UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

epl

Reconnaissance de codes de sécurité à partir d'images volées

Mémoire présenté par
Nicolas LEBLANC

en vue de l'obtention du grade de Master
Ingénieur Civil Électricien

Promoteurs
Christophe DE VLEESCHOUWER, François-Xavier STANDAERT

Lecteur
Benoit MACQ

Année académique 2016-2017

Résumé

Les smartphones sont presque omniprésents au quotidien et les possibilités qu'ils offrent sont de plus en plus variées. L'importance des données accessibles pose question quant à la manière dont elles sont sécurisées. Fréquemment, un simple code de sécurité permet de déverrouiller un téléphone et d'atteindre l'essentiel de ces données. Un utilisateur aura tendance à cacher son écran lorsqu'il tape son code, sans se rendre compte qu'il est exposé malgré cette précaution.

L'objectif de ce travail est de proposer une méthode de reconnaissance de codes de sécurité à partir d'images volées. Les images volées sont capturées par le smartphone de l'attaquant. L'algorithme permet d'identifier le code saisi grâce aux reflets de l'écran dans les verres de lunettes de soleil. La méthode est essentiellement basée sur des outils de traitement d'images. Ce travail commence par une présentation des principales méthodes récentes d'identification de données à partir d'images volées. Ensuite, il continue en abordant les notions théoriques et les outils de traitement d'images utilisés. Par après, la méthode ainsi que les algorithmes implémentés sont détaillés. Pour finir, nous présentons les résultats obtenus dans le cas d'images isolées et de vidéos, à partir d'ensembles de données spécialement acquises pour ce travail.

Remerciements

J'aimerais adresser mes premiers remerciements aux promoteurs de ce mémoire, les Pr. C. De Vleeschouwer et Pr. F.-X. Standaert. Merci à eux de m'avoir accompagné tout au long de ce travail, de m'avoir familiarisé avec ce sujet original, de m'avoir offert liberté et flexibilité et surtout d'avoir dirigé mes recherches vers des orientations qui m'étaient encore inconnues. Merci au Pr. B. Macq d'avoir accepté de lire cette thèse de master.

Je remercie aussi personnellement ma famille de m'avoir soutenu et d'avoir toujours cru en moi tout au long de ces études. Merci à toutes les personnes qui m'ont aidé, que ce soit à récolter des données, à décrire leur comportement avec leur téléphone portable, à analyser leurs captures d'écran et plus particulièrement encore à Antoine Poussart d'avoir joué le jeu et accepté de faire figurer son visage dans ces pages.

Merci à mes cokotteurs, mes amis et à Deezer d'avoir accompagné et soutenu ces longs moments de recherche et de rédaction.

Nicolas Leblanc

Table des matières

Chapitre 1

Introduction

Accéder à ses mails, gérer ses réseaux sociaux, faire des virements, payer un restaurant ou encore contrôler ses transactions financières sont autant d'applications réalisables avec son téléphone. Tantôt insignifiantes, tantôt privées ou confidentielles, la montagne de données accessibles uniquement du bout de son doigt est en constante expansion. L'importance et la place centrale des smartphones dans notre quotidien n'est plus à démontrer. Il convient de s'interroger sur la sécurité mise en place pour protéger ces informations. Un nombre considérable de ces applications demandent un simple code ou pire, sont directement utilisables lorsque le téléphone est déverrouillé.

Ce travail va s'intéresser à la récupération de tels codes de sécurité à partir d'images volées au moyen d'un autre smartphone. Nous allons montrer que dans certaines circonstances, même lorsque l'utilisateur pense taper son code à l'abri des regards, il est possible de récupérer des informations à partir de reflets dans des lunettes de soleil ou dans la pupille de l'œil. Assis dans un train, autour d'une table dans un bar, debout dans une file d'attente, les scénarios d'attaque envisageables par cette méthode sont extrêmement variés et peuvent se retrouver dans bon nombre de situations quotidiennes. L'évolution grandissante de la qualité des appareils photo embarqués dans les téléphones portables et le caractère anodin d'une manipulation de son smartphone rendent la menace liée à ce genre d'attaque presque indétectable.

Ce papier est divisé en quatre chapitres principaux. Le Chapitre 2 présentera les recherches effectuées dans le domaine de reconnaissance de codes de sécurité et développera trois méthodes récentes allant dans un sens similaire à ce travail. Nous comparerons notre méthode à ce qui a déjà été réalisé, en expliquant les apports nouveaux que nous proposons. Ensuite, nous introduirons au Chapitre 3 les notions théoriques en traitement de l'image utilisées dans le développement des algorithmes. Ces outils permettront au lecteur d'appréhender au mieux les enjeux et les difficultés rencontrés. Par la suite, le Chapitre 4 détaillera la méthode générale mise en place pour reconnaître les codes de sécurité. Nous approfondirons également chacun des algorithmes développés et implémentés en discutant des paramètres importants qui y sont liés. Le Chapitre 5 conclura en présentant les résultats de notre méthode obtenus sur un ensemble d'images et de vidéos acquises spécialement pour ce travail.

Chapitre 2

État de l'art

Dans ce chapitre, nous allons parcourir les différentes recherches publiées dans le domaine de la reconnaissance de texte ou de codes de sécurité saisis sur un smartphone. L'objectif est de dresser un bref inventaire des méthodes existantes, des scénarios d'attaque envisagés et des limitations rencontrées par ceux-ci. Nous approfondirons également trois démarches similaires au travail présenté par la suite, dans l'optique de comparer plus en détail leur fonctionnement et leurs résultats.

Dans un premier temps, la littérature s'est principalement intéressée aux moyens de voler [1] des images d'une part et d'arriver à en extraire de l'information grâce aux techniques de *computer vision* [2] d'autre part. Prenons par exemple Kuhn et al. [8] qui, avec du matériel coûteux et spécifique tel que des télescopes avec large ouverture, sont parvenus à identifier du texte affiché sur l'écran d'un ordinateur situé à plus de 60 m.

Backes et al. [1] [2] proposent de ne plus se limiter aux lignes de visée directes de l'écran mais d'exploiter des réflexions de celui-ci. Ils montrent ainsi la possibilité de reconstruire un texte de grande taille à partir de la réflexion dans la pupille de l'écran d'un ordinateur situé à 10 m. À nouveau, ils utilisent un télescope puissant au coût élevé et avec l'inconvénient de rendre très peu concevable la mise en pratique d'une telle attaque dans la vie courante. Notons aussi que leur méthode ne prend nullement en considération un mouvement potentiel de la victime ou de l'attaquant et ne s'est pas essayée à la tâche ardue d'automatiser la reconstruction du texte saisi.

Citons encore Raguram et al. [11] qui, avec du matériel moins onéreux, ont réussi à déterminer la touche pressée sur un clavier de smartphone en s'aidant de l'effet *"pop-out"* (une confirmation visuelle de la touche pressée, voir FIGURE 2.1). Leur méthode fonctionne bien lorsque l'écran est photographié directement mais est moins efficace quand ce sont les réflexions de l'écran qui sont photographiées.

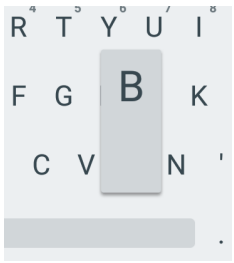


FIGURE 2.1 – Effet *"pop-out"*

D'autres chercheurs ont abordé le problème avec des approches assez semblables et tous ont rencontré les mêmes inconvénients à savoir :

- la nécessité de travailler à partir d'images de haute résolution (et donc un matériel cher) ;
- la nécessité aussi de capturer directement l'écran la plupart du temps ;
- une sensibilité importante liée au mouvement de l'appareil et à la distance par rapport à la

1. Le terme "voler" peut se comprendre ici comme *arriver à prendre des photos ou des vidéos d'une cible, appelée victime, sans que celle-ci ne s'en aperçoive.*

2. Branche de l'intelligence artificielle dont le principal but est de permettre à une machine d'analyser, traiter et comprendre une ou plusieurs images `https://en.wikipedia.org/wiki/Computer_vision`

victime ;
– des difficultés en présence d'occlusion de l'écran.

Ces inconvénients rendent les scénarios d'attaque proposés généralement inapplicables au quotidien. Nous allons aborder trois méthodes récentes qui font face à une majeure partie des problèmes suscités.

La méthode de Xu et al. [14] ne cherche pas à capturer immédiatement l'écran et se contente de reflets (parfois multiples) de mauvaise qualité, que ce soit dans des appareils électroménagers, des lunettes de soleil ou même directement dans la pupille de l'œil. Leur algorithme est capable de reconstituer plus que correctement un texte tapé sur un iPad ou un iPhone. Les scénarios étudiés sont plus variés et il est davantage concevable de les voir appliqués dans la réalité.

Fiebig et al. [5] se sont basés sur le travail de Xu et al. pour mettre en avant un système d'acquisition d'images différent. Grâce à l'installation d'une application malveillante, ils utilisent la caméra avant du smartphone pour prendre des vidéos de l'utilisateur pendant la saisie de son code PIN. Cette méthode permet d'obtenir des réflexions de l'écran de meilleure qualité.

Finalement, Shukla et al. [12] ont implémenté un algorithme semi-automatique permettant d'analyser le mouvement d'une main qui tape un code PIN pour retrouver ce dernier. Les images utilisées pour y parvenir sont enregistrées au moyen d'un autre smartphone, rendant ce scénario d'attaque exécutable par tous.

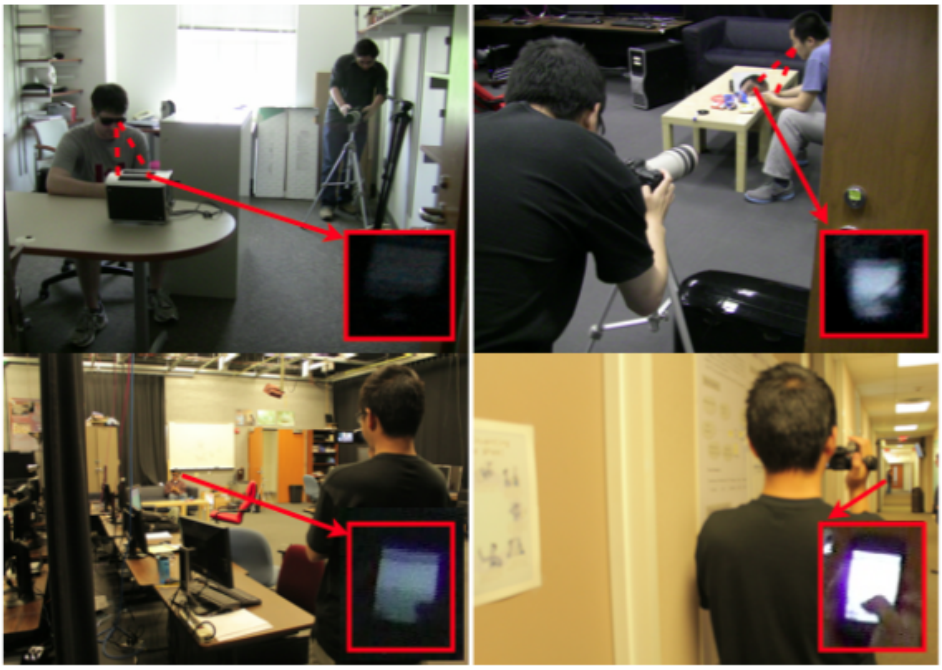


FIGURE 2.2 – Exemples de scénarios étudiés par Xu et al.. En haut à gauche : réflexions à partir de lunettes de soleil dans un grille-pain ; en haut à droite : réflexions à partir de la pupille dans un miroir ; en bas à gauche : réflexions dans des lunettes de soleil ; en bas à droite : l'écran vu à longue distance.

2.1 Xu et al. : Reconnaissance de textes tapés à partir de réflexions [14]

En 2013, Xu et al. ont pour objectif de retrouver le texte tapé par un utilisateur sur un iPhone ou un iPad dans des situations où celui-ci ne se rend pas compte qu'il est filmé. Pour y

parvenir, ils ne tentent pas de lire directement l'écran du smartphone, mais arrivent à localiser l'ombre du doigt qui tape le texte dans un reflet de l'écran. Avec la connaissance du clavier utilisé, ils parviennent à déduire correctement la touche pressée et à étendre ce procédé d'une simple image vers une vidéo pour trouver l'entièreté du texte tapé par l'utilisateur. Leur méthode permet l'utilisation d'images bruitées et de faible qualité, telles que des réflexions de l'écran dans la pupille ou dans des lunettes de soleil. Il leur est même possible d'utiliser des réflexions multiples comme le reflet du téléphone dans des lunettes de soleil dans un grille-pain (exemples FIGURE 2.2). Grâce à l'emploi d'images de plus faible résolution, ils peuvent travailler avec des appareils photos moins coûteux ($\sim 1000\ \$ - 2000\ \$$).

Leur algorithme se décompose en 5 étapes principales, suivies d'une étape facultative (voir FIGURE 2.3). Le reflet de l'écran est d'abord identifié et suivi dans la vidéo (1). Ensuite, ce reflet est aligné par rapport à un modèle du clavier utilisé (2). Par après, l'empreinte du doigt est extraite du reflet (3) afin d'analyser sa trajectoire (4). À partir de cette trajectoire, les touches pressées les plus probables sont identifiées (5). Finalement, les résultats de l'étape 5 peuvent être couplés à un modèle de langage afin d'améliorer la qualité du texte reconstruit (6) [3].

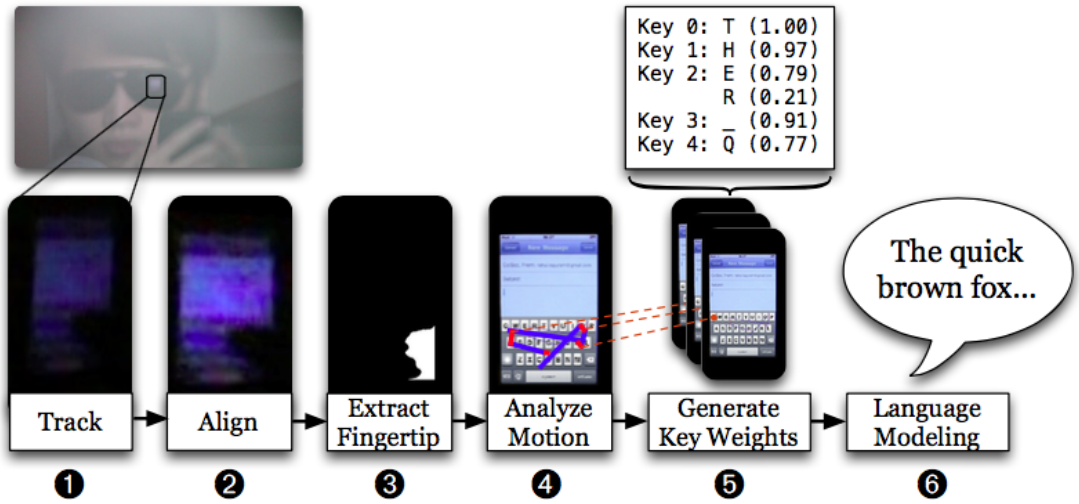


FIGURE 2.3 – Différentes étapes de l'algorithme de reconnaissance de Xu et al.

2.1.1 Algorithme

Étape 1 : Tracking D'abord, la zone utile (écran, reflet de l'écran) est repérée manuellement dès le début de la séquence pour permettre le suivi de cette zone tout au long de la vidéo grâce à une méthode de tracking basé sur AdaBoost [7] [4]. Afin de lisser l'image (*smoothing*) et de réduire le bruit, chaque image utilisée est la moyenne de 3 frames consécutives. Cela permet également d'améliorer le traquage.

Étape 2 : Alignment L'écran ou son reflet localisé dans la vidéo doit être remis horizontalement pour être analysé. Il passe d'abord part un filtre anisotropique qui permet de réduire le bruit tout en gardant la *line structure* intacte. Ensuite, les bords de l'image sont extraits et l'orientation de celle-ci est déterminée grâce à la transformée de Hough.

3. Les différentes étapes présentées ici seront parfois accompagnées de choix d'implémentation plus techniques dans l'optique de pouvoir comparer ces choix avec ceux des autres méthodes évoquées par la suite.
4. Algorithme de classification en machine learning basé sur l'apprentissage automatique.

Étape 3 : Fingertip Extraction Dans un premier temps, les pixels correspondant à l'ombre du doigt (pixels plus foncés) et à l'écran (pixels plus clairs) sont modélisés par deux distributions gaussiennes. Ces distributions sont calculées grâce à l'analyse des 50 premières frames de la vidéo (*learning set*). L'idée est d'associer chacun des pixels des frames suivantes à l'une des deux distributions.

Dans un second temps, le doigt est supposé s'apparenter à une ellipse et son point de contact avec l'écran est choisi comme l'intersection de l'extrémité de l'axe focal avec le clavier.

Étape 4 : Fingertip Motion Analysis Les positions récupérées à l'étape 3 sont combinées pour former une trajectoire qui est représentée dans un espace 3D (plan x-y de l'image et numéro de frame dans la vidéo). Il convient également de s'attarder sur le mouvement réel du doigt lorsque des touches sont pressées. Premièrement, le doigt reste plus longtemps sur une touche appuyée que lorsqu'il survole le reste du clavier. Deuxièmement, le doigt change souvent de direction après avoir appuyé sur une touche pour aller vers la suivante.

La trajectoire est approximée par des segments de droite dans l'espace 3D au moyen de la méthode RANSAC [6] [5]. Ensuite les positions d'arrêt sont déterminées par les segments perpendiculaires ou presque au plan x-y (moments où le doigt reste plus longtemps à une position), tandis que les changements brusques de direction sont déterminés par deux segments adjacents avec un important changement de direction.

Étape 5 : Inferring the Keys Pressed La difficulté à ce stade réside dans les différentes positions que peut prendre le doigt pour presser une même touche donnée. Les auteurs ont donc classé manuellement parmi 87 vidéos une multitude d'échantillons correspondant à des touches pressées. Ces échantillons ont ensuite servi d'exemples positifs (positive sample) pour la touche effectivement pressée et également d'exemples négatifs (negative sample) pour l'ensemble des autres touches.

