Repérage des constellations à partir d'une photographie prise par un télescope

Problématique: Comment localiser à partir d'une photographie et à l'aide d'une base de données sur les étoiles, les différentes constellations ?

Plan

- Résumé de l’étude réalisée par mon coéquipier
- Introduction
- Analyse des étoiles plus lumineuses
- Utilisation de base de données SQL
- Lien SQL / Photographie
- Affichage en pyplot
- Conclusion / Améliorations envisageables
- Problèmes dûs à la pollution lumineuse

De quelle manière peut-on se procurer une image nette du ciel dans le but de localiser, à l'aide d'une base de donnée sur les étoiles, les différents constellations ?

Les différents outils utilisés dans cette étude…

- lunettes astronomiques
- télescopes

…. Et les différents problèmes qui sont rencontrés:

- aberration chromatique
- résolution de la photo
- gêne de l'atmosphère → essayer de se placer au delà de l'atmosphère
- pollution lumineuse

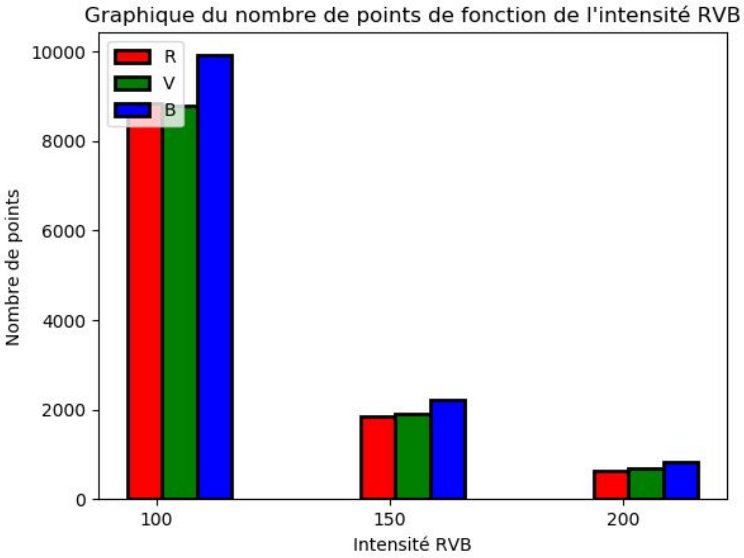
Comment déterminer efficacement ces constellations?

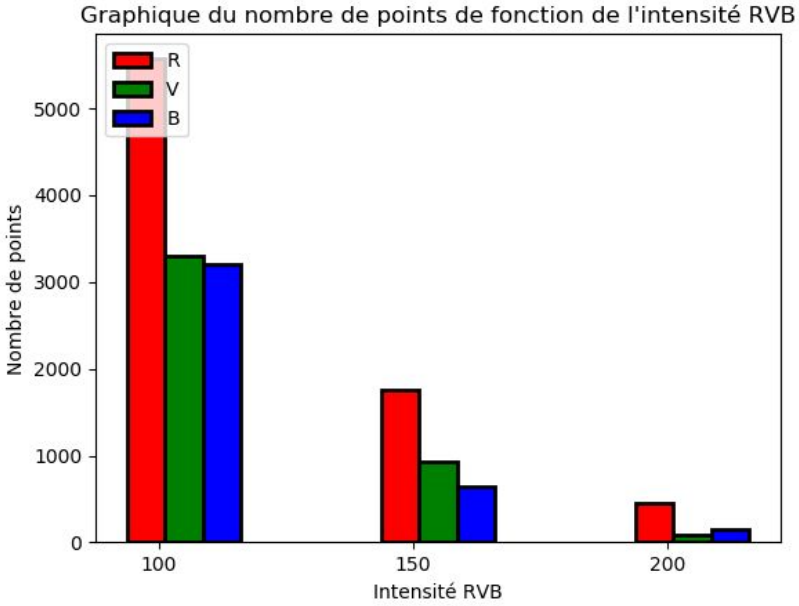
Constellation A

Constellation B

Etude du seuillage

Constellation A, seuillage : 207





Constellation B, seuillage : 195

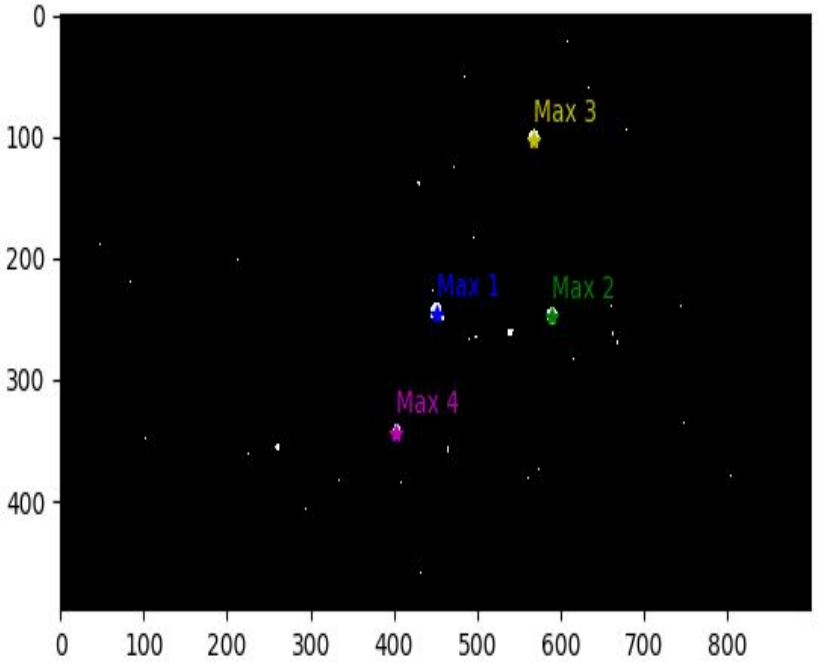
Programme de l'étude des maximums

L: liste de coordonnées supérieures au seuil
intervalle: déterminé à partir de la taille de l'image

- coordonnées point isolé
- nombre de pixels dans l'intervalle autour du point
- somme des coordonnées de x et de y

