

Exercice 0.1 Ouroboros

1 - Implémentez l'algorithme de snake décrit ci-dessus

Voir fichiers Ex2TD7.m, DerivGauss.m et innerDerivative.m.

2 - Testez l'algorithme

J'ai testé l'algorithme sur plusieurs objets de plusieurs images différentes, en particulier sur l'image motifs.jpg et aussi un peu avec montage.jpg. Voir la question suivante pour la discussion sur les performances de l'algorithme.

Le début du code du fichier Ex2TD7.m permet de charger les différents ensembles de points (initialisations) correspondant aux différents tests. Il suffit de décommenter les différents blocs de code pour visualiser les résultats obtenus avec un objet d'une image. Je laisse par défaut la carré central de l'image motifs.jpg avec 100 points.

3 - Faites un commentaire (ce qui marche, ce qui ne marche pas, les difficultés d'initialisation et de paramétrage) de l'algorithme.

J'ai utilisé pour cette section deux objets de l'image motifs.jpg pour essayer de tester différentes configurations et différents comportements de l'algorithme. Le choix des paramètres est assez difficile et je me suis basé sur les valeurs données par le cours pour étudier un à un chaque paramètre et son influence sur la performance de l'algorithme. Les conclusions que je propose ici ne sont pas absolues et correspondent aux observations que j'ai faites sur ces exemples limités.

Influence des paramètres

On regarde l'influence de la variation de chaque paramètre sur le résultat obtenu avec le carré au centre de motifs.jpg (voir figure 1).

Pour ce qui est des paramètres SIGMA=4, $\lambda = 1$, $\varepsilon = 0.1$ et max_iter=1000, on garde ces valeurs tout au long des tests car SIGMA semble vraiment bien choisi, et jouer sur λ ou ε revient à peu près au même (au fond, ça n'est que le nombre d'itérations qui change, même si c'est plus subtil que cela en réalité), sauf si le pas d'apprentissage est trop grand ; un pas d'apprentissage de 1 correspond à la limite supérieure que l'on peut raisonnablement mettre (avec l'expérience que j'en ai en tout cas ; c'est-à-dire d'après l'utilisation que j'en ai fait pour l'apprentissage de réseaux de neurones. Je ne sais pas si cela vaut ici aussi).

Sur la figure 1, on voit que :

- une valeur de α "trop grande" amène à des tracés complètement absurdes
- une valeur de α "trop petite" ne change en fait pas grand chose pour ce cas-là où il y a peu d'angles (ce ne serait pas le même constat pour le parapluie par exemple)

Le paramètre β a apparemment le même effet, dans cet exemple en tout cas.

Pour le paramètre γ (figure 2), qui correspond à "l'importance" qu'on accorde au gradient externe de l'image, une valeur trop élevée induit un comportement similaire à β ou α , en revanche on observe une différence pour γ "trop petit" ; les angles sont abordés de la même manière mais les zones rectilignes (les côtés du rectangle) n'évoluent que très peu. Cela est peut-être dû à la proximité d'autres rectangles sur les côtés gauche et droit, mais cela n'explique pas pourquoi par exemple le côté Sud du rectangle n'évolue que très peu (puisque'il n'y a aucune autre zone grise "en face" du côté Sud). Peut-être que le tracé initial est un peu loin de l'objet que l'on souhaite segmenter. On verra plus loin (6) qu'augmenter le nombre de points peut aider à résoudre ce problème.

Pour HSIZE (figure 3), des valeurs trop petites amènent à considérer une trop petite fenêtre qui fait que l'on ne détecte pas des zones intéressantes qui ne sont pas immédiatement voisines, et donc on voit que certains bords n'évoluent pas trop (comme pour γ). Des valeurs trop grandes produisent l'effet

inverse, on voit par exemple en haut à gauche de la figure correspondante que le bord détecté est le bord intérieur du rectangle et pas le bord extérieur, ce qui doit être lié à la taille de la fenêtre (toutes choses étant égales par ailleurs).

Rôle de l'initialisation

On regarde l'influence de l'initialisation avec le parapluie de `motifs.jpg`.

