

Documentation Librairie Python

DOBOT Magician :

Cette librairie a pour but de pouvoir contrôler le bras robot DOBOT Magician, la caméra rattachée à ce dernier, ainsi que la calibration entre les deux.

De plus, elle offre une solution clé en main pour les développeurs, chercheurs et étudiants souhaitant créer des applications de vision robotique (comme le tri d'objets, le suivi de trajectoire visuel ou le contrôle qualité). Grâce à une abstraction des commandes séries et des flux vidéo, vous pouvez vous concentrer sur la logique de votre application plutôt que sur la communication matérielle.

Pour cela, vous avez besoin d'un bras robot DOBOT Magician, d'une caméra, d'un ordinateur avec un IDE Python et de la librairie DobotVision.

Table des matières :

Préparation matérielle et calibration du robot et de la caméra :.....	3
Exemple d'initialisation :.....	5
Procédure d'extinction :.....	6
Librairie dobot-vision :.....	7

Préparation matérielle et calibration du robot et de la caméra :

Dans le cadre de l'initialisation du robot et de la caméra, il faut :

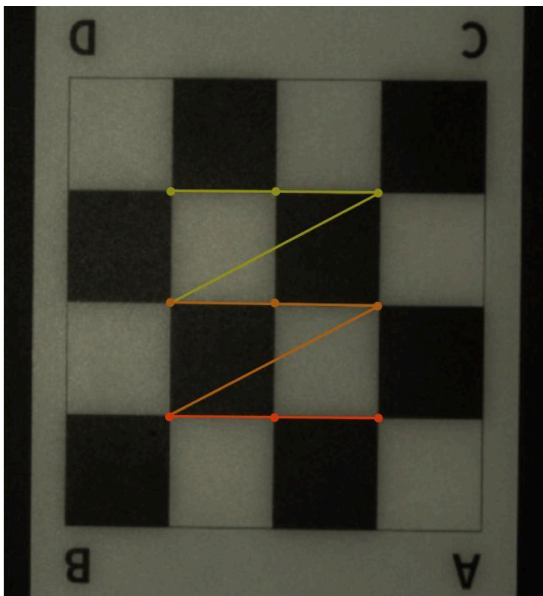
- Branchez la caméra et le bras robot DOBOT Magician au PC via les ports USB.
- Positionnez le damier sur le plateau sous la caméra.
- Importez la librairie DobotVision.

```
from DobotVision import *
```

- Assurez-vous d'avoir une lumière correcte, et lancer la fonction capture(image.jpg).

```
>>> capture("Image_plateau.jpg")  
'Image_plateau.jpg'
```

- Appliquez la fonction detect() à votre "Image.jpg".



```
>>> detect("Image_plateau.jpg")
```

(Gardez en tête l'ordre des 9 points, du rouge vert le jaune. Soit ci-dessus, de la droite vers la gauche, et du bas vers le haut)

- Allumez le robot, attendez le premier bip sonore. Puis, lancez la connexion entre le robot et le PC avec init() et patientez jusqu'au second bip sonore.
- Exécutez la fonction calib().



- Déplacez le bras au 9 points pour la calibration. Pour faire cela, maintenant le bouton surligné en rouge ci-dessus, déplacez le robot à la position désirée, relâchez le bouton et appuyez sur espace. Recommencez ensuite l'étape sur les 8 autres points, toujours dans le même ordre.
- Exécutez la fonction `go_home()` pour ramener le robot en position ($x=125$, $y=0$, $z=0$)
- Calculez la matrice d'homographie H en exécutant la fonction `homographie()`. Cette matrice permet de traduire les coordonnées en pixels vers des coordonnées en robot (de px en mm) et inversement.

Une fois ces étapes terminées, ne surtout pas déplacer la caméra ou tout devrait être recommencé. Le robot DOBOT Magician et la caméra sont maintenant calibrés entre eux.

A la suite de cette préparation, vous devriez pouvoir retrouver dans vos fichiers (au même endroit où se situe votre code Python) :

- Une image JPEG sous le nom de "Image.jpg" avec une photo du damier.
- Un fichier JSON "ValeursC" avec les coordonnées des 9 points du damier en px.
- Un fichier JSON "ValeursR" avec les coordonnées des 9 points du damier en mm.
- Un fichier JSON "H" avec la matrice d'homographie.