Pour chacun des patchs extraits, des descripteurs SIFT [13] sont calculés et utilisés afin d'entraîner un classificateur AdaBoost [7], ce qui permet de créer un distingueur pour chacune des touches. Avec cet ensemble de distingueurs, chaque position d'arrêt obtenue à l'étape 4 est analysée afin de déterminer la ou les touches potentiellement pressées. Les résultats sont donnés avec un intervalle de confiance compris entre $[0, 1]$.

Étape 6 : Language Model Cette dernière étape optionnelle permet d'améliorer les résultats de l'algorithme lorsque la victime tape du texte cohérent, contrairement à ce que pourrait être un mot de passe ou du langage SMS. Les résultats obtenus à l'étape 5 passent par un modèle de langage afin de déterminer les mots et les phrases qui sont tapés par l'utilisateur.

2.1.2 Résultats

Pour évaluer la qualité du texte reconstruit, Xu et al. utilisent la métrique METEOR [9]. Celle-ci ne calcule pas seulement les différences entre le texte original et le texte reconstruit mais prend également en compte la lisibilité et la compréhensibilité du texte. Ainsi, si le sens du message reconstruit est le même que celui du message original, le score METEOR restera élevé malgré des erreurs de reconstruction. Le score METEOR est compris entre $[0, 1]$ et on considère qu'à partir de 0,5 le texte est juste compréhensible, tandis qu'avec un score supérieur à 0,7 la reconstruction

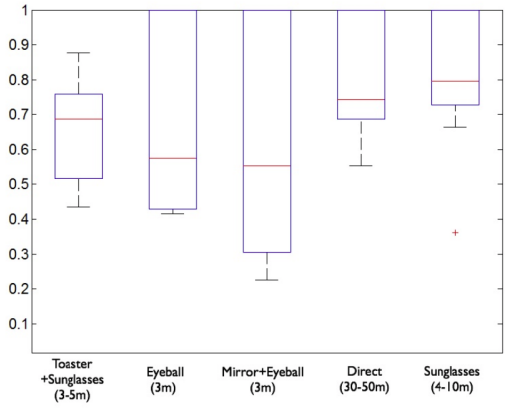


FIGURE 2.4 – Scores METEOR pour la méthode de Xu et al., classés par scénario.

5. Cette méthode sera approfondie au Chapitre 4.

est acceptable et facilement compréhensible. Les résultats sont présentés à la FIGURE 2.4. En prime, Xu et al. sont parvenus à reconstruire parfaitement 23 % des phrases, et 92 % des phrases ont un score METEOR supérieur à 0,5. Il convient néanmoins de nuancer ces résultats par les conditions toujours "contrôlées" (éclairage, focus, etc.) de la méthode d'acquisition. Dans le cadre d'environnements plus *challenging*, les auteurs admettent que certaines de leurs vidéos seraient inutilisables et conduiraient à des résultats plus faibles.

2.2 Fiebig et al. : Reconnaissance de codes PIN à partir de la caméra frontale d'un smartphone [5]

Toujours dans l'optique de récupérer des données saisies sur un smartphone grâce à des images volées, Fiebig et al. se sont inspirés de la méthode présentée par Xu et al. et proposent une manière différente d'acquérir les images volées.

Ils partent de deux constats importants :

1. La quasi totalité des smartphones disponibles sur le marché possèdent désormais une caméra frontale dont la résolution ne cesse d'augmenter de génération en génération.
2. Une application peut avoir accès à la caméra frontale du téléphone en demandant simplement la permission lors de son installation. Or des études [4] montrent que la majorité des utilisateurs ne prêtent pas (ou très peu) attention aux permissions requises pour installer une application.

L'idée de Fiebig et al. est donc d'utiliser cette caméra frontale afin de prendre des vidéos à l'insu de l'utilisateur lorsque celui-ci tape son code PIN. Pour avoir accès à cette caméra, ils présentent le scénario où un utilisateur voudrait installer une application *a priori* inoffensive (mini-jeux, lampe torche, etc.) dont l'accès à la caméra frontale figure dans les demandes de permission. Dans le cas prévisible où la victime accepterait ces permissions sans remarquer la faille, elle viendrait d'installer le *trojan*[6] qui permettra l'acquisition des vidéos.

Fiebig et al. utilisent ensuite les réflexions de l'écran dans des lunettes de soleil ou dans la pupille pour retrouver un code PIN, un peu comme le font Xu et al., à la différence notable que Fiebig et al. ne proposent pas de nouvelle méthode pour automatiser la reconnaissance du code tapé. Ils affirment que l'algorithme de Xu et al. est déjà très performant avec leur propre méthode d'acquisition et devrait encore mieux fonctionner avec les images de meilleure qualité qu'ils parviennent à voler. Ce raisonnement est soutenu par deux approches différentes : une approche théorique et une vérification expérimentale.

2.2.1 Approche théorique

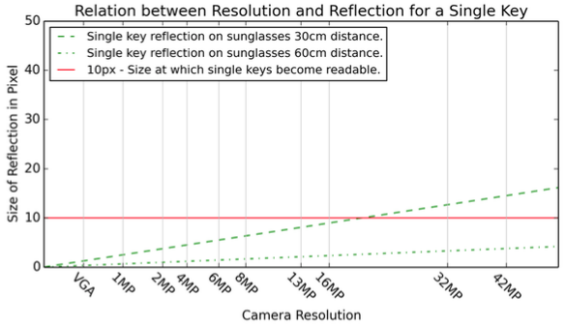
La qualité de l'image acquise dépend principalement de deux facteurs : la qualité de l'appareil photo et la distance entre la cible et l'appareil. Xu et al. [14] ont créé deux formules pour calculer la taille de la cible en fonction de ces facteurs. La première formule donne la taille en pixel par axe pour une observation directe de la cible.

$$\text{Size}_{\text{Direct}} = \frac{\text{SensorResolution}}{\text{SensorSize}} \cdot \frac{\text{ObjectSize}}{\frac{\text{TargetDistance}}{\text{FocalLength}} - 1} \tag{2.1}$$

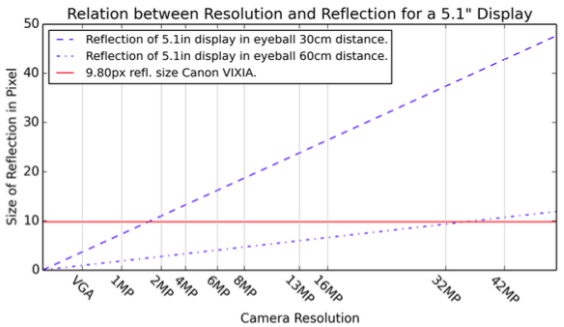
La seconde prend en compte le rayon de courbure de la surface dans le cas de réflexions.

$$\text{Size}_{\text{Reflexion}} = \text{Size}_{\text{Direct}} \cdot \left(\frac{2 \cdot \text{DistanceFromSurface}}{\text{CurvatureRadius}} + 1 \right)^{-1} \tag{2.2}$$

6. Logiciel malveillant d'apparence légitime, pour plus d'informations : `https://en.wikipedia.org/wiki/Trojan_horse_(computing)`.



(a) Réflexion d'une simple touche située entre 30 et 60 cm dans des lunettes de soleil



(b) Réflexion d'un écran 5.1" situé entre 30 et 60 cm dans la pupille

FIGURE 2.5 – La figure (a) représente la taille d'une simple touche dans une image reflétée dans des lunettes de soleil. La ligne rouge indique la taille à partir de laquelle les touches deviennent distinguables entre elles. La figure (b) représente la taille de la réflexion d'un écran 5.1" dans la pupille d'un utilisateur. La ligne rouge indique la taille de la réflexion à partir de laquelle une reconnaissance parfaite par la méthode de Xu et al. est presque garantie.

Au moyen de ces deux formules, Fiebig et al. ont comparé la taille et la qualité des images volées par rapport à la méthode d'acquisition pour les configurations suivantes :

- La victime utilise un téléphone OPPO N1 doté d'un écran de 13 cm × 7.5 cm et d'une caméra frontale de 13 MP [7].
- Le scénario d'attaque pour Xu et al. est une simple réflexion de l'écran dans l'œil de la victime, capturée par un CANON VIXIA à 4 m de distance.
- Le scénario d'attaque pour Fiebig et al. est une simple réflexion de l'écran dans l'œil de la victime, capturée par la caméra frontale du téléphone lui-même.

Les résultats obtenus avec le CANON VIXIA :

$$X_{\text{VIXIA}} = \frac{1920px}{4.84\ \text{mm}} \cdot \frac{130\ \text{mm}}{\frac{4000\ \text{mm}}{57\ \text{mm}} - 1} \cdot \frac{1}{\frac{2 \cdot 300\ \text{mm}}{8\ \text{mm}} + 1} \cong 9.80 \tag{2.3}$$

$$Y_{\text{VIXIA}} = \frac{1080px}{3.42\ \text{mm}} \cdot \frac{75\ \text{mm}}{\frac{4000\ \text{mm}}{57\ \text{mm}} - 1} \cdot \frac{1}{\frac{2 \cdot 300\ \text{mm}}{8\ \text{mm}} + 1} \cong 4.50 \tag{2.4}$$

Et ceux obtenus avec la caméra frontale de l'OPPO N1 :

$$X_{\text{OPPO}} = \frac{4160px}{4.4\ \text{mm}} \cdot \frac{130\ \text{mm}}{\frac{300\ \text{mm}}{5\ \text{mm}} - 1} \cdot \frac{1}{\frac{2 \cdot 300\ \text{mm}}{8\ \text{mm}} + 1} \cong 27.41 \tag{2.5}$$

$$Y_{\text{OPPO}} = \frac{3120px}{3.6\ \text{mm}} \cdot \frac{75\ \text{mm}}{\frac{300\ \text{mm}}{5\ \text{mm}} - 1} \cdot \frac{1}{\frac{2 \cdot 300\ \text{mm}}{8\ \text{mm}} + 1} \cong 14.50 \tag{2.6}$$

La taille des images volées grâce à la caméra frontale (27.41 px × 14.50 px) est presque neuf fois plus grande que celle des images volées grâce à l'appareil photo (9.80 px × 4.50 px) situé à 4 m de distance.

Ces résultats sont intéressants à mettre en parallèle avec les graphes de la FIGURE 2.5. On y observe qu'il devient possible de différencier les touches à partir d'une résolution aux alentours

7. Téléphone possédant la caméra frontale avec la plus haute résolution disponible sur le marché en 2014.

(a) L'utilisateur tape avec une main et tient le téléphone avec l'autre. L'écran est caché d'une vidéo qui filmerait l'arrière du téléphone.

(b) Une personne tape son code PIN à un distributeur automatique. Le main et les doigts sont visibles mais pas le clavier du distributeur

(c) L'utilisateur tape le code de la serrure d'une porte. Le doigt est visible mais pas le clavier de la serrure.

(d) L'utilisateur tape et tient le téléphone avec la même main. L'écran est caché d'une vidéo qui filmerait l'arrière du téléphone.

(e) L'utilisateur tape son code PIN à un distributeur automatique. Les doigts et une partie du clavier sont visibles.

(f) L'utilisateur tape le code de la serrure d'une porte. Le doigt et une partie du clavier sont visibles.

FIGURE 2.6 – Exemples de scénarios d'attaque potentiels gérés par Shukla et al.

de 16MP. En 2017 des smartphones possédant ces caractéristiques existent sur le marché [8], ce qui n'était pas le cas au moment de cette étude en 2014.

2.2.2 Vérification expérimentale

Pour se passer de reconnaissance automatique (ou semi-automatique), Fiebig et al. ont imaginé une autre façon de procéder. Ils ont demandé à des sujets de taper un code PIN de leur choix, compris entre 4 et 7 chiffres, pendant que la caméra frontale du téléphone les filmait. D'autres personnes ont ensuite essayé de deviner les codes saisis à partir des vidéos enregistrées. Les résultats montrent que dans 50 % des cas le code PIN est trouvé au premier essai. Ces résultats expérimentaux ne sont pas à prendre quantitativement mais qualitativement. Fiebig et al. veulent mettre en avant que si un œil humain non-entraîné est capable de retrouver le code tapé, alors il est quasi certain que la méthode de Xu et al. y arrivera également.

2.3 Shukla et al. : Reconnaissance de codes PIN à partir du mouvement de la main [12]

L'objectif de Shukla et al. est de retrouver le code PIN saisi sur un smartphone, mais leur méthode peut s'adapter également pour un code entré dans un distributeur automatique ou encore sur le verrou d'une porte (voir FIGURE 2.6). Contrairement à Xu et al. [14] et Fiebig et al. [5], les images volées n'ont plus pour objectif de capturer le clavier (par l'intermédiaire de l'écran) ou un quelconque reflet de celui-ci, mais simplement la main qui effectue le code et l'arrière du téléphone, ou tout au moins un point de repère dans celui-ci. Leurs algorithmes analysent le mouvement du doigt qui tape le code, et plus généralement de la main associée, pour identifier le code PIN correct.

Les vidéos utilisées sont enregistrées à partir d'un autre smartphone [9], ce qui rend dès lors les scénarios d'attaque difficilement détectables par la victime. Les algorithmes développés par Shukla et al. seront présentés succinctement étant donné que la matière théorique liée sort du cadre de ce travail, mais il est néanmoins intéressant d'attirer l'attention sur certaines nécessités requises pour assurer le bon fonctionnement de leur méthode.

- L'attaquant a besoin d'identifier manuellement la séquence de la vidéo où la victime tape son code PIN.
- L'attaquant doit connaître un couple *touche pressée/frame* pour calculer les positions relatives du doigt. Par exemple il sait que la dernière touche tapée par la victime doit être la touche 'enter' pour valider le code PIN.
- L'attaquant connait le modèle du téléphone et plus précisément la géométrie du clavier virtuel utilisé, la longueur et la largeur du téléphone.
- Il est nécessaire de sélectionner deux points d'intérêt visibles tout au long de la vidéo et appartenant respectivement à la main et à l'arrière du téléphone, afin de traquer ceux-ci. Il est aussi préférable de choisir le point d'intérêt appartenant à la main le plus proche possible de l'écran. L'algorithme de détection utilisé pour suivre ces points est le Tracking Learning Detection (TLD) [10], disponible en open source.

Leur méthode mesure d'abord la distance entre les deux points d'intérêts pour toutes les frames de la vidéo. Ils définissent alors les frames correspondantes à un minimum local comme les moments où la victime appuie sur l'écran. Ensuite, à l'aide de sa taille réelle, ils arrivent à calculer les dimensions (longueur et largeur) du téléphone dans l'image, même lorsqu'une partie de l'appareil est cachée par la main. Ces valeurs leur permettent de redimensionner la taille du clavier virtuel pour le faire correspondre à l'image. Ils peuvent alors estimer la touche pressée en associant les différences d'abscisses et d'ordonnées de la frame sélectionnée par rapport à la dernière frame pressée (correspondant à la touche "enter") aux différentes lignes et colonnes composant le clavier digital.

L'attaque mise en œuvre par Shukla et al. arrive à retrouver en moyenne aux alentours de 50% des codes PIN saisis au premier essai et 85% après 10 essais.

2.4 Positionnement de la méthode proposée dans ce travail

La méthode développée dans le cadre de ce travail essaye de tirer profit des avantages de ces trois méthodes.

Commençons par le schéma d'attaque. Tout comme Shukla et al. [12], nous proposons une méthode d'acquisition des images par la caméra arrière d'un smartphone. La démocratisation

8. Samsung Galaxy A5 (2017), Front Camera : 16MP, Rear Camera : 16MP `http://www.samsung.com/be_fr/smartphones/galaxy-a5-2017-a520/SM-A520FZKALUX/`.

9. Google Nexus 5, Front Camera : 12,3MP

	Xu et al.	Fiebig et al.	Shukla et al.	Notre méthode
Identification code PIN	Oui	Oui	Oui	Oui
Identification texte	Oui	Non	Non	Non
Méthode automatisée	Semi-automatique	Non	Semi-automatique	Semi-automatique
Matériel nécessaire	Caméra	Installation d'une application trojan	Smartphone avec caméra arrière	Smartphone avec caméra arrière
Prix du matériel	$\sim$ 1000 €	Difficile à estimer	$\sim$ 150 €	$\sim$ 150 €
Distance d'acquisition des images	4 m - 1 0m	30 cm - 60 cm	4 m - 5 m	1 m - 4 m
Complexité théorique des algorithmes	Oui	Non	Oui	Non
Phase de learning	Oui	Non	Oui	Non

TABLE 2.1 – Tableau comparatif des différentes méthodes

du prix des smartphones, leur utilisation quasi permanente ou encore la constante évolution de la qualité des appareils photos rendent ce scénario d'attaque dangereux car potentiellement omniprésent au quotidien. Tout le monde pourrait à la fois être la victime ou l'assaillant de ce genre d'attaque qui ne demande pas d'installer de trojan ou de posséder un matériel spécifique et coûteux.

Continuons avec le procédé d'identification des données saisies par l'utilisateur. Les algorithmes présentés au Chapitre 4 utilisent presque essentiellement des outils de traitement d'images, rapides et faciles à implémenter. Notre méthode offre une simplicité d'utilisation sans aucune nécessité de constituer une base de données pour effectuer une phase de *learning*. Rajoutons que si actuellement la méthode requiert une intervention manuelle pour sélectionner le visage de la victime, il existe déjà de nombreux détecteurs de visage qui pourraient rendre notre méthode totalement automatique.

Comparons également certaines hypothèses importantes. Tout d'abord, les quatre méthodes demandent une connaissance du modèle du clavier virtuel utilisé, nous en discuterons lors du Chapitre 5. Ensuite sauf pour la méthode de Shukla et al. [12], aucune des trois autres méthodes ne peut fonctionner dans des environnements défavorables à l'apparition de reflets : milieu en extérieur avec une forte luminosité, un écran pas assez lumineux, impossibilité de capturer le reflet, etc. Shukla et al. sont les seuls à apporter une solution dans ce genre de situations, mais en revanche leur méthode rencontre des difficultés s'ils ne peuvent trouver un point d'intérêt utilisable appartenant à la main, ce qui est fréquent lorsque l'utilisateur utilise son ou ses pouces pour utiliser le clavier.

Pour toutes ces raisons (acquisitions de la vidéo par un smartphone classique, algorithmes simples d'implémentations, etc) la méthode que nous proposons nous permet d'envisager une possible implantation directe dans des applications téléphoniques, avec pourquoi pas une adaptation à des situations en temps réel.

