

TIPE

Hydrophobie des matériaux : application aux vitrages

Par quels procédés physico-chimiques peut-on faire en sorte que les vitres d'un gratte-ciel restent propres le plus longtemps possible ?

Introduction

- I) L'hydrophobie dynamique
- II) L'aspect texturé des surfaces
- III) La composition chimique des substrats



Introduction

Objectifs de la cire :

Hydrophobe, pas très épaisse, transparente / incolore, qui ne reflète pas

Demande d'échantillon à l'entreprise Saint-Gobain

Nos cires :

- cire abeilles / meuble
- cire à épiler
- cire de chaussure
- cire de bougie
- suie

I) L'hydrophobie dynamique

Arrachements - 1

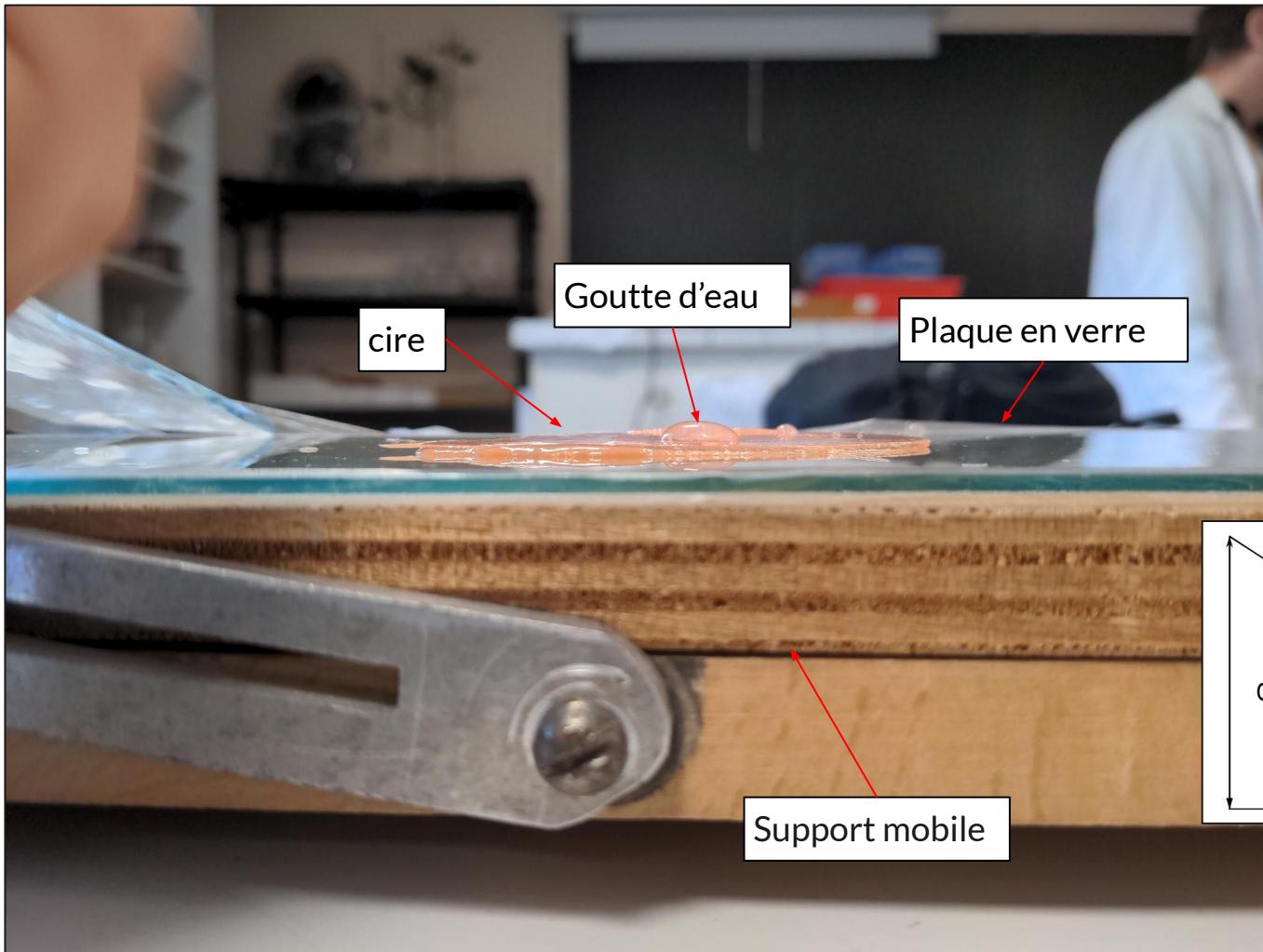
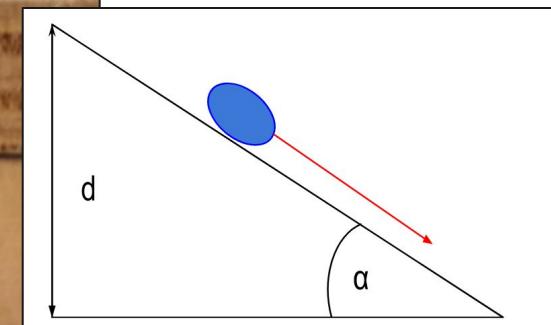


Schéma représentatif :

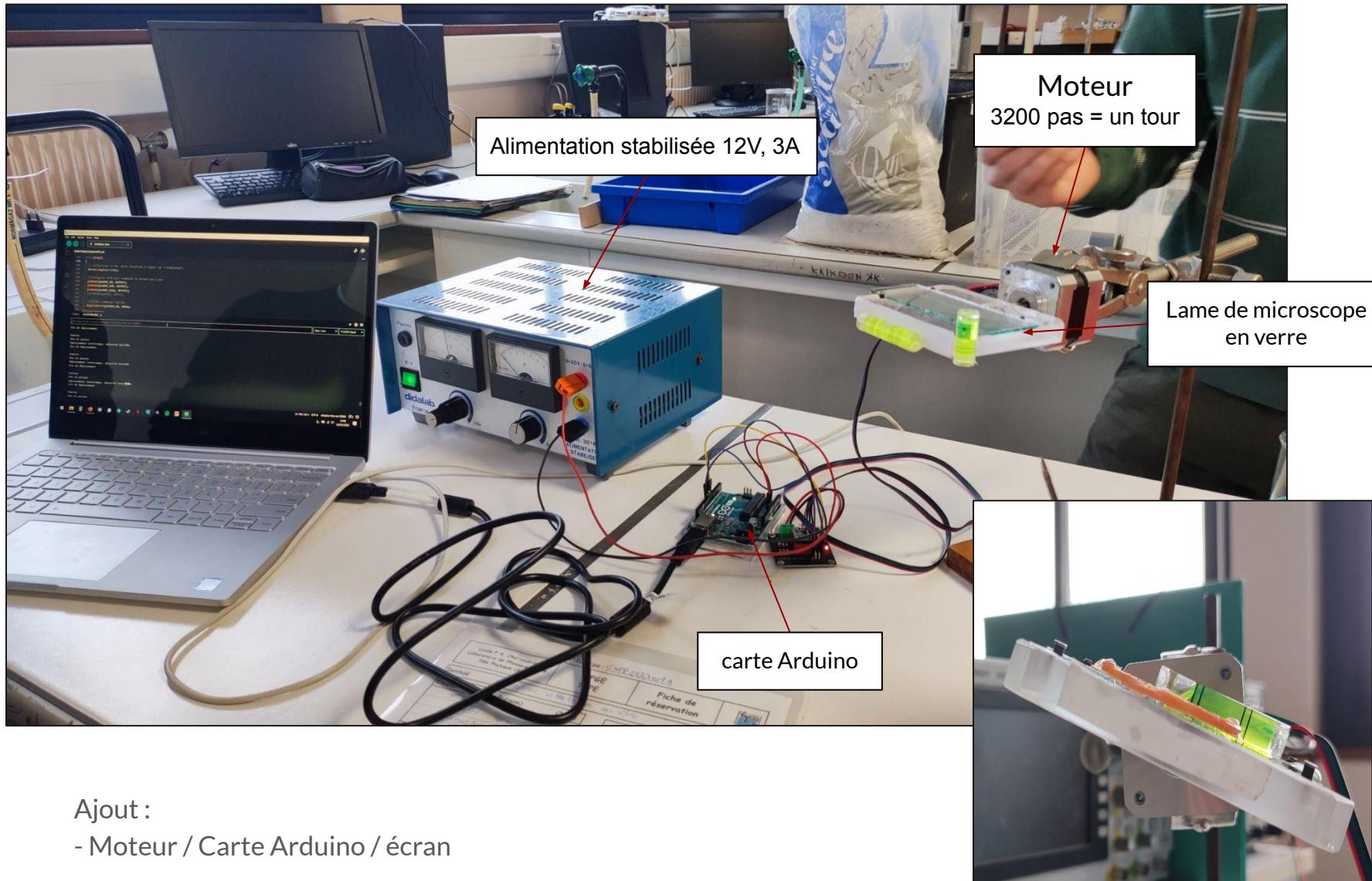


Objectif de l'expérience :