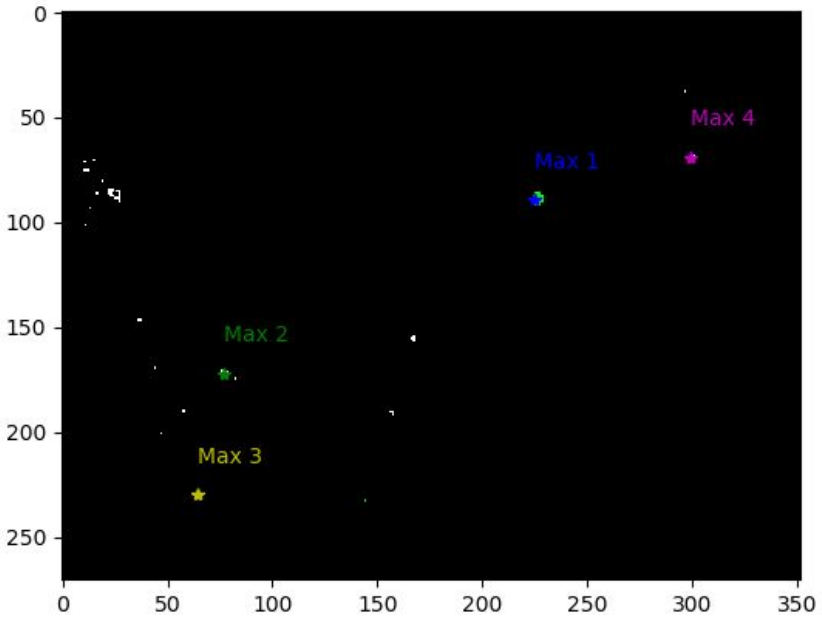
```
def pixel (L, intervalle) :
    pixel_iso_1 = [[L[1], 1, [L[1][0]], [L[1][1]]]]
    for k in L[1:]:
        j = True
        for i in pixel_iso_1:
            if math.sqrt((i[0][0] - k[0]) ** 2 + (i[0][1] - k[1]) ** 2) < intervalle :
                j = False
                i[1] += 1
                i[2].append(k[0])
                i[3].append(k[1])
        if j:
            pixel_iso_1.append([k, 1, [k[0]], [k[1]]])
```

Détermination des points les plus éclairés



Constellation A

Constellation B



Création de la base de données SQL

Tables

constellation
etoiles

Table “constellation”

nom	id
aigle	1
lyre	2
cygne	3
bouvier	4
cocher	5
cassiopee	6
scorpion	7
grande_ourse	8

Table “etoile”

Exemple pour la constellation Cassiopée:

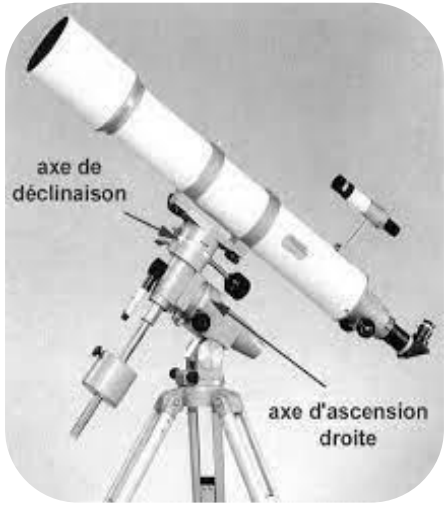
id	nom	constellation	dec	asc	lum
8	gamma	6	60.69	14.18	1
6	alpha	6	56.52	10.13	2
7	beta	6	59.14	2.30	3
12	delta	6	60.23	21.45	4
13	epsilon	6	63.64	28.60	5

dec: déclinaison **asc: ascension droite** **lum: luminescence**

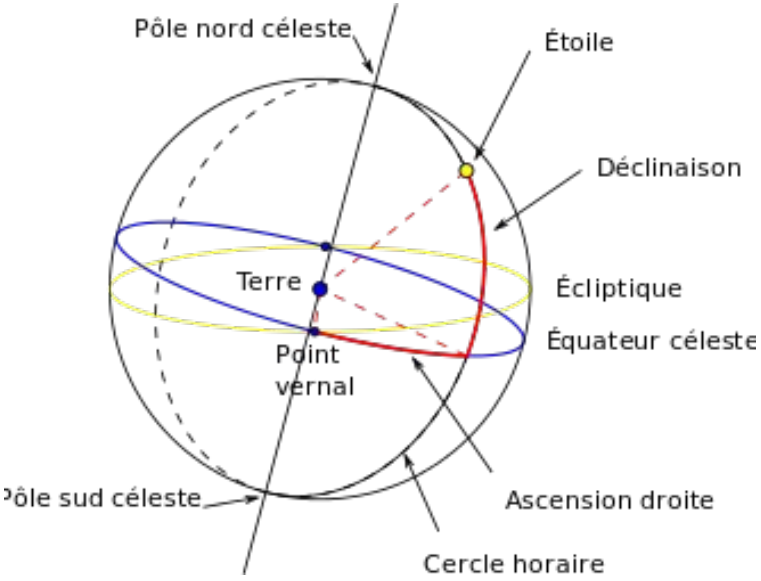
Détermination des angles

Conversion exemple:

- Ascension droite (entre 0 et 24h): 00h 56m 42.50s
  → 14.18°
- Déclinaison (entre + 90° et -90°): + 60° 40’ 00.3”
  → 60.68°



https://fr.wikipedia.org/wiki/Syst%C3%A8me_de_coordonn%C3%A9es_%C3%A9quatoriales#/media/Fichier:Coordonnees_equatoriales.svg

