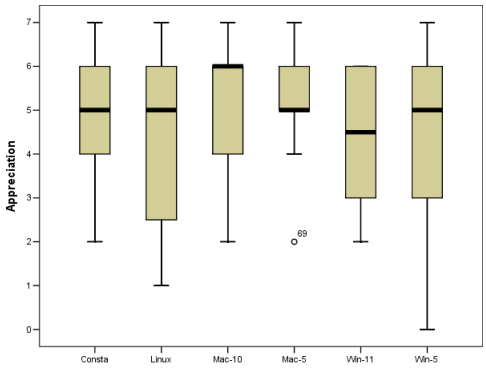

Figure 3.5 – Préférences moyennes des utilisateurs suivant les courbes

Fonction de répartition de la vitesse de la souris

En mesurant l'ensemble des vitesses parcourues par l'utilisateur, nous pouvons construire la courbe accumulée des vitesses, pour chaque fonction de gain (cf fig. 3.8). Sur cette courbe de répartition des vitesses, il apparaît de 2 groupes. Un des groupes comprend la courbe Windows-défaut et la courbe MacOS-défaut, avec une progression dans l'utilisation des vitesses élevées moins rapide que pour l'autre groupe (constitué des 4 autres courbes).

3.2 Discussion sur l'expérience Windows/Mac OS

La courbe ayant un effet significatif sur les performances en Temps, nous montrons ici que le réglage des paramètres de la souris, et donc le choix de la fonction de gain, a une influence sur les performances de l'utilisateur.

Effet de l'apprentissage

Nous pouvons observer un effet d'apprentissage lorsque la variabilité des performances, c'est-à-dire le temps nécessaire pour réaliser une tâche, ou lorsque le nombre d'erreurs effectuées lors de la réalisation de cette tâche, diminue. En comparant donc le taux d'erreurs et le temps du mouvement de blocs en blocs, nous pouvons établir la présence d'un apprentissage.

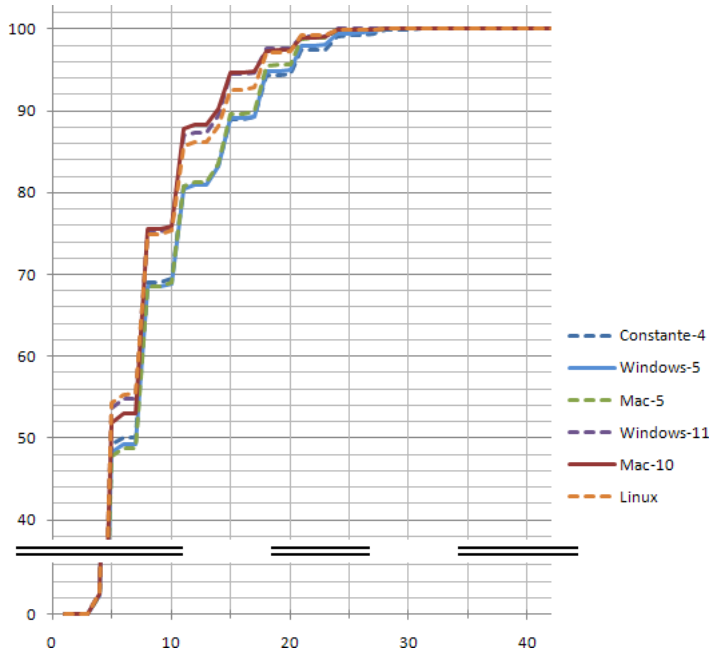

FIGURE 3.6 – Pourcentage d'accumulation des vitesses pour les différentes courbes, en cm/s

L'effet d'apprentissage s'effectuant sur un BLOC (pour le taux d'erreurs et le temps), nous en déduisons *qu'il faut au moins 10 clics pour qu'un utilisateur s'habitue à une courbe d'accélération donnée.*

Cette valeur doit être prise en compte dans le design d'un système permettant de choisir la courbe la plus adaptée à un utilisateur. Il est également à noter que le temps d'apprentissage peut être modifié lorsque l'utilisateur utilisera un autre périphérique, ou lorsque la population d'utilisateurs sera trop différente de la population moyenne étudiée ici (par exemple, des utilisateurs n'ayant jamais utilisé un ordinateur, ou des utilisateurs âgés et/ou souffrant d'un handicap moteur).

Différentes distances et largeurs

Les courbes donnent des performances (en temps, en nombre d'erreurs et en *overshoot*) différentes sur les différentes largeurs et distances ; de plus les profils de ces courbes sont différents. Nous pouvons donc en déduire que le gain, nécessairement différent à petite et grande vitesse, va avoir une influence sur la sélection des cibles de tailles et de distances variées.

La largeur et la distance, en plus d'avoir un effet sur le temps (en concordance avec la loi de Fitts[46]), ont également une influence sur le taux d'erreurs et sur la distance d'overshooting. MACOS-MAXIMUM, dont la valeur de gain maximum est beaucoup plus élevée que les autres courbes, obtient les plus mauvaises performances sur l'OVERSHOOT et le TEMPS, pour toutes les LARGEURS et toutes les DISTANCES. Puisque WINDOWS-MAXIMUM souffre de ce problème également, nous pouvons conjecturer que le gain à

vitesse élevé influence l'OVERSHOOT. En effet, le gain à vitesse élevé applique des amplitudes très importantes au curseur à l'écran. Nous pouvons conjecturer que ces grands déplacements, effectués à vitesse rapide sont plus complexes à anticiper par le système physioneurologique. De plus, étant moins souvent utilisée, cette plage de vitesse bénéficie moins de l'effet d'apprentissage, deux problèmes qui concourent à augmenter la difficulté de maîtrise du geste avec un gain élevé.

Erreurs et Overshoot

L'erreur et l'overshoot semblent être en corrélation. En effet, plus la largeur de la cible est petite, plus il y a d'erreurs et plus la quantité d'overshoot augmente. Ces erreurs et cet overshoot apparaissent au delà des 6.25 cm. La vitesse de la main semble expliquer les performances sur les erreurs et l'overshoot : l'utilisateur passe 87.8% du temps en dessous de 10 cm/sec pour la distance 6.25 cm, expliquant les performances optimales à cette distance. En effet, l'utilisateur passant la plus grande partie de son temps à faible vitesse, la planification des gestes est plus aisée, diminuant les erreurs et les clics hors de la cible.

La faible différence sur l'OVERSHOOTING entre les deux courbes Windows pourrait s'expliquer par le fait qu'il existe une partie presque similaire aux deux courbes, à faible vitesse. En effet, alors que les courbes MacOS sont différentes et ne commencent pas au même gain pour les différents niveaux d'accélération, la première partie des courbes Windows correspond à une mise à l'échelle entre les différents niveaux d'accélérations. La première partie des deux courbes Windows est similaire, et les deux courbes ne diffèrent qu'à vitesse plus élevée.

Préférences utilisateurs et performances

La courbe CONSTANTE apparaît appréciée des utilisateurs (elle est mieux notée). Or, cette courbe, par son design est plus prédictible dans son comportement dans l'espace moteur et l'espace visuel (le déplacement à l'écran sera toujours un multiple constant du déplacement du périphérique sur le plan de travail, permettant à la mémoire musculaire de la main de jouer). Par contre la courbe MACOS-MAXIMUM entraîne de mauvaises performances, mais apparaît également appréciée des utilisateurs. Les performances mesurées sur les COURBES ne sont dès lors pas corrélées aux préférences utilisateurs.

Les utilisateurs préfèrent utiliser certaines courbes plutôt que d'autres. De plus, un utilisateur nouvellement arrivé sur un nouveau système d'exploitation manifeste souvent son inconfort ou sa surprise lors de l'utilisation de la souris. Malgré tout cet inconfort disparaît avec le temps, et les nouveaux utilisateurs deviennent des utilisateurs réguliers, qui se sont adaptés au fonctionnement de leur périphérique.

Il pourrait être alors intéressant dans une future expérience d'évaluer dans quelle mesure l'impression de confort des utilisateurs (la préférence) peut pondérer le choix d'une courbe moins performante. Est-il pertinent de choisir au final la courbe préférée des utilisateurs, qui sera peut-être moins performante que la courbe optimale, alors qu'un utilisateur qui aura manipulé plus longtemps la courbe optimale s'y sera habitué après un certain temps, même si ce n'est pas la courbe qu'il préférait à l'origine ?

En d'autres termes, est-il préférable de laisser l'utilisateur choisir la courbe qu'il préfère, au risque de voir les performances ne pas être au maximum ?

Espace de définition de la courbe

Même si les systèmes d'exploitation définissent leurs courbes au-delà, les utilisateurs ne dépassent pas 40 cm/sec en vitesse avec leur main, et sont 98% du temps inférieur à 20 cm/sec.

Les courbes étant définies complètement au delà de cet "espace d'utilisation", ignorer ce fait pourrait mener l'expérimentateur à prendre du temps pour optimiser une courbe d'accélération dans une partie de l'espace vitesse qui ne serait utilisée que moins de 1% du temps. Il est donc nécessaire, avant toute étude d'optimisation, et ceci reste vrai pour le travail avec les périphériques autres que les souris, de définir l'intervalle réel de manipulation de l'appareil par les utilisateurs.

3.2.1 Limites de l'expérience

Pointage réciproque

L'expérience présentée ici est une expérience de pointage réciproque. Le mouvement est pendulaire, répété entre les 2 cibles successives. Guiard ([55]) ainsi que Fernandez *et al.*([45]) ont montré qu'une expérience de pointage réciproque ne peut se résumer à l'adjonction de gestes de pointages discrets : lorsque l'utilisateur arrivait sur la cible, il savait qu'il devait repartir sur la cible suivante, et ajustait son geste en conséquence. L'utilisation d'une GUI au quotidien implique des mouvements discontinus, avec des pauses de durée aléatoire. Il peut alors être intéressant de réaliser une expérience de pointage discret (qui implique alors d'obtenir moins de données, par rapport à l'expérience de pointage réciproque).

Fatigue musculaire

Nous n'avons pas quantifié un facteur lors de l'évaluation : la fatigue musculaire. Nous avons pu constater lors de l'expérience que les utilisateurs contractent et crispent les muscles plus ou moins fort sur la souris suivant la facilité d'utilisation de la courbe d'accélération. Pouvoir mesurer cette tension musculaire (à l'aide d'un EMG sur le poignet et/ou l'avant-bras, par exemple, ou d'un capteur de pression sur la souris) permettrait d'évaluer l'influence de ce facteur sur les performances de pointage.

Fréquence de fonctionnement USB

Afin de s'assurer que les courbes ramenées par notre dispositifs étaient valables, nous nous sommes assurés qu'un pointeur virtuel, dont le déplacement était calculé par nos courbes d'accélérations, se superposait bien au curseur système lorsque nous déplacions la souris. Les pointeurs ne sont superposés que si notre calcul sur leur pointeur virtuel se déroule à 512 Hz, et non à 125 Hz comme pour le pointeur système. C'est donc cette

valeur de 512 Hz qui a été conservé dans l'expérience, afin que les utilisateurs retrouvent un comportement du pointeur identique à celui qu'ils utilisent sur leurs sytèmes respectifs.

3.3 Détermination d'une famille de courbes optimales

Bien que les systèmes d'exploitation actuels fournissent chacun leur propre interprétation et implémentation d'une "bonne" courbe d'accélération du curseur de la souris, aucune étude n'a été réalisée sur le profil que devrait avoir une courbe d'accélération voulant optimiser les performances. Il a été montré précédemment dans notre travail que le profil particulier d'une courbe d'accélération avait un impact notable sur les performances. Il y a donc un intérêt à proposer une courbe ou une famille de courbes d'accélérations "optimales", ou au moins de proposer une série d'heuristiques afin de construire cette famille de courbes.

La présente partie de chapitre propose un protocole pour établir une étude de ce type, et dénombrer les heuristiques à mesurer dans ce genre d'étude.

Nous allons tout d'abord présenter la construction théorique du profil d'une courbe d'accélération. Ensuite nous listerons les hypothèses qui font d'une courbe d'accélération une "bonne" courbe. Enfin nous détaillerons une expérience que nous avons mener afin d'illustrer le protocole de construire d'une famille de courbes performantes.

3.3.1 Profil d'une courbe d'accélération

Comme expliqué précédemment, le profil optimal d'une courbe d'accélération semble se rapprocher d'une courbe en S. Cette sigmoïde dispose d'une première partie qui croit lentement, d'une partie centrale qui monte rapidement, et d'une troisième partie qui croit lentement à nouveau.

Ce type de courbe se définit à l'aide de 4 points de contrôle.

Points de contrôle

Le premier point de contrôle, appelé G_1, détermine la hauteur du gain minimal, c'est à dire la valeur de gain à faible vitesse. Le point de contrôle G_2 détermine la valeur de gain maximal, c'est à dire la hauteur de gain à vitesse maximale.

Les deux points V_1 et V_2 servent à marquer la pente de la droite qui va relier ces deux valeurs extrêmes de gain. Par construction, G_1->V_1 est parallèle a l'axe des abscisses, tout comme le segment V_2->G_2 (voir Figure 3.7).

Utilité de chaque segment de la courbe d'accélération

En considérant le profil de la courbe d'accélération, nous pouvons déterminer l'utilité de chaque segment de cette courbe.

— *Gain minimum*
A faible vitesse, la première partie de la courbe, de gain inférieur à 1, permet de sélectionner et de cliquer sur chacun des pixels à l'écran.

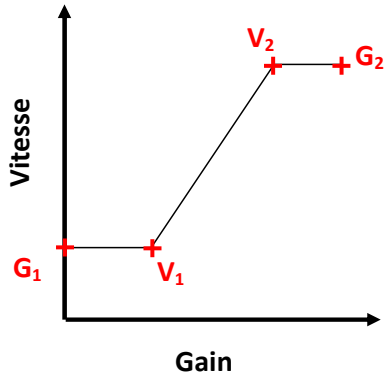

FIGURE 3.7 – Profil de courbe en S, avec les 4 points de contrôle, G_1, gain min., G_2 gain max., V_1 et V_2.

Si la valeur de gain est trop faible, et qu'elle reste faible sur une trop grande plage de vitesse, l'utilisateur a l'impression d'une souris qui "colle", car il doit faire des gestes plus vifs pour déplacer le curseur. Si la valeur de gain est trop élevée, nous pouvons nous retrouver avec une configuration qui empêche de sélectionner tous les pixels présents à l'écran (voir [29]). Cette valeur de gain minimale trop élevée peut également entraîner des dépassements de cibles à petite vitesse (c'est à dire le même problème que pour un gain trop faible), lorsque l'utilisateur va brusquement accélérer pour compenser l'effet "collant".

— *Gain maximum*
La valeur de gain maximum détermine si l'utilisateur devra débrayer la souris. Si la valeur de gain maximum est trop faible, l'utilisateur devra soulever la souris lorsqu'il arrivera au bout de son espace de travail pour la recentrer (le *débrayage*). Par contre, avec une valeur de gain trop élevée, ou qui arrive trop vite (c'est à dire qui est présent trop tôt dans la plage des vitesses de la main), l'utilisateur ne pourra pas contrôler le curseur qui va "sauter" à travers l'écran. Dès lors, l'utilisateur risque de rester dans une plage de vitesse peu élevée pour contrôler le curseur, au dépend des performances.

Après avoir regardé la construction du profil d'une courbe d'accélération, nous pouvons regarder ce qui fait une "bonne" courbe.

3.3.2 Qu'est-ce qu'une bonne courbe ?

Pour l'utilisateur final, une bonne courbe d'accélération permet d'obtenir le meilleur compromis entre le temps nécessaire pour pointer et le nombre d'erreurs (les clics hors de la cible, et dans une moindre mesure, les dépassements de cible - en effet, il est toujours possible de revenir en arrière sur la cible lorsque l'utilisateur l'a dépassée, mais un clic malencontreux peut activer une autre cible, et changer l'état de la tâche en cours, sans forcément pouvoir annuler l'action). Il vaut donc mieux diminuer le nombre d'erreurs, plutôt que d'accélérer les déplacements, car nous ne savons pas à priori si les actions erronées peuvent être annulées par l'utilisateur.