Montrer que les cires n'ont pas le même angle d'arrachement

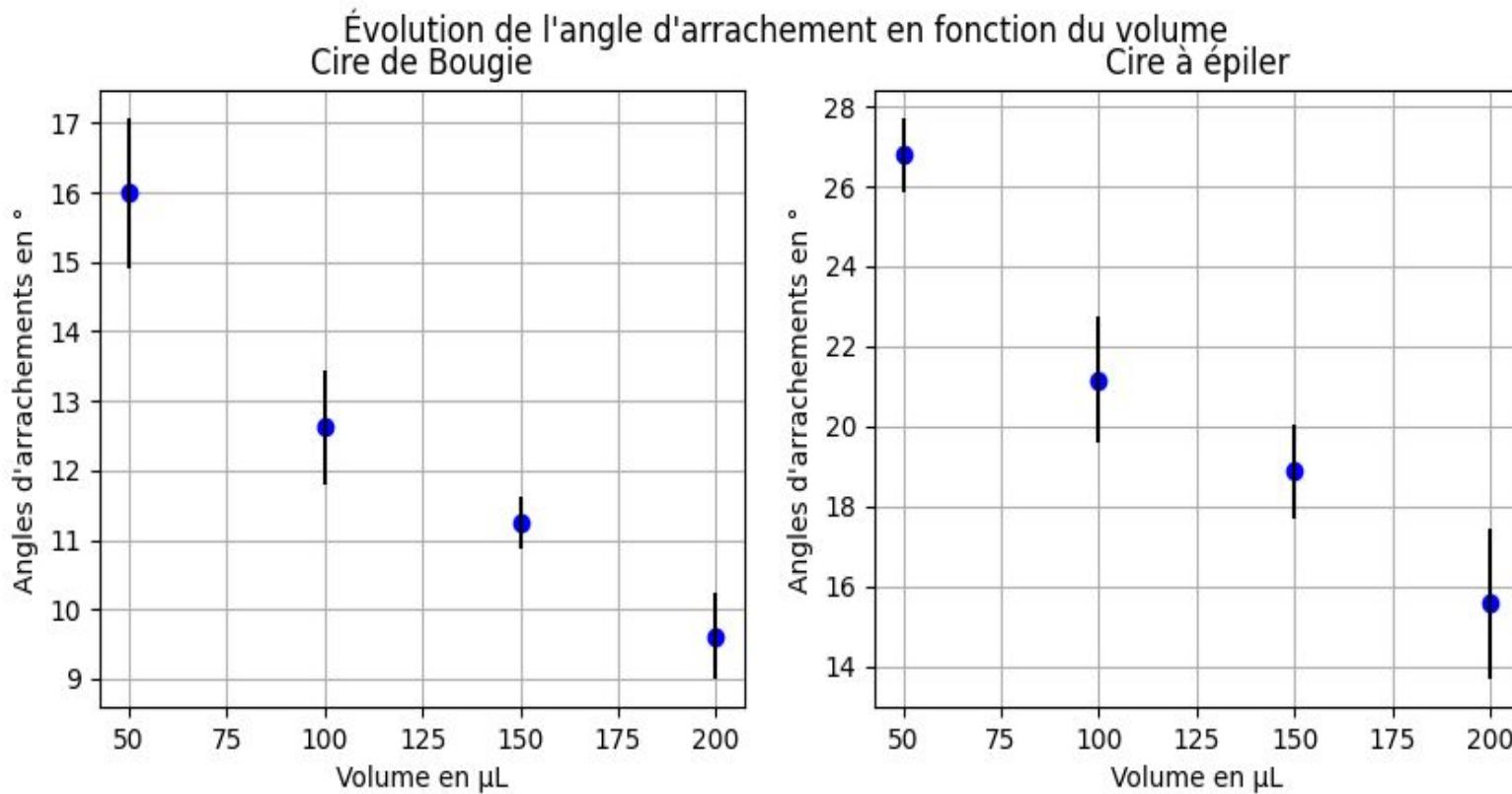
I) L'hydrophobie dynamique

Arrachements - 2



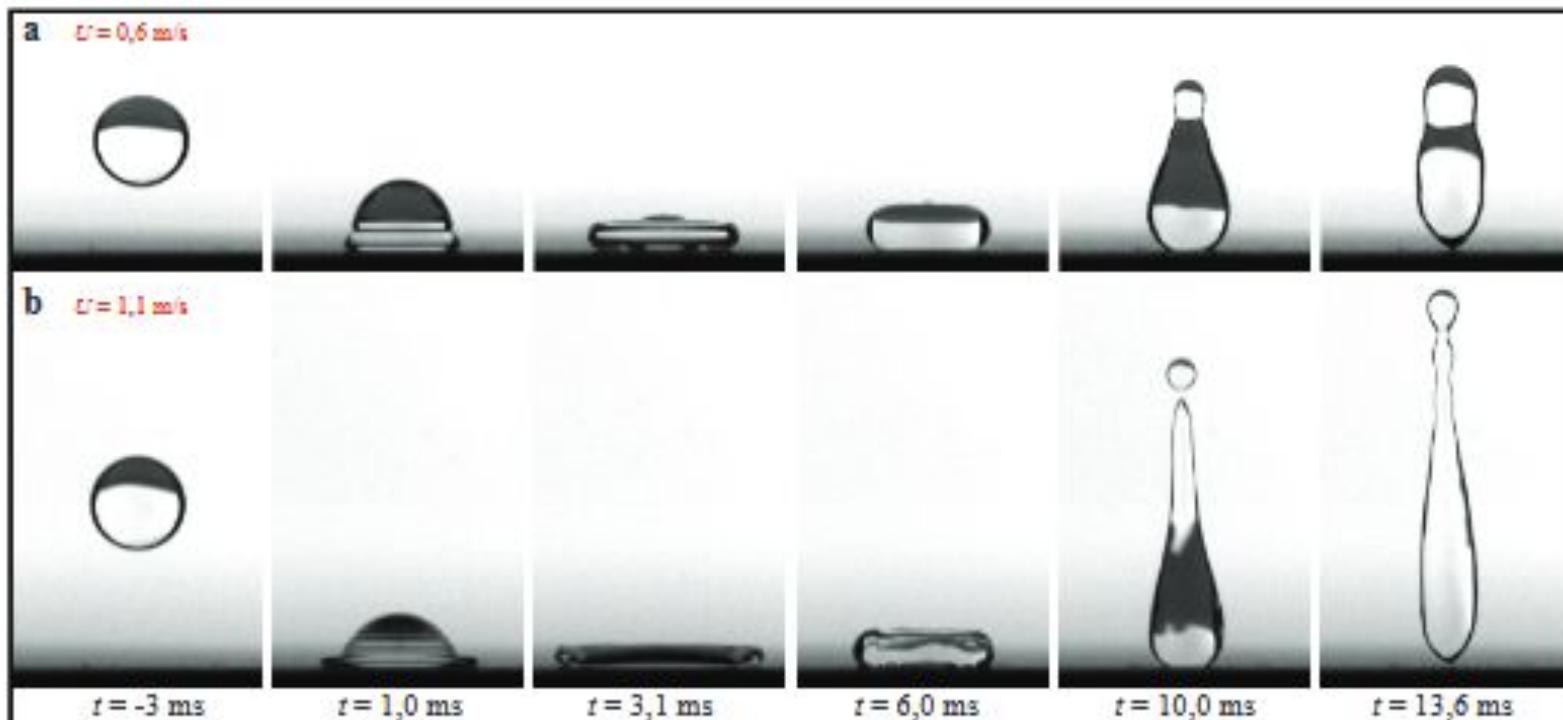
I) L'hydrophobie dynamique

Arrachements - Résultats



I) L'hydrophobie dynamique

Rebonds - Théorie



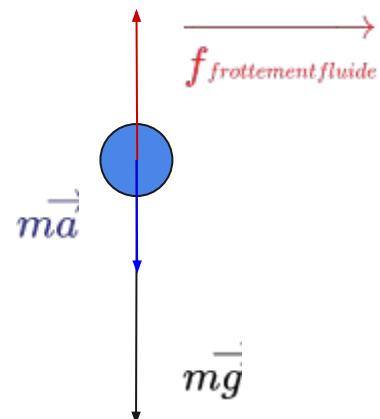
$$\tau_0 \sim \sqrt{\frac{\rho R^3}{\gamma}}.$$

$$R_{max} \sim R \left(\frac{\rho v^2 R}{\gamma} \right)^{\frac{1}{4}}$$

I) L'hydrophobie dynamique

Rebonds - Vitesse de la goutte

Schéma d'une goutte d'eau en chute libre :