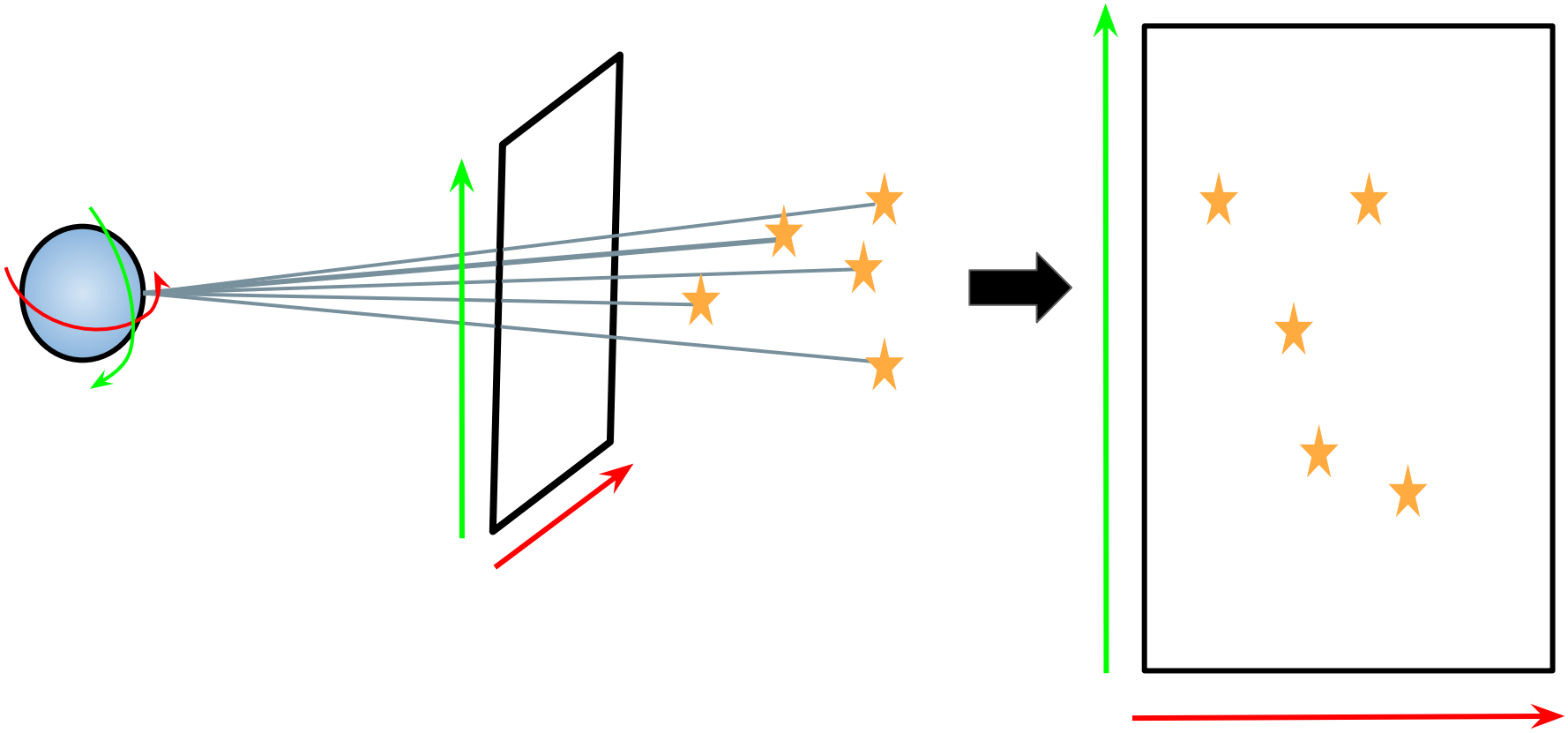


http://www.astrosurf.com/toussaint/dossiers/coordonnees/coordonnees.htm

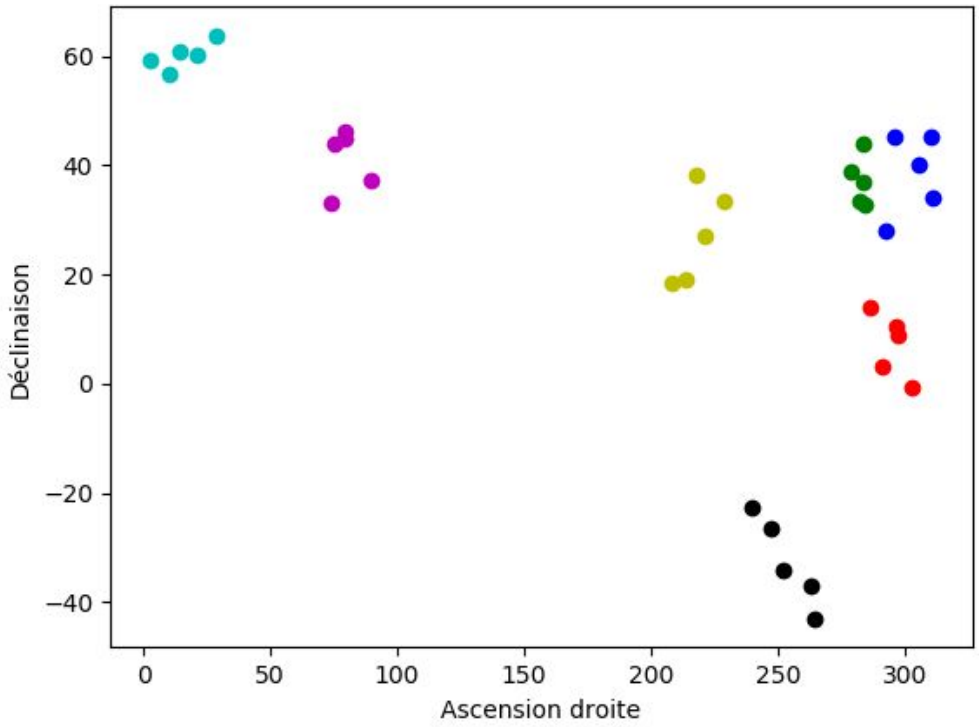
Représentation plane

Ascension droite

Déclinaison



Définition de la base angulaire des coordonnées de SQL

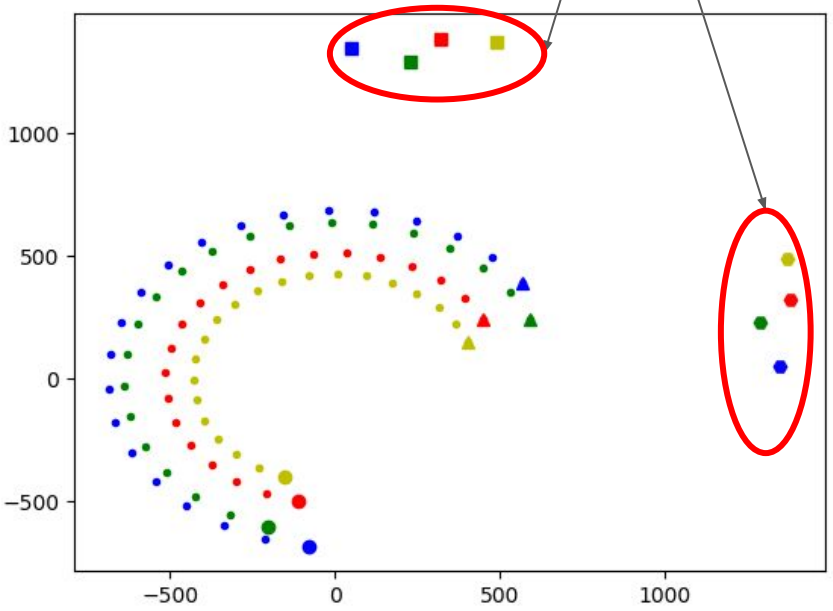


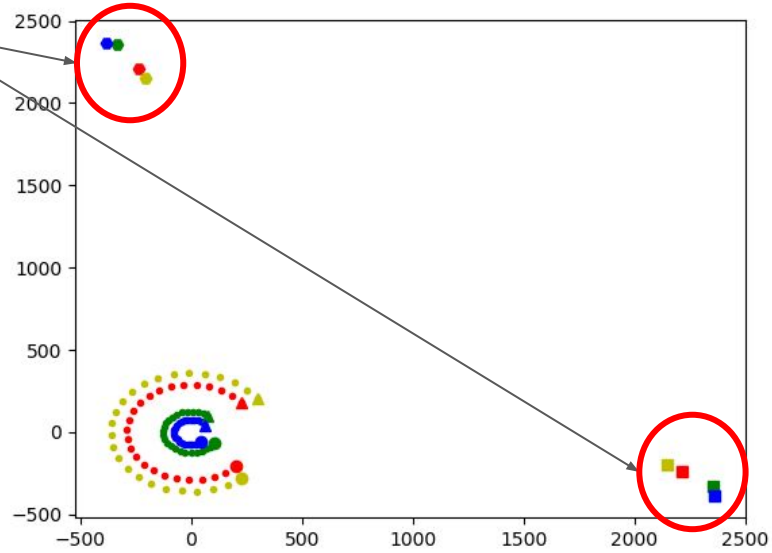
- Aigle
- Lyre
- Cygne
- Bouvier
- Cocher
- Cassiopée
- Scorpion