Afin de déterminer la meilleure performance sur une variable donnée, nous pouvons utiliser une mesure de performance en U. En comparant la performance d'une variable prenant plusieurs valeurs, si la performance pour la valeur centrale est meilleure que les valeurs qui l'entoure, nous disposons alors d'un point d'inflexion de cette performance. C'est alors un problème de minimisation.

En prenant par exemple comme critère de performance le temps (*MT - Mouvement Time*) nécessaire pour acquérir une cible, si la valeur centrale de notre variable testée donne un *MT* plus faible que pour les valeurs qui l'entoure, nous avons alors un minimum local, d'où le nom de performance *en U*.

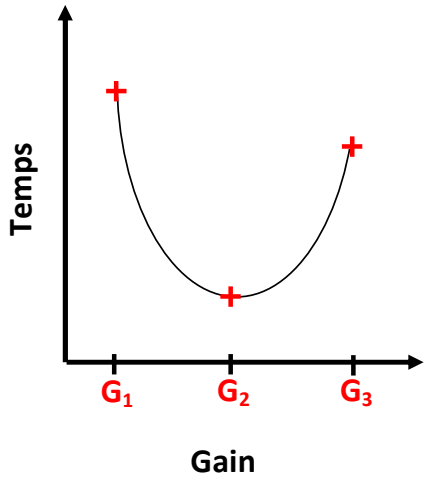

FIGURE 3.8 – Exemple de graphique de performance en U.

La figure 3.8 présente un exemple de graphique de performance en U. Pour trois valeurs de gains différentes, le temps moyen pour réaliser une tâche est différent. Dans cet exemple fictif, la valeur de Gain G_2 donne des performances optimales au regard du temps.

Pour construire un graphique de performance en U, il faut alors que la variable considérée pour la mesure de performance prenne au minimum 3 valeurs.

Pour construire notre courbe en S, nous pouvons faire varier les valeurs prises par les 4 points de contrôle. Chacun de ces points de contrôle peut prendre une valeur constituée d'un couple : vitesse de la main en abscisse et une valeur de gain en ordonnée.

Afin d'obtenir ces graphes de performances en U, nous présentons dans la suite de ce chapitre une expérience permettant de tester de multiples combinaisons de valeurs pour les différents points d'inflexion d'une courbe d'accélération.

3.3.3 Expérience

Hypothèse sur le profil des courbes

La courbe d'accélération, ou la famille de courbes d'accélération donnant les meilleures performances est une courbe en S, déterminée par 4 points de contrôle. Afin de déterminer ces 4 valeurs de contrôles, nous réalisons une expérience permettant de choisir les valeurs

de ces points dans un espace de solutions. L'expérience permet également de vérifier si une courbe composée des points de contrôles donnant chacun les meilleures performances est également la courbe donnant les meilleures performances.

Utilisateurs choisis pour l'expérience

L'expérience s'est déroulée avec 8 personnes, 5 hommes et 3 femmes.Tous les sujets avaient une vue normale ou corrigée pour être normale, et utilisaient WindowsXP (ou Vista, pas de Windows 7 [5]) et Linux et/ou MacOSX comme second système d'exploitation. Deux des utilisateurs ne travaillaient pas dans le domaine de l'informatique, les autres utilisaient quotidiennement un ordinateur. Personne ne jouait sur PC (ceux qui jouaient sur consoles utilisaient une manette. Pour eux, il n'y a alors pas de biais introduit par l'habitude d'utiliser intensément une souris non accélérée pour ce type de jeu, comme expliqué au chapitre précédent). Les sujets laissaient l'accélération de la souris sur leur système par défaut.

Matériel utilisé pour l'expérience

L'expérience s'est déroulée sur un PC Windows XP SP3. La souris optique est une Logitech G9, utilisée à 400DPI. Les utilisateurs travaillent sur un écran LCD 22 pouces, avec une résolution de 1680 x 1050 à 96 DPI (1 pixel = 0.282 mm).

L'application, programmée en C++/OpenGL, contourne le pilote de la souris. Elle récupère directement les informations de la souris via DirectInput, pour obtenir des données brutes, non prétraitées par le système. L'écran est rafraîchi à 60 fps, et les données récupérées à 512Hz sont sauvegardées à la fin de chaque bloc pour éviter des ralentissements dus aux accès disques.

Pour mesurer le débrayage (lorsque l'utilisateur soulève la souris de la table), nous avons installé un interrupteur ultra-léger sous la souris qui s'ouvre lorsque le périphérique ne touche plus la table. Les utilisateurs sont installés sur un bureau dégagé, à 50 cm de l'écran.

Les courbes sont stockées sous forme de table dans des fichiers XML (valeur de déplacement relatif de la souris en *dots* vs. déplacement du pointeur en pixels).

Tâche à réaliser pendant l'expérience

L'utilisateur doit accomplir une tâche de pointage réciproque, en utilisant le pointeur de la souris pour cliquer alternativement deux cibles verticales, sur l'écran. Lorsqu'une cible est effectivement cliquée, elle devient grise pour indiquer que l'autre cible doit être cliquée à son tour.

Les cibles prennent toute la hauteur de l'écran, le pointeur lui-même étant une ligne verticale de un pixel de large, ramenant la tâche à une dimension, puisque la seule variation possible de déplacement se fait sur l'axe horizontal. Les cibles sont de trois largeurs différentes, toutes placées à la même distance, de manière symétrique par rapport au

5. Casiez a pu déterminer, avec Mark Cranness, que l'implémentation par Windows 7 de l'accélération de la souris était différente des systèmes précédents *http ://donewmouseaccel.blogspot.com/*

centre de l'écran. Afin de ne pas prolonger l'expérience au delà d'un temps raisonnable pour l'utilisateur, nous n'avons pas utilisé plus d'un seule distance à la cible, en prenant l'hypothèse qu'une courbe apportant un gain de performances le fera sur les cibles avec le plus grand indice de difficulté, c'est à dire à la plus grande distance.

L'utilisateur doit cliquer sur une cible avant de pouvoir passer à la suivante, un bip sonore retentit si le clic se fait hors de la cible. Les erreurs impactent dès lors le temps nécessaire pour réaliser la tache de pointage sur une cible donnée.

Un taux d'erreur cumulatif est affiché tous les 10 clics. Un message encourage les participants à garder à un taux d'erreur de 4%, en ralentissant ou accélérant si nécessaire.

Ce dispositif permet de ne pas pousser l'utilisateur à finir l'expérience le plus vite possible, et de conserver, en conséquence, un taux d'erreur proche d'une utilisation "normale" de la souris.

Le curseur de la souris peut sortir des limites de l'écran, ce qui évite d'utiliser les bords de la fenêtre pour faciliter l'acquisition de la cible, comme expliqué précédemment.

L'utilisateur doit réaliser dix répétitions pour une distance et une largeur de cible donnée, et le répéter cinq fois de suite (blocs). Cette séquence est ensuite répétée pour chacune des trois distances et des trois largeurs.

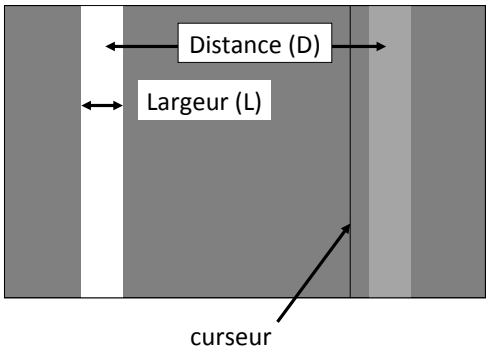

FIGURE 3.9 – Affichage de l'expérience à l'écran.

La figure 3.9 nous présente l'expérience telle qu'elle est affichée à l'écran. Les cibles sont affichées comme des barres verticales pleines, équidistantes du centre de l'écran, opposée l'une à l'autre, le long de l'axe horizontal. La cible à sélectionner est en blanc (a), et la dernière cible, qui est le point du départ du mouvement, en gris clair (b). Le curseur est représenté par une ligne verticale noire d'un pixel de large (c).

Conception du protocole

Nous construisons les différents courbes à tester à l'aide des 4 points de contrôle. Ces points de contrôles avaient été choisis à l'aide de deux critères : en premier lieux les valeurs correspondantes à de bonnes performances lors des pré-expériences, et qui permettent ensuite d'encadrer la courbe Windows par d'autres courbes.

Le point G_1 prend successivement la valeur 0 comme abscisse et (0.5,1,3,6) en ordonnée.

Le point V_1 prend successivement les valeurs (0,0.5,1) en abscisse, avec l'ordonnée égale à celle de G_1.

Le point V_2 prend successivement les valeurs (0.1,0.2,0.3) comme abscisse, avec l'ordonnée égale à celle de G_2.

Le point G_2 prend successivement les valeurs 1 comme abscisse et (6,12,24) en ordonnée.

Nous disposons dès lors de 108 courbes d'accélération.

Pour mesurer la fatigue de l'utilisateur, nous insérons une courbe de contrôle, toujours la même, toutes les 10 courbes. L'évolution de la mesure du *Mouvement Time* sur cette courbe illustre la fatigue éventuelle de l'utilisateur, avec une augmentation du MT traduisant une fatigue.

La courbe de contrôle est la courbe Windows par défaut, sans accélération.

Le design de l'expérience est intra-sujets. Les variables indépendantes sont les COURBES avec 119 niveaux : G_1,V_1,V_2,G_2, y compris la courbe de contrôle CONTROL.

Dans l'ensemble des courbes disponibles, les disparités de profils sont suffisantes pour complètement déstabiliser un utilisateur si le passage d'une courbe à une autre se fait au hasard. L'utilisateur n'arrive alors pas à s'habituer à la nouvelle courbe. Nous avons donc décider de passer d'une courbe à une autre en faisant varier la valeur d'un des points de contrôler à la fois, dans l'ordre G_2, puis V_2 puis V_1 puis G_1, afin de permettre aux utilisateurs de contrôler la souris plus facilement.

Afin de contrebalancer l'expérience, les utilisateurs ne testaient pas les courbes dans le même ordre (si un utilisateur commençait avec la courbe 0, le suivant commençait avec la courbe 119 et ainsi de suite).

Dans cet ensemble de courbes, certaines courbes sont "en escalier" (similaire aux systèmes Unix), par construction lors du choix des valeurs des points de contrôles.

Le design expérimental est alors :

10 sujets ×
119 Courbes ×
1 combinaisons D-L ×
5 Blocs ×
10 essais
= 59 500 essais au total

A la fin de chaque courbe, nous demandions à l'utilisateur de noter ses préférences pour la courbe, sur une échelle de 1 (très désagréable) à 7 (très agréable). Les participants recevaient l'instruction de prendre des pauses quand ils le désiraient, pendant qu'un des messages de l'expérience était affiché à l'écran.

L'expérience est longue, et s'est déroulée sur deux jours pour certains participants. Nous nous sommes alors assurés que les conditions de passage de l'expérience restent identiques.

3.4 Résultats et discussion sur l'expérience

Si cela n'est pas spécifié autrement, nous avons utilisé Bonferroni comme correction dans la suite de l'analyse pour les ANOVA.

Dans l'ensemble des résultats présentés dans la suite de notre travail, nous retirons les essais comprenant des erreurs (c'est-à-dire des clics en dehors de la cible). De même, le

premier essai de chaque série de 10 clics est supprimé, comme expliqué aux sujets, pour annuler l'effet de "redécouverte" de l'accélération de la souris après chaque pause due à l'affichage d'un message.

3.4.1 Résultats

P1-P2-P3-P4 - *Performance en U*

Sur le TEMPS, le meilleur choix pour P1 est 3,00 (1,65% meilleur que les autres en moyenne). Pour P2, 0 ou 0,05 sont similaires (1,15% meilleurs que P2=1), alors qu'il n'y a pas de choix discernable pour P3 et P4.

Les courbes avec $P1 = (0, 3)$, et $P2 = (0, 3)$ ou $P2 = (0, 0, 05)$ sont les courbes comprises entre la courbe n°54 et la courbe numéro 71.

La différence de la moyenne entre ces "bonnes" courbes et les "mauvaises" courbes n'est que de 0,44% sur le temps, inférieure à 0,01% sur le taux d'erreurs. Le dépassement de cible est 9,47% plus élevé pour les "bonnes" courbes que pour les "mauvaises".

De plus, l'utilisateur passe entre 78,9% et 81,9% du temps du mouvement entre 0 et 0,1 m/sec. Dès lors, le choix de P1 et P2 détermine plus que le choix de P3 et P4 les résultats du geste de pointage.

Temps moyen pour le clic

Nous supprimons les *outliers* sur le TEMPS, c'est-à-dire les essais pour lesquels le temps nécessaire pour réaliser un geste de pointage est supérieur à +/- 3 fois la déviation standard.

Notre ANOVA sur le temps nous indique que la COURBE ou le BLOC n'a pas d'effet significatif sur le TEMPS, alors qu'il y en a un pour BLOC×COURBE ($F_{428,2996} = 1,156$, $p<0,022$).

Au vu de ces résultats, et en prenant en compte les informations de la littérature, déjà présentées dans l'état de l'art, nous séparons l'expérience en deux catégories de gestes de pointage, c'est à dire l'ensemble des gestes de pointage à gauche et l'ensemble des gestes de pointage à droite.

Dès lors, et en conservant les mêmes exclusions d'essais présentées plus haut, la COURBE a une influence sur le temps de pointage à gauche ($F_{107,535} = 1,306$, $p<0,032$), de même pour BLOC×COURBE ($F_{428,2996} = 1,268$, $p<0,002$).

Dépassement de la cible

Pour l'analyse sur le dépassement de la cible, nous conservons les courbes qui ne sont pas des courbes de contrôle. De même, nous supprimons les *outliers*, c'est à dire les courbes dont le dépassement de cible est supérieur à +/- 3 fois la déviation standard.

Le choix de la COURBE a un effet significatif sur la distance de dépassement de la cible ($F_{107,749} = 4,317$, $p<0,0001$). De même le BLOC a une influence sur ce dépassement ($F_{4,28} = 11,029$, $p<0,0001$), tout comme BLOC×COURBE ($F_{428,2996} = 1,591$, $p<0,0001$).

La différence significative sur BLOC se porte entre le BLOC 1 et le BLOC 3 ($p<0,006$) ainsi qu'entre le BLOC 1 et le BLOC 5 ($p<0.05$), et entre le BLOC 2 et BLOC 3 ($p<0,042$) ainsi qu'entre le BLOC 2 et le BLOC 5 ($p<0,0001$)

En se basant sur les résultats de l'expérience du chapitre précédent (un BLOC est nécessaire pour l'apprentissage), nous avons retiré le premier BLOC des résultats. Nous avons toujours un effet significatif sur COURBE et BLOC. Lorsque nous retirons les deux premiers BLOCS, tous les effets significatifs disparaissent.

Pour les deux premiers BLOCS, le pourcentage du dépassement de cible, en distance totale du déplacement à effectuer, est de 3,15%, et de 2,73% pour les trois derniers BLOCS (une différence de 13,51%).

De manière similaire à l'analyse réalisée pour le temps, nous séparons les gestes effectués vers la gauche des gestes effectués vers la droite.

En ne considérant que les gestes de pointage vers la gauche, le choix de la COURBE influence la distance de dépassement de la cible ($F_{107,535} = 2,478$, $p<0,0001$).

Pour les gestes de pointage vers la droite, le choix de la COURBE influence également la distance de dépassement de la cible, mais dans d'autres proportions ($F_{107,642} = 2,626$, $p<0,0001$).

Préférences Utilisateurs

Les préférences accordées par les utilisateurs aux différentes courbes ne correspondent pas aux courbes donnant le plus faible nombre d'erreurs, ni même avec le temps de pointage le plus rapide.

En discutant avec ces utilisateurs, nous en avons déduit que la préférence se porte de manière subjective sur un "ressenti" de la courbe d'accélération, suivant si la courbe donne un impression d'être "trop lent" ou au contraire "trop rapide". Ce comportement est déterminé par les profils de courbe, en interaction avec la vitesse de la main de l'utilisateur. Comme la vitesse de déplacement de la main, ainsi que la plage dans laquelle un mouvement est considéré comme "confortable", dépend de chaque utilisateur, il peut être intéressant de construire un protocole expérimental demandant à chaque utilisateur les plages de vitesse dans lesquelles ils considèrent une courbe comme "trop lente" ou "trop rapide".