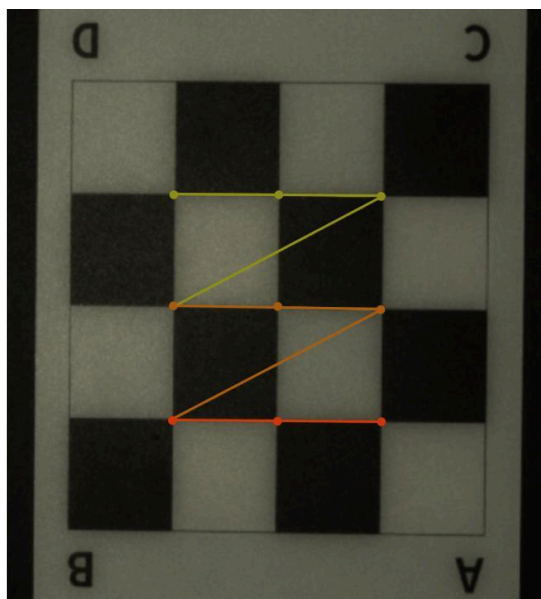
Note importante : si le robot reste trop longtemps sur une position au-dessus du plateau autre que la position neutre ($125, 0, 0$), il y a un risque que ce dernier réinitialise ses axes. Il est recommandé de constamment faire "`go_home()`" après une manipulation pour éviter cela. Si vous remarquez que les axes du robot changent, la calibration est alors à refaire.

Exemple d'initialisation :

Branchez le robot Dobot et la caméra au PC, et allumez le robot.

```
1 from calculs import *
2 from robot import *
3 from camera import *
4
5 capture(im = "Damier.jpg", bool = False, tps = 500000)
6 detect("Damier.jpg", nom_fichier = "ValeursC")
7
8 robot = RobotDobotVision()
9 robot.init()
10 robot.calib(nom_fichier = "ValeursR")
11
12 homographie(robot, "ValeursC", "ValeursR")
```

Après avoir importé la librairie (___), compilez ce code ci-dessus.



Lorsque cette image s'ouvre, observez les lignes, du rouge, vers le orange, puis vers le jaune. Fermez cette fenêtre. Dans cet ordre là, guidez le robot sur chacun des 9 points, et appuyez sur "espace" pour sauvegarder les coordonnées de calibration.

Une fois réalisé, la matrice d'homographie se calcule toute seule, et l'initialisation est terminée, vous pouvez retirer le plateau.

Procédure d'extinction :

Pour mettre fin au code, il suffit d'utiliser la fonction `eteindre()` :

```
>>> robot.eteindre()
```

Une fois la connexion coupée, vous pouvez débrancher le matériel et l'éteindre manuellement.

Librairie dobot-vision :

Les fonctions suivantes sont fournies dans ce module :

Fonctions DOBOT Magician	
bot.init(port = "COM3")	Prends en entrée une chaîne de caractères : - port : port de la connexion entre le robot et le PC Renvoie en sortie une chaîne de caractères. Initialise la connexion sur le port COM3 (port normalement par défaut lorsque l'on branche le DOBOT Magician au PC. Si le port est différent, appeler la fonction init() sur cet autre port. Ce processus prend environ 5 secondes. <pre>>>> robot = RobotDobotVision() >>> robot.init() Clearing alarms: 0. Robot connecté sur COM3 1</pre>
bot.go_home()	Ne prends aucun paramètre en entrée. Ne renvoie aucun paramètre en sortie. Ramène le DOBOT Magician bot à une position quasi-initiale par rapport à sa position lors de l'allumage du robot (x=125, y=0, z=0) Permet de dégager le plateau du bras robot. <pre>>>> robot.go_home() Position(x=125.0, y=0.0, z=1.52587890625e-05, r=0.0)</pre>
bot.go_to(x, y, z)	Prends en entrée trois entiers : - x : coordonnée x souhaitée - y : coordonnée y souhaitée - z : coordonnée z souhaitée Ne renvoie aucun paramètre en sortie. La fonction déplace le robot aux coordonnées x, y et z. La position renvoyée était sa position avant le déplacement. <pre>>>> robot.go_to(200, 100, 35) Position(x=104.83132934570312, y=0.0, z=-5.678955078125, r=0.0)</pre>