On observe sur la figure 4 que l'initialisation de la courbe joue un rôle important. En effet, toutes choses égales par ailleurs, le détourage est moins précis si la courbe est initialisée loin de l'objet étudié. Pourtant, l'initialisation de la figure (a) n'est pas non plus si éloignée que ça...

On remarque soit dit en passant que le résultat affiché figure (b) n'est pas non plus extrêmement satisfaisant ; des angles bien définis à l'origine sont arrondis par le snake. Cela est peut-être dû à SIGMA qui "aplatit" le gradient (convolution avec une gaussienne). Diminuer ou augmenter SIGMA ne corrige *pas toujours* ce comportement (figure 5 : augmenter SIGMA peut aider avec les angles dans certains cas).

Influence du nombre de points définissant le contour

On reprend le rectangle. Prenons deux exemples radicalement opposés : un exemple avec 4 points, et un exemple avec plusieurs milliers de points définis avec `linspace`.

Clairement, comme on le voit figure 6 plus on a de points, mieux l'algorithme se comporte. On a même un gain visible de performance entre le pointage à la main (environ 100 points) et les 4000 points définis par `linspace`.

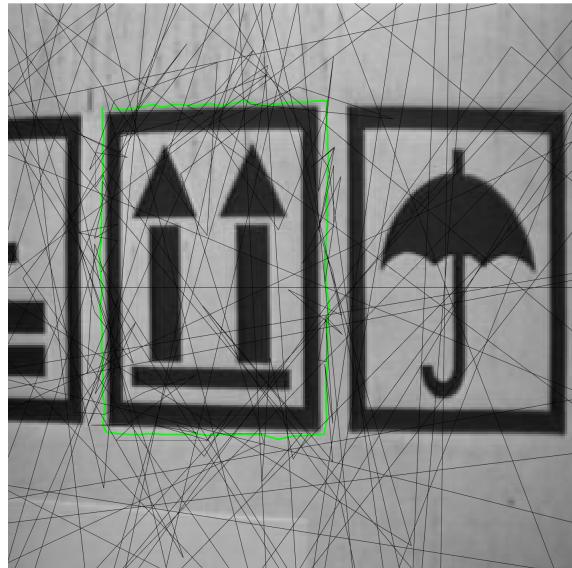
Commentaires et conclusion

On parvient finalement à des résultats qui sont plutôt bons même si parfois lacunaires (des bords mal détectés comme on l'a vu avec le premier rectangle 3 exemple, ou encore des angles difficiles à bien respecter dans le rectangle ou le parapluie figure 4). De manière générale, le *snake* gère un peu moins bien les angles (on le voit par exemple sur le parapluie même si on finit par arriver à quelque chose de relativement satisfaisant - en manipulant des paramètres comme SIGMA 5). Les zones concaves des objets posent aussi parfois problème (figure 4).

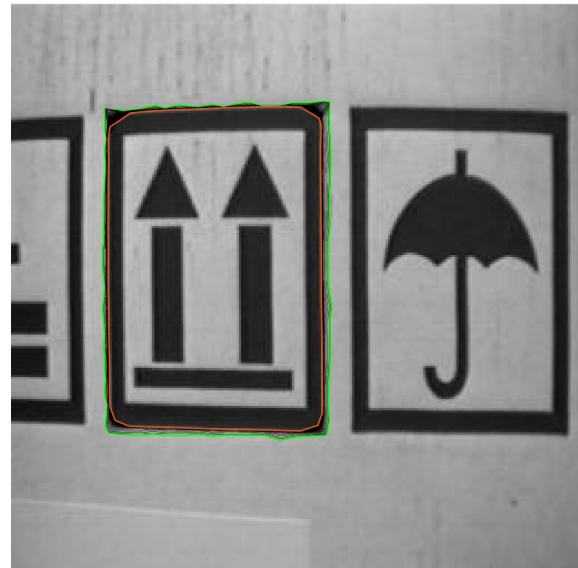
Il y a deux problèmes plus fondamentaux qui se manifestent au cours de ces tests :

