Agrandissement d'un écran mobile par réalité augmentée

par

Erwan NORMAND

MÉMOIRE PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
COMME EXIGENCE PARTIELLE À L'OBTENTION DE LA MAÎTRISE
AVEC MÉMOIRE EN GÉNIE, CONCENTRATION GÉNIE LOGICIEL
M. Sc. A.

MONTRÉAL, LE 30 AOÛT 2018

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE:

M. Michael J. McGuffin, Directeur de Mémoire
Département de génie logiciel et des TI à l'École de technologie supérieure

M. Roger Champagne, Président du Jury
Département de génie logiciel et des TI à l'École de technologie supérieure

M. David Labbé, Examinateur Externe
Département de génie logiciel et des TI à l'École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 29 AOÛT 2018

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

AVANT-PROPOS

2015 fut une année pleine de promesses et de changements. Après avoir terminé ma première expérience professionnelle en tant que développeur avec beaucoup de plaisir, à VideoStich (renommée Orah en 2016, https://www.orah.co/), start-up parisienne créant un logiciel de capture et d'édition de vidéo 360 en 4K, je terminais mon cursus ingénieur à l'Université de Technologie de Compiègne (https://www.utc.fr/) par quatre mois particulièrement intensifs et inspirants de cours, de projets et de contrats comme développeur indépendant. Immédiatement après, je m'envolai pour Montréal durant l'été pour une session d'échange à l'École de Technologie Supérieure. Ces quelques derniers mois de 2015 me permirent de prendre mes marques dans cette belle nouvelle vie québécoise. C'est dans un de mes cours de cette session que je fis la rencontre de mon futur directeur de recherche et que je pus alors commencer ma maîtrise en réalité augmentée début 2016.

Deux ans et quelques mois plus tard, je suis fier de pouvoir enfin présenter mon mémoire de cette maîtrise qui fut elle aussi pleine de défis, de rencontres et d'expériences.

REMERCIEMENTS

Je tiens tout d'abord à remercier mon directeur de maîtrise, Michael J. McGuffin, professeur à l'École de Technologie Supérieure (ÉTS), pour m'avoir laissé beaucoup de liberté dans le choix du sujet de ce mémoire, pour m'avoir guidé tout au long de ma maîtrise, pour avoir soutenu financièrement et matériellement ce projet, pour son aide lors de l'écriture de notre article scientifique et de ce mémoire, pour m'avoir permis d'avoir deux expérience professionnelles en RA à mi-temps en parallèle de cette maîtrise et enfin pour m'avoir permis de rencontrer de nombreux chercheurs à Montréal et à l'inspirante conférence UIST 2017.

Je remercie le Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG) pour avoir soutenu financièrement ce projet.

Je souhaite aussi remercier l'université de technologie de Compiègne, en particulier Annick Pourplanche, responsable des séjours à l'étranger, et Philippe Trigano, responsable du département de génie informatique, ainsi que les gouvernements français, québécois et canadiens pour m'avoir permis de réaliser ce diplôme et de vivre plus de deux ans dans un pays que je tenais beaucoup à découvrir.

Je remercie aussi le personnel du département de génie logiciel et des TI de l'ÉTS pour avoir répondu à mes nombreuses demandes d'aide technique.

Je remercie ma famille qui m'a toujours encouragé dans ce projet. Je remercie aussi mes amis, qu'ils viennent de Bretagne, de Compiègne, de Montréal ou d'ailleurs pour m'avoir soutenu et conseillé quand c'était nécessaire, je pense en particulier à Peet Vincenti et Audrey Bramy avec qui nous nous nous sommes accompagnés quelques temps sur nos maîtrises respectives à l'ÉTS. Je pense aussi à mes frères de Sigma Thêta Pi avec qui j'ai pu vivre et découvrir Montréal cette dernière année.

Surtout, je remercie Agathe Cestelli qui m'a soutenu et m'a attendu depuis la France : je suis particulièrement heureux de pouvoir enfin te retrouver...

AGRANDISSEMENT D'UN ÉCRAN MOBILE PAR RÉALITÉ AUGMENTÉE

Erwan NORMAND

RÉSUMÉ

Les récentes et importantes avancées dans la recherche et en industrie promettent à la réalité augmentée (RA) de s'intégrer profondément dans nos vies personnelles et professionnelles, malgré des limites technologiques encore existantes, pour augmenter nos perceptions du réel et les interactions avec nos environnements. Nous proposons d'utiliser la RA pour étendre l'écran d'un téléphone, au-delà de ses limites physiques, avec un écran virtuel coplanaire, synchronisé et suivant ses mouvements, créant la perception d'un unique grand écran étendu tenu en main. Nous appelons cette extension *Virtuality Extended Screen-Aligned Display* (VESAD). Nous décrivons les modes de vues possibles avec cette affichage : *vue multi-fenêtres*, supportant du multi-tâche, et *vue étendue*, où une application utilise tout l'écran étendu. Nous présentons aussi deux nouvelles techniques d'interaction : *wrist*, où l'utilisateur fait une rotation rapide du poignet pour substituer l'affichage avec un autre, et *slide-to-hang* pour détacher l'écran étendu en une fenêtre virtuelle séparée du téléphone. Nous décrivons ensuite la conception de notre propre visiocasque de RA à large champ de vision ($100° \times 98°$ pour chaque œil), permettant de voir complètement l'écran étendu, ainsi que le développement de notre bibliothèque libre de RA ArucoUnity, qui porte sur le moteur 3D Unity les fonctions de suivi de marqueurs et de calibration de caméra de la bibliothèque libre de vision par ordinateur OpenCV. Nous menons ensuite une étude expérimentale pour évaluer, d'une part, les avantages d'un téléphone à écran étendu par rapport à un téléphone non étendu et, d'autre part, pour comparer l'utilisation de l'écran tactile à des interactions avec la main sur l'écran virtuel. Nous répliquons une tâche de classement issue de la littérature sur les affichages muraux, impliquant de la navigation et des sélections. Nos résultats indiquent que la combinaison du téléphone à écran étendu avec les interactions tactiles s'est montrée la plus rapide, en particulier quand la tâche était maîtrisée des participants, et la plus performante en termes de navigation. Elle a également été préférée des participants. Néanmoins, plusieurs participants ont vu plus de potentiel dans les interactions avec la main virtuelle avec l'écran étendu. Nous donnons enfin des recommandations d'implémentation, de conception d'interface et de techniques d'interaction pour un VESAD.

Mots-clés: Réalité Augmentée, Interface Humain-Machine, Visiocasque, Écran tactile, Main virtuelle

EXTENSION OF A MOBILE SCREEN BY AUGMENTED REALITY

Erwan NORMAND

ABSTRACT

Significant and recent progress in research and in industry destined augmented reality to deeply integrate both in our personal and professional life. We propose using augmented reality to extend the screen of a smartphone beyond its physical limits with a virtual surface that is co-planar with the phone and that follows as the phone is moved. This creates the feeling of interaction with a unique big handheld screen. We call this extension a VESAD, or Virtually Extended Screen-Aligned Display. We illustrate first several ways that a VESAD could be used with two different viewing modes : *multi-windowed view*, to support multi-task, and *extended view* where an application uses the whole extended screen. We also present two novel interaction techniques : *wrist*, where the user performs a quick rotation of the phone to switch the information shown in the VESAD, and *slide-and-hang* whereby the user can detach a VESAD and leave it hanging in mid-air. We then describe our video-based augmented reality head-mounted display we developed with a large field of view ($100° \times 98°$ for each eye) and our free and open-source (FOSS) augmented reality library ArucoUnity. It ports the camera calibration and fiducial marker tracking functions from the FOSS computer-vision library OpenCV to the 3D engine Unity. We conducted an experiment that compared three interfaces used for an abstract classification task : the first using only a smartphone, the second using the phone for input but with a VESAD for output, and the third where the user performed input in mid-air on the VESAD (as detected by a Leap Motion). The second user interface was found to be superior in task completion time, furthermore with the last trials of the participants. It was also the most performant for navigation and was subjectively preferred over the other two interfaces. We finally report implementation, interfaces and interactions guidelines to create a VESAD.

Keywords: Augmented Reality, Human-Computer Interactions, Head-Mounted Display, Tactile Interactions, Mid-Air Interactions

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES EXTRAITS DE CODE

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

2D	Deux dimensions
3D	Trois dimensions
ANOVA	*Analysis of variance* (en français : analyse de variance)
CAVE	*Cave automatic virtual environment*
FoV	*Field of view* (en français : champ de vision)
DDL	Degré de liberté
DEL	Diode électroluminescente
DoF	*Degrees of freedom* (en français : degrés de liberté)
IC	Intervalle de confiance
FPS	*Frames per seconds* (en français : images par secondes)
HMD	*Head-mounted display* (en français : visiocasque)
IHM	Interface humain-machine
RA	Réalité augmentée
RV	Réalité virtuelle
RVB	Rouge, vert, bleu
VESAD	*Virtuality Extended Screen-Aligned Display* (en français : affichage virtuellement étendu)

INTRODUCTION

La réalité augmentée (RA) est une « technique [. . .] consistant à superposer en temps réel des images virtuelles [. . .] à des images issues du monde réel [. . .] ». (Office québécois de la langue française, 2017). Elle consiste à générer des objets virtuels en trois dimensions (3D) combinés avec l'environnement réel d'un utilisateur, donnant l'illusion que le virtuel coexiste avec le réel. Ainsi la RA permet d'*augmenter la perception* du réel et d'*augmenter les interactions* possibles d'un utilisateur avec son environnement (Azuma, 1997). La RA peut toucher tous les sens humains, mais est principalement utilisée pour le sens visuel.

a) Affichage d'un modèle 3D dans la pièce.

b) Les applications prennent généralement la forme de fenêtres virtuelles que l'utilisateur place sur un mur.

Figure 0.1 Illustrations publicitaires de démonstration du visiocasque de RA Microsoft HoloLens.
Tiré de Microsoft (2018).

Les deux précédentes années ont été très importantes pour la RA. Microsoft a tout d'abord apporté une innovation majeure, provoquant beaucoup d'intérêt, en proposant, en 2016, le HoloLens (https://www.microsoft.com/hololens). Ce visiocasque de RA portable a des performances bien supérieures à celles de ses concurrents (*Voir* Figure 0.1). Puis, en 2017, Google et Apple ont permis de concevoir des applications de RA sur les milliards de téléphones intelligents Android et iOS dans le monde avec leurs plateformes de développement ARCore (https://developers.google.com/ar/) et ARKit (https://developer.apple.com/arkit/). Couplé

aux annonces pour 2018 des visiocasques Magic Leap (https://www.magicleap.com/), North Star (https://developer.leapmotion.com/northstar/) et Meta 2 (https://www.metavision.com/), le marché de la RA est promis à devenir très important dans les années à venir.

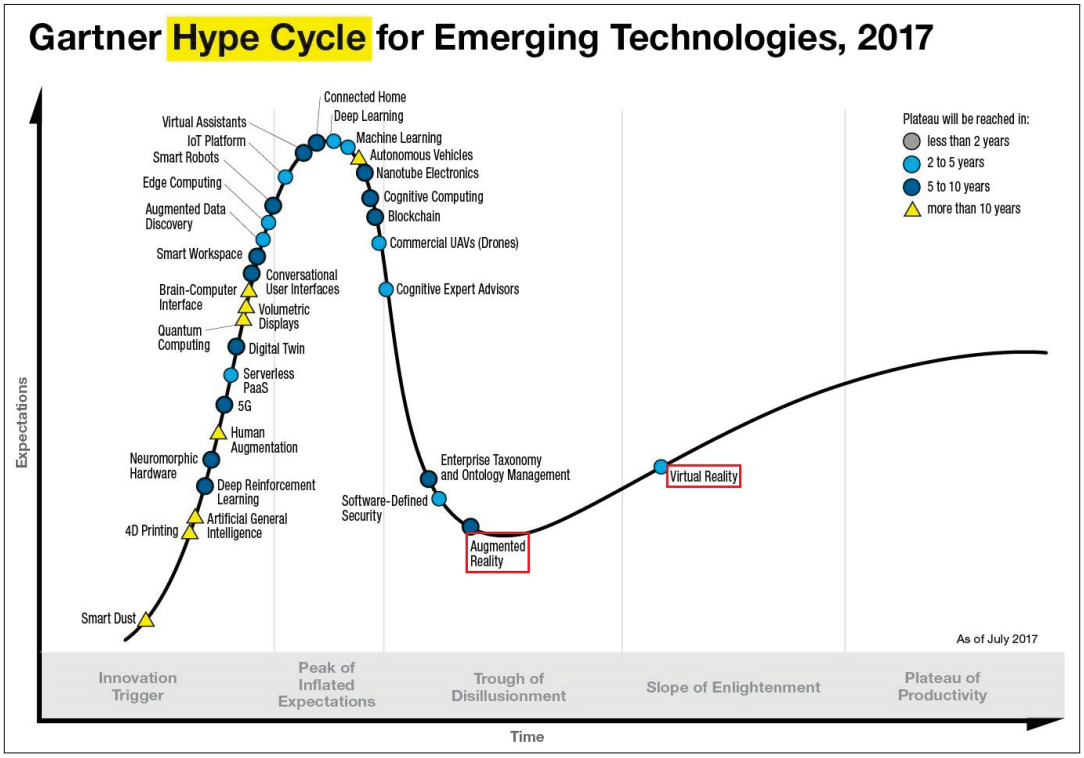
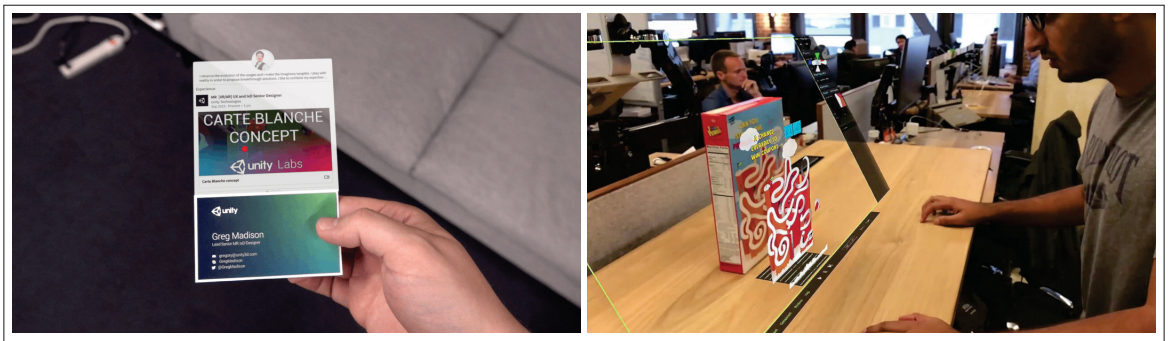


Figure 0.2 Courbe d'intérêt pour les nouvelles technologies, présenté en fonction du temps, en cinq phases. La RA est dans la phase trois du « gouffre des désillusions », la RV dans la phase quatre de la « pente de l'illumination ». Adapté de Gartner (2017).

La RA reste cependant une technologie encore peu connue et techniquement moins mature que la réalité virtuelle (RV). Elle est tout de même particulièrement prometteuse et va probablement profondément transformer nos quotidiens personnels et professionnels dans les cinq à dix prochaines années (*Voir* Figure 0.2), comme ont pu le faire les téléphones intelligents à la fin des années 2000 (Chaffey, 2018). La RA nous accompagnera en construisant des interfaces humain-machines (IHM) naturelles avec nos ordinateurs, téléphones et objets connectés. Cette vision est également partagée par des entreprises importantes dans les secteurs de la RV et la

RA, comme Unity : « We believe [augmented reality] should be human and object centric, and use the world as a device » (Gonzalez, 2017) qui ont proposé d'utiliser les visiocasques de RA pour étendre des cartes de visite (*Voir* Figure 0.3a), ou totalement remplacer l'ordinateur de bureau (*Voir* Figure 0.3b).



a) Extension par RA d'une carte de visite. b) Bureau de travail en RA.

Figure 0.3 Concepts d'IHMs de RA dans un environnement de bureaux.
Tiré de a) Gonzalez *et al.* (2017) et b) Gonzalez (2017).

Ainsi, comme pour les interfaces graphiques des ordinateurs dans les années 1990 ou celles des téléphones intelligents dans les années 2000, nous nous interrogeons sur la forme qu'aurait une IHM d'un système d'exploitation utilisant de la RA. En ce sens, nous pensons alors intéressant de combiner les téléphones intelligents avec des visiocasques de RA. Ces deux technologies sont en effet mobiles mais présentent des caractéristiques d'IHM différentes qui semblent complémentaires :

1. Les téléphones intelligents sont de plus en plus puissants et possèdent des écrans haute définition mais ont atteint une limite de taille physique d'écran pour pouvoir être tenus dans la main. Leurs utilisateurs pourraient bénéficier d'un plus grand écran pour utiliser plusieurs applications simultanément, de naviguer parmi de nombreux fichiers ou photos, ou encore de visualiser de grandes cartes ou des photos haute-résolution. De plus, le doigt est stabilisé sur la surface de l'écran, rendant les interactions tactiles précises, et demande de petits mouvements faciles à exécuter (Argelaguet & Andujar, 2013).

2. Les visiocasques de RA permettent d'entourer un utilisateur de nombreux et grands écrans virtuels (Ens *et al.*, 2014b). Il n'est pourtant pas encore clair quelles interfaces et techniques d'interactions sont les plus adaptées en RA (Piumsomboon *et al.*, 2013; Billinghurst *et al.*, 2015). Le pointage en 3D avec la main, par exemple, permet de pointer directement sur le contenu virtuel de manière similaire aux interactions avec des objets réels. Cependant, le manque de retour haptique de ces interfaces intangibles (Cha *et al.*, 2010), les mouvements complexes demandés au bras (Argelaguet & Andujar, 2013) ou encore les problèmes d'occlusion avec le contenu virtuel (Piumsomboon *et al.*, 2014) peuvent rendre cette technique difficile.

Figure 0.4 Photomontage d'une vue à la troisième personne de notre tâche expérimentale, dans la condition *VESAD* : une grille est affichée sur l'écran physique du téléphone et sur une fenêtre virtuelle située autour formant ainsi un unique écran étendu.

Nous proposons alors de concevoir un prototype de téléphone intelligent dont l'écran est étendu par RA. Nous appelons l'IHM de ce prototype *Virtuality Extended Screen-Aligned Display* (VESAD). Cela consiste en une fenêtre virtuelle affichée en RA étendant l'écran du téléphone. Elle est placée autour de l'écran du téléphone, coplanaire, synchronisée avec lui et le suivant dans ses mouvements. Cela donne ainsi à l'utilisateur le sentiment d'un seul grand écran

étendu tenu en main, permettant d'afficher beaucoup plus d'information en même temps (*Voir* Figure 0.4). Les interactions avec cette IHMs pourraient alors se faire sur l'écran tactile : elles seront alors précises, stables et déjà maîtrisées par les utilisateurs. Une autre alternative serait de manipuler directement la fenêtre virtuelle avec la main, ce qui pourrait être plus intuitif. On peut s'interroger alors si une technique d'interaction est à préférer. Ainsi, nous évaluons expérimentalement, dans ce travail de recherche, un VESAD appliqué à un téléphone intelligent comparé à un téléphone non étendu, ainsi que ces deux techniques d'interactions.

Après notre revue de littérature et la définition de notre problématique au chapitre 1, nous explorons le concept de notre prototype au chapitre 2, puis nous exposons son développement au chapitre 3. Nous menons ensuite une évaluation expérimentale comparant différents usages du VESAD face à un téléphone seul sur la tâche de classification de Liu *et al.* (2014) au chapitre 4 : nous y exposons sa conception ainsi que nos résultats. Enfin, nous discutons nos résultats et révisons notre concept au chapitre 5.

CHAPITRE 1

REVUE DE LITTÉRATURE

1.1 Historique et concepts de la RA

Sutherland (1968) conçoit le premier visiocasque de RA (*Voir* Figure 1.1) : ce prototype permet déjà de visualiser du contenu 3D affiché dans l'espace réel de l'utilisateur, donnant l'illusion que le contenu virtuel fait réellement partie de la pièce. Par la suite, la recherche académique en RA se développe lentement : les applications développées sont surtout à visées militaires et gouvernementales (Van Krevelen & Poelman, 2010). Il faut alors attendre les années 1990, avec la miniaturisation des PCs, pour que le domaine de recherche s'établisse enfin. Plusieurs conférences dédiées à la RA sont notamment créées, fusionnées aujourd'hui sous le nom d'International Symposium on Mixed and Augmented Reality (ISMAR), une conférence d'importance pour la recherche et l'industrie en RA (Azuma *et al.*, 2001).

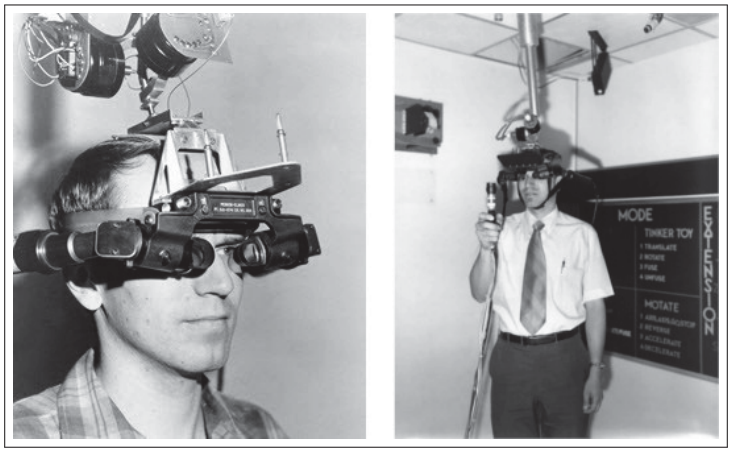


Figure 1.1 Photos du visiocasque de RA de Sutherland.
Tiré de Sutherland (1968).

Milgram & Kishino (1994) donne un premier cadre théorique au domaine, en proposant une échelle ordonnée nommée *Reality-Virtuality Continuum* (*Voir* Figure 1.2). Milgram & Kishino

y oppose deux extrêmes : les environnements réels et les environnements virtuels. Les IHMs graphiques, en deux dimensions (2D), sur ordinateurs et téléphones font partie de la première catégorie, tandis que les visiocasques de RV, qui immergent totalement leurs utilisateurs dans un monde virtuel en 3D, de la seconde. Entre ces deux extrêmes, les environnements de Réalité Mixte (RM), comme la RA, vont mélanger éléments réels et virtuels. La force de cette représentation est qu'il n'existe pas de catégories séparées entre réel, RA et RV mais que la RA peut se trouver d'un extrême à un autre le long de cette échelle.

Le second enseignement de l'échelle de Milgram & Kishino est que RA et RV sont techniquement très proches : les dispositifs de localisation, d'affichage et de génération de contenu sont les mêmes (Billinghurst *et al.*, 2015). Cependant, ces deux technologies n'ont pas les mêmes attentes. En effet, pour que l'immersion en RV fonctionne, il est nécessaire d'avoir un large champ de vision (*field of view* ou FoV en anglais) ; par exemple, le champ de vision du visiocasque de RV HTC Vive est de $100° \times 113°$ (horizontalement $\times$ verticalement) pour les deux yeux (Kreylos, 2016), ce qui est assez proche du champ de vision humain de $200° \times 120°$ pour les deux yeux (Cockburn *et al.*, 2008). La RA va en revanche demander tout d'abord une très grande précision et rapidité dans la localisation de l'utilisateur et des objets à augmenter pour donner le sentiment de « présence » du virtuel dans l'environnement réel.

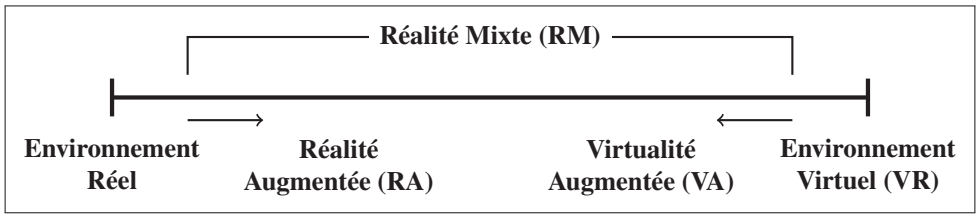


Figure 1.2 L'échelle du *Reality-Virtuality Continuum* de Milgram. Adapté de Milgram & Kishino (1994).

Rekimoto & Nagao (1995) apportent un second cadre théorique à la RA (*Voir* Figure 1.3). Ils montrent que les IHMs graphiques sont coupées des interactions avec l'environnement réel,

tandis que la RV isole l'utilisateur dans des IHMs totalement virtuelles. Deux alternatives à ces extrêmes sont la RA et l'informatique ubiquitaire (en anglais : *ubiquitous computing*). Cette dernière permet à l'utilisateur d'interagir avec des ordinateurs intégrés dans l'environnement réel, par exemple avec téléphones intelligents et des objets connectés, tandis que la RA fusionne réel et virtuel en un seul environnement pour l'utilisateur. Ainsi, avec une IHM bien faite, la RA s'intègre naturellement à l'environnement réel et devient invisible à l'utilisation.

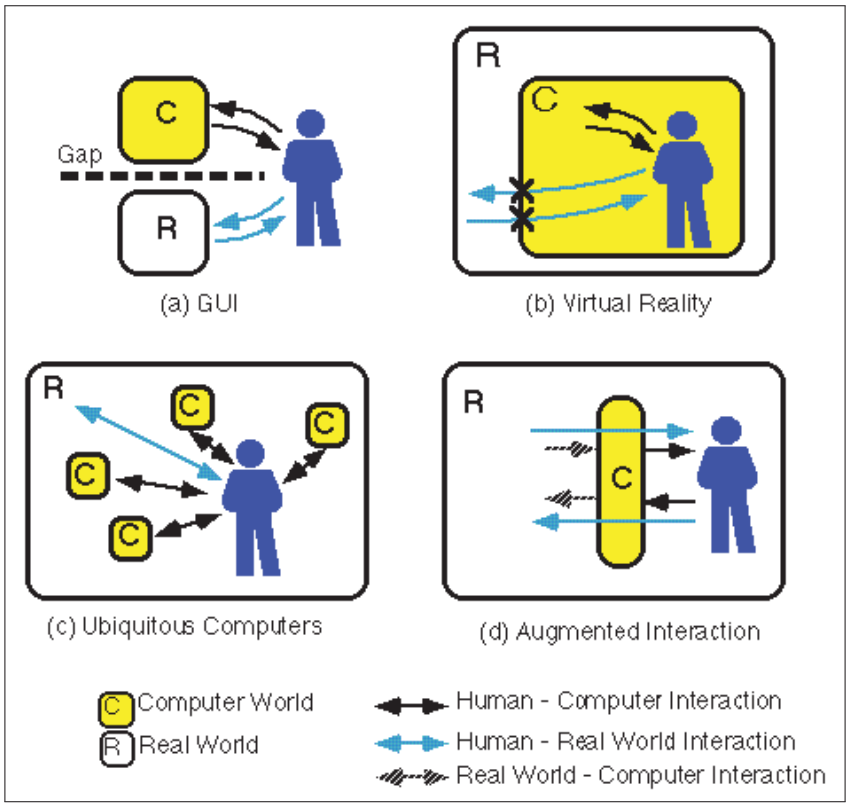


Figure 1.3 Comparaison de quatre styles d'IHMs : (a) les IHMs graphiques, coupées de l'environnement réel, (b) les IHMs en VR, isolant l'utilisateur dans un environnement virtuel, (c) l'informatique ubiquitaire, faite d'ordinateurs faisant partie intégrante de l'environnement réel et (d) les IHMs en RA faisant interface entre l'utilisateur et l'environnement réel.
Tiré de Rekimoto & Nagao (1995).

Une première définition formelle de la RA est par la suite proposée par Azuma (1997) dans le premier état de l'art du domaine. Ainsi, la RA :

1. combine des éléments réels et virtuels ;
2. est interactive en temps réel ;
3. aligne les éléments virtuels avec les éléments réels.

Ce sont les trois conditions techniques à respecter en RA qui vont donner le *sentiment de la présence du virtuel* dans l'environnement réel. Autrement dit, ces conditions vont permettre à notre cerveau de percevoir virtuel et réel comme un seul et même environnement. Cette définition a le mérite d'être assez générale pour s'appliquer tout autant à la RA visuelle, qu'au RAs auditives ou haptiques.

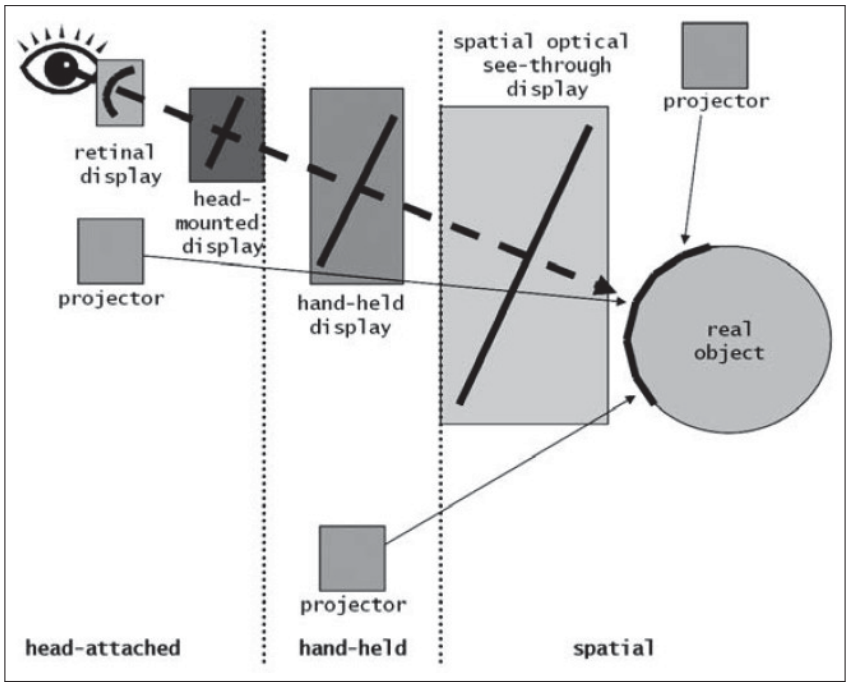


Figure 1.4 Différents types de dispositifs d'affichages en RA.
Tiré de Bimber & Raskar (2005).

Enfin, Buxton & Fitzmaurice (1998) repris par Bimber & Raskar (2005) catégorisent les différents dispositifs d'affichage en RA (*Voir* Figure 1.4). On retient essentiellement (nous excluons volontairement les projecteurs) :

- Le *cave automatic virtual environnement* (CAVE) : c'est un environnement immersif sous forme d'un cube de 3 m de côté, chaque face comportant un écran (*Voir* Figure 1.5). Les capteurs portés par l'utilisateur permettent au CAVE de suivre son mouvement et ainsi recalculer son champ de vision en temps réel. C'est un dispositif coûteux et encombrant, mais très utile pour prototyper des concepts.
- Les affichages fixes : la RA est affichée à travers un écran fixé dans l'environnement (*Voir* Figure 1.6).
- Les appareils mobiles : identique à un affichage fixe mais l'écran est tenu en main, comme une *fenêtre sur le contenu* en RA (*Voir* Figure 1.7). Ils sont populaires mais limités en taille et en puissance (Huang *et al.*, 2013).
- Les *head-mounted display* (HMD) ou visiocasques en français : ils sont portés sur la tête et projettent les images virtuelles directement aux yeux de l'utilisateur. Ils ont l'avantage de laisser les mains libres. On distingue deux technologies :
    - Les visiocasques vidéo : des caméras placées devant le casque filment l'environnement de l'utilisateur, les images capturées sont ensuite combinées avec le contenu virtuel puis affichées sur un écran (*Voir* Figure 1.8). Ils sont abordables et relativement facile à construire, mais la qualité de l'image est souvent faible (Steptoe, 2013; Piumsomboon *et al.*, 2014).
    - Les visiocasques optiques : le contenu virtuel est projeté sur un écran transparent par un système de miroirs. Ils sont difficiles à concevoir et proposent un petit champ de vision, mais une image de très bonne qualité. C'est l'approche la plus privilégiée par les visiocasques de l'industrie, comme le HoloLens ou le Meta 2.
- Les lentilles : elles sont identiques aux visiocasques mais posées directement sur les yeux. Peu utilisées, elles sont encore au stade de prototype mais, une fois maîtrisées, elles pourront être le dispositif de RA idéal (Van Krevelen & Poelman, 2010).

Figure 1.5 Un CAVE : un cube immersif d'écrans à taille humaine réagissant aux déplacements de l'utilisateur. Ici, l'utilisateur tape la main de son propre hologramme.
Tiré de Kreylos (2012).

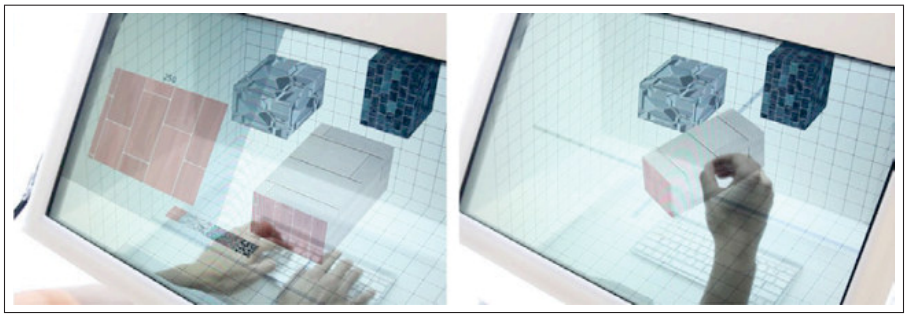


Figure 1.6 Photographies de SpaceTop : l'utilisateur peut déplacer des fenêtres dans espace 3D à travers un écran transparent en RA.
Tiré de Lee *et al.* (2013).

Ces dispositifs ont tous été explorés dans la littérature, mais ce sont essentiellement des prototypes sur appareils mobiles qui ont été développés : en effet, à partir des années 2000, les téléphones intelligents ont eu des caméras intégrées d'une qualité suffisante, tous les capteurs nécessaires et assez de puissance de calcul pour rendre la RA possible (Huang *et al.*, 2013).

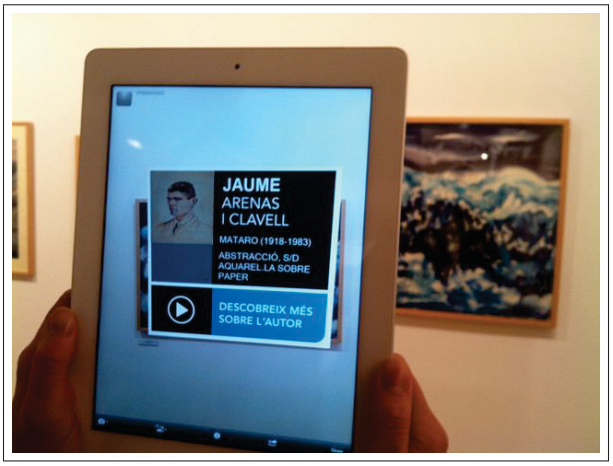


Figure 1.7 Une application sur tablette affichant un tableau de musée en RA.
Tiré de Kippelboy (2012).

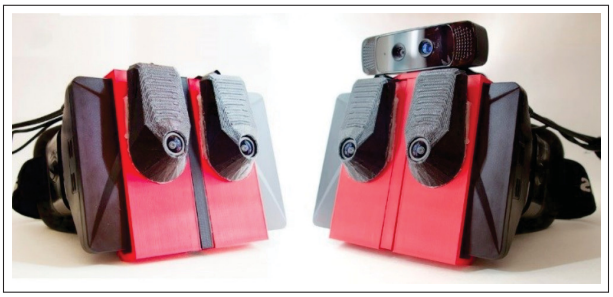


Figure 1.8 Photographie de l'AR-Rift, un visiocasque de RA vidéo dont la conception est documentée par Steptoe (2013) : les images filmées par deux caméras sont retransmises dans un visiocasque de RV.
Adapté de Piumsomboon *et al.* (2014).

Cependant, avec l'arrivée du HoloLens en 2016 les visiocasques vont pouvoir être à leur tour plus utilisés dans des recherches en RA.

Dans leur état de l'art, Azuma *et al.* (2001) identifient les trois obstacles à dépasser pour que la RA puisse être utilisable par le grand public : (1) les limites techniques, (2) les limites des IHMs et (3) les problèmes d'acceptation sociale (les visiocasques de RA comportent des caméras). Cependant, si quelques concepts et cadres théoriques pour la RA existent depuis plusieurs années, la recherche s'est malgré tout majoritairement consacrée à dépasser les limites tech-

niques de la RA, comme le souligne Zhou *et al.* (2008), Van Krevelen & Poelman (2010) et Billinghurst *et al.* (2015), dans leurs états de l'art respectifs. Ils indiquent également que trop peu de travaux ont été consacrés aux IHM et à l'expérience utilisateur en RA : « *there is a need to develop interface metaphors and interaction techniques specific to [augmented reality]* » (Billinghurst *et al.*, 2015).

1.2 Conception et évaluation d'IHMs de RA

1.2.1 IHMs en RA

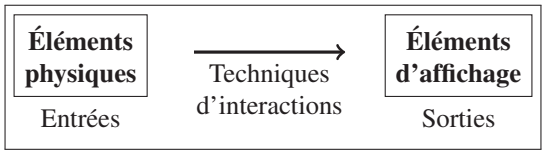


Figure 1.9 Principe d'une IHM : une technique interaction lie des entrées utilisateur, via un capteur physique, à un résultat affiché sur l'ordinateur. Adapté de Billinghurst *et al.* (2005).

Une IHM constitue la principale interface avec laquelle un utilisateur va interagir avec un ordinateur. La bonne conception de cet intermédiaire est donc nécessaire pour rendre accessible et efficace le travail sur une machine. Son rôle est simple : comme le montre la Figure 1.9, il s'agit de lier des *entrées utilisateurs* issues de capteurs physiques (souris, écran tactile, images d'une caméra) à des actions sur l'ordinateur représentées par un *résultat en sortie* (affichage, son, commande) via une *technique d'interaction* (Billinghurst *et al.*, 2005). La technique d'interaction est donc *une* méthode qui permet de traduire ces entrées en commandes : par exemple, le même mouvement avec une souris peut déplacer un curseur ou translater un objet le long d'un axe, ou encore un même déplacement de deux doigts sur un écran tactile peut faire une rotation ou un zoom sur un objet. Elle est également une métaphore qui permet à l'utilisateur d'associer ses actions avec des résultats sur l'ordinateur. Il existe de nombreux dispositifs d'entrée,

de sortie et de techniques d'interaction, le défi étant de : « *combine these together in a way that is most appropriate to the desired task, facilitates ease of use and learning and provides a high level of user performance and satisfaction* » (Billinghurst *et al.*, 2005).

Un des but de la recherche en IHM est de réduire cet écart entre les éléments physiques et d'affichages (Van Dam, 1997). Jacob *et al.* (2008) rappellent que les IHMs ont traversé plusieurs générations visant à chaque fois une utilisation plus simple, plus directe et plus proche des actions quotidiennes. Les ordinateurs s'utilisaient, avant les années 1960, par des cartes de commandes qui imprimaient alors leurs résultats. Puis ils ont été utilisés par ligne de commande via des terminaux jusque dans les années 1980. Enfin, les IHMs graphiques se sont démocratisées (*Voir* Figure 1.10), suivants le paradigme WIMP : un ensemble de fenêtres (*windows*), d'icônes (*icons*) et de menus (*menus*) accessibles par un appareil de pointage (*pointing device*), comme une souris. Ces IHMs sont excellentes pour des applications graphiques en 2D, mais limitées pour celles en 3D, entre autres [1]. Van Dam (1997) appela alors à développer une nouvelle génération d'IHMs post-WIMP, c'est-à-dire contenant des techniques d'interactions non dépendantes de fenêtre, icônes ou de menu en 2D. L'objectif étant de diversifier les types d'IHMs pour proposer le plus adapté selon les besoins de l'utilisateur.

Si de nombreuses IHMs post-WIMP ont par la suite été développées, comme celles désormais bien connues des téléphones intelligents à écrans tactiles (Jacob *et al.*, 2008), aucun paradigme n'est encore établi pour la RA (Van Krevelen & Poelman, 2010). Billinghurst *et al.* (2015) classent celles de RA en plusieurs catégories :

- Navigateurs d'informations : affichant de l'information en RA sur l'environnement réel. Ce sont principalement les IHMs que l'on retrouve dans les applications pour téléphones intelligents (*Voir* Figure 1.7) : elles permettent simplement d'afficher du contenu en sur-

1. Les IHMs en ligne de commande sont toujours utilisées et complémentaires des IHMs graphiques : elles peuvent être plus puissantes et mieux adaptées que ces dernières, même si elles sont plus difficiles d'accès par le temps d'apprentissage qu'elles demandent.

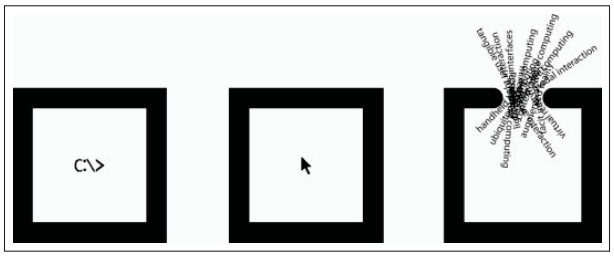


Figure 1.10 Illustrations des deuxième (en ligne de commande) et troisième (par interface graphique WIMP) générations d'IHMs, ainsi que les nombreux types émergeants d'IHMs post-WIMP. Toutes sont complémentaires, chacune adaptée à des usages spécifiques qu'étudie la recherche sur les IHMs. Tiré de Jacob *et al.* (2008).

impression de l'environnement réel, l'utilisateur se déplaçant simplement pour le voir sous différents angles. Des IHMs WIMP sont souvent utilisées, mais ce type d'interfaces est limité dans les interactions avec le contenu virtuel en 3D.

- Interfaces en 3D : utilisant des techniques d'interactions en 3D pour manipuler le contenu dans l'espace.
- Interfaces tangibles : utilisant des objets réels pour interagir avec du contenu virtuel.
- Interfaces naturelles : utilisant les mouvements du corps de l'utilisateur, par exemple des commandes gestuelles.
- Interfaces multimodales : utilisant simultanément des commandes vocales et gestuelles.

Pour Billinghurst *et al.* (2015), toutes ces IHMs importent et appliquent des techniques d'interactions WIMP, de la RV ou des écrans tactiles. En ce sens, elles atteignent seulement les première et deuxième étapes du processus de Billinghurst *et al.* (2005) qui décrit les étapes de développement d'un paradigme d'IHM pour un nouveau média :

1. développement de prototypes ;
2. adoption de techniques d'interactions d'autres types d'IHM ;
3. développement de métaphores et paradigmes d'IHMs adaptés au média ;

4. développement de modèles théoriques pour ces paradigmes.

La RA présente pourtant un excellent média, car l'écart entre les éléments physiques et d'affichage (le contenu virtuel) y est très réduit. Cela permettrait de fondre la présence de l'ordinateur dans l'environnement de l'utilisateur sans le restreindre sur un écran. Une bonne IHM de RA permettrait alors de manipuler naturellement et efficacement du contenu virtuel en 3D avec des techniques d'interactions basées sur des objets physiques, des commandes vocales ou des gestes utilisés au quotidien (Billinghurst *et al.*, 2005). C'est pourquoi il est important de consacrer un effort de recherche pour atteindre cette troisième étape, comme nous souhaitons le faire avec notre concept de téléphone à écran étendu, en utilisant des IHMs en 3D, les IHMs naturelles et les IHMs tangibles. Enfin, on remarque que la classification des IHMs de RA de Billinghurst *et al.* (2015) se concentre quasi exclusivement sur les techniques d'interactions. Pourtant, les données affichées ont toute leur importance, et il s'agit également de dépasser les concepts de fenêtre, menu et icônes des IHMs WIMP.

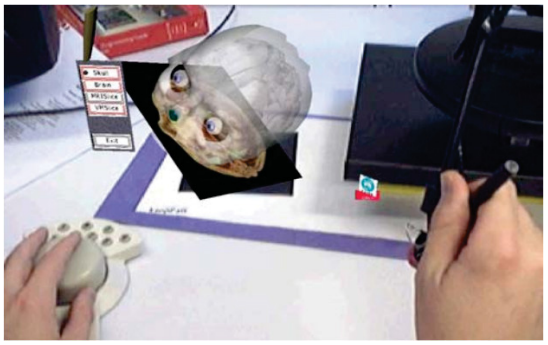
1.2.2 IHMs en 3D et IHMs naturelles

Les IHMs en 3D sont importées des recherches de la RV et dans les environnements virtuels (EV) en 3D. La recherche sur les EVs explore les interfaces et les interactions avec du contenu en 3D affiché sur un écran (et non de manière immersive comme en RA). Bowman *et al.* (2004) classent les techniques d'interactions en trois catégories : (1) navigation, (2) sélection (un ou plusieurs objets) et (3) manipulation (translation et rotation). Seules ces deux dernières sont applicables à la RA, le contenu virtuel étant aligné et intégré avec l'environnement réel, l'utilisateur se déplace simplement lui-même pour y naviguer.

De nombreux dispositifs d'entrée ont été développés pour les IHMs en 3D : des souris 3D, des joysticks, des stylos suivis en 3D ou des dispositifs haptiques (avec retour de force, donnant l'illusion de toucher le contenu virtuel). Tous permettent de sélectionner et manipuler avec

précision du contenu virtuel ; on parle alors de degrés de libertés (*degrees of freedom* ou DoF en anglais) pour caractériser les manipulations possibles, avec un degré par dimension donc maximum trois pour la translation et trois pour la rotation : une souris standard permet donc 2 DoFs, quand les dispositifs d'entrées des visiocasques de RV sont suivis avec 6 DoFs (*Voir* Figure 1.11a). Cependant, tous ces dispositifs présentent l'inconvénient d'occuper les mains de l'utilisateur : en RA, les espaces de visualisation et d'interaction sont donc séparés (*Voir* Figure 1.11b) et les techniques d'interactions avec les objets virtuels sont alors différentes de celles avec les objets réels (Billinghurst *et al.*, 2015).

a) Les dispositifs d'entrée du visiocasque de RV Oculus Rift : sans fils, ils permettent de sélectionner du contenu virtuel et de le manipuler avec 6 DoFs.

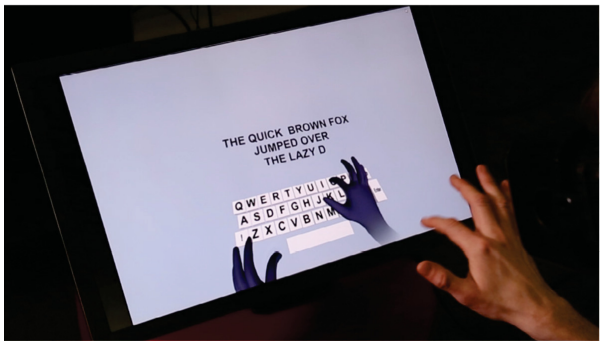


b) IHM de RA où la main gauche utilise plusieurs boutons pour interagir avec le menu 2D, tandis que la main droite tient un stylo haptique pour manipuler le crâne en 3D.

Figure 1.11 Exemples de dispositifs d'entrées pour IHMS en 3D.
Tiré de a) Evan-Amos (2017) et b) Adcock *et al.* (2003).

Les IHMs naturelles utilisent quant à elle directement les mouvements du corps de l'utilisateur comme dispositif d'entrée, en particulier les mains. De nombreuses méthodes de vision par ordinateur permettent ce suivi en utilisant simplement les images d'une caméra (Billinghurst *et al.*, 2015). Un utilisateur peut alors en théorie agir directement avec du contenu virtuel comme il le ferait avec des objets réels. Malgré tout, de telles interactions demandent souvent des mouvements complexes et fatigants à l'utilisateur (Bowman *et al.*, 2001). De plus, ces IHMs ne peuvent répliquer exactement le monde réel : le manque de contraintes physiques rend

les interactions imprécises (Cha *et al.*, 2010) et il peut être difficile de comprendre comment interagir avec un environnement virtuel utilisant une IHM naturelle (Argelaguet & Andujar, 2013).



a) Main virtuelle. b) Pointeur virtuel.

Figure 1.12 Techniques d'interactions de sélection dans les IHMs naturelles.
Tiré de a) Taylor *et al.* (2016) et b) Pfeuffer *et al.* (2017).

Un écart existant donc toujours entre éléments physiques et éléments graphiques, des techniques d'interactions sont encore nécessaires. Deux types de techniques sont majoritairement utilisées. La *main virtuelle* affiche un modèle 3D de main pouvant interagir avec le contenu virtuel et suivant la main réelle de l'utilisateur (*Voir* Figure 1.12a). Elle est la plus naturelle, mais souffre particulièrement des problématiques que nous venons d'énoncer, mais aussi de problème d'occlusion, car elle peut cacher des objets virtuels. La technique du *pointeur virtuel* permet à l'inverse de projeter un rayon, depuis la main ou la tête, pour sélectionner des objets (*Voir* Figure 1.12b) : il permet d'atteindre facilement des objets lointains avec peu d'effort, mais cela est difficile s'ils sont de trop petite taille. De plus, si l'action de sélection peut entraîner un déplacement involontaire du pointeur, par exemple en pressant un bouton (Argelaguet & Andujar, 2013). Cette technique a notamment été choisie pour le HoloLens : le pointeur suit les mouvements de la tête, stables et précis (Kytö *et al.*, 2018), tandis que la sélection se fait avec un geste de la main pour ne pas perturber le pointage.

La difficulté pour ce type d'IHM est donc de permettre des sélections et manipulations rapides, sans erreur ainsi que d'être faciles à comprendre et peu fatigantes (Argelaguet & Andujar, 2013). Pour cela, il est crucial de fournir des retours à l'utilisateur : Cha *et al.* (2010) proposent la combinaison de retours continus, pour que l'utilisateur situe le suivi de son corps, et de retours discrets pour confirmer ses actions. Le modèle 3D de la main virtuelle avec des ombres portées ou le rayon du pointeur virtuel sont donc des retours visuels continus, tandis qu'un bref changement de couleur de l'objet peut confirmer une sélection. Ces retours ne doivent cependant pas être excessifs, au risque de produire une distraction et réduire la performance des utilisateurs (Argelaguet & Andujar, 2013).

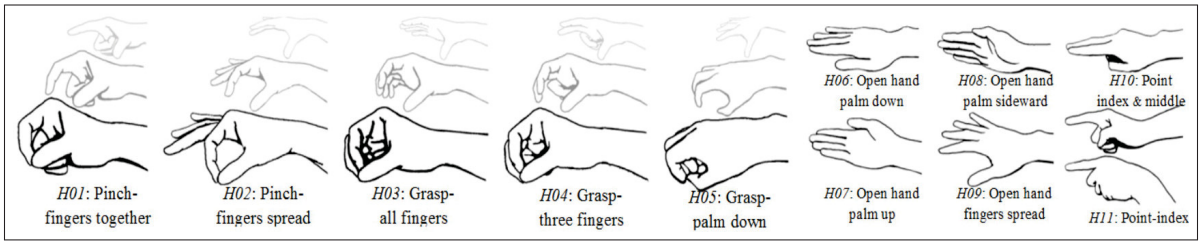


Figure 1.13 Les onze poses de mains utilisées pour des gestes en RA.
Tiré de Piumsomboon *et al.* (2013).

Si l'utilisation du pointeur virtuel telle qu'elle est faite par le HoloLens fonctionne bien (Kytö *et al.*, 2018), ce n'est pas le cas pour la main virtuelle dans les IHMs de RA malgré de nombreux prototypes (Piumsomboon *et al.*, 2013). Piumsomboon *et al.* (2013) ont alors établi de manière rigoureuse une liste de gestes utilisables avec une main virtuelle en RA. Pour cela, ils ont tout d'abord montré des animations de quarante tâches courantes dans la littérature dans un visiocasque de RA à vingt participants et leur ont demandé de mimer en même temps que chaque animation le geste qu'ils utiliseraient. Ils ont alors sélectionné les poses de main les plus utilisées (*Voir* Figure 1.13), ainsi que le ou les gestes majoritaires avec un score de consensus pour chaque tâche (*Voir* Figure 1.14). Beaucoup de tâches ont un faible consensus (peu de participants ont utilisé le même geste), néanmoins ceux pour les menus ont un haut score

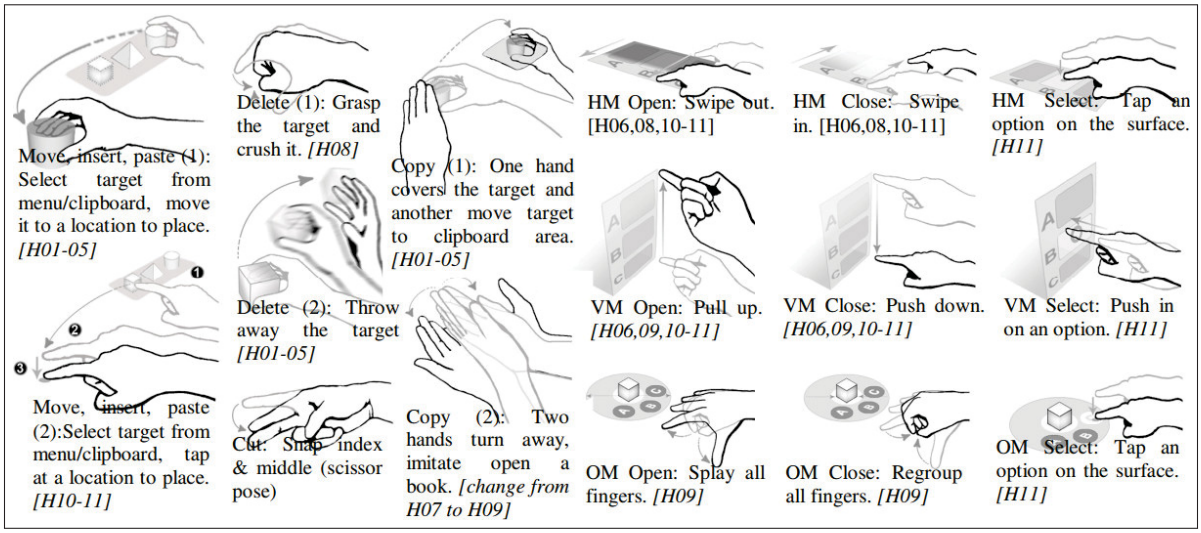


Figure 1.14 Extrait de la liste de gestes pour différentes tâches en RA. Les positions de mains utilisables sont indiquées entre crochets.
Tiré de Piumsomboon *et al.* (2013).

$\geq 80\%$ (*Voir* Figure 1.15). Enfin,la majorité de ces gestes sont réalisés « en l'air », à l'exception de ceux ouvrant, fermant et sélectionnant un menu horizontal qui ont été réalisés sur la surface de la table (*Voir* Figure 1.14).

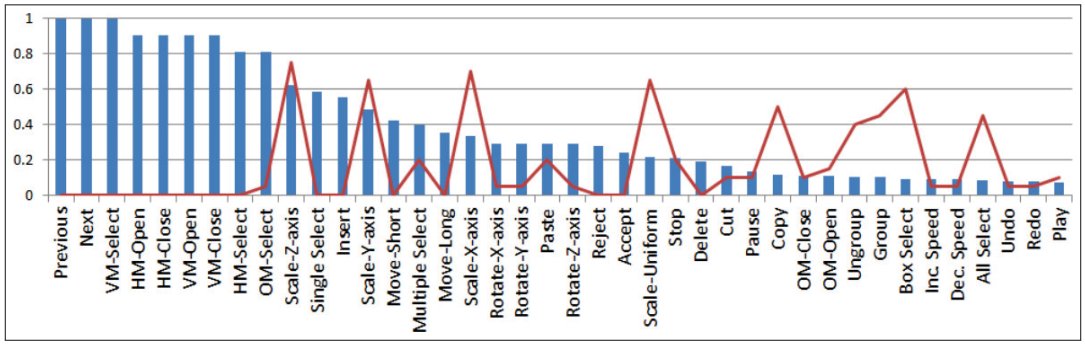
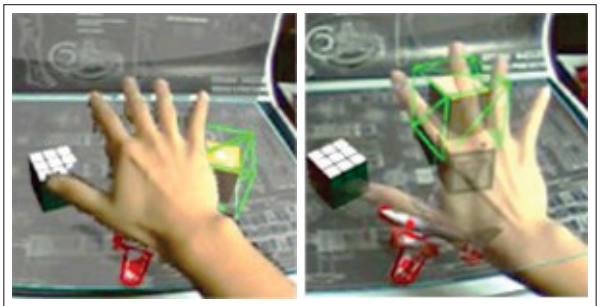


Figure 1.15 Score de consensus sur le geste à utiliser (barres bleues) ainsi que ratio de gestes à deux mains (ligne rouge) parmi les participants pour chaque tâche.
Tiré de Piumsomboon *et al.* (2013).

Par la suite, Piumsomboon *et al.* (2014) ont comparé formellement quelques-unes de ces techniques avec des commandes vocales pour la sélection et la manipulation d'objets en 3D dans

un visiocasque de RA. Leurs résultats indiquent que les participants étaient globalement plus rapides et préféraient utiliser des gestes pour manipuler les objets, mais des commandes vocales pour les redimensionner. Il serait cependant intéressant de reproduire cette étude avec des IHMs plus complexes que de simples objets 3D, par exemple avec des IHMs WIMP appliquées à la RA, comme le fait le HoloLens.



a) La main virtuelle fait occlusion avec le contenu 3D.

b) Le contenu 3D est transparent, laissant visible la main de l'utilisateur.

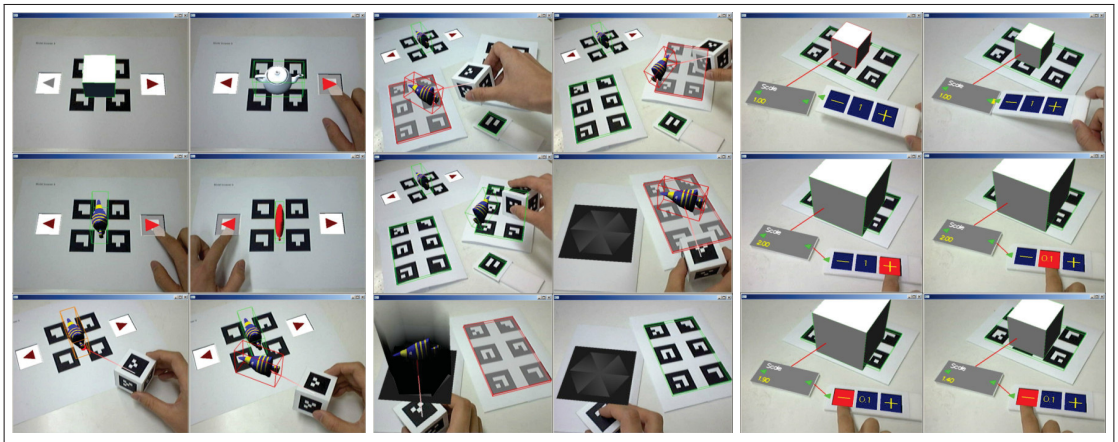
Figure 1.16 Différentes techniques d'occlusion de la main.
Tiré de Piumsomboon *et al.* (2014).

Enfin, Piumsomboon *et al.* (2013) et Piumsomboon *et al.* (2014) recommandent, à la suite d'études pilotes, que pour éviter les problèmes d'occlusion, les objets virtuels doivent toujours êtres visibles : soit en utilisant une main virtuelle transparente plutôt qu'opaque, soit en affichant leurs contours si elle les cache (*Voir* Figure 1.16).

1.2.3 IHMs de RA tangibles

Les IHMs de RA tangibles utilisent des objets réels comme dispositifs d'entrées et d'affichages, qui sont alors augmentés par RA (Kato *et al.*, 2000). Autrement dit, chaque objet virtuel est aligné avec un objet réel et est manipulé via cet objet réel (Billinghurst *et al.*, 2005) : les espaces d'interactions et d'affichage sont donc alignés et confondus. Ainsi, les interactions sont familières, faciles et intuitives et peuvent s'appuyer sur les contraintes physiques de ces

objets réels (Zhou *et al.*, 2008). Le principal inconvénient de ces IHM est de requérir des objets physiques, qui peuvent ne pas être présents dans l'environnement de l'utilisateur ou difficiles à transporter dans un usage mobile.



a) L'utilisateur presse les boutons du navigateur, puis créé l'objet courant en le pointant avec le cube de manipulation.

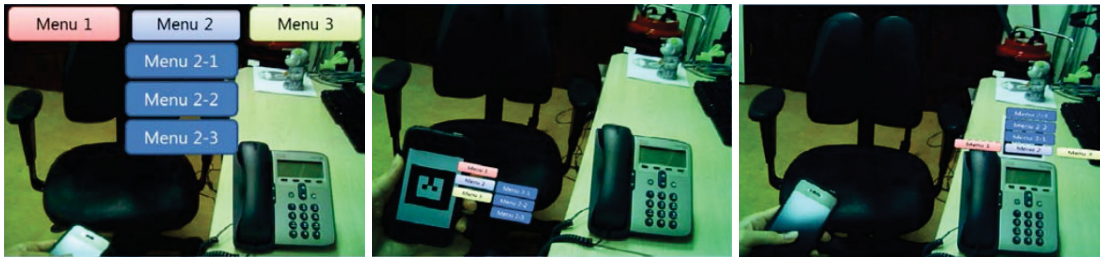
b) L'utilisateur déplace l'objet crée avec le cube, puis le fixe en cachant le haut du cube. Il le place dans la corbeille pour le détruire.

c) Redimensionnement d'un cube à l'aide d'un clavier connecté à un outil inspecteur affichant la propriété de taille du cube.

Figure 1.17 Exemple d'une IHM de RA tangible : les objets virtuels sont alignés avec des objets réels suivis avec 6 DoF et manipulés via des objets virtuels spécifiques utilisés comme outils.
Tiré de Lee *et al.* (2004).

Il existe toujours un écart entre entrées et affichages pour ce type d'IHM, en particulier pour certaines commandes : par exemple, pour changer les dimensions de l'objet virtuel ou, de manière générale, ses propriétés, l'activer ou encore réaliser des opérations de copie. Des techniques d'interactions sont donc toujours nécessaires, la difficulté étant de faire comprendre à l'utilisateur les commandes possibles et les conséquences de ces actions (Zhou *et al.*, 2008). Lee *et al.* (2004) proposent par exemple d'utiliser certains objets réels comme des outils pour créer, détruire les autres objets virtuels ainsi que visualiser et modifier leurs propriétés : l'utilisateur place les outils proches des objets pour les faires interagir puis passe son doigt dessus comme pour effectuer un clic pour les activer (*Voir* Figure 1.17).

Il existe deux approches pour concevoir une IHM de RA tangible : une seule fonction est attribuée à chaque outil dans une IHM à *multiplexage spatial*, tandis qu'un même outil est utilisé pour plusieurs fonctions selon le contexte dans une IHM à *multiplexage temporel* Billinghurst *et al.* (2015). Le multiplexage spatial est plus facile à apprendre, mais le multiplexage temporel permet plus de flexibilité dans la conception de l'IHM. Par exemple, le cube de manipulation dans l'IHM de Lee *et al.* (2004) est à multiplexage spatial, car il a une seule fonction, tandis que l'outil inspecteur est à multiplexage temporel, car il affiche des données différentes en fonction de l'objet et de la propriété inspectés.



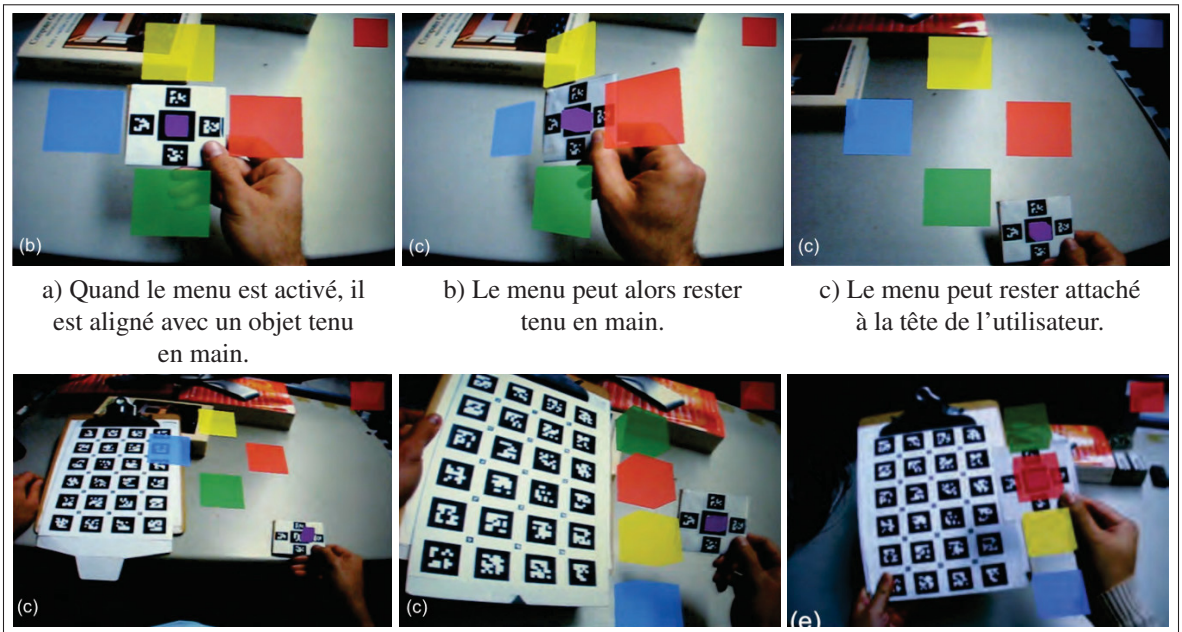
a) Navigateur d'information : le menu apparaît à distance fixe de l'utilisateur.

b) Tangible et aligné avec l'écran d'un téléphone tenu en main.

c) Tangible et aligné avec un objet de l'environnement.