bot.position()	Ne prends aucun paramètre en entrée. Renvoie en sortie une liste Affiche les coordonnées courantes du robot par rapport à sa position de départ sous la forme d'un tableau de string. <pre>>>> robot1.position() ['x = -77', 'y = 133', 'z = 0']</pre>
bot.calib(nom_fichier = "ValeursR")	Prends en entrée une chaîne de caractères : - nom_fichier : le nom que vous souhaitez donner au fichier de sauvegarde de la calibration. Ne renvoie aucun paramètre en sortie. Lance la fonction pour calibrer le robot a chacun des 9 points du damier. Une fois le robot positionné sur un point, appuyer sur espace pour garder en mémoire la position (x, y) du robot, stockée dans le fichier JSON "ValeursR". <pre>>>> robot.calib() [(170, 203)] [(170, 203), (157, 182)] [(170, 203), (157, 182), (142, 161)] [(170, 203), (157, 182), (142, 161), (149, 218)] [(170, 203), (157, 182), (142, 161), (149, 218), (135, 198)] [(170, 203), (157, 182), (142, 161), (149, 218), (135, 198), (122, 176)] [(170, 203), (157, 182), (142, 161), (149, 218), (135, 198), (122, 176)] [(170, 203), (157, 182), (142, 161), (149, 218), (135, 198), (122, 176)] [(170, 203), (157, 182), (142, 161), (149, 218), (135, 198), (122, 176)]</pre> <pre>[[223, 127], [204, 113], [185, 99], [211, 149], [191, 135], [170, 120], [196, 168], [175, 155], [153, 139]]</pre>
bot.eteindre()	Ne prends aucun paramètre en entrée. Ne renvoie aucun paramètre en sortie. Coupe la connexion entre le robot et le PC. Cette fonction permet de limiter les erreurs lors d'une future connexion entre le robot et le PC. <pre>>>> robot.eteindre() Connexion coupé avec le robot sur COM3 1</pre>

bot.rotation(angle)	Prends en entrée un entier : - angle : la valeur de l'angle de rotation de la ventouse Ne renvoie aucun paramètre en sortie. Pivote la ventouse à l'extrémité du bras robot de angle degrés, dans un intervalle compris entre [-90, 90]. Dépasser cet intervalle empêchera la ventouse de pivoter. <pre>>>> robot.rotation(30) Position(x=151.26632690429688, y=0.0, z=2.5797271728515625, r=0.0)</pre>
bot.ventouse(bool)	Prends en entrée un booléen : - bool : vaut True ou False, allume ou éteint l'aspiration de la ventouse Ne renvoie aucun paramètre en sortie. Permet d'allumer ou éteindre l'aspiration de la ventouse en fonction de bool = True (pour allumer) et bool = False (pour éteindre). <pre>>>> robot.ventouse(True) >>> robot.ventouse(False)</pre>
bot.ramasser(cx, cy, cz)	Prends en entrée trois entiers : - cx : coordonnée x souhaitée de l'objet en question - cy : coordonnée y souhaitée de l'objet en question - cz : hauteur de l'objet z souhaitée de l'objet en question Ne renvoie aucun paramètre en sortie. Déplace le bras aux coordonnées cx, cy, avant de s'attacher à l'objet avec la ventouse, et de remonter jusqu'à ce que z soit égal à 0. La ventouse restera allumée une fois la fonction terminée. cz est la hauteur de l'objet. D'un robot à un autre, il est possible que cette hauteur varie. <pre>>>> robot.ramasser(180, 50, -30) Position(x=151.26632690429688, y=0.0, z=2.5797271728515625, r=0.0) Position(x=180.00001525878906, y=50.000003814697266, z=7.62939453125) Position(x=180.0, y=49.999996185302734, z=-30.00000762939453, r=0.0)</pre>

