ÉTUDE DE L'ABRASION DE LA PEAU EN ESCALADE

Epreuve de TIPE
Session 2026

AUDOUZE Aloïs N°50886

SOMMAIRE

SUJET ET ENCRAGE AU THÈME :
INTRODUCTION

HISTOIRE DE L'ESCALADE

Mont Etna

Empédocle V siècle AV-JC

L'empereur Hadrien II siècle

source :https://caen.climb-up.fr/

source:https://arkose.com/

SUJET ET ENCRAGE AU THÈME :
INTRODUCTION


source : atomikclimbingholds.com


source : aporteededoigts.com

SUJET ET ENCRAGE AU THÈME :
INTRODUCTION

source : magnus mitbo ,
Secret training method of the World's strongest climber - Alex Megos

ALEXANDER MEGOS palmarès :

9A à vue

+ de 80 voies cotées dans le 9ème degré

2^{e} en difficulté aux Championnats du monde en 2019

PROBLÉMATIQUE

COMMENT MESURER ET LIMITER LA PERTE DE PEAU LORS DE FROTTEMENTS RÉPÉTÉS EN ESCALADE ?

OBJECTIFS

- **Mesurer les différents** coefficients de frottements entre la peau avec un matériau et une prise d'escalade
- **Déterminer une façon de mesurer** l'abrasion de la peau au fil de cycles de frottements
- **En déduire une façon de grimper** qui protège le plus possible la peau pour optimiser la séance

OBJECTIFS

ÉTUDE DES COEFFICIENTS DE FROTTEMENTS

MÉTHODE EXPÉRIMENTALE N°1

MÉTHODE D'ÉTUDE DU COEFFICIENT DE FROTTEMENT

Objectif: Mesurer les coefficients de frottements entre la peau (avec ou non des substances) et la prise d'escalade

Main à vide

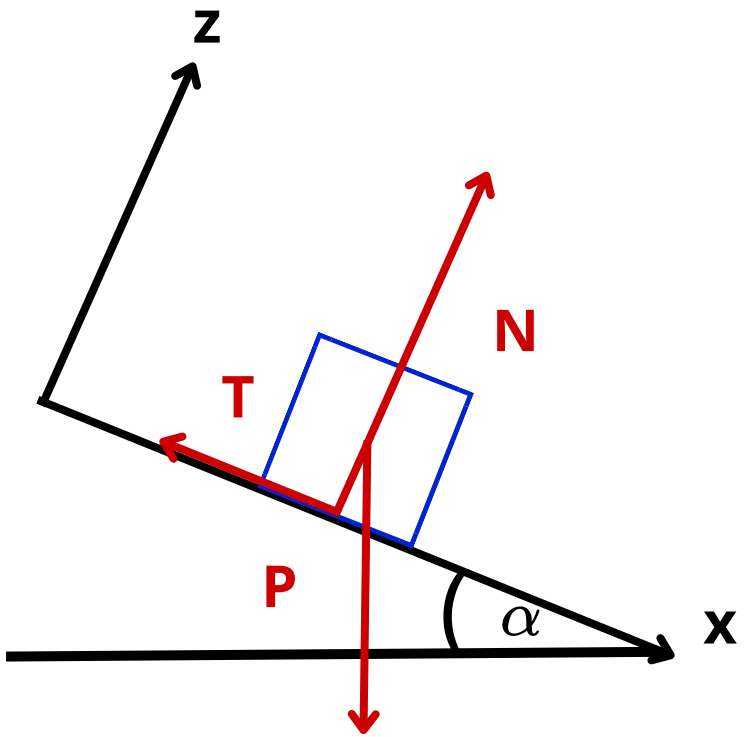
Main avec magnésie

Main avec tape

MÉTHODE EXPÉRIMENTALE N°1

MÉTHODE D'ÉTUDE DU COEFFICIENT DE FROTTEMENT

Schéma :



P : le poids de la prise

$$P = mg = mg(\sin\alpha \cdot u_x - \cos\alpha \cdot u_z)$$

N : la réaction normale du support

$$N = N \cdot u_z$$

T : la réaction tangentielle du support

$$T = -T \cdot u_x$$

MÉTHODE EXPÉRIMENTALE N°1

MÉTHODE D'ÉTUDE DU COEFFICIENT DE FROTTEMENT

Principe fondamental de la dynamique

$$\begin{aligned} u_n : \quad & -mg\cos\alpha + N = 0 \\ u_z : \quad & mg\sin\alpha - T = 0 \end{aligned} \quad \Longrightarrow \quad \begin{cases} N = mg\cos\alpha \\ T = mg\sin\alpha \end{cases}$$

Condition de non-glissement (Loi de Coulomb statique)

$$\|T\| \leq f_s \|N\|$$

On obtient ainsi une **relation sur le coefficient de frottement:**

$$f_s = \tan(\alpha_{\lim})$$

EXPÉRIENCE N°1

MÉTHODE D'ÉTUDE DU COEFFICIENT DE FROTTEMENT

Limite d'adhérence

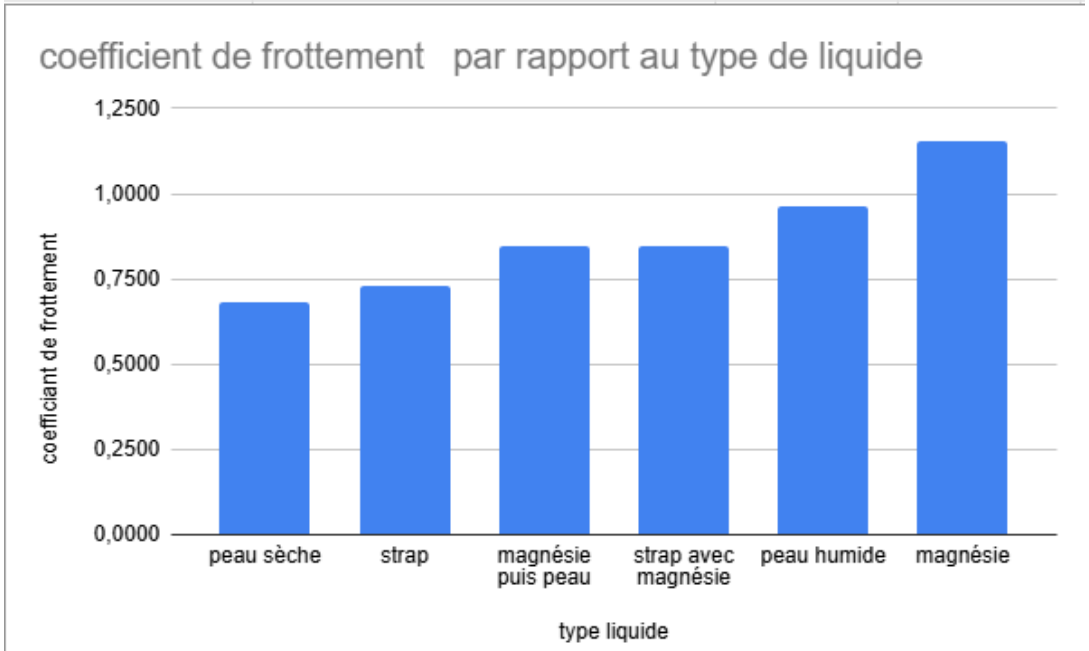
Angle limite

perte d'adhérence

EXPÉRIENCE N°1

RÉSULTATS

type liquide	coefficient de frottement	incertitude
peau sèche	0,6801	0,0008
strap	0,7283	0,0009
magnésie puis peau	0,8440	0,001
strap avec magnésie	0,8450	0,001
peau humide	0,9620	0,0011
magnésie	1,1535	0,0013



EXPÉRIENCE N°1

CONCLUSION

- La **magnésie** se présente comme le meilleur matériau pour **améliorer la friction entre la prise et la main**.

- **l'humidité de la main aide en théorie** en réalité cela rend les prises extrèmement glissantes.

- se protéger les doigts avec du **strap** est utile pour protéger la peau mais **fait perdre nettement en adhérence**.

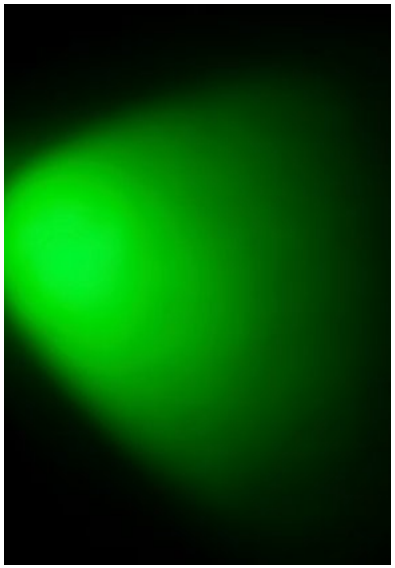
MÉTHODE EXPÉRIMENTALE N°2

MÉTHODE EXPÉRIMENTALE N°2

MÉTHODE EXPÉRIMENTALE N°2

PREMIÈRE IDÉE DE L'ÉTUDE DE L'ABRASION DE LA PEAU

Expérience avec une variation de l'intensité lumineuse



source :freepik.com

source: gotronic.com

MÉTHODE EXPÉRIMENTALE N°2

PROBLÈME DE CETTE MÉTHODE

L'épaisseur de l'épiderme est trop faible pour obtenir une difference notable de l'intensité lumineuse.