Principe fondamental de la dynamique appliquée à la goutte :

$$m\vec{a} = m\vec{g} - \frac{1}{2} C_x \rho_{air} S v \vec{v}$$

Par projection :

$$v = \sqrt{\frac{4Rg\rho_{eau}}{3C_x\rho_{air}}}$$

$$v = 7.7 \text{ m/s} = 28 \text{ km/h}$$

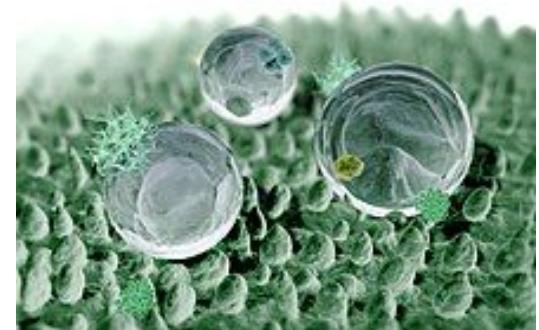
Donc, on a :

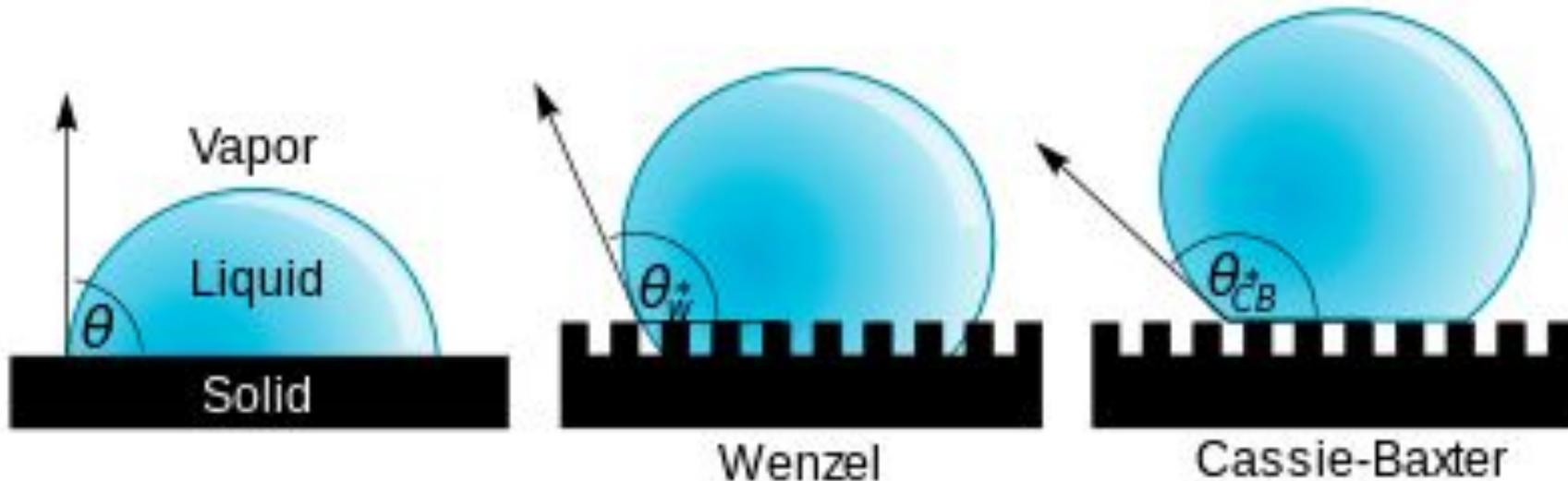
$$R_{max} = \frac{0,27}{\gamma^{\frac{1}{4}}}$$

$$\tau_0 = \frac{0,11}{\sqrt{\gamma}}$$

II) L'aspect texturé des surfaces

Le biomimétisme



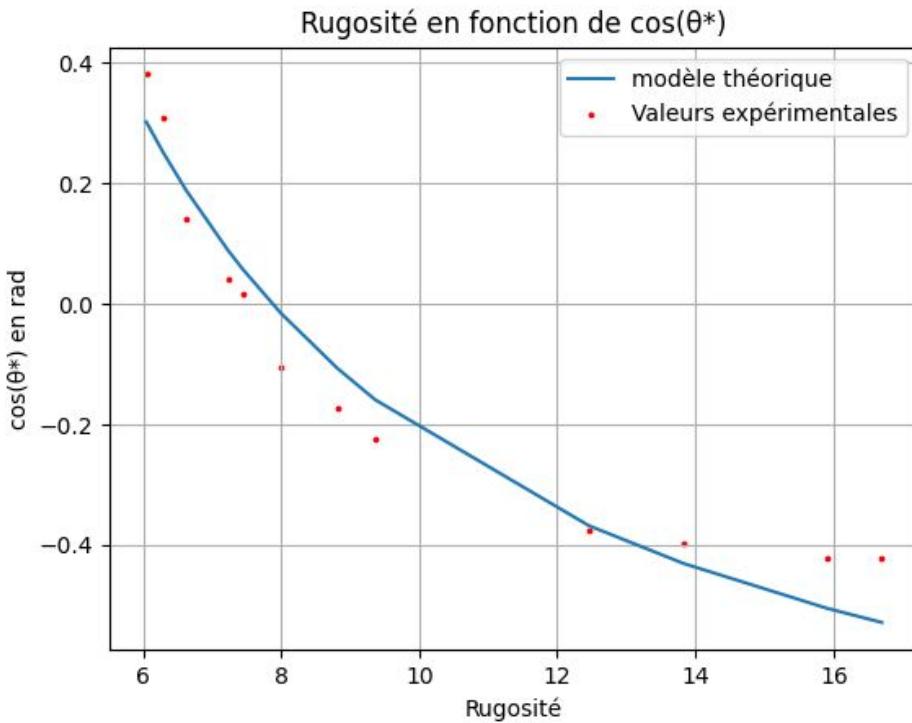


$$[2] \cos(\theta^*_W) = r^* \cos(\theta_e)$$

$$[3] \cos(\theta^*_{CB}) = -1 + \phi_s (\cos(\theta_e) + 1)$$

Rugosité : $r^* = S_{réelle} / S_{apparente}$

II) L'aspect texturé des surfaces



$$[3] \cos(\theta^{*CB}) = -1 + \phi_s(\cos(\theta_e) + 1)$$

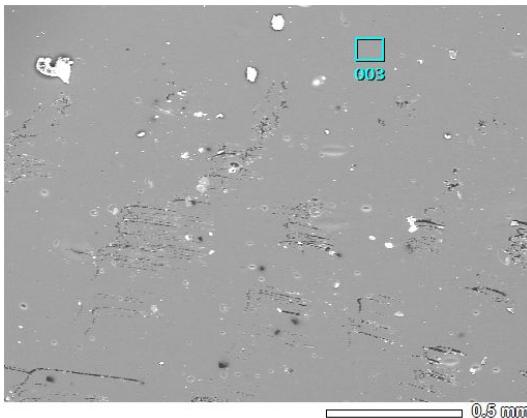
File - C:\Users\Nico\Documents\Etudes\PC\TPE\Courbe Cassie-Baxter.py