Figure 1.18 Affichage d'un même menu selon différents types d'IHM de RA. Tiré de Lee *et al.* (2011).

Un téléphone peut donc être utilisé comme base d'IHM de RA tangible, par exemple pour afficher des menus alignés avec l'écran (*Voir* Figure 1.18). White *et al.* (2009) ont comparé l'usage d'un tel menu aligné avec d'autres placements possibles, par rapport à la tête de l'utilisateur ou par rapport à l'environnement (*Voir* Figure 1.19). L'utilisateur sélectionne un élément du menu en le pointant avec l'objet tenu en main. Dans les résultats de leur étude expérimentale, où treize participants ont effectué chacun 80 sélections avec chaque type de menu, leurs résultats indiquent que les sélections ont été statistiquement plus rapides dans les cas du menu tenu en main ou attaché à la tête de l'utilisateur (en moyenne en 3 s contre 7 s à 8 s). Les utilisateurs ont également préféré ces deux techniques. Pour les auteurs, ces deux conditions étaient les plus



a) Quand le menu est activé, il est aligné avec un objet tenu en main.

b) Le menu peut alors rester tenu en main.

c) Le menu peut rester attaché à la tête de l'utilisateur.

d) Le menu peut rester fixe dans l'espace.

e) Le menu peut être attaché à une planche : la sélection est alors bi-manuelle.

f) La sélection d'un élément du menu se fait en le pointant avec l'objet tenu en main.

Figure 1.19 Différents placements possibles d'un menu de RA.
Tiré de White *et al.* (2009).

stables et les plus faciles à utiliser. Pourtant le menu attaché à la main est plus intéressant car l'utilisateur peut facilement le déplacer ou changer son orientation. Cette étude supporte donc la faisabilité d'une IHM de RA tangible basée sur un téléphone tenu en main.

1.2.4 Évaluation d'IHM en RA

Pour déterminer si une IHM est fonctionnelle et pertinente, on peut l'évaluer avec des études utilisateurs (Billinghurst *et al.*, 2015). Cela permet de mesurer efficacement l'utilisabilité de l'interface, les résultats obtenus permettent en outre de donner des clés de conception à d'autres chercheurs dans le domaine ainsi qu'à des concepteurs et développeurs (Jankowski & Hachet, 2015). Enfin, correctement décrites, ces études peuvent également être répliquées et leurs résultats vérifiés et discutés indépendamment.

Concrètement, plusieurs personnes sont recrutées pour utiliser différents systèmes (par exemple des IHMs) de RA dans différentes configurations sur une même tâche. L'objectif est alors d'évaluer et comparer ces systèmes et configurations, et non pas les utilisateurs. On cherche alors à répondre à des questions sur ces systèmes et leur usage ; on conçoit alors la tâche et des mesures pour obtenir des données pour répondre à ces questions. Swan II & Gabbard (2005) et Dünser *et al.* (2008) catégorisent les études utilisateurs en RA selon les questions qu'elles visent : (1) perception et cognition humaine, (2) interactions et (3) communication et collaboration. Seule une minorité des articles contenaient alors une étude utilisateur (environ 10%), la plus grande part d'entre elles étant consacrées aux interactions en RA.

Dünser *et al.* (2008) ont recensés différents types de mesures utilisées dans les études utilisateurs en RA :

1. *Mesures objectives* : ce sont les mesures de performance, comme le temps de complétion de la tâche, le nombre d'erreurs, la précision, les mouvements effectués ou encore le nombre d'actions. Des analyses statistiques sont ensuite fréquemment employées pour agréger et décrire les données. Ces mesures sont couramment utilisées.
2. *Mesures subjectives* : ce sont des mesures faites par les utilisateurs via des questionnaires, des notes et commentaires. Des analyses statistiques sont aussi utilisées, mais parfois seule une description des résultats est faite. Ces mesures sont assez souvent utilisées. De nombreux outils existent pour mesurer la charge de travail subjective d'une IHM, comme le NASA-TLX (Rubio *et al.*, 2004).
3. *Analyses qualitatives* : elles correspondent à des observations utilisateurs, des entrevues formelles, ou encore à une classification des comportements utilisateurs. Ces mesures sont assez peu utilisées.
4. *Évaluation d'utilisabilité* : utilisation de techniques comme des évaluations heuristiques, des évaluations d'experts, des analyses de tâche. La méthode du magicien d'Oz est une

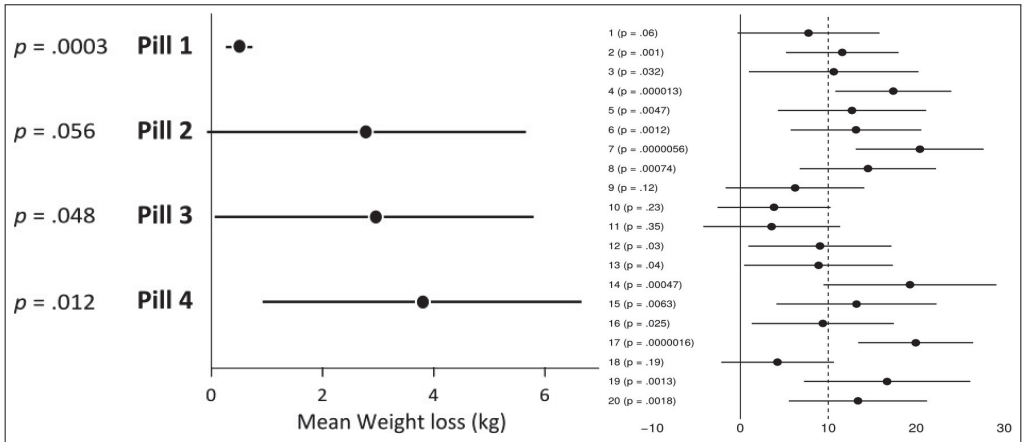
simulation d'IHM, où l'utilisateur simule ces actions et le chercheur simule les résultats, par exemple à l'aide d'éléments d'interface faits en cartons. Une dernière technique est de demander à l'utilisateur de décrire à voix hautes ces actions et réactions par rapport au système de RA. Ces techniques sont assez peu utilisées.

5. *Évaluations informelles* : ce sont des observations informelles d'utilisateurs ou des retours utilisateurs informels. Cela est utile pour des études pilotes ou pour donner des pistes de réponses sur le système de RA. Ces méthodes sont couramment utilisées.

Des analyses statistiques permettent de comprendre et interpréter des données mesurées. En pratique, on cherche souvent à indiquer des différences entre différents systèmes de RA, par exemple que les utilisateurs sont plus performants sur une IHM qu'avec une autre. On dit alors qu'une des IHM a un effet sur la performance (temps, erreurs) par rapport à l'autre. Or, on a pris des mesures que sur un *échantillon* de la *population*, qu'on espère suffisamment représentatif. On peut analyser les mesures de l'échantillon, mais on souhaiterait pouvoir dire avec un certain niveau de confiance que les résultats sont également applicables à la population.

Le test de l'hypothèse nulle est souvent utilisé pour cela dans la littérature en IHM. On postule une hypothèse alternative qu'on nomme l'hypothèse nulle : par exemple, il n'y a pas de différence d'effet sur la performance des utilisateurs entre les deux IHMs. On calcule alors la probabilité, la *valeur-p*, d'observer de tels résultats s'il n'y avait en réalité pas d'effet (si l'hypothèse nulle est vraie). Typiquement, on se donne 5% de chance d'erreur, c'est-à-dire qu'on considère que si $p < 0.05$ alors on peut rejeter l'hypothèse nulle et dire qu'il y a un effet *statistiquement significatif* : on considère que ce serait trop improbable d'obtenir de tels résultats s'il n'y avait pas d'effet. La valeur-p n'est cependant *pas* la probabilité qu'il n'y ait pas d'effet.

Pourtant cette pratique est controversée. Premièrement, si on se donne une chance d'erreur sur chaque test, plus on effectuera de tests, plus on aura de chance d'avoir rejeté à tord l'hypothèse nulle sur un des tests : c'est le problème des comparaisons multiples. Ainsi, pour 20



a) Moyennes de l'effet sur le poids de pilules amaigrissantes imaginaires, avec valeurs-p et intervalles de confiance.

b) Valeurs-p et intervalles de confiance de la moyenne d'échantillons (taille = 25) tirés d'une population normallement distribuée (moyenne = 10, écart-type = 20).

Figure 1.20 Comparaison des valeurs-p et des intervalles de confiance (sous forme de barres d'erreurs) sur des résultats.
Adapté de Dragicevic (2016).

tests, on a probablement déclaré $20 \times 0.05 = 1$ effet significatif alors qu'il n'y a en réalité pas (par exemple, échantillons 4 et 7 dans la Figure 1.20b). En pratique, il est cependant possible de réduire ou contrôler ce risque à l'aide de méthodes de correction (Holm-Bonferroni ou Benjamini-Hochberg par exemple). Deuxièmement, on ne sait pas la taille de l'effet en lui-même, ni on ne peut conclure quoi que ce soit si la valeur-p est trop élevée (par exemple, pilule 2 dans la Figure 1.20a).

Dragicevic (2016) propose alors de remplacer les tests de l'hypothèse nulle avec le calcul d'*intervalles de confiance* (IC) : c'est une marge d'erreur autour d'une estimation de l'échantillon, par exemple une moyenne, avec un degré de confiance fixé, typiquement typiquement 95%. Plus précisément, un intervalle de confiance à 95% contiendra 95 fois sur 100 l'estimation de la population (par exemple, on ne capture pas la moyenne de la population avec les échantillon 7 et 17 dans la Figure 1.20b). En pratique, on peut donc le considérer comme l'in-

tervalle des valeurs les plus plausibles de l'estimation. Ainsi, en comparant les intervalles de confiance de deux mesures, on détermine *l'importance significative* de l'effet.

Dès lors, on peut considérer que la valeur-p et l'intervalle de confiance donnent en pratique la même information Dragicevic (2016). Cependant, le test de l'hypothèse nulle est couramment utilisé dans la littérature en IHM. C'est pourquoi nous avons choisis de le conserver et de l'utiliser conjointement à des intervalles de confiances à 95%.

1.3 Espaces de travail en RA

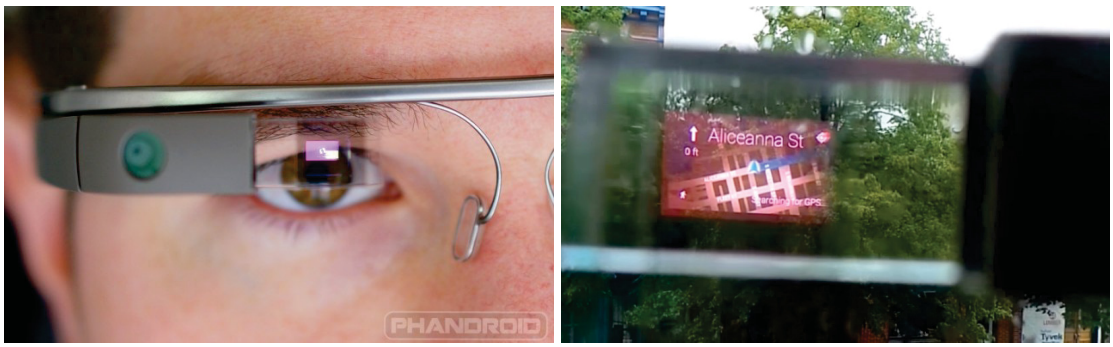
La séparation entre les styles d'IHMs faite par Rekimoto & Nagao (1995) (*Voir* Figure 1.3) est artificielle : pour Billinghurst *et al.* (2005) la RA devrait s'intégrer avec l'informatique ubiquitaire, les IHMs graphiques et la RV. Par exemple, Heun *et al.* (2013) utilise la RA comme interface pour interagir de façon naturelle avec les objets intelligents autour de soi (*Voir* Figure 1.21), tandis que le prototype SpaceTop de Lee *et al.* (2013) permet de placer les fenêtres d'une IHM WIMP sur une grille en 3D (*Voir* Figure 1.6).

Figure 1.21 Photographie du *Reality Editor*, un navigateur web de RA pour interagir facilement avec les objets intelligents autour de soi. Ici, une personne paye un parc-mètre via son téléphone.
Tiré de Heun *et al.* (2013).

Google Glass a été une expérience novatrice de visiocasque de RA. Malgré les problèmes de vie privée et d'acceptation sociale que son usage a posé, Koelle *et al.* (2015) rappellent que cela a montré qu'il était intéressant de combiner un visiocasque de RA avec un téléphone intelligent. L'utilisateur interagissait avec son téléphone via le visiocasque, par des interactions multimodales à la voix ou encore avec des gestes en l'air décodés par la caméra du visiocasque. Cette expérience montre ainsi que les visiocasques de RA vont d'abord prendre place dans nos quotidiens professionnels où ils seront mieux acceptés ; c'est également la stratégie de la nouvelle version de ce visiocasque Levy (2017). Il est donc plus intéressant d'orienter des recherches en IHM pour la RA.

Le Google Glass était un prototype limité visible seulement par l'œil droit avec un faible champ de vision (*Voir* Figure 1.22a). Mais surtout, il ne proposait pas une véritable IHM de RA : affichée sur un unique plan virtuel à une distance fixe des yeux de l'utilisateur (*Voir* Figure 1.22b), comme sur la Figure 1.19c, elle ne répond pas à l'alignement virtuel-réel dans la définition d'Azuma (1997). Aussi, cette fenêtre virtuelle peut faire occlusion avec l'environnement réel, et gêner la vue de l'utilisateur. De nombreux autres visiocasques, comme Epson Moverio ou Vusix, souffrent de ces mêmes limites. C'est pourquoi le HoloLens a apporté une innovation majeure dans l'industrie de la RA en proposant de placer de multiples fenêtres virtuelles contre les murs, comme on peut accrocher un tableau, donnant réellement le sentiment qu'elles font partie de l'environnement réel.

Partant de la même limite d'un seul écran de ces visiocasques, Ens *et al.* (2014b) proposent alors le Personal Cockpit, une IHM de RA affichant de multiples fenêtres autour de l'utilisateur (*Voir* Figure 1.23). Il peut y organiser l'information sur différents écrans qui le suivent dans ces déplacements et avec qu'il interagit directement avec une technique de main virtuelle. Les auteurs mesurent alors qu'un utilisateur travaille 40% plus rapidement avec de multiples fenêtres qu'avec une simple fenêtre fixe en faisant du multi-tâches. Ils y évaluent également

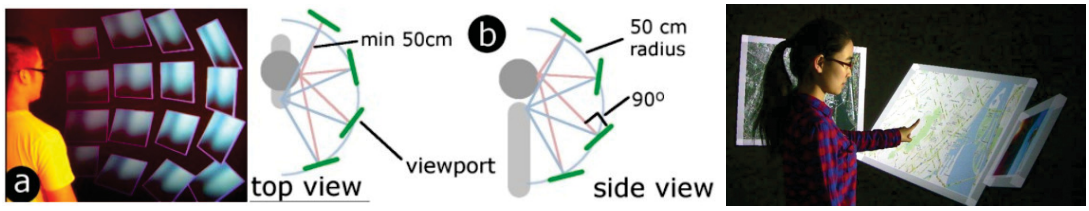


a) Utilisateur portant le visiocasque : le champ de vision est très petit et limité à l'œil droit.

b) Vue depuis le visiocasque : l'IHM est affichée sur un rectangle à distance fixe de l'utilisateur sans intégration avec l'environnement réel.

Figure 1.22 Photographies du Google Glass.
Tiré de Phandroid (2013).

plusieurs facteurs de conception : leurs résultats indiquent (1) qu'un affichage courbe est moins fatiguant et moins sujet à des erreurs de sélection, les fenêtres étant ainsi affichées à la même distance de l'utilisateur, (2) que les interactions avec une main virtuelle sont plus stables et faciles avec des fenêtres fixées dans l'espace que placées sur le corps, car l'utilisateur peut les faire bouger involontairement, et (3) que la distance minimum des fenêtres est à 50 cm de l'utilisateur, provoquant autrement de l'inconfort. Cependant, leur étude utilisait un CAVE simulant un petit champ de vision de $40° \times 30°$: il serait alors intéressant de la reproduire avec visiocasque à large champ de vision.



a) Positionnement idéal des fenêtres virtuelles en un affichage courbe.

b) Exemple d'utilisation avec une carte de navigation.

Figure 1.23 Photographies et illustrations du Personal Cockpit.
Tiré de Ens *et al.* (2014b).

Avec MultiFi, Grubert *et al.* (2015) explorent de leur côté comment combiner les entrées et les sorties des appareils intelligents que nous portons (téléphone, tablettes, montre) avec un visiocasque de RA. Ces appareils mobiles ne sont souvent pas conçus pour travailler ensemble, mais gagneraient à l'être. La Figure 1.25a montre par exemple une liste affichée sur une montre à l'écran étendu : en plaçant son téléphone sur un élément de la liste, l'utilisateur peut alors le voir en détails. Les différentes tailles et définitions d'affichage sont ainsi exploitées pour composer une seule vue pour l'utilisateur. Les auteurs distinguent alors trois alignements possibles entre une fenêtre virtuelle de RA et un appareil mobile (*Voir* Figure 1.24) :

1. mode *body-aligned* : la fenêtre a pour référence le corps de l'utilisateur, comme Ens *et al.* (2014b), l'appareil mobile formant alors une vue *detail* sur le contenu (Figure 1.25b, similaire à Bergé *et al.* (2014)) ;
2. mode *device-aligned* : la fenêtre est centrée et alignée sur l'appareil mobile dont elle étend l'écran (*Voir* Figure 1.25a) ;
3. mode *side-by-side* : un des appareils, visiocasque compris, redirige ces interactions vers un autre appareil, par exemple les interactions tactiles du téléphone vers une fenêtre virtuelle.

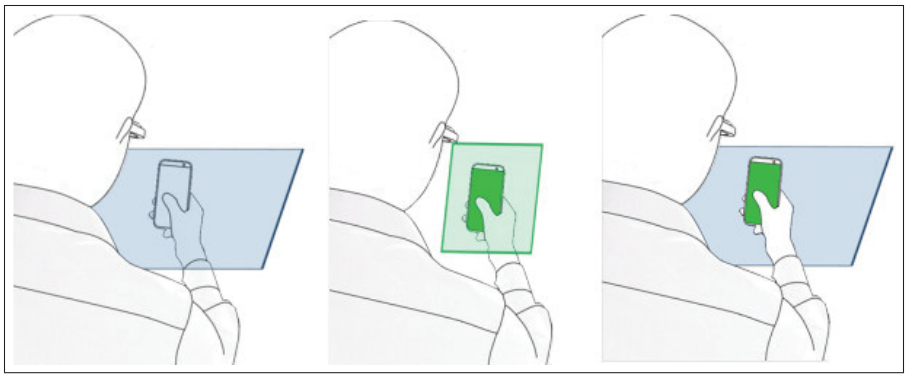
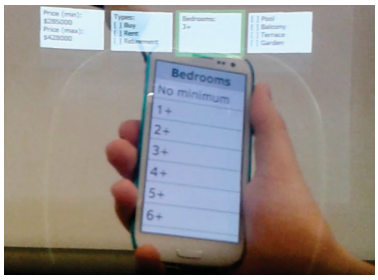


Figure 1.24 Illustrations des alignements possibles de contenu entre un visiocasque de RA et un appareil mobile : le mode *body-aligned* à gauche, le mode *device-aligned* au milieu et le mode *side-by-side* à droite.
Tiré de Grubert *et al.* (2015).

Leurs résultats expérimentaux, dans une tâche de recherche d'information et de pointage, montrent que le couplage des appareils mobiles avec un visiocasque de RA peut permettre des temps plus rapides par rapport aux appareils seuls, visiocasque compris, mais au détriment d'un plus grand effort perçu par les utilisateurs. En outre, les préférences des utilisateurs étaient variées, ce qui montre que le choix du couplage entre appareils mobiles et visiocasque doit être laissé à l'utilisateur. Cependant, le visiocasque utilisé était lourd et avait aussi un petit champ de vision de $31° \times 17°$: ne pouvant pas voir de large contenu en une fois, les participants devaient fréquemment bouger la tête. De plus, seules des interactions sur les écrans tactiles, ou en plaçant l'appareil mobile dans l'espace, ont été évaluées. Il serait intéressant de les comparer avec des interactions utilisant une main virtuelle, et dans un visiocasque avec un large champ de vision.

a) Une montre intelligente à écran étendu par RA en haut (mode *device-aligned*). Un téléphone intelligent placé sur la partie virtuelle de l'écran étendu (vue *overview*) permet de le voir plus détaillé (vue *detail*).



b) L'utilisateur pointe son téléphone vers des éléments affichés par le visiocasque en mode *body-aligned* pour les voir en détail.

c) L'écran haute résolution (vue *focus*) d'une montre intelligente étendu par le visiocasque de plus basse résolution (vue *context*).

Figure 1.25 Démonstrations de MultiFi.
Tiré de Grubert *et al.* (2015).

Serrano *et al.* (2015a) généralisent avec Gluey le mode *side-by-side* de MultiFi. Un visiocasque de RA affichant du contenu virtuel en permanence sans encombrer les mains de son utilisateur

est le support idéal pour transmettre de l'information facilement entre différents appareils. Par exemple, une personne peut utiliser un clavier, une souris ou un écran tactile et voir ses actions s'exécuter sur un autre appareil qu'il regarde (*Voir* Figure 1.26), ou encore copier des données depuis un ordinateur et les « coller » sur une imprimante pour les imprimer (*Voir* Figure 1.27). Pour les auteurs, les fonctionnalités d'une telle IHM sont (1) la redirection des entrées entre les appareils, (2) la migration du contenu entre les appareils, (3) la compatibilité de tous les appareils pour 1 et 2, (4) l'identification des appareils disponibles, (5) l'identification de leurs positions, (6) donner des retours de ces actions à l'utilisateur et (7) être mobile. Ainsi, MultiFi ne satisfait qu'aux critères 1, 5, 6 et 7. Cet article présente toutefois des limites par la faiblesse de son évaluation informelle mais aussi par le faible champ de vision de leur visiocasque.

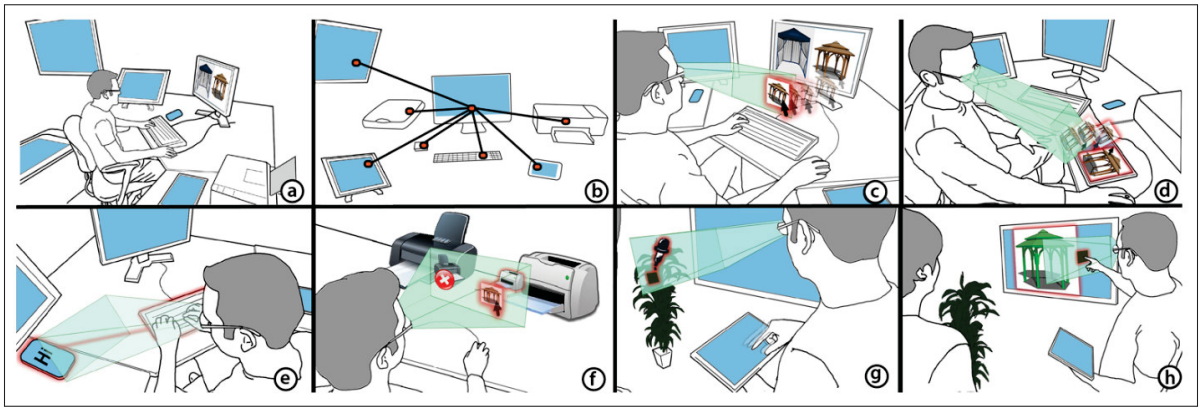
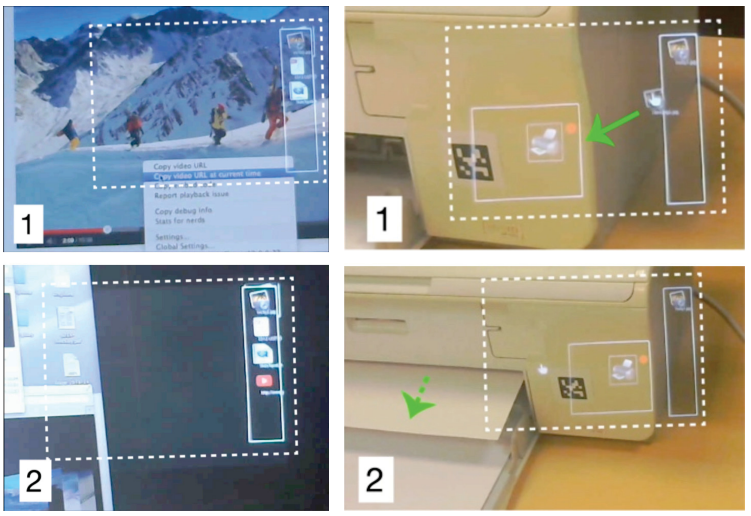


Figure 1.26 Illustration du concept de Gluey : le visiocasque de RA sert de support intermédiaire pour transmettre de l'information facilement entre tous les appareils disponibles.
Tiré de Serrano *et al.* (2015a).

Enfin, Serrano *et al.* (2015b) ont complété Gluey en proposant un concept de *bureau virtuel* qu'ils ont nommé Desktop-Gluey. Cette série d'IHMs permet d'intégrer des fenêtres virtuelles dans un environnement de travail de bureau. Leur première idée est de les utiliser en parallèle d'écrans d'ordinateur voire de les remplacer tout en continuant à utiliser clavier et souris, comme avec Gluey (*Voir* Figure 1.28a). Ces fenêtres virtuelles peuvent également être placées



a) Copie du lien d'une vidéo vers le presse-papier du visiocasque.

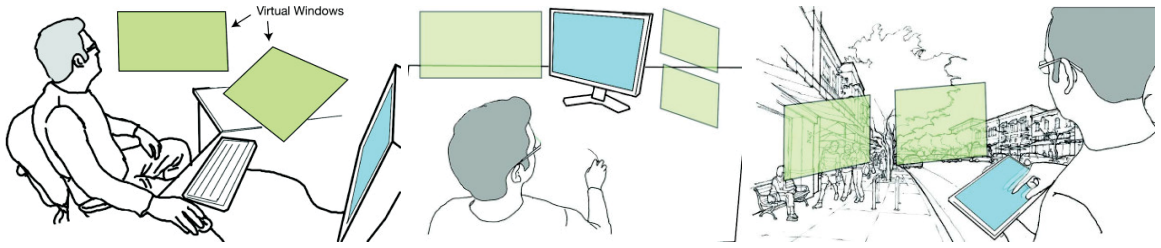
b) Impression d'une image en la déplaçant du presse-papier du visiocasque pour la déposer sur l'imprimante.

Figure 1.27 Démonstration de Gluey.
Adapté de Serrano *et al.* (2015a).

pour étendre des écrans physiques comme dans le mode *device-aligned* de MultiFi ou dans des dispositions spatiales similaires au Personal Cockpit (*Voir* Figure 1.28b). Enfin, le bureau virtuel peut être emmené avec soi, en utilisant son téléphone ou une tablette pour interagir avec ou par des interactions directes utilisant une main virtuelle (*Voir* Figure 1.28c). Ce court article ne présente en revanche ni prototype ni d'évaluation expérimentale.

On peut mieux comprendre comment toutes ces IHMs s'articulent entre elles en utilisant le cadre de conception Ethereal Planes (Ens *et al.*, 2014a). Il permet de classer et concevoir les IHMs de RA utilisant des fenêtres 2D selon sept dimensions :

- Cadre de référence :
  - *Perspective* : les fenêtres sont placées par rapport au corps ou à la tête de l'utilisateur (*égocentrique*) ou par rapport à l'environnement (*exocentrique*).
  - *Mobilité* : les fenêtres sont *mobiles* ou *fixes*.



a) Les fenêtres virtuelles peuvent remplacer des écrans physiques, tout en continuant à utiliser clavier et souris, comme dans Gluey.

b) Les fenêtres virtuelles peuvent étendre des écrans physiques.

c) En mode mobile, les fenêtres suivent l'utilisateur comme dans le Personal Cockpit.

Figure 1.28 Illustrations de Desktop-Gluey : ce concept de bureau virtuel explore comment les fenêtres virtuelles vue par RA peuvent s'intégrer dans un espace de travail de bureau.
Adapté de Serrano *et al.* (2015a).

- *Proximité* : la distance avec l'utilisateur, *sur le corps*, à portée de main (*proche*) ou au-delà (*lointain*).

- Manipulation spatiale :
  - *Mode d'entrée* : les techniques d'interactions sont soit *directes* (écran tactile, main virtuelle), soit *indirectes* (pointeur virtuel).
  - *Tangibilité* : les techniques d'interactions sont *tangibles* ou *intangibles*.
- Composition spatiale :
  - *Visibilité* : si le contenu est visible (*haut*), par exemple avec un grand champ de vision, ou s'appuie sur la mémoire spatiale de l'utilisateur (*bas*).
  - *Discrétisation* : décrit la fragmentation de l'espace virtuel, qui est plutôt *continu* (un seul affichage) ou *discret* (de multiples affichages disjoints).

Par exemple, Ens *et al.* (2014a) évaluent leur Personal Cockpit comme égocentrique et mobile, proche de l'utilisateur, à interactions indirectes et intangibles, avec une petite visibilité et continu (les écrans forment une « bulle » continue autour de l'utilisateur). Ils classent de la même manière un échantillon de 34 IHMs dans la littérature dans leur cadre en cinq catégories (*Voir* Figure 1.29). Par exemple, les applications de RA sur mobile, où la RA est vue à travers le téléphone tenu en main font partie de la catégorie *peephole*, tandis que le HoloLens fait partie de la catégorie *floating* où des fenêtres virtuelles sont placées par rapport aux murs. Ce cadre est également utile pour décrire plusieurs modes d'une IHM : un utilisateur de Desktop-Gluey place par exemple les fenêtres autour d'écran d'ordinateur dans un cadre alors exocentrique et fixé, ou autour d'un téléphone tenu en main dans un cadre égocentrique et mobile.

On pourrait s'étonner que toutes ces IHMs que nous venons de voir se limitent à des fenêtres 2D pourtant affichées dans un espace en 3D. Ens *et al.* (2014a) indiquent que nous devons déjà comprendre comment concevoir des affichages 2D en RA, car ils sont connus et bien maîtrisés sur les ordinateurs (IHMs WIMP), les téléphones ou montres intelligents et permettent donc de travailler rapidement et avec précision. De plus, les affichages 2D resteront toujours

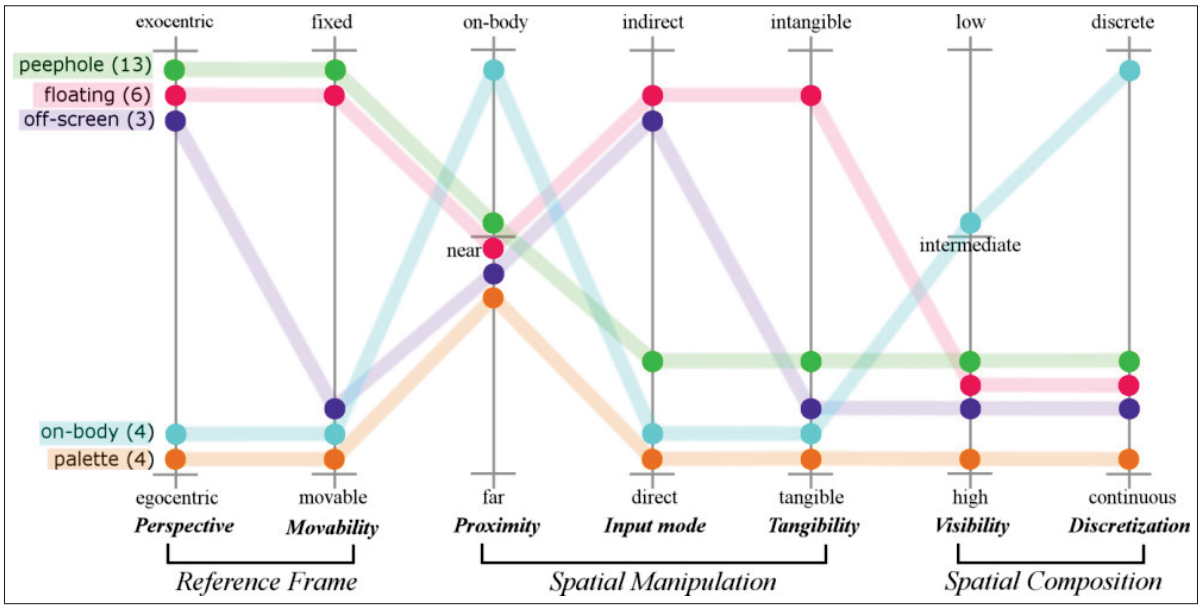


Figure 1.29 Application du cadre de conception *Ethereal Planes* à quelques exemples dans la littérature regroupés en cinq catégories.
Tiré de Ens *et al.* (2014a).

présents sous forme d'IHM et même physiquement, car [traduction] « ils resteront adaptés pour un large éventail d'utilisations, particulièrement celles impliquant des simplifications ou abstractions d'informations (par exemple, du texte, des plans, des panneaux de contrôles) » (Ens *et al.*, 2014a). De plus, l'étude d'affichages 2D en RA reste intéressante, car elle exclue des problématiques spécifiques aux IHMs 3D, comme l'occlusion ou la difficulté de s'orienter (Bergé *et al.*, 2014), et permet ainsi de se concentrer sur comment présenter l'information et comment interagir avec. Toutes ces IHMs et ce cadre de conceptions sont donc une passerelle pour ensuite bâtir des IHMs spécifiques à la RA (troisième étape du processus de Billinghurst *et al.* (2005)).

1.4 IHMs multi-échelles et larges affichages

Dépasser les limites physiques d'un écran est un thème courant dans la littérature. En effet, à partir des années 1990, les écrans des PCs ont été largement dépassés en taille par celle

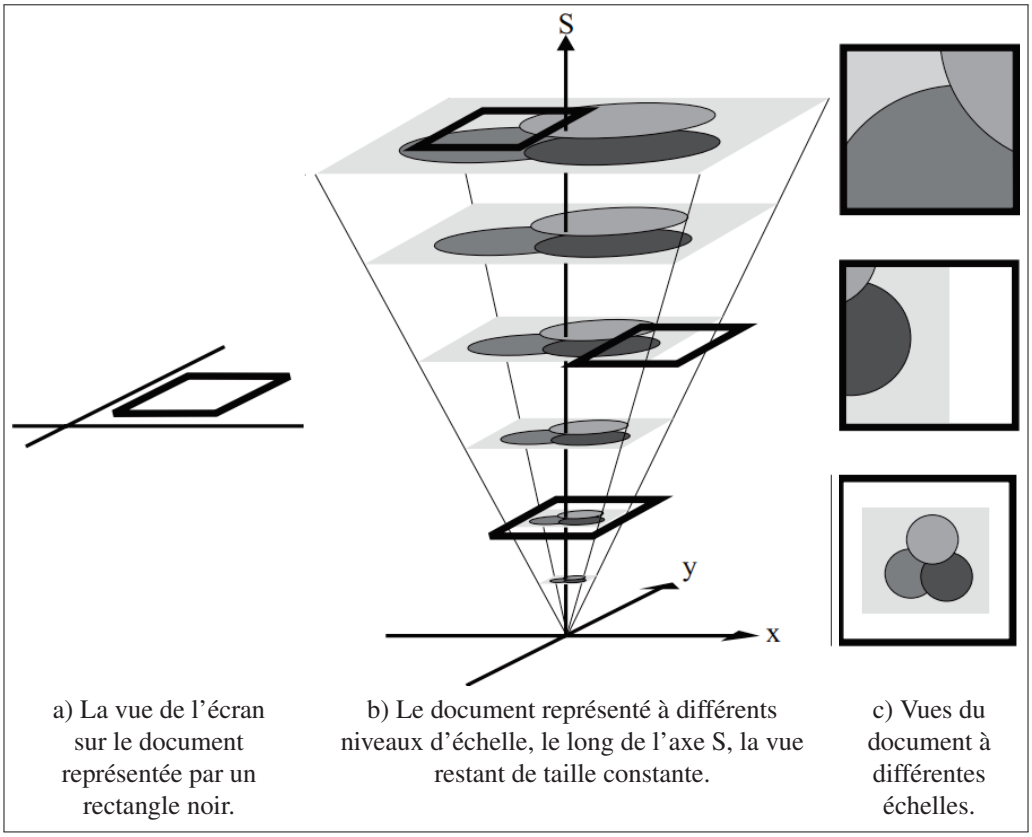


Figure 1.30 Visualisation d'un large document sur un écran plus petit. Adapté de Guiard & Beaudouin-Lafon (2004).

de certains documents (Guiard & Beaudouin-Lafon, 2004). Ces documents peuvent être des cartes géographiques, des plans, de la création graphique ; l'écran manquant de pixels (basse définition) [2] ou étant trop petit, il n'est alors pas possible de travailler correctement avec, car il est impossible visualiser en même temps une vue des détails et une vue d'ensemble (*Voir* Figure 1.30). En outre, Guiard & Beaudouin-Lafon (2004) montrent que la difficulté pour naviguer dans un grand document est inversement proportionnelle à la taille de l'écran (jusqu'à une certain seuil de taille). De même, un écran virtuel en RA de petite taille complique le multi-tâche (Ens *et al.*, 2014b). De manière plus générale, toute information ou IHM doit tenir compte des limites de l'écran sur laquelle elle est affichée car, même si un affichage parfait

existait, il serait limité par le champ de vision et la résolution de l'œil humain (Cockburn *et al.*, 2008).

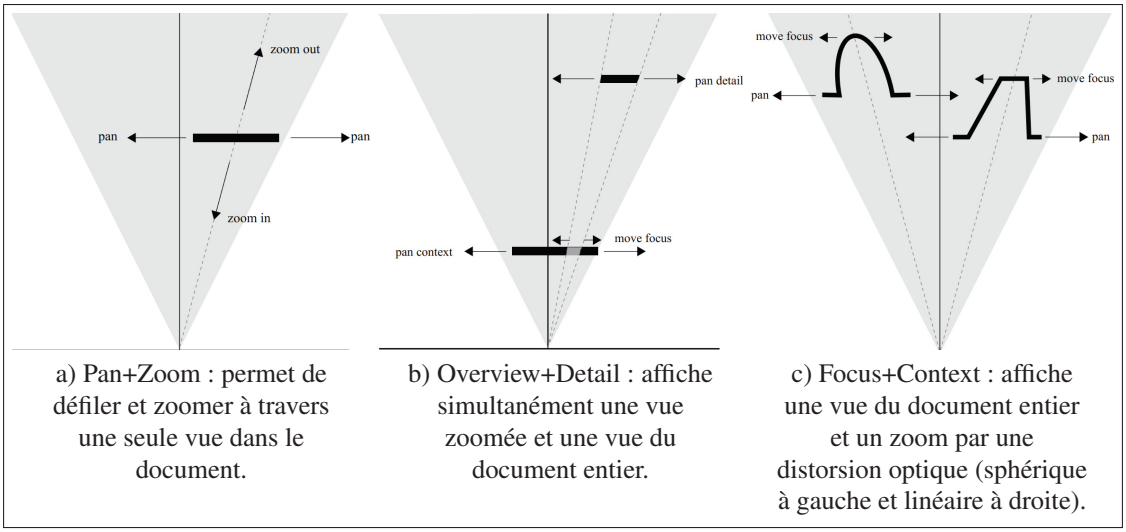
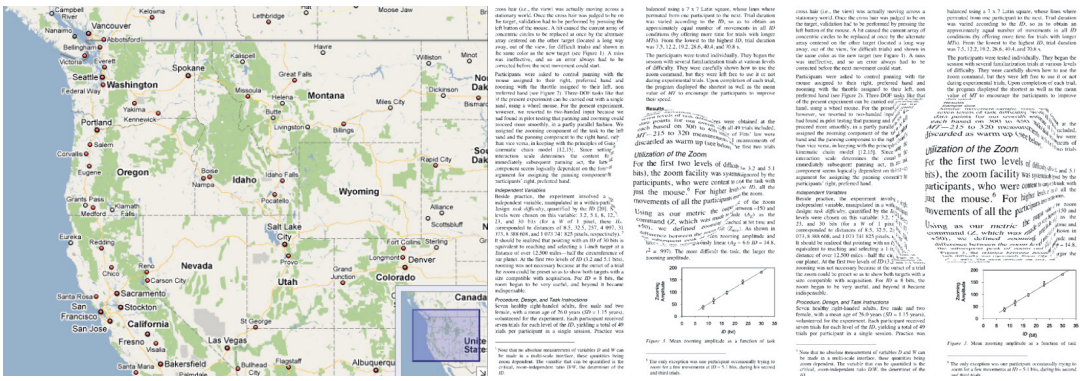


Figure 1.31 Techniques de visualisation et de navigation d'un large document sur un écran plus petit.
Adapté de Guiard & Beaudouin-Lafon (2004).

Une approche à ce problème est d'utiliser des IHMs multi-échelles (*multiscale interfaces* en anglais). Le principe est d'afficher à l'écran une ou plusieurs *vues* du document à différentes échelles et positions. Guiard & Beaudouin-Lafon (2004) décrivent trois techniques courantes :

- Pan+Zoom : une vue peut-être déplacée sur le document par défilement (*pan*), tandis que le zoom permet de changer l'échelle du document (*Voir* Figure 1.31a). Il y a donc une séparation temporelle entre les vues.
- Overview+Detail : une vue zoomée (*detail*) et une vue sur le document en entier (*overview*) sont affichées simultanément (*Voir* Figure 1.31b), créant une séparation spatiale de l'affichage. Elles sont placées côte-à-côte ou superposées et peuvent être déplacées indépendamment (*Voir* Figure 1.32a).

2. La résolution d'un écran, en pixels par pouce, permet de convertir sa définition (d_x, d_y), en pixels, vers sa taille physique (L, H), en centimètres, et vice-versa : $\begin{cases} L = d_x / resolution \\ H = d_y / resolution \end{cases}$.



a) Exemple d'IHM Overview+Detail avec une carte de navigation : une vue du document entier (en bas à droite) est affichée par-dessus une vue zoomée.

b) Exemple d'IHM Focus+Context avec une distorsion optique sphérique à gauche et linéaire à droite.

Figure 1.32 Exemples des techniques a) Overview+Detail et b) Focus+Context.
Adapté de a) Burigat & Chittaro (2013) et b) Guiard & Beaudouin-Lafon (2004).

- Focus+Context : une seule vue permet d'afficher le document en entier et le document zoomé (*Voir* Figure 1.31c), par exemple grâce à une distorsion optique (*Voir* Figure 1.32b).

Cockburn *et al.* (2008) y ajoutent une quatrième technique : les indications visuelles. Cela consiste à modifier le document pour pointer certaines informations, par exemple en les surlignant, en modifiant leur taille ou encore en effaçant les informations moins importantes. Elles ont toutes été étudiées exhaustivement. Pourtant, aucune n'est idéale : l'utilisateur doit comprendre la séparation temporelle de la vue pour Pan+Zoom ou la séparation spatiale, qui ne doit alors pas être trop importante entre les deux vues, pour Overview+Detail et la distorsion Focus+Context nuit à la performance dans les tâches de sélections (Cockburn *et al.*, 2008). Enfin, ces quatre techniques sont compatibles : beaucoup d'applications utilisent la combinaison Pan+Zoom, Overview+Detail avec des indications visuelles.

Une autre approche, complémentaire aux IHMs multi-échelles, est donc d'augmenter la taille et la définition de l'écran. Baudisch *et al.* (2002) ont par exemple agrandit un écran d'ordinateur

par un projecteur : cette combinaison forme une IHM Focus+Context, le projecteur ayant une résolution beaucoup plus réduite que celle de l'écran (*Voir* Figure 1.33). Comparé à des IHMs sur ordinateur Pan+Zoom et Overview+Detail dans une étude utilisateur avec deux tâches de navigation (une carte et un plan de circuit électronique), leur prototype est 21% à 36% plus rapide. Cela est dû probablement à sa grande taille qui permet, malgré sa faible résolution, à un utilisateur de reconstruire plus facilement une image mentale du document.

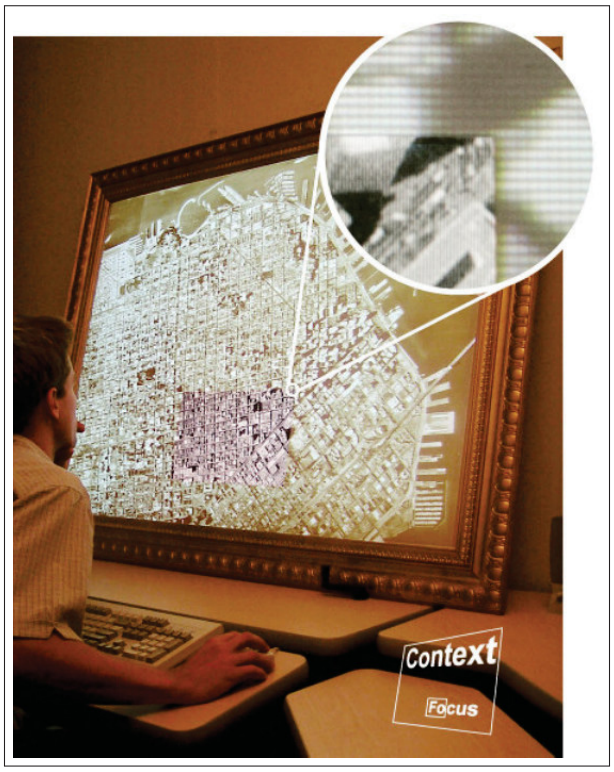


Figure 1.33 Combinaison d'un écran d'ordinateur haute résolution (*focus*) aligné avec un projecteur basse résolution (*context*).
Tiré de Baudisch *et al.* (2002).

Czerwinski *et al.* (2003) se demandent alors si de plus grands écrans hautes résolutions mènent à de même résultats pour des activités multi-tâches. Ils comparent alors un unique écran de 15 pouces avec un affichage courbe composé de multiples écrans (équivalent à trois écrans de 15 pouces côte-à-côte), dans une étude utilisateur demandant de travailler sur une tâche complexe

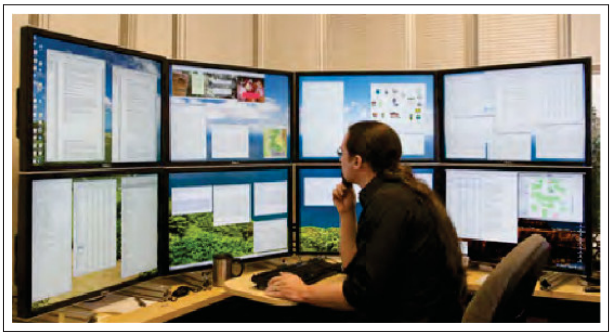
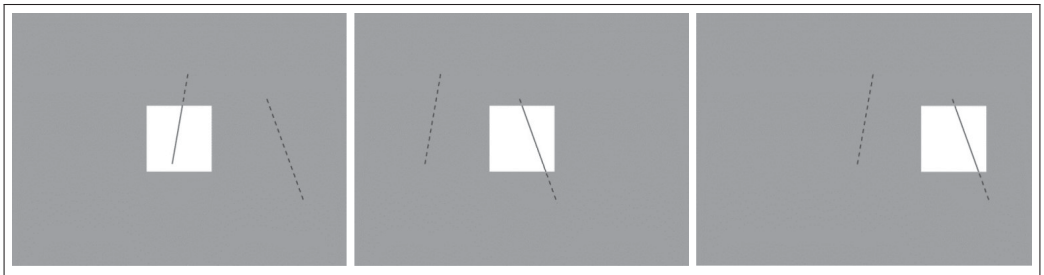


Figure 1.34 Large affichage d'une définition de 10 240 px × 3200 px composé de 4 × 2 écrans.
Tiré de Andrews *et al.* (2010).

avec de multiples fenêtres. La Figure 1.34 montre un exemple similaire d'affichage de plusieurs écrans. Leurs résultats indiquent que les participants sont significativement (9%) plus rapides avec l'affichage large et le préfèrent. Les auteurs ont observé que les utilisateurs perdaient du temps sur le petit écran à faire la gestion des fenêtres (changement de taille, repositionnement). Cependant, le grand affichage demandait un temps d'apprentissage, car il leur était inhabituel.



a) Vue initiale.

b) Défilement statique : l'utilisateur déplace le document (l'écran reste fixe).

c) Défilement dynamique : l'utilisateur déplace l'écran (le document reste fixe).

Figure 1.35 Différents types de défilement d'un large document. Le rectangle gris représente le document, le carré blanc l'écran.
Adapté de Mehra *et al.* (2006).

Les techniques des IHMs multi-échelles requièrent toutes des opérations de défilements de la part de l'utilisateur. Mehra *et al.* (2006) distinguent deux types de défilements : statiques, où

l'utilisateur déplace le document tandis que l'écran reste fixe (*Voir* Figure 1.35b), ou dynamiques, où l'utilisateur déplace l'écran tandis que le document reste fixe (*Voir* Figure 1.35c). L'avantage pour ce second type de défilement est que l'utilisateur peut se servir de sa mémoire spatiale pour se construire une représentation du document. Dans une étude utilisateur, Mehra *et al.* comparent les deux types de défilement : les participants ont navigué sur un écran par défilement à l'aide d'une souris dans un document affichant des lignes pour déterminer la plus grande (*Voir* Figure 1.35). Les réponses correctes et les temps de réactions sont meilleurs avec un défilement dynamique qui est également préféré des participants. Pour Mehra *et al.*, une personne utilisant le défilement dynamique a simplement à reconstruire le document à travers le temps, les lignes restant fixes par rapport à lui. Au contraire, un défilement statique demande plus d'effort avec une reconstruction du document à la fois à travers le temps et à travers l'espace, les lignes changeant de positions (*Voir* Figure 1.35b).

Dans la continuité des études sur la taille, le nombre et la résolution des écrans sur ordinateurs, beaucoup d'études sont faites sur les affichages muraux. Ces grands affichages ont suffisamment de définition pour afficher complètement un document sans le remettre à l'échelle. Ils sont aussi particulièrement adaptés pour du défilement dynamique : l'utilisateur se déplace physiquement, s'approchant pour voir le détail et reculant pour une vue d'ensemble.

Liu *et al.* (2014) ont alors comparé dans une étude utilisateur une IHM Pan+Zoom sur un écran d'ordinateur avec un affichage mural, sur une tâche de classification d'une grille de disques (*Voir* Figure 1.36). Une lettre de très petite taille était inscrite dans chaque disque, demandant de zoomer (ou s'approcher) pour la voir ; il fallait alors grouper les disques avec la même lettre. Leurs résultats sont fortement corrélés avec l'affichage : l'ordinateur était plus rapide et préféré des participants pour les tâches simples, l'affichage mural quand la tâche était plus difficile. Ces résultats s'expliquent probablement car la technique Pan+Zoom (défilement statique) sur l'ordinateur est connue et maîtrisée des utilisateurs, mais que l'affichage mural profite de sa

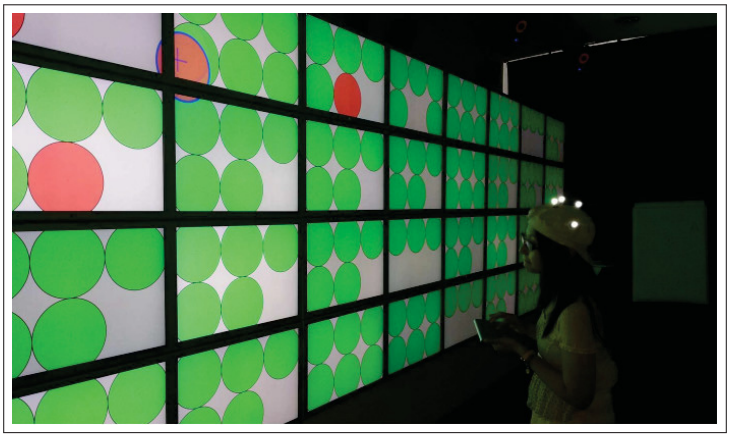
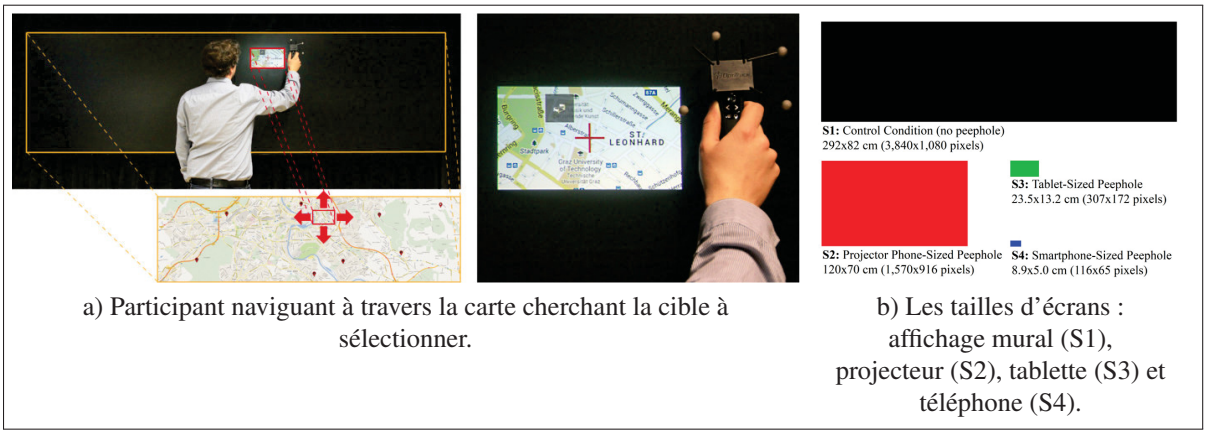


Figure 1.36 Participant déplaçant un disque (en rouge, surligné de bleu, en haut à gauche) pour le classer au bon endroit. La navigation est physique et dynamique : en se déplaçant face à l'affichage mural et en se rapprochant plus ou moins des écrans.
Tiré de Liu *et al.* (2014).

grande taille (Czerwinski *et al.*, 2003) et de son défilement physique (la personne se déplace) et dynamique (Mehra *et al.*, 2006). On peut alors penser qu'un téléphone à l'écran étendu permettrait de réaliser des tâches complexes plus facilement que sur un téléphone seul.



a) Participant naviguant à travers la carte cherchant la cible à sélectionner.

b) Les tailles d'écrans : affichage mural (S1), projecteur (S2), tablette (S3) et téléphone (S4).

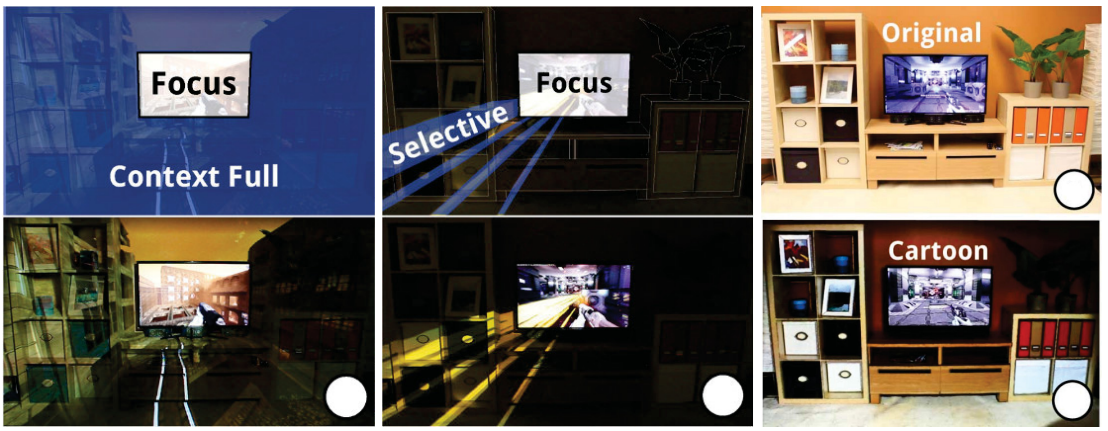
Figure 1.37 Comparaison dans une tâche de navigation (défilement dynamique) de différentes tailles d'écrans sur un affichage mural.
Adapté de Rädle *et al.* (2014).

Rädle *et al.* (2014) ont eux simulé sur un affichage mural différentes tailles possibles d'écrans (*Voir* Figure 1.37b) dans une tâche de navigation et sélection de cibles (*Voir* Figure 1.37a). Ils n'ont pas comparé différents appareils pour faire varier uniquement la taille de l'écran, et contrôler des variables parasites comme le poids, la latence ou la résolution (fixé à 33,5 px/po). Rädle *et al.* n'ont pu mesurer aucune différence en termes de temps, de distance de défilement, de préférence ou de charge mentale entre l'affichage mural (S1, 292 cm × 82 cm) et le projecteur (S2, 120 cm × 70 cm, couvrant 127° du champ de vision des participants). S2 est légèrement plus rapide que la tablette (S3, 23,5 cm × 13,2 cm), mais sans différence de préférences et charge mentale. Le téléphone (S4, 8,9 cm × 5 cm) est, sans surprises, surpassé par toutes les autres tailles. Par conséquent, pour Rädle *et al.*, une tablette peut être de taille suffisante pour naviguer dans de large document sans représenter une charge mentale supplémentaire. Ainsi, il serait intéressant de reproduire une expérimentation similaire pour comparer une navigation statique sur un téléphone seul contre un téléphone à l'écran étendu par RA.

1.5 Affichages étendus et affichages joints

La technique d'augmenter la taille d'un écran d'ordinateur à l'aide d'un projecteur formant une IHM Focus+Context de Baudisch *et al.* (2002), est transposable à d'autres types d'affichages. Elle permet également de résoudre d'autres problématiques que la visualisation de grands documents. De manière plus générale, comme Grubert *et al.* (2015) et Serrano *et al.* (2015a), il est possible de concevoir de multiples IHMs combinant les différents types d'affichages que nous avons vus : visiocasques, téléphones intelligents, projecteurs, affichages muraux.

Jones *et al.* (2013) ont utilisé un projecteur pour augmenter un écran de télévision. Ils ont alors cherché à exploiter les propriétés de la vue *context* avec 11 visualisations différentes utilisables dans un jeu-vidéo, comme simplement étendre l'écran (*Voir* Figure 1.38a), afficher seulement des éléments précis (*Voir* Figure 1.38b) ou encore modifier l'apparence de l'environnement (*Voir* Figure 1.38c). Ils les ont alors évaluées auprès de joueurs et de concepteurs de jeu :



a) L'affichage de la télévision est étendu par le projecteur.

b) Seuls quelques éléments précis sont affichés par le projecteur (ici des trajectoires de balles).

c) Le projecteur peut modifier l'apparence de l'environnement autour de la télévision.

Figure 1.38 Illumiroom : une télévision haute-définition (vue *focus*) augmentée par un projecteur (vue *context*).
Adapté de Jones *et al.* (2013).

tous ont été impressionnés et enthousiasmés par le concept. Cependant l'effet peut être trop envahissant sur le long terme : Jones *et al.* (2013) concluent qu'un équilibre doit être trouvé entre contenu et confort (Figure 1.38 (b)).

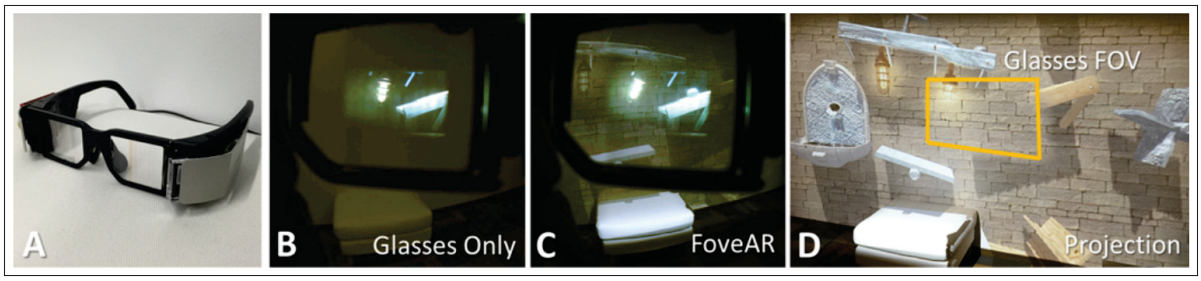


Figure 1.39 FoveAR : (a) le visiocasque de RA utilisé, (b) vue depuis le visiocasque non-étendu, (c) vue depuis le visiocasque étendu par un projecteur, (d) illustration du champ de vision du visiocasque étendu.
Tiré de Benko *et al.* (2015).

Benko *et al.* (2015) ont également utilisé un projecteur pour augmenter un visiocasque de RA de l'industrie (*Voir* Figure 1.39) à un champ de vision horizontal de 100°, plus proche de

celui de l'œil humain. Les résolutions des deux appareils sont proches, mais leurs propriétés différentes : l'IHM du projecteur est publique et partagée, tandis que celle du visiocasque est privée et en 3D. Mais ce concept est trop limité par sa complexité d'installation et son manque de portabilité.

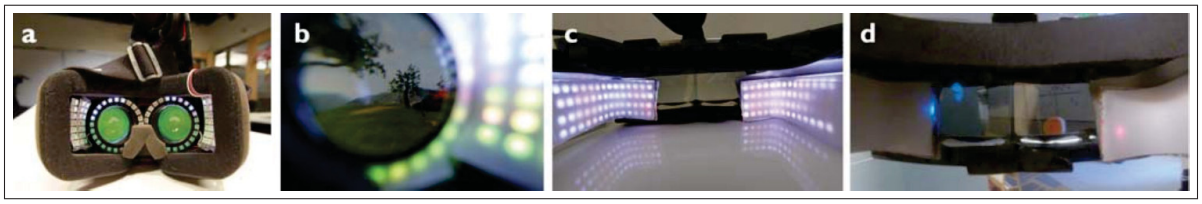


Figure 1.40 Les visiocasques de RV SparseLightVR (a, b) et de RA SparseLightAR (c, d) utilisant des LEDs pour étendre leur champ de vision. Tiré de Xiao & Benko (2016).

Une alternative portable pour augmenter le champ de vision d'un visiocasque de VR ou de RA est d'utiliser des diodes électroluminescentes (DEL), disposées autour de l'écran (*Voir* Figure 1.40). Ces diodes forment une vue *context* de très basse résolution mais peu chère, légère, avec un fort contraste et destinée à être vue par la vision périphérique de l'œil. Xiao & Benko (2016) ont alors appliqué cette technique au visiocasque de RV Oculus DK2 et sur un visiocasque de RA optique de leur conception ; ils nomment ces visiocasques des affichages périphériques clairsemés (*sparse peripheral display* en anglais). Les deux études expérimentales qu'ils ont menés indiquent que cette technique permet de réduire les nausées parfois ressenties en RV et RA. Enfin, comme Jones *et al.* (2013) l'ont explorée en détail, la vue *context* peut être utilisée de différentes manières : par exemple, simplement étendre la vue *detail* en plus basse résolution, ou encore afficher seulement des objets d'importance en périphérie.

Bergé *et al.* (2014) explorent l'aspect public d'un affichage mural, en proposant une IHM Overview+Detail utilisant le téléphone (vue *detail*) pour en voir une vue privée (vue *overview*). Ils comparent alors dans leurs deux études expérimentales trois techniques de défilement de la vue *detail* (*Voir* Figure 1.41) : soit avec l'écran tactile (défilement statique sur la vue du téléphone,

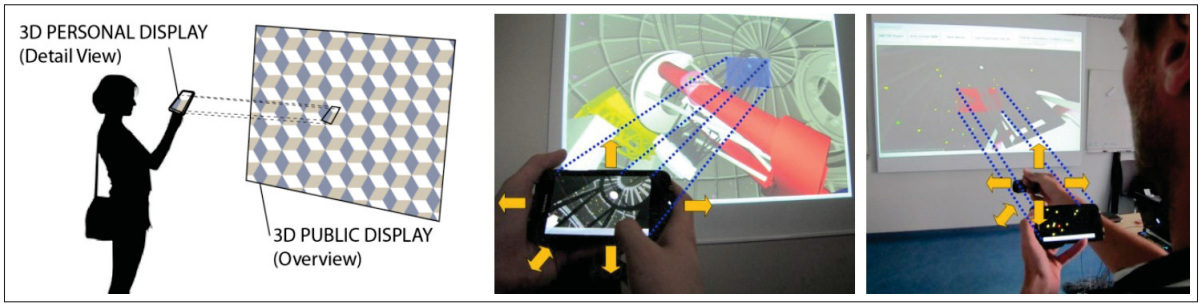


Figure 1.41 Combinaison d'un téléphone et d'un grand écran dans une IHM *Overview+Detail* : (a) illustration du concept, le téléphone est une vue *detail*, projetté sur le projecteur en *overview*, (b) la vue est déplacée avec le téléphone, (c) la vue est déplacée avec une main virtuelle.
Tiré de Bergé *et al.* (2014).