épaisseur du doigt ≈ 1 cm
épaisseur de l'épiderme ≈ 0,5 à 1,5 mm

Problèmes liés à l'hydratation de la peau , la température et de la variation de position du doigt = trop de paramètres à contrôler.

MÉTHODE EXPÉRIMENTALE N°2

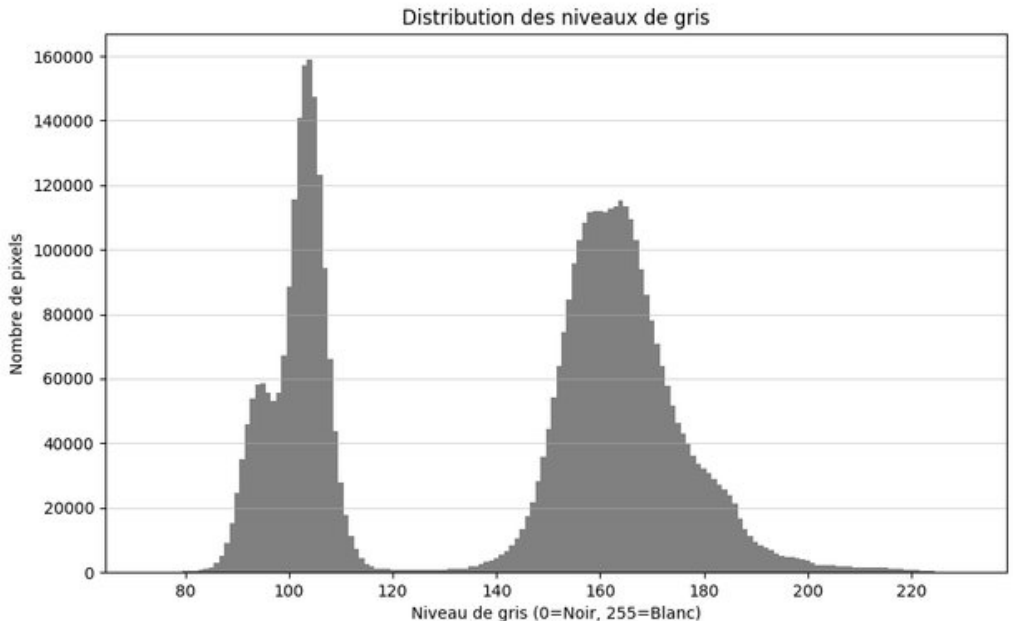
MÉTHODE DES NIVEAUX DE GRIS

La technique des niveaux de gris consiste à **représenter une image par des nuances de luminosité**, allant du noir pur au blanc pur, sans aucune information de couleur.

MÉTHODE EXPÉRIMENTALE N°2

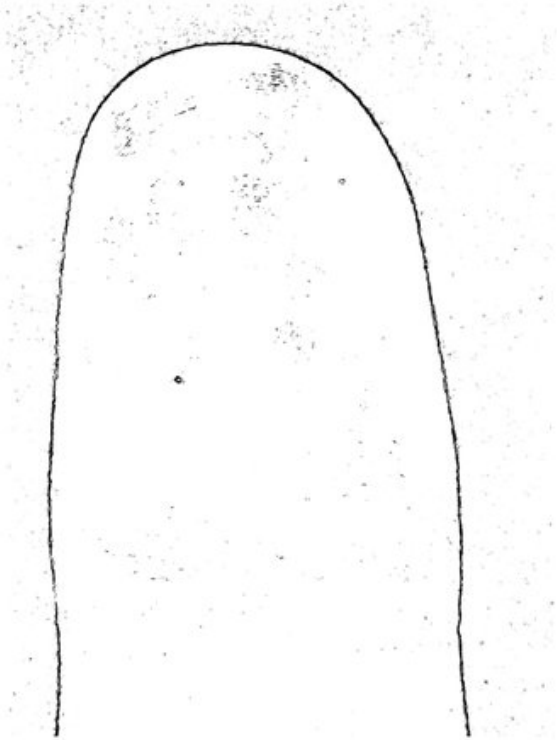
MÉTHODE DES NIVEAUX DE GRIS

En comptant le nombre de pixels en fonction de leurs valeur de gris comprise entre **0** et **255**, on obtient un histogramme de l'image.



MÉTHODE EXPÉRIMENTALE N°2

On isole l'arrière plan en appliquant un gradient.
On sélectionne ensuite les pixels dont le **gradient est supérieur à un seuil** fixé pour obtenir les bords de l'image.

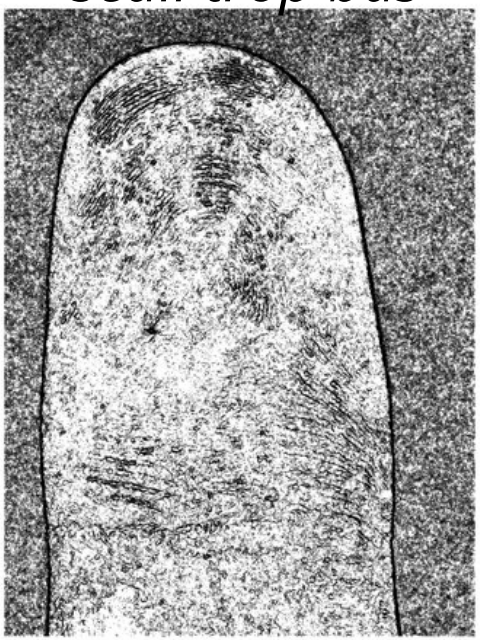


seuil = 1500

MÉTHODE EXPÉRIMENTALE N°2

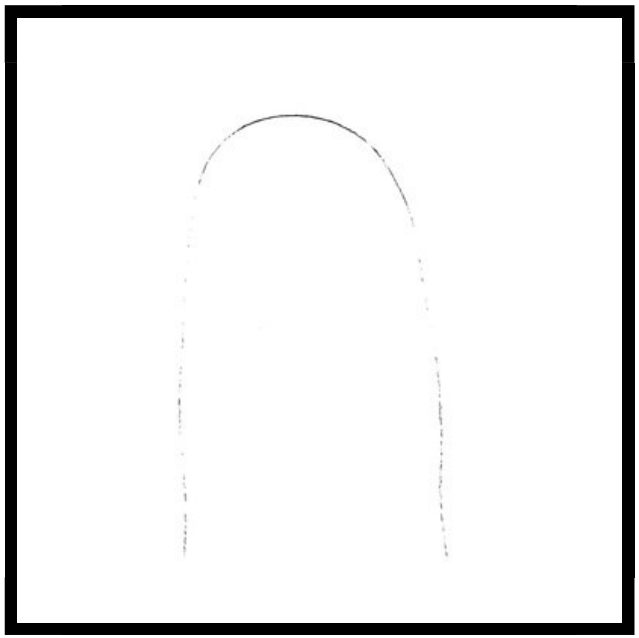
Un problème apparait :

seuil trop bas



seuil = 100

seuil trop haut



seuil = 8000

MÉTHODE EXPÉRIMENTALE N°2

MISE EN PLACE D'UNE MÉTHODE D'ISOLEMENT DU DOIGT

MÉTHODE EXPÉRIMENTALE N°2

Pour 2 voisins et un seuil de 5:

Image apres moyenne glissante

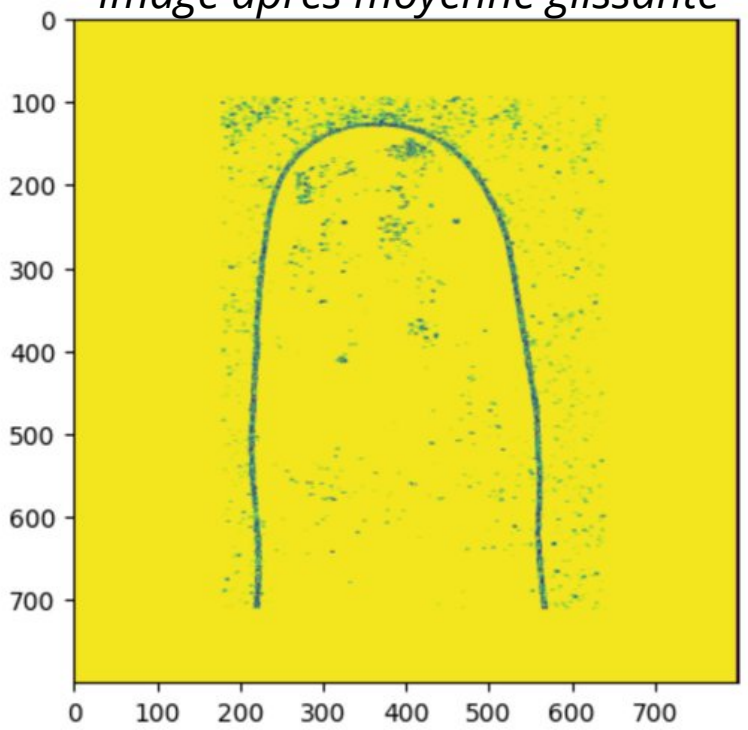
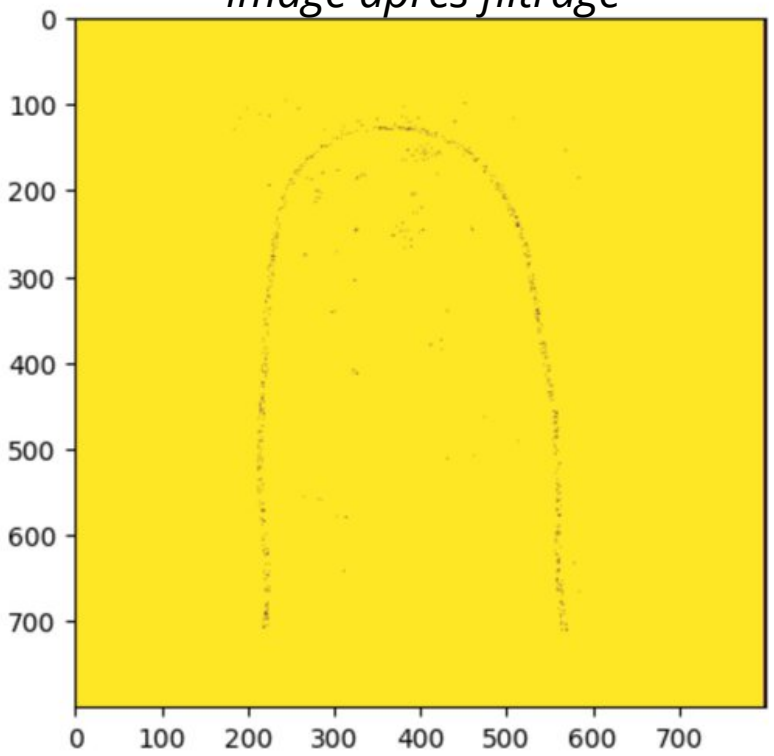


Image apres filtrage



MÉTHODE EXPÉRIMENTALE N°2

PROBLÈMES DE CETTE MÉTHODE

- **Ajout de seuils** pour en determiner un (nombre de voisin, seuil de moyennage).

- Réduction de bruit mais aussi **réduction de la qualité de l'image** ce qui rend difficile de détecter les bords (discontinuités et irrégularités).

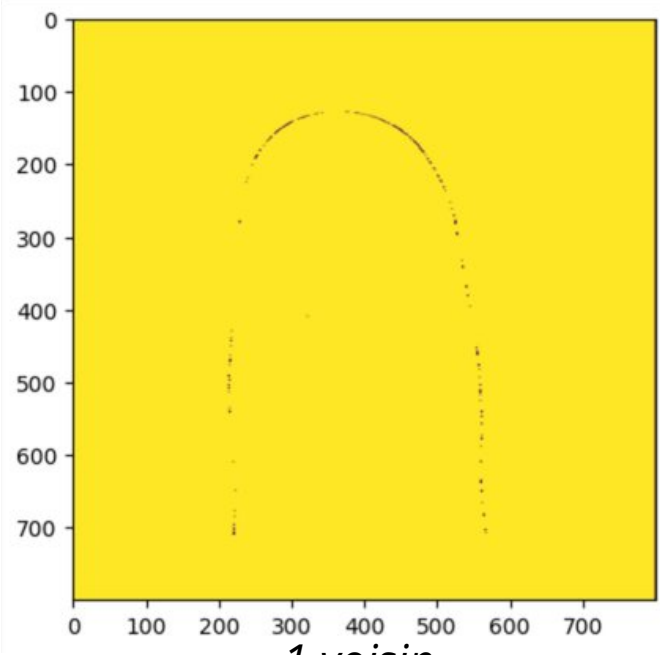
MÉTHODE EXPÉRIMENTALE N°2

DEUXIÈME IDÉE: PRODUIT DE CONVOLUTION

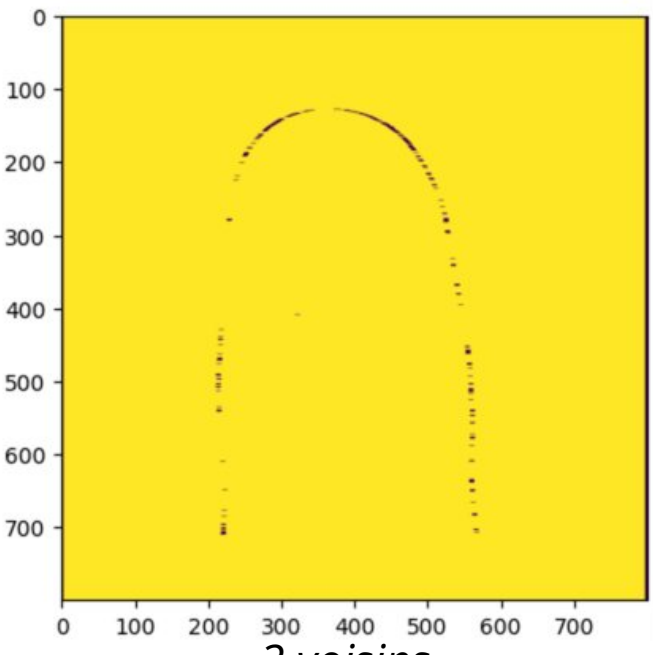
On réalise la même démarche avec un produit de convolution. Cela **permet de retirer le seuil de moyennage** car les pixels auront pour valeur 0 ou un entier non nul.

MÉTHODE EXPÉRIMENTALE N°2

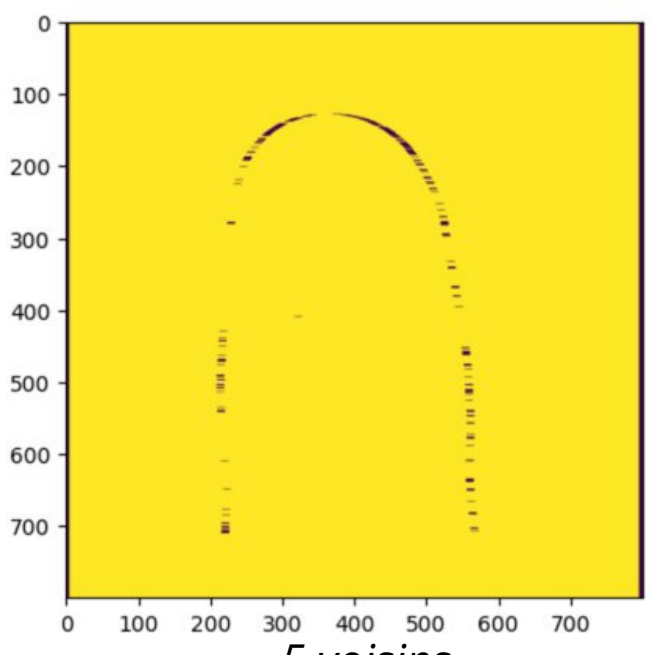
DEUXIÈME IDÉE: PRODUIT DE CONVOLUTION



1 voisin



3 voisins



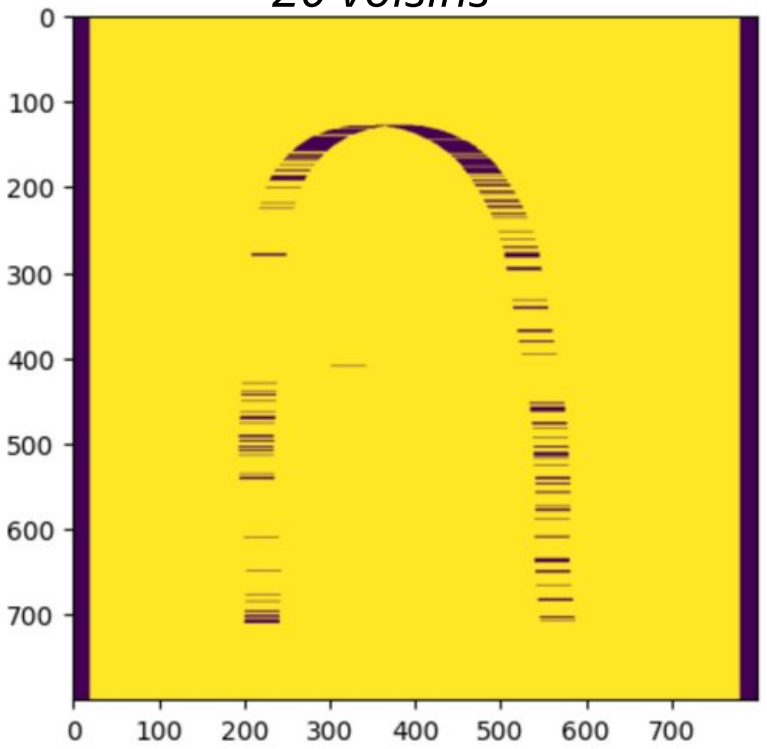
5 voisins

MÉTHODE EXPÉRIMENTALE N°2

PROBLÈMES DE CETTE MÉTHODE

1. On retouve la **perte de qualité de l'image** qui empeche de bien définir les bords de l'image.
2. Le nombre de voisin **impacte fortement la qualité et la résolution de l'image** obtenue.

