Ceci est un extrait électronique d'une publication de Diamond Editions :

http://www.ed-diamond.com

Retrouvez sur le site tous les anciens numéros en vente par correspondance ainsi que les tarifs d'abonnement.

Pour vous tenir au courant de l'actualité du magazine, visitez :

http://www.gnulinuxmag.com

Ainsi que :

http://www.linux-pratique.com

et

http://www.miscmag.com

Ceci est un extrait électronique d'une publication de Diamond Editions

http://www.ed-diamond.com

Creative Commons

Paternité - Pas d'Utilisation Commerciale - Pas de Modification

2.0 France

Vous êtes libres :

- de reproduire, distribuer et communiquer cette création au public.

Paternité. Vous devez citer le nom de l'auteur original de la manière indiquée par l'auteur de l'oeuvre ou le titulaire des droits qui vous confère cette autorisation (mais pas d'une manière qui suggérerait qu'ils vous soutiennent ou approuvent votre utilisation de l'oeuvre).

Pas d'Utilisation Commerciale. Vous n'avez pas le droit d'utiliser cette création à des fins commerciales.

Pas de Modification. Vous n'avez pas le droit de modifier, de transformer ou d'adapter cette création.

A chaque réutilisation ou distribution de cette création, vous devez faire apparaître clairement au public les conditions contractuelles de sa mise à disposition.

- Chacune de ces conditions peut être levée si vous obtenez l'autorisation du titulaire des droits.
- Rien dans ce contrat ne diminue ou ne restreint le droit moral de l'auteur ou des auteurs.

Ceci est le Résumé Explicatif du Code Juridique. La version intégrale du contrat est attachée en fin de document et disponible sur :

http://creativecommons.org/licenses/by-nc-nd/2.0/fr/legalcode

Par :: Fabrice Rossi :: Projet AxIS (INRIA) ::
:: Fabrice.Rossi@apiacoa.org :: http://apiacoa.org ::

La reconnaissance de gestes

Une interaction efficace avec un ordinateur nécessite une interface à l'ergonomie soignée. Bien que le clavier assisté par la souris soit la méthode la plus efficace à ce jour pour interagir avec un ordinateur, cette solution n'est pas envisageable dans certaines situations. C'est le cas par exemple pour les assistants personnels, les téléphones portables ou plus généralement pour tous les « wearable computers », ces ordinateurs qui doivent s'intégrer dans nos vêtements. De plus, certains utilisateurs ne peuvent pas se servir d'un clavier et d'une souris, même avec un ordinateur classique. Il est donc indispensable de fournir d'autres moyens d'interaction avec l'ordinateur, par le biais de stylet, d'interfaces non conventionnelles basées sur le suivi du regard, ou encore grâce à la parole. Dans cet article, nous allons étudier les techniques qui permettent à l'ordinateur de reconnaître les gestes au sens large, en nous focalisant sur l'interaction basée sur un stylet. Les algorithmes présentés ne seront pas cependant limités à cette application mais concerneront plus généralement la reconnaissance de forme.

Introduction

Les assistants personnels (PDA pour *Personal Digital Assistant* en anglais) sont devenus en quelques années des outils quasi indispensables, tant pour le *geek* qui ne saurait survivre loin d'un ordinateur que pour le cadre dynamique à l'agenda surchargé. Les premières versions des PDA, plutôt appelées « organiseurs électroniques » à l'époque, étaient encombrées par des claviers très peu ergonomiques en raison de leurs touches minuscules.

Grâce à des travaux de recherche commencés dans les années 1970, les claviers ont disparu au profit de stylets utilisés sur des écrans tactiles : il est fort probable que le succès des PDA soit en grande partie dû à ce nouveau mode d'interaction bien plus adapté à la petite taille de ces outils. Le stylet est tellement intuitif et facile d'accès qu'on en a presque oublié ses fondements algorithmiques. Il s'agit pourtant d'une avancée majeure de l'intelligence artificielle (IA), à savoir la mise au point d'algorithmes efficaces de reconnaissance automatique d'une forme simplifiée d'écriture manuscrite.

Ces algorithmes sont d'autant plus intéressants qu'ils sont basés sur une idée très simple : s'intéresser aux gestes plutôt qu'aux images. En d'autres termes, ce n'est pas le dessin final d'une lettre qu'on va reconnaître, mais la façon dont celle-ci a été tracée.

De prime abord, cette approche peut sembler étrange, tant nous sommes habitués à procéder différemment. Pourtant, elle illustre parfaitement un principe fondamental de l'IA : le but n'est pas de *copier* l'intelligence humaine, mais bien de *donner l'illusion* que l'ordinateur se comporte de façon intelligente.

En d'autres termes, il s'agit de reproduire les effets, pas nécessairement les causes. Nous avons vu dans [1] comment l'ordinateur peut utiliser sa mémoire incomparablement plus précise que la nôtre pour repérer des répétitions dans notre comportement et nous battre à des jeux simples en donnant l'impression d'avoir deviné nos intentions : pourtant, les algorithmes utilisés ne font aucune hypothèse psychologique, ne sont pas capables de décoder nos expressions corporelles pour savoir si nous bluffons, etc. L'ordinateur ne procède donc pas comme nous le ferions.

Or, comme nous allons le voir dans la suite de l'article, l'information contenue dans le geste qui produit le dessin d'une lettre, est beaucoup plus riche que celle qu'on peut extraire de l'image finale. En un certain sens, elle est même trop riche pour un humain : pour s'en convaincre, il suffit d'observer une autre personne écrire, puis de comparer avec sa propre méthode.

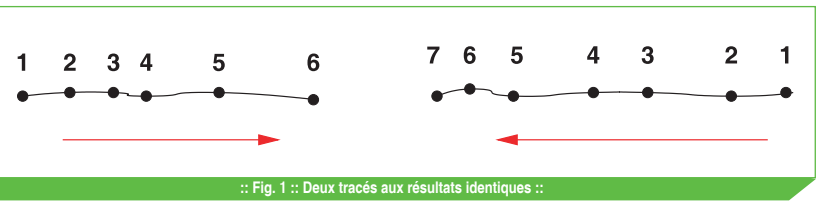
En général, on découvre de nombreuses variantes qui ne semblent pas avoir beaucoup d'influence sur le dessin final (en faisant abstraction du cas des *geeks* qui ne savent plus écrire avec cet objet préhistorique qu'est un stylo). Il serait très fastidieux d'avoir à apprendre ces nombreuses variantes, alors que reconnaître le dessin final suffit.

Pour l'ordinateur cependant, apprendre par cœur ne pose pas de problème quand les informations ne sont pas ambiguës. Or, introduire l'information additionnelle contenue dans le geste qui produit le dessin permet justement de réduire l'ambiguïté présente dans les dessins et facilite donc la tâche de l'ordinateur.

Des images aux gestes

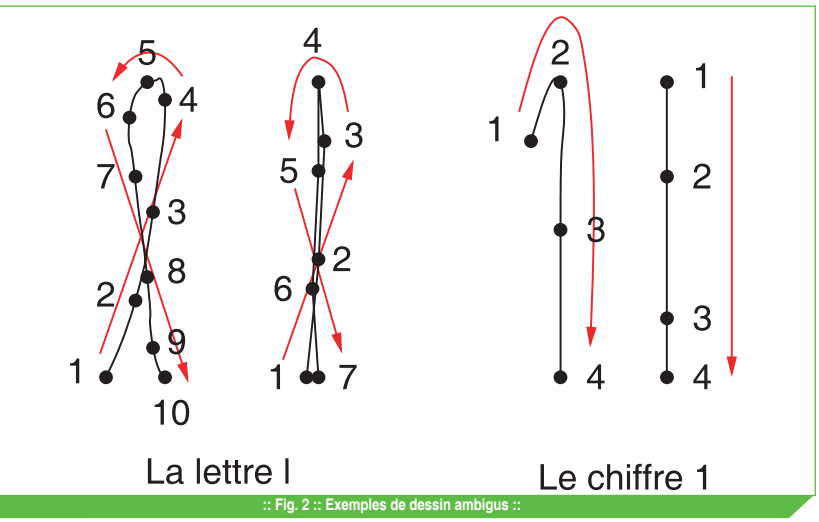
Le tracé est crucial

Pour bien comprendre la richesse apportée par l'étude du tracé plutôt que du résultat de celui-ci, il suffit de considérer la figure 1. Les deux tracés conduisent à des dessins identiques (un trait horizontal) mais sont associés à des actions différentes dans l'alphabet Graffiti® des Palms™ [2] : le dessin de gauche correspond à un espace alors que celui de droite provoque l'effacement du dernier caractère saisi.



:: Fig. 1 :: Deux tracés aux résultats identiques ::

Pour différencier les dessins, il faut impérativement tenir compte des tracés : l'espace est obtenu en traçant une ligne de gauche à droite, alors que l'effacement s'obtient au contraire en allant de droite à gauche.

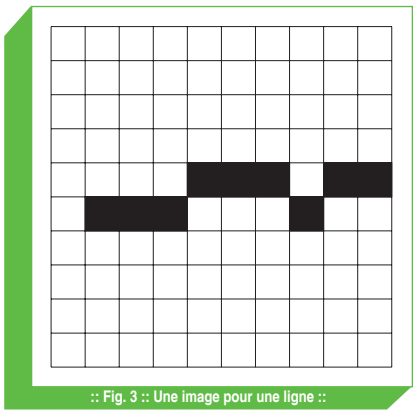


:: Fig. 2 :: Exemples de dessin ambigus ::

Cet exemple peut sembler pathologique, mais il est en fait assez révélateur. Pour s'en convaincre, il suffit de penser au cas de la lettre « l » et du chiffre « 1 ». Il n'est pas rare que les deux caractères soient très voisins, voire confondus dans l'écriture de la plupart des gens : on se retrouve en fait assez fréquemment avec une barre verticale qui n'est pas sans rappeler l'exemple de la barre horizontale. Le tracé est alors indispensable, comme le montre la figure 2. On constate en effet que sur une écriture lisible (le « l » de gauche et le « 1 » de gauche), les deux caractères sont discernables, alors que sur une écriture moins soignée (le « l » et le « 1 » de droite) les dessins des caractères sont presque identiques. Dans ce dernier cas, le tracé des caractères permet cependant de les différencier : le « l » est tracé de bas en haut puis de haut en bas, alors que le « 1 » est obtenu plus directement de haut en bas.

Représenter le tracé

Du point de vue conceptuel, il est donc clair que le tracé d'un caractère contient plus d'informations que l'image résultat. En pratique cependant, ceci a un coût, essentiellement lié au stockage de l'information. En effet, la nature même d'un tracé n'est pas évidente du point de vue informatique, contrairement à une image. Considérons de nouveau l'exemple de la figure 1, à savoir une ligne horizontale. La figure 3 donne une image d'une telle ligne, avec un grossissement suffisant pour qu'on puisse voir les pixels indépendamment les uns des autres.