- premièrement, l'algorithme semble assez **sensible à l'initialisation du contour**, ce qui est un vrai problème : c'est littéralement le serpent qui se mord la queue (figure 4) ! Et, surtout, il faut une autre méthode pour initialiser ce contour de manière suffisamment satisfaisante pour que le *snake* fonctionne correctement, par exemple.
- aussi, il est difficile de décider des **paramètres optimaux** (donnant le meilleur résultat possible) pour exécuter l'algorithme, d'autant plus que ces paramètres optimaux semblent varier d'une image à une autre, ou même d'un objet à un autre... Leurs influences se recoupent et il est difficile de déterminer exactement des critères objectifs pour choisir la valeur de paramètres comme α , β , γ ou encore `HSIZE` (figure 6).

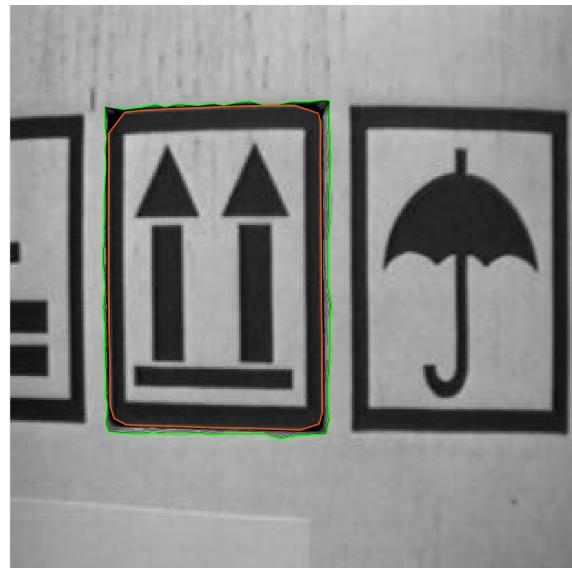
Ces complications expliquent que l'on ait cherché à améliorer cet algorithme : GVF, diffusion snakes, Geometric Active Contours évoqués par le cours et précisés par exemple sur la page [Wikipédia](#).



(a) Paramètres de l'énoncé, sauf $\alpha = 10$. 3 itérations de descente du gradient.

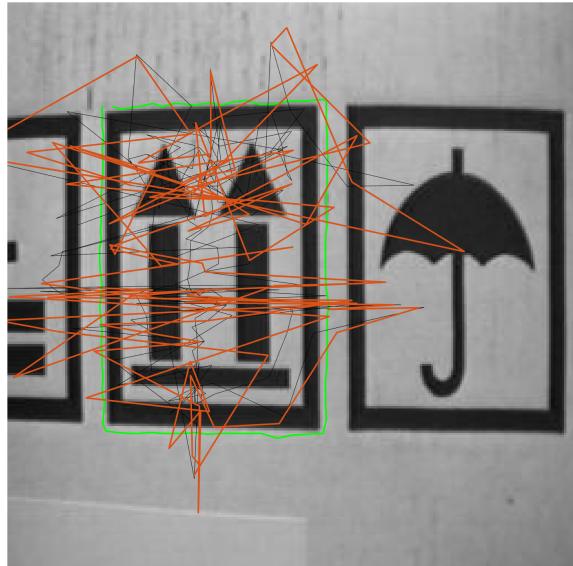


(b) Paramètres conseillés par l'énoncé. 313 itérations pour avoir une amélioration $< \varepsilon := 0.1$.

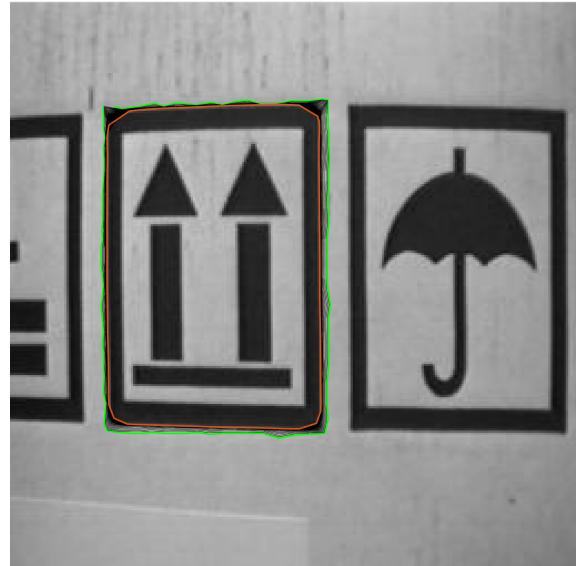


(c) Paramètres de l'énoncé, sauf $\alpha = 0.000001$. 148 itérations de descente du gradient