3.4.2 Discussion sur l'expérience

Effet d'apprentissage

Les résultats sur les BLOCS concordent avec les résultats de l'expérience présentée dans le chapitre précédent, avec au minimum 10 clics pour s'habituer à une courbe (20 clics dans cette expérience).

Malgré tout, l'expérience portait sur des courbes similaires en profils, ne comportant pas de valeurs très différentes entre elles, alors que certaines des courbes présentées ici sont très *désagréables* pour l'utilisateur (comme le montrent les préférences) avec des points de contrôles pouvant être très éloignés d'une courbe à l'autre.

Nous pouvons déduire des résultats de cette expérience que les courbes présentées ici, plus éloignées dans leur profil des courbes déjà connues des systèmes d'exploitations actuels nécessitent plus de temps pour leur apprentissage (ici 2 Blocs soit 20 clics, au lieu des 10 clics de l'expérience précédente).

Axes et directions du mouvement

Dans l'état de l'art, nous avons vu que les membres supérieurs humains n'ont pas les mêmes facilités de mouvement suivant les axes et angles considérés pour un déplacement.

Bien que l'expérience demande aux sujets d'effectuer un mouvement pendulaire, l'analyse montre que le choix de la direction du mouvement a une influence sur les performances de pointage (et nous pouvons émettre l'hypothèse qu'une différence se manifestera aussi des mouvements vers l'avant et l'arrière). Il peut être intéressant à l'avenir de concevoir une expérience afin de mesurer les performances suivant chaque quadrant dans lequel le bras se déplace, et d'implémenter une courbe différente pour chaque direction.

Dépassement de la cible

Le choix de la courbe ne semblant pas influencer les performances en temps, mais influence les performances sur le dépassement de la cible, nous pouvons supposer que lorsque l'utilisateur dépasse une cible, il effectue alors des gestes rapides pour compenser ce dépassement et revenir sur la cible. Ces gestes rapides rapprocheraient alors le temps de pointage de celui d'un pointage plus lent, mais ne dépassant pas la cible. Nous n'aurions alors pas de différence significative sur le temps, mais bien sur le dépassement de cible.

Profil du début de courbe

L'utilisateur passant près de 80% du temps d'un geste de pointage à moins de 10 centimètre par seconde, le choix du profil du début de courbe a beaucoup plus d'influence sur les performances que la fin de la courbe. Comme montré dans le précédent chapitre, l'amélioration des performances passe donc par une optimisation qui doit se focaliser sur cette première plage de vitesse.

3.5 Conclusion

L'analyse prouve que le choix de la courbe d'accélération influe sur les performances de l'utilisateur. Le profil de la courbe de gain aux petites et grandes vitesses a une influence également sur les performances.

L'étude des résultats de l'expérience de comparaison Windows/Mac OS permet déjà de donner quelques heuristiques quand au design d'une courbe de gain donnant de bonnes performances, présentée en discussion de l'expérience.

La première observation sur la deuxième expérience, qui complète l'expérience précédente, est l'importance des courbes d'accélérations dans les systèmes d'exploitations actuels. Lorsque les courbes utilisées dans les expériences sont relativement proches des

courbes utilisées par Windows ou MacOS, le temps nécessaire à l'apprentissage de la nouvelle courbe reste faible. Lorsque la courbe testée s'éloigne fortement des courbes systèmes, le temps d'apprentissage augmente rapidement. La population testée (des informaticiens en grande majorité) montre donc un effet d'habituation à l'utilisation de ces systèmes.

A l'aide de l'expérience présentée ici, nous pouvons conclure qu'une bonne courbe d'accélération ne peut se résumer à 4 points de contrôles. L'interaction entre ces 4 points de contrôles ne nous permet de réaliser une analyse en prenant isolément chacun de ces points afin de construire une "bonne" courbe.

Dès lors, nous pouvons nous poser la question de la nécessaire complexité du profil de ces familles de courbes. Un profil de courbe "en S" peut être trop réducteur pour définir la famille de courbe donnant des performances optimales.

L'utilisateur passant plus des 3/4 du temps du geste de pointage à moins de 10 cm/sec, il pourrait être intéressant d'apporter une modification à la première partie du "S", et d'obtenir ainsi une courbe plus complexe. Nous pourrions imaginer une courbe présentant une série de "marches d'escaliers" à la première partie de courbe.

Les systèmes d'exploitation actuels traitant indifféremment les différentes directions du mouvement, l'éventuel gain de performance apporté par l'implémentation de courbes différentes suivant la direction du mouvement mériterait d'être analysé. Il pourrait être intéressant de proposer un système anisotrope de courbes d'accélération, qui compenserait la perte de vitesse et de performances lorsque l'utilisateur éloigne la main et le bras du corps, sur l'axe gauche-droite, ainsi que de l'arrière vers l'avant.

Une proposition que nous n'avons pas eu le temps d'expérimenter, mais qui est apparue vraisemblable au vu des résultats de l'expérience, serait d'utiliser un système d'hystérésis pour la courbe d'accélération. L'idée serait de tester si l'utiliser d'une courbe d'accélération en période d'accélération différente de la période de décélération permet d'améliorer la fin du geste, lors du pointage, ou si la phase balistique est effectuée trop rapidement pour que l'utilisation de courbes différentes n'impacte réellement les performances.

Chapitre 4

Utilisation de la dynamique du geste pour le clic sans bouton

La dynamique de l'interaction d'un geste de pointage à l'aide d'une souris a été traitée dans les chapitres précédents. Nous allons étudier ici une utilisation différente de cette cinématique. Alors que nous nous intéressions avant à la possibilité d'accélérer le geste de pointage et sa précision, nous nous intéressons ici à la possibilité de réaliser une action discrète, telle qu'un clic, sans l'utilisation d'un bouton.

Ce chapitre est articulé en deux parties. La première partie présente le travail d'Ali Choumane, Géry Casiez et Laurent Grisoni sur un article publié à IEEE VR 2010 [34]. Le travail initialement publié ne comporte pas d'étude pour les petites cibles. Nous présentons dans la deuxième partie du chapitre l'extension du travail initial à cette configuration. Enfin, la troisième partie étudie la possibilité d'utiliser cette technique, développée en 3 dimensions, à un problème en 2 dimensions.

4.1 Travaux initiaux

La sélection et le positionnement sont deux tâches de base pour les applications 3D qui peuvent être réalisées en utilisant des boutons, des gestes [20], ou la reconnaissance vocale [16]. L'utilisation la plus commune d'un gant pour la sélection et le positionnement se fait à travers d'une métaphore telle que *pincer-pour-saisir* [17]. Cette technique n'est pas utilisable avec n'importe quel dispositif de suivi 3D, la plupart ne disposant que de trois ou six degrés-de-liberté (DOF). D'autres dispositifs tels que le Gametrak^{TM} ou l'EyeToy^{TM} n'ont aucun bouton, nécessitant l'utilisation de gestes spécifiques ou de la reconnaissance vocale pour sélectionner des options en 3D ou pour déplacer des objets. Les utilisateurs sont également réticents à lancer une procédure de calibration ainsi qu'à suivre les étapes d'apprentissage associées avec les gestes et la reconnaissance vocale.

Choumane *et al.* présentent une nouvelle approche basée sur l'analyse de la trajectoire et de la cinématique des gestes. Ils établissent que la sélection, ainsi que le *glisser-lâcher* peuvent être prédits à partir de la trajectoire et des schémas cinématiques de la main de l'utilisateur. La technique qu'ils proposent ne requiert aucun entraînement ou calibrage, et peut être utilisée sans périphérique d'interaction 3D. Le positionnement commence avec

un objet qui est sélectionné et s'arrête avec son lâcher. Ils proposent alors un algorithme basé sur les invariants trouvés dans leur étude initiale, pour remplacer l'utilisation des boutons pour la sélection et le glisser-déposer. Après leur présentation des résultats d'une expérience réalisée par eux-même pour évaluer leur algorithme, nous montrons ensuite comment améliorer sa robustesse pour les cibles de petite taille.

4.1.1 Travaux relatifs

La communauté IHM a utilisé à son avantage les travaux effectués sur le mouvement humain et les gestes en neuroscience. Par exemple, la loi de Fitts [47], qui met en relation le temps nécessaire pour réaliser un geste de pointage avec la taille et la distance à la cible est à la base de nombreux travaux (voir 1.2.0.0). La *Minimum Jerk Law* [58] (voir 1.2.0.0) et la *2/3 Power Law* [103] (voir 1.2.0.0) sont également utilisées communément lors de l'analyse de gestes non contraints. Depuis quelques années, la communauté IHM s'est intéressée à l'utilisation de l'analyse gestuelle pour proposer de nouvelles techniques d'interaction. Le système Shapewriter [76] introduit une manière efficace d'analyser les mouvements d'un stylet tracé sur un clavier. La sélection des *traits* est réalisée en identifiant les changements de mouvement significatifs dans les mouvements du stylet. Bien que dédié à une tâche spécifique (la sélection des *traits* sur un clavier affiché à l'écran), ce travail peut être vu comme une étape majeure vers l'analyse continue des gestes pour la simplification des interactions homme-machine. Lank et al. [80] proposent une méthode, basée sur l'analyse des mouvements, pour estimer la fin de la trajectoire. D'autres travaux [81] utilisent l'analyse des gestes pour l'estimation précise de la sélection, à partir d'un geste de sélection grossier.

Proposer une fonctionnalité de clic sans bouton est un problème complexe. Certaines approches essaient de fournir une interface qui ne nécessite aucun clic, alors qu'une interface classique en utilise habituellement. Par exemple, dans un environnement 2D, le projet *dontclick*[6] permet de naviguer un site internet sans utiliser de bouton de la souris. Le principe du site est que lorsque le curseur est sur un item, celui-ci est sélectionné. D'autres travaux utilisent la détection de contexte, ou certaines caractéristiques spécifiques, afin de rendre l'interaction non-ambiguë. Le système InkScribe [97] propose un mécanisme d'interaction utilisant le contexte des actions, afin d'inférer l'intention de l'utilisateur la plus plausible, à partir de l'action. Do [40] proposait une approche similaire, dans le contexte du design architectural. Quelques autres travaux proposent de réaliser des gestes avec le doigt pour fournir la fonctionnalité du clic. En utilisant le pointage à main libre, en suivant la main et les doigts, Vogel a développé deux techniques de clic pour les affichages très larges, à haute résolution[104]. La première technique, appelée *AirTap*, utilise un mouvement haut-bas de l'index pour signaler les clics. Les feedbacks auditifs et visuels sont substitués au feedback kinesthésique. Le doigt baissé permet de faire glisser les objets, et le clic est activé quand le doigt se relève. La seconde technique, appelée *Thumb-Trigger Clicking*, fonctionne de manière similaire en utilisant le pouce à la place de l'index. Eisenstein et al. ont comparé deux techniques de sélection basée sur la vision : la détection de mouvements, où l'utilisateur bouge la main au-dessus de la cible pour la sélectionner, et le suivi d'objet, où la cible sous l'objet suivi est automatiquement sélectionnée [41]. Aucun des travaux mentionnés précédemment ne propose de méthode pour combiner la

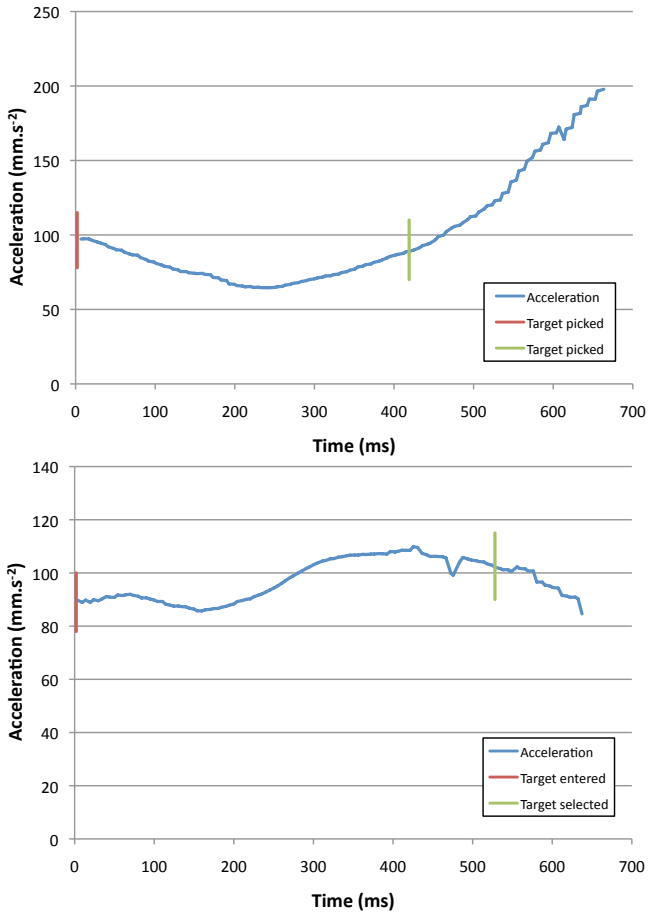

FIGURE 4.1 – Profil d'accélération pour les tâches de picking (en haut) et de sélection (en bas)quand le curseur entre dans la cible.

sélection et le glisser-lâcher sans utiliser de bouton, au sein de la même application, ce qui est le cœur de la contribution présentée ici. La section suivante présente l'étude initiale réalisée par Choumane *et al.*, et qui est à l'origine de l'algorithme proposé.

Étude initiale

Le but de cette étude initiale est d'étudier la trajectoire et les schémas cinématiques associés aux tâches de sélection et de glisser-lâcher quand elles sont effectuées avec un bouton. Il a été demandé à deux participants de sélectionner, et de glisser-lâcher des sphères positionnées de manière aléatoire dans un espace 3D. La main était suivie en utilisant un périphérique Gametrak et portait une souris sans-fil, pour réaliser la tâche. Le Gametrak a un espace de travail de 3 mètres cube, avec une résolution allant de 0,01 mm à 7 mm dans toutes les directions, et un taux d'échantillonnage de 125 Hz. Un total de 160 essais ont été collectés pour l'analyse. Pour chaque condition (sélection, glisser, lâcher), ont été calculés les profils de vitesse et d'accélération moyens. Cela donne un total

6. http ://www.dontclick.it/

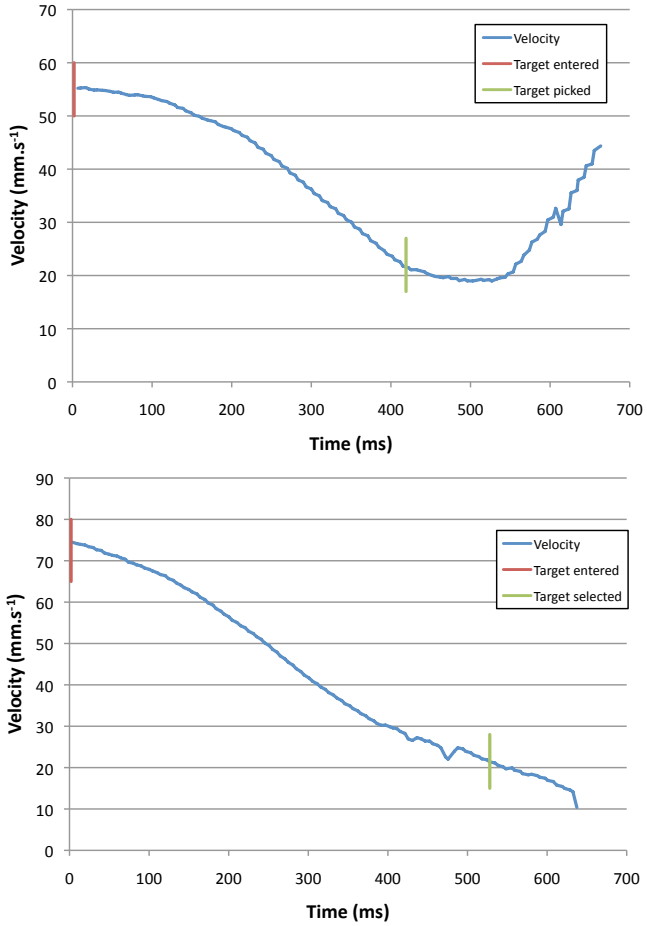

FIGURE 4.2 – Profil de vitesse pour les tâches de picking (en haut) et de sélection (en bas)quand le curseur entre dans la cible.