2.5 Conclusion

La reconnaissance de textes ou de codes de sécurité saisis sur un smartphone à partir d'images volées est un domaine récent de recherche qui n'en est pas moins déjà bien étudié. Les recherches montrent par la diversité des méthodes et des schémas d'attaque établis la difficulté de s'assurer une certaine discrétion et sécurité lors de la saisie de son code.

La méthode de Xu et al. [14] donne certainement les meilleurs résultats grâce à la possibilité de retrouver le texte tapé par l'utilisateur, mais nécessite du matériel coûteux d'une part et implique peut-être une des méthodes d'acquisition les moins envisageables en situation réelle d'autre part. Fiebig et al. [5] mettent en évidence un scénario intéressant d'acquisition par la caméra frontale sans pour autant proposer de nouvelles méthodes de traitement d'image. Ce scénario reste toutefois difficilement applicable si l'attaquant ne dispose pas de compétences importantes en informatique. Enfin, Shukla et al. [12] proposent à la fois une méthode d'acquisition très réaliste sans être onéreuse et un algorithme de reconnaissance performant mais contraint à des hypothèses fortes (avoir deux points de repère disponibles tout au long de la vidéo, connaissance du modèle de la victime, d'un couple *touche pressée/frame*, etc).

Chapitre 3

Méthodes de traitement de l'image

Ce chapitre va présenter les notions théoriques importantes pour assurer la bonne compréhension des algorithmes qui seront exposés par la suite.

Traitement de l'image

Cette partie [1] n'a nullement la prétention d'être une introduction aux domaines extrêmement vastes que représentent le traitement d'image (*image processing*) et la vision par ordinateur (*computer vision*). L'objectif ici est de rappeler au lecteur quelques notions et opérations essentielles pour comprendre les algorithmes développés, et appréhender au mieux l'analyse des résultats présentés aux chapitres suivants.

Les premiers rappels concerneront l'image numérique et sa représentation en général, ensuite nous nous intéresserons aux images en niveau de gris et à certaines opérations associées, pour finir avec les images binaires et quelques éléments de morphologie mathématique.

3.1 Représentation d'une image

Une des manières de représenter une image en traitement d'images numériques est d'utiliser la forme matricielle (exemple FIGURE 3.1). L'image est alors représentée par une matrice (tableau) de points à plusieurs dimensions (temporelles, spatiales ou autres). Dans le cadre de ce travail nous allons travailler avec deux types différents d'images : les photographies qui sont des images comportant deux dimensions spatiales et les vidéos qui sont des images à trois dimensions : deux dimensions spatiales et une dimension temporelle.

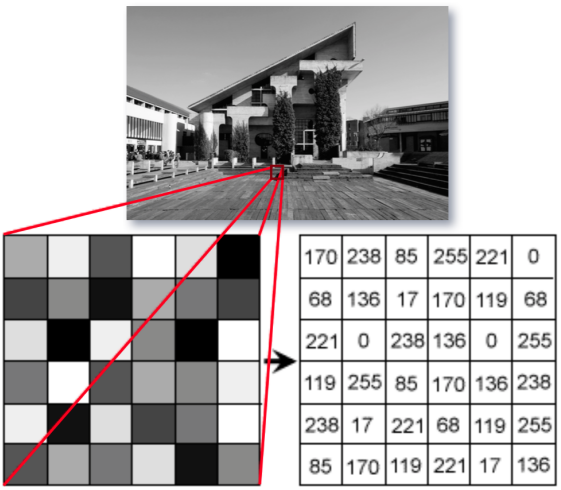


FIGURE 3.1 – Représentation sous forme matricielle d'une image en *grayscale*.

Il est d'abord utile de rappeler quelques concepts de base liés aux images et plus spécifiquement aux images numériques.

– Les points d'une image 2D sont appelés **pixels** (*px*).

1. Largement inspirée de l'excellent cours donné par les Pr. C. De Vleeschouwer et Pr. L. Jacques, ELEC2885 : *Image processing and Computer vision*.

- La taille d'une image 2D est définie par sa **hauteur** (*height*) et sa **largeur** (*width*).
- La **résolution** est le nombre de pixels composant l'image.
- La **profondeur** est le nombre de symboles binaires utilisés pour décrire chaque pixel et le *dynamic range* est l'ensemble de toutes les valeurs possibles.
- Le **contraste** caractérise la différence de luminosité entre les parties claires et sombres d'une image. Il a été défini mathématiquement par Michelson comme :

$$C_m = \frac{L_{\max} - L_{\min}}{L_{\max} + L_{\min}}$$

où L désigne la luminance.
- Une **frame** désigne une des nombreuses images fixes qui compose une vidéo et le *frame rate* est le nombre d'images capturées par unité de temps.

Les notions suivantes sont définies selon l'utilisation qui en est faite dans ce travail. Il existe d'autres manières de les définir mais cette différence n'a pas d'impact majeur sur les résultats obtenus. Dans le cas contraire leur impact sera discuté au Chapitre 5.

Images couleurs Les images en couleurs sont représentées selon le modèle RGB [2]. Chaque pixel est constitué d'une composante rouge (**R**ed), verte (**G**reen) et bleue (**B**lue). Par exemple, si l'image est représentée avec une profondeur de 24 bits, il y aura 256 ($= 2^8$) valeurs possibles pour la composante rouge, 256 pour la composante verte et 256 pour la composante bleue. Il sera donc possible d'afficher environ 16,7 millions de couleurs différentes.

Images en niveau de gris Les images en niveau de gris (exemple FIGURE 3.1), ou *grayscale* ou encore en "noir et blanc", sont des images où chaque pixel est défini par une seule composante de luminosité. Ces images sont composées uniquement de nuances de gris, allant du noir pour la luminosité la plus faible jusqu'au blanc pour la luminosité la plus forte.

Images binaires Les pixels d'une image binaire ne peuvent avoir que deux valeurs possibles ("0" et "1"), typiquement affichées en noir et blanc (associées respectivement à "0" et à "1") lors de la visualisation de l'image. Une image binaire est par exemple utilisée pour définir le masque d'avant plan d'une scène filmée. Dans ce cas, les pixels blancs sont désignés comme étant l'avant-plan (*foreground*) de l'image, et les pixels noirs comme étant l'arrière-plan (*background*).

Vidéo Les vidéos correspondent à une série d'images 2D consécutives. Dans notre travail, chaque image de la séquence est traitée indépendamment.

3.2 Opérations sur des images en niveau de gris

3.2.1 Conversion de RGB à grayscale

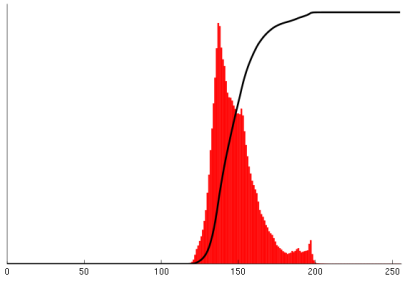
Convertir une image couleur en une image en niveau de gris s'effectue au moyen d'une somme pondérée des composantes RGB. La pondération utilisée est la suivante :

$$Y = 0.2989 * \mathrm{R} + 0.5870 * \mathrm{G} + 0.1140 * \mathrm{B}$$

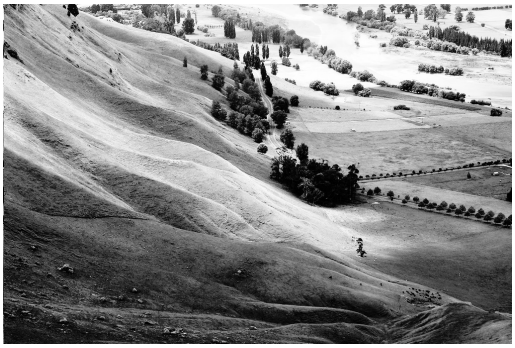
où Y correspond au signal de luminance ou *luma* [3].

2. voir `https://en.wikipedia.org/wiki/RGB_color_model`.
3. voir `https://fr.wikipedia.org/wiki/Luminance`.

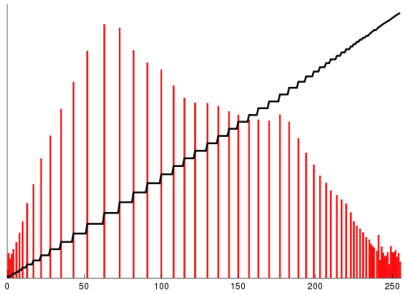
(a) Image avant égalisation d'histogramme



(b) Histogramme correspondant (rouge) et histogramme cumulatif (noir)



(c) Image après égalisation d'histogramme



(d) Histogramme correspondant (rouge) et histogramme cumulatif (noir)

FIGURE 3.2 – Exemple d'égalisation d'histogramme.

3.2.2 Histogramme et améliorations basiques d'images

Comme chaque pixel peut prendre un nombre discret de valeurs, il est possible d'associer à une image un histogramme représentant la distribution des intensités de celle-ci (exemple FIGURE 3.2). Cet histogramme peut être ajusté afin de transformer et d'"améliorer" l'image de base.

Linear Stretch

Cette méthode consiste à étirer l'histogramme de sorte à couvrir tout le *dynamique range*. Contrairement aux opérations présentées par après, celle-ci est linéaire.

Histogram equalization

L'égalisation d'histogramme (voir FIGURE 3.2) est une transformation non-linéaire sur chaque pixel. Elle consiste à répartir uniformément l'histogramme cumulé en "étalant" celui-ci. Cette méthode, rapide et facile d'implémentation, améliore le contraste global de l'image. Nous pouvons la définir mathématique de la manière suivante.

Pour une image $\{x\}$ codée sur L niveaux (profondeur), n_k est défini comme le nombre d'occurrences du niveau x_k. La fréquence d'occurrence d'un pixel de niveau x_k dans l'image est

$$p_x(x_k) = p(x = x_k) = \frac{n_k}{n}$$

avec n le nombre total de pixels de l'image.

FIGURE 3.3 – Filtre de Canny appliqué à une photographie.

La transformation T qui à chaque pixel de valeur x_k de l'image d'origine associe une nouvelle valeur $s_k = T(x_k)$ est alors définie par

$$T(x_k) = (L-1)\sum_{j=0}^{k} p_x(x_j)$$

où $\sum_{j=0}^{k} p_x(x_j)$ représente l'histogramme cumulé.

Adaptive Histogram Equalization

L'égalisation d'histogramme adaptative (*Adaptive Histogram Equalization* ou AHE) diffère de l'égalisation ordinaire car la méthode d'adaptation calcule plusieurs histogrammes, chacun correspondant à une section distincte de l'image. Contrairement à l'égalisation ordinaire, chaque pixel de l'image ne sera pas transformé de la même manière. Cette méthode améliore le contraste local mais peut avoir tendance à amplifier le bruit dans les régions relativement plus homogènes.

Il est possible de limiter le contraste lors de l'égalisation pour réduire ce problème. Cette variante se nomme le *Contrast Limited Adaptive Histogram Equalization* ou CLAHE [15].

3.2.3 Edge detection

La détection de contours vise à identifier les points d'une image où la luminosité change brusquement. Ces points sont généralement organisés en un ensemble de lignes et de courbes, et forment les contours ou bords de l'image. En plus de réduire considérablement la quantité de données à traiter, ceux-ci traduisent généralement des événements importants ou des changements dans les propriétés de l'image, tels que des discontinuités dans la profondeur, dans l'orientation d'une surface, dans les propriétés d'un matériau ou dans l'éclairage d'une scène. Le résultat de la détection est une image binaire.

Il existe plusieurs méthodes de détection (Sobel, Prewitt, Roberts, etc) mais nous nous intéresserons ici à l'algorithme proposé par Canny [3].

Canny edge detector[3]

La méthode de Canny (exemple FIGURE 3.3) trouve les contours en recherchant les maxima locaux du gradient d'intensité de l'image. Elle a pour avantages d'être moins sensible au bruit et de mieux détecter les contours "faibles" (où la différence d'intensité est moindre) par rapport aux autres méthodes évoquées. L'algorithme peut se décomposer en 4 étapes.

Étape 1 : Réduction du bruit Un filtre gaussien est appliqué pour lisser l'image (*smoothing*) et réduire le bruit. La variance et donc la taille du filtre utilisé va impacter les performances du détecteur.

Étape 2 : Calcul des gradients d'intensité L'opérateur composé de deux masques de convolution permet de calculer le gradient suivant les directions horizontale (X) et verticale (Y). Grâce à ces deux gradients, une carte de l'intensité et de l'orientation des contours est obtenue.

Étape 3 : Suppression des non-maxima Une forte intensité indique une forte probabilité de présence d'un contour. Afin d'éliminer les contours parasites détectés à l'étape précédente, seuls les maxima locaux sont considérés comme des contours effectifs.

Étape 4 : Seuillage par hysteresis À ce stade, il reste certains contours dus au bruit ou à la variation de couleurs. Pour éliminer ceux-ci, deux seuils sont fixés : un seuil haut et un seuil bas.

- Les pixels dont le gradient d'intensité est supérieur au seuil haut sont désignés bords forts (*strong edge*) et sont gardés.
- Les pixels dont le gradient est inférieur au seuil bas sont rejetés.
- Les pixels dont le gradient est compris entre les deux seuils sont désignés bords faibles (*weak edge*). Les bords faibles sont gardés uniquement s'ils sont connectés à des bords forts dans le but d'assurer la continuité de ces derniers.

3.3 Opérations sur des images binaires

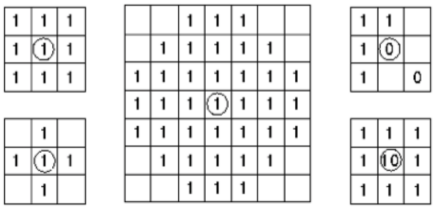


FIGURE 3.4 – Exemples d'éléments structurants.

L'idée principale de la morphologie mathématique [4] est de sonder une image binaire avec une forme simple et prédéfinie, en tirant des conclusions sur la façon dont cette forme touche ou est incluse dans l'image initiale. Cette sonde s'appelle élément structurant (*structuring element* (voir FIGURE 3.4)) et est elle-même une image binaire.

Cette section va présenter quelques opérateurs de base de morphologie mathématique. Il convient d'abord d'établir quelques notations.

- X : l'ensemble de coordonnées euclidiennes correspondant à l'image binaire d'entrée.
- K : l'ensemble de coordonnées correspondant à l'élément structurant.
- K_x : la translation de K de telle sorte que son origine soit située en x.

3.3.1 Dilation

La dilatation (*dilation*) fait partie des opérateurs morphologiques de base. Elle est définie comme l'ensemble de tous les points x tels que l'intersection de K_x avec X est non-vide.

$$X \oplus K = \{x \mid K_x \cap X \neq \varnothing\}$$

La dilatation de X par K peut être vue comme l'ensemble des pixels couverts par K lorsque le centre de K se déplace à l'intérieur de X (exemple FIGURE 3.5).

Son effet de base est d'élargir les limites des régions des pixels à l'avant-plan.

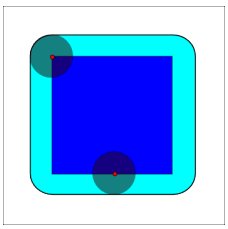


FIGURE 3.5 – Le carré bleu clair est le résultat de la dilatation du carré bleu foncé par un disque.

4. Théorie et technique d'analyse de structure, voir `https://fr.wikipedia.org/wiki/Morphologie_mathématique`.

3.3.2 Erosion

L'érosion (*erosion*) fait partie des opérateurs morphologiques de base. Elle est définie comme l'ensemble de tous les points x tels que K_x est inclus dans X.

$$X \ominus K = \{x \mid K_x \subseteq X\}$$

L'érosion de X par K peut être vue comme l'ensemble des points couverts par le centre de K lorsque K se déplace à l'intérieur de X (exemple FIGURE 3.6).

Son effet de base est de rétrécir les limites des régions des pixels à l'avant-plan.

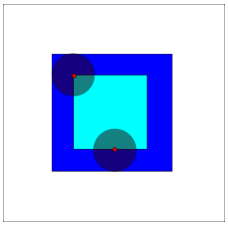


FIGURE 3.6 – Le carré bleu clair est le résultat de l'érosion du carré bleu foncé par un disque.

3.3.3 Opening

L'ouverture (*opening*) est obtenue par l'érosion de X par K, suivie de la dilatation par K de l'image résultante.

$$X \circ K = (X \ominus K) \oplus K$$

L'ouverture de X par K peut être vue comme l'ensemble des points couverts par K lorsque K se déplace à l'intérieur de X (exemple FIGURE 3.7).

Les ouvertures sont utilisées pour enlever des petits objets[5], des protubérances d'objets plus importants ou des connexions entre objets. Par exemple, une ouverture par un carré de 5 x 5 pixels va enlever tous les éléments qui ont moins de 5 pixels de hauteur ou de largeur.

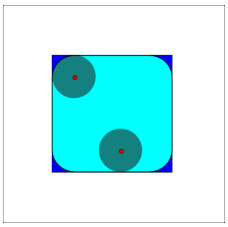


FIGURE 3.7 – Le carré bleu clair est le résultat de l'ouverture du carré bleu foncé par un disque.

Area Open

Une variante de l'ouverture est l'ouverture d'aire (*area open*). Cet opérateur ne prend pas d'élément structurant, et supprime simplement tous les objets qui sont composés de moins d'un certain nombre fixé de pixels.

3.3.4 Closing

La fermeture (*closing*) est obtenue par la dilatation de X par K, suivie de l'érosion par K de l'image résultante.

$$X \bullet K = (X \oplus K) \ominus K$$

La fermeture de X par K peut être vue comme l'ensemble des pixels non-couverts par K lorsque K se déplace à l'extérieur de X (exemple FIGURE 3.8).

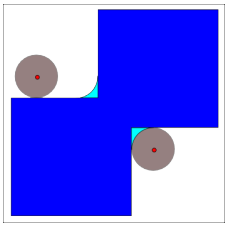


FIGURE 3.8 – En bleu (clair et foncé) le résultat de la fermeture du double carré en bleu foncé par un disque.

5. Objet peut se comprendre ici comme un ensemble de pixels connectés entre eux. Par défaut en 2D, un pixel peut avoir huit autres pixels connectés à lui.

Chapitre 4

Algorithmes développés

Dans ce chapitre, nous allons évoquer les différents algorithmes utilisés et implémentés. Nous commencerons par exposer la méthodologie globale et nous continuerons en approfondissant une à une les étapes décrites. Les algorithmes présentés seront illustrés par une implémentation en pseudo-code et par des exemples concrets d'images prises (volées) dans le cadre de ce travail.