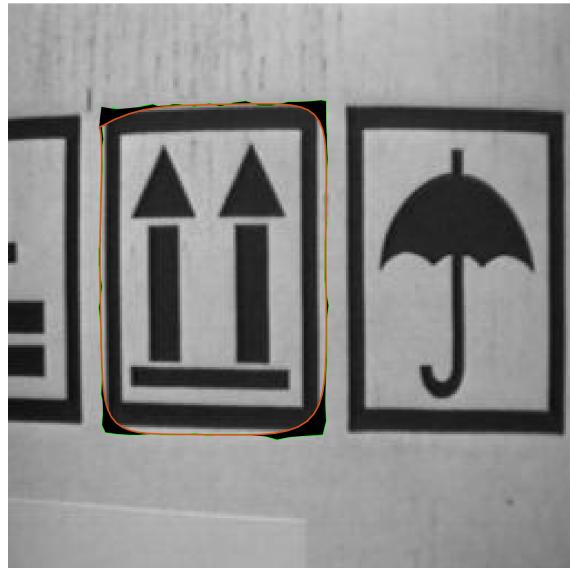
Figure 1 – Effets du paramètre α sur le *snake* obtenu. En vert : la courbe en entrée ; en rouge : la courbe obtenue par descente de gradient.



(a) Paramètres de l'énoncé, sauf $\gamma = 10$.

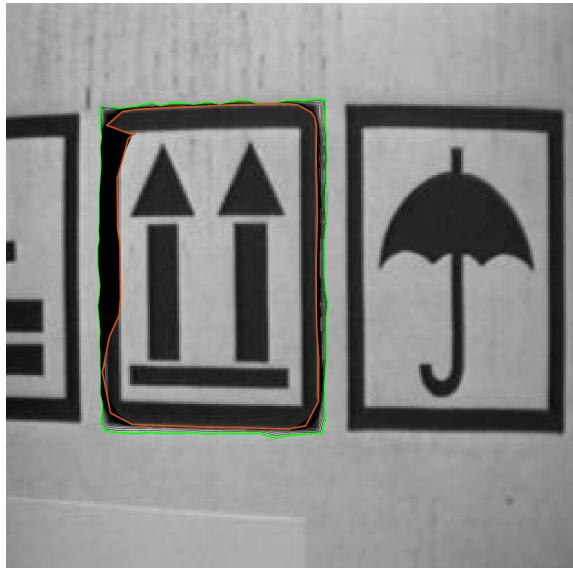


(b) Paramètres conseillés par l'énoncé. 313 itérations pour avoir une amélioration $< \varepsilon := 0.1$.

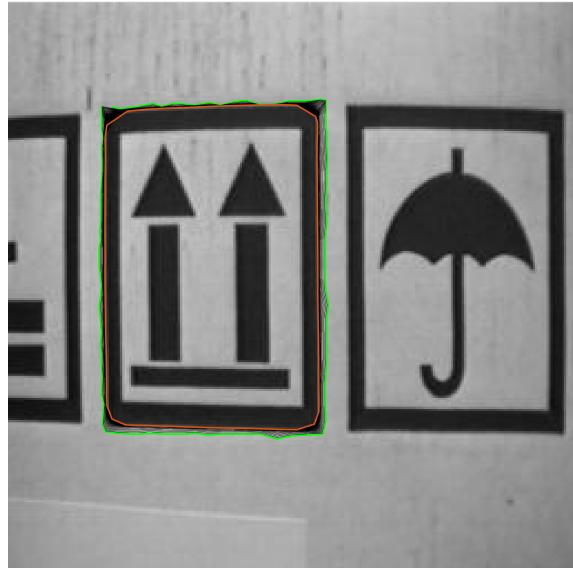


(c) Paramètres de l'énoncé, sauf $\gamma = 0.0000001$. 378 itérations de descente du gradient