:: Fig. 3 :: Une image pour une ligne ::

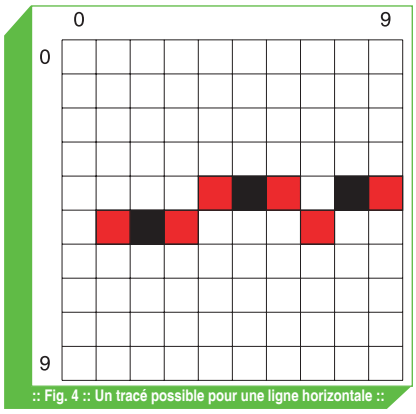
J'ai choisi ici d'utiliser une image de 10x10 pixels. En pratique, on pourrait bien sûr choisir une résolution plus précise, mais l'idée de base est de fixer une résolution et de toujours considérer des images d'une taille constante.

Ceci permet de comparer facilement des images entre elles, simplement en travaillant pixels par pixels. De plus, le capteur utilisé (l'écran tactile du PDA par exemple) impose une résolution, ce qui nous décharge dans un premier temps de la tâche de choisir celle-ci.

Le cas d'un tracé est beaucoup plus délicat. En effet, l'aspect temporel est moins contraint par le dispositif de saisie que l'aspect visuel. Sur un PDA, on ne peut pas sortir de l'écran (voire de la zone de saisie) quand on trace un caractère. De ce fait, on peut fixer une taille standard pour l'image du caractère. Par contre, l'utilisateur peut tracer très rapidement ou au contraire très lentement, ce qui engendre des variations non négligeables dans la durée d'un tracé.

Or, le capteur possède une résolution temporelle fixée : il mémorise la position du stylet toutes les *u* ms, où *u* est déterminé par la qualité du capteur. De ce fait, le nombre de positions mémorisées dépend du tracé et varie donc avec l'utilisateur, le caractère, les conditions de réalisation du tracé, etc. Ceci est illustré par les figures 1 et 2 : la ligne horizontale contient 6 ou 7 positions, selon le tracé, alors que la lettre « l » correspond à 7 ou 10 positions.

En fait, la description la plus complète d'un tracé est une liste de triplets. Chaque triplet *(x,y,t)* décrit une position (par ses coordonnées *x* et *y*) et l'instant *t* auquel elle correspond. En pratique, on peut en général se passer de *t* car les positions sont enregistrées régulièrement, toutes les *u* ms : si la liste contient 5 triplets, alors le premier correspond à l'instant *t*, le second à *t+u*, etc. jusqu'à *t+4u*, si *t* désigne l'instant de départ de l'enregistrement. La figure 4 donne un exemple de tracé possible pour le dessin de la figure 3 : chaque carré rouge correspond à une position mémorisée par le capteur. On obtient la liste de positions suivante : (1,5), (3,5), (4,4), (6,4), (7,4), (9,4).



:: Fig. 4 :: Un tracé possible pour une ligne horizontale ::

Avec un tracé plus lent, on pourrait obtenir une position enregistrée par pixel de l'image, soit la liste suivante : (1,5), (2,5), (3,5), (4,4), (5,4), (6,4), (7,4), (8,4), (9,4), c'est-à-dire une liste comportant 9 positions contre 6 positions dans l'exemple précédent (les positions soulignées sont présentes seulement dans le tracé lent). Les deux tracés produisent exactement la même image, ce qui achève de prouver que le geste contient plus d'informations que le dessin résultat, mais ce qui illustre aussi la difficulté liée à cette richesse : il va falloir comparer des choses assez différentes, plus précisément des objets de taille variable, ce qui n'est pas facile avec les méthodes classiques de reconnaissance de formes. Diverses techniques ont été mises au point afin de résoudre ce problème. On peut les répartir en deux grandes classes : les techniques qui effectuent des pré-traitements afin de transformer les tracés en des données plus faciles à traiter, et les techniques qui ont été spécifiquement développées pour traiter les données de taille variable.

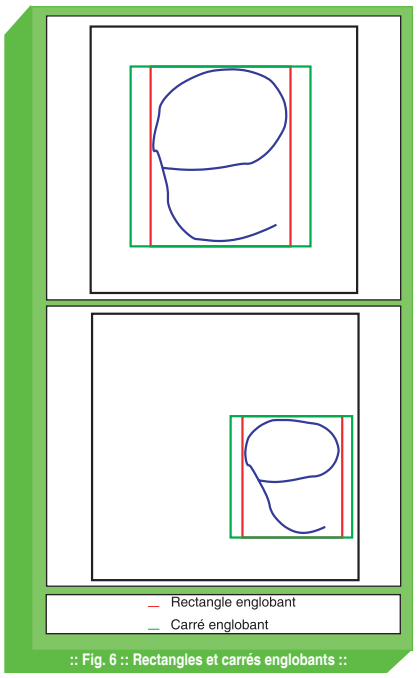
Notons avant de nous intéresser aux pré-traitements que nous nous limitons ici aux caractères tracés en une seule fois, c'est-à-dire sous forme d'une seule ligne. Il est bien sûr possible de travailler sur des tracés plus complexes, mais cela demande des algorithmes plus évolués que ceux que nous aborderons ici.

Transformer les gestes

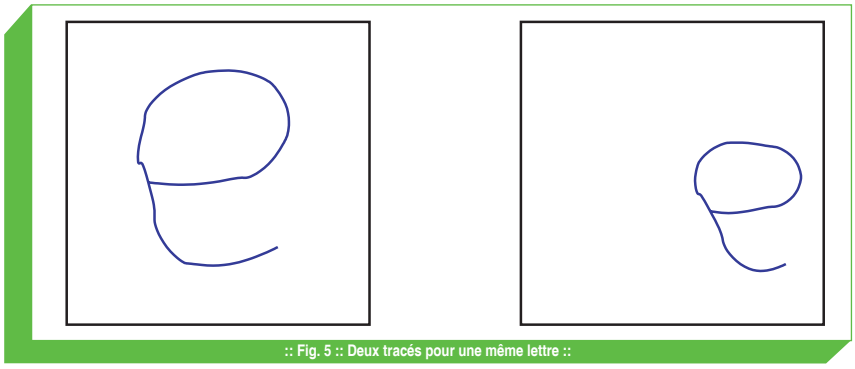
Nous allons maintenant nous intéresser aux transformations utiles pour faciliter la reconnaissance des tracés, en commençant par régler les problèmes d'échelle par une technique commune à la plupart des algorithmes de reconnaissance de gestes, puis nous présenterons les méthodes d'interpolation qui constituent l'une des solutions possibles pour s'affranchir de la taille variable des tracés.

Centrage et mise à l'échelle

La figure 5 illustre ce problème en donnant deux tracés pour la lettre « e ». Le tracé de gauche est centré et occupe une grande partie de la zone de saisie, alors que celui de droite est excentré et plutôt petit. La comparaison directe des images ou des tracés est ainsi rendue difficile par les nombreuses variantes possibles.



:: Fig. 6 :: Rectangles et carrés englobants ::


:: Fig. 5 :: Deux tracés pour une même lettre ::

Le problème le plus simple à régler est en fait commun à une approche basée sur les images et à celle qui nous intéresse, fondée sur les gestes : il s'agit de s'affranchir de la taille et de la position du dessin dans le capteur. En effet, selon les circonstances, un utilisateur peut être amené à tracer un caractère plus ou moins gros et plus ou moins centré dans la zone de saisie, ce qui rend une comparaison directe des dessins délicate.

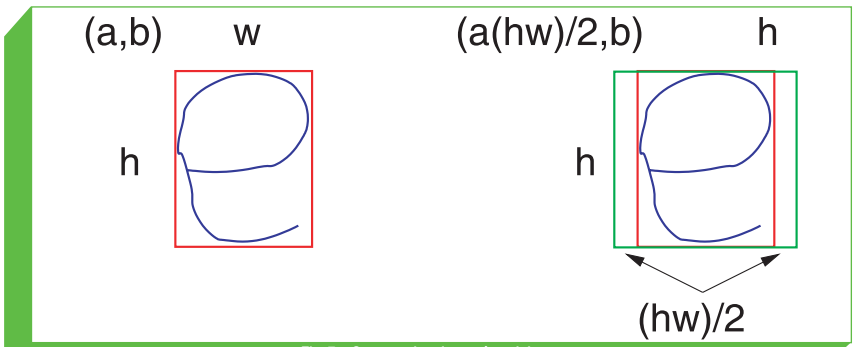
Pour résoudre le problème, on se contente de centrer et de mettre à l'échelle le tracé (ou l'image). Pour ce faire, on commence par déterminer un rectangle englobant pour le tracé, c'est-à-dire le plus petit rectangle qui contient tout le tracé : celui-ci s'obtient par une simple boucle sur les positions du tracé qui recherche le maximum et le minimum pour les deux coordonnées (l'abscisse horizontalement et l'ordonnée verticalement).

Les rectangles sont tracés en rouge sur la figure 6. On détermine ensuite le plus petit carré englobant en le centrant par rapport au rectangle (en vert sur la figure 6). Comme son nom l'indique, on recherche ainsi le plus petit carré qui contient tout le tracé. Celui-ci est nécessairement plus grand (au sens large) que le rectangle englobant. Le problème est que si le côté du carré est unique (il s'agit du plus grand côté du rectangle), on peut ensuite le positionner avec une certaine liberté car, en général, on a de la marge sur une des deux directions (horizontalement ou verticalement). La bonne approche consiste alors à centrer le carré par rapport au rectangle.

Une fois le carré englobant déterminé, on recalcule toute les positions par rapport à celui-ci plutôt qu'à la zone de saisie. Ceci revient en fait à centrer le tracé dans le carré puis à le mettre à l'échelle, comme l'illustre la figure 8 sur l'exemple des deux « e ».

On constate que les tracés sont beaucoup plus proches qu'ils ne l'étaient avant la transformation.