20 voisins



MÉTHODE EXPÉRIMENTALE N°2

Pour palier le problème de la qualité des pixels du bord on peut utiliser le module SCIPY qui fonctionne par **morphologie mathématique** pour "**éroder**" et "**dilater**" l'image afin de réduire le bruit.

$$\text{Dilatation}(x, y) = \max_{(i,j)\in\mathcal{B}} \left(\text{Image}(x+i, y+j)\right)$$

$$\text{Érosion}(x, y) = \min_{(i,j)\in\mathcal{B}} \left(\text{Image}(x+i, y+j)\right)$$

MÉTHODE EXPÉRIMENTALE N°2

Exemple dilatation:

123	71	62
6	43	568
62	35	14
90	2	67

123	71	62
6	568	568
62	35	14
90	2	67

MÉTHODE EXPÉRIMENTALE N°2

L'ordre des opérations a une importance :

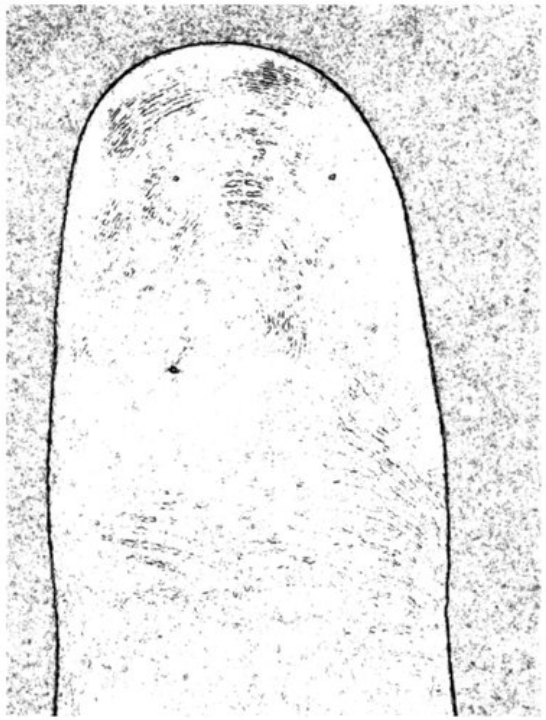
1. **Ouverture** : érosion puis dilatation, permet de retirer le bruit autour du doigt.

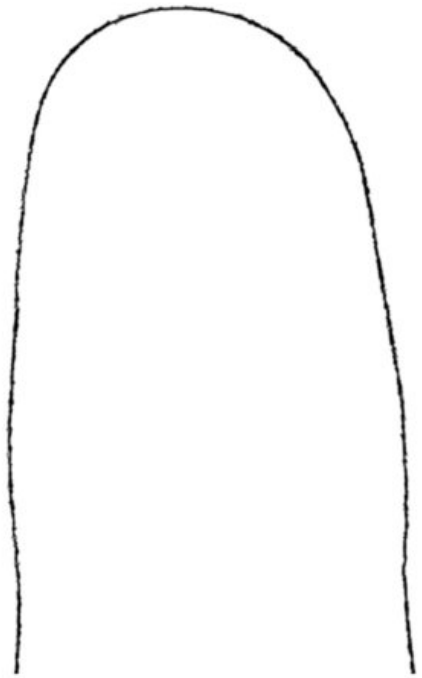
2. **Fermeture** : dilatation puis érosion, permet de retirer le bruit à l'intérieur du doigt.

On rebouche ensuite les trous formés par ces 2 opérations par du blanc pour obtenir une image nette.

MÉTHODE EXPÉRIMENTALE N°2

Application pour le doigt (seuil=380)





MÉTHODE EXPÉRIMENTALE N°2

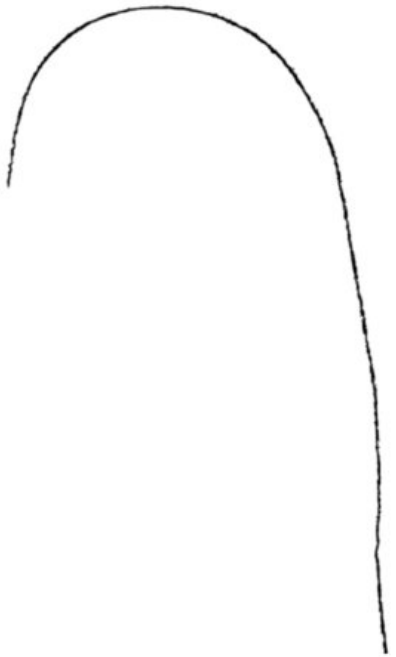
DÉTERMINATION DU BON SEUIL À APPLIQUER

De façon **empirique** on observe que les seuils sont compris **entre 200 et 1000.**

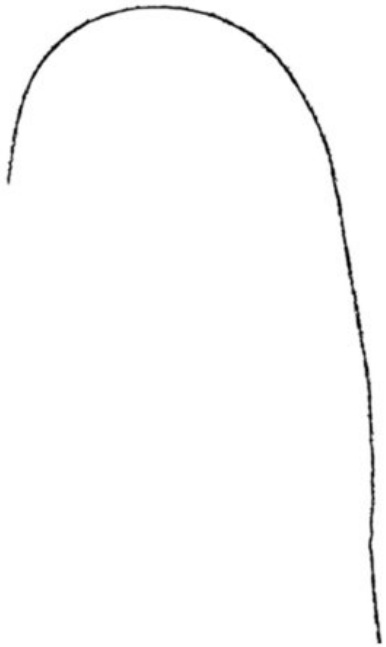
Ainsi pour **déterminer le seuil adapté pour chaque image** on part d'un seuil de 1000 puis on réduit progressivement le seuil pour déterminer à partir duquel on trouve les 2 bords du doigt.

MÉTHODE EXPÉRIMENTALE N°2

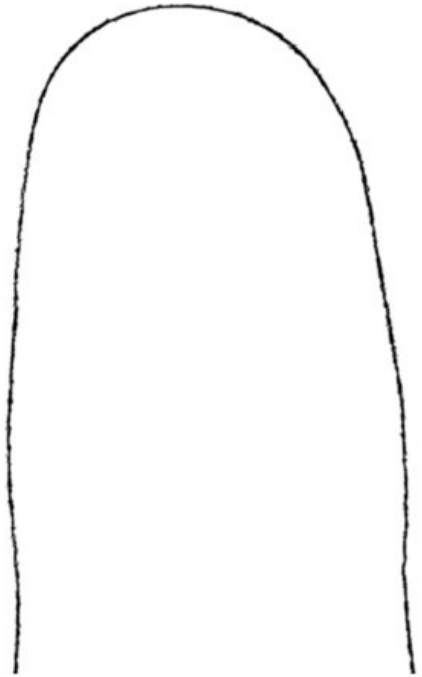
Visuellement :



seuil = 600



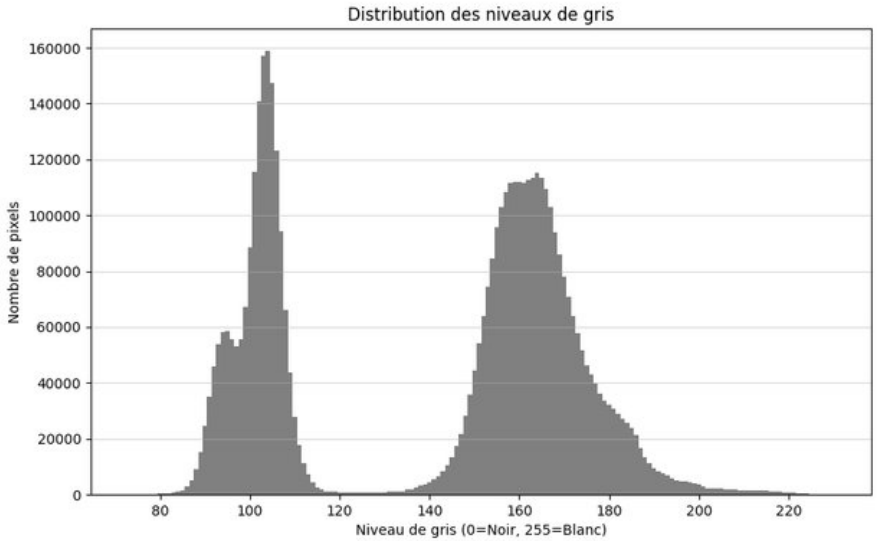
seuil = 500



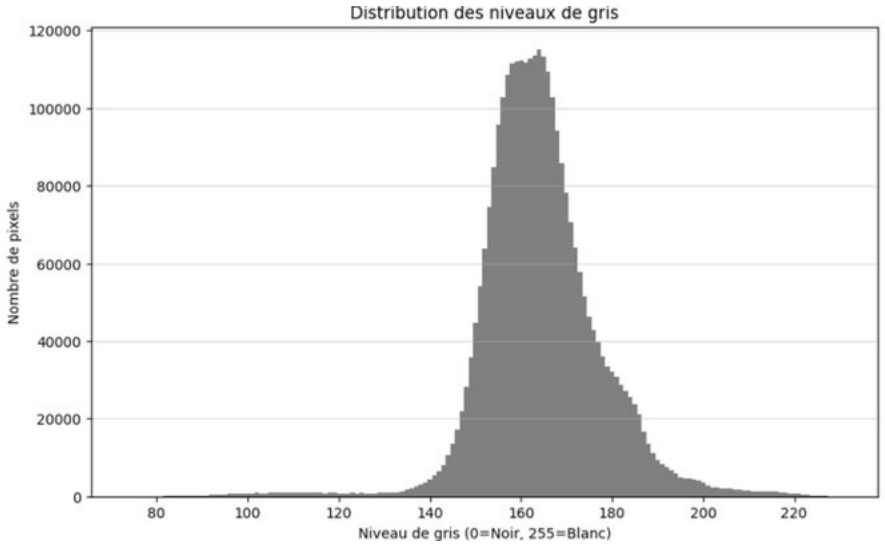
seuil = 400

MÉTHODE EXPÉRIMENTALE N°2

On obtient ainsi les **coordonnées de chaque ligne des bords** et on peut ainsi appliquer le niveau de gris sur la portion du doigt.