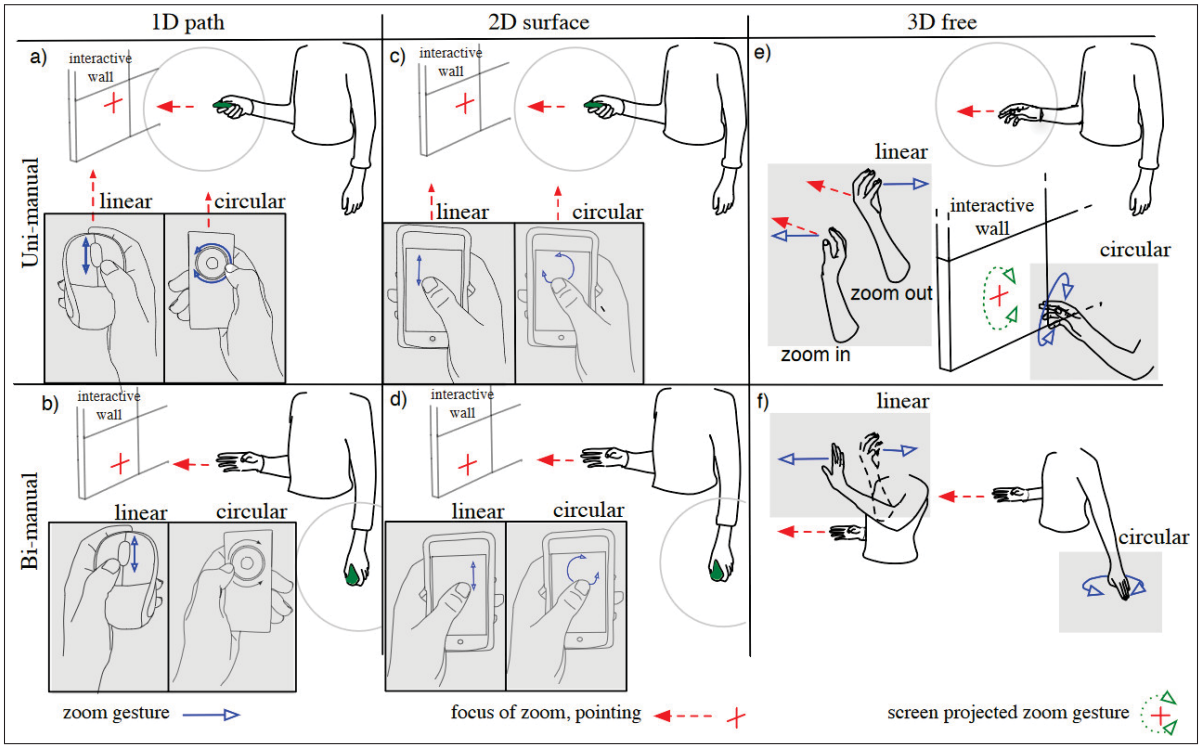


Figure 1.42 Différentes techniques d'interactions, uni ou bi-manuelles, utilisant une molette d'un objet tenu en main, un écran tactile ou une main virtuelle pour zoomer (translation avec 1 DoF) sur un affichage mural.
Tiré de Nancel *et al.* (2011).

via des joysticks), soit en déplaçant le téléphone face au mur (défilement dynamique), soit avec une main virtuelle (un bouton sur l'écran tactile active le suivi ou non de la main). Contrairement à la plupart des articles que nous avons présentés, la navigation est ici en 3D et non en 2D (translation avec 3 DoFs). Leurs résultats indiquent que la main virtuelle et le suivi du téléphone étaient les plus rapides et préférés des participants ; cependant, l'usage de joysticks plutôt que des défilements directs avec un doigt, et zoom avec deux doigts, ont pu jouer en défaveur de l'écran tactile. Nancel *et al.* (2011) avaient comparé des techniques d'interactions similaires pour zoomer seulement (translation avec 1 DoF) sur un affichage mural et avaient mis en évidence que les gestes utilisant une main virtuelle étaient moins efficaces et plus fatigants que ceux utilisant un écran tactile ou une molette tenue en main (*Voir* Figure 1.42). Il serait alors intéressant de comparer ces techniques de navigation avec un contenu en 2D.

Figure 1.43 MagicDesk : le clavier et la souris d'un ordinateur sont posés sur et augmentés par une table tactile.
Tiré de Bi *et al.* (2011).

Enfin, de manière similaire à Serrano *et al.* (2015a) avec un visiocasque de RA, Bi *et al.* (2011) proposent d'augmenter l'ordinateur en posant écran, clavier et souris sur une table tactile (*Voir* Figure 1.43) : des fenêtres et barres d'outils peuvent être déplacées de l'ordinateur vers la table tactile, des miniatures pour interagir avec des fenêtres ouvertes y sont affichées en bas de la

table tactile et le clavier et la souris sont augmentés par une IHM les suivant. Bi *et al.* ont évalué la faisabilité de leur concept : les participants pouvaient facilement faire aller et venir leurs mains du clavier et de la souris pour interagir à une main ou à deux mains sur la table tactile, plus facilement au-dessus et en dessous du clavier mais plus difficilement sur les côtés gauche et droit de la table tactile.

1.6 Problématique

Nous avons tout d'abord vu qu'il y a eu des efforts importants pour rendre la RA techniquement possible (Azuma *et al.*, 2001; Van Krevelen & Poelman, 2010). Malgré plusieurs cadres théoriques proposés (Milgram & Kishino, 1994; Rekimoto & Nagao, 1995; Bimber & Raskar, 2005; Ens *et al.*, 2014a), il y a encore un fort besoin de conception d'IHMs pour la RA (Billinghurst *et al.*, 2015), idéalement spécifiques à ce média et non importées d'autres types d'IHMs comme WIMP ou les écrans tactiles (Van Dam, 1997; Billinghurst *et al.*, 2005). En outre, il est peu clair quelles techniques d'interactions sont les plus adaptées pour la RA (Argelaguet & Andujar, 2013; Piumsomboon *et al.*, 2013, 2014). Pourtant, une des applications clé de la RA est, selon nous, dans un usage professionnel où le visiocasque de RA prendrait une place centrale parmi tous nos appareils numériques (Rekimoto & Nagao, 1995; Serrano *et al.*, 2015a). Il permettrait, entre autres applications, de joindre et étendre les écrans de ces appareils, en particulier ceux limités en taille comme les téléphones ou les montres intelligentes (Grubert *et al.*, 2015). En outre, l'étude de l'agrandissement des écrans, d'abord des ordinateurs (Baudisch *et al.*, 2002; Guiard & Beaudouin-Lafon, 2004) puis des affichages muraux (Liu *et al.*, 2014; Rädle *et al.*, 2014), montre que des écrans de grande taille permettent de travailler plus efficacement avec de larges documents (comme des cartes) ou sur du multitâche. Enfin, on se demande si les études de techniques d'interactions combinant affichages muraux et écrans tactiles (Nancel *et al.*, 2011; Bergé *et al.*, 2014) peuvent s'appliquer à un écran de téléphone étendu par RA.

Nous définissons la problématique de ce mémoire ainsi : est-ce qu'un téléphone à écran étendu donne un avantage à un utilisateur par rapport à un téléphone non étendu ? Est-il préférable d'interagir avec une main virtuelle directement sur l'écran virtuel, autour du téléphone, ou en utilisant seulement l'écran tactile ?

Les interactions tactiles ont l'avantage d'être précises, stables, fiables et bien connues des utilisateurs. Cependant, les interactions avec une main virtuelle permettent d'agir directement sur la partie virtuelle de l'écran étendu, ce qui pourrait être plus intuitif. De plus, une IHM pourrait bénéficier d'actions coordonnées des mains (White *et al.*, 2009). Pourtant pointer en 3D avec une main virtuelle est plus difficile que sur une surface tangible en 2D (Argelaguet & Andujar, 2013).

Nous formulons les hypothèses suivantes par rapport à cette problématique :

(H1) Un téléphone à écran étendu permettra à un utilisateur d'être plus performant sur des tâches de navigation, de classification ou demandant d'utiliser plusieurs applications en parallèle que sur un téléphone non étendu, quelle que soit la technique d'interaction utilisée.

(H2) Les utilisateurs apprécieront d'avantage pouvoir interagir directement avec la fenêtre virtuelle autour du téléphone : l'utilisateur agissant sur le contenu qu'il voit, cela lui semblera plus naturel et possiblement plus intuitif.

(H3) Les utilisateurs seront en revanche plus performants en interagissant seulement avec l'écran physique tactile du téléphone qu'en utilisant un main virtuelle, car les gestes y sont plus précis et moins fatigants sur un petit écran tactile tenu de manière stable.

Enfin, pour y répondre, nous divisons cette problématique en trois sous-problèmes :

1. Concevoir une IHM d'un téléphone à écran étendu.
2. Développer un visiocasque de RA à large champ de vision.

3. Réaliser une expérimentation évaluant différentes techniques d'interactions sur téléphone à écran étendu comparées à un téléphone non étendu.

Les résultats à ces objectifs permettront de donner des recommandations pour de futures recherches d'IHM en RA.

CHAPITRE 2

CONCEPT

2.1 Présentation

Un VESAD est une technique d'extension d'un écran physique par RA : un visiocasque de RA affiche une fenêtre virtuelle placée automatiquement autour d'un écran physique que l'on souhaite étendre, sur le même plan que lui. En synchronisant ces deux affichages, l'utilisateur peut alors voir et interagir avec comme s'il s'agissait d'un seul écran étendu. Cette technique peut ainsi être appliquée pour tout type d'écran, incluant ceux des ordinateurs, les télévisions ou les affichages muraux. Nous travaillons cependant seulement avec l'extension de l'écran d'un téléphone intelligent, notre premier sous-problème étant de concevoir l'IHM d'un VESAD sur un téléphone. Nous la situons d'abord par rapport à la littérature, puis nous décrivons quelques applications et techniques d'interactions potentielles.

Notre IHM s'inscrit dans la continuité de plusieurs travaux. Elle rend d'abord concret le concept de bureau de travail en RA Desktop Gluey (Serrano *et al.*, 2015b) dont nous implémentons donc le cas où les fenêtres virtuelles sont alignées avec un écran physique mobile. Elle est donc également un cas particulier du Personal Cockpit (Ens *et al.*, 2014b) où nous situons la fenêtre virtuelle non pas par rapport au corps mais autour du téléphone. Enfin, elle correspond au mode *device-aligned* dans MultiFi (Grubert *et al.*, 2015), dont nous reprenons l'idée d'écrans étendus par RA.

Nous situons donc notre IHM en utilisant le cadre de conception Ethereal Planes (Ens *et al.*, 2014a). Cela est pertinent, car nous concevons une IHM sous forme d'une fenêtre 2D dans un environnement de RA. La Figure 2.1 montre qu'un téléphone étendu par un VESAD suit les caractéristiques de la catégorie *palette* pour le cadre de référence et la composition spatiale.

Cependant, nous souhaitons comparer une technique d'interaction utilisant l'écran tactile (catégorie *palette*) à une main virtuelle (catégorie *floating*) dans notre évaluation expérimentale.

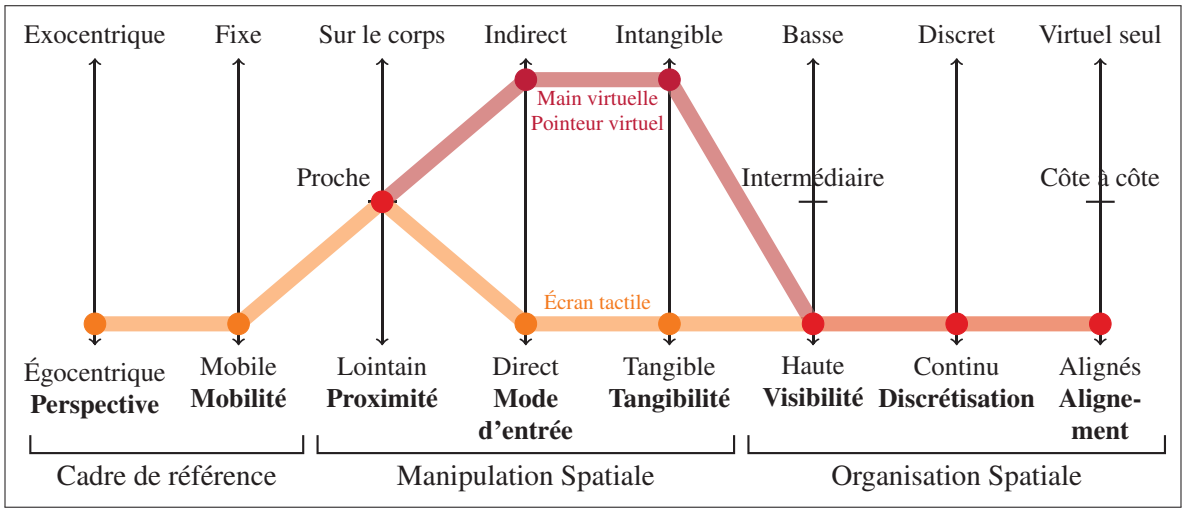


Figure 2.1 Évaluation de notre IHM de téléphone étendu par un VESAD avec notre version d'Ethereal Planes : il suit la catégorie *palette* (en orange), sauf pour le mode d'entrée et la tangibilité où nous proposons plusieurs techniques d'interactions utilisées dans les catégories *palette* et *floating* (en violet). Quand ces deux catégories se recoupent, on utilise le rouge.

En outre, nous avons ajouté une nouvelle dimension à ce cadre. Le mode d'alignement de MultiFi (Grubert *et al.*, 2015) décrit le placement de la fenêtre virtuelle, à la fois par rapport à l'utilisateur et par rapport au téléphone ou à la montre intelligente ainsi que les interactions possibles. Cette description est un peu floue et mélange en réalité les dimensions de perspective et de mode d'entrée du cadre avec une nouvelle dimension que nous appelons *alignement*. Cette dernière permet de répondre à la question de l'alignement virtuel-réel dans la définition d'Azuma (1997) : comment le contenu virtuel se mêle aux objets réels (par exemple des écrans physiques) ? En effet, en suivant l'échelle de Milgram & Kishino (1994), on imagine qu'un espace de travail en RA permet de :

- travailler seulement avec des fenêtres virtuelles sans lien avec des objets réels comme avec le Personal Cockpit ou le HoloLens (virtuel seul) ;

- utiliser ensemble des fenêtres virtuelles et des objets réels, sans qu'il n'y ait de relations spatiales entre eux, en redirigeant éventuellement entrées et sorties entre eux, comme dans Gluey ou Desktop-Gluey (côte à côte) ;
- étendre, joindre et augmenter des objets réels avec des fenêtres virtuels, comme avec MultiFi ou notre VESAD (alignés).

Ainsi, toutes ces IHMs d'espace de travail en RA que nous venons de citer dans ce chapitre sont compatibles et complémentaires. Nous les imaginons donc comme différents modes d'un seul système d'exploitation utilisant un visiocasque de RA à large champ de vision (haute visibilité) et les appareils intelligents disponibles. Nous les présentons sur le Tableau 2.1 : un tel système d'exploitation proposerait l'alignement avec les écrans physiques que souhaite l'utilisateur, dans tous les cadres de références et toutes les proximités possibles. La discrétisation contrôle à quel point l'information est fragmentée sur les écrans virtuels et physiques utilisés, ou s'ils forment un affichage continu. Plusieurs techniques d'interactions devraient être disponibles mais nous faisons l'hypothèse que certaines sont plus adaptées que d'autres en fonction du mode utilisé et du contexte d'utilisation.

Dans ce mémoire, nous explorons, implémentons, et évaluons donc seulement une technique d'alignement (écrans étendus ou VESAD) dans la continuité de MultiFi. Le téléphone étant tenu en main, il s'agit d'un cadre égocentrique mobile. Enfin, contrairement au prototype de MultiFi qui peut être qualifié d'une IHM Focus+Context (Baudisch *et al.*, 2002), le visiocasque étant de bien moindre résolution que les écrans qu'il étend, notre VESAD a une résolution identique sur tout l'écran étendu.

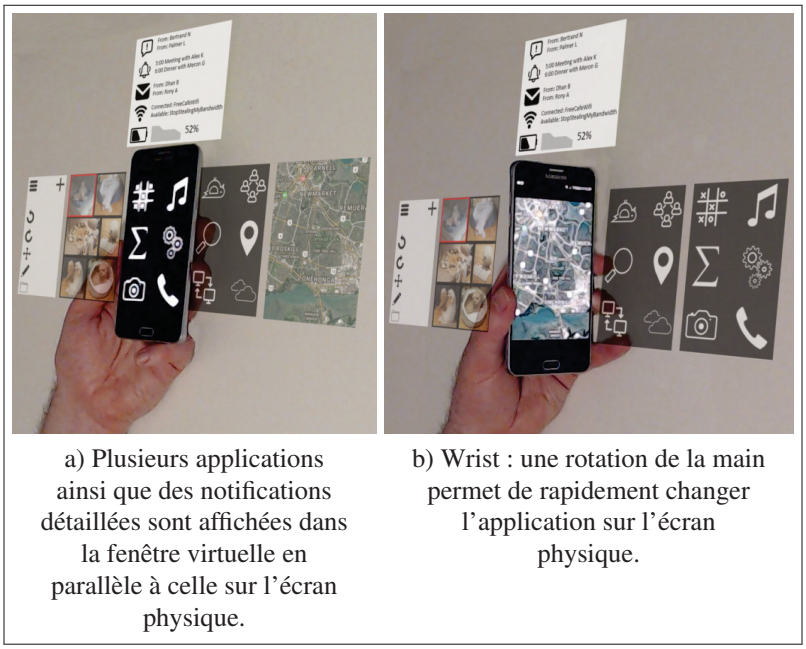
2.2 Applications potentielles

Nous distinguons deux modes de vues, correspondants à la dimension discrétisation d'Ethereal Planes, pour afficher du contenu dans un téléphone à écran étendu :

Tableau 2.1 Correspondance des différentes IHMs de la section 1.3 dans les dimension perspective, mobilité et alignement (avec des écrans physiques) de notre version d'Ethereal Planes.

			Alignement		
			Virtuel seul	**Côte à côte**	**Alignés**
Cadre de référence	**Exocentrique**	**Fixe**	HoloLens	Gluey / Desktop Gluey	VESAD / MultiFi
		Mobile			
	Égocentrique	**Fixe**	Personal Cockpit		
		Mobile			

- Vue multi-fenêtres : de multiples applications utilisées conjointement sont placées sur l'écran physique et le reste de l'écran étendu (*Voir* Figure 2.2).
- Vue étendue : une seule application utilise tout l'écran étendu (*Voir* Figure 2.3).



a) Plusieurs applications ainsi que des notifications détaillées sont affichées dans la fenêtre virtuelle en parallèle à celle sur l'écran physique.

b) Wrist : une rotation de la main permet de rapidement changer l'application sur l'écran physique.

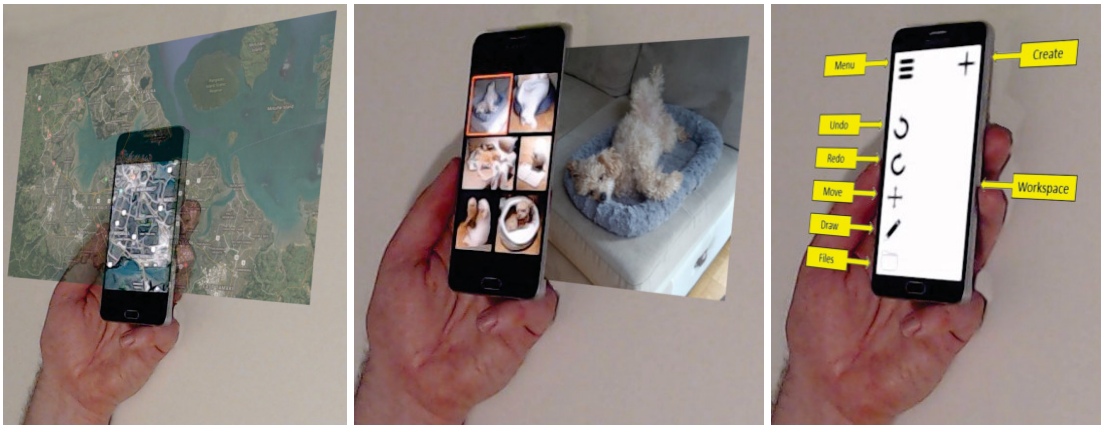
Figure 2.2 Photomontages d'un téléphone à écran étendu en vue multi-fenêtres, où de multiples applications se partagent l'écran étendu.

En vue multi-fenêtres, l'utilisateur peut donc interagir avec de multiples applications en même temps comme sur un ordinateur, qu'il organise librement, ou de manière plus structurée selon une grille, sur l'écran étendu. L'écran d'accueil du téléphone pourrait ainsi montrer l'ensemble

des applications installées, ou en cours d'exécution (*Voir* Figure 2.2a) permettant à un utilisateur de facilement changer l'application affichée sur l'écran physique. Un autre cas d'utilisation consiste en des prises de notes d'un cours ou d'une conférence : autour d'une application de notes sur l'écran physique, l'utilisateur pourrait placer la vidéo de l'intervention ainsi que les diapositives sur la droite de l'écran, et plusieurs onglets de pages web ou de références pour compléments d'informations sur la gauche. Il peut alors aller et venir d'un regard de ces notes, à la vidéo ou aux références, comme on peut le faire sur un ordinateur.

La fenêtre virtuelle peut également être utilisée par le système d'exploitation du téléphone par exemple pour afficher des informations plus détaillées que sur la barre de notifications en haut de l'écran, ou la barre des tâches d'un ordinateur : par exemple le nom et le sujet des messages et courriels non-lus, une liste des prochains événements du calendrier ou un graphique de l'état de la batterie depuis le dernier chargement (*Voir* Figure 2.2a). La zone en bas de l'écran peut, elle, être mise à profit pour visualiser la liste des données (texte, images) copiées par l'utilisateur et lui permettre de facilement sélectionner ce qu'il veut coller ; cette opération est difficile avec les IHMs des téléphones actuels. Enfin, en exploitant la 3D – en débordant donc du cadre d'une fenêtre virtuelle – les notifications pourraient s'accumuler en piles à l'arrière du téléphone : une rotation similaire à la Figure 2.2b permettrait alors de les visualiser complètement.

En vue étendue, une application occupe tout l'espace d'affichage. L'utilisateur peut souhaiter, par exemple, avoir une vue plein écran d'une application, une page web ou une carte de navigation en exploitant la totalité de l'écran étendu (*Voir* Figure 2.3a). Cela permet de visualiser un large document ou une version pour ordinateur dans le cas d'une page web. Une application peut également utiliser l'écran étendu pour structurer son information. Par exemple, une application de messagerie afficherait la conversation courante sur l'écran physique tandis que la liste de fichiers échangés dans cette conversation ainsi que la liste des contacts seraient placés au-



a) Une carte de navigation. b) Une galerie d'images. c) Info-bulles dans la fenêtre virtuelle.

Figure 2.3 Photomontages d'applications en vue étendue, exploitant tout l'écran étendu du VESAD.

tour, dans la partie virtuelle de l'écran étendu. Enfin, un navigateur web pourrait afficher autour de l'onglet courant sur l'écran physique la liste des onglets ouverts dans la fenêtre virtuelle : les gestes avec un doigt agissent sur l'écran physique tandis que ceux à deux doigts agissent sur l'écran étendu, par exemple pour faire défiler de gauche à droite les onglets ouverts, vers le bas pour fermer le courant, vers le haut pour le mettre en favoris ; des interactions directes avec une main virtuelle seraient aussi possibles.

Une application peut également se servir de la fenêtre virtuelle pour afficher des informations en relation avec l'écran physique, comme des infobulles d'aide (*Voir* Figure 2.3c), des commentaires sur un document PDF ou encore des interfaces de contrôle d'une vidéo ou d'une musique jouée. À l'inverse, Grubert *et al.* (2015) proposent d'utiliser un clavier sur tout l'écran physique en affichant le texte tapé dans une fenêtre virtuelle.

Enfin, on peut mettre en œuvre des techniques de visualisation comme Overview+Detail sur l'écran étendu. La vue détaillée peut être affichée sur l'écran physique ou sur la fenêtre virtuelle. Par exemple, la Figure 2.3b montre une image dans la fenêtre virtuelle (*detail*) sélec-

tionnée depuis une galerie d'images affichée sur l'écran physique (*overview*). Ces deux vues peuvent donc être inversées, par exemple pour profiter de la haute définition de l'écran physique.

2.3 Techniques d'interactions

Une de nos questions de recherche (*Voir* section 1.6) est quelle technique d'interaction est la plus adaptée pour un téléphone étendu par un VESAD. On peut en effet penser à différentes techniques qui ont été explorées dans la littérature pour interagir avec telle IHM :

- Pointer directement le contenu avec une main virtuelle, utilisé par le Personal Cockpit (Ens *et al.*, 2014b) et exploré par Piumsomboon *et al.* (2013).
- Utiliser l'écran tactile du téléphone, comme dans la condition *Smartwatch referenced* de MultiFi (Grubert *et al.*, 2015) ou dans une application ARCore ou ARKit.
- Utiliser un pointeur virtuel et des gestes de sélection (Wilson, 2006), comme sur un HoloLens.
- Via des commandes vocales, évaluées par Piumsomboon *et al.* (2014).

Cette question est ouverte car elle n'a été étudié que par Grubert *et al.* (2015). Or les concepts de Serrano *et al.* (2015a) puis Serrano *et al.* (2015b), dont nous prenons la suite avec le VESAD, proposent de laisser à l'utilisateur d'utiliser comme il le souhaite les dispositifs d'entrées et sorties disponibles. Comme nous évaluons le cas particulier d'un téléphone à écran étendu, nous souhaitons donc savoir si une technique d'interaction semble la plus adaptée.

Nous avons vu dans le chapitre 1 que le pointage direct utilisant une main virtuelle était difficile (Cha *et al.*, 2010; Argelaguet & Andujar, 2013). En particulier, Ens *et al.* (2014b) ont montré qu'une main virtuelle était plus stable et moins sujette aux erreurs avec des fenêtres virtuelles placées dans un cadre exocentrique plutôt qu'égocentrique. Un téléphone à écran étendu étant

tenu en main, on peut considérer que la fenêtre est alors placée dans l'environnement de manière stable, même si mobile, plutôt que par rapport au corps de l'utilisateur.

Figure 2.4 Un geste Slide to Hang permet à un utilisateur de détacher une application de l'écran étendu pour la fixer en l'air, contre une surface ou relativement à son corps. Dans ce photomontage, un utilisateur détache dans un premier temps une page web en l'air, puis un lecteur vidéo.

En outre, nous présentons deux nouvelles techniques d'interactions pour notre IHM :

1. Wrist : un mouvement de la main tenant le téléphone permet de changer rapidement l'affichage sur l'écran physique dans une vue multi-fenêtres. La Figure 2.2b montre par exemple l'utilisateur tournant son téléphone à 90° environ vers la droite pour changer l'application affichée sur l'écran physique. Un pointeur virtuel suivant le regard ou la tête de l'utilisateur permettrait de choisir l'application sur l'écran virtuel à utiliser, par exemple parmi une liste d'icônes d'applications.

2. Slide-to-hang : un geste de glissement permet de détacher une fenêtre de l'écran étendu et la fixer en l'air. La Figure 2.4 montre l'utilisateur faisant un geste de glissement (*slide* en anglais) sur le téléphone pour fixer en l'air la fenêtre qui y est affichée. La fenêtre peut être alors placée à un endroit fixe sur un mur ou en l'air, comme avec le HoloLens, ou relativement à son propre corps, comme dans le Personal Cockpit (Ens *et al.*, 2014b). Ce

geste de glissement peut se faire avec deux doigts simultanément sur le téléphone pour le distinguer d'actions faites sur l'application ou encore avec un pincement des doigts sur l'écran virtuel (Piumsomboon *et al.*, 2014). Un geste de pincement similaire ramenant la fenêtre fixée vers l'écran étendu permettrait de la rendre à nouveau relative au téléphone. Cette technique est particulièrement importante, car elle permet de déplacer du contenu de l'écran étendu vers des fenêtres dans l'environnement, et donc de rendre compatible le VESAD avec les autres modes du Tableau 2.1.

CHAPITRE 3

CONCEPTION D'UN VISIOCASQUE DE RA À LARGE CHAMP DE VISION

3.1 Motivation

Pour concevoir un VESAD, nous avons besoin d'un visiocasque avec un champ de vision suffisamment large pour visualiser complètement l'écran étendu, le cas contraire limite l'intérêt de notre concept. Notre second sous-problème a donc été de développer un visiocasque de RA à large champ de vision.

Un téléphone est tenu en moyenne à une distance $D = 34\,\text{cm}$ (Bababekova *et al.*, 2011) de la tête. Ainsi, si l'on souhaite étendre un téléphone à l'équivalent d'un écran 16:9 de 24 po, soit une taille de $(L,H) = 53\,\text{cm} \times 30\,\text{cm}$, on peut calculer le champ de vision (FoV_x, FoV_y) nécessaire pour chaque œil pour le visualiser complètement (*Voir* Figure 3.1), avec l'Équation 3.1 : il doit être d'au moins $76° \times 48°$.

$$\begin{cases} FoV_x &= 2\arctan(\frac{L}{2\times D}) &= 2\arctan(\frac{53}{2\times 34}) &= 76° \\ FoV_y &= 2\arctan(\frac{H}{2\times D}) &= 2\arctan(\frac{30}{2\times 34}) &= 48° \end{cases} \tag{3.1}$$

Nous avons d'abord voulu utiliser le Microsoft HoloLens, un visiocasque optique de RA avec une très courte latence, une excellente résolution et une très bonne documentation et un support natif sur le moteur de jeu Unity. Cependant, son champ de vision de $30° \times 17{,}5°$ pour chaque œil (Kreylos, 2015) est trop restreint, ne permettant de visualiser seulement l'équivalent d'un écran 16:9 de 7 po. Il n'existe en fait actuellement aucun visiocasque de RA sur le marché avec un grand champ de vision (Millette, 2016, p. 25).

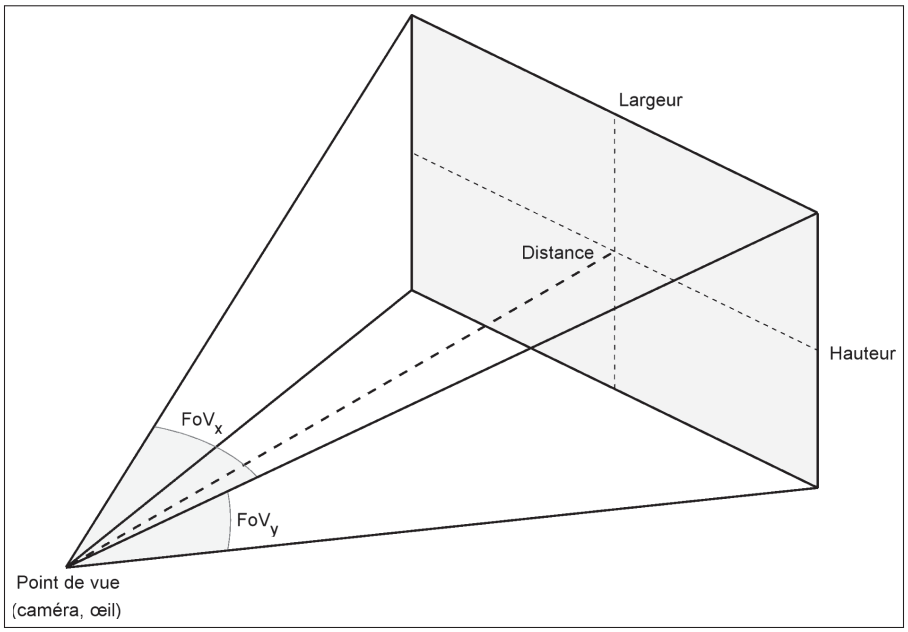


Figure 3.1 Représentation simplifiée du champ de vision (en anglais : *Field of View*), qui peut être mesuré en degrés horizontalement (FoV_x), verticalement (FoV_y) ou en diagonale. Comme le décrit l'Équation 3.1, pour qu'un plan soit visible dans un champ de vision donné, il doit être placé à une certaine distance de la caméra ou de l'œil.

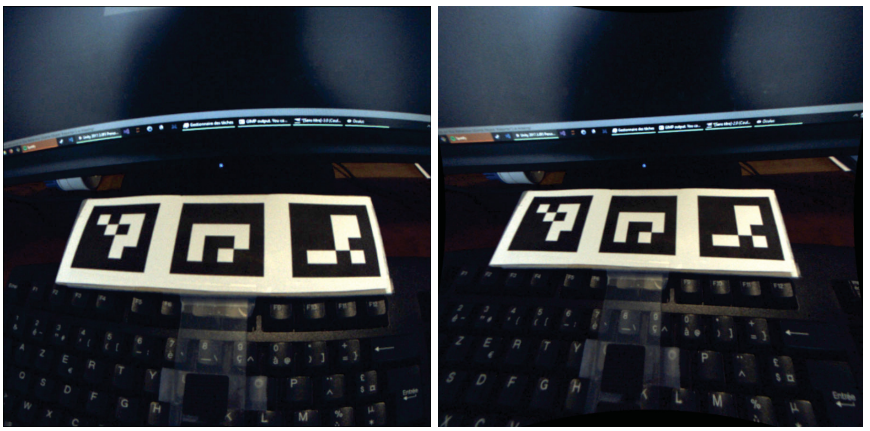
3.2 Solution retenue

3.2.1 Fonctionnement du visiocasque

Nous avons donc réalisé notre propre prototype, similaire à l'AR-Rift (Steptoe, 2013) : c'est un visiocasque de RA vidéo de conception simple qui a fait ses preuves, utilisé par Steptoe *et al.* (2014) et Piumsomboon *et al.* (2014). Il consiste à coller une caméra physique *stéréoscopique* (avec deux objectifs capturant chacun une image pour un œil) à un visiocasque de RV.

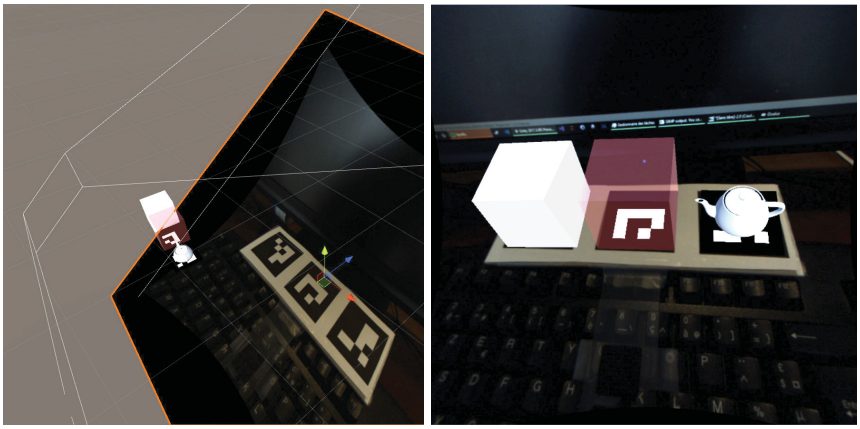
Notre visiocasque fonctionne sur le principe suivant :

1. La caméra stéréoscopique physique filme l'environnement réel de l'utilisateur (*Voir* Figure 3.2a).
2. Un algorithme *corrige* les déformations des deux images de cette caméra (*Voir* Figure 3.2b).



a) Image non corrigée : l'image présente des distorsions importantes (les lignes droites sont courbées).

b) Image corrigée : les lignes droites le sont à nouveau.



c) Image corrigée à travers Unity : l'image est affichée en arrière-plan de l'environnement virtuel filmé par la caméra virtuelle (champ de vision en lignes blanches) pour l'œil gauche. Les caméras physique et virtuelle sont alignées pour faire coïncider environnements virtuels et réels.

d) Image corrigée et augmentée : un objet virtuel se trouve par dessus chacun des trois marqueurs.

Figure 3.2 Vue de l'image capturée par l'objectif gauche de l'Ovrvision Pro, puis corrigée et augmentée pour être affichée dans le visiocasque.

3. Une caméra stéréoscopique virtuelle va filmer des éléments virtuels ainsi que les deux images de la caméra physique en arrière-plan (*Voir* Figure 3.2c).

4. Les deux images sont affichées dans le visiocasque, une pour chaque œil, affichant environnements virtuels et physiques superposés (*Voir* Figure 3.2d).

Pour donner l'illusion que les éléments virtuels sont alignés avec l'environnement réel, nous devons donc *aligner* la caméra virtuelle avec la caméra physique, dans un travail qui se fait en trois temps :

1. *Étalonner* (*calibrate* en anglais) la caméra physique, c'est-à-dire mesurer ses *paramètres intrinsèques* (ses propriétés, comme son champ de vision), et les *distorsions* de ses objectifs (les déformations sur les images capturées). L'étalonnage est à faire seulement une seule fois par caméra.
2. Au démarrage du visiocasque, aligner la caméra virtuelle en la configurant avec les paramètres intrinsèques de la caméra physique.
3. Pour chaque image capturée par la caméra physique, appliquer le procédé de la Figure 3.2.

La caméra virtuelle filmant ainsi du *même point de vue* que la caméra physique, on peut donc placer les éléments virtuels comme s'ils faisaient partie de l'environnement réel. On utilise alors un algorithme de *suivi de marqueurs* (*fiducial marker tracking* en anglais) : ce sont des codes-barres en 2D imprimés (*Voir* Figure 3.2b) et dont on peut déterminer très rapidement la position et l'orientation par rapport une caméra physique dont on connaît les paramètres intrinsèques Garrido-Jurado *et al.* (2014). Les marqueurs étant détectés en temps réel sur chaque image, on place les éléments virtuels à ces mêmes positions et orientations par rapport à la caméra virtuelle (*Voir* Figure 3.2c), donnant donc l'illusion qu'ils font partie de l'environnement réel (*Voir* Figure 3.2d).

La Figure 3.2 montre ce processus pour une caméra *monoscopique* (avec un seul objectif). Pour une caméra stéréoscopique, on mesure en plus la différence de position et d'orientation entre les deux objectifs lors de l'étalonnage. On crée ensuite deux caméras monoscopiques virtuelles placées de la même manière que les deux objectifs de la caméra physique. Chaque

caméra virtuelle va alors filmer le même environnement virtuel, d'un point de vue légèrement différent, mais avec seulement une image d'un des deux objectifs en arrière-plan.

3.2.2 Choix techniques

Nous avons tout d'abord choisi l'Ovrvision Pro comme caméra physique stéréoscopique (http://ovrvision.com/). Annoncée par son constructeur comme solution clé en main pour réaliser un visiocasque de RA, elle est prévue pour fonctionner avec le visiocasque de RV Oculus DK2 que nous avions déjà à disposition dans notre laboratoire. Composée de deux objectifs *fisheye* (en œil de poisson) d'une définition de 960 px $\times$ 950 px et un champ de vision de 100° $\times$ 98°, ses caractéristiques correspondent bien à ceux de l'Oculus DK2 (*Voir* Tableau 3.1). Elle est de plus intégrée avec les moteurs de jeu standards dans l'industrie : Unity et Unreal Engine. Son installation se fait simplement en la collant sur la face avant de l'Oculus DK2, comme le montre la Figure 3.3, et en le connectant par USB 3.0 au PC.

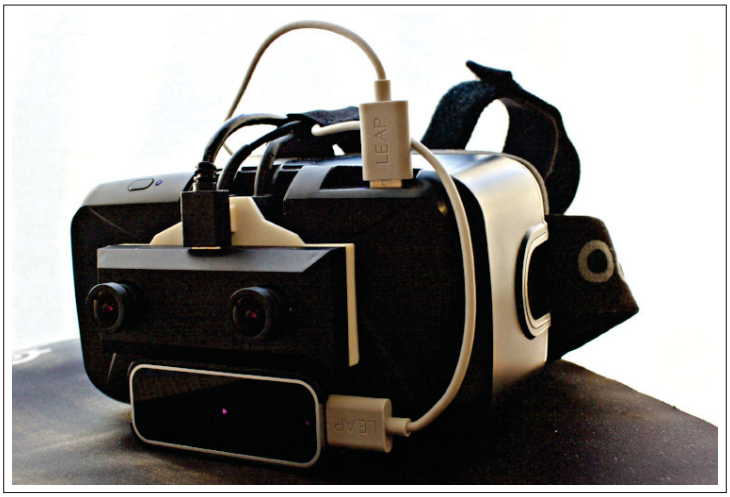


Figure 3.3 Notre prototype de visiocasque de RA, basé sur le concept de l'AR-Rift de Steptoe (2013) : il est composé du visiocasque de RV Oculus DK2, de la caméra stéréoscopique Ovrvision Pro et du capteur de reconnaissance des mains Leap Motion (sous la caméra).

Nous souhaitions ensuite rapidement prototyper du contenu sur notre visiocasque. Nous avons alors choisi le moteur de jeu Unity : gratuit (mais au code source propriétaire), il est le standard dans l'industrie du jeu-vidéo pour prototyper et supporte notre caméra et le Leap Motion. Il a été, de fait, très avantageux d'investir un peu de temps dans l'apprentissage de ce moteur de jeu, car il prend complètement en charge le rendu 3D, la simulation de la physique, les entrées sur clavier, souris ou écran tactile, l'affichage dans les visiocasques de RV, un support réseau et propose de nombreuses fonctions mathématiques. Unity est plus simple à prendre en main que son concurrent l'Unreal Engine, par son GUI intuitif et l'utilisation de scripts C#, un langage haut niveau efficace. Enfin, nous avions une expertise dans le laboratoire avec un projet de quatre mois sur le Microsoft HoloLens et la maîtrise de Millette (2016).

Après plusieurs essais de configuration et une lecture du code source de l'Ovrvision Pro (https://github.com/Wizapply/OvrvisionPro/), nous avons constaté que la bibliothèque fournie était non fonctionnelle : (1) les images affichées pour chaque œil dans le casque étaient décalées, rendant le visiocasque particulièrement inconfortable à utiliser et (2) un décalage important était présent entre le contenu virtuel et l'environnement réel. Nous avons donc besoin de ré-étalonner cette caméra et d'une bibliothèque de RA.

Nous avons donc réalisé la bibliothèque libre de RA par suivi de marqueur ArucoUnity (https://github.com/NormandErwan/ArucoUnity/) pour amener de la RA sur notre visiocasque. Utilisant la bibliothèque libre de vision par ordinateur OpenCV (https://opencv.org/), elle est la seule bibliothèque libre de RA par suivi de marqueurs sous Unity supportant plusieurs types de caméras : monoscopiques ou stéréoscopiques, avec des objectifs *fisheye* ou « classiques », dit *rectilinéaires* (similaire à la vision humaine). Elle est également la seule à permettre l'étalonnage de tout ces types de caméras. On utilise en particulier les modules d'OpenCV suivants :

- aruco pour suivre les marqueurs (https://docs.opencv.org/3.4/d9/d6a/group__aruco.html) ;

- calib3d pour étalonner et corriger les objectifs rectilinéaires (https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html) ;
- ccalib pour étalonner et corriger les objectifs *fisheye* (https://docs.opencv.org/3.4/d3/ddc/group__ccalib.html).

3.2.3 Discussion et limites

Nous pourrions simplifier le visiocasque en utilisant une caméra monoscopique, et afficher l'image capturée aux deux yeux. Cependant, comme le souligne Bourke (1999), la vision stéréoscopique est l'un des principaux indices utilisés par le cerveau pour percevoir la profondeur : les yeux étant séparés horizontalement par une distance appelée écart pupillaire, chaque œil perçoit une image légèrement différente (*Voir* Figure 3.4) permettant au cerveau de percevoir en 3D. C'est pourquoi nous choisissons d'utiliser une caméra stéréoscopique.

a) Vue de l'objectif gauche. b) Vue de l'objectif droit.

Figure 3.4 Images corrigées de l'Ovrvision Pro : les deux objectifs sont décalés horizontalement d'environ 60 mm pour simuler un écart pupillaire humain.

Ensuite, si notre visiocasque est relativement simple à concevoir, son principal inconvénient est la faible densité visuelle de l'image affichée. La densité visuelle permet de mesurer la finesse d'affichage d'une image à partir de sa définition horizontale et du champ de vision : $Densite = {}^{d_x}/_{FoV_x}$. Ainsi, sur les visiocasques de RV du marché et sur l'Ovrvision Pro, la ca-

méra stéréoscopique que nous utilisons, elle est limitée à environ $10\,\text{px}/^\circ$ (pixels par degré) (*Voir* Tableau 3.1). En comparaison, le Microsoft HoloLens a une densité visuelle d'environ $42\,\text{px}/^\circ$ quand la fovéa d'un œil humain est de $60\,\text{px}/^\circ$ à $80\,\text{px}/^\circ$ en moyenne (Kistner, 2014). Nous sommes donc face à une limite technologique : une meilleure qualité des images des caméras serait limitée par le visiocasque de RV, quelque qu'il soit. Une définition de 8K ($7680\,\text{px} \times 4320\,\text{px}$), à champ de vision égal, par œil est alors nécessaire pour atteindre la même densité visuelle : $Densite_{8K} = {}^{7680}/_{110} = 69{,}8\,\text{px}/^\circ$.

Tableau 3.1 Caractéristiques d'affichage de l'Ovrvision Pro et de visiocasques de RV et RA.

Nom	**Définition (pour chaque œil)**	**Champ de vision (pour chaque œil)**	**Densité visuelle**
Ovrvision Pro	$960\,\text{px} \times 950\,\text{px}$	$100^\circ \times 98^\circ$	$9{,}6\,\text{px}/^\circ$
Oculus DK2	$960\,\text{px} \times 1080\,\text{px}$	$94^\circ \times 105^\circ$	$10{,}2\,\text{px}/^\circ$
Oculus Rift	$1080\,\text{px} \times 1200\,\text{px}$	$94^\circ \times 93^\circ$	$11{,}5\,\text{px}/^\circ$
HTC Vive	$1080\,\text{px} \times 1200\,\text{px}$	$110^\circ \times 113^\circ$	$9{,}8\,\text{px}/^\circ$
Microsoft HoloLens	$1268\,\text{px} \times 720\,\text{px}$	$30^\circ \times 17{,}5^\circ$	$42{,}3\,\text{px}/^\circ$

Une seconde limite évidente est la portabilité de notre visiocasque (*Voir* Figure 3.5) : il est volumineux et dépendant d'un PC demandant une bonne puissance de calcul. Cela reste un prototype de laboratoire qui convient à notre recherche, mais un visiocasque pour usage professionnel voire personnel devra dépasser ces problématiques. Correctement optimisé, un visiocasque de RA ne demande pas énormément de ressources : le Microsoft HoloLens est portable et a une très bonne autonomie, tout comme les nouvelles versions de 2018 des visiocasques de RV Oculus Rift et HTC Vive. De même, les objectifs de la caméra peuvent être miniaturisés et intégrés dans le visiocasque, comme le fait le HoloLens déjà.

En outre, notre visiocasque souffre également d'un flux vidéo peu fluide à 30 images par secondes (*frames per second* ou FPS en anglais). Nous sommes limités artificiellement à cette vitesse par le visiocasque de RV Oculus que nous utilisons : il impose en effet aux applications

Figure 3.5 Notre prototype de visiocasque de RA porté par un utilisateur : il est assez volumineux et n'est pas portable. Un câble à l'arrière de la tête de l'utilisateur relie le visiocasque au PC. Des marqueurs infrarouges en bas du visiocasque sont illuminés.

d'atteindre 60 FPS mais les bride dans le cas contraire. Dans notre cas, la capture des images de l'Ovrvision puis la correction des distorsions ainsi que le suivi de marqueurs sont des opérations trop lourdes en calculs pour le processeur pour atteindre 60 FPS. Les performances étaient plus faibles encore à l'origine ; nous décrivons dans la sous-section 3.3.4 quelques optimisations que nous avons mis en place. 30 FPS nous semblant tout de même acceptable (les participants n'auront pas besoin de bouger la tête pour visualiser le VESAD), nous n'avons pas approfondi nos optimisations.

Enfin, l'utilisation d'une solution de RA par suivi de marqueurs est bonne pour du prototypage, car elle est techniquement simple à mettre en œuvre, mais elle est inadéquate pour un produit industriel ou grand public. En effet, le placement de marqueurs sur les objets que l'on veut détecter, comme un téléphone, est une étape fastidieuse. C'est pourquoi le Microsoft HoloLens et les bibliothèques ARKit pour les téléphones iOS et ARCore pour les téléphones Android préfèrent utiliser des technologies de localisation et cartographie simultanées, ou *Simultaneous Localisation And Mapping* (SLAM) en anglais. Un algorithme de SLAM permet à l'appareil

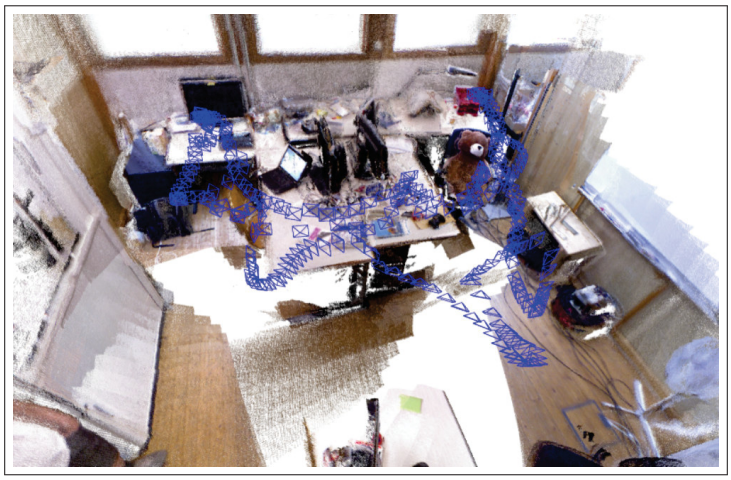


Figure 3.6 Carte d'une pièce par localisation et cartographie simultanées (SLAM) et les différentes positions passées, en bleu, de la caméra. Dressée en temps réel, cette carte permet à la caméra de s'y situer et d'y intégrer de la RA.
Tiré de Mur-Artal & Tardós (2017).

de se situer en temps réel dans son environnement et de l'augmenter en dressant une carte de cette pièce (*Voir* Figure 3.6). Pourtant cette méthode n'est pas adaptée pour détecter de petits objets en mouvement, comme un téléphone, nous ne pouvions alors pas l'utiliser.

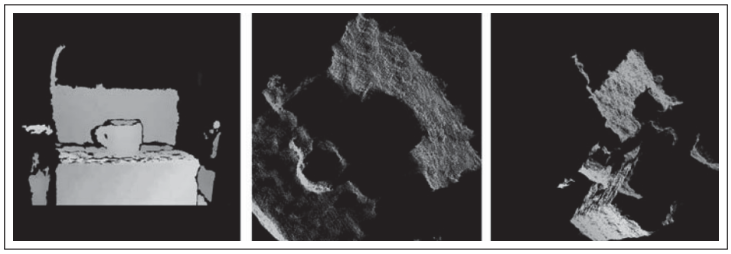


Figure 3.7 Carte de profondeur d'une tasse sur une chaise, reconstituée par vision stéréoscopique.
Adapté de Bradski & Kaehler (2008).

D'autres alternatives existent. Nous aurions pu par exemple nous appuyer sur la différence entre les deux images de la caméra pour faire de la vision stéréoscopique : de manière similaire à une méthode SLAM, on peut construire une carte de profondeur de ce qui est vu par la caméra (*Voir* Figure 3.7). Cette méthode nous a cependant semblé trop imprécise. Une

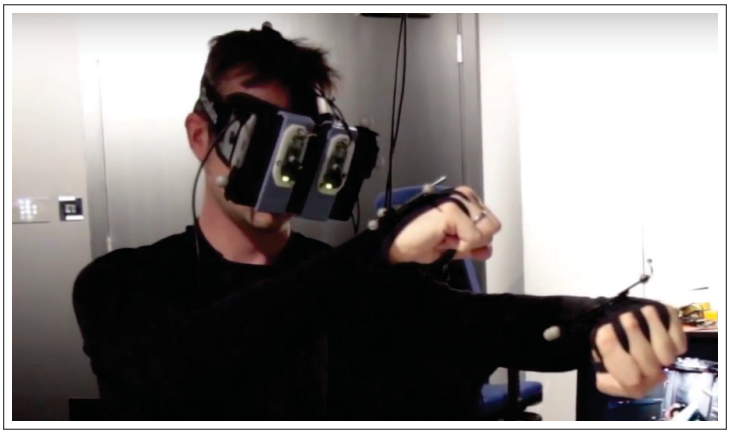


Figure 3.8 Visiocasque de Steptoe : deux caméras sont collées sur un visiocasque de RV, et des marqueurs infrarouges sous forme de boules grises permettent de suivre la position et l'orientation du visiocasque et de sa main. Adapté de Steptoe (2013).

troisième alternative serait d'utiliser des marqueurs infrarouges, comme sur les visiocasques de RV. Une ou deux caméras infrarouges filment le visiocasque contenant de nombreux marqueurs insérés sous sa coque (*Voir* Figure 3.5) et, connaissant la structure 3D de ces marqueurs, on détermine à partir des images précisément la position et l'orientation du visiocasque. Des systèmes équivalents, dit de capture de mouvements (*motion capture*), comme Vicon, MotionAnalysis ou OptiTrack sont excellents pour prototyper tout en conservant une précision inférieure au millimètre (*Voir* Figure 3.8). Étant malheureusement très chers, nous avons également écarté ces solutions. Enfin, Millette (2016) avait utilisé dans notre laboratoire le Polhemus Patriot (https://polhemus.com/motion-tracking/all-trackers/patriot), un système de suivi électromagnétique, mais qui s'est trouvé fortement imprécis s'il était placé trop près d'un téléphone.

3.3 Réalisation de la bibliothèque de réalité augmentée ArucoUnity

3.3.1 Motivation

Unity est un moteur de jeux-vidéos ainsi qu'une excellente solution pour développer avec des visiocasques de RV. Cependant, nous avons besoin de fonctionnalités manquantes pour permettre de la RA avec notre visiocasque. La bibliothèque OpenCV répond parfaitement à nos besoins, mais étant écrite en C++, elle est incompatible avec Unity utilisant le C#. Nous avons donc réalisé la bibliothèque ArucoUnity pour rendre disponible OpenCV dans Unity. Son utilisation est décrite dans la section 3.4.

La solution que nous avons retenue est d'utiliser le système de greffons (*plugin* en anglais, https://docs.unity3d.com/Manual/Plugins.html) d'Unity pour appeler des bibliothèques externes en C# ou en C. Cette technique, appelée *liaison* (*binding* en anglais) est classique dans les domaines des jeux-vidéos et des simulations physiques : elle permet de réutiliser du code existant sans avoir à le transposer dans un autre langage et surtout d'optimiser des portions de code critiques, le C++ étant conçu pour la performance.

Le code source d'OpenCV étant libre, une alternative aurait pu être de ré-écrire en C# les fonctionnalités dont nous avions besoin. Le code source du module aruco nous semblait assez court pour cela, étant lui-même une ré-implémentation propre de la bibliothèque originale (https://www.uco.es/investiga/grupos/ava/node/26) proposée par Garrido-Jurado *et al.* (2014). Une réécriture des fonctions d'étalonnage et de correction de caméra nous a semblé par contre trop risquée, n'ayant pas une bonne maîtrise de la théorie mis en œuvre De plus, de nombreux projets de liaison existent vers d'autres langages comme Python (https://docs.opencv.org/master/d6/d00/tutorial_py_root.html), Javascript (https://docs.opencv.org/master/d5/d10/tutorial_js_root.html) ou Java (https://opencv.org/platforms/android/). L'écriture d'une liaison de C++ vers C# semble donc plus sûre qu'une ré-écriture.

Enfin, quelques bibliothèques de RA par suivi de marqueurs sont déjà disponibles pour Unity, mais aucune ne nous a convenu :

- Vuforia (https://unity3d.com/fr/partners/vuforia) ne supporte pas notre caméra et le code source est propriétaire donc non modifiable.
- ARToolKit (https://github.com/artoolkit/arunity5) ne supporte pas notre caméra non plus, mais malgré un code source libre, le projet est abandonné et il semble difficile d'y ajouter les fonctions d'étalonnage pour une caméra stéréoscopique avec objectifs *fisheye*.
- OpenCV for Unity (https://enoxsoftware.com/opencvforunity/) propose des liaisons vers tous les modules d'OpenCV dont nous avons besoin, mais nous craignons de manquer de flexibilité avec son code source propriétaire si nous devons apporter des modifications dans le code source d'OpenCV.

3.3.2 Architecture

ArucoUnity est composé de trois couches logicielles, s'appuyant sur OpenCV (*Voir* Figure 3.9). Tout d'abord, nous avons créé une bibliothèque exposant les modules aruco, calib3d et ccalib d'OpenCV avec une interface en C (https://github.com/NormandErwan/ArucoUnityPlugin/), que l'on peut appeler depuis C#. Nous avons ensuite créé un projet Unity et placé la bibliothèque dans le dossier `Assets/Plugin` : elle est alors détectée comme un greffon par Unity (*Voir* Figure 3.10).

Nous avons ensuite écrit les classes de liaison dans ce projet (https://github.com/NormandErwan/ArucoUnity/tree/master/Assets/ArucoUnity/Scripts/Plugin/) : elles reproduisent en C# l'interface orientée objet d'OpenCV en appelant en arrière-plan notre greffon. On peut alors utiliser en toutes les fonctionnalités que nous avons portées d'OpenCV dans Unity de manière similaire qu'en C++.

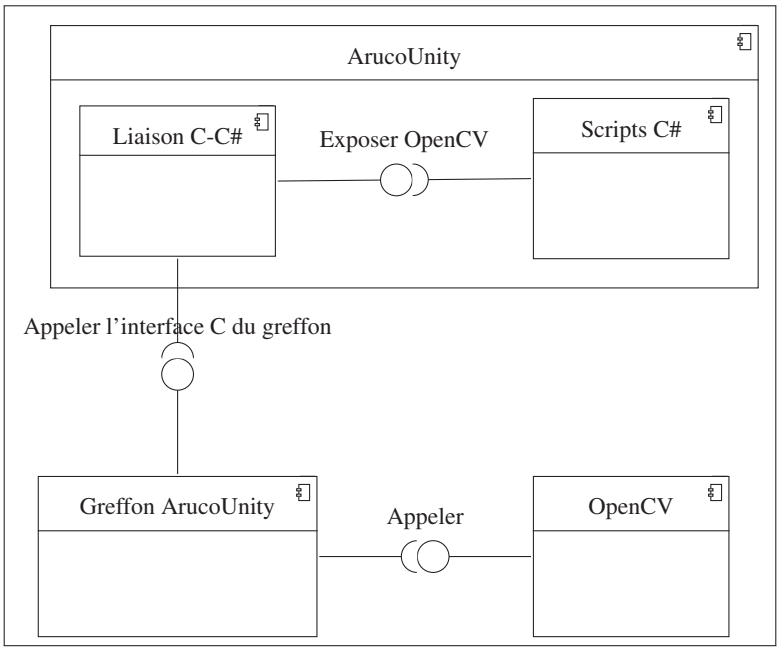


Figure 3.9 Diagramme des composants d'ArucoUnity. Le greffon enveloppe OpenCV dans une interface en C. Il est appelé par la liaison qui expose alors une interface orientée objet en C# similaire à celle d'OpenCV. Les scripts permettent de travailler directement dans l'interface d'Unity avec les fonctionnalités d'OpenCV sans avoir à programmer.

Enfin, nous avons écrit des scripts C# pour Unity que nous utilisons dans la section 3.4. Nous avions d'abord écrit de simples scripts séquentiels d'étalonnage et de suivi de marqueur, mais le besoin s'est vite faite sentir de les rendre plus robustes et performants. De plus, le travail dans Unity se fait préférentiellement en deux temps : les développeurs programment les fonctionnalités dans les scripts, et les artistes les utilisent et configurent à volonté avec l'interface graphique d'Unity sans avoir à toucher au code source. Notre bibliothèque ayant demandé un certain effort de développement, faciliter son usage nous permet d'encourager sa réutilisation pour des équipes souhaitant travailler sur des problématiques similaires.

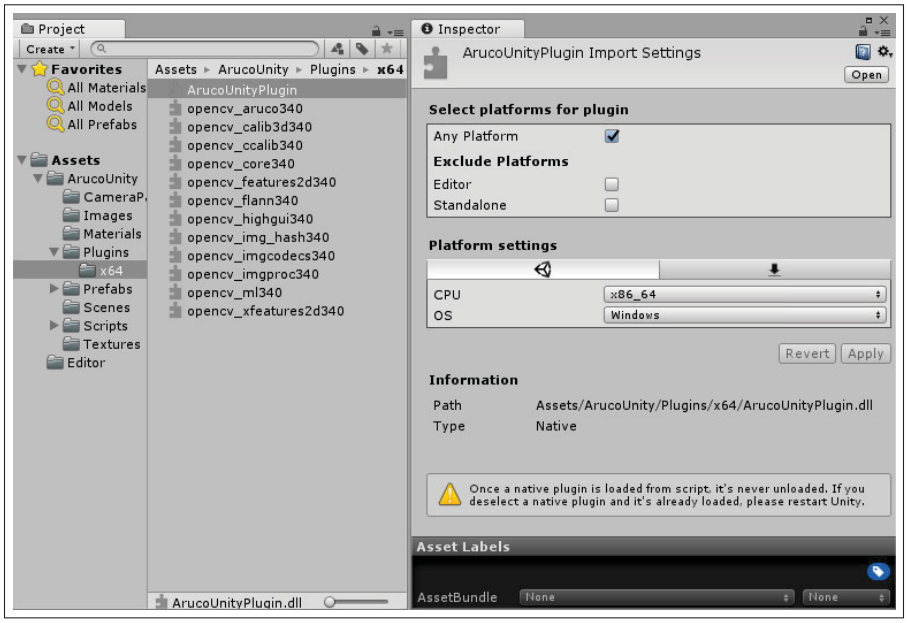


Figure 3.10 Le greffon d'ArucoUnity exposant les modules aruco, calib3d et ccalib d'OpenCV avec une interface en C dans le projet Unity d'ArucoUnity. Les autres modules présents sont des dépendances.

3.3.3 Réalisation du greffon et de la liaison

La première étape de réalisation a donc été le greffon pour Unity, pour rendre accessible les fonctions des modules aruco, calib3d et ccalib d'OpenCV. Il existe plusieurs techniques de liaison. Nous choisissons la technique la plus simple : P/Invoke (*Platform Invocation Services*). Le principal défi dans son utilisation est la rigueur de la gestion de la mémoire : d'une part on gère manuellement la création et la destruction d'objets et d'autre part le greffon et les scripts doivent se partager cette mémoire.

L'Extrait 3.1 montre un exemple simplifié de l'interface en C la classe `cv::Mat`, qui permet de manipuler matrices et images dans OpenCV. On utilise en fait le patron de conception adaptateur : chaque constructeur, destructeur et fonction est enveloppé par une fonction C, que l'on nomme `new` pour les constructeurs, `delete` pour les destructeurs ou avec le nom de la fonction appelée. Le polymorphisme n'étant pas supporté en C, chaque surcharge est également enve-

Extrait 3.1 Interface simplifiée en C autour de la classe Mat d'OpenCV dans le greffon d'ArucoUnity.

```
extern "C" {
  cv::Mat* au_cv_Mat_new1() {
    return new cv::Mat();
  }

  cv::Mat* au_cv_Mat_new2(int rows, int cols, int type) {
    return new cv::Mat(rows, cols, type);
  }

  void au_cv_Mat_delete(cv::Mat* mat) {
    delete mat;
  }

  int au_cv_Mat_total(cv::Mat* mat) {
    return mat->total();
  }
}
```

loppée par une fonction : on les numérote alors avec un suffixe de 1 à n. Le C ne comportant pas non plus d'espace de nom, on ajoute à chaque fonction le préfixe `au_cv_` suivi du nom de la classe pour éviter des collisions de noms. Enfin, on entoure toutes les fonctions avec l'instruction `extern "C"` pour indiquer au compilateur et à l'éditeur de liens que nous travaillons bien avec des fonctions en C.

Le passage de paramètres est la problématique principale avec cette technique. Certains types primitifs de C# comme `int` ou `float` ont des équivalences en C++, on peut alors les passer directement en paramètres au greffon. Cependant, d'autres types plus complexes sont représentés différemment dans les deux langages, par exemple les classes, les chaînes de caractères Peterson (2015). Nous avons alors voulu faire simple : tous les objets OpenCV sont alloués avec le code C++, via les fonctions C pour la construction et la destruction que nous venons de décrire. Elles retournent alors un pointeur (qui n'est qu'un entier sur une adresse mémoire) au code C#. Quand on souhaite exécuter une fonction sur un objet, il suffit de simplement y passer son pointeur comme paramètre. Sans rigueur, il y a plus de risques d'erreurs de segmentation

ou de fuites de mémoire mais notre couche de liaison prend en charge cette responsabilité sans que l'utilisateur n'ait à s'en soucier.