de six graphes. L'Appendice 4.5 détaille comment la vitesse et l'accélération ont été calculé et filtrés à partir des positions brutes. Les deux profils d'accélération et de vitesse, calculés à travers le mouvement complet, n'ont pas montré de schémas spécifiques. Au lieu de cela, une caractéristique discriminante à la fin de chaque mouvement peut être observée, quand le pointeur entre dans la cible. Parmi les différent profils calculés, la caractéristique la plus spécifique a été observée pour le picking. Pour cette tâche, l'accélération du périphérique en entrant dans la cible décélère jusqu'à atteindre un minimum et augmente jusqu'à ce que l'utilisateur appuie sur un bouton. Le profil correspondant est illustré Figure 4.1 (en haut). Le moment auquel le pointeur entre dans la cible est représenté par la ligne rouge, et la ligne verte représente le moment auquel le bouton est appuyé. La Figure 4.1 (en bas) représente le profil d'accélération correspondant à la tâche de sélection. Les auteurs n'ont pu trouver un profil similaire pour l'accélération entre le moment auquel nous entrons dans la cible et le moment auquel le bouton est pressé. Ce profil caractéristique pour la tâche de picking peut être expliqué par le fait que le cerveau planifie déjà une nouvelle trajectoire de mouvement, avant de presser le bouton. La trajectoire est planifiée pour entrer dans

la cible et sortir par la zone de lâché attendue dans une direction donnée. L'analyse de trajectoire a confirmé que la distance entre la position du pointeur et le centre de la cible augmente dès que le bouton est pressé. Pour la tâche de sélection, le profil de vélocité (Figure 4.2) représente le profil le plus caractéristique. La vélocité continue de décroître une fois que la cible est entrée (représenté par la ligne rouge) jusqu'à ce que le bouton soit pressé (ligne verte) et relâché (fin de la courbe bleue). La vitesse moyenne quand le bouton est pressé est de $21mm.s^{-1}$ (SD = $8mm.s^{-1}$). Pour les deux tâches de sélection et de pointage, la vitesse moyenne lors de l'entrée de la cible est égale à $75mm.s^{-1}$ (SD = $40mm.s^{-1}$). Dans la section suivante, nous détaillons l'algorithme proposé par Choumane *et al.*, prenant en compte les observations de l'étude présentée dans cette section pour prédire les tâches de sélection et de glisser-lâcher.

4.1.2 Algorithme proposé dans [34]

En se basant sur les résultats de l'étude préliminaire, les auteurs proposent un diagramme de transition d'état comme illustré dans la Figure 4.3 pour sélectionner, désélectionner, glisser et lâcher des objets. Comme il n'est possible de discriminer les profils d'accélérations que lorsque l'utilisateur entre dans une cible, l'algorithme vérifie d'abord si le pointeur est à l'intérieur d'un objet 3D qui peut être sélectionné ou pické (état 0). Quand la cible est entrée (état 1), le profil de vélocité, d'accélération et de trajectoire sont analysés. Il y a une vérification dans l'état 1 pour vérifier si la cible est prévue pour être sélectionnée ou pickée, en mesurant la vitesse du périphérique. Si c'est au dessus de $V_{threshold1}$ nous considérons que l'utilisateur passe à travers la cible sans aucune intention d'interactions avec. La valeur pour $V_{threshold1}$ est de $115mm.s^{-1}$ car l'étude préliminaire a montré que la vitesse moyenne lors de l'entrée d'une cible est égale à $75mm.s^{-1}$ avec une déviation standard égale à $40mm.s^{-1}$. Si la vitesse du périphérique reste sous $V_{threshold1}$ la discrimination se fait ensuite entre les tâches de sélection et de picking. La tâche de picking est conditionnée par les deux critères trouvés dans l'étude initiale : un pic minimum dans le profil d'accélération et une augmentation de la distance au centre de la cible (*isIncresead(D)*). L'augmentation de la distance est vérifiée après que le pic minimum d'accélération est détecté. Si les deux conditions sont remplies la cible est sélectionnée (état 3). Pour la tâche de sélection, il faut que la vitesse du périphérique reste sous une limite de vitesse $V_{threshold2}$ pendant une période de temps définie. D'après l'étude initiale, Choumane *et al.* ont trouvé une vitesse moyenne égale à $21mm.s^{-1}$ (SD = $8mm.s^{-1}$) quand le bouton est pressé. Ils utilisent une vitesse égale à $29mm.s^{-1}$ pour $V_{threshold2}$ et utilisent 180 ms pour la période de temps. L'état de la cible (sélectionnée ou désélectionnée) est alors activé. La relâche de la cible survient quand la vitesse du périphérique tombe sous une troisième limite de vitesse $V_{threshold3}$. La valeur moyenne utilisée est de $7mm.s^{-1}$ (trouvée dans l'étude initiale), avec la déviation standard correspondante.

La figure (4.3) présente le diagramme de transition d'états de l'algorithme pour les tâches de sélection, désélection, glisser et lâcher. D est la distance entre la position actuelle du pointeur et le centre de la cible entrée. La détection du pic de la tâche dépend, en partie, de la fonction $IsIncrease(D)$. Comme mentionnée avant, cette fonction permet de prédire que l'utilisateur quitte la cible. Dans l'expérience formelle (cf. section 4.1.3), les cibles sont des sphères. Dès lors D est la distance entre le centre de la sphère et la

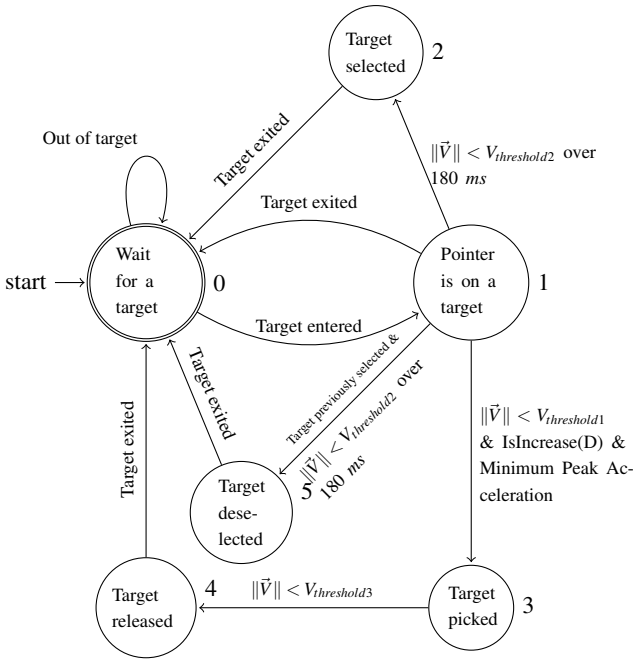

FIGURE 4.3 – Diagramme de transition d'état de l'algorithme

position du pointeur. Nous considérons que notre algorithme est adapté pour les cibles dont la forme géométrique peut être facilement incluse dans une sphère, par exemple, une pyramide, un cube, un cylindre. Avec ces formes, D est bien définie.

4.1.3 Expérience

Le but est d'évaluer l'efficacité de notre algorithme en mesurant le pourcentage d'actions correctement reconnues pour chaque sous-tâches : sélection, glisser-lâcher. Comme l'algorithme est basé sur l'analyse cinématique des gestes, il est également intéressant d'établir sa robustesse face aux paramètres qui peuvent affecter le profil cinématique : la taille de la cible (la taille de l'objet à sélectionner ou à picker), la taille de destination (la taille de la zone au sein duquel relâcher la cible picked) et l'orientation de la cible (mesurée comme l'angle formé entre le point de départ du mouvement la position de la cible et l'horizontale). La taille de la cible est connue pour affecter le profil de vélocité, avec les petites cibles demandant plus de précision et dès lors réduisant la vitesse du mouvement correctif [105]. L'orientation de la cible affecte le déplacement relatif de la main, ce qui peut créer une différence dans le profil de vitesse. Notre but est de comparer notre technique contre l'alternative du bouton en terme de performances (contre le temps et le taux d'erreurs) et les préférences subjectives des utilisateurs.

Participants

Quatre femmes et quatre hommes ont participé, avec un âge moyen de 26 ans (SD=1,8). Les participants avaient une longueur moyenne de l'avant-bras égale à 27 cm (SD=3), une longueur de bras égale à 29 cm (SD=2,5), et une hauteur moyenne égale à 173 cm (SD=9). Tous les participants étaient droitier et avaient une vision normale ou corrigée pour être normale. Deux participants ont un peu d'expérience avec la réalité virtuelle et les applications 3D, alors que et les six autres n'ont aucune expérience dans ce domaine. Malgré cela, l'échantillon est valable, car nous observons un comportement physique de plus bas niveau que lors de l'interaction avec un espace virtuel.

Parmi les participants, quatre étaient chercheurs en informatique, trois étaient des ingénieurs en électronique et un était docteur en médecine. Aucun n'avait participé à l'étude initiale.

Appareillage

L'expérience s'est déroulée sur un écran large rétro-projeté de 6500 mm, avec un affichage stéréoscopique de 2344 × 1050 pixels de résolution (202 × 203 mm), une densité de pixels de 96 DPI, et un rafraîchissement de 120 Hz. La position de la main était suivie en utilisant un périphérique DTrack avec un système optique infrarouge ARTtrack [7]. Le système ARTrack donne une position avec une résolution de 0,06 mm dans toutes les directions, 0,4 mm de précision et un échantillonnage à 60 Hz. Le périphérique DTrack était positionné au-dessus de la main. Pour la condition sur le bouton, nous avons utilisé le même dispositif, avec les participants tenant une souris sans fil dans la main qui était suivie. Le bouton gauche de la souris a été utilisé pour les interactions.

Tâche

Deux tâches furent utilisées dans l'expérience : une tâche de pointage multidirectionnelle et une tâche de glisser-lâcher multidirectionnelle. Les deux tâches étaient évaluées avec et sans le bouton (figure 4.4). Pour chaque tâche, quatre cibles étaient placées à égale distance autour d'un cercle positionné au centre de l'écran sur un plan parallèle à l'écran. Le pointeur était représenté par une sphère rose de 2 mm de diamètre, mesurée au centre du cercle. L'expérience utilise un gain constant de 1 :1 pour la mise en correspondance de la main avec la position du curseur 3D [29]. Pour la tâche de sélection, la cible à sélectionner apparaît en rouge, alors que les autres cibles restent grises. Après la sélection réussie de la cible, elle disparaît, et la cible suivante devient rouge. Les cibles *picked* sont comptées comme erreurs et doivent être relâchées n'importe où avant que la cible suivante devienne rouge. Un scénario similaire a été utilisé pour la tâche de pick-and-release, excepté pour la cible à choisir qui devient verte, et pour la zone de destination du geste, représentée par une sphère à une position pseudo-aléatoire. La sphère de destination apparaît blanche et devient verte quand la cible est complètement à l'intérieur de celle-ci. Les cibles qui ont été relâchées par inadvertance sont comptées comme erreur et doivent être sélectionnées à nouveau, jusqu'à être relâchée à la destination correcte. Les cibles sélectionnées par inadvertance sont comptées comme erreurs avant que la cible suivante à sélectionner devienne verte. En plus de la perspective et des occlusions, furent ajoutées la projection des ombres sur le sol pour le pointeur, les cibles et la zone de destination, afin d'améliorer la perception de la profondeur. La caméra reste fixe pendant toute la durée de

7. http ://www.ar-tracking.de/

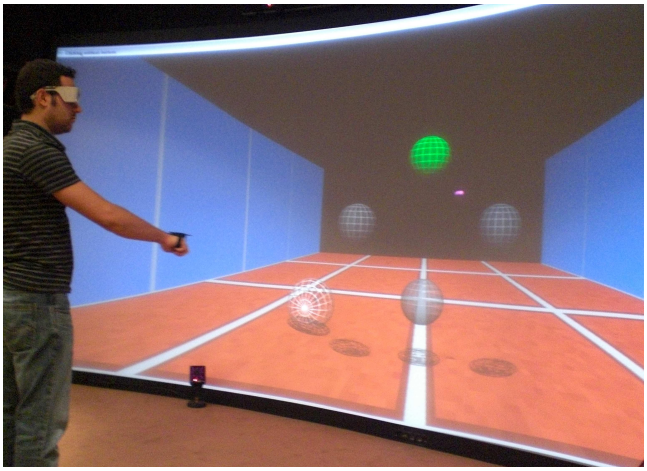

FIGURE 4.4 – Photo de l'appareillage pour l'expérience [34].

l'expérience. Les participants ont reçu l'instruction d'effectuer la tâche aussi précisément que possible. Pour tous les participants, les paramètres du modèle ont été établis aux mêmes valeurs, données par l'étude initiale.

La figure (4.4) présente une photo du dispositif expérimental pour les tâches de glisser-lâcher. Le matériel expérimental présent est constitué d'un écran large, d'un périphérique DTrack attaché au dos de la main du participant, les cibles à glisser-lâcher et la destination sur le sol.

Design

Un design de mesures répétées intra-sujets a été utilisé. Les variables indépendantes sont TECHNIQUE (le système utilisé avec ou sans bouton), TÂCHE (sélection, pick, relâcher), ORIENTATION (0°, 90°, 180°, 270°), TAILLE DE CIBLE (6 mm, 12 mm, et 24 mm - mesuré au centre du cercle), et TAILLE DE DESTINATION pour la tâche de release (10 mm et 20 mm plus grand que la taille de la cible). L'orientation est mesurée à partir de l'axe horizontal dans le sens contraire des aiguilles d'une montre (le 0° est positionné à droite de la scène). La tâche de pick-and-release est décomposée en deux sous-tâches : la tâche de picking et la tâche de release. Les essais ont été organisés en BLOCS pour mesurer l'effet d'apprentissage : six blocs pour la technique sans boutons et deux blocs pour la technique avec bouton. Chaque BLOC est composé était composé des trois tâches évaluées à chaque fois avec trois TAILLE DE CIBLE, quatre ORIENTATION et deux TAILLE DE DESTINATION pour la tâche de Release. TÂCHE et TECHNIQUE ont été contrebalancés à travers les participants. Cela donne un total de $8 \times 480 = 3840$ essais au total. L'expérience a duré approximativement 60 minutes.

4.1.4 Résultats

Les variables dépendantes sont le temps du mouvement et le taux de succès.

Temps du mouvement

Les essais marqués comme erreurs ont été enlevés de l'analyse du temps de mouvement. Des mesures répétées d'analyse de variance n'ont montré aucun effet significatif ($F_{1,7}$ = 0.017, p = 0,9) pour la TECHNIQUE, la TÂCHE ($F_{2,14}$ = 3.426, p = 0,061) et pas d'interaction significative entre TÂCHE et TECHNIQUE ($F_{2,14}$ = 2.91, p = 0,088) sur le temps du mouvement. Ces résultats nous ont menés à la conclusion q'il n'y a pas de dégradation dans le temps nécessaire au mouvement avec la technique sans bouton. D'autres évaluations sont nécessaire pour montrer qu'il n'y a pas de "petit effet".

Taux de succès

Le taux de succès pour la TÂCHE Release est calculé à partir des cibles picked avec succès. Des mesures répétées d'analyse de variance n'ont montré aucun effet significatif pour BLOC sur le taux de succès, montrant qu'il n'y a pas d'effet d'apprentissage.

La figure (4.5) montre le taux de succès moyen pour chaque TÂCHE et TAILLE DE CIBLE pour la technique sans bouton. Les barres d'erreurs représentent 95% de l'intervale de confiance.

Des mesures répétées d'analyse de variance ont montré un effet significatif de la TECHNIQUE ($F_{1,7} = 67.5$, $p < 0{,}001$) et un effet significatif entre TECHNIQUE et TÂCHE ($F_{2,14}$ = 9.0, p = 0.03) sur le taux de réussite. Le taux de réussite global est de 96,2% avec bouton et 90,1% sans bouton. La TÂCHE Release ne montre pas de différence significative (p = 0,37) avec un taux de réussite de 92,6% avec bouton et 90,7% sans bouton. Pour mieux comprendre les facteurs qui influencent le taux de succès sans l'utilisation de boutons, nous avons enlevé les données de la condition avec bouton pour la suite de l'analyse.

