Global Minimum for Active Contour Models: A Minimal Path Approach

JACOB Amaury

29 octobre 2025

Table des matières

1 Introduction **1**

2 Comment calculer un contour avec un chemin minimal **1**
2.1 Calcule du potentiel P 1
2.2 Calcul de la métrique $\tilde{P}$ 2
2.3 Equation Eikonal 2
2.4 Back-propagation du chemin minimal 2

3 Programmation **3**
3.1 Prétraitement de l'image 3
3.2 Calcul de $\tilde{P}$ 3
3.3 Fast Marching 3
3.3.1 Initialisation 3
3.3.2 Boucle principale 3
3.3.3 Solve eikonal 3
3.4 Fonction `backtrack` 4

4 Résultats **4**

1 Introduction

Le but de ce devoir est de comprendre, reproduire et critiquer un article. Mon article porte sur les contours actifs par une approche du chemin minimal. La méthode présentée dans cet article est une amélioration de la méthode des Snakes ; cependant, cette méthode ne nécessite que deux points comme initialisation. Elle a aussi pour avantage d'être moins sensible aux minima locaux et de donner le chemin minimal entre les deux points.

Dans un premier temps, nous allons voir les bases mathématiques nécessaires pour comprendre l'article.

2 Comment calculer un contour avec un chemin minimal

2.1 Calcule du potentiel P

Pour calculer le contour à l'aide du chemin minimal, il faut d'abord construire une carte de potentiel. Cette carte repose sur le gradient de l'image et sur la norme du gradient.

$$\nabla\mathbf{I} = \begin{bmatrix} I'_x \\ I'_y \end{bmatrix} \quad \text{et} \quad \|\nabla I\| = \sqrt{(I'_x)^2 + (I'_y)^2}. \tag{1}$$

Maintenant que nous avons calculé la norme du gradient, nous pouvons déterminer le potentiel P. Le potentiel $P(x, y)$ agit comme une mesure de "résistance" au passage du contour. Ainsi, plus le gradient est fort (bord marqué), plus le potentiel est faible. Le chemin minimal cherchera donc naturellement à suivre les bords de l'image.

Dans l'article de référence, aucune expression explicite n'est donnée pour P. J'ai donc cherché dans d'autres sources des définitions possibles du potentiel, en particulier dans deux travaux où les formules sont simples et peu coûteuses à calculer d'un point de vue numérique :

$$\begin{aligned} P_1(x,y) &= e^{-\alpha\|\nabla I(x,y)\|^2}, \\ P_2(x,y) &= \frac{1}{1+\alpha\|\nabla I(x,y)\|^2}. \end{aligned} \tag{2}$$

Ces deux définitions apparaissent respectivement dans les articles de Caselles *et al.*, *"Geodesic Active Contours"* (1997), et Sethian & Kimmel, *"Computing Geodesic Paths"* (1998).

C'est deux expression sont similaire mais une décroit de facons exponentiel alors que l'autre décroit en $\frac{1}{x}$ ce qui est plus lent. Le choix va dépendre de l'image.

2.2 Calcul de la métrique $\tilde{P}$

Une fois la carte de potentiel $P(x,y)$ calculée, elle est utilisée pour définir une métrique de longueur $\tilde{P}$.

$$\tilde{P} = \omega + P(p) \tag{3}$$

ω est un terme qui représente le lissage de la courbe et p est le point actuel. Cette métrique joue le role de vitesse dans l'équation Eikonal que nous allons voir.

2.3 Equation Eikonal

L'équation Eikonal relie $\tilde{P}(x,y)$ et une fonction de coût $U(x,y)$. Cette fonction représente en chaque point de l'image le coût pour aller du point de départ a un point (x,y) de l'image.

$$\|\nabla U(x,y)\| = \tilde{P}(x,y), \tag{4}$$

avec la condition initiale $U(p_0) = 0$.

Dans l'article de Cohen et Kimmel, cette équation est résolue numériquement par la méthode du *Fast Marching*. Cette méthode repose sur une propagation du front de niveau de U : le front avance plus rapidement dans les zones où $\tilde{P}$ est faible (autour des bords) et plus lentement dans les zones où $\tilde{P}$ est élevée.

Mathématiquement, l'espace est discrétisé en une grille, et pour chaque pixel (i,j), on cherche la valeur de $U_{i,j}$ qui satisfait la version discrète de l'équation :

$$(\max\{U - U_{i-1,j}, U - U_{i+1,j}, 0\})^2 + (\max\{U - U_{i,j-1}, U - U_{i,j+1}, 0\})^2 = \tilde{P}_{i,j}^2. \tag{5}$$

Cette équation quadratique est résolue localement à chaque itération en utilisant uniquement les voisins déjà mis à jour.