En pratique, cette transformation s'implémente très facilement. Considérons une position du tracé de coordonnées *(x,y)*. Par rapport au coin supérieur gauche du carré englobant de coordonnées *(u-(h-w)/2,v)* (dans le cas où la hauteur du rectangle est plus grande que sa largeur),



:: Fig. 7 :: Construction du carré englobant ::

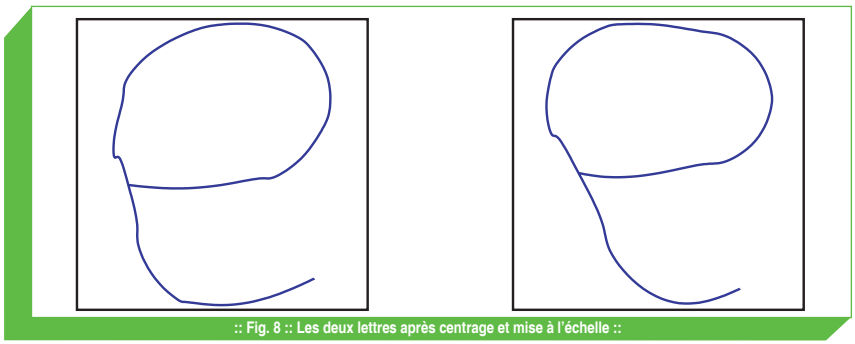
Pour ce faire, on détermine donc un rectangle dont le coin supérieur gauche est de coordonnées *(u,v)*, de largeur *w* et de hauteur *h*. Supposons par exemple que *h* soit plus grand que *w*, et donc que le carré englobant soit de côté *h*. Pour centrer le carré par rapport au rectangle, il faut que le premier dépasse de chaque côté du second de *(h-w)/2*. Les coordonnées du coin supérieur gauche du carré sont donc *(u-(h-w)/2,v)*. La figure 7 illustre la construction du carré englobant sur l'exemple du « e » proposé dans les figures 5 et 6.

les coordonnées de cette position sont :

$$\left(x-u+\frac{h-w}{2},\ y-v\right)$$

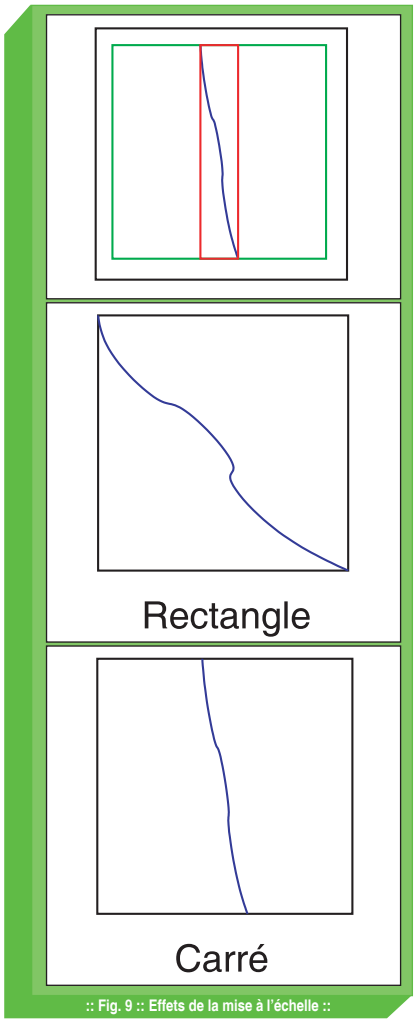
Ceci correspond au centrage. Nous décidons ensuite arbitrairement que le côté du carré englobant est notre unité de mesure. Après mise à l'échelle les coordonnées de la position deviennent :

$$\left(\frac{x-u+\frac{h-w}{2}}{h},\ \frac{y-v}{h}\right)$$



:: Fig. 8 :: Les deux lettres après centrage et mise à l'échelle ::

Une question naturelle doit titiller le lecteur attentif : pourquoi ne pas se contenter du rectangle englobant ? La réponse est illustrée par la figure 9 qui compare les mises à l'échelle induites respectivement par le rectangle englobant et par le carré. On constate que le rectangle déforme énormément le dessin d'origine, alors qu'il est simplement centré et redimensionné avec le carré.



:: Fig. 9 :: Effets de la mise à l'échelle ::

Le but de la transformation est bien de rendre différents tracés faciles à comparer, pas de déformer certains tracés, ce qui justifie donc l'utilisation du carré englobant.

Une variante possible consiste à grossir légèrement le carré dans toutes les directions afin d'éviter que le tracé ne touche les bords de la zone considérée.

Interpolation et mise à l'échelle temporelle

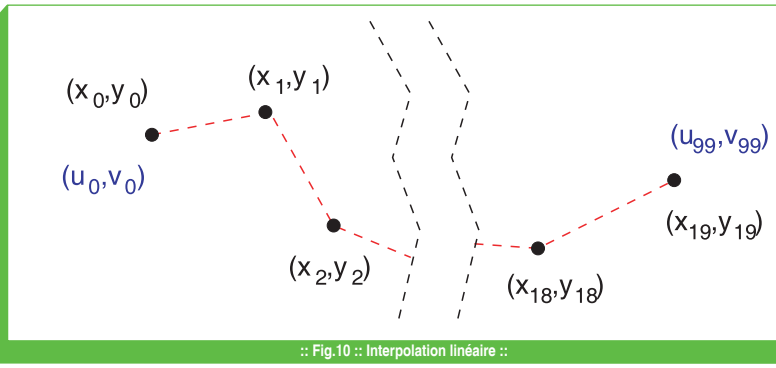
La première transformation proposée simplifie la comparaison des tracés, sans pour autant régler le problème de la variabilité de leur longueur. Une solution possible consiste à réaliser une interpolation. L'idée est très simple : nous allons nous arranger pour rendre les tracés comparables de façon élémentaire, en « remplissant les trous ». Dans la section précédente, nous avons vu comment transformer le tracé de manière à se ramener au cas simple dans lequel tout tracé est réalisé au centre d'une zone de taille fixe : nous normalisons l'espace. Le but de l'interpolation est de normaliser le temps, c'est-à-dire de s'arranger pour avoir des tracés de durée (et donc de longueur) fixée.

Pour comprendre l'interpolation, il est utile de revenir sur la figure 1. Les points représentent en effet les positions enregistrées par le capteur. En ce sens, comme pour la figure 2 d'ailleurs, le dessin est mensonger : les lignes qui joignent les points ne sont pas connues de l'ordinateur qui n'a enregistré que les listes de positions matérialisées sur les figures par les points numérotés. L'interpolation consiste justement à relier les points, c'est-à-dire à construire la portion de courbe qui manque entre deux points. La solution la plus simple pour réaliser cette tâche consiste à relier deux points consécutifs par des segments de droite : il s'agit d'une interpolation dite linéaire (par morceaux). Une solution plus évoluée passe par l'utilisation de splines (c'est ce qui a été utilisé pour les figures) : pour faire simple, il s'agit non plus de relier les points par des segments, mais par des courbes qui permettent d'éviter les points anguleux induits par les segments. L'interpolation par splines est bien décrite dans le cours d'analyse numérique de Jean-Michel Jolion (cf [3]) ainsi que dans les Numerical Recipes (cf [4]). Attention, le code contenu dans ce dernier ouvrage n'est pas libre et il est donc souhaitable de se contenter de la description algorithmique. On trouvera une implémentation libre d'une interpolation par splines dans la GSL [5]. Notons d'autre part que les codes et présentations sont en général destinés à une interpolation de fonction et non pas de courbe. Le passage à la notion de courbe implique en fait d'interpoler deux fonctions : $x(t)$ qui donne l'abscisse en fonction du temps et $y(t)$ qui donne l'ordonnée en fonction du temps (cf [4]).

Une fois qu'on a obtenu une courbe continue qui relie les positions enregistrées par le capteur, on transforme cette courbe en une liste de points d'une taille fixée à l'avance qui sera la même pour tous les tracés. Considérons un exemple simple pour bien comprendre comment mettre en oeuvre ce principe. Supposons qu'on décide arbitrairement de s'intéresser à des tracés contenant 100 points numérotés de 0 à 99. Soit maintenant un tracé comportant seulement 20 positions, les (x_i,y_i) pour i variant entre 0 et 19. Le but de la mise à l'échelle temporelle consiste donc à fabriquer une nouvelle liste de positions, les (u_j,v_j) pour j variant entre 0 et 99. Bien qu'il existe des solutions plus subtiles, on se contente en général de prendre comme début et fin de la nouvelle liste les positions des points correspondants de l'ancienne liste. Dans notre exemple on a donc $(u_0,v_0)=(x_0,y_0)$ (point de départ) et $(u_{99},v_{99})=(x_{19},y_{19})$ (point d'arrivée). La figure 10 illustre la situation d'origine, avant le calcul des autres points de la nouvelle liste (en bleu).

être entre le premier et le deuxième anciens points. De façon générale, on a n points dans la nouvelle liste et a points dans l'ancienne. Le point numéro k de la nouvelle liste correspond donc au point numéro $(kx(a-1))/(n-1)$ de l'ancienne liste. En général, ce nombre n'est pas entier, ce qui signifie que le point qu'on recherche est situé entre les deux points dont les numéros encadrent la valeur obtenue. En revenant à notre exemple, on a $(1x19)/99 = 0,19$ et donc le nouveau point numéro 1 est bien situé entre les anciens points 0 et 1. On remarque que l'application de la règle proposée permet de retrouver les règles spécifiques au premier et au dernier points. Par exemple, le dernier point de la nouvelle liste porte le numéro $n-1$ et la formule donne donc $a-1$, soit le dernier point de l'ancienne liste. En inversant la formule, on passe du numéro d'un point dans l'ancienne liste m au numéro correspondant dans la nouvelle, soit $(mx(n-1)/(a-1))$.

Le problème bien entendu est que pour la plupart des nouveaux points, nous allons tomber entre deux points de l'ancienne liste. C'est ici que l'interpolation entre en jeu. En effet, elle relie les positions enregistrées par le capteur entre elles, ce qui permet d'introduire des nouveaux points de façon simple et cohérente avec la forme générale du tracé. Continuons donc notre calcul avec une interpolation linéaire. Les points situés entre (x_0,y_0) et (x_1,y_1) se placent donc sur le segment reliant les deux positions. Or,



:: Fig.10 :: Interpolation linéaire ::

La situation se complique bien sûr pour les autres points ! Considérons par exemple le deuxième point de la nouvelle liste. Comme on doit transformer 20 positions en 100, il est clair que le deuxième nouveau point doit

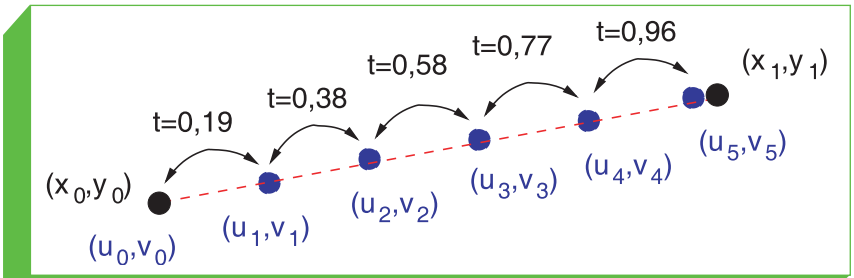
pour tout point de ce segment, il existe un nombre t compris entre 0 et 1 tel que les coordonnées (u,v) du point s'écrivent : $u=(1-t)x_0+tx_1$ et $v=(1-t)y_0+ty_1$. De plus, en faisant

varier *t* entre 0 et 1, on obtient bien tout le segment. Pour trouver le point qui nous intéresse, il suffit donc de choisir la bonne valeur de *t*. Au sens de *t*, la distance entre les extrémités du segment est de 1, puisqu'on obtient une extrémité avec *t=0* et une autre avec *t=1*. Par contre, au sens de la nouvelle courbe, la distance est de *(n-1)/(a-1)*. En effet, le numéro de (x_0,y_0) dans la nouvelle liste est 0 (car c'est le premier point), alors que le numéro de (x_1,y_1) est donné par la formule proposée au dessus. Pour obtenir le t correspondant au deuxième point de la nouvelle liste, il faut donc diviser 1 (la distance dans la nouvelle liste entre (x_0,y_0) et le point cherché) par *(n-1)/(a-1)*. On obtient donc *t=(a-1)/(n-1)*, c'est-à-dire *19/99* dans notre exemple. La figure 11 représente la construction des premiers éléments de la nouvelle liste.