Cependant, il convient d'abord de citer les quelques hypothèses fortes qui sont d'application.

- Seule l'identification de codes PIN et de schémas de déverrouillage sera considérée.
- La victime rentre son code ou son schéma avec le pouce ou l'index de sa main droite.
- Le vidéo est prise au moment où la victime tape son code ou son schéma.
- Le cas de la réflexion dans des lunettes de soleil est pris comme cas de base, bien que des résultats pour la réflexion dans l'œil seront proposés au chapitre suivant.
- Les valeurs chiffrées reprises dans les algorithmes en pseudo-code sont choisies pour des images capturées par un appareil photo de 13MP.

Nous discuterons des choix d'implémentation et de certaines hypothèses d'une part à la fin de chaque section pour tous les aspects algorithmiques, et d'autre part au chapitre suivant en détaillant leur influence sur les résultats que nous présenterons.

*Note 1 : Par souci d'esthétisme, certains graphes ont été modifiés (traits épaissis, image recoupée, etc) avant d'être intégrés comme illustration pour ce travail. Ce traitement a pour but de les rendre plus lisibles et plus compréhensibles pour le lecteur. Aucune de ces opérations n'a d'effet sur l'algorithme. Ces graphes seront signalés dans la légende par un *.*

Note 2 : Les algorithmes en pseudo-code présentés par la suite ont pour objectif d'aider à la bonne compréhension de leur fonctionnement et de leur implémentation. Leur implémentation en pseudo-code peut varier légèrement de leur réelle implémentation pour deux raisons. D'une part à cause de certaines des subtilités liées au langage de programmation utilisé (MatLab) et d'autre part, dans l'optique d'en simplifier la lecture.

Algorithme général

L'algorithme général proposé dans ce travail et illustré à la FIGURE 4.1 se rapproche assez de ce que Xu et al.[14] ont développé. Les fonctions de chacune des étapes peuvent paraître similaires, mais la différence avec Xu et al.[14] réside dans leurs implémentations qui ne sont que rarement semblables.

L'attaquant commence par prendre une vidéo de la victime au moment où elle tape son code ou son schéma (1). Il doit ensuite sélectionner une zone restreinte mais suffisamment large de l'image (comme le visage) où le reflet de l'écran apparaît (2). L'algorithme va y détecter les

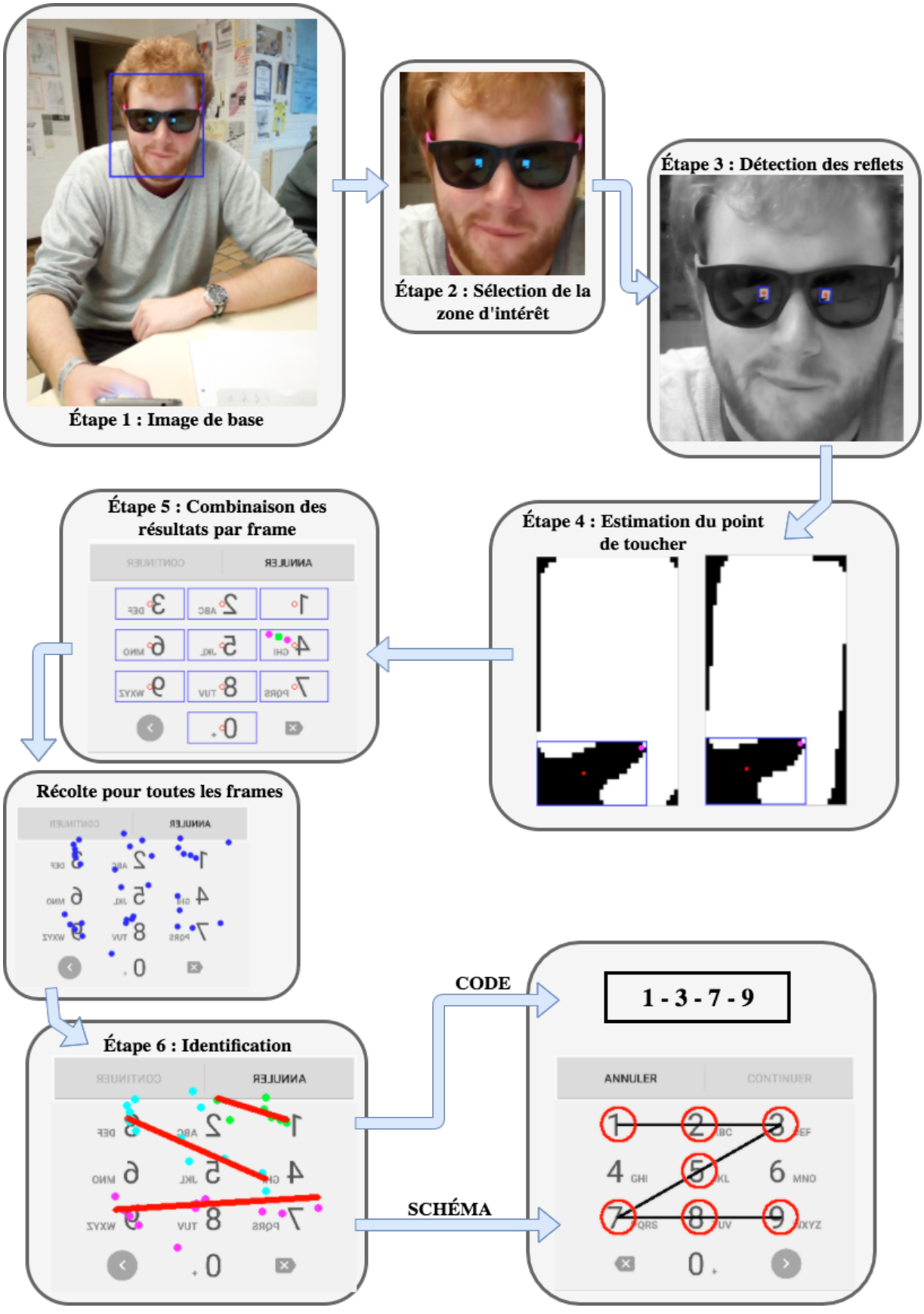


FIGURE 4.1 – Algorithme général

reflets potentiels de l'écran (3), pour ensuite extraire l'empreinte du doigt et identifier le point de contact (PC) avec l'écran (4). Les coordonnées du point de contact identifiées pour chaque reflet potentiel, l'algorithme va les associer pour estimer la touche pressée (5). Les étapes (3) à (5) sont répétées pour chaque frame de la vidéo et les résultats sont combinés afin de déterminer le code PIN ou le schéma rentré (6).

Étape 1 : Capture de la vidéo Cette phase peut paraître anodine mais elle est cruciale tant la qualité de la vidéo a une influence sur les résultats de l'algorithme. Il est nécessaire que la vidéo soit prise sous un angle où l'entièreté de l'écran se reflète dans les lunettes pour pouvoir y déceler de l'information.

Étape 2 : Sélection de la zone utile L'attaquant sélectionne manuellement une zone où apparaissent les reflets de l'écran, telle que le visage de la victime. La région choisie peut être plus large et ne doit pas se limiter à la tête. Le but de cette étape est d'accélérer l'exécution de la méthode et d'écarter de probables fausses détections d'écrans dans l'arrière-plan de la scène.

Étape 3 : Repérage des reflets de l'écran L'algorithme va repérer les candidats potentiels et sélectionner parmi eux les parties de l'image les plus susceptibles d'être un reflet du téléphone.

Étape 4 : Identification du point de contact avec l'écran Pour chaque sous-image jugée comme reflet probable, la méthode extrait l'empreinte du doigt et en déduit la position du toucher du doigt sur l'écran.

Étape 5 : Analyse des points de contact récupérés pour une même image Une fois le point de contact identifié dans chacun des reflets probables, l'algorithme va détecter la présence de faux positifs résiduels qui doivent être rejetés pour garder les véritables reflets. Il va ensuite combiner ces informations pour estimer la réelle position du point de contact dans la frame concernée.

Étape 6 : Identification du code ou du schéma Les étapes 3 à 5 sont exécutées pour toutes les frames de la vidéo et l'ensemble des points résultants est examiné pour déterminer le code ou le schéma tapé par la victime. La méthode utilisée n'est pas la même dans les deux cas de figure.

4.1 Capture de la vidéo

Objectif : Prendre une vidéo de qualité suffisante sans éveiller les soupçons de la victime.

L'attaquant doit veiller à l'angle de prise de la vidéo pour capturer le reflet en entier de l'écran. Cet angle dépend de paramètres différents tels que l'inclinaison de la tête de la victime, la courbure des lunettes de soleil, la position du téléphone, la distance entre l'attaquant et la victime, etc.

La qualité de la vidéo a aussi son importance, comme cela sera discuté au chapitre 5. La qualité du focus au niveau du visage et la netteté de l'image le long de la vidéo sont des facteurs qui influenceront positivement l'extraction d'informations par la suite.

4.2 Sélection de la zone utile

Objectif : Avoir tout au long de la vidéo une zone assez large pour que le reflet de l'écran y apparaisse et assez restreinte pour faciliter l'identification des reflets.

Manuellement l'utilisateur doit choisir une zone comprenant les reflets et facile à délimiter, comme par exemple le visage de la victime. Cette zone peut comprendre plus que la tête de la victime. Le but à ce stade est d'exclure un maximum l'arrière-plan où des faux positifs (comme des formes rectangulaires dans le décor) pourraient être repérés et d'accélérer le travail des étapes suivantes en restreignant la taille de l'image traitée.

4.2.1 Discussion

Cette étape est manuelle mais pourrait être assurée par un détecteur de visage par exemple. Ce genre de détecteurs est assez répandu dans les applications de photographie, et est souvent inclus dans les smartphones récents. Il serait même possible d'accélérer le procédé et d'améliorer encore plus les résultats avec un détecteur capable de repérer des lunettes de soleil ou des yeux.

L'objectif de ce travail n'étant pas d'assurer un quelconque *tracking*, on supposera que la zone utile choisie est assez large pour être valable pour toutes les frames de la vidéo, même si le visage de la victime bouge légèrement.

4.3 Repérage des reflets de l'écran

Objectif : Repérer les parties les plus susceptibles d'être un reflet de l'écran du téléphone.

Le reflet de l'écran dans l'œil ou dans des lunettes de soleil se traduit très généralement par une forme rectangulaire lumineuse contrastant avec le fond noir ou parfois teinté des lunettes ou de l'œil. Sa détection dans la scène filmée se fait en deux parties. Tout d'abord, l'image est traitée afin de trouver les formes qui s'y distinguent (par exemple avec un contraste important) et qui sont désignées comme candidats probables. Ensuite, ces candidats sont analysés afin de déterminer s'ils représentent un reflet de l'écran ou non. Cette deuxième étape se résume à vérifier si la forme analysée est rectangulaire. Cette dernière propriété sera détaillée par la suite.

4.3.1 Détection des candidats potentiels

Algorithme 1 Détection des candidats potentiels pour le reflet de l'écran

Entrée : une image couleur I qui correspond à la zone utile de l'image initiale
Sortie : l'ensemble S_I des images couleurs correspondant aux candidats détectés dans l'image I

1: $S_I \leftarrow \varnothing$
2: $I_g \leftarrow \mathsf{rgb2gray}(I)$
3: $I_b \leftarrow \mathsf{edgeCanny}(I_g)$ ▷ Détection des contours avec la méthode de Canny
4: $I_b \leftarrow \mathsf{areaOpen}(I_b, 20)$
5: $I_b \leftarrow \mathsf{fill}(I_b)$ ▷ Remplissage des contours fermés
6: $I_b \leftarrow \mathsf{open}(I_b, \mathrm{disk}, 6)$
7: $S_o \leftarrow \mathsf{detectObjects}(I_b)$
8: **for all** o **in** S_o **do** ▷ Récupération des régions dans l'image originale
9: $\quad S_I \leftarrow S_I \cup \mathsf{get}(I, o)$
return S_I

L'idée principale de l'algorithme 1 est d'essayer de repérer dans l'image les formes importantes qui contrastent le plus avec l'arrière-plan.

Dans cette optique, l'image est d'abord transformée en niveau de gris pour ensuite effectuer une détection de bords au moyen de la méthode de Canny [3]. L'image binaire résultante est une première fois débarrassée de toutes les petites structures grâce à une ouverture d'aire. Les contours fermés sont alors remplis et l'image passée par une ouverture par un disque de rayon de quelques pixels. Cette opération supprime tous les contours non-fermés et tous les petits objets qui étaient encore présents. Pour finir, les objets restants sont désignés comme candidats et sont récupérés dans l'image originale.

4.3.2 Éliminer les formes non-rectangulaires

Une fois les candidats potentiels détectés, il est nécessaire de trier ceux qui sont probablement des reflets de l'écran parmi les autres. Pour chaque région candidate, l'image est d'abord traitée de

(a) En niveau de gris (b)* Après détection de contour (c)* Après ouverture d'aire et remplissage (d) Après ouverture

FIGURE 4.2 – Détection des candidats potentiels.

Algorithme 2 Élimination des formes non-rectangulaires

Entrées : une image couleur I qui correspond à la zone utile de l'image initiale
Sorties : l'ensemble S_R d'images binaires correspondant aux candidats de forme rectangulaire

$S_r \leftarrow \varnothing$
for all I **in** S_I **do**
  $I_g \leftarrow \mathsf{rgb2gray}(I)$
  $I_a \leftarrow \mathsf{histoLin}(I_g)$ ▷ Ajustement linéaire de l'histogramme
  $I_b \leftarrow \mathsf{edgeCanny}(I_a)$
  $I_b \leftarrow \mathsf{areaOpen}(I_b, 30)$
  $isR \leftarrow \mathsf{isRectangle}(I_b)$
  if isR **then**
    $S_r \leftarrow S_r \cup I_b$
return S_r

manière similaire au point précédent afin de garder uniquement le contour de l'objet et d'enlever toutes les autres petites structures. Ensuite, le contour de la forme est étudié par la fonction isRectangle de façon à estimer la présence du caractère rectangulaire.

Vérification de la forme rectangulaire

Cet algorithme sert à discerner les formes rectangulaires de celles qui ne le sont pas. La difficulté dans cette partie réside dans l'allure variée des écrans. Ceux-ci ont une forme "à peu près" rectangulaire, sans avoir quatre angles droits. À cause de la déformation du reflet sur la surface incurvée, des transformations successives de l'image et surtout de la présence du doigt dans l'écran, l'image d'un reflet aura parfois des contours inclinés, parfois des légères imperfections sur les côtés ou encore un bord beaucoup plus petit que son opposé (exemples FIGURE 4.4 p.29).

Cette fonction va donc se focaliser sur trois parties du contour : le bord gauche, le bord supérieur et le bord droit. La forme du bord inférieur est trop souvent perturbée par la présence de la main de la victime que pour pouvoir en tirer des informations pertinentes. La méthode consiste à parcourir de haut en bas pour les bords latéraux et de gauche à droite pour le bord supérieur. La méthode démarre dans les coins supérieurs et s'arrête lorsqu'elle constate une variation trop importante de direction. Grâce aux positions de départ et d'arrêt, les longueurs et les angles des trois côtés sont calculés pour être comparés afin de décider si la forme a un aspect rectangulaire ou non.

Les critères de décisions sont :

1. Les longueurs des côtés doivent être supérieures à 5 pixels.

Algorithme 3 isRectangle

Entrées : une image binaire I_b qui correspond à un contour, h sa hauteur et w sa largeur
Sorties : un booléen b qui vaut *true* si la forme I_b remplit les critères pour être rectangulaire

$y \leftarrow \mathsf{maxY}(I_b(1 : \frac{w}{3}, \frac{2h}{3} : h))$ ▷ Neuvième supérieur gauche de l'image
$x \leftarrow \mathsf{minX}(I_b(:, y)$
$xy_{\text{startL}} \leftarrow (x, y)$
$xy_{\text{startU}} \leftarrow (x, y)$
$x \leftarrow \mathsf{maxX}(I_b(\frac{2w}{3} : w, \frac{2h}{3} : h))$ ▷ Neuvième supérieur droit de l'image
$y \leftarrow \mathsf{maxY}(I_b(x, :)$
$xy_{\text{startR}} \leftarrow (x, y)$
$xy_{\text{endL}} \leftarrow \mathsf{pathLeft}(I_b, xy_{\text{startL}})$ ▷ Parcours des trois bords
$xy_{\text{endU}} \leftarrow \mathsf{pathUp}(I_b, xy_{\text{startU}})$
$xy_{\text{endR}} \leftarrow \mathsf{pathRight}(I_b, xy_{\text{startR}})$
$A_L \leftarrow \mathsf{angle}(xy_{\text{startL}}, xy_{\text{endL}})$ ▷ Calcul des angles et vérification des conditions
$A_U \leftarrow \mathsf{angle}(xy_{\text{startU}}, xy_{\text{endU}})$
$A_R \leftarrow \mathsf{angle}(xy_{\text{startR}}, xy_{\text{endR}})$
$b_1 \leftarrow (y_{\text{endL}} - y_{\text{startL}}) > 5$ && $(y_{\text{endR}} - y_{\text{startR}}) > 5$ && $(x_{\text{endU}} - x_{\text{startU}}) > 5$
$b_2 \leftarrow |A_L - 90| < 20$
$b_3 \leftarrow |A_R - 90| < 20$
$b_4 \leftarrow |A_U| < 6$
$b_5 \leftarrow |A_L - A_R| < 10$
$b_{25} \leftarrow \mathsf{count}((b_2, b_3, b_4, b_5) == true) >= 3$
$b \leftarrow b_1$ && b_{25}
return b

2. Les angles des côtés gauche et droit doivent valoir 90° avec une marge de 20° [1]
3. L'angle du côté droit doit valoir 90° avec une marge de 20°.
4. L'angle du côté supérieur doit valoir 0° avec une marge de 6°
5. Les côtés gauche et droit doivent avoir le même angle avec une marge de 10°.

Pour que la forme soit considérée comme rectangulaire, le premier critère ainsi que trois des quatre critères suivants doivent être remplis.

L'algorithme 3 isRectangle calcule les positions de départ puis, après avoir récolté les positions d'arrêt, les angles et les longueurs des trois côtés. Grâce à ces informations, la fonction peut vérifier les conditions susmentionnées.