Lien entre les 2 bases de coordonnées

Coordonnées des étoiles en SQL

Constellation A





Constellation B

Programme de liaison

```
for k in range(len(constellation)):
    while (0.999 > kx / ky or 1.001 < kx / ky) or kx < 0:
        x1 = np.cos(i) * p1[0] + np.sin(i) * p1[1]
        y1 = np.cos(i) * p1[1] - np.sin(i) * p1[0]
        x2, y2 = …
        x3, y3 = …
        x4, y4 = …
        v1p = (x2 - x1, y2 - y1)
        v2p = …
        v3p = …
        kx = v1p[0] / v1[0]
        ky = v1p[1] / v1[1]
        i += 0.001
        …

        if abs(theta2) < 0.4 and abs(theta3) < 0.4:
            return k + 1
```

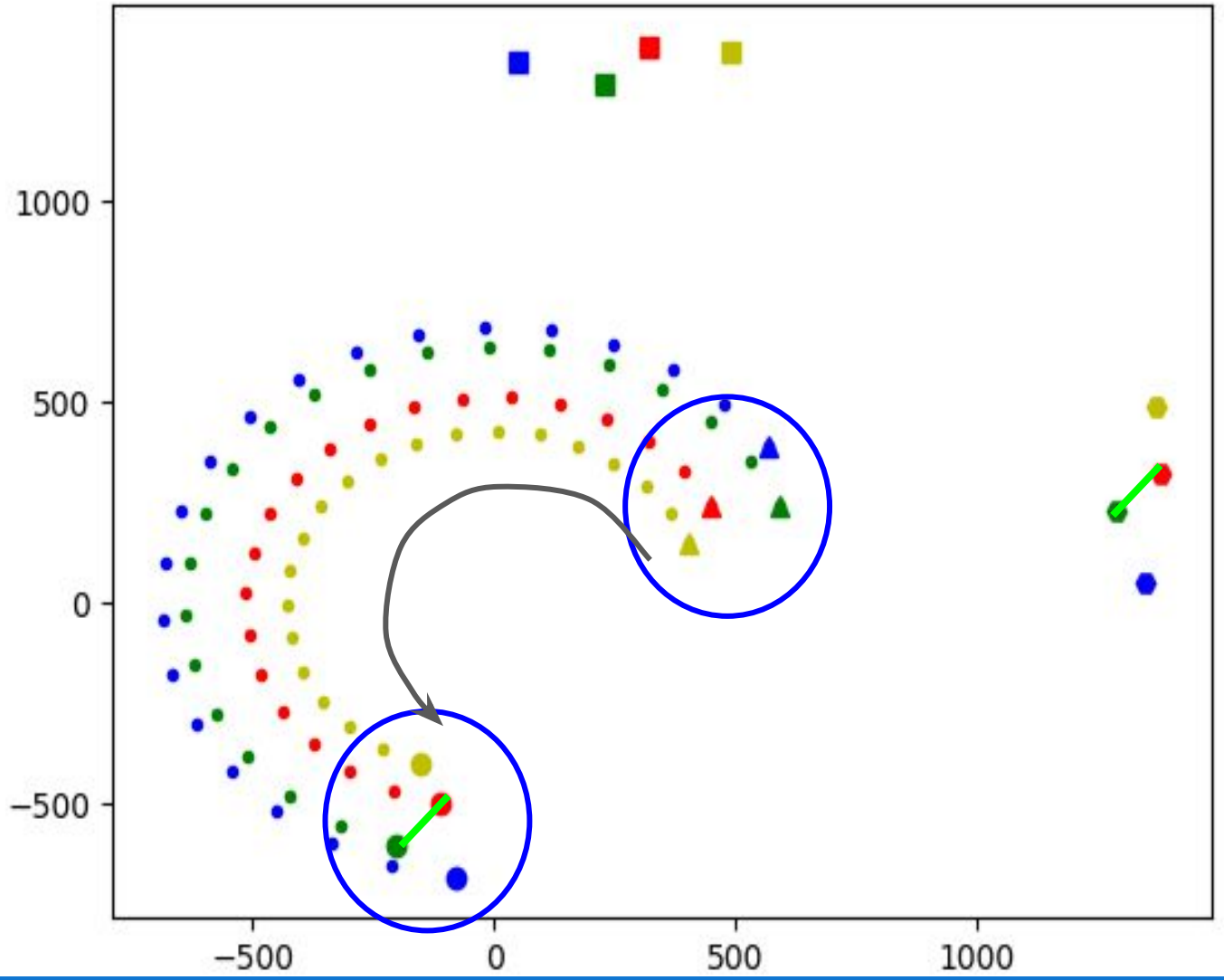
Constellation A

```
>>> (executing file "TIPE_presq_propre.py")
207
0.3305557484285003 1.2193382551389047
1.4841872700481817 1.032612576605162
1.0956181106750167 2.589623455883187
0.7191249078016887 0.3376726241390982
0.040703587163228895 1.2176741346265696
1.7740394313134547 2.8135603408089467
1.5640136250010228 2.553457554771222
2.905243356522917 1.478591613485333
0.024909396956333516 1.6485405363417724
1.789429104116649 2.3830984564974393
1.5780667378366444 1.9892976136837661
0.23667628064004131 0.26265321806030345
```

v1p, v2p, v3p sont les vecteurs obtenus à partir de l'image
v1, v2, v3 sont les vecteurs a partir de la base de données

Puis: produit scalaire de v2 et v2p, de v3 et v3p
> détermination de l'angle d'erreur (inférieur à 0.4 radians)

Exemple pour la constellation A:



Après la détection:

Constellation B : Scorpion / Constellation A : Cassiopee

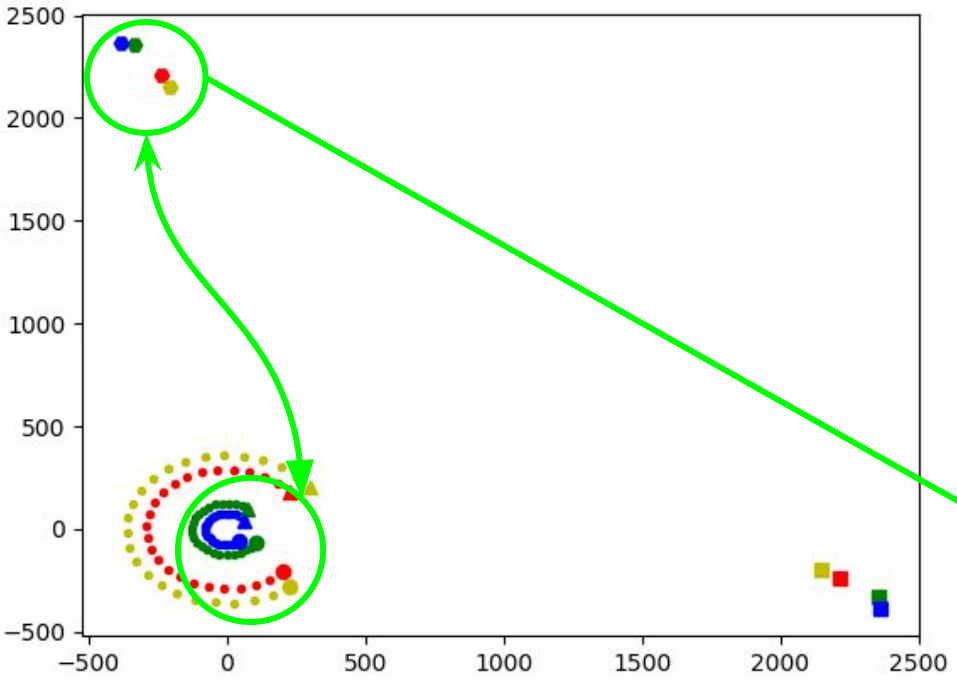
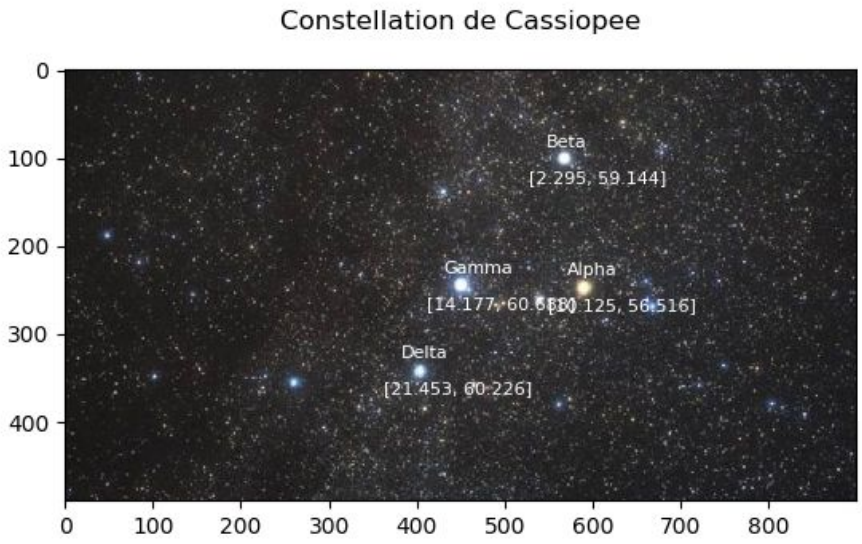


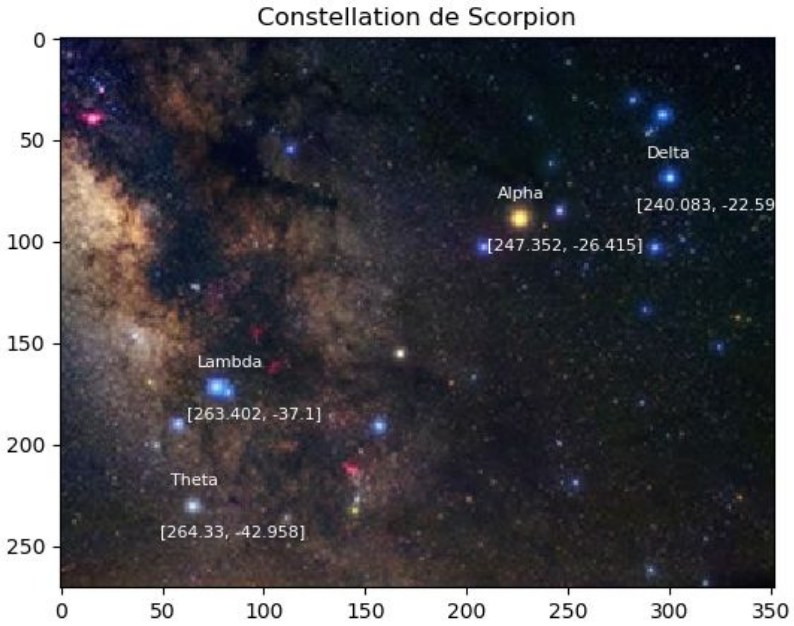
Table “constellation”

nom	id
aigle	1
lyre	2
cygne	3
bouvier	4
cocher	5
cassiopee	6
scorpion	7
grande_ourse	8

Affichage sur l'image initiale

Constellation A





Constellation B

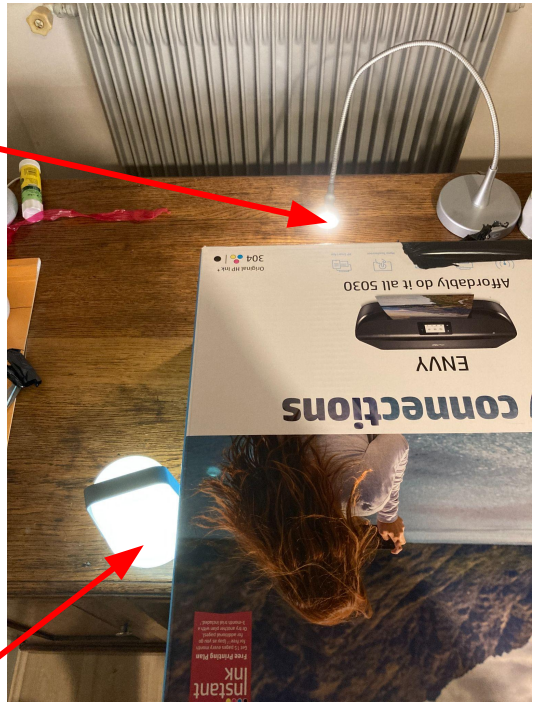
Améliorations possibles

- Être plus précis sur la reconnaissance des étoiles pour ne pas mélanger 2 constellations (exemple entre Aigle et Scorpion)
- Pour le seuillage, trouver directement une couleur adaptée pour le choix des maximums
- Gagner du temps sur l'analyse des images (plusieurs minutes)
- Ne pas analyser uniquement sur les angles mais aussi sur les distances séparant les différentes étoiles.