avant isolation



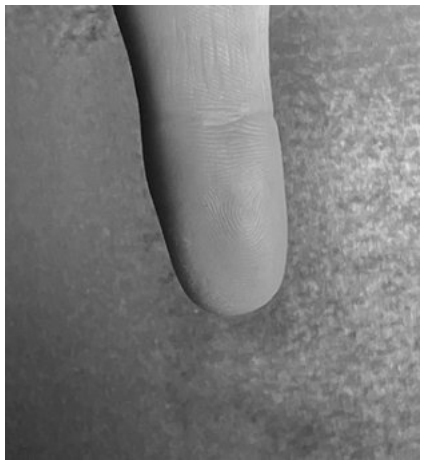
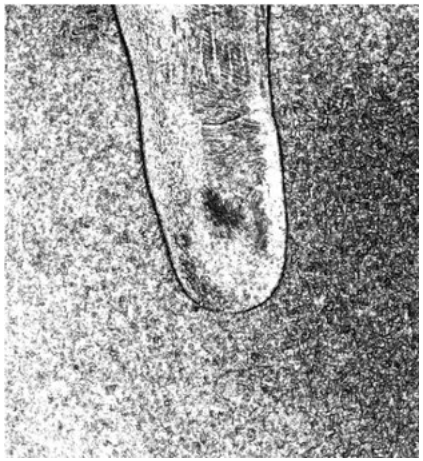
après isolation

MÉTHODE EXPÉRIMENTALE N°2

LIMITE DE CETTE MÉTHODE

Le fond ne doit pas présenter :

d'irrégularité / d'alternance de couleurs opposées


D'après le programme précedent , le seuil optimal est **100**

EXPÉRIENCE N°2

EXPÉRIENCE D'ABRASION DE LA PEAU

EXPÉRIENCE N°2

CONSIDÉRATIONS AVANT EXPÉRIENCE

- Cette abrasion de la peau se fait lors d'une **activité régulière et intense**, cette étude se livre donc pour des athlètes d'un niveau relativement élevé (entrainement 3-4 fois par semaine d'une intensité élevée).
- Pour accélerer la durée d'expérimentation, il faut **simuler les conditions d'abrasion** d'une façon plus rapide.

EXPÉRIENCE N°2

PROTOCLE

- **Abraser la peau** en frottant le doigt avec la prise.
- **Prendre en photo** le doigt avant et après abrasion.
- Étudier les niveaux de gris au fil des cycles d'abrasion pour **en déduire un ou plusieurs paramètre-s discriminant-s** pour étudier l'abrasion de la peau.

EXPÉRIENCE N°2

PREMIER SUPPORT DU DOIGT

Problème rencontré

Présence de réflexions de la lumière qui mettaient en péril la dernière méthode avec un fond trop peu homogène

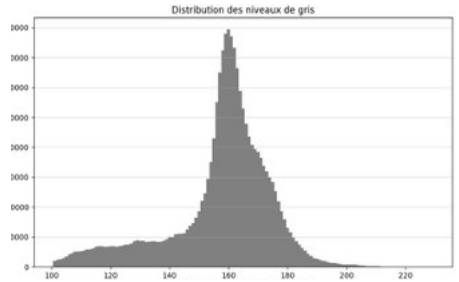
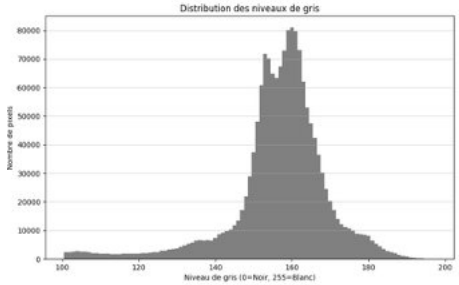
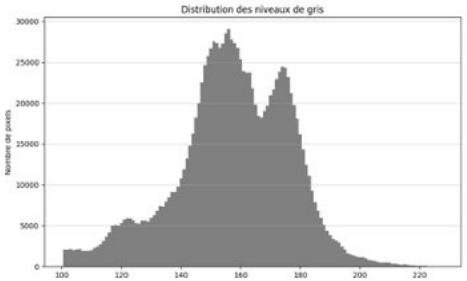
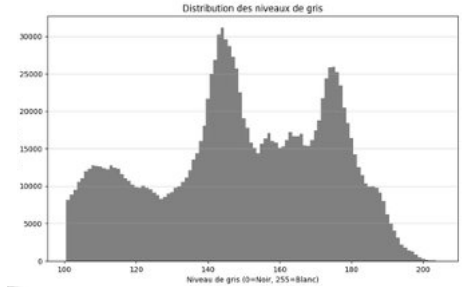
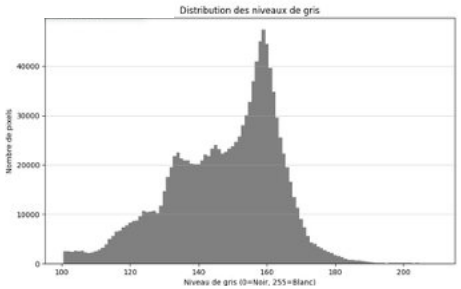
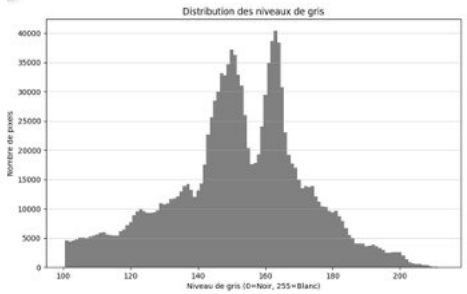
EXPÉRIENCE N°2

PREMIER CRITÈRE ENVISAGÉ

EXPÉRIENCE N°2

En considérant 3 angles pour la lumière :

Création de creux dans le niveau de gris qui s'explique par la présence de **pixels sains** et d'autres **abrasés**.

	Avant abrasion	*Après abrasion*
90°		
Côté droit		
Côté gauche		

EXPÉRIENCE N°2

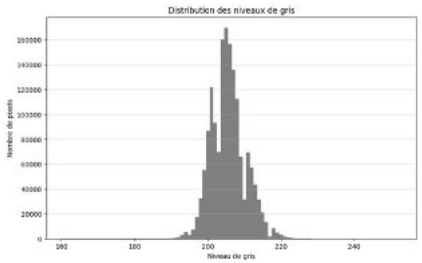
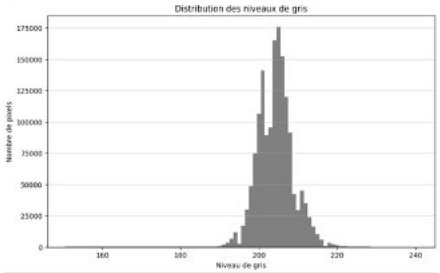
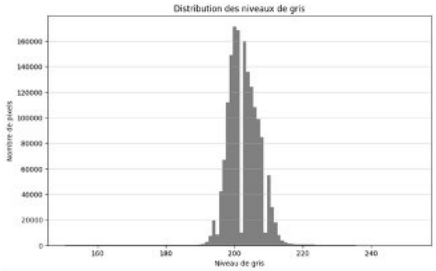
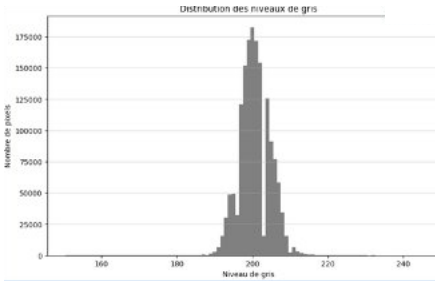
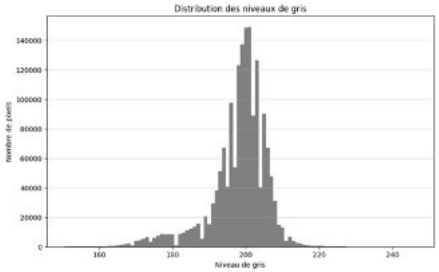
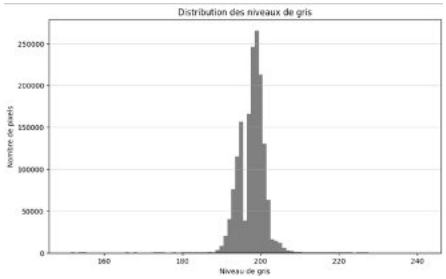
Pour **vérifier la validité de ce critère** j'ai essayé avec un fond uni et une lumière naturelle en essayant d'obtenir des images similaires.