Extrait 3.2 Liaison simplifiée pour exposer en C# la classe Mat d'OpenCV.

```
public class Mat
{
  [DllImport("ArucoUnityPlugin")]
  static extern IntPtr au_cv_Mat_new1();

  [DllImport("ArucoUnityPlugin")]
  static extern IntPtr au_cv_Mat_new2(int rows, int cols, int type);

  [DllImport("ArucoUnityPlugin")]
  static extern void au_cv_Mat_delete(IntPtr mat);

  [DllImport("ArucoUnityPlugin")]
  static extern int au_cv_Mat_total(IntPtr mat);

  public IntPtr Ptr { get; }

  public Mat()
  {
    Ptr = au_cv_Mat_new1();
  }

  public Mat(int rows, int cols, int type)
  {
    Ptr = au_cv_Mat_new2(rows, cols, type);
  }

  ~Mat()
  {
    au_cv_Mat_delete(Ptr);
  }

  public int Total()
  {
    return au_cv_Mat_total(Ptr);
  }
}
```

L'Extrait 3.2 montre la liaison associée à l'interface de l'Extrait 3.1. Elle est composée de deux parties. Il y a d'abord les appels au greffon : on déclare des fonctions statiques avec la même signature (nom et paramètres) que celles du greffon et on indique au compilateur de les appeler dans le greffon avec l'attribut `DllImport`. Ensuite, on construit une interface orientée objet

reproduisant celle d'OpenCV : on utilise le patron de conception façade pour masquer la liaison qui est complexe à utiliser et source d'erreurs de segmentations ou de fuites de mémoire si mal utilisée. La classe `Mat` reconstruite en C# appelle donc automatiquement un des constructeurs C++ quand l'utilisateur désire en créer une instance, par exemple `var mat = new Mat()`. Le pointeur vers la classe C++ sous-jacente est stocké dans l'instance C# correspondante. Nous avons placé toutes les classes de liaison dans l'espace de nom `ArucoUnity.Plugin`; pour simplifier les extraits de code, nous ne les affichons pas.

Enfin, nous avons intégré le conteneur `vector` de la bibliothèque standard de C++. Utilisé dans le module aruco, ce modèle (*template* en anglais) impose un peu plus de travail. Nous devons en effet créer une interface et une classe de liaison pour chaque cas d'utilisation : par exemple, `VectorMat` ou `VectorVectorInt` qui permettent de manipuler respectivement les objets `std::vector<cv::Mat>` et `std::vector<vector<int>>` via des fonctions C comme `IntPtr au_std_vectorMat_new()`. Nous avons décidé cette fois de ne pas répliquer l'interface C++ en utilisant des génériques C#, car chaque cas d'utilisation devant être codé, le générique aurait demandé un travail complexe non nécessaire.

Le code C++ travaille par instants avec les images capturées par la caméra, cette fois allouées en mémoire dans le code C#. On passe alors simplement un pointeur de l'image, ainsi que sa taille, aux fonctions d'OpenCV. Concrètement, on crée un objet `Mat` dans OpenCV et un objet `Texture2D` dans Unity pour manipuler l'image ainsi qu'un tableau partagé entre les deux objets pour représenter le contenu de l'image. On utilise le format le plus courant pour encoder l'image dans le tableau : chaque pixel est représenté en couleurs rouge, vert, bleu (RVB) avec un octet (8 bits) par couleur, soit un total de $3 \times 8 = 24$ bits par pixel. On travaille donc avec une paire d'images de $2 \times 3 \times 960 \times 950 = 5{,}472\,\text{MB}$ dans le cas de l'Ovrvision, ce qui est assez conséquent pour les opérations de suivi de marqueurs et de corrections d'images. C'est

Extrait 3.3 Classe C# ArucoCamera simplifée pour passer une image à OpenCV.

```
public class ArucoCamera : MonoBehaviour, IArucoCamera
{
  public Mat Image;
  public Texture2D Texture;
  public byte[] ImageData;
  public Size Resolution;

  public event Action UndistortImages;
  public event Action ImageUpdated;

  // Exécuté au démarrage
  protected virtual void Start()
  {
    Image = new Mat(Resolution, Format.8UC3); // Format RVB : 3 octets
      par pixels (3x8=24 bits)
    Texture = new Texture2D(Resolution.x, Resolution.y,
      TextureFormat.RGB24);
    ImageData = new byte[3 * Images[cameraId].Total()];
    Image.DataByte = ImageData;
  }

  // Exécuté à chaque frame
  protected virtual void Update()
  {
    UpdateImages(); // Copie de l'image de la caméra dans ImageData

    UndistortImages(); // Correction de l'image
    ImageUpdated(); // Suivi des marqueurs

    // Mise à jour de l'image Unity
    Texture.LoadRawTextureData(image.DataIntPtr, 3 *
      Images[cameraId].Total());
    Texture.Apply();
  }

  protected abstract void UpdateImages();
}
```

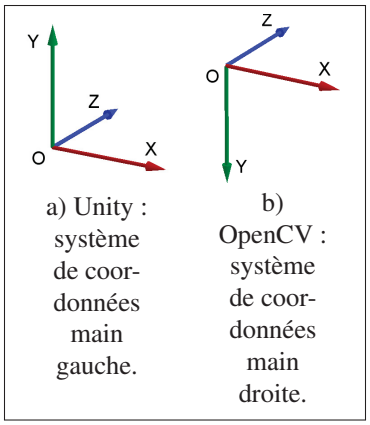
pourquoi on partage la mémoire pour éviter des copies entre Unity et le greffon, coûteuses en temps de processeur.

Nous avons aussi besoin d'accéder à la caméra Ovrvision Pro. La caméra est fonctionnelle, les deux images retournées sont synchronisées mais, comme nous l'avons expliqué dans la sous-section 3.2.2, son étalonnage et le suivis de marqueurs ne sont pas bons avec les scripts C#

Extrait 3.4 Classe C# OvrvisionArucoCamera simplifiée.

```
public class OvrvisionArucoCamera : StereoArucoCamera
{
  [DllImport("ovrvision")]
  static extern int ovOpen(int cameraId, float markerLength, int mode);
  ... // Autres liaisons

  protected override void Start()
  {
    Resolution = new Size(960, 950);
    ovOpen(cameraId: 0, markerLength: 1, mode:
      CameraMode.VR_960x950_60FPS);
    ovSetCamSyncMode(false);
    base.Start();
  }

  protected override void UpdateImages()
  {
    ovPreStoreCamData(ProcessingMode.Demosaic); // Retourner les images
      couleurs non corrigées
    ovGetCamImageRGB(ImageData[0], 0); // Copie image oeil gauche
    ovGetCamImageRGB(ImageData[1], 1); // Copie image oeil droit
  }
}
```

fournis par le constructeur. Un greffon similaire au nôtre étant disponible, nous avons développé la classe `OvrvisionArucoCamera` de liaison (*Voir* Extrait 3.4) pour pouvoir étalonner cette caméra avec ArucoUnity et y intégrer de la RA.



a) Unity : système de coordonnées main gauche.

b) OpenCV : système de coordonnées main droite.

Figure 3.11 Illustrations des systèmes de coordonnées utilisés par Unity et OpenCV ; une inversion de l'axe des Y permet de passer de l'un à l'autre.

Nous avons également ajouté quelques fonctions de conversion, Unity et OpenCV utilisant des conventions et des classes différentes pour encoder les transformations d'objets dans la scène. Pour encoder la position, Unity utilise un système de coordonnées main gauche (*Voir* Figure 3.11a) tandis qu'OpenCV utilise un système main droite (*Voir* Figure 3.11b). On inverse alors l'axe des Y pour passer de l'un à l'autre avec la fonction `Vector3 Vec3d.ToPosition()`. En outre, OpenCV utilise des matrices de rotation ou des angles de rotations, selon les fonctions. OpenCV propose des fonctions de conversions, que nous avons retranscrites dans la liaison. On convertit les angles de rotations en quaternions d'Unity suivant l'algorithme de Baker (1998) avec la fonction `Quaternion Vec3.ToRotation()`.

3.3.4 Réalisation des scripts pour Unity

Ce travail avec les images des caméras et les conversions montre pourquoi nous avons besoin d'une couche supplémentaire de scripts C# au-dessus de celle de la liaison pour faciliter l'usage d'OpenCV dans Unity. Les deux systèmes ne sont en effet pas conçus pour travailler ensemble : on masque ainsi cette complexité avec les scripts, formant une façade supplémentaire.

Avec ces scripts nous mettons aussi en place le fonctionnement de notre visiocasque décrit à la sous-section 3.2.1 en suivant la procédure que nous décrivons à la section 3.4. Nous avons donc d'abord écrit premier script d'étalonnage et un second qui configurait au démarrage puis qui corrigeait à chaque *frame* les images et plaçait les éléments virtuels. Nous y avons rapidement appliqué quelques principes de conception pour l'améliorer. Nous utilisons donc une seule responsabilité par classe (*Voir* Figure 3.12) :

- `IArucoCamera` récupère les images de la caméra physique.
- `IArucoCameraUndistortion` corrige les images capturées.
- `IArucoObjectsTracker` et `ArucoObject` détectent les marqueurs sur les images.
- `IArucoCameraDisplay` configure la caméra virtuelle et place les éléments virtuels.

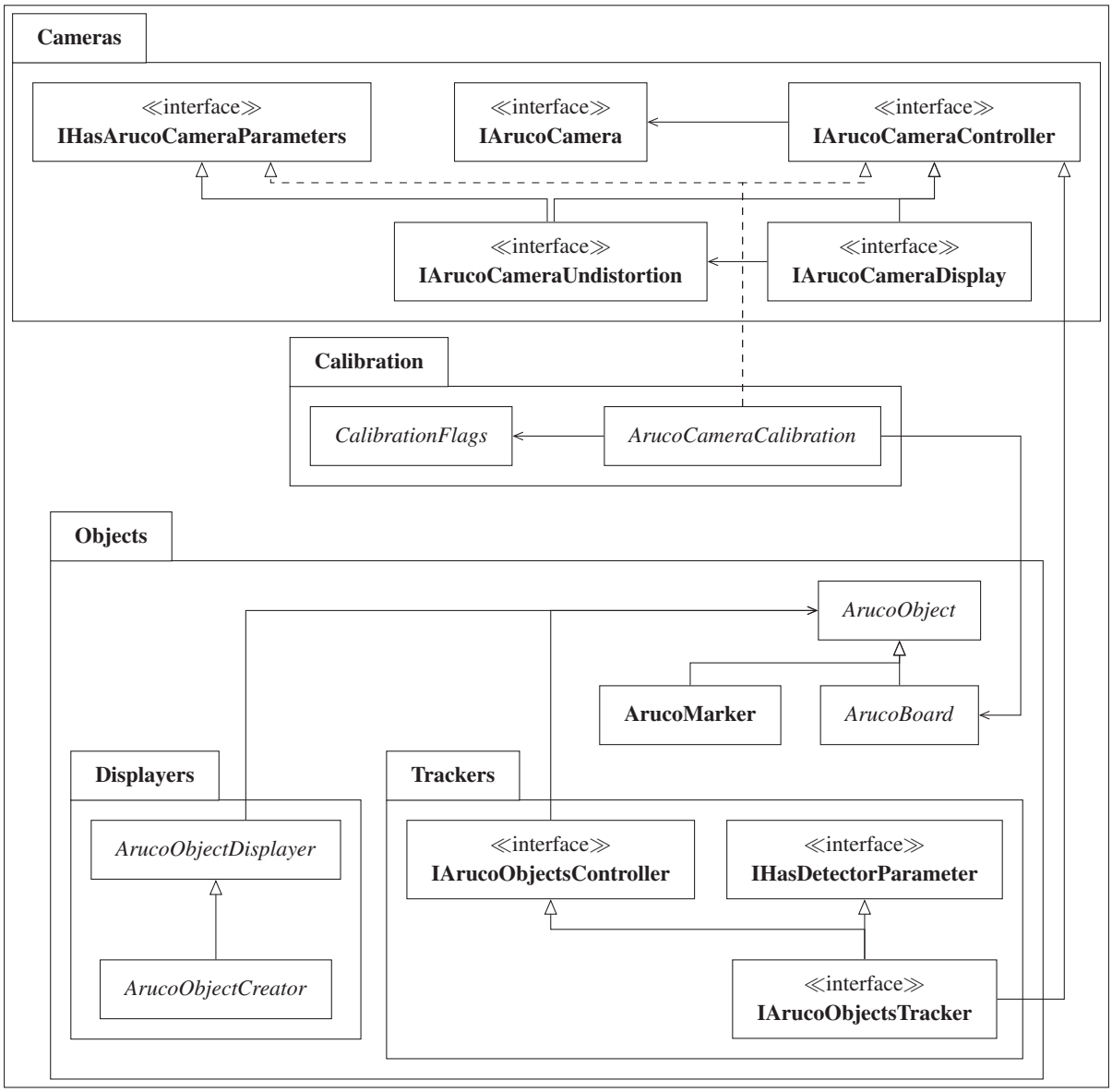


Figure 3.12 Diagramme de classes simplifié de la couche de scripts C# d'ArucoUnity. Seules les interfaces ou les classes abstraites de haut niveaux sont affichées : dans chaque cas, le patron de conception stratégie est appliqué pour supporter différents types de caméras. Elles font toutes parties de l'espace de nom `ArucoUnity`, non affiché ici.

- `ArucoCameraCalibration` étalonne la caméra physique.
- `CameraParametersController` gère les paramètres de l'étalonnage.

Pour chacune de ces interfaces, nous utilisons le patron de conception stratégie : nous créons une classe abstraite implémentant le procédé (nous les décrivons à la section 3.4) mais laissant les détails spécifiques à un type de caméra à des enfants de cette classe. Cela permet de multiples combinaisons entre les implémentations d'`IArucoCamera`, de `IArucoCameraDisplay` et de `IArucoCameraUndistortion`. Un développeur peut donc ajouter facilement le support dans ArucoUnity d'une nouvelle caméra, d'un nouveau type d'affichage ou la correction d'un nouveau type d'objectif photographique.

Enfin, nous améliorons les performances en externalisant certains calculs sur plusieurs fils d'exécution (*thread* en anglais). Par défaut, les scripts tournent en effet sur le fil d'exécution principal d'Unity. L'ensemble des scripts doit donc s'exécuter rapidement pour avoir une application fluide, en moins de 16 ms pour atteindre 60 FPS par exemple. La capture des images, leur correction et le suivi des marqueurs sont les opérations les plus lentes. Nous créons donc un fil d'exécution pour chacune de ces trois calculs ci-dessous : chacun tourne alors de manière indépendante et ne bloque plus celui d'Unity (*Voir* Figure 3.13). Si le fil d'exécution de la caméra a récupéré de nouvelles images, celui d'Unity va les copier dans les autres fils et recopie les images corrigées pour les afficher dans le visiocasque. Cette solution nous permet d'avoir d'excellentes performances avec une webcam où nous avions à l'origine peu de fluidité, mais n'est pas encore totalement satisfaisante avec l'Ovrvision (*Voir* sous-section 3.2.3). Dans cette configuration, ce sont les multiples copies des images qui prennent trop de temps au processeur, mais nous en sommes resté là par manque de temps et la vitesse de 30 FPS avec notre visiocasque étant tout de même satisfaisante.

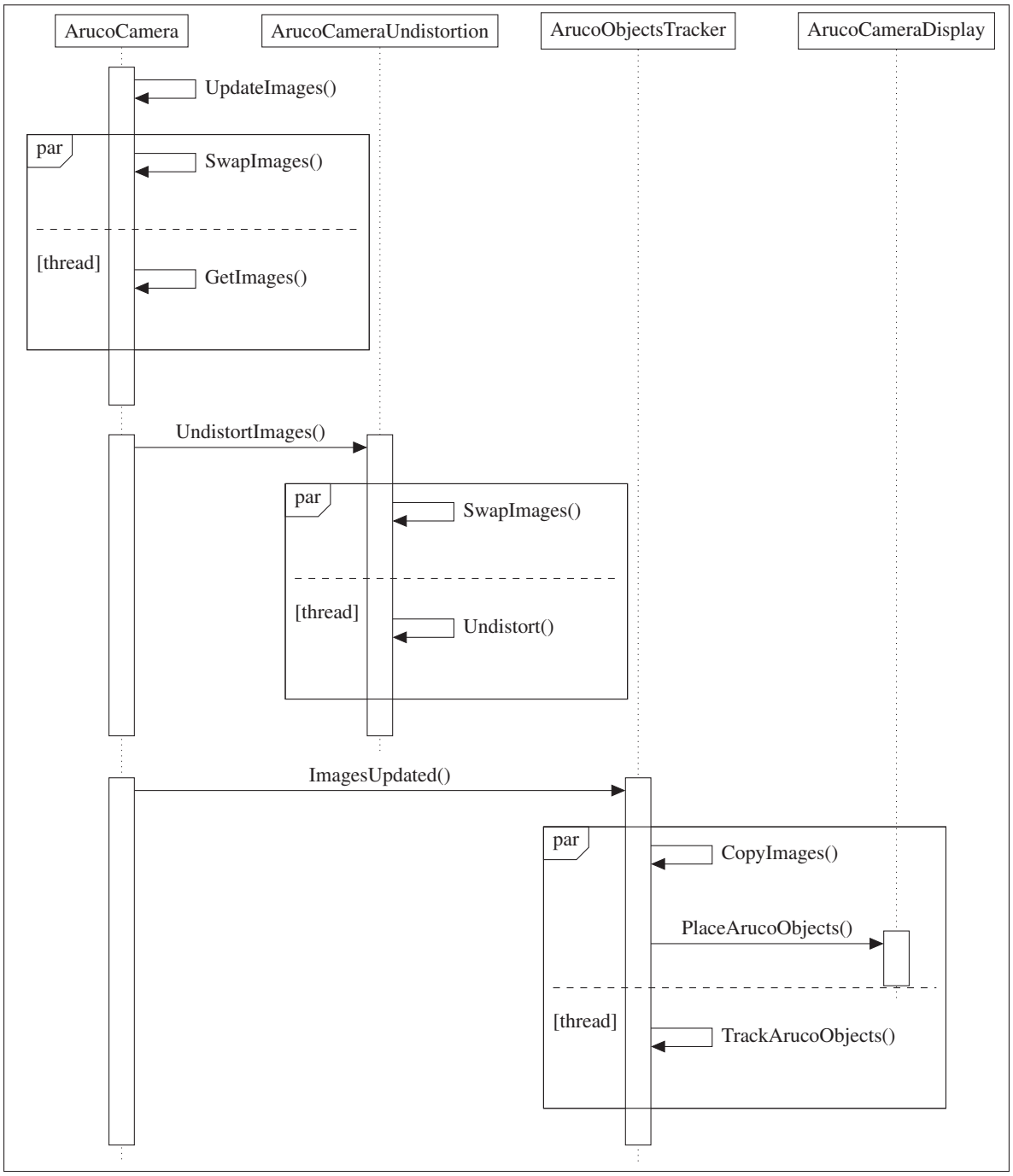


Figure 3.13 Diagramme de séquence simplifié de chaque *frame* de la couche de scripts C# d'ArucoUnity.

3.4 Intégration de la réalité augmentée dans le visiocasque

3.4.1 Procédure

Les fonctionnalités d'OpenCV étant maintenant disponibles dans Unity (*Voir* section 3.3), nous pouvons maintenant enfin mettre en place la RA dans notre visiocasque. Pour cela, nous devons tout d'abord comprendre le principe d'une caméra (*Voir* sous-section 3.4.2). Puis nous étudions le cas le plus simple de l'alignement d'une caméra physique monoscopique avec un objectif rectilinéaire (*Voir* sous-section 3.4.3). Nous itérons ensuite avec notre caméra stéréoscopique à objectifs *fisheye* (*Voir* sous-section 3.4.7). Les caméras physiques et virtuelles alignées nous pourrons alors utiliser les fonctions de suivi de marqueurs pour placer les éléments virtuels.

ArucoUnity permet d'étalonner une caméra facilement et directement dans Unity, sans avoir à programmer. Après l'étalonnage, notre bibliothèque de RA corrige la caméra physique et aligne la caméra virtuelle automatiquement sans que l'utilisateur n'ait à s'en soucier lors du suivi de marqueurs. Nous décrivons ainsi dans cette section l'utilisation des scripts C# d'ArucoUnity, le code OpenCV qu'ils exécutent en arrière-plan ainsi que la théorie nécessaire ; le lecteur peut se référer à (Kaehler & Bradski, 2016, chap. 18-19) pour une théorie et des procédures plus complètes.

Cette section a son équivalent en anglais sur plusieurs articles dans la documentation en ligne d'ArucoUnity (https://normanderwan.github.io/ArucoUnity/). Ils sont plus pratiques sans détailler la théorie pour permettre à nouvel utilisateur de rapidement construire son application de RA.

3.4.2 Fonctionnement d'une caméra

3.4.2.1 Caméra virtuelle

Une caméra est un dispositif qui capture une vue en 2D d'une scène en 3D. Cette vue est une projection en *perspective* rectilinéaire. Le plus simple des dispositifs de projection en perspective est le *sténopé* (*pinhole* en anglais) : c'est une simple boite percée d'une petite ouverture, le *centre de projection*, noté O, sur une de ses face, et d'un écran, *le plan image*, sensible à la lumière sur la face opposée. La scène n'est donc visible que par un unique point de vue : ainsi, chaque point sur le plan image ne provenant que d'un seul rayon de lumière à travers le centre de projection, une image nette et inversée de la scène va se former (*Voir* Figure 3.14).

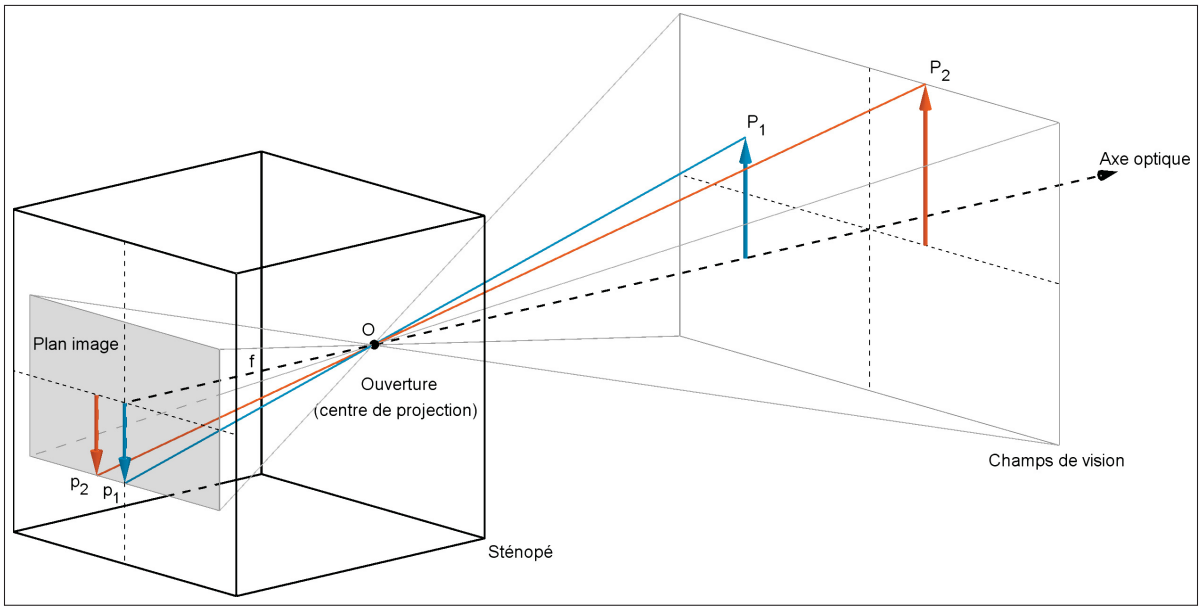


Figure 3.14 Un sténopé : une petite ouverture sur une face de la boite laisse entrer la lumière, projettant une scène 3D (par exemple les points P_1 et P_2) en une image inversée sur la face opposée à l'ouverture (les projections respectives sont les points p_1 et p_2).

L'image formée sur le plan image étant toujours nette, on peut modifier le champ de vision du sténopé en déplaçant le plan image le long de l'*axe optique*, la droite perpendiculaire au

plan image et passant par le centre de projection, la distance entre le plan image et le centre de projection étant la *longueur focale* f. On calcule alors le champ de vision du sténopé avec l'Équation 3.1 avec cette distance f, habituellement donnée en millimètres par le constructeur de la caméra. Ainsi, à taille équivalente du plan image, une longueur focale plus courte donne lieu à un plus grand champ de vision.

Pour simplifier nos équations, on peut modifier le modèle du sténopé tout en le gardant valide mathématiquement. On crée un nouveau plan image symétrique au véritable plan image par rapport au centre de projection, placé donc à la même distance f du centre optique. La vue générée sur ce plan image virtuel reste la même mais est non inversée : les rayons de la scène sont toujours projeté vers le centre de projection mais interceptés par ce nouveau plan image (*Voir* Figure 3.15). Le repère est placé sur le centre de projection.

De manière simplifiée, c'est également le fonctionnement d'une caméra virtuelle monoscopique. En suivant l'Équation 3.2, on projette chaque point $P = (X, Y, Z)$ de la scène en un point $p = (x, y, z)$ sur le plan image. Les unités sont arbitraires ; Unity utilise par exemple des mètres. En capturant un nombre suffisamment élevé d'images par secondes (au moins 15 FPS) avec cette caméra, on a l'illusion de voir une vidéo.

$$x = f\frac{X}{Z}, \qquad y = f\frac{Y}{Z}, \qquad z = f \tag{3.2}$$

3.4.2.2 Caméra physique

Le sténopé est un modèle de caméra également suffisant pour décrire une caméra physique monoscopique. On y ajoute alors un capteur numérique placé sur le plan image pour enregistrer les images qui se forment sur le plan image. Cet ajout introduit de nouveaux paramètres que l'on va mesurer lors de l'étalonnage de la caméra (*Voir* sous-section 3.4.3). Comme nous

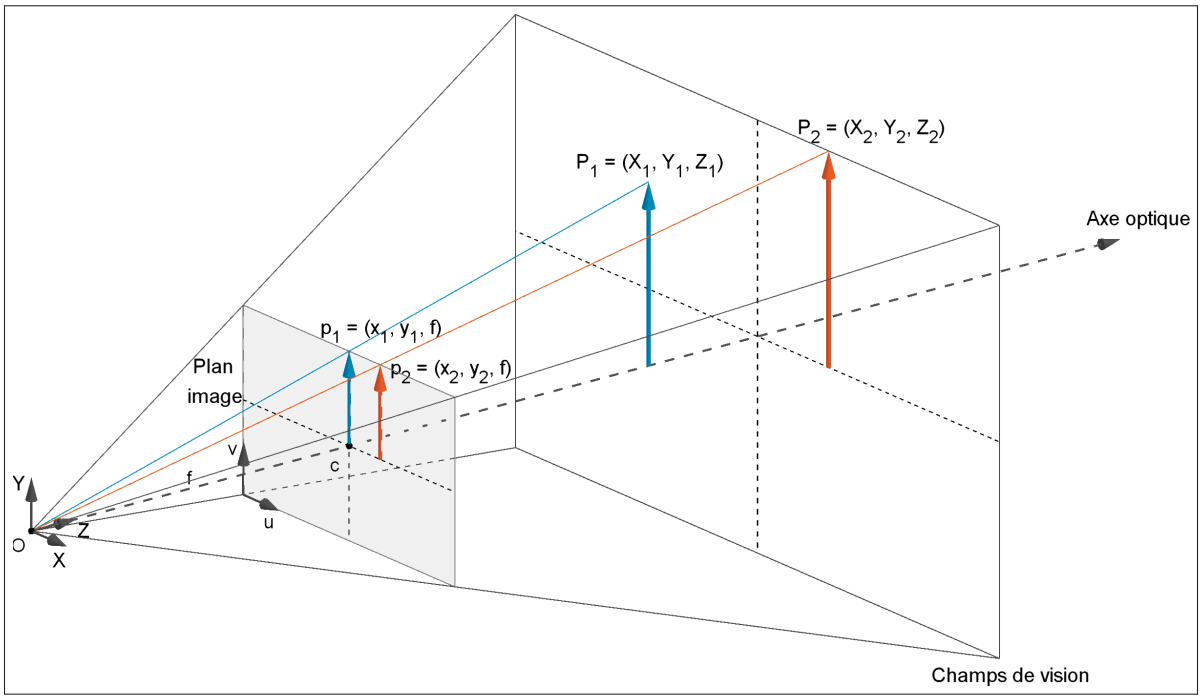


Figure 3.15 Projection en perspective rectilinéaire : un point $P = (X, Y, Z)$ va être projeté sur le plan image situé à une distance $z = f$ du centre de projection O en un point $p = (x, y, f)$. Une caméra virtuelle est basée sur ce principe et on peut également simplifier le sténopé à ce modèle.

travaillons seulement avec les images capturées dans ce procédé, les mesures sont donc faites en pixels et non plus en millimètres. De plus, nous déplaçons le repère dans le coin en bas à gauche du plan image.

Premièrement, le centre du capteur utilisé ne coïncide pas totalement avec l'axe optique : cela requerrait une précision trop importante et non nécessaire lors de la fabrication de la caméra. L'axe optique va donc intersecter le plan image non pas en son centre, mais sur un point décalé appelé *centre optique* et noté c. On mesure en pixels le décalage (c_x, c_y) sur l'image capturée.

En outre, la longueur focale mesurée de la caméra peut être différente sur les deux axes X et Y du plan image. On note f_x et f_y les deux longueurs focales mesurées. Cela est dû a certains capteurs, de mauvaise qualité, produisant des pixels rectangulaires et non carrés. On pourrait

également les calculer en connaissant la taille physique (L,H) et la définition (d_x,d_y) du capteur et la longueur focale physique de la caméra avec l'Équation 3.3, mais on n'a pas toujours accès avec fiabilité à toutes ces informations [1].

$$f_x = f\frac{L}{d_x}, \qquad f_y = f\frac{H}{d_y} \tag{3.3}$$

Ces nouveaux paramètres font partie de la *matrice intrinsèque* de la caméra, notée K. Elle permet de transformer un point P de la scène est transformé en un point p' (*Voir* Équation 3.4), qu'on projette ensuite sur le plan image en divisant ses coordonnées par Z comme dans l'Équation 3.2.

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \underbrace{\begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}}_{\mathrm{K}} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{3.4}$$

On résume donc la projection d'un point $P = (X,Y,Z)$ de la scène par une caméra physique monoscopique en un point $p = (x,y)$ sur l'image en suivant l'Équation 3.5.

$$x = f_x\frac{X}{Z} + c_x, \qquad y = f_y\frac{Y}{Z} + c_x \tag{3.5}$$

Enfin, une caméra physique monoscopique requiert également un objectif photographique. En effet, l'ouverture infiniment petite du sténopé ne laissant passer que trop peu de lumière demande infiniment de temps pour qu'une image visible se forme. C'est un problème pour une

1. Les propriétés d'un appareil photo sont bien décrites sur internet mais c'est souvent moins le cas de la caméra d'un téléphone par exemple.

caméra si l'on souhaite capturer plusieurs images par secondes. La solution est de faire une ouverture plus grande pour laisser passer plus de lumière et de placer donc un objectif photographique pour continuer à concentrer la lumière sur le plan image en une seule image nette. L'inconvénient est que les lentilles utilisées introduisent nécessairement des distorsions en barillet dans l'image (*Voir* Figure 3.16). De plus, le capteur et l'objectif ne sont pas forcément bien montés parallèles, entraînant une distorsion tangente supplémentaire, le plan image ne coïncidant pas totalement avec le capteur. D'autres types de distorsions existent mais peuvent être négligés (Bradski & Kaehler, 2008, p. 377). L'ensemble des *paramètres de distorsion* qui sont mesurés est noté *D*. Pour plus de détails sur ces paramètres, nous redirigeons le lecteur vers (Bradski & Kaehler, 2008, p. 375).

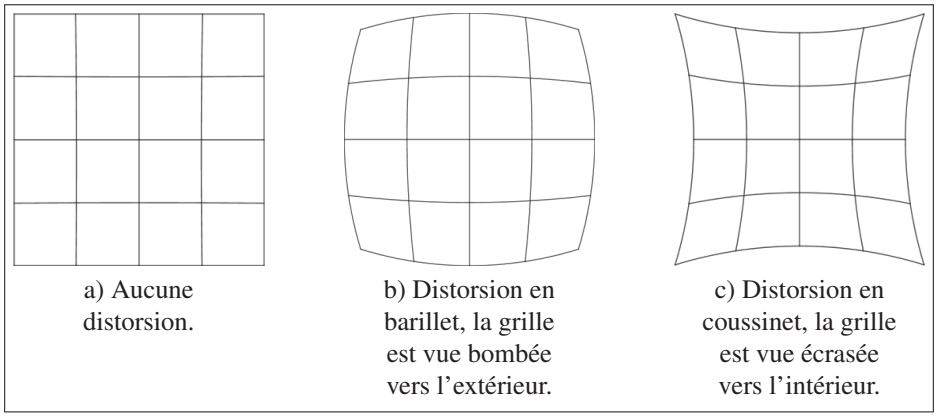


Figure 3.16 Illustrations d'une grille vue à travers un système optique avec différents types de distorsion.

Ces distorsions éloignent le fonctionnement de la caméra du modèle idéal du sténopé. C'est pourquoi il est important de corriger les déformations que cela produit sur les images, comme si elles avaient été capturées par un sténopé avec les mêmes paramètres intrinsèques et donc aucune distorsions, pour conserver un bon alignement avec la caméra virtuelle.

3.4.3 Étalonnage d'une caméra

Étalonner une caméra monoscopique rectilinéaire consiste donc à déterminer sa matrice intrinsèque K et ses paramètres de distorsion D. Nous avons décrit les équations qui permettent de projeter un point P de la scène en un point p sur le plan image. Connaissant les coordonnées de ces deux points, nous pouvons donc écrire les vecteurs de translation $t = (t_x, t_y, t_z)$ et rotation $r = (r_x, r_y, r_z)$ permettant de transformer un point P de la scène en un point p sur le plan image. En connaissant assez de couple de points (P_i, p_i), on peut alors poser un ensemble d'équations pour résoudre K. Ainsi, les déviations des points p_i du modèle de sténopé ainsi calculé permettent de déterminer D.

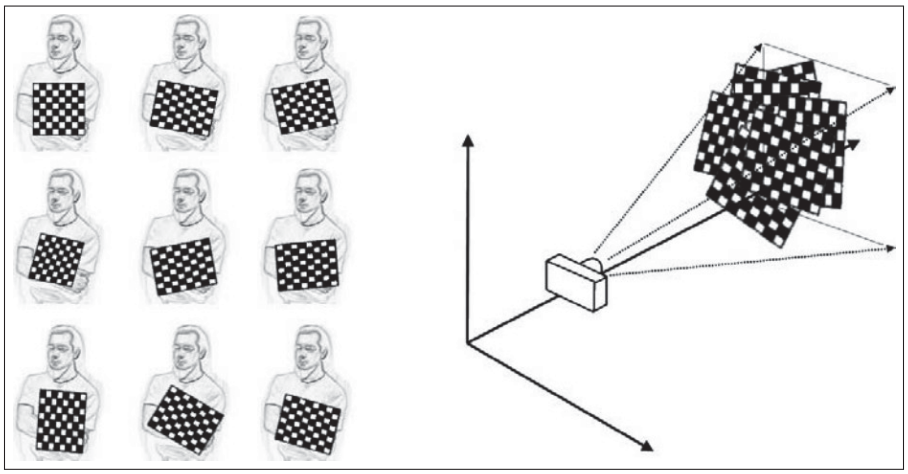


Figure 3.17 Une caméra physique monoscopique est étalonné avec OpenCV en capturant plusieurs images d'un échiquier imprimé sur une planche rigide capturé dans différentes positions et angles de vues.
Tiré de Bradski & Kaehler (2008).

Concrètement, on utilise une image d'un échiquier qu'on capture avec la caméra dans plusieurs positions et angles de vues (*Voir* Figure 3.17), les points utilisés étant les intersections entre les carrés noirs et blancs. On connaît par avance les coordonnées relatives des points P_i entre eux (il suffit de le mesurer sur l'échiquier) et ces intersections sont détectées facilement sur

l'image capturée par la caméra. La première étape de l'étalonnage consiste donc à capturer avec la caméra quelques images d'un échiquier dont on connaît la configuration.

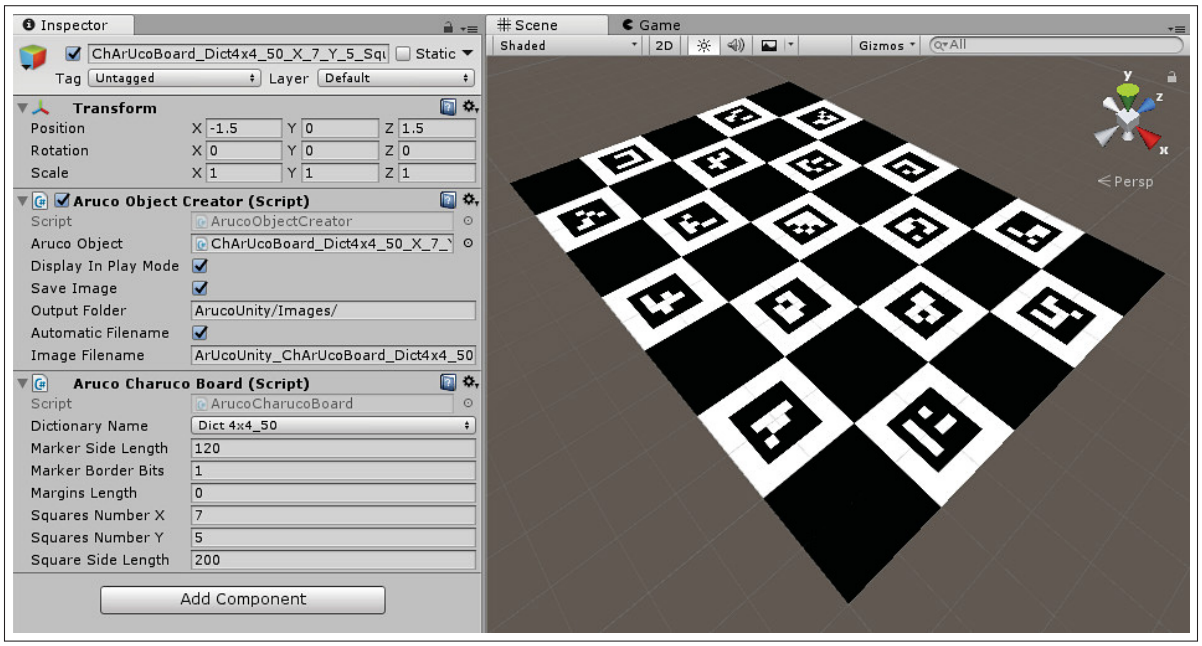


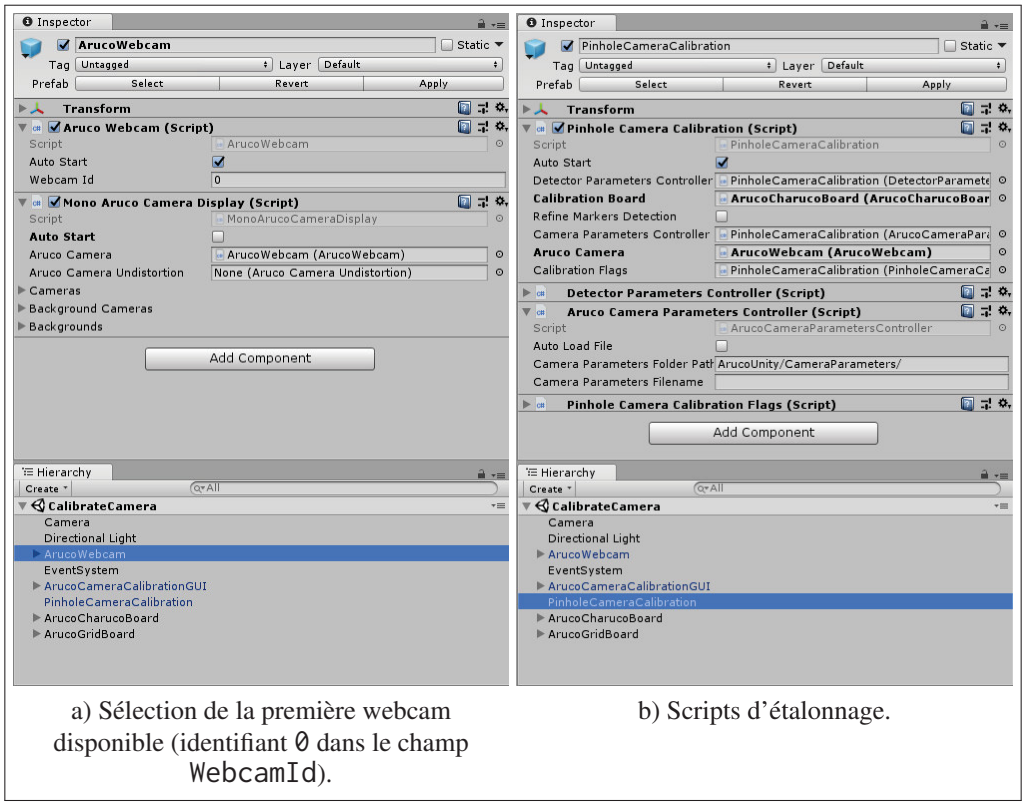
Figure 3.18 Création d'un échiquier d'étalonnage avec ArucoUnity : à gauche les scripts de configuration, à droite l'échiquier vu dans la scène Unity. Les marqueurs, décrits dans la section 3.3, dans les carrés blancs permettent l'amélioration de la détection de l'échiquier.

On crée alors notre échiquier d'étalonnage (*Voir* Figure 3.18) :

- On crée un projet Unity configuré avec ArucoUnity (*Voir* section 3.3).
- Dans ce projet, on ajoute un nouvel objet vide.
- On ajoute à l'objet le script `ArucoCharucoBoard`, qui représente l'échiquier, et on le configure :
  - la taille des carrés en pixels avec le champ `SquareSideLength` ;
  - le nombre de carrés avec `SquaresNumberX` et `SquaresNumberY` ;
  - la taille des marqueurs en pixels avec `MarkerSideLength` ;

- la marge en pixels entre les marqueurs et les carrés blancs qui les contiennent avec `MarginLength`.

- On ajoute à l'objet le script `ArucoObjectCreator`, qui permet de créer l'image de l'échiquier, et on fait pointer `ArucoObject` vers `ArucoCharucoBoard`.
- On démarre la scène Unity : l'image de l'échiquier s'affiche et est enregistrée sur le disque dur dans le dossier du projet (le chemin est indiqué par `OutputFolder`).
- On imprime l'image que l'on colle fermement sur une surface rigide pour qu'elle reste la plus plate possible.



a) Sélection de la première webcam disponible (identifiant `0` dans le champ `WebcamId`).

b) Scripts d'étalonnage.

Figure 3.19 Configuration de la scène Unity `CalibrateCamera.unity` pour étalonner une caméra monoscopique rectilinéaire.

On prépare ensuite la scène de calibration (*Voir* Figure 3.19) :

1. On ouvre la scène Unity `Assets/ArucoUnity/Scenes/CalibrateCamera.unity` déjà préparée pour l'étalonnage d'une caméra monoscopique rectilinéaire.
2. Sur le script `ArucoWebcam`, dans l'objet du même nom, on choisit la caméra à étalonner en renseignant son identifiant numérique dans le champ `WebcamId`.
3. On crée à nouveau un objet représentant notre échiquier imprimé, en y configurant cette fois la taille des carrés en mètres, permettant à ArucoUnity de construire automatiquement les positions relatives à échiquier des points P_i.
4. Sur le script `PinholeCameraCalibration`, dans l'objet du même nom, on fait pointer le champ `CalibrationBoard` vers notre objet d'échiquier.
5. On peut ajuster les paramètres de calibration (*calibration flags* en anglais) du module calib3d avec le script `PinholeCameraCalibrationFlags` et les paramètres de détection du module aruco avec le script `DectectorParametersController`, ces deux scripts ayant par défaut les valeurs recommandées par les documentations de leurs modules respectifs.

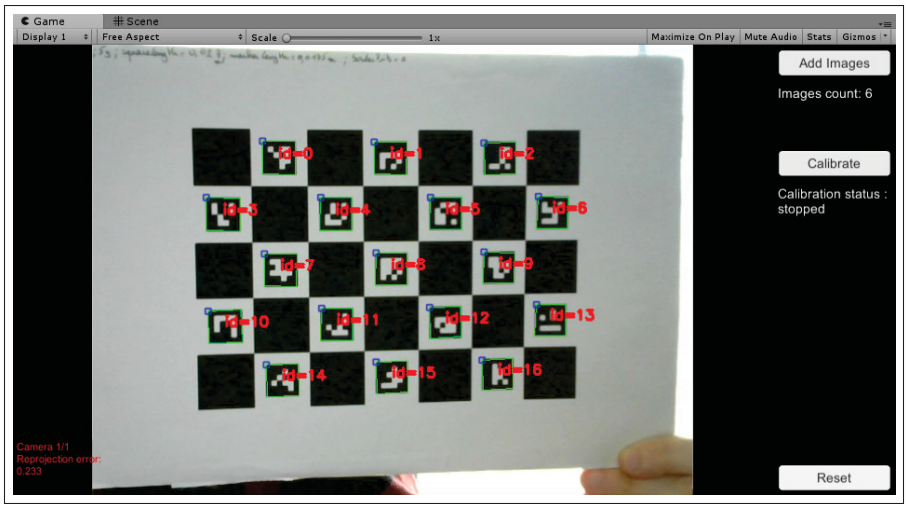


Figure 3.20 Scène `CalibrateCamera.unity` jouée pour étalonner une webcam. Les marqueurs de l'échiquier sont surlignés pour indiquer s'il est correctement détecté.

On peut maintenant débuter l'étalonnage en jouant la scène (*Voir* Figure 3.20). Le flux vidéo de la caméra est affiché alors dans Unity ainsi qu'une IHM : le bouton `Add Image` permet de cap-

turer l'image courante dans une liste, le bouton `Reset` permet de vider cette liste et le bouton `Calibrate` permet d'étalonner la caméra en utilisant cette liste d'images. Il est important de garder la caméra dans une position fixe durant l'étalonnage et de varier seulement la position et l'orientation de l'échiquier entre les images. On désactive aussi la mise au point automatique de la caméra, ce paramètre impactant en effet la matrice intrinsèque K de la caméra et les distorsions de l'objectif. Enfin, l'échiquier est surligné dans le flux vidéo pour aider à le positionner.

Extrait 3.5 Classe C# ArucoCameraCalibration simplifiée.

```
public abstract class ArucoCameraCalibration : ArucoCameraController,
  IArucoCameraCalibration, IHasCameraParameters
{
  public Board Board; // L'échiquier
  public CameraParameters CameraParameters;
  public double Error;
  public var ObjectPoints = new VectorVectorPoint3f(); // Points P
  public var ImagePoints = new VectorVectorPoint2f(); // Points p

  // (1) Détection des points P et p
  protected void AddImages()
  {
    VectorInt ids; // Identifiants des marqueurs
    VectorVectorPoint2f corners; // Coins des marqueurs
    DetectMarkers(ArucoCamera.Image, Board.Dictionary, out corners, out
      ids, DetectorParameters, out rejectedCandidateCorners);

    VectorPoint3f P;
    VectorPoint2f p;
    GetBoardObjectAndImagePoints(Board, corners, ids, out P, out p);
    ObjectPoints.PushBack(P);
    ImagePoints.PushBack(p);
  }

  public abstract void Calibrate(); // (2) Étalonnage
}
```

Nos scripts d'étalonnages sont simples : (1) on détecte l'échiquier dans et on extrait les points P_i et p_i pour chaque image de la liste (*Voir* Extrait 3.5), (2) on calcule la matrice intrinsèque K et les paramètres de distorsion D. Un script spécifique par type de caméra implémente cette seconde étape, par exemple l'Extrait 3.6 pour une caméra monoscopique rectilinéaire.

Extrait 3.6 Classe C# PinholeCameraCalibration simplifiée.

```
public class PinholeCameraCalibration : ArucoCameraCalibration
{
  public PinholeCameraCalibrationFlags Flags;

  public virtual void Calibrate()
  {
    VectorMat t, r; // Position et rotation de l'échiquier à chaque image
    Error = CalibrateCamera(ObjectPoints, ImagePoints,
      ArucoCamera.Resolution, CameraParameters.K, CameraParameters.D,
      out r, out t, Flags);
    CameraParameters.Save();
  }
}
```

Une fois l'étalonnage effectué, un score mesurant sa qualité est alors affiché. Il correspond à la somme de l'erreur de reprojection pour chaque image de la liste : c'est la distance en pixels entre les points p_i observés et la projection simulée des points P_i sur le plan image avec le modèle de sténopé mesuré par l'étalonnage. Ce score doit être le plus proche possible de zéro si l'on souhaite un bon alignement des deux caméras : un décalage trop important dans l'alignement des éléments virtuels sur l'image de la caméra physique briserait l'illusion de la RA. Enfin, le résultat de l'étalonnage est enregistré dans un fichier XML dans le dossier `Assets/ArucoUnity/CameraParameters/` que l'on pourra utiliser à volonté dans n'importe quelle application de RA utilisant cette caméra.

3.4.4 Correction des image de la caméra

Une fois la caméra étalonnée, nous pouvons y intégrer la RA. Nous devons donc tout d'abord corriger les images de la caméra physique. Cette opération se fait en deux temps : on calcule d'abord une fonction de mappage m entre les pixels de l'image corrigée I_c et ceux l'image originale I, selon l'Équation 3.6, que l'on pourra ensuite appliquer sur chacune des images.

$$I_c(x,y) = I(m_x(x,y), m_y(x,y)) \tag{3.6}$$

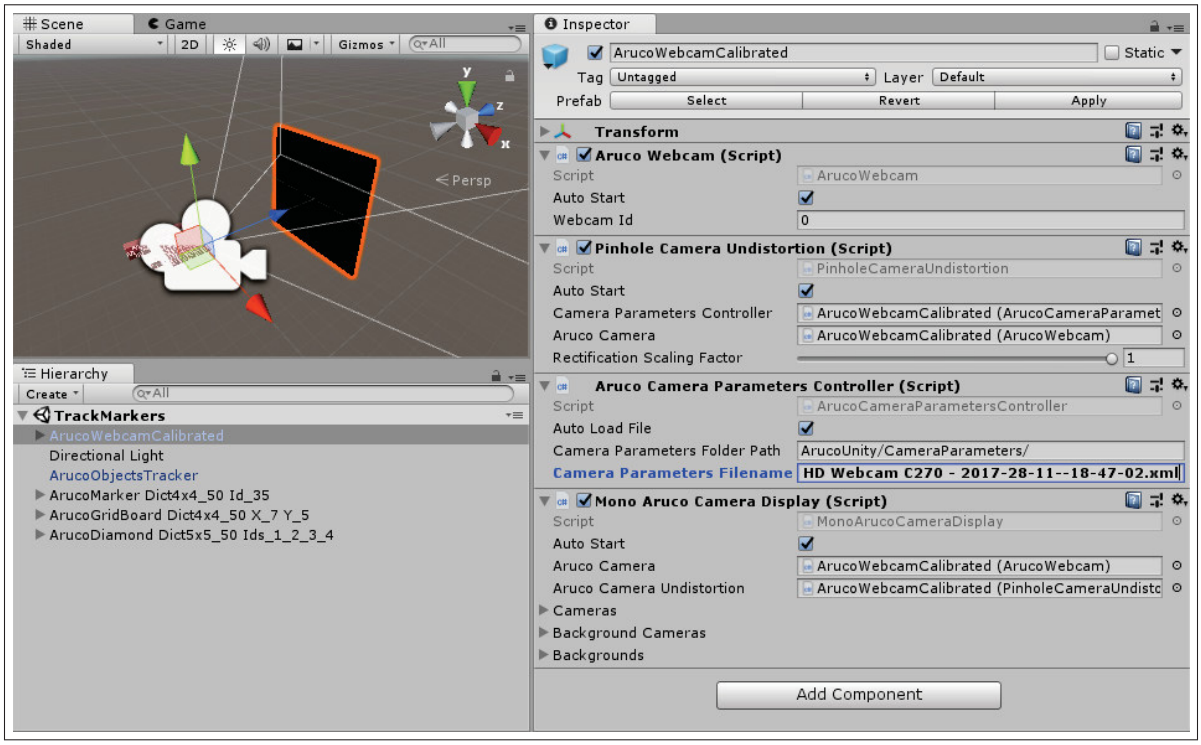


Figure 3.21 Configuration de la correction d'une caméra monoscopique rectilinéaire (`PinholeCameraUndistortion`) et de l'alignement d'une caméra virtuelle (`MonoArucoCameraDisplay`) pour permettre de la RA.

Cette opération est intéressante, car elle nous permet donc de choisir une autre matrice intrinsèque, comme si les images corrigées avaient été capturées par une autre caméra avec un autre plan image. Cela nous permet de replacer le centre optique au centre de l'image soit $c_x = {}^{d_x}/_2$ et $c_y = {}^{d_y}/_2$. Nous pourrions également changer la position et l'orientation de cette caméra corrigée de cette manière.

ArucoUnity effectue automatiquement la correction des images, lors du suivi de marqueurs, avec le script `ArucoCameraUndistortion`. On le met en place facilement (*Voir* Figure 3.21) :

1. On ouvre la scène `Assets/ArucoUnity/Scenes/TrackMarkers.unity` préparée pour le suivi de marqueurs avec une caméra monoscopique rectilinéaire.

Extrait 3.7 Classe C# ArucoCameraUndistortion simplifiée.

```
public class ArucoCameraUndistortion : ArucoCameraController,
  IArucoCameraUndistortion, IHasCameraParameters
{
  public CameraParameters CameraParameters;
  public Mat Knew;
  public float Fx, Fy;
  public Mat noD = new Mat(); // Aucune distortions
  protected Mat rectMapX, rectMapY;

  protected void Start()
  {
    InitializeUndistortion();
    Fx = (float)Knew.AtDouble(0,0), Fy = (float)Knew.AtDouble(1,1);
    ArucoCamera.UndistortImages += Undistort;
  }

  protected abstract void InitializeUndistortion();

  protected void Undistort() // (3) Correction des images
  {
    Remap(ArucoCamera.Image, ArucoCamera.Image, rectMapX, rectMapY,
      InterpolationFlags.Linear);
  }
}
```

Extrait 3.8 Classe C# PinholeCameraUndistortion simplifiée.

```
public class PinholeCameraUndistortion : ArucoCameraUndistortion
{
  public float Alpha;

  protected override void InitializeUndistortion()
  {
    // (1) Matrice intrinsèque de la caméra corrigée
    Knew = GetOptimalCameraMatrix(CameraParameters.K,
      CameraParameters.D, ArucoCamera.Resolution, Alpha);

    // (2) Fonction de mappage de correction
    var noR = new Mat(); // Aucune rotation de la caméra corrigée
    InitUndistortRectifyMap(CameraParameters.K, CameraParameters.D, noR,
      newK, ArucoCamera.Resolution, Type.CV_16SC2, out rectMapX, out
      rectMapY);
  }
}
```

2. Dans l'objet `ArucoWebcam`, sur le script `ArucoCameraParametersController`, on renseigne le nom du fichier d'étalonnage dans le champ `CameraParametersFilename`.
3. Le script `ArucoCameraUndistortion` est déjà configuré pour utiliser la caméra physique et le fichier d'étalonnage.

Dans les scripts de correction, on calcule tout d'abord (1) la nouvelle matrice intrinsèque et (2) la fonction de mappage (*Voir* Extrait 3.8) puis (3) on l'applique sur chaque image capturée par la caméra (*Voir* Extrait 3.7).

Cependant, lors de l'application de la fonction de mappage, il y a une ambiguïté à résoudre. La fonction *m* retourne des nombres décimaux alors que les coordonnées des pixels d'une image sont des entiers : il n'existe donc pas toujours une correspondance exacte entre les pixels des deux images. On choisit alors une méthode d'interpolation pour résoudre ce problème : l'interpolation bilinéaire donne de bons résultats sans trop augmenter le temps de calcul. En outre, certains pixels de l'image corrigée peuvent n'avoir aucune correspondance : on les colore en noir dans ce cas. Avec le paramètre α (*Voir* Figure 3.22), l'utilisateur peut décider de n'afficher que les pixels de l'image corrigée ayant une correspondance ($\alpha = 0$) ou de conserver tous les pixels de l'image originale, introduisant de nombreux pixels noirs ($\alpha = 1$). Nous conseillons toutefois ce dernier cas pour conserver les valeurs de longueur focale non-corrigées et éviter un effet de « zoom » non voulu.

3.4.5 Alignement de la caméra virtuelle

Nous pouvons maintenant configurer la caméra virtuelle en l'alignant avec la caméra physique corrigée. La matrice intrinsèque associée avec les images corrigées pouvant être modifiée par la correction, l'alignement doit donc se faire après.

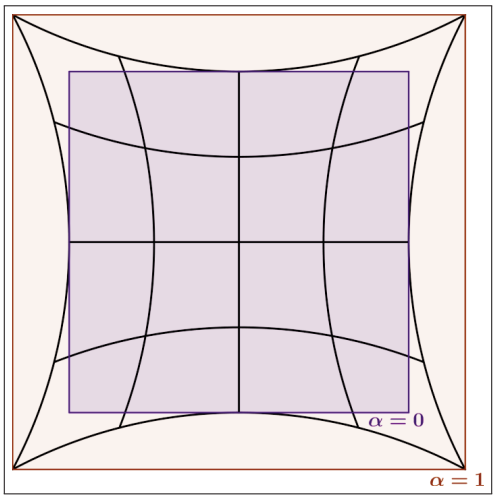


Figure 3.22 Image corrigée avec en surimpression la zone sélectionnée suivant la valeur du paramètre α : seulement les pixels valides dans l'image corrigée avec $\alpha = 0$ ou tous les pixels transformés de l'image originale $\alpha = 0$.

Comme pour la correction, ArucoUnity aligne automatiquement la caméra virtuelle lors du suivi de marqueurs. En reprenant la scène de la sous-section 3.4.4, on voit le script déjà configuré pour utiliser la caméra physique (*Voir* Figure 3.21).

On crée alors dans les scripts : (1) une caméra virtuelle dans Unity, placée à l'origine, avec un champ de vision calculé (*Voir* Équation 3.1) depuis la matrice intrinsèque corrigée, puis (2) l'arrière-plan sous forme d'un rectangle face à la caméra virtuelle, le long de son axe optique (nous avons corrigé le centre optique) à une distance arbitraire lontaine pour éviter une intersection avec les éléments virtuels (*Voir* Extrait 3.9). Enfin, on calcule la taille de ce rectangle en utilisant les triangles semblables : $d'_x = \text{distance} \times {}^{d_x}/_{f_x}$ (*Voir* Figure 3.1).

3.4.6 Réalité augmentée par suivi de marqueur

Nous pouvons utiliser le suivi de marqueur du module aruco pour savoir où placer nos éléments virtuels. ArucoUnity permet un suivi de marqueurs facile, à configurer directement dans l'éditeur d'Unity :

Extrait 3.9 Classe C# ArucoCameraDisplay simplifiée.

```
public abstract class ArucoCameraDisplay : ArucoCameraController,
  IArucoCameraDisplay
{
  public Camera Camera; // Caméra virtuelle d'Unity
  public ArucoCameraUndistortion Undistortion;
  protected float cameraFov;
  protected const int bgDistance = 1000;

  protected override void Start()
  {
    // (1) Configuration de la caméra virtuelle
    Camera = new GameObject().AddComponent<Camera>();
    Camera.transform.SetParent(this.transform.parent);
    Camera.transform.localPosition = Vector3.zero;
    Camera.transform.localRotation = Quaternion.identity;
    Camera.fieldOfView = cameraFov;

    // (2) Configuration de l'arrière-plan
    var bg = GameObject.CreatePrimitive(PrimitiveType.Quad);
    bg.transform.SetParent(this.transform.parent);
    bg.transform.localPosition = new Vector3(0, 0, bgDistance);
    bg.transform.LookAt(camera.transform);
    bg.transform.localScale = new Vector3(
      ArucoCamera.Resolution.Width / Undistortion.Fx * bgDistance,
      ArucoCamera.Resolution.Height / Undistortion.Fy * bgDistance, 1);
    bg.material.mainTexture = ArucoCamera.Texture;
  }

  public void PlaceArucoObject(ArucoObject obj, Vec3d pos, Vec3d rot)
  {
    obj.gameObject.SetActive(true);
    obj.transform.SetParent(Camera.transform);
    obj.transform.localPosition = new Vector3(pos);
    obj.transform.localRotation = new Quaternion(rot);
  }
}
```

1. On crée tout d'abord des marqueurs comme nous l'avons vu dans la sous-section 3.4.3 qu'on imprime et place dans l'environnement physique.
2. On reprends la scène configurée aux sections 3.4.4 et 3.4.5.
3. On y crée à nouveau tous marqueurs suivis en y les configurants avec les valeurs en mètres de ceux imprimés, et on y ajoute les éléments que l'on souhaite afficher en RA comme enfants de ces marqueurs (*Voir* Figure 3.23).

Extrait 3.10 Classe C# MonoArucoCameraDisplay simplifiée.

```
public class MonoArucoCameraDisplay : ArucoCameraDisplay
{
  protected override void Start()
  {
    cameraFov = 2f * Mathf.Atan(0.5f * ArucoCamera.Resolution.Height /
      fy) * Mathf.Rad2Deg;
    base.Start();
  }
}
```

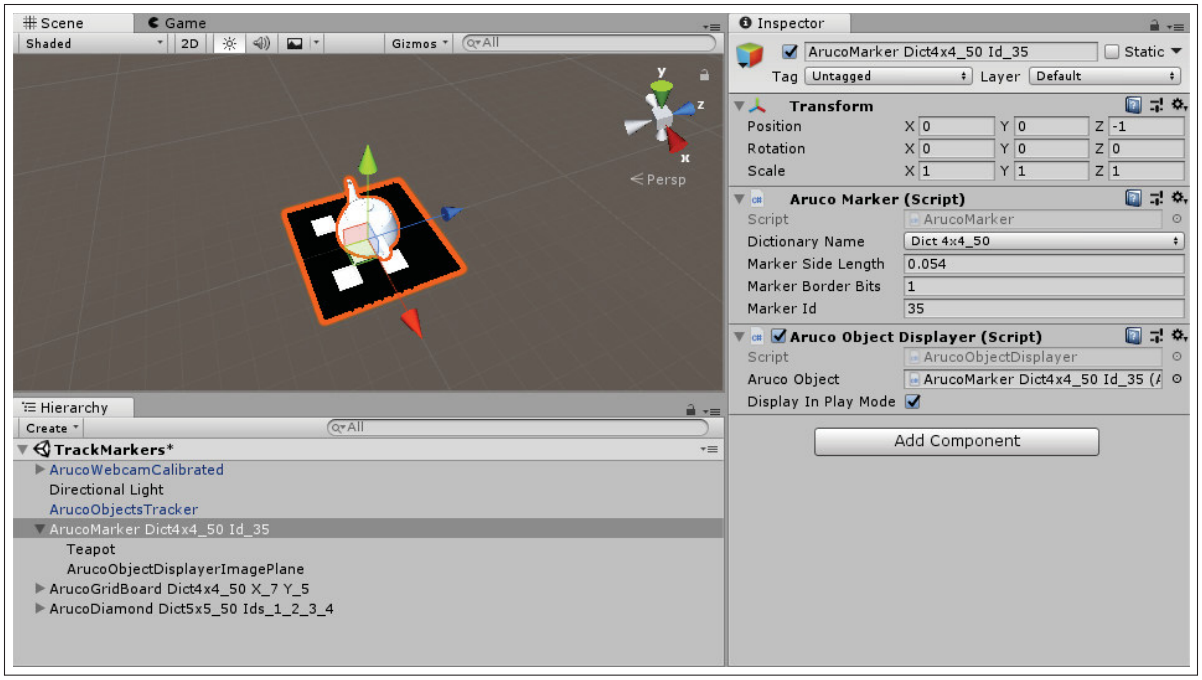


Figure 3.23 Configuration d'un marqueur mesurant 5,4 cm avec un modèle de théière comme élément enfant.

4. Dans le script `ArucoObjectsTracker` sur l'objet du même nom, on dépose les marqueurs virtuels dans la liste `ArucoObjects` (*Voir* Figure 3.24).

5. En jouant la scène, les marqueurs virtuels sont placés aux même positions et orientations que les marqueurs physiques détectés (*Voir* Figure 3.25).

On montre dans l'Extrait 3.11 l'utilisation du module aruco pour faire le suivi des marqueurs seuls. Le suivi des échiquiers fonctionne sur le même principe, avec des fonctions spécifiques

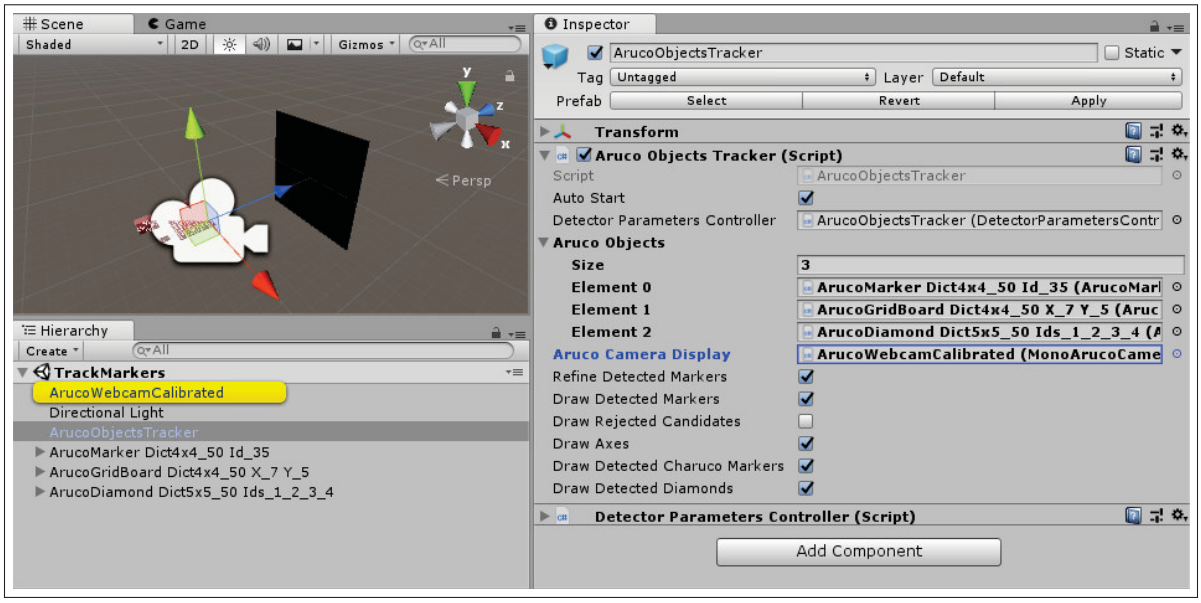


Figure 3.24 Configuration du suivi de marqueurs avec une liste de trois objets : deux échiquiers et du marqueur de la Figure 3.23. La caméra associée est surlignée en jaune.

a) Vue de dessus. b) Vue de face. c) Vue de côté.