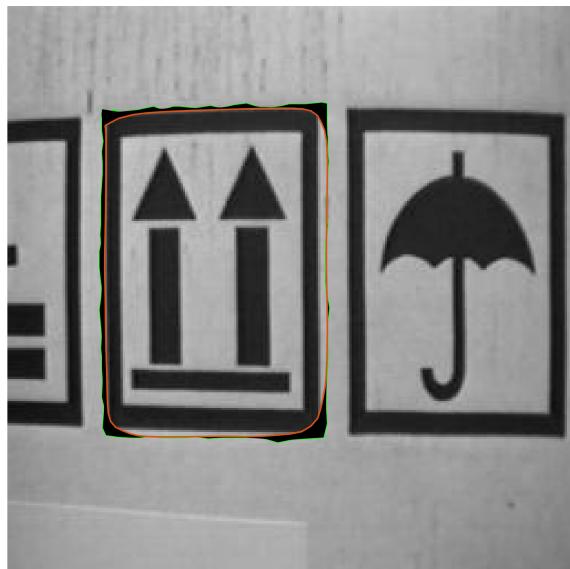
Figure 2 – Effets du paramètre γ sur le *snake* obtenu. En vert : la courbe en entrée ; en rouge : la courbe obtenue par descente de gradient.



(a) Paramètres de l'énoncé, sauf $HSIZE = 20$. 1000 itérations de descente du gradient (maximum atteint).



(b) Paramètres conseillés par l'énoncé. 313 itérations pour avoir une amélioration $< \varepsilon := 0.1$.

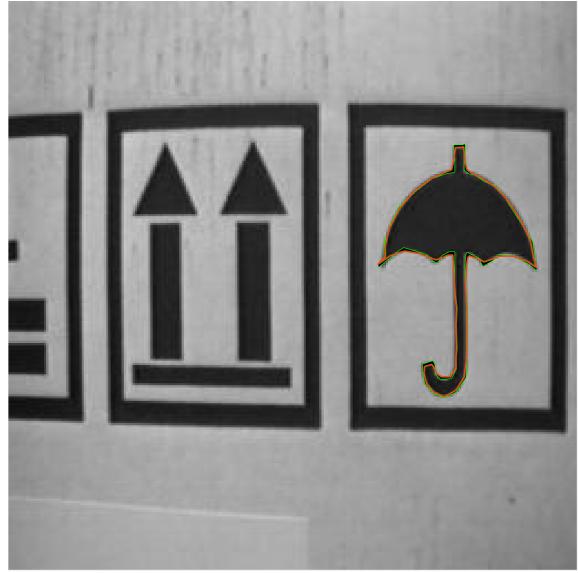


(c) Paramètres de l'énoncé, sauf $HSIZE = 1$. 252 itérations de descente du gradient

Figure 3 – Effets du paramètre $HSIZE$ sur le *snake* obtenu. En vert : la courbe en entrée ; en rouge : la courbe obtenue par descente de gradient.

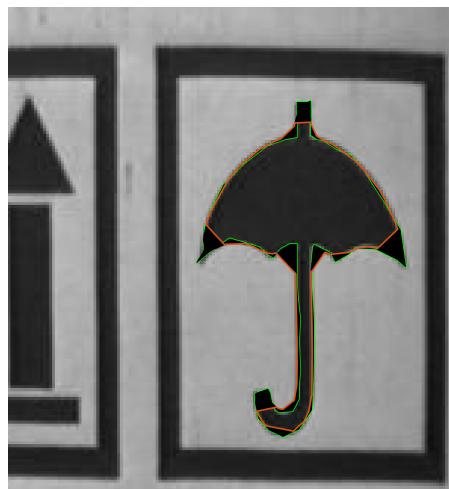


(a) Points initialisés un peu loin de l'objet étudié. les zones concaves posent certains problèmes, comme vu en cours.