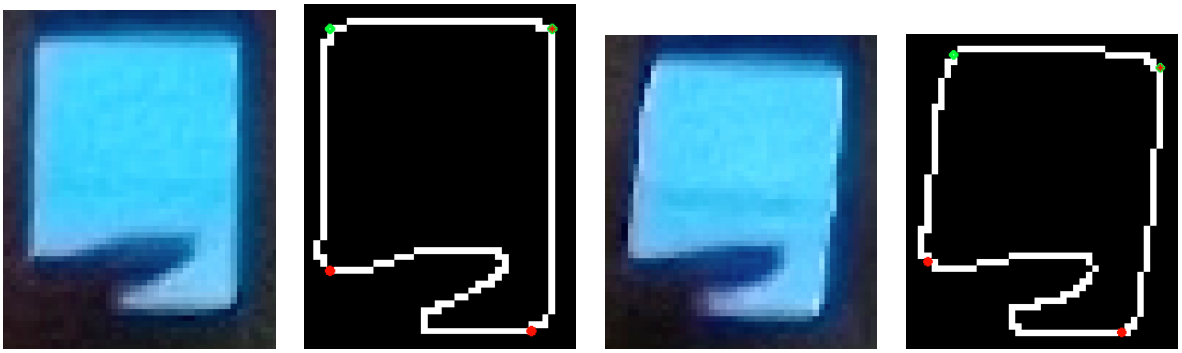
Une des fonctions de parcours du périmètre est présentée ici avec l'algorithme 4 pathLeft. Cette fonction peut être décrite comme suit :

- La méthode commence toujours par essayer de descendre tant que c'est possible.
- Si le contour remonte, la méthode s'arrête.
- Lorsque la méthode a dévié sur la gauche et sur la droite, elle s'arrête [2].

Cette dernière condition d'arrêt est utile dans les cas où le doigt ne remonte pas dans l'écran (voir **Cas particulier 1 : pouce plié**).

Le parcours du bord droit se fait de manière très similaire au bord gauche, tandis que pour le bord supérieur, le contour est parcouru de gauche à droite et s'arrête lorsque celui-ci commence à

1. L'algorithme travaille avec un nombre faible de pixels, ce qui fait qu'une différence d'un pixel peut parfois engendrer des variations de plus de 5°, d'où les marges importantes.
2. Dans l'algorithme en pseudo-code présenté ici la méthode s'arrête immédiatement lorsqu'elle a dévié une fois à gauche et une fois à droite, tandis que dans l'implémentation réelle un léger seuil de tolérance est appliqué.



(a) $A_L = 88,4°$; $A_R = 86,2°$; $A_U = 1,7°$ (b) $A_L = 84,6°$; $A_R = 81,7°$; $A_U = 5,4°$

FIGURE 4.3 – Parcours du contour des potentiels candidats : à chaque fois l'image originale à gauche et le résultat après transformations successives (étapes 3-6) à droite. En vert les xy_{start} et en rouge les xy_{end}

dévier verticalement. Afin de laisser le temps à l'algorithme d'être sur le bon côté avant de juger si le contour s'écarte de la direction convenue, une sorte de *slow start* est appliqué. Un certain délai est attendu au début et les premiers pixels du contour parcouru ne sont pas considérés pour la dernière condition d'arrêt, ce qui rend les calculs d'angle plus corrects lorsque la position initiale xy_{start} se situe sur un coin arrondi.

4.3.3 Discussion

Tout d'abord, nous avons pu constater que l'égalisation linéaire d'histogramme augmentait l'efficacité de la détection de contours plus que les autres méthodes d'amélioration de contrastes présentées au Chapitre 3. Ensuite pour la détection de bords, la méthode proposée par Canny a été choisie car elle donne de meilleurs résultats, surtout dans la détection de bords faibles par rapport à d'autres méthodes telles que Sobel, Roberts, etc. Rappelons encore que le choix du nombre de pixels pour les ouvertures dépend énormément de la résolution avec laquelle la vidéo a été prise et du type de réflexion (dans les lunettes de soleil ou dans l'œil). Les valeurs données dans ce chapitre sont adaptées à des photos prises à une résolution de 13 MP.

Les conditions de vérification d'une forme rectangulaire sont très certainement améliorables, bien qu'elles offrent actuellement des résultats corrects. Le choix des marges d'erreur des angles pourrait par exemple être peaufiné. Les conditions d'arrêt du parcours du contour sont certainement un des points qui demanderait le plus à être perfectionné afin de calculer des angles et des longueurs plus précises. Notons cependant qu'il est préférable d'écarter un écran que d'accepter un mauvais candidat. Comme nous travaillons avec des vidéos dont le *frame rate* se situe entre 20 et 30 frames par seconde, cela perturbe moins la suite de l'algorithme d'écarter l'un ou l'autre point de contact correct que de rajouter des points incorrects.

Une alternative pour faciliter cette partie de l'algorithme consisterait à appliquer un *tracking*. Si dans les premières frames de la vidéo nous pouvons arriver à estimer de manière presque certaine la position des reflets de l'écran, nous pourrions alors réduire davantage la zone d'intérêt et focaliser les tests sur la zone où l'écran a été détecté précédemment.

Pour finir, nous n'avons pas essayé d'implémenter ces méthodes avec l'aide du machine learning. Nous avons voulu entre autres montrer qu'il est possible d'implémenter l'algorithme en entier uniquement en utilisant du traitement d'images, ce qui rend la méthode rapide et facile d'utilisation. Il est néanmoins probable que certaines fonctions fourniraient de meilleurs résultats en ayant recours à du machine learning, cela sera discuté à l'aide des résultats présentés au Chapitre 5.

Algorithme 4 pathLeft

Entrées : une image binaire I_b qui correspond à un contour et xy_{startL} les coordonnées du point de départ

Sortie : xy_{endL} les coordonnées de la position d'arrêt

```
1:  xy ← xy_startL
2:  cnt = 3                                                    ▷ Slow start
3:  l ← false
4:  r ← false
5:  bool ← true
6:  while bool do                                              ▷ Côté gauche
7:      if isNextPosition(I_b(xy), 'down') then                ▷ ↓
8:          xy ← goto(xy, 'down')
9:      else if isNextPosition(I_b(xy), 'downRight') then      ▷ ↘
10:         xy ← goto(xy, 'downRight')
11:         if (!r && cnt < 0) then
12:             r ← true
13:     else if isNextPosition(I_b(xy), 'downLeft') then       ▷ ↙
14:         xy ← goto(xy, 'downLeft'))
15:         if (!l && cnt < 0) then
16:             l ← true
17:     else
18:         bool ← false
19:     if (r && l) then
20:         bool ← false
21:     cnt = cnt − 1
22: xy_endL ← xy return xy_endL
```

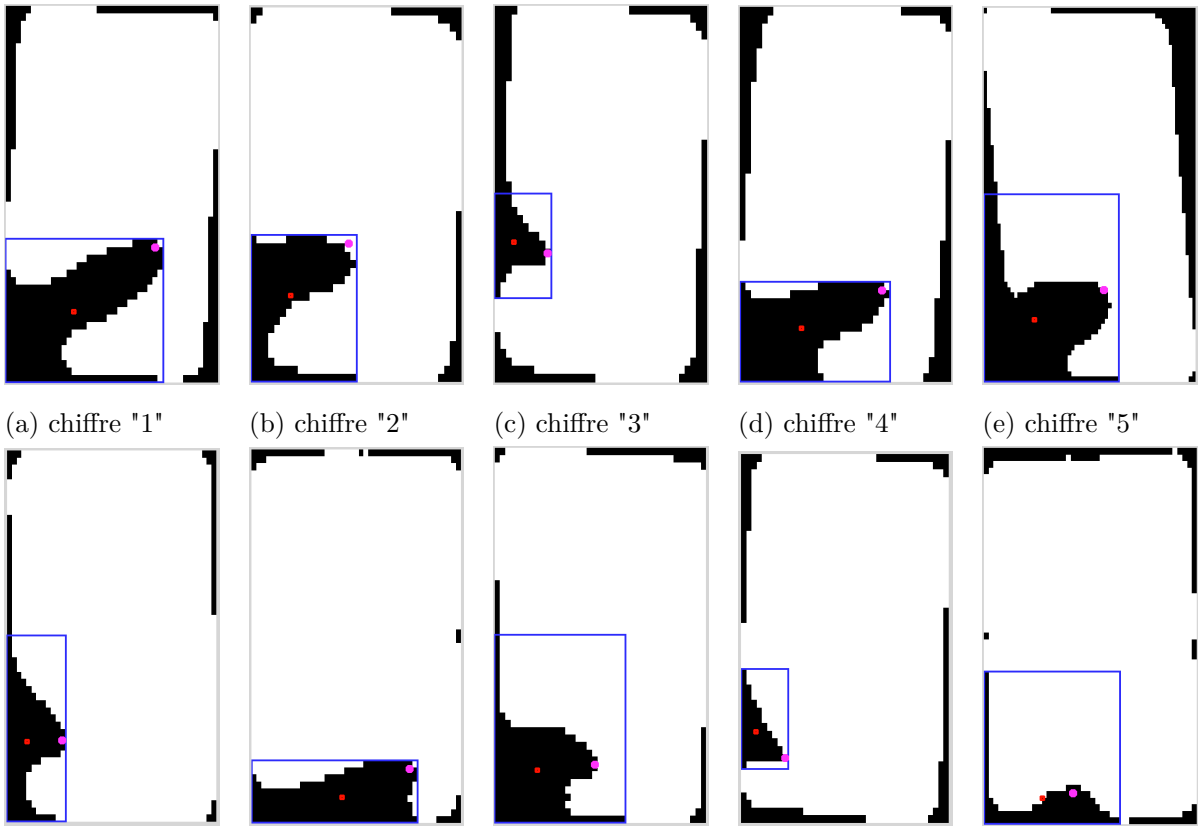
4.4 Identification du point de contact avec l'écran

Objectifs : Identifier l'empreinte du doigt dans l'image et en extraire le point de contact avec l'écran.

Les reflets de l'écran repérés, il est nécessaire de les analyser afin de trouver la partie qui est associée au doigt pour ensuite déterminer la position du toucher sur l'écran. Cette partie se décompose à nouveau en plusieurs étapes. Tout d'abord la région correspondant au doigt est détectée dans l'image. Ensuite le contour du doigt est parcouru une première fois en partant du coin supérieur gauche et une seconde fois en partant du coin inférieur droit. La combinaison de ces deux chemins permettra d'estimer la position du bout du doigt en contact avec l'écran.

Les difficultés à cette étape sont multiples. Tout d'abord, la forme de l'ombre du doigt sera très différente en fonction de la touche pressée (voir FIGURE 4.4). Ensuite, il arrive que l'ongle du doigt soit fort clair et soit associé aux pixels de l'écran lors de la détection de contours. Cela rend plus ardue l'estimation du point de contact avec l'écran. Enfin, une dernière difficulté est rencontrée lorsque la touche "0" est pressée. La partie du doigt s'immisçant dans l'écran est minime et située sur le côté inférieur, ce qui la rend plus difficile à discerner par rapport au bruit régulièrement présent aux bords de l'image. Les méthodes proposées dans cette section tentent d'apporter un maximum de solutions à ces obstacles.

Rappelons également que l'algorithme fonctionne sous l'hypothèse que les données sont tapées par le pouce ou l'index de la main droite de l'utilisateur. Les autres cas de figure seront discutés ultérieurement.



(a) chiffre "1" (b) chiffre "2" (c) chiffre "3" (d) chiffre "4" (e) chiffre "5"

(f) chiffre "6" (g) chiffre "7" (h) chiffre "8" (i) chiffre "9" (j) chiffre "0"

FIGURE 4.4 – Exemples pour chaque chiffre de la forme de l'écran. En bleu le périmètre de la zone sélectionnée pour le doigt, en rouge son centre de masse et en mauve l'estimation du point de contact.

4.4.1 Détection du doigt

Pour gagner en précision lors de l'estimation du point de contact avec l'écran, il est utile d'agrandir le reflet de l'écran, de l'adapter aux dimensions du modèle du smartphone utilisé par la victime si celui-ci est connu (cette hypothèse sera discutée par la suite). Avant d'être agrandi, le contour de l'écran est rempli. Après cette étape, il se peut qu'il y ait présence de bruit sur le bord de l'image, dû par exemple à une déformation du reflet de l'écran ou à son inclinaison. L'analyse et la comparaison de ces différentes régions de l'image agrandie permettent de choisir celle qui est la plus propice à correspondre au doigt.

Les conditions de sélection pour choisir la région correspondant à l'écran sont les suivantes :

1. Seule la moitié inférieure de l'image est considérée.
2. L'objet avec l'aire la plus importante est désigné.
3. Il faut que le centre de masse soit situé dans les 3/4 gauche de l'image[3] (cas d'un droitier).

4.4.2 Parcours du contour du doigt

Une fois que la forme représentant le doigt a été mise en évidence, il est nécessaire d'en extraire l'extrémité du doigt. Pour être le plus généraliste et le plus efficace possible, l'algorithme

3. Le reflet inverse la partie gauche et la partie droite, comme un *"effet-miroir"*.

Algorithme 5 Détection du point de contact

Entrées : une image binaire I_b qui correspond à un contour de l'écran, h_K la hauteur et w_K la largeur de l'écran du téléphone utilisé

Sorties : des coordonnées xy qui correspondent à l'estimation du point de contact avec l'écran

$I_b \leftarrow \mathsf{fill}(I_b)$ ▷ Détection du doigt
$I_b \leftarrow \mathsf{resize}(I_b, [w_K\ h_K], \text{'nearest'})$
$S_o \leftarrow \mathsf{detectObjects}(I_b(:, 1 : \frac{h_K}{2}))$
$bool \leftarrow true$
while *bool* **do**
    $o_{max} \leftarrow \mathsf{getMaxArea}(S_o)$
    if $\mathsf{getCentroidX}(o_{max}) < \frac{3w_K}{4}$ **then**
        $bool \leftarrow false$
    else
        $S_o \leftarrow S_o \backslash o_{max}$
$I_f \leftarrow \mathsf{boundaries}(o_{max})$
$x \leftarrow \mathsf{min}(I_f(:, \frac{h_K}{2} : h_k))$ ▷ Moitié supérieure de l'image
$y \leftarrow \mathsf{max}(I_f(x, :))$
$xy_{\text{startX}} \leftarrow (x, y)$
$y \leftarrow \mathsf{min}(I_f(\frac{w_K}{4} : w_k), :)$ ▷ 3/4 à droite de l'image
$x \leftarrow \mathsf{max}(I_f(:, y))$
$xy_{\text{startY}} \leftarrow (x, y)$
$[x_{\text{cut}},\ isFind,\ xy_{\text{findX}}] \leftarrow \mathsf{pathX}(I_f, xy_{\text{startX}})$ ▷ Parcours du contour
if $isFind$ **then** ▷ Cas particuliers 1, le PC est déjà identifié
    $xy_{\text{pos}} \leftarrow xy_{\text{findX}}$
else
    $[y_{\text{cut}},\ isFind,\ xy_{\text{findY}}] \leftarrow \mathsf{pathY}(I_f, xy_{\text{startY}})$
    if $isFind$ **then** ▷ Cas particuliers 2, le PC est déjà identifié
        $xy_{\text{pos}} \leftarrow xy_{\text{findY}}$
    else
        $x_{\text{pos}} \leftarrow \mathsf{maxX}(I_f(x_{\text{cut}} : w_K, y_{\text{cut}} : h_K) - 40$
        $y_{\text{pos}} \leftarrow \mathsf{maxY}(I_f(x_{\text{cut}} : w_K, y_{\text{cut}} : h_K) - 40$
        $xy_{\text{pos}} \leftarrow (x_{\text{pos}}, y_{\text{pos}})$
return xy_{pos}

se base sur deux constats importants : la majorité des utilisateurs utilisent leur pouce ou leur index pour introduire un code (cette hypothèse sera discutée par après) et dans le cas de l'index, le doigt est en extension pour chacune des touches ce qui n'est pas le cas du pouce. Les touches les plus proches du pouce ("3", "6" et "9" dans le cas d'un droitier, exemples FIGURE 4.4) demandent à ce que le doigt soit plié plutôt que tendu.

L'algorithme de parcours du contour a pour objectif de déterminer une abscisse et une ordonnée "de coupe" qui serviront à isoler l'extrémité du doigt de sa base et du bruit situé aux bords de l'image. Le premier parcours débutant dans le coin supérieur gauche est représenté par l'algorithme 6 pathX et est semblable à ce qui est proposé par pathLeft. L'algorithme pathY est similaire à pathX.

Doigt tendu

Lorsqu'il est en extension, on remarque que l'extrémité du doigt se situe dans la partie supérieure droite. L'algorithme va d'abord partir de la limite du pourtour supérieur gauche pour descendre jusqu'au moment où il sera obligé de remonter. Ce moment indique qu'il a quitté la

base du doigt ou le bruit présent en périphérie. Ce point d'arrêt détermine l'abscisse "de coupe". Ensuite l'algorithme démarrera de la limite inférieure droite et remontera le contour jusqu'au moment où il devra se diriger vers la droite. De manière similaire au premier parcours, cette position caractérise le début du doigt et définit l'ordonnée "de coupe".

Cas particulier 1 : pouce plié

Lorsque le pouce est plié, son extrémité n'est plus du tout située en haut à droite. Cependant, on observe que cela crée une sorte de légère "bosse" aux alentours de la touche (voir FIGURE 4.4(c), (f) et (i)). L'algorithme de parcours pathX va détecter cette bosse lorsqu'en partant du haut, il devra se diriger considérablement vers la gauche après s'être écarté suffisamment sur la droite de sa position initiale.

Dans ce cas de figure, la position du point de contact est directement définie comme étant la hauteur à la moitié de la bosse pour son ordonnée et l'endroit d'arrêt pour son abscisse.