Les mesures répétées d'analyse de variance ont montré un effet significatif pour TÂCHE ($F_{2,14} = 41.2$, $p < 0{,}001$), TAILLE DE CIBLE ($F_{2,14} = 17.4$, $p < 0{,}001$) sur le taux de réussite et une interaction significative entre TÂCHE et TAILLE DE CIBLE ($F_{4,28} = 10.8$, $p < 0{,}001$) sur le taux de réussite. La comparaison pair-à-pair montre des différences significatives (p < 0,018) entre les différentes tailles de cibles de la TÂCHE Select avec un taux de succès de 70,5% pour la cible de 6 mm, 82,8% pour la cible de 12 mm et 95,4% pour la cible de 24

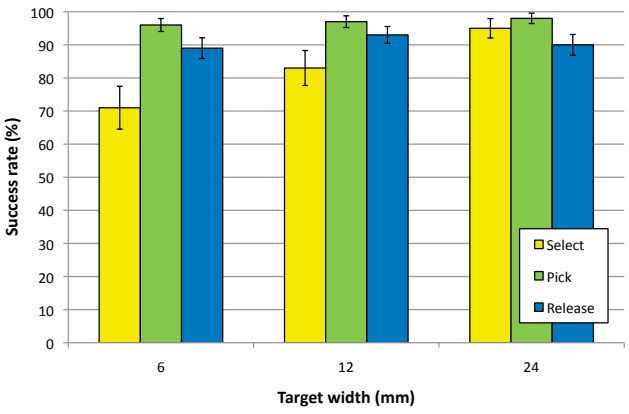

FIGURE 4.5 – Taux de succès moyen pour chaque Tâche et Taille de cible [34].

mm. Les mesures répétées d'analyse de variance ont montré un effet significatif principal sur la Taille de la Destination ($F_{1,7} = 5.9$, $p = 0{,}045$) sur le taux de succès avec 88,5% pour la taille la plus petite et 93% pour la la taille la plus large.

Résultats Subjectifs

Après chaque bloc, les participants ont évalué leur niveau de fatigue en utilisant un échelle de Likert sur 5 points (1 : aucune fatigue, 5 :très fatigué). La valeur moyenne pour tous tous les participants est restée constante autour de 3 à travers tous les blocs ne montrant aucune fatigue s'accroissant.

A la fin de l'expérience, il a été demandé aux participants de choisir leur technique favorite. La moité des participants ont déclaré préférer la technique sans bouton, et l'autre moitié la technique avec bouton. Les participants qui ont préféré la technique avec bouton ont précisé qu'elle était plus proche de la technique basée sur la souris de laquelle ils sont déjà familiers. Des feedbacks informels qualitatifs, il apparaît que les participants étaient au début surpris à propos de l'idée de sélectionner et de déplacer des objets sans bouton, mais ils ont rapidement adopté et apprécié la technique. Une observation : parfois les personnes fermaient intuitivement leur main pour pick la cible. Un participant a dit : "c'est comme si l'ordinateur lisait mon esprit", un autre a dit : "c'est beaucoup plus amusant sans bouton".

Discussion à propos de l'expérience

L'expérience a validé l'algorithme avec un taux de succès global de 90,1% à travers toutes les conditions. Le résultat est plus bas que le taux de succès de 96,2% pour la technique avec bouton, mais nous étions satisfaits que de voir que notre algorithme ne requiert presque aucun apprentissage et qu'il est robuste à travers tous les participants avec les paramètres établis dans l'étude initiale. Nous évaluons l'influence de l'orientation, la taille de la cible et la taille de destination comme paramètre parasite (hors de contrôle de l'expérimentateur). Les résultats expérimentaux montrent que l'orientation n'a aucune influence sur le taux de reconnaissance, alors que la taille de la cible affecte de manière significative le taux de réussite sur la Sélection, avec les cibles les plus petites faisant décroître le taux de reconnaissance. De plus, les résultats sur le temps de mouvement indiquent que la technique sans bouton est équivalente à la technique avec bouton, et les résultats sur la fatigue n'ont montré aucune fatigue spécifique associée à la technique sans bouton.

Suite à cette expérience, nous avons noté le taux de succès moins élevé pour les cibles de petites tailles. Afin de compléter la technique et de la rendre plus robuste, nous proposons dans la section suivante d'améliorer l'algorithme en se basant sur la taille des cibles.

Le travail qui suit présente maintenant les améliorations possibles pour la technique lorsque l'utilisateur travaille avec des petites cibles.

4.2 Améliorer du clic sans bouton pour les petites cibles

Les résultats de l'étude expérimentale effectuée dans [34] ont montré une dégradation des taux de reconnaissance pour les cibles de petite taille (<= 12 mm). Choumane *et al.* ont émis l'hypothèse que les utilisateurs altèrent leur profil cinématique en se basant sur la taille des cibles, donnant en résultat des profils pour les petites cibles qui diffèrent de ceux calculés dans la section 3 à travers toutes les tailles de cibles. Pour établir cette hypothèse, ils ont calculé le profil d'accélération moyenne pour chaque taille de cible et type de tâche à partir des données collectées dans l'expérience (cf. section 4.1.3).

Le calcul des profils d'accélération pour les petites cibles ($\leq$ 12 mm) et les larges en utilisant 60% de l'ensemble des données présente en Figure 6 les profils correspondant pour les petites cibles et en Figure 7 pour les cibles larges. Le moment auquel le pointeur entre dans la cible est représenté par la ligne rouge, et la ligne verte représente le temps auquel la cible est picked ou sélectionnée.

Pour les cibles larges, nous avons observé un pic d'accélération similaire à celui trouvé dans l'étude initiale (Figure 4.7). Au contraire, nous n'avons pu trouver un tel profil pour les petites cibles. En fait, le pic d'accélération est observé pour la tâche de sélection. Ceci explique pourquoi l'algorithme, basé sur la détection de ce minimum, échoue à discriminer entre la sélection et le picking pour les petites cibles alors que la technique présente un taux de reconnaissance élevé pour les cibles larges.

Après avoir essayé un jeu de candidats pour discriminer entre la sélection et le picking pour les petites cibles, on note une augmentation significative dans l'accélération juste avant que la cible soit actuellement choisie (Figure 4.7 en haut). Cette augmentation de l'accélération est définie comme $A_{threshold}$. En se basant sur cette observation, nous proposons de prendre en compte la taille de la cible dans notre algorithme proposé précédemment (cf figure 4.3) comme ceci :

- si Taille de la Cible > 12 mm, l'algorithme montré dans la figure 4.3 est utilisé
- si Taille de la Cible $\leq$ 12 mm, nous remplaçons dans la figure 4.3 la condition Pic Minimum d'Accélération pour la transition entre l'état 1 et 3 par $\left\|\vec{A}_t\right\| - \left\|\vec{A}_{(t-1)}\right\| > A_{threshold}$

où $\left\|\vec{A}_t\right\|$ est l'accélération instantanée actuelle, $\left\|\vec{A}_{(t-1)}\right\|$ est l'accélération instantanée à l'étape précédente et $A_{threshold} = 15\ mm.s^{-2}$. Cette valeur a été calculée à partir des 60% du jeu de données afin d'optimiser le taux de reconnaissance.

Dans la section suivante, nous présentons l'évaluation de ce nouvel algorithme en utilisant 40% du reste du jeu de données (le reste ayant été utilisé pour construire l'algorithme).

La figure (4.6) nous donne le profil d'accélération lorsque la taille de la cible est $\leq$ 12 mm : profil d'accélération pour les tâches de picking (en haut) et de sélection (en bas) lorsque le curseur entre dans la cible.

La figure (4.7) nous donne le profil d'accélération lorsque la taille de la cible est > 12 mm : profil d'accélération pour les tâches de picking (en haut) et de sélection (en bas) lorsque le curseur entre dans la cible.

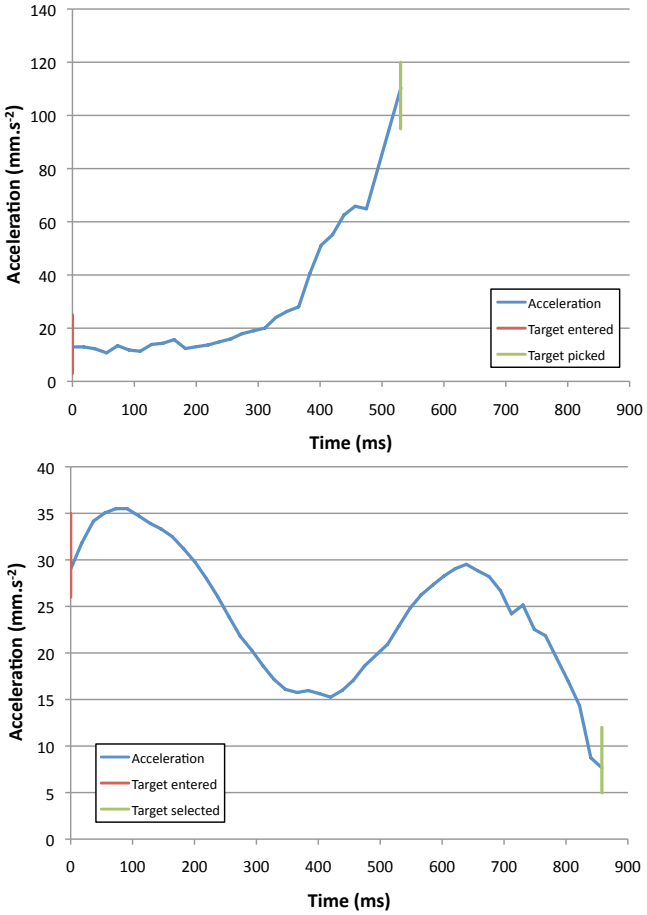

FIGURE 4.6 – Profil d'accélération pour les tâches de picking (en haut) et de sélection (en bas) lorsque le curseur entre dans la cible ($\leq$ 12 mm)

4.2.1 Évaluation

Nous avons évalué le nouvel algorithme pour les petites cibles (6 et 12 mm) pour mesurer le pourcentage d'actions correctement reconnues pour les tâches de sélection et pick. Les données pour les cibles larges (24 mm) ont été enlevées car la taille n'est pas affectée par l'amélioration proposée.

Les mesures répétées d'analyse de variance n'ont montré aucun effet significatif pour la TÂCHE, TAILLE DE LA CIBLE, et pas d'interaction significative entre TÂCHE et TAILLE DE LA CIBLE sur le taux de succès.

Le taux de succès pour la TÂCHE de Sélection est de 92,4% pour les cibles de 6 mm et 93,5% pour les cibles de 12 mm. Le taux de succès pour la TÂCHE Pick est de 96% pour les cibles de 6 mm et 95,1% pour la cible de 12 mm (Figure 8).

Le nouveau taux de succès de notre algorithme est de 94% à travers toutes les conditions, ce qui est supérieur à la proposition de référence ([34]). Ce résultat montre que notre méthode améliorée représente une alternative valide pour l'utilisation des boutons pour les tâches de sélection et de glisser-lâcher.

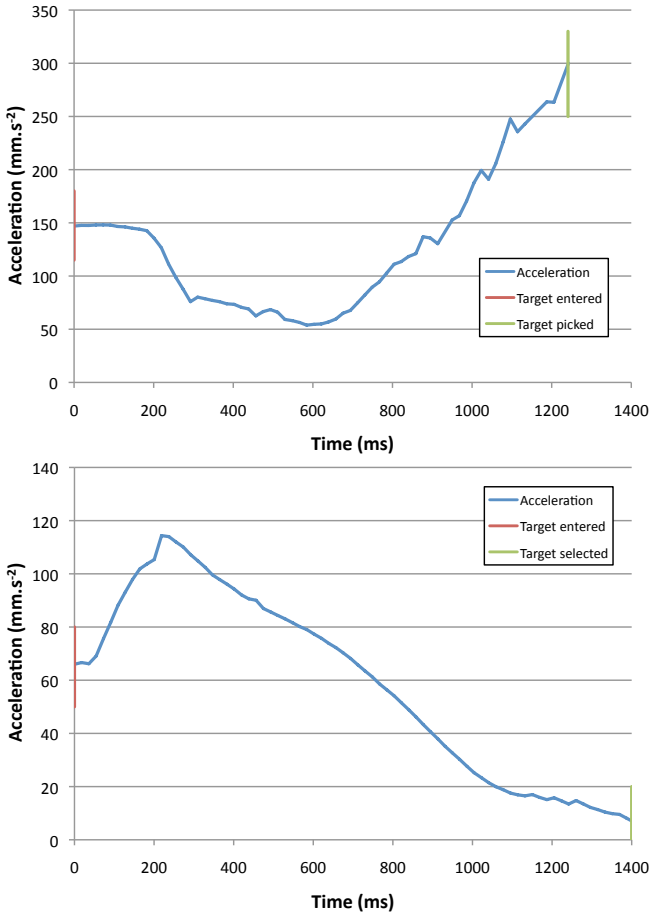

FIGURE 4.7 – Profil d'accélération pour les tâches de picking (en haut) et de sélection (en bas) lorsque le curseur entre dans la cible > 12 mm

4.2.2 Discussion globale

Petites cibles

Dans l'étude initiale, un invariant avait été identifié dans le profil d'accélération pour la tâche de picking. Nous avons ensuite proposé un algorithme basé sur les invariants d'accélération et les autres invariants cinématiques pour discriminer entre sélection, désélection et glisser-lâcher. Cet algorithme a été évalué dans une expérience contrôlée où les cibles de petites tailles ont montré un taux de réussite moins élevé alors que les cibles larges conservaient un taux de succès élevé. L'analyse du profil cinématique pour les petites cibles a révélé que le minimum du pic d'accélération pour la tâche de sélection. Les cibles utilisées dans l'étude initiale se sont révélées être trop larges pour observer ce comportement.

La figure (4.8) nous donne le taux de succès moyen pour les TÂCHES Pick et Select et les TAILLE DE CIBLE 6 et 12 mm pour la technique sans bouton. Les barres d'erreurs représentent 95% de l'intervale de confiance.

Nous avons émis l'hypothèse que cette différence dans les profils d'accélération entre

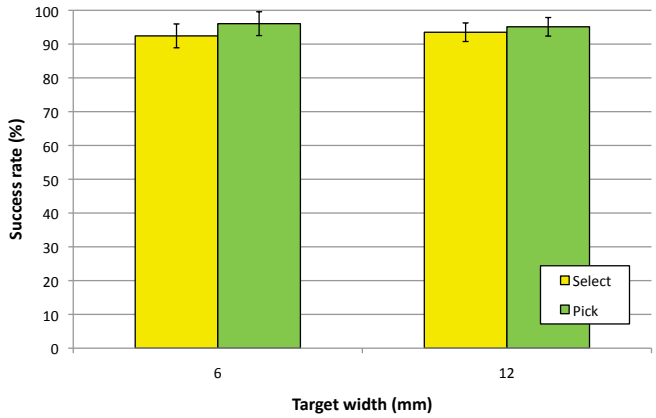

FIGURE 4.8 – Taux de succès moyen pour les tâches Pick et Select par taille de cible pour la technique sans bouton.

les petites cibles et cibles larges peut être expliqué par la difficulté à acquérir les petites cibles. Pour les cibles larges, les utilisateurs sont confiant dans la capacité de picker la cible et planifient déjà la suite du mouvement en se basant sur l'endroit où la cible va être relâchée. Le pick minimum d'accélération observé avant d'appuyer sur le bouton dans l'étude initiale corrobore la théorie développée par Guiard [55] sur la co-articulation des mouvements : la fin d'un mouvement est altéré par la planification du début de la suivante, un phénomène observé précédemment dans l'écriture manuscrite. Pour les petites cibles, nous avons émis l'hypothèse que les utilisateurs anticipent la difficulté du picking de la cible et ne peuvent alors plus co-articuler la fin du mouvement avec le début du suivant. Au lieu de cela, le pick minimum d'accélération est observé pour la sélection à cause des ajustements nécessaires avant la sélection de la cible. L'augmentation de l'accélération pour la tâche de pick et des petites cibles peut être expliqué par un geste rapide effectué dans la direction de la zone de relâchement quand le pointeur est sur la cible.