Fonctions Caméra

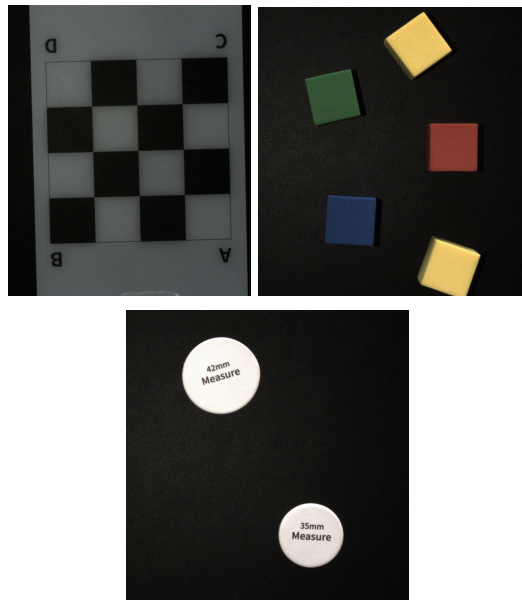
Prends en entrée une chaîne de caractères, un booléen et un entier :

- im : nom du fichier souhaité pour la capture. Ne pas oublier ".jpg" à la fin de ce dernier.
- bool : booléen permettant d'afficher la photo prise ou non.
- tps : temps d'exposition de la caméra en ms.

Renvoie une liste numpy en sortie.

Prends une photo du damier depuis la caméra, la stocke sur le PC sous le nom de "Photo.jpg". Si bool est égal à True, la photo s'affichera. Cela permet de vérifier les paramètres. tps est le temps d'exposition en ms.

```
capture(im = "Photo.jpg", bool =  
False, tps = 50000)
```



```
detect(im = "Photo.jpg",  
nom_fichier = "ValeursC")
```

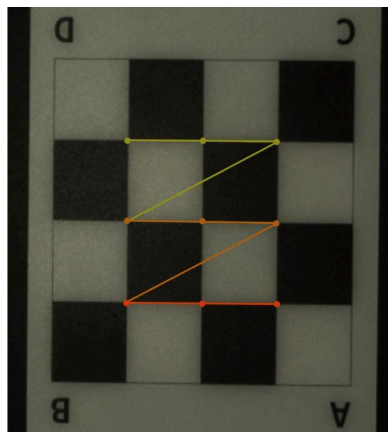
Prends en entrée deux chaînes de caractères :

- im : nom du fichier dont on veut détecter le dernier. Ne pas oublier ".jpg" à la fin de ce dernier.
- nom_fichier : nom du fichier JSON dans lequel on stocke les informations concernant les 9 points du damier par la caméra.

Ne renvoie aucun paramètre en sortie.

Cette fonction permet de détecter si un damier est présent sur la photo "Photo.jpg". Si celui-ci est détecté, la fonction reconnaîtra les 9 points centraux, ainsi que leurs coordonnées en pixels, et les stockera dans un fichier JSON sous le nom de "ValeursC".

Si aucun damier n'est détecté, la fonction renvoie "Aucun damier détecté ..."



```
[[1903, 1313], [1599, 1308], [1297, 1305],  
[1905, 1009], [1601, 1003], [1300, 1001],  
[1909, 704], [1605, 700], [1303, 698]]
```

`detecter_par_couleur(borne_basse, borne_haute, im)`

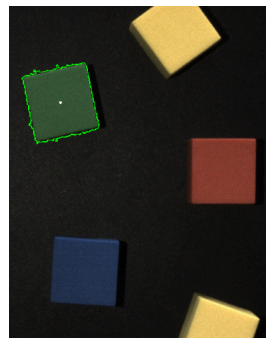
Prends en entrée deux listes et une chaîne de caractères :

- `borne_basse` : tableau à trois entiers, chacun représentant les valeurs de RGB pour la borne basse de la couleur souhaitée.
- `borne_haute` : tableau à trois entiers, chacun représentant les valeurs de RGB pour la borne haute de la couleur souhaitée.
- `im` : nom du fichier dont on veut détecter les couleurs. Ne pas oublier ".jpg" à la fin de ce dernier.

Renvoie une liste en sortie.