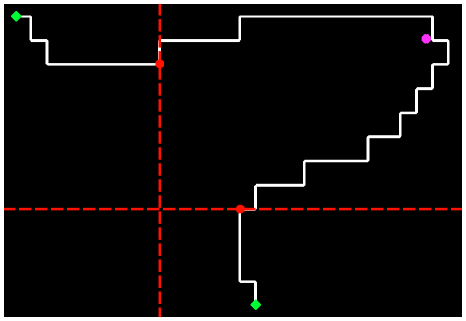


FIGURE 4.5 – En vert position de départ, en rouge position d'arrêt, en mauve estimation du PC. Les lignes illustrent l'ordonnée et l'abscisse "de coupe"

Cas particulier 2 : chiffre "0"

Dans certaines situations, nous pouvons constater un phénomène comparable à celui décrit ci-dessus lorsque la touche "0" est pressée. La bosse n'est alors plus verticale mais horizontale. Elle est détectée par la fonction pathY au moment où l'algorithme devra redescendre après s'être suffisamment élevé.

Les coordonnées du point de contact sont également directement définies par l'abscisse du milieu de la bosse et quelques pixels légèrement en-dessous de l'ordonnée de son sommet.

4.4.3 Estimation du toucher du doigt

À supposer que la position de contact n'ait pas déjà été fixée dans un des cas particuliers et une fois les coordonnées "de coupe" obtenues, l'algorithme sélectionne la région de l'image située à droite de x_{cut} et au-dessus de y_{cut}. Dans cette zone, on détermine l'abscisse du point le plus à droite $x_{\max}$ et l'ordonnée du point le plus élevé $y_{\max}$.

Les coordonnées du point de contact sont alors définies comme

$$(x_{\text{pc}}, y_{\text{pc}}) = (x_{\max} - 40,\ y_{\max} - 40)$$

On peut visualiser cette définition mathématique comme un léger rapprochement vers l'intérieur de ce qui serait le coin supérieur droit du doigt.

4.4.4 Discussion

La première hypothèse forte est la manière dont l'utilisateur saisit les données. L'algorithme ne fonctionne que dans le cas où la victime rentre le code de la main droite. Il serait cependant facilement adaptable au cas d'un gaucher. Il faudrait soit implémenter un nouvel algorithme basé sur un "retournement" du précédent (remplacer les "gauches" par des "droites" et vice-versa), soit appliqué le même algorithme en changeant au préalable la symétrie de l'image . Ainsi par exemple la fonction pathX commencerait dans le coin supérieur droit se dirigerait vers la droite. Nous aurions alors une fonction pour droitiers et une fonction pour gauchers. Pour déterminer laquelle des deux choisir, nous pourrions examiner la position du centre de masse de l'objet dont l'aire est maximale lors du repérage de la région du doigt. Il serait par contre plus difficile de repérer un mauvais candidat si la zone du doigt n'est pas celle qui a l'aire la plus grande.

Algorithme 6 pathX

Entrées : une image binaire I_b qui correspond à un contour et xy_{startL} les coordonnées du point de départ

Sortie : xy_{endL} les coordonnées de la position d'arrêt

Paramètres :*cnt* le délai pour le slow start, il est aussi possible d'ajouter un compteur pour l et r afin de laisser une plus grande tolérance

```
1: xy ← xy_startL
2: cntV ← 0
3: isFind ← false
4: bool ← true
5: while bool do
6:     if isNextPosition(I_b(xy), 'down') then                    ▷ ↓
7:         xy ← goto(xy, 'down')
8:         cntV ← cntV + 1
9:     else if isNextPosition(I_b(xy), 'downRight') then          ▷ ↘
10:        xy ← goto(xy, 'downRight')
11:        cntV ← 0
12:    else if isNextPosition(I_b(xy), 'right') then              ▷ →
13:        xy ← goto(xy, 'right'))
14:        cntV ← 0
15:    else if isNextPosition(I_b(xy), 'upRight') then            ▷ ↗
16:        x_cut ← x
17:        bool ← false
18:    else
19:        isFind ← true
20:        xy_find ← (x, y + cntV/2)
21:        bool ← false
   return [x_cut, isFind, xy_find]
```

Ensuite nous avons supposé que l'utilisateur utiliserait son pouce ou son index pour rentrer son code ou son schéma. De par nos observations, nous avons constaté qu'il était rare de voir des utilisateurs utiliser un autre doigt. Quant bien même cela ne devrait a priori pas poser de problèmes car la forme du doigt dans le reflet de l'écran devrait être la même. Il existe aussi de nombreuses situations où l'utilisateur utilise ses deux pouces pour taper sur son clavier. Ce cas de figure n'a pas été étudié. Il devrait être possible de repérer les deux doigts dans l'écran et d'estimer les deux points de contacts, par contre il serait plus difficile d'arriver à combiner correctement par la suite les résultats obtenus (quel doigt est actif quand, etc).

Nous avons fait le choix de définir le point de contact du doigt avec l'écran comme un point situé légèrement en-dessous du coin supérieur droit du doigt. Cette approche apporte une solution simple à un problème complexe. En effet, nous avons tenté d'observer cette partie du doigt lorsqu'elle rencontre l'écran, mais il est difficile de tirer des conclusions valables en toutes circonstances. Nous avons pu observer certaines tendances : quand le pouce doit être précis lors du toucher (ex : clavier azerty) l'utilisateur a naturellement tendance à l'incliner pour réduire la zone de contact et au contraire lorsque qu'il balaye l'écran la zone de contact sera plus située au bout du doigt (surface plus large).

Notre algorithme ne cherche pas à redresser un écran incliné. Cela n'a pas beaucoup d'impact pour un code PIN ou un schéma tant que la vidéo est bien alignée verticalement par rapport à la victime.

4.5 Analyse des points de contact récupérés pour une même image

Objectifs : Combiner les coordonnées récupérées au sein d'une même frame de façon à affiner le résultat et exclure les faux positifs.

Il est fréquent que plusieurs candidats soient désignés comme des formes rectangulaires. Plusieurs cas de figure sont alors possibles : soit l'algorithme a réussi à identifier le reflet de l'écran dans les deux verres de lunettes, soit il reste des faux positifs à écarter. Il existe aussi un cas un peu dans lequel l'écran est correctement détecté mais à un moment où l'utilisateur ne rentre pas de données, l'écran est donc "vide". Cette situation sera ici considérée comme à rejeter (cela sera discuté par la suite) car le point de contact qui y serait déterminé n'aurait aucun sens et parasiterait le reste de l'algorithme.

Cette section aura une double fonction : écarter les faux positifs et au besoin réduire les résultats à une paire de coordonnées par frame.

4.5.1 Écarter le bruit

Algorithme 7 Analyse des points de contacts pour une même frame

Entrées : un ensemble de points de contact set_{pc} correspondant à une même frame et une structure du modèle du clavier utilisé $KeyB$

Sortie : le chiffre estimé $digit_{\text{final}}$ et les coordonnées de la position $coord_{\text{final}}$

$S_d \leftarrow \varnothing$
for all pc **in** set_{pc} **do**
    if notFalsePositive$(pc, KeyB)$ **then**
        $S_d \leftarrow S_d \cap$ getCloserDigit$(pc.XY)$
$S_d \leftarrow$ sortBy$(S_d,$ 'distance'$)$
$d1 \leftarrow S_d(1)$
$d2 \leftarrow S_d(2)$
$d3 \leftarrow S_d(3)$
if $digit1 == digit2$ **then**
    $digit_{\text{final}} \leftarrow d1.digit$
    $coord_{\text{final}} \leftarrow$ mean$(d1.XY,\ d2.XY)$
else if $digit1 == digit3$ **then**
    $digit_{\text{final}} \leftarrow d1.digit$
    $coord_{\text{final}} \leftarrow$ mean$(d1.XY,\ d3.XY)$
else if $digit2 == digit3$ **then**
    $digit_{\text{final}} \leftarrow d2.digit$
    $coord_{\text{final}} \leftarrow$ mean$(d2.XY,\ d3.XY)$
else
    $digit_{\text{final}} \leftarrow d1.digit$
    $coord_{\text{final}} \leftarrow d1.XY$
return $[digit_{\text{final}},\ coord_{\text{final}}]$

La méthode de discernement se base sur les observations suivantes : les faux positifs sont souvent des formes rectangulaires mais à la différence des reflets ils n'ont pas la même incursion caractéristique du doigt dans le rectangle. Ainsi leurs points de contact se situeront généralement à proximité d'un des bords du rectangle.

L'algorithme va analyser les coordonnées de chaque point de contact précédemment trouvé. Si le point se situe trop près d'un des bords, il est éliminé. Ensuite grâce à la connaissance du

modèle du clavier, nous avons pu établir les zones de toucher "actives" propres à chaque chiffre et le centre de ces zones. Un point trop éloigné de ces zones sera aussi écarté.

4.5.2 Réduire à une paire de coordonnées

Afin d'estimer une position globale du toucher, on associe à chaque point de contact le chiffre dont il est le plus proche. On trie ensuite les données en fonction de leur distance par rapport à la distance avec le centre de la zone de ce chiffre. En considérant que plus une coordonnée est proche d'un centre, plus il est probable cette donnée soit valide, l'algorithme va examiner les trois données les plus proches d'un centre de zone "active".

L'algorithme va alors discerner deux cas de figure :

1. Soit deux chiffres sont les mêmes parmi les trois, alors la position retournée sera la moyenne de ces deux positions.
2. Soit le premier chiffre est considéré comme étant correct et sa position est retournée.

Ces conditions traduisent la présence d'une donnée erronée parmi les trois et s'appliquent également lorsqu'il y a moins de trois données récupérées.

4.5.3 Discussion

Écarter les points de contacts sur les bords et hors des zones actives, qu'il s'agisse de bruit ou non, ne porte pas trop à préjudice même si la donnée est valide. Les points situés trop près des bords n'apportent pas beaucoup d'informations par rapport aux points situés plus à l'intérieur du clavier. À nouveau, précisons que cette affirmation est fondée car nous disposons d'assez d'images par vidéos, grâce à des *frame rate* importants.

Ce n'est pas le même cas de figure lorsque nous rencontrons un écran vide. Cet écran peut traduire la fin d'un mot, d'un code, ou encore un changement d'évènement considérable (chargement d'une page, remplissage d'un autre champ lexical, etc) et peut donc apporter une information à l'attaquant. Cependant pour pouvoir utiliser ces indications, il faudrait différencier de manière plus rigoureuse les faux positifs de ce genre d'écrans vides pour s'assurer de ne pas rajouter des données parasites dans la suite de l'algorithme. L'utilisation d'une méthode de *tracking* pourrait également convenir dans de telles circonstances.

4.6 Identification du code ou du schéma

Objectif : Traduire un ensemble de points de contact en un code PIN ou un schéma de déverrouillage.

À ce stade, l'algorithme a identifié une série de points de contact qu'il a pu trier en fonction du numéro de frame dans la vidéo. Pour déterminer le code PIN ou le schéma de déverrouillage, nous avons développé deux algorithmes semblables mais distincts. Pour améliorer l'efficacité de l'algorithme, chacun des algorithmes essaye de prendre en considération les spécificités liées aux deux cas de figure.

Codes PIN Nous avons observé que la victime restait plus longtemps sur les touches pressées et beaucoup moins longtemps sur les touches situées entre deux chiffres choisis où le doigt survole l'écran. De plus, l'utilisateur peut presser plusieurs fois sur la même touche.

Schéma de déverrouillage Bien que le doigt de la victime reste plus longtemps sur les touches marquant un changement de direction, il effectue plus lentement les passages intermédiaires que pour le code PIN, car son doigt maintient le contact avec l'écran et en est en quelque sorte freiné. Il n'est également pas possible de changer plusieurs fois de direction au même chiffre.

(a) Ensemble des données (b) Division en clusters et approximation par une droite (c) Illustration du schéma estimé

FIGURE 4.6 – Différentes étapes pour déterminer le schéma de déverrouillage.

4.6.1 Schéma de déverrouillage

Algorithme 8 Analyse des points de contact pour renvoyer un schéma

Entrées : un ensemble de données *dataset* correspondant aux points de contact en 3D (2D + t) et *weightTime* le poids qu'on souhaite attribuer à la dimension temporelle
Sortie : les chiffres "extrémités" *schema* correspond à un schéma

$dataset(3) \leftarrow dataset(3) * weightTime$ ▷ Rajoute le poids désiré à la variable temporelle
$setC \leftarrow \mathsf{kmeans}(dataset, 3)$
for $k = 1$ **to** 3 **do**
  $set_{\text{best}}(k) \leftarrow \varnothing$
  $model_{\text{best}}(k) \leftarrow \varnothing$
  $error_{\text{best}}(k) \leftarrow \infty$
  for $i = 1$ **to** 2 **do**
    $[set,\ model,\ error] \leftarrow \mathsf{ransac}(setC(k), N, n, error_{\max}, l_{\min})$
    if $error < error\text{best}(k)$ **then**
      $set_{\text{best}}(k) \leftarrow set$
      $model_{\text{best}}(k) \leftarrow model$
      $error_{\text{best}}(k) \leftarrow error$
  $digit(2 * k - 1 : 2 * k) \leftarrow \mathsf{line2digit}(set_{\text{best}}(k), model_{\text{best}}(k))$
$schema \leftarrow reduceDigit(digit)$ ▷ Renvois les chiffres "extrémités"
return *schema*

L'idée de cet algorithme est de décomposer l'ensemble de données en clusters, d'approximer ensuite les données de chaque cluster par un segment de droite, et de finalement associer les différents segments pour déduire le schéma. Ces étapes peuvent être répétées afin de minimiser l'erreur et d'augmenter les chances de trouver le schéma correct.

Concrètement, comme par hypothèse le schéma est limité à quatre chiffres "extrêmes" (les chiffres marquant un changement de direction) ou trois changements de direction, le nombre de clusters est fixé à trois. L'approximation des ensembles de points par une droite est réalisée grâce à l'algorithme 9 basé sur la méthode RANSAC [6].

RANSAC

Nous présentons avec l'algorithme 9 une implémentation de la méthode RANSAC [6]. L'intérêt de cette méthode est qu'elle est construite de manière à écarter efficacement les valeurs aberrantes (*outliers*), qui correspondraient dans notre cas aux points estimés à partir de mauvaises détections

Algorithme 9 ransac

Entrée : un ensemble de données *dataset*
Sorties : l'ensemble de données correspondant au meilleur modèle set_{best}, les paramètres du meilleur modèle $model_{\text{best}}$ et l'erreur associée à ce modèle $error_{\text{best}}$
Paramètres : le nombre d'itérations à accomplir $N = 1000$, un nombre minimum de données pour établir le modèle $n = 3$, l'erreur maximale tolérée $error_{\max} = 100$ et un nombre minimum de données pour valider le modèle $l_{\min} = length(dataset)/2$

```
1: set_best ← ∅
2: model_best ← ∅
3: error_best ← ∞
4: for i = 1 to N do
5:     data_r ← selectRandom(dataset, n)
6:     model ← approxLin(data_r)
7:     for all data in dataset \ data_r do
8:         if error(model, data) < error_max then        ▷ Distance d'un point à une droite
9:             data_r ← data_r ∪ data
10:    if length(data_r) > l_min then
11:        model ← approxLin(data_r)
12:        err ← mse(model, data_r)                      ▷ Erreur quadratique moyenne
13:        if err < error_best then
14:            set_best ← data_r
15:            model_best ← model
16:            error_best ← err
   return [set_best, model_best, error_best]
```

de l'écran. L'idée de l'algorithme est de choisir au hasard n données dans l'ensemble *dataset*, d'établir un modèle à partir de ces données (approxLin), puis de rajouter tous les points qui correspondent au modèle avec une erreur inférieure à l'erreur maximale autorisée ($error_{\max}$). Finalement, si le nombre total de données initiales et rajoutées est supérieur au seuil minimal $l_{\min}$, un modèle est recalculé à partir de ces données. Ces étapes sont répétées un nombre N de fois et le meilleur modèle est défini par l'erreur minimale obtenue. Un exemple de la méthode est présenté à la FIGURE 4.6(b).

Ces variables sont fixées dans notre algorithme, à titre d'exemple nous avons mis dans la section **Paramètres** les valeurs utilisées dans le cadre de ce travail.

4.6.2 Code PIN

Nous allons d'abord différencier deux cas de figure : les codes où le même chiffre ne revient pas plusieurs fois, et les codes où le même chiffre apparaît plus d'une fois. Le principe de notre algorithme est le suivant.

D'abord il va considérer le code comme si c'était un schéma et accorder un poids moyen à la dimension temporelle, ce qui lui permettra de trouver un maximum de quatre chiffres "extrémités". S'il y en a quatre, il est probable que cela corresponde au code. Dans tous les cas, l'algorithme va ensuite accorder un poids plus conséquent à la dimension temporelle et va déterminer quatre clusters. Pour chacun des quatre clusters, il constituera l'ensemble de tous les points à moins d'une certaine distance fixée du centre du cluster, ensuite il calculera la position moyenne de ces points et l'associera à un chiffre. Il renverra ces quatre chiffres ainsi que le schéma déterminé pour que l'attaquant puisse comparer les deux.

Cet algorithme n'est certainement pas le plus rigoureux mais tente de présenter une méthode simple et efficace pour approximer le code rentré par l'utilisateur. Située à la fin de l'algorithme

Algorithme 10 Analyse des points de contact pour retourner un code PIN

Entrées : un ensemble de données *dataset* correspondant aux points de contact en 3D (2D + t), $weightTime$ le poids qu'on souhaite attribuer à la dimension temporelle et $dist_{\min}$ la distance minimale pour former les sous-ensembles de clusters

Sorties : les chiffres "extrémités" *schema* et un code de quatre chiffres *digit*

$dataset(3) \leftarrow dataset(3) * weightTime$ ▷ Adapte le poids désiré pour le temps
$schema \leftarrow \mathsf{data2schema}(dataset)$
$dataset(3) \leftarrow dataset(3) * 2$
$[setC,\ C] \leftarrow \mathsf{kmeans}(dataset, 4)$
for $k = 1$ **to** 4 **do**
    $subsetC(k) \leftarrow \varnothing$
    for all $data$ **in** $setC(k)$ **do**
        if $\mathsf{dist}(data, C(k)) < dist_{\min}(k)$ **then**
            $subsetC(k) \leftarrow subset(k) \cup data$
    $meanC(k) \leftarrow \mathsf{mean}(subsetC(k))$ ▷ Calcule la position 2D moyenne
    $digit(k) \leftarrow data2digit(meanC(k))$ ▷ Détermine le chiffre le plus proche du point
    return $[schema,\ digit]$

général, notre méthode ne doit pas nécessairement aboutir pas à une unique réponse. Comme cette information à l'usage de l'attaquant tient compte de ce manque de rigueur, il n'est pas dérangeant que l'algorithme retourne un double renseignement.