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 r = np.array([6.048, 6.302, 6.624, 7.232, 7.441, 7.
996, 8.813, 9.364, 12.459, 13.831, 15.9, 16.7])
5
6 costeta = np.cos(np.array([1.18, 1.256, 1.43, 1.53, 1
.553, 1.675, 1.745, 1.797, 1.955, 1.98, 2.007, 2.007
]))
7
8 g = -1 + (4.5/r)*(0.75 + 1)
9
10 plt.title("Rugosité en fonction de  $\cos(\theta^*)$ ")
11 plt.plot(r,g, label='modèle théorique')
12 plt.scatter(r, costeta, s=3, c='red', label =
'Valeurs expérimentales')
13 plt.xlabel('Rugosité')
14 plt.ylabel('cos( $\theta^*$ ) en rad')
15 plt.legend()
16 plt.grid()
17 plt.show()
```

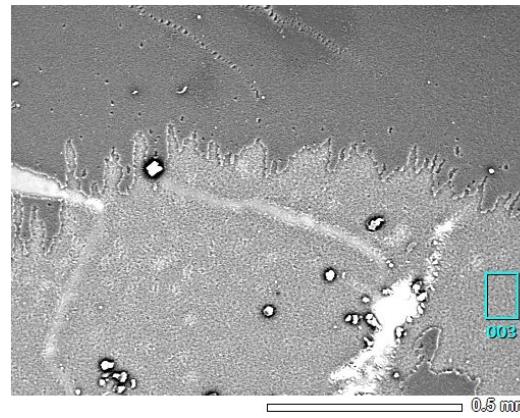
II) L'aspect texturé des surfaces

Microscope

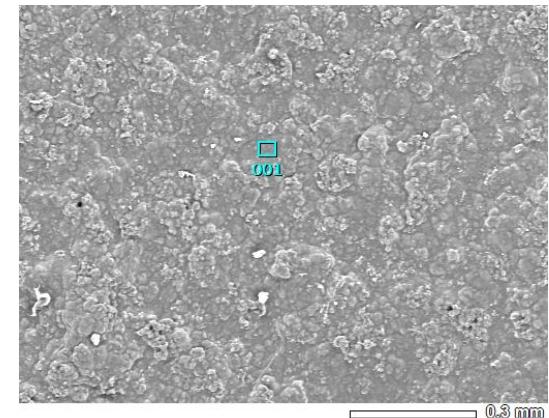
Cire épilation



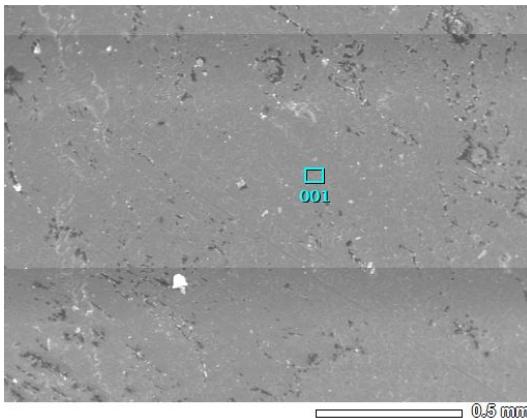
Suie



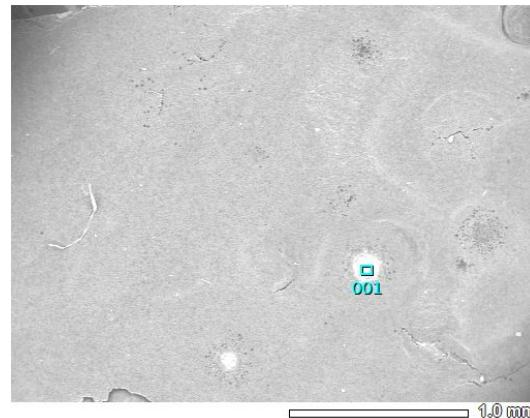
Cire de chaussures



Cire bougie

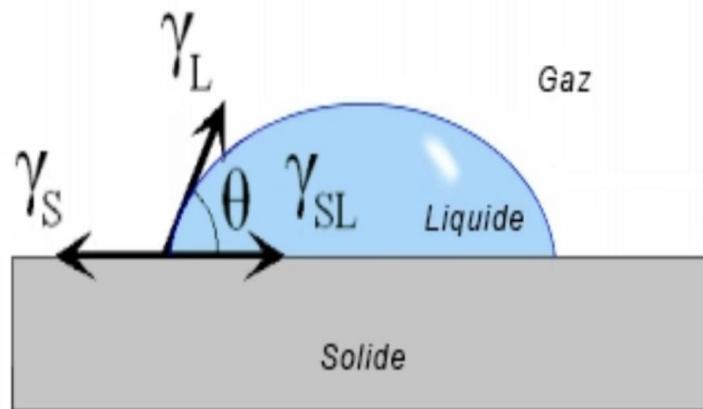


Cire à bois



III) La composition chimique des substrats

Théorie



- Définition du « Mouillage » S :

$$[2] S = \gamma_s - (\gamma_{SL} + \gamma)$$

Donc $S > 0$: mouillage total
Et $S < 0$: mouillage partiel

$$S = \gamma(\cos(\theta_e) + 1)$$

- Lien expérience et théorie :

On admet que $S = V_{s.l} - V_{l.l}$ sur une surface élémentaire donnée pour un nombre de mole donné.

Ainsi $S < 0$ pour $V_{s.l} < V_{l.l}$ et : $(V_{s.l} - V_{l.l}) \frac{n}{S} = \gamma(\cos(\theta_e) + 1)$

Ce qui donne : $\gamma = (V_{s.l} - V_{l.l}) / (\cos(\theta_e) + 1)$

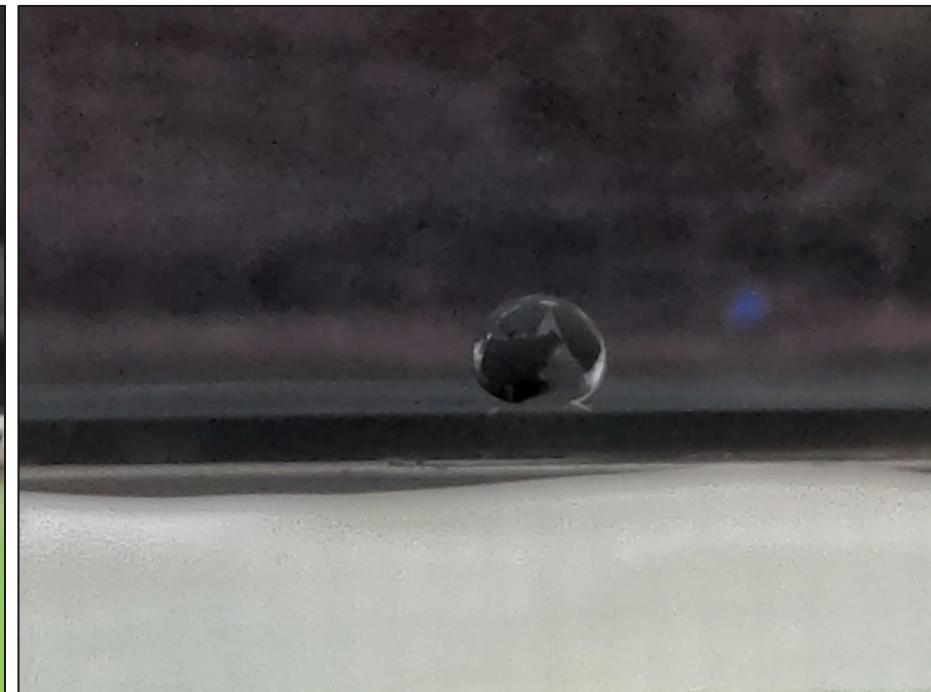
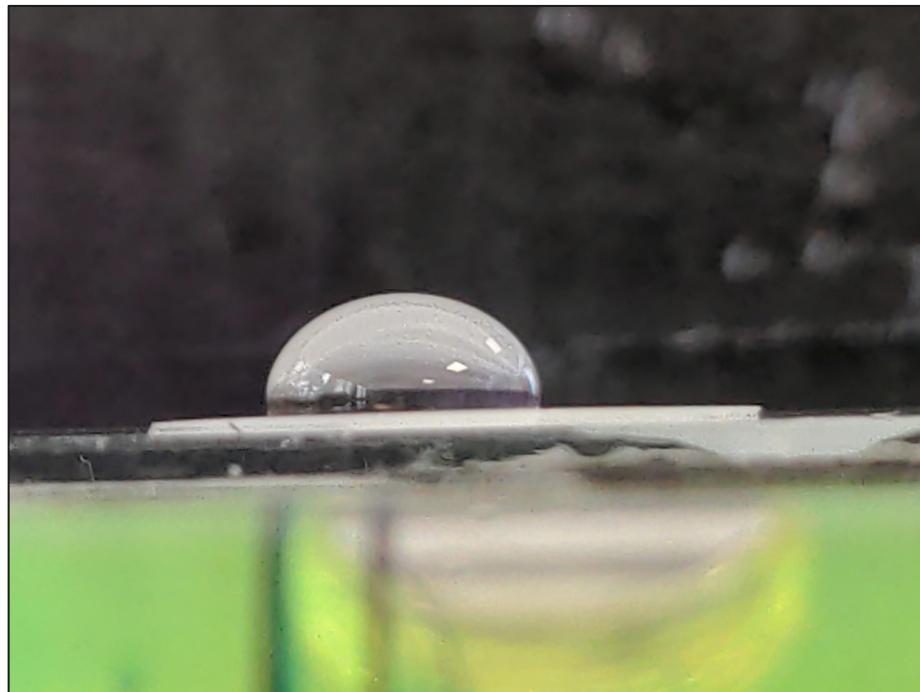
$$[1] \gamma \cos(\theta_e) = \gamma_{SL} - \gamma_s$$

On a une meilleure hydrophobie pour θ qui tend vers π (séparation à $\pi/2$)

On vérifie alors, par des mesures de θ que plus $|\gamma|$ augmente, plus l'hydrophobe d'une surface est élevée et que l'on obtient des valeurs cohérentes pour $|\gamma|$ en scalaire avec $V_{s.l}$ et $V_{l.l}$ connus.