avant abrasion

après abrasion

EXPÉRIENCE N°2

3 nouvelles configurations (**sans liquide** , avec de l'**eau** et avec de la **magnésie**) et j'ai pris en photo à intervalle régulier la peau à nu

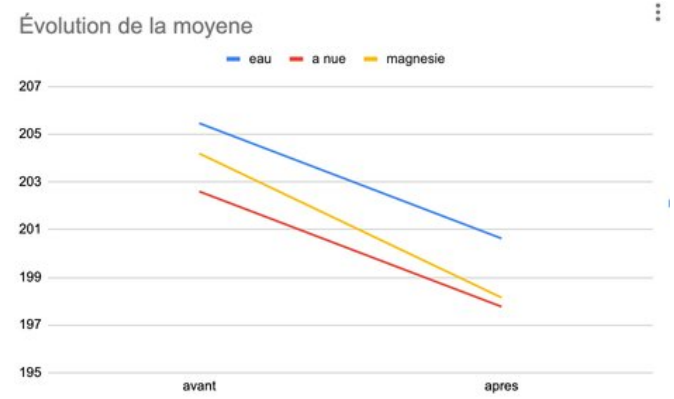
	abrasion eau	***abrasion magnésie***	***abrasion à vide***
Avant abrasion			
Après abrasion			

EXPÉRIENCE N°2

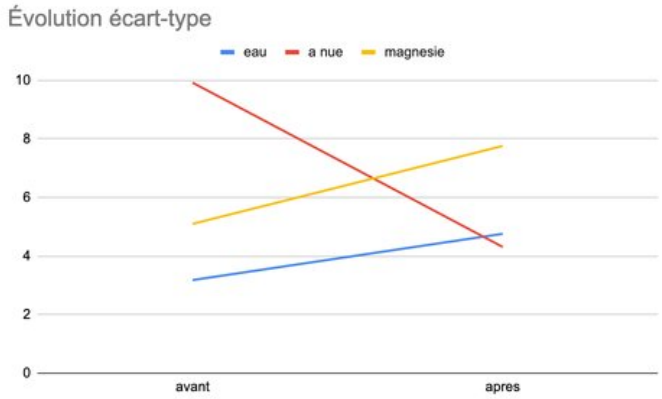
DEUXIÈME CRITÈRE ENVISAGÉ

EXPÉRIENCE N°2

Moyenne



Écart-type



EXPÉRIENCE N°2

INTERPRÉTATION ÉVOLUTION MOYENNE

- **Création de micro-creux** par l'abrasion sur la peau, phénomène de réflexion de la lumière, ce qui assombrit la peau sur l'image .

- Mise a nu des **couches plus foncées** (l'épaisseur de l'épiderme *(gris)* diminue et laisse apparaitre le derme *(rouge)*).

- **Variation de la moyenne** $\simeq 3\%$.

EXPÉRIENCE N°2

INTERPRÉTATION POUR L'ÉVOLUTION DE L'ÉCART-TYPE

- **Empreinte digitale effacée** (peau à nu) → **diminution** de l'écart type.

- **Les grains de magnésie abrasent** la peau.

- La kératine de la couche cornée absorbe l'eau et le coefficient de frottement entre la peau humide est l'eau est élevé (f=0,96), ainsi la prise **arrache des paquets de cellules** créant des creux sur la peau.

EXPÉRIENCE N°2

CONCLUSION SUR LA MÉTHODE DES NIVEAUX DE GRIS

- L'évolution de la moyene et de l'écart-type nous apprennent sur **l'abrasion de la peau sans quantification.**

- La peau **abrasée est plus rouge.** Analyser la **proportion de pixels rouge** semble donc logique.

EXPÉRIENCE N°2

TROISIÈME CRITÈRE ENVISAGÉ

EXPÉRIENCE N°2

ÉVOLUTION DE LA PROPORTION DE ROUGE DANS L'IMAGE

Évolution du pourcentage de rouge en fonction du matériau

Évolution du pourcentage de rouge en fonction du temps

EXPÉRIENCE N°2

ANALYSE DE L'ÉVOLUTION DE LA PROPORTION DE PIXELS ROUGES

- Doigt abrasé = **augmentation du nombre de pixels rouges**
- Il semble que **la peau avec de l'eau** s'abrase le plus , suivi de la **magnésie** puis de la **peau à nu** (en accord avec les sensations physiques)
- Cependant , cela dépend fortement de la **lumière** et du **positionnement du doigt**

CONSLUSION

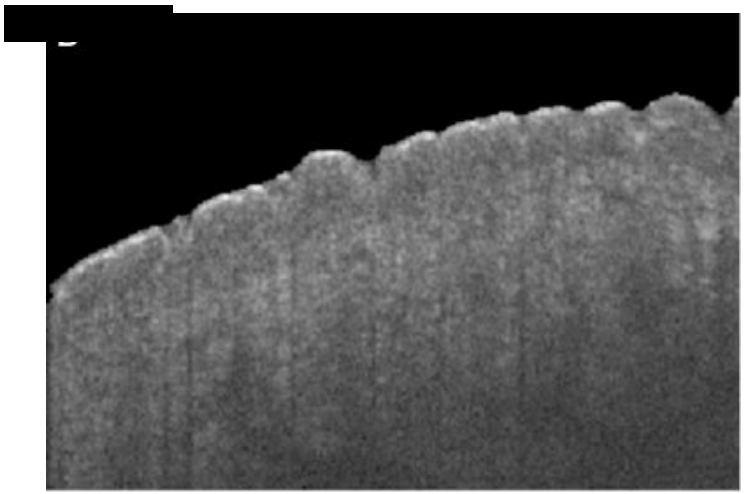
CONCLUSION SUR LA MÉTHODE D'ANALYSE DE L'ABRASION DE LA PEAU

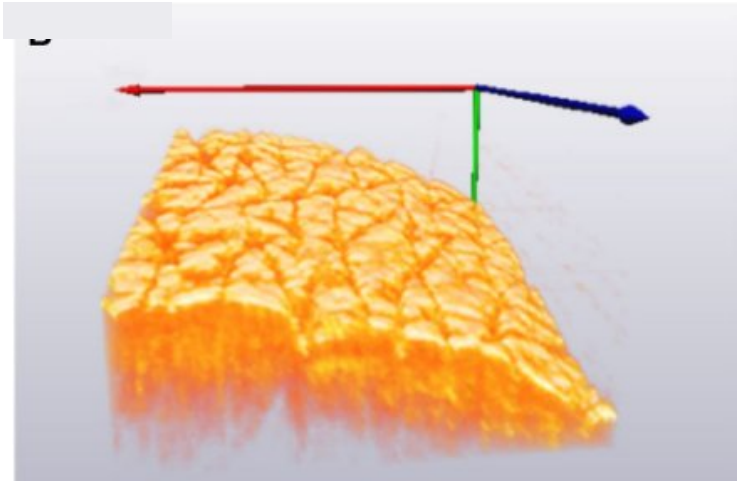
- **Mise en évidence de l'abrasion de la peau** par differents facteurs (pixels rouge , écart-type et moyenne)
- **Impossibilité de quantifier l'abrasion** a cause de trop nombreux facteurs extérieurs
- Possibilité de quantifier en faisant un rendu 3-D de la peau avec la **méthode par analyse tomographique**

CONCLUSION

MÉTHODE ANALYSE TOMOGRAPHIQUE

En analysant la façon dont se réflechit le laser envoyé pour chaque point de la peau, on créer des **images en coupes verticales** qui, compilées donnent un rendu 3D de la peau.





source:Estimation of skin surface roughness in vivo based on optical coherence tomography combined with convolutional neural network . https://www.frontiersin.org/journals/medicine/articles/10.3389/fmed.2024.1453405/ful63

MÉTHODE EXPÉRIMENTALE N°2

CONCLUSION SUR LA RÉDUCTION DE L'ABRASION DE LA PEAU

- Selon : "*Use of 'chalk' in rock climbing: sine qua non or myth?*" , *F.X.Li ,S.Margetts et I.Fowler*, la magnésie réduit le coefficient de frottement sur le long terme d'une séance (**asséchèment de la peau**, **couche superficielle de magnésie**)

- Ainsi pour optimiser une séance d'escalade , il faut **utiliser de la magnésie** , **nettoyer régulierement sa peau** pour retirer le surplus de magnésie et bien **hydrater la peau pour minimser l'assèchement**