2.4 Back-propagation du chemin minimal

Une fois la fonction $U(x,y)$ calculée sur toute l'image, on peut retrouver le contour recherché en suivant la direction opposée au gradient de U. Cette étape, appelée *back-propagation*, correspond à la descente du long de la pente de U depuis le point d'arrivée p_1 jusqu'au point de départ p_0. Le chemin obtenu correspond au chemin minimal reliant ces deux points dans la métrique $\tilde{P}$.

Mathématiquement, la courbe $C(s)$ vérifie :

$$\frac{dC(s)}{ds} = -\nabla U(C(s)), \quad C(0) = p_1, \tag{6}$$

Quand on intégre l'expression (6) nous obtenons le chemin qui suit la pente la plus forte

Dans l'article les auteurs expliquent que cela revient à une descente de gradient La méthode mache de facons suivante : On part de p_1, et à chaque itération on choisit le voisin possédant la plus petite valeur de U, jusqu'à rejoindre p_0. Cette procédure simple permet de reconstruire efficacement le contour global optimal après la propagation du front.

3 Programmation

3.1 Prétraitement de l'image

Le prétraitement est une étape clé pour obtenir des résultats cohérents. Cette étape est la plus simple du code. Dans un premier temps, je convertis mes images en niveaux de gris avec la fonction `open_image`. Puis, dans un second temps, j'applique un flou gaussien avec un noyau de taille 5×5. Enfin, je normalise l'image en divisant chaque pixel par 255 afin d'obtenir des valeurs de gris comprises entre $[0, 1]$.

3.2 Calcul de $\tilde{P}$

Comme déjà vu dans la section calcul de la métrique, j'ai commencé par calculer le gradient de l'image avec la fonction `np.gradient`, puis j'ai calculé la norme du gradient. Pour le calcul de P, je peux choisir entre deux modes correspondant aux deux expressions de P. Le choix du mode et des paramètres de la fonction dépend de l'image.

3.3 Fast Marching

Cette partie du code est la plus importante et aussi la plus longue en termes de calcul. Le Fast Marching va permettre de déterminer U (la carte de distance).

3.3.1 Initialisation

La carte de distance est de même dimension que l'image et, au départ, toutes les valeurs de U sont initialisées à l'infini, sauf le point de départ qui est initialisé à 0. Au même moment, j'initialise un autre tableau de même dimension que l'image nommé `labels`. `Labels` est constitué de trois valeurs décrites dans l'article : *Far*, *Trial* et *Alive*. Toutes les valeurs de `labels` sont initialisées à *Far*, sauf le point p_0 qui est à *Trial*. Il ne reste que la liste des points à traiter (`heaps`), initialisée avec le tuple des coordonnées de p_0. Dans l'article, les auteurs précisent qu'il est préférable d'utiliser une structure de type *min-heap*. En Python, il existe le module `heapq` pour cela, mais n'étant pas familier avec cette syntaxe, je n'ai pas réussi à faire une version de mon code opérationnelle avec.

3.3.2 Boucle principale

On va itérer jusqu'à ce que `heap` soit vide. À chaque itération, on va trouver la valeur de distance minimale ($\min(U)$) des points stockés dans `heap` et sauvegarder les coordonnées et la distance minimale. Puis nous retirons les coordonnées de ce pixel de `heap`, et le label de ce pixel passe à *Alive*.

Finalement, je regarde les voisins de ce pixel avec une connexité de 4. Si ces pixels ne sont pas à *Alive*, je calcule leur nouvelle distance à l'aide de la fonction `solve_eikonal` (qui sera détaillée dans la section suivante). Si cette distance est inférieure à la distance de la carte de distance, elle devient la nouvelle distance, et si le label de ce pixel est à *Far*, je l'ajoute à `heap` et son label passe à *Trial*.

3.3.3 Solve eikonal

Dans un premier temps, je récupère la valeur de $\tilde{P}$ au pixel courant. Ensuite, je regarde les voisins gauche et droite pour calculer la plus petite distance a, et les voisins haut et bas pour la plus petite distance b. Ces valeurs a et b représentent la distance minimale connue autour du pixel dans les deux directions.

Ensuite, je vérifie la différence entre a et b. Si le front arrive des deux directions, je résous l'équation quadratique issue de la discrétisation de l'équation d'Eikonal :

$$U_{new} = \frac{a + b + \sqrt{2\tilde{P}^2 - (a - b)^2}}{2}.$$

Enfin, la fonction retourne cette nouvelle valeur U_new,

3.4 Fonction `backtrack`

Cette fonction permet de retrouver le chemin minimal une fois la carte de distance U calculée. Pour cela, je calcule le gradient local de U autour du point courant, puis je me déplace dans la direction opposée au gradient normalisé. Le pas de déplacement est défini par la variable `step`. À chaque itération, je sauvegarde les nouvelles coordonnées dans la liste `paths`. Cette boucle se répète jusqu'à atteindre le point de départ ou jusqu'à atteindre le nombre maximal d'itérations `max_it`. Le résultat est donc une suite de points représentant le chemin minimal entre les deux points.

4 Résultats