Expérience

Lumière (étoiles)

Fond percé de trous



Pollution lumineuse

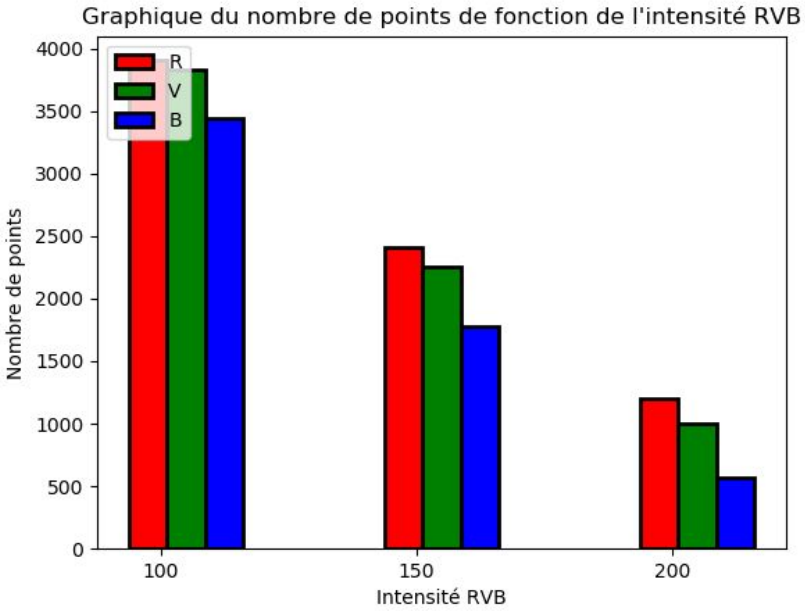
Résultats

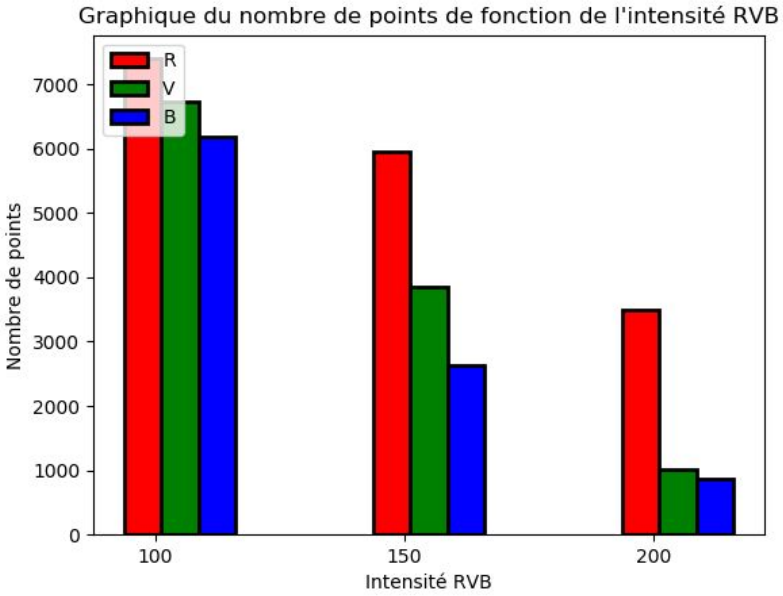
Sans pollution lumineuse

Avec pollution lumineuse

Affichage graphique

Sans pollution lumineuse





Avec pollution lumineuse

Annexes

```python
import math
import sqlite3
etoiles = sqlite3.connect('etoiles.db')
cursor = etoiles.cursor()
import numpy as np
import PIL
from PIL import Image
import matplotlib.pyplot as plt
```

```python
def transf(L1):
    for i in L1:
        i[0], i[1] = i[1], i[0]
    return L1
```

```python
def min(L):
    min = L[0]
    for i in L:
        if i < min:
            min = i
    return min

def max(L):
    max = L[0]
    for i in L:
        if i > max:
            max = i
    return max
```

```python
def extraction_image(nomFichier):
    '''
    retourne l'image sous forme de tableau
    '''
    img = Image.open(nomFichier)
    return  np.array(img)
```

```python
def est_blanc(pix):
    i,j  = pix
    return img[i,j,0]==img[i,j,1]==img[i,j,2]==255
```

```python
def est_dans_image(i0,j0,img):
    l, h, t = img.shape
    return 0 <= i0 < l and  0 <= j0 < h
```

Annexes: Extraction de l'image et analyse des points

```python
def det_seuil(img):
    a = 0
    b = 255
    nb_pix = 0
    l, h, t = img.shape
    while nb_pix > l * h / 800 or nb_pix < l * h / 1000:
        nb_pix = 0
        for i in range(l):
            for j in range(h):
                if img[i,j,1] >= (b + a) / 2:
                    nb_pix += 1
        if nb_pix > h * l / 600:
            a = (b + a) / 2
        else:
            b = (b + a) / 2
    return (a + b) / 2
```

```python
def image_mat(nom_fichier):
    img = Image.open(nom_fichier)
    return np.array(img)

img = image_mat("cass.png")
l, h, t = img.shape

def histoRVB(img):
    l, h, t = img.shape
    R, V, B = [], [], []
    for j in range(h):
        for i in range(l):
            R.append(img[i,j,0])
            V.append(img[i,j,1])
            B.append(img[i,j,2])
    return R, V, B

R, V, B = histoRVB(img)
```

```python
def seuil_rouge(img, seuil):
    l, h, t = img.shape
    rou = 0
    for i in range(l):
        for j in range(h):
            if img[i,j,0] > seuil:
                rou += 1
    return rou

def seuil_vert(img, seuil):
    l, h, t = img.shape
    ver = 0
    for i in range(l):
        for j in range(h):
            if img[i,j,1] > seuil:
                ver += 1
    return ver

def seuil_bleu(img, seuil):
    l, h, t = img.shape
    ble = 0
    for i in range(l):
        for j in range(h):
            if img[i,j,2] > seuil:
                ble += 1
    return ble
```

```
seuil1 = 200
seuil2 = 150
seuil3 = 100
abs_x = ['R', 'V', 'B']
abs_y1 = [seuil_rouge(img, seuil1), seuil_vert(img, seuil1), seuil_bleu(img, seuil1)]
abs_y2 = [seuil_rouge(img, seuil2), seuil_vert(img, seuil2), seuil_bleu(img, seuil2)]
abs_y3 = [seuil_rouge(img, seuil3), seuil_vert(img, seuil3), seuil_bleu(img, seuil3)]

repver = retour_vert(img)
reprou = retour_rouge(img)
repble = retour_bleu(img)


Categories = ["R","V","B"] # 3 échantillons
Sous_categories = ['100', '150', '200'] # comparés selon 3 critères
# Valeurs pour chaque catégories
y1 = [seuil_rouge(img, seuil3),seuil_rouge(img, seuil2),seuil_rouge(img, seuil1)] ; y2 = [seuil_vert(img, seuil3),seuil_vert(img, seuil2),seuil_vert(img,
seuil1)] ; y3 = [seuil_bleu(img, seuil3),seuil_bleu(img, seuil2),seuil_bleu(img, seuil1)]
nb_categories = len(Categories)
largeur_barre = floor(1*10/nb_categories)/20
x1 = range(len(y1))
x2 = [i + largeur_barre for i in x1]
x3 = [i + 2 * largeur_barre for i in x1]
plt.bar(x1, y1, width = largeur_barre, color = 'red',
          edgecolor = 'black', linewidth = 2)
plt.bar(x2, y2, width = largeur_barre, color = 'green',
          edgecolor = 'black', linewidth = 2)
plt.bar(x3, y3, width = largeur_barre, color = 'blue',
          edgecolor = 'black', linewidth = 2)
plt.xticks([r + largeur_barre / nb_categories for r in range(len(y1))],
           Sous_categories)
plt.legend(Categories,loc=2)
plt.xlabel("Intensité RVB")
plt.ylabel("Nombre de points")
plt.title("Graphique du nombre de points de fonction de l'intensité RVB")

plt.show()
```