ANNEXE

```python
from matplotlib import image
import cv2
import os
import imageio.v3 as iio
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
from scipy.signal import convolve2d
from scipy import ndimage
import json
import pandas as pd


def contour (g, s): # g: image en niveaux de gris, s: seuil
    o_f=None # chemin de sortie optionnel
    i_c="resultats" # répertoire des résultats
    s = float(s)
    i_c = np.where(g < s, 255, 0)
    d = i_c # répertoire de sortie
    h = o_f if o_f else datetime.now().strftime("%Y-%m-%d_%H-%M-%S") # horodatage
    os.makedirs(d, exist_ok=True)
    n_i = f"{d}/img_{h}.png" # nom du fichier image
    plt.figure(figsize=(8, 8))
    plt.imshow(i_c, cmap='gray', vmin=0, vmax=255)
    plt.axis('off')
    plt.savefig(n_i)
    plt.show()
    print(f"Image sauvegardée : {n_i}")
    return i_c

```

ANNEXE

```python
def analyse(i, o=None, f="resultats", t=None): # i: image à analyser, o: ID image originale, f: dossier sortie, t: horodatage personnalisé
    if isinstance(i, str):
        m = iio.imread(i) # image lue
    else:
        m = i
    d = f # répertoire de sortie
    os.makedirs(d, exist_ok=True)
    h = t if t else datetime.now().strftime("%Y-%m-%d_%H-%M-%S") # horodatage
    n = f"{d}/img_{h}.png" # nom du fichier image
    if m.ndim >= 2:
        iio.imwrite(n, m)
        print(f"Image sauvegardée : {n}")
    else:
        print("L'image est un tableau 1D; l'enregistrement en tant qu'image a été ignoré.")
    p = m.flatten() # pixels aplatis
    p_f = p[p>160] # pixels filtrés
    print(f"Nombre total de pixels analysés : {p.size}")
    u, c = np.unique(p_f, return_counts=True) # niveaux et comptes uniques
    me = np.median(p) # médiane
    mo = np.mean(p_f) # moyenne
    e = np.std(p_f) # écart-type
    q1 = np.percentile(p, 25) # 1er quartile
    q3 = np.percentile(p, 75) # 3ème quartile
    plt.figure(figsize=(10, 6))
    plt.bar(u, c, color='gray', width=1.0)
    plt.title("Distribution des niveaux de gris")
    plt.xlabel("Niveau de gris")
    plt.ylabel("Nombre de pixels")
    plt.grid(axis='y', alpha=0.5)
    n_g = f"{d}/graph_{h}.png" # nom du fichier graphique
    plt.savefig(n_g)
    plt.show()
    s = { # statistiques
        "original_image_id": o,
        "total_pixels": p.size,
        "ecart_type": float(e),
        "q1": float(q1),
        "q3": float(q3),
        "mediane": float(me),
        "moyenne": float(mo)}
    n_s = f"{d}/stats_{h}.json" # nom du fichier de statistiques
    with open(n_s, 'w') as jf:
        json.dump(s, jf, indent=4)
    print(f"Statistiques sauvegardées : {n_s}")
    return s

```

ANNEXE

```python
def gradient(i): # i: image
    if isinstance(i, str):
        m = iio.imread(i) # image lue
        g = np.dot(m[..., :3], [0.299, 0.587, 0.114]) # image en niveaux de gris
    elif isinstance(i, np.ndarray):
        g = i # image en niveaux de gris
        if len(g.shape) == 3:
            g = np.dot(g[..., :3], [0.299, 0.587, 0.114])
    k1 = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]) # noyau de Sobel
    k2 = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]]) # noyau de Sobel
    gx = convolve2d(g, k1, mode='same', boundary='symm') # gradient X
    gy = convolve2d(g, k2, mode='same', boundary='symm') # gradient Y
    gt = gx**2 + gy**2 # magnitude du gradient
    gd = np.clip(gt / np.max(gt) * 255, 0, 255).astype(np.uint8) # image du gradient normalisée
    return gt


def moyenne_glissante(i,w): # i: image, w: taille de la fenêtre
  m = iio.imread(i) # image lue
  g = m[:,:,0] # image en niveaux de gris
  h,l,a = np.shape(m) # hauteur, largeur, alpha
  R=np.zeros((h,l)) # résultat
  for r in range (h): # r: indice de ligne
    for c in range (w,l-w): # c: indice de colonne
      s=0.0 # somme des valeurs
      for o in range (0,2*w+1): # o: décalage
        s+=g[r,c-w+o]
      R[r,c]=s/(2*w+1)
  plt.imshow(R)
  plt.show()
  return R

```

ANNEXE

```python
def produit_glissant(m,w): # m: image, w: taille de la fenêtre
    d = iio.imread(m) # image lue
    h,l,a=np.shape(d) # hauteur, largeur, alpha
    g=d[:,:,0] # image en niveaux de gris
    R=np.zeros((h,l)) # résultat
    for r in range (h): # r: indice de ligne
        for c in range (w,l-w): # c: indice de colonne
            p=1.0 # produit des valeurs
            for o in range (0,2*w+1): # o: décalage
                p*=g[r,c-w+o]
            if p==0:
                R[r,c]=0
            else:
                R[r,c]=255
    plt.imshow(R)
    plt.show()
    return R


def filtrage_moyenne(L,s): # L: image d'entrée, s: seuil
    h,l=np.shape(L) # hauteur, largeur
    M=np.zeros((h,l)) # image de sortie
    for r in range (h): # r: indice de ligne
        for c in range (l): # c: indice de colonne
            if L[r,c]>=s:
                M[r,c]=255
            else:
                M[r,c]=0
    plt.imshow(M)
    plt.show()

```

ANNEXE

```python
def filtrage_produit(M): # M: image d'entrée
  h,l=np.shape(M) # hauteur, largeur
  L=np.zeros((h,l)) # image de sortie
  for r in range (h): # r: indice de ligne
    for c in range (l): # c: indice de colonne
      if M[r,c]==0:
        L[r,c]=0
      else:
        L[r,c]=255
  plt.imshow(L)
  plt.show()
  return L


def inversion_couleur(i): # i: image
    m = iio.imread(i) # image lue
    v = 255 - m # image inversée
    return v


def niveaudegris_opencv(i): # i: image
    g = None # image en niveaux de gris
    if isinstance(i, str):
        o = iio.imread(i) # image originale
        if o.shape[-1] == 3:
            b = cv2.cvtColor(o, cv2.COLOR_RGB2BGR) # image BGR
            g = cv2.cvtColor(b, cv2.COLOR_BGR2GRAY) # image en niveaux de gris
        elif o.shape[-1] == 4:
            b = cv2.cvtColor(o, cv2.COLOR_RGBA2BGR) # image BGR
            g = cv2.cvtColor(b, cv2.COLOR_BGR2GRAY) # image en niveaux de gris
        else:
            g = o
    elif isinstance(i, np.ndarray):
        if len(i.shape) == 3:
            g = cv2.cvtColor(i, cv2.COLOR_BGR2GRAY) # image en niveaux de gris
        elif len(i.shape) == 1:
            g = i
        else:
            g = i
    return g


```

ANNEXE

```python
def comptage_pixels_noir(i): # i: image
    m=iio.imread(i) # image lue
    g=m[:,:,0] # image en niveaux de gris
    h,l,a = m.shape # hauteur, largeur, alpha
    R=np.zeros((h,l)) # résultat
    for r in range (h): # r: indice de ligne
        cb=0 # compteur de noirs
        for c in range(l): # c: indice de colonne
            if g[r,c]==0:
                cb+=1
    return R


def extraire_doigt(i): # i: image
    h, l = i.shape # hauteur, largeur
    R = np.full((h, 2), -1, dtype=int) # tableau des coordonnées
    for r in range(h): # r: indice de ligne
        pc = -1 # colonne du premier pixel
        dc = -1 # colonne du dernier pixel
        for c in range(l): # c: indice de colonne
            if i[r, c] == 255:
                pc = c
                break
        for c in range(l - 1, -1, -1): # c: indice de colonne
            if i[r, c] == 255:
                dc = c
                break
        if pc != -1:
            R[r, 0] = pc
            R[r, 1] = dc
    return R
```

```python
def remplissage_liste(L): # L: liste de sous-listes
    m = max(len(s) for s in L) # longueur maximale
    p = [] # liste remplie
    for s in L: # s: sous-liste
        ps = s + [0] * (m - len(s)) # sous-liste remplie
        p.append(ps)
    return p


def gris(i): # i: image
    g = niveaudegris_opencv(i) # image en niveaux de gris
    if isinstance(i, str):
        id = os.path.splitext(os.path.basename(i))[0] # ID de l'image
        n = f"{id}_gris_output" # nom du dossier de sortie
    elif isinstance(i, np.ndarray):
        id = 'numpy_array_input' # identifiant de l'image
        n = 'numpy_array_gris_output' # dossier de sortie
    h = datetime.now().strftime("%Y-%m-%d_%H-%M-%S") # horodatage
    d = os.path.join(n, f"{id}_{h}") # dossier de sortie formaté
    return analyse(g, o=id, f=d, t=h)
```