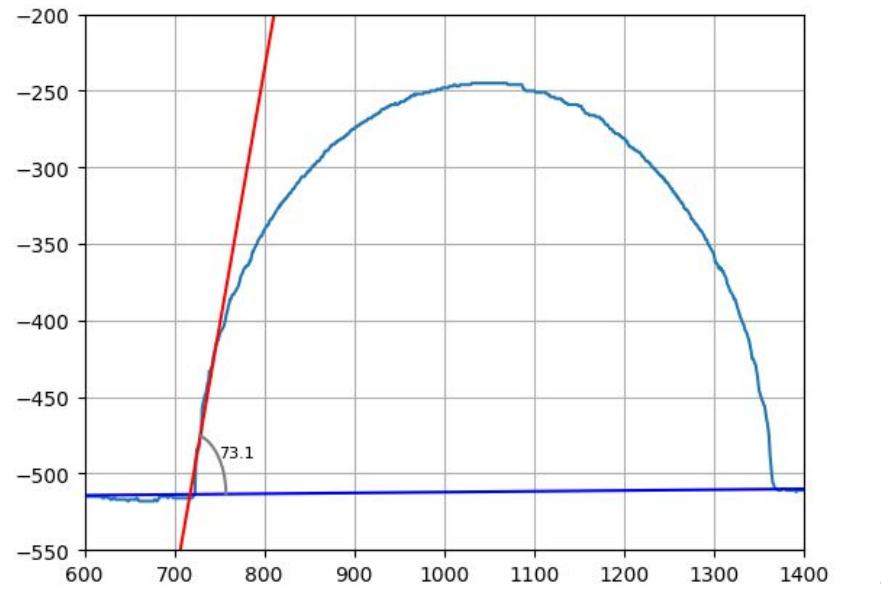
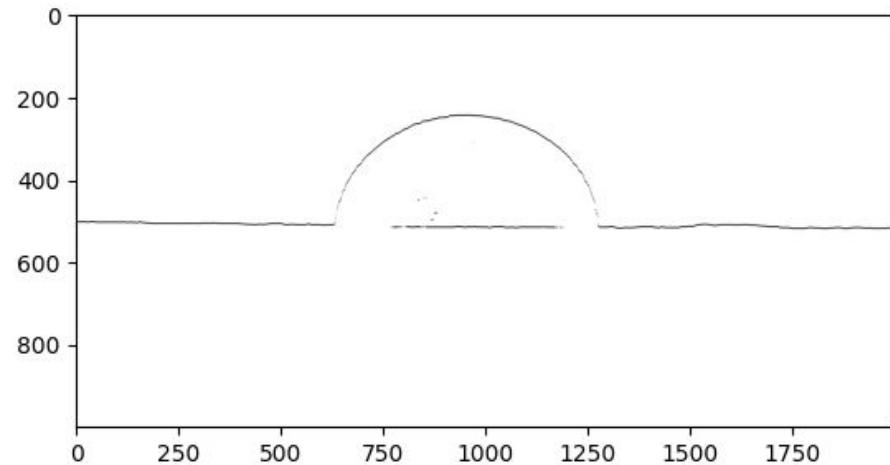
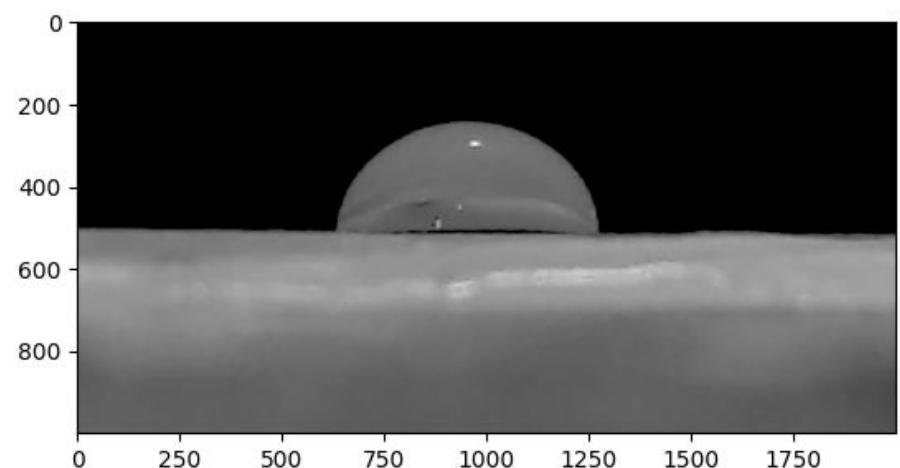
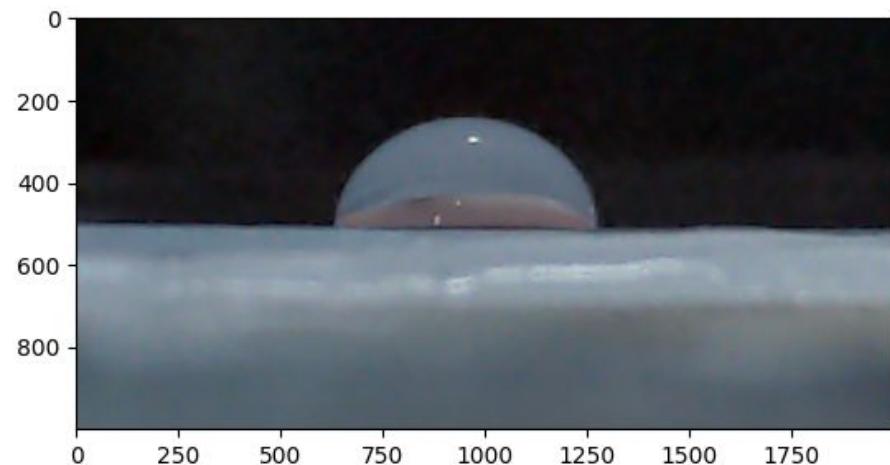
III) La composition chimique des substrats

Mesure des angles 1



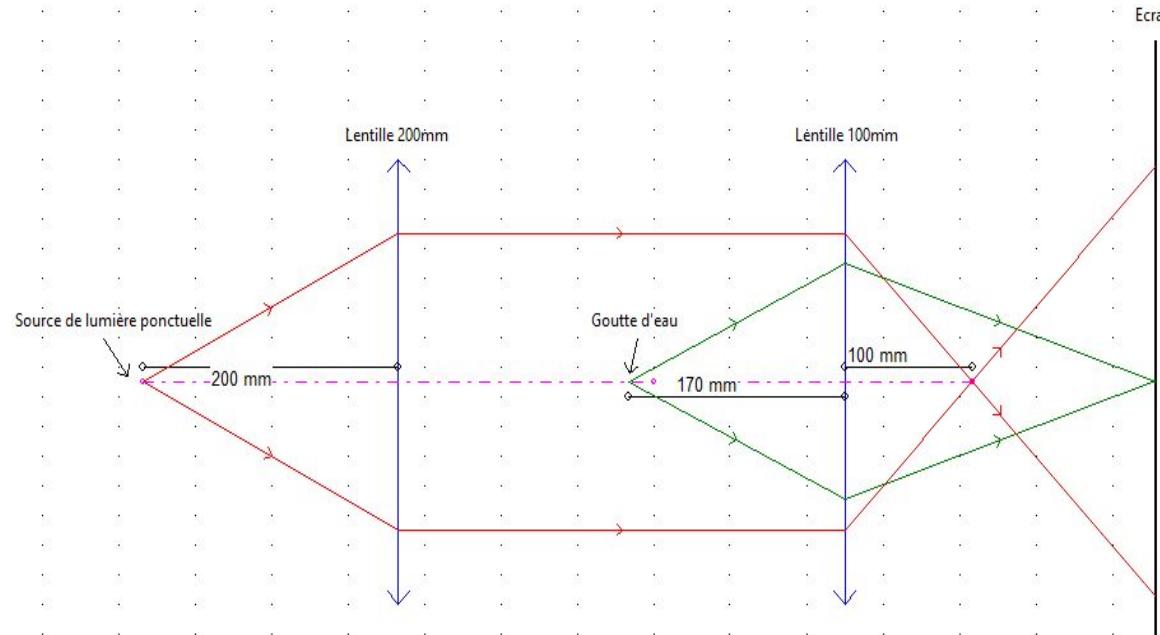
III) La composition chimique des substrats

Traitement python 1



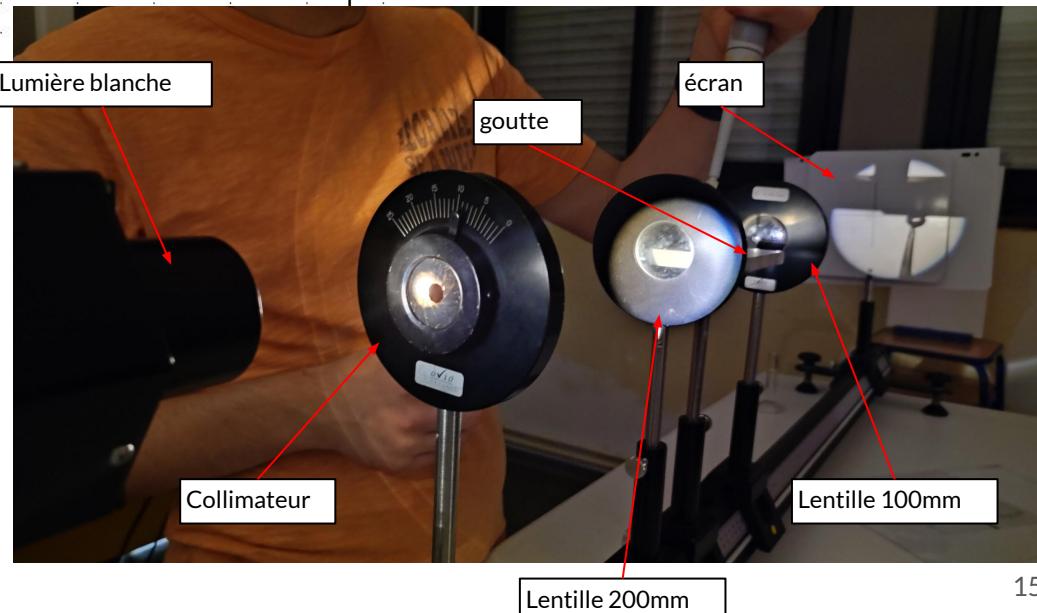
III) La composition chimique des substrats

Mesure des angles 2



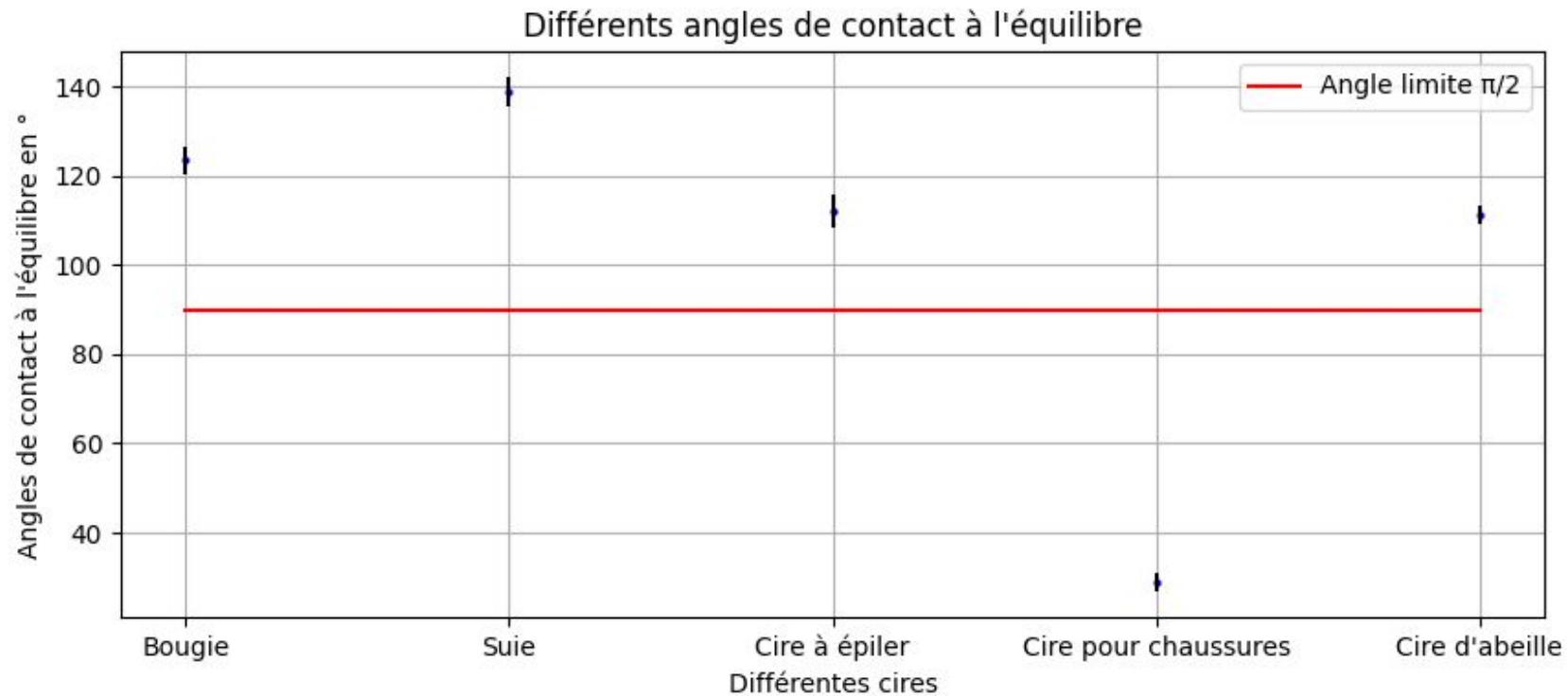
Schémas du montage

Photo du dispositif



III) La composition chimique des substrats

Résultats



Ces résultats, couplés à la composition chimique connue des différentes cires appliquées sur le verre nous permettent de montrer l'influence de la composition chimique du substrat dans l'hydrophobie.

III) La composition chimique des substrats

La silanisation

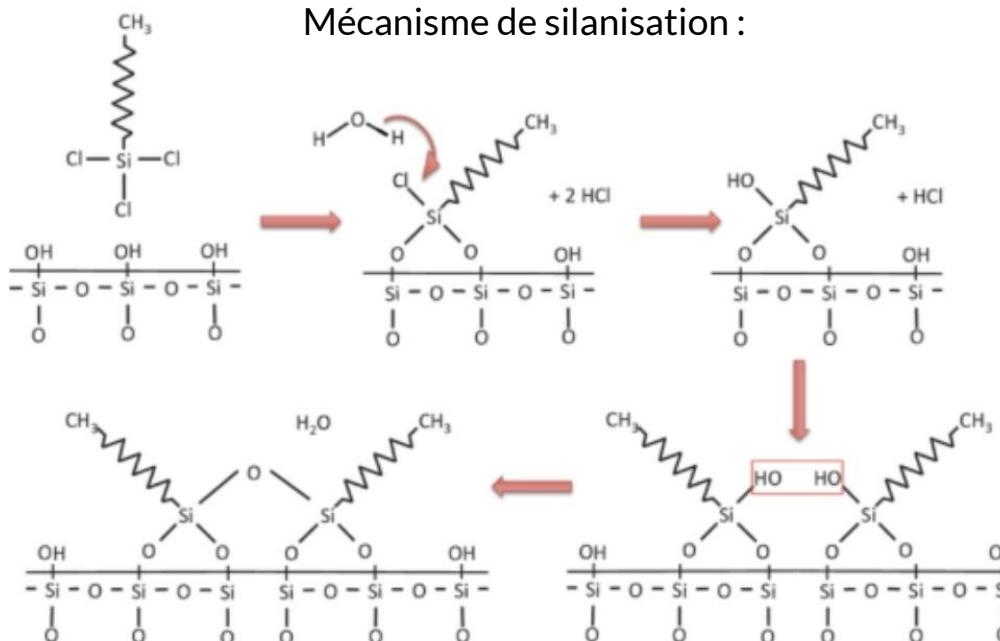
- La silanisation : processus chimique permettant de recouvrir une surface avec des molécules d'alcoxysilane organofonctionnelles.
- La silanisation (ou siliconisation) de la verrerie augmente son hydrophobie



Introduction de γ_c

Le but de la silanisation est d'abaisser le γ_c du verre et donc d'augmenter son hydrophobie.

Mécanisme de silanisation :



Conclusion

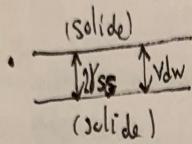


Annexe :

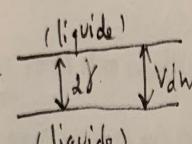
Éléments de théorie

Détermination théorique de S :

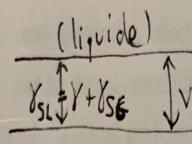
$$[\gamma] = \frac{MLT^{-2}}{L} = M\bar{T}^{-2} : \left[\begin{array}{c} \text{énergie} \\ \text{surface} \end{array} \right]$$

•  $\left. \begin{array}{l} \gamma_{SS} \\ \downarrow V_{dW} \end{array} \right\} 2\gamma_S = V_{s,S} \quad (V_{dW,S,S} \sim ?)$

 (solide) (solide)

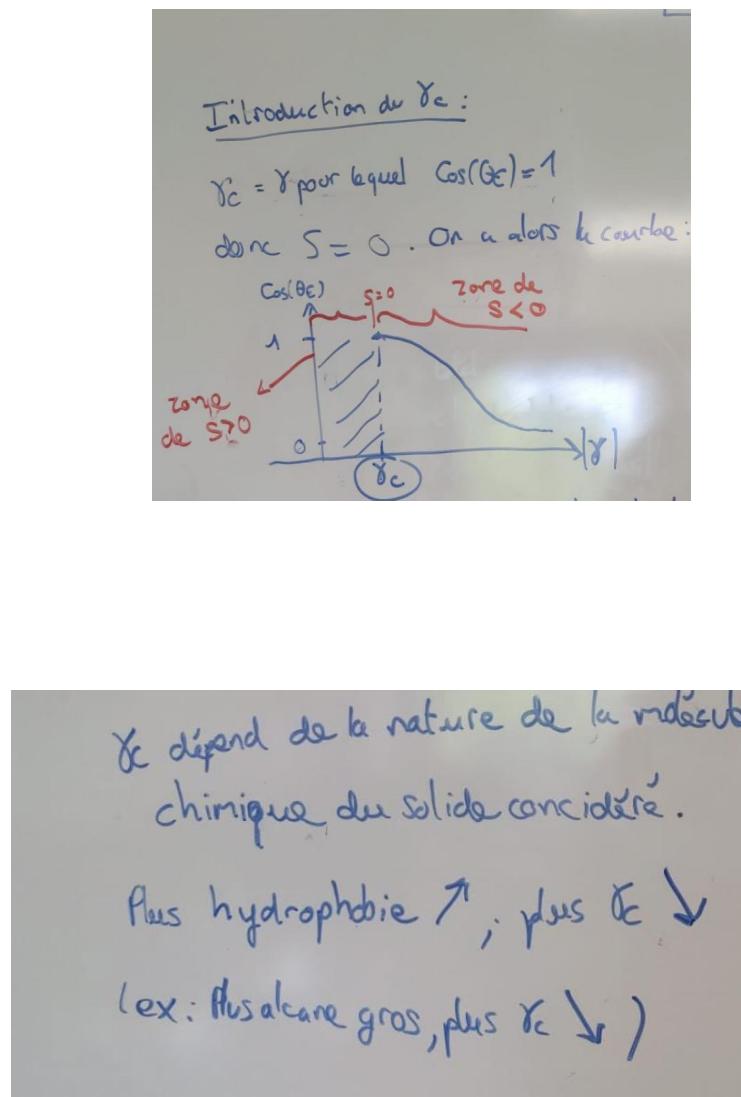