- Entrées : une liste (x_i,y_i) pour *i* variant entre 0 et *a-1*
- Sorties : une liste (u_j,v_j) pour *j* variant entre 0 et *n-1*
- Déroulement :

  1. pour *j* allant de 0 à *n-1* :
    1. calculer m=(jx(a-1))/(n-1)
    2. calculer k=[m], où [m] désigne la partie entière de m
    3. calculer t=m-k
    4. calculer $u_j=(1-t)x_k+tx_{k+1}$
    5. calculer $v_j=(1-t)y_k+ty_{k+1}$

Il y a une seule petite astuce dans cet algorithme : les étapes 4 et 5 peuvent poser problème pour *j=n-1*, puisque *k=a-1* et que x_{k+1} n'est donc pas défini. Mais en fait, on a alors *t=0*, ce qui règle le problème. Il faut cependant être attentif lors de l'implémentation.



:: Fig.11 :: Zoom sur l'interpolation ::

Pour bien fixer les idées, cherchons les coordonnées du point numéro 37 de la nouvelle liste. D'après la formule de passage à l'ancienne liste, on cherche donc le point numéro *(37x19)/99 = 7,1*. Il est donc situé entre les points numéro 7 et 8. La distance entre ces points au sens de la nouvelle liste est toujours de 99/19.

Le numéro de (x_7,y_7) dans la nouvelle liste est de 36,47 et la distance avec le point recherché est donc de 37-36,47=0,53. En divisant cette valeur par 99/19, on obtient 0,1, la valeur à utiliser pour t. On remarque qu'il s'agit de 7,1-7. En fait, on peut obtenir directement la valeur de t en prenant la valeur fractionnaire (après la virgule) du numéro du nouveau point dans l'ancienne liste. Finalement, on obtient donc $u_{37}=0,9x_7+0,1x_8$ et $v_{37}=0,9y_7+0,1y_8$.

L'algorithme général de la mise à l'échelle temporelle est donc le suivant :

Si on utilise une interpolation autre que l'interpolation linéaire (par exemple par splines), le calcul est légèrement différent (en particulier pour les étapes 4 et 5 qui se compliquent), mais le principe général reste le même.

Pour conclure cette partie sur l'interpolation, notons que nous réalisons ici une mise à l'échelle totalement temporelle. Nous aurions pu tenir compte de la distance entre les positions du tracé initial pour normaliser cette longueur plutôt que la durée totale du tracé. L'algorithme final serait légèrement différent, mais seulement dans les détails, pas dans le principe.

Alternatives à l'interpolation

Si on souhaite transformer les listes qui décrivent des tracés en des objets de taille fixe, l'interpolation n'est pas la seule solution. Cette technique est en effet d'assez bas niveau, puisqu'elle pourrait s'appliquer à des courbes quelconques et pas seulement à des caractères. Les techniques alternatives tentent d'utiliser les spécificités de l'écriture manuscrite pour définir des pré-traitements plus intelligents. L'idée est d'extraire du tracé un nombre fixe de caractéristiques choisies par des experts du domaine à partir de connaissances fines sur l'écriture manuscrite.

Les caractéristiques extraites sont assez variées. On peut considérer la longueur de la plus grande section linéaire, le rapport entre la largeur et la hauteur, le nombre de boucles, etc. Le lecteur intéressé trouvera dans [6] une description très complète de caractéristiques utiles pour la reconnaissance de caractères. Notons pour conclure cette section qu'il est tout à fait possible de combiner l'extraction de caractéristiques expertes avec l'interpolation ou avec les solutions qui fonctionnent même pour des données de tailles différentes.

Reconnaître les gestes

Une fois le tracé transformé pour le rendre facilement manipulable par l'ordinateur, le but est de lui associer un caractère ou une action, c'est-à-dire plus généralement de reconnaître le tracé. Je vais présenter ici un algorithme très intéressant qui peut s'appliquer dans de nombreux problèmes de reconnaissance de formes, et pas seulement à celui qui nous intéresse dans cet article.

L'apprentissage automatique

Avant de présenter l'algorithme retenu, j'aimerais insister sur un point très important qui concerne les applications de l'IA en général. Il est fréquent d'être confronté à des situations dans lesquelles on doit régler un grand nombre de paramètres pour obtenir un algorithme qui fonctionne. C'est le cas par exemple dans les applications de filtrage de spam comme Spamassassin [7] : ce système fonctionne grâce à une base de règles. Un mail entrant est confronté à cette base : dès que les conditions associées à la règle sont remplies, le niveau de spam

du mail augmente. Quand le niveau dépasse un certain seuil, le mail est rejeté. Pour que Spamassassin fonctionne, il faut écrire les règles et associer à chacune d'elle un niveau de spam (notons que les versions récentes de Spamassassin combinent une base de règles avec un apprentissage Bayésien, cf la suite du texte). On trouve un autre exemple dans le moteur d'intelligence artificielle du jeu d'échecs présenté dans [8]. Pour appliquer l'algorithme alpha-beta, il faut construire une fonction d'évaluation de l'échiquier qui doit associer une valeur aux positions des pièces.

Quand les paramètres (au sens large) sont bien réglés, les algorithmes obtenus donnent en général entière satisfaction. Malheureusement, le réglage est extrêmement difficile. Pour réaliser un bon programme d'échecs, il faut non seulement maîtriser l'algorithmique associé, mais surtout être capable de construire une fonction d'évaluation performante. De même, la force de Spamassassin ne réside pas dans son algorithme, mais bien dans sa base de règles.

En un certain sens, les méthodes qui reposent sur des éléments non algorithmiques qui doivent être réglés par un expert ne sont par vraiment « intelligentes ». En fait, la majeure partie de l'intelligence réside justement dans les éléments qui sont fournis pas l'expert. Or, la traduction d'un savoir expert en un programme est un processus coûteux, long et difficile. C'est pourquoi l'un des enjeux majeurs de la recherche en IA est de se passer de cet étape en favorisant les solutions basées sur l'apprentissage automatique (le *machine learning* en anglais). Le principe de base est de faire faire à l'ordinateur l'extraction et la modélisation de connaissances qu'on demande habituellement à l'expert. Pour ce faire, on procède en deux phases. Dans une première phase, un algorithme dit d'**apprentissage** analyse un ensemble d'exemples du comportement attendu pour le programme final (il s'agit de l'ensemble d'apprentissage). Le résultat de cette analyse est un algorithme complètement spécifié, c'est-à-dire sans paramètre à régler. La deuxième phase consiste en l'utilisation de l'algorithme produit par la première phase pour réaliser la tâche qui incombe au programme final.

Considérons par exemple le filtrage de *spams* [9]. Dans une première phase, on soumet à l'algorithme d'apprentissage un ensemble de *mails* dont on a précisé le statut, *spam* ou *ham*. Ce statut a bien sûr été déterminé par un expert humain. La grosse différence avec le système de règles de Spamassassin est que l'expert n'a pas à rendre explicite sa méthode : il doit simplement dire s'il considère qu'un mail est un spam ou pas, sans motiver les raisons de son choix. L'algorithme d'apprentissage déduit de l'ensemble de mails un algorithme (au sens large) de filtrage. Dans la deuxième phase, l'utilisateur reçoit un *mail* et le soumet à l'algorithme de filtrage qui lui répond en proposant un statut pour le *mail*. Si l'algorithme d'apprentissage a bien fait son travail et si l'ensemble de mails est représentatif du problème, le filtrage devrait être assez bon. En pratique, l'utilisation d'un réseau Bayésien très simple permet d'obtenir des filtres très efficaces (cf [9,10]). La phase d'apprentissage consiste à déterminer la structure du réseau Bayésien et les valeurs des différentes probabilités qui le caractérisent. L'algorithme de filtrage est alors celui qui calcule la probabilité que le *mail* reçu soit un *spam* en utilisant le réseau construit par l'algorithme d'apprentissage. En pratique, on est toujours dans ce type de situation : l'algorithme d'apprentissage règle des paramètres pour un algorithme fixé qui sera utilisé dans la deuxième phase. Il est très rare que l'algorithme d'apprentissage construise véritablement un autre algorithme à partir de rien.

L'algorithme des k plus proches voisins

La présente application est un cas particulier de reconnaissance de formes. Il s'agit en effet d'attribuer à une forme une classe ou une étiquette choisie parmi un nombre fixe prédéterminé de possibilités (par exemple spam/ham, ou encore « a »/« b »/.../« z »). De très nombreux algorithmes d'apprentissage automatique peuvent être utilisés pour résoudre les problèmes de reconnaissance de formes. L'un des plus connus et des plus simples est celui dit **des k plus proches voisins**. Malgré sa simplicité, il obtient en général de bonnes performances (ce point est même démontré mathématiquement pour certains problèmes). L'idée est vraiment basique. Comme toujours en *machine learning*, on suppose donné un ensemble d'apprentissage. Pour la reconnaissance de formes, il s'agit d'un ensemble de formes avec pour chacune d'elles l'étiquette souhaitée, c'est-à-dire une liste de couples (f_i,e_i) pour i allant de 0 à $N-1$ (f_i désigne une forme et e_i l'étiquette associée). Dans notre cas particulier, il s'agit donc d'un ensemble de tracés pour lesquels on connaît le caractère dessiné. La phase d'apprentissage est réduite au choix d'un entier k optimal. Comme elle s'appuie sur l'algorithme retenu pour la seconde phase, nous allons d'abord décrire ce dernier. Le principe est tout simplement de déterminer les k formes les plus proches de la forme à classer, puis de choisir comme étiquette celle qui est majoritaire parmi les étiquettes des formes proches (cf la figure 12). Plus formellement, l'algorithme est le suivant :

- Entrées : l'ensemble d'apprentissage (f_i,e_i) pour i allant de 0 à $N-1$, une nouvelle forme à classer, f, et k un entier
- Sortie : une étiquette pour f
- Déroulement :

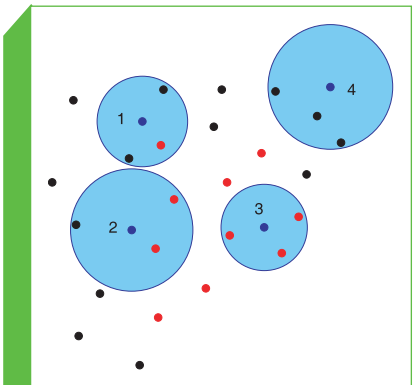
  - 1. allouer un tableau *ind* pouvant contenir k indices et un autre *dist* pour k distances (ce dernier doit être initialisé à l'infini dans toutes les cases)
  - 2. pour i allant de 0 à $N-1$
    - A. calculer la distance d_i entre f_i et f
    - B. initialiser j à $k-1$
    - C. tant que $j>=0$ et que d_i est strictement plus petit que *dist[j]*
      - I. placer *dist[j]* dans *dist[j+1]* (quand c'est possible)
      - II. placer *ind[j]* dans *ind[j+1]* (quand c'est possible)
      - III. placer *j-1* dans *j*

>>>■ D. si j<k-1
>>>>■ I. placer d_j dans *dist[j+1]*
>>>>■ II. placer *i* dans *ind[j+1]*
>>■ 3. allouer un tableau *eti* pouvant contenir *k* entiers, initialisé à 0 (sauf *eti[0]* qui doit valoir 1)
>>■ 4. pour *i* allant de 1 à *k-1*
>>>■ A. pour *j* allant de 0 à *i-1*
>>>>■ I. sortir de la boucle si $e_{ind[i]}==e_{ind[j]}$
>>>■ B. incrémenter *eti[j]* de 1
>>■ 5. trouver *r* tel que *eti[r]* soit maximal (par une simple boucle)
>>■ 6. le résultat est $e_{ind[r]}$

Détaillons les étapes de l'algorithme. La boucle 2 se contente de déterminer les *k* plus proches voisins de la forme *f*. En effet, elle calcule la distance entre chaque f_i et *f* (étape 2A), tout en conservant un tableau trié des *k* plus petites distances rencontrées jusqu'à présent (le tableau *dist*) associé aux indices des formes correspondantes (le tableau *ind*). Pour ce faire, la boucle 2C cherche l'emplacement de la distance d_i dans le tableau *dist* et en cas de succès insère à cet endroit la valeur, en décalant celles qui lui sont supérieures (2CI et 2CII dans la boucle pour le décalage, 2D pour l'insertion). Si *k* est grand, la technique employée est lente (il s'agit d'une forme de tri par insertion de complexité en *O(Nk)*) et il est préférable d'utiliser des techniques plus évoluées (comme un tas qui permet de passer en *O(N log k)*). Il existe aussi des techniques qui permettent de réduire la complexité en *N*.