4.7 Conclusion

Nous avons développé un algorithme semi-automatique basé presque essentiellement sur du traitement d'image, permettant d'obtenir un code ou un schéma à partir d'une simple vidéo filmée par un smartphone. Nous avons discuté des outils manquants pour rendre l'algorithme totalement automatique et expliqué quelques un de nos choix d'implémentation. Il est maintenant temps de présenter les résultats de notre méthode au chapitre suivant.

Chapitre 5

Résultats

Ce chapitre présente les résultats de notre méthode sur les différentes images recueillies dans le cadre de ce travail. Nous essayerons de pointer les forces et les faiblesses de l'algorithme, ainsi que les situations favorables comme défavorables à son efficacité.

5.1 Méthode de validation

Avant de présenter les résultats, il convient de définir plus rigoureusement l'environnement dans lequel nous avons travaillé pour collecter les données, les moyens mis en place pour garantir l'adaptation de notre méthode à une variété importante de situations et également la constitution des différents *dataset*.

5.1.1 Description du setup expérimental

Cette partie décrit le protocole d'acquisition des images pour lequel nous avons tenté de nous rapprocher le plus possible de situations réelles. Même si nos "victimes" étaient tout à fait au courant des circonstances, nous leur avons demandé de ne pas changer leur comportement habituel, en particulier la position du smartphone par rapport à leur corps, leur façon de tenir le téléphone, la manière employée pour appuyer sur l'écran, l'inclinaison de leur tête pour regarder l'appareil, etc.

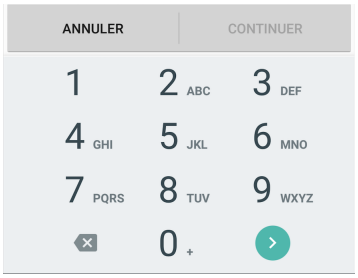


FIGURE 5.1 – Clavier virtuel du UMI Touch.

1. Le modèle du téléphone utilisé par la victime est un UMI Touch, l'écran mesure 5,5' (environ 14 cm).
2. Le clavier virtuel employé (voir FIGURE 5.1) est le clavier digital standard proposé par Android.
3. L'attaquant capture les photos/vidéos au moyen de l'appareil photo arrière d'un Huawei YII P6. La résolution des photos est de 13 MP (4160 x 3120 px) et des vidéos de 720p [1] (1280 $\times$ 720 px).
4. La distance entre l'attaquant et la victime varient entre 1 m et 4 m.
5. Sauf indication contraire, la victime porte des lunettes de soleil.
6. La victime saisit son code à l'aide du pouce de sa main droite.
7. L'attaquant prend la photo/vidéo en restant le plus "naturel" possible ou du moins, sans paraître "suspect". Il garde ses mains à proximité de la table et joue sur l'inclinaison du téléphone pour capturer le visage de la victime.

1. Pour plus d'informations, voir `https://en.wikipedia.org/wiki/720p`

(a) (b) (c) (d) (e)

FIGURE 5.2 – Exemples de différentes teintes de verres de soleil utilisés. (a), (b) et (c) sont les verres de teinte "classique" et (d) et (e) de type "miroir".

Spécificités pour les images isolées

Nous avons commencé par tester notre algorithme sur des images prises de manière indépendante (par opposition aux images provenant d'une même vidéo). Nous avons demandé à la victime de garder son doigt sur la touche concernée le temps de prendre la photo. L'attaquant peut aider le focus de l'appareil photo en désignant la zone sur son écran. Il ne le fait pas à chaque fois pour varier le *dataset*.

Spécificités pour les vidéos

Lors de l'acquisition des vidéos, nous avons soumis la victime à quelques contraintes légères afin de faciliter le travail d'identification de code de notre algorithme. Tout d'abord nous nous sommes restreints aux codes composés de quatre chiffres. Ensuite, nous avons demandé à la victime de taper les quatre chiffres à la suite et sans dévier son pouce du chemin "logique". Par chemin "logique", nous excluons le passage par des positions intermédiaires qui ne seraient pas sur la trajectoire directe entre deux chiffres du code. Cette contrainte cherche à proscrire les positions d'"attente" comme le centre du clavier où la victime "reposerait" son doigt le temps de réfléchir au prochain mouvement. Par exemple si le code est 1 - 3 - 6 - 8, la victime va directement vers la touche "3" après avoir pressé la touche "1". Elle ne passe pas par le "5" avant de se diriger vers le prochain chiffre.

Lorsqu'elle a fini de taper son code, soit la victime retire son doigt de l'écran soit elle presse la touche "enter". Les deux cas sont fondés, certains téléphones vérifient automatiquement le code lorsque le nombre suffisant de caractères est atteint (ex : iPhone) et d'autres attendent une confirmation par la touche "enter" (ex : la plupart des modèles Android).

5.1.2 Robustesse de la méthode

Pour s'assurer que la méthode développée soit robuste et puisse s'appliquer à une variété importante de cas de figure distincts, nous avons tenté de diversifier notre *dataset*. Comme l'**étape 2** de notre algorithme demande à l'attaquant de sélectionner une zone utile qui englobe généralement le visage, nous nous sommes focalisés sur les paramètres influençant l'aspect du reflet dans les lunettes de soleil, sans trop nous préoccuper de l'arrière-plan de la scène.

Nous avons entre autres fait varier les éléments suivants : la teinte des verres de lunettes, la luminosité propre à la scène, la résolution de l'appareil photo et la quantité de reflets parasites dans l'image. La distance d'acquisition des images a également un impact que nous avons laissé fluctuer naturellement en fonction des schémas d'attaque. La taille de l'écran en pixels diffère plus à travers d'autres paramètres tels que la résolution. Nous pouvons nous en convaincre grâce à la formule développée par Xu et al. [14] p. 7.

Nous allons aborder l'influence de ces paramètres sur le reflet de l'écran d'abord qualitativement, ensuite lorsque c'est possible nous l'illustrerons par des résultats concrets.

FIGURE 5.3 – À gauche un exemple d'image prise à contre-jour où le reflet est discernable, à droite un exemple à l'extérieur où la luminosité de la scène est trop importante pour distinguer le moindre reflet.

Modèle des lunettes de soleil

Comme présenté à la FIGURE 5.2, nous avions à notre disposition cinq modèles de lunettes de soleil se distinguant les uns des autres par la teinte, la forme et la courbure de leurs verres. La teinte des verres reste sûrement la caractéristique la plus influente. Nous avons trois variations de verres plus foncés, que nous appellerons de type "classique", où le reflet de l'écran est moins contrasté et où la présence de reflets parasites est faible. Ensuite, nous avons également deux surfaces beaucoup plus réfléchissantes, nommées type "miroir", où le reflet de l'écran est plus net et plus lumineux mais où la présence de réflexions parasites est plus importante.

La courbure du verre, qui agit sur la taille du reflet, n'est pas assez diversifiée parmi les modèles pour que cela soit notable.

Résolutions différentes

Nous avons testé deux résolutions différentes pour les images isolées, 13/16 MP (4614 x 3464 px) et deux autres pour les vidéos, 720p et FHD (1920 x 1080 px). La conséquence directe d'un changement de résolution est le changement de dimensions de l'écran dans l'image. Dans le cas des lunettes de soleil, le changement de taille entre ces trois résolutions n'influence guère la détection du reflet car celui-ci est toujours assez grand pour être repéré. Notons néanmoins qu'en augmentant la résolution, nous augmentons également le bruit présent dans les verres. Par exemple, un petit objet qui ne serait pas détecté comme potentiel candidat (**étape 3** de l'algorithme) à 8 MP pourrait l'être à 16 MP. Précisons aussi que nous avons analysé toutes les photographies avec les mêmes paramètres de l'algorithme (taille des ouvertures, etc).

Luminosité ambiante

Un facteur capital pour assurer l'efficacité de l'algorithme est la quantité de lumière environnante de la scène. De la même manière qu'il est pénible de lire l'écran de son téléphone, il peut être difficile voire impossible de distinguer les reflets dans des lunettes lorsque la luminosité est trop forte. Il est compréhensible que la méthode ne puisse aboutir en l'absence de reflet. Cette information de luminosité, délicate à quantifier, nous invite à privilégier les scénarios d'attaque en intérieur ou dans des endroits plus sombres.

Insistons aussi sur la différence entre une luminosité environnante importante et une photo prise à contre-jour. La photo à contre-jour peut permettre de discerner les écrans dans les verres de lunettes, même si l'arrière-plan de l'image est imperceptible (exemple FIGURE 5.3). Les scènes à contre-jour peuvent cependant compliquer la focalisation de l'appareil photo sur le visage de la victime.

Reflets parasites

Comme évoqué précédemment, il arrive de rencontrer la présence d'objets inopportuns dans les verres de lunettes de soleil. Nous avons souligné l'influence du type de verres dans l'affluence de ces objets. Plus le verre est réfléchissant, plus il est probable de se heurter à ce genre de bruit. Ce soucis est pris en charge par l'algorithme de vérification de forme rectangulaire isRectangle. Toujours dans le but de diversifier notre ensemble de d'images, il est par exemple possible de favoriser la présence de reflets parasites en rajoutant des objets sur la table près de la victime.

5.2 Images isolées

5.2.1 Collecte des données

Pour constituer un ensemble de données (*dataset*) diversifiées et réalistes, nous avons collecté les images en prenant soin de faire varier les paramètres susmentionnés. Nous avons recueilli au total 70 images, divisées en 7 séries de 10 images, chacune correspondant aux 10 chiffres du clavier. Nous pouvons classer ces sous-ensembles en deux catégories distinctes : les images où la victime porte des lunettes de type "miroir", représentant 3 séries sur 7, et les images où la victime porte des lunettes de type "classiques", représentant 4 séries sur 7.

Nous avons aussi voulu complexifier la tâche de l'algorithme en prenant des photos dans des situations de moins en moins favorables. Ainsi, pour les trois premières séries nous avons récolté les photos dans un environnement intérieur, où toute la scène est bien éclairée, en limitant le nombre de reflets parasites et en prenant le temps nécessaire à chaque image pour réaliser un focus optimal sur le visage de la victime et en s'assurant d'avoir des images bien nettes. Pour les quatre séries suivantes, nous avons "négligé" de manière intentionnelle ces paramètres. Nous retrouvons de manière sporadique dans ces sets des images légèrement floues, à contre-jour, un focus mal réalisé, ou encore une présence plus importante de reflets parasites.

5.2.2 Résultats

Notre méthode a été testée sur l'ensemble des images recueillies. Nous allons maintenant analyser les résultats pour essayer de tirer des conclusions et arriver à discerner : les étapes efficaces et problématiques de notre algorithme, les situations favorables et défavorables à l'identification du chiffre, les facteurs les plus influents, etc.

Un chiffre est correctement identifié au premier essai si le centre de sa zone "active" est le plus proche du point de contact par rapport aux centres des autres chiffres. Il est correctement identifié au deuxième essai si son centre est le second plus proche du point de contact.

Identification du chiffre

Tout d'abord comme nous le montre la FIGURE 5.4, sur l'ensemble des 70 images collectées, l'algorithme identifie correctement environ 50 % des chiffres au premier essai et près de 65 % au second. Nous pouvons directement contraster ces résultats en les comparant avec les taux de réussite non plus sur l'ensemble des images, mais uniquement sur celles où l'algorithme a identifié un point de contact [2]. Nous observons une amélioration très nette, nous passons à presque 70 %

2. Rappelons que l'algorithme peut estimer qu'il ne reconnait pas d'écrans dans l'image et à la place de se forcer à renvoyer un point de contact probablement erroné, il peut ne rien retourner du tout.

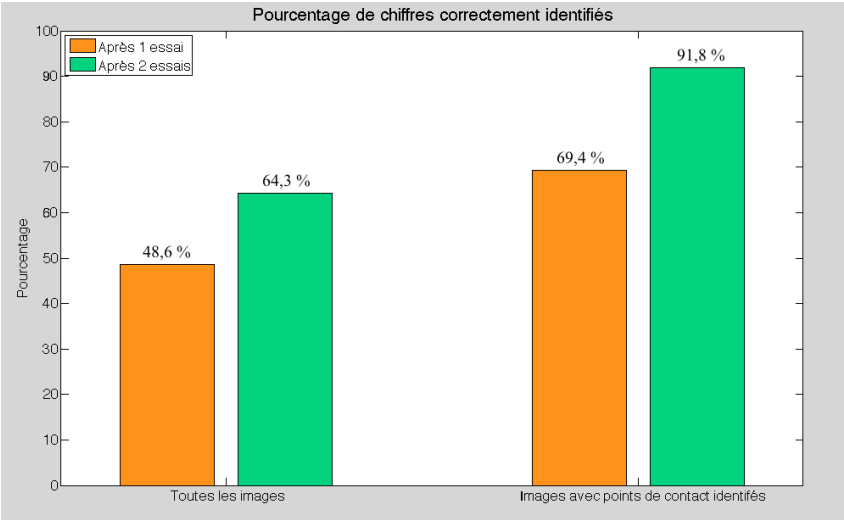


FIGURE 5.4 – Résultats de notre méthode : à gauche sur l'ensemble des images et à droite sur les images pour lesquelles la méthode a fourni un point de contact.

de chiffres identifiés correctement au premier essai et quasiment 92 % au second.

Le premier constat que nous pouvons tirer de ces chiffres est que les points de contact estimés par l'algorithme sont généralement corrects. Les étapes d'estimation du toucher à partir des reflets probables sont donc a priori efficaces. Comme nous le souhaitions lors de son élaboration, notre méthode préfère écarter une image plutôt que de fournir une donnée incorrecte, ce qui est préférable pour les vidéos dont le *frame rate* est assez élevé (entre 20 et 30 frames/seconde).

Détection des reflets de l'écran

Intéressons-nous maintenant aux étapes précédentes qui ont pour objectif de repérer les reflets probables de l'écran. La FIGURE 5.5 illustre trois cas de figure. Le premier sur la gauche donne le pourcentage d'images où au moins un des reflets de l'écran a été identifié (vérification "manuelle") mais où aucun point de contact n'a été fourni par l'algorithme. Le second au centre donne le pourcentage d'images où aucun reflet de l'écran n'a été détecté, tandis que le troisième sur la droite donne le pourcentage d'images où l'algorithme n'a estimé aucun point de contact.

Au préalable, énonçons quelques précisions sur cette classification. Pour la première catégorie, les reflets de l'écran auxquels nous faisons référence sont les reflets réels de l'écran, à ne pas confondre avec les candidats potentiels évoqués à l'**étape 3** de l'algorithme. Nous avons donc examiné "manuellement" qu'au moins un des deux reflets se trouvait dans l'ensemble des candidats potentiels détectés avant de passer par la fonction isRectangle. La seconde catégorie reprend aussi les images où un point de contact erroné a été estimé, ce qui explique pourquoi la dernière catégorie n'est pas "simplement" l'addition des deux premières. Précisons aussi que nous avons différencié les images provenant des catégories où la surfaces des verres est de type "miroir" (SM) ou "non-miroir" (SNM).

Nous observons qu'en moyenne moins de 10 % des images où un des écrans est détecté ne mènent pas à un résultat (valide ou non). Par contre, dans près de 25 % des images aucun écran n'est détecté et au total 30 % du set complet ne fournissent pas d'estimation. Nous constatons qu'une des raisons principales pour lesquelles l'algorithme n'a pas fourni de résultats est qu'il n'a pas détecté correctement le reflet de l'écran. Un autre constat est que dans une image sur trois de type "miroir", l'écran n'est pas repéré et la proportion d'images ne conduisant pas à un résultat est deux fois plus importante par rapport au cas "classique".

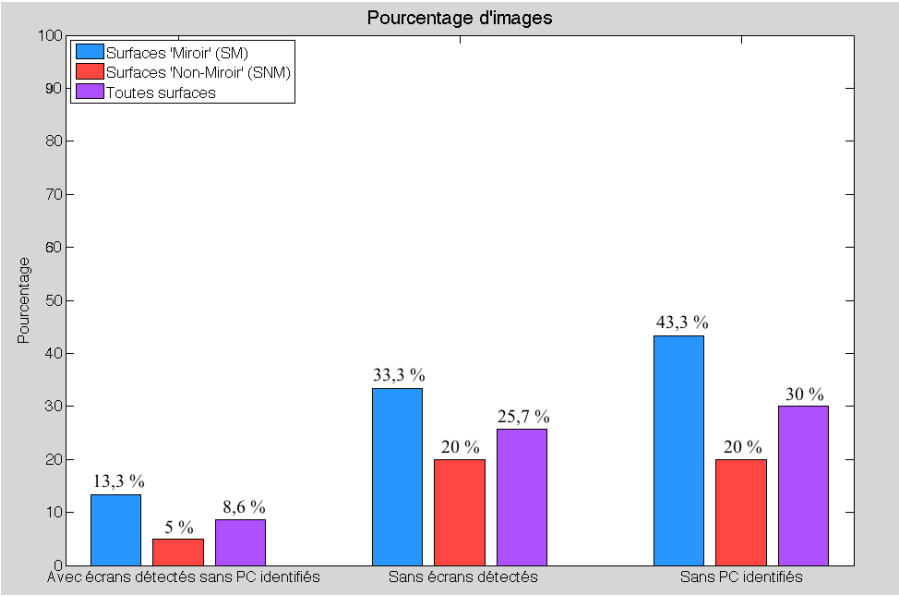


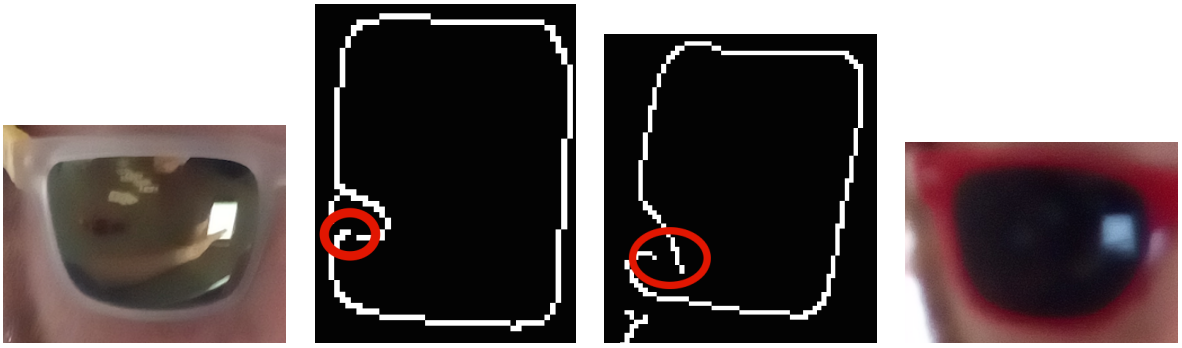
FIGURE 5.5 – Détails du pourcentage d'images où l'algorithme : a détecté un écran sans identifier de point de contact ; n'a pas détecté d'écran ; n'a pas identifié de point de contact.