Figure 3.25 Suivi du marqueur configuré à la Figure 3.23 dans plusieurs angles de vues.

de détection et d'estimation de position. Il fonctionne en quatre étapes : (1) on cache tous les marqueurs virtuels, (2) on détecte les marqueurs physiques sur l'image (extraction des points p comme durant l'étalonnage à la sous-section 3.4.3), (3) on détermine la position et l'orientation des marqueurs physiques détectés par rapport à la caméra physique et enfin (4) on place les

Extrait 3.11 Classe C# ArucoObjectsTracker simplifiée pour le suivi des marqueurs seuls.

```
public class ArucoObjectsTracker : ArucoCameraController,
  IArucoObjectsTracker
{
  public Dictionary<int, ArucoMarker> Markers; // Format: <id, marker>
  public ArucoCameraDisplay Display;

  private List<Aruco.Dictionary> dictionaries;
  private int length = 1; // Valeur fictive pour l'estimation

  protected void Start()
  {
    dictionaries = Markers.Select(m => m.Value.Dictionary).ToList();
    ArucoCamera.ImagesUpdated += TrackArucoObjects;
  }

  protected void TrackArucoObjects()
  {
    // (1) Masquer les éléments
    foreach (var marker in Markers.Values) {
      marker.gameObject.SetActive(false); }

    foreach (var dic in dictionaries)
    {
      // (2) Détection des marqueurs physiques sur l'image
      VectorVectorPoint2f corners; // Points p
      VectorInt ids; // Identifiants des marqueurs
      DetectMarkers(ArucoCamera.Image, dic, out corners, out ids);

      // (3) Estimation de la positions des marqueurs détectés
      VectorVec3d pos, rot;
      EstimatePoseSingleMarkers(corners, length, Display.Undistort.Knew,
        Display.Undistort.noD, rot, pos);

      // (4) Placement des marqueurs virtuels détectés
      for (i = 0; i < ids.Size(); i++)
      {
        if (markers.TryGetValue(ids[i], out marker))
        {
          var correctedPos = pos[i] * marker.Length / length;
          Display.PlaceArucoObject(marker, correctedPos, rot[i]);
        }
      }
    }
  }
}
```

marqueurs virtuels configurés dans ces mêmes positions et orientations par rapport à la caméra virtuelle.

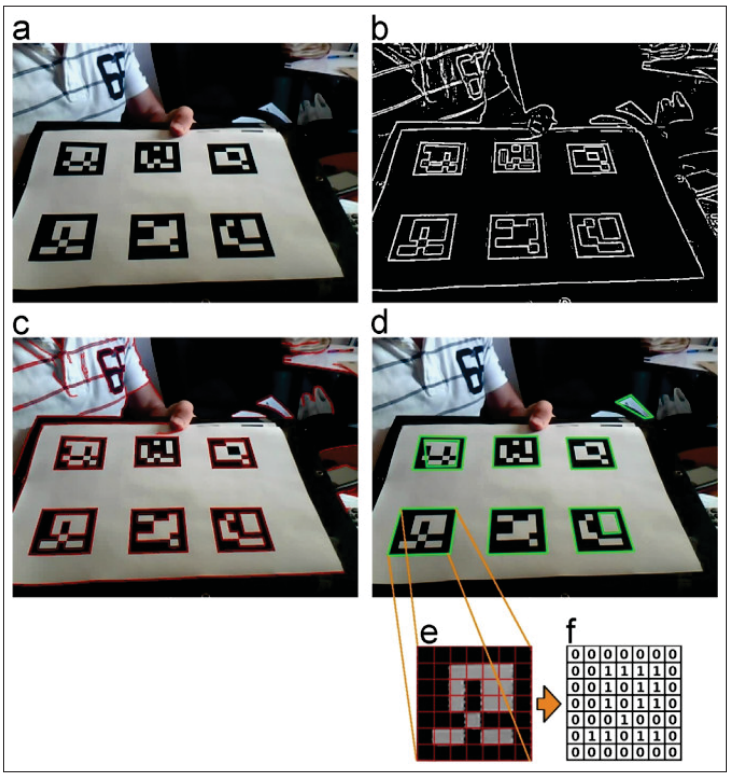


Figure 3.26 Procédé de détection des marqueurs du module aruco, utilisé avec un dictionnaire de marqueurs 5x5.
Tiré de Garrido-Jurado *et al.* (2014).

Comme les fonctions aruco détectent tous les marqueurs physiques sur l'image, nous devons faire une correspondance avec les marqueurs virtuels configurés. Le procédé de détection est simple et rapide à effectuer (*Voir* Figure 3.26) : (a) l'image est d'abord (b) segmentée pour permettre de (c) détecter les contours puis (d) d'extraire les polygones comme marqueurs potentiels. La (e) perspective est ensuite supprimée par reprojection en utilisant la matrice intrinsèque et (f) le code du marqueur est extrait. Chaque marqueur étant composé d'une grille de taille $n \times n$ de cellules noires ou blanches, entourée d'une bordure noire, il peut être facilement identifié par le code formé par les cellules. Ces identifiants sont connus d'avance, car la généra-

tion des marqueurs se fait avec un dictionnaire que l'on fournit à l'algorithme de détection, par exemple un dictionnaire de 50 marqueurs 4x4 dans la Figure 3.23 ou 5x5 dans la Figure 3.26. On fait alors la correspondance entre les marqueurs détectés et ceux configurés en utilisant ces identifiants.

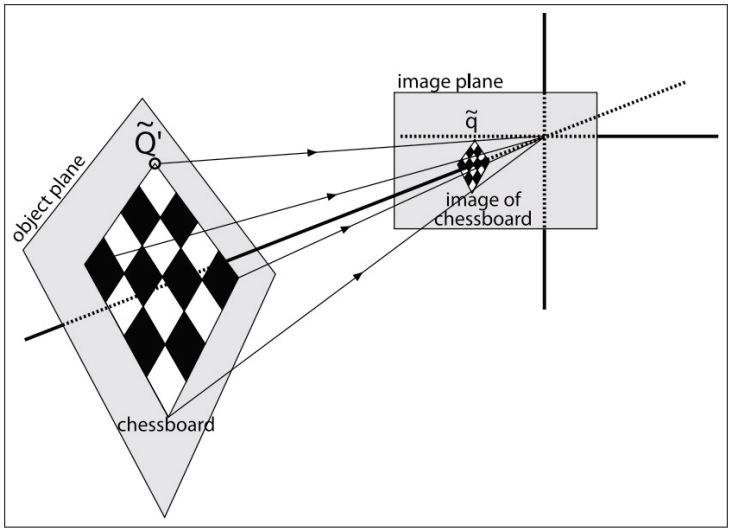


Figure 3.27 Un échiquier projeté sur le plan image d'une caméra. Les lignes construisant la projection montrent que de multiples échiquiers pourraient produite le même résultat sur le plan image.
Tiré de Bradski & Kaehler (2008).

Enfin, la position des marqueurs détectés est estimée avec une méthode de résolution au problème *perspective-n-points*, en utilisant la fonction `solvePnP` d'OpenCV (https://github.com/opencv/opencv/blob/master/modules/calib3d/src/solvepnp.cpp). La projection perspective faisant « perdre » une dimension, il existe de multiples solutions à la position d'un marqueur détecté : cela peut être l'image d'un petit marqueur proche de la caméra comme celle d'un grand marqueur loin de la caméra (*Voir* Figure 3.27). C'est pourquoi on doit fournir à l'algorithme la taille du marqueur, donc la longueur d'un côté, car c'est un carré. La fonction d'estimation de pose travaille en une fois avec tous les marqueurs détectés, en ne prenant qu'une seule longueur de côté. Comme l'on souhaite utiliser des marqueurs de tailles différentes, on fournit une longueur fictive pour ensuite corriger les positions calculées avec la vraie longueur des différents

marqueurs (*Voir* Extrait 3.11(4)). Ce *hack* (« triche ») nous évite de nombreux appels inutiles et coûteux à cette fonction.

3.4.7 Alignement avec une caméra stéréoscopique avec objectifs *fisheye*

3.4.7.1 Description

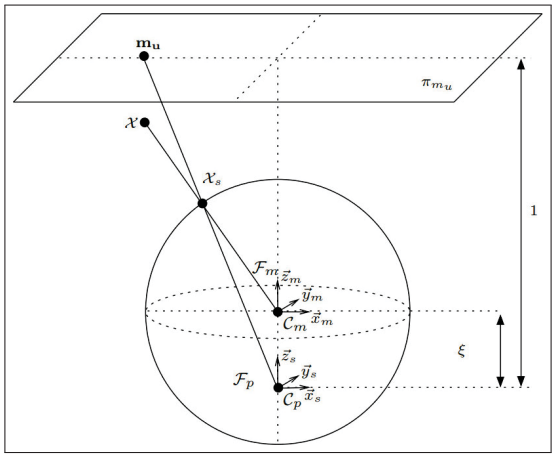


Figure 3.28 Illustration du modèle de Me & Rives (2007) : un point X de la scène est d'abord projeté en un point X_s sur une sphère puis projeté à nouveau en perspective rectilinéaire en un point p sur le plan image.
Tiré de Me & Rives (2007)

Les objectifs utilisés sur l'Ovrvision Pro étant *fisheye*, le modèle du sténopé ne s'y applique pas. En effet, contrairement à la perspective de l'œil humain ou des objectifs rectilinéaires, ces objectifs utilisent une distorsion en barillet volontairement importante pour capter un très grand champ de vision jusqu'à 180°.

Nous utilisons alors le modèle de Me & Rives (2007) utilisable pour les objectifs *fisheye* et implémenté par le module ccalib d'OpenCV. La caméra y est modélisée sur le principe suivant : la scène est d'abord projetée sur une sphère, puis une projection perspective rectilinéaire de la sphère est faite sur le plan image (*Voir* Figure 3.28). Comme pour une caméra avec un ob-

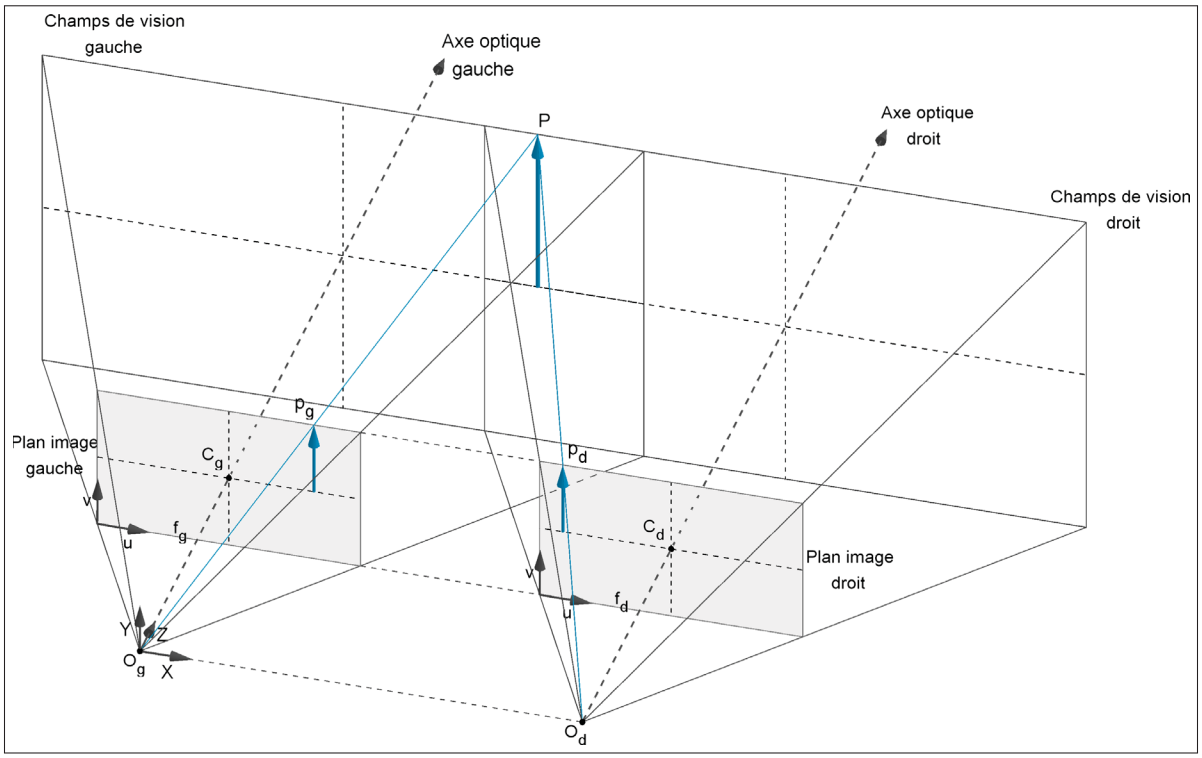


Figure 3.29 Modèle d'une caméra stéréoscopique rectilinéaire idéale : les deux plans images sont coplanaires, distants horizontalement, et les deux couples objectif-plan image ont les mêmes paramètres intrinsèques ($f_g = f_d$). Les objectifs pourraient également être placés verticalement.

jectif rectilinéaire, nous corrigeons ces distorsions : cela nous permet de reprojetter les images capturées en perspective rectilinéaire pour pouvoir les aligner avec la caméra virtuelle.

En outre, l'Ovrvision étant une caméra stéréoscopique nous avons besoin de déterminer sa géométrie. Sur une version idéale, les caméras gauche et droite auraient les mêmes paramètres intrinsèques et leurs plans images respectifs seraient coplanaires (donc leurs axes optiques parallèles) simplement séparés horizontalement de quelques centimètres (*Voir* Figure 3.29). Or nous avons vu que l'utilisation d'un objectif entraîne des distorsions radiales et tangentielles. À l'étalonnage, les caméras gauches et droites ont donc des paramètres intrinsèques légèrement différents et, leurs plans images n'étant pas coplanaires, on mesure deux nouveaux paramètres :

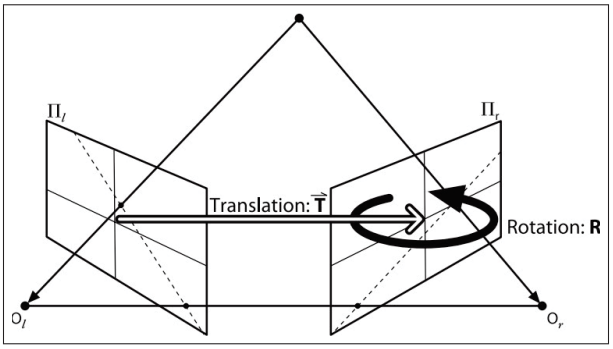


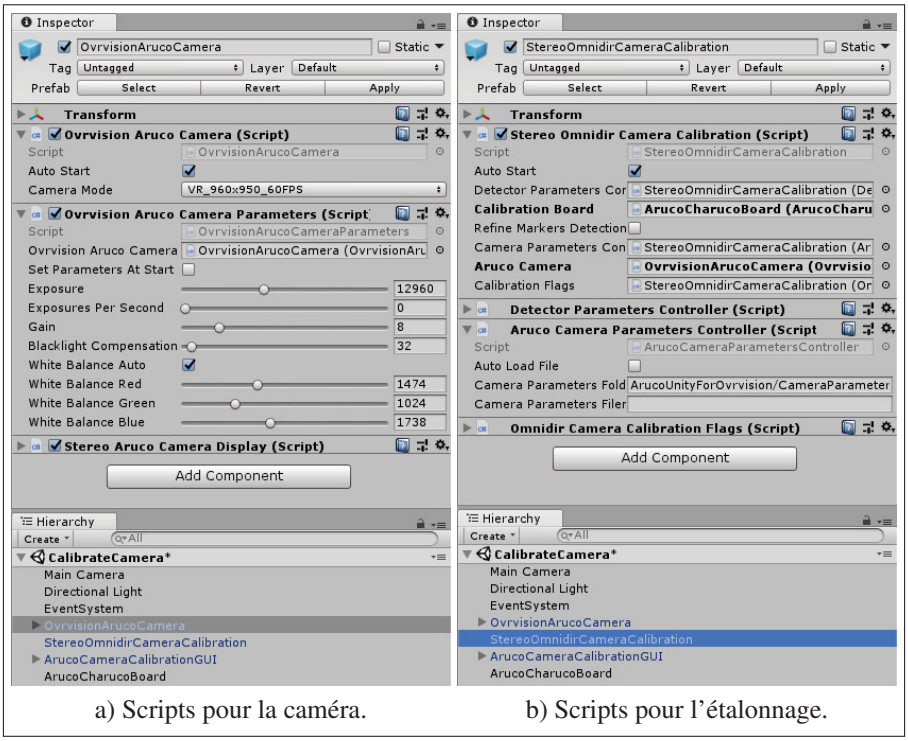
Figure 3.30 L'étalonnage d'une caméra stéréoscopique mesure en outre les vecteurs de translation T et de rotation R de chaque plan image droit Π_l par rapport au plan image gauche Π_r correspondant.
Tiré de Bradski & Kaehler (2008).

les vecteurs de translation T et de rotation R du plan image droit par rapport au plan image gauche (*Voir* Figure 3.30).

En plus de la correction de la caméra, nous devons la *rectifier* pour rendre à nouveau coplanaires les deux plans images. La correction étant faite en premier, le type d'objectif n'a pas d'impact sur la rectification. Comme pour les distorsions, nous n'avons pas besoin de comprendre la procédure de rectification ; pour plus de détails, nous redirigeons le lecteur vers (Bradski & Kaehler, 2008, p. 419).

3.4.7.2 Application dans ArucoUnity

Tous les procédés avec les caméras rectilinéaires que nous venons de décrire restent les mêmes (*Voir* sous-section 3.4.1). En revanche, nous utilisons quelques algorithmes différents, en créant simplement de nouveaux scripts spécifique d'étalonnage de correction t d'alignement de caméra. Comme la correction nous permet de travailler avec une caméra corrigée rectilinéaire, le script suivi de marqueurs reste inchangé. Nous avons cependant mis à jour tous les scripts vu ci-dessus pour qu'ils supportent des caméras capturant de multiples images, par exemple en remplaçant `Mat ArucoCamera.Image` par `Mat[] ArucoCamera.Image` initialisé au démarrage



a) Scripts pour la caméra. b) Scripts pour l'étalonnage.

Figure 3.31 Configuration de l'étalonnage dans Unity de la caméra Ovrvision.

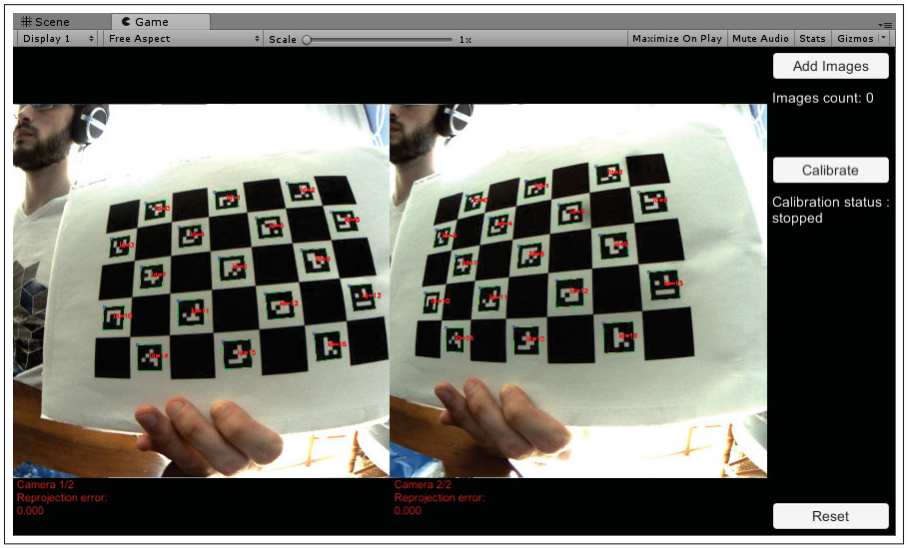


Figure 3.32 Étalonnage de la caméra Ovrvision.

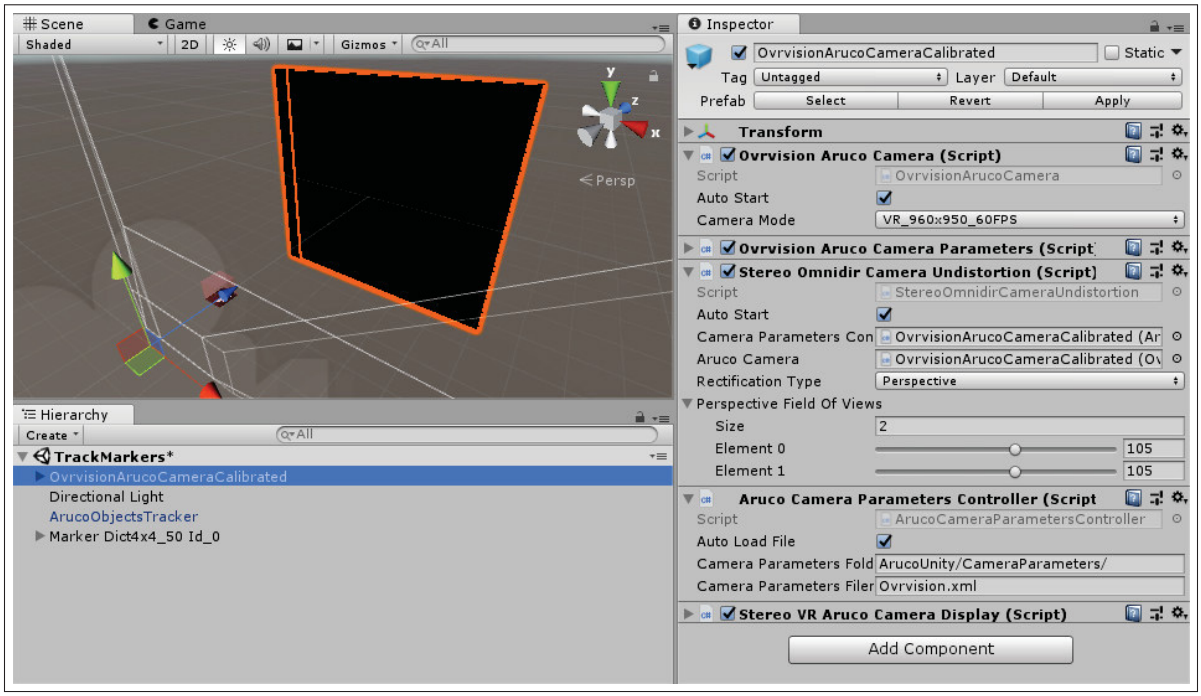


Figure 3.33 Configuration de la correction et la rectification de la caméra Ovrvision et de l'alignement d'une caméra virtuelle pour intégrer la RA dans notre visiocasque.

Figure 3.34 Vues des yeux gauche et droit dans notre visiocasque lors du suivi d'un marqueur.

Extrait 3.12 Classe C# StereoOmnidirCameraUndistortion simplifiée.

```
public class StereoOmnidirCameraUndistortion : ArucoCameraUndistortion
{
  public float[] Fovs;

  protected override void InitializeUndistortion()
  {
    // (1) Matrices intrinsèques pour la correction
    var newKs = new Mat[ArucoCamera.Images.Length]
    for (int i = 0; i < newKs.Length; i++)
    {
      float f = ArucoCamera.Resolutions[i].Height / (2f * Mathf.Tan(0.5f
        * Fovs[i] * Mathf.Deg2Rad));
      newKs[i] = new Mat(3, 3, Type.CV_64F, new double[9] {
        f, 0, ArucoCamera.Resolution[i].Width / 2,
        0, f, ArucoCamera.Resolution[i].Height / 2,
        0, 0, 1});
    }

    // (2) Rotations pour la rectification
    var R = new Mat[ArucoCamera.Images.Length];
    Omnidir.StereoRectify(CameraParameters.R, CameraParameters.T, out
      R[0], out R[1]);

    // (3) Fonction de mappage
    for (int i = 0; i < newKs.Length; i++)
    {
      Omnidir.InitUndistortRectifyMap(CameraParameters.K[i],
        CameraParameters.D[i], CameraParameters.Xi[i], R[i], newK[i],
        ArucoCamera.Resolutions[i], Type.CV_16SC2, out rectMapX[i], out
        rectMapY[i], Omnidir.Rectify.Perspective);
    }
  }
}
```

de l'application : tous les algorithmes que nous avons décrits sont alors exécutés pour chaque image capturée.

Pour l'étalonnage, la procédure à suivre est la même que dans la sous-section 3.4.3 à la différence que nous travaillons avec les objets `OvrvisionArucoCamera` pour la caméra et `StereoOmnidirCameraCalibration` pour l'étalonnage que nous glissons dans la scène depuis le dossier `Assets/ArucoUnity/Prefabs/` et nous désactivons `PinholeCameraCalibration` (*Voir* Figure 3.31). Ce nouveau script d'étalonnage que nous avons crée est très similaire

Extrait 3.13 Classe C# StereoVRArucoCameraDisplay simplifiée.

```
public class StereoVRArucoCameraDisplay : ArucoCameraDisplay
{
  protected override void Start()
  {
    // (1) Configuration des deux caméras virtuelles
    cameraFov[0] = Undistort.Fovs[0], cameraFov[1] = Undistort.Fovs[1];
    base.Start();

    // (2) Chaque caméra rend pour un oeil dans le visiocasque
    Cameras[0].stereoTargetEye = StereoTargetEyeMask.Left;
    Cameras[1].stereoTargetEye = StereoTargetEyeMask.Right;

    // (3) Placement des deux caméras
    Cameras[1].transform.localPosition += Undistort.CameraParameters.T;
  }
}
```

à Extrait 3.6 : nous y avons seulement remplacé la fonction d'étalonnage par `Cv.Omnidir.StereoCalibrate`. Cette fonction étalonne chaque caméra individuellement, puis calcule la position et l'orientation relative des deux plans images, et retourne tous ces paramètres que l'on enregistre dans un fichier d'étalonnage.

On met finalement en place la RA dans notre visiocasque. Comme pour l'étalonnage, la procédure est la même que dans les sous-sections 3.4.4 et 3.4.5, mais on utilise des objets différents : on désactive donc les objets de correction et d'affichage pour la webcam et on glisse dans la scène ceux pour une caméra stéréoscopique *fisheye* ainsi que `OvrvisionArucoCamera` (*Voir* Figure 3.33). Contrairement à la correction des caméras rectilinéaires, les fonctions du module omnidir nous permettent de choisir directement le champ de vision des caméras gauches et droites corrigées. On les définit toutes les deux à un champ de vision vertical de 105 deg, comme le visiocasque de RV que nous nous utilisons. Les marqueurs et leur suivi une fois configurés, notre visiocasque fonctionne comme nous l'avons décrit à la sous-section 3.2.1 (*Voir* Figure 3.34) !

Nous effectuons la correction et la rectification dans le même temps (*Voir* Extrait 3.12) : nous (1) créons les matrices intrinsèques pour les caméras corrigées rectilinéaires à partir des champs de visions choisis, puis on calcule (2) la rotation des deux plans images pour les rectifier, enfin (3) la fonction de mappage pour chaque caméra. Ces fonctions sont toujours appliquées sur toutes les images capturées dans le script parent (*Voir* Extrait 3.7).

Pour la configuration des caméras virtuelles, nous devons tout d'abord activer la RV dans Unity avec l'option `Virtual Reality Supported` dans le menu `Edit/Project Settings /Player/XR Settings`. Nous (1) créons et configurons ensuite deux caméras virtuelles, une pour chaque œil avec les champs de visions choisis. Puis nous (2) indiquons à Unity pour quel œil chaque caméra rend. Enfin, on (3) décale la caméra droite suivant le vecteur T.

CHAPITRE 4

ÉTUDE EXPÉRIMENTALE

4.1 Tâche expérimentale

Nous souhaitons évaluer dans cette expérience les avantages d'un téléphone dont l'écran étendu par un visiocasque de RA à large champ de vision, ainsi que comparer deux techniques d'interaction : navigation, pointage et sélection avec l'écran tactile ou avec une main virtuelle.

Plusieurs tâches auraient pu convenir à cette expérience. Comme le VESAD est une technique relativement nouvelle et peu étudiée (Grubert *et al.*, 2015), et la technique d'interaction de la main virtuelle peu maîtrisée (Argelaguet & Andujar, 2013; Piumsomboon *et al.*, 2013), nous avons besoin d'une tâche fondamentale demandant de la navigation, dont des zooms, et des sélections (Bowman *et al.*, 2004). Nous avons alors envisagé une tâche de pointage de Fitts (Soukoreff & MacKenzie, 2004; Bergé *et al.*, 2014), où l'utilisateur doit chercher et sélectionner des cibles, de navigation dans un large document, comme une carte (Baudisch *et al.*, 2002; Rädle *et al.*, 2014), ou encore un scénario de multi-tâches où le participant doit travailler avec plusieurs fenêtres (Czerwinski *et al.*, 2003; Ens *et al.*, 2014b).

Nous avons finalement décidé de répliquer la tâche de classification de Liu *et al.* (2014) : sa validité externe est plus importante (plus réaliste) qu'une pure tâche de pointage, demande beaucoup de navigations et de reconstruction mentale comme un large document tout en demandant de sélectionner et manipuler le contenu, enfin est plus facile à implémenter et à contrôler qu'un scénario multi-tâches. Cette tâche consistait en une grille de plusieurs cellules contenant des disques à classer (*Voir* Figure 1.36). Les participants devaient déplacer les disques mal classés dans la bonne cellule, avec des disques de même type. Une lettre inscrite en leur centre permettait de connaître leur type et les disques mal classés étaient colorés en rouge : cette indication visuelle facilitait la recherche et équilibrait avec le temps consacré au classement. Une

tâche plus réaliste demanderait d'examiner en détail chaque disque pour savoir les classer (Liu *et al.* prennent l'exemple de catégoriser les articles d'une conférence). Aussi, la petite taille des lettres sur les disques et la grandeur de la grille imposaient aux participants des zooms et défilements. Aussi, la grille était grande pour imposer des défilements Enfin, les classements se faisaient par un pointage vers le disque puis une sélection par un bouton, puis un second pointage et sélection dans la cellule. Le but de la tâche peut être résumé aux participants avec la consigne de [traduction] « tout mettre en vert » (Liu *et al.*, 2014).

Notre grille comporte 3×5 cellules rectangulaires, pouvant chacune contenir six disques maximum. La grille est générée aléatoirement à chaque nouvel essai, avec cinq disques par cellules dont au moins quatre correctement classés, et cinq disques mal classés au total dans la grille (*Voir* Figure 4.1). Il y a donc 15 cellules $\times$ 5 disques/cellule $=$ 75 disques dans la grille. Au début de chaque essai (pour toutes les conditions), elle est affichée à l'échelle, la taille d'une cellule étant celle de l'écran du téléphone que nous avons utilisé, soit $68\,\text{mm} \times 121\,\text{mm}$ (un disque mesure alors $33\,\text{mm} \times 33\,\text{mm}$) (*Voir* section 4.4). Ainsi, la grille a une forme approximativement carrée, un peu plus grande ($34\,\text{cm} \times 36{,}3\,\text{cm}$) que la condition tablette de Rädle *et al.* (2014). La vue de la grille est de taille limitée dans toutes les conditions : elle est soit limitée par l'écran du téléphone, soit nous donnons une taille fixée à l'écran étendu. Dans ce dernier cas, nous avons arbitrairement choisi de limiter l'écran étendu à la taille de la grille au début de l'essai ; ainsi, si un participant zoome la grille avec l'écran étendu, sa vue sera tronquée à $34\,\text{cm} \times 36{,}3\,\text{cm}$.

La tâche de Liu *et al.* (2014) opérationnalise un scénario d'allocation de ressources (par exemple, chaque catégorie a une capacité limitée). Le texte y est alors une variable indépendante importante. N'étudiant pas cette problématique, nous avons décidé d'utiliser plus simplement une seule lettre par cellule, de A à O. Chaque disque « appartient » donc à une cellule et ne peut y être correctement classé que dans celle-ci. Nous colorons également les items correctement

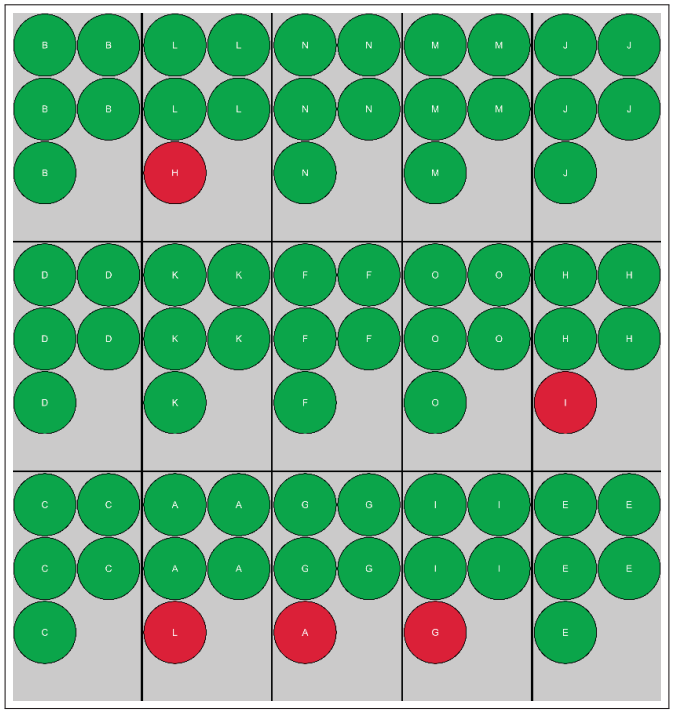


Figure 4.1 La grille de notre tâche expérimentale au début d'un essai dans la condition TAILLE=*Grand*.

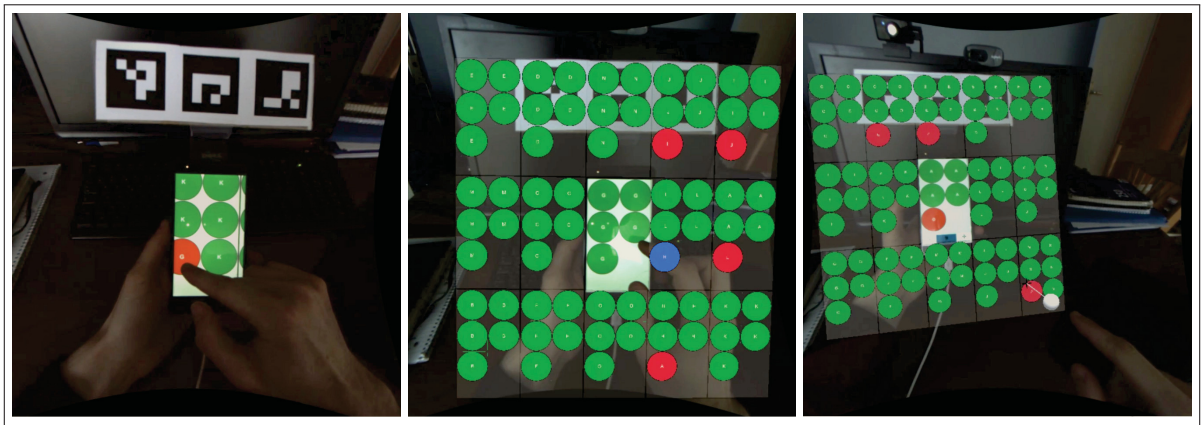
classés en vert et mal classés en rouge. Nous colorons en outre un disque sélectionné en bleu (*Voir* Figure 4.2b). De sorte qu'en participant doit lire la lettre d'un disque vert pour savoir le type de la cellule.

Nous avons placé le code source de l'expérience en ligne (https://github.com/NormandErwan/MasterThesisExperiment), sous la licence libre BSD-3, pour qu'il pouvoir être reproduit ou réutilisé au besoin. Nous l'avons développé et testé pour Unity 2017.

4.2 Plan expérimental

Nous avons manipulé simultanément les variables indépendantes suivantes :

- IHM : l'IHM évaluée, soit utilisation ou non d'un écran étendu, et la technique d'interaction :



a) Condition *Téléphone* b) Condition *VESAD tactile*. c) Condition *VESAD*.

Figure 4.2 La grille de notre tâche expérimentale pour chaque IHM. Pour *VESAD tactile*, la main de l'utilisateur est cachée par l'écran étendu et un disque est sélectionné (en bleu). Pour *VESAD*, une sphère blanche indique la position repérée de l'index de l'utilisateur, une croix sur la grille est la projection de cette sphère et un segment noir les relie ; ils deviennent bleus quand la sphère touche la grille.

- *Téléphone* : affichage et interactions seulement sur le téléphone (*Voir* Figure 4.2a) ;
- *VESAD tactile* : affichage sur l'écran étendu et interactions sur le téléphone (*Voir* Figure 4.2b) ;
- *VESAD* : affichage sur l'écran étendu et interactions dans la partie virtuelle de l'écran avec une main virtuelle (*Voir* Figure 4.2c).

- TAILLE : la taille du texte des disques :
  - *Grand* : 12 pt, soit 4,22 mm × 4,22 mm ou 12% du diamètre du disque ;
  - *Petit* : 10 pt, soit 3,51 mm × 3,51 mm ou 10% du diamètre du disque.
- DISTANCE : la distance moyenne des disques à classer avec leurs cellules respectives :
  - *Proche* : entre 1,25 et 1,45 ;
  - *Loin* : entre 2,5 et 2,75.
- GROUPE : l'ordre de passage des IHMs (*Voir* Tableau 4.1).

La variable indépendante IHM combine les deux affichages et les deux techniques d'interaction que nous souhaitons évaluer. La combinaison affichage sur le téléphone seul et interaction avec une main virtuelle ne nous intéressait pas. Jones *et al.* (2012) l'ont déjà été évaluée contre le téléphone seul (*Voir* sous-section 1.2.2).

La TAILLE et la DISTANCE nous permettent de contrôler la difficulté de la tâche. En réduisant la TAILLE de la lettre dans chaque disque, on impose au participant d'effectuer des zooms plus importants pour les lire, ce qui l'empêche alors de voir beaucoup de disques à la fois. En augmentant la DISTANCE, on impose plus de recherches au participant pour trouver la cellule correspondant à un disque à classer. Dans les deux cas, on lui demande un effort de mémoire plus important.

Comme Liu *et al.*, nous utilisons la distance euclidienne pour calculer la DISTANCE entre un disque mal classé et sa cellule respective. Nous considérons dans le calcul qu'une cellule fait une unité de largeur et une unité de hauteur : il y a donc une distance de 1 entre deux cellules adjacentes. La variable indépendante donne alors le nombre de cellules d'écart entre un disque mal classé et sa cellule respective, par exemple entre 2,5 et 2,75 cellules d'écart en moyenne pour la condition *Loin*. Les valeurs que nous utilisons sont celles de Liu *et al.* (2014). À la génération de la grille, on vérifie qu'elle respecte la condition courante, autrement on en régénère une nouvelle. Nous avions mené une étude pilote avec trois participants, demandant de classer dix disques avec de plus petites tailles de texte. Cependant, cette configuration rendait les essais particulièrement éprouvants et longs à compléter. C'est pourquoi nous avons réduit le nombre de disques à classer et augmenté la taille du texte (tout en la gardant suffisamment petite pour que les participants aient besoin de zoomer).

Notre plan expérimental est quasi complet. Les variables indépendantes IHM, TAILLE et DISTANCE sont croisées : les participants passent à travers toutes les conditions (les combinaisons des valeurs des variables). Cependant, pour contrôler un effet d'apprentissage parasite à travers

les IHMs, les participants sont partagés en trois GROUPE emboîtés : chaque groupe passe à travers les conditions IHM dans un ordre différent, suivant un carré latin 3×3 (*Voir* Tableau 4.1). L'ordre de passage de la TAILLE et de la DISTANCE est par contre fixe : du plus simple (avec les conditions *Grand* et *Proche*), au plus difficile (*Petit* et *Loin*). Nous avons donc mesuré 12 participants $\times$ 3 IHMs $\times$ 2 distances $\times$ 2 tailles = 144 observations. Pour réduire la variabilité de nos résultats, chaque participant a répété deux essais par condition, que nous avons ensuite moyennés pour avoir une observation par participant, et non par essai (Dragicevic, 2016, p. 24).

Tableau 4.1 Ordre de passage pour chaque GROUPE de chaque IHM suivant un carré latin.

Groupe	**IHM 1**	**IHM 2**	**IHM 3**
1	*Téléphone*	*VESAD tactile*	*VESAD*
2	*VESAD tactile*	*VESAD*	*Téléphone*
3	*VESAD*	*Téléphone*	*VESAD tactile*

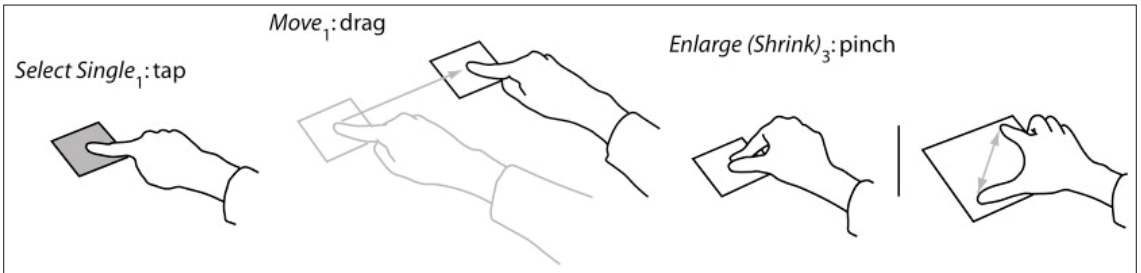
Enfin, les participants ont porté le visiocasque de RA (*Voir* chapitre 3) dans toutes les conditions, même dans la condition *Téléphone* qui ne bénéficie pas de la RA, pour garder identique le poids sur la tête ainsi que la même résolution et latence de l'image.

4.3 Techniques d'interactions

Les interactions des participants consistent à déplacer les disques pour les classer et naviguer dans la grille (par défilements et zooms). Le déplacement d'un disque consistait à, quelle que soit la condition IHM : (1) pointer puis sélectionner le disque puis (2) sélectionner un espace libre dans une cellule. Une deuxième sélection sur le disque sélectionné permettait de le désélectionner, tandis que sélectionner un second disque désélectionnait automatiquement le précédent. Si un disque est déposé dans une mauvaise cellule, il est alors coloré en rouge, et

cela est compté comme une erreur, augmentant d'un le nombre de disques à classer pour l'essai courant.

Avec le *Téléphone* et le *VESAD tactile*, les interactions se font via l'écran tactile : la sélection d'un disque se fait par un *tap* (*Voir* Figure 4.3a), le défilement par un *pan* (*Voir* Figure 4.3b) et le zoom par un *pinch* (*Voir* Figure 4.3c). Ces gestes, recommandés par Wobbrock *et al.* (2009), sont standards et connus des utilisateurs de téléphones intelligents (par exemple pour Android : https://material.io/design/interaction/gestures.html). Il s'agit d'un défilement statique (Mehra *et al.*, 2006). Ainsi, l'écran étant étendu ou non, il faut donc d'abord amener sur l'écran tactile le disque ou la cellule que l'on souhaite sélectionner, en déplaçant la grille par un défilement ou un zoom.



a) Sélection par un *tap* : un appui bref ($<$ 500 ms) de l'index.

b) Défilement par un *pan* : un déplacement de l'index.

c) Zoom par un *pinch* : un déplacement de l'index et du pouce, le changement de distance entre les deux doigts contrôlant le zoom.

Figure 4.3 Les différents gestes utilisés par les participants sur l'écran tactile du téléphone pour le *Téléphone* et le *VESAD tactile*.
Adapté de Wobbrock *et al.* (2009).

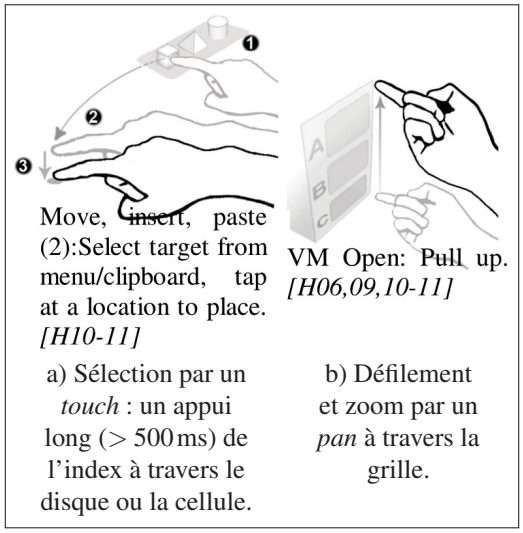
Pour le *VESAD*, les interactions se font via une main virtuelle, suivant les mouvements avec 6 DoFs de la main dominante du participant. La sélection se fait avec un appui long sur le disque ou la cellule à sélectionner (*Voir* Figure 4.4a), le défilement et le zoom par un *pan* à travers la grille (*Voir* Figure 4.4b). Ces trois techniques d'interactions utilisent donc le geste simple de pointer et déplacer l'index dans le contenu virtuel ; il s'agit du geste H11 que Piumsomboon

et al. (2013) recommandent (*Voir* Figure 1.13). Pendant le défilement, la grille suit simplement les déplacements de l'index. Pour le zoom, la grille grossit quand l'index la « tire » vers (quand il s'éloigne de) l'écran, mais se réduit quand il la « pousse » vers (quand il se rapproche de) l'écran.

Nous avions tout d'abord utilisé un *tap* pour la sélection et un geste de *pinch* pour le zoom dans l'étude pilote, mais il était difficile de faire de tels appuis brefs avec précision et, l'index et le pouce étant proches, il était courant de déclencher involontairement un zoom pendant un défilement. Nous avons donc ajouté simplement ajouté trois boutons en bas de l'écran du téléphone pour choisir entre activer la sélection, le défilement ou le zoom : le participant appuie sur le bouton correspondant avec le pouce de sa main tenant le téléphone pour changer le mode d'interaction. Ces boutons ne cachent pas les disques de la cellule (*Voir* Figure 4.2c) tout en restant assez gros (22 mm $\times$ 22 mm). Piumsomboon *et al.* (2013) n'ayant pas proposé de geste pour déplacer du contenu, nous utilisons celui nous semblant le plus facile, en utilisant un index. En outre, nous avons ajouté des retours visuels de la main virtuelle au participant pour le *VESAD* comme le conseille Cha *et al.* (2010). Nous avons ajouté des retours continus : une sphère blanche pour indiquer la position repérée de l'index de la main dominante, la projection de cette sphère sur la grille sous forme d'une croix noire, ainsi qu'un segment noir les reliant tous les deux (*Voir* Figure 4.2c). Nous avons aussi ajouté un retour discret pour confirmer quand l'index touche la grille : la sphère, la croix et la ligne sont alors colorés en bleu.

Pour les trois IHMs, le défilement a un gain de 1 : 1, tandis que le zoom est centré sur le téléphone et non sur la grille, comme le conseille Guiard & Beaudouin-Lafon (2004) et que nous avons vérifié en développant l'expérience : [traduction] « The focus point is generally coincident with the center of the view, more rarely with the cursor position. ».

Enfin, dans les conditions avec le *VESAD tactile* ou le *VESAD*, la grille est affichée par-dessus les images de la caméra (*Voir* sous-section 3.2.1). La main dominante utilisée pour les interac-



a) Sélection par un *touch* : un appui long ($>$ 500 ms) de l'index à travers le disque ou la cellule.

b) Défilement et zoom par un *pan* à travers la grille.

Figure 4.4 Les différents gestes utilisés par les participants avec la main virtuelle pour le *VESAD*.
Adapté de Piumsomboon *et al.* (2013).

tions est donc cachée en permanence par la partie virtuelle de la grille même quand le participant la place en avant du téléphone (*Voir* Figure 4.2b). Cela peut causer un sentiment étrange en créant un conflit avec les autres indices de profondeurs, l'expérience pouvant alors devenir inconfortable. Nous avons donc réduit ce problème d'occlusion d'une part en rendant la grille semi-transparente pour toujours laisser visible la main, proche de la solution de Piumsomboon *et al.* (2014) et, d'autre part, en indiquant la position repérée de l'index par une sphère blanche pour le *VESAD*. Aucun participant n'a fait de remarque à ce sujet.

4.4 Matériel

Nous avons bien évidemment utilisé notre visiocasque de RA, conçut avec un large champs de vision pour cette expérience et décrit au chapitre 3. Pour le faire fonctionner, nous avons utilisé un ordinateur de bureau sous Windows 10, avec un processeur Intel Core i5 7400 ($4 \times 3{,}0$ GHz), 8 GB DDR4 de mémoire vive, une carte graphique NVIDIA GeForce GTX 1060 de 6 GB. Pour le téléphone, nous avons utilisé un Xiaomi Redmi Note 4 : sous Android

7, il est récent et léger, à faible prix et possède une bonne puissance de calcul ainsi qu'écran 1920 px × 1080 px de 5,5 po. Le suivi du téléphone était fait grâce à notre bibliothèque de RA ArucoUnity (section 3.3) avec trois marqueurs imprimés sur une planche rigide fixée à l'arrière du téléphone (*Voir* Figure 4.5).

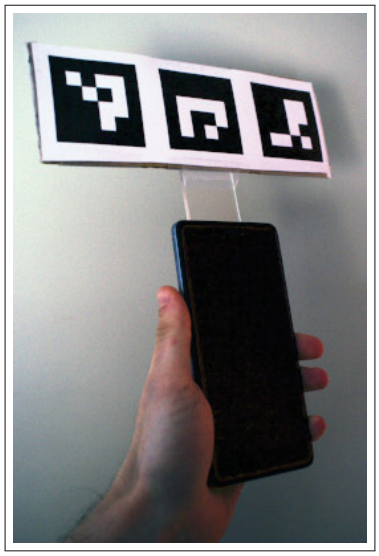


Figure 4.5 Le téléphone était suivi avec 6 DoFs par des marqueurs grâce à ArucoUnity.

Pour la localisation des mains nous avons utilisé un Leap Motion : c'est un dispositif peu dispendieux, particulièrement utilisé pour concevoir des IHMs avec une main virtuelle pour les visiocasques de RV et très bien intégré avec les moteurs de jeu Unity et Unreal Egine. Nous l'avons fixé la face avant du visiocasque sous la caméra (*Voir* Figure 3.3), et se connecte simplement au PC par USB 3.0.

Enfin, nous avons développé la bibliothèque DevicesSyncUnity pour synchroniser le visiocasque avec le téléphone (https://github.com/NormandErwan/DevicesSyncUnity). Basée sur la bibliothèque de mise en réseau haut-niveau UNet, fournie avec Unity, elle nous permet de facilement synchroniser les affichages sur le visiocasque et sur le téléphone en fonction des actions

de l'utilisateur, donnant le sentiment d'interagir avec un seul appareil. Il existe une latence perceptible de quelques centièmes de seconde, mais aucun participant n'en a fait la remarque.

4.5 Procédure de l'expérience

Nous avons tout d'abord demandé à chaque participant de lire soigneusement une copie imprimée du formulaire d'information et de consentement, en répondant à leurs questions. Nous avons ensuite ré-expliqué à l'oral les points les plus essentiels : pour qui est faite la recherche, le déroulé de l'expérience, la tâche, la garantie de l'anonymat de leurs données, enfin la possibilité de prendre une pause, de quitter l'expérience ou de demander la suppression de leurs données n'importe quand. Si la personne souhaitait participer, nous lui demandions de signer le formulaire d'information et de consentement, lui en proposions une copie, et lui faisions remplir le pré-questionnaire.

Nous faisions ensuite tester le visiocasque au participant, d'abord en mode RV, en utilisant l'écran d'accueil de l'Oculus, car moins susceptible de provoquer des nausées : il n'y a aucune latence perceptible dans ce mode et une meilleure densité visuelle de l'image (*Voir* sous-section 3.2.3). Puis en activant les caméras, nous demandions au participant de regarder autour de lui et de bouger ses mains pour s'habituer à la latence et à la dynamique plus restreinte de l'image (des ombres vues par l'œil humain sont totalement noires dans le visiocasque, comme on peut le voir sur la Figure 4.2a). Nous donnions alors le téléphone dans la main non-dominante du participant pour qu'il puisse démarrer les essais. Il y avait, pour chaque IHM, un essai d'entraînement au début (pas de consignes données et aucune mesures) et une pause obligatoire à la fin. L'ensemble des essais durait environ 50 minutes. Avant chaque essai, nous demandions au participant d'« aller le plus vite possible en faisant le moins d'erreurs possibles ». Nous lui précisions qu'une erreur était comptée seulement s'il déplaçait un disque dans une mauvaise cellule.

Une fois les essais terminés, nous faisions remplir le post-questionnaire pour recueillir les mesures subjectives, ainsi que le formulaire de compensation et leur donnions 20$ en échange de leur participation.

L'Annexe I contient les deux formulaires et les deux questionnaires.

4.6 Mesures

Dans notre pré-questionnaire, nous demandions d'abord la catégorie d'âge (catégories de cinq ans), le sexe et les problèmes de vues éventuel du participant. Nous demandions ensuite leur main dominante et la main utilisant la souris : quelques participants gauchers utilisaient leur main droite avec la souris, mais ont tout-de-même préféré tenir le téléphone avec leur main non dominante, après essais. Nous demandions enfin leur utilisation de l'ordinateur, de logiciels 3D et leurs expériences passées en VR et RA. Nous ne nous sommes pas servis de ces mesures dans nos analyses.

Nous avons effectué les mesures des essais seulement sur l'ordinateur, comme nous l'avions synchronisé avec le téléphone. Nous avons tout d'abord utilisé, comme principales mesures de la performance, le temps de complétion de chaque essai ainsi que le nombre d'erreurs (les disques placés dans une mauvaise cellule par le participant). Nous avons compté, en complément des erreurs, le nombre de disques sélectionnés pendant un essai : pendant sa recherche, un participant peut oublier le disque qu'il avait sélectionné ou décider de changer de disque à classer ; une sélection d'un nouveau disque désélectionne automatiquement le précédent, sans que ce soit considéré comme une erreur (cela était précisé aux participants). Le nombre de sélections peut être résumé ainsi : sélections $=$ 5 disques à classer $+$ erreurs $+$ sélections annulées.

Nous avons en outre fait des mesures sur la navigation des participants lors des essais : le nombre de défilements et de zooms, ainsi que le temps passé et la distance parcourue durant

ces opérations. Même si la main virtuelle était désactivée pour certaines conditions, nous suivions en continu la position de la main dominante, lors des essais ; la distance que nous avons enregistrée est celle de la projection de l'index de la main sur la grille. C'est une mesure simple mais fonctionnelle et permettant de comparer chaque IHM. Nous avons également compté le temps passé avec un disque sélectionné, pour quantifier le temps de recherche. Enfin, nous avons enregistré la distance entre la tête et le téléphone pour mesurer la navigation physique.

Notre post-questionnaire nous a permis de relever des mesures subjectives sur chaque IHM. Nous nous sommes principalement inspiré du NASA TLX Rubio *et al.* (2004), avec des questions portant sur l'exigence mentale, l'exigence physique, la performance et la frustration, auxquelles nous avons ajouté des questions sur la facilité de compréhension et la rapidité. Pour chaque question, le participant notait les trois IHMs sur une échelle de Likert (de 1 à 5). Nous avons par contre supprimé la question sur la charge temporelle, que nous trouvions redondante avec les autres. Le participant devait ensuite classer par ordre de préférence les trois IHMs. Enfin, nous avons présenté les scénarios d'applications, dont la carte de navigation en vue étendue, une application multi-fenêtres et des notifications en 3D, et recueilli leurs commentaires.

4.7 Participants

Nous avons recruté 16 personnes volontaires pour participer à l'expérience parmi notre entourage. Cependant, nous avons conservé les données de 12 d'entre eux : il y a eu, d'une part, des erreurs de mesures pour deux d'entre elles et, d'autre part, deux personnes ont arrêté l'expérience après l'entraînement, le visiocasque leur donnant des nausées.

Ces 12 participants, dont trois femmes, étaient âgés entre 18 et 49 ans (deux au-dessus de 25 ans). Tous avaient une vision normale ou portaient un dispositif de correction de vision ; une personne a dit ne pas savoir distinguer les couleurs pastels mais a été capable, lors de l'entraînement, de reconnaître les éléments à classer sur la grille. Dix étaient droitiers et deux

gauchers. Huit d'entre eux avaient déjà utilisé un visiocasque de RV, dont un avait déjà utilisé plusieurs visiocasques de RA. Tous utilisent tous les jours un ordinateur dont huit utilisent régulièrement des logiciels avec un environnement 3D.

4.8 Résultats

Nos mesures agrégées des essais et du post-questionnaire ainsi que nos analyses sont disponibles en ligne (https://github.com/NormandErwan/HandheldVesadAnalysis), sous la même licence que ce mémoire (Creative Commons BY-NC-ND). Nous gardons privées les informations du pré-questionnaire afin que les participants ne puissent être identifiés.

Toutes les barres d'erreurs dans nos figures montrent l'intervalle de confiance à 95%. De même, quand nous reportons une valeur X, nous précisons son IC à 95% ainsi : X [IC_{bas} ; IC_{haut}]. Nous calculons tous les ICs en utilisant la technique de *bootstraping* (Dragicevic, 2016, p. 25), avec $B = 1000$ simulations.

4.8.1 Temps de complétion

On transforme tout d'abord les mesures de temps de complétion d'un essai avec un logarithmique népérien pour rendre leurs distributions approximativement normales et réduire l'influence des mesures extrêmes (Dragicevic, 2016, p. 25). La Figure 4.6 montre les distributions des temps mesurés et transformés. On vérifie ensuite la normalité des distributions avec le test de Shapiro-Wilk [1] (Wobbrock & Kay, 2016) pour chaque IHM, car c'est la variable indépendante qui nous importe le plus : elles sont normales pour les trois IHMs *Téléphone* ($W = 0{,}98$; $p = 0{,}76$), *VESAD tactile* ($W = 0{,}97$; $p = 0{,}45$) et *VESAD* ($W = 0{,}97$; $p = 0{,}45$). En outre, le

1. L'hypothèse nulle testée est que la distribution suit la loi normale ; on attend donc une valeur-p supérieure à 0,05 pour ne pas la rejeter.

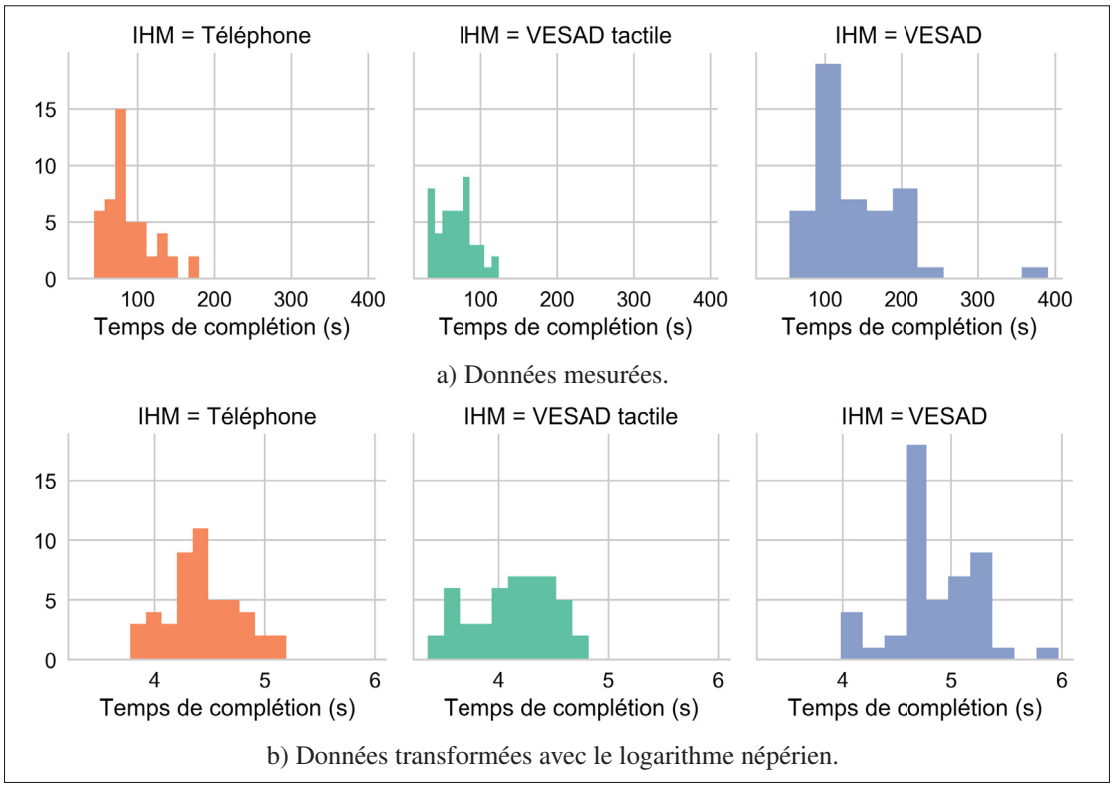


Figure 4.6 Histogrammes du temps de complétion d'un essai pour chaque IHM.

test de Levene [2] ($W = 0{,}76$; $p = 0{,}47$) nous permet de vérifier que leurs variances sont égales (Wobbrock & Kay, 2016).

On analyse alors l'effet de toutes les variables indépendantes sur le temps de complétion avec une analyse de variance (ANOVA). Nous pouvons utiliser ce test, car les distributions testées sont normales, indépendantes et ont la même variance (Wobbrock & Kay, 2016). On utilise le modèle suivant : TEMPS $\sim$ IHM $\times$ TAILLE $\times$ DISTANCE + IHM $\times$ GROUPE. Le Tableau 4.2 donne les résultats : les variables IHM ($F_{2,22} = 62{,}2$; $p = 2 \times 10^{-19}$) et GROUPE ($F_{2,22} = 2{,}1$; $p = 5 \times 10^{-5}$), ainsi que leur interaction IHM $\times$ GROUPE ($F_{4,44} = 3{,}1$; $p = 1 \times 10^{-5}$) ont un effet significatif sur le temps de complétion.

2. L'hypothèse nulle testée est que les variances des distributions sont égales.

Tableau 4.2 Résultats de l'ANOVA sur le temps de complétion pour toutes les variables indépendantes.

Effet	**Somme des carrés**	**Degrés de liberté** [3]	**Degrés de liberté de l'erreur** [4]	**F**	**p**
IHM	12,3	2	22	62,2	2×10^{-19}
TAILLE	0,1	1	11	1,5	2×10^{-1}
DISTANCE	0,03	1	11	0,3	6×10^{-1}
GROUPE	2,1	2	22	10,8	5×10^{-5}
IHM×TAILLE	0,4	2	22	2,0	1×10^{-1}
IHM×DISTANCE	0,07	2	22	0,4	7×10^{-1}
TAILLE×DISTANCE	0,3	1	11	2,9	9×10^{-2}
IHM×GROUPE	3,1	4	44	7,8	1×10^{-5}
IHM×TAILLE×DISTANCE	0,5	2	22	2,5	8×10^{-2}
Résidus	12,5	126			

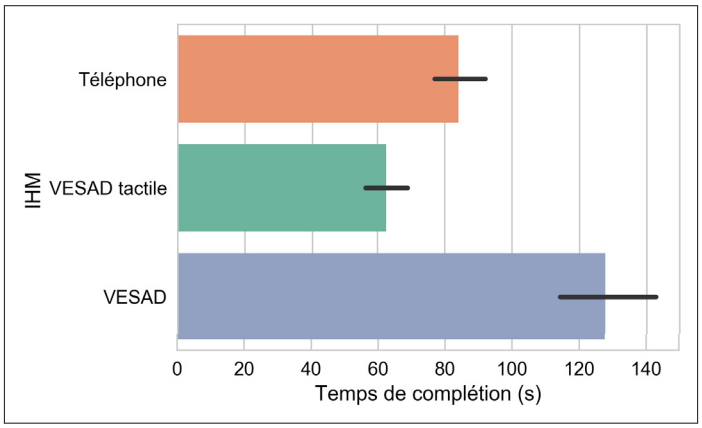


Figure 4.7 Temps de complétion moyen (moyenne géométrique) par essai pour chaque IHM.

On les compare alors deux-à-deux avec le test t de Student, avec une correction de Benjamini-Hochberg pour limiter le nombre de faux positifs (*Voir* sous-section 1.2.4). Ce test suppose aussi la normalité, l'indépendance et une variance égale des distributions. Les résultats, au Tableau 4.3, indiquent des différences significatives entre les trois IHMs. On calcule alors les moyennes avec intervalles de confiances à 95% sur les données transformées. En y appliquant la fonction exponentielle, on obtient donc les moyennes géométriques. La Figure 4.7 donne

une donc une bonne preuve que les différences sont également importantes : le *VESAD tactile* est plus rapide de 22 s (+33%) que le *Téléphone*, lui-même plus rapide de 49 s (+36%) que le *VESAD*.