La deuxième étape consiste à trouver l'étiquette la plus fréquente parmi celles des *k* plus proches voisins de *f*. Pour ce faire, on utilise un tableau *eti* qui compte le nombre d'apparitions de chaque étiquette. La boucle 4 parcourt la liste des *k* étiquettes. A chaque itération, la boucle 4A recherche une précédente apparition de l'étiquette testée. Comme dans la recherche des voisins, cet algorithme est assez basique et ne fonctionne efficacement que pour les petites valeurs de *k* (complexité en $O(k^2)$). Pour de plus grandes valeurs, il faut songer à utiliser une table de hachage sur les étiquettes ou encore un tableau indexé par les étiquettes si elles sont peu nombreuses.

La figure 12 illustre le fonctionnement de l'algorithme sur un cas simple avec deux étiquettes possibles, rouge et noire. Les points de ces couleurs sont les exemples de l'ensemble d'apprentissage. Les points bleus constituent de nouvelles formes à classer. Considérons par exemple le point 1 : les trois points les plus proches sont respectivement un point rouge et deux points noirs. On attribue donc au point 1 une étiquette noire (pour l'algorithme des 3 plus proches voisins). Pour le point 2, on trouve 2 voisins rouges et un voisin noir, d'où l'attribution d'une étiquette rouge. Enfin, les points 3 et 4 ont des voisins uniformes en termes d'étiquette et reçoivent donc respectivement une étiquette rouge et une étiquette noire. On remarque que le passage à l'algorithme du plus proche voisin (*k=1*) change seulement l'étiquette du point 1.



:: Fig. 12 :: Un exemple d'application avec 3 voisins ::

L'algorithme présenté est une version très simple qui ne s'embarrasse pas des cas particuliers. Il est en effet possible d'avoir une égalité dans les nombres d'apparitions des étiquettes dans les plus proches voisins. Dans ce cas, diverses stratégies sont envisageables, comme par exemple de choisir au hasard l'étiquette renvoyée parmi les plus fréquentes. Dans le cas d'un problème à deux classes (deux étiquettes possibles comme sur la figure 12), on prend une valeur impaire pour *k*, ce qui évite tout problème.

Notons pour finir que l'algorithme repose entièrement sur un calcul de distance. Il faut donc définir celle-ci de façon intelligente puisque le principe de l'algorithme est de dire que des formes proches (au sens de la distance) se ressemblent en termes d'étiquette. En d'autres termes, si je trace quelque chose qui ressemble à un « s », l'algorithme donnera l'étiquette « s » à ma production. Il faut donc que la notion de ressemblance induite par la distance soit adaptée au problème ce qui montre que l'expert n'est pas totalement inutile, même avec un algorithme d'apprentissage automatique !

Choisir k

Le seul paramètre de l'algorithme est la valeur de *k* qui doit donc être choisie par la phase d'apprentissage. Diverses stratégies sont possibles :

>■ L'*a priori* pur (un expert choisit « à la main » la valeur de *k*) : par exemple, pour des raisons mathématiques assez complexes, prendre comme valeur *k=3* est une bonne idée quand on travaille avec 2 classes (i.e., quand il n'y a que deux étiquettes possibles). De façon générale, certaines considérations expertes peuvent guider un choix *a priori* du nombre de voisins.
>■ Le choix automatique à partir de l'ensemble d'apprentissage : divers algorithmes permettent de régler automatiquement k à partir de l'ensemble d'apprentissage. Le plus simple d'entre eux est celui dit de « **validation croisée** ». Sa mise en œuvre est assez élémentaire. On commence par couper l'ensemble d'apprentissage en deux sous-ensembles de même taille, *A* et *B*, ceci de façon aléatoire. Pour chaque valeur de k envisageable (par exemple de 1 à 10), on applique l'algorithme des *k* plus proches voisins pour étiqueter les formes de *B* en utilisant comme ensemble d'apprentissage les formes de *A*. Comme on connaît les véritables étiquettes des formes de *B*, on en déduit le nombre d'erreurs d'étiquetage réalisées par l'algorithme pour les différentes valeurs de *k*. On choisit alors le *k* qui donne le moins d'erreurs. Il est très important de ne pas utiliser les formes de B pour appliquer les k plus proches voisins sur elles mêmes. En effet, en procédant de cette façon, on trouverait toujours que k=1 est la meilleure solution, puisque toute forme de

B se retrouverait dans l'ensemble d'apprentissage et donc dans ses propres plus proches voisins ! Diverses variantes de la validation croisée ont été proposées depuis de nombreuses années. Elles améliorent en général le choix de k, au prix de calculs plus longs et plus complexes.

Données de longueur fixe

Pour appliquer l'algorithme des k plus proches voisins à notre problème, il nous reste « simplement » à définir une distance qui permette de comparer deux tracés entre eux. Grâce aux pré-traitements proposés dans les sections précédentes, il est possible de se ramener au cas de données de longueur fixe. En d'autres termes, on peut convertir les tracés en une liste de p nombres réels (p est pair car on a deux coordonnées par position dans le tracé).

De façon générale, quand chaque forme est décrite par p nombres réels, on utilise classiquement la distance euclidienne pour mesurer les différences entre les formes (nous considérons en fait le carré de la distance euclidienne pour éviter des calculs inutiles). Plus précisément, si f et g sont deux formes, on a donc f qui est décrite par $(f_0,f_2,...,f_{p-1})$ et g par $(g_0,g_2,...,g_{p-1})$. La distance $d(f,g)$ entre f et g est donnée par :

$$d(f,g)=\sum_{i=0}^{p-1}(f_i-g_i)^2=(f_0-g_0)^2+(f_1-g_1)^2+\cdots+(f_{p-1}-g_{p-1})^2$$

Cette construction s'applique de façon très générale à des données de taille fixe et donc en particulier à notre problème si on utilise la méthode d'interpolation décrite précédemment. Par contre, l'interpolation nécessite le choix d'un paramètre additionnel : le nombre de points considérés dans la phase de mise à l'échelle temporelle (en multipliant le nombre de points par 2, on obtient le p de la formule ci-dessus). Ce paramètre pourra être choisi de la même façon que k, soit de façon experte, soit par un procédé de validation croisée, plus coûteux en temps de calcul mais plus performant.

Données brutes

L'algorithme des k plus proches voisins peut être mis en œuvre dès qu'une distance est définie sur les formes étudiées. Plutôt que de pré-traiter les tracés par interpolation, une solution alternative consiste à définir une distance calculable malgré les différences de tailles des listes de positions. Une telle distance a été proposée dans les années 1970, sous le nom de *Dynamic Time Warping* (DTW) ou d'*elastic matching,* pour comparer des enregistrements audio afin de reconnaître la parole. Charles Tappert a eu l'idée d'adapter cette technique à la reconnaissance de caractères en 1982 [11].

Le principe du DTW est de comparer les éléments des tracés deux à deux d'une façon assez souple, en autorisant une déformation du temps qui généralise l'idée de la mise à l'échelle temporelle opérée par l'interpolation. Considérons deux objets à comparer, décrits par des suites d'observations, éventuellement de longueurs différentes, soit donc $A=(a_0,a_2,...,a_{n-1})$ et $B=(b_0,b_2,...,b_{m-1})$. Dans notre application, les objets sont les tracés, et les observations sont les positions enregistrées par le capteur (après centrage et mise à l'échelle). On suppose qu'on peut comparer deux observations grâce à une distance d. Dans notre application, la distance peut être par exemple la distance euclidienne classique dans le plan. Pour pouvoir appliquer le DTW, il faut impérativement que les objets soient décrits par des suites, c'est-à-dire que l'ordre des observations ait un sens. Dans notre application, l'ordre correspond au tracé et le DTW est bien applicable.

Contrairement aux données de longueur fixe, on ne peut pas se contenter ici de comparer les a_i et les b_i. Le but du DTW est de définir un alignement entre les suites qui contourne le problème de leurs longueurs. Pour ce faire, on définit un alignement comme une suite de paires d'entiers, $W=(w_0=(u_0,v_0), w_1=(u_1,v_1),...,w_{p-1}=(u_{p-1},v_{p-1}))$ qui vérifie certaines propriétés que nous détaillerons par la suite. Comparer A et B grâce à l'alignement W revient à comparer a_{u0} avec b_{v0}, puis a_{u1} avec b_{v1}, et ainsi de suite jusqu'à a_{up-1} avec b_{vp-1}. Pour que cette comparaison soit pertinente, il faut que W vérifie les propriétés suivantes :

- 1. on compare obligatoirement les « extrémités » objets, c'est-à-dire que $w_1=(1,1)$ et $w_{p-1}=(n-1,m-1)$. En d'autres termes, on compare a_1 avec b_1 et a_n avec b_m, des opérations qu'on réalise aussi quand on interpole les tracés ;
- 2. l'alignement doit être **monotone**, c'est-à-dire que pour k compris en 0 et $p-2$, on a $u_{k+1}\geqslant u_k$ et $v_{k+1}\geqslant v_k$. En pratique, cela signifie qu'on ne peut pas revenir en arrière dans la comparaison : si on a déjà comparé a_2 avec b_3, on ne peut pas ensuite comparer a_1 avec b_3, mais on peut comparer a_3 avec b_3 par exemple ;
- 3. l'alignement doit être continu, c'est-à-dire que pour k compris en 0 et p-2, on a $u_{k+1}-u_k\leqslant 1$ et $v_{k+1}-v_k\leqslant 1$. En pratique, cela signifie qu'on ne peut sauter des points dans les deux suites. Ainsi, comme on commence par comparer a_1 avec b_1, on doit comparer ensuite a_1 avec b_2 ou a_2 avec b_1 ou encore a_2 avec b_2.