Control Display Gain

Dans l'étude initiale et l'expérience présentée par Choumane *et al.*, le pointeur se déplaçait à la même vitesse que la main. Cette mise en correspondance entre les mouvements de la main et du curseur est plus connu sous le nom de Control-Display Gain (CD gain), exprimé comme le ratio entre la vitesse du pointeur et la vitesse de la main [29]. Dans l'expérimentation, cela correspondait à un gain constant égal à un. Avec un écran plus petit, le gain CD devrait être réduit pour confiner les mouvements de la main au sein des bords de l'écran. Au contraire, un écran plus large devrait requérir d'augmenter le CD gain pour atteindre l'intégralité de l'espace de travail.

Casiez *et al.* ont montré que le CD gain a un impact sur le profil cinématique de la trajectoire de l'utilisateur avec les CD gains plus élevés réduisant la vitesse maximum du mouvement [29]. Par contre, leur expérience ne dit pas si le CD gain a un effet sur le profil cinématique de la fin du mouvement de l'utilisateur. Des expériences supplémentaires doivent être conduites pour répondre à cette question. Si le CD gain affecte la cinématique

de la main de l'utilisateur à la fin de la trajectoire, les seuils de vitesses différents définis dans notre algorithme pourraient avoir besoin d'être affinés. Malgré cela, nous émettons l'hypothèse que la tâche de Picking va montrer le même profil pour l'accélération.

Le travail effectué jusqu'à maintenant se plaçait dans un environnement en trois dimensions, pour plusieurs tailles de cibles. La suite de notre chapitre ramène le problème en deux dimensions, et cherche à déterminer si la technique sous-jacente à l'algorithme (la reconnaissance de caractéristique particulière dans le geste) est toujours valable.

4.3 Application de la technique en 2 dimensions

4.3.1 Tâches à deux dimensions

Le travail sur la méthode présentée précédemment pour estimer l'intention de l'utilisateur entre une action de glisser-lâcher ou une intention de clic a été mené en 3 dimensions.

L'interaction en 3 dimensions est spécifique, et le geste en 2 dimensions présente suffisamment de différences pour être plus qu'une simple projection. L'étude de ce travail en deux dimensions se justifie pour des raisons médicales, mais également propres aux nouveaux périphériques disponibles pour les utilisateurs.

L'utilisation prolongée d'un système clavier/souris amène des contraintes sur le système musculaire et osseux du membre supérieur. Un syndrome du canal carpien empêche l'action du clic(sur une souris), la contrainte neuro-musculaire étant trop importante. De même, les personnes handicapées ne pouvant mobiliser assez de force pour appuyer sur un bouton ne peuvent utiliser une souris sur une interface graphique classique.

Les nouvelles consoles de jeu sorties récemment sur le marché proposent une interaction en 3 dimensions, avec une gestuelle qui implique une certaine occupation de l'espace (par exemple, le Kinect pour la XBox de Microsoft, ou la Wiimote pour la Wii de Nintendo). C'est d'ailleurs cette gestuelle plus développée qui a amené le personnel hospitalier à utiliser ces consoles en rééducation pour les personnes handicapées ou en maison de retraite.

Ces périphériques impliquent un geste en 3 dimensions, projeté dans un espace en 3 dimensions, mais sans nécessairement proposer de dispositif pour cliquer (par exemple le Kinect). Une solution logicielle, qui se base sur le geste de l'utilisateur est alors nécessaire pour comprendre son intention et son interaction.

Tout les interactions ne sont pas projetées dans un espace en 3 dimensions. Il existe de nombreux jeux qui utilisent toujours un geste en 3 dimensions, mais qui se retrouve projeté sur une interface graphique en 2 dimensions. Il existe également des situations, hors du domaine du jeu, qui présentent ce genre d'interactions. Nous pouvons penser par exemple à un conférencier qui présente des transparents sur un mur, et qui désire pointer des éléments de la présentation à l'aide d'un périphérique qui se déplace en 3D, avec sa main, mais qui projette un pointeur dans un espace en 2D (la présentation elle-même).

Sur l'interface graphique d'un système informatique actuel, l'utilisateur peut effectuer des tâches de glisser-lâcher et des clics avec une souris posée sur un plan de travail, contrairement au système de suivi 3D qui était libre.

La configuration spatiale du corps étant différente, nous pouvons prendre l'hypothèse

qu'afin de faire fonctionner la technique dans ce nouveau contexte, il faut au minimum recalculer les valeurs de vitesses limites nécessaire au fonctionnement de la machine à état présentée figure 4.3.

Dans l'état actuel de la technologie et de l'utilisation quotidienne des systèmes informatiques, les utilisateurs n'ont pas l'habitude d'effectuer des tâches précises de pointages en 3 dimensions. Des lors, les bonnes performances de la technique présentée ici pourraient se voir dégradées, une fois cette technique nouvelle transposée à un contexte d'utilisation qui serait lui plus familier aux usagers.

4.3.2 Expérience

Pré-expérience

Nous avons réalisé une pré-expérience afin de déterminer les nouvelles valeurs pour les seuils de vitesse sur lesquels l'automate base ses décisions.

En affichant une série de cible, de manière similaire au protocole décrit par après pour l'expérience finale (une série de cibles circulaires), nous avons pu mesurer les vitesses lors des tâches de clics et de glisser-lâcher. Nous avons mesuré la vitesse de la souris lorsque le curseur entrait dans la cible, lorsque l'utilisateur cliquait avec le bouton de la souris et lorsque l'utilisateur en relâchait le bouton.

Les utilisateurs présents lors de cette pré-expérience présentaient une grande variabilité dans l'utilisation de leur souris. Certains poussent l'accélération système du curseur au maximum, d'autres presque au minimum. Certains des utilisateurs réalisent souvent des actions nécessitant une forte précision avec la souris (par exemple jouer à des jeux de tirs sur ordinateur), ce qui a tendance à augmenter la variabilité inter-sujets quand aux vitesses préférées pour utiliser la souris.

Ainsi, pour les sujets de la pré-expérience, la vitesse moyenne lors de l'entrée sur la cible est de 99 $mm.s^{-1}$, avec une Déviation Standard de 65 $mm.s^{-1}$. La variation sur les vitesses lors de la manipulation du bouton de la souris est moins importante. En effet, ici, les contraintes physiologiques du système neuromusculaire combinées à une posture de travail relativement similaire - hauteur du bureau et du siège ont tendance à amortir ces variations : 30 $mm.s^{-1}$ lors du clic sur la cible (SD = 7 $mm.s^{-1}$) et 29 $mm.s^{-1}$ (SD = 8 $mm.s^{-1}$) lorsque le bouton est relâché.

Hypothèse

Nous prenons l'hypothèse que la technique de clic sans bouton fonctionne également en deux dimensions, avec un périphérique de suivi reposant sur un plan de travail, par exemple une souris. L'efficacité de la méthode est conditionnée par les seuils de vitesse, qui sont à déterminer dans ce nouveau contexte. Les utilisateurs ayant par ailleurs l'habitude d'utiliser une souris pour réaliser ces mêmes gestes, les performances pourraient malgré tout être moins bonnes qu'en 3 dimensions.

Utilisateurs

9 personnes ont participé à l'expérience, 7 hommes et 2 femmes, (âge moyen 25 ans). Tous les utilisateurs sont droitiers, avec une vue normale ou corrigée pour être normale, et utilisateurs quotidien d'un ordinateur, avec une souris.

Matériel

Un PC sous Windows XP SP3 a été utilisé, avec une souris USB standard à 400 DPI. Le système est branché sur un écran LCD 19 pouces, d'une résolution de 1280 × 1024 à 96 DPI (1 pixel = 0.295 mm), rafraîchi à 60 fps.

L'application est programmée en C++/OpenGL, et contourne le pilote de la souris. Les informations de déplacement sont obtenues avec DirectInput, afin de garder les données non prétraitées par le système d'exploitation.

Les utilisateurs sont installés sur un bureau dégagé, à 50 cm de l'écran.

Tâche

L'utilisateur doit accomplir une tâche de pointage, en utilisant le pointeur de la souris pour, soit cliquer une cible, soit réaliser un mouvement de glisser-lâcher de cette cible vers son emplacement de destination. Les cibles et les destinations sont circulaires, ce qui permet de généraliser, par la suite, à des formes plus complexes (ces formes complexes seraient toujours comprises à l'intérieur d'un disque).

Lors qu'une cible a été saisie pour un glisser-lâcher et que l'utilisateur l'amène vers la destination, celle-ci change de couleur lorsque la cible l'intersecte, pour donner un *feedback* à l'utilisateur.

Une cible considérée comme cliquée par l'algorithme disparaît de l'écran, alors qu'une cible considérée comme saisie (pour une tâche de glisser-lâcher) se retrouve attachée au curseur, et placée en son centre.

Lorsqu'une cible est sélectionnée par erreur, la cible suivante est présentée automatiquement à l'utilisateur, alors qu'une cible saisie pour un glisser-lâcher par erreur doit être relâchée par l'utilisateur, afin que celui-ci puisse cliquer dessus ensuite. En effet, on peut considérer qu'une action de glisser-lâcher entamée par erreur peut être corrigée (éventuellement par l'utilisateur qui replace la cible sur son emplacement de départ), alors qu'un clic est une tâche *modale*, qui ne permet pas de de corriger l'action une fois qu'elle est réalisée. (Une tâche modale impose un changement de *mode*, autrement une action qui ne peut pas être interrompue une fois qu'elle s'effectue).

Tous les 10 clics, un message de fin de bloc indique aux utilisateurs à quelle étape de l'expérience ils se situent. Le curseur de la souris peut sortir des limites de l'écran, ce qui évite d'utiliser les bords de la fenêtre pour faciliter l'acquisition de la cible.

L'utilisateur doit réaliser dix répétitions de pointage, suivies de dix répétitions de glisser-lâcher, et ceci répété cinq fois, pour constituer cinq blocs. Un glisser-lâcher réalisé dans une tâche de pointage est considéré comme une erreur, tout comme l'inverse. Aucune accélération n'est appliquée au curseur de la souris, donnant une mise en correspondance entre le déplacement de la main et celui du curseur à l'écran de 1 pour 1.

Une pré-expérience a été construite, afin d'ajuster les variables de l'expérience à trois dimensions à celle à deux dimensions. Les utilisateurs ne pouvaient réaliser les tâches de pointages et de glisser-lâcher en utilisant les valeurs d'heuristiques déterminées dans l'expérience en trois dimensions. En effet, les seuils de vitesse utilisés ont été mesurés avec un bras tendu dans l'espace, alors que dans notre nouvelle configuration, le bras et le périphérique reposent sur un support solide.

Design

Le design de l'expérience est intra-sujets. Les variables indépendantes sont les Techniques avec deux niveaux : Clic, ou Glisser-Lâcher.

4.3.3 Résultats

L'expérience n'a pas pu être menée à son terme, dans la configuration décrite précédemment. En effet, bien que la pré-expérience se soit déroulée normalement pour tous les utilisateurs, et bien que les valeurs de vitesse collectées aient été revérifiées, les utilisateurs n'arrivaient pas à se saisir de la cible de manière satisfaisante. En quelques secondes, le taux d'erreurs montait à 100%, et les utilisateurs arrêtaient tout simplement l'expérience, de frustration.

Les raisons possibles de ce problème d'acquisition sont discutées dans la section suivante.

4.4 Discussion

Alors que les utilisateurs ont pu utiliser la technique décrite dans cette partie dans un environnement à 3 dimensions, cette technique s'est révélée inutilisable telle quelle une fois adaptée en 2 dimensions.

Plusieurs facteurs peuvent expliquer ce résultat. Tout d'abord, alors que, même pour des informaticiens, les interfaces en 3 dimensions sont rares à manipuler, les utilisateurs ont une certaine habitude des interfaces en 2 dimensions. Cette habitude a pu amener la création d'habitudes d'utilisation des périphériques tels que la souris, et des interfaces 2D, habitudes qui rentrent en conflit avec notre méthode. Une analyse rapide des vitesses utilisées par les sujets lors de manipulations sans instructions (pas d'obligation de faire soit un pointage, soit un glisser-lâcher) a montré que ces vitesses étaient en fait différentes des vitesses moyennes relevées lors de la pré-expérience. L'aspect artificiel de l'expérience a donc pu amener les sujets à modifier inconsciemment leur comportement, suffisamment pour mettre en défaut la machine à état de notre expérience. De plus, lors des discussions avec les sujets de l'expérience, l'auteur s'est rendu compte que les utilisateurs avaient la sensation de modifier leur geste "naturel", en anticipation de la tâche à réaliser (que ce soit pour le glisser-lâcher ou la sélection). L'auteur pensait alors réaliser une expérience où le système apprendrait, en temps-réel, les profils de vitesse et d'accélération de l'utilisateur, afin de construire un ensemble de valeurs les plus adaptées à chaque utilisateurs. Pour pouvoir tester ces valeurs, il faudrait pouvoir supprimer la possibilité de cliquer

pour l'utilisateur de temps à autre, afin de tester si les valeurs testées sont pertinentes. Malheureusement, l'interruption du processus naturel du geste de pointage ou de glisser-lâcher nous ramènerait dans une configuration où l'utilisateur prendrait conscience du geste effectué, et risquerait alors de modifier son geste (qui n'en serait plus "naturel"). Le système perdrait alors du temps à rechercher un objectif fuyant, car l'utilisateur le modifierait constamment.

La technique se base à l'origine sur une caractéristique *particulière* d'un geste *qui est en train d'être appris* (ici le pointage dans un espace en 3 dimensions) ; la technique peut alors mettre en exergue la caractéristique nécessaire à son fonctionnement. Elle doit être alors modifiée pour prendre en compte un geste *déjà appris*, et qui ne présente plus cette boucle éventuelle de rétro-action.

4.5 Conclusion

Nous avons uilisé l'approche de Choumane *et al.*. pour discriminer entre les tâches de sélection et de picking-and-releasing, basée sur l'analyse cinématique et de la trajectoire, ce qui est particulièrement utile quand la fonctionnalité du click n'est pas disponible. L'expérience contrôlée montre un taux de reconnaissance élevé. Sur cette approche novatrice, nous avons proposé une extension dans le cadre des petites cibles, une adaptation de la méthode qui améliore le taux de reconnaissance.

Comme travail futur, un point important serait d'obtenir une meilleure compréhension de l'impact du *CD gain* sur la technique proposée. En addition, nous voudrions investiguer comment une technique de ce genre peut être utilisée pour des tâches de manipulation complexes, comme l'assemblage d'objets.

Suite aux performances dégradées lors du passage de la technique de clic sans bouton de trois dimensions à deux dimensions, nous avons proposé une modification à l'algorithme utilisé à la base.

En prenant l'hypothèse que les utilisateurs sont moins performants lorsque les heuristiques de la machine à état (4.3) sont génériques et non adaptées à la personne, nous proposons de faire effectuer une série de clics et de glisser-lâcher à l'utilisateur afin d'établir *en ligne* les bornes de vitesse qui lui seront propres, et qui permettront d'améliorer ses performances. Nous demandons alors à l'utilisateur de réaliser une tâche rapide de 10 clics et de 10 glisser-lâcher. En rejetant le premier essai et prenant une moyenne des 9 essais restants, nous injectons alors les valeurs de $V_{threshold1}$, $V_{threshold2}$ et $V_{threshold3}$.

La technique s'avérant inexploitable dans un espace à 2 dimensions, en temps raisonnable pour les utilisateurs, lorsque nous nous bornons à mesurer de nouvelles vitesses pour l'interaction en 2D, l'auteur présente un argument sur l'apprentissage de ladite technique.

Il se pourrait que les utilisateurs, en apprenant une nouvelle technique d'interaction, dans un environnement nouveau qu'ils apprennent en même temps, modifient leur comportement par rétro-action afin "d'apprendre la machine". Cet apprentissage de l'environnement étant déjà effectué dans un contexte classique (les interfaces graphiques d'ordinateurs), la technique ne semble plus fonctionner.