Tableau 4.3 Résultats des tests t sur toutes les paires d'IHM.

Technique 1	**Technique 2**	**T**	**p**
Téléphone	*VESAD tactile*	4,1	9×10^{-5}
VESAD	*Téléphone*	5,6	3×10^{-5}
VESAD	*VESAD tactile*	9,0	6×10^{-14}

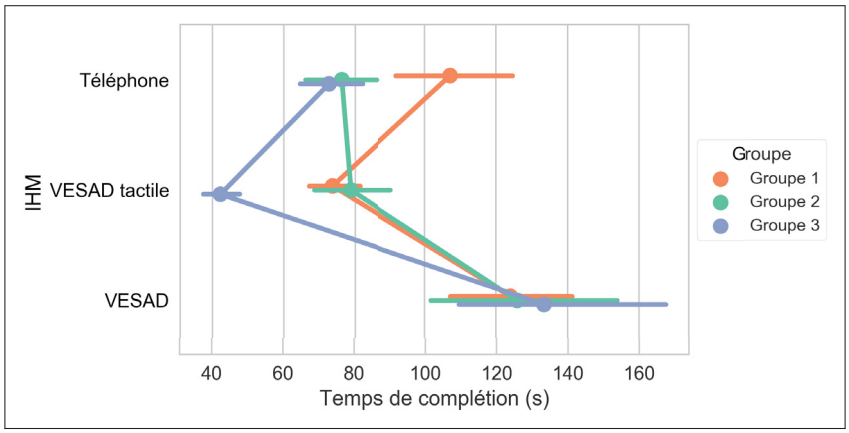


Figure 4.8 Temps de complétion moyens (moyennes géometriques) par essai pour chaque condition IHM $\times$ GROUPE.

Cependant, la Figure 4.8 indique que l'effet du GROUPE a été important seulement pour certaines conditions : les participants qui ont commencé avec le *Téléphone* (*Groupe 1*) ont été plus lents avec cette IHM, tandis que ceux qui ont terminé avec *VESAD tactile* (*Groupe 3*) ont été plus rapide sur cette IHM. Dans les autres conditions, il paraît ne pas avoir de différence entre les temps de complétion. Cela semble indiquer que, d'une part, il y a effectivement une courbe d'apprentissage avec cette tâche, en particulier quand les utilisateurs doivent reconstruire men-

4. On calcule les degrés de liberté (ddl) d'une variable indépendante ainsi : $ddl = conditions - 1$.
4. On calcule les ddl de l'erreur pour une variable indépendante ainsi : $ddl_{erreur} = ddl \times (observations - 1)$.

talement la grille, le *Téléphone* en affichant une vue non entière. D'autre part, le *VESAD tactile* semble être le plus rapide des trois IHMs quand la tâche est maîtrisée.

4.8.2 Erreurs et sélections

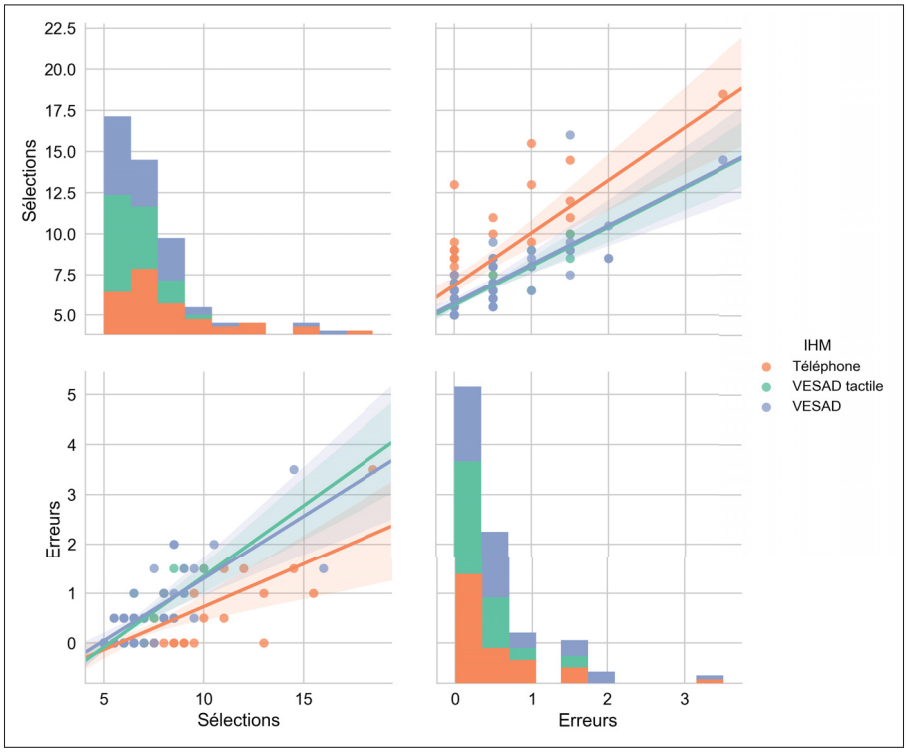
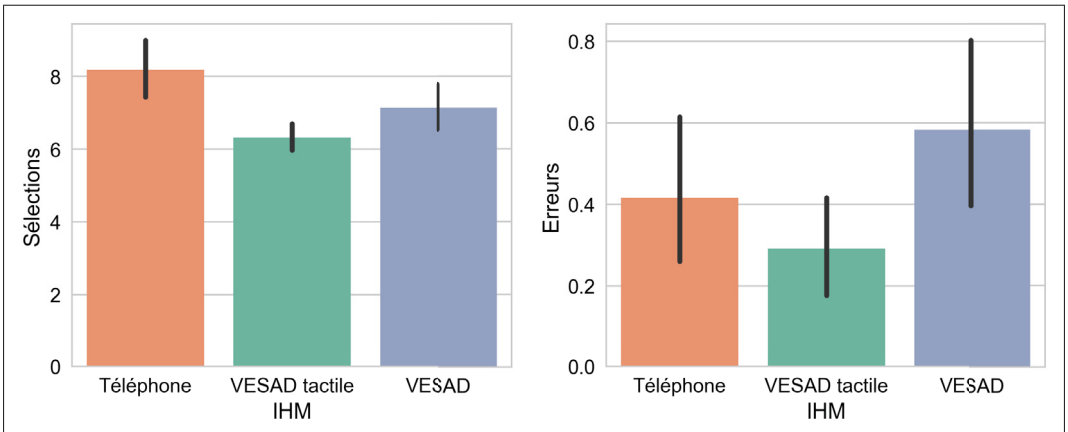


Figure 4.9 Histogrammes et nuages de points des erreurs et du nombre de sélections par IHM.

On visualise tout d'abord les distributions des erreurs et du nombre de sélections. La Figure 4.9 donne une bonne indication qu'il y a corrélation entre erreurs et sélections pour chaque IHM. Il semble également que les utilisateurs *Téléphone* font plus de sélections, à erreurs égales, que sur les autres IHMs.

Les distributions des deux variables ne suivant pas une loi normale, nous utilisons des tests non-paramétrique (Wobbrock & Kay, 2016). On utilise alors le test de Kruskal-Wallis (avec une correction de Benjamini-Hochberg) pour vérifier les effets des variables indépendantes.

Les résultats indiquent que seuls la IHM ($H = 15{,}1$; $p = 0{,}004$) et le GROUPE ($H = 11{,}5$; $p = 0{,}01$) ont un effet significatif sur le nombre de sélections, mais que seul le GROUPE ($H = 11{,}1$; $p = 0{,}01$) a un effet significatif sur les erreurs. La Figure 4.10b nous confirme que nous n'avons pas mesuré de différences importantes en termes d'erreurs pour la IHM.



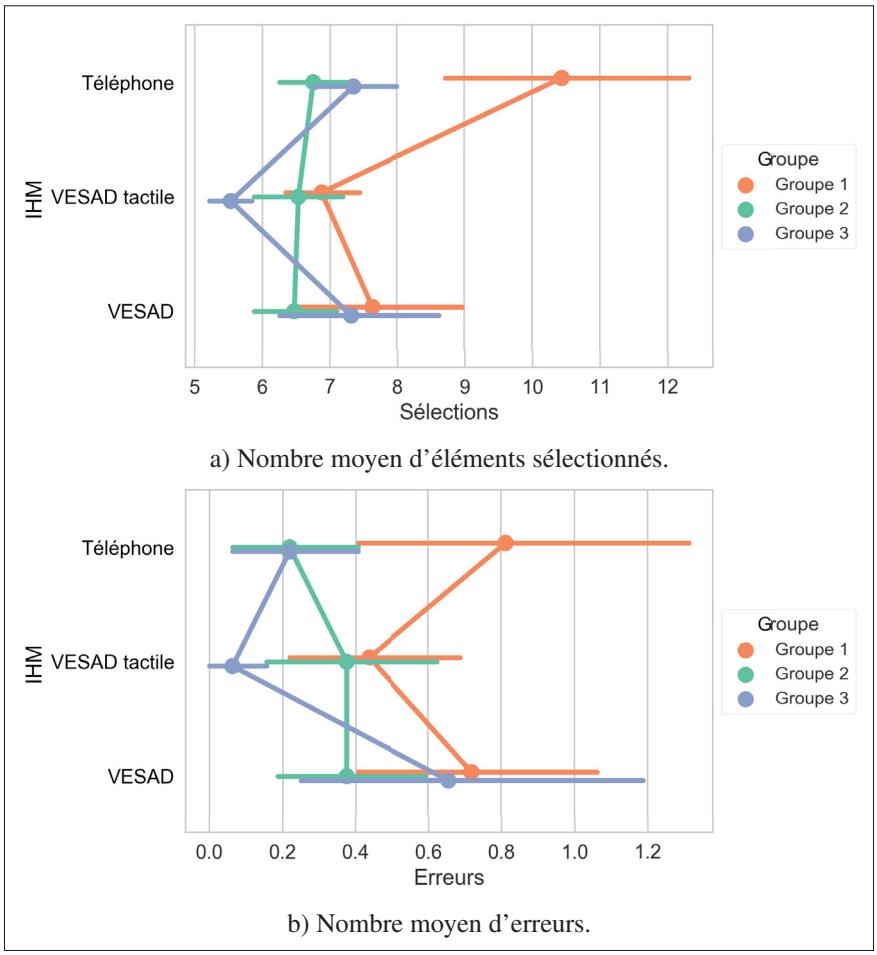
a) Nombre moyen d'éléments sélectionnés.

b) Nombre moyen d'erreurs (éléments qui ont été mal classés par le participant).

Figure 4.10 Sélections et erreurs moyennes par essai pour chaque IHM.

Pour mieux comprendre l'effet sur le nombre de sélection, on compare deux-à-deux les conditions IHM avec le test de Wilcoxon-Mann-Whitney (avec une correction de Benjamini-Hochberg). Les résultats indiquent qu'il y a des différences significatives entre pour les paires {*Téléphone*, *VESAD tactile*} ($p = 6 \times 10^{-4}$) et {*Téléphone*, *VESAD*} ($p = 0{,}03$), mais pas pour la paire {*VESAD tactile*, *VESAD*} ($p = 0{,}06$). La Figure 4.10a nous confirme que le *Téléphone* demande le plus de sélections, soit 8,18 [7,42 ; 9,06] en moyenne, et que le *VESAD tactile* en demande le moins, soit 6,31 [5,97 ; 6,66] en moyenne, mais cette différence est peu importante.

Comme pour le temps de complétion, il y une bonne indication que le GROUPE n'a un effet important que dans certaines conditions Figure 4.11 : le *Groupe 1* a commis un plus d'erreurs et effectué plus de sélections en commençant avec le *Téléphone*, tandis que le *Groupe 3* a fait un peu moins d'erreurs et sélections en terminant avec le *VESAD tactile*. Les autres conditions



a) Nombre moyen d'éléments sélectionnés.

b) Nombre moyen d'erreurs.

Figure 4.11 Sélections et erreurs moyennes par essai pour chaque condition IHM × GROUPE.

IHM × GROUPE ne paraissent pas importantes. Les participants dans la condition {*Groupe 1*, *Téléphone*}, n'ayant pas encore une bonne image mentale de la grille, pouvaient probablement parfois oublier quel disque était sélectionné ou changer d'avis sur le disque à classer. Enfin, le nombre un peu plus élevé d'erreurs pour *VESAD tactile* est probablement dû à des classements involontaires que nous avons parfois observés, le suivi de la main par le Leap Motion n'étant pas toujours bon.

4.8.3 Navigation et classements

Les comportements des utilisateurs durant les essais nous permettent de mieux comprendre ces différences entre les IHMs.

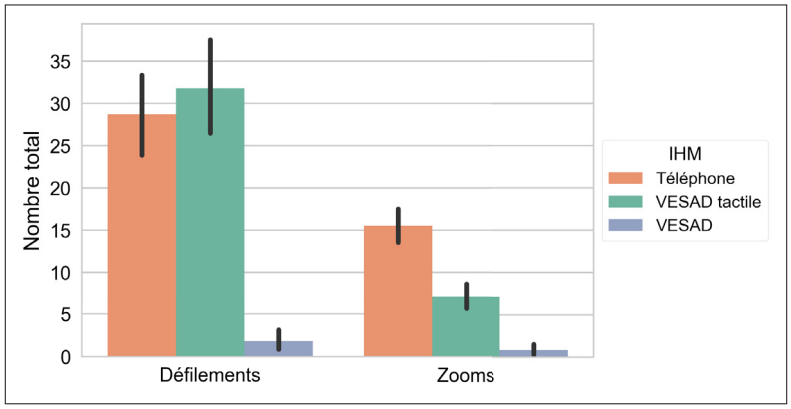


Figure 4.12 Nombre total moyen par essai de défilements et de zooms pour chaque IHM.

Tout d'abord, nous comparons navigations virtuelles (défilements et zooms) et physiques. La Figure 4.12 nous indique que la navigation virtuelle a été principalement utilisée avec le *Téléphone* et le *VESAD tactile* avec en moyenne plus de 25 défilements et cinq zooms par essai. Autrement dit, chaque disque a demandé en moyenne plus d'un défilement et au moins un zoom pour être classé. En revanche, les participants ont peu utilisé les techniques de défilement et de zoom virtuels avec le *VESAD*, mais privilégié une navigation physique (*Voir* Figure 4.13) ; nous avons observé des déplacements bi-manuels, ou des rotations de la grille, pour placer au même endroit la main virtuelle et un disque à sélectionner. Des participants ont, par ailleurs, fait la remarque que le suivi du téléphone était plus stable que celui de la main. La navigation physique reste importante pour les deux premières IHMs (*Voir* Figure 4.13) : nous avons fréquemment observé des mouvements de « rapides zooms physiques » dans toutes les conditions, où le participant rapprochait le téléphone de son visage, pour pouvoir lire une lettre d'un

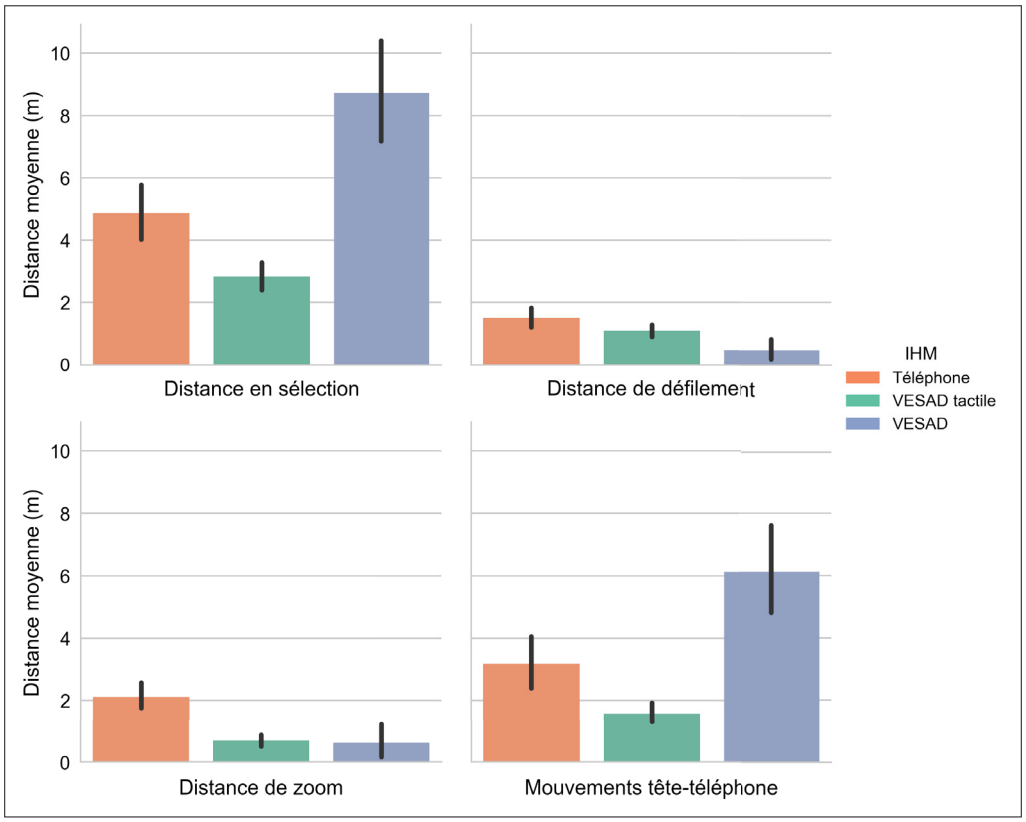


Figure 4.13 La distance parcourue par l'index (en haut à gauche) pendant qu'un disque était sélectionné, (en haut à droite) pendant les défilements, (en bas à gauche) pendant les zooms, ainsi que (en bas à droite) la distance des mouvements de la tête par rapport au téléphone. Toutes les mesures sont des moyennes par essai pour chaque IHM.

disque. Le *VESAD tactile* demande moins de ces mouvements, car le grand écran donnait la possibilité aux participants de laisser travailler avec une grille avec un plus fort zoom.

Ensuite, on peut mieux expliquer la meilleure performance du *VESAD tactile* par rapport *Téléphone*. Nous n'avons mesuré aucune différence importante en termes de nombre de défilements, ainsi que le temps passé (*Voir* Figure 4.14) et la distance parcourue pendant les défilements. Cependant, les participants ont utilisé seulement cinq zooms en moyenne par essai avec le *VESAD tactile* contre deux fois plus pour le *Téléphone*. De même, les durées et distances moyennes des zooms par essai (*Voir* Figure 4.14) sont toutes deux moitié moins importantes

avec le *VESAD tactile* qu'avec le *Téléphone*. Plus généralement, les participants passaient 30% moins de temps avec un disque sélectionné avec le *VESAD tactile* par rapport au *Téléphone* (*Voir* Figure 4.14).

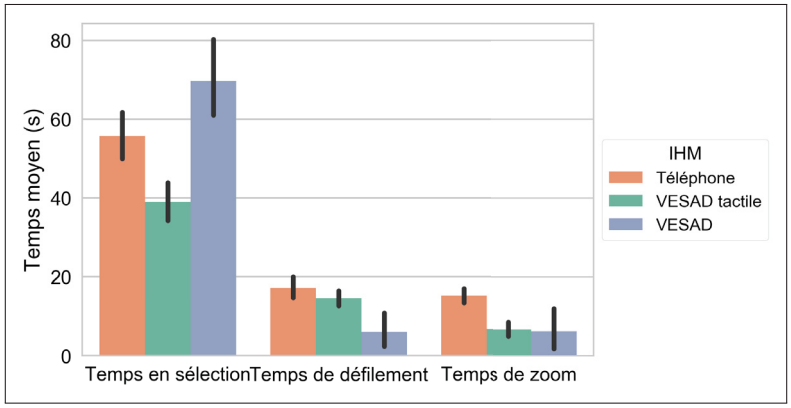


Figure 4.14 Le temps total moyen par essai passé (gauche) avec un disque sélectionné, (milieu) à défiler et (droite) à zoomer. Ces temps peuvent se supporposer.

On peut apporter enfin quelques explications supplémentaires des mauvaises performances du *VESAD*. Le temps passé avec un disque sélectionné est un peu plus important que celui du *Téléphone* : malgré le grand écran, nous avons observé des difficultés des participants pour pointer une cellule pour déplacer le disque. En effet, l'opération de zoom virtuel était jugée trop difficile (il y a très peu de zoom pour cette IHM, mais des durées moyennes longues) par les participants qui préféraient laisser la grille dans sa configuration initiale. Les disques et les cellules étaient alors petits et difficiles à pointer avec la main virtuelle. De plus, cette technique d'interaction demande beaucoup plus de mouvements que travailler avec l'écran tactile (*Voir* Figure 4.13). Par ailleurs, quand les disques à sélectionner se trouvaient du côté de la main tenant le téléphone (à gauche de la main gauche pour un droitier donc), les participants devaient croiser leurs mains, ce que la plupart d'entre eux ont trouvé cela particulièrement ardu et inconfortable.

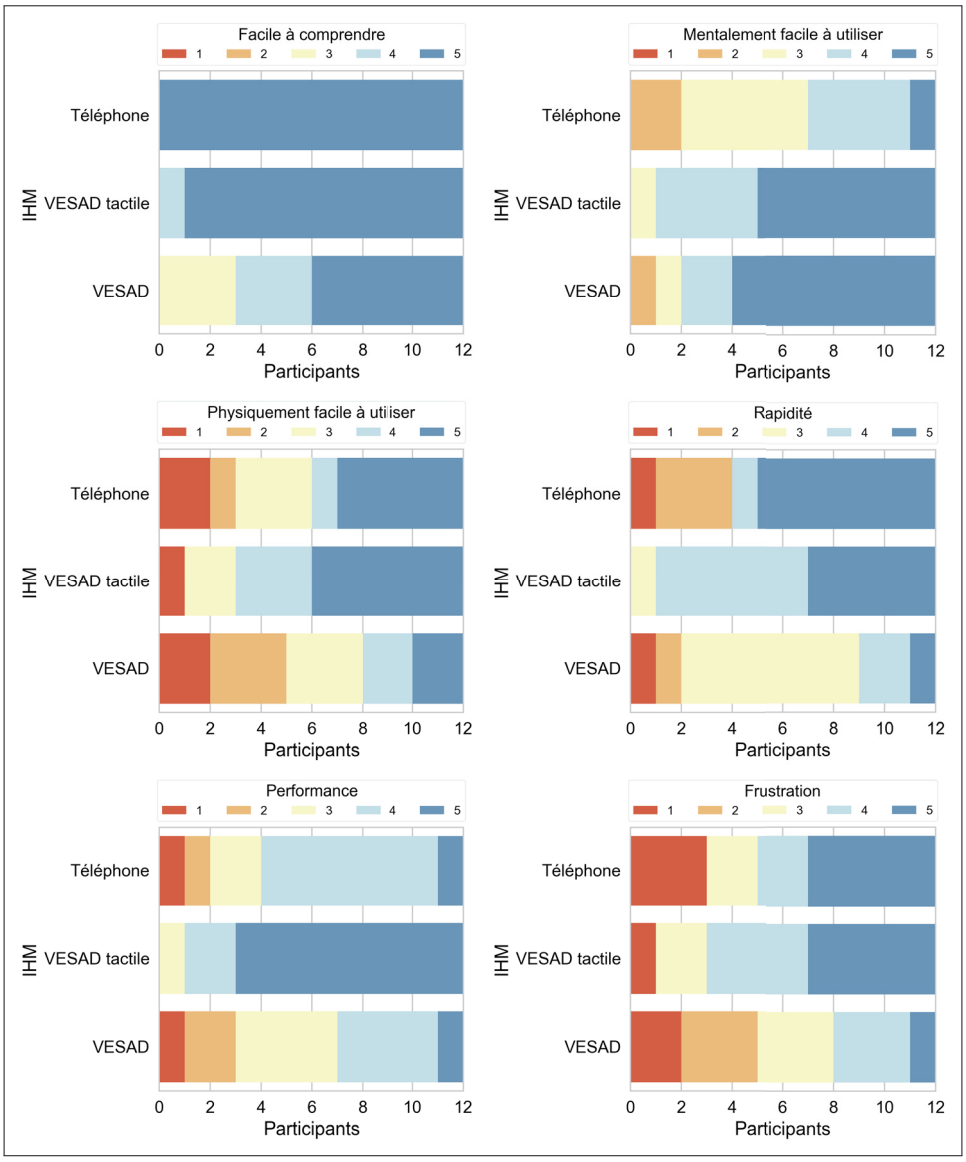
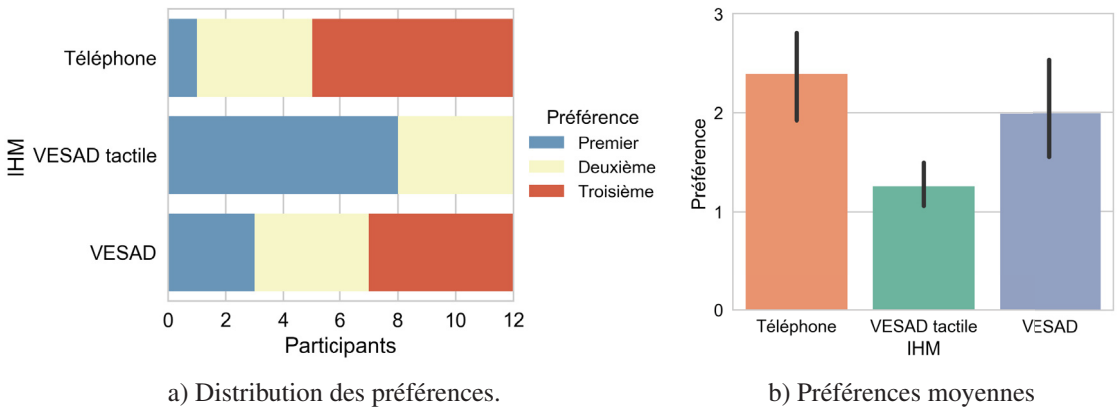


Figure 4.15 Distributions des notes données par les participants à chaque IHM. Une note élevée est meilleure (5 correspond à une faible frustration).

4.8.4 Évaluations des participants

Comme pour les erreurs, nous utilisons des tests non-paramétriques pour analyser les mesures subjectives du post-questionnaire, leurs distributions étant non-normales et le nombre Wobbrock & Kay (2016). La Figure 4.15 montre les distributions des notes et la Figure 4.16a celle des préférences. Comme pour le temps de complétion, on transforme d'abord les données avec



a) Distribution des préférences.

b) Préférences moyennes (moyennes géometriques), la meilleure note est 1 et la moins bonne 3.

Figure 4.16 Préférences des participants pour chaque IHM.

le logarithme népérien, pour y calculer moyennes avec IC à 95% que l'on reconvertit avec l'exponentielle.

On utilise alors le test de Kruskal-Wallis pour déterminer s'il y a des différences significatives aux réponses de chaque questions, dû à la IHM. Puis, on utilise un test de Wilcoxon-Mann-Whitney pour comparer les questions avec une différence significative. On applique une correction de Benjamini–Hochberg aux valeurs-p retournées par les deux tests. Les IC à 95% des moyennes des réponses aux notes (*Voir* Figure 4.17) et au classement (*Voir* Figure 4.16b) nous indiquent l'importance de l'effet. Les résultats obtenus sont les suivants :

- Facilité de compréhension ($p = 0{,}007$) : seul le *Téléphone* est significativement meilleur que le *VESAD* ($p = 0{,}01$), mais il semble que le *VESAD tactile* soit un peu moins bon aussi. Cela n'est guère étonnant, car le *Téléphone* est connu et maîtrisé des utilisateurs. Les différences ne sont tout de même pas trop importantes
- Mentalement exigeant ($p = 0{,}007$) : le *Téléphone* est significativement pire (-26%) que le *VESAD tactile* ($p = 0{,}005$) et le *VESAD* ($p = 0{,}01$). Cela donne une bonne preuve que le petit écran rend la tâche plus difficile.

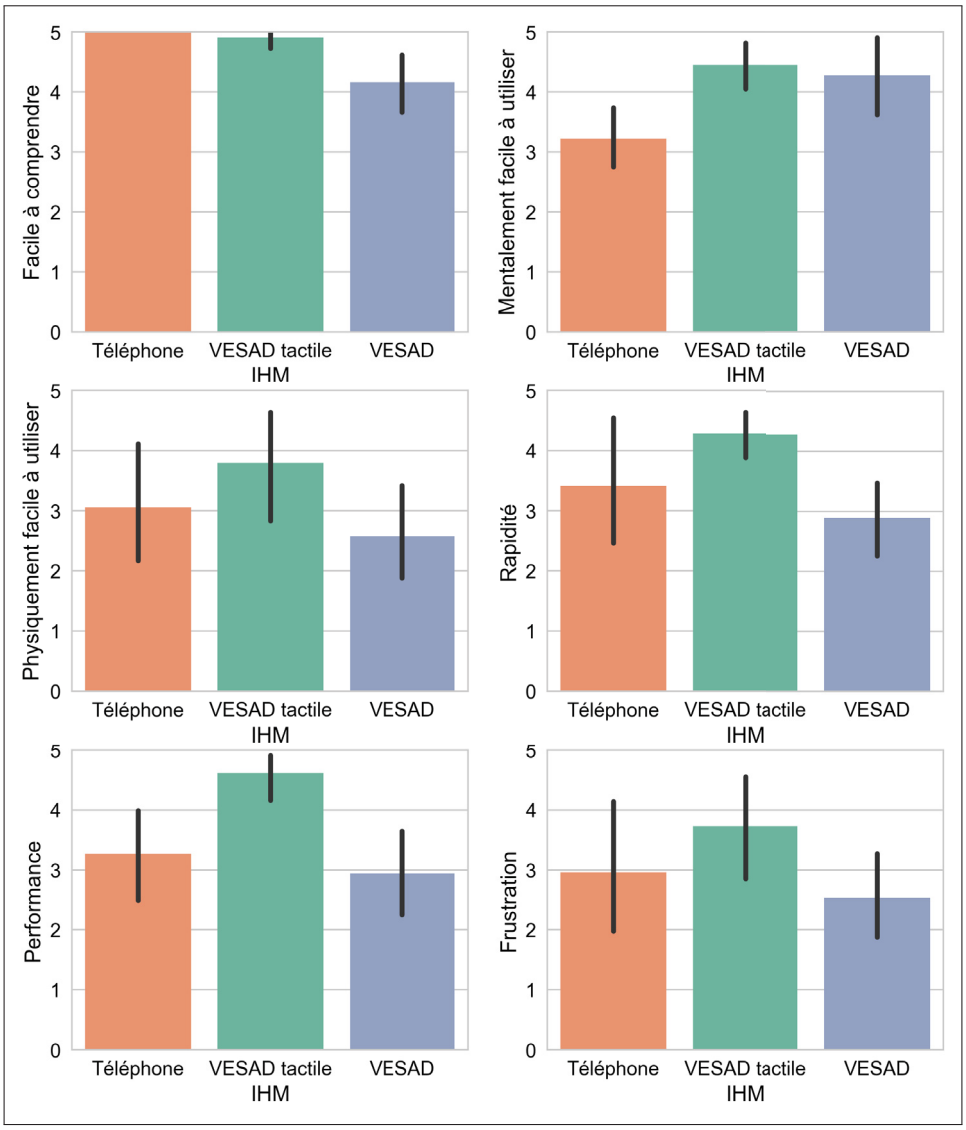


Figure 4.17 Notes moyennes (moyennes géometriques) des participants à chaque IHM (une note élevée est meilleure).

- Physiquement exigeant : aucune différence significative ($p = 0{,}1$). Les trois IHMs ont des valeurs assez basses ($\approx$3), ce qui semble souligner la difficulté de la tâche. Cependant, la distribution des notes indique que le *VESAD* a été le plus difficile (*Voir* Figure 4.15), probablement car pointer avec la main en 3D est plus dur et lent que sur une surface 2D (Argelaguet & Andujar, 2013). Quant au *Téléphone*, plusieurs participants se sont plaint de faire beaucoup de défilements et de zooms ; certains ont trouvé pénible de devoir amener

une cible sur l'écran tactile plutôt que pouvoir la sélectionner directement avec le *VESAD tactile*.

- Rapidité ($p = 0{,}04$) : le *VESAD tactile* a été jugé significativement plus rapide que le *VESAD* ($p = 0{,}005$) avec un effet qui semble important (+40%). Nous cependant n'avons pas pu mesurer de différence entre le *Téléphone* et les deux autres IHMs, les notes étant très variées (*Voir* Figure 4.15).
- Performance ($p = 0{,}006$) : le *VESAD tactile* est significativement, et de manière importante, plus rapide que le *Téléphone* ($p = 0{,}005$, +25%) et *VESAD* ($p = 0{,}01$, +47%). Nous n'avons cependant mesuré aucune différence entre ces deux dernières IHMs.
- Frustration : aucune différence significative ($p = 0{,}1$), même si les distributions semblent pointer que le *VESAD* l'était un peu plus, probablement dû au mauvais suivi de la main.
- Préférences ($p = 0{,}006$) : le *VESAD tactile* est significativement, et largement, préféré au *VESAD* ($p = 0{,}01$) et au *Téléphone* ($p = 0{,}004$). Il n'y a pas de différence significative entre ces deux dernières IHMs, même si le *VESAD* semble un peu plus préféré ; avec un suivi de la main fiable, plusieurs participants nous ont dit qu'ils le voyaient le plus prometteur.

Pour finir, tous les participants ont trouvé intéressant et utile l'idée du multi-tâche (multiples applications, ou application multi-fenêtres) sur un téléphone à écran étendu (*Voir* Figure 2.2a) et la majorité d'entre eux pour la carte de navigation (*Voir* Figure 2.3a).

CHAPITRE 5

DISCUSSION

5.1 Discussion des résultats

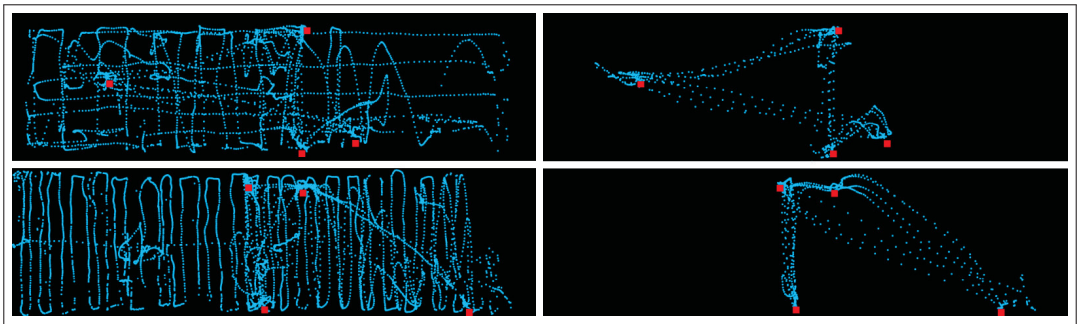
Dans l'ensemble, nos résultats indiquent que le *VESAD tactile* a été la meilleure des trois IHMs. Elle a tout d'abord été la plus rapide, avec le plus petit temps de complétion ($\approx 62\,s$), en particulier quand la tâche était maîtrisée des participants ($\approx 42\,s$). Elle a ensuite été largement préférée des participants, et jugé la plus performante. Enfin, elle a été la plus performante dans le classement : d'une part, car les participants utilisaient en moyenne une seule sélection et un seul zoom par disque à classer et, d'autre part, car le temps passé avec un disque sélectionné était beaucoup plus court qu'avec les autres IHMs. Ce n'est guère étonnant, le *VESAD tactile* bénéficiant à la fois d'un grand affichage et d'interactions stables et précises sur l'écran tactile. Malgré tout, notre première hypothèse est non supportée par nos résultats : le *Téléphone* (non étendu) n'a pas été le moins performant des trois IHMs testées.

Figure 5.1 Démonstration de la montre à écran étendu (condition SWRef) de Grubert *et al.* (2015), similaire à notre technique *VESAD tactile*.
Adapté de Grubert (2015), à 2 min 4 s.

On peut tout de même comparer le *VESAD tactile* et le *Téléphone*, car les techniques d'interactions utilisées sont les mêmes, seule change la taille de l'écran. Nous avons en effet observé que

les participants s'habituaient très vite à fusionner l'affichage sur l'écran tactile avec l'affichage virtuel autour du téléphone. La différence importante de temps de complétion entre ces deux IHMs va donc dans le sens des résultats de Rädle *et al.* (2014), qui avaient mis en évidence qu'une tâche de navigation dans un grand document devenait difficile avec un écran plus petit que celui d'une tablette (23,5 cm × 13,2 cm). En revanche, cette différence contraste avec les résultats de Grubert *et al.* (2015), qui avaient mesuré, dans une tâche de navigation, qu'une montre intelligente à l'écran étendu performait moins qu'un téléphone seul. Cependant, leur implémentation de l'écran étendu jouait en sa défaveur : le plan virtuel était bien aligné mais vu comme 2,5 m en arrière de la montre, en plus d'un champ de vision limité du visiocasque (30,5° × 17,2°), d'un écart de résolution entre l'écran et le visiocasque et d'une latence importante entre les deux affichages (*Voir* Figure 5.1). Contrairement à Grubert *et al.*, le *VESAD tactile* a surpassé le *Téléphone* dans notre étude. Il a été 30% plus rapide que le *Téléphone*, avec deux fois moins de zooms. En outre, les participants ont également jugé le *Téléphone* mentalement plus exigeant que les deux autres techniques. Il est intéressant de noter qu'il n'y a eu aucune différence entre ces deux IHMs en termes de défilements. Ainsi, il nous semble qu'*un VESAD bien conçu et implémenté est plus performant que l'écran seul*.

En outre, nous avons observé des tactiques similaires de sélection des disques entre le le *VESAD tactile* et le *Téléphone*. Une première tactique consiste à zoomer suffisamment la grille pour pouvoir lire le texte sur les disques et à parcourir toutes les cellules de la grille jusqu'à trouver celle correspondant au disque sélectionné. Rädle *et al.* (2014) avaient également remarqué ces mouvements de « *scanning* » (*Voir* Figure 5.2a) avec les tailles d'écrans correspondant à la tablette et au téléphone, quand les participants découvraient le document ; ces similitudes entre nos défilements statiques et leurs défilements physiques et dynamiques sont intéressantes. La seconde tactique est similaire a celle qui avait été décrite par Guiard & Beaudouin-Lafon (2004) pour les IHMs Pan+Zoom sur ordinateurs : (i) un dé-zoom pour repérer une cible ou la placer dans la vue, (ii) suivi de zooms et défilement pour la rendre assez grosse pour finale-



a) Mouvements de « *scanning* » lors des premiers essais (exploration du document).

b) Mouvements lors des derniers essais (le document est maintenant connu et mémorisé par le participant).

Figure 5.2 Mouvements (en bleu) face à l'affichage mural d'un participant avec l'écran de la taille (en haut) d'une tablette ou (en bas) d'un téléphone. Les points rouges sont les cibles à atteindre.
Tiré de Rädle *et al.* (2014).

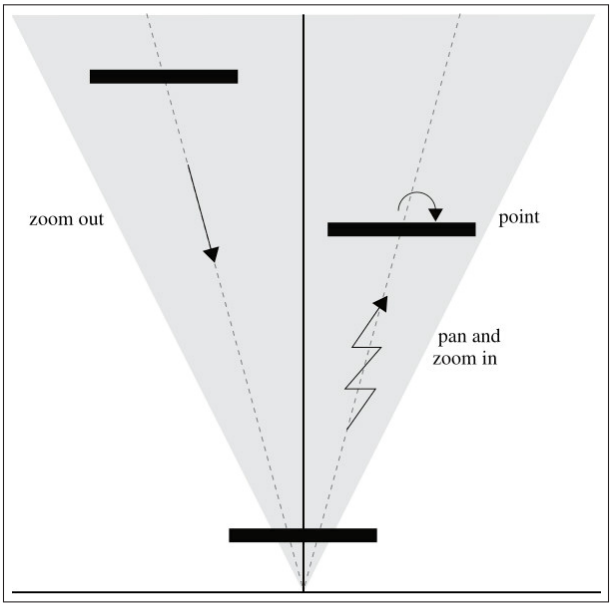


Figure 5.3 Séquence d'actions typique pour atteindre une cible avec une IHM Pan+Zoom sur un ordinateur : l'utilisateur dézoome pour la repérer, puis zoome et défile vers cette cible pour pouvoir la pointer.
Tiré de Guiard & Beaudouin-Lafon (2004).

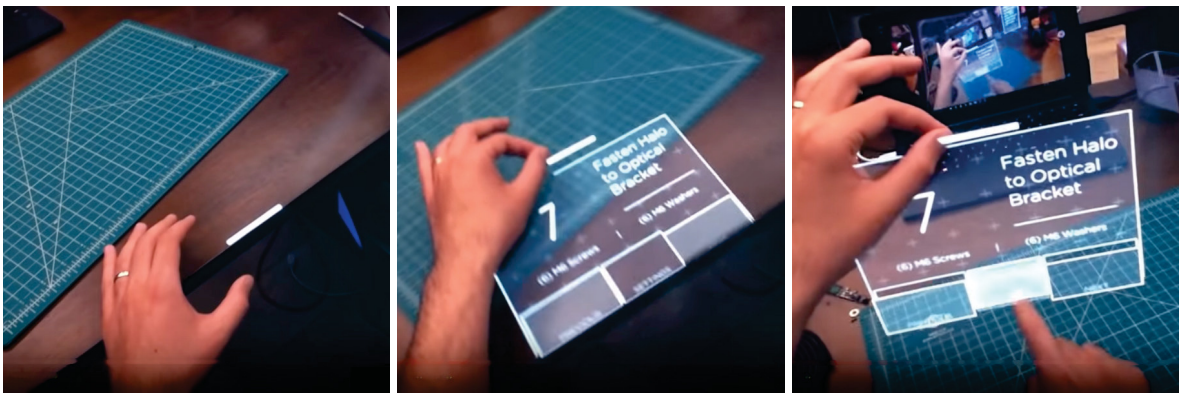
ment (iii) la sélectionner (*Voir* Figure 5.3). Les participants de notre expérience effectuaient ce mouvement pour lire le texte sur les disques. Cependant, Guiard & Beaudouin-Lafon (2004)

indiquaient à juste titre que la première phase n'était pas nécessaire si la difficulté était assez faible, ce que nous avons vu être souvent le cas pour le *VESAD tactile*. Certains participants profitaient en effet du grand écran et de sa résolution suffisante pour le laisser zoomé, n'effectuant alors que des défilements. Cela est en accord avec les mouvements observés par Rädle *et al.* (2014) quand les participants connaissaient le document (quand ils en avaient une carte mentale) (*Voir* Figure 5.2b).

Pourtant, plusieurs participants ont expliqué que le *VESAD* leur semblait l'IHM avec le plus de potentiel. Nous les avons souvent vu tenter spontanément de toucher la grille la première fois qu'il ont vu l'écran étendu. De plus, ils se sont jugés aussi performants qu'avec le *Téléphone* et l'ont un peu plus préféré que ce dernier. Ils l'ont aussi trouvé moins exigeant mentalement que le *Téléphone*, ce qui indique que l'écran étendu donnait un avantage. Quelques participants ont aussi fait remarquer qu'ils préféraient toucher directement une cible sur l'écran virtuel plutôt que devoir la déplacer sur l'écran tactile avec le *VESAD tactile*. Ajouter à cela la meilleure performance du *VESAD tactile*, notre deuxième hypothèse est donc supportée : *les participants ont préféré l'écran étendu au téléphone seul*.

Malgré tout, le *VESAD* a été le moins performant. Cela est principalement dû au suivi de la main avec le Leap Motion qui manquait de stabilité et de précision. Par ailleurs, nos gestes de défilement et de zoom n'étaient pas bien conçus. Il s'est en effet avéré difficile de faire un déplacement précis de la main le long d'un plan virtuel en 3D. Nous l'avions pourtant vu dans notre étude pilote, mais nos corrections n'ont malheureusement pas été suffisantes. Ces deux facteurs jouant ainsi en sa défaveur, la comparaison avec les deux autres IHMs est donc difficile. Pourtant, la littérature a montré que la sélection d'une cible avec une main virtuelle était plus difficile que sur une surface tangible (Cha *et al.*, 2010; Jones *et al.*, 2012; Argelaguet & Andujar, 2013). Les mauvaises notes sur l'exigence physique, la frustration ainsi que les longs temps passés avec un item sélectionné semblent aller dans ce sens. On peut donc

tout de même estimer que nos notre troisième hypothèse soit supportée : avec un téléphone à écran étendu, *les interactions sur l'écran tactile ont été plus performantes qu'avec une main virtuelle*.



a) La zone au-dessus d'une fenêtre virtuelle permet de la manipuler.

b) La manipulation est activée par un pincement sur le haut de la fenêtre virtuelle.

c) Le reste de la fenêtre réagit seulement aux sélections par pointage d'un doigt.

Figure 5.4 Démontration de l'IHM du visiocasque de RA North Star. L'interface est transparente et la main virtuelle fait une excellente occlusion avec le contenu 3D.
Adapté de Leap Motion (2018).

Enfin, de nos observations et des remarques des participants, nous voyons quelques pistes pour améliorer les gestes pour la main virtuelle. Nous avons omis une caractéristique essentielle de cette technique d'interaction : contrairement à une souris, ou un écran tactile, *le suivi de la main est réalisé en continu et sans retours physiques*, en plus d'être en 6 DoFs. Si on ne se sert que de cette information, cela devient très contraignant pour l'utilisateur qui ne peut pas *donner son intention* de vouloir interagir ou non avec l'objet qu'il « touche ». Ainsi, nous avons plusieurs fois observé des participants ne pas comprendre pourquoi ils avaient effectué une sélection ou une manipulation de la grille ou encore secouer la main comme pour enlever la sphère blanche. Comme en appuyant sur un bouton, il est nécessaire que l'utilisateur confirme une commande par une *action discrète*. Un objet physique, incluant écran tactile ou souris, est soit touché ou non, il n'y a pas d'intermédiaires. Leap Motion (2018) a récemment présenté son visiocasque

de RA, utilisant une main virtuelle : si leur travail n'est pas une publication scientifique il reste néanmoins intéressant, car il montre des experts concevant et utilisant une IHM de RA. S'ils proposent tout de même des sélections avec un simple pointage du doigt (geste H11 de Piumsomboon *et al.* (2013), Figure 5.4c), mais avec beaucoup de retours visuels, les manipulations des fenêtres virtuelles s'activent avec un pincement (geste H2 de Piumsomboon *et al.* (2013), Figure 5.4b). Ce geste de pincement est également utilisé par le HoloLens pour faire une sélection. Appliqué à notre VESAD, ce geste permettrait alors d'effectuer les défilements en pinçant la grille : ainsi, elle suivra les mouvements de la main (sous contrainte de rester sur le même plan) jusqu'au relâchement du pouce et de l'index. Le zoom pourrait alors se faire simplement par la combinaison d'un appui long sur l'écran tactile et d'un geste de pincement sur la grille. Il est également probable que le zoom soit à privilégier sur l'écran tactile.

Toutefois, notre expérience comporte plusieurs limites. Premièrement, nous avons mal contrôlé la difficulté de notre tâche. Nous n'avons en effet pas pu mesurer de différence entre les conditions de TAILLE et la DISTANCE, contrairement à Liu *et al.* (2014). Cela limite nos conclusions, car il aurait été intéressant de voir l'effet de ces quatre différents « niveaux » de difficulté sur la performance des trois IHMs. Nous avions dans notre étude pilote prévu 15 disques à classer au lieu de cinq et des tailles de texte beaucoup plus petites, qui s'étaient révélées impraticables avec le *VESAD*. Nous avons alors placé trop proches nos deux conditions de TAILLE. En outre, nous avons directement utilisé les valeurs DISTANCE de la tâche de Liu *et al.*. Il aurait été plus sage de faire une ou deux itérations supplémentaires pour tester la validité de la difficulté. Deuxièmement, nous l'avons décrit, le mauvais suivi de la main des participants et la mauvaise conception des techniques de navigation avec la main virtuelle ont défavorisé le *VESAD*, qui a été l'IHM la moins performante des trois évaluées.

Troisièmement, nous avons malheureusement du faire un compromis sur la densité visuelle de notre visiocasque pour avoir un large champ de vision. Si cela a désavantagé le *Téléphone*,

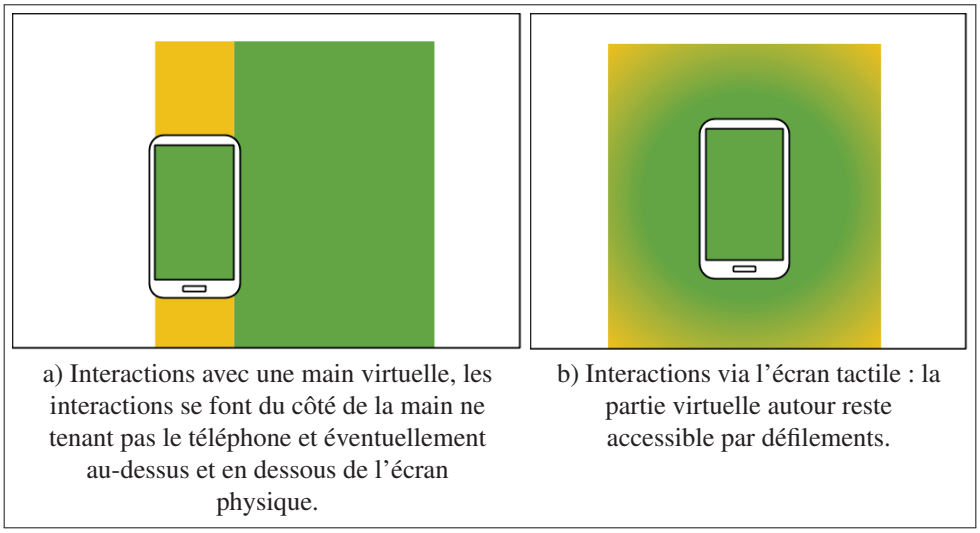
nous avons cependant maintenu constants la latence, la densité visuelle, le champ de vision et le poids sur la tête des participants en faisant utiliser le visiocasque pour chaque IHM, contrairement à Grubert *et al.* (2015). Ces variables parasites contrôlées, nos comparaisons entre le *Téléphone* et les deux autres IHMs ont alors une meilleure validité.

De manière plus générale, on peut voir l'utilisation nécessaire d'un visiocasque comme une limite. Nous avons en effet utilisé un visiocasque de RA vidéo, impliquant une vision dégradée (latence et densité visuelle) de l'environnement réel. Toutefois, grâce aux développements récents et importants dans l'industrie, nous anticipons un usage répandu à l'avenir de visiocasques de RA optiques plus légers, performants et avec large champ de vision. Ce type de visiocasque a l'avantage de ne pas dégrader la vision de l'environnement réel. Il est possible qu'un meilleur visiocasque change nos résultats, mais il nous semble que cette étude expérimentale supporte que *certaines tâches bénéficieraient d'un écran étendu*.

5.2 Conception d'une IHM pour un VESAD

Avant la conception d'une IHM, nous pensons crucial que le VESAD soit d'abord bien implémenté, c'est-à-dire avoir : (1) un excellent suivi avec 6 DoFs de l'écran physique à étendre, (2) un bon alignement entre l'écran virtuel et l'écran physique et (3) une synchronisation sans latence entre les deux écrans pour donner l'illusion d'un seul écran étendu. Notre bibliothèque de RA ArucoUnity répond à ces deux premières problématiques. Plusieurs participants nous ont cependant indiqué d'améliorer le troisième point. En outre, il nous semble important d'utiliser un visiocasque avec un grand champ de vision pour voir complètement l'écran étendu. Enfin, un visiocasque avec une bonne densité visuelle est préférable.

Pendant la conception de l'IHM pour un écran étendu, il est essentiel de *penser l'interface à travers la technique d'interaction* utilisée. En effet, comme l'ont très bien souligné Ens *et al.* (2014b), le temps et les erreurs de pointage augmentent pour les cibles plus lointaines en uti-



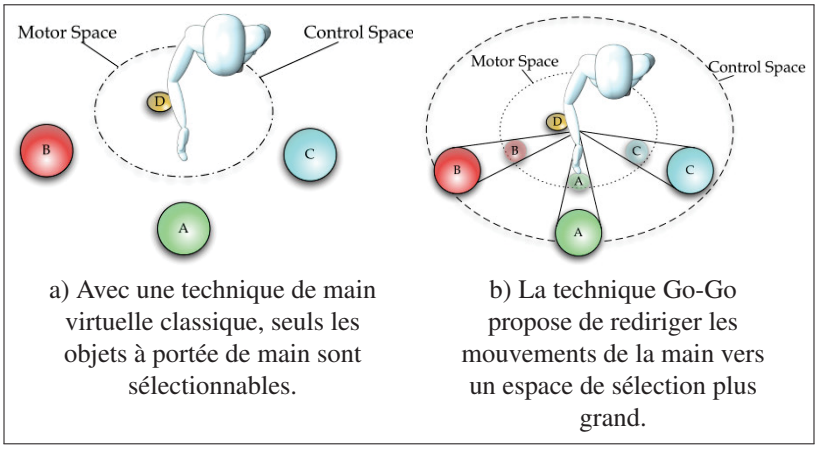
a) Interactions avec une main virtuelle, les interactions se font du côté de la main ne tenant pas le téléphone et éventuellement au-dessus et en dessous de l'écran physique.

b) Interactions via l'écran tactile : la partie virtuelle autour reste accessible par défilements.

Figure 5.5 Une IHM pour un téléphone à écran étendu contient des zones d'interactions (en vert), des zones pour les interactions moins fréquentes (en jaune) et des zones uniquement dédiée à la visualisation (en blanc).

lisant une main virtuelle. Cela a également été le cas avec le *VESAD tactile* : les participants devaient effectuer beaucoup de défilements et zooms pour amener une cible sur l'écran. Sur un écran tactile seul ou avec une souris sur un écran de PC, l'espace de visualisation est facilement accessible, quand il dépasse facilement celui d'interaction avec un VESAD. Des techniques comme Go-Go permettent d'atténuer ce problème pour la main virtuelle en agrandissant l'espace d'interaction accessible par un gain sur les mouvements de la main (*Voir* Figure 5.6). Ens *et al.* suggèrent de positionner des fenêtres virtuelles à équidistance de l'utilisateur dans une disposition sphérique (*Voir* Figure 5.7). Cette idée serait intéressante à explorer pour un VESAD : nous avons observé plusieurs participants tourner le téléphone pour rapprocher une cible. Une autre approche est d'utiliser des pointeurs virtuels (Argelaguet & Andujar, 2013).

Ainsi, nous recommandons de *favoriser les interactions seulement dans des zones facilement accessibles*, que nous résumons avec la Figure 5.5 : les zones blanches sont à réserver seulement pour l'affichage, les vertes sont les plus facilement accessibles tandis que les jaunes, plus difficiles d'accès, sont à réserver pour les interactions peu fréquentes. Ainsi, avec une main vir-



a) Avec une technique de main virtuelle classique, seuls les objets à portée de main sont sélectionnables.

b) La technique Go-Go propose de rediriger les mouvements de la main vers un espace de sélection plus grand.

Figure 5.6 Illustration de la technique Go-Go par rapport à une technique de main virtuelle.
Adapté de Argelaguet & Andujar (2013).

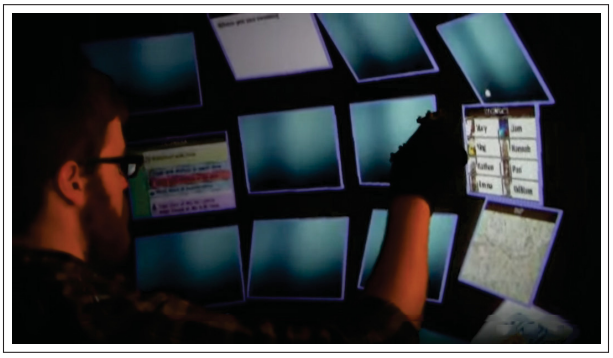


Figure 5.7 Démontration du Personal Cockpit : un ensemble de fenêtres 2D positionnées selon une sphère autour de l'utilisateur.
Adapté de Ens (2014).

tuelle, l'interface est plutôt asymétrique (*Voir* Figure 5.5a) : elle est alignée avec le téléphone qui est tenu avec une main tandis que les interactions se font avec l'autre main ; dès lors, quand le téléphone est tenu de la main gauche, l'espace d'interaction se trouve sur sa droite (et inversement). En outre, nous déconseillons les interactions du côté de la main tenant le téléphone (à gauche de la main gauche pour un droitier) qui demandent de croiser les mains. Nous améliorons alors notre basculement d'affichage wrist (*Voir* Figure 2.2), en plaçant les applications sélectionnables à droite du téléphone, à portée de la main (*Voir* Figure 5.8). À l'inverse, quand

l'écran tactile est utilisé, par exemple quand le téléphone est tenu à deux mains ou si un utilisateur préfère utiliser une seule main, l'interface est symétrique (*Voir* Figure 5.5b) : l'écran tactile est bien sûr accessible, ainsi que l'espace immédiatement autour, via des défilements.

Figure 5.8 Remaniement de wrist (*Voir* Figure 2.3a) : les applications sur lesquelles basculer sont placées à droite du téléphone, facilement sélectionnables. Les notifications sont placées au-dessus de l'écrans, car elles sont peu accédées.

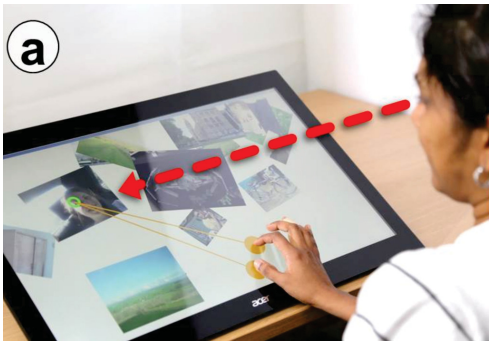
Enfin, tout comme la disposition d'un site web doit s'adapter (*responsiveness*) à la taille d'écran des différents appareils qui peuvent y accéder (par exemple : téléphone intelligent, tablette, ordinateur), *une IHM pour un VESAD doit s'adapter à différents scénarios d'interaction*. Ainsi, la navigation physique avec une main virtuelle et la navigation virtuelle via l'écran tactile doivent toutes deux être supportées, au choix de l'utilisateur. De plus, la disposition du contenu doit s'adapter dynamiquement au contexte d'utilisation (comment est tenu le téléphone) et au type d'interaction utilisé.

5.3 Directions futures

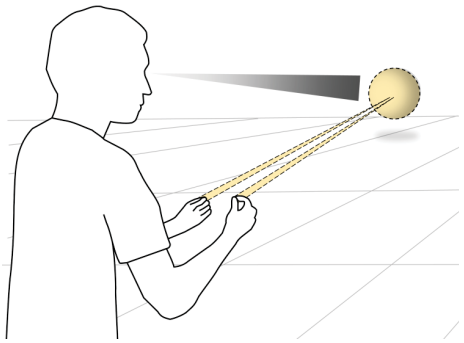
En premier lieu, la technique de main virtuelle de la condition *VESAD* devrait être améliorée. L'implémentation de meilleures techniques de suivi de pose de la main serait à explorer, par exemple en utilisant une caméra de profondeur, plus adaptée à la résolution de ce problème qu'une caméra RVB classique (Taylor *et al.*, 2016). Plus simplement, d'autres placements du Leap Motion pourrait améliorer la détection : par simplicité, nous avons placé ce petit boitier sous la caméra, mais il est probable que son angle de vue ne soit pas le meilleur ainsi. Le placer au-dessus du casque, légèrement incliné vers le sol pourrait améliorer le suivi.

Pour éviter de trop nombreuses comparaisons dans notre expérience, nous n'avons pas évalué des techniques présentes dans la littérature et l'industrie. Le HoloLens utilise un pointeur virtuel suivant les mouvements de la tête, les sélections se faisant avec un geste de pincement (geste H2 de Piumsomboon *et al.* (2013)) face au casque. Piumsomboon *et al.* (2014) avaient trouvé que les commandes vocales étaient plus efficaces qu'utiliser une main virtuelle pour changer l'échelle d'objets en 3D. Il serait alors intéressant de reproduire notre étude expérimentale pour les comparer à d'autres techniques d'interactions, en incluant également le geste de pincement virtuel que nous venons de proposer pour la main virtuelle. Une première variante du *VESAD tactile* utiliserait l'écran physique comme pavé tactile déplaçant un curseur sur tout l'écran étendu. Une seconde variante consisterait à contrôler un pointeur avec le regard et y rediriger les actions faites l'écran tactile (*Voir* Figure 5.9a) : cet agrandissement de l'espace d'interaction tout en gardant les entrées utilisateurs sur l'écran tactile permettrait de rapidement et facilement accéder à des éléments distants. Cette dernière pourrait aussi s'appliquer au pointeur virtuel du HoloLens (*Voir* Figure 5.9b).

De même, d'autres types de tâches devraient être évalués, comme un scénario de multi-tâche, similaire à Ens *et al.* (2014b), plus proche des usages quotidiens avec un téléphone intelligent. La faisabilité des deux nouvelles techniques d'interactions que nous avons proposée devrait



a) Gaze+Touch : les interactions sur l'écran tactile sont redirigées vers la cible du regard.



b) Gaze+Pinch : un geste de pincement sélectionne la cible (en 3D) du regard.

Figure 5.9 Interactions avec un pointeur virtuel suivant le regard. Adapté de a) Pfeuffer *et al.* (2014) et b) Pfeuffer *et al.* (2017).

aussi être évaluées, notamment wrist, dans une tâche de commutation d'applications. Slide-to-hang pourrait être implémenté, en mode multi-fenêtres dans le VESAD, avec un geste de pincement (*Voir* Figure 5.4b) pour manipuler les fenêtres.

Finalement, nous pensons que notre concept de téléphone à écran étendu mériterait d'être exploré. En particulier, nous nous demandons :

- Quelle taille devrait faire l'écran étendu ? Dans notre expérience, nous l'avons déterminée arbitrairement ; s'il n'y a virtuellement pas de limites, il est probable que certaines tailles d'écrans soient plus optimales en fonction de la tâche et du contexte d'utilisation.
- Explorer les changements d'usage et de contexte, notamment les différents basculements, automatiques et manuels, entre la main gauche, droite ou les deux tenant le téléphone.
- Concevoir et évaluer des applications, comme celles que nous avons présentées au chapitre 2 : par exemple, carte de navigation utilisant tout l'écran étendu, navigateur web ou application de messagerie fractionné en plusieurs onglets ou encore galerie d'images.
- Expérimenter des applications pour l'industrie : par exemple pour aider un employé d'un entrepôt à trouver un produit, un superviseur sur un site de construction ou un technicien sur

une machine à réparer. Toutes ces professions pourraient bénéficier d'un écran large tenant dans la main, par exemple pour afficher une vue d'ensemble de leur lieu de travail, tout en utilisant des interactions tactiles précises, stables et connues. La technique slide-to-hang permettrait de basculer vers un usage main libre au besoin.

CONCLUSION

Nous avons exploré, dans ce travail de recherche, l'extension de l'écran d'un téléphone intelligent par réalité augmentée, créant la perception d'un unique grand écran étendu tenu en main. Nous avons appelé cette extension : VESAD, pour *Virtuality Extended Screen-Aligned Display*. Plus précisément, nous souhaitions évaluer les avantages à utiliser un téléphone à écran étendu par rapport à un téléphone seul. Nous voulions également comparer différentes techniques d'interactions : soit en utilisant l'écran tactile, soit en manipulant la partie virtuelle de l'écran avec la main (technique de main virtuelle).

Pour répondre à ces questions, nous avons exploré la conception de cette interface humain-machine (IHM). Puis, nous avons développé notre visiocasque de RA avec un large champ de vision pour visualiser l'écran étendu au complet. Enfin, nous avons mené une étude expérimentale pour répondre à ces questions en comparant ces différentes techniques d'interactions sur le VESAD face à un téléphone seul. Nous posions les hypothèses que les participants auraient été plus performants avec l'écran étendu via les interactions tactiles, mais qu'ils auraient préféré utiliser la technique de main virtuelle (probablement perçue comme plus naturelle).

Nous avons tout d'abord motivé cette problématique par une revue de la littérature (*Voir* chapitre 1) où nous avons noté qu'il y avait un besoin de recherches sur les IHMs en RA (Billinghurst *et al.*, 2005, 2015), prioritairement via des études expérimentales. En particulier, les techniques d'interactions les plus adaptées pour la RA ne sont pas encore déterminées (Argelaguet & Andujar, 2013; Piumsomboon *et al.*, 2013, 2014), ni la forme des interfaces en RA (Van Dam, 1997; Ens *et al.*, 2014a; Serrano *et al.*, 2015a). Plusieurs études ont cependant pointé l'intérêt à combiner des visiocasques de RA avec des appareils que nous utilisons professionnellement (Grubert *et al.*, 2015; Serrano *et al.*, 2015b). Nous nous sommes alors interrogés si les résultats des études sur l'agrandissement des écrans (Baudisch *et al.*, 2002;

Guiard & Beaudouin-Lafon, 2004) et des affichages muraux (Liu *et al.*, 2014; Rädle *et al.*, 2014) s'appliquaient également à un téléphone à écran étendu.

Notre première contribution est l'exploration des IHMs et interactions possibles avec ce concept d'extension de l'écran (*Voir* chapitre 2) qui avait déjà été proposé par Grubert *et al.* (2015) pour étendre l'écran d'une montre intelligente. Un VESAD peut être utilisé selon deux modes de vues : en *vue multi-fenêtres*, pour supporter du multi-tâche, et en *vue étendue*, où une application utilise tout l'écran étendu (par exemple une carte de navigation). Nous présentons aussi deux nouvelles techniques d'interaction : *wrist*, où l'utilisateur fait une rotation rapide du poignet pour substituer l'affichage sur le VESAD avec un autre (similaire à un changement de bureau sur un PC), et *slide-to-hang* pour détacher l'écran étendu en une fenêtre virtuelle séparée du téléphone. Enfin, nous proposons une modification au cadre théorique Ethereal Planes (Ens *et al.*, 2014a), permettant de concevoir des IHMs utilisant des fenêtres 2D en RA.