Annexes: Affichage de l’histogramme

```python
def pixel (L, intervalle) :
    pixel_iso_1 = [[L[1], 1, [L[1][0]], [L[1][1]]]]
    for k in L[1:]:
        j = True
        for i in pixel_iso_1:
            if math.sqrt((i[0][0] - k[0]) ** 2 + (i[0][1] - k[1]) ** 2) < intervalle :
                j = False
                i[1] += 1
                i[2].append(k[0])
                i[3].append(k[1])
        if j:
            pixel_iso_1.append([k, 1, [k[0]], [k[1]]])
    pixel_iso = []
    for  k in pixel_iso_1:  # permet de centrer les points
        somx, somy = 0, 0
        long = len(k[2])
        for i in range(long):
            somx += k[2][i]
            somy += k[3][i]
        somx = somx // long
        somy = somy // long
        pixel_iso.append([[somx, somy], k[1]])
    pixel_iso_seul = []
    for k in pixel_iso:
        pixel_iso_seul.append(k[0])
    return pixel_iso_seul, pixel_iso
```

```python
def maxii(pixel1):
    Lmax = []
    for j in pixel1:
        Lmax.append(j[1])
    max1 = Lmax[0]
    ind1 = 0
    for i in range(len(Lmax)):
        if Lmax[i] > max1:
            max1 = Lmax[i]
            ind1 = i
    Lmax[ind1] = 0
    max2 = Lmax[0]
    ind2 = 0
    for i in range(len(Lmax)):
        if Lmax[i] > max2:
            max2 = Lmax[i]
            ind2 = i
    Lmax[ind2] = 0
    max3 = Lmax[0]
    ind3 = 0
    for i in range(len(Lmax)):
        if Lmax[i] > max3:
            max3 = Lmax[i]
            ind3 = i
    Lmax[ind3] = 0
    max4 = Lmax[0]
    ind4 = 0
    for i in range(len(Lmax)):
        if Lmax[i] > max4:
            max4 = Lmax[i]
            ind4 = i
```

```
### Evaluation des distances de SQL
def decasc(taille):
    donnees = []
    for i in range(1, taille + 1):
        cons = []
        cursor.execute("""SELECT COUNT (*) FROM etoile WHERE constellation = {}""".format(i))
        nbre = cursor.fetchall()[0][0]
        if nbre >= 3:
            for j in range(1,6):
                cursor.execute("""SELECT nom, asc, dec FROM etoile WHERE constellation = {} and lum = {}""".format(i,j))
                coor = cursor.fetchall()
                cons.append([coor[0][0],coor[0][1],coor[0][2]])
        donnees.append(cons)
    return donnees
```

```
list = ["r", "g", "b", "y", "m", "c", "k"]
for k in range(1,8):
    L = []
    M = []
    cursor.execute("""SELECT asc, dec, nom FROM etoile WHERE constellation = {}""".format(k))
    coor = cursor.fetchall()
    for i in range(len(coor)):
        plt.plot(coor[i][0],coor[i][1],  "o", color = list[k-1])
plt.xlabel("Ascension droite")
plt.ylabel("Déclinaison")
plt.show()
```