Comme ouverture pour de futurs travaux, afin de valider cette hypothèse, l'auteur propose de suivre de manière logicielle des personnes n'ayant pas eu à ce jour de contact avec

l'informatique (enfants, personnes âgées ou handicapées), afin de collecter des données anonymisées sur les profils d'utilisation, et leur évolution dans leur apprentissage.

Conclusion

Le travail réalisé durant cette thèse portait sur l'amélioration des tâches de pointage, à l'aide des courbes d'accélération de la souris, pour le pointage et la sélection sans bouton. En partant d'un état de l'art, nous avons pu lister l'ensemble des techniques disponibles actuellement pour aider le pointage. En établissant la bibliographie, nous nous sommes rendu compte que la littérature disponible sur la technique d'accélération du pointeur de la souris était réduite, bien qu'étant déjà utilisée dans les systèmes d'exploitation actuels.

Nous avons alors décidé d'axer notre travail sur cette technique. En effet, les contraintes sur la précision dans le pointage sont plus importantes qu'avant. La surface d'affichage des écrans grandit et les souris disponibles sur le marché sont de plus en plus précises, ce qui concourt à forcer l'utilisateur à effectuer des gestes plus grands, tout en ayant plus de difficultés à acquérir de petites cibles.

Il a fallu tout d'abord pouvoir obtenir les courbes d'accélération utilisées par les systèmes d'exploitation disponibles actuellement au public. Les informations d'implémentation disponibles sont inexistantes (pour MacOS), ou réduites et dépassées (Windows). En utilisant notre propre circuit électronique, destiné à émuler une souris auprès du système d'exploitation, nous pouvons contrôler les informations que nous envoyons au système. En retour, il est possible de mesurer le déplacement du curseur à l'écran. Le dispositif ne demande aucune modification du système d'exploitation, et peut être éventuellement adapté à d'autres périphériques de pointage.

Une fois les courbes obtenues, nous avons pu expliquer les impressions psychophysiologiques des utilisateurs lorsqu'ils changent de machine pour travailler. La sensation que le curseur se comporte différemment lorsque vous utilisez un autre système d'exploitation s'explique par le profil différent des courbes entre ces systèmes. Ces profils différents induisant des performances différentes, nous avons pu nous poser la question de l'impact de telle ou telle caractéristique de la courbe d'accélération sur les performances du pointage. Pour établir ces caractéristiques, nous devons nous comparer à l'existant. Pour ce faire, nous avons réalisé une expérience permettant à l'utilisateur d'effectuer une tâche de pointage utilisant successivement les courbes d'accélération des différents systèmes d'exploitation, et ceci sur la même plate-forme.

En comparant les performances des utilisateurs sur l'acquisition de la cible, nous pouvons établir quelques critères quand au profil des courbes à utiliser pour améliorer le pointage. Ces critères une fois établis, nous avons construit une expérience permettant à nouveau de comparer des courbes d'accélérations entre elles, avec différentes courbes prenant les différentes valeurs possibles dans l'intervalle défini par les critères précédents.

Il ressort de ce travail une série de règles à suivre pour construire une famille de

courbes d'accélération du pointage qui amélioreront les performances de l'utilisateur, ou ne les diminueront pas dans le pire des cas.

Ces critères, que nous résumons dans cette conclusion, sont indépendants de la plateforme utilisée. A charge donc pour le concepteur du système de s'assurer que l'implémentation fournie est bien indépendante du matériel et du logiciel utilisé (résolution et fréquence de l'écran, résolution et vitesse d'acquisition du périphérique de pointage, représentation des pixels...).

Comme montré dans nos expériences, cette indépendance de l'implémentation n'est pour l'instant pas assurée par la technologie disponible, pour les souris commercialisées actuellement dans tous les cas.

Critères :

- lors de la construction d'une technique d'interaction, il faut s'assurer que le système d'exploitation n'applique pas déjà une accélération ou une modification au curseur de la souris, impactant de fait les performances mesurées de la technique.
- après avoir contrôlé le système, il faut également contrôler le périphérique, afin de s'assurer que les pilotes constructeurs ne modifient pas le fonctionnement de la souris, au niveau matériel (vitesse d'échantillonnage, vitesse du bus), ou au niveau logiciel (courbes d'accélération propres au constructeur).
- le gain à faible vitesse va déterminer l'impression d'une souris "collante" pour l'utilisateur. Un gain trop faible va entraîner des mouvements trop vifs qui vont entraîner des dépassements de cibles, car la vitesse va amener des overshoots (voir le points suivant). La détermination du niveau de gain à faible vitesse doit prendre en compte la taille de l'écran (résolution et taille des pixels), pour s'assurer qu'aucune courbe n'interdise d'accéder à tous les pixels de l'écran à faible vitesse (et donc à faible gain).
- le gain à grande vitesse va déterminer *l'Overshoot*, et le nombre de clics hors cible à grande vitesse. L'utilisateur passe peu de temps dans cette gamme de gain, il est plus utile d'améliorer les performances aux gains les plus faibles, si un choix doit être fait. En effet, les utilisateurs ne dépassent pas 40 cm/sec en vitesse avec leur main, et sont 98% du temps inférieur à 20 cm/sec
- une courbe d'accélération est la résultante de l'interaction complexe entre plusieurs points pour créer son profil. Analyser chaque points indépendamment des autres sur les performances ne garantit pas que l'ensemble est le plus performant.
- il faut un nombre minimum de clics pour apprendre la technique d'accélération appliquée à une souris (10 dans nos expériences).
- les préférences de confort des utilisateurs ne se portent pas systématiquement sur les courbes leur assurant un faible taux d'erreurs, ou un parcours rapide.
- les performances établies lors d'expériences de pointage réciproque n'impliquent pas les mêmes performances lors de pointages discrets (ces premiers ne sont pas la simple concaténation de pointages discrets).

Le travail de bibliographie et l'étude de la physiologie du mouvement a permis de proposer une technique qui prolonge le travail du pointage. En effet, il est possible, en utilisant l'algorithme proposé ici, d'offrir à l'utilisateur la possibilité d'acquérir et de cliquer sur une cible lorsque le périphérique ne propose pas de bouton physique, ou que l'utilisateur n'est pas en mesure de le réaliser (dans le cadre d'un handicap par exemple).

Les performances sont bonnes en 3 dimensions, sur les grandes comme les petites cibles.

En deux dimensions, il faut concevoir cette technique différemment, car les utilisateurs sont habitués au fonctionnement de la souris, et cela a une influence sur leur approche de la technique.

Perspectives

La quantité de données récupérées lors du suivi de l'interaction d'un participant, ainsi que le grand nombre d'essais nécessaires induisent des critères de construction de courbe qui restent larges pour l'instant. La prise de conscience que lors d'un geste de pointage, les variables considérées pour évaluer le geste sont fortement corrélées devrait permettre de concevoir un protocole plus fin, prenant en compte l'intégralité du profil d'une courbe afin de raffiner ces critères de construction.

L'utilisation d'outils mathématiques plus puissants telles qu'une Analyse En Composantes Principales ou une recherche d'autocorrélation permettraient peut-être de mettre en évidence les parties du profil de pointage qui sont les plus importantes pour les performances. Dans ce cas, l'amélioration de la courbe d'accélération en regard des performances pourrait se concentrer sur ces segments là, et ainsi réduire l'espace de recherche.

Dans le cadre du handicap, il est possible, en définissant d'une autre manière les critères d'assistance au pointage, d'améliorer les performances de personnes handicapées. Ainsi, lors de troubles impactant le contrôle neuro-musculaire, l'utilisation de courbe de gains appropriées permettrait de réduire les mouvements erratiques et accidentels en fonction de leur vitesse, de l'angle ou de la distance du geste, pour quand même pouvoir acquérir la cible.

La construction de la courbe de gain peut également se faire sur d'autres critères que le taux d'erreurs ou le taux pour réaliser le mouvement. Ainsi, en situation de mobilité, un utilisateur peut désirer pouvoir utiliser sa souris ou son périphérique dans le peu d'espace qui lui est alloué. En prenant le critère d'une minimisation de l'empreinte spatiale du périphérique sur le plan de travail, nous pouvons construire une courbe d'accélération différente.

En ayant établi que le mouvement humain se compose de 2 phases, et une fois bien comprises les heuristiques pour construire une courbe de transfert, il peut être intéressant d'adapter la courbe à cette propriété physiologique. En introduisant la notion d'hystéresis dans la construction de la courbe, nous pouvons imaginer une courbe qui se comporte différemment lorsque l'utilisateur est en phase d'accélération ou en phase d'approche de la cible, ou il décélère.

Enfin, est-ce que le modèle se transpose facilement en 3 dimensions ?

Nombreuses sont les situations en Réalité Virtuelle qui demandent encore à l'utilisateur d'effectuer un geste identique en amplitude tant dans la réalité que dans l'environnement 3D. Dans ce contexte, les courbes d'accélération peuvent répondre à certains problèmes consécutifs à l'interaction 3D, tels que la fatigue musculaire, ou le manque de précision.

Annexe A

Filtrage de l'accélération et de la vitesse

Dans nos expérimentation, en considérant la faible résolution de notre système de suivi, le calcul direct de la vitesse et de l'accélération à partir des données brutes de position donne une information instable. A la place, nous avons utilisé le filtre passe-bas suivant pour calculer et filtrer la vitesse et l'accélération.

Nous avons utilisé un premier filtre passe-bas pour calculer et filtrer le vecteur vitesse $\mathbf{V_n^f}$ à partir de la position courante, non filtrée $\mathbf{P_n}$, de la position précédente $\mathbf{P_{n-1}}$ et du vecteur vitesse filtré à l'étape précédente $\mathbf{V_{n-1}^f}$ (Equation 2),τ est calculé en utilisant l'équation 1, où f_c est la fréquence de coupure du filtre, exprimé en hertz (dans notre expérience, nous avons utilisé une fréquence de coupure de 0,9 Hz), T_e est la période d'échantillonnage des données en secondes et est égale à l'inverse de la fréquence d'échantillonnage. L'accélération est calculé et filtré en utilisant une relation similaire (équation 3).

$$\tau = \frac{1}{2\,\pi\,f_c} \tag{1}$$

$$\mathbf{V_n^f} = \left(\frac{\mathbf{P_n} - \mathbf{P_{n-1}}}{T_e} + \frac{\tau}{T_e}\,\mathbf{V_{n-1}^f}\right)\frac{1}{1+\frac{\tau}{T_e}} \tag{2}$$

$$\mathbf{A_n^f} = \left(\frac{\mathbf{V_n^f} - \mathbf{V_{n-1}^f}}{T_e} + \frac{\tau}{T_e}\,\mathbf{A_{n-1}^f}\right)\frac{1}{1+\frac{\tau}{T_e}} \tag{3}$$

(*Publié dans [27]*)

Bibliographie

[1] Johnny Accot and Shumin Zhai. Beyond fitts' law : models for trajectory-based hci tasks. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 295–302. ACM, 1997.

[2] Johnny Accot and Shumin Zhai. More than dotting the i's—foundations for crossing-based interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems : Changing our world, changing ourselves*, pages 73–80. ACM, 2002.

[3] Golby C.W. 1 Kay H. Annet, J. The measurement of elements in an assembly task : The information output of the human motor system. *Quarterly Journal of Experimental Psychology*, 10 :1–11, 1958.

[4] Caroline Appert, Olivier Chapuis, and Michel Beaudouin-Lafon. Evaluation of pointing performance on screen edges. In *AVI '08 : Proceedings of the working conference on Advanced visual interfaces*, pages 119–126, New York, NY, USA, 2008. ACM.

[5] Tilley A.R. *The Measure Of Man And Woman. Human Factors In Design.* John Wiley & Sons, Inc., 2002.

[6] Takeshi Asano, Ehud Sharlin, Yoshifumi Kitamura, Kazuki Takashima, and Fumio Kishino. Predictive interaction using the delphian desktop. In *UIST '05 : Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 133–141, New York, NY, USA, 2005. ACM.

[7] Baudel T. Fitzmaurice G. Kurtenbach G. Balakrishnan, R. The rockin' mouse : Integral 3d manipulation on a plane. *Proc. of CHI'97*, 10 :311–318, 1997.

[8] Ravin Balakrishnan. "beating" fitts' law : virtual enhancements for pointing facilitation. *Int. J. Hum.-Comput. Stud.*, 61(6) :857–874, 2004.

[9] Ravin Balakrishnan and I Scott MacKenzie. Performance differences in the fingers, wrist, and forearm in computer input control. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 303–310. ACM, 1997.

[10] Patrick Baudisch, Edward Cutrell, Dan Robbins, Mary Czerwinski, Peter Tandler, Benjamin Bederson, Alex Zierlinger, et al. Drag-and-pop and drag-and-pick : Techniques for accessing remote screen content on touch-and pen-operated systems. In *Proceedings of INTERACT*, volume 3, pages 57–64, 2003.

[11] Patrick Baudisch, Alexander Zotov, Edward Cutrell, and Ken Hinckley. Starburst : a target expansion algorithm for non-uniform target distributions. In *AVI '08 : Proceedings of the working conference on Advanced visual interfaces*, pages 129–137, New York, NY, USA, 2008. ACM.

[12] Renaud Blanch. Pointage sémantique et distracteurs, la dynamique du pointage à la rescousse. In *Actes des secondes rencontres jeunes chercheurs en interaction homme-machine (RJC-IHM 2004)*, 2004.

[13] Renaud Blanch, Yves Guiard, and Michel Beaudouin-Lafon. Semantic pointing : improving target acquisition with control-display ratio adaptation. In *CHI '04 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 519–526, 2004.

[14] Renaud Blanch and Michaël Ortega. Rake cursor : improving pointing performance with concurrent input channels. In *CHI '09 : Proceedings of the 27th international conference on Human factors in computing systems*, pages 1415–1418, New York, NY, USA, 2009. ACM.

[15] Michael Bohan, Shelby G Thompson, and Peter J Samuelson. Kinematic analysis of mouse cursor positioning as a function of movement scale and joint set. In *Proceedings of the International Conference on Industrial Engineering–Theory, Applications and Practice*, pages 442–447. Citeseer, 2003.

[16] Richard A. Bolt. "put-that-there" : Voice and gesture at the graphics interface. In *SIGGRAPH '80 : Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, pages 262–270, New York, NY, USA, 1980. ACM.

[17] D. A. Bowman, C. A. Wingrave, J. M. Campbell, V. Q. Ly, and C. J. Rhoton. Novel uses of pinch gloves for virtual environment interaction techniques. *Virtual Reality*, 6(3) :122–129, 2002.

[18] W. Buxton. Lexical and pragmatic considerations of input structures. *Computer Graphics*, 17(1) :31–37, 1983.

[19] Xiang Cao, Nicolas Villar, and Shahram Izadi. Comparing user performance with single-finger, whole-hand, and hybrid pointing devices. In *Proceedings of the 28th international conference on Human factors in computing systems*, pages 1643–1646. ACM, 2010.

[20] Sébastien Carbini, Jean Emmanuel Viallet, and Lionel Delphin-Poulat. Context dependent interpretation of multimodal speech-pointing gesture interface. In *ICMI Doctoral Spotlight and Demo Papers*, pages 1–4. ACM, 2005.

[21] Stuart K. Card, Jock D. Mackinlay, and George G. Robertson. A morphological analysis of the design space of input devices. *ACM Trans. Inf. Syst.*, 9(2) :99–122, April 1991.

[22] Stuart K. Card, Allen Newell, and Thomas P. Moran. *The Psychology of Human-Computer Interaction*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 2000.

[23] L.G. Carlton. Processing visual feedback information for movement control. *Experimental Psychology : Human Perception and Performance*, 7 :1019–1030, 1981.

[24] Géry Casiez and Nicolas Roussel. No more bricolage ! : methods and tools to characterize, replicate and compare pointing transfer functions. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, UIST '11, pages 603–614, New York, NY, USA, 2011. ACM.

[25] Géry Casiez, Nicolas Roussel, Romuald Vanbelleghem, and Frédéric Giraud. Efficacité et robustesse aux distracteurs d'un retour tactile pour faciliter le pointage. In *Conference Internationale Francophone sur I'Interaction Homme-Machine*, IHM '10, pages 25–32, New York, NY, USA, 2010. ACM.