•  $\left. \begin{array}{l} \gamma_L \\ \downarrow V_{dW} \end{array} \right\} 2\gamma = V_{l,l} \quad (V_{dW,L,L} \sim 50 \text{ kJ.mol}^{-1})$

 (liquide) (liquide)

•  $\left. \begin{array}{l} \gamma_{SL} \\ \downarrow \gamma_{SG} \\ \downarrow V_{dW} \end{array} \right\} \gamma_{SL} = \gamma + \gamma_{SG} = V_{s,L}$

 (solide) (solide)

$\therefore S = \gamma_{SG} - (\gamma_{SL} + \gamma) = \gamma_{SG} - \gamma_{SL} - 2\gamma + V_{s,L} \left(\frac{\text{mol}}{\text{surface}} \right) \approx 1 \text{ mol.m}^{-2}$
 $= V_{s,L} - V_{L,L}$



Annexe :

Python - Incertitudes

File - C:\Users\Nico\Documents\Etudes\PC\Angles TIPE.py

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # Angles d'arrachement
5
6
7 V = [200, 150, 100, 50]
8
9 B200 = [8.76, 9.75, 10.47, 9.65, 9.44]
10 B150 = [11.52, 11.29, 11, 10.75, 11.64]
11 B100 = [13.52, 11.32, 12.66, 13.03, 12.58]
12 B50 = [17.13, 16.3, 16.78, 14.64, 15.1]
13 B = [B200, B150, B100, B50]
14 MoyB = [np.mean(B[0]), np.mean(B[1]), np.mean(B[2]),
           np.mean(B[3])]
15 uB = [np.std(B[0], ddof=1), np.std(B[1], ddof=1), np.
           std(B[2], ddof=1), np.std(B[3], ddof=1)]
16
17 E200 = [17.96, 16, 15.73, 12.69, 15.45]
18 E150 = [20.12, 19, 19.51, 16.95, 18.76]
19 E100 = [23.9, 21, 20.66, 20.38, 19.89]
20 E50 = [27.95, 26, 27.48, 26.66, 25.82]
21 E = [E200, E150, E100, E50]
22 MoyE = [np.mean(E[0]), np.mean(E[1]), np.mean(E[2]),
           np.mean(E[3])]
23 uE = [np.std(E[0], ddof=1), np.std(E[1], ddof=1), np.
           std(E[2], ddof=1), np.std(E[3], ddof=1)]
24
25 figure = plt.figure(figsize=(10, 4))
26
27 plt.subplot(121)
28 plt.plot(V, MoyB, 'ob')
29 plt.errorbar(V, MoyB, yerr=uB, fmt='k')
30 plt.ylabel("Angles d'arrachements en °")
31 plt.xlabel("Volume en µL")
32 plt.title("Cire de Bougie")
33 plt.suptitle("Évolution de l'angle d'arrachement en
               fonction du volume")
34 plt.grid()
35
36 plt.subplot(122)
```