Nous détaillons ensuite au chapitre 3 la conception de notre propre visiocasque de RA à large champ de vision ($100° \times 98°$ pour chaque œil). Nous reprenons le principe du visiocasque de Steptoe (2013), repris par Steptoe *et al.* (2014) et Piumsomboon *et al.* (2014), qui consiste à utiliser une caméra stéréoscopique *fisheye* avec un visiocasque de RV. Notre deuxième contribution est la documentation précise de ce développement, permettant à d'autres chercheurs de reproduire ce visiocasque. Un VESAD fonctionne à condition qu'il y ait (1) un excellent suivi en 3D de l'écran physique à étendre, (2) un bon alignement de l'écran virtuel avec l'écran physique et (3) une synchronisation sans latence entre ces deux écrans. Notre troisième contribution est la réalisation de la bibliothèque libre de réalité augmentée ArucoUnity (https://github.com/NormandErwan/ArucoUnity), qui porte sur le moteur 3D Unity les fonctions de suivi de marqueurs et de calibration de caméra de la bibliothèque libre de vision par ordinateur OpenCV. Elle résout les points (1) et (2).

Notre quatrième contribution sont les résultats de notre étude expérimentale (*Voir* chapitre 4), menée avec 12 participants. Nous y avons reproduit une tâche de classement issue de la littérature sur les affichages muraux, en mode vue étendue et impliquant de la navigation et des sélections. Nous y avons en outre implémenté des techniques d'interactions décrites par Wobbrock *et al.* (2009) pour l'écran tactile et Piumsomboon *et al.* (2013) pour la main virtuelle. Nos résultats indiquent que la combinaison du téléphone à écran étendu avec les interactions tactiles s'est montrée la plus rapide, en particulier quand la tâche était maîtrisée des participants, et la plus performante en termes de navigation. Les participants l'ont également préférée. Néanmoins, plusieurs d'entre eux ont vu plus de potentiel dans les interactions avec la main virtuelle avec l'écran étendu. Enfin, lors de la conception d'une IHM pour un VESAD, nous recommandons de limiter l'espace d'interaction aux zones faciles à accéder. Nous donnons également des pistes d'amélioration pour l'utilisation d'un écran étendu avec une main virtuelle.

Notre recherche présente bien évidemment des limites. Tout d'abord, le mauvais suivi de la main des participants dans l'étude expérimentale a joué en la défaveur du VESAD, qui a été la moins performante des trois IHMs évaluées. En outre, nous avons mal conçu les gestes pour manipuler l'écran étendu du VESAD, rendant les défilements et zooms avec la main virtuelle particulièrement difficiles. Ensuite, nous n'avons pas correctement contrôlé la difficulté de notre tâche, car nous n'avons mesuré aucune différence entre les conditions « faciles » et « difficiles » de notre tâche. Enfin, la principale limite de notre étude est la faible densité visuelle (mesuré en pixels par degré) ainsi que la latence de notre visiocasque, ce qui a désavantagé le téléphone seul. Il est possible qu'améliorer le visiocasque utilisé change nos résultats, mais il nous semble très probable que certaines tâches bénéficieraient d'un écran étendu.

Nous esquissons pour finir quelques directions futures à ce travail de recherche. L'amélioration du suivi de la main et de techniques d'interactions avec le VESAD nous semble être le plus essentiel. Il serait intéressant alors de reproduire notre étude expérimentale et d'y comparer

d'autres techniques d'interactions comme un pointeur virtuel (répandue parmi les visiocasques de l'industrie, comme le Microsoft HoloLens) ou des commandes vocales. Enfin, notre concept de téléphone à écran étendu devrait être évalué sur d'autre d'autres types de tâche comme, par exemple, un scénario multi-tâches, plus proche des usages quotidiens avec un téléphone intelligent.

ANNEXE I

FORMULAIRES POUR DE L'EXPÉRIENCE

Formulaire d'information et de consentement

Titre du projet de recherche

Agrandissement d'un écran mobile par réalité augmentée.

Nom des chercheurs

Michael McGuffin, professeur, ÉTS

Erwan Normand, étudiant de maîtrise, ÉTS

Source de financement

Ce projet est financé par le CRSNG

Invitation à participer à un projet de recherche

Le professeur Michael McGuffin, département de génie logiciel et des TIs, dirige un projet de recherche dans le but d'améliorer les interfaces utilisateurs mobiles. Nous sollicitons aujourd'hui votre participation. Nous vous invitons à lire attentivement ce formulaire d'information et de consentement afin de décider si vous souhaitez participer à ce projet de recherche. Il est important de bien comprendre ce formulaire. N'hésitez pas à poser des questions. Prenez le temps nécessaire pour prendre votre décision.

Nature du projet de recherche

Ce projet vise à mieux comprendre l'effet d'une surface d'affichage agrandie pour les dispositifs mobiles. Une compréhension approfondie de cet effet pourra éventuellement aider les chercheurs et concepteurs à développer de meilleures interfaces mobiles et/ou de réalité augmentée. Cette étude demande la participation de 12 personnes. Les participants doivent avoir une vision normale ou corrigée.

Déroulement du projet de recherche

Si vous êtes admissible et désirez participer, suite à la lecture et à la signature de ce Formulaire d'information et de consentement, nous allons vous demander d'effectuer des tâches avec un téléphone intelligent, et parfois avec un casque de réalité augmentée. On vous demandera aussi de remplir un pré-questionnaire et un post-questionnaire. Votre participation prendra un total d'environ 60 minutes (présentation et pré-questionnaire de 10 minutes, 45 minutes de tâches, post-questionnaire de 5 minutes). Pendant l'expérience, vous êtes libre de prendre des pauses.

Tâches en réalité augmentée

Les tâches en réalité augmentée consistent à porter un visiocasque (montrés ci-dessous : soit un HoloLens (à gauche) ou un OVRVision (à droite)) et d'effectuer des sélections avec vos doigts.

Cette expérience va demander d'être assis sur une chaise, avec le téléphone dans une main, et avec un visiocasque sur votre tête. L'expérience vous demande de faire des sélections de cibles avec vos doigts, aussi rapidement que possible, et en faisant le moins d'erreurs possibles.

Pré-questionnaire et post-questionnaire

Un pré-questionnaire d'environ 5 minutes sollicitera des informations générales à propos de vous. Un poste-questionnaire d'environ 5 minutes demandera vos opinions subjectives par rapport aux interfaces que vous avez utilisées dans l'expérience.

Avantages et bénéfices

Les résultats de ces recherches pourront permettre des futures améliorations aux interfaces mobiles.

Quels sont les inconvénients et les risques ?

Les casques de réalité augmentée peuvent causer de la nausée, le vertige, ou une fatigue des yeux. Si vous vous sentez mal à l'aise ou fatigué pendant l'étude, vous êtes exigé de prendre une pause. Vous êtes libres aussi de vous retirer si vous le désirez, sans aucune pénalité.

Confidentialité

Tous les renseignements obtenus pour ce projet de recherche seront confidentiels dans les limites prévues par la loi. Pour ce faire, ces renseignements seront codés. Les dossiers seront conservés pendant trois (3) années après la fin de la recherche, sous la responsabilité de Michael McGuffin à l'ÉTS. Les renseignements codés seront conservés sur un support informatique qui sera placé sous-clé dans le bureau du professeur McGuffin. Seuls les membres de l'équipe de recherche (Michael McGuffin, professeur, et Erwan Normand, étudiant de maîtrise) auront accès à votre dossier.

Par ailleurs, les résultats de cette recherche pourront être publiés ou communiqués dans un congrès scientifique, mais aucune information pouvant vous identifier ne sera alors dévoilée.

Les questionnaires et données enregistrées seront détruits trois (3) années après la fin du projet de recherche.

Compensation financière

Vous recevrez 20$ (environ 10€) en argent comptant en compensation pour votre participation dans l'étude. Si vous vous retirez de l'étude avant la fin, vos données seront détruites, mais vous aurez quand même droit à la carte cadeau.

Conflits d'intérêts

Il n'y a pas de conflits d'intérêt à déclarer.

Participation volontaire et droit de retrait

Votre participation à ce projet est volontaire. Cela signifie que vous acceptez de participer au projet sans aucune contrainte ou pression extérieure, et que par ailleurs vous être libre de mettre fin à votre participation en tout temps au cours de cette recherche. Dans le cas où vous décidiez de vous retirer du projet, soit en cours de réalisation du projet ou à tout autre moment, les renseignements recueillis seront détruits.

Votre accord à participer implique également que vous acceptez que l'équipe de recherche puisse utiliser les renseignements recueillis aux fins de la présente recherche pour la publication d'articles ou encore lors de conférences et communications scientifiques, à la condition qu'aucune information permettant de vous identifier ne soit divulguée publiquement.

Possibilité de commercialisation

Votre participation à ce projet de recherche pourrait mener à la création de produits commerciaux qui pourraient être éventuellement protégés par voie de brevet ou autres droits de propriété intellectuelle. Cependant, dans un tel cas, vous ne pourrez en retirer un avantage financier.

Questions sur le projet ou sur vos droits

L'étude est réalisée par le professeur Michael McGuffin (tél. : 514-396-8418, courriel : michael.mcguffin@etsmtl.ca) et par son étudiant de maîtrise Erwan Normand.

Le Comité d'éthique de la recherche avec des êtres humains de l'ÉTS a approuvé le projet de recherche auquel vous allez participer. Pour toute autre question concernant vos droits en tant que sujet de recherche, vous pouvez contacter le président du Comité d'éthique de l'ÉTS au (514) 396-8730.

À des fins de surveillance et de contrôle, votre dossier de recherche pourrait être consulté par une personne mandatée par le Comité d'éthique de la recherche de l'ÉTS ainsi que par des représentants des organismes subventionnaires réglementaires concernés. Toutes ces personnes adhèrent à une politique de confidentialité.

Consentement et assentiment

Je, soussigné(e), reconnais avoir lu le présent formulaire de consentement et consens volontairement à participer à ce projet de recherche. Je reconnais avoir disposé de suffisamment de renseignements et du temps nécessaire pour réfléchir à ma décision. Je comprends que ma participation à cette recherche est totalement volontaire et que je peux y mettre fin en tout temps, sans pénalité d'aucune forme, ni justification à donner. Le cas échéant, je m'engage à prévenir le responsable du projet.

En signant le présent formulaire, je ne renonce aucunement à mes droits ni ne libère le(s) chercheur(s) de leurs responsabilités légales et professionnelles.

Nom du participant(e) (lettres moulées)	Signature	Date

J'ai expliqué au participant tous les aspects pertinents de la recherche et j'ai répondu aux questions qu'il m'a posées. Je lui ai indiqué que la participation au projet de recherche est libre et volontaire et que sa participation peut être cessée en tout temps.

Nom de la personne qui a obtenu le consentement (lettres moulées)	Signature	Date

Pré-questionnaire

Agrandissement d'un écran mobile par réalité augmentée

- Numéro d'identification du participant :

- Sexe : M / F

- Lunettes ? Oui / Non Lentilles cornéennes ? Oui / Non Daltonien ? Oui / Non

- Âge :

[6-10] [11-15] [16-20] [21-25] [26-30] [31-35] [36-40]

[41-45] [46-50] [51-55] [56-60] [61-65] [66-70] [71-75]

- Laquelle de vos mains est dominante ? gauche / droite / ambidextre

- Quelle main utilisez-vous avec la souris ? gauche / droite / ambidextre

- Quelle est votre programme d'études / profession / champ d'expertise ?

- Combien d'heures par jour utilisez-vous un ordinateur ? _____

- Utilisez-vous régulièrement des logiciels 3D (jeux, logiciels de modélisation, etc.) ? Spécifiez lesquels

- Avez-vous déjà utilisé un visiocasque de réalité virtuelle ou de réalité augmentée ? Spécifiez lesquels

- Si oui, quelles tâches avez-vous effectuées avec ce(s) visiocasques ?

Post-questionnaire

Agrandissement d'un écran mobile par réalité augmentée

1. Est-ce que le fonctionnement de l'interface était facile à comprendre ? (encerclez un niveau de 1 à 5)

a) téléphone seulement : (difficile) 1 ... 2 ... 3 ... 4 ... 5 (facile)

b) téléphone pour toucher, espace agrandi pour afficher : (difficile) 1 ... 2 ... 3 ... 4 ... 5 (facile)

c) espace agrandi pour toucher et pour afficher : (difficile) 1 ... 2 ... 3 ... 4 ... 5 (facile)

2. Est-ce que l'interface était mentalement facile à utiliser (réflexion, décision, calcul, mémorisation, observation, recherche, etc.) ?

a) téléphone seulement : (difficile) 1 ... 2 ... 3 ... 4 ... 5 (facile)

b) téléphone pour toucher, espace agrandi pour afficher : (difficile) 1 ... 2 ... 3 ... 4 ... 5 (facile)

c) espace agrandi pour toucher et pour afficher : (difficile) 1 ... 2 ... 3 ... 4 ... 5 (facile)

3. Est-ce que l'interface était physiquement facile à utiliser (effort physique, tenue du téléphone, sélection des disques avec l'index, etc.) ?

a) téléphone seulement : (difficile) 1 ... 2 ... 3 ... 4 ... 5 (facile)

b) téléphone pour toucher, espace agrandi pour afficher : (difficile) 1 ... 2 ... 3 ... 4 ... 5 (facile)

c) espace agrandi pour toucher et pour afficher : (difficile) 1 ... 2 ... 3 ... 4 ... 5 (facile)

4. Pouviez-vous interagir rapidement avec l'interface ?

a) téléphone seulement : (pas du tout) 1 ... 2 ... 3 ... 4 ... 5 (très)

b) téléphone pour toucher, espace agrandi pour afficher : (pas du tout) 1 ... 2 ... 3 ... 4 ... 5 (très)

c) espace agrandi pour toucher et pour afficher : (pas du tout) 1 ... 2 ... 3 ... 4 ... 5 (très)

5. Quelle réussite pensez-vous avoir eu dans l'exécution de la tâche ?

a) téléphone seulement : (mauvaise) 1 ... 2 ... 3 ... 4 ... 5 (bonne)

b) téléphone pour toucher, espace agrandi pour afficher : (mauvaise) 1 ... 2 ... 3 ... 4 ... 5 (bonne)

c) espace agrandi pour toucher et pour afficher : (mauvaise) 1 ... 2 ... 3 ... 4 ... 5 (bonne)

6. Quel niveau de frustration aviez-vous eu pendant l'exécution de la tâche ? Découragé, irrité, stressé (frustration élevée), ou plutôt satisfait, relaxé (frustration faible) ?

a) téléphone seulement : (élevé) 1 ... 2 ... 3 ... 4 ... 5 (faible)

b) téléphone pour toucher, espace agrandi pour afficher : (élevé) 1 ... 2 ... 3 ... 4 ... 5 (faible)

c) espace agrandi pour toucher et pour afficher : (élevé) 1 ... 2 ... 3 ... 4 ... 5 (faible)

7. Classer les interfaces en fonction de celle que vous avez le plus aimé (1 pour la plus appréciée, 3 pour la moins appréciée) :

___ : a) téléphone seulement

___ : b) téléphone pour toucher, espace agrandi pour afficher

___ : c) espace agrandi pour toucher et pour afficher

8. Que pensez-vous des scénarios suivant d'utilisations potentielles de ce projet (intéressants, utiles) ?

1. Une carte de navigation type Google Maps en écran agrandi.
2. Des applications pouvant être placées autour de l'écran (par exemple : application de discussion avec les contacts à gauche du téléphone, les pages web de liens reçus à droite du téléphone)
3. Les notifications apparaissant en 3D au dessus du téléphone.

9. Avez-vous des suggestions sur la façon d'améliorer les interfaces b et c (avec l'espace agrandi) ?

10. Avez-vous d’autres commentaires ?

Merci pour votre temps et vos commentaires !

Formulaire de compensation de participant

Projet : Agrandissement d'un écran mobile par réalité augmentée

Responsable : prof. Michael McGuffin

Nom du participant :

Adresse :

Téléphone :

Je reconnais par la présente avoir reçu une somme de _____ dollars en argent comptant pour ma participation à cette étude.

Signature : _____ Date : _____

ANNEXE II

ANALYSE DÉTAILLÉE DE L'EXPÉRIENCE

The analysis of the experiment data is also accessible online: https://github.com/NormandErwan/HandheldVesadAnalysis/blob/master/Handheld%20VESAD%20Analysis.ipynb.

0.1 1. Data preparation

Configuration :

```
In [1]: # Imports
        import numpy as np
        import pandas as pd
        from pandas.api.types import CategoricalDtype
        import itertools

        import seaborn as sns
        import matplotlib as mpl
        import matplotlib.pyplot as plt
        import matplotlib.ticker as ticker
        import matplotlib.patches as patches

        from scipy import stats
        import statsmodels.api as sm
        from statsmodels.formula.api import ols
        from statsmodels.stats.multicomp import (MultiComparison, pairwise_tukeyhsd)
        from statsmodels.stats.multitest import multipletests
        from statsmodels.stats.libqsturng import psturng

        from ast import literal_eval
        from os import listdir
        from os.path import join

        from IPython.html.services.config import ConfigManager

C:\Users\Erwan\Miniconda\envs\master-thesis\lib\site-packages\IPython\html.py:14: ShimWarning: The
  "`IPython.html.widgets` has moved to `ipywidgets`.", ShimWarning)


In [2]: # Notebook configuration
        %matplotlib inline

        # Ruler
```

```
ip = get_ipython()
cm = ConfigManager(parent=ip)
cm.update('notebook', {"ruler_column": [80]})

# Style and size of the figures
subplotsize = (5, 4)
legend_fontsize = sns.plotting_context('notebook')['axes.labelsize']
legend_title_fontsize = legend_fontsize + 1
sns.set(context='notebook', style='whitegrid', font_scale=1.25,
        rc={'legend.fontsize': legend_fontsize, 'savefig.dpi': 300,
            'savefig.bbox': 'tight'})

# Render the plots with this language
#language = 'English'
language = 'Français'
```

Data are loaded in the following variables: - The ranks of the participants from the post-questionary: `ranks` - Trials of the participants: `raw_trials` - Trials averaged so that each participants ends up with a single observation (Tip 9 of Dragicevic, 2016) : `trials` > Dragicevic, P. (2016). Fair statistical communication in HCI. In Modern Statistical Methods for HCI (pp. 291-330). Springer, Cham.

```
In [3]: trials = pd.read_csv('participant_trials.csv')
        raw_trials = pd.read_csv('participant_trials.csv')
        ranks = pd.read_csv('participant_ranks.csv')
```

Creates independent variables lists (`ivs`) and dependent variables lists (`ranks_dvs`, `trials_dvs`) to make easier to use algorithms and to plot figures.

```
In [4]: # IVs
        if language == 'Français':
            iv_labels = ['IHM', 'Taille du texte', 'Distance', 'Groupe']
        else:
            iv_labels = ['Technique', 'Text Size', 'Distance', 'Ordering']

        ivs = ['technique', 'text_size', 'distance', 'ordering']
        ivs = pd.DataFrame(columns=ivs, index=['label', 'categorical', 'palette'])

        # Shortcuts
        technique = ivs['technique']
        text_size = ivs['text_size']
        distance = ivs['distance']
        ordering = ivs['ordering']

        # Setup categories and palettes
        palette = sns.color_palette('Set2', 8)
        for iv_id, iv_label in zip(ivs, iv_labels):
            iv_categories = trials.drop_duplicates(iv_id)\
                                  .sort_values([iv_id + '_id'])[iv_id]
            iv_categorical = pd.Categorical(iv_categories, iv_categories, ordered=True)
            ivs[iv_id] = [iv_label, iv_categorical, palette]
```

```
technique['palette'] = [palette[1], palette[0], palette[2]]
text_size['palette'] = sns.light_palette(palette[6], 3)[1:3]
distance['palette'] = sns.light_palette(palette[4], 3)[1:3]
ordering['palette'] = [palette[1], palette[0], palette[2]]
```

In [5]:
```
# Trials DVs
if language == 'Français':
    trials_dv_labels = ['Temps de complétion (s)', 'Sélections',
                        'Temps en sélection', 'Distance 3D en sélection',
                        'Distance en sélection', 'Déselections', 'Erreurs',
                        'Disques classés', 'Défilements', 'Temps de défilement',
                        'Distance 3D de défilement', 'Distance de défilement',
                        'Zooms', 'Temps de zoom', 'Distance 3D de zoom',
                        'Distance de zoom', 'Mouvements tête-téléphone',
                        'Distance relative tête-téléphone']
else:
    trials_dv_labels = ['Task Completion Time (s)', 'Selections',
                        'Selection Time', 'Selection Distance',
                        'Selection Distance on Grid', 'Deselections', 'Errors',
                        'Items Classified', 'Pans', 'Pan Time', 'Pan Distance',
                        'Pan Distance on Grid', 'Zooms', 'Zoom Time',
                        'Zoom Distance', 'Zoom Distance on Grid',
                        'Phone-Head Motion', 'Signed Phone-Head Motion']

trials_dvs = trials.loc[:, 'total_time':'signed_head_phone_distance'].columns
trials_dvs = pd.Series(data=trials_dv_labels, index=trials_dvs)

# Shortcuts
participant_id = 'Participant'
total_time = trials_dvs['total_time']
```

In [6]:
```
# Ranks DVs
if language == 'Français':
    ranks_dv_labels = ['Facile à comprendre', 'Mentalement facile à utiliser',
                       'Physiquement facile à utiliser', 'Rapidité',
                       'Performance', 'Frustration', 'Préférence']
else:
    ranks_dv_labels = ['Easy to Understand', 'Mentally Easy to Use',
                       'Physically Easy to Use', 'Subjective Speed',
                       'Subjective Performance', 'Frustration', 'Preference']

ranks_dv_scales = [pd.Categorical(list(range(1, 6)), list(
    range(1, 6)), ordered=True)] * len(ranks_dv_labels)

RdYlBu = sns.color_palette('RdYlBu', 5)
ranks_dv_palettes = [sns.color_palette('RdYlBu', 5)] * len(ranks_dv_labels)

ranks_dvs = ranks.loc[:, 'easy_understand':'preference'].columns
ranks_dvs = pd.DataFrame(data=[ranks_dv_labels, ranks_dv_scales,
                               ranks_dv_palettes],
                         columns=ranks_dvs, index=['label', 'scale', 'palette'])
```

```
        ranks_dvs.at['scale', 'preference'] = pd.Categorical(
            list(range(1, 4)), list(range(1, 4)), ordered=True)
        ranks_dvs.at['palette', 'preference'] = [RdYlBu[4], RdYlBu[2], RdYlBu[0]]
```

Clean the data:

```
In [7]: # Setup columns of trials and ranks
        columns = []
        for column in trials.columns:
            if (column in ivs.columns):
                columns.append(ivs[column]['label'])
            elif (column in trials_dvs.index):
                columns.append(trials_dvs[column])
            else:
                columns.append(column)
        columns[0] = participant_id

        trials.columns = columns
        ranks.columns = [participant_id, ordering['label'], technique['label']]\
                        + ranks_dvs.loc['label', :].tolist()

        # Setup translated column values of trials and ranks
        for iv_id in ivs:
            iv = ivs[iv_id]
            trials[iv['label']] = trials[iv['label']].astype(iv['categorical'])
            raw_trials[iv_id] = raw_trials[iv_id].astype(iv['categorical'])

        for iv_id in ['technique', 'ordering']:
            iv = ivs[iv_id]
            ranks[iv['label']] = ranks[iv['label']].astype(iv['categorical'])

        if language == 'Français':
            technique['categorical'].categories = ['Téléphone', 'VESAD tactile', 'VESAD']
            text_size['categorical'].categories = ['Grand', 'Petit']
            distance['categorical'].categories = ['Proche', 'Loin']
            ordering['categorical'].categories = ['Groupe 1', 'Groupe 2', 'Groupe 3']
        else:
            ordering['categorical'].categories = ['Group 1', 'Group 2', 'Group 3']

        for iv_id in ivs:
            iv = ivs[iv_id]
            trials[iv['label']].cat.categories = iv['categorical'].categories

        for iv_id in ['technique', 'ordering']:
            iv = ivs[iv_id]
            ranks[iv['label']].cat.categories = iv['categorical'].categories

In [8]: # Some participants are non valid or don't have complete measures
        invapars = [0, 4]
        ranks = ranks[~ranks[participant_id].isin(invapars)].reset_index(drop=True)
```

```
        trials = trials[~trials[participant_id].isin(invapars)].reset_index(drop=True)
        raw_trials = raw_trials[~raw_trials['participant_id'].isin(invapars)
                                ].reset_index(drop=True)
```

In [9]:
```
        # Some participants have wrong head phone mesures
        for dist in ['absolute_head_phone_distance', 'signed_head_phone_distance']:
            trials.loc[trials[trials_dvs[dist]] == 0, trials_dvs[dist]] = np.nan
            raw_trials.loc[raw_trials[dist] == 0, dist] = np.nan
```

In [10]:
```
        # Eval the arrays in some dvs
        def eval_if_str(data):
            return literal_eval(data) if isinstance(data, str) else data

        trials['grid_config'] = trials['grid_config'].apply(eval_if_str)
```

In [11]:
```
        # Average trials to have one observation per participant
        trials = trials.groupby([participant_id] + ivs.loc['label', :].tolist(),
                                observed=True).mean().reset_index()
        raw_trials = raw_trials.groupby(['participant_id'] + ivs.columns.tolist(),
                                observed=True).mean().reset_index()
```

Utilities:

In [12]:
```
        # Figure labels in the selected language
        labels = pd.Series()

        if language == 'Français':
            labels['category'] = 'Catégorie'
            labels['count'] = 'Nombre total'
            labels['distance'] = 'Distance moyenne (m)'
            labels['dv'] = 'VD'
            labels['iv'] = 'VI'
            labels['iv_value'] = 'Valeur VI'
            labels['mean_difference'] = 'Différence des moyennes'
            labels['mean_difference_percentage'] = 'Différence des moyennes (%)'
            labels['mean_rank'] = 'Note moyenne'
            labels['preferences'] = ['Premier', 'Deuxième', 'Troisième']
            labels['p_value'] = 'Valeur p'
            labels['question'] = 'Question'
            labels['rank'] = 'Note'
            labels['time'] = 'Temps moyen (s)'
            labels['t_statistic'] = 'Statistique T'
            labels['votes'] = 'Participants'
        else:
            labels['category'] = 'Category'
            labels['count'] = 'Count'
            labels['distance'] = 'Mean Distance (m)'
            labels['dv'] = 'DV'
            labels['iv'] = 'IV'
            labels['iv_value'] = 'IV Value'
            labels['mean_difference'] = 'Mean Difference'
```

```
            labels['mean_difference_percentage'] = 'Mean Difference Percentage'
            labels['mean_rank'] = 'Mean Rank'
            labels['preferences'] = ['First', 'Second', 'Third']
            labels['p_value'] = 'p-value'
            labels['question'] = 'Question'
            labels['rank'] = 'Rank'
            labels['time'] = 'Mean Time (s)'
            labels['t_statistic'] = 'T statistic'
            labels['votes'] = 'Participants'

In [13]: def mean_ci(x, which=95, n_boot=1000):
            """Returns the confidence interval of the mean"""
            x2 = [i for i in x if not np.isnan(i)]
            boots = sns.algorithms.bootstrap(x2, n_boot=n_boot)
            return sns.utils.ci(boots, which=which)

In [14]: def print_mean_ci(x, ci_which=95):
            """Returns a string containing the mean with the CI of x"""
            ci1, ci2 = mean_ci(x, which=ci_which)
            return '{:.2f} [{:.2f}, {:.2f}]'.format(np.mean(x), ci1, ci2)

In [15]: def exp_mean(x):
            return np.exp(np.mean(x))

In [16]: def print_exp_mean_ci(x, ci_which=95):
            ci1, ci2 = np.exp(mean_ci(x, which=ci_which))
            return '{:.2f} [{:.2f}, {:.2f}]'.format(exp_mean(x), ci1, ci2)

In [17]: def geometric_mean(x):
            return exp_mean(np.log(x))

In [18]: def print_geo_mean_ci(x, ci_which=95):
            ci1, ci2 = np.exp(mean_ci(np.log(x), which=ci_which))
            return '{:.2f} [{:.2f}, {:.2f}]'.format(geometric_mean(x), ci1, ci2)

In [19]: def mean_difference(a, b):
            """Returns the mean difference value and percentage between a and b"""
            mean_diff = np.mean(a) - np.mean(b)
            mean_diff_percentage = mean_diff / np.mean(b) * 100
            return (mean_diff, mean_diff_percentage)

In [20]: def p_values_correction(data, alpha=0.05, correction_method='fdr_bh'):
            if correction_method != None:
                reject, p_values_corrected, a1, a2 =\
                    multipletests(data[labels['p_value']].tolist(), alpha=alpha,
                                  method=correction_method)
                data[labels['p_value']] = p_values_corrected

In [21]: def subplots(nsubplots, ncols_max=2, subplotsize=subplotsize, *plt_args):
            ncols = min(ncols_max, nsubplots)
            nrows = ((nsubplots - 1) // ncols) + 1
            fig, axs = plt.subplots(nrows=nrows, ncols=ncols,\
```

```
                                    figsize=(subplotsize[0]*ncols, subplotsize[1]*nrows),
                                    *plt_args)

        if nrows == 1 and ncols == 1:
            axs = [axs]
        elif nrows >= 2 and ncols >= 2:
            axs = [ax for ax_row in axs for ax in ax_row]

        for ax in axs[::-1][0:len(axs) - nsubplots]:
            fig.delaxes(ax)

        return (fig, axs)
```

```
In [22]: def fix_legend_fontsize(ax_legend, fontsize=legend_title_fontsize):
             plt.setp(ax_legend.get_title(), fontsize=fontsize)
```

```
In [23]: def config_legend(ax, iv_id, fontsize=legend_title_fontsize):
             legend = ax.legend(title=ivs[iv_id]['label'], frameon=True,
                                loc='center left', bbox_to_anchor=(1, 0.5))
             fix_legend_fontsize(legend)
             return legend
```

0.2 2. Participant ranks

Some functions for the analysis:

```
In [24]: def get_ranks_count(iv_index, dv_index):
             iv, dv = ivs[iv_index], ranks_dvs[dv_index]

             ranks_counts_index = pd.MultiIndex.from_product([iv['categorical'],
                                                              dv['scale']],
                                                             names=[iv['label'],
                                                                    labels['rank']])
             # Zero counts by default and gets the counts
             default_counts = pd.Series(0, index=ranks_counts_index)
             ranks_counts = ranks.groupby([iv['label'], dv['label']]).size()

             # Merge and remove duplicated defaults
             ranks_counts = pd.concat([ranks_counts, default_counts])
             ranks_counts = ranks_counts[~ranks_counts.index.duplicated(keep='first')]

             ranks_counts.sort_index(inplace=True)
             ranks_counts.index = ranks_counts_index # Restore the index
             return ranks_counts
```

```
In [25]: def cumulated_barplot(data, palette, **args):
             for row_id, row in data.iloc[::-1].iterrows():
                 sns.barplot(y=data.columns, x=row, label=row_id,
                             color=palette[row_id-1], orient='h', **args)
```

```
In [26]: def plot_ranks_distributions(iv_id, dv_ids):
             iv = ivs[iv_id]
```

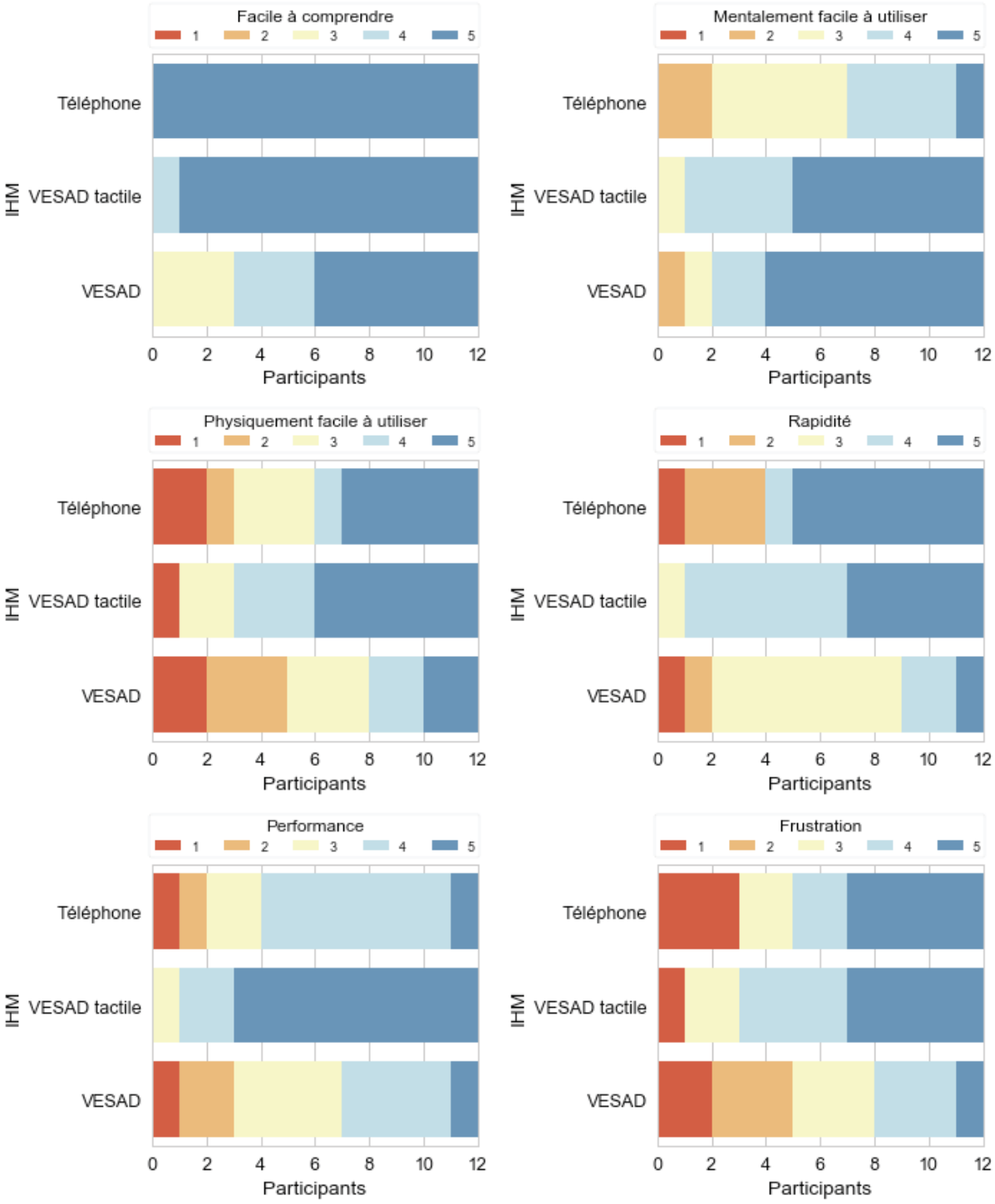
```
            fig, axs = subplots(len(dv_ids))
            for dv_id, ax in zip(dv_ids, axs):
                dv = ranks_dvs[dv_id]

                cumulated_ranks_count = get_ranks_count(iv_id, dv_id).unstack(level=0)\
                                        .cumsum()
                cumulated_barplot(cumulated_ranks_count, palette=dv['palette'], ax=ax)
                ax.set(xlabel=labels['votes'], xlim=(0, cumulated_ranks_count.max()[0]))
                ax.xaxis.set_major_locator(ticker.MultipleLocator(2)) # Fix the axis ticks

                ax_handles, ax_labels = ax.get_legend_handles_labels()
                legend = ax.legend(ax_handles[::-1], ax_labels[::-1], frameon=True,
                                   loc='lower center', bbox_to_anchor=(0.5, 1),
                                   mode=None, ncol=len(dv['scale']), title=dv['label'],
                                   fontsize=legend_fontsize-2)
                fix_legend_fontsize(legend)

            fig.tight_layout(h_pad=1) # Add padding to avoir legend and labels overlap
            return (fig, axs)

In [27]: def plot_ranks(iv_id, dv_ids, estimator=np.mean):
            iv = ivs[iv_id]

            fig, axs = subplots(len(dv_ids))
            for dv_id, ax in zip(dv_ids, axs):
                dv = ranks_dvs[dv_id]

                sns.barplot(x=iv['label'], y=dv['label'], palette=iv['palette'],
                            data=ranks, ax=ax, estimator=estimator)
                ax.set(ylim=(0, dv['scale'][-1]))
                ax.yaxis.set_major_locator(ticker.MultipleLocator(1)) # Fix the axis ticks

            return (fig, axs)

In [28]: def rank_samples(iv_id, dv_id):
            """Returns the list of ranks (DV) for each IV value"""
            samples = []
            iv = ivs[iv_id]
            for iv_value in iv['categorical']:
                dv_label = ranks_dvs[dv_id]['label']
                sample = ranks[ranks[iv['label']] == iv_value][dv_label]\
                         .reset_index(drop=True)
                samples.append(sample)
            return samples

In [29]: def test_ranks(iv_id, dv_ids, **args):
            results = []
            iv = ivs[iv_id]

            for dv_id in dv_ids:
```

```
            dv = ranks_dvs[dv_id]
            samples = rank_samples(iv_id, dv_id)
            H, p = stats.kruskal(*samples)
            results.append([iv['label'], dv['label'], H, p])

        results = pd.DataFrame(results, columns=[labels['iv'], labels['dv'],
                                                 'Kruskal-Wallis H',
                                                 labels['p_value']])
        p_values_correction(results, **args)
        return results
```

```
In [30]: def test_pairwise_ranks(iv_id, dv_ids, **args):
        iv = ivs[iv_id]
        iv_category_ids = range(len(iv['categorical']))

        results = []
        for dv_id in dv_ids:
            dv = ranks_dvs[dv_id]
            samples = rank_samples(iv_id, dv_id)
            sample_pairs = itertools.combinations(iv_category_ids, 2)
            for id1, id2 in sample_pairs:
                U, p = stats.mannwhitneyu(samples[id1], samples[id2])
                mean_diff, mean_diff_per = mean_difference(samples[id1], samples[id2])
                results.append([iv['label'], iv['categorical'][id1],
                                iv['categorical'][id2], dv['label'],
                                mean_diff, mean_diff_per, U, p])

        columns = [labels['iv'], labels['iv_value'] + ' 1',
                   labels['iv_value'] + ' 2', labels['dv'],
                   labels['mean_difference'], labels['mean_difference_percentage'],
                   'Mann-Whitney U', labels['p_value']]
        results = pd.DataFrame(results, columns=columns)
        p_values_correction(results, **args)
        return results
```

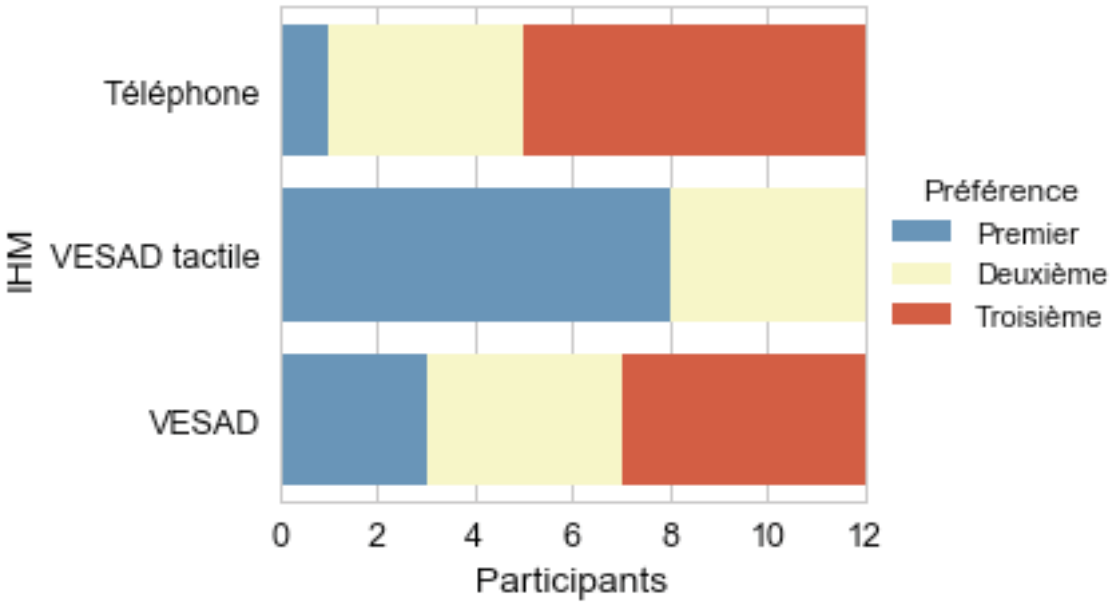
We display the rank and preference distributions:

```
In [31]: (fig, axs) = plot_ranks_distributions('technique', ranks_dvs.columns[0:-1])
         fig.savefig('ranks_distributions.png')
```




```
In [32]: (fig, axs) = plot_ranks_distributions('technique', ['preference'])

         ax_handles, ax_labels = axs[0].get_legend_handles_labels()
         legend = axs[0].legend(ax_handles[::-1], labels['preferences'],
                                loc='center left', bbox_to_anchor=(1, 0.5),
                                title=ranks_dvs['preference']['label'])
```

```
fix_legend_fontsize(legend)

fig.savefig('preferences_distribution.png')
```



We use the Kruskal-Wallis test (Benjamini–Hochberg correction) on each question to check if there is statistical significative differences between the ranks among TECHNIQUE:

```
In [33]: test_ranks('technique', ranks_dvs, correction_method='fdr_bh')
```

```
Out[33]:     VI                              VD  Kruskal-Wallis H  Valeur p
         0  IHM             Facile à comprendre         11.001420  0.007497
         1  IHM   Mentalement facile à utiliser         10.905742  0.007497
         2  IHM  Physiquement facile à utiliser          4.148121  0.125674
         3  IHM                        Rapidité          7.130846  0.039599
         4  IHM                     Performance         13.784269  0.006303
         5  IHM                     Frustration          4.500057  0.122962
         6  IHM                      Préférence         12.638889  0.006303
```

All the questions, except Physically Easy to Use and Frustration, are statistically significants: Easy to Understand ($p = 0.007$), Mentally Easy to Use ($p = 0.007$), Subjective Speed ($p = 0.04$), Subjective Performance ($p = 0.006$), Preference ($p = 0.006$).

We use then pairwise Mann-Whitney tests (Benjamini–Hochberg correction) for the significant questions above:

```
In [34]: test_pairwise_ranks('technique', ['easy_understand', 'mentally_easy_use',
                                            'could_go_fast', 'subjective_performance',
                                            'preference'], correction_method='fdr_bh')
```

```
Out[34]:     VI    Valeur VI 1    Valeur VI 2                            VD  \
         0  IHM      Téléphone  VESAD tactile           Facile à comprendre
         1  IHM      Téléphone          VESAD           Facile à comprendre
```

```
2   IHM  VESAD tactile          VESAD            Facile à comprendre
3   IHM      Téléphone  VESAD tactile  Mentalement facile à utiliser
4   IHM      Téléphone          VESAD  Mentalement facile à utiliser
5   IHM  VESAD tactile          VESAD  Mentalement facile à utiliser
6   IHM      Téléphone  VESAD tactile                      Rapidité
7   IHM      Téléphone          VESAD                      Rapidité
8   IHM  VESAD tactile          VESAD                      Rapidité
9   IHM      Téléphone  VESAD tactile                   Performance
10  IHM      Téléphone          VESAD                   Performance
11  IHM  VESAD tactile          VESAD                   Performance
12  IHM      Téléphone  VESAD tactile                    Préférence
13  IHM      Téléphone          VESAD                    Préférence
14  IHM  VESAD tactile          VESAD                    Préférence

    Différence des moyennes  Différence des moyennes (%)  Mann-Whitney U  \
0                  0.083333                     1.694915            66.0
1                  0.750000                    17.647059            36.0
2                  0.666667                    15.686275            40.5
3                 -1.166667                   -25.925926            23.0
4                 -1.083333                   -24.528302            28.5
5                  0.083333                     1.886792            69.5
6                 -0.500000                   -11.538462            70.5
7                  0.750000                    24.324324            48.5
8                  1.250000                    40.540541            21.0
9                 -1.166667                   -25.000000            22.5
10                 0.333333                    10.526316            57.0
11                 1.500000                    47.368421            17.5
12                 1.166667                    87.500000            16.0
13                 0.333333                    15.384615            56.0
14                -0.833333                   -38.461538            32.0

    Valeur p
0   0.215732
1   0.008605
2   0.020977
3   0.005031
4   0.010151
5   0.475027
6   0.475027
7   0.126838
8   0.004655
9   0.004655
10  0.215732
11  0.003904
12  0.003904
13  0.215732
14  0.013011
```

We display the geometric mean and 95% CI of each question:

```
In [35]: ranks.groupby([ivs['technique']['label']]).aggregate(print_geo_mean_ci)\
```

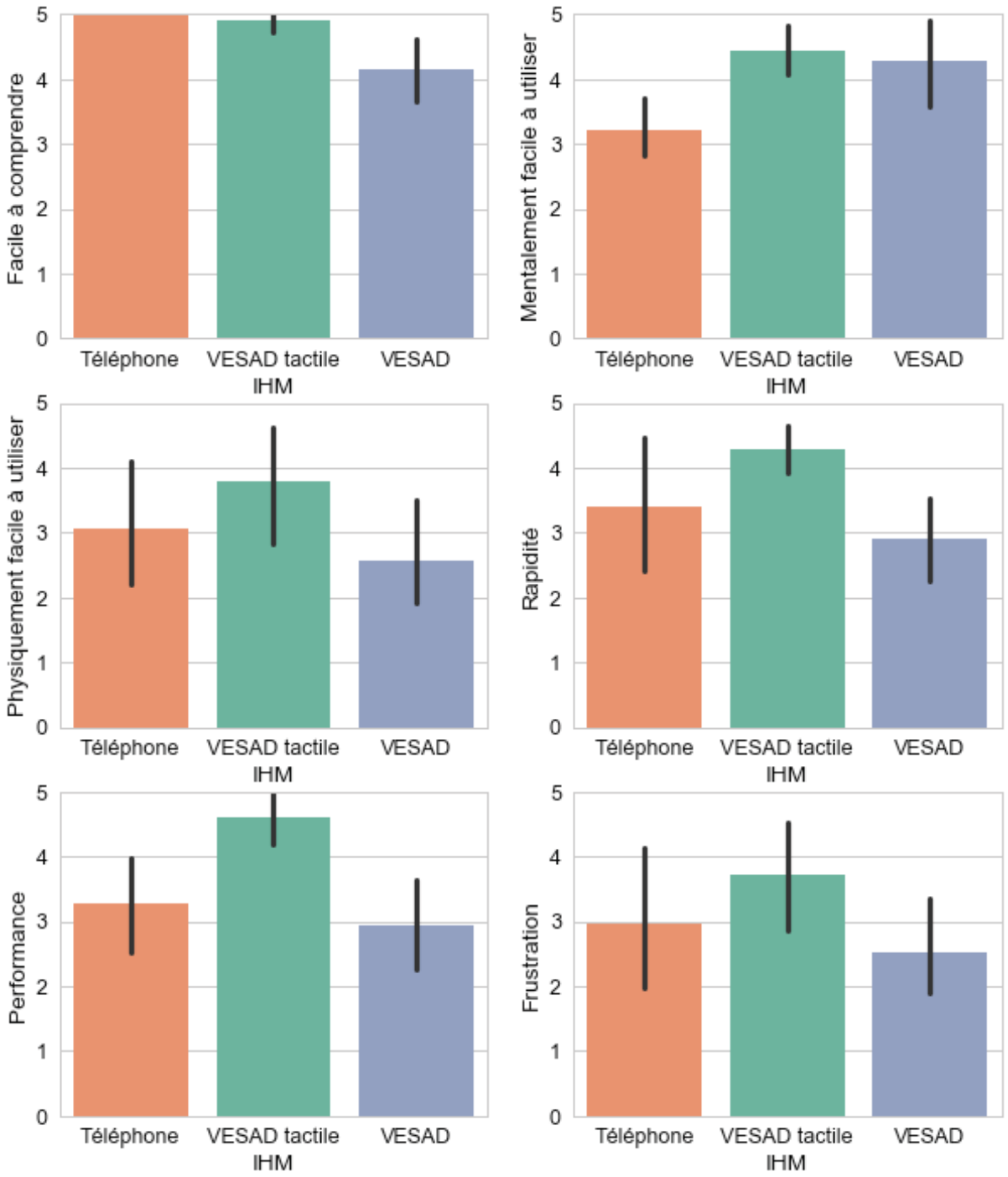
```
            .loc[:, ranks_dvs.loc['label', :]].transpose()
```

Out[35]:

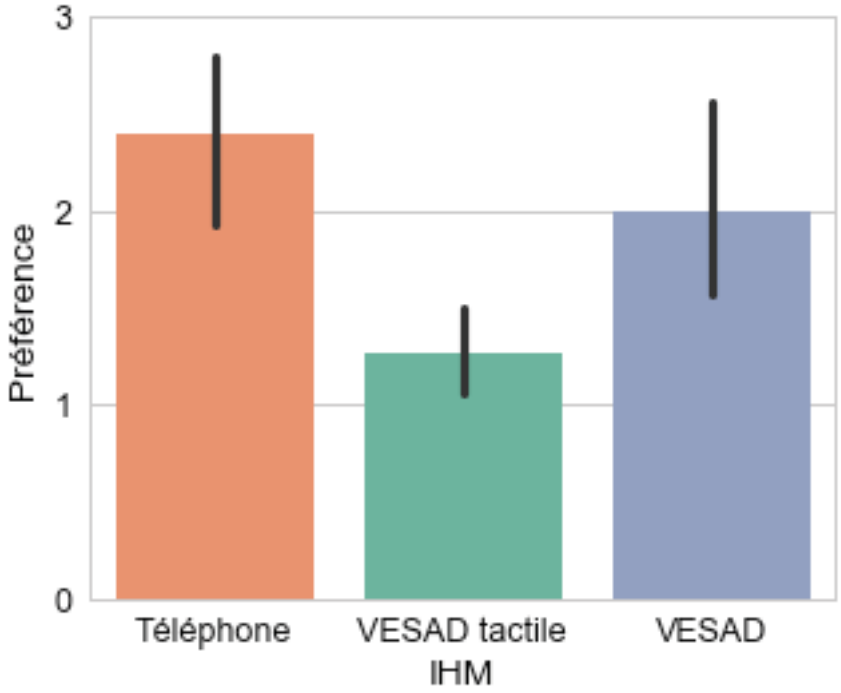
IHM		Téléphone	VESAD tactile	\
Facile à comprendre	5.00 [5.00, 5.00]	4.91 [4.73, 5.00]		
Mentalement facile à utiliser	3.22 [2.77, 3.68]	4.45 [4.03, 4.82]		
Physiquement facile à utiliser	3.06 [2.16, 4.16]	3.80 [2.75, 4.62]		
Rapidité	3.41 [2.42, 4.55]	4.29 [3.94, 4.64]		
Performance	3.27 [2.51, 3.98]	4.62 [4.16, 5.00]		
Frustration	2.96 [2.02, 4.14]	3.73 [2.71, 4.56]		
Préférence	2.39 [1.95, 2.80]	1.26 [1.06, 1.50]		

IHM	VESAD
Facile à comprendre	4.16 [3.73, 4.70]
Mentalement facile à utiliser	4.28 [3.60, 4.91]
Physiquement facile à utiliser	2.58 [1.88, 3.42]
Rapidité	2.90 [2.29, 3.51]
Performance	2.94 [2.27, 3.61]
Frustration	2.53 [1.88, 3.31]
Préférence	1.99 [1.55, 2.47]

```
In [36]: (fig, axs) = plot_ranks('technique', ranks_dvs.columns[0:-1],
                                 estimator=geometric_mean)
         fig.savefig('ranks.png')
```




```
In [37]: (fig, axs) = plot_ranks('technique', ['preference'],
                                 estimator=geometric_mean)
         fig.savefig('preferences.png')
```



Overall significant results are:

- **Easy to Understand**: *PhoneOnly* is significantly better than *MidAirInArOut* ($p = 0.009$), and seems a little better than *PhoneInArOut*.
- **Physically Easy to Use**: There is no significant differences due to TECHNIQUE; they seem scored similar.
- **Mentally Easy to Use**: *PhoneOnly* is statistically and practically worst than *PhoneInArOut* ($p = 0.005$) and *MidAirInArOut* ($p = 0.01$).
- **Subjective Speed**: *PhoneInArOut* is significantly better than *MidAirInArOut* ($p = 0.05$).
- **Subjective Performance**: *PhoneInArOut* is statistically better than *PhoneOnly* ($p = 0.005$) and *MidAirInArOut* ($p = 0.004$).
- **Frustration**: There is no significant differences due to TECHNIQUE; they seem scored similar.
- **Preference**: *PhoneInArOut* is significantly prefered to *PhoneOnly* ($p = 0.004$) and *MidAirInArOut* ($p = 0.01$).

0.3 3. Participant trials

Some functions for the analysis:

```
In [38]: def trial_samples(iv_id, dv_id, data=trials):
             iv, dv = ivs[iv_id], trials_dvs[dv_id]
             return [data[data[iv['label']] == iv_value][dv]\
                     for iv_value in iv['categorical']]
```

```
In [39]: def trial_means(iv_ids, dv_ids, data=trials, aggregate=print_mean_ci):
             iv_labels = [ivs.at['label', iv_id] for iv_id in iv_ids]
             dv_labels = [trials_dvs[dv_id] for dv_id in dv_ids]
             return data[iv_labels + dv_labels].groupby(iv_labels)\
                                               .aggregate(aggregate)
```

```
In [40]: def melt_trials(value_vars, var_name, value_name, data=trials):
             return pd.melt(data, id_vars=ivs.loc['label', :],
                            value_vars=value_vars, var_name=var_name,
                            value_name=value_name)

In [41]: def test_normality(iv_ids, dv_ids, data=trials, **args):
             results = []

             for dv_id in dv_ids:
                 for iv_id in iv_ids:
                     iv = ivs[iv_id]
                     samples = trial_samples(iv_id, dv_id, data)
                     for iv_value, sample in zip(iv['categorical'], samples):
                         W, p = stats.shapiro(sample)
                         results.append([iv['label'], iv_value, trials_dvs[dv_id], W, p])

             results = pd.DataFrame(results, columns=[labels['iv'], labels['iv_value'],
                                                      labels['dv'], 'Shapiro W',
                                                      labels['p_value']])
             p_values_correction(results, **args)
             return results

In [42]: def test_equal_variances(iv_ids, dv_ids, data=trials, levene_center='mean',
                                  **args):
             results = []

             for dv_id in dv_ids:
                 for iv_id in iv_ids:
                     samples = trial_samples(iv_id, dv_id, data)
                     W, p = stats.levene(*samples)
                     results.append([ivs[iv_id]['label'], trials_dvs[dv_id], W, p])

             results = pd.DataFrame(results, columns=[labels['iv'], labels['dv'],
                                                      'Levene W', labels['p_value']])
             p_values_correction(results, **args)
             return results

In [43]: def test_pairwise_trials(iv_id, dv_id, data=trials, log_data=False, **args):
             results = []
             iv, dv = ivs[iv_id], trials_dvs[dv_id]
             iv_category_ids = range(len(iv['categorical']))

             samples = trial_samples(iv_id, dv_id, data)
             sample_pairs = itertools.combinations(iv_category_ids, 2)
             for id1, id2 in sample_pairs:
                 T, p = stats.ttest_ind(samples[id1], samples[id2])

                 mean_diff, mean_diff_per = mean_difference(np.exp(samples[id1]),
                                                            np.exp(samples[id2]))\
                     if log_data else mean_difference(samples[id1], samples[id2])
```

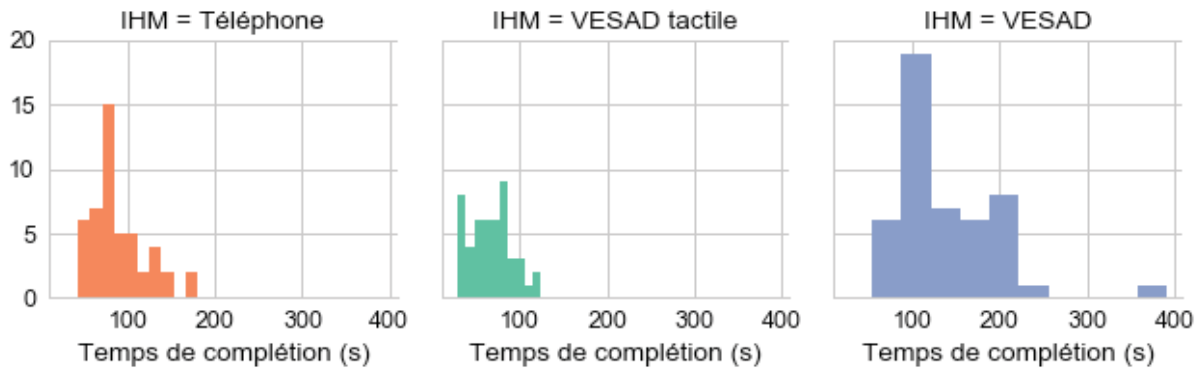
```
                results.append([iv['label'], iv['categorical'][id1],
                                iv['categorical'][id2], dv, mean_diff,
                                mean_diff_per, T, p])

        columns = [labels['iv'], labels['iv_value'] + ' 1',
                   labels['iv_value'] + ' 2', labels['dv'],
                   labels['mean_difference'], labels['mean_difference_percentage'],
                   labels['t_statistic'], labels['p_value']]
        results = pd.DataFrame(results, columns=columns)

        p_values_correction(results, **args)
        return results

In [44]: def test_non_normal_trials(iv_ids, dv_ids, data=trials, **args):
        results = []

        for dv_id in dv_ids:
            for iv_id in iv_ids:
                samples = trial_samples(iv_id, dv_id, data)
                H, p = stats.kruskal(*samples)
                results.append([ivs[iv_id]['label'], trials_dvs[dv_id], H, p])

        results = pd.DataFrame(results, columns=[labels['iv'], labels['dv'],
                                                 'Kruskal-Wallis H',
                                                 labels['p_value']])
        p_values_correction(results, **args)
        return results

In [45]: def test_pairwise_non_normal_trials(iv_ids, dv_ids, data=trials, **args):
        results = []

        for dv_id in dv_ids:
            for iv_id in iv_ids:
                iv, dv = ivs[iv_id], trials_dvs[dv_id]
                samples = trial_samples(iv_id, dv_id, data)

                iv_category_ids = range(len(iv['categorical']))
                sample_pairs = itertools.combinations(iv_category_ids, 2)
                for id1, id2 in sample_pairs:
                    U, p = stats.mannwhitneyu(samples[id1], samples[id2])
                    mean_diff, mean_diff_per = mean_difference(samples[id1],
                                                               samples[id2])
                    results.append([iv['label'], iv['categorical'][id1],
                                    iv['categorical'][id2], dv, mean_diff,
                                    mean_diff_per, U, p])

        columns = [labels['iv'], labels['iv_value'] + ' 1',
                   labels['iv_value'] + ' 2', labels['dv'],
                   labels['mean_difference'], labels['mean_difference_percentage'],
                   'Mann-Whitney U', labels['p_value']]
        results = pd.DataFrame(results, columns=columns)
```

```
            p_values_correction(results, **args)
            return results

In [46]: def plot_trials(iv_ids_list, dv_id, data=trials, kind='bar', **args):
            dv = trials_dvs[dv_id]
            if (len(iv_ids_list) == 0):
                iv_ids_list = [[iv_id] for id_id in ivs.columns]

            fig, axs = subplots(len(iv_ids_list))
            for id_ids, ax in zip(iv_ids_list, axs):

                iv_ids = [ivs[id_id] for id_id in id_ids]
                if (len(iv_ids) == 1):
                    if (kind == 'bar'):
                        sns.barplot(x=iv_ids[0]['label'], y=dv, data=data,
                                    palette=iv_ids[0]['palette'], ax=ax, **args)

                    elif (kind == 'box'):
                        sns.boxplot(x=iv_ids[0]['label'], y=dv, data=data,
                                    palette=iv_ids[0]['palette'], ax=ax, **args)

                    elif (kind == 'count'):
                        sns.countplot(hue=iv_ids[0]['label'], x=dv, data=data,
                                      palette=iv_ids[0]['palette'], ax=ax, **args)
                        ax.set(ylabel='Count')
                        ax.legend(loc='upper right', title=labels['count'],
                                  frameon=True)

                elif (len(ivs) == 2):
                    if (kind == 'bar'):
                        sns.barplot(x=iv_ids[1]['label'], y=dv, hue=iv_ids[0]['label'],
                                    data=data, palette=iv_ids[0]['palette'], ax=ax,
                                    **args)
                        ax.legend(frameon=True, loc='upper left', bbox_to_anchor=(1, 1))

            return (fig, axs)
```

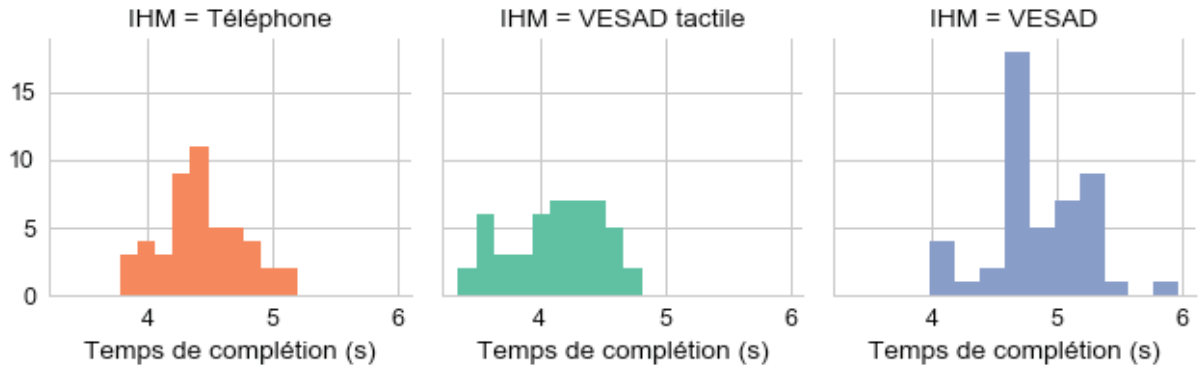
0.3.1 3.1. Task completion time

We first apply a log transform to TCT to approximate a normal distribution.

```
In [47]: g = sns.FacetGrid(trials, col=technique['label'], hue=technique['label'],
                           palette=technique['palette'])
         g = g.map(plt.hist, total_time)
         g.savefig('tct_distributions.png')
```




```
In [48]: trials[total_time] = np.log(trials[total_time])
         raw_trials['total_time'] = np.log(raw_trials['total_time'])

In [49]: g = sns.FacetGrid(trials, col=technique['label'], hue=technique['label'],
                           palette=technique['palette'])
         g = g.map(plt.hist, total_time)
         g.savefig('tct_distributions_log.png')
```



We test the normality of TCT distributions for each TECHNIQUE and their equality of variances, since it's the main factor of interest.

```
In [50]: test_normality(['technique'], ['total_time'])

Out[50]:     VI      Valeur VI                      VD  Shapiro W  Valeur p
         0  IHM      Téléphone  Temps de complétion (s)   0.983679  0.735995
         1  IHM  VESAD tactile  Temps de complétion (s)   0.971828  0.446912
         2  IHM          VESAD  Temps de complétion (s)   0.966510  0.446912

In [51]: test_equal_variances(['technique'], ['total_time'])

Out[51]:     VI                      VD  Levene W  Valeur p
         0  IHM  Temps de complétion (s)  0.756269  0.471309
```

We meet all the assumptions of an ANOVA. Trials were done independently, TCT distributions are normal and their variances are equal.

We perform a full factorial ANOVA with the model: TCT ~ TECHNIQUE x TEXT_SIZE x DISTANCE + TECHNIQUE x ORDERING.

```
In [52]: tct_model = ols('total_time ~ technique * text_size * distance'
                         + '+ technique * ordering', data=raw_trials).fit()
         sm.stats.anova_lm(tct_model, typ=2)
```

Out[52]:

	sum_sq	df	F	PR(>F)
technique	12.328279	2.0	62.210952	1.610604e-19
text_size	0.145175	1.0	1.465169	2.283752e-01
distance	0.029459	1.0	0.297314	5.865353e-01
ordering	2.133256	2.0	10.764832	4.829976e-05
technique:text_size	0.398420	2.0	2.010508	1.381943e-01
technique:distance	0.074854	2.0	0.377730	6.861892e-01
text_size:distance	0.291976	1.0	2.946742	8.850822e-02
technique:ordering	3.086868	4.0	7.788476	1.215733e-05
technique:text_size:distance	0.499563	2.0	2.520896	8.443561e-02
Residual	12.484644	126.0	NaN	NaN

The main significant effect on TCT is TECHNIQUE ($F = 62.2$, $p < 0.0001$), then ORDERING ($F = 10.8$, $p < 0.0001$). There is also an significant interaction effect: TECHNIQUE x ORDERING ($F = 2.5$, $p < 0.0001$).