Permet de détecter les couleurs comprises en bornes_basse et borne_haute (en RGB) de l'image "Photo.jpg". `borne_basse` et `borne_haute` sont des listes de 3 éléments sous la forme de `[x, y, z]` (R, G, B) respectivement, allant de 0 jusqu'à 255.

De plus, la fonction renvoie une liste de double, contenant les coordonnées (x, y) du centre de chaque forme détectée.



```
>>> detecter_par_couleur([35, 0, 0], [85, 255, 255], "Image_cube.jpg")  
[(124, 175)]
```

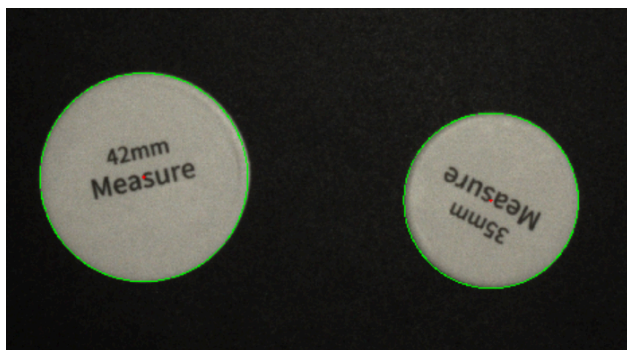
```
detecte_disque(nombre_disque,  
dp = 1, flou = 5, minD=200,  
p1=100, p2=60, minR=100,  
maxR=20000, img="PAD.jpg",  
bool = False)
```

Prends en entrée huit entiers, une chaîne de caractères et un booléen :

- nombre_disque : nombre de disques sur le plateau.
- dp : niveau de précision de la détection.
- flou : niveau de floutage pour effacer les bruits de l'image.
- minD : distance minimale en pixels entre deux disques.
- p1 : sensibilité pour détecter un contour net.
- p2 : niveau de sévérité de l'algorithme pour valider si c'est un vrai disque.
- minR : rayon minimum du disque en pixels.
- maxR : rayon maximum du disque en pixels.
- img : nom du fichier de l'image à traiter. Ne pas oublier ".jpg" à la fin de ce dernier.
- bool : choix d'afficher ou non la fenêtre de contrôle.

Renvoie en sortie une image et une liste.

Permet de détecter les disques présents sur l'image "Disque.jpg", prise en directe, avec nombre_disque le nombre de disques sur le plateau, dp = 1 est le niveau de précision, flou = 5 est le niveau de floutage de l'image pour simplifier la détection de disques, minD=200 la distance minimale entre deux disques, p1=100 le seuil haut de Canny (la sensibilité pour valider un contour net), p2=60 le seuil de vote, soit le niveau de sévérité de l'algorithme, et minR=100 et maxR=20000 les rayons min et max des disques. Renvoie ensuite l'image avec les disques détectés encadrés, ainsi que leurs coordonnées (x, y) et leurs rayons sous la forme d'une liste, en valeurs robot (mm).



```
>>> detecte_disque()  
[(199, 103, 28.757646560668945), (161, 162, 24.237937927246094)]
```

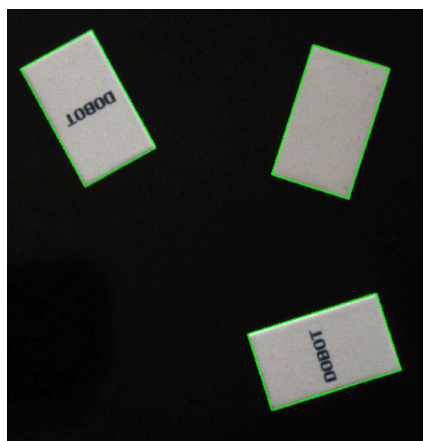
```
detecte_rect(nombre_rect, flou =  
5, binf = 100, bsup = 255,  
img="ARV.jpg", bool = False)
```

Prends en entrée quatre entiers, une chaîne de caractère et un booléen :

- nombre_rect : nombre de rectangles sur le plateau.
- flou : niveau de floutage pour effacer les bruits de l'image.
- binf : borne inférieure du seuil de luminosité pour isoler la couleur des rectangles.
- bsup : borne supérieure du seuil de luminosité pour transformer l'image en noir et blanc.
- img : nom du fichier de l'image à traiter. Ne pas oublier ".jpg" à la fin de ce dernier.
- bool : choix d'afficher ou non la fenêtre de contrôle.