Page 1 of 3

File - C:\Users\Nico\Documents\Etudes\PC\Angles TIPE.py

```
37 plt.plot(V, MoyE, 'ob')
38 plt.errorbar(V, MoyE, yerr=uE, fmt='k')
39 plt.ylabel("Angles d'arrachements en °")
40 plt.xlabel("Volume en µL")
41 plt.title("Cire à épiler")
42 plt.suptitle("Évolution de l'angle d'arrachement en
               fonction du volume")
43 plt.grid()
44
45 plt.show()
46
47 # Angles de contact à l'équilibre
48
49
50 Cires = ["Bougie", "Swie", "Cire à épiler", "Cire
           pour chaussures", "Cire d'abeille"]
51 B = [127.5, 124.5, 121, 123.5, 125, 120, 118.5, 124.5
      , 127.5, 122.5]
52 S = [141.5, 140, 138.5, 144.5, 136.5, 132, 140.5, 137
      , 137, 139.5]
53 E = [110, 112.25, 119.5, 113, 116, 111.5, 112.5, 109.
      5, 108, 108.5]
54 C = [27, 30, 28.5, 26, 29.5, 30, 27.5, 32, 31.5, 27]
55 A = [108, 114.5, 111, 110, 113.5, 112, 109.5, 114,
      110.5, 109]
56
57 M = [B, S, E, C, A]
58
59 MoyM = [np.mean(M[0]), np.mean(M[1]), np.mean(M[2]),
           np.mean(M[3]), np.mean(M[4])]
60 uM = [np.std(M[0], ddof=1), np.std(M[1], ddof=1), np.
           std(M[2], ddof=1), np.std(M[3], ddof=1), np.std(M[4
           ], ddof=1)]
61
62 figure = plt.figure(figsize=(10, 4))
63
64 plt.plot(Cires, MoyM, 'bo', markersize=2)
65 plt.errorbar(Cires, MoyM, yerr=uM, fmt='k')
66 plt.ylabel("Angles de contact à l'équilibre en °")
67 plt.xlabel("Différentes cires")
68 plt.title("Différents angles de contact à l'équilibre")
```

Page 2 of 3

File - C:\Users\Nico\Documents\Etudes\PC\Angles TIPE.py

```
68 ")
69 plt.grid()
70
71 plt.plot(Cires, [90] * len(Cires), "r", label="Angle
               limite π/2")
72 plt.legend(loc="upper right")
73 plt.show()
74
```

Page 3 of 3

▼ Fonctions

Import

```
1 from google.colab import drive # On autorise l'accès à notre drive
2 import matplotlib.pyplot as plt
3 import numpy as np
4 import cv2
```