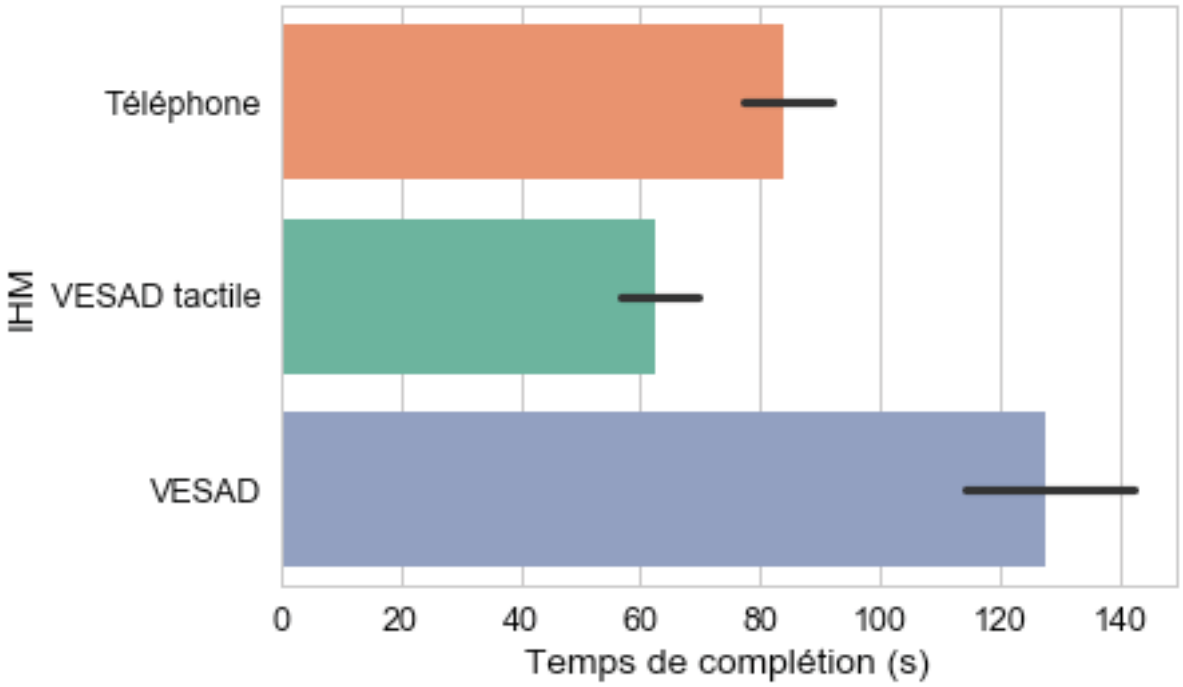
We display mean TCT values with 95% CI for these conditions:

```
In [53]: trial_means(['technique'], ['total_time'], aggregate=print_exp_mean_ci)
```

Out[53]:

	Temps de complétion (s)
IHM	
Téléphone	84.02 [76.81, 92.76]
VESAD tactile	62.59 [56.06, 69.33]
VESAD	127.71 [114.14, 141.96]

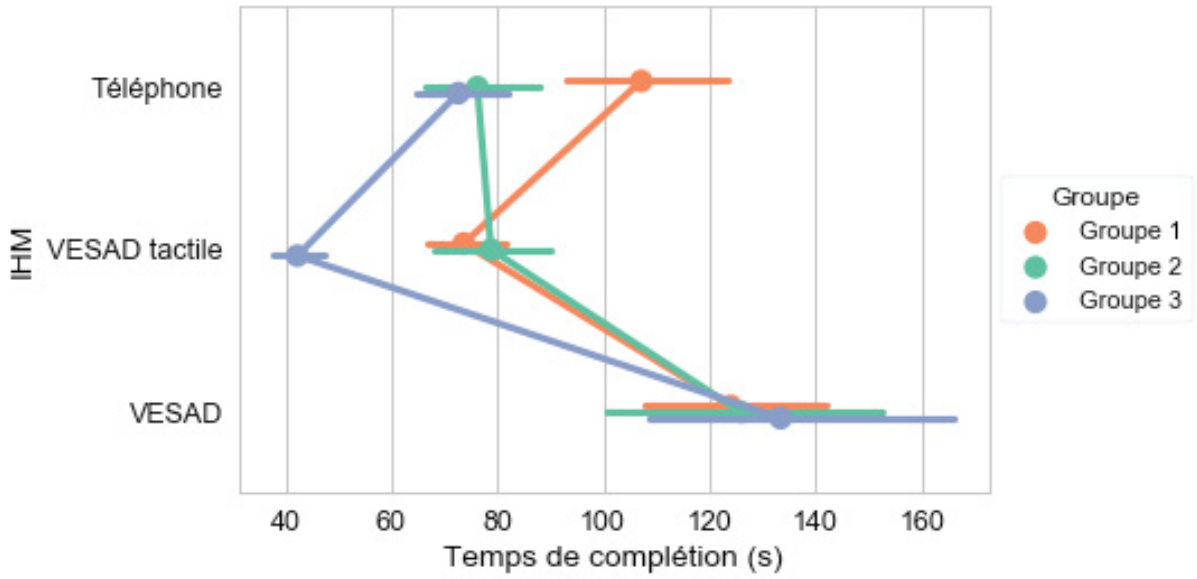
```
In [54]: ax = sns.barplot(x=trials_dvs['total_time'], y=technique['label'],
                          palette=technique['palette'], data=trials, estimator=exp_mean)
         ax.get_figure().savefig('tct.png')
```




```
In [55]: trial_means(['ordering', 'technique'], ['total_time'],
                    aggregate=print_exp_mean_ci).unstack()
```

```
Out[55]:          Temps de complétion (s)                          \
         IHM                     Téléphone         VESAD tactile
         Groupe
         Groupe 1  107.14 [92.23, 123.88]  73.63 [67.26, 81.12]
         Groupe 2    76.22 [66.31, 87.24]  78.79 [68.15, 89.81]
         Groupe 3    72.64 [64.96, 82.05]  42.27 [37.99, 47.55]


         IHM                          VESAD
         Groupe
         Groupe 1  124.00 [108.09, 141.20]
         Groupe 2  125.91 [101.42, 152.20]
         Groupe 3  133.41 [108.95, 169.41]
```

```
In [56]: ax = sns.pointplot(x=total_time, y=technique['label'], hue=ordering['label'],
                            palette=ordering['palette'], data=trials, dodge=True,
                            estimator=exp_mean)
         config_legend(ax, 'ordering')
         ax.get_figure().savefig('tct_ordering.png')
```



It seems that participants who started with *PhoneOnly* were slower with this technique than the the other groups. Similarly, participants who finished with *PhoneInArOut* were faster with this technique. It indicates there is a learning curve on the task, but interestingly participants from all groups performed equally with *MidAirInArOut* technique.

We compare the TCT for the three techniques only with pairwise t-tests (Benjamini–Hochberg correction) .

```
In [57]: test_pairwise_trials('technique', 'total_time', correction_method='fdr_bh',
                               log_data=True)
```

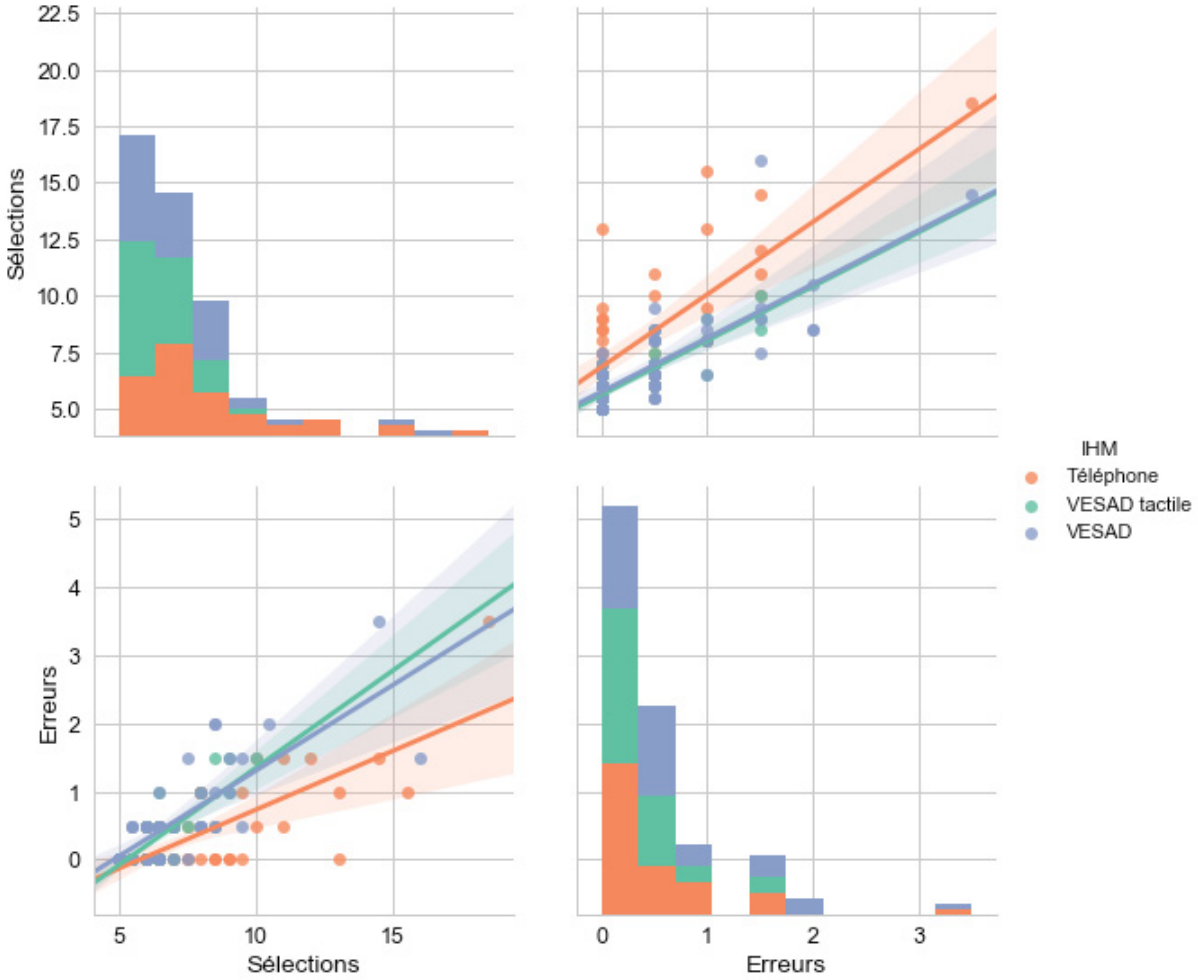
```
Out[57]:     VI    Valeur VI 1    Valeur VI 2                      VD  \
         0  IHM      Téléphone  VESAD tactile  Temps de complétion (s)
         1  IHM      Téléphone          VESAD  Temps de complétion (s)
         2  IHM  VESAD tactile          VESAD  Temps de complétion (s)

            Différence des moyennes  Différence des moyennes (%)  Statistique T  \
         0                21.840362                    32.666819       4.090635
         1               -49.467731                   -35.803106      -5.615875
         2               -71.308093                   -51.610437      -9.032751

                Valeur p
         0  9.077859e-05
         1  2.966612e-07
         2  6.282833e-14
```

- *PhoneInArOut* is 71*s* (+52%) faster than *MidAirInArOut* ($p < 0.0001$).
- *PhoneOnly* is 49*s* (+36%) faster than *MidAirInArOut* ($p < 0.0001$).
- *PhoneInArOut* is 22*s* (+33%) faster than *PhoneOnly* ($p < 0.0001$).

0.3.2 3.2. Errors

We visualize first the SELECTIONS and ERRORS distributions:

```
In [58]: g = sns.pairplot(trials, hue=technique['label'], kind='reg',
                          vars=[trials_dvs['selections_count'], trials_dvs['errors']],
                          palette=technique['palette'], size=4);
         g.savefig('selections_errors_distributions.png')
```



It seems that a user makes as much errors as she makes selections. The relation is almost the same for each technique, even if users seems to make more selections for the same number of errors with PhoneOnly.

We can't use ANOVA on SELECTIONS and ERRORS variables as their distributions are exponentials. We use instead the Kruskal-Wallis test (Benjamini–Hochberg correction) to check if there is significative differences due to TECHNIQUE, TEXT_SIZE, DISTANCE or ORDERING.

```
In [59]: test_non_normal_trials(['technique', 'text_size', 'distance', 'ordering'],
                               ['selections_count', 'errors'])
```

Out[59]:

	VI	VD	Kruskal-Wallis H	Valeur p
0	IHM	Sélections	15.125918	0.004155
1	Taille du texte	Sélections	0.003897	0.950224
2	Distance	Sélections	0.052701	0.935347
3	Groupe	Sélections	11.510535	0.010095

```
4              IHM     Erreurs          5.201541  0.148433
5  Taille du texte     Erreurs          0.103661  0.935347
6         Distance     Erreurs          0.201599  0.935347
7           Groupe     Erreurs         11.153096  0.010095
```

Only TECHNIQUE ($p = 0.004$) and ORDERING ($p = 0.01$) have a significant effect on SELECTIONS. But, only ORDERING ($p = 0.01$) have a significant effect on ERRORS.

We use then pairwise Mann-Whitney tests (Benjamini–Hochberg correction) for the significant questions above:

```
In [60]: test_pairwise_non_normal_trials(['technique', 'ordering'],
                                         ['selections_count', 'errors'])
```

Out[60]:

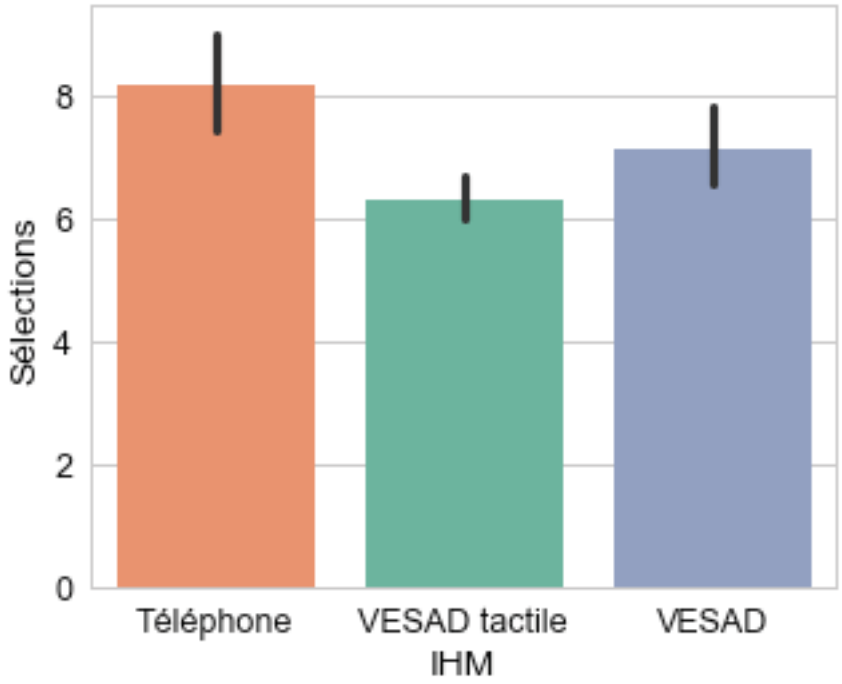
	VI	Valeur VI 1	Valeur VI 2	VD	Différence des moyennes	Différence des moyennes (%)	Mann-Whitney U	Valeur p
0	IHM	Téléphone	VESAD tactile	Sélections	1.864583	29.537954	627.0	0.000635
1	IHM	Téléphone	VESAD	Sélections	1.041667	14.598540	864.5	0.029356
2	IHM	VESAD tactile	VESAD	Sélections	-0.822917	-11.532847	915.0	0.059881
3	Groupe	Groupe 1	Groupe 2	Sélections	1.729167	26.265823	755.0	0.005196
4	Groupe	Groupe 1	Groupe 3	Sélections	1.583333	23.529412	755.0	0.005196
5	Groupe	Groupe 2	Groupe 3	Sélections	-0.145833	-2.167183	1136.0	0.454285
6	IHM	Téléphone	VESAD tactile	Erreurs	0.125000	42.857143	1083.5	0.312007
7	IHM	Téléphone	VESAD	Erreurs	-0.166667	-28.571429	957.0	0.083050
8	IHM	VESAD tactile	VESAD	Erreurs	-0.291667	-50.000000	873.0	0.026306
9	Groupe	Groupe 1	Groupe 2	Erreurs	0.333333	103.225806	846.5	0.020183
10	Groupe	Groupe 1	Groupe 3	Erreurs	0.343750	110.000000	761.5	0.005196
11	Groupe	Groupe 2	Groupe 3	Erreurs	0.010417	3.333333	1039.0	0.205336

```
In [61]: trial_means(['technique'], ['selections_count', 'errors'])
```

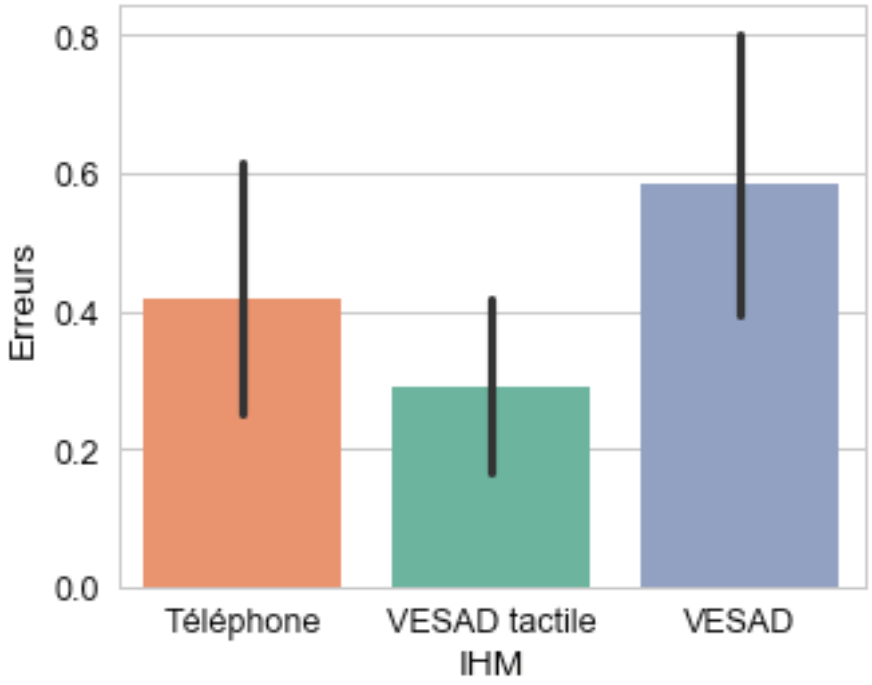
Out[61]:

IHM	Sélections	Erreurs
Téléphone	8.18 [7.42, 8.99]	0.42 [0.24, 0.59]
VESAD tactile	6.31 [5.97, 6.70]	0.29 [0.18, 0.42]
VESAD	7.14 [6.54, 7.86]	0.58 [0.39, 0.80]

```
In [62]: (fig, axs) = plot_trials([['technique']], 'selections_count')
         fig.savefig('selections.png')
```




```
In [63]: (fig, axs) = plot_trials([['technique']], 'errors')
         fig.savefig('errors.png')
```

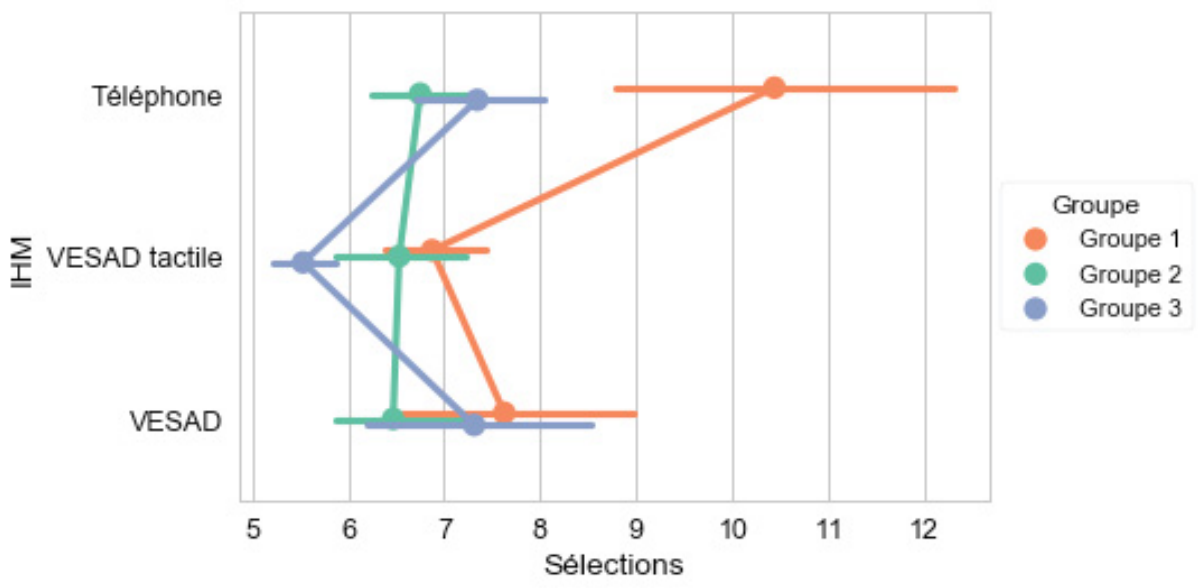



```
In [64]: trial_means(['ordering', 'technique'], ['selections_count', 'errors'])
```

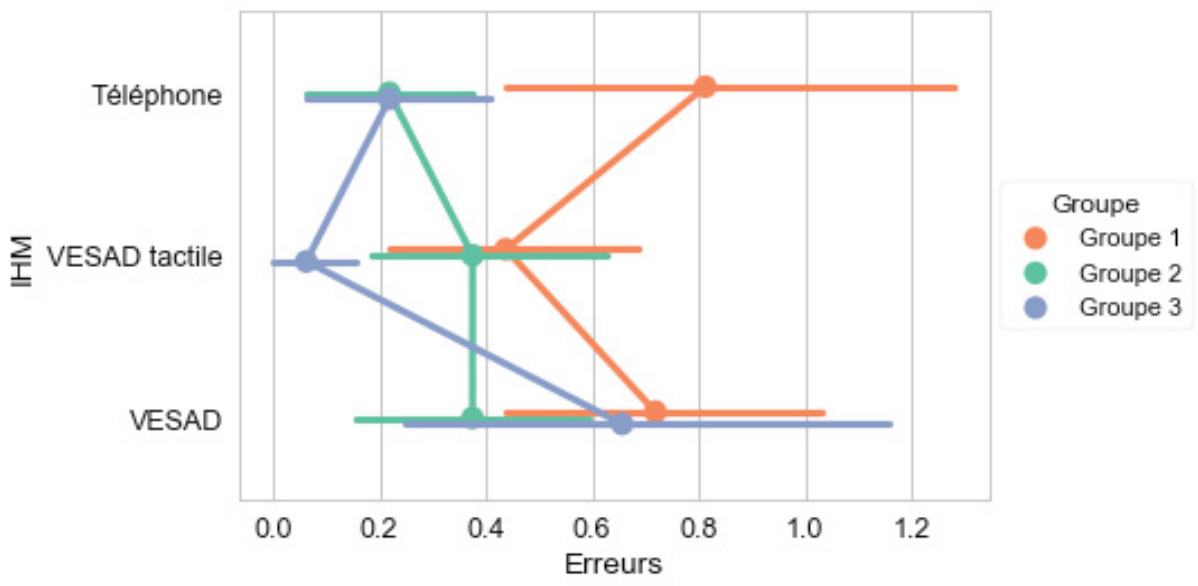
Out[64]:

Groupe	IHM	Sélections	Erreurs
Groupe 1	Téléphone	10.44 [8.72, 12.31]	0.81 [0.38, 1.28]
	VESAD tactile	6.88 [6.38, 7.50]	0.44 [0.22, 0.69]
	VESAD	7.62 [6.62, 9.12]	0.72 [0.41, 1.06]
Groupe 2	Téléphone	6.75 [6.25, 7.31]	0.22 [0.06, 0.41]
	VESAD tactile	6.53 [5.87, 7.22]	0.38 [0.16, 0.59]
	VESAD	6.47 [5.91, 7.13]	0.38 [0.19, 0.62]
Groupe 3	Téléphone	7.34 [6.72, 8.06]	0.22 [0.06, 0.38]
	VESAD tactile	5.53 [5.25, 5.84]	0.06 [0.00, 0.16]
	VESAD	7.31 [6.28, 8.69]	0.66 [0.25, 1.13]

In [65]:
```
ax = sns.pointplot(x=trials_dvs['selections_count'], y=technique['label'],
                   hue=ordering['label'], palette=ordering['palette'],
                   data=trials, dodge=True)
config_legend(ax, 'ordering')
ax.get_figure().savefig('selections_ordering.png')
```



In [66]:
```
ax = sns.pointplot(x=trials_dvs['errors'], y=technique['label'],
                   hue=ordering['label'], palette=ordering['palette'],
                   data=trials, dodge=True)
config_legend(ax, 'ordering')
ax.get_figure().savefig('errors_ordering.png')
```



0.3.3 3.3. Navigation

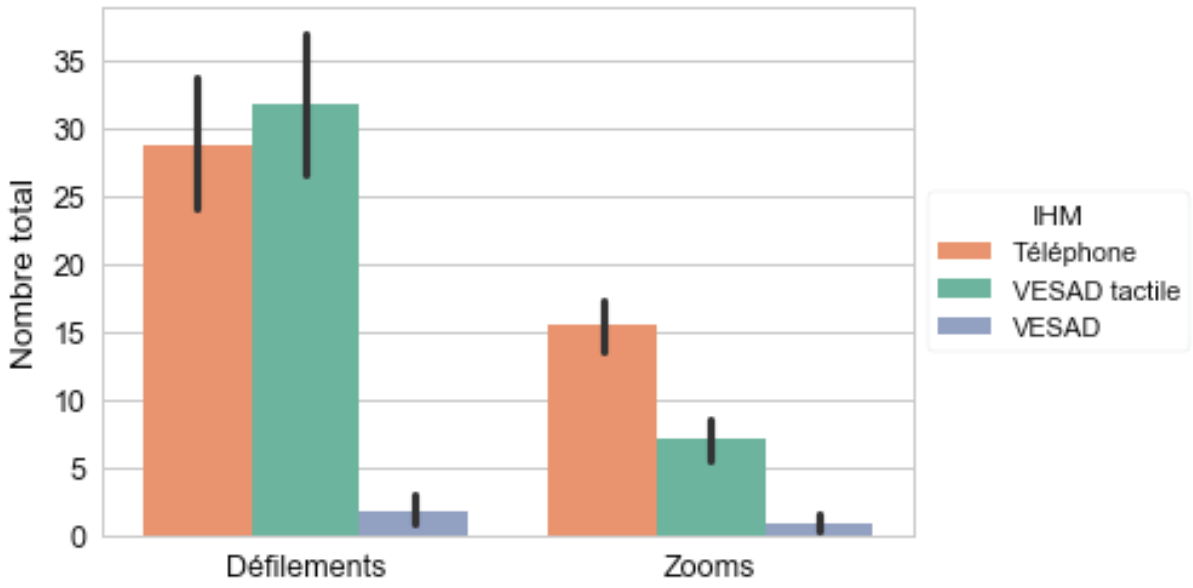
Variable meanings:

- Selection Time = time spent looking for where to drop an item that had been picked
- Selection Distance = distance travelled by the finger with an item selected
- Head Phone Distance = sum of the distance between the head and the phone

```
In [67]: # Data preparation
         trial_counts = melt_trials(var_name=labels['category'],
                                    value_name=labels['count'],
                                    value_vars=[trials_dvs['pan_count'],
                                                trials_dvs['zoom_count']])

         trial_times = melt_trials(var_name=labels['category'],
                                   value_name=labels['time'],
                                   value_vars=[trials_dvs['selections_time'],
                                               trials_dvs['pan_time'],
                                               trials_dvs['zoom_time']])

         trial_distances_dvs = [trials_dvs['selections_projected_distance'],
                                trials_dvs['pan_projected_distance'],
                                trials_dvs['zoom_projected_distance'],
                                trials_dvs['absolute_head_phone_distance']]
         trial_distances = melt_trials(var_name=labels['category'],
                                       value_name=labels['distance'],
                                       value_vars=trial_distances_dvs)

In [68]: trial_means(['technique'], ['pan_count', 'zoom_count'])

Out[68]:                          Défilements                Zooms
         IHM
```

```
Téléphone      28.70 [24.34, 33.28]  15.53 [13.62, 17.53]
VESAD tactile  31.81 [26.62, 37.60]     7.14 [5.72, 8.81]
VESAD             1.83 [0.81, 3.08]     0.83 [0.27, 1.53]
```

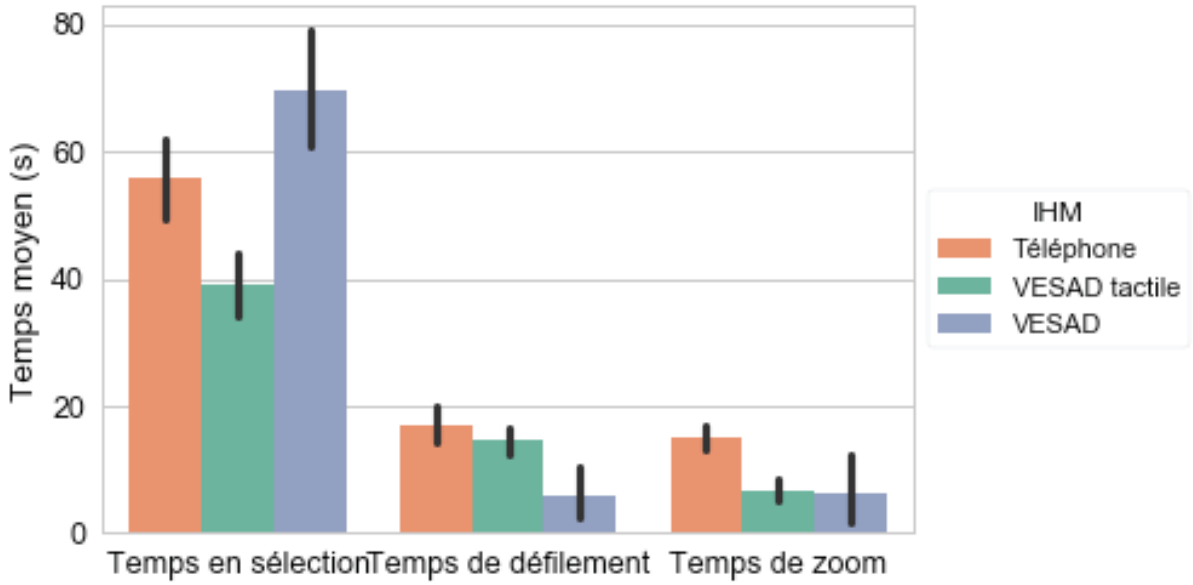
```
In [69]: ax = sns.barplot(x=labels['category'], y=labels['count'],
                          hue=technique['label'], palette=technique['palette'],
                          data=trial_counts)
         config_legend(ax, 'technique')
         ax.set(xlabel='')
         ax.get_figure().savefig('navigation_count.png')
```




```
In [70]: trial_means(['technique'], ['selections_time', 'pan_time', 'zoom_time'])
```

```
Out[70]:                  Temps en sélection   Temps de défilement  \
         IHM
         Téléphone      55.74 [50.20, 61.79]  17.12 [14.48, 19.97]
         VESAD tactile  38.98 [34.32, 43.80]  14.53 [12.51, 16.50]
         VESAD          69.68 [60.35, 79.78]    5.97 [2.15, 10.07]

                              Temps de zoom
         IHM
         Téléphone      15.18 [13.31, 17.18]
         VESAD tactile     6.70 [5.11, 8.62]
         VESAD             6.24 [1.67, 11.87]
```

```
In [71]: ax = sns.barplot(x=labels['category'], y=labels['time'], data=trial_times,
                          hue=technique['label'], palette=technique['palette'])
         config_legend(ax, 'technique')
         ax.set(xlabel='')
         ax.get_figure().savefig('navigation_time.png')
```



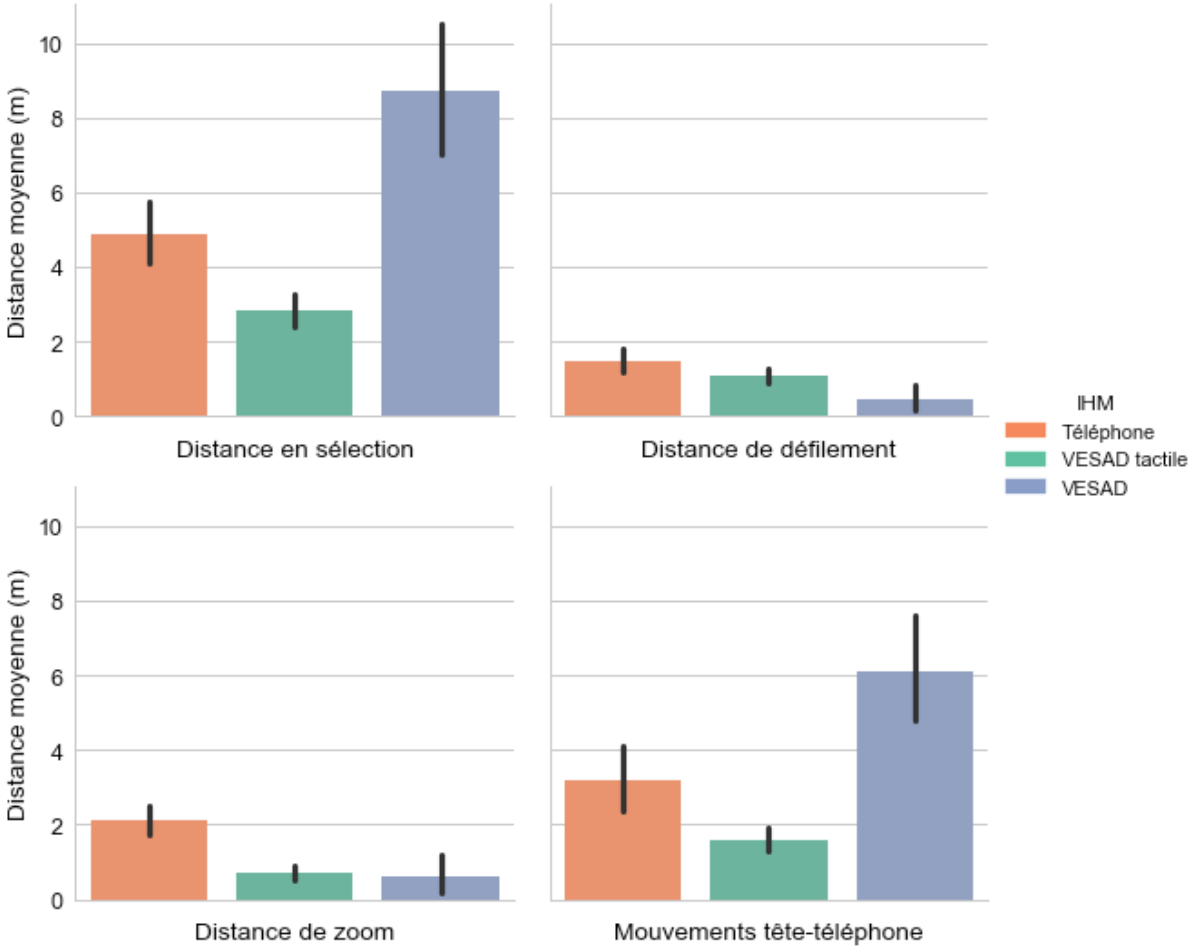

```
In [72]: trial_means(['technique'], ['selections_projected_distance',
                                    'pan_projected_distance',
                                    'zoom_projected_distance',
                                    'absolute_head_phone_distance'])
```

Out[72]:

IHM	Distance en sélection	Distance de défilement	Distance de zoom	Mouvements tête-téléphone
Téléphone	4.87 [4.09, 5.68]	1.50 [1.20, 1.82]	2.11 [1.75, 2.55]	3.18 [2.34, 4.12]
VESAD tactile	2.83 [2.40, 3.28]	1.09 [0.91, 1.29]	0.69 [0.51, 0.89]	1.57 [1.27, 1.91]
VESAD	8.72 [7.19, 10.53]	0.47 [0.18, 0.86]	0.62 [0.15, 1.19]	6.12 [4.93, 7.43]

```
In [73]: g = sns.factorplot(x=technique['label'], y=labels['distance'],
                           col=labels['category'], data=trial_distances,
                           palette=technique['palette'], kind='bar', col_wrap=2)

         g.set_titles('{col_name}') # Replace subplot titles
         for ax in g.axes:
             ax.title.set_position([0.5, -0.12])

         g.set_axis_labels('') # Custom legend
         g.set_xticklabels([])
         legend_handles = [patches.Patch(color=color, label=value)\
                           for value, color in zip(technique['categorical'],
                                                   technique['palette'])]
```

```
legend = plt.legend(handles=legend_handles, loc='center left',
                    bbox_to_anchor=(1, 1.1), title=technique['label'])
fix_legend_fontsize(legend)

g.savefig('navigation_distance.png')
```



BIBLIOGRAPHIE

Adcock, M., Hutchins, M. & Gunn, C. (2003). Augmented reality haptics : Using ARToolKit for display of haptic applications. Dans *Augmented Reality Toolkit Workshop, 2003* (pp. 1–2). Los Alamitos, CA, USA : IEEE.

Andrews, C., Endert, A. & North, C. (2010). Space to think : large high-resolution displays for sensemaking. Dans *Proceedings of the 28th International Conference on Human Factors in Computing Systems, CHI 2010, Atlanta, GA, USA, April 10-15, 2010* (pp. 55–64). New York, NY, USA : ACM. doi : 10.1145/1753326.1753336.

Argelaguet, F. & Andujar, C. (2013). A survey of 3D object selection techniques for virtual environments. *Computers & Graphics*, 37(3), 121–136. doi : 10.1016/j.cag.2012.12.003.

Azuma, R. (1997). A survey of augmented reality. *Presence : Teleoperators and Virtual Environments*, 6(4), 355–385. doi : 10.1162/pres.1997.6.4.355.

Azuma, R., Baillot, Y., Behringer, R., Feiner, S., Julier, S. & MacIntyre, B. (2001). Recent advances in augmented reality. *IEEE Computer Graphics and Applications*, 21(6), 34–47. doi : 10.1109/38.963459.

Bababekova, Y., Rosenfield, M., Hue, J. E. & Huang, R. R. (2011). Font size and viewing distance of handheld smart phones. *Optometry and Vision Science*, 88(7), 795–797. doi : 10.1097/opx.0b013e3182198792.

Baker, M. J. (1998). Maths - AxisAngle to Quaternion. Repéré à http://www.euclideanspace.com/maths/geometry/rotations/conversions/angleToQuaternion/.

Baudisch, P., Good, N., Bellotti, V. & Schraedley, P. K. (2002). Keeping things in context : A comparative evaluation of focus plus context screens, overviews, and zooming. Dans Wixon, D. R. (Éd.), *Proceedings of the 20th International Conference on Human Factors in Computing Systems, CHI 2002, Minneapolis, MN, USA, April 20-25, 2002* (pp. 259–266). New York, NY, USA : ACM. doi : 10.1145/503421.503423.

Benko, H., Ofek, E., Zheng, F. & Wilson, A. D. (2015). FoveAR : Combining an optically see-through near-eye display with projector-based spatial augmented reality. Dans Latulipe, C., Hartmann, B. & Grossman, T. (Éds.), *Proceedings of the 28th Annual ACM Symposium on User Interface Software and Technology, UIST 2015, Charlotte, NC, USA, November 8-11, 2015* (pp. 129–135). New York, NY, USA : ACM. doi : 10.1145/2807442.2807493.

Bergé, L., Serrano, M., Perelman, G. & Dubois, E. (2014). Exploring smartphone-based interaction with overview+detail interfaces on 3D public displays. Dans Quigley, A. J., Diamond, S., Irani, P. & Subramanian, S. (Éds.), *Proceedings of the 16th international conference on Human-computer interaction with mobile devices & services, MobileHCI 2014, Toronto, ON, Canada, September 23-26, 2014* (pp. 125–134). New York, NY, USA : ACM. doi : 10.1145/2628363.2628374.

Bi, X., Grossman, T., Matejka, J. & Fitzmaurice, G. W. (2011). Magic desk : Bringing multi-touch surfaces into desktop work. Dans Tan, D. S., Amershi, S., Begole, B., Kellogg, W. A. & Tungare, M. (Éds.), *Proceedings of the 29th International Conference on Human Factors in Computing Systems, CHI 2011, Vancouver, BC, Canada, May 7-12, 2011* (pp. 2511–2520). New York, NY, USA : ACM. doi : 10.1145/1978942.1979309.

Billinghurst, M., Grasset, R. & Looser, J. (2005). Designing augmented reality interfaces. *ACM SIGGRAPH Computer Graphics*, 39(1), 17–22. doi : 10.1145/1057792.1057803.

Billinghurst, M., Clark, A. J. & Lee, G. A. (2015). A survey of augmented reality. *Foundations and Trends in Human-Computer Interaction*, 8(2-3), 73–272. doi : 10.1561/1100000049.

Bimber, O. & Raskar, R. (2005). *Spatial augmented reality : Merging real and virtual worlds*. Boca Raton, FL : CRC Press.

Bourke, P. (1999). Calculating Stereo Pairs. Repéré à http://paulbourke.net/stereographics/stereorender/.

Bowman, D., Kruijff, E., J, J. J. L. & Poupyrev, I. P. (2004). *3D User interfaces : Theory and practice*. Redwood City, CA : Addison-Wesley.

Bowman, D. A., Kruijff, E., LaViola, J. J. & Poupyrev, I. (2001). An Introduction to 3-D User Interface Design. *Presence : Teleoperators and Virtual Environments*, 10(1), 96–108. doi : 10.1162/105474601750182342.

Bradski, G. & Kaehler, A. (2008). *Learning OpenCV : Computer vision with the OpenCV library*. Sebastopol, CA, USA : O'Reilly Media, Inc.

Burigat, S. & Chittaro, L. (2013). On the effectiveness of Overview+Detail visualization on mobile devices. *Personal and Ubiquitous Computing*, 17(2), 371–385. doi : 10.1007/s00779-011-0500-3.

Buxton, B. & Fitzmaurice, G. W. (1998). HMDs, caves & chameleon : A human-centric analysis of interaction in virtual space. *ACM SIGGRAPH Computer Graphics*, 32(4), 69–74. doi : 10.1145/307710.307732.

Cha, L., Kao, H., Chen, M. Y., Lee, M., Hsu, J. Y. & Hung, Y. (2010). Touching the void : Direct-touch interaction for intangible displays. Dans Mynatt, E. D., Schoner, D., Fitzpatrick, G., Hudson, S. E., Edwards, W. K. & Rodden, T. (Éds.), *Proceedings of the 28th International Conference on Human Factors in Computing Systems, CHI 2010, Atlanta, GA, USA, April 10-15, 2010* (pp. 2625–2634). New York, NY, USA : ACM. doi : 10.1145/1753326.1753725.

Chaffey, D. (2018, janvier, 30). Mobile Marketing Statistics compilation [Billet de blogue]. Repéré à https://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/.

Cockburn, A., Karlson, A. K. & Bederson, B. B. (2008). A review of overview+detail, zooming, and focus+context interfaces. *ACM Computing Surveys*, 41(1), 2 :1–2 :31. doi : 10.1145/1456650.1456652.

Czerwinski, M., Smith, G., Regan, T., Meyers, B., Robertson, G. G. & Starkweather, G. K. (2003). Toward Characterizing the Productivity Benefits of Very Large Displays. Dans Rauterberg, M., Menozzi, M. & Wesson, J. (Éds.), *INTERACT 2003 : 9th IFIP TC.13 International Conference on Human-Computer Interaction, Zurich, Switzerland, September 1-5, 2003* (pp. 9–16). Amsterdam, The Netherlands : IOS Press.

Dragicevic, P. (2016). Fair statistical communication in HCI. Dans *Modern Statistical Methods for HCI* (pp. 291–330). Springer. doi : 10.1007/978-3-319-26633-6_13.

Dünser, A., Grasset, R. & Billinghurst, M. (2008). *A survey of evaluation techniques used in augmented reality studies* (Rapport n°TR-2008-02). Christchurch : Human Interface Technology Laboratory New Zealand.

Ens, B. (2014, janvier, 30). The Personal Cockpit [Vidéo en ligne]. Repéré à https://www.youtube.com/watch?v=L0ZjmEP-c1E.

Ens, B., Hincapié-Ramos, J. D. & Irani, P. (2014a). Ethereal planes : A design framework for 2D information space in 3D mixed reality environments. Dans Wilson, A., Steinicke, F., Suma, E. A. & Stuerzlinger, W. (Éds.), *Proceedings of the 2nd ACM Symposium on Spatial User Interaction, SUI 2014, Honolulu, HI, USA, October 4-5, 2014* (pp. 2–12). New York, NY, USA : ACM. doi : 10.1145/2659766.2659769.

Ens, B. M., Finnegan, R. & Irani, P. P. (2014b). The personal cockpit : A spatial interface for effective task switching on head-worn displays. Dans Jones, M., Palanque, P. A., Schmidt, A. & Grossman, T. (Éds.), *Proceedings of the 32nd International Conference on Human Factors in Computing Systems, CHI 2014, Toronto, ON, Canada, April 26 - May 1, 2014* (pp. 3171–3180). New York, NY, USA : ACM. doi : 10.1145/2556288.2557058.

Evan-Amos. (2017). File :Oculus-Rift-Touch-Controllers-Pair.jpg. Repéré à https://en.wikipedia.org/wiki/File:Oculus-Rift-Touch-Controllers-Pair.jpg.

Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F. J. & Marín-Jiménez, M. J. (2014). Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6), 2280–2292. doi : 10.1016/j.patcog.2014.01.005.

Gartner. (2017, août, 15). Top Trends in the Gartner Hype Cycle for Emerging Technologies, 2017. Repéré à https://www.gartner.com/smarterwithgartner/top-trends-in-the-gartner-hype-cycle-for-emerging-technologies-2017/.

Gonzalez, D. (2017, octobre, 9). Looking to the future of mixed reality (Part III) [Billet de blogue]. Repéré à https://blogs.unity3d.com/2017/10/09/looking-to-the-future-of-mixed-reality-part-iii/.

Gonzalez, D., Alleyne, C., & Drouin, S. (2017, septembre, 21). Looking to the future of mixed reality (Part II) [Billet de blogue]. Repéré à https://blogs.unity3d.com/2017/09/21/looking-to-the-future-of-mixed-reality-part-ii/.

Grubert, J. (2015, février, 27). MultiFi Study Conditions [Vidéo en ligne]. Repéré à https://www.youtube.com/watch?v=k43S7ARjB5E.

Grubert, J., Heinisch, M., Quigley, A. J. & Schmalstieg, D. (2015). MultiFi : Multi fidelity interaction with displays on and around the body. Dans Begole, B., Kim, J., Inkpen, K. & Woo, W. (Éds.), *Proceedings of the 33rd International Conference on Human Factors in Computing Systems, CHI 2015, Seoul, Republic of Korea, April 18-23, 2015* (pp. 3933–3942). New York, NY, USA : ACM. doi : 10.1145/2702123.2702331.

Guiard, Y. & Beaudouin-Lafon, M. (2004). Target acquisition in multiscale electronic worlds. *International Journal of Human-Computer Studies*, 61(6), 875–905. doi : 10.1016/j.ijhcs.2004.09.005.

Heun, V., Hobin, J. & Maes, P. (2013). Reality Editor : Programming Smarter Objects. Dans Mattern, F., Santini, S., Canny, J. F., Langheinrich, M. & Rekimoto, J. (Éds.), *The 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing, UbiComp 2013, Zurich, Switzerland, September 8-12, 2013 - Adjunct Publication* (pp. 307–310). New York, NY, USA : ACM. doi : 10.1145/2494091.2494185.

Huang, Z., Hui, P., Peylo, C. & Chatzopoulos, D. (2013). Mobile augmented reality survey : A bottom-up approach. *CoRR*, abs/1309.4413. Repéré à http://arxiv.org/abs/1309.4413.

Jacob, R. J. K., Girouard, A., Hirshfield, L. M., Horn, M. S., Shaer, O., Solovey, E. T. & Zigelbaum, J. (2008). Reality-based interaction : A framework for post-WIMP interfaces. Dans Czerwinski, M., Lund, A. M. & Tan, D. S. (Éds.), *Proceedings of the 26th International Conference on Human Factors in Computing Systems, CHI 2008, Florence, Italy, April 5-10, 2008* (pp. 201–210). New York, NY, USA : ACM. doi : 10.1145/1357054.1357089.

Jankowski, J. & Hachet, M. (2015). Advances in interaction with 3D environments. *Computer Graphics Forum*, 34(1), 152–190. doi : 10.1111/cgf.12466.

Jones, B. R., Sodhi, R., Forsyth, D. A., Bailey, B. P. & Maciocci, G. (2012). Around device interaction for multiscale navigation. Dans Churchill, E. F., Subramanian, S., Baudisch, P. & O'Hara, K. (Éds.), *Proceedings of the 14th international conference on Human-computer interaction with mobile devices and services, Mobile HCI 2012, San Francsico, CA, USA, September 21-24, 2012* (pp. 83–92). New York, NY, USA : ACM. doi : 10.1145/2371574.2371589.

Jones, B. R., Benko, H., Ofek, E. & Wilson, A. D. (2013). IllumiRoom : Peripheral projected illusions for interactive experiences. Dans Mackay, W. E., Brewster, S. A. & Bødker,

S. (Éds.), *Proceedings of the 31st International Conference on Human Factors in Computing Systems, CHI 2013, Paris, France, April 27 - May 2, 2013* (pp. 869–878). New York, NY, USA : ACM. doi : 10.1145/2503368.2503375.

Kaehler, A. & Bradski, G. (2016). *Learning OpenCV 3 : Computer vision in C++ with the OpenCV library*. Sebastopol, CA, USA : O'Reilly Media, Inc.

Kato, H., Billinghurst, M., Poupyrev, I., Imamoto, K. & Tachibana, K. (2000). Virtual object manipulation on a table-top AR environment. Dans Klinker, G., Behringer, R., Mizell, D., Navab, N. & Stricker, D. (Éds.), *Proceedings of the IEEE and ACM International Symposium on Augmented Reality, ISAR 2000, Munich, Germany, October 5-6, 2000* (pp. 111–119). Los Alamitos, CA, USA : IEEE. doi : 10.1109/ISAR.2000.880934.

Kippelboy. (2012, avril, 12). File :Augmented reality at Museu de Mataró linking to Catalan Wikipedia (15).JPG. Repéré à https://commons.wikimedia.org/wiki/File:Augmented_reality_at_Museu_de_Matar%C3%B3_linking_to_Catalan_Wikipedia_(15).JPG.

Kistner, G. (2014). NVIDIA Automotive Screen Density Calculator. Repéré à http://phrogz.net/tmp/ScreenDensityCalculator.html.

Koelle, M., Kranz, M. & Möller, A. (2015). Don't look at me that way ! : Understanding user attitudes towards data glasses usage. Dans Boring, S., Rukzio, E., Gellersen, H. & Hinckley, K. (Éds.), *Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services, MobileHCI 2015, Copenhagen, Denmark, August 24-27, 2015* (pp. 362–372). New York, NY, USA : ACM. doi : 10.1145/2785830.2785842.

Kreylos, O. (2016, avril, 1). Optical Properties of Current VR HMDs [Billet de blogue]. Repéré à http://doc-ok.org/?p=1414.

Kreylos, O. (2012). Picture Gallery. Repéré à http://doc-ok.org/?page_id=45.

Kreylos, O. (2015, août, 18). HoloLens and Field of View in Augmented Reality [Billet de blogue]. Repéré à http://doc-ok.org/?p=1274.

Kytö, M., Ens, B., Piumsomboon, T., Lee, G. A. & Billinghurst, M. (2018). Pinpointing : Precise Head- and Eye-Based Target Selection for Augmented Reality. *Proceedings of the 36th International Conference on Human Factors in Computing Systems, CHI 2018, Montreal, QC, Canada, April 21-26, 2018*, pp. 81. doi : 10.1145/3173574.3173655.

Leap Motion. (2018, avril, 24). Project North Star : Desk UI [Vidéo en ligne]. Repéré à https://www.youtube.com/watch?v=6dB1IRg3Qls.

Lee, G. A., Nelles, C., Billinghurst, M. & Kim, G. J. (2004). Immersive authoring of tangible augmented reality applications. Dans Billinghurst, M., Baillot, Y., Klinker, G., Rolland, J. & Yamamoto, H. (Éds.), *3rd IEEE and ACM International Symposium on Mixed and Augmented Reality, ISMAR 2004, Arlington, VA, USA November 2-5, 2004* (pp. 172–181). Los Alamitos, CA, USA : IEEE. doi : 10.1109/ISMAR.2004.34.

Lee, H., Kim, D. & Woo, W. (2011). Graphical menus using a mobile phone for wearable AR systems. Dans Han, J. & Choi, A. (Éds.), *2011 International Symposium on Ubiquitous Virtual Reality (ISUVR), Jeju-si, Republic of Korea, July 1-3, 2011* (pp. 55–58). Los Alamitos, CA, USA : IEEE. doi : 10.1109/isuvr.2011.23.

Lee, J., Olwal, A., Ishii, H. & Boulanger, C. (2013). SpaceTop : Integrating 2D and spatial 3D interactions in a see-through desktop environment. Dans Mackay, W. E., Brewster, S. A. & Bødker, S. (Éds.), *Proceedings of the 31st International Conference on Human Factors in Computing Systems, CHI 2013, Paris, France, April 27 - May 2, 2013* (pp. 189–192). New York, NY, USA : ACM. doi : 10.1145/2470654.2470680.

Levy, S. (2017). Google Glass 2.0 Is a Startling Second Act. Repéré à https://www.wired.com/story/google-glass-2-is-here/.

Liu, C., Chapuis, O., Beaudouin-Lafon, M., Lecolinet, E. & Mackay, W. E. (2014). Effects of display size and navigation type on a classification task. Dans Jones, M., Palanque, P. A., Schmidt, A. & Grossman, T. (Éds.), *Proceedings of the 32nd International Conference on Human Factors in Computing Systems, CHI 2014, Toronto, ON, Canada, April 26 - May 1, 2014* (pp. 4147–4156). New York, NY, USA : ACM. doi : 10.1145/2556288.2557020.

Me, C. & Rives, P. (2007). Single view point omnidirectional camera calibration from planar grids. Dans Dario, P. & Luca, A. D. (Éds.), *Proceedings 2007 IEEE International Conference on Robotics and Automation, ICRA 2007, Roma, Italy, April 10-14, 2007* (pp. 3945–3950). Los Alamitos, CA, USA : IEEE. doi : 10.1109/robot.2007.364084.

Mehra, S., Werkhoven, P. J. & Worring, M. (2006). Navigating on handheld displays : Dynamic versus static peephole navigation. *ACM Transactions on Computer-Human Interaction*, 13(4), 448–457. doi : 10.1145/1188816.1188818.

Microsoft. (2018). Why HoloLens. Repéré à https://www.microsoft.com/fr-ca/hololens/why-hololens.

Milgram, P. & Kishino, F. (1994). A taxonomy of mixed reality visual displays. *IEICE Transactions on Information and Systems*, 77(12), 1321–1329.

Millette, A. (2016). *DualCAD : intégrer la réalité augmentée et les interfaces d'ordinateurs de bureau dans un contexte de conception assistée par ordinateur.* (Mémoire de maîtrise en génie logiciel, École de technologie supérieure, Montréal, QC, Canada). Repéré à http://espace.etsmtl.ca/1668/.

Mur-Artal, R. & Tardós, J. D. (2017). ORB-SLAM2 : An open-source SLAM system for monocular, stereo and RGB-D cameras. *IEEE Transactions on Robotics*, 33(5), 1255–1262. doi : 10.1109/tro.2017.2705103.

Nancel, M., Wagner, J., Pietriga, E., Chapuis, O. & Mackay, W. E. (2011). Mid-air pan-and-zoom on wall-sized displays. Dans Tan, D. S., Amershi, S., Begole, B., Kellogg,

W. A. & Tungare, M. (Éds.), *Proceedings of the 29th International Conference on Human Factors in Computing Systems, CHI 2011, Vancouver, BC, Canada, May 7-12, 2011* (pp. 177–186). ACM. doi : 10.1145/1978942.1978969.

Office québécois de la langue française. (2017). réalité augmentée. Repéré à http://gdt.oqlf.gouv.qc.ca/ficheOqlf.aspx?Id_Fiche=8390612.

Peterson, J. (2015). IL2CPP Internals : P/Invoke Wrappers [Billet de blogue]. Repéré à https://blogs.unity3d.com/2015/07/02/il2cpp-internals-pinvoke-wrappers/.

Pfeuffer, K., Alexander, J., Chong, M. K. & Gellersen, H. (2014). Gaze-touch : combining gaze with multi-touch for interaction on the same surface. *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology, UIST 2014, Honolulu, HI, USA, October 5-8, 2014*, pp. 509–518. doi : 10.1145/2642918.2647397.

Pfeuffer, K., Mayer, B., Mardanbegi, D. & Gellersen, H. (2017). Gaze + pinch interaction in virtual reality. *Proceedings of the 5th Symposium on Spatial User Interaction, SUI 2017, Brighton, United Kingdom, October 16 - 17, 2017*, pp. 99–108. doi : 10.1145/3131277.3132180.

Phandroid. (2013, mai, 9). Google Glass : Real Life Demo (Looking Through Glass) [Vidéo en ligne]. Repéré à https://www.youtube.com/watch?v=jK3WLILYhQs.

Piumsomboon, T., Clark, A., Billinghurst, M. & Cockburn, A. (2013). User-defined gestures for augmented reality. Dans Mackay, W. E., Brewster, S. A. & Bødker, S. (Éds.), *Proceedings of the 31st International Conference on Human Factors in Computing Systems, CHI 2013, Paris, France, April 27 - May 2, 2013* (pp. 955–960). New York, NY, USA : ACM. doi : 10.1007/978-3-642-40480-1_18.

Piumsomboon, T., Altimira, D., Kim, H., Clark, A., Lee, G. & Billinghurst, M. (2014). Grasp-Shell vs gesture-speech : A comparison of direct and indirect natural interaction techniques in augmented reality. Dans Klinker, G. & Navab, N. (Éds.), *Proceedings 2014 IEEE International Symposium on Mixed and Augmented Reality, ISMAR 2014, Munich, Germany, September 10-12, 2014* (pp. 73–82). Los Alamitos, CA, USA : IEEE. doi : 10.1109/ismar.2014.6948411.

Rädle, R., Jetter, H., Müller, J. & Reiterer, H. (2014). Bigger is not always better : Display size, performance, and task load during peephole map navigation. Dans Jones, M., Palanque, P. A., Schmidt, A. & Grossman, T. (Éds.), *Proceedings of the 32nd International Conference on Human Factors in Computing Systems, CHI 2014, Toronto, ON, Canada, April 26 - May 1, 2014* (pp. 4127–4136). New York, NY, USA : ACM. doi : 10.1145/2556288.2557071.

Rekimoto, J. & Nagao, K. (1995). The world through the computer : Computer augmented interaction with real world environments. Dans Robertson, G. G. (Éd.), *Proceedings of the 8th Annual ACM Symposium on User Interface Software and Technology, UIST 1995,*

Pittsburgh, PA, USA, November 14-17, 1995 (pp. 29–36). New York, NY, USA : ACM. doi : 10.1145/215585.215639.

Rubio, S., Díaz, E., Martín, J. & Puente, J. M. (2004). Evaluation of subjective mental workload : A comparison of SWAT, NASA-TLX, and workload profile methods. *Applied Psychology*, 53(1), 61–86. doi : 10.1111/j.1464-0597.2004.00161.x.

Serrano, M., Ens, B., Yang, X. & Irani, P. (2015a). Gluey : Developing a head-worn display interface to unify the interaction experience in distributed display environments. Dans Boring, S., Rukzio, E., Gellersen, H. & Hinckley, K. (Éds.), *Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services, MobileHCI 2015, Copenhagen, Denmark, August 24-27, 2015* (pp. 161–171). New York, NY, USA : ACM. doi : 10.1145/2785830.2785838.

Serrano, M., Ens, B., Yang, X. & Irani, P. (2015b). Desktop-Gluey : Augmenting desktop environments with wearable devices. Dans Boring, S., Rukzio, E., Gellersen, H. & Hinckley, K. (Éds.), *Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services, MobileHCI 2015, Copenhagen, Denmark, August 24-27, 2015* (pp. 1175–1178). New York, NY, USA : ACM. doi : 10.1145/2786567.2794348.

Soukoreff, R. W. & MacKenzie, I. S. (2004). Towards a standard for pointing device evaluation, perspectives on 27 years of Fitts' law research in HCI. *International Journal of Human-Computer Studies*, 61(6), 751–789. doi : 10.1016/j.ijhcs.2004.09.001.

Steptoe, W. (2013). AR-Rift (Part 1). Repéré à http://willsteptoe.com/post/66968953089/ar-rift-part-1.

Steptoe, W., Julier, S. J. & Steed, A. (2014). Presence and discernability in conventional and non-photorealistic immersive augmented reality. Dans Klinker, G. & Navab, N. (Éds.), *Proceedings 2014 IEEE International Symposium on Mixed and Augmented Reality, ISMAR 2014, Munich, Germany, September 10-12, 2014* (pp. 213–218). Los Alamitos, CA, USA : IEEE. doi : 10.1109/ismar.2014.6948430.

Sutherland, I. E. (1968). A head-mounted three dimensional display. *Proceedings of the AFIPS '68 Fall Joint Computer Conference*, 33(1), 757–764. doi : 10.1145/1476589.1476686.

Swan II, J. E. & Gabbard, J. L. (2005). Survey of user-based experimentation in augmented reality. Dans Arabnia, H. R. (Éd.), *Proceedings of 1st International Conference on Virtual Reality, Las Vegas, NV, USA, July 22-27, 2005* (pp. 1–9). CSREA Press.

Taylor, J., Bordeaux, L., Cashman, T. J., Corish, B., Keskin, C., Sharp, T., Soto, E., Sweeney, D., Valentin, J. P. C., Luff, B., Topalian, A., Wood, E., Khamis, S., Kohli, P., Izadi, S., Banks, R., Fitzgibbon, A. W. & Shotton, J. (2016). Efficient and precise interactive hand tracking through joint, continuous optimization of pose and correspondences. *ACM Transactions on Graphics (TOG)*, 35(4), 143 :1–143 :12. doi : 10.1145/2897824.2925965.

Van Dam, A. (1997). Post-WIMP user interfaces. *Communications of the ACM*, 40(2), 63–67. doi : 10.1145/253671.253708.

Van Krevelen, D. & Poelman, R. (2010). A survey of augmented reality technologies, applications and limitations. *International Journal of Virtual Reality*, 9(2), 1–20.

White, S., Feng, D. & Feiner, S. (2009). Interaction and presentation techniques for shake menus in tangible augmented reality. Dans Klinker, G., Saito, H. & Höllerer, T. (Éds.), *8th IEEE International Symposium on Mixed and Augmented Reality, ISMAR 2009, Orlando, FL, USA, October 19-22, 2009* (pp. 39–48). Los Alamitos, CA, USA : IEEE. doi : 10.1109/ismar.2009.5336500.

Wilson, A. D. (2006). Robust computer vision-based detection of pinching for one and two-handed gesture input. Dans Wellner, P. & Hinckley, K. (Éds.), *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology, UIST 2006, Montreux, Switzerland, October 15-18, 2006* (pp. 255–258). New York, NY, USA : ACM. doi : 10.1145/1166253.1166292.

Wobbrock, J. O. & Kay, M. (2016). Nonparametric Statistics in Human–Computer Interaction. Dans *Modern Statistical Methods for HCI* (pp. 135–170). Springer. doi : 10.1007/978-3-319-26633-6_7.

Wobbrock, J. O., Morris, M. R. & Wilson, A. D. (2009). User-defined gestures for surface computing. Dans Jr., D. R. O., Arthur, R. B., Hinckley, K., Morris, M. R., Hudson, S. E. & Greenberg, S. (Éds.), *Proceedings of the 27th International Conference on Human Factors in Computing Systems, CHI 2009, Boston, MA, USA, April 4-9, 2009* (pp. 1083–1092). New York, NY, USA : ACM. doi : 10.1145/1518701.1518866.

Xiao, R. & Benko, H. (2016). Augmenting the field-of-view of head-mounted displays with sparse peripheral displays. Dans Kaye, J., Druin, A., Lampe, C., Morris, D. & Hourcade, J. P. (Éds.), *Proceedings of the 34th International Conference on Human Factors in Computing Systems, CHI 2016, San Jose, CA, USA, May 7-12, 2016* (pp. 1221–1232). New York, NY, USA : ACM. doi : 10.1145/2858036.2858212.

Zhou, F., Duh, H. B. & Billinghurst, M. (2008). Trends in augmented reality tracking, interaction and display : A review of ten years of ISMAR. Dans Drummond, T., Julier, S., Livingston, M. A., Bimber, O. & Saito, H. (Éds.), *7th IEEE and ACM International Symposium on Mixed and Augmented Reality, ISMAR 2008, Cambridge, United Kingdom, September 15-18, 2008* (pp. 193–202). Los Alamitos, CA, USA : IEEE. doi : 10.1109/ismar.2008.4637362.