Renvoie en sortie une image et une liste.

Permet de détecter les rectangles présents sur l'image "Rect.jpg", prise en directe, avec nombre_rect le nombre de rectangles sur le plateau, flou = 5 est le niveau de floutage de l'image pour simplifier la détection, binf = 100 et bsup = 255 les seuils inférieur et supérieur de luminosité pour isoler la couleur des rectangles par binarisation, img="ARV.jpg" le nom du fichier image traité, et bool = False le choix d'afficher ou non la fenêtre de contrôle. Renvoie ensuite l'image avec les rectangles détectés encadrés, ainsi que leurs coordonnées (x, y), leur hauteur et largeur et leur angle de rotation sous la forme d'une liste, en valeurs robot (mm).



```
>>> detecte_rect("ARV.jpg")  
[((714, 638), (211, 125), -18),  
 ((724, 279), (213, 124), -73),  
 ((346, 258), (125, 210), -29)]
```

detecte_qrcode(im="QR.jpg")

Prends en entrée une chaîne de caractères :
- im : nom du fichier de l'image à traiter. Ne pas oublier ".jpg" à la fin de ce dernier.

Renvoie en sortie une image et une liste.

Permet de détecter les qr codes sur l'image "QR.jpg", prise en directe. Renvoie ensuite la liste contenant les coordonnées des qr codes (en px et en mm), ainsi que leur contenu.



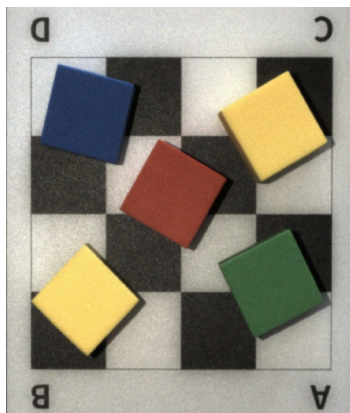
```
>>> detecte_qrcode("V.jpg")  
Contenu : Circle  
Position Camera : X=1511 mm, Y=650 mm
```

clic(im)

Prends en entrée une chaîne de caractères :
- im : nom du fichier de l'image à traiter. Ne pas oublier ".jpg" à la fin de ce dernier.

Renvoie en sortie des chaînes de caractères.

Affiche l'image im. L'utilisateur peut ensuite faire des clics gauches sur les pixels sur l'image pour obtenir leur valeur RGB. Ces valeurs sont affichées dans la console comme sur l'exemple ci-dessous :



```
>>> clic("AC2.jpg")  
Veuillez faire un clic gauche sur l'image pour enregistrer la couleur  
X : 1798, Y : 461, RGB: (248, 206, 106)  
X : 1396, Y : 667, RGB: (141, 65, 51)  
X : 1772, Y : 946, RGB: (59, 91, 54)
```

Fonctions Calculs	
homographie(bot, cam = "ValeursC", rob = "ValeursR")	Prends en entrée deux chaînes de caractères : - bot : nom du robot utilisé - cam : nom du fichier JSON avec les coordonnées des 9 points de calibration de la caméra. - rob : nom du fichier JSON avec les coordonnées des 9 points de calibration du robot. Renvoie en sortie une liste. Calcule la matrice d'homographie Permet de calculer la matrice d'homographie H à partir des valeurs des 9 points de la caméra "ValeursC" ainsi que les valeurs des 9 points du robot "ValeursR". Cette dernière est stockée dans un fichier JSON sous le nom de "H". <pre>>>> homographie() [[0.10619458437963704, 0.08693009352288841, -10.97880944141096], [0.07652357782108754, -0.06990123016519402, 120.74248042073823], [0.0001058548517471632, 0.0001245311674314257, 1.0]]</pre>
pixel_vers_robot(bot , x_pixel, y_pixel)	Prends en entrée deux entiers : - bot : nom du robot utilisé - x_pixel : valeur de la coordonnée en pixel x - y_pixel : valeur de la coordonnée en pixel y Renvoie en sortie un couple d'entiers. Permet de traduire des coordonnées caméra pixels (px) en coordonnées robot millimètres (mm). Le calcul de la matrice d'homographie est nécessaire pour que cette fonction renvoie au résultat attendu. <pre>>>> pixel_vers_robot(1340, 230) (104, 200)</pre>