On veut pouvoir tirer des informations de l'image, on va transformer l'image contenant des 3-uplets RGB en un seul élément de valeur = teinte de gris. Dans un second temps, on enlève toutes les valeurs "non blanches" pour faire ressortir la courbure de la goutte

```
1 # Contraste / Couleurs
2 def NoirBlanc(Image, teinte):
3     Original_2 = [] # Création d'une nouvelle image
4     for j in Image: # On prend une colonne
5         lst = [] # On crée la future ligne
6         for i in j: # Parcours des différents éléments de la colonne (donc une ligne)
7             teinte_de_gris = i[0] * 0.2125 + i[1] * 0.7174 + i[
8                 2] * 0.0721 # On remplace RGB par une teinte de gris (0,255)
9             if teinte_de_gris < teinte: # Si la teinte de gris est trop foncée, on la remplace par du noir
10                teinte_de_gris = 0
11            lst.append(teinte_de_gris) # On met dans la ligne les pixels en teinte gris ou noir
12        Original_2.append(lst) # On ajoute la ligne à la j ième colonne
13    return Original_2

1 def Contour(Image):
2     Courbe = [[ ] for k in range(len(Image))] # Création de la nouvelle matrice
3     for i in range(len(Image[0])): # On parcourt d'abord les lignes
4         n = 0
5         for j in range(len(Image)): # Puis les colonnes
```

```
7         i] > 0: # Si sur la colonne j et j-3, sur une même ligne, il y a des points noirs...
8             Courbe[n].append(0) # on ajoute tous les points qui pourraient faire partie de la courbe
9         else:
10             Courbe[n].append(255) # et sinon, on laisse en blanc
11         n += 1
12     return Courbe
```

On va maintenant sortir une courbe de notre image

```
1 def Courbe(Image):
2     Image = [[Image[j][i] for j in range(len(Image))] for i in range(len(Image[0])) - 1, 1, -1]]
3     # On tourne la matrice parce qu'on va parcourir l'image ligne par ligne mais comme on veut repérer le 1er point
4     # sur chaque colonne, on tourne la matrice de 90 degrés
5     X, Y = [], [] # On crée la liste de nos coordonnées
6     coordX = -1 # On commence à l'origine x=0
7     for i in Image: # On parcourt les colonnes (là ce sont les lignes de l'image d'au-dessus)
8         coordX += 1 # On incrémente X
9         lst = []
10        coordY = -1
11        for j in i: # On parcourt les lignes
12            coordY += 1
13            if j == 0: # On prend tous les points noirs
14                lst.append(coordY)
15            if len(lst) != 0: # Si il y a des points noirs sur une ligne,
16                coordY = lst[0] # on prend le 1er de ces éléments
17                X.append(coordX)
18                Y.append(-coordY)
19    return X, Y
```

On trouve l'angle

```
1 def Intersection_droite_cercle(m, b, x0, y0, r, x_list, y_list):
2     c1 = 1 + m ** 2
3     c2 = - 2.0 * x0 + 2 * m * (b - y0)
```

```
4     c3 = x0 ** 2 + (b - y0) ** 2 - r ** 2
5
6     delta = c2 ** 2 - 4.0 * c1 * c3 # On résout l'équation différentielle
7
8     x1, x2 = (- c2 + np.sqrt(delta)) / (2.0 * c1), (- c2 - np.sqrt(delta)) / (2.0 * c1)
9     y1, y2 = m * x1 + b, m * x2 + b
10
11    x_list.append(x1)
12    x_list.append(x2)
13    y_list.append(y1)
14    y_list.append(y2)
15    return None
16
17
18 def Angle(x, y, x0, y0):
19     num = x - x0
20     den = np.sqrt((x - x0) ** 2 + (y - y0) ** 2)
21     theta = np.arccos(num / den)
22     if not y - y0 >= 0: theta = 2 * np.pi - theta
23     return theta
```

On fait une régression linéaire des deux parties de la courbe et on trace les tangentes ainsi que le point d'intersection

```
1 def coeff_angle1(X, Y, K=int, x_list = [], y_list = [], theta_list = []): # K est un nombre entier qui dépend de la taille de l'image
2
3     # Régression linéaire du support
4     nb = len(X) // 2 # nombre de points mais on se restreint l'intervalle de recherche
5     max = 0
6     for i in range(K, nb):
7         a1, b1 = np.polyfit(X[i - K:i], Y[i - K:i], 1) # a et b sont les coefficients de la régression linéaire : y=ax+b
8         if a1 > max: # On cherche la pente la plus raide pour savoir où se situe le point anguleux
9             nbmax = i
10            max = a1
11
12     a1, b1 = np.polyfit(X[0:nbmax - K], Y[0:nbmax - K], 1) # On fait une régression linéaire sur le début de la courbe
13     # pour avoir l'équation de la droite qui vérifie le plan du support
14     Y_reg1 = [a1 * X[i] + b1 for i in range(len(X))] # On obtient notre droite
```

```
-- 
15
16 # Régression linéaire de la goutte
17 [a2, b2] = np.polyfit(X[nbmax - K:nbmax], Y[nbmax - K:nbmax], 1) # a et b sont les coefficients de la régression linéaire : y=ax
18 Y_reg2 = [a2 * X[i] + b2 for i in range(len(X))]
19
20 # Le point d'intersection
21 x0 = (b2 - b1) / (a1 - a2)
22 y0 = a1 * x0 + b1
23
24 # Recherche de l'angle : On va d'abord créer un cercle et chercher les points d'intersection de ce cercle avec nos droites
25
26 # Point intersection
27 Intersection_droite_cercle(a1, b1, x0, y0, K, x_list, y_list) # Avec K le rayon du cercle
28 Intersection_droite_cercle(a2, b2, x0, y0, K, x_list, y_list)
29
30 for i in range(len(x_list)):
31     theta_list.append(Angle(x_list[i], y_list[i], x0, y0))
32
33 # Angle
34 theta_1 = theta_list[0] # On choisit les deux angles qui nous intéressent
35 theta_2 = theta_list[2]
36 theta = np.linspace(theta_1, theta_2, 1000)
37 x1 = K * np.cos(theta) + x0 # Les coordonnées des deux points des intersections entre les droites et le cercle
38 x2 = K * np.sin(theta) + y0
39
40 mid_angle = (theta_1 + theta_2) / 2.0 # Pour l'affichage du texte
41 mid_angle_x = (K + 0.45) * np.cos(mid_angle) + x0
42 mid_angle_y = (K + 0.45) * np.sin(mid_angle) + y0
43
44 valeur_angle = round(np.rad2deg(abs(theta_1 - theta_2)), 2)
45
46 return Y_reg1, Y_reg2, theta_list, mid_angle_x, mid_angle_y, valeur_angle, x1, x2
```

Paramétrage

```
1 drive.mount('/content/drive') # On sélectionne notre dossier
2 file = '/content/drive/MyDrive/PC/TIPE spé/Images/10.jpg' # On importe notre image
3 Original = cv2.imread(file)

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

1 # On redimensionne
2 Taille = 2000
3 CoordX = 1200 # Point de départ en X
4 CoordY = 500 # Point de départ en Y
5 Original_resized = Original[CoordY:CoordY + Taille // 2, CoordX:CoordX + Taille]
6
7 # Teinte
8 teinte = 70
9 img_noir_blanck = NoirBlanc(Original_resized, teinte)
10
11 # Pour le tracé de la figure
12 X, Y = Courbe(Contour(img_noir_blanck))
13 K = 40 # Dépend de la taille de l'image
14 Y_reg1, Y_reg2, theta_list, mid_angle_x, mid_angle_y, valeur_angle, x1, x2 = coeff_angle1(X, Y, K)
```

Résultats !

```
1 # Originale
2 plt.figure()
3 plt.subplot(3, 2, 1)
4 plt.imshow(Original)
5 # Avec le bon cadrage
6 plt.subplot(3, 2, 2)
7 plt.imshow(Original_resized)
8 # En noir et blanc
9 plt.subplot(3, 2, 3)
10 plt.imshow(img_noir_blanck, cmap='gray', vmin=0, vmax=255)
11 # Avec les contours
12 plt.subplot(3, 2, 4)
```

```
12 plt.show()
13 plt.imshow(Contour(img_noir_blan), cmap="gray")
14 # La courbe que forme la goutte
15 plt.subplot(3, 2, 5)
16 plt.grid()
17 plt.plot(X, Y)
18 # La courbe que forme la goutte accompagnée des tangentes et de l'angle formé
19 plt.subplot(3, 2, 6)
20 plt.grid()
21 plt.plot(X, Y)
22 plt.plot(X, Y_reg1, 'b')
23 plt.plot(X, Y_reg2, 'r')
24 plt.ylim(-550, -200)
25 plt.xlim(600, 1400)
26 plt.plot(x1, x2, color='gray')
27 plt.text(mid_angle_x, mid_angle_y, valeur_angle, fontsize=8)
28 plt.show()
```