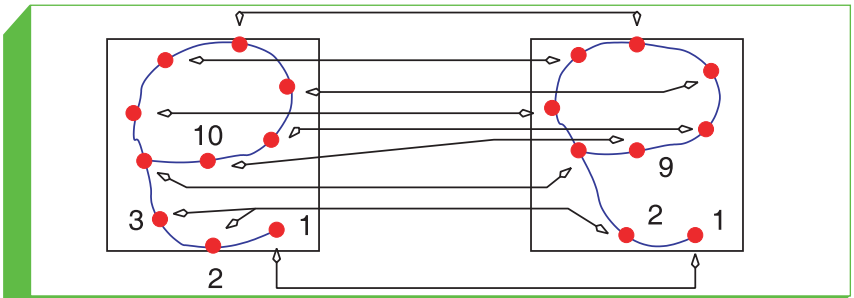
Une conséquence de ces propriétés est que la longueur p d'un alignement vérifie $\max(n,m)\leqslant p<m+n-1$. La distance entre les deux séquences A et B au sens de l'alignement W est donnée par la formule suivante :

$$D(A,B,W)=\sum_{k=0}^{p-1}d(a_{uk},b_{vk})$$

La distance entre A et B au sens du DTW (notée $DTW(A,B)$) est tout simplement celle donnée par l'alignement optimal, autrement dit la plus petite distance $D(A,B,W)$ quand W est un alignement correct (c'est-à-dire qui vérifie les trois conditions énoncées ci-dessus).

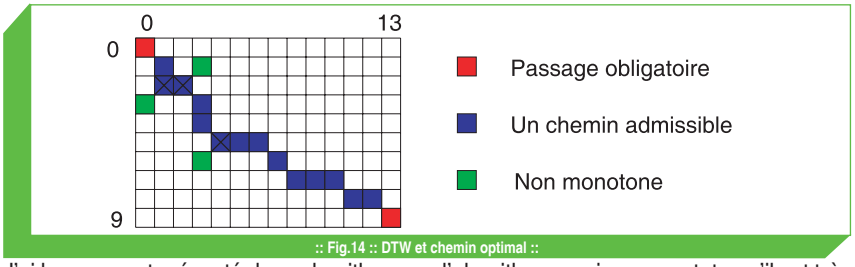
On voit que l'idée de base consiste à compenser les différences de longueur en comparant une valeur de la suite la plus courte avec plusieurs valeurs consécutives de la suite la plus longue. En fait, l'algorithme est même plus souple que cela car il peut aussi comparer une valeur de la suite la plus longue avec plusieurs valeurs consécutives de la suite la plus courte. L'idée est de s'affranchir des distorsions temporelles, par exemple celles induites par les conditions de tracé dans le cas de notre

application (quand on est coincé dans le métro, il est parfois difficile d'écrire sur son PDA...). La figure 13 illustre le principe du DTW appliqué aux deux tracés de la figure 5 (après centrage et mise à l'échelle). Le tracé de gauche comporte 10 points et celui de droite 9 : l'alignement n'est donc pas très difficile. On constate que l'algorithme choisit de comparer les points 2 et 3 du tracé de gauche au point 2 du tracé de droite. Pour les autres points, la comparaison est plus directe au sens où on compare simplement des points deux à deux sans comparer un point d'un tracé à plusieurs autres points de l'autre tracé.



:: Fig.13 :: Un exemple de dynamic time warping (chaque flèche indique une comparaison) ::

Pour mieux comprendre le DTW, on peut l'interpréter en termes de chemin optimal. On considère en effet un terrain rectangulaire constitué de n lignes pour m colonnes. Le passage d'un chemin par la case de coordonnées *(u,v)* contribue au coût total du chemin à hauteur de $d(a_u,b_v)$. Le but du jeu est de construire un chemin de coût minimal partant de la case *(0,0)* et arrivant à la case *(n-1,m-1)*. Ce chemin doit être continu (on ne peut passer que d'une case à une de ses voisines) et monotone, c'est-à-dire qu'il doit aller vers le bas et vers la droite du terrain (si le point de coordonnées *(0,0)* est situé en haut à gauche). La figure 14 illustre cette interprétation pour la recherche de l'alignement d'une séquence de 10 éléments avec une séquence de 14 éléments. Le chemin en bleu est admissible, c'est-à-dire qu'il respecte les conditions énoncées précédemment. Par contre, les cases en vert n'étaient pas des bifurcations acceptables depuis les cases marquées d'une croix car elles allaient à l'encontre de la règle de monotonie.



:: Fig.14 :: DTW et chemin optimal ::

J'ai longuement présenté deux algorithmes de calcul de chemin optimal dans de précédents articles de la rubrique IA (cf [12] et [13]). Cependant, le problème étudié ici possède une particularité intéressante qui permet d'éviter d'avoir recours à des algorithmes sophistiqués comme celui de Dijkstra [12] ou A* [13] : les chemins explorables sont très contraints par les trois conditions d'admissibilité d'un alignement. On a alors intérêt à utiliser un algorithme dit de programmation dynamique qui fonctionne de la façon suivante :

- Entrées : les objets $A=(a_0,a_2,...,a_{n-1})$ et $B=(b_0,b_2,...,b_{m-1})$
- Sortie : DTW(A,B)
- Déroulement :

  - 1. allouer une matrice *nxm M* initialisée à 0
  - 2. fixer M[0,0] à $d(a_0,b_0)$
  - 3. pour *i* allant de 0 à *n-1*
    - 1. pour *j* allant de 0 à *m*-1
      - 1. déterminer *u* comme le minimum de *M[i-1,j-1]*, *M[i-1,j]* et *M[i,j-1]* (en ne prenant en considération que les indices valides)
      - 2. placer dans *M[i,j]* la valeur $d(a_i,b_j)+u$
  - 4. le résultat est *M[n-1,m-1]*

Bien entendu, la toute première détermination de *u* pour *i=0* et *j=0* est impossible, ce qui explique l'initialisation de *M[0,0]*. En pratique, il faudra donc être attentif dans l'implémentation exacte de l'algorithme, mais on constate qu'il est très simple et de complexité *O(nm)*, tant en mémoire qu'en temps.

Résumé des solutions

Nous avons donc au final deux solutions possibles pour reconnaître un tracé : soit nous réalisons une interpolation, soit nous utilisons le *Dynamic Time Warping*. Tout le reste, en particulier l'utilisation de l'algorithme des k plus proches voisins, est commun aux deux solutions. Les différences sont donc minimes, ce qui permet de présenter une méthodologie générale unifiée :

- 1. on commence donc par obtenir un ensemble d'apprentissage constitué de tracés étiquetés par un expert. Il est souhaitable d'avoir plusieurs tracés par caractères, au moins une dizaine. Plus on dispose de tracés, plus la reconnaissance est précise, au prix d'un temps de calcul accru ;
- 2. les tracés sont centrés et mis à l'échelle pour simplifier leur comparaison ;
- 3. l'ensemble d'apprentissage est coupé en deux sous-ensembles pour déterminer le paramètre *k* de l'algorithme des plus proches voisins par validation croisée. Dans le cas d'une solution par interpolation, le nombre de points retenus pour la mise à l'échelle temporelle est déterminé de la même façon. La distance utilisée pour les plus proches voisins dépend de la solution retenue (euclidienne ou DTW) ;
- 4. dans le cas de l'interpolation, les tracés sont mis à l'échelle temporelle en utilisant le paramètre déterminé à l'étape 3 ;
- 5. pour classer un nouveau tracé :
  - 1. on le centre et on le met à l'échelle ;

>> 2. dans le cas d'une interpolation, on met le tracé à l'échelle temporelle ;

>> 3. on détermine l'étiquette du tracé grâce à l'algorithme des k plus proches voisins, en utilisant la distance appropriée.

Chaque solution a des avantages et des inconvénients :

> L'interpolation est coûteuse pendant la phase d'apprentissage car il faut choisir le paramètre *p* qui donne le nombre de points considérés pour la mise à l'échelle temporelle ;

>> Le DTW est coûteux dans la phase de mise en œuvre car le calcul de la distance DTW est en $O(m^2)$ si *m* désigne le nombre de positions dans le plus long tracé envisagé. Au contraire, la distance euclidienne se calcule en $O(p)$ où *p* est le nombre de points considérés pour la mise à l'échelle temporelle (en général *p* est du même ordre de grandeur que *m*) ;

>> L'interpolation est moins souple que le DTW et reconnaît donc moins facilement des caractères vraiment mal tracés.

En général, l'approche par DTW donne de meilleurs résultats en termes de taux de reconnaissance, mais l'algorithme est aussi plus lourd, ce qui limite son application à des PDA puissants.

Autres solutions

Bien que les solutions présentées dans cet article soient assez anciennes, elles sont très performantes en pratique. Cependant, elles ont un gros défaut : l'algorithme des k plus proches voisins a une complexité en mémoire assez importante. En effet, on doit stocker l'intégralité de l'ensemble d'apprentissage pour faire tourner l'algorithme sur un nouveau tracé. Sur un PDA, le moindre octet de mémoire doit être économisé et la consommation de l'algorithme est donc un vrai problème.

De nombreux travaux ont été menés depuis l'invention de l'algorithme afin de contourner le problème. L'idée de base est de réduire l'ensemble d'apprentissage en ne conservant que les exemples importants. Les algorithmes utilisés dépassent le cadre de cet article, mais ils permettent de limiter l'occupation mémoire de la méthode et de faciliter son utilisation sur des systèmes embarqués très peu puissants.

Dans des situations vraiment extrêmes, en particulier au début des années 1990 quand avoir 32 Mo de mémoire sur assistant personnel semblait relever de la plus pure science-fiction, l'algorithme des k plus proches voisins n'est pas une option tant pour des raisons d'occupation mémoire que de temps de calcul. On recourt alors à des solutions beaucoup moins performantes en termes de taux de reconnaissance, en les associant à une simplification de l'alphabet.

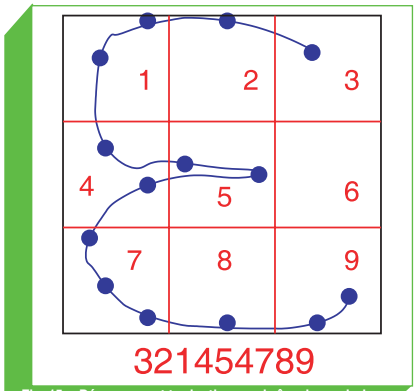
Simplifier le problème

L'idée de base de l'alphabet Graffitifi des Palms™ [2] est de simplifier les caractères afin de faciliter leur reconnaissance (et éventuellement leur tracé, mais ce dernier point est discutable). Au lieu de rendre le PDA intelligent, on utilise l'intelligence de son possesseur pour le faire s'adapter à l'outil. L'apprentissage est rapide et l'aspect ludique de l'objet ne fait que raccourcir la durée subjective de la prise en main.