[26] Géry Casiez, Nicolas Roussel, Romuald Vanbelleghem, and Frédéric Giraud. Surfpad : riding towards targets on a squeeze film effect. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 2491–2500, New York, NY, USA, 2011. ACM.

[27] Géry Casiez, Nicolas Roussel, and Daniel Vogel. 1€ filter : A simple speed-based low-pass filter for noisy input in interactive systems. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, pages 2527–2530. ACM, 2012.

[28] Géry Casiez and Daniel Vogel. The effect of spring stiffness and control gain with an elastic rate control pointing device. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pages 1709–1718, New York, NY, USA, 2008. ACM.

[29] Géry Casiez, Daniel Vogel, Ravin Balakrishnan, and Andy Cockburn. The impact of control-display gain on user performance in pointing tasks. *Human-Computer Interaction, Taylor and Francis*, 23(3) :215–250, 2008.

[30] Géry Casiez, Daniel Vogel, Qing Pan, and Christophe Chaillou. Rubberedge : reducing clutching by combining position and rate control with elastic feedback. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*, UIST '07, pages 129–138, New York, NY, USA, 2007. ACM.

[31] Olivier Chapuis and Pierre Dragicevic. Small targets : Why are they so difficult to acquire ? Technical Report 1508, LRI, Univ. Paris-Sud, France, December 2008. 10 pages.

[32] Olivier Chapuis, Jean-Baptiste Labrune, and Emmanuel Pietriga. Dynaspot : speed-dependent area cursor. In *CHI '09 : Proceedings of the 27th international conference on Human factors in computing systems*, pages 1391–1400, New York, NY, USA, 2009. ACM.

[33] Olivier Chapuis and Nicolas Roussel. Uimarks : quick graphical interaction with specific targets. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, UIST '10, pages 173–182, New York, NY, USA, 2010. ACM.

[34] Ali Choumane, Gery Casiez, and Laurent Grisoni. Buttonless clicking : Intuitive select and pick-release through gesture analysis. In *Proceedings of the 2010 IEEE Virtual Reality Conference*, VR '10, pages 67–70, Washington, DC, USA, 2010. IEEE Computer Society.

[35] Firth A. Cockburn, A. Improving the acquisition of small targets. *People and Computers XVII : British Computer Society Conference on Human-Computer Interaction.*, 7 :181–196, 2003.

[36] Firth A. Cockburn, A. Improving the acquisition of small targets. In *Proceedings of the British HCI Conference*, pages 181–196, 2003.

[37] ERFW Crossman and PJ Goodeve. Feedback control of hand-movement and fitts' law. *The Quarterly Journal of Experimental Psychology*, 35(2) :251–278, 1983.

[38] A. Chemero D. G. Dotov, L. Nie. A demonstration of the transition from ready-to-hand to unready-to-hand. *PLoS ONE*, 5(3), 2010.

[39] Jean-Philippe Deblonde, Géry Casiez, and Laurent Grisoni. Méthode de détermination des fonctions de gains. In *Conference Internationale Francophone sur I'Interaction Homme-Machine*, IHM '10, pages 141–144, New York, NY, USA, 2010. ACM.

[40] E. Do. Drawing marks, acts and reacts : Toward a computational sketching interface for architectural design. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, (16) :149–171, 2002.

[41] Jacob Eisenstein and Wendy E. Mackay. Interacting with communication appliances : an evaluation of two computer vision-based selection techniques. In *CHI '06 : Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 1111–1114, New York, NY, USA, 2006. ACM.

[42] D.C. Engelbart. A conceptual framework for the augmentation of man's intellect. *Vistas in Information Handling*, 1 :1–29, 1963.

[43] Engelbart D.C. Berman M. English, W.K. Display selection techniques for text manipulation. *IEEE Transactions on Human-Factors in Electronics*, 8(1) :5–15, 1967.

[44] Laure Fernandez. *Organisation d'un geste de pointage de précision : Le couplage information-mouvement.* PhD thesis, 2004.

[45] Warren W.H. Fernandez, L. and R.J. Bootsma. Kinematic adaptation to sudden changes in task constraints during reciprocal aiming. *Human Movement Science*, 25 :695–717, 2006.

[46] P.M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47 :381–391, 1954.

[47] P.M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47 :381–391, 1954.

[48] P.M. Fitts. Information capacity of discrete motor responses. *Journal of Experimental Psychology*, 67 :103–112, 1964.

[49] F.N. Freeman. Experimental analysis of the writing movement. *Psychological Review Monographs Supplement*, pages 1–46, 1914.

[50] C.B. Gibbs. Controller design : Interactions of controlling limbs, time-lags, and gains in positional and velocity systems. *Ergonomics*, 5 :385–402, 1962.

[51] J.J. Gibson. Adaptation after-effect and contrast in perception of curved lines. *Journal of Experimental Psychology*, 16 :1–31, 1933.

[52] E. Graham. Virtual pointing on a computer display : non-linear control display-mappings. *Graphics Interface Conference*, pages 36–46, 1996.

[53] Tovi Grossman and Ravin Balakrishnan. The bubble cursor : enhancing target acquisition by dynamic resizing of the cursor's activation area. In *CHI '05 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 281–290, 2005.

[54] Beaudouin-Lafon M. Guiard, Y. and D. Mottet. Navigation as multiscale pointing : Extending fitts' model to very high precision tasks. *Proceedings of ACM CHI'99 Conference on Human Factors in Computing Systems*, pages 450–457, 1999.

[55] Y. Guiard. On fitts' and hooke's laws : Simple harmonic movement in upper-limb cyclical aiming. *Acta Psychologica*, 82 :139–159, 1993.

[56] Yves Guiard, Renaud Blanch, and Michel Beaudouin-Lafon. Object pointing : a complement to bitmap pointing in guis. In *GI '04 : Proceedings of Graphics Interface 2004*, pages 9–16, 2004.

[57] W.E. Hick. On the rate of gain of information. *Quarterly Journal of Experimental Psychology*, 4 :11–26, 1952.

[58] N. Hogan and T. Flash. The coordination of arm movements : an experimentally confirmed mathematical model. *Journal of Neuroscience*, 5 :1688–1703, 1985.

[59] Juan Pablo Hourcade, Christopher M. Nguyen, Keith B. Perry, and Natalie L. Denburg. Pointassist for older adults : analyzing sub-movement characteristics to aid in pointing tasks. In *CHI '10 : Proceedings of the 28th international conference on Human factors in computing systems*, pages 1115–1124, New York, NY, USA, 2010. ACM.

[60] Juan Pablo Hourcade, Keith B. Perry, and Aditya Sharma. Pointassist : helping four year olds point with ease. In *IDC '08 : Proceedings of the 7th international conference on Interaction design and children*, pages 202–209, New York, NY, USA, 2008. ACM.

[61] Amy Hurst, Jennifer Mankoff, Anind K. Dey, and Scott E. Hudson. Dirty desktops : using a patina of magnetic mouse dust to make common interactor targets easier to select. In *UIST '07 : Proceedings of the 20th annual ACM symposium on User interface software and technology*, pages 183–186, New York, NY, USA, 2007. ACM.

[62] R. Hyman. Stimulus information as a determinant of reaction time. *Journal of Experimental Psychology*, 45 :188–196, 1953.

[63] R. Jacob and L. Sibert. The perceptual structure of multidimensional input devices selection. *Proc. of CHI'92*, pages 211–218, 1992.

[64] R. J. Jagacinski and D. L. Monk. Fitts' law in two dimensions with hand and head movements. *Journal of Motor Behavior*, 17 :77–95, 1985.

[65] Repperger D.W. Moran M.S. Ward S.L. Jagacinski, R. J. and B. Glass. Fitts' law and the microstructure of rapide discrete movements. *Journal of Experimental Psychology : Human Perception and Performance*, 6 :309–320, 1980.

[66] M. Jeannerod. The timing of natural prehension movements. *Journal of Motor Behavior*, 16(3) :235–254, 1984.

[67] H.D. Jellinek and S.K. Card. Powermice and user performance. *ACM CHI Conference on Human Factors in Computing Systems*, pages 213–220, 1990.

[68] W.L. Jenkins and M.B. Connor. Some design factors in two dimensions with hand and head movements. *Journal of Motor Behavior*, 17 :77–79, 1949.

[69] Dropkin J. Johnson, P. and D. Rempel. Office ergonomics : Motion analysis of computer mouse usage. *Proceedings of the American Industrial Hygiene Association*, pages 12–13, 1993.

[70] Paul Kabbash and William AS Buxton. The "prince" technique : Fitts' law and selection using area cursors. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 273–279. ACM Press/Addison-Wesley Publishing Co., 1995.

[71] Posner M.I. Keele, S.W. Processing visual feedback in rapid movement. *Experimental Psychology*, 77 :155–158, 1968.

[72] S.W. Keele. Movement control in skilled motor performance. *Psychological Bulletin*, 70 :387–403, 1968.

[73] B.A. Kerr and G.D. Langolf. Speed of aiming movements. *Quarterly Journal of Experimental Psychology*, 29 :475–481, 1977.

[74] R. Kerr. Movement time in an underwater environment. *Journal of Motor Behavior*, 5 :475–481, 1973.

[75] M. Kobayashi and T. Igarashi. Ninja cursors : using multiple cursors to assis target acquisition on large screens. *Proc. ACM CHI'08*, pages 949–958, 2008.

[76] Per Ola Kristensson and Shumin Zhai. Command strokes with and without preview : using pen gestures on keyboard for command selection. In *CHI '07 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1137–1146, New York, NY, USA, 2007. ACM.

[77] Ferrigno G. Pedotti A. Soechting J.F. Lacquaniti, F. and C. Terzuolo. Changes in spatial scale in drawing and handwriting : Kinematic contributions by proximal and distal joints. *Journal of Neuroscience*, 7 :819–828, 1987.

[78] Terzuolo C. Viviani P. Lacquaniti, F. The law relating the kinematic and figural aspects of drawing movements. *Acta Psychologica*, 54(3) :115–130, 1983.

[79] Chafin D.B. Langolf, G.D. and J.A. Foulke. An investigation of fitts' law using a wide range of movements amplitudes. *Journal of Motor Behavior*, 8 :113–128, 1976.

[80] Edward Lank, Yi-Chun Nikko Cheng, and Jaime Ruiz. Endpoint prediction using motion kinematics. In *CHI '07 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 637–646, New York, NY, USA, 2007. ACM.

[81] Edward Lank, Eric Saund, and Luping May. Sloppy selection : Providing an accurate interpretation of imprecise selection gestures. *Computers and Graphics*, 29 :490–500, 2005.

[82] Ian Scott Mackenzie. *Fitts' law as a performance model in human-computer interaction.* PhD thesis, Toronto, Ont., Canada, Canada, 1991. UMI Order No. GAXNN-65985.

[83] I.S. MacKenzie. Movement time prediction in human-computer interfaces. *Proceedings of Graphics Interface '92.*, 7(1), 1992.

[84] M. McGuffin and R. Balakrishnan. Acquisition of expanding targets. In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems*, pages 57–64, 2004.

[85] Abrams R.A. Kornblum S. Wright C.E. Meyer, D.E. and J.E.K. Smith. Optimality in human motor performance : Ideal control of rapid aimed movements. *Psychological Review*, 95 :340–370, 1988.

[86] Smith J. E. K. Meyer, D.E. and C.E. Wright. Models for the speed and accuracy of aimed movements. *Psychological Review*, 89 :449–482, 1982.

[87] P. Morasso. Spatial control of arm movements. *Experimental Brain Research*, 42 :223–227, 1981.

[88] J. Müsseler and C. Sutter. Perceiving one's own movements when using a tool. *Consciousness and Cognition*, 18 :359–365, 2009.

[89] J. G. Phillips and T. J. Triggs. Characteristics of cursor trajectories controlled by the computer mouse. *Ergonomics*, 44 :527–536, 2001.

[90] Philip Quinn, Andy Cockburn, Géry Casiez, Nicolas Roussel, and Carl Gutwin. Exposing and understanding scrolling transfer functions. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, UIST '12, pages 341–350, New York, NY, USA, 2012. ACM.

[91] Woodworth R. The accuracy of voluntary movement. *Psychology Review*, 13(3), 1899.

[92] Harris C.S. Rock I. Vision and touch. *Scientific American*, 216(5) :96–104, 1967.

[93] Nicolas Roussel, Géry Casiez, Jonathan Aceituno, and Daniel Vogel. Giving a hand to the eyes : leveraging input accuracy for subpixel interaction. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, UIST '12, pages 351–358, New York, NY, USA, 2012. ACM.

[94] Jaime Ruiz and Edward Lank. Effects of target size and distance on kinematic endpoint prediction. Technical report, Technical Report CS-2009-25, University of Waterloo, 2009.

[95] Jaime Ruiz, David Tausky, Andrea Bunt, Edward Lank, and Richard Mann. Analyzing the kinematics of bivariate pointing. In *GI '08 : Proceedings of graphics interface 2008*, pages 251–258, Toronto, Ont., Canada, Canada, 2008. Canadian Information Processing Society.

[96] M. Kitajima M. Sato, S. and Y. Fukui. Proposal of a new mouse with realtime mickey ratio adjuster controlled by grasping force. *SICE2002*, 2002.

[97] Eric Saund and Edward Lank. Stylus input and editing without prior selection of mode. In *UIST '03 : Proceedings of the 16th annual ACM symposium on User interface software and technology*, pages 213–216, New York, NY, USA, 2003. ACM.

[98] Zelaznik H.N.n Hawkins B. Frank J.S. Schmidt, R.A. and Jr Quinn, J.T. Motor output variability : A theory for the accuracy of rapid motor acts. *Psychological Review*, 86 :415–451, 1979.

[99] C.E. Shannon and W. Weaver. The mathematical theory of communication. *University of Illinois Press*, 1949.

[100] K Stuart, K William, and J Betty. Evaluation of mouse, rate-controlled isometric joystick, step keys, and text keys for text selection on a crt. *Ergonomics*, 21(8) :601–613, 1978.

[101] U. Trankle and D. Deutschmann. Factors influencing speed and precision of cursor positioning using a mouse. *Ergonomics*, 34 :161–174, 1991.

[102] P. Viviani. Do units of motor action really exist ? *Generation and Modulation of Action Patterns.*, pages 201–216, 1986.

[103] P. Viviani and T. Flash. Minimum jerk, two-thirds power law and isochrony : Converging approaches to movement planning. *Journal of Experimental Human Perception and Performance*, (21) :32–53, 1995.

[104] Daniel Vogel and Ravin Balakrishnan. Distant freehand pointing and clicking on very large, high resolution displays. In *UIST '05 : Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 33–42, New York, NY, USA, 2005. ACM.

[105] Deric Wisleder and Natalia Dounskaia. The role of different submovement types during pointing to a target. *Experimental Brain Research*, 176 :132–149, 2007.

[106] Jacob O. Wobbrock, James Fogarty, Shih-Yen (Sean) Liu, Shunichi Kimuro, and Susumu Harada. The angle mouse : target-agnostic dynamic gain adjustment based on angular deviation. In *CHI '09 : Proceedings of the 27th international conference on Human factors in computing systems*, pages 1401–1410, 2009.

[107] Walker N. Bharat K. Worden, A. and S. Hudson. Making computers easier for older adults to use : area cursors and sticky icons. *Proc. CHI'97*, pages 266–271, 1997.

[108] Hawkins B. Zelaznik H.N. and Kisselburgh L. Rapid visual feedback processing in single-aiming movements. *Journal of Motor Behavior*, 15 :217–236, 1983.

[109] S. Zhai, P. Milgram, and W. Buxton. The influence of muscle groups on performance of multiple degree-of-freedom input. *Proceedings of CHI'96 Conference on Human Factors in Computing Systems*, pages 308–315, 1996.

[110] S. Zhai, C. Morimoto, and S. Ihde. Manual and gaze input cascaded (magic) pointing. *Proc. ACM CHI'99*, 15 :246–25, 1999.

[111] Shumin Zhai. *Human performance in six degree of freedom input control.* PhD thesis, University of Toronto, 1995.

[112] Shumin Zhai, Stéphane Conversy, Michel Beaudouin-Lafon, and Yves Guiard. Human on-line response to target expansion. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '03, pages 177–184, New York, NY, USA, 2003. ACM.