Annexes: Affichage des etoiles de la base SQL

```
def test_lien(donnees , l, p1 = M1, p2 = M2, p3= M3, p4 = M4):
    p1, p2, p3, p4 = [p1[0], l - p1[1]], [p2[0], l - p2[1]], [p3[0], l - p3[1]], [p4[0], l - p4[1]]
    for k in range(len(donnees)):
        if donnees[k] != []:
            M1 = [donnees[k][0][1], donnees[k][0][2]]
            M2 = [donnees[k][1][1], donnees[k][1][2]]
            M3 = [donnees[k][2][1], donnees[k][2][2]]
            M4 = [donnees[k][3][1], donnees[k][3][2]]
            i = 0
            k12x, k12y = 40, 1
            while (0.999 > k12x / k12y or 1.001<  k12x / k12y) or k12x < 0:
                x1, y1 = np.cos(i) * p1[0] + np.sin(i) * p1[1], np.cos(i) * p1[1] - np.sin(i) * p1[0]
                x2, y2 = np.cos(i) * p2[0] + np.sin(i) * p2[1], np.cos(i) * p2[1] - np.sin(i) * p2[0]
                x3, y3 = np.cos(i) * p3[0] + np.sin(i) * p3[1], np.cos(i) * p3[1] - np.sin(i) * p3[0]
                x4, y4 = np.cos(i) * p4[0] + np.sin(i) * p4[1], np.cos(i) * p4[1] - np.sin(i) * p4[0]
                v1 = (M2[0] - M1[0], M2[1] - M1[1])
                v2 = (M3[0] - M1[0], M3[1] - M1[1])
                v3 = (M4[0] - M1[0], M4[1] - M1[1])
                v1p = (x2 - x1, y2 - y1)
                v2p = (x3 - x1, y3 - y1)
                v3p = (x4 - x1, y4 - y1)
                k12x = v1p[0] / v1[0]
                k12y = v1p[1] / v1[1]
                i += 0.001
            n2 = np.sqrt(v2[0] ** 2 + v2[1] ** 2)
            n2p= np.sqrt(v2p[0] ** 2 + v2p[1] ** 2)
            n3 = np.sqrt(v3[0] ** 2 + v3[1] ** 2)
            n3p= np.sqrt(v3p[0] ** 2 + v3p[1] ** 2)
            ps2 = v2[0] * v2p[0] + v2[1] * v2p[1]
            ps3 = v3[0] * v3p[0] + v3[1] * v3p[1]
            theta2 = np.arccos(ps2 / (n2 * n2p))
            theta3 = np.arccos(ps3 / (n3 * n3p))
            print(theta2, theta3)
            if abs(theta2) < 0.4 and abs(theta3) < 0.4:
                return k +1
```

```
        while (0.999 > k12x / k12y or 1.001<  k12x / k12y) or k12x < 0:
            x1, y1 = np.cos(i) * p1[0] + np.sin(i) * p1[1], np.cos(i) * p1[1] - np.sin(i) * p1[0]
            x2, y2 = np.cos(i) * p2[0] + np.sin(i) * p2[1], np.cos(i) * p2[1] - np.sin(i) * p2[0]
            x3, y3 = np.cos(i) * p3[0] + np.sin(i) * p3[1], np.cos(i) * p3[1] - np.sin(i) * p3[0]
            x4, y4 = np.cos(i) * p4[0] + np.sin(i) * p4[1], np.cos(i) * p4[1] - np.sin(i) * p4[0]
            v1 = (M2[1] - M1[1], M2[0] - M1[0])
            v2 = (M3[1] - M1[1], M3[0] - M1[0])
            v3 = (M4[1] - M1[1], M4[0] - M1[0])
            v1p = (x2 - x1, y2 - y1)
            v2p = (x3 - x1, y3 - y1)
            v3p = (x4 - x1, y4 - y1)
            k12x = v1p[0] / v1[0]
            k12y = v1p[1] / v1[1]
            i += 0.001
        n2 = np.sqrt(v2[0] ** 2 + v2[1] ** 2)
        n2p= np.sqrt(v2p[0] ** 2 + v2p[1] ** 2)
        n3 = np.sqrt(v3[0] ** 2 + v3[1] ** 2)
        n3p= np.sqrt(v3p[0] ** 2 + v3p[1] ** 2)
        ps2 = v2[0] * v2p[0] + v2[1] * v2p[1]
        ps3 = v3[0] * v3p[0] + v3[1] * v3p[1]
        theta2 = np.arccos(ps2 / (n2 * n2p))
        theta3 = np.arccos(ps3 / (n3 * n3p))
        print(theta2, theta3)
        if abs(theta2) < 0.4 and abs(theta3) < 0.4:
            return k +1
```

```
241 ### Donnees
242 cursor.execute("""SELECT COUNT (*) FROM constellation; """)
243 taille = cursor.fetchall()[0][0]
244 donnees = decasc(taille)
245 c = test_lien(donnees, l, M1, M2, M3, M4)
```

```
126 #### Données
127 img, img_init = extraction_image('cass.png'), extraction_image('cass.png')
128 l, h, t = img.shape
129 seuil = int(det_seuil(img))
130 print(seuil)
131 img = seuillage(img,seuil)[0]
132 L = seuillage(img,seuil)[1]
133 intervalle = int(l / 30)
134 p = pixel(L,intervalle)[0]
135 p1 = pixel(L,intervalle)[1]
136 maxim = maxii(p1)
137 M = transf(maxim)
```

```
139 ### Evaluation des distances sur image
140 M1 = M[0]
141 M2 = M[1]
142 M3 = M[2]
143 M4 = M[3]
```

```
## Affichage de l'image
cursor.execute("""SELECT nom, dec, asc FROM etoile WHERE constellation = {} ORDER BY lum ASC; """.format(c))
order = cursor.fetchall()
cursor.execute("""SELECT nom FROM constellation WHERE id = {};""".format(c))
const = cursor.fetchall()[0][0]
noms = [order[0][0].capitalize(), order[1][0].capitalize(), order[2][0].capitalize(), order[3][0].capitalize()]
coord = [[round(order[0][2],3), round(order[0][1], 3)], [round(order[1][2],3), round(order[1][1], 3)], [ round(order[2][2],3), round(order[2][1], 3)], [
round(order[3][2],3), round(order[3][1], 3)]]
axex = [M1[0], M2[0], M3[0], M4[0]]
axey = [M1[1], M2[1], M3[1], M4[1]]
plt.imshow(img_init)

for k in range(len(axex)):
   plt.text(axex[k] - 10, axey[k] - 10, noms[k], fontsize=8, color='w')
   plt.text(axex[k] - 15, axey[k] + 15, coord[k], fontsize=8, color='w')
plt.title("Constellation de {}".format(const.capitalize()))

plt.show()
```

```
h = 271
k12x = 40
k12y = 1
i = 0
M1 = [225, 89]
M2 = [77, 172]
M3 = [64, 230]
M4 = [299, 69]
M1p = [247.35, -26.41]
M2p = [263.40, -37.1]
M3p = [264.33, -43.0]
M4p = [240.08, -22.60]
```

```
n2 = np.sqrt(v2[0] ** 2 + v2[1] ** 2)
n2p= np.sqrt(v2p[0] ** 2 + v2p[1] ** 2)
n3 = np.sqrt(v3[0] ** 2 + v3[1] ** 2)
n3p= np.sqrt(v3p[0] ** 2 + v3p[1] ** 2)
ps2 = v2[0] * v2p[0] + v2[1] * v2p[1]
ps3 = v3[0] * v3p[0] + v3[1] * v3p[1]
theta2 = np.arccos(ps2 / (n2 * n2p))
theta3 = np.arccos(ps3 / (n3 * n3p))
plt.plot(k12x * M1p[0], k12x * M1p[1], marker = "s",color = 'r' )
plt.plot(k12x * M2p[0], k12x * M2p[1], marker = "s",color = 'g' )
plt.plot(k12x * M3p[0], k12x * M3p[1], marker = "s",color = 'b' )
plt.plot(k12x * M4p[0], k12x * M4p[1], marker = "s",color = 'y' )
plt.plot(k12x * M1p[1], k12x * M1p[0], marker = "H",color = 'r' )
plt.plot(k12x * M2p[1], k12x * M2p[0], marker = "H",color = 'g' )
plt.plot(k12x * M3p[1], k12x * M3p[0], marker = "H",color = 'b' )
plt.plot(k12x * M4p[1], k12x * M4p[0], marker = "H",color = 'y' )
plt.plot(x1, y1, marker = "o",color = 'r' )
plt.plot(x2, y2, marker = "o",color = 'g' )
plt.plot(x3, y3, marker = "o",color = 'b' )
plt.plot(x4, y4, marker = "o",color = 'y' )
plt.plot(M1[0], h - M1[1], marker = "^",color = 'r' )
plt.plot(M2[0], h - M2[1], marker = "^",color = 'g' )
plt.plot(M3[0], h - M3[1], marker = "^",color = 'b' )
plt.plot(M4[0], h - M4[1], marker = "^",color = 'y' )



plt.show()
```

```
while (0.9 > k12x / k12y or 1.1<  k12x / k12y) or k12x < 0:
    x1, y1 = np.cos(i) * M1[0] - np.sin(i) * (h - M1[1]), np.cos(i) * (h - M1[1]) + np.sin(i) * M1[0]
    x2, y2 = np.cos(i) * M2[0] - np.sin(i) * (h - M2[1]), np.cos(i) * (h - M2[1]) + np.sin(i) * M2[0]
    x3, y3 = np.cos(i) * M3[0] - np.sin(i) * (h - M3[1]), np.cos(i) * (h - M3[1]) + np.sin(i) * M3[0]
    x4, y4 = np.cos(i) * M4[0] - np.sin(i) * (h - M4[1]), np.cos(i) * (h - M4[1]) + np.sin(i) * M4[0]
    v1 = (M2p[1] - M1p[1], M2p[0] - M1p[0])
    v2 = (M3p[1] - M1p[1], M3p[0] - M1p[0])
    v3 = (M4p[1] - M1p[1], M4p[0] - M1p[0])
    v1p = (x2 - x1, y2 - y1)
    v2p = (x3 - x1, y3 - y1)
    v3p = (x4 - x1, y4 - y1)
    k12x = v1p[0] / v1[0]
    k12y = v1p[1] / v1[1]
    plt.plot(x1, y1, marker = ".",color = 'r' )
    plt.plot(x2, y2, marker = ".",color = 'g' )
    plt.plot(x3, y3, marker = ".",color = 'b' )
    plt.plot(x4, y4, marker = ".",color = 'y' )
    i += 0.2
```

Annexes: Affichage de la matrice rotation