Pour l'ordinateur, le gain est énorme. Aucune ambiguïté entre le" 1 " et le " l " par exemple, puisque les chiffres sont saisis dans une zone de l'écran et les lettres dans une autre ! Pas de risque de confusion entre un " i " et un " l ", non en raison du point (les Palm™ ne travaillent qu'avec des caractères tracés en une seule fois, au moins pour la version 1 de Graffiti®) mais parce que le « l » doit nécessairement comporter une barre horizontale (c'est en fait un « L ») alors que le « i » est une simple barre verticale.

Codage sous forme d'une chaîne de caractères

Une fois les caractères simplifiés, on peut les reconnaître facilement avec des techniques plus basiques que celles présentées précédemment. Après centrage et mise à l'échelle d'un tracé, on le transforme en effet en une chaîne de symboles en découpant simplement la zone de tracé en petits carrés. La figure 15 illustre la méthode avec l'exemple du tracé qui représente la lettre « e » dans l'alphabet Graffiti® : on découpe la zone de saisie en 9 carrés de taille égale, on attribue à chaque carré un symbole unique (ici un chiffre) puis on associe au tracé la liste des carrés traversés dans l'ordre du dessin.



:: Fig. 15 :: Découpage et traduction en chaîne de symboles ::

Si on fixe le découpage et les symboles associés aux carrés, on peut alors traduire n'importe quel tracé en une chaîne de symboles. Il faut prendre quelques précautions, en particulier éviter d'inclure dans la traduction le symbole d'un carré quand la portion de tracé contenu dans celui-ci est petite, mais, globalement, le principe est très simple.

On demande ensuite à un expert de déterminer les chaînes de symboles associées à chaque caractère de l'alphabet qu'on souhaite reconnaître (ce seront les chaînes canoniques).

Le « i » par exemple doit être tracé verticalement, de haut en bas, ce qui donne la chaîne « 258 ». Étant donné un tracé, il ne reste plus qu'à rechercher dans la liste des chaînes canoniques une correspondance pour lui associer une étiquette.

Comme les chaînes sont assez petites et peu nombreuses (quelques-unes par caractère), la mémoire utilisée par cette méthode est assez limitée.

Quant à la reconnaissance elle-même, elle consiste en une sorte de *k* plus proches voisins avec *k=1* et consomme donc assez

peu de temps calcul : on a peu de chaînes et la comparaison entre deux chaînes est rapide car elles sont courtes. Des variantes plus évoluées remplacent les chaînes canoniques par des expressions régulières (comme celles utilisées par Perl) : on gagne encore en occupation mémoire, au prix d'une recherche un peu plus lente.

Je renvoie les lecteurs intéressés par cette technique de reconnaissance au code source de *xstroke* [14], un logiciel sous GPL qui ajoute à X Window un module de reconnaissance de gestes réalisés à la souris basé sur l'algorithme qui vient d'être présenté. Ce logiciel est utilisé sur les PDA sous Linux pour la reconnaissance de caractères tracés au stylet.

Conclusion

Les algorithmes présentés dans cet article sont très généraux. Ils font partie de la boîte à outils de la reconnaissance de formes et s'appliquent dans des contextes très variés : reconnaissance de l'écriture, de la parole, de visages, etc.

Il existe bien entendu de très nombreux autres algorithmes, en particulier ceux basés sur une approche dite neuronale, c'est-à-dire (vaguement) inspirés de notre cerveau. Nous aurons l'occasion dans de prochains articles de la rubrique IA de côtoyer ces solutions.

En général, elles permettent d'obtenir de meilleures performances que l'algorithme des k plus proches voisins, soit en termes de qualité de la reconnaissance, soit en termes d'efficacité algorithmique (occupation mémoire et temps de calcul).

Elles sont cependant beaucoup plus difficiles à implémenter et demandent un bagage mathématique important : une simple (re)lecture de la présentation des réseaux Bayésiens dans un précédent article de la rubrique IA [10] devrait convaincre le lecteur dubitatif.

Même si ces solutions très évoluées sont indispensables dans certaines applications, la pratique de la reconnaissance de gestes, en particulier dans le cas des alphabets simplifiés des PDA, montre qu'il est parfois beaucoup plus simple de contourner le problème par une technique astucieuse que de l'attaquer frontalement avec des algorithmes très sophistiqués.

Il est en effet édifiant de voir que la simplification de l'alphabet associé avec un codage basique donne aux PDA une interface pilotée par un stylet très efficace et relativement simple d'accès alors que les algorithmes sous-jacents sont plus que basiques.

De plus, les interfaces basées sur des gestes réalisés à la souris (les *mouse gestures*) fonctionnent exactement de la même façon et remportent un grand succès auprès des utilisateurs, par exemple dans Mozilla/Firerox [15].

Cependant, l'expérience prouve aussi que le filtrage de spam n'est devenu vraiment efficace qu'avec la généralisation des techniques d'apprentissage automatique [7].

L'augmentation régulière de la puissance des ordinateurs, même quand ils sont minuscules, plaide sur le long terme pour une généralisation de l'apprentissage automatique, mais il reste encore de nombreuses situations dans lesquelles son application est difficile à envisager et qui nécessitent donc des solutions spécifiques.

Références

>■ [1] Fabrice Rossi - L'ordinateur peut-il lire dans votre esprit, GNU/Linux Magazine France, No 58, Février 2004.

>■ [2] Alphabets Graffiti® et Graffiti® 2 : http://www.palmone.com/us/products/input/

>■ [3] Jean-Michel Jolion - Analyse numérique. Disponible en ligne à l'URL : http://rfv.insa-lyon.fr/~jolion/ANUM/poly.html. La partie qui concerne les splines est intitulée « Interpolation ».

>■ [4] William Press, Saul Teukolsky, William Vetterling, Bian Flannery – Numerical recipes, Cambridge University Press. Disponible en ligne à l'URL : http://www.nr.com/. La partie qui concerne les splines est le chapitre 3 « Interpolation and extrapolation », en particulier les sections 3.3 et 3.6.

>■ [5] Gnu Scientific Library : http://sources.redhat.com/gsl/

>■ [6] Robert Kazimierz Powalka – An Algorithm Toolbox for on-Line Cursive Script Recognition, PhD Thesis, The Nottingham Trent University, 1995. Disponible en ligne : http://citeseer.ist.psu.edu/209471.html

>■ [7] Spamassassin : http://www.spamassassin.org

>■ [8] Eric Lacombe et Fabrice Rossi - L'intelligence artificielle des jeux de stratégie classiques, GNU/Linux Magazine France, No 55, Novembre 2003.

>■ [9] Fabrice Rossi - Filtrage de SPAM par méthodes probabilistes, Multi-System & Internet Security Cookbook (MISC), No 7, Mai 2003.

>■ [10] Jean-Michel Marin et Fabrice Rossi - Découvrez les réseaux Bayésiens, GNU/Linux Magazine France, No 60, Avril 2004.

>■ [11] Charles Tappert - Cursive Script Recognition by Elastic Matching, IBM Journal of Research and Development, volume 26, numéro 6, novembre 1982, pages 765-771. Disponible en ligne : http://www.research.ibm.com/journal/rd/266/ibmrd2606N.pdf

>■ [12] Fabrice Rossi - Recherche des chemins optimaux pour les jeux vidéo, GNU/Linux Magazine France, No 53, Septembre 2003.

>■ [13] Fabrice Rossi - L'algorithme A* ou comment calculer rapidement un chemin optimal, GNU/Linux Magazine France, No 54, Octobre 2003.

>■ [14] X Stroke : http://www.xstroke.org/

>■ [15] Les mouse gestures pour Mozilla/Firefox : http://optimoz.mozdev.org/gestures/

Creative Commons

Paternité - Pas d'Utilisation Commerciale - Pas de Modification 2.0

Creative Commons n'est pas un cabinet d'avocats et ne fournit pas de services de conseil juridique. La distribution de la présente version de ce contrat ne crée aucune relation juridique entre les parties au contrat présenté ci-après et Creative Commons. Creative Commons fournit cette offre de contrat-type en l'état, à seule fin d'information. Creative Commons ne saurait être tenu responsable des éventuels préjudices résultant du contenu ou de l'utilisation de ce contrat.

Contrat

L'Oeuvre (telle que définie ci-dessous) est mise à disposition selon les termes du présent contrat appelé Contrat Public Creative Commons (dénommé ici « CPCC » ou « Contrat »). L'Oeuvre est protégée par le droit de la propriété littéraire et artistique (droit d'auteur, droits voisins, droits des producteurs de bases de données) ou toute autre loi applicable. Toute utilisation de l'Oeuvre autrement qu'explicitement autorisée selon ce Contrat ou le droit applicable est interdite.

L'exercice sur l'Oeuvre de tout droit proposé par le présent contrat vaut acceptation de celui-ci. Selon les termes et les obligations du présent contrat, la partie Offrante propose à la partie Acceptante l'exercice de certains droits présentés ci-après, et l'Acceptant en approuve les termes et conditions d'utilisation.

1. Définitions

a. **« Oeuvre »** : oeuvre de l'esprit protégeable par le droit de la propriété littéraire et artistique ou toute loi applicable et qui est mise à disposition selon les termes du présent Contrat.
b. **« Oeuvre dite Collective »** : une oeuvre dans laquelle l'oeuvre, dans sa forme intégrale et non modifiée, est assemblée en un ensemble collectif avec d'autres contributions qui constituent en elles-mêmes des oeuvres séparées et indépendantes. Constituent notamment des Oeuvres dites Collectives les publications périodiques, les anthologies ou les encyclopédies. Aux termes de la présente autorisation, une oeuvre qui constitue une Oeuvre dite Collective ne sera pas considérée comme une Oeuvre dite Dérivée (telle que définie ci-après).
c. **« Oeuvre dite Dérivée »** : une oeuvre créée soit à partir de l'Oeuvre seule, soit à partir de l'Oeuvre et d'autres oeuvres préexistantes. Constituent notamment des Oeuvres dites Dérivées les traductions, les arrangements musicaux, les adaptations théâtrales, littéraires ou cinématographiques, les enregistrements sonores, les reproductions par un art ou un procédé quelconque, les résumés, ou toute autre forme sous laquelle l'Oeuvre puisse être remaniée, modifiée, transformée ou adaptée, à l'exception d'une oeuvre qui constitue une Oeuvre dite Collective. Une Oeuvre dite Collective ne sera pas considérée comme une Oeuvre dite Dérivée aux termes du présent Contrat. Dans le cas où l'Oeuvre serait une composition musicale ou un enregistrement sonore, la synchronisation de l'oeuvre avec une image animée sera considérée comme une Oeuvre dite Dérivée pour les propos de ce Contrat.
d. **« Auteur original »** : la ou les personnes physiques qui ont créé l'Oeuvre.
e. **« Offrant »** : la ou les personne(s) physique(s) ou morale(s) qui proposent la mise à disposition de l'Oeuvre selon les termes du présent Contrat.
f. **« Acceptant »** : la personne physique ou morale qui accepte le présent contrat et exerce des droits sans en avoir violé les termes au préalable ou qui a reçu l'autorisation expresse de l'Offrant d'exercer des droits dans le cadre du présent contrat malgré une précédente violation de ce contrat.