ANNEXE

```python
def segmenter_par_gradient_et_composantes_connexes(i, s, a=0, d=True, v=False,):
  # i: image d'entrée, s: seuil de gradient, a: aire minimale, d: afficher le masque final, v: afficher l'image d'entrée,
    g = None # image en niveaux de gris
    if isinstance(i, str):
        g = niveaudegris_opencv(i)
        if r:
            g = rotate(g)
    elif isinstance(i, np.ndarray):
        if len(i.shape) == 3:
            g = niveaudegris_opencv(i)

    else:
        raise TypeError("L'entrée 'image_entree' doit être un chemin de fichier (str) ou un tableau NumPy.")
    if g is None:
        print("Erreur: Impossible d'obtenir l'image en niveaux de gris pour la segmentation.")
        return None
    if g is None:
        print("Erreur: La rotation de l'image a échoué.")
        return None
    gr = gradient(g) # gradient de l'image
    m = (gr >= s) # masque des bords
    k = np.ones((3, 3), dtype=bool) # noyau de structuration
    me = ndimage.binary_erosion(m, structure=k, iterations=1) # masque érodé
    md = ndimage.binary_dilation(me, structure=k, iterations=1) # masque dilaté
    e, nc = ndimage.label(md) # étiquettes, nombre de composants
    tc = ndimage.sum(md, e, range(nc + 1)) # tailles des composants
    if nc > 0:
        tf = tc[1:] # tailles filtrées
        lf = np.arange(1, nc + 1) # labels filtrés
        mv = tf >= a # masque valide
        lfi = lf[mv] # labels filtrés indexés
        tfi = tf[mv] # tailles filtrées indexées
        mpf = np.zeros_like(g, dtype=np.uint8) # masque du plus grand composant
        if len(lfi) > 0:
            pgl = lfi[np.argmax(tfi)] # label du plus grand composant
            mpf = (e == pgl).astype(np.uint8) * 255
            mpf = ndimage.binary_fill_holes(mpf).astype(np.uint8) * 255
    else:
        mpf = np.zeros_like(g, dtype=np.uint8)
    ma = 255 - mpf # masque d'arrière-plan
    if d:
      plt.figure(figsize=(12, 6))
      if np.all(mpf == 0):
          print("Aucun objet détecté pour la segmentation finale. Affichage du masque d'arrière-plan vide (blanc).")
          plt.imshow(ma, cmap='gray')
          plt.title("Masque d'arrière-plan (aucun objet détecté)")
      else:
          plt.imshow(ma, cmap='gray')
          plt.axis('off')
      plt.show()
    return extraire_doigt(mpf)
```

ANNEXE

```
294 def application_ndg(n,i): # n: coordonnees des bords, i: image d'entree
295   L = n.tolist() # liste des coordonnées
296   M = [] # lignes de pixels
297   g = niveaudegris_opencv(i) # image en niveaux de gris
298   for r in range (len(L)): # r: indice de ligne
299     K = [] # pixels temporaires
300     if L[r][0] != -1 and L[r][1]!= -1 and L[r][1] > L[r][0]:
301       for c in range (L[r][0], L[r][1]): # c: indice de colonne
302         K.append(g[r][c])
303     if K!=[]:
304       M.append(K)
305   T = np.array(remplissage_liste(M), dtype=np.uint8) # tableau de pixels
306   return analyse(niveaudegris_opencv(T),o=None, f="resultats", t=None)
307
```

ANNEXE

```python
def find_optimal_threshold(i, mn=300, m=1500, st=30, ca=5000, w=5, clr=0.90):
    # i: image d'entrée, mn: seuil minimum, m: seuil maximum, st: pas du seuil,
    # ca: aire minimale du composant, w: largeur minimale, clr: ratio de continuité des lignes
    bt = -1 # meilleur seuil
    bb = None # meilleurs bords
    print(f"Recherche du meilleur seuil pour {i}...")
    print(f"Plage de seuils de gradient : de {m} à {mn} par pas de -{st}")
    try:
        d = iio.imread(i) # image lue
        h = d.shape[0] # hauteur de l'image
    except Exception as e:
        print(f"Erreur lors de la lecture de l'image pour obtenir la hauteur: {e}")
        return None, None
    rc = int(h * clr) # lignes continues minimales requises
    print(f"Exigence: au moins {rc} lignes consécutives avec des bords valides (soit {clr*100:.0f}% de la hauteur de l'image).")
    for gt in range(m, mn - 1, -st): # gt: seuil de gradient
        pl = segmenter_par_gradient_et_composantes_connexes(i,gt,ca,d=False,v=False)

        if pl is None:
            print(f"  Seuil {gt}: Segmentation a échoué.")
            continue
        mlf = 0 # longueur continue maximale trouvée
        cl = 0 # compteur de lignes continues
        for r in range(pl.shape[0]): # r: indice de ligne
            sc = pl[r, 0] # colonne de début
            ec = pl[r, 1] # colonne de fin
            if sc != -1 and ec != -1 and (ec - sc) >= w:
                cl += 1
            else:
                mlf = max(mlf, cl)
                cl = 0
        mlf = max(mlf, cl)
        print(f"  Seuil {gt}: Longueur continue max: {mlf} lignes.")
        if mlf >= rc:
            bt = gt
            bb = pl
            print(f"  Seuil {gt}: Critère de continuité atteint ({mlf} >= {rc} lignes). Arrêt de la recherche.")
            break
    print(f"\n--- Résultat de la recherche ---")
    if bt != -1:
        print(f"Meilleur seuil de gradient trouvé (le plus élevé avec continuité) : {bt}")
        print(f"Affichage de la segmentation avec le meilleur seuil trouvé :")
        segmenter_par_gradient_et_composantes_connexes(i,bt,ca,d=True,v=True)
        return bt, bb
    else:
        print(f"Aucun seuil n'a permis de détecter une séquence continue d'au moins {rc} lignes avec des bords distincts selon les critères.")
        return None, None


```

ANNEXE

```python
def vraindg(s,i): # s: seuil de gradient, i: image d'entrée
  v=segmenter_par_gradient_et_composantes_connexes(i,s,0, r=True) # coordonnées des bords segmentés
  return application_ndg(v,i)


def rotate(i, c=False): # i: image, c: conserver la couleur
  r = None # image à faire pivoter
  if isinstance(i, str):
    r = iio.imread(i)
  elif isinstance(i, np.ndarray):
    r = i
  if r is None:
      print("Error: Could not load image for rotation.")
      return None
  if not c and r.ndim == 3:
      if r.shape[-1] == 3:
          r = cv2.cvtColor(r, cv2.COLOR_RGB2GRAY)
      elif r.shape[-1] == 4:
          r = cv2.cvtColor(r, cv2.COLOR_RGBA2GRAY)
      elif r.shape[-1] == 1:
          r = r.squeeze()
  R = np.rot90(r) # image pivotée
  return R
```

ANNEXE

```python
def pourcentage_rouge(i, t=100): # i: image d'entrée, t: seuil minimum de rouge
    if isinstance(i, str):
        try:
            m = iio.imread(i) # image lue
        except Exception as e:
            print(f"Erreur lors de la lecture de l'image depuis le chemin: {e}")
            return 0.0
    elif isinstance(i, np.ndarray):
        m = i
    else:
        print("L'entrée doit être un chemin d'image (str) ou un tableau NumPy (np.ndarray).")
        return 0.0
    if m.ndim != 3 or m.shape[-1] < 3:
        print("L'image n'est pas une image couleur RGB/RGBA valide. Impossible de calculer la proportion de rouge.")
        return 0.0
    if m.shape[-1] == 4:
        m = m[..., :3]
    cr = m[:, :, 0] # canal R
    cg = m[:, :, 1] # canal G
    cb = m[:, :, 2] # canal B
    mp = (cr > cg) & \ # masque des pixels rouges
                     (cr > cb) & \
                     (cr > t)
    npix = np.sum(mp) # nombre de pixels rouges
    tp = m.shape[0] * m.shape[1] # total des pixels
    if tp == 0:
        return 0.0
    p = npix / tp # pourcentage de rouge
    return p

```

ANNEXE

```python
def extraire_interieur_doigt_couleur(b, o): # b: coordonnées des bords, o: image couleur originale pivotée
    h, l, _ = o.shape # hauteur, largeur
    fi = np.zeros_like(o, dtype=o.dtype) # image de l'intérieur du doigt
    for r in range(h): # r: indice de ligne
        if r < b.shape[0]:
            sc = b[r, 0] # colonne de début
            ec = b[r, 1] # colonne de fin
            if sc != -1 and ec != -1 and ec > sc:
                sc = max(0, sc)
                ec = min(l, ec)
                fi[r, sc:ec] = o[r, sc:ec]
    return fi


def calculer_proportion_rouge_doigt(i, s, t): # i: image d'entrée, s: seuil de gradient, t: seuil minimum de rouge
    oc = iio.imread(i) # image couleur originale
    if oc.shape[-1] == 4:
        oc = oc[..., :3]
    ocr = rotate(oc, c=True) # image couleur originale pivotée
    gr = niveaudegris_opencv(ocr) # image en niveaux de gris pivotée
    b = segmenter_par_gradient_et_composantes_connexes( # coordonnées des bords
        gr,
        s,
        a=0,
        d=False,
        v=False,
        r=False
    )
    if b is None:
        print(f"La détection des bords pour {i} a échoué.")
        return 0.0
    fi = extraire_interieur_doigt_couleur( # image de l'intérieur du doigt
        b,
        ocr
    )
    pr = pourcentage_rouge(fi, t=t) # proportion de rouge
    print(f"Proportion de pixels rouges dans l'intérieur du doigt: {pr:.2%}")
    return pr
```