Pour comprendre cette difficulté de détection des écrans, qui est accrue pour les surfaces plus réfléchissantes, nous avons analysé les images problématiques et essayé d'identifier les obstacles que rencontre l'algorithme. Dans un premier temps nous aurions pu penser qu'il serait plus aisé de repérer un reflet plus net dans les teintes de type "miroir", cependant cette qualité élevée rend plus difficile la tâche du détecteur de bords. Contrairement aux surfaces classiques, il y a beaucoup plus de contours qui ne sont pas entièrement fermés (voir FIGURE 5.6). Ceci peut s'expliquer par l'appauvrissement des couleurs et l'ombre du pouce plus marquée pour les surfaces plus foncées qui rendent le contraste entre l'écran et le reste du verre plus prononcé. Une raison valable quant à elle pour tous les types de surface est la netteté de l'image. Bien que difficilement quantifiable, l'algorithme repère beaucoup moins les écrans dans des images plus floues, en raison du mouvement de l'attaquant lors de la capture de la photo ou d'un mauvais focus.

Comparaison des types "classique" et "miroir"

Nous nous intéressons désormais à la comparaison entre les deux catégories. Les résultats sont exposés à la FIGURE 5.7. Pour les tests réalisés sur l'ensemble des données, le taux de réussite pour le type "miroir" est très faible comparé au type "classique" (37 et 50 % contre 57 et 75 %). Ce résultat aurait pu se prédire au vu des conclusions obtenues au point précédent. L'algorithme détecte moins les écrans pour le type "miroir", il est donc normal que le nombre d'images où le chiffre a été correctement identifié soit beaucoup plus faible. On retrouve le même écart d'environ 20 % que pour les images où il n'y a pas de point de contact identifié.

Par contre, les résultats sur les images fournissant une estimation du chiffre sont plus instructifs. On constate que les taux de réussite sont bien plus similaires (entre 65 et 72 % pour le premier essai, entre 88 et 94 % pour le second). Cela laisse à penser qu'une fois le reflet de l'écran correctement détecté, la suite de l'algorithme est relativement efficace indépendamment du genre de teinte.



(a) Image menant à des contours ouverts (b) Contours ouverts (c) Image trop floue

FIGURE 5.6 – Exemples d'obstacles rencontrés par la détection d'écrans. En (a) une image occasionnant des contours ouverts tels que ceux illustrés en (b). En (c) un exemple d'image trop floue.

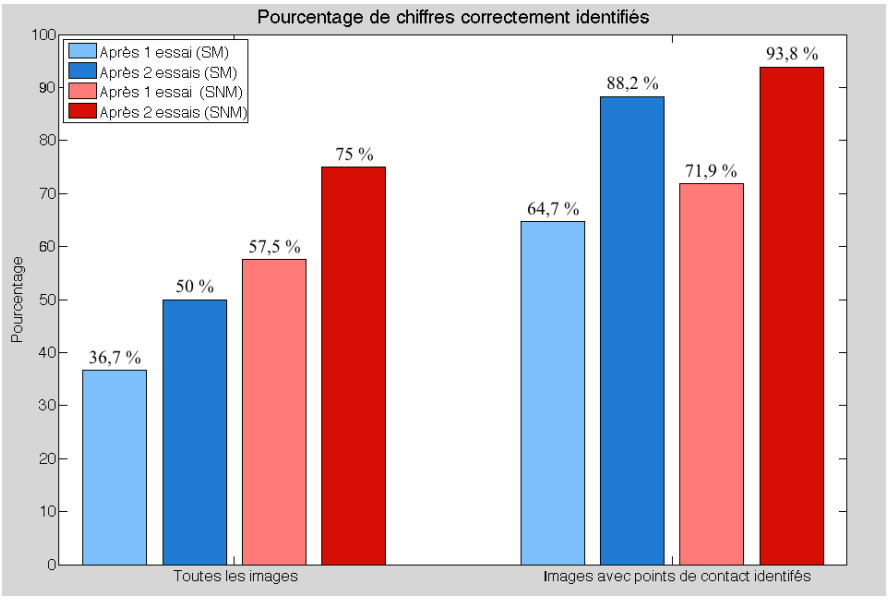
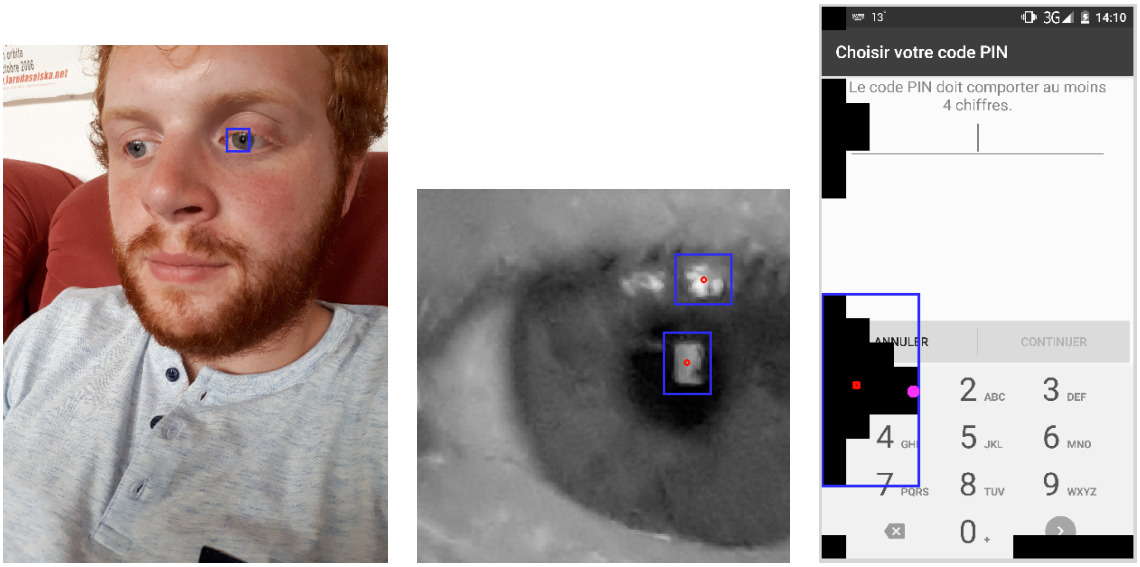


FIGURE 5.7 – Exemples de différentes teintes de verres de soleil utilisés



(a) Image de base (b) Candidats potentiels (c) Forme du doigt superposée au clavier

FIGURE 5.8 – Exemple de réflexion dans l'œil. En bleu les zones sélectionnées par l'utilisateur (a) et par l'algorithme (b). Le chiffre correct est 7 dans ce cas.

5.2.3 Discussion

Nous avons constaté que notre méthode procure de bons résultats lorsqu'elle détecte correctement un des reflets de l'écran. Pour améliorer les résultats globaux de la méthode, il convient d'essayer d'améliorer la détection des écrans. Les exemples d'images où l'écran n'est pas détecté nous amènent à deux constats : le contour est souvent presque entièrement fermé, mais tout de même ouvert, et les images floues ne conduisent que très rarement à une détection des écrans. Pour palier à ces soucis, nous pouvons envisager plusieurs solutions.

Pour obtenir des contours fermés nous avons pensé à effectuer une fermeture sur l'image binaire. Cependant dans la plupart des cas, cela rajoute trop de bruit dans l'image pour aider à la bonne détection de l'écran. Une piste à envisager serait de combiner cette fermeture à un algorithme de tracking qui réduirait encore plus la zone où chercher. Après avoir réduit manuellement la zone utile et forcé l'algorithme à effectuer une fermeture dans quelques cas, nous avons remarqué que la forme ainsi obtenue s'adapte pleinement à la suite de notre algorithme.

Ensuite pour améliorer le problème de netteté, nous avons pensé à l'utilisation d'un filtre anisotropique comme le font Xu et al.[14] p.7 mais nous n'avons pas eu l'occasion de tester cette solution.

5.2.4 Reflet dans l'œil

Afin de franchir une étape supplémentaire dans le réalisme des schémas d'attaque, nous avons testé notre algorithme dans le cas de figure où la victime ne porte plus de lunettes de soleil. Nous avons donc tenté d'identifier le chiffre directement à l'aide du reflet de l'écran dans sa pupille. Cette situation a demandé quelques adaptations.

1. Pour l'acquisition des photos, l'attaquant utilise un Samusung Galaxy A5 2017. La caméra arrière a une résolution de 16 MP (4614 x 3464 px).
2. Nous avons insisté pour que la victime ouvre suffisamment ses yeux, sans doute plus qu'en temps normal.

n°	Nombre de frames	Nombre de PC	Code saisi	Code estimé	Code décalé	Nombre corrects	Nombre proches
1	129	55	4 - 6 - 8 - 2	6 - 5 - 2 - X	Oui	2	$5 \rightarrow 8$
2	151	55	1 - 3 - 7 - 9	1 - 3 - 5 - 3	Non	2	/
3	115	33	4 - 5 - 6 - 5	5 - 6 - 5 - X	Oui	3	/
4	116	42	7 - 1 - 0 - 2	5 - 5 - 5 - 1	Non	0	$1 \rightarrow 2$

TABLE 5.1 – Tableau comparatif des résultats pour les vidéos testées.

3. L'attaquant est plus proche de la victime, à environ 50 cm.
4. La zone utile sélectionnée manuellement est beaucoup plus restreinte que dans le cas des lunettes de soleil. Nous avons délimité une zone autour d'un des yeux.
5. Nous avons rajouté une fermeture par un disque de rayon égal à 2 pixels.

Le but de cette démarche est de voir s'il est possible de récupérer de l'information dans le reflet des yeux. Nous avons testé la méthode sur un nombre réduit d'images (une dizaine) et dans deux cas seulement nous arrivons à récupérer un point de contact provenant d'un écran. Nous avons présenté un de ces cas favorable à la FIGURE 5.8. La victime tapait le chiffre 7 et notre méthode a retrouvé le chiffre 3, ce qui représente une translation verticale de quelques pixels seulement à l'échelle de l'image originale. Le reflet de l'écran mesure environ 10 x 20 px dans l'œil.

Nous avons constaté que les paramètres importants pour les lunettes de soleil (et non-liés au modèle des lunettes évidemment) sont encore plus accentués ici. Par exemple si l'image est légèrement floue il devient quasi impossible de détecter l'écran. Ou encore comme le montre la FIGURE 5.8, une imprécision de quelques pixels entraine un bond de deux chiffres vers le haut. L'étape d'acquisition des images est encore plus primordiale.

5.3 Vidéos

Selon le protocole définit précédemment, nous avons testé notre algorithme sur une dizaine de vidéos. Contrairement aux images isolées, il peut être plus difficile d'influencer certains paramètres lors de l'acquisition des vidéos. Par exemple, il arrive qu'un focus correctement établi au début de la vidéo se dégrade au long de celle-ci.

5.3.1 Code PIN

Nous avons rassemblé (voir TABLE 5.1) les résultats pour les vidéos où la proportion entre le nombre total de frames et le nombre de points de contact estimés est supérieur à un quart. Ces résultats sont assez mitigés. On remarque qu'aucun code n'est identifié dans son entièreté et souvent, l'algorithme peine à détecter un des chiffres. À l'instar de ce qui avait été constaté pour les images, le problème majeur réside dans l'identification des écrans. Dans les "meilleurs" cas, seulement une frame sur trois conduit à une estimation d'un point de contact. C'est un problème car les images écartées ne sont pas indépendantes entre elles. On constate que souvent ce sont les frames en rapport avec un même chiffre qui ne sont pas validées. Cela semble logique car la forme de l'écran est similaire dans ces images. Cependant, l'algorithme est capable d'apporter à l'attaquant des informations pertinentes sur le code tapé par la victime. Cela mène à penser qu'en perfectionnant la détection des écrans, les résultats devraient s'améliorer considérablement.

5.3.2 Schémas de déverrouillage

Au vu des résultats en demi-teinte obtenus pour le cas des codes PIN, nous n'avons pas testé l'identification de schémas sur des vraies vidéos. Il est plus que probable que nous aurions rencontré les mêmes soucis d'identification indépendamment du style de données saisies par l'utilisateur. Cependant, nous avons tenu à proposer une méthode de validation de notre algorithme à partir de données recueillies "artificiellement". Nous avons combiné les points de contact estimés dans des images isolées pour former un code chemin similaire au schéma qu'aurait rentré la victime. Afin de complexifier la tâche de l'algorithme, pour chaque chiffre nous avons utilisé des images provenant de différents sets, conduisant à un certain bruit dans les données.

Dans la quasi-totalité des cas, notre méthode identifie le schéma correctement. Toutefois, ces résultats restent de l'ordre du théorique car nous avions utilisé une manière de procéder semblable lors de la conception de cette méthode.

5.4 Conclusion

La méthode développée dans ce travail montre des résultats satisfaisants pour les images isolées mais mitigés lorsqu'elle est appliquée à des vidéos. Les observations indiquent que le point le plus important à améliorer est certainement la détection du reflet de l'écran dans l'image. Comme déjà discuté, une des solutions envisageables serait de rajouter un *tracking* du reflet couplé à un renforcement dans la détection dans la zone traquée. Un autre point positif à souligner est le faible taux de résultats complètement erronés.

Les résultats des différentes sections de ce chapitre permettent de croire que la reconnaissance de codes PIN ou de schémas complets n'est pas encore atteinte avec notre algorithme, mais est vraisemblable.

Chapitre 6

Conclusions

Les smartphones affichent une multitude d'applications et de possibilités, tout en cherchant à offrir à l'utilisateur un maximum de confort. Toutefois confort d'usage ne rime pas avec sécurité. Par ce travail, nous avons tenté d'apporter une méthode originale permettant d'exploiter cette faille de sécurité. L'algorithme que nous avons développé est redoutable par la diversité des schémas d'attaque qu'il propose, par sa simplicité algorithmique et parce qu'il constitue une menace presque indétectable.

Dans les premières pages de ce travail, nous avons détaillé comment le problème a été abordé précédemment à travers une série de méthodes récentes que nous avons ensuite pu comparer à notre méthode et mettre en évidence les outils nouveaux que nous apportions. Ensuite, après avoir parcouru les notions théoriques exploitées, nous avons présenté les algorithmes développés pour assurer le bon fonctionnement de notre méthode. Par ces algorithmes, nous avons montré qu'il est possible d'aborder un problème complexe en apparence et de lui apporter une solution relativement simple. Finalement, nous pouvont dire que nos résultats affichent de bons taux de réussite dans le cas d'images isolées, contrairement aux vidéos où les performances sont beaucoup plus mitigées. Nous avons discuté des perfectionnements à envisager, comme l'ajout d'un algorithme de tracking, pour améliorer l'identification des reflets dans les vidéos. Enfin, nous avons réussi à exploiter des reflets d'écran directement depuis l'œil de l'adversaire. Symboliquement d'une part et en considérant l'augmentation grandissante de la qualité des appareils photo intégrés aux smartphones d'autre part, cela ouvre des portes à de nombreuses applications futures, et ne fait qu'accentuer davantage la menace de ce genre de scénarios.

Finalement, nous ne pouvions conclure ce travail sans citer brièvement les contremesures déjà existantes. Que ce soit par des claviers à géométrie aléatoire, des reconnaissances faciales ou encore par de la lecture d'empreintes digitales, les moyens de sécuriser plus efficacement nos données privées sur notre smartphone existent, mais encore faut-il que l'utilisateur s'en serve.

Bibliographie

[1] M. Backes, T. Chen, M. Duermuth, H. P. Lensch, and M. Welk. Tempest in a Teapot : Compromising Reflections Revisited. pages 315–327. IEEE, May 2009.

[2] M. Backes, M. D, and D. Unruh. Compromising Reflections-or-How to Read LCD Monitors around the Corner. pages 158–169. IEEE, May 2008.

[3] J. Canny. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6) :679–698, Nov. 1986.

[4] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android permissions : User attention, comprehension, and behavior. In *Proceedings of the eighth symposium on usable privacy and security*, page 3. ACM, 2012.

[5] T. Fiebig, J. Krissler, and R. Hänsch. Security Impact of High Resolution Smartphone Cameras. In *WOOT*, 2014.

[6] M. A. Fischler and R. C. Bolles. Random sample consensus : a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6) :381–395, 1981.

[7] H. Grabner, M. Grabner, and H. Bischof. Real-time tracking via on-line boosting. In *Bmvc*, volume 1, page 6, 2006.

[8] M. G. Kuhn. *Compromising emanations : eavesdropping risks of computer displays*. PhD thesis, University of Cambridge, 2002.

[9] A. Lavie and M. J. Denkowski. The Meteor metric for automatic evaluation of machine translation. *Machine Translation*, 23(2-3) :105–115, Sept. 2009.

[10] M. A. R. Pathan and M. P. Talwai. Tracking, learning and detection of an obkect in video.

[11] R. Raguram, A. M. White, D. Goswami, F. Monrose, and J.-M. Frahm. iSpy : automatic reconstruction of typed input from compromising reflections. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 527–536. ACM, 2011.

[12] D. Shukla, R. Kumar, A. Serwadda, and V. V. Phoha. Beware, Your Hands Reveal Your Secrets ! pages 904–917. ACM Press, 2014.

[13] A. Vedaldi and B. Fulkerson. VLFeat : An open and portable library of computer vision algorithms. In *Proceedings of the 18th ACM international conference on Multimedia*, pages 1469–1472. ACM, 2010.

[14] Y. Xu, J. Heinly, A. M. White, F. Monrose, and J.-M. Frahm. Seeing double : reconstructing obscured typed input from repeated compromising reflections. pages 1063–1074. ACM Press, 2013.

[15] K. Zuiderveld. Contrast Limited Adaptive Histograph Equalization. In *Graphics gems IV*, The Graphics gems series, pages 474–485. AP Professional, Boston, 1994.

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**