2. Exceptions aux droits exclusifs. Aucune disposition de ce contrat n'a pour intention de réduire, limiter ou restreindre les prérogatives issues des exceptions aux droits, de l'épuisement des droits ou d'autres limitations aux droits exclusifs des ayants droit selon le droit de la propriété littéraire et artistique ou les autres lois applicables.

3. Autorisation. Soumis aux termes et conditions définis dans cette autorisation, et ceci pendant toute la durée de protection de l'Oeuvre par le droit de la propriété littéraire et artistique ou le droit applicable, l'Offrant accorde à l'Acceptant l'autorisation mondiale d'exercer à titre gratuit et non exclusif les droits suivants :

a. reproduire l'Oeuvre, incorporer l'Oeuvre dans une ou plusieurs Oeuvres dites Collectives et reproduire l'Oeuvre telle qu'incorporée dans lesdites Oeuvres dites Collectives;
b. distribuer des exemplaires ou enregistrements, présenter, représenter ou communiquer l'Oeuvre au public par tout procédé technique, y compris incorporée dans des Oeuvres Collectives;
c. lorsque l'Oeuvre est une base de données, extraire et réutiliser des parties substantielles de l'Oeuvre.

Les droits mentionnés ci-dessus peuvent être exercés sur tous les supports, médias, procédés techniques et formats. Les droits ci-dessus incluent le droit d'effectuer les modifications nécessaires techniquement à l'exercice des droits dans d'autres formats et procédés techniques. L'exercice de tous les droits qui ne sont pas expressément autorisés par l'Offrant ou dont il n'aurait pas la gestion demeure réservé, notamment les mécanismes de gestion collective obligatoire applicables décrits à l'article 4(d).

4. Restrictions. L'autorisation accordée par l'article 3 est expressément assujettie et limitée par le respect des restrictions suivantes :

a. L'Acceptant peut reproduire, distribuer, représenter ou communiquer au public l'Oeuvre y compris par voie numérique uniquement selon les termes de ce Contrat. L'Acceptant doit inclure une copie ou l'adresse Internet (Identifiant Uniforme de Ressource) du présent Contrat à toute reproduction ou enregistrement de l'Oeuvre que l'Acceptant distribue, représente ou communique au public y compris par voie numérique. L'Acceptant ne peut pas offrir ou imposer de conditions d'utilisation de l'Oeuvre qui altèrent ou restreignent les termes du présent Contrat ou l'exercice des droits qui y sont accordés au bénéficiaire. L'Acceptant ne peut pas céder de droits sur l'Oeuvre. L'Acceptant doit conserver intactes toutes les informations qui renvoient à ce Contrat et à l'exonération de responsabilité. L'Acceptant ne peut pas reproduire, distribuer, représenter ou communiquer au public l'Oeuvre, y compris par voie numérique, en utilisant une mesure technique de contrôle d'accès ou de contrôle d'utilisation qui serait contradictoire avec les termes de cet Accord contractuel. Les mentions ci-dessus s'appliquent à l'Oeuvre telle qu'incorporée dans une Oeuvre dite Collective, mais, en dehors de l'Oeuvre en elle-même, ne soumettent pas l'Oeuvre dite Collective, aux termes du présent Contrat. Si l'Acceptant crée une Oeuvre dite Collective, à la demande de tout Offrant, il devra, dans la mesure du possible, retirer de l'Oeuvre dite Collective toute référence au dit Offrant, comme demandé. Si l'Acceptant crée une Oeuvre dite Collective, à la demande de tout Auteur, il devra, dans la mesure du possible, retirer de l'Oeuvre dite Collective toute référence au dit Auteur, comme demandé.

b. L'Acceptant ne peut exercer aucun des droits conférés par l'article 3 avec l'intention ou l'objectif d'obtenir un profit commercial ou une compensation financière personnelle. L'échange de l'Oeuvre avec d'autres Oeuvres protégées par le droit de la propriété littéraire et artistique par le partage électronique de fichiers, ou par tout autre moyen, n'est pas considéré comme un échange avec l'intention ou l'objectif d'un profit commercial ou d'une compensation financière personnelle, dans la mesure où aucun paiement ou compensation financière n'intervient en relation avec l'échange d'Oeuvres protégées.
c. Si l'Acceptant reproduit, distribue, représente ou communique l'Oeuvre au public, y compris par voie numérique, il doit conserver intactes toutes les informations sur le régime des droits et en attribuer la paternité à l'Auteur Original, de manière raisonnable au regard au médium ou au moyen utilisé. Il doit communiquer le nom de l'Auteur Original ou son éventuel pseudonyme s'il est indiqué ; le titre de l'Oeuvre Originale s'il est indiqué ; dans la mesure du possible, l'adresse Internet ou Identifiant Uniforme de Ressource (URI), s'il existe, spécifié par l'Offrant comme associé à l'Oeuvre, à moins que cette adresse ne renvoie pas aux informations légales (paternité et conditions d'utilisation de l'Oeuvre). Ces obligations d'attribution de paternité doivent être exécutées de manière raisonnable. Cependant, dans le cas d'une Oeuvre dite Collective, ces informations doivent, au minimum, apparaître à la place et de manière aussi visible que celles à laquelle apparaissent les informations de même nature.
d. Dans le cas où une utilisation de l'Oeuvre serait soumise à un régime légal de gestion collective obligatoire, l'Offrant se réserve le droit exclusif de collecter ces redevances par l'intermédiaire de la société de perception et de répartition des droits compétente. Sont notamment concernés la radiodiffusion et la communication dans un lieu public de phonogrammes publiés à des fins de commerce, certains cas de retransmission par câble et satellite, la copie privée d'Oeuvres fixées sur phonogrammes ou vidéogrammes, la reproduction par reprographie.

5. Garantie et exonération de responsabilité

a. En mettant l'Oeuvre à la disposition du public selon les termes de ce Contrat, l'Offrant déclare de bonne foi qu'à sa connaissance et dans les limites d'une enquête raisonnable :
    i. L'Offrant a obtenu tous les droits sur l'Oeuvre nécessaires pour pouvoir autoriser l'exercice des droits accordés par le présent Contrat, et permettre la jouissance paisible et l'exercice licite de ces droits, ceci sans que l'Acceptant n'ait aucune obligation de verser de rémunération ou tout autre paiement ou droits, dans la limite des mécanismes de gestion collective obligatoire applicables décrits à l'article 4(e);
b. L'Oeuvre n'est constitutive ni d'une violation des droits de tiers, notamment du droit de la propriété littéraire et artistique, du droit des marques, du droit de l'information, du droit civil ou de tout autre droit, ni de diffamation, de violation de la vie privée ou de tout autre préjudice délictuel à l'égard de toute tierce partie.
c. A l'exception des situations expressément mentionnées dans le présent Contrat ou dans un autre accord écrit, ou exigées par la loi applicable, l'Oeuvre est mise à disposition en l'état sans garantie d'aucune sorte, qu'elle soit expresse ou tacite, y compris à l'égard du contenu ou de l'exactitude de l'Oeuvre.

6. Limitation de responsabilité. A l'exception des garanties d'ordre public imposées par la loi applicable et des réparations imposées par le régime de la responsabilité vis-à-vis d'un tiers en raison de la violation des garanties prévues par l'article 5 du présent contrat, l'Offrant ne sera en aucun cas tenu responsable vis-à-vis de l'Acceptant, sur la base d'aucune théorie légale ni en raison d'aucun préjudice direct, indirect, matériel ou moral, résultant de l'exécution du présent Contrat ou de l'utilisation de l'Oeuvre, y compris dans l'hypothèse où l'Offrant avait connaissance de la possible existence d'un tel préjudice.

7. Résiliation

a. Tout manquement aux termes du contrat par l'Acceptant entraîne la résiliation automatique du Contrat et la fin des droits qui en découlent. Cependant, le contrat conserve ses effets envers les personnes physiques ou morales qui ont reçu de la part de l'Acceptant, en exécution du présent contrat, la mise à disposition d'Oeuvres dites Dérivées, ou d'Oeuvres dites Collectives, ceci tant qu'elles respectent pleinement leurs obligations. Les sections 1, 2, 5, 6 et 7 du contrat continuent à s'appliquer après la résiliation de celui-ci.
b. Dans les limites indiquées ci-dessus, le présent Contrat s'applique pendant toute la durée de protection de l'Oeuvre selon le droit applicable. Néanmoins, l'Offrant se réserve à tout moment le droit d'exploiter l'Oeuvre sous des conditions contractuelles différentes, ou d'en cesser la diffusion; cependant, le recours à cette option ne doit pas conduire à retirer les effets du présent Contrat (ou de tout contrat qui a été ou doit être accordé selon les termes de ce Contrat), et ce Contrat continuera à s'appliquer dans tous ses effets jusqu'à ce que sa résiliation intervienne dans les conditions décrites ci-dessus.

8. Divers

a. A chaque reproduction ou communication au public par voie numérique de l'Oeuvre ou d'une Oeuvre dite Collective par l'Acceptant, l'Offrant propose au bénéficiaire une offre de mise à disposition de l'Oeuvre dans des termes et conditions identiques à ceux accordés à la partie Acceptante dans le présent Contrat.
b. La nullité ou l'inapplicabilité d'une quelconque disposition de ce Contrat au regard de la loi applicable n'affecte pas celle des autres dispositions qui resteront pleinement valides et applicables. Sans action additionnelle par les parties à cet accord, lesdites dispositions devront être interprétées dans la mesure minimum nécessaire à leur validité et leur applicabilité.
c. Aucune limite, renonciation ou modification des termes ou dispositions du présent Contrat ne pourra être acceptée sans le consentement écrit et signé de la partie compétente.
d. Ce Contrat constitue le seul accord entre les parties à propos de l'Oeuvre mise ici à disposition. Il n'existe aucun élément annexe, accord supplémentaire ou mandat portant sur cette Oeuvre en dehors des éléments mentionnés ici. L'Offrant ne sera tenu par aucune disposition supplémentaire qui pourrait apparaître dans une quelconque communication en provenance de l'Acceptant. Ce Contrat ne peut être modifié sans l'accord mutuel écrit de l'Offrant et de l'Acceptant.
e. Le droit applicable est le droit français.

Creative Commons n'est pas partie à ce Contrat et n'offre aucune forme de garantie relative à l'Oeuvre. Creative Commons décline toute responsabilité à l'égard de l'Acceptant ou de toute autre partie, quel que soit le fondement légal de cette responsabilité et quel que soit le préjudice subi, direct, indirect, matériel ou moral, qui surviendrait en rapport avec le présent Contrat. Cependant, si Creative Commons s'est expressément identifié comme Offrant pour mettre une Oeuvre à disposition selon les termes de ce Contrat, Creative Commons jouira de tous les droits et obligations d'un Offrant.

A l'exception des fins limitées à informer le public que l'Oeuvre est mise à disposition sous CPCC, aucune des parties n'utilisera la marque « Creative Commons » ou toute autre indication ou logo afférent sans le consentement préalable écrit de Creative Commons. Toute utilisation autorisée devra être effectuée en conformité avec les lignes directrices de Creative Commons à jour au moment de l'utilisation, telles qu'elles sont disponibles sur son site Internet ou sur simple demande.

Creative Commons peut être contacté à http://creativecommons.org/.