rayon_to_mm(bot, centre_px, rayon_px)	Prends en entrée le nom du robot, une liste et un entier : - bot : nom du robot utilisé - centre_px : liste à deux éléments, contenant les entiers des coordonnées x et y du centre du cercle en pixels - rayon_px : longueur du rayon du cercle en pixels Renvoie en sortie un entier. Permet de convertir un rayon de disque de px à mm.  (Sur le disque est affiché le diamettre) <pre>>>> detecte_disque(1, img = "CTTV.jpg", bool = True) [(1664, 940, 229)] >>> rayon_to_mm(robot, [1664, 940], 229) 18.067411422729492</pre>
lire_json(nom)	Prends en entrée une chaîne de caractères : - nom : le nom du fichier JSON que vous souhaitez ouvrir dans la console. Renvoie en sortie le contenu d'un fichier. Ouvre le fichier JSON nom et affiche le contenu dans la console. Le fichier doit être sauvegardé dans le même dossier que le code python pour que ce dernier puisse le trouver. <pre>>>> lire_json("ValeursR") [[223, 127], [204, 113], [185, 99], [211, 149], [191, 135], [170, 120], [196, 168], [175, 155], [153, 139]]</pre>

lire_im(im, bool = False)

Prends en entrée une chaîne de caractères et un booléen :

- im : nom du fichier de l'image à ouvrir. Ne pas oublier ".jpg" à la fin de ce dernier.
- bool : permet d'afficher l'image ou non.

Renvoie en sortie une liste numpy.

Ouvre l'image JPG im et affiche le contenu sous forme de liste array. L'image doit être sauvegardée dans le même dossier que le code python pour que ce dernier puisse le trouver. Si bool = True, alors l'image est également affichée.

```
>>> lire_im("AD.jpg", False)
array([[ 9, 11, 12],
       [ 8, 10, 11],
       [ 8, 10, 11],
       ...,
       [ 6,  6,  6],
       [ 7,  7,  7],
       [10, 10, 10]])
```

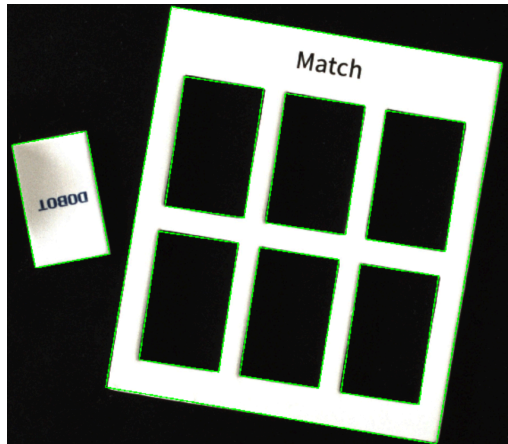
max_rect(tab)

Prends en entrée une liste :

- tab : la liste en sortie de la fonction detecte_rect.

Renvoie en sortie une liste.

Permet de déterminer parmi une liste de rectangles, lequel est le plus grand.



```
>>> detecte_rect(8, img = "rect.jpg", bool = True)
[((1445, 1105), (1220, 1409), -80), ((1607, 1511), (290, 475),
87, 474), -81), ((1668, 1147), (292, 476), -81), ((1108, 1052),
30, 779), (290, 477), -81), ((1167, 686), (293, 474), -82), ((11
-11)]

>>> max_rect([((1445, 1105), (1220, 1409), -80), ((1607, 1511),
47, 1417), (287, 474), -81), ((1668, 1147), (292, 476), -81), (
), -81), ((1730, 779), (290, 477), -81), ((1167, 686), (293, 474
(456, 274), -11)])
((1445, 1105), (1220, 1409), -80)
```