(b) Points initialisés plus proche.

Figure 4 – Effets de l'initialisation de la courbe sur le résultat final. $\alpha = 0.0001$; $\beta = 0.00001$; $\gamma = 0.001$; $SIZE = 5$; $SIGMA=4$; $\lambda = 1$; $\epsilon = 0.01$; $max_iter=1000$.



(a) Points initialisés proche de l'objet étudié. $SIGMA=2$

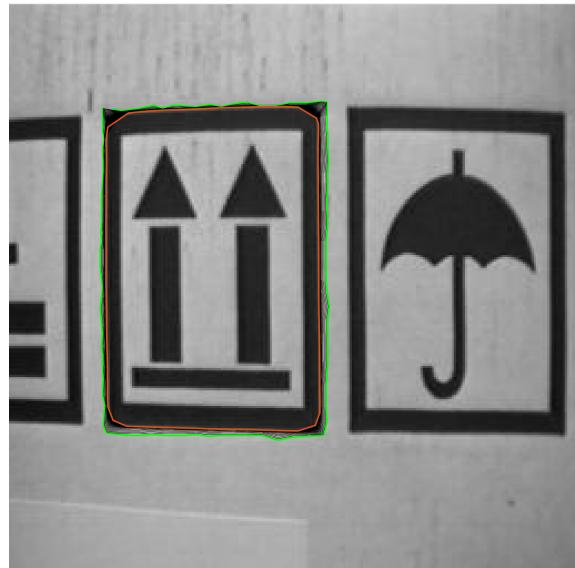


(b) Points initialisés proche de l'objet. $SIGMA=8$

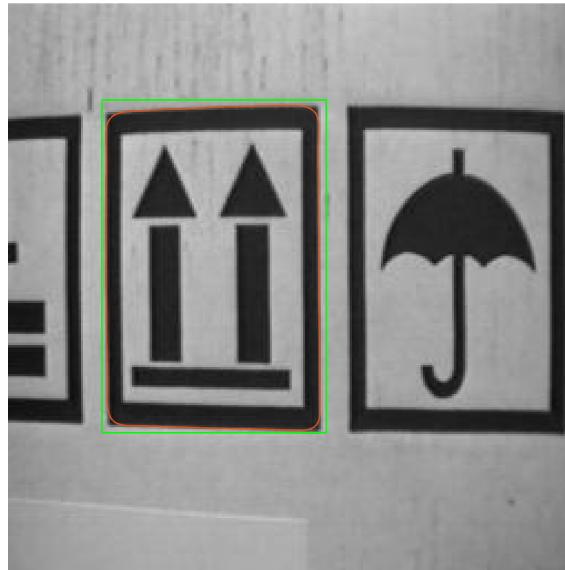
Figure 5 – Mêmes paramètres, sauf $SIGMA$ qui varie. Si $SIGMA=2$, l'effet d'aplatissement des angles est très marqué (logique car on diminue encore l'étalement du gradient). Si l'on augmente sigma, le contour n'est que très peu modifié par rapport à l'original, même si l'on augmente l'amélioration minimale pour continuer la descente de gradient ($\epsilon = 0.001$ au lieu de 0.1). Mais même en changeant epsilon, le contour évolue très peu (peut-être car il est "trop" bien défini à l'origine). Cependant les angles sont bel et bien moins lissés que dans la figure 4 (b).



(a) *Snake* initialisé avec 4 points. Le résultat final est absurde.



(b) Initialisation avec 100 points. Le résultat est meilleur mais quelques difficultés sur les angles.



(c) Initialisation avec 4000 points. Les angles posent toujours problème mais ils sont tout de même mieux respectés. On remarque aussi, contrairement à ce qu'on a pu dire à propos de l'initialisation, figure 2 que les bords mêmes lointains sont bien détectés.

Figure 6 – Influence du nombre de points utilisés pour initialiser la courbe sur les performances de l'algorithme de *snake*.